1. Javascript引用

（1）在head中引用

<script src="./src/js/a.js"></script>

（2）添加到标签中

<button onclick="document.write('abc')">button</button>  
<a href="javascript:alert('nihao')">link</a>

（3）添加到body中

<script></script>

1. 弹框

alert("tanchukuang");文字弹框有确定按钮

msg=confirm("是否学过H5？");

同步等待，确定取消框

name=prompt("请输入你的姓名：");

弹出框执行的优先级别较高，输入确定框

1. 文档基本操作

document.write("abc");

他会打开现有html中body部分，重新加入内容，如果是交互操作时，原来有内容，会删除原来的内容

document.body.innerHTML=document.body.innerHTML+"<ul><li>1</li><li>1</li></ul>";设置body标签中的所有html内容

div=document.getElementById("div0");根据id获取当前页面中标签DOM

1. 打印

console.log()

1. 变量常量

变量名规则  
变量名必须要有含义，除非在使用循环等内容需要用到无意义的递进变量时，其他时候变量必须有意义  
变量不能使用中文定义，变量名使用驼峰式命名方法,不能使用关键词和保留字  
临时变量或者参数定义时尽量以下划线开头width,\_height  
驼峰式命名：首字母小写，每个单词大写goodsNewPrice

常量名规定  
必须使用全大写，单词中间使用下划线连接,BUTTON\_CLICK\_VALUE

1. 基本数据类型

String 字符类型  
Number 数值类型 NaN 非数值，数值类型  
Boolean 布尔类型  
undefined 未定义 定义变量未赋值就是未定义  
null 空值 null的类型是对象类型  
Object 对象类型  
Symbol独一无二的类型

var goods={  
 name:"超强笔记本",  
 system:"window 10",  
 price:6000  
};

goods.price=8000;  
goods["price"]=8000;

1. 基本数据转换

三个强制转换  
Number();  
String();  
Boolean();

console.log(Number("a"));// NaN 非数值，数值类型  
console.log(Number(""));// ""强转后变为0  
console.log(Number("5"));//5  
console.log(Number("5a"));//NaN,只要有字符，在转换数值时一定就会变成NaN，不管第一个是数值还是字符  
console.log(Number(true));//true->1,false->0  
console.log(Number(null));//强转null变为0

console.log(String(0xFF));//所有数值都会先运算变为正式数值在强转  
  
console.log(Boolean(""));//false  
console.log(Boolean(0));//false  
console.log(Boolean(undefined));//false  
console.log(Boolean(null));//false  
console.log(Boolean(NaN));//false  
console.log(Boolean(false));//false  
  
强转布尔值时，如果值是""，0，undefined，null,NaN,false这些值时，就会变为false，除此之外强转都是true

1. 堆栈垃圾回收

堆栈溢出

当存储的数据达到某一限制时就会造成堆栈溢出内存泄漏 当不断向堆中存储数据，而不进行清理，这就是内存泄漏

垃圾回收机制

语言中一般分为两种，一种是自动清理，一种是手动清理（gc）。js只有自动清理，

垃圾回收机制就是将引用对中地址的对象设置为null，并且将所有引用该地址的对象都设置为null

不会即时清除，垃圾回收车会根据内存的情况在适当的时候清除堆中的孤儿对象

1. 类型转换

将数值转化为字符，数值类型调用

toString(r) r只能介于2-36之间

toFixed(n) 保留小数点后几位，0-100之间,会四舍五入

将字符串转换为数值，字符串类型调用

console.log(parseFloat("3.145"));//将字符转换为小数  
console.log(parseInt("3.145"));//将字符转换为整数，不做四舍五入

使用parseInt或者parseFloat时，转换根据第0位开始到字符所在位置的前一位，如果第0位就是字符，将转换为NaN

1. 运算

字符串运算  
var b1="a";  
var b2="b";  
var b3=10;  
var b4=true;NaN  
console.log(b1+b2);//字符串的运算，实际是首尾相加ab  
console.log(b1+b3);//现将数值转换为字符，然后在进行字符串相加a10  
console.log(b1+b4);//现将布尔值转换为字符，然后在进行字符串相加在运算中atrue

所有类型自动转换都是隐式转换，都是用强制转换

- \* / % 运算中会将值全部隐式转换为数值，再进行运算  
Infinity 正无穷  
var c1="10a";  
var c2=5;  
var c3="";  
var c4=true;  
var c5="50";  
var c6=null;  
var c7=undefined;  
console.log(c1-c2);//NaN  
console.log(c2-c3);//5  
console.log(c2-c4);//4  
console.log(c5/c3);//Infinity  
console.log(c5-c6);//50  
console.log(c5-c7);//undefined强转数值是NAN

i++ 和 ++i 独立运算时是,运算的结果是完全相同的但是如果将i++或者++i

赋值\打印\获取将运算的先后顺序和赋值先后顺序产生改变如果

i++,变量在前,先赋值,再运算

如果++i,变量在后,先运算,再赋值

1. 位运算符

位非 ~

+1取负

位与运算 &

位或运算 |

异或运算 ^ 相同的是0.不同的是1

密码 ^ 加密串====加密后密码  
加密后的密码 ^ 加密串===密码

左右移位 << >>

console.log(1<<6);//求2的n次幂

1. 关系运算符

== / ===

console.log(x=="3");//自带隐式转换  
console.log(x==="3");//表示判断两者之间值和类型是否都相同

null==undefined  
0==false==""  
NaN与任何内容不同,NaN也不等于NaN

isNaN()判断某个数值是否是非数值

true && true===true// && 两者都必须满足

true || false===true;// || 两者之间有一个满足条件就条件达成

条件 ? 值1 : 值2,如果条件为true,返回值1,如果条件为false,返回值2

s=s || 50; //s=s ? s : 50;熔断处理

判断obj是否是真,如果不是真,就设置为一个空对象  
obj=obj || {};//obj = obj ? obj : {};

1. 条件语句

if(条件){  
 执行语句  
 }  
  
 if(条件){  
 执行语句1  
 }else{  
 否则  
 执行语句2  
 }  
  
 if(条件){  
 执行语句1  
 }else if(条件2){  
 执行语句2  
 }else if(条件3){  
 执行语句3  
 }else{  
 执行语句4  
 }

switch (表达式){  
 case 值1:  
 ...  
 break;  
 case 值2:  
 ...  
 break;  
 case 值3:  
 ...  
 break;  
 default:  
 ...  
}

1. 循环语句

循环  
1、必须要条件初始变量  
2、必须要进入循环的条件  
3、必须要有向条件外发展的表达式

循环语句是瞬间全部执行完成的,在循环过程中所发生的事情都是瞬间的但是如果在循环写点击或者其他的异步方法时,这些操作一定是在循环完成后才开始的

不可以使用千万级以上的循环

break 跳出条件循环外,不再进入循环  
continue 跳出当前循环过程,继续下一次循环  
当执行break和continue时,都不在执行break或者continue后面的语句

do{}while(); 循环,先执行do的内容块,然后才去判断条件

如果条件满足时继续循环,如果不满足就不再循环了  
while()循环是先判断条件才去执行语句块的

for (表达式1;表达式2;表达式3)

表达式1一共运行了1次,循环之前运行  
表达式2是一个或者多个条件,不满足一个就跳出循环,循环多少次就执行多少次,

在运行循环之前先判断条件  
表达式3,循环多少次就执行多少次,循环语句块中所有语句完成后再执行

1. 函数

函数  
function 函数名(参数名1...){  
 语句块  
}

函数也是对象

var fn1=function(){} 匿名函数 将匿名函数赋值给变量,执行变量函数就是执行这个匿名函数 .匿名函数不能再定义该函数之前执行调用  
function fn2(){} 普通函数,可以在定义之前调用函数执行  
js是解释性语言,在当前script标签代码执行的开始阶段,就会将普通函数放入在堆中,只是放入在堆中,但是函数中的内容并没有解析执行,没进行一次函数执行就会解析一次

重名的危险  
1.普通函数和匿名函数重名,匿名函数会覆盖普通函数,原因是普通函数在script标签开始时就将函数放入在堆中,而匿名函数是在代码解析时候才存入堆中的,如果名称相同,就会覆盖原函数  
2.普通函数和变量重名,变量会覆盖普通函数

构造函数定义法

使用全字符串,参数在最前面,后面的字符串内容是执行代码的语句

缺点:

代码会做2次执行,第一次会将函数中的字符串解析成普通代码,第二次执行该代码,因此,效率极低

优点:

可以用任何语言传入该js代码并且执行  
var fn4=new Function("a","console.log(a)");  
fn4(10);

自执行函数

(function () {  
console.log("自执行函数")  
//只能执行一次,执行完成后再也找不到了,变成孤儿对象(有引用的,不可被回收)  
})();

1. 作用域

作用域:某个内容起作用的范围

全局变量  
1.当在script中定义变量,在当前script中的定义前调用是undefined,没有定义该变量时调用报错.原因是在内存中开辟了该变量的存储位置,当代码解释到该语句时才会将值存在栈中,没有执行之前调用该变量就是undefined,如果没有定义就不会再内存中开辟存储位置,因此就会报错  
2.如果在上一个script中调用下面的script的变量时,这时候不但没有值,而且没有开辟存储空间,因此,在上一个script中调用下面的变量永远都是报错的.  
3.一旦定义变量完成后,在后面的script中都是可以任意调用,因为这些变量都是全局变量

全局变量 就是在script标签下直接使用var定义的变量,全局变量可以应用于所有的代码中调用获取.范围非常大

局部变量 就是在函数中使用var定义的变量,仅用于当前函数的执行范围内低耦合(减少耦合度)

函数运行完成后,函数内定义的变量将会自动销毁

函数中修改全局变量后,全局变量就完成了修改,外面调用时,也是修改后的

全局变量和局部变量的重名

如果在函数中定义了某个局部变量名,那么在该函数中所有这个变量都是局部变量

不能通过直接使用变量名的方法调用到外部的全局变量了

window.s1+=10;//如果全局变量的名称在函数中和局部变量名相同,想要调用全局变量时,加上window前缀

var s2=10;  
function fn3() {  
//console.log(s2);//undefined,因为在函数中一旦第一局部变量,该函数中所有位置都是局部变量  
s2+=10;  
console.log(s2);//NaN,undefined+10  
var s2=20;  
}  
fn3();

1. 参数

弱类型语言,参数类型和初始值

参数类型,因为不能固定,因此,如果写给别人用的代码时,一定需要判断输入参数的类型

初始化:在ES5中函数的参数不能设置初始化值(如果执行函数的时候没有输入该参数,希望它有一个默认的初始值),也需要进行设置,ES6中是有初始化参数的,因此ES6中可以直接使用

定义函数中的参数叫做形参

设置函数中的参数值,参数值就是实参

函数参数---回调

参数如果传入的是一个函数名,在当前函数中运行了这个参数,这就是回调  
function fn5(o,fn) {  
 o.a+=1;  
 fn(o);  
}

函数执行自己-递归

function fn8() {  
 if(i===100){  
 return;//跳出函数外  
 }  
 sum1+=i;  
 i++;  
 fn8();  
}

参数如果写在所有参数的最前面,那就是必要参数  
ES5中所有的参数一般都需要按位填写,如果没有填写,就会变成undefined多填进入的参数值不会被直接获取到

function max() {  
console.log(arguments);//函数中就可以获取这个arguments  
var arr=Array.from(arguments);  
return arr.reduce(function (p1,p2) {  
return p1>p2 ? p1 : p2;  
});  
 }  
console.log(max(1,2,3,5,7,9,1,2,5));  
console.log(max(3,6,8,20,10));

function fn2() {  
arguments.callee //当前执行的函数  
console.log(arguments.callee.a);  
console.log(arguments.callee);  
console.log(arguments.callee.caller);//回调当前函数的父级函数  
console.log(arguments)  
}  
fn2.a=10;  
function fn3(fn) {  
fn();  
}  
fn3(fn2);

1. return

return

只能使用在函数中,直接跳出函数,不执行return以后的语句  
break;

只能使用在循环或者switch中,跳出当前循环,不执行break以后语句,并且不再进入函数  
continue 只能使用在循环中,跳出本次循环,当前不再继续执行continue后面的语句,但是继续下一次循环

return 可以返回一个数据给函数运行的外面

如果函数没有使用return或者直接使用return,后面没有值,这时候实际是返回一个undefined

返回参数传入的对象

function setObjProper(obj) {  
 obj=obj || {};//宽模式  
 obj.n=3;  
 return obj;  
}

参数是函数,返回回调函数的结果

function fn2(fn) {  
 var obj={};  
 return fn(obj);  
}

函数中返回一个函数,这叫做闭包

function fn5() {  
 return function () {  
 console.log("aaa");  
 }  
}  
fn5()();

点击  
div.onclick=function () {  
}

按下  
div.onmousedown=function () {  
}

释放  
div.onmouseup=function () {  
}

鼠标移动  
div.onmousemove=function () {  
}

鼠标经过  
div.onmouseover=function () {  
}

滑出  
div.onmouseout=function () {  
}

1. 对象遍历

for in 语句,遍历对象

for(var prop in obj){  
 console.log(prop,obj[prop]);  
}

forEach遍历当前数组,没有返回值,不会返回新数组  
map遍历当前数组,并且将当前数组中的元素返回给新数组,不使用return时,新数组的长度与原数组相同,但是每个元素都是undefined;

var arr=[1,2,4,6,7,8,5,9];  
arr.forEach(function (item,index,arr) {  
 console.log(item,index,arr);  
 })

var arr=[1,2,4,6,7,8,5,9];  
arr.map(function (t) {  
})

var bool=arr.some(function (t) {  
//判断所有元素是否有满足该条件的,  
//如果遇到一次满足则不再继续判断,直接返回true,否则返回false  
return t>10;  
});

判断数组中每一个是否满足条件,如果全部都满足,就返回true,否则返回false  
var bool=arr.every(function (t) {  
 return t>0;  
 });

filter过滤筛选

var arr1=arr.filter(function (t) {  
t++;  
return t>5;//返回的任然是原元素  
});

如果没有initValue值,sum就是第0项的值,item就是第一项值,所以index是从1开始

求和运算  
var s=arr.reduce(function (sum,item,index) {  
console.log(sum,item,index);  
return sum+item;  
},-10);//这里第二个参数时initValue初始值,如果设置这个初始值,index就从0开始

1. 数组

数据结构

数组将无序的数据做有序的排列,存储在一个变量中

原生JS中没有数组类型,原生JS中的数组可以存储多个不同类型的数据(弱类型)  
Array.isArray(arr) 判断arr是否是数组

数组中,标识元素所在的位置,叫做下标,也可以叫做索引

数组中,标识元素的内容,arr[0]这个就叫做下标变量

var arr=[1,2,3,4,5];//字面量创建

var arr1=new Object([1,2,3,4]);//对象构造函数创建

var arr2=new Array(1,2,3,4);//构造函数创建;

console.log(Array.isArray(arr));

判断arr是否是数组,注意typeof arr 值是object  
ar arr2=[1,false,"a",{a:1,b:2}];

这种类型不同的数据存储会造成数组操作时的效率降低

构造函数创建数组时,如果第一个参数是数值类型并且构造函数中只有这一个参数,这个参数就是设置数组的长度,这个长度不能是小数或者负数,否则会报错

如果参数有2个或2个以上,那么第一个元素就不是数组的长度了,而是数组的第0个元素,如果第一参数不是数值类型,且仅有一个,也是作为数组的第0个元素

var arr2=new Array(4);//4是长度

类名:类别的名称  
Array.isArray();//静态方法  
Array.from();//ES6的方法  
Array.prototype.slice.call(div);  
Array.from(div);//将类数组转换为数组

