Course - Laravel Framework

Phát triển package

Package là cách thức chính để thêm chức năng vào Laravel. Package có thể là bất cứ thư viện bổ sung nào chẳng hạn như để làm việc với các ngày tháng có package Carbon hoặc một package nào đó cho phép bạn liên kết tập tin với các mô hình Eloquent như Thư viện media Laravel "Spatie".

Tags: package development, phat trien package, laravel

Giới thiệu

Package là cách thức chính để thêm chức năng vào Laravel. Package có thể là bất cứ thư viện bổ sung nào chẳng hạn như để làm việc với các ngày tháng có package Carbon hoặc một package nào đó cho phép bạn liên kết tập tin với các mô hình Eloquent như Thư viện media Laravel "Spatie".

Có nhiều loại package khác nhau. Một số package có tính độc lập, có nghĩa là chúng hoạt động với bất kỳ PHP framework nào. `Carbon` và `PHPUnit` là những ví dụ về các package độc lập. Bất kỳ package nào trong số này đều có thể được sử dụng với Laravel bằng cách yêu cầu chúng trong tệp *composer.json* của bạn.

Mặt khác, cũng có các package khác được thiết kế đặc biệt để chỉ sử dụng với Laravel. Các package này có thể có route, controller, view và cấu hình dành riêng để nâng cao cho các ứng dụng Laravel. Hướng dẫn này chủ yếu nói về việc phát triển các package dành riêng cho Laravel.

Lưu ý về facade

Khi viết một ứng dụng Laravel, nhìn chung không thành vấn đề nếu bạn sử dụng contract hay facade vì cả hai đều cung cấp mức độ kiểm nghiệm về cơ bản khá ngang nhau. Tuy nhiên, khi viết package, package của bạn thường sẽ không có quyền truy cập vào tất cả các helper thử nghiệm của Laravel. Nếu bạn muốn có thể viết các bài kiểm nghiệm package của mình như thể package được cài đặt bên trong một ứng dụng Laravel điển hình, bạn có thể sử dụng package *Orchestral Testbench*.

Khám phá package

Trong tập tin cấu hình *config/app.php* của ứng dụng Laravel, tùy chọn `providers` sẽ xác định danh sách các service provider sẽ được tải bởi Laravel. Khi ai đó cài đặt package của bạn, bạn thường sẽ muốn service provider của mình được đưa vào danh sách này. Thay vì yêu cầu người dùng thêm service provider của bạn vào danh sách theo cách thủ công, thì bạn có thể xác định provider trong phần bổ sung `extra` của tập tin *composer.json* của package. Ngoài các provider service, thì bạn cũng có thể liệt kê bất kỳ facade nào mà bạn muốn đăng ký:

```
"extra": {
  "laravel": {
    "providers": [
```

```
      "Barryvdh\\Debugbar\\ServiceProvider"
    ],
    "aliases": {
      "Debugbar": "Barryvdh\\Debugbar\\Facade"
    }
  }
},
```

Khi package của bạn đã được cấu hình để có thể khám phá, thì Laravel sẽ tự động đăng ký các service provider và facade khi nó được cài đặt, tạo ra trải nghiệm cài đặt thuận tiện cho người dùng package của bạn.

Không cho khám phá package

Nếu bạn là người tiêu dùng của một package và muốn tắt tính năng khám phá package cho một package nào đó, bạn có thể liệt kê tên package trong phần bổ sung `extra` của tập tin *composer.json* của ứng dụng:

```
"extra": {
  "laravel": {
    "dont-discover": [
      "barryvdh/laravel-debugbar"
    ]
  }
},
```

Bạn có thể tắt tính năng khám phá package cho tất cả các package bằng ký tự wildcard * bên trong chỉ thị không tìm hiểu trong ứng dụng của bạn:

```
"extra": {
  "laravel": {
    "dont-discover": [
      "*"
    ]
  }
},
```

Các service provider

Các service provider là điểm kết nối giữa package của bạn và Laravel. Service provider sẽ chịu trách nhiệm ràng buộc mọi thứ vào service container của Laravel và thông báo cho Laravel nơi tải các tài nguyên của package như các tập tin view, cấu hình và ngôn ngữ bản địa.

Một service provider sẽ mở rộng class `Illuminate\Support\ServiceProvider` và chứa hai phương thức: `register` và `boot`. Class `ServiceProvider` cơ sở sẽ nằm trong package `illuminate/support`, bạn nên thêm class này vào các package phụ thuộc của riêng mình. Để tìm hiểu thêm về cấu trúc và mục đích của các service provider, hãy xem tài liệu về chúng.

Tài nguyên

Cấu hình

Thông thường, bạn sẽ cần xuất bản tập tin cấu hình của package vào thư mục *config* của ứng dụng. Điều này sẽ cho phép người dùng package của bạn dễ dàng ghi đè các tùy chọn cấu hình mặc định của bạn. Để cho phép các tập tin cấu hình của bạn được xuất bản, hãy gọi phương thức `publishes` từ phương thức `boot` của service provider của bạn:

```
/**
 * Bootstrap any package services.
 *
 * @return void
 */
public function boot()
{
  $this->publishes([
    __DIR__.'/../config/courier.php' => config_path('courier.php'),
  ]);
}
```

Bây giờ, khi người dùng package của bạn thực hiện lệnh `vendor:publish` của Laravel, tập tin của bạn sẽ được sao chép vào vị trí xuất bản được chỉ định. Khi cấu hình của bạn đã được xuất bản, các giá trị của nó có thể được truy cập giống như bất kỳ tập tin cấu hình khác:

```php
$value = config('courier.option');
```

> **Chú ý:** Bạn không nên tạo các hàm nặc danh trong các tập tin cấu hình của mình. Chúng không thể được mã hóa một cách chính xác khi người dùng thực hiện lệnh Artisan `config:cache`.

Cấu hình package mặc định

Bạn cũng có thể hợp nhất tập tin cấu hình package của riêng mình với bản sao đã xuất bản của ứng dụng. Điều này sẽ cho phép người dùng của bạn chỉ xác định các tùy chọn mà họ thực sự muốn ghi đè trong bản sao đã xuất bản của tập tin cấu hình. Để hợp nhất các giá trị tập tin cấu hình, hãy sử dụng phương thức `mergeConfigFrom` trong phương thức `register` của service provider của bạn.

Phương thức `mergeConfigFrom` chấp nhận đường dẫn đến tập tin cấu hình package của bạn làm đối số đầu tiên và tên của bản sao ứng dụng của tập tin cấu hình làm đối số thứ hai:

```php
/**
 * Register any application services.
 *
 * @return void
 */
public function register()
{
  $this->mergeConfigFrom(
    __DIR__.'/../config/courier.php', 'courier'
  );
}
```

> **Chú ý:** Phương thức này chỉ hợp nhất cấp độ đầu tiên của mảng cấu hình. Nếu người dùng của bạn có mảng cấu hình đa chiều, thì các tùy chọn có thể sẽ bị thiếu và không được hợp nhất.

Các route

Nếu package của bạn chứa các route, bạn có thể tải chúng bằng phương thức `loadRoutesFrom`. Phương thức này sẽ tự động xác định xem các route của ứng dụng có được lưu vào bộ nhớ đệm hay không và sẽ không tải các tập tin route của bạn nếu các route đã được lưu trong bộ đệm:

```
/**
 * Bootstrap any package services.
 *
 * @return void
 */
public function boot()
{
  $this->loadRoutesFrom(__DIR__.'/../routes/web.php');
}
```

Các migration

Nếu package của bạn chứa các migration cơ sở dữ liệu, bạn có thể sử dụng phương thức `loadMigrationsFrom` để cho Laravel biết cách tải chúng. Phương thức `loadMigrationsFrom` chấp nhận đường dẫn đến migration trong package của bạn làm đối số duy nhất của nó:

```
/**
 * Bootstrap any package services.
 *
 * @return void
 */
public function boot()
{
  $this->loadMigrationsFrom(__DIR__.'/../database/migrations');
}
```

Sau khi đăng ký migration cho package của bạn, chúng sẽ tự động được chạy khi lệnh migrate được thực thi. Bạn không cần xuất chúng vào thư mục `database/migrations` của ứng dụng.

