Allow method transplanting

Is there a technical reason to not allow re-binding a method from one module to any other module?

module M
def foo; "foo"; end
end

module N; end
N.send(:define_method, :foo, M.instance_method(:foo)) #=> should not raise

It's like monkey-patching. Powerful, dangerous, but also really useful. It could allow different variations of method_wrap or alias_method_chain that are not possible right now.

Associated revisions

Revision e6f1e3f4 - 06/25/2012 07:57 AM - nobu (Nobuyoshi Nakada)
method transplanting

  • proc.c (rb_mod_define_method): allow method transplanting from a module to either class or module. [ruby-core:34267][Feature #4254]
  git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@36214 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 36214 - 06/25/2012 07:57 AM - nobu (Nobuyoshi Nakada)
method transplanting

  • proc.c (rb_mod_define_method): allow method transplanting from a module to either class or module. [ruby-core:34267][Feature #4254]

Revision 36214 - 06/25/2012 07:57 AM - nobu (Nobuyoshi Nakada)
method transplanting

  • proc.c (rb_mod_define_method): allow method transplanting from a module to either class or module. [ruby-core:34267][Feature #4254]

Revision 36214 - 06/25/2012 07:57 AM - nobu (Nobuyoshi Nakada)
method transplanting

  • proc.c (rb_mod_define_method): allow method transplanting from a module to either class or module. [ruby-core:34267][Feature #4254]

Revision 36214 - 06/25/2012 07:57 AM - nobu (Nobuyoshi Nakada)
method transplanting

  • proc.c (rb_mod_define_method): allow method transplanting from a module to either class or module. [ruby-core:34267][Feature #4254]

Revision 36214 - 06/25/2012 07:57 AM - nobu (Nobuyoshi Nakada)
method transplanting

  • proc.c (rb_mod_define_method): allow method transplanting from a module to either class or module. [ruby-core:34267][Feature #4254]
Revision 7c3d0ce3 - 06/25/2012 08:06 AM - nobu (Nobuyoshi Nakada)
method transplanting

• proc.c (rb_mod_define_method): allow method transplanting from a module to either class or module. [ruby-core:34267][Feature #4254]
git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@36215 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 36215 - 06/25/2012 08:06 AM - nobu (Nobuyoshi Nakada)
method transplanting

• proc.c (rb_mod_define_method): allow method transplanting from a module to either class or module. [ruby-core:34267][Feature #4254]

Revision 36215 - 06/25/2012 08:06 AM - nobu (Nobuyoshi Nakada)
method transplanting

• proc.c (rb_mod_define_method): allow method transplanting from a module to either class or module. [ruby-core:34267][Feature #4254]

Revision 36215 - 06/25/2012 08:06 AM - nobu (Nobuyoshi Nakada)
method transplanting

• proc.c (rb_mod_define_method): allow method transplanting from a module to either class or module. [ruby-core:34267][Feature #4254]

Revision 36215 - 06/25/2012 08:06 AM - nobu (Nobuyoshi Nakada)
method transplanting

• proc.c (rb_mod_define_method): allow method transplanting from a module to either class or module. [ruby-core:34267][Feature #4254]

Revision 36215 - 06/25/2012 08:06 AM - nobu (Nobuyoshi Nakada)
method transplanting

• proc.c (rb_mod_define_method): allow method transplanting from a module to either class or module. [ruby-core:34267][Feature #4254]

Revision 06675418 - 12/13/2012 04:06 AM - nobu (Nobuyoshi Nakada)
method transplanting

• proc.c (umethod_bind): allow another form of method transplanting from a module via UnboundMethod. [ruby-core:34267][Feature #4254]
git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@38354 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 38354 - 12/13/2012 04:06 AM - nobu (Nobuyoshi Nakada)
method transplanting

• proc.c (umethod_bind): allow another form of method transplanting from a module via UnboundMethod. [ruby-core:34267][Feature #4254]

Revision 38354 - 12/13/2012 04:06 AM - nobu (Nobuyoshi Nakada)
method transplanting

• proc.c (umethod_bind): allow another form of method transplanting from a module via UnboundMethod. [ruby-core:34267][Feature #4254]

Revision 38354 - 12/13/2012 04:06 AM - nobu (Nobuyoshi Nakada)
method transplanting

• proc.c (umethod_bind): allow another form of method transplanting from a module via UnboundMethod. [ruby-core:34267][Feature #4254]
method transplanting

