# 通过案例学JavaScript

# 用JavaScript实现动画效果

## 案例：移动页面上的一个元素

### Step 1：在页面上对一个元素进行初始定位，然后移动这个元素

如下，假设要对页面上的元素`message`进行移动，首先用js的方式给这个元素一个初始位置，然后再通过js移动这个元素到目标位置：

```
<body>
    <div id="message">Whee!</div>
    <script type="text/javascript">
        // 页面加载完成的时候执行某个函数
        function addLoadEvent(func){
            var oldonload=window.onload;
            if(typeof window.onload != 'function'){
                window.onload=func;
            } else {
                window.onload=function(){
                    oldonload();
                    func();
                }
            }
        }
        //设置元素的位置
        function positionMessage(){
            // 浏览器DOM支持能力检测
            if(!document.getElementById){
                return false;
            }
            // 判断页面中是否存在需要的这个message元素
            if(!document.getElementById('message')){
                return false;
            }
            // 获取message元素并设置它的位置
            var elem=document.getElementById('message');
            elem.style.position='absolute';
            elem.style.left='50px',
            elem.style.top='100px';
        }
        //移动元素到需要的目标位置
        function moveMessage(){
            // 浏览器DOM支持能力检测与message元素是否存在一起判断
            if (!document.getElementById || !document.getElementByI
d('message')) {
                return false;
            }
            // 获取message元素并设置它的新位置
            var elem=document.getElementById('message');
            elem.style.left='200px'
        }
        // 页面加载完成的时候先用positionMessage函数设置初始位置，然后用mov
eMessage移动到目标位置
        addLoadEvent(positionMessage);
        addLoadEvent(moveMessage);
    </script>
```

```
</body>
```

在浏览器里面预览上面的效果我们可以发现，元素`message`从初始位置到目标位置会即刻发生改变，中间没有一个时间间隔，完全看不到变化的效果。我们需要`message`元素在经过一段时间之后才移动到目标位置。

## Step 2：让元素在经过一定时间之后移动到目标位置

JavaScript中有一个能让某个函数在经过一段时间之后才开始执行的函数`setTimeout("functionName",interval)`，这个函数的第一个参数是一个字符串，表示将要执行的那个函数的名字，第二个参数是一个以毫秒为单位的数值，表示要经过多少毫秒才执行第一个参数代表的函数。通过调用`setTimeout("functionName",interval)`，将要执行的函数就会进入到一个执行队列，如果需要取消这个正在排队等待执行的函数，先需要将`setTimeout("functionName",interval)`赋值个一个变量，即`var variable=setTimeout("functionName",interval)`，然后调用`clearTimeout(varibale)`来取消。

现在我们在***Step 1***基础上修改下`positionMessage`函数，让它在5秒之后去调用`moveMessage`函数来移动`message`元素，而不用`addLoadEvent`来执行`moveMessage`了：

```
//设置元素的位置
function positionMessage(){
    // 浏览器DOM支持能力检测
    if(!document.getElementById){
        return false;
    }
    // 判断页面中是否存在需要的这个message元素
    if(!document.getElementById('message')){
        return false;
    }
    // 获取message元素并设置它的位置
    var elem=document.getElementById('message');
    elem.style.position='absolute';
    elem.style.left='50px',
    elem.style.top='100px';
    // 500毫秒之后执行moveMessage函数移动元素
    movement=setTimeout("moveMessage()",500);
}
```

然后刷新页面，我们就能看到`message`元素从初始位置移动到新位置了，但是它是从初始位置跳动到新位置的，而不是一步一步移动到新位置的。要实现一步步移动到新位置我们需要这样考虑：

- 取得message元素的当前位置

- 如果当前位置距离目标位置还有一定距离，就移动到距离目标位置进一点的地方
- 如果移动到了目标位置就不用继续移动了
- 每隔一定时间重复以上动作

根据这个思路我们改造一下`moveMessage`函数，如下：

```
//移动元素到需要的目标位置
function moveMessage(){
    // 浏览器DOM支持能力检测与message元素是否存在一起判断
    if (!document.getElementById || !document.getElementById('messa
ge')) {
        return false;
    }
    // 获取message元素并设置它的新位置
    var elem=document.getElementById('message');
    // elem.style.left='200px'
    // 获取message元素当前的left值，并转换为纯数字，以便进行数学计算
    var xpos=parseInt(elem.style.left);
    // 获取message元素当前的top值，并转换为纯数字，以便进行数学计算
    var ypos=parseInt(elem.style.top);
    // 将message元素的当前left、top值与目标位置进行比较，如果刚好相等就不移动
了，
    // 如果距离目标元素还有一定距离就通过计算缩短一些这个距离
    if(xpos==200 && ypos==100){
        return true;
    }
    if(xpos<200){
        xpos++;
    }
    if(xpos>200){
        xpos--;
    }
    if(ypos<100){
        ypos++
    }
    if(ypos>100){
        ypos--;
    }
    // 设置message元素经过移动的新位置
    elem.style.left=xpos+'px';
    elem.style.top=ypos+'px';
    // 每隔10毫秒移动一次
    movement=setTimeout("moveMessage()",10);
}
```

这个时候再刷新下浏览器可以发现元素一步步移动到了目标位置。

> **提示：** `parseInt(str)` 这个函数能够从一个以数字开头的字符串中把数字提取出来，以上的函数中我们需要用到一些数学计算，只有数值才能参与数学计算，所以我们要从 `10px` 这种样式中提取出数字参与计算。

## Step 3：抽象出一个移动元素的函数moveElement

以上的`moveMessage`函数完成了一项特定的任务，把**特定的元素*message***经过**特定的时间**移动到了一个**特定的位置**，如果把这些特定的值改为变量通过参数传递，这个函数就会变得非常灵活和通用了。如下：

```
//设置元素的位置
function positionMessage(){
    // 浏览器DOM支持能力检测
    if(!document.getElementById){
        return false;
    }
    // 判断页面中是否存在需要的这个message元素
    if(!document.getElementById('message')){
        return false;
    }
    // 获取message元素并设置它的位置
    var elem=document.getElementById('message');
    elem.style.position='absolute';
    elem.style.left='50px',
    elem.style.top='100px';
    // 500毫秒之后执行moveMessage函数移动元素
    // movement=setTimeout("moveMessage()",500);
    // 调用moveElement
    moveElement('message',200,100,10);
}
//移动元素到需要的目标位置
function moveElement(elementID,final_x,final_y,interval){
    // 浏览器DOM支持能力检测与元素是否存在一起判断
    if (!document.getElementById || !document.getElementById(elemen
tID)) {
        return false;
    }
    // 获取元素并设置它的新位置
    var elem=document.getElementById(elementID);
    // elem.style.left='200px'
    // 获取元素当前的left值，并转换为纯数字，以便进行数学计算
    var xpos=parseInt(elem.style.left);
    // 获取元素当前的top值，并转换为纯数字，以便进行数学计算
    var ypos=parseInt(elem.style.top);
    // 将元素的当前left、top值与目标位置进行比较，如果刚好相等就不移动了，
    // 如果距离目标元素还有一定距离就通过计算缩短一些这个距离
    if(xpos==final_x && ypos==final_y){
        return true;
    }
    if(xpos<final_x){
        xpos++;
    }
    if(xpos>final_x){
        xpos--;
    }
    if(ypos<final_y){
        ypos++
    }
```

```
    if(ypos>final_y){
        ypos--;
    }
    // 设置元素经过移动的新位置
    elem.style.left=xpos+'px';
    elem.style.top=ypos+'px';
    // 每隔10毫秒移动一次
    // movement=setTimeout("moveMessage()",10);
    var repeat="moveElement('"+elementID+"',"+final_x+","+fina
l_y+","+interval+")";
    movement=setTimeout(repeat,interval);
}
```

抽象出了移动函数`moveElement`我们在调用它的是时候就能传入任意参数，这样元素就能自由的移动。

## Step 4：moveElement函数的优化

前面moveElement函数在获取元素当前位置的时候是这样做的：

```
// 获取元素当前的left值，并转换为纯数字，以便进行数学计算
var xpos=parseInt(elem.style.left);
// 获取元素当前的top值，并转换为纯数字，以便进行数学计算
var ypos=parseInt(elem.style.top);
```

