Ruby master - Feature #15330

autoload_relative

11/21/2018 11:43 PM - marcandre (Marc-Andre Lafortune)

Status:	Open	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:		

Description

I'd like to propose a way to autoload a constant using a relative path.

It could look like:

```
autoload_relative :MyConst, 'models/my_const'
```

My proposal raises two questions:

1) what's the future of autoload?

I believe that autoload has been there for years, it is used successfully and has no real alternative.

I looked at a sample of 430 top gems (took the 500 top ranked according to Libraries.io, removed those that I failed to process). The number of those gems that appear to use autoload at least once is 94 of those (22%).

The number of lines in the code where autoload is called can be quite big. The top 5 are:
vagrant: 235
yard: 206
ffaker: 155
aws-sdk: 152
rdoc: 92

This is a minimum bound, as some gems might be using loops, my processing would only detect the one place in the code with autoload.

2) are many autoladed paths relative?

My preliminary numbers indicate that of the 94 gems using autoload, at least 75 are autoloading some relative files. That's a lower bound, as my algorithm is pretty crude and will only count the simplest cases as being relative. An example of gem my algorithm does not detect is yard, because the author wrote a small method to map the relative paths to global paths (code here: https://github.com/lsegal/yard/blob/master/lib/yard/autoload.rb#L3)

Of those where my processing detects the relative requires, a vast majority are relative. The average is that 94% of autoloaded files are relative and would benefit from require_relative

In summary: I am convinced that autoload should remain in Ruby indefinitely. autoload_relative would actually be more useful than autoload. Even if the future of autoload remains uncertain, I would recommend adding autoload_relative; if it is ever decided to actually remove autoload, removing autoload_relative would not really add to the (huge) burden of gem maintainers.

History

#1 - 11/22/2018 05:40 AM - shevegen (Robert A. Heiler)

I do not have much to add to autoload_relative as such, although I would like to mention that, similar to require_relative, the shorter variants are easier to write - e. g. "require 'path'" versus "require_relative 'path'"; similar to autoload versus autoload_require.

However had, more importantly to the issue at hand, I think the functionality that autoload() offers should be retained. That does not necessarily mean that the name as such should remain (I am neutral about it; I use autoload myself but I don't mind either way, whether it is there or not), but the functionality can be useful. Sometimes in large projects, using a linear setup e. g. require one project after the other, can take quite some time to startup. This may normally not be a problem, but I found in one of my projects where I make use of a REPL, such as the user starting a shell,

before typing commands, I prefer a fast startup time. I was experimenting
with two approaches: one was via autoload, the other was a strange thought
of using Thread.new to load addons that take time, and check whether
the code was available or not via e. g. Object.const_defined? and something
like that. This would allow the user to start typing stuff, but not all
parts of the shell would work instantly; it may take up to 7 seconds or
so before all commands would available from that shell. (If anyone is
wondering why that takes so long, I literally have one admin-like shell
interface that pretty much loads ALL the gems and .rb files that I may
ever need; and I was mostly just experimenting anyway.)

The autoload part of the code was actually quite simple; I sort of defined
all autoloads that this shell would use in a single .rb file and not
worry about it lateron.

I think Hiroshi Shibata made some suggestion some time ago about extending
require (or the require-family in ruby); I myself also wonder how to
suggest some more flexible way of requiring files, including perhaps
autoload-functionality there, rather than autoload() as such; or the
ability to use "identifiers" that are mapped on a per-project basis
to a .rb file, so that we can use something like a single :symbol to
require files even from outside of a project, rather than rely on a
hardcoded path to .rb files that may change - but I am going off-topic.

I just wanted to mention this. So as summary, I think autoload-functionality
should remain; as to whether the name autoload() should remain or not
is, I think, secondary (to me at the least).

I understand that your suggestion is to extend on the autoload-family
with your suggestion :) - but I am really neutral about it either way,
so I don't mind what happens there. (Actually, for my idea of another
require way, I thought of identifiers via something like
require_project 'rack|base' or require_project 'rack@base' or something
like that, but I did not like my own API suggestion so I did let this
slide; I still think it would be nice to have a way to require .rb
files without having to hardcode paths to .rb files).

