Course - Laravel Framework

Controller

Thay vì khai báo hết các logic xử lý request của bạn trong các tập tin route dưới dạng hàm nặc danh, thì bạn có thể sử dụng các class "controller". Controller giúp nhóm các logic xử lý request có liên quan với nhau vào trong một class duy nhất.

Tags: controller, laravel

Giới thiệu

Ví dụ: có một class `UserController` có thể xử lý tất cả các yêu cầu đến liên quan đến user, bao gồm hiển thị, tạo mới, cập nhật và xóa user. Thông thường, thì các *controller* được lưu trữ trong thư mục *app/Http/Controllers*.

Tạo Controller

Controller căn bản

Hãy xem một ví dụ cơ bản về *Controller*. Chú ý, các *controller* sẽ mở rộng từ class `Controller` cơ sở đã có sẵn trong Laravel: `App\Http\Controllers\Controller`:

```php
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use App\Models\User;

class UserController extends Controller
{
  /**
   * Show the profile for a given user.
   *
   * @param  int  $id
   * @return \Illuminate\View\View
   */
  public function show($id)
  {
    return view('user.profile', [
      'user' => User::findOrFail($id)
    ]);
  }
}
```

Bây giờ, bạn có thể khai báo một route với phương thức của *controller* này như sau:

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

Route::get('/user/{id}', [UserController::class, 'show']);
```

Khi một request nào đó gửi đến mà khớp với URI của route trên, thì phương thức `show` của class `App\Http\Controllers\UserController` sẽ được gọi và tham số `{id}` của route cũng sẽ được truyền vào trong body của phương thức `show`.

> Các controller không bắt buộc phải mở rộng từ class cơ sở. Tuy nhiên, bạn sẽ không có quyền truy cập vào các tính năng tiện lợi như middleware và các phương thức của class cơ sở.

Controller có action duy nhất

Nếu một *controller action* nào đó quá phức tạp, thì bạn có thể nhận thấy sự thuận tiện khi dành toàn bộ class của *controller* cho *action* đơn lẻ đó. Để thực hiện điều này, bạn có thể khai báo phương thức `__invoke` trong class của *controller*:

```php
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use App\Models\User;

class ProvisionServer extends Controller
{
  /**
   * Provision a new web server.
   *
   * @return \Illuminate\Http\Response
   */
  public function __invoke()
  {
    // ...
  }
```

```
}
```

Khi đăng ký route cho các controller có một action duy nhất, thì bạn không cần ghi rõ phương thức của *controller*. Thay vào đó, bạn chỉ cần truyền tên của *controller* cho route:

```
use App\Http\Controllers\ProvisionServer;

Route::post('/server', ProvisionServer::class);
```

Bạn có thể tạo một *controller* không thể truy cập bằng cách sử dụng tùy chọn `--invokable` của lệnh Artisan `make:controller`:

```
php artisan make:controller ProvisionServer --invokable
```

> Stubs của controller có thể tuỳ biến tại stubs publishing.

Controller và Middleware

Middleware có thể được gắn vào các *controller action* trong các tập tin route của bạn:

```
Route::get('profile', [UserController::class, 'show'])->middleware('auth');
```

Hoặc, bạn có thể khai báo *middleware* trong *contructor* của *controller*. Bằng cách sử dụng phương thức `middleware` trong *contructor* của *controller*, bạn có thể gán *middleware* cho các *controller action* của *controller*:

```
class UserController extends Controller
{
  /**
   * Instantiate a new controller instance.
   *
   * @return void
   */
  public function __construct()
  {
```

```
        $this->middleware('auth');
        $this->middleware('log')->only('index');
        $this->middleware('subscribed')->except('store');
    }
}
```

Controller cũng cho phép bạn đăng ký *middleware* bằng cách sử dụng một hàm nặc danh. Điều này thuận tiện khi khai báo *inline middleware* (*middleware nội tuyến*) cho một *controller* có một *action* duy nhất mà không cần phải tạo ra một class middleware mới nào:

```
$this->middleware(function ($request, $next) {
    return $next($request);
});
```

Các controller kiểu tài nguyên (*resource controller*)

Nếu bạn coi mỗi *Eloquent model* trong ứng dụng của mình là một "resource", thì thông thường, bạn nên thực hiện cùng nhóm action cho từng tài nguyên trong ứng dụng của mình. Ví dụ, hãy tưởng tượng ứng dụng của bạn chứa các resource (tài nguyên), bao gồm một model tên Photo và một model tên Movie. Và người dùng có thể tạo, đọc, sửa hoặc xóa một cách nhanh chóng các tài nguyên này.

Do trường hợp sử dụng phổ biến này, mà Laravel sẽ chỉ định các route tạo, đọc, sửa và xóa ("CRUD") điển hình cho controller vào trong chỉ một dòng code. Để bắt đầu, chúng ta có thể sử dụng tùy chọn `--resource` của lệnh Artisan `make:controller` để tạo *controller* xử lý các hành động này:

```
php artisan make:controller PhotoController --resource
```

Lệnh này sẽ tạo một *controller* với tập tin `app/Http/Controllers/PhotoController.php`. Controller sẽ chứa các phương thức cho ứng với từng hành động thao tác tài nguyên. Tiếp theo, bạn có thể đăng ký một *resource route* trỏ đến *controller*:

```
use App\Http\Controllers\PhotoController;

Route::resource('photos', PhotoController::class);
```

Chỉ khai báo duy nhất *resource route* này là nó đã tạo ra nhiều route để xử lý nhiều hành động khác nhau trên tài nguyên. Controller được tạo sẽ có sẵn các phương thức cho mỗi hành động này. Hãy ghi nhớ, bạn luôn có thể xem tổng quan nhanh chóng về các route của ứng dụng bằng cách chạy lệnh Artisan `route:list`.

Bạn thậm chí có thể đăng ký nhiều *resource controller* cùng một lúc bằng cách truyền một mảng *controller* vào trong phương thức `resources`:

```
Route::resources([
  'photos' => PhotoController::class,
  'posts' => PostController::class,
]);
```

Các action trong resource controller

Method	URI	Action	Tên Route
GET	/photos	index	photos.index
GET	/photos/create	create	photos.create
POST	/photos/store	store	photos.store
GET	/photos/{photo}	show	photos.show
GET	/photos/{photo}/edit	edit	photos.edit
PUT/PATCH	/photos/{photo}	update	photos.update
DELETE	/photos/{photo}	delete	photos.delete

Ứng biến với lỗi thiếu model

Thông thường, một phản hồi HTTP 404 sẽ được tạo ra nếu không tìm thấy model tài nguyên được ngầm gán ghép. Tuy nhiên, bạn có thể ứng phó với tình huống này bằng cách gọi phương thức `missing` khi khai báo *resource route*. Phương thức `missing` này chấp nhận một hàm nặc danh mà sẽ được gọi nếu không thể tìm thấy model ngầm gán ghép cho bất kỳ route nào của tài nguyên:

```
use App\Http\Controllers\PhotoController;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Redirect;

Route::resource('photos', PhotoController::class)
```

```
    ->missing(function (Request $request) {
        return Redirect::route('photos.index');
    });
```

Tạo model resource

Nếu bạn đang sử dụng route gán ghép model và muốn các phương thức của *resource controller* khai kiểu cho một đối tượng model, bạn có thể sử dụng tùy chọn `--model` khi tạo controller:

```
php artisan make:controller PhotoController --model=Photo --resource
```

Tạo ra form request

Bạn có thể bổ sung tùy chọn `--requests` khi tạo *resource controller* để Artisan tạo các class *form request* cho các phương thức lưu trữ và cập nhật của controller:

```
php artisan make:controller PhotoController --model=Photo --resource --requests
```

Phân chia *resource route*

Khi khai báo một *resource route*, bạn cũng có thể tạo một nhóm action con cho controller sẽ xử lý, thay cho việc controller sẽ xử lý toàn bộ các action mặc định:

```
use App\Http\Controllers\PhotoController;

Route::resource('photos', PhotoController::class)->only([
    'index', 'show'
]);

Route::resource('photos', PhotoController::class)->except([
    'create', 'store', 'update', 'destroy'
]);
```

Resource route với API

Khi khai báo các resource route mà sẽ được sử dụng bởi các API, thông thường bạn sẽ muốn loại bỏ các route chứa các HTML form chẳng hạn như tạo và chỉnh sửa. Để thuận tiện, bạn có thể sử dụng phương thức `apiResource` để tự động loại bỏ hai route này:

```
use App\Http\Controllers\PhotoController;

Route::apiResource('photos', PhotoController::class);
```

Bạn có thể đăng ký nhiều *resource controller* với API cùng một lúc bằng cách truyền một mảng tới phương thức `apiResources`:

```
use App\Http\Controllers\PhotoController;
use App\Http\Controllers\PostController;

Route::apiResources([
  'photos' => PhotoController::class,
  'posts' => PostController::class,
]);
```

Để tạo *API resource controller* có thể loại bỏ các phương thức tạo hoặc chỉnh sửa, thì bạn hãy sử dụng tùy chọn `--api` khi thực hiện lệnh `make:controller`:

```
php artisan make:controller PhotoController --api
```

Tài nguyên lồng ghép (Nested Resource)

Đôi khi bạn có thể cần tạo các route cho các resource lồng ghép với nhau. Ví dụ: một ảnh có thể có nhiều comment được đính kèm theo. Để lồng các resource này với nhau, bạn có thể sử dụng ký hiệu "." trong khi khai báo route của mình:

```
use App\Http\Controllers\PhotoCommentController;

Route::resource('photos.comments', PhotoCommentController::class);
```

Route trên sẽ làm cho một *nested resource* có thể được truy cập bằng các URI như sau:

```
/photos/{photo}/comments/{comment}
```

Phạm vi của *nested resource*

Tính năng ngầm ghép model của Laravel có thể tự động xác định phạm vi của những việc lồng ghép sao cho model con có được, được xác nhận là thuộc về model mẹ. Bằng cách vận dụng các phương pháp xác định phạm vi khi tạo *nested resource*, bạn có thể kích hoạt tính năng tự động xác định phạm vi cũng như cho Laravel biết field tài nguyên con nào nên được truy xuất. Để biết thêm thông tin về cách thực hiện điều này, vui lòng xem tài liệu về scoping resource routes.

Lồng ghép với cấp số nhỏ

Thông thường, không hoàn toàn cần thiết phải có cả ID cha và ID con trong một URI vì ID con đã là một số nhận dạng. Khi sử dụng số nhận dạng, chẳng hạn như *khóa chính tăng tự động* (auto-increment primary key) để nhận dạng các model của bạn trong các phân đoạn URI, thì bạn có thể chọn sử dụng lồng ghép với cấp số nhỏ:

```
use App\Http\Controllers\CommentController;

Route::resource('photos.comments', CommentController::class)->shallow();
```

Khai báo route như thế này sẽ tạo ra các route sau:

Verb	URI	Action	Route Name
GET	/photos/{photo}/comments	index	photos.comments.index
GET	/photos/{photo}/comments/create	create	photos.comments.create
POST	/photos/{photo}/comments	store	photos.comments.store
GET	/comments/{comment}	show	comments.show
GET	/comments/{comment}/edit	edit	comments.edit
PUT/PATCH	/comments/{comment}	update	comments.update
DELETE	/comments/{comment}	destroy	comments.destroy

Đặt tên cho các resource route

Mặc định, `Route::resource` sẽ tạo các tham số route cho các *resource route* của bạn dựa trên phiên bản "singularized" của tên resource. Bạn có thể dễ dàng làm lại tên bằng cách sử dụng phương thức `parameters`. Mảng được truyền vào phương thức `parameters` phải là một mảng *assoc* với một bên là tên tài nguyên và một bên là tên tham số:

```
use App\Http\Controllers\AdminUserController;

Route::resource('users', AdminUserController::class)->parameters([
  'users' => 'admin_user'
]);
```

Ví dụ trên sẽ tạo ra URI như sau cho route `show` của resource.

```
/users/{admin_user}
```

Xác định phạm vi cho *resource route*

Tính năng ngầm gán ghép model theo phạm vi của Laravel có thể tự động xác định phạm vi lồng ghép sao cho các model con có được, được xác nhận là thuộc về model mẹ. Bằng cách sử dụng phương pháp xác định phạm vi khi tạo *nested resource*, bạn có thể bật tính năng tự động xác định phạm vi cũng như cho Laravel biết field nào tài nguyên con sẽ được truy xuất:

```
use App\Http\Controllers\PhotoCommentController;

Route::resource('photos.comments', PhotoCommentController::class)->scoped([
  'comment' => 'slug',
]);
```

Route trên sẽ làm cho một *nested resource* có phạm vi xác định có thể được truy cập với URI như bên dưới:

```
/photos/{photo}/comments/{comment:slug}
```

Khi sử dụng phương pháp ngầm gán ghép với khóa tuỳ biến làm tham số *nested route*,

Laravel sẽ tự động phân phạm vi truy vấn để truy xuất *nested model* bởi cha của nó bằng cách sử dụng các quy ước đoán tên mối quan hệ trên thành phần cha. Trong trường hợp này, giả định rằng `Photo` model có một mối quan hệ có tên là `comments` (số nhiều của tên tham số route) có thể được sử dụng để truy xuất model `Comment`.

Việt hóa resource URI

Mặc định, `Route::resource` sẽ tạo các URI tài nguyên bằng cách sử dụng các động từ trong tiếng Anh. Nếu bạn cần Việt hóa các động từ của hành động tạo và chỉnh sửa, bạn có thể sử dụng phương thức `Route::resourceVerbs`. Điều này có thể được thực hiện ngay từ lúc bắt đầu ứng dụng, nó trong phương thức `boot` của `App\Providers\RouteServiceProvider`:

```
/**
 * Define your route model bindings, pattern filters, etc.
 *
 * @return void
 */
public function boot()
{
  Route::resourceVerbs([
    'create' => 'tao',
    'edit' => 'sua',
  ]);

  // ...
}
```

Một khi các động từ đã được chỉnh sửa, thì việc đăng ký *resource route* như `Route::resource('fotos', PhotoController::class)` sẽ tạo ra các URI sau:

```
/fotos/tao

/fotos/{foto}/sua
```

Bổ sung resource controller

Nếu bạn cần thêm route bổ sung vào *resource controller* thay cho các *resource route* mặc định, thì bạn nên khai báo các route đó trước khi gọi phương thức `Route::resource`; nếu không, các route được tạo bởi phương thức `resource` có thể vô tình được ưu tiên hơn các route bổ sung của bạn:

```
use App\Http\Controller\PhotoController;

Route::get('/photos/popular', [PhotoController::class, 'popular']);
Route::resource('photos', PhotoController::class);
```

> Hãy nhớ giữ cho controller của bạn được tập trung. Nếu bạn nhận thấy cần các phương pháp bên ngoài các resource action thông thường, hãy xem xét chia controller của bạn thành hai, những controller nhỏ hơn.

Truyền thư viện vào controllers

Truyền bằng contructor

Service container của Laravel được sử dụng để có được tất cả các controller Laravel. Do đó, bạn có thể khai kiểu thư viện nào mà controller của bạn có thể cần trong contructor của nó. Các thư viện đã khai báo sẽ tự động tạo và đưa vào đối tượng controller:

```
<?php

namespace App\Http\Controllers;

use App\Repositories\UserRepository;

class UserController extends Controller
{
  /**
   * The user repository instance.
   */
  protected $users;

  /**
```

```
    * Create a new controller instance.
    *
    * @param  \App\Repositories\UserRepository  $users
    * @return void
    */
   public function __construct(UserRepository $users)
   {
     $this->users = $users;
   }
}
```

Truyền bằng phương thức

Ngoài phương pháp truyền bằng contructor, bạn cũng có thể khai kiểu các thư viện lên phương thức của controller. Có một trường hợp vận dụng khá phổ biến cho phương pháp này là đưa thư viện `Illuminate\Http\Request` vào các phương thức của controller:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class UserController extends Controller
{
  /**
    * Store a new user.
    *
    * @param  \Illuminate\Http\Request  $request
    * @return \Illuminate\Http\Response
    */
   public function store(Request $request)
   {
     $name = $request->name;

     //
   }
```

```
}
```

Nếu phương thức của controller của bạn cũng đang đợi đầu vào từ một tham số route, thì hãy liệt kê các đối số route ngay sau các thư viện. Ví dụ: nếu bạn khai báo route thì sẽ như sau:

```
use App\Http\Controllers\UserController;

Route::put('/user/{id}', [UserController::class, 'update']);
```

Bạn vẫn có thể khai kiểu `Illuminate\Http\Request` và truy cập tham số `id` của bạn bằng cách khai báo phương thức controller như sau:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class UserController extends Controller
{
  /**
   * Update the given user.
   *
   * @param  \Illuminate\Http\Request  $request
   * @param  string  $id
   * @return \Illuminate\Http\Response
   */
  public function update(Request $request, $id)
  {
    //
  }
}
```