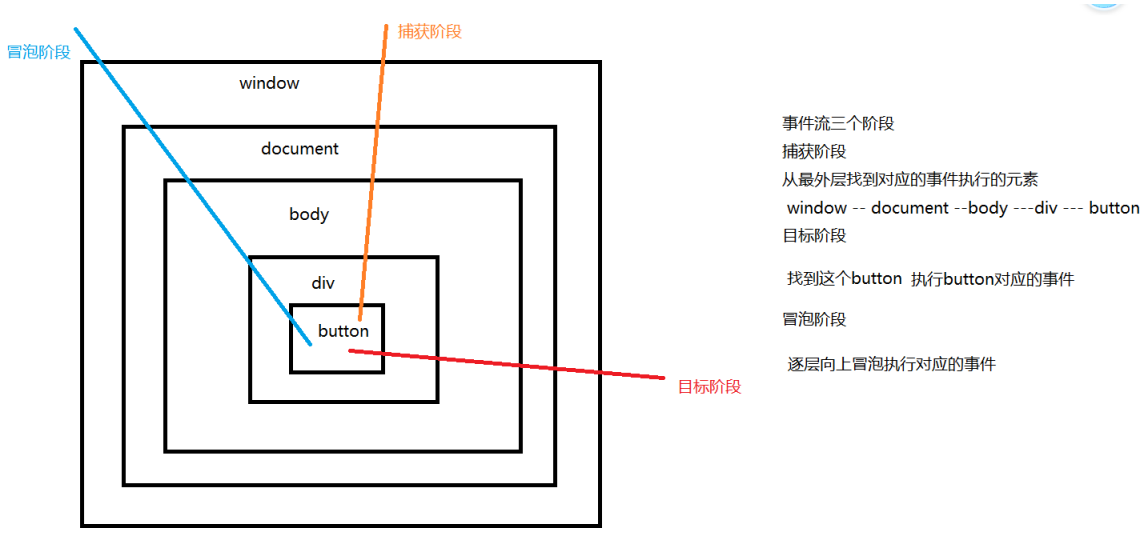
day12 事件下

事件流的传播流程

事件流的传播有三个阶段

- 捕获阶段
- 目标阶段
- 冒泡阶段



事件流的俩种模式

冒泡模式 （从里到外 逐层执行对应的事件）

冒泡模式是常用的模式，他现在默认设计的就是冒泡模式。

示例

```
<div onclick="alert(1)">
    <button onclick="alert(2)"></button>
</div>
<script>
    document.body.onclick = function () {
        alert(3)
    }
    document.onclick = function () {
        alert(4)
    }
    window.onclick = function () {
        alert(5)
    }
</script>
```

- 以上的这个示例 会从button开始不断向上执行 直到window停止 所以对应的执行结果为 2 1 3 4 5

如果我们不想触发对应的外层的事件 只是想触发本身的事件 那么我们就需要禁止事件冒泡了

禁止事件冒泡的处理

stopPropagation （event对象的方法 对于低版本ie浏览器不支持）

```
document.querySelector('button').onclick = function(e){
    //事件源对象
    e = e || window.event
    //禁止事件冒泡的方法
    e.stopPropagation()
    alert(2)
}
```

cancelBubble （event对象的属性）

```
//cancelBubble 取消冒泡 默认为false
e.cancelBubble = true
```

兼容写法

```
e.stopPropagation?e.stopPropagation():e.cancelBubble = true
```

捕获模式 （从外到里 逐层执行对应的事件）

捕获模式他是火狐提出来的模式，ie对应的6、7、8 不支持。现在的模式一般很少使用捕获模式。

默认行为

元素标签有其默认行为（a 标签会默认跳转页面 form标签里面submit默认提交（刷新）），对应的事件也有其默认行为（contextmenu 会出现对应的菜单栏等）。

e.defaultPrevented 检测当前是否禁止默认行为

```
 console.log(e.defaultPrevented) //是否阻止默认行为 只读 默认为false
```

禁止默认行为

preventDefault （event对象的方法）

```
e.preventDefault() //preventDefault阻止默认行为
```

returnValue （event对象的属性）

```
e.returnValue = false //兼容ie的
```

兼容写法

```
e.preventDefault?e.preventDefault():e.returnValue = false
```

return false

```
//对应的右键点击
window.oncontextmenu = function(e){
    console.log('右键点击了')
    // e.preventDefault?e.preventDefault():e.returnValue=false
    return false //一定要放在最后
}
```

示例

右键点击出现对应的菜单栏 这个菜单栏自定义（前进功能 后退功能 刷新功能 换肤 打印）

事件监听器

eventListener 他是一个标准的观察者模式，他是通过对应的监听器来监听事件的触发和执行。

主要有俩个方法

- addEventListener 添加事件监听器
- removeEventListener 移除事件监听器

addEventListener

传入对应的事件名及处理函数以及对应的是否冒泡

```
//获取按钮
var btn  = document.querySelector('button')
//添加事件监听器 传入 事件名 处理函数  是否捕获(默认的事件模式 冒泡 false 捕获 true)
// btn.addEventListener('click',function(){
//   console.log('按钮点击了')
// })//指定为冒泡模式
// btn.addEventListener('click',function(){
//   console.log('按钮点击了1')
// },true)//指定为捕获模式 先执行
// btn.addEventListener('click',function(){
//   console.log('按钮点击了2')
// })
//addEventListener 他可以给一个事件添加多个处理函数
//一个事件 有一个处理函数的数组
//事件监听器中的事件名 支持自定义
// btn.addEventListener('dblclick',function(){
//   console.log('双击')
// })
//直接使用onclick 进行赋值的操作他会被覆盖也就是说他只有一个处理函数 后写会覆盖先写的 （属性赋值操作）
// document.querySelector('div').onclick = function(){
//   console.log('div被点击了')
// }
btn.addEventListener('click',handler,false)//指定为冒泡模式
```

注意事项

- addEventListener 可以在一个事件中传入多个处理函数 （一个事件对应一个处理函数数组）
- EventListenter 支持自定义事件名
- 属性事件赋值不支持多个处理函数 （因为会被覆盖）

removeEventListener

移除对应的添加的事件监听器，传入事件名、处理函数、是否冒泡 每个都必须和添加的事件监听器一致 不然不能被移除

```
btn.addEventListener('click',handler,false)//指定为冒泡模式
// 要移除的事件名 要移除的处理函数(也要一致 如果是匿名函数那么就不能被移除 对象比对的是地址) 模式也要一致
btn.removeEventListener('click',handler,false)
function handler(){
    console.log('按钮点击了')
}
```

注意事项

- 如果添加事件监听器的时候传入处理函数为匿名处理函数 那么不能被移除（对象比对的是地址）

拖拽

拖拽原理

- 给对应的需要拖拽的元素添加鼠标按下事件
- 在按下事件内添加给区间的元素对应的鼠标移动事件
- 在按下事件内给document添加对应的鼠标弹起事件 在弹起事件中释放移动事件及释放弹起事件

基础拖拽

思路

- 获取拖拽的元素
- 给拖拽元素添加鼠标按下事件 并记录按下的坐标（在对应的盒子里的坐标）
- 在按下事件内给区间元素添加鼠标移动事件 并记录每次移动的坐标
- 在区间元素的鼠标移动事件中 设置对应的拖拽元素的坐标（移动的坐标 = 当前的坐标 - 鼠标点击位置的坐标 + 'px'）
- 在按下事件内在document中添加鼠标弹起事件 并释放之前的移动事件及自身的弹起事件

```
<div></div>
<script>
var box = document.querySelector('div')
box.onmousedown = function(e){
    e = e || window.event
    //记录在而盒子上的坐标
    var x = e.offsetX
    var y = e.offsetY
    //在document中移动
    document.onmousemove = function(e){
        e = e || window.event
        //获取页面上的坐标
        var currentX = e.pageX
        var currentY = e.pageY
        //移动的坐标 = 当前的坐标 - 鼠标点击位置的坐标 + 'px'
        box.style.left = currentX - x + 'px'
        box.style.top = currentY - y + 'px'
    }
    document.onmouseup = function(){
        document.onmousemove = null
        document.onmouseup = null
    }
```

```
}
</script>
```

区间拖拽

offset家族 （属于元素对象 element对象）

- offsetParent 偏移的父元素 （从里到外找有定位的父元素 没有的话就是body）
- offsetLeft 左偏移量 （不包含偏移的父元素本身的margin 包含偏移的父元素本身padding border）
- offsetTop 上偏移量
- offsetHeight 偏移元素的高度 （包含padding及border 不包含margin）
- offsetWidth 偏移元素的宽度

思路

- 获取拖拽的元素
- 给拖拽元素添加鼠标按下事件 并记录按下的坐标（在对应的盒子里的坐标）
- 在按下事件内给区间元素添加鼠标移动事件 并记录每次移动的坐标
- 在区间元素的鼠标移动事件中 获取对应的区间元素的位置 及 能够移动的距离 （区间元素的宽/高度 - 自身的宽/高度）
- 设置移动元素处在区间元素的位置 移动位置在父元素的坐标 = 页面的位置 - 父元素离页面的位置 - 鼠标点击的位置
- 对应坐标位置进行区间判断 小于0的时候值应该设置为0 大于能够移动的距离设为最大的距离
- 在按下事件内在document中添加鼠标弹起事件 并释放之前的移动事件及自身的弹起事件

```
<div>
    <button>
        移动的按钮
    </button>
</div>
<script>
    var box = document.querySelector('div')
    var button = document.querySelector('button')
    button.onmousedown = function(e){
        e = e || window.event
        //记录在而盒子上的坐标
        var x = e.offsetX
        var y = e.offsetY
        //在box中移动
        box.onmousemove = function(e){
            e = e || window.event
            //获取区间元素的位置
            var bx = this.offsetLeft
            var by = this.offsetTop
            //获取能够移动的最大距离
            var maxX =  this.offsetWidth - button.offsetWidth
            var maxY =  this.offsetHeight - button.offsetHeight
            //移动位置在父元素的坐标 = 页面的位置 - 父元素离页面的位置 - 鼠标点击的位置
            var targetX = e.pageX - bx - x
            var targetY = e.pageY - by - y
            //进行区间判断
            if(targetX < 0){
                targetX = 0
            }
            if(targetY < 0){
```

```
                targetY = 0
            }
            if(targetX > maxX){
                targetX = maxX
            }
            if(targetY > maxY){
                targetY = maxY
            }
            button.style.left = targetX + 'px'
            button.style.top = targetY + 'px'
        }
        document.onmouseup = function(){
            box.onmousemove = document.onmouseup  = null
        }
    }
</script>
```

封装一个方法找盒子到页面的距离

```
function getBoxToPageDistance(element){
    var distance = {
        x:0,
        y:0
    } //距离对象
    while(element.offsetParent){ //找到body就停止
        distance.x += element.offsetLeft
        distance.y += element.offsetTop
        element = element.offsetParent
    }
    return distance
}
```