설계 컨셉

# 1. Request-요청 라이프사이클

## 1.1.Introduction

우리가 응용 프로그램을 프레임워크를 통하여 개발할때, 프레임워크가 어떻게 내부적으로 동작을하고 흘러가는지에 대해서 아는 것이 중요합니다.

프레임워크를 이용하여 애플리케이션 개발을 할때에도 크게 다르지 않습니다. 프레임워크가 -어떻게 동작하는지 원리를 이해한다면 이를 보다 잘 사용하여 좋은 응용프로그램을 개발할 수 있기 때문 입니다.

이 문서의 목적은 어떻게 라라벨이 동작하는지 높은 수준의 개념을 잘 설명하는 것입니다. 전체 프레임워크를 더 잘 알게 됨으로써 모든 것을 덜 "혼란스러워" 할 수 있으며, 애플리케이션을 구성하는 데 좀 더 자신감을 가질 수 있습니다. 지금 당장 모든 것들을 이해하지 못하더라도 괜찮습니다. 그냥 무엇을 하고 있는지에 대한 기본적인 이해를 얻기 위해 노력하십시오, 여러분이 다른 문서의 여러 부분들을 살펴 볼 수록 지식이 더 쌓일 것입니다.

## 1.2.라이프사이클 개요

### 1.2.1 첫번째

라라벨 애플리케이션의 **모든 요청에 대한 시작점**은 `public/index.php` 파일입니다. 웹서버 (Apache / Nginx)의 설정에 따라 모든 요청은 이 파일에 전달됩니다. `index.php` 파일은 그다지 많은 코드를 가지고 있지 않습니다. 대신 프레임워크의 나머지 부분들을 **로딩하기 위한 시작점**이 됩니다.

`index.php` 파일은 컴포저가 생성 한 오토로더 정의를 로딩합니다. 그리고, `bootstrap/app.php` 스크립트에서 라라벨 애플리케이션의 인스턴스를 가져옵니다. **라라벨 자신의 첫 번째 동작은 [서비스 컨테이너](https://laravel.kr/docs/8.x/container) 인스턴스를 생성**하는 것입니다.

### 1.2.2 HTTP / Console 커널

다음으로 애플리케이션이 시작된 유형에 따라 전송된 요청을 **HTTP 커널**이나 **콘솔 커널** 둘 중 하나로 보냅니다. 이 두가지의 커널은 모든 요청의 흐름 중심에서 작동하게 됩니다.

여기에서는 `app/Http/Kernel.php` 에있는 HTTP 커널에 초점을 맞춰 봅시다.

HTTP 커널은 `Illuminate\Foundation\Http\Kernel` 클래스를 상속하고 있으며, 요청을 실행하기 전에 처리되는 `bootstrappers` (시작 코드)의 배열을 정의하고 있습니다.

이 시작 코드들은 에러 처리, 로그 설정, [애플리케이션 동작 환경의 감지](https://laravel.kr/docs/8.x/configuration#environment-configuration) 등 실제로 요청이 처리되기 전에 수행해야 되는 작업들을 의미합니다.

또한 HTTP 커널은 애플리케이션에서 요청이 처리되기 전에 통과해야하는 HTTP [미들웨어](https://laravel.kr/docs/8.x/middleware)의 목록을 정의하고 있습니다. 이 미들웨어들은 [HTTP 세션](https://laravel.kr/docs/8.x/session)을 읽고/쓰고, 애플리케이션이 유지 관리 모드인지 확인하고, [CSRF 토큰을 확인 하는](https://laravel.kr/docs/8.x/csrf) 작업들을 처리합니다.

HTTP 커널의 `handle` 메소드의 사용법은 매우 간단합니다. 단순하게는 `Request` 를 받고 `Response`를 반환합니다. 커널을 애플리케이션 전체를 나타내는 하나의 큰 블랙 박스라고 생각해봅시다. HTTP 요청이 입력되면 HTTP 응답이 반환됩니다.

#### 서비스 프로바이더

커널 부팅(부트스트래핑) 과정의 가장 중요한 것 중의 하나는 **애플리케이션의 [서비스 프로바이더](https://laravel.kr/docs/8.x/providers)를 로딩**하는 것입니다. 애플리케이션의 모든 서비스 프로바이더는 `config/app.php` 파일의 `providers` 배열에 설정되어 있습니다. 먼저, 모든 서비스 프로바이더의 `register` 메소드가 호출되고, 이후에 등록 된 모든 서비스 프로바이더의 `boot` 메소드가 호출되어 집니다.

서비스 프로바이더는 프레임워크의 데이터베이스, 큐, validation, 라우팅 컴포넌트와 같은 다양한 컴포넌트의 부트스트래핑(부팅과 같은 기초 작업들)의 처리를 책임집니다. 프레임워크가 제공하는 모든 기능을 초기화 하고 설정하는 것으로, 서비스 프로바이더는 라라벨의 부팅(부트스트래핑) 과정에서 가장 중요한 기능이라고 할 수 있습니다.

#### 요청 처리-디스패칭

애플리케이션이 부팅(부트스트래핑)되고 모든 서비스 프로바이더가 등록된 후, `Request`는 라우터 처리를 위해서 전달될 것입니다. 라우터는 라우팅 또는 컨트롤러로 요청-request을 전달할뿐만 아니라, 임의의 특정 라우트에 지정된 미들웨어도 실행합니다.

## 1.3.서비스 프로바이더

서비스 프로바이더는 라라벨 애플리케이션의 부팅(부트스트래핑) 단계의 주요한 핵심입니다. 애플리케이션의 인스턴스가 생성되고, 서비스 프로바이더가 등록된후 부트스트래핑 과정을 마친 프로그램이 요청을 처리합니다. 매우 간단합니다!

라라벨 애플리케이션이 어떻게 구성되어 있는지, 서비스 프로바이더를 통해 부트스트랩되는 과정을 구체적으로 이해하는 것은 매우 중요합니다. 여러분의 애플리케이션을 위한 기본 서비스 프로바이더는 `app/Providers` 디렉토리에 있습니다.

기본적으로 `AppServiceProvider` 는 거의 비어 있습니다. 이 프로바이더는 여러분의 고유한 부트스트래핑과 서비스 컨테이너 바인딩 코드를 추가하기 위한 곳입니다. 보다 큰 애플리케이션의 경우, 보다 세부적인 유형으로 구분된 종류별로 서비스 프로바이더를 만들 수도 있습니다.

**2.서비스 컨테이너**

**2.1.시작하기**

라라벨의 서비스 컨테이너는 클래스의 의존성을 관리하고 **의존성을 주입**하는 강력한 도구 입니다. 의존성 주입이라는 멋진 말의 의미는 다음과 같습니다. 클래스간의 의존성은 클래스 생성될 때 또는 경우에 따라 "setter" 메소드에 의해서 "주입" 된다는 의미입니다.

간단한 예제를 들어 봅시다.

<?php  
​  
namespace App\Http\Controllers;  
​  
use App\Http\Controllers\Controller;  
use App\Repositories\UserRepository;  
use App\Models\User;  
​  
class UserController extends Controller  
{  
  /\*\*  
    \* The user repository implementation.  
    \*  
    \* @var UserRepository  
    \*/  
  protected $users;  
​  
  /\*\*  
    \* Create a new controller instance.  
    \*  
    \* @param UserRepository $users  
    \* @return void  
    \*/  
  public function \_\_construct(UserRepository $users)  
  {  
      $this->users = $users;  
  }  
​  
  /\*\*  
    \* Show the profile for the given user.  
    \*  
    \* @param int $id  
    \* @return Response  
    \*/  
  public function show($id)  
  {  
      $user = $this->users->find($id);  
​  
      return view('user.profile', ['user' => $user]);  
  }  
}

이 예제에서 UserController는 데이터 소스로부터 사용자를 조회할 필요가 있습니다. 따라서 우리는 사용자를 조회할 수 있는 서비스를 **주입** 할 것입니다.

여기에서는, UserRepository 가 [Eloquent](https://laravel.kr/docs/8.x/eloquent)를 사용하여 데이터베이스로부터 사용자 정보를 조회합니다. repository 가 주입되었기 때문에, 원하는 경우 손쉽게 다른 구현 객체로 변경할 수 있습니다. 또한 애플리케이션을 테스트할 때 손쉽게 "목킹" 하거나, 더미 UserRepository 구현체를 생성할 수도 있습니다.

라라벨 서비스 컨테이너를 깊이 이해하는 것은 강력하고 큰 애플리케이션을 구축할 때나 라라벨 코어에 공헌하기 위해서 아주 중요한 부분입니다.

**2.2.바인딩**

**기본적인 바인딩**

대부분의 **서비스 컨테이너** 바인딩들은 [서비스 프로바이더](https://laravel.kr/docs/8.x/providers) 내에서 등록됩니다. 따라서 이러한 모든 예제들은 해당 컨텍스트에서 컨테이너를 사용하는 데모가 될 것입니다.

*{tip} 특정 인터페이스에 대한 의존성이 없을 때에는 컨테이너에 클래스를 바인딩 할 필요는 없습니다. 이러한 객체들은 리플랙션에 의해서 자동으로 의존성이 해결되기 때문에, 컨테이너가 각각의 객체들이 어떻게 생성될지 알 필요는 없습니다.*

**간단한 바인딩**

서비스 프로바이더 안에서는 항상 $this->app 속성을 통해서 컨테이너 인스턴스에 접근 할 수 있습니다. 또한 bind 메소드를 사용하여 클래스나 인터페이스 이름에 대한 의존성을 우리가 원하는 클래스의 인스턴스를 반환하는 Closure를 등록하여 바인딩 할 수 있습니다.

$this->app->bind('HelpSpot\API', function ($app) {  
  return new \HelpSpot\API($app->make('HttpClient'));  
});

클로저에서 컨테이너 자신을 인자로 전달 받고 있다는 것에 유의하십시오. 이를 통해서 연결된 객체의 의존성 문제를 위해서 컨테이너 자신을 사용할 수 있습니다.

**싱글톤으로 바인딩하기**

singleton 메소드로 클래스나 인터페이스를 바인딩 하면 컨테이너는 **한 번만 해당 의존성을 해결**합니다. 싱글톤 바인딩으로 의존성이 해결되면, 컨테이너의 다른 부분에서 호출될 때 **동일한 객체 인스턴스가 반환**될 것입니다.

$this->app->singleton('HelpSpot\API', function ($app) {  
  return new \HelpSpot\API($app->make('HttpClient'));  
});

**인스턴스를 바인딩하기**

instance 메소드를 사용하여 이미 존재하는 **객체의 인스턴스를 컨테이너에 바인딩** 할 수 있습니다. 이후 컨테이너에서 호출이 될 때는 매번 주어진 인스턴스가 반환됩니다.

$api = new \HelpSpot\API(new HttpClient);  
​  
$this->app->instance('HelpSpot\API', $api);

**인터페이스에 구현객체 바인딩하기**

서비스 컨테이너의 강력한 기능 중 하나는 주어진 **구현 객체**에 인터페이스를 바인딩 할 수 있다는 것입니다. 예를 들어 EventPusher 인터페이스와 RedisEventPusher 구현이 있다고 가정해 보겠습니다. 이 인터페이스를 구현한 RedisEventPusher 객체를 구성한 뒤에 이 객체를 다음과 같이 서비스 컨테이너에 등록할 수 있습니다.

$this->app->bind(  
  'App\Contracts\EventPusher',  
  'App\Services\RedisEventPusher'  
);

이것은 구문은 EventPusher 인터페이스의 구현 객체가 필요할 때 컨테이너가 RedisEventPusher 을 주입해준다는 것을 말합니다. 이제 우리는 EventPusher 인터페이스에 대한 타입을 생성자에 지정하면 어디에서라도 서비스 컨테이너가 의존성을 주입해줍니다.

use App\Contracts\EventPusher;  
​  
/\*\*  
\* Create a new class instance.  
\*  
\* @param EventPusher $pusher  
\* @return void  
\*/  
public function \_\_construct(EventPusher $pusher)  
{  
  $this->pusher = $pusher;  
}

**문맥에 따른 조건적 바인딩**

때때로 동일한 인터페이스에 대한 2가지 구현 객체가 있고, 각각의 클래스마다 다른 구현 객체를 전달하고자 할 수도 있습니다. 예를 들어 각각의 컨트롤러가 다른 Illuminate\Contracts\Filesystem\Filesystem [contract](https://laravel.kr/docs/8.x/contracts) 구현체에 의존한다면, 라라벨은 간단하고 유연한 인터페이스를 통해서 다음 행동을 정의합니다.

use App\Http\Controllers\PhotoController;  
use App\Http\Controllers\UploadController;  
use App\Http\Controllers\VideoController;  
use Illuminate\Contracts\Filesystem\Filesystem;  
use Illuminate\Support\Facades\Storage;  
​  
$this->app->when(PhotoController::class)  
        ->needs(Filesystem::class)  
        ->give(function () {  
            return Storage::disk('local');  
        });  
​  
$this->app->when([VideoController::class, UploadController::class])  
        ->needs(Filesystem::class)  
        ->give(function () {  
            return Storage::disk('s3');  
        });

**기본 타입 바인딩**

때로는, 클래스가 주입되는 클래스들을 받아들일 수도 있지만, 정수형과 같은 기본 타입의 값들을 주입 할 필요가 있을 수도 있습니다. 여러분은 손쉽게 문맥에 따라 조건적 바인딩을 통해서 클래스가 필요한 값을 주입할 수 있습니다.

$this->app->when('App\Http\Controllers\UserController')  
        ->needs('$variableName')  
        ->give($value);

때때로 클래스는 태그 된 인스턴스의 배열에 의존 할 수 있습니다. giveTagged 메소드를 사용하면 해당 태그로 모든 컨테이너 바인딩을 쉽게 삽입 할 수 있습니다.

$this->app->when(ReportAggregator::class)

->needs('$reports')

->giveTagged('reports');

**Typed Variadic 바인딩**

때때로 가변 생성자 인수를 사용하여 유형이 지정된 객체의 배열을 받는 클래스가 있을 수 있습니다.

class Firewall

{

protected $logger;

protected $filters;

public function \_\_construct(Logger $logger, Filter ...$filters)

{

$this->logger = $logger;

$this->filters = $filters;

}

}

컨텍스트 바인딩을 사용하면 해결 된 Filter 인스턴스의 배열을 반환하는 Closure와 함께 give 메서드를 제공하여 이 종속성을 해결할 수 있습니다.

$this->app->when(Firewall::class)

->needs(Filter::class)

->give(function ($app) {

return [

$app->make(NullFilter::class),

$app->make(ProfanityFilter::class),

$app->make(TooLongFilter::class),

];

});

편의를 위해 Firewall에 Filter 인스턴스가 필요 할 때마다 컨테이너에서 확인 할 클래스 이름 배열을 제공 할 수도 있습니다.

$this->app->when(Firewall::class)

->needs(Filter::class)

->give([

NullFilter::class,

ProfanityFilter::class,

TooLongFilter::class,

]);

**가변 태그 종속성**

때때로 클래스는 주어진 클래스 (Report ... $reports)로 타입 힌트 된 가변 종속성을 가질 수 있습니다. needs 및 giveTagged 메소드를 사용하면 주어진 종속성에 대해 해당 태그로 모든 컨테이너 바인딩을 쉽게 삽입 할 수 있습니다.

$this->app->when(ReportAggregator::class)

->needs(Report::class)

->giveTagged('reports');

**태깅**

가끔은, 바인딩의 특정 "카테고리" 전체에 대한 의존성 해결을 해야 할 때도 있습니다. 예를 들어, 서로 다른 Report 인터페이스의 구현 객체를 포함하는 배열을 전달받는 보고서 수집기를 개발하고 있다고 해봅시다. Report 구현 객체를 등록한 뒤에, tag 메소드를 사용하여 태그를 달 수 있습니다.

$this->app->bind('SpeedReport', function () {

//

});

$this->app->bind('MemoryReport', function () {

//

});

$this->app->tag(['SpeedReport', 'MemoryReport'], 'reports');

서비스에 태그가 붙으면 tagged 메소드를 사용하여 손쉽게 의존성을 해결할 수 있습니다.

$this->app->bind('ReportAggregator', function ($app) {

return new ReportAggregator($app->tagged('reports'));

});

**바인딩 확장**

extend 메소드로 서비스의 의존성을 수정할 수 있습니다. 예를 들어, 서비스의 의존성이 해결되었을 때, 서비스를 꾸미거나(decorate) 혹은 설정하는 위한 추가 코드를 실행할 수 있습니다. 클로저는 해결중인 서비스와 컨테이너 인스턴스를 입력 받습니다.

$this->app->extend(Service::class, function ($service, $app) {

return new DecoratedService($service);

});

**2.3.의존성 해결**

**make 메소드**

컨테이너 밖에서 클래스 인스턴스에 대한 의존성을 해결하기 위해서 make 메소드를 사용할 수 있습니다. make 메소드는 의존성 해결을 위해 여러분이 원하는 클래스나 인터페이스에 대한 이름을 전달받습니다.

$api = $this->app->make('HelpSpot\API');

$app 변수에 대한 접근을 가지고 있지 않은 코드에 위치하고 있다면, 글로벌 resolve 헬퍼 함수를 사용할 수 있습니다.

$api = resolve('HelpSpot\API');

클래스의 의존성이 컨테이너를 통해서 해결될 수 없다면, makeWith 메소드에 관련된 인자를 배열로 전달할 수도 있습니다.

$api = $this->app->makeWith('HelpSpot\API', ['id' => 1]);

**자동 주입**

앞서 이야기한 방법과 다르게, 그리고 가장 중요한 것은 [컨트롤러](https://laravel.kr/docs/8.x/controllers), [이벤트 리스너](https://laravel.kr/docs/8.x/events), [미들웨어](https://laravel.kr/docs/8.x/middleware) 등 을 포함하여 클래스의 생성자에 "타입-힌트" 를 선언함으로써 컨테이너가 의존성을 해결할 수 있도록 하는 것입니다. 또한 [queued jobs](https://laravel.kr/docs/8.x/queues)의 handle 메소드에도 종속성을 입력할 수 있습니다. 실제로 이 방법이 개발에서 컨테이너에 의해서 객체의 의존성을 해결할 때 가장 많이 사용되는 방법입니다.

예를 들어 컨트롤러의 생성자에서 타입 힌트로 지정된 Repository를 정의했다고 가정해 보겠습니다. 해당 Repository는 자동으로 의존성이 해결되어 클래스에 주입될 것입니다.

<?php

namespace App\Http\Controllers;

use App\Models\Users\Repository as UserRepository;

class UserController extends Controller

{

/\*\*

\* The user repository instance.

\*/

protected $users;

/\*\*

\* Create a new controller instance.

\*

\* @param UserRepository $users

\* @return void

\*/

public function \_\_construct(UserRepository $users)

{

$this->users = $users;

}

/\*\*

\* Show the user with the given ID.

\*

\* @param int $id

\* @return Response

\*/

public function show($id)

{

//

}

}

**2.4.컨테이너 이벤트**

서비스 컨테이너는 객체의 **의존성 해결**을 수행할 때마다 **이벤트를 발생**시킵니다. resolving 메소드를 사용하여 이 이벤트들에 대응할 수 있습니다.

$this->app->resolving(function ($object, $app) {

// Called when container resolves object of any type...

});

$this->app->resolving(\HelpSpot\API::class, function ($api, $app) {

// Called when container resolves objects of type "HelpSpot\API"...

});

보시다시피, 의존성이 해결된 객체가 콜백에 전달되어, 최종적으로 객체를 필요로 하는 **대상에 전달하기 전에 추가적으로 객체의 속성을 설정**할 수 있습니다.

**2.5.PSR-11**

라라벨의 서비스 컨테이너는 [PSR-11](https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-11-container.md) 인터페이스를 구현합니다. 따라서, PSR-11 인터페이스를 타입힌트 하여 라라벨의 컨테이너 인스턴스에 접근이 가능합니다.

use Psr\Container\ContainerInterface;

Route::get('/', function (ContainerInterface $container) {

$service = $container->get('Service');

//

});

지정된 식별자를 해결할 수 없는 경우는 예외가 던져집니다. 식별자가 바인드 되지 않은 경우 예외는 Psr\Container\NotFoundExceptionInterface 인스턴스입니다. 식별자가 바인드 되었지만 해결할 수 없는 경우에는 Psr\Container\ContainerExceptionInterface 인스턴스가 던져집니다.

# 3.서비스 프로바이더

## 3.1.시작하기

서비스 프로바이더는 라라벨 애플리케이션의 부팅(부트스트래핑)의 가장 핵심이라고 할 수 있습니다. 여러분의 애플리케이션과 마찬가지로 라라벨의 모든 코어 서비스는 서비스 프로바이더를 통해서 부트스트래핑 됩니다.

### 부트스트래핑이란?

그런데 "부트스트래핑" 이란 과연 무엇을 의미하는 것일까요?

일반적으로는 서비스 컨테이너에 바인딩을 등록하는 것을 포함해서 이벤트 리스너, 미들웨어 그리고 라우트등을 **등록** 하는 것을 의미합니다.

서비스 프로바이더는 애플리케이션 구성의 핵심입니다.

라라벨에 포함되어 있는 config/app.php 파일을 열어 본다면 providers 배열을 볼 수 있을 것입니다.

배열 안에 있는 모든 서비스 프로바이더 클래스가 애플리케이션에 로드됩니다. 대부분의 프로바이더는 "지연된" 프로바이더입니다.

이 말은 모든 요청에 대해서 반드시 로드되지 않고 실제로 필요할 때에 로드 된다는 것을 의미합니다.

여기에서는 서비스 프로바이더를 작성하는 방법과 라라벨 애플리케이션에 등록하는 방법을 배워봅시다.

## 3.2. 서비스 프로바이더 작성하기

라라벨의 서비스 프로바이더는 Illuminate\Support\ServiceProvider 클래스를 상속받습니다.

상속받은 서비스 프로바이더는 register 와 boot 2개의 메소드를 가지고 있습니다.

* register 메소드는 [**서비스 컨테이너**](https://laravel.kr/docs/8.x/container)**에 등록만을 하도록 합니다**.
* register 메소드는 다른 이벤트 리스너나 라우트 또는 기타 기능의 일부등을 등록하지 말아야 합니다.

아티즌 CLI에서는 make:provider 명령어를 통해서 새로운 프로바이더를 생성할 수 있습니다.

php artisan make:provider RiakServiceProvider

### Register 메소드

이미 설명한 바와 같이, register 메소드 안에서는, [서비스 컨테이너](https://laravel.kr/docs/8.x/container)에 무언가를 바인딩 하는 작업만을 수행 해야 합니다.

여러분은 register 메소드 안에서 어떠한 이벤트 리스너, 라우트, 또는 다른 기능들에 대한 코드를 넣어서는 절대 안됩니다. 그렇지 않으면, 서비스 프로바이더가 아직 로드 하지 않은 서비스를 의도치 않게 사용해 버리고 말것입니다.

기본적인 서비스 프로바이더를 살펴보겠습니다.

서비스 프로바이더의 메소드 안에서 언제든지 $app 속성을 사용할 수 있으며, 이를 통해 서비스 컨테이너에 접근할 수 있습니다.

<?php  
​  
namespace App\Providers;  
​  
use Illuminate\Support\ServiceProvider;  
use Riak\Connection;  
​  
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'));  
      });  
  }  
}

이 서비스 프로바이더는 register 메소드만 정의되어 있습니다.

그리고 이 메소드를 통해서 서비스 컨테이너에 Riak\Connection 구현 객체를 정의하고 있습니다.

서비스 컨테이너가 어떻게 작동하는지 이해하지 못하겠다면, [컨테이너 문서](https://laravel.kr/docs/8.x/container)를 확인하십시오.

