Ruby master - Feature #8206

Should Ruby core implement String#blank?

04/03/2013 09:32 AM - sam.saffron (Sam Saffron)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

There has been some discussion about porting the #blank? protocol over to Ruby in the past that has been rejected by Matz.

This proposal is only about String however.

At the moment to figure out if you have a blank string you would

```
"  ".strip.length == 0
```

The disadvantage is that this forces unneeded allocations and does too much work:

An optimal implementation would be:

```
static VALUE
rb_str_blank(VALUE str)
{
  rb_encoding *enc;
  char *s, *e;

  enc = STR_ENC_GET(str);
  s = RSTRING_PTR(str);
  if (!s || RSTRING_LEN(str) == 0) return Qtrue;

  e = RSTRING_END(str);
  while (s < e) {
      int n;
      unsigned int cc = rb_enc_codepoint_len(s, e, &n, enc);

      if (!rb_isspace(cc) && cc != 0) return Qfalse;
    s += n;
  }
  return Qtrue;
}
```

This in turn is about 5-8x than the regex solution to the problem and way faster than allocating one massive string with strip when length is large.

Should Ruby take on this method, to accompany #strip following its practice.

A slight caveat though is that active support has a somewhat different definition of blank?

```
const unsigned int as_blank[26] = {9, 0xa, 0xb, 0xc, 0xd,
  0x20, 0x85, 0xa0, 0x1680, 0x180e, 0x2000, 0x2001,
  0x2002, 0x2003, 0x2004, 0x2005, 0x2006, 0x2007, 0x2008,
  0x2009, 0x200a, 0x2028, 0x2029, 0x202f, 0x205f, 0x3000
};

static VALUE
rb_str_blank_as(VALUE str)
{
  rb_encoding *enc;
  char *s, *e;
  int i;
```

```
  int found;

  enc = STR_ENC_GET(str);
  s = RSTRING_PTR(str);
  if (!s || RSTRING_LEN(str) == 0) return Qtrue;

  e = RSTRING_END(str);
  while (s < e) {
      int n;
      unsigned int cc = rb_enc_codepoint_len(s, e, &n, enc);

    found = 0;
    for(i=0;i<26;i++){
      unsigned int current = as_blank[i];
      if(current == cc) {
        found = 1;
        break;
      }
      if(cc < current){
        break;
      }
    }

      if (!found) return Qfalse;
    s += n;
  }
  return Qtrue;
}
```

Clearly it makes no sense to have such a method.

If Ruby took over implementing String#blank? it would clash with Active Support. But imho would enforce better API consistency.

Thoughts?

Related issues:

Related to Ruby master - Feature #8110: Regex methods not changing global var...	**Closed**
Has duplicate Ruby master - Feature #12306: Implement String #blank? #present...	**Open**

History

#1 - 04/03/2013 11:18 AM - marcandre (Marc-Andre Lafortune)

Your rb_str_blank is 5-8x faster than regexp? You compared it to /\A[[:space:]]*\z/ =~ str?

#2 - 04/03/2013 11:21 AM - headius (Charles Nutter)

marcandre (Marc-Andre Lafortune) wrote:

> Your rb_str_blank is 5-8x faster than regexp? You compared it to /\A[[:space:]]*\z/ =~ str?

Regexp matches construct a MatchData and set $~. blank? would do neither, and have no allocation cost whatsoever.

#3 - 04/03/2013 01:55 PM - sam.saffron (Sam Saffron)

marcandre (Marc-Andre Lafortune) I tried pretty much every combination possible interestingly depending on the string /\A[[:space:]]*\z/ can be slower than the original regex, also afaik its not identical cause it misses some cases

#4 - 04/04/2013 02:55 AM - naruse (Yui NARUSE)

