

Публикация на тему

# Шаблонизатор blade

*Использование элементов представления в шаблоне проектирования HMVC фреймворка  
Laravel*

## Автор

[Михалькевич Александр Викторович](#)

## Публикация

**Наименование** Шаблонизатор blade

**Автор** А.В.Михалькевич

**Специальность** Использование элементов представления в шаблоне проектирования HMVC фреймворка Laravel ,

## Анотация

**Anotation in English**

**Ключевые слова**

**Количество символов** 17299

## Содержание

[Введение](#)

1 [Основы использования шаблонизатора blade в Laravel](#)

2 [Проверка шаблона на существование](#)

3 [View Composer](#)

[Заключение](#)

[Список использованных источников](#)

[Приложения](#)

## Введение

### 1 Основы использования шаблонизатора blade в Laravel

По умолчанию, laravel работает с шаблонизатором blade. Шаблоны создаются в папке app/views и имеют расширение blade.php. Шаблоны подключаются в экшне через хелпер view(), входящим параметром в который передается имя шаблона без расширения blade.php.

Сперва создадим в папке view папку layouts для хранения базовых шаблонов. В папке layouts создадим файл defaults.blade.php.

#### Базовый шаблон defaults.blade.php. Листинг 1.1

```
@include('layouts.header')
```

**Project name**

--блок ссылок --

- [Home](#)

```
@yield('content')
```

© Company 2014

```
@include('layouts.footer')
```

В файлах layouts/header.blade.php и layouts/footer.blade.php находится обычный html-код для шапки и футера сайта. Эти 3 файла – это неизменная часть шаблона.

@yield('content') – вывод переменной content. Саму переменную определим в меняющейся части шаблона.

Меняющуюся часть шаблона вынесем в отдельный файл index.blade.php.

#### Меняющаяся часть шаблона. Листинг 1.2

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

```
@section('content')
```

# Добро пожаловать на сайт

Текст на страницу

```
@stop
```

Если необходимо создать часть кода в шаблоне, которая в последствии будет либо заменена, либо добавлена, то можно воспользоваться директивой @show

```
@show
```

### Объявление переменной с директивой @show в базовом шаблоне. Листинг 1.3

```
@section('styles')
```

Далее к данному стилю можно добавить другие стили в файлах подшаблона. В подшаблоне обращаемся к директиве @parent переменной styles.

### Объявление переменной с директивой @show в базовом шаблоне. Листинг 1.4

```
@extends('public')
@section('styles')
    @parent
@stop
```

Обратите внимание на @extends в начале кода. В шаблонизаторе blade из контроллера мы обращаемся к подшаблону, а подшаблон с помощью директивы @extends сам себя вставляет в базовый шаблон public.

Подключение подшаблона index.blade.php осуществляется в экшне контроллера.

### Подключение подшаблона index.blade.php. Листинг 1.5

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

Передача массива в шаблон:

### Передача массива в шаблон. Листинг 1.6

```
$posts = array(1=>'One', 2=>'Two');
return view('index', $posts);
```

Имеется также ещё один способ передачи переменных в шаблон, через метод with():

### Передача переменной в шаблон с помощью метода with. Листинг 1.7

```
$posts = array(1=>'One', 2=>'Two');

return view('index')->with('posts', $posts)
```

Если вы не уверены в существовании передаваемой переменной, то нужно использовать with(), тогда не будет выводиться ошибка.

И ещё один способ передачи переменных:

### Магический метод передачи данных в шаблон. Листинг 1.8

```
$view = view('greeting')->withName('Victoria');
```

Для вывода переменных в шаблоне можно воспользоваться директивой {{{}}

### Вывод переменных шаблона. Листинг 1.9

```
{{ $name }} // простой вывод переменной

{{ isset($name)? $name: 'Default' }} // вывод либо переменной либо значения по умолчанию.

{{ $name or 'Default' }} // еще один способ вывода значения по умолчанию
```

Для предотвращения XSS-атак директива {{{}}} экранирует html-тэги. Если всё же необходимо вывести html, то можно воспользоваться другой директивой:

### Директива {!! !!}. Листинг 1.10

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

Вывод всех элементов массива на экран

#### **Вывод элементов массива на экран. Листинг 1.11**

```
@if($posts->count())  
@foreach($posts as $post)  
  
{{ $post }}  
  
@endforeach  
@endif
```

## **2 Проверка шаблона на существование**

Проверить существует ли шаблон по заданному пути, можно с помощью хелпера view() и метода exists(), входящим параметром в который передается путь к шаблону.

#### **Проверка шаблона на существование. Листинг 2.1**

```
if (view()->exists('templates.base')) {  
  
    // шаблон существует  
  
}else{  
  
    // шаблон не существует  
  
}
```

## **3 View Composer**

Композеры (view composers) - функции-замыкания или методы класса, которые вызываются, когда шаблон рендерится в строку. Если имеются данные, которые необходимо привязать к шаблону при каждом его рендеринге, то композеры помогут вам выделить такую логику в отдельное место.

Регистрировать композеры можно внутри сервис-провайдера. Мы будем использовать фасад View для того, чтобы получить доступ к имплементации контракта Illuminate\Contracts\View\Factory:

#### **Определение композера. Листинг 3.1**

```

<?php namespace App\Providers;

use View;

use Illuminate\Support\ServiceProvider;

use App\Http\ViewComposers\SiteComposer;

class ComposerServiceProvider extends ServiceProvider {

    public function boot()

    {

        View::composer('*', 'App\Http\ViewComposers\SiteComposer');

    }

    public function register()

    {

        //

    }

}

```

Файл композера необходимо зарегистрировать в config/app.php

#### **Регистрация файла композера. Листинг 3.2**

```
'App\Providers\ComposerServiceProvider',
```

Определение класса SiteComposer:

#### **Класс SiteComposer. Листинг 3.3**

```

<?php

namespace App\Http\ViewComposers;

use Illuminate\Contracts\View\View;

class SiteComposer

{

    public function compose(View $view)

    {

        $view->with('menu_items','TEST');

    }

}

```

Метод `compose` должен получать в качестве аргумента инстанс `Illuminate\Contracts\View\View`. Для передачи переменных в шаблон используйте метод `with()`.

Назначение композера для нескольких шаблонов

Вместо имени шаблона можно использовать массив имен.

#### **Назначение композера для нескольких шаблонов. Листинг 3.4**

```
View::composer(['profile', 'dashboard'],  
'App\Http\ViewComposers\MyViewComposer');
```

Композер для всех шаблонов

А вот так можно назначить композер для всех шаблонов:

#### **Назначение композера для всех шаблонов. Листинг 3.5**

```
View::composer('*', function()  
{  
    //  
});
```

Регистрация нескольких шаблонов

Можно использовать метод `composers`, чтобы зарегистрировать несколько композеров одновременно:

#### **Регистрация нескольких шаблонов одновременно. Листинг 3.5**

```
View::composers([  
    'App\Http\ViewComposers\AdminComposer' =>  
    ['admin.index', 'admin.profile'],  
    'App\Http\ViewComposers\UserComposer' => 'user',  
    'App\Http\ViewComposers\ProductComposer' => 'product'  
]);
```

## **Заключение**

## **Список использованных источников**

## **Приложения**