事件

何为事件

1. 说白了就是一个动作，与用户实现交互功能，你动一下，他会给你一个反馈

   ```
   div.onclick = function(){}
   ```

   div.onclick就是一个事件，后面的函数是响应
2. 重要吗？ — 交互体验的核心功能

如何绑定事件

1. ele.onxxx = function (event) {}
   - 兼容性很好，但是一个元素只能绑定一个处理程序
   - 基本等同于写在HTML行间上(onclick作为属性写在内联样式表里，，这种绑定方式叫做**句柄**的绑定方式)
2. ele.addEventListener(type, fn, false);
   - IE9以下不兼容，可以为一个事件绑定多个处理程序，触发之后会按照绑定顺序依次触发

     ```
     var div = document.getElementsByTagName('div')[0];

     div.addEventListener('click', function (){

         console.log('a');

     }, false);

     div.addEventListener('click', function() {
         console.log('b');
     }, false);
     ```
   - 如果同一个函数绑定多次，那么只执行一次

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

这两个函数长得一样，但是位置不同，所以输出a a

```
div.addEventListener('click', test, false);
div.addEventListener('click', test, false);

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

同一个函数，绑定多次，只执行一次，输出a

- type：事件类型，点击样式直接写click
- fn：处理函数

3. ele.attachEvent('on' + type, fn);
   - IE独有，一个事件同样可以绑定多个处理程序，但是如果同一个函数绑定多次，它也会执行多次
   - type：事件类型
   - fn： 处理函数

事件处理程序的运行环境

1. ele.onxxx = function (event) {}
   - 程序this指向是dom元素本身
2. obj.addEventListener(type, fn, false);
   - 程序this指向是dom元素本身
3. obj.attachEvent('on' + type, fn);

- 程序this指向window

4. 封装兼容性的 addEvent(elem, type, handle)方法
   事件处理函数handle写在外面，使this指向elem而不是指向window

```
function addEvent(elem,type,handle){
            if(elem.addEventListener){
                elem.addEventListener(type,handle,false);
            }else if(elem.attachEvent){
                elem.attachEvent("on"+type,function(){
                    handle.call(elem);
                })
            }else{
                elem["on" + type] = handle;
            }
        }
```

解除事件处理程序

1. ele.onclick = false/""/null;
2. ele.removeEventListener(type, fn, false);
3. ele.detachEvent('on' + type, fn);

注:若绑定匿名函数，则无法解除。

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

    console.log('a');

}, false);


div.removeEventListener('click', , false);
```

```
var div = document.getElementsByTagName('div')[0];
div.addEventListener('click', test, false);
function test() {
    console.log('a');
}

div.removeEventListener('click', test, false);
```

事件处理模型

1. 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，自子元素冒泡向父元素。（自底向上）
2. 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
    - 开启方式： `obj.addEventListener(type, fn, true);`
    - 其他事件捕获方式（IE特有）： `elem.setCapture();` HTML元素硬是任何地方发生的任何事件捕获到自己身上来，这就解决的物体拖拽时因为鼠标移动过快，而物体跟不上鼠标的过程，这种方法对其他元素是有害处的，其他元素的方法会不好使，所以使用完之后要用 `elem.releaseCapture();` 释放，这是一个过时的问题，但是面试官可能会问，**不用会操作，知道就行，**
3. 触发顺序：先捕获，后冒泡
4. 一个对象的一个事件类型上面绑定着一个处理函数只能遵循一个处理模型

点击黄色区域：

```
wrapper.addEventListener('click', function () {
    console.log('wrapper')
}, true);
content.addEventListener('click', function () {
    console.log('content')
}, true);
box.addEventListener('click', function () {
    console.log('box')
}, true);

wrapper.addEventListener('click', function () {
    console.log('wrapperBubble')
}, false);
content.addEventListener('click', function () {
    console.log('contentBubble')
}, false);
box.addEventListener('click', function () {
    console.log('boxBubble')
}, false);
```

```
wrapper
content
box
boxBubble
contentBubble
wrapperBubble
```

解析： 先捕获，后冒泡

点击黄色区域：

```
wrapper.addEventListener('click', function () {
    console.log('wrapperBubble')
}, false);
content.addEventListener('click', function () {
    console.log('contentBubble')
}, false);
box.addEventListener('click', function () {
    console.log('boxBubble')
}, false);

wrapper.addEventListener('click', function () {
    console.log('wrapper')
}, true);
content.addEventListener('click', function () {
    console.log('content')
}, true);
box.addEventListener('click', function () {
    console.log('box')
}, true);
```

```
wrapper
content
boxBubble
box
contentBubble
wrapperBubble
```

解析： 红色捕获，绿色捕获，黄色**执行**，黄色**执行**，绿色冒泡，红色冒泡，**黄色执行的顺序应该符合谁先绑定谁先执行**

5. focus，blur，change，submit，reset，select 等事件不冒泡

取消冒泡（系统有时候会自动添加冒泡）

1. 取消冒泡：
    - W3C标准 event.stopPropagation();但不支持ie9以下版本

```
document.onclick = function () {
    console.log('你闲的呀');
}
var div = document.getElementsByTagName('div')[0];

div.onclick = function (e) {

    e.stopPropagation();

    this.style.background = "green";
}
```

- IE谷歌：event.cancelBubble = true;

```
document.onclick = function () {
    console.log('你闲的呀');
}
var div = document.getElementsByTagName('div')[0];

div.onclick = function (e) {

    // e.stopPropagation();
    e.cancelBubble = true;

    this.style.background = "green";
}
```

- 封装取消冒泡的函数 stopBubble(event)

```
document.onclick = function () {
    console.log('你闲的呀');
}
var div = document.getElementsByTagName('div')[0];

div.onclick = function (e) {

    // e.stopPropagation();
    stopBubble(e);

    this.style.background = "green";
}
function stopBubble(event) {
    if(event.stopPropagation) {
        event.stopPropagation();
    }else{
        event.cancelBubble = true;
    }
}
```

阻止默认事件

1. 默认事件：表单提交，a标签跳转，右键菜单等
   - return false; 以对象属性的方式注册的事件才生效

```
document.oncontextmenu = function () {
    console.log('a');
    return false;
}
```

点击右键，只在控制台输出a但是不弹出菜单

   - event.preventDefault(); W3C标注，IE9以下不兼容

```
document.oncontextmenu = function (e) {
    console.log('a');
    e.preventDefault();
}
```

   - event.returnValue = false; 兼容IE

```
document.oncontextmenu = function (e) {
    console.log('a');
    e.returnValue = false;
}
```

- 封装阻止默认事件的函数 cancelHandler(event);

```
document.oncontextmenu = function (e) {
    console.log('a');
    cancelHandler(e);
}

function cancelHandler(event) {
    if(event.preventDefault) {
        event.preventDefault();
    }else{
        event.returnValue = false;
    }
}
```

事件对象

1. e（非IE）|| window.event（用于IE）

```
var div = document.getElementsByTagName('div')[0];
div.onclick = function (e) {

    var event = e || window.event;


}
```

2. 事件源（源头）对象:
   - event.target 火狐独有的
   - event.srcElement IE独有的
   - 这俩chrome都有

```
var wrapper = document.getElementsByClassName('wrapper')[0];
var box = document.getElementsByClassName('box')[0];
//事件源对象
wrapper.onclick = function (e) {

    var event = e || window.event;
    var target = event.target || event.srcElement;
    console.log(target);

}
```

3. 用处：事件委托
4. 兼容性写法

事件委托

1. 利用事件冒泡，和事件源对象进行处理
2. 优点：
   - 性能 不需要循环所有的元素一个个绑定事件
   - 灵活 当有新的子元素时不需要重新绑定事件
3. 例题：
   - 选中li，并输出对应内容，即使动态添加li，点击之后也能输出li内容

```
    <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 type="text/javascript">

    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);

    }
```

事件分类

1. 鼠标事件
   - click、mousedown、mouseup， mouseover、mouseout， mouseenter mouseleave, mousemove、contextmenu、
     - click、mousedown、mouseup(click = mousedown + mouseup)和绑定顺序没关系，和触发先后顺序有关

```
document.onclick = function () {
    console.log('click');
}

document.onmousedown = function () {
    console.log('mousedown');
}

document.onmouseup = function () {
    console.log('mouseup')
}
```

```
mousedown
mouseup
click
```

- mouseover鼠标覆盖是发生什么事，mouseout鼠标离开时发生什么事，实际上CSS中hover就是用JS写的
- mouseenter、mouseleave HTML5新规范，效果与上面的相同

```
<div style="width:100px;height:100px;background-color:red;"></div>
<script type="text/javascript">


    var div = document.getElementsByTagName('div')[0];
    div.onmouseenter = function () {
        div.style.background = "yellow";
    }

    div.onmouseleave = function () {
        div.style.background = "green";
    }
```

- 能区分鼠标左右键的只有mousedown、mouseup，用事件对象的button属性来区分鼠标的按键，0/1/2 --> 左键/滚动论/右键

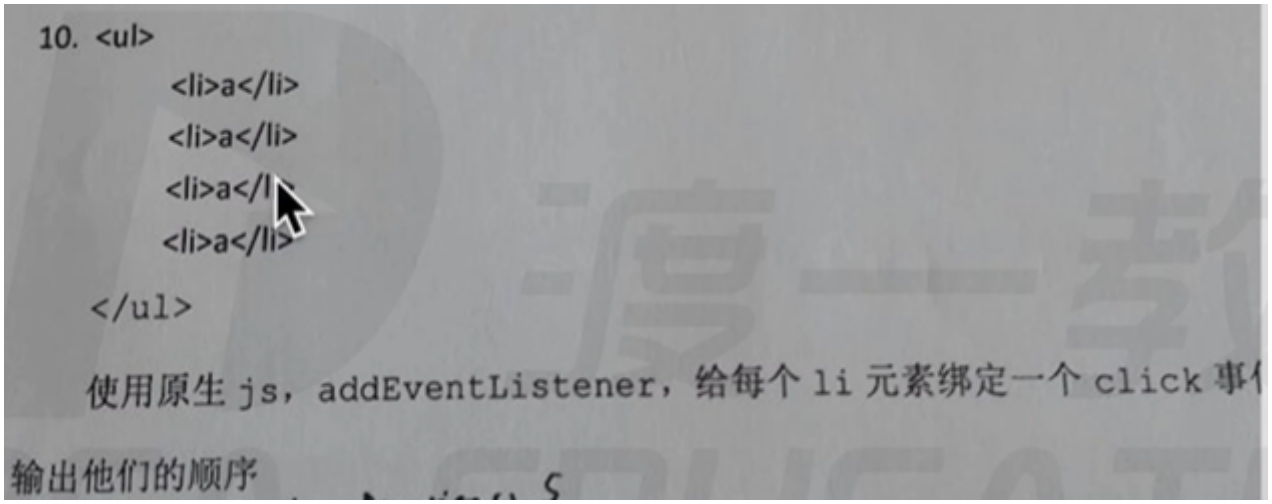
```
document.onmousedown = function (e) {
    if(e.button == 2) {
        console.log('right');
    }else if(e.button == 0){
        console.log('left');
    }
}
```

- DOM3标准规定:click事件只能监听左键,只能通过mousedown 和 mouseup来判断鼠标键
- 如何解决mousedown和click的冲突

2. 键盘事件
    - keydown keyup keypress
    - keydown > keypress > keyup
    - keydown和keypress的区别
        - keydown可以响应任意键盘按键，keypress只可以相应字符类键盘按键(ASCII码表里面有的都是字符类按键,fn不在ASCII码表里，所以是辅助按键)
        - keypress返回ASCII码，可以转换成相应字符
3. 文本类事件
    - input：只要文本有变化就会触发
    - change：监听聚焦和失去焦点两个状态，如果两个状态前后没变化则不触发，否则触发
    - focus：聚焦时状态
    - blur：失去焦点时的状态
4. 窗体操作类(window上的事件)
    - scroll：滚动条滚动，事件触发
    - load：没意义且最慢，当页面的解析加载等全部操作做完之后，只等待交互体验时才触发
    - 小练习:fixed定位 js兼容版（IE6没有fixed定位）

习题

习题1：



```
var liCol = document.getElementsByTagName("li");
        var len = liCol.length;

        for(var i = 0;i < len;i++){
            (function (i){
                liCol[i].addEventListener("click",function(){
                    console.log(i);
                },false)
            }(i))
        }
```

习题2： 拖拽应用

```
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";
    }
    document.onmouseup = function () {
        div.onmousemove = null;
    }
}
```

这种方法很粗糙，要求用addEventListener绑定事件，并且将所有代码封装到一个函数drg(elem)里去