Ruby - Feature #19063

Hash.new with non-value objects should be less confusing

10/16/2022 10:24 PM - baweaver (Brandon Weaver)

Status:	Rejected
Priority:	Normal
Assignee:	
Target version:	

Description

Related to #10713 and #2764.

Ruby's Hash.new accepts either a block or a param for its default value. In the case of non-value objects this leads to unexpected behaviors:

```
bad_hash_with_array_values = Hash.new([])
good_hash_with_array_values = Hash.new { |h, k| h[k] = [] }
```

While, as @hsbt (Hiroshi SHIBATA) has said in the past, this is behaving as intended for the Ruby language it has caused a lot of confusion in the community over the years and is a known sharp-edge.

My assertion is that this is not the intended behavior, and I cannot find a legitimate usecase in which someone intends for this to happen. More often new users to Ruby are confused by this behavior and spend a lot of time debugging.

We must consider the impact to Ruby users, despite what the intent of the language is, and make the language more clear where possible.

Given that, I have a few potential proposals for Ruby committers.

Proposal 1: Do What They Meant

When people use Hash.new([]) they mean Hash.new { |h, k| h[k] = [] }. Can we make that the case that if you pass a mutable or non-value object that the behavior will be as intended using dup or other techniques?

When used in the above incorrect way it is likely if not always broken code.

Proposal 2: Warn About Unexpected Behavior

As mentioned above, I do not believe there are legitimate usages of Hash.new([]), and it is a known bug to many users as they do not intend for that behavior. It may be worthwhile to warn people if they do use it.

Proposal 3: Require Frozen or Values

This is more breaking than the above, but it may make sense to require any value passed to Hash.new to either be frozen or a value object (e.g. 1 or true)

Updating RuboCop

Failing this, I am considering advocating for RuboCop and similar linters to warn people against this behavior as it is not intended in most to all cases:

https://github.com/rubocop/rubocop/issues/11013

...but as @ioquatix (Samuel Williams) has mentioned on the issue it would make more sense to fix Ruby rather than put a patch on top of it. I would be inclined to agree with his assessment, and would rather fix this at a language level as it is a known point of confusion.

Final Thoughts

I would ask that maintainers consider the confusion that this has caused in the community, rather than asserting this "works as intended." It does work as intended, but the intended functionality can make Ruby more difficult for beginners. We should keep this in mind.

Related issues:

Related to Ruby - Feature #19069: Default value assignment with `Hash.new` in...	**Rejected**

History

#1 - 10/17/2022 01:34 AM - ioquatix (Samuel Williams)

@matz (Yukihiro Matsumoto) what do you think?

#2 - 10/17/2022 02:47 AM - sawa (Tsuyoshi Sawada)

> When people use Hash.new([]) they mean Hash.new { |h, k| h[k] = [] }.

I don't think so. Can you prove it? For note, coming up with a lot of examples pointing towards that is not a proof.

> it is a known bug

This contradicts with what you have admitted:

> @hsbt (Hiroshi SHIBATA) (Hiroshi SHIBATA) has said in the past, this is behaving as intended

#3 - 10/17/2022 02:54 AM - baweaver (Brandon Weaver)

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

> > When people use Hash.new([]) they mean Hash.new { |h, k| h[k] = [] }.
>
> I don't think so. Can you prove it? For note, coming up with a lot of examples pointing towards that is not a proof.

Do you have examples of this? I have had several cases across multiple large Ruby companies where this behavior had confused people:

```
3.1.0 :001 > array = []
 => []
3.1.0 :002 > hash = Hash.new(array)
 => {}
3.1.0 :003 > hash[:a] << 1
 => [1]
3.1.0 :004 > hash[:b] << 2
 => [1, 2]
3.1.0 :005 > hash[:c] << 3
 => [1, 2, 3]
3.1.0 :006 > hash
 => {}
3.1.0 :007 > array
 => [1, 2, 3]
```

It is very similar to the default list argument in Python which is also confusing to people:

```
def append_to(element, to=[]):
    to.append(element)
    return to

my_list = append_to(12)
print(my_list)
# [12]

my_other_list = append_to(42)
print(my_other_list)
# [12, 42]
```

If you would like I can poll people, but when teaching Ruby it has been a very common source of confusion and I have yet to meet someone or seen code where this was intentional. Perhaps there are, but they are very rare, and this is a very common issue to Ruby users in the places I have worked.

> This contradicts with what you have admitted.

Yes and no. Yes, by the strict wording, it does and that was a mistake in calling it a "bug". The intent of that statement is that many Ruby users consider it to be a bug because it is unexpected.

#4 - 10/17/2022 02:56 AM - austin (Austin Ziegler)

This mistake is much more common than I would have thought, and it appears in some fairly large projects (not everyone may have access to this result because cs.github.com is still apparently in preview):

https://cs.github.com/?scopeName=All+repos&scope=&q=language%3Aruby+Hash.new%28%5B%5D%29

> Showing 1 - 20 of about 1.5k files found (in 612 milliseconds)

These are from the first twenty instances:

- ~~GitLab:~~ https://github.com/gitlabhq/gitlabhq/blob/eaeb21af27304897f46f91633e25cba23232349d/lib/gitlab/api_authentication/builder.rb#L12
- FactoryBot: https://github.com/thoughtbot/factory_bot/blob/ec71611954d07419c9b668be03641332443dcd78/lib/factory_bot/linter.rb#L22
- Fat Free CRM
  - 1: https://github.com/fatfreecrm/fat_free_crm/blob/5b73c62335835106c25f547d9810197d8b4c0ba8/lib/fat_free_crm/callback.rb#L58
  - 2: https://github.com/fatfreecrm/fat_free_crm/blob/5b73c62335835106c25f547d9810197d8b4c0ba8/lib/fat_free_crm/callback.rb#L74
- Instructure Canvas LMS: https://github.com/instructure/canvas-lms/blob/8bd46cb657ec925229bfcd5407c145ec121d85c7/app/models/context.rb#L182
- Chef
  - Knife: https://github.com/chef/chef/blob/e85fcb8c0ad1ea4ed6f830e4b585f27c2ada4994/knife/lib/chef/knife.rb#L173
  - Inspec: https://github.com/inspec/inspec/blob/af5b8335af052743914909e3833a88c7ba350ed8/lib/inspec/utils/nginx_parser.rb#L93
- Redmine:
  - 1: https://github.com/redmine/redmine/blob/823080b45e58563f989b992789ed340d358ed955/app/models/user.rb#660
  - 2: https://github.com/redmine/redmine/blob/823080b45e58563f989b992789ed340d358ed955/app/models/user.rb#693
- Browser CMS: https://github.com/browsermedia/browsercms/blob/0a7fb9219f6e24cce4271b420c2ea07febd1b748/app/models/cms/category_type.rb#L21
- AWS S3 SDK (feature testing, but…)
  - 1: https://github.com/aws/aws-sdk-ruby/blob/2e96092cca28a0422fc3950f078b0919b636c227/gems/aws-sdk-s3/features/client/step_definitions.rb#344
  - 2: https://github.com/aws/aws-sdk-ruby/blob/2e96092cca28a0422fc3950f078b0919b636c227/gems/aws-sdk-s3/features/client/step_definitions.rb#366
- Barkeep: https://github.com/ooyala/barkeep/blob/126b1b1f41f514396d3e22f9935fe4640de2a402/lib/string_filter.rb#L27
- Opal: https://github.com/opal/opal/blob/934aa61bdb80a9da6ce9c92b0eedba737bbc1f9f/lib/opal/compiler.rb#L255
- OpenProject:
  - 1: https://github.com/opf/openproject/blob/56738cbfe492dcde3ab80463bff33eed7ae765f4/lib/api/v3/activities/activity_eager_loading_wrapper.rb#L76
  - 2: (Hash.new({})) https://github.com/opf/openproject/blob/56738cbfe492dcde3ab80463bff33eed7ae765f4/lib/api/v3/activities/activity_eager_loading_wrapper.rb#L92
- Goldiloader: https://github.com/salsify/goldiloader/blob/ead058ab65ecf845c78b51307c048a9f09f87ef7/lib/goldiloader/auto_include_context.rb#L26
- rspec-core: https://github.com/rspec/rspec-core/blob/71823ba11ec17a73b25bdc24ebab195494c270dc/lib/rspec/core/filter_manager.rb#L205-L206
- ~~@shugo's textbringer:~~ https://github.com/shugo/textbringer/blob/65dff909295f14f466f7b27e824ea36fe1f9ddd7/bin/merge_mazegaki_dic#L5

So this is *very* widespread, and experienced Rubyists make this mistake.

Because this is something that would only affect Ruby versions going forward, I think that updating Rubocop and other linters is a necessary *first* step. Of the rest of the proposals, I think that in the short to medium term, the only viable option is *2. Warn About Unexpected Behavior*. I think that the *least* viable is *1. Do What They Meant*, and the better long time option is *3. Require Frozen or Values*.

Note: At least two of the examples later use hash[key] |= values, which is assignment, not resulting in the possible reuse of the default value in general. Others may result in the same.

#5 - 10/17/2022 03:01 AM - baweaver (Brandon Weaver)

> For note, coming up with a lot of examples pointing towards that is not a proof.

Then what would qualify as a proof? I am confused by this as it asserts that examples of people being confused are not proof, whenever by definition it proves that people are confused by this even if it does work as intended.

In Python there is the exit() syntax, and when one uses exit instead you get this:

```
>>> exit
Use exit() or Ctrl-D (i.e. EOF) to exit
```

One can argue that this works as intended, but the impact is that many people are still confused by this. We must not forget the impact of language choices, despite the intention behind them, as they make our language more unapproachable.

#6 - 10/17/2022 03:04 AM - baweaver (Brandon Weaver)

> austin (Austin Ziegler) wrote in #note-4:
> So this is *very* widespread, and experienced Rubyists make this mistake.
>
> Because this is something that would only affect Ruby versions going forward, I think that updating Rubocop and other linters is a necessary *first* step. Of the rest of the proposals, I think that in the short to medium term, the only viable option is *2. Warn About Unexpected Behavior*. I think that the *least* viable is *1. Do What They Meant*, and the better long time option is *3. Require Frozen or Values*.

I would agree that option 2 is the most viable without serious breaking behavior, and option 3 would be better long-term. Option 1 was an attempt at a compromise, but would have breaking changes as well.

#7 - 10/17/2022 03:16 AM - austin (Austin Ziegler)

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

> > When people use Hash.new([]) they mean Hash.new { |h, k| h[k] = [] }.
> > I don't think so. Can you prove it? For note, coming up with a lot of examples pointing towards that is not a proof.
>
> It partially comes from not necessarily understanding default_value in the Hash documentation. The documentation only uses value objects, not mutable objects, which leaves opportunities for this to be a mistake.
>
> It may not matter in short-lived scripts where the amount of bucketing in the Hash is small, but anything larger or longer-lived is likely to see unexpected behaviour. In the examples that I found on GitHub, none of the places where this was used seemed to be *intending* this to result in a single value, and none of them were doing hash[key] = [] explicitly to override the use of the default value.
>
> > it is a known bug
> > This contradicts with what you have admitted:
> > @hsbt (Hiroshi SHIBATA) (Hiroshi SHIBATA) has said in the past, this is behaving as intended
>
> This is not a contradiction. Every time I have seen this used, it is a *bug* in the system where the code was written.
>
> It's not a bug in Ruby…but perhaps it *should* be. I'm in favour of making it a warning, but it should be easy to turn off for the rare case when it is intended.

#8 - 10/17/2022 03:25 AM - austin (Austin Ziegler)

Edit on a couple of these. The GitLab behaviour, and @shugo's code both do hash[key] |= value_list, so they *do* effectively do hash[key] = value_list, which makes them safer to use *in these particular cases*. I don't think that this would necessarily be the case for every instance. I also happen to think that the use of hash[key] |= value_list is *clever* code, which is more likely to indicate mistakes.

I think that warning on the direct use of [], {}, and "" literals is a good choice (call that proposal 2.1), but *not* warning on the use of a variable. Maybe, as well, it only warns on the direct use of the *empty* literals.

#9 - 10/17/2022 03:37 AM - sawa (Tsuyoshi Sawada)

baweaver (Brandon Weaver) wrote in #note-5:

> Then what would qualify as a proof?

I don't think it is provable. That is my point. You made a statement that cannot be assured.

I know that a lot of beginners make the kind of mistake you mention. And I am not necessarily against improving that situation. However, I am against your description of the issue from a few respects.

1. Your assertion of this issue as being Ruby's bug or the behavior as unintended is a mistake as you have admitted in comment #3, and I consider it hostile to the Ruby developers. At most, you can say that it is a pitfall, but never as unintended. Code written by a beginner using this feature may likely (but not necessarily) be unintended, but the Ruby's behavior is not unintended. You should not confuse them.
2. Your proposal 1 would break one of the very basic principles of Ruby, or perhaps of any computer language, or even any natural language. That is, the arguments passed to a method are evaluated before (or independently of) the execution of the method. In other words, the semantics of the whole is calculated from the semantics of its parts. In the context of natural language, this principle is known as Frege's compositionality. What is relevant here is that the default object that is passed as an argument must be evaluated to be such object before initialization of the hash. A block does not have such limitation because it is not a default object per se, but is something that would be evaluated to be a default object.

Only proposal 2 seems to make sense to me, but with qualification that "it is a known bug" be removed. I do not like proposal 3, but at least it is better than proposal 1.

#10 - 10/17/2022 04:05 AM - ioquatix (Samuel Williams)

The current behaviour is ambiguous at best and as @austin has shown, buggy and incorrect at worst / in practice.

I prefer option 1 but it's true it can break compatibility with this poorly understood behaviour.

I agree in principle to issue a warning but I also don't think we should prevent the current usage, so I propose changing the interface slightly:

```
Hash.new(default_value, assign: true/false)
```

The current behaviour is retained by assign: false. Setting assign: true gives the proposal 1 behaviour, calling default_value.dup.

Then, we can warn if assign is not specified.

#11 - 10/17/2022 05:05 AM - duerst (Martin Dürst)

sawa (Tsuyoshi Sawada) wrote in #note-9:

> 1. Your proposal 1 would break one of the very basic principles of Ruby, or perhaps of any computer language, or even any natural language. That is, the arguments passed to a method are evaluated before (or independently of) the execution of the method. In other words, the semantics of the whole is calculated from the semantics of its parts. In the context of natural language, this principle is known as Frege's compositionality. What is relevant here is that the default object that is passed as an argument must be evaluated to be such object before initialization of the hash. A block does not have such limitation because it is not a default object per se, but is something that would be evaluated to be a default object.

There would definitely be quite a few details to be worked out, but starting with Hash.new([]), I don't see any problem. One of the basic principles of method arguments is that methods can respond differently to different arguments. That's the whole point of arguments. Of course, Hash.new(my_method_returning_an_empty_array) would behave exactly the same as Hash.new([]) (assuming my_method_returning_an_empty_array) really did what its name said). So compositionality would still hold.

#12 - 10/17/2022 05:11 AM - duerst (Martin Dürst)

ioquatix (Samuel Williams) wrote in #note-10:

> The current behaviour is ambiguous at best and as @austin has shown, buggy and incorrect at worst / in practice.
>
> I prefer option 1 but it's true it can break compatibility with this poorly understood behaviour.

There is definitely a chance of breaking compatibility. It would be good to find actual examples where Hash.new([]) (or anything similar to it) was intended as specified by Ruby. I haven't seen any such examples yet.

> I agree in principle to issue a warning but I also don't think we should prevent the current usage, so I propose changing the interface slightly:
>
> ```
> Hash.new(default_value, assign: true/false)
> ```
>
> The current behaviour is retained by assign: false. Setting assign: true gives the proposal 1 behaviour, calling default_value.dup.
>
> Then, we can warn if assign is not specified.

I think assign is ambiguous, as in any case something gets assigned somehow. I'd prefer something like dup: true/false, or maybe even Hash.new(dup: []). The later would be a shorthand for Hash.new { |h,k| h[k] = [] }. But that wouldn't eliminate the potential for misunderstandings for Hash.new([]).

#13 - 10/17/2022 05:46 AM - sawa (Tsuyoshi Sawada)

duerst (Martin Dürst) wrote in #note-11:

> Hash.new(my_method_returning_an_empty_array) would behave exactly the same as Hash.new([]) (assuming my_method_returning_an_empty_array) really did what its name said). So compositionality would still hold.

There is no problem with whatever my_method_returning_an_empty_array is as long as it is defined independently of/prior to its caller Hash.new. Proposal 1 is suggesting to let Hash.new control what my_method_returning_an_empty_array refers to. That breaks compositionality. Or else, it is suggesting to let the argument provide something like "a prototype of the default object" instead of the default object itself. That would change the whole role of the argument for this method.

#14 - 10/17/2022 05:59 AM - sawa (Tsuyoshi Sawada)

This issue is essentially the same issue as people mistakenly writing Array.new(5, []) with the intention to actually do Array.new(5){[]}. And the same

issue can appear elsewhere.

Users need to have better understanding of how argument passing works. Better documentation and warning may be a good idea. An ad hoc modification to the feature as in Proposal 1 would not really solve the problem.

#15 - 10/17/2022 06:28 AM - sawa (Tsuyoshi Sawada)

I can imagine why you particularly picked up (or people are more troubled with) Hash.new rather than Array.new. Probably, you are feeling that Hash.new {|h, k| h[k] = []} is too long to write. So I have an even better idea. If possible, what about actually allowing:

```
Hash.new{[]}
```

to be equivalent to:

```
Hash.new{|h, k| h[k] = []}
```

paying attention to the block variable signature? When the block signature is absent, the return value of the block would be automatically assigned to the hash with the key in question. This would mean something like this:

```
h1 = Hash.new{|h, k| h[k] = "foo"}
h2 = Hash.new{|h, k| "foo"}
h3 = Hash.new{"foo"}

h1[:a] # => "foo"
h2[:a] # => "foo"
h3[:a] # => "foo"

h1 # => {:a=>"foo"}
h2 # => {}
h3 # => {:a=>"foo"}
```

Then, you can suggest people to write Hash.new{[]} instead of Hash.new([]) if that is what is intended.

