单一责任原则

一个方法应该只有一个责任，一个强大的功能由多个单一职责的方法组合而成，而不是把所有功能写到一个方法里，如果那样的话方法复用的可能性几乎为零，并且难以维护。

一个类同样应该是针对单一事物的抽象，里面包含的方法和属性应该只与类代表的事物有关。

model层只写与数据表抽象相关的代码

业务代码单独抽离到service层，由service再去调用model里定义的方法来实现数据表操作

Routes目录：

web.php文件，所有路由都应该再这个文件定义

控制反转（IoC）模式主要是用来解决系统组件间相互依赖关系的一种模式（解耦）

依赖：可以理解为一个对象实现某个功能需要其他对象相关功能的支持；

耦合：使用new关键字在一个对象内部实例化另一个对象；

控制反转是将这种依赖关系通过动态注入的方式实现，内部初始化行为转移到外部；

高内聚，低耦合

文档5.5：

Composer安装laravel ：

composer create-project --prefer-dist laravel/laravel blog "5.5.\*"

安装完成后的配置：

安装 Laravel 之后，你要将 Web 服务器的根目录指向 public 目录。该目录下的 index.php 文件将作为所有进入应用程序的 HTTP 请求的前端控制器；

Laravel 框架的所有配置文件都放在 config 目录中；

需要给这两个文件配置读写权限：storage 目录和 bootstrap/cache 目录应该允许 Web 服务器写入；

Nginx，需要在站点配置中加入以下内容，它将会将所有请求都引导到 index.php 前端控制器：

location / {

try\_files $uri $uri/ /index.php?$query\_string;

}

.env 文件不应该提交到应用程序的源代码控制系统中，可以使用 env 函数检索这些变量的值。

例如 'debug' => env('APP\_DEBUG', false),

传递给 env 函数的第二个值是「默认值」。如果给定的键不存在环境变量，则会使用该值。

使用全局 config 函数来访问配置值。配置值的访问可以使用「点」语法，这其中包含了要访问的文件和选项的名称。还可以指定默认值，如果配置选项不存在，则返回默认值：

$value = config('app.timezone');

目录：

app 目录包含应用程序的核心代码。app 目录里的许多类都可以通过 Artisan 命令来生成。要查看可用的命令，只要在终端里运行 php artisan list make 命令即可。

Http 目录包含了控制器、中间件和表单请求。几乎所有的进入应用的请求的处理逻辑都被放在这里。

bootstrap 目录包含引导框架并配置自动加载的文件。该目录还包含了一个 cache 目录，存放着框架生成的用来提升性能的文件，比如路由和服务缓存文件。

CSRF 白名单

不需要 CSRF 保护的路由可以放到 routes/web.php 外，因为 RouteServiceProvider 的 web 中间件适用于该文件中的所有路由。不过，你也可以通过将这类 URI 添加到 VerifyCsrfToken 中间件中的 $except 属性来排除对这类路由的 CSRF 保护：app/Http/Middleware/VerifyCsrfToken.php

protected $except = [

'stripe/\*',

];

单个行为控制器

如果你想定义一个只处理单个行为的控制器，你可以在控制器中放置一个 \_\_invoke 方法：

public function \_\_invoke($id)

{

return view('user.profile', ['user' => User::findOrFail($id)]);

}

注册单个行为控制器的路由时，不需要指定方法：

Route::get('user/{id}', 'ShowProfile');

使用依赖注入

可以在控制器方法中 参数的类型提示 指定依赖项。最常见的用法就是将 Illuminate\Http\Request 实例注入到控制器方法中：

use Illuminate\Http\Request;

class UserController extends Controller{

public function store(Request $request) //依赖注入

{

$name = $request->name;

}

}

也可以在构造函数中实现依赖注入

use App\Repositories\UserRepository;

………

public function \_\_construct(UserRepository $users)

{

$this->users = $users;

}

………

注意，如果控制器方法要从路由参数中获取数据，则应在其他依赖项之后列出路由参数。

例如路由定义：Route::put('user/{id}', 'UserController@update');

方法参数：public function update(Request $request, $id)

数据库的配置文件放置在 config/database.php 文件中

**运行原生 SQL**

**use Illuminate\Support\Facades\DB;**

**查询select：**

$users = DB::select('select \* from users where active = ?', [1]);

第二个参数则是传递需要绑定到查询中的参数值，参数绑定提供了对防止 SQL 注入的保护。

除了使用 ? 来表示参数绑定外，你也可以使用命名绑定来执行一个查询：

$results = DB::select('select \* from users where id = :id', ['id' => 1]);

select 方法将始终返回一个数组。数组中的每个结果都是一个 PHP StdClass 对象，可以像下面这样访问结果的值:

foreach ($users as $user) {

echo $user->name;

}

插入insert

DB::insert('insert into users (id, name) values (?, ?)', [1, 'Dayle']);

更新 update: 返回受影响的行数

DB::update('update users set votes = 100 where name = ?', ['John']);

删除语句 delete

DB::delete('delete from users');

运行普通语句

DB::statement('drop table users');

**查询构造器**

use Illuminate\Support\Facades\DB;

**查询：**

$users = DB::table('users')->get();

返回PHP StdClass 对象的实例

$user = DB::table('users')->where('name', 'John')->first();

只从数据库表中获取一行数据，这个方法将返回一个 StdClass 对象：

$user->name;

$users = DB::table('users')->select('name', 'email as user\_email')->get();

使用 select 方法自定义一个 select 语句来指定查询的字段：

$users = DB::table('users')

->select(DB::raw('count(\*) as user\_count, status'))

->where('status', '<>', 1)

->groupBy('status')

->get();

orderBy 方法的第一个参数是你想要用来排序的字段，而第二个参数控制排序的方向，可以是 asc 或 desc

**插入：**

查询构造器也提供了将记录插入数据库表的 insert 方法。insert 方法接受一个字段名和值的数组作为参数：

DB::table('users')->insert([

['email' => 'taylor@example.com', 'votes' => 0],

['email' => 'dayle@example.com', 'votes' => 0]

]);

若数据表存在自增的 ID，则可以使用 insertGetId 方法来插入记录然后获取其 ID：

**update 更新**

使用 update 来更新已存在的记录。update 方法和 insert 方法一样，接受包含要更新的字段及值的数组。你可以使用 where 语句来约束 update 的查询：

**自增 & 自减**

DB::table('users')->increment('votes');

DB::table('users')->increment('votes', 5);

DB::table('users')->decrement('votes');

DB::table('users')->decrement('votes', 5);

Laravel 的 Eloquent ORM

每个数据库表都有一个对应的「模型」用来与该表交互。

可把每个 Eloquent 模型想像成强大的 [查询构造器](https://learnku.com/docs/laravel/5.5/queries)，它让你可以流畅地查询与该模型相关联的数据库表。

观察器

Eloquent 的模型的检索、保存或者更新等操作可以触发以下几个事件： retrieved、creating、created、updating、updated、saving、saved、deleting、deleted、restoring、restored。

检索时会触发 retrieved 事件

新模型第一次保存 触发creating 以及 created 事件，saving / saved 事件

模型调用了save 方法，触发updating 和 updated 事件，saving / saved 事件

观察器类里的方法名应该对应 Eloquent 中你想监听的事件。每种方法接收 model 作为其唯一的参数。Laravel 没有为观察器设置默认的目录，所以你可以创建任何你喜欢你的目录来存放：