

前端开发框架—Vue.js,webpack

Vue.js：轻量高效的前端组件化方案。

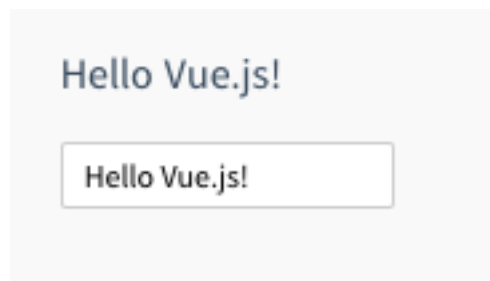
Vue.js 是一款极简的 mvvm 框架，如果让我用一个词来形容它，就是“轻·巧”。如果用一句话来描述它，它能够集众多优秀逐流的前端框架之大成，但同时保持简单易用。废话不多说，来看几个例子

：<script src="vue.js"></script>

```
<div id="demo">
  {{message}}
  <input v-model="message">
</div>
```

```
<script>
var vm = new Vue({
  el: '#demo',
  data: {
    message: 'Hello Vue.js!'
  }
})
</script>
```

首先，代码分两部分，一部分是 html，同时也是视图模板，里面包含一个值为 message 的文本和一个相同值的输入框；另一部分是 script，它创建了一个 vm 对象，其中绑定的 dom 结点是 #demo，绑定的数据是 {message: 'Hello Vue.js'}，最终页面的显示效果就是一段 Hello Vue.js 文本加一个含相同文字的输入框，更关键的是，由于数据是双向绑定的，所以我们修改文本框内文本的同时，第一段文本和被绑定的数据的 message 字段的值都会同步更新——而这底层的复杂逻辑，Vue.js 已经全部帮你做好了。