这里存在一个假设，elem元素一定有left、top样式属性，但如果之前没有设置过元素的left、top值，那么函数运行至此就会终止，所以我们可以在获取left、top值之前做一个判断，如果没有设置元素的left、top值，就给它一个默认值，如下：

```
// 如果元素不存在left值，默认为0
if(!elem.style.left){
    elem.style.left='0px';
}
// 如果元素不存在top值，默认为0
if(!elem.style.top){
    elem.style.top='0px';
}
// elem.style.left='200px'
// 获取元素当前的left值，并转换为纯数字，以便进行数学计算
var xpos=parseInt(elem.style.left);
// 获取元素当前的top值，并转换为纯数字，以便进行数学计算
var ypos=parseInt(elem.style.top);
```

另外，`moveElement`函数中，移动元素我们是通过`xpos++`或`xpos--`这种方式来移动的，每次移动1px，并且速度是固定的，如果一个元素的移动距离较长，那这个动画执行完需要的时间就比较长，而且移动的速度固定不变动画就显得生硬、不自然，可以试着改变下移动的速度，比如如果元素距离目标位置较远那就每次移动距离长一些，距离目标位置较近就移动距离短一点，这样元素从初始位置移动到目标位置的速度就由快到慢变化，动画就更自然一些。按照这种思路来改造下，比如让元素每次移动距离为当前位置到终点位置的十分之一，距离越远每次移动距离越长，距离越近每次移动距离越短，如下：

```
//移动元素到需要的目标位置
function moveElement(elementID,final_x,final_y,interval){
    // 浏览器DOM支持能力检测与元素是否存在一起判断
    if (!document.getElementById || !document.getElementById(elemen
tID)) {
        return false;
    }
    // 获取元素并设置它的新位置
    var elem=document.getElementById(elementID);
    // 如果元素不存在left值，默认为0
    if(!elem.style.left){
        elem.style.left='0px';
    }
    // 如果元素不存在top值，默认为0
    if(!elem.style.top){
        elem.style.top='0px';
    }
    // elem.style.left='200px'
    // 获取元素当前的left值，并转换为纯数字，以便进行数学计算
    var xpos=parseInt(elem.style.left);
    // 获取元素当前的top值，并转换为纯数字，以便进行数学计算
    var ypos=parseInt(elem.style.top);
    // 设置一个变量记录元素每次需要移动的距离
    var dist=0;
    // 将元素的当前left、top值与目标位置进行比较，如果刚好相等就不移动了，
    // 如果距离目标元素还有一定距离就通过计算缩短一些这个距离
    if(xpos==final_x && ypos==final_y){
        return true;
    }
    if(xpos<final_x){
        // xpos++;
        dist=Math.ceil((final_x-xpos)/10);
        xpos=xpos+dist;
    }
    if(xpos>final_x){
        // xpos--;
        dist=Math.ceil((xpos-final_x)/10);
        xpos=xpos-dist;
    }
    if(ypos<final_y){
        // ypos++
        dist=Math.ceil((final_y-ypos)/10);
        ypos=ypos+dist;
    }
    if(ypos>final_y){
        // ypos--;
        dist=Math.ceil((ypos-fina_y)/10);
        ypos=ypos-dist;
```

```
    }
    // 设置元素经过移动的新位置
    elem.style.left=xpos+'px';
    elem.style.top=ypos+'px';
    // 每隔10毫秒移动一次
    // movement=setTimeout("moveMessage()",10);
    var repeat="moveElement('"+elementID+"',"+final_x+","+fina
l_y+","+interval+")";
    movement=setTimeout(repeat,interval);
}
```

此时再刷新页面，发现元素从初始位置到终点位置的移动过程就是由快变慢，效果就自然一些了。

> **提示：** `moveElement` 这个函数中元素每次移动距离为当前位置到终点位置距离的十分之一，这样就有可能会产生小数，另外如果距离小于10的时候不能被10整除，移动距离就小于1，而我们电脑屏幕最小是1px，所以用 `Math.ceil(number)` 这个方法来对参数 `number` 进行向上取整，如： `Math.ceil(0.2)` 向上取整结果等于1。除此之外还有 `Math.floor(number)` 向下取整，如： `Math.floor(0.2)` 向上取整结果等于0。四舍五入取整 `Math.round(number)` ，如 `Math.round(0.2)` 结果为0， `Math.round(0.6)` 结果为1。