## Ruby master - Feature #5005

### Provide convenient access to original methods

07/09/2011 08:05 PM - lazaridis.com (Lazaridis Ilias)

<table>
<thead>
<tr>
<th>Status</th>
<th>Rejected</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>2.6</td>
</tr>
</tbody>
</table>

### Description

The language allows a class to be "reopened", thus its behavior can be redefined:

```ruby
class String
  def any_method
    # custom code
  end
end
```

The original method can be called, using this construct:

```ruby
class String
  alias_method :original_any_method, :any_method
  def any_method(*args)
    # custom code
    original_any_method(*args)
    # custom code
  end
end
```

In order to make this more convenient, the following construct could be provided:

```ruby
class String
  def any_method(*args)
    # custom code
    original # call the original String#any_method, passing *args (similar to "super")
    # custom code
  end
end
```

"original" would behave similar to "super"

The term "original" can be replaced by any term which describes this concept better.

### History

#### #1 - 07/09/2011 09:39 PM - nobu (Nobuyoshi Nakada)
- Target version set to 3.0

#### #2 - 07/09/2011 09:55 PM - antares (Michael Klishin)

This approach with aliasing seems to be pretty popular because many people do not know that it is possible to define a module with `#any_method` and include it into a class (in your example, `String`) and call `super` in that module method. This feature will open a whole new wave of reinvented OOP wheels. Adding new language features just to work around controversial OOP practices does not sound like a good idea to me personally.

#### #3 - 07/09/2011 10:43 PM - neleai (Ondrej Bilka)

Michael Klishin wrote:

> This approach with aliasing seems to be pretty popular because many people do not know that it is possible to define a module with `#any_method` and include it into a class (in your example, `String`) and call `super` in that module method. This feature will open a whole new wave of reinvented OOP wheels. Adding new language features just to work around controversial OOP practices does not sound like a good idea to me personally.

Could you elaborate what did you mean?
module B
  def foo
    super+"b"
  end
end
class A
  def foo
    "a"
  end
  include B
end
puts A.new.foo
returns a which is not what was wanted

#4 - 07/09/2011 11:13 PM - antares (Michael Klishin)
Ondrej,

When module is included into a class, Ruby adds a new anonymous (in a sense that it will be skipped by Class#superclass calls) class into the
inheritance chain: https://gist.github.com/1073599. So it is appending, not prepending. This is useless for core classes like String, however, most
cases of blatant monkey-patching that I see look more like this: https://gist.github.com/1073605. In that case, aliased methods is a reinvented wheel.

#5 - 07/09/2011 11:27 PM - matz (Yukihiro Matsumoto)
The idea to address the issue is the "prepend" feature planned for Ruby 2.0.

class Original
def any_method
  ...
end
def any_method
  #custom code
  super # call the original any_method, passing *args
  #custom code
end
def any_method
  ...
end

reopen class
class Original
  prepend Modifier
end

The prepend method pushes a module in front of a class so that super calls the original method.

matz.

#6 - 07/10/2011 01:53 AM - neleai (Ondrej Bilka)
One of answers is use AOP like aquarium(But I didn't tried it so I dont
know if it isn't overkill).
Currently best solution is use unbound method which is not much intuitive
From perspective that easiest solution should be too wrong we could
introduce wrap_method. It avoids problems when alias_method break
alias_method of previous guy.
class Module
  def wrap_method(foo)
    i=instance_method(foo)
    define_method(foo){|*a|
      yield(i.bind(self),*a)
    }
  end
end
class A
  def foo
    1
  end
  wrap_method{foo}{|prev|
    prev()+1
  }

On Sat, Jul 09, 2011 at 11:27:46PM +0900, Yukihiro Matsumoto wrote:

Issue #5005 has been updated by Yukihiro Matsumoto.

The idea to address the issue is the "prepend" feature planned for Ruby 2.0.

```ruby
class Original
  def any_method
    ...
  end
end

module Modifier
  def any_method
    #custom code
    super # call the original any_method, passing *args
    #custom code
  end
end

reopen class

class Original
  prepend Modifier
end
```

The prepend method pushes a module in front of a class so that super calls the original method.

matz.

