Ruby master - Feature #15592

mode where "autoload" behaves like an immediate "require"

02/07/2019 06:58 AM - akr (Akira Tanaka)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

How about a feature to switch "autoload" behavior to "require" immediately.

autoload is a feature for lazy loading.

matz dislikes autoload as [Feature #5653].
I heard that he dislikes class (and other) definitions at arbitrary timing.
I agree that eager loading is safer than lazy loading.

However, lazy loading realize shorter loading time and
it makes development cycle shorter.
It is more important for larger applications as Eregon said in
https://bugs.ruby-lang.org/issues/5653#note-39 .
It is especially important when library loading causes I/O (code generation from DB schema).

These two, safety of eager loading and easier development of lazy loading, conflicts.
But if we can distinguish production mode and development mode,
we can enjoy both benefits.

So, I propose a feature to select autoload behavior from two modes:

- autoload behaves as lazy loading as now in development mode
- autoload behaves as eager loading (immediately invokes "require") in production mode.

There are several idea to switch the mode:

- $AUTOLOAD_MODE = :eager or :lazy
- RubyVM.autoload_mode = :eager or :lazy
- ObjectSpace.autoload_mode = :eager or lazy

I'm not sure there is a good enough one in above list, though.

Related issues:

Related to Ruby master - Feature #5653: "I strongly discourage the use of aut...	**Closed**

History

#1 - 02/07/2019 07:02 AM - akr (Akira Tanaka)

- Related to Feature #5653: "I strongly discourage the use of autoload in any standard libraries" (Re: autoload will be dead) added

#2 - 02/07/2019 02:35 PM - jeremyevans0 (Jeremy Evans)

akr (Akira Tanaka) wrote:

> How about a feature to switch "autoload" behavior to "require" immediately.

I would definitely use this, or something similar that immediately references all constants that would be autoloaded, as it would make it much easier to use libraries that use autoload in applications using chroot(2).

#3 - 02/07/2019 06:13 PM - Eregon (Benoit Daloze)

This sounds interesting, and would definitely help debugging autoload code.

As a note, it can already be achieved on existing releases by monkey-patching Kernel#autoload and Kernel.autoload.

I'm against RubyVM.autoload_mode as long as RubyVM is supposed to be MRI-only.

Global variables seem generally deprecated.

Maybe it should be a command-line switch/"feature" like frozen string literals: --enable-eager-autoload?

#4 - 02/07/2019 08:21 PM - fxn (Xavier Noria)

Many people do not realize that in order to eager load a project tree you need to autoload. Talking about the general case.

When Zeitwerk eager loads, it issues require calls, but the autoloads are set orderly still because otherwise it would not work generally.

In a Rails application, for example, you may have something like

```
module CountriesHelper
  # Cache the countries table for performance.
  ALL = Country.all
end
```

or explicit namespaces as in

```
class Hotel < ActiveRecord::Base
  include Hotel::Pricing
end
```

In the case of the proposal here, you could have (schematic):

```
# m.rb
module M
  autoload :X, 'm/x'
  autoload :Y, 'm/y'
end
```

and then

```
# m/x.rb
module M
  module X
    include Y
  end
end
```

You need to resolve M::Y before its autoload has been set.

The key observation is that <u>any</u> constant can be referenced at the top-level of execution.

I think this proposal won't work in the general case.

#5 - 02/07/2019 08:45 PM - Eregon (Benoit Daloze)

fxn (Xavier Noria) I updated your comment a bit for clarity by adding filenames.

Indeed, autoloading eagerly is not as simple as redefining autoload as require.
I should have remembered, because we did that originally in TruffleRuby and quickly realize it just doesn't work (NameError in scenarios like above).

Would it be enough to record all autoloads, and when the application is "loaded" trigger them all?
We probably need some help from the application to know when it's "loaded".
Maybe resolving autoloads every time a top-level require returns is a good enough approximation for most cases?

#6 - 02/07/2019 08:47 PM - Eregon (Benoit Daloze)

An idea: a simple interface could be SomeNamespaceToBeDefined.eager_load_autoloads, which is then invoked by the user after the application is "loaded".

#7 - 02/07/2019 08:49 PM - Eregon (Benoit Daloze)

About where to define such a method, I'd propose on the Kernel singleton class (i.e., Kernel.eager_load_autoloads), since #autoload and #require are defined there too (but not as a Kernel instance method since it wouldn't be a frequently-used method).

