Course - Laravel Framework

Helper

Laravel bao gồm nhiều hàm PHP thuần "helper". Nhiều hàm trong số này được sử dụng bởi chính framework; tuy nhiên, bạn có thể tự do sử dụng chúng trong các ứng dụng của riêng mình nếu bạn thấy thuận tiện.

Tags: helper, laravel 8

Giới thiệu

Laravel bao gồm nhiều hàm PHP thuần "helper". Nhiều hàm trong số này được sử dụng bởi chính framework; tuy nhiên, bạn có thể tự do sử dụng chúng trong các ứng dụng của riêng mình nếu bạn thấy thuận tiện.

Các phương thức có sẵn

Mảng và các đối tượng

`Arr::accessible()`

Phương thức `Arr::accessible` sẽ xác định xem giá trị đã cho có thể truy cập mảng hay không:

```
use Illuminate\Support\Arr;
use Illuminate\Support\Collection;

$isAccessible = Arr::accessible(['a' => 1, 'b' => 2]);

// true

$isAccessible = Arr::accessible(new Collection);

// true

$isAccessible = Arr::accessible('abc');

// false

$isAccessible = Arr::accessible(new stdClass);

// false
```

`Arr::add()`

Phương thức `Arr::add` sẽ thêm một cặp <khóa/giá trị> đã cho vào một mảng nếu khóa đã cho chưa tồn tại trong mảng hoặc được đặt thành `null`:

```
use Illuminate\Support\Arr;

$array = Arr::add(['name' => 'Desk'], 'price', 100);

// ['name' => 'Desk', 'price' => 100]

$array = Arr::add(['name' => 'Desk', 'price' => null], 'price', 100);

// ['name' => 'Desk', 'price' => 100]
```

Arr::collapse()

Phương thức `Arr::collapse()` sẽ thu gọn một mảng tập hợp các mảng thành một mảng duy nhất:

```
use Illuminate\Support\Arr;

$array = Arr::collapse([[1, 2, 3], [4, 5, 6], [7, 8, 9]]);

// [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Arr::crossJoin()

Phương thức `Arr::crossJoin()` sẽ kết hợp các mảng đã cho, trả về một tích Cartesian với tất cả các hoán vị có thể có:

```
use Illuminate\Support\Arr;

$matrix = Arr::crossJoin([1, 2], ['a', 'b']);

/*
  [
    [1, 'a'],
```

```
    [1, 'b'],
    [2, 'a'],
    [2, 'b'],
  ]
*/

$matrix = Arr::crossJoin([1, 2], ['a', 'b'], ['I', 'II']);

/*
  [
    [1, 'a', 'I'],
    [1, 'a', 'II'],
    [1, 'b', 'I'],
    [1, 'b', 'II'],
    [2, 'a', 'I'],
    [2, 'a', 'II'],
    [2, 'b', 'I'],
    [2, 'b', 'II'],
  ]
*/
```

`Arr::divide()`

Phương thức `Arr::divide()` trả về hai mảng: một mảng chứa các khóa và mảng kia chứa các giá trị của mảng đã cho:

```
use Illuminate\Support\Arr;

[$keys, $values] = Arr::divide(['name' => 'Desk']);

// $keys: ['name']

// $values: ['Desk']
```

`Arr::dot()`

Phương thức `Arr::dot()` làm phẳng một mảng đa chiều thành một mảng một cấp duy nhất sử dụng ký hiệu "dấu chấm" để biểu thị độ sâu:

```php
use Illuminate\Support\Arr;

$array = ['products' => ['desk' => ['price' => 100]]];

$flattened = Arr::dot($array);

// ['products.desk.price' => 100]
```

Arr::except()

Phương thức `Arr::except()` sẽ loại bỏ các cặp khóa/giá trị đã cho khỏi một mảng:

```php
use Illuminate\Support\Arr;

$array = ['name' => 'Desk', 'price' => 100];

$filtered = Arr::except($array, ['price']);

// ['name' => 'Desk']
```

Arr::exists()

Phương thức `Arr::exists()` sẽ kiểm tra xem khóa đã cho có tồn tại trong mảng được cung cấp hay không:

```php
use Illuminate\Support\Arr;

$array = ['name' => 'John Doe', 'age' => 17];

$exists = Arr::exists($array, 'name');

// true
```

```php
$exists = Arr::exists($array, 'salary');

// false
```

Arr::first()

Phương thức `Arr::first()` sẽ trả về phần tử đầu tiên của một mảng vượt qua một bài kiểm tra tuyển lọc nào đó:

```php
use Illuminate\Support\Arr;

$array = [100, 200, 300];

$first = Arr::first($array, function ($value, $key) {
    return $value >= 150;
});

// 200
```

Giá trị mặc định cũng có thể được truyền vào đối số thứ ba cho phương thức. Giá trị này sẽ được trả về nếu không có giá trị nào vượt qua bài kiểm tra tuyển lọc cụ thể:

```php
use Illuminate\Support\Arr;

$first = Arr::first($array, $callback, $default);
```

Arr::flatten()

Phương thức `Arr::flatten()` làm phẳng một mảng đa chiều thành một mảng một cấp duy nhất:

```php
use Illuminate\Support\Arr;

$array = ['name' => 'Joe', 'languages' => ['PHP', 'Ruby']];
```

```php
$flattened = Arr::flatten($array);

// ['Joe', 'PHP', 'Ruby']
```

Arr::forget()

Phương thức `Arr::forget()` sẽ xóa một cặp khóa/giá trị nhất định khỏi một mảng được lồng sâu vào nhau bằng cách sử dụng ký hiệu "dấu chấm":

```php
use Illuminate\Support\Arr;

$array = ['products' => ['desk' => ['price' => 100]]];

Arr::forget($array, 'products.desk');

// ['products' => []]
```

Arr::get()

Phương thức `Arr::get()` lấy một giá trị từ một mảng lồng sâu với nhau bằng cách sử dụng ký hiệu "dấu chấm":

```php
use Illuminate\Support\Arr;

$array = ['products' => ['desk' => ['price' => 100]]];

$price = Arr::get($array, 'products.desk.price');

// 100
```

Phương thức `Arr::get()` cũng chấp nhận một giá trị mặc định, giá trị này sẽ được trả về nếu khóa được chỉ định không có trong mảng:

```php
use Illuminate\Support\Arr;
```

```php
$discount = Arr::get($array, 'products.desk.discount', 0);

// 0
```

Arr::has()

Phương thức `Arr::has()` kiểm tra xem một mục nhất định hoặc các mục có tồn tại trong một mảng hay không bằng cách sử dụng ký hiệu "dấu chấm":

```php
use Illuminate\Support\Arr;

$array = ['product' => ['name' => 'Desk', 'price' => 100]];

$contains = Arr::has($array, 'product.name');

// true

$contains = Arr::has($array, ['product.price', 'product.discount']);

// false
```

Arr::hasAny()

Phương thức `Arr::hasAny()` kiểm tra xem có bất kỳ mục nào trong một tập hợp đã cho tồn tại trong một mảng hay không bằng cách sử dụng ký hiệu "dấu chấm":

```php
use Illuminate\Support\Arr;

$array = ['product' => ['name' => 'Desk', 'price' => 100]];

$contains = Arr::hasAny($array, 'product.name');

// true

$contains = Arr::hasAny($array, ['product.name', 'product.discount']);
```

```php
// true

$contains = Arr::hasAny($array, ['category', 'product.discount']);

// false
```

Arr::isAssoc()

Phương thức `Arr::isAssoc()` sẽ trả về `true` nếu mảng đã cho là mảng kết hợp. Một mảng được coi là "kết hợp" nếu nó không có các khóa số tuần tự bắt đầu bằng 0:

```php
use Illuminate\Support\Arr;

$isAssoc = Arr::isAssoc(['product' => ['name' => 'Desk', 'price' => 100]]);

// true

$isAssoc = Arr::isAssoc([1, 2, 3]);

// false
```

Arr::last()

Phương thức `Arr::last()` trả về phần tử cuối cùng của một mảng vượt qua một bài kiểm tra tuyển lọc nào đó:

```php
use Illuminate\Support\Arr;

$array = [100, 200, 300, 110];

$last = Arr::last($array, function ($value, $key) {
    return $value >= 150;
});

// 300
```

Giá trị mặc định có thể được truyền vào đối số thứ ba cho phương thức. Giá trị này sẽ được trả về nếu không có giá trị nào vượt qua bài kiểm tra tuyển lọc cụ thể:

```
use Illuminate\Support\Arr;

$last = Arr::last($array, $callback, $default);
```

Arr::only()

Phương thức `Arr::only()` chỉ trả về các cặp khóa/giá trị được chỉ định từ mảng đã cho:

```
use Illuminate\Support\Arr;

$array = ['name' => 'Desk', 'price' => 100, 'orders' => 10];

$slice = Arr::only($array, ['name', 'price']);

// ['name' => 'Desk', 'price' => 100]
```

Arr::pluck()

Phương thức `Arr::pluck()` sẽ lấy tất cả các giá trị cho một khóa đã cho từ một mảng bằng cách sử dụng cú pháp "dấu chấm":

```
use Illuminate\Support\Arr;

$array = [
    ['developer' => ['id' => 1, 'name' => 'Taylor']],
    ['developer' => ['id' => 2, 'name' => 'Abigail']],
];

$names = Arr::pluck($array, 'developer.name');

// ['Taylor', 'Abigail']
```

Bạn cũng có thể chỉ định cách danh sách kết quả được kết hợp với khóa mảng:

```
use Illuminate\Support\Arr;

$names = Arr::pluck($array, 'developer.name', 'developer.id');

// [1 => 'Taylor', 2 => 'Abigail']
```

Arr::prepend()

Phương thức `Arr::prepend()` sẽ đưa một mục tiêu bên ngoài vào hàng đầu của một mảng:

```
use Illuminate\Support\Arr;

$array = ['one', 'two', 'three', 'four'];

$array = Arr::prepend($array, 'zero');

// ['zero', 'one', 'two', 'three', 'four']
```

Nếu cần, bạn cũng có thể chỉ định khóa sẽ được kết hợp cho giá trị:

```
use Illuminate\Support\Arr;

$array = ['price' => 100];

$array = Arr::prepend($array, 'Desk', 'name');

// ['name' => 'Desk', 'price' => 100]
```

Arr::pull()

Phương thức `Arr::pull()` sẽ lấy về và xóa một cặp khóa/giá trị khỏi một mảng:

```php
use Illuminate\Support\Arr;

$array = ['name' => 'Desk', 'price' => 100];

$name = Arr::pull($array, 'name');

// $name: Desk

// $array: ['price' => 100]
```

Giá trị mặc định có thể được truyền vào đối số thứ ba cho phương thức. Giá trị này sẽ được trả về nếu khóa không tồn tại:

```php
use Illuminate\Support\Arr;

$value = Arr::pull($array, $key, $default);
```

`Arr::query()`

Phương thức `Arr::query()` chuyển đổi mảng thành một chuỗi truy vấn URL:

```php
use Illuminate\Support\Arr;

$array = [
  'name' => 'Taylor',
  'order' => [
    'column' => 'created_at',
    'direction' => 'desc'
  ]
];

Arr::query($array);

// name=Taylor&order[column]=created_at&order[direction]=desc
```

Arr::random()

Phương thức `Arr::random()` sẽ trả về một giá trị ngẫu nhiên từ một mảng:

```
use Illuminate\Support\Arr;

$array = [1, 2, 3, 4, 5];

$random = Arr::random($array);

// 4 - (retrieved randomly)
```

Bạn cũng có thể chỉ định số lượng mục được trả về bằng đối số thứ hai, đối số này không bắt buộc phải khai báo. Lưu ý rằng việc cung cấp đối số này sẽ trả về một mảng ngay cả khi chỉ có một mục:

```
use Illuminate\Support\Arr;

$items = Arr::random($array, 2);

// [2, 5] - (retrieved randomly)
```

Arr::set()

Phương thức `Arr::set()` sẽ đặt một giá trị trong một mảng lồng sâu vào nhau bằng cách sử dụng ký hiệu "dấu chấm":

