2017-11-10

html(5)+css(3)+js

=======================

js ------- javascript

html 负责页面的结构（从语义的角度）

css 负责页面的表现(美化页面)

js 负责页面的交互或行为

=======================

js

1 作用

数据验证

操作元素

网页特效

web游戏

nodejs实现服务器编程

...

移动端页面

pc端页面

2 简介

网景公司 livescript java -------> javascript

96 js1.0

97 js1.1 ------> ECMA(欧洲计算机制造商协会)

98 js2.0

99 ecmascript3.0

07 ecmascript4.0

11.6 ecmascript5.1

2015年6月 ECMAScript6 (es6) ECMAScript2015

es7

3 js组成部分

js三大部分 基础语法+DOM+BOM(了解)

=====================js语法=============

枯燥（但是重要）训练逻辑思维的阶段

1 从hello world说起 见1.html

<script type="text/javascript">

alert("Hello world");

</script>

\* js语言需要配合html页面一起使用

\* js书写位置

内嵌式---把js代码写到一对script标签内部 ，这对标签理论上可以放在页面的任何位置，但是从效率来说建议大家

放在body结束标签之前

行内式---不推荐

外链式--- 把js代码写在一个单独js文件中，在html中引入js即可

2 介绍一些常用js语句

alert() js内置一个功能，弹出窗口

语法 alert("信息");

注意: 符号都是英文状态，除非汉字

prompt() 对话框 见2.html

console.log() 比如 console.log("我会显示在控制台");

1 找错或者学习一个新东西

2 不会阻塞程序的运行

3 注释------提高程序的阅读性

html <!--注释-->

css /\* \*/

js 单行注释 // sublime快捷方式 ctrl+/

多行注释 /\*

sublime快捷方式 ctrl+shift+/

\*/

4 字面量(常量)

固定值的表示方法

\* 数字字面量

123 -12 1.6 ... 789

Infinity NaN(本身就是一个数字)

\* 字符串字面量

一对双引号或单引号引起来的 "hello world" "789"

""123"

\n 换行

\t 制表符

\"

\'

\\ 一个反斜杠

小练习 输出以下字符串 我说:"明天双十一，剁手节快乐"

c:\my\music\单身情歌.mp3

5 变量------相当于一个容器

\* 变量定义和赋值

// 定义变量

var a;

// 给变量赋值

a = 100;

// 使用变量

alert(a);

或 var b = 200;

\* 变量名的命名规则

字母、数字、下划线或$

开头不能为数字

不能与js的关键字（在js中有特殊含义的词语）冲突 比如var var;

不要使用保留字 比如 int short

区分大小写 var a 与var A 不同的变量

练习

var a; // 正确

var \_; // 正确

var $;// 正确

var var;// 错误 关键字冲突

var 2; // 错误

var ~abc; //错误

var a-b; // 连字符不允许出现

\* 变量赋值

var a;

console.log(a); // undefined 此处变量定义了但未赋值，此时js会给他一个默认值undefined

console.log(b); // 报错 b is not defined(…)

\* 变量类型提升

2017-11-13

==========================================================================================================

js

\* 变量的数据类型

js直接量或变量都是有类型的

js数据类型分为简单的数据类型和复杂的数据类型

简单的数据类型分为5种

Number

整数 浮点数（小数） 特殊值(NaN)

String

以双引号或单引号引起的一系列字符

undefined

一个变量定义但未赋值，此时变量值就是undefined,它的类型是undefined

Boolean 布尔类型 只有两个值true/false

null null类型 后面细说

复杂的数据类型

Object 后面细讲

\* 数据类型的转换

1 数字可以转字符串

20+""-----> "20"

""+5------>"5"

2 字符串转数字

"123" ------ > 123

我们可以使用parseInt("520") 进行转换

parseInt() 可以把字符串转为整数

也可以对数字类型的进行取整操作

prompt()的结果是一个字符串

场景（作业） 让用户输入两个数字 计算他们的和并在控制台输出

\* 运算符（操作符）

数学运算符

+ 2+3 2+""(此时加表示字符串拼接)

- 7-3 a-b num1-num2

\* 12\*3

/ 10/5 10/3 10.5/2

% 取模运算符（求余运算符） 关心两数的余数

判断奇偶性 num%5==0

a%2==0 偶数

123%10==3 得到个位上的数

与数学运算相关的对象 Math对象提供了很多属性或方法

++ 自增运算符

-- 自减运算符

\* 流程控制语句

2017-11-15

==========================================================================================================

运算符

1 数学运算符 + - \* / % ++ --

2 比较运算符（关系运算符）

> < <= >= == != === !==

\* 比较运算符的结果是布尔类型 true或false

\* == 不严格等于 只要求值或者内容一样就认为相等

=== 严格等于 不仅要求内容一样 而且数值类型也要一样

!= 是对 == 的否定

!== 是对===的否定

\* 注意 = 不是比较运算符 是赋值运算符

\* 数字与其他类型进行比较时候，其他类型会转化成数字

true----->1 false------>0 null----->0

""------->0

\* 字符串与字符串比较

依次比较字符编码（数字）顺序 字符相同再比较下一个

"abc" "14d"

10>8>4

3 逻辑运算符

&& 与

|| 或

!

true&&true ------> true

true&&false ------>false

false&&true -------> false

false&&false------->false

总结 都真才真，有假就假

对于|| 有真就真，都假才假

对于! !true -----> false

!false ----->true

注意

console.log(45&&null); // 非零数字看成true null看成false

console.log(false&&null); // false

console.log("abc"||true); // "abc"

4 赋值运算符

= += -= \*= /= %=

var a = 3;

a+=b 等价于 a=a+b

5 综合运算

++ -- ！ >算术>比较>逻辑>赋值

var a = 2

a\*=4\*(3-5)&&++a||6>8&&9<10||!6

a\*=4\*(-2)&&++a||6>8&&9<10||!6

a\*=-8&&++a||6>8&&9<10||!6

a\*=-8&&3||6>8&&9<10||!6 // 此时a变成3

a\*=-8&&3||6>8&&9<10||false

a\*= -8&&3||false||false

a\*= 3||false||false

a\*= 3||false

a\*= 3

a=a\*3 // a用原始值

a=6

===============流程控制语句============

程序的结构分为三种

1 顺序结构 程序从上往下 从左往右执行

2 选择结构（分支结构）

3 循环结构（重复结构）

选择结构

1 if(条件表达式) {

语句

}

当程序遇到上述结构的时候，先判断条件表达式真假，假如为真

则执行大括号里面的语句，假如条件为假语句则不执行

2 if(条件表达式) {

语句1

} else {

语句2

}

当程序遇到上述结构的时候，先判断条件表达式真假，假如为真

则执行大括号里面的语句，假如条件为假语句则执行else后面的语句2

2017-11-17

==========================================================================================================

流程控制语句

1 顺序结构

2 选择结构(分支)

3 循环结构(重复)

选择结构

1 if(条件) {

语句

}

2 if(条件) {

语句1

} else {

语句2

}

3 if......else if .... else if .... else

// 1 提示用户输入一个数 并且保存

var score = parseFloat(prompt("请输入你的成绩!!!"));

// 2 可以对数字进行判断

if(score >= 90) {

alert("优秀,继续保持");

} else if(score>=80){

alert("良好,再努力一点");

} else if(score >= 60) {

alert("合格,继续加油");

} else {

alert("不及格 等着补考！！！");

}

// "跳楼现象"

alert("over");

4 if的嵌套

5 三目运算符 ?:

表达式1?表达式2:表达式3

执行流程： 先执行表达式1，假如表1的结果为true,则执

行表2并且把表2的结果作为整个表达式的结果，假如表1的结果是false，则计算表3并且表3的结果是整个表达式的结果

a+b

6 switch结构（多分支结构）

switch(表达式) {

case 常量: 语句

break;

case 常量: 语句

break;

case 常量: 语句

break;

...

defult: 语句

break;

}

\* switch case defult break都是关键词

break是跳出switch结构

\* 执行的流程

先计算出switch后面的小括号里的值，接着拿该值去里面找有没有对应的case,

找到对应的case之后就执行case后面的语句，直到遇到break或者switch结尾结束

找不到对应的case,则执行default后面的语句,

直到遇到break或者switch结尾结束

作业1 用户输入一个月份 显示对应的季节（两种方法）

循环结构

for循环 while循环 do-while循环

\* for(表达1;表达2;表达3) {

循环体

}

for(var i=10; i<=72; i++) {

console.log("阿龙的体重为"+i+"kg");//循环体

}

表达式1一般用于变量的赋值

表达式2一般放置循环条件

表达式3一般是对循环变量进行控制

循环体表示重复的语句

执行流程（推导）

1 先执行表达式1

2 执行表达式2 假如表2为真 则进行第三步,假如表2为假

则循环结束

3 执行循环体

4 执行表达式3

5 接着第二步继续

分析 定义 i = 10 ----> i<=72 (10<=72) 成立 ---> 执行

循环体一次 打印"阿龙的体重为10kg" ---> i++ 此时i为11 --->

i<=72(11<=72) ---> 循环体一次 打印"阿龙的体重为11kg" --->

i为12--->....... i 73

应用（会分析别人的代码 会用循环解决问题）

1 经典题目 1+2+3+4+...+100 (编程计算)

var sum = 0;

// 不断加一个数 i (1~100)

for(var i=1; i<=100; i++) {

sum = sum + i;

}

分析 i=1,1<=100成立，sum=0+1=1 ,i=2,2<=100,sum=1+2=3,

i=3 3<=100 sum=1+2+3,i=4 4<=100 sum=1+2+3+4 ...

2017-11-22

==========================================================================================================

函数

1 含义： 将一些代码封装起来实现一个特定的功能这样的代码块 我们称之函数

{

//.....

}

解决的问题 代码的复用

2 具体的定义

function 函数名() {

// ...

}

3 函数调用

函数定义之后，并不能自动执行，需要程序员去调用

函数调用: 函数名();

4 函数的参数

有时候，我们需要根据参数决定执行相应的结果

function 函数名(参数1,参数2,...) {

// ...

}

// 定义一个函数 求两个数的和

// 参数之间用逗号隔开

function add(a,b) {

var sum = a+b;

console.log(sum);

}

// 调用

add(10,20);

5 函数参数的进一步说明以及函数调用的流程

// 定义一个函数 求两个数的和

// 函数定义的时候，小括号里的参数我们叫形式参数（形参）

function add(a,b) {

var sum = a+b;

console.log(sum);

}

// 调用它 此时的小括号里的参数叫实际参数(实参)

add(50,20);

以上函数的执行流程：执行add(50,20)的时候，开始函数的调用,调用函数过程中，将实参按照对应的位置传递给形参（50->a 20->b）,继续执行函数体 var sum = 50+20, sum的值为70，最后打印70

实参个数少于形参个数,多余的形参采用undefined值

add(80); 80->a b没有值即undefined

实参个数多于形参个数，此时多余的实参被忽略，不会影响程序的正常执行

add(10,15,20);

所有的实参会存在函数内部一个叫arguments类数组对象中

6 函数的返回值

函数内部可以使用return关键词去返回一个值或表达式

renturn还有一个作用，一旦return得到执行，函数结束调用

function add(a,b) {

var sum = a+b;

return sum;

}

function f1() {

if(条件) {

return;

}

}

7 模块化编程

// 1~10000的完美数

// 1~10000的完美数--->什么是完美数--->约数和

8 函数表达式

另外一种函数定义的方式 var a = 10;

var f1 = function() {

alert(1);

}

f1(); // 调用 变量名()

\* 将一个匿名函数(没有名称的函数)赋值给一个变量,原理把赋值号的右边的函数地址赋值了给变量

var f1 = function() {

alert(1);

}

var b = 10;

f1(); // f1存储的是函数地址 引用数据类型

b();// 报错 b存储的是一个普通数值

\* 定义函数表达式，假如函数带名字了，此时不能用函数名调用 只能用变量名调用,但是在函数内部是可以用函数名调用

var f1 = function fun() {

alert(1);

}

总结数据类型

基本数据类型

var a = 10;// a存储的就是10数据本身

var b = a;

b++;

a? b?

引用数据类型

2017-11-24

==========================================================================================================

函数进一步讨论

1 函数声明提升

console.log(a); // undefined

var a = 10;

\* 变量提升只是定义 赋值不提升

函数声明提升 函数可以先调用 后声明 不会出错,对函数表达式

不能进行函数声明提升的,而是进行变量定义提升，没有函数赋值

fun();

fun1();

function fun() {

console.log(1);

}

var fun1 = function() {

alert(2);

}

\* 出现相同的函数名、变量名，JS预解析的时候，优先提升的是函数声明

/\*

fun();

function fun() {

alert(1);

}

var fun = 4;

\*/

var fun = 4;

function fun() {

alert(1);

}

console.log(typeof fun);

fun();

2 作用域

作用域：在作用域内定义的函数或变量，在作用域外不能访问的。

JS中只有函数能够封闭住一个作用域

\* 函数内部定义的变量或函数，只能在函数内部使用，函数外是不能用的

function f1() {

var num = "haha";

console.log(num); // haha

}

f1();

console.log(num); //函数外调用会报错

\* 全局变量

// 参数a,b相当于函数内部的局部变量

function add(a,b) {

var sum = a+b;

}

3 作用域链

当遇到一个变量，js引擎会从该变量所在的作用域依次向外层查找，查找会在找到第一个匹配的标识符为止。在多层嵌套的作用域中可以定义同名的标识符 会发生"覆盖效应"

4 隐式全局变量

函数内部定义变量不写var ,相当于在全局定义

5 全局变量的应用

var num = 4;

function add() {

return ++num;

}

function plus() {

return --num;

}

6 函数作用域：函数也有作用域，定义时所在的作用域就是该函数的作用域

function out() {

function inner() {

alert(1);

}

inner();

}

out(); // 弹出1

inner(); // 报错

2017-11-27

==========================================================================================================

闭包

1 每个函数都是一个闭包

function out() {

var a = 10;

function inner() {

console.log(a);

}

return inner; // 返回的是函数名 函数名就代表函数地址 指针

}

var fun = out(); // var fun = inner

// inner函数相当于记住了自己定义时的作用域，该作用域内有个变量a=10

fun(); // inner() -----> 10

inner函数把它自己内部的语句和自己声明时所处的作用域一起封装成一个封闭的环境，称之为"闭包",函数天生能记住自己的语句以及环境

2 例子1

function out(a) {

function inner(b) {

console.log(a+b);

}

return inner;

}

var fun = out(6);

fun(8);

例子2

function fun1(a,b) {

function fun2(a) {

alert(a+b);

}

return fun2;

}

var f = fun1(3,4); // 3->a(fun1内部的a) 4->b f = fun2

f(6); // fun2(6)----> 6--->a(fun2内部的a) 打印a+b=6+4 10

例子3

function out() {

var num = 8;

function inner() {

return ++num;

}

return inner;

}

var func = out(); // func = inner 记住num=8

console.log(func()); // inner() num=9

console.log(func()); // 10

console.log(func()); // 11

console.log(func()); // 12

学习闭包 为了面试 遇到由于闭包产生的问题会解决

数组

1 概念

一组有序的数据集合,内存中存放的是一组数据

2 定义

// 创建方式一 数组直接量

var arr = [10,20,30,40]; // \*数组元素与元素之间用逗号隔开

\* 数组中的元素可以为不同的类型，但是不建议这样样，一般都是把相同类型的数据放在一起

var arr2 = ["abc",20,13.2,true,[5,6],function fun() {

alert("ok")

}];

3 数组元素的引用

数组名[索引或角标或下标]

计算机在存储数组的时候，会自动把数组元素从第一个到最后一个进行编号，编号从0开始，0 1 2 ... 这样的数字称之角标

角标范围 0~数组元素个数-1 0~数组.length-1

\* 通过给超过最大项数的数组下标赋值 增加数据长度

var arr = [1,3,5,7]; // 此时数组长度(数组元素的个数)为4

arr[8] = 19; // 给第九个元素赋值 此时数组长度为9

console.log(arr[6]);// undefined

console.log(arr);

4 数组长度

\* 数组元素的个数

\* 数组中有一个length属性，可以获取数组元素的个数

\* 用法：arr.length

数组的最后一个元素 arr[arr.length-1]

5 数组的类型

引用类型 对象类型

6 数组的遍历(访问数组中的每一个元素)

7 数组中常用方法

\* 首尾操作

push() 在原数组末尾追加一个或多个元素，返回值是增加后的数组长度

pop() 删除数组最后一项，返回值是被删除的那个元素

shift() 在数组开头删除一个

unshift() 在数组开头追加一个或多个元素

2017-11-29

==========================================================================================================

闭包

1 每个函数都是一个闭包

function out() {

var a = 10;

function inner() {

console.log(a);

}

return inner; // 返回的是函数名 函数名就代表函数地址 指针

}

var fun = out(); // var fun = inner

// inner函数相当于记住了自己定义时的作用域，该作用域内有个变量a=10

fun(); // inner() -----> 10

inner函数把它自己内部的语句和自己声明时所处的作用域一起封装成一个封闭的环境，称之为"闭包",函数天生能记住自己的语句以及环境

2 例子1

function out(a) {

function inner(b) {

console.log(a+b);

}

return inner;

}

var fun = out(6);

fun(8);

例子2

function fun1(a,b) {

function fun2(a) {

alert(a+b);

}

return fun2;

}

var f = fun1(3,4); // 3->a(fun1内部的a) 4->b f = fun2

f(6); // fun2(6)----> 6--->a(fun2内部的a) 打印a+b=6+4 10

例子3

function out() {

var num = 8;

function inner() {

return ++num;

}

return inner;

}

var func = out(); // func = inner 记住num=8

console.log(func()); // inner() num=9

console.log(func()); // 10

console.log(func()); // 11

console.log(func()); // 12

学习闭包 为了面试 遇到由于闭包产生的问题会解决

数组

1 概念

一组有序的数据集合,内存中存放的是一组数据

2 定义

// 创建方式一 数组直接量

var arr = [10,20,30,40]; // \*数组元素与元素之间用逗号隔开

\* 数组中的元素可以为不同的类型，但是不建议这样样，一般都是把相同类型的数据放在一起

var arr2 = ["abc",20,13.2,true,[5,6],function fun() {

alert("ok")

}];

3 数组元素的引用

数组名[索引或角标或下标]

计算机在存储数组的时候，会自动把数组元素从第一个到最后一个进行编号，编号从0开始，0 1 2 ... 这样的数字称之角标

角标范围 0~数组元素个数-1 0~数组.length-1

\* 通过给超过最大项数的数组下标赋值 增加数据长度

var arr = [1,3,5,7]; // 此时数组长度(数组元素的个数)为4

arr[8] = 19; // 给第九个元素赋值 此时数组长度为9

console.log(arr[6]);// undefined

console.log(arr);

4 数组长度

\* 数组元素的个数

\* 数组中有一个length属性，可以获取数组元素的个数

\* 用法：arr.length

数组的最后一个元素 arr[arr.length-1]

5 数组的类型

引用类型 对象类型

6 数组的遍历(访问数组中的每一个元素)

熟练

7 数组中常用方法

\* 首尾操作

push() 在原数组末尾追加一个或多个元素，返回值是增加后的数组长度

pop() 删除数组最后一项，返回值是被删除的那个元素

shift() 在数组开头删除一个

unshift() 在数组开头追加一个或多个元素

\* 合并与拆分

合并 concat() 将两个数组合并为一个数组，结果是一个合并后的数组，原来数组不变

拆分 slice(start,end) start表示起始索引位置 end最终索引位置，但是不包含最终的位置

demo1

var arr = [1,2,3,4,5,6,7,8];

var arr2 = arr.slice(3,7); // slice() 表示提取一部分 3 索引的起始位置 7 索引的最终位置但是不包含

console.log(arr); // [1, 2, 3, 4, 5, 6, 7, 8]

console.log(arr2); // [4, 5, 6, 7]

demo2

var arr3 = arr.slice(1); // [2, 3, 4, 5, 6, 7, 8]一个参数表示开始索引 一直截取到末尾

\* 插入 删除 替换

splice(start,howmany,ele1,ele2,...);

-- start 表示删除元素的起始位置 正数表示索引值，负数表示从后面往前数第几个

-- howmany 删除元素的个数 不能为负

-- element 删除之后替换删除部分的其他新数据（不是必填）

\* 数组转字符串

join() 将数组的所有元素连接成一个字符串

\* 排序

\* 倒序 reverse()

\* 排序 sort()

sort() 默认排序 按照字符编码顺序 0~9 A~Z a~z

我们可以实现自定义规则

字符串

1 length属性

表示字符串中字符的个数（长度）

2 常用方法

concat() 连接两个字符串，形成一个新的长的字符串

字符串1+字符串2也可以实现

charAt(index) 返回指定索引位置的字符 字符串索引也是从0计算

indexOf() 参数可以传一个字符或字符串，返回的是该字符或字符串在整个字符串中第一次出现的位置

大小写转换 toLowerCase() toUpperCase()

split 通过将字符串分离一个个子串，形成一个字符串数组

slice 截取字符串的片段 并在新的字符串中返回被提取的部分

slice() 与数组方法使用一样

substr

substring

trim 去掉字符串首尾的空白字符

var str = " anh hshh";

var res = str.trim();

console.log(res); // "anh hshh"

2017-12-01

==========================================================================================================

基础语法补充

1 函数中arguments对象

\* js函数内部有一个arguments对象,函数在执行过程中，传的实参最后都会保存在arguments对象(类数组对象)中，可以当数组来用，注意该对象只能在函数内部使用，不能在函数外使用

\*arguments对象(类数组对象)可以获得长度 每一项数据 ，但是有的方法不能用 slice

\* 实现重载

用户使用一个数，让数字乘以2，用户使用两个个数，求其和，用户使用三个数，我们计算三个数的最大值，超过3个数，报错

2 IIFE 即时调用函数表达式

函数定义时，就执行这个函数，函数执行语法函数名()或变量名()

参考案例3.html

实际工作 用一对() 包裹一个匿名函数实现IIFE

IIFE定义的函数是一个局部作用域的函数，()关住了它的作用域，作用域外访问不了

(function func() {

console.log("函数aaaa");

})();

func();

常用 方式

(function() {

console.log("函数aaaa");

})();

结合数组观察闭包

===============================

DOM概述

DOM(文档对象模型 Document Object Model)描绘了一种层次结构的树,允许"码农"添加，删除，修改页面的某一部分。

Dom是js提供我们一套操作html结构和样式的api

DOM的学习就是学习document对象，其中封装了很多的方法和属性，可以完成大部分的DOM操作

document

|

html(根元素)

|

head body

| |

meta title div script

(id="box")

概念 1 节点（一切皆节点）

标签（标记 元素） -------> 元素节点

属性节点 比如div中的id属性

文本节点 盒子里面的内容

习惯上一般认为节点指的就是标签元素

2017-12-04

==========================================================================================================

ECMASCRIPT js as(ActionScript)

DOM(重点)

BOM(了解)

=========================

DOM

1 文档对象模型，js把文档抽象出一个树，把树的组成部分称之节点（元素节点 属性节点 文本节点）

2 DOM规范

98年之前的 0级规范

98 10 DOM1级规范

... 2级规范

... 3级规范

3 DOM的东西

获得元素、css样式、事件、运动、元素尺寸，节点操作等

===================================

1 获取dom元素（前提）

document.getElementById("id名字");

// get------得到

// Element------元素 单个

// ById 通过id

注意 id一般区分大小写 ie6/7不区分

ie7 id获取元素时候 name属性也会当成id属性获取

<p name="p1"></p>

document.getElementById("p1");

2 操作html

<img src="..." />

js提供两种选择 1 getAttribute()和setAttribute()

2 点语法

总结 除了自定义属性用getAttribute()和setAttribute(),其他尽量使用点语法(尤其style)

3 js操作css

\* 对象打点调用style属性，是一个行内样式的样式对象，继续打点得到具体样式属性，并且可以进行更改

\* 对于class定义的样式 用className属性

.demo {

width: 200px;

height: 200px;

}

// 获取元素---->元素.className ="原来 demo"

4 innerHTML与innerText和value

获取元素的内容 innerHTML支持标签

innerText只获取文本

value 获取变单元素内部的内容

5 js事件

事件是js给我们封装好的具体的行为，常见的事件如鼠标单击，鼠标悬停，鼠标的双击，键盘的单击....

