# 工程化

简单举例：

在代码书写过程中，我们追求的是“语义化”。能够让别“人”看得懂你的代码。

在代码运行过程中，我们追求的是“体积小”。能够让传输过程尽可能的减短。

代码书写是给人看。代码运行是机器在运行。于是我们就想办法将代码“压缩”。在保证执行结果不变的情况下，将代码文件的体积尽可能的减小。

这是一种工程化。

# Gulp

## 2.1 简介

Gulp是一个NodeJS的第三方模块。用于工程化。

注：Gulp本身只是一个定义任务的工具。具体的实现依赖插件。

## 2.2 安装

注：因为最新版本的gulp是4.x 使用的是ES6语法，所以我们要学习的是3.x版本的书写方式。

安装需要两次：

npm install gulp@3.9.1 -g 全局安装

npm install gulp@3.9.1 本地安装

第一次安装：全局安装

|  |
| --- |
|  |

安装完毕之后，就可以立即在控制台中调用 gulp 命令：当然，会报错

|  |
| --- |
|  |

根据提示，我们进行第二次安装：

注：千万要记得统一版本 如果不统一版本，则会出现各种情况的错误。



gulp -v 查看版本号

现在下载正常的版本：

|  |
| --- |
|  |

## 2.3 配置文件

下载完毕模块之后，我们要开始决定怎样去进行工程化行为。

由什么文件决定，哪些文件需要压缩，哪些文件需要合并呢？

Gulp提供了一个配置文件：gulpfile.js

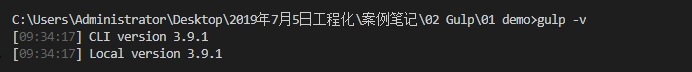
在该文件下，我们可以定义“任务”。每一个任务实现一个功能或者多个功能。

gulpfile.js:

|  |
| --- |
| // 引入gulp  var gulp = require("gulp");  // 定义任务  gulp.task("default", function() {  gulp.src("./origin/\*.js")  .pipe(gulp.dest("./publish"));  }); |

注：必须定义一个default任务。作为入口。

安装完毕的标识：



## 2.4 task

gulp.task(name, content); 该方法用于定义“任务”。

name: 任务名称，是一个字符串

content: 执行内容

可以是函数 表示当任务被执行的时候所运行的代码。

可以是数组 表示当任务被执行的时候所执行的多个任务集合。

demo1:

|  |
| --- |
| 1. // 定义任务 第一种，第二个参数是函数 2. gulp.task("default", function() { 3. console.log("任务执行"); 4. }); |

demo2:

|  |
| --- |
| // 定义任务 第二种，第二个参数是数组  gulp.task("default", ["a", "b", "c"]);  gulp.task("a", function() {  console.log("a");  });  gulp.task("b", function() {  console.log("b");  });  gulp.task("c", function() {  console.log("c");  }); |

## 2.5 src

gulp.src(path); 该方法用于定位资源 它的作用是将选定的资源从物理磁盘中，读入到内存中。

path: 被定位的资源的路径

有三种写法：

直接写某一个文件的路径

数组

glob表达式

demo1: 文件路径

|  |
| --- |
| 1. gulp.src("./origin/a.js") |

demo2:

|  |
| --- |
| 1. gulp.src(["./origin/a.js", "./origin/b.js"]) |

demo3:

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

## 2.6 pipe

注：pipe不是gulp的方法 而是gulp.src的返回值的方法

该方法用于定义操作行为。针对的是通过gulp.src选中的那些文件。

pipe是管道的意思，gulp执行任务时，使用的是管道流，

比如一个工程化任务包含：压缩、合并、重命名这三方面

pipe会建立三个管道也就是调用三次pipe方法

第一个pipe里放的是压缩任务

第二个pipe里放的是合并任务

第三个pipe里放的是重命名任务

每一个pipe的输出 作为下一个pipe的输入。

## 2.7 dest

gulp.dest(dirPath) 是gulp内置的唯一的一个操作行为。用于将经过pipe处理完毕的结果，发布到某一个目录下

dirPath: 被发布到的目录 如果该目录存在，则使用，如果不存在，则先创建，再使用

demo:

