对任何一个 Web 应用框架而言，通过 HTTP 协议处理用户请求并返回响应都是核心必备功能，也就是说，对于我们学习和使用一个 Web 框架，第一件要做的事情就是定义应用路由，否则，将无法与终端用户进行交互。而我们的 Laravel 从入门到精通系列教程之旅也将从路由开始，在这篇真正意义上的开篇教程中，我们将学习如何定义路由，然后将其指向要执行的代码，并处理各种路由需求。

**路由入门**

在 Laravel 应用中，定义路由有两个入口，一个是 routes/web.php，用于处理终端用户通过 Web 浏览器直接访问的请求，另一个是 routes/api.php，用于处理其他接入方的 API 请求（通常是跨语言、跨应用的请求）。在本章中，我们将主要聚焦于 routes/web.php，关于 routes/api.php 将会在后面编写 API 章节中重点介绍。

定义路由最简单的方式就是在 routes/web.php 中定义一个路径以及一个映射到该路径的[闭包函数](http://php.net/manual/zh/class.closure.php)：

// routes/web.php

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

return 'Hello, World!';

});

这样，当我们访问应用首页 http://blog.test 时，就可以看到页面显示 Hello, World! 这一行字符串。这就是一个最简单的 Laravel 路由定义，但是涵盖了一个 Web 框架的基本功能：处理请求，返回响应。

注：这里需要注意的是，我们并没有通过 echo 或 print 显示输出内容，而是通过 return 将其返回，Laravel 会通过内置的响应栈和中间件对返回内容进行处理。

很多简单的静态 Web 站点通过这种最基本的路由定义就可以完成了，比如一些企事业单位宣传网站，只有一些静态页面，通过几个 GET 路由以及视图模板就可以搞定了：

// 首页

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

return view('welcome');

});

// 关于我们

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

return view('about');

});

// 产品页

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

return view('products');

});

// 服务页

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

return view('services');

});

使用 Laravel 开发静态站点，就是这么简单！

**路由动作**

你可能已经注意到我们在上面的路由定义中使用了 Route::get，这种语法的含义是只匹配 GET 请求路由，那如果提交的是 POST 请求，或者 PUT、DELETE 请求呢？Laravel 框架也为我们提供了相应的路由定义方法：

Route::post('/', function () {});

Route::put('/', function () {});

Route::delete('/', function () {});

此外，还可以通过 Route::any 定义一个可以捕获任何请求方式的路由：

Route::any('/', function () {});

从安全角度说，并不推荐上述这种路由定义方式，但是兼顾到便利性，我们可以通过 Route::match 指定请求方式白名单数组，比如下面这个路由可以匹配 GET 或 POST 请求：

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

**复杂业务逻辑处理**

当然，传递闭包并不是定义路由的唯一方式，闭包简单快捷，但是随着应用体量的增长，将日趋复杂的业务逻辑全部放到路由文件中显然是不合适的，另外，通过闭包定义路由也无法使用路由缓存（稍后会讲到）从而优化应用性能。对于稍微复杂一些的业务逻辑，我们可以将其拆分到控制器方法中实现，然后在定义路由的时候使用控制器+方法名来取代闭包函数：

Route::get('/', 'WelcomeController@index');

这段代码的含义是将针对 / 路由的 GET 请求传递给 App\Http\Controllers\WelcomeController 控制器的 index 方法进行处理。你可以将之前定义的闭包函数内的代码移植到 index 方法中，效果完全一样（WelcomeController 需要自己创建，我们会在后面的控制器部分讲解如何快速创建），现在先略过不表。

**路由参数**

如果你定义的路由需要传递参数，只需要在路由路径中进行标识并将其传递到闭包函数即可：

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

return "用户ID: " . $id;

});

这样，当你访问 http://blog.test/user/1000 的时候，就可以在浏览器看到 用户ID: 1000 字符串。

此外，你还可以定义可选的路由参数，只需要在参数后面加个 ? 标识符即可，同时你还可以为可选参数指定默认值：

Route::get('user/{id?}', function ($id = 1) {

return "用户ID: " . $id;

});

这样，如果不传递任何参数访问 http://blog.test/user，则会使用默认值 1 作为用户 ID。

更高级的，你还可以为路由参数指定正则匹配规则：

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

return '页面ID: ' . $id;

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

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

return '页面名称: ' . $name;

