

Ruby - Feature #5553

A method for Hash that works differently depending on whether a key exists

11/03/2011 03:41 AM - sawa (Tsuyoshi Sawada)

Status:	Rejected	
Priority:	Normal	
Assignee:		
Target version:		
Description		
A method Hash# <i>if_key</i> (key, [default], &pr) which works like the following will be often used, and is useful.		
<pre>a = {morning: "morning", daytime: "daytime", evening: "evening", nothing: nil} a.if_key(:morning){ str "#{str}!"} #=> "morning!" a.if_key(:nothing){ str "#{str}!"} #=> "" a.if_key(:midnight){ str "#{str}!"} #=> nil a.if_key(:nothing, "no"){ str "#{str}!"} #=> "no"</pre>		
That is, when key exists, then the corresponding value will be passed to pr. Otherwise, the given `default` or the implicit default will be returned.		

History

#1 - 11/03/2011 07:22 AM - alexeymuranov (Alexey Muranov)

In your example, the :nothing key exists, so shouldn't it be
a.if_key(:nothing, ""){|str| "#{str}!"} #=> "" ?

Why would the code with this method be better than the following one:

```
if a.has_key?(key)
```

block here

else

default value or another block here

end

or (a.has_key?(key) ? simple operation : default)

This seems easier to read.

If you plan to always use it with the same block, maybe it should be made into a separate class with an appropriate method?

#2 - 11/03/2011 05:43 PM - sawa (Tsuyoshi Sawada)

The whole point of this suggestion is that I feel some redundancy to have to call the method fetch or [] on the hash after checking that the key exists using key?. With this proposed method, the method call fetch or [] is unnecessary, and you can just refer to a block variable.

#3 - 11/03/2011 07:48 PM - matz (Yukihiro Matsumoto)

- Status changed from Open to Feedback

```
fetch if key
else
  block
end
```

#4 - 11/03/2011 08:44 PM - alexeymuranov (Alexey Muranov)

I see. But then you may also want to pass a second procedure which is run if the key does not exist, and to have both the key and the value passed to the procedures, that is to introduce a method of the form #fetch_and_use(key, proc_if_key_found, proc_if_not). I am afraid that optimizing like this for all possible use cases may require introducing many new methods.

On the other hand, if you want to use a hash in this way with same procedures multiple times, then what about adding a method Hash#on_found_proc= analogous to Hash#default_proc= to be able to do like this:

```
a = {morning: "morning", daytime: "daytime", evening: "evening", nothing: nil}
a.default = "default"
a.on_found_proc = proc do |hash, key, value|
  "#{value}!"
end
a[:morning] # => "morning!"
a[:night]   # => "default"
```

Update 4/11/2011. If the goal is to simply avoid searching for the key twice, then there is Hash#assoc:

```
key_if_exists, value = h.assoc(key)
if key_if_exists.nil?
  puts "Bad key"
else
  puts "The value is #{value}"
end
```

However, if you use nil as a key, then this will not work. It is still possible to do

```
if pair = h.assoc(key)
  puts "The value is #{pair[1]}"
else
  puts "Bad key"
end
```

Or to introduce Hash#paranoiac_fetch :) :

```
key_exists, value = h.paranoiac_fetch(key)
if key_exists
  puts "The value is #{value}"
else
  puts "Bad key"
end
```

As to the original proposal, maybe defining some kind of a lazy Hash#map_values(&block) would be a better? (Or Hash#on_found_proc=(&block), that would behave identically.)

I saw [#4890](#), but there it is exclusively about Enumerable.

#5 - 11/20/2012 08:58 PM - mame (Yusuke Endoh)

- Status changed from Feedback to Rejected

No feedback, looks hopeless to me. Closing.

--

Yusuke Endoh mame@tsg.ne.jp