链滴

(转载) Laravel 最佳实践

作者：xiaoxiezaijia

原文链接：https://ld246.com/article/1568601671883

来源网站：链滴

单一职责原则

一个类和一个方法应该只有一个责任。

例如:

```
public function getFullNameAttribute()
{
    if (auth()->user() && auth()->user()->hasRole('client') && auth()->user()->isVerified()) {
        return 'Mr. ' . $this->first_name . ' ' . $this->middle_name . ' ' . $this->last_name;
    } else {
        return $this->first_name[0] . '. ' . $this->last_name;
    }
}
```

更优的写法:

```
public function getFullNameAttribute()
{
    return $this->isVerifiedClient() ? $this->getFullNameLong() : $this->getFullNameShort();
}

public function isVerifiedClient()
{
    return auth()->user() && auth()->user()->hasRole('client') && auth()->user()->isVerified();
}

public function getFullNameLong()
{
    return 'Mr. ' . $this->first_name . ' ' . $this->middle_name . ' ' . $this->last_name;
}

public function getFullNameShort()
{
    return $this->first_name[0] . '. ' . $this->last_name;
}
```

保持控制器的简洁

如果您使用的是查询生成器或原始SQL查询，请将所有与数据库相关的逻辑放入Eloquent模型或Repository类中。

例如:

```
public function index()
{
    $clients = Client::verified()
        ->with(['orders' => function ($q) {
            $q->where('created_at', '>', Carbon::today()->subWeek());
        }])
        ->get();

    return view('index', ['clients' => $clients]);
```

```
}
```

更优的写法:

```
public function index()
{
    return view('index', ['clients' => $this->client->getWithNewOrders()]);
}

class Client extends Model
{
    public function getWithNewOrders()
    {
        return $this->verified()
            ->with(['orders' => function ($q) {
                $q->where('created_at', '>', Carbon::today()->subWeek());
            }])
            ->get();
    }
}
```

使用自定义Request类来进行验证

把验证规则放到 Request 类中.

例子:

```
public function store(Request $request)
{
    $request->validate([
        'title' => 'required|unique:posts|max:255',
        'body' => 'required',
        'publish_at' => 'nullable|date',
    ]);

    ....
}
```

更优的写法:

```
public function store(PostRequest $request)
{
    ....
}

class PostRequest extends Request
{
    public function rules()
    {
        return [
            'title' => 'required|unique:posts|max:255',
            'body' => 'required',
            'publish_at' => 'nullable|date',
        ];
```

```
    }
}
```

业务代码要放到服务层中

控制器必须遵循单一职责原则，因此最好将业务代码从控制器移动到服务层中。

例子:

```
public function store(Request $request)
{
    if ($request->hasFile('image')) {
        $request->file('image')->move(public_path('images') . 'temp');
    }

    ....
}
```

更优的写法:

```
public function store(Request $request)
{
    $this->articleService->handleUploadedImage($request->file('image'));

    ....
}

class ArticleService
{
    public function handleUploadedImage($image)
    {
        if (!is_null($image)) {
            $image->move(public_path('images') . 'temp');
        }
    }
}
```

DRY原则 不要重复自己

尽可能重用代码，SRP可以帮助您避免重复造轮子。 此外尽量重复使用Blade模板，使用Eloquent的 scopes 方法来实现代码。

例子:

```
public function getActive()
{
    return $this->where('verified', 1)->whereNotNull('deleted_at')->get();
}

public function getArticles()
{
    return $this->whereHas('user', function ($q) {
            $q->where('verified', 1)->whereNotNull('deleted_at');
```

```
        })->get();
}
```

更优的写法:

```
public function scopeActive($q)
{
    return $q->where('verified', 1)->whereNotNull('deleted_at');
}

public function getActive()
{
    return $this->active()->get();
}

public function getArticles()
{
    return $this->whereHas('user', function ($q) {
            $q->active();
        })->get();
}
```

使用ORM而不是纯sql语句，使用集合而不是数组

使用Eloquent可以帮您编写可读和可维护的代码。 此外Eloquent还有非常优雅的内置工具，如软删，事件，范围等。

例子:

```
SELECT *
FROM `articles`
WHERE EXISTS (SELECT *
              FROM `users`
              WHERE `articles`.`user_id` = `users`.`id`
              AND EXISTS (SELECT *
                          FROM `profiles`
                          WHERE `profiles`.`user_id` = `users`.`id`)
              AND `users`.`deleted_at` IS NULL)
AND `verified` = '1'
AND `active` = '1'
ORDER BY `created_at` DESC
```

更优的写法:

```
Article::has('user.profile')->verified()->latest()->get();
```

集中处理数据

例子:

```
$article = new Article;
$article->title = $request->title;
$article->content = $request->content;
```

```
$article->verified = $request->verified;
// Add category to article
$article->category_id = $category->id;
$article->save();
```

更优的写法:

```
$category->article()->create($request->validated());
```

不要在模板中查询，尽量使用惰性加载

例子 (对于100个用户，将执行101次DB查询):

```
@foreach (User::all() as $user)
    {{ $user->profile->name }}
@endforeach
```

更优的写法 (对于100个用户，使用以下写法只需执行2次DB查询):

```
$users = User::with('profile')->get();

...

@foreach ($users as $user)
    {{ $user->profile->name }}
@endforeach
```

注释你的代码，但是更优雅的做法是使用描述性的语言来编写你的代码

例子:

```
if (count((array) $builder->getQuery()->joins) > 0)
```

加上注释:

```
// 确定是否有任何连接
if (count((array) $builder->getQuery()->joins) > 0)
```

更优的写法:

```
if ($this->hasJoins())
```

不要把 JS 和 CSS 放到 Blade 模板中，也不要把任何 HTML 代码放到 PHP 代码里

例子:

```
let article = `{{ json_encode($article) }}`;
```

更好的写法:

```
<input id="article" type="hidden" value="@json($article)">
```

Or

```
<button class="js-fav-article" data-article="@json($article)">{{ $article->name }}<button>
```

在Javascript文件中加上:

```
let article = $('#article').val();
```

当然最好的办法还是使用专业的PHP的JS包传输数据。

在代码中使用配置、语言包和常量，而不是使用硬编码

例子:

```
public function isNormal()
{
    return $article->type === 'normal';
}

return back()->with('message', 'Your article has been added!');
```

更优的写法:

```
public function isNormal()
{
    return $article->type === Article::TYPE_NORMAL;
}

return back()->with('message', __('app.article_added'));
```

使用社区认可的标准Laravel工具

强力推荐使用内置的Laravel功能和扩展包，而不是使用第三方的扩展包和工具。

如果你的项目被其他开发人员接手了，他们将不得不重新学习这些第三方工具的使用教程。

此外，当您使用第三方扩展包或工具时，你很难从Laravel社区获得什么帮助。 不要让你的客户为额的问题付钱。

想要实现的功能	标准工具	第三工具
权限	Policies	Entrust, Sentinel 者其他扩展包
资源编译工具	Laravel Mix	Grunt, Gulp, 或者其他第三方包
开发环境	Homestead	Docker
部署	Laravel Forge	Deployer 者其他解决方案
自动化测试	PHPUnit, Mockery	hpspec
页面预览测试	Laravel Dusk	Cod ception

DB操纵	Eloquent	SQL, Doctrine
模板	Blade	Twig
数据操纵	Laravel集合	数组
表单验证	Request classes	他第三方包,甚至在控制器中做验证
权限	Built-in	他第三方包或者你自己解决
API身份验证	Laravel Passport	三方的JWT或者 OAuth 扩展包
创建 API	Built-in	Dingo API 或类似的扩展包
创建数据库结构	Migrations	直用 DB 语句创建
本土化	Built-in	第三方包
实时消息队列	Laravel Echo, Pusher	用第三方包或者直接使用WebSockets
创建测试数据	Seeder classes, Model Factories, Faker	动创建测试数据
任务调度	Laravel Task Scheduler	本和第三方包
数据库	MySQL, PostgreSQL, SQLite, SQL Server	ongoDB

遵循laravel命名约定

来源 PSR standards.

