目录

[javascript进阶 1](#_Toc76)

[一、 javascript 3](#_Toc24447)

[二、 DOM 3](#_Toc28386)

[2.1 dom的组成 3](#_Toc26099)

[2.2 节点类型 4](#_Toc19591)

[2.2 childNodes属性 5](#_Toc17187)

[2.3 封装函数，解决childNodes兼容性问题 5](#_Toc30517)

[2.3 节点属性 6](#_Toc3712)

[2.4 节点关系 7](#_Toc31883)

[三、 元素操作 8](#_Toc27473)

[3.1 创建元素 8](#_Toc13880)

[3.2 上树 8](#_Toc2569)

[3.3 下树 9](#_Toc17013)

[下午复习： 11](#_Toc18757)

[3.4 插入节点 13](#_Toc398)

[3.5 替换 13](#_Toc7097)

[3.6 克隆 14](#_Toc18580)

[四、 jquery中的节点操作 15](#_Toc17054)

[4.1 创建元素 15](#_Toc15013)

[4.2 上树 15](#_Toc10323)

[4.3 wrap 17](#_Toc1391)

[4.4 wrapAll 18](#_Toc17750)

[4.5 unwrap 18](#_Toc25309)

[4.6 replaceWith、 replaceAll 18](#_Toc2501)

[4.7 empty 19](#_Toc4361)

[4.8 remove 19](#_Toc22519)

[4.9 clone 20](#_Toc15084)

[五、 call与apply方法 20](#_Toc16639)

[5.1 call 21](#_Toc24419)

[5.2 apply 21](#_Toc31244)

[六、 事件流程 22](#_Toc29057)

[6.1 DOM2级绑定方式 23](#_Toc31237)

# javascript

javascript这门语言是由DOM、BOM、ECMAScript组成

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

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

ECMAScript： 核心语法

包含的有如何定义变量、运算符、表达式、流程控制语句、数据类型

js基础部分我们学习的是核心语法， js进阶部分我们学习剩下的两个

# DOM

我们可以通过dom的getElement来获取元素

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

还可以通过dom的style来设置样式

等等……

## 2.1 dom的组成

dom是由节点组成， 元素/标签，它们只是节点的一种

节点一共分为12种类型：

其实有四种我们需要记住：

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

## 2.2 节点类型

每一个元素都是一个节点，但是每一个节点不一定是一个元素，元素是节点的一种分类

DOM结构：

|  |
| --- |
| 1. <div id="box">你好，世界<div class="one"></div><!-- 我是一个注释 --></div> |

元素.nodeType:

|  |
| --- |
| 1. // 查看元素的节点类型 2. console.log(box.nodeType); // 这里输出1 是元素类型 |

文本.nodeType:

|  |
| --- |
|  |

注释.nodeType:

|  |
| --- |
|  |

文档.nodeType:

|  |
| --- |
|  |

## 2.2 childNodes属性

该属性指向的是一个节点的所有子节点的集合

DOM结构：

|  |
| --- |
| 1. <div class="box" id="box"> 2. 你好，世界 3. <div class="one"></div> 4. <!-- 我是注释 --> 5. </div> |

执行代码：

|  |
| --- |
| 1. // 获取元素 2. var box = document.getElementById("box"); 3. // 查看box的所有子节点 4. var arr = box.childNodes; 5. console.log(arr); |

