目录

[javascript进阶 1](#_Toc25148)

[复习： 3](#_Toc29877)

[一、 IE中的高级绑定方式 7](#_Toc4337)

[二、 DOM2级与DOM0级绑定方式之间的区别 7](#_Toc8824)

[2.1 绑定数量 7](#_Toc22429)

[2.2 执行顺序 8](#_Toc26031)

[2.3 是否可以同时存在 8](#_Toc13036)

[2.4 this指向 8](#_Toc9486)

[三、 attachEvnet与DOM0级之间的区别 8](#_Toc16524)

[3.1 绑定数量 8](#_Toc23058)

[3.2 执行顺序 8](#_Toc17515)

[3.3 是否可以同时存在 8](#_Toc12242)

[3.4 this指向 8](#_Toc16567)

[四、 事件的移除方式 8](#_Toc23914)

[4.1 DOM2事件移除方式 8](#_Toc32542)

[4.2 IE中的移除方式 9](#_Toc16584)

[4.3 事件绑定函数 10](#_Toc7400)

[五、 事件对象与属性 10](#_Toc9443)

[5.1 offsetX、 offsetY 11](#_Toc30927)

[5.2 clientX、 clientY 12](#_Toc13988)

[5.2 pageX、 pageY 12](#_Toc6350)

[5.3 screenX、 screenY 13](#_Toc1537)

[下午复习： 14](#_Toc3906)

[六、 IE中的事件对象 15](#_Toc7460)

[七、 阻止冒泡 16](#_Toc25088)

[八、 阻止默认事件 16](#_Toc17809)

[九、 快捷尺寸 17](#_Toc19295)

[9.1 clientWidth、 clientHeight 17](#_Toc5027)

[9.2 offsetWidth、offsetHeight 18](#_Toc11327)

[9.3 clientLeft、 clientTop 19](#_Toc21216)

[十、 jquery中的快捷尺寸 19](#_Toc32658)

[十一、 定位父元素与定位值 20](#_Toc492)

[10.1 定位父元素 20](#_Toc25339)

[10.2 定位值 21](#_Toc17194)

[10.3 jquery中的定位值 22](#_Toc9587)

# 复习：

jacascript这门语言是由DOM、BOM、ECMAScript组成

dom的组成：

dom是由节点组成， 一共分为12种

1. 元素类型

3 文本类型

8 注释类型

9 文档类型

节点属性：

nodeType: 该属性标记的是节点的类型

nodeName: 该属性标记的是节点的名称

nodeValue: 该属性标记的是节点的值

childNodes属性：

指向调用该属性的节点的所有子节点的集合，是一个类数组对象，具体类型是NodeList

节点关系：

一共分为三种：

父子：

father.childNodes: 获取所有的子节点

father.firstChild: 获取第一个子节点

father.lastChild: 获取最后一个子节点

子父：

child.parentNode: 获取父节点

兄弟：

node.nextSibling: 获取下一个兄弟节点

node.previousSibling: 获取上一个兄弟节点

元素操作：

创建元素：document.createElement(type)

上树：father.appendChild(child)

下树：father.removeChild(child)

插入：father.insertBefore(newChild, oldChild)

替换：father.replaceChild(newChild, oldChild)

克隆：node.cloneNode(bool)

jquery中的节点操作：

创建元素：

我们可以利用$函数的功能创建一个元素

$(<div class=’aaa’>content</div>)

上树的方法：

jquery中上树的方法有很多

父元素选择子元素：

$(father).append(child)

$(father).prepend(child)

子元素选择父元素：

$(child).appendTo(father)

$(child).prependTo(father)

兄弟选择兄弟：

$(dage).after(erdi)

$(erdi).before(dage)

$(sandi).insertAfter(erge)

$(sandi).insertBefore(sidi)

wrap: 用于将匹配到元素的外层添加一层元素

wrapAll: 用于将匹配到的所有元素的外层添加一层元素

unwrap: 用于将匹配到元素的外层去掉一层元素

replaceWith: 用于将匹配到元素替换掉指定的元素

replaceAll: 用指定的元素替换掉匹配到的元素

empty: 清空后代， 自己还在

remove: 表示自杀， 自己和后代都没有了

clone: 接收一个参数是布尔值，默认是false, 天生连同子节点一起复制，如果传递的true，连同事件一起复制

call和apply:

这两个方法都是执行函数并改变函数的作用域

call:

接收多个参数

第一个参数： 要改变的作用域

从第二个参数开始， 都是原函数所需要的参数

apply:

接收两个参数

第一个参数：要改变的作用域

第二个参数： 是一个数组，数组中的每一项都是原函数所需要的参数

事件流程：

分为捕获和冒泡：

捕获：事件从最顶层的元素开始执行， 一层一层往下执行，直到最精确定的元素

冒泡：事件从最精确的元素开始执行， 一层一层往上执行，直到最顶层的元素

当元素处于最精确元素的时候， 是不区分捕获和冒泡

animate函数：

function animate(dom, json, time, callBack) {

// 定义计数器

var count = 0;

// 定义间隔时间

var interval = 20;

// 定义总次数

var allCount = time / interval;

// 定义对象 保存元素的初始值

var nowJson = {};

for (var i in json) {

nowJson[i] = parseInt(getComputedStyle(dom)[i]);

}

// 定义对象 保存元素的步长值

var stepJson = {};

for (var i in json) {

setpJson[i] = (json[i] - nowJson[i]) / allCount;

}

var timer = setInterval(function() {

count++;

// 不断的改变dom的状态

for (var i in json) {

dom.style[i] = nowJson[i] + stepJson[i] \* count + “px”;

}

// 判断是否到达目标值

if (count >= allCount) {

// 拉终

for (var i in json) {

dom.style[i] = json[i] + “px”;

}

// 停表

clearInterval(timer);

// 执行回调函数

callBack && callBack.call(dom);

}

})

}

# IE中的高级绑定方式

经过测试， 在IE中不支持addEventListener，支持attachEvnet绑定事件方式

适用方式：

dom.attachEvent(type, fn)

type: 事件类型， 注意是带on的事件， 比如：onclick、onmousedown

没有第三个参数， 也就是说不支持捕获

举例：

|  |
| --- |
| 1. // 获取元素 2. var box = document.getElementById("box"); 3. var box1 = document.getElementById("box1"); 4. var box2 = document.getElementById("box2"); 5. // attachEvent 6. box.attachEvent("onclick", function() { 7. console.log("box"); 8. }) 9. box1.attachEvent("onclick", function() { 10. console.log("box1") 11. }) 12. box2.attachEvent("onclick", function() { 13. console.log("box2") 14. }) |

结果：

|  |
| --- |
|  |

# DOM2级与DOM0级绑定方式之间的区别

## 2.1 绑定数量

dom0级事件只可以给一个元素的一个事件绑定一个函数

dom2级事件可以给一个元素的一个事件绑定多个事件函数

## 2.2 执行顺序

是按照代码的绑定顺序

## 2.3 是否可以同时存在

是

## 2.4 this指向

都指向触发事件的对象

# attachEvnet与DOM0级之间的区别

## 3.1 绑定数量

dom0级: 只可以给一个元素的一个事件绑定一个函数

attachEvent 可以给一个元素的一个事件绑定多个函数

## 3.2 执行顺序

优先执行dom0级， 再逆序执行attachEvent事件

## 3.3 是否可以同时存在

是

## 3.4 this指向

dom0级指向触发事件的对象

attachEvent指向window

# 事件的移除方式

## 4.1 DOM2事件移除方式

事件名称: removeEventListener

该方法用于移除addEventListener绑定的事件

使用方式：

dom.removeEventListener(type, fn, bool)

type: 事件类型

fn: 执行的函数

bool： 决定要移除的是捕获阶段还是冒泡阶段

错误的移除方式：

|  |
| --- |
| 1. // 获取元素 2. var box = document.getElementById("box"); 3. var btn = document.getElementById("btn"); 4. // 使用DOM2级绑定方式 5. box.addEventListener("click", function() { 6. console.log("事件执行了"); 7. }) 8. // 当点击btn的时候移除事件 9. btn.onclick = function() { 10. box.removeEventListener("click", function() { 11. console.log("事件执行了"); 12. }) 13. } |

正确的移除方式：

|  |
| --- |
| // 提取到外部并且是声明函数  function click() {  console.log("事件执行了");  }  // 使用DOM2级绑定方式  box.addEventListener("click", click);  // 当点击btn的时候移除事件  btn.onclick = function() {  box.removeEventListener("click", click)  } |

## 4.2 IE中的移除方式

事件名称: detachEvent

该方法用于移除attachEvent绑定的事件

使用方式与addEventListener是一致的，只不过没有了第三个参数而已

举例：

|  |
| --- |
| // 获取元素  var box = document.getElementById("box");  var btn = document.getElementById("btn");  // 提取到外部并且是声明函数  function click() {  console.log("事件执行了");  }  // 使用DOM2级绑定方式  box.attachEvent("onclick", click);  // 当点击btn的时候移除事件  btn.onclick = function() {  box.detachEvent("onclick", click);  } |

结果：

|  |
| --- |
|  |

当再次点击box的时候， 事件无法执行

## 4.3 事件绑定函数

|  |
| --- |
| 1. /\*\* 2. \* bindEvent方法 兼容多个浏览器之间事件绑定方式 3. \* @dom 要绑定事件元素 4. \* @type 事件的类型 5. \* @fn 执行事件函数 6. \*\*/ 7. function bindEvent(dom, type, fn) { 8. // 利用能力检测 9. // 能力检测就是当一个对象去读取一个属性的时候，如果读取到就直接输出， 如果没有获取到，会输出undefined，而不是报错的这个特点 10. if (dom.addEventListener) { 11. // DOM2级绑定事件方式 12. dom.addEventListener(type, fn, false); 13. } else if (dom.attachEvent) { 14. // 是IE浏览器 15. dom.attachEvent("on" + type, fn); 16. } else { 17. // 一些不知名的浏览器 18. // 使用dom0级绑定方式 19. dom["on" + type] = fn; 20. } 21. } |

# 事件对象与属性

当事件执行的时候，会产生一系列的信息，浏览器将这些信息封装起来并传递到事件函数中

查看：

|  |
| --- |
|  |

## 5.1 offsetX、 offsetY

这两个属性标记的是鼠标位于元素内部的距离， 会受到子元素的影响

举例：

|  |
| --- |
|  |

会受到子元素的影响:

|  |
| --- |
|  |

## 5.2 clientX、 clientY

这两个属性标记的是鼠标位于视口中的位置

举例：

|  |
| --- |
|  |

## 5.2 pageX、 pageY

这两个属性标记的是鼠标位于页面中的位置

当页面打开的时候，默认处于首屏的位置，鼠标位于视口中的位置和鼠标位于页面中的位置是一致的，当页面出现滚动条的时候， 此时改变滚动条的位置， 这两个值将产生差异

|  |
| --- |
|  |

## 5.3 screenX、 screenY

这两个属性标记的是鼠标位于屏幕中的位置

|  |
| --- |
|  |

## 下午复习：

事件绑定方式与区别：

dom0级绑定方式： dom.onclick = function() {}

只可以给一个元素的一个事件绑定一个函数

在IE中是优先执行

在高级浏览器中按照代码的绑定顺序

thi指向触发事件的对象

dom2级绑定方式：dom.addEventListener(type, fn, bool)

可以给一个元素的一个事件绑定多个函数

是按照代码的绑定顺序执行的

this指向触发事件的对象

attachEvent：dom.attachEvent(type, fn)

可以给一个元素的一个事件绑定多个函数

是逆序执行事件

this指向是window

事件的移除方式：

DOM2级事件移除方式：

dom.removeEventListener(type, fn, bool)

需要注意的是删除的事件函数一定是与绑定的事件函数是同一个

IE中的事件移除方式:

detachEvent(type, fn)

使用方式与removeEventListener()是一致的，只不过没有了第三个参数而已

事件对象与属性：

当一个事件执行的时候，会产生一些列的信息，这些信息会被浏览器收集起来并封装为对象传递到事件函数中

offsetX. offsetY: 标记的是鼠标位于元素内部的位置

clientX clientY: 鼠标位于视口中的位置

pageX pageY: 鼠标位于页面中的位置

screenX screenY: 鼠标位于屏幕中的位置

# IE中的事件对象

经过测试， 在IE中事件对象没有传递到事件函数中

其实在window.event身上

由于在IE中不能够打开列表， 所以我们是用for in进行遍历

兼容方式：

var e = e || window.event  
查看：

|  |
| --- |
|  |

总结：

在DOM0级事件中

在IE中没有将事件对象传递到事件函数中

在高级浏览器中，可以将事件对象传递到事件函数中

this指向触发事件对象

在DOM2级事件绑定方式中:

事件对象可以传递到事件函数中

this指向触发事件的对象

在attachEvent事件中：

事件对象可以传递到事件函数中

this指向window

# 阻止冒泡

举例：

|  |
| --- |
| // 我们希望当点击box2的时候，不要继续往上执行了（不要冒泡）  box2.onclick = function(e) {  // 兼容方式  var e = e || window.event;  console.log("box2");  // 在高级浏览器中阻止冒泡的方式  // console.log(e);  // e.stopPropagation();  // 在IE中阻止冒泡的方式  e.cancelBubble = true;  } |
|  |

**总结：**

**在高级浏览器中阻止冒泡的方式： e.stopPropagation()  
 在IE中阻止冒泡的方式： e.cancelBubble = true**

# 阻止默认事件

当点击某些标签的时候， 会发生一些默认的行为、事件

比如：

submit会默认提交表单

a标签，如果有href属性，默认跳转页面

当页面出现滚动条的时候， 此时滚动鼠标滚轮的时候， 会默认改变滚动条的位置，这些都是默认行为、事件

举例：

|  |
| --- |
| // 获取元素  var a = document.getElementsByTagName("a")[0];  // 给a元素添加点击事件  // a.onclick = function(e) {  // 兼容方式  // var e = e || window.event;  // 我们希望当点击的时候不要跳转页面  // 在高级浏览器中阻止默认事件的方式  // e.preventDefault();  // 在IE中阻止默认事件的方式  // e.returnValue = false;  // 如果是dom0级事件，可以使用return false  // return false;  // }  // DOM2级事件  // 在DOM2级事件中， 不能使用return false 进行阻止默认事件  // a.addEventListener("click", function(e) {  // // return false;  // e.preventDefault();  // })  // attachEvent  a.attachEvent("onclick", function(e) {  return false;  }) |

**总结：**

**在高级浏览器中阻止默认事件的方式： e.preventDefault()**

**在IE中阻止默认事件的方式： e.returnValue = false**

**如果是DOM0级事件绑定方式， 还可以使用return false**

# 快捷尺寸

## 9.1 clientWidth、 clientHeight

样式：

|  |
| --- |
|  |

输出： clientWidth、clientHeight:

|  |
| --- |
| // 添加onload事件  window.onload = function() {  console.log("clinetWidth的值是:", box.clientWidth);  console.log("clientHeight的值是:", box.clientHeight);  } |

结果：

|  |
| --- |
|  |

**总结：**

**clientWidth、clientHeight包含的是: content + padding**

## 9.2 offsetWidth、offsetHeight

样式：

|  |
| --- |
|  |

输出: offsetWidth、 offsetheight:

|  |
| --- |
| 1. // 添加onload事件 2. window.onload = function() { 3. console.log("offsetWidth的值是:", box.offsetWidth); 4. console.log("offsetHeight的值是:", box.offsetHeight); 5. } |

结果：

|  |
| --- |
|  |

## 9.3 clientLeft、 clientTop

|  |
| --- |
|  |

输出clientLeft、 clientTop:

|  |
| --- |
| 1. console.log("clientLeft的值是:", box.clientLeft); 2. console.log("clientTop的值是:", box.clientTop); |

结果：

|  |
| --- |
|  |

# jquery中的快捷尺寸

样式：

|  |
| --- |
|  |

执行代码：

|  |
| --- |
| 1. // content 2. console.log("content的宽是:", $("#box").width()); 3. console.log("content的高是:", $("#box").height()); |

结果：

|  |
| --- |
|  |

content + padding:

|  |
| --- |
| 1. // content + padding   console.log("innerWidth的值是:", $("#box").innerWidth());  console.log("innerHeight的值是:", $("#box").innerHeight()); |

结果：

|  |
| --- |
|  |

content + padding + border:

|  |
| --- |
| 1. // content + padding + border   console.log("outerWidth的值是:", $("#box").outerWidth());  console.log("outerHeight的值是:", $("#box").outerHeight()); |

结果：

|  |
| --- |
|  |

content + padding + border + margin:

|  |
| --- |
| 1. // content + padding + border + margin 2. console.log("outerWidth(true)的值是:", $("#box").outerWidth(true)); 3. console.log("outerHeight(true)的值是:", $("#box").outerHeight(true)); |

结果：

|  |
| --- |
|  |

# 定位父元素与定位值

## 10.1 定位父元素

指向子绝父相中的“父”

它不一定是该元素parentNode

DOM结构：

|  |
| --- |
| <div id="box">  <span>box</span>  <div id="box1">  <span>box1</span>  <!-- 此时， box2是相对于box进行定位，该元素的parentNode是box1-->  <div id="box2">  <span>box2</span>  </div>  </div>  </div> |

样式：

|  |
| --- |
|  |

此时，box2是相对于box进行定位的

如果根据box2得到他的定位父元素呢？

|  |
| --- |
| 1. console.log(box2.offsetParent); |

结果：

|  |
| --- |
|  |

## 10.2 定位值

元素的定位值也是经常要使用的值，所以在DOM中也提供了相应的属性用于快速获取

该属性存在兼容性的问题：

在高级浏览器中， 是从子元素的边框外到父元素的边框内

在IE中，从子元素的边框外到父元素的边框外

查看：

|  |
| --- |
| 1. // 获取box2的定位值 2. console.log(box2.offsetLeft); // 距离自己定位父元素的左边距离 3. console.log(box2.offsetTop); // 距离自己定位父元素的上边的距离 |

在高级浏览器中：

|  |
| --- |
|  |

在IE中：

|  |
| --- |
|  |

## 10.3 jquery中的定位值

样式：

|  |
| --- |
|  |

执行代码：

|  |
| --- |
| 1. // 获取元素的定位值 2. var $box2\_position = $("#box2").position(); 3. console.log($box2\_position); |

结果：

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |