13 Jan 2016

TRABALHANDO COM REPOSITÓRIOS NO LARAVEL: IMPLEMENTANDO UMA INTERFACE E UTILIZANDO O CONTAINER

Nesse post fala-se sobre a implementação de interfaces e uso do Container do Laravel e conclui o post sobre o Repository Pattern

Atenção: se você não leu a primeira parte desse post, por favor, leia. Usando repositórios no Laravel**.**

No post anterior sobre repositórios falei sobre como criar uma camada entre o banco (ou o que quer que entregue dados para nós) e os controllers. Para isso, usamos os seguintes exemplos: `app/Http/Controllers/PostsController.php`e `app/Repositories/PostRepository.php` - para ver os códigos, é só clicar no link do post ali em cima.
No post mostrei como injetar a classe PostRepository - que é responsável por fazer as chamadas para o nosso Model - no nosso `PostsController`. Isso nos ajudou a ter mais controle sobre as nossas queries etc. No entanto, isso não nos permite, por exemplo, flexibilidade na hora de trocar de ORM ou banco. Para isso, precisamos criar uma `Interface` - algo que funciona como um contrato, dizendo a uma classe quais métodos ela deve implementar sem definir como esses métodos serão tratados, clique aqui para ler mais -, fazer com que o Laravel saiba qual classe usar quando essa `Interface` for chamada e aí sim injetar ela no `controller`. Dessa forma, podemos fazer modificações sem ter que mexer no controller. Para isso, vamos primeiro criar a Interface. Você pode colocar o arquivo onde quiser, eu o colocarei em `app/Repositories/Contracts`.

```php
<?php
// app/Repositories/Contracts/PostRepositoryInterface.php

namespace App\Repositories\Contacts;

interface PostRepositoryInterface
{
    public function find();

    public function findBy($att, $column);
}

?>
```

Bem simples: dizemos quais métodos a classe que implementa a interface `PostRepositoryInterface` deve ter - ela **precisa** ter esses métodos, ou você receberá um erro.

Agora, a gente tem que implementar essa interface no nosso `PostRepository`. Nosso antigo código era assim:

```php
<?php
// app/Repositories/PostRepository.php
namespace App\Repositories;

use App\Post;

class PostRepository
{
        protected $post;

        public function __construct(Post $post)
        {
```

```
                $this->post = $post;
        }

        public function find($id)
        {
                return $this->post->find($id);
        }

        public function findBy($att, $column)
        {
                return $this->post->where($att, $column)
        }
}
?>
```

Para implementar a nossa `PostRepositoryInterface`, basta usar o operador `implements` na declaração da classe. Temos, também, que importar a classe da interface. Fica assim:

```
<?php
// app/Repositories/PostRepository.php
namespace App\Repositories;

use App\Repositories\Contracts\PostRepositoryInterface;
use App\Post;

class PostRepository implements PostRepositoryInterface
{
        protected $post;

        public function __construct(Post $post)
        {
                $this->post = $post;
        }
```

```
        public function find($id)
        {
                return $this->post->find($id);
        }

        public function findBy($att, $column)
        {
                return $this->post->where($att, $column)
        }
}
?>
```

Muito simples! Agora, se você quisesse, poderia criar também outro repositório (usando doctrine, por exemplo) e chamá-lo de, sei lá, `PostRepositoryDoctrine` e também implementar essa interface. Dessa forma, teremos duas (ou mais) classes (repositórios) que implementam a mesma `PostRepositoryInterface`. Mas como nosso controller saberá qual repositório ele deve injetar? Simples: **ele não sabe**. Ao invés de injetarmos diretamente o nosso repositório, iremos injetar a interface e usaremos o Service Container do Laravel para decidir qual repositório usar - ou melhor, fazer a ligação entre a interface e a classe que deve ser usada. Vamos, primeiramente, definir isso. Para isso, abra o arquivo `app/Providers/AppServiceProvider.php`. No método `register`, iremos ligar a interface ao repositório usando o método `bind`. Mais ou menos assim:

```php
<?php

namespace App\Providers;

use Illuminate\Support\ServiceProvider;

class AppServiceProvider extends ServiceProvider
{
    /**
     * Bootstrap any application services.
     *
     * @return void
     */
    public function boot()
    {
        //
    }

    /**
     * Register any application services.
     *
     * @return void
     */
    public function register()
    {
        $this->app->bind('App\Repositories\Contracts\PostRepositoryInterfac
    }
}
?>
```

Nós basicamente dissemos ao Laravel que quando uma classe (no nosso caso, será o `PostController`) precisar da interface `PostRepositoryInterface`, o Laravel irá injetar a classe

`PostRepository`. Obviamente, escrevemos também o namespace de cada uma das classes.

Agora, de volta ao nosso `PostController`: iremos colocar a interface `PostRepositoryInterface` no construtor da classe e o Laravel irá "decidir" (conforme configuramos no `AppServiceProvider`) qual classe injetar, no nosso caso `PostRepository`.

```
<?php
// app/Http/Controllers/PostsController.php
namespace App\Http\Controllers;

// antigamente: use App\Repositories\PostRepository; agora não precisamos m
use App\Repositories\Contracts\PostRepositoryInterface;

class PostsController
{
        protected $post;

        public function __construct(PostRepositoryInterface $post)
        {
                $this->post = $post;
        }

        public function show($slug)
        {
                return $this->post->findBy('slug', $slug);
        }
}
?>
```

No final das contas, você acaba retornando uma coleção do Eloquent, então se você fosse trocar para outra ORM (Doctrine, por exemplo), ainda teria trabalho para ajeitar os métodos, já que você não mais retornaria os resultados da consulta ao banco como Eloquent. Muitos chamarão isso de overengineering e às vezes eu também considero.

Conforme explicado no post interior, os repositórios acabam sendo apenas uma forma de organizar o código (uma camada de abstração) para quem usa o Eloquent. Se você realmente quer usar repositórios da forma “certa” (apesar de que você provavelmente nunca vai sair do Eloquent, eu nem vejo porquê), é melhor dar uma olhada em algo como - novamente - o Doctrine.

Espero que eu tenha ajudado e quaisquer dúvidas favor deixar nos comentários. Se eu escrevi alguma besteira, seria legal também me corrigir :). Obrigado!