Ruby master - Feature #16029

Expose fstring related APIs to C-extensions

07/31/2019 01:06 AM - byroot (Jean Boussier)

Status:	Closed	
Priority:	Normal	
Assignee:		
Target version:		

Description

As discussed with @tenderlove here: https://github.com/ruby/ruby/pull/2287#issuecomment-513865160

We'd like to update various data format parsers (JSON, MessagePack, etc) to add the possibility to deduplicate strings while parsing.

But unfortunately the rb_fstring_* family of functions isn't available to C-extensions, so the only available fallback is rb_funcall(str, rb_intern("-@")) which most parsers will likely consider too slow. So the various rb_fstring_* functions would need to be public.

Proposed patch: https://github.com/ruby/ruby/pull/2299

Related issues:

Is duplicate of Ruby master - Feature #13381: [PATCH] Expose rb_fstring and i...	**Assigned**

History

#1 - 09/03/2019 03:32 PM - byroot (Jean Boussier)

mame (Yusuke Endoh) this feature was added by nobu (Nobuyoshi Nakada) in the last developer's meeting issue [#15996], but I suppose it wasn't discussed by lack of time.

Should I just add it back to the next developer's meeting issue once it is created?

#2 - 09/03/2019 04:42 PM - mame (Yusuke Endoh)

ko1 (Koichi Sasada) and nobu (Nobuyoshi Nakada) discussed the issue. The memo says:

> ko1, nobu: now it is difficult to expose them because of current implementation.

I didn't follow the discussion, so I don't know the detail. ko1 (Koichi Sasada) , nobu (Nobuyoshi Nakada) , could you explain them?

#3 - 10/07/2019 10:25 PM - sam.saffron (Sam Saffron)

I think the larger change here is allowing for a new type of API.

From a performance perspective the people using the new API would like to avoid allocating an RVALUE unless needed, the current fstring APIs require an RVALUE unless you give it a constant string I think?

Perhaps what we need here is the ability to ask Ruby: "Hey do you have an fstring for cstr X? If so then you use it, otherwise you do the slow path of allocating an RVALUE and passing it in to the fstring function.

#4 - 10/08/2019 11:07 AM - byroot (Jean Boussier)

sam.saffron (Sam Saffron) Unless I'm missing something, that's exactly what rb_fstring_new / rb_fstring_cstr does.

```
VALUE rb_fstring_new(const char *ptr, long len);
VALUE rb_fstring_cstr(const char *str);
```

AFAICT It does allocate an RVALUE to lookup the table, but it does it on the stack, so I think it's fine GC wise.

#5 - 10/08/2019 11:34 PM - sam.saffron (Sam Saffron)

I think when it gets called it expects to reuse the memory allocated by the cstr eventually

https://github.com//blob/96753e8475ee69537134ab3d966c3d25cb5c467c/string.c#L287-L292

So if your library is in charge of the memory for the object this is not desirable, you want to simply ask the question "do you have an fstring for the current string eg"

VALUE rb_fstring_lookup(char *ptr);

This non const char means it would not take ownership and the thing can return Qnil if there is no fstring.

Then if 99.999% of the string your library has are already fstrings, the lookup becomes a super cheap function you can use to lookup.

#6 - 10/14/2019 05:22 AM - sam.saffron (Sam Saffron)

I was thinking something like?

```
VALUE
rb_fstring_lookup(char *ptr, rb_encoding *enc)
{
    st_data_t fstring;
    struct RString fake_str;
    setup_fake_str(&fake_str, ptr, len, ENCINDEX_US_ASCII)

    st_table *frozen_strings = rb_vm_fstring_table();

    if (!st_lookup(frozen_strings, (st_data_t)fake_str, &fstring)) {
    return Qnil;
    }

    return ret;
}
```

#7 - 10/17/2019 05:55 AM - ko1 (Koichi Sasada)

Hi.

(1) implementation

Current implementation can have issue (related to shared string) and this issue can cause something wrong behavior for C-extension.
Sorry, we need a time to confirm.

(2) naming

i think it should be rb_str_... prefix.

#8 - 10/17/2019 09:09 PM - sam.saffron (Sam Saffron)

Koichi,

What about rb_str_fstring_lookup and rb_str_fstring_lookup_enc? Both will not create strings so shared strings should not be a problem.

To be honest creation can be somewhat inefficient, the one place I can see this being used is in DB drivers like PG where a consumer keeps asking for field names over and over.

#9 - 10/28/2019 12:59 PM - byroot (Jean Boussier)

> What about rb_str_fstring_lookup and rb_str_fstring_lookup_enc?

I don't think a lookup would be enough for what I'd like to do.

A typical use case would be a JSON document with lots of duplicated strings.

If we only lookup the fstring table, then only the strings already present in the codebase would be deduplicated. IMHO it would be much better to create them all as fstrings.

#10 - 12/31/2019 06:46 AM - sam.saffron (Sam Saffron)

byroot (Jean Boussier)

I think it heavily depends on usage... MySQL gem / PG would benefit from "lookup" followed by "create fstring if missing" cause vast majority of string it is creating when querying tables.

My proposal is for the minimal building block we could use for getting fstrings unconditionally which would offer a fast path for fstring reuse, then new fstrings can be created using todays awkward API.

#11 - 12/31/2019 09:56 AM - k0kubun (Takashi Kokubun)

- Is duplicate of Feature #13381: [PATCH] Expose rb_fstring and its family to C extensions added

#12 - 01/01/2020 11:05 AM - byroot (Jean Boussier)

sam.saffron (Sam Saffron) I agree that both would be nice.

> then new fstrings can be created using todays awkward API

You mean rb_funcall(str, rb_intern("-@")) ?

#13 - 01/05/2020 12:58 AM - sam.saffron (Sam Saffron)

byroot (Jean Boussier), yeah.

rb_funcall(str, rb_intern("-@")) works and is backwards compatible, but I entirely agree that it makes sense to add an official API as well that does not depend on dynamic invocation.

#14 - 02/14/2020 02:19 PM - byroot (Jean Boussier)

> it makes sense to add an official API as well that does not depend on dynamic invocation

It's not just about the dynamic invocation. Sure it's a bit slower but not a huge deal.

It's mostly that to get to that invocation you need to first allocate a duplicate string.

The ideal API would allow to pass a char* to avoid all this garbage.

For context our app loads a lot of static data from flat files (json / yaml / messagepack) for a total of over 5M objects (~25% of the allocations during boot). 80% of that ends up deduplicated later on by String#-@.

I believe that if json / psych / messagepack had access to rb_fstring_cstr, I could avoid allocating all these useless objects and save a lot of GC time (~30% of our boot time is spent in GC, 25% marking and 5% sweeping).

#15 - 07/17/2020 09:23 AM - Eregon (Benoit Daloze)

See https://bugs.ruby-lang.org/issues/13381#note-6

#16 - 07/17/2020 10:39 AM - byroot (Jean Boussier)

Yes I saw it this morning. Now that it's assigned and likely to be merged soon. We can probably close this issue as duplicate.

#17 - 07/19/2020 11:35 PM - hsbt (Hiroshi SHIBATA)

- Status changed from Open to Closed