**Livewire 简介**

现如今的 JavaScript 有些疯狂，各种框架和工具层出不穷，有种「山中方一日，世上一千年」的感觉，有的时候，即使是很简单的功能，比如弹出模态框或者对前端交互做一些微调，我们也不得不引入大量的代码（比如安装一个 NPM 包），这无疑提高了应用的复杂性，也让项目代码库显得臃肿不堪，而且有的时候真的有点杀鸡焉用牛刀。

其实我们大可不必如此。

因为有了 [Livewire](https://livewire-framework.com/)，Livewire 是一个适用于 Laravel 的全栈框架，我们可以通过它来构建动态的前端功能，就像写原生 PHP 代码一样：



正如 Livewire 的 Slogan 一样，它就像 Vue 和 Blade 的孩子，有了它之后，可以抛开 Vue/React 之类的 JavaScript 框架，只需要在 PHP 服务端和 Blade 视图模板文件中，通过 PHP 代码即可完成前端视图组件的开发，和我们之前通过 Vue 实现的视图组件功能一样。

注：Livewire 目前还处于早期阶段，这一点通过 [Github 仓库](https://github.com/calebporzio/livewire)的代码提交时间也能看出来，建议先在本地进行测试体验。

**快速上手**

**准备工作**

下面我们来体验下如何通过 Livewire 开发前端视图组件。

首先通过 Composer 在项目根目下安装这个扩展包：

composer require calebporzio/livewire -vvv

然后在使用 Livewire 的页面中通过 @livewireAssets 指令引入对应的 JavaScript 代码，通常我们会在公共的布局文件中引入，比如 layouts.app：

...

@livewireAssets

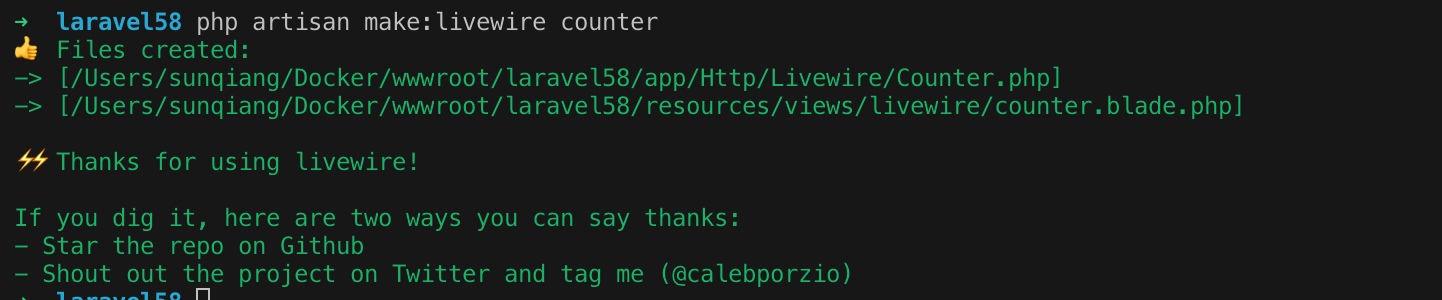
</body>

</html>

**创建组件**

接下来我们通过如下 Artisan 命令来创建一个新的视图组件：

php artisan make:livewire counter



初次执行 make:livewire 命令会在 Laravel 项目的 app/Http 目录下创建一个 Livewire 子目录，用于存放组件逻辑服务端实现代码类，然后在该目录下创建 counter 对应组件类 Counter：

<?php

namespace App\Http\Livewire;

use Livewire\Component;

class Counter extends Component

{

public function render()

{

return view('livewire.counter');

}

}

默认只有一个方法 render 用于指定对应的组件路径。

此外该命令还会在 resources/views 目录下创建一个 livewire 子目录用于存放 Livewire 组件（就像之前编写 Vue 组件要在 resources/js/components 目录下创建组件文件那样），显然本次生成的是 counter.blade.php，默认什么也不做：

<div>

{{-- The Master doesn't talk, he acts. --}}

</div>

我们可以将其改为一段欢迎文本：

<div>

<h1>Hello Livewire!<h1>

</div>

**引入组件**

然后创建一个测试视图 resources/views/counter.blade.php 将上面这个 counter 组件引入：

@extends("layouts.app")

@section('content')

@livewire('counter')

@endsection

接着在 routes/web.php 路由文件中注册一个测试路由：

Route::get('/counter', function () {

return view('counter');

});

在浏览器中访问这个路由，就可以看到 Livewire 组件渲染的结果了：



是不是非常简单！整个过程我们没有编写任何 JavaScript 及 Vue 组件代码，一切都是通过 PHP 代码和 Blade 模板代码实现的。

**与服务端的交互**

上述 Livewire 组件只是一段静态的 HTML 代码，而我们的组件实际上要实现的是计数器功能，这意味要要实现动态的效果，之前编写 Vue 组件的时候，相应的功能需要在 JavaScript 脚本中完成，在 Livewrie 框架中，我们把对应的业务逻辑搬到了 app/Http/Livewire 下的服务端组件类中，在 Counter 组件中，我们新增两个方法用于对计数器进行加减操作：

<?php

namespace App\Http\Livewire;

use Livewire\Component;

class Counter extends Component

{

public $count = 0;

public function increment()

