2019/07/16 08:28

ele.onclick = function(){}

兼容性很好，但是同一个元素 只能绑定一个处理函数

等同于

```
<div onclick="console.log('a')"></div>
```

obj.addEventListener(事件类型，处理函数，false)

```
div.addEventListener('click',function(){
   console.log('a');
},false);
```

可以给一个对象的一个处理事件绑定多个函数

但不可以同一个函数绑定多次

```
 var div = document.getElementsByTagName('div')[0];
       div.addEventListener('click',test,false);
       div.addEventListener('click',test,false);

      function test(){
         console.log('a');
      }
```

```
div.addEventListener('click',function(){
   console.log('a');
},false);
div.addEventListener('click',function(){
   console.log('a');
},false);
```

但是这种方式可以

IE8及以下不兼容

attachEvent('on' + type,fn);

IE独有，一个事件同样可以绑定多个处理程序

同一个函数绑定多次也能执行多次

removeEventListener

移除事件

若绑定匿名事件，则无法解除

事件冒泡

结构上（非视觉上）嵌套关系的元素，会存在事件冒泡的功能，即同一事件，自子元素冒泡向父级元素（自底向上）

所以说哪怕方块移开，只要存在结构关系，就会冒泡

即子元素触发事件，会渗透到父级，使之触发事件，从div上看如同冒泡

点击父级的子元素上，父级也会触发事件

```
<style>
    .wrapper {
        height: 300px;
        width: 300px;
        background-color: red;
    }
    .content {
        height: 200px;
        width: 200px;
        background-color: #f40;
    }
    .box {
        height: 100px;
        width: 100px;
        background-color: green;
    }
</style>
<body>
    <div class="wrapper">
        <div class="content">
            <div class="box"></div>
        </div>
    </div>
    <script>
        var wrapper = document.getElementsByTagName('div')[0];
        var content = document.getElementsByTagName('div')[1];
        var box = document.getElementsByTagName('div')[2];
        wrapper.addEventListener('click', function () {
            console.log('wrapper');
        }, false);
        content.addEventListener('click', function () {
            console.log('concent');
        }, false);
        box.addEventListener('click', function () {
            console.log('box');
        }, false);
    </script>
</body>
```

事件捕获

结构上（非视觉上）嵌套关系的元素，会存在事件捕获的功能，即同一事件，自父元素捕获至子元素（事件源元素）。自顶向下

IE没有捕获事件

即addEventListener的false改为true

并且和事件冒泡是相反的，先执行父级的事件

```
<style>
    .wrapper {
        height: 300px;
        width: 300px;
        background-color: red;
    }

    .content {
        height: 200px;
        width: 200px;
        background-color: #f40;
    }
    .box {
        height: 100px;
        width: 100px;
        background-color: green;
    }
</style>
<body>
    <div class="wrapper">
        <div class="content">
            <div class="box"></div>
        </div>
    </div>
    <script>
        var wrapper = document.getElementsByTagName('div')[0];
        var content = document.getElementsByTagName('div')[1];
        var box = document.getElementsByTagName('div')[2];
        wrapper.addEventListener('click', function () {
            console.log('wrapper');
        }, true);
        content.addEventListener('click', function () {
            console.log('concent');
        }, true);
        box.addEventListener('click', function () {
            console.log('box');
        }, true);
    </script>
</body>
```

一个对象的一个事件类型上面绑定的一个处理函数只能遵循一个事件处理类型

如果同时存在，捕获先于原事件先于冒泡

focus blur change submit reset select 等事件不冒泡

事件对象

```
div.onclick = function(e){
    {}
    var event = e || window.event
    e.stopPropagation();
    但不支持IE8及以下
    执行这个函数就会取消冒泡事件
    e.cancelBubble = true;
    方法同上，IE独有
    this.style.background-color = green;
}
//系统会自动传入参数，这个参数是一个记录事件发生的所有值的对象
//IE浏览器下e就会失效
    //IE会在window.event上记录事件信息
```

右键事件

```
document.oncontextmenu = function(){
    //右键默认出菜单
    return false;
    //阻止默认事件，兼容性很好
    //以对象属性的方式注册的事件才生效

}


document.oncontextmenu = function(e){
    e.preventDefault();
    //阻止默认时间IE9以下不兼容
    e.returnValue = false;
    //兼容IE
}


封装阻止默认事件的函数
function cancelHandler(event){
    if(e.preventDefault){
        event.preventDefault();
    }else{
        event.returnValue = false;
    }

}
```

事件源对象

专门记录事件源！！！

即事件发生的标签

event.target 火狐只有这个

event.srcElement IE只有这个

这两个chrome都有

```
<style>
  .wrapper{
      height: 200px;
      width: 200px;
      background-color: red;
  }
  .box{
      height: 100px;
      width: 100px;
      background-color: green;
  }
</style>
 <div class="wrapper">
    <div class = "box"></div>
 </div>
 <script>
      var wrapper = document.getElementsByClassName('wrapper')[0];
      var box = document.getElementsByClassName('box')[0];
      wrapper.onclick = function(e){
          var event = e||window.event;
          var target = event.target || event.srcElement;
      console.log(event);

      }
 </script>
```

点击相应li出现相应数字

正常情况下，使用循环，缺点：数据多效率低，不可再添加li标签

使用事件源，直接找到鼠标点击的标签，可扩充

```
<body>
  <ul>
      <li>1</li>
      <li>2</li>
      <li>3</li>
      <li>4</li>
      <li>5</li>
      <li>6</li>
      <li>7</li>
      <li>8</li>
      <li>9</li>
      <li>10</li>
  </ul>
 <script>
      var ul = document.getElementsByTagName('ul')[0];
```

```
    ul.onclick = function(e){
        var event = e||window.event;
        var target = event.target || event.srcElement;
    console.log(target.innerText);

    }
</script>
```

事件委托

利用事件冒泡，和事件源对象进行处理

有点：

1，性能不需要循环所有元素一个个绑定事件

2，灵活 当有新的子元素时不需要重新绑定事件

事件

onmouseenter	:鼠标在标签上时
onmouseleave	:鼠标在标签外时
onmousemove	:鼠标移动时
onmousedown	:鼠标点击时
onmouseup	:松开鼠标时

实现div的拖拽

```
<style>
    .box {
        height: 100px;
        width: 100px;
        background-color: red;
    }
</style>
<body>
    <div style="background-color: green;height: 100px;width: 100px;position:
absolute;left: 0px;top: 0px;"></div>
    <script>
        var div = document.getElementsByTagName('div')[0];
        var disX,disY;
        div.onmousedown = function(e){
            disX = e.pageX - parseInt(div.style.left);
            disY = e.pageY - parseInt(div.style.top);
            document.onmousemove = function(e){
                var event = e || window.event;
                div.style.left = e.pageX - disX + "px";
                div.style.top = e.pageY -disY + "px";
            }
            div.onmouseup = function(){
                document.onmousemove = null;
```

```
        }
    }
</script>
```

另一种方法

div.setCapture();

div会捕获页面上发生的所有事件 到自己身上触发

div.releaseCapture();

使用完之后的释放

只有IE能用

click = mousedown + mouseup

触发顺序

先down后up再click

mouseover == mouseenter 鼠标再标签上

mouseout == mouseleave 鼠标离开标签

移动端 touchstart touchmove touchend

能区分左右键的只有

mouseup mousedown

click 不能

e.button == 2 -->右键

e.button == 0 -->左键

```
//判断拖拽，通过事件
var firstTime = 0;
var lastTime = 0;
var key = false;
document.onmousedowm = function(){
    firstTime = new Date().getTime();
}
document.onmouseup = function(){
    lastTime = new Date().getTime();
    if(lastTime - firstTime < 300){
        key = true;
    }
}
document.onclick = function(){
    if(key){
        console.log('click');
        key = false;
```

```
  }
}
```

键盘事件

~~onkeypress = onkeydown + onkeyup~~

触发顺序

先down --> press --> up

连续按时，一直触发down和press

keydown和keypress的区别

keydown除了fn键都能监测到　　只能监测是哪个键，不能监测大小写，which属性表示键的序号

keypress只能监测字符类按键，并且能区分大小写　　charCode属性记录ASCII码

文本类操作事件

oninput

无论删除还是输入输入框的文本，都会触发事件

onchange

输入框聚焦-> 发生改变(删除或者增加文本) -> 失去焦点 -> 触发事件

(对比的是聚焦前后的文本)

onfocus　　聚焦后触发事件

onblur　　失去焦点，触发事件

```
<input type="text" value="请输入用户名" style="color:#999"
onfocus="if(this.value=='请输入用户名'){this.value='';
this.style.color='#424242';}" onblur="if(this.value=='')
{this.value='请输入用户名';}">
```

窗体类操作事件（window上的事件）

window.onscroll

滚动条滚动 事件触发

es6下没有fixed定位

通过滚动条事件模拟fixed定位

window.onload

整个页面全部加载完后，等待交互之前，window.onload执行（效率低）