Ruby master - Feature #8977

String#frozen that takes advantage of the deduping

10/02/2013 01:12 PM - sam.saffron (Sam Saffron)

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

Description

During memory profiling I noticed that a large amount of string duplication is generated from non pre-determined strings.

Take this report for example https://gist.github.com/SamSaffron/6789005 (generated using the memory_profiler gem that works against head)

">=" x 4953
/Users/sam/.rbenv/versions/2.1.0-dev/lib/ruby/2.1.0/rubygems/requirement.rb:93 x 4535

This string is most likely extracted from a version.

Or

"/Users/sam/.rbenv/versions/2.1.0-dev/lib/ruby/gems" x 5808
/Users/sam/.rbenv/versions/2.1.0-dev/lib/ruby/gems/2.1.0/gems/activesupport-3.2.12/lib/active_support/dependencies.rb:251 x 3894

A string that can not be pre-determined.

It would be nice to have

"hello,world".split(",")[0].frozen.object_id == "hello"f.object_id

Adding #frozen will give library builders a way of using the de-duping. It also could be implemented using weak refs in 2.0 and stubbed with a .dup.freeze in 1.9.3 .

Thoughts ?

Related issues:

Related to Ruby master - Feature #8976: file-scope freeze_string directive	**Closed**	
Has duplicate Ruby master - Bug #9229: [patch] expose rb_fstring() as String#...	**Closed**	**12/08/2013**

History

#1 - 10/02/2013 02:00 PM - charliesome (Charlie Somerville)

- Target version set to 2.1.0

I would love to see this feature in 2.1.

These are the top duplicated strings in an app I work on:

```
irb(main):023:0> GC.start; h = ObjectSpace.each_object(String).to_a.group_by { |s| s }.map{ |s, objs| [s, objs
.size] }; h.sort_by { |s, count| -count }.take(10).each do |s| p s end; nil
["/", 5241]
["(eval)", 3207]
["application", 2389]
["", 1908]
["html.erb", 1720]
["base64", 1520]
["erb", 1464]
["IANA", 1389]
["initialize", 1147]
["recognize", 1036]
```

Most of these could be deduplicated with String#frozen.

#2 - 10/02/2013 02:22 PM - nobu (Nobuyoshi Nakada)

Won't those strings be shared with frozen string literal?

#3 - 10/02/2013 02:30 PM - sam.saffron (Sam Saffron)

nobu (Nobuyoshi Nakada)

"html.erb" is very unlikely to be shared cause it is a result of a parse. "base64" and "IANA" are coming from the super dodgy mime types gem here: https://github.com/halostatue/mime-types/blob/master/lib/mime/types/application it starts off unsplit.

#4 - 10/02/2013 10:12 PM - charliesome (Charlie Somerville)

ko1 and I discussed this in IRC and decided that #frozen would be too easily confused with #freeze. An idea that came up was to use #dedup or #pooled instead. What do you think Sam?

#5 - 10/03/2013 02:58 AM - headius (Charles Nutter)

How is this not just a symbol table of another sort? When do these pooled strings get GCed? Do they ever get GCed? What if the encodings differ?

There's a whole bunch of implementation details that scare me about this proposal.

#6 - 10/03/2013 03:07 AM - headius (Charles Nutter)

After thinking a bit, I guess what your'e asking for is a method that gives you the VM-level object that would be returned for a literal frozen version of the same string. However, it's unclear to me what #frozen or #dedup or #pooled would do if there were no such string.

If they'd return the original uncached object, you'd never know if you're actually saving anything.

If they would cache the string, the concerns in my previous comment apply.

Can you clarify?

#7 - 10/03/2013 10:11 AM - sam.saffron (Sam Saffron)

@hedius

the request is all about exposing:

```
VALUE
rb_fstring(VALUE str)
{
    st_data_t fstr;
    if (st_lookup(frozen_strings, (st_data_t)str, &fstr)) {
    str = (VALUE)fstr;
    }
    else {
    str = rb_str_new_frozen(str);
    RBASIC(str)->flags |= RSTRING_FSTR;
    st_insert(frozen_strings, str, str);
    }
    return str;
}
```

the encoding concerns are already handled by st_lookup afaik, as is the gc concern

```
def test; x = "asasasa"f; x.object_id; end
test
=> 70185750124120
undef :test
GC.start
def test; x = "asasasa"f; x.object_id; end
test
=> 70185736068940
```

Overall this feature has some parity with Java / .NETs intern, adapted to the world where MRI is not allow you to shift objects around

charlie (charlie ablett) I like #pooled , #dedup feels a bit odd ... I totally understand the concern about #frozen vs #frozen? it can be confusing

#8 - 10/03/2013 08:18 PM - headius (Charles Nutter)

sam.saffron (Sam Saffron) wrote:

> the request is all about exposing:

> VALUE
> rb_fstring(VALUE str)
> ...
> the encoding concerns are already handled by st_lookup afaik, as is the gc concern

I went to the source to understand how this is implemented. Summarized here for purposes of discussion.

"fstrings" in source are added to the fstring table. Normally this would mean they're hard-referenced forever, but fstrings also get an FSTR header bit that the GC uses (via rb_str_free) to also remove the fstring table entry.

So you're right, the fstrings will not fill up memory like the global symbol table and there's probably no DOS potential from creating lots of fstrings via eval or #frozen.

I guess my next question is why we need a new method. Why can't String#freeze just do what you want String#frozen to do? Risk of too many strings going into that table?

My other concerns are addressed by the handling of the fstring table. I think in JRuby we'd implement this as a weak hash map.

> def test; x = "asasasa"f; x.object_id; end
> test
> => 70185750124120
> undef :test
> GC.start
> def test; x = "asasasa"f; x.object_id; end
> test
> => 70185736068940

I ran this in a loop and the object_id eventually stabilizes. I am not sure why.

I also ran a version that loops forever creating new test methods with different fstrings, and confirmed that memory stays level.

#9 - 10/03/2013 08:21 PM - headius (Charles Nutter)

headius (Charles Nutter) wrote:

> I ran this in a loop and the object_id eventually stabilizes. I am not sure why.

I think I realize why: eventually the only GC is for the objects in the loop, which are allocated and deallocated the same way every time. So although a new fstring is defined each time, it lives at the same location in memory as the one from the previous loop.

#10 - 10/04/2013 09:14 AM - nobu (Nobuyoshi Nakada)

I don't think it needs a new method nor class.

```
frozen_pool = Hash.new {|h, s| h[s.freeze] = s}

3.times {
  p frozen_pool["foo"].object_id
}
```

#11 - 10/04/2013 09:29 AM - ko1 (Koichi Sasada)

(2013/10/04 9:14), nobu (Nobuyoshi Nakada) wrote:

> I don't think it needs a new method nor class.
>
> ```
> frozen_pool = Hash.new {|h, s| h[s.freeze] = s}
>
> 3.times {
>   p frozen_pool["foo"].object_id
> }
> ```

for `f' syntax, we prepare frozen string table. Let it name "FrozenTable".

This proposal String#frozen can be defined by:

```
class String
  def frozen
    FrozenTable[frozen] ||= self.freeze # rb_fstring(self) in C level
  end
```

```
  end
```

The difference between hash table approach and rb_fstring() is GC.
Frozen strings returned by Strign#frozen are collected if frozen strings
are not marked. But hash table approach doesn't allow collecting.

--
// SASADA Koichi at atdot dot net

#12 - 10/04/2013 09:35 AM - nobu (Nobuyoshi Nakada)

It differs from the original proposal, which is called explicitly by applications/libraries.
I think such pooled strings should not go beyond app/lib domains.

#13 - 10/04/2013 09:59 AM - ko1 (Koichi Sasada)

(2013/10/04 9:35), nobu (Nobuyoshi Nakada) wrote:

> It differs from the original proposal, which is called explicitly by applications/libraries.

Not different. I described the implementation of String#frozen.

> I think such pooled strings should not go beyond app/lib domains.

I can accept the way to get the string which we can get "foo"f dynamically.

--
// SASADA Koichi at atdot dot net

#14 - 10/04/2013 02:33 PM - sam.saffron (Sam Saffron)

@hedius

I have seen the suggestion around of having String#freeze amend the object id on the current string, so for example

> "hi"f.object_id
> 10
> a="hi"; a.object_id
> => 100
> a.freeze; a.object_id
> => 10

However how would such an implementation work with c extensions where we leak out pointers? (I think this would work fine though for JRuby)

#15 - 10/04/2013 02:36 PM - charliesome (Charlie Somerville)

> I have seen the suggestion around of having String#freeze amend the object id on the current string, so for example
> However how would such an implementation work with c extensions where we leak out pointers?

C extensions aren't the only reason this wouldn't work. Consider:

```
a = "foo"
b = a
a.freeze
puts a.object_id
puts b.object_id
```

Are both 'a' and 'b' updated to refer to the new object?

#16 - 10/04/2013 03:38 PM - sam.saffron (Sam Saffron)

nobu (Nobuyoshi Nakada)

You can implement a separate string pool in 2.0 using like so:

```
require 'weakref'

class Pool
  def initialize
```

```
    @pool = {}
  end
  def get(str)
    ref = @pool[str]

    # GC may run between alive? and __getobj__
    copy = ref && ref.weakref_alive? && copy = ref.__getobj__ rescue nil
    unless copy
      copy=str.dup
      ref=WeakRef.new(copy)
      copy.freeze
      @pool[str] = ref
    end
    copy
  end

  def scrub!
    GC.start
    @pool.delete_if{|k,v| v.nil?}
  end

  def length
    @pool.length
  end
end

@pool = Pool.new
def test
  puts @pool.get("test").object_id
end

test #69822933011880
test #69822933011880

p @pool.length #1

@pool.scrub!

test #69822914568080
test #69822914568080

p @pool.length #1
```

but the disadvantage is that it is fiddely, requires manual management and will not reuse FrozenTable

#17 - 10/10/2013 12:46 AM - headius (Charles Nutter)

I believe we should just make #freeze use the fstring pool as mentioned in #8992. I do not see any disadvantages to this other than having strings stick around a bit longer in implementations that keep them in a weak map.

#18 - 10/10/2013 01:01 AM - headius (Charles Nutter)

Actually, I'm getting pretty down on having the fstring cache at all. It seems like if we want a string pool, it should be via a library. Adding something into Ruby that pools strings for you just seems like asking for trouble, either due to GC overhead (cleaning up that hash for tons of transient frozen strings) and semantics (abuse of #frozen or #freeze to do pooling implicitly).

#19 - 12/09/2013 07:08 AM - tmm1 (Aman Gupta)

- Status changed from Open to Assigned

- Assignee set to matz (Yukihiro Matsumoto)

I just made some more arguments for this feature in #9229.

The goal here is to provide runtime access to the frozen string literal table.
This is not a new idea. For instance, see String.Intern in .NET: http://msdn.microsoft.com/en-us/library/system.string.intern(v=vs.110).aspx

@samsaffron also made a good point above: older versions and other implementations of ruby can easily provide their own implementations of String#frozen:

> It also could be implemented using weak refs in 2.0 and stubbed with a .dup.freeze in 1.9.3 .

ko1 (Koichi Sasada) also agrees with the feature above:

> I can accept the way to get the string which we can get "foo"f dynamically.

So the remaining question (as always) is a naming issue.
I like String#frozen, but maybe there is some argument that it can be confused with #freeze.

matz (Yukihiro Matsumoto) Do you approve of String#frozen, or do you have some other preference? Other proposals are:

String#dedup
String#pooled
String::frozen(str)

#20 - 12/09/2013 07:23 AM - phluid61 (Matthew Kerwin)

headius (Charles Nutter) wrote:

> Actually, I'm getting pretty down on having the fstring cache at all. It seems like if we want a string pool, it should be via a library. Adding something into Ruby that pools strings for you just seems like asking for trouble, either due to GC overhead (cleaning up that hash for tons of transient frozen strings) and semantics (abuse of #frozen or #freeze to do pooling implicitly).

Is it any worse than the fact that String#intern returns a Symbol? IIRC this whole effort started because people were using Symbols as interned Strings (in the Java sense), but Symbols can't be GC'ed, so there were memory leak-type issues. If we're viewing the fstring cache as an effort to allow GC'ing of Symbols (effectively, though not in name) then it seems the issues and complexities are a given.

I agree that we should make #freeze use the pool. If people really, really want to have a version that returns the same object (frozen), we could introduce String#freeze!

- rb_define_method(rb_cString, "freeze", rb_obj_freeze, 0);
- rb_define_method(rb_cString, "freeze", rb_fstring, 0);
- rb_define_method(rb_cString, "freeze!", rb_obj_freeze, 0);

This is based on my (possibly flawed) understanding that Ruby seems willing to make not-backwards-compatible changes between minor versions (1.8 -> 1.9), even if not between majors (1.9.3 -> 2.0). The benefits of having a pooled #freeze seem to outweigh the risk of someone depending on it returning the same object, especially if that person has an upgrade path to get their old functionality back.

#21 - 12/09/2013 03:35 PM - ko1 (Koichi Sasada)

Now, I have one concern about security concern.

This kind of method can be used widely and easily.

And if this method is used with external string getting from IO,
fstring table can be grow and grow easily.

I'm afraid about such kind of security risk:
(1) DoS attack
(2) Side channel attack (observe from outside)

But I'm not a security expert. So I want to ask experts.

Note that this problem has not impact than Symbol related DoS attack
because these keys are collected.

I think multiple tables support solve kind of issues.

To solve such issue (and continue to discuss this issue to 2.2),
Ruby level implementation and gem is reasonable alternative, I believe.

However, in ruby-level, we can't do that same thing.

Therefor nobu made a patch for WeakHash, a variant of WeakMap (we will make another ticket for it).

WeakMap is object_id -> Object map.
WeakHash is Object -> Object map.

With this class, we can make fstring technique
with multiple tables easily.

```
class FrozenStringTable
  def initialize
    @table = {} # WeakHash.new
  end
```

```
  def get str
    raise TypeError unless str.kind_of?(String)

    unless @table.has_key? str
      str.freeze
      @table[str] = str
    end
    @table[str]
  end
end

F1 = FrozenStringTable.new
p F1.get('foo').object_id #=> 8274120
p F1.get('foo').object_id #=> 8274120
```

In this comment, I show (1) security concern, and (2) alternative approach.

#22 - 12/09/2013 06:30 PM - tmm1 (Aman Gupta)

ko1 (Koichi Sasada) and I discussed this at length earlier.

Although a frozen string table could be implemented in ruby (with the help of C-ext like WeakHash above), the current implementation of the finalizer table adds overhead that would make it unsuitable for long-lived strings. In particular, each finalizer currently requires 2 extra object VALUE slots and the finalizer_table is marked on every minor mark.

The main concern with exposing the fstr table to ruby is that it could be easily be misused. Feeding a large number of entries into this table would slow down lookup times and subsequent calls to rb_fstring(). Currently rb_fstring() calls are isolated to boot-up and compile time, so runtime performance is not a factor. Misuse from ruby-land could increase the number of frozen_strings hash lookups, possibly introducing performance or security concerns.

As as alternative, we discussed including a C-only api for 2.1. This would limit possible misuse/abuse, yet still allow for responsible use of de-duplication features present in 2.1. We should include the size of the frozen_strings table to encourage monitoring and size caps. This API might look something like the following (naming suggestions welcome):

VALUE rb_str_frozen_dedup(VALUE str)
size_t rb_str_frozen_table_size()

#23 - 01/30/2014 06:17 AM - hsbt (Hiroshi SHIBATA)

- Target version changed from 2.1.0 to 2.2.0

#24 - 01/05/2018 09:00 PM - naruse (Yui NARUSE)

- Target version deleted (2.2.0)

#25 - 01/27/2018 06:29 AM - sam.saffron (Sam Saffron)

I think this can be closed as complete cause we have -"test" now so we can do it.