

## Ruby master - Feature #3905

### rb\_clear\_cache\_by\_class() called often during GC for non-blocking I/O

10/05/2010 06:56 AM - normalperson (Eric Wong)

<b>Status:</b>	Closed	
<b>Priority:</b>	Normal	
<b>Assignee:</b>		
<b>Target version:</b>	2.0.0	
<b>Description</b> =begin This still causes performance problems with frequent EAGAIN compared to 1.9.1  While akr fixed extend to no longer clear cache with empty modules in r28813, the GC phase still scans and clears the cache when the extended object is collected.  ref: [ruby-core:32507], [ruby-core:32508]  A proposed patch to add memoizing of extended objects with IO::Wait{Read,Writ}able is attached. Comments/feedback appreciated. =end		
<b>Related issues:</b> Related to Ruby master - Bug #4289: Timeouts in threads cause SEGV <span style="float: right;">Closed 01/18/2011</span>		

#### Associated revisions

##### Revision 8c09664a - 06/12/2011 08:53 AM - ko1 (Koichi Sasada)

- vm\_method.c (rb\_clear\_cache\*): update only vm state version.
- vm\_method.c (rb\_method\_entry\_get\_without\_cache, rb\_method\_entry): Fill method cache entry with vm state version, and check current vm state version for method (cache) look up. This modification speed-up invaridating of global method cache table. [Ruby 1.9 - Feature #3905] [ruby-core:36908]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@32024 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

##### Revision 32024 - 06/12/2011 08:53 AM - ko1 (Koichi Sasada)

- vm\_method.c (rb\_clear\_cache\*): update only vm state version.
- vm\_method.c (rb\_method\_entry\_get\_without\_cache, rb\_method\_entry): Fill method cache entry with vm state version, and check current vm state version for method (cache) look up. This modification speed-up invaridating of global method cache table. [Ruby 1.9 - Feature #3905] [ruby-core:36908]

##### Revision 32024 - 06/12/2011 08:53 AM - ko1 (Koichi Sasada)

- vm\_method.c (rb\_clear\_cache\*): update only vm state version.
- vm\_method.c (rb\_method\_entry\_get\_without\_cache, rb\_method\_entry): Fill method cache entry with vm state version, and check current vm state version for method (cache) look up. This modification speed-up invaridating of global method cache table. [Ruby 1.9 - Feature #3905] [ruby-core:36908]

##### Revision 32024 - 06/12/2011 08:53 AM - ko1 (Koichi Sasada)

- vm\_method.c (rb\_clear\_cache\*): update only vm state version.
- vm\_method.c (rb\_method\_entry\_get\_without\_cache, rb\_method\_entry): Fill method cache entry with vm state version, and check current vm state version for method (cache) look up. This modification speed-up invaridating of global method cache table. [Ruby 1.9 - Feature #3905] [ruby-core:36908]

##### Revision 32024 - 06/12/2011 08:53 AM - ko1 (Koichi Sasada)

- vm\_method.c (rb\_clear\_cache\*): update only vm state version.
- vm\_method.c (rb\_method\_entry\_get\_without\_cache, rb\_method\_entry): Fill method cache entry with vm state version, and check current vm state version for method (cache) look up. This modification speed-up invaridating of global method cache table. [Ruby 1.9 - Feature #3905] [ruby-core:36908]

## Revision 32024 - 06/12/2011 08:53 AM - ko1 (Koichi Sasada)

- vm\_method.c (rb\_clear\_cache\*): update only vm state version.
- vm\_method.c (rb\_method\_entry\_get\_without\_cache, rb\_method\_entry): Fill method cache entry with vm state version, and check current vm state version for method (cache) look up. This modification speed-up invaridating of global method cache table. [Ruby 1.9 - Feature #3905] [ruby-core:36908]

## Revision 32024 - 06/12/2011 08:53 AM - ko1 (Koichi Sasada)

- vm\_method.c (rb\_clear\_cache\*): update only vm state version.
- vm\_method.c (rb\_method\_entry\_get\_without\_cache, rb\_method\_entry): Fill method cache entry with vm state version, and check current vm state version for method (cache) look up. This modification speed-up invaridating of global method cache table. [Ruby 1.9 - Feature #3905] [ruby-core:36908]

## History

---

### #1 - 10/30/2010 05:59 PM - akr (Akira Tanaka)

=begin

2010/10/5, Eric Wong [redmine@ruby-lang.org](mailto:redmine@ruby-lang.org):

Feature #3905: rb\_clear\_cache\_by\_class() called often during GC for non-blocking I/O  
<http://redmine.ruby-lang.org/issues/show/3905>

This still causes performance problems with frequent EAGAIN compared to 1.9.1

While akr fixed extend to no longer clear cache with empty modules in r28813, the GC phase still scans and clears the cache when the extended object is collected.

The following patch fix the problem?

```
% svn diff --diff-cmd diff -x '-u -p'
```

```
Index: gc.c
```

```
=====
```

```
-- gc.c (revision 29630)
```

```
+++ gc.c (working copy)
```

```
@@ -2210,7 +2210,8 @@ obj_free(rb_objspace_t *objspace, VALUE
```

```
break;
```

```
case T_MODULE:
```

```
case T_CLASS:
```

- rb\_clear\_cache\_by\_class((VALUE)obj);
- if (RCLASS\_M\_TBL(obj)->num\_entries != 0)
- rb\_clear\_cache\_by\_class((VALUE)obj); rb\_free\_m\_table(RCLASS\_M\_TBL(obj)); if (RCLASS\_IV\_TBL(obj)) { st\_free\_table(RCLASS\_IV\_TBL(obj)); -- Tanaka Akira

```
=end
```

### #2 - 11/01/2010 09:18 PM - akr (Akira Tanaka)

=begin

2010/10/30 Tanaka Akira [akr@fsj.org](mailto:akr@fsj.org):

2010/10/5, Eric Wong [redmine@ruby-lang.org](mailto:redmine@ruby-lang.org):

Feature #3905: rb\_clear\_cache\_by\_class() called often during GC for non-blocking I/O  
<http://redmine.ruby-lang.org/issues/show/3905>

This still causes performance problems with frequent EAGAIN compared to 1.9.1

While akr fixed extend to no longer clear cache with empty modules in r28813, the GC phase still scans and clears the cache when the extended object is collected.

The following patch fix the problem?

The following patch may be better.

```
% svn diff --diff-cmd diff -x '-u -p'
Index: vm_method.c
=====
--- vm_method.c (revision 29630)
+++ vm_method.c (working copy)
@@ -85,6 +85,9 @@ rb_clear_cache_by_class(VALUE klass)
{
struct cache_entry *ent, *end;

  • if (RCLASS_M_TBL(klass)->num_entries == 0)

  • ██████████

+
+ rb_vm_change_state();

if (!ruby_running)

Tanaka Akira

=end
```

**#3 - 11/03/2010 04:25 AM - normalperson (Eric Wong)**

=begin  
Tanaka Akira [akr@fsij.org](mailto:akr@fsij.org) wrote:

2010/10/30 Tanaka Akira [akr@fsij.org](mailto:akr@fsij.org):

2010/10/5, Eric Wong [redmine@ruby-lang.org](mailto:redmine@ruby-lang.org):

Feature [#3905](#): rb\_clear\_cache\_by\_class() called often during GC for non-blocking I/O  
<http://redmine.ruby-lang.org/issues/show/3905>

This still causes performance problems with frequent EAGAIN compared to 1.9.1

While akr fixed extend to no longer clear cache with empty modules in r28813, the GC phase still scans and clears the cache when the extended object is collected.

The following patch fix the problem?

The following patch may be better.

Yes, I've only tried the second one as it looks cleaner and it brings non-blocking I/O performance back up to 1.9.1 levels.

Thank you!

--  
Eric Wong

=end

**#4 - 11/03/2010 07:47 AM - akr (Akira Tanaka)**

- Status changed from Open to Closed
- % Done changed from 0 to 100

=begin  
This issue was solved with changeset r29673.  
Eric, thank you for reporting this issue.  
Your contribution to Ruby is greatly appreciated.

May Ruby be with you.

=end

**#5 - 04/29/2011 10:32 AM - kosaki (Motohiro KOSAKI)**

=begin

r29673 caused a regression (see [Bug #4289]). Then, I reverted it by r31378.  
Alternative fixing way is discussed by [Bug #4289] thread.

=end

**#6 - 04/30/2011 03:23 AM - normalperson (Eric Wong)**

=begin

Motohiro KOSAKI [kosaki.motohiro@gmail.com](mailto:kosaki.motohiro@gmail.com) wrote:

r29673 caused a regression (see [Bug #4289]). Then, I reverted it by r31378.  
Alternative fixing way is discussed by [Bug #4289] thread.

I've pushed the ephemeral class patches up to a new branch:

```
$ git pull git://bogomips.org/ruby ephemeral-class
```

```
1) introduce ephemeral class flag for short lived class
2) vm_method.c: ephemeral classes do not write/expire cache
3) IO::Wait*able-extended singleton classes are ephemeral
```

--

Eric Wong

=end

**#7 - 05/17/2011 11:16 PM - akr (Akira Tanaka)**

- *Status changed from Closed to Open*

Sorry for late reply.

I'd like to incorporate 0001-error.c-rb\_mod\_sys\_fail-use-subclass-and-cache.patch.

Although I'm not sure how to view the ephemeral class patches (I don't know git well), I guess it is too intrusive.

