**jquery 浏览器性能优化**

1. 使用最新版本的jQuery

2. 用对选择器

在jQuery中，你可以用多种选择器，选择同一个网页元素。每种选择器的性能是不一样的，你应该了解它们的性能差异。

（1）最快的选择器：id选择器和元素标签选择器

举例来说，下面的语句性能最佳：

$(‘#id')

$(‘form')

$(‘input')

遇到这些选择器的时候，jQuery内部会自动调用浏览器的原生方法（比如getElementById()），所以它们的执行速度快。

（2）较慢的选择器：class选择器

$(‘.className')的性能，取决于不同的浏览器。

Firefox、Safari、Chrome、Opera浏览器，都有原生方法getElementByClassName()，所以速度并不慢。但是，IE5-IE8都没有部署这个方法，所以这个选择器在IE中会相当慢。

（3）最慢的选择器：伪类选择器和属性选择器

先来看例子。找出网页中所有的隐藏元素，就要用到伪类选择器：

$(‘:hidden')

属性选择器的例子则是：

$(‘[attribute=value]‘)

这两种语句是最慢的，因为浏览器没有针对它们的原生方法。但是，一些浏览器的新版本，增加了querySelector()和querySelectorAll()方法，因此会使这类选择器的性能有大幅提高。

3. 理解子元素和父元素的关系

下面六个选择器，都是从父元素中选择子元素。你知道哪个速度最快，哪个速度最慢吗？

$(‘.child', $parent)

$parent.find(‘.child')

$parent.children(‘.child')

$(‘#parent > .child')

$(‘#parent .child')

$(‘.child', $(‘#parent'))

我们一句句来看。

(1) $(‘.child', $parent)

这条语句的意思是，给定一个DOM对象，然后从中选择一个子元素。jQuery会自动把这条语句转成$.parent.find(‘child')，这会导致一定的性能损失。它比最快的形式慢了5%-10%。

(2) $parent.find(‘.child')

这条是最快的语句。.find()方法会调用浏览器的原生方法（getElementById，getElementByName，getElementByTagName等等），所以速度较快。

(3) $parent.children(‘.child')

这条语句在jQuery内部，会使用$.sibling()和javascript的nextSibling()方法，一个个遍历节点。它比最快的形式大约慢50%。

(4) $(‘#parent > .child')

jQuery内部使用Sizzle引擎，处理各种选择器。Sizzle引擎的选择顺序是从右到左，所以这条语句是先选.child，然后再一个个过滤出父元素#parent，这导致它比最快的形式大约慢70%。

(5) $(‘#parent .child')

这条语句与上一条是同样的情况。但是，上一条只选择直接的子元素，这一条可以于选择多级子元素，所以它的速度更慢，大概比最快的形式慢了77%。

(6) $(‘.child', $(‘#parent'))

jQuery内部会将这条语句转成$(‘#parent').find(‘.child')，比最快的形式慢了23%。

所以，最佳选择是$parent.find(‘.child')。而且，由于$parent往往在前面的操作已经生成，jQuery会进行缓存，所以进一步加快了执行速度。

具体的例子和比较结果，请看这里。

4. 不要过度使用jQuery

jQuery速度再快，也无法与原生的javascript方法相比。所以有原生方法可以使用的场合，尽量避免使用jQuery。

请看下面的例子，为a元素绑定一个处理点击事件的函数：

$(‘a').click(function(){

alert($(this).attr(‘id'));

});

这段代码的意思是，点击a元素后，弹出该元素的id属性。为了获取这个属性，必须连续两次调用jQuery，第一次是$(this)，第二次是attr(‘id')。

事实上，这种处理完全不必要。更正确的写法是，直接采用javascript原生方法，调用this.id：

$(‘a').click(function(){

alert(this.id);

});

根据测试，this.id的速度比$(this).attr(‘id')快了20多倍。

5. 做好缓存

选中某一个网页元素，是开销很大的步骤。所以，使用选择器的次数应该越少越好，并且尽可能缓存选中的结果，便于以后反复使用。

比如，下面这样的写法就是糟糕的写法：

jQuery(‘#top').find(‘p.classA');

jQuery(‘#top').find(‘p.classB');

更好的写法是：

var cached = jQuery(‘#top');

cached.find(‘p.classA');

cached.find(‘p.classB');

根据测试，缓存比不缓存，快了2-3倍。

6. 使用链式写法

jQuery的一大特点，就是允许使用链式写法。

$(‘div').find(‘h3′).eq(2).html(‘Hello');

采用链式写法时，jQuery自动缓存每一步的结果，因此比非链式写法要快。根据测试，链式写法比（不使用缓存的）非链式写法，大约快了25%。

7. 事件的委托处理（Event Delegation）

javascript的事件模型，采用”冒泡”模式，也就是说，子元素的事件会逐级向上”冒泡”，成为父元素的事件。

利用这一点，可以大大简化事件的绑定。比如，有一个表格（table元素），里面有100个格子（td元素），现在要求在每个格子上面绑定一个点击事件（click），请问是否需要将下面的命令执行100次？

$(“td”).bind(“click”, function(){

$(this).toggleClass(“click”);

});

回答是不需要，我们只要把这个事件绑定在table元素上面就可以了，因为td元素发生点击事件之后，这个事件会”冒泡”到父元素table上面，从而被监听到。

因此，这个事件只需要在父元素绑定1次即可，而不需要在子元素上绑定100次，从而大大提高性能。这就叫事件的”委托处理”，也就是子元素”委托”父元素处理这个事件。

具体的写法有两种。第一种是采用.delegate()方法：

$(“table”).delegate(“td”, “click”, function(){

$(this).toggleClass(“click”);

});

第二种是采用.live()方法：

$(“table”).each(function(){

$(“td”, this).live(“click”, function(){

$(this).toggleClass(“click”);

});

});

这两种写法基本等价。唯一的区别在于，.delegate()是当事件冒泡到指定的父元素时触发，.live()则是当事件冒泡到文档的根元素后触发，因此.delegate()比.live()稍快一点。此外，这两种方法相比传统的.bind()方法还有一个好处，那就是对动态插入的元素也有效，.bind()只对已经存在的DOM元素有效，对动态插入的元素无效。

根据测试，委托处理比不委托处理，快了几十倍。在委托处理的情况下，.delegate()又比.live()大约快26%。

8. 少改动DOM结构

（1）改动DOM结构开销很大，因此不要频繁使用.append()、.insertBefore()和.insetAfter()这样的方法。

如果要插入多个元素，就先把它们合并，然后再一次性插入。根据测试，合并插入比不合并插入，快了将近10倍。

（2）如果你要对一个DOM元素进行大量处理，应该先用.detach()方法，把这个元素从DOM中取出来，处理完毕以后，再重新插回文档。根据测试，使用.detach()方法比不使用时，快了60%。

（3）如果你要在DOM元素上储存数据，不要写成下面这样：

var elem = $(‘#elem');

elem.data(key,value);

而要写成：

var elem = $(‘#elem');

$.data(elem,key,value);

根据测试，后一种写法要比前一种写法，快了将近10倍。因为elem.data()方法是定义在jQuery函数的prototype对象上面的，而$.data()方法是定义jQuery函数上面的，调用的时候不从复杂的jQuery对象上调用，所以速度快得多。（此处可以参阅下面第10点。）

9. 正确处理循环

循环总是一种比较耗时的操作，如果可以使用复杂的选择器直接选中元素，就不要使用循环，去一个个辨认元素。

javascript原生循环方法for和while，要比jQuery的.each()方法快，应该优先使用原生方法。

10. 尽量少生成jQuery对象

每当你使用一次选择器（比如$(‘#id')），就会生成一个jQuery对象。jQuery对象是一个很庞大的对象，带有很多属性和方法，会占用不少资源。所以，尽量少生成jQuery对象。

举例来说，许多jQuery方法都有两个版本，一个是供jQuery对象使用的版本，另一个是供jQuery函数使用的版本。下面两个例子，都是取出一个元素的文本，使用的都是text()方法。你既可以使用针对jQuery对象的版本：

var $text = $(“#text”);

var $ts = $text.text();

也可以使用针对jQuery函数的版本：

var $text = $(“#text”);

var $ts = $.text($text);

由于后一种针对jQuery函数的版本不通过jQuery对象操作，所以相对开销较小，速度比较快。

一、选择器性能优化建议

1. 总是从#id选择器来继承

这是jQuery选择器的一条黄金法则。jQuery选择一个元素最快的方法就是用ID来选择了。

复制代码 代码如下:

$('#content').hide();

或者从ID选择器继承来选择多个元素:

复制代码 代码如下:

$('#content p').hide();

2. 在class前面使用tag

jQuery中第二快的选择器就是tag选择器（如$(‘head')），因为它和直接来自于原生的Javascript方法getElementByTagName()。所以最好总是用tag来修饰class(并且不要忘了就近的ID)

复制代码 代码如下:

var receiveNewsletter = $('#nslForm input.on');

jQuery中class选择器是最慢的，因为在IE浏览器下它会遍历所有的DOM节点。尽量避免使用class选择器。也不要用tag来修饰ID。下面的例子会遍历所有的div元素来查找id为'content'的那个节点：

