#Giới thiệu

Blade là một templating engine đơn giản nhưng rất mạnh mẽ được cung cấp cùng với Laravel. Không giống các templating engines phổ biến khác của PHP, Blade không hạn chế bạn sử dụng code PHP thuần trong view. Thực tế,tất cả các views của Blade được compile thành mã PHP thuần và được cache lại cho tới khi chúng bị chỉnh sửa, nghĩa là Blade về cơ bản không làm tăng thêm chi phí ban đầu nào trong ứng dụng của bạn. Các file của Blade view sử dụng đuôi files là `.blade.php` và được lưu trong thư mục `resources/views`.

#Kế thừa template

Định nghĩa một layout

Hai lợi ích chính của việc sử dụng Blade là kế thừa template và sections. Để bắt đầu,hãy cùng xem một ví dụ đơn giản.Đầu tiên, chúng ta cùng xem một trang layout "master". Vì hầu hết các ứng dụng web đều duy trì một mẫu layout chung giữa nhiều trang với nhau, sẽ rất là thuận tiện để định nghĩa ra layout này như một Blade view riêng biệt:

```
<!-- Stored in resources/views/layouts/app.blade.php -->
<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>
```

Như bạn thấy, file này có chứa mã HTML cơ bản. Tuy nhiên, hãy chú ý ở hai directive `@section` và `@yield`. Về `@section` directive, đúng như tên gọi của nó, đinh nghĩa một nội dung, trong khi `@yield` lại được sử dụng để hiển thị nội dung ở một vị trí đặt trước.
Bây giờ chúng ta cần định nghĩa một layout cho chương trình của chúng ta , hãy cùng nhau tạo ra các trang con kế thừa từ layout này.

Mở rộng layout

Khi tạo một trang con, bạn có thể sử dụng `@extends` để cho biết là layout của trang con này sẽ thực hiện "kế thừa" từ đâu. View mà kế thừa một Blade layout có thể inject nội dung vào trong mục `@section`. Nhớ rằng, ở ví dụ trước, nội dung của những section này sẽ được hiển thị khi sử dụng `@yield`:

```
<!-- Stored in resources/views/child.blade.php -->

@extends('layouts.app')

@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
```

Ở ví dụ này, phần `sidebar` có sử dụng `@parent` để thực hiện thêm nội dung vào trong sidebar (thay vì ghi đè toàn bộ). `@parent` sẽ được thay thế bởi nội dung của layout khi view được render ra.

Blade view có thể được trả về từ route sử dụng helper view toàn cục:

```
Route::get('blade', function () {

    return view('child');

});
```

#Các Thành phần và khía cạnh

Components and slots cung cấp nhưng lợi ích tương tự như sections and layouts; Tuy nhiên, một số có lẽ tìm được mental model of components and slots dễ dàng hơn để hiểu .Đầu tiên, hãy tưởng tượng một thành phần "alert" có thể sử dụng lại được chúng ta có thể sử dụng lại xuyên suốt cả ứng dụng :

```
<!-- /resources/views/alert.blade.php -->

<div class="alert alert-danger">
    {{ $slot }}
</div>
```

Biến `{{ $slot }}`sẽ bao gồm nội dung mà có thể inject tới các thành phần. Bây giờ , to construct this component, chúng ta có thể sử dụng `@component`Blade directive:

```
@component('alert')
    <strong>Whoops!</strong> Something went wrong!
@endcomponent
```

Đôi lúc,nó rất hữu ích cho việc đinh nghĩa nhiều slots cho các components. Hãy chỉnh sửa alert component để cho phép inject một tiêu đề .Tên của slots được hiển thị bởi biến "echoing"phù hợp với tên

```
<!-- /resources/views/alert.blade.php -->


<div class="alert alert-danger">

    <div class="alert-title">{{ $title }}</div>


    {{ $slot }}

</div>
```

Bây giờ,chúng ta có thể inject nội dung vào tên slots sử `@slot` directive.Những thứ khác sẽ được thông qua thành phần trong biến `$slot` :

```
@component('alert')

    @slot('title')

        Forbidden

    @endslot


    You are not allowed to access this resource!

@endcomponent
```

Passing Additional Data To Components

Đôi khi bạn có thể cần pass thông tin thêm vào một ccomponent.Vì lý do trên, bạn có thể thông qua một mảng dữ liệu như là một tham số thứ hai @component .Tất cả dữ liệu sẽ được tạo thành biến cho template thành phần như là các biến

```
@component('alert', ['foo' => 'bar'])

    ...

@endcomponent
```

#Hiển thị dữ liệu

Bạn có thể hiển thị data truyền vào trong Blade views bằng cách đặt biến vào trong cặp dấu ngoặc nhọn. Ví dụ, với route dưới đây:

```
Route::get('greeting', function () {
    return view('welcome', ['name' => 'Samantha']);
});
```

Bạn có thể hiển thị nội dung của biến `name` variable như sau:

```
Hello, {{ $name }}.
```

Dĩ nhiên là bạn không hề bị giới hạn trong việc hiển thị nội dung của biến truyền vào trong view. Bạn cũng có thể hiển thị kết quả của bất cứ hàm PHP nào. Chính xác hơn, bạn có thể đặt bất cứ mã PHP nào bạn muốn vào trong một mệnh đề hiển thị của Blade:

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

Cặp `{{ }}` của Blade được tự động gửi tới hàm `htmlentities` của PHP để ngăn chặn các hành vi tấn công XSS
.

Mặc định, cặp `{{ }}` được tự động gửi qua hàm `htmlentities` của PHP để ngăn chặn tấn công XSS. Nếu bạn không muốn dự liệu bị escaped,bạn có thể sử dụng cú pháp:

```
Hello, {!! $name !!}.
```

Phải cẩn thận khi hiện nội dung được người dùng cung cấp. Luôn luôn sử dung cặp ngoặc nhọn để ngăn chặn tấn công XSS attacks khi hiển thị dữ liệu được cung cấp.

Blade & JavaScript Frameworks

Vì nhiều framework Javascript cũng sử dụng cặp dấu ngoặc nhọn để cho biết một biểu thức cần được hiển thị lên trình duyệt, bạn có thể sử dụng dấu @ để báo cho Blade biết là biểu thức này cần được giữ lại. Ví dụ:

```
<h1>Laravel</h1>

Hello, @{{ name }}.
```

Trong ví dụ này, biểu tượng `@` sẽ bị xóa bởi Blade ; tuy nhiên, `{{ name }}` được dữ lại cho phép nó được render tiếp bởi Javascript framkwork của bạn.

The `@verbatim` Directive

Nếu bạn muốn hiển thị biến JavaScript trong phần lớn template của bạn, bạn có thể bọc chúng trong directive khi đó bạn sẽ không cần tiền tố `@` trước biểu thức điều kiện:

```
@verbatim
    <div class="container">
        Hello, {{ name }}.
    </div>
```

#Control Structures

Ngoài template inheritance và hiển thị dữ liệu, Blade còn cung cấp một số short-cuts PHP control structures, như biểu thức điều kiện và vòng

lặp. Các short-cuts provide rất rõ ràng, là cách ngắn gọn khi làm việc với PHP control structures, và giống cấu trúc của PHP counterparts.

Cấu trúc điều kiện

Bạn có xây dựng cấu trúc `if` sbằng cách sử dụng `@if`, `@elseif`, `@else`, và `@endif` directives. Những directives tương ứng giống các từ khóa của PHP:

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

Một sự trùng hợp, Blade cũng có thể cung cấp an `@unless` directive:

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

Thêm vào đó ,directive điều kiện đã được thảo luận, the `@isset` and `@empty` directives sẽ được sử dụng như là như những shortcuts cho các hàm PHP lần lượt:

```
@isset($records)
    // $records is defined and is not null...
@endisset

@empty($records)
    // $records is "empty"...
@endempty
```

Vòng lặp

Ngoài cấu trúc điều kiện, Blade cũng cung cấp phương thức hỗ trợ cho việc xử lý vòng lặp. Một lần nữa, mỗi directives tương ứng giống các từ khóa PHP:

```
@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
```

Trong vòng lặp, bạn có thể sử dụng biến vòng lặp để lấy được thông tin giá trị của vòng lặp, chẳng hạn như bạn muốn lấy giá trị đầu tiên hoặc cuối cùng của vòng lặp.

Khi sử dụng vòng lặp bạn cũng có thể kết thúc hoặc bỏ qua vòng lặp hiện tại:

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

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

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

Bạn cũng có thể thêm điều kiện directive biểu diễn trong một dòng:

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

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

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

Biến vòng lặp

Trong vòng lặp, một biến `$loop` sẽ tồn tại bên trong vòng lặp. Biến này cho phép ta truy cập một số thông tin hữu ích của vòng lặp như index của vòng lặp hiện tại và vòng lặp đầu hoặc vòng lặp cuối của nó:

```
@foreach ($users as $user)
    @if ($loop->first)
        This is the first iteration.
    @endif

    @if ($loop->last)
        This is the last iteration.
    @endif

    <p>This is user {{ $user->id }}</p>
@endforeach
```

Nếu bạn có vòng lặp lồng nhau, bạn có thể truy cập biên `$loop` của vòng lặp tra qua thuộc tính `parent`:

```
@foreach ($users as $user)
    @foreach ($user->posts as $post)
        @if ($loop->parent->first)
```

```
            This is first iteration of the parent loop.
        @endif
    @endforeach
@endforeach
```

Biến `$loop` còn chứa một số thông tin hữu ích:

Thuộc tính	Miêu tả
`$loop->index`	Chỉ số index hiện tại của vòng lặp (starts at 0).
`$loop->iteration`	Các vòng lặp hiện tại (starts at 1).
`$loop->remaining`	Số vòng lặp còn lại.
`$loop->count`	Tổng số vòng lặp.
`$loop->first`	Vòng lặp đầu tiên.
`$loop->last`	Vòng lặp cuối cùng.
`$loop->depth`	Độ sâu của vòng lặp hiện tại.
`$loop->parent`	Biến parent loop của vòng lặp trong 1 vòng lặp lồng.