**#8 - 05/18/2011 02:59 AM - normalperson (Eric Wong)**

- *File full-ephemeral-class.diff added*

I think subclassing + cache broke some testcases with my updated patch:

<http://redmine.ruby-lang.org/issues/4289#note-5>

The ephemeral class patch series is smaller and cleaner, it makes no user-visible changes.

Somehow, I think I managed to upload the patches to [#4289](#) incorrectly.

I'm uploading a full diff for ephemeral-class since it's small.

Also if you want to try git:

```
# get official mirror
```

```
git clone git://github.com/ruby/ruby.git
```

```
cd ruby
```

```
# add my repo and fetch
```

```
git remote add bogomips git://bogomips.org/ruby
```

```
git fetch bogomips
```

```
# view diff of trunk to my ephemeral-class branch
```

```
git diff origin/trunk bogomips/ephemeral-class
```

```
# view patch series of trunk to my ephemeral-class branch
```

```
git log -p origin/trunk..bogomips/ephemeral-class
```

```
# export patch series of trunk to my ephemeral-class branch
```

# (this will output filenames of mbox patches it makes)  
git format-patch origin/trunk..bogomips/ephemeral-class

I also have a web viewer: [http://bogomips.org/ruby\\_git?h=ephemeral-class](http://bogomips.org/ruby_git?h=ephemeral-class)

**#9 - 05/18/2011 04:23 AM - normalperson (Eric Wong)**

Eric Wong [normalperson@yhbt.net](mailto:normalperson@yhbt.net) wrote:

The ephemeral class patch series is smaller and cleaner, it makes no user-visible changes.

The ephemeral class flag can eventually be expanded for use in other modules, not just I/O ones.

The unique subclass used in timeout.rb to distinguish nested timeouts is one example (especially if we decide to rewrite timeout.rb in C).

--  
Eric Wong

**#10 - 05/19/2011 10:52 PM - akr (Akira Tanaka)**

I don't against for the ephemeral class flag but it needs discussion with ko1 and/or matz.

**#11 - 06/07/2011 02:59 AM - normalperson (Eric Wong)**

Akira Tanaka [akr@fsij.org](mailto:akr@fsij.org) wrote:

I don't against for the ephemeral class flag but it needs discussion with ko1 and/or matz.

Can either of them comment please? I would really like to see this performance regression fixed in 1.9.3.

--  
Eric Wong

**#12 - 06/08/2011 04:51 PM - headius (Charles Nutter)**

What's the effect of the EPHEMERAL flag if someone takes an object with an attached ephemeral class and starts making singleton changes to that object? Do those changes properly flush cache?

If this flag only helps cases where you're extending a module with no methods, it seems extremely niche...why don't we just reverse course on extending these modules at all?

**#13 - 06/09/2011 06:23 AM - normalperson (Eric Wong)**

Charles Nutter [headius@headius.com](mailto:headius@headius.com) wrote:

What's the effect of the EPHEMERAL flag if someone takes an object with an attached ephemeral class and starts making singleton changes to that object? Do those changes properly flush cache?

No, it's a situation where the user must be careful and not shoot themselves in the foot. It is C, after all.

Nowadays since internal.h exists, it would be safer to only expose ephemeral in the new "internal.h" header and not make it part of the public C API.

If this flag only helps cases where you're extending a module with no methods, it seems extremely niche...why don't we just reverse course on extending these modules at all?

This would break code already written for Ruby 1.9.2. Otherwise, I would love to do it (not that I have the power to actually do it).

I absolutely *HATE* the way Ruby extends classes and throws exceptions for *EAGAIN*, but there's not much one can do about it.

A better idea would be to get a kgio-like API into Ruby itself and encourage people to start using that. kgio itself will never take off since it's \*nix-only and written in C, so it should be moved into Ruby and the Ruby spec itself if people really want it (without the ugly "kgio\_" prefixes everywhere).

[1] - <http://bogomips.org/kgio/>

--

Eric Wong

**#14 - 06/09/2011 02:23 PM - headius (Charles Nutter)**

On Wed, Jun 8, 2011 at 4:00 PM, Eric Wong [normalperson@yhbt.net](mailto:normalperson@yhbt.net) wrote:

Charles Nutter [headius@headius.com](mailto:headius@headius.com) wrote:

What's the effect of the EPHEMERAL flag if someone takes an object with an attached ephemeral class and starts making singleton changes to that object? Do those changes properly flush cache?

No, it's a situation where the user must be careful and not shoot themselves in the foot. It is C, after all.

But isn't this an exception object that will be raised into Ruby code?  
In other words...

```
begin
io.read_nonblock
rescue WaitReadable => e
class << self
# add something cute
end
end
```

If this flag only helps cases where you're extending a module with no methods, it seems extremely niche...why don't we just reverse course on extending these modules at all?

