**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：



.father {

/\* position: relative; 有定位\*/

width: 200px;

height: 200px;

background-color: pink;

margin: 150px;

}

.son {

width: 100px;

height: 100px;

background-color: purple;

margin-left: 45px;

}

.w {

height: 200px;

background-color: skyblue;

margin: 0 auto 200px;

padding: 10px;

border: 15px solid red;

}

</style>

</head>

<body>

<div class="father">

<div class="son"></div>

</div>

<div class="w"></div>

<script>

// offset 系列

var father = document.querySelector('.father');

var son = document.querySelector('.son');

// 1.可以得到元素的偏移 位置 返回的不带单位的数值

console.log(father.offsetTop);

console.log(father.offsetLeft);

// 它以带有定位的父亲为准 如果么有父亲或者父亲没有定位 则以 body 为准

console.log(son.offsetLeft);

var w = document.querySelector('.w');

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

console.log(w.offsetWidth);

console.log(w.offsetHeight);

// 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

</script>

</body>

**1.2 offset 与 style 区别**

|  |  |
| --- | --- |
| **offset** | **style** |
| offset 可以得到任意样式表中的样式值 | style 只能得到行内样式表中的样式值 |
| offset 系列获得的数值是没有单位的 | style.width 获得的是带有单位的字符串 |
| offsetWidth 包含padding+border+width | style.width 获得不包含padding和border 的值 |
| offsetWidth 等属性是只读属性，只能获取不能赋值 | style.width 是可读写属性，可以获取也可以赋值 |
| 所以，我们想要获取元素大小位置，用offset更合适 | 所以，我们想要给元素更改值，则需要用style改变 |

<style>

.box {

width: 200px;

height: 200px;

background-color: pink;

padding: 10px;

}

</style>

<body>

<div class="box" style="width: 200px;"></div>//style 只能得到行内样式表中的样式值

<script>

// offset与style的区别

var box = document.querySelector('.box');

console.log(box.offsetWidth);//220

console.log(box.style.width);//200px

//offsetWidth 等属性是只读属性，只能获取不能赋值

// box.offsetWidth = '300px';

box.style.width = '300px';

</script>

</body>

**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) {

//e鼠标事件对象

// console.log(e.pageX);

// console.log(e.pageY);

// console.log(box.offsetLeft);

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：拖动模态框**

弹出框，我们也称为模态框。

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。

⑩ 鼠标松开，就停止拖拽，就是可以让鼠标移动事件解除

<script>

// 1. 获取元素

var login = document.querySelector('.login');

var mask = document.querySelector('.login-bg');

var link = document.querySelector('#link');

var closeBtn = document.querySelector('#closeBtn');

var title = document.querySelector('#title');

// 2. 点击弹出层这个链接 link 让mask 和login 显示出来

link.addEventListener('click', function() {

mask.style.display = 'block';

login.style.display = 'block';

})

// 3. 点击 closeBtn 就隐藏 mask 和 login

closeBtn.addEventListener('click', function() {

mask.style.display = 'none';

login.style.display = 'none';

})

// 4. 开始拖拽

// (1) 当我们鼠标按下， 就获得鼠标在盒子内的坐标，鼠标移动和弹起事件都写在鼠标按下事件内

title.addEventListener('mousedown', function(e) {

var x = e.pageX - login.offsetLeft;

var y = e.pageY - login.offsetTop;

// (2) 鼠标移动的时候，把鼠标在页面中的坐标，减去 鼠标在盒子内的坐标就是模态框的left和top值；事件源是document因为在页面任何一个位置都可以进行鼠标移动

document.addEventListener('mousemove', move)

//把函数单独写出来命名为move，利于mousemove、mouseup事件的传参

function move(e) {

login.style.left = e.pageX - x + 'px';

login.style.top = e.pageY - y + 'px';

}

// (3) 鼠标弹起，就让鼠标移动事件移除

document.addEventListener('mouseup', function() {

document.removeEventListener('mousemove', move);

})

})

</script>

**案例：仿京东放大镜**

**step one**

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

**step two**

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

**step three**

① 黄色的遮挡层跟随鼠标功能。

② 把鼠标坐标给遮挡层不合适。因为遮挡层坐标以父盒子为准。

③ 首先是获得鼠标在盒子的坐标。

④ 之后把数值给遮挡层做为left 和top值。

⑤ 此时用到鼠标移动事件，但是还是在小图片盒子内移动。

⑥ 但发现遮挡层位置不对即遮挡层的左上角跟随鼠标移动不是中间位置跟随鼠标移动，需要再减去盒子自身高度和宽度的一半。

⑦ 遮挡层不能超出小图片盒子范围。

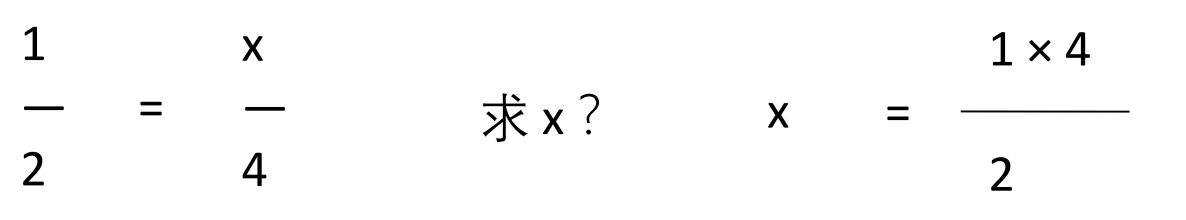
⑧ 如果小于零，就把坐标设置为0

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

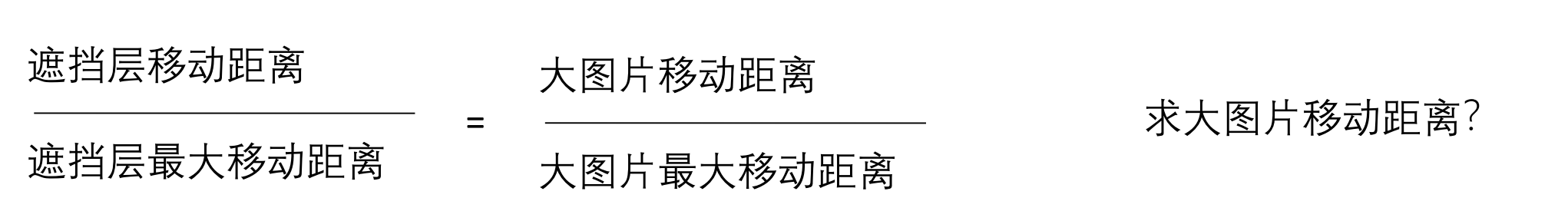
⑩ 遮挡层的最大移动距离： 小图片盒子宽度 减去 遮挡层盒子宽度

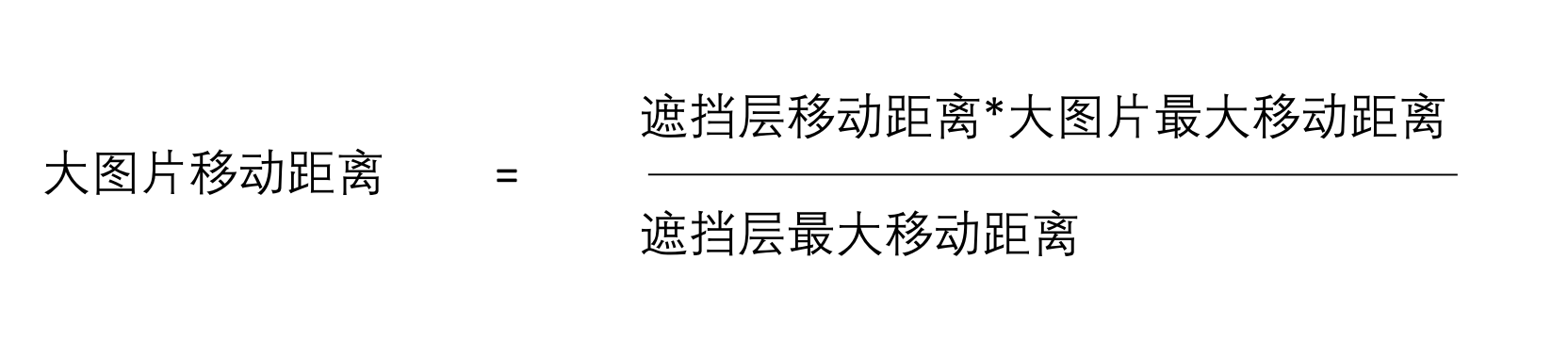
**step four**

**① 移动黄色遮挡层，大图片跟随移动功能。**



**② 求大图片的移动距离公式**



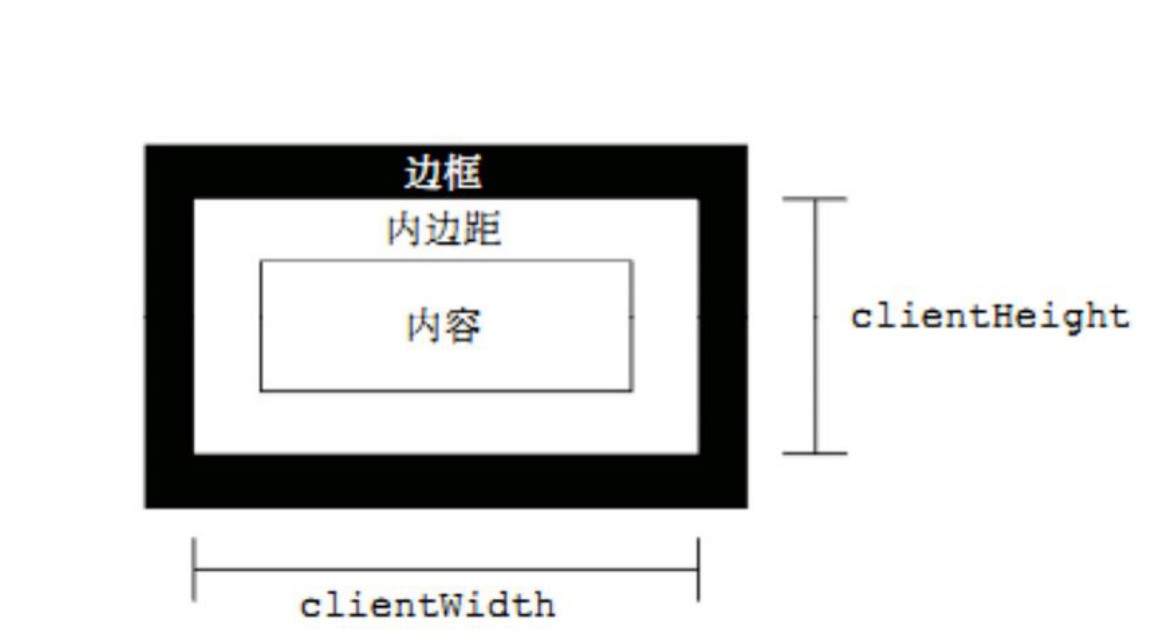


**2. 元素可视区 client 系列**

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。





**案例： 淘宝 flexible.js 源码分析**

知道代码实现是什么功能并会使用即可

立即执行函数 (function() {})() 或者 (function(){}())

主要作用：立即执行， 创建一个独立的作用域，里面所有的变量都是一个局部变量。 避免了命名冲突问题

(function(a,b){

console.log(a+b);

})(1,2)；//第二个小括号可以看做是调用函数，并且可以传递参数进来，如果有多个立即执行函数记得用；隔开，表示是两个独立的立即执行函数

(function(a,b){

console.log(a+b)

}(2,3))//5

(function sum(a,b){

console.log(a+b)

}(2,3))//也可以给立即执行函数命名

(function flexible(window, document) {

// 获取的html 的根元素相当于获取html文件

var docEl = document.documentElement

// dpr 物理像素比 获取查看当前浏览器物理像素比

