**BOM的概念**

BOM 是Browser Object Model的缩写，简称浏览器对象模型, 提供了独立于内容而与浏览器窗口进行交互的对象，用于访问浏览器的功能。

JavaScript语法的标准化组织是ECMA，DOM的标准化组织是W3C, 但是BOM缺乏标准.由于BOM 缺少规范，每个浏览器提供商又按照自己想法去扩展它，那么浏览器共有对象就成了事实的标准。

**window对象**

window对象是BOM的核心；

window对象是表示整个浏览器窗口的一个对象, 每个浏览器窗口都只有一个window对象与之对应. 同时window也是网页中的全局对象

window对象是BOM的顶层(核心)对象，所有对象都是通过它延伸出来的.

**window的属性对象**

BOM对象在浏览器中都是作为window对象的属性保存的

可以通过window对象来使用，也可以直接使用

**document(核心):**

文档对象，让我们可以在js脚本中直接访问页面元素(DOM)

**history(重要):**

历史对象,代表浏览器的历史记录，可以通过该对象来操作浏览器的历史记录

由于隐私原因，该对象不能获取到具体的历史记录，只能操作浏览器向前或者向后翻页

而且该操作只在当次访问有效

**属性**

history.length; //history对象中的记录数 获取到当前访问的链接数量

**history的方法：**

**1.前往浏览器历史条目前一个URL, 作用和浏览器的回退按钮一样**

**语法：**

history.back();

**2.前往浏览器历史条目下一个URL, 作用和浏览器的前进按钮一样**

**语法：**

history.forward();

**3.浏览器在history对象中向前或向后**

**语法：**

history.go(-1); //(-1代表后退, 1代表向前,0表示刷新当前页面)

参数：主要一个整数作为参数

1:表示向前跳转一个页面 相当于forward()

2:表示向前跳转两个页面

-1:表示向后跳转一个页面

-2:表示向后跳转两个页面

**location(重要):**

代表浏览器当前的地址栏信息，可以用来刷新本页面或跳转到新页面。它提供了与当前窗口中加载的文档有关的信息, 还提供了一些导航功能。
事实上，location对象是window对象的属性，也是document对象的属性; 所以window.location和document.location等效。

如果直接打印location则可以获取到地址栏的信息

**完整路径：href**

**协议：protocol**

http 默认端口80

https 默认端口443 加密（下载证书，比较安全）

**语法：**

location.protocol//获取当前协议

**主机：host**

主机名+端口

**语法：**

location.host//获取主机名+端口

location.hostname//获取主机名

location.port//获取端口

**路径：**

路径名+查询字符串 | search = ？+ 参数

**语法：**

location.path//获取路径

location.pathname//获取路径名

location.search//获取查询字符串

参数分为?+query

query分类键和值:query = string

**哈希：**

能实现单页面应用程序（Single Page Application）

好处：提高用户体验 减少重绘和回流

**语法：**

location.hash = ‘#1’; //设置#后的字符串，并跳转

**获取整个路径**

语法：

location.href



**location的方法:**

**1.location的方法值做跳转**

btn.onclick = function (){

//如果直接将location属性修改为一个完整的路径，或相对路径

//则我们页面会自动跳转到该路径，并且会生成相应的历史记录

location = "https://www.baidu.com";

//用来跳转到其他的页面，作用和直接修改location一样

location.href = "https://www.baidu.com";

location.assign("https://www.baidu.com");

//前面三种跳转都能回退

//第四种只是把地址替换，就不能回退

//用新的URL替代,可以避免产生历史记录

//可以使用一个新的页面替换当前页面，调用完毕也会跳转页面

//不会生成历史记录，不能使用回退按钮回退

location.replace("https://www.baidu.com");

}

**2.强制重新加载 从服务器源头重新加载**

**语法：**

location.reload(true);

