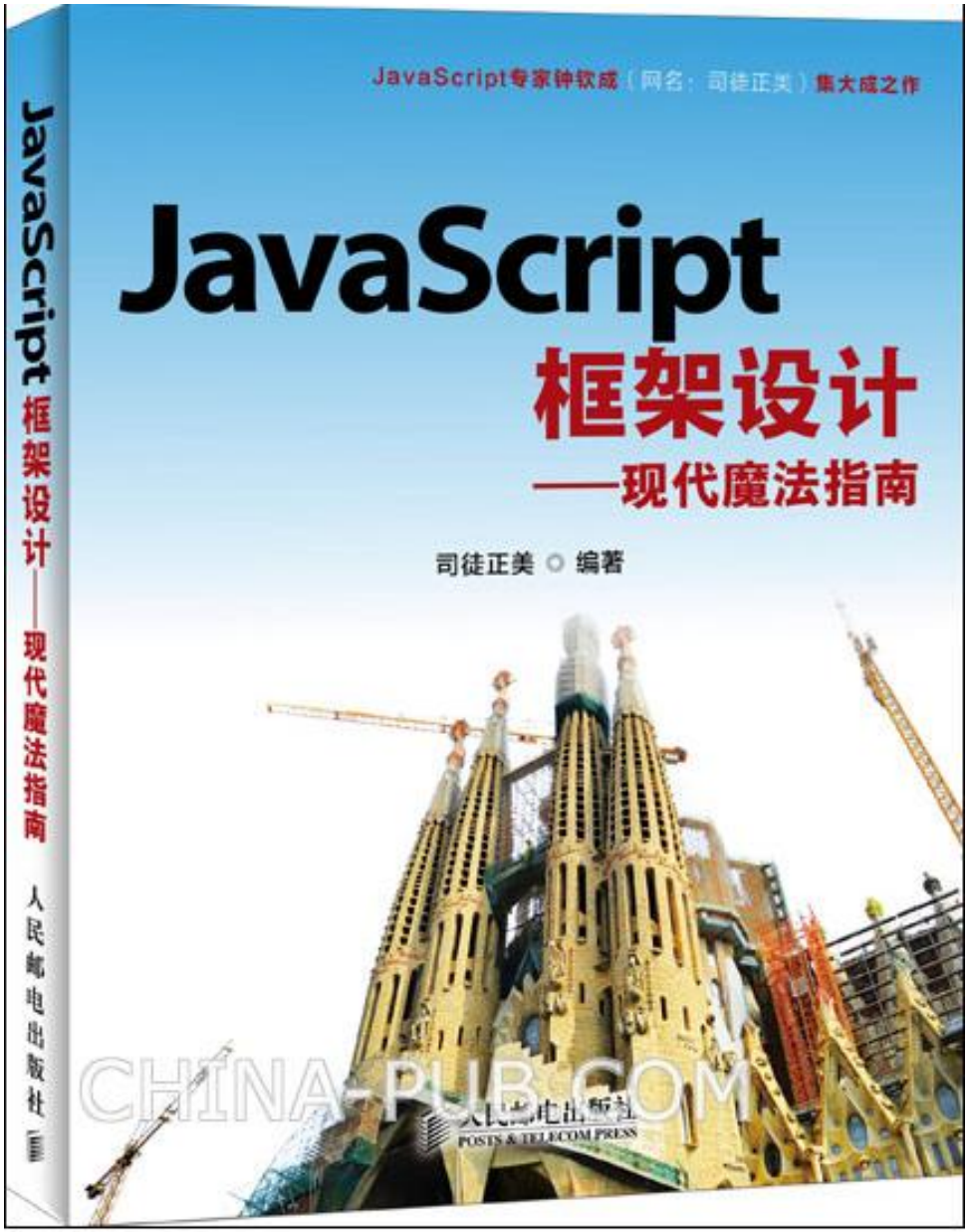
JavaScript专家钟软成（网名：司徒正美）集大成之作
JavaScript
框架设计
——现代魔法指南
司徒正美 ○ 编著
JavaScript 框架设计——现代魔法指南
人民邮电出版社
CHINA-PUB.COM
POSTS & TELECOM PRESS

第 10 章　属性模块

通常我们把对象的非函数成员叫做属性。对于元素节点来说，其属性大体分成两大类，固有属性与自定义属性（特性）。固有属性一般遵循驼峰命名风格，拥有默认值，并且无法删除。自定义属性是用户随意添加的键值对，由于元素节点也是一个普通的 JavaScript 对象，没有什么严格的访问操作，因此命名风格林林总总，值的类型也乱七八糟。但是随意添加属性显然不够安全，比如引起循环引用什么的，因此，浏览器提供了一组 API 来供人们操作自定义属性，即 setAttribute、getAttribute、removeAttribute。当然还有其他 API，不过这是标准套装，只有在 IE6、IE7 那样糟糕的环境下，我们才求助于其他 API，一般情况下这三个足矣。我们通称它们为 DOM 属性系统。DOM 属性系统对属性名会进行小写化处理，属性值会统一转字符串。

```
var el = document.createElement("div")
el.setAttribute("xxx", "1")
el.setAttribute("XxX", "2")
el.setAttribute("XXx", "3")
console.log(el.getAttribute("xxx"))
console.log(el.getAttribute("XxX"))
```

IE6、IE7 会返回"1"，其他浏览器返回“3”。在前端的世界，我们真是走到哪都能碰到兼容性问题。这只是冰山一角，IE6、IE7 在处理固有属性时要求进行名字映射，比如 class 变成 className，for 变成 htmlFor。对于布尔属性（一些只返回布尔的属性），浏览器间的差异更大，具体如表 10.1 所示。

```
<input type="radio"  id="aaa">
<input type="radio" checked id="bbb">
<input type="radio" checked="checked" id="ccc">
<input type="radio" checked="true" id="ddd">
<input type="radio" checked="xxx" id="eee">

"aaa,bbb,ccc,ddd,eee".replace(/\w+/g,function( id ){
   var elem = document.getElementById( id )
   console.log(elem.getAttribute("checked"));
})
```

表 10.1

	#aaa	#bbb	#ccc	#ddd	#eee
IE7	false	true	true	true	true
IE8	""	"checked"	"checked"	"checked"	"checked"

续表

IE9	null	""	"checked"	"true"	"xxx"
FF15	null	""	"checked"	"true"	"xxx"
Chrome23	null	""	"checked"	"true"	"xxx"

因此框架很有必要提供一些 API 来屏蔽这些差异性。但在 IE6 统治时期，这个需求并不明显，因为 IE6、IE7 不区分固有属性与自定属性，setAttribute 与 getAttribute 在当时的人看来只是一个语法糖，用 el.setAttribute("innerHTML","xxxx")与用 el.innerHTML = "xxx"效果一样，而且后者更方便。即使早期应用那么广泛的 Prototype.js，提供属性操作 API 也非常贫乏，只有 identify、readAttribute、writeAttribute、hasAttribute、classNames、hasClassName、addClassName、removeClassName、toggleClassName 与$F 方法。Prototype.js 也察觉到固有属性与自定义属性在 DOM 属性系统的差异，在它的内部，搞了个 Element._attributeTranslations。然而，Prototype.js 这个属性系统内部还是优先使用 el[name] 方式来操作属性，而不是 set/getAttribute（接下来的章节，我会分析它的实现）。

jQuery 早期的 attr 方法，其行为与 Prototype 一模一样。只不过 jQuery1.6 之前，是使用 attr 方法同时实现了读、写、删掉这三种操作。从易用性来说，不区分固有属性与自定义属性，由框架自动内部处理应该比 attr、prop 分家更容易接受。那么是什么逼迫 jQuery 这样做的呢？是选择器引擎。

jQuery 是最早以选择器为向导的类库。它最开始的选择器引擎是 Xpath 式，后来换成 Sizzle，以 CSS 表达式风格来选取元素。在 CSS2.1 引入了属性选择器[aaa=bbb]，IE7 也开始残缺支持。Sizzle 当然毫不含糊地实现了这语法。属性选择器是最早突破类名与 ID 的限制求取元素的。为了显摆它的强大，设计者让它拥有多种形态，满足人们各种匹配需要。比如，它可以只写属性名，[checked]，那么上例中的后四个元素都选中。[checked=true]选中第四个元素。[checked=xxx]选中第四个元素。true 与 xxx 都是用户在标签的预设值。而一字不差地取回这个预设值的工作也只有 setAttribute 才能做到。根据标准，setAttribute 是应该返回用户设置的那个字符串。而 el[xxx]这样的取法就不一定了，比如 el.checked 就返回布尔值，表示两种状态。这种通过状态取元素的方式就不归属性选择器管了，CSS3 又新设了个状态伪类满足人们的需求。

此外，属性还能以[name^=value]、[name*=value]、[name$=value]等更精致的方式来甄选元素，而这一切都建立在获取用户预设值的基础上。因此 jQuery 下了很大决心，把 prop 从 attr 切割出来。虽然为了满足用户的向前兼容需求，又偷偷地让 attr 做了 prop 的事，但以此为契机，jQuery 团队挖掘出更多兼容性问题与相应解决方案。元素内部撑起整个属性系统的 attributes 类数组属性也从幕后走到前台，为世人所知。

浏览器经过这么多年的发展，谁也说不清某个元素节点拥有多少个属性。for..in 循环也不行，

显式属性就是被显式设置的属性，分两种情况，一种是写在标签内的 HTML 属性，一种是通过 setAttribute 动态设置的属性，这些属性不分固有还是自定义，只要设置了，就出现在 attributes 中。在 IE6、IE7 中，我们也可以通过特性节点的 specified 属性判定它是否被显式设置。在 IE8 或其他浏览器，我们想判定一个属性是否为显式属性，直接用 hasAttribute API 判定。

```
var  isSpecified = !"1"[0] ? function(el, attr){
     return el.hasAttribute(attr)
} : function(el, attr){
     var val = el.attributes[attr]
     return !!val && val.specified
}
```

此外，HTML5 对属性进行了更多的分类，从外包到不同的对象。dataset 对象装载着所有以 data- 开头的自定义属性。classList 装载着元素的所有类名，并且提供一套 API 来操作它们。formData 装载着所有要提供到后台的数据，以表单元素的 name 值与 value 值构成的不透明对象。尽管如此，还是有大量属性是没有编制的，它们代表着元素的各种状态以及其他元素的联结。正因为如此，它们的值才五花八门，如 uniqueNumber、tabIndex、colspan、rowspan 为整数，designMode、unselectable、autocomplete 的值不是 on 就是 off，iframe 通过 frameborder 的值是 0 还是 1 决定显示边框，通过 scrolling 的值是 yes 还是 no 决定显示滚动条，表单元素的 form 属性总是指向其外围的表单对象，表单元素的 checked、disabled、readOnly 等属性总是返回布尔值……

面对如此庞杂的属性，主流框架也纷纷建立了对应的模块来整治它。在 jQuery 中有 attributes 模块，YUI3 是 dom-class、dom-attrs 模块，dojo 是 dom-attr、dom-prop、dom-class 模块。从内容来看，类名都被单独提出来处理，在 jQuery 中，表单元素的 value 也被单独提出来处理。Prototype.js 虽然没有划分出来，但对付一般属性有 readAttribute 与 writeAttribute，ID 有 identify，表单元素的 value 有$F，类名更是对应多个方法，这个阵营与 jQuery 的属性模块一模一样。在 1.5 之前，Prototype.js 的属性模块一直是 jQuery 属性模块的蓝本。

10.1 如何区分固有属性与自定义属性

在 jQuery、mass Framework 中，提供两组方法来处理它们。但用户首先要知道他正在处理的东西是何物。虽然我们可以在以下链接查到每个元素拥有什么方法与属性，但显然不是每个人都那么勤奋。

http://msdn.microsoft.com/library/ms533029%28v=VS.85%29.aspx

我们需要做些实验找出其规律。

```
<!doctype html>
<html lang="en">
```

```
        var a = document.getElementById("test");
        a.setAttribute("title", "title");
        a.setAttribute("title2", "title2");
        alert(a.parentNode.innerHTML);
      }
    </script>
  </head>
  <body>
    <p><strong id="aaa">司徒正美</strong></p>
    <p><button type="button" onclick="test()">点我，进行测试</button></p>
  </body>
</html>
```

IE8 下打印出：

```
<STRONG id=test title=title title2="title2">司徒正美</STRONG>
```

Firefox15、Chrome23、IE9 下打印出：

```
<strong title2="title2" title="title" id="test">司徒正美</strong>
```

再将 a.setAttribute("title", "title"); a.setAttribute("title2", "title2")这两行改成 a.title = "title";a.title2 = "title2"。

IE8 下打印出：

```
<STRONG id=test title=title title2="title2">司徒正美</STRONG>
```

Firefox15、Chrome23、IE9 下打印出：

```
<strong title2="title2" title="title" id="test">司徒正美</strong>
```

经过观察，旧版本 IE 下固有属性，如 title、id 是不带引号的，自定义属性是带引号的，但在标准浏览器下，我们无法区分，否决此方案。

在元素中有一个 attributes 属性，里面有许多特性节点，这些特性节点在 IE 下有一种名为 expando 的布尔属性，可以判定它是否为自定义属性。但在标准浏览器下没有此属性，否决此方案。

```
function isAttribute(attr, host){ //仅有 IE
  var attrs = host.attributes;
  return attrs[attr] && attrs.expando == true
}
```

我们再换一个角度来看，如果是固有属性，以 el[xxx] = yyy 的形式赋值，再用 el.getAttribute() 来取值，肯定能取到东西，但自定义属性就不一样。

```
var a = document.getElementById("test");
```

```
console.log(a.custom)         //undefined
console.log(typeof a.custom)   //"undefined"
```

不过要注意 IE6、IE7 下的特例：

```
a.setAttribute("innerHTML","xxx")
console.log(a.innerHTML)          //"xxx"
```

即使如此，我们也可以轻松绕过这个陷井，建一个干净的同类型元素作为测试样本就行了。

```
var a = document.createElement("div")
console.log(a.getAttribute("title"))
console.log(a.getAttribute("innerHTML"))
console.log(a.getAttribute("xxx"))
console.log(a.title)
console.log(a.innerHTML)
console.log(a.xxx)
```

IE6、IE7 下返回"","", null, "","",undefined。IE8、IE9、Chrome23、FF15，Opera12 下返回 null，null,null,"","",undefined。

因此我们可以推导出这样一个方法，回答我们这一节的标题。

```
function isAttribute(attr, host){
   //有些属性是特殊元素才有的，需要用到第二个参数
   host = host || document.createElement("div");
   return host.getAttribute(attr) === null && host[attr] === void 0
}
```

10.2 如何判定浏览器是否区分固有属性与自定义属性

经过社区的努力，现在大家都知道 IE6、IE7 不区分固有属性与自定义属性。这带来的结果是，对某个固有属性进行 setAttribute，我们不需要名字映射就能生效。但如果想通过浏览器嗅探法来识别 IE6、IE7，是远远不够的，因为用户可能使用旧版的标准浏览器上网。另外，我们也不得不考虑国内可恶的加壳 IE 浏览器。因此我们最好是通过特征侦测来判定浏览器是否支持此特性。

mootools 与 jQuery 各自使用了两种截然不同的方法来判定。mootools 以属性法设置一个自定义属性，然后通 getAttribute 去取，看是不是等于预设值，是就证明它不区分固有属性与自定义属性。jQuer 则是先用 setAttribute 去设置 className，然后看它是当作固有属性还是自定义属性，如果是自定义属性，我们用 el.className 是取不到值的，具体如表 10.2 所示。

```
var el = document.createElement("div")
el.random = 'attribute';              //mootools
console.log(el.getAttribute("random") != 'attribute')
```

Mootools	false,	false	false	true	true	true
jQuery	false	false	true	true	true	true

要看哪个更精确，只需要 IE8 浏览器就行了。IE8 大肆重写内核，是区分固有属性与自定义属性的，因此 el.className 应该返回 undefined，导致结果为 true，因此 jQuery 获胜。jQuery 把这个特性称之为 getSetAttribute，意即 get/SetAttribute 没有 BUG；mass Framework 称之为 attrInnateName，意即不需要名字映射用原名就可以取值。它们都位于 supports 模块中。

10.3 IE 的属性系统的三次演变

微软在 IE4（1997 年）添加 setAttribute、getAttribute API，当时，DOM 标准（1998 年）还没有出来呢！而它的对手 NS6 到 2000 年才难产出来。

早期的 DOM API 于微软来说，只是它已有的一些方法的再包装，这些包装方法无法与它原来的那一套相媲美。document.getElementById("xxx")取元素节点，IE 下，这些带 ID 的元素节点自动就映射成一个个全局变量，直接 xxx 就拿到元素节点了，很便捷，或者使用 document.all[ID]来取，无论哪种都比标准的短；又如 getElementsByTagName，IE 下有 document.all.tags()方法，此方法直到 IE9 还有效。而 setAttribute("xxx","yyy")与 var ret = getAttribute("xxx")只不过是 el.xxx = "yyy"与 var ret = el.xxx 的另一种操作形式罢了。明白这一点我们就立即理解 IE 下这两个 API 的一些奇怪行为了。

el.setAttribute("className","aaa")是可行的，但 el.setAttribute("class","aaa")失败，因为我们可以用 className 修改类名，但不能用 class。

el.setAttribute("onclick", Function("alert(1)")) 是可行的，但 el.setAttribute("onClick", Function("alert(1)"))失败，因为我们是通过 el.onclick 绑定事件，而不是 el.onClick。

el.setAttribute("innerHTML"," test ")是可行的，因为我们可以用 innerHTML 添加内容。

element.setAttribute("style", "background-color: #fff; color: #000;")失败，因此 style 在 IE 下是个对象，"background-color: #fff; color: #000;"只能作为它的 cssText 属性的值。

DOM level 1 隔年就制定出来了。setAttribute/getAttribute 并没有微软想象得那么简单，它早期规定 getAttribute 必须也返回字符串，就算不存在也是空字符串。到后来，setAttribute/getAttribute 会对属性名进行小写化处理。用 getAttribute 去取没有显式设置的固有属性时，返回默认值（多数时候它为 null 或空字符串），对于没有显式设置的自定义属性，则返回 undefined。于是微软傻了眼，第一次改动就是匆匆忙忙支持小写化处理，并在 getAttribute 方法添加第二参数，以实现 DOM1 的效果。

getAttribute 的第二个参数有四个预设值：0 是默认，照顾 IE 早期的行为；1 属性名区分大小写；2 取出源代码中的原字符串值(注，IE5～IE7 对动态创建的节点没效，IE5～IE8 对布尔属性无

准浏览器也是这样做的。但标准浏览器很快就变脸了，统一返回用户的预设值。IE8 变得两边都不讨好，尽管它一心想与标准保持一致，标榜自己才是最标准的。比如它在当时还推出了 Object.defineProperty、querySelector、postMessage 等具有革命意义的新 API，但它们都与 W3C 争吵完的结果有出入。

第三次演变，不再对属性值进行干预，用户设什么返回什么，忠于用户的决定。对于这尘埃落定的方案，IE9 终于与标准吻合了。这也说明 IE 这种慢吞吞的大版本发布方式已经落伍了，虽然大家都对 FF 的版本狂飙有微词，但人家却保住了与 Chrome 叫板的地位。

综观 IE 的属性系统的悲剧，都因为微软总想抢占先机，而又与标准同步太慢所致。

第 11 章　事件系统

事件系统是一个框架非常重要的部分，用于响应用户的各种行为。

浏览器提供了三种层次的 API。

最原始的是写在元素标签内。

再次是在脚本中，以 el.onXXX= function 绑定的方式，通称为 DOM0 事件系统。

最后是多投事件系统，一个元素的同一类型事件可以绑定多个回调，通称为 DOM2 事件系统。由于浏览器大战，现存在两套 API。

- IE 与 Opera 方

绑定事件：el.attachEvent(“on”+type，callback)

卸载事件：el.detachEvent(“on”+type，callback)

创建事件：document.createEventObject()

派发事件：el.fireEvent(type, event)

- W3C 方

绑定事件：el.addEventListener(type, callback, [phase])

卸载事件：el.removeEventListener(type, callback, [phase])

创建事件：el.createEvent(types)

初始化事件：event.initEvent()

派发事件：el.dispatchEvent(event)

从 API 的数量与形式来看，W3C 提供的复杂很多，相对应也强大很多，下面我会逐一详述。

首先呈上我几个简单的实现。如果是简单的页面，就用简单的方式打发，没有必要动用框架。不过，事实上，整个事件系统就建立在它们的基础上。

```
function addEvent(el, type, callback, useCapture ){
   if(el.dispatchEvent){//W3C 方式优先
      el.addEventListener( type, callback, !!useCapture );
   }else {
      el.attachEvent( "on"+type, callback );
   }
```

```
    return callback;//返回 callback 方便卸载时用
}

function removeEvent(el, type, callback, useCapture ){
    if(el.dispatchEvent){//W3C 方式优先
        el.removeEventListener( type, callback, !!useCapture );
    }else {
        el.detachEvent( "on"+type, callback );
    }
}

function fireEvent(el, type, args, event){
    args = args || {}
    if(el.dispatchEvent){//W3C 方式优先
        event = document.createEvent("HTMLEvents");
        event.initEvent(type, true, true );
    }else {
        event = document.createEventObject();
    }
    for(var i in args)  if(args.hasOwnProperty(i)){
        event[i] = args[i]
    }
    if(el.dispatchEvent){
        el.dispatchEvent(event);
    }else{
        el.fireEvent('on'+type,event)
    }
}
```

11.1 onXXX 绑定方式的缺陷

onXXX 既可以写在 HTML 标签内，也可以独立出来，作为元素节点的一个特殊属性来处理。不过作为一种古老的绑定方式，它很难预测到后来人对这方面的扩展。

总结起来有以下不足。

（1）onXXX 对 DOM3 新增事件或 FF 某些私有实现无法支持，主要有以下事件：

DOMActivate
DOMAttrModified
DOMAttributeNameChanged
DOMCharacterDataModified
DOMContentLoaded
DOMElementNameChanged
DOMFocusIn
DOMFocusOut

DOMNodeRemoved

DOMNodeRemovedFromDocument

DOMSubtreeModified

MozMousePixelScroll

不过稍安勿躁，上面的事件就算是框架，也只用到 DOMContentLoaded 与 DOMMouseScroll。DOMContentLoaded 用于检测 DomReady，DOMMouseScroll 用于在 FF 模拟其他浏览器的 mousewheel 事件。

（2）onXXX 只允许元素每次绑定一个回调，重复绑定在冲掉之前的绑定。

（3）onXXX 在 IE 下回调没有参数，在其他浏览器下回调的第一个参数是事件对象。

（4）onXXX 只能在冒泡阶段可用。

11.2 attachEvent 的缺陷

attachEvent 是微软在 IE5 添加的 API，Opera 也支持，也对于 onXXX 方式，它可以允许同一元素同种事件绑定多个回调，也就是所谓的多投事件机制。但它带来的麻烦只多不少，存在以下几点缺陷。

（1）IE 下只支持微软系的事件，DOM3 事件一概不能用。

（2）IE 下 attchEvent 回调中的 this 不是指向被绑定元素，而是 window!

（3）IE 下同种事件绑定多个回调时，回调并不是按照绑定时的顺序依次触发的！

（4）IE 下 event 事件对象与 W3C 的存在太多差异了，有的无法对上号，比如 currentTarget。

（5）IE 还是只支持冒泡阶段。

不过 IE 还是不断进步的。

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <script>
            window.attachEvent("onload", function(e) {
                alert(e);
            });
        </script>
    </head>
    <body>
        <div>TODO write content</div>
    </body>
</html>
```

IE6、IE7（见图 11.1）