BOM（浏览器对象模型）

一、BOM概述

BOM（Browser Object Model）即浏览器对象模型，它提供了独立于内容而与浏览器窗口进行交互的对象，其核心对象是 window。

BOM 由一系列相关的对象构成，并且每个对象都提供了很多方法与属性。

BOM 缺乏标准，JavaScript 语法的标准化组织是 ECMA，DOM 的标准化组织是 W3C，BOM 最初是 Netscape 浏览器标准的一部分。

DOM

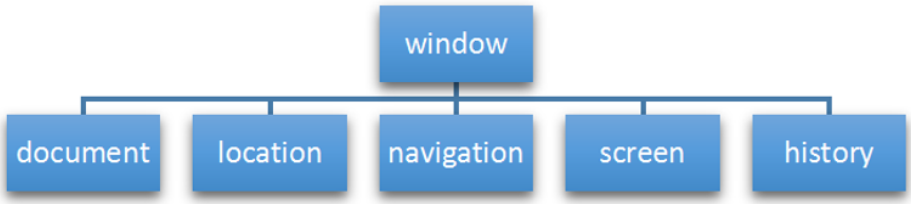
- 文档对象模型
- DOM 就是把「文档」当做一个「对象」来看待
- DOM 的顶级对象是 document
- DOM 主要学习的是操作页面元素
- DOM 是 W3C 标准规范

BOM

- 浏览器对象模型
- 把「浏览器」当做一个「对象」来看待
- BOM 的顶级对象是 window
- BOM 学习的是浏览器窗口交互的一些对象
- BOM 是浏览器厂商在各自浏览器上定义的，兼容性较差

BOM的构成

BOM 比 DOM 更大，它包含 DOM。



> window 对象是浏览器的顶级对象，它具有双重角色。
>
> 1. 它是 JS 访问浏览器窗口的一个接口。
> 2. 它是一个全局对象。定义在全局作用域中的变量、函数都会变成 window 对象的属性和方法。
>
> 在调用的时候可以省略 window，前面学习的对话框都属于 window 对象方法，如 alert()、prompt() 等。
>
> 注意：window下的一个特殊属性 window.name

二、window对象的常见事件

1、窗口加载事件

```
window.onload = function(){}
或者
window.addEventListener("load",function(){});
```

> **window.onload 是窗口（页面）加载事件，当文档内容完全加载完成会触发该事件(包括图像、脚本文件、CSS 文件等)，就调用的处理函数。**
>
> 注意：
>
> 1. 有了 window.onload 就可以把 JS 代码写到页面元素的上方，因为 **onload 是等页面内容全部加载完毕，再去执行处理函数。**
> 2. window.onload 传统注册事件方式 只能写一次，如果有多个，会以最后一个 window.onload 为准。
> 3. 如果使用 addEventListener 则没有限制

```
document.addEventListener('DOMContentLoaded',function(){})
```

DOMContentLoaded 事件触发时，仅当DOM加载完成，不包括样式表，图片，flash等等。

Ie9以上才支持

如果页面的图片很多的话, 从用户访问到onload触发可能需要较长的时间, 交互效果就不能实现，必然影响用户的体验，此时用 DOMContentLoaded 事件比较合适。

2、调整窗口大小事件

```
window.onresize = function(){}

window.addEventListener("resize",function(){});
```

> window.onresize 是调整窗口大小加载事件，当触发时就调用的处理函数。
>
> 注意：
>
> 1. 只要窗口大小发生像素变化，就会触发这个事件。
> 2. 我们经常利用这个事件完成响应式布局。 window.innerWidth 当前屏幕的宽度

三、定时器

window 对象给我们提供了 2 个非常好用的方法-定时器。

- setTimeout()
- setInterval()

1、setTimeout()定时器

```
window.setTimeout(调用函数, [延迟的毫秒数]);
```

> **setTimeout() 方法用于设置一个定时器，该定时器在定时器到期后执行调用函数。**
>
> 注意：

1. **window 可以省略。**
2. 这个调用函数可以直接写函数，或者写函数名或者采取字符串'函数名()'三种形式。第三种不推荐
3. 延迟的毫秒数省略默认是 0，如果写，必须是毫秒。
4. 因为定时器可能有很多，所以我们经常给定时器赋值一个标识符。

setTimeout() 这个调用函数我们也称为回调函数 callback

普通函数是按照代码顺序直接调用。而这个函数，需要等待时间，时间到了才去调用这个函数，因此称为回调函数。

简单理解： 回调，就是回头调用的意思。上一件事干完，再回头再调用这个函数。

以前我们讲的 element.onclick = function(){} 或者 element.addEventListener("click", fn); 里面的 函数也是回调函数。

停止setTimeout定时器

```
window.clearTimeout(timeoutID)
```

clearTimeout()方法取消了先前通过调用 setTimeout() 建立的定时器。

注意：

1. window 可以省略。
2. 里面的参数就是定时器的标识符 。

2、setInterval()定时器

```
window.setInterval(回调函数, [间隔的毫秒数]);
```

setInterval() 方法重复调用一个函数，**每隔这个时间**，就去调用一次回调函数。

注意：

1. window 可以省略。
2. 这个调用函数可以直接写函数，或者写函数名或者采取字符串 '函数名()' 三种形式。
3. 间隔的毫秒数省略默认是 0，如果写，必须是毫秒，表示每隔多少毫秒就自动调用这个函数。
4. 因为定时器可能有很多，所以我们经常给定时器赋值一个标识符。
5. 第一次执行也是间隔毫秒数之后执行，之后每隔毫秒数就执行一次。