```
use Illuminate\Support\Arr;

$array = ['products' => ['desk' => ['price' => 100]]];

Arr::set($array, 'products.desk.price', 200);

// ['products' => ['desk' => ['price' => 200]]]
```

Arr::shuffle()

Phương thức `Arr::shuffle()` sẽ trộn ngẫu nhiên thứ tự các phần tử trong mảng:

```php
use Illuminate\Support\Arr;

$array = Arr::shuffle([1, 2, 3, 4, 5]);

// [3, 2, 5, 1, 4] - (generated randomly)
```

Arr::sort()

Phương thức `Arr::sort()` sẽ sắp xếp một mảng theo thứ tự alphabet các giá trị của nó:

```php
use Illuminate\Support\Arr;

$array = ['Desk', 'Table', 'Chair'];

$sorted = Arr::sort($array);

// ['Chair', 'Desk', 'Table']
```

Bạn cũng có thể sắp xếp mảng theo kết quả của một hàm đã cho:

```php
use Illuminate\Support\Arr;

$array = [
  ['name' => 'Desk'],
  ['name' => 'Table'],
  ['name' => 'Chair'],
];

$sorted = array_values(Arr::sort($array, function ($value) {
  return $value['name'];
}));

/*
  [
```

```
    ['name' => 'Chair'],
    ['name' => 'Desk'],
    ['name' => 'Table'],
  ]
*/
```

Arr::sortRecursive()

Phương thức `Arr::sortRecursive()` sẽ sắp xếp đệ quy một mảng bằng cách sử dụng hàm sắp xếp cho các mảng con được lập chỉ mục số và hàm `ksort` cho các mảng kết hợp con:

```
use Illuminate\Support\Arr;

$array = [
  ['Roman', 'Taylor', 'Li'],
  ['PHP', 'Ruby', 'JavaScript'],
  ['one' => 1, 'two' => 2, 'three' => 3],
];

$sorted = Arr::sortRecursive($array);

/*
  [
    ['JavaScript', 'PHP', 'Ruby'],
    ['one' => 1, 'three' => 3, 'two' => 2],
    ['Li', 'Roman', 'Taylor'],
  ]
*/
```

Arr::toCssClasses()

`Arr::toCssClasses()` sẽ biên dịch có điều kiện một chuỗi lớp CSS. Phương thức chấp nhận một mảng các class trong đó khóa mảng chứa class hoặc các class mà bạn muốn thêm vào, trong khi giá trị là một biểu thức `boolean`. Nếu phần tử mảng có khóa số, nó sẽ luôn được đưa vào danh sách class được hiển thị:

```php
use Illuminate\Support\Arr;

$isActive = false;
$hasError = true;

$array = ['p-4', 'font-bold' => $isActive, 'bg-red' => $hasError];

$classes = Arr::toCssClasses($array);

/*
  'p-4 bg-red'
*/
```

Phương thức này hỗ trợ chức năng của Laravel cho phép hợp nhất các class với bộ thuộc tính của component Blade cũng như chỉ thị `@class` Blade.

`Arr::undot()`

Phương thức `Arr::undot()` mở rộng mảng một chiều sử dụng ký hiệu "dấu chấm" thành một mảng đa chiều:

```php
use Illuminate\Support\Arr;

$array = [
  'user.name' => 'Kevin Malone',
  'user.occupation' => 'Accountant',
];

$array = Arr::undot($array);

// ['user' => ['name' => 'Kevin Malone', 'occupation' => 'Accountant']]
```

`Arr::where()`

Phương thức `Arr::where()` sẽ tuyển lọc một mảng bằng cách sử dụng hàm nặc danh đã cho:

```php
use Illuminate\Support\Arr;

$array = [100, '200', 300, '400', 500];

$filtered = Arr::where($array, function ($value, $key) {
  return is_string($value);
});

// [1 => '200', 3 => '400']
```

`Arr::whereNotNull()`

Phương thức `Arr::whereNotNull()` sẽ loại bỏ tất cả các giá trị `null` khỏi mảng đã cho:

```php
use Illuminate\Support\Arr;

$array = [0, null];

$filtered = Arr::whereNotNull($array);

// [0 => 0]
```

`Arr::wrap()`

Phương thức `Arr::wrap()` sẽ bao bọc giá trị đã cho trong một mảng. Nếu giá trị đã cho đã là một mảng, nó sẽ được trả về mà không cần bất kỳ chuyển đổi nào nữa:

```php
use Illuminate\Support\Arr;

$string = 'Laravel';

$array = Arr::wrap($string);

// ['Laravel']
```

Nếu giá trị đã cho là `null`, một mảng trống sẽ được trả về:

```
use Illuminate\Support\Arr;

$array = Arr::wrap(null);

// []
```

data_fill()

Hàm `data_fill` đặt một giá trị bị thiếu trong một mảng hoặc đối tượng lồng nhau bằng cách sử dụng ký hiệu "dấu chấm":

```
$data = ['products' => ['desk' => ['price' => 100]]];

data_fill($data, 'products.desk.price', 200);

// ['products' => ['desk' => ['price' => 100]]]

data_fill($data, 'products.desk.discount', 10);

// ['products' => ['desk' => ['price' => 100, 'discount' => 10]]]
```

Hàm này cũng chấp nhận dấu hoa thị làm ký tự đại diện và sẽ điền vào mục tiêu tương ứng:

```
$data = [
  'products' => [
    ['name' => 'Desk 1', 'price' => 100],
    ['name' => 'Desk 2'],
  ],
];

data_fill($data, 'products.*.price', 200);

/*
```

```
  [
    'products' => [
      ['name' => 'Desk 1', 'price' => 100],
      ['name' => 'Desk 2', 'price' => 200],
    ],
  ]
*/
```

data_get()

Hàm `data_get` sẽ truy xuất một giá trị từ một mảng hoặc đối tượng lồng nhau bằng cách sử dụng ký hiệu "dấu chấm":

```
$data = ['products' => ['desk' => ['price' => 100]]];

$price = data_get($data, 'products.desk.price');

// 100
```

Hàm `data_get` cũng chấp nhận một giá trị mặc định, giá trị này sẽ được trả về nếu không tìm thấy khóa được chỉ định:

```
$discount = data_get($data, 'products.desk.discount', 0);

// 0
```

Hàm cũng chấp nhận các ký tự đại diện sử dụng dấu hoa thị, có thể nhắm mục tiêu bất kỳ khóa nào của mảng hoặc đối tượng:

```
$data = [
  'product-one' => ['name' => 'Desk 1', 'price' => 100],
  'product-two' => ['name' => 'Desk 2', 'price' => 150],
];

data_get($data, '*.name');
```

```
// ['Desk 1', 'Desk 2'];
```

`data_set()`

Hàm `data_set` sẽ cập nhật giá trị cho một phần tử của một mảng hoặc đối tượng lồng nhau bằng cách sử dụng ký hiệu "dấu chấm":

```
$data = ['products' => ['desk' => ['price' => 100]]];

data_set($data, 'products.desk.price', 200);

// ['products' => ['desk' => ['price' => 200]]]
```

Hàm này cũng chấp nhận các ký tự đại diện sử dụng dấu hoa thị và sẽ cập nhật các giá trị trên phần tử phù hợp:

```
$data = [
 'products' => [
   ['name' => 'Desk 1', 'price' => 100],
   ['name' => 'Desk 2', 'price' => 150],
 ],
];

data_set($data, 'products.*.price', 200);

/*
  [
    'products' => [
      ['name' => 'Desk 1', 'price' => 200],
      ['name' => 'Desk 2', 'price' => 200],
    ],
  ]
*/
```

Theo mặc định, mọi giá trị hiện có đều bị ghi đè. Nếu bạn muốn chỉ cập nhật một giá trị nếu nó không tồn tại, bạn có thể chuyển `false` làm đối số thứ tư cho hàm:

```php
$data = ['products' => ['desk' => ['price' => 100]]];

data_set($data, 'products.desk.price', 200, $overwrite = false);

// ['products' => ['desk' => ['price' => 100]]]
```

`head()`

Hàm `head()` sẽ trả về phần tử đầu tiên trong mảng đã cho:

```php
$array = [100, 200, 300];

$first = head($array);

// 100
```

`last()`

Hàm `last()` sẽ trả về phần tử cuối cùng của mảng đã cho:

```php
$array = [100, 200, 300];

$last = last($array);

// 300
```

Paths (Đường dẫn)

`app_path()`

Hàm `app_path` trả về đường dẫn đến thư mục *app* của ứng dụng của bạn. Bạn cũng có thể sử dụng hàm `app_path` để tạo một đường dẫn đến một tập tin liên quan đến thư mục *app*:

```php
$path = app_path();
```

```php
$path = app_path('Http/Controllers/Controller.php');
```

base_path()

Hàm `base_path` trả về đường dẫn đủ điều kiện đến thư mục gốc của ứng dụng của bạn. Bạn cũng có thể sử dụng hàm `base_path` để tạo một đường dẫn đến một tập tin liên quan đến thư mục gốc của dự án:

```php
$path = base_path();

$path = base_path('vendor/bin');
```

config_path()

Hàm `config_path` trả về đường dẫn đến thư mục *config* của ứng dụng của bạn. Bạn cũng có thể sử dụng hàm `config_path` để tạo một đường dẫn đến một tập tin liên quan thư mục *config* của ứng dụng:

```php
$path = config_path();

$path = config_path('app.php');
```

database_path()

Hàm `database_path` trả về đường dẫn đến thư mục *database* của ứng dụng của bạn. Bạn cũng có thể sử dụng hàm `database_path` để tạo một đường dẫn đến một tập tin nhất định trong thư mục *database*:

```php
$path = database_path();

$path = database_path('factories/UserFactory.php');
```

mix()

Hàm `mix` sẽ trả về đường dẫn đến tập tin Mix được đánh phiên bản:

```
$path = mix('css/app.css');
```

public_path()

Hàm `public_path` sẽ trả về đường dẫn đến thư mục *public* của ứng dụng của bạn. Bạn cũng có thể sử dụng hàm `public_path` để tạo một đường dẫn đến một tập tin nào đó liên quan thư mục *public*:

```
$path = public_path();

$path = public_path('css/app.css');
```

resource_path()

Hàm `resource_path` sẽ trả về đường dẫn đến thư mục *resource* của ứng dụng của bạn. Bạn cũng có thể sử dụng hàm `resource_path` để tạo một đường dẫn đến một tập tin nhất định trong thư mục *resource*:

```
$path = resource_path();

$path = resource_path('sass/app.scss');
```

storage_path()

Hàm `storage_path` trả về đường dẫn đến thư mục *storage* của ứng dụng của bạn. Bạn cũng có thể sử dụng hàm `storage_path` để tạo một đường dẫn đến một tập tin nào đó liên quan thư mục *storage*:

```
$path = storage_path();

$path = storage_path('app/file.txt');
```

String (Chuỗi câu)

__()

Hàm __() dịch chuỗi thông dịch nào đó hoặc khóa dịch bằng cách sử dụng các tập tin việt hóa của bạn:

```
echo __('Welcome to our application');

echo __('messages.welcome');
```

Nếu chuỗi hoặc khóa thông dịch được chỉ định không tồn tại, hàm __() sẽ trả về giá trị đã cho. Vì vậy, khi sử dụng ví dụ trên, hàm __() sẽ trả về nguyên văn chuỗi "messages.welcome" nếu khóa thông dịch này không tồn tại.

class_basename()

Hàm class_basename sẽ trả về tên class của class đã cho với namespace của class đó bị lượt xóa đi:

```
$class = class_basename('Foo\Bar\Baz');

// Baz
```

e()

Hàm e() chạy hàm htmlspecialchars của PHP thuần với tùy chọn double_encode được đặt thành true theo mặc định:

```
echo e('<html>foo</html>');

// &lt;html&gt;foo&lt;/html&gt;
```

preg_replace_array()

Hàm preg_replace_array thay thế một biểu thức mẫu đã cho trong chuỗi một cách

tuần tự từ đầu đến cuối bằng cách sử dụng một mảng:

