# 一 Babel

## 1 Babel的安装与使用

Babel是ES6、React等浏览器不支持的语言、框架的编译器。

第一步：项目根目录安装：

npm install -D babel-cli babel-preset-env

第二步：项目根目录下创建 .babelrc 文件，内容如下：

{ "presets": ["env"] }

第三步：编译src目录下所有文件到dist目录下 --out-dir可以简写为 -d

./node\_modules/.bin/babel src --out-dir dist

这样每次输入很长的babel目录、命令、命令参数很费事，可以直接通过npm脚本实现，在package.json文件的script中添加如下脚本：

**"build"**: **"babel src --watch -d dist"**

执行： npm run build #即可实现编译，--watch（放在别的命令之前），当目录有了变化，自动编译。

## 2 Babel插件

babel-cli只是一个执行babel命令行工具，本身不具备编译功能，编译功能是由插件babel-preset-env 提供的。env是最新的babel编译工具，包含了所有的ES\*功能，如果我们不需要这么多的新特性，可以有选择的安装编译插件：

# ES2015转码规则

$ npm install --save-dev babel-preset-es2015

# react转码规则

$ npm install --save-dev babel-preset-react

# ES7不同阶段语法提案的转码规则（共有4个阶段），选装一个

$ npm install --save-dev babel-preset-stage-0

$ npm install --save-dev babel-preset-stage-1

$ npm install --save-dev babel-preset-stage-2

$ npm install --save-dev babel-preset-stage-3

我们可以通过配置文件来让上述的编译工具一个、多个生效，新建配置文件.babelrc:

{

"presets": [

"es2015",

"react",

"stage-2"

],

"plugins": []

}

## 3 babel高级用法

见webpack文档内 webpack与babel介绍

# 二 工作流

在开发阶段，我们往往使用ES6、less、coffeescript来提升开发效率，但是这些文件不能直接部署在生产环境中，即在开发部署时，需要对css、js等文件执行编译、压缩等等一系列流程任务，我们称之为工作流。

人工处理这些任务的代价是很高的，利用构建工具可以编写一些任务，按照我们需要的流程来执行。

常见的处理任务包括：预处理语言的编译、代码压缩混淆、图片体积优化；

常见的构建工具包括：Grunt（已衰落）、Gulp、F.I.S（百度出品）、webpack(也能承担一部分工作流任务，webpack主要功能是打包)。

# 三 gulp

## 1 gulp使用

1 先全局安装： npm install gulp -g (需要全局安装原因：gulp是命令行工具)

2 再本地安装： npm install gulp -D (需要本地安装原因：gulp也是个工具包)

3 创建配置文件：gulpfile.js

const gulp = require('gulp');  
*//创建一个默认任务*gulp.task('default',function () {console.log("hello gulp");  
});

4 命令行执行： gulp default //由于default是默认任务，这里可以省略

## 2 任务

### 2.1 定义任务

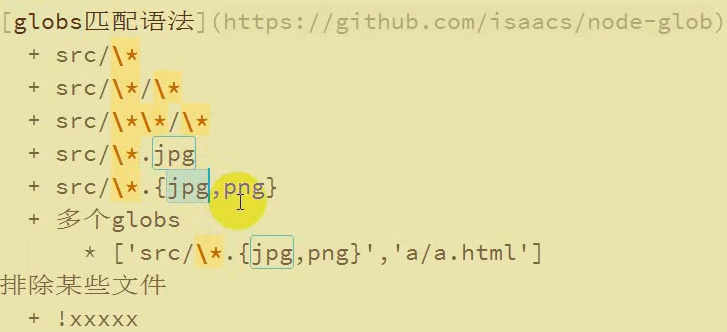
gulp.task('js',function () {  
 gulp.src('src/js/\*\*/\*.\*') *// \*\*代表递归文件夹* .pipe(gulp.dest('dist/'));  
});

gulp.src() 是获取需要构建资源的路径，也可以使用[]参数（正则也可以），!表示不匹配

.src(['src/js/\*\*/\*.\*','!src/demo.html'])

