Ruby - Bug #1852

Enumerable's #hash Raises ArgumentError When Recursive Values are Present

08/01/2009 03:47 PM - runpaint (Run Paint Run Run)

Status:	Closed		
Priority:	Normal		
Assignee:	marcandre (Marc-Andre Lafortune)		
Target version:	1.9.2		
ruby -v:	ruby 1.9.2dev (2009-08-01 trunk 24343) [i686-linux]	**Backport:**	

Description

=begin
Enumerable's #hash methods now raise ArgumentErrors when the structure contains a recursive reference. This is hopefully a bug. #hash is expected to return a Fixnum, not make judgments about the object. #to_s and #inspect both work with such objects, which reinforces my assumption.

```
h={}
=> {}
h[:key] = h
=> {:key=>{....}}
h.hash
ArgumentError: recursive key for hash



a=[]
=> []
a << a
=> [[....]]
a.hash
ArgumentError: recursive key for hash
```

(Note that the exception message is wrong here, too).

This is presumably the same reason that methods like Hash#invert and Array#sort now raise under the same conditions. I suppose this could be intentional, but if so could this policy be clarified? The status quo seems to be that:

- Recursive Hash keys are prohibited. An ArgumentError is raised if an attempt is made to create one.
- Recursive values (for hashes and arrays) are allowed.
- Some methods (I've yet to compile a list) will raise an ArgumentError when called on an object with recursive values.

This behavior doesn't seem desirable to me. I'd rather one or the other: Enumerable methods simply make the best of recursive structures (as before); or, recursive values are prohibited outright (as with Hash keys).
=end

History

#1 - 08/01/2009 05:25 PM - runpaint (Run Paint Run Run)

=begin

> Is it valuable to implement such function?

Yes. Even if it simply returned a constant for the degenerate case. (I'd rather a collision than an exception). To me, #hash is an integral part of Ruby objects. Having it raise with specific instances of a couple of classes just feels broken to me.

Again, though, I'd prefer that the policy question be addressed before a fix is determined. Is it desirable that recursive values be allowed yet raise when standard methods are called on their container?

=end

#2 - 08/02/2009 02:46 PM - runpaint (Run Paint Run Run)

=begin

- several real applications are found

This change broke a lot of specs which was how I noticed it. For example, Array#&, Array#-, Array#|, Array#sort, Array.flatten, Array#uniq, and Array#uniq! all now raise ArgumentErrors when previously they didn't. And that's just the Array methods for which this scenario was being verified. That's quite wide-scale breakage, and I still don't understand the reason. Why is it beneficial for these methods to raise exceptions when previously they dealt with recursive structures?

From what you're saying the plan is to allow recursive values but leave it to the individual methods to decide whether they want to behave or not. This will lead to more exception handling code scattered around because previously compliant functions are now temperamental. Or something equally ugly like "ary.uniq! unless ary.include?(ary)".

- O(n) behavior by constant hash value is not a problem for the applications
- someone write a patch

This would be a reasonable requirement if it were a feature being suggested, but that's not the case here. A multitude of methods have suddenly started raising exceptions because of what appears to be a new feature, and there has been no explanation why. That confuses me.

--
Run Paint Run Run

=end

#3 - 08/03/2009 12:24 PM - runpaint (Run Paint Run Run)

=begin

- several real applications are found

> This change broke a lot of specs which was how I noticed it. For

> I don't count RubySpec as a real application.

So your argument requires:

a) Enough "real" users running bleeding-edge Ruby to make it statistically likely that the regression will be noticed.
b) A multiplicity of those users to:
i) Determine why their application is now periodically dying.
ii) File a bug report that points back to this change.

What's more, the bug report that seemingly triggered this change was not derived from a "real application". So why the uneven burden of proof?

Again, all I'm asking for is an explanation of this change. It's hard enough to spec trunk Ruby without seemingly arbitrary changes being made "just because". :-/

--
Run Paint Run Run

=end

#4 - 08/03/2009 12:55 PM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:24697] Re: [Bug #1852] Enumerable's #hash Raises ArgumentError When Recursive Values are Present"
on Mon, 3 Aug 2009 02:07:47 +0900, Tanaka Akira akr@fsij.org writes:

|>> * several real applications are found
|>
|> This change broke a lot of specs which was how I noticed it. For
|
|I don't count RubySpec as a real application.

Crashing Array methods (and Set) seems enough important for me. So,
the point is how to fix it.

```
      matz.
```

=end

#5 - 08/05/2009 05:33 AM - marcandre (Marc-Andre Lafortune)

=begin
I agree wholeheartedly with Run Paint. As I pointed in my patch for recursive equality (http://redmine.ruby-lang.org/issues/show/1448), the ideal solution is that recursive structures have their hash collide to the same value. That value can be chosen arbitrarily. I should have updated ticket #1298 in that regard.