```
$string = 'The event will take place between :start and :end';

$replaced = preg_replace_array('/:[a-z_]+/', ['8:30', '9:00'], $string);

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

Str::after()

Phương thức `Str::after()` trả về mọi thứ đằng sau giá trị đã cho trong một chuỗi câu. Toàn bộ chuỗi câu sẽ được trả về nếu giá trị không tồn tại trong chuỗi câu:

```
use Illuminate\Support\Str;

$slice = Str::after('This is my name', 'This is');

// ' my name'
```

Str::afterLast()

Phương thức `Str::afterLast()` trả về mọi thứ đằng sau lần xuất hiện cuối cùng của giá trị đã cho trong một chuỗi câu. Toàn bộ chuỗi câu sẽ được trả về nếu giá trị không tồn tại trong chuỗi câu:

```
use Illuminate\Support\Str;

$slice = Str::afterLast('App\Http\Controllers\Controller', '\\');

// 'Controller'
```

Str::ascii()

Phương thức `Str::ascii()` sẽ cố gắng chuyển đổi chuỗi câu thành giá trị ASCII:

```php
use Illuminate\Support\Str;

$slice = Str::ascii('û');

// 'u'
```

Str::before()

Phương thức `Str::before()` sẽ trả về mọi thứ đằng trước giá trị đã cho trong một chuỗi câu:

```php
use Illuminate\Support\Str;

$slice = Str::before('This is my name', 'my name');

// 'This is '
```

Str::beforeLast()

Phương thức `Str::beforeLast()` sẽ trả về mọi thứ đằng trước lần xuất hiện cuối cùng của giá trị đã cho trong một chuỗi câu:

```php
use Illuminate\Support\Str;

$slice = Str::beforeLast('This is my name', 'is');

// 'This '
```

Str::between()

Phương thức `Str::between()` sẽ trả về một phần của chuỗi câu giữa hai giá trị đã cho:

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

```php
$slice = Str::between('This is my name', 'This', 'name');

// ' is my '
```

Str::camel()

Phương thức `Str::camel()` sẽ chuyển đổi chuỗi câu đã cho thành kiểu chuỗi câu `camelCase`:

```php
use Illuminate\Support\Str;

$converted = Str::camel('foo_bar');

// fooBar
```

Str::contains()

Phương thức `Str::contains()` sẽ xác định xem chuỗi câu đã cho có chứa giá trị đã cho hay không. Phương thức này phân biệt chữ hoa chữ thường:

```php
use Illuminate\Support\Str;

$contains = Str::contains('This is my name', 'my');

// true
```

Bạn cũng có thể chuyển đổi một mảng giá trị để xác định xem chuỗi câu đã cho có chứa bất kỳ giá trị nào trong mảng hay không:

```php
use Illuminate\Support\Str;

$contains = Str::contains('This is my name', ['my', 'foo']);

// true
```

Str::containsAll()

Phương thức Str::containsAll() sẽ xác định xem chuỗi câu đã cho có chứa tất cả các giá trị trong một mảng nhất định hay không:

```
use Illuminate\Support\Str;

$containsAll = Str::containsAll('This is my name', ['my', 'name']);

// true
```

Str::endsWith()

Phương thức `Str::endsWith()` sẽ xác định xem chuỗi câu đã cho có kết thúc bằng giá trị đã cho hay không:

```
use Illuminate\Support\Str;

$result = Str::endsWith('This is my name', 'name');

// true
```

Bạn cũng có thể truyền một mảng giá trị để xác định xem chuỗi câu đã cho có kết thúc bằng bất kỳ giá trị nào trong mảng hay không:

```
use Illuminate\Support\Str;

$result = Str::endsWith('This is my name', ['name', 'foo']);

// true

$result = Str::endsWith('This is my name', ['this', 'foo']);

// false
```

Str::finish()

Phương thức `Str::finish()` thêm một đối tượng đơn nhất của giá trị đã cho vào một chuỗi câu nếu nó chưa kết thúc bằng giá trị đó:

```
use Illuminate\Support\Str;

$adjusted = Str::finish('this/string', '/');

// this/string/

$adjusted = Str::finish('this/string/', '/');

// this/string/
```

Str::headline()

Phương thức `Str::headline()` sẽ chuyển đổi các chuỗi câu được phân tách bằng cách viết hoa, dấu gạch nối hoặc dấu gạch dưới thành một chuỗi được phân tách bằng dấu cách với chữ cái đầu tiên của mỗi từ được viết in hoa:

```
use Illuminate\Support\Str;

$headline = Str::headline('steve_jobs');

// Steve Jobs

$headline = Str::headline('EmailNotificationSent');

// Email Notification Sent
```

Str::is()

Phương thức `Str::is()` sẽ xác định xem một chuỗi câu nhất định có khớp với một biểu thức mẫu nhất định hay không. Dấu hoa thị có thể được sử dụng làm giá trị ký tự đại diện:

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

```php
$matches = Str::is('foo*', 'foobar');

// true

$matches = Str::is('baz*', 'foobar');

// false
```

Str::isAscii()

Phương thức `Str::isAscii()` sẽ xác định xem một chuỗi câu đã cho có phải là ASCII 7 bit hay không:

```php
use Illuminate\Support\Str;

$isAscii = Str::isAscii('Taylor');

// true

$isAscii = Str::isAscii('ü');

// false
```

Str::isUuid()

Phương thức `Str::isUuid()` sẽ xác định xem chuỗi câu đã cho có phải là UUID hợp lệ hay không:

```php
use Illuminate\Support\Str;

$isUuid = Str::isUuid('a0a2a2d2-0b87-4a18-83f2-2529882be2de');

// true

$isUuid = Str::isUuid('laravel');
```

```
// false
```

Str::kebab()

Phương thức `Str::kebab()` sẽ chuyển đổi chuỗi câu đã cho thành `kebab-case`:

```
use Illuminate\Support\Str;

$converted = Str::kebab('fooBar');

// foo-bar
```

Str::length()

Phương thức `Str::length()` sẽ trả về độ dài của chuỗi câu đã cho:

```
use Illuminate\Support\Str;

$length = Str::length('Laravel');

// 7
```

Str::limit()

Phương thức `Str::limit()` cắt ngắn chuỗi đã cho theo độ dài được chỉ định:

```
use Illuminate\Support\Str;

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

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

Bạn có thể truyền đối số thứ ba cho phương thức này để thay đổi chuỗi câu sẽ được nối vào cuối chuỗi bị cắt ngắn:

```php
use Illuminate\Support\Str;

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

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

Str::lower()

Phương thức `Str::lower()` sẽ chuyển đổi chuỗi câu đã cho thành chữ thường:

```php
use Illuminate\Support\Str;

$converted = Str::lower('LARAVEL');

// laravel
```

Str::markdown()

Phương thức `Str::markdown()` chuyển đổi Markdown theo phong cách GitHub thành HTML bằng CommonMark:

```php
use Illuminate\Support\Str;

$html = Str::markdown('# Laravel');

// <h1>Laravel</h1>

$html = Str::markdown('# Taylor <b>Otwell</b>', [
    'html_input' => 'strip',
]);

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

Str::mask()

Phương thức `Str::mask()` che lấp một phần chuỗi câu có ký tự lặp lại và có thể được sử dụng để làm xáo trộn các phân đoạn của chuỗi câu như địa chỉ email và số điện thoại:

```
use Illuminate\Support\Str;

$string = Str::mask('taylor@example.com', '*', 3);

// tay***************
```

Nếu cần, bạn cũng có thể cung cấp một số nguyên âm làm đối số thứ ba cho phương thức `mask`, điều này sẽ khiến cho phương thức bắt đầu tạo khoảng lấp ở khoảng cách cụ thể bắt đầu từ cuối chuỗi cầu về trước chuỗi câu:

```
$string = Str::mask('taylor@example.com', '*', -15, 3);

// tay***@example.com
```

Str::orderedUuid()

Phương thức `Str::orderedUuid()` sẽ tạo chuỗi UUID "dấu thời gian đầu tiên" có thể được lưu trữ hiệu quả trong cột cơ sở dữ liệu được lập chỉ mục index. Mỗi UUID như thế này khi được tạo bằng phương thức `orderedUuid` sẽ được xếp đứng sau các UUID đã được tạo trước đó:

```
use Illuminate\Support\Str;

return (string) Str::orderedUuid();
```

Str::padBoth()

Phương thức `Str::padBoth()` sẽ bao bọc hàm `str_pad` của PHP thuần, nghĩa là nó sẽ đệm thêm cả hai bên của một chuỗi câu bằng một chuỗi khác cho đến khi chuỗi kết quả cuối cùng đạt đến độ dài như bạn muốn:

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

```php
$padded = Str::padBoth('James', 10, '_');

// '__James___'

$padded = Str::padBoth('James', 10);

// '  James   '
```

Str::padLeft()

Phương thức `Str::padLeft()` sẽ bao bọc hàm `str_pad` của PHP thuần, nó sẽ đệm thêm bên trái của một chuỗi câu bằng một chuỗi câu khác cho đến khi chuỗi câu kết quả cuối cùng đạt đến độ dài như bạn muốn:

```php
use Illuminate\Support\Str;

$padded = Str::padLeft('James', 10, '-=');

// '-=-=-James'

$padded = Str::padLeft('James', 10);

// '     James'
```

Str::padRight()

Phương thức `Str::padRight()` sẽ bao bọc hàm `str_pad` của PHP thuần, nó sẽ đệm bên phải của một chuỗi câu bằng một chuỗi câu khác cho đến khi chuỗi câu kết quả cuối cùng đạt đến độ dài như bạn muốn:

```php
use Illuminate\Support\Str;

$padded = Str::padRight('James', 10, '-');

// 'James-----'
```

```php
$padded = Str::padRight('James', 10);

// 'James     '
```

Str::plural()

Phương thức Str::plural() sẽ chuyển đổi một chuỗi câu từ ngữ số ít sang dạng chuỗi câu từ ngữ số nhiều của nó. Chức năng này hỗ trợ bất kỳ ngôn ngữ nào được hỗ trợ bởi chương trình đa số hóa cho một danh từ của Laravel:

```php
use Illuminate\Support\Str;

$plural = Str::plural('car');

// cars

$plural = Str::plural('child');

// children
```

Bạn có thể cung cấp một số nguyên làm đối số thứ hai cho hàm để truy xuất dạng số ít hoặc số nhiều của chuỗi:

```php
use Illuminate\Support\Str;

$plural = Str::plural('child', 2);

// children

$singular = Str::plural('child', 1);

// child
```

Str::pluralStudly()

Phương thức `Str::pluralStudly()` sẽ chuyển đổi một chuỗi câu từ ngữ số ít được định dạng trong trường hợp chữ hoa đặc biệt sang dạng từ ngữ số nhiều của nó. Chức năng này hỗ trợ bất kỳ ngôn ngữ nào được hỗ trợ bởi chương trình đa số hóa cho một danh từ của Laravel:

```
use Illuminate\Support\Str;

$plural = Str::pluralStudly('VerifiedHuman');

// VerifiedHumans

$plural = Str::pluralStudly('UserFeedback');

// UserFeedback
```

Bạn có thể cung cấp một số nguyên làm đối số thứ hai cho hàm để truy xuất dạng số ít hoặc số nhiều của chuỗi:

```
use Illuminate\Support\Str;

$plural = Str::pluralStudly('VerifiedHuman', 2);

// VerifiedHumans

$singular = Str::pluralStudly('VerifiedHuman', 1);

// VerifiedHuman
```

`Str::random()`

Phương thức `Str::random()` tạo ra một chuỗi câu ngẫu nhiên có độ dài được chỉ định cụ thể. Hàm này sử dụng hàm `random_bytes` của PHP thuần:

```php
use Illuminate\Support\Str;

$random = Str::random(40);
```

Str::remove()

Phương thức Str::remove() sẽ loại bỏ giá trị cụ thể nào đó hoặc mảng giá trị nào đó ra khỏi chuỗi câu:

```php
use Illuminate\Support\Str;