{

$this->count++;

}

public function decrement()

{

$this->count--;

}

public function render()

{

return view('livewire.counter');

}

}

然后在客户端视图组件 resources/views/livewire/counter.blade.php 中，实现计数器功能代码如下：

<div style="text-align: center">

<button wire:click="increment">+</button>

<h1>{{ $count }}</h1>

<button wire:click="decrement">-</button>

</div>

可以看到，我们可以在客户端组件中直接引用服务端组件类中的公共属性以及方法，按钮点击事件通过 wire:click 来指定，对应的方法实现定义在服务端代码中，然后通过 Blade 语法引用 $count 变量渲染计数器的值，我们可以通过点击指定按钮对其进行加减操作。

无需编译前端资源，刷新之前的 /couter 路由页面，即可看到计数器组件的渲染效果：



注：Livewire 组件动态交互功能默认不兼容基于 Swoole 的 HTTP 服务器，请使用基于 PHP-FPM 的 HTTP 服务器进行测试。

**实现原理**

Livewire 组件的实现原理和 Vue/React 并无本质区别，如果换作是 Vue 组件实现上述计数器组件的话，对应组件模板代码如下：

<template>

<div style="text-align: center">

<button @click="increment">+</button>

<h1>{{ count }}</h1>

<button @click="decrement">-</button>

</div>

</template>

<script>

export default {

data: {

count: 0

},

methods: {

increment() {

this.count++

},

decrement() {

this.count--

},

},

}

</script>

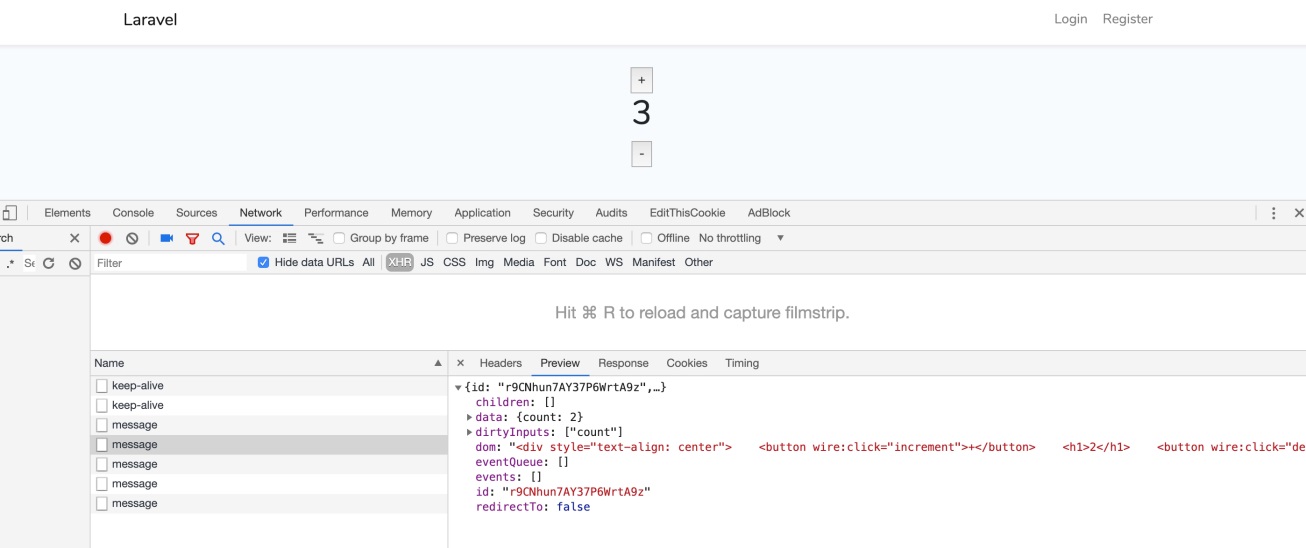
以点击「+」按钮为例，Vue 的底层实现逻辑如下：

* Vue 会监听 @click="increment" 点击事件；
* 然后调用脚本代码 methods 列表中定义的 increment 方法来更新 count 值；
* 最后重新渲染模板代码并更新 DOM。

Livewire 的实现逻辑也是一样的：

* 通过 wire:click="increment" 监听点击事件；
* 将调用 JavaScript 脚本代码改为发送 Ajax 请求到 PHP 服务器，通过 PHP 代码调用服务端 Livewire 组件类的 increment 方法更新 count 的值；
* PHP 服务器重新渲染 Blade 模板并将其返回给客户端，Livewire 客户端收到响应后更新 DOM。

这一点，可以通过在开发者工具控制台中监听网络请求看到：



所以与 Vue/React 相比，Livewire 的区别就是将原本本地 JavaScript 实现的逻辑通过 HTTP 请求发送到服务端处理，然后将结果返回给客户端，再由客户端更新 DOM。也正是因为这个原因，Livewire 在实现一些本不必服务端处理的组件时性能不如 Vue/React，因为多了一层网络请求，但如果原本 JavaScript 脚本也要通过 Ajax 与服务端进行交互，这个时候使用 Livewire 和 Vue/React 在性能上就不相上下了。不过对于性能没有过高要求以及前端小白来说，Livewire 倒是个不错的选择。

https://laravelacademy.org/post/19857.html