# 09.1

搜索 看到这了

实现原理：css

步骤：

1. 布局
2. 原理
3. 了解js语法

让某个元素消失的方法：

visibility:hidden让元素隐藏占位

display:none显示为无

宽高为0

透明度为0

改变定位值，移出视线

用一个元素盖住

Js书写标签

<script>

</script>

Js中如何获取元素L:

1.通过id名称 document get element by id ’link’书写：document.getElementById(‘link’) 注意大小写单词首字母大写

事件：鼠标事件键盘事件系统事件表单事件自定义事件

鼠标事件onclickonmouseover移入onmouseout移出onmousemoveonmouseuponmousedown ......

如何添加事件：元素.事件

函数：可以理解为命令 function 名字随便起 function abc(){ 发生的事件}函数不会主动执行

让函数执行的方法：1.直接调用：abc();

2.事件调用：元素.事件 =函数名Div.onclick = abc;

3.

学会测试保持随时写随时测试的习惯

alert(1);带一个确定按钮的警告框放在可能错误的行后就知道哪里错误了

alert(‘ok’);单引号双引号都可以字符串

变化样式document.getElementById(‘ul1’).sytle.display = ‘block’;

变量：var name=document.getElementById(‘’)

Js执行时间onload加载完以后执行window.onload = 事情

function (){}匿名函数用直接调用document.getElementById(‘div’).onclick = function (){}

331217227 微信

ECMAScript BOM浏览器对象模型 DOM文档对象模型

外部js <script src=”./hello.js”></script> 里面不能再写代码

标识符的命名规则：由数字、字母、下划线和美元符组成

首字符不能使数字

不能使用js的关键字和保留字

变量： var 声明变量 =代表赋值 赋值运算符，不是等于

如果声明一个变量，没有初始化那么默认值是udefined（未定义）

alert会暂停js代码的执行

向控制台打印内容 console.log()

通过css选择器选择 document.querySelector(‘’) 通过

匿名函数不能单独存在

div.style.cssText 会先清空元素对象之前的所有行间样式，然后添加他自己的新样式

同时声明多个变量的时候可以使用“，”分开 var a = 1 , b =2 , c = 3

github 注册

# 9.6

元素的属性操作 读和写

元素.属性的名字 属性的名字（key） 属性对应的值（value）

可以用点的地方都可以用中括号，可以用中括号的地方不一定能用点

元素[‘属性的名字’] 当不符合命名规范的时候使用中括号

text.focus() 让文本框自动聚焦

class是js的关键字，箱操作元素的class需要使用className

获取属性操作获取的颜色不一定准确

属性的读取，读取的都是行间的样式，如果行间没有设置，那么是获取不到的

获取路径 图片的src属性，获取到的是一个绝对路径

innerHTML 表示元素标签内的html内容 可以读取和修改

innerHTML里面的都是字符串

字符串 ‘’ “” `` length 用来获取字符串的长度

字符串一旦创建就不可以再改变 所以他的length属性只可以读取不可以修改

字符串拼接 用+ 中间没有空格 当做加法运算的时候如果表达式里面偶一个是字符串，那么不做加和操作，而是做字符串的拼接操作。而且在数学运算符号中，只有+是这样的。

innerHTML会先清空之前的内容。然后再添加新的内容

+= 也就是本身+

=左右先运算右侧

用``的话${}也可以拼接 里面可以写变量

元素. sList 使元素的所有的class的一个集合 元素.classList.add(‘class名字’) 添加class

元素.classList.remove(‘class名字’)移除class 不用加空格

元素.classList.contains(‘class名字’)如果元素有这个class 则返回true 否则返还false

# 9.7

比较操作符

==只比较值，而不比较数据类型

=== 严格比较，不仅仅比较值，而且比较类型

>大于

>=大于等于

<小于

<=小于等于

布尔值 true false 仅有这两个值 true代表真 false 代表假

语句控制流程

判断 基本语法

if (条件1){

条件1成立时候的代码(这里面的代码只有当条件成立的时候才会执行)

}

else{

如果条件1不成立时候的代码

}

if (条件1){

条件1成立时候的代码(这里面的代码只有当条件成立的时候才会执行)

}

else if(条件2){

如果条件2成立时候的代码

}

else if(条件3){

如果条件3成立时候的代码

}

…….

else{

上述条件都不成立时候的代码

}

使用流程控制的时候 只要遇到了第一个成立的条件就不会执行后面的条件，即使成立也不执行

new Date().get(Day);获取今天是星期几 获取后是0~6的数字 代表星期日到六

三元操作符 又叫做三元表达式

条件成立吗？成立就执行这句话：条件不成立执行的代码； 如果不想执行就写””

如果只执行一行代码 可以用三元操作符 多行代码不要用

三元表达式一般情况下多用在赋值操作

如果已知条件解决不了当下的问题，首先要想到回过头去声明一个变量

if(开关){

想发生的事

开关 = false

}

else{

想发生的事

开关 = true

}

!取反、非 !false = true 可以多次取反

% 取模 求余数

乘方 a\*\*n n次方

自增自减

自增就是某个数值+1 自减就是某个数值-1

num-- 后自减

--num 前自减

num++ 后自增

++num 前自增

前与后在单独使用的时候没有任何区别

在遇到表达式或者等式的时候 前自减会先执行再赋值 后自减会先赋值在自减

唯一一个不用写等号也能改变值的

# 9.8

函数可以在声明前写也可以在声明后写

数组

声明一个数组 []数组里面每一项用逗号隔开，最后一项后面不能加，

数组里面可以储存任意类型的数据，包括数组本身

数组也是一个对象

var arr = [0,1,2,3,4]

如何使用数组里面的数据 arr[0] 可以通过下标(索引值)的方式去获取数组中对应的数据

索引值：从零开始依次往下排的整数

数组的属性 length，代表数组的长度arr[0]数组的第一个 arr[arr.length - 1] 数组的最后一个

arr.length = 1让数组的长度变成1 数组的length是可以写的

初始化页面操作

默认显示的内容

循环：用来帮你做重复的事（有规律的事，差不多的事）

关键字 for(初始化的条件 var i(j,k) = 0; 条件 i<5; 条件变换 i++){

如果条件成立，那么就执行这里面的代码（循环体）

}

两个；；必不可少

1. 执行初始化条件
2. 看条件是否成立
3. 如果成立那么执行循环体的代码，如果不成立就跳出循环
4. 如果第三部条件成立了，执行完循环体的代码之后。执行条件变化，之后重复

进行大量数据的for循环时候尽量放在内存中进行：声明一个空字符串变量，

循环控制 break中断 continue 跳过此次 配合if判断使用

！= 不严格不等于等于 ！== 严格不等于

# 9.11

当循环嵌套循环的时候，两个循环不能用同一个变量

~~1.2 = 1

任意一个小数前面加两个~等于前面的等数

一维数组

var arr1 = [1,2,3,4];

