Ruby - Feature #7791

Let symbols be garbage collected

02/06/2013 10:37 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Status:	Closed	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:	2.6	

Description

Lots of Denial-of-Service security vulnerabilities exploited in Ruby programs rely on symbols not being collected by garbage collector.

Ideally I'd prefer symbols and strings to behave exactly the same being just alternate ways of writing strings but I'll let this to another ticket.

This one simply asks for symbols to be allowed to be garbage collected when low on memory. Maybe one could set up some up-limit memory constraints dedicated to storing symbols. That way, the most accessed symbols would remain in that memory region and the least used ones would be reclaimed when the memory for symbols is over and a new symbol is created.

Or you could just allow symbols to be garbage collected any time. Any reasons why this would be a bad idea? Any performance benchmark demonstrating how using symbols instead of strings would make a real-world software perform much better?

Currently I only see symbols slowing down processing because people don't want to worry about it and will often use something like ActiveSupport Hash#with_indifferent_access or some other method to convert a string to symbol or vice versa...

History

#1 - 02/06/2013 10:48 PM - shyouhei (Shyouhei Urabe)

- Status changed from Open to Feedback

@rosenfeld do you have any conceptual patch to implement it? The reason Symbols aren't GCed is simply because the committers (including matz) don't have any idea how.

#2 - 02/06/2013 10:57 PM - Anonymous

Adding to shyouhei's comment, Symbols are actually used for quite a specific purpose inside MRI. Because they are essentially integers with names, they are used as keys in method tables/ivar tables/etc. as they speed up lookup quite a lot (integer comparison is faster than string comparison)

I think there would be a significant amount of work required to make symbols garbage collectable.

#3 - 02/06/2013 10:58 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Sorry @shyouhei (Shyouhei Urabe), but I don't know Ruby's code base. But I believe I won't be able to understand how this could work if any of the core committers could get it to work :P

I didn't know this was the reason why they weren't collected by GC either. This is good to know! At least now I know that you're willing to accept any patch to make them GCed. :)

#4 - 02/06/2013 11:04 PM - trans (Thomas Sawyer)

Is it possible to set a maximum size to the number of symbols allowed? An error would be raised if the limit was hit. Would that at least suffice to stop DoS attacks?

#5 - 02/06/2013 11:09 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

There is already a limit (memory bounded) and the problems begin after the limit is reached. If you just limit the amount of symbols using another kind of limit you won't prevent DoS attacks. What would happen once the limit is reached in such case?

#6 - 02/06/2013 11:14 PM - trans (Thomas Sawyer)

An error would be raised.

#7 - 02/06/2013 11:23 PM - ko1 (Koichi Sasada)

(2013/02/06 22:50), shyouhei (Shyouhei Urabe) wrote:

> @rosenfeld do you have any conceptual patch to implement it? The reason Symbols aren't GCed is simply because the committers (including

> matz) don't have any idea how.

One rough idea (but not verified) is:

Separated Symbols into two sets:
(a) Symbols created by rb_intern()
(b) Symbols created from String object (String#to_sym)

(a) is internal symbols which are used by the interpreter such as method
names, attribute names and so on.

(b) is mainly created by ruby program (and used for DoS attack).

I think (hope) (b) can be collected at GC timing with some development
efforts. But not touched. Sorry.

PS. Of course, a program making symbols belong to (a) will be DoS
attack. For example, the program makes many methods or attributes by
untrusted data, it will be same problem (but I believe nobody makes such
a bad program).

--
// SASADA Koichi at atdot dot net

#8 - 02/06/2013 11:23 PM - yorickpeterse (Yorick Peterse)

Garbage collecting Symbols would most likely do more harm than good.
Sure, a ddos based on Symbols may no longer be possible but there's a
big chance it will introduce a significant performance penalty as a
result of having to create new Symbol instances every time a set of is
gabrage collected.

I also feel that garbage collecting symbols is the wrong way to solve
the problem. The problem is that the code you're using blindly converts
user based input to Symbols. The obvious solution to this problem would
be to simply stop doing that.

To give you an idea of how many Symbols there are by default you can run
the following:

```
  puts Symbol.all_symbols.size
```

For me this results in 1859. However, when running this in a Pry session
(just to give an idea of how big this number can be) this results in
6823.

Yorick

#9 - 02/06/2013 11:26 PM - shyouhei (Shyouhei Urabe)

rosenfeld (Rodrigo Rosenfeld Rosas) wrote:

> At least now I know that you're willing to accept any patch to make them GCed. :)

Yes. I promise I vote a big +1 for such thing. It cannot be a boring patch.

#10 - 02/06/2013 11:45 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

trans (Thomas Sawyer) wrote:

> An error would be raised.

So you would have a successful DoS attack, right? Because the attacker would cause an exception after the symbols limit has been reached. After that, any request relying on #to_sym of an inexistent symbol would result in an exception being raised. And it is not uncommon that some web frameworks will use #to_sym sometimes I guess. Or even some action that hasn't be called yet would no longer work after the DoS attack. For instance, suppose you use some symbol for i18n in some action that hasn't been called yet before the DoS attack.

#11 - 02/06/2013 11:53 PM - ko1 (Koichi Sasada)

(2013/02/06 23:21), SASADA Koichi wrote:

> One rough idea (but not verified) is:

> Separated Symbols into two sets:
> (a) Symbols created by rb_intern()
> (b) Symbols created from String object (String#to_sym)

Additional consideration about performance:

(a) is same as current performance.

(b) takes a time.
(b-1) Additional creation time (same as String creation and a bit)
(b-2) Slowdown GC time

I think normal programs doesn't create (b) (sorry I'm not sure about
ActiveSupport), so I think performance down doesn't affect people.

--
// SASADA Koichi at atdot dot net

#12 - 02/06/2013 11:53 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Em 06-02-2013 12:21, SASADA Koichi escreveu:

> (2013/02/06 22:50), shyouhei (Shyouhei Urabe) wrote:
>
>> @rosenfeld do you have any conceptual patch to implement it? The reason Symbols aren't GCed is simply because the committers (including matz) don't have any idea how.
>> One rough idea (but not verified) is:
>
> Separated Symbols into two sets:
> (a) Symbols created by rb_intern()
> (b) Symbols created from String object (String#to_sym)
>
> (a) is internal symbols which are used by the interpreter such as method
> names, attribute names and so on.
>
> (b) is mainly created by ruby program (and used for DoS attack).
>
> I think (hope) (b) can be collected at GC timing with some development
> efforts. But not touched. Sorry.

Your idea would work quite well I think.

> PS. Of course, a program making symbols belong to (a) will be DoS
> attack. For example, the program makes many methods or attributes by
> untrusted data, it will be same problem (but I believe nobody makes such
> a bad program).

If an attacker could create new methods than I believe that symbols not
being collected is the least problem ;)

#13 - 02/06/2013 11:53 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Em 06-02-2013 12:35, SASADA Koichi escreveu:

> (2013/02/06 23:21), SASADA Koichi wrote:
>
>> One rough idea (but not verified) is:
>>
>> Separated Symbols into two sets:
>> (a) Symbols created by rb_intern()
>> (b) Symbols created from String object (String#to_sym)
>> Additional consideration about performance:
>
> (a) is same as current performance.
>
> (b) takes a time.
> (b-1) Additional creation time (same as String creation and a bit)
> (b-2) Slowdown GC time
>
> I think normal programs doesn't create (b) (sorry I'm not sure about

> ActiveSupport), so I think performance down doesn't affect people.

I agree that to_sym is the big problem here. I don't think using to_sym
is usually considered a best practice but when you have to do that it is
unlikely that you're concerned about the performance hit, don't you agree?

#14 - 02/06/2013 11:53 PM - trans (Thomas Sawyer)

I'd don't think the problem is an error being raised. Just handle the exception. The issue is having a process eat up all the systems memory. Isn't it?

#15 - 02/07/2013 12:01 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

It depends. If the process eats up all system memory it will eventually crash and release all memory used by the Ruby process, right? This doesn't seem like a really big problem once the limit is reached and the application crashed.

But anyway, I believe that DoS attacks are usually targeted at applications. If you make your application always raise for some actions, then I'd say your DoS attack is successful.

#16 - 02/07/2013 12:12 AM - trans (Thomas Sawyer)

> But anyway, I believe that DoS attacks are usually targeted at applications. If you make your application always raise for some actions, then I'd say your DoS attack is successful.

That doesn't make any sense to me. Errors happen, you rescue them and send the user to an appropriate landing page. In this case you would send them to the "Don't DoS me, bitch!" page.

#17 - 02/07/2013 01:23 AM - now (Nikolai Weibull)

On Wed, Feb 6, 2013 at 2:37 PM, rosenfeld (Rodrigo Rosenfeld Rosas)
rr.rosas@gmail.com wrote:

> Lots of Denial-of-Service security vulnerabilities exploited in Ruby programs rely on symbols not being collected by garbage collector.

I'm out of the loop on this one. I'm assuming that this occurs when a
program creates a lot of symbols based on user input. If that's the
attack vector, shouldn't that be fixed instead of letting the GC
collect Symbols, which, as has already been stated, seems very hard to
do correctly, if at all?

#18 - 02/07/2013 01:29 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

Em 06-02-2013 14:00, Nikolai Weibull escreveu:

> On Wed, Feb 6, 2013 at 2:37 PM, rosenfeld (Rodrigo Rosenfeld Rosas)
> rr.rosas@gmail.com wrote:
>
>> Lots of Denial-of-Service security vulnerabilities exploited in Ruby programs rely on symbols not being collected by garbage collector.
>> I'm out of the loop on this one. I'm assuming that this occurs when a
>> program creates a lot of symbols based on user input. If that's the
>> attack vector, shouldn't that be fixed instead of letting the GC
>> collect Symbols, which, as has already been stated, seems very hard to
>> do correctly, if at all?

Nikolai, it is possible to fix the applications/frameworks against this
kind of attack, but people will keep finding new ways of doing that and
the fact that symbols do not get their memory reclaimed back makes some
decisions a bit complicate to decide against.

For instance, YAML#safe_load should allow restoring symbols? If symbols
are collected by the GC, then it is safe for safe_load to convert
symbols from YAML input. Otherwise, it is not safe and you wouldn't be
able to load symbols from YAML input using safe_load. Do you see?

#19 - 02/07/2013 01:32 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

trans (Thomas Sawyer) wrote:

>> But anyway, I believe that DoS attacks are usually targeted at applications. If you make your application always raise for some actions, then

> > I'd say your DoS attack is successful.
>
> That doesn't make any sense to me. Errors happen, you rescue them and send the user to an appropriate landing page. In this case you would send them to the "Don't DoS me, bitch!" page.

I think it would help if I could write a more detailed example. Suppose you have this controller:

class MyController
def my_action
render text: I18n.translate(:my_action_successful)
end
end

Suppose you boot your application and the DoS attack begins. Now you can't create any more symbols because trying to do so would raise an exception. Now suppose my_action has never been called on your application between the time it booted and the time the DoS attack began.

Your users will never be able to use the part of your application that depend on the my_action action because every try to make a call to that action would result in a 500 Internal Server Error.

What is your solution for fixing this problem?

#20 - 02/07/2013 01:53 AM - now (Nikolai Weibull)

On Wed, Feb 6, 2013 at 5:24 PM, Rodrigo Rosenfeld Rosas
rr.rosas@gmail.com wrote:

> Em 06-02-2013 14:00, Nikolai Weibull escreveu:
>
> Nikolai, it is possible to fix the applications/frameworks against this kind
> of attack, but people will keep finding new ways of doing that and the fact
> that symbols do not get their memory reclaimed back makes some decisions a
> bit complicate to decide against.

“Fixing” Symbols seems more complicated.

> For instance, YAML#safe_load should allow restoring symbols? If symbols are
> collected by the GC, then it is safe for safe_load to convert symbols from
> YAML input. Otherwise, it is not safe and you wouldn't be able to load
> symbols from YAML input using safe_load. Do you see?

I see you answering your own question.

#21 - 02/07/2013 02:02 AM - trans (Thomas Sawyer)

@resenfeld After the exception is raised, you can still create symbols. The point is, now you know somethings up and you need to deal with it. So once raised the idea is that you'd report the error (log and redirect user) and then shut down the process, effectively clearing the memory.

Admittedly I am thinking more in terms of short-running processes. If you are using a long-running process to serve many user requests then maybe that's a not as ideal here. But the basic idea remains. Handle the error and shut down. Your server should spin up a new process to take its place on demand. Added bonus, you can log the request IP, maybe if it happens a few times from the same address, you block it.

#22 - 02/07/2013 02:21 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

@trans, yes I'm talking about threaded servers as I really believe they are a better model than multi-process (maybe not with MRI, but in the broad sense and I still believe MRI will improve support for threads at some point).

If you're only concerned about web apps deployed using a multi-process approach, like Unicorn, then your solution would help a lot certainly.

But for applications (web or not) that stay alive and get input from untrusted data and don't fork this would still be an issue.

Anyway, suppose your action should be transactional. So you've made some changes in your controller and then you needed a symbol. I guess you won't create a begin-rescue block everytime you need a symbol, right? In this case you would interrupt what should be a transactional action in the middle of processing causing unexpected things to happen.

While it is possible to create transactions in most databases, not all of the actions happen in the database. Maybe you need to send an e-mail after saving some records to the database. Maybe you need a symbol for i18n translations to use in your e-mail. For such cases, raising an exception on reaching symbols limit isn't acceptable. Also, if you have to restart your server every few seconds while under a DoS attack by symbols exhaustation, then you are not really preventing the DoS attack, right?

#23 - 02/07/2013 03:07 AM - trans (Thomas Sawyer)

@rosenfeld I get your points. My solutions certainly is not perfect and has tough issues to deal with, certainly. But what alternative do you have really? You are under DoS attack, your whole server is about to go down! Better to restart the server, automate the blocking of the offending IP and carry on. The alternative right now is just crash and burn.

Also, how much is GCing symbols going to help anyway? DoS attacks usually involve multiple requests. If they start feeding your app symbols ad nauseum, then you server is just going spend most of its time throwing away symbols. It might not complete crash but it will run like mud.

But I am no expert in any of this. Which is why I originally posited it as a question.

#24 - 02/07/2013 03:56 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

@trans, if symbols are collected, instantiating many symbols wouldn't exhaust the memory so an attempt of a DoS attack by creating many symbols wouldn't succeed.

Of course someone could always overload your server with many requests but this is only one way of DoS. This doesn't relate to the other more common way of DoS attack being explored lately due to symbols not being garbage collected. This ticket only aims in fixing the latter case, not the former one.

The latter case only needs a few requests until the memory is over and after that your application would stop working. The former would affect your application only during the requests made by the attacker. It won't let your application down in most cases. In that case you could just block the offending IP as a first measure without having to restart your application and without losing any data (although the site may remain innaccessible to some users while the attack isn't stopped).

We might be talking about different types of DoS attacks...

#25 - 02/07/2013 04:28 AM - trans (Thomas Sawyer)

@rosenfeld Just had a thought... maybe we could combine these approaches. Maybe GCing Symbols is too difficult. But what if it were done manually? Would that be more doable? .e.g

begin
...
rescue SymbolExhaustion
purge_symbol_table(0.5) # 50%
retry
end

This would allow you to continue on, but also log the offence.

Of course, I say this not knowing what the underlying problem with GCing symbols is, so maybe manually doing is no better. As I said, just a thought.

#26 - 02/07/2013 07:23 AM - Anonymous

@trans

You cannot 'purge the symbol table'. If a symbol used by a method name, etc is purged then you will encounter strange problems like:

- not being able to call the method
- the wrong code gets invoked when you call a method
- Ruby crashes

Symbols are integral to the implementation of MRI. You cannot change how they work this easily.

ko1's approach is probably the best, but even then it still has a few issues.

For example, say you create a GC-able Symbol and pass it to send. That symbol will need to be interned, but then you need a way to let the Symbol object know that it now represents an internal ID (or else you break the property of Symbols being unique)

On 07/02/2013, at 6:28 AM, "trans (Thomas Sawyer)" transfire@gmail.com wrote:

> Issue #7791 has been updated by trans (Thomas Sawyer).
>
> @rosenfeld Just had a thought... maybe we could combine these approaches. Maybe GCing Symbols is too difficult. But what if it were done manually? Would that be more doable? .e.g
>
> begin
> ...
> rescue SymbolExhaustion
> purge_symbol_table(0.5) # 50%
> retry
> end
>
> This would allow you to continue on, but also log the offence.
>
> ## Of course, I say this not knowing what the underlying problem with GCing symbols is, so

maybe manually doing is no better. As I said, just a thought.

> Feature #7791: Let symbols be garbage collected
> https://bugs.ruby-lang.org/issues/7791#change-35943
>
> Author: rosenfeld (Rodrigo Rosenfeld Rosas)
> Status: Feedback
> Priority: Normal
> Assignee: matz (Yukihiro Matsumoto)
> Category: core
> Target version: next minor
>
> Lots of Denial-of-Service security vulnerabilities exploited in Ruby programs rely on symbols not being collected by garbage collector.
>
> Ideally I'd prefer symbols and strings to behave exactly the same being just alternate ways of writing strings but I'll let this to another ticket.
>
> This one simply asks for symbols to be allowed to be garbage collected when low on memory. Maybe one could set up some up-limit memory constraints dedicated to storing symbols. That way, the most accessed symbols would remain in that memory region and the least used ones would be reclaimed when the memory for symbols is over and a new symbol is created.
>
> Or you could just allow symbols to be garbage collected any time. Any reasons why this would be a bad idea? Any performance benchmark demonstrating how using symbols instead of strings would make a real-world software perform much better?
>
> Currently I only see symbols slowing down processing because people don't want to worry about it and will often use something like ActiveSupport Hash#with_indifferent_access or some other method to convert a string to symbol or vice versa...
>
> --
> http://bugs.ruby-lang.org/

#27 - 02/07/2013 11:59 AM - duerst (Martin Dürst)

On 2013/02/07 2:02, trans (Thomas Sawyer) wrote:

> Issue #7791 has been updated by trans (Thomas Sawyer).
>
> @resenfeld After the exception is raised, you can still create symbols. The point is, now you know somethings up and you need to deal with it. So once raised the idea is that you'd report the error (log and redirect user) and then shut down the process, effectively clearing the memory.
>
> Admittedly I am thinking more in terms of short-running processes. If you are using a long-running process to serve many user requests then maybe that's a not as ideal here. But the basic idea remains. Handle the error and shut down. Your server should spin up a new process to take its place on demand. Added bonus, you can log the request IP, maybe if it happens a few times from the same address, you block it.

Here's an idea for a little improvement. Make it possible to increase
the overall number of symbols. Symbol-creating DoS attacks would then be
caught as above, get logged/sent a "stop it" page,..., blocked,
whatever, but the exception catching code would increase the limit so
the process could continue to run with other threads.

That assumes that you have a pretty good idea of the amount of symbols
your application creates without attacks, and that that number is
stable, and not too big (e.g. 5000 or 10000 or so as in a previous mail
in this thread).

It also assumes that symbols get created up-front, when the application
is created, and not as a result of running a benign thread. That may be
somewhat difficult to do (but not impossible).

That way, you could e.g. detect a DoS attack per 1000 symbols, which
means that you can absorb quite a few DoS attacks per 1G of memory. Of
course, if I were you, I'd probably restart processes that suffered such
a DoS attack sooner rather than later, just in case, but that would be a
separate decision.

Regards, Martin.

#28 - 02/07/2013 12:23 PM - duerst (Martin Dürst)

A slightly different idea, closer to the existing garbage collection:

The existing garbage collection is based on finding all pointers to
locations that can possibly be heap locations. This is done by scanning
the stack and all kinds of other locations that may keep heap pointers.

So to garbage collect symbols, one would scan all the locations that
possibly might contain symbols. As far as I remember from Minero Aoki's
black book, pointers, integers, and symbols (and true/false/nil)
partition the space of 32-bit (or 64-bit) integers.

This scan would have to include all the Ruby-internal data structures
that use symbols. As with the current "pessimistic" garbage collector,
any 32-bit (or 64-bit) value that is found and is the same as an
existing symbol would make that symbol non-garbage-collectible. If no
such symbol is found, the symbol would be collected.

Anyway, this is just an idea, there may be quite a few downsides to it.

Regards,   Martin.

On 2013/02/06 23:21, SASADA Koichi wrote:

> One rough idea (but not verified) is:
>
> Separated Symbols into two sets:
> (a) Symbols created by rb_intern()
> (b) Symbols created from String object (String#to_sym)
>
> (a) is internal symbols which are used by the interpreter such as method
> names, attribute names and so on.
>
> (b) is mainly created by ruby program (and used for DoS attack).
>
> I think (hope) (b) can be collected at GC timing with some development
> efforts. But not touched. Sorry.
>
> PS. Of course, a program making symbols belong to (a) will be DoS
> attack. For example, the program makes many methods or attributes by
> untrusted data, it will be same problem (but I believe nobody makes such
> a bad program).

#29 - 02/07/2013 12:53 PM - ko1 (Koichi Sasada)

(2013/02/07 12:20), "Martin J. Dürst" wrote:

> Anyway, this is just an idea, there may be quite a few downsides to it.

The downside is performance and availability. Mark will be spread wide
and wide. And it is difficult to find out all of marking places.

However, for example using Boeham GC will be good solution to do it.

We need challengers :)

--
// SASADA Koichi at atdot dot net

#30 - 02/07/2013 01:53 PM - Student (Nathan Zook)

I can't claim to know much about ruby's internals, but this strikes me as the wrong approach.

Symbols have a number of features which differentiate them from other objects--strings in particular. These features play important/central roles in MRI, are the basis of a significant performance improvements, and are relied upon throughout the code.

And substantial change to the behaviour of symbols would require complete audits at multiple levels.

It may be even worse in userland. These same properties are exploited by programmers in the same ways. Architectural decisions will need to be audited, and may need to be rethought.

Furthermore, the use of intern verses to_sym is arbitrary and incomplete. There are many good reasons to dynamically generate methods, and in at least some of these cases, it would be desirable to generate the name of the method outside of the method declaration itself. On the other hand, string-to-symbol conversions might well be done inside an eval.

One option that would me seem to be more feasible would be to create a class GCSymbol < Symbol. GCSymbol use might initially be significantly restrained, and eased as the core team gained confidence in the idea. I still don't like this.

The problem is that the utility of symbols is high enough that programmers very much want to take advantage of them. The fact that a DOS exploit becomes available in certain use cases is the problem, and ought to be directly addressed. Two options come to mind. 1) Define Symbol.defined? In current code, it would look like this:

class Symbol
def self.defined?(string)
all_symbols.any?{|sym| sym.to_s == string}
end
end

1. Define #to_existing_sym in the appropriate places. This would raise an argument error if a proposed symbol was not already defined.

Either of these would provide the necessary facilities for HashWithIndifferentAccess, YAML, or similar constructs to proceed in safety, should they choose to do so, and neither requires tearing out & redoing a fundamental part of ruby.

#31 - 02/07/2013 08:29 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

I think Koichi's approach is a better one. I don't think there are any
needs to actually garbage collect internally used symbols like method
names and the like.

You'll usually want to collect symbols created with to_sym. They should
probably be stored in a separate location.

Em 07-02-2013 01:20, "Martin J. Dürst" escreveu:

> A slightly different idea, closer to the existing garbage collection:
>
> The existing garbage collection is based on finding all pointers to
> locations that can possibly be heap locations. This is done by
> scanning the stack and all kinds of other locations that may keep heap
> pointers.
>
> So to garbage collect symbols, one would scan all the locations that
> possibly might contain symbols. As far as I remember from Minero
> Aoki's black book, pointers, integers, and symbols (and
> true/false/nil) partition the space of 32-bit (or 64-bit) integers.
>
> This scan would have to include all the Ruby-internal data structures
> that use symbols. As with the current "pessimistic" garbage collector,
> any 32-bit (or 64-bit) value that is found and is the same as an
> existing symbol would make that symbol non-garbage-collectible. If no
> such symbol is found, the symbol would be collected.
>
> Anyway, this is just an idea, there may be quite a few downsides to it.
>
> Regards,   Martin.

