爱创课堂前端培训

# Gulp与Webpack

第1天课堂笔记（本课程共1天）

班级：北京前端训练营19期

讲师：李兰波

日期：2019年1月30日

爱创课堂官网 ：[www.icketang.com](http://www.icketang.com)

# 工程化

前端工程师写的代码，分两个阶段。一个是编写阶段。一个是运行阶段。

编写阶段是人在做。所以要求有“语义”。比如，定义变量、函数名称、模块名称等。

运行阶段是机器在做。对“语义”的要求几乎没有。

所以，编写阶段的代码最终如果直接运行在运行阶段，没有太大的问题，但是不够完美。所以，对于机器来说，能够短的就别长。于是我们在代码编写完成之后，要“压缩”。所谓的压缩，指的是在不影响代码的最终执行结果的情况下，尽可能减少代码体积的一个操作。这属于工程化的一个范围。

一个HTML页面，发出了100个JS请求，得到了100个JS文件。与发送了1次请求，得到100个文件的内容总和。效果是一致的。但是过程是不一致的。第一种情况会占用浏览器的请求的并发数量。于是，就需要合并，或者叫做打包。

这也属于工程化。

如果以上的工作都由人去完成。成本太高。于是就有人写了一些工具。这些工具，能够帮助人们完成“压缩、合并等”行为。

这些工具叫做工程化工具。

# Gulp

## 2.1 简介

这是一个前端工程化工具。同时也是一个NodeJS的第三方模块。

安装：安装分两次，第一次是全局安装 负责提供gulp命令 第二次是本地安装 负责提供本地模块

|  |
| --- |
| 1. npm install gulp -g 2. npm install gulp |

第一次安装完成之后：尝试调用gulp命令

|  |
| --- |
|  |

于是开始第二次安装：

|  |
| --- |
| 1. npm install gulp |

安装完毕之后，再次调用gulp

|  |
| --- |
|  |

## 2.2 gulpfile.js

该文件用于定义gulp的使用方式。

通俗点：你想用gulp干什么，你就写在该文件里面。

demo:

|  |
| --- |
| 1. // 既然是NodeJS的第三方模块 而当前又是一个js文件 2. var gulp = require("gulp"); 3. console.log(gulp); 4. // gulp执行的时候使用的是定义任务 执行任务的方式 5. gulp.task("default", function() { 6. gulp.src("./origin/index.html") 7. .pipe(gulp.dest("./publish")); 8. }); |

执行结果：将origin/index.html 发布到了publish/index.html

## 2.3 Gulp方法

### 2.3.1 task

该方法用于定义任务

接收2个参数

第一个是任务名称

第二个是任务内容

可以是函数

可以是数组

函数：

|  |
| --- |
| 1. // 定义任务1 2. gulp.task("task1", function() { 3. console.log("1111"); 4. }); |

数组：

|  |
| --- |
| 1. // 定义任务2 2. gulp.task("task2", ["task1"]); |

注：必须有一个名字叫做“default”的任务，才可以成功执行。否则会报错

|  |
| --- |
|  |

### 2.3.2 src

该方法用于定位资源 当程序执行的时候，它会将src所指向的那些文件，都读取到内存中。

接收1个参数： 该参数最终表达的是一个、些文件的位置

可以是字符串

可以是数组

可以是glob

字符串：

|  |
| --- |
| 1. gulp.src("./origin/index.html"); |

数组：

|  |
| --- |
| 1. gulp.src(["./origin/index.html"]) |

glob:

|  |
| --- |
| 1. gulp.src("./origin/\*.html") |

### 2.3.3 pipe

该方法不是gulp对象的方法，而是src方法返回的对象的方法。

该方法用于定义如何操作读取到的文件。

该方法可以连续多次调用。每一次接收一个“指令”。也就是一次操作。

demo:

|  |
| --- |
| 1. gulp.src("./origin/\*.html") 2. // 发布到publish下 3. .pipe(gulp.dest("./publish/")); |

注：红色部分是一次操作。

### 2.3.4 dest

该方法用于将操作完毕的文件内容，从内存中输出到磁盘中。

参数是一个路径字符串

demo:

|  |
| --- |
| 1. gulp.dest("./publish/") |

### 2.3.5 watch

该方法用于持续观察某一个、些文件，当这些文件保存时，自动执行某个、些任务。

接收2个参数：

第一个参数是目标文件的路径

可以是字符串

可以是数组

可以是glob

第二个参数是要执行的内容

可以是函数

可以是数组

demo:

|  |
| --- |
| 1. // 调用watch 2. gulp.watch("./origin/index.html", ["default"]); |

## 2.4 glob表达式

### 2.4.1 单层级全部

\* 表示单层级全部

demo：

|  |
| --- |
|  |

result：

|  |
| --- |
|  |

glob:./origin/\*.js 它表示将origin目录下的所有js文件选中 但是不会选中更下一层级的内容

### 2.4.2 多层级全部

\*\* 表示多层级全部

demo:

|  |
| --- |
| 1. // 定位资源 2. gulp.src("./origin/\*\*/\*.js") 3. .pipe(gulp.dest("./publish")); |

层级关系

|  |
| --- |
|  |

结果：

|  |
| --- |
|  |

### 2.4.3 范围

[] 表示一定的范围 在范围内的选中

demo:

|  |
| --- |
|  |

result:

|  |
| --- |
|  |

globs:./origin/[abc].js

### 2.4.4 范围取反

范围的意思，是：“选中在范围内包括的内容”

范围取反，就是不要这些

demo:

|  |
| --- |
|  |

result:

|  |
| --- |
|  |

globs: "./origin/[^abc].js"

### 2.4.5 任意字符

？表示任意单个字符

demo:

|  |
| --- |
|  |

result:

|  |
| --- |
|  |

globs: "./origin/??.js"

## 2.5 压缩JS插件

插件名称： gulp-uglify

下载： npm install gulp-uglify

|  |
| --- |
|  |

使用：

|  |
| --- |
| 1. // 引入gulp 2. var gulp = require("gulp"); 3. // 引入压缩JS的插件 4. var uglify = require("gulp-uglify"); 5. // 定义任务 6. gulp.task("default", function() { 7. // 定位资源 8. gulp.src("./origin/\*.js") 9. // 定义操作 压缩 10. .pipe(uglify()) 11. // 定义操作 发布 12. .pipe(gulp.dest("./publish/")); 13. }); |

压缩之前：265KB

|  |
| --- |
|  |

压缩之后：84.3KB

|  |
| --- |
|  |

## 2.6 压缩CSS

插件名称： gulp-clean-css

下载：npm install gulp-clean-css

|  |
| --- |
|  |

使用：

|  |
| --- |
| 1. // 引入gulp 2. var gulp = require("gulp"); 3. // 引入压缩CSS的插件 4. var clean = require("gulp-clean-css"); 5. // 定义任务 6. gulp.task("default", function() { 7. // 定位资源 8. gulp.src("./origin/\*.css") 9. // 定义操作 压缩 10. .pipe(clean()) 11. // 定义操作 发布 12. .pipe(gulp.dest("./publish/")); 13. }); |

压缩之前：142KB

|  |
| --- |
|  |

压缩之后：117KB

|  |
| --- |
|  |

因为CSS规则的特殊性，选择器、属性名、属性值都不可以压缩 能够允许压缩的有注释、回车、换行等。

## 2.7 压缩HTML

插件名称：gulp-minify-html

下载：npm install gulp-minify-html

|  |
| --- |
|  |

使用:

|  |
| --- |
| 1. var gulp = require("gulp"); 2. var minify = require("gulp-minify-html"); 3. gulp.task("default", function() { 4. gulp.src("./origin/\*.html") 5. .pipe(minify()) 6. .pipe(gulp.dest("./publish")) 7. }) |

压缩之前：9.62KB

|  |
| --- |
|  |

压缩之后: 4.33KB

|  |
| --- |
|  |

## 2.8 压缩图片

插件名称：gulp-imagemin

下载: npm install gulp-imagemin

使用：

|  |
| --- |
| 1. var gulp = require("gulp"); 2. var imagemin = require("gulp-imagemin"); 3. gulp.task("default", function() { 4. gulp.src("./origin/\*.png") 5. .pipe(imagemin()) 6. .pipe(gulp.dest("./publish/")) 7. }) |

压缩之前：7.69KB

|  |
| --- |
|  |

压缩之后：4.77KB

|  |
| --- |
|  |

## 2.9 合并文件

模块名称：gulp-concat

下载: npm install gulp-concat

使用:

|  |
| --- |
| 1. var gulp = require("gulp"); 2. var gulp\_concat = require("gulp-concat"); 3. gulp.task("default", function() { 4. gulp.src(["./origin/jquery.min.js", "./origin/bootstrap.min.js"]) 5. .pipe(gulp\_concat("jb.js")) 6. .pipe(gulp.dest("./publish")) 7. }); |

注：src方法在合并的时候，如果使用数组，则合并顺序按照数组来排序。如果使用globs，会按照字母顺序排序。

## 2.10 重命名

插件名称：gulp-rename

下载: npm install gulp-rename

使用:

|  |
| --- |
| 1. var gulp = require("gulp"); 2. var rename = require("gulp-rename"); 3. var uglify = require("gulp-uglify"); 4. gulp.task("default", function() { 5. gulp.src("./origin/\*.js") 6. .pipe(uglify()) 7. .pipe(rename({ 8. prefix: "ickt.", 9. suffix: ".min." 10. })) 11. .pipe(gulp.dest("./publish")) 12. }); |

rename函数所接收的参数对象的属性：

dirname: 文件夹路径

basename: 文件名

prefix: 前缀

suffix: 后缀

extname: 拓展名

## 2.11 语法校验

模块名称：gulp-jslint

下载: npm install gulp-jslint

使用:

|  |
| --- |
| 1. var gulp = require("gulp"); 2. var jslint = require("gulp-jslint"); 3. gulp.task("default", function() { 4. gulp.src("./origin/a.js") 5. .pipe(jslint()) 6. .pipe(jslint.reporter("default")) 7. }); |

通过校验时：

|  |
| --- |
|  |

# WebPack

这是一个模块化打包工具。

（Node使用CommonJS模块规范，内置的require命令用于加载模块文件。

require命令的基本功能是，读入并执行一个JavaScript文件，然后返回该模块的exports对象。如果没有发现指定模块，会报错。）

## 2.1 安装

与Gulp类似，也是需要两次安装。

第一次，全局安装 提供webpack命令

|  |
| --- |
| 1. npm install webpack -g |

第二次，本地安装 提供webpack模块文件

|  |
| --- |
| 1. npm install webpack |

当安装完毕之后，确实可以使用webpack命令，但是会提示有内容没有安装，这是一个webpack-cli的模块。我们需要将它也安装两次

|  |
| --- |
| 1. npm install webpack-cli -g 2. npm install webpack-cli |

## 2.2 webpack.config.js

与Gulp一致，webpack也需要一个文件。但是又有不同。

Gulp中执行的内容，是任务。

Webpack中执行的是各种打包，以及配置。

Webpack遵循的是CommonJS规范。

demo:

|  |
| --- |
| 1. // 这个文件是配置文件 2. module.exports = { 3. // 入口 4. entry: "./modules/main", 5. // 出口 6. output: { 7. // 文件夹所在 该配置必须是绝对路径 8. path: \_\_dirname, 9. // 文件名称 10. filename: "aaa.js" 11. } 12. } |

## 2.3 概念介绍

模块：对于webpack来说，一切都是模块。

打包：webpack最终会生成一个打包的整个的文件。

压缩：webpack现在已经是自动压缩内容了。

入口：打包开始的入口 打包的起点。

出口：打包最终生成的文件

加载机：webpack默认只加载js。其余的文件不加载。此时就需要加载机的帮助。

插件：各种插件。

## 2.4 入口的配置

单页面应用程序：因为只有一个入口，只需要打一个包，所以最终的包名可以定死。所以入口就只需要写文件的入口地址即可。

|  |
| --- |
| 1. module.exports = { 2. // 入口 因为这是一个单页面应用程序 所以最终只需要打成一个包即可。 3. entry: "./modules/index\_main.js", 4. // 出口 5. output: { 6. path: \_\_dirname, 7. filename: "aaa.js" <= 写死 8. } 9. } |

多页面应用程序：因为有多个页面，所以有多个入口，需要打多个包，所以最终的包名不可以定死。所以就需要使用占位符去占位。[name] 该位置是留给entry的key的。

|  |
| --- |
| 1. module.exports = { 2. // entry入口 3. entry: { 4. // key表示最终生成的打包文件的名 5. // value表示入口文件地址 6. hahahaha: "./modules/index\_main.js", 7. nihao: "./modules/portfolio\_main.js" 8. }, 9. // output出口 10. output: { 11. path: \_\_dirname, 12. filename: "[name].js" // [name] 表示占位符 name表示文件名 具体指的就是entry的每一个key 13. } 14. } |

最终，会生成两个文件： hahahaha.js 和 nihao.js

其中，hahahaha.js是由 ./modules/index\_main.js为入口

nihao.js 是由 ./modules/portfolio\_main.js为入口

## 2.5 CSS加载机

webpack天生能够处理JS，但是无法处理其它类型的文件。

此时，需要加载机的帮助。

尝试加载CSS文件时：

|  |
| --- |
|  |

处理CSS文件的加载机:

|  |
| --- |
| 1. style-loader css-loader |

下载:

|  |
| --- |
| 1. npm install style-loader css-loader |

配置：

|  |
| --- |
| 1. module: { 2. // 高版本中 这里是rules 以前的版本中 这里是 loaders 3. rules: [ 4. { 5. test: /\.css$/, 6. loaders: "style-loader!css-loader" 7. } 8. ] 9. } |

## 2.6 图片加载机

加载机模块：url-loader

如果不使用url-loader 则会报错

|  |
| --- |
|  |

下载：

|  |
| --- |
|  |

再次尝试：

|  |
| --- |
| 1. module.exports = { 2. entry: "./modules/main.js", 3. output: { 4. path: \_\_dirname, 5. filename: "bundle.js" 6. }, 7. module: { 8. rules: [ 9. { 10. test: /\.(jpg|gif|png|jpeg|svg)$/, 11. loaders: "url-loader?limit=40960" 12. } 13. ] 14. } 15. } |

limit: 表示限制，当图片的体积小于该数值时，就将该图片转为base64编码字符串。嵌入在页面中，如果大于该数值，则不作处理，发送http请求。

## 2.7 webpack-dev-server

这是一个webpack开发服务器

需要两次安装:

|  |
| --- |
| 1. npm install webpack-dev-server -g 2. npm install webpack-dev-server |

主要作用就是开启一个服务器。你可以在服务器下实时体验到打包之后的效果。

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |