**Blade 概述**

与[视图文件](https://laravelacademy.org/post/9620.html)紧密关联的就是模板代码，我们在视图文件中通过模板代码和 HTML 代码结合实现视图的渲染。和很多其他后端语言不同，PHP 本身就可以当做模板语言来使用，但是这种方式有很多缺点，比如安全上的隐患、容易产生业务逻辑与视图模板的耦合，而且在视图文件中到处使用 <?php 内联代码一点都不优雅，甚至是 ugly code，所以你会看到绝大多数现代框架都会提供一套模板引擎，比如 [Smarty](https://www.smarty.net/)，[Twig](https://twig.symfony.com/)，以及 Laravel 使用的 [Blade](https://laravelacademy.org/post/9550.html)。

注：不同于其他基于 Symfony 的 PHP 框架，Laravel 没有使用 Twig 模板引擎，不过你想要使用的话，可以借助 [TwigBridge](https://github.com/rcrowe/TwigBridge) 扩展包来实现。

Blade 模板引擎是由 Laravel 框架提供的自有实现，借鉴了 .NET 的 [Razor](http://www.w3school.com.cn/aspnet/razor_intro.asp) 引擎语法，其语法简洁，易于上手，同时提供了强大而直观的继承模型，而且方便扩展。下面是一个简单的 Blade 模板代码示例：

<h1>{{ $group->title }}</h1>

{!! $group->imageHtml() !!}

@forelse ($users as $user)

{{ $user->username }} {{ $user->nickname }}<br>

@empty

该组中没有任何用户

@endforelse

正如你所看到的，Blade 模板引擎有三种常见的语法：

* 通过 {{ }} 渲染 PHP 变量（最常用）
* 通过 {!! !!} 渲染原生 HTML 代码（用于富文本数据渲染）
* 通过以 @ 作为前缀的 Blade 指令执行一些控制结构和继承、引入之类的操作

下面我们就来逐一介绍这些语法。

注：Blade 模板代码存放在以 .blade.php 后缀结尾的视图文件中，最终会被编译为原生 PHP 代码，并缓存起来，直到视图模板有修改才会再次编译，所以拥有与原生 PHP 几乎一致的性能，这些编译后的代码位于 storage/framework/views 目录下。你当然可以在 Blade 模板中使用原生 PHP 代码，但是不建议这么做，如果你非要这么做的话，可以通过 [@php](https://laravelacademy.org/post/8773.html#toc_13) 指令引入。

**渲染数据**

首先我们来看一下 {{}} 语法，我们通过该语法包裹需要渲染的 PHP 变量，如 {{ $variable }}，你可以将其类比为 <?php echo $variable; ?>，但是 Blade 模板代码的功能要更强大，通过 {{}} 语法包裹渲染的 PHP 变量会通过 htmlentities() 方法进行 HTML 字符转义，从而避免类似 XSS 这种攻击，提高了代码的安全性，所以 {{ $variable }} 编译后的最终代码是：

<?php echo htmlentities($variable); ?>

但是某些情况下不能对变量中 HTML 字符进行转义，比如我们在表单通过富文本编辑器编辑后提交的表单数据，这种场景就需要通过 {!! !!} 来包裹待渲染数据了：

{!! $variable !!}

这样编译后的代码就是 <?php echo $variable; ?> 了。

注：对于富文本数据 XSS 攻击防护，可以参考[这篇教程](https://laravelacademy.org/post/9476.html)。

最后，关于数据变量渲染，我们还要注意的是，很多前端框架也是通过 {{}} 来输出 JavaScript 变量数据的，比如 Laravel 的好基友 Vue.js 就是，对于这种情况，我们需要在渲染前端 JavaScript 变量的 {{}} 前面加上 @ 前缀，这样，Blade 模板引擎在编译模板代码的时候会跳过带 @ 前缀的 {{}} 数据渲染，并将 @ 移除从而可以后续执行对应的 JavaScript 框架渲染逻辑：

// Blade 引擎会将其编译为对应的 PHP 代码

{{ $phpData }}

// Blade 引擎编译时会移除 @，保留 {{ $vueData }} 结构

@{{ $vueData }}

如果要注释一段 PHP 代码，可以通过 {{-- 注释内容 --}} 实现。

**控制结构**

Blade 中的控制结构语法和 PHP 大同小异，学习成本几乎为零，不过 Blade 为我们额外提供了一些有用的辅助变量和方法，方便我们进行条件判断。

**条件语句**

**@if、@else、@elseif**

Blade 模板中的 @if 等价于 PHP 的 <?php if ($condition):，@else 和 @elseif 依次类推，最后以一个 @endif 收尾：

@if (count($students) === 1)

操场上只有一个同学

@elseif (count($students) === 0)

操场上一个同学也没有

@else

操场上有 {{ count($students) }} 个同学

@endif

和原生 PHP 中的用法如出一辙。

**@unless**

@unless 是 Blade 提供的一个 PHP 中没有的语法，用于表示和 @if 条件相反的条件，@unless($condition) 可以理解为 <?php if (!$condition):，然后以 @endunless 收尾：

@unless ($user->hasPaid())

用户没有支付

@endunless

**@isset、@empty**

这两个指令和 PHP 中的 isset() 和 empty() 方法等价：

@isset($records)

// 记录被设置

@endisset

@empty($records)

// 记录为空

@endempty

后面两个都是语法糖，如果你不想记太多东西，不妨都用 @if 来实现对应的逻辑了。

**@switch**

顾名思义，Blade 中的 @switch 指令和 PHP 中的 switch 语句等价，我们可以通过 @switch、@case、@break、@default 和 @endswitch 指令构建对应逻辑：

@switch($i)

@case(1)

// $i = 1 做什么

@break

@case(2)

// $i = 2 做什么

@break

@default

// 默认情况下做什么

@endswitch

**循环结构**

**@for、@foreach 和 @while**

和 PHP 一样，在 Laravel 中，我们可以通过与之等价的 @for、@foreach 和 @while 实现循环控制结构，使用语法和 PHP 代码相仿：

// for 循环

@for ($i = 0; $i < $talk->slotsCount(); $i++)

The number is {{ $i }}<br>

@endfor

// foreach 循环

@foreach ($talks as $talk)

{{ $talk->title }} ({{ $talk->length }} 分钟)<br>

@endforeach

// while 循环

@while ($item = array\_pop($items))

{{ $item->orSomething() }}<br>

@endwhile

**@forelse**

这个指令是 PHP 中具备的，可以理解为处理以下 PHP 代码逻辑：

<?php

if ($students) {

foreach ($students as $student) {

// do something ...

}

} else {

// do something else ...

}

在 Blade 模板中我们可以使用 @forelse 指令通过以下代码实现上述逻辑：

@forelse ($students as $student)

// do something ...

@empty

// do something else ...

@endforelse

**@foreach 和 @forelse 中的 $loop 变量**

在循环控制结构中，我们要重磅介绍的就是 Blade 模板为 @foreach 和 @forelse 循环结构提供的 $loop 变量了，通过该变量，我们可以在循环体中轻松访问该循环体的很多信息，而不用自己编写那些恼人的面条式代码，比如当前迭代索引、嵌套层级、元素总量、当前索引在循环中的位置等，$loop 实例上有以下属性可以直接访问：



下面是一个简单的使用示例：

<ul>

@foreach ($pages as $page)

@if ($loop->first)

// 第一个循环迭代

@endif

<li>{{ $loop->iteration }}: {{ $page->title }}

@if ($page->hasChildren())

<ul> @foreach ($page->children() as $child)

<li>{{ $loop->parent->iteration }}. {{ $loop->iteration }}: {{ $child->title }}</li>

@endforeach

</ul>

@endif

</li>

@if ($loop->last)

// 最后一个循环迭代

@endif

@endforeach

</ul>

有了这个 $loop 变量，确实能够帮我们节省很多重复的逻辑判断和编码工作，推荐使用。