# 一、异型滚动

异型滚动就是没有火车的概念，每个“车厢”都是单独的个体，点击按钮之后：下一个图片变为上一个图片的状态，包括宽度、高度、位置都会进行轮替。也可以叫做“样式轮替轮播”。上图中的轮播看似是5张图片，实际上需要7张图片。因为左右各有小图片当做猫腻。

jQuery的animate是不能把一个left，动画为right定位的。比如一个div天生left:100px;现在你让他动画到right:10px。jQuery无能为力！因为一个是left属性一个是right属性。

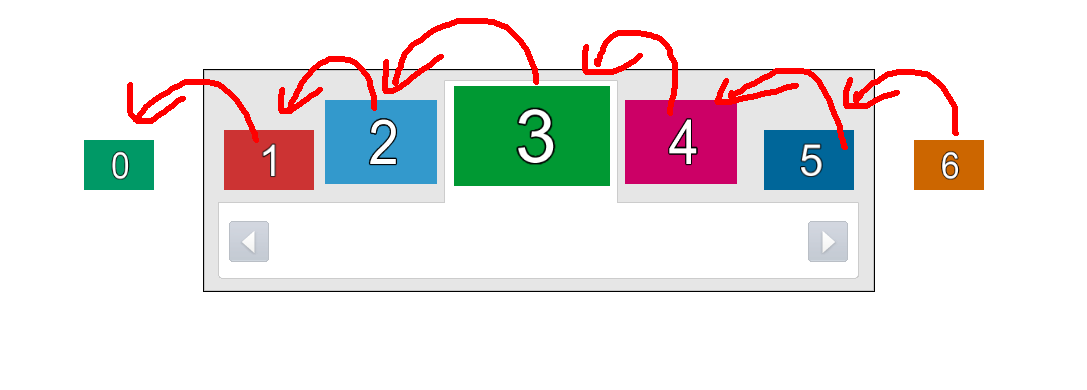
原理就是3号元素变为2号元素的样子，left、top、width、height四个属性都变。

2号变成1号……

1号变成0号……

0号变成6号……

|  |
| --- |
| $(".no1").animate({"width": 70,"height": 50,"left":-120,"top":70}); |



## 1.1 普通效果

我们不需要自己去罗列所有JSON，让程序自己去寻找所有JSON即可，放入数组：

|  |
| --- |
| 1. // 将所有元素状态保存在json数组中 2. // 一般在书写时常量使用大写字母 3. var JSONARR = []; 4. // 使用遍历将7个li状态保存在json 5. // 保存li，width height left top 6. for(var i = 0 ; i <= 6 ; i ++){ 7. JSONARR.push({ 8. "width" : $(".no" + i).css("width"), 9. "height" : $(".no" + i).css("height"), 10. "left" : $(".no" + i).css("left"), 11. "top" : $(".no" + i).css("top"), 12. }); 13. } |

此时noX就能运动到X-1的位置：

|  |
| --- |
| 1. // 右按钮点击事件 2. $("#rightBtn").click(function(){ 3. // 将noX移动到 noX-1状态 4. $(".no1").animate(JSONARR[0],400); 5. $(".no2").animate(JSONARR[1],400); 6. $(".no3").animate(JSONARR[2],400); 7. $(".no4").animate(JSONARR[3],400); 8. $(".no5").animate(JSONARR[4],400); 9. $(".no6").animate(JSONARR[5],400); 10. // no0瞬移到no6 11. $(".no0").css(JSONARR[6]); 12. }); |

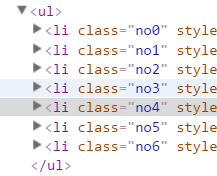
我们发现移动之后，每个li的类名和自己所处的位置不统一了：

我们现在就要换类名。不仅仅是为了类名统一，而且有一个意想不到的作用。

那我们再回到我们的程序，点击按钮之后，命令no3去JSONARR[2]的位置，命令no4去JSONARR[3]的位置……

但是我们换好位置之后，类名进行交换！no3去了JSONARR[2]，然后no3就改为no2。no4去了JSONARR[3]位置了，就改名为no3。 再次点击按钮，还是命令no3去JSONARR[2]的位置，命令no4去JSONARR[3] ，此时那个人已经变了！！

类的轮替：



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 点击按钮 | 点击按钮 | 点击按钮 | 点击按钮 | 点击按钮 | 点击按钮 | 点击按钮 |
| no0  no1  no2  no3  no4  no5  no6 | no6  no0  no1  no2  no3  no4  no5 | no5  no6  no0  no1  no2  no3  no4 | no4  no5  no6  no0  no1  no2  no3 | no3  no4  no5  no6  no0  no1  no2 | no2  no3  no4  no5  no6  no0  no1 | no1  no2  no3  no4  no5  no6  no0 |

我们借助数组实现类名轮替非常方便

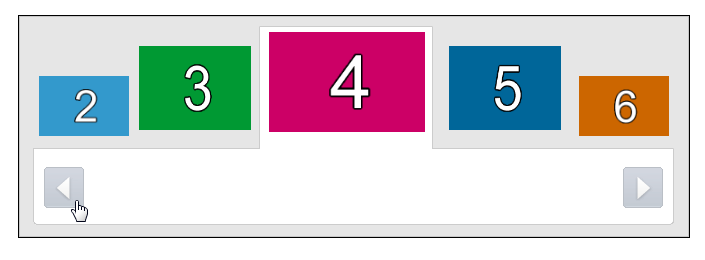
|  |
| --- |
| 1. classNameArr.unshift(classNameArr.pop()); |

