# Laravel笔记

1. 安装
   1. composer安装

composer create-project --prefer-dist laravel/laravel blog "5.2.\*"

1. 配置
   1. 使用config()获取你的设置值，设置值可以用[点]语法来获取：文件名.选项名称；

$value = config(‘app.timezone’);// show “UTC”

若要在运行期间修改设置，需要给config()传一个数组；

config([‘app.timezone’ => ‘Amercia/Chicago’]);

* 1. 维护模式

当你的应用处于维护模式时，所有传递至应用程序的请求都会显示出一个自定义视图。

启用维护模式：php artisan down

关闭维护模式：php artisan up

维护模式的默认模板在：resources/views/errors/503.blade.php

* 1. 数据库迁移

Laravel 的数据库迁移提供了一个简单的方式，使用流畅、一目了然的 PHP 代码来定义数据表的结构与修改。你无需再告诉团队成员要手动增加字段至他们本机的数据库中，他们只需要运行你提交到版本控制的迁移，即可应用数据库修改。

使用Artisan命令：

php artisan make:migration create\_tasks\_table --create=tasks//创建数据库迁移文件，里面等于是建表语句，默认放在database/migrations 目录中。

php artisan migrate //运行迁移，等于执行上面的迁移文件

* 1. Eloquent 模型

每个 Eloqunet 模型会直接对应一张数据表。

使用make:model命令来生成tasks数据表的模型Task（是个空模型，放在app目录下）：

php artisan make:model Task //默认创建的Task模型对应tasks数据表

1. 路由
   1. 响应多个HTTP动作的路由

Route::match(['get', 'post'], '/', function () {

//

});

//匹配所有请求方式

Route::any('foo', function () {

//

});

* 1. 路由参数
     1. 必选参数

Route::get('posts/{post}/comments/{comment}', function ($postId, $commentId) {

//

});

路由参数都放在花括号内，参数不能含‘-’；

* + 1. 可选参数

Route::get('user/{name?}', function ($name = 'John') {

return $name;

});

在参数后面加‘？’表示可选；

* + 1. 正则限制参数

//使用where（）方法，前面是参数名，后面是正则；若同时限制多个参数，采用关联数组方式。

Route::get('user/{id}', function ($id) {

//

})

->where('id', '[0-9]+');

Route::get('user/{id}/{name}', function ($id, $name) {

//

})

->where(['id' => '[0-9]+', 'name' => '[a-z]+']);

* + 1. 全局限制

若希望路由参数一直遵循正则表达式，在RouteServiceProvider 的 boot 方法里定义这些模式：

public function boot(Router $router)

{

$router->pattern('id', '[0-9]+');

parent::boot($router);

}

模式一旦被定义，便会自动应用到所有使用该参数名称的路由上

* + 1. 命名路由

------//为这个路由指定一个名字；

Route::get('user/profile', ['as' => 'profile', function () {

//

}]);

------//也可以为控制器的方法指定路由名字：

Route::get('user/profile', [

'as' => 'profile', 'uses' => 'UserController@showProfile'

]);

------//还可以在路由定义后链式调用name()方法

Route::get('user/profile', 'UserController@showProfile')->name('profile');

一旦你在指定的路由中分配了名称，则可通过 route 函数来使用路由名称生成 URLs 或重定向：

$url = route('profile');

$redirect = redirect()->route('profile');

若路由中带参数，可以用 route(‘profile’,[‘id’=>12]); 直接将参数加入到url中；

-------//在路由群组的属性数组中指定 as 关键字，会为群组中所有路由设置相同的前缀

Route::group(['as' => 'admin\_'], function () {

Route::get('dashboard', ['as' => 'dashboard', function () {

// 路由名称为「admin\_dashboard」

return route(‘admin\_dashboard’);

}]);

});

* + 1. 路由前缀

为路由指定的URI加上前缀

Route::group(['prefix' => 'admin'], function () {

Route::get('users', function () {

// 符合「/admin/users」URL

});

});

* 1. CSRF 保护

（1）

Laravel 会自动生成一个 CSRF token 给每个用户的 Session。该 token 用来验证登录用户和发起请求者是否是同一人。可以使用 csrf\_field 辅助函数来生成一个包含 CSRF token 的 \_token 隐藏表单字段：

<?php echo csrf\_field(); ?>

csrf\_field 辅助函数会生成以下的 HTML：

<input type="hidden" name="\_token" value="<?php echo csrf\_token(); ?>">

在Blade模板中使用 {{csrf\_field}}

不需要手动验证请求的csrf token, VerifyCsrfToken HTTP 中间件 将自动验证请求与 session 中的 token 是否相符。

（2）

并非所有请求都需要避免CSRF攻击，比如第三方请求。此时需要在：app/Http/Middleware/VerifyCsrfToken.php中间件中将需要排除的url添加到$except数组中即可

（3）

如果使用Ajax提交POST表单，又该如何处理呢？我们可以将Token设置在meta中：

<meta name="csrf-token" content="{{ csrf\_token() }}">

然后在全局Ajax中使用这种方式设置X-CSRF-Token请求头并提交：

$.ajaxSetup({

headers: {

'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')

}

});

Laravel的VerifyCsrfToken中间件会检查X-CSRF-TOKEN请求头，如果该值和Session中CSRF值相等则验证通过，否则不通过。

(4)原理：首先在laravel开启session时，会生成一个token放在session中，（ Illuminate\Session\Store.php第90行start方法）

然后看VerifyCsrfToken.php的handle方法，首先是isReading方法判断请求方式，如果请求方法是HEAD、GET、OPTIONS其中一种，则不做CSRF验证；

然后，再通过shouldPassThrough方法判断请求路由是否在$excpet属性数组中进行了排除，如果做了排除也不做验证；

最后通过tokensMatch方法判断请求参数中的CSRF TOKEN值和Session中的Token值是否相等，如果相等则通过验证，否则抛出TokenMismatchException异常。

注：tokensMatch方法首先从Request中获取\_token参数值，如果请求中不包含该参数则获取X-CSRF-TOKEN请求头的值，如果该请求头也不存在则获取X-XSRF-TOKEN请求头的值，需要注意的是X-XSRF-TOKEN请求头的值需要调用Encrypter的decrypt方法进行解密。

* 1. 路由分组
     1. 中间件

生成中间件的命令：

php artisan make:middleware TestMiddleware

而后把中间件添加到/app/Http/Kernal.php文件的$routeMiddleware属性：

protected $routeMiddleware = [

'test' => \App\Http\Middleware\TestMiddleware::class,

];之后即可在路由中使用中间件。

若要让中间件在全局有效，每次请求都会被调用，则将中间件追加到$middleware属性数组（在路由中不用middleware 指明了）

使用：

Route::get('admin/profile', ['middleware' => 'auth', function () {}]);

或Route::get('/', ['middleware' => ['first', 'second'], function () {}]);

或Route::get('/', function () {})->middleware(['first', 'second']);

自定义中间件类只需要定义一个handle方法即可，将主要的业务逻辑定义在该方法中。

若想要在请求处理前执行业务逻辑，则在$next闭包执行前执行业务逻辑。

public function handle($request, Closure $next)

{

// 执行业务逻辑操作

return $next($request);

}

若想要在请求处理后执行中间件业务逻辑，则在$next闭包执行后操作

public function handle($request, Closure $next)

{

$response = $next($request);

// 执行动作

return $response;

}

中间件参数：

public function handle($request, Closure $next, $gender)

{

if($request->input('age')>=18 && $gender==$request->input('gender')){

return $next($request);

}else{

return redirect()->route('refuse');

}

}

路由配置：

Route::group(['middleware'=>'test:male'],function(){

Route::get('/write/laravelacademy',function(){

//使用Test中间件

});

Route::get('/update/laravelacademy',function(){

//使用Test中间件

});

});

访问[www.blog.com/se?age=19&gender=male](http://www.blog.com/se?age=19&gender=male)时会通过

* + 1. 命名空间

默认情况下，routes.php中的定义的控制器位于App\Http\Controllers命名空间下，所以如果要指定命名空间，只需指定App\Http\Controllers之后的部分即可：

Route::group(['namespace' => 'DOCS'], function()

{

// 控制器在 "App\Http\Controllers\LaravelAcademy\DOCS" 命名空间下

});