This would break code already written for Ruby 1.9.2. Otherwise, I would love to do it (not that I have the power to actually do it).

I absolutely *HATE* the way Ruby extends classes and throws exceptions for EAGAIN, but there's not much one can do about it.

Ok, I'm glad we agree here :) In fact, here's the code that extends WaitReadable in JRuby:

```
// FIXME: oif 1.9 actually does this
if (ruby.is1_9()) {
eagain.getException().extend(new IRubyObject[]
{ruby.getIO().getConstant("WaitReadable")});
}
```

Perhaps I should also express my disapproval through song?

A better idea would be to get a kgio-like API into Ruby itself and encourage people to start using that. kgio itself will never take off since it's \*nix-only and written in C, so it should be moved into Ruby and the Ruby spec itself if people really want it (without the ugly "kgio\_" prefixes everywhere).

At least on the JRuby side of things, I'd love to build this in as a shipping (but nonstandard) library. Java's NIO has similar goals in mind...specifically, if you need to try again on a nonblocking read, it just returns a boolean rather than raising some big heavy error. In fact, kgio may map very well to NIO, at least for the common cases.

Interested in the overhead of this EAGAIN nonsense, I ran a quick benchmark. I include it here for the amusement of all. It demonstrates

pretty clearly the impact of the extend(WaitReadable), since that's really the only thing that differs between the two (at least in JRuby).

```
~/projects/jruby [] ruby -v -rbenchmark -rsocket -e "def
loop_eagain(sock); i = 0; begin; sock.read_nonblock(1); rescue
Errno::EAGAIN; return if i >= 10_000; i+= 1; retry; end; end; 10.times
{ sock = TCPSocket.new('google.com', 80); puts Benchmark.measure {
loop_eagain(sock) } }
"
```

ruby 1.8.7 (2009-06-12 patchlevel 174) [universal-darwin10.0]

```
0.110000 0.020000 0.130000 ( 0.130989)
0.110000 0.020000 0.130000 ( 0.128334)
0.110000 0.020000 0.130000 ( 0.135947)
0.110000 0.020000 0.130000 ( 0.131490)
0.110000 0.020000 0.130000 ( 0.131814)
0.110000 0.020000 0.130000 ( 0.132031)
0.110000 0.020000 0.130000 ( 0.129517)
0.110000 0.020000 0.130000 ( 0.128233)
0.110000 0.020000 0.130000 ( 0.128804)
0.110000 0.020000 0.130000 ( 0.127877)
```

```
~/projects/jruby [] ruby1.9 -v -rbenchmark -rsocket -e "def
loop_eagain(sock); i = 0; begin; sock.read_nonblock(1); rescue
Errno::EAGAIN; return if i >= 10_000; i+= 1; retry; end; end; 10.times
{ sock = TCPSocket.new('google.com', 80); puts Benchmark.measure {
loop_eagain(sock) } }
"
```

ruby 1.9.2p160 (2011-01-16 revision 30579) [x86\_64-darwin10.6.0]

```
0.260000 0.030000 0.290000 ( 0.287646)
0.280000 0.030000 0.310000 ( 0.315121)
0.260000 0.020000 0.280000 ( 0.288908)
0.270000 0.030000 0.300000 ( 0.291922)
0.260000 0.020000 0.280000 ( 0.292273)
0.270000 0.020000 0.290000 ( 0.301361)
0.260000 0.030000 0.290000 ( 0.291552)
0.270000 0.020000 0.290000 ( 0.298062)
0.270000 0.030000 0.300000 ( 0.337271)
0.280000 0.040000 0.320000 ( 0.348292)
```

```
~/projects/jruby [] jruby -v -rbenchmark -rsocket -e "def
loop_eagain(sock); i = 0; begin; sock.read_nonblock(1); rescue
Errno::EAGAIN; return if i >= 10_000; i+= 1; retry; end; end; 10.times
{ sock = TCPSocket.new('google.com', 80); puts Benchmark.measure {
loop_eagain(sock) } }
"
```

jruby 1.7.0.dev (ruby-1.8.7-p330) (2011-06-08 c1029d9) (Java HotSpot(TM) 64-Bit Server VM 1.6.0\_22) [darwin-x86\_64-java]

```
1.102000 0.000000 1.102000 ( 1.051000)
0.583000 0.000000 0.583000 ( 0.583000)
0.607000 0.000000 0.607000 ( 0.607000)
0.120000 0.000000 0.120000 ( 0.120000)
0.119000 0.000000 0.119000 ( 0.119000)
0.123000 0.000000 0.123000 ( 0.123000)
0.113000 0.000000 0.113000 ( 0.113000)
0.120000 0.000000 0.120000 ( 0.120000)
0.124000 0.000000 0.124000 ( 0.124000)
0.117000 0.000000 0.117000 ( 0.117000)
```

```
~/projects/jruby [] jruby --1.9 -v -rbenchmark -rsocket -e "def
loop_eagain(sock); i = 0; begin; sock.read_nonblock(1); rescue
Errno::EAGAIN; return if i >= 10_000; i+= 1; retry; end; end; 10.times
{ sock = TCPSocket.new('google.com', 80); puts Benchmark.measure {
loop_eagain(sock) } }
"
```

jruby 1.7.0.dev (ruby-1.9.2-p136) (2011-06-08 c1029d9) (Java HotSpot(TM) 64-Bit Server VM 1.6.0\_22) [darwin-x86\_64-java]

```
1.965000 0.000000 1.965000 ( 1.964000)
1.369000 0.000000 1.369000 ( 1.369000)
0.712000 0.000000 0.712000 ( 0.712000)
0.567000 0.000000 0.567000 ( 0.566000)
0.208000 0.000000 0.208000 ( 0.208000)
0.209000 0.000000 0.209000 ( 0.209000)
0.204000 0.000000 0.204000 ( 0.204000)
0.207000 0.000000 0.207000 ( 0.207000)
```

0.207000 0.000000 0.207000 ( 0.206000)  
0.213000 0.000000 0.213000 ( 0.212000)

- Charlie

#### #15 - 06/09/2011 05:23 PM - normalperson (Eric Wong)

Charles Oliver Nutter [headius@headius.com](mailto:headius@headius.com) wrote:

On Wed, Jun 8, 2011 at 4:00 PM, Eric Wong [normalperson@yhbt.net](mailto:normalperson@yhbt.net) wrote:

Charles Nutter [headius@headius.com](mailto:headius@headius.com) wrote:

What's the effect of the EPHEMERAL flag if someone takes an object with an attached ephemeral class and starts making singleton changes to that object? Do those changes properly flush cache?

Nevermind, I misread the first time and got ordering of your question mixed up in my mind.

Once a class is tagged RCLASS\_EPHEMERAL, it's impossible for it to write to the cache. There's no need to flush the cache for ephemeral classes because...

No, it's a situation where the user must be careful and not shoot themselves in the foot. ??It is C, after all.

...the the /only/ safe way to use RCLASS\_EPHEMERAL is before any methods are called (and cached) for the singleton class.

But isn't this an exception object that will be raised into Ruby code?  
In other words...

```
begin
  io.read_nonblock
  rescue WaitReadable => e
  class << self
    # add something cute
```

Any methods defined here will never be cached, because RCLASS\_EPHEMERAL was set before we re-entered Ruby-land.

#### #16 - 06/09/2011 05:53 PM - normalperson (Eric Wong)

Charles Oliver Nutter [headius@headius.com](mailto:headius@headius.com) wrote:

Interested in the overhead of this EAGAIN nonsense, I ran a quick benchmark. I include it here for the amusement of all. It demonstrates pretty clearly the impact of the extend(WaitReadable), since that's really the only thing that differs between the two (at least in JRuby).

Since you provided the benchmark code, I reformatted and made a version of it for kgio (see below).

Summary: ephemeral-class performance noticeably (and will have a bigger impact for bigger applications, not small benchmark scripts)

kgio reduces overhead greatly by avoiding exceptions. Real-world results (see the dalli README) are less impressive, of course, but still noticeable.