用于重新加载当前页面，作用和刷新按钮一样

- 如果在方法中传递一个true，作为参数，则会强制清空缓存刷新页面

**frames:**

框架对象，可以获取页面框架内容

**screen:**

包含有关客户端显示屏幕的信息，通过该对象可以获取到用户的显示器的相关的信息

**navigator:**

导航对象, 它包含了浏览器的信息, 如: 浏览器名称,版本,浏览器所在的电脑操作系统等。通过该对象可以识别不同的浏览器

navigator.appName; //浏览器名称

navigator.appVersion; //浏览器版本

navigator.platform; //操作系统

//最新的浏览器已经全面放弃以上这些属性

由于历史原因，Navigator对象中的大部分属性都已经不能帮助我们识别浏览器了

一般的我们只使用userAgent来判断浏览器的信息

navigator.userAgent); //用户代理信息, 通过该属性可以获取浏览器及操作系统信息

如果通过UserAgent不能判断，还可以通过一些浏览器中特有的对象，来判断浏览器的信息

比如：ActiveXObject

if("ActiveXObject" in window){

alert("你是IE11，枪毙了你~~~");

}

**重绘和回流**

当Render Tree(渲染树)中部分或全部元素的尺寸、结构、或某些属性发生改变时，浏览器重新渲染部分或全部文档的过程称为回流。

导致回流的操作：

1、页面首次渲染

2、浏览器窗口大小发生改变3、元素尺寸或位置发生改变

4、元素内容变化(文字数量或图片大小改变而引起的计算值宽度和高度改变)

5、元素字体大小变化

6、添加或者删除可见的DOM元素

7、激活css伪类〔例如: :hover)

8、查询某些属性或调用某些方法

9、offsetwidth ,width,clientwidth,scrollTop/scrollHeight的计算,会使浏览器将渐进回流队列Flush,立即执行回流。

当页面中元素样式的改变并不影响它在文档流中的位置时（例如: color、background-color、visibility等)，浏览器会将新样式赋予给元素并重新绘制它，这个过程称为重绘。

回流一定会发生重绘，重绘不一定会发生回流

一个是返厂 一个是改装

回流能避免就避免，重绘避免不了。

影响性能的是回流，不是重绘

**如何避免重绘和回流？**

**css**

1.避免使用table布局，可能很小的一个小改动会造成整个table的重新布局

2.尽可能在DOM树的最末端改变class。

3.避免设置多层内联样式。

4.将动画效果应用到position属性为absolute或fixed的元素上。

5.动画实现的速度的选择，动画速度越快，回流次数越多，也可以选择使用requestAnimationFrame(相当于定时器，根据刷新率执行)

6.避免使用CSS表达式（例如: ca1c() )

7.使用transform替代top

8.使用visibility替换display: none，因为前者只会引起重绘，后者会引发回流(改变了布局)将频繁重绘或者回流的节点设置为图层，图层能够阻止该节点的渲染行为影响别的节点

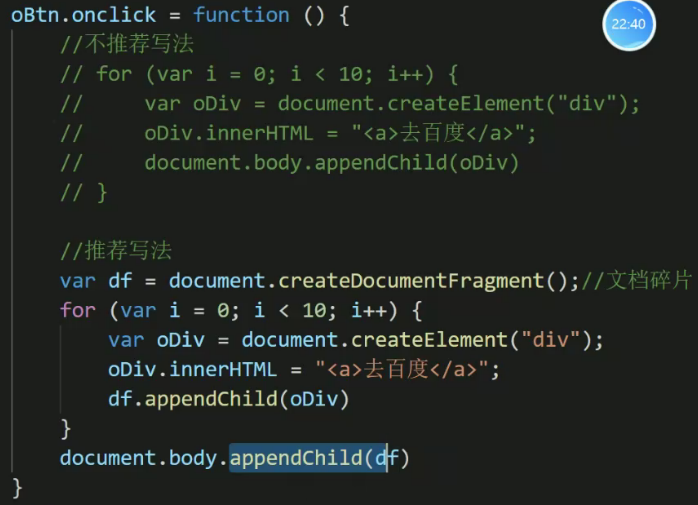
**js**

1.避免频繁操作样式，最好一次性重写style属性,cssText，或者将样式列表定义为class并一次性更改class属性。

//推荐

oDiv.style.cssText="width:200px;height: 200px;background-color:pink;"

