Introduce inline cache for class variable reads

@tenderlove and I would like to introduce an inline cache for class variable reads. We’ve attached a patch that introduces the cache. Class variable reads are popular in Rails applications for example, Active Record's #logger.

GitHub PR: [https://github.com/ruby/ruby/pull/4340](https://github.com/ruby/ruby/pull/4340)

**Cache Design**

This patch introduces a hash table that’s stored on the same class as the class variable value.

For example:

```ruby
class A
  @@foo = 1
end

class B < A
  def self.read_foo
    @@foo
  end
end
```

The above code stores the value for @@foo on the A class and stores an inline cache value on the A class as well. The instruction sequences for the read_foo method point at the CVAR inline cache entry stored on class A.

The lifecycle of these caches are similar to instance variable inline caches.

**Diagram of the cache:**

[![Diagram of the cache:](cvar%20cache@2x%20(6).png)](cvar%20cache@2x%20(6).png)

**Performance Characteristics**

When class variables are read, Ruby needs to check each class in the inheritance tree to ensure that the class variable isn't set on any other classes in the tree. If the same cvar is set on a class in the inheritance tree then a "cvar overtaken" error will be raised.

Because of how cvar reads work, the more classes in the inheritance tree the more expensive a cvar read is. To demonstrate this here is a benchmark that reads a cvar from a class with 1 module, 30 modules, and 100 modules in the inheritance chain. On Ruby master 100 modules is 8.5x slower than including 1 module. With the cache, there is no performance difference between including 1 module and including 100 modules.

Benchmark script:

```ruby
require "benchmark/ips"

MODULES = [
```

03/24/2022
class A
@@foo = 1

def self.foo
@@foo
end

eval <<-EOM
module #{MODULES.first}
end

include #{MODULES.first}
EOM
end

class Athirty
@@foo = 1

def self.foo
@@foo
end

MODULES.take(30).each do |module_name|
  eval <<-EOM
    module #{module_name}
  end

  include #{module_name}
  EOM
end
end

class Ahundred
@@foo = 1

def self.foo
@@foo
end

MODULES.each do |module_name|
  eval <<-EOM
    module #{module_name}
  end

  include #{module_name}
  EOM
end
end

Benchmark.ips do |x|
  x.report "1 module" do
    A.foo
  end

  x.report "30 modules" do
    Athirty.foo
  end

  x.report "100 modules" do
    Ahundred.foo
  end

  x.compare!
end

Ruby 3.0 master:
Warming up  --------------------------------------

<table>
<thead>
<tr>
<th>Modules</th>
<th>i/100ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.231M</td>
</tr>
<tr>
<td>30</td>
<td>432.020k</td>
</tr>
<tr>
<td>100</td>
<td>145.399k</td>
</tr>
</tbody>
</table>

Calculating  -------------------------------------

<table>
<thead>
<tr>
<th>Modules</th>
<th>i/s</th>
<th>i/s -</th>
<th>in</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12.210M (+ 2.1%)</td>
<td>61.553M</td>
<td>5.043400s</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>4.354M (+ 2.7%)</td>
<td>22.033M</td>
<td>5.063839s</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>1.434M (+ 2.9%)</td>
<td>7.270M</td>
<td>5.072531s</td>
<td></td>
</tr>
</tbody>
</table>

Comparison:

<table>
<thead>
<tr>
<th>Modules</th>
<th>i/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12209958.3</td>
</tr>
<tr>
<td>30</td>
<td>4354217.8</td>
</tr>
<tr>
<td>100</td>
<td>1434447.3</td>
</tr>
</tbody>
</table>

Ruby 3.0 with cvar cache:

Warming up  --------------------------------------

<table>
<thead>
<tr>
<th>Modules</th>
<th>i/100ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.641M</td>
</tr>
<tr>
<td>30</td>
<td>1.655M</td>
</tr>
<tr>
<td>100</td>
<td>1.620M</td>
</tr>
</tbody>
</table>

Calculating  -------------------------------------

<table>
<thead>
<tr>
<th>Modules</th>
<th>i/s</th>
<th>i/s -</th>
<th>in</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16.279M (+ 3.8%)</td>
<td>82.038M</td>
<td>5.046923s</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>15.891M (+ 3.9%)</td>
<td>79.459M</td>
<td>5.007958s</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>16.087M (+ 3.6%)</td>
<td>81.005M</td>
<td>5.041931s</td>
<td></td>
</tr>
</tbody>
</table>

Comparison:

<table>
<thead>
<tr>
<th>Modules</th>
<th>i/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16279458.0</td>
</tr>
<tr>
<td>30</td>
<td>15891406.2</td>
</tr>
</tbody>
</table>

Rails Application Benchmarks

We also benchmarked ActiveRecord::Base.logger since logger is a cvar and there are 63 modules in the inheritance chain. This is an example of a real-world improvement to Rails applications.

Benchmark:

```ruby
require "benchmark/ips"
require_relative "config/environment"

Benchmark.ips do |x|
  x.report "logger" do
    ActiveRecord::Base.logger
  end
end
```

Ruby 3.0 master:

Warming up  --------------------------------------

<table>
<thead>
<tr>
<th>Modules</th>
<th>i/100ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>logger</td>
<td>155.251k</td>
</tr>
</tbody>
</table>

Calculating  -------------------------------------

<table>
<thead>
<tr>
<th>Modules</th>
<th>i/s</th>
<th>i/s -</th>
<th>in</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>logger</td>
<td>14.857M (+ 4.8%)</td>
<td>74.198M</td>
<td>5.006202s</td>
<td></td>
</tr>
</tbody>
</table>

We also measured database queries in Rails and with the cvar cache they are about ~9% faster.

Benchmark code:

```ruby
class BugTest < Minitest::Test
  def test_association_stuff
```
post = Post.create!

Benchmark.ips do |x|
  x.report "query" do
    Post.first
  end
end
end
end

Ruby 3.0 master / Rails 6.1:

Warming up  --------------------------------------
query   790.000  i/100ms
Calculating -------------------------------------
query      7.601k (± 3.8%) i/s - 38.710k in 5.100534s

Ruby 3.0 cvar cache / Rails 6.1:

Warming up  --------------------------------------
query   731.000  i/100ms
Calculating -------------------------------------
query      7.089k (± 3.3%) i/s - 35.819k in 5.058215s

History

#1 - 03/30/2021 06:24 PM - eileencodes (Eileen Uchitelle)
This is the missing benchmark I copy and pasted incorrectly.

Ruby master / Rails 6.1:

```
Warming up --------------------------------------
query   900.000  i/100ms
Calculating -------------------------------------
query 1.502M (± 4.5%) I/s - 7.607M in 5.076869s
```

This branch / Rails 6.1:

```
Warming up --------------------------------------
query   1.546M  i/100ms
Calculating -------------------------------------
query 14.857M (± 4.8%) I/s - 74.198M in 5.006202s
```

#2 - 03/30/2021 07:05 PM - Eregon (Benoit Daloze)

Nice work.
I guess using a global serial here is the only way to handle overtaking without redoing the lookup every time like before.
I wonder, could the serial be global (not per module) but per name? (not asking to change anything, just curious)

Personally I wish we would just deprecate class variables, because I believe nobody wants the strange semantics, and they are inherently less efficient than instance variables on modules.
Or potentially make them equivalent in behavior to instance variables on modules (#14541).
But that is probably never going to happen.

---

#3 - 03/30/2021 07:26 PM - tenderlovemaking (Aaron Patterson)
Eregon (Benoit Daloze) wrote in #note-2:

Nice work.

I guess using a global serial here is the only way to handle overtaking without redoing the lookup every time like before.
I wonder, could the serial be global (not per module) but per name? (not asking to change anything, just curious)

I think we probably could do that. Keeping a global counter just seemed like the easiest solution at the moment. Also, cvars seem very unpopular (compared with ivars) so I'm not sure adding complexity would be worth while.

