——高性能PHP应用开发之

第9讲 请求和响应

目录

CONTENTS

目录

CONTENTS

Laravel请求对象

- 请求对象类：Illuminate\Http\Request
  - ✓ 使用方法：**依赖注入机制**（通过参数形式）
  - ✓ 若有路由参数，写在$request之后

```php
use Illuminate\Http\Request;

class UserController extends Controller
{
    public function index(Request $request)
    {
        dump($request);
    }
}
```

- 请求门面类：Illuminate\Support\Facades\Request
  - ✓ 使用方法：**静态方法**

```php
use Illuminate\Support\Facades\Request;
class UserController extends Controller
{
    public function index()
    {
        dump(Request::all());
    }
}
```

获取请求路径和请求方法

■ 获取请求路径：

✓ path( ) 方法：返回请求的路径信息

✓ url( ) 方法：返回不带查询字符串的完整URL

✓ fullUrl( ) 方法：返回包含查询字符串的完整URL

✓ is( ) 方法：允许验证进入的请求是否与给定模式匹配

```
if($request->is('admin/*')){
    //
}
```

```
dump($request->path());
dump($request->url());
dump($request->fullUrl());
```

```
"user"
```

```
"http://127.0.0.1:8000/user"
```

```
"http://127.0.0.1:8000/user?id=3"
```

■ 获取请求方法：

✓ method( )方法：返回 HTTP 请求方式

✓ isMethod( )方法：验证 HTTP 请求方式是否匹配给定字符串

```
dump($request->method());
dump($request->isMethod('get'));
```

获取请求参数

■ 获取请求参数：

- ✓ 注意：无论是GET参数（url参数或表单参数）还是POST参数（任何HTTP请求参数），均可以被请求对象获取到。
- ✓ all( ) 方法：以数组格式获取所有请求参数
- ✓ input($param) 方法：获取给定参数$param所对应的请求参数（若$param为空，以数组返回所有请求参数)
- ✓ has( ) 方法：判断某一个请求参数是否存在
- ✓ only( )方法：只获取指定的请求参数
- ✓ except( )方法：获取指定参数之外的所有请求参数

```
dump($request->all());
dump($request->input('id'));
dump($request->input());
dump($request->has('name'));
dump($request->only(['name']));
dump($request->except(['name']));
```

获取上一次请求参数

Laravel 允许你在两次请求之间保存输入数据，这个特性在检测校验数据失败后需要重新填充表单数据时很有用。

■ 存储一次性数据：

✓ 简单存储：flash( )方法、flashOnly( )方法、flashExcept( )方法

✓ 存储并重定向： redirect( )->withInput( )

■ 使用一次性数据：

✓ 获取上一次数据：old( )方法

```php
public function index(Request $request)
{
    // 存储一次性flash数据
    $request->flash('id');
    // 重定向
    return "<a href='/user/test'>jump to test</a>";
}

public function test(Request $request)
{
    dd($request->old());
}
```

Laravel响应对象

- Laravel中的响应对象分为三类：

✓ **Illuminate\Http\Response** 类：基本响应类，可以使用 **依赖注入或门面** 方式访问

✓ Illuminate\Http\JsonResponse类 和 Illuminate\Http\RedirectResponse类：

- JsonResponse类：响应Json消息
- **RedirectResponse**类：重定向响应

✓ **Illuminate\Routing\ResponseFactory** 类：响应工厂类，提供一系列方法响应不同类型内容

Illuminate\Http\Response 类

- 直接返回响应对象：
  - ✓ 返回字符串：return ‘hello’；
  - ✓ 返回视图： return view(‘test’);
- 依赖注入使用响应对象：
  - ✓ setContent( ) 方法，设置响应内容
- 门面Facades使用响应对象：
  - ✓ Response::make( ) 创建响应对象并设置内容
- 响应对象的其它方法：
  - ✓ header( )方法 和 withHeaders( )：设置响应消息头
  - ✓ cookie( )方法：设置cookie消息

```
return view('index');
```

```
public function index(Response $response)
{
    $response->setContent('setContent');
    return $response;
}
```

```
return Response::make('hello response');
```

重定向响应对象

- 获取重定向响应对象：
  - ✓ **redirect( )**辅助函数、**back( )**辅助函数
  - ✓ 直接构造： new RedirectResponse( );
- **Redirector类**实例对象重定向方法：$redirector = redirect( );
  - ✓ 重定向到URL路径：$redirector->**to**(URL路径);
  - ✓ 重定向到外网地址：$redirector->**away**(外网URL);
  - ✓ 重定向到命名路由：$redirector->**route**(路由名称);
  - ✓ 重定向到控制器动作：$redirector->**action**(控制器动作);
    - 注意，不能为Restful控制器的自定义动作
  - ✓ 返回上一个页面：$redirector->**back**( );
  - ✓ 刷新当前页面：$redirector->refresh( );

```
return new RedirectResponse('/photo');
```

```
return redirect()->to('photo');
```

```
return redirect()->away('http://www.baidu.com');
```

```
return redirect()->route('photo.index');
```

```
return redirect()->action('UserController@test');
```

```
return redirect()->back();
```

