目录

学习目标

- 能够说出常见 client 系列属性的作用
- 能够说出常见 scroll 系列属性的作用
- 能够封装简单动画函数
- 能够理解缓动动画的封装
- 能够使用动画函数

案例:仿京东放大镜

要求:

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。

分析:

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"小盒子"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. 发现，遮挡层位置不对，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

11. 移动换色遮挡层,大图片跟随移动功能

12. 求大图片的移动距离的公式

① **移动黄色遮挡层，大图片跟随移动功能。**

$$\frac{1}{2} = \frac{x}{4} \quad \text{求 x ?} \quad x = \frac{1 \times 4}{2}$$

② **求大图片的移动距离公式**

$$\frac{\text{遮挡层移动距离}}{\text{遮挡层最大移动距离}} = \frac{\text{大图片移动距离}}{\text{大图片最大移动距离}} \quad \text{求大图片移动距离？}$$

得出结论: 大图片的移动距离 = 遮挡层移动距离 * 大图片最大移动距离/遮挡层的最大移动距离

代码:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        /* 初始化样式开始 */
        *{
            margin: 0;
            padding: 0;
        }
        /* 去掉图片底部的缝隙 */
        img{
            vertical-align: bottom;
        }
        /* 初始化样式结束 */


        /* 具体样式,从大到小,先整体后细节,先上后下,先左后右 */
        .box{
            width: 1200px;
            margin:100px auto;
            position: relative;
        }
        .box .smallBox{
            width: 450px;
            height: 450px;
            border:1px solid #eeeeee;
            position: relative;
        }
        .box .smallBox .mask{
            width: 300px;
```

```
            height: 300px;
            background:rgba(254, 222, 79,0.5);
            position: absolute;
            left: 0;
            top: 0;
            /* 模拟鼠标指针样式 */
            cursor: move;
            display: none;
        }
        .box .bigBox{
            width: 540px;
            height: 540px;
            border:1px solid #eeeeee;
            /* 溢出隐藏 */
            overflow: hidden;
            position: absolute;
            top: 0;
            left: 451px;
            display: none;
        }
    </style>
</head>
<body>
    <!-- 最外层盒子 -->
    <div class="box">
        <!-- 小盒子 -->
        <div class="smallBox">
            <!-- 小图片 -->
            <img src="images/v30_small.jpg"/>
            <!-- 遮罩层 -->
            <div class="mask"></div>
        </div>
        <!-- 大盒子 -->
        <div class="bigBox">
            <!-- 大图片 -->
            <img src="images/v30_big.jpg"/>
        </div>
    </div>
```

```
    <script>
        // 获取相关对象
        // 获取最外层盒子对象
        var box = document.querySelector(".box");
        // 小盒子对象
        var smallBox = document.querySelector(".smallBox");
        // 遮罩层对象
        var mask = document.querySelector(".mask");
        // 大盒子对象
        var bigBox = document.querySelector(".bigBox");
        // 获取大盒子中图片对象
        var bigImg = document.querySelector(".bigBox img");


        // 1. 实现鼠标移上小盒子以后,显示遮罩层,显示大盒子
        smallBox.onmouseover = function(){
            mask.style.display = "block";
            bigBox.style.display = "block";
        }
```

```
    // 2. 实现鼠标移出小盒子以后,隐藏遮罩层,隐藏大盒子
    smallBox.onmouseout = function(){
        mask.style.display = "none";
        bigBox.style.display = "none";
    }
    // 3. 实现遮罩层跟随鼠标在小盒子移动,而且遮罩层是在小盒子里面移动的
    smallBox.onmousemove = function(e){
        e = e || window.event;
        // 先找到鼠标在盒子中的位置    当前鼠标在文档中的坐标 - 小盒子的偏移量
        var x = e.pageX || e.x;
        var y = e.pageY || e.y;


        // 获取小盒子的偏移量
        // element.offsetTop 返回元素相对带有"非静态定位"父辈元素上方的偏移量 如果父辈都没有定位则返回相对"body"的"上方"偏移量
        // element.offsetLeft 返回元素相对带有"非静态定位"父辈元素左方的偏移量 如果父辈都没有定位则返回相对"body"的"左"方偏移量
        // var smallBoxLeft = smallBox.offsetLeft;
        // var smallBoxTop = smallBox.offsetTop;
        // console.log( smallBoxLeft,smallBoxTop ); // 0 0


        // 因为获取小盒子的偏移量有问题,但是我们发现最外层.box盒子的偏移量跟小盒子偏移量是一样的,所以我们就求.box盒子的偏移量即可
        // console.log( box.offsetLeft , box.offsetTop);
        var boxLeft = box.offsetLeft;
        var boxTop  = box.offsetTop;


        // 得到鼠标在小盒子中的位置, 就是遮罩层左上角的位置
        var maskX = x - boxLeft;
        var maskY = y - boxTop;

        // 4. 实现鼠标出现在遮罩层的中间    让 水平坐标减去遮罩层宽度的一半   让垂直坐标减去遮罩层高度的一半
        maskX = maskX - (mask.offsetWidth/2);
        maskY = maskY - (mask.offsetHeight/2);


        // 5. 设置遮罩层可以移动的水平位置
        if( maskX < 0){ //左边界,不能小于0,小于0就让maskX等于0
            maskX = 0;
        }else if(maskX > smallBox.offsetWidth - mask.offsetWidth){// 右边界, 小盒子的宽度 - 遮罩层的宽度
            maskX = smallBox.offsetWidth - mask.offsetWidth;
        }


        // 6. 设置遮罩层可以移动的垂直位置
        if( maskY < 0){ //上边界,不能小于0,小于0就让maskY等于0
            maskY = 0;
        }else if(maskY > smallBox.offsetHeight - mask.offsetHeight){// 下边界, 小盒子的高度度 - 遮罩层的高度度
            maskY = smallBox.offsetHeight - mask.offsetHeight;
        }


        // 设置遮罩层的位置
        mask.style.left = maskX + "px";
        mask.style.top  = maskY + "px";


        // 7. 让大盒子中的图片跟随相反方向遮罩层移动
        // 大图片的移动距离 = 遮挡层移动距离 * 大图片最大移动距离/遮挡层的最大移动距离


        // 求大图片最大移动的水平距离
        var bigImgMoveX = bigImg.offsetWidth - bigBox.offsetWidth;
        // 求大图片最大移动的垂直距离
        var bigImgMoveY = bigImg.offsetHeight - bigBox.offsetHeight;

        // 求大图片移动的水平坐标
        var bigImgX = maskX * bigImgMoveX / (smallBox.offsetWidth - mask.offsetWidth);
        // 求大图片移动的垂直坐标
        var bigImgY = maskY * bigImgMoveY / (smallBox.offsetHeight - mask.offsetHeight);
        console.log( bigImgX, bigImgY);
        // 设置大图片的位置
        // 注意:因为我们需要通过给大图片设置偏移量移动图片,所以大图片不能是静态定位
        bigImg.style.position = "absolute";
        // 因为我们的大图片需要跟遮罩层相反的方向移动,所以需要取负
        bigImg.style.left = -bigImgX + "px";
        bigImg.style.top  = -bigImgY + "px";
    }
</script>
```

元素可视区 client 系列

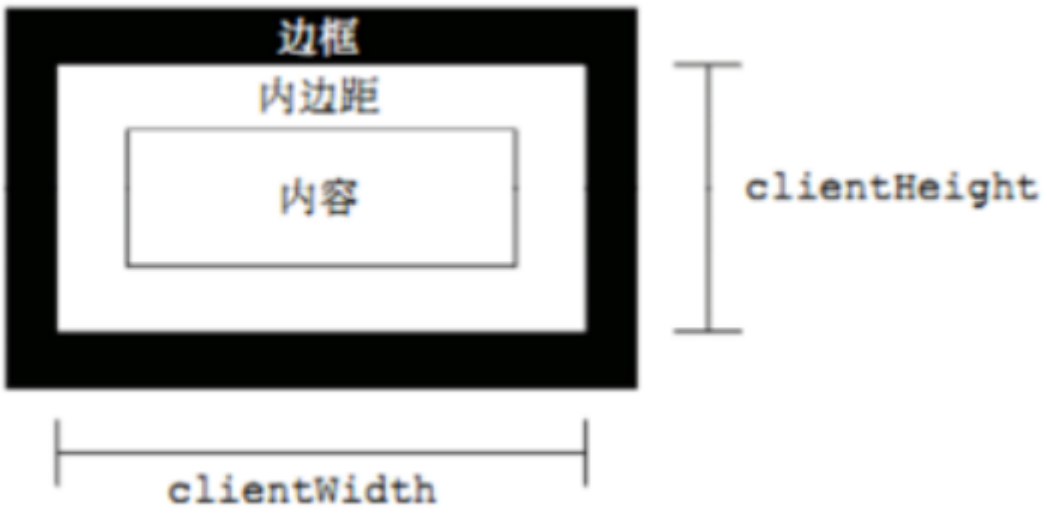
client概述

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	返回上边框的大小
element.clientLeft	返回左边框的大小
element.clientWidth	返回自身包括padding、内容区宽度 ,不含边框 , 返回数值不带单位
element.clientHeight	返回自身包括padding、内容区高度 ,不含边框 , 返回数值不带单位

注意:clientTop可以获取上边框的大小,clientLeft可以获取左边框的大小,**没有clientBottom跟clientRight**

小结: client 宽度 和我们offsetWidth 最大的区别就是 **不包含边框**



举例:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        div {
            width: 200px;
            height: 200px;
            background-color: pink;
            border-top:10px solid red;
            border-bottom:20px solid red;
            border-left:30px solid red;
            border-right:40px solid red;


            padding-top:10px;
            padding-bottom:20px;
            padding-left:30px;
            padding-right:40px;
        }
    </style>
</head>
<body>
    <div>div的内容</div>
    <script>
        // client概述
        // 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。


        // 获取div元素
        var div = document.querySelector("div");


        // element.clientTop 返回上边框的大小
        console.log("div的上边框大小为:" + div.clientTop );
        // element.clientLeft 返回左边框的大小
        console.log("div的左边框大小为:" + div.clientLeft );


        // element.clientWidth 返回自身包括padding、内容区宽度 ,不含边框 , 返回数值不带单位
        console.log("div的宽度+padding左右为:" + div.clientWidth );

        // element.clientHeight 返回自身包括padding、内容区高度 ,不含边框 , 返回数值不带单位
        console.log("div的高度+padding上下为:" + div.clientHeight );

        // 注意:clientTop可以获取上边框的大小,clientLeft可以获取左边框的大小,没有clientBottom跟clientRight
        // 没有clientBottom跟clientRight
        console.log( div.clientBottom );// undefined
        console.log( div.clientRight );// undefined


        // 小结: client宽度 和我们offsetWidth 最大的区别就是 不包含边框
        // 小结: client高度 和我们offsetHeight 最大的区别就是 不包含边框
    </script>
</body>
</html>
```

元素滚动 scroll 系列

scroll 概述

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.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

注意: scrollHeight跟clientHeight的区别在于,文字内容超出盒子高度的时候,scrollHeight获取的是内容的高度,而clientHeight获取的还是盒子的高度

举例: scroll滚动事件当我们滚动条发生变化会触发的事件

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        div{
            background:pink;
            width: 200px;
            height: 200px;
            margin-left: 100px;
            margin-top: 100px;
            /* 设置溢出内容自动显示滚动条 */
            overflow:auto;
            /* 设置边框 */
            border: 5px solid blue;
            /* 设置内填充 */
            padding: 10px;
        }
        p{
            width: 200px;
            background:skyblue;
            margin-left: 100px;
            margin-top: 50px;
            /* 设置超出内容,自动显示滚动条 */
            overflow: auto;
        }
    </style>
</head>
<body>
    <div>盒子里面的内容可能很多,很多很多很多盒子里面的内容可能很多,很多很多很多盒子里面的内容可能很多,很多很多很多盒子里面的内容可能很多,很多很多很多盒子里面的内容可能很多,很多很多很多盒子里面的内容可能很多,很多很多很多盒子里面的内容可能很多,很多很多很多盒子里面的内容可能很多,很多很多很多盒子里面的内容可能很多,很多很多很多盒子里面的内容可能很多,很多很多很多</div>


    <!-- 默认英文字母内容是不会换行的,而中文内容是会自动换行 -->
    <p>pppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp</p>
    <script>
        // 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。
        // 获取div元素
        var div = document.querySelector("div");


        // element.scrollWidth 返回自身实际的宽度,不含边框,返回数值不带单位


        // 因为水平内容没有超出宽度,没有显示滚动条,所以 clientWidth跟scrollWidth是一样的数值
        console.log("div元素可视区宽度" +  div.clientWidth );
        console.log("div元素内容宽度" +  div.scrollWidth );
        console.log( "" );


        // element.scrollHeight 返回自身实际的高度,不含边框,返回数值不带单位
        console.log("div元素可视区高度" + div.clientHeight );
        // 有垂直滚动条的时候, scrollHeight获取的是内容的高度,而clientHeight获取的还是盒子的高度
        // 如果垂直没有滚动条,scrollHeight跟clientHeight得到的值也是一样
        // 注意:   scrollHeight跟clientHeight的区别在于,文字内容超出盒子高度的时候,scrollHeight获取的是内容的高度,而clientHeight获取的还是盒子的高度
        console.log("div元素内容高度" + div.scrollHeight );
        console.log( "" );


        // 元素滚动事件 onscroll  当元素有滚动条滚动的时候触发
        div.onscroll = function(){
            // element.scrollTop 返回被卷去的上侧距离,返回数值不带单位 这个scrollTop用的比较多,其实就是滚动条滚动的距离
            console.log("被卷去的上侧距离为:" + div.scrollTop);
            console.log("");
        }


        // 小结
        // 最大的滚动距离 = 元素.scrollHeight - 元素.clientHeight

        var p = document.querySelector("p");
        console.log("p段落元素可视区宽度" + p.clientWidth );
        console.log("p段落内容宽度" + p.scrollWidth );
        p.onscroll = function(){
            // element.scrollLeft 返回被卷去的左侧距离,返回数值不带单位
            console.log("被卷去的左侧距离为:" + p.scrollLeft);
            console.log("");
        }
    </script>
</body>
</html>
```

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。

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        body{
            height: 2000px;
            background:skyblue;
        }
    </style>
</head>
<body>
    <script>
        // 页面被卷去的头部,其实就页面滚动的垂直滚动条滚动的距离
        // 页面什么时候回出现垂直滚动条,就是当内容的高度很多时候,超出浏览器可以显示的范围,浏览器显示不下,浏览器就会出现滚动条
        // 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。

        // 获取body标签
        var objBody = document.body;
        // 滚动条在滚动的时候,可以触发onscroll事件
        objBody.onscroll = function(){
            // 对象.scrollTop 就可以返回对象被卷去的上侧距离
            console.log( objBody.scrollTop );// 一直输出都是0
        }
    </script>
</body>
</html>
```

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`

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
```

这个就是声明了DTD

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        body{
            height: 2000px;
            background:skyblue;
        }
    </style>
</head>
<body>
    <script>
        // 页面被卷去的头部,其实就页面滚动的垂直滚动条滚动的距离
        // 页面什么时候回出现垂直滚动条,就是当内容的高度很多时候,超出浏览器可以显示的范围,浏览器显示不下,浏览器就会出现滚动条
        // 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
        // 如果HTML代码页面中有doctype html 这样的,我们就叫做声明了DTD, 那么这个时候想要获取页面被卷去的头部距离,就需要通过"HTML"标签


        // 获取body标签
        var objBody = document.body;
        // 获取html标签
        var objHTML = document.documentElement;

        // 注意:页面滚动的时,需要绑定onscroll事件的时候,则要给body标签绑定或者给window对象绑定才可以触发onscroll事件;  如果给html标签绑定,则触发不了onscroll事件
        window.onscroll = function(){
            // 因为声明DTD,所有通过HTML对象获取scrollTop值
            console.log( objHTML.scrollTop );
        }
    </script>
</body>
</html>
```

2. 未声明 DTD，使用 `document.body.scrollTop`

```html
<!-- 我们把DTD注释了,那么代表没有声明DTD -->
<!-- <!DOCTYPE html> -->
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        body{
            height: 2000px;
            background:skyblue;
        }
    </style>
</head>
<body>
    <script>
        // 2. 未声明 DTD，使用  document.body.scrollTop
        // 给window对象绑定滚动事件
        window.onscroll = function(){
            // 没有声明DTD的时候,想要获取页面被卷去的距离,需要通过body标签
            console.log( document.body.scrollTop );
        }
    </script>
</body>
</html>
```

3. 新方法 window.pageYOffset和 window.pageXOffset，**IE9 开始支持**

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        body{
            height: 2000px;
            background:skyblue;
        }
    </style>
</head>
<body>
    <script>
        // 新方法 window.pageYOffset和 window.pageXOffset，IE9 开始支持
        window.onscroll = function(){
            //  window.pageYOffset不管有没声明DTD,都可以获取到页面被卷去的距离 IE9 开始支持
            console.log( window.pageYOffset );
        }
    </script>
</body>
</html>
```

解决方法: 我们可以封装一个兼容函数　　了解

```
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   或者  getScroll().top
```

测试代码:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        body{
            height: 2000px;
            background: pink;
        }
    </style>
</head>
<body>
    <script>
        function getScroll() {
            // return返回了对象,这个对象里面有两个属性,一个是left,一个是top
            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   或者  getScroll().top


        window.onscroll = function(){
            console.log( getScroll().top );
        }
    </script>
</body>
</html>
```

案例:仿淘宝固定右侧侧边栏

要求:

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，会让 返回顶部显示出来

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
6. 为了解决侧边栏跳跃问题,需要设置固定定位以后 top的值为 侧边栏层.offsetTop - 轮播层.offsetTop的值;

特别注意:

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**

代码:

```
<!DOCTYPE html>
<html lang="en">


<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <style>
        .slider-bar {
            position: absolute;
            left: 50%;
            top: 300px;
            margin-left: 600px;
            width: 45px;
            height: 130px;
            background-color: pink;
        }

        .w {
            width: 1200px;
            margin: 10px auto;
        }

        .header {
            height: 150px;
            background-color: purple;
        }

        .banner {
            height: 250px;
            background-color: skyblue;
        }

        .main {
            height: 1000px;
            background-color: yellowgreen;
        }

        .goBack {
            display: none;
            position: absolute;
            bottom: 0;
            text-decoration: none;
            color:gray;
        }
    </style>
</head>


<body>
    <div class="slider-bar">
        <a href="#" class="goBack">返回顶部</a>
    </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>
</body>
<script>
    function getScroll(){
        return {
            left : window.pageXOffset || document.body.scrollLeft || document.documentElement.scrollLeft || 0,
            top  : window.pageYOffset || document.body.scrollTop || document.documentElement.scrollTop || 0
        }
    }


    // 获取.header对象
    var objHeader = document.querySelector(".header");
    // 获取.banner对象
    var objBanner = document.querySelector(".banner");
    // 获取侧边栏对象
    var sliderBar = document.querySelector(".slider-bar");
```

```
    var sliderBarOffsetTop = sliderBar.offsetTop;


    // 为了解决跳跃问题,算出侧边栏顶部距离.banner层顶部的距离
    var sliderBarTop =  sliderBar.offsetTop - objBanner.offsetTop;


    // 获取.main对象
    var main = document.querySelector(".main");
    var mainTop = main.offsetTop;


    // 获取返回顶部对象
    var goBack = document.querySelector(".goBack");


    // 页面滚动事件
    window.onscroll = function(){
        // 获取页面滚动的距离
        var pageTop =  getScroll().top ;
        // top这个变量名最好也不要用,因为在window对象下,也有一个叫做top的属性
        // console.log( PageTop );

        // 1. 当页面滚动到.banner层的时候,把侧边栏变成固定定位;如果在.banner层之前侧边栏都是绝对定位
        if( pageTop > objBanner.offsetTop ){
            sliderBar.style.position = "fixed";
            // 为什么会跳跃,因为设置了固定定位,但是没有设置垂直方向的偏移量
            sliderBar.style.top = sliderBarTop + "px";
        }else{
            sliderBar.style.position = "absolute";
            sliderBar.style.top = sliderBarOffsetTop + "px";
        }


        // 2. 当页面滚动到主体内容部分时候,显示返回顶部a链接;当页面没有滚动到主体内容部分时候,则隐藏返回顶部a链接
        if( pageTop > mainTop ){// 如果当前滚动的距离大于了main层的上外边距
            goBack.style.display = "block";// 显示返回顶部链接
        }else{
            goBack.style.display = "none";// 隐藏返回顶部链接
        }
    }
</script>
</html>
```

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**元素**滚动距离 `scrollTop` `scrollLeft`

4.注意:获取**页面**滚动的距离是通过 `window.pageYOffset` 获得

mouseenter 和mouseover的区别 经典面试题

- 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

举例: 分别给.father绑定mouseover和mouseenter事件

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        .father{
            width: 500px;
            height: 500px;
            background:skyblue;
            margin:100px auto;
        }
        .father .son{
            width: 100px;
            height: 100px;
            background: red;
        }
    </style>
</head>
<body>
    <div class="father">
        <div class="son">子元素</div>
    </div>
    <script>
        // mouseenter 和mouseover的区别
        // mouseenter一般跟mouseleave搭配
        // mouseover一般跟mouseout搭配


