Ruby - Feature #11911

Immutable method definitions and/or static dispatch

12/28/2015 06:17 PM - mlarraz (Matt Larraz)

Status:	Feedback	
Priority:	Normal	
Assignee:		
Target version:		

Description

One of Ruby's biggest strengths is the ability for anyone, at any time, to redefine (almost) any behavior. But this is also one of its biggest weaknesses.
Ruby has a very liberal dynamic dispatch, so any method can be redefined anywhere in the code, meaning we can never have any guarantees about behavior.

Other languages with dynamic dispatch (like C++ or Java) also allow for static dispatch. In particular, Java has dynamic dispatch by default, with the final keyword marking a method as immutable.

In Ruby, this might look something like:

```
def foo
 'foo'
end

final :foo

# Raises an exception
def foo
 'bar'
end
```

I see this as analogous to freezing a string.

Note that if somebody really needs to overwrite an immutable method, they can still do so, just in a more explicit way:

```
undef_method :foo

# Works as expected
def foo
 'bar'
end
```

This eliminates some ambiguity.

I'm not sure how feasible this is (or whether this is the ideal syntax), but I'd like to hear what the community thinks of such a concept in general.

History

#1 - 12/30/2015 12:41 AM - duerst (Martin Dürst)

- Status changed from Open to Feedback

Ruby's ability to change any method anytime, and C++/Java's ability to overwrite some methods in subclasses, are conceptionally quite different. Which one are you interested in, and why? What is your use case?

#2 - 01/03/2016 07:04 AM - mlarraz (Matt Larraz)

I suppose I'm talking specifically about the first, that is, the ability to change any method at any time.

The most obvious use case I can imagine is an application that wants to guarantee that it's running the stock stdlib, with no monkey patches. Given a large enough number of gems (or even files in the codebase itself), auditing all of them for monkey patches becomes expensive. *Consistently* auditing all of them to ensure no monkey patches get introduced becomes cost-prohibitive. In such a case, it might also be convenient to have a command-line flag that disables any modifications to the stdlib.

As a highly contrived example, a malicious gem author could hide a monkey patch in the middle of his codebase, overwriting Kernel#puts to spy on all of the application's output. There is presumably a non-negligible number of Ruby developers who would like to easily guard against something like this.