Ruby master - Bug #10713

Assigning default value for a Hash as an empty Array creating unpredictable results

01/07/2015 11:59 PM - punjab (Arvinder Singh)

Status:	Rejected		
Priority:	Normal		
Assignee:			
Target version:	2.2.0		
ruby -v:	ruby 2.2.0p0 (2014-12-25 revision 49005) [x86_64-darwin14]	**Backport:**	2.0.0: UNKNOWN, 2.1: UNKNOWN, 2.2: UNKNOWN

Description

Creating a Hash with a default value works fine, unless the default value is an empty Array.

E.g. the following returns an empty Hash...

```
irb(main):001:0> letters = Hash.new([])
=> {}
irb(main):002:0> letters[:a] << 1
=> [1]
irb(main):003:0> letters[:a] << 2
=> [1, 2]
irb(main):004:0> letters[:a]
=> [1, 2]
irb(main):005:0> letters
=> {}
```

whereas the following code explicitly defining hash keys works.

```
irb(main):001:0> letters = {a: [], b: []}
=> {:a=>[], :b=>[]}
irb(main):002:0> letters[:a] << 1
=> [1]
irb(main):003:0> letters[:a] << 2
=> [1, 2]
irb(main):004:0> letters[:a]
=> [1, 2]
irb(main):005:0> letters
=> {:a=>[1, 2], :b=>[]}
```

Is this an unpredictable(bug) or an expected behavior(feature). I tend to lean towards the former, but I might be wrong.

History

#1 - 01/08/2015 12:02 AM - hsbt (Hiroshi SHIBATA)

- Status changed from Open to Rejected

It's expected behavior

#2 - 01/08/2015 01:16 AM - duerst (Martin Dürst)

Hiroshi SHIBATA wrote:

> It's expected behavior

Hiroshi, can you tell us why it's expected behavior? It looks quite surprising.

#3 - 01/08/2015 01:33 AM - donburks (Don Burks)

Martin Dürst wrote:

> Hiroshi SHIBATA wrote:

> > It's expected behavior
>
> Hiroshi, can you tell us why it's expected behavior? It looks quite surprising.

I would agree that this is surprising behaviour. It would appear that in this case, the append operator is not re-assigning the value, the way it does any other time it is used. And it would appear to be not doing this specifically in the case where the default value for the Hash is specified as an empty array. I would like to understand why this behaviour is the way it is.

#4 - 01/08/2015 01:38 AM - austin (Austin Ziegler)

It's quite expected because the default array is created exactly once:

letters = Hash.new([])
letters.default.object_id # => 70310393550400
letters[:a].object_id # => 70310393550400
letters[:b].object_id # => 70310393550400

The pattern that Arvinder *wants* is:

letters = Hash.new { |h, k| h[k] = [] }
letters.default # => nil
letters.default_proc # => #Proc:0x007fe4d4099988@(irb):7
letters[:a].object_id # => 70310393174180
letters[:b].object_id # => 70310393171680

It's possible to make the correct pattern do the "wrong thing":

array = []
array.object_id # => 70310393550400
letters = Hash.new { |h, k| h[k] = array }
letters.default # => nil
letters.default_proc # => #Proc:0x007fe4d4099988@(irb):7
letters[:a].object_id # => 70310393550400
letters[:b].object_id # => 70310393550400

-a

On Wed, Jan 7, 2015 at 8:16 PM, duerst@it.aoyama.ac.jp wrote:

> Issue #10713 has been updated by Martin Dürst.
>
> Hiroshi SHIBATA wrote:
>
> > It's expected behavior
>
> Hiroshi, can you tell us why it's expected behavior? It looks quite
> surprising.
>
> ---
>
> Bug #10713: Assigning default value for a Hash as an empty Array creating
> unpredictable results
> https://bugs.ruby-lang.org/issues/10713#change-50846
>
> - Author: Arvinder Singh
> - Status: Rejected
> - Priority: Normal
> - Assignee:
> - Category: core
> - Target version: current: 2.2.0
> - ruby -v: ruby 2.2.0p0 (2014-12-25 revision 49005) [x86_64-darwin14]
>
> ## * Backport: 2.0.0: UNKNOWN, 2.1: UNKNOWN, 2.2: UNKNOWN
>
> Creating a Hash with a default value works fine, unless the default value
> is an empty Array.
>
> E.g. the following returns an empty Hash...
>
> ```
> irb(main):001:0> letters = Hash.new([])
> => {}
> irb(main):002:0> letters[:a] << 1
> => [1]
> ```

```
    irb(main):003:0> letters[:a] << 2
    => [1, 2]
    irb(main):004:0> letters[:a]
    => [1, 2]
    irb(main):005:0> letters
    => {}
```

    whereas the following code explicitly defining hash keys works.

```
    irb(main):001:0> letters = {a: [], b: []}
    => {:a=>[], :b=>[]}
    irb(main):002:0> letters[:a] << 1
    => [1]
    irb(main):003:0> letters[:a] << 2
    => [1, 2]
    irb(main):004:0> letters[:a]
    => [1, 2]
    irb(main):005:0> letters
    => {:a=>[1, 2], :b=>[]}
```

    Is this an unpredictable(bug) or an expected behavior(feature). I tend to
    lean towards the former, but I might be wrong.

    --
    https://bugs.ruby-lang.org/

--
Austin Ziegler * halostatue@gmail.com * austin@halostatue.ca
http://www.halostatue.ca/ * http://twitter.com/halostatue

#5 - 01/08/2015 04:42 AM - david_macmahon (David MacMahon)

Arvinder Singh wrote:

```
    irb(main):001:0> letters = Hash.new([])
    => {}
    irb(main):002:0> letters[:a] << 1
    => [1]
```

The letters[:a] part of the second line returns the "default value" of the Hash because the :a key does not exist in the Hash. The << 1 part pushes a 1 onto the end of the default value, but it does not assign the default value (and certainly not a modified copy of the default value) to the :a key of the Hash.

To get the behavior you expect/desire you can use letters[:a] <<= 1, which is equivalent to letters[:a] = letters[:a] << 1, but be careful when using Hashes with default objects:

```
irb(main):001:0> letters = Hash.new([]) # Default object == single instance
=> {}
irb(main):002:0> letters[:a] <<= 1
=> [1]
irb(main):003:0> letters[:a]
=> [1]
irb(main):004:0> letters
=> {:a=>[1]}
irb(main):005:0> letters[:b] # Returns default object (same object as letters[:a])!
=> [1]
```

To get a different object for each fetch of a non-existent key, use a Hash with a default Proc that returns new object every time:

```
irb(main):001:0> letters = Hash.new {[]} # Default Proc returns new instance each call
=> {}
irb(main):002:0> letters[:a] <<= 1
=> [1]
irb(main):003:0> letters[:a]
=> [1]
irb(main):004:0> letters
=> {:a=>[1]}
irb(main):005:0> letters[:b] # Returns different empty Array until letters[:b] is assigned
=> []
irb(main):006:0> letters[:b].object_id
=> 2152992900
irb(main):007:0> letters[:b].object_id
=> 2153118760
```

```
irb(main):008:0> letters[:b].object_id
=> 2153112180
irb(main):009:0> letters[:b] = letters[:b]
=> []
irb(main):010:0> letters[:b].object_id
=> 2156907140
irb(main):011:0> letters[:b].object_id
=> 2156907140
```

Given all the trickiness and because sometimes you don't have control over the creation of the Hash, I generally end up doing this instead:

```
>> # h is any Hash, e.g. from method call
>> h[:a] ||= []
=> []
>> h[:a] << 1
=> [1]
# etc...
```

This uses the "||=" idiom, which can obliterate an existing nil or false value, so you still have to be a little careful (though it's very rarely an issue).