I came up with an idea, String#include? with regexp without backref.
Could you try and comment this?

```
% ruby -e'p [$&," foo".include?(/[[:space:]]/),$&]'
[nil, true, nil]
```

```
diff --git a/re.c b/re.c
index 16d7e34..8c7d9de 100644
--- a/re.c
+++ b/re.c
```

```
@@ -1352,18 +1352,19 @@ rb_reg_adjust_startpos(VALUE re, VALUE str, long pos, int reverse)
 }

 /* returns byte offset */
-long
-rb_reg_search(VALUE re, VALUE str, long pos, int reverse)
+static long
+rb_reg_search0(VALUE re, VALUE str, long pos, int reverse, int backref)
 {
     long result;
     VALUE match;
-    struct re_registers regi, *regs = &regi;
+    struct re_registers regi;
+    struct re_registers *regs = NULL;
     char *range = RSTRING_PTR(str);
-    regex_t *reg;
+    regex_t *reg = NULL;
     int tmpreg;

     if (pos > RSTRING_LEN(str) || pos < 0) {
-   rb_backref_set(Qnil);
+   if (backref) rb_backref_set(Qnil);
    return -1;
     }

@@ -1371,18 +1372,21 @@ rb_reg_search(VALUE re, VALUE str, long pos, int reverse)
     tmpreg = reg != RREGEXP(re)->ptr;
     if (!tmpreg) RREGEXP(re)->usecnt++;

-    match = rb_backref_get();
-    if (!NIL_P(match)) {
-   if (FL_TEST(match, MATCH_BUSY)) {
-       match = Qnil;
+    if (backref) {
+   regs = &regi;
+   match = rb_backref_get();
+   if (!NIL_P(match)) {
+       if (FL_TEST(match, MATCH_BUSY)) {
+       match = Qnil;
+       }
+       else {
+       regs = RMATCH_REGS(match);
+       }
    }
-   else {
-       regs = RMATCH_REGS(match);
+   if (NIL_P(match)) {
+       MEMZERO(regs, struct re_registers, 1);
    }
     }
-    if (NIL_P(match)) {
-   MEMZERO(regs, struct re_registers, 1);
-    }
     if (!reverse) {
    range += RSTRING_LEN(str);
     }
@@ -1416,29 +1420,44 @@ rb_reg_search(VALUE re, VALUE str, long pos, int reverse)
    }
     }

-    if (NIL_P(match)) {
-   match = match_alloc(rb_cMatch);
-   onig_region_copy(RMATCH_REGS(match), regs);
-   onig_region_free(regs, 0);
-    }
-    else {
-   if (rb_safe_level() >= 3)
-       OBJ_TAINT(match);
-   else
-       FL_UNSET(match, FL_TAINT);
-    }
+    if (backref) {
+   if (NIL_P(match)) {
+       match = match_alloc(rb_cMatch);
+       onig_region_copy(RMATCH_REGS(match), regs);
```

```
+        onig_region_free(regs, 0);
+    }
+    else {
+        if (rb_safe_level() >= 3)
+        OBJ_TAINT(match);
+        else
+        FL_UNSET(match, FL_TAINT);
+    }

-     RMATCH(match)->str = rb_str_new4(str);
-     RMATCH(match)->regexp = re;
-     RMATCH(match)->rmatch->char_offset_updated = 0;
-     rb_backref_set(match);
+    RMATCH(match)->str = rb_str_new4(str);
+    RMATCH(match)->regexp = re;
+    RMATCH(match)->rmatch->char_offset_updated = 0;
+    rb_backref_set(match);

-     OBJ_INFECT(match, re);
-     OBJ_INFECT(match, str);
+    OBJ_INFECT(match, re);
+    OBJ_INFECT(match, str);
+     }

      return result;
 }

+/* returns byte offset */
+long
+rb_reg_search(VALUE re, VALUE str, long pos, int reverse)
+{
+    return rb_reg_search0(re, str, pos, reverse, TRUE);
+}
+
+long
+rb_reg_search_without_backref(VALUE re, VALUE str, long pos, int reverse)
+{
+    return rb_reg_search0(re, str, pos, reverse, FALSE);
+}
+
 VALUE
 rb_reg_nth_defined(int nth, VALUE match)
 {
diff --git a/string.c b/string.c
index 8bbd8a4..64d53be 100644
--- a/string.c
+++ b/string.c
@@ -4335,6 +4335,7 @@ rb_str_reverse_bang(VALUE str)
      return str;
 }

+long rb_reg_search_without_backref(VALUE re, VALUE str, long pos, int reverse);

 /*
   *  call-seq:
@@ -4353,8 +4354,13 @@ rb_str_include(VALUE str, VALUE arg)
 {
      long i;

-     StringValue(arg);
-     i = rb_str_index(str, arg, 0);
+     if (RB_TYPE_P(arg, T_REGEXP)) {
+    i = rb_reg_search_without_backref(arg, str, 0, FALSE);
+     }
+     else {
+    StringValue(arg);
+    i = rb_str_index(str, arg, 0);
+     }

      if (i == -1) return Qfalse;
      return Qtrue;
```

#5 - 04/05/2013 12:19 PM - sam.saffron (Sam Saffron)

This is a MASSIVE improvement:

```
#!/usr/bin/env ruby
$: << File.dirname(__FILE__)+'/lib'
require 'benchmark'
require 'fast_blank'

class String
  # active support implementation
  def slow_blank?
    self !~ /[^[:space:]]/
  end
end


n = 1000000


strings = [
  "",
  "\r\n\r\n  ",
  "this is a test",
  "   this is a longer test",
  "   this is a longer test
      this is a longer test
      this is a longer test
      this is a longer test
      this is a longer test"
]

strings.each do |s|
  raise "failed on #{s.inspect}" if s.blank? != s.slow_blank?
end

Benchmark.bmbm  do |x|
  strings.each do |s|
    x.report("Fast Blank #{s.length}    :") do  n.times { s.blank? }  end
    x.report("Fast Blank (Active Support)  #{s.length}    :") do  n.times { s.blank_as? }  end
    x.report("Slow Blank #{s.length}    :") do  n.times { s.slow_blank? }  end
    x.report("include? #{s.length}    :") do  n.times { !s.include?(/[^[:space]]/) }  end
  end
end
```

```
                                        user     system      total        real
Fast Blank 0    :                   0.080000   0.000000   0.080000 (  0.077008)
Fast Blank (Active Support)  0    :  0.080000   0.000000   0.080000 (  0.076362)
Slow Blank 0    :                   0.380000   0.000000   0.380000 (  0.378698)
include? 0    :                     0.180000   0.000000   0.180000 (  0.184465)

Fast Blank 6    :                   0.180000   0.000000   0.180000 (  0.180450)
Fast Blank (Active Support)  6    :  0.210000   0.000000   0.210000 (  0.207886)
Slow Blank 6    :                   0.590000   0.000000   0.590000 (  0.588945)
include? 6    :                     0.190000   0.000000   0.190000 (  0.190898)

Fast Blank 14    :                  0.090000   0.000000   0.090000 (  0.088225)
Fast Blank (Active Support)  14    : 0.130000   0.000000   0.130000 (  0.131408)
Slow Blank 14    :                  0.670000   0.000000   0.670000 (  0.674838)
include? 14    :                    0.190000   0.000000   0.190000 (  0.191627)

Fast Blank 24    :                  0.190000   0.000000   0.190000 (  0.186498)
Fast Blank (Active Support)  24    : 0.140000   0.010000   0.150000 (  0.147858)
Slow Blank 24    :                  0.770000   0.000000   0.770000 (  0.767816)
include? 24    :                    0.220000   0.000000   0.220000 (  0.220636)

Fast Blank 136    :                 0.150000   0.000000   0.150000 (  0.150967)
Fast Blank (Active Support)  136    : 0.150000   0.000000   0.150000 (  0.147665)
Slow Blank 136    :                 0.770000   0.000000   0.770000 (  0.779459)
include? 136    :                   0.200000   0.000000   0.200000 (  0.189744)
```

Some notes:

1. I am noticing ruby head as a 20% or so faster regex going on that 2.0 for these tests
2. the include? method is only 30% or so percent slower than hand coding, though empty strings need special casing. Essentially include? should be short cutting if the string length is zero and returning false if the regex past in is a non-global setting one.
3. I love this improvement to include?, totally support it accepting regexes. Though I very much worry about consistency here.

My suggestion would be:

1. Amend include? to accept a regex
2. Keep in line with the changes in https://bugs.ruby-lang.org/issues/8110 ... so for it to skip globals you MUST pass in /regx/S (a regex that skips setting globals)

I very much worry about having a mishmash in the language where some methods avoid global settings and others do not. The cleanest way of introducing this change is simply to allow for the new rege modifier and keep all places that accept regexes in MRI consistent.

#6 - 04/05/2013 05:53 PM - now (Nikolai Weibull)

On Fri, Apr 5, 2013 at 5:19 AM, sam.saffron (Sam Saffron)
sam.saffron@gmail.com wrote:

> Essentially include? should be short cutting if the string length is zero and returning false.

The empty string is included by quite a few regular expressions, so
that can't be done. Why not simply define String#blank? as

```
class String
  def blank?
    empty? or not include?(/[^[:space:]]/)
  end
end
```

#7 - 04/05/2013 08:14 PM - sam.saffron (Sam Saffron)

Fair enough:

> "" =~ /()|()/
> => 0
> "".include? ""
> => true

I guess optimising for the empty string is tough.

The empty? trick does reduce the cost significantly for the empty string (0.06 vs 0.18) though it increases cost for all the other cases by 20%

We are dealing with a pretty subtle micro optimisation here.

#8 - 04/28/2016 02:04 PM - nobu (Nobuyoshi Nakada)

- Description updated

#9 - 04/28/2016 02:05 PM - nobu (Nobuyoshi Nakada)

- Description updated

#10 - 04/28/2016 02:07 PM - nobu (Nobuyoshi Nakada)

- Has duplicate Feature #12306: Implement String #blank? #present? and improve #strip and family to handle unicode added