Ruby master - Feature #16993

Sets: from hash keys using Hash#key_set

06/26/2020 08:31 PM - marcandre (Marc-Andre Lafortune)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

To create a set from hash keys currently implies a temporary array for all keys, rehashing all those keys and rebuilding a hash. Instead, the hash could be copied and its values set to true.

```
h = {a: 1}
# Now:
Set.new(h.keys) # => Set[:a]
# After
h.key_set # => Set[:a], efficiently.
```

Related issues:

Related to Ruby master - Feature #16989: Sets: need ♥	**Assigned**

History

#1 - 06/26/2020 08:47 PM - marcandre (Marc-Andre Lafortune)

- Related to Feature #16989: Sets: need ♥ added

#2 - 06/27/2020 01:55 AM - sawa (Tsuyoshi Sawada)

Would

```
Set.new(h.each_key)
```

not work?

#3 - 06/27/2020 05:44 AM - marcandre (Marc-Andre Lafortune)

sawa (Tsuyoshi Sawada) wrote in #note-2:

> Would
>
> ```
> Set.new(h.each_key)
> ```
>
> not work?

It will definitely work, but it will be the same as Set.new(h.keys) (a bit slower because enumerator is a bit slow). It still iterates on the keys, and add them to the new hash.

Here's a comparison:

```
# frozen_string_literal: true
require 'set'
require 'benchmark/ips'

class Hash
  def key_set
    s = Set.new
    s.instance_variable_set(:@hash, transform_values { true })
    s
  end
end

size = 50
h = (1..size).to_h { |i| ['x' * i, nil] }

Benchmark.ips do |x|
  x.report("key_set")          { h.key_set }
  x.report("keys")             { Set.new(h.keys)  }
```

```
  x.report("new(each_key)")     { Set.new(h.each_key)  }
  x.report("each_key{}")        { h2 = {}; h.each_key {|k| h2[k] = true} ; h2  }
```

The last example builds a Hash, not a Set, but it is to show that you can not be quicker than that if you rehash the keys.
I get these results:

```
Calculating -------------------------------------
             key_set    244.549k (± 7.4%) i/s -      1.219M in   5.017876s
                keys     82.661k (± 2.3%) i/s -    417.400k in   5.052408s
       new(each_key)     75.002k (± 5.0%) i/s -    377.400k in   5.045102s
          each_key{}    114.757k (± 3.8%) i/s -    582.000k in   5.079700s
```

If you increase the size, the ratio will be bigger.
A builtin keyset would be even faster, since it would avoid calling the block { true }; yielding is not super fast in Ruby.