Ruby trunk - Feature #14602

Version of dig that raises error if a key is not present

03/13/2018 06:29 PM - amcaplan (Ariel Caplan)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

Currently, if I have a hash like this:

```
{
    :name => {
        :first => "Ariel",
        :last => "Caplan"
    }
}
```

and I want to navigate confidently and raise a KeyError if something is missing, I can do:

```
hash.fetch(:name).fetch(:first)
```

Unfortunately, the length of the name, combined with the need to repeat the method name every time, means most programmers are more likely to do this:

```
hash[:name][:first]
```

which leads to many unexpected errors.

The Hash#dig method made it easy to access methods safely from a nested hash; I'd like to have something similar for access without error protection, and I'd think the most natural name would be Hash#dig!. It would work like this:

```
hash = {
    :name => {
        :first => "Ariel",
        :last => "Caplan"
    }
}
hash.dig!(:name, :first) # => Ariel
hash.dig!(:name, :middle) # raises KeyError (key not found: :middle)
hash.dig!(:name, :first, :foo) # raises TypeError (String does not have #dig! method)
```

History

#1 - 03/13/2018 07:42 PM - shevegen (Robert A. Heiler)

I think this may be somewhat problematic since it does not appear to fit to other methods that end with a "!", such as .chop() versus .chop!() for a String or .map() versus .map!() for an Array.

In the past I thought that "!" would mean mostly "modify in place", but matz wrote somewhere on the bug tracker that it is more meant as a "caution" indicator to the ruby user.

Another problem is, I think, that your suggestion of .dig!() does something that .dig() itself isn't doing (raising multiple different errors). But that may just be me, perhaps others see no problem - at the end of the day you'd only have to convince matz anyway. :)

#2 - 03/14/2018 02:56 AM - duerst (Martin Dürst)

Would a keyword parameter to dig work for you?

E.g. hash.dig!(:name, :middle, raise_error: true) or something similar.

#3 - 03/14/2018 12:58 PM - amcaplan (Ariel Caplan)

shevegen (Robert A. Heiler) wrote:

> I think this may be somewhat problematic since it does not appear to fit to other methods that end with a "!", such as .chop() versus .chop!() for a String or .map() versus .map!() for an Array.
>
> In the past I thought that "!" would mean mostly "modify in place", but matz wrote somewhere on the bug tracker that it is more meant as a "caution" indicator to the ruby user.
>
> Another problem is, I think, that your suggestion of .dig!() does something that .dig() itself isn't doing (raising multiple different errors). But that may just be me, perhaps others see no problem - at the end of the day you'd only have to convince matz anyway. :)

You have a good point about the bang methods often signifying an in-place operation rather than an error-prone one; the latter is more of a Rails convention than a Ruby one, and I mixed things up. Thank you for pointing that out. *Mea culpa.*

Ideally, we'd just have Hash#fetch take an arbitrary number of arguments, each representing another layer of depth. However, that option is closed off due to backwards compatibility issues, since it already takes an optional second argument representing a default return value in case the item isn't found.

So it sounds like we need a new method name. Perhaps something like #deep_fetch would work, but I'll have to think on it more to see if I can come up with something better. **EDIT: Apparently there's already a** deep_fetch gem **that does exactly this, which might indicate that the name #deep_fetch would be understood intuitively by the community.**

In terms of the multiple errors: This isn't actually new, and in fact I copied the current behavior of Hash#dig with the exception of adding the KeyError:

```
hash = {
    :name => {
        :first => "Ariel",
        :last => "Caplan"
    }
}

hash.dig(:name, :first) # => Ariel
hash.dig(:name, :middle) # => nil
hash.dig(:name, :first, :foo) # raises TypeError (String does not have #dig method)
```

#4 - 03/14/2018 01:01 PM - amcaplan (Ariel Caplan)

duerst (Martin Dürst) wrote:

> Would a keyword parameter to dig work for you?
>
> E.g. hash.dig!(:name, :middle, raise_error: true) or something similar.

I appreciate the thought. I personally would be more likely to do hash.fetch(:name).fetch(:middle) instead of adding a keyword argument to #dig, unless the list was extremely long (probably at least 4 consecutive keys), which I'd suspect is unusual enough that it's not worth adding to Ruby core for that unusual case.