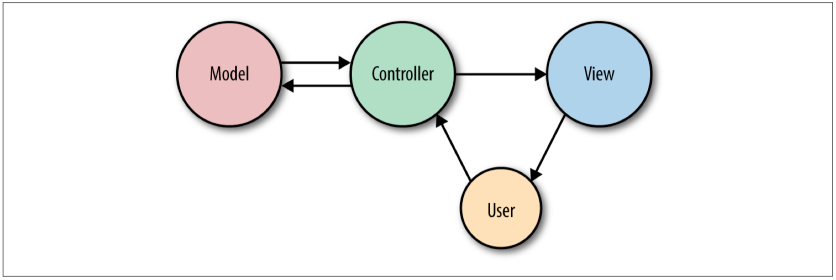
**控制器概述**

到目前为止，我们定义的所有路由都是基于闭包函数实现的，前面已经提到过，随着应用体量的增长，不可能将所有路由都定义在单个文件中，且对于复杂的业务逻辑，闭包函数也不足以支撑，所以和其他 Web 应用框架一样，我们还可以通过控制器来定义路由。

说到这里，我们就不得不提一下 MVC 设计模式，这个模式最早在 Ruby On Rails 中引入，然后被基本上所有的 Web 框架所借鉴和遵循，Laravel 也不例外。在 MVC 模式中，M 代表模型（Model），V 代表视图（View），C 代表控制器（Controller），控制器负责组织路由和业务逻辑（当然，对于更加复杂的业务逻辑还会引入 Service 层），模型类负责底层数据存取与处理，而视图层负责数据渲染与页面交互。对于一些 CRUD 操作（数据库增删改查操作的简写）来说，常见的业务逻辑也就是从模型类获取数据并将其渲染到页面，或者从页面获取用户提交数据并将其存储到模型类：



将所有业务逻辑一股脑放到控制器听起来挺不错，但是控制器更适合承担的角色其实是负责对 HTTP 请求进行路由，因为还有很多其他访问应用的方式，比如 Artisan 命令、队列、调度任务等等，控制器并非唯一入口，所以不适合也不应该将所有业务逻辑封装于此，过度依赖控制器会对以后应用的扩展带来麻烦。所以，你应该具备这样的意识：控制器的主要职责就是获取 HTTP 请求，进行一些简单处理（如验证）后将其传递给真正处理业务逻辑的职能部门，如 Service。

注：当然，如果是非常简单的应用，比如只是简单的数据库增删改查或数据渲染，放到控制器里面也无妨，但是如果后续需要调用控制器方法才能完成某个功能，那么是时候将这个控制器方法里的业务逻辑拆分到 Service 里面了。

**控制器入门**

具备以上理论知识后，下面我们来创建一个控制器，我们可以通过 [Artisan 命令](https://laravelacademy.org/post/9562.html)快速创建一个控制器：

php artisan make:controller TaskController

该命令会在 app/Http/Controllers 目录下创建一个新的名为 TaskController.php 的文件，默认生成的控制器代码如下：

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class TaskController extends Controller

{

//

}

我们为该控制器添加一个简单的 home() 动作方法：

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class TaskController extends Controller

{

public function home()

{

return 'Hello, World!';

}

}

然后我们来定义一个指向该控制器动作的路由：

Route::get('/task', 'TaskController@home');

这样，我们访问 /task 就能看到「Hello, World!」了。

注：这里需要注意的是控制器 TaskController 的完整命名空间是 App\Http\Controllers\TaskController，但是我们在定义路由的时候只用了类名，关于这一点我们在[上一篇教程的命名空间前缀部分](https://laravelacademy.org/post/9612.html" \l "toc_3)已经提到过，默认情况下，如果没有指定完整的命名空间，那么路由文件 web.php 中所有控制器都位于 App\Http\Controllers 命名空间下，所以在定义控制器路由的时候可以省略这个命名空间前缀。

实际开发中，很少有返回字符串的场景，常见的控制器方法代码如下：

public function index()

