confusion between x=x+y, x+=y, x.concat(y) and y.each{|z| x<<z}

while discussing a ticket I have noticed that there is no documentation for +=

I was expecting += to behave as concat but instead it behaves as x=x+y which for every operation clones the array and updates the variable with new value so it behaves similarly to x=x.dup.concat(y) and is slightly faster, but using plane x.concat(y) is a lot faster from both each<< and +=

I would either like to get:

- updated docs that describe concept of += and show the difference from concat
- or change += to use concat which is faster - and add docs ;) (I would expect += to use concat when available)

description

test:

```ruby
require 'benchmark'

rep = 10_000

Benchmark.bmbm do |x|
  { 1..25 => [],
    "a"..'z' => 
    }.each do |base, storage|
    base = base.to_a
    basej = base
    class_name = storage.class.to_s
    x.report(class_name+'#concat') do
      a = storage.dup
      basej = base.join if storage == 
      rep.times { a.concat(basej) }
    end

    x.report(class_name+'#<<') do
      a = storage.dup
      basej = base.join if storage == 
      rep.times { base.each { |e| a << e } }
    end

    x.report(class_name+'#+=') do
      a = storage.dup
      basej = base.join if storage == 
      rep.times { a += basej }
    end

    x.report(class_name+'#dup.concat') do
      a = storage.dup
      basej = base.join if storage == 
      rep.times { a = a.dup.concat(basej) }
    end
  end
end

and here are results on my machine:
```
+= doesn't change the object itself.

For strings for example:

```ruby
x = y = 'foo'
x += 'bar'
x #=> 'foobar'
y #=> 'bar'
```

As well as for integers:

```ruby
x = y = 7
x += 3
x #=> 10
y #=> 7
```

That is x += y is semantically identical to x = x + y.

concat on the other hand does change the object itself (and is specific to arrays, not everything that has + defined):

```ruby
x = y = [1,2,3]
x.concat([4, 5, 6])
x #=> [1,2,3,4,5,6]
y #=> [1,2,3,4,5,6]
```

I disagree with making the proposed change to +=. I would find it extremely surprising for += to modify an existing Array object. I really can't imagine any mental model of Ruby where it would make intuitive sense to do that. It goes against existing Ruby semantics and would have be taught as a special case. It will likely break existing Ruby code. It increases the complexity of Ruby semantics and Ruby implementations. It introduces implicit mutation, which is probably something we want less of, not more.

However I do agree that we need better documentation for things like +=. I'm not sure where I would look for documentation of something like that. We don't really have language documentation, do we?

I would also be in favour of transparently implementing Array#+ as something similar to concat where through escape analysis it can be determined that the original object is never needed again, but that is a lot of ask of Ruby implementations.
I added some explanation to the documentation of Array#+ in r48682. I haven't been able to make RDoc create a separate entry for Array#+=, but I'm not an expert on RDoc.

Of note, Python does implement += this way. += is (almost) an alias for list.extend:

```python
>>> x = []
>>> y = x
>>> x += [42]
>>> y
[42]
```

I think Ruby's behavior is more sensible, but I can imagine this causing confusion for Ruby users with a Python background - I, for one, was mildly surprised when I first found this out.

I imagine that Python also lets one override the += method. I think this is impossible on Ruby because I think it is the parser that interprets a += b to be a = a + b, so only the #+ method will be called (with coercion) regardless of which way it was expressed in the code. This was the conclusion I reached after trying to add a #+= method to a class (in a C extension for MRI) and discovering that it was never called by a += b. FWIW, it was callable via #send.

Maybe this could become a feature request to allow <OP>= to be methods instead of just syntactic sugar?

Of course you can.

I believe it will be rejected soon, however.

- Target version deleted (2.2.0)