二维数组（非对称数组和对称数组） 数组中嵌套一个数组 三维等同理

var arr1 = [‘a’,’b’,’c’],[‘d’,’e’][‘1’,’2’,’3’,’’] 非对称

var arr2 = [[1,2,3], [4,5,6]] 对称

# 9.12

获取一组元素

元素对象.querySelectorAll(‘css选择器’); 非动态获取元素

获取到的不是一个数组，这一组元素的lengh属性只能读不能写，称之为类数组

通过元素的标签名获取

元素对象.getElementsByTagName(‘标签名’); 动态获取元素 实时监测

操作一组元素

this:是JS的一个关键字，是预先定义好的这个变量，这个变量里的值是不固定的，会随着执行环境（上下文）的改变而变化

1. 只要this在函数外，也就是全局下，则它代表window
2. 元素.事件 在函数内，this不一定是什么

-元素.事件 = 事件处理函数 那么事件处理函数里的this就是触发这个事件里的元素

自定义属性

元素对象.自定义属性名 = 值

可以利用自定义属性 利用for循环让一组元素的一个自定义属性等于他的索引值，再利用这个自定义属性的值控制其他组元素同索引值的元素。

# 9.13

\*\*选项卡的基本原理

由一组元素控制另外一组元素 首先想到自定义属性 这个自定义属性要控制方去添加而不是被控制的添加

a标签href‘javascript:;’这样点击就不会跳转了

课件发下来看一下flex布局

# 9.15

mvc 思想

m model 模型（数据）

v view 视图（html+css）

c control 控制器（js脚本）

用数据生成内容 用变量代替重复使用的数据

处理用户和浏览器之间的交互

初始化操作

# 9.18

第二章

jS中的数据类型

1. 基本数据类型

Undefined 未定义

Null 空

Number 数字

Boolean 布尔值

String 字符串

Symbol 符号类型

1. 引用数据类型

Object 对象

Dom元素

[]

function

JS里除了基本数据类型，剩下的都是对象。

基本数据类型引用数据类型的区别

基本数据类型是真正的赋值

引用数据类型是复制的引用

引用数据类型

如何区分数据类型

使用 typeof空格要判断的数据typeof 的返回结果都是一个字符串

undefined类型，只有一个值就是undefined 代表未定义

如果变量声明但是没有被初始化 那么储存的就是undefined

如果一个对象的属性名不存在，然后你去访问他，那么就是undefined

一个函数，如果没有返回值，那么执行之后的结果是undefined

Null 类型，只有一个值，就是null 代表空

JS中 数据类型返回‘object’是JS的一个bug

凡是在内存中转换成二进制前三位是0的，使用typeof判断都会返回‘object’，null代表空，他转换成二进制全是0.，所以也返回object

当null和undefined作比较的时候 两个等号的情况下 undefined实际上是派生自null，所以在比较值的时候是相等的

函数类型

函数也是一种数据

函数表达式

var fn1 = function(){};

不能在表达式前面调用

函数声明

function fn2(){};

可以在声明前调用

tyoepf 函数，返回的不是’object’而是function

函数也是对象，但是他是一个可以执行的对象，所以很有必要和其他对象做一个区分

函数表达式的名字

var fn = function ffn(){}

ffn这个函数名字只能在这个函数内部可以使用

fn可以直接使用

判断一个数据是不是null的时候不能使用typeof

对象类型的数据结构又称为字典类型的数据结构，和数组非常类似，但不同于数组的是，数组的key只能是从0开始的数字，而对象数据结构的key可以使任意的字符串。

定义一个对象使用{},对象主要包括 属性和方法，每个属性和方法都由键值对（key和value）组成

自定义一个对象

var obj1 = {};

var obj2 = {

tagName:’div’ 属性名

a:1,

‘abc’:[1,2,3], 符合明明规范也可以加’’

‘b-2’:’str’, 不符合命名规范要加’’

fn1:function(){},

fn2(){alert(2)} 简写形式

};

console.log(obj2[‘b-2’]);

修改

obj2.a = 10;

调用

obj2.fn1();

obj2.fn2();

数组就是一种特殊格式的对象

JS里面进行数据类型转换的时候只能转换成以下三种

Number（数字）

String（字符串）

Boolean（布尔值）

数字类型

整数（正整数和负整数）

浮点数（就是小数）

NaN：Not a number 不是一个数字的数字类型

正无穷(Infinity)和负无穷(-Infinity)

最小值:5\*e的-324次方

最大值1.7976931348623157e的308次方

小数

var n1 = 1.2; 小数

var n2 = 2.0; 整数 JS自动变为整数 因为占用内存小

小数计算并不精确

可以使用数字（小数）.toFixed（要保留的小数位数）这个方法会四舍五入

整数需要些两个点 或者用括号包起来

NaN在平时使用的时候，不会主动的去用，但是一旦某个结果是NaN 了，那么代码有误，说明计算失败了

NaN任何数值都不相等，包括自己

NaN和任何数值进行计算，结果都是NaN

数据类型转换：转数字

1.Number(要转换的数据) 是一种系统内置的函数

2parseInt(要转换的数据[,如果是数字可以指定进制，默认是十进制])

1. parseFloat(要转换的数据)

只要字符串里面有一个不是数字那么就无法转换

parse的两种方法：

空数组转换成数字 是NaN

空字符串转换成数字是NaN

parseInt parse 解析 Int 整数

parseInt(对象) 都是NaN

转换的时候只会保留小数点之前的部分 类似于~~ >>0

parseFloat

无法指定进制

小数点后面的也会保留

会从第一个不是数字的位置截止转换

判断是不是NaN

Number.isNaN（）

总结：

数据类型 两大类 7种

基本数据类型和引用数据类型的区别

基本数据类型是按值传递的

引用数据类型是按引用传递的

数据类型转换要么转换成数字 要么转换成布尔值 要么转换成字符串

NaN

Number parseInt parseFloat

isNaN Number.isNaN

数字里面的0在做判断的时候是false，除了0以外其他的数字都是true

数字类型里面，只有 NaN和0在做判断的时候是false

多条件判断

并且&& 都为真 结果为真

或者|| 只要其中一个为真 结果为真

在赋值的时候 如果单独使用的情况下 如果使用或者 JS会在碰到假命题时候继续向后找，直到第一个真命题为止

如果使用并且 JS会在碰到真命题之后继续向后找

# 9.19

字符串：由0个或者多个字符组成的集合，并且放到一对单引号或者双引号或者反引号中的。

属性：length 代表字符串的长度。length属性只可以读取，不可以修改

字符串一旦创建不可以再修改

字符串的转化，将其他数据类型转换成字符串的方法：

1. String（要转换的数据） JS系统提供的内置函数

所有的对象转换的结果 都是 [object Object]

1. 要转换的数据.toString（）

undefined 和null 不能使用这个方法

像这样直接去调用某个方法进行转换的称为显示类型转换

隐式数据类型转换，是JS引擎帮你自动做的事情。

console.log(‘a’+1); ‘a1’

console.log(‘1’\*2); 2

console.log(‘abc’ \* 1) NaN

console.log(‘123’ \* 1) 123 想让字符串转换成数字直接乘1 最快的把数字字符串转换成为数字

console.log(100+’’) ‘100’ 最快的把数字转换成字符串的方法

布尔值

true 真

false 假

转换成布尔值的方法：

Boolean(要转换的数据)

数字里面除了0和NaN其他的转布尔值都是true

undefined和null转布尔值是false

‘’false ‘ ’ true 除了空字符串转布尔值是假的，其他的都是真的

所有的对象都是真的

最快的转布尔值的方法：!!数据

console.log([] == ![]);//true

空数组首先取反得false 左边的空数组优先向数字类型转换 得0 0代表false 所以false == false 结果是true

逻辑与 &&

逻辑或 ||

逻辑非 ！ 返回值一定是一个布尔值

在赋值的时候

逻辑与 会一直向后找，直到找到第一个转布尔值为false的值，如果没有就是最后一个值

逻辑或 会一直向后找，直到找到第一个转布尔值为true的值，如果没有就返回最后一个

隐式数据类型转换

类数组

、

作业要求： 首字符不能是0

五位以上 十位以内

不能是字母

成功则qq通过

# 9.20

自动选中文本框所有内容

text.select(); //选中所有文字

函数：可以对任意多条语句进行封装，代表一个功能的整体。

函数的分类:

函数声明

函数表达式

匿名函数

函数是一种特殊格式的对象，所以函数也是一种数据

一个完整函数的定义

关键字 function fn函数名（参数1，参数2……）{代码1 代码2 return；（返回值）}

函数表达式

var fn =function (){}

fn(); //必须在声明之后调用这个函数

//函数表达式 （匿名函数）

(function (){}); //用小括号包起来 是函数表达式；要其自执行在括号后面再加一个小括号

也可以在前面用！和~

建议每个函数都要有一个名字

function add(a,b){

//a b 叫做形参 ，形式上的参数，可以有可无

}

add(1,2); ab叫做实参，就是实际传入函数的参数，形参和实参是一一对应的。 函数调用多次互不影响

arguments:函数内部存在的一个对象，是一个类数组，他的作用是：保存实参的集合

return返回值 默认undefined return空格一个字符串

1. 作为函数的返回值，后面是什么就返回什么；
2. 凡是return后面的代码，都不会再继续执行

# 9.21

整理昨天的添加到第一个和最后一个函数 和今天的删除第一个和最后一个函数

向数组最后面添加数据

|  |
| --- |
| function push(arr){ |
|  | var arg = arguments; |
|  | for(var i=1; i<arg.length; i++){ |
|  | arr[arr.length] = arg[i]; |
|  | } |
|  | return arr.length; |
|  | } |

向数组最前面添加数据

|  |
| --- |
| function unshift(arr){ |
|  | var arg = arguments, len = arg.length; |
|  |  |
|  | for(var j=1; j<len; j++){ |
|  | for(var i=arr.length; i>0; i--){ |
|  | arr[i] = arr[i-1]; |
|  | } |
|  | arr[0] = arg[j]; |
|  | } |
|  |  |
|  | return arr.length; |
|  | } |

浅拷贝：是对引用数据类型进行拷贝操作，但是只拷贝第一层，如果这个引用数据类型里面还有引用数据类型，那么不对里面这个引用数据类型进行拷贝。

# 9.22

作用域和作用域链

浏览器运行JS脚本的过程：至少包含三个部分

1. 编译器：负责把JS代码编译成计算机可以识别的数据（语法分析和词法分析） 不会对函数进行编译，对函数作用域进行编译
2. JS引擎：负责对编译后的代码进行逐行运行
3. 作用域：用来管理标识符的一套规则，它规定了每个标识符可以在什么范围内被访问和修改

一个标识符 同时作为函数和变量的名字 优先变量 后面的覆盖前面的

作用域：

1. 全局作用域：在script标签内，并且在script标签内的函数之外的部分
2. （通常指函数作用域）局部作用域：凡是在函数内部的都属于函数作用域

一旦进入一个新的作用域的时候 首先要进行域解析

作用域链 规定标识符的查找顺序，只能由内向外查找，而不能由外向内查找

函数内部声明的变量，在函数外部是访问不到的

就近原则 自己里面有，先用自己的

闭包 函数就是闭包，但是通常是指函数嵌套函数。当一个函数可以记住它被声明的作用域，并且可以访问那个作用域里面的所有的标识符，就会形成闭包

栈 LIFO 后进先出 后进来的先出去，凡是储存在栈中的变量都会被操作系统自动释放。在JS当中，通常储存局部变量。

堆 是一种树状的数据结构，在JS中，引用数据类型，通常会储存在堆中。堆中的数据，不会被自动释放

静态区 在JS当中，全局变量都存放在这个区域。不会被垃圾回收器回收

垃圾回收器：对不再引用的数据进行回收（释放内存空间）。

# 9.25

获取计算后的样式(浏览器实际显示元素的样式)

getComputedStyle()

例子 getComputedStyle(box).width

获取transform 是一个矩阵

for…in循环是用来遍历对象的

for(var key in obj){

key 代表这个obj对象的每个属性

对象每个数据对应的值 obj[key]

}

in 操作符 可以知道某个属性是否在这个对象当中

switch 用于多个条件判断，使用case语句进行配合 每个case语句之后需要一个break来终止后续条件判断，如果不写，那么凡是条件成立的语句都会执行，直到遇见一个break为止。如果每个case都不成立 则运行defult

case后面使用的是 严格比较

swich(){

case 0:

alert(‘星期日’);

case 1:

}

while循环 凡是终止条件不确定的时候，多用while循环

while(i<5){console.log(i++)}

函数的递归

递归一定要有一个终止的条件

var a =0;

function(i){

console.log(i);

if(i<3){

fn(i++);

}

}

fn(a);

循环搞不定 递归一定能搞定

# 9.26

定时器：两大类

重复性定时器：以指定的时间间隔，做同一件事，只要不停止，一直做下去

延迟性定时：在指定的时间间隔后，做一件事，只做一次

重复型

setInterval(fn,delay,arg1,arg2…)

-fn:要做的事

-delay:时间间隔(单位:ms)

-arg1,arg2…:如果fn有参数，那么这些代表fn的参数

-返回值:当前定时器的编号（不同的浏览器这个编号是不一样的）

拓展 setInterval(‘fn(1)’,1000)

延迟型

setTimeout(fn,delay.arg1,arg2)

参数和setInterval是一样的

返回值也是当前这个定时器的编号

清除定时器

clearInterva(要清除的定时器编号)

clearTimeout(要清除的定时器编号)

可以通用

这两个方法的参数，不论你给他什么类型，都不会报错

html5的最新规范，规定了定时器的最短时间间隔是4ms

css给一些元素之间加空隙

元素 + 元素

# 9.27

时间对象：JS里的时间对象是基于1970年1月1日00:00:00（格林威治时间）的。

创建时间对象：new Date();

单独使用new Date()那么代表当前你的计算机的系统时间

这个时间，获取的是这行代码执行时候的时间，不会自动更新

var time = new Date();

时间对象分解

time.getFullYear();获取年份

time.getMonth();获取月份,月份的范围是0~11 0代表一月 11代表12月，所以得到当前的月份，需要+1

time.getDate();获取日

time.getDay();获取星期 0代表日

time.getHours();获取小时

time.getMinutes();获取分钟

time.getSeconds();获取秒钟

time.getMilliseconds();获取毫秒 (0~999)

\*时分秒要加s

time.getTime(); timestamp 时间戳，获取的是1970年1月1日00:00:00到现在的毫秒数

Date.now() 快速获取时间戳的方法

设定日期对象

var time = new Date(‘2016 08 24 13:00:00’);

var time = new Date(2018,1,26,13,13,13); 2018年二月26日13点13分13秒

如果设置了不存在的日期，那么会自动跳转

单独设置时间的某个值

setFullYear(); 可以传年月日

setMonth(); 可以传月日

setDate();

没有设置周

setHours(); 可以传时分秒

setMiniutes(); 可以传时分

setSeconds(); 可以传秒

set后就变成时间戳了

# 9.28

如何获取一个元素到他定位父级的距离

ele.offsetLeft 获取Left值

ele.offsetTop 获取Top值

上面的方法不需要元素一定是定位的

获取元素的大小

ele.offsetWidth ele.offsetHeight

获取元素包括边框以里的大小

ele.clientWidth ele.clientHeight

获取元素边框以内（不包括边框的）大小

ele.getBoundingClientRect

获取元素到屏幕左侧和上侧的绝对位置

获取元素的offsetWidth和offsetHeight

获取到的是包括边框的 里面的bottom是从包括边框的最下面到浏览器最上方的距离 其他方向同理

动画帧函数：  
 window.requestAnimationFrame();

window.cancelAnimationFrame();

他们的用法和setTimeout是非常相似的

example：

var timer;

function fn(){

timer = window.requestAnimationFrame(fn);

console.log(1);

}

fn();

document.onclick = function(){

window.cancelAnimationFrame(timer);

}

让一个物体运动必须要知道的几个量

time：已经运动的时间

begin:起始位置

change:总路程

duration:总时间

匀速运动 小： current= begin+ (change/duration\*time)

定时器管理：就是在开启定时器之前，先清除定时器

# 9.29

回调函数：就是当做参数传入到另外一个函数的函数

this:JS的关键字，在不同执行环境下代表不同的指向。

全局作用域下：this就是window

函数作用域下：

元素.事件 this就是触发这个事件的元素对象

对象.函数(); 某个对象调用自身的某个方法的时候，那么这个函数内的this就是调用这个函数的那个对象。

函数直接调用，那么相当于window对象去调用这个函数，所以this就是window

定时器里的所调用的函数，函数内部this是window

下面三个方法，可以修改函数内部的this指向

fn.call(要指向的对象，参数1，参数2，…)

fn.apply(要指向的对象，[参数1，参数2，…]) 数组或者类数组

call 和 apply的作用是一模一样的，只是参数有区别

fn.bind(要指向的对象，参数1，参数2，…)

bind方法不会立即执行当前这个函数，而是返回的是一个新的函数，并且这个新的函数里面的this永远指向调用bind时候指向的对象

props 配置参数 是一个对象

# 10.9

## 复习：

变量：

调试：debugger; Call Stack 栈 Scope 作用域

onselectstart=”return false;” 禁止选中

# 10.10

抖动函数

# 10.11

将时间对象转成字符串 不能用toString 因为里面会有中文

new Date().toUTCString

new Date().toGMTString

命名空间

# 10.12

字符串的方法

所有的字符串操作，都不会修改原来的字符串。

原因：字符串一旦创建就不可以再被修改。

str.indexOf(查找的字符串片段，从第几个字符开始查) 查找位置 如果有多个一样的，默认只找到第一个。 如果第一个参数的是一个空字符串，第二个参数>=length 返回的是length

str.charCodeAt() 返回对应位置字符的Unicode编码

String.fromCharCode(num1,….numN)

该方法返回使用指定的Unicode值序列创建的字符串

str.repeat(count)重复字符串 count为0到正无穷之间

str.substr(start,length) 返回一个字符串中从指定位置开始到指定字符串长度，不传长度就到最后

如果start是一个负值，那么就被看作strLength+start当做起始位置 如果加完依旧是负值 则相当于从0开始，返回整个字符串

str.substring(indexStart,indexEnd) 截取到结束位置之前的 截取类的都是起始位置包括，结束位置不包括

如果起始位置大于结束位置，那么会自动颠倒起始位置和结束位置

str.slice(startInndex,endIndex)

str.split(‘想变成分隔符的字符串片段’)分割

传不存在的字符则还是原来的字符串

传空字符串则完全分割

分割左或右没东西了则分割出一个空字符串

最快的把字符串转换成数组的方法[…str]

str.toLowerCase()将字符串转为小写形式，并返回

str.toUpperCase()将字符串转为大写形式，并返回

str.trim()从字符串两端删除空白字符

str.trimLeft()清除左侧空格

str.trimRight()清除右侧空格

# 10.13

数组的方法

创建数组

字面量表示法: var arr = [];

实例化数组:var arr = new Array(); 当new Array()的参数只有一个，并且这个参数是一个数字的时候，那么代表创建一个长度为这个数字的空数组

Array.isArray(数据)可以判断这个数据是不是数组，如果是就返回true，否则就返回false

Array.from(类数组)把类数组转换成数组，可以用这个方法操作一组元素，凡是类数组都能转换

arr.indexOf() 返回数组中可以找到一个给定元素的第一个索引，如果不存在则返回-1

arr.push() 将一个或多个元素添加到数组的末尾，并返回数组的新长度

arr.pop() 删除数组最后一个元素，并返回该元素的值，此方法更改数组的长度 没有参数

arr.shift()删除数组第一个元素，并且返回该元素 值，此方法更改数组的长度 没有参数

arr.unshift()将一个或多个元素添加到数组的开头，并返回新数组的长度

arr.splice(start, count)删除指定位置开始 删除个数的元素 返回删除的元素组成的数组 只传一个参数 则一直删除到结尾

arr.splice(start,deletcount,item1,item2)删除指定位置开始 删除个数的元素 并将新元素添加到删除的位置 返回删除的元素组成的数组

arr.slice(start，end) 返回一个从开始到结束 不包括结束 选择的数组的一部分浅拷贝到一个新数组对象，原数组不会被修改

arr.forEach(cb(){}[,thisArg])自动循环 cb有三个参数(item,index,self)元素的值 元素的索引 数组本身，还可以设定这个函数的this值，不设置为window 没有返回值

arr.map(cb(){}[,thisArg])和forEach很像 但是有返回值 返回值为每个cb的返回值组成的数组

arr.filter(cb(){}[,thisArg])和map很像， 返回的是符合条件的元素组成的数组

arr.every(cb(){}[,thisArg]) 判断是不是所有元素都符合条件 是就返回true 不是就返回false

arr.some(cb(){}[,thisArg]) 判断是不是有元素符合条件 有至少一个就返回true 一个都没有就返回false

arr.find(cb(){}[,thisArg]) 找到符合回调函数返回值条件的那一项

arr.findIndex(cb(){}[,thisArg])找到符合回调函数返回值条件的那一项的索引值

arr.join()将一个数组转成字符串 参数是想用的分隔符

arr.reverse() 反转数组 会修改原数组

arr.sort() 在适当的位置对数组的元素进行排序，并返回数组，sort 排序不一定是稳定的，按照unicode码点排列 可以传参一个函数 返回值为条件

随机数 Math.random();是JS的随机数，返回的是[0,1)之间的随机小数

arr.fill(要替换成的值，起始，结束) 用一个固定值填充一个数组中从起始索引到中之索引内的全部元素

arr.concat(arr1,arr2,arr3…)将arr和后面参数里面的数组拼接起来，返回拼接后的数组

更简单的拼接方法[…arr,…arr1,…arr2]扩展运算符

# 10.16

看到这了

Math对象

包含了数学中的常熟和一些属性与方法

Math.abs(x); 求X这个数值的绝对值

Math.PI;代表圆周率，也就是圆的周长和直径的比值 代表弧度的角度单位

1角度等于Math.PI/180的弧度

三角函数

Math.sin(弧度) 正弦值

Math.cos(弧度) 余弦值

Math.tan(弧度) 正切值

反三角函数

Math.asin

Math.acos

\*\*Math.atan2(y,x) 得到一个弧度

高级坐标旋转公式

X\*cos(vr) + y \*sin(vr)

y\*cos(vr) -x \*sin(vr)

向上取整(往大了取)

Math.ceil(num)

向下取整(往小了取)

Math.floor(num)

四舍五入

Math.round(num)

最大值

Math.max(num,num…….) 如果是数组 Math.max(…arr) 或者用apply(null,arr)

最小值

Math.min(num,num…….) 如果是数组 Math.min(…arr) 或者用apply(null,arr)

随机数

Math.random();返回的是[0,1)之间的随机小数

万能公式 rp() 求任意范围之间的随机数 还是整数

Math.random()\*(最大值-最小值)+最小值

求次幂

Math.pow(x,y) 返回X的y次幂

求开方

Math.sqrt(num) 求num的开二次方

布局转换

就算当前元素不是定位元素，他有定位父级，他其实也有定位值 就是他的位置

# 10.17

冒泡排序

[5,4,1,7,6]

数组的解构赋值

快速排序

-选取一个基点（base point） 通常都是数组中中间的那个数，并且将这个数拿出来

-将数组中剩下的数字，分别的和这个基点进行比较，如果小放左边，如果大放右边

-对放到做左侧和右侧的两组数字，重复上面的两个步骤

# 10.19

BOM 浏览器对象模型 刷新按钮 主页 地址栏 前进后退 滚动条

DOM 文档对象模型 针对XML的扩展，就是为JS操作页面元素提供了一套应用程序编程接口（API）

DOM把整个文档映射成一个树状结构，它里面的所有东西，都是一个节点。

API 应用程序编程接口

节点至少拥有 nodeType、nodeName和nodeValue三个基本属性，节点类型不同，属性的值也不同

节点类型：12种 看课件

记住 1元素节点 2属性节点 3文本节点 8注释节点 9文档节点

注释节点、文本节点有值 元素节点没有

通过ele.childNodes可以获取第一层的所有子节点的集合

通过ele.attributes可以拿到ele元素属性节点的集合

ele.parentNode 可以获取到ele的父节点

ele.offsetParent 可以获取到ele的定位的父节点

ele.children 可以获取到ele的第一层的所有得元素子节点 并且是动态获取

ele.previousSibling 获取ele的上一个兄弟节点

ele.nextSibling 获取ele的下一个兄弟节点

ele.previousElementSibling 获取ele的上一个元素兄弟节点

ele.nextElementSibling 获取ele的下一个元素兄弟节点

toggle 如果没有这个class就添加 如果有就删除，如果添加了这个class 那么就返回true 否则就是false

ele.firstElementChild 获取到ele的第一个元素子节点

ele.lastElementChild 获取到ele的最后一个元素子节点

# 10.20

创建和插入节点

document.createElement(标签的名字tag 字符串)；可以创建一个DOM元素

parent.appendChild(ele);把ele这个元素对象插入到parent这个父容器的最后 做的是剪切操作 返回值就是插入的这个元素

parent.insertBefore(new,old);把new这个元素放到父容器为parent的old元素的前面 不会影响之前元素 做的是剪切操作 返回值就是插入的这个元素

删除节点

parent.removeChild(ele);删除parent下面的ele元素 返回值就是被删除的元素

克隆节点

ele.cloneNode([deep]);克隆ele这个节点，如果传了deep参数为true那么就是深克隆(里面的节点也克隆) 事件是不会克隆的

替换节点

parent.replaceChild(new,old)吧parent下的old节点替换成new节点

# 10.23

看到这了

DOM操作表格

table.tHead

table.tFoot

table.tBody 获取的是一个数组 表格体

table.tBody[0].rows[0].cells[0]

table.creatTHead()创建并插入一个thead 同样适用于创建tfoot tbody

tbody.insertRow（pos）向tbody指定位置添加一行tr并返回这行 默认插到最后面

insertCell()插入一列

document.forms 获取到页面上所有的表单

forms.someInputName 获取到对应name的表单内部元素 直接使用里面的name就可以

DOM操作补充

快速的获取body document.body

快速的获取html document.documentElement

快速的获取title document.title（可读可写）

HTML5自定义属性

data-XXX =””

JS 中 使用ele.dataset可以获取到所有自定义属性集合 如果date-后面是一个驼峰命名的，那么会把所有的字母都变成小写 如果想使用驼峰命名，比如divEle 需要写成data-div-ele

ele.getAttribute(‘属性名’) 可以拿到元素身上属性对应的值，而且这个属性名对应的值怎么写的，获取到的就是怎么样的 可以用于获取src的相对路径版本

ele.setAttribute(‘属性名’,’属性值’) 可以设置

如果insetBeforeChild的第二个参数是null，那么相当于appendChild

# 10.24

BOM browser Object Model 浏览器对象模型 他的核心是window 对象，同时window又是JS的顶层对象

1. alert() 弹出带确定按妞的提示框
2. prompt()带输入和确定按妞的弹框 返回值是输入的内容
3. confirm() 带确定和取消的提示框 返回值是true 和false

用户代理信息 weindow.navigator.userAgent

window.location

一个完整的url地址

<https://www.XXX.com:80/a/b/c>?a=1&b=2&c=3#x=123

-https 称为协议 protocol

-www.XXX.com:80 称为主机名 host

-www.XXX.com 称为hostname

-a/b/c 称为路径

-?a=1&b=2&c=3 称为queryString查询字符串 修改queryString 会刷新浏览器

-#x=123 称为hash 不会刷新页面

window.location.href 跳转页面 可以获取到当前的url地址 也可以设置当前的url地址

window.location.hash 可以获取到#之后的内容 包括#

当hash改变的时候会触发onhashchange时间

window.location.search 可以获取到查询字符串，从?到#之间的内容，包括?不包括 #

window.history 历史信息 主要包括浏览器的前进后退等方法

window.history.back() 跳转到上一帧

window.history.forward()跳转到下一帧

window.history.pushState(null,’abc’,’https://www.baidu.com/a/b/c’)

# 10.25

Screen对象 window.screen 获取屏幕分辨率

窗口的大小 window.innerWidth/innerHeight 用于获取浏览器文档区的宽高

DOM想获取文档的宽高 使用：

document.documentElement.offsetWidth/offsetHeight10.26

事件：事件是文档或者浏览器窗口中发生的特定的交互瞬间

事件名：click mouseover……

事件处理函数：当某个事件发生的时候，执行的对应的函数称为事件处理函数，如果一个元素发生了某件事，但是没有这件事对应的事件处理函数，那么这种情况称为事件流失

事件流：事件流描述的是元素对象从页面中接受事件的顺序。事件发生时会在元素节点与根节点之间按照特定的顺序进行传播，路径所经过的所有节点都会接收到该事件，这个传播过程即DOM事件流。

-事件冒泡：事件的传播是从最具体的事件目标到最不具体的事件目标。即从DOM树的叶子到根

-事件捕获：事件的传播是从最不特定的事件目标到最特定的事件目标。即从DOM树的根到叶子

标准事件流 三个阶段 事件捕获 处于目标阶段（可以包含于冒泡阶段） 事件冒泡阶段

on+事件名都是事件冒泡 DOM一级事件 不能为同一个事件名添加多个事件处理函数

DOM二级事件 采用事件监听的形式 语法 ele.addEventListener(‘事件名’,事件处理函数,是否捕获) 是否捕获：默认值是false

ele.removeEventListener(‘事件名’,事件处理函数,是否捕获) 一定要和添加的时间处理函数一一对应

目标的触发函数顺序不一样那么执行的顺序也不一样

事件对象 在触发DOM上的某个事件时，会产生一个事件对象event。这个对象中包含着所有与事件相关的信息。包含导致事件的元素，事件的类型以及其他与特定事件相关的信息

事件对象是事件处理函数的第一个参数，不同的事件，事件对象里面的属性是不一样的

阻止事件冒泡 e.stopPropagation() 写在事件处理函数的最前面 最通用

在on事件的时候 可以在里面添加一个e.cancelBubble() 也能取消冒泡

事件委托:就是当需要给某个元素下的某个子节点添加事件的时候，由于子节点触发事件的时候会发生冒泡，冒泡的父节点，所以当需要给多个子元素添加相同的事件的时候，可以把这个事件丢给父元素来做，通过事件源（target）来判断触发事件的对象是什么，然后执行制定好的事件处理函数即可

# 10.27

事件 1.load 当加载完成时候触发

2.scroll 当滚动条滚动的时候触发 按照频率

3.resize 当浏览器窗口大小发生改变的时候触发 按照频率

谷歌浏览器下 滚动距离是body的

document.body.scrollTop/scrollLeft

火狐和IE是html的 (现在都是这个了)

document.documentElement.scrollTop/scrollLeft

mouseover 移入

mouseout 移出

mouseenter 移入 (不冒泡)

mouseleave 移出 (不冒泡)

mousemove 鼠标抚摸事件

click 触发的前提条件是 必须在同一个对象身上按下和抬起鼠标之后才会触发

dblclick 鼠标双击事件 尽量避免在同一个元素身上同时使用单击和双击事件

mousedown 鼠标按下

mouseup 鼠标抬起

获取鼠标的坐标 需要用到事件对象的属性

e.clientX/e.clientY 获取的是不包括滚动条距离的

e.pageX e.pageY 获取的是包括

浏览器默认行为

DOM1级事件 可以使用return false；

DOM1级事件和2级事件都可以使用e.preventDefault(); 和冒泡都是尽量写在处理函数的最开始

contexmenu 右键菜单

表单事件

focus 聚焦 浏览器窗口也可以聚焦失焦

blur 失焦

select 选中文本时

change 一次检测 （失焦时）

input 实时检测 每次修改和失焦

上面两个监听表单变化的事件，只对用户输入有效，如果是js去改变的，是无效的

键盘事件： 注意 监听window

keydown 键盘按下

keyup 键盘抬起

keypress 键盘按下 不包括功能键 注意回车不是功能键

# 10.30

拖拽

基本原理

先清除默认行为

使用变量控制拖拽

碰撞检测

九宫格法

# 11.1

鼠标滚轮事件 有兼容性问题 if(window.onmousewhell ===null){

window.addEventListener(‘mousewhell’,fn)

}else{

window.addEventListener(‘DOMMouseScroll’,fn)

}

在火狐浏览器下，使用鼠标滚轮事件，必须使用DOM2级事件监听形式，监听‘DOMMouseScroll’

e.detail 检测向上还是向下 向上负数，向下正数

在谷歌、IE其他浏览器使用mousewhell事件

e.wheelDelta 检测

# 11.2

webpack 基础实用教程

jS算法与数据结构系列视频

react最佳入门 第十个开始.

# 11.3

块作用域： {这里面就代表块作用域}

let 声明变量 用来声明一个变量，并且这个变量只在其所在的代码块内有效 变量不会被提升 不能重复声明已经声明过的变量 会绑定其所在的块级作用域

可以直接在for里使用 就不用闭包或者自定义属性来记录索引值了

const 可以声明一个常量，常量就是不可以改变的量 特性和let一模一样 const必须进行初始化（赋值）

数组的解构赋值

let [a=1,b=2] =[3] a=3 b=2

只有undefined会不改变赋值

胖子函数 (箭头函数)

var fn = function(a,b){return a-b}

var fn = (a,b)=>a-b;

var fn = function(a,b){

a=b+1;

return a/b

}

var fn =(a,b) =>{

a=b+1;

return a/b;

}

胖子函数没有this 没有arguments 如果用的话会找上一层作用域的 胖子函数通常用于回调函数里面

# 11.6

数据格式 fileData ={

‘0’:{

id:0,

name:’根目录’

},

‘1’:{

id:1,

pId:0,

name:’电影’

},

‘2’:{

id:2,

pId:1,

name:’战狼二’

path:”a/b/c.avi”

}

}

根据id获取指定的数据

function getItemById(db,id){

return db[id];

}

根据id获取当前层级的所有子文件

function getChildrenById(db,id){

}

根据指定的id找到当前这个文件以及它的所有的父级

function getAllParent(db,id){

return

}

# 11.8

判断是否选中

提示是否确认删除（动态生成 做一个函数 返回值就是生成的东西）

删除东西的时候要删除数据 还有checkedbuffer里也要删除 length也要—

用delete 删除对象里的数据

# 11.9

新建文件夹 注意全选框

Object.assign(空对象，想拷贝的) 浅拷贝

# 11.13

JSON JavaScript Object Notation js对象标记 是一种轻量级的数据交换格式

实际上就是一种字符串格式的对象

JSON对象和JSON字符串可以相互转换

JSON.parse(json数据) 可以将json字符转化为对象

JSON.stringify(对象数据) 可以将普通的对象转换成json字符串

在深拷贝的时候 如果对象里没有函数 最简单的方法就是先转换成json 在转回

注意事项

-JSON里面的key必须用双引号包起来

-JSON里面的数据不能有函数

const json = `{

“a”:1,

“b”:”2”

}`

里面的键值对名称必须都加上双引号 不能使用单引号

localStorage 本地存储

* localStorage.setItem(‘key’,’value’) 储存数据 key只能是字符串 value只能是字符串或者数字
* localStorage.getItem(key) 读取数据
* localStorage.removeItem(key) 删除数据
* localStorage.clear 清空数据

# 11.14

正则表达式 使用单个字符串来描述·匹配一系列符合某个语句规则的字符串（按照某种 规则 去匹配符合条件的字符串）

1. /abc/ 正则的字面量表示法
2. new RegExp(arg1规则,arg2修饰符) 实例化表示法 修饰符用字符串表示

str.replace(reg,’’) 替换 规则和要替换成的东西

修饰符

g:global 全文搜索，不添加，搜索到第一个就会停止

i:ignore case 略大小写，默认区分大小写

m:multiple lines 多行搜索

正则表达式由两种基本字符类型组成

* 原义文本字符
* 元字符：在正则表达式中有着特殊含义的字符

\* + & ^ . | \ [] {} ()

类[] 字符类 匹配某类具有某种特征类的字符，可以使用[]来构建一个简单的类

例如：[abc]代表匹配 a或b或c [^abc]代表匹配非a非b非c的自他字符 可以连用几个 来匹配多个字符 例如 [abc][123] 匹配a1a2a3 b1b2b3 c1c2c3

范围类 用来匹配某个区间的所有字符

例如：[a-z]代表匹配a到z的26个小写字符 [0-9]代表匹配0-9之间的任意数字 [^a-z0-9] 可以这样写

预定义类 .代表除了回车和换行符之外的任意字符

\d 代表数字

\D 代表非数字

\s 代表空白字符

\S 代表非空白字符

\w 单词字符(字母，数字，下划线)

\W:非单词字符

^ 以xxx开始

$ 以xxx结束

\b 单词边界

\B 非单词边界

量词

?代表出现0次或者1次最多1次

+ 出现1次或多次 至少一次

\*出现任意次

{n}出现n次

{n,m}出现n到m次

{n,}出现至少n次的

# 11.15

贪婪模式和非贪婪模式，默认是贪婪模式

转非贪婪模式 在量词后面写一个?

‘123456’.replace(/\d{2,4}?/,’’)

分组：正则中可以使用小括号进行分组

匹配子项 使用了分组之后，从左向右看，每一个小括号代表一个子项，顺序是依次排开的

$1 $2 $3……

replace 的高级用法

‘a1b2c3’.replace(/(\w)(\d)/g,(match,$1,$2,index,str) =>{return $1 + (parseInt($2)+1)})

match a1 b2 c3

$1 1 2 3

index 0 2 4

search 搜索

str.search(//) 看一下课件

str.match(reg) 1.reg没有全局修饰符

-返回一个数组 数组第一个是匹配的内容 后面的是匹配的子项 数组里index属性是匹配的位置 input属性是原字符串

2.reg有全局修饰符 直接返回一个数组 就是所有匹配的内容

3.如果没有匹配的内容 那么返回的是null

以下只可以读不可以写

正则对象的属性 global 默认是false

ignore case 默认是false

multeline 默认是false

lastIndex 默认是0 代表当前匹配的结果的下一个位置，并且如果是全局匹配，那么下次会从lastIndex的位置开始查找 所以使用test时候不要加g

source 代表原正则表达式的字符串

reg.test(‘str’)用来测试某个字符串 如果通过则返回true 否则false

匹配中文 [\u4e00-\u9fa5]

# 11.16

\1表示重复正则第一个圆括号内匹配到的内容

\2表示重复正则第二个圆括号内匹配到的内容

前瞻表达式

(?=exp)正向前瞻

(?!exp)匹配后面不满足表达式exp的位置

# 11.20

面向对象(oop)是一种编程思想，对象都是通过实例化类得到的。对象是对客观事物的抽象，类是对对象的抽象。

对象 = new 类();

对象是通过实例化类得到的，实例化的过程就是new XXX 的过程

封装：不用关心某个类的内部实现，只需要去使用它

继承：通常是类和类之间的行为，子类继承与父类

多态：继承之后的子类，可以有自己独特的属性和行为

定义一个类 使用class类的名字{}; 只是定义一个类 这个类里什么都没有

类的名字通常都是首字符大写的

class Ball {};

const ball = new Ball(); 这样去实例化一个类，只会得到一个空对象。

constructor 构造器(构造函数)

class Persion{

constructor(){ 只能使用简写的形式 代表Persion类的构造函数 这里可以定义这个类的属性（通常只是属性） 会自动调用 这里面的this带代表实例化出来的对象 如果这里不写return 默认返回的是实例化出来的这个对象，如果写了return，如果返回的是一个额基本的数据类型，那么会忽略，依然返回当前实例化的这个对象，如果return后面是一个对象，那么就返回这个对象，而不是实例化出来的对象了。

}

}

class Person{

constructor(name,age){ 这里写属性

this.name = name;

this.age = age;

} 这里不要加逗号

这里写方法

say(){ 凡是这个类所拥有的方法，内部的this默认都指向实例化出来的对象

let {name,age} = this;

console.log(`My name is ${name},my age is ${age}`)

}

}

定义类的表达式：

const Person = class{}; // ==> class Person{}

const Person = class Per{} Person 才是名字 Per不是

静态方法

使用static 来定义静态方法，静态方法是类的方法，而不是对象的方法

class Person {

staic sayClass(){ 静态方法里的this都是类 不是实例化的对象

}

}

继承

使用 extends 父类{}就可以继承了

class Abc extends Person{

constructor(){

super(name,age); 这个方法必须调用，否则会报错 只有调用了super方法之后，才能使用this，否则子类没有自己的this super不能在自定义的方法中去当成函数调用，但是可以代表继承的那个父类，然后调用父类具有的方法

}

};

# 11.21

基于原型的系统 Object.prototype 对象原型

Function.prototype 函数原型

obj.\_\_proto\_\_ 可以找到这个对象的原型

每个原型也是一个对象，在原型身上有一个constructor属性，用来表明这个原型的构造函数是谁

new Object() 相当于得到一个{}那么这个Object实际上就是一个构造函数。通过这个例子，可以发现构造函数的特征:

使用new去调用的函数

函数的名字的首字符是大写的(只是一个约定)

不写返回值 默认返回实例化的对象 函数里面的this默认指向实例化的对象

通过new去调用一个构造函数，称之为对象的实例化。

除了JS自带的构造函数，自己也可以去定义自己的构造函数。就像使用class去定义一个类一样

枚举 能否用for in 循环得到

每次实例化构造函数的时候 都会得到一个新的方法（浪费内存） 解决方法 把方法放到原型身上

原型链 凡是出现在原型链上的方法都可以共用

对象.hasOwnProperty(‘属性’)，用来判断这个属性是不是这个对象自身的。

//in不能判断是否是自身的属性

instanceof 运算符表示a(对象)是否是b(构造函数)的实例 a instanceof b

实际上是判断实例对象是否出现在构造函数的原型链上

1. 所有的对象都是数据，但是数据不一定都是对象
2. 要得到一个对象，js中不是实例化类，而是找到一个对象并且克隆他
3. 对象会记住自己的原型
4. 如果一个而对象无法响应某个请求，他会把这个请求委托给自己的原型

# 11.22

继承 拷贝继承 原型继承

Object.is(a,b);判断a和b是不是写得一样（基本数据类型，如果不是同一个对象 都是false）

Object.assign(target,source);浅拷贝

Object.keys(obj) 可以得到obj所有key的集合

Object.values(obj) 可以得到obj所有值的集合

Object.entries(obj) 把每个键值对转成数组 返回这个集合

Object.create(obj) 创建一个对象，并且指定这个对象的原型是obj 这个方法不建议批量调用，原因性能非常低

Object.setPrototypeOf(obj,prototype);设置某个对象的原型

Object.getPrototypeOf(obj);获取某个对象的原型

凡是设置或者读取某个对象的原型的操作，都不建议使用 obj.\_\_proto\_\_,原因是这个属性是浏览器自己实现的，而不是规范

对于原型的设置操作 无论使用哪种方法，效率都是非常低的，所以不建议无故的去批量设置对象的原型。

深拷贝写法！

# 11.23

无new话处理

function \_(){

return new \_\_()

}

简单服务器

# 11.24

app.js：

express模块

path 路径模块 nodejs提供的

path.parse(‘/home/user/dir’)

path.join();

path.resolve()

favicon express提供的

logger

cookieParser

bodyParser

路由 host后面的可变的部分

index

users

app

app.set(‘views’,path.join(\_\_dirname,’views’)) 设置视图模板存放的目录

app.set(‘view engine’,html); 设置视图引擎为html文件

app.set(‘html’,hbs.\_\_express); 设置html文件使用hbs的模板引擎

get和post（提交和上传） put delete

创建一个网站

npm init

在package.jason 的script里面加一个”start” :"node ./index.js"

npm I express

routers文件夹代表路由

npm I nodemon 可以热重启 在script里面加一个

"dev": "nodemon index.js"

新建一个views文件夹用来存放模板

建一个layout 默认模板

安装 body-parser

# 11.27

Ajax的全称是Asynchronous JavaScript and XML中文名称定义为异步的JavaScriot和XML

简单的说：AJAX就是用JS向服务端发起一个请求，并获取服务器返回的内容。

创建一个xhr对象 “打开了浏览器”

var xhr = new XMLHttpRequest();

设定要请求的地址 “在浏览器地址输入了一个url”

第一个参数 设定请求的方式，例如get post…

第二个参数，设定请求的地址

第三个参数，是否同步

xhr.open(“get”,”/api/users”) 绝对路径

*//*设置要传输的数据格式是json的

xhr.setRequestHeader('Content-Type','application/json');

*//*告诉后端咱们使用的是ajax请求

xhr.setRequestHeader('X-Requested-With','XMLHttpRequest');

发起请求 “按下键盘的回车”

xhr.send(data);

监听请求结果

xhr.onload = function(){

请求到服务器返回的结果会放到xhr.responseText

判断请求成功的条件

xhr.status >=200&&<300 ||===304成功 304是本地缓存

}

# 11.28

ajax上传文件

*//*将表单进行序列化

const fd = new FormData();

拖拽文件事件

box.ondragenter

box.ondragover

box.ondrop 放下去的时候就直接打开了 每一个都要清除默认事件

还有ondragleave

想要监听上传进度需要在send前监听

xhr.upload.onprogress = function(e){

e.loaded 代表已经上传的数量

e.total 代表总文件大小

}

# 11.29

console.log() %d数字占位符 %j JSON占位符 %w 字符串占位符 第二个参数替换他

supervisor 自动重启插件 和nodemon一样

跨域 只要协议 主机名 子域名 端口号其中有一个不同就属于跨域 ajax默认不能进行跨域请求

promise是一个构造函数，它代表一个容器，容器内部有一件将来才能完成的事情。每个promise对象都有三种状态 pending 正在处理 resolve 请求已经成功 reject 请求已经失败。

promise对象装固态改变，只能从pendding 🡺 resolve或者从pendding🡺reject 这两种情况

then(cb) 代表异步执行成功的结果的回调函数

catch(cb) 代表异步执行失败的结果的回调函数

Promise.all([p1,p2,p3…])可以接受多个promise对象 并且如果所有的Promise对象都成功了，那么才会返回成功状态，只要有一个失败了，那么就返回reject

Promose.resolve()会直接返回一个状态是resolve的Promise对象

axios

网络是怎样连接的

数学与生活

深入浅出node//不建议

canvas.migong.org

# 11.30

jsonp

encodeURIComponent()将中文转成可以识别的

新的循环

for of 没啥用

let arr = [1,2,3,4,5];

for(let item of arr ){

console.log(item); //item就是value

}

let arr2 = [

{a:1},

{b:2}

]

for(let {key,val} of arr2){

console.log()

}

# 12.1

cookie

session 代表关闭浏览器清除

document.cookie = `name=value; expires=${new Date()}; path=/; domain=.aaa.com; secure`(一定要用分号加空格隔开 expires有效时间 应该new Date().toTimeString)