事件三要素：事件名称 事件源 事件响应（事件处理）

打开计算机

事件源 电源开关

事件 按下去

事件处理 计算机开机

猎人打猎物

事件源 枪

事件 打

事件响应 子弹飞出去 猎物被吓跑了

js事件类型 DOM0级事件

onclick 单击

ondblclick 双击

onmouseover 鼠标移上事件

onmouseout 鼠标移走事件

onmousedown 鼠标点击按下事件

onmouseup 鼠标点击后弹起

onfocus 获取焦点事件

onblur 失去焦点

onload 页面加载事件

2017-12-06

==========================================================================================================

js事件处理

事件源.事件 = function() {

}

===========================

查找元素基本方法(无兼容问题)

1 根据id 一个元素

2 getElementsByTagName()

通过标签名找一堆元素 结果是一个数组

\* document.getElementsByTagName()

元素.getElementsByTagName()

var ul = document.getElementsByTagName("ul")[0];

应用: 批量操作

表格隔行变色

批量添加事件

2017-12-08

==========================================================================================================

2017-12-11

==========================================================================================================

DOM案例 1 轮播图 2 tab栏切换

计算后样式

1 概念 指的是html元素在多种css选择器综合作用下,得到最终样式

2 点语法 obj.style.width 只能得到行内式

3 不同浏览器实现的方式不一样

IE9+以及其他高级浏览器

window.getComputedStyle(元素) 该方法接受一个DOM元素，返回一个可以进行属性查询的接口，接口提供了一个方法getPropertyValue(具体的属性)

\*

window.getComputedStyle(box).getPropertyValue("border-width")

getPropertyValue()方法得到样式属性值,样式属性的书写用横线方式，不能用驼峰式名称

\* []语法

console.log(window.getComputedStyle(box)["border-style"]); // solid

console.log(window.getComputedStyle(box)["borderStyle"]); // solid 支持驼峰式命名写法

IE9以下

IE9以前的版本，有一个专门的api，附加在元素身上的currentStyle属性

box.currentStyle.width

兼容的处理

2017-12-13

==========================================================================================================

offsetLeft和offsetTop

1 计算后样式得到的属性值是字符串类型，js提供了一些快捷获得尺寸和位置的方法，得到的数据时数值类型

300->350

"300px"+"50px"-----> "300px50px"

offsetLeft和offsetTop

offsetWidth和offsetHeight

clientWidth和clientHeight

2 offsetLeft和offsetTop

兼容问题

元素天生就有一个属性offsetParent偏移参考对象

offsetLeft:元素到offsetParent之间的距离

offsetTop: 同理

\* offsetParent确定

IE9+以及高级浏览器,

1 offsetParent是距离元素自身最近的带有定位的祖先元素 ，假如祖先中没有任何定位的元素，此时参考元素就是body

2 offsetParent与自身元素是否定位无关

3 offsetLeft 元素自身的左边框外部到offsetParent左边框内部的距离

IE6/7

offsetParent:

1 自身假如没有定位，offsetParent距离它最近的有宽高的祖先元素，假如没有宽高，参考body

2 绝对定位的元素即便不设置宽高，它的宽高有内容自动撑开，此时相当于设置了宽高

3 自身有定位，与高级浏览器方法一样

IE8

offsetParent: 与高级浏览器判断依据一样

offsetLeft: 比高级浏览器多一条offsetParent的左边框宽度

\* offsetLeft兼容处理（css巧妙设置）

设置：自身元素设置定位（让不同浏览器判断依据一样）

offsetParent自身定位不设置边框

3 offsetWidth和offsetHeight

与offsetParent无关，就是元素实际占有的宽高，无兼容问题

offsetWidth = width + padding(左右) + border(左右)

offsetHeight = height + padding(上下)+border(上下)

假如没有设置宽度，自动得到一个加载的宽度 或者撑满元素 或者被子元素撑开

4 clientWidth与clientHeight

基本上兼容 IE6,7有问题

clientWidth = width + padding

对于高级浏览器，若不设置宽高，得到的值就是撑开的宽和高

IE6/7 若不设置宽高 结果输出都是0

5 定时器

我们有时候希望每隔一段时间，代码就执行一次，js提供一个定时器机制

window对象提供一个方法 setInterval() 方法，作用每隔一个时间 重复执行一次函数

具体语法 window.setInterval(函数,时间间隔);

或setInterval(函数，时间间隔);

2017-12-15

==========================================================================================================

