Right size regular expression compile buffers for literal regexes and on Regexp#freeze

11/27/2019 01:01 AM - methodmissing (Lourens Naudé)

Status: Closed
Priority: Normal
Assignee: nobu (Nobuyoshi Nakada)
Target version: 3.0

Description

As a continuation of type specific resize on freeze implementations of String and Array and looking into the Regexp type I found these memory access patterns for regular expression literals:

```
==22079== ------------------------ 12 of 500 ------------------------
==22079== max-live: 1,946,560 in 4,345 blocks
==22079== tot-alloc: 1,946,560 in 4,345 blocks (avg size 448.00)
==22079== deaths: none (none of these blocks were freed)
==22079== acc-ratios: 1.36 rd, 0.98 wr (2,651,994 b-read, 1,908,158 b-written)
==22079== at 0x4C2DECF: malloc (in /usr/lib/valgrind/vgpreload_exp-dhat-amd64-linux.so)
==22079== by 0x24C496: onig_new_with_source (re.c:844)
==22079== by 0x24C496: make_regexp (re.c:874)
==22079== by 0x24C496: rb_reg_initialize (re.c:2858)
==22079== by 0x24C496: rb_reg_initialize_str (re.c:2892)
==22079== by 0x24C496: rb_reg_compile (re.c:2982)
==22079== by 0x24C496: reg_resize (static as ary_resize)
==22079== After compiling a literal regular expression.
==22079== Implement an explicit type specific rb_reg_freeze and point Regexp#compile to it
==22079== I also follow the chain member which points to another regex_t on the struct if present, but have not been able to find references to it in the source tree other than for freeing a regex or inspecting it's memory footprint.

Digging a little further and remembering some context of previous oniguruma memory investigation I remembered the pattern buffer struct has a compile buffer with a simple watermark for tracking used space. This changeset implements reg_resize (static as ary_resize) which attempts to right size the compile buffer if over allocated at the following sites:

- After compiling a literal regular expression.
- Implement an explicit type specific rb_reg_freeze and point Regexp#compile to it
- I also follow the chain member which points to another regex_t on the struct if present, but have not been able to find references to it in the source tree other than for freeing a regex or inspecting it's memory footprint.

I introduced 2 new debug counters, which yields the following results on booting Redmine on Rails 5:

[RUBY_DEBUG_COUNTER] obj_regexp_lit_extracapa 6319
[RUBY_DEBUG_COUNTER] obj_regexp_lit_extracapa_bytes 301685

About 300kb reallocated across 6319 oversized instances.

An example of Regexp#freeze

```ruby
irb(main):007:0> r = Regexp.compile("(?!%\h|[^$~&(-;=?-_a-~].")
```

03/18/2022
There is likely more layers that can be peeled back here, but keeping it simple and concise for review.

I think it's possible to get towards a state where 5 to 10MB RSS can be shaved off a standard Rails process by:

- Being careful with buffer defaults for objects that have auxiliary buffers
- Identifying hooks where excess allocation can be reduced to current watermark if the object buffer does not need to grow anymore (literal object, frozen object etc.)
- A GC hook on promotion to OLD object to trim excessive capacity, which I'd consider as a special kind of garbage, as outlined in [https://bugs.ruby-lang.org/issues/15402](https://bugs.ruby-lang.org/issues/15402)

Would any further incremental work towards this be considered worthwhile from a ruby-core and community perspective?

---

**History**

**#1 - 11/27/2019 02:43 AM - shevegen (Robert A. Heiler)**

Would any further incremental work towards this be considered worthwhile from a ruby-core and community perspective?

I can not answer for the ruby-core team, or anyone else, but matz has the motto of 3x3 for ruby 3.0 which is destined for next year.

Some time ago, I think, matz said that with mjit the goal has been more or less achieved - but matz also likes to say that nobody minds more speed improvement, even if they are only little opt carrot improvements to get the duck going (I think it's a duck wanting to get to the carrot). So that would leave lots of time for next year still, even with small improvements - might be even a fun competition to get to the goal without mjit. :)

I guess the core team may have a look at any potential side effects or trade offs of any changes.

**#2 - 11/27/2019 04:10 PM - methodmissing (Lourens Naudé)**

shevegen (Robert A. Heiler) wrote:

Would any further incremental work towards this be considered worthwhile from a ruby-core and community perspective?

I can not answer for the ruby-core team, or anyone else, but matz has the motto of 3x3 for ruby 3.0 which is destined for next year.

Some time ago, I think, matz said that with mjit the goal has been more or less achieved - but matz also likes to say that nobody minds more speed improvement, even if they are only little opt carrot improvements to get the duck going (I think it's a duck wanting to get to the carrot). So that would leave lots of time for next year still, even with small improvements - might be even a fun competition to get to the goal without mjit. :)

I guess the core team may have a look at any potential side effects or trade offs of any changes.

Yes this isn't focused on speed, but more so on being careful with auxiliary memory buffers and inferring where to reduce over allocated ones (especially on immutable or old objects). I expect to never post to the list with an order of magnitude improvement, but do think incrementally scavenging a few kilobytes here with goals of

- Not complicating MRI
- Backwards compatible and non breaking changes
- Not negatively affecting performance in order to reduce memory overhead (usually 1 or the other, rarely can have both)

eventually can add up in the low single digit or even MB reduction range for Rails processes.

03/18/2022
After a brief exchange with Jean Boussier (byroot), he noted that regular expressions can be considered immutable when allocated and initialized and as such I dropped the Regexp#freeze API addition and instead applied the resize optimization to every allocation callsite (4 of them) and got a 50% improvement in compile buffer cleanup by effectively extending the net to a few more regex objects (300kb -> 469kb):

<table>
<thead>
<tr>
<th>Reference</th>
<th>Count</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[RUBY_DEBUG_COUNTER]</td>
<td>7249</td>
<td>obj-regexp_lit_extracapa</td>
</tr>
<tr>
<td>[RUBY_DEBUG_COUNTER]</td>
<td>481204</td>
<td>obj-regexp_lit_extracapa_bytes</td>
</tr>
</tbody>
</table>

References [https://github.com/ruby/ruby/pull/2696#issuecomment-559147405](https://github.com/ruby/ruby/pull/2696#issuecomment-559147405)

#3 - 12/26/2019 02:32 AM - mame (Yusuke Endoh)
- Target version set to 36
- Assignee set to nobu (Nobuyoshi Nakada)
- Tracker changed from Misc to Feature

nobu (Nobuyoshi Nakada) will review the patch.

methodmissing (Lourens Naudé) Could you file a ticket about performance improvement and/or refactoring into the Feature tracker?

#4 - 12/26/2019 03:08 AM - methodmissing (Lourens Naudé)

mame (Yusuke Endoh) wrote:

nobu (Nobuyoshi Nakada) will review the patch.

methodmissing (Lourens Naudé) Could you file a ticket about performance improvement and/or refactoring into the Feature tracker?

Thanks for the consideration and discussing the proposal. Do you mean file a refactored version of this ticket which has drifted in scope on the Feature tracker with the memory improvements?

#5 - 12/26/2019 03:54 AM - mame (Yusuke Endoh)

methodmissing (Lourens Naudé) wrote:

Thanks for the consideration and discussing the proposal. Do you mean file a refactored version of this ticket which has drifted in scope on the Feature tracker with the memory improvements?

Sorry, I meant please choose "Feature" (not "Misc") as a tracker if you register a ticket about improvements from the next time.

I've already moved this ticket to Feature tracker. You have to do nothing about this ticket unless nobu requests.

#6 - 09/29/2020 03:37 AM - hsbt (Hiroshi SHIBATA)
- Target version changed from 36 to 3.0

#7 - 12/10/2020 09:23 AM - mame (Yusuke Endoh)
- Status changed from Open to Closed

Merged by nobu at 9a17437558e42aa1da372b515ba8bc18067d578c