1. Bom

window.document;(DOM)  
window是权限最高的对象,他可以找到我们要操作的所有子目标

打开网页  
var a=window.open("http://www.163.com","网易","width=400,height=400");

关闭a网页  
a.close();

a网页弹框  
a.alert("确定")

关闭浏览器  
window.close();

窗口内部宽高,不含工具栏,地址栏等内容,包含滚动条  
console.log(window.innerWidth,window.innerHeight);

当前窗口宽高包含工具等.  
console.log(window.outerWidth,window.outerHeight);

窗口当前的位置(屏幕),只能获取,不能设置  
console.log(window.screenLeft,window.screenTop);  
   
location.href="a.html";//跳转页面,有历史记录  
location.assign("http://www.163.com");//跳转页面,有历史记录  
location.replace("http://www.163.com");//跳转页面,没有历史记录

console.log(location.hash);//获取地址栏中#后面内容   
console.log(location.search);//获取地址栏中?后面的内容  
  
console.log(location.hostname);//域名  
console.log(location.pathname);//目录  
console.log(location.port);//端口号  
 console.log(location.protocol);//协议  
  
相当于设备屏幕宽高  
console.log(screen.availWidth,screen.availHeight);//不带屏幕下面的系统部件  
console.log(screen.width,screen.height);//包含系统部件  
  
console.log(navigator.userAgent);   
console.log(navigator.appName);   
console.log(navigator.appVersion);   
console.log(navigator.platform);

1. 单例类

class Box{  
 constructor(){  
 }  
 static get getInstance(){  
 if(!Box.\_instance) Object.defineProperty(Box,"\_instance",{enumerable:false,value:new Box()});  
 }  
 return Box.\_instance;  
 }  
}

1. DOM

//通过ID获取标签

var div0=document.getElementById("div0");

//false浅拷贝不拷贝子标签，拷贝节点  
var div1=div0.cloneNode(false);

//添加子标签到body中  
document.body.appendChild(div1);

节点是树形结构,不能交叉,因此整个DOM就是一个节点数  
nodeName 如果是标签,nodeName就是标签名  
nodeType 可以用来判断当前是不是文档节点了,文档节点是9  
nodeValue 如果是注释,nodeValue就是注释的内容

document.getElementById("div0");//根据id名获取DOM元素,唯一一个  
var lis=document.getElementsByTagName("li");//根据标签名获取所有的列表  
lis=document.getElementsByClassName("abc");//根据标签中class名获取元素列表  
console.log(lis);//HTMLCollection HTML标签元素列表类型,是类数组,但是不能使用数组的方法  
var radio=document.getElementsByName("ab");//根据表单元素的name获取节点列表

NodeList 节点列表 具有forEach的遍历方法

根据选择器选择元素  
document.querySelector();  
document.querySelectorAll();  
var li1=document.querySelectorAll("#b .abc");//NodeList  
var li2=document.querySelector("[abc=r]");//属性选择器中属性值等于什么全文档查找  
var ul0=document.getElementById("a");

从ul0这个父级元素的子元素中查找className是abc的元素  
var li3=ul0.getElementsByClassName("abc");

1. 节点

console.log(div.childNodes);//所有子节点,包含换行和元素,注释;NodeList  
console.log(div.children);//获取所有子元素,只获取标签;HTMLCollection  
console.log(div.parentNode);//获取父节点  
console.log(div.parentElement);//获取父元素  
console.log(div.firstChild);//获取第一个子节点  
console.log(div.firstElementChild);//获取第一个子元素  
console.log(div.lastChild);//获取最后一个子节点  
console.log(div.lastElementChild);//获取最后一个子元素  
console.log(div.previousSibling);//获取上一个兄弟节点  
console.log(div.previousElementSibling);//获取上一个兄弟元素  
console.log(div.nextSibling);//获取下一个兄弟节点  
console.log(div.nextElementSibling);//获取下一个兄弟元素

1. 节点操作

var text=document.createTextNode("按钮");//创建文本节点;一般用于节点内有其他元素

document.body.innerHTML+="<div></div>";//因为会被覆盖,因此不可以创建多次  
var div=document.createElement("div");//根据标签名创建元素  
document.body.appendChild(div);//把div插入在body的尾部

var elem=document.createDocumentFragment();//文档碎片,为了让更多元素最后在加入到body中  
for(var j=0;j<20;j++){  
var divs=document.createElement("div");  
elem.appendChild(divs);  
}  
document.body.appendChild(elem);

