**简介**

[Laravel Mix](https://github.com/JeffreyWay/laravel-mix) 提供了一套流式 API，使用一些通用的 CSS 和 JavaScript 预处理器为 Laravel 应用定义 Webpack 构建步骤。通过简单的方法链，你可以流式定义资源管道。例如：

mix.js('resources/assets/js/app.js', 'public/js')

.sass('resources/assets/sass/app.scss', 'public/css');

如果你对如何开始使用 Webpack 和前端资源编译感到困扰，那么你会爱上 Laravel Mix。不过，并不是强制要求在开发期间使用它。你可以自由选择使用任何前端资源管道工具，或者压根不使用。

**安装 & 设置**

**安装 Node**

在开始接触 Mix 之前，必须首先确保 Node.js 和 NPM 在机器上已经安装：

node -v

npm -v

默认情况下，Laravel Homestead 已经包含了你所需要的一切；不过，如果你没有使用 Homestead，你也可以从[Node 的下载页面](https://nodejs.org/en/download/)轻松的下载安装最新版本的 Node 和 NPM。

**Laravel Mix**

接下来，需要安装 Laravel Mix，在新安装的 Laravel 根目录下，你会发现有一个 package.json 文件。该文件包含你所需要的一切，和 composer.json 类似，只不过是用来定义 Node 依赖而非 PHP 依赖，你可以通过运行如下命令来安装需要的依赖：

npm install

如果你正在 Windows 系统上开发，需要在运行 npm install 命令时带上 --no-bin-links：

npm install --no-bin-links

**运行 Mix**

Mix 是位于 [Webpack](https://webpack.js.org/) 顶层的配置层，所以要运行 Mix 任务你只需要在运行包含在默认 package.json 文件中的其中某个 NPM 脚本即可：

// 运行所有 Mix 任务...

npm run dev

// 运行所有 Mix 任务并减少输出...

npm run production

**监控前端资源改变**

npm run watch 命令将会持续在终端运行并监听所有相关文件的修改，Webpack将会在发现修改后自动重新编译资源文件：

npm run watch

你可能会发现文件变更的时候特定环境的 Webpack 不会更新，如果你遇到了这样的问题，可以考虑使用 watch-poll 命令：

npm run watch-poll

**处理样式表**

webpack.mix.js 是所有资源编译的入口，可以将其看作 Webpack 的轻量级配置封装层。Mix 任务可以以方法链的方式被链在一起来定义前端资源如何被编译。

**Less**

要将 [Less](http://lesscss.org/) 编译成 CSS，可以使用 less 方法。下面让我们来编译 app.less 文件到 public/css/app.css：

mix.less('resources/assets/less/app.less', 'public/css');

多次调用 less 方法可用于编译多个文件：

mix.less('resources/assets/less/app.less', 'public/css')

.less('resources/assets/less/admin.less', 'public/css');

如果你想要自定义编译后文件的输出位置，可以将完整的路径信息作为第二个参数传递到 less 方法：

mix.less('resources/assets/less/app.less', 'public/stylesheets/styles.css');

如果你需要覆盖[底层 Less 插件选项](https://github.com/webpack-contrib/less-loader#options)，可以传递一个对象作为 mix.less() 的第三个参数：

mix.less('resources/assets/less/app.less', 'public/css', {

strictMath: true

});

**Sass**

sass 方法允许你将 [Sass](http://sass-lang.com/) 编译成 CSS。你可以像这样使用该方法：

mix.sass('resources/assets/sass/app.scss', 'public/css');

同样，和 less 方法一样，你可以将多个 Sass 文件编译成单个 CSS 文件，甚至自定义结果 CSS 的输出路径：

mix.sass('resources/assets/sass/app.sass', 'public/css')

.sass('resources/assets/sass/admin.sass', 'public/css/admin');

额外的 [Node-Sass 插件选项](https://github.com/sass/node-sass#options)可以以第三个参数的形式提供：

mix.sass('resources/assets/sass/app.sass', 'public/css', {

precision: 5

});

**Stylus**

和 Less 和 Sass 类似，stylus 方法允许你将 [Stylus](http://stylus-lang.com/) 编译成 CSS：

mix.stylus('resources/assets/stylus/app.styl', 'public/css');

你还可以安装额外的 Stylus 插件，例如 [Rupture](https://github.com/jescalan/rupture)，首先，通过 NPM 安装这个插件（npm install rupture）然后在调用 mix.stylus() 时引入它：

mix.stylus('resources/assets/stylus/app.styl', 'public/css', {

use: [

require('rupture')()

]

});

**PostCSS**

[PostCSS](http://postcss.org/)，是一个转化 CSS 的强大工具，在 Laravel Mix 中开箱可用。默认情况下，Mix 使用了流行的 [Autoprefixer](https://github.com/postcss/autoprefixer) 插件来自动添加所需要的 CSS3 浏览器引擎前缀。不过，你也可以添加与应用适配的其他额外插件。首先，通过 NPM 安装需要的插件，然后在 webpack.mix.js 文件中引用：

mix.sass('resources/assets/sass/app.scss', 'public/css')

.options({

postCss: [

require('postcss-css-variables')()

]

});

**原生 CSS**

如果你只想要将多个原生 CSS 样式文件合并到一个文件，可以使用 styles 方法：

mix.styles([

'public/css/vendor/normalize.css',

'public/css/vendor/videojs.css'

], 'public/css/all.css');

**URL 处理**

因为 Laravel Mix 是基于 Webpack 开发的，所以了解一点关于 Webpack 的概念很重要。对 CSS 编译而言，Webpack 会在样式表中重写并优化所有 url() 调用，虽然这可能最初听上去很奇怪，但这确实个不折不扣的强大功能。假设我们想要编译包含图片相对 URL 的 Sass：

.example {

background: url('../images/example.png');

}

注：任意给定 url() 的绝对路径都会从 URL 重写中排除，例如，url('/images/thing.png') 或 url('http://example.com/images/thing.png') 将不会被修改。

默认情况下，Laravel Mix 和 Webpack 会找到 example.png，将其拷贝到 public/images 目录下，然后在生成的样式表中重写 url()，因此，编译后的 CSS 如下所示：

.example {

background: url(/images/example.png?d41d8cd98f00b204e9800998ecf8427e);

}

和这个功能一样有用的是，可能已存在的目录结构已经配置成你想要的方式，这种情况下，你可以禁用 url() 重写：

mix.sass('resources/assets/app/app.scss', 'public/css')

.options({

processCssUrls: false

});

如果添加了这项配置到 webpack.mix.js 文件，Mix 将不再匹配 url() 或拷贝资源到 public 目录。换句话说，编译过的 CSS 和编译前输入的一样：

.example {

background: url("../images/thing.png");

}

**Source Map**

虽然 Source Map 默认被禁用，但是可以通过在 webpack.mix.js 文件中调用 mix.sourceMaps() 来激活。尽管这会带来编译/性能开销，不过在编译资源的时候可以提供额外的调试信息给浏览器的开发者工具：

mix.js('resources/assets/js/app.js', 'public/js')

.sourceMaps();

**处理 JavaScript**

Mix 还提供了多个特性帮助你处理 JavaScript 文件，例如编译 ECMAScript 2015，模块捆绑，最小化以及合并原生 JavaScript 文件。更妙的是，这些都是无缝集成的，不需要额外的自定义配置：

mix.js('resources/assets/js/app.js', 'public/js');

通过这一行代码，你可以使用如下功能：

* ES2015 语法
* 模块
* 编译 .vue 文件
* 最小化生产环境

**提取 Vendor 库**

捆绑所有应用特定 JavaScript 和 vendor 库的一个潜在缺点是进行长期缓存将变得更加困难，例如，单次更新应用代码将会强制浏览器下载所有 vendor 库，即使它们并没有更新。

如果你想要频繁更新应用的 JavaScript，需要考虑对 vendor 库进行提取和拆分，这样的话，对应用代码的一个修改不会影响 vendor.js 文件的缓存。Mix 的 extract 方法可以实现这样的功能：

mix.js('resources/assets/js/app.js', 'public/js')

.extract(['vue'])

extract 方法接收包含所有库的数组或你想要提取到 vendor.js 文件的模块，使用上述代码作为示例，Mix将会生成如下文件：

* public/js/manifest.js：*Webpack manifest runtime*
* public/js/vendor.js：*vendor 库*
* public/js/app.js：*应用代码*

要避免 JavaScript 错误，确保以正确顺序加载这些文件：

<script src="/js/manifest.js"></script>

<script src="/js/vendor.js"></script>

<script src="/js/app.js"></script>

**React**

Mix 可以自动为安装 Babel 插件以便支持 React，我们可以将 mix.js() 调用替换为 mix.react() 来实现：

mix.react('resources/assets/js/app.jsx', 'public/js');

在这个场景背后，Mix 会下载并引入合适的 Babel 插件 babel-preset-react。

**Vanilla JS**

和使用 mix.styles() 合并样式表类似，你可以通过 scripts() 方法合并并最小化任意数量的 JavaScript 文件：

mix.scripts([

'public/js/admin.js',

'public/js/dashboard.js'

], 'public/js/all.js');

这一功能对那些不需要 Webpack 对 Javascript 进行编译的传统应用来说很有用。

注：mix.scripts() 的一个轻微调整是 mix.babel()，它的方法签名和 scripts 一样，不同之处是合并的文件会经过 Babel 编译，从而将所有 ES2015 代码转化成所有浏览器都支持的原生 JavaScript。

**自定义 Webpack 配置**

在场景背后，Laravel Mix 引用了预配置的 webpack.config.js 文件来尽可能快的启动和运行。个别情况下，你需要手动编辑这个文件。你可能有一个被引用的特定的加载器或插件，或者可能倾向于使用 Stylus 而不是 Sass，在这些情况下，你有两个选择：

**合并自定义配置**

Mix 提供了一个有用的 webpackConfig 方法，从而允许你合并任意简短的 Webpack 配置覆盖。这是一个很吸引人的选择，因为不需要你拷贝或维护自己的webpack.config.js 文件副本，webpackConfig 方法接收一个对象，该对象包含了任意你想要应用的[Webpack指定配置](https://webpack.js.org/configuration/)：

mix.webpackConfig({

resolve: {

modules: [

path.resolve(\_\_dirname, 'vendor/laravel/spark/resources/assets/js')

]

}

});

**自定义配置文件**

第二个选择是拷贝 Mix 的 webpack.config.js 到自己的项目根目录：

cp node\_modules/laravel-mix/setup/webpack.config.js ./

接下来，将 package.json 文件中的所有 --config 引用指向拷贝后的新配置文件。如果你选择使用这种自定义方式，以后只要 Mix 的 webpack.config.js 有升级变更都要手动将变更合并到自定义的新文件。

**拷贝文件/目录**

你可以使用 copy 方法拷贝文件/目录到新路径，这在将 node\_modules 目录下的特定资源文件重新放置到 public 目录下时很有用：

mix.copy('node\_modules/foo/bar.css', 'public/css/bar.css');

拷贝目录的时候，copy 方法将会铺平目录结构，要维持目录的原始结构，需要使用 copyDirectory 方法：

mix.copyDirectory('assets/img', 'public/img');

**版本号/缓存刷新**

很多开发者会给编译的前端资源添加时间戳或者唯一令牌后缀以强制浏览器加载最新版本而不是代码的缓存副本。Mix 可以使用 version 方法为你处理这种场景。

version 方法会自动附加唯一哈希到已编译文件名，从而方便实现缓存刷新：

mix.js('resources/assets/js/app.js', 'public/js')

.version();

生成版本文件后，还不知道提取的文件名，所以，你需要在[视图](http://laravelacademy.org/post/8779.html)中使用 Laravel 全局的 mix 函数来加载相应的带哈希值的前端资源。mix 函数会自动判当前的已哈希文件名：

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

由于版本文件在本地开发中没有什么用，你可以只在运行 npm run production 期间进行版本处理操作：

mix.js('resources/assets/js/app.js', 'public/js');

if (mix.config.inProduction) {

mix.version();

}

**BrowserSync 重新加载**

[BrowserSync](https://browsersync.io/) 会自动监控文件修改，并将修改注入浏览器而不需要手动刷新，你可以通过调用 mix.browserSync() 方法启用该支持：

mix.browserSync('my-domain.test');

// Or...

// https://browsersync.io/docs/options

mix.browserSync({

proxy: 'my-domain.test'

});

你可以传递一个字符串（代理）或对象（BrowserSync 设置）到该方法。接下来，使用 npm run watch 命令来启动 Webpack 的开发服务器，现在，当你编辑一个 JavaScript 脚本或 PHP 文件时，会看到浏览器会立即刷新以响应你的修改。

**环境变量**

你可以通过在 .env 文文件添加 MIX\_ 前缀将环境变量注入 Mix：

MIX\_SENTRY\_DSN\_PUBLIC=http://example.com

在 .env 文件中定义好变量之后，可以通过 process.env 对象进行访问（如果在运行 watch 任务期间变量值有变动，需要重启任务）：

process.env.MIX\_SENTRY\_DSN\_PUBLIC

**通知**

在有效的情况下，Mix 会自动为每个捆绑显示操作系统通知，这可以给你一个及时的反馈：编译成功还是失败。不过，某些场景下你可能希望禁止这些通知，一个典型的例子就是在生产境服务器触发 Mix。通知可以通过 disableNotifications 方法被停用：

mix.disableNotifications();

https://laravelacademy.org/post/9553.html