千锋教育
www.mobiletrain.org

千锋HTML5学院

# 第二阶段javascript课程课件

# 事件目录

# 什么是事件

## 什么是事件：

在日常生活中事件就是发生并得到处理的操作：事情来了，然后处理。

比如：

1、电话铃声响起（事件发生）---需要接电话（处理）

2、学生举手请教问题（有事了）---需要解答（处理）

3、咱们班有个同学被欺负了（出事了）---去给他报仇（处理）

而在JS中的事件是：用户在页面上操作，然后我们要调用函数来处理.

比如：

1，点击了登录按钮，调用登录函数执行登录操作

2，鼠标拖拽，调用函数实现拖拽

# 什么是事件

- 事件触发：

  用户在页面上操作时(如点击按钮, 鼠标滑过, 鼠标点击, 鼠标松开, 文本框获得焦点, 失去焦点等)， 就是事件触发

# 事件模式

JavaScript有两种事件实现模式：
1, 内联模式,
2, 脚本模式。

# 内联模式

内联模式：直接在HTML标签中添加事件. 这种模型是最传统简单的一种处理事件的方法。但是这种模式中事件和HTML是混写的，并没有将JS与HTML分离, 也就是说没有将行为和结构样式分离, 这种写法。

```
//在 HTML中把事件处理函数作为属性执行JS代码
<input type="button"value="按钮" onclick="alert('hello');" />
//注意单双引号


//在HTML中把事件处理函数作为属性执行 JS 函数
<input type="button"value="按钮" onclick="btnClick();" />
//执行JS 的函数
```

注: 内联模式调用函数, 则函数不能放到window.onload 里面，否则会找不到该函数。

# 脚本模式

内联模式违反了HTML与JavaScript代码层次分离的原则。所以我们可以采用另一种方式：脚本模式; 这样我们就可以在JavaScript中处理事件, 而不需要在HTML中写JS代码, 可以实现HTML和JS代码的分离, 这种脚本调用的方式在我们以后会经常用到。

使用脚本模式我们需要先获取到元素节点对象，再针对该节点对象添加事件，如我们可以采用三种方式来获得节点对象：getElementById(), getElementsByTagName(), getElementsByName()

```
//得到box节点对象
var box = document.getElementById('box');
var box = document.getElementsByTagName('input')[0];
var box = document.getElementsByName('name');
```

# 脚本模式

```
//得到box节点对象
var box = document.getElementById('box');
```

添加事件方式一： 通过匿名函数，可以直接触发对应的代码

```
//给box节点对象添加点击事件onclick
box.onclick = function() {
        console.log('Hello world!');
};
```

添加事件方式二： 通过指定的函数名赋值的方式 来执行函数

```
//给box节点对象添加点击事件onclick
box.onclick = func;
function func() {
        console.log('Hello world!');
};
```

# 事件处理函数

**所有的事件处理函数都会都有两个部分组成，on+ 事件名称;**

例如：onclick事件处理函数就是由on加上click;
onkeydown事件处理函数就是由on加上keydown;
onload事件处理函数就是由on加上load;

注: 事件处理函数一般都是小写字母

**JavaScript 可以处理的事件种类有三种：**

**1, 鼠标事件、**
**2, 键盘事件、**
**3, HTML事件。**

# 鼠标事件

1. 鼠标事件，页面所有元素都可触发鼠标事件;

**click：** 当单击鼠标按钮并在松开时触发

```
onclick = function() {
        console.log('单击了鼠标');
};
```

**dblclick：** 当双击鼠标按钮时触发。

```
ondbclick = function() {
        console.log('双击了鼠标');
};
```

**mousedown：** 当按下了鼠标还未松开时触发。

```
onmousedown = function() {
        console.log('按下鼠标');
};
```

# 鼠标事件

**mouseup：** 释放鼠标按钮时触发。

```
onmouseup = function() {
        console.log('松开了鼠标');
};
```

**mouseover：** 当鼠标移入某个元素的那一刻触发。

```
onmouseover = function() {
        console.log('鼠标移入了');
};
```

**mouseout：** 当鼠标刚移出某个元素的那一刻触发。

```
onmouseout = function() {
        console.log('鼠标移出了');
};
```

**mousemove：** 当鼠标指针在某个元素上移动时触发。

```
onmousemove = function() {
        console.log('鼠标移动了');
};
```

# 鼠标事件

**示例:**

1, 有一块空白区域,
当鼠标移动到区域内,显示"亲爱的, 我爱你",
当我鼠标移开的时候,显示"对不起, 开玩笑",
当我鼠标不停的在区域内移动的时候, 变换颜色

# 键盘事件

2. 键盘事件，在键盘上按下键时触发的事件;
(一般由window对象或者document对象调用)

**keydown：**当用户按下键盘上某个键触发，如果按住不放，会重复触发。

```
window.onkeydown = function() {
        console.log(按下了键盘上的某个键);
};
```

**keypress：**当用户按下键盘上的字符键触发，如果按住不放，会重复触发

```
window.onkeypress = function() {
        console.log('按下了键盘上的字符键');
};
```

**keyup：**当用户释放键盘上的某个键触发。

```
window.onkeyup = function() {
        console.log(松开键盘上的某个键);
};
```

# HTML事件

3. HTML事件，跟HTML页面相关的事件;

**load：当页面完全加载后触发**

```
window.onload = function() {
        console.log('页面已经加载完毕');
};
```

**unload：当页面完全卸载后触发**

```
window.onunload = function() {
        console.log('页面已经卸载完毕');
};
```

# HTML事件

**select：当用户选择文本框(input 或 textarea)中的内容触发。**

```
input.onselect = function() {
        console.log('选择了文本框中的内容');
};
```

**change：当文本框(input 或 textarea)内容改变且失去焦点后触发。**

```
input.onchange = function() {
        console.log('文本框中内容改变了');
};
```

**focus：当页面或者元素获得焦点时触发。**

```
input.onfocus = function() {
        console.log('文本框获得焦点');
};
```

**blur：当页面或元素失去焦点时触发。**

```
input.onblur = function() {
        console.log('文本框失去焦点');
};
```

# HTML事件

**submit：当用户点击提交按钮在<form>元素节点上触发。**

```
form.onsubmit = function() {
        console.log('提交form表单');
};
```

**reset：当用户点击重置按钮在<form>元素节点上触发。**

```
form.onreset = function() {
        console.log('重置form表单');
};
```

**scroll：当用户滚动带滚动条的元素时触发。**

```
window.onscroll= function() {
        console.log('滚动了滚动条了');
};
```

# HTML事件

## 示例:

1, 有一个文本框, 默认文字为”我是初始化字符串”, 当文字被选择时, 在红色区域显示”文本被选中”, 当文字修改后, 显示“文本被修改”

# 事件处理的三个组成部分

事件处理由三个部分组成：
1, 触发事件的节点对象
2, 事件处理函数
3, 事件执行函数。

例如：单击文档任意处。

```
document.onclick = function(){
        console.log('单击了文档页面的某一个地方');
};
```

在上面的程序中：
document：是触发事件的对象, 表示触发事件的元素所在区域;
onclick：表示一个事件处理函数(其中click表示一个事件类型:单击)
function(){}：匿名函数是被执行的函数, 用于触发事件后执行;

# this关键字

在JS事件中, this表示触发事件的元素节点对象;

```
var box = document.getElementById('box');
box.onclick = function() {
    console.log(this.nodeName);  //this表示box对象
};
```

通过for循环添加事件, 使用this

```
var aInput = document.getElementsByTagName('input');
for (var i=0; i<aInput.length; i++) {
    aInput[i].onclick = function() {
        console.log(this.value);
        //这里的this表示被点击的那个input元素节点对象
    };
}
```

# event对象的获取

event对象(事件对象)是在触发事件时, 浏览器会通过函数把事件对象作为参数传递过来, 在事件触发执行函数时一般会得到一个隐藏的参数, 该参数也是放在arguments数组中.

```
//普通函数
function func() {
    console.log(arguments.length); //1，得到传递的参数
}
func(“hello”);

//事件绑定的执行函数
box.onclick = function(){
     console.log(arguments.length); //1，得到一个隐藏参数
};
```

【注】通过上面两组函数中，我们发现，通过事件绑定的执行函数是可以得到一个隐藏参数的。 说明，浏览器会自动分配一个参数，这个隐藏参数其实就是**event对象(事件对象)**。

# event对象的获取

```
//arguments[0]
box.onclick = function() {
    //获得该事件对象([object MouseEvent])
    console.log(arguments[0]);
};
```

我们还可以使用更简单的获取事件对象的方式: 通过给函数添加一个参数

```
//接受事件对象，名称不一定非要evt(这里的evt是形参,也可以自己给定其他名称)
box.onclick = function(evt){
    console.log(evt);   //[object MouseEvent]
};
```

# event对象的获取

通过事件的执行函数传入的event对象(事件对象) 不是在所有浏览器都有值，在IE浏览器上event对象并没有传过来, 这里我们要用window.event来获取, 而在火狐浏览器上window.event无法获取, 而谷歌浏览器支持event事件传参和window.event两种, 为了兼容所有浏览器, 我们使用以下方式来得到event事件对象

```
box.onclick = function(evt){
    if (evt) {
        alert("evt: " + evt);
    }
    else {
        alert("window.event: " + window.event);
    }
};
```

# event对象的获取

这里我们还有更简单的获取方式

```
box.onclick = function(evt){
    var oEvent = evt || window.event; //获取到event对象(事件对象)
    alert(oEvent);
};
```

其中window.event中的window可以省略, 最终我们可以写成:

```
box.onclick = function(evt){
    var oEvent  = evt || event; //获取到event对象(事件对象)
    alert(oEvent);
};
```

# event对象的属性

鼠标事件是 Web 上面最常用的一类事件，因为鼠标是Web页面最主要的外部设备。通过事件对象可以获取到鼠标按钮信息和屏幕坐标获取等。

## 1.button属性

| 值 | 说明 |
| --- | --- |
| 0 | 表示主鼠标按钮(常规一般是鼠标左键) |
| 1 | 表示中间的鼠标按钮(鼠标滚轮按钮) |
| 2 | 表示次鼠标按钮(常规一般是鼠标右键) |

```
document.onclick = function(evt) {
        var oEvent  = evt || event;
        console.log(oEvent  .button);
};
```

# event对象的属性

2.可视区坐标及屏幕坐标

事件对象提供了两组来获取浏览器坐标的属性，一组是页面可视区坐标，另一组是屏幕区坐标。

坐标属性

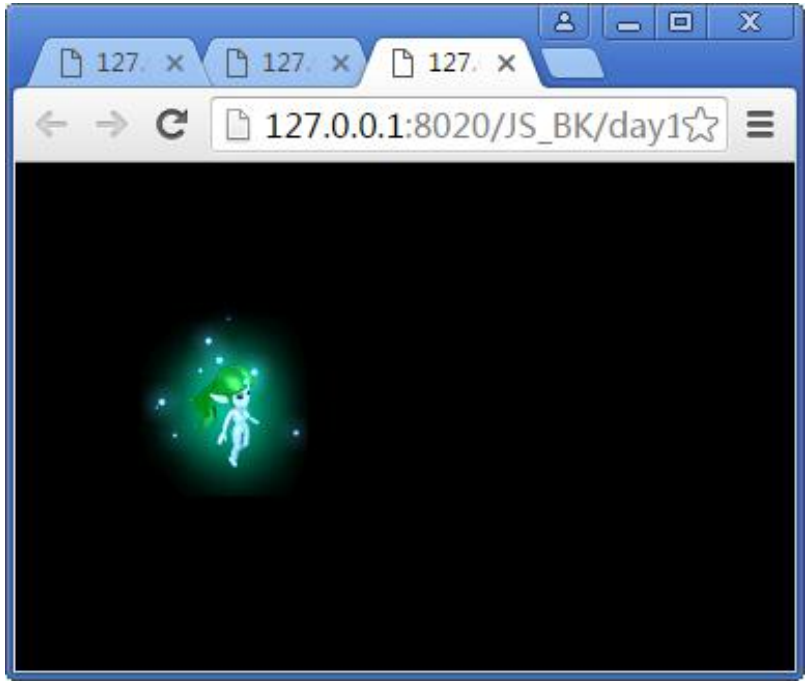
| 属性 | 说明 |
| --- | --- |
| clientX | 可视区 X 坐标，距离左边框的位置 |
| clientY | 可视区 Y 坐标，距离上边框的位置 |
| screenX | 屏幕区 X 坐标，距离左屏幕的位置 |
| screenY | 屏幕区 Y 坐标，距离上屏幕的位置 |

```
document.onclick = function(evt) {
        var oEvent  = evt || event;
         console.log(oEvent.clientX + ',' + oEvent.clientY);
         console.log(oEvent.screenX + ',' + oEvent.screenY);
};
```

# event对象的属性

## 示例:

1, 鼠标点击某处, 让精灵移动到该处 (如下图)
2, 鼠标移动时, 让精灵跟随鼠标移动



# event对象的属性/鼠标事件(修改键)

有时，我们需要通过键盘上的某些键来配合鼠标来触发一些特殊的事件。这些键为： Shfit、Ctrl、Alt 和 Meat(Windows 中就是 Windows 键，苹果机中是 Cmd 键)，它们经常被用来修改鼠标事件和行为，所以叫修改键。

修改键属性

| 属性 | 说明 |
| --- | --- |
| shiftKey | 判断是否按下了 Shfit 键 |
| ctrlKey | 判断是否按下了 ctrlKey 键 |
| altKey | 判断是否按下了 alt 键 |
| metaKey | 判断是否按下了 windows 键，IE 不支持 |

# event对象的属性/键盘事件(键码)

## 1. 键码 : keyCode属性

所有按键, 包括功能键(control, alt,shift, tab, 方向键等), (不包括亮度,音量..的按键)在发生 keydown和keyup 事件时，event对象的keyCode属性中会包含一个代码，与键盘上一个特定的键对应。对数字字母字符集，keyCode属性的值与 ASCII 码中对应**大写字母或数字的编码相同,** 为大写字母。

```
document.onkeydown = function(evt) {
    var oEvent = evt || window.event;
    alert(oEvent.keyCode);  //按任意键，得到相应的 keyCode
};
```

# event对象的属性/键盘事件(字符码)

## 2.字符编码：charCode属性

Firefox、Chrome 和 Safari 的event对象支持charCode属性，这个属性只有在发生keypress事件时才包含值，而且这个值是按下的那个键所代表字符的 ASCII 编码。此时的keyCode通常等于0或者也可能等于所按键的编码

**( 键盘上的数字, 字母(区分大小写), 空格, 回车)**

```
document.onkeypress = function(evt) {
        var oEvent = evt || event;
        console.log(oEvent.charCode);
}
```

注：可以使用 String.fromCharCode()将 ASCII 编码转换成实际的字符

# 练习

## 练习:

有一个红色的div块, 点击鼠标
1, 如果我按下ctrl+c变换颜色
2, 如果我按下ctrl + shift + r 重置颜色,回复初始颜色
3, 如果我按下向上箭头,向上移动, 同理还可以向下,左,右移动
4, 如果我按下ctrl + 上下左右,走的步数变大

# 事件的目标(target)

target: 目标对象,存放绑定事件的元素节点对象

```
document.onclick = function(evt) {
    var oEvent = evt || event;
    console.log("document: " + oEvent.target); //HTMLHtmlElement
}

box.onclick = function(evt) {
    var oEvent = evt || event;
    console.log("box: " + oEvent.target); //HTMLDivElement
}

oInput.onclick = function(evt) {
    var oEvent = evt || event;
    console.log("input: " + oEvent.target); //HTMLInputElement
}
```

# 事件的冒泡

## 事件流

事件流是描述的从页面接受事件的顺序，当几个都具有事件的元素层叠在一起的时候， 那么你点击其中一个元素，并不是只有当前被点击的元素会触发事件，而层叠在你点击范围 的所有元素都会触发事件。事件流包括两种模式：冒泡和捕获。

## 事件冒泡

是从里往外逐个触发。事件捕获，是从外往里逐个触发。那么现代的浏览器 默认情况下都是冒泡模型

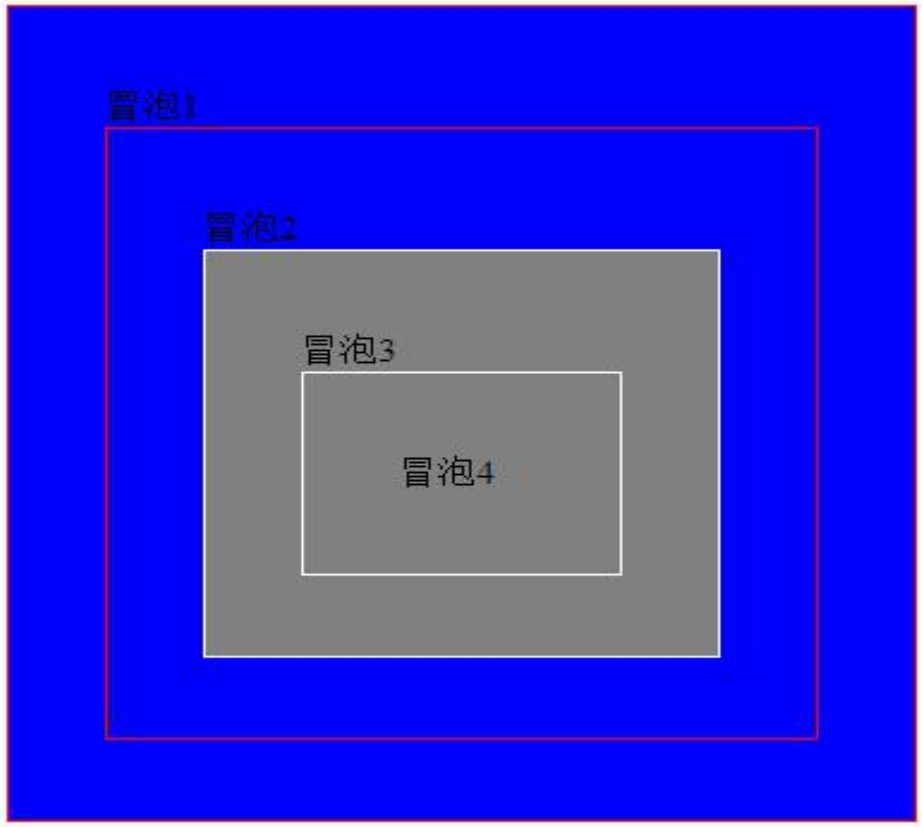
# 事件的冒泡

事件的冒泡: 指的是在页面上层节点触发的事件会继续传递给下层节点, 这种传递方式,我们称之为事件的冒泡传递;

```
document.onclick=function(){
    alert('我是 document');
};
document.documentElement.onclick=function() {
    alert('我是 html');
};
document.body.onclick= function(){
    alert('我是 body');
};
document.getElementById('box').onclick=function() {
    alert('我是 div');
};
document.getElementsByTagName('input')[0].onclick= function(){
    alert('我是 input');
};
```

# 事件的冒泡

点击冒泡2, 会触发到下层的, 如下图



# 事件的冒泡

但是一般我们只在指定的节点上添加事件, 而不想让其传递到下层节点触发事件, 这样我们就需要阻止冒泡;

阻止冒泡的方式有两种:
( 在指定不想再继续传递事件的节点的事件执行函数中使用)

```
//1, 取消冒泡
    oEvent.cancelBubble = true;
//2, 停止传播
    oEvent.stopPropagation();
```

```
document.getElementsByTagName('input')[0].onclick= function(){
    var oEvent = evt || window.event;
    //可以通过下述两种方式取消事件冒泡
    //oEvent.cancelBubble = true; //1, 取消冒泡
    oEvent.stopPropagation(); //2, 停止传播
};
```

# 作业

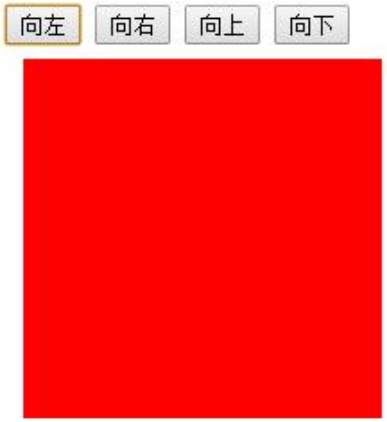
1, 课堂案例敲一遍

2, 如下图, 在输入框中输入用户名和密码,

1, 当鼠标失去焦点时: 检测用户名长度至少6位, 且只能为数字和字母;检测密码长度至少8位

2, 给登录按钮添加点击事件: 点击后弹出用户名和密码


3, 有一个红色div块，有四个button, 上下左右, 当我点击对应的按钮,这个块就进行对应的行走。



# 作业

4, 制作下拉菜单:

1, 最开始效果如左图1
2, 鼠标滑过”请选择游戏名称”区域时,效果如图2
3, 鼠标滑过下拉选项区域时, 让下拉选项可以继续显示
4, 鼠标在选项中滑过时, 显示效果图3
5, 选择某一项, 将顶部的名称改成你选择的游戏名称

# 作业

5, 制作如下图效果: 当鼠标滑过小图片时, 让大图片也显示该图