复制代码 代码如下:

var content = $('div#content'); // 非常慢，不要使用

用ID来修饰ID也是画蛇添足：

复制代码 代码如下:

var traffic\_light = $('#content #traffic\_light'); // 非常慢，不要使用

3. 使用子查询

将父对象缓存起来以备将来的使用

复制代码 代码如下:

var header = $('#header');

var menu = header.find('.menu');

// 或者

var menu = $('.menu', header);

4. 优化选择器以适用Sizzle的“从右至左”模型

自版本1.3之后，jQuery采用了Sizzle库，与之前的版本在选择器引擎上的表现形式有很大的不同。它用“从左至右”的模型代替了“从右至左”的模型。确保最右的选择器具体些，而左边的选择器选择范围较宽泛些：

复制代码 代码如下:

var linkContacts = $('.contact-links div.side-wrapper');

而不要使用

复制代码 代码如下:

var linkContacts = $('a.contact-links .side-wrapper');

5. 采用find()，而不使用上下文查找

.find()函数的确快些。但是如果一个页面有许多DOM节点时，需要来回查找时，可能需要更多时间:

复制代码 代码如下:

var divs = $('.testdiv', '#pageBody'); // 2353 on Firebug 3.6

var divs = $('#pageBody').find('.testdiv'); // 2324 on Firebug 3.6 - The best time

var divs = $('#pageBody .testdiv'); // 2469 on Firebug 3.6

6. 利用强大的链式操作

采用jQuery的链式操作比缓存选择器更有效：

复制代码 代码如下:

$('li.menu-item').click(function () {alert('test click');})

.css('display', 'block')

.css('color', 'red')

fadeTo(2, 0.7);

7. 编写属于你的选择器

如果你经常在代码中使用选择器，那么扩展jQuery的$.expr[':']对象吧，编写你自己的选择器。下面的例子中，我创建了一个abovethefold选择器，用来选择不可见的元素：

复制代码 代码如下:

$.extend($.expr[':'], {

abovethefold: function(el) {

return $(el).offset().top < $(window).scrollTop() + $(window).height();

}

});

var nonVisibleElements = $('div:abovethefold'); // 选择元素

二、优化DOM操作建议

8. 缓存jQuery对象

将你经常用的元素缓存起来：

复制代码 代码如下:

var header = $('#header');

var divs = header.find('div');

var forms = header.find('form');

9. 当要进行DOM插入时，将所有元素封装成一个元素

直接的DOM操作很慢。尽可能少的去更改HTML结构。

复制代码 代码如下:

var menu = '<ul id="menu">';

for (var i = 1; i < 100; i++) {

menu += '<li>' + i + '</li>';

}

menu += '</ul>';

$('#header').prepend(menu);

// 千万不要这样做：

$('#header').prepend('<ul id="menu"></ul>');

for (var i = 1; i < 100; i++) {

$('#menu').append('<li>' + i + '</li>');

}

10. 尽管jQuery不会抛出异常，但开发者也应该检查对象

尽管jQuery不会抛出大量的异常给用户，但是开发者也不要依赖于此。jQuery通常会执行了一大堆没用的函数之后才确定一个对象是否存在。所以在对一个作一系列引用之前，应先检查一下这个对象存不存在。

11. 使用直接函数，而不要使用与与之等同的函数

为了获得更好的性能，你应该使用直接函数如$.ajax()，而不要使用$.get(),$.getJSON(),$.post()，因为后面的几个将会调用$.ajax()。

12. 缓存jQuery结果，以备后来使用

你经常会获得一个javasript应用对象——你可以用App.来保存你经常选择的对象，以备将来使用：

复制代码 代码如下:

App.hiddenDivs = $('div.hidden');

// 之后在你的应用中调用：

App.hiddenDivs.find('span');

13. 采用jQuery的内部函数data()来存储状态

不要忘了采用.data()函数来存储信息：

复制代码 代码如下:

$('#head').data('name', 'value');

// 之后在你的应用中调用:

$('#head').data('name');

14. 使用jQuery utility函数

不要忘了简单实用的jQuery的utility函数。我最喜欢的是$.isFunction(), $isArray()和$.each()。

15. 为HTML块添加“JS”的class

当jQuery载入之后，首先给HTML添加一个叫”JS”的class

复制代码 代码如下:

$('HTML').addClass('JS');

只有当用户启用JavaScript的时候，你才能添加CSS样式。例如：

复制代码 代码如下:

/\* 在css中 \*/

.JS #myDiv{display:none;}

所以当JavaScript启用的时候，你可以将整个HTML内容隐藏起来，用jQuery来实现你想实现的（譬如：收起某些面板或当用户点击它们时展开）。而当Javascript没有启用的时候，浏览器呈现所有的内容，搜索引擎爬虫也会勾去所有内容。我将来会更多的使用这个技巧。

三、关于优化事件性能的建议

16. 推迟到$(window).load

有时候采用$(window).load()比$(document).ready()更快，因为后者不等所有的DOM元素都下载完之前执行。你应该在使用它之前测试它。

17. 使用Event Delegation

当你在一个容器中有许多节点，你想对所有的节点都绑定一个事件，delegation很适合这样的应用场景。使用Delegation，我们仅需要在父级绑定事件，然后查看哪个子节点(目标节点)触发了事件。当你有一个很多数据的table的时候，你想对td节点设置事件，这就变得很方便。先获得table，然后为所有的td节点设置delegation事件：

复制代码 代码如下:

$("table").delegate("td", "hover", function(){

$(this).toggleClass("hover");

});

18. 使用ready事件的简写

如果你想压缩js插件，节约每一个字节，你应该避免使用$(document).onready()

复制代码 代码如下:

// 也不要使用

$(document).ready(function (){

// 代码

});

// 你可以如此简写：

$(function (){

// 代码

});

四、测试jQuery

19. jQuery单元测试

测试JavaSript代码最好的方法就是人来测试。但你可以使用一些自动化的工具如Selenium，Funcunit，QUit，QMock来测试你的代码（尤其是插件）。我想在另外一个专题来讨论这个话题因为实在有太多要说的了。

20. 标准化你的jQuery代码

经常标准化你的代码，看看哪个查询比较慢，然后替换它。你可以用Firebug控制台。你也可以使用jQuery的快捷函数来使测试变得更容易些：

复制代码 代码如下:

// 在Firebug控制台记录数据的快捷方式

$.l($('div'));

复制代码 代码如下:

// 获取UNIX时间戳

$.time();

复制代码 代码如下:

// 在Firebug记录执行代码时间

$.lt();

$('div');

$.lt();

复制代码 代码如下:

// 将代码块放在一个for循环中测试执行时间

$.bm("var divs = $('.testdiv', '#pageBody');"); // 2353 on Firebug 3.6

五、其他常用jQuery性能优化建议

21. 使用最新版本的jQuery

最新的版本往往是最好的。更换了版本后，不要忘记测试你的代码。有时候也不是完全向后兼容的。

22. 使用HMTL5

新的HTML5标准带来的是更轻巧的DOM结构。更轻巧的结构意味着使用jQuery需要更少的遍历，以及更优良的载入性能。所以如果可能的话请使用HTML5。

23. 如果给15个以上的元素加样式时，直接给DOM元素添加style标签

要给少数的元素加样式，最好的方法就是使用jQuey的css()函数。然而更15个以上的较多的元素添加样式时，直接给DOM添加style 标签更有效些。这个方法可以避免在代码中使用硬编码(hard code)。

复制代码 代码如下:

$('<style type="text/css"> div.class { color: red; } </style>')

.appendTo('head');

24. 避免载入多余的代码

将Javascript代码放在不同的文件中是个好的方法，仅在需要的时候载入它们。这样你不会载入不必要的代码和选择器。也便于管理代码。

25. 压缩成一个主JS文件，将下载次数保持到最少

当你已经确定了哪些文件是应该被载入的，那么将它们打包成一个文件。用一些开源的工具可以自动帮你完成，如使用Minify(和你的后端代码集成)或者使用JSCompressor，YUI Compressor 或 Dean Edwards JS packer等在线工具可以为你压缩文件。我最喜欢的是JSCompressor。

26. 需要的时候使用原生的Javasript

使用jQuery是个很棒的事情，但是不要忘了它也是Javascript的一个框架。所以你可以在jQuery代码有必要的时候也使用原生的Javascript函数，这样能获得更好的性能。

27. 从Google载入jQuery框架

当你的应用正式上线的时候，请从Google CDN载入jQuery，因为用户可以从最近的地方获取代码。这样你可以减少服务器请求，而用户如果浏览其他网站，而它也使用Google CDN的jQuery时，浏览器就会立即从缓存中调出jQuery代码。

复制代码 代码如下:

// 链接特定版本的压缩代码

<script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/jquery/1.5.0/jquery.min.js"></script>

28. 缓慢载入内容不仅能提高载入速度，也能提高SEO优化 （Lazy load content for speed and SEO benefits）

使用Ajax来载入你的网站吧，这样可以节约服务器端载入时间。你可以从一个常见的侧边栏widget开始。