结果：

|  |
| --- |
|  |

现在输出5个节点，而我们明明只书写了3个节点

这是因为在高级浏览器中空白折叠现象形成的空白符也会当做文本节点存在

再到IE中去观察：

由于在IE中无法打开列表，我们输出arr.length

结果：

|  |
| --- |
|  |

此时， IE反而比高级浏览器还好用

## 2.3 封装函数，解决childNodes兼容性问题

|  |
| --- |
| 1. function getChildNodes(dom) { 2. /\*\* 3. \* 步骤: 4. \* 1: 循环查看dom的所有子节点， 查看有没有文本类型 5. \* 2: 如果有文本类型， 定义正则表达式去匹配 6. \* 3: 定义一个数组，存储不是空白符的节点 7. \* 4: 将不是空白符的节点存储起来 8. \* 5: 返回该数组 9. \*\*/ 10. // 定义数组 11. var arr = []; 12. for (var i = 0; i < dom.childNodes.length; i++) { 13. // 查看子节点中有没有文本类型 14. // console.log(dom.childNodes[i].nodeType); 15. // 判断是否是文本节点 16. if (dom.childNodes[i].nodeType == 3) { 17. // 是文本类型 18. // 定义正则表达式去匹配 19. var reg = /^\s+$/; 20. // 验证 21. if (reg.test(dom.childNodes[i].data)) { 22. // 说明是纯空白 23. console.log("第" + i + "项是纯空白"); 24. } else { 25. // 说明不是空白符, 进入到数组中 26. arr.push(dom.childNodes[i]); 27. } 28. } else { 29. // 不是文本类型 可以直接进入数组中 30. arr.push(dom.childNodes[i]); 31. } 32. } 33. // 循环完毕之后，返回数组 34. return arr; 35. } |

## 2.3 节点属性

nodeType: 标记的是节点的类型

nodeName: 标记的是节点的名称

nodeValue: 标记的是节点的值

DOM结构：

|  |
| --- |
| 1. <div id="box"> 2. <div class="one"></div> 3. 你好, 明天 4. <!-- 我是一个注释 --> 5. </div> |

执行代码：

|  |
| --- |
| 1. // 获取元素 2. var box = document.getElementById("box"); 3. // 获取box的所有子节点 4. var box\_children = getChildNodes(box); 5. // console.log(box\_children); 6. for (var i = 0; i < box\_children.length; i++) { 7. // nodeType属性标记的是该节点的类型 8. console.log(box\_children[i].nodeType); 9. // nodeName属性标记的是该节点的名称 10. console.log(box\_children[i].nodeName); 11. // nodeValue属性标记的是该节点的值 12. console.log(box\_children[i].nodeValue); 13. } 14. // 输出文档类型 文档类型的值 15. console.log(document, document.nodeValue); |

结果：

|  |
| --- |
|  |

## 2.4 节点关系

在节点关系中可以分为三种：

父子：

|  |
| --- |
| 1. father.childNodes // 获取所有的子节点 2. father.firstChild // 获取第一个子节点 3. father.lastChild // 获取最后一个子节点 |

子父：

|  |
| --- |
| 1. child.parentNode // 获取父节点 |

兄弟：

|  |
| --- |
| 1. node.nextSibling // 下一个兄弟节点 2. node.previousSibling // 上一个兄弟节点 |

举例：

|  |
| --- |
| 1. var box = document.getElementById("box"); 2. var ul = box.childNodes[0]; 3. // 获取第一个子节点 4. var first = ul.firstChild; 5. // console.log(first); 6. // 获取最后一个子节点 7. var last = ul.lastChild; 8. // console.log(last); 9. // 中间的这些元素没有办法直接获取， 但是可以通过childNodes[1 | 2 | 3]来获取 10. // 父节点 11. var father1 = first.parentNode; 12. var father2 = last.parentNode; 13. // console.log(father1 === father2); 14. // 兄弟之间的关系 15. var li3 = ul.childNodes[2]; 16. // 下一个兄弟节点 17. var li4 = li3.nextSibling; 18. console.log(li4); |

# 元素操作

## 创建元素

使用方式:

document.createElement(type)

type: 字符串， 元素的类型

返回值: 创建出来的元素

举例：

|  |
| --- |
| 1. var p = document.createElement("p"); 2. console.log(p); |

结果：

|  |
| --- |
|  |

## 上树

使用方式：

father.appendChild(child)

child：要追加的子节点

father: 父节点

返回值: child

最终效果: child作为father中最后一个子节点存在

举例：

|  |
| --- |
| 1. // 获取元素 2. var box = document.getElementById("box"); 3. // 创建元素 可以利用DOM中提供的相应的方法 4. var p = document.createElement("p"); 5. console.log(p); 6. // 将创建出来的p元素追加到box中 7. box.appendChild(p); |

结果：

|  |
| --- |
|  |

## 下树

使用方式：

father.removeChild(child)

father: 父节点

child: 要移除的子节点

返回值: child

最终效果：father将child从所有的子节点中移除

举例：

|  |
| --- |
| 1. // 获取元素 2. var box = document.getElementById("box"); 3. var btn1 = document.getElementById("btn1"); 4. var btn2 = document.getElementById("btn2"); 5. // 创建一个p元素 6. var p = document.createElement("p"); 7. // console.log(p); 8. // 给btn1注册点击事件 9. btn1.onclick = function() { 10. // 追加子元素 11. box.appendChild(p); 12. } 13. // 当点击btn2的时候， 移除子元素 14. btn2.onclick = function() { 15. box.removeChild(p); 16. } |

当点击btn1的时候：

|  |
| --- |
|  |

当点击btn2的时候：

|  |
| --- |
|  |

## 下午复习：

javascript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

chidNodes属性：

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

节点属性：

nodeType:该属性标记的是节点的类型

nodeNameL: 该属性标记的是节点的名称

nodeValue: 该属性标记的是节点的值

节点关系：

父子：

father.childNode 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: 下一个兄弟节点

node.previousSibling: 上一个兄弟节点

元素操作：

创建元素：

document.createElement(type)

type: 元素的类型 字符串

上树：

father.appendChild(child)

child: 要追加的元素

下树：

father.removeChild(child)

child: 要移除的元素

## 插入节点

使用方式：

father.insertBefore(newChild, oldChild)

newChild: 要插入的元素

oldChild: 参照元素

返回值： newChild

最终效果：newChild作为oldChild的兄弟节点存在

举例:

|  |
| --- |
| 1. // 获取元素 2. var ul = document.getElementsByTagName("ul")[0]; 3. // 获取ul中所有子节点 4. var ul\_children = getChildNodes(ul); 5. // 创建一个元素 6. var li = document.createElement("li"); 7. // 设置li元素的内部文本 8. li.innerHTML = 11; 9. // insertBefore接收两个参数， 第一个参数是要追加的新的元素， 第二个参数可以是null 10. // ul.insertBefore(li, null); // 此时， 该方法的效果等价于appendChild方法 11. // 我们要将创建出来的li元素追加到li5的前面 12. ul.insertBefore(li, ul\_children[4]); |

结果：

|  |
| --- |
|  |

## 替换

使用方式：

father.replaceChild(newChild, oldChild)

newChild: 替换上的元素

oldChild: 替换下的元素

返回值： oldChild

最终效果： newChild替换掉oldChild

举例：

|  |
| --- |
| 1. // 获取元素 2. var ul = document.getElementsByTagName("ul")[0]; 3. // 获取所有的li元素 4. var lis = document.getElementsByTagName("li"); 5. // 创建一个元素 6. var li = document.createElement("li"); 7. // 设置内部文本 8. li.innerHTML = 1111; 9. setTimeout(function() { 10. // 替换掉li5 11. ul.replaceChild(li, lis[4]); 12. }, 3000) |

结果：

|  |
| --- |
|  |

## 克隆

使用方式：

node.cloneNode(bool)

接收一个参数是布尔值， 默认是false, 表示只复制自身， 如果传递的true， 表示连同子节点一起复制

举例：

|  |
| --- |
| 1. // 获取元素 2. var ul = document.getElementsByTagName("ul")[0]; 3. // 克隆 4. var newUl = ul.cloneNode(); 5. // 传递true 6. var newUl2 = ul.cloneNode(true); |