#8 - 02/07/2019 08:58 PM - jeremyevans0 (Jeremy Evans)

Eregon (Benoit Daloze) wrote:

> About where to define such a method, I'd propose on the Kernel singleton class (i.e., Kernel.eager_load_autoloads), since #autoload and #require are defined there too (but not as a Kernel instance method since it wouldn't be a frequently-used method).

I think this is a good idea. Some libraries already implement such a feature themselves: https://github.com/mikel/mail/blob/master/lib/mail.rb#L53

#9 - 02/08/2019 12:38 AM - akr (Akira Tanaka)

Eregon (Benoit Daloze) wrote:

> An idea: a simple interface could be SomeNamespaceToBeDefined.eager_load_autoloads, which is then invoked by the user after the application is "loaded".

Great.
I feel it is better than my idea.

#10 - 02/08/2019 04:52 AM - duerst (Martin Dürst)

- Subject changed from mode that "autoload" behaves "require" immediately to mode where "autoload" behaves like an immediate "require"

#11 - 02/08/2019 06:54 AM - fxn (Xavier Noria)

Zeitwerk has an interface to eager load similar to what it is being suggested, but it is able to do so easily because it controls the autoloads that it sets based solely in the file system.

A generic way to register and eager load as outlined above seems like a chicken and egg problem to me, because you have to evaluate the files in order to execute the autoloads, no? You need to eager load at least a bit to register the autoloads that trigger eager loading.

TBH, the experience that I like is Zeitwerk's (that is why I wrote it :), just don't do anything! No requires, and no autoloads. Everything available everywhere, efficiently, and without any work needed by the programmer.

#12 - 02/09/2019 01:36 PM - akr (Akira Tanaka)

fxn (Xavier Noria) wrote:

> Zeitwerk has an interface to eager load similar to what it is being suggested, but it is able to do so easily because it controls the autoloads that it sets based solely in the file system.

I guess it needs some convention between constant names and filenames.
I agree it is good if such convention is available. (convention over configuration)
Unfortunately, Ruby itself has no such reliable convention.

> A generic way to register and eager load as outlined above seems like a chicken and egg problem to me, because you have to evaluate the files in order to execute the autoloads, no? You need to eager load at least a bit to register the autoloads that trigger eager loading.

eager_load_autoloads method would load libraries until no autoload constant:

```
while there are at least one autoload constant
  load one of them (it may add more autoload constants)
end
```

This loop should terminate because there are only finite ruby library files.

Initial list of autoload constants are defined before user application before
eager_load_autoloads.

I think this can be implemented without much problems.

#13 - 02/09/2019 05:47 PM - fxn (Xavier Noria)

> I guess it needs some convention between constant names and filenames.
> I agree it is good if such convention is available. (convention over configuration)
> Unfortunately, Ruby itself has no such reliable convention.

Yes.

> Initial list of autoload constants are defined before user application before
> eager_load_autoloads.
> I think this can be implemented without much problems.

Agree.

#14 - 02/10/2019 03:20 PM - MSP-Greg (Greg L)

I'm very in favor of some type of switching mechanism to allow 'autoload' to switch between immediate & lazy loading.

I'll call it 'AorR', for 'autoload or require?'. I'm confused about the desired scope of this. Given previously:

```
# m.rb
module M
  autoload :X, 'm/x'
  autoload :Y, 'm/y'
end
```

Two possible options:

1. Adding an optional parameter to autoload to control whether it acts like a require statement.

2. Adding a mechanism to control AorR within a file/module.

Regarding option 1:

It seems like too fine grained a level of control. After all, one could just require specific files. Then again, see 'Off-Topic'

Regarding option 2:

I think the idea of being able to place all of the AorR switches in one place (affecting all autoload statements in the app) would be very helpful. One might also like them in separate places. Maybe something like a (global) array that contains the namespaces (stored as symbols or strings) where autoload statements switch to requires? I assume this would become much more complex if it needed to affect previously loaded files.

Off Topic:

The coupling between files and objects has been somewhat tight in many languages. Obviously, it's not necessarily a one-to-one relationship (example - the io/* std-lib's). But, one could state that a goal of a higher level language would be to abstract them. Hence, as an example, an application (or a gem) could totally change their file structure without breaking any applications that use it.