As documented instance_methods(false) works as follows:

module A
  def method1
  end
end
class B
  include A
  def method2
  end
end

p B.instance_methods(false) #=> [:method2]

However, calling protected on the method defined by A, unexpectedly changes the result of instance_methods(false) on B, even though the owner of the method is still A:

module A
  def method1
  end
end
class B
  include A
  protected :method1
  def method2
  end
end

p B.instance_methods(false) #=> [:method1, :method2]
p B.instance_method(:method1).owner #=> A

In contrast, calling private or public on the same method does not cause any changes on the result of B.instance_methods(false).

This feels like a bug in the implementation of instance_methods(false), but, if it is by design, it should at least be documented on Module#instance_methods.

This reproduction script gives the same output all the way from Ruby 2.0 up to Ruby-HEAD: https://wandbox.org/permlink/LqbXMBTYxURRZmDz

Related issues:
Related to Ruby master - Bug #18729: Method#owner and UnboundMethod#owner are... Closed
Related to Ruby master - Bug #18751: Regression on master for Method#== when ... Closed
Related to Ruby master - Feature #11689: Add methods allow us to get visibili... Open

Associated revisions
Revision 58dc8bf8 - 01/14/2022 09:46 PM - jeremyevans (Jeremy Evans)
Add a visibility member to struct METHOD storing the original
method visibility, and use that, instead of taking the visibility from the stored method entry (which may have different visibility for ZSUPER methods).

Consider Method/UnboundMethod objects different if they have different visibilities.

Fixes [Bug #18435]

Revision 8212aab8 - 08/20/2022 11:44 AM - jeremyevans (Jeremy Evans)
Make Object#method and Module#instance_method not skip ZSUPER methods
Based on https://github.com/jeremyevans/ruby/commit/c95e7e5329140f640b649790548576153386d967
Among other things, this fixes calling visibility methods (public?, protected?, and private?) on them. It also fixes #owner to show the class the zsuper method entry is defined in, instead of the original class it references.

For some backwards compatibility, adjust #parameters and #source_location, to show the parameters and source location of the method originally defined. Also have the parameters and source location still be shown by #inspect.

Clarify documentation of {Method,UnboundMethod}#owner.

Add tests based on the description of https://bugs.ruby-lang.org/issues/18435 and based on https://github.com/ruby/ruby/pull/5356#issuecomment-1005298809
Fixes [Bug #18435] [Bug #18729]

Co-authored-by: Benoit Daloze eregontp@gmail.com

History

#1 - 12/26/2021 09:09 PM - jeremyevans0 (Jeremy Evans)
I don't think this is a bug, I think it is expected behavior. When you call protected in the class, it creates a method entry in the class with a different visibility, even if the owner of the method is still the module. Hopefully a committer with more experience can confirm that.

I don't think this should be documented in instance_methods. After all, it is not unique to instance_methods, but all related methods (methods, protected_instance_methods, etc.). If we document this behavior at all, it would be better to document the creation of method entries in doc/syntax/modules_and_classes.rdoc, in the section on Visibility. However, I don't think that this behavior is worth documenting.

#2 - 12/27/2021 12:34 PM - ufuk (Ufuk Kayserilioglu)
I understand why the difference in behaviour is happening, but I respectfully disagree that this is not a bug.

Regardless of how protected is implemented internally, the return value of instance_methods(false) should not include methods that explicitly say that their owner is a different constant in the ancestor chain. The fact that those methods are being returned is a leak of internal implementation details. Users of the method should not need to know how and why protected would have such a side-effect. Moreover, as I stated in my original report private does not have a similar problem, either.

Basically the documentation of instance_methods explicitly states:

If the optional parameter is false, the methods of any ancestors are not included.

and, in this case, that statement is not correct.

#3 - 12/27/2021 04:23 PM - jeremyevans0 (Jeremy Evans)
ufuk (Ufuk Kayserilioglu) wrote in #note-2:

I understand why the difference in behaviour is happening, but I respectfully disagree that this is not a bug.

That's fair. It's not 100% clear that this isn't a bug. Which I why I would like to get input from other committers.

Moreover, as I stated in my original report private does not have a similar problem, either.

This is incorrect, private has exactly the same issue, it's just that instance_methods doesn't include private methods:

module A

def method1() end
end
class B
  include A
  private :method1
  private def method2() end
end

p B.private_instance_methods(false) #=> [:method1, :method2]
p B.instance_method(:method1).owner #=> A

public has the same issue:

module A
  private def method1() end
end
class B
  include A
  public :method1
  def method2() end
end

p B.instance_methods(false) #=> [:method1, :method2]
p B.instance_method(:method1).owner #=> A

Basically the documentation of instance_methods explicitly states:

If the optional parameter is false, the methods of any ancestors are not included.

and, in this case, that statement is not correct.

This depends on your definition of "methods of any ancestors". As I mentioned, public/private/protected create method entries in the current class if the method whose visibility they are affecting is defined in the parent class. I think that makes them methods of the current class, even if the definition occurs in the ancestor.

After doing some more testing, I think there is a bug here, but it's related to instance_method/method returning the wrong information. Evidence of this behavior can be found via the newly introduced methods for checking method visibility:

module A
  def method1() end
end
class B
  include A
  protected :method1
end

p A.instance_method(:method1).public? #=> true
p A.instance_method(:method1).public? #=> true (should be false)
p A.instance_method(:method1).protected? #=> false
p B.instance_method(:method1).protected? #=> false (should be true)
p B.new.method(:method1).public? #=> true (should be false)
p B.new.method(:method1).protected? #=> false (should be true)

I'll see if I can work on a patch to fix this.

#4 - 12/27/2021 07:28 PM - jeremyevans0 (Jeremy Evans)
I've submitted a pull request to fix this issue: [https://github.com/ruby/ruby/pull/5356](https://github.com/ruby/ruby/pull/5356). It makes method/instance_method (and similar methods) no longer skip ZSUPER method entries, so owner will correctly show the class where the method entry is defined, and the visibility methods will return the correct results.

#5 - 01/05/2022 01:23 AM - alanwu (Alan Wu)
I agree this is a confusing part of the API surface. This ticket reminds me of the discussion from [Bug #16106](https://github.com/ruby/ruby/pull/5356).
Consider the following setup:

```ruby
class Parent
  def foo; end
end

class Child < Parent
  protected :foo
end
```

```ruby
p Child.instance_method(:foo).owner #=> Parent
```

Module#instance_methods (plural) only returns public and protected methods, but Module#instance_method (singular) doesn't filter based on visibility. Also, as Jeremy pointed out in [ruby-core:106839](https://github.com/ruby/ruby/pull/5356), Child.protected_instance_methods(false) gives [:foo], but Child.instance_method(:foo).protected? gives false surprisingly.

So currently, Module#protected_instance_methods and similar APIs can provide more information than Module#instance_method. APIs with plural names can observe the effects of using Child.protected(:foo).

An important question is whether Module#protected and other visibility change APIs semantically define new methods when used in a subclass. If not, the particular wording for relevant APIs covers the current behavior:

- Module#protected: ... With arguments, sets the named methods to have protected visibility ...
- UnboundedMethod#owner: Returns the class or module that defines the method.

The wording for Module#instance_method is unclear as to what should happen when there is a visibility difference:

```
Module#instance_method: Returns an +UnboundMethod+ representing the given instance method in mod.
```

I think Jeremy's [PR](https://github.com/ruby/ruby/pull/5356) makes it a rule that using visibility change methods in subclasses semantically define new methods when used in a subclass. If not, the particular wording for relevant APIs covers the current behavior:

- Module#protected: ... With arguments, sets the named methods to have protected visibility ...
- UnboundedMethod#owner: Returns the class or module that defines the method.

I think it's worth mentioning in docs that Module#instance_method[s,] are very different despite having names that imply a relationship.

---

**#6 - 01/07/2022 05:46 PM - Eregon (Benoit Daloze)**

I'm not fully clear on the expected semantics of private/protected/public in a subclass, that would probably be good to improve in the docs of these methods or in general docs.

However since the current semantics seems to clearly create new method entries on the subclass (when the visibility isn't already what's requested) then I think it would make a lot of sense to update the owner of the new method entry to be the subclass, since it's effectively defined on that class (the source_location would remain untouched, of course).

**#7 - 01/14/2022 12:56 AM - jeremyevans0 (Jeremy Evans)**

I've pushed a simpler fix that doesn't change the semantics of method, but still returns the correct visibility for ZSUPER methods, by storing the correct visibility as a member of struct METHOD: https://github.com/ruby/ruby/pull/5356

It may be worth discussing whether to change the semantics of ZSUPER Method objects (such as who the owner should be), but for right now, it's probably best to just fix the bug.

**#8 - 01/14/2022 09:46 PM - jeremyevans (Jeremy Evans)**

- Status changed from Open to Closed

Applied in changeset git|58dc8bf8f15df9a33d191074e8a5d4946a3d59d5.
Fix (Method, UnboundMethod)#{public?, private?, protected?} for ZSUPER methods

Add a visibility member to struct METHOD storing the original method visibility, and use that, instead of taking the visibility from the stored method entry (which may have different visibility for ZSUPER methods).

Consider Method/UnboundMethod objects different if they have different visibilities.

Fixes [Bug #18435]

#9 - 06/03/2022 06:48 AM - matz (Yukihiro Matsumoto)
- Related to Bug #18729: Method#owner and UnboundMethod#owner are incorrect after using Module#public/private added

#10 - 06/03/2022 06:49 AM - matz (Yukihiro Matsumoto)
- Related to Bug #18751: Regression on master for Method#== when comparing public with private method added

#11 - 06/03/2022 06:54 AM - matz (Yukihiro Matsumoto)
- Status changed from Closed to Open

I thought it was OK to accept this behavior, but it caused issues like #18729 and #18751. At the time of the decision, I haven't noticed those corner cases. Although it has already been shipped with 3.1, I proposed to revert this change. I estimate the impact of reverting incompatibility is minimal.

Instead of changing old behavior, the documentation of instance_methods(false) should be updated to explain why method1 is included in the above example (visibility changes or making aliases are considered as definition by instance_methods).

Matz.

#12 - 06/09/2022 10:23 AM - Eregon (Benoit Daloze)

I believe the only way that makes sense here is to remove "ZSUPER methods" altogether. Other Ruby implementations do not have this needless complexity and near-impossible-to-understand semantics. [https://bugs.ruby-lang.org/issues/18751#note-11](https://bugs.ruby-lang.org/issues/18751#note-11) might help to be closer, but IMHO we should remove "ZSUPER methods" altogether.

public/private/protected should shallow-copy a method entry (but still change Method#owner of course), just like alias_method already behaves. This is what TruffleRuby and JRuby do, it's simpler, it is what Ruby users expect (Method is a Ruby object that captures a method entry, at the time it was requested), and it is consistent (.owner is always the module which has that method entry in its method table: #18729).

For instance this should be :p1
:orig1 but currently it's :p2
:orig1 on CRuby.
I claim no Ruby user expects that, because Method should capture a specific method entry, that's why we have bind/call and that's how Method objects are used.

class P
  private
  def m
    :p1
  end

  public
  def orig
    :orig1
  end
end

class C < P
  public :m
  alias_method :alias, :orig
end

class P
  private
  def m
    :p2
  end

  public
  def orig
    :orig2
  end
end
@matz (Yukihiro Matsumoto) OK to remove "ZSUPER methods" and make public/protected/private much simpler by having them shallow copy method entries, just like alias_method already does it?
This will solve a lot of confusion and inconsistency for Method objects of methods defined by public/protected/private.

#13 - 08/10/2022 08:32 PM - jeremyevans0 (Jeremy Evans)
- Status changed from Open to Closed
- Backport changed from 2.6: UNKNOWN, 2.7: UNKNOWN, 3.0: UNKNOWN, 3.1: UNKNOWN to 2.6: DONTNEED, 2.7: DONTNEED, 3.0: DONTNEED, 3.1: REQUIRED

matz (Yukihiro Matsumoto) wrote in #note-11:

I thought it was OK to accept this behavior, but it caused issues like #18729 and #18751. At the time of the decision, I haven't noticed those corner cases. Although it has already been shipped with 3.1, I proposed to revert this change. I estimate the impact of reverting incompatibility is minimal.

I committed the revert at ff42e2359bdfb37e1721a82b4cd95b31f494f3f. My understanding is that @matz (Yukihiro Matsumoto) would like this backported to Ruby 3.1, so marking for backport.

Instead of changing old behavior, the documentation of instance_methods(false) should be updated to explain why method1 is included in the above example (visibility changes or making aliases are considered as definition by instance_methods).

I'll see if I can update the documentation for instance_methods to explain this.

#14 - 08/13/2022 03:27 AM - nagachika (Tomoyuki Chikanaga)

I agree to backport the revert to ruby_3_1, but I think removing the existing methods in stable releases could be a severe damage to the depending applications/libraries.
I'd like to add dummy methods [Method,UnboundMethod]#public?,protected?,private?] with warnings in 3.1.3.
@matz (Yukihiro Matsumoto)@jeremyevans0 (Jeremy Evans) I propose to the dummy implementation of [Method,UnboundMethod]#public? return true constantly, and the others return false. Do you have any objection for adding dummy methods?

#15 - 08/13/2022 03:31 AM - jeremyevans0 (Jeremy Evans)

nagachika (Tomoyuki Chikanaga) wrote in #note-14:

I agree to backport the revert to ruby_3_1, but I think removing the existing methods in stable releases could be a severe damage to the depending applications/libraries.
I'd like to add dummy methods [Method,UnboundMethod]#public?,protected?,private?] with warnings in 3.1.3.
@matz (Yukihiro Matsumoto)@jeremyevans0 (Jeremy Evans) I propose to the dummy implementation of [Method,UnboundMethod]#public? return true constantly, and the others return false. Do you have any objection for adding dummy methods?

I don't have a preference on whether the methods are kept or removed in 3.1. However, I think if we are going to keep the methods in 3.1, we should just have them emit a deprecation warning instead of changing their behavior.

#16 - 08/15/2022 10:55 AM - Eregon (Benoit Daloze)

protected methods like in this issue are part of the method table, so I think it makes no sense to hide them.
What we need to do is to stop hiding ZSUPER methods: https://bugs.ruby-lang.org/issues/18751#note-12

Ruby implementations which do not use "ZSUPER methods" (i.e., both JRuby and TruffleRuby) already expose such methods in instance_methods, as it should be semantically.

I don't think it makes sense to backport this, it's just likely to cause unnecessary incompatibilities and confusion for people updating from 3.1.2 to 3.1.3.

#17 - 08/15/2022 11:11 AM - Eregon (Benoit Daloze)

My understanding of this issue is it's the same as #18729.
The bug is not in instance_methods which is correct.
It's in owner, because of the (undesirable) ZSUPER methods hiding.

#18 - 08/15/2022 01:08 PM - Eregon (Benoit Daloze)
- Status changed from Closed to Open

Reopening this as because of the revert it is still inconsistent and unsolved.
PR to fix this by no longer hiding ZSUPER methods: [https://github.com/ruby/ruby/pull/6242](https://github.com/ruby/ruby/pull/6242)

#19 - 08/18/2022 09:24 AM - Eregon (Benoit Daloze)
@matz (Yukihiro Matsumoto) agreed to remove ZSUPER methods, because the current illusion is incomplete. In practice the semantics are the same (code is unlikely to monkey-patch a method after making it public), except owner/instance_methods is more consistent with "alias/copy" semantics.
I'll make a PR.
EDIT: see [https://github.com/ruby/ruby/pull/6242#issuecomment-1219593598](https://github.com/ruby/ruby/pull/6242#issuecomment-1219593598)

#20 - 08/18/2022 09:27 AM - Eregon (Benoit Daloze)
- Assignee set to Eregon (Benoit Daloze)

#21 - 08/20/2022 11:44 AM - jeremyevans (Jeremy Evans)
- Status changed from Open to Closed

Applied in changeset [git|8212aab81a77a2a9f7c1681b4968171193b48f](https://github.com/ruby/ruby/pull/6242#issuecomment-1219593598).

---

Make Object#method and Module#instance_method not skip ZSUPER methods

Based on [https://github.com/jeremyevans/ruby/commit/c95e7e5329140f640b64979054857613336d967](https://github.com/jeremyevans/ruby/commit/c95e7e5329140f640b64979054857613336d967)

Among other things, this fixes calling visibility methods (public?, protected?, and private?) on them. It also fixes #owner to show the class the zsuper method entry is defined in, instead of the original class it references.
For some backwards compatibility, adjust #parameters and #source_location, to show the parameters and source location of the method originally defined. Also have the parameters and source location still be shown by #inspect.

Clarify documentation of (Method,UnboundMethod)#owner.

Add tests based on the description of [https://bugs.ruby-lang.org/issues/18435](https://bugs.ruby-lang.org/issues/18435) and based on [https://github.com/ruby/ruby/pull/5356#issuecomment-1005298809](https://github.com/ruby/ruby/pull/5356#issuecomment-1005298809)

Fixes [Bug #18435](https://bugs.ruby-lang.org/issues/18435) [Bug #18729](https://bugs.ruby-lang.org/issues/18729)

Co-authored-by: Benoit Daloze eregontp@gmail.com

#22 - 08/20/2022 12:16 PM - Eregon (Benoit Daloze)
- Related to Feature #11689: Add methods allow us to get visibility from Method and UnboundMethod object. added

#23 - 08/20/2022 12:30 PM - Eregon (Benoit Daloze)
@nagachika (Tomoyuki Chikanaga) wrote in #note-14:

I agree to backport the revert to ruby_3_1, but I think removing the existing methods in stable releases could be a severe damage to the depending applications/libraries.
I'd like to add dummy methods (Method,UnboundMethod)#public?,protected?,private? with warnings in 3.1.3.
@matz (Yukihiro Matsumoto)@jeremyevans0 (Jeremy Evans) I propose to the dummy implementation of (Method,UnboundMethod)#public? return true constantly, and the others return false. Do you have any objection for adding dummy methods?

Since [https://github.com/ruby/ruby/pull/6242](https://github.com/ruby/ruby/pull/6242) these methods are no longer problematic (Method == method entry).
Also see [https://bugs.ruby-lang.org/issues/11689#note-27](https://bugs.ruby-lang.org/issues/11689#note-27)

I am not sure what to backport if anything.
Possibilities I see:

- Keep the methods for 3.1.x and accept they are confusing for zsuper methods in 3.1 (fixed in 3.2). Would be good to document that in the methods' rdoc and possibly release notes. Maybe even emit some warning as Jeremy suggests.
- Backport [https://github.com/ruby/ruby/pull/6242](https://github.com/ruby/ruby/pull/6242), since it does fix bugs #18435#18720 (18751 too but I that seems master-only). Unsure if it's a good idea to backport since it's a bigger change (and every bug fix is a potential incompatibility).
- Remove the (public?,protected?,private?) methods. I think this makes sense because we want people to not use them in 3.1 as they are kind of broken, and code which feature checks correctly with method.respond_to?(public) will then work correctly no matter <3.1, 3.1 or 3.2 (assuming the code uses a fallback).

#24 - 08/20/2022 12:38 PM - Eregon (Benoit Daloze)
FWIW, TruffleRuby and JRuby don't have ZSUPER methods, so it's like they always had the fix of [https://github.com/ruby/ruby/pull/6242](https://github.com/ruby/ruby/pull/6242). And so they could add correct {public?, protected?, private?} methods in 3.1 easily.

From that point of view it seems fairly safe to backport that PR (seems extremely unlikely to cause incompatibilities in practice). Might still cause some confusion though if method/instance_method behavior changes between 3.1.2 and 3.1.3 for zsuper methods.