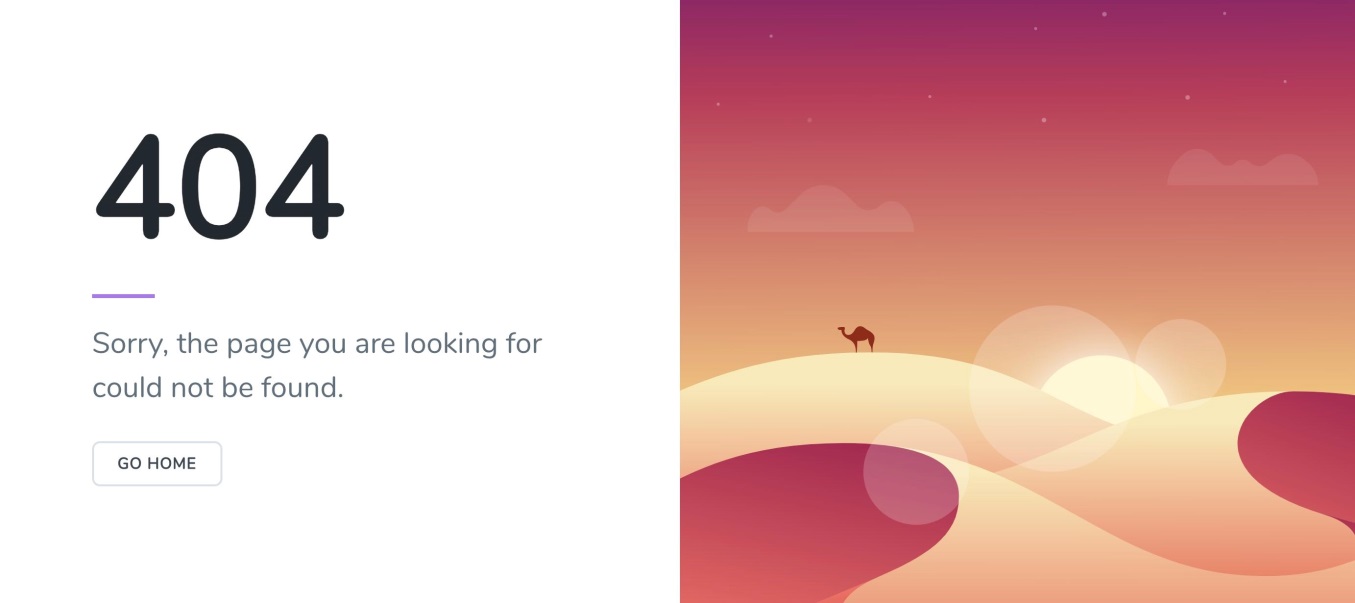
})->where('name', '[A-Za-z]+');

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

return $id . ':' . $slug;

})->where(['id' => '[0-9]+', 'slug' => '[A-Za-z]+']);

如果传入的路由参数与指定正则不匹配，则会返回 404 页面：



**路由命名**

在应用其他地方引用路由的最简单的方式就是通过定义路由的第一个路径参数，你可以在视图中通过辅助函数 url() 来引用指定路由，该函数会为传入路径加上完整的域名前缀，所以 url('/') 对应的输出是 http://blog.test。你可以在视图文件中这么使用：

<a href="{{ url('/') }}">

此外，Laravel 还允许你为每个路由命名，这样一来，不必显式引用路径 URL 就可以对路由进行引用，这样做的好处是你可以为一些复杂的路由路径定义一个简单的路由名称从而简化对路由的引用，另一个更大的好处是即使你调整了路由路径（在复杂应用中可能很常见），只要路由名称不变，那么就无需修改前端视图代码，提高了系统的可维护性。

路由命名很简单，只需在原来路由定义的基础上以方法链的形式新增一个 name 方法调用即可：

Route::get('user/{id?}', function ($id = 1) {

return "用户ID: " . $id;

})->name('user.profile');

前端视图模板中可以通过辅助函数 route 并传入路由名称（如果有路由参数，则以数组方式作为第二个参数传入）来引用该路由：

<a href="{{ route('user.profile', ['id' => 100]) }}">

// 输出：http://blog.test/user/100

如果没有路由参数，通过 route('user.profile') 引用即可。此外，我们还可以简化对路由参数的传递，比如上例可以简化为：

<a href="{{ route('user.profile', [100]) }}">

这样调用的话，数组中的参数顺序必须与定义路由时的参数顺序保持一致，而使用关联数组的方式传递参数则没有这样的约束。

注：在实际开发过程中，推荐使用路由命名来引用路由。