Ruby master - Feature #15565

Circular dependency warnings - suggestions/ideas to improve the output from ruby

01/25/2019 11:33 PM - shevegen (Robert A. Heiler)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

Hello ruby folks (and everyone else) reading this suggestion.

I will start with a mini-summary first, a TL;DR - this proposal suggests to "make it easier/better for us ruby users to solve/resolve/handle circular dependency warnings" or situations similar to these cases.

For example, take three files, called - "a.rb", "b.rb", "c.rb", without the "" quotes.

a.rb has a require line on b.rb, which in turn has a require line on c.rb, which in turn has a require line on a.rb. So they are inter-dependent on one another - a circular dependency situation.

Before I describe the suggestion in more detail, I will take shyouhei's general advice and describe the primary use case, or problem domain that I sometimes face in reguards to circular dependency warnings or circular dependency situations in a ruby project.

The next section thus is an attempt to describe the problem domain, in particular from my point of view (I can not talk for others, but I can describe my use case or problem):

In my larger ruby projects, in particular for projects where you may have quite a lot of files, say, ... upwards of +50 .rb files, I sometimes have to re-arrange the project.

When I write to "re-arrange" the project, this means to rewrite some old code, sometimes delete old code, sometimes add new code, and moving files around, creating or removing or renaming directories. Things like that.

I guess everyone is doing this in general, even on windows (just when you use a GUI to do most of this), but we also have to pay attention in general as to where files are, so that we can find them. In ruby this is similar, where we are often using require, require_relative or load - and we have to know at the least the relative path to other .rb files. And the file also must exist at that position, too (though you can rescue non-existing files through a LoadError, of course).

Handling a ruby project is not so difficult when the project is in an overall good shape - but I also have old code which is of a lower quality. And, even more importantly, I make mistakes too. Sometimes I end up with circular dependency warnings because I may require a .rb file that may pull in another .rb file which may pull in another file ... I guess this may happen for other people too sometimes. To me it is not always instantly obvious which .rb file is the culprit. Sometimes I only notice this at a later time.

This does bring me sometimes into a situation where I have a circular dependency, and it is not very easy to see which file depends on which other file - at the least not for me.

Sometimes I also forget how I structured a project after some time, and it may

not always be easy to clean up a large project, due to inter-dependencies and similar constraints. I have to think which .rb files should be loaded first, for example. This is not so difficult, but it still takes time and concentration.

Such situations are not always a lot of fun to resolve. They also take away time, since we may have to figure out which .rb file depends on which other .rb file. (In the event where the project has been written by me, it is my own code, so I blame myself for problems of course; but my general idea is that this part of ruby could perhaps be improved, a bit similar how the did-you-mean gem may have improved spotting typos in ruby code, or how rubocop can warn about certain code layout issues. Things like that.)

There are of course some strategies to cope with a situation like a circular dependency. A simple one that I have found works fairly well is to ... try to avoid any situation that can lead to a circular dependency - which I found works best when you try to be really minimal when starting with a (new) project.

You can start with the most important basic .rb files, and slowly extend from this point onward, by adding more code.

For example, I often create base-files that I need to re-use, such as in a way to keep track of the base directory of the given project (the one from where the other .rb files may be called); then constants that are re-used within the given project. Following this I may then add a base class from which I may be able to subclass, followed by some other toplevel methods (such as "Foobar.create_directory()"), and then often more specialized classes that do more project-specific activities.

When I approach projects in this way, I rarely end up with a circular dependency warning

- but sometimes I still make mistakes, and in fact when I was adding functionality to one of my larger projects a few hours ago, I had a circular dependency warning. This was also the primary reason, or the main trigger, for writing this issue suggestion here. :)

If we look back at, I think, ruby 1.8.x, we did not have circular dependency warnings. At the least I do not remember that we had ...

While this may not be a perfect situation either (since avoiding circular dependency will also lead to cleaner, clearer and more consistently based projects), it was easier for me to not have to think about circular dependency situations. When ruby 1.9.x or 2.0.x emerged, I also had to resolve several of these circular dependency problems, which was not as much fun. It's still not that much fun either, which is a major reason why I wrote the suggestion here.

On a sidenote - I did want to suggest a better/different way before in how we can handle/cope with larger ruby projects, many months ago already. One smaller component was to make or ideally, avoid, circular dependency warnings/situations. But I still do not have a good proposal for improving the way how ruby deals with lots of .rb files in a new/better way, so this suggestion here is only about circular dependency warnings. Perhaps the current status quo situation could be improved in regards to circular dependency warnings.

I will next try to briefly describe what I would typically see when I invoke a large ruby project that has circular dependencies, from the commandline.

I may see something like 2 up to 3 full-terminal sized output, where ruby would tell me that there is a circular dependency in effect.

I always run my .rb files with the -w flag, so ruby sort of puts me into the "mood" to get rid of any warnings here. I like to use -w, so I will always try to resolve a situation rather than avoid it, even if it is "just" a warning.

From the output that ruby printed, I had a vague idea which file would complain, but looking at that file I noticed that it would point to another file, which in turn would point to several other files which may point to lots of other .rb files ... and then I am confused.

And to be honest, I am not clever enough to easily figure out which file is to be blamed, from the generated output alone. This is another reason why I often "re-do" a project from the get-go, and put things back into place one-by-one - a bit as if you were to repair a mechanical clock that has many tiny pieces, where you re-build it systematically step-by-step. It's still tedious, though, and not a lot of fun.

It is not ruby's fault that my projects are not always perfect; the mistake is a user error in the end. But the situation is not a lot of fun, and I liked the approach in 1.8.x because ... I did not have to deal with these situations. Avoidance-behaviour from me, and I could stay lazy. :D

In fact, I am rewriting the project where I had this circular dependency warning right now, starting with the "bottom-to-top" approach described above; and I know that I will eventually resolve this problem, because I did so many times before, and it always worked. It also made the new code cleaner, and clearer, because I would often also improve it as I would "re-assemble" things - but it still takes quite a lot of time, and is not a lot of fun. I would like to make it easier to resolve such a problem.

What are the problems that I see?

I think I can only mention two points, a smaller one and then a larger one:

(1) Part of the problem may have to do with the changed way how errors are shown on the commandline; the older output may have been a bit less confusing for me, but I do not feel that strongly about it, only may have a slight preference for the older variant. But this is probably not the large issue, as the warning may still not be very helpful, so let's go to the larger problem perceived (2).

(2) I think the larger issue is that the output related from circular dependency warnings isn't too terribly useful as it is. If you compare the situation to the did-you-mean gem, or rubocop warnings in general, then the latter two are a lot more useful to me.

I do not have a good suggestion as to how to make the output from circular dependency warnings much better, but I would think that something like the did-you-mean gem would possibly be useful here. There could be suggestion, at the least into which files may have a problem; and perhaps even so which files are involved in the circular warning. That does not have to be a full stack trace, mind you; just something that may help get ruby folks started.

For example, a table or tabular output of which file may pull in which other file, sorted in some way could be used. The most problematic file could be shown on top of that table output or something like that (e. g. the file that may impact many other .rb files). To not make the output too long, perhaps only a maximum of 6-8 lines should be shown here. Ideally this would be enabled to be the default in a future ruby version (if the changes are approved), or it could be a gem, similar like the did-you-mean gem. That way people could decide on their own what they would prefer (since I like the did-you-mean gem, I always have it on).

When I write "the most problematic file could be shown on top", this may be a bit similar in how gcc or llvm/clang may indicate where an error is, in a given C or C++ file. Like the output with the " ^~~~~" indicator line; I read that LLVM shows somewhat better messages here.

Coming back to ruby here, I would like to see which files are the one that are directly involved with a given circular dependency situation. For example, if it is these three files a.rb, b.rb and c.rb given out, the output could be something like:

```
    a.rb -> depends on b.rb
    b.rb -> depends on c.rb
    c.rb -> depends on a.rb
                       ^^^
```

The three ^ could be colourized in red (but this may be optional). Perhaps we could also get some more detailed output, like a verbose-flag, or the like. A small treeview showing which file may depend on which other file; to not show too much perhaps only three levels of directory structure, at max, should be

checked. And, as said, perhaps only some .rb files checked that way on a given run.

Perhaps only show for the very last c.rb that it has a circular dependency - this could simplify the suggestion a little.

Of course it is up to the ruby user to resolve any situation related to circular dependencies in the end, but any extra information ruby could give could be helpful, in my opinion.

The above just shows some suggestions, of course; other layouts may be helpful as well - for example, it could be counted how often a file is required, or how many circular dependencies that file has, and this finding be reported, so that the ruby user can first look at such a file (biggest troublemaker).

I understand that the above may not be very trivial to write code for as of yet, but part of my suggestion is to get the ruby core team aware that there might be situations where parts of ruby could possibly be improved (perhaps in ruby 3.0).

I remember the time when we did not have the did-you-mean gem, and in fact, many years ago I mentioned that I sometimes accidentally write "def intialize", and the did-you-mean gem also handles such a situation. So sometimes it takes a while for change to happen. :)

(I also understand that not everyone may wish to see more verbose output in regards to circular-dependency warnings perhaps, which is why a gem may be ideal; but part of my suggestion is also to try to see that MRI ruby could be improved in this regard. Perhaps some of you have suggestions how to make MRI ruby behaviour more useful or better here.)

My hope is that we can, in the long run, lessen any frustration with circular dependencies in ruby; and, even more importantly so, be able to more easily/quickly resolve these situations. It is of course a user error in the end, but some projects are more complex than others, so it may be more likely that you may add a circular dependency without this being intentional.

I finally found the time to go ahead and write this suggestion. Sorry for it being long - the TL;DR comes again at the end to help anyone who only wants to jump through it and perhaps quickly make a short comment:

- Please consider making it easier/better to resolve/handle circular dependency warnings in ruby, especially in larger ruby projects. Possibly with visual output/identifiers/cues similar to the did-you-mean gem.

History

#1 - 01/25/2019 11:36 PM - shevegen (Robert A. Heiler)

(The output here on redmine is a bit different to how I wrote it in a local editor - sorry for some of the overflowing lines; I tried to align it to ~80 characters. I am not sure how to edit the first thread, only replies to an issue.)