**javaScript基础知识**

**alert() //弹出一个对话窗**

**Debugger 断点**

**<script> </script>**

**window.location.search 有多个search的时候用&号链接**

**delete删除**

**confirm()弹窗 有返回值turn**

**prompt()可输入的弹窗**

**item() 和[] 用法是一样的意思**

**返回最大值c = Math.max ( parseFloat(danjia[i].innerHTML),c);**

**调试： alert() 弹窗 会阻止后续的代码执行**

**console.log() 检查工具中显示**

**console.dir(div) 查看元素自身的细节包括事件 chrom专用**

**console.time（）开始**

**console.timeEnd() 结束 打印每次执行的时间**

**在a标签里 javascript:; 代替# 号 来代替链接**

**获取元素： document.querySelectorAll();**

**1通过id获取元素：document.getElementById文档 获取 元素 通过id**

**2 通过标签的: document.getElementsByTagName （‘div’） （tag标签的意思） 与获取ID的区别：获取到的是一个集合**

**事件:**

**移动端事件：**

**touchstart 相当于mousedown**

**touchmove 相当于 mousemove**

**touchend 相当于 mouseup**

**-1 window.onload 页面加载完成事件**

**0 window.onscroll 滚动条改变事件**

**1 onclick 点击事件**

**2 onmouseover 鼠标移入**

**3 onmouseout 鼠标移出**

**4.onchange 改变事件**

**5.onmousedown 鼠标按下事件**

**6.onmouseup 鼠标抬起事件**

**7 onmousemove 鼠标在元素上移动 事件**

**8 onkeydown 键盘按下事件**

**9 onkeyup 键盘抬起事件**

**10 onfocus 获得焦点 (聚焦) 事件**

**11 onblur 失去焦点 (失焦) 事件**

**12 oninput 当输入框的内容改变的时候出发 事件 不兼容**

**13 oncontextmenu 鼠标右键的点击事件**

**14 ondblclick 双击事件**

**15 onmousewheel 滚轮事件 ie和chrome 他俩兼容**

**16 DOMContentLoaded 当DOM加载完**

**function abc(){} 功能**

**JS的引入：** 1 行间 2内部

3 外部样式表：<script type="text/javascript" src="index.js"></script>

**给元素添加一个点击事件**

**document.getElementById('btn').onclick = function() {**

**点击事件之后，要执行的代码};**

**js 操作元素的样式，可以直接通过操作元素的style属性，操作元素的行间样式**

**在js中元素必须被加载了 才可以被找到**

**onload 当加载完成之后 window.onload 当整个页面加载完成之后**

**加载完成后执行window.onload = function() {代码}**

**js执行顺序：**

**- 从上到下**

**- 从左到右**

**- 赋值 （带等号的）**

**-从右向左**

**变量: 变量用于保存数据 (给比较长的代码起一个简单的名字,用起来方便)**

**使用关键字 var**

**声明变量**

**var 变量名;**

**赋值**

**变量名 = 数据;**

**声明并赋值**

**var 变量名= 数据;**

**不加引号**

**类型： 数字（不加引号） 字符串（加引号）**

**小知识点：**

**注释的写法 1 单行注释 // 2 多行注释/\* \*/ 3 区分大小写 4 Js当中是忽略空格的 所以书写是为了美观 添加应有的空格**

**5 分号的问题**

**赋值语句一定是要加 ; 的**

**函数声明: function name() { }**

**直接调用 name()**

**name()**

**name()写几次就调用几次 不调用 不会自己执行**

**匿名函数：（匿名就是不写名字的函数，暂时匿名只能写在事件上，否则会报错）**

**function (){ }**

**函数调用： 1. 正常调用 name();**

**2. 事件调用 onclick = name;**

**理解事件中的函数**

**-函数名方式**

**-行间事件隐藏函数体**

**Js中的属性读操作：**

**属性操作第一种方法:"."**

**window.onload = function(){**

**var box = document.getElementById('box');**

**var img = document.getElementById('img');**

**var a = document.getElementById('a');**

**box.onclick = function() {alert("今天");};**

**alert(box.id); 获取元素id值**

**alert(box.className); 获取元素class值获取class是写className**

**alert(box.tagName); 获取元素的标签名**

**alert(box.onclick);**

**alert(img.src); href和src获取到的都是绝对路径**

**alert(a.href);**

**alert(box1.innerHTML); 显示闭合标签里的内容包括里面写的标签 };**

**innerHTML**

**box.innerHTML =’dad’ 可以改变里面的内容成为dad**

**alert(box1.style.width);//注意 style操作的是 元素的style属性 也就是行间样式 如果没有获取不到**

**实例：留言板**

**属性的写操作 =**

**var box1 = document.getElementById('box1');**

**box1.style.width = "300px";**

**属性操作的第一种方法： "."**

**属性操作的第二种方法: "[]"**

**注意第二种方法接受的是一个字符串**

**在js中一对引号包起来的字符 都叫做字符串**

**window.onload = function(){**

**var box1 = document.getElementById('box1');**

**//box1.style.width = "300px";**

**box1["className"] = "show"; };**

**window.onload = function(){**

**var box1 = document.getElementById('box1');**

**var attr = "className";**

**box1[attr] = "show";**

**//[]的这种操作可以在里边写一个变量注意变量的类型必须是字符串**

**//box1.attr = "show"; 这是错误的相当于box1."className" = "show"; 格式不对**

**//当属性被赋值到变量上是 用中括号 };**

**在js中不允许出现 “-”，所以去掉“-”首字母大写**

**字符串： 一对引号包起来的一串字符**

**2016-08-30判断**

**字符串：**

**+ 号在Js中有两个作用：**

1. **加法运算**
2. **字符串连接**

**使用+的时候如果碰到字符串的话 就会进行链接操作- 号就是减法运算**

**变量调用的时候都不加引号 加引号会被认为是字符串**

**Var onOff = false; 为判断开关对住没法设定判断值的应用 ！为取反的意思onOff = ! onOff;**

**条件语句: 布尔类型 布尔值（boolean）: true 真、false假**

**If (判断条件){如果（判断条件）条件成立（条件为真）要执行的代码}else{否则条件不成立( 条件为假)要执行的代码}**

**示例：if( 条件){ alert(1) } else{ alert(2) }**

**第一种：只有if没有else**

**if(a > b) {alert("哈哈");}**

**第二种： 有if 有else**

**if(a > b) {alert("哈哈");} else {alert("呵呵");}**

**第三种写法： if else if else**

**if(a > b) {alert("哈哈");} else if( a == b) {alert("呵呵");} else {alert("嘿嘿");}**

**第四种写法： if else if**

**实例： 可以写下拉菜单**

**btn.onclick = function () {**

**if(list.style.display == "block") {list.style.display = "none";}**

**else {list.style.display = "block";} };**

**注意： 1. 行间没有样式，所以第一获取的是个空的**

**2. 第一次判断的时候，“”不管和“none”或者和“block”比较都是一个false（假）他肯定会走else里的值 所以else里要写第一次点击就显示的block**

**所以尽量不要修改行间样式，可以修改className,来调取不同的css样式**

**示例：if(list.className == "show") {list.className = "hide";}**

**else {list.className = "show";}**

**运算符：**

**+ - \* / % (取膜，除法取余数 7%10 位7)**

**加号问题: 运算符 链接符 变量连接操作**

**> 大于、<小于、== 等于、!=不等于、>=大于等于、<=小于等于**

**==为等于号 一个=号在js中是赋值的**

**运算符： 简写**

**Var a = 3; a = a + 5;**

**a += 5;**

**a = a + 1; a++; 自增 适用于 自身加一的 a—适用于自身减一的**

**a++和++a 在赋值的时候的运算顺序不一样 前面的是3 后面的是4**

**box.innerHTML = text.value;直接给innerHTML赋值的话会替换掉里边的内容**

**box.innerHTML = box.innerHTML + text.value; 先拿到现有的inner 然后 加上新的之后，一块放进去**

**可以改变上面等号后面的box.innerHTML位置前后顺序 改变添加进去元素的上下位置**

**Img 、src、 href所有的相对路径 都别拿来做判断**

**Color 颜色值不要拿来做判断**

**innnerHTML 值别拿来做判断 所以可以用开关来当判断控制**

**var isTab = false; 开关，如果没有判断条件，就可以自定义一个开关**

**取膜写法：var now = 0;**

**img.onclick = function() {**

**now = now + 1;**

**img.src = "img2/" + (now%2) + ".jpg";**

**};**

**2016-08-31数组**

**数组应用：**

**array(数组的一个单词一般写arr)**

**var arr = [];可以理解为十一个放数据的仓库 每个数据用，号个开 字符串记得用‘’号 左后一个别放，号**

**有一个length 属性可以调取仓库的长度**

**调取长度 alert ( arr[arr.lenght] );**

**找到最后一个 alert ( arr[arr.lenght-1] );**

**调取用下标调取 alert ( arr[0] ); 下标从0开始**

**有一个push 属性可以 往数组的最后一位添加数据**

**arr.push();**

**数组 []**

**1.数组中可以存放任意类型的数据，但是一般每个数组只存一种类型的数据**

**2.数组中每个数据之间用 ，号隔开，最后的数据不佳， 号**

**下标 arr[下标(number)]**

1. **js中从0开始计数;**

**arr.length 长度 数组中存了多少条数据**

**数组是从第0位开始计数的，所以数组的最后一位是 数组length-1位**

**2016-09-1 for循环**

**获取元素的第二种方法**

**Document.getElementById(‘’) ; 静态方法**

**Document.getElementsTagName(‘’); 动态方法**

**通过标签名字获取元素 前面的document也可以改变成父级的ID**

**This**

**var input = box.getElementsByTagName("input");**

**input.style.background = "red";错误写法 js中不能直接操作一组元素;**

**document.getElementsByTagName("input");**

**在document中，通过标签名字来获取一组元素**

**1.在js中一组元素是不能直接操作的**

**2.一组元素的话，就会有下标[]和length这两个属性**

**3.如果同时操作一组元素的话，用循环来操作**

**for循环：**

**1.初始值**

**2.执行条件**

**3.执行代码**

**4.自增**

**循环语句： for 执行步骤**

**For(var i=0;i<3;i++) {alert(i);}**

**1定义一个初始值i=0 2判断真假 3执行alert(i) 4 走i++**

**再接2着判断、3执行、4走i++ 直到判断为假的时候停止循环**

**死循环 永远不会结束的循环 一定注意不要写死循环**

**例如 for(var i = 0; i < 5; i-=2){ alert(i);}；**

**this 代表发生当前事件的元素**

**Document.title = 123; 可以直接找到这个title 给其直接添加修改内容**

**Document.body.innerHTML = ‘abc’; 直接boyd 就可以找到唯一的body**

**随机点数 parseInt(Math.random()\*64)**

**取整 随机0-1之间的小数**

**cssText 元素行间style中的所有内容**

**this指向(**当前操作的元素**)**

for(var i=0; i<a.lenght; i++ ) {

a[i].onclick = function(){

alert(i)； 结果会显示最后判断为假的数值，因为点击的时候这个循环已经走完了 所以下面的书写就是错误的不能实现的

a[i].style.height = ‘200px’； 这个就错误

this.style.height = ‘200px’;