1. 数组的长度

length 数组的长度

数组的长度是可以设置的

将数组的长度设置为0,表示清空数组

如果数组的长度小于原数组长度,意味着将数组从0位截取到指定位置

arr[arr.length-1]=6;//将数组中最后一个元素替换为6  
arr[arr.length]=6;//给数组的尾部添加一个新元素6

使用for循环数组,只能循环数组的每个元素

使用for in循环数组,不但可以循环数组的每个元素,还可以循环到他的属性

arr.a=100;

1. 排序

冒泡排序

外层循环从后向前  
for(var j=arr.length;j>0;j--){  
//内层循环从前向后  
for(var i=0;i<arr.length-1;i++){  
//只判断内层循环的当前位和下一位的大小,互换  
if(arr[i]>arr[i+1]){  
//console.log(arr[i],arr[i+1]);  
var temp=arr[i];  
arr[i]=arr[i+1];  
arr[i+1]=temp;  
}  
}  
}

选择排序

外层循环从头到尾  
for(var j=0;j<arr.length;j++){  
var min=j;//每次初始最小值是当前循环所在的值  
//从当前所在值的下一项到尾部循环  
for(var i=min+1;i<arr.length;i++){  
//比较大小,找出在后面所有最小所在下标  
if(arr[min]>arr[i]){  
min=i;  
}  
}  
//互换当前下标值和最小所在的下标值  
var temp=arr[j];  
arr[j]=arr[min];  
arr[min]=temp;  
}

1. 字符串与对象

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

var str=arr.toString();

1,2,3,4,5

将对象转换为JSON字符串  
var str=JSON.stringify(obj);

将JSON字符串逆转回对象  
var obj1=JSON.parse(str);

利用JSON转换可以深复制

console.log("b" in obj);//属性名是否在对象下

//数组中是否有该下标,不是值  
console.log(3 in arr);

1. 数组操作

push添加元素到数组后面返并回数组长度

var len=arr.push();//返回数组的新长度  
len=arr.push(6);//[1,2,3,4,5,6],6;  
len=arr.push(6,7,8);//[1,2,3,4,5,6,7,8],8;  
len=arr.push([6,7,8]);//[1,2,3,4,5,[6,7,8]],6;

unshift添加元素到数组前面返并回数组长度，减少使用

var len=arr.unshift();  
var len=arr.unshift(0);  
var len=arr.unshift(-2,-1,0);

pop删除数组尾部的元素(1个)  
var t=arr.pop();//返回被删除的元素

shift删除数组的头部元素(1个)  
var t=arr.shift();//返回被删除的元素

join将数值元素相连形成字符串

console.log(arr.join());//arr.toString()相同  
console.log(arr.join("#"));//设置一个符号,用这个符号链接数组的每个元素,形成新字符串  
console.log(arr.join(""));//元素紧密相连

concat拼接数值，返回新的数值原数组不变

var arr1=arr.concat();//没有参数时,是复制数组

var arr1=arr.concat(5,6);//将arr数组链接,5,6两个元素,形成新数组,原数组不变  
var arr1=arr.concat([5,6,7]);//将arr数组和数组[5,6,7]合并形成新数组

var arr1=arr.concat(arr2);//将两个数组链接形成新数组

splice 数组插入删除替换元素,并且返回被删除元素组合新数组

arr.splice();//创建一个新的空数组返回

var arr1=arr.splice(3);//从下标是3开始删除到尾部,返回删除的内容

var arr1=arr.splice(0);//将arr的所有元素导入到arr1中.清空arr

var arr1=arr.splice(0,2);//从arr数组的下标0开始删除2位元素

var arr1=arr.splice(0,0,-1);//在第0位插入一个-1;

var arr1=arr.splice(-1,0,-1);//在第-1位(倒数第1位)插入一个-1;

var arr1=arr.splice(arr.length,0,-1);//在尾部插入一个-1  
var arr1=arr.splice(1,2,-1,-2);//从第1位开始替换两个元素为-1,-2

