Ruby - Feature #14564

`dig` opposite method

03/01/2018 08:57 AM - nilcolor (Aleksey Blinov)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

We have nice dig method that helps a lot.
Though we didn't have an opposite method that allows setting a value.
I know we already have these:
https://bugs.ruby-lang.org/issues/11747
https://bugs.ruby-lang.org/issues/13179
Both were closed because of name or lack of use cases. Let me promote the new name for this:

```
class Hash
  def expand(*where, value)
    where[0..-2].reduce(self) { |h, key|
      h[key] = h[key] || {}
    }[where[-1]] = value
    self
  end
end

{}.expand(:a, :b, :c, 42)                  # => {:a=>{:b=>{:c=>42}}}
{}.expand(:a, 0, :c, 42)                   # => {:a=>{0=>{:c=>42}}}
{a: {}}.expand(:a, :b, :c, 42)             # => {:a=>{:b=>{:c=>42}}}
{a: {b: nil}}.expand(:a, :b, :c, 42)       # => {:a=>{:b=>{:c=>42}}}
{a: {foo: "bar"}}.expand(:a, :b, :c, 42)   # => {:a=>{:foo=>"bar", :b=>{:c=>42}}}
{a: {b: "wat"}}.expand(:a, :b, :c, 42)
# => TypeError: no implicit conversion of Symbol into Integer


class Array
  def expand(*where, value)
    where[0..-2].reduce(self) { |a, idx|
      a[idx] = a[idx] || []
    }[where[-1]] = value
    self
  end
end

[].expand(2, 1, 3, "?")               # => [nil, nil, [nil, [nil, nil, nil, "?"]]]
[1, [0, 2], []].expand(1, 1, "BAM")   # => [1, [0, "BAM"], []]
[1, [0, 2], []].expand(2, 0, "BAM")   # => [1, [0, 2], ["BAM"]]
```

Use cases: working with deeply nested structures, used as parameters (params[:a][:nested][:some_id] = 42).
In general, I think it's mostly useful for Hashes. Though having this on Array may be useful as well.

Related issues:

Related to Ruby - Feature #19699: Need a way to store values like dig	**Closed**

History

#1 - 03/01/2018 12:55 PM - shevegen (Robert A. Heiler)

I have nothing against the functionality, but I think the name .expand()
is not a good one. When I read .expand, I think of the opposite of
flatten; or it reminds me of .extend.

I don't have a better name suggestion myself, though.

#2 - 03/01/2018 01:10 PM - zverok (Victor Shepelev)

Was already proposed as a "bury" (direct antonym to dig) and rejected: https://bugs.ruby-lang.org/issues/11747 and https://bugs.ruby-lang.org/issues/13179

Matz's response:

- "It's not clear to generate either Hash, Array, or Struct (or whatever) to bury a value.
  So it's better to reject now." to first and
- "You have to come up with a better name candidate and concrete use-case." to second.

BTW, you may be interested to take a look at my experimental hm gem, which defines some declarative hash processing helpers, including bury. It, BTW, decides to generate Array on numeric bury key, and Hash on any other, but I understand that it could be too vague for some cases.

```
Hm({a: {foo: "bar"}}).bury(:a, :b, :c, 42).to_h
# => {:a=>{:foo=>"bar", :b=>{:c=>42}}}

Hm({a: {b: "wat"}}).bury(:a, :b, :c, 42).to_h
# TypeError: String is not diggable

Hm([1, [0, 2], []]).bury(2, 0, "BAM").to_h # well, to_h is weird here, but works
# => [1, [0, 2], ["BAM"]]

Hm([]).bury(2, 1, 3, "?").to_h
# => [nil, nil, [nil, [nil, nil, nil, "?"]]]
```

#3 - 03/07/2018 05:29 PM - nilcolor (Aleksey Blinov)

I know about those 2 proposals. I references them in the description)

Name is a hard topic. As for types - I'd say that having same type as a receiver is enough.
So, {}.whatever_name(*) will generate nested Hashes. [].whatever_name(*) - Arrays. This will cover the majority of cases.

Names... Personally, I find bury nice) Though a bit of horrory. expand - yeah, maybe not that good.

What about "ruin", "embed", or "melt"?

#4 - 03/07/2018 11:24 PM - phluid61 (Matthew Kerwin)

Clearly the only accurate name for this method is store_recursive_with_autovivification

A lot of people ask for it, but I still can't see how this is a good thing to add to the language. In your personal code, sure -- maybe even in a gem -- but not the core.

#5 - 07/12/2022 06:10 PM - professor (Todd Sedano)

Often my team needs to modify deep hash structures and we created another implementation of the bury method.

We suggested our code as a modification to Hash in ActiveSupport PR. First, we wanted to verify that the ruby language does not want a bury method on a Hash.

I find the code in our PR easier to understand than the implementation suggested in this issue 14564 and in 13179.

#6 - 07/13/2022 04:00 PM - chrisseaton (Chris Seaton)

Instead of different dig and bury tools, a 'lens' abstraction could combine the two.

```
deep_hash = {a: {b: {c: {d: 100}}}}

p deep_hash.dig(:a, :b, :c, :d)

class Lens
  def self.lens(*keys)
    lens = Leaf.new(keys.pop)
    lens = Node.new(keys.pop, lens) until keys.empty?
    lens
  end

  class Node
    def initialize(key, child)
      @key = key
      @child = child
    end

    def get(object)
      @child.get object[@key]
    end
```

```
    def set(object, value)
      @child.set object[@key], value
    end
  end

  class Leaf
    def initialize(key)
      @key = key
    end

    def get(object)
      object[@key]
    end

    def set(object, value)
      object[@key] = value
    end
  end
end

lens = Lens.lens(:a, :b, :c, :d)
p lens.get(deep_hash)
lens.set deep_hash, 14
p lens.get(deep_hash)
```

#7 - 05/30/2023 10:04 PM - byroot (Jean Boussier)

- Related to Feature #19699: Need a way to store values like dig added