**Javascript调试**

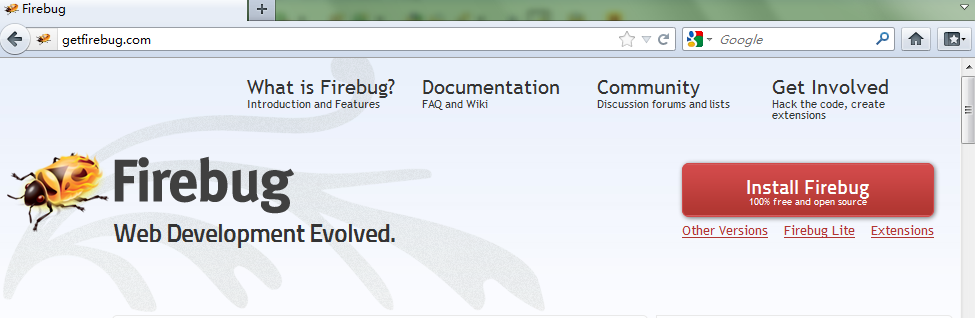
目前主流的浏览器有ie、firefox、chrome、safari、opera、maxthon等，虽然这几款浏览器对js都提供了调试功能，但本人觉得还是firefox中的firebug功能比较强大，而且基于firebug的扩展插件也挺多，下面主要介绍firebug的使用和一些基于firebug的插件使用。Firefox版本为：11，Firebug版本为：1.9.1。

这里需要说明的是，对于ajax加载的js，在firebug中不能打断点调试，但在IE下可以调试。

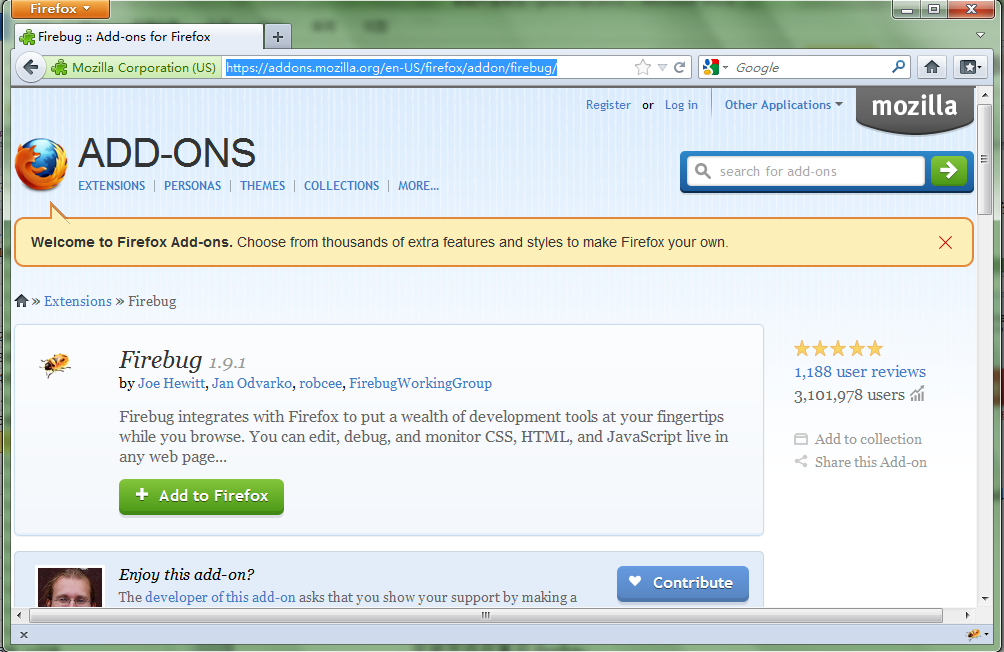


## 安装Firebug

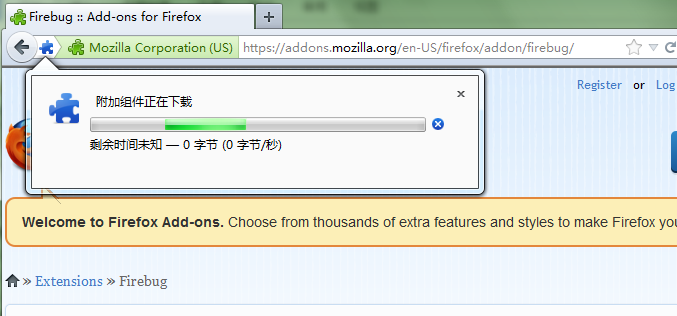
首先到官网下载最新版的firebug插件，官网地址：<http://getfirebug.com/。打开firefox>浏览器，输入<http://getfirebug.com/>，如下图



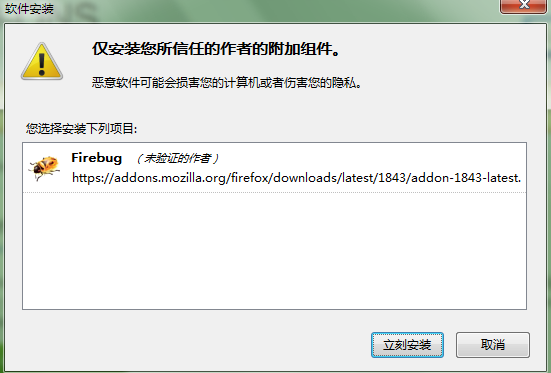
点击Install Firebug进入插件下载页面，选择合适的firebug版本，点击download后进入下载页面，如图



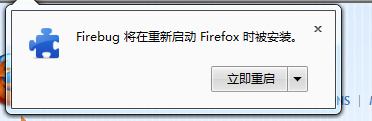
点击 Add to Firefox后开始下载，如图



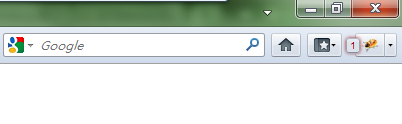
下载完成后提示安装信息，如图



点击立刻安装按钮开始安装，安装完成后提示是否重启firefox，如图



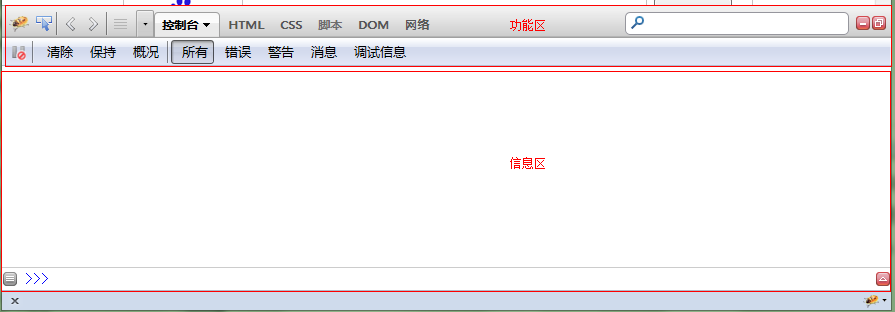
点击立即重启按钮重启firefox，重启后可以看到firebug差价已安装完成，如图



用户可以定制firebug显示位置，右击菜单栏选择定制，打开定制窗口后，根据需要可以把firebug拖放到要显示的位置，一般拖放到后下角

## 打开和关闭Firebug

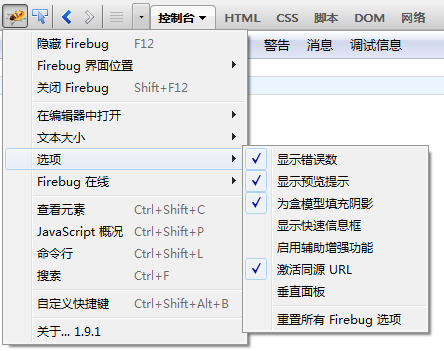
单击Firebug的图标或者按F12键你会发现页面窗口被分成了两部分，上半部分是浏览的页面，下半部分则是Firebug的控制窗口，如图。如果你不喜欢这样，可以按CTRL+F12或在前面操作后单击右上角的最大化按钮，弹出一个新窗口作为Firebug的控制窗口。



从上图可以看到firebug共有控制台、HTML、CSS、脚本、DOM和网络六个标签页，开发人员可以查看不同的内容。整个firebug窗口主要有两个区域，一个是功能区，一个是信息区。选择功能区第二行的不同标签，信息区的显示会有不同，搜索框的搜索方式也会不同。点击信息区右下角的图标可以把该区分成左右两部分，方便查看控制台输入的运行脚本，如图

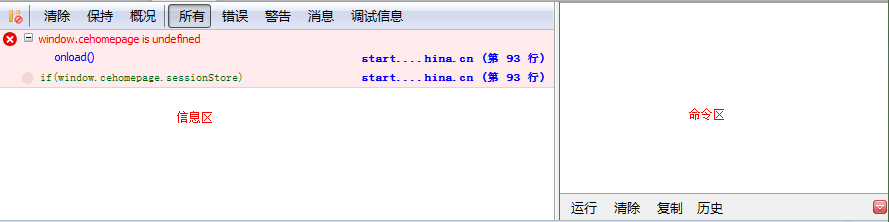


开发人员可以在右侧输入测试代码，点击运行按钮来测试javascript片段。点击firebug窗口左上角第一个图标时会打开firebug主菜单，主要是一些关于firebug的操作功能和选项。如图



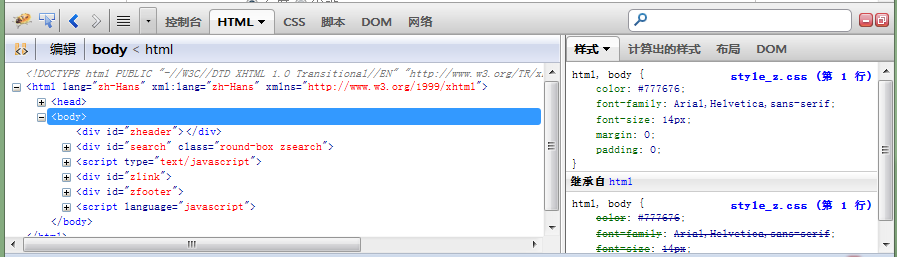
## 控制台（console）

单击功能区第二栏的“控制台”标签可切换到控制台。控制台的作用是显示各种信息（可在下拉菜单中定义显示的类型），显示脚本代码中内嵌的控制台调试信息，通过命令行对脚本进行调试。 控制台分两个区域，一个是信息区，一个是命令区。如图



## HTML

单击功能区第二栏的“HTML”标签可切换到源代码查看功能。虽然Firefox也提供了查看页面源代码的功能，但它显示的只是页面文件本身的源代码，通过脚本输出的HTML源码是看不到。而Firebug则是所见即所得，是最终解析后的源代码。

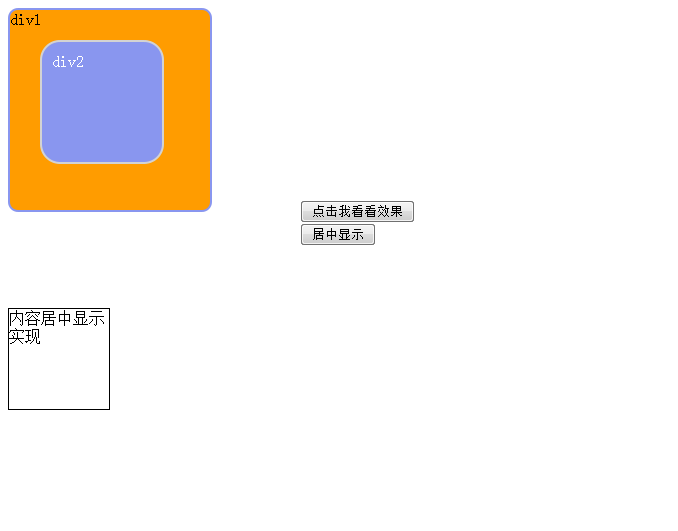


从上图中我们可以看到，信息区被分成了两个部分，左边是显示源代码，右边是一个功能区，可以从这里查看到HTML Element中的CSS定义、计算出的样式、布局情况和DOM结构。我们还可以看到，源代码是按DOM结构分层次显示的，通过层次折叠功能，我们就可以很方便分析代码。在功能区的第一行还根据你的选择，清晰的按子、父、根列出了当前源代码的层次，单击各部分，则会即刻转到该部分的源代码。下面以一个例子来详细讲解firebug的用法。

例子源代码

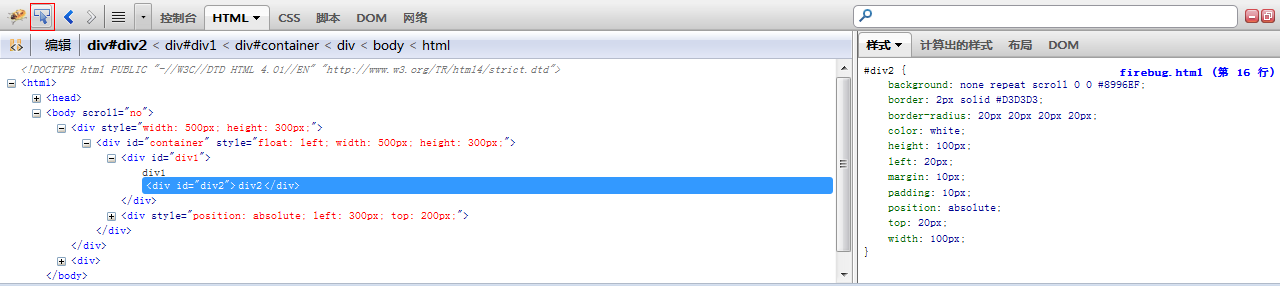
|  |
| --- |
| <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">  <html>  <head>  <title>firebug 简单例子</title>  <meta http-equiv=*"content-type"* content=*"text/html; charset=utf-8"*>  <style>  *#div1* {  background: *#FF9C00*;  width: *200px*;  height: *200px*;  position: *absolute*;  border: *2px solid #8996EF*;  border-radius: *10px 10px 10px 10px*;  }  *#div2* {  background: *#8996EF*;  width: *100px*;  height: *100px*;  margin: *10px*;  padding: *10px*;  border: *2px solid #d3d3d3*;  border-radius: *20px 20px 20px 20px*;  color: *white*;  position: *absolute*;  top: *20px*;  left: *20px*;  }  *#div3* {  background: *#EF9CF0*;  border: *1px solid #8996EF*;  width: *200px*;  height: *100px*;  margin-left: *25px*;  float: *right*;  text-align: *center*;  vertical-align: *middle*;  }  *#container2* {  position: *relative*;  height: *100px*;  width: *100px*;  border: *1px solid #000*;  }  *.content1* {  position: *relative*;  width: *250px*;  }  </style>  <script type=*"text/javascript"*>  **function** addDiv() {  **var** div = document.createElement('div');  div.id = 'div3';  div.innerHTML = '新加入的div';  document.getElementById('container').appendChild(div);  }  **function** centerMiddle() {  **var** container = document.getElementById("container2");  **var** content = document.getElementById("content1");  **var** contentOffsetHeight = content.offsetHeight;  **var** contentOffsetWidth = content.offsetWidth;  **var** containerClientHeight = container.clientHeight;  **var** containerClientWidth = container.clientWidth;  content.style.left = (containerClientWidth - contentOffsetWidth) / 2  + "px";  content.style.top = (containerClientHeight - contentOffsetHeight) / 2  + "px";  }  </script>  </head>  <body scroll=*"no"*>  <div style="width: *500px*; height: *300px*;">  <div id=*"container"* style="float: *left*; width: *500px*; height: *300px*;">  <div id=*"div1"*>  div1  <div id=*"div2"*>div2</div>  </div>  <div style="position: *absolute*; left: *300px*; top: *200px*;">  <input type=*"button"* value=*"点击我看看效果"* onclick="addDiv()"> <br>  <input type=*"button"* value=*"居中显示"* onclick="centerMiddle()">  </div>  </div>  </div>  <div>  <div id=*"container2"*>  <span id=*"content1"* class=*"content1"*>内容居中显示实现</span>  </div>  </div>  </body>  </html> |

运行效果为



HTML中的操作有以下几点

* 打开firebug，切换到HTML标签上，可以用左侧第二个图标来定位页面中的DOM节点，按下该按钮后，把鼠标移动到要查看的DOM节点，然后点击鼠标，HTML控制台就可以显示要查看的DOM节点html代码了（会高亮显示该段代码），如图所示：



对应页面中DOM元素为div2。

* 在HTML标签中选中某段代码，点击【编辑】按钮，可以编辑该段代码，如果要放弃修改，可以按ESC键退出。因为是所见即所得的，所以我们在修改的时候，页面会同时刷新显示。
* 选中DOM节点对应的源代码，右击鼠标，弹出的菜单列举了许多功能，用户可以根据需要选择操作，来完成调试功能，比如删除HTML、记录事件、新建属性（该功能可以直接在html中添加相关属性）等，如图



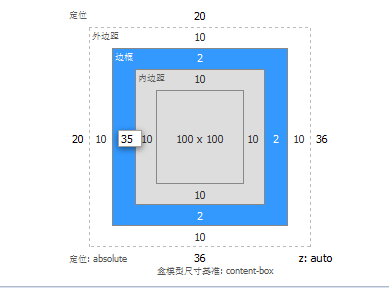
* 选中DOM节点对应的源代码后，调试窗口后侧会显示样式、计算出的样式、布局、DOM，如图



其中“样式”中显示了该dom节点所有设置的样式，包括style和class的设置，用户可以双击样式节点来动态的编辑样式，firebug提供了智能提示功能，即使你记不清所有的样式，也能通过智能提示来完成修改。我们可以对CSS定义进行添加、编辑、删除、禁止等操作。我们还可以通过右击鼠标来完成这些功能。样式中还可以根据需要过滤显示样式，用户只需点击样式右侧的下拉箭头来完成操作。这些修改都是所见即所得，会同步刷新页面的。

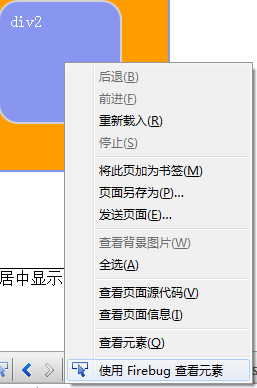
“计算出的样式”列举了该DOM节点所有style（包括默认的）。

“布局”中显示了盒子模型。页面设计中，我们有时候最头疼的是什么？是盒子模型！为了调整一个Element的margin、border、padding和相对位置，我们往往需要花很多工夫去修改源代码，然后刷新页面查看效果。有了Firebug，你就可以轻松应对了。我们将右边的区域切换到“布局”标签，你会看到一个盒子模型，里面从外到里通过不同的线和颜色划分出了offset、margin、border、padding和内容五个区域，里面4个区域在每个边上都有1个数值，表示该方向上的调整值。用户可以动态的修改值来调整DOM元素显示效果，如图

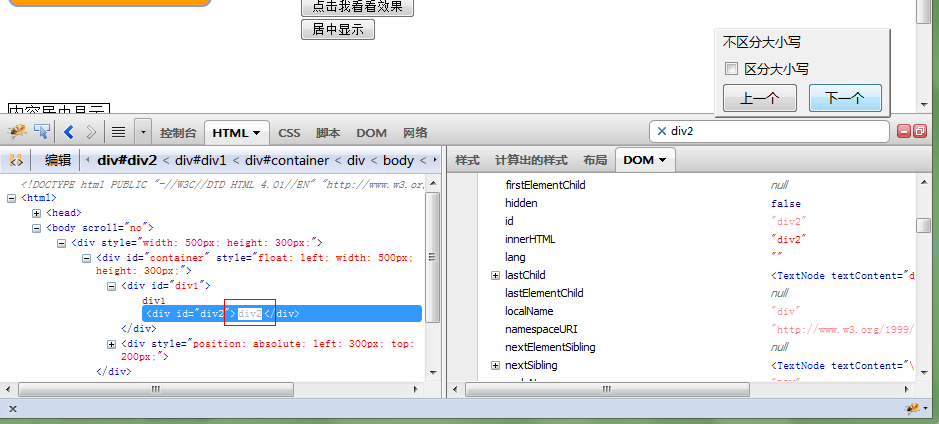


“DOM”中显示了该DOM节点对应的属性、方法、函数、常量等DHTML。

* 在页面中右击鼠标，选择【使用Firebug查看元素】，也可以定位到html源代码，如图所示



* 如果页面中的元素很多，对应的源代码也很多，或者有些区域在页面中不方便鼠标指定时怎么办？没关系，我们还有一个厉害武器，搜索功能。输入要查询的内容，可以是任意内容、样式、文本等，如图

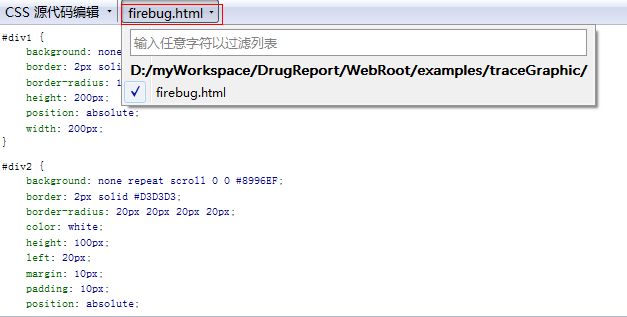


譬如我们要查找“div2”，但在层层叠叠的源代码中不好找，在页面中鼠标也很难找到，那我们就在功能区的搜索框中输入“div2”，再看看源代码区域，“div2”被加亮显示出来了。还可以按照是否区分大小写来查找。

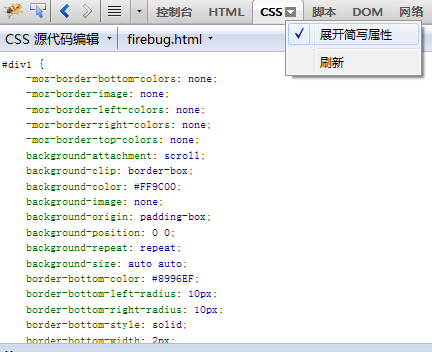
上面就是HTML标签中的基本操作，**这里要提醒大家一下，在HTML里调试出正确的源代码和CSS后，别忘记将源代码和CSS的修改结果复制到你的源代码文件中，不然你的调试结果在页面刷新后会付之东流。切记！切记！**

## CSS

将功能区第二行的标签切换到“CSS”，在这里我们可以查看页面中所有的CSS定义，包括链接的CSS文件。通过功能区的文件选择按钮可以选择不同的含有CSS的文件（图中红色圈住部分）。



用户可以对CSS进行添加、编辑、删除、禁止等操作，跟HTML标签中的CSS操作类似，这里就不多讲了。需要注意的是可以通过“展开简写属性”来详细查看每个CSS的定义。



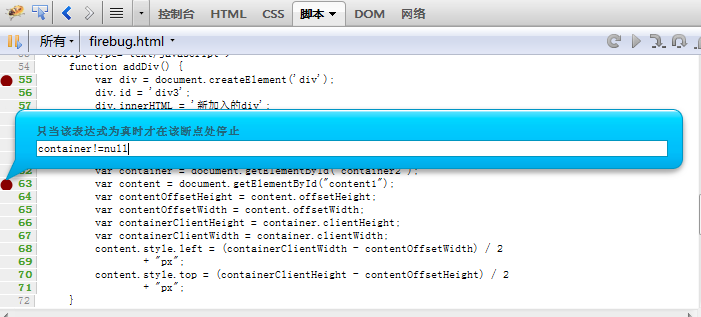
## 脚本

将功能区第二行的标签切换到“脚本”，在这里我们可以对页面中所有的脚本进行调试，包括链接的脚本。和CSS一样，可以通过文件选择按钮选择不同的脚本文件，当然也可以过滤要显示的脚本信息，如图



当然，要想定位脚本位置，用查询功能最方便了，参见HTML标签中讲解的查询功能。

如果要在脚本中设置一个断点，可以单击行号旁边的空白区域，这时会出现一个红色的点表示在这里设置了断点，如图。当脚本运行到此会停止运行，等待你的操作。在右边的小窗口将标签切换到“断点（Breakpoints）”可以查看我们设置的所有断点，单击左上角的checkbox可以禁用该断点，如果要删除断点可以单击右上角的删除图标。通过“Options”菜单的“禁用所有断点（Disable All Breakpoints）”可暂时禁用所有断点，而“删除所有断点（Remove All Breakpoints）”可删除所有断点。在断点标记的红点上单击右键还可以设置断点条件，在符合条件的时候才会停止脚本的执行。





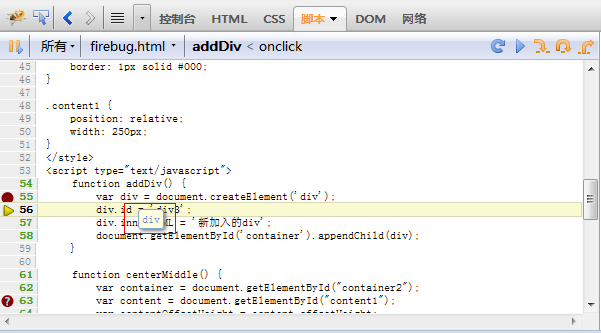
下面讲解一下调试功能的使用，首先在测试页面中设置断点，如在方法addDiv中第一行设置断点，该方法的功能是在页面中动态添加一个div元素

|  |
| --- |
| **function** addDiv() {  **var** div = document.createElement('div');  div.id = 'div3';  div.innerHTML = '新加入的div';  document.getElementById('container').appendChild(div);  } |

然后在页面中调用该方法

|  |
| --- |
| <input type=*"button"* value=*"点击我看看效果"* onclick="addDiv()"> |

接下来在页面中点击该按钮，firebug控制台会显示如下图的调试界面

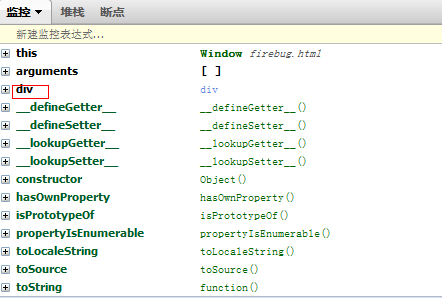


图中右上角五个按钮从左往右的功能依次为：

* 重新运行
* 继续（F8）会跳到下一个断点，没有则结束调试
* 单步进入（F11）
* 单步跳过（F10）
* 单步退出（Shift + F11）

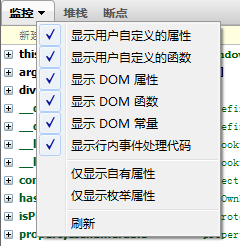
用户通过点击这些按钮完成代码的跟踪调试。

当鼠标移动到某个变量上，如“div”变量，这时会出现一个小方框，里面有一个数值，这就是变量“div“的当前值，如上图。在脚本调试的时候，你可以通过这个方法很方便的了解到当前变量的值。你还可以通过右边窗口的“监控（Watch）”标签来查看到“div”变量的值，如图