停止setInterval定时器

```
window.clearInterval(intervalID);
```

clearInterval()方法取消了先前通过调用 setInterval()建立的定时器。

注意：

1. window 可以省略。
2. 里面的参数就是定时器的标识符 。

3、this

this的指向在函数定义的时候是确定不了的，只有函数执行的时候才能确定this到底指向谁，**一般情况下this的最终指向的是那个调用它的对象**

现阶段，我们先了解一下几个this指向

- 全局作用域或者普通函数中this指向全局对象window（注意定时器里面的this指向window）
- 方法调用中谁调用this指向谁
- 构造函数中this指向构造函数的实例

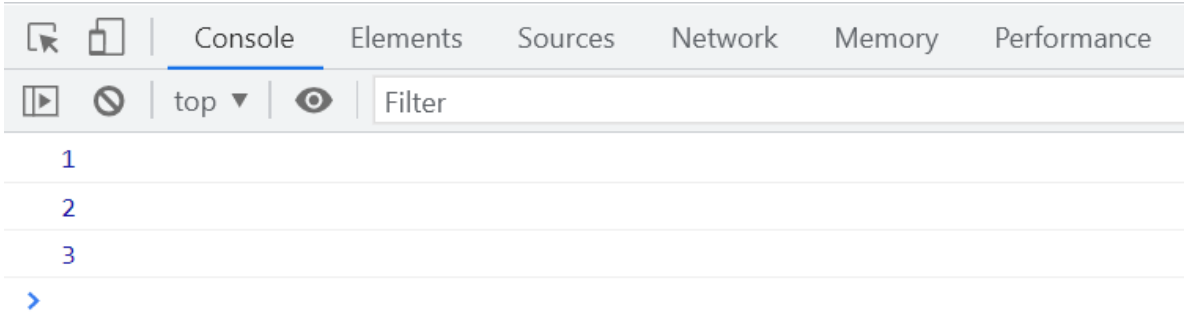
四、JS执行队列

1、JS是单线程

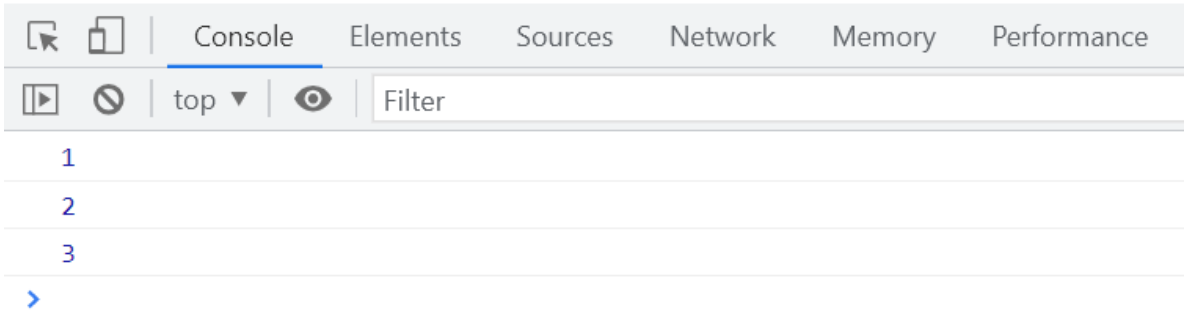
JavaScript 语言的一大特点就是单线程，也就是说，同一个时间只能做一件事。这是因为 Javascript 这门脚本语言诞生的使命所致——JavaScript 是为处理页面中用户的交互，以及操作 DOM 而诞生的。比如我们对某个 DOM 元素进行添加和删除操作，不能同时进行。 应该先进行添加，之后再删除。

单线程就意味着，所有任务需要排队，前一个任务结束，才会执行后一个任务。这样所导致的问题是：如果 JS 执行的时间过长，这样就会造成页面的渲染不连贯，导致页面渲染加载阻塞的感觉。

```
console.log(1)


setTimeout(function () {
  console.log(3)
}, 1000)


console.log(2)
```



```
console.log(1)


setTimeout(function () {
  console.log(3)
}, 0)


console.log(2)
```



2、同步和异步

为了解决这个问题，利用多核 CPU 的计算能力，HTML5 提出 Web Worker 标准，允许 JavaScript 脚本创建多个线程。于是，JS 中出现了同步和异步。

同步

> 前一个任务结束后再执行后一个任务，程序的执行顺序与任务的排列顺序是一致的、同步的。比如做饭的同步做法：我们要烧水煮饭，等水开了（10分钟之后），再去切菜，炒菜。

异步

> 你在做一件事情时，因为这件事情会花费很长时间，在做这件事的同时，你还可以去处理其他事情。比如做饭的异步做法，我们在烧水的同时，利用这10分钟，去切菜，炒菜。

同步任务

同步任务都在主线程上执行，形成一个执行栈。

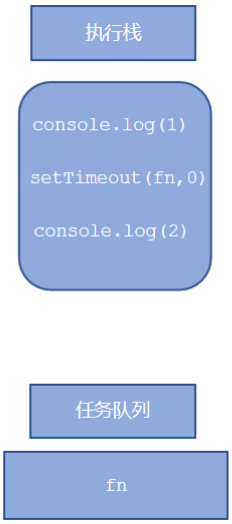
异步任务

JS 的异步是通过回调函数实现的。

一般而言，异步任务有以下三种类型：

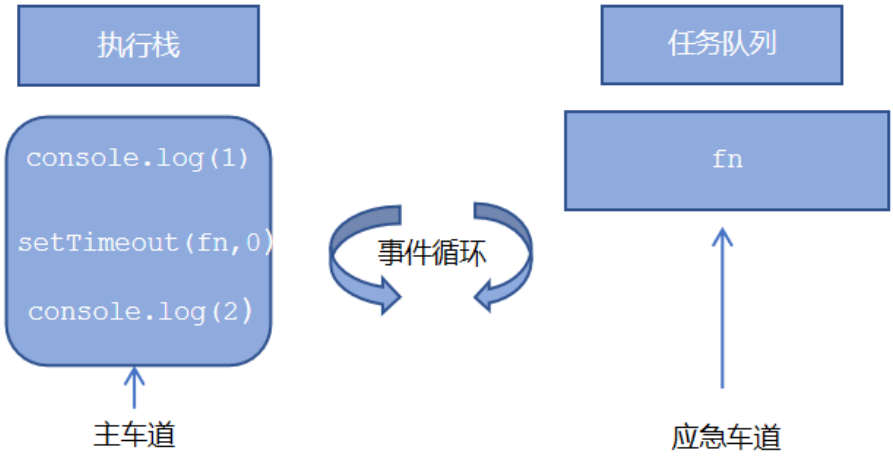
> 1、普通事件，如 click、resize 等
>
> 2、资源加载，如 load、error 等
>
> 3、定时器，包括 setInterval、setTimeout 等

异步任务相关回调函数添加到任务队列中（任务队列也称为消息队列）。

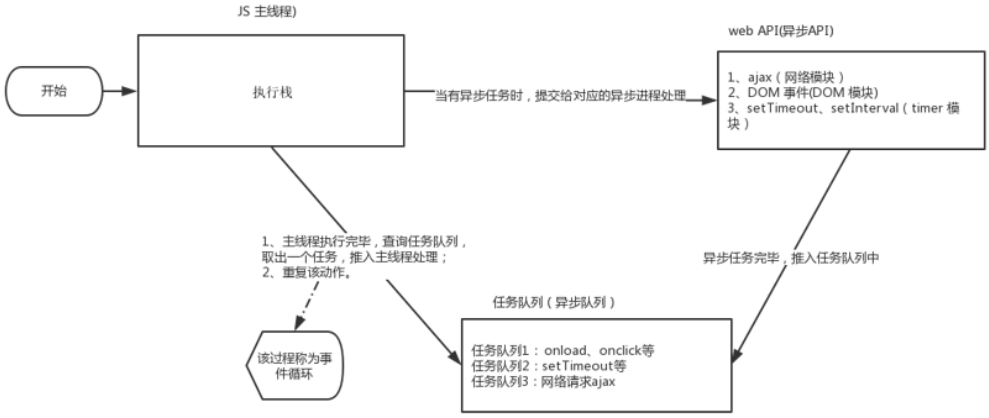


3、JS执行机制！！！

1. 先执行执行栈中的同步任务。
2. **异步任务（回调函数）放入任务队列中。**
3. **一旦执行栈中的所有同步任务执行完毕，系统就会按次序读取任务队列中的异步任务，于是被读取的异步任务结束等待状态，进入执行栈，开始执行。**




```
console.log(1);
document.onclick = function() {
  console.log('click');
}
console.log(2);
setTimeout(function() {
  console.log(3)
}, 3000)
```



由于主线程不断的重复获得任务、执行任务、再获取任务、再执行，所以这种机制被称为事件循环（event loop）。

五、location对象

window 对象给我们提供了一个 location 属性用于获取或设置窗体的 URL，并且可以用于解析 URL 。**因为这个属性返回的是一个对象，所以我们将这个属性也称为 location 对象。**

1、URL

统一资源定位符 (Uniform Resource Locator, URL) 是互联网上标准资源的地址。**互联网上的每个文件都有一个唯一的 URL，它包含的信息指出文件的位置以及浏览器应该怎么处理它。**

URL 的一般语法格式为：

```
protocol://host[:port]/path/[?query]#fragment

http://www.itcast.cn/index.html?name=andy&age=18#link
```

组成	说明
protocol	通信协议 常用的http,ftp,maito等
host	主机（域名） www.itheima.com
port	端口号 可选，省略时使用方案的默认端口 如http的默认端口为80
path	路径 由 零或多个'/'符号隔开的字符串，一般用来表示主机上的一个目录或文件地址
query	参数 以键值对的形式,通过 & 符号分隔开来
fragment	片段 #后面内容 常见于链接 锚点

2、location对象的属性

location对象属性	返回值
location.href	获取或者设置 整个URL
location. host	返回主机（域名） www.itheima.com
location.port	返回端口号 如果未写返回 空字符串
location.pathname	返回路径
location. search	返回参数
location. hash	返回片段 #后面内容 常见于链接 锚点

重点记住： href 和 search

3、location对象的方法

location对象方法	返回值
location.assign()	跟 href 一样，可以跳转页面（也称为重定向页面）
location.replace()	替换当前页面，因为不记录历史，所以不能后退页面
location.reload()	重新加载页面，相当于刷新按钮或者 f5 如果参数为true 强制刷新 ctrl+f5

六、navigator对象

navigator 对象包含有关浏览器的信息，它有很多属性，我们最常用的是 userAgent，该属性可以返回由客户机发送服务器的 user-agent 头部的值。

下面前端代码可以判断用户那个终端打开页面，实现跳转

```
if((navigator.userAgent.match(/(phone|pad|pod|iPhone|iPod|ios|iPad|Android|
Mobile|BlackBerry|IEMobile|MQQBrowser|JUC|Fennec|wOSBrowser|BrowserNG|WebOS
|Symbian|Windows Phone)/i))) {
    window.location.href = "";      //手机
 } else {
    window.location.href = "";      //电脑
 }
```

七、history对象

window 对象给我们提供了一个 history 对象，与浏览器历史记录进行交互。该对象包含用户（在浏览器窗口中）访问过的 URL。

history对象方法	作用
back()	可以后退功能
forward()	前进功能
go(参数)	前进后退功能 参数如果是 1 前进1个页面 如果是-1 后退1个页面

PC端网页特效

一、元素偏移量 offset 系列

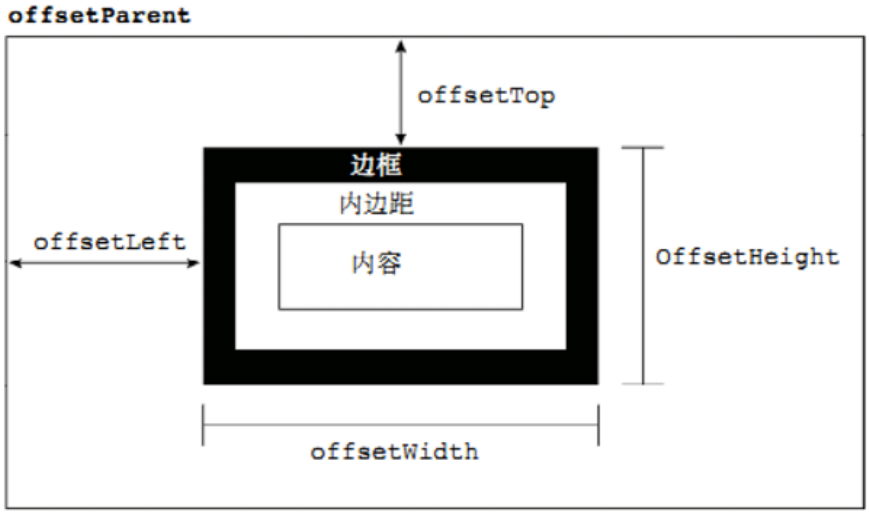
1、offset概述

offset 翻译过来就是偏移量， 我们使用 offset 系列相关属性可以动态的得到该元素的位置（偏移）、大小等。

- 获得元素距离带有定位父元素的位置
- 获得元素自身的大小（宽度高度）
- **注意： 返回的数值都不带单位**

offset系列常用属性

offset系列属性	作用
element.offsetParent	返回作为该元素带有定位的父级元素 如果父级都没有定位则返回body
element.offsetTop	返回元素相对带有定位父元素上方的偏移
element.offsetLeft	返回元素相对带有定位父元素左边框的偏移
element.offsetWidth	返回自身包括padding 、 边框、内容区的宽度，返回数值不带单位
element.offsetHeight	返回自身包括padding、边框、内容区的高度，返回数值不带单位



offset和style的区别

offset

- offset 可以得到任意样式表中的样式值
- offset 系列获得的数值是没有单位的
- offsetWidth 包含padding+border+width
- offsetWidth 等属性是只读属性，只能获取不能赋值
- 所以，我们想要获取元素大小位置，用offset更合适

style

- style 只能得到行内样式表中的样式值
- style.width 获得的是带有单位的字符串
- style.width 获得不包含padding和border 的值
- style.width 是可读写属性，可以获取也可以赋值
- 所以，我们想要给元素更改值，则需要用style改变

案例：获取鼠标在盒子内的坐标

> 1. 我们在盒子内点击，想要得到鼠标距离盒子左右的距离。
> 2. 首先得到鼠标在页面中的坐标（e.pageX, e.pageY）
> 3. 其次得到盒子在页面中的距离 (box.offsetLeft, box.offsetTop)
> 4. 用鼠标距离页面的坐标减去盒子在页面中的距离，得到 鼠标在盒子内的坐标
> 5. 如果想要移动一下鼠标，就要获取最新的坐标，使用鼠标移动事件 mousemove

```
var box = document.querySelector('.box');
box.addEventListener('mousemove', function(e) {
var x = e.pageX - this.offsetLeft;
var y = e.pageY - this.offsetTop;
this.innerHTML = 'x坐标是' + x + ' y坐标是' + y;
})
```

案例：模态框拖拽

> - 弹出框，我们也称为模态框。
> - 点击弹出层， 会弹出模态框， 并且显示灰色半透明的遮挡层。
> - 点击关闭按钮，可以关闭模态框，并且同时关闭灰色半透明遮挡层。
> - 鼠标放到模态框最上面一行，可以按住鼠标拖拽模态框在页面中移动。
> - 鼠标松开，可以停止拖动模态框移动。

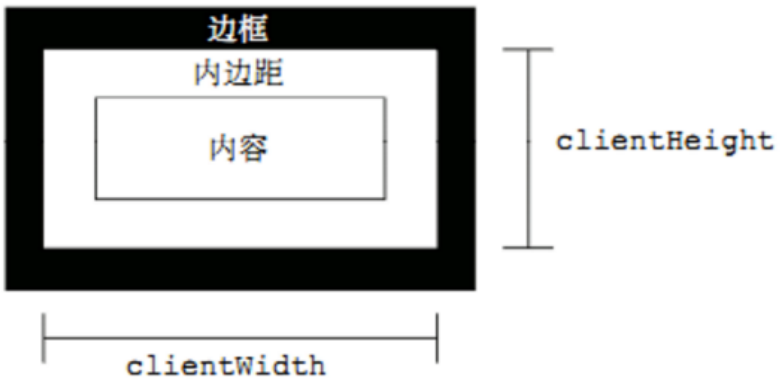
1. 点击弹出层， 模态框和遮挡层就会显示出来 display:block;
2. 点击关闭按钮，模态框和遮挡层就会隐藏起来 display:none;
3. 在页面中拖拽的原理： 鼠标按下并且移动， 之后松开鼠标
4. 触发事件是鼠标按下 mousedown， 鼠标移动mousemove 鼠标松开 mouseup
5. 拖拽过程: 鼠标移动过程中，获得最新的值赋值给模态框的left和top值， 这样模态框可以跟着鼠标走了
6. 鼠标按下触发的事件源是 最上面一行，就是 id 为 title
7. 鼠标的坐标 减去 鼠标在盒子内的坐标， 才是模态框真正的位置。
8. 鼠标按下，我们要得到鼠标在盒子的坐标。
9. 鼠标移动，就让模态框的坐标 设置为： 鼠标坐标 减去盒子坐标即可，注意移动事件写到按下事件里面。
10. 鼠标松开，就停止拖拽，就是可以让鼠标移动事件解除

二、元素可视区 client 系列

client 翻译过来就是客户端，我们使用 client 系列的相关属性来获取元素可视区的相关信息。通过 client 系列的相关属性可以动态的得到该元素的边框大小、元素大小等。

client系列属性

client系列属性	作用
element.clientTop	返回元素上边框的大小
element.clientLeft	返回元素左边框的大小
element.clientWidth	返回自身包括padding 、 内容区的宽度，不含边框，返回数值不带单位
element.clientHeight	返回自身包括padding 、 内容区的高度，不含边框，返回数值不带单位



立即执行函数：

> 立即执行函数 (function() {})() 或者 (function(){}())
>
> 主要作用： 创建一个独立的作用域。 避免了命名冲突问题

下面三种情况都会刷新页面都会触发 load 事件。

1. a标签的超链接
2. F5或者刷新按钮（强制刷新）
3. 前进后退按钮

但是 火狐中，有个特点，有个“往返缓存”，这个缓存中不仅保存着页面数据，还保存了DOM和JavaScript的状态；实际上是将整个页面都保存在了内存里。

所以此时后退按钮不能刷新页面。

此时可以**使用 pageshow事件来触发。，这个事件在页面显示时触发，无论页面是否来自缓存。在重新加载页面中，pageshow会在load事件触发后触发；根据事件对象中的persisted来判断是否是缓存中的页面触发的pageshow事件，注意这个事件给window添加。**

三、元素滚动 scroll 系列

scroll 翻译过来就是滚动的，我们使用 scroll 系列的相关属性可以动态的得到该元素的大小、滚动距离等。

scroll系列属性

scroll系列属性	作用
element.scrollTop	返回被卷去的上侧距离，返回数值不带单位
element.scrollLeft	返回被卷去的左侧距离，返回数值不带单位
element.scrollWidth	返回自身实际的宽度，不含边框，返回数值不带单位
element.scrollHeight	返回自身实际的高度，不含边框，返回数值不带单位

页面被卷去的头部

如果浏览器的高（或宽）度不足以显示整个页面时，会自动出现滚动条。当滚动条向下滚动时，页面上面被隐藏掉的高度，我们就称为页面被卷去的头部。滚动条在滚动时会触发 onscroll 事件。

> 页面被卷去的头部：可以通过window.pageYOffset 获得 如果是被卷去的左侧 window.pageXOffset
>
> 注意，**元素被卷去的头部是 element.scrollTop ,** 如果是**页面被卷去的头部 则是 window.pageYOffset**
>
> 其实这个值可以通过盒子的 offsetTop 可以得到，如果大于等于这个值，就可以让盒子固定定位了

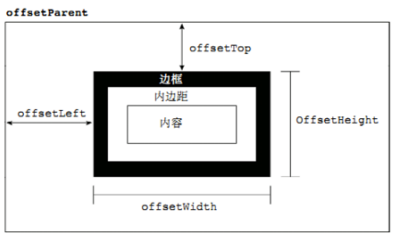
需要注意的是，页面被卷去的头部，有兼容性问题，因此被卷去的头部通常有如下几种写法：

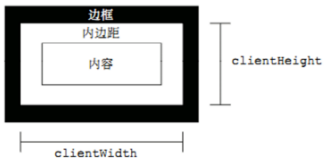
1. 声明了 DTD，使用 document.documentElement.scrollTop
2. 未声明 DTD，使用 document.body.scrollTop
3. **新方法 window.pageYOffset 和 window.pageXOffset，IE9 开始支持**

```
 function getScroll() {
    return {
      left: window.pageXOffset || document.documentElement.scrollLeft ||
document.body.scrollLeft||0,
      top: window.pageYOffset || document.documentElement.scrollTop || document.body.scrollTop || 0
    };
 }
使用的时候  getScroll().left
```

三大系列总结

三大系列大小对比	作用
element.offsetWidth	返回自身包括padding 、 边框、内容区的宽度，返回数值不带单位
element.clientWidth	返回自身包括padding 、 内容区的宽度，不含边框，返回数值不带单位
element.scrollWidth	返回自身实际的宽度，不含边框，返回数值不带单位





> 他们主要用法：
>
> 1. offset系列经常用于获得元素位置 offsetLeft offsetTop
> 2. client 经常用于获取元素大小 clientWidth clientHeight
> 3. scroll 经常用于获取滚动距离 scrollTop scrollLeft
>
> 注意页面滚动的距离通过 window.pageXOffset 获得

mouseenter和mouseover的区别

mouseenter鼠标事件

- 当鼠标移动到元素上时就会触发 mouseenter 事件
- 类似 mouseover，它们两者之间的差别是
- mouseover 鼠标经过自身盒子会触发，经过子盒子还会触发。 mouseenter 只会经过自身盒子触发

- 之所以这样，就是因为mouseenter不会冒泡
- 跟mouseenter搭配 鼠标离开 mouseleave 同样不会冒泡

四、动画函数封装

1、动画实现原理

核心原理：通过定时器 setInterval() 不断移动盒子位置。

> 实现步骤：！！！
>
> 1. 获得盒子当前位置
> 2. 让盒子在当前位置加上1个移动距离
> 3. 利用定时器不断重复这个操作
> 4. 加一个结束定时器的条件
> 5. 注意此元素需要添加定位，才能使用element.style.left

```
var div = document.querySelector('div');
var timer = setInterval(function () {
    if (div.offsetLeft >= 400) {
        // 停止动画实质是停止定时器
        clearInterval(timer);
    }
    div.style.left = div.offsetLeft + 1 + 'px';

}, 30);
```

2、动画函数简单封装

注意函数需要传递2个参数，**动画对象和移动到的距离。**

```
    function animate(obj, target) {
        var timer = setInterval(function () {
            if (obj.offsetLeft >= target) {
                // 停止动画实质是停止定时器
                clearInterval(timer);
            }
            obj.style.left = obj.offsetLeft + 1 + 'px';
            console.log(11);

        }, 30);
    }
    var div = document.querySelector('div');
    var span = document.querySelector('span');
    // 调用函数
    animate(div, 400);
    animate(span, 300);
</script>
```

3、动画函数给不同元素记录不同定时器

如果多个元素都使用这个动画函数，每次都要var 声明定时器。我们可以给不同的元素使用不同的定时器（自己专门用自己的定时器）。

核心原理：利用 JS 是一门动态语言，可以很方便的给当前对象添加属性。

```
        // 简单动画函数封装
        // 函数需要传递2个参数：obj：动画对象和target：移动到的距离


        // var obj = {};
        // obj.name = 'andy';
        function animate(obj, target) {
            // 当我们不断地点击按钮，元素的速度会越来越快，因为开启了太多的定时器
            // 解决方案：让元素只有一个定时器执行
            // 先清除之前的定时器，只保留当前的一个定时器执行
            clearInterval(obj.timer);
            obj.timer = setInterval(function () {
                if (obj.offsetLeft >= target) {
                    // 停止动画实质是停止定时器
                    clearInterval(obj.timer);
                }
                obj.style.left = obj.offsetLeft + 1 + 'px';
                console.log(11);


            }, 30);
        }
        var div = document.querySelector('div');
        var span = document.querySelector('span');
        var btn = document.querySelector('button');
        // 调用函数
        animate(div, 400);
        btn.addEventListener('click', function () {
            animate(span, 300);
        })
    </script>
```

4、缓动效果原理

缓动动画就是让元素运动速度有所变化，最常见的是让速度慢慢停下来

思路：

1. 让盒子每次移动的距离慢慢变小，速度就会慢慢落下来。
2. 核心算法：（目标值 - 现在的位置) / 10 做为每次移动的距离 步长
3. 停止的条件是： 让当前盒子位置等于目标位置就停止定时器
4. 注意步长值需要取整

```
// 匀速动画 就是 盒子是当前的位置 +  固定的值 10
// 缓动动画就是  盒子当前的位置 + 变化的值(目标值 - 现在的位置) / 10）
```

5、动画函数多个目标值之间移动

可以让动画函数从 800 移动到 500。

当我们点击按钮时候，判断步长是正值还是负值

1. 如果是正值，则步长 往大了取整
2. 如果是负值，则步长 向小了取整

6、动画函数添加回调函数

回调函数原理：函数可以作为一个参数。将这个函数作为参数传到另一个函数里面，当那个函数执行完之后，再执行传进去的这个函数，这个过程就叫做回调。

回调函数写的位置：定时器结束的位置。

7、动画函数封装到单独JS文件里面

因为以后经常使用这个动画函数，可以单独封装到一个JS文件里面，使用的时候引用这个JS文件即可。

1. 单独新建一个JS文件。
2. HTML文件引入 JS 文件。

案例：网页轮播图

> 轮播图也称为焦点图，是网页中比较常见的网页特效。
>
> 功能需求：
>
> 1. 鼠标经过轮播图模块，左右按钮显示，离开隐藏左右按钮。
> 2. 点击右侧按钮一次，图片往左播放一张，以此类推， 左侧按钮同理。
> 3. 图片播放的同时，下面小圆圈模块跟随一起变化。
> 4. 点击小圆圈，可以播放相应图片。
> 5. 鼠标不经过轮播图， 轮播图也会自动播放图片。
> 6. 鼠标经过，轮播图模块， 自动播放停止。

思路分析

> - 因为js较多，我们单独新建js文件夹，再新建js文件， 引入页面中。
> - 此时需要添加 load 事件。
> - 鼠标经过轮播图模块，左右按钮显示，离开隐藏左右按钮。
> - 显示隐藏 display 按钮。

动态生成小圆圈

核心思路：小圆圈的个数要跟图片张数一致

- 所以首先先得到ul里面图片的张数（图片放入li里面，所以就是li的个数）
- 利用循环动态生成小圆圈（这个小圆圈要放入ol里面）
- 创建节点 createElement('li')
- 插入节点 ol. appendChild(li)
- 第一个小圆圈需要添加 current 类

小圆圈的排他思想

- 点击当前小圆圈，就添加current类
- 其余的小圆圈就移除这个current类

注意： 我们在刚才生成小圆圈的同时，就可以直接绑定这个点击事件了。

点击小圆圈滚动图片

此时用到animate动画函数，将js文件引入（注意，因为index.js 依赖 animate.js 所以，animate.js 要写到 index.js 上面）

使用动画函数的前提，该元素必须有定位

注意是ul 移动 而不是小li

滚动图片的核心算法： 点击某个小圆圈， 就让图片滚动小圆圈的索引号乘以图片的宽度做为ul移动距离

此时需要知道小圆圈的索引号， 我们可以在生成小圆圈的时候，给它设置一个自定义属性，点击的时候获取这个自定义属性即可。

点击右侧按钮一次，就让图片滚动一张。

声明一个变量num， 点击一次，自增1， 让这个变量乘以图片宽度，就是 ul 的滚动距离。

图片无缝滚动原理

1. 把ul 第一个li 复制一份，放到ul 的最后面
2. 当图片滚动到克隆的最后一张图片时， 让ul 快速的、不做动画的跳到最左侧： left 为0
3. 同时num 赋值为0，可以从新开始滚动图片了

克隆第一张图片

- 克隆ul 第一个li cloneNode() 加true 深克隆 复制里面的子节点 false 浅克隆
- 添加到 ul 最后面 appendChild

点击右侧按钮， 小圆圈跟随变化

最简单的做法是再声明一个变量circle，每次点击自增1，注意，左侧按钮也需要这个变量，因此要声明全局变量。

但是图片有5张，我们小圆圈只有4个少一个，必须加一个判断条件

如果circle == 4 就 从新复原为 0

自动播放功能

添加一个定时器

1. 自动播放轮播图，实际就类似于点击了右侧按钮
2. 此时我们使用手动调用右侧按钮点击事件 arrow_r.click()
3. 鼠标经过focus 就停止定时器
4. 鼠标离开focus 就开启定时器

节流阀

> 防止轮播图按钮连续点击造成播放过快。
>
> **节流阀目的：当上一个函数动画内容执行完毕，再去执行下一个函数动画，让事件无法连续触发。**
>
> 核心实现思路：利用回调函数，添加一个变量来控制，锁住函数和解锁函数。

> 开始设置一个变量 var flag = true;
>
> If(flag) {flag = false; do something} 关闭水龙头
>
> 利用回调函数 动画执行完毕， flag = true 打开水龙头

> 滚动窗口至文档中的特定位置。
>
> window.scroll(x, y)
>
> 注意，里面的x和y 不跟单位，直接写数字

移动端网页特效

一、触屏事件

1、触摸事件

移动端浏览器兼容性较好，我们不需要考虑以前 JS 的兼容性问题，可以放心的使用原生 JS 书写效果，但是移动端也有自己独特的地方。比如触屏事件 touch（也称触摸事件），Android 和 IOS 都有。

touch 对象代表一个触摸点。触摸点可能是一根手指，也可能是一根触摸笔。触屏事件可响应用户手指（或触控笔）对屏幕或者触控板操作。

常见的触屏事件如下：

触屏touch事件	说明
touchstart	手指触摸到一个 DOM 元素时触发
touchmove	手指在一个 DOM 元素上滑动时触发
touchend	手指从一个 DOM 元素上移开时触发

2、触摸事件对象（TouchEvent）

TouchEvent 是一类描述手指在触摸平面（触摸屏、触摸板等）的状态变化的事件。这类事件用于描述一个或多个触点，使开发者可以检测触点的移动，触点的增加和减少，等等

touchstart、touchmove、touchend 三个事件都会各自有事件对象。

触摸事件对象重点我们看三个常见对象列表：

触摸列表	说明
touches	正在触摸屏幕的所有手指的一个列表
targetTouches	正在触摸当前 DOM 元素上的手指的一个列表
changedTouches	手指状态发生了改变的列表， 从无到有，从有到无变化

3、移动端拖动元素

1. touchstart、touchmove、touchend 可以实现拖动元素
2. 但是拖动元素需要当前手指的坐标值 我们可以使用 targetTouches[0] 里面的pageX 和 pageY
3. 移动端拖动的原理： 手指移动中，计算出手指移动的距离。然后用盒子原来的位置 + 手指移动的距离
4. 手指移动的距离： 手指滑动中的位置 减去 手指刚开始触摸的位置

拖动元素三步曲：

- 触摸元素 touchstart： 获取手指初始坐标，同时获得盒子原来的位置
- 移动手指 touchmove： 计算手指的滑动距离，并且移动盒子
- 离开手指 touchend

注意： 手指移动也会触发滚动屏幕所以这里要阻止默认的屏幕滚动 e.preventDefault();

```
        var div = document.querySelector('div');
        // 获取手指初始坐标
        var startX = 0;
        var startY = 0;
        // 获得盒子原来的位置
        var x = 0;
        var y = 0;
        div.addEventListener('touchstart', function (e) {
            // 获得手指坐标
            startX = e.targetTouches[0].pageX;
            startY = e.targetTouches[0].pageY;
            // 获得盒子位置
            x = this.offsetLeft;
            y = this.offsetTop;
        })
        div.addEventListener('touchmove', function (e) {
            // 计算手指的移动距离：手指移动之后的坐标减去手指初始的坐标
            var moveX = e.targetTouches[0].pageX - startX;
            var moveY = e.targetTouches[0].pageY - startY;
            // 移动盒子 盒子原来的位置+手指移动的位置
            this.style.left = x + moveX + 'px';
            this.style.top = y + moveY + 'px';
            // 阻止默认的屏幕滚动 e.preventDefault();
            e.preventDefault();
        })
    </script>
```

二、移动端常见特效

案例：移动端轮播图

主要js代码：

```
1  window.addEventListener('load', function () {
2      // alert(1);
3      // 1. 获取元素
4      var focus = document.querySelector('.focus');
5      var ul = focus.children[0];
```

```
    // 获得focus 的宽度
    var w = focus.offsetWidth;
    var ol = focus.children[1];
    // 2. 利用定时器自动轮播图图片
    var index = 0;
    var timer = setInterval(function () {
        index++;
        var translatex = -index * w;
        // 过渡效果
        ul.style.transition = 'all .3s';
        ul.style.transform = 'translateX(' + translatex + 'px)';
    }, 2000);
    // 等着我们过渡完成之后，再去判断
    // 监听过渡完成的事件 transitionend
    ul.addEventListener('transitionend', function () {
        // 无缝滚动
        if (index >= 3) {
            index = 0;
            // console.log(index);
            // 去掉过渡效果 这样让我们的ul 快速的跳到目标位置
            ul.style.transition = 'none';
            // 利用最新的索引号乘以宽度 去滚动图片
            var translatex = -index * w;
            ul.style.transform = 'translateX(' + translatex + 'px)';
        } else if (index < 0) {
            index = 2;
            ul.style.transition = 'none';
            // 利用最新的索引号乘以宽度 去滚动图片
            var translatex = -index * w;
            ul.style.transform = 'translateX(' + translatex + 'px)';
        }
        // 3. 小圆点跟随变化
        // 把ol里面li带有current类名的选出来去掉类名 remove
        ol.querySelector('.current').classList.remove('current');
        // 让当前索引号 的小li 加上 current   add
        ol.children[index].classList.add('current');
    });

    // 4. 手指滑动轮播图
    // 触摸元素 touchstart：  获取手指初始坐标
    var startX = 0;
    var moveX = 0; // 后面我们会使用这个移动距离所以要定义一个全局变量
    var flag = false;
    ul.addEventListener('touchstart', function (e) {
        startX = e.targetTouches[0].pageX;
        // 手指触摸的时候就停止定时器
        clearInterval(timer);
    });
    // 移动手指 touchmove：  计算手指的滑动距离， 并且移动盒子
    ul.addEventListener('touchmove', function (e) {
        // 计算移动距离
        moveX = e.targetTouches[0].pageX - startX;
        // 移动盒子：   盒子原来的位置 + 手指移动的距离
        var translatex = -index * w + moveX;
        // 手指拖动的时候，不需要动画效果所以要取消过渡效果
        ul.style.transition = 'none';
        ul.style.transform = 'translateX(' + translatex + 'px)';
        flag = true; // 如果用户手指移动过我们再去判断否则不做判断效果
```

```
        e.preventDefault(); // 阻止滚动屏幕的行为
    });
    // 手指离开 根据移动距离去判断是回弹还是播放上一张下一张
    ul.addEventListener('touchend', function (e) {
        if (flag) {
            // (1) 如果移动距离大于50像素我们就播放上一张或者下一张
            if (Math.abs(moveX) > 50) {
                // 如果是右滑就是 播放上一张 moveX 是正值
                if (moveX > 0) {
                    index--;
                } else {
                    // 如果是左滑就是 播放下一张 moveX 是负值
                    index++;
                }
                var translatex = -index * w;
                ul.style.transition = 'all .3s';
                ul.style.transform = 'translateX(' + translatex + 'px)';
            } else {
                // (2) 如果移动距离小于50像素我们就回弹
                var translatex = -index * w;
                ul.style.transition = 'all .1s';
                ul.style.transform = 'translateX(' + translatex + 'px)';
            }
        }
        // 手指离开的时候就重新开启定时器
        clearInterval(timer);
        timer = setInterval(function () {
            index++;
            var translatex = -index * w;
            ul.style.transition = 'all .3s';
            ul.style.transform = 'translateX(' + translatex + 'px)';
        }, 2000);
    });


    // 返回顶部模块制作
    var goBack = document.querySelector('.goBack');
    var nav = document.querySelector('nav');
    window.addEventListener('scroll', function () {
        if (window.pageYOffset >= nav.offsetTop) {
            goBack.style.display = 'block';
        } else {
            goBack.style.display = 'none';
        }
    });
    goBack.addEventListener('click', function () {
        window.scroll(0, 0);
    })
})
```

三、移动端常用开发插件

JS 插件是 js 文件，它遵循一定规范编写，方便程序展示效果，拥有特定功能且方便调用。如轮播图和瀑布流插件。

特点：它一般是为了解决某个问题而专门存在，其功能单一，并且比较小。比如移动端常见插件：iScroll、Swiper、SuperSlider 。

1、插件的使用

1. 引入 js 插件文件。
2. 按照规定语法使用。

2、Swiper 插件的使用

中文官网地址： https://www.swiper.com.cn/

superslide： http://www.superslide2.com/

iscroll： https://github.com/cubiq/iscroll

3、click 延时解决方案

移动端 click 事件会有 300ms 的延时，原因是移动端屏幕双击会缩放(double tap to zoom) 页面。

解决方案：

1. 禁用缩放。 浏览器禁用默认的双击缩放行为并且去掉 300ms 的点击延迟。

```
<meta name="viewport" content="user-scalable=no">
```

2. 使用插件。 fastclick 插件解决 300ms 延迟。

```
document.addEventListener('DOMContentLoaded',function () {
    /*等页面文档加载完成 不需要等所有的资源*/
    FastClick.attach(document.body);
});
```

移动端视频插件 zy.media.js

H5 给我们提供了 video 标签，但是浏览器的支持情况不同。在移动端我们可以使用插件方式来制作。

四、移动端常用开发框架

框架，顾名思义就是一套架构，它会基于自身的特点向用户提供一套较为完整的解决方案。框架的控制权在框架本身，使用者要按照框架所规定的某种规范进行开发。

前端常用的框架有 Bootstrap、Vue、Angular 等。

1、Bootstrap

Bootstrap 是一个简洁、直观、强悍的前端开发框架，它让 web 开发更迅速、简单。

2、MUI 原生UI前端框架

MUI 是一个专门用于做手机 APP 的前端框架。

MUI 的 UI 设计理念是：以 IOS 为基础，补充 Android 平台特有的控件。因此 MUI 封装的控件，UI 上更符合app 的体验。

MUI 中文官网地址：http://dev.dcloud.net.cn/mui/

本地存储

一、window.sessionStorage

本地存储特性

1、数据存储在用户浏览器中

2、设置、读取方便、甚至页面刷新不丢失数据

3、容量较大，sessionStorage约5M、localStorage约20M

4、只能存储字符串，可以将对象JSON.stringify() 编码后存储

> 1. 生命周期为关闭浏览器窗口
> 2. 在同一个窗口(页面)下数据可以共享
> 3. 以键值对的形式存储使用

存储数据

```
sessionStorage.setItem(key, value)
```

获取数据

```
sessionStorage.getItem(key)
```

删除数据

```
sessionStorage.removeItem(key)
```

删除所有数据

```
sessionStorage.clear()
```

二、window.localStorage

> 1. 声明周期永久生效，除非手动删除 否则关闭页面也会存在
> 2. 可以多窗口（页面）共享（同一浏览器可以共享）
> 3. 以键值对的形式存储使用

存储数据

```
localStorage.setItem(key, value)
```

获取数据

```
localStorage.getItem(key)
```

删除数据

```
localStorage.removeItem(key)
```

删除所有数据

```
localStorage.clear()
```