}

}

**样式操作：**

**-不能直接改css块**

**-可操作元素属性**

**-通过点链接**

**-复合样式的问题（在语法中不能出现- 改变首字母大小写方式）**

**a.style.fontSize = ‘60px’ ； 这种是累加的**

**-cssText**

**a.style.cssText = ‘width:300px; height: 200px;等等’；**

**这种方式 会出现覆盖的问题**

**-className (Js不能直接操作class 通过操作className来操作)**

**在判断中 0 代表false 除0以外的任何数都代表true**

**while循环**

**var n = 7; //**

**while(n >= 0){**

**console.log(n);**

**n--;**

**}**

**2016-09-5 自定义属性**

**document.querySelectorAll();**

**document.querySelector();**

**接收的参数是一个css 选择器**

**document.querySelectorAll() 获取一组元素**

**document.querySelector(); 只获取1个，如果css选择器是 获取到的是一组元素，那就只获取第0个**

**自定义属性，元素没有这个属性，我们自己给他起的新属性，自定义属性**

**一般用index命名 又叫索引值**

**Onselectstart=”” 禁止复制 图片不被禁止 不全 网上再搜搜**

**N%=arr.length**

**2016-9-12 第二章 ECMAScript数据类型、函数传参**

**JS三大组成：**

**ECMAScript：javascript(**标准) **组成语言的语法和基本对象 也是一个解析器**

**DOM：文档对象模型，描述处理网页内容的方法和接口 现在用DOM3**

**BOM：浏览器对象模型，描述与浏览器进行交互的方法和接口**

**ECMAScript 6 -- ECMAScript 2015 最新版本**

**ECMAScript 5.1 现在在用 基本都兼容**

**数据类型： （面试必问 根据typeof 划分 和官方化分 两种）**

typeof 判断数据类型

number 数字 可以运算操作

string 字符串（引号包着的就是字符串） 可以连接

object 对象 – 页面元素都属于对象 可以添加属性

数组也属于对象类型的

Boolean 布尔值 可以用来判断

function 函数类型 可以调用

undefined 未定义类型 不可以进行操作

数字类型没有length 属性

**通过 typeof 在js数据总共分成 6种类型，分别是： number(数字),string(字符串),object(对象),boolean(布尔值),function(函数),undefined(未定义);**

**官方分类：**

**基本（简单）数据类型**

String Number Boolean Undefined Null

String类型 由零个或多个字符组成的串。

\*必须使用单引号对或双引号对引起来

Number类型 为整数或浮点数（小数）

取值范围 +Infinity ~ -Infinity 之间

Boolean类型 为true或false

Undefined类型 为undefined

Null类型 （空对象）类型 为null

**复合（复杂,对象）数据类型**

复合类型是由简单和复合的数据类型组成的。（数组为复杂对象）

**数据类型转换 –隐式转换**

**隐式类型转换:**

**在使用+号的时候，如果碰到了 字符串(或者不能进行 +法运算的时候 ) 就会把 + 的左右两侧 都转换成字符串 然后连接起来**

**true 和 false 在运算中 分别代表 1和0;**

**!号 会把其他类型的数据转换成 布尔值**

**NaN(not a Number) 不是一个数字**

**-,\*,/,%,-=,\*=,/=,%= 这些数据都只能 进行数字运算，所以运算的过程中，会把其他类型的数据 转换成 数字类型**

**parseInt 转换为整数**

**parseInt(要转换的内容) 返回值 转换之后的结果**

**转换规则:从左向右 一位一位，如果碰到了非数字 就把之前的内容返回来**

**问题：**

**parseInt()默认会根据前缀自动选择进制**

**0X -- 16进制**

**0 - f(0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f)**

**parseInt(要转换的内容,进制)** 默认十进制

**parseFloat 转换为浮点型(浮点数,小数)**

**parseFloat(要转换的内容) 返回值 转换之后的结果**

**总左向右 一位一位，如果碰到了非数字就把之前的内容返回来和parseInt相比可以多识别一个小数点**

**Number(要转换的数据) 返回值 数字**

**用Number转换的时候，它会从左向右 看完之后，完全符合数字的规则才去转换，否则返回NaN**

**NaN 非数字**

**类型： number**

**如何判断一个数字 是否是 NaN：**

**NaN 不等于任何数字，并且不等于它自己**

**isNaN(数值); 如果这个数值转成数字之后，是NaN就返回 true 否则返回false**

**Boolean(要转换的数据) 转化为布尔值**

**真：非0的数字,true,非空字符串,非空对象**

**假：0,NaN,false,空字符串,null,undefined**

**== 可以进行类型转换**

**=== 不能进行类型转换**

**!= 不相等 （可以进行类型转换）**

**!== 不相等 （不进行类型转换）**

**并且 && 或||**

**返回最大值c = Math.max ( parseFloat(danjia[i].innerHTML),c);**

**Js在计算小数的时候会出现精度bug 尽量不要拿小数计算**

**0/0 ==NaN 整数/0 == +Infinity 负数/0 == -Infinity**

**2016-9-13 第二章 流程控制**

**一元 a++ 二元 a+b**

**三元运算 三目运算符**

**判断条件 ? 成立要做的事情 : 不成立要做的事情**

**示例： a > 5 ? alert("真"):alert("假");**

**switch (要判断的内容) {**

**case 1: 如果要判断的内容 === 1**

**== 1要做的事;**

**break; //终止一个语句**

**case 2: 如果要判断的内容 === 2**

**== 2要做的事;**

**break;**

**default: 以上条件都不成立的时候**

**都不成立要做的事**

**swicth 穿透 switch有穿透性**

**break和continue 区别：**

**continue; //跳过本次循环**

**break; //终止一个语句**

**for: 可以命名**

**示例： for1:for(var i = 0; i < 3; i++) {**

**for(var j = 0; j < 3; j++){**

**if(i == 1){**

**continue for1;**

**}**

**console.log(i,j,i\*j);**

**}**

**}**

**用for判断小数：**

**var val = input[0].value;**

**for(var i = 0; i < val.length;i++){**

**if(val[i] == "."){**

**alert("请不要输入小数");**

**break;**

**}**

**用for判断最大值:**

**var arr = [23,2,123,34,1];**

**var nub = 0;**

**for(var i = 0; i < arr.length; i++) {**

**if(nub < arr[i]){**

**nub = arr[i];**

**}**

**}**

**alert(nub);**

**用for代替索引：**

**for(var i = 0; i < btn.length; i++){**

**if( this == btn[i]) {**

**// 如果 this 等于 btn[i],那个btn[i]就是当前我们点击的这个 按钮**

**console.log(i);**

**}**

**btn[i].className = "";**

**div[i].style.cssText = "";**

**}**

**while 循环**

**whlie(条件) { 执行代码} 示例: var i=0; while(i<5){ …; i++;};**

**null空对象 不可以做属性操作**

**for in**

**in 判断对象中有没有这个属性 有这个就返回true否则就返回false**

**示例： alert("className" in div); div 的 className属性**

**for in 遍历循环 对象的所有属性**

**//alert("className" in div);**

**for(var s in div){**

**console.log(s,div[s]); //s会依次变成 对象的属性名 s的类型是个字符串 这里的div[s]是查看了div的每个的属性值**

**数组的官方声明方法（了解一下）var arr = new Array(1,2,3,4);**

**函数传参**

**形参： 形式上的参数(函数定义的时候)**

**在定义函数的小括号中，多个参数以逗号分开**

**只能在函数的内部使用**

**实例:** f**unction tab1(btnSelector, divSelector, Event) {};**

**实参： 实际传入的参数(函数调用的时候)**

**调用函数时传入，多个参数以逗号分开**

**实例:** **tab1(".wrap2 input", ".wrap2 div", "onmouseover");**

**注意： 实参和形参一一对应**

**倒叙公式 正数第n个 = 倒数(length - 1 - n)个**

**不定参数**

**arguments 不定参 函数的所有参数的集合 数组形式**

**只能在函数中使用**

**类数组，保存实际传入的参数的集合**

**拥有length属性**

**可用下标获取实际参数中对应的值**

**求所有 参数的和**

**function fn(){**

**var nub = 0;**

**for(var i = 0; i < arguments.length; i++){**

**nub += arguments[i];**

**}**

**console.log(nub);**

**}**

**fn(1,2,3,4,5,6,7);**

**函数返回值: 函数执行之后的结果**

**1. 在函数中，return后边写的内容就是我们的函数返回值**

**当我们需要在函数外边 拿到函数执行的结果的时候，就需要使用 函数返回值；**

**为什么要用函数返回值 --- 方便函数复用**

**函数执行完成后都有返回值**

**不使用return默认返回值为undefined**

**使用return 返回值为return后面的值**

**用返回值 声明变量找到元素 的函数传参**

**function $(selector) {**

**return document.querySelectorAll(selector);**

**}**

**window.onload = function () {**

**var a = $("#div");**

**console.log(a);**

**第二种：**

**function $(selector) {**

**var arr = [];**

**var selectos = document.querySelectorAll(selector);//获取的数组**

**for(var i = 0; i< selectos.length; i++){**

**arr[i] = selectos [i]**

**}**

**return arr.length === 1 ? arr[0] : arr;**

**}**

**return的问题：**

**1. return只能用在函数中**

**2. 在函数中 return 后边的代码 就不在执行了**

**获取计算后样式**

计算后样式，当前元素显示出来的样式

**getComputedStyle(要获取样式的元素)["样式名"];**

**1. 颜色值 拿到的是 rgb样式**

**2. getComputedStyle只能获取 不能设置**

**3. IE6.7.8不支持 getComputedStyle**

**要获取样式的元素.currentStyle["样式名"];**

**1. 不能获取复合样式**

**2. 只兼容IE8包含IE8之前的IE浏览器**

**box.currentStyle 在IE下可以获取到一个对象 在标准下 是undefined**

**getComputedStyle的兼容性 计算后样式**

**function css(el,attr){**

**if(el.currentStyle) {//这是兼容IE的**

**return el.currentStyle[attr];**

**}**

**return getComputedStyle(el)[attr];//这是兼容正常的**

**}**

**//alert(css(box,"width"));//这是调用是输入对应的传参值**

**作用域**

**作用域：一套管理变量的规则**

**作用范围**

**函数，变量**

**全局作用域 ---- 不在任何函数中声明的变量和函数属于全局作用域，全局作用域的内容在任何作用域都可以访问。**

**局部作用域 ---- 在某个函数中的声明的变量和函数属于局部作用域，局部作用域的内容只能在局部作用域和子级作用域才能使用。**

**作用域关系：**

**1、父级作用域不能访问子级作用域**

**2、子级作用域可以访问父级。**

**作用域链:**

**在一个作用域中访问一个变量或者函数时，如果当前作用域没有这个变量或者函数，这时候会去寻找父级的作用域是否有这个变量或者函数，如果有就调用，如果没有就继续向上查找。如果找到全局作用域还没有就会报错。**

**变量预解析：**

**把变量的声明提前到当前作用域的顶部。**

**函数预解析：变量的预解析高于函数的预解析**

**函数表达式走的是变量预解析**

**函数声明走函数预解析----函数预解析把整个函数提升到当前作用域的顶部。**

**与解析是时候 var在前面 函数在后面**

**闭包**

**闭包是一种书写形式，表现为 函数中嵌套函数**

**函数嵌套函数，内层函数可以访问外层函数中的变量（全局变量都可以访问）**

**特型： 1 子函数可以访问父函数的作用域 –作用域链**

**2 提供一个命名空间，避免了全局污染，提升了代码的私有性 --- 避免命名冲突**

**Js垃圾回收机制**

1. **变量都存在哪？ --- 内存**
2. **满足某些条件的时候js 就把存入内存的数据删掉**
3. **关闭页面**
4. **当这条数据不在被调用的时候**

**堆栈**

**先进先出 先进后出**

**递归**

**定时器 setInterval(fn,间隔时间) setTimeout(fn,延迟时间);**

**set 是设置的意思**

**定时器是什么**

**JavaScript中系统提供的**

**setInterval();(系统内置函数) 间隔定时器 每间隔多长时间就执行某件事情**

**setInterval(fn,间隔时间); 两个参数 (fn, 间隔时间)**

**fn 我们要重复做的事情 间隔时间单位：ms**

**1s = 1000ms**

**返回值： 定时器编号**

**setTimeout();(系统内置函数) 延迟型定时器 延时定时器 只执行一次**

**延迟多长时间去做某件事**

**setTimeout(fn,延迟时间);**

**返回值：定时器编号**

**1点击之后，由于定时器第一次执行 是在间隔时间之后执行，所以先手动调用一次，这样体验比较好**

**2 setInterval 第一次执行 也是在间隔时间之后执行的**

**定时器关闭 clearInterval(timer);**

**开启定时器之前，先关闭，确保每次只有一个定时器在执行**

**clearInterval(编号一般是这个变量timer) --- 清除重复执行定时器**

**var timer = 0;设置变量 记录定时器返回的编号**

**timer = setInterval() 执行的时候赋值记录**

**记得在开始执行定时器前 关闭定时器防止 停止效果失效**

**一个元素内分先后进行的计时器 可以用一个变量来记录**

**定时器中的this指向window**

**\_符号代表常量**

**注意在定时器中this指向window**

**要用this 就 var \_this = this; 在定时器外边记录this**

**在定时器里用常量 \_this 代用this**

**Tween 运动**

**t: 执行至 第几次**

**b: 起始值**

**c: 差值(目标点和起始值的差值)**

**d: 执行总次数**

**return 本次应该在的位置**

**// 对象这个类型，是把数据存在他的属性里**

**var left = Tween["bounceOut"](t,b,c,d);**

**// el 元素，attr(要运动的样式),target(目标点),time(时间),type(运动类型)**

**function mTween(el,attr,target,time,type) {**

**有一个Tween和css的封装包**

**linear 匀速**

**easeIn 加速**

**easeOut 减速**

**先加速后减速**

**easeInStrong 二次方加速**

**easeOutStrong 二次方减速**

**easeBothStrong 二次方先加速后减速**

**elasticIn 弹性在开始方向**

**elasticout 弹性在结束方向**

**elasticBoth 弹性 开始和结束都有**

**backIn 回弹 在开始方向**

**backOut 回弹 在结束方向**

**backBoth 回弹 开始和结束都有**

**bounceIn 碰撞 在开始方向**

**bounceOut 碰撞 在结束方向**

**bounceBoth 碰撞 开始和结束都有**

**时间戳**

**function getTime() {**

**var date = new Date().getTime(); //时间戳**

**console.log(date);**

**}**

**for(var i = 0; i < 100; i++){**

**setTimeout(**

**getTime,**

**200); }**

**js本身是单线程操作,定时器除外**

**每开启一个定时器，就相当于开启了一个新的线程**

**For(var s in obj){}**

**for in循环**

**遍历对象需要用到 for in 循环**

**console.log(obj["job"]);**

**for 需要一个数字变量**

**for in 需要一个字符串变量**

**这个字符串会在循环中依次变成 对象的属性名**

**for (var s in obj) {**

**console.log(s); //s为属性名**

**console.log(obj[s]);//这是obj对象的s属性的值**

**}**

**对象的定义**

**obj = { index: 0,**

**name: "付强",**

**age: 18,**

**gender: "mela",**

**job: "FE",**

**abbr: "front-end Web developer"**

**};**

**或**

**var obj = {};**

**obj["name"] = "leo";**

**obj.age = 40;**

**console.log(obj);**

**// && 从左向右执行，先查看左侧是否成立，左侧不成立直接返回false，不在执行右侧，如果左侧成立 在执行右侧代码，如果也成立 返会 true 否则 false;**

**// || 从左向右执行，先查看左侧是否成立，左侧成立直接返回true,不在执行右侧,如果左侧不成立，再执行右侧代码，如果成立 返会 true 否则 false;**

**数组的push的属性 在数组的末尾添加一位**

**实例 srr.push(‘0’) y意思在数组最后一位添加个0字符串**

**日期对象**

**var arr = new Array(); 创建新数组**

**var obj = new Object(); 创建新对象的方法**

**var date = new Date(); 创建时间对象**

**new Date();**

**创建一个跟日期时间有关系的对象 为本地(电脑)时间**

**设置时间日期从大到小(年月日时分秒)否则会出现小小错乱 Js是着行执行的**

**获取时间 获取月份 0 - 11 获取到的月份要加1才是当前月份**

**var date = new Date(); 创建时间对象**

**var year = date.getFullYear(); //获取年份**

**var month = date.getMonth(); //获取月份 0 - 11 获取到的月份要加1才是当前月份**

**var date2 = date.getDate(); //获取日期 1 - 31**

**var day = date.getDay(); //获取一周内的第几天 周日为0**

**var hours = date.getHours();// 获取当前小时数 0 - 23**

**var minutes = date.getMinutes(); //获取分钟数 0 - 59**

**var seconds = date.getSeconds(); // 获取秒钟数 0 - 59**

**var mSeconds = date.getMilliseconds(); //获取毫秒数 0 - 999**

**var time = date.getTime(); //时间戳，当前时间 距离 国际标准时间(1970-01-01 00:00:00) 的时间差 (单位,毫秒)**

**var date = new Date();//获取时间对象**

**var year = date.getFullYear(); //获取年份**

**var month = toDB(date.getMonth() + 1);**

**//获取月份 0 - 11 获取到的月份要加1才是当前月份**

**var date2 = toDB(date.getDate()); //获取日期 1 - 31**

**var hours = toDB(date.getHours());// 获取当前小时数 0 - 23**

**var minutes = toDB(date.getMinutes()); //获取分钟数 0 - 59**

**var seconds = toDB(date.getSeconds()); // 获取秒钟数 0 - 59**

**var day = date.getDay(); //获取一周内的第几天 周日为0**

**var seven = document.getElementsByClassName('seven')[0];**

**//获取星期几的那个img元素**

**var time = year + month + date2 + hours + minutes + seconds;**

**//拼成一串数字**

**function toDB(nub) {//这是设置单双数加零的函数**

**return nub < 10?"0"+nub : ""+nub}**

**一月份=JAN. Jan.=January**

**二月份=FEB. Feb.=February**

**三月份=MAR. Mar.=March**

**四月份=APR. Apr.=April**

**五月份=MAY. May=May**

**六月份=JUN. Jun.=June**

**七月份=JUL. Jul.=July**

**八月份=AUG. Aug.=August**

**九月份=SEP. Sep.=September**

**十月份=OCT. Oct.=October**

**十一月份=NOV. Nov.=November**

**十二月份=DEC. Dec.=December**

**input[i].focus(); 焦点给到当前为空的那个input**

**设置月份的时候要记得减-1**

**设置日期的时候，记得从大到小设置**

**var date = new Date();**

**date.setDate(31);**

**console.log(date);**

**date.setMonth(1);**

**date.setFullYear(2017);**

**offset布局转换 把浮动布局转换成绝对定位布局**

**offsetLeft,offsetTop 元素如果有定位就是和定位父级之间的距离，如果没有定位就是和父级之间距离**

**注意 如果有margin的话 他会把本身的margin也加到定位值里，所以有magin的话下面记得用循环清除一组的margin**

**for(var i = 0; i<nubs.length; i++){//循环加定位值**

**nubs[i].style.top = nubs[i].offsetTop + "px";**

**nubs[i].style.left = nubs[i].offsetLeft + "px";**

**}**

**for(var i = 0; i<nubs.length; i++){//循环加定位**

**nubs[i].style.position = "absolute";**

**}**

**记得先加定位值后加定位，否则会都叠加在一起**

**Math**

**小数位超过十五位JS会出现小数bug**

**取绝对值 Math.abs(nub); return nub的绝对值**

**向上取整 Math.ceil(nub); return nub向上取整的结果1.2 返回值位 2**

**向下取整 Math.floor(nub); return nub向下取整的结果**

**四舍五入 Math.round(nub); return nub小数位四舍五入的结果**

**Math.max(12,23); 取最大值**

**Math.min(12,23); 取最小值**

**随机数 Math.random(nub) return 为 0——1之间的数字 不包含1**

**求平方根 Math.sqrt(nub) 返回nub的平方根**

**X的y次幂 Math.pow(x,y)**

**Math.pow(x,y) 返回 x 的 y 次幂(方)**

**圆周率 Math.PI**

**常用的取整公式：**

**x ~ y : Math.round(Math.random()\*(y-x) + x)**

**0 ~ x : Math.round(Math.random()\*x)**

**1 ~ x : Math.ceil(Math.random()\*x)||1**

**0 ~ x-1 : Math.floor(Math.random()\*x)**

**寻找数组的最大值 Math.max.apply(null,arr)**

**apply 改变this指向的**

**var m = Math.max.apply(null,arr); //求数组中的最大值**

**var n = Math.min.apply(null,arr); // 求数组中的最小值**

**2016-10-11字符串方法**

**字符串方法不会改变原来的字符串内容**

**\ 转义符**

**数据类型 都有toString为转化成字符**

**join把数组转换成字符串**

**示例data.join('');//join把数组转换成字符串**

**模板字符串 对于要替换成变量的字符串 用一对大括号包起来 然后前面写一个$符号 注意：这些字符串放在一对反引号里 ``**

**var str = ` ${h} ${m} ${s}秒`;**

**模板字符串 var str = `当前的时间是${h}小时${m}分钟${s}秒`;**

**str .indexOf(meedle,index) 用于查找字符串，第一个参数是要查找的字符串，第二个参数可选，用于指定从哪里开始查找**

**// 如果找到了返回找到字符的位置，如果没找到返回 -1**

**对应的**

**lastIndexOf() 从后往前查找**

**str.includes() 用来查找指定字符串，找打了返回ture否则false**

**str.startsWith() 用来判断要查找的字符串是否在开头位置**

**str.endsWith() 用来判断要查找的字符串是否在结尾位置**

**//如果在开头或者结尾位置就返回 true 否则 就是false**

**var str = 'zhouyi';**

**str.slice(start,end) 用来截取字符串的，接受两个参数**

**// 起始位 和 结束位 ，返回的结果包含起始位不包含结束位**

**//如果 start > end 返回 ''，如果 start < 0,或者 end < 0**

**// 那么 就等于 从 start + length 到 end + length**

**// 如果start < - length 那么 默认就等于 0**

**// 如果 end > length 那么默认就等于 length**

**var str = 'nieweihao';**

**str.substr(start,nums) 用来截取字符串，第一个是从哪里截取**

**//第二个参数 是截取多少位 如果start < 0 = start + length;**

**str.substring(start,end); 如果 start end < 0**

**// 默认等于0 如果 start > end 会交换位置**

**var str = 'hanzhaoke';**

**str.split(separator,[limit]); 把字符串转化成数组**

**用来分割字符串，第一个参数是用什么来分割字符串，第二个参数是取分割后数组中的前几位（可选）**

**//str.split('',2) // ['h','a']**

**//str.split('adsd') // ['hanzhaoke']**

**//str.split('zhao') // ['han','ke']**

**//str1.toUpperCase().toLowerCase().substr(0,2)//可线性书写**

**str.toUpperCase() == str.toLocaleUpperCase();**

**//讲字符串所有的字母 转化为 大写**

**str.toLowerCase() == str.toLocaleLowerCase();**

**//讲字符串所有的字母 转化为 小写**

**str.charAt(position) if position false = > ‘’;可直接用下标的方法代替**

**//用来查找对应下标的字符 如果是负数 返回 ‘’；**

**str.charCodeAt(position) 用来返回对应字符的Unicode编码**

**/\*var str = '123j4';**

**for(var i=0; i<str.length; i++){**

**if(str.charCodeAt(i) < 48 || str.charCodeAt(i) > 57){**

**alert('不全是数字');**

**}**

**}\*/**

**var str = 'zhaonana'；**

**replace(value,newValue) 用来查找指定的内容，并替换为新内容**

**//var str1 = str.replace('zhaonana','liuxin');**

**// === 》 'liuxin';**

**tirm() 用来去掉字符串的 前后空格**

**var str = ' 121 ';**

**var str1 = str.trim() // 123**

**var str = 'haoge';**

**str.repeat(3) // 'haogehaogehaoge'复制的**

**var str2 = str.repeat(0) // ''**

**//var str3 = str.repeat(-1) //报错**

**总结**

**字符串：**

**substring(start,end(但不包含结束位置))**

**<!--substr(start,num)**

**slice**

**charAt-->**

**replace**

**indexOf**

**<!--lastIndexOf-->**

**toLowerCase**

**toUpperCase**

**split**

**trim**

**2016-10-12 数组详解 object**

**Array.from(v);//类数组转化成数组**

**数组方法都会改变原数组slice()截取不会改变原数组**

**toFixen(nub) 数字方法 参数写几 就保留几位小数 返回值 是字符串**

**数组的 length 属性 不仅能读 还能 写，在设置数组length属性的时候**

**如果 设置 length 小于 原来数组的长度 那么就会截取数组的前 length 个**

**如果设置length 大于 原来的长度，那么原来的数组 就会用对应个数的undefined去填充**

**创建数组的三种方法：**

**1. var arr = [];**

**2. var arr = new Array(l1,l2...) 如果只有一个参数，当这个参数是数字的时候，创建这个参数长度的数组数组中的每一项都是undefined，如果是大于一个参数，那么就是和 [] 是一样的。如果参数是个0 那么就会创建一个 空数组**

**3. var arr = Array.of(1)传一位的话就是数组的第一位是1**

**Array.isArray(arr) 用来判断某个对象是不是数组，如果是返回true如果不是返回 false；**

**arr.indexOf(ele,index) 用来查找数组中的某一项，如果找到就返回对应位置**

**如果没找到 就返回 -1 从前向后查找 index,为可选参数，从第几位开始查找**

**arr.lastIndexOf(ele,index) 是从后向前查找**

**以下方法都会改变原数组**

**arr.push(l1,l2,l3....) 向数组最后面添加新的数据，返回数组新的长度**

**arr.pop() 删除数组中最后一位，返回值是被删除的最后一位**

**arr.unshift(l1,l2,l3...)向数组最最前面添加新的数据，返回数组新的长度**

**arr.shift() 删除数组中第一位，不用参数 返回值是被删除的第一个数据**

**arr.slice(start,end) 用来截取数组，返回值是个数组 包括start 不包括 end，如果start | end < 0就等于 start + length | end + lenght如果 start > end 就返回 [],其它情况返回被截取到的数组**

