

Ruby - Feature #7839

Symbol.freeze_symbols

02/13/2013 06:37 AM - tenderlovmaking (Aaron Patterson)

Status:	Rejected	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:	2.6	
Description		
Hi,		
On team Rails, we're having troubles with Symbol creation DoS attacks. From our perspective, there should be a point in the application where symbols should stabilize, meaning we don't expect the number of symbols to increase while the process is running.		
I'd like to be able to call a method like Symbol.freeze_symbols which would essentially freeze the symbol hash, such that if any new symbols are created, an exception would be thrown.		
I can work on a patch for this, but I wanted to throw the idea out there.		
Related issues:		
Related to Ruby - Feature #7854: New method Symbol[string]		Closed 02/15/2013

History

#1 - 02/13/2013 08:56 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

I really believe that letting the symbols to be garbage collected ([#7791](#)) is the way to go here.

You have mentioned that changing YAML#load to be safe by default would break existing apps and now you suggest that it would be safe to freeze symbols at some point as if it wouldn't break existing apps relying on marshalling behavior of Ruby libraries. Could you please explain better why do you think it is ok to freeze symbols in Rails and break existing apps but it is not ok to make YAML#load safe (preferring a new safe_load method instead)?

#2 - 02/13/2013 09:36 AM - phluid61 (Matthew Kerwin)

rosenfeld (Rodrigo Rosenfeld Rosas) wrote:

Could you please explain better why do you think it is ok to freeze symbols in Rails and break existing apps but it is not ok to make YAML#load safe (preferring a new safe_load method instead)?

Users can choose whether or not to call Symbol.freeze_symbols, and 100% of current apps do not call it, so those apps will continue to function exactly as they always have until they are updated. Similary, they can choose to update their apps to use YAML#safe_load instead of #load.

Changing the behaviour of YAML#load would change the behaviour of existing apps without any opt-in from their maintainers.

#3 - 02/13/2013 09:55 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

yeah, but if Rails calls Symbol.freeze_symbols it could break all Rails applications relying on YAML#load as an unmarshall method. Are you suggesting that Rails shouldn't care about breaking existing Rails apps but that Ruby should care about breaking existing Ruby apps?

#4 - 02/13/2013 10:23 AM - normalperson (Eric Wong)

"tenderlovmaking (Aaron Patterson)" aaron@tenderlovmaking.com wrote:

I'd like to be able to call a method like Symbol.freeze_symbols which would essentially freeze the symbol hash, such that if any new symbols are created, an exception would be thrown.

How about the option to do a soft freeze which issues a warning instead of exception? (but support both). Start using soft freeze, and move to a real freeze later when apps/gems are fixed.

#5 - 02/13/2013 10:25 AM - phluid61 (Matthew Kerwin)

rosenfeld (Rodrigo Rosenfeld Rosas) wrote:

yeah, but if Rails calls Symbol.freeze_symbols it could break all Rails applications relying on YAML#load as an unmarshall method. Are you

suggesting that Rails shouldn't care about breaking existing Rails apps but that Ruby should care about breaking existing Ruby apps?

Nobody said rails would call it. How would rails know when your particular app's symbols have stabilised? The biggest automation I'd have expected would be that they provide a trigger hook, so you can instruct rails/ruby to freeze the table when some condition is met. If they were going to add an automagic trigger at, for example, "all classes loaded" (whatever that means), surely it would be an opt-in configuration-driven behaviour.

#6 - 02/13/2013 10:39 AM - phluid61 (Matthew Kerwin)

normalperson (Eric Wong) wrote:

"tenderlovmaking (Aaron Patterson)" aaron@tenderlovmaking.com wrote:

I'd like to be able to call a method like `Symbol.freeze_symbols` which would essentially freeze the symbol hash, such that if any new symbols are created, an exception would be thrown.

How about the option to do a soft freeze which issues a warning instead of exception? (but support both). Start using soft freeze, and move to a real freeze later when apps/gems are fixed.

Also, would you expect to be able to thaw it out again? It might be enough in the short term to, e.g.

```
begin
Symbol.freeze_symbols
YAML.load(...)
ensure
Symbol.thaw_symbols
end
```

#7 - 02/13/2013 11:16 AM - shugo (Shugo Maeda)

phluid61 (Matthew Kerwin) wrote:

Also, would you expect to be able to thaw it out again? It might be enough in the short term to, e.g.

```
begin
Symbol.freeze_symbols
YAML.load(...)
ensure
Symbol.thaw_symbols
end
```

If this is a main use case of `Symbol.freeze_symbols`, it might be better to have `String#intern`'s option to control whether a symbol creation is allowed and to make `YAML.safe_load` to use it.

```
:foo
"foo".intern #=> :foo
"bar".intern #=> :bar
"foo".intern(allow_new: false) #=> :foo
"bar".intern(allow_new: false) #=> error
```

I guess it can be implemented easily compared to `Symbol GC`.

#8 - 02/13/2013 12:23 PM - Anonymous

On Wed, Feb 13, 2013 at 11:16:31AM +0900, shugo (Shugo Maeda) wrote:

Issue [#7839](#) has been updated by shugo (Shugo Maeda).

phluid61 (Matthew Kerwin) wrote:

Also, would you expect to be able to thaw it out again? It might be enough in the short term to, e.g.

```
begin
Symbol.freeze_symbols
YAML.load(...)
ensure
Symbol.thaw_symbols
end
```

I think having a freeze and thaw would be fine.

If this is a main use case of `Symbol.freeze_symbols`, it might be better to have `String#intern`'s option to control whether a symbol creation is allowed and to make `YAML.safe_load` to use it.

```
:foo
"foo".intern #=> :foo
"bar".intern #=> :bar
"foo".intern(allow_new: false) #=> :foo
"bar".intern(allow_new: false) #=> error
```

The problem with this is we can be calling foreign code. We have to force all library authors to use it. Library authors may not expect that calls to `".intern"` will be fed user input.

Most security issues we have to deal with (even the YAML example) are cases where we do not expect to process foreign input.

I guess it can be implemented easily compared to Symbol GC.

Even freezing and thawing would definitely be easier than Symbol GC.

--

Aaron Patterson

<http://tenderlovmaking.com/>

#9 - 02/13/2013 01:10 PM - shugo (Shugo Maeda)

If this is a main use case of `Symbol.freeze_symbols`, it might be better to have `String#intern`'s option to control whether a symbol creation is allowed and to make `YAML.safe_load` to use it.

(snip)

The problem with this is we can be calling foreign code. We have to force all library authors to use it. Library authors may not expect that calls to `".intern"` will be fed user input.

Most security issues we have to deal with (even the YAML example) are cases where we do not expect to process foreign input.

Hmm.... I'm worried that the following code is not thread safe.

```
begin
  Symbol.freeze_symbols
  YAML.load(...)
ensure
  Symbol.thaw_symbols
end
```

I guess it can be implemented easily compared to Symbol GC.

Even freezing and thawing would definitely be easier than Symbol GC.

Sasada-san might implement Symbol GC for MRI.

So I'm worried that he might get less popular with women.

If it can be implemented in other implementations, it would be the best solution.

#10 - 02/13/2013 02:15 PM - Student (Nathan Zook)

Creating global state has global implications. Yes, there are threading implications. Yes, there are issues with called libraries. But what really annoys me is yet again some software providers (again from the Rails team) that thinks he understands everything that users of his software might do.

I'm NOT just talking about poorly designed code. Honestly, I think that normal Rails behaviour is not expected to match this. I know that dynamic finders are supposed to be 'passthrough', but if Rails continues to implement ANY dynamic method generation that happens at runtime, then there is no way to know when the particular piece of code that does the generation gets called. In fact, this entire conversation got started because the Rails

framework intentionally generated symbols during runtime. That's not to mention (Rails) application code doing so for perfectly reasonable conditions.

I very much favour making the symbol table queriable. ([#7795](#)) Let `YAML.safe_load` check it. Let `to_json(:safe => true)` check it. But don't break existing code simply because you think your global solution is going to make life good for everyone.

#11 - 02/13/2013 05:14 PM - alexeymuranov (Alexey Muranov)

Sorry about a naïve idea, but what would you say about prohibiting interning tainted strings instead?

#12 - 02/13/2013 06:30 PM - yhara (Yutaka HARA)

- *Category set to core*

- *Target version set to 2.6*

#13 - 02/14/2013 09:00 AM - Student (Nathan Zook)

+1 to prohibiting interning of tainted strings. Probably `$SAFE >= 1`, though. Currently, this is a problem for `$SAFE <= 2`.

#14 - 02/14/2013 09:53 AM - Anonymous

On Wed, Feb 13, 2013 at 05:15:25PM +0900, alexeymuranov (Alexey Muranov) wrote:

Issue [#7839](#) has been updated by alexeymuranov (Alexey Muranov).

Sorry about a naïve idea, but what would you say about prohibiting interning tainted strings instead?

It seems like a good idea, except that tainting is not reliable. For example:

```
irb(main):001:0> require 'json'
=> true
irb(main):002:0> user_input = '{"foo":"bar"}'.taint
=> '{"foo":"bar"}'
irb(main):003:0> user_input.tainted?
=> true
irb(main):004:0> params = JSON.parse user_input
=> {"foo"=>"bar"}
irb(main):005:0> params['foo'].tainted?
=> false
```

In this case, a tainted string becomes untainted. I guess it's OK for tainted strings to become untainted, but how is someone supposed to verify if the JSON is safe without first parsing the JSON before sending the JSON to the JSON parser? My head is spinning.

C extension authors need to use `rb_tainted_str*`, but most of them just use `rb_str_new`. You can increase `$SAFE` high enough that all objects are tainted on creation, but nobody uses `$SAFE` (I think MRI is the only implementation of Ruby that has it).

In short: :(

--

Aaron Patterson

<http://tenderlovmaking.com/>

#15 - 02/14/2013 09:53 AM - Anonymous

On Wed, Feb 13, 2013 at 01:10:24PM +0900, shugo (Shugo Maeda) wrote:

Issue [#7839](#) has been updated by shugo (Shugo Maeda).

If this is a main use case of `Symbol.freeze_symbols`, it might be better to have `String#intern`'s option to control whether a symbol creation is allowed and to make `YAML.safe_load` to use it.
(snip)

The problem with this is we can be calling foreign code. We have to force all library authors to use it. Library authors may not expect that calls to `".intern"` will be fed user input.

Most security issues we have to deal with (even the YAML example) are cases where we do not expect to process foreign input.

Hmm.... I'm worried that the following code is not thread safe.

```
begin
  Symbol.freeze_symbols
  YAML.load(...)
ensure
  Symbol.thaw_symbols
end
```

Yes, this is probably not thread safe, but if people *know* it's not thread safe, then they can lock:

```
begin
  Symbol.lock
  Symbol.freeze_symbols
  YAML.load(...)
ensure
  Symbol.thaw_symbols
  Symbol.unlock
end
```

I guess it can be implemented easily compared to Symbol GC.

Even freezing and thawing would definitely be easier than Symbol GC.

Sasada-san might implement Symbol GC for MRI.

So I'm worried that he might get less popular with women.

If it can be implemented in other implementations, it would be the best solution.

I think so too. If Symbol GC turns out to be too hard though, I'd like to explore this option! :-)

--

Aaron Patterson

<http://tenderlovmaking.com/>

#16 - 02/14/2013 09:53 AM - Anonymous

On Wed, Feb 13, 2013 at 10:08:28AM +0900, Eric Wong wrote:

"tenderlovmaking (Aaron Patterson)" aaron@tenderlovmaking.com wrote:

I'd like to be able to call a method like Symbol.freeze_symbols which would essentially freeze the symbol hash, such that if any new symbols are created, an exception would be thrown.

How about the option to do a soft freeze which issues a warning instead of exception? (but support both). Start using soft freeze, and move to a real freeze later when apps/gems are fixed.

Even having a warning would be nice!

--

Aaron Patterson

<http://tenderlovmaking.com/>

#17 - 02/14/2013 09:53 AM - Anonymous

On Thursday, 14 February 2013 at 11:30 AM, Aaron Patterson wrote:

Yes, this is probably not thread safe, but if people *know* it's not thread safe, then they can lock:

```
begin
Symbol.lock
Symbol.freeze_symbols
YAML.load(...)
ensure
Symbol.thaw_symbols
Symbol.unlock
end
```

I think this is a perfect contender for a block. There can be a global 'symbol_freeze_count' that is incremented when the block enters, and decremented on the way out.

For example:

```
Symbol.frozen do
:foo # ok because this is symbol-ified by the parser beforehand
eval ":foo" # exception
"foo".to_sym # exception
end
```

This is better because it is re-entrant, unlike a pair of 'freeze' and 'thaw' methods.

#18 - 02/15/2013 08:58 AM - Student (Nathan Zook)

Would my proposal [#7854](#) not allow an easy and clean solution to this problem?

#19 - 02/15/2013 10:40 AM - shugo (Shugo Maeda)

Student (Nathan Zook) wrote:

+1 to prohibiting interning of tainted strings. Probably \$SAFE >= 1, though. Currently, this is a problem for \$SAFE <= 2.

\$SAFE is not implemented in other implementations, so this issue should be addressed without \$SAFE.

#20 - 02/18/2013 09:19 AM - ko1 (Koichi Sasada)

- Assignee set to matz (Yukihiro Matsumoto)

#21 - 02/25/2014 03:03 PM - naruse (Yui NARUSE)

- Related to Feature #7854: New method Symbol[string] added

#22 - 02/25/2014 03:16 PM - matz (Yukihiro Matsumoto)

- Status changed from Open to Rejected

Although I agree that we need something to prevent Symbol DoS attack, #freeze_symbols has too destructive I think. I think that something should be a variation of #intern that refuses addition to the symbol table.

Matz.