2.避免频繁操作DOM，创建一个documentFragment，在它上面应用所有DOM操作，最后再把它添加到文档中。



3.也可以先为元素设置display: none，操作结束后再把它显示出来。因为在display属性为none的元素上进行的DOM操作不会引发回流和重绘。

4.避免频繁读取会引发回流/重绘的属性，如果确实需要多次使用，就用一个变量缓存起来。

5.对具有复杂动画的元素使用绝对定位，使它脱离文档流，否则会引起父元素及后续元素频繁回流。

**window的方法**

**alert(text):**

弹出提示框(警告框)

**confirm():**

创建一个需要用户确认的对话框

**prompt(text,defaultInput) :**

创建一个对话框要求用户输入信息

**open(url,name,[options]) :**

打开一个新窗口并返回新 window 对象

第一个参数：链接

第二个参数：打开方式 可以省略

第三个参数：打开窗口的大小、位置

open("http://www.baidu.com","\_self","width=300px,height=300px,left=200px, top=200px");

**setInterval():**

设置定时器

**clearInterval() :**

移除定时器

**setTimeOut() :**

设置延时器

**clearTimeOut():**

移除延时器

**close():**

关闭窗口

**print():**

调出打印对话框

window对象的属性和方法的调用方式，可以使用 window.属性、window.方法()或者一般可以直接使用: 属性、方法()的方式调用。

例如：window.alert() 和 alert()效果是一样的

**延时器**

**setTimeOut()：设置延时器**

语法：

var timer = setTimeOut(fn,1000);//传入的函数参数不能加括号

var timer = setTimeOut(function(){

console.log("1111");

},1000)

如果要给函数传递参数则添加第三个（或第3-n个）参数

var timer = setTimeOut(function(str){

console.log(str);

},1000,"刘德华")

**clearTimeOut()：关闭延时器 也能关闭定时器**

**定时器**

**setInterval()：设置定时器**

var timer = setInterval(fn,1000);//传入的函数参数不能加括号

var timer = setInterval(function(){

console.log("1111");

},1000)

如果要给函数传递参数则添加第三个（或第3-n个）参数

var timer = setInterval(function(str){

console.log(str);

},1000,"刘德华")

**clearInterval():关闭定时器 也能关闭延时器**

返回值：

返回一个Number类型的数据

这个数字用来作为定时器的唯一标识

定时器里的this默认是window

1.可以在定时器的函数外创建一个变量接收this然后在定时器的函数内使用这个变量

2.bind()函数可以改变函数内部的this指向

**动画函数**

//尝试创建一个可以执行简单动画的函数

/\*

**\* 参数：**

\* obj:要执行动画的对象

\* attr:要执行动画的样式，比如：left top width height

\* target:执行动画的目标位置

\* speed:移动的速度(正数向右移动，负数向左移动)

\* callback:回调函数，这个函数将会在动画执行完毕以后执行

\*/

function move(obj, attr, target, speed, callback) {

//关闭上一个定时器

clearInterval(obj.timer);

//获取元素目前的位置

var current = parseInt(getStyle(obj, attr));

//判断速度的正负值

//如果从0 向 800移动，则speed为正

//如果从800向0移动，则speed为负

if(current > target) {

//此时速度应为负值

speed = -speed;

}

//开启一个定时器，用来执行动画效果

//向执行动画的对象中添加一个timer属性，用来保存它自己的定时器的标识

obj.timer = setInterval(function() {

//获取box1的原来的left值

var oldValue = parseInt(getStyle(obj, attr));

//在旧值的基础上增加

var newValue = oldValue + speed;

//判断newValue是否大于800

//从800 向 0移动

//向左移动时，需要判断newValue是否小于target

//向右移动时，需要判断newValue是否大于target

newValue = target;

}

//将新值设置给box1

obj.style[attr] = newValue + "px";

//当元素移动到0px时，使其停止执行动画

if(newValue == target) {

//达到目标，关闭定时器

clearInterval(obj.timer);

//动画执行完毕，调用回调函数

//有回调函数才调 没有就不调

callback && callback();

}

}, 30);

**调用：**

move(box2 , "width" , 100 , 10 , function(){

alert("111");

});