Ruby master - Feature #14329

Speedup `block.call` where `block` is passed block parameter.

01/07/2018 04:49 PM - ko1 (Koichi Sasada)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>ko1 (Koichi Sasada)</td>
</tr>
<tr>
<td>Target version:</td>
<td>2.6</td>
</tr>
</tbody>
</table>

Description

abstract

Speedup block.call where block is passed block parameter with a special object named "proxyblock" which responds to call and invoke passed block.

background

Ruby 2.5 improved performance of passing a passed block parameter by lazy Proc creation ([Feature #14045]). However, block.call (block is a passed block parameter) is not optimized and need to create a Proc object immediately.

proposal

We need to make Proc creation lazily for performance. There are several way to achieve it, but I propose to use special object named "blockproxy" object.

This is a pseudo code to use it:

```ruby
# block is given block parameter
block.call(1, 2, 3)
#=> translate at compile time
if block is not modified and
   block is ISeq/IFnuc block
   tmp = blockproxy
else
   tmp = block # create Proc and so on
end
tmp.call(1, 2, 3)
```

blockproxy.call invoke given block if Proc#call is not redefined, otherwise make a Proc and call Proc#call as usual.

Advantage of this method is we can also use this technique with the safe navigation operator (block&.call). If block is not given, then tmp will be nil, and no method dispatched with &..

Note that this technique depends on the assumption "we can't access to the evaluated receiver just before method dispatch". We don't/can't access blockproxy object at method dispatch, and no compatibility issue.

evaluation

Using [https://github.com/k0kubun/benchmark_driver](https://github.com/k0kubun/benchmark_driver)

prelude: |
```ruby
def block_yield
  yield
end

bp_yield &b
```
b.call
end
def bp_safe_call &b
  b&.call
end

benchmark:
- block_yield{}
- bp_yield{}
- bp_call{}
- bp_safe_call{}
- bp_safe_call

Result:
Warming up --------------------------------------
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>block_yield{}</td>
<td>1.298M</td>
<td>1/100ms</td>
</tr>
<tr>
<td>bp_yield{}</td>
<td>1.177M</td>
<td>1/100ms</td>
</tr>
<tr>
<td>bp_call{}</td>
<td>447.723k</td>
<td>1/100ms</td>
</tr>
<tr>
<td>bp_safe_call{}</td>
<td>413.261k</td>
<td>1/100ms</td>
</tr>
<tr>
<td>bp_safe_call</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Calculating -------------------------------------
<table>
<thead>
<tr>
<th></th>
<th>trunk</th>
<th>modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>block_yield{}</td>
<td>20.672M</td>
<td>20.808M</td>
</tr>
<tr>
<td>bp_yield{}</td>
<td>16.294M</td>
<td>16.220M</td>
</tr>
<tr>
<td>bp_call{}</td>
<td>5.626M</td>
<td>14.752M</td>
</tr>
<tr>
<td>bp_safe_call{}</td>
<td>5.555M</td>
<td>14.557M</td>
</tr>
<tr>
<td>bp_safe_call</td>
<td>31.339M</td>
<td>23.561M</td>
</tr>
</tbody>
</table>

The patch is here:
https://gist.github.com/ko1/d8a1a9d92075b27a8e95ca528cc57fd2

History
#1 - 01/07/2018 04:57 PM - ko1 (Koichi Sasada)
- Status changed from Open to Closed

move to ruby-core.