Ruby trunk - 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().

現在のメソッドがどこから呼び出されたかを知りたい場合がときどきある。
こういう場合は通常 Kernel#caller() が使われる。

しかし Kernel#caller() は、こういった用途ではパフォーマンスが非常に悪い。

- caller() はスタックフレームをすべて取り出す。これは非常に重い操作。
- caller() は "ファイル名:行番号 in `メソッド名'" という文字列の配列を返す。ユーザは正規表現を使ってこの文字列をわざわざパースしなければならない。これも重い操作。

そのため、Kernel#called_from() を追加することを提案する。
このメソッドは caller() と比べて非常に動作が軽く、ベンチマークでは
called_from() は caller() と比べて20倍以上高速。

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::Peding
def pending(description = "", &block)
:
#caller[0] =~ (/(.):(.):in `(.*)'/)             # original
#@@pending_cases << "#{$3} at #{$1}, line #{$2}"  # original
#print "P"                                      # original
filenemae, 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 = 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 gobal 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 'conbine' flag.
  For example:

  ```
  start = 1
  count = nil
  conbine = false
  caller(start, count, conbine)
  #=> [["filename", linenum, "method"],
  #    ["filename", linenum, "method"],
  ```

```
#    .... ]
```

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

新しいグローバル関数を導入するのは拒絶される可能性が高い。
その場合は、caller()を拡張してcalled_from()相当のことができるように
してもらえるとうれしい。
あるいは Kernel#called_from() ではなくても called_from.so を標準添付
する方針でもいい。

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.

- 実は上記のanother solutionsを実装しようとしたが、called_from() では
  直近のスタックフレームから辿りたいのに対し、vm_backtrace_each() は
  逆の順番で辿ることしかできないようなので、実装を諦めた。

- Ruby 1.8 では拡張モジュールからスタックフレームにアクセスできるので
  ライブラリとして実装した。
  http://rubygems.org/gems/called_from
  けど1.9ではスタックフレームへのアクセスができないので、ライブラリが
  作れない。そのため今回このような提案をしてみた。
  =end

Related issues:

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

History

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

=begin
□ささだです。

□基本的に賛成なんですが、alternative なアイデアということで。

(1) caller の拡張案

(a) 配列の要素数を指定するオプショナル引数を加える

```
caller(n, m) => n 個上の caller から、m 個だけ情報を取り出す
caller(0, 1) => ['...']
caller(0, 2) => ['...', '...']
```

□＊一個しかほしくない時に、1要素の配列になってしまうのが嬉しくない。
□＊1 を指定したら配列じゃ無くす、というのは、きっと良くないんだろうな。

□(b) 配列の要素をFrameInfo オブジェクトとかいうものをでっちあげる

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

```
FrameInfo#type     # iseq->type 相当（でいいのか？）
FrameInfo#name     # iseq->name 相当、メソッド名とか
FrameInfo#line_no  # これまでと同じ行番号
FrameInfo#filename # これまでと同じファイル名
```

FrameInfo#to_str を定義することで、文字列を期待していた今までのコード
もそのまま動く（ほんとかな？）。

□iseq を持たせるような実装になるのかしらん。

□FrameInfo#binding とかやると、caller_binding みたいな話になるんだけ
ど、これはやらない。

□＊CallerEntry とかの名前のほうがいいのかしらん。

(2) called_from の返値を FrameInfo みたいなのにする

□FrameInfo の定義は上記の通り。

□以上です。

□vm_backtrace_each の順番は変えちゃうかも。でも、今 iseq 周りの実装をそ
う取っ替えしているので、この変更はもうちょっと待ってほしいかも。

日本語が混ざってると、つい日本語を追っちゃって、
意外と読みづらいという現象を発見しました…。

(2010/10/08 0:46), makoto kuwata wrote:

> Feature [#3917](http://redmine.ruby-lang.org/issues/show/3917): [proposal] called_from() which is much faster than caller()
> http://redmine.ruby-lang.org/issues/show/3917
>
> 起票者: makoto kuwata
> ステータス: Open, 優先度: Normal
> カテゴリ: core, Target version: 1.9.x
>
> 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().
>
> 現のメソッドがどこから呼び出されたかを知りたい場合がときどきある。
> こういう場合は通常 Kernel#caller() が使われる。
>
> しかし Kernel#caller() は、こういった用途ではパフォーマンスが非常に悪い。
>
> - caller() はスタックフレームをすべて取り醜。これは非常に重い操作。
> - caller() は "ファイル名:行番号 in `メソド名'" という文字列の配列を返す。ユーザは正規表を使ってこの文字列をわざわざパースしなければならない。これも修操作。
>
> そため、Kernel#called_from() を追加することを提案する。
> このメソドは caller() と比べて非常に動作が軽く、ベンチマークでは
> called_from() は caller() と比べて20倍以上高速。
>
> ## Spec
>
> call-seq:
> called_from(start=   -> array or nil
>
> Returns file name, line number, and method name of the stack.
> The optional <u>start</u> parameter represents the number of stack
> entries to skip.
>
> Returns +nil+ if <u>start</u> is greater than the size of
> current execution stack.
>
> Raises ArgumentError if <u>start</u> 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, _ =alled_from()   # !!!
@logger.info "#{filename}:#{linenum}: #{message}"
end
```

Case 2: debug print

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

Case 3: deprecation message

```
def send(args)
filename, linenum, _ =alled_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::Peding
def pending(description ='', &block)
:
#caller[0] =(/(.):(.):in `(.*)'/)              # original
#@@pending_cases << "#{$3} at #{$1}, line #{$2}"  # original
#print "P"                                      # original
filenemae, linenum, method =alled_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 'conbine' flag.
  For example:

  ```
  start =
  count =il
  conbine =alse
  caller(start, count, conbine)
  #=[["filename", linenum, "method"],
  #    ["filename", linenum, "method"],
  #    .... ]
  ```

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

新しいグローバル関数を導入するのは拒絶される可能が高い。
そ場合は、caller()を拡張してcalled_from()相当のことができるように
してもらえるとうれしい。
あるいは Kernel#called_from() ではなくても called_from.so を標什付
する方針でもいい。

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.
>
> - 実は上記のanother solutionsを実装しようとしたが、called_from() では
>   直近のスタックフレームから辿りたいのに対し、vm_backtrace_each() は
>   逆の瞬で辿ることしかできないようなので、実装を諦めた。
>
> - Ruby 1.8 では拡張モジュールからスタックフレームにアクセスできるので
>   ライブラリとして実装した。
>   http://rubygems.org/gems/called_from
>   けど1.9ではスタックフレームへのアクセスができないので、ライブラリが
>   作れない。そため今回このような提案をしてみた。
>
> ---
>
> http://redmine.ruby-lang.org

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

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

=begin
桑田です。日本語が文字化けしてたようで申し訳ないです。

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

> □ささだです。
>
> 基本的に賛成なんですが、alternative なアイデアということで。
>
> (1) caller の拡張案
>
> (a) 配列の要素数を指定するオプショナル引数を加える
>
> caller(n, m) => n 個上の caller から、m 個だけ情報を取り出す
> caller(0, 1) => ['...']
> caller(0, 2) => ['...', '...']

もとのproposalにもcallerの拡張案を載せていますが、
それと同じという解釈でいいでしょうか。
私としてはcallerの拡張でも結構です。

> ＊一個しかほしくない時に、1要素の配列になってしまうのが嬉しくない。

確か似そうですが、さほど困るほどでもないと思います。'.first' を付けるだけだし。
もともと、caller()やcalled_from()は主にライブラリ作成者が使うことが多く、
ライブラリ使用者であるユーザがcaller()を直接使う頻度は少ないので、
多少使いにくくてもいいかなと思います。
それより戻り値に一貫性があることや、妙なルールが導入されないことの
ほうが重要だと思いました。

> ＊1 を指定したら配列じゃ無くす、というのは、きっと良くないんだろうな。

そうですね、戻り値に一貫性がなくなるのはあまりよくないと思います。

> (b) 配列の要素をFrameInfo オブジェクトとかいうものをでっちあげる
>
> caller => [frameinfo1, frameinfo2, ...]
>
> FrameInfo#type　# iseq->type 相当（でいいのか？）
> FrameInfo#name　# iseq->name 相当、メソッド名とか
> FrameInfo#line_no　# これまでと同じ行番号
> FrameInfo#filename # これまでと同じファイル名
>
> FrameInfo#to_str を定義することで、文字列を期待していた今までのコード

> もそのまま動く（ほんとかな? ）。
>
> iseq を持たせるような実装になるのかしらん。
>
> FrameInfo#binding とかやると、caller_binding みたいな話になるんだけど、これはやらない。
>
> ＊CallerEntry とかの名前のほうがいいのかしらん。
>
> (2) called_from の返値を FrameInfo みたいなのにする
>
> FrameInfo の定義は上記の通り。

これについては私のほうでは善し悪しが判断できないので、開発陣の判断にお任せします。

> 以上です。
>
> vm_backtrace_each の順番は変えちゃうかも。でも、今 iseq 周りの実装をそう取っ替えしているので、この変更はもうちょっと待ってほしいかも。

了解です。

--
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)