String#% behavior

Right now String#% is calling #to_ary on the its arguments for every case. Should it call it on cases where it only receives one argument?

Example:

"%c" % 65   # the call's not really necessary but it's done here.

We'd like to have this clarified for Rubyspec and for Rubinius.

Thanks

<table>
<thead>
<tr>
<th>Status:</th>
<th>Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>matz (Yukihiro Matsumoto)</td>
</tr>
<tr>
<td>Target version:</td>
<td>2.0.0</td>
</tr>
</tbody>
</table>

History

#1 - 08/28/2008 06:56 AM - nobu (Nobuyoshi Nakada)

Sorry but I can't get your point.
String#% always can receive only one argument.

#2 - 08/28/2008 07:26 AM - rue (Eero Saynatkari)

The potential issue is that String#% always calls #to_ary when available, even if the format string only has one substitution. In the case of "%c" % obj, #to_ary should not be called but for "%c %s" % obj it should. Would that be sensible behaviour?

#3 - 08/28/2008 07:52 AM - nobu (Nobuyoshi Nakada)

It's an implementation detail.
I don't think it should be a part of the spec.

#4 - 08/28/2008 07:07 PM - larsch (Lars Christensen)

> It's an implementation detail.
> I don't think it should be a part of the spec.

Somewhat contrived, but still a surprise to me:

class A
def to_s; "Hello"; end
def to_ary; ["Goodbye"]; endend

puts sprintf("%s", A.new) # => Hello
puts "%s" % A.new   # => Goodbye

On Ruby 1.8.6-p111, this prints "Hello" two times instead.

Lars

#5 - 08/28/2008 11:55 PM - rue (Eero Saynatkari)
Nakada said: "It's an implementation detail"

I would disagree it is an implementation detail for the reason that Lars posted an example about and the inverse of the example is also true, if someone expects to have #to_ary called but it is not. I think perhaps we see the problem from different aspects. Am I correct in assuming that your point is that String#% always expects an Array argument (whether true Array or #to_ary)?

I suppose a third option would be to specify that the more specific conversion is attempted first (e.g. #to_s(tr) for %s, #to_i(nt) for %c etc.) and if it does not exist, #to_ary is attempted. To me it is more logical to never convert to Array when only one value is asked for to begin with.

My preference is only calling #to_ary when multiple substitutions exist, but it does not really matter which option is chosen. I do think it must be specified to behave one way or the other, even if it is the current implementation.

(In my opinion, any use of #to_ary, #to_int, etc. or even #to_a, #to_i can never be an implementation detail because it affects user code.)

IMHO, String#% should always expect an array as the right part. Having it to expect an object and calling #to_ary on that object is not desirable and ambiguous. It might also lead to some sort (speculating here) of performance penalty since it must check the number of needed arguments before checking if the right part should be array or not. I would expect:

"%d" % [1]
to work, and
"%d" % 1
to fail with ArgumentError or something like that. I think (again, IMHO) that this might lead to a simpler and more efficient implementation.

regards,
--
Rolando Abarca M.

#7 - 08/29/2008 12:58 AM - rue (Eero Saynatkari)

Well, we get into the semantics of #to_ary there. My view is that an object responding to #to_ary means that the object is an Array for all intents and purposes. #to_a, on the other hand, is only an Array representation of the object. I assume that is the reason for the current implementation.

#8 - 02/03/2009 12:16 PM - shyouhei (Shyouhei Urabe)

- Assignee set to matz (Yukihiro Matsumoto)

#9 - 04/04/2010 12:19 AM - znz (Kazuhiro NISHIYAMA)

- Category set to core
- Target version set to 2.0.0

#10 - 09/14/2010 04:58 PM - shyouhei (Shyouhei Urabe)

- Status changed from Open to Assigned

#11 - 02/07/2012 09:28 PM - mame (Yusuke Endoh)

- Status changed from Assigned to Rejected

Hello,

I'm rejecting this feature ticket because no progress has been made for a long time. See [ruby-core:42391].

The following is just my personal comment.

An implementation detail is an implementation detail, even if it affects user code.

```ruby
class A
def to_s; "Hello"; end
def to_ary; ["Goodbye"]; end
end

puts sprintf("%s", A.new) #=> Hello
puts "%s" % A.new    #=> Goodbye
```

This comparison does not make sense because this compares different things.
puts sprintf("%s", A.new) #=> Hello
puts "%s" % [A.new]  #=> Hello

There is nothing strange.
Do you complain the following behavior, too?

    sprintf("%d %d", 1, 2) #=> "1 2"
    "%d %d"%(1, 2)       #=> wrong number of arguments(2 for 1)

--
Yusuke Endoh mame@tsg.ne.jp