Ruby master - Bug #16951

Consistently referer dependencies

06/11/2020 11:06 AM - vo.x (Vit Ondruch)

Status:	Open		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 2.7.1p83 (2020-03-31 revision a0c7c23c9c) [x86_64-linux]	**Backport:**	2.5: UNKNOWN, 2.6: UNKNOWN, 2.7: UNKNOWN

Description

It seems that the default gems interdependencies in Ruby are mess. Years ago, when JSON was merged into StdLib, there was big movement and everybody dropped their references to JSON "because it is part of StdLib and therefore it is not needed". I always thought that removing the references was mistake.

Now, there are other interesting cases. Let me name two I know about:

1) REXML is going to be removed from default gems in Ruby 2.8, so some packages already started to introduce the dependency explicitly 1. So once somebody uses Kramdown on older Ruby, the external REXML of whatever version is going to be used.

2) There are also gems in StdLib, such as IRB, which are specifying their dependencies in .gemspec file.

This is unfortunately causing very inconsistent user experience, depending if RubyGems are enabled/disabled, if one is using Bundler or not, if somebody explicitly states something somewhere and what dependencies are transitively pulled in.

I would really appreciate, if Ruby upstream finally paid attention to this problem. My suggestion is that if some gem depends on some other gem, this dependency should be always explicitly stated in the .gemspec file. This would provide clear precedence and guideline to others. This would save all possible surprises and hidden issues, suddenly using dependency of different version, which is pulled in transitively.

History

#1 - 06/11/2020 01:31 PM - deivid (David Rodríguez)

For what it's worth, I also agree that once a library is gemified and promoted to a default gem, gems depending on it should add a sane explicit dependency on them in their gemspec, protecting them, for example, from eventual major version releases with breaking changes.

I've seen this kind of breakage, for example, with BigDecimal 2.0 which removed BigDecimal.new. Had dependant gems had their dependency explicited in their gemspec as s.add_dependency "bigdecimal", "~> 1.0", no breakage would've occurred.

#2 - 06/15/2020 06:59 PM - Eregon (Benoit Daloze)

deivid (David Rodríguez) wrote in #note-1:

> Had dependant gems had their dependency explicited in their gemspec as s.add_dependency "bigdecimal", "~> 1.0", no breakage would've occurred.

OTOH bigdecimal 1.x will probably not work on the newer Ruby versions, so it's not great either.
But at least it'd break when trying to support a new Ruby version, not before, which is nice.

I'm concerned that C extensions of default gems are only tested against the latest Ruby, and so e.g., might not work on older Rubies.

Also stdlibs can have a different implementation in alternative Ruby implementations if wanted/needed, but that becomes not possible/hacky if there is an explicit dependency on a default gem like bigdecimal ~> 1.0.
Take the example of JRuby and the bigdecimal gem for instance, it fails to gem install.

The way forward is probably to explicitly handle alternative Ruby implementations in all default gems as needed (which might be by using the vendored/stdlib version of BigDecimal on JRuby, but then it might not actually use the expected version of bigdecimal).

Also, I feel not specifying dependencies of the stdlib gems is better in the sense that it will use the version of the default gem that's shipped with that Ruby, and was extensively tested.
Using any other version will not achieve the same level of testing, and risk incompatibilities.

I guess any way long term all default/bundled gems will need support for alternative implementations more explicitly, but right now it's just not the case.

Regarding TruffleRuby, the plan is to support default gems C extensions as much as possible, yet it might be difficult for some default gems because some of them reach very deep in the MRI implementation, more than the average gem with a C extension.

headius (Charles Nutter) Any thoughts on this?

#3 - 06/17/2020 08:34 AM - deivid (David Rodríguez)

> OTOH bigdecimal 1.x will probably not work on the newer Ruby versions, so it's not great either. But at least it'd break when trying to support a new Ruby version, not before, which is nice.

This is the standard experience when your gem breaks on the newest rubies, and as you point out it's better than the alternative.

> I'm concerned that C extensions of default gems are only tested against the latest Ruby, and so e.g., might not work on older Rubies.

Why is this if I may ask? I tend to recall the default gems upstreams testing against several rubies according to their support range.

> Also stdlibs can have a different implementation in alternative Ruby implementations if wanted/needed, but that becomes not possible/hacky if there is an explicit dependency on a default gem like bigdecimal ~> 1.0. Take the example of JRuby and the bigdecimal gem for instance, it fails to gem install.
>
> The way forward is probably to explicitly handle alternative Ruby implementations in all default gems as needed (which might be by using the vendored/stdlib version of BigDecimal on JRuby, but then it might not actually use the expected version of bigdecimal).

I understand that there are some issues at the moment with default gems and alternative implementations. In those cases, I agree it's probably better to not specify requirements if you want to support the alternative implementations.

> Also, I feel not specifying dependencies of the stdlib gems is better in the sense that it will use the version of the default gem that's shipped with that Ruby, and was extensively tested.
> Using any other version will not achieve the same level of testing, and risk incompatibilities.

I don't really buy this argument. The most stable version of a library should be the latest stable, because it includes the latest bug fixes, security releases, and so on. I don't think that's a principle we should give up on, specially since most default gems are still maintained by the core team itself.