Ruby trunk - Feature #15781

Unify Method List Introspection?

04/21/2019 08:43 PM - rbjl (Jan Lelis)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

Although Ruby has many core methods for retrieving the list of methods available to an object, or to the instances of a class, I believe they have gotten a little confusing (also see):

- Object#methods and Module#instance_methods do not include **private** methods (at the same time they do include **protected** ones). There is already Object#public_methods (and Object#protected_methods) for distinguishing visibility scope , but no way to get *all* methods of an object.
- There is the inconsistency that in most cases the argument being passed to *_methods methods let's you decide if you want to consider the inheritance chain, or not - But the prominent exception is Object#methods which instead toggles inheritance to singleton only! (for which we also have Object#singleton_methods)
- There is no direct API for getting a list of private singleton methods

Now that we have keyword arguments, we could provide a single API for listing methods. One way of doing so could be the Object#shadow's methods method. Having a keyword arguments based API would allow users to specify the dimensions of their requests better - should it:

- return the object's methods, or methods of its instances?
- return only methods of a specific visibility scope?
- return only methods of a specific inheritance level (e.g. only singleton, or all the way down to BasicObject)?

What do you think about having one unified way for retrieving an object's method list?

History

#1 - 04/21/2019 11:08 PM - shevegen (Robert A. Heiler)

Now this is for matz to consider, so I'll not comment much on the suggestion itself.

I would, however had, still like to point out a few things:

(1) Even if we assume that there may be confusion (let's only assume so for now), I think your example of a "shadow's methods method" is even more confusing to me. I don't even know what a shadow per se is; although I remember having once had created a class where methods could be "switched out" and then "switched back in" again many years ago. Not sure why, but I don't seem to need it anymore. It may have had something to do with structs or something ... or trying to make an object like struct, but storing data internally (I don't even remember whether that was possible; I think Struct does something magical :P but ... I really don't remember).

(2) The thing with private, public and protected is ... well. Ruby's OOP model is a bit different and depending on your point of view, you can come from a more smalltalk-specific view (so .send() is great), or a more java-esque view (stricter separation). Ultimately this is all a design decision, but personally I feel that ruby's OOP model is much, much closer to smalltalk. I am not saying that this is the "only true way", since evidently we can find a counter-argument with the addition of the method called .public_send(), but my personal opinion is that ruby is closer to the smalltalk-ish view. From that point of view, the more radical simplification would be to only have .methods(), and remove the rest. ;)

Actually that is what I would think of Objects#methods in the first place. To be honest, I do not even seem to often need to find out all available methods, but I can see how this may be important somtimes, e. g. irb/pry and similar projects.

(3) Personally I don't like keyword arguments that much; I find them confusing, so it would be a bit amusing to me to see them used here, with the goal of ... simplifying the code. ;) I have had to use APIs where a keyword argument was required, and it annoyed me, because I was thinking how passing an oldschool hash into that method would have been so much simpler instead (since there would be no runtime error, or whatever it was). Specifically I have had this error with the web-scraper called kimurai. Now that project is actually really good and useful, but the top-level APIs were a nightmare to use, lots of warnings called when run under -w too.

This may not be representative of every other project, but I don't know ... I find keyword arguments to not be that useful for my own use cases. Matz also pointed out already before RubyKaigi that the keyword arguments may be modified.

(4) I somewhat agree that it could be useful to have simpler means in order to obtain all available methods for any given object, but I am not completely sure that the suggestions achieve that as such. To me, personally, I think the by far most logical name would simply be .methods().

#2 - 04/21/2019 11:16 PM - shevegen (Robert A. Heiler)

Another idea - partially related perhaps. Or just totally random ...

If we were to ignore the current API, then we could have:

```
Object#methods
```

To obtain all methods, as an Array. And then we could still apply filtering on this result, such as via .public? and .private? and .protected?

Such as:

```
Object.methods.protected? # return all protected methods
Object.methods.private?   # return all private methods
```

This presently would not work because .methods will return an Array, but I was thinking of a special Array that could still carry "meta-information" like this but would behave as Array for all other purposes. (Though it would be fun to have meta-tags on Arrays ... but I am really just throwing out random ideas here without having thought about that much at all.)

Alternatively:

```
Object.protected_methods?
Object.public_methods?
Object.private_methods?
Object.methods?
```

(The trailing '?' is as an example; it could possibly be omitted. I just like trailing '?' methods in general, they are really great.)