String#strip! raises RuntimeError on Frozen String Despite Making No Changes

06/01/2009 08:52 PM - runpaint (Run Paint Run Run)

Status: Closed
Priority: Normal
Assignee:
Target version: 2.0.0

ruby -v:
ruby 1.9.2dev (2009-05-28 trunk 23601) 
[i686-linux]

Description
=begin
Calling String#strip! on a frozen string raises a RuntimeError even if the string was not changed. String#rstrip! doesn't raise an exception in this scenario. I believe that the latter behaviour is correct; #strip! should only raise a RuntimeError if the string would be changed.

$ rubybleed -ve ""ruby".freeze.strip!"
ruby 1.9.2dev (2009-05-28 trunk 23601) [i686-linux]
-e:1:in strip!: can't modify frozen string (RuntimeError)
from -e:1:in'

$ rubybleed -ve ""ruby".freeze.rstrip!"
ruby 1.9.2dev (2009-05-28 trunk 23601) [i686-linux]

1.9.1 behaves the same as 1.9.2. 1.8.7 behaves correctly.
=end

Associated revisions
Revision 58880 - 05/25/2017 04:25 AM - watson1978 (Shizuo Fujita)
Improve performance of rb_equal()

- object.c (rb_equal): add optimized path to compare the objects using rb_equal_opt(). Previously, if not same objects were given, rb_equal() would call ""=="" method via rb_funcall() which took a long time.

  rb_equal_opt() has provided faster comparing for Fixnum/Float/String objects. Now, Time#eql? uses rb_equal() to compare with argument object and it will be faster around 40% on 64-bit environment.

- array.c (rb_ary_index): remove redundant rb_equal_opt() calling.

  Now, rb_equal() was optimized using rb_equal_opt(). If rb_equal_opt() returns Qundef, it will invoke rb_equal() -> rb_equal_opt(), and it will cause the performance regression.

  So, this patch will remove first redundant rb_equal_opt() calling.

- array.c (rb_ary_rindex): ditto.

- array.c (rb_ary_includes): ditto.

[ruby-core:80360] [Bug #13365] [Fix GH-#1552]

Before

<table>
<thead>
<tr>
<th>Method</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time#eql? with other</td>
<td>7.309M</td>
<td>10.321M</td>
</tr>
<tr>
<td>Array#include?(val)</td>
<td>1.433M</td>
<td>1.474M</td>
</tr>
<tr>
<td>Array#include?(val)</td>
<td>1.418M</td>
<td>1.451M</td>
</tr>
</tbody>
</table>

After

<table>
<thead>
<tr>
<th>Method</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time#eql? with other</td>
<td>10.321M</td>
<td>51.684M</td>
</tr>
<tr>
<td>Array#include?(val)</td>
<td>1.474M</td>
<td>7.439M</td>
</tr>
</tbody>
</table>
Revision 58880 - 05/25/2017 04:25 AM - watson1978 (Shizuo Fujita)

Improve performance of rb_equal()

- object.c (rb_equal): add optimized path to compare the objects using rb_equal_opt(). Previously, if not same objects were given, rb_equal() would call `==` method via rb_funcall() which took a long time.

  rb_equal_opt() has provided faster comparing for Fixnum/Float/String objects. Now, Time#eql? uses rb_equal() to compare with argument object and it will be faster around 40% on 64-bit environment.

- array.c (rb_ary_index): remove redundant rb_equal_opt() calling.
  Now, rb_equal() was optimized using rb_equal_opt().
  If rb_equal_opt() returns Qundef, it will invoke rb_equal() -> rb_equal_opt(), and it will cause the performance regression.

  So, this patch will remove first redundant rb_equal_opt() calling.

- array.c (rb_ary_index): ditto.

- array.c (rb_ary_includes): ditto.

[ruby-core:80360] [Bug #13365] [Fix GH-#1552]
require 'benchmark/ips'

Benchmark.ips do |x|
  t1 = Time.now
  t2 = Time.now

  x.report "Time#eql? with other" do |i|
    i.times { t1.eql?(t2) }
  end

  # Benchmarks to check whether it didn't introduce the regression
  obj = Object.new
  x.report "Array#index(val)" do |i|
    ary = [1, 2, true, false, obj]
    i.times { ary.index(obj) }
  end

  x.report "Array#rindex(val)" do |i|
    ary = [1, 2, true, false, obj].reverse
    i.times { ary.rindex(obj) }
  end

  x.report "Array#include?(val)" do |i|
    ary = [1, 2, true, false, obj]
    i.times { ary.include?(obj) }
  end
end

Revision 58880 - 05/25/2017 04:25 AM - watson1978 (Shizuo Fujita)

Improve performance of rb_equal()

- object.c (rb_equal): add optimized path to compare the objects using rb_equal_opt(). Previously, if not same objects were given, rb_equal() would call `==' method via rb_funcall() which took a long time.

  rb_equal_opt() has provided faster comparing for Fixnum/Float/String objects. Now, Time#eql? uses rb_equal() to compare with argument object and it will be faster around 40% on 64-bit environment.

- array.c (rb_ary_index): remove redundant rb_equal_opt() calling.
  Now, rb_equal() was optimized using rb_equal_opt().
  If rb_equal_opt() returns Qundef, it will invoke rb_equal() -> rb_equal_opt(), and it will cause the performance regression.

  So, this patch will remove first redundant rb_equal_opt() calling.

- array.c (rb_ary_rindex): ditto.

- array.c (rb_ary_includes): ditto.

[ruby-core:80360] [Bug #13365] [Fix GH-#1552]

Before

Time#eql? with other  7.309M (± 1.4%) i/s - 36.647M in  5.014964s
Array#index(val)  1.433M (± 1.2%) i/s -  7.207M in  5.030942s
Array#index(val)  1.418M (± 1.6%) i/s -  7.103M in  5.009164s
Array#include?(val)  1.451M (± 0.9%) i/s -  7.295M in  5.026392s

After

Time#eql? with other  10.321M (± 1.9%) i/s - 51.684M in  5.009203s
Array#index(val)  1.474M (± 0.9%) i/s -  7.433M in  5.044384s
Array#index(val)  1.449M (± 1.7%) i/s -  7.292M in  5.034436s
Array#include?(val)  1.466M (± 1.7%) i/s -  7.373M in  5.030047s

Test code

require 'benchmark/ips'

Benchmark.ips do |x|
  t1 = Time.now
end

09/26/2021
t2 = Time.now

x.report "Time#eql? with other" do |i|
  i.times { t1.eql?(t2) }
end

# Benchmarks to check whether it didn't introduce the regression
obj = Object.new
x.report "Array#index(val)" do |i|
  ary = [1, 2, true, false, obj]
  i.times { ary.index(obj) }
end

x.report "Array#rindex(val)" do |i|
  ary = [1, 2, true, false, obj].reverse
  i.times { ary.rindex(obj) }
end

x.report "Array#include?(val)" do |i|
  ary = [1, 2, true, false, obj]
  i.times { ary.include?(obj) }
end

History

#1 - 06/04/2009 02:51 AM - matz (Yukihiro Matsumoto)
- Status changed from Open to Closed

=begin
changed to the opposite way for consistency.
=end