Thông dịch

Nếu package của bạn chứa các tập tin thông dịch, thì bạn có thể sử dụng phương thức `loadTranslationsFrom` để cho Laravel biết cách tải chúng. Ví dụ: nếu package của bạn được đặt tên là `courier`, thì bạn nên thêm thông tin sau vào phương thức `boot` của service provider:

```
/**
 * Bootstrap any package services.
 *
 * @return void
 */
public function boot()
{
  $this->loadTranslationsFrom(__DIR__.'/../resources/lang', 'courier');
}
```

Các bản thông dịch package được tham chiếu bằng cách sử dụng quy ước cú pháp `package::file.line`. Vì vậy, bạn có thể tải dòng chào mừng của package courier từ tập tin tin nhắn như sau:

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

Xuất bản bản dịch

Nếu bạn muốn xuất bản các bản dịch của package vào thư mục *resources/lang/vendor* của ứng dụng, bạn có thể sử dụng phương thức `publishes` của service provider. Phương thức `publishes` chấp nhận một loạt các đường dẫn package và vị trí xuất bản mong muốn của chúng. Ví dụ: để xuất bản các tập tin bản dịch cho package `courier`, thì bạn có thể làm như sau:

```
/**
 * Bootstrap any package services.
 *
 * @return void
 */
public function boot()
```

```
{
  $this->loadTranslationsFrom(__DIR__.'/../resources/lang', 'courier');

  $this->publishes([
    __DIR__.'/../resources/lang' => resource_path('lang/vendor/courier'),
  ]);
}
```

Bây giờ, khi người dùng package của bạn thực thi lệnh Artisan của Laravel vendor:publish, các bản dịch của package của bạn sẽ được xuất bản đến vị trí xuất bản được chỉ định.

Views

Để đăng ký các view của package với Laravel, bạn cần cho Laravel biết vị trí của các view. Bạn có thể thực hiện việc này bằng phương thức `loadViewsFrom` của service provider. Phương thức `loadViewsFrom` chấp nhận hai đối số: đường dẫn đến các template của bạn và tên package của bạn. Ví dụ: nếu tên package của bạn là `courier`, bạn sẽ thêm thông tin sau vào phương thức `boot` của service provider:

```
/**
 * Bootstrap any package services.
 *
 * @return void
 */
public function boot()
{
  $this->loadViewsFrom(__DIR__.'/../resources/views', 'courier');
}
```

View trong package được tham chiếu bằng cách sử dụng cú pháp quy ước `package::view`. Vì vậy, khi đường dẫn đến view của bạn được đăng ký trong một service provider, thì bạn có thể tải view cho dashboard từ package `courier` như sau:

```
Route::get('/dashboard', function () {
  return view('courier::dashboard');
});
```

Chồng đè lên view của package

Khi bạn sử dụng phương thức `loadViewsFrom`, thì Laravel sẽ thực sự đăng ký hai vị trí cho các view của bạn: thư mục *resources/views/vendor* của ứng dụng và thư mục bạn chỉ định. Vì vậy, sử dụng package `courier` làm ví dụ, trước tiên Laravel sẽ kiểm tra xem phiên bản tùy chỉnh của view đã được nhà phát triển đặt trong thư mục *resources/views/vendor/courier* hay chưa. Sau đó, nếu view chưa được tùy chỉnh, Laravel sẽ tìm kiếm thư mục view của package mà bạn đã chỉ định trong lệnh gọi `loadViewsFrom`. Điều này giúp người dùng của package dễ dàng tùy chỉnh/ghi đè các view trong package của bạn.

Xuất bản view

Nếu bạn muốn cung cấp các view của mình để xuất bản lên thư mục *resources/views/vendor* của ứng dụng, thì bạn có thể sử dụng phương thức `pubishes` của service provider. Phương thức `publishes` chấp nhận một loạt các đường dẫn view của package và các vị trí xuất bản mong muốn của chúng:

```
/**
 * Bootstrap the package services.
 *
 * @return void
 */
public function boot()
{
  $this->loadViewsFrom(__DIR__.'/../resources/views', 'courier');

  $this->publishes([
    __DIR__.'/../resources/views' => resource_path('views/vendor/courier'),
  ]);
}
```

Bây giờ, khi người dùng package của bạn thực hiện lệnh Artisan `vendor:publish`, thì các view của package của bạn sẽ được sao chép vào vị trí xuất bản được chỉ định.

Các component view

Nếu package của bạn chứa các component về view, thì bạn có thể sử dụng phương thức `loadViewComponentsAs` để cho Laravel tải chúng. Phương thức

`loadViewComponentsAs` chấp nhận hai đối số: tiền tố tag cho các component view của bạn và một mảng tên class của component view của bạn. Ví dụ: nếu tiền tố package của bạn là `courier` và bạn có các component view như `Alert` và `Button`, thì bạn sẽ thêm phần sau vào phương thức `boot` của service provider:

```
use Courier\Components\Alert;
use Courier\Components\Button;

/**
 * Bootstrap any package services.
 *
 * @return void
 */
public function boot()
{
  $this->loadViewComponentsAs('courier', [
    Alert::class,
    Button::class,
  ]);
}
```

Sau khi các component view của bạn được đăng ký trong một service provider, thì bạn có thể tham chiếu chúng trong view của mình như sau:

```
<x-courier-alert />

<x-courier-button />
```

Các component ẩn danh

Nếu package của bạn chứa các component ẩn danh, thì chúng phải được đặt trong thư mục *components* của thư mục "*views*" của package của bạn (như được chỉ định bởi `loadViewsFrom`). Sau đó, bạn có thể hiển thị chúng bằng cách thêm tiền tố tên component với namespace của view trong package:

```
<x-courier::alert />
```

Phương thức `command`

Để đăng ký các lệnh Artisan trong package của bạn với Laravel, bạn có thể sử dụng phương thức `command`. Phương thức này chấp nhận một mảng tên của các class lệnh. Khi các lệnh đã được đăng ký, bạn có thể thực hiện chúng bằng Artisan CLI:

```
use Courier\Console\Commands\InstallCommand;
use Courier\Console\Commands\NetworkCommand;

/**
 * Bootstrap any package services.
 *
 * @return void
 */
public function boot()
{
  if ($this->app->runningInConsole()) {
    $this->commands([
      InstallCommand::class,
      NetworkCommand::class,
    ]);
  }
}
```

Tài nguyên Public

Package của bạn có thể có các nội dung như JavaScript, CSS và hình ảnh. Để xuất bản các nội dung này lên thư mục *public* của ứng dụng, thì hãy sử dụng phương thức `publishes` của service provider. Trong ví dụ này, chúng tôi cũng sẽ thêm một tag để nhóm nội dung `public`, thẻ này có thể được sử dụng để dễ dàng xuất bản các nhóm nội dung có liên quan:

```
/**
 * Bootstrap any package services.
 *
 * @return void
 */
public function boot()
```

```
{
  $this->publishes([
    __DIR__.'/../public' => public_path('vendor/courier'),
  ], 'public');
}
```

Bây giờ, khi người dùng package của bạn thực hiện lệnh `vendor:publish`, nội dung của bạn sẽ được sao chép vào vị trí xuất bản được chỉ định. Vì người dùng thường sẽ cần ghi đè nội dung mỗi khi package được cập nhật, nên bạn có thể sử dụng thêm tùy chọn `--force`:

```
php artisan vendor:publish --tag=public --force
```

Xuất bản nhóm tập tin

Bạn có thể muốn xuất bản các nhóm nội dung gói và tài nguyên riêng biệt. Ví dụ: bạn có thể muốn cho phép người dùng xuất bản tập tin cấu hình package của bạn mà không bị buộc phải xuất bản nội dung package của bạn. Bạn có thể làm điều này bằng cách "tagging" chúng khi gọi phương thức `publishes` từ service provider của package. Ví dụ: hãy sử dụng các tag để tạo hai nhóm xuất bản cho package `courier` (`courier-config` và `courier-migrations`) trong phương thức `boot` của service provider của package:

```
/**
 * Bootstrap any package services.
 *
 * @return void
 */
public function boot()
{
  $this->publishes([
    __DIR__.'/../config/package.php' => config_path('package.php')
  ], 'courier-config');

  $this->publishes([
    __DIR__.'/../database/migrations/' => database_path('migrations')
  ], 'courier-migrations');
}
```

Giờ đây, người dùng của bạn có thể xuất bản các nhóm này một cách riêng biệt bằng cách tham chiếu tag của chúng khi thực thi lệnh `vendor:publish`:

```
php artisan vendor:publish --tag=courier-config
```