目录

```
var target = e.target || e.srcElement; //兼容写法:
```

- 阻止事件冒泡:

```
window.event.cancelBubble = true; // ie  和 谷歌支持
e.stopPropagation();  //  火狐和谷歌支持  ie不支持

//三木运算符解决兼容性
e.cancelBubble ? e.cancelBubble = true : e.stopPropagation();
```

  - 不考虑IE浏览器,直接写 `e.stopPropagation();`

1.节点操作

- `document.write()`
  - 会覆盖原有的内容

- `document.createElement()`：创建元素节点
  - 格式：`document.createElement("标签名")`;
  - 功能: 创建元素节点

```
var oP = document.createElement("p");   //一个空的P标签
```

    - 只是创建节点(没有文本和属性).

- `document.createTextNode()`：创建文本节点
  - 格式：`document.createTextNode("文本")`;
  - 功能：创建文本节点,纯文本。(不解析标签)

- `node.appendChild(新节点)`：末尾插入节点
  - 格式：`node1.appendChild(node2)`;
  - 功能：将`node2`节点插入到`node1`子节点的末尾

```
oDiv.appendChild(oP);
```

- `node.insertBefore(新节点,子节点)`：节点的某一个子节点前插入一个新节点
  - 格式：`node1.insertBefore(node2 , node1子节点)`
  - 功能：将`node2`节点插入到 指定`node1`子节点 的前面

```
oDiv.insertBefore(oP,oEm[0]); //将op节点插入到EM节点前
```

- `node.replaceChild(新节点,子节点)`：将节点的某一个子节点替换
  - 格式：`node1.replaceChild(node2 , node1子节点)`

- 功能：`node2`节点替换`node1`指定的子节点

```
oDiv.replaceChild(oP , oEm[0]); //将EM节点替换为OP节点
```

- `node.cloneNode()`：复制节点(包括节点属性)
  - 格式1：`node.cloneNode()`，克隆节点本身(包括属性)
  - 格式2：`node.cloneNode(true)`，克隆节点本身和子节点
  - 功能：复制节点
  - 返回值：克隆出来的新节点

- `node.removeChild(子节点)`：删除子节点
  - 格式：`node1.removeChhild(node1子节点)`
  - 功能：将指定的`node1`的子节点删除

- 扩展：
  - `parentNode`是获取父节点，`oP.parentNode`
  - `body`节点的获取：`document.body`
- 节点操作案例(2_节点操作案例.html)
  - 增加 删除 拷贝

2.找到点击按钮的下标

- 点击按钮输出当前按钮的下标 (this关键字)

```
// 刻舟求剑(错误)
for(var i = 0; i <oBtns.length; i++){
  oBtns[i].onclick = function(){
    alert(i);
  }
}
```

  - 点击事件执行时，`i`的值已经变成`oBtns.length - 1`，输出的都是同一个

3.this关键字

- `this`概念：任何一个**函数**系统都会**内置**一个叫`this`的变量。
  - `this`变量**存储的是地址**，是**当前函数主人的地址。**
  - `this`**永远指向当前函数的主人。**函数的主人要通过当前上下文去判断
- this指向常见的情况：
  1. 一般情况下,**对象的方法中**的`this`，**指向的是对象**
  2. **全局函数**，如果**函数没有主人**，**默认指向**`window`**对象**
  3. **事件函数**`this`**指向 点击的按钮对象**(元素节点)
  4. `this`在**延时器**中，**指向**`window`**对象**

- `this` 点击按钮输出当前按钮的下标

```
for(var i = 0; i <oBtns.length; i++){
  oBtns[i].setAttribute("index",i);
  oBtns[i].onclick = function(){
    alert(this.getAttribute("index"));
  }
}
alert("循环结束了:" + i);
```

5.选项卡案例(5_选项卡案例.html)

- 点击按钮切换内容
- 思路:
  - 点击按钮后清除所有内容
  - 再通过 `this` 关键字添加内容

6.offset系列方法

- IE: `node.getcurrentStyle["width"]` 谷歌: `getComputedStyle(node)["width"]` 获取到的，是**样式里书写的'width'的值**，并不会计算边框、padding

- `node.offsetWidth` `node.offsetHeight` ： 返回**数字**
  - 获取的是 `border` + `padding` + `content` 的值
  - 获取的值是数字，省略了 `px`;

- `node.offsetLeft` `node.offsetTop`： 返回**数字**
  - 获取**元素和第一个有定位的父节点之间的距离**;没有定位，则计算和浏览器的距离
    - 计算 `node` 的 `border` 距离定位元素之间的距离，`node` 的 `margin` 也会被计算其中

- `node.offsetParent`
  - 获取元素和**第一个有定位的父节点。**
    1. 当容器元素的 `style.display` 被设置为 `"none"` 时（译注：IE和Opera除外）， `offsetParent` 属性 返回 `null` 。
    2. `body` 节点 `offsetParent` 为 `null`