$string = 'Peter Piper picked a peck of pickled peppers.';

$removed = Str::remove('e', $string);

// Ptr Pipr pickd a pck of pickld ppprs.
```

Bạn cũng có thể truyền giá trị `false` làm đối số thứ ba cho phương thức `remove` để bỏ qua trường hợp khi xóa chuỗi câu.

Str::replace()

Phương thức `Str::replace()` sẽ thay thế chuỗi câu đã cho trong một chuỗi nào đó:

```php
use Illuminate\Support\Str;

$string = 'Laravel 8.x';

$replaced = Str::replace('8.x', '9.x', $string);

// Laravel 9.x
```

Str::replaceArray()

Phương thức `Str::replaceArray()` sẽ thay thế một giá trị đã cho trong chuỗi câu một cách tuần tự bằng cách sử dụng một mảng:

```php
use Illuminate\Support\Str;

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

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

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

Str::replaceFirst()

Phương thức `Str::replaceFirst()` sẽ thay thế lần phát hiện đầu tiên của một giá trị đã cho trong một chuỗi câu:

```php
use Illuminate\Support\Str;

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

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

Str::replaceLast()

Phương thức `Str::replaceLast()` sẽ thay thế lần phát hiện cuối cùng của một giá trị đã cho trong một chuỗi câu:

```php
use Illuminate\Support\Str;

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

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

Str::reverse()

Phương thức `Str::reverse()` sẽ đảo ngược chuỗi đã cho:

```php
use Illuminate\Support\Str;

$reversed = Str::reverse('Hello World');

// dlroW olleH
```

Str::singular()

Phương thứcStr::singular() sẽ chuyển đổi một chuỗi câu thành dạng từ ngữ số ít của nó. Chức năng này dành cho bất kỳ ngôn ngữ nào được hỗ trợ bởi chương trình đa số hóa cho một danh từ của Laravel:

```php
use Illuminate\Support\Str;

$singular = Str::singular('cars');

// car

$singular = Str::singular('children');

// child
```

Str::slug()

Phương thức Str::slug() tạo ra một "slug" thân thiện với URL từ chuỗi câu đã cho:

```php
use Illuminate\Support\Str;

$slug = Str::slug('Laravel 5 Framework', '-');

// laravel-5-framework
```

Str::snake()

Phương thức Str::snake() sẽ chuyển đổi chuỗi câu đã cho thành solid_case:

```php
use Illuminate\Support\Str;

$converted = Str::snake('fooBar');

// foo_bar

$converted = Str::snake('fooBar', '-');

// foo-bar
```

Str::squish()

Phương thức `Str::squish()` sẽ loại bỏ tất cả khoảng trắng không liên quan khỏi một chuỗi câu, bao gồm cả khoảng trắng không liên quan giữa các từ:

```php
use Illuminate\Support\Str;

$string = Str::squish('    laravel    framework    ');

// laravel framework
```

Str::start()

Phương thức `Str::start()` sẽ thêm một trường hợp duy nhất của giá trị đã cho vào một chuỗi câu nếu nó chưa bắt đầu bằng giá trị đó:

```php
use Illuminate\Support\Str;

$adjusted = Str::start('this/string', '/');

// /this/string

$adjusted = Str::start('/this/string', '/');

// /this/string
```

Str::startsWith()

Phương thức `Str::startsWith()` sẽ xác định xem chuỗi câu đã cho có bắt đầu bằng giá trị đã cho hay không:

```
use Illuminate\Support\Str;

$result = Str::startsWith('This is my name', 'This');

// true
```

Nếu một mảng các giá trị được truyền vào ở đối số thứ hai, thì phương thức `startWith` sẽ trả về `true` nếu chuỗi bắt đầu bằng bất kỳ giá trị nào trong số các giá trị đã cho:

```
$result = Str::startsWith('This is my name', ['This', 'That', 'There']);

// true
```

Str::studly()

Phương thức `Str::studly()` sẽ chuyển đổi chuỗi câu đã cho thành `StudlyCase`:

```
use Illuminate\Support\Str;

$converted = Str::studly('foo_bar');

// FooBar
```

Str::substr()

Phương thức `Str::substr()` sẽ trả về phần chuỗi câu được chỉ định bởi các tham số bắt đầu và độ dài:

```
use Illuminate\Support\Str;

$converted = Str::substr('The Laravel Framework', 4, 7);
```

```
// Laravel
```

Str::substrCount()

Phương thức `Str::substrCount()` sẽ trả về số lần xuất hiện của một giá trị nhất định trong chuỗi đã cho:

```
use Illuminate\Support\Str;

$count = Str::substrCount('If you like ice cream, you will like snow cones.', 'like');

// 2
```

Str::substrReplace()

Phương thức `Str::substrReplace()` sẽ thay thế văn bản trong một phần của chuỗi câu, bắt đầu từ vị trí được chỉ định bởi đối số thứ ba và thay thế số ký tự được chỉ định bởi đối số thứ tư. Truyền 0 vào đối số thứ tư của phương thức sẽ chèn chuỗi vào vị trí được chỉ định mà không thay thế bất kỳ ký tự hiện có nào trong chuỗi câu:

```
use Illuminate\Support\Str;

$result = Str::substrReplace('1300', ':', 2);
// 13:

$result = Str::substrReplace('1300', ':', 2, 0);
// 13:00
```

Str::swap()

Phương thức `Str::swap()` sẽ thay thế nhiều giá trị trong chuỗi câu đã cho bằng cách sử dụng hàm `strtr` của PHP thuần:

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

```php
$string = Str::swap([
  'Tacos' => 'Burritos',
  'great' => 'fantastic',
], 'Tacos are great!');

// Burritos are fantastic!
```

Str::title()

Phương thức `Str::title()` sẽ chuyển đổi chuỗi câu đã cho thành `Title Case`:

```php
use Illuminate\Support\Str;

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

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

Str::toHtmlString()

Phương thức `Str::toHtmlString()` sẽ chuyển đổi thể hiện chuỗi câu thành một thể hiện của `Illuminate\Support\HtmlString`, có thể được hiển thị trong các mẫu Blade:

```php
use Illuminate\Support\Str;

$htmlString = Str::of('Nuno Maduro')->toHtmlString();
```

Str::ucfirst()

Phương thức `Str::ucfirst()` sẽ trả về chuỗi câu đã cho với ký tự đầu tiên được viết hoa:

```php
use Illuminate\Support\Str;

$string = Str::ucfirst('foo bar');
```

```
// Foo bar
```

Str::ucsplit()

Phương thức `Str::ucsplit()` chia chuỗi câu đã cho thành một mảng bằng các ký tự viết hoa:

```
use Illuminate\Support\Str;

$segments = Str::ucsplit('FooBar');

// [0 => 'Foo', 1 => 'Bar']
```

Str::upper()

Phương thức `Str::upper()` sẽ chuyển đổi chuỗi câu đã cho thành chữ hoa:

```
use Illuminate\Support\Str;

$string = Str::upper('laravel');

// LARAVEL
```

Str::uuid()

Phương thức `Str::uuid()` sẽ tạo ra một UUID (phiên bản 4):

```
use Illuminate\Support\Str;

return (string) Str::uuid();
```

Str::wordCount()

Phương thức `Str::wordCount()` sẽ trả về số từ mà một chuỗi chứa:

```php
use Illuminate\Support\Str;

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

Str::words()

Phương thức `Str::words()` sẽ giới hạn số lượng từ trong một chuỗi. Một chuỗi bổ sung có thể được chuyển đến phương thức này thông qua đối số thứ ba của nó để chỉ định chuỗi nào sẽ được nối vào cuối chuỗi bị cắt ngắn:

```php
use Illuminate\Support\Str;

return Str::words('Perfectly balanced, as all things should be.', 3, ' >>>');

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

str()

Hàm str sẽ trả về một đối tượng `Illuminate\Support\Stringable` mới của chuỗi đã cho. Hàm này tương đương với phương thức `Str::of`:

```php
$string = str('Taylor')->append(' Otwell');

// 'Taylor Otwell'
```

Nếu không có đối số nào được cung cấp cho hàm `str`, hàm sẽ trả về một thể hiện của `Illuminate\Support\Str`:

```php
$snake = str()->snake('FooBar');

// 'foo_bar'
```

trans()

Hàm `trans` sẽ dịch khóa thông dịch nhất định bằng cách sử dụng các tập tin việt hóa của bạn:

```
echo trans('messages.welcome');
```

Nếu khóa thông dịch được chỉ định không tồn tại, hàm `trans` sẽ trả về nguyên văn chuỗi khóa đã khai báo. Vì vậy, khi sử dụng ví dụ trên, hàm `trans` sẽ trả về `messages.welcome` nếu khóa thông dịch không tồn tại.

`trans_choice()`

Hàm `trans_choice` dịch khóa thông dịch đã cho với một biến động:

```
echo trans_choice('messages.notifications', $unreadCount);
```

Nếu khóa thông dịch được chỉ định không tồn tại, hàm `trans_choice` sẽ trả về khóa đã cho. Vì vậy, bằng cách sử dụng ví dụ trên, hàm `trans_choice` sẽ trả về `messages.notifications` nếu khóa thông dịch không tồn tại.

Xử lý mạch lạc chuỗi câu

Chuỗi câu có cung cấp giao diện hướng đối tượng, cho phép xử lý mạch lạc hơn khi làm việc với các giá trị chuỗi, nó cho phép bạn liên kết nhiều hoạt động trên chuỗi câu với nhau bằng cách sử dụng cú pháp dễ đọc hơn so với các hoạt động chuỗi câu truyền thống.

`after`

Phương thức `after` sẽ trả về mọi thứ đứng phía sau giá trị đã cho trong một chuỗi câu. Toàn bộ chuỗi câu sẽ được trả về nếu giá trị không tồn tại trong chuỗi câu:

```
use Illuminate\Support\Str;

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

// ' my name'
```

afterLast

Phương thức `afterLast` trả về mọi thứ sau lần xuất hiện cuối cùng của giá trị đã cho trong một chuỗi. Toàn bộ chuỗi sẽ được trả về nếu giá trị không tồn tại trong chuỗi:

```
use Illuminate\Support\Str;

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

// 'Controller'
```

append

Phương thức `append` sẽ nối các giá trị đã cho vào chuỗi câu:

```
use Illuminate\Support\Str;

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

// 'Taylor Otwell'
```

ascii

Phương thức `ascii` sẽ cố gắng truyền chuỗi câu thành giá trị ASCII:

```
use Illuminate\Support\Str;

$string = Str::of('ü')->ascii();

// 'u'
```

basename

Phương thức `basename` sẽ trả về thành phần tên theo sau của chuỗi câu đã cho:

```php
use Illuminate\Support\Str;

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

// 'baz'
```

Nếu cần, bạn có thể cung cấp một "tiện ích mở rộng" sẽ bị xóa khỏi thành phần đang theo sau:

```php
use Illuminate\Support\Str;

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

// 'baz'
```

before

Phương thức `before` sẽ trả về mọi thứ trước giá trị đã cho trong một chuỗi câu:

```php
use Illuminate\Support\Str;

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

// 'This is '
```

beforeLast

Phương thức `beforeLast` sẽ trả về mọi thứ trước lần xuất hiện cuối cùng của giá trị đã cho trong một chuỗi câu:

```php
use Illuminate\Support\Str;

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

// 'This '
```

between

Phương thức `between` trả về một phần của chuỗi câu giữa hai giá trị:

```php
use Illuminate\Support\Str;

$converted = Str::of('This is my name')->between('This', 'name');

// ' is my '
```

betweenFirst

Phương thức `betweenFirst` sẽ trả về phần nhỏ nhất có thể có của một chuỗi câu giữa hai giá trị:

```php
use Illuminate\Support\Str;

$converted = Str::of('[a] bc [d]')->betweenFirst('[', ']');

// 'a'
```

camel

Phương thức `camel` sẽ chuyển đổi chuỗi câu đã cho thành `camelCase`:

```php
use Illuminate\Support\Str;

