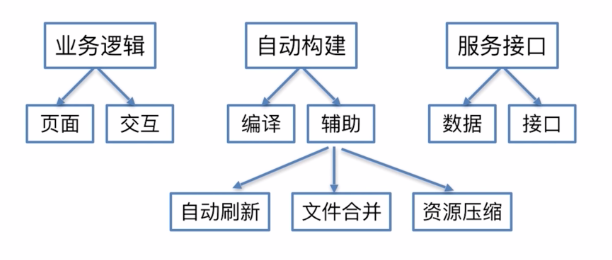
# 项目构建

## 2.1 项目构建

### 1 基础架构

了解前端架构思想：放哪些目录 哪些目录放置哪些文件 为什么这么放 它的合理性在哪

架构目录



编译： 把es6编译成es5 或es3

辅助-自动刷新：

资源压缩： css压缩 js压缩 图片转base64编码

### 2 任务自动化（gulp）

**什么是任务自动化**

减少人工操作，让机器自动监听所有的操作，自动相应，提高效率

文件合并 项目依赖 模块化 文件压缩

**什么是gulp**(基于nodejs)

Grunt 级联操作 不是文件流操作 操作慢

http://www.gulpjs.com.cn/

Gulp的作用

了解如何使用gulp完成任务自动化

### 3 编译工具(babel,webpack)

**什么是babel、webpack**

babel 专门编译es6

编译工具

webpack 处理模块化依赖

babel的核心用法

了解webpack及webpack-stream的作用

### 4 代码实现

## 2.2项目目录创建

|  |
| --- |
| mkdir app #创建app文件夹  type NUL > test.js #创建test.js空文件  echo.>index.js #创建index.js空文件  echo indexjs > index.js #创建含有indexjs内容的index.js文件  tree 或者dir 可查看文件夹目录  dir / b #将目录内的内容以清单的形式列出  dir /b/ad/s/b #可列出当前目录下的所有文件夹及子文件夹  cd ../ #回退上级目录 |

express 是ejs模板引擎 依赖nodejs

**安装express:**

|  |
| --- |
| # 全局安装express  npm install express-generator -g |

express -e .

# express 脚手架工具

# -e ejs模板引擎

# . 是在当前目录执行

|  |
| --- |
| #在当前目录下运行express脚手架工具创建ejs模板引擎  express -e .    npm install    121312 |

App --- 放置前端代码

Css --- css文件

Js -- js文件

Class -- 类文件

Test.js

Index.js --- 入口文件

Views --- 模板

Error.ejs --- 错误模板

Index.ejs -- 入口模板文件

Server ---- 服务器目录

tasks --- 构建工具目录(创建很多任务的js， 比如说文件的合并、 脚本的编译、 模板的自动更新等等，在这些命令当中有一个是需要处理命令行参数 也就是说我们输入一段命令行工具 这个命令行工具它最后执行的东西要做解析)

Util

args.js

package.js 安装依赖包所需

Package.js可以手动创建也可以自动创建

npm init 自动创建package.js

.babelrc 设置babel编译工具的文件 文件名不可更改（固定的） babel编译的时候自动会找这个文件

创建gulp

gulpfile.babel.js # 使用.babel是因为接下来写的构建脚本都是使用es6的语法，如果不加.babel会报错

### 项目目录

tree /f /a >filelist.md 导出生成的文件目录(保存在当前所在文件夹目录下）

|  |
| --- |
| # 目录列表  | .babelrc                // 设置babel编译工具的文件  |                         // babel编译的时候自动会找这个文件,文件名不可更改（固定的）  | gulpfile.babel.js // 使用.babel是因为接下来写的构建脚本都是使用es6的语法，  | // 如果不加.babel会报错  | package.json // 安装依赖包所需 npm init可自动创建  |  +---app // 放置前端代码  | +---css // css  | +---js // js  | | \---class // 类文件  | | index.js // 入口文件  | | test.js  | |  | \---views // 模板  | error.ejs // 错误模板  | index.ejs // 入口模板文件  |  +---server // 服务器目录  | | app.js  | | package.json  | |  | +---bin  | | www  | |  | +---node\_modules  | |  | +---public  | | +---images  | | +---javascripts  | | \---stylesheets  | | style.css  | |  | +---routes  | | index.js  | | users.js  | |  | \---views  | error.ejs  | index.ejs  |  \---tasks // 构建工具目录  |  \---util  args.js |

## 2.3命令行处理，创建JS编译任务脚本

### 创建命令行参数

tasks/util/args.js

### 创建构建脚本 对js做处理

tasks/scripts.js

**安装依赖包**

npm install gulp gulp-if gulp-concat webpack webpack-stream vinyl-named gulp-livereload gulp-plumber gulp-rename gulp-uglify gulp-util yargs --save-dev

--save-dev 表示文件除了安装以外，还要在package.json文件中创建一个安装包依赖的字段

**创建gulp任务**

## 2.4 创建模板、服务任务脚本

### 创建模板脚本

cd tasks

echo.>pages.js

### 创建css脚本

cd tasks

echo.>css.js

### 创建服务器脚本

cd tasks

echo.>server.js

## 2.5 文件自动监听，项目构建测试

### 文件自动监听

cd tasks

echo.>browser.js

### 创建clean任务

cd tasks

echo.>clean.js

### 安装依赖包

npm install gulp-live-server del --save-dev

echo.>clean.js

### 把所有的任务串起来

cd tasks

echo.>build.js

### default 默认启动项

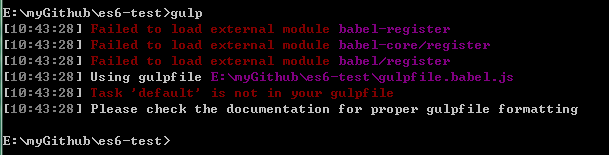
cd tasks

echo.>default .js

### 全局安装gulp

npm install --global gulp

**执行gulp**



执行gulp时第一个找的是gulpfile.babel.js文件

### 安装babel-loader依赖包

npm install babel-loader babel-core babel-preset-env --save-dev

babel-loader依赖babel-core

### 编辑gulpfile.babel.js

|  |
| --- |
| // 把task目录下的所有文件加进来 让它去执行  import requireDir from 'require-dir'  requireDir('./tasks'); |

安装require-dir

再执行gulp命令试试

|  |
| --- |
|  |

这是因为.babelrc 还没配置

配置.babelrc

|  |
| --- |
| {  *"presets"*: ["es2015"]  } |

安装依赖包

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

运行gulp

|  |
| --- |
|  |

### 热更新配置

为什么没有自动刷新？

> ### 热更新配置

// 在server/app.js中添加如下代码:

// [必须放在app.use(express.static(path.join(\_\_dirname, 'public')));后面]

app.use(require('connect-livereload')())

// 安装connect-livereload

npm install connect-livereload --save-dev

# es6语法

## 3.1 let,const命令

### let 和const

参考[阮一峰 ES6入门 let和const命令](http://es6.ruanyifeng.com/docs/let) | http://es6.ruanyifeng.com/#docs/let

**作用域的概念**

Es6之前有2个作用域：全局作用域、函数作用域

Es6新增一个作用域：块作用域

怎么区分块作用域？

Ps: Es6强制开启严格模式

使用let 不能重复定义

**如何使用let和const**

const声明的常量不能修改

但是const声明的是对象的话，对象是可以改变的

[对象是引用类型，最后返回值是对象存储的内存中的指针，指针是不变的，对象本身是可以变的]

(详见 [es6 const 本质](http://es6.ruanyifeng.com/#docs/let" \l "本质) | http://es6.ruanyifeng.com/#docs/let#本质)

const与let在块作用域上使用是相同的

const 声明的时候必须赋值

## 3.2 解构赋值

什么是解构赋值

参考 <http://es6.ruanyifeng.com/#docs/destructuring>

### 解构赋值的分类

|  |  |  |
| --- | --- | --- |
| 数组结构赋值 | 对象结构赋值 | 字符串结构赋值 |
| 布尔值解构赋值 | 函数参数解构赋值 | 数值解构赋值 |

## 3.3 正则扩展

参考 <http://es6.ruanyifeng.com/#docs/regex>

### 正则新增特性

|  |  |  |
| --- | --- | --- |
| 构造函数的变化 | 正则方法的扩展 | u修饰符 |
| y修饰符 | s修饰符 |  |

3.4 字符串扩展(上)

3.5 字符串扩展(下)

3.6 数值扩展

3.7数组扩展

3.8 函数扩展

3.9对象扩展

3.10 Symbol用法

3.11 set-map 数据结构

3.12 map-set与数组和对象的比较

3.13 Proxy和Reflect

3.14 类与对象

3.15 Promise

3-16 Iterator

3-17 Generator

3-18 Decorators

3-19 Module模块化

1. 项目实战
2. 课程总结