## JS半知半解

**1.offsetleft:**

此属性可以返回当前元素距离某个父辈元素左边缘的距离，当然这个父辈元素也是有讲究的。

(1).如果父辈元素中有定位的元素，那么就返回距离当前元素最近的定位元素边缘的距离。

(2).如果父辈元素中没有定位元素，那么就返回相对于body左边缘距离。

语法结构:

obj.offsetleft

特别说明:此属性是只读的，不能够赋值。

**Cookie: 默认是临时存储的**

indexOf() 方法可返回某个指定的字符串值在字符串中首次出现的位置。

substring() 方法用于提取字符串中介于两个指定下标之间的字符。

decodeURIComponent() 函数可对 encodeURIComponent() 函数编码的 URI 进行解码

decodeURIComponent(URIstring)

encodeURIComponent() 函数可把字符串作为 URI 组件进行编码。

document.cookie = “username=marh”;

alert(document.cookie) // username=marh; age=23 字符串类型，用‘; ’分隔开

如果我们想要长时间存放cookie：需要在设置这个cookie的时候同时设置一个过期的时间: document.cookie = “名称=值;expires=” + 时间，时间必须是str形式

var oDate = new Date()

oDate.setDate( oDate.getDate() + 30 ); // 30天后过期

oDate.toUTCString()

document.cookie = “username=marh;expires=” + oDate;

内容最好用编码存放，防止有些特殊符号不能显示，例如：\n换行。

encodeURI()

decodeURI( document.cookie )

删除cookie: 删除时不必指定 cookie 的值，过期时间设为负数

document.cookie = "username=; expires=” + -1

HTML5 提供了两种在客户端存储数据的新方法：

localStorage - 没有时间限制的数据存储

sessionStorage - 针对一个 session 的数据存储

-----------------------------------------------------------------------

var obj = {

toString: function() {

return 'obj';

},

valueOf: function() {

return 1;

}

};

console.log(1 + obj); // 2

console.log('1' + obj); // 11

// 也很简单

var strtime = '2014-04-23 18:55:49:123';

var date = new Date(strtime); //传入一个时间格式，如果不传入就是获取现在的时间了，这样做不兼容火狐。

// 可以这样做

var date = new Date(strtime.replace(/-/g, '/'));

// 有三种方式获取，在后面会讲到三种方式的区别

time1 = date.getTime();

time2 = date.valueOf();

time3 = Date.parse(date);

/\*

三种获取的区别：

第一、第二种：会精确到毫秒

第三种：只能精确到秒，毫秒将用0来代替

比如上面代码输出的结果(一眼就能看出区别)：

1398250549123

1398250549123

1398250549000

\*/

一：时间转时间戳：javascript获得时间戳的方法有四种，都是通过实例化时间对象 new Date() 来进一步获取当前的时间戳

1.var timestamp1 = Date.parse(new Date()); // 结果：1477808630000 不推荐这种办法，毫秒级别的数值被转化为000

console.log(timestamp1);

2.var timestamp2 = (new Date()).valueOf(); // 结果：1477808630404 通过valueOf()函数返回指定对象的原始值获得准确的时间戳值

console.log(timestamp2);

3.var timestamp3 = new Date().getTime(); // 结果：1477808630404 ，通过原型方法直接获得当前时间的毫秒值，准确

console.log(timestamp3);

4.var timetamp4 = Number(new Date()) ; //结果：1477808630404 ,将时间转化为一个number类型的数值，即时间戳

console.log(timetamp4);

**2.map()**

易犯错误:

通常情况下，map 方法中的 callback 函数只需要接受一个参数（很多时候，自定义的函数形参只有一个），就是正在被遍历的数组元素本身。

但这并不意味着 map 只给 callback 传了一个参数（会传递3个参数）。这个思维惯性可能会让我们犯一个很容易犯的错误。

// 下面的语句返回什么呢:

["1", "2", "3"].map(parseInt);

// 你可能觉的会是[1, 2, 3]

// 但实际的结果是 [1, NaN, NaN]

// 通常使用parseInt时,只需要传递一个参数.但实际上,parseInt可以有两个参数.第二个参数是进制数.可以通过语句"alert(parseInt.length)===2"来验证.

