Ruby trunk - Bug #13857

frozen string literal: can freeze same string into two unique frozen strings

09/01/2017 07:16 AM - chucke (Tiago Cardoso)

Status:	Rejected		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	2.3.4, 2.4.1	**Backport:**	2.2: UNKNOWN, 2.3: UNKNOWN, 2.4: UNKNOWN

Description

Running an interpreter with --enable-frozen-string-literal on, I get the following:

```
> "bang".object_id #=> 70303434940940  GOOD!
> "bang".object_id #=> 70303434940940  GOOD!
> "bang".object_id #=> 70303434940940  GOOD!
> c = "bang"
> c.object_id #=> 70303434940940 GOOD!
> c.downcase #=> "bang"
> c.downcase.object_id #=> 70303430619560  SO SO!
> c.downcase.freeze.object_id #=> 70303430601780  BAD!
```

This ticket concerns the last two examples. In the case of performing an operation on the string, it makes sense to return a new string, even if the result is the same. However, I think that the last one could be done differently, in that the frozen result of the downcased value should be the original literal.

I didn't see yet how the frozen string literals are implemented, so this might be dependent on it, but I think that this misses a few optimization use cases. One notable example is keeping a headers hash from an http library. net/http keeps a version of the headers hash with the keys downcased, only to capitalize them on send. Something like this:

```
request["Content-Type"] = "text/html" #=> key stored in request will be "content-type"
```

will create more allocations than expected.

Related issues:

Related to Ruby trunk - Feature #13725: [PATCH] Hash#[]= deduplicates string ...	**Closed**

History

#1 - 09/01/2017 09:20 AM - nobu (Nobuyoshi Nakada)

- Related to Feature #13725: [PATCH] Hash#[]= deduplicates string keys if (and only if) fstring exists added

#2 - 09/01/2017 09:22 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Rejected

chucke (Tiago Cardoso) wrote:

> Running an interpreter with --enable-frozen-string-literal on, I get the following:
>
> ```
> > c.downcase.object_id #=> 70303430619560  SO SO!
> > c.downcase.freeze.object_id #=> 70303430601780  BAD!
> ```

These are not literals, so not subjects of frozen-string-literal.

BTW:

> ```
> request["Content-Type"] = "text/html" #=> key stored in request will be "content-type"
> ```
>
> will create more allocations than expected.

Hash key strings are deduped in the trunk already.

```
$ ruby -v --enable-frozen-string-literal -e 'request = {}; key = "Content-Type".downcase; request[key] = "text
/html"; newkey = request.keys[0]; p key.equal?(newkey), newkey.equal?("content-type")'
ruby 2.5.0dev (2017-08-31 trunk 59695) [universal.x86_64-darwin15]
false
true
```

#3 - 09/01/2017 09:27 AM - chucke (Tiago Cardoso)

> These are not literals, so not subjects of frozen-string-literal.

I'd argue, that's an implementation detail. According to the principle of least surprise, I'd expect them to be the same.

#4 - 09/01/2017 09:28 AM - nobu (Nobuyoshi Nakada)

chucke (Tiago Cardoso) wrote:

> > These are not literals, so not subjects of frozen-string-literal.
>
> I'd argue, that's an implementation detail.

It's not an implementation detail, but a language spec.

#5 - 09/01/2017 09:31 AM - nobu (Nobuyoshi Nakada)

chucke (Tiago Cardoso) wrote:

> According to the principle of least surprise, I'd expect them to be the same.

And proposals based on "the principle of least surprise" will be rejected in common.
It's not our "principle".

#6 - 09/01/2017 10:08 AM - shyouhei (Shyouhei Urabe)

chucke (Tiago Cardoso) wrote:

> > These are not literals, so not subjects of frozen-string-literal.
>
> I'd argue, that's an implementation detail. According to the principle of least surprise, I'd expect them to be the same.

Literals are literals. Those c.downcase-generated strings definitely aren't. I don't see any implementation details here.

#7 - 09/01/2017 10:54 AM - chucke (Tiago Cardoso)

Please don't get me wrong, I'm not arguing that the spec for the feature is vague.

I understood that the introduction of the feature was to reduce memory consumption in template generation (like erb templates), and to avoid those CONTENT_LENGTH = "Content-Length".freeze assignments seen a bit everywhere from ruby web servers to rack. In most of these libs (here's rack's example), there's an header hash abstraction which applies downcase operation to the keys, and then (optionally) freezes.

Point being, at any given time, we might have two strings in memory (ex: "Content-Length"), both frozen, one of them a literal.

#8 - 09/01/2017 11:19 AM - shyouhei (Shyouhei Urabe)

OK, I see what you want. So maybe what is wanted is a String#downcase variant which "inherits" frozenness of the source string. That way we can dedup them when necessary. I think there are rooms for such feature. Not sure how to achieve that yet though.

#9 - 09/01/2017 12:10 PM - chucke (Tiago Cardoso)

Or maybe a String#freeze which returns an already existing frozen literal if such already exists. I think that the #downcase result makes sense, and would break existing code otherwise.

To sum it up, this is what I think could make sense:

```
c = "bang" #=> frozen literal enable, this one is frozen
c.object_id #=> 70303434940940 GOOD!
c.downcase #=> "bang"
c.downcase.object_id #=> 70303430619560  NEW STRING!
```

```
    c.downcase.freeze.object_id #=> 70303434940940 LITERAL GOOD!!!
```

I don't see currently a real world use case where that would break code. And I also can't evaluate the feasibility of the feature, as I don't know the source code that well. But if there's an hash table where literals are stored, there could be a possibility that calling #freeze could do a lookup, replace with frozen literal, or freeze otherwise.

#10 - 09/07/2017 04:52 PM - kernigh (George Koehler)

I expect String#freeze to return the receiver. For example,

```
str = "Content-Length".downcase
str.freeze         # freezes and returns str
str.concat(": ")  # raises RuntimeError: can't modify frozen String
```

I know that str.freeze returns str, so I ignore the return value, and continue using str. Tiago Cardoso proposes that str.freeze would return a deduplicated string, perhaps not str. But my program would ignore the deduplicated string and use str, so the deduplication would be useless.