Ruby master - Feature #15225

OpenStruct: Recursively converting child Hash objects to OpenStruct objects.

10/15/2018 02:03 PM - abhionlyone (Abhilash Reddy)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

Currently, OpenStruct allows this:

```
person = OpenStruct.new
person.age = 25
person.name = "Abhilash"
puts person.name # "Abhilash"
```

But something like this is not possible:

```
person.more_info = {country: "Japan",  skills: ["Ruby", "C++"]}
puts person.more_info.country
# NoMethodError (undefined method `country' for {:country=>"Japan"}:Hash)
```

It will be very helpful if we can provide an option to recursively convert child Hash objects to OpenStruct objects. By default it can be set to false.

I've done a basic implementation of this and created a PR here: https://github.com/ruby/ostruct/pull/5

Please let me know if there are any objections. If this can be accepted, I would like to add test cases for the same.

History

#1 - 10/15/2018 02:03 PM - abhionlyone (Abhilash Reddy)

- Subject changed from OpenStruct: Recursively converting child Hash object attributes to OpenStruct objects. to OpenStruct: Recursively converting child Hash objects to OpenStruct objects.

#2 - 10/15/2018 08:35 PM - shevegen (Robert A. Heiler)

This is an interesting suggestion.

I actually implemented something similar for a class that I tend to call Configuration.

This class also allows for using its keys as method names, so I can relate to your suggestion here. (If you are curious, class Configuration usually reads .yml files from a configuration directory, and then enables/disables certain functionality and information on a per-project basis).

On the other hand, I am not sure if this should be the (new) default for OpenStruct, despite feeling that it may be useful. Perhaps something of a toggle way, to allow a variant of OpenStruct that responds to methods as you describe, without changing the status quo of OpenStruct's behaviour (You should also define the behaviour what happens when a key is removed again; will the method remain or be removed as well? Or just return nil but remain there?)

I myself probably do not need it, primarily because I rarely use Struct/OpenStruct; and for keys-as-methods, I already use specialized classes such as my custom class Configuration (although I have this for my colour-class too, where I can simply use .slateblue() or .royalgreen() etc... to make use of the "HTML web-colours"), yet I can see that this may be useful for other people too, so it may be worthwhile to think about it (that is of course my personal opinion; at the end of the day you only have to convince matz).

I think we could actually decouple your suggestion, away from OpenStruct, to a more general question:

Should core-ruby allow for a class or a pattern that allows for keys in a key-value pair (a hash) to be used/usable as method calls? I can not answer this question really

but I think it is worth to have a deeper look at it.

Last but not least, I would encourage a little patience in regards to the proposal, mostly so that other people can also comment on the usefulness of the underyling pattern (even though your suggestion is solely about OpenStruct, I think the idea and intent behind the suggestion can also be asked more generally).

Having this as the new default may be somewhat difficult though, as the path towards ruby 3.x will probably not have too many incompatibilities (e. g. as opposed to the full transition from ruby 1.8.x to 2.x) - but again, that is only my own perception, I have been very wrong before. :)

#3 - 10/15/2018 11:35 PM - mame (Yusuke Endoh)

Objection. It is too implicit and ad-hoc to convert a symbol-key hash to an OpenStruct only when assigning it to a field of another OpenStruct. I believe, in many cases, it can be statically determined that the value being assigned is Hash or not. It would be readable and easy to maintain to convert it explicitly only when it is a Hash.

I somewhat agree with shevegen: accessing an entry of a symbol-key hash in a method style would be useful, without OpenStruct. Though, it is very very big change, so I'm unsure if it is acceptable or not.

#4 - 10/16/2018 05:07 AM - abhionlyone (Abhilash Reddy)

shevegen (Robert A. Heiler) wrote:

> This is an interesting suggestion.
>
> I actually implemented something similar for a class that I tend to call Configuration.
>
> This class also allows for using its keys as method names, so I can relate to your suggestion here. (If you are curious, class Configuration usually reads .yml files from a configuration directory, and then enables/disables certain functionality and information on a per-project basis).
>
> On the other hand, I am not sure if this should be the (new) default for OpenStruct, despite feeling that it may be useful. Perhaps something of a toggle way, to allow a variant of OpenStruct that responds to methods as you describe, without changing the status quo of OpenStruct's behaviour (You should also define the behaviour what happens when a key is removed again; will the method remain or be removed as well? Or just return nil but remain there?)
>
> I myself probably do not need it, primarily because I rarely use Struct/OpenStruct; and for keys-as-methods, I already use specialized classes such as my custom class Configuration (although I have this for my colour-class too, where I can simply use .slateblue() or .royalgreen() etc... to make use of the "HTML web-colours"), yet I can see that this may be useful for other people too, so it may be worthwhile to think about it (that is of course my personal opinion; at the end of the day you only have to convince matz).
>
> I think we could actually decouple your suggestion, away from OpenStruct, to a more general question:
>
> **Should core-ruby allow for a class or a pattern that allows for keys in a key-value pair (a hash) to be used/usable as method calls?** I can not answer this question really but I think it is worth to have a deeper look at it.
>
> Last but not least, I would encourage a little patience in regards to the proposal, mostly so that other people can also comment on the usefulness of the underyling pattern (even though your suggestion is solely about OpenStruct, I think the idea and intent behind the suggestion can also be asked more generally).
>
> Having this as the new default may be somewhat difficult though, as the path towards ruby 3.x will probably not have too many incompatibilities (e. g. as opposed to the full transition from ruby 1.8.x to 2.x) - but again, that is only my own perception, I have been very wrong before. :)

Thanks for taking the time to review this feature request. If possible please take a look at this PR https://github.com/ruby/ostruct/pull/5/files.

I'm really sorry that I've not explained what I'm trying to do in a proper manner.

```
person = OpenStruct.new({name: "Abhilash", more_info: {country: "JAPAN", skills: ["Ruby", "C++"]}})
puts person.more_info # This will be still a Hash

person = OpenStruct.new({name: "Abhilash", more_info: {country: "JAPAN", skills: ["Ruby", "C++"]}}, {
```

```
recursive: true})
puts person.more_info # Now this will be a OpenStruct object
```

So unless you explicitly pass **recusrive: true** to the OpenStruct constructor, The child Hash Objects will never be converted to OpenStruct objects.

So, This will not create any compatibility issues for the existing code.

#5 - 10/16/2018 05:18 AM - shevegen (Robert A. Heiler)

My apologies; I wasn't quite fully awake when I wrote my above comment since, obviously, easy "creation of ad-hoc methods" is a characteristic trait of struct/openstruct already (not sure why I somewhat forgot this; guess I have not used these two for a while), so please disregard some of my above comment accordingly

It may still be worthwhile to have a pattern that may be used for classes, e. g. a module that may extend this functionality onto different classes - but this may be for another suggestion altogether rather than distract from what the author of the issue request here.