定时器

1 每隔一段时间需要执行一段代码

2 语法

window.setInterval()或setInterval()

setInterval(函数,时间间隔);

setInterval(function() {

// 重复执行的代码

},30);

setInterval(move, 30);

function move() {

}

3 函数什么被调用？

a 自己调用 b 事件触发 c 定时器工作

4 物体的运动

原理：定时器的应用

运动的速度如何控制 1 更改间隔时间 2 更改运动的步长

帧频------1s走的次数

100ms 10fps

5 取消定时器

clearInterval(定时器id);

6 案例2带来一些问题和新的需求

问题1 定时器定义放在事件函数中，当事件多次触发，执行多次事件函数，导致开启多个定时器，每隔一段时间，函数被执行n次，运动速度越来越快，定时器无法清除

解决方案(设表先关) 在开启定时器之前，先清除一次定时器

问题2 需要让运动停止在指定的位置,假如步长设置不合理或时间间隔不合理，终点位置会出现误差

解决方案 强制将值拉回终点，然后再停止定时器

需求 需要在一个固定时长内，达到终点，间隔时间你可以自动设置，合理的步长如何确定？

3s 起点 终点 时间间隔 50ms 计算步长

0 500

运动次数 3000/50 (终点-起点)/60

每一次的步长 = (终点-起点)/(总时间/时间间隔)

7 定时器的应用

简单无缝滚动

升级版无缝滚动

变化 1 li元素宽高不设置死，用内容撑开

2 用js动态复制html结构

3 img将来会撑开li的宽度，li将来会撑开ul的宽度

ul的宽度一半即临界值

2017-12-18

==========================================================================================================

8 json

1 概念 javascript object notation js对象表示方法

2 创建 var json1 = {"k":"v","k":"v",...}，k必须要双引号

3 作用 存储数据

4 json属性的操作

属性的获取

方式一json对象名.属性名

方式二json对象名["属性名"]

属性的更改

//修改json属性 修改值

myJson.age = 31;

console.log(myJson.age);

myJson["height"] = 180;

console.log(myJson.height);

// 添加数据

myJson.job = "IT teacher";

console.log(myJson);

myJson["hobby"] = ["sing","coding","running"];

console.log(myJson);

myJson.birthday = {

"year": 1988,

"month": 10,

"day": 1

}

console.log(myJson);

// 如何获取出生的月份

var month = myJson.birthday.month;

alert(month);

5 json对象的遍历(for in)

for...in 专门遍历json对象

for(var k in json) {

// k 表示json对象所有的属性名

}

2017-12-20

==========================================================================================================

复习

1 offsetLeft和offsetTop

div.offsetLeft

2 offsetWidth和offsetHeight

width+padding+border（无兼容问题）

3 clientWidth和clientHeight

width+padding（不含border,兼容性基本兼容 ie6 7 若不设置宽高 得到值0）

4 setInterval()定时器

window对象 setInterval(函数，事件ms)

clearInterval()取消定时器

问题1 定时器定义放在事件函数中，当事件多次触发，执行多次事件函数，导致开启多个定时器，每隔一段时间，函数被执行n次，运动速度越来越快，定时器无法清除

解决方案(设表先关) 在开启定时器之前，先清除一次定时器

问题2 需要让运动停止在指定的位置,假如步长设置不合理或时间间隔不合理，终点位置会出现误差

解决方案 强制将值拉回终点，然后再停止定时器

需求 需要在一个固定时长内，达到终点，间隔时间你可以自动设置，合理的步长如何确定？

3s 起点 终点 时间间隔 50ms 计算步长

0 500

运动次数 3000/50 (终点-起点)/60

每一次的步长 = (终点-起点)/(总时间/时间间隔)

5 json

{"k":v,"k":v} {k:v,k:v}

对象表示方法 1 json格式的对象 2 普通的对象

=====================================================================================

延时器-setTimeout(函数，时间)

\* window对象的方法

\* 延时器：在多久后，执行第一个参数的函数一次 “定时炸弹”

\*

2017-12-20

==========================================================================================================

2017-12-22

==========================================================================================================

异步和回调函数

1 异步:异步处理不用阻塞当前线程来等待处理完成，允许后续操作，直至其他线程处理完成，并回调通知次线程

同步: js假如没有特殊的结构，js执行是同步的，意思就是说前面的语句必须完成之后，再执行后面的语句，假如前面的语句没有执行完，后面的语句继续等待

举例子：

老板让秘书去喊你谈话

异步处理：setTimeout()

setInterval()

Ajax

Node.js

2 回调函数

3 call与apply

我们希望在普通函数中自定义自己的this,此时可以用call()或apply()

区别:函数传参不一样

函数名.call(obj,参数1,参数2,...)

函数名.apply(obj,[参数1，参数2，参数3...])

4 函数节流

我们绑定事件，会设置函数，事件可以被反复触发,函数在短时间内执行多次，可能会影响其他函数执行

实际工作，为防止用户快速点击，点击一次之后，隔一段时间才再次触发，中间即便有点击 也不能去执行函数内部语句

思路:

假如第一次单击 此时函数肯定需要执行

执行过程中，防止用户快速点击，可以设置一把"锁",

过段时间把锁打开 其他人点击就会执行

setTimeout

2017-12-25

==========================================================================================================

补充

opacity

===================

1 什么是DOM?

DOM开发 html+css+js

DOM能做哪些事情

\* 操作 html、css

\* 页面元素的事件绑定

\* DOM可以删除 更改页面的元素

\* 元素动画

...

2011之前 100%就是DOM特效开发

今天：

canvas特效开发

Ajax(请求数据 建立模板 组件页面)

Node.js 后台服务器

ng/vue/react(前端mvvm框架)

DOM开发(基本功)

jquery 世界公认的开发DOM之神

jquery哲学：简化dom开发 为兼容而生

2 jquery入门

\* 设计理念 write less,do more

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

\* jquery不是一门新的语言 只是一个js库 本质上就是一个js文件

兼容性做的很棒

动能强大 DOM操作 事件机制 动画 ajax

可扩展

...

\* jquery版本

1.\* 兼容ie6 7 8

2.\*和3.\* 不再兼容ie 6 7 8

\* jquery的初步使用

下载 官网

2017-12-27

==========================================================================================================

1 整体感知

$()

a 改变css属性 $("li").css("background-color","red");

b Dom元素绑定事件

c 可以给dom元素追加 删除 节点

d 可以做动画 animate()

2 jquery特点 批量处理

3 jquery对象与原生dom元素之间的转换

原生obj ------>$(obj) 变成了jquery对象

this ------>$(this)

$("div") ------> 原生 $("div")[索引]或$("div").get(索引)

4 jquery选择器

基础选择器（效仿css选择器）

自己特有的选择器

first last odd even lt(index) gt(index) eq(index)

5 jquery操作css

\* 方法 css() 方法

\* $(selector).css("属性名","属性值")

比如 $(".demo").css("width","200px");

$(".demo").css("height","200px");

$(".demo").css("color","purple");

$(".demo").css("text-align","center");

...

// 多个属性同时设置

$(selector).css({

"k":"v",

"k":"v",

"k":"v"

});

注意 1 css()对象里的复合属性可以用-隔开或者用驼峰式命名方式

2 属性值是数字的时候，可不加引号

6 jquery动画

show() hide() toggle()

fadeIn() fadeOut() fadeToggle() fadeTo()

slideDown slideUp slideToggle

7 计算节点的个数

size()

length

8 jquery常用方法

html() ------原生js innerHTML

\* 作用 获取节点中所有的内容或设置节点的内容

\* 支持标签

text() ------原生js innerText textContent

\* 作用 只获取节点中所有的内容(不包含标记)

attr()

\* 访问节点属性值 attr("K")

\* 设置节点属性 attr("k","v");

\* 注意单选按钮和多选框的状态有专门的方法prop()

2017-12-29

==========================================================================================================

2018-01-03

9 jquery事件绑定

$(selector).click(function() {}); 单击事件

$(selector).mouseenter(...);

$(selector).mouseleave(...);

$(selector).on(事件名称,function() {}) 推荐

10 获取宽高

$(selector).width();

$(selector).height();

2018-01-05

11 节点关系

什么节点关系？

$(this) -------> 当前事件源本身

children() -------> 亲儿子

parent() ------> 亲父亲

siblings() ------> 自己的同级元素（不包含自己）

eq()

index()

12 addClass() removeClass() hasClass()

原生js className

13 each()方法

遍历

$(selector).each(function(i,el) {

// i 表示当前遍历元素的索引

// el 当前遍历的元素（DOM元素）

// 业务逻辑

});

$("div").css("color","red");

14 animate()方法

animate(json,time,callback);

json必须的参数,表示动画终点的状态值，可以由什么构成?

参数由数字控制的，不能改变背景色，背景定位

time可选 不写默认值为400 单位毫秒

callback 元素动画完成后调用的方法

\* animate()动画排队

动画在同一个元素上是排队的，先写的动画先执行，后写的后执行

$(this).parent().animate({"height":"45px"},1000);

$(this).parent().animate({"width":"45px"},1000,function() {

// 让a元素增加类样式

$("#btn").addClass("current");

});

$(this).parent().animate({"width":"45px","height":"45px"},1000,function() {

// 让a元素增加类样式

$("#btn").addClass("current");

});

2018-01-08

15 stop()方法

stop(参数1,参数2)

参数都是boolean,参数1表示是否清空动画排队序列，参数2是否立即完成当前动画，stop()方法一定要加在动画方法之前

用法 stop() === stop(false,false)

stop(true) === stop(true,false) 最常用

stop(true,true)

stop(false,true)

16 动画节流

我们在写jquery动画的时候，往往一个元素身上发生多个动画事件，这个时候，产生动画积累，为防止动画的积累产生,

就用动画节流 用is()方法

$(selector).is(":animated") 判断当前元素是否在运动

$(selector).is(":visible") 判断是否元素可见

$(selector).is(".demo") 判断有没有class名为demo的类

案例 手风琴