Ruby trunk - Feature #12247

accept multiple arguments at Array#delete

04/03/2016 08:51 AM - usa (Usaku NAKAMURA)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

I found that it's very useful if Array#delete accepts multiple arguments.

```
ary = [1, 2, 3, 4, 5]
ary.delete(1, 3)        #=> [1, 3]
ary                     #=> [2, 4, 5]
```

Related issues:

Related to Ruby trunk - Feature #12333: `String#concat`, `Array#concat`, `Str...	**Closed**

History

#1 - 04/03/2016 09:45 AM - usa (Usaku NAKAMURA)

- File deleted (delete_multi.patch)

#2 - 04/03/2016 09:45 AM - usa (Usaku NAKAMURA)

- File delete_multi.patch added

#3 - 04/04/2016 07:03 AM - shyouhei (Shyouhei Urabe)

+1 I needed this before too. For API consistency I would also want Hash#delete to accept multiple argumenrts as well.

#4 - 04/04/2016 09:11 AM - skalee (Sebastian Skalacki)

Given:

```
ary = [1, 2, 3, 4, 5]
```

What is going to be returned by:

```
ary.delete(1, 6)
```

?

#5 - 04/04/2016 01:48 PM - shevegen (Robert A. Heiler)

I guess this is a simple change. I have nothing against it.

As for the last question:

```
ary = [1, 2, 3, 4, 5]
ary.delete(1, 6) # => [1, nil]
ary              # => [2, 3, 4, 5]
```

I guess that would be the most consistent behaviour since this
is how it already behaves as-is.

#6 - 04/04/2016 02:04 PM - usa (Usaku NAKAMURA)

With my implementation (attached patch),

```
ary = [1, 2, 3, 4, 5]
ary.delete(1, 6) # => [1]
ary              # => [2, 3, 4, 5]
```

Because if you want to get [1, nil], you can write:

```
ary = [1, 2, 3, 4, 5]
ary.delete(1, 6){ nil } # => [1, nil]
```

```
ary                     # => [2, 3, 4, 5]
```

That said, I am not convinced that this is the best behavior.

#7 - 04/04/2016 02:45 PM - sawa (Tsuyoshi Sawada)

I like the idea, but I feel a slight inconsistency of the proposed output with the existing behavior. When the argument is single, you get the relevant element. But when the argument is multiple, the return value becomes an array.

```
[1, 2, 3].delete(1) # => 1
[1, 2, 3].delete(1, 2) # => [1, 2]
[1, 2, 3].delete(1, 2, 3) # => [1, 2, 3]
```

When you directly give the arguments, this may not be a big problem, but when you have an array array of arbitrary (> 0) length, and do this:

```
[1, 2, 3].delete(*array)
```

depending on the length of array, you get different types of results. It can be a gotcha.

Note that the return value is not much informative in the first place. For example, consider this case (assuming non-frozen string):

```
a1 = "a"
a2 = "a"
a1.object_id # => 70317858422560
a2.object_id # => 70317858414960
[a1, a2].delete("a").object_id # => 70317858414960
```

When there are different objects that are equal under ==, only one of the deleted elements (the last one?) is returned. Considering that, it might be more consistent to just return one of the deleted elements with multiple arguments too:

```
a = [1, 2, 3]
a.delete(1, 2) # => 2
a # => [3]
a = [1, 2, 3]
a.delete(1, 2, 3) # => 3
a # => []
```

And when you need all the deleted elements, you can have a method with a different name. Such method should always return an array, and even when the argument is single, it may return more than one element:

```
["a", "a"].another_delete_method("a") # => ["a", "a"]
[1, 2, 3].another_delete_method(1) # => [1]
[1, 2, 3].another_delete_method(1, 2) # => [1, 2]
[1, 2, 3].another_delete_method(1, 2, 3) # => [1, 2, 3]
```

There may be some better way, but I am not sure.

#8 - 04/04/2016 02:48 PM - skalee (Sebastian Skalacki)

If the return value has as many items as arguments passed to the method and the items order is preserved, then the return value decomposition will be possible, for example:

```
ary = %w[c a e c]
a, b, c = ary.delete("a", "b", "c") # => ["a", nil, "c"]
ary # => ["e"]
a # => "a"
b # => nil
```

That said, I'm not sure if such construct is useful at all.

Anyway, I think that the return value could be better described in docs. I mean both the elements contained and their order. Right now it only says that the return value is an empty array when no items are removed.

#9 - 04/04/2016 03:00 PM - usa (Usaku NAKAMURA)

Tsuyoshi Sawada wrote:

> depending on the length of array, you get different types of results. It can be a gotcha.

At first, I also think so, but nobu told me a precedent, p.
In fact, compatibility and convenience would be more important than consistency.

#10 - 04/04/2016 03:02 PM - usa (Usaku NAKAMURA)

Sebastian Skalacki wrote:

> Anyway, I think that the return value could be better described in docs.

Agreed.
Please discuss better behavior to decide what we should describe in docs :-P

#11 - 04/04/2016 06:15 PM - naruse (Yui NARUSE)

With gcc version 5.3.0 20151204 (Ubuntu 5.3.0-3ubuntu1~14.04), following result:

```
%  time ./miniruby54488 -e'i=1000000;while (i-=1)>0;[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);[1,2,3,4,5].de
lete(1);[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);end'
./miniruby54488   2.07s user 0.01s system 99% cpu 2.081 total
%  time ./miniruby54488 -e'i=1000000;while (i-=1)>0;[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);[1,2,3,4,5].de
lete(1);[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);end'
./miniruby54488   2.08s user 0.00s system 99% cpu 2.082 total
%  time ./miniruby54488 -e'i=1000000;while (i-=1)>0;[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);[1,2,3,4,5].de
lete(1);[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);end'
./miniruby54488   1.92s user 0.00s system 99% cpu 1.925 total
%  time ./miniruby54488 -e'i=1000000;while (i-=1)>0;[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);[1,2,3,4,5].de
lete(1);[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);end'
./miniruby54488   1.95s user 0.00s system 99% cpu 1.957 total
%  time ./miniruby54488 -e'i=1000000;while (i-=1)>0;[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);[1,2,3,4,5].de
lete(1);[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);end'
./miniruby54488   1.98s user 0.01s system 99% cpu 1.988 total

%  time ./miniruby -e'i=1000000;while (i-=1)>0;[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(
1);[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);end'
./miniruby   1.98s user 0.00s system 99% cpu 1.985 total
%  time ./miniruby -e'i=1000000;while (i-=1)>0;[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(
1);[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);end'
./miniruby   1.96s user 0.01s system 99% cpu 1.977 total
%  time ./miniruby -e'i=1000000;while (i-=1)>0;[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(
1);[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);end'
./miniruby   2.11s user 0.00s system 99% cpu 2.121 total
%  time ./miniruby -e'i=1000000;while (i-=1)>0;[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(
1);[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);end'
./miniruby   2.01s user 0.01s system 99% cpu 2.021 total
%  time ./miniruby -e'i=1000000;while (i-=1)>0;[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(
1);[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);end'
./miniruby   2.04s user 0.01s system 99% cpu 2.056 total
%  time ./miniruby -e'i=1000000;while (i-=1)>0;[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(
1);[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);end'
./miniruby   1.99s user 0.00s system 99% cpu 1.997 total
%  time ./miniruby -e'i=1000000;while (i-=1)>0;[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(
1);[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);end'
./miniruby   2.10s user 0.01s system 99% cpu 2.119 total
%  time ./miniruby -e'i=1000000;while (i-=1)>0;[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(
1);[1,2,3,4,5].delete(1);[1,2,3,4,5].delete(1);end'
./miniruby   2.00s user 0.00s system 99% cpu 1.997 total
```

#12 - 04/04/2016 10:16 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

+1 I needed this a few days ago

#13 - 05/28/2016 04:40 AM - hsbt (Hiroshi SHIBATA)

- Related to Feature #12333: `String#concat`, `Array#concat`, `String#prepend` to take multiple arguments added

Files

delete_multi.patch	4.6 KB	04/03/2016	usa (Usaku NAKAMURA)