(location、end) (location、length)(location、length、addContent) return sp

slice()数组截取元素,返回被截取的新元素,原数组不改变  
var arr2=arr.slice();//复制arr的所有元素给arr2,没有引用关系  
var arr2=arr.slice(0);//复制arr的所有元素给arr2,没有引用关系  
var arr2=arr.slice(3);//将数组从下标3开始到结尾截取形成新数组  
var arr2=arr.slice(-2);//将数组从倒数第2位开始到结尾截取形成新数组  
var arr2=arr.slice(3,4);//将数组从下标3开始到下标4截取形成新数组  
var arr2=arr.slice(-2,4);//将数组从倒数第2位开始到下标4截取形成新数组  
var arr2=arr.slice(-2,-1);//将数组从倒数第2位开始到倒数第1位截取形成新数组

(location、end) (location、location) return cp

var arr2=Array.prototype.slice.call(div);//将非数组元素转换为数组

indexOf()

在数组中查找元素,可以根据第二个参数的起始位置开始查找,返回查找到元素的下标,如果没有找到返回-1

console.log(arr.indexOf(2));//从0开始向后查找2  
console.log(arr.indexOf(2,2)); //从下标2开始向后查找2

(searchContent)(searchContent,loaction) rentrn index

lastIndexOf

从后向前搜索,查找到元素的下标返回

1. 倒序和排列

reverse 倒序

原数组倒序改变,返回的新数组和原数组是引用关系  
var arr=[2,3,5,7,9,1,4,6,8,10];  
var arr1=arr.reverse();  
console.log(arr,arr===arr1);

sort();

排序数组中的元素,原数组发生改变,返回的新数组和原数组是引用关系  
sort(function(a,b){})  
/\*由小到大排序\*/  
arr.sort(function (pre,next) {  
 return pre-next;  
});  
/\*由大到小排序\*/  
arr.sort(function (pre,next) {  
 return next-pre;  
});

普通的随机  
 while (arr2.length<4){  
 var num=parseInt(Math.random()\*100);  
 if(arr2.indexOf(num)===-1){  
 arr2.push(num);  
 }  
}

排序的随机乱序  
for(var i=0;i<100;i++){  
 arr2.push(i);  
 }  
 arr2.sort(function () {  
 return Math.random()-0.5;  
 });  
 arr2.length=4;  
 console.log(arr2);

拼音字符和汉字排序  
 arr3.sort(function (pre,next) {  
 return pre.charCodeAt(0)-next.charCodeAt(0);  
 });

编码转字符串

console.log(String.fromCharCode(97));”a”  
console.log(String.fromCharCode(0x4e00));“一”

1. Math方法

console.log(30\*(Math.PI/180))  
console.log(Math.SQRT2);

console.log(Math.abs(-3.14));//绝对值  
console.log(Math.round(3.56));//四舍五入;  
console.log(Math.ceil(3.10));//向上舍入;  
console.log(Math.floor(3.60));//向下舍入;  
console.log(Math.random()\*1000+500);//随机  
console.log(Math.max(3,5,7));//最大值  
console.log(Math.min(3,5,7));//最小值

var arr=[1,3,5,7,9,10];  
var max=Math.max.apply(null,arr);//查找数组中最大的数  
var min=Math.min.apply(null,arr);//查找数组中最小的数  
function randoms(min,max) {//min到max之间的随机数  
 return Math.floor(Math.random()\*(max-min)+min);  
}

console.log(Math.sqrt(2));//开方

console.log(Math.pow(3,2));//指数运算3^2  
console.log(Math.pow(2,6));//2^6  
console.log(1<<6);  
console.log(Math.pow(8,1/3));//8^(1/3)

1. Number

var s1=new Number(10);//构造函数创建的数值对象（object类型）

console.log(Number.MAX\_VALUE);  
console.log(Number.MIN\_VALUE);//小数点后324位  
console.log(Number.NEGATIVE\_INFINITY);//负无穷  
console.log(Number.POSITIVE\_INFINITY);//正无穷

Number.NaN和NaN相同  
NaN!==NaN

var s2=13;  
s2.toString(16);   
s2.toFixed(2);   
console.log(s2.toExponential(2));//2是小数点后几位,用科学计数法表示,转换字符串  
console.log(s2.toPrecision(2));//2是数字是几位,用科学计数法表示(如果刚好,不需要是科学计数法),转换字符串

1. String

var str=new String("b");//str 的类型是object  
str=str+2;//运算法则使str自动转换字符型  
console.log(typeof str);//string

var str1="abc";  
console.log(str1[1]);  
str1[0]="d";//字符串可以通过[索引值]获取这个位置字符,但是不能设置这个位置的字符

str1.length=2;//字符串的length不可修改,只能获取

str.charAt(1);//与str[1]   
str.charCodeAt(1);//获取字符串中指定索引位unicode编码   
String.fromCharCode(87);//将一个unicode转换为字符  
console.log(str.concat("hd"));//合并字符串,+相同

str.indexOf("a");//查找字符a所在位置

str.lastIndexOf();//从后向前查找  
console.log(str.search("a"));//search用于正则表达式查找 1

replace替换字符,可以将字符替换后返回一个替换后的新字符,原字符不变目前我们只能替换一次,如果要替换所有,就需要写循环  
replace多用于正则表达式中的替换  
str=str.replace("a","h");  
console.log(str.toLowerCase());//转换小写 4  
console.log(str.toUpperCase());//转换大写 4

str.slice(1,2);//与数组中的slice完全相同,用于截取字符中某个内容

str.substr(开始位置,要截取字符串的长度)   
console.log(str.substr(0,3));

str.substring(开始位置,截取的结束位置) 和slice不同在与,负数不是代表从后向前数,而是位置0的前面  
console.log(str.substring(1,3));  
console.log(str.substring(3,-1));//从第3个字符串开始向前截取  
console.log(str.substring(5,2));//可以开始位置大于结束位置,这就是逆向截取

split根据分割符将字符串分割为数组,和数组中方法join相对,是join的逆方法 var str="a,b,c,d,e";  
var arr=str.split(",");

1. Date

var date=new Date();   
console.log(date);  
console.log(date.getFullYear());//中国的年  
console.log(date.getMonth());//月份显示是从0开始计算,1就是2月  
console.log(date.getDate());//几号  
console.log(date.getDay());//星期几 0-6 0是星期日  
console.log(date.getHours());//小时  
console.log(date.getMinutes());//分  
console.log(date.getSeconds());//秒  
console.log(date.getMilliseconds());//毫秒

set的所有方法  
date.setFullYear(2020);  
date.setMonth(12);//如果设置12大于了11,就会进1年,当前为1月,2021年1月

var id= setInterval(每次执行函数,多少毫秒执行一次)返回一个id,这个id的作用可以是clearInterval清除该id,这样就停止了这个时间间隔  
clearInterval(id) 清除时间间隔执行函数

var id=setInterval(fn,16);  
var num=0;  
function fn() {  
console.log(id);  
num++;  
if(num>=3){  
clearInterval(id);  
}  
}

var id=setTimeout(需要以后执行的函数,需要多少毫秒以后) 延迟执行函数返回一个id,用来作为清除该延迟使用的  
clearTimeout 清除延迟处理

var id=setTimeout(fn1,4000);  
function fn1() {  
//只要使用setTimeout就一定在函数中使用clearTimeout清除它  
clearTimeout(id);  
}

1. 时间戳

var date=new Date();  
var time=date.getTime();//从1970.1.1 0:0:0到现在的毫秒数  
//永远不会重复

分解代码运行效率 获取代码之间的时间差  
var time=new Date().getTime();  
 for(var i=0;i<10000000;i++){  
 }  
console.log(new Date().getTime()-time);

时间戳

因为每次访问服务器的时间不同,因此,我们可以通过这个时间戳增加在要访问的地址后面这样就不会读取缓存的数据

设置不重复id