爱创课堂前端培训

javascript第二天

班级：爱创课堂十九期

讲师：彭帅伟

日期：2018年12月14日

目录

[复习： 2](#_Toc24617)

[一、 call和apply 5](#_Toc30646)

[1.1 call 6](#_Toc30072)

[1.2 apply 6](#_Toc167)

[二、 事件流程 7](#_Toc7336)

[三、DOM2级绑定事件方式 8](#_Toc28459)

[四、 IE中的高级绑定方式 10](#_Toc9855)

[五、 DOM2级绑定方式与DOM0级绑定方式之间的区别 11](#_Toc19608)

[5.1 绑定数量 11](#_Toc11264)

[5.2 执行顺序 11](#_Toc14048)

[5.3 是否可以同时存在 11](#_Toc97)

[5.4 this指向 11](#_Toc29998)

[六、 attachEvent绑定方式与dom0级绑定方式之间的区别 12](#_Toc13256)

[6.1 绑定数量 12](#_Toc11815)

[6.2 执行顺序 12](#_Toc13560)

[6.3 是否可以同时存在 12](#_Toc13675)

[6.4 this指向 12](#_Toc20816)

[七、 移除事件 13](#_Toc21016)

[7.1 dom2级绑定事件移除方式 13](#_Toc3526)

[7.2 IE中的事件移除方式 13](#_Toc11862)

[下午复习： 14](#_Toc26828)

[八、事件绑定函数 15](#_Toc24620)

[九、事件对象与属性 16](#_Toc21561)

[9. 1 offsetX offsetY 17](#_Toc15928)

[9.2 clientX clientY 17](#_Toc28596)

[9.3 pageX pageY 18](#_Toc1799)

[9.4 screenX screenY 19](#_Toc3885)

[9.5 IE中的事件对象 19](#_Toc22539)

[十、 阻止冒泡 20](#_Toc13930)

[十一、 阻止默认事件 21](#_Toc6548)

# 复习：

javascript这门语言由DOM、 BOM、ECMAScript组成

DOM：文档对象模型

BOM: 浏览器对象模型

ECMAScript： 核心语法

包含：如何定义变量，运算符， 表达式，循环语句， 流程控制语句，数据类型

dom是由节点组成：

一共分为12种：

我们需要记住的有四种：

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: 该属性标记是的节点的名称

nodeValue: 该属性标记的是节点的值

节点关系：

一共分为3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.previousSibling // 获取前一个兄弟节点

node.nextSibling // 获取下一个兄弟节点

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中创建元素：

利用$()函数的功能创建元素

$(“<div class=’aaa’ data-info=’nihao’>content</div>”

jquery中上树方法有8种：

可以是父元素选择子元素， 可以是子元素选择父元素， 还可以是兄弟选择兄弟

父元素选择子元素：

$(box).append(child) : 在box的后面追加child

$(box).prepend(child): 在box的前面追加child

子元素选择父元素：

$(box).appendTo(father)： 将box追加到father的后面

$(box).prependTo(fahter): 将box追加father的前面

兄弟：

$(erdi).before(dage)

$(erge).after(sandi)

$(erdi).insertBefore(sandi)

$(erdi).insertAfter(dage)

jquery中的节点操作：

wrap: 用于给匹配到的元素外层添加一层元素

wrapAll: 给匹配到的所有元素外层添加一层元素

unwrap: 用于给匹配到的元素的外层去掉一层元素

replaceWith: 用于将匹配到的元素替换掉指定元素

replaceAll: 将指定的元素替换掉所有匹配到的元素

empty: 清空后代， 自己还在

remove: 表示自杀， 自己和后代都没有了

clone(bool): 默认是false, 连同子节点一起复制， 如果是true 连同事件一起复制

动画函数：

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

// 定义计数器

var count = 0;

// 间隔时间

var interval = 20;

// 总次数

var allCount = time / interval;

// 定义对象用于保存元素的初始值

var nowJson = {};

for (var i in json) {

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

}

// 循环完毕之后 初始值也就有了

// 计算步长值

var stepJson = {};

for (var i in json) {

步长值 = （目标值 - 初始值）/ 总次数

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

}

var timer = setInterval(function() {

count++;

for (var i in json) {

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

}

// 判断是否到达目标位置

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);

}

}, interval)

}

# call和apply

在函数定义的时候，里面的this是无法确定的， 只有当函数执行的时候才能确定下来

简单来说： 谁调用，指向谁

举例：

|  |
| --- |
| 1. // 定义函数 2. function fun() { 3. console.log(this, arguments); 4. } 5. // 函数自执行 6. fun(); 7. // 作为事件函数 8. document.onclick = fun; 9. document.body.onclick = fun; |

输出结果：

|  |
| --- |
|  |

## call

call方法是每一个函数天生可以调用的方法

作用：执行函数并且改变函数的this指向

使用方式：

函数.call()

接收多个参数

第一个 参数就是要改变的this指向

从第二个参数开始，都是原函数所需要的参数

举例：

|  |
| --- |
| 1. <body style="height: 100px"> 2. <button id="btn">按钮</button> 3. <script type="text/javascript"> 4. // 获取元素 5. var btn = document.getElementById("btn"); 6. // 定义函数 7. function fun() { 8. console.log(this, arguments); 9. } 10. // 函数自执行 11. // fun(); 12. // 作为事件函数 13. // document.onclick = fun; 14. // document.body.onclick = fun; 15. // 使用call方法改变fun的this指向 16. fun.call(document, 1, 2, 3, 4, 5); 17. fun.call(btn); 18. </script> 19. </body> |

结果：

|  |
| --- |
|  |

## apply

apply方法与call方法类似都是执行函数并改变函数的this指向

apply与call方法的区别是

apply接收两个参数：

第一个参数： 要改变的this指向

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

举例：

|  |
| --- |
| 1. // 使用apply改变函数的this指向 2. fun.apply(document, [1, 2, 3]); 3. fun.apply(btn, [1, 2, 3]); |

结果：

|  |
| --- |
|  |

# 二、 事件流程

整个事件流程分为捕获和冒泡：

捕获： 事件从最顶层元素开始执行，一层一层往下执行，直到最精确的元素

冒泡： 事件从最精确的元素开始执行， 一层一层往上执行， 直到最顶层元素

当处于最精确元素的时候，是不分区捕获和冒泡的

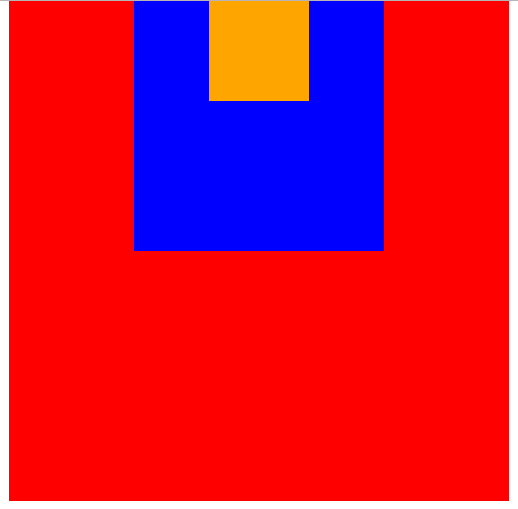
举例：

DOM结构：

|  |
| --- |
| 1. <div id="box"> 2. <div id="box1"> 3. <div id="box2"></div> 4. </div> 5. </div> |

样式：

|  |
| --- |
| 1. #box { 2. width: 500px; 3. height: 500px; 4. background-color: red; 5. margin: 0 auto; 6. } 7. #box1 { 8. width: 250px; 9. height: 250px; 10. margin: 0 auto; 11. background-color: blue; 12. } 13. #box2 { 14. width: 100px; 15. height: 100px; 16. margin: 0 auto; 17. background-color: orange; 18. } |



执行代码：

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

结果：

|  |
| --- |
|  |

# 三、DOM2级绑定事件方式

事件名称： addEventListener

这个是一个方法，每一个元素都可以调用

使用方式：

dom.addEventListener(type, fn, bool)

type: 事件类型 注意： 不带“on” 比如: click, mousedown, mouseup

fn: 事件函数

bool: 是一个布尔值，该参数决定了绑定到捕获还是冒泡阶段， 默认是false， 表示绑定到冒泡阶段，如果是true， 表示绑定到捕获阶段

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. // 绑定到冒泡阶段 6. box.addEventListener("click", function() { 7. console.log("box 冒泡"); 8. }, false) 9. box1.addEventListener("click", function() { 10. console.log("box1 冒泡"); 11. }, false) 12. box2.addEventListener("click", function() { 13. console.log("box2 冒泡"); 14. }, false) 15. // 绑定到捕获阶段 16. box.addEventListener("click", function() { 17. console.log("box 捕获"); 18. }, true) 19. box1.addEventListener("click", function() { 20. console.log("box1 捕获"); 21. }, true) 22. box2.addEventListener("click", function() { 23. console.log("box2 捕获"); 24. }, true) |

结果：

|  |
| --- |
|  |

# IE中的高级绑定方式

在IE中不支持addEventListener事件绑定方式， 支持自己的attachEvent绑定事件方式

使用方式：

dom.attachEvent(type, fn)

type: 字符串， 表示事件类型， 注意： 带“on”的， 比如： onclick, onmousedown , onmouseup

fn: 事件函数

没有第三个参数，也就是说不支持捕获阶段

举例：

|  |
| --- |
| 1. box.attachEvent("onclick", function() { 2. console.log("attachEvent"); 3. }) 4. box1.attachEvent("onclick", function() { 5. console.log("attachEvent1") 6. }) 7. box2.attachEvent("onclick", function() { 8. console.log("attachEvent2") 9. }) |

结果：

|  |
| --- |
|  |

# DOM2级绑定方式与DOM0级绑定方式之间的区别

## 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

dom2级: 可以给一个元素的一个事件绑定多个函数

## 执行顺序

按照代码的绑定顺序

## 是否可以同时存在

可以

## this指向

都指向触发事件的对象

举例：

|  |
| --- |
| 1. // 获取元素 2. var box = document.getElementById("box"); 3. // 添加dom0级事件 4. // box.onclick = function() { 5. // console.log(2); 6. // } 7. // 如果两条输出语句都可以执行， 说明可以绑定多个事件函数， 如果只输出一个，说明只能绑定一个事件函数 8. // 经过测试，只能输出一条语句， 并且是后面的会覆盖前面的 9. // dom2级 10. box.addEventListener("click", function() { 11. console.log("addEventListener"); 12. console.log(this); 13. }) 14. box.onclick = function() { 15. console.log(1); 16. console.log(this); 17. } 18. // box.addEventListener("click", function() { 19. // console.log("addEventListener1"); 20. // }) 21. // 总结: dom0级和dom2绑定的事件可以同时存在 22. // 事件执行的顺序是: 代码的绑定顺序 23. // this指向触发事件的对象 |

# attachEvent绑定方式与dom0级绑定方式之间的区别

## 6.1 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: 可以给一个元素绑定多个事件函数

## 6.2 执行顺序

优先执行dom0级事件，逆序执行attachEvent事件

## 6.3 是否可以同时存在

可以

## 6.4 this指向

dom0级指向触发事件的对象

attachEvent事件中的this指向window

举例:

|  |
| --- |
| 1. // 获取元素 2. var box = document.getElementById("box"); 3. // 添加dom0级事件 4. // attachEvent 5. // box.attachEvent("onclick", function() { 6. // console.log("attachEvent2"); 7. // }) 8. box.onclick = function() { 9. console.log(1); 10. console.log(this); 11. } 12. box.attachEvent("onclick", function() { 13. console.log("attachEvent1"); 14. console.log(this); 15. }) 16. // 总结: 17. // attachEvent可以绑定多个事件函数 18. // 执行顺序: dom0级优先执行， 逆序执行attachEvent事件 19. // this指向: attachEvent事件指向window |

# 移除事件

## 7.1 dom2级绑定事件移除方式

事件名称：removeEventListener

作用：该方法用于移除addEventListent绑定的事件函数

使用方式：

dom.removeEventListener(type, fn, bool)

type: 表示事件类型

fn: 事件函数

bool: 是一个布尔值， 决定要移除哪个阶段的事件函数

错误的移除事件方式：

|  |
| --- |
| 1. box.addEventListener("click", function() { 2. console.log("事件执行了"); 3. }) 4. // 当点击btn的时候移除box的事件 5. btn.onclick = function() { 6. box.removeEventListener("click", function() { 7. console.log("事件执行了"); 8. }) 9. } |

正确的移除方式：

|  |
| --- |
| 1. // 将函数提取到外部，并且声明 2. function click() { 3. console.log("事件执行了"); 4. } 5. box.addEventListener("click", click); 6. // 当点击btn的时候移除box的事件 7. btn.onclick = function() { 8. box.removeEventListener("click", click); 9. } |

## 7.2 IE中的事件移除方式

事件名称：detachEvent

作用：用于移除attachEvent绑定的事件

使用方式与removeEventListener是一致的，只是没有第三个参数而已

举例：

|  |
| --- |
| 1. // 将函数提取到外部，并且声明 2. function click() { 3. console.log("事件执行了"); 4. } 5. box.attachEvent("onclick", click); 6. // 当点击btn的时候移除box的事件 7. btn.onclick = function() { 8. box.detachEvent("onclick", click); 9. } |

# 下午复习：

call与apply:

作用：执行函数并改变函数的this指向

call:

接收多个参数

第一个参数： 就是要改变的this指向

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this指向

第二个参数是： 数组， 数组中的每一项是原函数所需要的参数

事件流程：

整个事件流程分为捕获和冒泡：

捕获：事件从最顶层元素开始执行，一层一层往下， 直到最精确的元素

冒泡： 事件从最精确的元素开始执行， 一层一层往上， 直到最顶层元素

绑定事件的方式与区别：

dom绑定方式： dom.onclick = function() {}

只绑定到冒泡阶段

只能够给一个元素的一个事件绑定一个函数

执行事件的顺序：

在IE中， 优先执行

在高级浏览器中， 按照代码的绑定顺序执行

this指向触发事件的元素

dom2级绑定方式：

dom.addEventListener(type, fn, bool)

可以绑定多个事件函数

执行事件的顺序：

代码的绑定顺序

可以与dom0级事件同时存在

this指向触发事件的对象

IE中的高级绑定方式：

dom.attachEvent(type, fn)

可以绑定多个事件函数

执行事件的顺序：

先绑定的后执行， 后绑定的先执行

dom0级优先执行

this指向window

移除事件：

dom2级的移除事件方式：

dom.removeEventListener(type, fn, bool)

需要注意的是匿名函数无法被移除

IE中的移除方式：

dom.detachEvent(type, fn)

使用方式与removeEventListener是一致的， 只是，没有了第三个参数而已

# 八、事件绑定函数

|  |
| --- |
| 1. /\*\* 2. \* bindEvent 实现多个浏览器的事件绑定 3. \* @dom 要绑定事件的元素 4. \* @type 事件类型 5. \* @fn 事件执行函数 6. \*\*/ 7. function bindEvent(dom, type, fn) { 8. // 使用能力检测，判断当前浏览器支持哪种能力 9. // 能力检测: 就是利用，当一个对象读取一个属性的时候， 如果能够读取到，就输出， 如果读取不到， 会输出undefined, 而不是报错这个特点 10. if (dom.addEventListener) { 11. // console.log(123); 12. // 说明是高级浏览器 13. dom.addEventListener(type, fn, false); 14. } else if (dom.attachEvent) { 15. // 说明是IE中的高版本 16. dom.attachEvent("on" + type, fn); 17. } else { 18. // IE中的低版本或者是一些不知名的浏览器 19. dom["on" + type] = fn; 20. } 21. } |

# 九、事件对象与属性

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

举例：

|  |
| --- |
| 1. // 添加点击事件 2. box.onclick = function(e) { 3. // 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成一个对象传递到事件函数中 4. console.log(e); 5. } |

查看：

|  |
| --- |
|  |

## 1 offsetX offsetY

这两个属性标记的是鼠标位于元素内部的位置， 会受到子元素的影响

举例：

|  |
| --- |
|  |

会受到子元素的影响：

|  |
| --- |
|  |

## 9.2 clientX clientY

这两个属性标记的是鼠标位于视口的位置

|  |
| --- |
|  |

整个页面：

|  |
| --- |
|  |

视口：

|  |
| --- |
|  |

## 9.3 pageX pageY

这两个属性标记的是鼠标位于页面位置

当页面打开的时候，默认是处于首屏，此时鼠标位于视口中的位置与位于页面中的位置是一致的，当页面出现滚动条并且改变滚动条位置的时候，这两个值将产生差异

举例：

|  |
| --- |
|  |

## 9.4 screenX screenY

这两个属性标记是鼠标位于屏幕的距离

|  |
| --- |
|  |

## 9.5 IE中的事件对象

经过测试，在dom0级事件中，IE中没有将事件对象传递到事件函数中

其实，在window.event身上

由于在IE中不能点开，所以我们使用for in 进行查看

兼容方式：

var e = e || window.event;

查看：

|  |
| --- |
|  |

总结：

dom0级：

在高级浏览器中， 可以将事件对象到事件函数中

在IE中，没有将事件对象传递进来，在window.event身上

this指向触发事件的对象

dom2级：

可以将事件对象传递到事件函数中

this指向触发事件的对象

attachEvent：

可以将事件对象传递进来

this指向window

# 阻止冒泡

在高级浏览器中：

|  |
| --- |
| 1. // 在高级浏览器中阻止冒泡的方式 2. e.stopPropagation(); |

在IE中阻止冒泡的方式：

|  |
| --- |
| 1. e.cancelBubble = true; |

# 阻止默认事件

有些标签当点击的时候或者是进行一些其他操作的时候， 会有默认的行为、事件

比如：

submit, 会默认提交表单

a标签，如果有herf属性，会默认跳转页面

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

举例：

|  |
| --- |
| 1. // 添加点击事件 2. a.onclick = function(e) { 3. // 兼容IE 4. var e = e || window.event; 5. // 在高级浏览器中阻止默认事件的方式 6. // e.preventDefault(); 7. // 在IE阻止默认事件的方式 8. // e.returnValue = false; 9. // 如果是dom0级事件， 还可以使用 return false; 10. return false; 11. } 12. // dom2级中 return false 不能阻止默认事件的 13. a.addEventListener("click", function(e) { 14. return false; 15. }) 16. // 在attachEvent事件中， 可以使用return false 17. a.attachEvent("onclick", function(e) { 18. return false; 19. }) |

如果是dom0级事件， 还可以使用 return false;

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |