Ruby trunk - Bug #14541

Class variables have broken semantics, let's fix them

02/22/2018 11:47 AM - Eregon (Benoit Daloze)

Status:	Open		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 2.6.0dev (2018-01-29 trunk 62091) [x86_64-linux]	**Backport:**	2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN

Description

Class variables have the weird semantics of being tied to the class hierarchy and being inherited between classes.
I think this is counter-intuitive, dangerous and basically nobody expects this behavior.

To illustrate that, we can break the tmpdir stdlib by defining a top-level class variable:

```
$ ruby -rtmpdir -e '$SAFE=1; @@systmpdir=42; p Dir.mktmpdir {}'
-e:1: warning: class variable access from toplevel
Traceback (most recent call last):
    3: from -e:1:in `<main>'
    2: from /home/eregon/prefix/ruby-trunk/lib/ruby/2.6.0/tmpdir.rb:86:in `mktmpdir'
    1: from /home/eregon/prefix/ruby-trunk/lib/ruby/2.6.0/tmpdir.rb:125:in `create'
/home/eregon/prefix/ruby-trunk/lib/ruby/2.6.0/tmpdir.rb:125:in `join': no implicit conversion of I
nteger into String (TypeError)
```

Or even simpler in RubyGems:

```
$ ruby -e '@@all=42; p Gem.ruby_version'
-e:1: warning: class variable access from toplevel
Traceback (most recent call last):
    3: from -e:1:in `<main>'
    2: from /home/eregon/prefix/ruby-trunk/lib/ruby/2.6.0/rubygems.rb:984:in `ruby_version'
    1: from /home/eregon/prefix/ruby-trunk/lib/ruby/2.6.0/rubygems/version.rb:199:in `new'
/home/eregon/prefix/ruby-trunk/lib/ruby/2.6.0/rubygems/version.rb:199:in `[]': no implicit convers
ion of String into Integer (TypeError)
```

So defining a class variable on Object removes class variables in all classes inheriting from Object.
Maybe @@systmpdir is not so prone to conflict, but how about @@identifier, @@context, @@locales, @@sequence, @@all, etc which are class variables of the standard library?

Moreover, class variables are extremely complex to implement correctly and very difficult to optimize due to the complex semantics.
In fact, none of JRuby, TruffleRuby, Rubinius and MRuby implement the "setting a class var on Object removes class vars in subclasses".
It seems all implementations but MRI print :foo twice here (instead of :foo :toplevel for MRI):

```
class Foo
  @@cvar = :foo
  def self.read
    @@cvar
  end
end

p Foo.read
@@cvar = :toplevel
p Foo.read
```

Is there any library actually taking advantage that class variables are inherited between classes? I would guess not or very few.
Therefore, I propose to give class variable intuitive semantics: no inheritance, they behave just like variables of that specific class, much like class-level instance variables (but separate for compatibility).

Another option is to remove them completely, but that's likely too hard for compatibility.

Thoughts?

History

#1 - 02/22/2018 11:49 AM - Eregon (Benoit Daloze)

To clarify, "setting a class var on Object removes class vars in subclasses" means:

Setting a class variable in some class removes class variables *of the same name* in all descendant classes of that class.

#2 - 02/22/2018 01:19 PM - shevegen (Robert A. Heiler)

I'd rather just remove them altogether. :P

However had, since I myself do not use them anyway, it is not really important to me whether they are there or not - I only use a subset of ruby which I like. I mostly store stuff that should be available in a project in the main "namespace", such as module Foobar; end and using accessors on "Foobar"; and "module-level instance variables" as I call them, like:

```
module Foobar

  @use_colours = true

  def self.use_colours?
    @use_colours
  end

  def self.disable_colours
    @use_colours = false
  end

end
```

I also do not really need another way how to use @@class variables, e. g. to act as a counter. I don't think I ever needed to have a counter available to keep track of how many ruby instances are created/instantiated of a given class.

> Moreover, class variables are extremely complex to implement correctly

Another reason to remove them. ;)

> Is there any library actually taking advantage that class variables are inherited between classes? I would guess not or very few.

I think that class variables are not really used that much. I have not done any analysis but my intuition tells me that most of the highly used and popular gems available on rubygems.org do not use them. Again, that is a statement not based on any statistics at all, but I agree with your comment - I say none of the gems in the top 20 make use of @@variables. I'd even say that IF there were any use, you could easily write code to NOT require them, too.

> Another option is to remove them completely, but that's likely too hard for compatibility.

Oops, I just read your comment here. So you suggest removing too.

I agree with it as well. :)

Compatibility is important but ruby 3.x may make changes to code. I think matz said so in one of the presentations, so removing class variables would be a possibility for 3.x.

I do not know whether matz wants to remove them or not. But perhaps it could be mentioned in the next ruby developer meeting.

I think there may be quite many ruby hackers who would not have any problem if @@class variables would be removed. Since I do

not have any of them in my code, removing class variables would not impact me at all whatsoever. :)

I think perhaps it would be good to hear:

a) other ruby hackers who used ruby a lot, what they have to say about @@class variables

and, perhaps even more importantly,

b) anyone who may still, for one reason or the other, use @@class variables and how easy it may be for them to transition (I guess it would be very simple because you can do everything without class variables too, I think, in pure ruby, right? You only need to be able to catch the event whenever a ruby object is instantiated and I think that can be done via one of the hooks).

#3 - 02/22/2018 01:42 PM - Eregon (Benoit Daloze)

shevegen (Robert A. Heiler) wrote:

> I'd rather just remove them altogether. :P

That might be possible in Ruby 3, but unlikely in Ruby 2.x.
Even then, I don't think we want to break compatibility too much for Ruby 3.
I would rather see this fixed before Ruby 3.

There are currently 228 instances of "@@" in the standard library alone, so it seems that breaking those and many gems would be unbearable.

Moreover, manually defining class-level instance variables with

```
class MyClass
  @classvar = :initial_value
  class << self
    attr_accessor :@classvar
  end

  def some_use_of_classvar
    MyClass.classvar ||= ...
  end
end
```

is quite cumbersome, verbose and error-prone (to define the accessors on the singleton class).

So I think having the current class variables (@@) but with simple semantics would be convenient.
And I believe that would incidentally achieve what most of class variables usages in the wild expect (no inheritance, just state on the specific class).

#4 - 02/22/2018 02:16 PM - dsferreira (Daniel Ferreira)

Eregon (Benoit Daloze) wrote:

> Moreover, manually defining class-level instance variables with
>
> ```
> class MyClass
>   @classvar = :initial_value
>   class << self
>     attr_accessor :@classvar
>   end
>
>   def some_use_of_classvar
>     MyClass.classvar ||= ...
>   end
> end
> ```
>
> is quite cumbersome, verbose and error-prone (to define the accessors on the singleton class).

This is one of those change requests that I have thought about for a long time already but didn't request it because I thought it would not be accepted.
I totally support it.

This kind of code example is kind of what I end up with (add private methods to it) in order to avoid all the quirks of current class variables.
Making class variables predictable by not giving them inheritance would be good but then how would we share state between the class hierarchy?
Using a top level class accessor? That works for me. Any situation where it would not apply?

Hope this can be accepted.

#5 - 03/09/2018 02:42 AM - shevegen (Robert A. Heiler)

> This is one of those change requests that I have thought about for a long time already but didn't request it because I thought it would not be accepted.

I may be wrong but I think that matz has indicated a possibility to change the behaviour (for ruby 3.x, I assume, probably not in the 2.x branch). Perhaps it may not be accepted for 2.x, but I assume that there may be a possibility for 3.x.

If I recall correctly - and this is mostly based on matz giving presentations over the last ~4 years or so, also in particular the one about "good change, bad change" - matz said that there was pain/problems/complaints in regards to the change between ruby 1.8.x and 1.9.x leading up towards 2.x and he wants to avoid that when possible.

And this may be one of the primary reasons why some changes may not happen too quickly, if only to not avoid breaking a lot of code.

Though matz has also said that ruby 3.x can make backwards incompatible changes, so there may be a possibility to see @@variables revisited. It's not on my personal list of very important things, since I do not use them - but I otherwise agree with Benoit. :) I think the main thing is that @@variables do not really provide a huge, compelling advantage over other means to write ruby code. At the least I have not found a situation where I would miss @@variables - in the past I misused CONSTANTS to store data, until I found out that I could use @foo variables too and use setters/getters on the module/class level. :D

I think what may be useful in regards to any possibility to revisit @@variables is if other (more) ruby hackers could add to the tracker here whether they use class variables; and what their usage pattern is for them. So far it seems as if those who commented here, do not really use/need them.

It may be that almost nobody uses them these days, I have no idea :) - but in the event that some day @@variables may be changed or removed, perhaps those who may be affected by it could comment; and a transition path could be suggested for those who still use them. Transition meaning such as a warning for deprecation and perhaps a documentation explaining how to avoid having to use them or suggest alternatives, in a doc at https://www.ruby-lang.org/en/. A bit similar to the doc that explains the "Symbol versus String" situation that has been added recently.

