Ruby - Feature #9704

Refinements as files instead of modules

04/04/2014 08:41 PM - trans (Thomas Sawyer)

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

Description

If refinements are to remain file-scoped, then it would be more convenient if using worked at the file level, akin to require, rather than behave like a module include. For instance, instead of:

```
# foo.rb
module Foo
  refine String do
    def some_method
      ...
    end
  end
end

require 'foo'
using Foo
```

We could do:

```
# foo.rb
class String
  def some_method
    ...
  end
end

using 'foo'
```

This would make require and using, in a certain sense, *polymorphic* --if we require it will extend the classes directly, but if using then they will be refined instead.

Related issues:

Related to Ruby - Feature #12737: Module#defined_refinements	**Closed**

History

#1 - 04/06/2014 12:07 PM - matz (Yukihiro Matsumoto)

So your key idea is sharing implementation between monkey-patching and refinements.
The motivation behind introducing refinements is discouraging monkey-patching. So it is not attractive for me to something that helps monkey-patching.

What is your intention behind the proposal? Maybe from monkey-patching to refinements transition?

Matz.

#2 - 04/06/2014 04:52 PM - trans (Thomas Sawyer)

Yes, the transition from monkey-patching to refinements is a major part of the intention.

But I also do not expect monkey-patching to ever completely go away. (Do you?) Monkey patching is more convenient, in that it can be done via one require for an entire project, where as refining has to specified per-file. So there will be cases where monkey-patching is still preferred.

Besides personal projects and small end-user tools, a good example, I think, is ActiveSupport. While some of it could be used as refinements, I suspect much of it will remain core extensions b/c it represents Rails' DSL, so to speak. And the parts that can become refinements, I would imagine users would still have an option to load them in as extensions.

Even so, I think the main intent is to ask the question: Do refinements *need* to be modules? If there is no compelling reason for them to be so, then making refinements file-based would simplify reuse and allow us to shed unnecessary boiler-plate.

#3 - 04/22/2014 02:39 AM - trans (Thomas Sawyer)

Just FYI, I created this project https://github.com/rubyworks/reusing. I don't much care for what I had to do to implement it, i.e.

- it's not possible to simply override #using
- had to use my finder gem to find libraries
- had to use clunky gsub and eval

But it appears to be working.

#4 - 04/22/2014 03:57 PM - trans (Thomas Sawyer)

I realized there is a downside to this approach.

```
# a.rb
require 'b'
class String
  def ab
    self + "a" + b
  end
end

# b.rb
class String
  def b
    self + "b"
  end
end
```

So what would happen with b.rb when using 'a'? Should the require 'b' become a using? If so, then what happens when it requires something that is not an extension, e.g. set?

If that is an unsolvable issue. The only fix, it seems, would be a way to have something like require_or_using which would act according to the initial loading method used. And that starts to look pretty ugly.

#5 - 04/25/2014 07:00 AM - shugo (Shugo Maeda)

Monkey-patching style has another problem that super cannot be supported in monkey patching.

I prefer another approach to extend classes both globally and locally in the same manner.
How about to provide a way to activate refinements globally?

using Foo, global: true

Refinements in Foo are activated globally.

Module#refine creates a module as a refinement, so it can be globally activated by Module#prepend-ing that module to the target class.

#6 - 04/25/2014 03:46 PM - trans (Thomas Sawyer)

Ah, attack the problem from the other way round. That's a great idea!

#7 - 08/03/2016 04:54 AM - trans (Thomas Sawyer)

Has any more thought been given to this? I had a recent request to have Ruby Facets to work as refinements, but I hesitate b/c it literally means creating a second copy of every extension method.

#8 - 08/03/2016 05:06 AM - shugo (Shugo Maeda)

- Assignee set to matz (Yukihiro Matsumoto)

Thomas Sawyer wrote:

> Has any more thought been given to this? I had a recent request to have Ruby Facets to work as refinements, but I hesitate b/c it literally means creating a second copy of every extension method.

What do you think of the following idea, Matz?

> Monkey-patching style has another problem that super cannot be supported in monkey patching.
>
> I prefer another approach to extend classes both globally and locally in the same manner.
> How about to provide a way to activate refinements globally?

```
using Foo, global: true
# Refinements in Foo are activated globally.
```

Module#refine creates a module as a refinement, so it can be globally activated by Module#prepend-ing that module to the target class.

#9 - 08/08/2016 03:59 PM - pabloh (Pablo Herrero)

It would be interesting to be able apply a refinement to a whole app (without its gems) or inside a single gem without propagating to the rest of the app which loaded it.
Although I'm not really sure about the syntax or API to do it.

#10 - 09/08/2016 05:38 AM - shugo (Shugo Maeda)

- Related to Feature #12737: Module#defined_refinements added

#11 - 10/11/2016 06:36 PM - trans (Thomas Sawyer)

Shugo Maeda wrote:

> Monkey-patching style has another problem that super cannot be supported in monkey patching.
>
> I prefer another approach to extend classes both globally and locally in the same manner.
> How about to provide a way to activate refinements globally?
>
> using Foo, global: true
>
> # Refinements in Foo are activated globally.
>
> Module#refine creates a module as a refinement, so it can be globally activated by Module#prepend-ing that module to the target class.

That last line just clicked with me, and I see old problem. This only works at one level. How would one write a refinement that adds methods to class level and instance level at the same time?

#12 - 04/03/2024 03:50 AM - hsbt (Hiroshi SHIBATA)

- Status changed from Open to Assigned