父容器.appendChild(子元素) 如果重新插入已有的元素,相当于移位

父容器.insertBefore(子元素,要插入在谁的前面)  
document.body.insertBefore(elem,ul)  
document.body.insertBefore(elem,document.body.firstElementChild);//插入在最前面

var div1=document.createElement("div");  
div1.textContent="内容";//给标签添加文本内容;节点内没有其他元素,可以直接使用

remove() 自己删自己  
removeChild() 删除子元素

ul.remove();  
var li1=ul.firstElementChild;  
ul.removeChild(li1);//删除第一个子元素

复制当前元素及当前元素的子元素 ,DOM元素深复制 cloneNode(true)仅复制当前元素及元素的属性,DOM元素浅复制 cloneNode(false);

document.body.appendChild(div2.cloneNode(true));//深复制  
document.body.appendChild(div2.cloneNode(false));//浅复制

父容器.replaceChild(新元素,旧元素);

document.body.replaceChild(p,div2);//替换节点

1. history

history.forward();//在有历史记录时,跳转到前一个页面  
history.back();//历史记录必须在通过href或者assign访问的页面中来操作返回,提交的页面中  
history.length;//历史记录的数目,有多少条历史记录  
history.go(0);//刷新页面  
history.go(1);//向前跳转  
history.go(-1);//向后跳转

1. Dom属性

<!-- HTML 标签属性命名,不区分大小,所以全部用小写,单词与单词之间使用横杆连接,标签属性可以看为一个公开属性-->  
<div id="div0" class="divStyle" title="abc" xietian="abc" main-data="abc"></div>

DOM 元素有两个属性 标签属性 对象属性

部分系统的标签属性和对象属性完全相同  
id name type checked title src href style

注意:  
class属性在对象属性中是className

div.a=10;//DOM 对象属性

console.log(div0.className);  
console.log(div0.title);

标签属性

通过js设置获取标签属性

设置属性或者获取属性,属性值都是字符类型

console.log(document.querySelector("[main-data]"));  
console.log(div0.main-data);

//获取标签属性  
console.log(div0.getAttribute("main-data"));  
console.log(div0.getAttribute("xietian"));

设置标签属性 属性名 属性值  
div0.setAttribute("xietian",100);  
div0.setAttribute("boolean",false);  
//删除属性  
div0.removeAttribute("boolean");

DOM 样式

取消原有css中-,将-后面的字母变大写就可以

这种方法是增加行内样式的方法

div0.style.backgroundColor="red";

console.log(div0.style.width)  
console.log(getComputedStyle(div0).width);//除IE外所有浏览器使用这种方法获取计算后样式  
console.log(div0.currentStyle.width);//IE获取计算后样式  
var width;  
try {  
width=getComputedStyle(div0).width;  
}catch (error){  
width=div0.currentStyle.width;  
}  
console.log(width);

1. load和scroll

var img=new Image();  
 img.addEventListener("load",loadHandler);  
 img.src="img/icon01.png";  
 var s=0;  
 s++;  
function loadHandler(e) {  
 console.log(this.width);//微任务和宏任务  
 document.body.appendChild(this);  
 }  
 console.log(s);

滚动条滚动事件  
document.addEventListener("scroll",scrollHandler);  
function scrollHandler(e) {  
 console.log("aaa");  
 }

1. DOM宽高和位置

如果使用overflow:scroll

width-scrollWidth+padding宽度,scrollWidth 17px  
console.log(div.clientWidth,div.clientHeight);//width+padding  
console.log(div.offsetWidth,div.offsetHeight);//width+padding+border

如果使用overflow:hidden

div左padding+里面内容的最大宽度  
console.log(div.scrollWidth,div.scrollHeight);  
//默认body的高度是0,如果给里面放入了标签,内容会被撑开  
console.log(document.body.clientWidth,document.body.clientHeight);

console.log(document.documentElement);//html标签

当前页面文档的宽高 console.log(document.documentElement.clientWidth,document.documentElement.clientHeight)

document 文档,dom根元素  
document.documentElement html标签  
document.body body标签

console.log(div.clientLeft,div.clientTop);//左边线和上边线  
console.log(div.offsetLeft,div.offsetTop);//如果没有使用定位,左边距和顶边距(到页面左上顶部)  
console.log(div.scrollLeft,div.scrollTop);//滚动条位置

如果父容器使用了定位,这个就是该元素到父容器左上角的位置  
console.log(div1.offsetLeft,div1.offsetTop);

如果父容器没有使用定位,这个元素定位了,这个元素到文档左顶部的位置  
console.log(div1.offsetLeft,div1.offsetTop);

