[第一章：概念介绍 3](#_Toc510945303)

[（1）理论与背景 3](#_Toc510945304)

[什么是构建？为什么要自动化构建 3](#_Toc510945305)

[构建工具的演进之路 3](#_Toc510945306)

[前端构建工具介绍 3](#_Toc510945307)

[（2）JavaScript的构建 3](#_Toc510945308)

[什么是Browserify ? 3](#_Toc510945309)

[使用Browserify加载第三方类库 3](#_Toc510945310)

[使用Browserify实现JS的模块化加载 3](#_Toc510945311)

[使用coffeescipt 和 ES6来优化代码 3](#_Toc510945312)

[（3）css的构建 3](#_Toc510945313)

[gulp-clean-css的介绍和使用 3](#_Toc510945314)

[SASS来帮忙 3](#_Toc510945315)

[第二章 Javascript的构建(browserify) 3](#_Toc510945316)

[（1）什么是Browserify ? 3](#_Toc510945317)

[1.1 官网：http://broswerify.org/ 3](#_Toc510945318)

[1.2 Broswerify 是一个管理前端依赖的工具。可以实现JS的模块化加载 3](#_Toc510945319)

[（2）Browserify实现JS的模块化加载 3](#_Toc510945320)

[1.1 Browserify使用了node.js的CommonJS模块规范来实现JS的模块化加载 3](#_Toc510945321)

[1.2 什么是CommonJS ？ 3](#_Toc510945322)

[1.2.1 CommonJS 简单来说是使用module.exports来定义JS模块；使用require语句来加载JS模块。 3](#_Toc510945323)

[第三章 使用gulp进行构建 4](#_Toc510945324)

[（1）Gulp 的基本用法 4](#_Toc510945325)

[1.3 npm install gulp 安装gulp 4](#_Toc510945326)

[1.4 新建 gulpfile.js文件 4](#_Toc510945327)

[1.5 利用gulp的api task 定义任务。 Gulp中默认的任务是 default 。 4](#_Toc510945328)

[（2）Gulp中使用Browserify 4](#_Toc510945330)

[1.3 Browserify可以作为可执行的文件在shelljs中调用；其本身也有专门的api可以作为node模块来调用。 4](#_Toc510945331)

[1.3.1 通过**shelljs** 调用： 4](#_Toc510945332)

[1.3.1.1 安装: npm install shelljs ，然后在gulpfile中引用。 4](#_Toc510945333)

[1.3.2 通过**node**调用： 4](#_Toc510945335)

[1.3.2.1 安装browserify： npm install browserify （之前-g是全局安装作为全局的可执行文件） 4](#_Toc510945336)

[1.3.2.2 browserify().add('js/index.js').bundle().pipe(fs.createWriteStream('js/main.js')) 4](#_Toc510945337)

gulp慕课网-前端内容的自动化构建

第一章：概念介绍

# （1）理论与背景

* 1. 什么是构建？为什么要自动化构建
  2. 构建工具的演进之路
  3. 前端构建工具介绍

# （2）JavaScript的构建

* 1. 什么是Browserify ?
  2. 使用Browserify加载第三方类库
  3. 使用Browserify实现JS的模块化加载
  4. 使用coffeescipt 和 ES6来优化代码

# （3）css的构建

* 1. gulp-clean-css的介绍和使用
  2. SASS来帮忙

第二章 Javascript的构建(browserify)

# （1）什么是Browserify ?

* 1. 官网：<http://broswerify.org/>
  2. Broswerify 是一个管理前端依赖的工具。可以实现JS的模块化加载

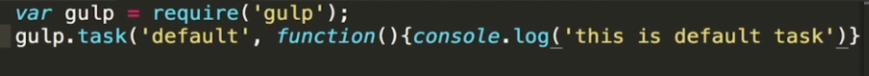
# （2）Browserify实现JS的模块化加载

* 1. Browserify使用了node.js的CommonJS模块规范来实现JS的模块化加载
  2. 什么是CommonJS ？
     1. CommonJS 简单来说是使用module.exports来定义JS模块；使用require语句来加载JS模块。

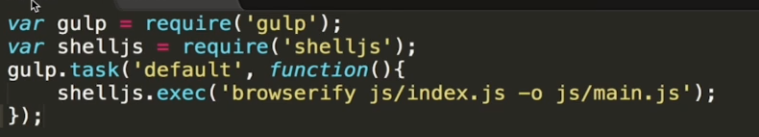
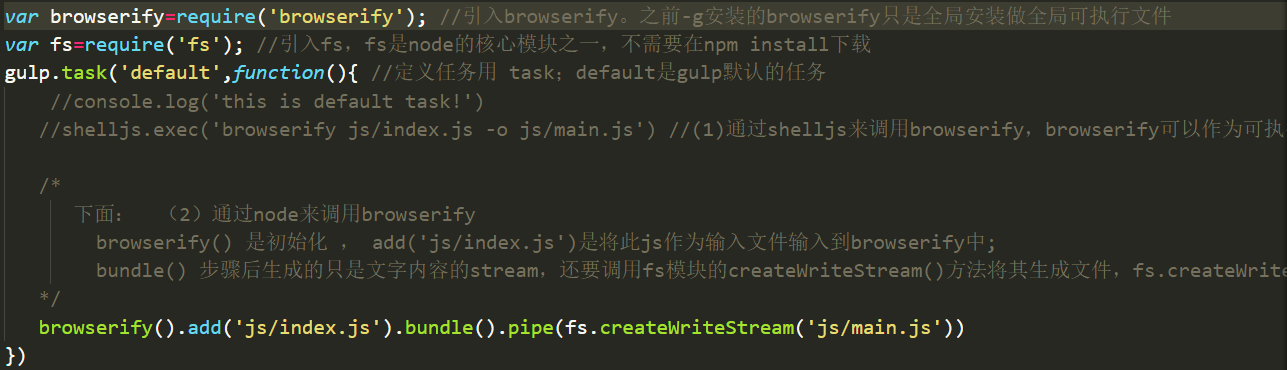
第三章 使用gulp进行构建

# （1）Gulp 的基本用法

* 1. npm install gulp 安装gulp
  2. 新建 gulpfile.js文件
  3. 利用gulp的api task 定义任务。 Gulp中默认的任务是 default 。



# （2）Gulp中使用Browserify

* 1. Browserify可以作为可执行的文件在shelljs中调用；其本身也有专门的api可以作为node模块来调用。
     1. 通过**shelljs** 调用：
        1. 安装: npm install shelljs ，然后在gulpfile中引用。
     2. 通过**node**调用：
        1. 安装browserify： npm install browserify （之前-g是全局安装作为全局的可执行文件）
        2. browserify().add('js/index.js').bundle().pipe(fs.createWriteStream('js/main.js'))

# （3）自动化Javascript构建

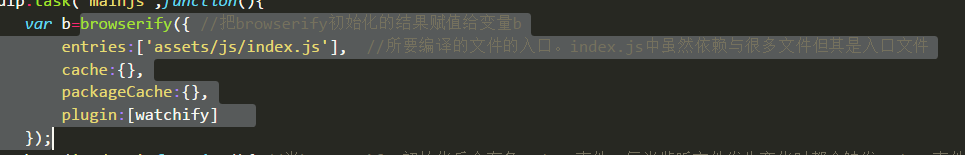
* 1. **gulp.watch**：
     1. gulp.watch 是用来检测文件的修改，一旦文件发生变动，gulp.watch就可以触发我们之前定义好的任何任务。
  2. **在构建过程中需要将源文件和目标生成文件分开，故新建assest 文件夹用来存放 add-todo-controller.js / todo-controller.js / todos-controller.js /index.js 等各种资源文件**
  3. 操作步骤：
     1. 建立assest 文件夹用来存放相关资源文件。
     2. npm install run-sequence 并在gulpfile 中引入。
     3. 定义mainjs任务来调用browserify()方法构建js。
     4. 定义watch 任务并采用gulp.watch来监听assets/js 下所有js文件，若发生改变则通过sequence() 执行mainjs任务
     5. 在default 任务中使用sequence() 来调用mainjs以及watch 任务。



# （3-2）自动化Javascript构建-2

* 1. **使用Gulp的watchify 插件来构建**：
     1. Gulp的watchify插件：<http://github.com/substack/watchify>，这个插件也是用于监听文件的变化以便用于目标文件的重新编译。其却是可以理解js文件中的依赖关系的，即我们的 require() 语句。我们只需要告诉watchify browserify编译所需要的入口文件,watchify 会清空所有依赖的变化并重新进行编译。同时它采用缓存机制，每次只找寻有变化的文件，可以提高重编译速度。

**\*\*\* Watchify 可自动执行但不会在命令行中显示任何输出**

* + 1. 使用方法：<http://github.com/substack/watchify> 中method 。
       1. 安装watchify 插件。 npm install watchify 并在gulpfile中引用。
       2. 初始化browserify 并将初始化结果赋值给变量b 。entries:[‘assets/js/index.js’] 是所要编译文件的入口，index.js中虽然依赖与很多文件但其是入口文件,watchify会去解析其所需的依赖
       3. 定义update事件 并 调用bundle() 方法，当监听的文件发生变化时会触此事件。
       4. 调用bundle( )方法
       5. default 任务中 sequence( )就不需要调用watch 任务了，也不需要在gulpfile 中定义watch 任务。

第四章 使用Browserify加载第三方类库

# （1）为什么要用browserity来管理第三方类库?

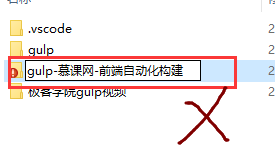
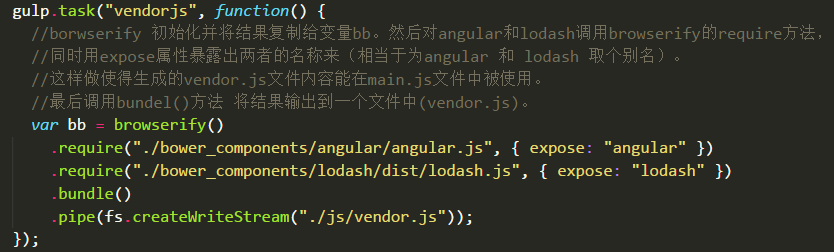
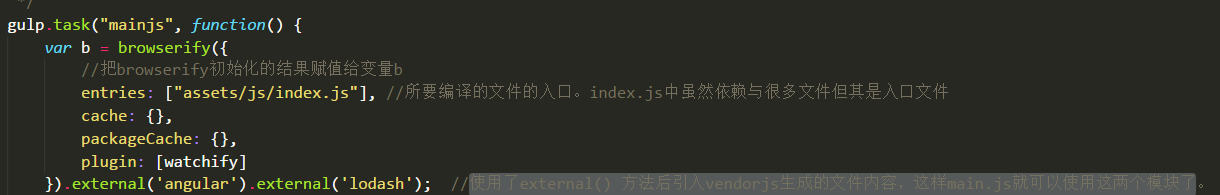
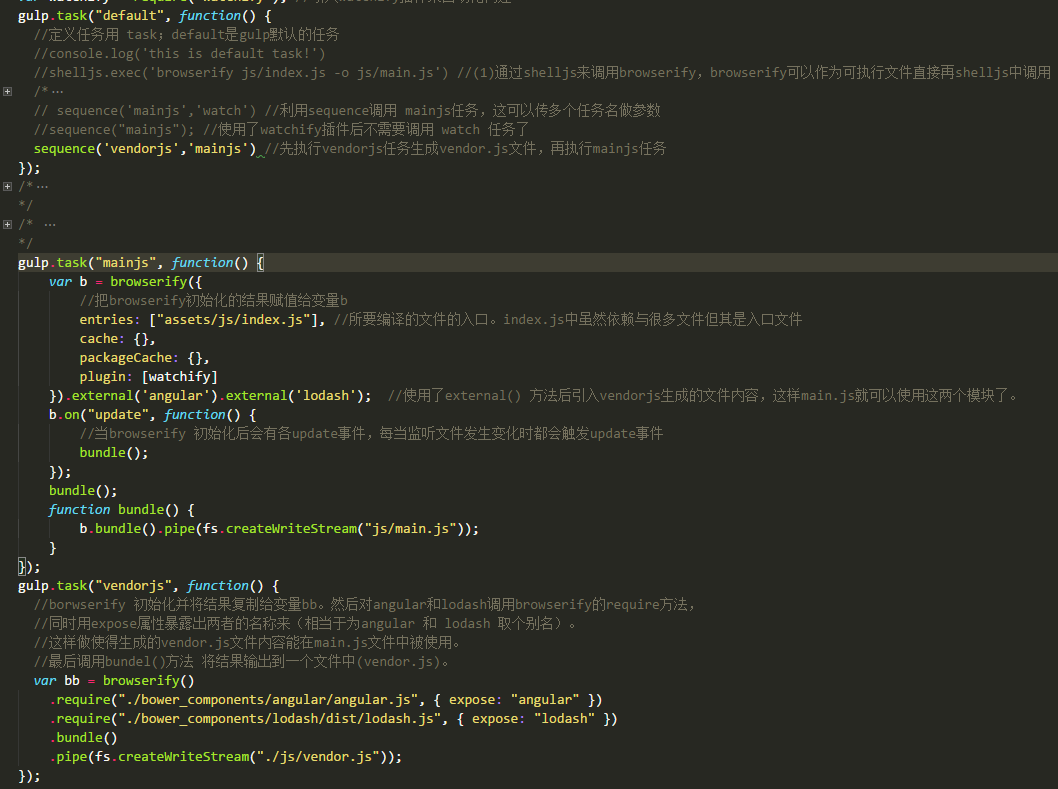
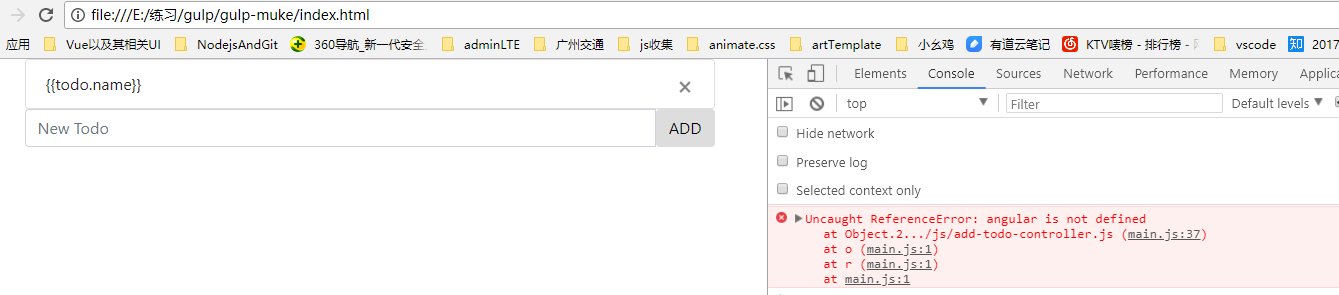
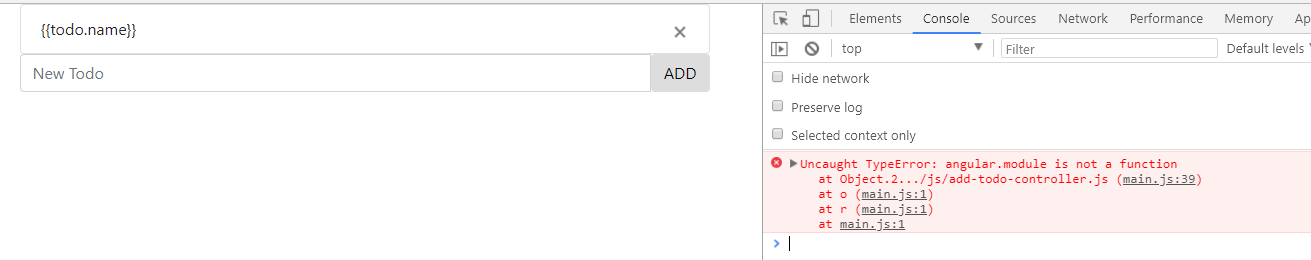
* 1. 通过browserify 打包第三方类库（框架）成一个文件那么可以减少html页面的复杂度**（简单来说是不用写很多script标签来引入多个js文件 ）**，方便管理。
  2. 在减少html页面复杂度同时可以减少js文件的数量，从而减少页面在客户端打开时所需要发送的http请求的数量。
  3. 可以在构建js代码的过程中加入特殊的处理，例如使用Uglify 是代码 “丑化”（简化变量命名和删除空行空格）来使代码难以阅读。一方面可以保护源码 ，另一方面可以减少文件体积，提升用户体验。

