Ruby master - Feature #15804

A generic method to resolve the indexing on a sequence

04/27/2019 08:50 PM - Eregon (Benoit Daloze)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

Currently, the implementation of begin/end-less Ranges leaks the representation of infinite ranges (it's a nil value for begin or end).

For instance, this "works" but I think it's potentially confusing:

```
s = "abcdef"
to = some_computation_that_unexpectedly_returns_nil
s[1..to] # => "bcdef"
```

Particularly problematic, this means Array#[], String#[], MatchData#[], ..., all have to handle Ranges and nil explicitly as a special value, in addition to already treat e.g., negative integers specially.
That's not very nice, why should Array#[] know how an infinite Range is represented? IMHO it should not need to (separation of concerns).

Here are the current ways to index a sequence:

```
  seq[index]          index can be negative, but nil is returned if too big or small
  seq[start, length]  start can be negative, but nil is returned if too big or small, length positive or nil is returned
  seq[range]          range begin and end can both be negative, but nil is returned if too big or small
```

User-defined classes have no (efficient) way to reuse that logic to resolve indices, they have to rewrite it in Ruby and given it's not trivial and tends to change (e.g., endless ranges) it might be buggy or incomplete.
For instance, maybe one day we might want to support indexing with a Enumerator::ArithmeticSequence or even more elaborate ways.

I would like to have a standard way to resolve indexing on a 0-indexed sequence with a given size to a span of selected indices.

This is still an early idea, feedback welcome.
For instance, we could have:

```
resolve(size, *args)
# Either:
# => idx, with 0<=idx<size, which means selecting a single element at index `idx`
# => [from, len] with 0<=from<=from+len<=size, which means selecting elements from `from` included to `from+len` excluded.
# => nil if indices are out of bounds
resolve(5, 3)    # => 3
resolve(5, 33)   # => nil
resolve(5, 1..)  # => [1, 4]
resolve(5, 1...) # => [1, 4]
resolve(5, -3..) # => [2, 3]
resolve(5, ..-2) # => [0, 4]
resolve(5, -4..-2) # => [1, 3]
```

I'm not sure what a good name for this method would be, or what a good namespace would be.
Maybe an Enumerable.resolve(size, *args), (0...size).resolve(*args) or Integer.indexing(size, *args)?

Such functionality can currently be worked around by reusing Array#[] with size.times.to_a(*args), but that's of course quite inefficient.

A smaller-scoped version of this would be to just have a way to resolve a Range object with a size like:

```
(-2...).resolve(5) # => [3, 2]
```

That part would at least let Range take care of resolving indices and begin/end-less ranges, instead of duplicating that logic in many places.

History

#1 - 04/27/2019 08:52 PM - Eregon (Benoit Daloze)

- Description updated

#2 - 04/28/2019 12:54 PM - mame (Yusuke Endoh)

I'm unsure how many cases people want to implement the index resolving code, but if it is not rare, I'm for the proposal.

I don't think that we will often add a new concept (like begin/end-less Ranges) that affects the resolving code, but even if we ignore a new feature, the resolver is bothersome and difficult for each user to implement.