{

return view('task.index')

->with('tasks', Task::all());

}

这段代码的含义是通过 Task::all() 查询所有任务数据，并将其赋值给 tasks 变量在视图 task.index （resources/views/task/index.blade.php）中渲染出来。关于视图和模型我们后面会单独讲解。

**获取用户输入**

除了数据渲染之外，还可以在控制器中获取用户输入并进行处理，下面我们来看两个例子：

Route::get('task/create', 'TaskController@create');

Route::post('task', 'TaskController@store');

我们通过 create() 方法来渲染一个任务提交表单， 然后通过 store() 方法来存储提交的任务数据。关于表单渲染我们放到后面去讨论，现在我们直接跳到表单数据处理上，所以编写 store() 方法：

public function store(Request $request)

{

$task = new Task();

$task->title = $request->input('title');

$task->description = $request->input('description');

$task->save();

return redirect('task'); // 重定向到 GET task 路由

}

这里我们用到了 Eloquent 模型类 Task 和重定向方法 redirect()，后续会一一详述，现在只关注用户数据处理的逻辑：我们将用户提交数据收集起来，保存到 Task 模型类，然后将用户重定向到显示所有任务的页面。这里我们通过 $request 对象来获取用户输入，此外还可以通过 Input [门面](https://laravelacademy.org/post/9536.html) 获取用户输入：

$task->title = Input::get('title');

注：使用这种方式需要引入 Input 门面：use Illuminate\Support\facades\Input

门面仅仅是静态代理，底层调用的还是 $request->input 方法，语法糖而已，建议大家还是用 $request 来获取。

使用上述获取方式可以获取用户提供的任何输入数据，不管是查询字符串还是表单字段。

**依赖注入**

正如前面介绍的 Input 门面一样，Laravel 中的门面为 Laravel 代码库中的大部分类提供了简单的接口调用，通过门面你可以轻松从当前获取各种请求数据，比如用户输入、Session、Cookie 等，但不是所有的类都有对应的门面（当前的映射关系可以查看[门面列表](https://laravelacademy.org/post/9536.html" \l "toc_6)），对于这些类提供的方法我们可以通过更底层的依赖注入来调用，本质上来看，门面仅仅是一种设计模式，是对底层复杂 API 的上层静态代理，主要目的在于简化代码调用，所以可以用门面调用的方法肯定可以用依赖注入来实现，而可以通过依赖注入实现的功能不一定可以通过门面来调用，除非你自定义实现这个门面。

提到依赖注入，就绕不开[服务容器](https://laravelacademy.org/post/9534.html)，关于服务容器后面我们会单独讲解，而现在你只需了解服务容器是一个绑定多个接口与具体服务实现类的容器，而依赖注入则是在代码编写时以接口（或者叫做类型提示）方式作为参数，不必传入具体实现类，在代码运行时会根据配置从服务容器获取接口对应的实现类执行具体的接口方法，从而极大提高了代码的可维护性和可扩展性。

在 Laravel 中所有的控制器方法（包括构造函数）都会在服务容器中进行解析，这意味着所有方法中传入的可以被容器解析的接口/类型提示对应服务实现都会被自动注入，我们将这个过程称之为依赖注入。我们上面演示的通过 $request 对象获取用户请求数据就是采用依赖注入的方式。

在日常开发中，推荐大家使用依赖注入而非门面来获取用户输入数据，除此之外，还可以通过 $request 对象获取 Session、Cookie 数据。

**资源控制器**

有时候在编写控制器时命名方法名称可能是最困难的，好在 Laravel 为常见的 REST/CRUD 控制器（在 Laravel 中称之为「资源控制器」）提供了一套约定规则，并为此提供了相应的 Artisan 生成器和路由定义方法，从方便我们一次为所有控制器方法定义路由。

首先，我们使用这个 Artisan 生成器来生成一个资源控制器（在之前命名后加上 --resource 选项）：

php artisan make:controller PostController --resource

现在，打开 app/Http/Controllers/PostController.php 文件，即可看到 PostController 代码：

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class PostController extends Controller

{

/\*\*

\* Display a listing of the resource.