1. 距离位置

var div1=document.querySelector("#div1");  
var range=div1.getBoundingClientRect();  
console.log(range);//可以获取到div的范围  
\* x 元素到页面最左边的距离  
\* y 元素到页面最上面的距离  
\* left x  
\* top y  
\* right 元素最右边边线到页面最左边距离  
\* bottom 元素最下边边线到页面最上面的距离  
\* width 元素的宽度 width+border+padding  
\* height 元素的高度  
console.log(range.x,range.y,range.width,range.height);

1. 预加载

var arr=[];  
var num=2;  
//加载第一张图片  
var img=new Image();  
img.addEventListener("load",loadHandler);  
img.src="./img/"+num+"-.jpg";  
function loadHandler(e) {  
//把当前加载进来图片,push到数组中  
arr.push(this);  
//上一次的侦听事件删除  
this.removeEventListener("load",loadHandler);  
//让num++,可以加载下张图片的名称  
num++;  
//如果加载的图片到尾部  
if(num>79){  
//打印所有的图片地址  
for(var i=0;i<arr.length;i++){  
console.log(arr[i].src);  
}  
return;  
}  
//创建新图片,加载这个新图片,并且重新进入当前事件函数中  
var imgs=new Image();  
imgs.addEventListener("load",loadHandler);  
imgs.src="./img/"+num+"-.jpg";

}

var arr=[];  
 for(var i=2;i<80;i++){  
 arr.push("./img/"+i+"-.jpg");  
 }  
getImage(arr,callBackFun);  
function getImage(arr,callback) {  
 var img=new Image();  
 img.arr=arr;  
 img.callback=callback;  
 img.imgList=[];  
 img.num=0;  
 img.addEventListener("load",loadHandler);  
 img.src=arr[img.num];  
 }  
 function loadHandler() {  
 this.imgList.push(this.cloneNode(false));  
 this.num++;  
 if(this.num>this.arr.length-1){  
 this.callback(this.imgList);  
 this.removeEventListener("load",loadHandler);  
 return;  
 }  
 this.src=this.arr[this.num];  
 }

function callBackFun(arr) {  
 for(var i=0;i<arr.length;i++){  
 console.log(arr[i].src);  
 }  
}

1. promise

var promise=new Promise(function (resolve,reject) {  
 var img=new Image();  
 img.src="./img/1-.jpg";  
 img.onload=function () {  
 resolve(this);  
 };  
 img.onerror=function () {  
 reject("加载失败");  
 }  
 });  
promise.then(function (data) {  
 console.log(data,"\_\_\_\_\_\_\_\_\_\_");  
},function (error) {  
 console.log(error,"|||||||||");  
})

1. event loop

同步和异步

同步,是指代码从上向下执行,执行完一条,才去执行下一条,是按照顺序按照步骤的执行  
异步,代码执行需要有一个过程,或者需要一定的时间,或者开始的时间不确定,这时候  
我们先让别的不相关的代码执行,而当前代码当执行完成后去执行一个回调函数

注意:

如果代码写在script中,并且写在函数外部,那么这个代码他只能执行一次,并且是在开始时就同步执行了,显然这种方式不利于代码中出现交互,因此,代码就需要写在

函数中,减少代码之间同步执行方式,函数外通常仅用来定义变量(全局)和执行初始化函数

var sum=0;  
 var bn=document.querySelector("button");  
//所有的事件都是异步  
 bn.addEventListener("click",clickHandler);  
 function clickHandler(e) {  
 sum++;  
 oneFn(sum);  
//return sum;//所有实践函数都不能return  
}

宏任务 setTimeout setInterval

微任务 Promise

同一个队列中,先执行的是宏任务,在执行其他任务,最后执行微任务

在当前队列中出现的异步,如果是微任务就会放在当前任务队列最底端,

如果当前队列出现的异步是宏任务,就会出现在下一个队列最顶端

也就是说在同一个队列中触发异步,微任务先执行,宏任务后执行

1. 事件

javascript是事件驱动语言

事件的作用是触发并且传递数据

bn0.addEventListener("chelema",fn1);//侦听事件  
bn0.dispatchEvent(new Event("chiliema"))  
bn0.addEventListener("click",fn1);  
var evt=new Event("chelema");  
evt.n=3;//事件对象是可以携带一些非系统规定的属性(原event对象的属性不能覆盖)  
bn0.dispatchEvent(evt);//抛发事件  
//同一个对象(抛发的和接收的)只要事件的type相同,那么就会侦听事件接收到  
//console.log(evt);  
//事件函数中只有一个参数,这个参数就是事件对象  
function fn1(e) {  
console.log(e);  
}

var num=1;  
div0.addEventListener("click",clickHandler0);  
function clickHandler0(e) {  
//e.target 目标对象,e.currentTarget和this相同,都是侦听对象  
// console.log(e.target,e.currentTarget,this);  
 setTimeout(function (div) {  
 div.style.backgroundColor="rgba(0,0,255,0.2)"  
 },num\*1000,this);  
 num++;  
}