Comments

Blade còn cho phép bạn comment trong view. Tuy nhiên, không như comment của HTML, comment của Blade không đi kèm nội dung HTML được trả về:

```
{{-- This comment will not be present in the rendered HTML --}}
```

PHP

Trong một số trường hợp, thật hữu ích khi nhúng mã PHP vào trong views của bạn.Bạn có thể sử dung Blade `@php`directive để chạy một khối của mã thuần PHP template:

```
@php
```

```
//
@endphp
```

Trong khi Blade cung cấp những đặc điểm trên,sử dung nó thường xuyên cho thấy rằng bạn đã nhúng nhiều logic vào trong template của bạn.

Including Sub-Views

Blade's `@include` directive cho phép bạn chèn một Blade view từ một view khác. Tất cả các biến tồn tại trong view cha đều có thể sử dụng ở viree chèn thêm:

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

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

Mặc dù các view được chèn thêm kế thừa tất cả dữ liệu từ view cha, bạn cũng có thể truyền một mảng dữ liệu bổ sung:

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

Bạn nên tránh sử dụng `__DIR__` và `__FILE__` ở trong Blade views, vì chúng sẽ tham chiếu tới vị trí file bị cache.

Rendering Views cho Collections

Bạn có thể kết hợp vòng lặp và view chèn thêm trong một dòng với `@each` directive:

```
@each('view.name', $jobs, 'job')
```

Tham số thứ nhất là tên của view partial để render các element trong mảng hay collection. Tham số thứ hai là một mảng hoặc collection mà bạn muốn lặp, tham số thứ ba là tên của biến được gán vào trong vòng lặp bên view. Vì vậy, ví dụ, nếu bạn muốn lặp qua một mảng tên `jobs`, bạn phải truy xuất vào mỗi biến `job` trong view partial. key của vòng lặp hiện tại sẽ tồn tại như là `key` bên trong view partial.

Bạn cũng có thể truyền tham số thứ tư vào `@each` directive. tham số này sẽ chỉ định view sẽ dduwcoj render nếu như mảng bị rỗng.

```
@each('view.name', $jobs, 'job', 'view.empty')
```

Stacks

Blade cho phép bạn đẩy tên stack để cho việc render ở một vị trí nào trong view hoặc layout khac. Việc này rất hữu ích cho việc xác định thư viện JavaScript libraries cần cho view con:

```
@push('scripts')
    <script src="/example.js"></script>
@endpush
```

Bạn có thể đẩy một hoặc nhiều vào stack. Để render thành công một nội dung stack, truyền vào tên của stack trong `@stack` directive:

```
<head>
    <!-- Head Contents -->

    @stack('scripts')
</head>
```

Service Injection

Để `@inject` directive có thể được sử dụng để lấy lại một service từ Laravel service container. Tham số thứ nhất `@inject` là tên biến của service sẽ được đặt vào, tham số thứ hai là class hoặc tên interface của service bạn muốn resolve:

```
@inject('metrics', 'App\Services\MetricsService')

<div>
    Monthly Revenue: {{ $metrics->monthlyRevenue() }}.
</div>
```

#Mở rộng Blade

Blade còn cho phép bạn tùy biên directives bằng phương thức `directive`. Khi trình viên dich của Blade gặp directive, nó sẽ gọi tới callback được cung cấp với tham số tương ứng.

Ví dụ dưới đây tạo một `@datetime($var)` directive để thực hiện format một biết `$var`, nó sẽ là một thể hiện của `DateTime`:

```
<?php

namespace App\Providers;

use Illuminate\Support\Facades\Blade;
use Illuminate\Support\ServiceProvider;

class AppServiceProvider extends ServiceProvider
```

```
{
  /**
   * Perform post-registration booting of services.
   *
   * @return  void
   */
  public function boot()
  {
     Blade::directive('datetime', function($expression) {
       return "<?php echo $expression->format('m/d/Y H:i'); ?>";
            });
      }

      /**
   * Register bindings in the container.
   *
   * @return  void
   */
      public  function  register()
      {
          //
      }
}
```

Như bạn thấy, chúng ta sẽ móc lỗi phương thức `format` trong bất cứ biểu thức nào được gửi qua directive. Vì vậy, Trong ví dụ trên, Mã PHP được tạo ra bởi directivesẽ là:

```
<?php echo $var->format('m/d/Y H:i'); ?>
```

Sau khi cập nhật logiccủa mộtBlade directive, bạn cần xóa hết tất cả các Blade view bị cache. cache Blade views có thể xóa bằng lệnh `view:clear` Artisan.