#16 - 10/17/2022 06:59 AM - ioquatix (Samuel Williams)

@sawa I appreciate your technical knowledge and insight.

However... the practical reality is the vast majority of the users are using this interface incorrectly. This might imply that the theoretical basis for how Hash.new(default_value) and Array.new(count, default_value) work, is impractical and not what users expect.

There are different ways you can frame this, all equally valid from a technical point of view. However, the current framing of the problem is causing real problems in production code.

So we need to propose how to

1. introduce a safer interface (dup or assign style options, {} constructor with 0-arity), and
2. inform existing users of the problem (warning, deprecation, etc),

Your proposal can help with (1), but I'm not sure it can help with (2). The question is, are there valid use cases for Hash.new(default_value) and/or Array.new(count, default_value) where the user expects mutable values to be shared? I have not yet seen a single compelling use case. If we can assert that there are no valid use cases, it presents other ways we can satisfy (1) and (2) since there is no valid case for compatibility, and all existing usage is either incorrect and/or buggy.

(If we did decide to introduce Hash.new{default_value} we should also consider Array.new(count){default_value} if it doesn't already exist.)

#17 - 10/17/2022 07:03 AM - ioquatix (Samuel Williams)

In support of Hash.new{default_value}, Array.new(count){default_value} works in the same way.

#18 - 10/17/2022 01:02 PM - Dan0042 (Daniel DeLorme)

This is a good idea. Even though the current behavior is "logical", and experienced rubyists have gotten so used to this that it's second nature, there's an opportunity to make ruby more friendly for beginners by removing a well-known gotcha.

Proposal 1 is just too magical for me. Using an empty array would mean something special and different from everything else.

Proposal 2 would cause warnings for valid code such as h[1] += [2]. It's good to help future rubyists write future code, but we have to consider existing code that is not broken. It's not good to force a change on valid code just because "this is the new way to do it".

Proposal 3 is imho the best. Existing code like h[1] += [2] would still work. It would cause code like h[1] << 2 to raise an exception, but that code is *already* broken anyway. So the backward compatibility issue is minor I think. But this could be a good opportunity to introduce #16153 as a way to have a deprecation phase for this change. So the previous code could warn "[] will be eventually frozen" instead of raising an exception.

#19 - 10/17/2022 01:47 PM - Dan0042 (Daniel DeLorme)

sawa (Tsuyoshi Sawada) wrote in #note-15:

```
h2 = Hash.new{|h, k| "foo"}
h3 = Hash.new{"foo"}

h2[:a] # => "foo"
h3[:a] # => "foo"

h2 # => {}
h3 # => {:a=>"foo"}
```

I *love* this idea! It makes the common case so much easier and uncluttered. It's easy to bypass if you need to. It works great in combination with a frozen default value; when Hash.new([]) doesn't work because you can't append to a frozen value, you then reach for the next option Hash.new{[]}

The main downside is incompatibility with previous ruby versions. If this change makes it into 3.2 and you use that version when writing Hash.new{[]}, then someone else running that code with ruby 3.1 will get buggy hard-to-diagnose behavior instead of a simple exception. So maybe this would be better as a new method, like Hash.new!{[]} or Hash{[]}.

Note that I haven't found that many gems that use Hash.new{value} syntax, and most of them would work fine with this change:

```
#actionpack-7.0.3/lib/action_controller/metal/parameter_encoding.rb
49:        @_parameter_encodings[action.to_s] = Hash.new { Encoding::ASCII_8BIT }

#actionview-7.0.3/lib/action_view/helpers/tag_helper.rb
40:      PRE_CONTENT_STRINGS              = Hash.new { "" }

#bunny-2.19.0/lib/bunny/authentication/credentials_encoder.rb
29:        @@registry ||= Hash.new { raise NotImplementedError }

#cucumber-core-10.1.1/lib/cucumber/core/test/filters/tag_filter.rb
32:            @test_cases_by_tag_name = Hash.new { [] }

#cucumber-core-10.1.1/lib/cucumber/core/test/result.rb
340:            @totals = Hash.new { 0 }

#cucumber-core-11.0.0/lib/cucumber/core/test/filters/tag_filter.rb
32:            @test_cases_by_tag_name = Hash.new { [] }

#cucumber-core-11.0.0/lib/cucumber/core/test/result.rb
340:            @totals = Hash.new { 0 }

#ice_nine-0.11.2/spec/unit/ice_nine/freezer/hash/class_methods/deep_freeze_spec.rb
13:      Hash.new {}.update(Object.new => Object.new)

#resque-2.2.1/lib/resque/server.rb
110:        hosts = Hash.new { [] }

#rspec-core-3.11.0/lib/rspec/core/formatters/deprecation_formatter.rb
144:            @seen_deprecations = Hash.new { 0 }

#rspec-core-3.11.0/lib/rspec/core/hooks.rb
496:          :before => Hash.new { BeforeHook },
497:          :after  => Hash.new { AfterHook  },
498:          :around => Hash.new { AroundHook }

#sequel-5.57.0/lib/sequel/extensions/_pretty_table.rb
42:      sizes = Hash.new {0}

#sprockets-4.1.1/lib/sprockets/transformers.rb
150:        transformers          = Hash.new { {} }
151:        inverted_transformers = Hash.new { Set.new }
```

#20 - 10/17/2022 03:02 PM - sawa (Tsuyoshi Sawada)

Dan0042 (Daniel DeLorme) wrote in #note-19:

> The main downside is incompatibility with previous ruby versions. [...] Note that I haven't found that many gems that use Hash.new{value} syntax, and most of them would work fine with this change

Right. If it causes any problems, a redundant |h, k| would need to be added to the block in such cases. But I think that transition would not be so difficult.

#21 - 10/17/2022 03:45 PM - jeremyevans0 (Jeremy Evans)

ioquatix (Samuel Williams) wrote in #note-16:

> The question is, are there valid use cases for Hash.new(default_value) and/or Array.new(count, default_value) where the user expects mutable values to be shared?

Sure:

```
class A; end
class B < A; end
class C < A; end
class_map = Hash.new(A)
class_map[:b] = B
class_map[:c] = C
```

Just because the default value is mutable doesn't mean the user's code plans to mutate it.

#22 - 10/17/2022 05:49 PM - baweaver (Brandon Weaver)

I would agree that Hash.new { [] } would be very nice, and as mentioned has a dual in Array.new(5) { [] }.

While I do not think that it is a bug (and my apologies for earlier bad wording) to have Hash.new([]) do what it currently does, I would like to clarify what I meant. My intended meaning was that new Ruby users *think* it is a bug when they realize the behavior, which can be the cause of confusion. My desire is to make Ruby more understandable for new users, and prevent confusion where possible.

I appreciate that newer versions of the docs explicitly call this behavior out, I had not seen that in the past, so I would like to thank whoever added that.

Given the above conversation I would still be in favor of a warning on instances of Hash.new when called with mutable objects like [] as this does a few things:

1. Call out potentially confusing behavior for new users
2. Not break compatibility
3. Could be turned off by advanced users who intend this behavior

Outside of clever usages in experimental repos I have not been finding intentional usages of this syntax, so I would think this would be an acceptable compromise given the frequency of unintended usage as noted by @austin.

@sawa / @matz (Yukihiro Matsumoto) - Should we open a new ticket for this shorthand Hash.new { [] } syntax? It may be worth its own new conversation, but I am very positive of this syntax.

#23 - 10/17/2022 06:29 PM - sawa (Tsuyoshi Sawada)

@baweaver (Brandon Weaver) wrote:

> Should we open a new ticket for this shorthand Hash.new { [] } syntax?

Either way is okay with me.

#24 - 10/18/2022 09:57 AM - etienne (Étienne Barrié)

While I like the idea of checking the arity of the block, maybe the core issue isn't entirely with the API for Hash.new, but the fact that default value/default proc are not visible. From the examples above, when we print the value of hash, it appears empty: {}.

By tweaking what is displayed by Hash#inspect and Hash#pretty_print, I believe it would be clearer to beginners and experienced Rubyists alike:

```
array = []
# => []
hash = Hash.new(array)
# => {}
hash[:a] << 1
# => [1]
hash[:b] << 2
# => [1, 2]
hash[:c] << 3
# => [1, 2, 3]
hash
# => #<Hash default=[1, 2, 3] {}>
```

#25 - 10/18/2022 10:51 AM - byroot (Jean Boussier)

I'm also of the stance that this shouldn't surprise someone familiar with how Ruby is evaluated, but it is true that people have to start from somewhere and it's always good to try to help them get that understanding.

The problem with warning on mutable default is:

- As mentioned before, yes mutable default is rare, but no unimaginable. It's a tough sell for a programming language to ban some usage like this.
- Few people run Ruby in verbose mode, and certainly not the beginners for whom this is intended.
- There's no convenient way to turn of a specific warning, short of adding another parameter or something.

That's why maybe this makes more sense as a rubocop cop.

That said, Hash.new { [] } would be a very welcome addition though, and is easy to implement. I agree you should open a feature request for it.

#26 - 10/18/2022 12:00 PM - sawa (Tsuyoshi Sawada)

I opened a new issue #19069 regarding my proposal https://bugs.ruby-lang.org/issues/19063#note-15. If you have comments related to that idea, please continue on #19069.

