Ruby master - Feature #15837

Module#name_components

05/08/2019 03:49 AM - mrkn (Kenta Murata)

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

Description

I sometimes wrote the expression like below to extract the components of class path:

```
klass.name.split('::').last
```

Similar expressions can be found in ruby-trunk:

```
mrkn@mrkn-devel:~/src/github.com/ruby/ruby$ git grep split..::
ext/json/lib/json/common.rb:      path.to_s.split(/::/).inject(Object) do |p, c|
ext/openssl/lib/openssl/config.rb:              refsec, ref = ref.split('::', 2)
ext/psych/lib/psych/visitors/yaml_tree.rb:          method = "visit_#{(klass.name || '').split('::
').join('_')}"
lib/bundler/cli/gem.rb:      constant_array = constant_name.split("::")
lib/bundler/vendor/molinillo/lib/molinillo/errors.rb:      solver_name = opts.delete(:solver_name)
 { self.class.name.split('::').first }
lib/bundler/vendor/thor/lib/thor/parser/argument.rb:      class_name = self.class.name.split("::")
.last
lib/bundler/vendor/thor/lib/thor/parser/arguments.rb:      class_name = self.class.name.split("::"
).last.downcase
lib/optparse/version.rb:          pkg = pkg.split(/::|\//).inject(::Object) {|m, c| m.const_get(c)
}
lib/optparse/version.rb:    path.split(/::|\//).inject(base) do |klass, name|
lib/rdoc/any_method.rb:                         name = @full_name.split('::')
lib/rdoc/context.rb:          names = ename.split('::')
lib/rdoc/context.rb:    path = [prefix] + path.split('::')
lib/rdoc/method_attr.rb:        $1.split('::').last.  # ClassName => class_name
lib/rdoc/parser/c.rb:                           ([\w\. \t]+ = \s+)?rb_define_(class|module)_under[\t
\w, ()*?"(#{class_name.split('::').last})"%xm then
lib/rdoc/parser/ruby.rb:      obj = name_t[:text].split("::").inject(Object) do |state, item|
lib/rdoc/store.rb:    name = klass_name.split('::').last
lib/rdoc/store.rb:    File.join @path, *klass_name.split('::')
lib/rdoc/store.rb:    method_name = method_name.split('::').last
lib/rss/atom.rb:        "#{self.class.name.split(/::/).last.downcase}="
lib/rss/atom.rb:        target.__send__(self.class.name.split(/::/).last.downcase) {|x| x}
lib/rss/atom.rb:        target.__send__("new_#{self.class.name.split(/::/).last.downcase}")
lib/rss/rss.rb:        tag_name = klass.name.split(/::/).last
spec/bundler/support/artifice/fail.rb:    const = name.split("::").reduce(Object) {|mod, sym| mod.
const_get(sym) }
spec/mspec/lib/mspec/utils/name_map.rb:    c.split('::').inject(base) do |dir, name|
spec/mspec/lib/mspec/utils/name_map.rb:        name = mapping[c.split('::').last] || mapping.fetch
(:default)
```

I think we need Module#name_components method that returns the array of symbols (or string, I prefer symbols) which comes from splitting name by ::.

History

#1 - 05/08/2019 10:29 AM - shevegen (Robert A. Heiler)

I was about to write a long reply, but I think it would become too long, so just a shorter comment.

I use something like this in a "base" gem that I tend to use. Example:

```
class Foo
  class Bar
    NAMESPACE = inspect
```

And then I may split on the NAMESPACE constant, via .split('::'), in order to lateron display to the ruby user who is running the code either: the full name, or only a partial name (such as Bar:), or no name at all, on the commandline, including different colours (for example, if something went wrong, I tend to use red colour tone; and the leading "namespace" I tend to use with a light greyish colour if the terminal background is dark).

Example (even though I can not use colours here on the bug tracker):

```
Foo:Bar: The network is currently unavailable; the dataset could not be obtained.
```

Or something like that.

This is especially useful to me in order to know when something has went wrong and I need to debug the situation at hand. I even have another project that e. g. accepts input such as "Foo::Bar" and then opens these .rb files in my editor, at the least for somewhat smaller classes. I sort of use ruby as an "IDE" here, strange as that may sound, but it is quite effective, actually.

I am not sure if I was able to explain that particular use case well; I try to not write too much. The use case is mostly to just display more information on the commandline, e. g. which .rb file is responsible for some particular output, and sometimes I may only show the name of the class rather than the full name, hence why I need .split('::') - but the more important aspect of this comment is that I can agree with mrkn that there may be common (or somewhat similar) use cases. Ruby users may have to work with .split('::') specifically, also as the example above showed.

I have no particular opinion on the name of the method itself; or what it should return (this is secondary to me) and I also can not answer the question as to whether this functionality may be sufficiently useful to other ruby users or not, and may thus encourage the addition of a new method (I have no strong opinion either way, neither about the name. I am sure if it would be approved then the name will be good, anyway), but I did want to describe one more use case, even if it is quite special and not that applicable to others.

To conclude - I agree with mrkn here; in my opinion it is somewhat common to see. There may also be more use cases; two I can possibly think of may be in rails/active* and another one possibly in introspection of ruby code, such as in pry. I actually looked at pry code and grep would display this:

```
wrapped_module/candidate.rb:         [/^\s*#{mod_type_string}\s+(?:(?:\w*)::)*?#{wrapped.name.split(/::/).last}
/,
wrapped_module/candidate.rb:          /^\s*(::)?#{wrapped.name.split(/::/).last}\s*?=\s*?#{wrapped.class}/,
wrapped_module/candidate.rb:          /^\s*(::)?#{wrapped.name.split(/::/).last}\.(class|instance)_eval/]
```

I think in particular the last two lines may help reinforce the sample listing by mrkn. It may not be a huge use case per se, but it may also not be totally uncommon in different projects, in my opinion. Thanks!