// map方法在调用callback函数时,会给它传递三个参数:当前正在遍历的元素, 元素索引, 原数组本身.

// 第三个参数parseInt会忽视, 但第二个参数不会,也就是说,parseInt把传过来的索引值当成进制数来使用.从而返回了NaN.

/\*

//应该使用如下的用户函数returnInt

function returnInt(element){

return parseInt(element,10);

}

["1", "2", "3"].map(returnInt);

// 返回[1,2,3]

\*/

**3.事件代理**

var ev = ev || window.event;

var target = ev.target || ev.srcElement;

IE/Opera：window.event

FF：event

IE：window.event.srcElement ，

FF：window.event.target；

**滚轮事件：**

ie/chrome：onmousewheel

event.wheelDelta 上：120 下：-120

FF: DOMMouseSroll, 必须用addEventListener 绑定

event.detail 上：-3，下：3

阻止事件的默认行为:

return false; // 阻止的是 obj.on事件名称=fn所触发的默认行为

preventDefault; // addEventListener绑定的事件需要通过event

下面的preventDefault();

去google一搜，全都是说return false = event.preventDefault() + stop.stopPropagation()。

我去尝试了一下，发现完全不是这么回事。

事件处理程序的返回值只对通过属性注册的处理程序才有意义。

也就是说:如果我们是直接不通过addEventListener()函数来绑定事件的话，我们要禁止默认事件的话，用的就是return false。

但是如果我们要用addEventListener()或者attachEvent()来绑定的话，就要用preventDefault()方法或者设置事件对象的returnValue属性。

**焦点事件：**

onfocus

obblur

**clientX, client:**

当一个事件发生的时候，鼠标到页面可视区的距离。

**文档高宽及窗口事件**

**可视区域尺寸**

document.documentElement.clientWidth

document.documentElement.clientHeight

**滚动距离**

chrome:document.body.scrollTop/scrollLeft;

ie: document.documentElement.scrollTop/scrollLeft;

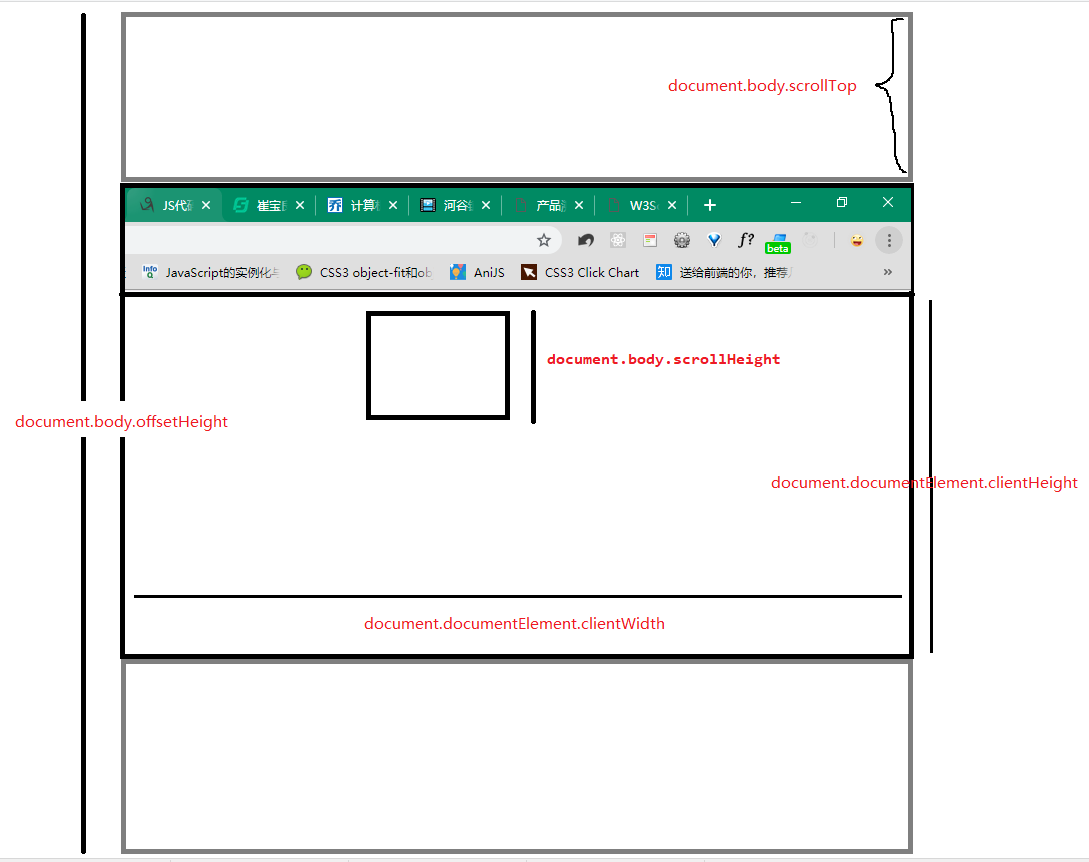
**内容高度（元素内容的总高度）**

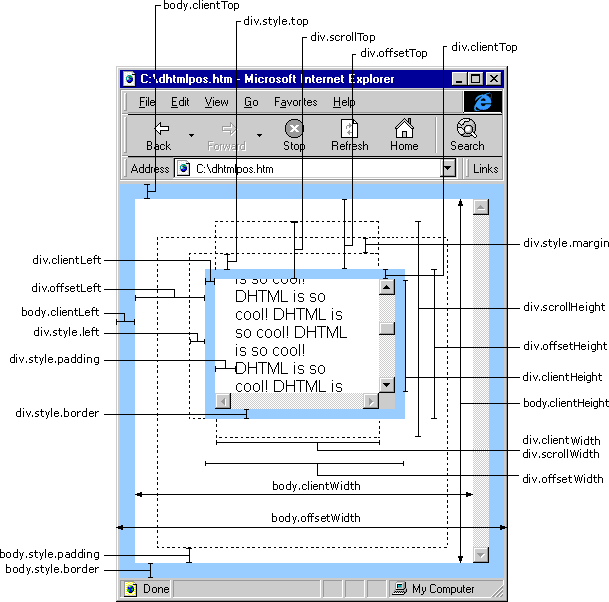
document.body.scrollHeight

**文档高度**

document.documentElement.offsetHeight

document.body.offsetHeight





Event

onscroll:当滚动条滚动的时候触发

onresize:当页面窗口大小发生改变的时触发

键盘事件

onkeydown/onkeyup, ev.keycode

ctrlKey /altKey/shiftKey

拖拽功能 （牛刀小试 > JS BOM\EVENT）

拖拽时，如果有文字被选中会产生问题。

原因：当鼠标按下的时候，如果页面中有文字被选中，会触发浏览器中默认拖拽文字的效果

解决：

标准下：阻止onmousedown的默认行为，return false；

非标准下：obj鼠标按下时设置 setCapture,抬起清除

obj.setCapture() //设置全局捕获，当我们给一个元素设置全局捕获，这个元素就会监听后续 发生的所有事件，当有事件发生，就会被obj所触发。

将浏览器缩小，点击浏览器外也可以捕获。

ie: 有该方法， 有效果

ff: 有该方法， 但是没有效果

chrome：没有该方法，会报错

obj.releaseCapture() 释放全局捕获

**4.快速删除node\_modules**

npm install **rimraf** -g

rimraf node\_modules

**5.display 和 visibility，display与transition的冲突**

由于display:none会引起页面的重绘事件，所以它是一个异步的延时事件，所以浏览器其实会先解析animate的代码，然后再执行display:none。

**可以用 visibility来代替display。**

！！！ 但元素现在可以点击吗？答案是不可以的。

设为visibility:hidden后，元素现在是不可以选择的既然无法选择，即使绑定了click事件，也自然无法点击啦！

**6.模块（你不知道的javascript 上卷 CH5.5）**

**(1)模块**

**function CoolModule(){**

**var something = ‘cool’;**

**var another = [1,2,3];**

**function doSomething(){**

**console.log(something);  
 }**

**function doAnother(){**

**console.log(another.join(“ ! ”));  
 }**

**return{**

**doSomething: doSomething,  
 doAnother: doAnother**

**}**

**}**

**var foo = CoolModule();**

**foo.doSomething(); // cool**

**foo.doAnother(); // 1 ! 2 ! 3**

**将(1)修改成单例模式：**

**var foo = (function CoolModule(){**

**var something = ‘cool’;**

**var another = [1,2,3];**

**function doSomething(){**

**console.log(something);  
 }**

**function doAnother(){**

**console.log(another.join(“ ! ”));  
 }**

**return{**

**doSomething: doSomething,  
 doAnother: doAnother**

**}**

**})();**

**foo.doSomething(); // cool**

**foo.doAnother(); // 1 ! 2 ! 3**

* 1. **通过在模块实力的内部保留对公共API对象的引用，可以从内部对模块实例进行修改，包括添加或删除方法和属性，以及修改他们的值**

**var foo = (function CoolModule(id){**

**function change(){**

**publicAPI.identify = identify2;**

**}**

**function identify1(){**

**console.log(id);**

**}**

**function identify2(){**

**console.log(id.toUpperCase());**

**}**

**var publicAPI = {**

**change: change,**

**identify: identify1**

**}**

**return publicAPI;**

**})(‘foo module’);**

**foo.identify(); // foo module**

**foo.change();**

**foo.identify(); // FOO MODULE**

**7.实时请求JSON数据**

**会涉及到浏览器缓存的问题导致JSON中数据修改后，页面内容没有改变。**

**因此在ajax发送请求前加上 xmlHttpRequest.setRequestHeader(“Cache-Control”,”no-cache”);**

xmlhttp.open("GET", url);

**xmlhttp.setRequestHeader("Cache-Control","no-cache");**

xmlhttp.send();

**8.对象（你不知道的javascript 上卷 CH3）**

**3.3.4 复制对象**

* + 1. **浅拷贝**

**var a = {  
 “name” : “Robot”,**

**“msg” : {  
 “year” : 18,  
 “height”: 182  
 }  
}**

* + 1. **function easyCopy(o){  
        if(typeof(o) != ‘object’){return o;}**
    2. **for(var i in )  
       }**
    3. **深拷贝**

**9.去除字符串空格**

去除所有空格:

str = str.replace(/\s+/g,"");

去除两头空格:

str = str.replace(/^\s+|\s+$/g,"");

去除左空格：

str = str.replace( /^\s\*/, '');

去除右空格：

str = str.replace(/(\s\*$)/g, "");

**10.Ajax**

**设置请求头**

if (window.XMLHttpRequest) {

**// IE7+, Firefox, Chrome, Opera, Safari**

xmlhttp = new XMLHttpRequest();

} else {

**// IE6, IE5**

xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");

}

xmlhttp.onreadystatechange = function(){

if(xmlhttp.readyState == 4 && xmlhttp.status == 200){

var data = xmlhttp.response;

TYPENAME = JSON.parse(data);

}

}

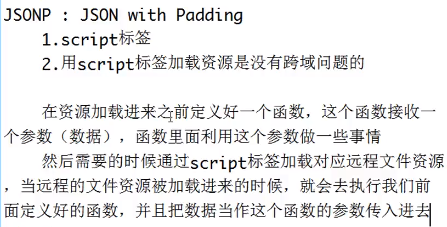
xmlhttp.open("GET","");

**// 实时的读取JSON文件时，因为在浏览器中会有缓存，所以设置请求头**

**xmlhttp.setRequestHeader("Cache-Control","no-cache");**

xmlhttp.send();