**\*\*\*\*\* 在线Uglify :**[**https://skalman.github.io/UglifyJS-online/**](https://skalman.github.io/UglifyJS-online/)

# （2）Browserify来构建第三方类库

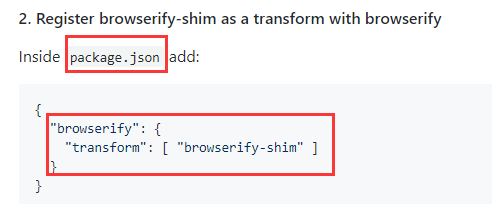
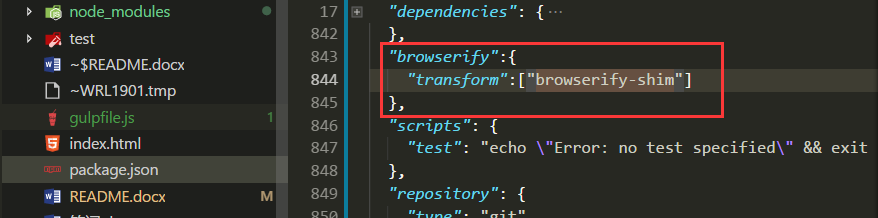
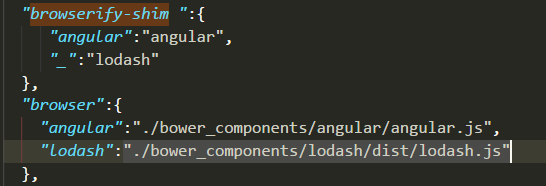
## \*\*\* browserify-shim 是构建第三方类库的常用工具

## \*\*\* bower 是node上用来管理第三方类库(框架) 的工具。

* 1. **第一种： 不使用browserify-shim 构建第三方类库**
     1. 安装 bower 工具来管理第三方模块 ： npm install bower 。 使用bower的优势在于可以升级所下载的类库。 通过bower 下载后会生成**bower\_components** 文件夹来保存下载的文件。
     2. **bower install angular** 命令 可以下载angular
     3. **bower update angular** 命令即可升级angular 的版本。其它类库升级同理。
     4. **\*\*\*\* 利用 bowser 下载更新类库，项目名 等的名称不能中文 也 不能大写。**
     5. gulpfile 中新建 **vendorjs** 任务，(在前端开发中有个约定俗成的规矩 就是将打包的第三方类库（框架）文件叫做 **vendorjs** ) 。
     6. **接下来生成一个vendor.js 文件，来被main.js 引用, vendorjs 文件的内容是打包后的第三方类库(此项目中用到的是angular 和 lodash)代码，所以需要用到borserify 的require 和 external 两个方法。**[http://github.com/substack/node-borwserify#usage](http://github.com/substack/node-borwserify%23usage)。
        1. **Browserify 的require 方法 作用是使一个模块在生成的文件之外被引用。**(参照这里也即是说生成的vendor.js其要在main.js 中被使用)。
        2. **Browserify 的external 方法作用是 引用另一个包中的文件**
     7. **然后在mainjs任务中使用browserify的external 方法，使用了此方法后则可以引入vendorjs 生成的文件内容。这样main.js就可以使用这两个模块(angular 和 lodash)了。**
     8. **然后修改default 任务执行顺序，先执行vendorjs 后执行mianjs**
     9. 上述操作后 浏览器会报错误，

原因是我们使用了browserify后所有的模块都需要通过**require()** 来引入。而且我们引入的模块必须是采用CommonJS标出来定义的模块。即：

但是bowser 管理的很多类库并没有采用CommonJS形式，对我们构建第三方类库带来很多麻烦所以 推荐使用 [**browserify-shim**](https://github.com/thlorenz/browsrify-shim)

* 1. **第二种： 使用browserify-shim 构建第三方类库**
     1. 安装： npm install -D browserify-shim
     2. 将 browserify-shim 注册为browserify的转换。 按照其官网 api 要求在package.json 中添加配置。其配置语句意思是在browserigy 过程中添加一个transform操作，用browserify-shim来转换。
     3. 提供browserify-shim 的配置。您可以通过[browser字段](https://gist.github.com/defunctzombie/4339901#replace-specific-files---advanced)以不同的名称显示文件，并在shim config中的别名下引用它们：
     4. 上述步骤完成后可以**将 vendorjs 集成到mainjs中**，这也是比较常用的做法。 **这样在gulpfile 中就不需要 定义 vendorjs 任务了。mainjs任务中也不需要 external( ) 两个模块了 。default 中也不需要执行vendorjs 任务。同时index.html 中也不需要加载 vendor.js 文件**