#27 - 10/19/2022 12:34 AM - nobu (Nobuyoshi Nakada)

Adding a new element in the default proc will add by just referencing.
I'm not for doing it implicitly.

#28 - 10/19/2022 12:39 AM - nobu (Nobuyoshi Nakada)

austin (Austin Ziegler) wrote in #note-4:

> **Note**: At least two of the examples later use hash[key] |= values, which is assignment, not resulting in the possible reuse of the default value in general. Others may result in the same.

Another example in enc/make_encmake.rb:

```
  deps = Hash.new {[]}

  # ...
        deps[e] <<= n unless default_deps.include?(n)
```

#29 - 10/19/2022 12:52 AM - baweaver (Brandon Weaver)

etienne (Étienne Barrié) wrote in #note-24:

> While I like the idea of checking the arity of the block, maybe the core issue isn't entirely with the API for Hash.new, but the fact that default value/default proc are not visible. From the examples above, when we print the value of hash, it appears empty: {}.
>
> By tweaking what is displayed by Hash#inspect and Hash#pretty_print, I believe it would be clearer to beginners and experienced Rubyists alike:
>
> ```
> array = []
> # => []
> hash = Hash.new(array)
> # => {}
> hash[:a] << 1
> # => [1]
> hash[:b] << 2
> # => [1, 2]
> hash[:c] << 3
> # => [1, 2, 3]
> hash
> # => #<Hash default=[1, 2, 3] {}>
> ```

I would also be very positive of this change, potentially for Array as well, to make this visible to folks working in the language. What would be the impact of changing inspect in this way? I do not think it would be breaking, but would want others opinions on this.

#30 - 10/19/2022 11:51 AM - Eregon (Benoit Daloze)

baweaver (Brandon Weaver) wrote in #note-29:

> I would also be very positive of this change, potentially for Array as well, to make this visible to folks working in the language.

For Array it doesn't make sense since the default value or block is evaluated only upon construction, and there is no need to keep it after.

> What would be the impact of changing inspect in this way? I do not think it would be breaking, but would want others opinions on this.

I'd think it'd break many programs which currently assume eval hash.inspect works, plus surprise people.

Based on Jeremy's example in https://bugs.ruby-lang.org/issues/19063#note-21 I think there is nothing we can do here, there are valid usages of a mutable default value.
It should be clear to any most programmers that Hash.new(...) only evaluates the value once (unlike Hash.new { ... }, and whether it .dup or so on each access is documented (it doesn't .dup).

#31 - 10/20/2022 02:41 AM - shugo (Shugo Maeda)

austin (Austin Ziegler) wrote in #note-8:

> Edit on a couple of these. The GitLab behaviour, and @shugo's code both do hash[key] |= value_list, so they *do* effectively do hash[key] = value_list, which makes them safer to use *in these particular cases*. I don't think that this would necessarily be the case for every instance. I also happen to think that the use of hash[key] |= value_list is *clever* code, which is more likely to indicate mistakes.
>
> I think that warning on the direct use of [], {}, and "" literals is a good choice (call that proposal 2.1), but *not* warning on the use of a variable. Maybe, as well, it only warns on the direct use of the *empty* literals.

hash[key] |= value_list is trivial code for those who understand side effects.
I'm against introducing such false positive warnings.

#32 - 10/20/2022 02:45 AM - mame (Yusuke Endoh)

Summary so far

Is there any example of Hash.new([]) being used intentionally?

Yes. Shugo's example is a valid usage of Hash.new([]).

Is there any example of Hash.new(mutable object) being used intentionally?

Yes. Jeremy's example is a valid usage of a mutable default value.

What will Proposal 1 break?

Jeremy's example can lead to strange situation because class objects are dup'ed.

What will Proposal 2 break?

Shugo's example will be warned nevertheless it is legitimate.

What will Proposal 3 break?

Both Shugo's example and Jeremy's example will require modification.

Next, we have a question: is this change worth the pain of incompatibility? I think people have different opinions.

Here's my personal opinion, however, I don't think we need to discuss this if we are going to add a cop to Rubocop. Rubocop is already known for excessive warnings, and provides a mechanism to selectively suppress warnings. So I agree with @byroot (Jean Boussier): it would make more sense as a Rubocop cop.

#33 - 10/20/2022 09:25 AM - shugo (Shugo Maeda)

austin (Austin Ziegler) wrote in #note-4:

> This mistake is much more common than I would have thought, and it appears in some fairly large projects (not everyone may have access to this result because cs.github.com is still apparently in preview):

It seems that only AWS S3 SDK has invalid code except when their clients mutate Hash values.