7.文档碎片

- 测试代码所用时长:

```
console.time("text1");  // text1为自定义参数
// 代码区域...
console.timeEnd("text1");
// 控制台查看返回时间
```

- 例子：在页面上创建十万个 `div`

- 每新建一个节点，就在页面上进行一次插入操作(十万次插入)

```
console.time("test1");
for(var i = 0; i < 100000; i++){
  var newElement = document.createElement("div");
  document.body.appendChild(newElement);
}
console.timeEnd("test1");  // 167ms
```

- 文档碎片，创建十万个 `div`
  - 创建一个节点，再创建十万个节点放进这一个节点里，最后将一个节点插入页面(只执行了一次插入)

```
console.time("test2");
var node = document.createElement("div");
for(var i = 0; i < 10000; i++){
    var oDiv = document.createElement("div");
    node.appendChild(oDiv);
}
document.body.appendChild(node);
console.timeEnd("test2");  // 14ms
```

8.数组和对象遍历的方法

- 数组：
  - `for循环`
  - `for...in` 快速遍历 (少用)

```
for(var i in arr){
  document.write(i + arr[i]); //i代表的是数组的下标
}
```

  - `forEach(function(item,index,arr){ })`
- 对象：
  - `for...in` 快速遍历

```
for(var i in person){
  document.write(i + person[i]);
  //i代表的是对象的下标，对象的属性名
}
```

9.认识事件和绑定事件的方式

- 什么是事件：事件是发生并得到处理的操作，即：事件发生，然后处理
  - 例：电话响了 - 挂断/接听

- 绑定事件方式
  1. 内联模式

```
<button onclick="show()"></button> <!-- 写在内联 -->
```

2. 外联模式/脚本模式（多使用） //写在script中

- `外联模式/脚本模式`-绑定事件格式：
  - `node.on` + 事件类型 = 匿名函数。

```
oDiv.onclick = function(){/* 函数体 */}
```

    - click 事件类型
    - onclick 事件处理函数
      - 绑定的事件函数可以修改或者调用,因为它本身就是一个函数

```
oDiv.onmousemove = oDiv.onclick
```

10.事件类型的种类

- 鼠标事件
  - `click` 单击
  - `dblclick` 双击
  - `mouseover` 鼠标移入(经过子节点重复触发)
  - `mouseout` 鼠标移出(经过子节点重复触发)
  - `mousemove` 鼠标移动(会不停触发)
  - `mouseenter` 鼠标移入
  - `mouseleave` 鼠标移出
  - `mousedown` 鼠标按下(不会重复触发)
  - `mouseup` 鼠标抬起
    - `mouseover/mouseout` 和 `mouseenter/mouseleave` 的区别
      - `mouseover/mouseout` 经过子节点会重复触发
      - `mouseenter/mouseleave`经过子节点不会重复触发。 IE8以后才有

  - `mousewheel` **鼠标滚轮滚动**
    - Firefox 3.5+ 中提供了另外一个等同的事件：`'DOMMouseScroll'` （事件和事件属性的测试案例）。
    - 兼容写法: 写两个`addEventListener('事件名称', 函数, false(同步))`

```
window.addEventListener('mousewheel',scroll,false)
window.addEventListener('DOMMouseScroll',scroll,false)
```

    - **判断鼠标滚轮的方向**，有着两个：一是谷歌、IE；二是火狐
      - 在谷歌、IE中，给我们提供了`onmousewheel`方法，该方法给我们提过了一个`e.wheelDelta`属性，该属性的返回值：正值与负值，其中正值表示滚轮向上滚动；负值表示滚轮向下滚动。

```
node.onmousewheel = function(ev){
  var e = ev || window.event;
  e.wheelDelta
}
```

- 火狐中，提供了`DOMMouseScroll`方法，该方法给我们提供了一个`detail`属性，该属性的返回值：正值与负值，其中正值表示滚轮向下滚动；负值表示滚轮向上滚动。

```
node.DOMMouseScroll = function(ev){
  var e = ev || window.event;
  e.detail
}
```

- 键盘事件 (表单元素,全局window)
  - `keydown` ： 键盘按下
  - `keyup` ： 键盘抬起
  - `keypress` ： 键盘按下 (只支持字符键)
    - `keydown`和`keypress`如果按下不放手，会一直触发，触发时间是 文本输入之前

  - 扩展： `document.title` ： 获取`title`标签
    - 不用加innerHTML,直接可以改变内容

  - 键盘事件在window、文本框节点下可用，但是在div这种节点上不可用。
  - 键盘事件在中文输入法时会触发两次

- HTML事件(window、表单)
  - window事件
    - `load`： 当页面加载完成以后会触发
    - `unload`： 当前页面解构的时候触发(刷新页面,关闭当前页面) IE浏览器兼容
    - `scroll`： 页面滚动触发 (不停的触发)
      - `scroll`事件也可以用在其他元素节点上。
    - `resize`： 窗口大小发生变化时触发 (不停的触发)
  - 表单事件
    - `blur`： 失去焦点
    - `focus`： 获取焦点
    - `select`： 当我们在输入框选中文本会触发
    - `change`： 对输入框文本进行修改 并 失去焦点
    - `input`： 当表单的值发生变化时触发,只要发生变化就会触发
    - `submit`： 点击`form`表单下的`submit`按钮
    - `reset`： 点击`form`表单下的`reset`按钮
      - 表单事件必须添加在`form`表单元素上

11.事件对象

- 事件绑定：`node.on + 事件类型 = 匿名函数;`
  - 系统会在**事件绑定完成的时，生成一个事件对象。**
    - 触发事件的时候，系统会自动调用绑定事件的函数。**将事件对象当做第一个参数传入**
- 拿到事件对象的方法
  1. `arguments[0]` ： **少使用**,看不懂什么意思
  2. 函数的第一个形参会储存第一个传入的参数 IE8以下不兼容

```
function show(ev){
  alert(ev);
}
//IE8以下
window.event;
```

  - 兼容写法

```
var e = ev || window.event  //或运算短路操作 / 三目运算符
```

12.鼠标事件对象的属性

- `e.button` ： 只在鼠标事件中拥有
  - 0 左键， 1 滚轮， 2 右键
- 获取当前鼠标位置：(原点位置不一样)
  - `clientX,clientY` 原点位置: **可视窗口的左上角为原点**
  - `pageX,pageY` 原点位置: **整个页面左上角 [包含滚动距离]**
  - `screenX,screenY` 原点位置: **电脑屏幕的左上角**
  - `offsetX,offsetY` 获取**鼠标距离target元素的位置**
    - 返回值：纯数字.
    - 以上都是鼠标事件的属性
  - 扩展:
    `window.screenX/Y` ： 获取的是**浏览器距离屏幕左上的位置**

13.跟随鼠标移动的提示框

- 跟随移动的显示框显示层级会比事件元素高，要偏移点距离，使鼠标可以点击事件元素。

14.事件对象的属性（所有事件对象都可以，主要组合使用）修改键

- `shiftKey`： 按下`shift`时`e.shiftkey`为`true`,默认为`false`
- `altKey`
- `ctrlKey`
- `metaKey`： `windows`系统按下`windows`(开始)键为`true`,macos(苹果)系统 按下`command`键为`true`
  - 和别的操作进行组合,形成一些快捷键操作。

15.键盘事件对象的属性

- `e.keyCode` 键码 不同的浏览器支持不同的键码获取方式，注意兼容
  - 格式： `var which = e.which || e.keyCode;`
  - 返回值：键码**返回**的是**大写的ASCII码值。不区分大小写。**
    - `keyCode` 只在 `keydown` 下支持。
- `e.charCode` 字符码
  - 格式： `var which = e.which || e.charCode;`
  - 返回值：字符码**返回的是ASCII码值。区分大小写。**
    - `e.charCode` 只在 `keypress` 下支持。只支持字符键。
- 返回值都是数字。
- 扩展： 微博发布快捷键

16.目标对象和this

- `target` ： 目标对象/触发对象 (这个事件是由谁而起的。)

```
e.srcElement;  //IE8以下不兼容
var target = e.target || e.srcElement; //兼容写法:
```

  - **子节点层级 会比 父节点高**，给标签添加事件，**用target可以找到是在哪一个子节点上触发**的。
- `this` : 永远指向当前函数的主人

17.事件冒泡

```
// 写一个三个div嵌套结构
// 给每个div添加点击事件,输出对应的ID

var aDivs = document.getElementsByTagName("div");
for(var i = 0; i < aDivs.length; i++){
    aDivs[i].onclick = function(){
        alert(this.id);
    }
}
// 最外层div点击后    弹出div1
// 中间一层点击后     弹出div2 div1
// 里面一层点击后     依次弹出div3 div2 div1
```

- 浏览器天生的一个特点: 事件流
  - 事件冒泡： **由里向外,逐级触发。**
  - 事件捕获： **由外向里,逐级触发。**

  - 事件冒泡的问题：点击里面的节点，会将外面的父节点点击事件一同触发。
    - 阻止事件冒泡

```
window.event.cancelBubble = true; // ie  和 谷歌支持
e.stopPropagation();  //  火狐和谷歌支持  ie不支持

//三木运算符解决兼容性
e.cancelBubble ? e.cancelBubble = true : e.stopPropagation();
```

  - 都是事件对象的属性和方法
  - 和阻止默认行为一样, IE的都已经弃用, 使用`e.stopPropagation()`

- 事件练习
  1. 跟随鼠标移动的gif图
  2. 跟随鼠标移动的一串div