Ruby master - Feature #10177

Hash#has_key? and Hash#has_value? should be deprecated

08/27/2014 10:57 PM - gogotanaka (Kazuki Tanaka)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

I referred to this Matz's remark.

http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-core/43765

And I agree with him, I supposed Hash#has_key? should be preferred over Hash#key?,

so I replaced all of Hash#has_key? and Hash#has_value? in lib/* with Hash#key? and Hash#value?

History

#1 - 08/28/2014 12:29 PM - Eregon (Benoit Daloze)

I guess your second sentence is meant the other way round:

"And I agree with him, I supposed Hash#key? should be preferred over Hash#has_key?, "

#2 - 08/28/2014 08:14 PM - gogotanaka (Kazuki Tanaka)

@ Mr. Benoit Daloze

oh.. Yes, that's what I meant!

Thank you.

#3 - 08/29/2014 06:36 PM - shevegen (Robert A. Heiler)

I liked has_key? but if matz prefers the other way I am fine,
and I like that there is a consistent definition there.

Making stdlib / corelib more consistent would be great, and hopefully
gem-installable in a modular way too, so

+1

for this.

I'll copy paste Matz's remark here again, just so that others don't
have to click an external link:

"The basic naming for methods in standard class libraries are:

- use basic form (include not includes)
- put question mark for predicates
- put bang mark for "dangerous" version of methods

"is_a" and "has_key" are exceptions. "is_a" (or "isa") used very often for inheritance in OO context.
"has_key" has already been deprecated by "key?"

Besides that, backward incompatibility introduced by renaming them would be unbearable."

(Note: I think matz actually meant "has_key?" rather than "has_key" in that email back then)

If this is changed and a new release of ruby is done, please don't forget to comment
on this in the changelog, so I can update my own ruby docu as well. Thanks!)

#4 - 09/07/2014 08:33 PM - rits (First Last)

in the interest of maximizing the English readability and sensibility (self documentability) of predicates, it is key? that should be deprecated.

when you have an instance method predicate in the form receiver.noun? this reads in English as "Receiver, are you a noun?", when in the form

receiver.adjective?, "Receiver, are you adjective?" There is simply not enough information in a single noun or adjective to ask anything else.

So hash.key? reads, "hash, are you a key?"

If you want to ask a question of the receiver other than "Are you a noun?" or "Are you adjective?" the predicate has to be modified to reflect the question.

has_key? is perfect, it has enough information to be self documenting, it also implies that grammatical object of the question (the key) needs to be passed in.

#5 - 09/07/2014 11:42 PM - jeremyevans0 (Jeremy Evans)

I agree with First Last that has_key? is better than key? from an English readability standpoint. Not knowing the history but being aware of both methods, I've always used has_key? instead of key? for that reason.

In my opinion, the backwards compatibility cost of deprecating then removing either has_key? or key? outweighs the benefit of removing one and standardizing on the other.

#6 - 09/13/2014 11:28 PM - gogotanaka (Kazuki Tanaka)

robert (Robert Gleeson) A. Heiler
@First Last
@Jeremy Evans

Thank you so much for replying. I could acquire new point of view. I really appreciate it.

I got your point why has_key? is better than key?.

As First Last said,

> So hash.key? reads, "hash, are you a key?"

does make sense for me.

Through my thinking for some time, I came to the conclusion that has_key? is better than key? too

But next issue is, as Jeremy Evans said

> the backwards compatibility cost of deprecating then removing either has_key? or key? outweighs the benefit of removing one and standardizing on the other.

In my opinion, removing one and standardizing on the other at Ruby2.2.x is painful.

But it's better to do that in the future, so we start to deprecate either one.

Thank you. gogo.

#7 - 09/14/2014 04:15 AM - sawa (Tsuyoshi Sawada)

has_key? is one of the few exceptions to the naming convention in that it uses the third person singular present form of a verb. To keep the method names as general as possible, it is better to use the most general form of a verb and avoid third person singular present (which is what Matz has done besides the exceptions). In that respect, it is much better if has_key? is deprecated than key? is deprecated.

First Last, given hashes like the following, I don't understand how has_key? is better "in the interest of maximizing the English readability and sensibility" than key?.

```
I = {name: "Yamada", gender: "male"}
I.has_key?(:name)

you = {name: "Tanaka", gender: "female"}
you.has_key?(:gender)

we = {location: "Tokyo", company: "Foo corporation"}
we.has_key?(:location)
```

#8 - 09/14/2014 04:43 AM - rits (First Last)

Tsuyoshi Sawada wrote:

> I don't understand how has_key? is better

I already explained it, all I can do is restate it.

When the predicate consists of a just single noun or adjective, there is only enough information in it to ask one type of question (Are you?).

Are you a {noun}? (e.g. proc.lambda? => "proc, are you a lambda?")
Are you an {adjective}? (e.g. array.empty? => "array, are you empty?")

It can't mean anything whatsoever (have you?, should you?, countless other inquiries), because then the predicate name would be inscrutable, which is the opposite of the purpose of naming.

If you want to ask something other than "Are you?" there needs to be more information in the method name (e.g. has_key?)

That the conjugation can be off is nothing, next to not knowing what question is being asked.

#9 - 09/14/2014 05:00 AM - sawa (Tsuyoshi Sawada)

First Last wrote:

> That the conjugation can be off is nothing, next to not knowing what question is being asked.

Then, it is not "in the interest of maximizing the English readability and sensibility." It is a matter of semantics.

#10 - 09/14/2014 06:29 PM - mame (Yusuke Endoh)

I guess the culture gap (not language gap) is leading to imbroglio in this case.

http://en.wikipedia.org/wiki/High-_and_low-context_cultures

Many Japanese people feel comfortable with "hash.key?(:foo)", because they are in a higher-context culture.

--
Yusuke Endoh mame@ruby-lang.org

#11 - 09/15/2014 07:38 AM - spatulasnout (B Kelly)

Salutations,

First Last wrote:

> Tsuyoshi Sawada wrote:
>
> > I don't understand how has_key? is better
>
> I already explained it, all I can do is restate it.
>
> When the predicate consists of a just single noun or adjective, there is only enough information in it to ask one type of question (Are you?).

I wonder if this might be a circumstance where the Unix
philosophy of preferring shorter names for frequently used
operations might apply.

#key? must be presumably one of the most common queries to
Hash.

Regards,

Bill

#12 - 09/19/2014 06:01 PM - avit (Andrew Vit)

+1 for has_key? from me.

It's more readable as natural English as pointed out earlier.

has_key? is more clear because key(value) and key?(key) expect different arguments, so the names should be more clearly different. There is a common ruby idiom for x? == !!x as paired method names. In this case these are not at all similar.

has_key? is less likely to conflict with other objects that define dynamic predicate methods, which is better for duck-typing dynamic model.x? objects when wrapping it with method_missing for example. (OpenStruct, ActiveModel, ActiveSupport::StringInquirer are just a few examples.) This common use of predicate methods also has zero arity which is different from key? / has_key?.

Files

Replace_Hash#has_something.PATCH	20.4 KB	08/27/2014	gogotanaka (Kazuki Tanaka)