var dpr = window.devicePixelRatio || 1

// adjust body font size 设置我们body 的字体大小

function setBodyFontSize() {

// 如果页面中有body 这个元素 就设置body的字体大小

if (document.body) {

document.body.style.fontSize = (12 \* dpr) + 'px'

} else {

// 如果页面中没有body 这个元素，例如html文件里在<head>里引入js文件，代码从上往下执行，这时body元素还没有拿到，则等着 我们页面主要的DOM元素加载完毕再去设置body

// 的字体大小

document.addEventListener('DOMContentLoaded', setBodyFontSize)

}

}

setBodyFontSize();

// set 1rem = viewWidth / 10 设置我们html 元素的文字大小

function setRemUnit() {

var rem = docEl.clientWidth / 10

docEl.style.fontSize = rem + 'px'

}

setRemUnit()

// reset rem unit on page resize 当我们页面尺寸大小发生变化的时候，要重新设置下rem 的大小

window.addEventListener('resize', setRemUnit)

// pageshow 是我们重新加载页面触发的事件

window.addEventListener('pageshow', function(e) {

// e.persisted 返回的是true 就是说如果这个页面是从缓存取过来的页面，也需要从新计算一下rem 的大小，从而解决火狐浏览器的问题

if (e.persisted) {

setRemUnit()

}

})

// detect 0.5px supports 有些移动端的浏览器不支持0.5像素的写法以下代码就是解决方案 知道这段代码作用即可

if (dpr >= 2) {

var fakeBody = document.createElement('body')

var testElement = document.createElement('div')

testElement.style.border = '.5px solid transparent'

fakeBody.appendChild(testElement)

docEl.appendChild(fakeBody)

if (testElement.offsetHeight === 1) {

docEl.classList.add('hairlines')

}

docEl.removeChild(fakeBody)

}

}(window, document))//把window，document这两个对象传参传给立即执行函数，使用第二种方式进行声明

为什么用pageshow事件，不用load事件？

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。

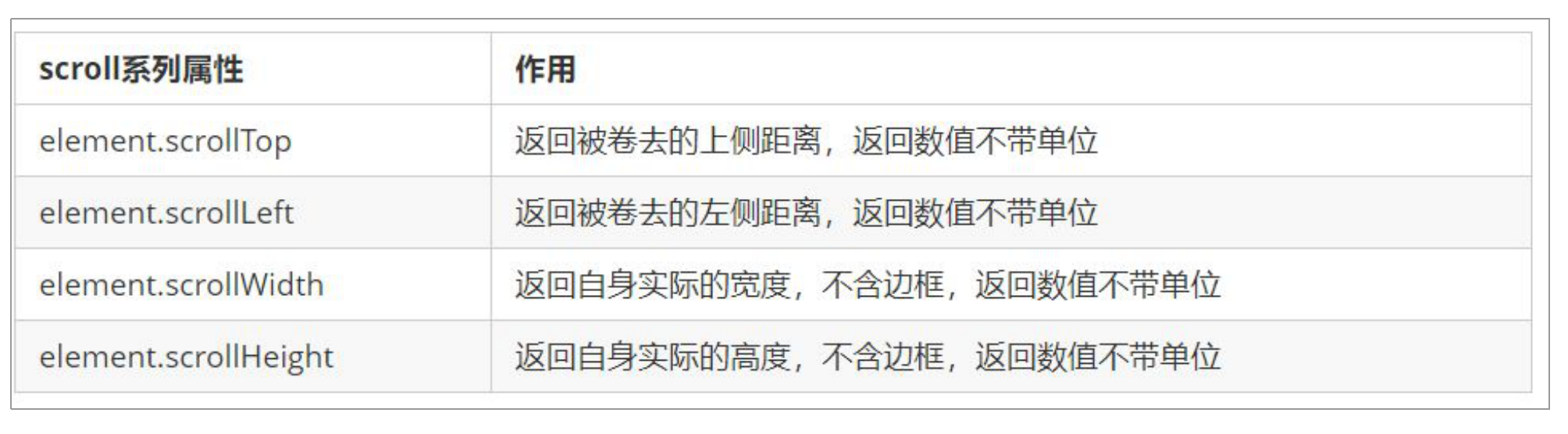
此时可以使用 pageshow事件来触发。，这个事件在页面显示时触发，无论页面是否来自缓存。在重新加载页

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

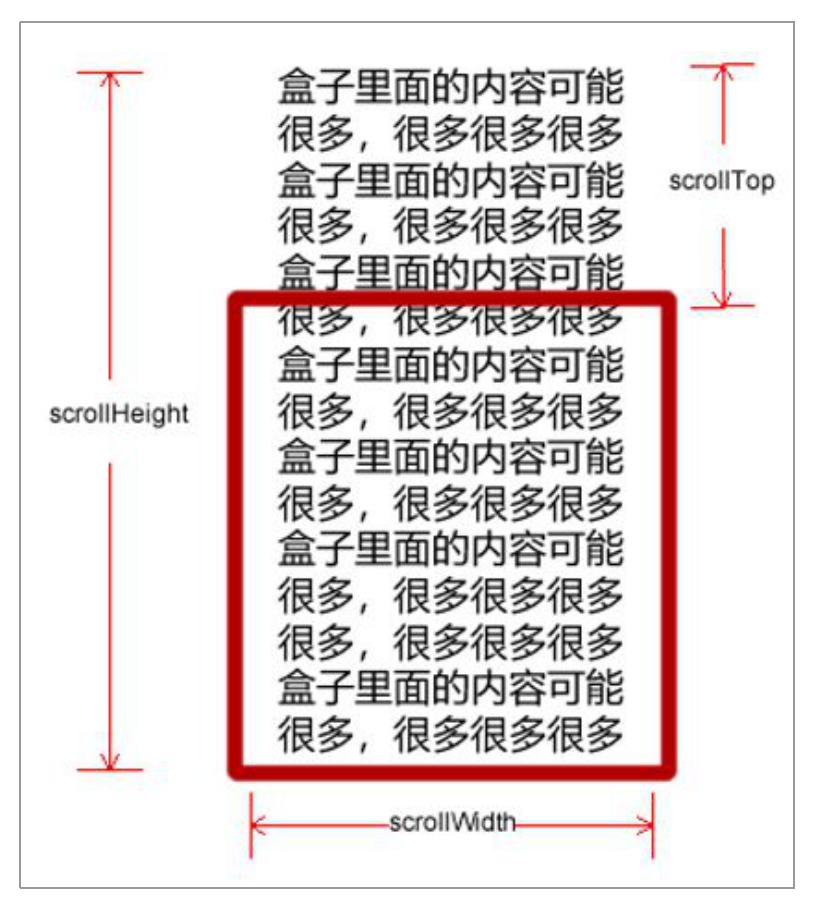
pageshow事件，注意这个事件给window添加。

**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。



注意：scroll与client有本质的区别！



**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（css：overflow：auto；）。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。

var div = document.querySelector('div');

console.log(div.scrollHeight);

console.log(div.clientHeight);

// scroll滚动事件当我们滚动条发生变化会触发的事件

div.addEventListener('scroll', function() {

console.log(div.scrollTop);

**案例：仿淘宝固定右侧侧边栏**

1.原先侧边栏是绝对定位

2. 当页面滚动到一定位置，侧边栏改为固定定位

3. 页面继续滚动，侧边栏会让返回顶部显示出来

**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

⑤ 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

.slider-bar {

position: absolute;

top: 300px;

left: 50%;//cssp231固定到版心右侧

margin-left: 600px;//cssp231固定到版心右侧

width: 45px;

height: 130px;

background-color: pink;

}

span {

display: none;

position: absolute;

}

<body>

<div class="slider-bar">

<span class="goBack">返回顶部</span>

</div>

<div class="header w">头部区域</div>

<div class="banner w">banner区域</div>

<div class="main w">主体部分</div>

<script>

//1. 获取元素

var sliderbar = document.querySelector('.slider-bar');

var banner = document.querySelector('.banner');

