

1.1. 元素偏移量 offset 系列

1.1.1 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。

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

| 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 |

1.1.2 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

因为平时我们都是给元素注册触摸事件，所以重点记住 targetTouches

offset系列属性

```
<style>
  * {margin: 0;padding: 0;}

  .box {/* position: relative; */width: 200px;height: 200px;background-color: pink;margin:
```

```

150px;}}
    .one {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>
<body>
<div class="box">
    <div class="one"></div>
</div>
<div class="w"></div>
<script>
    // offset 系列
    var box = document.querySelector('.box');
    var one = document.querySelector('.one');
    // 1.可以得到元素的偏移 位置 返回的不带单位的数值
    console.log(box.offsetTop);
    console.log(one.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(one.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(one.offsetParent); // 返回带有定位的父亲 否则返回的是body
    console.log(one.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>

```

offset与style的区别

```

<style>
    .box { width: 200px;height: 200px;background-color: pink;padding: 10px;}
</style>
<body>
    <div class="box" style="width: 200px;"></div>
    <script>
        // offset与style的区别
        var box = document.querySelector('.box');
        console.log(box.offsetWidth);
        console.log(box.style.width);
        // box.offsetWidth = '300px';
        box.style.width = '300px';
    </script>
</body>

```

计算鼠标在盒子内的坐标

```

<style>
    .box {width: 300px;height: 300px;background-color: pink;margin: 200px;}
</style>
</head>

```

```

<body>
  <div class="box"></div>
  <script>
    // 我们在盒子内点击，想要得到鼠标距离盒子左右的距离。
    // 首先得到鼠标在页面中的坐标 ( e.pageX, e.pageY )
    // 其次得到盒子在页面中的距离(box.offsetLeft, box.offsetTop)
    // 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
    var box = document.querySelector('.box');
    box.addEventListener('mousemove', function(e) {
      // 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;
    })
  </script>
</body>

```

模态框拖拽

弹出框，我们也称为模态框。

```

<style>
  .login-header {width: 100%;text-align: center;height: 30px;font-size: 24px;line-height: 30px;}
  ul,li,ol,dl,dt,dd,div,p,span,h1,h2,h3,h4,h5,h6,a {padding: 0px;margin: 0px;}
  .login {display: none;width: 512px;height: 280px;position: fixed;border: #ebebeb solid 1px;left: 50%;top: 50%;background: #ffffff;
    box-shadow: 0px 0px 20px #ddd;z-index: 9999;transform: translate(-50%, -50%);}
  .login-title {width: 100%;margin: 10px 0px 0px 0px;text-align: center;line-height: 40px;height: 40px;font-size: 18px;position: relative;cursor: move;}
  .login-input-content {margin-top: 20px;}
  .login-button {width: 50%;margin: 30px auto 0px auto;line-height: 40px;font-size: 14px;border: #ebebeb 1px solid;text-align: center;}
  .login-bg {display: none;width: 100%;height: 100%;position: fixed;top: 0px;left: 0px;background: rgba(0, 0, 0, .3);}
  a {text-decoration: none;color: #000000;}
  .login-button a { display: block;}
  .login-input input.list-input {float: left;line-height: 35px;height: 35px; width: 350px;border: #ebebeb 1px solid;
    text-indent: 5px;}
  .login-input { overflow: hidden;margin: 0px 0px 20px 0px;}
  .login-input label {float: left;width: 90px;padding-right: 10px;text-align: right;line-height: 35px;height: 35px;font-size: 14px;}
  .login-title span {position: absolute;font-size: 12px;right: -20px;top: -30px;background: #ffffff;border: #ebebeb solid 1px;width: 40px;height: 40px;border-radius: 20px;}
</style>
</head>
<body>
  <div class="login-header"><a id="link" href="javascript:;">登录</a></div>
  <div id="login" class="login">

```

```

        <div id="title" class="login-title">登录会员
            <span><a id="closeBtn" href="javascript:void(0);" class="close-login">关闭</a>
</span>
    </div>
    <div class="login-input-content">
        <div class="login-input">
            <label>用户名 : </label>
            <input type="text" placeholder="请输入用户名" name="info[username]" id="username"
class="list-input">
        </div>
        <div class="login-input">
            <label>登录密码 : </label>
            <input type="password" placeholder="请输入登录密码" name="info[password]"
id="password" class="list-input">
        </div>
        <div id="loginBtn" class="login-button"><a href="javascript:void(0);" id="login-button-
submit">登录会员</a></div>
    </div>
    <!-- 遮盖层 -->
    <div id="bg" class="login-bg"></div>
    <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
        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
            document.addEventListener('mousemove', move)
            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>

