爱创课堂前端培训

# javascript进阶

第1天课堂笔记（本课程共5天）

班级：北京前端训练营27期

讲师：蔡久明

日期：

爱创课堂官网 ：[www.icketang.com](http://www.icketang.com)

目录

**[复习](#_Toc12778_WPSOffice_Level1)** **[3](#_Toc12778_WPSOffice_Level1)**

**[一、javascript](#_Toc17_WPSOffice_Level1)** **[3](#_Toc17_WPSOffice_Level1)**

**[二、DOM](#_Toc2837_WPSOffice_Level1)** **[4](#_Toc2837_WPSOffice_Level1)**

[2.1 dom的组成](#_Toc11567_WPSOffice_Level2) [4](#_Toc11567_WPSOffice_Level2)

[2.2 节点的属性](#_Toc8860_WPSOffice_Level2) [4](#_Toc8860_WPSOffice_Level2)

[2.3 childNodes 子节点](#_Toc32665_WPSOffice_Level2) [5](#_Toc32665_WPSOffice_Level2)

[2.4 childNodes属性的兼容处理](#_Toc32645_WPSOffice_Level2) [5](#_Toc32645_WPSOffice_Level2)

[复习](#_Toc3846_WPSOffice_Level2) [6](#_Toc3846_WPSOffice_Level2)

**[三、 节点操作（增删改查）](#_Toc30441_WPSOffice_Level1)** **[7](#_Toc30441_WPSOffice_Level1)**

[3.1 节点查找（节点的关系）](#_Toc28540_WPSOffice_Level2) [7](#_Toc28540_WPSOffice_Level2)

[3.2 节点删除（下树）](#_Toc23227_WPSOffice_Level2) [8](#_Toc23227_WPSOffice_Level2)

[3.3 节点的替换](#_Toc10504_WPSOffice_Level2) [9](#_Toc10504_WPSOffice_Level2)

[3.4 创建元素节点](#_Toc8498_WPSOffice_Level2) [9](#_Toc8498_WPSOffice_Level2)

[3.4 添加节点（上树）](#_Toc5624_WPSOffice_Level2) [10](#_Toc5624_WPSOffice_Level2)

[3.5 cloneNode](#_Toc19595_WPSOffice_Level2) [10](#_Toc19595_WPSOffice_Level2)

**[四、 jquery的节点操作](#_Toc9232_WPSOffice_Level1)** **[11](#_Toc9232_WPSOffice_Level1)**

[4.1 内部插入](#_Toc15861_WPSOffice_Level2) [11](#_Toc15861_WPSOffice_Level2)

[4.2 外部插入](#_Toc6236_WPSOffice_Level2) [11](#_Toc6236_WPSOffice_Level2)

[4.3 元素的包裹](#_Toc13464_WPSOffice_Level2) [12](#_Toc13464_WPSOffice_Level2)

[4.4 元素替换](#_Toc12930_WPSOffice_Level2) [13](#_Toc12930_WPSOffice_Level2)

[4.5 元素的删除](#_Toc536_WPSOffice_Level2) [13](#_Toc536_WPSOffice_Level2)

[4.6 元素克隆](#_Toc13257_WPSOffice_Level2) [13](#_Toc13257_WPSOffice_Level2)

# 复习

1. 函数节流

控制函数执行的频率

1. 声明锁
2. 判断锁
3. 更改锁
4. 重置锁
5. 延时器

setTimeout(fn, time) fn:要执行的回调函数， time延迟执行时间

1. this

this代表触发该事件的对象，如果在元素遍历的时候，表示遍历到的当前元素。无主函数this指向window

1. call

call可以改变this的指向。 fn.call(target) fn:要调用的函数名 target：表示this指向的元素

1. each

遍历对象及合：

$(dom).each(function(i) {

i:遍历当前对象的索引

this：遍历当前的对象

})

遍历数组和对象：

$.each(target, function(i) {

i:遍历当前元素的索引

this：遍历当前元素的值

})

# 一、javascript

javascript这门语言是由DOM、BOM、ECMAScript组成

其中DOM指的是: document object model 文档对象模型 体现在代码中是document

BOM指的是: browser object model 浏览器对象模型 体现在代码中 window

ECMAScript: 核心语法

里面包含的是如何定义变量、运算符、表达式、流程控制语句、数据类型

在js基础部分我们学习的是核心语法，在进阶部分我们学习另外两个

# 二、DOM

我们可以通过dom的onclick、onmousedown、onmouseup来添加事件

还可以通过dom的getElementById获取元素

还可以通过dom的style来设置样式

……

## 2.1 dom的组成

dom是由节点组成的，元素/标签 它们是节点中的某一种

节点一共分为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

全部节点类型：

|  |
| --- |
| 1. 元素节点 　　Node.ELEMENT\_NODE(1) 2. 属性节点 　　Node.ATTRIBUTE\_NODE(2) 3. 文本节点 　　Node.TEXT\_NODE(3) 4. CDATA节点 Node.CDATA\_SECTION\_NODE(4) 5. 实体引用名称节点 　　Node.ENTRY\_REFERENCE\_NODE(5) 6. 实体名称节点 　　Node.ENTITY\_NODE(6) 7. 处理指令节点 　　Node.PROCESSING\_INSTRUCTION\_NODE(7) 8. 注释节点 　 Node.COMMENT\_NODE(8) 9. 文档节点 　 Node.DOCUMENT\_NODE(9) 10. 文档类型节点 　　Node.DOCUMENT\_TYPE\_NODE(10) 11. 文档片段节点 　　Node.DOCUMENT\_FRAGMENT\_NODE(11) 12. DTD声明节点 Node.NOTATION\_NODE(12) |

## 2.2 节点的属性

nodeType:节点的属性

nodeName:节点的名称

nodeValue:节点的值

执行代码：

|  |
| --- |
| 1. // 获得元素对象 2. var box = document.getElementById('box'); 3. // 获得box的节点属性 4. console.log('节点属性:', box.nodeType, '节点名称：', box.nodeName, '节点值:', box.nodeValue); 5. console.log('节点属性:', document.nodeType, '节点名称：', document.nodeName, '节点值:', document.nodeValue); |

结果：

|  |
| --- |
|  |

## 2.3 childNodes 子节点

获得所有的子节点

结构：

|  |
| --- |
| 1. <div id="box"><!--我是一个伟大的注释-->i love you<b></b></div> |

执行代码:

|  |
| --- |
| 1. // 获得元素对象 2. var box = document.getElementById('box'); 3. // 获取所有的子节点 4. console.log(box.childNodes); 5. // 查看子节点的具体属性 6. for (var i = 0; i < box.childNodes.length; i++) { 7. console.log('节点属性:', box.childNodes[i].nodeType, '节点名称：', box.childNodes[i].nodeName, '节点值:', box.childNodes[i].nodeValue); 8. } |

结果：

|  |
| --- |
|  |

|  |
| --- |
|  |

## 2.4 childNodes属性的兼容处理

IE浏览器会忽略空白文本节点，而高级浏览器则会查看空白节点。因为空白节点没有意义，所以做一个兼容处理都忽略空白节点

|  |
| --- |
|  |

执行代码：

|  |
| --- |
| 1. // 获得元素对象 2. var box = document.getElementById('box'); 3. // 兼容处理 4. function getNodes(dom) { 5. // 声明一个容器，用来返回处理后的子节点 6. var nodes = []; 7. // 遍历所有的子节点，将非空白节点放入到nodes数组中 8. for (var i = 0; i < dom.childNodes.length; i++) { 9. // 判断是否为空白节点 10. // 正则表达式验证空白 11. var reg = /^\s+$/; 12. if (dom.childNodes[i].nodeType === 3 && reg.test(dom.childNodes[i].nodeValue)) { 13. // 是空白节点 14. continue;// 略过本次循环 15. } 16. // 非空白节点 17. nodes.push(dom.childNodes[i]); 18. } 19. // 返回所有的子节点 20. return nodes; 21. } 22. // 获取子节点 23. console.log(getNodes(box)); |

