Support optional inherit argument for Module#method_defined?

07/27/2018 04:59 PM - jeremyevans0 (Jeremy Evans)

Status: Closed
Priority: Normal
Assignee:
Target version:

Description
Module has many introspection methods for methods and constants that either return an array or return true or false for whether the method or constant is defined. Most of these methods support an optional argument that controls whether to consider inheritance. Currently, the following Module methods support such a argument:

- const_defined?
- constants
- instance_methods
- private_instance_methods
- protected_instance_methods
- public_instance_methods

and the following methods do not:

- method_defined?
- private_method_defined?
- protected_method_defined?
- public_method_defined?

This patch supports such an argument for the *method_defined? methods.

While you can currently work around the lack of support via:

```ruby
mod.instance_methods(false).include?(:method_name)
```

This patch allows the simpler and more efficient:

```ruby
mod.method_defined?(:method_name, false)
```

One case where you want to exclude inheritance when checking for a method definition is when you want to replace a method that may already exist. To avoid a verbose warning, you want to remove the method only if it is already defined:

```ruby
remove_method(:foo) if method_defined?(:foo, false)
define_method(:foo){}
```

You can't call remove_method without checking for the method definition, as that can raise a NameError, and you don't want to include inheritance because remove_method will still raise a NameError if the method is defined by an ancestor and not by the module itself.

Related issues:
- Related to Ruby master - Bug #15597: syscall not returning true from private_... - Rejected
- Has duplicate Ruby master - Feature #10797: `inherit` parameter for `..._defi... - Closed
- Is duplicate of Ruby master - Feature #9322: method_defined? family of of met... - Closed

Associated revisions
Revision bccb24a8 - 08/13/2018 12:42 PM - usa (Usaku NAKAMURA)
Support optional inherit argument for Module#method_defined?

Module has many introspection methods for methods and constants that either return an array or return true or false for whether the method or constant is defined. Most of these methods support an optional argument that controls whether to consider inheritance. Currently, the following Module methods support such a argument:

- `const_defined?`
- `constants`
- `instance_methods`
- `private_instance_methods`
- `protected_instance_methods`
- `public_instance_methods`

and the following methods do not:

- `method_defined?`
- `private_method_defined?`
- `protected_method_defined?`
- `public_method_defined?`

This patch supports such an argument for the *method_defined?* methods.

While you can currently work around the lack of support via:

```
mod.instance_methods(false).include?(:method_name)
```

This patch allows the simpler and more efficient:

```
mod.method_defined?(:method_name, false)
```

One case where you want to exclude inheritance when checking for a method definition is when you want to replace a method that may already exist. To avoid a verbose warning, you want to remove the method only if it is already defined:

```
remove_method(:foo) if method_defined?(:foo, false)
define_method(:foo)
```

You can't call remove_method without checking for the method definition, as that can raise a NameError, and you don't want to include inheritance because remove_method will still raise a NameError if the method is defined by an ancestor and not by the module itself.

[ruby-core:88140] [Feature #14944]

From: Jeremy Evans code@jeremyevans.net

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64337 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64337 - 08/13/2018 12:42 PM - usa (Usaku NAKAMURA)

Support optional inherit argument for Module#method_defined?

Module has many introspection methods for methods and constants that either return an array or return true or false for whether the method or constant is defined. Most of these methods support an optional argument that controls whether to consider inheritance. Currently, the following Module methods support such a argument:

- `const_defined?`
- `constants`
- `instance_methods`
- `private_instance_methods`
- `protected_instance_methods`
- `public_instance_methods`

and the following methods do not:

- `method_defined?`
- `private_method_defined?`
- `protected_method_defined?`
This patch supports such an argument for the *method_defined?* methods.

While you can currently work around the lack of support via:

```ruby
mod.instance_methods(false).include?(:method_name)
```

This patch allows the simpler and more efficient:

```ruby
mod.method_defined?(:method_name, false)
```

One case where you want to exclude inheritance when checking for a method definition is when you want to replace a method that may already exist. To avoid a verbose warning, you want to remove the method only if it is already defined:

```ruby
remove_method(:foo) if method_defined?(:foo, false)
define_method(:foo)
```

You can't call remove_method without checking for the method definition, as that can raise a NameError, and you don't want to include inheritance because remove_method will still raise a NameError if the method is defined by an ancestor and not by the module itself.

[Feature #14944]

From: Jeremy Evans code@jeremyevans.net

Revision 64337 - 08/13/2018 12:42 PM - usa (Usaku NAKAMURA)

Support optional inherit argument for Module#method_defined?

Module has many introspection methods for methods and constants that either return an array or return true or false for whether the method or constant is defined. Most of these methods support an optional argument that controls whether to consider inheritance. Currently, the following Module methods support such a argument:

- `const_defined?`
- `constants`
- `instance_methods`
- `private_instance_methods`
- `protected_instance_methods`
- `public_instance_methods`

and the following methods do not:

- `method_defined?`
- `private_method_defined?`
- `protected_method_defined?`
- `public_method_defined?`

This patch supports such an argument for the *method_defined?* methods.

While you can currently work around the lack of support via:

```ruby
mod.instance_methods(false).include?(:method_name)
```

This patch allows the simpler and more efficient:

```ruby
mod.method_defined?(:method_name, false)
```

One case where you want to exclude inheritance when checking for a method definition is when you want to replace a method that may already exist. To avoid a verbose warning, you want to remove the method only if it is already defined:

```ruby
remove_method(:foo) if method_defined?(:foo, false)
define_method(:foo)
```

You can't call remove_method without checking for the method definition.
definition, as that can raise a NameError, and you don't want
to include inheritance because remove_method will still raise
a NameError if the method is defined by an ancestor and not
by the module itself.

[ruby-core:88140] [Feature #14944]

From: Jeremy Evans code@jeremyevans.net

Revision 64340 - 08/13/2018 01:27 PM - znz (Kazuhiro NISHIYAMA)
Revert “Support optional inherit argument for Module#method_defined?”

Revision 64343 - 08/13/2018 01:48 PM - usa (Usaku NAKAMURA)
Fix problem about notimplemented case
Re-revert r64340, and take care about notimplemented case.

History

#1 - 07/31/2018 02:46 PM - znz (Kazuhiro NISHIYAMA)
If you want to suppress warnings, you can use undef_method instead of remove_method for the time being.

```
module M
  def foo; end
end
class C
  include M
  undef_method(:foo) if method_defined?(:foo)
  define_method(:foo){}
end
```

#2 - 08/09/2018 07:50 AM - matz (Yukihiro Matsumoto)
I think this is a good idea. Accepted.

Matz.

#3 - 08/13/2018 12:42 PM - usa (Usaku NAKAMURA)
- Status changed from Open to Closed

Applied in changeset trunk|r64337.

Support optional inherit argument for Module#method_defined?

Module has many introspection methods for methods and constants that
either return an array or return true or false for whether the method
or constant is defined. Most of these methods support an optional
argument that controls whether to consider inheritance. Currently,
the following Module methods support such an argument:

- const_defined?
- constants
- instance_methods
- private_instance_methods
- protected_instance_methods
- public_instance_methods

and the following methods do not:

- method_defined?
- private_method_defined?
- protected_method_defined?
- public_method_defined?

This patch supports such an argument for the *method_defined? methods.

While you can currently work around the lack of support via:

mod.instance_methods(false).include?(method_name)
This patch allows the simpler and more efficient:

    mod.method_defined?(:method_name, false)

One case where you want to exclude inheritance when checking for a method definition is when you want to replace a method that may already exist. To avoid a verbose warning, you want to remove the method only if it is already defined:

    remove_method(:foo) if method_defined?(:foo, false)
define_method(:foo)

You can't call remove_method without checking for the method definition, as that can raise a NameError, and you don't want to include inheritance because remove_method will still raise a NameError if the method is defined by an ancestor and not by the module itself.

[ruby-core:88140] [Feature #14944]

From: Jeremy Evans code@jeremyevans.net

#4 - 02/11/2019 06:53 AM - nobu (Nobuyoshi Nakada)
- Related to Bug #15597: syscall not returning true from private_method_defined? added

#5 - 01/04/2020 06:46 PM - sawa (Tsuyoshi Sawada)
Actually, this proposal is a duplicate of #10797

#6 - 01/04/2020 08:29 PM - Eregon (Benoit Daloze)
- Has duplicate Feature #10797: `inherit` parameter for `..._defined?` methods in Module added

#7 - 10/06/2021 08:42 PM - jeremyevans0 (Jeremy Evans)
- Is duplicate of Feature #9322: method_defined? family of of methods should support the exclusion of ancestors added

Files

| 0001-Support-optional-inherit-argument-for-Module-method_.patch | 16.5 KB | 07/27/2018 | jeremyevans0 (Jeremy Evans) |