Ruby master - Feature #9807

String.new with block

05/06/2014 11:22 AM - citizen428 (Michael Kohl)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

After a discussion in our team chat today, I wondered if it would be a good idea to have a version of String.new that accepts a block and works as a string builder. Something like

```
string = String.new("foo") do |s|
  s << "bar"
end
string #=> "foobar"
```

If the argument is omitted an empty string would be passed to the block instead.

That could be a nice solution to avoid all the Array#join and "".tap hacks for string creation.

History

#1 - 05/06/2014 11:38 AM - phluid61 (Matthew Kerwin)

> That could be a nice solution to avoid all the Array#join and "".tap
> hacks for string creation.

Which hacks are these? Also, I don't see how it's different from
´"foo".tap{|s|s<<"bar"}´

Can you give some examples?

#2 - 05/07/2014 05:54 AM - duerst (Martin Dürst)

Michael Kohl wrote:

> After a discussion in our team chat today, I wondered if it would be a good idea to have a version of String.new that accepts a block and works as a string builder. Something like
>
> ```
> string = String.new("foo") do |s|
>   s << "bar"
> end
> string #=> "foobar"
> ```
>
> If the argument is omitted an empty string would be passed to the block instead.

Like Matthew, I'd also like to see some examples, in particular one that shows how this is different from String.new("foobar") (or even better, from "foobar").

> That could be a nice solution to avoid all the Array#join and "".tap hacks for string creation.

I think "empty string would be passed to the block" may be quite misleading, because that way, people understand the block variable as a string, which would mean that with multiple <<, it's very inefficient.

I think using a different block variable could make things clearer. And showing a simple implementation may make things ever clearer:

```
class String
  def initialize(...)
    # current stuff omitted
    if block_given?
      builder = []
      yield builder
```

```
        replace builder.join
      end
    end
  end
```

That would put the "hack" to collect a large number of Strings in an array to avoid O(n**2) performance penalty of repeated string concatenation "under the hood". The problem I see is that making the builder array available inside a block limits its usability. That's where examples would help.

I was just looking at examples of where I use the above "hack", and one I found, which might be fairly typical, is something like:

```
result = []
foos.each do |foo|
  result << foo.process
end
result.join
```

That would now become something like

```
String.new do |buffer|
  foos.each do |foo|
    buffer << foo.process
  end
end
```

Is that what you have in mind?