WEB 前端开发规范

1. HTML 部分

1.1. 遵循 xhtml 1.0 规则：

1) 所有标签必须结束
2) 所有标签必须小写
3) 标签属性必须使用成对引号（单引号或双引号）
4) 标签属性必须有值：

```
<select>
    <option selected="selected"></option>
</select>
<input type="checkbox" checked="checked" />
```

5) 所有特殊符号必须转义（&、<、>、©、»...）
6) 文档类型声明及编码：统一为 html5 声明类型<!DOCTYPE html>；编码统一为<meta charset="utf-8" />，书写时利用 IDE 实现层次分明的缩进;
7) 2. 非特殊情况下样式文件必须外链至<head>...</head>之间;非特殊情况下 JavaScript 文件必须外链至页面底部;

1.2. 标签属性命名规范

id： 连接符命名法“hello-world”

class: 连接符命名法“hello-world”

name： 驼峰式命名法“helloWorld”

1) 用于结构布局的元素 id 命名：

主容器：	main
页头：	header
页脚：	footer
内容区域：	content
LOGO：	logo
主导航：	main-nav
二级导航：	sub-nav

2) name 命名：

一般用于表单元素，根据当前元素 id 属性值命名，去掉 id 属性值的前缀和所有连接符，使用驼峰式命名法命名，例如 id=”id-card”，那么 name=”idCard”。

1.3. 合理使用标签，语义化结构

1) 标签合理嵌套

1. span、strong、em、p、h1~h6 等行级元素不能包含块级元素：div、ul、ol、dl、p

尽可能减少 div 嵌套，如 欢迎访问 XXX，您的用 户名是

用户名 完全可以用以下代码替代: 欢迎 访问 XXX，您的用户名是用户名 ;

2) 严禁多 div 症、多 span 症、多 table 症，正确使用标签表现 DOM 结构，在文档去除 css 的情况下，任然具有结构和可读性，以下是 html 标记的使用规范：

p：文本段落；

dl：定义列表，可包括标题和内容简介的列表；

ul：无序列表；

ol：有序列表；

h1~h6：文章标题、内容区块标题，根据重要性由大到小区分，h1 一个页面只出现一次；

strong/em：强调文本；

img：图像，必须加上 alt 属性，当图像无法显示时，可表示图像信息，背景和按钮使用 css 处理，不使用 img 元素；

table：数据网格，规则的分栏布局，尽可能显性的定宽和定高。

3) 充分利用无兼容性问题的 html 自身标签，比如 span, em, strong, optgroup, label,等等；需要为 html 元素添加自定义属性的时候，首先 要考虑下有没有默认的已有的合适标签去设置，如果没有，可以使用须以"data-"为前缀来添加自定义属性，避免使用"data:"等其他命名方式;

4) 书写链接地址时，必须避免重定向，例如：href="http://itaolun.com/"，即须在 URL 地址后面加上“/”；

5) 合理化表单结构

a) 使用 fieldset 元素包裹相同类别的字段；

b) 使用 legend 元素表示字段类别名称；

c) 使用 label 表示字段文本，添加必要的 for 属性，以在点击字段文本时，文本框能获得焦点；

d) 文本框不使用 size 属性定义宽度，而使用 css 的 width 属性；

e) 添加 maxlength 属性限制输入字符的长度。（<input type="password" id="password1" maxLength="8" /> 允许输入密码 8 位数）

2. CSS 部分

2.1. Css 命名规则

1) 样式类名全部用小写，首字符必须是字母，禁止数字或其他特殊字符。由以字母开头的小写字母（a-z）、数字（0-9）、下划线（-）组成;

2) 可以是单个单词，也可以是组合单词，要求能够描述清楚模块和元素的含义，使其具有语义化。避免使用 123456…red,blue,left,right 之类的（如颜色、字号大小等）矢量命名，如 class=”left-news”、class=”2” ，以避免当状态改变时名称失去意义;

3) 尽量用单个单词简单描述 class 名称;

4) 双单词或多单词组合方式：形容词-名词、命名空间-名次、命名空间-形容词-名词。例如：news-list、mod-feeds、mod-my-feeds、cell-title;

5) css 文件命名: 英文命名，后缀.css. 共用 wichung.css, 首页 index.css, 其他页面依实际模块需求命名;

6)

2.2. 命名空间

在编码思想上，我们可以将页面拆分成不同的层级（布局、模块、元件）。

什么是 CSS 命名空间？

通过统一的命名规范定义命名的范围，成为 CSS　class & id 命名空间。

布局：以语义化的单词 layout 作为命名空间，例如主栏布局命名 layout-main，

只改变 layout-命名空间后面的命名，layout 始终保留。布局的命名空间为 layout-xxx。

模块：页面是由一个或多个模块组成，模块的英文单词是 module，规范简写成 mod，如新闻模块 mod-news，照片展示模块 mod-photo-show。模块的命名空间为 mod-xxx 。

元件：元件是属于模块内部的，也可以说模块是由元件和它内部的自有元素组成。如用户照片信息元件 cell-user-photo。元件的命名空间为 cell-xxx 。

2.3. 添加文档样式

1) 引用外部文件方式添加样式
2) 严禁编写标记内代码，比如 就应该写成 ，然后在样式表中去完成样式代码编写。
3) 严禁在文档中使用<style type="text/css"></style>代码块。
4) 如果是发布版本，请压缩合并代码，将多个样式文件合并为单个文件，在线 css 代码压缩工具：http://www.csscompressor.com/

2.4. 属性简写

为了节省字节数及文件大小，以下属性请使用简写方式：

padding:	top right bottom left;
margin:	top right bottom left;
border:	style width color;
border-top:	style width color;
border-right:	style width color;
border-bottom:	style width color;
border-left:	style width color;
border-color:	top right bottom left;
border-style:	top right bottom left;
border-width:	top right bottom left;
background:	color url(image) repeat position;
list-style:	type position url(image);
font-weight:	400 / 700;

2.5. 字体：

1) 全局定义字体：body{font: 12px arial, helvetica, sans-serif; line-height: 1.5;}
2) font-family:
   a) 等宽字体组合：Arial, Helvetica, sans-serif;
   b) 不等宽（宽扁）字体组合： Verdana, Trebuchet MS, sans-serif;
   c) 中文字体：除非内容文本需要，不推荐强制定义

2.6. 编写兼容的 CSS 代码

1) 页面必须在 ie6~8、firefox、opera、safari、chrome 下显示兼容；

2) 尽量不使用 IE 有条件注释方式，对某一版本浏览器编写额外的样式表。针对某一版本浏览器编写额外的样式表，会对维护和管理照成困难。如出现有显示不兼容现象应首先考虑代码是否有问题（如属性是否对当前浏览器支持），必须编写与常用浏览器都兼容的代码；

不推荐的引用方式：

```
<!--[if IE 6]>
    <link type="text/css" rel="stylesheet" href="ie6.css" />
<![endif]-->
<!--[if lte ie 7]>
    <link type="text/css" rel="stylesheet" href="ie7.css" />
<![endif]-->
```

3) 尽量不要使用!important 或下划线等招数编写代码。

2.7. CSS 通用命名

(1)页面框架命名，一般具有唯一性，推荐用 ID 命名

ID 名称	命名	ID 名称	命名
头部	header	主体	main
脚部	footer	容器	wrapper/container
侧栏	side-bar	栏目	column
布局	layout		

(2)模块结构命名

CLASS 名称	命名	CLASS 名称	命名
模块(如：新闻模块)	mod (mod-news)	标题栏	title
内容	content	次级内容	sub-content

(2)导航结构命名

CLASS 名称	命名	CLASS 名称	命名
导航	nav	主导航	main-nav
子导航	sub-nav	顶部导航	top-nav
菜单	menu	子菜单	sub-menu

(3)一般元素命名

CLASS 名称	命名	CLASS 名称	命名
二级	sub	面包屑	breadcrumb
标志	logo	广告	Bner

			(禁用 banner 或 ad)
登陆	login	注册	regsiter/reg
搜索	search	加入	join
状态	status	按钮	btn
滚动	scroll	标签页	tab
文章列表	list	短消息	msg/message
当前的	current	提示小技巧	tips
图标	icon	注释	note
指南	guide	服务	service
热点	hot	新闻	news
下载	download	投票	vote
合作伙伴	partner	友情链接	link
版权	copyright	演示	demo
下拉框	select	摘要	summary
翻页	pages	主题风格	themes
设置	set	成功	suc
按钮	btn	文本	txt
颜色	color/c	背景	bg
边框	border/bor	居中	center
上	top/t	下	bottom/b
左	left/l	右	right/r
添加	add	删除	del
间隔	sp	段落	p
弹出层	pop	公共	global/gb
操作	op	密码	pwd
透明	tran	信息	info
重点	hit	预览	pvw
单行输入框	input	首页	index
日志	blog	相册	photo
留言板	guestbook	用户	user
确认	confirm	取消	cancel
报错	error		

3. 图片部分

1) 背景图片：bg001,bg002,bg003……
2) 一般图片：img001,img002,img003……
3) 特定图片：如 banner,logo 按照具体情况命名
4) 按钮图片：btn-submit,btn-cancel…….

5) 运用 css sprite 技术集中小的背景图或图标，减小页面 http 请求，但注意，请务必在对应的 sprite psd 源图中划参考线，并保存至 images 目录下
6) 图片格式仅限于 gif || png || jpg
7) 在保证视觉效果的情况下选择最小的图片格式与图片质量，以减少加载时间

4. JavaScript 部分

4.1. Js 命名规范：

1) Js 文件命名：英文命名，后缀.js. 共用 wichung-common.js，其他依实际模块需求命名

4.2. 在文档中引用 js：

1) 使用外部文件方式引用 js；
2) 将引用 js 的代码集中放置在一起，可放置在</head>之前或</body>之前，严禁在 body 间分散放置；
3) 使 DOM 结构和 js 代码分离，禁止写在标记内部；
4) 如果是发布版本，请将多个稳定版本的 js 文件压缩、归类放置到单个文件内，**压缩和最小化 js 文件**，在线 packer 压缩工具：http://dean.edwards.name/packer/；
5) 引入 JS 库文件，文件名须包含库名称及版本号及是否为压缩版，比如 jquery-1.4.1.min.js；引入插件，文件名格式为库名称+插件名称，比 如 jQuery.cookie.js;

4.3. 优化 jQuery 代码，提高性能：

1. 原则上仅引入 jQuery 库，若需引入第三方库，须与团队其他人员讨论决定；
2. jQuery 变量命名：$name，普通变量命名：name；
3. 选择器从最近的 ID 开始继承或直接使用 ID 选择器：$(”#id tag”)；
4. 选择器在使用 class 前加上标签名：$(”span.span1”)；
5. 尽量使用 ID 选择器代替 class；
6. 要获取子元素请使用子选择器，而不要使用后代选择器：$(”#id > span”)；
7. 缓存 jQuery 对象，不要在代码中重复出现相同的选择器：var btn=$(”#id”)；
8. 使用 data()存储临时变量；
9. 避免使用 live()方法绑定事件；
10. 在父级元素监听事件，对目标元素进行操作：
    $(”#id”).click(function(e){var input=$(e.target);});
11. 推迟加载拖放、动画、视觉特效等代码，把可能会影响页面加载速度的代码绑定到$(window).load()事件中。

下面是一些关于 jQuery 优化方面的建议：

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);
        }
    });
})
```

13，尽量使用原生的 JavaScript 方法

看下面一段代码，它用来判断多选框是否被选中：

```
$(document).ready(function(){
    var $cr = $("#cr");   //jQuery 对象
     $cr.click(function(){
          if($cr.is(":checked")){ //jQuery 方式判断
             alert("感谢你的支持!你可以继续操作！");
          }
     })
});
```

上面代码中，判断是否选中是用了 jquery 的方法，但这里可以直接使用原生的 JavaScript 方法，看下面代码：

```
$(document).ready(function(){
```

```
    var $cr = $("#cr");    //jQuery 对象
    var cr = $cr.get(0); //DOM 对象，获取  $cr[0]
        $cr.click(function(){
            if(cr.checked){ //原生的 JavaScript 方式判断
                alert("感谢你的支持!你可以继续操作!");
            }
        })
});
```

毋庸置疑，第二种方式效率高于第一种方式，因为他不需要拐弯抹角的去调用许多函数。

更多：

```
$(this).css("color","red");     .....
```

改成：

```
this.style.color ="red"     .....
```

```
$("<p></p>")
```

改成:

```
$( document.createElement("p") )
```

有时候你也许根本不需要 jQuery。（ 如果不涉及兼容性问题和自己的水平问题。）

14，使用 for 代替 each()方法

这个跟 13 有点类似，但我特意提取出来，让大家注意下。

```
var array = new Array ();
for (var i=0; i<10000; i++) {
	array[i] = 0;
}

$.each (array, function (i) {
	array[i] = i;
});
```

15，使用 join()来拼接字符串。

也许你之前一直使用 " + " 来 拼接长字符串。现在你可以改改了。虽然它可能会有点奇怪，但它确实有助于优化性能，尤其是长字符串处理的时候。

首先创建一个数组，然后循环，最后使用 join()把数组转化为字符串。

```
	var array = [];
	for (var i=0; i<=10000; i++) {
		array[i] = '<li>'+i+'</li>';
	}
	$('#list').html (array.join (''));
```

16，Ajax 中文乱码。

Ajax 传递中文时会乱码，传统方式大家都是利用 escape 去编码，然后去后台解码。

```
$.ajax({
    type: "POST",
    url: "AjaxTest.jsp",
    data: "txt="+$('#txt').val(),
    success: function(msg){
       $("#AjaxResponse").text(msg);
    }
});
```

现在你可以换一换了，你可以非常简单的解决这个问题，只需要添加一参数即可：

```
 $.ajax({
    type: "POST",
```

```
    url: "AjaxTest.jsp",
    data: "txt="+$('#tbox1').val(),
    success: function(msg){
       $("#AjaxResponse").text(msg);
    },
    contentType:"application/x-www-form-urlencoded;charset=utf-8"
});
```