**跨域请求 jsonP**



<script>

**function callback(data){ alert(data) }**

</script>

<script src=`2.txt`></script>

**文件2.txt:** callback([1,2,3])

**向后台传输数据*（ POST方法发送数据）***

var sendData = { 'name':'bob' }; **xmlhttp.send(JSON.stringify(sendData));**

**flask 获取前台ajax传输的数据**

**data\_json=request.get\_data().decode('utf-8')**

**data\_dict=json.loads(data\_json)**

**11.JS快速解析URL**

function **parseURL**(url) {

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

a.href = url;

*// var a = new URL(url);*

return {

source: url,

protocol: a.protocol.**replace**(':', ''),

host: a.hostname,

port: a.port,

query: a.search,

params: (function() {

var params = {},

seg = a.search.**replace**(/^\?/, '').**split**('&'),

len = seg.length,

p;

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

if (seg[i]) {

p = seg[i].**split**('=');

params[p[0]] = p[1];

}

}

return params;

})(),

hash: a.hash.**replace**('#', ''),

path: a.pathname.**replace**(/^([^\/])/, '/$1')

};

}

**console**.**log**(**parseURL**('https://test.com:8080/path/index.html?name=angle&age=18#top'));



**12.JS获取非行内样式**

let style = **window.getComputedStyle**(element, [pseudoElt]);

IE：obj.**currentStyle()**

**element**

用于获取计算样式的[Element](https://developer.mozilla.org/zh-CN/docs/Web/API/Element)。

**pseudoElt** 可选

指定一个要匹配的伪元素的字符串。必须对普通元素省略（或null）。

**13.JS中 this 和 prototype 的区别**

var **Foo** = function(name){

var age = 18;

this.name = name;

this.**say** = function(){

**console**.**log**(this);

**console**.**log**(age); // 18

**console**.**log**(this.name);

}

}

**Foo**.prototype = {

**speak**:function(){

**console**.**log**(this);

**console**.**log**(this.age); // undefined

**console**.**log**(this.name);

}

}

var foo = new **Foo**('bob');

**区别1：**

利用 this 实现的公共方法中可以访问类的私有成员(用 var 声明的变量)，私有方法(用 function 直接定义的方法)；

利用原型扩展实现的方法中，无法调用私有成员和变量。

**区别2：**

当访问对象的属性或者方法时，将按照搜索原型链的规则进行。先查找自身的静态属性、方法，继而查找构造上下文的可访问属性、方法，最后查找构造的原型链

function **Test**() {

this.text = function() {

**alert**("defined by this");

}

}

**Test**.prototype.**test** = function() {

**alert**("defined by prototype");

}

var \_o = new **Test**();

\_o.**test**(); *//输出"defined by this"*

**区别3：**

“this”与“prototype”定义的另一个不同点是属性的占用空间不同。使用“this”关键字，示例初始化时为每个实例开辟构造方法所包含的所有属性、方法所需的空间，而使用“prototype”定义，由于“prototype”实际上是指向父级的一种引用，仅仅是个数据的副本，因此在初始化及存储上都比“this”节约资源。

## 你所不知道的Javascript (中)

**CH2.值**

**2.1.2 数组**

**1.使用delete可以将单元从数组中删除，但是length属性不会发生变化**

**2.如果字符串的键值能够被强制类型转换为十进制数字的话，他就会被当作数字索引来处理**

**!!! 数组中加入字符串键值/属性**

**a[“13”]=42;  
 a.length // -> 14**

**3.**

**4.**

**2.1.2 数组方法**

**join()**

**var arr = [1,2,3];**

**console.log(arr.join());** // 1,2,3

**console.log(arr.join("-"));** // 1-2-3

**console.log(arr);** // [1, 2, 3]（原数组不变）

**push()** 返回修改后数组的长度

**pop()** 返回移除的项 **shift() 和 unshift()  
 sort()  
 reverse()  
 concat()  
 slice()  
 splice()  
 indexOf()和 lastIndexOf() （ES5新增）  
 forEach() （ES5新增）  
 map() （ES5新增）  
 filter() （ES5新增）  
 every() （ES5新增）  
 some() （ES5新增）  
 reduce()和 reduceRight() （ES5新增）**