jQuery 性能优化指南

本教程由零度编程收集（www.starcloud.me）

欢迎加入零度编程 QQ 群：1888845

1，总是从 ID 选择器开始继承

在 jQuery 中最快的选择器是 ID 选择器，因为它直接来自于 JavaScript 的 getElementById() 方法。
例如有一段 HTML 代码：

```
<div id="content">
<form method="post" action="#">
<h2>交通信号灯</h2>
<ul id="traffic_light">
<li><input type="radio" class="on" name="light" value="red" /> 红色</li>
<li><input type="radio" class="off" name="light" value="yellow" /> 黄色</li>
<li><input type="radio" class="off" name="light" value="green" /> 绿色</li>
</ul>
<input class="button" id="traffic_button" type="submit" value="Go" />
</form>
</div>
```

如果采用下面的选择器，那么效率是低效的。

```
var traffic_button = $("#content .button");
```

因为 button 已经有 ID 了，我们可以直接使用 ID 选择器。如下所示：

```
var traffic_button = $("#traffic_button");
```

当然 这只是对于单一的元素来讲。如果你需要选择多个元素，这必然会涉及到 DOM 遍历和循环，
为了提高性能，建议从最近的 ID 开始继承。
如下所示：

```
var traffic_lights = $("#traffic_light input");
```

2，在 class 前使用 tag(标签名)

在 jQuery 中第二快的选择器是 tag(标签)选择器(比如：$("head"))。
跟 ID 选择器累时，因为它来自原生的 getElementsByTagName() 方法。
继续看刚才那段 HTML 代码：

```
<div id="content">
<form method="post" action="#">
<h2>交通信号灯</h2>
<ul id="traffic_light">
<li><input type="radio" class="on" name="light" value="red" /> 红色</li>
<li><input type="radio" class="off" name="light" value="yellow" /> 黄色</li>
<li><input type="radio" class="off" name="light" value="green" /> 绿色</li>
</ul>
<input class="button" id="traffic_button" type="submit" value="Go" />
</form>
</div>
```

比如需要选择 红绿 单选框，
那么可以使用一个 tag name 来限制(修饰)class ，如下所示：

```
var active_light = $("input.on");
```

当然也可以结合 就近的 ID，如下所示：

```
var active_light = $("#traffic_light input.on");
```

在使用 tag 来修饰 class 的时候，我们需要注意以下几点：
（1） 不要使用 tag 来修饰 ID，如下所示：

```
var content = $("div#content");
```

这样一来，选择器会先遍历所有的 div 元素，然后匹配#content。
（好像 jQuery 从 1.3.1 开始改变了选择器核心后，不存在这个问题了。暂时无法考证。）
（2）不要画蛇添足的使用 ID 来修饰 ID，如下所示：

```
var traffic_light = $("#content #traffic_light");
```

注：如果使用属性选择器，也请尽量使用 tag 来修饰，如下所示：
`$('p[row="c3221"]').html();`而不是这样：`$('[row="c3221"]').html();`

特别提示：
tag.class 的方式 在 IE 下的性能 好于 .class 方式。
但在 Firefox 下 却低于 直接 .class 方式。
Google 浏览器下两种都差不多。
我页面上有 300 个元素，他们的性能差距都在 50 毫秒以内。

3，将 jQuery 对象缓存起来

把 jQuery 对象缓存起来 就是要告诉我们 要养成将 jQuery 对象缓存进变量的习惯。
下面是一个 jQuery 新手写的一段代码：

```
$("#traffic_light input.on").bind("click", function(){ ... });
$("#traffic_light input.on").css("border", "1px dashed yellow");
$("#traffic_light input.on").css("background-color", "orange");
$("#traffic_light input.on").fadeIn("slow");
```

但切记不要这么做。
我们应该先将对象缓存进一个变量然后再操作，如下所示：

```
var $active_light = $("#traffic_light input.on");
$active_light.bind("click", function(){ ... });
$active_light.css("border", "1px dashed yellow");
$active_light.css("background-color", "orange");
$active_light.fadeIn("slow");
```

记住，永远不要让相同的选择器在你的代码里出现多次.
注：（1）为了区分普通的 JavaScript 对象和 jQuery 对象，可以在变量首字母前加上 $ 符号。
（2）上面代码可以使用 jQuery 的链式操作加以改善。如下所示：

```
var $active_light = $("#traffic_light input.on");
$active_light.bind("click", function(){ ... })
                .css("border", "1px dashed yellow")
                .css("background-color", "orange")
                .fadeIn("slow");
```

如果你打算在其他函数中使用 jQuery 对象，那么你必须把它们缓存到全局环境中。
如下代码所示：

```
// 在全局范围定义一个对象 (例如: window 对象)
window.$my = {
head : $("head"),
traffic_light : $("#traffic_light"),
traffic_button : $("#traffic_button")
};
function do_something(){
// 现在你可以引用存储的结果并操作它们
var script = document.createElement("script");
 $my.head.append(script);
// 当你在函数内部操作是, 可以继续将查询存入全局对象中去.
$my.cool_results = $("#some_ul li");
$my.other_results = $("#some_table td");
 // 将全局函数作为一个普通的 jquery 对象去使用.
$my.other_results.css("border-color", "red");
 $my.traffic_light.css("border-color", "green");
}
```

```
//你也可以在其他函数中 使用它
```

4，对直接的 DOM 操作进行限制

这里的基本思想是在内存中建立你确实想要的东西，然后更新 DOM 。
这并不是一个 jQuery 最佳实践，但必须进行有效的 JavaScript 操作 。直接的 DOM 操作速度很慢。
例如，你想动态的创建一组列表元素，千万不要这样做,如下所示：

```
var top_100_list = [...], // 假设这里是 100 个独一无二的字符串
$mylist = $("#mylist"); // jQuery 选择到 <ul> 元素
for (var i=0, l=top_100_list.length; i<l; i++){
  $mylist.append("<li>" + top_100_list[i] + "</li>");
}
```

我们应该将整套元素字符串在插入进 dom 中之前先全部创建好，如下所示：

```
var top_100_list = [...],$mylist = $("#mylist"), top_100_li = ""; // 这个变量将用来存储我们的列表元素
for (var i=0, l=top_100_list.length; i<l; i++){
   top_100_li += "<li>" + top_100_list[i] + "</li>";
}
$mylist.html(top_100_li);
```

注：记得以前还看过一朋友写过这样的代码：

```
for (i = 0; i < 1000; i++) {
    var $myList = $('#myList');
    $myList.append('This is list item ' + i);
}
```

呵呵，你应该已经看出问题所在了。既然把#mylist 循环获取了 1000 次！！！

5，冒泡

除非在特殊情况下，否则每一个 js 事件(例如:click, mouseover 等.)都会冒泡到父级节点。
当我们需要给多个元素调用同个函数时这点会很有用。
代替这种效率很差的多元素事件监听的方法就是，你只需向它们的父节点绑定一次。
比如，我们要为一个拥有很多输入框的表单绑定这样的行为：当输入框被选中时为它添加一个 class
传统的做法是，直接选中 input，然后绑定 focus 等，如下所示：

```
$("#entryform input").bind("focus", function(){
   $(this).addClass("selected");
}).bind("blur", function(){
   $(this).removeClass("selected");
});
```

当然上面代码能帮我们完成相应的任务，但如果你要寻求更高效的方法，请使用如下代码：

```
$("#entryform").bind("focus", function(e){
    var $cell = $(e.target); // e.target 捕捉到触发的目标元素
    $cell.addClass("selected");
}).bind("blur", function(e){
    var $cell = $(e.target);
    $cell.removeClass("selected");
});
```

通过在父级监听获取焦点和失去焦点的事件，对目标元素进行操作。
在上面代码中，父级元素扮演了一个调度员的角色，它可以基于目标元素绑定事件。
如果你发现你给很多元素绑定了同一个事件监听，那么现在的你肯定知道哪里做错了。

同理，在 Table 操作时，我们也可以使用这种方式加以改进代码：
普通的方式：

```
$('#myTable td').click(function(){
    $(this).css('background', 'red');
});
```

改进方式：

```
$('#myTable').click(function(e) {
     var $clicked = $(e.target);
     $clicked.css('background', 'red');
});
```

假设有 100 个 td，在使用普通的方式的时候，你绑定了 100 个事件。
在改进方式中，你只为一个元素绑定了 1 个事件，
至于是 100 个事件的效率高，还是 1 个事件的效率高，相信你也能自行分辨了。

6，推迟到 $(window).load

jQuery 对于开发者来说有一个很诱人的东西，可以把任何东西挂到$(document).ready 下。
尽管$(document).ready 确实很有用，它可以在页面渲染时，其它元素还没下载完成就执行。
如果你发现你的页面一直是载入中的状态，很有可能就是$(document).ready 函数引起的。
你可以通过将 jQuery 函数绑定到$(window).load 事件的方法来减少页面载入时的 cpu 使用率。
它会在所有的 html(包括<iframe>)被下载完成后执行。

```
$(window).load(function(){
    // 页面完全载入后才初始化的 jQuery 函数.
});
```