数组变了之后没有用，必须把数组上的元素引用在DOM身上！就要写for循环：

|  |
| --- |
| 1. // 将类名体现在DOM元素身上 2. for(var i = 0 ; i <= 6 ; i ++){ 3. **$("li").eq(i).attr("class",classNameARR[i]);** 4. } |

左按钮点击事件：

|  |
| --- |
| 1. // 左按钮点击事件 2. $("#leftBtn").click(function(){ 3. // 防流氓 4. if($("li").is(":animated")){ 5. return; 6. } 7. // .noX 移动到 noX + 1状态 8. $(".no0").animate(JSONARR[1],400); 9. $(".no1").animate(JSONARR[2],400); 10. $(".no2").animate(JSONARR[3],400); 11. $(".no3").animate(JSONARR[4],400); 12. $(".no4").animate(JSONARR[5],400); 13. $(".no5").animate(JSONARR[6],400); 14. // .no6瞬移 15. $(".no6").css(JSONARR[0]); 16. // 将类名数组进行轮换 17. // 删除数组第一项添加到数组最后 18. classNameARR.push(classNameARR.shift()); 20. // 体现li身上 21. for(var i = 0 ; i <= 6 ; i ++){ 22. $("li").eq(i).attr("class",classNameARR[i]); 23. } 24. }); |



# 函数节流

函数节流：对一些频繁触发的事件行为进行限制，可以使函数在一定的时间内或者是满足一定的条件之后再去执行

定义锁 取值是布尔值

当lock = true 的时候函数可以执行

当lock = false 的时候函数不可以执行

举例：

|  |
| --- |
| 1. // 定义锁 2. var lock = true; 3. // 定义函数 4. function fun() { 5. if (!lock) { 6. return; 7. } 8. // 把锁关闭 9. lock = false; 10. console.log(123); 11. } |

举例：

|  |
| --- |
| 1. <button id="btn">点我执行</button> 2. <script type="text/javascript" src="js/jquery-1.12.3.min.js"></script> 3. <script type="text/javascript"> 4. // 定义锁 5. var lock = true; 6. // 给btn添加点击事件 7. $("#btn").click(function() { 8. // 函数节流 9. if (!lock) { 10. return; 11. } 12. // 把锁关闭 13. lock = false; 14. console.log("函数执行了"); 15. }) 16. </script> |

# 延时器

延时器就好比是一个定时炸弹， 在一定的时间内只会执行一次

使用方式：

setTimeout(fn, time)

fn: 执行的函数

time: 延迟的时间

举例：

|  |
| --- |
| 1. // 定义锁 2. var lock = true; 3. // 给btn添加点击事件 4. $("#btn").click(function() { 5. // 函数节流 6. if (!lock) { 7. return; 8. } 9. // 把锁关闭 10. lock = false; 11. // 开启一个延时器 12. window.setTimeout(function() { 13. // 开锁 14. lock = true; 15. console.log("延迟结束"); 16. }, 3000) 17. console.log("函数执行了"); 18. }) |

结果：

3s之后可以再次执行：

|  |
| --- |
|  |

# this

this在js中指向触发事件的对象

each方法接收一个参数是函数， 函数中有一个参数i, 函数中也有一个this， 指向遍历到的该对象

普通函数里面也有一个this， 当函数自执行的时候，里面的this指向window

举例：

|  |
| --- |
| 1. // 定义一个函数 2. function fun() { 3. console.log(this); 4. } |

执行：

|  |
| --- |
| 1. // 函数自执行 2. fun(); |

结果：

|  |
| --- |
|  |

可以通过call方法改变函数中的this指向， call方法中的第一个参数就是要改变的this指向

举例：

|  |
| --- |
| 1. // 定义一个函数 2. function fun() { 3. console.log(this); 4. } 5. // 使用call方法改变函数的this指向 6. fun.call($("#btn")); 7. fun.call(document); 8. fun.call(document.body); |

结果：

|  |
| --- |
|  |

# each方法

each方法的使用方式有两种：

第一种使用方式：

对象.each(fn)

fn: 执行的函数

有两个参数：

第一个参数： 索引值

第二个参数： 成员值

作用域遍历到的该对象

DOM结构：

|  |
| --- |
| 1. <div>1</div> 2. <div>2</div> 3. <div>3</div> 4. <div>4</div> 5. <div>5</div> |

执行代码:

|  |
| --- |
| 1. // 第一种使用方式 2. $("div").each(function() { 3. console.log(this, arguments); 4. }) |

输出结果：

|  |
| --- |
|  |

第二种使用方式：

$.each(target, fn)

target: 要遍历的目标， 可以是数组， 可以是对象

fn: 要执行的函数

函数中有两个参数：

第一个参数： 索引值

第二个参数： 成员值

作用域遍历到该对象

举例：

|  |
| --- |
| 1. // 定义一个数组 2. var arr = ["red", "blue", "green", "pink"]; 3. // 使用第二种方式 4. $.each(arr, function() { 5. console.log(arguments); 6. }) |

结果：

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |