1.元素偏移量offset系列

1.1 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

注意:

1.如果没有父级或父级没有定位，则以body为主

2.offsetWidth/Height可以得到元素的大小宽度和高度是包含padding + border + width

3.

```
//3.返回带有定位的父亲否则返回的是body
console.log(son.offsetParent); // 返回带有定位的父亲,否则返回的是body
console.log(son.parentNode); // 返回父亲是最近一级的父亲,亲爸爸不管父亲有没有定位
```

4.offsetTop\offsetLeft只有获取距左和上的偏移，没有右和下

1.2 offset与style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

案例分析:

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(boxoffsetLeft, box offsetTop)

④核心思想：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 div = document.querySelector('div');
        div.addEventListener('mousemove', function (e) {
            var x = e.pageX - this.offsetLeft;//鼠标x的坐标-盒子距离左页面距离
            var y = e.pageY - this.offsetTop;//鼠标y点坐标-盒子距离上页面距离
            div.innerHTML = 'x坐标：' + x + 'y坐标：' + y;
        })
```

案例:模态框拖拽

弹出框,我们也称为模态框。

案例要求:

1.点击弹出层，会弹出模态框,并且显示灰色半透明的遮挡层。

2.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。

3.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。

4.鼠标松开,可以停止拖动模态框移动。

案例分析:

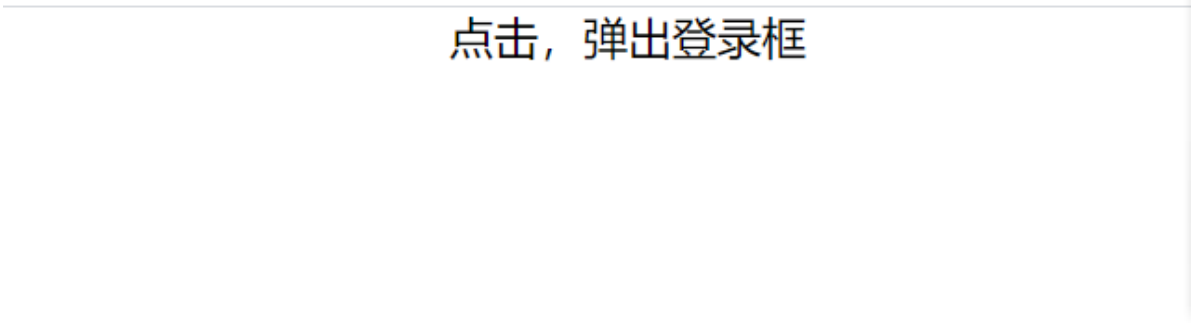
①点击弹出层，模态框和遮挡层就会显示出来display:block;
②点击关闭按钮，模态框和遮挡层就会隐藏起来display:none;
③在页面中拖拽的原理:鼠标按下并且移动，之后松开鼠标
④触发事件是鼠标按下mousedown,鼠标移动mousemove 鼠标松开mouseup
⑤拖拽过程:鼠标移动过程中，获得最新的值赋值给模态框的left和top值，这样模态框可以跟着鼠标走了
⑥鼠标按下触发的事件源是最上面一行，就是id为title
⑦鼠标的坐标减去鼠标在盒子内的坐标，才是模态框真正的位置。
⑧鼠标按下，我们要得到鼠标在盒子的坐标。
⑨鼠标移动，就让模态框的坐标设置为:鼠标坐标减去盒子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

```
    var link = document.querySelector('.login-header');
        var closeBtn = document.querySelector('.close');
        var mask = document.querySelector('.bg');
        var login = document.querySelector('.login');
        var title = document.querySelector('.login-title');
        //1.点击弹出层，弹出登录和遮罩
        link.addEventListener('click', function () {
            mask.style.display = 'block';
            login.style.display = 'block';
        })
        //2.关闭 隐藏登录和遮罩
        closeBtn.addEventListener('click', function () {
            mask.style.display = 'none';
            login.style.display = 'none';
        })
        //3.开始拖拽
        //1)鼠标按下，鼠标按下时就获得鼠标在盒子内相应坐标
        title.addEventListener('mousedown', function (e) {
            var x = e.pageX - login.offsetLeft;
            var y = e.pageY - login.offsetTop;
            // console.log(e.pageX);
            // console.log(login.offsetLeft);
            //2)鼠标移动，通过鼠标在盒子内的坐标不变，改变盒子的left、right来实现移动
            document.addEventListener('mousemove', move)
            function move(e) {
                login.style.left = e.pageX - x + 'px';//别忘记单位
                login.style.top = e.pageY - y + 'px';
            }
            //3）鼠标弹起，移除事件，由于需要函数名，所以将移动的函数外写
            document.addEventListener('mouseup', function () {
                document.removeEventListener('mousemove', move);
            })
        })
```

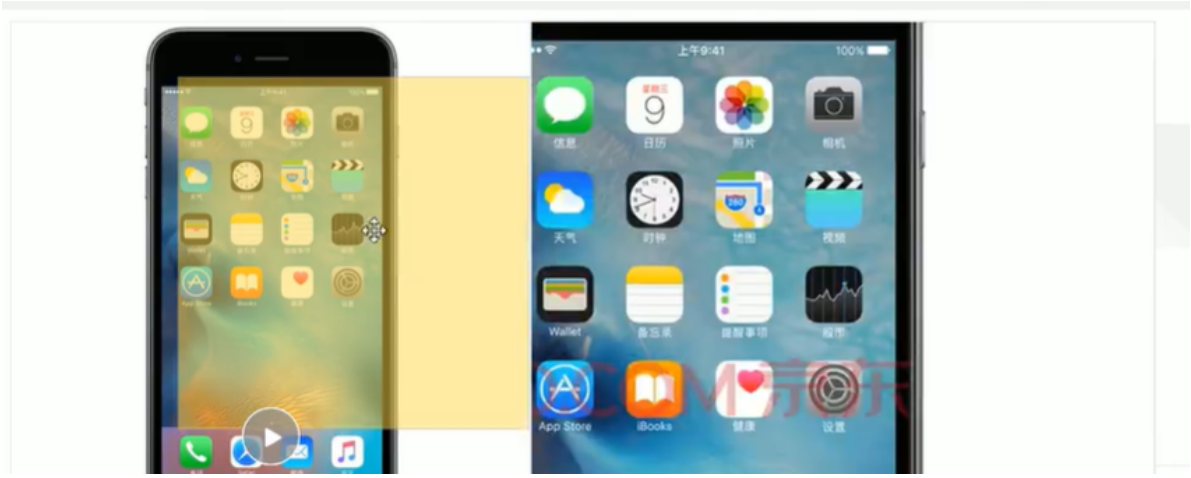
点击，弹出登录和遮罩

可进行移动

点击关闭



案例：仿京东放大镜效果



模块划分：

preview_img盒子

里面放手机图片盒子box，黄色盒子mask（红色边框），存放放大图片的盒子big

mask黄色遮罩使用绝对定位放入

big使用绝对定位放在手机盒子右侧及详情上放，需要**层级高**于底下的详情

需求分析

①整个案例可以分为三个功能模块

②鼠标经过小图片盒子，黄色的遮挡层和大图片盒子显示,离开隐藏2个盒子功能

③黄色的遮挡层跟随鼠标功能。

④移动黄色遮挡层,大图片跟随移动功能。

具体分析

显示与隐藏功能

①鼠标经过小图片盒子，黄色的遮挡层和大图片盒子显示，离开隐藏2个盒子功能
②就是显示与隐藏

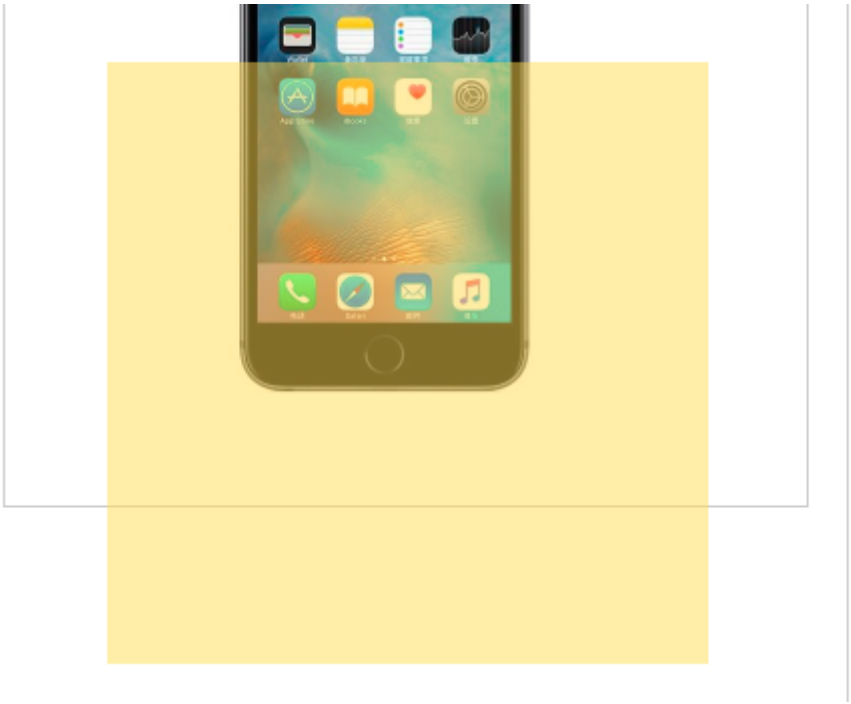
遮罩层跟随鼠标功能

①把鼠标坐标给遮挡层不合适。因为遮挡坐标以父盒子为准。

②首先是获得**鼠标在盒子的坐标。**

③之后**把数值给遮挡层做为left和top值，同时各减去遮挡层的宽度/高度的一半，使鼠标在遮罩层中心。**

④此时用到鼠标移动事件,但是还是在小图片盒子内移动。



注意遮挡层超出盒子问题，解决方法：

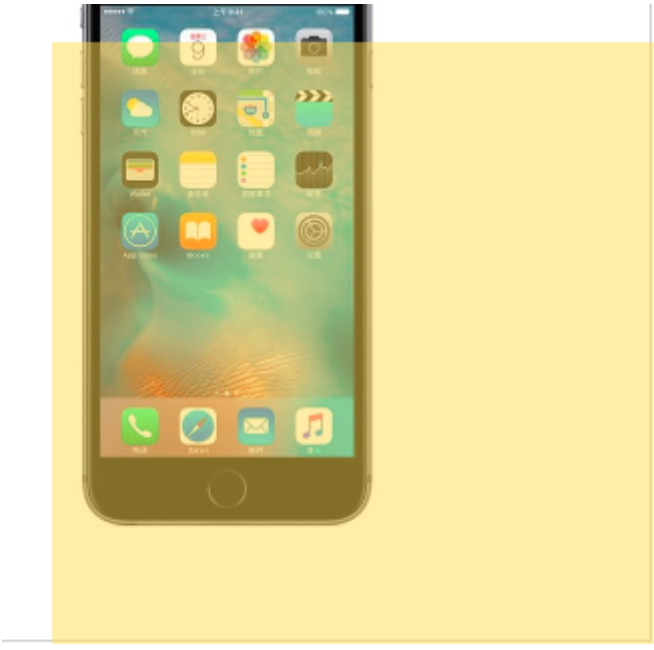
⑦遮挡层不能超出小图片盒子范围。

⑧如果小于零,就把坐标设置为0（左右、上下判断）

⑨如果大于遮挡层最大的移动距离,就把坐标设置为最大的移动距离

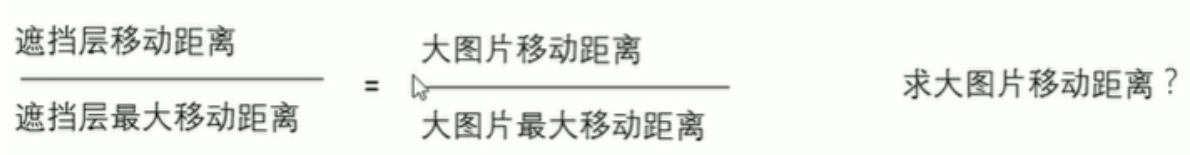
遮挡层的最大移动距离:小图片盒子宽度减去遮挡层盒子宽度

实现后：



移动黄色遮挡层,大图片跟随移动功能。

求大图片的移动距离公式



大图片移动距离=遮挡层移动距离*大图片最大移动距离/遮挡层最大移动距离

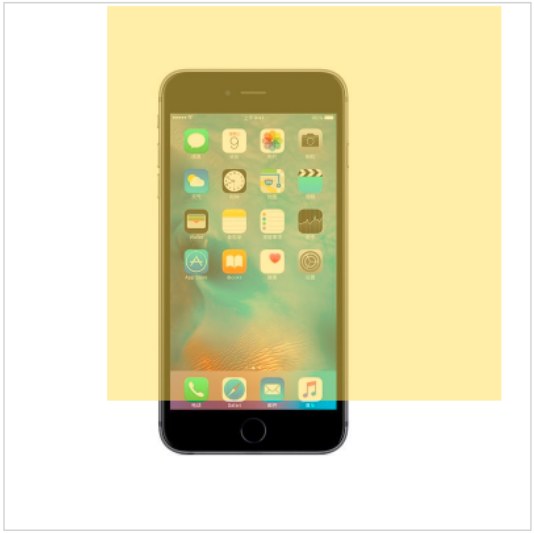
而不是遮挡层移动距离*大图片宽度/小图片宽度，如最大遮挡层移动距离是100px，那么x800/400后是200px，而大图片盒子是500px，只是移动200px无法达到盒子最右端。因为大图片与小图片之间有比例，两个相应的盒子大小之间也有比例，这两个比例并不相同。

```
var box = document.querySelector('.preview_img');
        var mask = document.querySelector('.mask');
        var big = document.querySelector('.big');
        var b = document.querySelector('.b');
        //1.鼠标经过显示mask和big,鼠标离开时隐藏
        box.addEventListener('mouseover', function () {
            mask.style.display = 'block';
            big.style.display = 'block';
        })
        box.addEventListener('mouseout', function () {
            mask.style.display = 'none';
            big.style.display = 'none';
        })
        //2.鼠标移动黄色遮罩层跟随移动
        box.addEventListener('mousemove', function (e) {
            //1)先计算鼠标在盒子内的坐标
            var x = e.pageX - this.offsetLeft;//此时要注意box的父级有没有定位,若有定位
offsetLeft就为与父级间的距离，而不是页面
            var y = e.pageY - this.offsetTop;
            //鼠标在盒子内的坐标即遮罩层的left top
            //2)为了让鼠标在遮罩层中间显示，需要再各自减去遮罩层的宽度、高度的一半
            //可通过offsetWidth/Height/2实现，而不是定死输入数值
            var maskX = x - mask.offsetWidth / 2;
            var maskY = y - mask.offsetHeight / 2;
            //3)通过限制遮罩层的left top的大小来限制遮罩层移动范围，防止移出图片盒子外面
            //left 遮罩层在最左边等于0，最右边等于图片盒子宽度减遮罩层宽度,top同理
            var smallMax = box.offsetWidth - mask.offsetWidth;//遮挡层最大移动距离,
由于是正方形，左右和上下最大移动距离一样
            var bigMax = b.offsetWidth - big.offsetWidth;//大图片最大移动距离
            if (maskX <= 0) {
                maskX = 0;
            } else if (maskX >= smallMax) {
                maskX = smallMax;
            }
            if (maskY <= 0) {
                maskY = 0;
            } else if (maskY >= smallMax) {
                maskY = smallMax;
            }
            mask.style.left = maskX + 'px';//注意单位
            mask.style.top = maskY + 'px';
            //3.移动黄色遮挡层,大图片跟随移动功能
            //大图片移动距离=遮挡层移动距离*大图片最大移动距离/遮挡层最大移动距离
            //注意！！遮挡层随鼠标右走，大图片方向相反是左走。两者移动方向正好相反
            b.style.left = -maskX * bigMax / smallMax + 'px';
```

```
            b.style.top = -maskY * bigMax / smallMax + 'px';
        })
```

注意：

遮罩层、大图片都要随着鼠标走，通过left、top赋值实现，所以必须都是绝对定位



2.元素可视区client系列

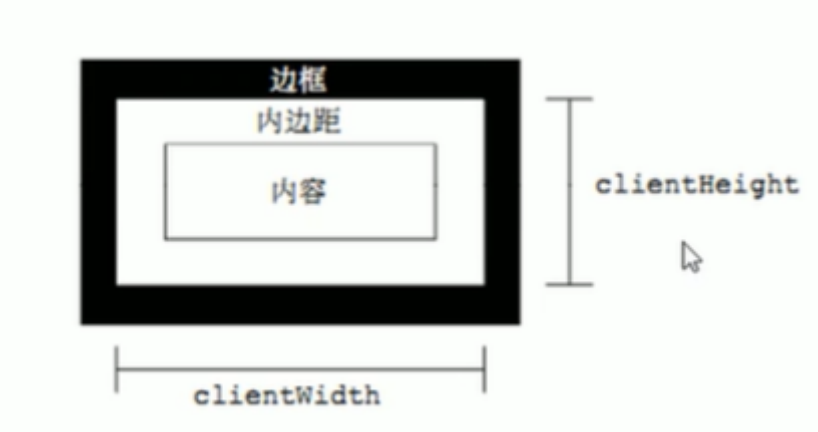
client翻译过来就是客户端,我们使用client系列的相关属性来获取元素可视区的相关信息。通过client系列
的相关属性可以动态的得到该元素的边框大小、元素大小等。

client系列属性	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，含边框，返回数值不带单位

注意：

1.offsetWidth返回包含padding+width+border；clientWidth返回包括padding+width，不包含边框。返回数值都不带单位

2.由于不包含边框，因此有两个属性专门返回上和左边框大小



案例：淘宝flexibleJS源码分析

立即执行函数

立即执行函数：不需要调用，立马能够执行自己的函数。

写法：

(function(){})()或者(function(){}())

第一种写法：第一个小括号将匿名函数包起来(若没有会报错)，第二个小括号可看作函数调用。第二种写法同理。

主要作用:创建一个独立的作用域。避免了命名冲突问题

注意：

1.多个立即执行函数之间用分号隔开

2.函数里也可以添加参数

3.也可以给匿名函数起名字

示例：

```
//1）(function(){})() 第二个小括号可看作调用函数 可以起名
        (function test() {
            console.log('1');
        })();
        //2）(function(){}()) 也可以传递参数
        (function (a, b) {
            console.log(a + b);
        }(1, 2))
```

```
1
3
>
```

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"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。

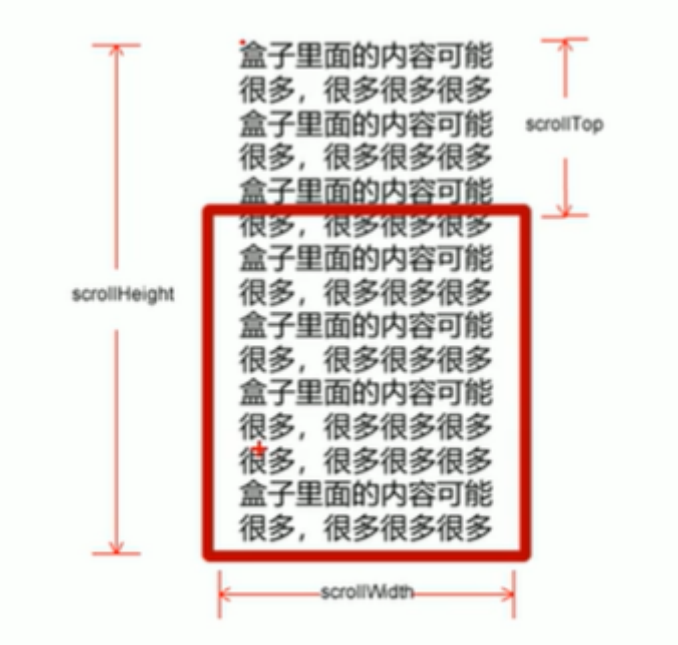
此时可以**使用pageshow事件——重新加载页面 来触发**, 这个事件在页面显示时触发,无论页面是否来自缓存，在重新加载页面中, pageshow会在load事件触发后触发;根据事件对象中的persisted来判断是否是缓存中的页面触发的pageshow事件,**注意这个事件给window添加。**

3.元素滚动scroll 系列

3.1元素scroll系列属性

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	作用
element.scrollTop	返回被卷去的上侧距离，返回数值不带单位
element.scrollLleft	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



红色边框为我们定义的盒子。

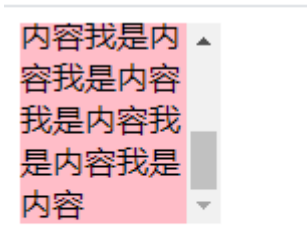
3.2页面被卷去的头部

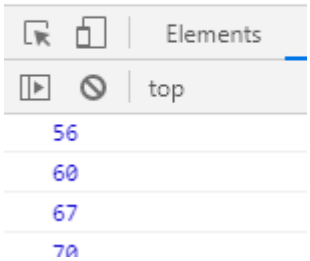
如果浏览器的高(或宽)度不足以显示整个页面时,会自动出现滚动条。当滚动条向下滚动时,页面上面被隐藏
掉的高度,我们就称为页面被卷去的头部，看上图scrollTop。**滚动条在滚动时会触发onscrol事件。**

示例:

```
div {
            width: 100px;
            height: 100px;
            background-color: pink;
            /* 添加滚动条 */
            overflow: auto;
        }

var div = document.querySelector('div');
        //scroll滚动事件，滚动条发生变化时就触发事件
        div.addEventListener('scroll', function () {
            console.log(div.scrollTop);
        })
```





3.3页面被卷去的头部兼容性解决方案

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. scrol1Top

3.新方法window . pageyof fset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
```

案例:仿淘宝固定右侧侧边栏

滚动到一定程序侧边栏显示"顶部"

案例需求分析:

1.**原先侧边栏是绝对定位**

2.当页面滚动到一定位置，走完头部区域，**侧边栏改为固定定位**

3.页面继续滚动,会让返回项部显示出来

案例分析:

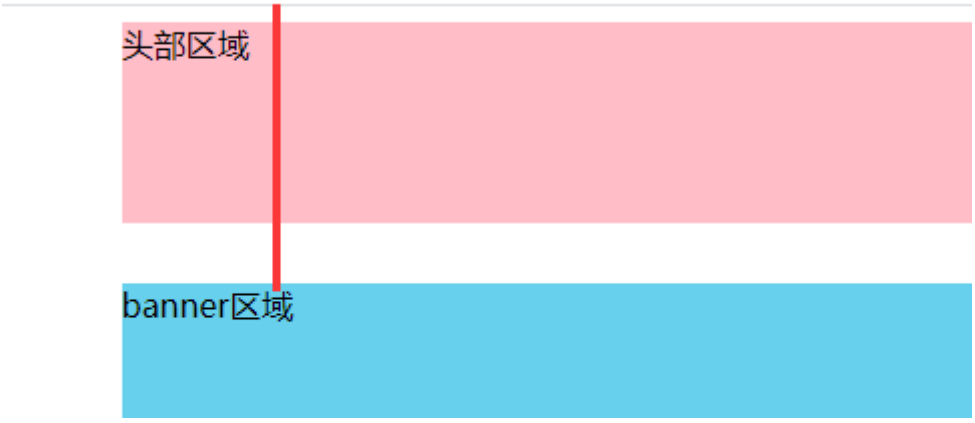
①需要用到页面滚动事件scroll因为是页面滚动,所以事件源是document

②滚动到某个位置,就是判断页面被卷去的上部值。

③**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**

④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**

问题1：头部滚动完后将侧边栏绝对定位改为固定定位，那头部该滚动多少距离呢即红线所指的距离



解决方法:

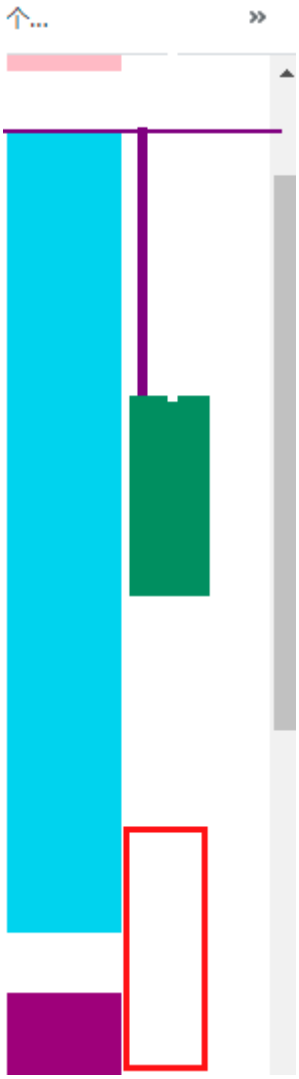
1）可通过人工滚动输出获得，边滚动变输出页面被卷去的上部值，当头部被卷去时输出的即我们需要的距离。缺点：需要人工判断，且数据定死，不灵活，不安全

```
console.log(window.pageYOffset);//183
```

2）**通过banner.offsetTop获得**，父元素没有定位，则获得的是与body的距离，在页面不滚动前，该值即为红线高度值，正是我们需要的

```
var bannerTop = banner.offsetTop;
```

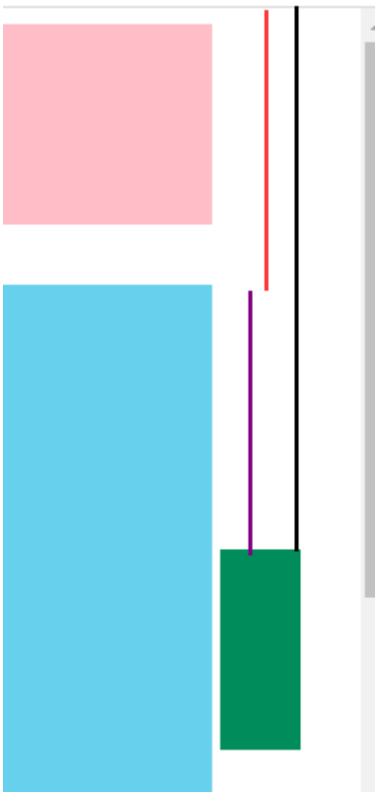
问题2：当侧边栏绝对定位变为固定定位的一瞬出现弹跳，如何解决。如图，下一秒盒子位置是红色框，直接从绿色框到红色框位置弹跳



解决方法：

让盒子改为固定定位同时，让侧边栏就停留在之前的位置，始终贴近蓝色banner区域，即紫线的距离

注意固定定位改为绝对定位同时，也要将此高度手动修改为原来的高度top-270px



该距离的求法：

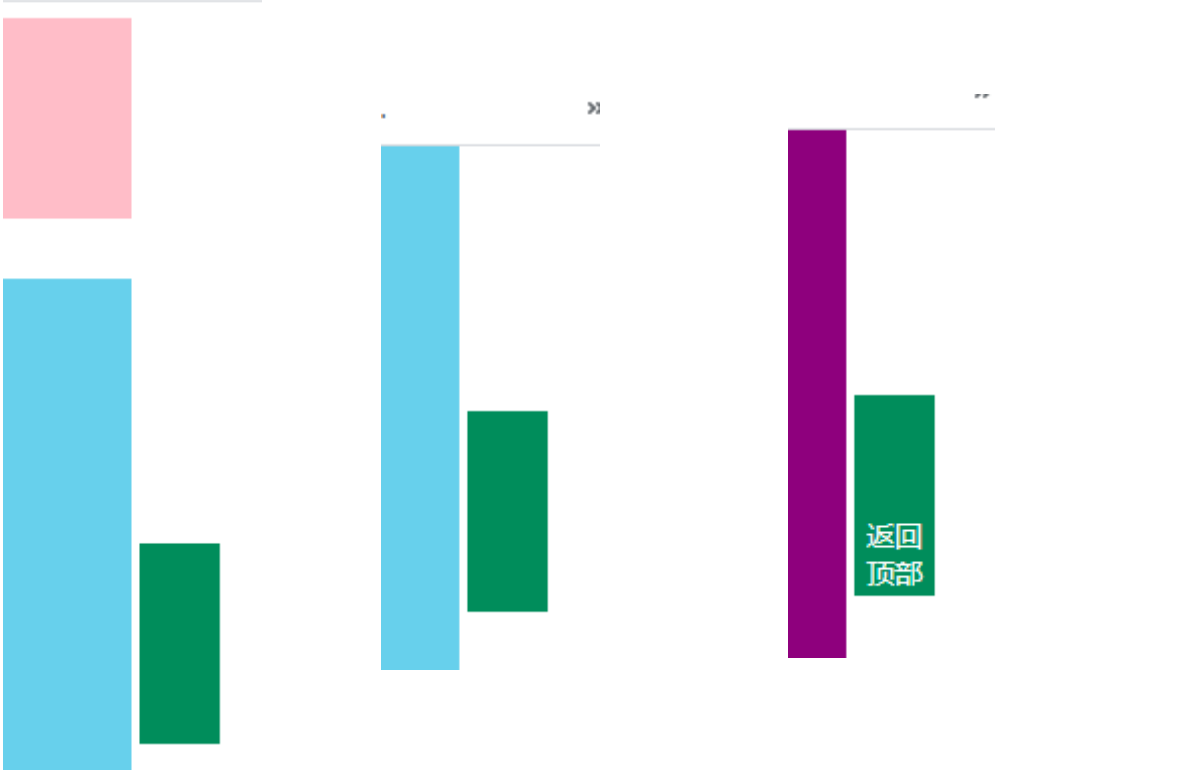
黑线距离（侧边栏盒子的offsetTop）减去红线距离（页面被卷去的头部高度）：即盒子的offsetTop减去bannerTop

```
var sliderbarTop = sliderbar.offsetTop - bannerTop;
```

完整js

```
 var sliderbar = document.querySelector('.slider-bar');
        var banner = document.querySelector('.banner');
        var main = document.querySelector('.main');
        var goTop = document.querySelector('.goTop');
        //offset属性在页面滚动前获取，否则数据会改变
        var bannerTop = banner.offsetTop;
        var sliderbarTop = sliderbar.offsetTop - bannerTop;
        var mainTop = main.offsetTop;
        // var top = sliderbar.offsetTop;
        //1.页面滚动事件
        document.addEventListener('scroll', function () {
            // console.log(window.pageYOffset);//借此知道滚动完top区域上面被卷高度是
183
            //2.当页面滚动完top区域时，将侧边栏绝对定位改为固定定位;返回时记得改为绝对定位
            //1)写死不安全，而banner.offsetTop页面尚未滚动时，该值等于被卷头部的高度
            //2)解决从侧边栏从绝对定位到固定定位盒子弹跳问题
            //原因：侧边栏盒子本身有一定高度
            if (window.pageYOffset >= bannerTop) {
                sliderbar.style.position = 'fixed';
                sliderbar.style.top = sliderbarTop + 'px';//记得加单位
            } else {
                sliderbar.style.position = 'absolute';
                //手动输入之前的top值
                sliderbar.style.top = '270px';
            }
```

```
            //3.页面滚动到main区域，显示返回顶部，否则隐藏
            if (window.pageYOffset >= mainTop) {
                goTop.style.display = 'block';
            } else {
                goTop.style.display = 'none';
            }
        })
```



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

相同点:返回数值都不带单位

不同点：offset宽度=width+padding+border;client宽度=width+padding，不含边框；scroll宽度=内容宽度（包括padding）不包括边框

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、scrolLeft

4.注意**页面滚动的距离通过 window. pageXoffset获得**

4.动画函数封装

4.1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 offset

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来停止动画

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
                clearInterval(timer);
            } else {
                div.style.left = div.offsetLeft + 1 + 'px';
            }
        }, 30)
```

从左往右缓慢移动，在left400px下停止

注意：**改变的是div.style.left，offsetLeft只读不可写**

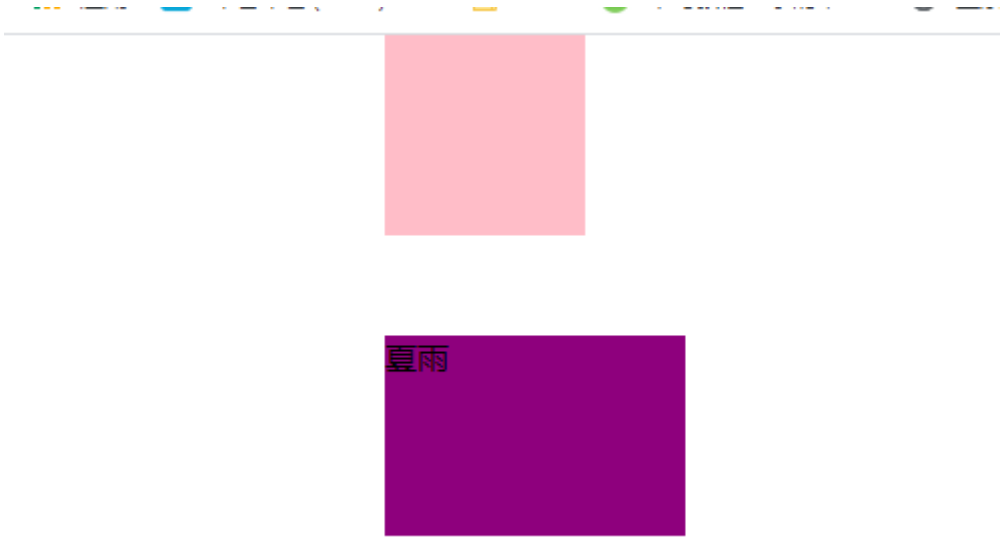
4.2动画函数简单封装

注意函数需要传递**2个参数,动画对象和移动到的距离。**

示例:

```
var div = document.querySelector('div');
        var span = document.querySelector('span');
        //简单动画函数封装，obj目标对象,target目标位置
        function animate(obj, target) {
            var timer = setInterval(function () {
                if (obj.offsetLeft >= target) {
                    //停止动画
                    clearInterval(timer);
                } else {
                    obj.style.left = obj.offsetLeft + 1 + 'px';
                }
            }, 30)
        }
        animate(div, 300);
```

```
        animate(span, 200)
```



4.3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。**

示例:

```
//obj是对象，给它添加timer属性，这样每个调用的对象都有自己专门的定时器
        function animate(obj, target) {
            //清除以前的定时器
            clearInterval(obj.timer);
            //只保留当前的一个定时器执行
            obj.timer = setInterval(function () {
                if (obj.offsetLeft >= target) {
                    clearInterval(obj.timer);
                } else {
                    obj.style.left = obj.offsetLeft + 1 + 'px';
                }
            }, 30)
        }
        var btn = document.querySelector('button');
        //点击按钮次数越多，盒子走的越来越快，原因是每点击一次就开启一个定时器，解决方法：让我们元素只有一个定时器执行
        btn.addEventListener('click', function () {
            animate(span, 400)
        })
        animate(div, 300);
```

注意：

1.obj是对象，给它添加timer属性，这样每个调用的对象都有自己专门的定时器

2.点击按钮次数越多，盒子走的越来越快，原因是每点击一次就开启一个定时器，**解决方法：让我们元素只有一个定时器执行。再设置当前的定时器前清除掉以前的定时器。**

4.4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

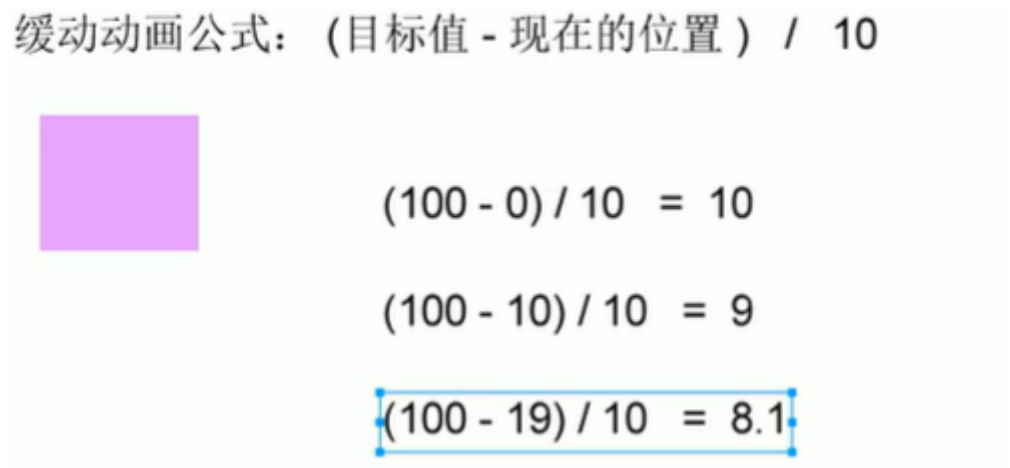
思路:

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

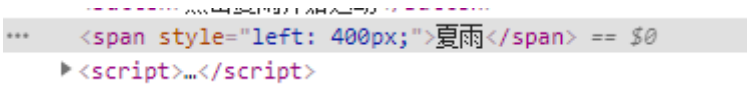


现在的位置坐标不断变大，目标值减去现在位置的值也越来越小，每次迈达步子也就越来越小。10可以改成其他数值。

示例：

```
function animate(obj, target) {
            //清除以前的定时器
            clearInterval(obj.timer);
            //只保留当前的一个定时器执行
```

```
            obj.timer = setInterval(function () {
                //步长一定要写在定时器里面，每定一次改变一次步长值
                //步长得是整数，因此我们需要去掉小数
                //步长为正时，向上取整；步长为负时，向下取整
                var step = (target - obj.offsetLeft) / 10;
                step = step > 0 ? Math.ceil(step) : Math.floor(step);
                //改成等号即可，现在的位置等于目标位置就停止了
                if (obj.offsetLeft == target) {
                    clearInterval(obj.timer);
                } else {
                    //步长公式：（目标值-现在都位置）/10
                    obj.style.left = obj.offsetLeft + step + 'px';
                }
            }, 15)
        }
```



效果：盒子由快到慢，减速停止

注意：

1.步长公式：（目标值-现在都位置）/10，因此可能出现小数，因此我们需要取整。**步长为正时，向上取整，往右走；步长为负时，向下取整，往左走。**这样让盒子运动到700再回来到400时，位置也是整数。

2.步长一定要写在定时器里面，每定一次改变一次步长值，让步长值越来越小

3.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)

4.**定时器时间一般写15**

4.5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。

示例：让盒子到达目标位置后变背景颜色

```
function animate(obj, target, callback) {
            //相当于callbac=function(){} 调用的时候callback()
            clearInterval(obj.timer);
            obj.timer = setInterval(function () {
                var step = (target - obj.offsetLeft) / 10;
                step = step > 0 ? Math.ceil(step) : Math.floor(step);
                if (obj.offsetLeft == target) {
                    clearInterval(obj.timer);
                    if (callback) {
                        //回调函数写到定时器结束里面
                        //让盒子到目标位置时再触发函数内容
                        callback();
                    }
                } else {
                    obj.style.left = obj.offsetLeft + step + 'px';
                }
            }, 15)
```

```
        }
btn.addEventListener('click', function () {
            //将函数作为参数传入
            animate(span, 400, function () {
                span.style.backgroundColor = 'red';
            })
        })
```

点击夏雨开始运动

夏雨

案例：引用动画函数

实现：鼠标经过盒子划出，鼠标离开盒子回收效果

将一个大盒子分为移动盒子和箭头盒子两部分，使用绝对定位

```
.sliderbar {
            position: absolute;
            top: 100px;
            right: 0;
            width: 40px;
            height: 40px;
            line-height: 40px;
            font-size: 20px;
            text-align: center;
            margin: 100px auto;
            background-color: purple;
            color: #fff;
            cursor: pointer;
        }

        .con {
            position: absolute;
```

```
            top: 0;
            left: 0;
            width: 200px;
            height: 40px;
            background-color: purple;
            z-index: -1;
        }
<div class="sliderbar">
        <span>←</span>
        <div class="con">问题反馈</div>
    </div>
```

```
var sliderbar = document.querySelector('.sliderbar');
        var con = document.querySelector('.con');
        var juli = con.clientWidth - sliderbar.clientWidth
        //鼠标经过，盒子左移
        sliderbar.addEventListener('mouseenter', function () {
            //animate(obj,target,callback),target=con的宽度-sliderbar的宽度
            //因为左移，所以需要加负号
            // 当我们动画执行完毕，就把←改为→
            animate(con, -juli, function () {
                sliderbar.children[0].innerHTML = '→'
            })
        })
        //鼠标离开，盒子回去
        sliderbar.addEventListener('mouseleave', function () {
            animate(con, 0, function () {
                sliderbar.children[0].innerHTML = '←'
            })//注意回去回到0位置，不是juli
        })
```

注意：

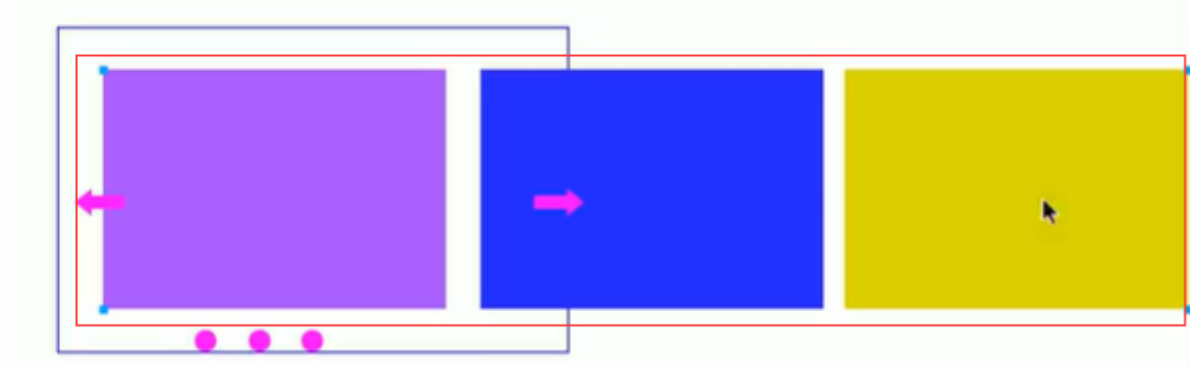
1.为了让文字显得居中对齐，需要采用和箭头盒子一样的颜色，否则效果不好看，如下图

2.鼠标离开，盒子回去，让盒子回到0位置，而不是200-40=160，target是目标位置，不是目标移动距离

3.鼠标经过，盒子左移，左移相对于原位置是负的，需要加负号

5.常见网页特效案例

5.1网页轮播图



模块划分：

1.蓝色盒子放轮播图

2.盒子内需要两个箭头，三个小圆圈，通过绝对定位实现

3.图片放在ul下的每个li内，从而添加浮动让li在一行上显示

4.由于ul宽度不够，不足以让li一行显示，**增大ul的宽度**，后期给蓝色盒子添加溢出隐藏即可

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。

根据功能案例分析

1.鼠标经过轮播图左右按钮显示，离开隐藏按钮。

案例分析

①因为js较多，我们单独新建j文件夹，再新建j文件，引入页面中。
②此时需要添加load事件。
③鼠标经过轮播图模块，左右按钮显示，离开隐藏左右按钮。

```
//1.鼠标经过显示箭头，鼠标离开隐藏箭头
    focus.addEventListener('mouseenter', function () {
        arrowl.style.display = 'block'
        arrowr.style.display = 'block'
    })
    focus.addEventListener('mouseleave', function () {
        arrowl.style.display = 'none'
        arrowr.style.display = 'none'
    })
```

①动态生成小圆圈

②核心思路:**小圆圈的个数要跟图片张数一致**
③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)
④利用循环动态生成小圆圈(这个小圆圈要放入ol里面)
⑤创建节点createElement('li'')
⑥插入节点ol. appendChild(li)
⑦第一个小圆圈需要添加current类

```
//2.动态生成小圆圈，有多少张图片就有多少个小圆圈
    //小圆圈个数取决于图片张数，即多少个li，可以通过ul的子元素节点个数获得
    var ul = document.querySelector('.pic');
    var circle = document.querySelector('.circle');
    for (var i = 0; i < ul.children.length; i++) {
        //创建元素
        var li = document.createElement('li')
        //添加元素
        circle.appendChild(li);
    }
    //给第一个孩子添加current类，默认选中第一个
    circle.children[0].className = 'current'
```

①小圆圈的排他思想，

②点击当前小圆圈，就添加current类
③其余的小圆圈就移除这个current类
④**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.**

①点击小圆圈滚动图片

②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)

③使用动画函数的前提，该元素必须有定位

④**注意是ul移动而不是小li**，ul移动了li也就跟随移动了，若移动li，那么后面的li也要跟着移动麻烦

⑤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**

⑥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。

注意：由于给ul添加了绝对定位，箭头和圆圈的绝对定位可能受影响被压住，需要提高他们的层级z-indent

②引用animate文件

```
    <!-- 引入动画函数js -->
    <script src="js/animate.js"></script>
    <!-- 引入首页的js -->
    <script src="js/index.js"></script>
```

③添加ul绝对定位

```
.focus .pic{
    position: absolute;
    top:0;
    left:0;
    width: 500%;
}
```

```
for (var i = 0; i < ul.children.length; i++) {
        //创建元素
        var li = document.createElement('li')
        //添加自定义属性来记录当前小圆圈的索引号
        li.setAttribute('index', i)
        //添加元素
        circle.appendChild(li);
        //3.小圆圈的排他思想，在生成小圆圈的同时就可以添加点击事件了
        li.addEventListener('click', function () {
            //干掉其他人
            for (var i = 0; i < circle.children.length; i++) {
                circle.children[i].className = ''
            }
            this.className = 'current'
            //4.点击小圆圈，图片移动，注意移动的是ul
            //ul的移动距离为li的索引号*图片的宽度，注意图片往左走，应添加负号
            //li的索引号可以在生成li的时候添加自定义属性，通过自定义属性获得索引号
            //点击了某个li，就拿到当前li的索引号
            var index = this.getAttribute('index')
            var imgWidth = focus.offsetWidth
            animate(ul, -index * imgWidth)
        })
```

①点右侧按钮一次，就让图片滚动一张。

②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。

③图片无缝滚动原理

④把ul第一个li复制一份，放到ul的最后面。注意这步通过克隆节点完成，不要手动复制，否则小圆圈个数会多一个

⑤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，回到第一张图片,然后num++滚到第二张图

⑥同时num赋值为0，可以重新开始滚动图片了

①克隆第一张图片

②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

③添加到ul最后面appendChild

```
//5.克隆第一张图片，并放入最后  注意在动态生成li之后克隆，否则影响小圆圈数量
    var first = ul.children[0].cloneNode(true);//深拷贝
    ul.appendChild(first)
    //6.点击右键按钮，滚动图片
    var num = 0
    arrowr.addEventListener('click', function () {
```

```
        //图片滚动到最后一张时，立马跳到第一张图
        if (num == ul.children.length - 1) {
            ul.style.left = 0
            num = 0
        }
        //跳到第一张图后立马滚动到第二张图，注意下面两句不能放入else里面
        //移动距离 num*图片宽度，注意右键图片左走，负值
        num++
        animate(ul, -num * imgWidth)
    })
```

①点击右侧按钮，小圆圈跟随变化。放入右侧箭头的点击事件内

②最简单的做法是再声明一个变量count，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。**

③注意圆圈以图片数量（不包括拷贝的）为一个循环，所以通过取模来决定当前的圆圈设置格式

```
        count++
        //干掉其他人
        for (var i = 0; i < circle.children.length; i++) {
            circle.children[i].className = ''
        }
        //只设置自己
        //这个自己取决于 count 取模 不包括克隆的图片数量，点击第一次显示第一张，第二次显示第二张，每图片数量一个循环
        circle.children[count % (ul.children.length - 1)].className = 'current'
```

解决小BUG

1.点击第三个圆圈显示第三张图片，此时再点击右侧按钮却并未显示的第四张图片。

解决方法：将index值赋值给num，让图片显示第四张。

2.图片虽然显示第四张，但圆圈显示第二个

解决方法：将index赋值给count，让圆圈显示第四个

index控制圆圈，num控制右侧按钮，count控制圆圈和按钮连接，index与num、count存在信息查，index改变时也需要改变num、count，告知他们目前信息

左侧按钮滚动图片

思路与右侧差不多，当图片到第一张时再点击按钮，里面跳到拷贝到那种图片，即最后一张图片，然后利用图片滚动到倒数第二张图，此时小圆圈需要到最后一个。

```
//7.点击左侧按钮，滚动图片
    arrowl.addEventListener('click', function () {
        //图片滚动到第一张时，立马跳到拷贝的那张，即最后一张;
        if (num == 0) {
            num = ul.children.length - 1
            ul.style.left = -num * imgWidth + 'px'//注意左走取负值，单位
        }
        //再跳到倒数第二张num--
        num--
        animate(ul, -num * imgWidth)
        //7.点击右侧按钮，小圆圈随着变化，通过再声明一个变量来控制小圆圈现在的样式
```

```
        count--
        //干掉其他人
        for (var i = 0; i < circle.children.length; i++) {
            circle.children[i].className = ''
        }
        //只设置自己
        //第一张就到倒数第二张,小圆圈就到最后一个
        if (count < 0) {
            count = circle.children.length - 1
        }
        circle.children[count].className = 'current'
    })
```

①自动播放功能

②添加一个定时器

③自动播放轮播图，实际就**类似于点击了右侧按钮**

④此时我们使用**手动调用右侧按钮点击事件arrow_ r.click()**

⑤鼠标经过focus就停止定时器

⑥鼠标离开focus就开启定时器

```
//9.自动播放
    var timer = setInterval(function () {
        //手动调用右侧按钮
        arrowr.click()
    }, 2000)

focus.addEventListener('mouseenter', function () {
        arrowl.style.display = 'block'
        arrowr.style.display = 'block'
        //暂停定时器
        clearInterval(timer)
        timer = null //最好再清空定时器变量
    })
    focus.addEventListener('mouseleave', function () {
        arrowl.style.display = 'none'
        arrowr.style.display = 'none'
        //开启定时器
        timer = setInterval(function () {
            //手动调用右侧按钮
            arrowr.click()
        }, 2000)
    })
```

5.2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

```
var flag = true
    arrowr.addEventListener('click', function () {
        if (flag) {
            flag = flase //关闭水龙头
            //图片滚动到最后一张时，立马跳到第一张图
            if (num == ul.children.length - 1) {
                ul.style.left = 0
                num = 0
            }
            //跳到第一张图后立马滚动到第二张图，注意下面两句不能放入else里面
            //移动距离 num*图片宽度，注意右键图片左走，负值
            num++
            animate(ul, -num * imgWidth, function () {
                flag = true //回调完成后再开启水龙头
```

强制一张图片滚动完才能再次点击按钮滚动图片

案例:筋头云案例

鼠标经过某个小li，筋斗云跟这到当前小li位置
鼠标离开这个小li，筋斗云复原为原来的位置
鼠标点击了某个小i,筋斗云就会留在点击这个小li的位置

知识点与逻辑中断应用补充

```
/* if (callback) {
                //回调函数写到定时器结束里面
                //让盒子到目标位置时再触发函数内容
                callback();
            } */
            //等价于
            callback && callback();
            //利用与逻辑的中断，如果有callback参数则再看callback()即调用函数；如果没有就直接结束
```