#32 - 02/07/2013 10:53 PM - ko1 (Koichi Sasada)

(2013/02/07 20:25), Rodrigo Rosenfeld Rosas wrote:

> I think Koichi's approach is a better one. I don't think there are any
> needs to actually garbage collect internally used symbols like method
> names and the like.

Note that my idea is not verified.

--
// SASADA Koichi at atdot dot net

#33 - 02/14/2013 01:53 AM - evanphx (Evan Phoenix)

I have a worry about Koichi's approach. There are going to have to be places where (b) type Symbols are converted into (a) type Symbols. All those places will turn into DOS vectors. This is because SYMBOL_P() will have to detect both types, but calling SYM2ID() will have to convert type (b) into type (a) so that an ID can be returned.

Doing a quick grep of 1.9.3-p194, I see 93 uses of SYM2ID, so all of those would have to be changed/audited as well any future uses of SYM2ID would have to be done very carefully.

This is not unsurmountable, but it's something that will have to be dealt if the 2 types approach is used.

--
Evan Phoenix // evan@phx.io

On Thursday, February 7, 2013 at 5:35 AM, SASADA Koichi wrote:

> (2013/02/07 20:25), Rodrigo Rosenfeld Rosas wrote:
>
> > I think Koichi's approach is a better one. I don't think there are any
> > needs to actually garbage collect internally used symbols like method
> > names and the like.
>
> Note that my idea is not verified.
>
> --
> // SASADA Koichi at atdot dot net

#34 - 02/14/2013 08:23 AM - ko1 (Koichi Sasada)

(2013/02/14 1:38), Evan Phoenix wrote:

> I have a worry about Koichi's approach. There are going to have to be
> places where (b) type Symbols are converted into (a) type Symbols. All
> those places will turn into DOS vectors. This is because SYMBOL_P() will
> have to detect both types, but calling SYM2ID() will have to convert
> type (b) into type (a) so that an ID can be returned.
>
> Doing a quick grep of 1.9.3-p194, I see 93 uses of SYM2ID, so all of
> those would have to be changed/audited as well any future uses of SYM2ID
> would have to be done very carefully.
>
> This is not unsurmountable, but it's something that will have to be
> dealt if the 2 types approach is used.

Absolutely. We need to change code bases to keep (b) symbols. It may
affect compatibility of C extensions. This is why I repeating it is
`unverified'.

--
// SASADA Koichi at atdot dot net

#35 - 02/15/2013 08:57 AM - Student (Nathan Zook)

It seems to me that the motivation behind this proposal is to allow the interning of untrusted data to not affect the system. It seems to me that the proposal to garbage collect some symbols is complicated, and as such, likely to generate bugs. Perhaps my proposal #7854 addresses the problem in a way that is less likely to complicate the life of the ruby core team & ruby users.

#36 - 02/15/2013 03:25 PM - marcandre (Marc-Andre Lafortune)

I don't think the goal is to simplify the life of ruby-core people.

I feel the goal is to have the best language possible be reasonably secure by default.

It would be sad if Ruby and its community suffered because of a reputation of insecurity, especially an insecurity by design that ruby-core won't address.

Maybe it will take a corporate sponsorship and a couple of weeks (months?) of full time work for someone to resolve the issue, but I don't see a fundamental reason why it couldn't be done.

#37 - 02/15/2013 05:05 PM - Student (Nathan Zook)

My definition of complicating the life of the core team is undertaking projects that are likely to cause bugs.

That also complicates the life of everyone using Ruby.

Garbage collection only secures one part of the to_sym problem, that of the memory leak. It does nothing to address detainting. It does nothing to address random symbol creation. Symbol[] provides a facility that can be used to guarantee this.

I can think of no improvement that GCing Symbols does that can not be matched by simply refusing to intern tainted strings. Of course, that would expose a lot of brain-dead code. It would also break a lot of acceptable code that is only acceptable because of the details of the environment in which it runs.

As for the proposal here, I have already pointed out, it is not enough to use the two different ways that symbols are created to partition the set of symbols for GC purposes. You don't want to GC any method names of linked classes, and it doesn't matter how they were first accessed. If the idea is to limit memory leakage, you DO want to GC method names of classes which are delinked. Which makes a complete hash of the proposal to do a partial GC.

But a full GC destroys many of the advantages of having symbols in the first place. Furthermore, it is going to impact performance in hard-to-predict ways.

Security is not optional. But step 1 of security is to be bug free. That means simple implementations where possible. And Symbol[] provides a facility that addresses the memory leak completely without affecting core behaviour. What it doesn't do is correct a lot of brain-dead behaviour favoured by the users of some web frameworks (which, in their defence, is actually encouraged by the developers of these frameworks). I don't think it is the responsibility of the core team to try to protect people from being idiots.

#38 - 02/27/2013 04:20 AM - alexeymuranov (Alexey Muranov)

=begin
How about adding a new class between ((({Symbol}))) and ((({String}))) instead of garbage collecting symbols? Something like ((({StaticString}))) with a special literal notation, like

`"There exist only one copy of this string"("utf-8") # a static string in UTF-8 encoding

`this_is_also_unique   # in the default encoding

`"the 3rd letter of this string is"[2]   # => "e"

'this is unique!'.equal? 'this is unique!' # => true
They would be indexed and garbage collected constant strings. Using the same literal twice would actually use the same object.

Other alternatives for the literal notation: ((({"This is static!"}))), ((({|"This is static!"}))), ((({%S"This is static!"}))).
=end

#39 - 03/07/2013 10:08 AM - trans (Thomas Sawyer)

Would something like this do the trick? http://github.com/rubyworks/safe_symbol

#40 - 03/07/2013 12:30 PM - jeremyevans0 (Jeremy Evans)

trans (Thomas Sawyer) wrote:

> Would something like this do the trick? http://github.com/rubyworks/safe_symbol

No.

:a != SafeSymbol('a')

Even if you fixed that, Symbols are immediates in ruby's C API, and passing a SafeSymbol instance in place of a symbol is likely to work incorrectly if the method is implemented in C (consider SYM2ID, SYMBOL_P). This is true not just for external C extensions, but also for core ruby classes (example: file.advise(SafeSymbol('normal')).

#41 - 03/07/2013 10:32 PM - trans (Thomas Sawyer)

@jeremyevans0 (Jeremy Evans) Thanks for clarifying these limitations.

So, how far might this be remedied if it is made a C extension? For instance, it is not possible to make SafeSymbol a subclass of Symbol in pure Ruby, but I suspect it could be managed in C. To what degree might that help? Is there anything that can be done to SafeSymbol to make macros like SYM2ID or SYMBOL_P work? Or is the only fix to modify the macros? (Sorry, I am not a C programmer, these are called macros, right?)

Also, I do not think it must necessarily be a 100% perfect drop in replacement for Symbol to be useful. Where it is most likely to be important is for hash keys and hash values and comparisons between them. An error due to one the limitations you mention could serve as an indication to the developer that they have to make a choice at that point as to whether the SafeSymbol is okay to convert to a real Symbol, or if they need to handle things differently.

#42 - 03/08/2013 01:19 AM - jeremyevans0 (Jeremy Evans)

I don't think there is any way to implement GC-able symbols via an extension library. Significant changes to the interpreter will be required to implement them.

Personally, I think the String#to_existing_sym and/or Symbol[string] proposals are a far better way of handling this type of issue. Having GC-able symbols only encourages bad behavior. Symbols and strings are different and should remain different, and I think GC-able symbols would add complexity without benefit. The only reason non-GC-able symbols are an issue is DoS attacks, and String#to_existing_sym and/or Symbol[string] solves that problem in a better way than GC-able symbols.

#43 - 03/08/2013 01:52 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

Jeremy, I don't understand how #to_existing_sym could avoid the issue. Take your gem "sequel" for instance. In my application there is a class that will generate dynamic queries based on the user params. Sequel will always convert those dynamic column alias (like "v786") to symbols. Those symbols won't exist when the application boots. If Sequel used #to_existing_sym for creating the symbol it would raise since that symbol wouldn't exist.

#44 - 03/08/2013 02:02 AM - jeremyevans0 (Jeremy Evans)

rosenfeld (Rodrigo Rosenfeld Rosas) wrote:

> Jeremy, I don't understand how #to_existing_sym could avoid the issue. Take your gem "sequel" for instance. In my application there is a class that will generate dynamic queries based on the user params. Sequel will always convert those dynamic column alias (like "v786") to symbols. Those symbols won't exist when the application boots. If Sequel used #to_existing_sym for creating the symbol it would raise since that symbol wouldn't exist.

I wouldn't use #to_existing_sym for identifiers returned from databases. It is assumed that there is a fixed number that will be used, and allowing user-controlled identifiers can be a security issue. If your identifiers are based on user params, and you aren't validating them, you currently have denial of service at the least and possible SQL injection depending on the adapter in use.

There are other cases in Sequel where #to_existing_sym would might make sense using, though (connection string options, JSON/XML deserialization).

#45 - 03/08/2013 02:17 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

Of course the params are verified and there is no risk of SQL injection in my case either. I'm joining the same table multiple times at least once for each condition in the search query request that contains a reference to some field. The number of fields is limited by the ones existing in the database. To give you a concrete example, suppose we have two fields with ids 786 and 2048 and both are of type 'string'. So there is a table in the database called "string_value". I need to join the same table twice once for each field. The query will then generate the aliases v786 and v2048 to the same table in different joins. And the columns would be aliased "v786_value" and "v2048_value" for instance. It doesn't make any sense in my opinion that I should have to worry about memory starvation in my application just because Sequel will convert all my columns to symbols after running my dynamic queries. I don't want even to worry if they are symbols or strings, that's why I proposed that other ticket to make symbols and strings behave the same.

#46 - 03/15/2013 02:34 AM - kstephens (Kurt Stephens)

Mark and sweep the symbol table like any other GC heap.

However there are some issues in the C API.

IDs in the C API are distinct from other values and Ruby extensions expected them to be pinned (never GCd).
Therefore, the C API must support registering ID variables with the GC or mark all Symbols requested through the C API as pinned.
The latter could be achieved by having a "pinned" flag in the String->Symbol table entries.
Symbols that are created and reachable by usual means (i.e.: String#to_sym) do not initially set this "pinned" flag.

When the GC is marking Symbols, it set a "marked" flag in the corresponding symbol table entry.
The GC sweep phase scans the symbol table and reclaims entries where !(e.pinned && e.marked) and clears e.marked.

This behavior can be a run-time option.

My employer will sponsor me to do this work.

#47 - 03/15/2013 03:00 AM - marcandre (Marc-Andre Lafortune)

kstephens (Kurt Stephens) wrote:

> Mark and sweep the symbol table like any other GC heap.
>
> However there are some issues in the C API.
>
> IDs in the C API are distinct from other values and Ruby extensions expected them to be pinned (never GCd).
> Therefore, the C API must support registering ID variables with the GC or mark all Symbols requested through the C API as pinned.
> The latter could be achieved by having a "pinned" flag in the String->Symbol table entries.
> Symbols that are created and reachable by usual means (i.e.: String#to_sym) do not initially set this "pinned" flag.
>
> When the GC is marking Symbols, it set a "marked" flag in the corresponding symbol table entry.
> The GC sweep phase scans the symbol table and reclaims entries where !(e.pinned && e.marked) and clears e.marked.

+1, exactly what I was thinking.

> My employer will sponsor me to do this work.

Awesome news!

My personal thanks to your employer (Enova Financials, right?)

#48 - 03/15/2013 10:09 AM - Student (Nathan Zook)

kstephens (Kurt Stephens) wrote:

> Mark and sweep the symbol table like any other GC heap.
>
> However there are some issues in the C API.
>
> IDs in the C API are distinct from other values and Ruby extensions expected them to be pinned (never GCd).
> Therefore, the C API must support registering ID variables with the GC or mark all Symbols requested through the C API as pinned.
> The latter could be achieved by having a "pinned" flag in the String->Symbol table entries.
> Symbols that are created and reachable by usual means (i.e.: String#to_sym) do not initially set this "pinned" flag.

Questions:

1. How certain are you that this covers all of the cases?
2. In order to actually recover the memory, the symbol table has to be walked each time a symbol is created. What are the implications of this?

#49 - 03/15/2013 10:22 AM - Student (Nathan Zook)

rosenfeld (Rodrigo Rosenfeld Rosas) wrote:

> Of course the params are verified and there is no risk of SQL injection in my case either. I'm joining the same table multiple times at least once for each condition in the search query request that contains a reference to some field. The number of fields is limited by the ones existing in the database. To give you a concrete example, suppose we have two fields with ids 786 and 2048 and both are of type 'string'. So there is a table in the database called "string_value". I need to join the same table twice once for each field. The query will then generate the aliases v786 and v2048 to the same table in different joins. And the columns would be aliased "v786_value" and "v2048_value" for instance. It doesn't make any sense in my opinion that I should have to worry about memory starvation in my application just because Sequel will convert all my columns to symbols after running my dynamic queries. I don't want even to worry if they are symbols or strings, that's why I proposed that other ticket to make symbols and strings behave the same.

First, there is nothing about Symbol[] (which I much prefer to my earlier suggestion) that is mandatory--and it makes no sense to use it for things that come from database column names. That data better be trustable, or you have bigger problems.

Second, I really don't understand your example. How do particular values for a particular role become part of the alias for an entire column?

Certainly, it is possible to generate arbitrary symbols under some excuse or the other everytime a website gets hit. But the problem is not with Symbol. The problem is that the code abusing Symbol. Symbols are not supposed to be transient. Strings are. If what you are creating is transient, it is not a Symbol. If you are overloading the notion of a Symbol to also be something more (ie: some sort of wildly dynamic column-related thing), then this is an error. Subclass String, and go from there.

#50 - 03/15/2013 09:57 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Student (Nathan Zook) wrote:

> ...Second, I really don't understand your example. How do particular values for a particular role become part of the alias for an entire column?

I think you won't understand until I give you an overview of how the application works. There is a dynamic fields tree template that is managed by an editor of the application. Editors will decide upon what fields should exist in a given template, define who is child of whom, and give them a name and a type. Since we use PostgreSQL (a RDBMS) we're forced to have a separate table per possible field type (text, number, date, etc) so that we can have non-null constraints more easily (the alternative being having a single table with different column values, one per type, and a trigger to make sure at least one of them is filled in, but I don't like that approach actually for many reasons). Then some attorneys will link some document excerpts to those fields for a certain transaction. The client interface consists mainly on a Search interface that will allow the clients to search all transactions matching certain conditions involving those fields and their values.

Some fields may be just displayed without any conditions attached. So, basically I have to join the same values tables multiple times. For instance, if 3 date fields were requested by the client, the date_values table will be "left join"ed 3 times and they will be aliased as v#{field_id}. The value column for each table would be aliased as v#{field_id}_value. Please let me know if I still couldn't manage to convince you that I'm talking about a real use case here.

I'm not overloading the notion of a Symbol. I simply can't request Sequel to return me the columns as strings instead of symbols and Sequel simply assumes that the columns will be limited to the real column names as declared in the table schema. I'm showing you that this is not always true by giving you a real use case of how I use Sequel. I'm also stating that this is a headache that Ruby will force me to think about using strings or symbols because of possible memory leak when I don't really care about it. I only care about being able to retrieve the values from a dynamic query :(

#51 - 03/16/2013 01:32 AM - kstephens (Kurt Stephens)

Student (Nathan Zook) wrote:

> Questions:

> 1. How certain are you that this covers all of the cases?

With the unit and functional tests that already exist. What cases do you have in mind?

> 1. In order to actually recover the memory, the symbol table has to be walked each time a symbol is created. What are the implications of this?

The symbol table only has to be walked during sweep and can be done incrementally.

Progress:

I have a branch that alters global_symbols st_tables to map id->symbol_entry and string->symbol_entry. This works but complicates reuse of collected IDs, they are finite. The costs of determining which IDs are reclaimed make this design inefficient and convoluted. So...

I'm considering creating a first-class struct RSymbol and making ID synonymous with VALUE, such that all IDs are VALUEs pointing to RSymbols; ID2SYM() and SYM2ID() become identity functions.

It may break C ABI because sizeof(ID) may not be equal to sizeof(VALUE) on some platforms, and requires minor changes to parser.y, but will make everything else much simpler. Subsequently, all IDs in internal structures must be rb_gc_mark()ed.

Assuming that ID rb_intern(const char *x) means "pin the symbol with the name x", VALUE rb_intern_str_collectible(VALUE x) means "the symbol with the name x, pinned or not". This should reduce required changes to C API extensions that will not rb_gc_mark() on all IDs, until they adopt a new contract.

#52 - 03/16/2013 01:47 AM - marcandre (Marc-Andre Lafortune)

kstephens (Kurt Stephens) wrote:

> I'm considering creating a first-class struct RSymbol and making ID synonymous with VALUE

I might be mistaken, but wouldn't that be a problem because many external libraries assume, like MRI does, that rb_intern("*") == '*', and the same for '+', ... This would no longer be the case, right?

#53 - 03/16/2013 02:06 AM - kstephens (Kurt Stephens)

marcandre (Marc-Andre Lafortune) wrote:

> kstephens (Kurt Stephens) wrote:
>
> > I'm considering creating a first-class struct RSymbol and making ID synonymous with VALUE
>
> I might be mistaken, but wouldn't that be a problem because many external libraries assume, like MRI does, that rb_intern("*") == '*', and the same for '+', ... This would no longer be the case, right?

Yup. That's an assumption I can't fix. Unless we have a class of VALUES < 127 that are immediate single-ASCII character Symbols, and we cant do that because it will collide with other immediates. Ugh. More magic, lookup tables and special cases. :(

I was gonna define CHAR2SYM(C) and SYM2CHAR(VALUE) for things like ID2SYM('+') in parse.y and elsewhere.

Are there really that many extensions that rely on rb_symbol("+") == '+', or switch stmts? They should rb_intern("+") into a static ID (or VALUE!) variable, like most extensions.

IMO, ID/VALUE distinction and magic is an unnecessary abomination.

#54 - 03/16/2013 03:33 AM - headius (Charles Nutter)

I was thinking about how to implement this in JRuby. Couldn't you just have *all* Symbols be weakly-referenced in the symbol table, and just root the ones that are created from C?

I'll try to prototype something in JRuby.

#55 - 03/16/2013 03:54 AM - marcandre (Marc-Andre Lafortune)

kstephens (Kurt Stephens) wrote:

> Unless we have a class of VALUES < 127 that are immediate single-ASCII character Symbols, and we cant do that because it will collide with other immediates.

Wouldn't it be safe to do this in ID2SYM though? I.e. keep the distinction between ID and VALUE, where in most case they are equal, but there's a mapping for a few special IDs < 127?

> Are there really that many extensions that rely on rb_symbol("+") == '+', or switch stmts? They should rb_intern("+") into a static ID (or VALUE!) variable, like most extensions.

You're right, they should. I was horrified the first time I realized that '+' was used as the ID for the :+ symbol!

I double checked Rubinius and that magic relation does not hold. In particular:

#define ID2SYM(id)   (id)

So it's probably only MRI that does stuff like rb_funcall(obj, '>', 1, INT2FIX(0))

#56 - 03/16/2013 06:05 AM - Student (Nathan Zook)

kstephens (Kurt Stephens) wrote:

> Student (Nathan Zook) wrote:
>
> > Questions:
> >
> > 1. How certain are you that this covers all of the cases?
>
> With the unit and functional tests that already exist. What cases do you have in mind?

The existing test cases all assume that symbol behaves the way that it currently does. There are tests to check a against certain deviations, but not all possible ones. This is what I meant earlier about the probability that such a change will introduce bugs. We absolutely cannot rely on the current test suite to guarantee the safety/sanity of a change of this magnitude.

> > 1. In order to actually recover the memory, the symbol table has to be walked each time a symbol is created. What are the implications of this?
>
> The symbol table only has to be walked during sweep and can be done incrementally.

That assumes that you reassign all symbols after a gap. DANGER, WILL ROBINSON!!! DANGER!!!

#57 - 03/16/2013 06:49 AM - Student (Nathan Zook)

I'm sorry, but this example just gets more strange the more you explain it. Are you saying that there is one table with one two columns (id & value) for all of the text fields? That would certainly parallel your previous statement.

But when you do your queries, why do you alias the columns against the id field value that your are matching? Why not alias against the column number in the template? (This solution actually has me concerned on a number of points).

More to the point, you ARE doing exactly what I said--you are creating transitory Symbols, which is an abuse. Most ORMs assume a constant or near-constant schema. You have implemented a solution which breaks that assumption, so you need to use an ORM (or make one) that does not have that assumption.

No tool can be all things to all people. Symbols are a tool that ruby uses to solve certain problems. Their permanence is a feature, not a bug. There is a lot of code that relies upon this fact, and it can be hard to see what all.

rosenfeld (Rodrigo Rosenfeld Rosas) wrote:

> Student (Nathan Zook) wrote:
>
> > ...Second, I really don't understand your example. How do particular values for a particular role become part of the alias for an entire column?
>
> I think you won't understand until I give you an overview of how the application works. There is a dynamic fields tree template that is managed by an editor of the application. Editors will decide upon what fields should exist in a given template, define who is child of whom, and give them a name and a type. Since we use PostgreSQL (a RDBMS) we're forced to have a separate table per possible field type (text, number, date, etc) so that we can have non-null constraints more easily (the alternative being having a single table with different column values, one per type, and a trigger to make sure at least one of them is filled in, but I don't like that approach actually for many reasons). Then some attorneys will link some document excerpts to those fields for a certain transaction. The client interface consists mainly on a Search interface that will allow the clients to search all transactions matching certain conditions involving those fields and their values.
>
> Some fields may be just displayed without any conditions attached. So, basically I have to join the same values tables multiple times. For instance, if 3 date fields were requested by the client, the date_values table will be "left join"ed 3 times and they will be aliased as v#{field_id}. The value column for each table would be aliased as v#{field_id}_value. Please let me know if I still couldn't manage to convince you that I'm

> talking about a real use case here.
>
> I'm not overloading the notion of a Symbol. I simply can't request Sequel to return me the columns as strings instead of symbols and Sequel simply assumes that the columns will be limited to the real column names as declared in the table schema. I'm showing you that this is not always true by giving you a real use case of how I use Sequel. I'm also stating that this is a headache that Ruby will force me to think about using strings or symbols because of possible memory leak when I don't really care about it. I only care about being able to retrieve the values from a dynamic query :(

#58 - 03/16/2013 07:10 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

Student (Nathan Zook) wrote:

> I'm sorry, but this example just gets more strange the more you explain it. Are you saying that there is one table with one two columns (id & value) for all of the text fields?

Basically, yes.

> But when you do your queries, why do you alias the columns against the id field value that your are matching?

I have to alias it to some name to be able to fetch it by name, right? Using the id just seems natural to me and makes it easy to understand my queries.

> More to the point, you ARE doing exactly what I said--you are creating transitory Symbols, which is an abuse.

I'm not creating any symbols myself. Sequel is.

> Most ORMs assume a constant or near-constant schema. You have implemented a solution which breaks that assumption, so you need to use an ORM (or make one) that does not have that assumption.

Yeah, that makes sense. I should create a new ORM just because symbols are not garbage collected. That is just an example of how Ruby is developed towards developers' happiness.

#59 - 03/16/2013 08:55 AM - kstephens (Kurt Stephens)

marcandre (Marc-Andre Lafortune) wrote:

> So it's probably only MRI that does stuff like rb_funcall(obj, '>', 1, INT2FIX(0))
> That can be changed to:

```
#define CHAR2SYM(X) _rb_char_symbol_table[X]
rb_funcall(obj, CHAR2SYM('>'), 1, INT2FIX(0))
```

or similar.

#60 - 06/26/2014 12:42 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

I think I read somewhere this has been already implemented in trunk. If that's true I think this ticket should be closed as resolved.

#61 - 06/26/2014 01:01 PM - hsbt (Hiroshi SHIBATA)

- Status changed from Feedback to Closed

#62 - 07/15/2014 03:27 PM - Ajedi32 (Ajedi32 W)

Yes, this is a duplicate of 9634.