- proc.c (umethod_bind): allow another form of method transplanting from a module via UnboundMethod. [ruby-core:34267][Feature #4254]

Revision 38354 - 12/13/2012 04:06 AM - nobu (Nobuyoshi Nakada)
method transplanting

- proc.c (umethod_bind): allow another form of method transplanting from a module via UnboundMethod. [ruby-core:34267][Feature #4254]

History

#1 - 01/09/2011 10:01 PM - zenspider (Ryan Davis)
=end

On Jan 9, 2011, at 04:58, Jonas Pfenniger wrote:

Feature #4254: Allow method transplanting
http://redmine.ruby-lang.org/issues/show/4254

Author: Jonas Pfenniger
Status: Open, Priority: Normal
Category: core, Target version: 1.9.3

Is there a technical reason to not allow re-binding a method from one module to any other module?

I would like to see this approved. I think it could open up a lot of interesting programming styles we can't even currently consider. This would allow for instance-oriented programming among other things.

=end

#2 - 01/09/2011 10:08 PM - rkh (Konstantin Haase)
=end

Wouldn't that be unnecessary if matz' trait proposal would be implemented?

On Jan 9, 2011, at 13:58, Jonas Pfenniger wrote:

Feature #4254: Allow method transplanting
http://redmine.ruby-lang.org/issues/show/4254

Author: Jonas Pfenniger
Status: Open, Priority: Normal
Category: core, Target version: 1.9.3

Is there a technical reason to not allow re-binding a method from one module to any other module?

module M
def foo; "foo"; end
end

module N; end
N.send(:define_method, :foo, M.instance_method(:foo)) #=> should not raise

It's like monkey-patching. Powerful, dangerous, but also really useful. It could allow different variations of method_wrap or alias_method_chain that are not possible right now.

http://redmine.ruby-lang.org

=end

#3 - 01/09/2011 10:17 PM - zimbatm (zimba tm)
=end

2011/1/9 Haase, Konstantin Konstantin.Haase@student.hpi.uni-potsdam.de:

Wouldn't that be unnecessary if matz' trait proposal would be implemented?

I agree it's not the cleanest, but it's not like I would be adding any
feature, rubyspec doesn't give me any addition error, expect the one that explicitly tests for the exception.

What I'm more worried about, is if other ruby implementations can come up with a working solution. For example if instance variables are transformed into an array index at compile-time (rubinius?), then the index might not be the same in the other object.

=end

#4 - 01/10/2011 04:53 AM - Isegal (Loren Segal)
=begin
On 1/9/2011 8:17 AM, Jonas Pfenniger (zimbatm) wrote:

I agree it's not the cleanest, but it's not like I would be adding any feature, rubyspec doesn't give me any addition error, expect the one that explicitly tests for the exception.

FWIW I've implemented something similar a while back as a gem called "force_bind". You call .force_bind on an UnboundMethod instead of bind.

https://rubygems.org/gems/force_bind

Although it would be great to see #bind (and define_method) support this out of the box, adding this behaviour through a gem isn't so bad.

=end

#5 - 01/10/2011 04:57 AM - Isegal (Loren Segal)
=begin
On 1/9/2011 2:53 PM, Loren Segal wrote:

FWIW I've implemented something similar a while back as a gem called "force_bind". You call .force_bind on an UnboundMethod instead of bind.

By the way, if this patch is accepted, we should also patch UnboundMethod#bind to allow binding to different module/classes-- as force_bind does-- to keep things consistent.

=end

#6 - 01/10/2011 05:33 AM - headius (Charles Nutter)
=begin
On Sun, Jan 9, 2011 at 1:53 PM, Loren Segal lsegal@soen.ca wrote:

FWIW I've implemented something similar a while back as a gem called "force_bind". You call .force_bind on an UnboundMethod instead of bind.

https://rubygems.org/gems/force_bind

Although it would be great to see #bind (and define_method) support this out of the box, adding this behaviour through a gem isn't so bad.

JRuby has shipped this feature since 2006 or so, called JRubyExtensions.steal_method(target, source, :name). We originally did it because we needed only some methods from ActiveRecord on our hacked adapters around JDBC. I'm unsure if it's used anymore.

• Charlie

=end

#7 - 01/10/2011 08:56 AM - Isegal (Loren Segal)
=begin
On 2011-01-09, at 2:53 PM, Loren Segal wrote:
Although it would be great to see #bind (and define_method) support this out of the box, adding this behaviour through a gem isn't so bad.

