

Ruby - Feature #3917

[proposal] called_from() which is much faster than caller()

10/08/2010 08:45 AM - kwatch (makoto kuwata)

Status:	Closed	
Priority:	Normal	
Assignee:	ko1 (Koichi Sasada)	
Target version:	2.0.0	

Description

=begin

I propose to introduce Kernel#called_from() which is similar to caller() but much faster than it.

Background

There are some cases to want to know from where current method is called. In this case, Kernel#caller() is used.

But Kernel#caller() has performance issues for these cases.

- caller() retrieves entire stack frame. It is too heavy.
- caller() returns an array of "filename:linenum in `method'" string. User must parse it and retrieve filename and linenum by rexp. It is also very heavy weight task.

Therefore I propose Kernel#called_from() which is very light weight compared to caller(). A certain benchmark shows that called_from() is more than 20 times faster tan caller().

```
#####
##### Kernel#caller() #####

### Kernel#caller() #####

• caller() #####
• caller()  "filename:line in `method'" #####
#####

#####Kernel#called_from() #####
##### caller() #####
called_from()  caller()  20#####
```

Spec

call-seq:

called_from(start=1) -> array or nil

Returns file name, line number, and method name of the stack. The optional *start* parameter represents the number of stack entries to skip.

Returns +nil+ if *start* is greater than the size of current execution stack.

Raises ArgumentError if *start* is negative value.

Example code

example.rb

```

1: def f1()
2:   f2()
3: end
4: def f2()
5:   f3()
6: end
7: def f3()
8:   p called_from()      #=> ["example.rb", 5, "f2"]
9:   p called_from(0)     #=> ["example.rb", 9, "f3"]

10: p called_from(1)      #=> ["example.rb", 5, "f2"]
11: p called_from(2)      #=> ["example.rb", 2, "f1"]
12: p called_from(3)      #=> ["example.rb", 15, ""]
13: p called_from(4)      #=> nil
14: end
15: f1()

```

Use Case

Case 1: logging method

```

def log_info(message)
  filename, linenum, _ = called_from() # !!!
  @logger.info "#{filename}:#{linenum}: #{message}"
end

```

Case 2: debug print

```

def debug(message)
  filename, linenum, _ = called_from() # !!!
  $stderr.puts "**** DEBUG: #{filename}:#{linenum}: #{message}"
end

```

Case 3: deprecation message

```

def send(args)
  filename, linenum, _ = called_from() # !!!
  msg = "send()' is deprecated. use send() instead."
  msg << " (file: #{filename}, line: #{linenum})"
  $stderr.puts "*** warning: #{msg}"
  send(*args)
end

```

Case 4: ActiveSupport::Testing::Pending

```

module ActiveSupport::Testing::Pending
  def pending(description = "", &block)
    :
    #caller[0] =~ /(.(.):\.(.):\.(.):\.(.))/ # original
    #@@pending_cases << "#{$3} at #{$1}, line #{$2}" # original
    #print "P" # original
    filename, linenum, method = called_from() # !!!
    @@pending_cases << "#{method} at #{filename}, line #{linenum}"
    print "P"
    :
  end
end

```

Case 5: activsupport/lib/active_support/core_ext/module/delegation.rb

```

class Module
  def delegate(*methods)
    :
    #file, line = caller.first.split(':', 2) # original
    #line = line.to_i # original
    file, line, _ = called_from() # !!!

```

```
:
module_eval(<<-EOS, file, line - 5)
:
end
end
```

Case 6: caching helper for template system

```
def cache_with(key)
  data, created_at = @_cache_store.get(key)
  filename, = called_from() # !!!
  ## if template file is newer than cached data then clear cache.
  ## (performance is very important in this case.)
  if created_at < File.mtime(filename)
    data = nil
    @_cache_store.del(key)
  end
  ##
  if data.nil?
    len = @_buf.length
    yield
    data = @_buf[len..-1]
    @_cache_store.set(key, data)
  else
    @_buf << data
  end
  nil
end
```

in template file

```
<% cache_with("orders/#{@order.id}") do %>
```

```
Order ID: <%=h @order.id %>
```

```
Customer: <%=h @order.customer.name %>
```

```
<% end %>
```

Benchmark

Attached benchmark shows that `called_from()` is much faster than `caller()`.
This is very important for logging or template timestamp check.

```
$ ./ruby -s bench.rb -N=100000
```

	user	system	total	real
caller() [0]	1.890000	0.010000	1.900000 (	1.941812)
caller() [0] (retrieve)	2.190000	0.010000	2.200000 (	2.225966)
called_from()	0.100000	0.000000	0.100000 (	0.102810)
called_from() (retrieve)	0.100000	0.000000	0.100000 (	0.102133)

Another Solutions

Adding new global function may be refused.
The followings are another solutions instead of new global function.

- Extend `caller()` to take 'count' parameter.
For example:

```
start = 1
count = 1
caller(start, count) #=> ["filename:linenum in `method`"]
```

- Extend `caller()` to take 'combine' flag.
For example:

```

start = 1
count = nil
combine = false
caller(start, count, combine)
#=> [{"filename", linenum, "method"},
#   ["filename", linenum, "method"],
#   .... ]

```

- Add new standard library 'called_from.so' instead of Kernel#called_from().

```

#####
#####caller()#####called_from()#####
#####
##### Kernel#called_from() ##### called_from.so #####
#####

```

Note

- I tried to implement the above solutions, but failed because vm_backtrace_each() seems to search stack frames in the reverse order of what called_from() requires.
- I can implement called_from() as user library in Ruby 1.8.
http://rubygems.org/gems/called_from
It is allowed to access to stack frame in Ruby 1.8, but no in 1.9.
This is why I submit this propose.
- #####another solutions#####called_from() #####
#####vm_backtrace_each() #####
#####
- Ruby 1.8 #####

http://rubygems.org/gems/called_from
#####1.9#####

=end

Related issues:

Related to Ruby - Feature #1906: Kernel#backtrace: Objectifying Kernel#caller	Closed	08/07/2009
Related to Ruby - Feature #5016: Kernel#caller with negative limit should lim...	Closed	07/11/2011

History

#1 - 10/08/2010 12:38 PM - ko1 (Koichi Sasada)

```

=begin
#####

```

#####alternative #####

(1) caller #####

(a) #####

```

caller(n, m) => n ##### caller #####m #####
caller(0, 1) => ['...']
caller(0, 2) => ['...', '...']

```

```

#####1#####
#####1 #####

```

(b) #####FrameInfo #####

```

caller => [frameinfo1, frameinfo2, ...]

```

```

FrameInfo#type      # iseq->type #####

```


Returns file name, line number, and method name of the stack.
The optional *start* parameter represents the number of stack entries to skip.

Returns `+nil+` if *start* is greater than the size of current execution stack.

Raises `ArgumentError` if *start* is negative value.

Example code

example.rb

```
1: def f1()
2:   f2()
3: end
4: def f2()
5:   f3()
6: end
7: def f3()
8:   p called_from() #=["example.rb", 5, "f2"]
9:   p called_from(0) #=["example.rb", 9, "f3"]
10:  p called_from(1) #=["example.rb", 5, "f2"]
11:  p called_from(2) #=["example.rb", 2, "f1"]
12:  p called_from(3) #=["example.rb", 15, ""]
13:  p called_from(4) #=nil
14: end
15: f1()
```

Use Case

Case 1: logging method

```
def log_info(message)
  filename, linenum, _ =called_from() # !!!
  @logger.info "#{filename}:#{linenum}: #{message}"
end
```

Case 2: debug print

```
def debug(message)
  filename, linenum, _ =called_from() # !!!
  $stderr.puts "**** DEBUG: #{filename}:#{linenum}: #{message}"
end
```

Case 3: deprecation message

```
def send(args)
  filename, linenum, _ =called_from() # !!!
  msg =send()' is deprecated. use send()' instead."
  msg << " (file: #{filename}, line: #{linenum})"
  $stderr.puts "*** warning: #{msg}"
  send(*args)
end
```

Case 4: ActiveSupport::Testing::Pending

```
module ActiveSupport::Testing::Pending
  def pending(description =, &block)
    :
    #caller[0] =~/(.):(.)in `(.*)'/ # original
    #@@pending_cases << "#{$3} at #{$1}, line #{$2}" # original
    #print "P" # original
    filename, linenum, method =called_from() # !!!
    @@pending_cases << "#{method} at #{filename}, line #{linenum}"
    print "P"
    :
  end
end
```

Case 5: ActiveSupport/lib/active_support/core_ext/module/delegation.rb

```
class Module
```

```

def delegate(*methods)
:
#file, line =aller.first.split(':', 2) # original
#line =ine.to_i           # original
file, line, _ =alled_from()      # !!!
:
module_eval(<<-EOS, file, line - 5)
:
end
end

```

Case 6: caching helper for template system

```

def cache_with(key)
data, created_at = _cache_store.get(key)
filename, =alled_from() # !!!
## if template file is newer than cached data then clear cache.
## (performance is very important in this case.)
if created_at < File.mtime(filename)
data =il
@_cache_store.del(key)
end
##
if data.nil?
len = _buf.length
yield
data = _buf[len..-1]
@_cache_store.set(key, data)
else
@_buf << data
end
nil
end

```

in template file

```
<% cache_with("orders/#{@order.id}") do %>
```

```
Order ID: <%= @order.id %>
```

```
Customer: <%= @order.customer.name %>
```

```
<% end %>
```

Benchmark

Attached benchmark shows that called_from() is much faster than caller().
This is very important for logging or template timestamp check.

```

$ ./ruby -s bench.rb -N0000

```

	user	system	total	real
caller() [0]	1.890000	0.010000	1.900000	(1.941812)
caller() [0] (retrieve)	2.190000	0.010000	2.200000	(2.225966)
called_from()	0.100000	0.000000	0.100000	(0.102810)
called_from() (retrieve)	0.100000	0.000000	0.100000	(0.102133)

Another Solutions

Adding new gobal function may be refused.
The followings are another solutions instead of new global function.

- Extend caller() to take 'count' parameter.
For example:

```

start =
count =
caller(start, count) #=["filename:linenum in `method`"]

```

- Extend caller() to take 'combine' flag.
For example:

```

start =
count =il

```

```

combine = false
caller(start, count, combine)
#=[["filename", lineno, "method"],
#  ["filename", lineno, "method"],
#  .... ]

```

- Add new standard library 'called_from.so' instead of Kernel#called_from().

```

#####
##### caller() ##### called_from() #####
#####
##### Kernel#called_from() ##### called_from.so #####
#####

```

Note

- I tried to implement the above solutions, but failed because vm_backtrace_each() seems to search stack frames in the reverse order of what called_from() requires.
- I can implement called_from() as user library in Ruby 1.8.
http://rubygems.org/gems/called_from
 It is allowed to access to stack frame in Ruby 1.8, but no in 1.9.
 This is why I submit this propose.
- ##### another solutions ##### called_from() #####
 ##### vm_backtrace_each() #####
 #####
- Ruby 1.8 #####
 #####
http://rubygems.org/gems/called_from
 ##### 1.9 #####
 #####

<http://redmine.ruby-lang.org>

```

--
// SASADA Koichi at atdot dot net
=end

```

#2 - 10/10/2010 01:25 AM - kwatch (makoto kuwata)

```

=begin
#####

```

2010/10/8 SASADA Koichi ko1@atdot.net:

#####

alternative

(1) caller #####

(a) #####

```

caller(n, m) => n ##### caller ##### m #####
caller(0, 1) => [...]
caller(0, 2) => [..., '...']

```

```

##### proposal ##### caller #####
#####
##### caller #####

```

+

```

##### 'first' #####
##### caller() ##### called_from() #####
##### caller() #####

```

```

00000000000000000000
000000000000000000000000000000000000
0000000000000000

```

```

01 000000000000000000000000000000000000

```

```

000000000000000000000000000000000000

```

```

(b) 000000FrameInfo 00000000000000000000

```

```

caller => [frameinfo1, frameinfo2, ...]

```

```

FrameInfo#type # iseq->type 000000000000
FrameInfo#name # iseq->name 000000000000
FrameInfo#line_no # 000000000000
FrameInfo#filename # 000000000000

```

```

FrameInfo#to_str 000000000000000000000000
00000000000000000000

```

```

iseq 000000000000000000000000

```

```

FrameInfo#binding 000000caller_binding 000000000000
000000000000

```

```

0CallerEntry 00000000000000000000

```

```

(2) called_from 0000 FrameInfo 0000000000

```

```

FrameInfo 000000000000

```

```

000000000000000000000000000000000000

```

```

000000

```

```

vm_backtrace_each 0000000000000000 iseq 00000000
000000000000000000000000000000000000

```

```

000000

```

```

--
makoto kuwata

```

```

=end

```

#3 - 10/29/2010 03:56 AM - runpaint (Run Paint Run Run)

```

=begin
I can't follow this discussion, but I'll note proposal #1906 that also aims to provide an alternative to #caller. (Oddly, I, too, hacked on
vm_backtrace_each(), and related functions, to try to implement the aforementioned). I'd prefer a Hash/Struct to be returned, rather than an Array,
and favour #backtrace or #caller taking a count argument rather than defining a new top-level method for such a narrow use.
=end

```

#4 - 10/29/2010 11:21 AM - naruse (Yui NARUSE)

```

- Status changed from Open to Assigned
- Assignee set to ko1 (Koichi Sasada)

```

```

=begin
ko1 changes implementation around iseq and may change the order of vm_backtrace_each,
so he want us to wait for completion those changes.
=end

```

#5 - 11/06/2010 02:21 PM - kwatch (makoto kuwata)

```

=begin
Hi Run Run, thank you for your comment about Kernel#caller.

```

```

2010/10/29 Run Paint Run Run redmine@ruby-lang.org:

```

```

I can't follow this discussion, but I'll note proposal #1906 that

```

also aims to provide an alternative to #caller. (Oddly, I, too, hacked on vm_backtrace_each(), and related functions, to try to implement the aforementioned). I'd prefer a Hash/Struct to be returned, rather than an Array, and favour #backtrace or #caller taking a count argument rather than defining a new top-level method for such a narrow use.

Ko1 has the same idea as what you proposed on [#1906](#).
He want to change return value from "filename:linenum: in `name`"
to structured data type which is very similar to what you proposed.

And I already proposed to add new argument to Kernel#caller() in order to specify backtrace depth, instead of adding new top-level function.
Please check out 'Another Solutions' section on this ticket.

Ko1, could you read [#1906](#) if you change return value of caller()?

--
regards,
makoto kuwata

=end

#6 - 07/12/2011 05:13 PM - ddebernardy (Denis de Bernardy)

I'm not making much sense of the japanese in this ticket. Is this (or [#1906](#), which also looks neat) anything that might make it into ruby 1.9.3? I was wondering how to get the calling file's name earlier today without resorting to caller() -- which yields an unnecessarily large string array.

As an aside, there's this sender gem written in C here, in the meanwhile:

<https://github.com/Asher-/sender>

#7 - 07/13/2011 11:23 AM - Anonymous

Feature [#3917](#): [proposal] called_from() which is much faster than caller()
<http://redmine.ruby-lang.org/issues/3917>

+1 for this one--I was wishing for this the other day (or for caller
to give you depth starting from the current method...)
-roger-

#8 - 06/26/2012 04:46 AM - ko1 (Koichi Sasada)

- Description updated
- Status changed from Assigned to Feedback

I made caller_locations() (see [\[ruby-core:45253\]](#) [RFC] RubyVM::FrameInfo.caller method).

Maybe the last problem is naming of this method.

#9 - 06/26/2012 05:56 AM - trans (Thomas Sawyer)

#caller_locations, is this what I just (re)proposed in [#6646](#)? Sorry if so, I can't read Japanese :(

Maybe better name: #callstack ?

#10 - 10/27/2012 06:02 AM - ko1 (Koichi Sasada)

- Status changed from Feedback to Closed

I close this ticket.
Let's discuss naming issue with
<https://bugs.ruby-lang.org/issues/7051>

Files

0001-add-Kernel-called_from-which-is-similar-to-caller.patch	3.7 KB	10/08/2010	kwatch (makoto kuwata)
0002-add-test-script-for-Kernel-called_from.patch	2.21 KB	10/08/2010	kwatch (makoto kuwata)
bench.rb	1.14 KB	10/08/2010	kwatch (makoto kuwata)