Ruby master - Feature #16049

optimization for frozen dynamic string literals "#{exp}".dup and +"#{exp}"

08/06/2019 07:25 PM - Dan0042 (Daniel DeLorme)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

When the decision was made that frozen_string_literal: true should also apply to dynamic string literals, it was mitigated with the following explanation:

> "#{exp}".dup can be optimized so it won't allocate extra objects like "...".freeze

https://docs.google.com/document/u/1/d/1D0Eo5N7NE_unIySOKG9lVj_eyXf66BQPM4PKp7NvMyQ/pub

However that does not appear to be the case currently.

Using this script that generates 100k String objects:

```
# frozen_string_literal: true

def allocated
  GC.stat[:total_allocated_objects]
end

GC.disable
c = ARGV.shift.to_sym
x_eq_i = ARGV.shift=="i"
x = "x"
before = allocated

100_000.times do |i|
  x = i.to_s if x_eq_i
  case c
  when :normal then v = "#{x}"
  when :freeze then v = "#{x}".freeze
  when :minus  then v = -"#{x}"
  when :dup    then v = "#{x}".dup
  when :plus   then v = +"#{x}"
  else raise
  end
end

after = allocated
printf "%d\n", after-before
```

I get the following number of objects allocated

```
x=  frozen_string_literal   normal  freeze  minus   dup     plus
'x' false                   100001  100001  100001  200001  100001
'x' true                    100001  100001  100001  200001  200001
i   false                   200001  200001  299999  300001  200001
i   true                    200001  200001  200001  300001  300001
```

We can see that "#{x}".dup and +"#{x}" allocate an extra object per iteration

I also tested with x = i.to_s to see if deduplication of 100k identical strings was different from 100k different strings. In addition to the expected extra strings created by i.to_s, there's an additional 100k extra strings created for -"#{i}" when frozen_string_literal is false??? There may also be a memory leak here because while the number of objects increases by x3, memory usage increases by x4.

Summary:
I expected "#{v}".dup and +"#{v}" to behave the same regardless of frozen_string_literal (and optimize down to just one allocation)
I expected "#{v}".freeze and -"#{v}" to behave the same regardless of frozen_string_literal (and optimize down to just one allocation)
but they do not. I think they should. It would be nice.

History

#1 - 08/07/2019 10:13 AM - Eregon (Benoit Daloze)

Part of the explanation:
String#freeze never allocates a new String, it just freezes the receiver in place.
String#-@ deduplicates/interns the String, to do so it needs to return a new String instance (unless it's already an interned String, or it cannot be interned because e.g., it has instance variables).

#2 - 08/09/2019 09:54 PM - Dan0042 (Daniel DeLorme)

- Description updated

#3 - 08/14/2019 04:30 PM - jeremyevans0 (Jeremy Evans)

- Backport deleted (2.5: UNKNOWN, 2.6: UNKNOWN)

- ruby -v deleted (ruby 2.7.0dev (2019-04-22 trunk 67701) [x86_64-linux])

- Tracker changed from Bug to Feature