原 荐 前端打包构建工具Gulp、Rollup、Webpack、Webpack-stream

别人说我名字很长 发表于 20小时前 阅读 1296 收藏 77 点赞 1 评论 5

☆ 收藏

摘要: Gulp、Rollup、Webpack、Webpack-stream使用入门

Gulp
rollup
命令行方式
javascript 调用方式
webpack
安装webpack-dev-ser
安装Babel加载器
CSS加载器
autoprefixer
代码压缩
gulp与webpack-strea

Gulp

gulp是一个前端自动化构建工具，通过代码优于配置的策略，Gulp 让简单的任务简单，复杂的任务可管理。

全局安装

```
npm install --global gulp
```

作为项目的开发依赖(devDependencies)安装

```
npm install --save-dev gulp
```

在项目根目录下创建一个名为`glupfile.js`的文件

```
var gulp = require('gulp');

gulp.task('default', function() {
  // 将你的默认的任务代码放在这
});
```

运行`gulp`

```
gulp
```

实验

> 我们使用gulp-uglify来压缩js文件，减少文件大小

在刚才建立的项目里安装`gulp-uplify`插件

```
npm install gulp-uglify --save-dev
```

再安装一个合并文件的插件`gulp-concat`

```
npm install gulp-concat --save-dev
```

修改上面的`glupfile.js`

```
var gulp = require('gulp'),
uglify = require('gulp-uglify'),
concat = require('gulp-concat');

//建立一个名为jsmin的任务
gulp.task('jsmin',function(){
    // 压缩src目录下(**表示子目录)的所有.js文件，
    // 压缩test/test.js文件，
    // 不包括src目录（**子目录）下的test1.js和test2.js
    // 作为下一个任务的依赖，我们返回这个执行流，这样就可以在本任务执行完后执行下一个任务
    return gulp.src(['src/**/*.js','test/test.js','!src/**/{test1,test2}.js']) //多个文件以数组形式传入
        .pipe(uglify()) //使用相应的工具
```

```
        .pipe(gulp.dest('dist/js')) //执行压缩后保存的文件夹
});

//建立一个名为testConcat的任务,在jsmin任务执行完后执行
gulp.task('testConcat',['jsmin'],function(){
    gulp.src('dist/js/**/*.js')
        .pipe(concat('all.js')) //合并之后的文件名
        .pipe(gulp.dest('dist/js2')) //合并之后保存的路径
})

//建立一个默认执行的任务，这个任务顺序执行上面创建的N个任务
gulp.task('default',['testConcat'])
```

命令行里运行`gulp`命令，就发现dist/js2/all.js 合并压缩了之前指定的文件，上面的例子本可以在一个任务里完成，我是为了演示多任务依赖而分成了两个任务。

相关资料

> "
>
> http://www.ydcss.com/archives/54

rollup

Rollup 是下一代的 javascript 打包器，它使用 tree-shaking 的技术使打包的结果只包括实际用到的 exports。 使用它打包的代码，基本没有冗余的代码，减少了很多的代码体积

命令行方式

全局安装Rollup

```
npm install -g rollup
```

打包命令

```
rollup src/index.js -o  bundle.js -f cjs
```

javascript 调用方式

新建package.json文件

```
npm init
```

安装依赖

打开package.json文件，增加依赖项

```
"devDependencies": {
    "babel-core": "^6.22.1",
    "babel-preset-es2015-rollup": "^3.0.0",
    "rollup": "^0.41.4",
    "rollup-plugin-babel": "^2.7.1",
    "rollup-plugin-commonjs": "^7.0.0",
    "rollup-plugin-node-resolve": "^2.0.0",
    "rollup-plugin-uglify": "^1.0.1"
  }
```

命令行运行

```
npm install
```

编写打包的代码

建一个build.js文件，内容如下

```
var rollup = require('rollup');
var babel = require('rollup-plugin-babel');
var uglify = require('rollup-plugin-uglify');
var npm = require('rollup-plugin-node-resolve');
var commonjs = require('rollup-plugin-commonjs');

rollup.rollup({
    entry: 'src/index.js', //打包入口文件
    plugins: [
        npm({ jsnext: true, main: true }),
```

```
        uglify(), //压缩代码
        commonjs(), //支持CommonJS模块语法
        babel({  //babel配置
            exclude: 'node_modules/**',
            presets: [ "es2015-rollup" ]
        })
    ]
}).then(function(bundle) {
    bundle.write({
         // output format - 'amd', 'cjs', 'es6', 'iife', 'umd'
       // amd – 使用像requirejs一样的银木块定义
       // cjs – CommonJS，适用于node和browserify / Webpack
       // es6 (default) – 保持ES6的格式
       // iife – 使用于<script> 标签引用的方式
       // umd – 适用于CommonJs和AMD风格通用模式
        format: 'cjs',  //指定要打包成什么格式
        dest: 'dist/main.js' //编译完的文件需要被存放的路径
    });
});
```

打包

```
node build.js
```

相关资料

> http://blog.csdn.net/gccll/article/details/52785754
>
> http://www.tuicool.com/articles/q26jEjz

webpack

webpack是一个前端模块打包器

安装命令

```
npm install webpack -g
```

使用webpack

```
npm init #会生成package.json文件
npm install webpack --save-dev #将webpack增加到package.json文件中
```

配置文件

根目录下新建webpack.config.js

```
module.exports = {
    //页面入口文件配置
    entry:{
        index:'./src/index.js'
    },
    //编译输出配置
    output:{
        filename:'bundle.js'
    }
}
```

根目录下新建 `index.html`,内容如下

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
</head>
<body>
    <script src="bundle.js"></script>
</body>
</html>
```

命令行运行

```
webpack
```

就可以把 `src/index.js` 的内容打包输出到 `bundle.js` ，其他主要参数有：

```
webpack --config xxx.js #使用另一份配置文件来打包
webpack --watch  #监听变动并自动打包
webpack -p  #压缩混淆脚本
webpack -d #生成map映射文件，告知哪些模块被最终打包到哪里了
```

安装webpack-dev-server

在全局环境中安装`webpack-dev-server`

```
npm install webpack-dev-server -g
```

在项目根目录下执行命令：

```
webpack-dev-server
```

打开浏览器`http://localhost:8080/index.html`就可以看到输出了

安装Babel加载器

babel可以翻译es6到es5，使用下面命令安装babel-loader

```
npm install babel-loader babel-core babel-preset-es2015 --save-dev
```

配置webpack.config.js,添加babel-loader引导器,在module.exports添加module：

```
module.exports = {
    //页面入口文件配置
    entry:{
        index:'./src/index.js'
    },
    //编译输出配置
    output:{
        filename:'bundle.js'
    },
    module:{
        //模块加载器
        loaders:[{
            test:/\.js$/,
            loaders:['babel-loader?presets[]=es2015'],
            exclude:/node_modules/
        }]
    }
}
```

运行`webpack-dev-server`命令发现支持es6了

CSS加载器

在webpack里，CSS同样可以模块化，使用import导入，因此我们不再使用 link 标签来引用 CSS，而是通过 webpack 的 style-loader 及 css-loader。前者将 css 文件以 <style></style> 标签插入 <head></head> 头部，后者负责解读、加载 CSS 文件。

安装CSS相关的加载器

```
npm install style-loader css-loader --save-dev
```

配置webpack.config.js文件

```
module:{
    //模块加载器
    loaders:[{
        test:/\.js$/,
        loaders:['babel-loader?presets[]=es2015'],
        exclude:/node_modules/
    },{
        test:/\.css$/,
        loaders:['style','css']
    }]
},
```

autoprefixer

我们在写 CSS 时，按 CSS 规范写，构建时利用 autoprefixer 可以输出 -webkit、-moz 这样的浏览器前缀，webpack 同样是通过 loader 提供该功能。

安装autoprefixer-loader

```
npm install autoprefixer-loader --save-dev
```

配置webpack.config.js

```
module:{
    //模块加载器
    loaders:[{
        test:/\.js$/,
        loaders:['babel-loader?presets[]=es2015'],
        exclude:/node_modules/
    },{
        test:/\.css$/,
        loaders:['style-loader','css-loader','autoprefixer-loader']
    }]
},
```

重启webpack-dev-server，发现可以为css添加浏览器前缀了。

代码压缩

webpack自带了一个压缩插件UglifyJsPlugin,只需要在配置文件中引入即可。

```
plugins:[
    new webpack.optimize.UglifyJsPlugin({
        compress:{
            warnings:false
        }
    })
]
```

加入了这个插件后，编译的速度会明显变慢，所有一般只在生产环境中启用。

相关资料

> http://www.jianshu.com/p/b95bbcfc590d
>
> https://gold.xitu.io/entry/574fe7c579bc440052f6d805
>
> http://www.w2bc.com/Article/50764

gulp与webpack-stream集成配置

webpack非常强大,但是也有不足的地方,批量式处理依然是gulp更胜一筹.我们是否可以将两者的优点结合起来呢?这篇文章就是讲述如何集成gulp和webpack

安装

```
npm init #生成package.json
npm install --save-dev webpack-stream vinyl-named #vinyl-named用来保持输入和输出的文件名相同，否则会自动生成一个has
npm install --save-dev gulp #安装gulp
npm install babel-loader babel-core babel-preset-es2015 --save-dev #安装babel
touch gulpfile.js  #创建gulp配置文件
 touch webpack.config.js #创建webpack配置文件
```

配置gulpfile.js

```
var gulp = require('gulp');
var webpack = require('webpack-stream');
var named = require('vinyl-named');
var webpackConfig = require('./webpack.config.js');

//定义一个webpack任务
gulp.task('webpack',function(){
    return gulp.src('./src/index.js') //目标文件
        .pipe(named())  //vinyl-named用来保持输入和输出的文件名相同，否则会自动生成一个hash.
        .pipe(webpack(webpackConfig)) //调用webpack来处理流
        .pipe(gulp.dest('./dist/')) //处理完成后保存的目录
})

//定义一个默认任务
gulp.task('default',['webpack']); //定义一个默认gulp任务，让它运行webpack任务
```

配置webpack.config.js

```
module.exports = {
    watch:true,
    devtool:'source-map',
    resolve:{
        extensions:['','.js']
    },
    module:{
        loaders:[{
            test:/\.js$/,
            loader:'babel-loader',
            query:{
                presets:['es2015']
            }
        }]
    }
}
```

注意：用webpack-stream不需要配置entry和output

运行任务

在src目录下新建index.js主入口文件，内容如下

```
import {sum,square,variable,MyClass} from './import';
console.log(square(5));
var cred={
    name:'hello world',
    enrollmentNo:11115078
}
var x = new MyClass(cred);
console.log(x.getName());
```

index.js文件引入了import.js的文件，内容如下

```
var sum = (a, b = 6)=>(a + b);
var square = (b)=> {
    "use strict";
    return b * b;
};
var variable = 8;
class MyClass{
    constructor(credentials){
        this.name = credentials.name;
        this.enrollmentNo = credentials.enrollmentNo
    }
    getName(){
        return this.name;
    }
}
export {sum,square,variable, MyClass};
```

现在运行：

```
gulp
```

gulp已经使用webpack监测了index.js文件的改变，每当index.js有改变的时候，会重新打包生成dist/index.js

相关资料

> "
>
> https://my.oschina.net/wolfx/blog/673922

分类：javascrip　字数：1837

别人说我名字很长

程序员 济南

粉丝 38 | 博文 210 | 码字总数 69266

+ 关注

相关博客

下一代前端打包工具与tree-shaking

ouven

2545 5

构建web应用前端方面的一点点心得

慢头

1967 10

前端

zreoCode

73 0

评论 (5)

Ctrl+Enter 发表评论

notreami

1楼 2017/02/13 19:23

gulp目测快淘汰了，webpack目前正火，听闻rollup.js 号称下一代

554330833a

2楼 2017/02/14 08:55

一定要用nodejs的吗

NiceIve

3楼 2017/02/14 09:35

webpack最不爽的是多了很多额外得代码，但是看上面介绍说rollup没有这种问题，那么下一步就是试试rollup

伟子啊

4楼 2017/02/14 11:25

弱弱的问一句，nodejs有没有界面可以控制我每个组件的版本或者卸载的，每次都好多包 ，头都晕了

o轻扬o

5楼 2017/02/14 12:13

> 引用来自“伟子啊”的评论
> 弱弱的问一句，nodejs有没有界面可以控制我每个组件的版本或者卸载的，每次都好多包 ，头都晕了

package.json 😪