Ruby master - Feature #5673

undef_method probably doesn't need to raise an error

11/26/2011 12:33 PM - trans (Thomas Sawyer)

Status:	Feedback	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:		

Description

Is there any significant reason for #undef_method to raise an error if the method is already undefined? If no, then change it to just continue on. It can return true/false to relay if was undefined vs already undefined.

Related issues:

Related to Ruby master - Feature #6241: Module#method_defined? with inherited...	**Rejected**	**04/01/2012**

History

#1 - 11/26/2011 06:47 PM - alexeymuranov (Alexey Muranov)

I can imagine that raising errors in such cases might be meant to discourage excessive metaprogramming.

#2 - 03/28/2012 12:49 AM - mame (Yusuke Endoh)

- Status changed from Open to Assigned

- Assignee set to matz (Yukihiro Matsumoto)

- Target version changed from 1.9.4 to 2.0.0

#3 - 04/01/2012 01:30 AM - matz (Yukihiro Matsumoto)

- Status changed from Assigned to Feedback

I think raising error can catch potential bugs earlier. What is the benefit of ignoring error?

Matz.

#4 - 04/01/2012 03:19 AM - trans (Thomas Sawyer)

B/c often times it's not an error. Cases such as undefining method before redefining new one to suppress warning message of overriden method. Or different versions of a library might get used where one has such method and another does not.

If we need to remove a method from a class/module that may or may not have the method defined, it's less optimal. We either have do something like:

```
if instance_methods(false).include?(:foo)
undef_method(:foo)
end
```

Or,

```
begin
undef_method(:foo)
rescue NameError
end
```

Both of which entail more overhead and considerations than is really necessary.

On the other hand, if it did not raise an error and returned true/false instead, then it is easy enough for us to handle the error if it truly matters.

```
success = undef_method(:foo)

raise NameError, "undefined method foo' for class#{self}'" unless success
```

#undef_method is not something that's used very frequently, so I think code improvements for these cases is worth it.

#5 - 04/01/2012 03:53 AM - aprescott (Adam Prescott)

What about two methods, undef_method and undef_method!, one which returns a
boolean, one which raises?

#6 - 04/01/2012 08:10 AM - nobu (Nobuyoshi Nakada)

trans (Thomas Sawyer) wrote:

> If we need to remove a method from a class/module that may or may not have the method defined, it's less optimal. We either have do something like:
>
> if instance_methods(false).include?(:foo)
> undef_method(:foo)
> end

You can use Module#method_defined? in this case.

```
if method_defined?(:foo)
undef_method(:foo)
end
```

#7 - 04/01/2012 12:19 PM - trans (Thomas Sawyer)

nobu (Nobuyoshi Nakada) Well, first let me point out that if method #foo is private, then it ain't so simple again. But secondly and really more importantly, this would be because I constantly confuse #undef_method for #remove_method and vice-versa. Really, the name undef always throws me off and has me thinking it's the opposite of def, but it's not. (Yea, okay another pet-peeve.)

So let me back up to the beginning and substitute #remove_method for where I had been saying #undef_method.

However, now that both of these methods have been brought up, in truth, I think it is equality applicable to both. And really which functionality is "right", just depends on the developers particular usecase.

So after some thought, I think Adam's proposal might be the best idea. It gives us the option of either functionality as needed.

#8 - 11/20/2012 10:26 PM - mame (Yusuke Endoh)

- Target version changed from 2.0.0 to 2.6

#9 - 12/25/2017 06:15 PM - naruse (Yui NARUSE)

- Target version deleted (2.6)