- FactoryBot: https://github.com/thoughtbot/factory_bot/blob/ec71611954d07419c9b668be03641332443dcd78/lib/factory_bot/linter.rb#L22

valid code: result[factory] |= errors

- Fat Free CRM
  - 1: https://github.com/fatfreecrm/fat_free_crm/blob/5b73c62335835106c25f547d9810197d8b4c0ba8/lib/fat_free_crm/callback.rb#L58

valid code: response[op[:position]] += [op[:proc].call(caller, context)]

- 2: https://github.com/fatfreecrm/fat_free_crm/blob/5b73c62335835106c25f547d9810197d8b4c0ba8/lib/fat_free_crm/callback.rb#L74

valid code: @view_hooks[hook] += [{ proc: proc,

- Instructure Canvas LMS: https://github.com/instructure/canvas-lms/blob/8bd46cb657ec925229bfcd5407c145ec121d85c7/app/models/context.rb#L182

valid code: ids_by_type[type] += [id]

- Chef
  - Knife: https://github.com/chef/chef/blob/e85fcb8c0ad1ea4ed6f830e4b585f27c2ada4994/knife/lib/chef/knife.rb#L173

valid code: ids_by_type[type] += [id]

- Inspec: https://github.com/inspec/inspec/blob/af5b8335af052743914909e3833a88c7ba350ed8/lib/inspec/utils/nginx_parser.rb#L93

valid code: agg_conf[x.key] += [x.vals]

- Redmine:
  - 1: https://github.com/redmine/redmine/blob/823080b45e58563f989b992789ed340d358ed955/app/models/user.rb#660

valid code result[role] = Project.where(:id => ids).to_a

- 2: https://github.com/redmine/redmine/blob/823080b45e58563f989b992789ed340d358ed955/app/models/user.rb#693

valid code: result[role] = Project.where(:id => ids).to_a

- Browser CMS: https://github.com/browsermedia/browsercms/blob/0a7fb9219f6e24cce4271b420c2ea07febd1b748/app/models/cms/category_type.rb#L21

valid code: map[ct.id.to_s] = ct.category_list.map { |c| [c.path, c.id.to_s]

- AWS S3 SDK (feature testing, but…)
  - 1: https://github.com/aws/aws-sdk-ruby/blob/2e96092cca28a0422fc3950f078b0919b636c227/gems/aws-sdk-s3/features/client/step_definitions.rb#344

invalid code: @tracker[type.to_sym] << event

- 2: https://github.com/aws/aws-sdk-ruby/blob/2e96092cca28a0422fc3950f078b0919b636c227/gems/aws-sdk-s3/features/client/step_definitions.rb#366

invalid code: @tracker[type.to_sym] << event

- Barkeep: https://github.com/ooyala/barkeep/blob/126b1b1f41f514396d3e22f9935fe4640de2a402/lib/string_filter.rb#L27

valid code: filter_methods.merge!({ method_name => filter_methods[method_name] + [filter_block] })

- Opal: https://github.com/opal/opal/blob/934aa61bdb80a9da6ce9c92b0eedba737bbc1f9f/lib/opal/compiler.rb#L255

valid code: @comments = ::Parser::Source::Comment.associate_locations(first_node, comments)

- OpenProject:
  - 1: https://github.com/opf/openproject/blob/56738cbfe492dcde3ab80463bff33eed7ae765f4/lib/api/v3/activities/activity_eager_loading_wrapper.rb#L76

valid code: hash[class_name] = class_name

- 2: (Hash.new({})) https://github.com/opf/openproject/blob/56738cbfe492dcde3ab80463bff33eed7ae765f4/lib/api/v3/activities/activity_eager_loading_wrapper.rb#L92

valid code: hash[class_name] = class_name

- Goldiloader: https://github.com/salsify/goldiloader/blob/ead058ab65ecf845c78b51307c048a9f09f87ef7/lib/goldiloader/auto_include_context.rb#L26

valid code: register_models(nested_models, nested_associations[association])

- rspec-core: https://github.com/rspec/rspec-core/blob/71823ba11ec17a73b25bdc24ebab195494c270dc/lib/rspec/core/filter_manager.rb#L205-L206

valid code:

```
        no_location_filters = locations[
          File.expand_path(ex_metadata[:rerun_file_path])
        ].empty?
```

#34 - 12/15/2022 08:46 AM - hsbt (Hiroshi SHIBATA)

- Related to Feature #19069: Default value assignment with `Hash.new` in block form added

#35 - 12/15/2022 08:50 AM - matz (Yukihiro Matsumoto)

- Status changed from Open to Rejected

Sorry for being late to reply. As @shugo (Shugo Maeda) mentioned above, there are a lot of code that use this functionality valid.
Considering this fact, we are not going to add any warning here for the core.

Matz.