我们在[上一篇教程](https://laravelacademy.org/post/9676.html)中已经演示了如何通过 Request 请求实例获取各种文本输入数据，但是还有一种输入数据我们没有涉及到，那就是文件上传。我们可以通过 Request 请求实例提供的 file 方法获取用户上传文件，并将其保存到指定目录从而完成文件上传，接下来，我们将从前端到后端实现一个完整的用户上传文件功能，包括视图、路由、控制器部分代码。

**定义文件上传路由**

首先我们在 routes/web.php 中定义上传文件涉及到的路由：

// 用于显式上传表单

Route::get('form', 'RequestController@formPage');

// 用于处理文件上传

Route::post('form/file\_upload', 'RequestController@fileUpload');

我们定义了一个 GET 路由 /form，用于渲染用户上传表单页面，然后定义了一个 POST 路由 /form/file\_upload，用于实现文件上传逻辑。

接下来去控制器中初始化这两个方法。

**初始化控制器方法**

打开 app/Http/Controllers/RequestController.php，初始化路由定义中指定的控制器方法，首先是 formPage 方法用于渲染表单视图，我们约定视图路径是 request.form（可以先创建一个空的视图文件 resources/views/request/form.blade.php）：

public function formPage()

{

return view('request.form');

}

然后是 fileUpload 方法，用于处理 POST 请求实现文件上传，假定前端文件输入框对应 name 属性是 picture，如果请求内容中包含该字段，则将对应文件实例打印出来（文件上传保存实现代码后面再完善）：

public function fileUpload(Request $request)

{

if ($request->hasFile('picture')) {

dd($request->file('picture'));

}

}

**编写前端表单视图**

接下来，就可以到前端编写视图文件了，我们将通过单独的 Vue 组件实现前端文件异步上传操作，所以编写视图文件 resources/views/request/form.blade.php 代码如下：

<!doctype html>

<html lang="{{ app()->getLocale() }}">

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<meta name="csrf-token" content="{{ csrf\_token() }}">

<title>表单请求</title>

<link href="{{ asset('css/app.css') }}" rel="stylesheet">

</head>

<body>

<div id="app">

<div class="container">

<form>

<fileupload-component></fileupload-component>

<button type="submit" class="btn btn-primary">提交</button>

</form>

</div>

</div>

<script src="{{ asset('js/app.js') }}"></script>

</body>

</html>

我们会在表单控件中使用 Bootstrap 样式，所以引入了 css/app.css 文件，同时为 HTML 元素设置相应的 class 属性，将文件上传控件拆分成一个独立的 Vue 组件，并通过 <fileupload-component></fileupload-component> 引入。此外，需要注意的是我们在页面顶部添加了如下这行代码：

<meta name="csrf-token" content="{{ csrf\_token() }}">

这是为了后续通过 [axios](https://github.com/axios/axios) 发送 POST 请求的时候（axios 是一个功能强大的基于 Promise 的 JavaScript HTTP 客户端，推荐使用它来替代传统的 ajax 或 XMLHttpRequest API 发送 HTTP 请求），方便在请求头中全局设置 CSRF Token，在 axios 请求头添加 CSRF Token 的逻辑位于 resources/js/bootstrap.js 文件：

let token = document.head.querySelector('meta[name="csrf-token"]');

if (token) {

window.axios.defaults.headers.common['X-CSRF-TOKEN'] = token.content;

} else {

console.error('CSRF token not found: https://laravel.com/docs/csrf#csrf-x-csrf-token');

}

意思是从当前页面 meta 元标签中获取 [name="csrf-token"] 的值并将其设置到 axios 的请求头字段 X-CSRF-TOKEN 中，每次发送 POST 请求时会自动带上它，这样经过 CSRF 保护中间件时校验该请求头字段通过则认为是安全请求放行（具体原理不了解的话可以看 [CSRF 防护教程](https://laravelacademy.org/post/9616.html)）。

**编写文件上传 Vue 组件**

完成视图模板文件的编写之后，接下来就可以编写文件上传 Vue 组件了。在 resources/js/components 目录下新增 FileUploadComponent.vue，编写代码如下：

<style scoped>

div.form-group {

margin-top: 10px;

}

</style>

<template>

<div class="form-group">

<label for="picture">上传一张图片</label>

<input type="file" class="form-control-file" id="picture" ref="picture" v-on:change="uploadFile"/>