1. 事件流程

事件触发是由3个阶段组成  
\* 捕获阶段 目标阶段 冒泡阶段  
\* 外 目标 内  
\* | |  
\* 内 外

阻止冒泡 可以在冒泡阶段也可以在捕获阶段  
e.stopPropagation();//阻止冒泡  
e.cancelBubble=true;//IE 8及以下  
IE8 事件  
1，onclick 它不可以一个事件做多个方法，addEventListener可以针对同一对象是一个事件做多个方法,并且可以自由选择中间停止某些方法  
2.onclick不能设置捕获和冒泡阶段那个优先,addEventListener可以设置捕获冒泡的优先,及是否选择确定一个被另一个覆盖  
3.onclick这种事件没有自由度,addEventListener可以侦听所有自定义的事件

IE8及以下浏览器无法通过这个事件选择捕获或者优先的级别  
div0.attachEvent("onclick",clickHandler0);//addEventListener  
div0.detachEvent("onclick",clickHandler0);//removeEventListener

1. 鼠标事件

阻止默认系统事件  
1.阻止提交表单  
2.阻止系统右键菜单(contextmenu事件中使用)  
3.阻止图片拖动时的禁用效果

var bn=document.querySelector("[type=submit]");  
bn.addEventListener("click",clickHandler);  
function clickHandler(e) {  
 console.log(e);  
 e.preventDefault();//阻止默认事件,  
 e.returnValue=false;//IE8 及以下浏览器  
 }

e.x e.y e.clientX e.clientY 鼠标相对文档左上顶部位置(当有滚动条或者滚动条改变时,注意)  
e.pageX e.pageY 鼠标相对页面左上角位置  
e.offsetX e.offsetY 鼠标距离侦听目标的左上角位置  
e.layerX, e.layerY 容器内元素针对侦听目标的左上角位置  
e.screenX e.screenY 针对显示屏的左上角位置  
e.movementX e.movementY 本次鼠标移动距离(mousemove)  
e.ctrlKey e.shiftKey e.altKey e.metaKey 鼠标事件触发时按下了键盘的那个辅助键,是true就是按下了  
div.addEventListener("click",clickHandler);//点击(鼠标左键)  
div.addEventListener("dblclick",clickHandler);//双击  
div.addEventListener("mousemove",clickHandler);//在目标上移动鼠标  
div.addEventListener("mouseover",clickHandler);//鼠标经过 针对e.target 目标(目标改变就会触发)  
div.addEventListener("mouseout",clickHandler);//鼠标滑出 针对e.target 目标(目标改变就会触发)  
div.addEventListener("mouseenter",clickHandler);//鼠标进入针对e.currentTarget侦听的对象  
div.addEventListener("mouseleave",clickHandler);//鼠标离开针对e.currentTarget侦听的对象  
div.addEventListener("mousedown",clickHandler);//按下鼠标(鼠标三键都会触发)  
div.addEventListener("mouseup",clickHandler);//释放鼠标(鼠标三键都会触发)  
div.addEventListener("contextmenu",clickHandler);//右键点击呼出菜单

1. 拖拽

var div=document.querySelector("div");  
 div.onmousedown=function (evt) {  
 var x=evt.offsetX;  
 var y=evt.offsetY;  
 document.onmousemove=function (e) {  
 div.style.left=e.clientX-x+"px";  
 div.style.top=e.clientY-y+"px";  
 };  
 div.onmouseup=function () {  
 document.onmousemove=null;  
 }  
 }

1. event对象

表单中如果表单元素的value值或者选中属性被修改,并且与原来的值不同当失去焦距时触发change事件  
var inputs=document.querySelector("#inputs");  
inputs.addEventListener("change",changeHandler);  
function changeHandler(e) {  
 console.log(e);  
}

selects.selectedIndex 当前选择下拉菜单的索引值  
selects.selectedOptions 是一个列表,当选择下拉菜单时,如果没有指定多选

这个列表中只有一个元素,如果指定多选时,可以有多个元素  
 var selects=document.getElementById("selects");  
selects.addEventListener("change",changeHandler);  
function changeHandler(e) {  
console.log(selects.selectedIndex);  
console.log(selects.selectedOptions)  
console.log(selects.selectedOptions[0].textContent);  
 }

图片,脚本,css样式表文件的加载时,如果指定地址错误加载失败,都可以通过侦听error来判断是否错误  
ajax也可以用来判断通信失败  
 var img=new Image();  
 img.addEventListener("error",errorHandler);  
 img.src="img/1.jpg";  
 function errorHandler(e) {  
 console.log("加载错误");  
 }

Load

加载,用于浏览器全部加载完成判断,图片加载成功判断,视频,声音等加载

ＡＪＡＸ也可以使用ｌｏａｄ判断加载完成

因为script标签比img更早执行完毕  
var img=document.getElementById("imgs");  
console.log(img);//null\*/  
window.onload=function () {  
//当整个页面全部加载完毕,然后图片加载完成,  
// 这个时候在执行这个load完成后的事件就可以获取页面中任意位置的img  
var img=document.getElementById("imgs");  
console.log(img);  
}

submit 和 reset事件都是针对表单form的侦听

通常用于提交表单和重置表单时,需要取消系统默认的提交事件  
var form1=document.getElementById("form1");  
form1.addEventListener("submit",formHandler);  
form1.addEventListener("reset",formHandler);  
function formHandler(e) {  
 e.preventDefault();  
 console.log("aaa")  
 }

页面随着窗口大小的变化自适应缩放内部元素的大小  
resize事件是可以获取window窗口变化缩放的事件侦听,当窗口大小发生改变时,收到该事件  
 window.addEventListener("resize",resizeHandler);  
 function resizeHandler(e) {  
 var scale=document.documentElement.clientWidth/1343;  
 document.documentElement.style.fontSize=100\*scale+"px";  
 }

select当文本框中的值被选中时,激活该事件  
texts.selectionStart texts.selectionEnd 被选中的元素  
var texts=document.getElementById("texts");  
texts.addEventListener("select",selectHandler);  
function selectHandler(e) {  
// console.log(e)  
//console.log(texts.selectionStart,texts.selectionEnd)  
texts.value=texts.value.slice(0,texts.selectionStart)+ texts.value.slice(texts.selectionStart,texts.selectionEnd).toUpperCase()+  
texts.value.slice(texts.selectionEnd);  
}

scroll 滚动条滚动事件(不是滚轮滚动)  
window.addEventListener("scroll",scrollHandler);  
function scrollHandler(e) {  
 console.log("aa");  
 }

1. foucus和input

focus 聚焦  
blur 失焦

input.addEventListener("focus",focusHandler);  
input.addEventListener("blur",focusHandler);  
function focusHandler(e) {  
//console.log(e.relatedTarget);  
//e.relatedTarget 上一次焦点汇聚的元素对象  
 if(e.type==="blur"){}  
 }else if(e.type==="focus"){}  
}

input事件, 输入事件,当改变文本框内容内容时,收到该事件  
var input=document.getElementById("texts");  
input.addEventListener("input",inputHandler);  
e.data 当前输入的内容  
inputType 输入的类型,插入文本,删除文本  
isComposing 是否开启输入法  
function inputHandler(e) {  
console.log(e);  
}

1. 按键事件

document.addEventListener("keydown",keydownHandler);  
document.addEventListener("keyup",keydownHandler);  
e.code 当前按下的键 Key+键名  
e.key 当前按下的键 键名  
e.keyCode 键码值 用这个...

左上右下 37,38,39,40  
function keydownHandler(e) {  
 console.log(e);  
}

1. 滚轮事件

\* 火狐 DOMMouseScroll  
\* 谷歌和IE mousewheel  
\* 滚轮事件  
\* mousewheel  
\* deltaX:-0  
\* deltaY:-100 纵向向上 -100,向下是100  
\* deltaZ:0  
\* detail:0  
\* wheelDelta:120  
\* wheelDeltaX:0  
\* wheelDeltaY:120 纵向向上 120 向下 -120  
\* DOMMouseScroll  
\* detail: -3 滚轮向上-3 滚轮向下3;  
document.addEventListener("DOMMouseScroll",mouseHandler);  
document.addEventListener("mousewheel",mouseHandler);  
function mouseHandler(e) {  
 console.log(e);  
}

1. 正则表达式基础

正则表达式目的在于处理字符串,查找,验证,修改,替换

var reg1=new RegExp("a","g");//构造函数方法  
var reg2=/a/g;//字面量写法

正则表达式方法就2个  
test(); 判断字符串是符合正则表达式条件,返回布尔值  
exec();返回一个数组,包括正则表达式在字符串中符合条件的元素和位置

var reg=/[ec]/g;  
console.log(reg.test("ebcd"));  
console.log(reg.exec("aebcd"));