\*

\* @return \Illuminate\Http\Response

\*/

public function index()

{

//

}

/\*\*

\* Show the form for creating a new resource.

\*

\* @return \Illuminate\Http\Response

\*/

public function create()

{

//

}

/\*\*

\* Store a newly created resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @return \Illuminate\Http\Response

\*/

public function store(Request $request)

{

//

}

/\*\*

\* Display the specified resource.

\*

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function show($id)

{

//

}

/\*\*

\* Show the form for editing the specified resource.

\*

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function edit($id)

{

//

}

/\*\*

\* Update the specified resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function update(Request $request, $id)

{

//

}

/\*\*

\* Remove the specified resource from storage.

\*

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function destroy($id)

{

//

}

}

**资源控制器方法列表**

以上 PostController 控制器的每个方法都有对应的请求方式、路由命名、URL、方法名和业务逻辑约定。

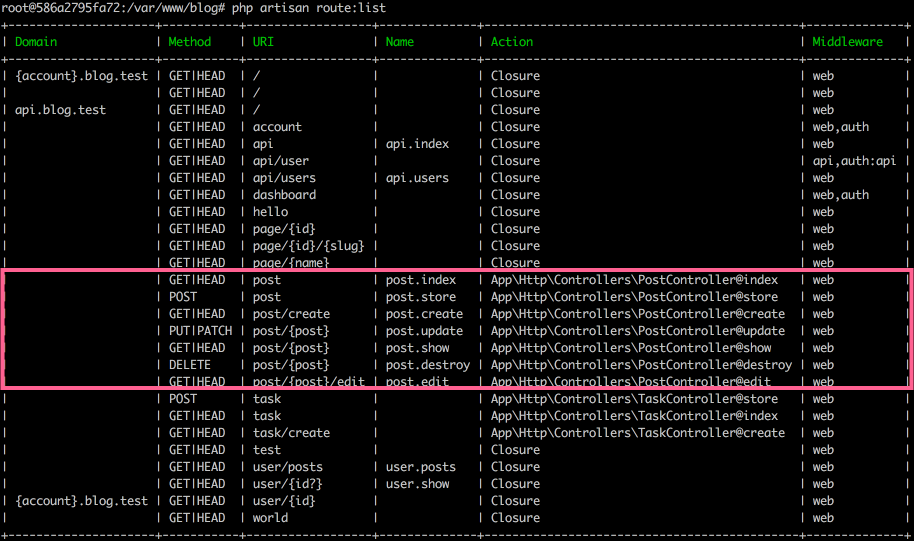
| **HTTP请求方式** | **URL** | **控制器方法** | **路由命名** | **业务逻辑描述** |
| --- | --- | --- | --- | --- |
| GET | post | index() | post.index | 展示所有文章 |
| GET | post/create | create() | post.create | 发布文章表单页面 |
| POST | post | store() | post.store | 获取表单提交数据并保存新文章 |
| GET | post/{post} | show() | post.show | 展示单个文章 |
| GET | post/{id}/edit | edit() | post.edit | 编辑文章表单页面 |
| PUT | post/{id} | update() | post.update | 获取编辑表单输入并更新文章 |
| DELETE | post/{id} | destroy() | post.desc | 删除单个文章 |

**绑定资源服务器**

通过上面的表格已经了解了 Laravel 中对资源路由的命名约定，Laravel 还为我们提供了一个 Route::resource 方法用于一次注册包含上面列出的所有路由，并且遵循上述所有约定：

Route::resource('post', 'PostController');

你可以通过 Artisan 命令 php artisan route:list 查看应用的所有路由：



我们可以以 post.show 路由为例演示下资源路由的访问：

public function show($id)

{

return 'Post ' . $id . ' Link: ' . route('post.show', [$id]);

}

在浏览器中访问 http://blog.test/post/1，页面显示如下：

Post 1 Link: http://blog.test/post/1