---

#4 - 03/30/2021 07:45 PM - Eregon (Benoit Daloze)
Eregon (Benoit Daloze) wrote in #note-2:

they are inherently less efficient than instance variables on modules.

chrisseaton (Chris Seaton) told me that it's not necessarily the case, and I think they can be exactly the same for a regular read (Shape check + load field in TruffleRuby).
I was thinking before this optimization, where CRuby would do an ancestor lookup every time for the overtaking logic IIRC.

Also, cvars seem very unpopular (compared with ivars) so I'm not sure adding complexity would be worth while.

Yeah, definitely. Which is why I've been hesitant about optimizing class variables, it's unclear if the cost of the not-so-trivial optimization would pay off. It clearly seems worth it if code like the Rails logger keep using class variables though.

---

#5 - 03/30/2021 11:27 PM - duerst (Martin Dürst)
Eregon (Benoit Daloze) wrote in #note-4:

Eregon (Benoit Daloze) wrote in #note-2:

Also, cvars seem very unpopular (compared with ivars) so I'm not sure adding complexity would be worth while.

Yeah, definitely. Which is why I've been hesitant about optimizing class variables, it's unclear if the cost of the not-so-trivial optimization would pay off. It clearly seems worth it if code like the Rails logger keep using class variables though.

I'm not sure that "better optimize, because some important code keeps using this, but don't really optimize all the way, because it's no so popular" makes sense. (I'm not blaming Eregon, nor Aaron, nor Eileen, nor anybody else.)

What would it take e.g. to switch Rails logger to something else?

If the reason that some places are keeping class variables, then maybe we need to up with more convenient syntax for class instance variables.

Just trying to think out loud, sorry.

---

#6 - 03/30/2021 11:57 PM - tenderlovemaking (Aaron Patterson)
duerst (Martin Dürst) wrote in #note-5:

What would it take e.g. to switch Rails logger to something else?

If the reason that some places are keeping class variables, then maybe we need to up with more convenient syntax for class instance variables.

Class variables have different semantics than class instance variables. It's possible to switch some things to use class instance variables without breaking existing behavior, but in other places we would have to basically re-implement class variable behavior (and at that point, you may as well use class variables).

I think the logger is one of the places where we depend on class variable semantics and it's unlikely to change.
Also in this case, it makes me feel weird to change the implementation of Rails when we can just make Ruby perform better. Changing Rails to suit Ruby seems like the tail wagging the dog (obviously not all cases are clear cut though)

#7 - 03/31/2021 10:15 AM - Eregon (Benoit Daloze)
tenderlovemaking (Aaron Patterson) wrote in #note-6:

Also in this case, it makes me feel weird to change the implementation of Rails when we can just make Ruby perform better. Changing Rails to suit Ruby seems like the tail wagging the dog (obviously not all cases are clear cut though)

I have a different view of this, before this change, class variables were always extremely slow compared to class instance variables (10x in the case of ActiveRecord::Base.logger).
So changing it in Rails would fix it for all Ruby versions.

The logger seems handled by mattr_accessor:
https://github.com/rails/rails/blob/d612542336d9a61381311c95a27d801bb4094779/activerecord/lib/active_record/core.rb#L20

And mattr_accessor is what uses class variables:
https://github.com/rails/rails/blob/d612542336d9a61381311c95a27d801bb4094779/activerecord/lib/active_record/core.rb
But it seems like it could use an instance variable on the class instead, and use attr_accessor + for instance methods def foo; self.class.foo and the writer.

but in other places we would have to basically re-implement class variable behavior (and at that point, you may as well use class variables).

That might be worth it for performance, i.e., using class methods + class instance variables instead of class variables.
Also https://github.com/rails/rails/blob/d612542336d9a61381311c95a27d801bb4094779/activerecord/lib/active_record/core.rb#L85 seems to have some inheritance and yet already does not use class variables.

My personal point of view is class variables are de-facto deprecated syntax due to their strange and often unexpected semantics, and which about every Ruby tutorial/book recommends against using.
The least usages of class variables we have the better we are IMHO.
I once went through removing many class variables in the CRuby repo and it was fairly easy for most cases IIRC, in part because most of these cases did not want any inheritance.

To clarify, my point is this is good work and now that it's done and I think we should merge it, but it seems worthwhile to look in Rails and other places if class variables are really needed.
If they can be replaced with class ivars, we could speed those accesses on all Ruby versions, not just 3.1+

#8 - 05/21/2021 11:35 AM - Eregon (Benoit Daloze)
FWIW, ActiveRecord::Base.logger no longer uses class variables since
https://github.com/rails/rails/commit/dcc2530af74c86f355a9206bb1d0b084734fae3e (nice!)

Maybe we should really just deprecate class variables since their semantics are confusing (see below), their performance is not great / they are complicated to optimize, and they seem to be recommended to not be used since a long time in Ruby (de-facto deprecated).
That could start with a $VERBOSE=true warning, then $VERBOSE=false warning and then removal.

Example of weird semantics, which I would think no one would ever want:

class C; @@a = 1; end
class D < C; end
class D; @@a = 2; end
class C; p @@a; end # => 2

#9 - 05/21/2021 12:58 PM - byroot (Jean Boussier)

FWIW, ActiveRecord::Base.logger no longer uses class variables since
https://github.com/rails/rails/commit/dcc2530af74c86f355a9206bb1d0b084734fae3e

Yes, we've talked about changing it since, it's not really related to this ticket. IMHO the two are orthogonal, AR::Base.logger just happened to be a good example of the potential performance impact.

Also note that it was replaced by class_attribute, which is a "Railsism" and has a non-trivial implementation. Non-rails projects with a similar problems don't really have a solution for this.

Maybe we should really just deprecate class variables since their semantics are confusing
The problem is that in some cases there's not really any alternative, so I'm all for deprecating them, but IMHO a replacement with better semantic is needed.

Or maybe their behavior could be changed with some kind of switch:

```ruby
class MyClass
  new_class_variable_semantic
  @@foo = 1
end
```

Which would open the door to changing their semantic over the course of a few releases.

#10 - 06/02/2021 02:13 AM - ko1 (Koichi Sasada)

We also benchmarked ActiveRecord::Base.logger since logger is a cvar and there are 63 modules in the inheritance chain. This is an example of a real-world improvement to Rails applications.

Calculating -------------------------------------

It seems you missed to write the result.

We also measured database queries in Rails and with the cvar cache they are about ~9% faster.

It seems w/o cache is faster. Could you check it? (opposite results?)

Do you have application benchmark (request per second, for example)?

I understand the performance benefit, but it can introduce complexity so I want to confirm the value of this proposal.

Thanks,
Koichi

#11 - 06/02/2021 09:10 AM - byroot (Jean Boussier)

It seems you missed to write the result.

It's in the first comment: [https://bugs.ruby-lang.org/issues/17763#note-1](https://bugs.ruby-lang.org/issues/17763#note-1)

Ruby master / Rails 6.1:

```
Warming up _________________________________________________________________
Calculating _____________________________________________________________
logger 155.251k I/100ms
```

This branch / Rails 6.1:

```
Warming up _________________________________________________________________
Calculating _____________________________________________________________
logger 1546M I/100ms
```

#12 - 06/04/2021 05:46 PM - eileencodes (Eileen Uchitelle)

I ran benchmarks using railsbench and the branch with the CVAR cache is a lot faster, 657 requests per second over the 615 requests per second on master.

Master:

```
$ RAILS_ENV=production INTERVAL=100 WARMUP=1 BENCHMARK=10000 ruby bin/bench
ruby 3.1.0dev (2021-06-04T00:24:57Z master 91c542ad05) [x86_64-darwin19]
Warming up...
Warmup: 1 requests
Benchmark: 10000 requests
Request per second: 615.1 [#/s] (mean)
```

03/24/2022
Percentage of the requests served within a certain time (ms)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>1.57</td>
</tr>
<tr>
<td>66%</td>
<td>1.68</td>
</tr>
<tr>
<td>75%</td>
<td>1.74</td>
</tr>
<tr>
<td>80%</td>
<td>1.78</td>
</tr>
<tr>
<td>90%</td>
<td>1.91</td>
</tr>
<tr>
<td>95%</td>
<td>2.06</td>
</tr>
<tr>
<td>98%</td>
<td>2.36</td>
</tr>
<tr>
<td>99%</td>
<td>2.67</td>
</tr>
<tr>
<td>100%</td>
<td>35.15</td>
</tr>
</tbody>
</table>

CVAR Branch

```bash
$ RAILS_ENV=production INTERVAL=100 WARMUP=1 BENCHMARK=10000 ruby bin/bench
ruby 3.1.0dev (2021-06-04T17:40:20Z add-cache-for-clas.. 37c96af98b) [x86_64-darwin19]
Warming up...
Warmup: 1 requests
Benchmark: 10000 requests
Request per second: 657.1 [#/s] (mean)
```

Percentage of the requests served within a certain time (ms)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>1.46</td>
</tr>
<tr>
<td>66%</td>
<td>1.56</td>
</tr>
<tr>
<td>75%</td>
<td>1.63</td>
</tr>
<tr>
<td>80%</td>
<td>1.68</td>
</tr>
<tr>
<td>90%</td>
<td>1.82</td>
</tr>
<tr>
<td>95%</td>
<td>2.01</td>
</tr>
<tr>
<td>98%</td>
<td>2.28</td>
</tr>
<tr>
<td>99%</td>
<td>2.50</td>
</tr>
<tr>
<td>100%</td>
<td>35.13</td>
</tr>
</tbody>
</table>

#13 - 06/04/2021 05:47 PM - eileencodes (Eileen Uchitelle)
This is the script/app I used to measure https://github.com/k0kubun/railsbench

#14 - 06/09/2021 12:31 AM - ko1 (Koichi Sasada)
Thank you for the benchmark.

How about that?

We also measured database queries in Rails and with the cvar cache they are about ~9% faster.

It seems w/o cache is faster. Could you check it? (opposite results?)

#15 - 06/11/2021 05:27 PM - eileencodes (Eileen Uchitelle)
I re-ran the benchmarks. I accidentally switched them when I pasted them into here.

Before: Rails 6.1 / Ruby master

```bash
ruby 3.1.0dev (2021-06-04T00:24:57Z master 91c542ad05) [x86_64-darwin19]
-- create_table(:posts, {:force=>true})
  -> 0.0067s
-- create_table(:comments, {:force=>true})
  -> 0.0003s
Run options: --seed 33069
```

# Running:

```bash
# Running:

Warming up --------------------------------------
query 708.000 i/100ms
Calculating -------------------------------------
query 7.177k (± 5.6%) i/s - 36.108k in 5.047561s
```

After: Rails 6.1 / CVAR cache branch

```bash
ruby 3.1.0dev (2021-06-04T17:40:20Z add-cache-for-clas.. 37c96af98b) [x86_64-darwin19]
last_commit=Add a cache for class variables
```

03/24/2022
-- create_table(:posts, {:force=>true})
  -> 0.0067s
-- create_table(:comments, {:force=>true})
  -> 0.0003s
Run options: --seed 45750

# Running:

Warming up -----------------------------
  query 837.000 i/100ms
Calculating ----------------------------
  query 8.343k (+1.3%) i/s - 41.850k in 5.017218s

#16 - 06/17/2021 07:19 AM - matz (Yukihiro Matsumoto)
It seems great. The code complexity is my concern. But I agree with trying it.
Matz.

#17 - 12/24/2021 02:58 PM - ko1 (Koichi Sasada)
- Status changed from Open to Closed

merged: b91b3bc7717a97f4f1cdef6131b1688e1958ddefed