Ruby trunk - Feature #6842

Add Optional Arguments to String#strip

08/07/2012 08:24 PM - wardrop (Tom Wardrop)

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

Description

=begin
One of the very few things I miss from the ol' php days many years ago, was the ability to easily strip arbitrary characters from the ends of string using trim(). Typically, this is whitespace characters, and #strip currently fulfils that use case, but there are also instances where it'd be nice to be able to strip any range of characters from the ends of a string. It goes well with Array#join as often when joining strings with a delimiter, you want to make sure those strings don't already begin or end with that character.

For a full-featured #strip, I'd like to see it have the option of accepting both an Array or String. If a string is provided, each character in that string will be stripped. If an array of strings is given, each element of the array is stripped from the ends of the string - this allows for multi-character delimiters for example. Of course you could go really nuts and supports regex as well (or instead of arrays). To demonstrate the difference...

```
"<b>bold text</b>".strip("</b>") #=> "old text"
"<b>bold text</b>".strip(["<b>", "</b>"]) #=> "bold text"
"<em><b>bold text</b></em>".strip(["<b>", "</b>", "<em>", "</em>"]) #=> "bold text"
"<em><b>bold text</b></em>".strip(/<\/?.+?>/) #=> "bold text"
```

A simple real-world example; this is actually what I was wanting to do right before I came here to raise this feature request, but there's been all kinds of other use cases I've hit in the past:

```
['some', '/chunked', 'path/'].map{ |v| v.strip('/') }.join('/') #=> "some/chunked/path"
```

File#join does something similar, but when you need control over the joining character, this is the way you'd do it.

I've lost count of how many times I've wanted this in Ruby, and there's really no nice workaround. Here's an example on StackOverflow of someone asking how to achieve this stripping behaviour in ruby:
http://stackoverflow.com/questions/3453262/how-to-strip-leading-and-trailing-quote-from-string-in-ruby

Obviously, you'd do the same for #lstrip and #rstrip, and all the mutable variants (#strip!, #lstrip!, #rstrip!). Looking forward to others thoughts on this one.
=end

Related issues:

Has duplicate Ruby trunk - Feature #12694: Want a String method to remove hea...	**Closed**

History

#1 - 08/07/2012 11:10 PM - trans (Thomas Sawyer)

The first example doesn't seem to make sense, e.g.

```
"<b>bold text</b>".strip("</b>") #=> "old text"
```

I also think the interface needs to be a bit more specific about right and left stripping. Maybe use options:

```
strip([String,Array])  # both left and right
strip(:left=>[String,Array], :right=>[String,Array])  # selective right vs left
```

#2 - 08/08/2012 10:51 AM - wardrop (Tom Wardrop)

"b" from the start of the string, and the "" from the end. This is how I remember trim() from php behaved, which I found quite succinct, but maybe that's because defining arrays in PHP is fairly verbose. Ruby has the shorthand array literal syntax %w{< / b >}, so I don't mind if #strip was made to treat a single argument and an array of arguments the same, rather than treating a single string as an array of characters. Actually I think I'd prefer that API - certainly less confusing.

So in that case, the output of the original example would become:

```
"<b>bold text</b>".strip("</b>") #=> "<b>bold text"
```

And you'd do the following to achieve the same result as the original example:

```
"<b>bold text</b>".strip(%w{< / b >}) #=> "old text"
```

As for differentiating left and right, isn't that what #lstrip and #rstrip are for? You could easily chain them to get the desired result. E.g.

```
"<b>bold text</b>".lstrip("<b>").rstrip("</b>") #=> "old text"
```

#3 - 10/12/2012 09:18 PM - mame (Yusuke Endoh)

- Status changed from Open to Assigned

- Assignee set to matz (Yukihiro Matsumoto)

#4 - 11/24/2012 08:45 AM - mame (Yusuke Endoh)

- Target version changed from 2.0.0 to 2.6

#5 - 11/24/2012 12:31 PM - trans (Thomas Sawyer)

Might support regexp instead of arrays,

"bold text".strip(/[\/]/) #=> "old text"

Also note, we partially have this for #rstrip already in the form of #chomp.

#6 - 11/24/2012 01:08 PM - charliesome (Charlie Somerville)

I don't agree with the options hash that was proposed for allowing separate strings to be stripped from the left and the right.

This:

str.strip(:left => "foo", :right => "bar")

is not as clean as something like this IMO:

str.lstrip("foo").rstrip("bar")

#7 - 11/17/2013 01:49 PM - wardrop (Tom Wardrop)

I still look forward to this feature. So much more convenient and readable than the current work-around of using regex and sub/gsub. Is there anything more that needs to be done to make this happened for Ruby 2.1 or 2.2?

#8 - 11/17/2013 01:59 PM - fuadksd (Fuad Saud)

I need this a couple of weeks ago. Looks like a preety commo use case.

#9 - 08/24/2016 12:24 AM - shyouhei (Shyouhei Urabe)

- Has duplicate Feature #12694: Want a String method to remove heading substr added

#10 - 08/24/2016 05:50 AM - sonots (Naotoshi Seo)

I am working on implementing this at https://github.com/ruby/ruby/compare/trunk...sonots:lstrip_arg?expand=1

```
"hello".lstrip!("hell") #=> "o"
"hello".lstrip!("ello") #=> nil
"hello".lstrip!(/hell/) #=> "o"
"hello".lstrip!(/ello/) #=> nil
"hello".lstrip("hell") #=> "o"
"hello".lstrip("ello") #=> "hello"
"hello".lstrip(/hell/) #=> "o"
"hello".lstrip(/ello/) #=> "hello"

"hello".rstrip!("ello") #=> "o"
"hello".rstrip!("hell") #=> nil
"hello".rstrip!(/ello/) #=> "o"
"hello".rstrip!(/hell/) #=> nil
"hello".rstrip("ello") #=> "h"
"hello".rstrip("hell") #=> "hello"
"hello".rstrip(/ello/) #=> "h"
"hello".rstrip(/hell/) #=> "hello"
```

About regular expression argument

I am wondering about specification for regular expression argument.

1. Adding \A (lstrip) or \z (rstrip) for the argument regular expression automatically for matching
2. Let users to choose to add \A or \z (Current implementation is this way)
3. Drop regular expression agument support

For example, "foofoo".rstrip(/foo/) #=> "foo" with 1. With 2, "foofoo".rstrip(/foo/) #=> "foofoo" (mismatch because /foo/ matches with **heading** foo), but `"foofoo".rstrip(/foo\z/) #=> "foo".

The advantage of 1. is more natural interface for users I think. However, it will be slower because we have to add \z for the given regular expression and re-compile internally.

\3. (dropping regexp support) would be fine because we have String#sub(regexp, ''). String#chomp does not support regular expression argument anyway.

About strip

Also, I am not implementing for strip yet. But, my personal thinking is as:

The below interface is not necessary

```
str.strip!(:left => "foo", :right => "bar")
```

because below is sufficient and shorter.

```
str.lstrip!('foo')&.rstrip!('bar')
```

Below interface would be nice to have:

```
str.strip!('foo')
```

Any opinions?