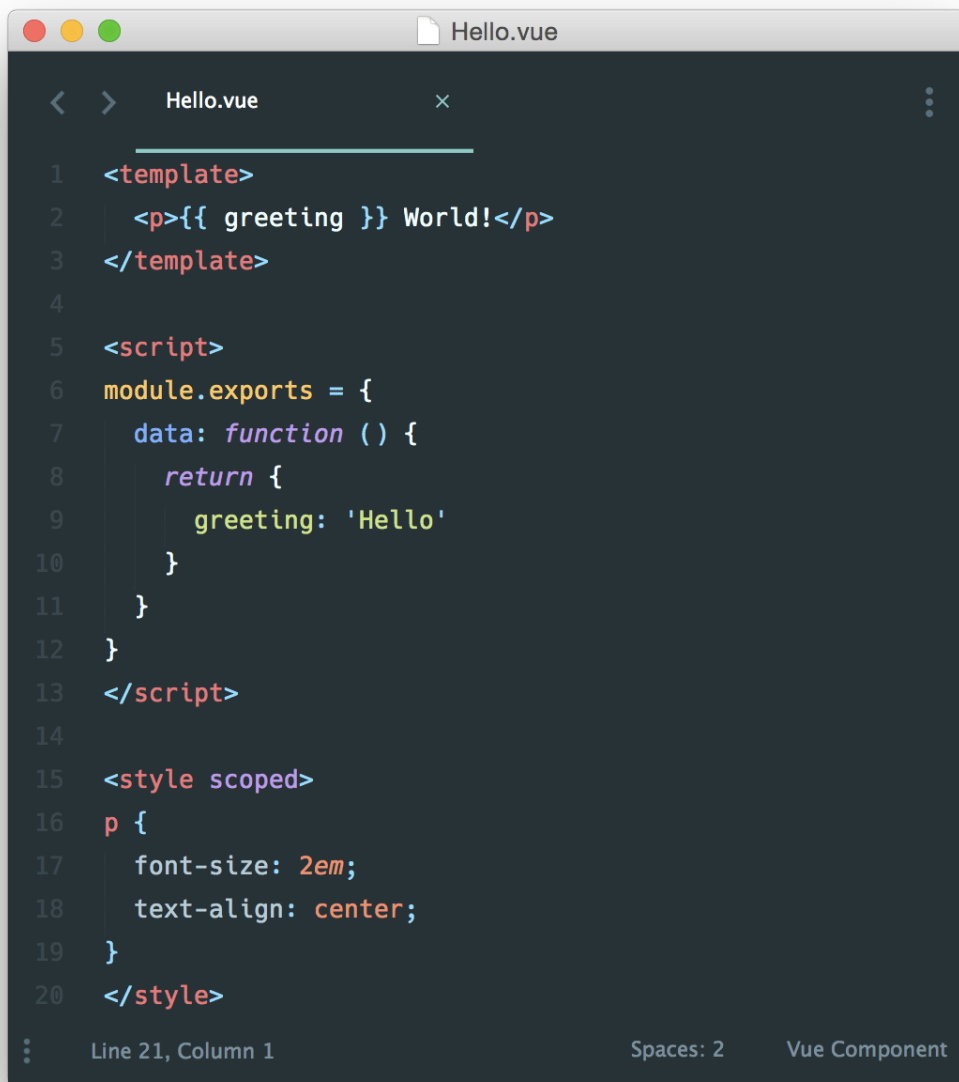


web 组件化

最后要介绍 Vue.js 对于 web 组件化开发的思考和设计

如果我们要开发更大型的网页或 web 应用，web 组件化的思维是非常重要的，这也是今天整个前端社区长久之不衰的话题。

Vue.js 设计了一个 *.vue 格式的文件，令每一个组件的样式、模板和脚本集成了一整个文件，每个文件就是一个组件，同时还包含了组件之间的依赖关系，麻雀虽小五脏俱全，整个组件从外观到结构到特性再到依赖关系都一览无余：

A screenshot of a code editor window titled 'Hello.vue'. The editor shows a Vue.js component with three sections: a template, a script, and a style section. The template section contains a paragraph element with a binding for 'greeting' and the text 'World!'. The script section contains a module.exports object with a data function that returns an object with 'greeting' set to 'Hello'. The style section contains a scoped style for the paragraph element, setting font-size to 2em and text-align to center. The editor interface includes line numbers from 1 to 20, a breadcrumb 'Hello.vue', and a status bar at the bottom indicating 'Line 21, Column 1', 'Spaces: 2', and 'Vue Component'.

从功能角度，template, directive, data-binding, components 各种实用功能都齐全，而 filter, computed var, var watcher, custom event 这样的高级功能也都洋溢着作者的巧思；从开发体验角度，这些设计几乎是完全自然的，没有刻意设计过或欠考虑的感觉，只有个别不得已的地方带了自己框架专属的 v- 前缀。从性能、体积角度评估，Vue.js 也非常有竞争力！

webpack 是另一个近期发现的好东西。它主要的用途是通过 CommonJS 的语法把所有浏览器端需要发布的静态资源做相应的准备，比如资源的合并和打包。

举个例子，现在有个脚本主文件 app.js 依赖了另一个脚本 module.js

```
// app.js
var module = require('./module.js')
... module.x ...
```

```
// module.js
exports.x = ...
```

则通过 webpack app.js bundle.js 命令，可以把 app.js 和 module.js 打包在一起并保存到 bundle.js

同时 webpack 提供了强大的 loader 机制和 plugin 机制，loader 机制支持载入各种各样的静态资源，不只是 js 脚本、连 html, css, images 等各种资源都有相应的 loader 来做依赖管理和打包；而 plugin 则可以对整个 webpack 的流程进行一定的控制。

比如在安装并配置了 css-loader 和 style-loader 之后，就可以通过 require('./bootstrap.css') 这样的方式给网页载入一份样式表。非常方便。

webpack 背后的原理其实就是把所有的非 js 资源都转换成 js (如把一个 css 文件转换成“创建一个 style 标签并把它插入 document”的脚本、把图片转换成一个图片地址的 js 变量或 base64 编码等)，然后用 CommonJS 的机制管理起来。一开始对于这种技术形态我个人还是不太喜欢的，不过随着不断的实践和体验，也逐渐习惯并认同了。

最后，对于之前提到的 Vue.js，作者也提供了一个叫做 vue-loader 的 npm 包，可以把 *.vue 文件转换成 webpack 包，和整个打包过程融合起来。所以有了 Vue.js、webpack 和 vue-loader，我们自然就可以把它们组合在一起试试看！

项目实践流程

回到正题。今天要分享的是，是基于上面两个东西：Vue.js 和 webpack，以及把它们串联起来的 vue-loader

Vue.js 的作者以及提供了一个基于它们三者的项目示例。而我们的例子会更贴近实际工作的场景，同时和团队之前总结出来的项目特点和项目流程相吻合。

目录结构设计

<components> 组件目录，一个组件一个 .vue 文件

a.vue

b.vue

<lib> 如果实在有不能算组件，但也不来自外部 (npm) 的代码，可以放在这里

foo.css

bar.js

<src> 主应用/页面相关文件

app.html 主 html

app.vue 主 vue

app.js 通常做的事情只是 var Vue = require('vue'); new Vue(require('./app.vue'))

<dist> (ignored)

<node_modules> (ignored)

gulpfile.js 设计项目打包/监听等任务

package.json 记录项目基本信息，包括模块依赖关系

README.md 项目基本介绍

打包

通过 gulpfile.js 我们可以设计整套基于 webpack 的打包/监听/调试的任务

在 gulp-webpack 包的官方文档里推荐的写法是这样的：

```
var gulp = require('gulp');
var webpack = require('gulp-webpack');
var named = require('vinyl-named');
gulp.task('default', function() {
  return gulp.src(['src/app.js', 'test/test.js'])
    .pipe(named())
    .pipe(webpack());
});
```

```
.pipe(gulp.dest('dist/'));
});
```

我们对这个文件稍加修改，首先加入 vue-loader

```
tnpm install vue-loader --save
```

```
.pipe(webpack({
  module: {
    loaders: [
      { test: /\.vue$/, loader: 'vue' }
    ]
  }
}))
```

其次，把要打包的文件列表从 gulp.src(...) 中抽出来，方便将来维护，也有机会把这个信息共享到别的任务

```
var appList = ['main', 'sub1', 'sub2']
```

```
gulp.task('default', function() {
  return gulp.src(mapFiles(appList, 'js'))
  ...
})
```

```
/**
 * @private
 */
function mapFiles(list, extname) {
  return list.map(function (app) {return 'src/' + app + '.' + extname})
}
```

现在运行 gulp 命令，相应的文件应该就打包好并生成在了 dist 目录下。然后我们在 src/*.html 中加入对这些生成好的 js 文件的引入：

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Main</title>
</head>
<body>
  <div id="app"></div>
  <script src="../dist/main.js"></script>
</body>
</html>
```

用浏览器打开 src/main.html 这时页面已经可以正常工作了

加入监听

监听更加简单，只要在刚才 webpack(opt) 的参数中加入 watch: true 就可以了。

```
.pipe(webpack({
  module: {
    loaders: [
      { test: /\.vue$/, loader: 'vue' }
    ]
  }
}))
```

```

    },
    watch: true
  )))

```

当然最好把打包和监听设计成两个任务，分别起名为 bundle 和 watch：

```

gulp.task('bundle', function() {
  return gulp.src(mapFiles(appList, 'js'))
    .pipe(named())
    .pipe(webpack(getConfig()))
    .pipe(gulp.dest('dist/'))
})

```

```

gulp.task('watch', function() {
  return gulp.src(mapFiles(appList, 'js'))
    .pipe(named())
    .pipe(webpack(getConfig({watch: true})))
    .pipe(gulp.dest('dist/'))
})

```

```

/**
 * @private
 */
function getConfig(opt) {
  var config = {
    module: {
      loaders: [
        { test: /\.vue$/, loader: 'vue' }
      ]
    }
  }
  if (!opt) {
    return config
  }
  for (var i in opt) {
    config[i] = opt[i]
  }
  return config
}

```

现在你可以不必每次修改文件之后都运行 `gulp bundle` 才能看到最新的效果，每次改动之后直接刷新浏览器即可。