Feature #5005: Provide convenient access to original methods
[http://redmine.ruby-lang.org/issues/5005](http://redmine.ruby-lang.org/issues/5005)

Author: Lazaridis Ilias
Status: Open
Priority: Normal
Assignee: Category: core
Target version: 2.0

The language allows a class to be "reopened", thus it's behaviour can be redefined:

```ruby
class String
  def any_method
    #custom code
  end
end
```

the original method can be called, using this construct:

```ruby
class String
  alias_method :original_any_method, :any_method
  def any_method(*args)
    original_any_method(*args)
    #custom code
  end
end
```

In order to make this more convenient, the following construct could be provided:

```ruby
class String
  def any_method(*args)
    original # call the original String#any_method, passing *args (similar to "super")
    #custom code
  end
end
```
"original" would behave similar to "super"

The term "original" can be replaced by any term which describes this concept better.

http://redmine.ruby-lang.org

Your EMAIL is now being delivered by the USPS.

#7 - 07/13/2011 12:39 AM - lazaridis.com (Lazaridis Ilias)
=begin
I've notice a related issue, #3688, which suggests the introduction of "redef". This, in combination of "original", would be an extension more native to the language - avoiding this way to (ab)use the general construct "super(class-method)", which has a very specific meaning in standard OO.

"redef" would allow to keep track of the redefined method.

Another way would be, that the usual "def" detects that a redefinition happens, thus is keeps automatically a copy of the original method, which "original" accesses then.

This way, "original" would take care about multiple redefinitions, calling always the right one, without any effort for the user.
=end

#8 - 07/13/2011 08:32 AM - lazaridis.com (Lazaridis Ilias)
Lazaridis Ilias wrote:

The languag allows a class to be "reopened", thus it's behaviour can be redefined:
[...]
"original" would behave similar to "super"

The term "original" can be replaced by any term which describes this concept better.

Just realized that "previous" would be a more concise term.

- super : the super(class) method, standard OO terminology
- original : the original, the first, the basic version of the method (e.g. C-level one)
- previous : the previous version of this method, the overridden one (either the original one, or an already redefined)

#9 - 07/13/2011 10:04 AM - duerst (Martin Dürst)

Just some general thoughts on making metaprogramming easier:

Using metaprogramming in Ruby is on average somewhat more lengthy/clumsy than straightforward/plain (i.e. non-meta) programming. I have been thinking about why that may be. It could just be that most of the attention has been on plain programming, and metaprogramming hasn't been polished as much yet as plain programming. But over time, I got the impression that to some extent, leaving metaprogramming to be somewhat more clumsy may have been intentional.

Metaprogramming is a very powerful tool, and therefore programmers should think hard about how to use it well. If it gets too easy and straightforward, then it's going to be overused. I agree that method patching as in this feature request and in #3688 is very frequent. But it is often a quick patch to an underlying problem that might benefit from further thought. If we make it easier with 'redef' or 'original' or some other keyword or syntax, then it will easily become even more over/abused. This may also apply to other cases of metaprogramming.

#10 - 10/18/2011 09:16 AM - naruse (Yui NARUSE)
- Project changed from Ruby master to 14
- Category deleted (core)
- Target version deleted (3.0)

#11 - 10/23/2011 05:21 PM - naruse (Yui NARUSE)
- Project changed from 14 to Ruby master

#12 - 03/25/2012 04:44 PM - mame (Yusuke Endoh)
- Status changed from Open to Assigned
- Assignee set to matz (Yukihiro Matsumoto)
#13 - 11/20/2012 09:45 PM - mame (Yusuke Endoh)
- Target version set to 2.6

#14 - 12/12/2012 10:19 PM - nobu (Nobuyoshi Nakada)
- Status changed from Assigned to Rejected

Use Module#prepend and super.