Ruby master - Feature #16822

Array slicing: nils and edge cases

04/30/2020 10:44 AM - zverok (Victor Shepelev)

Status:	Rejected
Priority:	Normal
Assignee:	
Target version:	

Description

(First of all, I understand that the proposed change can break code, but I expect it not to be a large amount empirically.)

I propose that methods that slice an array (#slice and #[]) and return a sub-array in the normal case, should **never** return nil. E.g.,

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

- 1. Non-empty slice--how it works currently

```
a[1..2] # => [2, 3]
a[1...-1] # => [2]
```

- 2. Empty slice--how it works currently

```
a[1...1] # => []
a[3...] # => []
a[-1..-2] # => []
```

- 3. Sudden nil—**what I am proposing to change**

```
a[4..] # => nil
a[-10..-9] # => nil
```

I believe that it would be better because the method would have cleaner "type definition" (If there is nothing in the array at the requested address, you'll have an empty array).

Most of the time, the empty array doesn't require any special handling; thus, ary[start...end].map { ... } will behave as expected if the requested range is outside of the array boundary.

It is especially painful with off-by-one errors; for an array of three elements, if ary[3...] (just outside the boundary) is [] while a[4...] (one more step outside) is nil, it typically results in some nasty NoMethodError for NilClass.

A similar example is ary[1..].reduce { } (everything except the first element--probably the first element was used to construct the initial value for reducing) with ary being non-empty 99.9% of the times. Then you meet one of the 0.1% cases, and instead of no-op reducing nothing, NoMethodError is fired.

History

#1 - 04/30/2020 12:07 PM - sawa (Tsuyoshi Sawada)

- Description updated

#2 - 04/30/2020 07:42 PM - shevegen (Robert A. Heiler)

I do not have a strong preference here either way; I guess one can reason in favour for both behaviour types/styles, and I think a primary point in the suggestion is that it refers to startless/endless situations, such as "5..", which I don't use myself, but one slight concern is this one:

```
a[-1..-2] # => []
a[-10..-9] # => nil
```

Is this certain to not break a lot of code? I have not checked myself and I rarely use #slice anyway, but I do use a lot of [] in general. It's one of my favourite method calls in general, in ruby. :)

Admittedly I actually don't remember off-hand having ever used two negative

indices here ... for some reason, I seem to use 0 or positive numbers a lot more.

No idea how/if other ruby users use or rely on that behaviour though but I think it would be important to get some specific overview about any potential effect (or side-effect) of proposed changes, even if the reasoning given is ok.

#3 - 05/04/2020 03:01 AM - Dan0042 (Daniel DeLorme)

Slicing returns nil when the index is out of bounds, and that can be a useful signal that something is wrong and we should fail fast. Having that nil return value provides information that is not present if it's auto-converted to an empty array, and it's easy to disregard that information by using .to_a

arr[1..]
Takes all items after the first one, but if there's no first item it can be argued this is an invalid input and returning nil is safer (fail fast) than pretending everything is as expected.

arr[-5..-1]
Takes the last 5 items but if the array has less than 5 items it's an invalid input and we return nil (unlike arr.last(5)). If this proposal is accepted I'm not sure that returning an empty array makes sense here.

Now, all that being said... personally I don't remember ever having depended on array slicing returning a nil for out-of-bounds checking, but I do remember adding a bunch of .to_a or &.each to my code to account for this case. So I am tentatively positive about the idea. But caution is required.

#4 - 05/04/2020 06:34 AM - zverok (Victor Shepelev)

- Description updated

#5 - 05/05/2020 08:23 PM - marcandre (Marc-Andre Lafortune)

I'm strongly against this, for compatibility reasons and because current choice is a consistent convention.

Before proposing any incompatible change, especially for an API that is very much in use, please provide a compelling use case. If you write ary[1..].reduce { }, you must give a context (what contains ary, why would you want to skip the first value, why not use values_at(1..), etc.).

#6 - 05/14/2020 04:52 AM - matz (Yukihiro Matsumoto)

I don't think the benefit of changing outweighs the pain of incompatibility. Rejected.

Matz.

#7 - 05/14/2020 09:32 AM - mrkn (Kenta Murata)

- Status changed from Open to Rejected