FEB 17TH, 2014

The Elements of Style in Ruby #13: Length vs Size vs Count

One of the problems newcomers to Ruby experience is that there are often quite a few ways to do same thing. For instance – you can obtain the number of items in `Enumerable` objects (instances of classes using the `Enumerable` mixin, which would often be collections like `Array`, `Hash`, `Set`, etc) by either using `Enumerable#count` or the methods `length` and its alias `size` that such classes often provide.

```ruby
arr = [1, 2, 3]

arr.length # => 3
arr.size # => 3
arr.count # => 3

h = { a: 1, b: 2 }

h.length # => 2
h.size # => 2
h.count # => 2

str = 'name'
str.length # => 4
str.size # => 4
# str.count won't work as String does not include Enumerable
```

Which one should you use? Let me help with this choice.

`length` is a method that's not part of `Enumerable` – it's part of a concrete class (like `String` or `Array`) and it's usually running in `O(1)` (constant) time. That's as fast as it gets, which means that using it is probably a good idea.

Whether you should use `length` or `size` is mostly a matter of personal preference. Personally I use `size` for collections (hashes, arrays, etc) and `length` for strings, since for me objects like hashes and stacks don't have a length, but a size (defined in terms of the elements they contain). Conversely, it's perfectly normal to assume that some text has some length. Anyways, in the end you're invoking the same method, so the semantic distinction is not important.

`Enumerable#count`, on the other hand, is a totally different beast. It's usually meant to be used with a block or an argument and will return the number of matches in an `Enumerable`:

```
arr = [1, 1, 2, 3, 5, 6, 8]

arr.count(&:even?) # => 3
arr.count(1) # => 2
```

You can, however, invoke it without any arguments and it will return the size of the enumerable on which it was invoked:

```
arr.count # => 7
```

There's a performance implication with this, though – to calculate the size of the enumerable the `count` method will traverse it, which is not particularly fast (especially for huge collections). Some classes (like `Array`) implement an optimized version of `count` in terms of `length`, but many don't.

The takeaway for you is that you should avoid using the `count` method if you can use `length` or `size`.

A note to Rails developers – `ActiveRecord::Relation`'s `length`, `size` and `count` methods have a totally different meaning, but that's irrelevant to our current discussion. (`Sean Griffin` has written a comment regarding it).