Ruby trunk - Feature #7797

Hash should be renamed to StrictHash and a new Hash should be created to behave like AS HashWithIndifferentAccess

02/07/2013 08:19 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Status:	Rejected
Priority:	Normal
Assignee:	matz (Yukihiro Matsumoto)
Target version:	Next Major

Description

Since #7792 has been rejected (although I don't really understand the reason except people being afraid of changing I guess) I'd like to propose an alternative solution to most of the problems caused by the differences between symbols and strings.

From my previous experience, most of the time I'm accessing a hash, I'd prefer that it behaved like HashWithIndifferentAccess (HWIA from now) from active_support gem.

Transforming all possible hashes in some object to HWIA is not only boring to do code but also time consuming.

Instead, I propose that {}.class == Hash, with Hash being implemented as HWIA and the current Hash implementation renamed to StrictHash.

That way, this should work:

```
a = {a: 1, 'b' => 2}
a[:a] == a['a'] && a['b'] == a[:b]
```

I don't really see any real use case where people really want to have a hash like this:

```
h = {a: 1, 'a' => 2}
```

This would only confuse people.

It also avoids confusion when parsing/unparsing from popular serialization formats, like JSON:

currently:

```
h = {a: 1}
j = JSON.unparse h
h2 = JSON.parse j
h[:a] != h2[:a]
```

With the new proposition (I'm assuming JSON should use Hash instead of StrictHash when parsing) h[:a] == h2[:a].

This is just a small example but most real-world usage for hashes would benefit from regular hashes behaving like HWIA.

Related issues:

Has duplicate Ruby trunk - Feature #8939: symbol / string invariance (for has...	**Closed**	**09/23/2013**

History

#1 - 02/07/2013 10:13 PM - trans (Thomas Sawyer)

I offer a slight modification that would ease the transition somewhat. Instead of renaming Hash, just add a new code class, e.g. Map or Index, that does as you suggest. Then at some major version change (3.0?) use {} -> Map, instead of {} -> Hash. This would give developers plenty of time to switch from {} to Hash.new where it would be necessary to do so.

Of course, the odds of this happening, despite the fact that it would improve the language by making it more intuitive in precisely the manner you describe, is asymptote to zilch.

#2 - 02/07/2013 11:39 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Thomas, the reason I didn't suggest your approach is because it wouldn't really fix several use cases, such as the example pointed out in the description using the json gem. It will use rb_hash_new for instance:

https://github.com/flori/json/blob/master/ext/json/ext/parser/parser.c#L112

#3 - 02/07/2013 11:40 PM - shevegen (Robert A. Heiler)

> Since #7792 has been rejected (although I don't really understand the
> reason except people being afraid of changing I guess)

Several reasons were given to you. I do not understand why you try to
"summarize" them all by stating that "people are afraid of change".

Symbols and Strings are not the same internally.

If ruby would get rid of symbols, the language would be simnpler for
newcomers. But why have symbols been exposed at the ruby-language
level at all? That's right, because they are simply more efficient
than Strings. And that has nothing to do with anyone being afraid
of change.

What you basically tried in your earlier proposal was to change
ruby to a completely different language, where Symbols would be
gone. This is a huge conceptual change, an undertaking that would
even interfere with major release versions. It would break old
ruby code for only marginal gains. How could you state that
people are "afraid of change" and this is the sole reason for
the rejection of your proposal?

> h = {a: 1, 'a' => 2} This would only confuse people.

Yes, this crap would confuse people.

I myself use the old syntax, and it is also the one
ruby uses internally. You can see this by pasting
the hash there into IRB - you will see that a: 1
will become :a => 1.

I myself would rather like if this new syntax would
not have been added at all. I dont like it, and even
though you save a few keys, I think it adds more to
confusion rather than make things really better.

I do agree with you that it is annoying that symbols
and strings can be used for hash keys. When I write
ruby code, I always have to ask myself whether I
want to use a symbol or a string as key.

I do not however see how your proposal makes THIS
very decision any simpler.

#4 - 02/08/2013 12:03 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

shevegen (markus heiler) wrote:

> > Since #7792 has been rejected (although I don't really understand the
> > reason except people being afraid of changing I guess)
>
> Several reasons were given to you. I do not understand why you try to
> "summarize" them all by stating that "people are afraid of change".

The only real reason I remember people mentioning is the performance thing and how they are different internally. It shouldn't really matter to the language design its implementation details (if they are different internally or not, for instance). And for the performance argument I really believe symbols cause more harm than good to the overall performance given all conversions that it will require in most code bases (even if you don't do that directly it is likely that you rely on some gem that will do that).

> If ruby would get rid of symbols, the language would be simnpler for
> newcomers. But why have symbols been exposed at the ruby-language
> level at all?

Have no idea! My suspect is that someone thought: "hey maybe it would be a great idea if we could optimize some constant strings - we could create a symbol for that - hey, look at my micro-benchmarks, it really is a great idea!".

> That's right, because they are simply more efficient than Strings.

On micro-benchmarks I agree they may do some small difference. On real world-case apps though I'd suspect its presence is actually causing code to perform worse.

> What you basically tried in your earlier proposal was to change
> ruby to a completely different language...

Why do you think that removing symbols would change Ruby in a fundamental way? I believe 99.9% of all Ruby code would still work without any changes if :symbol was the same as 'symbol'.freeze.

> This is a huge conceptual change

This is your opinion. I don't agree.

> ..I myself would rather like if this new syntax would
> not have been added at all. I dont like it, and even
> though you save a few keys

The problem is not saving keys for my particular case. It is pretty natural to me when I'm typing to find the comma (:). Greater than signals (>) and, specially the equals (=) symbol require much more effort, which makes a big difference when you're creating a hash with many keys. Those two symbols are also very far from each other.

> I do agree with you that it is annoying that symbols
> and strings can be used for hash keys. When I write
> ruby code, I always have to ask myself whether I
> want to use a symbol or a string as key.

That is exactly the reason why I created this ticket.

> I do not however see how your proposal makes THIS
> very decision any simpler.

Simple, you don't have to worry about it anymore. Just don't think. If you've typed a symbol, that's fine. A string instead? No problem. Both should work interchangeable:

h = {a: 1, 'b' => 1}
h['a'] == h[:b] # == 1

#5 - 02/08/2013 12:53 AM - yorickpeterse (Yorick Peterse)

> The only real reason I remember people mentioning is the performance
> thing and how they are different internally. It shouldn't really
> matter to the language design its implementation details (if they are
> different internally or not, for instance).

There are so many cases where the implementation of X does affect its
presentation. The way your database tables are layed out will ultimately
affect your views. Is this bad too? No, it's a fundamental part of how
things work.

While in many cases you can work around that and come up with a language
design that isn't too heavily influenced by its implementation it would
in this case fundamentally change the way MRI works. On top of that it
would mean that all the other Ruby implementations such as JRuby and
Rubinius had to be modified as well to be compatible with the Ruby
specification.

I think the latter is something a lot of people don't think about. A
change this big doesn't just affect the people using MRI, it affects
every developer and every implementation out there. It will also affect
existing resources such as books and tutorials as basically everything
has to be re-written from scratch. The amount of effort required for
this is way too much for it to make a change that can be solved in many
different (and much easier) ways.

> And for the performance argument I really believe symbols cause more
> harm than good to the overall performance given all conversions that
> it will require in most code bases (even if you don't do that directly
> it is likely that you rely on some gem that will do that).

I'd be interested in seeing the actual numbers for this. I also find it interesting to see that your general opinion seems to be "the performance overhead of this change is not important" yet you do seem to care about the overhead of converting Symbols to Strings. I find this a rather contradicting opinion but maybe I'm misunderstanding things.

> Have no idea! My suspect is that someone thought: "hey maybe it would
> be a great idea if we could optimize some constant strings - we could
> create a symbol for that - hey, look at my micro-benchmarks, it really
> is a great idea!".

Yes, Matz took some crystal meth one evening and decided to introduce Symbols because he thought it was a fun idea to (apparently) piss people off with it.

If you honestly believe choices like these were made randomly or because of some hypothetical micro benchmark I suggest you actually spend some time getting to know the MRI code base. As with many things getting to know the inner workings of something will help you understand why choices were made, in this case I think it can greatly help you clear up your mind about making such a big change as well as understanding as to why it's not going to happen any time soon.

> On micro-benchmarks I agree they may do some small difference. On real
> world-case apps though I'd suspect its presence is actually causing
> code to perform worse.

Based on what? The conversion of a Symbol to a String is no more in-efficient than creating a String yourself. Again show a benchmark if you're so convinced this is not the case.

> Why do you think that removing symbols would change Ruby in a
> fundamental way? I believe 99.9% of all Ruby code would still work
> without any changes if :symbol was the same as 'symbol'.freeze.

See my comment above about all the implementations and resources having to be changed. It would also require substantial changes of the MRI source code. This would be a non trivial change.

> This is your opinion. I don't agree.

Again take a look at the MRI code base before you disagree on this matter.

On another note, you also seem to be unaware that it's not trivial to suddenly replace the usage of Hash with LazyDeveloperHash. Yes, in the Ruby source code it might be easier but at least in the C code it's going to take time and effort for something that has little value.

In your other threads I gave you various examples on how you can work around your issue and even do so in such a way that will save you a lot of future work. However, it seems you're hell bent on ignoring that and continue to argue about why you disagree with something you (at least from my point of view) don't have a good understanding of.

This is also going to be my last Email for these 3 (or 4 now?) discussions. I've given plenty of examples and explanations (and so have others) and I have no interest anymore in trying to explain it.

Yorick

#6 - 02/08/2013 03:23 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

Em 07-02-2013 13:51, Yorick Peterse escreveu:

> ...
>
> > And for the performance argument I really believe symbols cause more
> > harm than good to the overall performance given all conversions that
> > it will require in most code bases (even if you don't do that directly
> > it is likely that you rely on some gem that will do that).
>
> I'd be interested in seeing the actual numbers for this. I also find it
> interesting to see that your general opinion seems to be "the
> performance overhead of this change is not important" yet you do seem to
> care about the overhead of converting Symbols to Strings. I find this a
> rather contradicting opinion but maybe I'm misunderstanding things.

https://gist.github.com/rosenfeld/4732737

So, if your program relies on HWIA behavior and is converting regular
hashes into HWIA, it would perform much worse then if the hash syntax
already created HWIA. Also libraries like Sequel will call to_sym on
string values returned from the database as column names. Then the
overall performance would be slower due to unnecessary conversions
between symbols and strings that wouldn't be required if regular hashes
behaved like HWIA.

And this is just a simplistic case. What if I had to scan an entire
random object resulted from a call to JSON.parse and find all hashes to
manually replace them by a HWIA version?

This is way more common than you might expect in lots of gems and code
out there. That is why I believe that the overall performance of most
Ruby code in the real-world industry have their performance degraded by
the existence of symbols and by them returning false when compared to
equivalent strings.

> ...
> On another note, you also seem to be unaware that it's not trivial to
> suddenly replace the usage of Hash with LazyDeveloperHash. Yes, in the
> Ruby source code it might be easier but at least in the C code it's
> going to take time and effort for something that has little value.

You're saying that it has little value. What if someone decides that he
would like to spend his (not yours) time making the required changes in
the C code? Would this problem be gone?

> In your other threads I gave you various examples on how you can work
> around your issue

You didn't understand the issue. I showed you a real example where you
could show me how to optimize it but you haven't done so:

https://bugs.ruby-lang.org/issues/7792#note-23

You just assumed that Sequel took the wrong decision while deciding for
using symbols as keys instead of strings. Jeremy Evans (Sequel's
maintainer) doesn't seem to support your opinion on that and explains
why he thinks Sequel should keep using symbols as keys.

But this is a real gem used in a real-world application published in the
web. And this application would not only be easier to write and maintain
if hash syntax created HWIA but this app would also perform better.

#7 - 02/08/2013 08:23 AM - spatulasnout (B Kelly)

rosenfeld (Rodrigo Rosenfeld Rosas) wrote:

> Have no idea! My suspect is that someone thought: "hey maybe it would
> be a great idea if we could optimize some constant strings - we could
> create a symbol for that - hey, look at my micro-benchmarks, it really
> is a great idea!".

Ruby's symbols hail directly from Smalltalk (early 1970's) and are analogous to Lisp atoms (early 1960's.)

I think the challenge in Ruby is there are multiple competing reasons symbols
are used, from a developer perspective:

- performance (A symbol exists once in memory and can be compared quickly)
- immutability (Symbols implicitly behave as though 'frozen')
- appealing syntax (If symbols were uglier to create than strings, instead
  of being totally sexy, we likely wouldn't be having this
  discussion.)

Presumably many Rubyists gravitate toward the appealing syntax, without
being aware of Symbol's other properties -- and, well, the accompanying half-
century of computing history?

Personally I'd love for symbols to be able to be garbage collected, though I
understand the technical challenges. I'd be happy if a symbol-string
"indifferent" Hash alternative were available in core (is there a better name
than the Rails thing?) If we were starting over at Ruby 1.0, I probably would
be fine with the :symbol syntax just being an alternative way to create
regular strings. Today, though, symbols are part of the language, and they're
distinct from strings. I'm OK with that.

Regards,

Bill

#8 - 09/23/2013 07:27 PM - Sing9898 (Sing Lou)

I am so glad this ticket wasn't rejected because it is a constructive solution to one of the biggest Ruby frustrations. (Even if you understand the difference between strings and symbol it still causes many hidden bugs).