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

```
// fooBar
```

classBasename

Phương thức `classBasename` sẽ trả về tên class của class đã cho với namespace của class đó bị xóa:

```
use Illuminate\Support\Str;

$class = Str::of('Foo\Bar\Baz')->classBasename();

// Baz
```

contains

Phương thức `contains` xác định xem chuỗi câu đã cho có chứa giá trị đã cho hay không. Phương thức này phân biệt chữ in hoa và chữ thường:

```
use Illuminate\Support\Str;

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

// true
```

Bạn cũng có thể truyền một mảng giá trị để xác định xem chuỗi câu đã cho có chứa bất kỳ giá trị nào trong mảng hay không:

```
use Illuminate\Support\Str;

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

// true
```

containsAll

Phương thức `containsAll` sẽ xác định xem chuỗi câu đã cho có chứa tất cả các giá trị trong mảng đã cho hay không:

```
use Illuminate\Support\Str;

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

// true
```

dirname

Phương thức `dirname` sẽ trả về phần thư mục mẹ của chuỗi câu đã cho:

```
use Illuminate\Support\Str;

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

// '/foo/bar'
```

Nếu cần, bạn có thể chỉ định số lượng cấp thư mục bạn muốn cắt khỏi chuỗi câu:

```
use Illuminate\Support\Str;

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

// '/foo'
```

excerpt

Phương thức `excerpt` trích một đoạn trích từ chuỗi khớp với đối tượng đầu tiên của một cụm từ trong chuỗi câu đó:

```
use Illuminate\Support\Str;

$excerpt = Str::of('This is my name')->excerpt('my', [
```

```
  'radius' => 3
]);

// '...is my na...'
```

Tùy chọn `radius`, mặc định là `100`, cho phép bạn xác định số ký tự sẽ xuất hiện trên mỗi bên của chuỗi câu bị cắt ngắn. Ngoài ra, bạn có thể sử dụng tùy chọn `omission` để thay đổi chuỗi câu sẽ được thêm vào trước và nối vào chuỗi bị cắt ngắn:

```
use Illuminate\Support\Str;

$excerpt = Str::of('This is my name')->excerpt('name', [
  'radius' => 3,
  'omission' => '(...) '
]);

// '(...) my name'
```

endsWith

Phương thức `endWith` sẽ xác định xem chuỗi câu đã cho có kết thúc bằng giá trị đã cho hay không:

```
use Illuminate\Support\Str;

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

// true
```

Bạn cũng có thể truyền một mảng giá trị để xác định xem chuỗi câu đã cho có kết thúc bằng bất kỳ giá trị nào trong mảng hay không:

```
use Illuminate\Support\Str;

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

```php
// true

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

// false
```

exactly

Phương thức `exactly` sẽ xác định xem chuỗi câu đã cho có phải là một kết hợp chính xác với một chuỗi câu khác hay không:

```php
use Illuminate\Support\Str;

$result = Str::of('Laravel')->exactly('Laravel');

// true
```

explode

Phương thức `explode` chia chuỗi câu theo dấu phân cách đã cho và trả về một tập hợp chứa từng phần của chuỗi câu phân tách:

```php
use Illuminate\Support\Str;

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

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

finish

Phương thức `finish` sẽ thêm một đối tượng đơn lẻ của giá trị đã cho vào một chuỗi câu nếu nó chưa kết thúc bằng giá trị đó:

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

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

// this/string/

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

// this/string/
```

inlineMarkdown

Phương thức `inlineMarkdown` sẽ chuyển đổi Markdown theo phong cách GitHub thành HTML inline bằng cách sử dụng CommonMark. Tuy nhiên, không giống như phương thức markdown, nó không bao bọc tất cả HTML được tạo trong một phần tử block-level:

```php
use Illuminate\Support\Str;

$html = Str::of('**Laravel**')->inlineMarkdown();

// <strong>Laravel</strong>
```

is

Phương thức `is` xác định xem một chuỗi câu nào đó có khớp với một mẫu cụ thể hay không. Dấu hoa thị có thể được sử dụng làm giá trị ký tự đại diện.

```php
use Illuminate\Support\Str;

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

// true

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

// false
```

isAscii

Phương thức `isAscii` sẽ xác định xem một chuỗi câu đã cho có phải là chuỗi câu ASCII hay không:

```
use Illuminate\Support\Str;

$result = Str::of('Taylor')->isAscii();

// true

$result = Str::of('ü')->isAscii();

// false
```

isEmpty

Phương thức `isEmpty` sẽ xác định xem chuỗi câu đã cho có trống không:

```
use Illuminate\Support\Str;

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

// true

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

// false
```

isNotEmpty

Phương thức `isNotEmpty` sẽ xác định xem chuỗi câu nào đó không trống hay ngược lại:

```
use Illuminate\Support\Str;

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

```php
// false

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

// true
```

isJson

Phương thức `isJson` sẽ xác định xem một chuỗi câu nào đó có phải là JSON hợp lệ hay không:

```php
use Illuminate\Support\Str;

$result = Str::of('[1,2,3]')->isJson();

// true

$result = Str::of('{"first": "John", "last": "Doe"}')->isJson();

// true

$result = Str::of('{first: "John", last: "Doe"}')->isJson();

// false
```

isUuid

Phương thức `isUuid` sẽ xác định xem một chuỗi câu đã cho có phải là UUID hay không:

```php
use Illuminate\Support\Str;

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

// true
```

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

// false
```

kebab

Phương thức `kebab` sẽ chuyển đổi chuỗi câu đã cho thành `kebab-case`:

```php
use Illuminate\Support\Str;

$converted = Str::of('fooBar')->kebab();

// foo-bar
```

lcfirst

Phương thức `lcfirst` sẽ trả về chuỗi câu đã cho với ký tự đầu tiên được viết thường:

```php
use Illuminate\Support\Str;

$string = Str::of('Foo Bar')->lcfirst();

// foo Bar
```

length

Phương thức `length` sẽ trả về độ dài của chuỗi câu đã cho:

```php
use Illuminate\Support\Str;

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

// 7
```

limit

Phương thức `limit` sẽ cắt ngắn chuỗi câu đã cho theo độ dài được chỉ định:

```
use Illuminate\Support\Str;

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

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

Bạn cũng có thể truyền đối số thứ hai một chuỗi câu mà sẽ được nối vào cuối chuỗi câu bị cắt ngắn trước đó:

```
use Illuminate\Support\Str;

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

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

lower

Phương thức `lower` chuyển đổi chuỗi câu đã cho thành chữ thường:

```
use Illuminate\Support\Str;

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

// 'laravel'
```

ltrim

Phương thức `ltrim` sẽ cắt bỏ phần bên trái của chuỗi câu:

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

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

// 'Laravel  '

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

// 'Laravel/'
```

markdown

Phương thức `markdown` chuyển đổi Markdown theo phong cách GitHub thành HTML:

```php
use Illuminate\Support\Str;

$html = Str::of('# Laravel')->markdown();

// <h1>Laravel</h1>

$html = Str::of('# Taylor <b>Otwell</b>')->markdown([
    'html_input' => 'strip',
]);

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

mask

Phương thức `mask` che lấp một phần của chuỗi câu bằng các ký tự lặp lại và có thể được sử dụng để làm xáo trộn các phân đoạn của chuỗi câu như địa chỉ email và số điện thoại vốn là những tài liệu nhạy cảm:

```php
use Illuminate\Support\Str;

$string = Str::of('taylor@example.com')->mask('*', 3);

// tay***************
```

Nếu cần, bạn cung cấp một số âm làm đối số thứ ba cho phương thức `mask`, điều này sẽ khiến cho phương thức bắt đầu tạo mặt nạ ở khoảng cách nào đó từ cuối chuỗi câu:

```
$string = Str::of('taylor@example.com')->mask('*', -15, 3);

// tay***@example.com
```

match

Phương thức `match` sẽ trả về một phần của chuỗi câu nào khớp với biểu thức mẫu đã quy định:

```
use Illuminate\Support\Str;

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

// 'bar'

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

// 'bar'
```

matchAll

Phương thức `matchAll` sẽ trả về một tập hợp chứa những phần của chuỗi câu nào khớp với biểu thức mẫu đã quy định:

```
use Illuminate\Support\Str;

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

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

Nếu bạn chỉ định một biểu thức so sánh nhóm trong biểu thức mẫu, thì Laravel sẽ trả về một tập hợp các kết quả phù hợp của nhóm đó, ví dụ:

```php
use Illuminate\Support\Str;

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

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

Nếu không tìm thấy kết quả phù hợp, thì khi đó một bộ sưu tập trống sẽ được trả lại.

newLine

Phương thức `newLine` sẽ nối một ký tự "cuối dòng" vào một chuỗi câu:

```php
use Illuminate\Support\Str;

$padded = Str::of('Laravel')->newLine()->append('Framework');

// 'Laravel
//  Framework'
```

padBoth

Phương thức `padBoth` bao bọc hàm `str_pad` của PHP thuần, đệm cả hai bên của một chuỗi câu bằng một chuỗi ký tự khác cho đến khi chuỗi kết quả cuối cùng đạt đến độ dài mong muốn:

```php
use Illuminate\Support\Str;

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

// '__James___'

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

// '  James   '
```

padLeft

Phương thức `padLeft` bao bọc hàm `str_pad` của PHP thuần, đệm bên trái của một chuỗi câu bằng một chuỗi ký tự khác cho đến khi chuỗi câu kết quả cuối cùng đạt đến độ dài mong muốn:

```
use Illuminate\Support\Str;

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

// '-=-=-James'

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

// '     James'
```

padRight

Phương thức `padRight` sẽ bao bọc hàm `str_pad` của PHP thuần, đệm thêm bên phải của một chuỗi câu bằng một chuỗi ký tự khác cho đến khi chuỗi câu kết quả cuối cùng đạt đến độ dài mong muốn:

```
use Illuminate\Support\Str;

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

// 'James-----'

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

// 'James     '
```

pipe

Phương thức `pipe` cho phép bạn biến đổi chuỗi câu bằng cách truyền giá trị hiện tại của nó vào hàm xử lý đã cho:

```php
use Illuminate\Support\Str;

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

// 'Checksum: a5c95b86291ea299fcbe64458ed12702'

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

// 'bar'
```

plural

Phương thức `plural` sẽ chuyển đổi một chuỗi ký tự từ số ít sang dạng số nhiều của nó. Chức năng này hỗ trợ bất kỳ ngôn ngữ nào được hỗ trợ bởi chương trình đa số hóa cho một danh từ của Laravel:

```php
use Illuminate\Support\Str;

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

// cars

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

// children
```

Bạn có thể cung cấp một số nguyên làm đối số thứ hai cho hàm `plural` để truy xuất cụ thể dạng số ít hoặc số nhiều của chuỗi câu:

```php
use Illuminate\Support\Str;

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

// children
```

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

// child
```

prepend

Phương thức `prepend` sẽ thêm các giá trị đã truyền vào đằng trước chuỗi câu:

```php
use Illuminate\Support\Str;

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

// Laravel Framework
```

remove

Phương thức `remove` sẽ loại bỏ giá trị đã cho hoặc mảng giá trị muốn loại bỏ khỏi chuỗi câu:

```php
use Illuminate\Support\Str;

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

// Arkansas is beautiful!
```

Bạn cũng có thể chuyển `false` làm đối số thứ hai để bỏ qua sự so sánh chữ hoa-thường khi xóa chuỗi câu.

replace

Phương thức `replace` sẽ thay thế một chuỗi câu đã cho trong chuỗi câu:

```php
use Illuminate\Support\Str;

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

```
// Laravel 7.x
```

replaceArray

Phương thức `replaceArray` sẽ thay thế một giá trị đã cho trong chuỗi một cách tuần tự bằng cách sử dụng một mảng:

```
use Illuminate\Support\Str;

$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

Phương thức `replaceFirst` sẽ thay thế lần phát hiện đầu tiên của một giá trị đã cho trong một chuỗi câu:

```
use Illuminate\Support\Str;

$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

Phương thức `replaceLast` sẽ thay thế lần phát hiện cuối cùng của một giá trị đã cho trong một chuỗi câu:

```
use Illuminate\Support\Str;

$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

Phương thức `replaceMatches` sẽ thay thế tất cả các phần của một chuỗi câu nào khớp với một biểu thức mẫu bằng chuỗi thay thế cụ thể đã cho:

```
use Illuminate\Support\Str;

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

// '15015551000'
```

Phương thức `replaceMatches` cũng chấp nhận một hàm xử lý sẽ được gọi với mỗi phần chuỗi mà khớp với biểu thức mẫu đã cho, cho phép bạn thực hiện logic thay thế trong hàm xử lý và sẽ trả về giá trị được thay thế:

```
use Illuminate\Support\Str;

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

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

rtrim

Phương thức `rtrim` sẽ cắt bỏ phần bên phải của chuỗi câu đã cho:

```
use Illuminate\Support\Str;

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

// '  Laravel'
```

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

// '/Laravel'
```

scan

Phương thức `scan` sẽ phân tích cú pháp đầu vào từ một chuỗi câu thành một tập hợp theo định dạng được hỗ trợ bởi hàm `sscanf` PHP:

```php
use Illuminate\Support\Str;

$collection = Str::of('filename.jpg')->scan('%[^.].%s');

// collect(['filename', 'jpg'])
```

singular

Phương thức `singular` sẽ chuyển đổi một chuỗi câu thành dạng số ít của nó. Chức năng này sẽ hỗ trợ bất kỳ ngôn ngữ nào được hỗ trợ bởi chương trình đa số hóa với danh từ của Laravel:

```php
use Illuminate\Support\Str;

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

// car

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

// child
```

slug

Phương thức `slug` tạo ra một "slug" thân thiện với URL từ chuỗi câu đã cho:

```php
use Illuminate\Support\Str;

$slug = Str::of('Laravel Framework')->slug('-');

// laravel-framework
```

snake

Phương thức `snake` sẽ chuyển đổi chuỗi câu đã cho thành `solid_case`:

```php
use Illuminate\Support\Str;

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

// foo_bar
```

split

Phương thức `split` chia một chuỗi câu thành một tập hợp bằng cách sử dụng một biểu thức mẫu:

```php
use Illuminate\Support\Str;

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

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

squish

Phương thức `squish` sẽ loại bỏ tất cả khoảng trắng không liên quan khỏi một chuỗi, bao gồm cả khoảng trắng không liên quan giữa các từ:

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

```php
$string = Str::of('    laravel    framework    ')->squish();

// laravel framework
```

start

Phương thức `start` sẽ thêm ký tự đã cho vào phần đầu của một chuỗi câu nếu chuỗi câu đó chưa có phần đầu bằng ký tự đó:

```php
use Illuminate\Support\Str;

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

// /this/string

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

// /this/string
```

startsWith

Phương thức `startedWith` sẽ xác định xem chuỗi câu đã cho có bắt đầu bằng giá trị đã cho hay không:

```php
use Illuminate\Support\Str;

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

// true
```

studly

Phương thức `studly` sẽ chuyển đổi chuỗi câu đã cho thành `StudlyCase:`

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

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

// FooBar
```

substr

Phương thức `substr` sẽ trả về một phần của chuỗi câu được chỉ định bởi hai đối số đại diện cho vị trí bắt đầu và độ dài của chuỗi câu kết quả:

```php
use Illuminate\Support\Str;

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

// Framework

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

// Frame
```

substrReplace

Phương thức `substrReplace` sẽ thay thế văn bản trong một phần của chuỗi, bắt đầu từ vị trí được chỉ định bởi đối số thứ hai và thay thế với số lượng ký tự được chỉ định bởi đối số thứ ba. Nếu truyền 0 vào đối số thứ ba của phương thức sẽ chèn chuỗi câu vào vị trí được chỉ định thay vì thay thế bất kỳ ký tự hiện có nào trong chuỗi:

```php
use Illuminate\Support\Str;

$string = Str::of('1300')->substrReplace(':', 2);

// 13:

$string = Str::of('The Framework')->substrReplace(' Laravel', 3, 0);

// The Laravel Framework
```

swap

Phương thức `swap` sẽ thay thế nhiều giá trị trong chuỗi bằng cách sử dụng hàm `strtr` của PHP thuần:

```php
use Illuminate\Support\Str;

$string = Str::of('Tacos are great!')
  ->swap([
    'Tacos' => 'Burritos',
    'great' => 'fantastic',
  ]);

// Burritos are fantastic!
```

tap

Phương thức `tap` sẽ truyền chuỗi đến hàm xử lý đã cho, cho phép bạn kiểm tra và tương tác với chuỗi trong khi không ảnh hưởng đến chính chuỗi. Chuỗi đầu vào sẽ được trả về bởi phương thức `tap` bất kể cái gì được trả về bởi hàm xử lý đã cho:

```php
use Illuminate\Support\Str;

$string = Str::of('Laravel')
  ->append(' Framework')
```

```
    ->tap(function ($string) {
        dump('String after append: '.$string);
    })
    ->upper();

// LARAVEL FRAMEWORK
```

test

Phương thức `test` sẽ xác định xem một chuỗi câu có khớp với biểu thức mẫu chính quy đã cho hay không:

```
use Illuminate\Support\Str;

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

// true
```

title

Phương thức `title` sẽ chuyển đổi chuỗi câu đã cho thành `TitleCase`:

```
use Illuminate\Support\Str;

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

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

trim

Phương thức `trim` sẽ cắt chuỗi câu đã cho:

```
use Illuminate\Support\Str;

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

```php
// 'Laravel'

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

// 'Laravel'
```

ucfirst

Phương thức `ucfirst` trả về chuỗi đã cho với ký tự đầu tiên được viết hoa:

```php
use Illuminate\Support\Str;

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

// Foo bar
```

ucsplit

Phương thức `ucsplit` sẽ chia chuỗi câu đã cho thành một tập hợp bằng các ký tự viết hoa:

```php
use Illuminate\Support\Str;

$string = Str::of('Foo Bar')->ucsplit();

// collect(['Foo', 'Bar'])
```

upper

Phương thức `upper` sẽ chuyển đổi chuỗi câu đã cho thành chữ hoa:

```php
use Illuminate\Support\Str;

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

```
// LARAVEL
```

when

Phương thức `when` sẽ gọi một hàm xử lý đã cho nếu một điều kiện đã cho nào đó là đúng. Hàm này sẽ nhận được đối tượng chuỗi câu:

```
use Illuminate\Support\Str;

$string = Str::of('Taylor')
  ->when(true, function ($string) {
    return $string->append(' Otwell');
  });

// 'Taylor Otwell'
```

Nếu cần, bạn có thể chuyển một hàm xử lý khác làm đối số thứ ba cho phương thức when. Hàm này sẽ thực thi nếu đối số điều kiện đánh giá là `false`.

whenContains

Phương thức `whenContains` sẽ gọi một hàm xử lý đã cho nếu chuỗi câu chứa giá trị đã cho. Hàm này sẽ nhận được đối tượng chuỗi câu:

```
use Illuminate\Support\Str;

$string = Str::of('tony stark')
  ->whenContains('tony', function ($string) {
    return $string->title();
  });

// 'Tony Stark'
```

Nếu cần, bạn có thể truyền thêm một hàm xử lý khác làm đối số thứ ba cho phương thức `when`. Hàm này sẽ thực thi khi chuỗi câu không chứa giá trị đã cho.

Bạn cũng có thể chuyển một mảng giá trị để xác định xem chuỗi câu đã cho có chứa bất kỳ

giá trị nào trong mảng hay không:

```
use Illuminate\Support\Str;

$string = Str::of('tony stark')
  ->whenContains(['tony', 'hulk'], function ($string) {
    return $string->title();
  });

// Tony Stark
```

whenContainsAll

Phương thức `whenContainsAll` sẽ gọi hàm xử lý đã cho nếu chuỗi chứa tất cả các chuỗi con đã cho. Hàm này sẽ nhận được đối tượng chuỗi câu:

```
use Illuminate\Support\Str;

$string = Str::of('tony stark')
  ->whenContainsAll(['tony', 'stark'], function ($string) {
    return $string->title();
  });

// 'Tony Stark'
```

Nếu cần, bạn có thể truyền thêm một hàm xử lý khác làm đối số thứ ba cho phương thức `when`. Hàm này sẽ thực thi nếu đối số điều kiện đánh giá là `false`.

whenEmpty

Phương thức `whenEmpty` sẽ gọi một hàm xử lý đã cho nếu chuỗi câu trống. Nếu hàm này trả về một giá trị, thì giá trị đó cũng sẽ được trả về bởi phương thức `whenEmpty`. Nếu hàm này không trả về giá trị, thì đối tượng chuỗi câu sẽ được trả về:

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

```php
$string = Str::of('  ')->whenEmpty(function ($string) {
  return $string->trim()->prepend('Laravel');
});

// 'Laravel'
```

whenNotEmpty

Phương thức `whenNotEmpty` sẽ gọi hàm đã cho nếu chuỗi câu không trống. Nếu hàm này trả về một giá trị, thì giá trị đó cũng sẽ được trả về bởi phương thức `whenNotEmpty`. Nếu hàm này không trả về giá trị, thì đối tượng chuỗi câu sẽ được trả về:

```php
use Illuminate\Support\Str;

$string = Str::of('Framework')->whenNotEmpty(function ($string) {
  return $string->prepend('Laravel ');
});

// 'Laravel Framework'
```

whenStartsWith

Phương thức `whenStartsWith` sẽ gọi hàm xử lý đã cho nếu chuỗi câu bắt đầu bằng chuỗi con đã cho. Hàm này sẽ nhận được đối tượng chuỗi câu:

```php
use Illuminate\Support\Str;

$string = Str::of('disney world')->whenStartsWith('disney', function ($string) {
  return $string->title();
});

// 'Disney World'
```

whenEndsWith

Phương thức `whenEndsWith` gọi hàm xử lý đã cho nếu chuỗi câu kết thúc bằng chuỗi con

đã cho. Hàm này sẽ nhận được đối tượng chuỗi câu:

```
use Illuminate\Support\Str;

$string = Str::of('disney world')->whenEndsWith('world', function ($string) {
  return $string->title();
});

// 'Disney World'
```

whenExactly

Phương thức `whenExactly` sẽ gọi hàm xử lý đã cho nếu chuỗi câu khớp hoàn toàn với chuỗi ký tự đã cho. Hàm này sẽ nhận được đối tượng chuỗi câu:

```
use Illuminate\Support\Str;

$string = Str::of('laravel')->whenExactly('laravel', function ($string) {
  return $string->title();
});

// 'Laravel'
```

whenNotExactly

Phương thức `whenNotExactly` sẽ gọi hàm xử lý đã cho nếu chuỗi câu không khớp hoàn toàn với chuỗi đã cho. Hàm này sẽ nhận được đối tượng chuỗi câu:

```
use Illuminate\Support\Str;

$string = Str::of('framework')->whenNotExactly('laravel', function ($string) {
  return $string->title();
});

// 'Framework'
```

whenIs

Phương thức `whenIs` sẽ gọi hàm xử lý đã cho nếu chuỗi câu khớp với một biểu thức mẫu đã cho. Dấu hoa thị có thể được sử dụng để làm giá trị ký tự đại diện (wildcard). Hàm này sẽ nhận được đối tượng chuỗi câu:

```
use Illuminate\Support\Str;

$string = Str::of('foo/bar')->whenIs('foo/*', function ($string) {
  return $string->append('/baz');
});

// 'foo/bar/baz'
```

whenIsAscii

Phương thức `whenIsAscii` sẽ gọi hàm xử lý đã cho nếu chuỗi câu dạng 7 bit ASCII. Hàm này sẽ nhận được đối tượng chuỗi câu:

```
use Illuminate\Support\Str;

$string = Str::of('foo/bar')->whenIsAscii('laravel', function ($string) {
  return $string->title();
});

// 'Laravel'
```

whenIsUuid

Phương thức `whenIsUuid` sẽ gọi một hàm xử lý đã cho nếu chuỗi câu là một UUID hợp lệ. Hàm này sẽ nhận được đối tượng chuỗi câu:

```
use Illuminate\Support\Str;

$string = Str::of('foo/bar')->whenIsUuid('a0a2a2d2-0b87-4a18-83f2-2529882be2de', function ($strir
  return $string->substr(0, 8);
```

```
});

// 'a0a2a2d2'
```

whenTest

Phương thức `whenTest` sẽ gọi hàm xử lý đã cho nếu chuỗi câu khớp với biểu thức mẫu đã cho. Hàm này sẽ nhận được đối tượng chuỗi câu:

```
use Illuminate\Support\Str;

$string = Str::of('laravel framework')->whenTest('/laravel/', function ($string) {
  return $string->title();
});

// 'Laravel Framework'
```

wordCount

Phương thức `wordCount` sẽ trả về số từ mà một chuỗi câu chứa:

```
use Illuminate\Support\Str;

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

words

Phương thức `words` giới hạn số lượng từ trong một chuỗi câu. Nếu cần, bạn có thể chỉ định một chuỗi bổ sung sẽ được thay thế vào chuỗi đã bị cắt đi:

```php
use Illuminate\Support\Str;

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

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

URLs

action()

Hàm `action()` sẽ tạo một URL cho một action trong controller đã cho:

```php
use App\Http\Controllers\HomeController;

$url = action([HomeController::class, 'index']);
```

Nếu route có các tham số truy vấn, thì bạn có thể truyền chúng làm đối số thứ hai cho phương thức:

```php
$url = action([UserController::class, 'profile'], ['id' => 1]);
```

asset()

Hàm `asset()` sẽ tạo URL cho asset (asset là các tập tin được nhúng vào trang web như; ảnh, phim, text, css, vv...) bằng cách sử dụng scheme hiện tại của yêu cầu (HTTP hoặc HTTPS):

```php
$url = asset('img/photo.jpg');
```

Bạn có thể cấu hình máy chủ lưu trữ asset URL bằng cách đặt biến `ASSET_URL` trong tập tin `.env` của mình. Điều này có thể hữu ích nếu bạn lưu trữ asset của mình trên một dịch vụ bên ngoài như *Amazon S3* hoặc *CDN* khác:

```php
// ASSET_URL=http://example.com/assets
```

```php
$url = asset('img/photo.jpg'); // http://example.com/assets/img/photo.jpg
```

route()

Hàm `route()` sẽ tạo ra một URL cho một route đã được đặt tên cụ thể:

```php
$url = route('route.name');
```

Nếu route có chứa các tham số truy vấn, bạn có thể truyền chúng làm đối số thứ hai cho hàm:

```php
$url = route('route.name', ['id' => 1]);
```

Theo mặc định, hàm `route` sẽ tạo một URL tuyệt đối. Nếu bạn muốn tạo một URL tương đối, bạn có thể chuyển `false` làm đối số thứ ba cho hàm:

```php
$url = route('route.name', ['id' => 1], false);
```

secure_asset()

Hàm `secure_asset()` sẽ tạo một URL cho một asset bằng HTTPS:

```php
$url = secure_asset('img/photo.jpg');
```

secure_url()

Hàm `secure_url()` sẽ tạo một URL HTTPS đầy đủ cho đường dẫn đã cho. Các phân đoạn URL bổ sung có thể được truyền vào đối số thứ hai của hàm:

```php
$url = secure_url('user/profile');

$url = secure_url('user/profile', [1]);
```

`to_route()`

Hàm `to_route` sẽ tạo phản hồi HTTP chuyển hướng cho một route được đặt tên nhất định:

```
return to_route('users.show', ['user' => 1]);
```

Nếu cần, bạn có thể truyền mã trạng thái HTTP sẽ được gán cho việc chuyển hướng và bất kỳ tiêu đề phản hồi bổ sung nào làm đối số thứ ba và thứ tư cho phương thức `to_route`:

```
return to_route('users.show', ['user' => 1], 302, ['X-Framework' => 'Laravel']);
```

`url()`

Hàm `url` sẽ tạo ra một URL đầy đủ cho đường dẫn đã cho:

```
$url = url('user/profile');

$url = url('user/profile', [1]);
```

Nếu không có đường dẫn nào được cung cấp, một đối tượng `Illuminate\Routing\UrlGenerator` sẽ được trả về:

```
$current = url()->current();

$full = url()->full();

$previous = url()->previous();
```

Các tiện ích khác

`abort()`

Hàm `abort()` được dùng để đưa ra một HTTP Exception sẽ được chương trình xử lý exception hiển thị:

```
abort(403);
```

Bạn cũng có thể cung cấp thông báo của exception và tiêu đề phản hồi HTTP tự chọn mà sẽ được gửi đến trình duyệt:

```
abort(403, 'Unauthorized.', $headers);
```

abort_if()

Hàm `abort_if()` giúp đưa ra một HTTP exception nếu một biểu thức boolean đã cho được đánh giá là `true`:

```
abort_if(! Auth::user()->isAdmin(), 403);
```

Giống như phương thức `abort`, bạn cũng có thể cung cấp văn bản phản hồi của exception làm đối số thứ ba và một mảng tiêu đề phản hồi tự chọn làm đối số thứ tư cho hàm.

abort_unless()

Hàm `abort_unless` sẽ đưa ra một HTTP exception nếu một biểu thức boolean đã cho được đánh giá là `false`:

```
abort_unless(Auth::user()->isAdmin(), 403);
```

Giống như phương thức `abort,` bạn cũng có thể cung cấp văn bản phản hồi của exception làm đối số thứ ba và một mảng tiêu đề phản hồi tự chọn làm đối số thứ tư cho hàm.

app()

Hàm `app()` sẽ trả về đối tượng service container:

```
$container = app();
```

Bạn có thể truyền một class hoặc tên interface để trả lại chúng từ container:

```php
$api = app('HelpSpot\API');
```

auth()

Hàm `auth()` sẽ trả về một đối tượng xác thực người dùng. Bạn có thể sử dụng nó như một sự thay thế cho facade `Auth`:

```php
$user = auth()->user();
```

Nếu cần, bạn có thể chỉ định cụ thể đối tượng xác thực nào bạn muốn truy cập:

```php
$user = auth('admin')->user();
```

back()

Hàm `back()` sẽ tạo phản hồi HTTP chuyển hướng đến trang trước đó của người dùng:

```php
return back($status = 302, $headers = [], $fallback = '/');

return back();
```

bcrypt()

Hàm `bcrypt()` sẽ băm giá trị đã cho bằng cách sử dụng phương pháp mật mã hóa *Bcrypt*. Bạn có thể sử dụng hàm này như một sự thay thế cho facade `Hash:`

```php
$password = bcrypt('my-secret-password');
```

blank()

Hàm `blank` xác định xem giá trị đã cho có phải là "trống" hay không:

```php
blank('');
```

```
blank('   ');
blank(null);
blank(collect());

// true

blank(0);
blank(true);
blank(false);

// false
```

Đối với nghịch đảo của hàm `blank`, hãy xem phương thức `filled`.

`broadcast()`

Chức năng `boardcast` phát event đã cho cho chương trình theo dõi của nó:

```
broadcast(new UserRegistered($user));

broadcast(new UserRegistered($user))->toOthers();
```

`cache()`

Chức năng cache() có thể được sử dụng để lấy các giá trị từ bộ nhớ cache. Nếu khóa đã cho không tồn tại trong bộ nhớ cache, một giá trị mặc định tùy chọn sẽ được trả về:

```
$value = cache('key');

$value = cache('key', 'default');
```

Bạn có thể thêm các mục vào bộ nhớ cache bằng cách truyền một mảng các cặp khóa/giá trị vào hàm. Bạn cũng nên truyền số giây hoặc thời lượng mà giá trị được lưu trong bộ nhớ cache phải được coi là hợp lệ:

```
cache(['key' => 'value'], 300);
```

```php
cache(['key' => 'value'], now()->addSeconds(10));
```

class_uses_recursive()

Hàm `class_uses_recursive()` sẽ trả về tất cả các trait được sử dụng bởi một class, bao gồm các trait được sử dụng bởi tất cả các class cha của nó:

```php
$traits = class_uses_recursive(App\Models\User::class);
```

collect()

Hàm `collect()` sẽ tạo một đối tượng bộ sưu tập từ giá trị đã cho:

```php
$collection = collect(['taylor', 'abigail']);
```

config()

Hàm `config()` nhận giá trị của một biến cấu hình. Các giá trị cấu hình có thể được truy cập bằng cú pháp "dấu chấm", bao gồm tên của tập tin và tùy chọn bạn muốn truy cập. Giá trị mặc định có thể được chỉ định và được trả về nếu tùy chọn cấu hình không tồn tại:

```php
$value = config('app.timezone');

$value = config('app.timezone', $default);
```

Bạn có thể đặt các biến cấu hình trong thời gian chạy (runtime) bằng cách truyền một mảng các cặp khóa/giá trị. Tuy nhiên, lưu ý rằng hàm này chỉ ảnh hưởng đến giá trị cấu hình cho yêu cầu hiện tại và không cập nhật giá trị cấu hình thực của bạn:

```php
config(['app.debug' => true]);
```

cookie()

Hàm `cookie()` sẽ tạo một đối tượng cookie mới:

```
$cookie = cookie('name', 'value', $minutes);
```

csrf_field()

Hàm `csrf_field` tạo field ẩn HTML chứa giá trị của mã thông báo CSRF. Ví dụ: sử dụng cú pháp Blade:

```
{{ csrf_field() }}
```

csrf_token()

Hàm `csrf_token()` sẽ truy xuất giá trị của mã thông báo CSRF hiện tại:

```
$token = csrf_token();
```

decrypt()

Hàm `decrypt()` sẽ giải mã giá trị đã cho. Bạn có thể sử dụng hàm này thay thế cho facade Crypt:

```
$password = decrypt($value);
```

dd()

Hàm `dd()` sẽ biểu lộ các biến đã cho và kết thúc thực thi chương trình:

```
dd($value);

dd($value1, $value2, $value3, ...);
```

Nếu bạn không muốn dừng chương trình của mình, thì bạn hãy sử dụng hàm `dump` thay thế.

dispatch()

Hàm dispatch() sẽ đẩy công việc đã cho vào hàng chờ công việc Laravel:

```
dispatch(new App\Jobs\SendEmails);
```

dump()

Hàm `dump()` sẽ biểu lộ các biến đã cho:

```
dump($value);

dump($value1, $value2, $value3, ...);
```

Nếu bạn muốn dừng thực thi chương trình sau khi biểu lộ các biến, hãy sử dụng hàm `dd` để thay thế.

encrypt()

Hàm `encrypt()` mã hóa giá trị đã cho. Bạn có thể sử dụng hàm này để thay thế cho facade `Crypt`:

```
$secret = encrypt('my-secret-value');
```

env()

Hàm `env()` truy xuất giá trị của một biến môi trường hoặc trả về giá trị mặc định:

```
$env = env('APP_ENV');

$env = env('APP_ENV', 'production');
```

Nếu bạn thực thi lệnh `config:cache` trong quá trình triển khai, bạn nên đảm bảo rằng bạn chỉ đang gọi hàm `env` từ bên trong các tập tin cấu hình của mình. Khi cấu hình đã được lưu vào bộ đệm, tập tin `.env` sẽ không được tải và tất cả các lệnh gọi đến hàm `env` sẽ trả về `null`.

event()

Hàm `event()` gửi sự kiện đã cho đến các chương trình theo dõi của nó:

```
event(new UserRegistered($user));
```

fake()

Hàm `fake()` sẽ giải quyết một singleton của Faker từ container, có thể hữu ích khi tạo dữ liệu giả trong các model factory, tạo cơ sở dữ liệu, thử nghiệm và khảo sát view:

```
@for($i = 0; $i < 10; $i++)
  <dl>
    <dt>Name</dt>
    <dd>{{ fake()->name() }}</dd>

    <dt>Email</dt>
    <dd>{{ fake()->unique()->safeEmail() }}</dd>
  </dl>
@endfor
```

Theo mặc định, hàm fake sẽ sử dụng tùy chọn cấu hình `app.faker_locale` trong tập tin cấu hình `config/app.php` của bạn; tuy nhiên, bạn cũng có thể chỉ định ngôn ngữ bằng cách chuyển nó cho hàm `fake.` Mỗi ngôn ngữ sẽ phân giải một singleton riêng lẻ:

```
fake('nl_NL')->name()
```

filled()

Hàm filled sẽ xác định xem giá trị đã cho không phải là "trống" hay không:

```
filled(0);
filled(true);
filled(false);

// true
```

```
filled('');
filled('   ');
filled(null);
filled(collect());

// false
```

Đối với nghịch đảo của `filled`, hãy xem phương thức `blank`.

info()

Hàm `info()` sẽ ghi thông tin vào nhật ký ứng dụng của bạn:

```
info('Some helpful information!');
```

Một mảng dữ liệu theo ngữ cảnh cũng có thể được truyền cho hàm:

```
info('User login attempt failed.', ['id' => $user->id]);
```

logger()

Hàm `logger()` có thể được sử dụng để viết thông báo các cấp độ gỡ lỗi vào tập tin log:

```
logger('Debug message');
```

Một mảng dữ liệu theo ngữ cảnh cũng có thể được truyền cho hàm:

```
logger('User has logged in.', ['id' => $user->id]);
```

Một đối tượng logger sẽ được trả về nếu không có giá trị nào được truyền vào cho hàm:

```
logger()->error('You are not allowed here.');
```

`method_field()`

Hàm `method_field()` sẽ tạo một field ẩn HTML chứa giá trị giả mạo của động từ HTTP của biểu mẫu. Ví dụ: sử dụng cú pháp Blade:

```
<form method="POST">
  {{ method_field('DELETE') }}
</form>
```

`now()`

Hàm `now()` sẽ tạo một đối tượng `Illuminate\Support\Carbon` mới cho thời điểm hiện tại:

```
$now = now();
```

`old()`

Hàm `old()` sẽ truy xuất một giá trị đầu vào trước đó mà đã được đưa vào session:

```
$value = old('value');

$value = old('value', 'default');
```

Vì "giá trị mặc định" được cung cấp làm đối số thứ hai cho hàm `old` thường là một thuộc tính của mô hình Eloquent, Laravel cho phép bạn chỉ cần truyền toàn bộ mô hình Eloquent làm đối số thứ hai cho hàm `old.` Khi làm như vậy, Laravel sẽ giả sử đối số đầu tiên được cung cấp cho hàm `old` là tên của thuộc tính Eloquent nên được coi là "giá trị mặc định":

```
{{ old('name', $user->name) }}

// Is equivalent to...

{{ old('name', $user) }}
```

optional()

Hàm `optional()` chấp nhận bất kỳ đối số nào và cho phép bạn truy cập thuộc tính hoặc gọi phương thức trên đối tượng đó. Nếu đối tượng đã cho là `null`, thì các thuộc tính và phương thức sẽ trả về `null` thay vì tạo ra một lỗi gì đó:

```
return optional($user->address)->street;

{!! old('name', optional($user)->name) !!}
```

Hàm `optional()` cũng chấp nhận một hàm xử lý làm đối số thứ hai của nó. Hàm xử lý sẽ được gọi nếu giá trị được cung cấp làm đối số đầu tiên không phải là `null`:

```
return optional(User::find($id), function ($user) {
  return $user->name;
});
```

policy()

Phương thức `policy()` dùng để truy xuất một đối tượng chính sách cho một class nhất định:

```
$policy = policy(App\Models\User::class);
```

redirect()

Hàm `redirect()` sẽ trả về phản hồi HTTP chuyển hướng hoặc trả về đối tượng chuyển hướng nếu được gọi mà không có đối số:

```
return redirect($to = null, $status = 302, $headers = [], $https = null);

return redirect('/home');

return redirect()->route('route.name');
```

`report()`

Hàm `report()` sẽ báo cáo một exception bằng cách sử dụng chương trình xử lý exception của bạn:

```
report($e);
```

Hàm `report()` cũng chấp nhận một chuỗi câu làm đối số. Khi một chuỗi câu được cấp cho hàm, hàm sẽ tạo một exception với chuỗi câu đã cho dưới dạng thông báo của nó:

```
report('Something went wrong.');
```

`request()`

Hàm `request()` sẽ trả về đối tượng request hiện tại hoặc lấy giá trị của trường đầu vào từ request hiện tại:

```
$request = request();

$value = request('key', $default);
```

`rescue()`

Hàm `rescue()` sẽ thực thi hàm xử lý đã cho và bắt bất kỳ exception nào xảy ra trong quá trình thực thi của nó. Tất cả các exception được bắt sẽ được gửi đến chương trình xử lý exception của bạn; tuy nhiên, request sẽ tiếp tục xử lý:

```
return rescue(function () {
  return $this->method();
});
```

Bạn cũng có thể truyền thêm đối số thứ hai vào hàm `rescue.` Đối số này sẽ là giá trị "mặc định" sẽ được trả về nếu một exception xảy ra trong khi thực thi hàm xử lý:

```
return rescue(function () {
```

```
  return $this->method();
}, false);

return rescue(function () {
  return $this->method();
}, function () {
  return $this->failure();
});
```

resolve()

Hàm `resolve()` sẽ phân giải một class hoặc tên interface đã cho thành một đối tượng bằng cách sử dụng service container:

```
$api = resolve('HelpSpot\API');
```

response()

Hàm `response()` sẽ tạo một đối tượng phản hồi hoặc lấy một đối tượng của factory phản hồi:

```
return response('Hello World', 200, $headers);

return response()->json(['foo' => 'bar'], 200, $headers);
```

retry()

Hàm `retry()` sẽ cố gắng thực thi lệnh callback đã cho cho đến khi đạt đến ngưỡng thử tối đa nhất định. Nếu lệnh callback không đưa ra bất kỳ exception nào, thì giá trị trả về của nó sẽ được trả về. Nếu lệnh callback đưa ra ngoại lệ, thì nó sẽ tự động được thử lại. Nếu giá trị tối đa số lần thử bị vượt quá hạn mức, exception sẽ thực sự được đưa ra:

```
return retry(5, function () {
  // Attempt 5 times while resting 100ms between attempts...
}, 100);
```

Nếu bạn muốn tính toán thủ công số mili giây để tạm nghỉ xử lý giữa các lần thử, bạn có thể truyền một hàm xử lý làm đối số thứ ba cho hàm `retry:`

```
return retry(5, function () {
  // ...
}, function ($attempt, $exception) {
  return $attempt * 100;
});
```

Để thuận tiện, bạn có thể cung cấp một mảng làm đối số đầu tiên cho hàm `retry`. Mảng này sẽ được sử dụng để xác định có bao nhiêu mili giây để nghỉ giữa các lần thử tiếp theo:

```
return retry([100, 200], function () {
  // Sleep for 100ms on first retry, 200ms on second retry...
});
```

Để chỉ thử lại trong các điều kiện cụ thể, bạn có thể truyền một hàm xử lý khác làm đối số thứ tư cho hàm `retry` như sau:

```
return retry(5, function () {
  // ...
}, 100, function ($exception) {
  return $exception instanceof RetryException;
});
```

`session()`

Hàm `session()` có thể được sử dụng để lấy hoặc đặt các giá trị session:

```
$value = session('key');
```

Bạn có thể đặt giá trị bằng cách truyền một mảng các cặp khóa/giá trị vào hàm:

```
session(['chairs' => 7, 'instruments' => 3]);
```

Lưu trữ session sẽ được trả về nếu không có giá trị nào được truyền cho hàm:

```
$value = session()->get('key');

session()->put('key', $value);
```

tap()

Hàm `tap()` chấp nhận hai đối số: một biến `$value` tùy ý và một hàm xử lý. Biến `$value` sẽ được truyền cho hàm xử lý và sau đó được trả về bởi hàm `tap`. Giá trị trả về của hàm xử lý không liên quan:

```
$user = tap(User::first(), function ($user) {
  $user->name = 'taylor';

  $user->save();
});
```

Nếu không có hàm xử lý nào được truyền vào hàm `tap`, thì bạn có thể gọi bất kỳ phương thức nào trên biến `$value` đã cho. Giá trị trả về của phương thức bạn gọi sẽ luôn là biến `$value`, bất kể phương thức có thực sự trả về giá trị gì trong định nghĩa của nó. Ví dụ: Phương thức `update` của Eloquent thường trả về một số nguyên. Tuy nhiên, chúng ta có thể buộc phương thức trả về chính model bằng cách xâu chuỗi tiếp nối các phương thức theo sau phương thức `update` thông qua hàm `tap`:

```
$user = tap($user)->update([
  'name' => $name,
  'email' => $email,
]);
```

Để thêm phương thức `tap` vào một class, thì bạn có thể thêm trait `Illuminate\Support\Traits\Tappable` vào class. Phương thức `tap` của trait này chấp nhận một Hàm làm đối số duy nhất của nó. Đối tượng object sẽ được tự truyền vào hàm và sau đó được trả về bằng phương thức `tap`:

```
return $user->tap(function ($user) {
```

```
  //
});
```

`throw_if()`

Hàm `throw_if()` sẽ đưa ra exception đã cho nếu một biểu thức boolean nhất định cho giá trị `true`:

```
throw_if(! Auth::user()->isAdmin(), AuthorizationException::class);

throw_if(
  ! Auth::user()->isAdmin(),
  AuthorizationException::class,
  'You are not allowed to access this page.'
);
```

`throw_unless()`

Hàm `throw_unless()` sẽ đưa ra exception đã cho nếu một biểu thức boolean nhất định cho giá trị `false`:

```
throw_unless(Auth::user()->isAdmin(), AuthorizationException::class);

throw_unless(
  Auth::user()->isAdmin(),
  AuthorizationException::class,
  'You are not allowed to access this page.'
);
```

`today()`

Hàm `today()` sẽ tạo một đối tượng `Illuminate\Support\Carbon` mới cho ngày hiện tại:

```php
$today = today();
```

trait_uses_recursive()

Hàm `trait_uses_recursive()` sẽ trả về tất cả các trait được sử dụng bởi một trait nào đó:

```php
$traits = trait_uses_recursive(\Illuminate\Notifications\Notifiable::class);
```

transform()

Hàm `transform()` sẽ thực thi một hàm xử lý trên một giá trị nhất định nếu giá trị đó không trống và sau đó trả về giá trị trả về của hàm xử lý:

```php
$callback = function ($value) {
  return $value * 2;
};

$result = transform(5, $callback);

// 10
```

Giá trị mặc định hoặc hàm xử lý có thể được truyền vào làm đối số thứ ba cho hàm. Giá trị này sẽ được trả về nếu giá trị đã cho trống:

```php
$result = transform(null, $callback, 'The value is blank');

// The value is blank
```

validator()

Hàm `validator()` sẽ tạo một đối tượng chương trình xác thực mới với các đối số đã cho. Bạn có thể sử dụng nó như một sự thay thế cho facade `Validator`:

```php
$validator = validator($data, $rules, $messages);
```

value()

Hàm `value()` sẽ trả về giá trị mà nó được cung cấp. Tuy nhiên, nếu bạn truyền một hàm xử lý vào hàm `value,` thì hàm xử lý này sẽ được thực thi và giá trị trả về của nó cũng sẽ được trả về:

```php
$result = value(true);

// true

$result = value(function () {
  return false;
});

// false
```

view()

Hàm `view()` sẽ truy xuất một đối tượng view:

```php
return view('auth.login');
```

with()

Hàm `with()` sẽ trả về giá trị mà nó được cung cấp. Nếu một hàm nặc danh được truyền làm đối số thứ hai cho hàm, thì hàm nặc danh sẽ được thực thi và giá trị trả về của nó sẽ được trả về:

```php
$callback = function ($value) {
  return is_numeric($value) ? $value * 2 : 0;
};

$result = with(5, $callback);
```

```php
// 10

$result = with(null, $callback);

// 0

$result = with(5, null);

// 5
```