在“监控（Watch）”标签窗口我们可以通过点击向下箭头弹出菜单过滤要显示的监控内容，过滤条件有以下几种：

* 显示用户自定义的属性（Show User-defined Properties）
* 显示用户自定义的函数（Show User-defined Functions）
* 显示DOM属性（Show DOM Properties）
* 显示DOM函数(Show DOM Functions)
* 显示DOM常数(Show DOM Constants)。
* 显示行内事件处理代码
* 仅显示自有属性
* 仅显示枚举属性

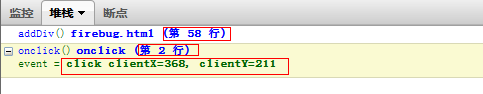


我们还可以通过单击“新建监控表达式（New watch expression…）”加入自己想跟踪的内容。例如我们想跟踪一下“div ”的显示内容，就可以单击“新建监控表达式…”，加入“div.innerHTML”，输入中可通过TAB键自动完成关键字的输入。如果不想跟踪了，可单击最右边的删除图标取消跟踪。如图

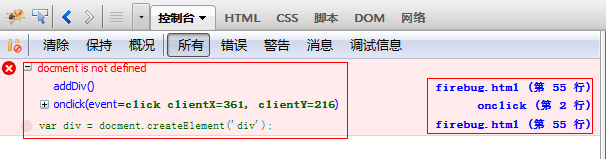


该标签中对应的搜索框的作用和HTML源代码查看是一样的，不过有一点不同，就是输入“#n”（n≥1），可以直接跳到脚本的第n行。

调试过程中还可以通过查看堆栈来跟踪各方法（函数）的调用情况，比如a方法调用b方法，b方法又调用c方法，当调试代码进入c方法时，就会显示堆栈a->b->c的过程。堆栈中还会显示鼠标或键盘事件定位的坐标（如果调试的是鼠标或键盘事件），还可以通过堆栈来定位方法的位置，如图

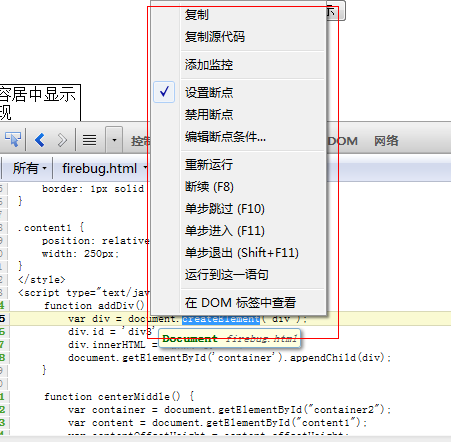


如果调试过程中遇到错误，调试会中断结束，同时会在控制台中显示错误信息，以及对应的错误代码，用户根据错误提示可以很方便的找到产生错误的原因，如图



可以点击【清除】来清空控制台的错误信息

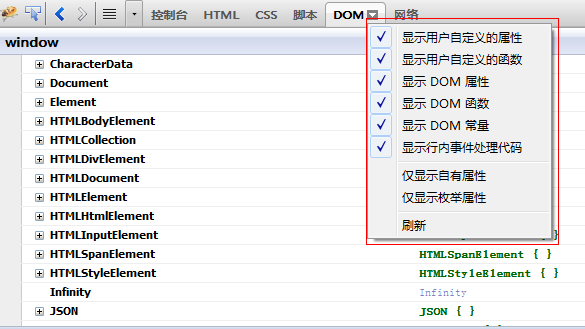
调试过程中，用户还可以针对某行代码来设置操作。右击代码行，弹出相应的菜单选项来完成相应功能，如图



## DOM查看

将firebug调试窗口第二行的标签切换到“DOM”时，可以按层次查看整个页面的DOM结构。通过选项菜单（点击下拉箭头）可自定义过滤要查看的DOM，主要有

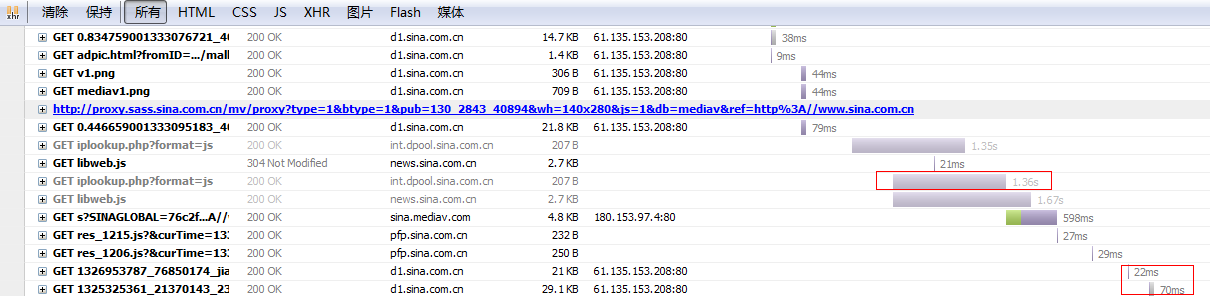
* 显示用户自定义的属性（Show User-defined Properties）
* 显示用户自定义的函数（Show User-defined Functions）
* 显示DOM属性（Show DOM Properties）
* 显示DOM函数(Show DOM Functions)
* 显示DOM常数(Show DOM Constants)
* 显示行内事件处理代码
* 仅显示自有属性
* 仅显示枚举属性



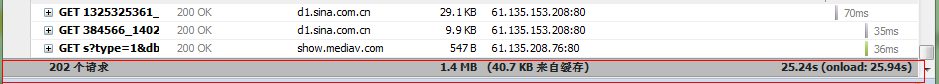
通过双击可以修改DOM里面的属性值（部分属性是不可以修改的，比如函数的length属性）。而方法和函数是不让修改的。

## 网络

作为开发人员，是否会经常听到老板或客户抱怨页面下载太慢了。于是你就怀疑是否脚本太多了？忘记压缩图片了？服务器太慢了？网络太慢？确实是头疼的事情。有了Firebug，你就可以很容易的对这个问题进行分析和判断了。请将Firebug的当前标签切换到“Net”，如图



上图是访问sina.com.cn部分分析结果。从图中我们可以看到，页面中每一个下载文件或服务器请求都用一个灰色条表示它相对其它是从什么时候开始下载的，下载时间是多少。在底部我们可以看到页面发送了多少个请求，下载总量是多少，有多少是有缓存的，下载总共花费了多少时间等信息。如图

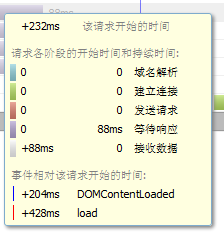


如果只想了解某一样文件类型的下载情况，你可以单击功能区第一栏的文件分类按钮来过滤文件，如图



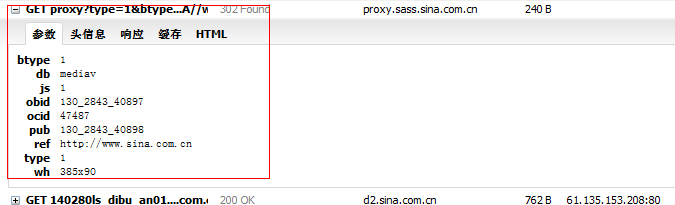
如果显示为红色的文件名，则表示该文件在服务器中不存在，不能下载，这样你就要检查一下文件的路径是否正确或者是否上传了该文件。

我们可以把鼠标停放在时间轴上，这时候会显示该请求的详细信息，如图

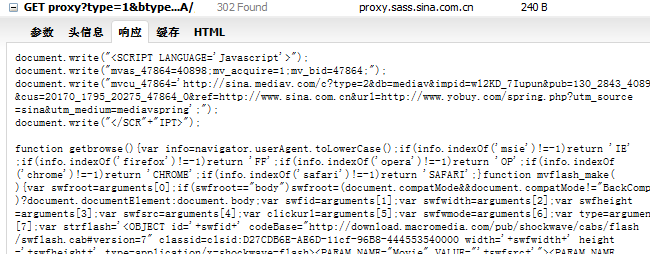


我们可以展开某个请求，查看它的HTTP头信息和返回结果的信息。如果请求的是一个动态页面或XMLHttpRequest，则还可以查看提交的变量。通过查看提交的变量和返回信息，我们可以很方便的调试程序是否正确提交了需要的变量和返回了正确的数据。如图





从上图中，我们可以看到发送请求的情况，提交的参数有9个，从这里我们可以很方便的知道我们脚本操作提交的参数是否正确。切换到“响应（Response）”页签可以看到返回的结果，如图



在这里你可以对返回结果进行检查。还可以查看“头信息”、“缓存”、“HTML”。

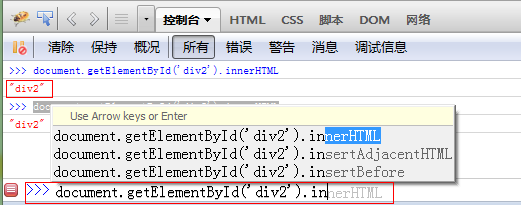
如果你感觉在这里查看结果很乱，你可以单击鼠标右键，在弹出菜单中选择“复制响应头信息（Copy Response body）”复制结果到编辑器查看，你还可以选择“在新页签中打开（Open in New Tab）”打开一个新标签浏览。

通过右键菜单你可以复制文件地址（Copy Location）、HTTP请求头信息(Copy Request Headers)和HTTP响应头信息（Copy Response Headers）。

## 命令行工具

### 控制台命令介绍

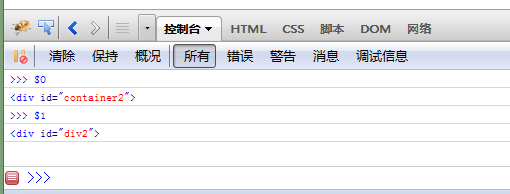
在“Console”标签中有一个命令行工具，我们可以在这里运行一些脚本对页面进行调试。 我们在命令行中输入“document.getElementById('div2').innerHTML”点击回车或【运行】按钮，在信息区会显示了当前“div2”的显示内容。如图



要输入“document.getElementById”是不是觉得很麻烦？这里有一个简单的办法，用“$”符号代替“document.getElementById”，我们再在命令行中输入“$('div2').innerHTML”，然后看看结果，是一样的。

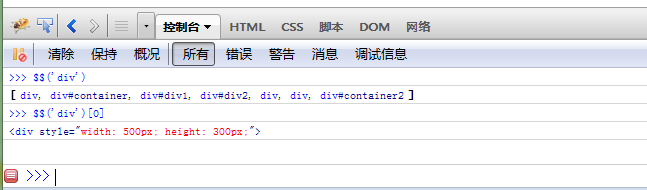
### “$n”(n>0)的用法

当你通过 “Inspect”锁定了一些HTML Element时，你可以通过“$0”来访问最后一个锁定的Element，依次类推，我们可以通过“$n”(n>0)来访问依次倒序访问锁定的Element。我们来实践一下，刷新一下测试页面，然后按下“Inspect”按钮，鼠标单击“div2”，然后在按下“Inspect”按钮，单击“内容居中显示实现”。将firebug窗口切换回“Console”标签，然后输入“$0”，回车后再输入“$1”，查看一下结果，正是我们之前锁定过的Element。如图



### $$(HTML 标签)的用法

在命令行还可以通过“$$(HTML 标记)”返回一个Element数组。我们在测试页输入“$$('div')”看看，我们再输入“$$('div') [0]”看看，如图。是不是很便于我们对HTML进行调试。



### 命令行所有特殊函数

请看下表

|  |  |
| --- | --- |
| 命令 | 说明 |
| $(id) | 通过id返回Element。 |
| $$(selector) | 通过CSS选择器返回Element数组。 |
| $x(xpath) | 通过xpath表达式返回Element数组。 |
| dir(object) | 列出对象的所有属性，和在DOM标签页查看该对象的是一样的。 |
| dirxml(node) | 列出节点的HTML或XML的源代码树，和在HTML标签页查看改节点一样。 |
| cd(window) | 默认情况下，命令行相关的是顶层window对象，使用该命令可切换到frame里的window独享。 |
| clear() | 清空信息显示区，和单击按钮Clear功能一样。 |
| inspect(object[, tabName]) | 监视一个对象。tabName表示在那个标签页对该对象进行监视，可选值为“html”、“css”、“script”和“dom”。 |
| keys(object) | 返回由对象的属性名组成的数组。 |
| values(object) | 返回由对象的属性值组成的数组。 |
| debug(fn) | 在函数的第一行增加一个断点。 |
| undebug(fn) | 移除在函数第一行的断点。 |
| monitor(fn) | 跟踪函数fn的调用。 |
| unmonitor(fn) | 不跟踪函数fn的调用。 |
| monitorEvents(object[, types]) | 跟踪对象的事件。Types的可选值为“composition”、 “contextmenu”、 “drag”、 “focus”,、“form”、“key”、 “load”、“mouse”、“mutation”、“paint”、“scroll”、“text”、“ui”和“xul”。 |
| unmonitorEvents(object[, types]) | 不跟踪对象的事件。Types的可选值为“composition”、 “contextmenu”、 “drag”、 “focus”,、“form”、“key”、 “load”、“mouse”、“mutation”、“paint”、“scroll”、“text”、“ui”和“xul”。 |
| profile([title]) | 开始对脚本进行性能测试，可选参数title将作为测试结果的标题。 |
| profileEnd() | 结束脚本性能测试。 |

命令行有命令记忆功能，可通过上、下箭头键选择已经输入过的命令。

### alert的替代

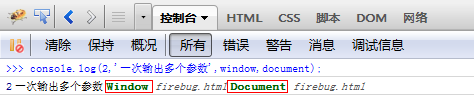
1、有木有对脚本调试中经常需要alert感到厌烦呢？有了Firebug，你就可以放下alert了，因为Firebug有功能比alert更强大的console先让我们来认识一下console吧。先看以下代码

|  |
| --- |
| console.log('Hello World!'); |

在控制台运行该代码后，会输入Hello World!，console.log可以象c语言的printf一样实现格式化输出。我们再在脚本区加入一条语句：

|  |
| --- |
| console.log(2, '一次输出多个参数',window ,document); |

输出结果如图

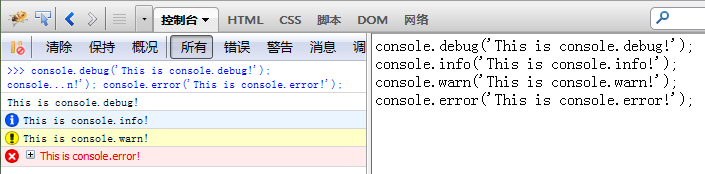


从上图可以看出console.log可以连续输出多个对象，而且如果对象是DOM、函数，还可以直接点击去查看该对象。

2、除了console.log，console还提供了console.debug、 console.info、 console.warn和console.error，这些函数分别会用不同的背景颜色和文字颜色来显示信息。我们看如下代码

|  |
| --- |
| console.debug('This is console.debug!');  console.info('This is console.info!');  console.warn('This is console.warn!');  console.error('This is console.error!'); |

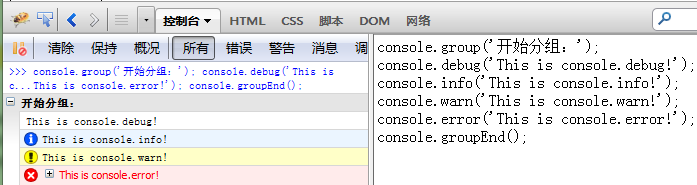
运行效果如图



3、有时候，为了更清楚方便的查看输出信息，我们可能需要将一些调试信息进行分组输出，那么可以使用console.group来对信息进行分组，在组信息输出完成后用console.groupEnd结束分组。我们测试一下把刚才的4个输出作为一个分组输出，修改代码为：

|  |
| --- |
| console.group('开始分组：');  console.debug('This is console.debug!');  console.info('This is console.info!');  console.warn('This is console.warn!');  console.error('This is console.error!');  console.groupEnd(); |

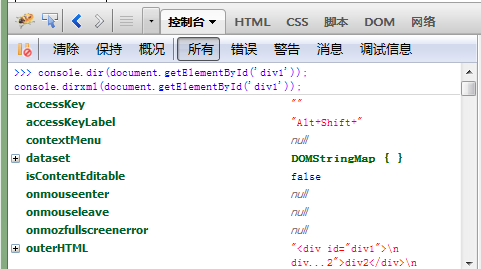
运行效果为：



4、有时候，我们需要写一个for循环列出一个对象的所有属性或者某个HTML Element下的所有节点，有了firebug后，我们不需要再写这个for循环了，我们只需要使用console.dir(object)或console.dirxml(element)就可以了。

|  |
| --- |
| console.dir(document.getElementById('div1'));  console.dirxml(document.getElementById('div1')); |

dir方法运行效果 ，显示该element下所有属性和方法



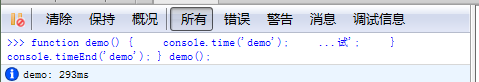
dirxml方法运行效果，显示该element下所有节点



5、是否想知道代码的运行速度？很简单，使用console.time和console.timeEnd就可以。例如我们测试以下方法的运行效率

|  |
| --- |
| **function** demo() {  console.time('demo');  **for** (**var** i = 0; i < 1000; i++) {  document.getElementById('div2').innerHTML = 'console.time测试';  }  console.timeEnd('demo');  }  demo(); |

运行效果



6、是否想知道某个函数是从哪里调用的？console.trace()可帮助我们进行追踪。比如在示例页面中函数addDiv的结尾加入：

|  |
| --- |
| console.trace(); |

刷新页面后，点击按钮【点击我看看效果】，控制台显示效果，如图



跟调试时查看堆栈信息类似。

7、如果想在脚本某个位置设置断点，可以在脚本中输入“debugger”作为一行。当脚本执行到这一行时会停止执行等待用户操作，这时候可以通过切换到“脚本”标签对脚本进行调试。

8、Firebug调试函数，如下表

|  |  |
| --- | --- |
| 函数 | 说明 |
| console.log(object[, object, ...]) | 向控制台输出一个信息。可以输入多个参数，输出将已空格分隔各参数输出。  第一参数可以包含格式化文本，例如：  console.log(‘这里有%d个%s’,count,apple);  字符串格式：  %s ：字符串。  %d, %i：数字。  %f： 浮点数。  %o -超链接对象。 |
| console.debug(object[, object, ...]) | 向控制台输出一个信息，信息包含一个超链接链接到输出位置。 |
| console.info(object[, object, ...]) | 向控制台输出一个带信息图标和背景颜色的信息，信息包含一个超链接链接到输出位置。 |
| console.warn(object[, object, ...]) | 向控制台输出一个带警告图标和背景颜色的信息，信息包含一个超链接链接到输出位置。 |
| console.error(object[, object, ...]) | 向控制台输出一个带错误图标和背景颜色的信息，信息包含一个超链接链接到输出位置。 |
| console.assert(expression[, object, ...]) | 测试一个表示是否为true，如果为false，提交一个例外信息到控制台。 |
| console.dir(object) | 列出对象的所有属性。 |
| console.dirxml(node) | 列出HTML或XML Element的XML源树。 |
| console.trace() | 输出堆栈的调用入口。 |
| console.group(object[, object, ...]) | 将信息分组再输出到控制台。通过console.groupEnd()结束分组。 |
| console.groupEnd() | 结束分组输出。 |
| console.time(name) | 创建一个名称为name的计时器，计算代码的执行时间，调用console.timeEnd(name)停止计时器并输出执行时间。 |
| console.timeEnd(name) | 停止名称为name的计时器并输出执行时间。 |
| console.profile([title]) | 开始对脚本进行性能测试，title为测试标题。 |
| console.profileEnd() | 结束性能测试。 |
| console.count([title]) | 计算代码的执行次数。titile作为输出标题。 |
|  | |

## IE中插件Firebug Lite

Firebug是Firefox的一个插件，但是基于IE开发的项目怎么调试我的页面呢？如果在页面脚本中加入console.log()将调试信息写到Friebug，在IE中肯定是提示错误的（IE9也支持console的部分功能），怎么办？不用担心，Frirebug提供了Frirbug Lite插件，可以插入页面中模仿Firebug控制台。我们可以从以下地址下载firebug lite：<http://getfirebug.com/firebuglite>查找下载链接，或者直接从<https://getfirebug.com/releases/lite/latest/firebug-lite.tar.tgz>下载。

下载后在要调试的页面中加入如下代码

<script type="text/javascript" src="/local/path/to/firebug-lite.js"></script>

或者直接远程加载

<script type="text/javascript" src="https://getfirebug.com/firebug-lite.js"></script>

如果你不想在IE中模仿Friebug控制台，只是不希望console.log()等脚本出现错误信息，那么在页面中加入如上语句即可。

如果你不想安装Firebug Lite，只是想避免脚本错误，那么可以在脚本中加入以下语句：

|  |
| --- |
| **if** (!window.console || !console.firebug) {  **var** names = ["log", "debug", "info", "warn", "error", "assert", "dir", "dirxml", "group",  "groupEnd", "time", "timeEnd", "count", "trace", "profile", "profileEnd"];  window.console = {};  **for** (**var** i = 0; i < names.length; ++i)  window.console[names[i]] = **function**() {  }  } |

## FireBug快捷键和鼠标操作

|  |  |
| --- | --- |
| 全局操作 | |
| 打开Firebug窗口 | F12 |
| 关闭Firebug窗口 | F12 |
| 在新窗口打开Firebug | Ctrl+F12 |
| 往前切换标签 | Ctrl+` |
| 将光标移到命令行 | Ctrl+Shift+L |
| 将光标移到搜索框 | Ctrl+Shift+K |
| 进入Inspect模式 | Ctrl+Shift+C |
| 进行JavaScript性能测试 | Ctrl+Shift+P |
| 重新执行最后一条命令行命令 | Ctrl+Shift+E |
| HTML标签 | |
| 编辑属性 | 单击属性名或值 |
| 编辑文本节点 | 单击文本 |
| 编辑Element | 双击Element标记 |
| 移到路径里的下一个节点 | Ctrl+. |
| 移到路径里的上一个节点 | Ctrl+, |
| HTML编辑 | |
| 完成编辑 | Return |
| 取消编辑 | Esc |
| 移到下一个区域 | Tab |
| 移到上一个区域 | Shift+Tab |
| HTML Inspect 模式 | |
| 取消Inspect | Esc |
| Inspect 父节点 | Ctrl+Up |
| Inspect 子节点 | Ctrl+Down |
| Script标签 | |
| 继续运行 | F5 |
|  | Ctrl+/ |
| 单步执行（Step Over） | F10 |
|  | Ctrl+' |
| 单步执行（Step Into） | F11 |
|  | Ctrl+; |
| 退出函数（Step Out） | Shift+F11 |
|  | Ctrl+Shift+; |
| 设置断点 | 单击行号 |
| 禁止断点 | 在行号上Shift+Click |
| 编辑断点条件 | 在行号上Right-Click |
| 运行到当前行 | 在行号上Middle-Click |
|  | 在行号上Ctrl+Click |
| 移到堆栈中的下一个函数 | Ctrl+. |
| 移到堆栈中的上一个函数 | Ctrl+, |
| 将焦点切换到Scripts菜单 | Ctrl+Space |
| 将焦点切换到Watch编辑 | Ctrl+Shift+N |
| DOM 标签 | |
| 编辑属性 | 双击在空白 |
| 移到路径中下一个对象 | Ctrl+. |
| 移到路径中上一个对象 | Ctrl+, |
| DOM 和Watch编辑 | |
| 结束编辑 | Return |
| 取消编辑 | Esc |
| 自动完成下一个属性 | Tab |
| 自动完成上一个属性 | Shift+Tab |
| CSS标签 | |
| 编辑属性 | 单击属性 |
| 插入新属性 | 双击空白处 |
| 移动焦点到Style Sheets菜单 | Ctrl+Space |
| CSS编辑 | |
| 完成编辑 | Return |
| 取消编辑 | Esc |
| 移到下一个区域 | Tab |
| 移到上一个区域 | Shift+Tab |
| 按步长1增加数值 | Up |
| 按步长1减少数值 | Down |
| 按步长10增加数值 | Page Up |
| 按步长10减少数值 | Page Down |
| 自动完成下一个关键字 | Up |
| 自动完成上一个关键字 | Down |
| Layout标签 | |
| 编辑值 | 单击值 |
| Layout编辑 | |
| 完成编辑 | Return |
| 取消编辑 | Esc |
| 移到下一个区域 | Tab |
| 移到上一个区域 | Shift+Tab |
| 按步长1增加数值 | Up |
| 按步长1减少数值 | Down |
| 按步长10增加数值 | Page Up |
| 按步长10减少数值 | Page Down |
| 自动完成下一个关键字 | Up |
| 自动完成上一个关键字 | Down |
| 命令行 (小) | |
| 自动完成上一个属性 | Tab |
| 自动完成下一个属性 | Shift+Tab |
| 执行 | Return |
| Inspect结果 | Shift+Return |
| 打开结果鼠标右键菜单 | Ctrl+Return |
| 命令行 (大) | |
| 执行 | Ctrl+Return |

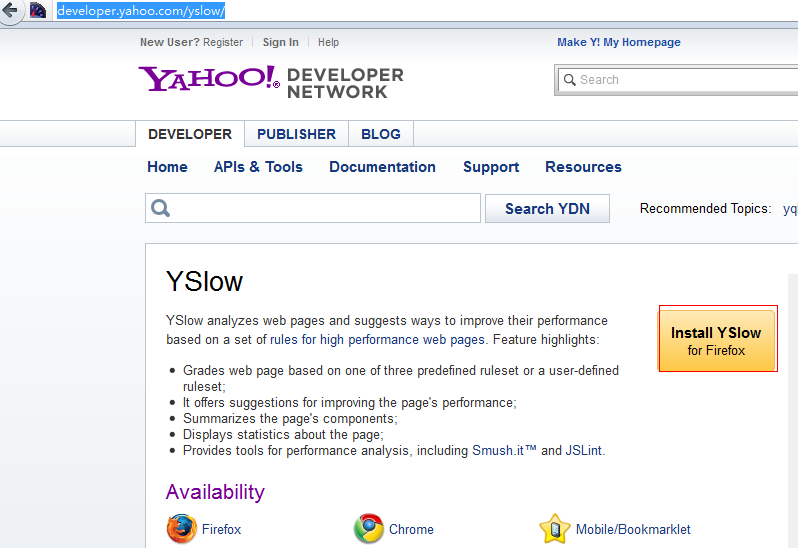
## 其他插件

### YSlow

1、安装

YSlow是Yahoo开发的，基于Firebug的用于分析网页性能的工具，可以提供如何提高网站性能的一套规则，用于优化网页的速度和建立高性能的网页。YSlow可以分析任何网站，并为每一个规则产生一个整体报告，如果页面可以进行优化，则YSlow会列出具体的修改意见。

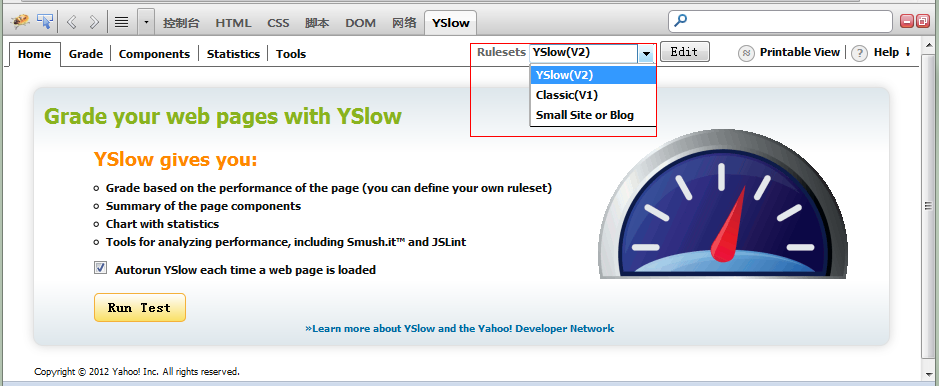
安装地址为：<http://developer.yahoo.com/yslow/>，打开Firefox，在地址栏中输入该地址，如图



点击【Install YSlow for Firefox】进入插件安装界面，界面中会提示你安装，安装过程可参考Firebug的安装。

2、概述

安装好后打开YSlow，如图



点击 Run Test 运行Yslow，也可以点击 Grade、Components、Statistics或Tools选项开始对页面分析。如果在 Autorun YSlow each time a web page is loaded 上打上对勾，它将自动对以后打开页面进行分析。

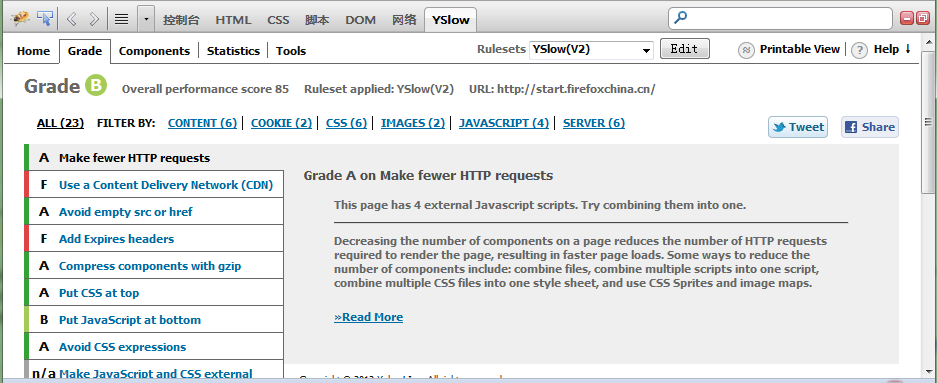
注意图中的红框，这里是规则集，YSlow （V2）包含了所有23个测试的规则，Classic （V1）包含原始13规则，Small Site or Blog（小网站或博客）这个规则集包含14个规则，适用于小型网站或博客，建议对号入座。

3、雅虎评估网站性能的23条规则

雅虎曾经针对网站速度提出了非常著名34条准则：《Best Practices for Speeding Up Your Web Site》。而现在将34条精简为更加直观的23条，并针对每一条给出从F~A的评分以及最终的总分。而现在23条网站性能优化建议在YSlow的官网首页（http://developer.yahoo.com/yslow/）就能看到。在使用Yslow后，在控制面板里就会给你评分提示和改进建议。下面列出了雅虎定义的Web性能最佳实践和规则

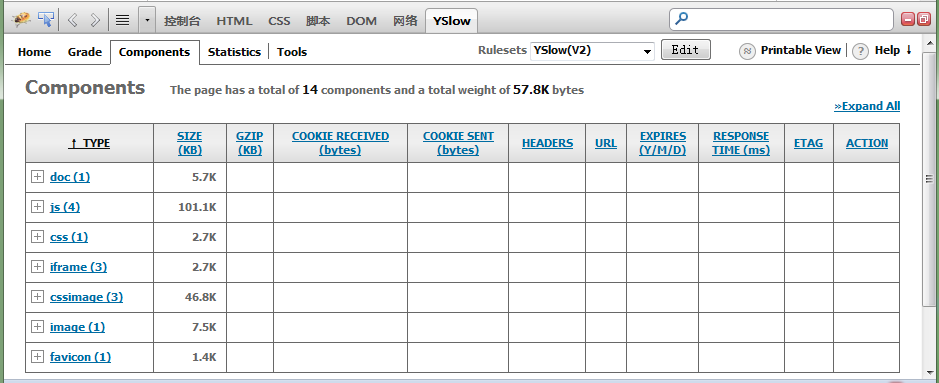
1. [Minimize HTTP Requests](http://developer.yahoo.com/performance/rules.html#num_http) 尽可能减少HTTP请求次数
2. [Use a Content Delivery Network](http://developer.yahoo.com/performance/rules.html#cdn) 使用CDN
3. [Avoid empty src or href](http://developer.yahoo.com/performance/rules.html#emptysrc) 避免空的src和href标签
4. [Add an Expires or a Cache-Control Header](http://developer.yahoo.com/performance/rules.html#expires) 加入一个Expires或Cache-Control Header
5. [Gzip Components](http://developer.yahoo.com/performance/rules.html#gzip) 使用Gzip压缩组件
6. [Put StyleSheets at the Top](http://developer.yahoo.com/performance/rules.html#css_top) 在html文件顶部放置样式表
7. [Put Scripts at the Bottom](http://developer.yahoo.com/performance/rules.html#js_bottom) 在html文件底部放置JavaScript脚本
8. [Avoid CSS Expressions](http://developer.yahoo.com/performance/rules.html#css_expressions) 避免使用CSS表达式
9. [Make JavaScript and CSS External](http://developer.yahoo.com/performance/rules.html#external) 使用外部JavaScript和CSS外部文件
10. [Reduce DNS Lookups](http://developer.yahoo.com/performance/rules.html#dns_lookups) 减少使用DNS查找次数
11. [Minify JavaScript and CSS](http://developer.yahoo.com/performance/rules.html#minify) 精简JavaScript和CSS
12. [Avoid Redirects](http://developer.yahoo.com/performance/rules.html#redirects) 避免重定向
13. [Remove Duplicate Scripts](http://developer.yahoo.com/performance/rules.html#js_dupes) 移除重复的脚本
14. [Configure ETags](http://developer.yahoo.com/performance/rules.html#etags) 配置ETag
15. [Make AJAX Cacheable](http://developer.yahoo.com/performance/rules.html#cacheajax) 缓存AJAX
16. [Use GET for AJAX Requests](http://developer.yahoo.com/performance/rules.html#ajax_get) 使用GET完成AJAX请求
17. [Reduce the Number of DOM Elements](http://developer.yahoo.com/performance/rules.html#min_dom) 减少DOM元素数量
18. [No 404s](http://developer.yahoo.com/performance/rules.html#no404) 避免404
19. [Reduce Cookie Size](http://developer.yahoo.com/performance/rules.html#cookie_size) 减少Cookie大小
20. [Use Cookie-Free Domains for Components](http://developer.yahoo.com/performance/rules.html#cookie_free) 使用无Cookie的域
21. [Avoid Filters](http://developer.yahoo.com/performance/rules.html#no_filters) 避免使用滤镜
22. [Do Not Scale Images in HTML](http://developer.yahoo.com/performance/rules.html#no_scale) 不要在HTML中缩放图片
23. [Make favicon.ico Small and Cacheable](http://developer.yahoo.com/performance/rules.html#favicon) 使用小favicon.ico文件，并让其可缓存

4、Grade(等级视图)—Yslow的第二个选项卡



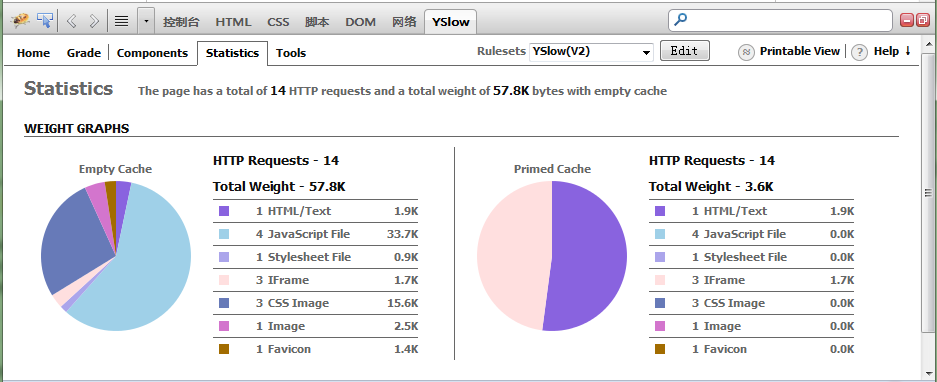
Yslow给出的网站性能评分，从F~A，A是最好的，通过测试Firebox首页（<http://start.firefoxchina.cn/>）来看，网站有2处得分最低。这里推荐一篇文章[提高网站打开速度的7大秘籍](http://lusongsong.com/reed/360.html)。

5、Components（组件视图）—YSlow的第三个选项卡



通过Components考验查看网页各个元素占用的空间大小，包括doc、js、css、iframecssimage、image、favicon，通过分析可以查看那些资源文件可以优化。

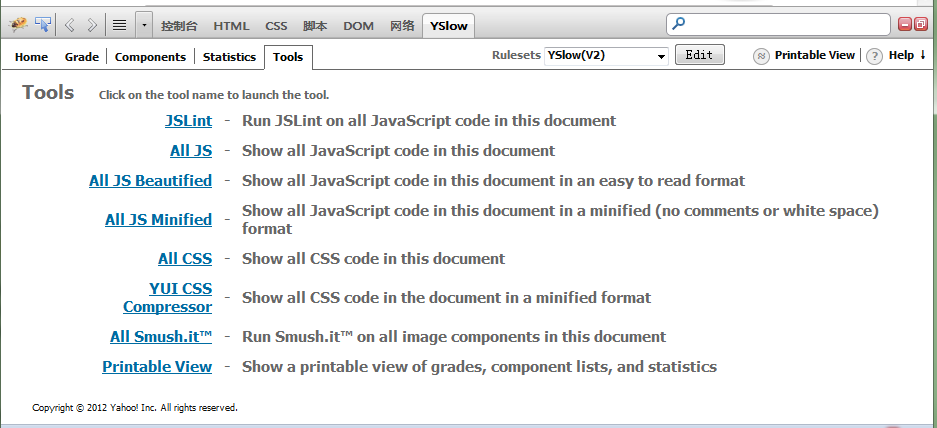
6、Statistics（统计信息视图）—Yslow的第四个选项卡



左侧图表显示是页面元素在空缓存的加载情况，右侧为页面元素使用缓存后的页面加载情况。从图中可以直观的看出，这个网页有14个HTTP请求，网页的大小达到57.8K，意味着第一次打开一个页面需要下载57.8K的东西，而通过使用缓存后我们可以看到js文件、图片、css文件等基本靠缓存，而网页的总大小也变为3.6K。对应网页比较大的网站，利用缓存处理是不错的选择。

Statistics这个统计信息视图工具和Components（第三选项卡）一样，只是效果更直观，如果要获得性能优化建议还是要看Grade（第二选项卡）的详细建议。

7、Tools（辅助工具）—Yslow的第五个选项卡



在该标签下提供了几个实用的工具

* JSLint是一个强大的工具，它可以检验HTML代码以及内联的Javascript代码，通过JSLint发现了google analytics上的一个js错误。
* ALL JS：查看你这个网页上一共引用了多少JS。
* All JS Beautified：把所有JS放在打开的页面中，利用站长统一检查（我感觉作用不大）。
* All JS Minified：同上，但它显示的是压缩过的js代码，如果你要JS优化，它已经给你优化好了，来过来直接用。
* All CSS：显示你网页所有CSS文件。
* YUI CSS Compressor：显示网页压缩后的CSS文件，也是拿过来可以直接用的。
* All Smush.it™：图片在线优化网站，点击它后会自动跳到smushit网站上给你自动优化CSS图片，该网站提供了优化前与优化后的对比，点击直接下载优化后的图片，在覆盖到自己网站上就可以了，强烈推荐。
* Printable View：这个是打印用的，部门开会、前端设计师讨论、向老板汇报时估计用的上。

关于YSlow的用法可以参考以下网址

<http://www.cnblogs.com/fullhouse/archive/2012/01/05/2313105.html>

### JSLint

JSLint是一个JavaScript验证工具（非开源），可以扫描JavaScript源代码来查找问题。如果JSLint发现一 个问题，JSLint就会显示描述这个问题的消息，并指出错误在源代码中的大致位置。有些编码风格约定可能导致未预见的行为或错误，JSLint除了能指 出这些不合理的约定，还能标志出结构方面的问题。尽管JSLint不能保证逻辑一定正确，但确实有助于发现错误，这些错误很可能导致浏览器的 JavaScript引擎抛出错误。

JSLint定义了一组编码约定，这比ECMA定义的语言更为严格。这些编码约定汲取了多年来的丰富编码经验，并以一条年代久远的编程原则 作为宗旨：能做并不意味着应该做。JSLint会对它认为有的编码实践加标志，另外还会指出哪些是明显的错误，从而促使你养成好的 JavaScript编码习惯。

JSLint可能会把一些结构方面的错误标志为可疑的编码实践，以下列出了其中一部分（完整的列表可以参考JSLint的文档）。

JSLint要求所有代码行都以分号结束。尽管JavaScript确实允许将换行符作为行结束符，但一般认为这种做法是不明确的，而且是不好的编码风格。

使用if和for的语句必须使用大括号把语句块括起来。

不同于其他编程语言，在JavaScript中，块不会作为变量的作用域。JavaScript只支持函数级作用域。因此，JSLint只接受作为function、if、switch、while、for、do和try语句一部分的块，其他的块都会标志为错误。

var只能声明一次，而且在使用之前必须声明。

JSLint会把出现在return、break、continue或throw语句后面的代码标志为不可达的代码。这些语句后面必须紧跟一个结束大括号。

对于JavaScript来说，JSLint是一个非常好的工具，因为它会教你一些好的JavaScript编码实践。由于JSLint能 把可能导致逻辑错误或其他未预见行为的部分标出来，因此可以减少调试时间。如果你调试一段JavaScript代码时遇到困难，可以试试JSLint。

**在YSlow中辅助工具中可以看到JSLint，点击可以分析网站javascript**

更详细的介绍可以参考文章

<http://hi.baidu.com/fengwill/blog/item/c2ec8a501e696775843524f3.html>

### [Page Speed](http://code.google.com/speed/page-speed/)

可以从以下网址安装该插件

<http://code.google.com/intl/zh-CN/speed/page-speed/>

Page Speed是Google开发的类似YSlow的工具，也是基于Firebug插件，网站管理员和网络开发人员可以使用 Page Speed 来评估他们网页的性能，并获得有关如何改进性能的建议。

详细说明可参见百度百科<http://baike.baidu.com/view/7053766.htm>和

<http://www.infoq.com/cn/news/2009/06/Page-Speed>

### 其他可选插件

主要列举了几个Firefox比较常用的插件，尤其是对于开发人员来说，可以到Firefox插件库中直接查找安装。

* JSONView 在浏览器中直接查看json文件，可以高亮显示代码
* Web Developer 该插件集成了一系列Web开发工具，例如HTML、CSS校验、FORM修改等功能，拥有强大的CSS调试和Form调试能力，对页面的分析非常全面，可以禁止页面的任一内联、文档、和外部CSS，可以直接编辑当前CSS。

## 参考资料

以下网址详细的介绍了firebug的使用，大家可以参考这些

Firebug的安装和使用：

<http://hi.baidu.com/wgxg/blog/item/fa443912da5ae759f819b819.html>

以下介绍了Firebug的安装和使用以及技巧，可以选择性的查看

<http://www.iteye.com/topic/976935>

<http://www.360doc.com/content/11/0816/08/7461278_140697849.shtml>

<http://www.360doc.com/content/11/0610/19/6598137_126071096.shtml>

<http://www.iteye.com/news/23953>

<http://www.ooso.net/archives/294>

YSlow 介绍

<http://lusongsong.com/reed/362.html>