目录

[Highcharts、git、seajs 1](#_Toc31529)

[复习： 3](#_Toc143)

[一、 大数据可视化 5](#_Toc13647)

[二、git 6](#_Toc14886)

[2.1 git指令 6](#_Toc11410)

[2.2 介绍git 7](#_Toc21479)

[2.3 使用git 7](#_Toc7623)

[2.4建立信任关系 8](#_Toc3453)

[2.5 创建项目 10](#_Toc28556)

[2.6 建立连接 11](#_Toc10809)

[2.7 上传代码 11](#_Toc14935)

[2.8 浏览项目 11](#_Toc24956)

[2.9 克隆项目 11](#_Toc23996)

[下午复习： 12](#_Toc27487)

[三、 seajs介绍 13](#_Toc29951)

[3.1 体验seajs 13](#_Toc21367)

[3.2 定义模块 14](#_Toc19767)

[3.3 require模块 15](#_Toc19184)

[四、 加载具有id的模块 17](#_Toc20875)

[五、 接口定义 17](#_Toc15447)

[六、 模块信息对象 18](#_Toc17999)

# 复习：

严格模式：

ES5拓展了严格模式，可以使我们的代码更加的安全可靠

开启方式：

如果在js第一行加入“use strict”此时将处于“全局严格模式”

如果在某个函数的第一行加入“use strict”当函数执行的时候，此时将处于“局部严格模式”

严格模式与正常模式之间的区别：

在严格模式下定义变量必须使用var

在严格模式下定义数值的时候不能使用八进制

在严格模式下不能使用argumens.callee

在严格模式下不是使用delete删除变量

在严格模式下不能使用保留字定义变量

在严格模式下不能使用eval函数

在严格模式下不能使用with方法

在严格模式下全局函数作用域是undefined

在严格模式下定义对象中的属性不能是同名的

在严格模式下定义函数中的参数不能是同名的

特性：

特性就是用来描述属性的

设置单一属性特性：

Object.defineProperty(obj, property, descript)

obj: 要设置的对象

property: 要设置特性的属性

descript: 描述特性对象

// 配置值

value: “”

//配置可修改性

Writable: true | false

// 配置可枚举性

enumerable: true | false

// 可配置性

configurable: true | false

设置多个特性属性：

Object.defineProperties(obj, options)

obj: 要设置的对象

options: 配置对象

key: 要设置的特性属性

value: 描述特性对象

特性方法：

get方法用于获取值的

没有参数

返回值就是要获取的属性值

作用域是当前对象

千万不要在方法中获取该属性值，否则会递归死循环

通常我们是获取该属性值的一个备用值

set方法用于设置值的

参数就是要设置的值

没有返回值

作用域当前对象

千万不要在方法中设置该属性值，否则会递归死循环

通常我们是设置该属性值的一个备用属性

原型方法：

isPrototypeOf:   
 判断原型对象是否是参数对象的原型

参数就是实例化对象

在查找的过程中会查找整个原型链

getPrototypeOf:

用于获取原型对象的

setPrototypeOf:

设置原型对象

创建对象的新方式：

Object.create(prototype, options)

# 大数据可视化

简单来说： 做图表的

官网：<https://www.highcharts.com.cn/>

下载：

|  |
| --- |
|  |

2：

|  |
| --- |
|  |

3

|  |
| --- |
|  |

跳出弹出框的时候不需要理会，稍后会进行下载

# 二、git

linux将代码发布到网站了，开源了，因此很多开发者不断的贡献代码，这样代码越来越多了，就有了管理代码的问题了，所以创建git，就是用来管理linux代码的，后来发现很好用，所以越来越多的项目开始放在git上了，后来项目越来越多了，其他系统也就开始支持git了（windows， mac）

我们要使用git就要安装客户端

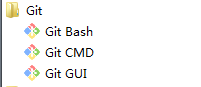


双击进行下一步安装

安装完成，提供了控制台

Git Bash 为linux家族（unix，mac等）使用的

Git CMD 为window使用的



由于git为linux提供的，所以建议我们使用linux指令

## 2.1 git指令

cd 打开目录

ls 查看当前目录中的文件

pwd 查看目录所在的系统文件路径

mkdir 用于创建文件夹

touch 创建文件的

echo 内容 >> 文件中

小案例：在C:\Users\Administrator\Desktop\2019年4月22日大数据与git\案例目录文件下，使用指令创建一个demo文件夹，创建一个redeme.txt文件并且写入爱创课堂

cd 目录

pwd 查看所在的系统文件路径

mkdir 创建一个demo文件夹

cd demo文件夹目录下

touch readme.txt文件

## 2.2 介绍git

在git中分为三类文件

第一类是未纳入缓存的文件

这类文件，一旦删除将无法找到

第二类是纳入缓存的文件

这类文件，可以通过git找到

第三类文件是纳入版本库的文件

这类文件，可以通过计算机的各个位置找到

## 2.3 使用git

首先我们可以通过git init 来初始化一个项目文件

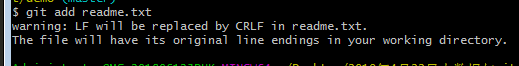
objects 存储纳入缓存的文件

refs 用来存储纳入版本库的文件

我们可以通过指令实现三类文件的切换

git add 文件

实现将未纳入缓存的文件纳入缓存 （第二类文件）



git rm --cached 文件

可以实现将纳入缓存的文件再次变为第一类文件



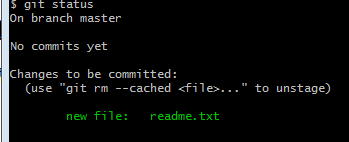
git add \*

可以实现将当前目录中的所有文件纳入缓存



git status

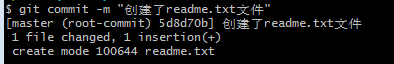
用于查看文件的状态



一旦纳入缓存， 将有机会纳入版本库

git commit -m “说明”

说明一定要有语义化，让别人直到你提交的内容



一旦纳入版本库之后，这些文件可以通过计算机的各个位置找到，但是，当计算机坏掉了，文件就无法获取，所以我们可以将文件上传到云端（服务器），即使电脑坏掉，文件也可以被找到

## 2.4建立信任关系

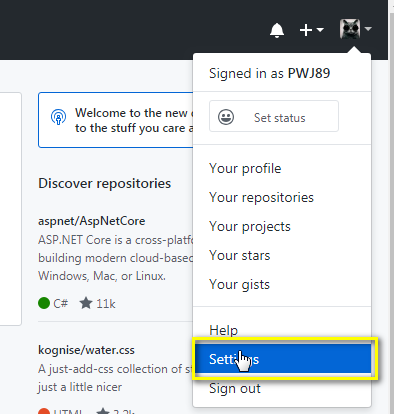
执行指令 ssh-keygen -t rsa -C 邮箱

进入目录C:\Users\think\.ssh

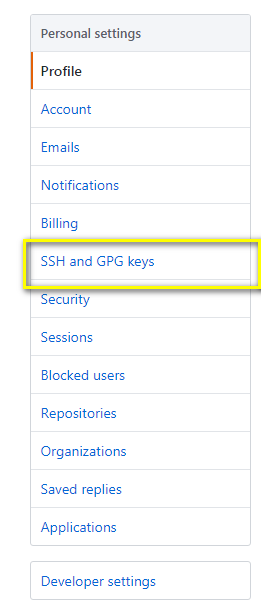
打开id\_rsa.pub文件，复制里面的内容

进入gitHub:

1：



2：

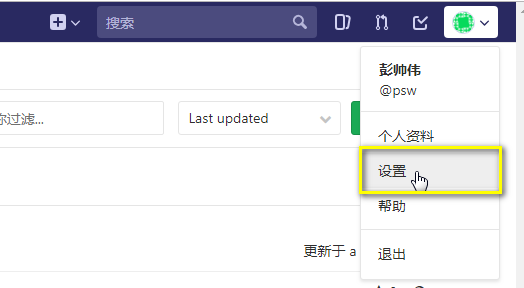


3：



进入gitlab中：

1：



2：



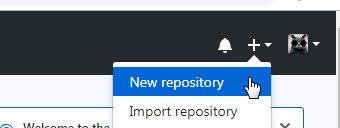
3：



## 2.5 创建项目

进入gitHub中：

1：

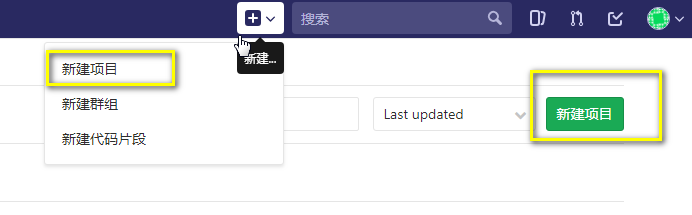


2：

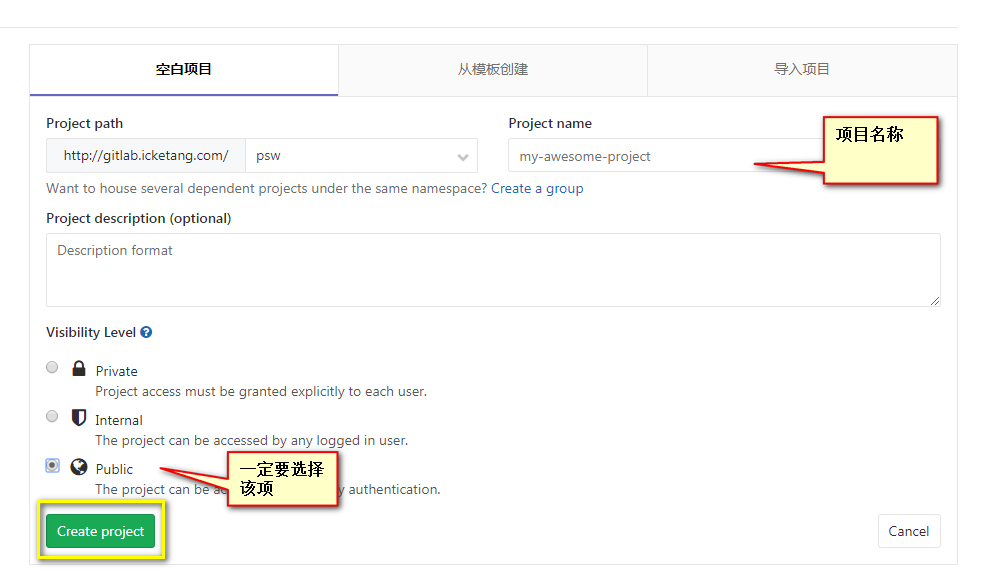


进入gitlab中：

1：



2：



## 2.6 建立连接

git remote add origin <https://github.com/PWJ89/demo2.git>

## 2.7 上传代码

git push -u origin master

## 2.8 浏览项目

在git中提供了可以浏览项目的地址

<http://htmlpreview.github.io/>

我们可以在地址栏输入项目的首页网址查看项目

## 2.9 克隆项目

我们可以通过 git clone 文件 去克隆别人的项目

## 下午复习：

git指令：

cd 打开目录

ls 查看当前目录中的文件

pwd 查看当前目录所在的系统文件路径

mkdir 创建文件夹

touch 文件 创建文件的

echo 内容 >> 文件

git介绍:

在git中有三类文件

第一类是未纳入缓存的文件

这类文件一旦删除将无法找到

第二类是纳入缓存的文件

这类文件可以通过git找到

第三类是纳入版本库的文件

这类文件，可以通过计算机的各个位置找到

git使用：

git init 初始化一个项目文件

我们可以通过git指令实现三类文件的切换

git add 文件  
 实现将未纳入缓存的文件变为第二类文件（纳入缓存）

git add \*

实现将文件目录中所有文件都变为第二类文件

git rm --cached 文件

实现将第二类文件变为第一类文件（未纳入缓存的文件）

git status

查看文件的状态

一旦纳入缓存将有机会纳入版本库

git commit -m “说明”

