FLUENT STRINGS

Fluent strings provide a more fluent, object-oriented interface for working with string values, allowing you to chain multiple string operations together using a more readable syntax compared to traditional string operations.

Lưu ý:

```
use Illuminate\Support\Str;
```

after

The `after` method returns everything after the given value in a string. The entire string will be returned if the value does not exist within the string:

```
$slice = Str::of('This is my name')->after('This is'); // ' my name'
```

safterLast

The `afterLast` method returns everything after the last occurrence of the given value in a string. The entire string will be returned if the value does not exist within the string:

```
$slice = Str::of('App\Http\Controllers\Controller')->afterLast('\\');

// 'Controller'
```

append

The `append` method appends the given values to the string:

```
$string = Str::of('Taylor')->append(' Otwell'); // 'Taylor Otwell'
```

ascii

The `ascii` method will attempt to transliterate the string into an ASCII value:

```
$string = Str::of('ü')->ascii(); // 'u'
```

basename

The `basename` method will return the trailing name component of the given string:

```
$string = Str::of('/foo/bar/baz')->basename(); // 'baz'
```

If needed, you may provide an "extension" that will be removed from the trailing component:

```
$string = Str::of('/foo/bar/baz.jpg')->basename('.jpg'); // 'baz'
```

before

The `before` method returns everything before the given value in a string:

```
$slice = Str::of('This is my name')->before('my name'); // 'This is '
```

beforeLast

The `beforeLast` method returns everything before the last occurrence of the given value in a string:

```
$slice = Str::of('This is my name')->beforeLast('is'); // 'This '
```

camel

The `camel` method converts the given string to `camelCase`:

```
$converted = Str::of('foo_bar')->camel(); // fooBar
```

contains

The `contains` method determines if the given string contains the given value. This method is case sensitive:

```
$contains = Str::of('This is my name')->contains('my'); // true
```

You may also pass an array of values to determine if the given string contains any of the values in the array:

```
$contains = Str::of('This is my name')->contains(['my', 'foo']); // true
```

containsAll

The `containsAll` method determines if the given string contains all of the values in the given array:

```
$containsAll = Str::of('This is my name')->containsAll(['my', 'name']); // true
```

dirname

The `dirname` method returns the parent directory portion of the given string:

```
$string = Str::of('/foo/bar/baz')->dirname(); // '/foo/bar'
```

If necessary, you may specify how many directory levels you wish to trim from the string:

```
$string = Str::of('/foo/bar/baz')->dirname(2); // '/foo'
```

endsWith

The `endsWith` method determines if the given string ends with the given value:

```
$result = Str::of('This is my name')->endsWith('name'); // true
```

You may also pass an array of values to determine if the given string ends with any of the values in the array:

```
$result = Str::of('This is my name')->endsWith(['name', 'foo']); // true

$result = Str::of('This is my name')->endsWith(['this', 'foo']); // false
```

exactly

The `exactly` method determines if the given string is an exact match with another string:

```php
$result = Str::of('Laravel')->exactly('Laravel'); // true
```

explode

The `explode` method splits the string by the given delimiter and returns a collection containing each section of the split string:

```php
$collection = Str::of('foo bar baz')->explode(' ');

// collect(['foo', 'bar', 'baz'])
```

finish

The `finish` method adds a single instance of the given value to a string if it does not already end with that value:

```php
$adjusted = Str::of('this/string')->finish('/'); // this/string/

$adjusted = Str::of('this/string/')->finish('/'); // this/string/
```

is

The `is` method determines if a given string matches a given pattern. Asterisks may be used as wildcard values

```php
$matches = Str::of('foobar')->is('foo*'); // true

$matches = Str::of('foobar')->is('baz*'); // false
```

isAscii

The `isAscii` method determines if a given string is an ASCII string:

```php
$result = Str::of('Taylor')->isAscii(); // true

$result = Str::of('ü')->isAscii(); // false
```

isEmpty

The `isEmpty` method determines if the given string is empty:

```
$result = Str::of('  ')->trim()->isEmpty(); // true

$result = Str::of('Laravel')->trim()->isEmpty(); // false
```

isNotEmpty

The `isNotEmpty` method determines if the given string is not empty:

```
$result = Str::of('  ')->trim()->isNotEmpty(); // false

$result = Str::of('Laravel')->trim()->isNotEmpty(); // true
```

isUuid

The `isUuid` method determines if a given string is a UUID:

```
$result = Str::of('5ace9ab9-e9cf-4ec6-a19d-5881212a452c')->isUuid(); // true

$result = Str::of('Taylor')->isUuid(); // false
```

kebab

The `kebab` method converts the given string to `kebab-case`:

```
$converted = Str::of('fooBar')->kebab(); // foo-bar
```

length

The `length` method returns the length of the given string:

```
$length = Str::of('Laravel')->length(); // 7
```

limit

The `limit` method truncates the given string to the specified length:

```
$truncated = Str::of('The quick brown fox jumps over the lazy dog')->limit(20);

// The quick brown fox...
```

You may also pass a second argument to change the string that will be appended to the end of the truncated string:

```
$truncated = Str::of('The quick brown fox jumps over the lazy dog')->limit(20, '
(...)');

// The quick brown fox (...)
```

lower

The `lower` method converts the given string to lowercase:

```
$result = Str::of('LARAVEL')->lower(); // 'laravel'
```

ltrim

The `ltrim` method trims the left side of the string:

```
$string = Str::of('  Laravel  ')->ltrim(); // 'Laravel  '

$string = Str::of('/Laravel/')->ltrim('/'); // 'Laravel/'
```

markdown

The `markdown` method converts GitHub flavored Markdown into HTML:

```
$html = Str::of('# Laravel')->markdown(); // <h1>Laravel</h1>

$html = Str::of('# Taylor <b>Otwell</b>')->markdown([

    'html_input' => 'strip',

]);
```

```
// <h1>Taylor Otwell</h1>
```

match

The `match` method will return the portion of a string that matches a given regular expression pattern:

```
$result = Str::of('foo bar')->match('/bar/'); // 'bar'

$result = Str::of('foo bar')->match('/foo (.*)/'); // 'bar'
```

matchAll

The `matchAll` method will return a collection containing the portions of a string that match a given regular expression pattern:

```
$result = Str::of('bar foo bar')->matchAll('/bar/'); // collect(['bar', 'bar'])
```

If you specify a matching group within the expression, Laravel will return a collection of that group's matches:

```
$result = Str::of('bar fun bar fly')->matchAll('/f(\w*)/');

// collect(['un', 'ly']);
```

If no matches are found, an empty collection will be returned.

padBoth

The `padBoth` method wraps PHP's `str_pad` function, padding both sides of a string with another string until the final string reaches the desired length:

```
$padded = Str::of('James')->padBoth(10, '_'); // '__James___'

$padded = Str::of('James')->padBoth(10); // '  James   '
```

padLeft

The `padLeft` method wraps PHP's `str_pad` function, padding the left side of a string with another string until the final string reaches the desired length:

```
$padded = Str::of('James')->padLeft(10, '-='); // '-=-=-James'

$padded = Str::of('James')->padLeft(10); // '     James'
```

padRight

The `padRight` method wraps PHP's `str_pad` function, padding the right side of a string with another string until the final string reaches the desired length:

```
$padded = Str::of('James')->padRight(10, '-'); // 'James-----'

$padded = Str::of('James')->padRight(10); // 'James     '
```

pipe

The `pipe` method allows you to transform the string by passing its current value to the given callable:

```
$hash = Str::of('Laravel')->pipe('md5')->prepend('Checksum: ');

// 'Checksum: a5c95b86291ea299fcbe64458ed12702'

$closure = Str::of('foo')->pipe(function ($str) {

    return 'bar';

});

// 'bar'
```

plural

The `plural` method converts a singular word string to its plural form. This function currently only supports the English language:

```
$plural = Str::of('car')->plural(); // cars

$plural = Str::of('child')->plural(); // children
```

You may provide an integer as a second argument to the function to retrieve the singular or plural form of the string:

```
$plural = Str::of('child')->plural(2); // children

$plural = Str::of('child')->plural(1); // child
```

prepend

The `prepend` method prepends the given values onto the string:

```
$string = Str::of('Framework')->prepend('Laravel '); // Laravel Framework
```

remove

The `remove` method removes the given value or array of values from the string:

```
$string = Str::of('Arkansas is quite beautiful!')->remove('quite');

// Arkansas is beautiful!
```

You may also pass `false` as a second parameter to ignore case when removing.

replace

The `replace` method replaces a given string within the string:

```
$replaced = Str::of('Laravel 6.x')->replace('6.x', '7.x'); // Laravel 7.x
```

replaceArray

The `replaceArray` method replaces a given value in the string sequentially using an array:

```
$string = 'The event will take place between ? and ?';

$replaced = Str::of($string)->replaceArray('?', ['8:30', '9:00']);

// The event will take place between 8:30 and 9:00
```

replaceFirst

The `replaceFirst` method replaces the first occurrence of a given value in a string:

```
$replaced = Str::of('the quick brown fox jumps over the lazy dog')-
>replaceFirst('the', 'a');

// a quick brown fox jumps over the lazy dog
```

replaceLast

The `replaceLast` method replaces the last occurrence of a given value in a string:

```
$replaced = Str::of('the quick brown fox jumps over the lazy dog')-
>replaceLast('the', 'a');

// the quick brown fox jumps over a lazy dog
```

replaceMatches

The `replaceMatches` method replaces all portions of a string matching a pattern with the given replacement string:

```
$replaced = Str::of('(+1) 501-555-1000')->replaceMatches('/[^A-Za-z0-9]++/', '')

// '15015551000'
```

The `replaceMatches` method also accepts a closure that will be invoked with each portion of the string matching the given pattern, allowing you to perform the replacement logic within the closure and return the replaced value:

```
$replaced = Str::of('123')->replaceMatches('/\d/', function ($match) {
    return '['.$match[0].']';

});

// '[1][2][3]'
```

rtrim

The `rtrim` method trims the right side of the given string:

```
$string = Str::of('  Laravel  ')->rtrim(); // '  Laravel'

$string = Str::of('/Laravel/')->rtrim('/'); // '/Laravel'
```

singular

The `singular` method converts a string to its singular form. This function currently only supports the English language:

```
$singular = Str::of('cars')->singular(); // car

$singular = Str::of('children')->singular(); // child
```

slug

The `slug` method generates a URL friendly "slug" from the given string:

```
$slug = Str::of('Laravel Framework')->slug('-'); // laravel-framework
```

snake

The `snake` method converts the given string to `snake_case`:

```
$converted = Str::of('fooBar')->snake(); // foo_bar
```

split

The `split` method splits a string into a collection using a regular expression:

```
$segments = Str::of('one, two, three')->split('/[\s,]+/');

// collect(["one", "two", "three"])
```

start

The `start` method adds a single instance of the given value to a string if it does not already start with that value:

```
$adjusted = Str::of('this/string')->start('/'); // /this/string

$adjusted = Str::of('/this/string')->start('/'); // /this/string
```

startsWith

The `startsWith` method determines if the given string begins with the given value:

```
$result = Str::of('This is my name')->startsWith('This'); // true
```

studly

The `studly` method converts the given string to `StudlyCase`:

```
$converted = Str::of('foo_bar')->studly(); // FooBar
```

substr

The `substr` method returns the portion of the string specified by the given start and length parameters:

```
$string = Str::of('Laravel Framework')->substr(8); // Framework

$string = Str::of('Laravel Framework')->substr(8, 5); // Frame
```

tap

The `tap` method passes the string to the given closure, allowing you to examine and interact with the string while not affecting the string itself. The original string is returned by the `tap` method regardless of what is returned by the closure:

```
$string = Str::of('Laravel')

    ->append(' Framework')

    ->tap(function ($string) {

        dump('String after append: ' . $string);

    })

    ->upper();

// LARAVEL FRAMEWORK
```

test

The `test` method determines if a string matches the given regular expression pattern:

```
$result = Str::of('Laravel Framework')->test('/Laravel/'); // true
```

title

The `title` method converts the given string to `Title Case`:

```
$converted = Str::of('a nice title uses the correct case')->title();

// A Nice Title Uses The Correct Case
```

trim

The `trim` method trims the given string:

```
$string = Str::of('  Laravel  ')->trim(); // 'Laravel'

$string = Str::of('/Laravel/')->trim('/'); // 'Laravel'
```

ucfirst

The `ucfirst` method returns the given string with the first character capitalized:

```
$string = Str::of('foo bar')->ucfirst(); // Foo bar
```

upper

The `upper` method converts the given string to uppercase:

```
$adjusted = Str::of('laravel')->upper(); // LARAVEL
```

when

The `when` method invokes the given closure if a given condition is `true`. The closure will receive the fluent string instance:

```
$string = Str::of('Taylor')

                ->when(true, function ($string) {

                    return $string->append(' Otwell');

                });

// 'Taylor Otwell'
```

If necessary, you may pass another closure as the third parameter to the `when` method. This closure will execute if the condition parameter evaluates to `false`.

whenEmpty

The `whenEmpty` method invokes the given closure if the string is empty. If the closure returns a value, that value will also be returned by the `whenEmpty` method. If the closure does not return a value, the fluent string instance will be returned:

```
$string = Str::of('  ')->whenEmpty(function ($string) {

    return $string->trim()->prepend('Laravel');
});

// 'Laravel'
```

wordCount

The `wordCount` function returns the number of words that a string contains:

```
Str::of('Hello, world!')->wordCount(); // 2
```

words

The `words` method limits the number of words in a string. If necessary, you may specify an additional string that will be appended to the truncated string:

```
$string = Str::of('Perfectly balanced, as all things should be.')->words(3, '
>>>');

// Perfectly balanced, as >>>
```