Results below:

```
== Ruby trunk
ruby 1.9.3dev (2011-06-09 trunk 31961) [x86_64-linux]
0.150000 0.000000 0.150000 ( 0.158108)
0.150000 0.000000 0.150000 ( 0.156812)
```

```
0.150000 0.000000 0.150000 ( 0.157045)
0.150000 0.000000 0.150000 ( 0.156393)
0.150000 0.000000 0.150000 ( 0.156002)
0.150000 0.000000 0.150000 ( 0.159343)
0.150000 0.010000 0.160000 ( 0.159755)
0.150000 0.000000 0.150000 ( 0.158942)
0.150000 0.000000 0.150000 ( 0.158270)
0.150000 0.000000 0.150000 ( 0.158734)
```

```
== git clone git://bogomips.org/ruby.git ephemeral-class
```

```
ruby 1.9.3dev (2011-06-09 trunk 31961) [x86_64-linux]
```

```
loop_eagain (read_nonblock)
```

```
0.100000 0.000000 0.100000 ( 0.101462)
0.080000 0.010000 0.090000 ( 0.100878)
0.100000 0.000000 0.100000 ( 0.100596)
0.100000 0.000000 0.100000 ( 0.101341)
0.100000 0.000000 0.100000 ( 0.100753)
0.100000 0.000000 0.100000 ( 0.099882)
0.090000 0.000000 0.090000 ( 0.100205)
0.100000 0.000000 0.100000 ( 0.099895)
0.100000 0.000000 0.100000 ( 0.101218)
0.090000 0.010000 0.100000 ( 0.100429)
```

```
loop_wait_readable (Kgio::Socket#kgio_tryread)
```

```
0.000000 0.010000 0.010000 ( 0.004570)
0.010000 0.000000 0.010000 ( 0.005165)
0.010000 0.000000 0.010000 ( 0.004236)
0.000000 0.000000 0.000000 ( 0.004767)
0.000000 0.000000 0.000000 ( 0.004186)
0.000000 0.000000 0.000000 ( 0.004813)
0.000000 0.000000 0.000000 ( 0.004186)
0.000000 0.000000 0.000000 ( 0.004755)
0.000000 0.000000 0.000000 ( 0.004168)
0.000000 0.000000 0.000000 ( 0.004199)
```

```
loop_wait_readable (Kgio::Pipe#kgio_tryread)
```

```
0.000000 0.000000 0.000000 ( 0.005383)
0.000000 0.000000 0.000000 ( 0.004787)
0.010000 0.000000 0.010000 ( 0.005284)
0.000000 0.000000 0.000000 ( 0.004784)
0.010000 0.000000 0.010000 ( 0.005311)
0.000000 0.000000 0.000000 ( 0.004770)
0.010000 0.000000 0.010000 ( 0.005299)
0.000000 0.000000 0.000000 ( 0.004811)
0.000000 0.000000 0.000000 ( 0.005347)
0.000000 0.000000 0.000000 ( 0.004770)
```

I added a separate set of tests for Kgio::Pipe vs Kgio::Socket since Kgio::Socket has a small, Linux-only optimization to avoid fcntl() syscalls entirely.

```
# script I used:
```

```
# based on one liner by Charles Nutter [ruby-core:36875]
```

```
# reformatted and added kgio tests
```

```
----- 8< -----
```

```
require 'kgio'
```

```
require 'benchmark'
```

```
def loop_eagain(sock)
```

```
  i = 0
```

```
  begin
```

```
    sock.read_nonblock(1)
```

```
  rescue Errno::EAGAIN
```

```
    return if i >= 10_000
```

```
    i += 1
```

```
  retry
```

```
end
```

```
end
```

```
def loop_wait_readable(sock)
```

```
  i = 0
```

```
  case sock.kgio_tryread(1)
```

```
  when :wait_readable
```

```
    return if i >= 10_000
```

```
    i += 1
```

```
  when String then break # success
```

```
when nil then break # EOF
end while true
end
```

```
host = 'yhbt.net' # yhbt.net webmaster is OK with testing against it
puts "loop_eagain (read_nonblock)"
10.times {
  sock = TCPSocket.new(host, 80)
  puts Benchmark.measure { loop_eagain(sock) }
}
```

```
puts "loop_wait_readable (Kgio::Socket#kgio_tryread)"
addr = Socket.pack_sockaddr_in(80, host) # kgio doesn't do DNS lookups
10.times {
  sock = Kgio::Socket.new(addr)
  puts Benchmark.measure { loop_wait_readable(sock) }
}
```

```
puts "loop_wait_readable (Kgio::Pipe#kgio_tryread)"
addr = Socket.pack_sockaddr_in(80, host) # kgio doesn't do DNS lookups
10.times {
  r, w = Kgio::Pipe.new
  puts Benchmark.measure { loop_wait_readable(r) }
}
```

----- 8< -----

--

Eric Wong

#### #17 - 06/10/2011 08:23 AM - normalperson (Eric Wong)

Eric Wong [normalperson@yhbt.net](mailto:normalperson@yhbt.net) wrote:

A better idea would be to get a kgio-like API into Ruby itself and encourage people to start using that. kgio itself will never take off since it's \*nix-only and written in C, so it should be moved into Ruby and the Ruby spec itself if people really want it (without the ugly "kgio\_" prefixes everywhere).

I've started working on a "try" branch on top of Ruby trunk:

<http://bogomips.org/ruby.git?h=try>

I've only made one commit implementing IO#trywrite:

<http://bogomips.org/ruby.git/commit/?h=try&id=bd0d59fe162d0f1069df4cd3dc86a7a303c4930d>

If there's interest, I'll work on it more (and more frequently) or email patches/pull requests.

--

Eric Wong

#### #18 - 06/10/2011 10:53 AM - ko1 (Koichi Sasada)

Hi,

(2011/06/07 2:55), Eric Wong wrote:

Akira Tanaka [akr@fsij.org](mailto:akr@fsij.org) wrote:

I don't against for the ephemeral class flag but it needs discussion with ko1 and/or matz.

Can either of them comment please? I would really like to see this performance regression fixed in 1.9.3.