On the other hand, if not many ruby hackers complain about @@vars, then perhaps this is an indication that class variables are not really that much in use in the first place. In these cases, changing or removing them may not affect many people. For example, to me the two biggest problems from ruby 1.8.x and ruby versions past that were the encoding situation and the yaml->psych transition. These days I am not affected that much by either, but back then it was a lot harder to transition. (I am also trying to stay up to date when it comes to ruby versions, so the xmas releases systematically replace my current ruby.)

The upcoming ruby developer meeting at:

https://bugs.ruby-lang.org/projects/ruby/wiki/DevelopersMeeting20180315Japan

will discuss it soon, so anyone who wants to chime in before the meeting, please do so, no matter if you like or dislike, use or don't use class variables.

#6 - 03/09/2018 07:18 AM - dsferreira (Daniel Ferreira)

shevegen (Robert A. Heiler) wrote:

> A bit similar to the doc that explains the "Symbol versus String" situation that has been added recently.

Can you please paste here the link to the “Symbol versus String” doc?

#7 - 03/09/2018 10:00 AM - Eregon (Benoit Daloze)

If changing class variables to no longer be inherited between classes is considered too hard for compatibility (but I'd like a real-world example),
how about at least removing the semantics that defining a class variable in a superclass removes class variables of the same name in all subclasses?

As said above, it seems no implementation but MRI implements that currently, showing how little Ruby code relies on that.
I argue it's also a very confusing behavior for the programmer, as illustrated by the examples in this bug description.
Defining a class variable at the top-level basically breaks encapsulation for all class variables of the same name, which sounds like something nobody wants.

I do believe much simpler semantics for class variables without inheriting between classes is what most Ruby developers want,
and I would be interested to see if there is any Ruby code using class variable inheritance on purpose.

#8 - 03/15/2018 07:15 AM - matz (Yukihiro Matsumoto)

Although the use of class variables is not recommended (like global variables), proposed behavior changes introduce huge incompatibility.

Error-prone cases like above examples, we already give warnings.

- "warning: class variable access from toplevel"
- "warning: class variable @foo of D is overtaken by C"

So this so-called bug is a consequence of ignoring the warnings. Don't.

It's possible to make those warning replaced by exceptions. I am strongly positive about it.

Matz.

#9 - 03/15/2018 12:24 PM - Eregon (Benoit Daloze)

matz (Yukihiro Matsumoto) wrote:

> It's possible to make those warning replaced by exceptions. I am strongly positive about it.

OK, at least that would remove the very confusing semantics of a class variable in a superclass removes class variables with the same name in subclasses.
What exception class should be used? A RuntimeError?

FWIW, I tried removing the class hierarchy lookup for class variables.
Here is an incomplete patch, but it passes ruby/spec and almost test-all (a couple tests hang, I didn't have time to investigate those):
https://github.com/ruby/ruby/compare/trunk...eregon:classvar_not_inherited

CGI and Gem::TestCase seem to use class variables pretty much like global variables, so I replaced those with class-level instance variables.
I think it would be worth to merge those changes, regardless of what happens to class variables, since it makes the logic much clearer.
I hope I'll have time for it.

#10 - 03/15/2018 01:17 PM - Hanmac (Hans Mackowiak)

Eregon (Benoit Daloze): the problem is the other way around ...

a class variable in a superclass DOES NOT removes class variables in subclasses.

BUT if you define the class variable in the superclass BEFORE the one is defined in the subclasses,

THEN the subclass will use the parent one instead

#11 - 03/15/2018 03:00 PM - Eregon (Benoit Daloze)

Hanmac (Hans Mackowiak): Both problems exist, see the example in the description.
The semantics are confusing, that's why I wish we could simplify to avoid inheritance between class variables.
But, matz says it is too incompatible.

#12 - 10/08/2018 03:16 PM - jwmittag (Jörg W Mittag)

Hanmac (Hans Mackowiak) wrote:

> Eregon (Benoit Daloze): the problem is the other way around ...
>
> a class variable in a superclass DOES NOT removes class variables in subclasses.

> BUT if you define the class variable in the superclass BEFORE the one is defined in the subclasses,
>
> THEN the subclass will use the parent one instead

In addition to what Benoit said, there is another problem, namely that class definitions in Ruby are never "finished", so when talking about making changes to classes, "before" and "after" doesn't even make sense. You could always add a variable to either a subclass or a subclass at any time!