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

## 2.8 watch

gulp.watch(target, content); 该方法用于监听、监视某一个、些文件。当这些文件发生了变化的时候，自动执行后面的任务

target: 目标

单个文件路径

数组

glob表达式

content: 执行内容

函数

数组 数组中的每一个都是其它任务的名称

demo:

|  |
| --- |
| var gulp = require("gulp");  // 这一条的存在意义就是不让gulp报错  gulp.task("default", function() {});  // 监视./origin/a.js文件 每当它发生变化的时候就发布到publish下  gulp.watch("./origin/a.js", function() {  gulp.src("./origin/a.js")  .pipe(gulp.dest("./publish"));  }); |

# Glob表达式

Glob表达式与正则表达式类似，但是又有区别。

glob表达式只用于路径中。

## 3.1 多层级

\*\* 表示多个层级 gulp.src("./origin/\*\*/\*.js")

## 3.2 单层级

\* 表示单个层级 gulp.src("./origin/\*/\*.js")

\* 如果出现在文件名中，表示“全部”的意思

## 3.3 单个字符分组

[abc] 表示当前的字符可以是 abc中的任何一个

gulp.src("./origin/a[abc]c.js")

以上的表达式会选中 origin目录下的aac.js abc.js acc.js 文件

## 3.4 单个字符反选分组

[^abc] 表示当前的字符可以是除了abc之外的任何一个

gulp.src("./origin/a[^abc]c.js")

以上的表达式

不会选中 origin目录下的aac.js abc.js acc.js 文件

会选中以a开头，中间有一个字符，以c结尾的其它所有文件

## 3.5 单个字符

? 表示单个任意字符

gulp.src("./origin/?.js")

以上的表达式会选中origin下的任何一个字母文件名的js文件。

# 插件

## 4.1 压缩JS

插件名称: gulp-uglify

使用:

|  |
| --- |
| 1. // 引入gulp   var gulp = require("gulp");  // 引入压缩插件  var uglify = require("gulp-uglify");  // 定义默认任务  gulp.task("default", function() {  // 定位文件  gulp.src("./origin/jquery.js")  // 第一根管道  .pipe(uglify())  // 第二根管道  .pipe(gulp.dest("./publish"));  }); |

压缩之前:

|  |
| --- |
|  |

压缩之后：

|  |
| --- |
|  |

## 4.2 压缩CSS

插件名称：gulp-clean-css

使用:

|  |
| --- |
| var gulp = require("gulp");  var clean = require("gulp-clean-css");  // 定义默认任务  gulp.task("default", function() {  gulp.src("./origin/bootstrap.css")  .pipe(clean())  .pipe(gulp.dest("./publish"));  }); |

压缩之前：

|  |
| --- |
|  |

压缩之后：

|  |
| --- |
|  |

## 4.3 压缩HTML

插件名称：gulp-minify-html

使用：

|  |
| --- |
| var gulp = require("gulp");  var minify = require("gulp-minify-html");  // 定义默认任务  gulp.task("default", function() {  gulp.src("./origin/regist.html")  .pipe(minify())  .pipe(gulp.dest("./publish"));  }); |

压缩之前：

|  |
| --- |
|  |

压缩之后：

|  |
| --- |
|  |

## 4.4 压缩图片

插件名称：gulp-imagemin

使用：

|  |
| --- |
| var gulp = require("gulp");  var imagemin = require("gulp-imagemin");  // 定义默认任务  gulp.task("default", function () {  // 定位资源  gulp.src(["./origin/bd\_logo1.png", "./origin/default.jpg"])  .pipe(imagemin())  .pipe(gulp.dest("./publish"));  }); |