结果：

|  |
| --- |
|  |

## 复习

dom由节点组成，节点分为12种。

四种

元素节点 1

文本节点 3

注释节点 8

文档节点 9

节点属性

类型：nodeType

名称：nodeName

值： nodeValue

属性：

childNodes 获取子节点

IE和高级浏览器的区别：IE会忽略空白节点

兼容性处理

# 节点操作（增删改查）

## 3.1 节点查找（节点的关系）

父子关系

获取所有的子节点：childNodes

获取第一个子节点：firstChild

获取最后一个子节点：lastChild

获取所有的子元素节点：children

结构：

|  |
| --- |
| 1. <ul> 2. <li>1</li> 3. <li>2</li> 4. <li>3</li> 5. <li>4</li></ul> |

执行代码：

|  |
| --- |
| 1. // 获取元素 2. var ul = document.getElementsByTagName('ul')[0]; 3. // 获得ul下的所有子节点 4. console.log(ul.childNodes); 5. // 获取ul下的第一个子节点 6. console.log(ul.firstChild); 7. // 获取ul的最后一个子节点 8. console.log(ul.lastChild); 9. // 获取ul下的所有子元素节点 10. console.log(ul.children); 11. // 获取第一个子元素节点 12. console.log(ul.firstElementChild); 13. // 获取最后一个子元素节点 14. console.log(ul.lastElementChild); |

结果：

|  |
| --- |
|  |

子父关系：

查找父节点：parentNode

执行代码：

|  |
| --- |
| 1. // 获得元素<li>3</li>的父节点 2. var li = document.getElementsByTagName('li')[2]; 3. console.log(li.parentNode); |

结果：

|  |
| --- |
|  |

兄弟关系：

上一个兄弟节点：previousSibling

下一个兄弟节点：nextSibling

执行代码：

|  |
| --- |
| 1. // 获得上一个兄弟节点 2. console.log(li.previousSibling.previousSibling); 3. // 获得下一个兄弟节点 4. console.log(li.nextSibling.nextSibling); 5. // 获得上一个兄弟元素节点 6. console.log(li.previousElementSibling); 7. // 获得下一个兄弟元素节点 8. console.log(li.nextElementSibling); |

## 3.2 节点删除（下树）

删除节点：parent.removeChild（child）

结构：

|  |
| --- |
| 1. <ul> 2. <li>1</li> 3. <li>2</li> 4. <li>3</li> 5. <li>4</li></ul> |

执行代码：

|  |
| --- |
| 1. // 获取元素 2. var ul = document.getElementsByTagName('ul')[0]; 3. var li = document.getElementsByTagName('li')[2]; 4. // 删除节点 5. ul.removeChild(li); |

结果：

|  |
| --- |
|  |

## 3.3 节点的替换

parent.replaceChild(new, old)

parent:父节点

new：要替成的节点

old：要被替换的节点

old.replaceWith(new)

new：要替成的节点

old：要被替换的节点

## 3.4 创建元素节点

document.createElement(‘标签名’)

执行代码：

|  |
| --- |
| 1. // 获取元素 2. var ul = document.getElementsByTagName('ul')[0]; 3. var li = document.getElementsByTagName('li')[2]; 4. // 创建一个元素节点 5. var div = document.createElement('div'); 6. // 节点替换 7. // li.replaceWith(div); 8. ul.replaceChild(div, li); |

结果：

|  |
| --- |
|  |

## 3.4 添加节点（上树）

添加到末尾： parent.appendChild(child) 将子节点添加到父节点的末尾

添加到指定的位置： parent.insetrtBefore(newChild, oldChild) 将新的子节点添加到指定的父节点中的oldChild节点的前面

执行代码：

|  |
| --- |
| 1. // 获取元素 2. var ul = document.getElementsByTagName('ul')[0]; 3. var li = document.getElementsByTagName('li')[2]; 4. // 创建一个元素节点 5. var div = document.createElement('div'); 6. div.innerHTML = '久哥最帅'; 7. // 添加节点 8. ul.appendChild(div); 9. // 将div添加到li的前面 10. ul.insertBefore(div, li); |

结果：

|  |
| --- |
|  |

## 3.5 cloneNode

克隆节点

cloneNode(bool)

bool:表示是否克隆子节点， 默认值为false

|  |
| --- |
| 1. var div = document.createElement('div'); 2. div.innerHTML = '久哥最帅'; 3. var div1 = div.cloneNode(); 4. // div1.innerHTML = '久哥真英俊'; 5. console.log(div1); 6. var div2 = div.cloneNode(true); 7. console.log(div2); |

结果：

|  |
| --- |
|  |

注意：原生js，cloneNode不会克隆事件，但是jquery的克隆可以克隆事件的

# jquery的节点操作

## 4.1 内部插入

parent.append(child) 在父元素末尾插入子元素

child.appendTo(parent) 将子元素添加到父元素的末尾

parent.prepend(child) 在父元素头部插入子元素

child.prependTo(parent) 将子元素添加到父元素的头部

执行代码：

|  |
| --- |
| 1. // append：在父元素末尾插入子元素 2. $('ul').append($('<div>张弘真漂亮</div>')); 3. // appendTo； 将子元素插入到父元素的末尾 4. $('<li>张弘真靓</li>').appendTo($('ul')); 5. // prepend：在父元素的头部插入子元素 6. $('ul').prepend($('<b>靓仔张弘</b>')); 7. // prependTo:将子元素插入到父元素的头部 8. $('<i>靓仔弘</i>').prependTo($('ul')); |

结果：

|  |
| --- |
|  |

## 4.2 外部插入

A.after(B):在A元素后面添加兄弟元素B

A.insertAfter(B):将A元素添加到B元素的后面

A.before(B):在A元素的前面添加B元素

A.innsertBefore(B):将A添加到B元素的前面

执行代码：

|  |
| --- |
| 1. // 获得元素 2. var $li = $('#li'); 3. // A.after(B):在A元素后面添加兄弟元素B 4. $li.after($('<li>张弘</li>')); 5. // A.insertAfter(B):将A元素添加到B元素的后面 6. $('<li>丁秋文</li>').insertAfter($li); 7. // A.before(B):在A元素的前面添加B元素 8. $li.before($('<li>BOSS蔡久明</li>')); 9. // A.innsertBefore(B):将A添加到B元素的前面 10. $('<li>老鸨：白杰</li>').insertBefore($li); |

结果：

|  |
| --- |
|  |

## 4.3 元素的包裹

wrap元素包裹

unwrap 解除元素的包裹

wrapAll 将所有匹配的元素使用一个元素进行包裹

wrapInner 将匹配的元素和子元素之间使用一个元素进行包裹

执行代码：

|  |
| --- |
| 1. // wrap元素包裹 2. $('li').wrap($('<ol></ol>')); 3. // unwrap 解除元素的包裹 4. $('li').unwrap($('<ol></ol>')); 5. // wrapAll 将所有匹配的元素使用一个元素进行包裹 6. $('li').wrapAll($('<ol></ol>')); 7. // wrapInner 将匹配的元素和子元素之间使用一个元素进行包裹 8. $('li').wrapInner($('<b></b>')); |

结果：

|  |
| --- |
|  |

## 4.4 元素替换

A.replaceWith(B) 使用B替换A

A.replaceAll(B) 使用A替换B

执行代码：

|  |
| --- |
| 1. // 元素替换 replaceWith 2. $('li').replaceWith($('<b>久哥最帅</b>')); 3. // 元素替换 replaceAll 4. $('<h1>张弘真漂亮</h1>').replaceAll($('b')); |

结果：

|  |
| --- |
|  |

## 4.5 元素的删除

remove:连自己都删除

empty：删除自己的内部，不会删除自己

结构：

|  |
| --- |
| 1. <li>香蕉</li> 2. <p>榴莲</p> |

执行代码：

|  |
| --- |
| 1. // empty:移除元素内的所有节点，不会删除元素节点本身 2. $('li').empty(); 3. // remove:移除元素内部的所有子节点，包括自己本身 4. $('p').remove(); |

结果：

|  |
| --- |
|  |

## 4.6 元素克隆

clone(bool) 默认克隆所有的子元素

bool:控制是否克隆该元素的事件， 默认值为false

执行代码：

|  |
| --- |
| 1. // 绑定事件 2. $('#btn').click(function() { 3. console.log('人不狠站不稳'); 4. }); 5. // 克隆btn 6. $('#btn').clone().css('color', 'red').appendTo($(document.body)); 7. $('#btn').clone(true).css('color', 'blue').appendTo($(document.body)); |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |