Course - Laravel Framework

Facades

Xuyên xuốt tài liệu Laravel, bạn sẽ thấy các đoạn code ví dụ sử dụng các tính năng Laravel qua các Facades. Facade cung cấp giao diện tĩnh cho các class trong các service container của ứng dụng. Laravel chứa nhiều facades nhằm truy cập vào các tính năng của Laravel.

Tags: Facades

Xuyên xuốt tài liệu Laravel, bạn sẽ thấy các đoạn code ví dụ sử dụng các tính năng Laravel qua các *Facades*. Facade cung cấp giao diện tĩnh cho các class trong các service container của ứng dụng. Laravel chứa nhiều facades nhằm truy cập vào các tính năng của Laravel.

Facade đóng vai trò trung chuyển với các class hạ tầng trong service container. Cung ứng các cú pháp ngắn gọn tiện nghi mà vẫn đảm bảo tính thực dụng và linh hoạt so với các phương thức truyền thống. Đảm bảo mọi thứ trong khi bạn chưa thực sự hiểu hết cách thức facade vận hành bên dưới - chỉ cần theo đúng luồng hoạt động và tiếp tục tìm hiểu các chức năng khác của Laravel.

Tất cả các facade của Laravel đều được khai báo trong namespace `Illuminate\Support\Facades`. Do đó, chúng ta có thể dễ dàng truy cập vào một facade như sau:

```
use Illuminate\Support\Facades\Cache;
use Illuminate\Support\Facades\Route;

Route::get('/cache', function () {
    return Cache::get('key');
});
```

Xuyên suốt tài liệu Laravel, có nhiều ví dụ sẽ sử dụng facade để dẫn chứng cho các loại đặc trưng khác nhau của framework.

Hàm helper

Để bổ sung cho các facade, Laravel đưa ra một loạt các hàm helper nhằm để truy cập một cách dễ dàng vào các tính năng thông dụng trong Laravel. Các hàm helper này có thể tương tác với `view`, `resource`, `url`, `config` và vân vân. Do các hàm helper có phạm vi tổng thể, nên bạn không cần phải chèn bất kỳ class nào để sử dụng chúng.

```
use Illuminate\Support\Facades\Response;

Route::get('/users', function () {
    return Response::json([
        // ...
    ]);
});

Route::get('/users', function () {
```

```
    return response()->json([
        // ...
    ]);
});
```

Khi nào sử dụng facade

Laravel có rất nhiều lợi ích. Chúng cung cấp cú pháp ngắn gọn tiện nghi cho phép bạn sử dụng các tính năng của Laravel mà không cần phải nhớ dài dòng tên class được tiêm chèn hay được cấu hình bằng tay. Do cách sử dụng thống nhất với các phương thức PHP, nên chúng rất dễ dàng khi test.

Tuy nhiên, mối lo ngại chính của các facade là class "scope creep" - không quá chặt chẽ về phạm vi. Do facade dễ dàng sử dụng mà không đòi hỏi tiêm chèn, nên nó có thể dễ dàng để cho các class của bạn tiếp diễn và sử dụng nhiều facade trong một class duy nhất. Việc sử dụng Dependency Injection, sẽ có tiềm tàng làm cho constructor trở nên lớn hơn, các class của bạn sẽ trở nên quá lớn. Vì vậy, khi sử dụng facade, hãy chú ý đến kích thước của class sao cho phạm vi của nó tương đương với khả năng của nó. Nếu như class của bạn quá lớn, hãy chia nhỏ chúng ra thành nhiều class nhỏ hơn.

Facade và Dependency Injection

Một trong những lợi ích chính trong việc của dependency injection là khả năng tiếp cận phần thực thi của class được chèn.

```
use Illuminate\Support\Facades\Cache;

Route::get('/cache', function () {
    return Cache::get('key');
});
```

Khi test phương thức bằng facade, bạn có thể viết đoạn code sau để xác nhận phương thức `Cache::get` được gọi các tham số nào đó.

```
use Illuminate\Support\Facades\Cache;

/**
```

```
     * A basic functional test example.
     *
     * @return void
     */
    public function testBasicExample()
    {
        Cache::shouldReceive('get')
             ->with('key')
             ->andReturn('value');

        $response = $this->get('/cache');

        $response->assertSee('value');
    }
```

Facade và helper function

Laravel chứa một loạt các hàm helper cho phép thức hiện các lệnh thường dùng như tạo view, gọi event, phát đi job, và gửi đi phản hồi HTTP. Nhiều hàm helper thực hiện cùng nhiệm vụ với các facade tương ứng. Ví dụ, hàm helper sau và facade sau tương đương nhau.

```
return Illuminate\Support\Facades\View::make('profile');

return view('profile');
```

Vì không có sự khác biệt giữa facade và hàm helper, khi sử dụng hàm helper, bạn sẽ có thể test chúng với facade tương xứng. Ví dụ, hãy xem route sau.

```
Route::get('/cache', function () {
    return cache('key');
});
```

Trong đoạn code dưới đây, helper `Cache` sẽ gọi phương thức `get` trên class trong facade `Cache`. Do đó mà cho dù chúng ta sử dụng hàm helper, chúng ta có thể viết đoạn test sau để xác minh việc phương thức đó được gọi với tham số đã cho.

```php
use Illuminate\Support\Facades\Cache;

/**
 * A basic functional test example.
 *
 * @return void
 */
public function testBasicExample()
{
    Cache::shouldReceive('get')
         ->with('key')
         ->andReturn('value');

    $response = $this->get('/cache');

    $response->assertSee('value');
}
```

Nhìn chung facade và hàm helper có thể xem tương đương nhau khi thực hiện cùng nhiệm vụ nào đó.

Cách mà facade làm việc

Trong ứng dụng Laravel, một facade là một class sẽ cung cấp phương pháp truy cập vào một đối tượng trong container. Cơ cấu hoạt động chủ yếu dựa trên class `Facade`, các facade của Laravel và bất kỳ facade mở rộng nào do bạn tạo nên đều được dựa theo class `Illuminate\Support\Facades\Facade`.

Class dựa trên `Facade` sẽ sử dụng phương thức magic `__callStatic()` để hoản lại các cuộc gọi từ facade của bạn đến một đối tượng được resolve từ container. Trong ví dụ bên dưới, có một cuộc gọi được tạo để gọi hệ thống cache của Laravel. Xem chi tiết đoạn code bên dưới, có thể giả định là phương thức tĩnh `get` sẽ được gọi bằng class `Cache`;

```php
<?php

namespace App\Http\Controllers;
```

```php
use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\Cache;

class UserController extends Controller
{
    /**
     * Show the profile for the given user.
     *
     * @param  int  $id
     * @return Response
     */
    public function showProfile($id)
    {
        $user = Cache::get('user:'.$id);

        return view('profile', ['user' => $user]);
    }
}
```

Ở trên đỉnh của đoạn code ví dụ trên, chúng ta sẽ chèn facade `Cache` vào. class facade này hoạt động như một proxy trung chuyển đến phần thực thi của interface `Illuminate\Contracts\Cache\Factory`. Bất kỳ cuộc gọi nào chúng ta tạo ra bằng facade đều sẽ được truyền xuống đối tượng service cache bên dưới của ứng dụng Laravel.

Nếu bạn xem cụ thể code của class `Illuminate\Support\Facades\Cache`, thì bạn sẽ không thấy phương thức `get` nào trong đó.

```php
class Cache extends Facade
{
    /**
     * Get the registered name of the component.
     *
     * @return string
     */
    protected static function getFacadeAccessor() { return 'cache'; }
}
```

Trong khi đó, class `Cache` được mở rộng từ class `Facade` và khai báo phương thức `getFacadeAccessor()`, giá trị trả lại của phương thức này là tên của một ràng buộc service container. Khi người dùng tham chiếu đến phương thức tĩnh nào của facade `Cache`, thì Laravel sẽ resolve ràng buộc binding cache từ service container và sẽ chạy phương thức đã request (trong trường hợp này, `get`) với đối tượng đó.

Các facade real-time

Sử dụng facade real-time, giúp bạn tiếp xúc bất kỳ class nào trong ứng dụng của bạn khi nó là một facade. Để minh họa cách mà cái này có thể được sử dụng, đầu tiên hãy thử nghiệm đoạn code không sử dụng các facade real-time. Ví dụ, hãy giả định có một model Poscast trong đó có một phương thức publish. Tuy nhiên, để xuất bản podcast, chúng ta cần chèn một instance Publisher.

```php
<?php

namespace App\Models;

use App\Contracts\Publisher;
use Illuminate\Database\Eloquent\Model;

class Podcast extends Model
{
    /**
     * Publish the podcast.
     *
     * @param  Publisher  $publisher
     * @return void
     */
    public function publish(Publisher $publisher)
    {
        $this->update(['publishing' => now()]);

        $publisher->publish($this);
    }
}
```

Việc tiêm chèn phần thức thi một publisher vào trong phương thức này cho phép chúng ta dễ dàng kiểm tra phương thức một cách tách biệt vì thế chúng ta có thể giả lập publisher được tiêm chèn. Tuy nhiên nó đòi hỏi chúng ta truyền một publisher mỗi khi chúng ta gọi phương thức publish. Với việc sử dụng facade real-time, chúng ta có thể di trùy việc test trong khi không cần đòi hỏi truyền công khai một đối tượng `Publisher`. Để tạo ra một facade real-time, chỉ cần gắn phía trước namespace của class được chèn với `Facades`.

```php
<?php

namespace App\Models;

use Facades\App\Contracts\Publisher;
use Illuminate\Database\Eloquent\Model;

class Podcast extends Model
{
    /**
     * Publish the podcast.
     *
     * @return void
     */
    public function publish()
    {
        $this->update(['publishing' => now()]);

        Publisher::publish($this);
    }
}
```

Khi sử dụng facade real-time, phần thực thi publisher sẽ được giải quyết ra khỏi service container bằng cách chia tên interface hay class nào xuất hiện sau prefix Facades. Khi kiểm tra, chúng ta có thể sử dụng facade testing helper có sẵn của Laravel để thử gọi phương thức này.

```php
<?php

namespace Tests\Feature;
```

```php
use App\Models\Podcast;
use Facades\App\Contracts\Publisher;
use Illuminate\Foundation\Testing\RefreshDatabase;
use Tests\TestCase;

class PodcastTest extends TestCase
{
    use RefreshDatabase;

    /**
     * A test example.
     *
     * @return void
     */
    public function test_podcast_can_be_published()
    {
        $podcast = Podcast::factory()->create();

        Publisher::shouldReceive('publish')->once()->with($podcast);

        $podcast->publish();
    }
}
```

Bảng tham chiếu Facade

Bên dưới là một loạt các facade và phần thực thi bên dưới của chúng.

Facade	Class	Service Container Binding
App	Illuminate\Foundation\Application	`app`
Artisan	Illuminate\Contracts\Console\Kernel	`artisan`
Auth	Illuminate\Auth\AuthManager	`auth`
Auth (Instance)	Illuminate\Contracts\Auth\Guard	`auth.driver`
Blade	Illuminate\View\Compilers\BladeCompiler	`blade.compiler`
Broadcast	Illuminate\Contracts\Broadcasting\Factory	
Broadcast (Instance)	Illuminate\Contracts\Broadcasting\Broadcaster	
Bus	Illuminate\Contracts\Bus\Dispatcher	

Facade	Class	Service Container Binding
Cache	Illuminate\Cache\CacheManager	`cache`
Cache (Instance)	Illuminate\Cache\Repository	`cache.store`
Config	Illuminate\Config\Repository	`config`
Cookie	Illuminate\Cookie\CookieJar	`cookie`
Crypt	Illuminate\Encryption\Encrypter	`encrypter`
Date	Illuminate\Support\DateFactory	`date`
DB	Illuminate\Database\DatabaseManager	`db`
DB (Instance)	Illuminate\Database\Connection	`db.connection`
Event	Illuminate\Events\Dispatcher	`events`
File	Illuminate\Filesystem\Filesystem	`files`
Gate	Illuminate\Contracts\Auth\Access\Gate	
Hash	Illuminate\Contracts\Hashing\Hasher	`hash`
Http	Illuminate\Http\Client\Factory	
Lang	Illuminate\Translation\Translator	`translator`
Log	Illuminate\Log\LogManager	`log`
Mail	Illuminate\Mail\Mailer	`mailer`
Notification	Illuminate\Notifications\ChannelManager	
Password	Illuminate\Auth\Passwords\PasswordBrokerManager	`auth.password`
Password (Instance)	Illuminate\Auth\Passwords\PasswordBroker	`auth.password.broker`
Queue	Illuminate\Queue\QueueManager	`queue`
Queue (Instance)	Illuminate\Contracts\Queue\Queue	`queue.connection`
Queue (Base Class)	Illuminate\Queue\Queue	
Redirect	Illuminate\Routing\Redirector	`redirect`
Redis	Illuminate\Redis\RedisManager	`redis`
Redis (Instance)	Illuminate\Redis\Connections\Connection	`redis.connection`
Request	Illuminate\Http\Request	`request`
Response	Illuminate\Contracts\Routing\ResponseFactory	
Response (Instance)	Illuminate\Http\Response	
Route	Illuminate\Routing\Router	`router`
Schema	Illuminate\Database\Schema\Builder	
Session	Illuminate\Session\SessionManager	`session`
Session (Instance)	Illuminate\Session\Store	`session.store`
Storage	Illuminate\Filesystem\FilesystemManager	`filesystem`
Storage (Instance)	Illuminate\Contracts\Filesystem\Filesystem	`filesystem.disk`
URL	Illuminate\Routing\UrlGenerator	`url`
Validator	Illuminate\Validation\Factory	`validator`
Validator (Instance)	Illuminate\Validation\Validator	
View	Illuminate\View\Factory	`view`
View (Instance)	Illuminate\View\View	