#9 - 09/23/2013 07:35 PM - Sing9898 (Sing Lou)

the confusing became probably more immediate after the introduction of the new {a:"b"} syntax.
because for novices it might not be clear whether {a:"b"} means {"a"=>"b"} or {:a =>"b"}

#10 - 09/23/2013 07:43 PM - Hanmac (Hans Mackowiak)

Sing9898 (Sing Lou) no you cant do that because it could break existing ruby code

{a:"b"} is the short form for {:a => "b"}, you can not easily change it to {"a"=>"b"} without breaking multiple ruby programs

i think you still does not understand that Symbols are sometimes better than Strings

#11 - 06/26/2014 12:02 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

- File feature-7797.pdf added

Attached an slide for this proposal

#12 - 06/30/2014 04:42 AM - naruse (Yui NARUSE)

received, thanks!

#13 - 06/30/2014 02:25 PM - hsbt (Hiroshi SHIBATA)

Hi Rodrigo,

Cur redmine couldn't handle your file named feature-7797.pdf.
Please remove & upload it again.

#14 - 06/30/2014 05:05 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

- File feature-7797.pdf added

- File redmine-bug.jpg added

I don't think I have permissions to remove it. I'm reattaching it.

But indeed Redmine is buggy here on Chrome when I try to attach a PDF. See attached screenshot (It attached the PDF twice and I removed the duplicate)

#15 - 06/30/2014 05:11 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Chrome reported this in the console: "Uncaught RangeError: Maximum call stack size exceeded."

The stack goes something like this:

```
v.extend.clone jquery-1.8.3-ui-1.9.2-ujs-2.0.3.js?1402908441:2
(anonymous function) jquery-1.8.3-ui-1.9.2-ujs-2.0.3.js?1402908441:2
(anonymous function) jquery-1.8.3-ui-1.9.2-ujs-2.0.3.js?1402908441:2
v.extend.map jquery-1.8.3-ui-1.9.2-ujs-2.0.3.js?1402908441:2
v.fn.v.map jquery-1.8.3-ui-1.9.2-ujs-2.0.3.js?1402908441:2
v.fn.extend.clone jquery-1.8.3-ui-1.9.2-ujs-2.0.3.js?1402908441:2
addInputFiles attachments.js?1402908441:118
onchange 7797:1
v.extend.clone jquery-1.8.3-ui-1.9.2-ujs-2.0.3.js?1402908441:2
...
```

#16 - 06/30/2014 05:14 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

- File feature-7797.pdf added

Trying again, using Firefox now.

#17 - 07/26/2014 04:47 AM - naruse (Yui NARUSE)

- File deleted (feature-7797.pdf)

#18 - 07/26/2014 04:47 AM - naruse (Yui NARUSE)

- File deleted (feature-7797.pdf)

#19 - 07/26/2014 04:51 AM - matz (Yukihiro Matsumoto)

- Status changed from Open to Rejected

Hi,

I have to reject this issue, since this will introduce a huge incompatibility. Even I don't have right to break thousands of programs that use both symbols and strings in keys.

Another idea is introducing mode to make a hash indifferent, but introducing mode is kinda against recent trend of immutability.

Matz.

#20 - 07/26/2014 05:10 AM - matz (Yukihiro Matsumoto)

Note that we alrady have Hash#compare_by_identity.

Matz.

#21 - 07/28/2014 01:15 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

What do you mean by introducing a mode to make a hash indifferent? You mean another method changing how the Hash works, just like compare_by_identity? Or do you mean some Ruby command-line flag option? I wouldn't like the latter but maybe the former could work fine.

If you were talking about the former would you mind expanding on why such change would be relevant to the immutability thing? I don't really think worrying about immutability makes much sense in Ruby. Languages that want to get the performance benefits from immutable constructs usually do that from the very beginning since it's very hard to try to introduce this concept later in a general purpose OO language...

Immutability is not free and giving up of some features because of immutability specially when the lack of the features won't give you any immediate advantages on performance doesn't make much sense to me.

I don't see immutability as the single path to meet performance or parallelism. C++, Java and other languages can perform pretty well with mutable structs. I do think MRI should be more concerned about removing the lock and allow multiple threads to run Ruby code at the same time like other Ruby VMs do. This makes more sense to me than trying to move Ruby to a direction where immutability would be the main goal and then prevent some useful features from being implemented instead.

Files

redmine-bug.jpg	107 KB	06/30/2014	rosenfeld (Rodrigo Rosenfeld Rosas)
feature-7797.pdf	29.3 KB	06/30/2014	rosenfeld (Rodrigo Rosenfeld Rosas)