# 事件对象

\* - 当浏览器调用事件的响应函数时，每次都会传递一个事件对象作为参数，可以定义一个形参event来使用

\* 在事件对象封装了当前事件相关的信息，比如：鼠标的坐标 键盘哪个按键被按下……

\* 我们可以通过该对象来获取事件相关的信息

在IE8及以下的浏览器中，没有将事件对象作为参数传递进函数，

\* 所以我们就不能以参数的形式来使用事件对象。

\* 在IE8中，是将事件对象作为window对象的属性保存的

\* 需要使用window.event来使用

处理兼容性的问题

if(!event){

event = window.event;

}

更常见的写法：event = event || window.event ;

clientX可以获取鼠标指针的水平坐标

clientY可以获取鼠标指针的垂直坐标

事件对象的clientX和clientY获取的坐标是鼠标相对于浏览器窗口可见区域的坐标

具体看练习

# 冒泡（Bubble）

\* - 冒泡简单来说就是事件的向上传导，当后代元素上的事件被触发时，将会导致祖先元素上的相同事件也被触发

\* - 大部分情况下冒泡都是对开发有利的，可以简化我们的开发

\* - 如果不希望发生事件的冒泡，则可以通过事件对象来取消冒泡

\* - 将事件对象的cancelBubble属性设置true，即可取消冒泡

冒泡与绑不绑定事件没关系。

# 事件委派

指统一将多个元素上的相同事件绑定到他们共同的祖先元素上，这样只需要绑定一次即可同时处理多个元素上的相同的事件，这样简化了代码的开发，也可以确保新添加的元素上也可以有事件响应函数

<body>

<button id="btn01">点我一下</button>

<ul id="u1">

<li><a href="javascript:;" >超链接一</a></li>

<li><a href="javascript:;" >超链接二</a></li>

<li><a href="javascript:;" >超链接三</a></li>

<li><a href="javascript:;" >超链接四</a></li>

</ul>

</body>

<script type="text/javascript">

window.onload = function(){

//点击按钮以后，向ul中添加一个新的超链接

//获取按钮对象和ul对象

var btn01 = document.getElementById("btn01");

var u1 = document.getElementById("u1");

//为按钮绑定单击响应函数

btn01.onclick = function(){

//创建一个li

var li = document.createElement("li");

//设置li中的内容

li.innerHTML = '<a href="javascript:;">新超链接</a>';

//将li添加到ul中

u1.appendChild(li);

};

//获取所有的超链接

var links = document.getElementsByTagName("a");

//为所有的超链接都绑定一个单击响应函数

for(var i=0 ; i<links.length ; i++){

links[i].onclick = function(){

alert("hello");

\* 这里我们通过for循环的形式来为每一个超链接都绑定了一个相同的单击响应函数，

\* 这里for循环每执行一次，函数就会重新创建一次，而且这些函数的功能还都是相同的。

\* 而且我们所绑定事件都是仅限于现在已有的元素，而新添加的元素上是没有响应函数，

\* 还需要单独绑定，这样实在是太麻烦了

};

}

</script>

我们希望，响应函数只绑定一次，就可以应用到所有的元素上（包括新添加的元素）

如果给a绑定，肯定得绑定多次，如果给ul绑定呢？

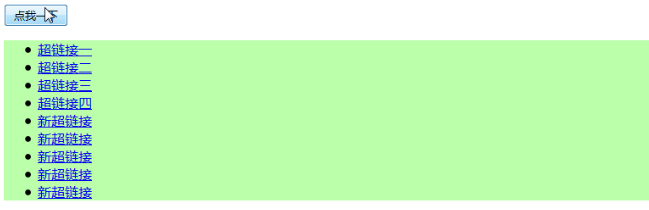
将上面for循环注释掉，

u1.onclick = function(){

alert(“ul上的单击响应函数”);

};

利用冒泡，点了a就相当于点了ul，这样新添加的a也有单击响应函数，但点击ul的作用范围都会触发单击响应函数，给ul设置背景，如下图

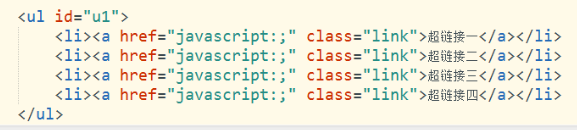


图中点击绿色背景都会触发单击响应函数，但我们希望单击响应函数只对超链接起作用，我们希望判断当前触发事件的是否是超链接，如果是超链接，则触发事件，如果不是则不触发

由于事件是绑定给ul的，this是ul，不能使用this判断是否是超链接

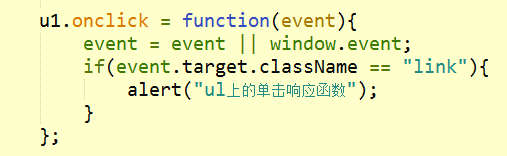
事件是由谁触发的是事件相关的信息，事件相关的信息找事件对象event，事件对象中有一个target属性，事件是由谁触发的，target就是谁

这样只需要判断target是不是超链接就行，可以给超链接加一个class，当然新添加的超链接也需要加class





这样判断className是不是link就可以判断是不是超链接了



# 同一个事件绑定多个处理函数

使用 对象.事件 的形式不能同时为一个元素的同一个事件绑定多个处理函数，如果绑定了多个，则后边的会将前边的覆盖掉

如果需要同时为一个事件绑定多个响应函数，则可以使用addEventListener()这个方法来绑定响应函数

\* 参数：

\* 1.要绑定的事件（字符串 不要on）

\* 2.回调函数（事件触发时，该函数将会执行）

\* 3.是否在捕获阶段触发事件（都传false）

\* 使用这种方式可以同时为一个事件绑定多个响应函数，响应函数按照绑定的顺序执行，先绑定先执行

\* 该方法不支持IE8及以下的浏览器，在这些浏览器中需要使用 attachEvent()方法来实现相同的功能

btn01.addEventListener("click",function(){

alert(1);

},false);

btn01.addEventListener("click",function(){

alert(2);

},false);

btn01.addEventListener("click",function(){

alert(3);

},false);

attachEvent()

\* - 参数：

\* 1.事件的类型（字符串 要on）

\* 2.回调函数

\* - 可以同时为一个事件绑定多个响应函数，但是它的执行顺序不一定

\* ie9 ie10 先绑定先执行，如下代码输出为1，2，3

\* ie8 后绑定先执行，如下代码输出为3，2，1

\* 这个方法只支持IE10及以下的浏览器

btn01.attachEvent("onclick" , function(){

alert(1);

});

btn01.attachEvent("onclick" , function(){

alert(2);

});

btn01.attachEvent("onclick" , function(){

alert(3);

});

解决兼容性问题：

自定义一个函数，来兼容所有的浏览器

\* 参数：

\* obj 要绑定事件的对象

\* eventStr 事件的字符串,不要on

\* callback 回调函数，事件触发时调用的函数

function bind(obj , eventStr , callback){

if(obj.addEventListener){

//如果是正常浏览器

obj.addEventListener(eventStr , callback , false);

}else{

//IE8

obj.attachEvent("on"+eventStr ,callback);

}

}

在正常浏览器中使用addEventListener()来绑定，它的响应函数中的this就是绑定事件的对象

而在IE中，使用的是attachEvent()来绑定的事件，它的响应函数中的this是window

需要统一一下this，this是window或者是绑定事件的对象都行，但this是绑定事件的对象最好，所以还需要修改bind函数

attachEvent()中的回调函数的this是window，需要修改为obj

this是谁由函数的调用方式决定

1.以函数的形式调用，this是window // attachEvent()

2.以方法的形式调用，this是调用方法的对象 // addEventListener

3.以构造函数的形式调用，this是新创建的对象

4.使用call和apply调用时，this是第一个参数

想要改this的前提是函数得由我调用，而现obj.attachEvent("on"+eventStr ,callback);中函数是浏览器调用的

在attchEvent()中不传递callback而是传递一个匿名函数，这样在事件触发时，浏览器不会调用callback而是调用匿名函数，在匿名函数中来调用回调函数（由我自己调用），利用call可以指定任意对象为this

## 最终bind函数：

function bind(obj , eventStr , callback){

if(obj.addEventListener){

//如果是正常浏览器

obj.addEventListener(eventStr , callback , false);

}else{

//IE8

obj.attachEvent("on"+eventStr , function(){

callback.call(obj);

});

}

}

# 事件的传播

关于事件的传播微软和网景公司有着不同的理解

微软公司，认为事件应该是从后代元素向祖先元素传播，即从里向外传播，也就是我们所谓事件的冒泡

网景公司，认为事件应该是从祖先元素向后代元素传播，即从外向里传播，这一阶段我们称为事件的捕获

- W3C综合了两个公司的方案,将事件的传播分成了三个阶段

1.捕获阶段

- 事件从最外层的元素（document），向目标元素进行事件的捕获

- 此阶段默认不会触发事件

2.目标阶段

- 目标指的是触发事件的元素，捕获到目标元素则捕获阶段停止

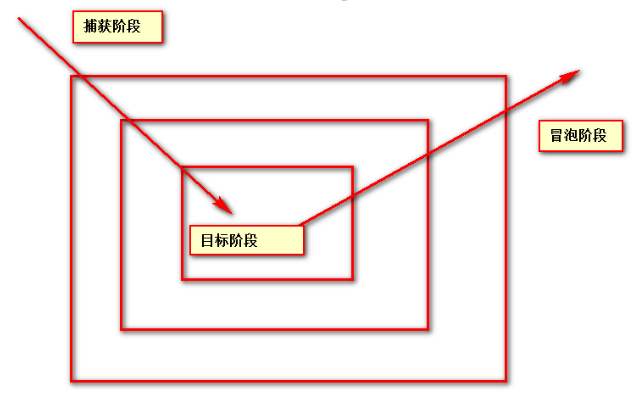
3.冒泡阶段

- 事件从目标元素向祖先元素中冒泡，此时开始触发事件

- 默认事件都是在冒泡阶段执行的

事件都是默认在冒泡阶段执行的，一般不需要在捕获阶段触发事件，如果希望在捕获阶段执行事件，则需要将addEventListener()的第三个参数修改为true

IE8及以下的浏览器没有捕获阶段，也不能设置在捕获阶段触发事件



# 拖拽

使鼠标可以在页面中来拖拽box1，box1需要开启绝对定位

\* 拖拽的流程：

\* 1.当鼠标在元素上按下时，开始拖拽 onmousedown

\* 2.当鼠标移动时，元素跟随鼠标移动 onmousemove

\* 3.当鼠标松开时，元素固定在当前位置，拖拽结束 onmouseup

//1.当鼠标在元素上按下时，开始拖拽 onmousedown

//为box1绑定一个鼠标按下的事件

var box1 = document.getElementById("box1");

box1.onmousedown = function(){

//2.当鼠标移动时，元素跟随鼠标移动 onmousemove

//为document绑定一个鼠标移动的事件

document.onmousemove = function(event){

event = event || window.event;

//获取鼠标的坐标

var left = event.clientX;

var top = event.clientY;

//修改box1的位置

box1.style.left = left + "px";

box1.style.top = top + "px";

};

//3.当鼠标松开时，元素固定在当前位置，拖拽结束 onmouseup

document.onmouseup = function(){

//当鼠标松开时，将元素固定在当前位置，取消document的onmousemove事件

document.onmousemove = null;

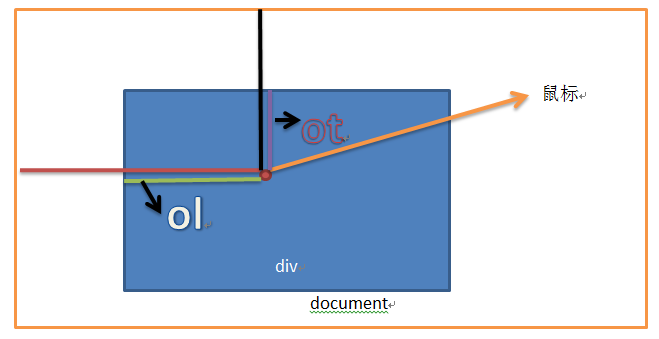
//onmouseup事件只需要执行一次，执行过一次以后就没有存在的意义了

document.onmouseup = null;

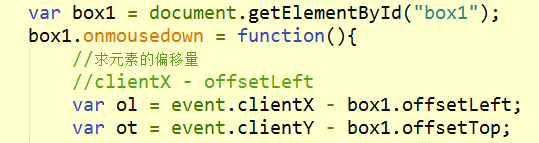
};

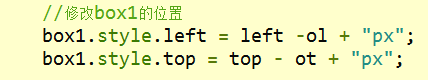
};

目前我们的元素的左上角和鼠标指针重合，我们希望我们点击元素时，鼠标和元素的相对的位置保持不变



如上图，当点击div时， div会移动使其左上角会与鼠标指针重合，如果希望鼠标和元素的相对的位置保持不变，div需要向左移动ol长度，需要向上移动ot长度，而这个长度是在点击时才有的，所以需要在点击时计算





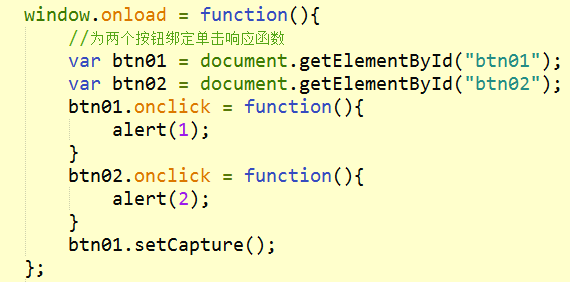
当我们在浏览器中选中一个文字或一个内容并拖动时，浏览器会自动去搜索引擎中搜索该内容，但是这个行为的出现会导致拖拽功能出现异常，这一行为是浏览器的默认行为，如果不希望发生该行为，则可以在onmousedown事件中取消默认行为：

在onmousedown最后加return false

但是这个方法对IE8不起作用，需要设置捕获来处理

# 捕获setCapture()

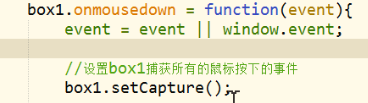
当元素设置setCapture()以后，元素将会自动将下一次鼠标点击相关的事件捕获到自身上



如上图，给btn01设置捕获后，第一次点击事件会被btn01捕获，此时不论点击什么，如点击浏览器最小化、点击btn02,、点击桌面、点击任意位置都会被btn01捕获，触发btn01的点击事件

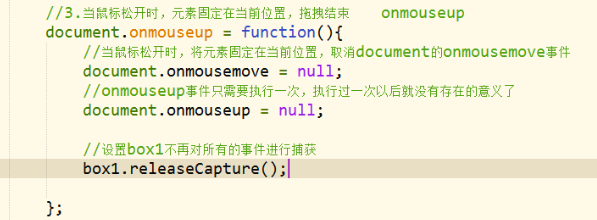
# 完成拖拽

所以可以设置box1捕获所有的鼠标按下的事件



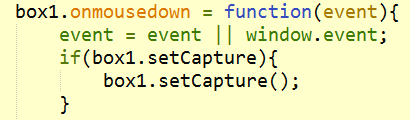
但每当鼠标按下时（不一定在box1上按下）都会触发捕获，所以当点击空白区域时，box1也会移动

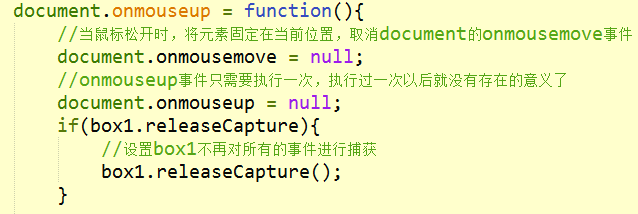
可以在鼠标松开时设置box1不再对所有的事件进行捕获



这样IE就设置完成了，但在chrome中没有setCapture()和releaseCapture()方法，所以调用会导致浏览器报错

需要判断是否有setCapture()和releaseCapture()方法





还可以将if替换成&&来判断

&&为短路的与，当第一个为false，则不看第二个

box1.setCapture && box1.setCapture();

box1.releaseCapture && box1.releaseCapture();

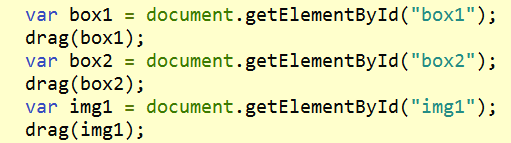
# 拖拽函数

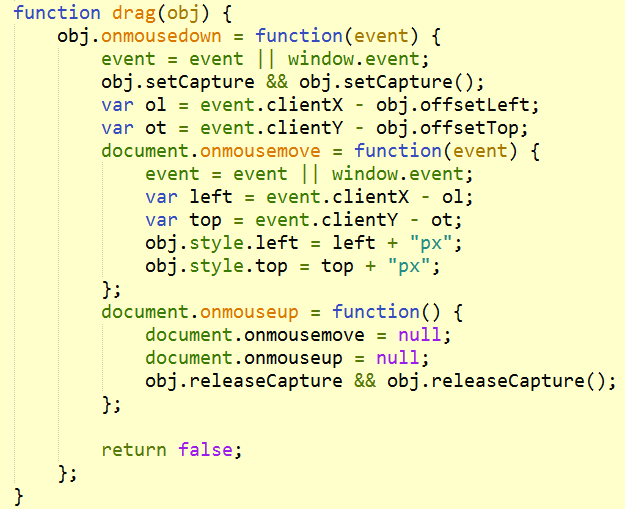
以上代码为box1的拖拽代码，而如果需要给box2设置拖拽，还需要复制一遍，将box1改为box2，可以设置一个拖拽函数，需要拖拽哪个就给哪个设置

提取一个可以拖拽任意元素的函数

\* obj 要拖拽的元素

这样想拖拽谁只要函数参数传谁就可以了





# 滚轮事件

使div可以跟随鼠标滚轮滚动来改变高度，滚轮向下滚，div变高 滚轮向上滚 div变短

onmousewheel

\* - 鼠标滚轮滚动的事件

box1.onmousewheel = function(){

alert(“滚轮滚了~~~”);

}

但是该事件火狐浏览器并不支持,在火狐中需要使用DOMMouseScroll，这个事件只能通过addEventListener()来绑定

box1.addEventListener(“DOMMouseScroll”,function(){

alert(“滚轮滚了~~~”);

},false);

但IE8会报错，还需要bind函数

bind(box1 , "DOMMouseScroll" , function(){

alert(“滚轮滚了~~~”);

});

function bind(obj , eventStr , callback){

if(obj.addEventListener){

//如果是正常浏览器

obj.addEventListener(eventStr , callback , false);

}else{

//IE8

obj.attachEvent("on"+eventStr , function(){

callback.call(obj);

});

}

}

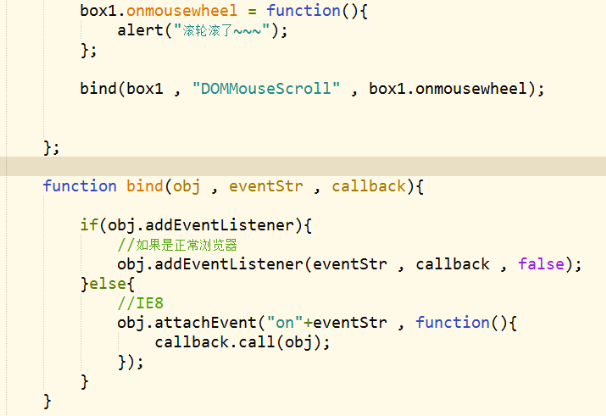
DOMMouseScroll在chrome和IE中也会绑定，但chrome和IE并不认识这个事件，所以即使绑了也不会有影响

而且这两个函数是同一个函数，可以写为一个函数：

将bind中callback设置为box1.onmousewheel

bind(box1 , "DOMMouseScroll" , box1.onmousewheel);

这样以后的代码直接写在box1.onmousewheel函数中就可以为两个函数同时设置



接着需要判断滚轮滚动的方向，还需要找事件对象

box1.onmousewheel = function(event){

在event中有一个属性wheelDelta

wheelDelta

- 事件对象中的属性，可以用来判断鼠标滚轮滚动的方向

- 向下滚 -120 向上滚 +120

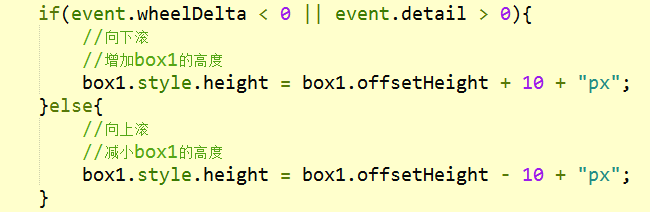
- 该属性的值并不重要，重要的是值的符号，需要通过符号来判断滚动的方向

- 但是该属性火狐浏览器并不支持

火狐中通过detail来判断方向

- 向上滚 -3 向下滚 +3

这样就可以设置div随滚轮滚动来改变高度



当页面中有滚动条时，由于滚轮滚动的默认行为会导致页面整体下移，效果就不好了

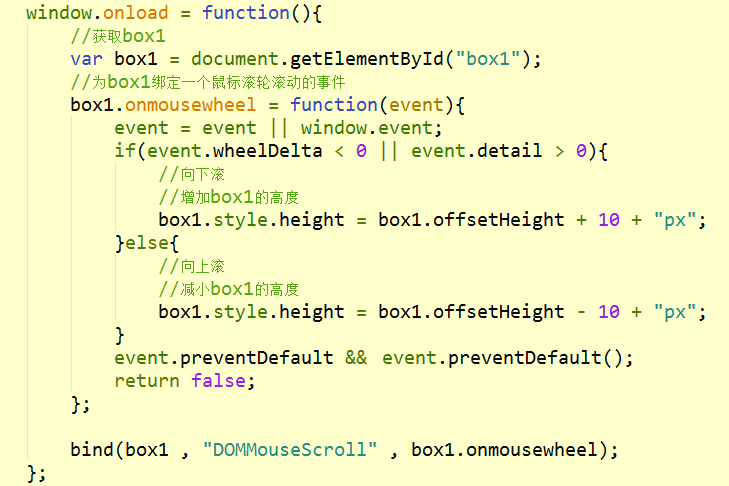
需要取消默认行为：return false;

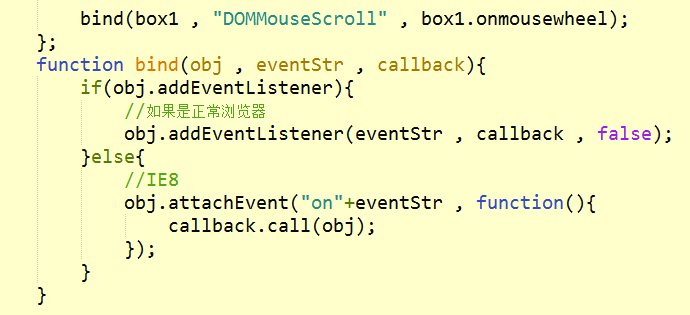
使用addEventListener()绑定的事件，不能通过return false来取消默认行为，需要调用事件对象的 preventDefault()方法来取消默认行为，但是在IE8中没有该方法

还需要判断是否有该方法

event.preventDefault &&　event.preventDefault();

这样一个完整的鼠标滚轮滚动事件就完成了





# 键盘事件

键盘事件一般都要绑定给可以获取焦点的元素（一般指表单项），或者是document

\* onkeydown

\* - 按键按下事件

\* onkeyup

\* - 按键松开事件

当按下键盘上的一个按键一直不松开时，会连续的触发onkeydown事件，但是第一次触发和第二次触发的间隔会比较长，而第二次以后会一直连续触发，这种设计是为了防止误操作

获取按的是哪个按键，还需要找event

event中有一个keyCode，可以用来获取当前按键的编码

\* 可以通过keyCode来识别用户按下的按键

\* 除了keyCode，还提供了三个属性

\* altKey

\* ctrlKey

\* shiftKey

\* - 专门用来判断alt ctrl shift是否被按下

\* 如果按下返回true，没有返回false

## 判断e和ctrl是否被同时按下 //e编码为69

if(event.keyCode == 69 && event.ctrlKey){

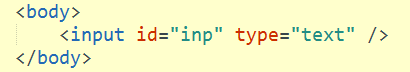
console.log("e和ctrl被按下了~~~");

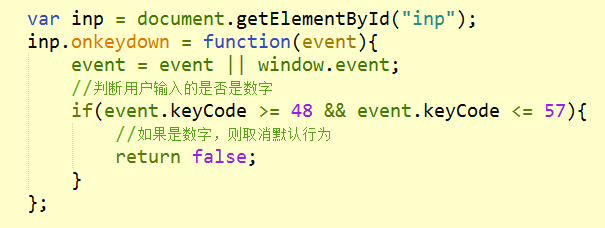
}

## 使用户不能在文本框中输入数字：

在文本框中输出文字，是onkeydown的默认行为，如果在最后取消了默认行为，则不会在文本框中显示内容

数字0-9的keyCode为48-57





但复制粘贴还可以输入进去，所以最后还需要用正则表达式验证一下

## 键盘控制div移动

