一：js基础

js：一种解释型语言，不需编译，由浏览器进行解析，和Java没关系（前端语言）

作用：增强客户端的交互效果（）；对网页增删改查；

JavaScript技术体系：

ECMAscript：ES5，ES6：定义核心语法及提供一些内置对象

BOM（浏览器对象模型）：提供对于浏览器的相关操作

DOM：提供对网页的相关操作

1:引入方式

内嵌式<script></script> 一般写在body内的最后

外部引入<script src="xx.js"></script>

直接通过元素引用 <div onclik=""></div>以on开头

伪URL方式引入<a href="javascript:函数"></a>

2：变量：用来临时存储数据的容器/空间

var a = 10; 把10赋值给变量a

声明：var

变量名（标识符-概念更大）a：必须是英文、数字、$、\_；开头不能是数字

变量名不能是关键字，保留字

“=”是赋值符号

命名规则：1：见名知意

2：驼峰命名：首个单词首字母小写，其他单词首字母大写

3：帕斯卡命名：每个单词首字母大写

3：输出方式

document.write(a); 在页面输出

alert(a); 弹窗

console.log(a); 在控制台输出

4：输入方式

var number;

number=window.prompt(“提示语言”); 不输入值直接点击确定会打印一个空格

点击取消会打印一个null

console.log(number);

window.prompt获取的都是字符串类型，

二：数据类型

1：基本类型（栈）

1：undefined：变量已定义，但未赋值

2：null：表示不存在对象，由undefined派生而来

3：boolean：只有两个值true/false

4：number：数值类型

5：string：字符串类型 用“”/' '表示

+号代表拼接 “123”+“123”输出123123

2：引用类型（堆）

1：object

2：function

3：数组

3：数据类型的转换（转换方法首字母都是大写）:

转换为数字

1：Number：转换时是整体转换

字符串，如果是数字直接转为对应的数字，如果有字母直接转为NaN

null：0 ; undefined：NaN; true：1； false：0；

2：var a="123";

var b=parseInt(a);将字符串类型转为整型，不包括小数点

只能转数值类型，后面出现不适数值类型的会直接被砍掉，例如：

"a123"----NaN; "1a23"---1; "123aa"---123

3： parseFloat():转换为浮点型，包括小数点，只认一个小数点，小数点后面的0会自动去掉

字符串转换

null和undefined本身不存在，不能使用字符串的转换方法，用了会报错.

直接加一个引号

var a=10;

var b=a.tostring(); 将数值转为字符串类型

Boolean转换

var a=0;

a=Boolean(a);

console.log(a); ----输出false

"0"/非0的数字/字符串---true

0/未赋值/空字符串/null---false

三：运算符

算术运算符：+ - \* / %

+：当+号两边都是数字时，表示进行加法运算

当+号两边有一边是字符串时，表示连接

- ：var box=10-"a" // NaN

除法：alert(parseInt(10/3)); //3取整

% 取模：alert(10%3); //1 求余数 模的正负只跟除数有关

-10%/3 -1; -10%/-3 -1; 10%/3 1;

3：小数点减法bug 解决办法 先放大后缩小

var a=0.4;

var b=0.3;

console.log((0.4\*10-0.3\*10)/10) ;

一元运算符

++ --

box=box+1等同于box++ 等同于++box

如果在复合表达式中++在前表示先进行+1再使用变量；++在后先使用变量再+1；

box=box-1等同于box--等同于--box

前置与后置的区别

var box=100;

var num=box++; box先赋值后自加

console.log(num); //100

console.log(box); //101

var box=100;

console.log(box+++5); //105 box先运算+5

console.log(box); //101 box自加

关系运算符 结果都是boolean类型（true/false）

NaN和任何进行比较结果都是NaN

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

== ：比较的只是值 5==“5” //true

===：比较的是值和类型 5==="5" //false

！=：比较的只是值 5！=“5” //false

！==：比较的是值和类型 5！==“5” //true

关系运算符比较是：两个操作数有一个是数值，则将另一个转换为数值，再进 行数值比较

逻辑运算符 一般和关系运算符配合使用

&&逻辑与：所有条件为真才为真，有一个条件为假结果就为假，并且这个条件之后的的条件不在进行判断

||逻辑或：只要有一个条件为真，结果就为真

优先级：|| < && < !

**三元运算符**

条件表达式？表达式1：表达式2

如果为真执行表达式1，否则执行表达式2

运算符的优先级

1：逻辑非运算符！= 最高

2：算术运算符 ++ --

3：关系运算符 ==；===；！==；>=;<=;>,<

4：逻辑运算符 && ||

5：最低的是赋值运算符 =

复合表达式：用运算符连接

任何数字和NaN进行运算结果都是NaN

四：循环

程序：一系列有序的指令

程序=算法+数据结构

变量的声明一般写在循环的外面；要清楚是从哪一项开始，一共有多少项；

编程3大结构：控制程序执行的顺序

1：顺序结构--从上往下执行

2：选择结构--单分支、双分支、多分支 通过判断选择要执行的语句

3：循环结构--重复执行，知道满足条件才停止

选择结构 ：判断的都是一个变量

1：单分支

if(条件表达式){ } ；满足条件才会执行

条件表达式带有逻辑或关系运算符（条件的结构是true/false）

2：双分支

if(条件表达式){满足条件执行}

else{若条件不满足执行}

一定会执行一条

3：多分支

if(条件表达式){满足条件执行}

else if(条件表达式){满足条件执行}有一个条件成立后面的都不再判断

else{若条件不满足执行}

多个if:每个if的条件都会进行判断；多用于区间或等值

嵌套if:判断条件是不一样的

4：switch：条件对于多个 ；多用于等值

switch（表达式）{

case 常量1: //case值判断的是全等,case后面的值必须是常量,常量值不能相同

case没有顺序之分

执行语句；

break; //条件满足就结束运行，跳出switch结构

case 常量2:

执行语句；

break;

default: //当输入的方案没有时就结束，跳出；default可以省略;

可以放在任意位置

执行语句；

break;

}

当条件满足某一个情况时，就不在进行条件的判断，从上到下去执行，知道遇到break结束

var day= parseInt(window.prompt());

switch（day）{

case "M":console.log("周一");break;

case "s":console.log("周二");break;

default:console.log("输入有误，重新输入");

}

不加break表示从哪个位置开始执行，之后的都会执行

break:结束运行，跳出，后面的不运行

default:当输出的方案没有时，跳出

循环结构

循环是什么：重复执行的代码块。

循环的特点：重复，大量，有规则

循环的组成：循环条件--初始条件，条件的判断，条件的改变

循环操作

当型循环：先判定后执行，可能一次也不执行 while for

直到型循环：先执行后判断，至少执行一次 do while

while(条件表达式){循环体} 先验证再循环，执行0次或多次

do{循环体} while（布尔表达式）; 后验循环，至少执行一次

次数的判断一般用while,不是次数的一般用do while

for（表达式1；表达式2；表达式3）{循环体} 先验循环

for （var a=1;a<=10;a++）{

console.log("hello world")

}

**使用场景**

次数固定用for循环

次数不固定while 或 do while :先判断后执行while; 先执行后判断do while

**break;continue的区别：**

continue只能用于循环结构，结束本次循环进入下一次循环

break可用于switch和循环结构，跳出当前结构

**嵌套循环/双重循环：**

外层循环一次，内层遍历一遍

外层循环控制行，内层循环控制列

五：数组

数组：数组中可以放不同的数据类型

按一定顺序排列，具有某种相同性质变量的集合，这些变量具有相同的名字，在内存中顺序排列，并通过下标相互区分。

数组的typeof都是object类型

创建的方法

1：var list=new Array();

2：var list=[];

3：var list=new Array(10); 带有长度的创建不是赋值

var list=new Array("10"); 赋值

4：var list=new Array(“xiaoming”,9,8,7);带有值:的创建 数据类型可不同

5：var list=[1,2,3,3,4,5];带有默认值得创建

6：var list= Array（）；

数组一般搭配循环使用；

数组长度通过length属性来获取，位置时下标

下标越界：返回值是undefined

arr.join(" | ");数组里的值得分隔符。

arr.reverse();倒序，原数组会被改变

arr.sort();从小到大排序，原数组会被改变;需写一个函数

arr1.concat(arr2);数组连接在一起，原数组不会改变

pop push--先进后出；堆栈操作

shift unshift--先进先出；队列操作

var list=new Array();

list[1]="第一个 "；

console.log(list.length); //2 位置0的值是undefined

如果取值的长度超过了定义的长度：叫下标越界

var list=new Array();

for( var i=0;i<20;i++){

list[i]=parseInt(window.prompt()); //赋值

}

添加数据：

push；list.push(parseInt(window.prompt()));往数组添加一个数组元素，会改变数组的length属性；不是赋值,长度每次都在增加会导致死循环。该方法可以添加任意数量的数据，以逗号形式隔开；并把他们逐个添加到数组的末尾，并改变数组的长度。

onshift：添加到数组的开头。

删除数据：

pop(); list.pop()；默认删除最后一个元素

shift（）：从开头开始删除；

splice；list.splice(2,2); 会改变原数组的长度；返回删除后的数组；第一参数代表删除的起始下标位置；第二个参数代表要删除的元素个数；第三个参数代表插入的元素，可以插入多个元素，是插在开始位置的前面（相当于替换元素）；

一般删除数组元素都倒着删

六：函数

函数：是由事件驱动的或者当他被调用时执行的可重复使用的代码块

作用：

1：重用--避免代码的重复写入，提高代码优化率

2：分离--以功能模块方式将代码分离，提高代码的可读性

3：封装--

定义：定义一个函数，执行程序将函数加入到内存，等待调用

调用才会执行

1：var myFun=function (){ //函数的表达式，调用要在后面

执行的语句

}

myFun(); //调用； 这个调用方法必须放在函数的后面

这种定义方式必须放在调用的前面否则会被认为没有定义undefined;

一般放在程序的最前面，程序开始执行才加载

2：function 函数名（参数1 ，参数2，......）{ .//函数的定义式

函数体---要执行的语句

}

myfunction(); //调用该函数，函数执行 不管写在哪个位置都可以；程序开始时就已经在内存中加载完毕了

变量与函数都会提前申明，变量的赋值不会提前

函数的调用：

可以在函数里面调用其他函数；自己调用自己就叫递归函数

从形式上看，函数分为两类：无参函数 有参函数

形参与实参；return 可有可无

形参：定义参数中的变量；函数定义时，申明的参数

实参 ：调用函数时所传入的参数（具体值），实参的值复制给形参

基本数据类型复制的是一个值

引用数据类型复制的是一个内存地址，指向同一个地方的值

有返回值/无返回值

有return：只能返回一个变量不能返回多个//可用数组返回多个值

遇见return函数会直接跳出，后面的代码语句不会再执行

不能写在for循环里面

不一定要返回值，可用于结束当前操作

无return：没有返回值时，返回的是undefined；

全局变量与局部变量：

局部变量：在函数内声明的变量，只有当前的函数才可以访问这个变量

全局变量：在函数外声明的变量，任何函数都可以访问；当省略var关键字时，默认

当前的是一个全局变量

break：跳出本层循环，而不是所有循环结束

continue：结束本次循环，直接进入到下一次循环，与外层循环无关

charAt()获取字符

空格与空字符的区别：Boolean(" ");//true Boolean(""); //falsee

七：string 通过new出来的都是对象

String 也是引用类型

js的字符串是不可变的，String类 定义的方法都不能改变字符串的内容，它返回的是全新的字符串，而不是修改原字符串

创建：

1：var a="mystring"; typeof a------string

2：var b=new String("mystring"); 创建一个string对象 typeof b------object 说明是引用类型

3：var c= String("mystring"); typeof c------string

属性：就是一个变量---length

字符方法：就是一个函数

1：concat();字符串拼接，连接等同于+

2：substring(a,b); 截字符串，截取位置a-b包含a，不包含b;; 参数只能为正数、0

substring(a); 从位置a开始截取，后面全部截取

如果开始位置比结束位置大，在提取之前会先自动交换这两个参数；

3： substr(a,b);a表示开始的位置；b表示截取的长度；a可以是负数，b不能为负

4：slice(a,b); 截字符串，截取位置a-b包含a，不包含b; b=a+截取的长度

只有一个参数时，表示后面的全部截取

参数可以为负数倒着算；-3表示从最后一位到-3这位置的所有字符返回被提取的字符串:

5：indexof();搜索首次出现的位置

indexof("a",0); 从位置0 开始查找a出现的位置

6：replace(); 替换

replace(“a”"b"); 把a替换成b 只会替换第一个出现的a

7：search("a"); 返回a首次出现的位置，没有则返回-1；

方法用于检索字符串中指定的子字符串，或检索与正则表达式相匹配的子字符串..

8：split();拆分；返回的是字符串数组

var a="hello,world,morning,evening" ;

a.split(","); //从逗号的地方分割

a.split(''); //一个字符一个字符的分割开

9：trim(); 删除字符串前面和后面的空格，中间的空格不能去

var a=" abcdcf ";

alert(a.trim()); //abcdcf 空格被取消

10：charAt();获取字符

var name="hello world";

name.charAt(1); // e 返回字符串的位置1的字符

八：Date

var a=new Date(); //显示的是当前计算机上的系统时间 国际标准

a.setFullYear(2008); //设置年份

a.setFullMonth(8); //设置月份，从0开始的

a.setFullDate(7); //设置日

a.toLocalTimeString(); //转成 10:22:12

a.toLocalDateString(); //转成 2017-6-22

a.getYear(); //获取年 117 从1900到现在的年份之差

a.getFullYear(); //获取年 2017

a.getMouth(); //获取月 从0开始算的 一般用+1

a.getDate(); //获取日

a.getDay(); //获取星期几

a.getHours();

a.getMinutes();

a.getSeconds();

a.getTime(); //从1970年 到现在经过的毫秒数

将输入的日期转换为Date类型：

var date1=window.prompt("请输入第一个日期");

var date=new Date(date1);

var day1=new Date("2015-1-1");

var day2=new Date("2015-3-1");

var day=day2-day2 ; //相差的是毫秒数

day/1000/3600/24 //相差的天数

九：正则

定义的规则

var a = /^ $/; 中间是匹配的规则

var r=new RegExp("[0-9]"); ""内 是匹配的规则 构造函数的方式

[a-zA-Z]：匹配a-zA-Z的任意一个字符

[a-zA-Z]{1,10} ：至少一个最多10英文

\b：单词边界

\r：回车符

\d：匹配0-9的数字；等同于[0-9]

\D：匹配飞数字字符；等同于[^0-9]

\n：换行匹配符

\f：换页匹配符

\t：匹配制表符

{n}：正好匹配n次，是非负整数；“o{2}”与“Bob”中的“o”不匹配，但与“food”中的两个“o”匹配

{n,}：至少匹配n次

{n,m}：至少匹配n次，最多匹配m次

正则表达式本身的方法：

1：test函数，是正则提供的函数，不是字符串自带的函数

var a="13";

var b=/^13$/;

console.log(b.test(a)); //true/false

不是每次匹配都从0开始匹配的，会记录之前的结果 ，从上次的匹配好的位置后开始匹配

可以加个g标识 全局匹配

2：exec(); 参数是一个字符串，

若没有匹配的文本则返回null，否则返回一个结果数组：

—index 声明匹配的文本的第一个字符的位置

—input 存放被检索的字符串string

注意两点

不能用关系运算符来进行关系判定，在正则中用match()a来判定

var a="13";

cosole.log(a.match(/^13$/)); // 不是null / null

RegExp:

var a=/at/gim; //匹配字符串所有"at"的实例

var a=/.at/gim; //匹配字符串所有"at"结尾的实例

g：表示全局匹配； i：不区分大小写； m：多行模式

十：DOM

1：每一个标签都是一个节点，只要获得了节点就获取了他的所有信息，包括他的子节点及元素的属性，并且可以利用JS对其任意更改

2：关键点——找到要控制的节点

3：节点类型

document——文档节点，代表整个文档，也是根节点

Element——元素节点

Text——文本节点

Attr——属性节点

Comment——注释节点

DocumentType——文档类型节点

DocumentFragment——文档片断节点

文本节点

4：hasChildNodes：判断是否有子节点 true/false

childNodes：返回获得的所有子节点 是一个数组

childNodes.length 获得子节点的长度

5：节点属性

nodeName：节点名字

元素节点的nodeName与标签名相同；属性节点的nodeName与属性名相同；文本节点的nodeName始终是#text

nodeValue：文本节点的值(nodeValue/null)

元素节点的nodeValue是undefined或null；属性节点的nodeValue是属性值；文本节点的nodeValue是本身

nodeType：节点类型 1表示元素节点 2表示属性节点 3表示文本节点

firstChild,lastChild：第一个、最后一个节点

6：获取元素节点

getElementById（“ID”）——取得唯一一个 元素节点

getElenmentsByTagName（"h1"）——取得所有的h1标签 返回一个数组

getElenmentsByClassName（"class"）—取得类名相同的所有元素节点数组

getElenmentsByName（"name"）——取得name属性值相同的元素节点数组

7：增加/创建

createElement("h1")——创建元素节点

createTextNode("hello world")——创建文本节点

parent.appenChild(child)——将子元素添加到父元素

8：删除节点

father.removeChild(list[0]); //获得的是一个数组就要 按数组的方式来操作

删除父元素下的元素节点

9：移动节点——将现有的添加到另外一个地方，那么原来的就没有了，到了新的地方

属性节点

1：属性值都是可以修改的

<div id="main" class="test"></div>

var mainObj=document.getElementById("main");

mainObj.id; //main

mainObj.className; / mainObj.getAttribute("class") //test

img.src; //图片的路径

a.href="http//www.baidu.com"; //修改属性为

2：name属性的获取与改变

getAttribute("name"); //获取name属性的值

setAttribute("name","abc"); //改变或者设置name的值

style：

1：修改样式

main.style.width="140px"; //改变宽度

js修改style一般先在css中写好一个样式，在js中动态修改class或者id

2：外部引入的css属性js不能直接获取？

因为影响div样式的是内嵌的类选择，而通过节点获取的style属性此时并没有设置，所以取到的值为空白字符串，需要先取到最终显示在浏览器上的样式

3：获取最终样式：只能获取值，不能设置值

var styleSheet=document.defaultView.getComputedStyle(mainObj,null) //得到的是整个样式表的对象 不能够通过 .width去改变

mainObj.style.width=parseInt(styleSheet.width)+300+"px"

//改变样式 styleSheet是获得的最终样式

操作节点

1：替换节点——replaceChild(新节点，旧节点) 需要找到被替换节点的父节点

2：插入节点——inserBefore(新节点，需要插入到的节点) 需找到被插入节点的父节点

3：获取下一个兄弟节点——nextSibling

获取上一个兄弟节点——previousSibling

获取父节点——parentNode

操作文本节点的方法

1：删除——deleteData(0,5) 从0号位置开始插入5个

2：加入——appendData("新加入的内容")

3：插入——insertData(1,"插入的内容") 在1号位置后年加入内容

4：替换——rerplaceData(0,5,"要替换的内容") 从0号位置开始替换5个 并加入新内容

5：复制——cloneNode.(false) 浅克隆 克隆标签和属性 所有子节点不克隆

cloneNode.(true) 深克隆 包括内容 全部克隆

6：innerHTML：可以被赋值也可以取出来用； 可以在前正在后加入内容 不适合中间插入

xx.innerHTML="<h1>新的标题</h1><p>新的元素</p>" 原本的节点会被清空

xx.innerHTML+ 保留之前的

添加一个ul li标签

var ulObj=document.createElement("ul");

for(var i=0;i<5;i++){

ulObj.innerHTML=ulObj.innerHTML+"<li>"+i+"</li>"

}

naimObj.appenChild(ulObj); //添加到后面

innerText：没包含标签，，只有内容

xx.innerText="新的内容" 会替换所有的内容，包括标签 不会对标签名解析

outerHTML：外部加入HTML 之前的会不见 可以对标签名解析

在xx.的后面增加