Ruby - Feature #5550

Hash#depth, Hash#flat_length for recursive hashes

11/03/2011 03:10 AM - sawa (Tsuyoshi Sawada)

Status:	Rejected
Priority:	Normal
Assignee:	
Target version:	

Description

I often have a hash whose value is recursively a hash, which may look like the following:

```
{"Japan" =>
    {"Hokkaido" => "Sapporro", ...},
    {"Honhuu" =>
        {"Aomori" => "Hirosaki", ...},
        {"Akita" => ...},
        ...
    },
    {"Shikoku" => ...},
    ...
}
```

In these cases, it will be convenient if there is a way to know the (maximum) depth of he original hash, and the numbers of all the "terminal nodes". I would like to propose two methods Hash#depth and Hash#flat_length, whose Ruby implementation can be as follows:

```
class Hash
def depth
 1 + (values.map{|v| Hash === v ? v.depth : 1}.max)
end
def flat_length
 values.inject(0){|sum, v| sum + (Hash === v ? v.flat_length : 1)}
end
end
```

History

#1 - 11/03/2011 07:56 PM - matz (Yukihiro Matsumoto)

- Status changed from Open to Feedback

Hashの本質はkey-valueのマッピングなので、valueが再帰的にHashであることを想定した(再帰的なHashでなければ役に立たない)メソッドを追加することには抵抗があります。
わずか6行のmonkey patchingを避けるためにすべてのRubyに追加すべきメソッドですか?

#2 - 11/06/2011 12:46 AM - alexeymuranov (Alexey Muranov)

Excuse me, can you be more precise with your example please? Ruby does not accept it (after removing the dots "..."). Are you talking about nested hashes? How about creating a class Tree that would inherit from Hash and define additional methods there?

#3 - 11/06/2011 05:35 PM - trans (Thomas Sawyer)

I take it you meant nested hash. I think your methods will infinite loop on recursive hash --and that needs to be considered.

I understand #depth, Array might use such a method too. But #flat_length, I don't quite get what is being counted.

#4 - 11/20/2012 08:59 PM - mame (Yusuke Endoh)

- Status changed from Feedback to Rejected

No feedback, looks hopeless to me. Closing.

--
Yusuke Endoh mame@tsg.ne.jp