I only see disadvantages to the new behavior of raising an exception. As was pointed out, real world usage of recursive structures is probably rare, but I believe a reasonable solution can be implemented and could only improve Ruby. The recent behavior of disallowing recursive structures as hash indexes could also be reverted (ticket #1298).

The algorithm is simple: the top-level #hash should return the magic constant when detection is detected.

Although the C implementation will probably be pretty different, here's a Ruby implementation given as an example. This relies on an exception being thrown when all the catches have been exited, i.e. we're at the top level #hash method.

HASH_FOR_RECURSIVE_STRUCTURES = -42

class Array
def hash
catch :recursion_detected_for_hash do
return calculate_hash_value unless recursion_detected
end
throw :recursion_detected_for_hash rescue HASH_FOR_RECURSIVE_STRUCTURES
end
end

class Hash
exact same code...
end

Same with Structs and Range (see ticket #1885)

Note: To be completely consistent, it is imperative that the top level returns the constant, even if the recursion is detected in another class. This is to avoid some complex cases like:

a = []; h = {:e => a}; a << h
b = []; i = {:e => b}; b << i
j = {:e => b}

h, i and j should have the same hash.

j.hash will detect the recursion in Array#hash (for b),

while h.hash and i.hash will detect it in Hash#hash (for h and i)

=end

#6 - 08/05/2009 12:53 PM - mame (Yusuke Endoh)

=begin
Hi,

2009/8/3 Tanaka Akira akr@fsij.org:

> In article 67e307490908012245x3bf3be81oc7f060c2569ad4ab@mail.gmail.com,
> Run Paint Run Run runrun@runpaint.org writes:
>
> > - several real applications are found
> >
> > This change broke a lot of specs which was how I noticed it. For
>
> I don't count RubySpec as a real application.

I found one example that seems to be practical and considerable.

The following program implements a graph with adjacency list by
using Array and Struct:

```
Node = Struct.new(:item, :neighbors)
node1 = Node.new(1, [])
node2 = Node.new(2, [node1])
node1.neighbors << node2
```

Then consider a function of depth first search with Hash:

```
def dfs(node, visited = {}, &block)
return if visited[node]
yield(node)
visited[node] = true
node.neighbors.each {|n| dfs(n, visited, &block) }
end

dfs(node1) {|n| p n.item }
```

This function worked on 1.8.7 and 1.9.1p243, but not on trunk.

If hash of recursive structure were a constant, this function
would work. Collisions, however, would be terribly frequent.

I wonder that it might be intrinsically unreasonable to use
primitive data types for such a purpose.

--
Yusuke ENDOH mame@tsg.ne.jp

=end

#7 - 09/05/2009 12:29 AM - runpaint (Run Paint Run Run)

=begin
If this is, as matz implies, a regression, it would be useful to get it fixed before the 1.9.2 freeze.
=end

#8 - 09/09/2009 06:25 PM - runpaint (Run Paint Run Run)

=begin
Tanaka,

Thank you for your explanation. :-) I haven't fully understood the
issues involved yet, but assuming that we don't revert this commit...

In the case of Hash, if recursive keys are to be prohibited, why don't
methods like Hash#merge! also reject them? Hash values are still
allowed to be recursive, but then #invert is fatal, as is #hash. It is
most unorthodox for an object to be legally constructed yet refuse to
compute a hash code. :-/

IOW, even if we do accept this fix, do we acknowledge that there's
more work to be done here?

> I guess the fix mentioned in [ruby-core:24761] may have some
> performance impact for usual non-cyclic hash key since all
> top-level #hash method call needs catch(). If the
> performance impact is small enough, no problem for the fix.

If a fix for special case noticeably impacted performance for the general case, that would certainly be unacceptable. But at the moment we're just guessing. Would you be willing to try an implementation of Marc-Andre's idea? Then we'll have a better idea of what options are available.

--
Run Paint Run Run

=end

#9 - 09/13/2009 02:46 PM - marcandre (Marc-Andre Lafortune)

- File recursive_hash.diff added

=begin
Here is a patch that provides a hash for recursive Array, Hash, Struct and Range. As discussed, there will be hash collisions when recursion is detected, although the hash of a recursive array with 1 element will be different from a recursive hash with 2 elements or from a recursive hash.

This makes it possible to use recursive structures as keys. It also re-enables the other related functionalities (Array#-, Set, etc...)