结果：

|  |
| --- |
|  |

# jquery中的节点操作

## 4.1 创建元素

我们可以利用$函数的功能去创建一个元素

举例：

|  |
| --- |
| 1. // 创建一个元素 2. var $div = $("<div class='aaa' data-info='nihao'>conetent</div>"); |

结果：

|  |
| --- |
|  |

## 4.2 上树

jquery中上树的方法有很多种， 可以是父元素选择子元素，可以是子元素选择父元素， 还可以兄弟选择兄弟

父元素选择子元素:

$(dom).append(child): 在dom的后面追加child

$(dom).prepend(child): 在dom的前面追加child

子元素选择父元素：

$(child).appendTo(dom): 将child追加到dom的后面

$(child).prependTo(dom): 将child追加到dom的前面

|  |
| --- |
| 1. // 父元素选择子元素 往后面追加 2. var $list = $("#list"); 3. 创建一个li元素 4. var li = document.createElement("li"); 5. 设置内部文本 6. li.innerHTML = 1; 7. 原生元素 8. $list.append(li); 9. jquery对象 10. $list.append($("<li>2</li>")) 11. html 12. $list.append("<li>3</li>"); 13. 父元素选择子元素 往前面追加 14. var $list = $("#list"); 15. 创建一个li元素 16. var li = document.createElement("li"); 17. 设置内部文本 18. li.innerHTML = 1; 19. 原生元素 20. $list.prepend(li); 21. jquery对象 22. $list.prepend($("<li>2</li>")) 23. html 24. $list.prepend("<li>3</li>"); 25. 子元素选择父元素 往后面追加 26. var $li1 = $("<li>1</li>"); 27. var $li2 = $("<li>2</li>"); 28. var $li3 = $("<li>3</li>"); 29. 原生元素 30. $li1.appendTo(document.getElementById("list")); 31. jquery对象 32. $li2.appendTo($("#list")); 33. 选择器 34. $li3.appendTo("#list"); 35. // 子元素选择父元素 往前面追加 36. var $li1 = $("<li>1</li>"); 37. var $li2 = $("<li>2</li>"); 38. var $li3 = $("<li>3</li>"); 39. // 原生元素 40. $li1.prependTo(document.getElementById("list")); 41. // jquery对象 42. $li2.prependTo($("#list")); 43. // 选择器 44. $li3.prependTo("#list"); |

兄弟选择兄弟：

$(dom).after(element): 往dom的后面追加某个元素

$(dom).before(element)： 往dom的前面追加某个元素

$(dom).insertBefore(element): 将dom插入到某个元素的前面

$(dom).insertAfter(element): 将dom插入到某个元素的后面

|  |
| --- |
| // 创建一个li元素  var $li = $("<li>999</li>");  var $li1 = $("<li>000</li>");  // 将创建出来的li插入到four的前面  $li.insertBefore($("#four"));  // 将创建出来的li1插入到four的后面  $li1.insertAfter($("#four"));  // 在four的前面追加某个元素  $("#four").before("<li>哈哈</li>");  // 在four的后面追加某个元素  $("#four").after("<li>嘎嘎</li>"); |

## 4.3 wrap

DOM结构：

|  |
| --- |
| 1. <p>A</p> 2. <p>B</p> 3. <p>C</p> 4. <p>D</p> 5. <p>E</p> |

执行代码：

|  |
| --- |
| 1. // wrap 用于将匹配的元素外层多一层元素 2. $("p").wrap("<div></div>"); |

结果：

|  |
| --- |
|  |

## 4.4 wrapAll

DOM结构：

|  |
| --- |
| 1. <p>A</p> 2. <p>B</p> 3. <p>C</p> 4. <p>D</p> 5. <p>E</p> |

执行代码：

|  |
| --- |
| 1. // wrapAll 用于将所有匹配到的元素的外层添加一层元素 2. $("p").wrapAll("<div></div>"); |

结果：

|  |
| --- |
|  |

## 4.5 unwrap

DOM结构：

|  |
| --- |
| 1. <div id="box"> 2. <div> 3. <h3 id="h3">h3</h3> 4. </div> 5. </div> |

执行代码：

|  |
| --- |
| 1. // unwrap 用于将匹配到的元素去掉一层外层元素 2. $("#h3").unwrap(); |

结果：

|  |
| --- |
|  |

## 4.6 replaceWith、 replaceAll

DOM结构：

|  |
| --- |
| 1. <p>A</p> 2. <p>B</p> 3. <p>C</p> 4. <p>D</p> 5. <p>E</p> |

执行代码：

|  |
| --- |
| 1. // replaceWith 将匹配到的元素替换成参数元素 2. $("p").replaceWith("<div>123</div>"); 3. // replaceAll 将指定的元素替换掉参数元素 4. $("<div>123</div>").replaceAll("p"); |

结果都是：

|  |
| --- |
|  |

## 4.7 empty

DOM结构：

|  |
| --- |
| 1. <div id="box"> 2. <ul> 3. <li>1</li> 4. <li>2</li> 5. <li>3</li> 6. <li>4</li> 7. <li>5</li> 8. </ul> 9. </div> |

执行代码：

|  |
| --- |
| 1. // empty 表示清空后代节点， 自己还在 2. $("#box").empty(); |

结果：

|  |
| --- |
|  |

## 4.8 remove

DOM结构：

|  |
| --- |
| 1. <div id="box"> 2. <ul> 3. <li>1</li> 4. <li>2</li> 5. <li>3</li> 6. <li>4</li> 7. <li>5</li> 8. </ul> 9. </div> |

执行代码：

|  |
| --- |
| 1. // remove 表示自杀， 自己和后代都没有了 2. $("#box").remove(); |

结果：

|  |
| --- |
|  |

## 4.9 clone

clone方法接收一个参数是bool，默认是false， 天生连同子节点一起复制， 当传递true的时候，连同事件一起复制

|  |
| --- |
| // 给box注册一个点击事件  $("#box").on("click", function() {  alert(123);  })  // 再次克隆 并且传递true  $("#box").after($("#box").clone(true)); |

结果：

|  |
| --- |
|  |

# call与apply方法

call方法和apply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作用域

简单来说: 谁调用，就指向谁

举例：

|  |
| --- |
| 1. // 定义一个函数 2. function fun() { 3. console.log(this, arguments); 4. } 5. // 函数自执行 6. fun(1, 2, 3); 7. // 作为事件属性函数 8. document.onclick = fun; 9. document.body.onclick = fun; 10. // 定义对象 11. var obj = { 12. a: fun 13. } 14. obj.a(); |

结果：

|  |
| --- |
|  |

## call

该方法的作用是执行函数并且改变函数的作用域

接收多个参数

第一个参数就是要改变的作用域

从第二个参数开始，都是原函数所需要的参数

举例：

|  |
| --- |
| 1. // 使用call方法 2. fun.call(document, 1, 2, 3); |

结果：

|  |
| --- |
|  |

## apply

该方法与call方法类似， 都是执行函数并改变作用域

它们的区别是参数

apply方法可以接收两个参数

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

举例：

|  |
| --- |
| 1. // 使用apply方法 2. fun.apply(document.body, [1, 2, 3]); |

结果：

|  |
| --- |
|  |

# 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，直到最顶层的元素

当元素处于最精确元素的时候， 是不分区捕获和冒泡的

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. background-color: blue; 11. margin: 0 auto; 12. } 13. #box2 { 14. width: 100px; 15. height: 100px; 16. background-color: orange; 17. margin: 0 auto; 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. // 分别为每一个box注册点击事件 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：

|  |
| --- |
|  |

## DOM2级绑定方式

使用方式：

dom.addEventListener(type, fn, bool)

type: 事件的类型， 是一个字符串， 注意: 不带on, 比如: click、 mousedown、mouseup

fn: 事件处理函数

bool: 是一个布尔值， 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. // 使用DOM2级绑定方式 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) |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |