Ruby master - Bug #17429
Prohibit include/prepend in refinement modules
12/23/2020 02:22 AM - shugo (Shugo Maeda)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>shugo (Shugo Maeda)</td>
</tr>
<tr>
<td>Target version:</td>
<td>ruby -v:</td>
</tr>
<tr>
<td>Backport:</td>
<td>2.5: UNKNOWN, 2.6: UNKNOWN, 2.7: UNKNOWN</td>
</tr>
</tbody>
</table>

**Description**

include/prepend in refinement modules has implementation difficulties such as #17007 and #17379, and tends to be misleading like #17374.

How about to prohibit it in future versions?

Method copy like #17380 may be more convenient, but it's confusing to use names include and prepend because semantics is different from the original ones.

**Related issues:**

- Related to Ruby master - Bug #17007: SystemStackError when using super inside...
  - Closed
- Related to Ruby master - Bug #17374: Refined methods aren't visible from a re...
  - Rejected
- Related to Ruby master - Bug #17379: Refinement with modules redefinition bug
  - Closed
- Related to Ruby master - Bug #18021: Mixins in Refinements: possibly multiple...
  - Closed
- Related to Ruby master - Feature #18270: Refinement#(extend_object,append_fea...
  - Closed

**Associated revisions**

Revision 66065971 - 10/21/2021 07:31 AM - shugo (Shugo Maeda)

Deprecate include/prepend in refinements and add Refinement#import_methods instead

Refinement#import_methods imports methods from modules. Unlike Module#include, it copies methods and adds them into the refinement, so the refinement is activated in the imported methods.

[Bug #17429] [ruby-core:101639]

**History**

#1 - 12/23/2020 05:44 AM - matz (Yukihiro Matsumoto)

I basically agree. Combination of refinement and include/prepend only cause confusion.

Matz.

#2 - 12/23/2020 11:00 AM - Eregon (Benoit Daloze)

+1 from me!

I think a new Module method to copy all methods to another Module could be useful. Something like A.copy_methods(B).

It seems there is no need to copy constants, because the constant scope of "copied" methods would still be A (and lexical parents).

Actually the method would only do a shallow copy of each method, i.e., still use the same bytecode, etc, so maybe another name than copy_methods would be clearer.

The docs of Module#append_features make it sounds like #append_features would do that, but it doesn't. Actually, #append_features adds the given module in the ancestors chain (include = append_features + included). Would be a good occasion to clarify the docs of #append_features.

#3 - 12/23/2020 05:04 PM - marcandre (Marc-Andre Lafortune)

Is there a proposal to import modules in refinements?

Something like this?

```ruby
module Code
  # ...
end
```
refine Object, import: Code do
  # extra methods
end

I still think that include and prepend within the refine block could have that function.

I think that having a nice way to implement methods that can be used with include or using would help adoption of refinements for gems.

#4 - 12/23/2020 07:19 PM - Dan0042 (Daniel Delorme)
I agree the current situation needs to be fixed, and prohibiting include/prepend is the simplest way. But I also think there has to be a way to achieve what Marc-Andre was trying in #17374.

marcandre (Marc-Andre Lafortune) wrote in #note-3:

refine Object, import: Code do
  # extra methods
end

That looks pretty good I think.

But there's not much benefit to that is there? Having a different name such as import feels cleaner. Using include or prepend within a refine block could result in a warning/error along the lines of "include is not supported in refinements but you can use the almost-equivalent import argument."

#5 - 01/06/2021 01:49 AM - jeremyevans0 (Jeremy Evans)
I've added a pull request for this: https://github.com/ruby/ruby/pull/4029

#6 - 01/06/2021 02:00 AM - jeremyevans0 (Jeremy Evans)
- Related to Bug #17007: SystemStackError when using super inside Module included and lexically inside refinement added

#7 - 01/06/2021 02:01 AM - jeremyevans0 (Jeremy Evans)
- Related to Bug #17374: Refined methods aren't visible from a refinement's module added

#8 - 01/06/2021 02:02 AM - jeremyevans0 (Jeremy Evans)
- Related to Bug #17379: Refinement with modules redefinition bug added

#9 - 01/13/2021 08:08 AM - matsuda (Akira Matsuda)
Calling include in refinement modules has certain use cases.

Here's an actual example.
https://github.com/tomykaira/rspec-parameterized/blob/v0.4.2/lib/rspec/parameterized/table_syntax.rb#L27-L61

This gem uses Module#include to avoid code repetition, which to me looks quite natural and basic usage of Module.

If we prohibit include in refinement modules, can this code still be written equally simply?

#10 - 01/13/2021 11:49 AM - Eregon (Benoit Daloze)
With https://bugs.ruby-lang.org/issues/17429#note-3 it would.
I think it makes sense adding such functionality at the same time as no longer allowing include for refinement modules.

#11 - 01/14/2021 03:15 AM - shugo (Shugo Maeda)
marcandre (Marc-Andre Lafortune) wrote in #note-3:

Is there a proposal to import modules in refinements?

Something like this?

module Code
  # ...
end

refine Object, import: Code do
I prefer the following way, but I'm not sure about the name import.

```ruby
refine Object do
  import Code
end
```

The behavior is similar to Module#mix proposed by Matz before.

Visually it's a more pleasing API than the import keyword, but would it be available in any module or just refinements? If any module, we'd now have three mixin mechanisms: include, prepend, import. IMO that's overly complex. If just refinements, it feels inconsistent. IMO we'll have people asking why they can't use import in classes and modules. As a keyword it's clear this is a refinement-only behavior.

It's enough to changing the class of a module created by refine to the following subclass of Module, isn't it?

```ruby
class Refinement < Module
  [:include, :prepend].each do |name|
    define_method(name) do |*args|
      warn("#{name} in a refinement is deprecated; use mix instead", uplevel: 1, category: :deprecated)
      super(*args)
    end
  end

  def mix(*args)
    ...
  end
end
```

@Eregon (Benoit Daloze)@marcandre (Marc-Andre Lafortune)

Having a Module subclass for Refinements seems nice and useful :+1:

#15 - 01/22/2021 06:30 PM - marcandre (Marc-Andre Lafortune)

I like @shugo's approach too.

For anyone interested, I released the refine_export gem that makes @jeremyevans' nice hack easy to use:

https://github.com/marcandre/refine_export#usage

#16 - 01/26/2021 01:47 AM - shugo (Shugo Maeda)

- Assignee set to matz (Yukihiro Matsumoto)

@Eregon (Benoit Daloze)@marcandre (Marc-Andre Lafortune) Thanks for your feedback.

The remaining issue is the name of the new method.
I came up with the following options:

1. Refinement#import
2. Refinement#mix
3. Refinement#include (different behavior from Module#include)

@matz (Yukihiro Matsumoto) Which do you like, or do you have another option in mind?

#17 - 01/29/2021 09:34 AM - ko1 (Koichi Sasada)
I like Module#mix for all modules (not only for refinement).

#18 - 01/29/2021 12:21 PM - Eregon (Benoit Daloze)
k01 (Koichi Sasada) wrote in #note-17:
I like Module#mix for all modules (not only for refinement).

I think they need different semantics.
For refinements, we will need to do a deep copy of the method, or at least of the inline caches, so that the copied methods see the other refined methods of the refinement module.
That's quite expensive in footprint, but it probably makes sense for this use case with refinements.

For a general Module#mix, I don't think that is needed. Also what's the advantage of a general Module#mix over include/prepend?

#19 - 07/04/2021 10:19 AM - Eregon (Benoit Daloze)
- Related to Bug #18021: Mixins in Refinements: possibly multiple bugs, workarounds are awkward added

#20 - 07/04/2021 10:26 AM - Eregon (Benoit Daloze)
One more issue reported due these confusing semantics of include/prepend inside refine: #18021.

I suggest we warn in 3.1, raise in 3.2.
And I suggest to add Refinement#import, because:

- mix seems to imply other things, and if it's ever added to Module with different semantics we will just get more confusion.
- include seems likely to cause confusion because the behavior would not be Module#include-like. It is also potentially backward-incompatible, raises the question about what would happen for prepend and consistency. And finnaly include would not longer mean "define higher in the ancestor" for this context, i.e., defining the same method in the refine block would replace, not just override).

#21 - 07/04/2021 10:35 AM - Eregon (Benoit Daloze)
I'll mention it here as it may be helpful.
If you want to define the same method in multiple refinements before this is fixed, the current workaround is to use class_eval/module_eval inside refine and have those shared methods in a String.
Not so pretty, but it works and it's simple.

#22 - 07/20/2021 07:53 AM - shugo (Shugo Maeda)
Eregon (Benoit Daloze) wrote in #note-20:
One more issue reported due these confusing semantics of include/prepend inside refine: #18021.

I suggest we warn in 3.1, raise in 3.2.
And I suggest to add Refinement#import, because:

- mix seems to imply other things, and if it's ever added to Module with different semantics we will just get more confusion.
- include seems likely to cause confusion because the behavior would not be Module#include-like. It is also potentially backward-incompatible, raises the question about what would happen for prepend and consistency. And finnaly include would not longer mean "define higher in the ancestor" for this context, i.e., defining the same method in the refine block would replace, not just override).

I've implemented Refinement#import in https://github.com/shugo/ruby/pull/3

In the current implementation, the module in cref is replaced with the refinement like Module#dup, so constants in the imported module are not accessible from the copied methods.

Maybe Refinement#import_methods is a better name if we keep the current behavior.

#23 - 07/23/2021 10:01 AM - Eregon (Benoit Daloze)
shugo (Shugo Maeda) wrote in #note-22:
In the current implementation, the module in cref is replaced with the refinement like Module#dup, so constants in the imported module are not accessible from the copied methods.
Could you show an example that would not work due to that? Methods from the imported module should be able to access constants from the imported module, otherwise I think it is very surprising. They should not be able to access constants from the refinement module, that's fine they were declared in the imported module.

#24 - 07/26/2021 12:02 AM - shugo (Shugo Maeda)

Eregon (Benoit Daloze) wrote in #note-23:

shugo (Shugo Maeda) wrote in #note-22:

In the current implementation, the module in cref is replaced with the refinement like Module#dup, so constants in the imported module are not accessible from the copied methods.

Could you show an example that would not work due to that? Methods from the imported module should be able to access constants from the imported module, otherwise I think it is very surprising. They should not be able to access constants from the refinement module, that's fine they were declared in the imported module.

For me, it's surprising if the imported methods cannot access constants of the refinement. However, constant assignments in refine block define constants not in the refinement but in the outer scope, so it may not be a problem actually.

```
module Ext
  refine Object do
    X = 1 # defines Ext::X
    const_set(:Y, 2) # defines #<refinement:Object@Extension>::X
  end
end
```

Is it enough that the imported methods can access only constants in the original context?

#25 - 07/26/2021 11:12 AM - Eregon (Benoit Daloze)

Yes, I think that's completely fine.

In code, this should work:

```
module Shared
  A = 1
  def foo
    A
  end
end
refine SomeClass do
  import Shared
end
SomeClass.new.foo #=> 1
```

And this should not:

```
module Shared
  def foo
    A
  end
end
refine SomeClass do
  self::A = 1
  import Shared
end
SomeClass.new.foo #=> NameError
```

That would be the equivalent of dynamic rebinding or so, I think nobody expects that, the constant scope has always been lexical (+ ancestors of the first enclosing module).

Could you add tests (or better, specs under spec/ruby) for that?

Then I think it should be good to go.

#26 - 07/26/2021 11:15 AM - Eregon (Benoit Daloze)

In other words, Refinement#import copies methods but does not attempt to change anything lexical, except which refinements are applied to these
methods.

#27 - 07/27/2021 02:37 AM - shugo (Shugo Maeda)
I've changed the behavior and have added tests in https://github.com/shugo/ruby/pull/3/commits/3aaca9217f958640495c11b08948960d1f30

#28 - 09/10/2021 07:42 AM - shugo (Shugo Maeda)
In the current implementation:

- Refinement#import raises an ArgumentError if the specified module has methods written in C.
- Only methods defined directly in the specified module are imported.
- Importing ancestors' methods may be confusing because Refinement#import doesn't work with super.

#29 - 09/16/2021 02:09 PM - mame (Yusuke Endoh)
In today's dev meeting, matz accepted the concept, but wanted to take some time to consider the name import.

#30 - 09/16/2021 02:11 PM - mame (Yusuke Endoh)
BTW, the change seems to add a top-level new constant ::Refinement. I'm not against the addition, but unsure about the impact. Is it okay?

#31 - 09/17/2021 02:57 AM - shugo (Shugo Maeda)
mame (Yusuke Endoh) wrote in #note-30:

BTW, the change seems to add a top-level new constant ::Refinement. I'm not against the addition, but unsure about the impact. Is it okay?

I found a gem named refinement....
https://github.com/square/refinement/blob/master/lib/refinement.rb

#32 - 09/17/2021 09:48 AM - shugo (Shugo Maeda)
shugo (Shugo Maeda) wrote in #note-31:

mame (Yusuke Endoh) wrote in #note-30:

BTW, the change seems to add a top-level new constant ::Refinement. I'm not against the addition, but unsure about the impact. Is it okay?

I found a gem named refinement....
https://github.com/square/refinement/blob/master/lib/refinement.rb

I've created an issue on the project: https://github.com/square/refinement/issues/71.

By gem-codesearch, I've found another gem which defines ::Refinement.


However, it's a gem for very old Ruby versions without Refinements, so I believe there's no problem.

#33 - 09/28/2021 11:21 PM - shugo (Shugo Maeda)
- Status changed from Open to Assigned

shugo (Shugo Maeda) wrote in #note-32:

I found a gem named refinement....
https://github.com/square/refinement/blob/master/lib/refinement.rb

I've created an issue on the project: https://github.com/square/refinement/issues/71.

The maintainer of the gem agreed with introducing the built-in class Refinement.

Matz, is the method name import OK?
I agreed with import_methods, which is more descriptive and clear.

Matz.

Assignee changed from matz (Yukihiro Matsumoto) to shugo (Shugo Maeda)

Status changed from Assigned to Closed

Applied in changeset 108d6606597109b9db535a150606323ce3d8f5750a1f6.

Deprecate include/prepend in refinements and add Refinement#import_methods instead

Refinement#import_methods imports methods from modules.
Unlike Module#include, it copies methods and adds them into the refinement,
so the refinement is activated in the imported methods.

[Bug #17429] [ruby-core:101639]

Related to Feature #18270: Refinement#{extend_object,append_features,prepend_features} should be removed added