压缩结果：

|  |
| --- |
|  |

## 4.5 合并

插件名称：gulp-concat

使用方式：

|  |
| --- |
| var gulp = require("gulp");  var concat = require("gulp-concat");  var uglify = require("gulp-uglify");  // 定义默认任务  gulp.task("default", function() {  // 定位资源  gulp.src("./origin/\*.js")  .pipe(concat("ickt.js"))  .pipe(gulp.dest("./publish"));  }); |

合并之后：

|  |
| --- |
|  |

# 复习

工程化：写好代码之后的打包、编译、合并、压缩等行为。

Gulp：是一个NodeJS第三方模块，专门用于工程化。

安装：

第一次，安装到全局 npm install [gulp@3.9.1](mailto:gulp@3.9.1) -g

第二次，安装到本地 npm install [gulp@3.9.1](mailto:gulp@3.9.1)

注：所谓的本地，指的是你的开发目录

检测是否安装完毕： gulp -v

配置文件：

使用gulp需要配置一个配置文件 文件名称gulpfile.js

里面需要书写node代码，代码中用到了gulp模块的功能。

自带的方法：

task(taskName, content); 该方法用于定义一个任务 注：必须有一个default名称的任务

taskName: 任务名称，字符串

content: 任务执行代码或者是数组 数组中的每一项都是其它任务的名称

src(target); 该方法用于定位资源，会将定位到的资源从磁盘中读取到内存中

target: 目标的路径书写方式

第一种：单个路径字符串

第二种：数组 数组中的每一个都是单个路径字符串

第三种：glob表达式

pipe(operate); 该方法是src方法的返回值的方法。用于定义一次操作。可以链式调用，每一次都表示一次操作。上一次的操作结果会作为下一次的参数。

operate: 要执行的操作 比如 gulp.dest、uglify、minify、concat等

dest(path); 该方法是用于发布文件到某个位置的方法。与src方法相反。用于将内存中的操作完毕的内容，重新放回磁盘中。

path: 放回到磁盘中的位置，字符串。

watch(target, content); 该方法用于监听、监视某一个、些文件。当这些文件发生变化时，会自动执行任务。

target: 与src方法的参数一致

content: 与task的第二个参数一致

Glob表达式：

\*\* 表示多个层级（其实是任意层级）

\* 如果在路径中，表示单个层级 如果在文件名中，表示任意

[abc] 单个字符分组 表示该位置可以选中的字符，只能从[]中选择

[^abc] 单个字符反选 表示该位置可以选中的字符，不能出现在[]中

? 任意单个字符

插件： gulp本身并不能够做任何工程化的事情。所有的一切都依赖插件完成。

压缩JS:

插件名称： gulp-uglify

使用： pipe(require("gulp-uglify")())

压缩CSS:

插件名称： gulp-clean-css

使用： pipe(require("gulp-clean-css")())

压缩HTML:

插件名称： gulp-minify-html

使用： pipe(require("gulp-minify-html")())

压缩图片：

插件名称： gulp-imagemin

使用： pipe(require("gulp-imagemin")())

合并：

插件名称： gulp-concat

使用： pipe(require("gulp-concat")(新文件的名称))

## 4.6 重命名

插件名称： gulp-rename

使用：

|  |
| --- |
| gulp.task("default", function() {  // 定位资源  gulp.src("./origin/a.txt")  // .pipe(rename("b.txt"))  .pipe(rename({  prefix: "ickt.",  suffix: ".min",  basename: "aaa",  extname: ".js"  }))  .pipe(gulp.dest("./publish"));  }); |

## 4.7 语法校验

插件名称： gulp-jslint

使用：

|  |
| --- |
| // 引入gulp  var gulp = require("gulp");  var jslint = require("gulp-jslint");  // 定义默认任务  gulp.task("default", function() {  gulp.src("./origin/jquery.js")  .pipe(jslint())  .pipe(jslint.reporter("default"));  }); |

出错：

|  |
| --- |
|  |

通过：

|  |
| --- |
|  |

# Webpack

## 5.1 简介

Webpack也是一个NodeJS的第三方模块。

它是一个模块化打包工具。

天生只能够打包JS。

在Webpack看来，一切都是模块。

## 5.2 安装

webpack的安装也分两次。

第一次，全局安装：安装到C:\Users\Administrator\AppData\Roaming\npm

npm install webpack -g

npm install webpack-cli -g

第二次，本地安装：安装到命令所在目录（遵循npm下载的规则）

npm install webpack

npm install webpack-cli

总计安装四个包。

全局安装：npm install webpack -g

|  |
| --- |
|  |

此时，已经可以调用webpack命令，但是依然会提示：

|  |
| --- |
|  |

意思是：请安装webpack-cli这个模块

我们可以自己安装webpack-cli模块，安装到全局:

|  |
| --- |
|  |

此时，真的可以无障碍调用webpack命令了。但是因为需要配置文件，所以会报错。

|  |
| --- |
|  |

## 5.3 配置文件

webpack提供了一个webpack.config.js，作为webpack的配置文件。

我们重点学习的就是该文件如何配置。

重点概念：

入口：webpack打包的入口。

出口：打包完成之后生成的那个文件的相关信息

加载机：因为webpack天生只能够识别js文件，所以想要加载其它的类型文件，就需要加载机的帮助。

插件：webpack自身不具备的功能。

webpack.config.js: 示例

|  |
| --- |
| 1. // 当前是webpack.config.js 2. // webpack是一个模块化打包工具。 3. // 模块化的规范？ 遵循的是CommonJS规范。因为它是一个NodeJS第三方模块。 4. module.exports = { 5. // 入口 6. entry: "./js/main.js", 7. // 出口 8. output: { 9. path: \_\_dirname, 10. filename: "./abcd.js" 11. } 12. } |

### 5.3.1 入口的配置方式

entry表示入口。也就是打包开始的地方。

它的属性值可以是单个字符串，表示入口文件。 对应的是单页面应用程序。(Single Page Application)

它的属性值还可以是对象，表示多个入口文件。 对应的是多页面应用程序。

注：一个入口文件对应一个页面。

单页面：

|  |
| --- |
| 1. module.exports = { 2. entry: "./modules/main.js", 3. output: { 4. filename: "ab.js" 5. } 6. } |

此时，entry是一个字符串，只表示出了“入口所在”这一层含义。

所以，output的filename就必须指定具体的打包之后的名称。

多页面：

|  |
| --- |
| 1. // 当前的项目中有两个页面（多页面） 2. // index.html中有一堆逻辑代码 second.html中也有一堆逻辑代码 3. // 此时，需要给将index.html中所执行的代码打包成一个文件， 需要给second.html中所执行的代码打包成一个文件 4. // 也就是需要打包两个文件。 5. module.exports = { 6. // 所以此时entry就不能再是一个字符串 而应该是多个字符串 7. entry: { 8. // 此时key就表示打包之后的文件的名称 9. // value表示入口 10. "index": "./modules/index/main.js", 11. "second": "./modules/second/main.js" 12. }, 13. output: { 14. path: \_\_dirname, 15. filename: "[name].js" 16. } 17. } |

### 5.3.2 出口的配置方式

出口是配置打包之后的文件，存放的位置的。

有两个配置项：

path: 决定打包之后的文件所存储的目录

必须是绝对路径 可以不配置 默认是 \_\_dirname + "/dist"

filename: 决定打包之后的文件名

必须的选项

出口配置一：

|  |
| --- |
| 1. module.exports = { 2. entry: "./modules/main.js", 3. output: { 4. filename: "ab.js" 5. } 6. } |

出口配置二：

|  |
| --- |
| 1. module.exports = { 2. // 所以此时entry就不能再是一个字符串 而应该是多个字符串 3. entry: { 4. // 此时key就表示打包之后的文件的名称 5. // value表示入口 6. "index": "./modules/index/main.js", 7. "second": "./modules/second/main.js" 8. }, 9. output: { 10. path: \_\_dirname, 11. filename: "[name].js" 12. } 13. } |

注： [name]表示的是代替名称的含义。最终使用的值，是entry的属性名。

### 5.3.3 加载机的配置

加载机的使用场景：需要加载非JS文件时，需要用到加载机。

比如CSS：

main.js中有一条代码：该css文件中有css代码

|  |
| --- |
| 1. require("./xxx.css"); |

如果此时配置如下：

|  |
| --- |
| 1. module.exports = { 2. entry: "./modules/main.js", 3. output: { 4. path: \_\_dirname, 5. filename: "./bundle.js" 6. } 7. } |

则会报错：

|  |
| --- |
|  |

这就是需要加载机。

## 5.4 加载CSS

因为加载CSS是超出webpack原生能力范围之外的。所以需要加载机的帮助。

加载CSS文件需要加载机模块：

style-loader css-loader

下载这两个模块： npm install style-loader css-loader

加载机的配置：

|  |
| --- |
| 1. module.exports = { 2. entry: "./modules/main.js", 3. output: { 4. path: \_\_dirname, 5. filename: "./bundle.js" 6. }, 7. // 配置加载机的配置项 8. module: { 9. // 配置加载机的规则 对应的是一个数组 每一个数组成员是一个配置对象 用于配置一个加载机 10. rules: [ 11. { 12. // test配置项 是一个正则表达式 用于匹配文件名 13. test: /\.css$/, 14. // 加载css文件所需的模块 15. loaders: "style-loader!css-loader" 16. } 17. ] 18. } 19. } |

## 5.5 加载图片

加载图片所使用的加载机模块： url-loader、file-loader

配置：

|  |
| --- |
| 1. module.exports = { 2. entry: "./modules/main.js", 3. output: { 4. path: \_\_dirname, 5. filename: "./bundle.js" 6. }, 7. // 配置加载机的配置项 8. module: { 9. // 配置加载机的规则 对应的是一个数组 每一个数组成员是一个配置对象 用于配置一个加载机 10. rules: [ 11. // 匹配加载图片的加载机 12. { 13. test: /\.(png|jpg|gif)$/, 14. loader: "url-loader" 15. } 16. ] 17. } 18. } |

打包之后的图片，是base64字符串

|  |
| --- |
|  |

分析：

base64的目的，是为了减少http请求数量。为了优化页面的加载。但是如果base64的体积，还要比原图片或者资源的体积要大。反而得不偿失。

所以，通常会对资源的尺寸要求有限制。比如大于10KB的图片就不再转为base64了。

限制方式：

|  |
| --- |
| 1. module.exports = { 2. entry: "./modules/main.js", 3. output: { 4. path: \_\_dirname, 5. filename: "./bundle.js" 6. }, 7. // 配置加载机的配置项 8. module: { 9. // 配置加载机的规则 对应的是一个数组 每一个数组成员是一个配置对象 用于配置一个加载机 10. rules: [ 11. // 匹配加载图片的加载机 12. { 13. test: /\.(png|jpg|gif)$/, 14. loader: "url-loader?limit=" + 1024 \* 10 15. } 16. ] 17. }, 18. mode: "development" 19. } |

该功能，需要file-loader的支持。

经过限制之后，因为jpg图片是大于10KB的，另外一个不大于10Kb。如下：

|  |
| --- |
|  |

## 5.6 webpack服务器

webpack提供了一个模块，用于搭建webpack的服务器。

模块名称： webpack-dev-server

安装：需要两次安装

第一次：全局安装 npm install webpack-dev-server -g

第二次：本地安装 npm install webpack-dev-server

安装完毕之后，就可以使用webpack-dev-server命令，代替webpack命令。

与此同时会搭建起一个服务器。每一次的修改会自动重启服务器。而且你不会发现真正的编译出来的文件。因为它们都在内存中。

第一次安装：

|  |
| --- |
|  |

调用：

|  |
| --- |
|  |

此时我们可以通过访问服务器: http://localhost:8080/ 测试我们的当时的书写代码状态。

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |