Ruby master - Feature #7546

Change behavior of `Array#slice` for an argument of `Range` class

12/12/2012 01:23 AM - alexeymuranov (Alexey Muranov)

Status:	Open	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:	3.0	

Description

=begin
This is a concrete proposal to "fix" #4541.

It is also related to #7545.
For this proposal to make good sense, i think it would be nice if #7545 was at least partially accepted.

=== Main proposal

I propose (({Array#slice})) with (({Range})) type argument to work as follows:

```
a = ['0', '1', '2', '3']
a[1..2]   # => ['1', '2']
a[-2..-1] # => ['2', '3']
a[2..1]   # => ['2', '1']
a[-1..-2] # => ['3', '2']
a[-1..1]  # => ['3', '0', '1']
a[1..-1]  # => ['1', '0', '3']
a[1..1]   # => ['1']
a[1...1]  # => []
a[4..4]   # => [nil]
a[4...4]  # => []
a[9..9]   # => [nil]
a[9...9]  # => []
a[1..5]   # => ['1', '2', '3', nil, nil]
```

=== Secondary proposal: consider adding new instance methods to (({Array})) to compensate the changed behavior of (({Array#slice}))

If this proposal is accepted, the code "(({a[1..-2]}))" for an array (({a})) will not work as before.
This can be compensated by adding new instance methods to (({Array})).
For example the following ones.

1. (({Array#clip(fixnum, fixnum)})):

```
['0', '1', '2', '3'].clip(1, 1) # => ['1', '2']
```

Thus (({a.clip(1, 1)})) would be a replacement for (({a[1..-2]})).

(It looks strange to have to convert a pair of numbers ((*m*)) and ((*n*)) into a range (({m..(-1-n)})) to simply ask an array to remove ((*m*)) elements from the beginning and ((*n*)) elements from the end.
If #7545 is accepted, then the "(({a[1..-2]}))" syntax for "clipping" an array will make not much sense and maybe will not be possible.)

1. (({Array#from(fixnum)})), (({Array#till(fixnum)})):

```
a = ['0', '1', '2', '3']
a.from(1)         # => ['1', '2', '3']
a.till(1)         # => ['0', '1']
a.from(1).till(-2) # => ['1', '2']
```

In fact, in ((*Rails*)) (({ActiveSupport})) there are methods (({Array#from})) and (({Array#to})) like this, but unfortunately they do not accept negative indices.

((*Remark*)). It would also be possible to have (({Array#clip!})), (({Array#from!})), (({Array#till!})).
=end

History

#1 - 12/12/2012 01:51 AM - drbrain (Eric Hodel)

- Target version set to 3.0

This will break existing code so I set it to next major.

#2 - 12/12/2012 04:28 AM - marcandre (Marc-Andre Lafortune)

- Assignee set to matz (Yukihiro Matsumoto)

-5 from me:

- this doesn't solve any real-life problem I can think of
- it will introduce incompatibilities
- those incompatibilities would be very difficult to find by code review/grep/whatever
- this would make array[42..n] not always the same as array[42...n+1]
- what about String#slice?

The goal is not to invent a new language.

#3 - 12/12/2012 06:04 AM - alexeymuranov (Alexey Muranov)

marcandre (Marc-Andre Lafortune) wrote:

> - this doesn't solve any real-life problem I can think of

For me it solves one: the current behavior does not make sense to me, or i do not understand which abstract object is modeled by Range :).

> - this would make array[42..n] not always the same as array[42...n+1]

For integer n ≥ 42 it should be the same, otherwise i propose to consider #7545.

> - what about String#slice?

First the same, then discard the nil values to get a possibly empty string.

By the way, why would you slice a 5-element array or a 5-letter string by something like 2..42 ? (And how about Array#from ?)

Edited

#4 - 12/12/2012 06:27 AM - phluid61 (Matthew Kerwin)

alexeymuranov (Alexey Muranov) wrote:

> marcandre (Marc-Andre Lafortune) wrote:
>
> > - this doesn't solve any real-life problem I can think of
>
> For me it solves one: the current behavior does not make sense to me, or i do not understand which abstract object is modeled by Range :).

At the risk of sounding glib, there is an alternative solution: learn it.

#5 - 12/13/2012 06:36 AM - alexeymuranov (Alexey Muranov)

In fact, i do not request particularly this part:

```
a = ['0', '1', '2', '3']
a[4..4]   # => [nil]
a[9..9]   # => [nil]
a[1..5]   # => ['1', '2', '3', nil, nil]
```

The following alternative, closer to the current behavior, would be fine with me:

```
a = ['0', '1', '2', '3']
a[4..4]   # => []
a[4...4]  # => []
a[9..9]   # => []
a[9...9]  # => []
a[1..5]   # => ['1', '2', '3']
```