# 1.形状变换

<button>与<input>的区别

JS中变换css属性，div.style[width]=

用this.index代替遍历中的i，打破闭包的限制

不要忘记先使用window.onload

# 2.网页换肤

body{font: 12px/1.5 Tahoma;} //像素12px行高1.5

Zoom的使用方法：   
zoom :﻿ normal | number   
normal :　 默认值。使用对象的实际尺寸   
number :　 百分数 | 无符号浮点实数。浮点实数值为1.0或百分数为100%时相当于此属性的 normal 值用白话讲解就是zoom:后面的数字即放大的倍数，可以是数值，也可以是百分比。如：zoom:1，zoom:120%。而这个属性只要在IE中才起作用，所以很少用到它的实际用途，而最经常用到作用是清除浮动等，如：

\_zoom是CSS hack中专对IE6起作用的部分。IE6浏览器会执行zoom:1表示对象的缩放比例，但这里   
overflow:hidden;和\_zoom:1;是连起来用的，作用是清除border内部浮动。

text-indent 属性规定文本块中首行文本的缩进。 注释:允许使用负值。如果使用负值,那么首行会被缩进到左边

head里的link标签也是可以提取的

# 4.div循环变色

center标签的使用

for(var i in oDiv) oDiv[i].style.background="red" 可以实现效果，但是会提示错误

# 5.onmouseover,onmouseout

# 6

vertical-align:center 垂直居中

# 第二课

# 1.

this.value.replace(/[^(\d)|(,)]/,"")

var oInput = document.getElementsByTagName("input")[0].value.split(",");

for(var i in oInput){

sum += parseInt(oInput[i]);

}

4.弹出层

很多朋友问过我absolute与relative怎么区分，怎么用？我们都知道absolute是绝对定位，relative是相对定位，但是这个绝对与相对是什么意思呢？绝对是什么地方的绝对，相对又是相对于什么地方而言的呢？那他们又有什么样的特性，可以做出什么样的效果呢？关于两者之间又有什么样的技巧呢？下面我们就来一一解读。

Absolute，CSS中的写法是：position:absolute; 他的意思是绝对定位，他是参照浏览器的左上角，配合TOP、RIGHT、BOTTOM、LEFT(下面简称TRBL)进行定位，在没有设定TRBL，默认依据父级的做标原始点为原始点。如果设定TRBL并且父级没有设定position属性，那么当前的absolute则以浏览器左上角为原始点进行定位，位置将由TRBL决定。

一般来讲，网页居中的话用Absolute就容易出错，因为网页一直是随着分辨率的大小自动适应的，而Absolute则会以浏览器的左上角为原始点，不会应为分辨率的变化而变化位置。很多人出错就在于这点上出错。而网页居左其特性与Relative很相似，但是还是有本质的区别的。

Relative，CSS中的写法是：position:relative;  他的意思是绝对相对定位，他是参照父级的原始点为原始点，无父级则以BODY的原始点为原始点，配合TRBL进行定位，当父级内有padding等CSS属性时，当前级的原始点则参照父级内容区的原始点进行定位。

**父元素设置为 relative ,但是不设置TRBL.。子元素设置为absolute，那子元素就是相对于父元素的TRBL**

## [offsetLeft与style.left的区别](http://www.cnblogs.com/woshilee/articles/1951457.html)

offsetLeft 获取的是相对于父对象的左边距

left 获取或设置相对于 具有定位属性(position定义为relative)的父对象 的左边距

如果父div的position定义为relative,子div的position定义为absolute，那么子div的style.left的值是相对于父div的值，  
这同offsetLeft是相同的，区别在于：  
1. style.left 返回的是字符串，如28px，offsetLeft返回的是数值28，如果需要对取得的值进行计算，  
还用offsetLeft比较方便。  
2. style.left是读写的，offsetLeft是只读的，所以要改变div的位置，只能修改style.left。  
3. style.left的值需要事先定义，否则取到的值为空。而且必须要定义在html里，我做过试验，如果定义在  
css里，style.left的值仍然 为空，这就是我刚开始碰到的问题，总是取不到style.left的值。

offsetLeft则仍然能够取到，无需事先定义div的位置。

offsetParent：该属性返回一个对象的引用，这个对象是距离调用offsetParent的元素最近的（在包含层次中最靠近的），已进行过CSS定位的容器元素。 如果这个容器元素未进行CSS定位, 则offsetParent属性的取值为body元素的引用。 当容器元素的style.display 被设置为 "none"时（译注：IE和Opera除外），offsetParent属性 返回 null。

 top：该属性一般对用过css定位的元素有效（position为“static”时为auto，不产生效果），定义了一个top属性有效的元素（即定位元素）的上外边距边界与其包含块上边界之间的偏移。

clientTop：元素上边框的厚度，当没有指定边框厚底时，一般为0。

scrollTop：位于对象最顶端和窗口中可见内容的最顶端之间的距离，简单地说就是滚动后被隐藏的高度。

offsetTop：获取对象相对于由offsetParent属性指定的父坐标(css定位的元素或body元素)距离顶端的高度。

clientHeight：内容可视区域的高度，也就是说页面浏览器中可以看到内容的这个区域的高度，一般是最后一个工具条以下到状态栏以上的这个区域，与页面内容无关。

scrollHeight：IE、Opera 认为 scrollHeight 是网页内容实际高度，可以小于 clientHeight。FF 认为 scrollHeight 是网页内容高度，不过最小值是 clientHeight。

offsetHeight：获取对象相对于由offsetParent属性指定的父坐标(css定位的元素或body元素)的高度。IE、Opera 认为 offsetHeight = clientHeight + 滚动条 + 边框。FF 认为 offsetHeight 是网页内容实际高度，可以小于 clientHeight。offsetHeight在新版本的FF和IE中是一样的，表示网页的高度，与滚动条无关，chrome中不包括滚动条。

 \*诸如left、clientLeft、offsetLeft、clientWidth、scrollWidth等，和top、height类似，不再赘述。

============================分割线=====================================

clientX、clientY：相对于浏览器窗口可视区域的X，Y坐标（窗口坐标），可视区域不包括工具栏和滚动条。IE事件和标准事件都定义了这2个属性。

pageX、pageY：类似于event.clientX、event.clientY，但它们使用的是文档坐标而非窗口坐标。这2个属性不是标准属性，但得到了广泛支持。IE事件中没有这2个属性。

offsetX、offsetY：相对于事件源元素（target或srcElement）的X,Y坐标，只有IE事件有这2个属性，标准事件没有对应的属性。

screenX、screenY：相对于用户显示器屏幕左上角的X,Y坐标。标准事件和IE事件都定义了这2个属性

**1、offsetWidth width+padding+border)**

当前对象的宽度。

style.width也是当前对象的宽度(width+padding+border)。

区别：1)style.width返回值除了数字外还带有单位px；

           2)如对象的宽度设定值为百分比宽度,则无论页面变大还是变小，

              style.width都返回此百分比,而offsetWidth则返回在不同页面中对象的宽度值而不是百分比值；

           3)如果没有给 HTML 元素指定过 width样式，则 style.width 返回的是空字符串；

**2、offsetHeight :(Height+padding+border)**

当前对象的高度。

style.height也是当前对象的高度(height+padding+border)。

区别：1)style.height返回值除了数字外还带有单位px；

           2)如对象的高度设定值为百分比高度,则无论页面变高还是变矮，

              style.height都返回此百分比,而offsetHeight则返回在不同页面中对象的高度值而不是百分比值；

           3)如果没有给 HTML 元素指定过 height样式，则 style.height返回的是空字符串；

**3、offsetLeft :**

当前对象到其上级层左边的距离。

不能对其进行赋值.设置对象到其上级层左边的距离请用style.left属性。

style.left当前对象到其上级层左边的距离。

区别：1)style.left返回值除了数字外还带有单位px；

           2)如对象到其上级层左边的距离设定值为百分比，

              style.left返回此百分比,而offsetLeft则返回到其上级层左边的距离的值；

           3)如果没有给 HTML 元素指定过 left样式，则 style.left返回的是空字符串；

**4、offsetTop :**

当前对象到其上级层顶部边的距离。

不能对其进行赋值.设置对象到上级层顶部边的距离请用style.top属性。

**offsetTop就是当前对象border的外顶点到父元素border的内顶点。不带px。**

**style.top就是当前对象的margin的外顶点到父元素border的内顶点。 带px。**

**没有Margin的时候两者相等。**

style.top当前对象到其上级层顶部边的距离。

区别：1)style.top返回值除了数字外还带有单位px；

           2)如对象到其上级层顶部边的距离设定值为百分比，

              style.top返回此百分比,而offsetTop则返回到其上级顶部边的距离的值；

           3)如果没有给 HTML 元素指定过 top样式，则 style.top返回的是空字符串；

注意：如果上级层为body，由于IE、FF对padding、margin的解释不一样所以要明确规定处理不是下列的区别就不成立了。

IE   1)如果Div的上级层是body，而div与body之间有个div，如body->div->divo；divo的offsetTop=div的padding+margin+boder；

       2）如果Div的上级层是body，如body>divo；divo的offsetTop=div的padding+margin+boder；

             这divo的offsetTop=divo的margin >body.padding则为divo的margin，否则为body.padding谁大是谁？

FF  上述两种情况：offsetTop=margin+padding ;

(IE与FF中的body默认padding为10)在IE6.0 FF3.6.13

**5、scrollWidth:**获取对象的滚动宽度 。

**6、scrollHeight:** 获取对象的滚动高度。

**7、scrollLeft:**设置或获取位于对象左边界和对象中目前可见内容的最左端之间的距离(width+padding为一体)

**8、scrollTop:**设置或获取位于对象最顶端和对象中可见内容的最顶端之间的距离；(height+padding为一体)

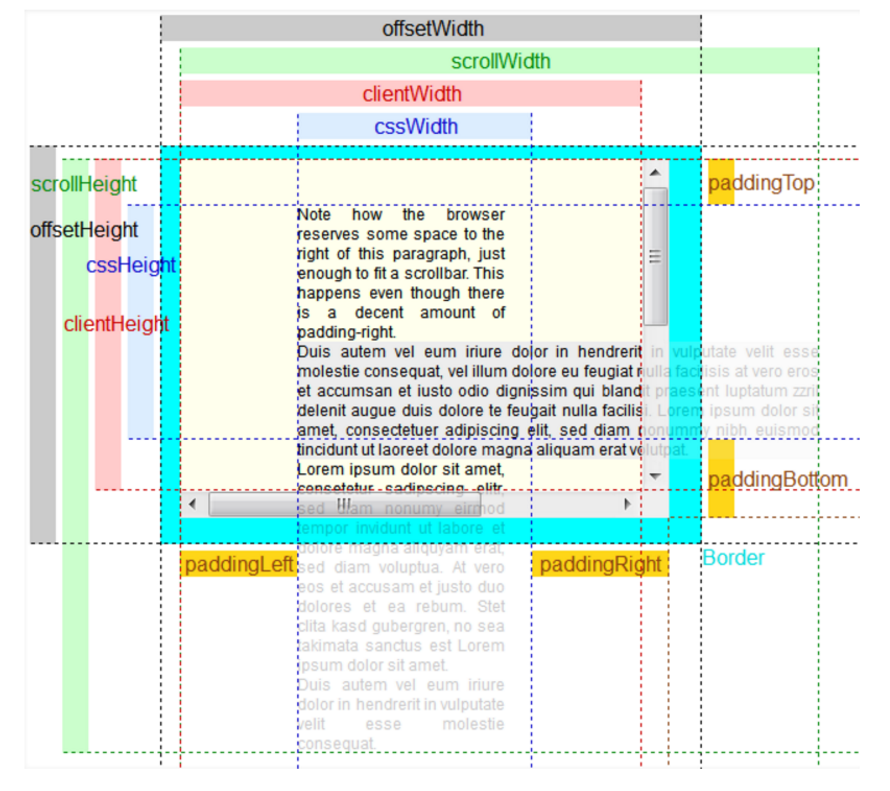
**9、clientWidth:** 获取对象可见内容的宽度，不包括滚动条，不包括边框；

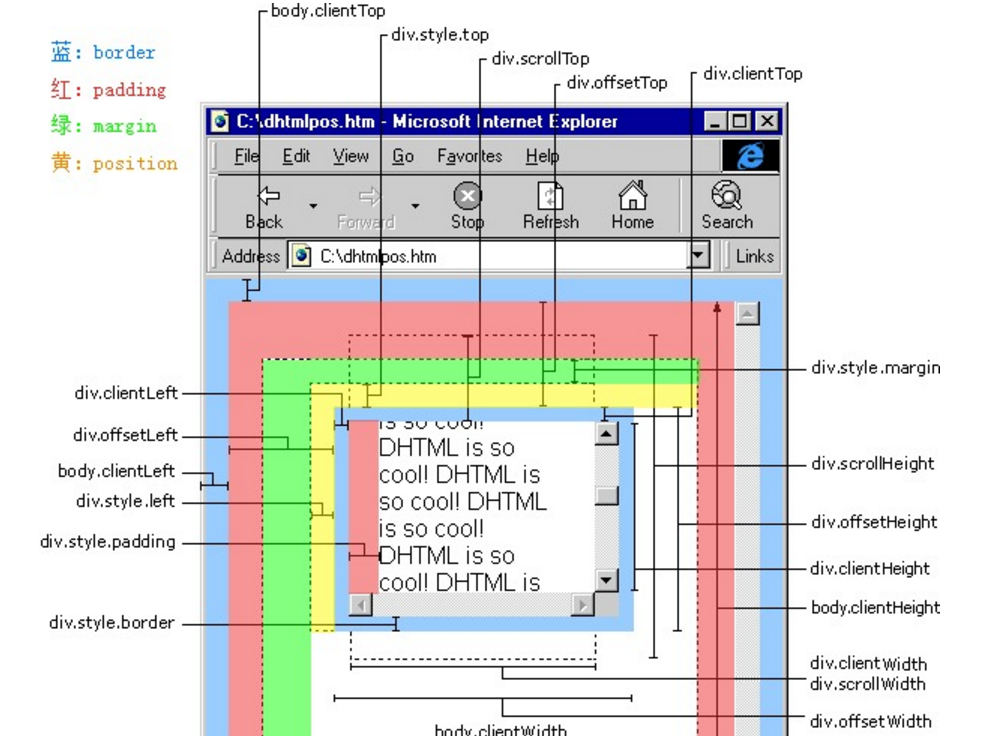
**10、clientHeight:** 获取对象可见内容的高度，不包括滚动条，不包括边框；

**11、clientLeft:** 获取对象的border宽度

**12、clientTop**：获取对象的border高度

**13、offsetParent :**当前对象的上级层对象.





# 5.函数传参

oDiv.removeAttribute("style");不能用style=””。

# 6.移入透明

padding顺序：上右下左顺时针。

opacity: 1;filter: alpha(opacity=100)为什么要一起写？ie低版本不支持opacity的  只支持filter  。为了兼容

# 7.简易选择卡

this.className

for(var n in oTab) oTab[n].className = "";

this.className = "current" //可以用for in

for(var x=0; x<oUl.length;x++) oUl[x].style.display = "none"; //不可以用 for in

# 10.提示框效果

CSS 伪类(Pseudo-classes)

CSS伪类是用来添加一些选择器的特殊效果。



使用[content](http://www.runoob.com/cssref/pr-gen-content.html) 属性来指定要插入的内容。

# 11.图片显示

在父元素上使用 overflow:hidden; zoom:1; 可以清除子元素浮动造成的体积影响。

可以插入伪元素 :after (content = “ ..”) 来实现同样的效果

要达实现加载过程中的过度动画效果。可以创建一个新的img对象。利用 img.complete? 来判断是否加载成功。

var img = new Image();

oDiv.style.display = "block";

img.src = oImg[0].src = this.src.replace(/small/,"big");

img.complete?oDiv.style.display = "none":(oImg[0].onload=function(){oDiv.style.display = "none"});

# 12.复选框

**不要忘记！！！！**

//根据每个复选框状态更新全选框状态。。。。

for(var i=1;i<oInput.length;i++){

oInput[i].onclick = function(){

isCheckAll();

}

}

# 第三课

1.typeof

2.parseInt

提取text文本框内容要用value属性。

首先，对限定文本框内容进行检查。利用正则表达式和value.replace()

然后，检查文本框是否为空。如果有空要提醒用户填写。

# 5.页面自计数

var updateNum = function(){

oBody.innerHTML = ++i;

}

updateNum(); //先调用一次，保证从1开始，前面没有空白时间

setInterval(updateNum,1000);

var 定义的函数，要在使用函数之前定义。

function updateNum (){

oBody.innerHTML = ++i;

}

function定义的函数可以放在函数使用后面

# 6.检查两位数

var x = /^\d{2}$/.test(parseInt(oInput.value));

x?alert("√ 是两位数"):alert("这是" + oInput.value.length + "位数");

正则表达式。

7.简易计算器

eval() 函数可计算某个字符串，并执行其中的的 JavaScript 代码。

语法

eval(string)

|  |  |
| --- | --- |
| **参数** | **描述** |
| string | 必需。要计算的字符串，其中含有要计算的 JavaScript 表达式或要执行的语句。 |

该方法只接受原始字符串作为参数，如果 string 参数不是原始字符串，那么该方法将不作任何改变地返回。因此请不要为 eval() 函数传递 String 对象来作为参数。

如果试图覆盖 eval 属性或把 eval() 方法赋予另一个属性，并通过该属性调用它，则 ECMAScript 实现允许抛出一个 EvalError 异常。

## 逻辑运算符 ||， && 的短路规则：

**|| 从左向右开始计算，当遇到为真的条件时停止计算，整个表达式为真；所有条件为假时表达式才为假。**

**&& 从左向右开始计算，当遇到为假的条件时停止计算，整个表达式为假；所有条件为真时表达式才为真。**

最后，我们做以下总结：  
对于“与”（**&&**） 运算： **x && y** 当**x**为**false**时，直接跳过，不执行**y**；相当于 if(x==true){y;}  
对于“或”（**||**） 运算 ：   **x||y** 当**x**为**true**时，直接跳过，不执行**y**。

if(x==false){y;}

onfocuse="this.blur()"   
onfocuse是聚焦的意思，当你把光标放在文本框上输入的时候，就是聚焦，但这里添加了"this.blur()"，blur的作用就是去除聚焦，也就是你不能把光标放在这个文本框上，换句话说就是你不能输入文本了 .

在网上看到有的人用onFocus="this.blur()"来消除虚线框，

## 定义和用法

NaN 属性是代表非数字值的特殊值。该属性用于指示某个值不是数字。可以把 Number 对象设置为该值，来指示其不是数字值。

**提示：**请使用 isNaN() 全局函数来判断一个值是否是 NaN 值。

定义和用法

slice() 方法可从已有的数组中返回选定的元素。

语法

arrayObject.slice(start,end)

参数 描述

start 必需。规定从何处开始选取。如果是负数，那么它规定从数组尾部开始算起的位置。也就是说，-1 指最后一个元素，-2 指倒数第二个元素，以此类推。

end 可选。规定从何处结束选取。该参数是数组片断结束处的数组下标。如果没有指定该参数，那么切分的数组包含从 start 到数组结束的所有元素。如果这个参数是负数，那么它规定的是从数组尾部开始算起的元素。

本身有逻辑漏洞

# 8.简易时钟

function format(a)   
{   
return a.toString().replace(/^(\d)$/, "0$1")   
}   
这个函数实现啥功能

在只有数字的一行前面加0.

# 9.倒计时器

this.className = this.className=="" ? "cancel" : "";

this.className=="cancel" ? (timer = setInterval(updateTime,1000),updateTime()) : (clearInterval(timer));

# 第四课

# 4.arguments

## 类数组对象：arguments

在js中在调用一个函数的时候，我们经常会给这个函数传递一些参数，js把传入到这个函数的全部参数存储在一个叫做arguments的东西里面。

在js中万物皆对象，甚至数组字符串函数都是对象。所以这个叫做arguments的东西也是个对象，而且是一个特殊的对象，它的属性名是按照传入参数的序列来的，第1个参数的属性名是’0’，第2个参数的属性名是’1’，以此类推，并且它还有length属性，存储的是当前传入函数参数的个数，很多时候我们把这种对象叫做类数组对象。类数组对象和数组都是对象这个妈生的，但是数组是大哥比类数组对象多了很多其他的玩具(方法)，类数组对象只是长得很像数组的弟弟而已。

慢着，刚刚不是说数组也是对象吗，现在这个类数组对象又是什么？ 没办法，js就是这么的灵活。这个类数组对象不仅存储给函数传入的参数，也具有一些其他的属性，等下会一一道来。

因为类数组对象和数组有很多的共性，所以我们经常可以用call方法，让类数组对象也使用的数组的一些方法，就是让这个弟弟去玩哥哥的玩具，比如……，还是不扯远了，这篇文章只是说什么是arguments，想知道更多关于对象如何借调数组方法的话，请参考这篇文章。

有了这个对象我们以后写函数的时候，就不用给所有的形参指定参数名，然后通过参数名的方式获取参数了，我们可以直接使用arguments对象来获取实参，这样是不是方便了很多呢。   
有些语言在我们给函数指定了参数名之后，当调用函数时，会判断当前传入的参数是否与函数定义的参数个数相等，不相等就会报错，但是灵活的js（不是我说，js是真的灵活）并不会验证传递给函数的参数个数是否等于函数定义的参数个数。所以为了装逼（代码的简洁度），我们使用arguments调用参数可以不混淆不同函数之间的参数名。另外为了装逼（代码的严整度），我们也能用arguments来判断当前传入参数的个数是否与我们需要的数量一致。

最后我们还可以看到arguments还有一个叫做callee的属性，这个属性是表示的是当前函数的一个引用，简单点说，这个属性里面存储的我们调用的这个函数的代码，实在无法理解的时候，又到了console.log大显身手的时候了。

function showcallee() {

var a = '这里是代码';

var b = '这是另一段代码';

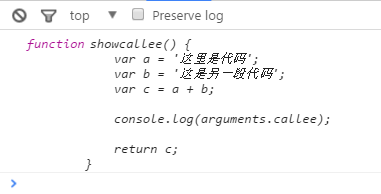
var c = a + b;

console.log(arguments.callee);

return c;

}

showcallee();



看到结果的你是不是和我一样惊呆了呢，这不就是我写的代码吗，arguments.callee完完整整的把这个函数的这段代码返回了。

## arguments的一些妙用

**1.利用arguments实现方法的重载**

下面我们利用arguments对象来实现一个参数相加的函数，不论传入多少参数都行，将传入的参数相加后返回。

function add() {

var len = arguments.length,

sum = 0;

for(;len--;len<0){

sum += arguments[len];

}

return sum;

}

console.log( add(1,2,3) ); //6

console.log( add(1,3) ); //4

console.log( add(1,2,3,5,6,2,7) ); //26

由于js是一种弱类型的语言，没有重载机制，当我们重写函数时，会将原来的函数直接覆盖，这里我们能利用arguments，来判断传入的实参类型与数量进行不同的操作，然后返回不同的数值。

**1.利用arguments.callee实现递归**

先来看看之前我们是怎么实现递归的，这是一个结算阶乘的函数

function factorial(num) {

if(num<=1) {

return 1;

}else {

return num \* factorial(num-1);

}

}

但是当这个函数变成了一个匿名函数时，我们就可以利用callee来递归这个函数。

function factorial(num) {

if(num<=1) {

return 1;

}else {

return num \* arguments.callee(num-1);

}

}

这个方法虽然好用，但是有一点值得注意，ECMAScript4中为了限制js的灵活度，让js变得严格，新增了严格模式，在严格模式中我们被禁止不使用var来直接声明一个全局变量，当然这不是重点，重点是arguments.callee这个属性也被禁止了。不过这都不是事儿，ES6为我们新增了很多好用的变量声明方式和新的语法糖，作为一个时髦的前端，我们赶紧学习一些ES6的新语法吧。

### 剩余参数, 默认参数 和 解构赋值参数

arguments 对象可以与剩余参数、默认参数和结构赋值参数结合使用。

function foo(...args) {

return args;

}

foo(1, 2, 3); // [1,2,3]

在严格模式下,剩余参数剩余参数、默认参数和结构赋值参数的存在不会改变 arguments 对象的行为,但是在非严格模式下就有所不同了。

当在非严格模式函数中 没有包含 剩余参数、默认参数或结构赋值参数,那么 arguments 对象将会追踪参数的变化,反之亦然。看下面的代码:

function func(a) {

arguments[0] = 99; // 更新了arguments[0] 同样更新了a

console.log(a);

}

func(10); // 99

并且

function func(a) {

a = 99; // 更新了a 同样更新了arguments[0]

console.log(arguments[0]);

}

func(10); // 99

当在非严格模式函数中 包含有 剩余参数、默认参数或结构赋值参数,那么 arguments 对象将不会追踪参数的变化,反之亦然。相反, arguments 反映了调用时提供的参数 :

function func(a = 55) {

arguments[0] = 99; // 提供了默认参数,更新 arguments[0] 不会更新 a

console.log(a);

}

func(10); // 10

并且

function func(a = 55) {

a = 99; // 提供了默认参数,更新 a 不会更新 arguments[0]

console.log(arguments[0]);

}

func(10); // 10

并且

function func(a = 55) {

console.log(arguments[0]);

}

func(); // undefined

//上面函数调用的时候没有传递参数,所有 arguments[0] 值为 undefined

# 7.数组

split() 字符串拆分成数组

join() 数组转变成字符串

shift() 删除第一项

pop() 删除最后一项

push() 在数组最后插入

reverse() 颠倒顺序

arrayObject.slice(start,end) 选择特定顺序的值，包含从 start 到 end （不包括该元素）  
arrayObject.sort(sortby) 排序

\s 任何空白字符

aArr.splice(0,3);删除从0开始，后面的3项，但是返回的是被删除的数组，会改变实际对象

aArr.splice(1,0,"orange","purple");第二个参数为0，不删除，是必须参数，从第2项开始插入



# 第六课

## 理解SetCapture、ReleaseCapture、GetCapture

**正常情况下，鼠标指针位于哪个窗口区域内，鼠标消息就自动发给哪个窗口。如果调用了SetCapture，之后无论鼠标的位置在哪，鼠标消息都发给指定的这个窗口，直到调用ReleaseCapture或者调用SetCapture设置另一个窗口为止。**  
很多时候，窗口或控件在鼠标按下后，需要知道鼠标移动和放开的情况，例如按钮在鼠标按下后会变为“按下”状态，鼠标移出按钮区域时“弹起”，所以按钮控件需要在鼠标按下时SetCapture，鼠标放开后ReleaseCapture。  
GetCapture只是检查一下当前是哪个窗口捕获了鼠标，通常不需要调用。

拖曳过程有一个更新参数的监听函数。要注意阻止冒泡函数的写法。

(event || window.event).cancelBubble = true;

图像的左侧和右侧均不允许出现浮动元素：

img

{

float:left;

clear:both;

}

正则表达式中 ？ 允许匹配一个或者0个前面的

//鼠标移出样式,active前面允许有一个空格或tab或回车

oThis.className = oThis.className.replace(/\s?active/,"")

//鼠标移入样式..一定要注意，这里有一个空格！！！多个className之间要有一个空格键

oThis.className += " active";

规定段落中的文本不进行换行：

p

{

white-space: nowrap

}

<http://www.zhangxinxu.com/wordpress/2012/05/getcomputedstyle-js-getpropertyvalue-currentstyle/>

# 6.2模仿微博

## 实例

规定可以由用户调整 div 元素的大小：

div

{

resize:both;

overflow:auto;

}

# 提取现在的style属性

 IE中使用的是obj.currentStyle方法，而FF是用的是getComputedStyle 方法  。“DOM2级样式”增强了document.defaultView，提供了getComputedStyle()方法。这个方法接受两个参数：要取得计算样式的元素和一个伪元素字符串（例如“:after”）。如果不需要伪元素信息，第二个参数可以是null。getComputerStyle()方法返回一个CSSStyleDeclaration对象，其中包含当前元素的所有计算的样式。以下面的HTML页面为例：

# 设置css格式函数理解

function css(obj,attr,value){

//检验给函数的参数个数

switch(arguments.length){

case 2:

//例如css(oLi, {"opacity" : "0", "height" : "0"});设置了对象obj,第二个对象类型是object,批量设置属性。style[i]=attr[i]。attr[opacity] = 0。

if(typeof argument[1] == "object"){

for(var i in attr) i == "opacity" ?(obj.style["filter"] = "alpha(opacity=" + attr[i] + ")") : obj.style[i] = attr[i];

//例如css(oLi, "height");第二个对象类型是string,读取属性

}else{

//IE用currentStyle方法，没有的话，用getComputedStyle方法

return obj.currentStyle ? obj.currentStyle[attr] : getComputedStyle(obj,null)[attr]

}

break;

//有三个参数，单一设置属性。

case 3:

attr == "opacity" ? (obj.style["filter"] = "alpha(opacity=" + value + ")",obj.style[attr] = value/100) : obj.style[attr] = value;

break;

}

};

# 重置表单

reset() 方法可把表单中的元素重置为它们的默认值。

charAt() 方法可返回指定位置的字符。

请注意，JavaScript 并没有一种有别于字符串类型的字符数据类型，所以返回的字符是长度为 1 的字符串。

### 语法

stringObject.charAt(index)

# 第7课

关于setInterval(function(){…},300);

如果有个函数不带参数，function next(){}.可以写成setInterval(next,300);

如果这个函数带参数，function next(a){}.一定要写成setInterval(function(){next(a)},300);

事情是这样的

function move(){} <=> move = function(){}

也就是说 move是个变量，存储的是函数本身

你把函数本身传给setInterval，他就会每隔100毫秒执行一次 这个函数

而加了括号以后，函数直接就运行了，所以你把move()运行后的值传给了setInterval,函数move又没有返回值，那setInterval每100毫秒执行一次“空”，所以啥都没干

# 第八课

# [js中return；、return true、return false;区别](http://www.cnblogs.com/weiwang/p/3268374.html)

一、返回控制与函数结果，

语法为：return 表达式;

语句结束函数执行，返回调用函数，而且把表达式的值作为函数的结果

 二、返回控制，

无函数结果，语法为：return;

 在大多数情况下,为事件处理函数返回false,可以防止默认的事件行为.例如,默认情况下点击一个<a>元素,页面会跳转到该元素href属性指定的页.  

 Return False 就相当于终止符，Return True 就相当于执行符。  

 在js中return false的作用一般是用来取消默认动作的。比如你单击一个链接除了触发你的  

 onclick时间（如果你指定的话）以外还要触发一个默认事件就是执行页面的跳转。所以如果  

 你想取消对象的默认动作就可以return false。

首先在js中，我们常用return false来**阻止提交表单**或**者继续执行下面的代码，**通俗的来说就是阻止执行默认的行为。

function a（）{

   if（True）  
       return false;  
},这是没有任何问题的。

如果我改成这种

function Test（）{

   a（）;

   b（）;

   c（）;

}

即使a函数返回return false 阻止提交了，但是不影响 b（）以及 c（）函数的执行。在Test（）函数里调用a()函数，那面里面

return false 对于Test（）函数来说，只是相当于返回值。而不能阻止Test（）函数执行。

**总之：return false 只在当前函数有效，不会影响其他外部函数的执行。**

**三：总结**

**retrun true； 返回正确的处理结果。**

**return false；分会错误的处理结果，终止处理。**

**return；把控制权返回给页面。**

# 8.2鼠标滚轮事件

学习本节时，除Firefox(妈的，firefox终于各色了一把)外所有浏览器都支持“mousewheel”事件，但firefox支持DOMMouseScroll 而DOM3规范草案建议使用名“wheel”替代“mousewheel”

传递“mousewheel”处理程序的事件对象有wheelDeta属性，其指定用户滚动滚轮有多远。远离用户方向一次鼠标滚轮“单击”的wheelDelta值通常是120，而接近用户方向的一次“单击”值是-120. Safari 和chrome中，为了支持使用二维轨迹球的apple鼠标，除了wheelDeta属性外，事件对象不宜 wheelDeltaX和wheelDeltaY。

在Firefox中，可以使用非标准的DOMMouseScroll事件取代mousewheel，使用事件对象的detail属性取代wheelDelta。但是detail属性值的缩放比率和正负符号不同于wheelDelta,detail值乘以-40和wheelDelta值相等。

对于所有这些事件类型来说，其事件对象就像鼠标事件对象：它包括鼠标指针和键盘辅助键状态。

# 8.3

## 定义和用法

z-index 属性设置元素的堆叠顺序。拥有更高堆叠顺序的元素总是会处于堆叠顺序较低的元素的前面。

**注释：**元素可拥有负的 z-index 属性值。

**注释：**Z-index 仅能在定位元素上奏效（例如 position:absolute;）！

### 说明

该属性设置一个定位元素沿 z 轴的位置，z 轴定义为垂直延伸到显示区的轴。如果为正数，则离用户更近，为负数则表示离用户更远。

|  |  |
| --- | --- |
| **默认值：** | auto |
| **继承性：** | no |
| **版本：** | CSS2 |
| **JavaScript 语法：** | *object*.style.zIndex="1" |

# 9.2

创建一个新的空白的文档片段( [DocumentFragment](https://developer.mozilla.org/en-US/docs/DOM/DocumentFragment))。

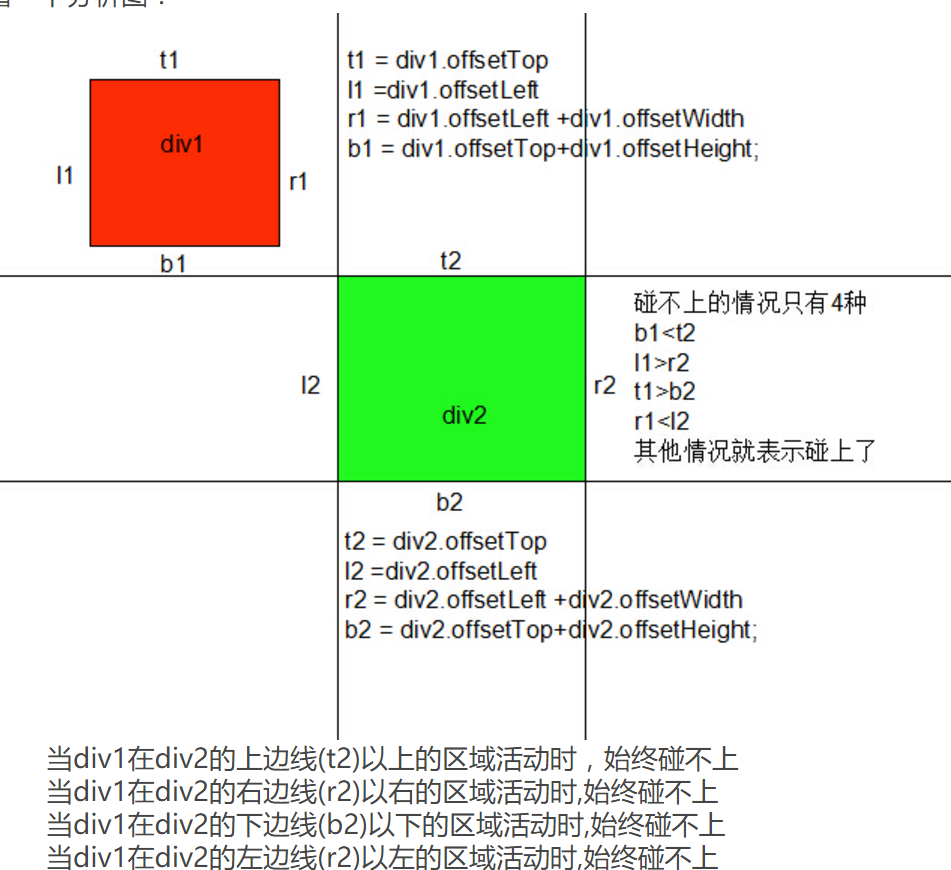
## 语法

let docFragment = document.createDocumentFragment();

docFragment 是一个对空文档片段 [DocumentFragment](https://developer.mozilla.org/en-US/docs/DOM/DocumentFragment) 对象的引用。

## 描述

[DocumentFragments](https://developer.mozilla.org/en-US/docs/DOM/DocumentFragment) 是DOM节点。它们不是主DOM树的一部分。通常的用例是创建文档片段，将元素附加到文档片段，然后将文档片段附加到DOM树。在DOM树中，文档片段被其所有的孩子所代替。



碰撞测试；

# JavaScript getTime() 方法

[JavaScript Date 对象](http://www.w3school.com.cn/jsref/jsref_obj_date.asp)

## 定义和用法

getTime() 方法可返回距 1970 年 1 月 1 日之间的毫秒数。

### 语法

dateObject.getTime()

### 返回值

dateObject 指定的日期和时间距 1970 年 1 月 1 日午夜（GMT 时间）之间的毫秒数。

// div1.nextSibling.style.color="red";  获取下一个兄弟节点 只有IE 6 7 8兼容  
// div1.nextElementSibling.style.color="red";   其他浏览器兼容

var one = div1.nextElementSibling || div1.nextSibling;// 所以用这种处理兼容的办法：中间通过一个变量，获取其中一个值   注意：先写正常浏览器，在写要去兼容的IE其他浏览器

正好这段时间在重新看这部分，写一篇回答来梳理一下吧。

\_\_proto\_\_（隐式原型）与prototype（显式原型）

1.**是什么**

* 显式原型 explicit prototype property：

每一个函数在创建之后都会拥有一个名为prototype的属性，这个属性指向函数的原型对象。

**Note**：通过Function.prototype.bind方法构造出来的函数是个例外，它没有prototype属性。

* 隐式原型 implicit prototype link：

JavaScript中任意对象都有一个内置属性[[prototype]]，在ES5之前没有标准的方法访问这个内置属性，但是大多数浏览器都支持通过\_\_proto\_\_来访问。ES5中有了对于这个内置属性标准的Get方法Object.getPrototypeOf().   
Note: Object.prototype 这个对象是个例外，它的\_\_proto\_\_值为null

* 二者的关系：

隐式原型指向**创建**这个对象的函数(constructor)的prototype

显式原型的作用：用来实现基于原型的继承与属性的共享。

隐式原型的作用：构成原型链，同样用于实现基于原型的继承。举个例子，当我们访问obj这个对象中的x属性时，如果在obj中找不到，那么就会沿着\_\_proto\_\_依次查找。

**\_\_proto\_\_的指向**  
\_\_proto\_\_的指向到底如何判断呢？根据ECMA定义 'to the value of its constructor’s "prototype" ' ----指向创建这个对象的函数的显式原型。所以关键的点在于找到创建这个对象的构造函数，接下来就来看一下JS中对象被创建的方式，一眼看过去似乎有三种方式：（1）对象字面量的方式 （2）new 的方式 （3）ES5中的Object.create() 但是我认为本质上只有一种方式，也就是通过new来创建。为什么这么说呢，首先字面量的方式是一种为了开发人员更方便创建对象的一个语法糖，本质就是 var o = new Object(); o.xx = xx;o.yy=yy; 再来看看Object.create(),这是ES5中新增的方法，在这之前这被称为原型式继承，

因此由Object.create(o)创建出来的对象它的隐式原型指向o。好了，对象的创建方式分析完了，现在你应该能够判断一个对象的\_\_proto\_\_指向谁了。

好吧，还是举一些一眼看过去比较疑惑的例子来巩固一下。

* 构造函数的显示原型的隐式原型：

1. 内建对象(built-in object)：比如Array()，Array.prototype.\_\_proto\_\_指向什么？Array.prototype也是一个对象，对象就是由 Object() 这个构造函数创建的，因此Array.prototype.\_\_proto\_\_ === Object.prototype //true，或者也可以这么理解，所有的内建对象都是由Object()创建而来。

* 自定义对象

1. 默认情况下：

function Foo(){}

var foo = new Foo()

Foo.prototype.\_\_proto\_\_ === Object.prototype //true 理由同上

2. 其他情况：   
（1）

function Bar(){}

//这时我们想让Foo继承Bar

Foo.prototype = new Bar()

Foo.prototype.\_\_proto\_\_ === Bar.prototype //true

（2）

//我们不想让Foo继承谁，但是我们要自己重新定义Foo.prototype

Foo.prototype = {

a:10,

b:-10

}

//这种方式就是用了对象字面量的方式来创建一个对象，根据前文所述

Foo.prototype.\_\_proto\_\_ === Object.prototype

**注**： 以上两种情况都等于完全重写了Foo.prototype，所以Foo.prototype.constructor也跟着改变了，于是乎constructor这个属性和原来的构造函数Foo（）也就切断了联系。

* 构造函数的隐式原型

既然是构造函数那么它就是Function（）的实例，因此也就指向Function.prototype,比如 Object.\_\_proto\_\_ === Function.prototype

**4. instanceof**instanceof 操作符的内部实现机制和隐式原型、显式原型有直接的关系。instanceof的左值一般是一个对象，右值一般是一个构造函数，用来判断左值是否是右值的实例。它的内部实现原理是这样的：

//设 L instanceof R

//通过判断

L.\_\_proto\_\_.\_\_proto\_\_ ..... === R.prototype ？

//最终返回true or false

也就是沿着L的\_\_proto\_\_一直寻找到原型链末端，直到等于R.prototype为止。知道了这个也就知道为什么以下这些奇怪的表达式为什么会得到相应的值了

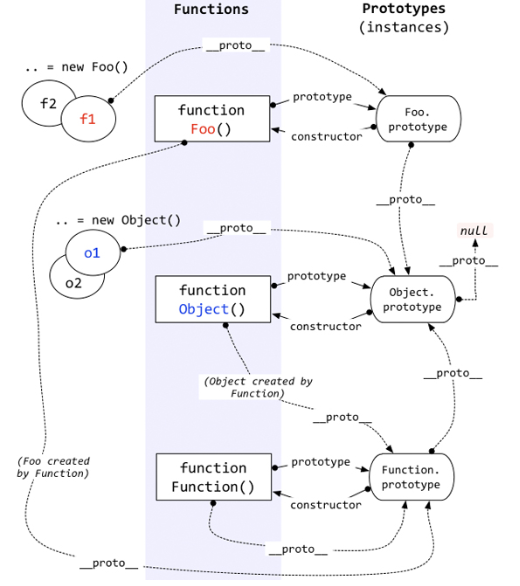
Function instanceof Object // true

Object instanceof Function // true

Function instanceof Function //true

Object instanceof Object // true

Number instanceof Number //false



一、代码片段：

1、Class定义如下

**[javascript]** [view plain](http://blog.csdn.net/beitiandijun/article/details/39579425) [copy](http://blog.csdn.net/beitiandijun/article/details/39579425)

1. **var** Class = {
2. create: **function**() {
3. **return** **function**() {
4. **this**.initialize.apply(**this**, arguments); }
5. }
6. }

2、Class使用方式：

**[javascript]** [view plain](http://blog.csdn.net/beitiandijun/article/details/39579425) [copy](http://blog.csdn.net/beitiandijun/article/details/39579425)

1. **var** SimpleDrag = Class.create();
2. SimpleDrag.prototype = {
3. initialize: **function**(drag) {
4. },
5. Start: **function**(oEvent) {
7. },
8. Move: **function**(oEvent) {
9. },
10. Stop: **function**() {
11. }
12. };

二、分析

1、Class定义的另一种写法可以为：

**[javascript]** [view plain](http://blog.csdn.net/beitiandijun/article/details/39579425) [copy](http://blog.csdn.net/beitiandijun/article/details/39579425)

1. **var** Class = {
2. create: **function**() {
3. **return** **this**.initialize.apply(**this** , arguments);
4. }
5. }

或者不适用apply方法

**[javascript]** [view plain](http://blog.csdn.net/beitiandijun/article/details/39579425) [copy](http://blog.csdn.net/beitiandijun/article/details/39579425)

1. **var** Class = {
2. create: **function**() {
3. **return** **this**.initialize(arguments[0], arguments[1].....);
4. }
5. }

2、create方法的作用：

         Class对象的create方法返回的是对象的构造函数，即create方法的作用就是进行对象实例的初始化。

3、initialize方法的作用：

         这里的构造函数就是initialize函数，而实际上这里的initialize函数只是声明，并没有实际的定义。 这里的Create声明实际上是将initialize方法封装了一下，更加的明确功能。initialize函数的定义是在原型（prototype）中进行的。当使用Class对象的create方法定义新对象实例时，就会执行initialize方法，即执行已经声明好的构造函数。initialize只是你所声明的构造函数的名字，如果你有兴趣，你可以将任何名字定义为构造函数，当使用create方法创建新对象实例时，就会执行你所声明构造函数。

3、apply方法的作用：

         call和apply看做是某个对象的方法，目的是通过间接方式进行调用函数。call和apply的第一个实参是要调用函数的母对象，call方法是将所有参数罗列出来，而apply是将所有参数放在一个数组中。所以使用apply方法比较利于传递参数。