实现将第二类文件纳入版本库

# 三、 seajs介绍

seajs是一套模块化开发框架，是由淘宝工程师王伯所写的，有中文文档，学习起来比较简单

开源库代码：

<https://github.com/seajs/seajs>

由于Seajs出现的比较晚，因此借鉴了nodejs的commonjs规范（加载时是同步的），但是在前端使用文件的时候都是要异步加载的，加载完毕之后才能使用，又借鉴了前端的AMD规范，seajs的规范称之为cmd规范，

这套框架在使用的时候，建议我们使用commonjs规范

## 3.1 体验seajs

当引入seajs之后会向全局暴露两个变量： seajs、 define

在模块文件外部想要引入一个模块文件要使用seajs.use方法

该方法接收两个参数

第一个参数是一个数组， 数组中的每一项都是一个模块文件路径（当引入的只有一个文件的时候， 可以省略数组）

第二个参数是回调函数

回调函数中的参数就是前面模块向外暴露的功能（一一对应的）

根目录：

seajs引入的文件路径是以seajs所在的目录决定的（seajs所在的目录就是根目录）

通常我们要把seajs文件放在最外面与index.html文件在同一级

为了引入文件方便

seajs对js文件敏感，因此可以省略.js后缀

seajs:

|  |
| --- |
|  |

seajs本身是一个对象， 对象中包含各种功能函数以及相关信息

seajs.use方法： 规定了如何使用模块

seajs.config方法： 规定了如何配置模块

define:

|  |
| --- |
|  |

define规定了如何定义模块

## 3.2 定义模块

seajs提供了一个define方法用于定义模块，可以传递三个参数，因此提供了六种定义模块的方式：

可以传递一个参数

传递的是值类型的时候 （数字、字符串、布尔值）

当传递的是值类型的时候这些会作为接口直接暴露

传递引用类型 （对象、数组）

这些也会作为接口直接暴露

传递函数(90%以上都是使用这种方式，基于commonjs规范)  
 此时函数中有三个参数

第一个参数require 用于引入其它模块文件的

第二个参数是exports 向外暴露功能的对象

第三个参数module 模块信息对象

exports和module用于定义向外暴露功能的对象

|  |
| --- |
| // 传递值类型  // 传递数字  define(123);  // 传递字符串  define("abc");  // 传递布尔值  define(true);  // 传递引用类型  // 传递对象  define({  a: 1,  b: 2  })  // 传递数组  define(["red", "green", "blue"]);  // 传递函数  define(function(require, exports, module) {  // console.log(this, arguments);  // 这里的this指向window 因此不要添加任何属性和方法  this.color = "red";  }) |
|  |

传递两个参数：

第一个参数可以是字符串，表示模块的id

此时seajs.use方法将无法直接使用 （要配合require方法）

第一参数还可以是数组，表示模块的依赖集合（集合中的模块文件不管有没有使用，都会加载）

第二个参数是回调函数（与上面第三个定义模块方式中的函数表现形式是一致的）

|  |
| --- |
| // define("abc", function() {  define("module/main", function() {  console.log(this, arguments);  }) |

传递三个参数：

第一个参数字符串，表示模块的id

第二个参数是数组，表示模块的依赖集合

第三个参数是回调函数（同第三种定义模块方式中的函数表现形式一致）

|  |
| --- |
| // define("abc", ["module/dom"], function() {  define("module/main", ["module/dom"], function() {  console.log(this, arguments);  }) |

## 3.3 require模块

在一个模块文件中引入其它模块要分为两步走

第一步要通过require方法引入模块文件

第二步要通过模块文件读取该模块

举例：

|  |
| --- |
| 1. // 定义模块 2. define(function(require, exports, module) { 3. // console.log(this, arguments); 4. // 第一步要通过require方法引入其它模块文件 5. var dom = require("module/dom"); 6. // 读取 7. console.log(dom); 8. }) |

dom.js:

|  |
| --- |
| 1. // 定义模块 2. define({ 3. num: 100, 4. color: "red" 5. }) |

结果：

|  |
| --- |
|  |

1 require不能简写

|  |
| --- |
| 1. // 1 require不能被简写 2. define(function(req, exports, module) { 3. var dom = req("module/dom"); 4. console.log(dom); 5. }) |

2 require不能被修改

1 reuqire不能被赋值

2 require不能赋值给其他变量

3 不能作为子函数中的参数传递

4 不能在子函数中被修改 （以上3点）

|  |
| --- |
| 1. // 2 require不能被修改 2. define(function(require, exports, module) { 3. // 1 require不能被赋值 4. // require = 123; 5. // 2 require不能赋值给其它变量 6. // var a = require; 7. // 3 require不能作为子函数中的参数传递 8. function demo(require) { 9. // 1 require不能被修改 10. // var a = require; 11. // 2 require不能被赋值 12. // requrie = 123; 13. var dom = requrie("module/dom"); 14. console.log(dom); 15. } 16. demo(require); 17. // // 引入其它模块文件 18. // var dom = require("module/dom"); 19. // console.log(dom); 20. }) |

3 require只能是字符串，不能拼接

|  |
| --- |
| 1. // 3 require只能是字符串 不能拼接 2. define(function(require, exports, module) { 3. // 引入dom.js文件 4. // var dom = require("module/dom"); 5. // 字符串拼接 6. var dom = require("module" + "/" + "dom"); 7. console.log(dom); 8. }) |

# 加载具有id的模块

如果一个模块中存在显示的id, 此时将无法直接使用

如果想要加载具有id的模块要分为两步走

第一步通过模块的依赖集合加载该模块文件

第二步通过require方法指定模块的id

当一个模块文件中存在两个相同id的模块，前面的覆盖后面的模块

当一个模块文件中存在两个没有id的模块，后面的模块覆盖前面的

当一个模块文件名中存在多个相同id的模块，可以同时存在，引入的模块是以require为准

|  |
| --- |
| 1. // 引入具有id的模块 2. define(["module/dom"], function(require, exports, module) { 3. // 第一步通过模块的依赖集合加载该模块 4. // 第二步通过require指定模块的id 5. var dom1 = require("myId1") 6. var dom2 = require("myId2") 7. console.log(dom1, dom2); 8. }) |

# 接口定义

|  |
| --- |
| 1. // 定义模块 2. define(function(requrie, exports, module) { 3. // console.log(module) 4. // 第一种定义接口的方式 5. exports.title = "nihao"; 6. // 以下方式是绝对不允许使用 7. // exports = { 9. // } 10. // 第二种定义接口的方式 11. module.exports.a = 123; 12. // 第三种 13. // 这种方式会覆盖掉上面向外暴露的内容 14. module.exports = { 15. color: "red" 16. } 17. // 4 18. module.exports = ["red", "green", "blue"]; 19. // 5 20. // 这种方式外部想要使用要作为方法来使用 21. module.exports = function() { 22. console.log("success"); 23. } 24. // 6 25. return 123; 26. // 7 27. return { 28. num: 100, 29. color: "red" 30. } 31. // 8 32. // 这种方式外部想要使用要作为方法来使用 33. return function() { 34. console.log("success"); 35. } 36. }) 37. // 9 38. // define(123); 39. // define("abc"); 40. define(true); 41. // 10 42. define({ 43. color: "red", 44. num: 100 45. }) |

# 模块信息对象

每一个模块都有一个对象用来保存模块的信息

dependencies: 是模块的依赖集合，是一个数组， 数组中的每一项都是集合中的成员

deps: 是根据依赖集合产生的

exports: 定义向外暴露功能的对象

id: 表示模块的id

status: 表示模块的状态

uri: 表示模块的文件路径，默认情况下（没有显示的id）与模块的id是一致的

模块信息对象：

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |