Course - Laravel Framework

Service Provider

Service Provider chiếm giữ vị trí trung tâm của tất cả quá trình khởi động ứng dụng Laravel. Từ phần ứng dụng của riêng bạn cho tới các service cốt lõi của Laravel, tất cả đều được khởi động dựa trên service provider.

Tags: service provider, laravel

Service Provider chiếm giữ vị trí trung tâm của tất cả quá trình khởi động ứng dụng Laravel. Từ phần ứng dụng của riêng bạn cho tới các service cốt lõi của Laravel, tất cả đều được khởi động dựa trên service provider.

"Bootstrapped" trong Laravel có nghĩa là gì? - Nó có nghĩa chung chung là đăng ký. Nó đăng ký những thứ bao gồm như các ràng buộc service container, event listener, middleware và thậm chí là các route, service provider là nơi trung tâm để cấu hình ứng dụng Laravel.

Nếu bạn mở ra tập tin *config/app.php* bên trong Laravel, bạn sẽ nhìn thấy một mảng `providers`. Nó chứa các service provider mà sẽ được tải vào trong ứng dụng của bạn. Đương nhiên sẽ có một nhóm các service provider lõi của Laravel được liệt kê trong mảng này. Những provider này bao gồm các component như, email, queue, cache, và vân vân. Một vài trong những provider này sẽ "chờ" xem xét các request gửi đến có phù hợp hay không trước khi thực sự được tải vào trong ứng dụng.

Trong bài học tổng quan này, chúng ta sẽ xem qua phương pháp viết cho mình những service provider và đăng ký chúng với ứng dụng Laravel.

Viết Service Provider

Mọi service provider đều được viết từ class `Illuminate\Support\ServiceProvider`. Hầu hết các service provider đều có hai phương thức `register` và `boot`. Với phương thức `register`, bạn sẽ chỉ có thể ràng buộc mọi thứ vào service container, chứ không bao giờ có thể đăng ký bất kỳ event listener, routes, hay bất kỳ một cơ chế nào khác với phương thức `register`.

Lệnh Artisan CLI có thể tạo ra một provider mới thông qua lệnh `make:provider`.

```
php artisan make:provider RiakServiceProvider
```

Phương thức register

Như đã đề cập ở trên, với phương thức `register`, bạn sẽ chỉ có thể ràng buộc mọi thứ vào trong service container. Bạn sẽ không bao giờ đăng ký bất kỳ event listener, routes, hay bất kỳ cơ chế nào khác với phương thức `register`. Tuy nhiên, bạn cũng có thể sử dụng một service được cung cấp bởi service provider mà chưa từng được tải.

Bên trong service provider, bạn có thể truy cập thuộc tính `$app` để sử dụng service container.

```php
<?php

namespace App\Providers;

use App\Services\Riak\Connection;
use Illuminate\Support\ServiceProvider;

class RiakServiceProvider extends ServiceProvider
{
  /**
   * Register any application services.
   *
   * @return void
   */
  public function register()
  {
    $this->app->singleton(Connection::class, function ($app) {
      return new Connection(config('riak'));
    });
  }
}
```

Service provider này chỉ khai báo duy nhất một phương thức `register`, và sử dụng phương thức đó để khai báo một bản thực thi của `App\Services\Riak\Connection` trong service container. Nếu bạn chưa biết về service container thì hãy quay lại bài học trước.

Các thuộc tính `bindings` và `singletons`

Nếu service provider của bạn đăng ký nhiều binding, thì bạn có thể sử dụng thuộc tính `bindings` và `singletons` thay thế cho việc đăng ký bằng tay từng container.

```php
<?php

namespace App\Providers;

use App\Contracts\DowntimeNotifier;
```

```php
use App\Contracts\ServerProvider;
use App\Services\DigitalOceanServerProvider;
use App\Services\PingdomDowntimeNotifier;
use App\Services\ServerToolsProvider;
use Illuminate\Support\ServiceProvider;

class AppServiceProvider extends ServiceProvider
{
    /**
     * All of the container bindings that should be registered.
     *
     * @var array
     */
    public $bindings = [
        ServerProvider::class => DigitalOceanServerProvider::class,
    ];

    /**
     * All of the container singletons that should be registered.
     *
     * @var array
     */
    public $singletons = [
        DowntimeNotifier::class => PingdomDowntimeNotifier::class,
        ServerProvider::class => ServerToolsProvider::class,
    ];
}
```

Khi service provider được tải bởi framework, nó sẽ tự động kiểm tra những thuộc tính ràng buộc này và đăng ký các ràng buộc này.

Phương thức boot

Cái gì sẽ đăng ký view với các service provider? Đó chính là phương thức `boot`. Phương thức `boot` sẽ được gọi ngay sau khi tất cả các service provider được đăng ký.

```php
<?php
```

```php
namespace App\Providers;

use Illuminate\Support\Facades\View;
use Illuminate\Support\ServiceProvider;

class ComposerServiceProvider extends ServiceProvider
{
    /**
     * Bootstrap any application services.
     *
     * @return void
     */
    public function boot()
    {
        View::composer('view', function () {
            //
        });
    }
}
```

Sau khi đã đăng ký các service provider bạn sẽ có thể truy cập vào tất cả các service này bởi framework.

Tiêm chèn thư viện này vào phương thức `boot`

Bạn có thể khai báo kiểu các thư viện vào phương thức `boot` của service provider, service container sẽ tự động tiêm bất kỳ thư viện nào bạn cần.

```php
use Illuminate\Contracts\Routing\ResponseFactory;

/**
 * Bootstrap any application services.
 *
 * @param  \Illuminate\Contracts\Routing\ResponseFactory  $response
 * @return void
 */
public function boot(ResponseFactory $response)
```

```
{
    $response->macro('serialized', function ($value) {
        //
    });
}
```

Đăng ký provider

Tất cả các service provider được đăng ký trong tập tin cấu hình *config/app.php*. Tập tin này chứa một mảng `providers`, trong đó liệt kê các tên class của các service provider của bạn. Mặc định, nó sẽ có sẵn vài service provider cốt lõi trong Laravel. Những provider này chứa các component Laravel như, mail, queue, cache và những thứ khác.

Sẽ dễ dàng đăng ký các provider, bằng việc thêm nó vào mảng `provider` như sau.

```
'providers' => [
    // Other Service Providers

    App\Providers\ComposerServiceProvider::class,
],
```

Đăng ký tạm provider

Nếu bạn chỉ đăng ký tạm các binding trong service container, thì bạn có thể hoãn lại việc đăng ký cho tới khi một trong những ràng buộc này thực sự cần thiết. Việc trì hoãn này sẽ giúp cải thiện hiệu năng của ứng dụng của bạn, do nó không nhất thiết phải tải lên trong mọi request của bạn.

Laravel sẽ biên dịch và lưu trữ tất cả các service được cung ứng bởi các service providers tạm với tên của các class service này. Và sau đó chỉ khi bạn đang cố resolve một trong những service này thì Laravel mới thực sự tải service provider.

Để trì hoãn việc tải một provider, hãy thực thi interface `\Illuminate\Contracts\Support\DeferrableProvider` và khai báo phương thức `providers`. Phương thức `providers` sẽ trả lại các ràng buộc đã được đăng ký bởi provider.

```
<?php
```

```php
namespace App\Providers;

use App\Services\Riak\Connection;
use Illuminate\Contracts\Support\DeferrableProvider;
use Illuminate\Support\ServiceProvider;

class RiakServiceProvider extends ServiceProvider implements DeferrableProvider
{
    /**
     * Register any application services.
     *
     * @return void
     */
    public function register()
    {
        $this->app->singleton(Connection::class, function ($app) {
            return new Connection($app['config']['riak']);
        });
    }

    /**
     * Get the services provided by the provider.
     *
     * @return array
     */
    public function provides()
    {
        return [Connection::class];
    }
}
```