Clearing method caching cause the following 2 overheads:

- (1) Clearing overhead
- (2) Cache misses because of clearing methods

Which is your purpose?

For (1), I made an alternative patch:

<http://www.atdot.net/sp/readonly/x8wjml>

For (2), ephemeral class seems good.

--  
// SASADA Koichi at atdot dot net

**#19 - 06/10/2011 01:23 PM - normalperson (Eric Wong)**

SASADA Koichi [ko1@atdot.net](mailto:ko1@atdot.net) wrote:

Clearing method caching cause the following 2 overheads:

- (1) Clearing overhead
- (2) Cache misses because of clearing methods

Which is your purpose?

I used oprofile last year and think I was measuring CPU time, so (1)...

For (1), I made an alternative patch:  
<http://www.atdot.net/sp/readonly/x8wjml>

Awesome! It gives roughly the same performance as my ephemeral class patch in my measurement script below and less intrusive.

Thank you very much for your feedback.

For (2), ephemeral class seems good.

Your patch for (1) improves (2), too. However, I think cache miss is already a huge problem because cache-clearing is called during GC.

"perf -e cache-misses" reported the same results (~910 for either patch vs 1K on unpatched trunk) with a formatted version of the one-liner Charles posted earlier:

```
----- 8< -----
require 'benchmark'
require 'socket'

def loop_eagain(sock)
  i = 0
  begin
    sock.read_nonblock(1)
  rescue Errno::EAGAIN
    return if i >= 10_000
  end
  i += 1
  retry
end

host = 'yhbt.net' # yhbt.net webmaster is OK with testing against it
10.times {
  sock = TCPSocket.new(host, 80)
  puts Benchmark.measure { loop_eagain(sock) }
}
--
Eric Wong
```

**#20 - 06/10/2011 04:53 PM - headius (Charles Nutter)**

On Thu, Jun 9, 2011 at 3:17 AM, Eric Wong [normalperson@yhbt.net](mailto:normalperson@yhbt.net) wrote:

Nevermind, I misread the first time and got ordering of your question mixed up in my mind.

Once a class is tagged RCLASS\_EPHEMERAL, it's impossible for it to write to the cache. There's no need to flush the cache for ephemeral classes because...

...the the /only/ safe way to use RCLASS\_EPHEMERAL is before any methods

are called (and cached) for the singleton class.

```
begin
  io.read_nonblock
  rescue WaitReadable => e
  class << self
    # add something cute
```

Any methods defined here will never be cached, because RCLASS\_EPHEMERAL was set before we re-entered Ruby-land.

Ok, not being familiar with the MRI code, and not seeing more than a few lines of context in the patch, I didn't get this.

So summarizing in non-code:

- Ephemeral class creation does not flush global cache
- ...because ephemeral class methods will never be cached

But I'm confused; if code has already cached a method from an ephemeral class's superclass, and someone adds to the ephemeral class, does the new method get picked up? Hopefully adding methods to an ephemeral class still clears cache, because otherwise invocation won't see such changes. Am I following?

I need to look at ko1's patch since that seems to please you.

- Charlie

#### #21 - 06/10/2011 04:53 PM - headius (Charles Nutter)

On Thu, Jun 9, 2011 at 3:52 AM, Eric Wong [normalperson@yhbt.net](mailto:normalperson@yhbt.net) wrote:

Since you provided the benchmark code, I reformatted and made a version of it for kgio (see below).

Summary: ephemeral-class performance noticeably (and will have a bigger impact for bigger applications, not small benchmark scripts)

kgio reduces overhead greatly by avoiding exceptions. Real-world results (see the dalli README) are less impressive, of course, but still noticeable.

Yeah, I'm not surprised by these numbers. In JRuby the cost of EAGAIN was so great I had to turn off having it generate backtraces; exception backtraces are much more expensive on JVM (and by extension JRuby) so flow control based on exception-handling is absolutely terrible. Only by making EAGAIN backtrace-free could I get reasonable perf. I'd prefer no exception at all for errno's that are nonfatal.

I will have to try doing something like your ephemeral patch in JRuby. For us, extending does not flush the global cache, but still has the cost of creating the singleton class. Not free, but not too bad.

- Charlie

#### #22 - 06/10/2011 06:53 PM - normalperson (Eric Wong)

Charles Oliver Nutter [headius@headius.com](mailto:headius@headius.com) wrote:

On Thu, Jun 9, 2011 at 3:17 AM, Eric Wong [normalperson@yhbt.net](mailto:normalperson@yhbt.net) wrote:

Nevermind, I misread the first time and got ordering of your question mixed up in my mind.

Once a class is tagged RCLASS\_EPHEMERAL, it's impossible for it to write to the cache. There's no need to flush the cache for ephemeral classes because...

...