响应其它类型

- 响应JSON数据
- ✓ **json** 方法会自动将 Content-Type 头设置为 application/json，并使用 PHP 函数 json_encode 方法将给定数组转化为 JSON。
- 文件下载
- ✓ **download** 方法用于生成强制用户浏览器下载给定路径文件的响应。download 方法接受文件名作为第二个参数，该参数决定用户下载文件的显示名称，你还可以将 HTTP 头信息作为第三个参数传递到该方法)。
- 直接响应文件内容
- ✓ **file**方法可用于直接在用户浏览器显示文件，例如图片或PDF，而不需要下载，该方法接收文件路径作为第一个参数，头信息数组作为第二个参数。

目录

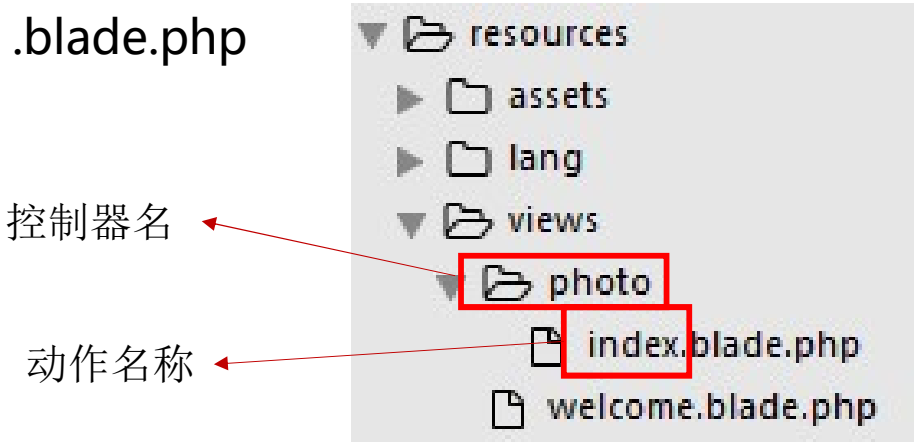
CONTENTS

视图

■ 控制器动作和视图

- ✓ 视图文件和控制器动作的关系：一般情况下，控制器的一个动作对应一个视图文件；另一方面，一个视图文件也可以被多个动作所加载
- ✓ 视图文件目录：/resources/views/ 目录下；该目录下最好针对不同控制器创建不同的子目录，在子目录中存放某一个动作所对应的视图文件（非强制性要求）
- ✓ 视图文件名：动作名.blade.php，注意扩展名为 .blade.php
- ✓ 控制器中加载视图方法：
- • 使用辅助函数 view( ) 加载指定的视图文件

```
public function index() {
    return view('welcome');
}
```



视图对象

- **使用view( )辅助函数获取视图对象**
  - ✓ 不传入参数，获取 Illuminate\View\Factory对象：$factory = view( );
  - ✓ 传入参数，获取 Illuminate\View\View对象：$view = view("hello");
- **使用Facades门面方式使用视图对象**
  - ✓ View::make( )、View::share( )、View::composer( ) 等
- **借助IOC容器获取视图对象**
  - ✓ **app( )**：获取IOC容器对象
  - ✓ app(参数)：获取IOC容器对象中绑定的依赖类实例对象
  - ✓ $view = app(“view”);

视图

■ **向视图中传递数据：使用辅助函数 view( )**

✓ 传递单个数据：view("index")->with(key, value);

```
public function index() {
    return view('index')->with('name', 'test');
}
```

✓ 数组类型：直接在view( )函数中给出第2个参数（数组数据）表示待传入的数据

```
public function index() {
    return view('index', [
        'name'  =>  'test',
        'age'   =>  18,
    ]);
}
```

视图

- **在视图中显示数据**

✓ 直接使用原生PHP代码

```
name => <?php echo $name; ?>
age => <?php echo $age; ?>
```

✓ 使用Blade模板语法

```
name => {{ $name }}
age => {{ $age }}
```

视图间共享数据

- **视图间共享简单数据**
- ✓ 在ServiceProvider中注册数据
  - 可以在 AppServiceProvider 中绑定，也可以在自定义的Provider中绑定

```php
// 视图间共享数据
view()->share('sitename', 'Laravel学院');
View::share('sitename', 'Laravel学院');
```

- ✓ 在视图文件中使用数据
- **视图组件共享数据：View Composer**
- ✓ 在ServiceProvider中绑定composer接口
  - 可以绑定闭包函数
  - 可以绑定类文件（必须实现compose方法）

```php
// 视图composer
view()->composer('hello', function($view) {
    $view->with('user', [
        'name'   =>  'test',
        'age'    =>  18
    ]);
});
```

- ✓ 在视图文件中使用数据

```php
view()->composer('*', 'App\Http\ViewComposers\UserViewComposer');
```

目录

CONTENTS

Blade模板

- Blade模板 是 Laravel 提供的一个非常简单但很强大的模板引擎，不同于其他流行的 PHP 模板引擎，Blade 在视图中并不约束你使用 PHP 原生代码。所有的 Blade 视图都会被编译成原生 PHP 代码并缓存起来直到被修改，这意味着对应用的性能而言 Blade 基本上是零开销。Blade 视图文件使用 .blade.php 文件扩展并存放在 resources/views 目录下。

- **显示数据**

✓ 显示简单数据

✓ 输出函数：直接书写函数调用即可

✓ 默认值输出

```
name => {{ $name }}
age => {{ $age }}
```

```
The current UNIX timestamp is {{ time() }}.
```

```
{{ $name or 'Default' }}
```

Blade模板

- **模板继承**
  - ✓ @section ：占位符
  - ✓ @yield ： 替换模板输出
  - ✓ @extends ： 模板继承
  - ✓ @parent ： 输出父视图中指定内容

```
<html>
    <head>
        <title>App Name - @yield('title')</title>
    </head>
    <body>
        @section('sidebar')
            This is the master sidebar.
        @show

        <div class="container">
            @yield('content')
        </div>
    </body>
</html>
```

```
@extends('layouts.master')

@section('title', 'Page Title')

@section('sidebar')
    @parent
    <p>This is appended to the master sidebar.</p>
@endsection

@section('content')
    <p>This is my body content.</p>
@endsection
```

Blade模板

```
@section('content')
默认内容
@stop
```

```
@extends('layout')

@section('content')
    @parent
    新的内容
@stop
```

- **@section结束标签的区别：**
- ✓ **@show** ： 执行到此处时将该 section 中的内容输出到页面
- ✓ **@stop** ：只是进行内容解析，并且不再处理当前模板中后续对该section的处理
- ✓ @endsection ：已经被弃用，用 @stop 代替

- **@yield 和 @section 区别：**
- ✓ @yield 不能被子视图继承（即父视图内容会丢失）
- ✓ @yield 是单标签，不能有结束标签
- ✓ @section 可以被子视图继承（使用 @parent ）

```
@yield('title', '默认标题')

@section('content')
默认内容
@show
```

```
@extends('layout')

@section('title')
    @parent
    新的标题
@stop
@section('content')
    @parent
    新的内容
@stop
```

Blade模板

- **流程控制**

✓ if语句：可以使用 @if , @elseif ，@else 和 @endif 来构造 if 语句，这些指令函数和 PHP 的相同

```
@if (count($records) === 1)
    I have one record!
@elseif (count($records) > 1)
    I have multiple records!
@else
    I don't have any records!
@endif
```

✓ Blade 还提供了 @unless 指令

```
@unless (Auth::check())
    You are not signed in.
@endunless
```

Blade模板

- **流程控制**

✓ 循环语句

```
@for ($i = 0; $i < 10; $i++)
    The current value is {{ $i }}
@endfor

@foreach ($users as $user)
    <p>This is user {{ $user->id }}</p>
@endforeach

@forelse ($users as $user)
    <li>{{ $user->name }}</li>
@empty
    <p>No users</p>
@endforelse

@while (true)
    <p>I'm looping forever.</p>
@endwhile
```

✓ 在循环的时候可以使用$loop变量获取循环信息，例如是否是循环的第一个或最后一个迭代。

```
@foreach ($users as $user)
    @if ($user->type == 1)
        @continue
    @endif

    <li>{{ $user->name }}</li>

    @if ($user->number == 5)
        @break
    @endif
@endforeach
```

Blade模板

- **流程控制**
  - ✓ 循环语句
  - ✓ 在循环的时候可以使用$loop变量获取循环信息，例如是否是循环的第一个或最后一个迭代。
  - ✓ $loop变量

属性	描述
`$loop->index`	当前循环迭代索引 (从0开始).
`$loop->iteration`	当前循环迭代 (从1开始).
`$loop->remaining`	当前循环剩余的迭代
`$loop->count`	迭代数组元素的总数量
`$loop->first`	是否是当前循环的第一个迭代
`$loop->last`	是否是当前循环的最后一个迭代
`$loop->depth`	当前循环的嵌套层级
`$loop->parent`	嵌套循环中的父级循环变量

Blade模板

- **包含子视图**

✓ Blade 的 @include 指令允许你很简单的在一个视图中包含另一个 Blade 视图，所有父级视图中变量在被包含的子视图中依然有效。

✓ 尽管被包含的视图继承所有父视图中的数据，你还可以传递额外参数到被包含的视图。

```
<div>
    @include('shared.errors')

    <form>
        <!-- Form Contents -->
    </form>
</div>
```

```
@include('view.name', ['some' => 'data'])
```

感谢聆听！

THANK YOU FOR YOUR ATTENTION