# jQuery框架

# 目录

[jQuery框架 1](#_Toc21028)

[目录 2](#_Toc26977)

[一、认识框架 3](#_Toc10764)

[二、节点关系 4](#_Toc25810)

[2.1 原生JS中的节点关系 4](#_Toc20063)

[2.2 jQuery中的节点关系 6](#_Toc21391)

[三、节点操作 8](#_Toc4132)

[3.1 原生JavaScript节点操作 8](#_Toc228)

[3.1.1 createElement()和appendChild() 8](#_Toc23129)

[3.1.2 insertBefore 9](#_Toc11593)

[3.1.3 removeChild() 9](#_Toc12094)

[3.1.4 replaceChild() 9](#_Toc12034)

[3.1.5 cloneNode() 9](#_Toc32000)

[3.2 jQuery中的节点操作 10](#_Toc20630)

[3.2.1 append就是往最后添加： 10](#_Toc27859)

[3.2.2 appendTo 表示被动， 儿子appendTo父亲 10](#_Toc9231)

[3.2.3 after 插入兄弟 before 插入兄弟 11](#_Toc10165)

[3.2.4 insertBefore、insertAfter 11](#_Toc31010)

[3.2.5 wrap、warpAll 11](#_Toc9842)

[3.2.6 empty() 11](#_Toc21934)

[3.2.7 remove() 12](#_Toc3746)

[3.2.8 clone() 12](#_Toc2877)

一、认识框架

**计算机世界和物理化学不一样，计算机世界底层API少，逐渐一层一层的API丰富起来**。所谓的API就是你能够调用的方法、函数、语法。计算机的最底层是二进制，上面就是机器语言，再上面是汇编语言，再上面是高级语言。

汇编语言中，实际上就提供了不到100个API，比如汇编语言中求1~100的和：

|  |
| --- |
| 1. DSEG SEGMENT 2. RESULT DW ?;存放和 3. DSEG ENDS 4. CSEG SEGMENT 5. ASSUME CS:CSEG,DS:DSEG 6. START: 7. MOV AX,DSEG 8. MOV DS,AX 9. MOV AX,1;从1开始加 10. MOV RESULT,0;和的初值赋0 11. MOV CX,100;循环100次 12. NEXT: 13. ADD RESULT,AX;每次向和中累加AX 14. INC AX;然后AX加1 15. LOOP NEXT 16. MOV AX,4C00H;结束程序 17. INT 21H 18. CSEG ENDS 19. END START |

这里都是内存的变化，用内存的变化来编程。比如MOV表示把变量从内存一个地方移动到另一个地方。汇编语言非常晦涩，都是内存的变化。所以高级语言应运而生，C语言是最最著名的代表：

|  |
| --- |
| 1. int sum = 0; 2. for(int i = 1 ; i <= 100 ; i++){ 3. sum += i; 4. } 5. prinf("%d",sum); |

c语言编译的过程，实际上就是把for这些字词翻译为汇编语言。大学里一门课程《编译原理》就是研究编译过程的。

上层的框架屏蔽了下层的语言的一些麻烦、不方便的东西，并且提供更方便的API。

jQuery就是干这个事情的，把JS中的不方便封装起来，暴露的API都是非常简便的。

**jQuery的哲学就是DOM编程领域的霸主，操作DOM节点、绑定监听、运动、css样式、Ajax等等都有封装。**

工作上都是用jQuery，如果不用jQuery也是用类似的东西。没有人会不用轮子去开发页面效果。

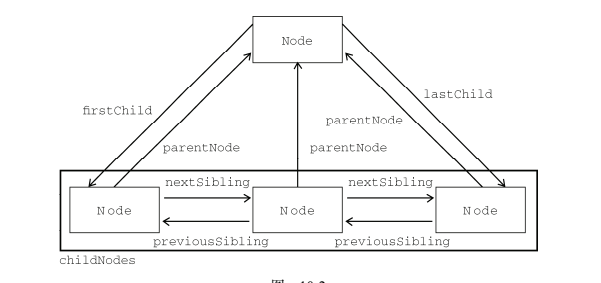
JavaScript中Library表示“库”，如果这个库的功能很强大，甚至颠覆了传统编程的语法、行文习惯，我们就可以叫做“框架”。

# 二、节点关系

## 2.1 原生JS中的节点关系

原生JS中提供的节点关系很少：

childNodes、firstChild、lastChild、parentNode、nextSibling、previousSibling



关于childNodes一定要记住，IE6、7、8和高级浏览器的不一致，高级浏览器视所有的换行为空文本节点，而IE6、7、8无视这个空文本节点。

|  |
| --- |
| 1. <div id="box"> 2. <p></p> 3. <p></p> 4. <p></p> 5. <p></p> 6. </div> |

|  |
| --- |
| 1. oDiv.childNodes.length; //chrome数值是9 IE6、7、8数值是4 |

为了解决兼容性问题（到底空文本算不算儿子，所以要用函数来解决）：

|  |
| --- |
| 1. //可以封装一个children函数，这个函数返回obj的所有节点儿子 2. function children(obj,num){ 3. var arr = []; //存放所有儿子 4. for(var i = 0 ; i < obj.childNodes.length ; i++){ 5. //寻找真正的HTML节点，不要考虑文本： 6. if(obj.childNodes[i].nodeType == 1){ 7. arr.push(obj.childNodes[i]); 8. } 9. } 10. //返回的是：如果用户传入了num，返回某一个儿子；如果没有num返回所有儿子 11. return num ? arr[num] : arr; 12. } |

同样的道理firstChild、lastChild也有兼容性问题。

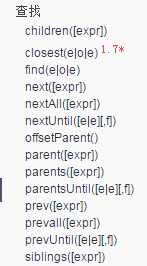
这个兼容性问题一直存在，比如前一个兄弟：

|  |
| --- |
| 1. <div id="box"> 2. <p></p> 3. <p></p> 4. <p></p> 5. <p></p> 6. <p id="xiaoming">小明</p> 7. <p></p> 8. <p></p> 9. <p></p> 10. </div> |

兼容问题：

|  |
| --- |
| 1. xiaoming.previousSibling; //chrome是空文本节点； IE6、7、8就会认为是上一个p |

## 2.2 jQuery中的节点关系



感恩的心，太好用了！！注意，都是方法！！因为里面暗含了迭代器了。

● children()方法

所有亲儿子节点。就是儿子，孙子不在children()里面。

让box的所有儿子变红：

|  |
| --- |
| 1. $("#box").children().css("background-color","red"); |

儿子有很多，我们可以用选择器来表示怎么样的儿子：

|  |
| --- |
| 1. $("#box").children(**"h3"**).css("background-color","red"); |

还可以用筛选器：

|  |
| --- |
| 1. $("#box").children(**":odd"**).css("background-color","red"); |

jQuey屏蔽掉了浏览器兼容问题，不存在空文本节点也是儿子了，事实上children()返回的只能是节点，所有的文本都不算做是儿子了。

● find()

所有后代元素。和children()不一样，children()返回的是自己的亲儿子元素列表，而find()返回的是自己的后代所有元素的列表。

|  |
| --- |
| 1. $("#box").find("p").css("background-color","red"); |

注意，和children()方法不一样，find()方法里面，必须写参数，表示后代的谁？

说白了find是寻找的意思，就是你要在后代寻找谁。

● parent()

亲爸爸。任何元素都只有一个亲爸爸。

|  |
| --- |
| 1. $("p").parent().css("background-color","red"); |

● parents()

这个元素的所有的祖先节点。

● siblings()

亲兄弟

|  |
| --- |
| 1. $(".xiaoming").siblings().css("background-color","red"); |

同样的，jQuery只返回节点元素，对于文本、注释都不视为兄弟。

可以加选择器、筛选器：

|  |
| --- |
| 1. $(".xiaoming").siblings(":odd").css("background-color","red"); |

排他，妈妈再也不用担心我写for循环了：

|  |
| --- |
| 1. $(this).addClass("cur").siblings().removeClass("cur"); |

我加cur，我的兄弟们去cur。

● prev()、next()、prevAll()、nextAll()

前一个兄弟、后一个兄弟、前所有兄弟、后所有兄弟。

总结：感恩！jQuery屏蔽了所有的不兼容，只选择nodeType=1的元素，不选择文本、注释等等。

并且不用for循环了，一次性得到所有东西了，原理我们后面的课程揭示。

# 三、节点操作

HTML节点我们原来最多最多就是改改HTML属性，比如src属性改改；或者改改css样式，比如.style或者.css()。

现在的问题是，我们要增加节点、删除节点、移动节点、替换节点。

## 3.1 原生JavaScript节点操作

### 3.1.1 createElement()和appendChild()

|  |
| --- |
| 1. var ul = document.getElementsByTagName("ul")[0]; 2. //创建一个li标签，用变量oLi来表示。创建出来的节点不是任何节点的儿子， 3. //也就是说没有在DOM树上， 4. var oLi = **document.createElement**("li"); 5. oLi.innerHTML = "DDDD"; //改变这个节点里面的内容 6. //把新创建的节点，追加到DOM树上 7. **ul.appendChild(oLi);** |

创建节点的API：

|  |
| --- |
| 1. document.createElement() |

create创建，Element元素。接收一个参数，就是创建的标签是什么。

追加节点的API：

创建出来的节点不在DOM树上，所以就应该用appendChild()来添加到DOM树上：

|  |
| --- |
| 1. 父亲.appendChild(新儿子); |

appendChild()一般来说就是用来追加新创建的节点。如果你试图把页面上已经有的节点，appendChild()到别的地方，那么这个节点将移动。也就是说，同一个节点不可能在页面上的两个地方出现。

比如结构：

|  |
| --- |
| 1. <div id="box1"> 2. <p id="xiaoming">我是小明</p> 3. </div> 4. <div id="box2"> 5. </div> |

语句：

|  |
| --- |
| 1. box2.appendChild(xiaoming); |

将使xiaoming移动位置，从box1里面移动到box2里面。

事实上，工作的时候很少使用createElement。因为innerHTML足够好用，innerHTML也可以用来创建节点，**甚至效率比createElement还高**。

|  |
| --- |
| 1. box.innerHTML = "<ul><li>哈哈哈</li><li>做顿饭</li></ul>"; |

### 3.1.2 insertBefore

我们刚才说的appendChild是把新节点在父亲的所有儿子后添加，也就是说添加的节点就是父亲的最后一个儿子。

我们可以在任意一个位置添加节点。

|  |
| --- |
| 1. 父亲.insertBefore(新儿子,原有标杆儿子); |

会在原有标杆儿子之前插入。

如果想每次添加都在开头添加，那么就是：

|  |
| --- |
| 1. ul.insertBefore(oLi, lis[0]); |

lis这个变量是动态的，这次添加的li，下回就是lis[0]

### 3.1.3 removeChild()

|  |
| --- |
| 1. 父亲.removeChild(儿子); |

如果要自杀，也要找到爸爸

|  |
| --- |
| 1. this.parentNode.removeChild(this); |

### 3.1.4 replaceChild()

替换节点

|  |
| --- |
| 1. 父亲.replaceChild(新儿子, 老儿子); |

### 3.1.5 cloneNode()

克隆节点，参数true表示深复制，节点里面的所有内容一同复制。

复制之后的节点是个孤儿节点，所以也需要使用appendChild、inserBefore、replaceChild来添加上DOM树。

|  |
| --- |
| 1. ul.appendChild(lis[0].cloneNode(true)); |

## 3.2 jQuery中的节点操作



### 3.2.1 append就是往最后添加：

父亲.append(儿子)

|  |
| --- |
| 1. $("#box").append("<p>么么哒</p>"); |

jQuery节点操作非常简单，不用createElement的过程了。实际上内部就是用innerHTML来给你加进去的。

特别的，如果我们想先创建一个孤儿节点，给这个孤儿节点加样式、加监听、加动画，可以的，那么创建孤儿节点，也是用$()函数。 $()不仅仅能够选择页面上的节点，也能够创建新的孤儿节点：

|  |
| --- |
| 1. var $obj = **$(**"<p></p>"**)**; 2. $obj.css("background-color","red"); //jQuery对象就可以直接调用css方法 3. $("#box").append($obj); |

### 3.2.2 appendTo 表示被动， 儿子appendTo父亲

|  |
| --- |
| 1. $("<p>么么哒</p>").appendTo($("#box")); |

prepend 插入为第一个儿子， prependTo表示被动

|  |
| --- |
| 1. $("<p>么么哒啊！</p>").prependTo($("#box")); |

### 3.2.3 after 插入兄弟 before 插入兄弟

比如HTML结构：

|  |
| --- |
| 1. <div id="box"> 2. <p>哈哈</p> 3. <p>嘻嘻</p> 4. </div> |

执行代码：

|  |
| --- |
| 1. $("p").after("<h3>我是h3</h3>"); |

HTML将变为：

|  |
| --- |
| 1. <div id="box"> 2. <p>哈哈</p> 3. <h3>我是h3</h3> 4. <p>嘻嘻</p> 5. <h3>我是h3</h3> 6. </div> |

### 3.2.4 insertBefore、insertAfter

比如HTML结构：

|  |
| --- |
| 1. <div id="box"> 2. <p>000</p> 3. <p>111</p> 4. <p>222</p> 5. <p>333</p> 6. <p>444</p> 7. <p>555</p> 8. </div> |

代码：

|  |
| --- |
| 1. $("<p>么么哒</p>").insertBefore($("p")[2]); |

HTML结构将变为：

|  |
| --- |
| 1. <div id="box"> 2. <p>000</p> 3. <p>111</p> 4. <p>么么哒</p> 5. <p>222</p> 6. <p>333</p> 7. <p>444</p> 8. <p>555</p> 9. </div> |

### 3.2.5 wrap、warpAll

在外面包裹

|  |
| --- |
| 1. $("a").wrap("<span></span>"); //每个a都用span包裹起来 2. $("a").wrapAll("<span></span>"); //所有a都用span包裹起来 |

### 3.2.6 empty()

|  |
| --- |
| 1. $("p").empty(); |

等价于

|  |
| --- |
| 1. $("p").html(“”); |

### 3.2.7 remove()

删除页面上所有p标签：

|  |
| --- |
| 1. $("p").remove(); |

### 3.2.8 clone()

克隆节点

克隆ul li里面的第1个，然后ul追加：

|  |
| --- |
| 1. $(“ul”).append($(“ul li”).eq(0).clone()); |

反过来：

|  |
| --- |
| 1. $(“ul li”).eq(0).clone().appendTo($(“ul”)); |

不要背API，常用的慢慢就记住了。要思考业务逻辑。