#13 - 12/11/2018 06:24 PM - lamont (Lamont Granquist)

Inheritance of @@class variables is precisely what makes them useful, since they're truly one single global value.

And in addition to what matz points out rubocop has the Cop::Style::ClassVars warning.

As someone who manages a 10 year old, 250,000 line long ruby codebase, please don't remove them or change the semantics. The class variables we have now in our project have often been around for 10 years and while in some purist sense they are all technical debt, they're not doing any harm -- forcing me to have a flag day to fix them all would be a large waste of my time (some of them have been fixed as they've been found to be problematic and/or the code in question simply got cleaned up as part of a larger refactoring pass).

#14 - 01/03/2019 03:26 PM - shevegen (Robert A. Heiler)

> Inheritance of @@class variables is precisely what makes them useful,
> since they're truly one single global value.

You can always find pros/cons. Some will find a feature useful, others will not. My personal opinion, for example, is that I find @@vars largely unnecessary, so I don't use them in my own code; another smaller reason is that I find the two @ not so elegant. Others may have another opinion and already expressed so, e. g. at a developer meeting some months ago - I think that will very often be the case where people have different opinions. And ultimately matz decides and he already decided and explained here, and elsewhere. :)

I can achieve "inheritance" via @instance variables and specifying access to it via custom code too (well, methods), but I think it is not a very strong argument to refer to it as what makes @@class variables that useful to begin with, because ruby is so dynamic that inheritance and access-specifiers aren't a very strict concept. For example, we can obtain and change variables at "runtime" at any moment in ruby as-is, e. g. instance_variable_get/set and so forth. Ruby is ultimately a "tool-box" of code and it has another concept for both OOP but also how to interface with it (from the human side), compared to, say, C++ and Java.

> And in addition to what matz points out rubocop has the
> Cop::Style::ClassVars warning.

Rubocop, and neither the style guide, are not designing ruby though. It's great that rubocop exists; and it is great that it can be of help keeping code bases sane. But that is only one part - the other part is the design of ruby as such in itself. The ruby parser can also be thought of some kind of "style guide", e. g. what it enforces, or what warnings it will show, and so forth. It would be nice if we could customize it a bit more in general when it comes to warnings/notifications, a bit like rubocop - but keeping this simple, too. Rubocop can become a bit complicated if you look at all the different cop-rules that projects can use. I like simplicity too.

> As someone who manages a 10 year old, 250,000 line long ruby
> codebase, please don't remove them or change the semantics.

I have been using ruby for a very, very, very long time but ... how can you manage to write 250.000 lines of ruby code? I assume that must have been written by more than one person.

> The class variables we have now in our project have often been
> around for 10 years and while in some purist sense they are all
> technical debt, they're not doing any harm -- forcing me to have
> a flag day to fix them all would be a large waste of my time
> (some of them have been fixed as they've been found to be
> problematic and/or the code in question simply got cleaned
> up as part of a larger refactoring pass).

I think you need not worry - matz already said that the incompatibility issue is a real one, so it is super-unlikely that class variables will be changed; most definitely not for ruby 3.0 but I think probably also not at a later time. So a lot of this discussion here is mostly a purely hypothetical one. There have been other ruby users who expressed that they use @@class vars and that they also like them. There is no "wrong" use of code as such per se; people will use features and functionality when it is made available.

I think very large code bases are always problematic, not just in regards to ruby alone but in general.

Matz also mentioned several times in presentations that he wants to avoid changes such as from ruby 1.8.x to (ultimately) ruby 2.0 as that was a pain point for quite a few people. So I think when discussing it, we should mostly refer to this as a purely theoretical discussion.

I would also like to propose to eventually close this particular issue here eventually when eregon is ok with it - ideally before ruby 3.0 is released, simply to keep the amount of issues a bit smaller. (We could always have another discussion in the future after 3.0 but I think the chances for change here are quite low, and about 0% for ruby 3.0 anyway; possibly even 0% at a later time but who knows.)

By the way I also agree that it is not something that is hugely important from a practical point of view - people who like class variables can use them; those who don't like them can avoid them. Ultimately project owners can specify what they need/accept in code bases that they maintain. I have a long list of ruby code I would reject - others may probably feel similar in a different way about ruby code they find acceptable and code they don't. :) (I actually found that one of the biggest problem is the lack of documentation and comments - some people never write any comments and barely any documentation, and it is very often that code written by them is either brilliant or totally awful. And in both cases comments/documentation would help OTHER people immensely, if it is up-to-date and of high quality.)