#2 - 11/23/2018 08:32 AM - matz (Yukihiro Matsumoto)

I do not like autoload for its indeterministic nature (#5653).
It caused issues (#10892, #11384, #12688) and they are hard to solve.
But autoload is so widely used (especially in Rails), we couldn't have removed it.

So, even though I understand the value of your proposal, I don't think it's wise to make imperfect autoloading more convenient.

Matz.

#3 - 11/23/2018 05:35 PM - Eregon (Benoit Daloze)

matz (Yukihiro Matsumoto) wrote:

> I do not like autoload for its indeterministic nature (#5653).
> It caused issues (#10892, #11384, #12688) and they are hard to solve.
> But autoload is so widely used (especially in Rails), we couldn't have removed it.

It seems all of these 3 bugs are solved nowadays.

#5653 mentions "But autoload itself has fundamental flaw under multi-thread environment.", but does not give an example.
What's the fundamental flaw, and what example fails due to it?

I thought autoload would make no effort to save cases like accessing a constant while the module is being defined,
but it actually appears that MRI handles that case fine, and only makes the constant visible to other threads once it's fully defined.

Autoload is a nightmare implementation-wise (also, is it documented anywhere how it works? The code can't be said to be easily readable).
But, if it must remain for compatibility, then I agree with Marc-André that it should not be a reason to not improve autoload (i.e., I think we should add autoload_relative).

#4 - 11/23/2018 06:15 PM - jeremyevans0 (Jeremy Evans)

matz (Yukihiro Matsumoto) wrote:

> I do not like autoload for its indeterministic nature (#5653).
> It caused issues (#10892, #11384, #12688) and they are hard to solve.
> But autoload is so widely used (especially in Rails), we couldn't have removed it.

autoload functionality is fairly easy to remove while remaining backwards compatible syntax-wise, just have autoload ignore the first argument and require the second. :)

autoload is used far less than omitting braces for literal hashes as the final argument to a method (#14183), so if autoload can't be removed due to current usage, that shouldn't be either. :)

#5 - 11/23/2018 06:28 PM - rafaelfranca (Rafael França)

Rails plans to implement its reload feature that today is based on const_missing using autoload, so we would prefer if that feature would not be removed, and instead improved. Our experience with const_missing is that it as indeterministic as autoload in a multi-threaded environment and its behavior is less intuitive than autoload. The read of autoload_reloader explain a little what are the advantages of autoload compared with const_missing today. https://github.com/Shopify/autoload_reloader#why-is-using-moduleautoload-awesome-compared-to-const_missing

#6 - 11/23/2018 08:31 PM - Eregon (Benoit Daloze)

jeremyevans0 (Jeremy Evans) wrote:

> autoload functionality is fairly easy to remove while remaining backwards compatible syntax-wise, just have autoload ignore the first argument and require the second. :)

In theory, but not in practice.
That was the initial implementation of #autoload in TruffleRuby but many gems fail that way, because they rely on those files being loaded later (e.g., depending on a method defined in the file, after the autoload).

#7 - 12/10/2018 07:10 AM - naruse (Yui NARUSE)

- Target version deleted (2.6)

#8 - 01/27/2019 04:12 PM - MSP-Greg (Greg L)

I think autoload_relative should be added.

At present, opinion seems to be that autoload cannot be removed.

Additionally (and a separate discussion), if autoload_relative is added to 2.7, might it provide a path to deprecating/removing autoload sometime in the future?

#9 - 02/07/2019 06:01 PM - marcandre (Marc-Andre Lafortune)

Matz, given the renewed support for autoload, would you please reconsider the addition of autoload_relative?

As a reminder, the top Ruby gems using autoload are doing it for relative files in 94% of the cases.

#10 - 04/09/2020 07:59 PM - brodock (Gabriel Mazetto)

autoload is important for big rails applications, as for things like Rake scripts, being able to load only what is really being used can shave several seconds.

outside of rails world, you are forced to mess with $LOAD_PATH constant which is not fun.

autoload_relative seems to be a logical step here.