I just gave force_bind a shot after having not used it in a long while, and I found out that it's no longer working in 1.9.2+. I originally wrote it when 1.9.1 was latest. It seems that the method structures have changed quite a bit since then, and that means it's possible they might change more in the future.

I therefore go back on what I said above: supporting this feature as a gem doesn't really seem to be a maintainable solution. Official support would be much better.

#8 - 01/10/2011 09:06 AM - matz (Yukihiro Matsumoto)

Hi,

Class to class method transplanting could cause serious problem, but it might be able to relax module to module transplanting.

matz.

In message "Re: [ruby-core:34267] [Ruby 1.9-Feature#4254][Open] Allow method transplanting" on Sun, 9 Jan 2011 21:58:16 +0900, Jonas Pfenniger redmine@ruby-lang.org writes:

|Author: Jonas Pfenniger |
|Status: Open, Priority: Normal |
|Category: core, Target version: 1.9.3 |

|Is there a technical reason to not allow re-binding a method from one module to any other module ? |
|module M |
|  def foo; "foo"; end |
|end |
|module N; end |
|N.send(:define_method, :foo, M.instance_method(:foo)) #=> should not raise |

|It's like monkey-patching. Powerful, dangerous, but also really useful. It could allow different variations of method_wrap or alias_method_chain that are not possible right now. |

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://redmine.ruby-lang.org">http://redmine.ruby-lang.org</a></td>
</tr>
<tr>
<td>[2 0001-method-transplanting-Allow-to-set-an-UnboundMethod-t.patch ]</td>
</tr>
<tr>
<td>[From 0336ec334f7eb66d62cf05bd7a29d748780d6044e Mon Sep 17 00:00:00 2001 ]</td>
</tr>
<tr>
<td>[Date: Sun, 9 Jan 2011 00:13:57 +0000 ]</td>
</tr>
<tr>
<td>[Subject: [PATCH] method transplanting: Allow to set an UnboundMethod to any module. ]</td>
</tr>
<tr>
<td>It's time to grow up, remove the security nets.</td>
</tr>
</tbody>
</table>

---
| proc.c | 11 --------- |
| 1 files changed, 0 insertions(+), 11 deletions(-) |
|diff --git a/proc.c b/proc.c |
|index 7df2e8...652b3e1 100644 |
|--- a/proc.c |
|+++ b/proc.c |
|@@ -1293,17 +1293,6 @@ rb_mod_define_method(int argc, VALUE *argv, VALUE mod) |
| if (rb_obj_is_method(body)) { |
|   struct METHOD *method = (struct METHOD *)DATA_PTR(body); |
|   VALUE rclass = method->rclass; |
|   if (rclass != mod && !RTEST(rb_class_inherited_p(mod, rclass))) { |
|     if (FL_TEST(rclass, FL_SINGLETON)) { |
|       rb_raise(rb_eType_error, |
|         "can't bind singleton method to a different class"); |
|       return; |
|     } else { |
|       rb_raise(rb_eType_error, |
|         "bind argument must be a subclass of %s", |
|         rb_class2name(rclass)); |
|     } |
|   } |
|   rb_method_entry_set(mod, id, &method->me, noex);
Class to class method transplanting could cause serious problem, but it might be able to relax module to module transplanting.

matz.

Hi matz,

Are you thinking of a method is defined in C that uses Data_Wrap_Struct ?

matz.

The same applies in JRuby for a slightly different reason: methods on core classes are usually defined as instance methods on Java-based classes like RubyArray or RubyString. Transplanting them to another class would, for example, attempt to call a RubyArray method against a RubyString instance, and throw a Java ClassCastException. Essentially...it would crash :)

This is also the reason JRuby does not change the class of the IO object when doing reopen; the concrete Java type is set at construction time, and can't be changed (e.g. a file is RubyFile, and always will be).

- Charlie
Hi,

: At Mon, 10 Jan 2011 10:14:34 +0900,
: Yukihiro Matsumoto wrote in [ruby-core:34283]:
No, I am thinking of methods defined for modules, that have no
assumption of the type of receivers. Ruby methods in C does not check
the type of the receiver, so that if you move a method from String
class to, say, Array, the interpreter will crash when you call the
transplanted method.

Just between modules?

  * proc.c (rb_mod_define_method): allow method transplanting between modules. [ruby-core:34267]

---

diff --git i/proc.c w/proc.c
index 9ecf626...6affba6 100644
--- i/proc.c
+++ w/proc.c
@@ -1303,7 +1303,8 @@ rb_mod_define_method(int argc, VALUE *argv, VALUE mod)
 if (rb_obj_is_method(body)) {
   struct METHOD *method = (struct METHOD *)DATA_PTR(body);
   VALUE rclass = method->rclass;
- if (rclass != mod && !RTEST(rb_class_inherited_p(mod, rclass))) {
+ if (rclass != mod && !(RB_TYPE_P(rclass, T_MODULE) && RB_TYPE_P(mod, T_MODULE)) &&
   !RTEST(rb_class_inherited_p(mod, rclass))) {     if (FL_TEST(rclass, FL_SINGLETON)) {     rb_raise(rb_eTypeError,          "can't bind singleton
method to a different class");
   }

--- Nobu Nakada
=end
You've argued that being able to define != contradictory with == (a serious problem in my mind) is ok because we trust rubyists to do the right thing. How is this any different? If the method transplanted is pure ruby, why not allow it?

#14 - 01/10/2011 04:20 PM - headius (Charles Nutter)

=begin
On Mon, Jan 10, 2011 at 1:15 AM, Ryan Davis ryand-ruby@zenspider.com wrote:

Matz,

You've argued that being able to define != contradictory with == (a serious problem in my mind) is ok because we trust rubyists to do the right thing. How is this any different? If the method transplanted is pure ruby, why not allow it?

Pure Ruby methods would be fine in JRuby as well. They're just bags of code at that point, and can only see the object's structure/representation as much as any other Ruby code. They might do things that cause Ruby exceptions to raise, but probably not crash.

- Charlie
=end

#15 - 01/10/2011 08:50 PM - Eregon (Benoit Daloze)

=begin
Hi,
On 10 January 2011 02:14, Yukihiro Matsumoto matz@ruby-lang.org wrote:

Hi,
Class to class method transplanting could cause serious problem, but it might be able to relax module to module transplanting.

matz.

As Ryan mentioned, I think it is a danger a developer using #bind can live with.

The best would be able to check if it is actually impossible (so detect if a method is implemented in C), and raise in that case. I do not know if that is possible and reliable. (eg: Method#source_location.nil?)

But I think it is best allowed for all than none class methods.
=end

#16 - 01/11/2011 06:17 AM - lsegal (Loren Segal)

=begin
On 1/10/2011 2:15 AM, Ryan Davis wrote:

If the method transplanted is pure ruby, why not allow it?

I should follow up on this because I was thinking the same thing. Is there a way to detect that a method is "native"? If so, we should only block definitions/rebinds on methods that are native code. FWIW force_bind has an example or two that rebinds instance methods of classes (pure-ruby) to other classes and they work quite well, even when interacting with class things like @ivars.
=end

#17 - 01/11/2011 07:30 AM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:34291] Re: [Ruby 1.9-Feature#4254][Open] Allow method transplanting" on Mon, 10 Jan 2011 16:15:40 +0900, Ryan Davis ryand-ruby@zenspider.com writes:

|You've argued that being able to define != contradictory with == (a serious problem in my mind) is ok because we trust rubyists to do the right thing.
How is this any different? If the method transplanted is pure ruby, why not allow it?

Although I trust the programmers, we have another rule for the interpreter, that Ruby should not cause segmentation fault from any Ruby program.

matz.

#18 - 01/11/2011 07:33 AM - matz (Yukihiro Matsumoto)
=begin
Hi,

In message "Re: [ruby-core:34303] Re: [Ruby 1.9-Feature#4254][Open] Allow method transplanting" on Mon, 10 Jan 2011 20:50:32 +0900, Benoit Daloze eregontp@gmail.com writes:

|As Ryan mentioned, I think it is a danger a developer using #bind can live with. |
|The best would be able to check if it is actually impossible (so detect if a method is implemented in C), and raise in that case. |
|I do not know if that is possible and reliable. (eg: Method#source_location.nil?) |
|But I think it is best allowed for all than none class methods. |

I disagree. The implementation language of a method may change between versions. Relying on that may make programs fragile. Trusting programmers should not be excuse for making Ruby a dangerous place for the programmers.

matz.
=end

#19 - 01/11/2011 10:25 AM - zenspider (Ryan Davis)
=begin
On Jan 10, 2011, at 14:30 , Yukihiro Matsumoto wrote:

    Hi,

    In message "Re: [ruby-core:34291] Re: [Ruby 1.9-Feature#4254][Open] Allow method transplanting" on Mon, 10 Jan 2011 16:15:40 +0900, Ryan Davis ryand-ruby@zenspider.com writes:

    |You've argued that being able to define != contradictory with == (a serious problem in my mind) is ok because we trust rubyists to do the right thing. How is this any different? If the method transplanted is pure ruby, why not allow it? |

    Although I trust the programmers, we have another rule for the interpreter, that Ruby should not cause segmentation fault from any Ruby program.

    How can transplanting a pure-ruby method cause a segfault?

    I see nothing in the original example that could cause that:

        module M
        def foo; "foo"; end
        end

        module N; end
        N.send(:define_method, :foo, M.instance_method(:foo)) #=> should not raise

    When we were trying it we were using bind on instances, not define_method on classes/modules. That seems even safer.

=end

#20 - 01/11/2011 11:29 AM - matz (Yukihiro Matsumoto)
=begin
Hi,
In message "Re: [ruby-core:34319] Re: [Ruby 1.9-Feature#4254][Open] Allow method transplanting" on Tue, 11 Jan 2011 10:24:55 +0900, Ryan Davis ryand-ruby@zenspider.com writes:

> Although I trust the programmers, we have another rule for the
> interpreter, that Ruby should not cause segmentation fault from any
> Ruby program.
>
| How can transplanting a pure-ruby method cause a segfault?
>
See [ruby-core:34319]. From above rule we cannot allow all method transplanting from classes. Since we don't want program fragility between versions, I don't want to allow method transplanting from classes at all.

Remember the beginning of 1.8.7; rather small incompatibility hindered the reputation of the release so bad. I don't want to see the situation like that again.

matz.

=end

#21 - 01/13/2011 01:48 AM - zimbatm (zimba tm)

=begin

2011/1/11 Yukihiro Matsumoto matz@ruby-lang.org:

See [ruby-core:34319]. From above rule we cannot allow all method transplanting from classes. Since we don't want program fragility between versions, I don't want to allow method transplanting from classes at all.

Remember the beginning of 1.8.7; rather small incompatibility hindered the reputation of the release so bad. I don't want to see the situation like that again.

I agree with matz. If it's something we want for the next releases, then it should not break anything. Having transplantable Module methods already unlocks lots of scenarios, where a module can act as a bag of methods. For example it allows using methods with define_method, which allows having regular argument-passing, in contract with blocks, which capture the current context and don't allow a block as argument. This in turn allows to find new patterns that can replace alias_method_chain. If we have that, then it's already wonderful.

For the ones who are not satisfied with the current scope of the ticket, I propose to open a new 2.0+ issue for the research that is needed to come up for a universal solution.

=end

#22 - 01/13/2011 08:57 AM - nobu (Nobuyoshi Nakada)

=begin

Hi,

At Thu, 13 Jan 2011 01:48:33 +0900,
Jonas Pfenniger (zimbatm) wrote in [ruby-core:34444]:

I agree with matz. If it's something we want for the next releases, then it should not break anything. Having transplantable Module methods already unlocks lots of scenarios, where a module can act as a bag of methods. For example it allows using methods with define_method, which allows having regular argument-passing, in contract with blocks, which capture the current context and don't allow a block as argument. This in turn allows to find new patterns that can replace alias_method_chain. If we have that, then it's already wonderful.

1.9 already allows a block having block-passing.

class A
def foo
    yield "A"
end

class B < A
    a_foo = instance_method(:foo)
define_method(:foo) do |&block|
a_foo.bind(self).call {|s| block.call(s+"B")}
end

B.new.foo {x|x} #=> "AB"

--
Nobu Nakada
=end

#23 - 06/28/2011 06:20 AM - nahi (Hiroshi Nakamura)
- Target version changed from 1.9.3 to 2.0.0

#24 - 03/18/2012 06:56 PM - nahi (Hiroshi Nakamura)
- Status changed from Open to Rejected
- Assignee set to matz (Yukihiro Matsumoto)

#25 - 06/01/2012 02:53 PM - nobu (Nobuyoshi Nakada)
- Description updated
- Status changed from Rejected to Assigned

=end

What status is this proposal?
I don't think the original proposal has been rejected, but only other additional extensions.
Possible choice would be:

(1) keep current behavior, all method transplanting is prohibited.
(2) accept the original proposal only, transplanting is allowed only between modules but disallowed between class and module/class.
(3) still look for "relaxed" condition, actually same as (1) but leave this ticket opened.
(4) others.
I think there is no probable "relaxed" versions, though.

#26 - 06/25/2012 03:38 PM - matz (Yukihiro Matsumoto)
I'd like to allow method transplanting from a module, not a class, to either class or module. Any objection?
Matz.

#27 - 06/25/2012 03:51 PM - duerst (Martin Dürst)
matz (Yukihiro Matsumoto) wrote:

I'd like to allow method transplanting from a module, not a class, to either class or module. Any objection?

I think this is a step in the right direction. Both JavaScript and Python allow functions (including what are essentially methods) to be assigned arbitrarily. There are often better ways to achieve the same thing in Ruby, but sometimes, there's not much of an alternative.

[Some of my students were working on a project to automatically convert Python programs to Ruby, and this is a very hard wall we bumped into.]
method transplanting

- proc.c (rb_mod_define_method): allow method transplanting from a module to either class or module. [ruby-core:34267][Feature #4254]

#29 - 06/26/2012 12:08 AM - trans (Thomas Sawyer)
What prevents methods from being transplanted from a class?

#30 - 06/26/2012 12:47 AM - rosenfeld (Rodrigo Rosenfeld Rosas)
I guess it wouldn't make much sense and would lead to other developer's confusion as the method may rely on some internal state of an instance of that class...

#31 - 06/26/2012 02:17 AM - trans (Thomas Sawyer)
That makes sense. But I am not sure it matters a great deal. Modules too can have interdependent methods and reference instance variables. I get the concept, but ultimately I'd just prefer to have full flexibility and be done with it, rather than having to fuss with the limitations. This is a meta-programming feature and like all such techniques it requires care by the developer.

However, I am also coming at this with very particular use case in mind. I would make little girl squeally noises to be able to define a method that could include a module that would also include it's class methods. Being able to transplant class methods would make that possible (albeit not the most elegant approach). If there were another means of doing that, then I wouldn't care as much about transplanting class methods.

#32 - 05/25/2013 03:06 AM - saturnflyer (Jim Gay)
=begin
I am interested in this feature although with a different use.

It appears that 2.0.0-p0 allowed methods to be unbound from a module and bound to any object.

I've forked rubyspec and created a failing spec ((URL:https://github.com/saturnflyer/rubyspec/compare/a4b320efc34c...668f4be5999852a))

In 2.0.0-p0 this code works: (((SomeModule.instance_method(:the_method).bind(Object.new)))) but this seems to be broken in 2.1-dev (and is not present in previous versions)

This project uses this feature to temporarily add behavior to an object ((URL:https://github.com/saturnflyer/casting)) and Travis CI is running the tests at ((URL:https://travis-ci.org/saturnflyer/casting/builds/7432955))

The specs fail on ruby-head ((URL:https://travis-ci.org/saturnflyer/casting/jobs/7432959)) but pass for 2.0 ((URL:https://travis-ci.org/saturnflyer/casting/jobs/7432957))

This feature is useful in allowing an object to run methods inside a block:

object.hello_world #=> NoMethodError
Casting.delegating(object => GreetingModule) do
object.hello_world #=> "Hello world!"
end
object.hello_world #=> NoMethodError

The above example uses method_missing to unbind methods from the given module then bind them to self and call them.

If there is a better place to post this, or if I should open a new ticket, please correct me.
=end

#33 - 05/26/2013 12:03 AM - nobu (Nobuyoshi Nakada)
=begin
Seems fine.

$ ruby -v -e 'module M; def foo; :foo;end; end; p m = M.instance_method(:foo).bind(Object.new); p m.call'

=end
ruby 2.1.0dev (2013-05-25 trunk 40923) [universal.x86_64-darwin11]
#
:foo

Please open new ticket if needed.
=end

#34 - 05/26/2013 10:21 PM - nobu (Nobuyoshi Nakada)

=begin
:saturnflyer (Jim Gay) wrote:
The specs fail on ruby-head ((URL:https://travis-ci.org/saturnflyer/casting/jobs/7432959)) but pass for 2.0 ((URL:https://travis-ci.org/saturnflyer/casting/jobs/7432957))

From your "ruby-head" log:

$ ruby --version
ruby 2.0.0dev (2012-12-12) [x86_64-linux]

It's too old.
=end

#35 - 05/31/2013 01:53 PM - saturnflyer (Jim Gay)

thanks nobu! I blindly trusted the travis-ci build. good to know this is still valid behavior.

Files

| 0001-method-transplanting-Allow-to-set-an-UnboundMethod-t.patch | 1.07 KB | 01/09/2011 | zimbatm (zimba tm) |