一些特效的功能，例如拖放，视觉特效和动画，预载入隐藏图像等等，都是适合这种技术的场合。

7，压缩 JavaScript

压缩和最小化你的 JavaScript 文件。

在线压缩地址: http://dean.edwards.name/packer/
压缩之前，请保证你的代码的规范性，否则可能失败，导致 Js 错误。

8，尽量使用 ID 代替 Class。

前面性能优化已经说过，ID 选择器的速度是最快的。所以在 HTML 代码中，能使用 ID 的尽量使用 ID 来代替 class。
看下面的一个例子：

```
// 创建一个 list
var $myList = $('#myList');
var myListItems = '<ul>';
for (i = 0; i < 1000; i++) {
    myListItems += '<li class="listItem' + i + '">This is a list item</li>'; //这里使用的是 class
 }
myListItems += '</ul>';
$myList.html(myListItems);
// 选择每一个 li
 for (i = 0; i < 1000; i++) {
   var selectedItem = $('.listItem' + i);
}
```

在代码最后，选择每个 li 的过程中，总共用了 5066 毫秒，超过 5 秒了。
接着我们做一个对比，用 ID 代替 class：

```
// 创建一个 list
var $myList = $('#myList');
var myListItems = '<ul>';
for (i = 0; i < 1000; i++) {
   myListItems += '<li id="listItem' + i + '">This is a list item</li>'; //这里使用的是 id
}
myListItems += '</ul>';
$myList.html(myListItems);
 // 选择每一个 li
for (i = 0; i < 1000; i++) {
```

```
    var selectedItem = $('#listItem' + i);
}
```

在上段代码中，选择每个 li 总共只用了 61 毫秒，相比 class 的方式，将近快了 100 倍。

9，给选择器一个上下文

jQuery 选择器中有一个这样的选择器，它能指定上下文。

```
jQuery( expression, context );
```

通过它，能缩小选择器在 DOM 中搜索的范围，达到节省时间，提高效率。

普通方式：

```
$('.myDiv')
```

改进方式：

```
$('.myDiv' , $("#listItem") )
```

10，慎用 .live()方法（应该说尽量不要使用）

这是 jQuery1.3.1 版本之后增加的方法，这个方法的功能就是为 新增的 DOM 元素 动态绑定事件。

但对于效率来说，这个方法比较占用资源。所以请尽量不要使用它。

例如有这么一段代码:

```
<script type="text/javascript" >
$(function(){
 $("p").click(function(){
    alert( $(this).text() );
 });
$("button").click(function(){
   $("<p>this is second p</p>").appendTo("body");
});
}) </script>
<body>
<p>this is first p</p> <button>add</button>
</body>
```

运行后，你会发现 新增 的 p 元素，并没用被绑定 click 事件。

你可以改成.live("click")方式解决此问题，代码如下：

```
$(function(){
$("p").live("click",function(){ //改成 live 方式
    alert( $(this).text() );
```

```
 });
$("button").click(function(){ $("<p>this is second
p</p>").appendTo("body"); });})
```

但我并不建议大家这么做，我想用另一种方式去解决这个问题，代码如下：

```
$(function(){
$("p").click(function(){
   alert( $(this).text() );
});
$("button").click(function(){
   $("<p>this is second p</p>").click(function(){  //为新增的元素重新绑定一次
         alert( $(this).text() );
   }).appendTo("body");
});
})
```

虽然我把绑定事件重新写了一次，代码多了点，但这种方式的效率明显高于 live()方式，特别是在频繁的 DOM 操作中，这点非常明显。

11，子选择器和后代选择器

后代选择器经常用到，比如：$("#list p");
后代选择器获取的是元素内部所有元素。
而有时候实际只要获取 子元素，那么就不应该使用后代选择器。
应该使用子选择器，代码如下：

```
$("#list > p");
```

12，使用 data()方法存储临时变量

下面是一段非常简单的代码，

```
$(function(){
   var flag = false;
   $("button").click(function(){
      if(flag){
         $("p").text("true");
         flag=false;
      }else{
         $("p").text("false");
         flag=true;
      }
   });
})
```

改用 data()方式后，代码如下：

```
$(function(){
    $("button").click(function(){
        if( $("p").data("flag") ){
            $("p").text("true");
            $("p").data("flag",false);
        }else{
             $("p").text("false");
             $("p").data("flag",true);
        }
    });
})
```

jQuery 性能优化指南到此结束。