**arr.splice(0从哪删,1删除多少，‘aaa要添加的内容’)返回值是删除的那个**

**第三个参数选填**

**arr.splice(index,delete\_num,[l1,l2,l3...]) 用来删除或者替换数组中指定位置开始的多少个元素。返回值是被删除的那个数组（内的数据是被删除那个位置的）**

**实例：var arr = ["zhaonana", "hanjun", "hanxu", "hanmengtong", "苏鼎立", "yini", "hanzhaoke"];**

**arr.splice(0,1) // ["zhaonana"]**

**arr.splice(0,1,'zhangmeng') // [ "zhangmeng", "hanxu", "hanmengtong", "苏鼎立", "yini", "hanzhaoke"]; => ['hanjun'];**

**Array.from(类数组) 把一个类数组转换为真正的数组**

**Array.reverse() 用来反转数组**

**arr.forEach(function(item,i,arr){ })用来便利数组中的每一项 接收的参数是函数 是回调函数**

**item 代表 数组中的每一项i 代表每一项对应的下标arr 代表数组本身**

**arr.map(fn) 使用方法和forEach(fn)一样，只不过它接受的函数参数可以有返回值**

**arr.fill(context,[start,end]); 用来填充(替换)数组，第一个参数是要填充的内容**

**后面两个可选参数，从哪里开始填充，到哪里结束，返回值就是填充后修改的数组**

**arr.concat()数组链接**

**实例： var arr1 = [1,2];**

**var arr2 = [3,4];**

**var arr3 = arr1.concat(arr2); // [1,2,3,4];**

**arr.join() 把数组转换为字符串，如果不穿参数，默认用 ‘，’连接如果传了 空字符串 ‘’ 全部连接到一起，如果是其它呢，用其它连接注意：原来的数组不发生改变**

**arr.sort() 数组的排序 如果不传参数 默认按照 数据的 ASC码排序可以接收一个可选的 排序函数**

**实例：**

**arr.sort(function(a,b){ //从小到大**

**return a - b;**

**});**

**arr.sort(function(a,b){ // 从大到小**

**return b - a;**

**});**

**arr.sort(function( ){ //随机排序**

**return 0.5 - Math.random();**

**});**

**排序方法**

**冒泡排序（Bubble Sort）是最易懂的排序算法，但是效率较低，生产环境中很少使用。**

**它的基本思想是：依次比较相邻的两个数，如果不符合排序规则，则调换两个数的位置。这样一遍比较下来，能够保证最大（或最小）的数排在最后一位。**

**再对最后一位以外的数组，重复前面的过程，直至全部排序完成。由于每进行一次这个过程，在该次比较的最后一个位置上，正确的数会自己冒出来，就好像“冒泡”一样，这种算法因此得名。**

**冒泡排序实例：**

**for(var i=0; i<arr.length; i++){//冒泡排序**

**for(var j=0; j<arr.length - i; j++){**

**if(arr[j] > arr[j+1]){**

**var temp = arr[j];**

**arr[j] = arr[j+1];**

**arr[j+1] = temp;**

**}**

**}**

**}**

**for(var i=0; i<arr.length -1; i++){//也是冒泡排序 因为最后 字词是正确的可以不去执行最后一次**

**for(var j=0; j<arr.length - i; j++){**

**if(arr[j] > arr[j+1]){**

**var temp = arr[j];**

**arr[j] = arr[j+1];**

**arr[j+1] = temp;**

**}**

**}**

**}**

**for(var i=0; i<arr.length; i++){//这也是冒泡 装B用**

**for(var j=0; j<arr.length - i; j++){**

**if(arr[j] > arr[j+1]){**

**[arr[j],arr[j+1]] = [arr[j+1],arr[j]];**

**}**

**}**

**}**

**arr.forEach(function(){**

**arr.forEach(function(item,i,arr){ 冒泡 装B写法**

**if(arr[i] > arr[i+1]){**

**[arr[i],arr[i+1]] = [arr[i+1],arr[i]];**

**}**

**});**

**});**

**原生sort()排序原理**

**for(var i=0; i<arr.length;i++){**

**var pre = i - 1,cur = arr[i];**

**while(pre >= 0 && arr[pre] > cur){**

**arr[pre + 1] = arr[pre];**

**pre --;**

**}**

**arr[pre + 1] = cur;**

**}**

**快排的写法：**

**function quickSort(arr) {**

**if(arr.length <= 1){ //停止条件**

**return arr;**

**}**

**var first = arr.shift(); //截取数组第一个**

**var left = []; //比第一个小的**

**var right = []; 比第一个大的**

**while(arr.length){ //循环**

**if(arr[0] < first){ //剩下数组的第0个小于原来第0个**

**left.push(arr.shift())**

**}else{**

**right.push(arr.shift())**

**}**

**}**

**return quickSort(left).concat(first,quickSort(right));//函数递归 链接数组**

**}**

**var newArr = quickSort(arr);//原数组arr变为空了**

**函数的递归：在函数内部调用函数自己**

**实例： var n = 0;**

**function fn() {**

**if（n >= 10）{**

**return 10;**

**}**

**n++;**

**console.log(n);**

**setTimeout(fn, 1000);**

**}**

**fn();**

**------------DOM----------------**

**DOM: Document Object Model (文档对象模型)**

**文档对象模型（Document Object Model，简称DOM），是W3C组织推荐的处理可扩展标志语言的\*标准\*编程接口。在网页上，组织页面（或文档）的对象被组织在一个\*树形结构\*中，用来表示文档中对象的标准模型就称为DOM。**

**通过document对开发者提供了一些操作页面的接口，赋予开发者操作页面的能力。**

**DOM树**

**DOM结构划分：**

**1.按照结构划分：父子级 兄弟级 祖孙级（子孙级）|| （当前节点往上的所有节点都是祖先节点 ||当前节点往下的所有节点都是孙级节点）**

**2.按照节点类型划分：**

**文档中的所有内容都是节点对象。**

**nodeType: 返回（number值）DOM对象上的一个属性。通过这个属性就能查看到某个节点是什么类型的。**

**查看某个节点下的\*所有\*子级（高版本浏览器）：**

1. **obj.childNodes, 查看某个节点下的\*所有\*子级 复合属性（如果选择指定的某个，那么要加下标）**

**注意：在低版本浏览器下(IE9以下浏览器)查看（获取）某个节点下的\*元素\*节点。**

**1 元素节点[所有标签] （数字1）**

**2 属性节点 （比较特殊） (数字2)**

**ul.attributes[0].nodeValue//查看ul的属性节点第一个的名字**

**3 文本节点 （数字3）**

**注意：纯文字、空格、空白文本、制表符都是文本节点。**

**8 注释节点 comment (数字8)**

**9 document (数字9)**

**nodeName: 查看某个节点的名字**

**nodeValue：查看某个节点的值**

**问题：**

**在高版本浏览器下childNodes获取的是所有子节点，会把文本节点和元素节点..都获取到**

**解决：**

**思路：只要元素节点（nodeType == 1就是元素节点）**

**children:获取某个节点下的所有元素节点。（它不是标准，但是所有浏览器都兼容）**

**查找某个节点的父级节点：**

**elem.parentNode**

**如果要查看父级的父级：**

**elem.parentNode.parentNode**

**整个页面的最大的为document**

**总结：**

**nodeType 查看节点类型**

**nodeName 查看节点名字**

**nodeValue 查看节点值**

**attributes 查看属性 //复合属性使用的时候加下标**

**childNodes 获取到某个节点下的所有的子节点 获取到的叫NodeList**

**children :获取到某个节点下的所有的元素子节点。**

**parentNode : 父节点**

**firstElementChild : 第一个元素节点**

**lastElementChild ：最后一个元素节点**

**第一个或者最后一个**

**node.firstElementChild（获取node中第一个元素子节点）**

**node.lastElementChild（获取node中最后一个元素子节点）**

**兄弟节点**

**obj.previousElementSibling (找到某个元素的上一个兄弟元素节点)**

**obj.nextElementSibling (找到某个元素的下一个兄弟元素节点)**

**obj.offsetParent:当前元素最近的定位的父级，如果都没定位默认body为定位父级 示例：div3.offsetParent.tagName**

**offsetParent 正确使用：**

**1.要有定位父级（包括本身也要有定位）**

**2.本元素要有宽、高（一定要能够触发haslayout）zoom:1**

**3.重置字体大小,清除默认样式**

**left：//获取方法**

**1 obj.style.left**

**2 getComputedStyle(obj).left**

**3 obj.offsetLeft**

**obj.offsetTop**

**从元素的左外边框到定位父级的左内边框的距离**

**获取到的值为number类型的是不带单位的。**

**正确使用：**

**1.要有定位父级（包括本身也要有定位）**

**2.本元素要有宽、高（一定要能够触发haslayout）zoom:1**

**3.重置字体大小，清除默认样式**

**在火狐(FF)下，如果父级加了边框同时也加了overfllow:\* 时，火狐会重新计算offsetLeft值 为 当前的left值 - 边框值**

**解决办法：把结构再包一层 把宽度和overflow 拆开给不同元素**

**当前的left值 - 边框值**

**绝对位置**

**带边框和overflow 有问题所以不常用 主要是了解方法**

**var div3 = document.getElementById('div3');**

**var left = 0;//记录绝对位置的变量**

**var obj = div3; //记录要获取绝对位置的元素**

**while(obj) {** **//有边框的话 高版本可以 低版本就不准了**

**left += obj.offsetLeft;// 每次获取的到定位父级的位置和**

**obj = obj.offsetParent ;//改变要找的的元素 为它的定位父级**

**}**

**(元素eles).getBoundingClientRect() 绝对位置 没参数**

**可以获取某个元素到顶边的绝对位置，返回的范围值为对象**

**注意：获取的值会随着滚动条改变。**

**设置属性的3种方法**

**div.id =1;**

**div[‘div’] = 1;**

**obj.setAttribute(key,value) 设置属性（并且是显示在页面中的）:在行间加了一个自定义属性。两个参数 示例：div.setAttribute('index',1);**

**obj.getAttribute(key) 获取行间属性，包括src路径**

**obj.removeAttribute(key) 删除行间的属性**

**总结：**

**上一个兄弟节点： previousElementSibling**

**下一个兄弟节点： nextElementSibling**

**当前元素的定位父级 offsetParent**

**当前元素的左外边框到定位父级的左内边框 offsetLeft**

**当前元素的上外边框到定位父级的上内边框 offsetTop**

**绝对位置 getBoundingClientRect().top**

**设置行间属性**

**setAttribute(key,value) 设置属性**

**getAttribute(key) 获取属性 行间的自定义属性 .方法获取不到**

**removeAttribute(key) 删除属性**

h2.classList 获取元素的class名字 返回类数组 h2.classList .add() 添加class h2.classList .remove() 删除class

h2.classList.contains()判断是否有class名字 返回布尔值 h2.classList.toggle()有就删除没有就添加 都不兼容

**box.hasChildNodes 查看元素下面是否有元素节点 返回布尔值**

**. 的方法写在内存里的 setAttribute 写在文档里的**

**———————data 自定义属性————————**

**dataset 获取行间的自定义属性**

**设置的时候 有的时候 就修改 没有的时候就添加**

**示例：**

**<div data-index="0" data-src="哈哈哈" ></div>**

**box.dataset.index // 0**

**box.dataset['index'] //0**

**console.log(box.getAttribute('data-index'));**

**自定义属性的值得类型是字符串**

**client 获取宽高不包含边框**

**可视区高度:** **document.documentElement.clientHeight**

**可视区宽度:** **document.documentElement.clientWidth**

**用法:** **box.clientWidth**

**offset 可视的意思**

**box.offsetLeft 元素和offsetParent之间的距离**

**box.clientLeft 边框宽度**

**box.left 元素距离可视区左上角的距离**

**offset(可视宽): //可以查看四个值**

**left/top/width/height**

**client: // width获取可是宽高 不包含边框 top是上边框宽度**

**left/top/width/height**

**getBoundingClientRect(): //绝对位置**

**left/top/width/height/right/bottom**

**1 document.createElement("tag") 创建节点元素**

**document.createTextNode(‘’) 创建文本节点**

**特殊的img 可以快速创建 var img = new Image();//创建img对象**

**2 （parent父级元素）.appendChild(el); 在元素内容的末尾添加子元素**

**哪个元素后面.appendDate()添加文本节点**

**3 parentNode.removeChild(childNodes) 删除子元素**

**4 parentNode.insertBefore(newChildNode,oldChildNode); 在一个节点前插入一个指定的节点**

**// 把节点newChildNode 放在oldChildNode前边**

**5 node.cloneNode(boolean) 克隆节点 参数是布尔值 return 克隆好的节点**

**不克隆节点原来的事件**

**替换节点：**

**6 parentNode.replaceChild(xin的, 要提换的)**

**示例：var div = document.createElement("div");**

**body.appendChild(div);**

**body.removeChild(div);或**

**info.parentNode.removeChild(info);**

**list.insertBefore(li,lists[0]);**

**注意:如果元素中需要添加内容，最好实在把元素放入文档之前，把内容添加好**

**appendChild 在某个元里最后面添加进去 一个参数（添加的内容）**

**在操作已有元素时 是剪切行为 并且保留元素身上原有的事件**

**insertBefore 在某个元里选位置添加 两个参数 添加元素，在哪个元素之前添加**

**实例：div2.appendChild(this);**

**示例：div2.insertBefore(this,div2.children[0]);**

**注意appendChild insertBefore在操作一个已有的元素时， 是将已有的元素移动，而不是复制一份进行操作。**

**替换节点：**

**parentNode.replaceChild(xin的, 要提换的)**

**parentNode.replaceChild(newChild,oldChild);**

**示例：div1.replaceChild(strong,this);**

**克隆节点：**

**node.cloneNode(boolean) 参数是布尔值 return 克隆好的节点**

**true：克隆元素和元素包含的子孙节点**

**flase：克隆元素但不包含元素的子孙节点(包含文本节点)**

**示例：var el = this.cloneNode(true);**

**——————————JSON.parse（）————————————**

**json是一种类似JS中的对象格式的字符串**

**//标准中规定 在{}的时候，属性的名字也得用引号包起来**

**JSON.parse(JSONString) json转换成对象 (不能转函数 和undefined)**

**JSON.stringify(object) 对象转换成json (不能转函数 和undefined)**

**使用方法：JSON.parse(elea)**

**注意 json的属性名 必须要加引号，否则转换的时候会出错**

**JSON.parse() 字符串转对象 二个参数没有第三个参数**

**JSON.stringify() 对象转字符串 三个参数：**

**1：放对象的**

**2.回调函数，可以处理数据key为每一项**

**3.为了好看 控制格式的 传入一个数字即可**

**实例： 参数2函数可以替换里面选项的内容，但是格式固定两个return**

**JSON.stringify(obj,function(key,val){**

**if(key == 'name'){**

**return val.replace(/m/g,'M');**

**}**

**//console.log(1111)**

**return val;**

**},1)**

**JSON.parse(json,function(key,val){**

**if(key == 'name'){**

**return 'MoMo';**

**}**

**if(key == 'age'){**

**return 18;**

**}**

**return val;**

**})**

**JSON.stringify(obj,function(key,val){**

**if(key == 'name'){**

**return val.replace(/m/g,'M');**

**}**

**console.log(1111)**

**return val;**

**},1)**

**————————递归的理解—————————**

**// 一个函数执行完之后，在执行下边的代码 --- js本身是单线程的语言**

**//碰到函数执行必须把这个函数执行完 在执行下边的代码**

**函数执行完 在执行下一步**

**// 执行代码写在函数调用前边，先进去，先执行**

**// 执行代码写在函数调用后边，先进去，后执行**

**--------------表格的操作方法-----------------**

**var table = document.querySelector('table');**

**table.tHead 获取表格头部**

**table.tFoot 获取表格底部**

**table.tBodies 获取tbody 因为一个table可以有多个tbody所以获取到是一组元素 使用的时候加下标**

**table.tBodies[0].rows //获取tr**

**table.tBodies[0].rows[0].cells //获取td或th**

**---------------表单的操作方法----------------**

**鼠标的事件不是表单的: onmousedown 鼠标按下事onmouseup 鼠标抬起事件**

**Form.name 通过name获取元素**

**else.onfocus 获得焦点 (聚焦) 事件**

**else.onblur 失去焦点 (失焦) 事件**

**input[1].focus(); // 自动聚焦方法 设置默认聚焦元素**

**input[0].blur(); // 失去焦点方法**

**ele.onchange 改变事件**

**注意onchange 表单的空间的内容发生变化的时候**

**但是input的onchange 发生在失去焦点之后**

**ele.oninput input事件 input的value发生变化(当input在输入内容的时候执行)**

**只属于form 的两种方法:**

**form.onsubmit 表单提交事件 当表单提交的时候触发 里写return false; 阻止提交 用于提交时候的判断验证书写**

**form.submit();提交表单方法 用于提交按钮不是form标签的时候 比如提交按钮是a标签**

**—————————————DOM结束——————————————**

**2016-10-31**

**—————————————BOM开始——————————————**

**也有聚焦 和 失焦事件**

**BOM: Browser Object Model 浏览器 对象 模型**

**BOM 是没有标准的 所以好多方法都不兼容**

**BOM 是关于浏览器的方法，属性，事件**

**window.open(); 方法 打开一个新开窗口 里面有几个参数示例：**

**window.open("http://www.baidu.com","\_blank(\_self)","width=500,height=500");第二个默认为\_blank可以写””不写\_blank**

**window.close() 方法 关闭当前窗口**

**window.navigator.userAgent --- 用户代理信息 判断浏览器时使用**

**window.location --- 地址栏信息**

**onhashchange 哈希值发生变化的时候 的事件**

**//hash 哈希 #号后面的 锚点 不会刷新页面**

**//host 主机名称和端口号**

**//search 提交信息(对有form表单的）？到#号之间的内容(不含#号) 会刷新页面**

**//hostname 主机名称**

**//href 地址，读写属性**

**//origin**

**//pathname**

**window.location.href;**

**字符串版的地址栏信息**

**window.location.search**

**地址栏查询信息 （问号到#号之间的所有内容）**

**window.location.hash**

**锚点信息（#号后面的所有内容）**

**BOM常用属性：**

* + **可视区尺寸 获取可视区宽高 (包含滚动条)**

**window.innerWidth / window.innerHeight**

* + **滚动距离 横向滚动条滚动距离 纵向滚动条滚动距离**

**window.pageXOffset / window.pageYOffset(IE不兼容)**

**window.scrollTo(x,y)//设置滚动距离(兼容的)**

* + **内容高度 : 被内容所撑出来的高度**

**document.body.scrollHeight或obj.scrollHeight**

* + **文档高度**

**document.body.offsetHeight**

**浏览器窗口的大小**

**window.outerWidth**

**几个获取滚动条距离的的兼容性**

**// window.pageYOffset chrome和firefox及IE高版本**

**// document.body.scrollTop //滚动条的距离 chrome**

**// document.documentElement.scrollTop IE和火狐**

**//window.scrollY 用在移动端的**

**示例兼容可以这么写：chrome，IE和火狐**

**var sTop =**

**document.documentElement.scrollTop||document.body.scrollTop;**

**BOM常用事件**

* + **window.onscroll 滚动条改变事件**

**当滚动条滚动时触发**

* + **window.onresize**

**当窗口的大小发生变化时触发**

**例子：登录框+**

**window.onhashchange 哈希值发生变化的时候 的事件 锚点改变事件**

**window.close();**

**//目前FF需要通过open打开的方式才能关闭，别的直接使用即可。**

**document.getElementById('btn').onclick = function(){**

**window.close();**

**}**

**文档输入流:**

**当在别的页面使用了，document.wirt()的时候，会开放文档输入流，**

**如果不关闭，那么会一直等待输入，当注入代码之后会影响JS正常运行，**

**所以，在打开文档输入流，还要关闭：（close()）**

**newOpen.document.write(val);**

**newOpen.document.close();**

**—————————————BOM结束——————————————**

**—————————页面相关尺寸对比—————————**

**关于元素的尺寸的获取：**

**1. getComputedStyle 获取元素的计算后样式 可以直接获取元素的内容宽度，padding和margin等具体值**

**2. offsetWidth/offsetHeight 获取元素的可视宽/高 不含margin含边框**

**3. offsetLeft/offsetTop 获取的是元素距离定位父级的left或top值**

**4. clientWidth/clientHeight 获取元素的可视宽/高 - 边框（不含边框）**

**5. clientLeft/clientTop 获取元素的左边框宽度和上边框宽度**

**6. getBoundingClientRect() 获取绝对位置**

**width height/ 获取元素的可视宽/高**

**left top / 相对于可视区的左侧和顶部的绝对坐标**

**——————————————————————————**

**7. 可视区大小**

**window.innerWidth(包含滚动条宽度)/window.innerHeight**

**document.documentElement.clientWidth(不包含滚动条宽度)/document.documentElement.clientHeight**

**8. 获取滚动距离**

**window.pageYOffset / window.pageXOffset chrome和firefox及IE高版本**

**document.body.scrollTop/document.body.scrollLeft 兼容chrome**

**document.documentElement.scrollTop/document.documentElement.scrollLeft IE和火狐**

**window.scrollY/window.scrollX 用在 兼容移动端的**

**9. 设置滚动距离**

**window.scrollTo(x,y);**

**document.documentElement.scrollTop = document.body.scrollTop = 0;**

**10. 内容高度**

**document.body.scrollHeight(文档高度+margin)**

**ela. scrollHeight(不含元素边框)获取元素的文档高度**

**11. 文档高度**

**document.body.offsetHeight**

**注意：document.body.scrollHeight body的高度如果小于可视区高度，就获取可视区高度，否则获取body的占位高度**

**—————————页面相关尺寸结束—————————**

**———————————数组去重开始——————————**

**var arr = [1,1,5,5,7,8,6,4,7,8]//这种方法最简单set方法**

**var set = new Set(arr);**

**console.log(set)**

**1．这一种只去重 不排序 没有重复次数 推荐**

**var arr = [12,234,23,12,124,3,23,12,3];**

**var re = []; //记录去重后的数组**

**arr.sort(); //不传参对数组排序 目的把一样的都挨在一起**

**re[0] = arr[0]; //把第一个放到新数组里**

**for(var i = 1; i < arr.length; i++){//跳过原数组第一位循环**

**if(arr[i] !== re[re.length-1]){//若剩下不等于刚添加到新数组这项**

**re.push(arr[i]);//就添加到新数组**

**}**

**}**

**console.log(re);**

**2. 这一种只去重了不排序 没有重复次数**

**var arr = [12,234,23,12,124,3,23,12,3];**

**for(var i = 0; i < arr.length; i++){**

**remove(i);**

**}**

**function remove(index){**

**var inner = arr[index]; //记录数组当前这一位内容**

**index = index+1; //把位置变到下一位**

**while(arr.indexOf(inner,index)>=0){//如果这一位存在后面还有这一个数值**

**index = arr.indexOf(inner,index);//就记录这一位内容**

**arr.splice(index,1);//并把重复的这个截取这出来位**

**}**

**}**

**console.log(arr);**

**3. 这个是 去重不排序 记录次数**

**var arr = [12,234,23,12,124,3,23,12,3];**

**var newArr = []; //记录去重后的新数组**

**var obj = {}; //记录重复次数的对象;**

**for(var i = 0; i < arr.length; i++){//新数组查找原数组的每一位**

**if(newArr.indexOf(arr[i]) < 0){//如果查找不到(返回-1)**

**newArr.push(arr[i]);//就把这一位添加到新数组中**

**obj[arr[i]] = 1;//并在对象中记录第一次**

**} else {//否则就是新数组中已经存在,就再次记录次数**

**obj[arr[i]]++;**

**}**

**}**

**console.log(newArr,obj);**

**4. 这个是去重排序记录次数**

**var arr = [12,234,23,12,124,3,23,12,3];**

**var obj = {}; //记录每个数字,在数组中保存了多少个**

**for(var i = 0 ; i < arr.length; i++){**

**if(obj[arr[i]]){如果对象中这个属性存在就记录次数加1**

**obj[arr[i]] += 1;**

**} else {//如果对象中这个属性不存在就记录次数为1**

**obj[arr[i]] = 1;**

**}**

**}**

**console.log(obj);**

**var newArr = [];//创建新数组记录排序去重好的数组**

**for(var s in obj){//遍历对象每一位**

**newArr.push(s);**

**}**

**console.log(newArr);**

**————————————数组去重结束——————————**

**————————————事件——————————**

**window.onload 页面加载完成事件**

**0 window.onscroll 滚动条改变事件**

**1 onclick 点击事件**

**2 onmouseover 鼠标移入**

**3 onmouseout 鼠标移出**

**4.onchange 改变事件**

**5.onmousedown 鼠标按下事件**

**6.onmouseup 鼠标抬起事件**

**7 onmousemove 鼠标在元素上移动 事件**

**8 onkeydown键盘按下事件**

**9 onkeyup 键盘抬起事件**

**10 onfocus 获得焦点 (聚焦) 事件**

**11 onblur 失去焦点 (失焦) 事件**

**12 oninput 当输入框的内容改变的时候出发 事件 不兼容**

**13 oncontextmenu 鼠标右键的点击事件**

**14 ondblclick 双击事件**

**15 onmousewheel 滚轮事件 ie和chrome 他俩兼容**

**16 DOMContentLoaded 当DOM加载完**

**表单内容选中 select(); 选中内容方法**

**event 事件对象 类型:对象类型 在事件里 才能用**

**target 事件源**

**示例：box.onclick = function(event){//event 事件对象**

**console.log(event.target);};**

**keyCode 获取键值 示例：function(e){ console.log(e.keyCode);**

**ctrlKey 按下的如果是 ctrl键 返回true 否则返回false ctrl的键值 17**

**shiftKey 按下的如果是 shift键 返回true 否则返回false shift的键值16**

**altKey 按下的如果是 alt 返回true 否则返回false alt的键值 18**

**clientX //鼠标在可视区的x坐标**

**clientY //鼠标在可视区的y坐标**

**//阻止冒泡**

**box.cancelBubble = true 都适用**

**e.stopPropagation()只有标准浏览器识别**

**示例：span.onclick = function(e){**

**e.cancelBubble = true;//取消冒泡**

**//e.stopPropagation(); 取消冒泡 标准浏览器识别};**

**window.onkeydown = function(e){**

**console.log(e.keyCode);**

**//console.log(e.ctrlKey);**

**//console.log(e.shiftKey);**

**console.log(e.altKey);**

**};**

**键值：左上右下 37-40 shift: 16 crtl: 17 alt: 18 空格 32**

**—**

**注意在鼠标拖拽是 先写鼠标按下 在套进去 document的鼠标移动document的鼠抬起 抬起里在写入 移动和抬起事件 为null 写document的事件是防止鼠标移动过快移除区域 元素的停止跟随**

**——————————————事件————————————————**

**事件： 在用户操作页面的时候，用户为被监听者，浏览器为监听者来监听用户的操作行为，一旦触发了某个行为（onmouseover,onclick…）浏览器回去查看触发的这个行为有没有事件绑定函数，有就执行，否则结果什么也没发生**

**焦点： 当打开页面的时候就已经有焦点了，默认在document上，选中焦点元素时，焦点会从document上到焦点元素身上。当离开焦点元素的时候默认回到document身上。**

**oninput 当输入框的内容改变的时候出发 事件 不兼容**

**显示 获取内容长度 是可以开定时器 去监听valuede 长度 去兼容**

**onchange 当表单控件发生改变的事件 兼容**

**10 onfocus 获得焦点 (聚焦) 事件**

**11 onblur 失去焦点 (失焦) 事件**

**ele.focus(); 自动聚焦**

**input.select()/textarea.select() 选中input或textarea中的内容,也是自动聚焦方法**

**event 事件对象 类型:对象类型 在事件里 才能用 event 就是arguments[0]**

**当以用户操作某个元素的时候，如果该元素有时间绑定函数，那么在这个函数中的第一个形参记录了用户操作似的细节信息**

**它是一个对象**

**clientX //鼠标在可视区的x坐标 clientY //鼠标在可视区的y坐标**

**注意:当页面的滚动宽高度发生变化时这个值依然不变 根据可视区计算的**

**e.pageY 这个值会随滚动条距离变化而变化 常用 好用 根据文档计算的**

**e.pageX**

**keyCode 获取键值 示例：function(e){ console.log(e.keyCode);**

**注意：键值覆盖问题 当按键时，如果在按键的基础上又按了一个键值，这个时候第二次会把第一次的键值覆盖 keyCode只认最后一次的按键键值。组合键时（特殊键除外用定时 器监控 ）**

**特殊按键：**

**ev.shiftKey 键值16**

**ev.ctrlKey 键值17**

**ev.altKey 键值18 他们的默认键值为false 一旦按下 为true**

**带特殊按键 判断时 分情况 要把没用的键子也添加进去 判断他没有按下**

**按键 不松是 会有一下卡顿 浏览器给用户体验的 解决办法:定时器监测抬起**

**2016-11-9 事件：**

**放大镜：原理： 一个大图一个小图，当前移动的距离比上小图的最大移动距离 求出运动比例，右侧大图的位置为 这个比例乘以 大图的最大移动距离（大图的大小减去大图的显示框大小）**

**事件绑定：**

**obj.addEventListener('事件名不带on',事件绑定函数,是否补获[默认false]) 第三个参数可不写**

**注意: 若绑定的函数名一样，那么同样会被覆盖，否则绑定几次就执行几次**

**事件解除：**

**传统的事件解除，将某个元素的事件赋值为null即可**

**事件绑定解除:**

**如果要解除事件绑定的函数，那么在绑定的时候\*第二个参数必须是函数名\***

**解除： obj.removeEventListener('click',fn,false)**

**\*\*\*事件流\*\*\*（事件模型） 面试常问**

**False 为冒泡 true 是捕获 例:obj.addEventListener('click',fn,是否补获[默认false]) 注意：是绑定匿名函数还是函数名**

**事件流：当某个元素事件被触发的时候，从捕获阶段到冒泡阶段的整个过程就叫做事件流（事件模型） 从window起 到window结束**

**冒泡：**

**当在某个元素身上触发了某个事件并且这个元素身上有事件绑定函数，那么不但会执行这个函数，还会从当前元素向上（从父级）执行这个事件，如果向上的祖先节点也有同样事件绑定函数，那么就会执行。一直到window就结束了**

**传统添加事件绑定函数的时候只有冒泡过程可以监听到，捕获过程不能监听到的。**

**捕获： 从window开始，向下传递，直到目标点的这个过程就叫捕获**

**注意：**

**如果事件绑定函数为一个匿名函数，那么有多个相同的事件、元素，那么会依次触发，如果为有名函数，那么只会执行一次。**

**（\*）如果目标点相同事件并且有多个事件绑定函数，那么会按顺序依次执行。**

**如果不是目标点，那么会走事件流机制，先捕获然后冒泡，如果捕获或者冒泡有多个事件绑定函数（同一个事件），那么会按顺序依次执行**

**传统添加事件绑定函数的时候只有冒泡过程可以监听到，捕获的过程不能够监听到**

**冒泡的缺点：**

**//阻止冒泡**

**ev.cancelBubble = true; 阻止冒泡 （不是标准，但所有浏览器都兼容）**

**ev.stopPropagation(); 是标准，但是低版本IE不兼容。阻止冒泡**

**\*\*\*\* 因为冒泡是个向上的过程（一般都是触发了子级的事件绑定函数，不让祖先级的事件绑定函数触发），阻止冒泡一般都在自身设置。**

**冒泡的优点：**

**target（事件监听/事件委托） 事件源：事件触发的源头**

**注意：监听事件源只能监听父级中的子级，父级的兄弟元素是监听不到父级的子元素的**

**事件默认行为：**

**浏览器默认行为：**

**我们没有让它那么做，浏览器偷偷地帮我们做了某些事，这样事就叫默认行为。 或：浏览器的默认行为是指非js添加的默认就有的一些功能**

**阻止默认行为 rerurn false; ev.preventDefault();低版本不兼容**

**解决浏览器默认行为:**

**传统的事件绑定： return false;**

**注意：要解除默认行为，要找到触发浏览器默认行为的源头。**

**事件绑定的情况下 ev.preventDefault();移动端常用 注意:低版本IE不支持。**

**示例：document.addEventListener('mousedown',function(ev){**

**ev.preventDefault();});**

**右键菜单：**

**oncontextmenu 鼠标右键的点击事件 这是一个类似的事件 不是真实事件**

**示例：document.oncontextmenu = function(){return false; }**

**拖拽三大事件：**

**onmousedown（按下document.onmousemove（移动documentonmouseup（抬起 记得阻止默认行为。return false;**

**磁力吸附；还没到顶边就强行拉到顶边。**

**碰撞原理： 排除法：检测碰撞**

**滚轮事件**

**滚轮事件 onmousewheel chrome和IE都兼容**

**ev.wheelDelta 监听上下滚轮的 上正下负**

**上滚动： 120下滚动: -120**

**总结： \*下滚动为负值，上滚动为正值**

**兼容FF火狐的:**

**DOMMouseScroll必须用事件绑定的方式去绑定**

**ev.detail 监听上下滚轮的 上负下正**

**上滚动： -3下滚动: 3**

**总结： \*下滚动为正值，上滚动为负值**

**滚动兼容： 兼容函数的封装**

**git 和 github**

**github.com**

**//账号xiaogang**

**密码renxiaogang123**

**邮箱**

**Username: xiaogang0-0**

**mingzi**

**xiaogang0-0.github.io**

//-------------------------------------------------------------------------

**概念 git工具，版本控制**

**github \*\*\*网站，社交平台，开源项目，远程仓库**

**svn与git区别**

**集成式**

**有个中央服务器，所有的版本都要通过中央服务器进行控制。**

**分布式**

**每个计算机都是台版本控制器，不需要网络就能进行版本控制。**

**github作用： 存储版本的远程仓库**

**git + github**

**git + 码云**

**git + gitlab**

**新建项目：**

**在github上，有个加号，第一个就是新建项目**

**填写项目名称、项目的描述、（勾选）README.md -> 点击创建按钮**

**cd命令进入目录 比如:c/github5/2016-11-16（空文件夹）**

**第一种：**

**cd c:**

**cd github5**

**查看文件：**

**ll 或者 ls 查看**

**cd 2016-11-16**

**第二种：**

**cd c:\github5\2016-11-16**

**小技巧：**

**可以通过tab键自动补全**

**cd .. 返回上一层目录**

**按方向上键重新查找上一次回话输入的命令**

**git clone 远程地址。**

**git的三个区**

**工作区（本地）**

**暂存区**

**作为过渡层**

**避免误操作**

**保护工作区和版本区**

**分支处理**

**版本区（库）**

**在工作区创建一个文件去查看状态（git status） 显示红色**

**将工作区放到暂存区**

**git add 文件名**

**将暂存区放到版本区**

**git commit -m "注释"**

**查看版本：**

**git log(如果出现:号，按q键返回)**

**对比：**

**查看暂存区与工作区的差异：**

**git diff**

**查看暂存区与版本区的差异：**

**git diff --cached**

**查看本地与版本区的差异：**

**git diff master**

**撤销：**

**暂存区的控制撤销到工作区**

**git reset HEAD 文件名**

**工作区代码还原成暂存区或版本区**

**git checkout -- 文件名**

**如果有2个以上文件，一个提交到版本库了，**

**另一个忘记提交，可以先将没提交的文件拉到缓存区，**

**然后通过**

**git commit -m "注释" --amend 撤销回来，**

**最后自动一次性提交暂存区中的所有文件，**

**git log查看是否提交成功**

**小技巧：**

**快速增加多个文件到暂存区：**

**git add .**

**快速版本控制：**

**git commit -a -m "注释"**

**清除屏幕： clear**

**删除：**

**前提本地删除的时候，也想删除暂存区**

**git rm 文件名**

**一次性删除暂存区及本地：**

**git rm -f 文件名**

**不想删除本地，只想删除暂存区**

**git rm --cached 文件名**

**恢复（回滚）：**

**回滚指定文件：**

**git checkout 文件名**

**回滚指定版本文件：**

**git checkout 版本号（5位） 文件名**

**回滚所有整个版本的文件：**

**git reset --hard 版本号**

**git reflog 查看已经操作过的历史记录**

**回滚当前历史记录的上一个记录**

**git reset --hard HEAD^**

**回滚当前历史记录的上几个记录**

**git reset --hard HEAD~2**

**同步远程仓库：**

**git push origin master**

**origin : 远程仓库的名字（默认的）**

**查看远程仓库的名字：**

**git remote**

**git remote -v 可以查看拉取下来的地址和推送的地址是否保持一致。**

**——————————————2016-11-17———————————————**

**设置多人开发权限:**

**1.先进入github项目**

**2.点击 加号 New collaborator**

**3.加开发伙伴**

**4.等待伙伴确认**

**5.有权限的伙伴,克隆地址**

**6.进行开发**

**如果有冲突的情况：**

**1.git fetch 先将远程仓库的代码拉取下来**

**2.git diff master origin/master 查看版本库与远程的差异（冲突）**

**3.git merge origin/master 版本库与远程的代码进行合并**

**4.手动修改冲突**

**5.修改完之后，版本控制 -> push**

**如果没有权限还要参与别人的项目：**

**1.fork下别的人项目**

**2.自己仓库会多一份fork到的项目**

**3.git clone 项目**

**4.该修改就修改，该push就push**

**5.创建一个new pull request**

**6.等待别人回复**

**当收到别人发来的修改请求：**

**1.先查看别人修改了什么？**

**2.如果合适就合并**

**点击 merge pull request按钮进行合并**

**分支： 默认为matser分支**

**查看分支： git branch (当前分支会有\*号)**

**创建分支： git branch 分支名**

**切换分支: git checkout 分支名**

**快速创建并且切换 git checkout -b 新的分知名**

**git branch -d 分支名 （合并完分支后删除）**

**git branch -D 分支名 （强制删除没有合并的分支）**

**git merge 分支名 （合并分支）**

**git branch --no-merged （查看没有合并的分支）**

**查看当前主干上有多少个分支已经合并: git branch --merged**

**//----------------------------------------------------------------**

**git:工具，版本控制**

**github :网站，社交平台， 开源项目，远程仓库**

**svn 与git**

**svn 有中央服务器 集成式 需要有网络**

**git**

ssh-keygen -t rsa -C "361204134@qq.com"

git config --global user.name

git config --global user.email

cd c:

git clone 后面写地址

cd .. 返回上一层目录

git status 查看有哪些文件

**git add 文件名 放到暂存区**

**//git chea**

git commit -m "名字" 放到版本区

git log 查看版本（如果出现：号 按q键返回）

git diff 暂存区和本地对比

git diff -–cached 暂存区和版本区的差异

git diff master 查看本地与版本区的差异

撤销

git reset HEAD 文件名 暂存区撤销到本地区

git checkout –-（空格）文件名 工作区代码还原成 暂存区 或版本区

**git add . 把所有文件添加到暂存区**

**git commit –a –m “”快速添加到版本区**

**删除**

**git rm 文件名 本地已经删除 现在是删除暂存区的**

**git rm –f 文件名 删除本地和暂存区的 一次性删除**

**git rm –cached 文件名 只删除暂存区不删除本地**

**恢复(回滚)：操作版本区**

**git checkout 版本号（前五六位就行）文件名**

**git reset –hard 版本号 回滚整个版本的文件**

**git reflog 查看已经操作过的历史记录**

git remote 查看远程仓库的名字

git remote –v 可以查看拉取下来的地址 和推送的地址 是否保持一致

git push origin master 同步到远程仓库啊 最新版本 （文件夹的）

**粘贴 用Shift + Insert**

**工作区(本地) 暂存区 版本区**

**q 返回操作区域**

**设置多人开发权限**

**git fetch 查看冲突 先将远程拉取下来**

**git diff master origin/master 查看版本库与远程的差异（冲突）**

**git merge origin/master 版本库与远程的代码合并**

**手动修改冲突**

**版本控制 再 push**

**git clone 远程地址。**

**git add 文件名**

**git commit -m "注释"**

**查看版本： git log**

**git push origin master**

**git branch 分支名 创建分支**

**git branch 查看分支当前分支会有\*好**

**git checkout 分知名 切换分支**

**git checkout –b 新分支名 并切换到新分支（快速创建切换）**

**git branch –d 分支名 删除分支**

**git branch –D 强制删除 没有合并时**

**合并分支 先进入主分支**

**git merge 分知名 合并分支**

**-------正则—2016/11/28--------**

**字符串的下标在IE6下不兼容所以用str.charAt(i)方法代替**

**正则的特性：**

**懒惰：匹配到某个规则之后，那么直接返回。**

**贪婪:会一直找量词中匹配的字符，有多少就匹配多少。**

**charAt 获取某个字符**

**search 查找**

**substring 获取子字符串**

**charAt 获取某个字符**

**split 分割字符串，获得数组**

**str.split(separator,[limit]); 把字符串转化成数组**

**用来分割字符串，第一个参数是用什么来分割字符串，第二个参数是取分割后数组中的前几位（可选）**

**arr.join() 把数组转换成字符串**

**正则：主要用来匹配（擅长匹配模糊范围的）字符串，是一种规则，效率更高**

**速度更快，但是，可维护性较差（因为读起来很费劲）**

**查找匹配某些符合规则的字符串，正则表达式就是写这种规则的工具**

**写法一：简写：/规则/ (/a/I ) 这种性能要更高。**

**写法二： new RegExp(“字符串的规则”,修饰符);**

**转义符： \+字符，如果这个组合在系统中已经有默认定制，那么就为特殊的转义符 (转义符后面跟上一个普通的字符，代表一个或一段特殊的内容)**

**\n 换行 \r 回车 \t 制表符**

**正则的方法：**

**1.test查看正则表达式与指定的字符串是否匹配,如果是返回布尔值（true），否则false 写法：（re正则）re. test(str) \*正则身上的方法。**

**2.match：将匹配到的字符放到一个数组中。 返回数组**

**字符串的方法: str.match(正则)**

**3 search:找到字符串中指定字符(正则)的位置,找到返回位置，没找到返回-1**

**字符串的方法： str.search(''||re);**

**4 replace()：将字符串指定的字符（正则）替换成另外的字符。**

**字符串的方法: str.replace(要替换哪个||正则,替换成什么||回调函数);**

**replace如只传一个参数，那么匹配到的字符就会被替换为undefined**

**写回调函数注意：**

**1回调中必须写return（其实是要替换成的值）不然会为替换成undefined**

**2回调函数形参**

**$0就为正则每次匹配的字符**

**$1就为每次匹配到字符的位置。index**

**$2整个字符串 all**

**$3 undefined 后面的参数都是未定义。**

**子项 () 子项的意思 从左往右依次计算**

**只要在正则有子项，那么replace回调函数中的第二个参数开始，就是匹配到的子项（不再是索引）**

**示例: $1,$2,$3 分别为子项1子项2子项3 $4就是all $5 undefined**

**var s = str.replace(/(\d+)\D+(\d+)\D+(\d+)\D?/,function($0,$1,$2,$3){**

**return $1+'年'+$2+'月'+$3+'日'; });**

**() 子项、提权 括号除了子项，还有提权**

**()\1 重复子项**

**重复子项： e\1 -> ee e\1+ -> eeeeeeee**

**(\d) -> 5 (\d)\1 -> 55**

**$0:匹配到的字符**

**$1:子项**

**重复**

**示例：var str = '55555qqq111';//找到重复的数字**

**str.replace(/(\d)\1+(\D)\2+(\d)\3+/,function($0,$1,$2,$3){**

**console.log($3); });**

**修饰符：修饰符：不分前后顺序**

**g -> global 全局全局匹配**

**i -> ignore 忽略大小写**

**量词：**

**+ {1,}最少有1个，最多不限 量词变化：\d \d\d和\d+**

**? {0,1}最小可以没有，最大有一个**

**{n,m} 最小n个，最大m个**

**{n,} 最小多少，最多不限**

**{n} 正好n次**

**\* 任意次 {0,}**

**{0,} 最小可以没有，最大无限**

**注意：**

**量词都是修饰前面那个规则的**

**比如： a\* a可以没有也可以无限**

**中文的区间范围 中文编码： /[\u4e00-\u9fa5]/**

**特殊的转义符： \d [0-9] 一个数字**

**\D [^0-9] 非数字**

**\n 换行**

**\w [a-z0-9\_] 英文数字下划线**

**\W [^a-z0-9\_] 除了英文数字下划线**

**\s 空格**

**\S 非空格**

**.（点）——任意字符(\n除外)**

**\b 单词的边界符 只有\w [a-z0-9\_] 英文数字下划线 有单词边界**

**\B 非边界符**

**\1\2\3 重复子项**

**[]里面若干个找任意一个**

**^ 开头 $ 结尾**

**^ 如果在正则开头写：为开头的意思**

**$ 如果在正则要结束的位置写：为结尾**

**如果 [] + ^ 那么写就为 排除 排除 [^a] 除了以外**

**new RegExp 问题：**

**字符串中也不能随便写\，如果非要写，就转义\**

**new RegExp(必须为字符串)**

**\*\*\*如果在字符串中有\必须转义。**

**str.trim()去字符串开头空格 不兼容IE 可以用 下面的正则**

**str.replace(/^\s+|\s+$/g,'') 去开头结尾空格**

**tirm 去空格不兼容 className不兼容 indexOf不兼容**

**去除空格：^\s\*|\s\*$**

**------面向对象—2016/11/30--------**

**不能再系统对象中随意附加方法、属性，否则会覆盖已有的方法、属性**

**一般属性挂在类上，而方法挂在类的原型上**

**http://www.ruanyifeng.com/blog/2010/05/object-oriented\_javascript\_encapsulation.html**

**面向对象：**

**换了一种（写法）思想去写代码，可能在某个时间点上结果是一样，但是经过多次迭代结果一定不一样，用面向对象的写法会越写越简单，而用面向过程的方式写会越写越复杂。**

**把具有相同特征的代码抽离出来封装为一类，在这一类上扩充方法或者属性，让这个类（功能）更强大。**

**属性一定是在某个(非空) 对象上的。 null（空对象） 只有(非空)对象上才能加属性 [] {}**

**JS面向对象编程（OOP）三大特点：**

**抽象：抓住核心问题**

**封装：不考虑内部实现，只考虑功能使用**

**继承：从已有对象上，继承出新的对象**

1. **多重继承**
2. **多态**

**对象的组成：**

**方法：函数（过程 动态的）对象下面的函数：叫做对象的方法**

**属性：变量（状态、静态的）对象下面的变量 我们叫做对象的属性**

**属性或者方法前面都是有主的，都是挂在某个对象上的。**

**属性一定是在某个（非空）(对象)上的. 空是null**

**JS中可以理解为 类就是构造函数**

**New 后面调用的函数：叫做构造函数**

**构造函数 ：**

**这个函数专门拿来构造一个对象的**

**工厂函数：**

**有原材料，有加工，有出厂的函数**

**工厂方式 构建对象 构造函数**

**缺点： 1 没有new**

**2 每个对象都有一套自己的函数 （浪费空间）**

**new : 运算符 专门运算函数的。**

**用了new后函数可以不用加括号就调用,new运算之后的()主要是为传参用的**

**new之后偷偷的在函数内容创建了一个空白的对象**

**new之后函数的默认返回值就为new出来的这个对象。**

**return之后如果后面跟着为对象类型也包括函数，那么返回值就为return后面的值,否则就返回new出来的对象。**

**实例：var f2 = new fn1(1);//类的实例化过程。 f2就是实例化对象**

**原型：prototype :要写在构造函数下面 (与class方法类似)**

**当一个函数被申明的时候，该函数下默认有一个属性：prototype，（该属性的值是一个\*\*对象\*\*）当我们去调用一个对象的属性或者方法的时候，如果该对象自身没有该属性或方法，则会调用到该对象的构造函数下的prototype下面属性或方法.**

**原型：去改写对象下面公用的方法或者属性，让公用的方法或属性在内存中只存在一份（提高性能）**

**Array. Prototype.（方法名字）= function(){}**

**原型的一个重要应用： 给系统的类添加方法**

**prototype是给这个类的实例化对象使用的。（自身用不了）**

**原型链: \_\_proto\_\_**

**对象有原型链 (对象与构造函数原型的桥梁)**

**函数有原型 prototype**

**对象没有原型只有链**

**碰到对象找原型链，碰到函数找原型（始终要遵循,原型只给这个类的实例化对象用）**

**思路重点：**

**如果是对象（new 的），那么会先找链 -> 构造函数的原型 -> 构造函数的原型的链 -> 对象的原型 （不回去找函数跳过了函数）**

**如果是找函数：**

**那么会先找链（因为函数的原型是给这个函数的实例化对象使用的）**

**构造函数的原型(最后都会找到obj)**

**1.这个函数就是Function -> Function.prototype -> Function.prototype.\_\_proto\_\_ -> Object.prototype**

**2.Array.\_\_proto\_\_ -> Function.prototype -> Function.prototype.\_\_proto\_\_ -> Object.prototype**

**Obj最大 接着是 function 函数的也先走函数的在找obj**

**this的定义:当前的方法属于谁**

**当new去调用一个函数：这个时候函数中的this就是创建出来的对象，而且函数的返回值直接就是this了（隐式返回）**

**面向过程**

**关注实现过程和每一步的实现细节**

**面向对象**

**关注特性和功能**

**面向对象的好处**

**1、代码的层次结构更清晰**

**2、更容易复用**

**3、更容易维护**

**4、更容易扩展**

**5、more...**

**什么是包装对象(正规版)**

**当使用原始类型的值（当简单类型string、number、boolean），在调用对应属性和方法的时候，内部会自动转成对应的对象。隐式创建的这个对象，就成为包装对象**

**内置的对象**

**String、Number、Boolean**

**就算内置对象，也不能随意去修改属性 ，只能读。**

**包装对象的特点**

**隐式创建对象后，可以调用对应的属性和方法，使用后，立马销毁，所以不能给原始类型的值添加属性和方法**

**hasOwnProperty：查看某个属性是不是这个对象自身的。**

**注意：\*\*\*自身，一定是在当前这个对象中（不会沿着原型链查找属性，只查找自身属性**

**如果，当前（某个属性）属性是挂在这个对象的原型上，不算是自身的属性**

**实例：object.hasOwnProperty(prop); 返回boolean(布尔值)**

**t1.hasOwnProperty('name')**

**深度克隆 用for in去循环 (面试会问)**

**因为简单类型的赋值就是赋值**

**简单类型的赋值是不会影响另一个数据**

**而对象类型的赋值是赋址**

**对象类型赋值都是引用一个内存地址，所以说，改变arr2，同样会改变arr**

**解决：**

**通过循环，将对象中每个简单类型赋值给另一个对象，就不会出现引用关系。**

**写个函数，只要是复合类型，就继续循环，否则直接赋值（递归）**

**实例：**

**//////////////////////////////////////////////////////////**

**var arr = [1,2,3,4,[1,2],function abc(){},{name:'momo'}];**

**var arr2 = extend(arr);//调用这个递归的克隆函数**

**function extend(obj){**

**var o = !!obj.push?[]:{};//用有没有push方法判断是数组还是对象**

**for(var attr in obj){**

**if(typeof obj[attr] == 'object'){**

**o[attr] = extend(obj[attr]);//里面的子项是对象就递归**

**}else{**

**o[attr] = obj[attr];**

**}**

**}**

**return o;**

**}**

**//////////////////////////////////////////////////////////////**

**constructor**

**对象.constructor 属性 为个对象的函数的构造函数，用来查找他的构造函数**

**调用方法：obj. constructor == Objcet 用来查找判断他的构造函数**

**constructor:可以查看某个对象的构造函数是谁**

**obj.constructor == Objcet**

**注意：**

**constructor会无形被修改,虽然不会改变某个方法的执行结果，但是会影响你的客户维护。**

**解决：**

**手动修改constructor指向**

**原型的另一种写法**

**Teacher.prototype = {**

**constructor:Teacher,//改变着个指向这个函数**

**say:function(){**

**alert(this.name);**

**}**

**}**

**instanceof: 二元运算符**

**运算检测一个函数的原型是否在要检测的对象的原型链上**

**检测某个对象（字符串等不行）的老爹是不是某个构造函数**

**用法： obj instanceof Objcet 如果是就true 不是 false**

**检测对象的，不能检测简单类型**

**instanceof 就是实例化对向与构造函数之间的对比**

**一元运算符： ！ new typeof**

**二元运算符（+ —）**

**继承：（面试会问）**

**继承：在JavaScript中，继承就是让一个对象拥有另一个对象的方法或者属性**

**常用的继承： 拷贝继承、类式继承、对象继承 （任选其一即可）**

**改变this指向继承属性 拷贝继承方法 （直接复制等于的话 是负值 后期操作会修改父类的方法）**

**Call()方法：**

**当某个函数声明的时候系统就默认给这个函数加了一些属性或者方法其中一个就有一个call方法、**

**参数：**

**有无数个参数，**

**第一个参数 改变this指向 写谁就是谁**

**第二个参数以后 就是正常的传参的实参了**

**用法**

**函数会被:事件调用、函数名+括号、定时器、匿名函数自执行、new、call、apply**

**apply()：**

**同样改变this指向**

**只有两个参数：**

**第一个参数 改变this指向 写谁就是谁**

**第二个参数 是一个数组 是所传实参的集合，有几个实参就在数组中写几个对应的实参**

**属性继承**

**调用父类并且将父类的this修改成子类，这个时候子类就拥有了父类的属性**

**属性继承是通过改变this指向做到的**

**实例：function Drag2(id){**

**Drag.call(this,id);**

**}**

**方法继承：**

**1 拷贝继承：（把父级方法拷贝到了自己身上，自己实实在在的有这些方法）**

**通过 for in遍历父类的原型porototype上的方法，赋值给子类的原型，从而达到方法继承的目的**

**注意： 它把祖宗的方法也复制过来了 所以 复制的时候加个自身判断（hasOwnProperty）来截取一下**

**实例：**

**for(var s in Move.prototype){**

**//如果这些属性是Drag.prototype对象自身的，那么才赋值。**

**if(Move.prototype.hasOwnProperty(s)){**

**Move2.prototype[s] = Move.prototype[s];**

**}**

**}**

**2 （类式）继承：（没有真正的复制到自己身上只是通过一个对象中转去借用）所以注意修改方法的指向**

**（类式）继承比拷贝继承的性能要高，因为少了for in循环 for in 非常耗性能**

**1.创建一个新类**

**2.将这个新类的原型等于父类的原型**

**3.将这个新类实例化对象赋值给子类的原型**

**实例1：**

**paohui.prototype = Move.prototype;**

**function paohui(){}**

**var ph = new paohui;// new paohui 是个对象 开启一个新内存地址**

**Move2.prototype = ph;**

**\*Move2.prototype.constructor= Move2;//把指向指向自己**

**实例2：**

**Move2.prototype = new Move;**

**``````\*Move2.prototype.constructor= Move2;//把指向指向自己**

**3对象继承（原型继承）：**

**1.创建一个新类**

**2.将这个新类的原型等于要继承的父对象**

**3.将这个新类实例化对象赋值给子对象。**

**var obj = {**

**say:function(){**

**console.log(1)**

**},**

**running:function(){**

**console.log(2)**

**}**

**}**

**这中方式必须有炮灰**

**paoh.prototype = obj;**

**function paoh(){}**

**var ph = new paoh;**

**obj2 = ph;**

**obj2.say = function(){//这是后在修改就不会再改变了**

**alert(1);**

**}**

**toString:方法 将某种数据类型转成字符串**

**任何的内置对象都会有toString将某种数据类型转成字符串**

**String -> 字符串**

**Array -> 数组**

**Object -> 对象**

**Number -> 数字**

**只要使用alert,那么系统就会自动调用该数据类型的toString**

**使用方法：Obj.toString();**

**查看数据类型的方法： 面试可能会问 把对象的方法写下来**

**[object(始终是对象类型) Object(构造函数是谁)]**

**var tS = Object.prototype.toString;//记录对象的toString方法**

**tS.call(arr2)//使用时改变this指向 用对象的方法查看arr2的数据类型**

**想查看谁的准确类型 就直接在call 后面把指向变成谁**

**对象的原型下的toString方法啊**

**delete arr.a 删除某个对象上的属相**

**类数组转化成数组var lis = Array.prototype.slice.call(divs);传入类数组**

**关于this：**

**先分清this属性的值是什么类型;**

**1 单纯的 函数名+括号， this -> window**

**2 将一个函数交给事件调用 this —>是触发事件的这个对象（在PC上）**

**3 定时器中this -> window{**

**只要定时器是直接调用某个函数或者方法，那么this就是window**

**否则this该是谁就是谁（定时器里面的）**

**}**

**4 有主函数一般情况下 this就是主,如果被定时器调用，那么this一样是window但是在定时器里面的话this该是谁就是谁**

**5 new ->就是直接调用this为对象**

**6 call、apply 写的this是谁就是谁**

**7 匿名函数 this window**

**小技巧：**

**匿名函数自执行的时候，前面加 ; 目的是为了避免报错。**

**移动端事件：**

**touchstart 相当于mousedown**

**touchmove 相当于 mousemove**

**touchend 相当于 mouseup**

**移动端的拖拽事件就不用一层层嵌套了 直接评级写就行 而且不用document转化移动事件了 因为移动端有一个机制 会检测到是不是在本元素上触发的起始事件 是的话她自己会调用下面相应的 移动 和up 事件**

**----------------组件开发---2016-12-5--------**

**组件：提供一个完善的功能的方法或数据的封装**

**组件要求： 易扩展，易维护**

**json 多个配置的方法：**

**设置一个内置的参数对象**

**this.settings = {**

**d:function(){},**

**m:function(){},**

**u:function(){}**

**}**

**在init调用的时候在循环吧配置了的参数修改一下并把参数通过init传入**

**for(var attr in json){**

**this.settings[attr] = json[attr];**

**}**

**、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、**

**addEventListener运行机制：同一个事件绑定一次或者多次的时候，将同样事件的事件函数，提取出来**

**当这个事件被触发的时候，依次执行这个事件下的所有事件函数**

**映射： 自定义事件**

**核心：找到某个事件函数,只要调用一次addEvent，就往一个对象的数组中push这个函数 ，当触发某个事件的时候，循环这个数组，依次调用**

**实例：在这里obj可以看成box（其实就是触发事件的元素）**

**绑定事件：目的{记录对应事件名的事件函数}**

**function addEvent(obj,events,fn){//参数1元素2事件自定义名字3执行的函数**

**obj.Event = obj.Event || {};**

**obj.Event[events] = obj.Event[events] || [];**

**obj.Event[events].push(fn);**

**}**

**//若这个对象有这个属性，就从这个属性下去建数组，若没有，就等于空对象**

**//如果这个对象的这个属性下有这个事件名，说明之前绑定过，从之前的数组中push新的函数就可以了，否则创建一个空白数组**

**//把当前绑定的这个函数push到这个对象 Event 属性的事件名数组中**

**触发：**

**//一次性执行某个对象的对应事件名的所有函数，在调用的时候就调用它**

**function tirgger(obj,events){ //参数1元素2事件自定义名字**

**if(!obj.Event[events])return null;**

**obj.Event[events].forEach(function(e){**

**e(obj);//obj绑定的对象可传可不传，传入防止里面用到**

**});**

**}**

**//若这个对象没这个事件名，直接返回空**

**//有的情况下就循环这个事件下的所有函数，并且调用，再把绑定的对象传入进去**

**下面是维护的时候直接照着绑定事件的那个函数写就可以**

**addEvent(document,'点击',function(){//参数1元素2事件自定义名字**

**alert(1);**

**});**

**组件 预习JQ做点练习**

**功能型 百度云 webqq 计价**

**功能 炫酷 手机端**

**Babel 转兼容用的网站**

**张鑫旭 css88 H5梦工厂 设计 韩雪冬**

**W3cfuns 转发博客 css3 html5 网站主页 作品 个人介绍**

前端工具gulp、git、angularJs、Ajax、less、sars

react vue angular

**没写完 商品下架 10.20日多级下拉菜单布局练习 韩雪东 手写快排 文件夹创建 窗口模拟 10.31广告吸附**

[**www.xunlei.com**](http://www.xunlei.com)

**DOMContentLoaded 当DOM加载完成就执行里面的代码不兼容低版本IE**

**（高版本用） 示例：只能写绑定事件**

**document.addEventListener('DOMContentLoaded',function(){alert(1);});**

**性能很低，在try中如果有报错，那么会走catch，保证之后的代码正常运行。**

**try{a}catch(e){a=10}**

**ondblclick 双击事件**

**创建一个img元素，给它src加上一个图片地址，**

**通过onload事件可以监听这个图片是否加载完成。**

**替换字符串 replace(/要替换的/,替换成的)**

obj.selectedIndex:

select选中的索引，可读可写 默认从0开始计算

**+new Data(); 时间戳 的简便方法**

**HTML5新增 worker 模拟多线程**

**Apply()改变当前函数的this指向,两个参数 第一个指向谁 第二个是参数是数组**

**函数平缓运动写法：**

**function update(){**

**if(box.offsetLeft < 200){**

**box.style.left = box.offsetLeft + 3 + 'px';**

**}**

**window.requestAnimationFrame(update);**

**}**

**数组是按引用（指针）传递**

**Callee 属性代表函数本身**

**Fn1.callee代表调用这个函数的函数**

**arguments.callee 匿名函数的定时器里调用**

**console.dir()打印详细信息**

**不定内容均分宽度**

**父级display: table**

**子集display: table-cell;虎爷操作**

**或另一种**

**父级display: flex;**

**子集 flex: 1;**

**闭包调用**

**计算后样式返回值 闭包**

**function css(el,attr){ getComputedStyle的兼容性**

**if(el.currentStyle) {//这是兼容IE的**

**return el.currentStyle[attr];**

**}**

**return getComputedStyle(el)[attr];//这是兼容正常的**

**}**

**//alert(css(box,"width"));//这是调用是输入对应的传参值**

**// var bannerH = parseFloat(css(banner,'height')); 这是调用转化成数字**

**兼容性**

**Input 的placeholder 输入框默认内容 ie8一下不兼容**

**兼容 用一个san标签去模拟 隐藏显示**

**oninput 当输入框的内容改变的时候出发 事件 不兼容**

**显示 获取内容长度 是可以开定时器 去监听valuede 长度**

关于表单的input 的type值不要 进行去点击修改type的类型 否则IE6/7/8会出现兼容性问题

可以点击的时候吧一个按钮隐藏 再把另外一个按钮显示出来 模拟效果

**浮动的问题**

Div.style.float = ‘left’; 浏览器不认

div.style.styleFloat = ‘left’;

div.style.cssFloat = ‘left 把这两个都写上 处理兼容

IE（styleFloat）非IE（cssFloat）

更好的方法是 写在css的 class名里 直接去修改他的class名字 效果更好

. 后面的值无法修改 所以用[]

**默认样式初始化**

**css reset 原则：**

**但凡是浏览默认的样式，都不要使用。**

**body,dl,dd,p,h1,h2,h3,h4,h5,h6{margin:0;font-size:12px;}**

**ol,ul {margin:0;**

**padding:0;**

**list-style:none;}**

**a {text-decoration:none;}**

**img { border:none**

**vertical-align: top（上对齐）;}**

**.clearFix:after { content:"";**

**display:block;**

**clear:both;}**

**.clearFix{zoom:1;}**

**table { border-collapse: collapse; }（合并表格边框）**

**td,th { padding: 0;**

**form { margin: 0; } IE6下 form上有一个margin值**

**input { outline: none;**

**margin: 0;**

**padding: 0**

**box-sizing: content-box; 标准盒模型**

**或box-sizing:border-box 怪异盒模型 }**

**/\* border-width 在各浏览器下不一致，需要根据设计图单独设置一下\*/**

**textarea { margin: 0;**

**padding: 0;**

**resize: none;**

**overflow: auto;**

**outline: none; 轮廓线无}**

**strong { font-weight: normal;**

**em { font-style: normal;**

**mark { background: none;**