Ruby master - Feature #16847

Cache instruction sequences by default

05/11/2020 10:16 AM - byroot (Jean Boussier)

Status:	Closed
Priority:	Normal
Assignee:	
Target version:	

Description

Instruction sequence caching is available since Ruby 2.3, and on recent rubies it speeds up code loading by about 30%.

I just benchmarked it on Redmine's master, using bootsnap with only that optimization enabled:

```
if ENV['CACHE_ISEQ']
  require 'bootsnap'
  Bootsnap.setup(
    cache_dir:            'tmp/cache',
    development_mode:     false,
    load_path_cache:      false,
    autoload_paths_cache: false,
    disable_trace:        false,
    compile_cache_iseq:   true,
    compile_cache_yaml:   false,
  )
end

$ RAILS_ENV=production time bin/rails runner 'p 1'
        2.70 real         2.02 user         0.67 sys
$ RAILS_ENV=production time bin/rails runner 'p 1'
        2.70 real         2.02 user         0.67 sys
$ CACHE_ISEQ=1 RAILS_ENV=production time bin/rails runner 'p 1'
        1.89 real         1.27 user         0.60 sys
$ CACHE_ISEQ=1 RAILS_ENV=production time bin/rails runner 'p 1'
        1.90 real         1.28 user         0.61 sys
```

Since Bootsnap is installed by default when you create a new Rails app, many Ruby users already benefit from it, however not all applications are Rails applications, and some users remove it because they tend to blame it as it appear on most backtrace.

Having read previous discussions about it, my understanding is that caching instruction sequences by default is only a matter of agreeing on a storage mechanism.

Python store them alongside source files as .pyc. If I remember correctly Matz wasn't very kin on introducing .rbc files.
The alternative would be to store them in a dedicated directory, that you could define with an environment variable (e.g. $RUBY_CACHE_PATH), and would have a sane default. The downside here of course is permission management, especially on shared systems.
You don't want to load cache files that might have been generated by another users, potentially a malicious one.

I'm not particularly opinionated on which storage mechanism should be used, but it's disappointing that so many Ruby users pass out on this fairly significant optimization because it's opt-in.

History

#1 - 05/12/2020 07:21 AM - shyouhei (Shyouhei Urabe)

Why not compile into native binary then? I have recently learnt that Emacs is moving from its .elc bitecodes to .eln native shared object
https://arxiv.org/abs/2004.02504

Because we already have JIT it seems now possible for us to AOT.

#2 - 05/12/2020 07:38 AM - byroot (Jean Boussier)

shyouhei (Shyouhei Urabe) that would be a radical change in how Ruby is deployed.

Also boot performance is particularly important during development, where AOT doesn't make sense.

#3 - 05/12/2020 07:54 AM - shyouhei (Shyouhei Urabe)

byroot (Jean Boussier) wrote in #note-2:

> shyouhei (Shyouhei Urabe) that would be a radical change in how Ruby is deployed.

Agreed, however isn't that also true for instruction sequence caches?

> Also boot performance is particularly important during development, where AOT doesn't make sense.

This point is valid, and makes me wonder how a cached instruction sequence shall be invalidated.

#4 - 05/12/2020 08:12 AM - byroot (Jean Boussier)

> isn't that also true for instruction sequence caches?

No, caching instruction sequence can be done without any functional change for users.

> makes me wonder how a cached instruction sequence shall be invalidated.

That depend of the storage mechanism. But assuming you have 1 cache file for each source file. You simply compare the mtime of each. If the cache is older than the source file, you invalidate it. That's how Bootsnap does it today, that's also how https://github.com/ko1/yomikomu does it in most of it's backends, and that's also how .pyc files work in Python.

#5 - 05/12/2020 08:39 AM - shyouhei (Shyouhei Urabe)

byroot (Jean Boussier) wrote in #note-4:

> > isn't that also true for instruction sequence caches?
>
> No, caching instruction sequence can be done without any functional change for users.

I mean, that can also be achieved when you compile your ruby script into an shared object ahead-of-time. Basically there must be no difference between caching an instruction sequence and caching its compiled binary (apart form compile time overhead).

