# 第七周 前端快速开发工具

# 第一章 图表库和模板引擎

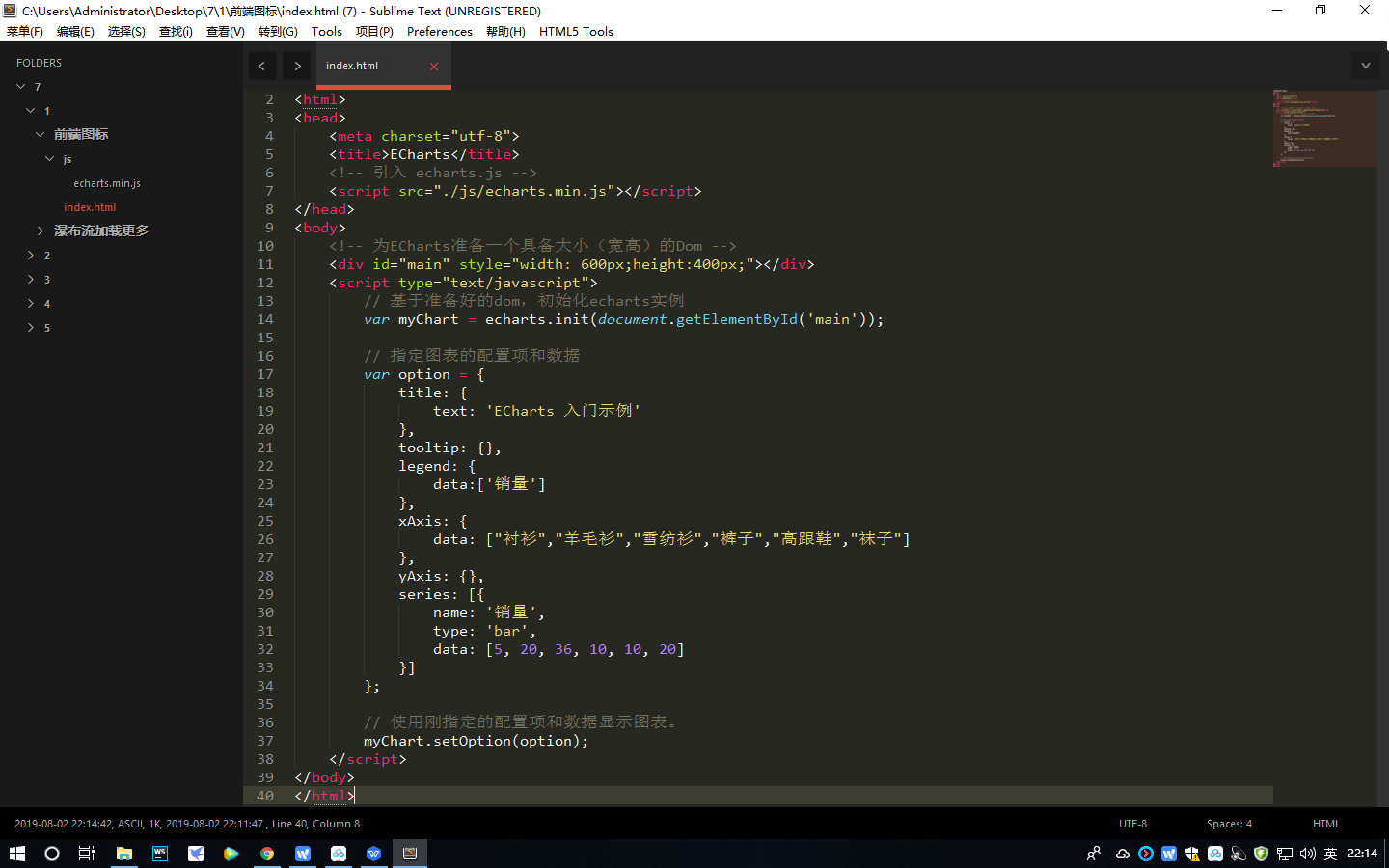
## 一、ECharts

使用 JavaScript 实现的开源可视化库，兼容当前绝大部分浏览器（IE8/9/10/11，Chrome，Firefox，Safari等），提供直观，交互丰富，可高度个性化定制的数据可视化图表。

使用步骤：

1. 引入ECharts
2. 为ECharts准备一个具备大小（宽高）的DOM
3. 基于准备好的dom，初始化echarts实例
4. 指定图表的配置项和数据
5. 使用刚指定的配置项和数据显示图表

【练习:柱状图】



【练习:饼状图】

## 二、模板引擎

## 1原理剖析

其本质是利用正则表达式，替换模板当中预先定义好的标签。

## 2流行模板引擎

**BaiduTemplate**：http://tangram.baidu.com/BaiduTemplate/

**ArtTemplate：https://github.com/aui/artTemplate**

**velocity.js：**https://github.com/shepherdwind/velocity.js/

**Handlebars：**http://handlebarsjs.com/

## 3 artTemplate

参考文档：https://aui.github.io/art-template/zh-cn/docs/installation.html

1. 引入template-web.js。

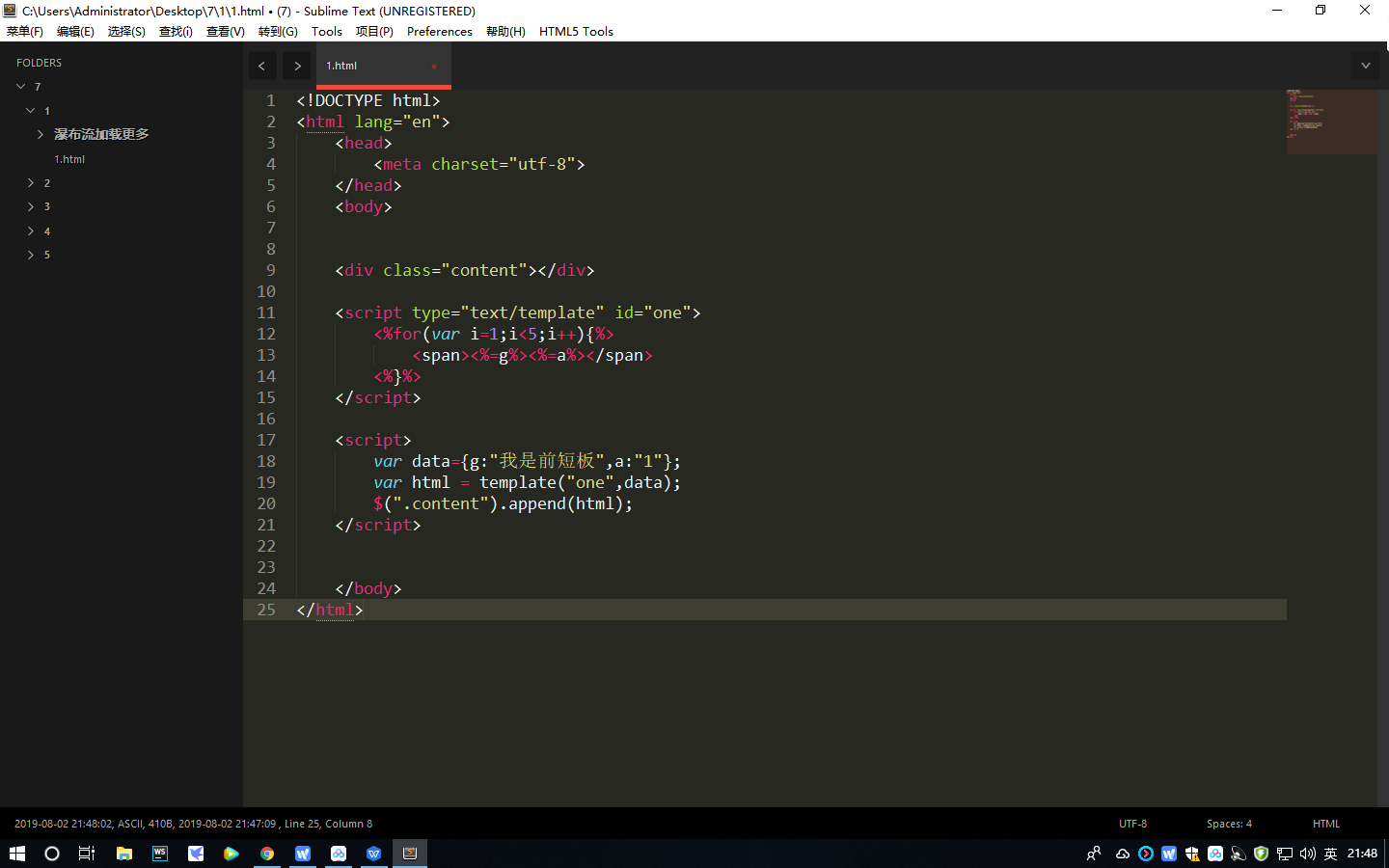
2、art-template 同时支持原始语法和标准语法。

2.1原始语法：

<% 与  %> 符号包裹起来的语句则为模板的逻辑表达式

<%= content %>为输出表达式

<%- content %>为原文输出



2.2 标准语法

{{value}} 输出

{{@ value }} 原文输出

{{if v1}} ... {{else if v2}} ... {{/if}} 条件判断

{{each target}} {{$index}} {{$value}} {{/each}} 循环

# 【案例讲解：瀑布流加载更多】

# Node服务器和路由

## 一、简介 （背景故事，了解即可）

众所周知，在Netscape设计出JavaScript后的短短几个月，JavaScript事实上已经是前端开发的唯一标准。

后来，微软通过IE击败了Netscape后一统桌面，浏览器毫无进步。微软认为IE6浏览器已经非常完善，几乎没有可改进之处，然后解散了IE6开发团队！而Google却认为支持现代Web应用的新一代浏览器才刚刚起步，尤其是浏览器负责运行JavaScript的引擎性能还可提升10倍。

先是Mozilla借助已壮烈牺牲的Netscape遗产在2002年推出了Firefox浏览器，紧接着Apple于2003年在开源的KHTML浏览器的基础上推出了WebKit内核的Safari浏览器，不过仅限于Mac平台。

随后，Google也开始创建自家的浏览器。他们也看中了WebKit内核，于是基于WebKit内核推出了Chrome浏览器。

Chrome浏览器是跨Windows和Mac平台的，并且，**Google认为要运行现代Web应用，浏览器必须有一个性能非常强劲的JavaScript引擎，于是Google自己开发了一个高性能JavaScript引擎，名字叫V8**，以BSD许可证开源。

浏览器大战和Node有何关系？

话说有个叫Ryan Dahl的歪果仁，他的工作是用C/C++写高性能Web服务。对于高性能，异步IO、事件驱动是基本原则，但是用C/C++写就太痛苦了。于是这位仁兄开始设想用高级语言开发Web服务。他评估了很多种高级语言，发现很多语言虽然同时提供了同步IO和异步IO，但是开发人员一旦用了同步IO，他们就再也懒得写异步IO了，所以，最终，Ryan瞄向了JavaScript。

因为JavaScript是单线程执行，根本不能进行同步IO操作，所以，JavaScript的这一“缺陷”导致了它只能使用异步IO。

选定了开发语言，还要有运行时引擎。这位仁兄曾考虑过自己写一个，不过明智地放弃了，因为V8就是开源的JavaScript引擎。让Google投资去优化V8，咱只负责改造一下拿来用，还不用付钱，这个买卖很划算。

于是在**2009年，Ryan正式推出了基于JavaScript语言和V8引擎的开源Web服务器项目，命名为Node.js。虽然名字很土，但是，Node第一次把JavaScript带入到后端服务器开发，加上世界上已经有无数的JavaScript开发人员，所以Node一下子就火了起来**。

#### 1 node的安装

#### 2 npm

npm是随同NodeJS一起安装的包管理工具，能解决NodeJS代码部署上的很多问题，常见的使用场景有以下几种：

* 允许用户从NPM服务器下载别人编写的第三方包到本地使用。
* 允许用户从NPM服务器下载并安装别人编写的命令行程序到本地使用。
* 允许用户将自己编写的包或命令行程序上传到NPM服务器供别人使用。

由于新版的nodejs已经集成了npm，所以之前npm也一并安装好了。同样可以通过输入 "**npm -v**" 来测试是否成功安装。

#### 2.1、安装模块

npm install 模块名@版本号

**注意：关于版本号**

版本格式：主版本号.次版本号.修订号，版本号递增规则如下：

* 主版本号：当你做了不兼容的API修改
* 次版本号：当你做了向下兼容的功能性新增
* 修订号：当你做了向下兼容的问题修正

例如："benchmark": "^1.0.0" :便是兼容于1.0.0版本，即大于等于1.0.0，小于2.x.x版本

* 安装时如果不写版本号表示安装最新版本
* 只写主版本号表示安装主版本号下下的最新版本

**注意：关于全局安装与本地安装**

* npm install express # 本地安装
* npm install express -g     # 全局安装

### 本地安装

* 将安装包放在 ./node\_modules 下（运行 npm 命令时所在的目录），如果没有 node\_modules 目录，会在当前执行 npm 命令的目录下生成 node\_modules 目录。
* 可以通过 require() 来引入本地安装的包。

### 全局安装

* 将安装包放在 /usr/local 下或者你 node 的安装目录。
* 可以直接在命令行里使用。
* 你可以使用以下命令来查看所有全局安装的模块：npm ls -g

#### 2.2依赖处理

**我的项目：my-project**

VUE2.0

Bootstrap3.3.7

jQuery1.12.4

如上图所示，一个项目依赖于多个外部的库（模块），并且模块和模块之间也有依赖关系，如果我在自己以此下载，并根据依赖关系引入文件，难度较大，可以使用package.json来记录相关项目这方面你的相关配置。

package.json是一个很重要的文件，用于管理模块的通用脚本、依赖等核心的配置数据,创建package.json文件，通过命令行：npm init或者手动书写.

package.json 位于模块的目录下，用于定义项目的属性。

* name - 本项目名称
* version - 项目的版本号
* description - 项目的描述
* homepage - 项目的官网 url
* author - 项目的作者姓名
* dependencies - 依赖包列表。如果依赖包没有安装，npm 会自动将依赖包安装在 node\_module 目录下
* repository - 包代码存放的地方的类型，可以是 git 或 svn，git 可在 Github 上
* main - 模块的入口文件
* scripts - 配置之后可以使用npm [命令名] 来执行对应的命令
* devDependencies - 运行依赖的模块
* keywords - 关键字

#### 2.3 卸载模块

我们可以使用以下命令来卸载模块:**npm uninstall 模块名**

#### 2.4 国内镜像

npm全称Node Package Manager，是Node.js的模块依赖管理工具。由于npm的源在国外，所以国内用户使用起来各种不方便。下面整理出了一部分国内优秀的npm镜像资源，国内用户可以选择使用。

**淘宝npm镜像**

* 搜索地址：http://npm.taobao.org/
* registry地址：**http://registry.npm.taobao.org/**

**cnpmjs镜像**

* 搜索地址：http://cnpmjs.org/
* registry地址：http://r.cnpmjs.org/

具体如何方法

1）临时使用

npm --registry https://registry.npm.taobao.org install jquery

  2）持久使用

npm config set registry https://registry.npm.taobao.org

// 配置后可通过下面方式来验证是否成功

npm config get registry

// 或

npm info jquery

## 二 nodejs模块导入和导出

#### 1、先用编辑器编写一个first.js 文件：

'use strict';

console.log('Hello, world.');

#### 2、然后在js文件所在文件夹中运行命令行：

node first.js

#### 3 NODE中的模块

在NODEJS中有两种模块：

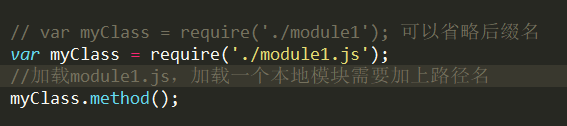
* 第一种是核心模块：node.exe文件中已经编译好了的模块，我们称之为核心模块，它是随着node.exe而发布的，核心模块的优势：执行高效和节省成本
* 第二种是文件模块：自己根据业务需求来编写的 .js文件，我们称之为文件模块

node有一套内置的模块系统，对象、方法以及变量可从一个文件exports出来用在其他地方。

【案例1 导出模块：使用module.exports 】



【案例2 导入模块：使用rerquire加载模块 】



**提问1：如果加载本地模块的时候不加路径名会怎么样？**

node将试图在$NODE\_PATH中寻找符合的模块，然后是./node\_modules、$HOME/.node\_modules、$HOME/.node\_libraries，或者$PREFIX/lib/node 。

**提问2：文件后缀省略node如何查找？**

node会找以.js、.json，以及以.node为后缀的文件。

## 三 常用模块

#### 1、常用全局变量：

\_\_dirname //当前脚本文件所在目录的绝对路径

\_\_filename //当前脚本文件的绝对路径

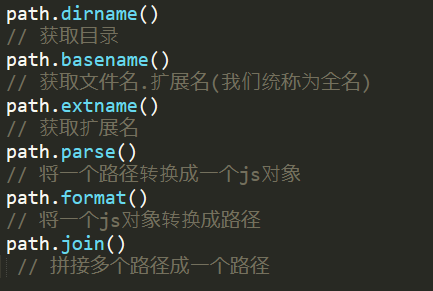
#### 2、path模块

想要在.js中使用path模块，一定要先引入:

var path = require('path');

path：表示路径的意思， 比如: c:\dd\cc

常用api演示：



#### 3、url模块

想要在.js中使用url模块，一定要

var url = require('url');

常用api演示：

url.parse()//将一个url转换成js对象

url.format()//将一个urlobj对象转换成url字符串

#### 4、querystring模块

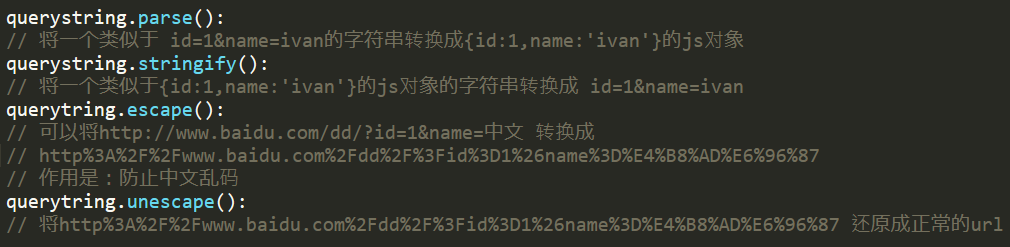
想要在.js中使用querystring模块，一定要

var querystring = require('querystring');

主要是用来解析一个url中的参数值的

例如：解析出 https://github.com/?type=delete&id=1&dd=2 中的 type = delete id =1 dd =2

常用api演示：



#### 5、FS模块

## 【课堂案例：简单文件拷贝】



## 【作业：当文件有变化是自动拷贝文件(可以利用上面写好的copy模块)】



#### 6 HTTP模块

nodemon : 可以实现当js文件代码有改变的时候，自动重启node 环境

  使用步骤：

  1、 npm install -g nodemon

  2、在cmd中使用 nodemon xx.js

**【案例：利用http开启web服务】**



#### 7 路由实现



# Gulp

## 一 、模块化开发

#### 1 模块概要

JavaScript在早期的设计中就没有模块、包、类的概念，开发者需要模拟出类似的功能，来隔离、组织复杂的JavaScript代码，我们称为模块化。

模块就是一个实现特定功能的文件，有了模块我们就可以更方便的使用别人的代码，要用什么功能就加载什么模块。

模块化开发的四点好处：

　　（1）避免变量污染，命名冲突

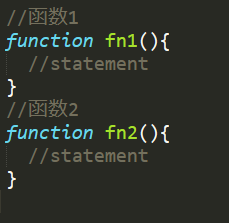
　　（2）提高代码复用率

　　（3）提高了可维护性

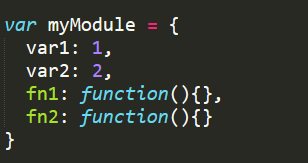
　　（4）方便依赖关系管理

为了避免缺少模块带来的问题，我们可以看看程序员应对的历程：

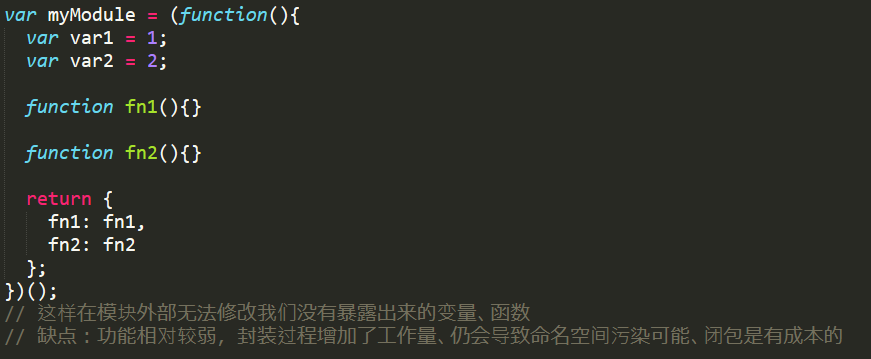
* 函数封装



* 对象封装



* 立即执行函数表达式（IIFE）



* 模块化

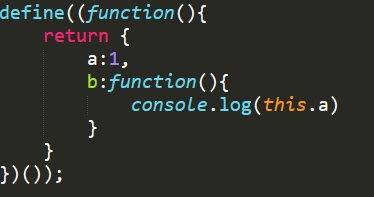
#### 2 常见模块化规范

1)ES6 MODULE

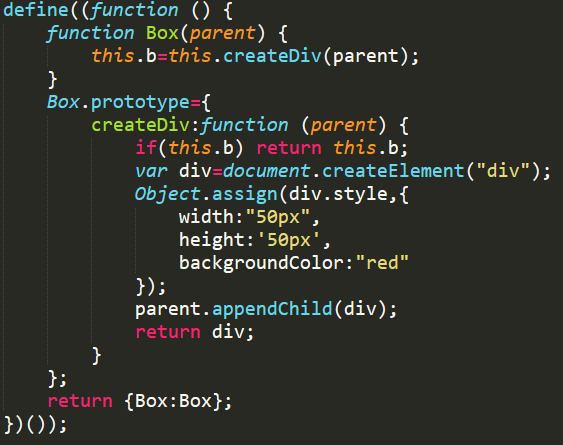
2)COMMONJS:nodejs使用的模块规范

3)AMD(Asynchromous Module Definition) 异步模块定义,代表:require.js

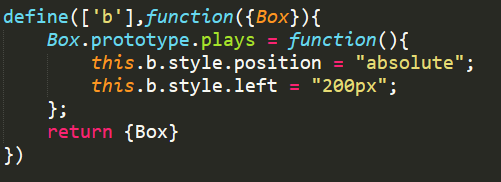
a.js



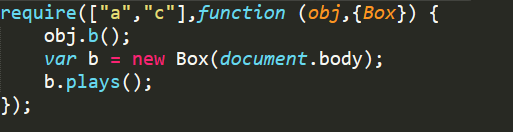
b.js



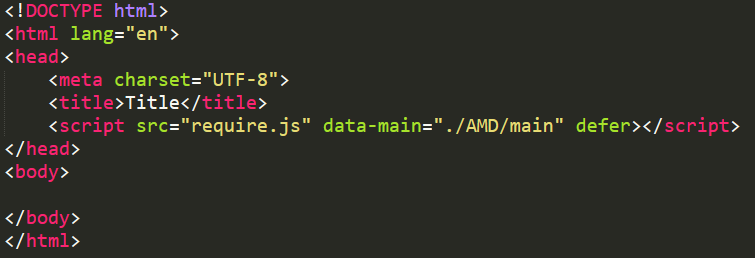
c.js



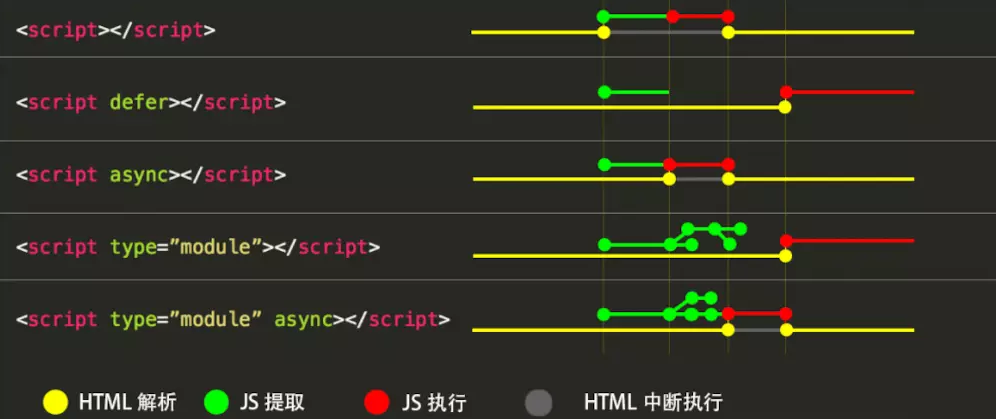
main.js



Amd.html



4)CMD(Common Module Definition) 通用模块定义,代表Sea.js



**defer:延缓**的意思，也就是拖拖拉拉了，所以比较懒，也就是说什么都不想做，也就是哪怕你把饭端在我面前，我也懒得动嘴的那种，这么一想，我们不就记住了，哪怕你客户端把JS文件下载好了，我也懒得执行，最后实在是大家都干完事了，我才不情愿的去执行JS文件。

**async:异步**的意思，异步异步，不就是一步一步嘛，什么都想一步到位，也就是说，只要下载完我就立马执行，至于其他的想都不想。

#### 3 AMD和CMD的区别

第一、对于依赖的模块，AMD 是提前执行，CMD 是延迟执行。不过 RequireJS 从 2.0 开始，也改成可以延迟执行（根据写法不同，处理方式不同）。CMD 推崇 as lazy as possible.

第二、CMD 推崇依赖就近，AMD 推崇依赖前置。

第三、 AMD 的 API 默认是一个当多个用，CMD 的 API 严格区分，推崇职责单一。比如 AMD 里，require 分全局 require 和局部 require，都叫 require。CMD 里，没有全局 require，而是根据模块系统的完备性，提供 seajs.use 来实现模块系统的加载启动。CMD 里，每个 API 都简单纯粹。

### 二 gulp

### 1 、什么是gulp

gulp是一个自动化工具，前端开发者可以使用它来处理常见任务：

* 搭建web服务器
* 文件保存时自动重载浏览器
* 使用预处理器如sass，less
* 优化资源，比如压缩css，javascript，压缩图片

### 2、gulp的安装

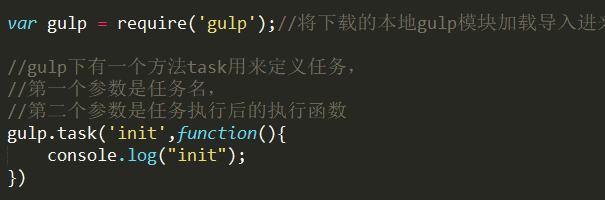
* npm初始化，新建package.json,使用命令：IMG_256
* 先全局安装：IMG_256
* 再需要在当前项目中安装：IMG_256

**扩展：为了项目的可移植性，不建议使用全局安装，你无法保证你代码运行的下一**

**台计算机是否有这个全局gulp，建议在项目目录下使用npx gulp代替全局的gulp命令，npm 5.2.0 以上版本自动安装了npx 。**

### 3、开始使用gulp

* 在根目录下新建gulpfile.js文件
* 这个文件就是配置运行文件，我们先尝试写一个最简单的内容



* 在命令行输入gulp init，这个init就是上面起的任务名，执行后就会执行函数，

打印出init

* 如果任务名是default，命令行输入gulp 即可，不用输入任务，default默认执

行任务名

### gulp的方法

#### 4.1 gulp工作方式:

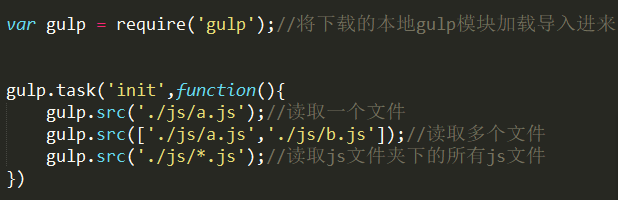
gulp实际工作是使用nodejs中的stream来处理的，简单来说，如果需要合并两个文件，就需要将两个文件都加载进来，然后再处理将文件合并，主力一这里加载进来的文件并不是文本形式加载，而是以二进制数据流的方式，也就是stream加载进来。被加载进来的文件是以二进制数据流模式存在的，内容中除了含有文件内容，还包括了文件的地址，文件名等等一系列相关信息，然后再将所有的数据处理产生新文件，因此这里就用到了nodejs中的stream的方法pipe

pipe:管道，流向的意思，可以在这里理解为写入到，例如：a.pipe(b),就是将a写入到b中。

gulp常用方法有以下几个：src、dest、task和watch。

#### 4.2 src方法

**功能：读取文件的数据流**



**匹配条件：**

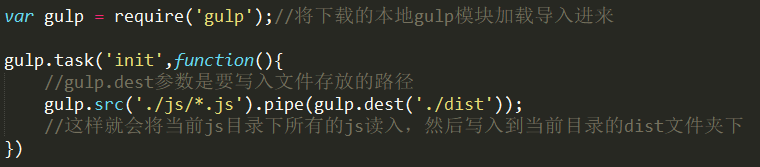
**\***匹配文件路径中的0个或多个字符，但不会匹配路径分割符，除非路径分隔符出现在末尾

**\*\*** 匹配路径中的0个回落多个目录及其子目录，需要单独出现，即它左右不能有其他东西次了，如果出现在末尾，也能匹配文件，也就是能匹配某个目录下所有文件包括其子目录下的所有内容

**？**匹配文件路径中的一个字符（不会匹配路径分隔符）

#### 4.3 dest方法

**功能：给文件写入数据流**



用gulp.dest()把文件流下写入文件后，文件流仍然可以继续使用

#### 4.5 task方法

**功能：定义任务，如果需要有多个任务时就需要定义任务**

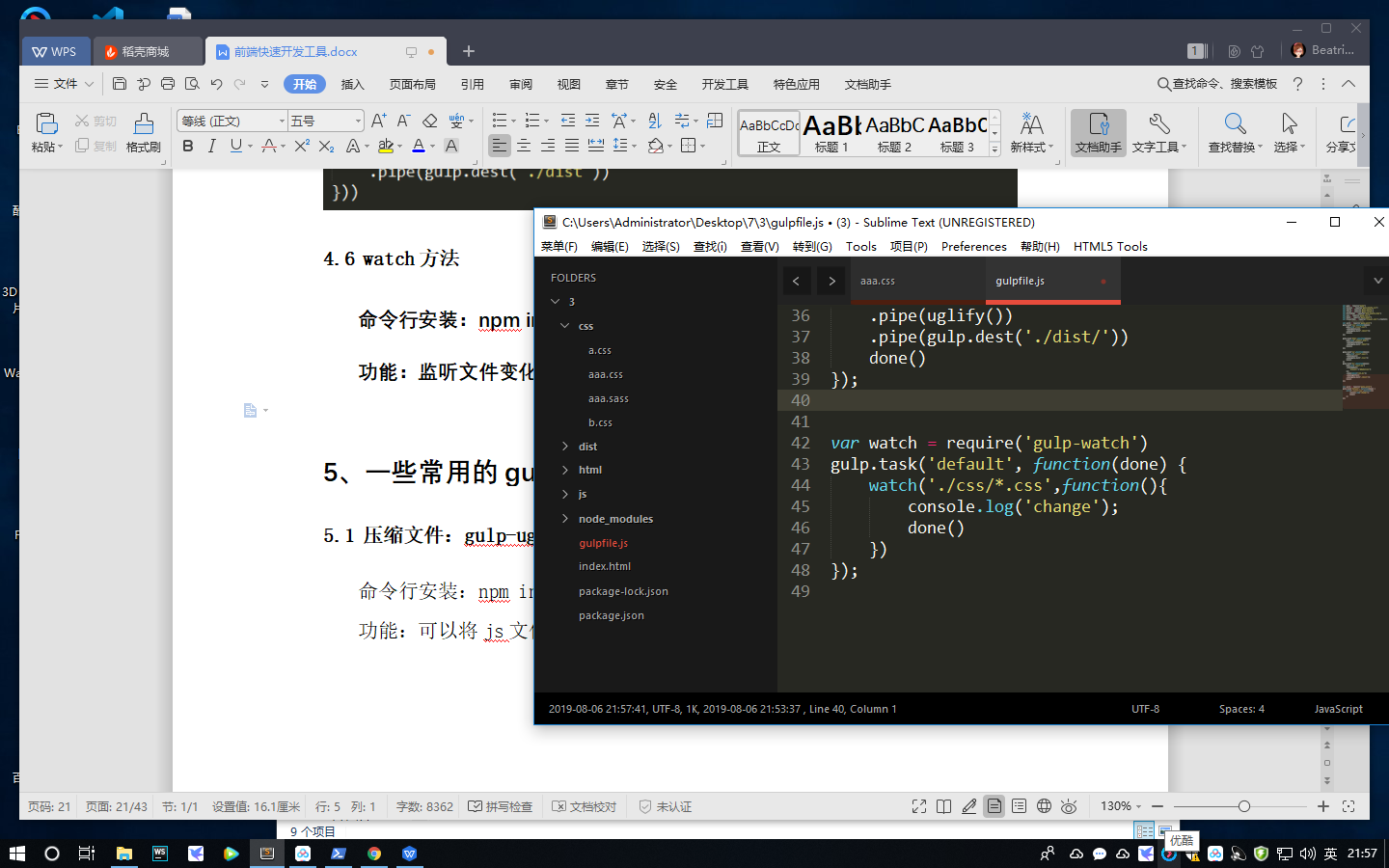


#### 4.6 watch方法

**gulp.watch**(“要观测的文件’,文件发生变化的回调函数)，如果发现该发放不能监听多次便可，可以下载下面的插件。

**命令行安装：npm install gulp-watch -save-dev**

**功能：监听文件变化**

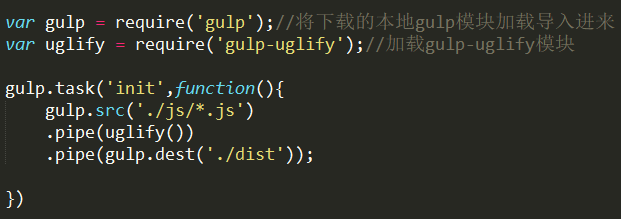


### 5、一些常用的gulp插件

#### 5.1 压缩文件：gulp-uglify

命令行安装：npm install gulp-uglify --save-dev

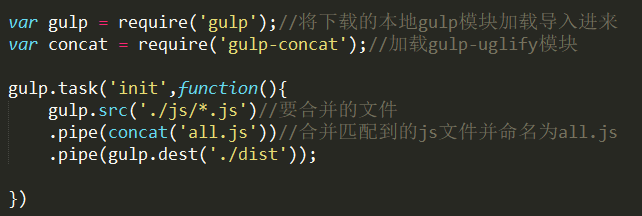
功能：可以将js文件中的空格去除



#### 5.2 文件合并：gulp-concat

命令行安装：npm install gulp-concat --save-dev

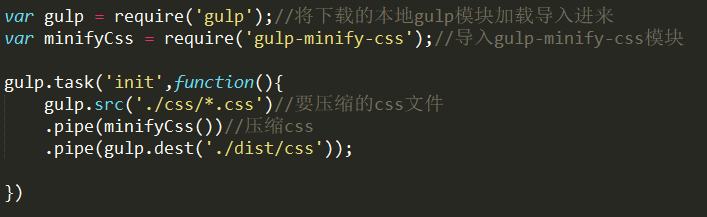
功能：可以合并多个文件



#### 5.3 css文件压缩:gulp-minify-css

命令行安装：npm install gulp-minify-css --save-dev

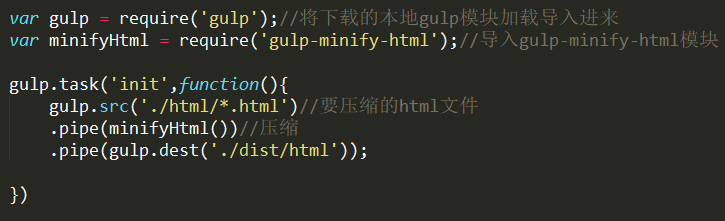
功能：可以对css文件进行压缩



#### 5.4 HTML文件压缩:gulp-minify-html

命令行安装：npm install gulp-minify-html --save-dev

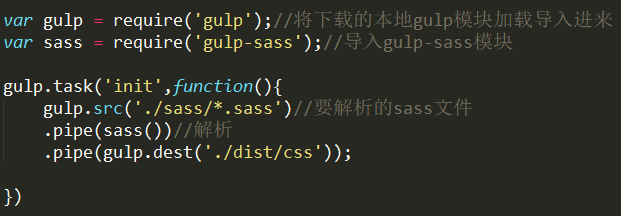
功能：可以对html文件进行压缩



#### 5.5 sass解析：gulp-sass

命令行安装：npm install gulp-sass --save-dev

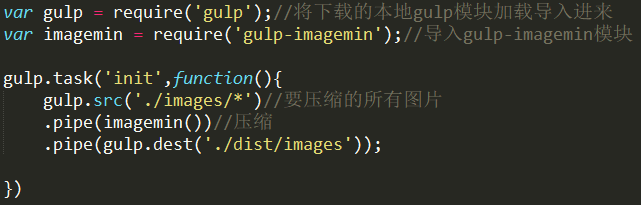
功能：可以对sass文件进行解析



#### 5.6 图片压缩：gulp-imagemin

命令行安装：npm install gulp-imagemin --save-dev

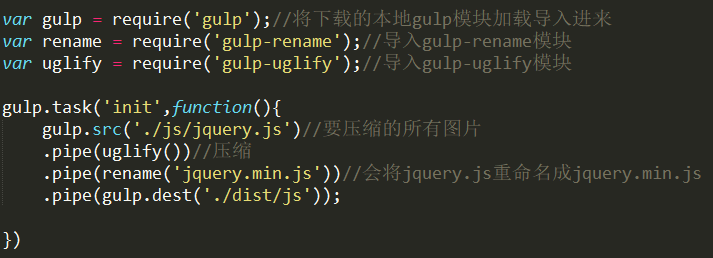
功能：可以对图片文件进行压缩



#### 5.7 重命名：gulp-rename

命令行安装：npm install gulp-rename --save-dev

功能：可以重新改文件起名



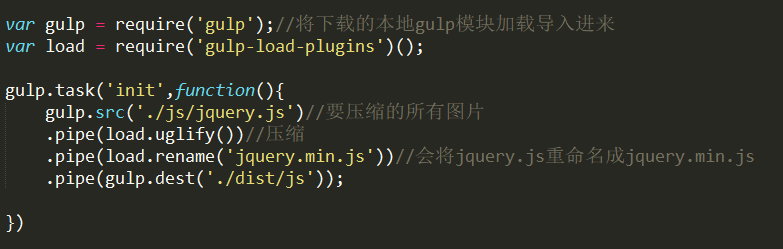
#### 5.8自动加载插件：gulp-load-plugins

命令行安装：npm install gulp-load-plugins --save-dev

功能：在gulpfile中如果需要使用别的插件，导入方法我们使用require，那么如果需要加载很多就会出现使用这个插件的情况

用法：

* 安装并导入gulp-load-plugins
* 在package.json的DevDependencies中写下需要加载的插件，这个是开发以来
* 在gulpfile要导入插件就可以这样写：

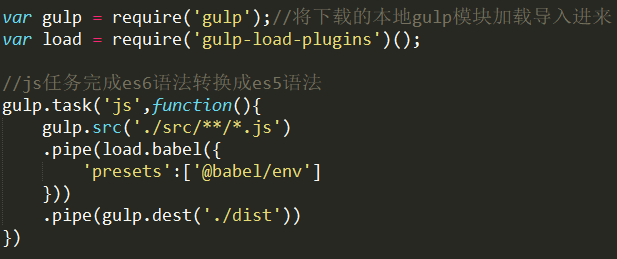


#### 5.9 ES6语法在转ES5：gulp-babel

命令行安装：

npm install gulp-babel @babel/core @babel/preset-env --save-dev

功能：es6语法转es5语法



#### 5.10 静态服务器搭建：browser-sync

命令行安装：

npm install browser-sync --save-dev



#### 6 注意事项：

第一 每个任务的回调函数会传入实参done，把当前任务执行状态变成完成状态。

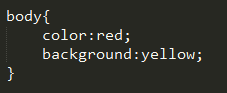
第二 gulp.series(任务1，任务2，任务3......),可以串行任务。

第三 gulp.parallel(任务1，任务2，任务3......)，可以并行任务。

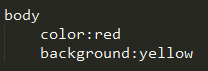
# SASS

#### sass介绍

* 完全兼容 CSS3
* 在 CSS 语言基础上添加了扩展功能，比如变量、嵌套 (nesting)、混合 (mixin)
* 对颜色和其它值进行操作的{Sass::Script::Functions 函数}
* 函数库控制指令之类的高级功能
* 良好的格式，可对输出格式进行定制
* 支持两种语法
  + 第一种被称为 SCSS (Sassy CSS)，是一个 CSS3 语法的扩充版本，这种语种语法的样式表文件需要以 .scss 扩展名。



* + 第二种被称为缩排语法， 提供了一种更简洁的 CSS 书写方式。 它不使用花括号，而是通过缩排的方式来表达选择符的嵌套层级，I 而且也不使用分号，而是用换行符来分隔属性，此种语法的样式表文件需要以 .sass 作为扩展名。

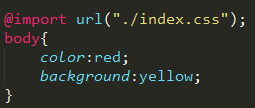


#### 环境搭建及编译指令搭建

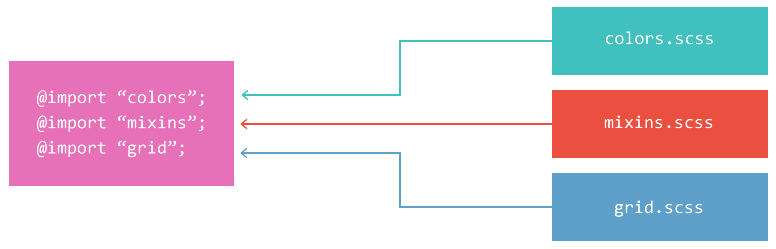


#### sass-导入

css有一个特别不常用的特性，即@import规则，它允许在一个css文件中导入其他css文件，然而，后果是只有执行到@import时，浏览器才会去下载其他css文件，这导致页面加载起来特别慢。



sass也有一个@import规则，但不同的是，sass的@import规则在生成css文件时就把相关文件导入进来。这意味着所有相关的样式被归纳到了同一个css文件中，而无需发起额外的下载请求，使用sass的@import规则并不需要指明被导入文件的全名。你可以省略.sass或.scss文件后缀。



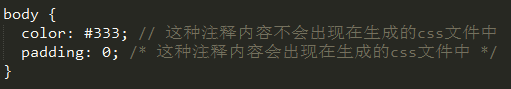
sass不会编译以下划线开头的.scss文件（**局部文件**），而只把这个文件用作导入。当你@import一个局部文件时，还可以不写文件的全名，即省略文件名开头的下划线。例如，你想导入themes/\_night-sky.scss这个局部文件里的变量，你只需在样式表中写@import "themes/night-sky";

**原生的CSS导入，如果出现以下情况：**

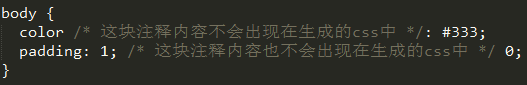
* 被导入文件的名字以.css结尾；
* 被导入文件的名字是一个URL地址；
* 被导入文件的名字是CSS的url()值；

**注释问题：**

在原生的css中，注释对于其他人是直接可见的，但sass提供了一种方式可在生成的css文件中按需抹掉相应的注释——静默注释，静默注释的语法跟JavaScriptJava等类C的语言中单行注释的语法相同，它们以//开头，注释内容直到行末。

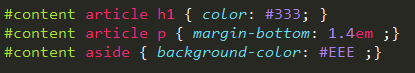


css的标准注释格式/\* ... \*/内的注释内容也可在生成的css文件中抹去。当注释出现在原生css不允许的地方，sass将不知如何将其生成到对应css文件中的相应位置，于是这些注释被抹掉。

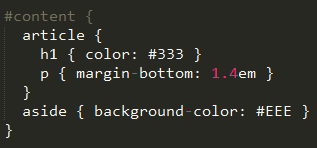


#### sass-嵌套

问题1：css中重复写选择器是非常恼人的，如下图情况：



解决：sass可以让你只写一遍，且使样式可读性更高。在Sass中，你可以像俄罗斯套娃那样在规则块中嵌套规则块。sass在输出css时会帮你把这些嵌套规则处理好，避免你的重复书写，上面代码可以用sass改造成：



问题2：伪元素和伪类怎么嵌套写？

解决：父选择器的标识符是 **&**

|  |  |
| --- | --- |
| **编译前** | **编译后** |
| IMG_256 | IMG_256 |
| IMG_256 | IMG_256 |
| IMG_256 | IMG_256 |

#### 问题3：属性怎么嵌套写？

#### 解决：嵌套属性的规则是这样的：把属性名从中划线-的地方断开，在根属性后边添加一个冒号:，紧跟一个{ }块，把子属性部分写在这个{ }块中

|  |  |
| --- | --- |
| **编译前** | **编译后** |
| IMG_256 | IMG_256 |

#### sass-变量定义

sass的变量名可以与css中的属性名和选择器名称相同，包括中划线和下划线。

|  |  |
| --- | --- |
| **编译前** | **编译后** |
| IMG_256 | IMG_256 |

注意：1 反复声明一个变量，只有最后一处声明有效；

2 $fancybox-width: 400px **!default**;如果这个变量被声明赋值了，那就用它声明的值，否则就用这个默认值

#### sass-判断语句

当 @if 的表达式返回值不是 false 或者 null 时，条件成立，输出 {} 内的代码

|  |  |
| --- | --- |
| **编译前** | **编译后** |
| IMG_256 | IMG_256 |

#### sass-循环语句

@for 指令可以在限制的范围内重复输出格式，每次按要求（变量的值）对输出结果做出变动。这个指令包含两种格式：

@for $var from <start> through <end>

@for $var from <start> to <end>

区别：

through：条件范围包含 <start> 与 <end> 的值

to：条件范围只包含 <start> 的值不包含 <end> 的值

|  |  |
| --- | --- |
| **编译前** | **编译后** |
| IMG_256 | IMG_256 |

#### sass-mixin函数

当你的样式越来越复杂，你需要大段大段的重用样式的代码，变量就没办法应付这种情况，你可以通过sass的混合器实现大段样式的重用。

**混合器使用@mixin标识符定义**，这个标识符给一大段样式赋予一个名字，你可以**通过@include引用这个名字重用**这段样式。如下边的sass代码，定义了一个简单的混合器，目的是添加跨浏览器的圆角边框。

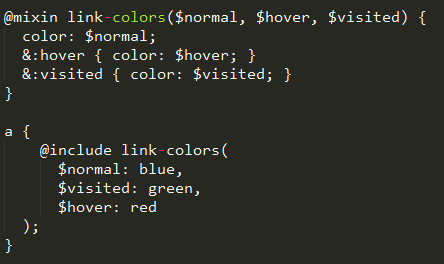
|  |  |
| --- | --- |
| **编译前** | **编译后** |
|  | IMG_256 |

**给混合器传参**

可以通过在@include引入混合器时给混合器传参，来定制混合器生成的精确样式。

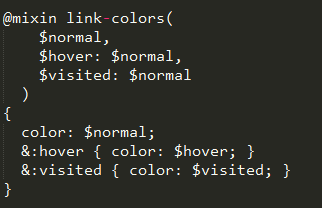
|  |  |
| --- | --- |
| **编译前** | **编译后** |
| IMG_256 | IMG_256 |

当你@include混合器时，sass允许通过语法$name: value的形式指定每个参数的值，这种参数顺序就重要了，只需要保证没有漏掉参数即可：



**默认参数值**

参数默认值使用$name: default-value的声明形式，默认值可以是任何有效的css属性值，甚至是其他参数的引用，如下代码：



#### sass-function介绍

Sass 支持自定义函数，并能在任何属性值或 Sass script 中使用

|  |  |
| --- | --- |
| **编译前** | **编译后** |
| IMG_256 | IMG_256 |

# 第五章 Git

##### 一 、版本控制工具

如果你用Microsoft Word写过长篇大论，那你一定有这样的经历：

想删除一个段落，又怕将来想恢复找不回来怎么办？有办法，先把当前文件“另存为……”一个新的Word文件，再接着改，改到一定程度，再“另存为……”一个新文件，这样一直改下去，最后你的Word文档变成了这样：



过了一周，你想找回被删除的文字，但是已经记不清删除前保存在哪个文件里了，只好一个一个文件去找，真麻烦。

看着一堆乱七八糟的文件，想保留最新的一个，然后把其他的删掉，又怕哪天会用上，还不敢删，真郁闷。

更要命的是，有些部分需要你的财务同事帮助填写，于是你把文件Copy到U盘里给她，然后，你继续修改Word文件。一天后，同事再把Word文件传给你，此时，你必须想想，发给她之后到你收到她的文件期间，你作了哪些改动，得把你的改动和她的部分合并，真困难。

于是你想，如果有一个软件，不但能自动帮我记录每次文件的改动，还可以让同事协作编辑，这样就不用自己管理一堆类似的文件了，也不需要把文件传来传去。如果想查看某次改动，只需要在软件里瞄一眼就可以，岂不是很方便？

这个软件用起来就应该像这个样子，能记录每次文件的改动：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本 | 文件名 | 用户 | 说明 | 日期 |
| 1 | index.html | Lucy | 改动了轮播图的小圆点 | 8/12 10:38 |
| 2 | index.html | Lily | 增加了侧边栏 | 8/12 11:40 |
| 3 | index.html | Lucy | 删除了导航栏的联系我们菜单 | 8/13 9:00 |
| 4 | index.html | Lily | 增加了底部的回到顶部按钮 | 8:13 11:00 |

这样，你就结束了手动管理多个“版本”的史前时代，进入到版本控制的20世纪。

##### 二、版本管理工具的主要类型

##### 1 、SVN（集中式）

Subversion(SVN) 是一个开源的版本控制系統, 也就是说 Subversion 管理着随时间改变的数据。 这些数据放置在一个中央资料档案库(repository) 中。 这个档案库很像一个普通的文件服务器, 不过它会记住每一次文件的变动。 这样你就可以把档案恢复到旧的版本, 或是浏览文件的变动历史。

**SVN 的一些概念**

**repository**（源代码库）:源代码统一存放的地方

**Checkout**（提取）:当你手上没有源代码的时候，你需要从repository checkout一份

**Commit**（提交）:当你已经修改了代码，你就需要Commit到repository

**Update** (更新):当你已经Checkout了一份源代码， Update一下你就可以和Repository上的源代码同步，你手上的代码就会有最新的变更

日常开发过程其实就是这样的（假设你已经Checkout并且已经工作了几天）：Update(获得最新的代码) -->作出自己的修改并调试成功 --> Commit(大家就可以看到你的修改了)。

如果两个程序员同时修改了同一个文件呢, SVN 可以合并这两个程序员的改动，实际上SVN管理源代码是以行为单位的，就是说两个程序员只要不是修改了同一行程序，SVN都会自动合并两种修改。如果是同一行，SVN 会提示文件 Conflict, 冲突，需要手动确认。

**具体使用方法**：

SVN服务器：VisualSVN-Server

SVN客户端：TortoiseSVN

#### 2、Git（分布式）

##### 1、git的发展史

很多人都知道，Linus在1991年创建了开源的Linux，从此，Linux系统不断发展，已经成为最大的服务器系统软件了。

Linus虽然创建了Linux，但Linux的壮大是靠全世界热心的志愿者参与的，这么多人在世界各地为Linux编写代码，那Linux的代码是如何管理的呢？

事实是，在2002年以前，世界各地的志愿者把源代码文件通过diff的方式发给Linus，然后由Linus本人通过手工方式合并代码！

你也许会想，为什么Linus不把Linux代码放到版本控制系统里呢？不是有CVS、SVN这些免费的版本控制系统吗？因为Linus坚定地反对CVS和SVN，这些集中式的版本控制系统不但速度慢，而且必须联网才能使用。有一些商用的版本控制系统，虽然比CVS、SVN好用，但那是付费的，和Linux的开源精神不符。

不过，到了2002年，Linux系统已经发展了十年了，代码库之大让Linus很难继续通过手工方式管理了，社区的弟兄们也对这种方式表达了强烈不满，于是Linus选择了一个商业的版本控制系统BitKeeper，BitKeeper的东家BitMover公司出于人道主义精神，授权Linux社区免费使用这个版本控制系统。

安定团结的大好局面在2005年就被打破了，原因是Linux社区牛人聚集，不免沾染了一些梁山好汉的江湖习气。开发Samba的Andrew试图破解BitKeeper的协议（这么干的其实也不只他一个），被BitMover公司发现了（监控工作做得不错！），于是BitMover公司怒了，要收回Linux社区的免费使用权。

Linus可以向BitMover公司道个歉，保证以后严格管教弟兄们，嗯，这是不可能的。实际情况是这样的：

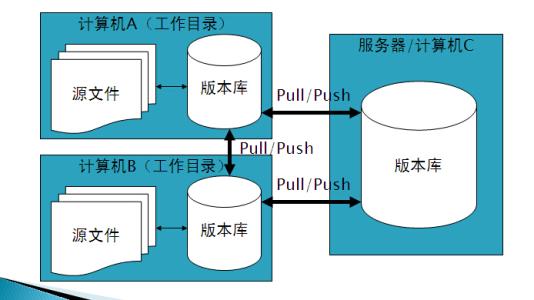
**Linus花了两周时间自己用C写了一个分布式版本控制系统，这就是Git**！一个月之内，Linux系统的源码已经由Git管理了！牛是怎么定义的呢？大家可以体会一下。

Git迅速成为最流行的分布式版本控制系统，尤其是2008年，GitHub网站上线了，它为开源项目免费提供Git存储，无数开源项目开始迁移至GitHub，包括jQuery，PHP，Ruby等等。

历史就是这么偶然，如果不是当年BitMover公司威胁Linux社区，可能现在我们就没有免费而超级好用的Git了。

##### 2 Git和SVN的区别：

Git是一个分布式版本管理工具，和SVN不同的是git把更多的内容放在了本地仓库，让本地仓库有了更多的权限，从而不占用网络资源快速地在本地创建各种版本信息，在最终确定版本之后进行统一地推送，提高了版本管理效率，减轻了网络资源负担。如下图 : 每个计算机都拥有一个独立的版本库，通过 pull 和 push 等操作对线上进行保存。



同时git的线上仓库（或者叫开源社区）也日渐成熟，国内有 gitee , 国际上有 github,都可以免费地对私人代码进行远程托管 ，当然因为网络限制所以github速度有些感人，网络条件不好的小伙伴推荐使用gitee。 这些线上的开源社区中也存在大量的代码供我们学习交流使用，我们也可以开源属于我们的代码，进入一个更大的世界。

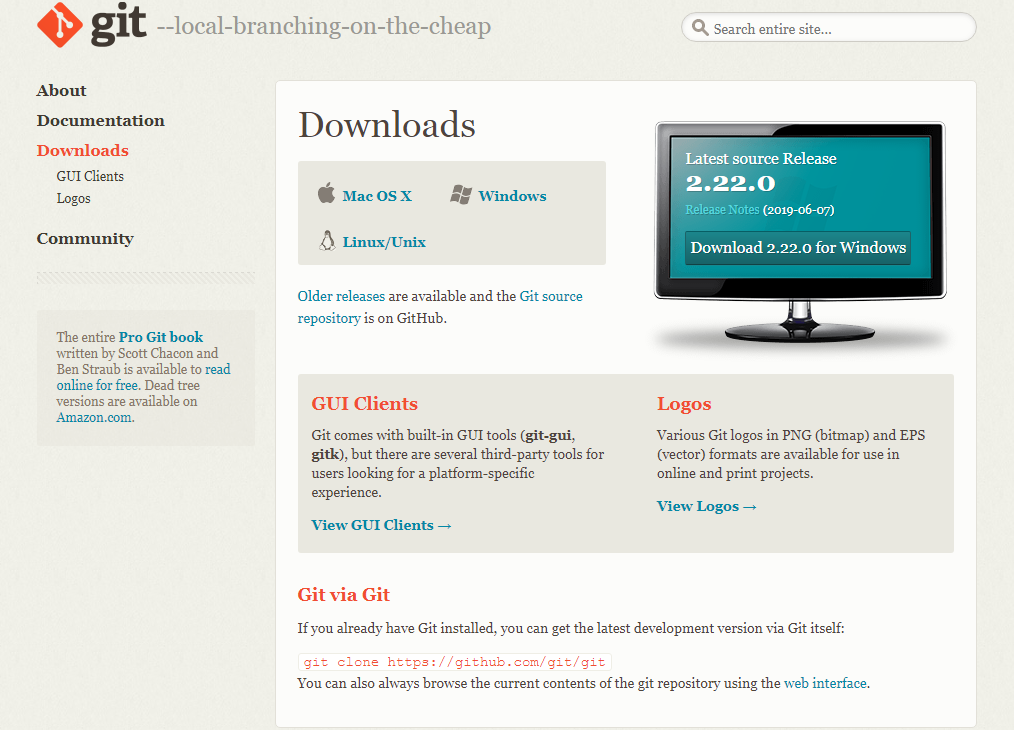


### 二 具体使用方法

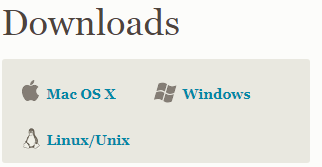
* 本地仓库
* 远程仓库
* 分支管理

##### 1、开始学习之前,先安装gitbash 工具:

Git平台安装包下载地址为:http://git-scm.com/downloads(下载速度较慢)



选择相应版本安装即可:



安装完成后，需要配置个人用户信息和电子邮箱地址，在命令行输入：

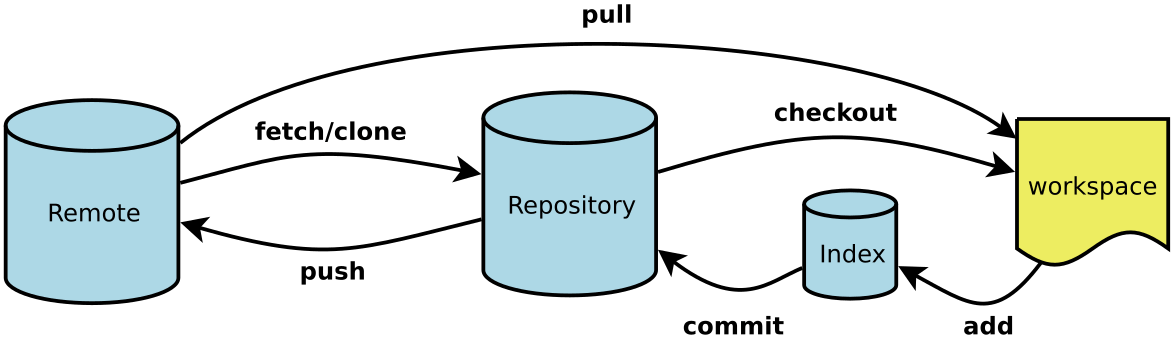
$ git config --global user.name "chenyanrui"

$ git config --global user.email "chenyanrui@1000phone.com"

这一步操作可以是说是至关重要的，因为我们需要告知gitbash你的基本信息，便于gitbash去进行记录，在必要的时候进行使用,通过命令行:git config --list可以查看配置.

##### 2.初始化一个仓储/仓库:git init

本质创建了一个名为.git的隐藏目录



* Workspace：工作区
* Index / Stage：暂存区
* Repository：仓库区（或本地仓库）
* Remote：远程仓库

##### 3.添加一个文件：git add 文件路径

可以使用命令**git add .** 一次性把所有修改过或新添加的文件添加的仓库门口

这里的.表示当前路径

从暂存区删除文件 **git rm 文件路径 --cache**

## 4.提交一个文件：git commit -m "本次提交的简要说明信息"

## 5.合并操作(不推荐,管理自己代码时可以这么做)(了解)：git commit -a -m "提交的信息"

## 6.对比与文件变化:git diff

用于项目中的代码与add后的代码的区别

##### 7.版本回退

* 命令: git reset --hard Head
  + Head 表示得到最近的一次提交的代码
  + Head^表示上一次提交的代码，Head^^表示上上次提交的代码，当然往上100个版本写100个^比较容易数不过来，所以写成HEAD~100
* 命令： git reset --hard 提交的版本号
  + 可以回退到指定的版本(根据每次提交的版本的号)
* 命令: git reflog 可以查看每一次的提交的版本号
* 命令: git log命令显示从最近到最远的提交日志

注意：

版本回回退后，你如果用git log命令查看就会发现回退前的一个版本不见了，你可以通过git reflog查看所有的版本号，通过git reset --hard 提交的版本号，回到指定的版本。版本号没必要写全，前几位就可以了，Git会自动去找。当然也不能只写前一两位，因为Git可能会找到多个版本号，就无法确定是哪一个了。

**在项目中我们也经常会使用到撤销操作,** **撤销操作主要有如下几种**

* git commit --amend 撤销上一次提交 并讲暂存区文件重新提交
* git checkout -- 文件路径 拉取暂存区文件 并将其替换成工作区文件
* git reset HEAD -- 文件路径 拉取最近一次提交到版本库的文件到暂存区 改操作不影响工作区

##### 8.忽略文件

* 新建 **.gitignore**文件
* 在这个文件里写一些想要被git忽略的文件或者文件夹
* 可以通过命令**git status**查看仓库状态

##### 分支管理

分支就是科幻电影里面的平行宇宙，当你正在电脑前努力学习Git的时候，另一个你正在另一个平行宇宙里努力学习SVN。

如果两个平行宇宙互不干扰，那对现在的你也没啥影响。不过，在某个时间点，两个平行宇宙合并了，结果，你既学会了Git又学会了SVN！



分支在实际中有什么用呢？假设你准备开发一个新功能，但是需要两周才能完成，第一周你写了50%的代码，如果立刻提交，由于代码还没写完，不完整的代码库会导致别人不能干活了。如果等代码全部写完再一次提交，又存在丢失每天进度的巨大风险。

现在有了分支，就不用怕了。你创建了一个属于你自己的分支，别人看不到，还继续在原来的分支上正常工作，而你在自己的分支上干活，想提交就提交，直到开发完毕后，再一次性合并到原来的分支上，这样，既安全，又不影响别人工作。

其他版本控制系统如SVN等都有分支管理，但是用过之后你会发现，这些版本控制系统创建和切换分支比蜗牛还慢，简直让人无法忍受，结果分支功能成了摆设，大家都不去用。

但Git的分支是与众不同的，无论创建、切换和删除分支，Git在1秒钟之内就能完成！无论你的版本库是1个文件还是1万个文件。

* 创建分支：**git branch 分支名**
* 列出所有分支：**git branch**， 在列出所以分支中，**当前分支前会有个\*号**
* 切换分支：**git checkout 分支名**
* 创建和切换分支可以合并为：**git checkout -b 分支名**

    在自己的分支上做一些提交操作后，当我们在新创建的分支中把功能完成之后，需要把功能给别人用，就需要把功能拿到主分支上去，具体方法如下：

* 切换到主分支:**git checkout master**
* 合并分支,把新创建的分支的提交合并到主分支中: **git merge 分支名**，把指定分支名的分支合并到当前分支
* 合并分支之后，我们可以**清除之前临时使用的分支：git branch -d 分支名**

冲突解决方法：

* 当两个人同时修改一个文件，并提交后，合并这两个分支回发生冲突，
* Git用<<<<<<<，=======，>>>>>>>标记出不同分支的内容
* 我们可以手动修改冲突文件，再次提交，即可解决冲突
  + $ git add readme.txt
  + $ git commit -m "conflict fixed"
  + [master cf810e4] conflict fixed

##### 10 Git与GitHub的关系

* Git版本管理工具
* GitHub就是一个网站：注册github账号

##### 11 推送(push)与拉取(pull)与克隆（clone）

每天的工作日常：

第一：下载远程仓库到本地：git clone 远程仓库地址

第二:使用cd命令进入下载的文件夹(或者双击文件夹)

第三：切换到你的工作本支并更新分支代码

* git clone是将整个工程复制下来，不需要本地是仓库

第三：把本地代码提交远程仓库

* git remote add origin <https://github.com/cyanrui/hkhk.git>
* git push -u origin master

注意：

git clone：从远程服务器克隆一个一模一样的版本库到本地,复制的是整个版本库.（clone是将一个库复制到你的本地，是一个本地**从无到有**的过程）

git pull：从远程服务器获取到一个branch分支的更新到本地，并**更新本地库**.（pull是指同步一个在你本地有版本的库内容更新的部分到你的本地库）