Course - Laravel Framework

Việt hóa

Các tính năng bản địa của Laravel cung cấp tiện ích để truy xuất các chuỗi bằng nhiều ngôn ngữ khác nhau, cho phép bạn dễ dàng hỗ trợ nhiều ngôn ngữ trong ứng dụng của mình.

Tags: laravel viet hoa, laravel

Giới thiệu

Các tính năng bản địa hóa của Laravel cung cấp tiện ích để truy xuất các chuỗi bằng nhiều ngôn ngữ khác nhau, cho phép bạn dễ dàng hỗ trợ nhiều ngôn ngữ trong ứng dụng của mình.

Laravel cung cấp hai cách để quản lý các chuỗi thông dịch ngôn ngữ. Đầu tiên, các chuỗi ngôn ngữ có thể được lưu trữ trong các tập tin trong thư mục *resources/lang*. Trong thư mục này, có thể có các thư mục con cho mỗi ngôn ngữ được ứng dụng hỗ trợ. Đây là cách tiếp cận mà Laravel sử dụng để quản lý các chuỗi thông dịch cho các tính năng Laravel được tích hợp sẵn, chẳng hạn như thông báo lỗi xác thực:

```
/resources
  /lang
    /en
      messages.php
    /es
      messages.php
```

Hoặc, các chuỗi thông dịch có thể được xác định trong các tập tin JSON được đặt trong thư mục *resources/lang*. Khi thực hiện phương pháp này, mỗi ngôn ngữ được ứng dụng của bạn hỗ trợ sẽ có một tập tin JSON tương ứng trong thư mục này. Cách tiếp cận này được khuyến nghị cho các ứng dụng có số lượng lớn các chuỗi có thể thông dịch được:

```
/resources
  /lang
    en.json
    es.json
```

Chúng ta sẽ thảo luận về từng cách tiếp cận để quản lý chuỗi thông dịch trong tài liệu này.

Định cấu hình ngôn ngữ

Ngôn ngữ mặc định cho ứng dụng của bạn được lưu trữ trong tùy chọn cấu hình `locale` của tập tin cấu hình `config/app.php`. Bạn có thể tự do sửa đổi giá trị này để phù hợp với nhu cầu của ứng dụng của mình.

Bạn có thể sửa đổi ngôn ngữ mặc định cho một yêu cầu HTTP request duy nhất trong thời

gian chạy bằng cách sử dụng phương thức `setLocale` được cung cấp bởi facade `App`:

```
use Illuminate\Support\Facades\App;

Route::get('/greeting/{locale}', function ($locale) {
  if (! in_array($locale, ['en', 'es', 'fr'])) {
    abort(400);
  }

  App::setLocale($locale);

  //
});
```

Bạn có thể cấu hình "ngôn ngữ dự phòng", ngôn ngữ này sẽ được sử dụng khi ngôn ngữ hiện tại không chứa một chuỗi thông dịch nào đó. Giống như ngôn ngữ mặc định, ngôn ngữ dự phòng cũng được cấu hình trong tập tin cấu hình config/app.php:

```
'fallback_locale' => 'en',
```

Xác định ngôn ngữ hiện tại

Bạn có thể sử dụng các phương thức `currentLocale` và `isLocale` trên facade `App` để xác định ngôn ngữ hiện tại hoặc kiểm tra xem ngôn ngữ có phải là một giá trị nhất định hay không:

```
use Illuminate\Support\Facades\App;

$locale = App::currentLocale();

if (App::isLocale('en')) {
  //
}
```

Xác định chuỗi thông dịch

Sử dụng các shortcut key

Thông thường, các chuỗi thông dịch được lưu trữ trong các tập tin trong thư mục *resources/lang*. Trong thư mục này, phải có một thư mục con cho mỗi ngôn ngữ được ứng dụng của bạn hỗ trợ. Đây là cách tiếp cận mà Laravel sử dụng để quản lý các chuỗi thông dịch cho các tính năng Laravel được tích hợp sẵn, chẳng hạn như thông báo lỗi xác thực:

```
/resources
  /lang
    /en
      messages.php
    /es
      messages.php
```

Tất cả các tập tin ngôn ngữ trả về một mảng các chuỗi có khóa. Ví dụ:

```
<?php

// resources/lang/en/messages.php

return [
  'welcome' => 'Welcome to our application!',
];
```

> **Chú ý:** Đối với các ngôn ngữ khác nhau theo lãnh thổ, bạn nên đặt tên cho các thư mục ngôn ngữ theo ISO 15897. Ví dụ: "en_GB" nên được sử dụng cho tiếng Anh Anh thay vì "en-gb".

Sử dụng chuỗi thông dịch làm khóa

Đối với các ứng dụng có số lượng lớn các chuỗi có thể thông dịch được, việc xác định mọi chuỗi bằng "khóa ngắn" có thể trở nên khó hiểu khi tham chiếu đến các khóa trong template của bạn và việc liên tục tạo ra các khóa cho mọi chuỗi thông dịch được ứng dụng của bạn hỗ trợ sẽ rất phức tạp.

Vì lý do này, Laravel cũng cung cấp hỗ trợ xác định chuỗi thông dịch bằng cách sử dụng bản dịch "mặc định" của chuỗi làm khóa. Tập tin thông dịch sử dụng chuỗi thông dịch làm

khóa được lưu trữ dưới dạng tập tin JSON trong thư mục *resources/lang*. Ví dụ: nếu ứng dụng của bạn có bản dịch tiếng Tây Ban Nha, bạn nên tạo một tập tin *resources/lang/es.json*:

```
{
  "I love programming.": "Me encanta programar."
}
```

Xung đột khóa/tập tin

Bạn không nên xác định các khóa chuỗi thông dịch xung đột với các tên tập tin thông dịch khác. Ví dụ: dịch __('Action') cho ngôn ngữ "NL" trong khi tập tin *nl/action.php* tồn tại nhưng tập tin *nl.json* không tồn tại sẽ dẫn đến việc người dịch trả lại nội dung của *nl/action.php*.

Truy xuất chuỗi thông dịch

Bạn có thể truy xuất các chuỗi dịch từ các tập tin ngôn ngữ của mình bằng chức năng trợ giúp `__()`. Nếu bạn đang sử dụng "phím ngắn" để xác định chuỗi thông dịch của mình, bạn nên truyền tập tin chứa khóa và chính khóa đó vào hàm `__()` bằng cú pháp "dấu chấm". Ví dụ: hãy lấy chuỗi thông dịch `welcome` từ tập tin ngôn ngữ *resources/lang/en/messages.php*:

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

Nếu chuỗi thông dịch được chỉ định không tồn tại, hàm `__()` sẽ trả về khóa của chuỗi thông dịch. Vì vậy, khi sử dụng ví dụ trên, hàm `__()` sẽ trả về `messages.welcome` nếu chuỗi thông dịch không tồn tại.

Nếu bạn đang sử dụng các chuỗi thông dịch mặc định của mình làm khóa dịch, bạn nên chuyển bản dịch mặc định của chuỗi của mình cho hàm `__()`;

```
echo __('I love programming.');
```

Một lần nữa, nếu chuỗi thông dịch không tồn tại, hàm `__()` sẽ trả về khóa chuỗi thông dịch mà nó đã được cấp.

Nếu bạn đang sử dụng công cụ tạo template Blade , bạn có thể sử dụng cú pháp echo `{{`

}} để hiển thị chuỗi thông dịch:

```
{{ __('messages.welcome') }}
```

Tham số trong chuỗi thông dịch

Nếu muốn, bạn có thể xác định các tham số trong chuỗi thông dịch của mình. Tất cả các tham số dành sẵn đều có tiền tố là a. Ví dụ, bạn có thể xác định thông báo chào mừng với tham số `:name` như sau:

```
'welcome' => 'Welcome, :name',
```

Để thay thế các tham số khi truy xuất một chuỗi thông dịch, bạn có thể truyền một mảng thay thế làm đối số thứ hai cho hàm `__()`:

```
echo __('messages.welcome', ['name' => 'dayle']);
```

Nếu các tham số của bạn chứa tất cả các chữ cái viết hoa hoặc chỉ có chữ cái đầu tiên được viết hoa, giá trị đã thông dịch sẽ được viết hoa tương ứng:

```
'welcome' => 'Welcome, :NAME', // Welcome, DAYLE
'goodbye' => 'Goodbye, :Name', // Goodbye, Dayle
```

Đa dạng hóa

Đa phương hóa là một vấn đề phức tạp, vì các ngôn ngữ khác nhau có nhiều quy tắc phức tạp để đa phương hóa; tuy nhiên, Laravel có thể giúp bạn dịch các chuỗi theo cách khác nhau dựa trên các quy tắc đa dạng hóa mà bạn xác định. Sử dụng ký tự `|`, bạn sẽ có thể phân biệt các dạng số ít và số nhiều của một chuỗi:

```
'apples' => 'There is one apple|There are many apples',
```

Tất nhiên, đa dạng hóa cũng được hỗ trợ khi sử dụng chuỗi dịch làm khóa :

```
{
  "There is one apple|There are many apples": "Hay una manzana|Hay muchas manzanas"
}
```

Bạn thậm chí có thể tạo các quy tắc đa dạng phức tạp hơn xác định các chuỗi thông dịch cho nhiều phạm vi giá trị:

```
'apples' => '{0} There are none|[1,19] There are some|[20,*] There are many',
```

Sau khi xác định một chuỗi dịch có các tùy chọn đa dạng, bạn có thể sử dụng hàm `trans_choice` để truy xuất dòng cho một "số lượng" nhất định. Trong ví dụ này, vì số lượng lớn hơn một, dạng số nhiều của chuỗi dịch được trả về:

```
echo trans_choice('messages.apples', 10);
```

Bạn cũng có thể xác định các thuộc tính tham số trong các chuỗi đa hóa. Các tham số này có thể được thay thế bằng cách truyền một mảng làm đối số thứ ba cho hàm `trans_choice`:

```
'minutes_ago' => '{1} :value minute ago|[2,*] :value minutes ago',

echo trans_choice('time.minutes_ago', 5, ['value' => 5]);
```

Nếu bạn muốn hiển thị giá trị số nguyên đã được truyền cho hàm `trans_choice`, thì bạn có thể sử dụng tham số `:count` được tích hợp sẵn:

```
'apples' => '{0} There are none|{1} There is one|[2,*] There are :count',
```

Ghi đè tập tin language của một package

Một số package có thể gửi kèm theo các tập tin ngôn ngữ riêng của chúng. Thay vì thay đổi các tập tin cốt lõi của pacakge để điều chỉnh ngôn ngữ, thì bạn có thể ghi đè chúng bằng cách đặt tập tin vào trong thư mục *resources/lang/vendor/{package}/{locale}*.

Ví dụ: nếu bạn cần ghi đè các chuỗi dịch tiếng Anh *messages.php* cho một package có tên

skyrim/hearthfire, thì bạn nên đặt tập tin ngôn ngữ tại *resources/lang/vendor/hearthfire /en/messages.php*. Trong tập tin này, bạn chỉ nên xác định các chuỗi thông dịch mà bạn muốn ghi đè. Mọi chuỗi thông dịch mà bạn không ghi đè sẽ vẫn được tải từ các tập tin ngôn ngữ gốc của package.