...the the /only/ safe way to use RCLASS\_EPHEMERAL is before any methods are called (and cached) for the singleton class.

```
begin
  io.read_nonblock
rescue WaitReadable => e
  class << self
    # add something cute
```

Any methods defined here will never be cached, because RCLASS\_EPHEMERAL was set before we re-entered Ruby-land.

Ok, not being familiar with the MRI code, and not seeing more than a few lines of context in the patch, I didn't get this.

So summarizing in non-code:

- Ephemeral class creation does not flush global cache
- ...because ephemeral class methods will never be cached

Yes, to both. Creation of new classes (ephemeral or not) never touches the method cache, only destruction clears the cache. The global method cache includes the (exact) class of each method along with the method ID.

But I'm confused; if code has already cached a method from an ephemeral class's superclass, and someone adds to the ephemeral class, does the new method get picked up? Hopefully adding methods to an ephemeral class still clears cache, because otherwise invocation won't see such changes. Am I following?

Yes a method gets picked up in the ephemeral class if the (non-ephemeral) superclass is modified.

The MRI method cache relies on the class of the calling object matching /exactly/ with the class of the cached method for a hit. Modifying a superclass of any class

I think adding any method anywhere will clear the cache in MRI. But it's not needed for adding methods to ephemeral classes since they'd never have any methods in the cache in the first place.

I need to look at ko1's patch since that seems to please you.

Yes, it's the best patch (along with r28813) I've seen for this issue. Easy to understand, too.

The only thing I can imagine being better is to make uncached method lookup fast enough to where the cache becomes obsolete. I think that would be difficult, though.

--  
Eric Wong

**#23 - 06/10/2011 06:53 PM - normalperson (Eric Wong)**

Eric Wong [normalperson@yhbt.net](mailto:normalperson@yhbt.net) wrote:

For (1), I made an alternative patch:  
<http://www.atdot.net/sp/readonly/x8wjml>

Awesome! It gives roughly the same performance as my ephemeral class patch in my measurement script below and less intrusive.

One possible issue is the VM state counter overflowing. Maybe we should empty the method cache on the rare event of a VM state counter overflow to avoid false positives?

--  
Eric Wong

**#24 - 06/10/2011 06:59 PM - ko1 (Koichi Sasada)**

Hi,

(2011/06/10 13:10), Eric Wong wrote:

For (2), ephemeral class seems good.

Your patch for (1) improves (2), too. However, I think cache miss is already a huge problem because cache-clearing is called during GC.

My patch reduces "clearing method cache" time. And because of my patch clears all cache entries, method cache misses should be increase (in other words, it is not a solution for (2)).

I will solve this problem with other techniques.

First of all, can I commit my patch?

--

// SASADA Koichi at atdot dot net

**#25 - 06/10/2011 07:23 PM - ko1 (Koichi Sasada)**

(2011/06/10 18:50), Eric Wong wrote:

One possible issue is the VM state counter overflowing. Maybe we should empty the method cache on the rare event of a VM state counter overflow to avoid false positives?

Good point. We need to care overflow. However, current MRI lacks this process.

--

// SASADA Koichi at atdot dot net

**#26 - 06/11/2011 03:23 AM - normalperson (Eric Wong)**

SASADA Koichi [ko1@atdot.net](mailto:ko1@atdot.net) wrote:

(2011/06/10 13:10), Eric Wong wrote:

For (2), ephemeral class seems good.

Your patch for (1) improves (2), too. However, I think cache miss is already a huge problem because cache-clearing is called during GC.

My patch reduces "clearing method cache" time. And because of my patch clears all cache entries, method cache misses should be increase (in other words, it is not a solution for (2)).

Oh, I thought you meant CPU cache misses. Method cache miss shouldn't be a problem at all because the class is GC'ed.

I will solve this problem with other techniques.

First of all, can I commit my patch?

Please do (with workaround for overflow). Thank you!

--

Eric Wong

**#27 - 06/12/2011 05:53 PM - ko1 (Koichi Sasada)**

- Status changed from Open to Closed

This issue was solved with changeset r32024.

Eric, thank you for reporting this issue.  
Your contribution to Ruby is greatly appreciated.  
May Ruby be with you.

---

- `vm_method.c (rb_clear_cache*)`: update only vm state version.
- `vm_method.c (rb_method_entry_get_without_cache, rb_method_entry)`: Fill method cache entry with vm state version, and check current vm state version for method (cache) look up. This modification speed-up invalidating of global method cache table. [Ruby 1.9 - Feature [#3905](#)] [ruby-core:36908]

## Files

---

0001-error.c-rb_mod_sys_fail-use-subclass-and-cache.patch	2 KB	10/05/2010	normalperson (Eric Wong)
full-ephemeral-class.diff	2.29 KB	05/18/2011	normalperson (Eric Wong)