</div>

</template>

<script>

export default {

methods: {

uploadFile() {

let formData = new FormData();

formData.append('picture', this.$refs.picture.files[0]);

axios.post(

'/form/file\_upload',

formData,

{

headers: {

'Content-Type': 'multipart/form-data'

}

}

).then(function (response) {

console.log(response);

}).catch(function (error) {

console.log(error);

});

}

}

}

</script>

这算是一个比较完整的 Vue 组件代码了，既有 HTML 模板代码，又有 CSS 和 JavaScript 代码，代码逻辑很简单，就是监听到文件上传控件有变动时调用 uploadFile 方法，通过 axios 发送包含文件信息的 POST 请求到 /form/file\_upload 路由，由于我们发送的是上传文件请求，所以必须将内容类型设置为 multipart/form-data，如果后端处理成功则打印响应信息，否则打印失败信息。

然后在 resources/js/app.js 文件中将这个组件全局注册到 Vue 实例：

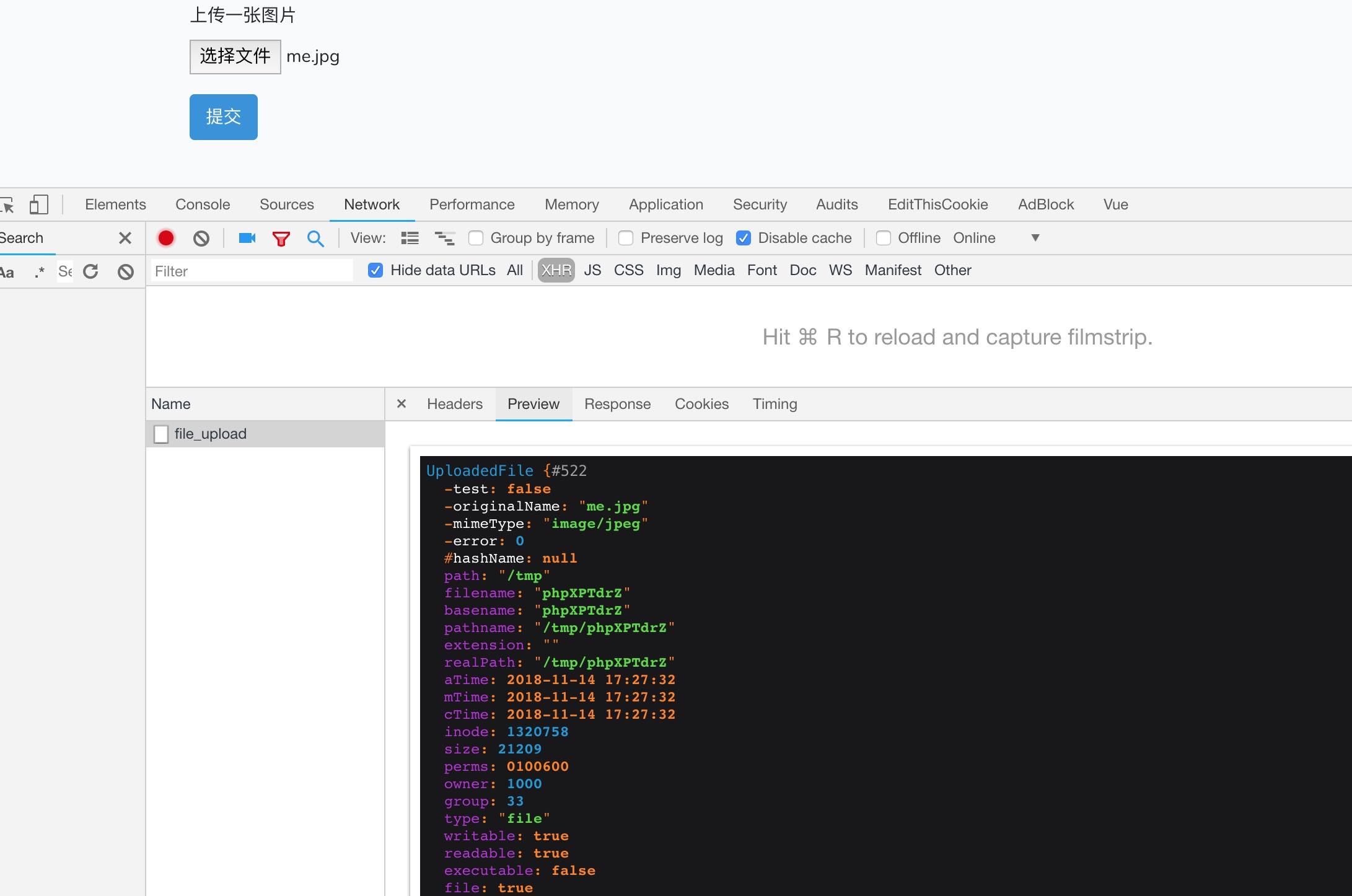
Vue.component('fileupload-component', require('./components/FileUploadComponent.vue'));

注：如果是在 Laravel 5.8 中，需要这样注册：Vue.component('fileupload-component', require('./components/FileUploadComponent.vue').default);，否则在使用的时候会报错。

这样在 form.blade.php 视图中就可以正常引入该组件了。

**测试文件上传功能**

至此，我们完成了前端视图和 Vue 组件的编写，运行 npm run dev 重新编译前端资源，访问 http://blog.test/form 就可以测试文件上传了，先打开 F12 监听上传请求，然后上传一张图片，上传成功后，就可以看到后端打印的文件信息了：



**完善后端文件上传代码**

通过打印信息可以看出，$request->file() 方法获取的是一个 Illuminate\Http\UploadedFile 对象实例，该类继承自 PHP SPL 库中提供与文件交互方法的 SplFileInfo 类，我们通过该类提供的方法就可以完成上传文件保存操作：

public function fileUpload(Request $request)

{

if ($request->hasFile('picture')) {

$picture = $request->file('picture');

if (!$picture->isValid()) {

abort(400, '无效的上传文件');

}

// 文件扩展名

$extension = $picture->getClientOriginalExtension();

// 文件名

$fileName = $picture->getClientOriginalName();

// 生成新的统一格式的文件名

$newFileName = md5($fileName . time() . mt\_rand(1, 10000)) . '.' . $extension;

// 图片保存路径

$savePath = 'images/' . $newFileName;

// Web 访问路径

$webPath = '/storage/' . $savePath;

// 将文件保存到本地 storage/app/public/images 目录下，先判断同名文件是否已经存在，如果存在直接返回

if (Storage::disk('public')->has($savePath)) {

return response()->json(['path' => $webPath]);

}

// 否则执行保存操作，保存成功将访问路径返回给调用方

if ($picture->storePubliclyAs('images', $newFileName, ['disk' => 'public'])) {

return response()->json(['path' => $webPath]);

}

abort(500, '文件上传失败');

} else {

abort(400, '请选择要上传的文件');

}

}

关于 UploadedFile 提供的更多方法，可以去源码中查看。我们使用了 Storage::disk('public') 磁盘将上传文件保存到本地，关于该磁盘的自定义配置信息可以去 config/filesystems.php 文件中查看，我们将其保存到此磁盘的原因是图片一般都是提供对外访问的，如果上传的是其他格式的私密文件，不想被外部访问，需要将其保存到 local 磁盘，你还可以将其保存到云存储服务中，关于完整的文件系统我们放到后面去讲，现在你只需了解保存到哪里，以及如何配置自定义路径。

如果要让上传到 storage/app/public 目录的文件可以被外部访问，还要执行以下命令：

php artisan storage:link

该命令会在项目根目录下的 public 中创建一个软链 storage 指向 storage/app/public，这样，后者下面的文件才可以真正通过 Web 路径访问。

**优化前端图片上传组件代码**

接下来，回到 resources/js/components/FileUploadComponent.vue 组件，对前端文件上传代码进行调整和优化。文件上传成功后，将返回路径更新到一个隐藏的字段，以便后续跟随表单上传，并且提供图片预览功能，以便拥有更好的用户体验：

<template>

<div class="form-group">

<label for="picture">上传一张图片</label>

<input type="file" class="form-control-file" id="picture" ref="picture" v-on:change="uploadFile"/>

<input type="hidden" id="picture-path" value="">

<div id="picture-preview">

</div>

</div>

</template>

<script>

export default {

methods: {

uploadFile() {

let formData = new FormData();

formData.append('picture', this.$refs.picture.files[0]);

axios.post(

'/form/file\_upload',

formData,

{

headers: {

'Content-Type': 'multipart/form-data'

}

}

).then(function (response) {

$('#picture-path').val(response.data.path);

$('#picture-preview').html('<img src="' + response.data.path + '">')

}).catch(function (error) {

console.log(error);

});

}

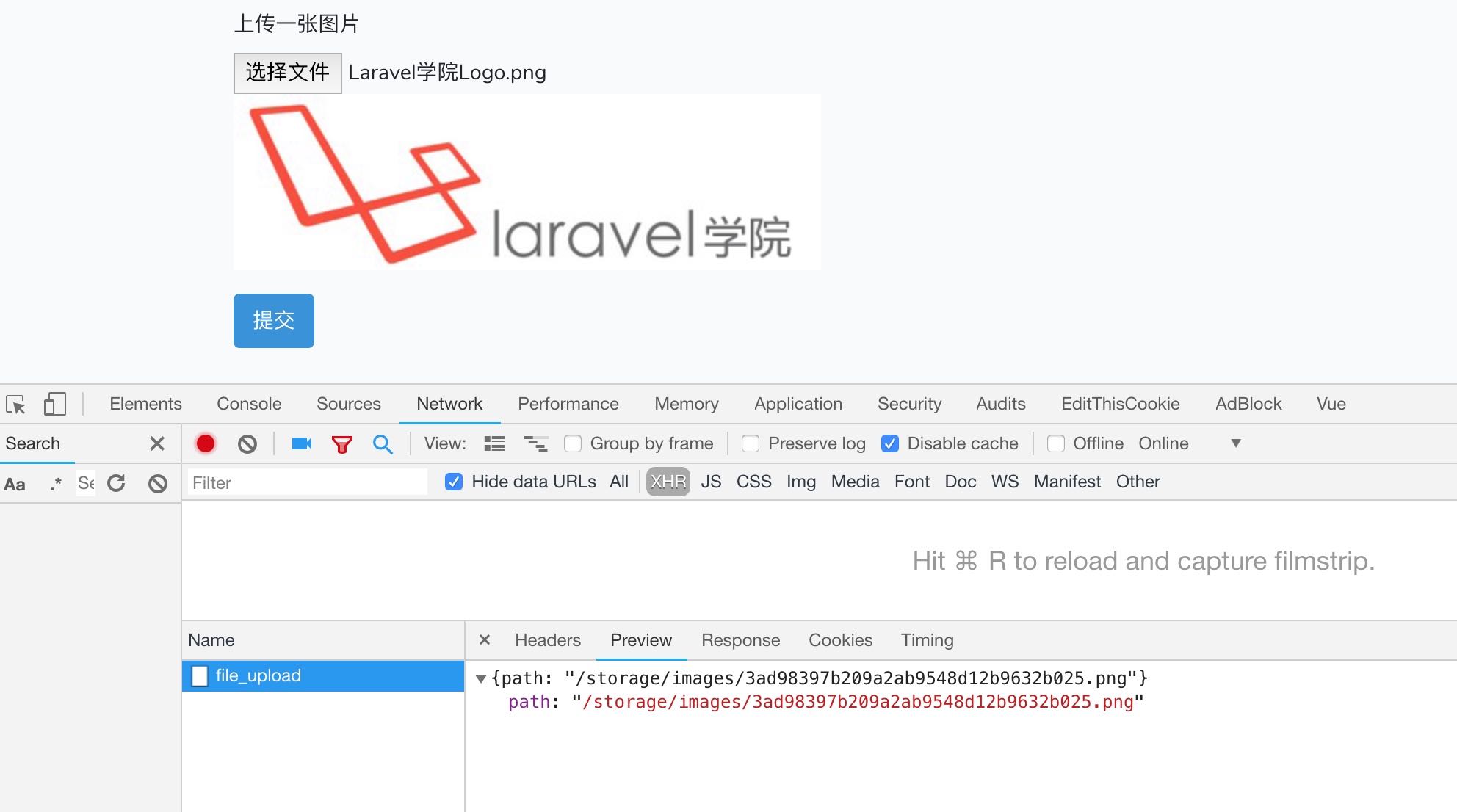
}

}

</script>

**再次测试文件上传功能**

至此，基于 Laravel + Vue 组件的文件异步上传功能就全部完成了。运行 npm run dev 重新编译前端资源，再次访问表单页面，重新上传一张新的图片：



上传成功后，就能通过图片 Web 路径预览刚刚上传的图片了。

如果你想对上传图片进行裁剪、压缩、生成缩略图等操作，可以试试 [Intervention Image](https://laravelacademy.org/post/3585.html) 这个扩展包，功能非常强大，基本能够满足你的所有图片处理需求。