\* String字符串方法  
\* search 搜索,查找 返回下标  
\* match 搜索查找 返回数组,被查到的所有符合条件的元素  
\* replace 替换 返回被替换的新字符串  
\* split 切割 返回被正则查找到元素切割的数组

console.log(str.search(/[bd]/g));//返回被查找到的下标,返回是第一个被查找到的  
console.log(str.match(/c/));//返回被查找到的所有元素数组,  
// 如果不使用全局查找,作用于正则方法exec相同  
//console.log(str.match(/[bd]/g));//返回被查找到的所有元素数组  
// console.log(str.match(/a/g).length);//可以判断字符被重复的次数  
//console.log(str.replace(/c/g,"f"));//可以完成所有元素的替换,或者不同元素替换同一个元素  
//console.log(str.split(/b/g))  
//console.log(str.split(/[bd]/g));//可以做不同条件同时切割为数组

1. 元字符

组成正则表达式最基础的字符称为元字符

在正则表达式中直接使用字母字符,表示查找这个连续的字符

var str="cataCat";  
console.log(str.match(/cat/));  
console.log(str.match(/cat/g));  
console.log(str.match(/cat/gi));

元字符. 通配符

一个.只能代表一个字符

如果需要使用.字符,而非通配符,就需要使用转义 \.

console.log(str.match(/c.t/g));//.在这里是通配符的作用  
console.log(str.match(/c..t/g));//.在这里是通配符的作用  
console.log(str.match(/c\.t/g))

1. 字符类

\* []  
\* 在中括号中的字符仅可以选择一个字符 可以理解为或的关系  
\* [ac] 表示可以判断字符串是否有一个字符或者一个字符c  
\* [a] 中括号仅有一个字符,那么这个中括号就每有意义了,这个和a字符相同  
\* 一般来说中括号中代表的是两个及以上的字符中任意一个  
\* 可以多个中括号来匹配更多组合  
\* 在中括号中.的作用仅代表字符.不是通配符  
\* 如果在字符类中需要使用[]字符,那就需要使用\[\]来做转义,如果仅使用左[,可以不用转义

\* 在字符类中[]里面不能出现重复字符,因为[]内只代表一个字符,重复是没有意义的  
\* 字符类的范围  
\* 字符范围 [a-z] [A-Z]  
\* 数字范围[0-9]

console.log("asjdkWj12i1D ".match(/[a-z]/g));  
console.log("asjdkWj12i1D ".match(/[a-zA-Z]/g));

console.log("1247985ac".match(/[0-9]/g));

1. 反义和转义

^ 只用在[]中表示,中扩号中的这些之外都可以满足条件

如果^不在中括号的最前面,代表的是^字符

var reg=/[^0-9]/g;//除了0-9的所有字符  
console.log("1098a".match(reg));

转义都代表一个字符  
\w [a-zA-Z0-9\_]  
\W [^a-zA-Z0-9\_]  
\d [0-9]  
\D [^0-9]  
\s 空白字符  
\S 非空白字符

1. 重复

{n} n表示重复次数 n必须大于1,n等于1就是不重复  
{0} 表示匹配空字符

重复范围  
a{2,10} a重复次数不固定,最少重复2次,最多重复10,如果这个重复次数大于10,那么先找出最大为10的重复次数,这种重复范围都属于贪婪匹配.先找到最多的  
a{2,2} 如果最大和最小匹配相同那就是普通的重复  
minutes{0,1} 0次是空字符,1次表示有这个字符,因此,最少就是minute符合条件最多就是minutes符合条件

console.log("minutes".match(/minutes{0,1}/));  
console.log("aaaaa".match(/a{3,}/));//3-无限次  
a{1,} 重复和不重复都行,会筛选最大重复次数

\* {0,} 没有或者重复若干次  
+ {1,} 至少一次以上的重复

? {0,1} 没有或者有都可以

console.log("abcdef".match(/.\*/g));//\* 还有重复0次,所有空字符串也算  
console.log("abcdef".match(/.+/g));//+ 至少重复1次,所以没有空字符串  
console.log("abcdef".match(/.?/g));//? 最多1次,那就意味着只能找到1个字符,每个字符都满足,包括空字符串

1. 非贪婪匹配、选择、起始结束

任何范围性重复,在重复的次数后增加?,就可以实现最小匹配次数

如果该匹配内容前后没有任何内容,就符合重复的最小次数

如果前后有其他内容,中间使用的是非贪婪匹配,就根据实际中

最少能匹配的次数来确定

console.log("asdjasdi".match(/\w{1,8}?/g));//找到8个,不带空字符  
console.log("asdjasdi".match(/\w?/g));//找到9个,带空字符串  
console.log("asdjasdi".match(/\w{2,6}?/g));//找到4次

[]只能或者一个字符  
| 可以或者多个字符  
a|b|c--->[abc]

console.log("catdog".match(/cat|dog/g));

^ $  
^ 写在[]外的作用就是起始,一般用在正则表达式的第一个位置,如果不是第一个位置就做为字符^使用  
$ 写在正则表达式的最尾部作为整个正则判断的结束

console.log("1982-09-10".match(/^\d{4}-\d{2}-\d{2}$/));

1. 分组和紧跟其后

()可以完成分组  
$n是第n个分组,在replace可以作为替换保留原来内容

var str="<ul><li></li><li></li></ul>";  
console.log(str.replace(/(<\/?)(\w+?)>/g,"$1div>"));

?= 紧跟其后是

?! 紧跟其后不是

console.log("abAc".match(/[aA](?=c)/g));  
console.log("abAc".match(/[aA](?!c)/g));

1. 严格模式

"use strict";

a=5;//变量必须先要定义  
function fn1(a,a) {  
//函数参数名不能相同  
}

document.body.style.color="red";  
document.body.style.backgroundColor="#000000";  
with (document.body.style){  
color="red";  
backgroundColor="#000000";  
//不允许使用with  
}

"abc".length=10;//只读属性不允许写入,否则报错

var s1=067;//不允许使用8进制

var arr=[1,2,3,4];  
delete arr.length;\*//不可删除的属性不能删除  
var obj={};  
 Object.defineProperty(obj,"a",{  
 value:10,  
 configurable:false  
 });  
delete obj.a;//不可删除的属性不能删除

delete window.sum;//变量不能删除

eval可以将任何字符串转为对象（不能使用）

eval("div"+i).addEventListener("click",clickHandler);

eval和argument不能重新赋值  
eval=10;  
argument=10

function fn2() {

//不能使用callee,caller  
arguments.callee;  
arguments.callee.caller  
}

var a=10;  
window.a;  
this.a//不允许,ES5的this是window,ES6严格模式this就是undefined

不能使用fn.callee和fn.arguments调用函数堆栈.

建议在ES6 不再使用arguments  
protected static interface 增加三个保留字

1. 代理

let obj={a:1,\_b:2};  
 let obj1={a:1,\_b:2};  
 let handler={  
//设置对象的属性值  
 set:function (target,key,value) {  
 if(key.startsWith("\_")){  
 console.log("不能设置私有变量的值");  
 return;  
 }  
 target[key]=value;  
 },  
//获取对象的属性值  
 get:function (target,key) {  
 if(key.startsWith("\_")){  
 console.log("不能读取私有变量的值");  
 return;  
 }  
 return target[key];  
 },  
//判断属性是否是对象的对象属性  
 has:function (target,key) {  
 return key in target;  
 }  
 };  
 let p=new Proxy(obj,handler);  
 let p1=new Proxy(obj1,handler);  
 p.\_b=20;  
 /\*p是obj的代理，p就可以完成对于obj属性的操作代理也可以完成新属性的添加\*/  
console.log(p.\_b);

1. let和const

let a=10;//定义变量

let 的作用域就是语句块

常量的值和引用地址都是不可以被改变的  
const EVENT\_ID="123";

1. 箭头函数

1.如果没有参数需要小括号  
2.如果有一个参数可以不用小括号  
3.如果有多个参数必须加小括号  
4.如果仅有一条语句,并且返回该内容,(ES5中return) 可以不写return  
5.如果有多条语句,必须加{}将函数内容括起来,否则将会认为是函数外语句,  
并且如果return 就需要增加return语句  
6.箭头函数中this的指向是原代码块的this指向,不受其他执行影响

let div=document.querySelector("div");  
div.onclick=e=>{  
};  
let fn2=(a,b)=>{a\*=a;b\*=b;return a\*b};

let fn1=a=>console.log(a);

1. 解析赋值

解构赋值多个变量赋值

let [a,b,c]=[3,4,5];  
console.log(a,b,c);

默认值  
let [d,e=0]=[5];//5,0;  
console.log(d,e);

函数参数解构赋值  
function fn1([a,b]) {  
 console.log(a+b);  
}  
fn1([3,5]);

数组解构赋值是一一顺序对应

对象解构赋值是属性名称对应

对象解构赋值  
let {sum,a1,b1}={  
 a1:2,  
 b1:4,  
 sum:function () {  
 return a1+b1;  
 }  
};

字符串解构赋值,按照数组结构赋值模式

字符串结构赋值只能完成长度对象解构

let [i1,i2,i3,i4]="abcd";  
console.log(i1,i2,i3,i4);  
let {length:len}="abcde";  
console.log(len);

交换变量  
let s1=10,s2=20;  
[s1,s2]=[s2,s1];  
console.log(s1,s2);

函数中返回多个值

function fn5() {  
 let a=10;  
 let b=20;  
 let c=30;  
//return [a,b,c];//如果数组解构返回,外面的变量可以和里面的不同,但是按照顺序接收  
//return {a:a,b:b,c:c};//如果对象解构返回,外面的变量必须和返回的属性相同  
 }

1. 字符串方法

查找字符,返回布尔值  
console.log("abcdef".includes("c"));

判断是否由某个字符起头  
console.log("abcdef".startsWith("ab"));

判断是否由某个字符结尾  
console.log("abcdef".endsWith("ef"));

让字符串重复多次  
console.log("abc".repeat(3));

判读字符串长度是否达到固定长度,如果没有达到在前面补0  
console.log("abc".padStart(4,"0"));

字符模板  
let a=10;  
let b=20;

console.log(`a:${a},b:${b}`);

函数的参数数组可以直接在函数参数中设置默认值

function fn1(a,b=0,...arg) {  
 console.log(a,b,arg);  
}  
fn1(3,5,7,8,9,0);

1. Symbol

Symbol;//第七种类型

独一无二的类型  
let a=Symbol("a");

console.log(a.toString());  
消除魔术字符串  
const EVENT\_ID="abcde";  
switch (type){  
case EVENT\_ID:  
break;  
}  
const EVENT\_ID=Symbol();  
 switch (type){  
case EVENT\_ID:  
break;  
}

1. set和map

let a=new Set([1,2,3,4,5,7]);  
console.log(a);

arr=Array.from(new Set(arr));//数组去重

let arr=new Set();  
arr.add(10);  
arr.add(20);  
arr.add(30);  
arr.add(10);//添加元素,如果有重复不添加  
arr.delete(20);//没有下标,直接删除元素

console.log(arr.has(10));//判断当前元素是不是该set列表中的元素,返回布尔值  
arr.clear();//删除所有元素

Set没有下标,因此不能使用for循环遍历  
Set也没有属性,因此for in也不可以遍历  
let arr1=[1,2,3,5,6,7];  
for(let elem of arr1){  
console.log(elem);  
}

console.log(arr.size);//可以获取这个类数组的长度  
arr.forEach(t=>{console.log(t)});//可以用forEach遍历Set

let m=new Map();  
m.set("a",2);

console.log(m.get("c"));  
console.log(m.size);

for(let elem of m){  
 console.log(elem[1]);  
 }

for(let elem of m.entries()){  
console.log(elem);//[key,value];  
 }  
for(let key of m.keys()){  
console.log(key)  
}

m.forEach(t=>console.log(t));

1. 生成器函数

生成器函数就是为了完成异步调用等待再继续执行其他内容的方式

function\* abc(n) {  
 yield n  
 let s=5;  
 n+=s;  
 yield n;  
 n\*=n;  
 yield n;  
 return n;  
}

{done:false,value:yield返回值}

while (!v.done){  
 console.log(v.value);  
 v=s.next();  
}

1. promise函数

function getImage(src) {  
 return new Promise(function (res,rej) {  
 let img=new Image();  
 img.src=src;  
 img.onload=function () {  
 res(img);  
 };  
 img.onerror=function () {  
 rej("加载错误")  
 }  
 })  
}

链式异步

getImage("./img/3-.jpg").then(function (data) {  
 console.log(data);  
 return getImage("./img/4-.jpg");  
 }).then(function (data) {  
 return getImage("./img/5-.jpg");  
 }).then(function (data) {  
 return getImage("./img/6-.jpg")  
 })

then有两个参数,都是函数,第一个函数是成功调用函数,第二个函数是失败调用函数

catch 失败后执行方法,有一个参数,这个参数是函数,失败后执行

let list=[];  
 for(let i=3;i<80;i++){  
 list.push(getImage("./img/"+i+"-.jpg"));  
 }

Promise.all 执行多个异步数组,最后返回所有结果组成列表数组

Promise.all(list).then(function (arr) {  
 arr.forEach(t=>console.log(t.src));  
 });

race 赛跑执行多个异步数组,谁先完成异步,返回谁

Promise.race(list).then(function (data) {  
 console.log(data);  
})

1. 异步等待

async function abc() {  
 let s=await a();//同步  
 return "a";  
}  
async function a() {  
 return "ab";  
}