BLADE TEMPLATE LARAVEL

Blade Template là gì?

- Blade là một Template Engine đơn giản nhưng mạnh mẽ được cung cấp bởi Laravel
- Blade không giới hạn bạn trong việc sử dụng mã PHP đơn giản trong View
- Tất cả các Blade View sẽ được biên dịch thành mã PHP và được lưu vào bộ đệm cho đến khi chúng được sửa đổi
- Các file Blade View có phần mở rộng là `.blade.php` và lưu trong thư mục `resources/views`.

Hiển thị dữ liệu

- Cú pháp 1: {{ $variable }}: Hiển thị dữ liệu dạng thực thể
- Cú pháp 2: {!! $variable !!}: Hiển thị dữ liệu có biên dịch mã HTML

Lưu ý: Cú pháp hiển thị này có thể áp dụng toán tử 3 ngôi

Vòng lặp

Vòng lặp for

```
@for($i = 1; $i <= 10; $i++)
    <p>Phần tử: {{$i}}</p>
@endfor
```

Vòng lặp while

```
@while ($condition)
<p>Xử lý trong này</p>
@endwhile
```

Vòng lặp foreach

```
@foreach($dataArr as $item)
    <p>Phần tử: {{$item}}</p>
@endforeach
```

Vòng lặp forelse

Vòng lặp này chính là phiên bản nâng cấp của foreach và có kiểm tra điều kiện nếu mảng trống

```
@forelse($dataArr as $item)
    <p>Phần tử: {{$item}}</p>
@empty
    <p>Không có phần tử</p>
@endforelse
```

Câu lệnh rẽ nhánh

Câu lệnh if

```
@if ($condition)
<p>Kết quả đúng</p>
@endif
```

Câu lệnh if else

```
@if ($condition)
<p>Kết quả đúng</p>
@else
<p>Kết quả sai</p>
@endif
```

Câu lệnh if..elseif

```
@if ($condition1)
<p>Kết quả 1</p>
@elseif ($condition2)
<p>Kết quả 2</p>
@elseif ($condition3)
<p>Kết quả 2</p>
@else
<p>Kết quả 4</p>
@endif
```

Câu lệnh switch...case

```
@switch($i)
    @case(1)
        First case...
        @break

    @case(2)
        Second case...
        @break

    @default
        Default case...
@endswitch
```

Viết mã php

```
@php
Câu lệnh PHP trong này
@endphp
```

Comment

```
{{--Nội dung comment--}}
```

Blade & Javascript Framework

Hiện nay có rất nhiều Javascript Framework sử dụng ký hiệu {{ }} để render dữ liệu. Nếu bạn không muốn Blade biên dịch dữ liệu, bạn chỉ cần ký tự @ phía trước

```
@{{$name}}
```

Nếu không muốn Blade biên dịch nhiều dòng có thể sử dụng `@verbatim`

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

Include View

```
@include('path_view')
```

Master Layout và kế thừa Master Layout

Tạo master layout

Tạo file layout `app.blade.php` tại đường dẫn sau: `resources/views/layouts`

```
<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>
```

@yield: dùng để chỉ định section có name trong yield sẽ được hiển thị.
@section: dùng để định nghĩa nội dung của một section.

Kế thừa Master Layout

Để kế thừa Master Layout, hãy dùng cú pháp sau:

```
@extends('path_layout')
```

Tạo file `home.blade.php` tại đường dẫn sau: `resources/views`

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

@section('title')
Học lập trình Laravel
@endsection

@section('sidebar')
    @parent

    <p>This is appended to the master sidebar.</p>
@endsection

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

Làm việc với Form

- @csrf: Tạo input hidden chứa csrf_token
- @method('PUT'): Tạo input hidden chứa HTTP Method
- @error('name'): Hiển thị error của form validate

Stack - Push - Prepend

File Layout: `resources/views/layouts/app.blade.php`

```
<html>
<head>
    <title>App Name - @yield('title')</title>
</head>
<body>

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

@stack('scripts')

</body>
</html>
```

File View: `resources/views/home.blade.php`

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

@push('scripts', '<script>console.log('script
01')</script>')

@push('scripts')
    <script>
        console.log('script 02');
```

```
    </script>
@endpush
```

- @push: Sẽ đẩy nội dung mới vào @stack (Thêm xuống dưới)
- @prepend: Sẽ đẩy nội dung mới vào @stack (Thêm lên trên)

Tự định nghĩa directive thông thường

```
use Illuminate\Support\Facades\Blade;

Blade::directive('datetime', function ($expression) {
    return "<?php echo ($expression)->format('d-m-y
h:i'); ?>";
});
```

- **datetime**: Tên directive muốn định nghĩa
- Gọi ra với cú pháp: @datetime(‘value’)

Tự định nghĩa directive dạng rẽ nhánh

```
use Illuminate\Support\Facades\Blade;

Blade::if('disk', function ($value) {
    return config('filesystems.default') === $value;
});
```

Gọi directive vừa định nghĩa

```
@disk('local')
    <!-- The application is using the local disk... -->
@elsedisk('s3')
    <!-- The application is using the s3 disk... -->
@else
    <!-- The application is using some other disk... -->
```

```
@enddisk
```

Cache View

- Mặc định, Laravel sẽ cache view vào trong folder: `storage/framework/views`
- Khi có Request mới, Laravel sẽ kiểm tra xem file cache có tồn tại hoặc hết hạn
    - Nếu hết hạn hoặc không tồn tại: Hệ thống sẽ tạo bản cache mới
    - Nếu tồn tại và chưa hết hạn: Hệ thống sẽ giữ bản cache cũ cho đến khi hết hạn hoặc bị xoá
- Laravel cung cấp cho chúng ta 2 Command Line để thuận tiện hơn cho lập trình viên trong quá trình phát triển ứng dụng:
    - **Cache View**: `php artisan view:cache`
    - **Clear Cache View**: `php artisan view:clear`

Component

Component trong Laravel về cơ bản khá giống những thành phần như: section, include, layouts,...Tuy nhiên cách gọi sẽ đơn giản hơn rất nhiều và thường được dùng để tái sử dụng với những thành phần được sử dụng nhiều

Từ phiên bản Laravel 7.x, component có cách gọi giống như thẻ HTML (Khác biệt với các phiên bản trước đó)

Tạo Component

Để tạo component, hãy chạy câu lệnh sau:

```
php artisan make:component Alert
```

Hệ thống sẽ tạo class trong thư mục: `app\View\Components` và view trong thư mục: `resources/views/components`

Đăng ký component

Để đăng ký component, bạn thêm đoạn code sau vào Service Provider

```
use Illuminate\Support\Facades\Blade;

public function boot()
{
    Blade::component('package-alert', Alert::class);
}
```

Trong đó:

- package-alert: Tên dùng để gọi component bên view
- Alert: Tên class component đã tạo ở trên

Gọi component

```
<x-alert/>
```

```
<x-user-profile/>
```

Trong đó:

- x: là cú pháp bắt buộc
- alert, user-profile: tên đã đăng ký ở Service Provider

Nếu class component nằm trong 1 thư mục con, hãy gọi theo cú pháp sau:

```
<x-inputs.button/>
```

Lúc này bạn hiểu rằng class component có cấu trúc như sau:

```
app\View\Components\Inputs\Button.php
```

Truyền dữ liệu vào component

```
<x-alert type="error" :message="$message"/>
```

Dữ liệu được truyền vào giống thuộc tính của HTML. Nếu muốn biên dịch biến, bạn cần thêm dấu : phía trước thuộc tính

Lúc này, class component cần khai báo tham số tại phương thức `__construct`

```
public function __construct($type, $message){
    $this->type = $type;
    $this->message = $message;
}
```

Bên view của component chỉ cần gọi tên biến

```
<div class="alert alert-{{ $type }}">
    {{ $message }}
</div>
```

Lưu ý:

- Bạn cần đặt thuộc tính bên class component là public
- Nếu tên thuộc tính trong class component ở dạng camelCase => Khi gọi component cần chuyển thuộc tính thành dạng dấu gạch ngang (-)

```
public function __construct($alertType)
{
    $this->alertType = $alertType;
}
```

```
<x-alert alert-type="danger" />
```