Ruby master - Feature #11747

"bury" feature, similar to 'dig' but opposite

11/27/2015 08:16 PM - dam13n (damien sutevski)

Status:	Rejected
Priority:	Normal
Assignee:	matz (Yukihiro Matsumoto)
Target version:	

Description

In Matz's recent Rubyconf talk, he used this example for the new 'dig' feature coming in Ruby 2.3:

```
# we want this
data[:users][0][:name]

# we can do this w/o nil errors
data.dig(:users, 0, :name)
```

What I'm proposing is a 'bury' feature that is the opposite of 'dig' in a sense. It inserts a value at an arbitrary depth, for example:

```
data.bury(:users, 0, :name, 'Matz')
```

This will create a nested hash or an array automatically at each step if it doesn't already exist, and that can be inferred from the what the user is passing (such as a symbol or string for a hash or an integer for an array). It's similar to autovivification but more powerful!

This behavior is very common, at least in my experience, so a dry method built into Ruby would be awesome!

Related issues:

Has duplicate Ruby master - Feature #13179: Deep Hash Update Method	**Rejected**

History

#1 - 11/27/2015 10:33 PM - nobu (Nobuyoshi Nakada)

- Description updated

- Status changed from Open to Feedback

How can it know what should be created, hash, array, or struct?

#2 - 11/28/2015 02:38 PM - sawa (Tsuyoshi Sawada)

> inferred from the what the user is passing (such as a symbol or string for a hash or an integer for an array)

I don't think this is a good idea. I think it should rather depend on the class of the receiver.

```
{}.bury(:users, 0, :name, 'Matz') # => {:users => {0 => {:name => "Matz"}}}
[].bury(:users, 0, :name, 'Matz') # => error
{}.bury(0, 1, 2, :foo) # => {0 => {1 => {2 => :foo}}}
[].bury(0, 1, 2, :foo) # => [[nil, [nil, nil, :foo]]]
```

and similar for struct.

#3 - 12/02/2015 11:36 PM - dam13n (damien sutevski)

Tsuyoshi Sawada wrote:

> > inferred from the what the user is passing (such as a symbol or string for a hash or an integer for an array)
>
> I don't think this is a good idea. I think it should rather depend on the class of the receiver.
>
> ```
> {}.bury(:users, 0, :name, 'Matz') # => {:users => {0 => {:name => "Matz"}}}
> [].bury(:users, 0, :name, 'Matz') # => error
> {}.bury(0, 1, 2, :foo) # => {0 => {1 => {2 => :foo}}}
> [].bury(0, 1, 2, :foo) # => [[nil, [nil, nil, :foo]]]
> ```

> and similar for struct.

I agree. I should clarify that I was assuming the class of the receiver (data) was known in my example. The inference I was talking about was that a buried 0 would imply an array position by default instead of a hash key. But if it was strictly determined by the receiver class, that'd still be useful.

#4 - 12/07/2015 07:29 AM - matz (Yukihiro Matsumoto)

- Status changed from Feedback to Rejected

It's not clear to generate either Hash, Array, or Struct (or whatever) to bury a value.
So it's better to reject now.

Matz.

#5 - 02/23/2017 06:48 AM - nobu (Nobuyoshi Nakada)

- Has duplicate Feature #13179: Deep Hash Update Method added

#6 - 05/17/2018 04:42 PM - briankung (Brian Kung)

- File bury_examples.rb added

matz (Yukihiro Matsumoto) wrote:

> It's not clear to generate either Hash, Array, or Struct (or whatever) to bury a value.

Would it be desirable to specify the new object in a block? That would make it somewhat symmetrical to how Hash.new takes a block as a default value. For example:

```
[{users: ['skipped']].bury(:users, 1, :name, 'Matz') { Hash.new }
# [{:users => ['skipped', {:name => 'Matz'}]}]

{users: {0 => nil}}.bury(:users, 0, :name, 'Matz') { |next_arg| Struct.new(next_arg).new }
# {:users => {0 => #<struct name="Matz">}}
#
# If the one of the retrieved values is nil, in this case {0 => nil},
# should #bury overwrite it?
```

If this is okay, then it might even be nice if #dig took a block as well as a fallback value:

```
[].dig(1) { 'default' }
#=> "default"
```

Additional #bury examples have been attached as bury_examples.rb.

#7 - 07/14/2019 02:52 AM - bkatzung (Brian Katzung)

Much of this has been available through my XKeys gem since Q2 2014.

```
data = {}.extend XKeys::Auto # Vs ::Hash, uses arrays for int keys
data[:users, 0, :name] # nil
data[:users, 0, :name, :raise => true] # KeyError
data[:users, :[], :name] = 'Matz' # :[] is next index, 0 in this case
# {:users=>[{:name=>"Matz"}]}
pick = [:users, 0, :name]
data[*pick] # Matz
data[:users, 0, :accesses, :else => 0] += 1
# {:users=>[{:name=>"Matz", :accesses=>1}]}
```

#8 - 01/11/2020 08:38 PM - cvss (Kirill Vechera)

A proposal to specify the path for bury with classes as values of a hash arg:

```
{}.bury(users: Array, 0 => Hash, name: Hash, something: 'Value') # {user: [{name: {something: 'Value'}]}
```

So all absent nodes could be created via klass.new

#9 - 01/11/2020 08:44 PM - cvss (Kirill Vechera)

A one-liner alternative for hash-only cases can be implemented using Enumerable#reduce:

```
root = {}
```

```
[:a, :b, :c].reduce(root){@1[@2]||={}}[:d] = 'E' # root => {:a=>{:b=>{:c=>{:d=>"E"}}}}
```

#10 - 04/16/2020 08:39 AM - schwad (Nick Schwaderer)

I think the issues/problems specified in the comments are not present with a Hash-only implementation. :)

I would be supportive of re-considering this feature just for use with a Hash, where I believe 80% of the real-life use cases would (and do) exist. I have encountered this need before in the wild, but not with Arrays. In fact, I'm only here because it seems like something one would 'expect' ruby already to do.

Thanks much for hearing me out.

Files

bury_examples.rb	1 KB	05/17/2018	briankung (Brian Kung)