I have updated Rubyspecs to test for this, including some complex cases.

Is this ok to be committed?
=end

#10 - 09/15/2009 02:51 AM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:25543] [Bug #1852] Enumerable's #hash Raises ArgumentError When Recursive Values are Present"
on Sun, 13 Sep 2009 14:46:43 +0900, Marc-Andre Lafortune redmine@ruby-lang.org writes:

|Here is a patch that provides a hash for recursive Array, Hash, Struct and Range. As discussed, there will be hash collisions when recursion is detected, although the hash of a recursive array with 1 element will be different from a recursive hash with 2 elements or from a recursive hash.
|
|This makes it possible to use recursive structures as keys. It also re-enables the other related functionalities (Array#-, Set, etc...)
|
|I have updated Rubyspecs to test for this, including some complex cases.
|
|Is this ok to be committed?

Two points:

- I prefer using rb_catch_obj(), instead of using TAG_THROW directly, unless using it hinders performance badly.
- test suites (test/ruby/test_*.rb) should be updated as well (test_array.rb, test_thread.rb), when you check in the patch.

```
        matz.
```

=end

#11 - 09/15/2009 06:59 AM - marcandre (Marc-Andre Lafortune)

- File hash_additional.diff added

- File hash_merged.diff added

- Assignee set to marcandre (Marc-Andre Lafortune)

- Target version set to 1.9.2

=begin
I wanted to use rb_catch_obj at first but I saw the following issues:

1. rb_catch_obj sends :catch to nil. If for some strange evil purpose catch was redefined by the user (in Kernel, Object or as a singleton method of nil), then [].hash would call that code, which seems like a bad idea. I believe that #hash's use of throw & catch should be invisible to the user.
2. rb_catch_obj creates an instruction sequence. As such, it will show if "caller" is called within #hash, or in the backtrace of exceptions raised within #hash. Again I feel it should be invisible.

3. rb_catch_obj creates a temporary object to hold the instruction sequence (NEW_IFUNC). I lack the experience to be certain on how big of a performance penalty if would be, but I didn't like the idea of creating an object for every #hash call and leaving it for the GC to handle later.

I agree that it would be best to factorize the TAG_THROW handling. I extracted it from rb_f_catch and created a new function "rb_catch_func" that does the low-level catching. Using this new function for #hash should have negligible performance impacts compared to my previous version. I've attached the resulting patch in two formats. Is this ok?
=end

#12 - 09/16/2009 05:36 AM - marcandre (Marc-Andre Lafortune)

- File catch.diff added

- File hash_merged2.diff added

=begin
It turns out that Nobu had already a patch that changed rb_catch_obj to be exactly like my rb_catch_func but he never got the time to commit it. He also pointed out a small C90 compatibility issue.

After this change, exec_recursive can call rb_catch_obj and no new rb_catch_func is needed.

Unless there are additional comments or objections, I will commit the changes in a few days. Attached are the patches, one adapted from Nobu's patch for the changes to *catch*, the other for stuff relating to exec_recursive & hash (with modification to tests)
=end

#13 - 09/16/2009 05:59 AM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:25603] [Bug #1852] Enumerable's #hash Raises ArgumentError When Recursive Values are Present"
on Wed, 16 Sep 2009 05:36:25 +0900, Marc-Andre Lafortune redmine@ruby-lang.org writes:

|It turns out that Nobu had already a patch that changed rb_catch_obj to be exactly like my rb_catch_func but he never got the time to commit it. He also pointed out a small C90 compatibility issue.
|
|After this change, exec_recursive can call rb_catch_obj and no new rb_catch_func is needed.
|
|Unless there are additional comments or objections, I will commit the changes in a few days. Attached are the patches, one adapted from Nobu's patch for the changes to *catch*, the other for stuff relating to exec_recursive & hash (with modification to tests)

Thank you, and go ahead to check it in.

```
	matz.
```

=end

#14 - 09/16/2009 06:37 AM - marcandre (Marc-Andre Lafortune)

- Status changed from Open to Closed

- % Done changed from 0 to 100

=begin
Applied in changeset r24943.
=end

Files

recursive_hash.diff	7.24 KB	09/13/2009	marcandre (Marc-Andre Lafortune)
hash_additional.diff	5.04 KB	09/15/2009	marcandre (Marc-Andre Lafortune)
hash_merged.diff	10.2 KB	09/15/2009	marcandre (Marc-Andre Lafortune)
catch.diff	2.18 KB	09/16/2009	marcandre (Marc-Andre Lafortune)
hash_merged2.diff	9.04 KB	09/16/2009	marcandre (Marc-Andre Lafortune)