JS中的事件及事件代理

- 什么是事件?
  - 元素天生具备的行为事件都有哪些?
  - 事件绑定
- 事件对象
  - 鼠标事件对象
  - 常用属性兼容处理
  - 键盘事件对象
  - 小例子：禁止按F5刷新
  - 小例子：推盒子
  - A标签的默认行为
- 事件传播
  - 捕获、目标、冒泡
  - 冒泡传播
  - 哪些事件行为没有传播机制
  - mouseenter和mouseover的区别?
  - 小案例：京东放大镜

JS中的事件及事件代理

什么是事件?

```
1. document.body.onclick=functio
   n(){
2.      ...
3. }
```

如果上面的代码不写,BODY也具备点击事件,当我们拿鼠标点击BODY的时候,同样会触发它的CLICK行为；所以事件不是由我们的JS代码所创建的，而是元素天生就有的；

> 事件：元素天生就具备的行为，当我们去操作某个行为的时候，元素的相关事件就会被触发

元素天生具备的行为事件都有哪些?

PC端

1. click：点击事件
2. dblclick：双击
3. mouseover：鼠标滑入
4. mouseout：鼠标滑出
5. mouseenter：鼠标进入
6. mouseleave：鼠标离开
7. mousemove：鼠标移动
8. mousedown：鼠标左键按下
9. mouseup：鼠标左键抬起
10. mousewheel：鼠标滚轮滚动
11.
12. keydown：键盘按下
13. keypress：键盘长按
14. keyup：键盘抬起
15.
16. scroll：滚动条滚动
17. load：加载成功 (window.onload:当页面中的资源文件都加载完成，触发执行这个事件)
18. error：加载失败
19. resize：大小改变 (window.onresize:当浏览器的窗口大小发生了改变，触发执行这个事件)
20.

```
21.  focus：表单获取焦点
22.  blur：表单失去焦点
23.  change：表单内容改变
24.
25.  ...
26.  在元素对象的私有属性上 onxxx 的属性
     一般都是它的事件属性(很多很多)
27.  console.dir(document.body);
```

移动端

1. PC端的鼠标事件在移动端都不是很实用,因为手机上不可能拿鼠标操作,我们都需要通过手指来操作
2. -> click在移动端也能用，但是有300MS延迟的问题(当我们点击一下,浏览器需要在300MS后在触发事件执行：因为它要去看一下，在这个时间内是否触发了第二次点击，如果触发了，它认为CLICK没触发，dblclick触发了；移动端认为click是单击事件而不是点击)
3.
4. 移动端的事件是手指为主的事件
5. ->单手指操作事件
6. touchstart：手指按到屏幕上
7. touchmove：手指在屏幕上移动
8. touchend：手指离开屏幕
9. touchcancel：手指操作取消(一般都是意外状况导致操作取消)
10.
11. ->多手指操作事件
12. gesturestart：多手指按下
13. gesturechange：手指改变位置
14. gestureend：手指离开
15.
16.

```
16.  我们平时的点击、双击、长按、滑动、拖拽、旋转、缩放等操作，都是基于上述事件模拟出来的效果
17.
18.  移动端的键盘和PC也不一样，手机上一般都是虚拟键盘
19.  ->移动端的 keyup/keydown/keypress 等键盘事件，大部分手机都不兼容，如果需要监听表单内容的改变，我们需要使用 input 事件
```

特殊(新版ES标准中增加的)

```
1. dragstart：拖拽开始
2. drag：拖动中
3. dragend：拖拽结束
4. ...
```

事件绑定

我们之前写的JS代码

`xxx.onclick=function...`这样的操作，不是让其拥有事件行为(行为是天生自带的)，而是给某个行为绑定一个方法，当行为触发的时候，浏览器会自动的把绑定的方法执行，这样我们就可以控制在触发某个行为的时候，实现一些需求或者功能

```
1. document.body.onclick = functi
   on (e) {
2.     //->e:设置一个形参(可以随便起
   名字,我们一般都用e/ev来命名)接收当
   点击行为触发执行方法的时候,浏览器传
   递个方法的那个实参 <-> argument
   s[0]也可以获取到这个结果
3. }
4.
5. //->给BODY的点击事件行为绑定了一个
   方法,当触发这个行为的时候,浏览器会把
   绑定的方法执行
6. //=>重点：不仅仅把方法执行，而且还给
   方法传递了一个实参值，我们把这个值称
   之为“事件对象”
```

事件对象

> 当元素的某个行为被触发，浏览器会把对应绑定的方法执行，而且会给方法传递一个实参值，这个实参就是“事件对象”
>
> 标准浏览器是这样的机制，但是在IE6~8浏览器中，浏览器执行绑定的方法时候，没有给传递‘事件对象’这个实参值，如果我们想获取事件对象，我们使用 `window.event`

鼠标事件对象

> 标准浏览器：MouseEvent
> IE浏览器：PointerEvent
>
> 实例(私有属性) - MouseEvent.prototype - UIEvent.prototype - Event.prototype - Object.prototype

`clientX / clientY`

> 当前鼠标操作这一点距离浏览器窗口左上角的X轴和Y轴的坐标值 (兼容)

`pageX / pageY`

> 当前鼠标操作这一点距离BODY(浏览器第一屏幕)左上角的X轴和Y轴的坐标
>
> IE6~8中没有这个属性:
> pageX = clientX + 浏览器的scrollLeft
> pageY = clientY + 浏览器的scrollTop

`type`

> 当前操作的行为类型 ‘click’... (兼容)

`target`

> 当前操作的事件源(当前鼠标是在哪个元素上触发的,这个元素就是事件源)
>
> IE6~8中没有这个属性，想要获取事件源需要使用 `srcElement` 这个属性

`preventDefault()`

> 阻止当前元素的默认行为
>
> 默认行为:
> A的默认行为有点击跳转页面的效果
> INPUT的默认行为:当在文本框中输入内容的时候,内容填入到文本框中
>
> 所有浏览器天生给它规定的行为都是默认行为
>
> IE6~8下没有这个属性方法，我们需要使用 `returnValue=false` 来阻止默认行为

`stopPropagation()`

> 阻止事件的冒泡传播
>
> IE6~8下不兼容，如果想要实现这个操作，需要使用 `cancelBubble=true`

……
以上仅仅是个人认为的一些常用的属性和方法，其它更多的属性方法，私下里可以逐一的查找和学习

常用属性兼容处理

> 如果当前的浏览器是IE6~8,我们把所有不兼容的属性处理兼容了,以后再使用的时候,我们只需按照标准的属性和方法执行即可

```
document.body.onclick = function (e) {
    //->IE6~8:让低版本浏览器仿照标准浏览器,把常用的事件对象属性和方法处理兼容了,这样以后按照标准浏览器的属性和方法使用即可
    if (typeof e === 'undefined') {
        e = window.event;
        e.target = e.srcElement;
        e.pageX = e.clientX + (document.documentElement.scrollLeft || document.body.scrollLeft);
        e.pageY = e.clientY + (document.documentElement.scrollTop || document.body.scrollTop);
        e.preventDefault = function () {
            e.returnValue = false;
        };
```

```
            e.stopPropagation = function () {
                e.cancelBubble = true;
            };
        }
        //->以后按照标准浏览器的属性来用即可
        ...
}

//->JQ中的事件对象是JQ已经处理好兼容的,我们直接按照标准的方式来用即可(原理和上面的相同)
$('body').on('click',function(e){
    e.preventDafault();//->直接按照标准浏览器的属性使用即可(JQ完成了兼容的处理)
});
```

键盘事件对象

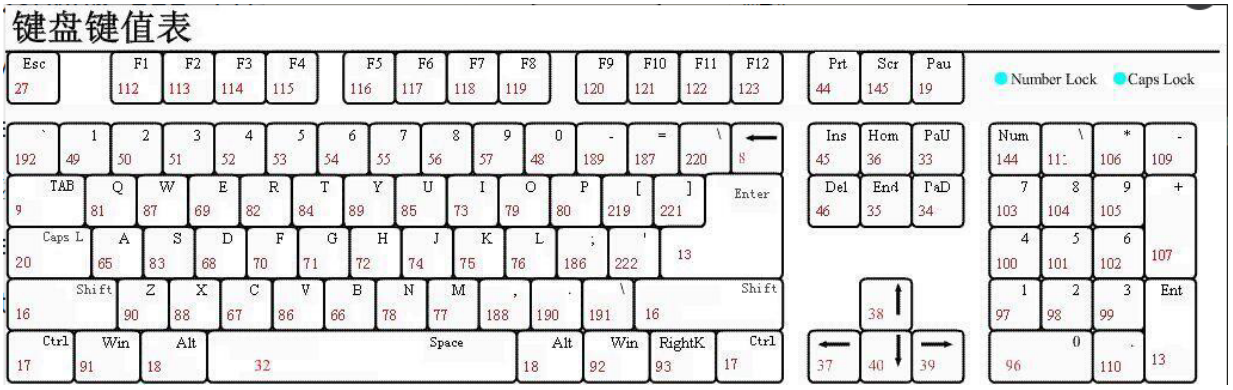
> KeyboardEvent
>
> 键盘事件对象(私有属性) - KeyboardEvent.prototype - UIEvent.prototype - Event.prototype - Object.prototype

`which`

> 当前按键的‘键盘码’值
>
> IE6~8下是没有这个属性的，需要我们使用 `keyCode` 属性

常用按键的键盘码

```
1. 空格键(SPACE): 32
2. 回车键(ENTER): 13
3. 回退键(BACK SPACE): 8
4. 删除键(DELETE): 46
5. 取消键(ESC): 27
6. 制表符键(TAB): 9
7.
8. 左: 37
9. 上: 38
10. 右: 39
11. 下: 40
12.
13. SHIFT: 16
14. CTRL: 17
15. ALT: 18
16. CapsLock: 20
17.
18. 字母键: 65~90 (a-z)
19. 数字键: 48~57 (0-9)
20. F5: 116
21. ...
```



小例子：禁止按F5刷新

```
1.  document.onkeydown = documen
    t.onkeypress = document.onkeyu
    p = function (e) {
2.      e = e || window.event;
3.      var keyNum = e.which ||
    e.keyCode;
4.      if (keyNum === 116) {
5.          //->F5
6.          e.keyCode = 0;//->IE下
    还需要把KEY-CODE设置为零才管用
7.          e.preventDefault ? e.p
    reventDefault() : e.returnValu
    e = false;
8.      }
9.  }
```

小例子：推盒子

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>珠峰培训-事件</title>
    <style>
        * {
            margin: 0;
            padding: 0;
        }

        html, body {
            width: 100%;
            height: 100%;
            overflow: hidden;
            cursor: pointer;
        }

        #box {
            position: absolut
e;
            top: 0;
            left: 0;
            width: 100px;

```

```
                height: 100px;
                background: red;
            }
        </style>
</head>
<body>
<div id="box"></div>

<script src="js/utils201708.min.js"></script>
<script src="js/animate.min.js"></script>
<script src="js/box.js"></script>
</body>
</html>
```

box.js

```
var minL = 0,
    minT = 0,
    maxL = (document.documentElement.clientWidth || document.body.clientWidth) - box.offsetWidth,
    maxT = (document.documentElement.clientHeight || document.body.clientHeight) - box.offsetHeight;

document.onkeydown = document.onkeypress = document.onkeyup = function (e) {
    e = e || window.event;
    var keyNum = e.which || e.keyCode,
        curL = utils.css(box, 'left'),
        curT = utils.css(box, 'top');
    switch (keyNum) {
        case 37:
            curL -= 100;

```

```
                break;
            case 38:
                curT -= 100;
                break;
            case 39:
                curL += 100;
                break;
            case 40:
                curT += 100;
                break;
        }
        curL = curL < minL ? minL
    : (curL > maxL ? maxL : curL);
        curT = curT < minT ? minT
    : (curT > maxT ? maxT : curT);

        zhufengAnimate({
            curEle: box,
            target: {
                left: curL,
                top: curT
            },
            duration: 300
        });

```

```
        //->按SPACE键蹦一下
        if (keyNum === 32) {
            zhufengAnimate({
                curEle: box,
                target: {top: curT - 100},
                duration: 200,
                effect: zhufengEffect.Back.easeOut,
                callBack: function () {
                    zhufengAnimate({
                        curEle: box,
                        target: {top: curT + 100},
                        duration: 200,
                        effect: zhufengEffect.Bounce.easeOut
                    });
                }
            });
        }
```

```
54.     };
```

A标签的默认行为

> A标签的默认行为有两个
>
> - 超链接,点击跳转页面
> - HSAH定位(锚点定位)

超链接

```
1.  <a href="http://www.zhufengpeixun.cn/">珠峰培训</a>
2.  <a href="http://www.zhufengpeixun.cn/" target="_blank">珠峰培训</a>
3.  //->target="_blank" 在新窗口打开需要跳转的页面，不加是本窗口跳转
```

HASH定位

当点击A标签的时候会在当前页面URL地址后面加 #box

http://…/link.html#box

在URL地址栏#后面出现的值，我们把它称之为HASH值(哈希值)，出现了HASH值，浏览器在渲染完成页面后，会直接定位到ID为HASH值盒子所在的位置

有时候，也会出现URL后面有HASH值，但是页面中并没有ID为它的盒子，此时我们利用HASH值可能是为了实现前端路由

```
<a href="#box">HASH定位</a>
//->当我们点击A标签的时候可以快速定位到当前页面ID为BOX盒子的位置

<a href="http://www.zhufengpeixun.cn/#course-list" target="_blank">珠峰培训最新全栈视频(广告)</a>
//->我们想当点击这个广告的时候，跳转到珠峰培训官网，并且直接定位到视频区域的位置，我们就可以在跳转URL的末尾加上对应的HASH值即可
```

> 在某些项目中，例如：京东的楼层导航，当我们点击某一个楼层按钮的时候，可以快速定位到对应楼层的位置，此时就可以基于HASH定位来做(没有动画效果)，京东是基于JS动画来做的，自己可以思考一下如何处理?

前端路由

在单页面应用中，我们经常使用HASH这种方式，来控制页面具体显示哪部分的内容，这就是前端路由一个初步的体现

单页面应用

http://kbs.sports.qq.com

在原始网站中,如果点击某一个导航或者按钮,我们想看到不同的内容,需要跳转到新的页面来观看(弊端:页面来回的跳转，体验度不好，而且每跳转到新页面，所有的内容都需要重新的渲染...)

后来随着互联网技术的发展,出现了一种新的模式`单页面应用`：它是把之前我们需要好多页面来展示的内容都汇总到一个页面中(在一个页面中集成了多个页面),我们通过一些机制来控制不同区域内容的展示,不需要跳转页面，但是所有内容合在一起不是手动完成的，而是通过gulp/webpack等自动化平台合并到一起的

在项目中，有时候我们会使用A标签来实现一个按钮，点击按钮页面不跳转，也不会定位到其它的位置

> 使用A标签实现按钮,我们可以充分利用A:HOVER兼容所有浏览器的机制,给予按钮好看的样式，语义化也不错，所以现在很多大型网站中的按钮，大部分都是基于A标签来做的
>
> 此时当用户点击A的时候，我们需要把它的所有默认的行为给阻止掉才可以

```
1. //->HTML直接处理
2. <a href='javascript:;'>I AM BU
   TTON</a>
3. <a href='javascript:void 0;'>I
   AM BUTTON</a>
```

```
1.  //->使用JS处理
2.  //当我们点击A标签的时候，它是先触发A
    的CLICK事件行为，然后才继续触发默认
    行为的
3.  oLink.onclick=function(){
4.      return false;//->返回FALS
    E，终止A标签继续执行默认行为的操
    作，阻止了默认行为(只有返回FALSE才
    可以)
5.  }
6.
7.  oLink.onclick=function(e){
8.      e=e||window.event;
9.      e.preventDefault?e.prevent
    Default():e.returnValue=false;
10. }
```

事件传播

捕获、目标、冒泡

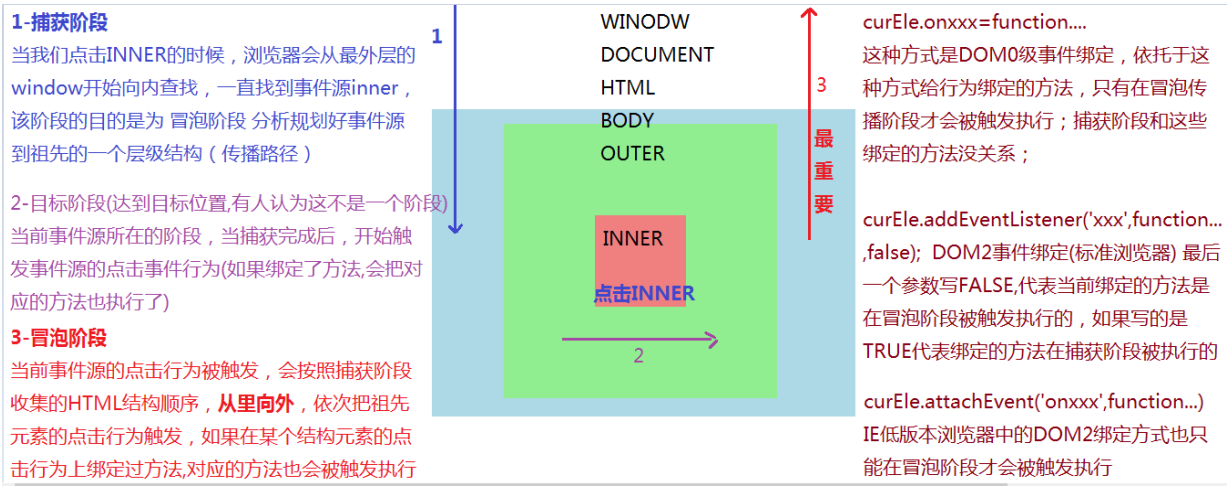
Event.prototype

NONE：0

CAPTURING_PHASE：1 捕获阶段

AT_TARGET：2 目标阶段

BUBBLING_PHASE：3 冒泡阶段



三个阶段，项目中我们最常使用的是冒泡传播阶段，捕获阶段基本不处理任何事情

冒泡传播

> 当前元素的某一个事件行为被触发，那么其祖先元素的相关事件行为都会被依次触发，这种机制就是冒泡传播机制
>
> 大部分事件行为天生就存在冒泡传播，不是事情行为是不存在冒泡传播的
>
> 我们可以基于 `e.stopPropagation/e.cancelBubble=true` 来阻止事件的冒泡传播

哪些事件行为没有传播机制

load

error

scroll

表单的一些事件行为：focus、blur 没有冒泡传播的机制 [change、key系列的行为，存在传播的机制]

mouseenter和mouseleave不存在冒泡传播的机制

mouseenter和mouseover的区别?

`mouseover`

> 1、存在事件的冒泡传播
> 2、从容器子元素中重新滑入到父容器中，也会重新触发父元素的mouseover行为
> =>mouseover是滑入不是进入

`mouseenter`

> 1、默认阻止了冒泡传播机制
> 2、从子元素中重新进入到父容器中，不会再重复触发父元素的mouseenter行为
> =>mouseenter是进入行为

`mouseout和mouseleave`

> 也是同样的机制，mouseleave不存在冒泡传播，而mouseout存在冒泡传播；从父元素进入到子元素，会触发父元素的mouseout，但是mouseleave不会被触发，因为鼠标还没有离开父容器呢，只是进入里面小容器中了而已

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>珠峰培训</title>
    <link rel="stylesheet" href="css/reset.min.css">
    <style>
        html, body {
            width: 100%;
            height: 100%;
            background: lightblue;
            overflow: hidden;
        }

        #outer {
            position: relative;
            margin: 20px auto;
            width: 300px;
            height: 300px;
            background: lightgreen;

```

```
            }

            #inner {
                position: absolut
    e;
                top: 50%;
                left: 50%;
                margin: -50px 0 0
    -50px;
                width: 100px;
                height: 100px;
                background: lightc
    oral;
            }
        </style>
    </head>
    <body>
    <div id="outer">
        <div id="inner"></div>
    </div>

    <script>
        outer.onmouseover = functi
    on () {
            console.log('OUTER OVE
```

```
R');
42.     };
43.     outer.onmouseout = functio
n () {
44.         console.log('OUTER OU
T');
45.     };
46.     inner.onmouseover = functi
on () {
47.         console.log('INNER OVE
R');
48.     };
49.     inner.onmouseout = functio
n () {
50.         console.log('INNER OU
T');
51.     };
52.
53.     outer.onmouseenter = funct
ion () {
54.         console.log('OUTER ENT
ER');
55.     };
56.     outer.onmouseleave = funct
ion () {
57.
```

```
            console.log('OUTER LEA
    VE');
58.     };
59.     inner.onmouseenter = funct
    ion () {
60.         console.log('INNER ENT
    ER');
61.     };
62.     inner.onmouseleave = funct
    ion () {
63.         console.log('INNER LEA
    VE');
64.     };
65. </script>
66. </body>
67. </html>
```

自己回去按照上去代码试试，结果就知道了

小案例：京东放大镜

放大镜的原理:

1、左侧的盒子和右侧盒子是相同大小的

2、左侧MARK:左侧盒子获取的比例 === 左侧小图片:右侧大图片

3、当MARK在左侧中移动的时候，我们同时控制大图片在右侧盒子中也移动，而且移动的距离正好是左侧MARK的一定比例(MARK/左侧盒子)

鼠标在MARK中间，MARK位置计算原理


SMALL距离BODY的上偏移
e.pageY 鼠标位置距离BODY的Y轴坐标
? MARK的TOP
SMALL距离BODY的左偏移
?
MARK的LEFT
e.pageX 鼠标位置距离BODY的X轴坐标
LEFT = e.pageX - SMALL左偏移 - MARK宽度的一半
TOP = e.pageY - SMALL上偏移 - MARK高度的一半

结构和样式

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>珠峰培训-放大镜</title>
    <link rel="stylesheet" href="css/reset.min.css">
    <style>
        .container {
            margin: 20px auto;
            width: 600px;
        }

        .conLeft, .conRight {
            position: relative;
            float: left;
            width: 300px;
            height: 300px;
        }

        .conLeft img {
            display: block;

```

```
            width: 100%;
            height: 100%;
        }

        .conLeft .mark {
            display: none;
            position: absolute;
            top: 0;
            left: 0;
            width: 100px;
            height: 100px;
            background: lightcoral;

            opacity: 0.5;
            filter: alpha(opacity=50);
            cursor: move;
        }

        .conRight {
            display: none;
            overflow: hidden;
        }

```

```
45.             .conRight img {
46.                 position: absolut
    e;
47.                 top: 0;
48.                 left: 0;
49.                 width: 300%;
50.                 height: 300%;
51.             }
52.         </style>
53.     </head>
54.     <body>
55.     <div class="container clear">
56.         <!--SMALL IMG-->
57.         <div class="conLeft" id="s
    mallBox">
58.             <img src="img/1.jpg">
59.             <div class="mark" i
    d="mark"></div>
60.         </div>
61.
62.         <!--BIG IMG-->
63.         <div class="conRight" i
    d="bigBox">
64.             <img src="img/2.jpg" i
```

```
    d="bigImg">
65.      </div>
66.  </div>
67.  </body>
68.  </html>
```

JS

```
var smallBox = document.getElementById('smallBox'),
            bigBox = document.getElementById('bigBox'),
            mark = document.getElementById('mark'),
            bigImg = document.getElementById('bigImg');

smallBox.onmouseenter = function (e) {
    bigBox.style.display = mark.style.display = 'block';
    computedMark(e);
};
smallBox.onmousemove = computedMark;
smallBox.onmouseleave = function (e) {
    bigBox.style.display = mark.style.display = 'none';
};

//=>计算MARK的位置

```

```
//1、鼠标进入和鼠标在SMALL-BOX中移动的时候都要计算位置
//2、MARK永远走不出SMALL-BOX的范围(边界判断)
function computedMark(e) {
    e = e || window.event;
    e.pageX = e.clientX + (document.documentElement.scrollLeft || document.body.scrollLeft);
    e.pageY = e.clientY + (document.documentElement.scrollTop || document.body.scrollTop);
    var smallOffset = offset(smallBox),
            markW = mark.offsetWidth,
            markH = mark.offsetHeight;
    var markL = e.pageX - smallOffset.left - markW / 2,
            markT = e.pageY - smallOffset.top - markH / 2;
    //->边界判断
    var maxL = smallBox.offset
```

```
    Width - markW,
29.             maxT = smallBox.of
    fsetHeight - markH;
30.         markL = markL < 0 ? 0 : (m
    arkL > maxL ? maxL : markL);
31.         markT = markT < 0 ? 0 : (m
    arkT > maxT ? maxT : markT);
32.         mark.style.left = markL +
    'px';
33.         mark.style.top = markT +
    'px';
34.         //->让大图也按照比例随着MARK一
    起移动(方向相反)
35.         bigImg.style.left = -markL
    * 3 + 'px';
36.         bigImg.style.top = -markT
    * 3 + 'px';
37.     }
38.
39.     //=>获取当前元素距离BODY的偏移
40.     function offset(curEle) {
41.         var l = curEle.offsetLeft,
42.             t = curEle.offsetT
    op,
43.             p = curEle.offsetP
```

```
    arent;
44.     while (p && p !== documen
    t.body) {
45.         if (!/MSIE 8/i.test(na
    vigator.userAgent)) {
46.             l += p.clientLeft;
47.             t += p.clientTop;
48.         }
49.         l += p.offsetLeft;
50.         t += p.offsetTop;
51.         p = p.offsetParent;
52.     }
53.     return {left: l, top: t};
54. }
```