FORGE

Deploy Laravel and PHP applications on DigitalOcean, Linode, & AWS.

ADS VIA CARBON

Contracts

Introduction

Laravel's Contracts are a set of interfaces that define the core services provided by the framework. For example, a `Illuminate\Contracts\Queue\Queue` contract defines the methods needed for queueing jobs, while the `Illuminate\Contracts\Mail\Mailer` contract defines the methods needed for sending e-mail.

Each contract has a corresponding implementation provided by the framework. For example, Laravel provides a queue implementation with a variety of drivers, and a mailer implementation that is powered by SwiftMailer.

All of the Laravel contracts live in their own GitHub repository. This provides a quick reference point for all available contracts, as well as a single, decoupled package that may be utilized by package developers.

Contracts Vs. Facades

Laravel's facades and helper functions provide a simple way of utilizing Laravel's services without needing to type-hint and resolve contracts out of the service container. In most cases, each facade has an equivalent contract.

Unlike facades, which do not require you to require them in your class' constructor, contracts allow you to define explicit dependencies for your classes. Some developers prefer to explicitly define their dependencies in this way and therefore prefer to use contracts, while other developers enjoy the convenience of facades.

Most applications will be fine regardless of whether you prefer facades or contracts. However, if you are building a package, you should strongly consider using contracts since they will be easier to test in a package context.

When To Use Contracts

As discussed elsewhere, much of the decision to use contracts or facades will come down to personal taste and the tastes of your development team. Both contracts and facades can be used to create robust, well-tested Laravel applications. As long as you are keeping your class' responsibilities focused, you will notice very few practical differences between using contracts and facades.

However, you may still have several questions regarding contracts. For example, why use interfaces at all? Isn't using interfaces more complicated? Let's distill the reasons for using interfaces to the following headings: loose coupling and simplicity.

Loose Coupling

First, let's review some code that is tightly coupled to a cache implementation. Consider the following:

```php
<?php

namespace App\Orders;

class Repository
{
    /**
     * The cache instance.
     */
    protected $cache;

    /**
     * Create a new repository instance.
     *
```

```
     * @param  \SomePackage\Cache\Memcached  $cache
     * @return void
     */
    public function __construct(\SomePackage\Cache\Memcached $cache)
    {
        $this->cache = $cache;
    }

    /**
     * Retrieve an Order by ID.
     *
     * @param  int  $id
     * @return Order
     */
    public function find($id)
    {
        if ($this->cache->has($id)) {
            //
        }
    }
}
```

In this class, the code is tightly coupled to a given cache implementation. It is tightly coupled because we are depending on a concrete Cache class from a package vendor. If the API of that package changes our code must change as well.

Likewise, if we want to replace our underlying cache technology (Memcached) with another technology (Redis), we again will have to modify our repository. Our repository should not have so much knowledge regarding who is providing them data or how they are providing it.

Instead of this approach, we can improve our code by depending on a simple, vendor agnostic interface:

```
<?php

namespace App\Orders;

use Illuminate\Contracts\Cache\Repository as Cache;

class Repository
{
    /**
     * The cache instance.
     */
    protected $cache;

    /**
     * Create a new repository instance.
     *
     * @param  Cache  $cache
     * @return void
     */
    public function __construct(Cache $cache)
    {
        $this->cache = $cache;
    }
}
```

Now the code is not coupled to any specific vendor, or even Laravel. Since the contracts package contains no implementation and no dependencies, you may easily write an alternative implementation of any given contract, allowing you to replace your cache implementation without modifying any of your cache consuming code.

Simplicity

When all of Laravel's services are neatly defined within simple interfaces, it is very easy to determine the functionality offered by a given service. **The contracts serve as succinct documentation to the framework's features.**

In addition, when you depend on simple interfaces, your code is easier to understand and maintain. Rather than tracking down which methods are available to you within a large, complicated class, you can refer to a simple, clean interface.

How To Use Contracts

So, how do you get an implementation of a contract? It's actually quite simple.

Many types of classes in Laravel are resolved through the service container, including controllers, event listeners, middleware, queued jobs, and even route Closures. So, to get an implementation of a contract, you can just "type-hint" the interface in the constructor of the class being resolved.

For example, take a look at this event listener:

```php
<?php

namespace App\Listeners;

use App\Events\OrderWasPlaced;
use App\User;
use Illuminate\Contracts\Redis\Factory;

class CacheOrderInformation
{
    /**
     * The Redis factory implementation.
     */
    protected $redis;

    /**
     * Create a new event handler instance.
     *
     * @param  Factory  $redis
     * @return void
     */
    public function __construct(Factory $redis)
    {
        $this->redis = $redis;
    }

    /**
     * Handle the event.
     *
     * @param  OrderWasPlaced  $event
     * @return void
     */
    public function handle(OrderWasPlaced $event)
    {
        //
    }
}
```

When the event listener is resolved, the service container will read the type-hints on the constructor of the class, and inject the appropriate value. To learn more about registering things in the service container, check out its documentation.

Contract Reference

This table provides a quick reference to all of the Laravel contracts and their equivalent facades:

Contract	References Facade
Illuminate\Contracts\Auth\Access\Authorizable	
Illuminate\Contracts\Auth\Access\Gate	`Gate`
Illuminate\Contracts\Auth\Authenticatable	
Illuminate\Contracts\Auth\CanResetPassword	
Illuminate\Contracts\Auth\Factory	`Auth`
Illuminate\Contracts\Auth\Guard	`Auth::guard()`
Illuminate\Contracts\Auth\PasswordBroker	`Password::broker()`
Illuminate\Contracts\Auth\PasswordBrokerFactory	`Password`
Illuminate\Contracts\Auth\StatefulGuard	
Illuminate\Contracts\Auth\SupportsBasicAuth	
Illuminate\Contracts\Auth\UserProvider	
Illuminate\Contracts\Bus\Dispatcher	`Bus`
Illuminate\Contracts\Bus\QueueingDispatcher	`Bus::dispatchToQueue()`
Illuminate\Contracts\Broadcasting\Factory	`Broadcast`
Illuminate\Contracts\Broadcasting\Broadcaster	`Broadcast::connection()`
Illuminate\Contracts\Broadcasting\ShouldBroadcast	
Illuminate\Contracts\Broadcasting\ShouldBroadcastNow	
Illuminate\Contracts\Cache\Factory	`Cache`
Illuminate\Contracts\Cache\Lock	
Illuminate\Contracts\Cache\LockProvider	
Illuminate\Contracts\Cache\Repository	`Cache::driver()`
Illuminate\Contracts\Cache\Store	

Contract	References Facade
Illuminate\Contracts\Config\Repository	`Config`
Illuminate\Contracts\Console\Application	
Illuminate\Contracts\Console\Kernel	`Artisan`
Illuminate\Contracts\Container\Container	`App`
Illuminate\Contracts\Cookie\Factory	`Cookie`
Illuminate\Contracts\Cookie\QueueingFactory	`Cookie::queue()`
Illuminate\Contracts\Database\ModelIdentifier	
Illuminate\Contracts\Debug\ExceptionHandler	
Illuminate\Contracts\Encryption\Encrypter	`Crypt`
Illuminate\Contracts\Events\Dispatcher	`Event`
Illuminate\Contracts\Filesystem\Cloud	`Storage::cloud()`
Illuminate\Contracts\Filesystem\Factory	`Storage`
Illuminate\Contracts\Filesystem\Filesystem	`Storage::disk()`
Illuminate\Contracts\Foundation\Application	`App`
Illuminate\Contracts\Hashing\Hasher	`Hash`
Illuminate\Contracts\Http\Kernel	
Illuminate\Contracts\Mail\MailQueue	`Mail::queue()`
Illuminate\Contracts\Mail\Mailable	
Illuminate\Contracts\Mail\Mailer	`Mail`
Illuminate\Contracts\Notifications\Dispatcher	`Notification`
Illuminate\Contracts\Notifications\Factory	`Notification`
Illuminate\Contracts\Pagination\LengthAwarePaginator	
Illuminate\Contracts\Pagination\Paginator	
Illuminate\Contracts\Pipeline\Hub	
Illuminate\Contracts\Pipeline\Pipeline	
Illuminate\Contracts\Queue\EntityResolver	
Illuminate\Contracts\Queue\Factory	`Queue`
Illuminate\Contracts\Queue\Job	
Illuminate\Contracts\Queue\Monitor	`Queue`
Illuminate\Contracts\Queue\Queue	`Queue::connection()`
Illuminate\Contracts\Queue\QueueableCollection	
Illuminate\Contracts\Queue\QueueableEntity	
Illuminate\Contracts\Queue\ShouldQueue	
Illuminate\Contracts\Redis\Factory	`Redis`
Illuminate\Contracts\Routing\BindingRegistrar	`Route`
Illuminate\Contracts\Routing\Registrar	`Route`
Illuminate\Contracts\Routing\ResponseFactory	`Response`
Illuminate\Contracts\Routing\UrlGenerator	`URL`
Illuminate\Contracts\Routing\UrlRoutable	
Illuminate\Contracts\Session\Session	`Session::driver()`
Illuminate\Contracts\Support\Arrayable	
Illuminate\Contracts\Support\Htmlable	
Illuminate\Contracts\Support\Jsonable	
Illuminate\Contracts\Support\MessageBag	
Illuminate\Contracts\Support\MessageProvider	
Illuminate\Contracts\Support\Renderable	
Illuminate\Contracts\Support\Responsable	
Illuminate\Contracts\Translation\Loader	
Illuminate\Contracts\Translation\Translator	`Lang`
Illuminate\Contracts\Validation\Factory	`Validator`
Illuminate\Contracts\Validation\ImplicitRule	
Illuminate\Contracts\Validation\Rule	
Illuminate\Contracts\Validation\ValidatesWhenResolved	

Contract	References Facade
Illuminate\Contracts\Validation\Validator	`Validator::make()`
Illuminate\Contracts\View\Engine	
Illuminate\Contracts\View\Factory	`View`
Illuminate\Contracts\View\View	`View::make()`

Become a Laravel Partner

Laravel Partners are elite shops providing top-notch Laravel development and consulting. Each of our partners can help you craft a beautiful, well-architected project.

Our Partners

Laravel

Highlights

- Release Notes
- Getting Started
- Routing
- Blade Templates
- Authentication
- Authorization
- Artisan Console
- Database
- Eloquent ORM
- Testing

Resources

- Laracasts
- Laravel News
- Laracon
- Laracon EU
- Laracon AU
- Jobs
- Certification
- Forums

Partners

- Vehikl
- Tighten Co.
- Kirschbaum
- Byte 5
- 64Robots
- Cubet
- DevSquad
- Ideil
- Cyber-Duck
- ABOUT YOU
- Become A Partner

Ecosystem

- Vapor
- Forge
- Envoyer
- Horizon
- Lumen
- Nova
- Echo
- Valet
- Mix
- Spark
- Cashier
- Homestead
- Dusk
- Passport
- Scout
- Socialite

Laravel is a web application framework with expressive, elegant syntax. We believe development must be an enjoyable and creative experience to be truly fulfilling. Laravel attempts to take the pain out of development by easing common tasks used in most web projects.

Laravel is a Trademark of Taylor Otwell.
Copyright © 2011-2019 Laravel LLC.