```

```

    })
  </script>
</body>

```

放大镜

```

<!-- 引入css 初始化的css 文件 -->
<link rel="stylesheet" href="css/base.css">
<!-- 引入公共样式的css 文件 -->
<link rel="stylesheet" href="css/common.css">
<!-- 引入详情页面的css文件 -->
<link rel="stylesheet" href="css/detail.css">
</head>
<body>
  <div class="preview_wrap fl">
    <div class="preview_img">
      
      <div class="mask" style="display: none; left: 98px; top: 98px;"></div>
      <div class="big" style="display: none;">
        
      </div>
    </div>
    <div class="preview_list">
      <a href="#" class="arrow_prev"></a>
      <a href="#" class="arrow_next"></a>
      <ul class="list_item">
        <li>
          
        </li>
        <li class="current">
          
        </li>
        <li>
          
        </li>
        <li>
          
        </li>
        <li>
          
        </li>
      </ul>
    </div>
  </div>
  <script>
    window.addEventListener('load', function() {
      var preview_img = document.querySelector('.preview_img');
      var mask = document.querySelector('.mask');
      var big = document.querySelector('.big');
      // 1. 当我们鼠标经过 preview_img 就显示和隐藏 mask 遮挡层 和 big 大盒子

      preview_img.addEventListener('mouseover', function() {

```

```

        mask.style.display = 'block';
        big.style.display = 'block';
    })
    preview_img.addEventListener('mouseout', function() {
        mask.style.display = 'none';
        big.style.display = 'none';
    })
    // 2. 鼠标移动的时候，让黄色的盒子跟着鼠标来走
    preview_img.addEventListener('mousemove', function(e) {
        // (1). 先计算出鼠标在盒子内的坐标
        var x = e.pageX - this.offsetLeft;
        var y = e.pageY - this.offsetTop;
        // console.log(x, y);
        // (2) 减去盒子高度 300的一半 是 150 就是我们mask 的最终 left 和top值了
        // (3) 我们mask 移动的距离
        var maskX = x - mask.offsetWidth / 2;
        var maskY = y - mask.offsetHeight / 2;
        // (4) 如果x 坐标小于了0 就让他停在0 的位置
        // 遮挡层的最大移动距离
        var maskMax = preview_img.offsetWidth - mask.offsetWidth;
        if (maskX <= 0) {
            maskX = 0;
        } else if (maskX >= maskMax) {
            maskX = maskMax;
        }
        if (maskY <= 0) {
            maskY = 0;
        } else if (maskY >= maskMax) {
            maskY = maskMax;
        }
        mask.style.left = maskX + 'px';
        mask.style.top = maskY + 'px';
        // 3. 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
        // 大图
        var bigImg = document.querySelector('.bigImg');
        // 大图片最大移动距离
        var bigMax = bigImg.offsetWidth - big.offsetWidth;
        // 大图片的移动距离 x y
        var bigX = maskX * bigMax / maskMax;
        var bigY = maskY * bigMax / maskMax;
        bigImg.style.left = -bigX + 'px';
        bigImg.style.top = -bigY + 'px';
    })
})
</script>

```

1.2. 元素可视区 client 系列

1.2.1 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 |



1.2.2. 淘宝 flexible.js 源码分析(移动端自适应方案)

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。

立即执行函数

```
<script>
// 1.立即执行函数：不需要调用，立马能够自己执行的函数
function fn() {

    console.log(1);
```

```
}
fn();
// 2. 写法 也可以传递参数进来
// 1.(function() {}){}() 或者 2. (function(){}());
(function(a, b) {
    console.log(a + b);
    var num = 10;
})(1, 2); // 第二个小括号可以看做是调用函数
(function sum(a, b) {
    console.log(a + b);
    var num = 10; // 局部变量
}(2, 3));
// 3. 立即执行函数最大的作用就是 独立创建了一个作用域，里面所有的变量都是局部变量 不会有命名冲突的情况
</script>
```

像素比和pageshow事件

```
<body>
  <script>
    // console.log(window.devicePixelRatio);
    window.addEventListener('pageshow', function() {
      alert(11);
    })
  </script>
  <a href="http://www.baidu.cn">链接</a>
</body>
```

1.3.元素滚动 scroll 系列

1.3.1. 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 |


```

<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;}
    span {display: none;position: absolute;bottom: 0;}
</style>
</head>
<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.offsetTop 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
        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';
            }
            // 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>

```

1.3.5.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.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
```

1.4. 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
- 4.注意页面滚动的距离通过 window.pageXOffset 获得

1.5. mouseenter 和mouseover的区别

- 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和mouseover的区别

```

<style>
  . {width: 300px;height: 300px;background-color: pink;margin: 100px auto;}
  .one {width: 200px;height: 200px;background-color: purple;}
</style>
</head>
<body>
  <div class="father">
    <div class="son"></div>
  </div>
  <script>
    var box = document.querySelector('.box');
    var one = document.querySelector('.one');
    box.addEventListener('mouseenter', function() {
      console.log(11);
    })
  </script>
</body>

```

1.6. 动画函数封装

1.6.1. 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

```

<style>
  div {
    position: absolute;
    left: 0;
    width: 100px;
    height: 100px;
    background-color: pink;
  }
</style>
</head>
<body>
  <div></div>
  <script>
    // 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';
}, 30);
</script>
</body>

```

简单动画函数封装

```

<style>
    div {position: absolute;left: 0;width: 100px;height: 100px;background-color: pink;}
    span {position: absolute;left: 0;top: 200px;display: block;width: 150px;height:
150px;background-color: purple;}
</style>
</head>
<body>
    <div></div>
    <span>北京</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>
</body>

```

1.6.2. 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。

动画函数给不同元素记录不同定时器

```
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);
    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);
}
```

1.1. 动画函数封装

1.1.1 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

1.1.2 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

1.1.3 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。

1.1.4 动画

缓动动画原理

```
<style>
    div {position: absolute;left: 0;width: 100px;height: 100px;background-color: pink;}

    span {position: absolute;left: 0;top: 200px;display: block;width: 150px;height:
```

```

150px;background-color: purple;}
</style>
</head>
<body>
  <button>点击</button>
  <span></span>
  <script>
    // 缓动动画函数封装obj目标对象 target 目标位置
    // 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
    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
        var step = (target - obj.offsetLeft) / 10;
        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);
    }
    var span = document.querySelector('span');
    var btn = document.querySelector('button');
    btn.addEventListener('click', function() {
      // 调用函数
      animate(span, 500);
    })
    // 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
  </script>
</body>

```

缓动动画多个目标值之间移动

```

<style>
  div {position: absolute;left: 0;width: 100px;height: 100px;background-color: pink;}
  span {position: absolute;left: 0;top: 200px;display: block;width: 150px;height:
150px;background-color: purple;}
</style>
</head>
<body>
  <button class="btn500">点击500</button>
  <button class="btn800">点击800</button>
  <span></span>
  <script>
    // 缓动动画函数封装obj目标对象 target 目标位置
    // 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
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;
        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);
}

var span = document.querySelector('span');
var btn500 = document.querySelector('.btn500');
var btn800 = document.querySelector('.btn800');
btn500.addEventListener('click', function() {
    // 调用函数
    animate(span, 500);
});
btn800.addEventListener('click', function() {
    // 调用函数
    animate(span, 800);
});
// 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
</script>
</body>

```

缓动动画添加回调函数

```

<style>
    div {position: absolute;left: 0;width: 100px;height: 100px;background-color: pink;}
    span {position: absolute;left: 0;top: 200px;display: block;width: 150px;height:
150px;background-color: purple;}
</style>
</head>
<body>
    <button class="btn500">点击到500</button>
    <button class="btn800">点击到800</button>
    <span></span>
    <script>
        // 缓动动画函数封装obj目标对象 target 目标位置
        // 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
    </script>

```



```

function animate(obj, target, callback) {
    // console.log(callback); callback = function() {} 调用的时候 callback()
    // 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;
        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);
            // 回调函数写到定时器结束里面
            if (callback) {
                // 调用函数
                callback();
            }
        }
        // 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);
}

var span = document.querySelector('span');
var btn500 = document.querySelector('.btn500');
var btn800 = document.querySelector('.btn800');
btn500.addEventListener('click', function() {
    // 调用函数
    animate(span, 500);
})
btn800.addEventListener('click', function() {
    // 调用函数
    animate(span, 800, function() {
        // alert('你好吗');
        span.style.backgroundColor = 'red';
    });
})
// 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
</script>
</body>

```

```

function animate(obj, target, callback) {
    // console.log(callback); callback = function() {} 调用的时候 callback()
    // 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;
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);
    // 回调函数写到定时器结束里面
    // if (callback) {
    //     // 调用函数
    //     callback();
    // }
    callback && callback();
}
// 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);
}

```

引用animate动画函数

```

<style>
    .sliderbar {position: fixed;right: 0;bottom: 100px;width: 40px;height: 40px;text-align:
center;line-height: 40px;
        cursor: pointer;color: #fff;}
    .con {position: absolute;left: 0;top: 0;width: 200px;height: 40px;background-color:
purple;z-index: -1;}
</style>
<script src="animate.js"></script>
</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>
```

1.2. 常见网页特效案例

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。

```
window.addEventListener('load', function() {
    // 1. 获取元素
    var arrow_l = document.querySelector('.arrow-l');
    var arrow_r = document.querySelector('.arrow-r');
    var focus = document.querySelector('.focus');
    var focusWidth = focus.offsetWidth;
    // 2. 鼠标经过focus 就显示隐藏左右按钮
    focus.addEventListener('mouseenter', function() {
        arrow_l.style.display = 'block';
        arrow_r.style.display = 'block';
        clearInterval(timer);
        timer = null; // 清除定时器变量
    });
    focus.addEventListener('mouseleave', function() {
        arrow_l.style.display = 'none';
        arrow_r.style.display = 'none';
        timer = setInterval(function() {
            // 手动调用点击事件
            arrow_r.click();
        }, 2000);
    });
    // 3. 动态生成小圆圈 有几张图片，我就生成几个小圆圈
    var ul = focus.querySelector('ul');
    var ol = focus.querySelector('.circle');
    // console.log(ul.children.length);
    for (var i = 0; i < ul.children.length; i++) {
        // 创建一个小li
        var li = document.createElement('li');
        // 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
        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
    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++) {
            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(focusWidth);
        console.log(index);
        animate(ul, -index * focusWidth);
    })
}
// 把ol里面的第一个小li设置类名为 current
ol.children[0].className = 'current';
// 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;
// circle 控制小圆圈的播放
var circle = 0;
// flag 节流阀
var flag = true;
arrow_r.addEventListener('click', function() {
    if (flag) {
        flag = false; // 关闭节流阀
        // 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, function() {
            flag = true; // 打开节流阀
        });
        // 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;
        }
        // 调用函数
        circleChange();
    }
});

// 9. 左侧按钮做法
arrow_l.addEventListener('click', function() {
    if (flag) {
        flag = false;
        if (num == 0) {
            num = ul.children.length - 1;
            ul.style.left = -num * focusWidth + 'px';
        }
    }
});

```

```

        num--;
        animate(ul, -num * focusWidth, function() {
            flag = true;
        });
        // 点击左侧按钮，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 说明第一张图片，则小圆圈要改为第4个小圆圈（3）
        // if (circle < 0) {
        //     circle = ol.children.length - 1;
        // }
        circle = circle < 0 ? ol.children.length - 1 : circle;
        // 调用函数
        circleChange();
    }
});
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 = '';
    }
    // 留下当前的小圆圈的current类名
    ol.children[circle].className = 'current';
}
// 10. 自动播放轮播图
var timer = setInterval(function() {
    //手动调用点击事件
    arrow_r.click();
}, 2000);
})

```

1.2.2. 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

返回顶部

1. 带有动画的返回顶部
2. 此时可以继续使用我们封装的动画函数
3. 只需要把所有的left 相关的值改为 跟 页面垂直滚动距离相关就可以了
4. 页面滚动了多少，可以通过 window.pageYOffset 得到
5. 最后是页面滚动，使用 window.scroll(x,y)

```

//1. 获取元素
var sliderbar = document.querySelector('.slider-bar');

```

```

var banner = document.querySelector('.banner');
// banner.offsetTop 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
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';
    }
    // 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';
    }
})
// 3. 当我们点击了返回顶部模块, 就让窗口滚动的页面的最上方
goBack.addEventListener('click', function() {
    // 里面的x和y 不跟单位的 直接写数字即可
    // window.scrollTo(0, 0);
    // 因为是窗口滚动 所以对象是window
    animate(window, 0);
});

```

筋斗云案例

1. 利用动画函数做动画效果
2. 原先筋斗云的起始位置是0
3. 鼠标经过某个小li, 把当前小li的offsetLeft 位置做为目标值即可
4. 鼠标离开某个小li, 就把目标值设为 0
5. 如果点击了某个小li, 就把li当前的位置存储起来, 做为筋斗云的起始位置

```

window.addEventListener('load', function() {
    // 1. 获取元素
    var cloud = document.querySelector('.cloud');
    var c_nav = document.querySelector('.c-nav');
    var lis = c_nav.querySelectorAll('li');
    // 2. 给所有的小li绑定事件

    // 这个current 做为筋斗云的起始位置

```

```
var current = 0;
for (var i = 0; i < lis.length; i++) {
    // (1) 鼠标经过把当前小li 的位置做为目标值
    lis[i].addEventListener('mouseenter', function() {
        animate(cloud, this.offsetLeft);
    });
    // (2) 鼠标离开就回到起始的位置
    lis[i].addEventListener('mouseleave', function() {
        animate(cloud, current);
    });
    // (3) 当我们鼠标点击, 就把当前位置做为目标值
    lis[i].addEventListener('click', function() {
        current = this.offsetLeft;
    });
}
})
```