> > makes me wonder how a cached instruction sequence shall be invalidated.
>
> That depend of the storage mechanism. But assuming you have 1 cache file for each source file. You simply compare the mtime of each. If the cache is older than the source file, you invalidate it. That's how Bootsnap does it today, that's also how https://github.com/ko1/yomikomu does it in most of it's backends, and that's also how .pyc files work in Python.

I guess the "depend of the storage mechanism" part is what is about to be discussed in this ticket.

#6 - 05/12/2020 09:55 AM - byroot (Jean Boussier)

> I mean, that can also be achieved when you compile your ruby script into an shared object ahead-of-time.

Sure, but then it's no longer "by-default", as in all users get it for free without having to configure anything nor changing their workflow.

#7 - 05/12/2020 06:39 PM - ko1 (Koichi Sasada)

isn't it enough to use libraries such as bootsnap for it?
This kind of technique can cause something strange behavior and users may know what they are doing. Using a library is good opt-in method, IMO.

#8 - 05/12/2020 07:18 PM - byroot (Jean Boussier)

> This kind of technique can cause something strange behavior

Can you tell me what kind of behavior you are thinking of? I don't think that was ever a problem with bootsnap (it has a few corner cases, but none that I can think of for the IseqCache).

I don't think I ever heard any complaints about strange behaviors caused by .pyc files either.

> Using a library is good opt-in method, IMO.

That's what I thought for a long time as well, however I think having it by default would be a major improvement because several reasons.

First not every body know it actually exist. So defaulting to faster (assuming no drawback) is preferable, and it's also fairly counter intuitive to people that adding more code will make their code faster.

But I also recently discovered as part of Rails's "May Of WTFs" that many Rails users opt-out of Bootsnap because since it appears very low in backtraces, it tend to take the blame for their unrelated loading issues.

And more generally I'm thinking it's disappointing to see sow many people pass out on such fairly simple optimization when there's very little missing in ruby core to have it enabled by default.

#9 - 05/13/2020 03:48 AM - naruse (Yui NARUSE)

- Status changed from Open to Feedback

How cache is effective and how large cache storage is needed depend an application.
But Ruby don't have enough example of such applications other than Rails.

#10 - 05/13/2020 08:47 AM - byroot (Jean Boussier)

> How cache is effective and how large cache storage is needed depend an application.

Of course there is some variance, but from my testing the improvement is relatively constant. The cache is approximately 1.5 times as large as the source file, but load approximately 30% faster. In today's usage of Ruby I can't think of anyone that wouldn't accept such tradeoff.

#11 - 05/13/2020 08:49 AM - byroot (Jean Boussier)

Also I forgot to mention. Python 3 no longer store this cache alongside source files, but in a subdirectory.

Python 2 use to store path/foo.py cache as path/foo.pyc, Python 3 now stores it as path/__pycache__/foo.pyc.

#12 - 05/14/2020 02:58 AM - shevegen (Robert A. Heiler)

I think python changed that default mostly because it can be annoying to see all .py files give rise to .pyc
files in the very same working directory.

(I know way too little to comment on the content of the issue at hand here, so this was just a short
statement to python.)

#13 - 05/14/2020 08:49 AM - mrkn (Kenta Murata)

Julia uses ~/.julia/compiled/vX.Y directory to store precompiled cache files.

#14 - 05/27/2020 12:12 AM - matz (Yukihiro Matsumoto)

I am negative. Cacheing compiled binary caused a lot of problems in Python, especially with multiple versions installed. We'd rather improve Bootsnap if they need support from the core.

Matz.

#15 - 05/27/2020 10:39 AM - byroot (Jean Boussier)

> We'd rather improve Bootsnap if they need support from the core.

Ok. Then we can close this issue.

That part of Bootsnap doesn't need any support from core, it works fine except for what I explained, it takes the blame for many issues because it appear in backtraces.

What would really help though is https://bugs.ruby-lang.org/issues/16848, but that a different concern.

#16 - 05/27/2020 02:12 PM - jeremyevans0 (Jeremy Evans)

- Status changed from Feedback to Closed