Ruby - Feature #13077

[PATCH] introduce String#fstring method

12/27/2016 01:50 AM - normalperson (Eric Wong)

Status:	Closed
Priority:	Normal
Assignee:	
Target version:	

Description

introduce String#fstring method

This exposes the rb_fstring internal function to return a
deduped and frozen string. This is useful for writing all sorts
of record processing key values maybe stored, but certain keys
and values are often duplicated at a high frequency,
so memory savings can noticeable.

Use cases are many:

- email/NNTP header processing

  There are some standard header keys everybody uses
  (From/To/Cc/Date/Subject/Received/Message-ID/References/In-Reply-To),
  as well as common ones specific to a certain lists:
  (ruby-core has X-Redmine-* headers)
  It is also useful to dedupe values, as most inboxes have
  multiple messages from the same sender, or MUA.

- package management systems -
  things like RubyGems stores identical strings for licenses,
  dependency names, author names/emails, etc

- HTTP headers/trailers -
  standard headers (Host/Accept/Accept-Encoding/User-Agent/...)
  are common, but there are also uncommon ones.
  Values may be deduped, as well, as it is likely a user
  agent will make multiple/parallel requests to the same
  server.

- version control systems -
  this can be useful for deduplicating names of frequent
  committers (like "nobu" :)

  In linux.git and git.git, there are also common
  trailers such as Signed-Off-By/Acked-by/Reviewed-by/Fixes/...
  as well as less common ones.

- audio metadata -

  There are commonly used tags (Artist/Album/Title/Tracknumber),
  but Vorbis comments allows arbitrary key values to be stored.
  Music collections contain songs by the same artist or mutiple
  songs from the same album, so deduplicating values will be
  helpful there, too.

- JSON, YAML, XML, HTML processing

  certain fields, tags and attributes are commonly used
  across the same and multiple documents

Related issues:

Has duplicate Ruby - Feature #17147: New method to get frozen strings from St...	**Feedback**

Associated revisions

Revision 4e90dcc9 - 02/24/2017 01:01 AM - Eric Wong

string.c (str_uminus): deduplicate strings

This exposes the rb_fstring internal function to return a
deduped and frozen string when a non-frozen string is given.
This is useful for writing all sorts of record processing key
values maybe stored, but certain keys and values are often
duplicated at a high frequency, so memory savings can
noticeable.

Use cases are many:

- email/NNTP header processing

  There are some standard header keys everybody uses
  (From/To/Cc/Date/Subject/Received/Message-ID/References/In-Reply-To),
  as well as common ones specific to a certain lists:
  (ruby-core has X-Redmine-* headers)
  It is also useful to dedupe values, as most inboxes have
  multiple messages from the same sender, or MUA.

- package management systems -
  things like RubyGems stores identical strings for licenses,
  dependency names, author names/emails, etc

- HTTP headers/trailers -
  standard headers (Host/Accept/Accept-Encoding/User-Agent/...)
  are common, but there are also uncommon ones.
  Values may be deduped, as well, as it is likely a user
  agent will make multiple/parallel requests to the same
  server.

- version control systems -
  this can be useful for deduplicating names of frequent
  committers (like "nobu" :)

  In linux.git and git.git, there are also common
  trailers such as Signed-Off-By/Acked-by/Reviewed-by/Fixes/...
  as well as less common ones.

- audio metadata -

  There are commonly used tags (Artist/Album/Title/Tracknumber),
  but Vorbis comments allows arbitrary key values to be stored.
  Music collections contain songs by the same artist or mutiple
  songs from the same album, so deduplicating values will be
  helpful there, too.

- JSON, YAML, XML, HTML processing

  Certain fields, tags and attributes are commonly used
  across the same and multiple documents

There is no security concern in this being a DoS vector by
causing immortal strings. The fstring table is not a GC-root
and not walked during the mark phase. GC-able dynamic symbols
since Ruby 2.2 are handled in the same manner, and that
implementation also relies on the non-immortality of fstrings.

[Feature #13077] [ruby-core:79663]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@57698 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 15ef28a9 - 02/25/2017 02:31 AM - Eric Wong

NEWS: document String#-@ change

- test/ruby/test_string.rb (test_uplus_minus): test deduplication
  [ruby-core:79747] [Feature #13077]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@57710 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision bb1c3418 - 04/15/2024 09:06 AM - byroot (Jean Boussier)

Add more assertions in test_uplus_minus

Not along after 1b830740ba8371c4bcfdfc6eb2cb7e0ae81a84e0 CI
started to rarely fail this test:

```
TestString#test_uplus_minus: Test::Unit::AssertionFailedError: uminus deduplicates [Feature #13077].
  1) Failure:
TestString#test_uplus_minus [/tmp/ruby/src/trunk/test/ruby/test_string.rb:3368]:
```

It's unclear what is going on, but one possibility is that
"bar".freeze might no longer compile correctly.

Another possibility is that another test redefine String#freeze,
causing opt_str_freeze to no longer return an fstring.

History

#1 - 12/27/2016 03:54 PM - shevegen (Robert A. Heiler)

I have no particular pro or con opinion on the proposal in itself so feel free to ignore this.

The only comment I have is that the name .fstring() is a bit strange. On first read, I
assumed that it was short for "format_string" like % on class String or sprintf.

In the proposal I read that it is for frozen_string e. g. rb_fstring. While I don't have
anything against the functionality, and I also don't fully mind a method called
fstring(), I think that at the least a longer alias name to it would be nice to have too
such as frozen_string or something that is more readable on a first look. (I can't comment
on whether the functionality in itself is useful or not but I assume that Eric has had
a good reason which he described too, so I have no qualms at all with the functionality
in itself, only the method-name part.)

#2 - 12/27/2016 06:41 PM - normalperson (Eric Wong)

shevegen@gmail.com wrote:

> The only comment I have is that the name .fstring() is a bit strange. On first read, I
> assumed that it was short for "format_string" like % on class String or sprintf.

Yeah, the name isn't final, of course; naming is the hardest
problem in computer science :<

Maybe "dedup" is better and still short (I would expect the user
to know deduplication implicitly requires a frozen string)

#3 - 12/27/2016 10:15 PM - Eregon (Benoit Daloze)

So this is essentially like Java's String.intern()?

There is already String#intern in Ruby but it returns a Symbol.
Depending on the use-case, I guess this might be less convenient than getting a de-duplicated String.
String#dedup or sounds better than #fstring.

#4 - 12/27/2016 11:41 PM - normalperson (Eric Wong)

eregontp@gmail.com wrote:

> So this is essentially like Java's String.intern()?
>
> There is already String#intern in Ruby but it returns a Symbol.
> Depending on the use-case, I guess this might be less convenient than getting a de-duplicated String.

Yeah, I considered using intern/to_sym for my use case;
but the problem is it that still creates a new string object
whenever it needs to be written/printed/concatenated.

And I also feel using symbol like this is ugly (just a gut
feeling), despite having GC-able symbols since 2.2.

> String#dedup or sounds better than #fstring.

Yes. Lets wait for Matz to comment.

#5 - 12/28/2016 12:08 AM - nobu (Nobuyoshi Nakada)

Why not String#-@?

#6 - 12/28/2016 03:03 AM - normalperson (Eric Wong)

nobu@ruby-lang.org wrote:

> Why not String#-@?

As in the following? (short patch, full below)

```
--- a/string.c
+++ b/string.c
@@ -10002,7 +9989,7 @@ Init_String(void)
     rb_define_method(rb_cString, "scrub!", str_scrub_bang, -1);
     rb_define_method(rb_cString, "freeze", rb_str_freeze, 0);
     rb_define_method(rb_cString, "+@", str_uplus, 0);
-    rb_define_method(rb_cString, "-@", str_uminus, 0);
+    rb_define_method(rb_cString, "-@", rb_fstring, 0);

     rb_define_method(rb_cString, "to_i", rb_str_to_i, -1);
     rb_define_method(rb_cString, "to_f", rb_str_to_f, 0);
```

Changing existing behavior method might break compatibility;
but test-all and test-rubyspec seems to pass...

full: https://80x24.org/spew/20161228024937.9345-1-e@80x24.org/raw

#7 - 02/22/2017 07:34 AM - matz (Yukihiro Matsumoto)

For the time being, let us make -@ to call rb_fstring.
If users want more descriptive name, let's discuss later.
In my opinion, fstring is not acceptable.

Matz.

#8 - 02/22/2017 09:41 AM - normalperson (Eric Wong)

matz@ruby-lang.org wrote:

> For the time being, let us make -@ to call rb_fstring.
> If users want more descriptive name, let's discuss later.
> In my opinion, fstring is not acceptable.

OK, I think the following is always backwards compatible,
unlike my previous [ruby-core:78884]:

```
--- a/string.c
+++ b/string.c
@@ -2530,7 +2530,7 @@ str_uminus(VALUE str)
   return str;
     }
     else {
- return rb_str_freeze(rb_str_dup(str));
+ return rb_fstring(str);
     }
 }
```

Will commit in a day or two.

#9 - 02/23/2017 12:51 AM - shyouhei (Shyouhei Urabe)

A bit of security consideration:

Am I correct that rb_vm_fstring_table() is never GCed? If so feeding user-generated strings to that table needs extra care. Malicious user input might let memory exhausted.

#10 - 02/23/2017 01:00 AM - nobu (Nobuyoshi Nakada)

Shyouhei Urabe wrote:

> Am I correct that rb_vm_fstring_table() is never GCed?

That table is not a GC-root, and registered strings get GCed as usual.

#11 - 02/23/2017 09:20 AM - shyouhei (Shyouhei Urabe)

Nobuyoshi Nakada wrote:

> Shyouhei Urabe wrote:
>
> > Am I correct that rb_vm_fstring_table() is never GCed?
>
> That table is not a GC-root, and registered strings get GCed as usual.

So this is a kind of weak reference? No security concern then.

#12 - 02/24/2017 01:01 AM - Anonymous

- Status changed from Open to Closed

Applied in changeset r57698.

---

string.c (str_uminus): deduplicate strings

This exposes the rb_fstring internal function to return a
deduped and frozen string when a non-frozen string is given.
This is useful for writing all sorts of record processing key
values maybe stored, but certain keys and values are often
duplicated at a high frequency, so memory savings can
noticeable.

Use cases are many:

- email/NNTP header processing

  There are some standard header keys everybody uses
  (From/To/Cc/Date/Subject/Received/Message-ID/References/In-Reply-To),
  as well as common ones specific to a certain lists:
  (ruby-core has X-Redmine-* headers)
  It is also useful to dedupe values, as most inboxes have
  multiple messages from the same sender, or MUA.

- package management systems -
  things like RubyGems stores identical strings for licenses,
  dependency names, author names/emails, etc

- HTTP headers/trailers -
  standard headers (Host/Accept/Accept-Encoding/User-Agent/...)
  are common, but there are also uncommon ones.
  Values may be deduped, as well, as it is likely a user
  agent will make multiple/parallel requests to the same
  server.

- version control systems -
  this can be useful for deduplicating names of frequent
  committers (like "nobu" :)

  In linux.git and git.git, there are also common
  trailers such as Signed-Off-By/Acked-by/Reviewed-by/Fixes/...
  as well as less common ones.

- audio metadata -

  There are commonly used tags (Artist/Album/Title/Tracknumber),
  but Vorbis comments allows arbitrary key values to be stored.
  Music collections contain songs by the same artist or mutiple
  songs from the same album, so deduplicating values will be
  helpful there, too.

- JSON, YAML, XML, HTML processing

  Certain fields, tags and attributes are commonly used
  across the same and multiple documents

There is no security concern in this being a DoS vector by
causing immortal strings. The fstring table is not a GC-root
and not walked during the mark phase. GC-able dynamic symbols
since Ruby 2.2 are handled in the same manner, and that
implementation also relies on the non-immortality of fstrings.

[Feature #13077] [ruby-core:79663]

#13 - 02/24/2017 01:05 AM - normalperson (Eric Wong)

shyouhei@ruby-lang.org wrote:

> Nobuyoshi Nakada wrote:
>
> > Shyouhei Urabe wrote:
> >
> > > Am I correct that rb_vm_fstring_table() is never GCed?
> >
> > That table is not a GC-root, and registered strings get GCed as usual.
>
> So this is a kind of weak reference? No security concern then.

Right. Also, keep in mind that dynamic GC-able symbols from
2.2+ also stores symbol names as fstrings. Thus GC-able symbols
would not work if fstrings could not be GC-ed.

Anyways, committed as r57698

#14 - 02/24/2017 10:28 AM - Eregon (Benoit Daloze)

Eric Wong wrote:

> Anyways, committed as r57698

This should have a NEWS entry and tests since it changes the semantics.

BTW, should my_string.freeze behave similarly to String#@-?
Otherwise String#freeze only dedup if the String is a literal.
Always deduping for String#freeze would make the semantics more consistent.

#15 - 02/25/2017 02:11 AM - normalperson (Eric Wong)

eregontp@gmail.com wrote:

> Eric Wong wrote:
>
> > Anyways, committed as r57698
>
> This should have a NEWS entry and tests since it changes the semantics.

Sorry, I forgot; will do. Thanks for the reminder.

> BTW, should my_string.freeze behave similarly to String#@-?
> Otherwise String#freeze only dedup if the String is a literal.
> Always deduping for String#freeze would make the semantics more consistent.

No. There is existing code which assumes #freeze always returns
the same object as its caller. Changing #freeze will break
existing code.

We can only cheat with String literals (opt_str_freeze) because
literals are not assigned to user-visible variables, yet.

#16 - 02/25/2017 08:53 PM - Eregon (Benoit Daloze)

Eric Wong wrote:

> No. There is existing code which assumes #freeze always returns
> the same object as its caller. Changing #freeze will break
> existing code.
>
> We can only cheat with String literals (opt_str_freeze) because
> literals are not assigned to user-visible variables, yet.

Oh indeed, that slipped my mind, thanks for the explanation.

#17 - 09/04/2020 09:58 AM - naruse (Yui NARUSE)

- Has duplicate Feature #17147: New method to get frozen strings from String objects added

Files

0001-introduce-String-fstring-method.patch	3.47 KB	12/27/2016	normalperson (Eric Wong)