#### bindings 과 singletons 속성

서비스 프로바이더가 동일한 (simple) 바인딩 여러개를 등록한다면, 각각의 컨테이너 바인딩을 일일이 등록하는 대신에 bindings 와 singletons 속성을 사용할 수 있습니다. 프레임워크에서 서비스 프로바이더가 로드 될 때, 이 속성들을 자동으로 체크하고 바인딩을 등록합니다.

<?php  
​  
namespace App\Providers;  
​  
use App\Contracts\DowntimeNotifier;  
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,  
  ];  
}

### Boot 메소드

그럼 이제 서비스 프로바이더 안에서 [뷰 컴포저](https://laravel.kr/docs/8.x/views#view-composers)를 등록할 필요가 있다면 어떻게 해야 할까요?

그런 작업은 boot 메소드 안에서 해야합니다. **이 메소드는 모든 다른 서비스 프로바이더들이 등록된 이후에 호출됩니다** 즉, 프레임 워크에 의해 등록된 다른 모든 서비스들에 액세스 할 수 있다는 것을 의미합니다.

<?php  
​  
namespace App\Providers;  
​  
use Illuminate\Support\ServiceProvider;  
​  
class ComposerServiceProvider extends ServiceProvider  
{  
  /\*\*  
    \* Bootstrap any application services.  
    \*  
    \* @return void  
    \*/  
  public function boot()  
  {  
      view()->composer('view', function () {  
          //  
      });  
  }  
}

#### Boot 메소드의 의존성 주입

여러분은 서비스 프로바이더의 boot 메소드에서 의존성 주입을 위해서 타입 힌트를 사용할 수 있습니다. [서비스 컨테이너](https://laravel.kr/docs/8.x/container)는 자동으로 필요한 의존 객체를 주입할 것입니다.

use Illuminate\Contracts\Routing\ResponseFactory;  
​  
public function boot(ResponseFactory $response)  
{  
  $response->macro('caps', function ($value) {  
      //  
  });  
}

## 3.3.프로바이더 등록하기

모든 서비스 프로바이더들은 config/app.php 설정 파일에 등록되어 있습니다.

이 파일에는 서비스 프로바이더들의 클래스 이름을 나열하고 등록할 수 있는 providers 배열이 포함되어 있습니다. 기본적으로는 라라벨의 코어 서비스 프로바이더들이 배열에 나열되어 있습니다. 이 프로바이더들이 라라벨의 메일러, 큐, 캐시등과 같은 핵심적인 컴포넌트들을 부트스트랩핑 하게 됩니다.

여러분의 프로바이더들을 등록하려면 이 배열에 추가 하면 됩니다.

'providers' => [  
  // Other Service Providers  
​  
  App\Providers\ComposerServiceProvider::class,  
],

## 3.4.지연(deferred) 프로바이더

만약 여러분의 프로바이더가 **단지** [서비스 컨테이너](https://laravel.kr/docs/8.x/container)에 바인딩을 등록하기만 한다면, 등록된 바인딩이 실제로 필요할때까지 등록 자체를 지연(deferred) 시킬 수 있습니다. 이러한 프로바이더 로딩의 지연(deferred)은 모든 요청에 프로바이더를 파일 시스템에서 로드하지 않으므로 애플리케이션의 성능을 향상시킬 것입니다.

라라벨은 지연된 서비스 프로바이더가 제공하는 모든 서비스 목록과 해당 서비스 프로바이더 클래스 이름을 컴파일하고 저장합니다. 그런 다음 이러한 서비스들 중 하나를 의존성 해결하려고 할 때에만, 라라벨이 서비스 프로바이더를 로드합니다.

프로바이더를 지연(defer) 로딩 하려면, \Illuminate\Contracts\Support\DeferrableProvider 인터페이스를 구현하고 provides 메소드를 정의하면 됩니다. provides 메소드는 프로바이더에 의해서 바인딩이 등록된 서비스 컨테이너를 리턴해야 합니다.

<?php  
​  
namespace App\Providers;  
​  
use Illuminate\Contracts\Support\DeferrableProvider;  
use Illuminate\Support\ServiceProvider;  
use Riak\Connection;  
​  
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];  
  }  
}

# 4.파사드

## 4.1.시작하기

파사드는 애플리케이션의 [서비스 컨테이너](https://laravel.kr/docs/8.x/container)에서 사용가능한 클래스들에 대한 **"정적" 인터페이스를 제공**합니다.

라라벨은 대부분의 라라벨의 기능에 엑세스하는 많은 파사드들을 제공합니다. 라라벨의 파사드 는 서비스 컨테이너에 등록된 클래스들에 대한 일종의 "**정적 프록시**" 역할을 수행하는데, 이를 통해서 전통적인 정적 메소드 보다 간결한 문법과 테스트의 용이성 그리고 코드의 유연성을 유지하는 이점을 제공합니다.

모든 라라벨의 파사드는 Illuminate\Support\Facades 네임스페이스 안에 정의되어 있습니다. 따라서 다음처럼 손쉽게 파사드에 엑세스 할 수 있습니다.

use Illuminate\Support\Facades\Cache;  
​  
Route::get('/cache', function () {  
  return Cache::get('key');  
});

라라벨 문서상에서 프레임 워크의 다양한 기능의 사용법을 익히기 위해서 파사드를 사용하는 예제를 많이 확인할 수 있습니다.

## 4.2.언제 파사드를 사용하는가?

파사드는 다양한 장점이 있습니다.

파사드는 의존성으로 주입되거나 직접 설정해야 하는 **긴 이름의 클래스**들을 일일이 기억하지 않고서도 알 수 있는, 간결하고 기억하기 쉬운 문법을 제공합니다. 또한 PHP의 독특한 다이나믹 메소드의 사용 덕분에, 테스트도 쉬워집니다.

그러나 파사드를 사용하는데 몇 가지 주의해야 할 점도 존재합니다. 파사드의 가장 주요한 위험은 **클래스의 책임 범위의 잘못된 사용**입니다. 파사드는 사용하기가 매우 쉽고, **의존성 주입도 필요하지 않기 때문**에, 하나의 클래스 파일 안에서 여러개의 파사드를 사용하는 코드가 자꾸 늘어나도록 클래스를 구성하기가 매우 쉽습니다.

의존성 주입을 사용하면 클래스가 커짐에 따라서 **생성자가 비대**해진다는 시각적인 피드백을 주게 되어 이러한 잠재적 위험이 억제됩니다. 따라서 파사드를 사용할 때에는, 클래스의 책임 범위를 작게 유지하도록 **클래스의 사이즈를 구성**하는데 특별한 주의가 필요합니다.

*{tip} 라라벨과 연관된 써드파티 패키지를 구성할 때, 파사드 사용하는 대신에 [라라벨 contracts](https://laravel.kr/docs/8.x/contracts)를 주입하는 것이 더 나은 방법입니다.* ***패키지는 라라벨의 외부에서 구성****되기 때문에, 라라벨의 파사드 테스팅 헬퍼에 엑세스할 필요는 없습니다.*

### 파사드 Vs. 의존성 주입

의존성 주입의 주요한 장점중 하나는 주입된 클래스의 구현체를 변경할 수 있다는 특성입니다. 이는 테스팅을 수행하는 동안 모킹 객체(mock) 과 스터브(stub) 를 주입할 수 있게 하고, 다양한 메소드가 호출되는 것을 확인할 수 있게 하여 유용합니다.