# 12.4

webpack 模块打包器

export 向外输出一个模块

import 加载一个模块 例如 import {test1} from ‘./test1.js’ 也可以写 test2 as testTwo 意思就是将test2 重命名为testTwo

export default 默认导出模块 只能使用一次

一定要有一个配置文件 webpack.config.js

webpack配置最核心的四个概念

entry入口

output 输出

loader

plugin

module.exports = {

entry:[’./js/app.js’,’./js/app2.js’], 指定入口文件 相对路径 可以加载多个文件 (数组或对象) 前面的先执行 在用对象格式的时候需要使用占位符作为文件名filename:’[name].js’ [name],[chunkhash]都是占位符

output:{

path:path.resolve(\_\_dirname,’dist’), 出口文件需要绝对路径 需要引入path模块

filename:’bundle.js’

}

}

插件plugins 使用插件 先在当前目录安装webpack 安装完成之后require，再引入想使用的插件

plugins:[] 先安装对应的插件 然后在这个数组中实例化

默认只能处理js文件 如果需要处理别的文件需要loader

webpack --watch

babel babel-core babel-loader babel-preset-env

效率低 exclude 代表排除的目录 include 代表只扫描的目录

# react

npm install -g create-react-app

create-react-app my-app

声明式 快速渲染用户交互界面

默认的入口文件 放在src里面 index.js 再新建一个demo1文件夹 新建App.js

*//*引入react

import React from 'react';

import ReactDom from 'react-dom';

react developer tools

reactdev tools

state 生命周期

state代表组件内部自身的状态，state里面的耨个数据不可以被直接修改，如果想修改state里面的数据，必须使用this.setState() 如果state里面的数据发生了变化，那么组件会自动触发自己的render

每个组件都有自己的生命周期

componentWillMount()

render()

componentDidMount()和 componentWillReceiveProps()最常用

componentWillUnmount()

木偶组件 没有功能的组件（只是用来展示数据的）

用function(props){return (……………)}写

属性验证 propstype

表单

设置表单元素的默认value不能再在行间直接使用value 这个属性

需要使用 defaultValue

如果需要使用value这个属性，那么一定要将表单组件变成受控组件（就是在constructor里面state里面的属性）

ref可以找到原生的dom元素 最佳实践 可以用react解决的问题，尽量不要用ref

mongoDB

放到根目录

mongo.config打开 有两行代码 改盘符

进入bin打开cmd

./mongod –-config ..\mongo.config

localhost:27017

net start Mongodb

mongod –config E:\mongodb\mongo.config –install -serviceName Mongodb

keyi net stop

react-router-dom

BrowserRouter 只有单一子节点 可以 as Router 就不用打那么长了

Route path=”/” component={Home} exact可以严格匹配

Link

Switch 当有一个路由符合条件 就不继续匹配

withRouter 可以包装一层

<BrowserRouter> basename属性

<div>

<Link to=”/”>Home</Link>

</div>

</BrowserRouter>

路由分为静态路由和动态路由

静态路由 一个url对应一个组件

动态路由：形式是 某个路由/:变量名 会根据不同的参数渲染不同的组件 让某个被渲染的组件展示出不同的状态

每个组件里面只能有一个子元素，就是必须包在一个div里面！！

Redirect from to 重定向

babel-plugin-transform-decorators-legacy

redux 核心的概念

-store

-reducer

-dispatch

-action

-subscribe

-getState

redux是用来做状态管理的

1 新建store

import {createStore} from ‘redux’;

const reducer = function(state,action){

//state就是数据 action是一个对象，这里面需要一个type属性

//2 定义action(动作)

const INCREMENT = ‘INCREMENT’ 全部大写

}

const store = creatStore(reducer);

npm init main src/index.js

npm i –S react react-dom react-router-dom redux redux-thunk react-redux

npm i –D webpack webpack-dev-server babel-core babel-loder babel-preset-env babel-preset-react html-webpack-plugin rimraf css-loader file-loader url-loader style-loader react-hot-loader

react-redux

import{Provider} from ‘react-redux’

<Provider store = {store}><Provider />

import{connect} from ‘react-redux’ 输出的时候 export default connect(mapStateToProps,)(App) 老写法

npm I babel-plugin-transform-decorators-legacy

新写法 @connect(state =>({num:state})第一个参数是一个函数，函数的返回值必须是一个对象,{addFn,cutFn,………}第二个参数是action)

git remote add origin git@github.com:flow-stateM/react-redux-first.git

html-webpack-plugin

设置title

<%= htmlWebpackPlugin.options.title %>

**new** HtmlWebpackPlugin({

title: '111',

template: './resources/assets/template/index.html', // Load a custom template

inject: true // Inject all scripts into the body

})

# 12.14.2017

构建项目

index.js引入app app中写所有的路由 app中引入reducers 并创建store provider里面传入store（store={store}） reducers是所有reducer的集合（用redux的combineReducers）每个reducer里面就是声明各种行为 将行为封装变为action函数（返回值是{type:…….}） 写一个reducer函数 返回值是state 小的模块里需要用到redux的state就@connect(state=>想引入的reducer,{需要用到的action函数})

项目规范

先打开github

新建仓库 git init 新建文件.gitignore

npm init entry 入口文件 gitrespository仓库地址

建文件夹build public assets静态资源 src src里面再建views 每个页面对应一个文件夹 utils

eslint查询错误

git 上传项目方法

git add .

git commit -m "描述"

git remote add origin https://自己的仓库url地址

git push -u origin master

# 12.15.2017

ctrl+`是打开终端（cmd）

emm设置成alt+enter

搭建react

npm init

npm i –D webpack webpack-dev-server

npm i –D babel-core babel-preset-env babel-preset-react babel-eslint babel-loader eslint-loader html-webpack-plugin rimraf cross-env babel-plugin-transform-decorators-legacy

npm i–D react-hot-loader

npm i –S react react-dom react-router-dom redux redux-thunk axios react-redux

npm i –D eslint //语法检测

npm i –D eslint-config-standard //js标准

npm i –D eslint-pligon-import eslint-pligon-node eslint-pligon-promise

npm i –D eslint-config-airbnb

npm i –D eslint-plugin-jsx-a11y eslint-plugin-react

npm i –D husky webpack-merge(用来做合并配置的)

config 文件夹 webpack.base.js(出入口) webpack.dev.js(打包的) webpack.prod.js(生产环境)

dev里面 引入webpack path html-webpack-plugin

const config ={

entry:{

app:path.join(\_\_dirname,’../src/index.js’)

},

output:{

filename:’[name].[hash]’.js,

path:path.join(\_\_dirname,’../dist’),

publicPath:’/public/’ //和后端定的 静态资源目录

}，

module{

rules:[

{

enforce:’pre’

test:/\.jsx?/,

loader:’eslint-loader’,

exclude:[path.join(\_\_dirname,’../node\_modules’)]

},

]

},

plugins:[

new HtmlPlugin({

template:path.join(\_\_dirname,’../index.html’),

filename:’index.html’,

title:’’

})

]

}

if(process.env.NODE\_ENV ===”development”){

config.entry={

app:[

‘react-hot-loader/patch’,

path.join(\_\_dirname,’../src/index.js’)

]

};

config.plugins.push(new webpack.HotModuleReplacementPlugin());

config.devServer = {

hot:’localhost’,

port:8080,

open:true,

overlay:{

errors:true

}

contentBase:path.join(\_\_dirname,’../dist’),

publicPath:config.output.publicPath,

historyApiFallBack

}

}

在根目录 新建 index.html模板文件 .babelrc里面

{

“presets”:[

“env”,

“react”

]

“plugins”:[

[“transform-decorators-legacy”],

[“react-hot-loader/babel”]

]

}

vscode 插件 editorconfig babeles6es7 npm npmintenlisten icon