

Ruby trunk - Feature #12282

Hash#dig! for repeated applications of Hash#fetch

04/14/2016 06:35 AM - robb (Robb Shecter)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	
Description	
<p>A new feature for your consideration: #dig! which is to #fetch as #dig is to #[]. For me and maybe many others, Hash#fetch is used much more than Hash#[]. And traversing multiple fetches isn't very convenient nor Ruby-like, e.g.: places.fetch(:countries).fetch(:canada).fetch(ontario).</p> <p>Here's how it would work:</p> <pre>places = { countries: { canada: true } } places.dig :countries, :canada # => true places.dig! :countries, :canada # => true places.dig :countries, :canada, :ontario # => nil places.dig! :countries, :canada, :ontario # => KeyError: Key not found: :ontario</pre> <p>Here's an implementation and tests: https://gist.github.com/dogweather/819ccdb41c9db0514c163cfdb1c528e2</p>	
Related issues:	
Has duplicate Ruby trunk - Feature #15563: #dig that throws an exception if a... Open	

History

#1 - 04/14/2016 06:56 AM - sawa (Tsuyoshi Sawada)

This makes sense only within limited cases, i.e. when the same key never appears at different depths. For example, if you get an error:

```
hash.dig!(:foo, :bar, :foo) # => KeyError: Key not found: :foo
```

you cannot tell whether the :foo at the first depth or the third depth (or both) is missing. In such case, there is not much difference from doing:

```
hash[:foo][:bar][:foo] # => NoMethodError: undefined method `[]' for nil:NilClass
```

from the point of view of information the error provides. (With dig!, all you can tell is that the error was not caused by :bar.) I do not see much value in having a method for such limited use case.

#2 - 04/14/2016 07:21 AM - nobu (Nobuyoshi Nakada)

- Description updated

I'm negative because:

1. This example is wrong.

```
places.dig :countries, :canada, :ontario # => nil
```

It raises a TypeError.

2. It feels curious to me that the method with '!' raises an exception whereas the method without '!' doesn't.

#3 - 04/14/2016 07:22 AM - nobu (Nobuyoshi Nakada)

Nobuyoshi Nakada wrote:

It raises a TypeError.

So you have the method which raises an exception already.

#4 - 04/14/2016 08:09 AM - sawa (Tsuyoshi Sawada)

Nobuyoshi Nakada wrote:

It raises a TypeError.

I think it is a typographical error of

```
places.dig :countries, :ontario # => nil
places.dig! :countries, :ontario # => KeyError: Key not found: :ontario
```

#5 - 04/15/2016 03:03 AM - shyouhei (Shyouhei Urabe)

I don't like the name. It doesn't uniform other usage of bang in method names.

#6 - 04/16/2016 04:20 AM - shevegen (Robert A. Heiler)

I concur with Shyouhei Urabe - the name seems to not entirely fit the given outcome.

More typical use cases of methods with ! bang, if we ignore any exception, would be more akin to things such as:

```
x = "abc"      # => "abc"
x.delete 'c'   # => "ab"
x              # => "abc"
x.delete! 'c'  # => "ab"
x              # => "ab"
```

On the topic of hashes as data structures, assumingly that they may be more nested than the usual array, I tend to always attempt to have all hashes and arrays as simple as possible, if and when that is possible (it is not always possible, see the older discussions about before .dig was added; but dig! is a weird name, we want to obtain something, not change the data structure right?).

#7 - 08/15/2017 06:35 PM - k0kubun (Takashi Kokubun)

How about this name?

```
places.deep_fetch(:countries, :canada, :ontario)
```

I've encountered the case which I did "places.fetch(:countries).fetch(:canada).fetch(:ontario)" multiple times. I want this method.

#8 - 12/28/2018 05:15 AM - robb (Robb Shecter)

Thanks everyone, for the discussion. I realize that my original comparison with #dig had a typographical error. Here's a gem implementation, with examples that are correct: <https://github.com/dogweather/digbang>

```
require 'dig_bang'

places = {
  world: {
    uk: true,
    usa: true
  }
}

# No difference when the key exists
places.dig :world, :uk # true
places.dig! :world, :uk # true

# A relevant error when the key is missing
places.dig :world, :uk, :alaska # nil
places.dig! :world, :uk, :alaska # KeyError: Key not found: :alaska
```

About the method name with the bang. I see that this might be more of a Rails naming convention, which ! methods perform the same action, but throw an error instead of returning a nil on failure. And that's exactly my intent with dig! vs. dig. Basically, a "checked" dig. I don't think that Ruby has a naming convention for an alternate interface which throws an exception vs. return a nil. (?)

```
module DigBang
  def self.fetch_all(fetchable, keys)
    keys.reduce(fetchable) { |a, e| a.fetch(e) }
  end
end
```

```
class Hash
  def dig!(*keys)
    DigBang.fetch_all(self, keys)
  end
end

class Array
  def dig!(*keys)
    DigBang.fetch_all(self, keys)
  end
end
```

#9 - 12/28/2018 05:31 AM - robb (Robb Shecter)

Another naming idea is #fetch_all, signalling that this is essentially a #fetch over a list of keys.

#10 - 01/26/2019 02:18 PM - k0kubun (Takashi Kokubun)

- Has duplicate Feature #15563: #dig that throws an exception if an key doesn't exist added