另外，遵循Laravel社区认可的命名约定：

对象	规则	更优的写法	避免的写法
控制器	单数	ArticleController	rticlesController
路由	复数	articles/1	artic e/1
路由命名	带点符号的蛇形命名	users.show_active	sers.show-active, show-active-users
模型	单数	User	Users
hasOne或belongsTo关系	单数	articleComment	rticleComments, article_comment
所有其他关系	复数	articleComments	rticleComment, article_comments
表单	复数	article_comments	rticle_comment, articleComments

透视表	按字母顺序排列模型	article_user	ser_article, articles_users
数据表字段	使用蛇形并且不要带表名	meta_title	etaTitle; article_meta_title
模型参数	蛇形命名	$model->created_at	model->createdAt
外键	带有_id后缀的单数模型名称	article_id	rticleId, id_article, articles_id
主键	-	id	custom_id
迁移	-	2017_01_01_000000_create_articles_table	017_01_01_000000_articles
方法	驼峰命名	getAll	get all
资源控制器	table	store	s veArticle
测试类	驼峰命名	testGuestCannotSeeArticle	est_guest_cannot_see_article
变量	驼峰命名	$articlesWithAuthor	articles_with_author
集合	描述性的, 复数的	$activeUsers = User::activ	()->get() active, data
对象	描述性的, 单数的	$activeUser = User::active	)->first() users, obj
配置和语言文件索引	蛇形命名	articles_enabled	rticlesEnabled; articles-enabled
视图	短横线命名	show-filtered.blade.php	howFiltered.blade.php, show_filtered.blade.php
配置	蛇形命名	google_calendar.php	oogleCalendar.php, google-calendar.php
内容 (interface)	形容词或名词	Authenticatable	uthenticationInterface, IAuthentication
Trait	使用形容词	Notifiable	otificationTrait

尽可能使用简短且可读性更好的语法

例子:

```
$request->session()->get('cart');
$request->input('name');
```

更优的写法:

```
session('cart');
$request->name;
```

更多示例:

常规写法	更优雅的写法
Session::get('cart')	session('cart')
$request->session()->get('cart')	session('cart')
Session::put('cart', $data)	session(['cart' => $data])
$request->input('name'), Request::get('name')	request->name, request('name')
return Redirect::back()	return back()
is_null($object->relation) ? null : $object->relation->id	optional($object->relation)->id
return view('index')->with('title', $title)->with('client', $client)	return view('index', compact('title', 'client'))
$request->has('value') ? $request->value : 'default';	request->get('value', 'default')
Carbon::now(), Carbon::today()	now(), today()
App::make('Class')	app('Class')
->where('column', '=', 1)	->where('column', 1)
->orderBy('created_at', 'desc')	->latest()
->orderBy('age', 'desc')	->latest('age')
->orderBy('created_at', 'asc')	->oldest()
->select('id', 'name')->get()	->get(['id', 'name'])
->first()->name	->value('name')

使用IOC容器来创建实例 而不是直接new一个实例

创建新的类会让类之间的更加耦合，使得测试越发复杂。请改用IoC容器或注入来实现。

例子:

```
$user = new User;
$user->create($request->validated());
```

更优的写法:

```
public function __construct(User $user)
{
    $this->user = $user;
}

....

$this->user->create($request->validated());
```

避免直接从 .env 文件里获取数据

将数据传递给配置文件，然后使用config（）帮助函数来调用数据

例子:

```
$apiKey = env('API_KEY');
```

更优的写法:

```
// config/api.php
'key' => env('API_KEY'),

// Use the data
$apiKey = config('api.key');
```

使用标准格式来存储日期，用访问器和修改器来修改日期格式

例子:

```
{{ Carbon::createFromFormat('Y-d-m H-i', $object->ordered_at)->toDateString() }}
{{ Carbon::createFromFormat('Y-d-m H-i', $object->ordered_at)->format('m-d') }}
```

更优的写法:

```
// Model
protected $dates = ['ordered_at', 'created_at', 'updated_at'];
public function getSomeDateAttribute($date)
{
    return $date->format('m-d');
}

// View
{{ $object->ordered_at->toDateString() }}
{{ $object->ordered_at->some_date }}
```

其他的一些好建议

永远不要在路由文件中放任何的逻辑代码。

尽量不要在Blade模板中写原始 PHP 代码。

原文作者： ikidnapmyself

原文链接：https://github.com/alexeymeze...