        // 获取父元素对象
        var father =  document.querySelector(".father");

        // 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



        // mouseenter  代表鼠标进入元素
        // mouseenter  只会经过自身盒子触发
        father.onmouseenter = function(){
            console.log( "mouseenter输入进入元素了 ");
            console.log("");
        }
        // mouseover 代表鼠标移上元素
        // mouseover 鼠标经过自身盒子会触发，经过子盒子还会触发
        father.onmouseover = function(){
            console.log( "mouseover输入移上元素了 ");
            console.log("");
        }
    </script>
</body>
</html>
```

动画函数封装

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

举例:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        div{
            width: 100px;
            height: 100px;
            background: pink;
            /* 设置绝对定位 */
            position: absolute;
            left: 0;
            top: 0;
        }
    </style>
</head>
<body>
    <div>div的内容</div>


    <script>
        // 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


        // 获取div对象
        var div = document.querySelector("div");
        // 开启定时器,让div开始移动
        var timer = setInterval(function(){
            // 假设我们div移动到400距离的时候,我们就停止移动
            if(div.offsetLeft == 400){
                // 清除定时器
                clearInterval( timer );
            }else{
                // 让盒子在当前位置加上1个移动距离,再通过style属性设置left属性值
                div.style.left = div.offsetLeft + 1 + "px";
            }
        },15);
    </script>
</body>
</html>
```

简单动画函数封装

简单动画函数封装　函数需要两个参数　obj目标对象　　target 目标位置

举例:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        div{
            width: 100px;
            height: 100px;
            background: pink;
            /* 设置绝对定位 */
            position: absolute;
            left: 0;
            top: 0;
        }
        span{
            width: 150px;
            height: 150px;
            background:orange;
            /* 设置绝对定位 */
            position: absolute;
            left: 0;
            top: 200px;
        }
        button{
```

```
            position:absolute;
            top: 400px;
        }
    </style>
</head>
<body>
    <div>下冰雹</div>
    <span>夏雨荷</span>
    <button>点我,夏雨荷和下冰雹开始运动</button>


    <script>
        // 为了后续方便的时候,我们传了两个参数,一个是需要做动画的对象,一个是目标位置
        // 简单动画函数封装     函数需要两个参数    obj目标对象    target 目标位置
        function animate(obj,target){
            // 开启定时器,让对象开始移动
            var timer = setInterval(function(){
                if(obj.offsetLeft == target){
                    // 清除定时器
                    clearInterval( timer );
                }else{
                    obj.style.left = obj.offsetLeft + 1 + "px";
                }
            },15);
        }


        // 获取按钮对象
        var btn = document.querySelector("button");
        // 获取div对象
        var div = document.querySelector("div");
        // 获取span对象
        var span = document.querySelector("span");


        btn.onclick = function(){
            animate( div, 400 );
            animate( span, 800 );
        }
    </script>
</body>
</html>
```

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**。

另外:

```
1 还有一个小bug,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
2 解决方案就是 让我们元素只有一个定时器执行
3 先清除以前的定时器，只保留当前的一个定时器执行
```

代码:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        div{
            width: 100px;
            height: 100px;
            background: pink;
            /* 设置绝对定位 */
            position: absolute;
            left: 0;
            top: 0;
        }
        span{
            width: 150px;
            height: 150px;
            background:orange;
            /* 设置绝对定位 */
            position: absolute;
            left: 0;
            top: 200px;
        }
        button{
            position:absolute;
            top: 400px;
        }
    </style>
</head>
<body>
    <div>下冰雹</div>
    <span>夏雨荷</span>
    <button>点我,夏雨荷和下冰雹开始运动</button>


    <script>
        // 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。
        function animate(obj,target){
            // 开启定时器,让对象开始移动
            obj.timer = setInterval(function(){
                if(obj.offsetLeft == target){
                    // 清除定时器
                    clearInterval( obj.timer );
                }else{
                    obj.style.left = obj.offsetLeft + 1 + "px";
                }
            },15);


            // console.dir可以输出对象的属性和方法 比console.log更为强大
            // 可以通过对象.属性名 = 属性 给对象 给对象添加属性
            // obj.abc = 10;
            // console.dir( obj );
            // console.log("");
        }



        // 获取按钮对象
        var btn = document.querySelector("button");
        // 获取div对象
        var div = document.querySelector("div");
        // 获取span对象
        var span = document.querySelector("span");


        btn.onclick = function(){
            animate( div, 400 );
            animate( span, 800 );
        }
    </script>
</body>
</html>
```

另外:

```
1 还有一个小bug,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
2 解决方案就是 让我们元素只有一个定时器执行
3 先清除以前的定时器，只保留当前的一个定时器执行
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        div{
            width: 100px;
            height: 100px;
            background: pink;
            /* 设置绝对定位 */
            position: absolute;
            left: 0;
            top: 0;
        }
        span{
            width: 150px;
            height: 150px;
            background:orange;
            /* 设置绝对定位 */
            position: absolute;
            left: 0;
            top: 200px;
        }
        button{
            position:absolute;
            top: 400px;
        }
    </style>
</head>
<body>
    <div>下冰雹</div>
    <span>夏雨荷</span>
    <button>点我,夏雨荷和下冰雹开始运动</button>


    <script>
        function animate(obj,target){
            // 还有一个小bug,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 );


            // 开启定时器,让对象开始移动
            obj.timer = setInterval(function(){
                if(obj.offsetLeft == target){
                    // 清除定时器
                    clearInterval( obj.timer );
                }else{
                    obj.style.left = obj.offsetLeft + 1 + "px";
                }
```

```
            },15);
        }


        // 获取按钮对象
        var btn = document.querySelector("button");
        // 获取div对象
        var div = document.querySelector("div");
        // 获取span对象
        var span = document.querySelector("span");


        btn.onclick = function(){
            animate( div, 400 );
            animate( span, 800 );
        }
    </script>
</body>
</html>
```

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 这个距离一般称为步长

缓动动画公式:(目标值 - 现在的位置) / 10

假设 目标值为100 现在的位置为 0

第一次运动 (100-0)/10 = 10

第二次运动 (100-10)/10 = 9

第三次运动 (100-19)/10 = 8.1

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** 步长值是正值**向上**取整 步长值是负值**向下**取整 如果不取整,容易导致达不到目标值的问题

代码: 一个按钮,点击以后元素去到500的位置

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>Document</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        div{
            width: 100px;
            height: 100px;
            background: pink;
            /* 设置绝对定位 */
            position: absolute;
            left: 0;
            top: 0;
        }
        button{
            position:absolute;
            top: 120px;
        }
    </style>
</head>
<body>
    <div>下冰雹</div>
    <button>点我,夏雨荷开始运动</button>


    <script>
        // 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     这个距离一般称为步长
        缓动动画公式:(目标值 - 现在的位置) / 10

        假设 目标值为100 现在的位置为 0

        第一次运动 (100-0)/10 = 10
        第二次运动 (100-10)/10 = 9
        第三次运动 (100-19)/10 = 8.1 */


        // 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      步长值是正值向上取整     步长值是负值向下取整     如果不取整,容易导致达不到目标值的问题


        function animate(obj,target){
            // 先清除定时器
            clearInterval( obj.timer );

           // 再开启定时器,让对象开始移动
           obj.timer = setInterval(function(){
               // 缓动动画公式:(目标值 - 现在的位置) / 10
                /*
                假设target = 400;
                step = (400 - 0) / 10 = 40   现在的位置变成的 40
                step = (400 - 40)/ 10 = 36   现在的位置变成了 40+36 = 76
                step = (400 - 76)/ 10 = 32.4 现在的位置变成了 76+32.4
                ...
                */
                // 定义一个变量 step 保存对象每次移动的速度  这个step我们也会叫步长值
                var step = (target - obj.offsetLeft)/10;
                // 注意步长值需要取整    步长值是正值向上取整     步长值是负值向下取整     如果不取整,容易导致达不到目标值的问题
                step = Math.ceil( step );

                if(obj.offsetLeft == target){
                    // 到达目标值的时候,清除定时器
                    clearInterval( obj.timer );
                }else{
                    obj.style.left = obj.offsetLeft + step + "px";
                }
            },15);
        }


        // 获取按钮对象
        var btn = document.querySelector("button");
        // 获取div对象
        var div = document.querySelector("div");


        btn.onclick = function(){
            animate( div, 500 );
```

```
        }
    </script>
</body>
</html>
```

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 往大的 取整

2.如果是负值，则步长 向小的 取整

代码： 两个按钮,一个点击以后元素去到500的位置,另一个点击以后元素去到800的位置

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        div{
            width: 100px;
            height: 100px;
            background: pink;
            /* 设置绝对定位 */
            position: absolute;
            left: 0;
            top: 0;
        }
        button{
            margin-top: 200px;
        }
    </style>
</head>
<body>
    <div>下冰雹</div>
    <button>点我去到500位置</button>
    <button>点我去到800位置</button>


    <script>
        // 动画函数
        function animate(obj,target){
            // 先清除定时器
            clearInterval( obj.timer );

           // 再开启定时器,让对象开始移动
           obj.timer = setInterval(function(){
                // 缓动动画公式:(目标值 - 现在的位置) / 10


                // 目标值是 500
                // 现在的位置 是 800
                // var step = (500-800)/10 = -30    现在位置变成了 770
                // var step = (500-770)/10 = -27    现在位置变成了 743
                // var step = (500-743)/10 = -24.3  现在位置变成了 718.7
                // ...


                // 定义一个变量 step 保存对象每次移动的速度  这个step我们也会叫步长值
                var step = (target - obj.offsetLeft)/10;
                // 注意步长值需要取整   步长值是正值向上取整    步长值是负值向下取整    如果不取整 容易导致达不到目标值的问题
```

```
            /* if( step > 0){// 步长值是正值向上取整
                step = Math.ceil( step );
            }else{// 步长值是负值向下取整
                step = Math.floor( step );
            } */


            // 上面的if...else可以写成三目运算符
            step = step > 0  ? Math.ceil( step ) : Math.floor( step ) ;

            if(obj.offsetLeft == target){
                // 到达目标值的时候,清除定时器
                clearInterval( obj.timer );
            }else{
                obj.style.left = obj.offsetLeft + step + "px";
            }
        },15);
    }


    // 获取div对象
    var div = document.querySelector("div");
    // 获取所有的按钮对象
    var btns = document.querySelectorAll("button");
    btns[0].onclick = function(){
        animate( div, 500 );
    }
    btns[1].onclick = function(){
        animate( div, 800 );
    }
</script>
</body>
</html>
```

动画函数添加回调函数

回调函数原理:函数也可以作为一个形式参数传递给其他函数。**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。**

代码: 点击按钮,然后元素开始移动,移动到目标值以后,改变元素的背景颜色为红色

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        div{
            width: 100px;
            height: 100px;
            background: pink;
            /* 设置绝对定位 */
            position: absolute;
            left: 0;
            top: 0;
        }
        button{
            margin-top: 200px;
        }
    </style>
</head>
<body>
    <div>下冰雹</div>
    <button>点我开始运动</button>


    <script>
        // 动画函数添加回调函数
        // 回调函数原理:函数也可以作为一个形式参数传递给其他函数。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。


        // 动画函数
        function animate(obj,target,callback){
            // 先清除定时器
            clearInterval( obj.timer );

           // 再开启定时器,让对象开始移动
           obj.timer = setInterval(function(){
                // 定义一个变量 step 保存对象每次移动的速度  这个step我们也会叫步长值
                var step = (target - obj.offsetLeft)/10;


                // 步长值是正值向上取整     步长值是负值向下取整
                step = step > 0  ? Math.ceil( step ) : Math.floor( step ) ;

                if(obj.offsetLeft == target){
                    // 到达目标值的时候,清除定时器
                    clearInterval( obj.timer );


                    // console.log( callback );
                    // 因为不一定每次调用animate都需要回调函数
                    if( callback ){// 判断callback参数是否存在,不存在默认callback为undefined
                        callback();// 调用callback函数
                    }


                }else{
                    obj.style.left = obj.offsetLeft + step + "px";
                }
            },15);
        }


        // 获取div对象
        var div = document.querySelector("div");
        // 获取所有的按钮对象
        var btn = document.querySelector("button");
        btn.onclick = function(){
            // 要求:点击按钮,然后元素开始移动,移动到目标值以后,改变元素的背景颜色为红色
            // animate( div, 500 );


            // 在调用animate的时候,可以给第三个参数参一个匿名函数
            animate( div, 500 , function(){
                // 改变div的背景颜色
                div.style.backgroundColor = "red";
            });
        }
    </script>
</body>
</html>
```

今日总结

xmind要做,也要交

今日作业

具体查看作业文件夹