일반적으로, 정적 클래스 메소드에 대해서는 모킹 객체(mock) 나 스터브(stub) 사용이 불가합니다. 하지만 파사드는 서비스 컨테이너에 의해서 의존성이 해결되는 클래스 객체의 프록시 메소드로 다이나믹 메소드를 사용하기 때문에, 실제 주입된 클래스 인스턴스를 테스트하는 것과 마찬가지로 파사드를 테스트할 수 있습니다. 예를 들어 다음의 주어진 라우트를 보겠습니다.

use Illuminate\Support\Facades\Cache;  
​  
Route::get('/cache', function () {  
  return Cache::get('key');  
});

우리는 Cache::get 메소드가 원하는 인자와 함께 호출되었는지 확인하기 위해서 다음의 테스트를 작성할 수 있습니다

use Illuminate\Support\Facades\Cache;  
​  
/\*\*  
\* A basic functional test example.  
\*  
\* @return void  
\*/  
public function testBasicExample()  
{  
  Cache::shouldReceive('get')  
        ->with('key')  
        ->andReturn('value');  
​  
  $this->visit('/cache')  
        ->see('value');  
}

### 파사드 Vs. 헬퍼 함수

파사드 뿐만 아니라, 라라벨은 뷰 파일을 생성하거나, 이벤트를 발생시키거나, Job을 실행시키거나 또는 HTTP 응답을 반환하는등의 공통된 작업을 수행하는 다양한 "헬퍼" 함수를 포함하고 있습니다. 이러한 다수의 헬퍼 함수들은 파사드와 일치하는 동일한 동작들을 수행합니다. 예를 들어 다음의 파사드 호출과 헬퍼 함수 호출은 동일합니다.

return View::make('profile');  
​  
return view('profile');

이 파사드와 헬퍼함수 사이에는 차이점이 전혀 없습니다. 헬퍼함수를 사용하는 경우에도 일치하는 파사드와 같이 동일하게 테스트 할 수 있습니다. 예를 들어 다음의 주어진 라우트를 보겠습니다.

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

위의 코드에서 cache 헬퍼 함수는 Cache 파사드의 get 메소드를 호출합니다. 따라서 헬퍼 함수를 사용하더라도, 기대되는 인자와 함께 메소드가 호출되었는지 확인하기 위해서 다음의 테스트를 작성할 수 있습니다.

use Illuminate\Support\Facades\Cache;  
​  
/\*\*  
\* A basic functional test example.  
\*  
\* @return void  
\*/  
public function testBasicExample()  
{  
  Cache::shouldReceive('get')  
        ->with('key')  
        ->andReturn('value');  
​  
  $this->visit('/cache')  
        ->see('value');  
}

## 4.3.파사드는 어떻게 동작하는가

라라벨 애플리케이션에서, 파사드는 **컨테이너의 객체에 엑세스하는 방법**을 제공하는 클래스라고 할 수 있습니다. 이 작업을 수행하는 주요 매커니즘이 파사드 클래스안에 있습니다. 라라벨의 파사드들과 여러분이 작성한 파사드들은 기본 Illuminate\Support\Facades\Facade 클래스를 상속받습니다.

Facade 기본 클래스는 \_\_callStatic() 매직 메소드를 사용하여 여러분이 작성한 파사드에 대한 호출을 **컨테이너에서 의존성이 해결된 객체로 전달**합니다.

다음의 예제에서 라라벨의 캐시 시스템을 호출합니다. 이 코드를 보자면, 아마 Cache 클래스의 get static 메소드를 호출한다고 생각할 수 있습니다.

<?php  
​  
namespace App\Http\Controllers;  
​  
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]);  
  }  
}

파일의 상단에 Cache 파사드를 사용하고 있는 부분에 주목해 주십시오. 이 파사드는 Illuminate\Contracts\Cache\Factory 인터페이스의 구현체에 접속할 수 있는 프록시로 동작합니다. 파사드를 사용한 어떠한 호출이라도 라라벨의 캐시 서비스의 구현체에 전달됩니다.

실제로 Illuminate\Support\Facades\Cache를 **살펴보면 get이라는 static 메소드는 찾을 수가 없습니다**.

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

대신에, Cache 파사드는 기본 Facade 클래스를 상속하고 getFacadeAccessor() 메소드를 정의하고 있습니다. 이 메소드의 역할이 서비스 컨테이너의 **바인딩 이름을 반환**한다는 것입니다.

사용자가 Cache 파사드의 어떤 정적 메소드를 참조하려고 할 때, 라라벨은 [서비스 컨테이너]로 부터 cache 로 이름지어진 **바인딩 객체를 찾아** 메소드 호출을 요청할 것입니다(이 경우 get 메소드)

## 4.4.실시간 파사드

실시간 파사드를 사용하여 애플리케이션의 모든 클래스를 파사드처럼 취급 할 수 있습니다. 어떻게 이를 사용할 수 있는지 알기 위해서, 다음의 경우를 살펴보겠습니다.

예를 들어 Podcast 객체모델에서 publish 메소드를 가지고 있다고 가정해 보겠습니다. 이때, podcast를 publish 하기 위해서는 Publisher 인스턴스를 주입해야 합니다.

<?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);

}

}

메소드에 publisher 구현체를 주입하면 주입된 publisher 는 mock 할 수 있기 때문에, 메소드를 분리시켜 테스트를 용이하게 합니다. 그렇지만, publish 메소드를 **호출할 때마다 매번 publisher 인스턴스를 전달할 필요**가 있습니다.

리얼타임 파사드를 사용하면, 동일한 테스트 유효성을 유지하면서도, 명시적으로 Publisher 인스턴스를 전달하지 않아도 됩니다. 리얼타임 파사드를 생성하기 위해서는 Import 한 클래스 이름 앞에 Facades 를 붙이면 됩니다.

<?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);

}

}

리얼타임 파사드가 사용되면, publisher 구현체는 Facades 클래스네임 뒤에 정의된 인터페이스 또는 클래스 네임을 사용하여 서비스 컨테이너에서 의존성이 해결됩니다. 테스트를 진행할 때에는, 이 메소드 호출을 mock 하기 위해서 라라벨에 내장된 파사드 테스팅 헬퍼를 사용할 수 있습니다.

<?php

namespace Tests\Feature;

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 = factory(Podcast::class)->create();

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

$podcast->publish();

}

}

## 4.5.파사드 클래스 목록

다음은 모든 파사드 와 실제 클래스에 대한 목록입니다. 여러분이 특정한 파사드를 기준으로 API 문서를 빠르게 찾고자 하는 경우에 도움이 될것입니다. 응용이 가능하도록 [서비스 컨테이너 바인딩](https://laravel.kr/docs/8.x/container)도 포함되어 있습니다.

| **Facade** | **Class** | **Service Container Binding** |
| --- | --- | --- |
| App | [Illuminate\Foundation\Application](https://laravel.com/api/8.x/Illuminate/Foundation/Application.html) | app |
| Artisan | [Illuminate\Contracts\Console\Kernel](https://laravel.com/api/8.x/Illuminate/Contracts/Console/Kernel.html) | artisan |
| Auth | [Illuminate\Auth\AuthManager](https://laravel.com/api/8.x/Illuminate/Auth/AuthManager.html) | auth |
| Auth (Instance) | [Illuminate\Contracts\Auth\Guard](https://laravel.com/api/8.x/Illuminate/Contracts/Auth/Guard.html) | auth.driver |
| Blade | [Illuminate\View\Compilers\BladeCompiler](https://laravel.com/api/8.x/Illuminate/View/Compilers/BladeCompiler.html) | blade.compiler |
| Broadcast | [Illuminate\Contracts\Broadcasting\Factory](https://laravel.com/api/8.x/Illuminate/Contracts/Broadcasting/Factory.html) |  |
| Broadcast (Instance) | [Illuminate\Contracts\Broadcasting\Broadcaster](https://laravel.com/api/8.x/Illuminate/Contracts/Broadcasting/Broadcaster.html) |  |
| Bus | [Illuminate\Contracts\Bus\Dispatcher](https://laravel.com/api/8.x/Illuminate/Contracts/Bus/Dispatcher.html) |  |
| Cache | [Illuminate\Cache\CacheManager](https://laravel.com/api/8.x/Illuminate/Cache/CacheManager.html) | cache |
| Cache (Instance) | [Illuminate\Cache\Repository](https://laravel.com/api/8.x/Illuminate/Cache/Repository.html) | cache.store |
| Config | [Illuminate\Config\Repository](https://laravel.com/api/8.x/Illuminate/Config/Repository.html) | config |
| Cookie | [Illuminate\Cookie\CookieJar](https://laravel.com/api/8.x/Illuminate/Cookie/CookieJar.html) | cookie |
| Crypt | [Illuminate\Encryption\Encrypter](https://laravel.com/api/8.x/Illuminate/Encryption/Encrypter.html) | encrypter |
| DB | [Illuminate\Database\DatabaseManager](https://laravel.com/api/8.x/Illuminate/Database/DatabaseManager.html) | db |
| DB (Instance) | [Illuminate\Database\Connection](https://laravel.com/api/8.x/Illuminate/Database/Connection.html) | db.connection |
| Event | [Illuminate\Events\Dispatcher](https://laravel.com/api/8.x/Illuminate/Events/Dispatcher.html) | events |
| File | [Illuminate\Filesystem\Filesystem](https://laravel.com/api/8.x/Illuminate/Filesystem/Filesystem.html) | files |
| Gate | [Illuminate\Contracts\Auth\Access\Gate](https://laravel.com/api/8.x/Illuminate/Contracts/Auth/Access/Gate.html) |  |
| Hash | [Illuminate\Contracts\Hashing\Hasher](https://laravel.com/api/8.x/Illuminate/Contracts/Hashing/Hasher.html) | hash |
| Http | [Illuminate\Http\Client\Factory](https://laravel.com/api/8.x/Illuminate/Http/Client/Factory.html) |  |
| Lang | [Illuminate\Translation\Translator](https://laravel.com/api/8.x/Illuminate/Translation/Translator.html) | translator |
| Log | [Illuminate\Log\LogManager](https://laravel.com/api/8.x/Illuminate/Log/LogManager.html) | log |
| Mail | [Illuminate\Mail\Mailer](https://laravel.com/api/8.x/Illuminate/Mail/Mailer.html) | mailer |
| Notification | [Illuminate\Notifications\ChannelManager](https://laravel.com/api/8.x/Illuminate/Notifications/ChannelManager.html) |  |
| Password | [Illuminate\Auth\Passwords\PasswordBrokerManager](https://laravel.com/api/8.x/Illuminate/Auth/Passwords/PasswordBrokerManager.html) | auth.password |
| Password (Instance) | [Illuminate\Auth\Passwords\PasswordBroker](https://laravel.com/api/8.x/Illuminate/Auth/Passwords/PasswordBroker.html) | auth.password.broker |
| Queue | [Illuminate\Queue\QueueManager](https://laravel.com/api/8.x/Illuminate/Queue/QueueManager.html) | queue |
| Queue (Instance) | [Illuminate\Contracts\Queue\Queue](https://laravel.com/api/8.x/Illuminate/Contracts/Queue/Queue.html) | queue.connection |
| Queue (Base Class) | [Illuminate\Queue\Queue](https://laravel.com/api/8.x/Illuminate/Queue/Queue.html) |  |
| Redirect | [Illuminate\Routing\Redirector](https://laravel.com/api/8.x/Illuminate/Routing/Redirector.html) | redirect |
| Redis | [Illuminate\Redis\RedisManager](https://laravel.com/api/8.x/Illuminate/Redis/RedisManager.html) | redis |
| Redis (Instance) | [Illuminate\Redis\Connections\Connection](https://laravel.com/api/8.x/Illuminate/Redis/Connections/Connection.html) | redis.connection |
| Request | [Illuminate\Http\Request](https://laravel.com/api/8.x/Illuminate/Http/Request.html) | request |
| Response | [Illuminate\Contracts\Routing\ResponseFactory](https://laravel.com/api/8.x/Illuminate/Contracts/Routing/ResponseFactory.html) |  |
| Response (Instance) | [Illuminate\Http\Response](https://laravel.com/api/8.x/Illuminate/Http/Response.html) |  |
| Route | [Illuminate\Routing\Router](https://laravel.com/api/8.x/Illuminate/Routing/Router.html) | router |
| Schema | [Illuminate\Database\Schema\Builder](https://laravel.com/api/8.x/Illuminate/Database/Schema/Builder.html) |  |
| Session | [Illuminate\Session\SessionManager](https://laravel.com/api/8.x/Illuminate/Session/SessionManager.html) | session |
| Session (Instance) | [Illuminate\Session\Store](https://laravel.com/api/8.x/Illuminate/Session/Store.html) | session.store |
| Storage | [Illuminate\Filesystem\FilesystemManager](https://laravel.com/api/8.x/Illuminate/Filesystem/FilesystemManager.html) | filesystem |
| Storage (Instance) | [Illuminate\Contracts\Filesystem\Filesystem](https://laravel.com/api/8.x/Illuminate/Contracts/Filesystem/Filesystem.html) | filesystem.disk |
| URL | [Illuminate\Routing\UrlGenerator](https://laravel.com/api/8.x/Illuminate/Routing/UrlGenerator.html) | url |
| Validator | [Illuminate\Validation\Factory](https://laravel.com/api/8.x/Illuminate/Validation/Factory.html) | validator |
| Validator (Instance) | [Illuminate\Validation\Validator](https://laravel.com/api/8.x/Illuminate/Validation/Validator.html) |  |
| View | [Illuminate\View\Factory](https://laravel.com/api/8.x/Illuminate/View/Factory.html) | view |
| View (Instance) | [Illuminate\View\View](https://laravel.com/api/8.x/Illuminate/View/View.html) |  |

# 5.Contracts

## 5.1.시작하기

라라벨의 Contract는 프레임워크에서 제공하는 코어 서비스들을 정의한 **인터페이스들의 모음**입니다.

예를 들어, Illuminate\Contracts\Queue\Queue Contract에는 어떤 작업들을 큐에서 다룰때 필요한 메소드들이 정의되어 있고, Illuminate\Contracts\Mail\Mailer Contract에는 이메일을 보내기 위해 필요한 메소드들을 정의되어 있습니다.

라라벨 프레임워크에는 각각의 Contract에 상응하는 구현체(구현 클래스)가 있습니다. 예를 들어, 라라벨은 다양한 **드라이버로 구현된 queue의 구현체**를 가지고 있고, [SwiftMailer](https://swiftmailer.symfony.com/)를 mailer의 구현체로 가지고 있습니다.

라라벨의 모든 Contract는 [각각의 Github 저장소](https://github.com/illuminate/contracts)를 가지고 있습니다. 이것은 별도의 패키지에 의존하지 않는 **각각의 단일 패키지**로, 개발자들이 사용할 수 있도록 하는 contract를 위한 하나의 레퍼런스를 제공합니다.

### Contracts VS Facades

라라벨의 [파사드](https://laravel.kr/docs/8.x/facades)는 서비스 컨테이너 외부에서 타입 힌트나, contracts 의 의존성 없이도 라라벨의 서비스를 시용할 수 있는 쉬운 방법을 제공합니다.

클래스 생성자에서 요구하지 않아도 되는 facade와 달리 contracts를 통해 클래스에 대한 명시적 의존성을 정의 할 수 있습니다. 일부 개발자는 이러한 방식으로 의존성을 명시적으로 정의하는 contracts를 선호하지만 대다수의 개발자는 facades의 편리함을 누리고 있습니다.

*{tip} 대부분의 애플리케이션은 facades나 contract중 선호하는 어느것을 사용해도 무방합니다. 그러나* ***패키지를 빌드****하는 경우 패키지 컨텍스트에서 테스트하기 쉽기 때문에* ***contract 사용을 강력하게 고려****해야합니다.*

## 5.2.Contracts 사용 시기

다른 곳에서 논의 된 것처럼, contract나 facade를 사용하기로 한 결정의 대부분은 개인적인 취향과 개발 팀의 취향에 달려 있습니다. contract와 facades 모두 강력하고 잘 테스트 된 Laravel 애플리케이션을 작성하는 데 사용할 수 있습니다. 클래스가 제 역할을 하는데에 contract와 facades를 사용하는 데는 실제적인 차이점이 거의 없습니다.

그러나 계약(contracts)과 관련하여 몇 가지 의문을 떠올릴 수도 있습니다. 예를 들어, 인터페이스를 사용하는 이유는 무엇인가? 인터페이스를 더 복잡하게 사용하고 있지는 않은가? 입니다. 다음의 *느슨한 결합 및 단순성* 부분에서 인터페이스를 사용하는 이유를 설명합니다.

### 느슨한 결합

우선, 한 캐시 구현체에 강하게 결합돼 있는 코드를 살펴봅시다.

<?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)) {  
          //  
      }  
  }  
}

이 클래스의 코드는 주어진 캐시 구현체와 밀접하게 결합돼 있습니다. 특정 패키지 벤더의 캐시 구상클래스에 의존하기 때문에 이 코드는 캐시 클래스와 밀접하게 결합돼 있는 것입니다. 만약 이 패키지의 API가 변경되면 예시로든 이 코드 또한 변경되어야 합니다.

또한, 코드가 사용하는 캐시(Memcached)를 다른 것(Redis)으로 변경하고자 하는 경우, 역시나 Repository 클래스를 다시 수정해야만 할 것입니다. 저장소 클래스는 누가 어떻게 데이터를 제공하는지에 대한 정보를 너무 많이 가지고 있어서는 안 됩니다.

**이렇게 접근하는 대신, 특정 벤더에 구속되지 않고 단순한 인터페이스에 의존하도록 하여 코드를 개선할 수 있습니다.**

<?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;  
  }  
}

이제 코드는 어떤 특정 벤더, 심지어 라라벨과도 결합되지 않습니다. Contract는 구현체를 가지지 않고, 의존성도 없기 때문에, 주어진 Contract의 다른 구현체를 쉽게 작성할 수 있습니다. 캐시를 사용하는 코드를 수정하지 않고도 캐시 구현체를 대체할 수 있습니다.

### 단순함

라라벨의 모든 서비스들이 단순한 인터페이스로 보기 좋게 정의돼 있기 때문에, 그 서비스들에 의해 제공되는 기능을 알아내는 것이 매우 쉽습니다. **Contract들이 프레임워크의 기능들에 대한 간결한 도큐먼트의 역할을 하는 것입니다.**

또한, 여러분이 간단한 인터페이스에 의존하게 되면, 여러분의 코드는 이해하거나 유지 보수하기가 더 쉬워집니다. 크고 복잡한 클래스에서 사용할 수 있는 메소드들을 훑어보는 대신, 단순하고 깨끗한 인터페이스를 참고할 수 있습니다.

## 5.3.Contract 레퍼런스

그럼 어떻게 Contract의 구현체를 얻을 수 있을까요? 사실 매우 간단합니다.

라라벨에 있는 여러 종류의 클래스들은 컨트롤러, 이벤트리스너, 미들웨어, 큐 작업, 라우트 클로저들을 관리하는 **[서비스 컨테이너](https://laravel.kr/docs/8.x/container)를 통해 의존성 해결(resolve)** 되고 있습니다. 따라서 의존성이 해결되는 어떤 클래스가 특정 Contract의 구현체를 얻으려면 그 클래스의 **생성자에 그 인터페이스를 "type-hint"로 지정**해놓으면 됩니다

그 예로 아래의 이벤트 리스너를 보겠습니다.

<?php  
​  
namespace App\Listeners;  
​  
use App\Events\OrderWasPlaced;  
use App\Models\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)  
  {  
      //  
  }  
}

이벤트리스너가 의존성 해결될 때, 서비스 컨테이너는 클래스의 **생성자에 있는 타입힌트**를 읽고, 그에 적합한 값을 주입해 줍니다. 서비스 컨테이너에 무언가를 등록하는 것에 대하여 더 알고싶다면, [이 문서](https://laravel.kr/docs/8.x/container)를 보시기 바랍니다.

## 5.4.Contract 참조

아래는 대부분의 라라벨 Contract와 그에 대응되는 파사드들의 레퍼런스입니다.

| **Contract** | **References Facade** |
| --- | --- |
| [Illuminate\Contracts\Auth\Access\Authorizable](https://github.com/illuminate/contracts/blob/8.x/Auth/Access/Authorizable.php) |  |
| [Illuminate\Contracts\Auth\Access\Gate](https://github.com/illuminate/contracts/blob/8.x/Auth/Access/Gate.php) | Gate |
| [Illuminate\Contracts\Auth\Authenticatable](https://github.com/illuminate/contracts/blob/8.x/Auth/Authenticatable.php) |  |
| [Illuminate\Contracts\Auth\CanResetPassword](https://github.com/illuminate/contracts/blob/8.x/Auth/CanResetPassword.php) |  |
| [Illuminate\Contracts\Auth\Factory](https://github.com/illuminate/contracts/blob/8.x/Auth/Factory.php) | Auth |
| [Illuminate\Contracts\Auth\Guard](https://github.com/illuminate/contracts/blob/8.x/Auth/Guard.php) | Auth::guard() |
| [Illuminate\Contracts\Auth\PasswordBroker](https://github.com/illuminate/contracts/blob/8.x/Auth/PasswordBroker.php) | Password::broker() |
| [Illuminate\Contracts\Auth\PasswordBrokerFactory](https://github.com/illuminate/contracts/blob/8.x/Auth/PasswordBrokerFactory.php) | Password |
| [Illuminate\Contracts\Auth\StatefulGuard](https://github.com/illuminate/contracts/blob/8.x/Auth/StatefulGuard.php) |  |
| [Illuminate\Contracts\Auth\SupportsBasicAuth](https://github.com/illuminate/contracts/blob/8.x/Auth/SupportsBasicAuth.php) |  |
| [Illuminate\Contracts\Auth\UserProvider](https://github.com/illuminate/contracts/blob/8.x/Auth/UserProvider.php) |  |
| [Illuminate\Contracts\Bus\Dispatcher](https://github.com/illuminate/contracts/blob/8.x/Bus/Dispatcher.php) | Bus |
| [Illuminate\Contracts\Bus\QueueingDispatcher](https://github.com/illuminate/contracts/blob/8.x/Bus/QueueingDispatcher.php) | Bus::dispatchToQueue() |
| [Illuminate\Contracts\Broadcasting\Factory](https://github.com/illuminate/contracts/blob/8.x/Broadcasting/Factory.php) | Broadcast |
| [Illuminate\Contracts\Broadcasting\Broadcaster](https://github.com/illuminate/contracts/blob/8.x/Broadcasting/Broadcaster.php) | Broadcast::connection() |
| [Illuminate\Contracts\Broadcasting\ShouldBroadcast](https://github.com/illuminate/contracts/blob/8.x/Broadcasting/ShouldBroadcast.php) |  |
| [Illuminate\Contracts\Broadcasting\ShouldBroadcastNow](https://github.com/illuminate/contracts/blob/8.x/Broadcasting/ShouldBroadcastNow.php) |  |
| [Illuminate\Contracts\Cache\Factory](https://github.com/illuminate/contracts/blob/8.x/Cache/Factory.php) | Cache |
| [Illuminate\Contracts\Cache\Lock](https://github.com/illuminate/contracts/blob/8.x/Cache/Lock.php) |  |
| [Illuminate\Contracts\Cache\LockProvider](https://github.com/illuminate/contracts/blob/8.x/Cache/LockProvider.php) |  |
| [Illuminate\Contracts\Cache\Repository](https://github.com/illuminate/contracts/blob/8.x/Cache/Repository.php) | Cache::driver() |
| [Illuminate\Contracts\Cache\Store](https://github.com/illuminate/contracts/blob/8.x/Cache/Store.php) |  |
| [Illuminate\Contracts\Config\Repository](https://github.com/illuminate/contracts/blob/8.x/Config/Repository.php) | Config |
| [Illuminate\Contracts\Console\Application](https://github.com/illuminate/contracts/blob/8.x/Console/Application.php) |  |
| [Illuminate\Contracts\Console\Kernel](https://github.com/illuminate/contracts/blob/8.x/Console/Kernel.php) | Artisan |
| [Illuminate\Contracts\Container\Container](https://github.com/illuminate/contracts/blob/8.x/Container/Container.php) | App |
| [Illuminate\Contracts\Cookie\Factory](https://github.com/illuminate/contracts/blob/8.x/Cookie/Factory.php) | Cookie |
| [Illuminate\Contracts\Cookie\QueueingFactory](https://github.com/illuminate/contracts/blob/8.x/Cookie/QueueingFactory.php) | Cookie::queue() |
| [Illuminate\Contracts\Database\ModelIdentifier](https://github.com/illuminate/contracts/blob/8.x/Database/ModelIdentifier.php) |  |
| [Illuminate\Contracts\Debug\ExceptionHandler](https://github.com/illuminate/contracts/blob/8.x/Debug/ExceptionHandler.php) |  |
| [Illuminate\Contracts\Encryption\Encrypter](https://github.com/illuminate/contracts/blob/8.x/Encryption/Encrypter.php) | Crypt |
| [Illuminate\Contracts\Events\Dispatcher](https://github.com/illuminate/contracts/blob/8.x/Events/Dispatcher.php) | Event |
| [Illuminate\Contracts\Filesystem\Cloud](https://github.com/illuminate/contracts/blob/8.x/Filesystem/Cloud.php) | Storage::cloud() |
| [Illuminate\Contracts\Filesystem\Factory](https://github.com/illuminate/contracts/blob/8.x/Filesystem/Factory.php) | Storage |
| [Illuminate\Contracts\Filesystem\Filesystem](https://github.com/illuminate/contracts/blob/8.x/Filesystem/Filesystem.php) | Storage::disk() |
| [Illuminate\Contracts\Foundation\Application](https://github.com/illuminate/contracts/blob/8.x/Foundation/Application.php) | App |
| [Illuminate\Contracts\Hashing\Hasher](https://github.com/illuminate/contracts/blob/8.x/Hashing/Hasher.php) | Hash |
| [Illuminate\Contracts\Http\Kernel](https://github.com/illuminate/contracts/blob/8.x/Http/Kernel.php) |  |
| [Illuminate\Contracts\Mail\MailQueue](https://github.com/illuminate/contracts/blob/8.x/Mail/MailQueue.php) | Mail::queue() |
| [Illuminate\Contracts\Mail\Mailable](https://github.com/illuminate/contracts/blob/8.x/Mail/Mailable.php) |  |
| [Illuminate\Contracts\Mail\Mailer](https://github.com/illuminate/contracts/blob/8.x/Mail/Mailer.php) | Mail |
| [Illuminate\Contracts\Notifications\Dispatcher](https://github.com/illuminate/contracts/blob/8.x/Notifications/Dispatcher.php) | Notification |
| [Illuminate\Contracts\Notifications\Factory](https://github.com/illuminate/contracts/blob/8.x/Notifications/Factory.php) | Notification |
| [Illuminate\Contracts\Pagination\LengthAwarePaginator](https://github.com/illuminate/contracts/blob/8.x/Pagination/LengthAwarePaginator.php) |  |
| [Illuminate\Contracts\Pagination\Paginator](https://github.com/illuminate/contracts/blob/8.x/Pagination/Paginator.php) |  |
| [Illuminate\Contracts\Pipeline\Hub](https://github.com/illuminate/contracts/blob/8.x/Pipeline/Hub.php) |  |
| [Illuminate\Contracts\Pipeline\Pipeline](https://github.com/illuminate/contracts/blob/8.x/Pipeline/Pipeline.php) |  |
| [Illuminate\Contracts\Queue\EntityResolver](https://github.com/illuminate/contracts/blob/8.x/Queue/EntityResolver.php) |  |
| [Illuminate\Contracts\Queue\Factory](https://github.com/illuminate/contracts/blob/8.x/Queue/Factory.php) | Queue |
| [Illuminate\Contracts\Queue\Job](https://github.com/illuminate/contracts/blob/8.x/Queue/Job.php) |  |
| [Illuminate\Contracts\Queue\Monitor](https://github.com/illuminate/contracts/blob/8.x/Queue/Monitor.php) | Queue |
| [Illuminate\Contracts\Queue\Queue](https://github.com/illuminate/contracts/blob/8.x/Queue/Queue.php) | Queue::connection() |
| [Illuminate\Contracts\Queue\QueueableCollection](https://github.com/illuminate/contracts/blob/8.x/Queue/QueueableCollection.php) |  |
| [Illuminate\Contracts\Queue\QueueableEntity](https://github.com/illuminate/contracts/blob/8.x/Queue/QueueableEntity.php) |  |
| [Illuminate\Contracts\Queue\ShouldQueue](https://github.com/illuminate/contracts/blob/8.x/Queue/ShouldQueue.php) |  |
| [Illuminate\Contracts\Redis\Factory](https://github.com/illuminate/contracts/blob/8.x/Redis/Factory.php) | Redis |
| [Illuminate\Contracts\Routing\BindingRegistrar](https://github.com/illuminate/contracts/blob/8.x/Routing/BindingRegistrar.php) | Route |
| [Illuminate\Contracts\Routing\Registrar](https://github.com/illuminate/contracts/blob/8.x/Routing/Registrar.php) | Route |
| [Illuminate\Contracts\Routing\ResponseFactory](https://github.com/illuminate/contracts/blob/8.x/Routing/ResponseFactory.php) | Response |
| [Illuminate\Contracts\Routing\UrlGenerator](https://github.com/illuminate/contracts/blob/8.x/Routing/UrlGenerator.php) | URL |
| [Illuminate\Contracts\Routing\UrlRoutable](https://github.com/illuminate/contracts/blob/8.x/Routing/UrlRoutable.php) |  |
| [Illuminate\Contracts\Session\Session](https://github.com/illuminate/contracts/blob/8.x/Session/Session.php) | Session::driver() |
| [Illuminate\Contracts\Support\Arrayable](https://github.com/illuminate/contracts/blob/8.x/Support/Arrayable.php) |  |
| [Illuminate\Contracts\Support\Htmlable](https://github.com/illuminate/contracts/blob/8.x/Support/Htmlable.php) |  |
| [Illuminate\Contracts\Support\Jsonable](https://github.com/illuminate/contracts/blob/8.x/Support/Jsonable.php) |  |
| [Illuminate\Contracts\Support\MessageBag](https://github.com/illuminate/contracts/blob/8.x/Support/MessageBag.php) |  |
| [Illuminate\Contracts\Support\MessageProvider](https://github.com/illuminate/contracts/blob/8.x/Support/MessageProvider.php) |  |
| [Illuminate\Contracts\Support\Renderable](https://github.com/illuminate/contracts/blob/8.x/Support/Renderable.php) |  |
| [Illuminate\Contracts\Support\Responsable](https://github.com/illuminate/contracts/blob/8.x/Support/Responsable.php) |  |
| [Illuminate\Contracts\Translation\Loader](https://github.com/illuminate/contracts/blob/8.x/Translation/Loader.php) |  |
| [Illuminate\Contracts\Translation\Translator](https://github.com/illuminate/contracts/blob/8.x/Translation/Translator.php) | Lang |
| [Illuminate\Contracts\Validation\Factory](https://github.com/illuminate/contracts/blob/8.x/Validation/Factory.php) | Validator |
| [Illuminate\Contracts\Validation\ImplicitRule](https://github.com/illuminate/contracts/blob/8.x/Validation/ImplicitRule.php) |  |
| [Illuminate\Contracts\Validation\Rule](https://github.com/illuminate/contracts/blob/8.x/Validation/Rule.php) |  |
| [Illuminate\Contracts\Validation\ValidatesWhenResolved](https://github.com/illuminate/contracts/blob/8.x/Validation/ValidatesWhenResolved.php) |  |
| [Illuminate\Contracts\Validation\Validator](https://github.com/illuminate/contracts/blob/8.x/Validation/Validator.php) | Validator::make() |
| [Illuminate\Contracts\View\Engine](https://github.com/illuminate/contracts/blob/8.x/View/Engine.php) |  |
| [Illuminate\Contracts\View\Factory](https://github.com/illuminate/contracts/blob/8.x/View/Factory.php) | View |
| [Illuminate\Contracts\View\View](https://github.com/illuminate/contracts/blob/8.x/View/View.php) | View::make() |