// banner.offestTop banner没有父元素则其就是被卷去头部的大小 一定要写到滚动的外面

var bannerTop = banner.offsetTop

// 当我们侧边栏固定定位之后应该变化的数值 侧边栏就不会突然跳动

var sliderbarTop = sliderbar.offsetTop - bannerTop;

// 获取main 主体元素

var main = document.querySelector('.main');

var goBack = document.querySelector('.goBack');

var mainTop = main.offsetTop;

// 2. 页面滚动事件 scroll

document.addEventListener('scroll', function() {

// console.log(11);

// window.pageYOffset 页面被卷去的头部

// console.log(window.pageYOffset);

// 3 .当我们页面被卷去的头部大于等于了 172 此时 侧边栏就要改为固定定位

if (window.pageYOffset >= bannerTop) {

sliderbar.style.position = 'fixed';

sliderbar.style.top = sliderbarTop + 'px';

} else {

sliderbar.style.position = 'absolute';

sliderbar.style.top = '300px';//记住300px加''

}

// 4. 当我们页面滚动到main盒子，就显示 goback模块

if (window.pageYOffset >= mainTop) {

goBack.style.display = 'block';

} else {

goBack.style.display = 'none';

}

})

</script>

</body>

**3.3 页面被卷去的头部兼容性解决方案（了解）**

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即<!DOCTYPE html>，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

};

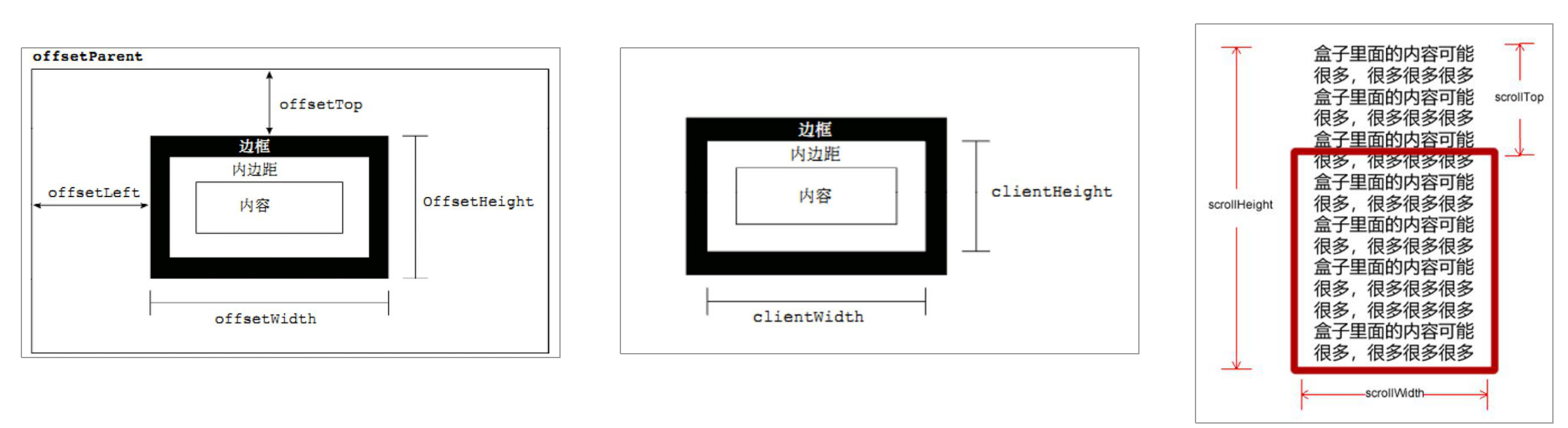
}

使用的时候 getScroll().left

**三大系列总结**



即：offsetwidth包含边框，clientwidth、scrollwidth不包含边框，scrollwidth包含元素卷出部分



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

4. 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（如果给父盒子添加mouseover鼠标在经过子盒子时没有事件发生，再冒泡给父盒子发生事件）

跟mouseenter搭配 鼠标离开 mouseleave 同样不会冒泡

**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

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

// 动画原理

// 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 本质是停止定时器

clearInterval(timer);

}

div.style.left = div.offsetLeft + 1 + 'px';//offsetLeft只能获值，不能赋值，所以要重新赋值给style

}, 30);

**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。

span {

position: absolute;//千万别忘记加定位！

left: 0;

top: 200px;

display: block;

width: 150px;

height: 150px;

background-color: purple;

}//style

<div></div>

<span>夏雨荷</span>

<script>

// 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 本质是停止定时器

clearInterval(timer);

}

obj.style.left = obj.offsetLeft + 1 + 'px';

}, 30);

}

var div = document.querySelector('div');

var span = document.querySelector('span');

// 调用函数

animate(div, 300);

animate(span, 200);

</script>

**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 声明定时器每次都要开辟一个内存空间，浪费内存资源。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。

<script>

// var obj = {};

// obj.name = 'andy';给对象添加属性的方法给其进行赋值操作

// 给不同的元素指定了不同的定时器

function animate(obj, target) {

// 当我们不断的点击按钮，这个元素的速度会越来越快，因为开启了太多的定时器

// 解决方案就是 让我们元素只有一个定时器执行

// 先清除以前的定时器，只保留当前的一个定时器执行

clearInterval(obj.timer);

//JS中获取过来的元素都是对象，直接添加方法或属性

//在 JavaScript 中，对象是一组无序的相关属性和方法的集合，所有的事物都是对象，例如字符串、数值、数组、函数等

obj.timer = setInterval(function() {

if (obj.offsetLeft >= target) {

// 停止动画 本质是停止定时器

clearInterval(obj.timer);

}

obj.style.left = obj.offsetLeft + 1 + 'px';

}, 30);

}

var div = document.querySelector('div');

var span = document.querySelector('span');

var btn = document.querySelector('button');

// 调用函数

animate(div, 300);

btn.addEventListener('click', function() {

animate(span, 200);

})

</script>

**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【步长公式】

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，出现小数的话会达不到目标距离就停止

并且要往上取整，假如剩下9px，step=9/10=0.9再取floor的话，step就是0了，所以要取ceil

**代码与4.5合并**

**4.5 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

function animate(obj, target) {

// 先清除以前的定时器，只保留当前的一个定时器执行

clearInterval(obj.timer);

obj.timer = setInterval(function() {

// 步长值写到定时器的里面

// 把我们步长值改为整数 不要出现小数的问题

// var step = Math.ceil((target - obj.offsetLeft) / 10);

var step = (target - obj.offsetLeft) / 10;

//前进时step为正值，后退时step为负值，正值时向上取整，负值时向下取整

step = step > 0 ? Math.ceil(step) : Math.floor(step);

if (obj.offsetLeft == target) {

// 停止动画 本质是停止定时器

clearInterval(obj.timer);

}

// 把每次加1 这个步长值改为一个慢慢变小的值 步长公式：(目标值 - 现在的位置) / 10

obj.style.left = obj.offsetLeft + step + 'px';

}, 15);

}

**4.6 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。

function animate(obj, target, callback) {

// console.log(callback); callback = 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);

// 回调函数写到定时器结束里面

if (callback) {

// 调用函数

callback();

}

}

obj.style.left = obj.offsetLeft + step + 'px';

}, 15);

}

animate(span, 800, function() {

// alert('你好吗');

span.style.backgroundColor = 'red';

});

**4.7 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。

<script src="animate.js"></script>//别忘记引入js文件

</head>

<body>

<div class="sliderbar">

<span>←</span>

<div class="con">问题反馈</div>

</div>

<script>

// 1. 获取元素

var sliderbar = document.querySelector('.sliderbar');

var con = document.querySelector('.con');

// 当我们鼠标经过 sliderbar 就会让 con这个盒子滑动到左侧

// 当我们鼠标离开 sliderbar 就会让 con这个盒子滑动到右侧

sliderbar.addEventListener('mouseenter', function() {

// animate(obj, target, callback);

animate(con, -160, function() {

// 当我们动画执行完毕，就把 ← 改为 →

sliderbar.children[0].innerHTML = '→';

});

})

sliderbar.addEventListener('mouseleave', function() {

// animate(obj, target, callback);

animate(con, 0, function() {

sliderbar.children[0].innerHTML = '←';

});

})

</script>

</body>

**5. 常见网页特效案例**

**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。



1.结构搭建

源码在jsapis\_material/第五天/轮播图制作

>>index.html

<div class="focus fl">

<!-- 左侧按钮 -->

<a href="javascript:;" class="arrow-l">

&lt;

</a>

<!-- 右侧按钮 -->

<a href="javascript:;" class="arrow-r">  </a>

<!-- 核心的滚动区域 一定要用ul来布局 -->

<ul>

<li>

<a href="#"><img src="upload/focus.jpg" alt=""></a>

</li>

<li>

<a href="#"><img src="upload/focus1.jpg" alt=""></a>

</li>

<li>

<a href="#"><img src="upload/focus2.jpg" alt=""></a>

</li>

<li>

<a href="#"><img src="upload/focus3.jpg" alt=""></a>

</li>

</ul>

<!-- 小圆圈 -->

<ol class="circle">

</ol>

</div>

>>index.css

.focus {

position: relative;

width: 721px;

height: 455px;

background-color: purple;

//超出部分隐藏

overflow: hidden;

}

.focus ul {

position: absolute;

top: 0;

left: 0;

//2.宽度为父盒子的600%，足够宽可以让轮播图图片并列成一行

width: 600%;

}

.focus ul li {

//1.添加浮动以后并没有让四个图片并列起来，因为ul不够宽

float: left;

}

2.鼠标经过轮播图显示左右按钮，离开则消失

**① 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。**

**② 此时需要添加 load 事件，当页面加载完以后再加载js。**

**③ 鼠标经过轮播图模块，左右按钮显示，离开隐藏左右按钮。**

④ 显示隐藏 display 按钮。

>>index.js

window.addEventListener('load', function () {

//1.获取元素,注意变量命名只允许\_和$符号，- 不可以

var arrow\_l = document.querySelector('.arrow-l');

var arrow\_r = document.querySelector('.arrow-r');

var focus = document.querySelector('.focus');

//当鼠标经过和离开大的focus盒子，箭头显示和隐藏

focus.addEventListener('mouseenter', function () {

arrow\_l.style.display = 'block';

arrow\_r.style.display = 'block';

})

focus.addEventListener('mouseleave', function () {

arrow\_l.style.display = 'none';

arrow\_r.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

//3.动态生成小圆圈,有几张图片我就生成几个小圆圈

//注意不能直接document.querySelector('ul');取focus里的ul，focus元素在上面已经获取好了

var ul = focus.querySelector('ul');

var ol = focus.querySelector('.circle');

// console.log(ul.children.length);//4

for (var i = 0; i < ul.children.length; i++) {

//创建一个小li 注意'li'

var li = document.createElement('li');

//把小li插入到ol里面 注意li

ol.appendChild(li);

}

//把ol里第一个li设置类名为current，获取current类选择器里的css样式，背景颜色为白色

ol.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。

for (var i = 0; i < ul.children.length; i++) {

//创建一个小li 注意'li'

var li = document.createElement('li');

//把小li插入到ol里面 注意li

ol.appendChild(li);

// 4. 小圆圈的排他思想 我们可以直接在生成小圆圈的同时直接绑定点击事件，记住写在小圆圈生成循环里面

li.addEventListener('click', function () {

// 干掉所有人 把所有的小li 清除 current 类名

for (var i = 0; i < ol.children.length; i++) {

//❤️记住一定要for循环用children[i]，不能直接ol.children.className = '';没办法直接给所有子节点进行改变

ol.children[i].className = '';

}

// 留下我自己 当前的小li 设置current 类名

this.className = 'current';

})

}

**① 点击小圆圈滚动图片**

② 此时index.js文件用到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

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。

>>index.html

<!❤️-- 这个animate.js 必须写到 index.js的上面引入 -->

<script src="js/animate.js"></script>

<!-- 引入我们首页的js文件 -->

<script src="js/index.js"></script>

>>index.css

.focus {

//使用动画函数的前提，该元素必须有定位

position: relative;

width: 721px;

height: 455px;

background-color: purple;

overflow: hidden;

}

>>index.js

for (var i = 0; i < ul.children.length; i++) {

//创建一个小li 注意'li'

var li = document.createElement('li');

//⭐️ 看下面的图!记录当前小圆圈的索引号 通过自定义属性来做

li.setAttribute('index', i);

//把小li插入到ol里面 注意li

ol.appendChild(li);

// 4. 小圆圈的排他思想 我们可以直接在生成小圆圈的同时直接绑定点击事件，记住写在小圆圈生成循环里面

li.addEventListener('click', function () {

// 干掉所有人 把所有的小li 清除 current 类名

for (var i = 0; i < ol.children.length; i++) {

//⭐️记住一定要for循环用children[i]，不能直接ol.children.className = '';没办法直接给所有子节点进行改变

ol.children[i].className = '';

}

// 留下我自己 当前的小li 设置current 类名

this.className = 'current';

// !5. 点击小圆圈，移动图片 当然移动的是 ul

// !ul 的移动距离 小圆圈的索引号 乘以 图片的宽度 注意是负值

// !当我们点击了某个小li 就拿到当前小li 的索引号

var focusWidth = focus.offsetWidth;

var index = this.getAttribute('index');

// console.log(index);

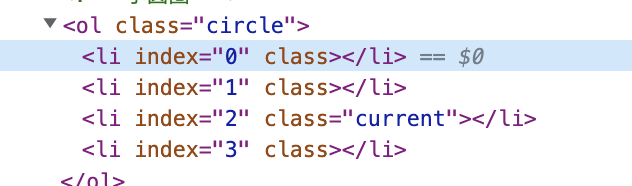
//！记住target值赋值负值，因为是往左移

animate(ul, -index \* focusWidth);

})

}

⭐️：



**① 点击右侧按钮一次，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

⑤ 当图片滚动到克隆的最后一张图片时【一定要克隆一张不然就没有滚动效果了】， 让ul 快速的、不做动画的跳到最左侧： left 为0

⑥ 同时num 赋值为0，可以从新开始滚动图片了

**怎么克隆第一张图片？**

1. 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

2.添加到 ul 最后面 appendChild

**注意：这时候发现左右箭头消失了，是因为定位元素的层级问题；**z-index适用于定位元素

.focus ul {

position: absolute;

top: 0;

left: 0;

width: 600%;

}

.arrow-l,

.arrow-r {

display: none;

position: absolute;

top: 50%;

margin-top: -20px;

width: 24px;

height: 40px;

background: rgba(0, 0, 0, .3);

text-align: center;

line-height: 40px;

color: #fff;

font-family: 'icomoon';

font-size: 18px;

//增加层级

z-index: 2;

}

>>index.js

// 6. 克隆第一张图片(li)放到ul 最后面

var first = ul.children[0].cloneNode(true);

ul.appendChild(first);

// 7. 点击右侧按钮， 图片滚动一张

var num = 0;

arrow\_r.addEventListener('click', function () {

//记得把原来的局部变量focusWidth提取到函数外变成全局变量

// 如果走到了最后复制的一张图片，此时 我们的ul 要快速复原 left 改为 0

if (num == ul.children.length - 1) {

ul.style.left = 0;

num = 0;

}

num++;

animate(ul, -num \* focusWidth);

})

**① 点击右侧按钮， 小圆圈跟随变化**

② 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。

③ 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

④ 如果circle == 4 就 从新复原为 0

// 8. 点击右侧按钮，小圆圈跟随一起变化 可以再声明一个变量控制小圆圈的播放

circle++;

// 如果circle == 4 说明走到最后我们克隆的这张图片了 我们就复原

if (circle == ol.children.length) {

circle = 0;

}

//先清除其余小圆圈的current类名

for (var i = 0; i < ol.children.length; i++) {

ol.children[i].className = '';

}

//留下当前小圆圈类名

ol.children[circle].className = 'current';

两个小bug解决方案：

bug1：箭头的移动并不能和圆圈的点击相承接，例如点击了第三个圆圈以后，再点击右箭头却换到第二个图片

因为点击箭头移动是靠num来控制的，但num与li点击事件没有任何关系

解决方案：

当我们点击了某个小li 就要把这个li 的索引号给 num ;num是全局变量所以可以在函数里被改变

bug2：点击了箭头以后小圆点没跟着变化

因为点击箭头移动是靠circle来控制的，但num与li点击事件没有任何关系

解决方案：

当我们点击了某个小li 就要把这个li 的索引号给circle ;circle是全局变量所以可以在函数里被改变

li.addEventListener('click', function () {

// 干掉所有人 把所有的小li 清除 current 类名

for (var i = 0; i < ol.children.length; i++) {

//⭐️记住一定要for循环用children[i]，不能直接ol.children.className = '';没办法直接给所有子节点进行改变

ol.children[i].className = '';

}

// 留下我自己 当前的小li 设置current 类名

this.className = 'current';

// !5. 点击小圆圈，移动图片 当然移动的是 ul

// !ul 的移动距离 小圆圈的索引号 乘以 图片的宽度 注意是负值

// !当我们点击了某个小li 就拿到当前小li 的索引号

var index = this.getAttribute('index');

//！！！当我们点击了某个小li 就要把这个li 的索引号给 num

num = index;

// ！！！当我们点击了某个小li 就要把这个li 的索引号给 circle

circle = index;

// num = circle = index;

// console.log(index);

//！记住target值赋值负值，因为是往左移

animate(ul, -index \* focusWidth);

})

左侧按钮功能制作：

arrow\_l.addEventListener('click', function () {

//记得把原来的局部变量focusWidth提取到函数外变成全局变量

// 如果走到了最后复制的一张图片，此时 我们的ul 要快速复原 left 改为 0

if (num == 0) {

num = ul.children.length - 1;

//千万别忘记加px,以及是负号，因为往左移动到最后一张图片

ul.style.left = -num \* focusWidth + 'px';

}

num--;

animate(ul, -num \* focusWidth);

// 8. 点击右侧按钮，小圆圈跟随一起变化 可以再声明一个变量控制小圆圈的播放

circle--;

//！！！ 如果circle <0 则小圆圈要变成第四个小圆圈

//！！！也可以优化为circle = circle < 0 ? ol.children.length - 1 : circle;

if (circle < 0) {

circle = ol.children.length - 1;

}

circleChange();

})

//！！！左右按钮点击事件这段代码一致所以可以封装出来

function circleChange() {

//先清除其余小圆圈的current类名

for (var i = 0; i < ol.children.length; i++) {

ol.children[i].className = '';

}

//留下当前小圆圈类名

ol.children[circle].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()即不用再一次一次点击右键即可调用右键点击事件

⑤ 鼠标经过focus 就停止定时器

⑥ 鼠标离开focus 就开启定时器

focus.addEventListener('mouseenter', function () {

arrow\_l.style.display = 'block';

arrow\_r.style.display = 'block';

//鼠标经过时，定时器停止

clearInterval(timer);

timer = null; // 清除定时器变量，释放该变量空间

})

focus.addEventListener('mouseleave', function () {

arrow\_l.style.display = 'none';

arrow\_r.style.display = 'none';

//不用再加var声明了，全局变量timer已经声明过了

timer = setInterval(function () {

//手动调用点击事件

arrow\_r.click();

}, 2000);

···················

//10.自动播放轮播图

var timer = setInterval(function () {

//手动调用点击事件

arrow\_r.click();

}, 2000);