## Ruby master - Feature #6452

Allow extend to override class methods

05/19/2012 01:58 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Assigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>matz (Yukihiro Matsumoto)</td>
</tr>
<tr>
<td>Target version:</td>
<td>3.0</td>
</tr>
</tbody>
</table>

### Description

```ruby
module A
  def self.a
    'a1'
  end
end

module B
  def a
    'a2'
  end

  def b
    'b'
  end
end

A.extend B

assert A.a == 'a2' # this is the change I'm proposing - currently it is 'a1'
assert A.b == 'b'
```

Would this change be possible for 3.0?

### Related issues:

- Related to Ruby master - Feature #1102: Prepend Module
  - Closed 02/04/2009

### History

**#1 - 05/19/2012 08:31 AM - mame (Yusuke Endoh)**

- Status changed from Open to Assigned

Hello,

rosenfeld (Rodrigo Rosenfeld Rosas) wrote:

    Would this change be possible for 3.0?

Why don't you add a new method instead of changing an existing one?

What you want is allowed by a feature planned for Ruby 2.0 called Module#prepend:

```ruby
class << A
  prepend B
end
```

See #1102 and the thread starting with [ruby-core:25208].
If you want to do so without opening singleton class, please propose a new method name.

--

Yusuke Endoh mame@tsg.ne.jp

**#2 - 05/19/2012 08:37 AM - rosenfeld (Rodrigo Rosenfeld Rosas)**
I'm not sure if prepend would have the same effect. I was expecting to call "super" in B and it would call A.a.

Does it make sense?

I don't really care if this would have another method name.

Would you mind to explain why the current behavior is useful instead of doing what I'm proposing?

I really don't understand why it was implemented this way...

#3 - 05/21/2012 08:11 PM - mame (Yusuke Endoh)

Hello,

2012/5/19 rosenfeld (Rodrigo Rosenfeld Rosas) rr.rosas@gmail.com:

Would you mind to explain why the current behavior is useful instead of doing what I'm proposing?

Just example:

```ruby
module GenericConnection
def connect(domain)
  TCPSocket.open(domain)
end
end

class Google
  extend GenericConnection
def self.connect
    super(*)
  end
end
class Twitter
  extend GenericConnection
def self.connect
    super(*)
  end
end
```

I really don't understand why it was implemented this way...

You may think so because you declared modules and methods in weird order.
You should declare a smaller, more generic, or more independent module first, and then a bigger one by using already-defined ones.
Do you still expect 'a2' for the following code?

```ruby
module B
def a
  'a2'
end
end

module A
  extend B
def self.a
    'a1'
  end
end

p A.a #=> 'a1'
```

---
Yusuke Endoh mame@tsg.ne.jp

#4 - 05/21/2012 09:08 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Yeah, sorry I should have thought about this before.

You're right, for normal usage the current behavior is more useful indeed.

I was willing to patch a class from a third-party library, that is why I wanted my module method to take precedence.
Would you consider to add a new method for such case? And an equivalent one for include as well?

Should I open a new ticket and close this one?

#5 - 06/26/2014 02:19 PM - rosenfeld (Rodrigo Rosenfeld Rosas)
Would Module.preextend be a good name for that? It would indicate it's similar to prepend but applied to extend...

#6 - 07/26/2014 04:33 PM - Anonymous
Rodrigo Rosenfeld Rosas wrote:

Would Module.preextend be a good name for that? It would indicate it's similar to prepend but applied to extend...

Does this do what you want to do?

class Object
def preextend m
  singleton_class.prepend m
end
end

If yes, then there is no urgent need to ask for a new feature from the core team.

#7 - 07/28/2014 12:08 PM - rosenfeld (Rodrigo Rosenfeld Rosas)
Hi Boris,

there's indeed no urgency in asking for such a feature but I don't think it's a required reason for a feature request, is it?

But thanks for the suggested implementation, really useful.

#8 - 10/22/2014 11:00 PM - Anonymous
There is the problem of feature creep, like classes becoming harder and harder to learn... That's what I meant by no urgent need. I might be mistaken.