gulp.pipe() 管道，将需要构建的资源“输送”给插件。

gulp.dest() 构建任务完成后资源存放的路径（会自动创建）



### 2.2 watch

watch用于监视文件的改变，自动构建：

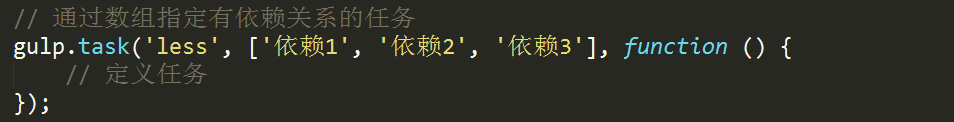
gulp.task('js',function () {

*//src下文件发生改变，自动执行default任务*  
 gulp.watch('src/\*',['default']);

});

### 2.3 任务依赖

不同任务间存在依赖关系时，可以指定依赖，如下图



## 3 Gulp工作原理

Gulp本身只有文件复制等基础API，通过不同的插件实现构建任务，Gulp只是按着配置文件调用执行了这些插件。



## 4 插件使用

### 1 使用步骤

比如需要编译less，需要先安装编译less的gulp插件：npm install gulp-less -S

const gulp = require('gulp');  
const less = require('gulp-less');  
gulp.task('less',function () {  
 gulp.src('src/\*\*/\*.less')  
 .pipe(less())  
 .pipe(gulp.dest('dist/'));  
});

### 2 gulp服务器插件

const gulp = require('gulp');

const connect = require('gulp-connect');

gulp.task('server',()=>{

connect.server({

root: 'src',

livereload: true

});

gulp.watch('src/\*\*/\*.\*',['reload']);

});

gulp.task('reload',()=>{

gulp.src('src/\*\*/\*.\*')

.pipe(connect.reload());

});

## 5 常用Gulp插件

gulp-less 编译LESS文件

gulp-cssmin 压缩CSS

gulp-rname 重命名

gulp-imagemin 图片压缩

gulp-uglify 压缩JS

gulp-concat 合并

gulp-htmlmin 压缩HTML

gulp-autoprefixer 添加CSS私有前缀

gulp-rev 添加版本号

gulp-rev-collector 内容替换

gulp-connect 创建服务器，默认监听8080端口

gulp-useref

gulp-if

## 6 缓存处理

浏览器在加载html网页时，网页内部的css，js，图片都会再次让浏览器发起请求，多次刷新html页面时，这些东西就会造成资源浪费，浏览器因此有了缓存系统，会将css等内容缓存下来。同时，也带来了弊病，一旦后台有修改，前天刷新很容易出现bug.

浏览器缓存机制：按资源路径和资源名缓存

使用index.css?v=时间戳” 命名css

这样书写就会造成每次出现的资源名都会不一样，就可以解决缓存问题。

但是仍然有不如意的地方：如果时间戳每天变一次，那么在统一内，后台修改了2次css，那么这时候用户仍然无法体验到修改。。

gulp-rev 添加版本号 可以让css文件路径名进行改变

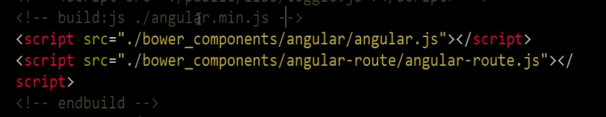


以下能实现：修改css名后，并修改html中的lin引用名

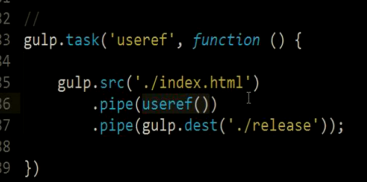


## 7 gulp-useref

当文件中有以下标记时：



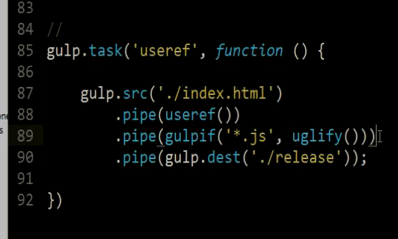
插件处理代码：



经过这个插件处理，压缩后直接变为：



## 8 gulp-if



代码的意思：

这个代码将会对被标记的所有代码进行userref合并，但是如果有js代码。还会进行丑化处理