
目錄

Introduction	1.1
学习基础	1.2
概览	1.2.1
开发工作流	1.2.2
使用控制台	1.2.3
小技巧——控制台篇	1.2.4
小技巧——Timeline篇	1.2.5
小技巧——Profiles篇	1.2.6
小技巧——代码篇	1.2.7
小技巧——页面元素篇	1.2.8
小技巧——网络篇&设置	1.2.9
工具使用	1.3
样式编辑与DOM介绍(todo)	1.3.1
CSS预处理器(todo)	1.3.2
应用存储(todo)	1.3.3

Chrome 开发者工具

author:leeon

- 学习基础
 - 概览
 - 开发工作流
 - 使用控制台
 - 小技巧——控制台篇
 - 小技巧——Timeline篇
 - 小技巧——Profiles篇
 - 小技巧——代码篇
 - 小技巧——页面元素篇
 - 小技巧——网络篇&设置
- 工具使用
 - 样式编辑与DOM介绍(todo)
 - CSS预处理器(todo)
 - 应用存储(todo)

Chrome DevTools 概览

Chrome开发者工具（简称DevTools）是一组网页制作和调试的工具，内嵌于Google Chrome浏览器中。DevTools使开发者更加深入的了解浏览器内部以及他们编写的应用。通过使用DevTools，可以更加高效的定位页面布局问题，设置JavaScript断点并且更好的理解代码优化。

注意: 如果你是一个web开发者并且希望获取最新版本的DevTools，你可以使用 [Google Chrome Canary](#).

访问 DevTools

访问DevTools，首先用Chrome打开一个web页面或web应用，也可以通过下面的方式：

- 在浏览器窗口的右上方选择Chrome菜单, 然后选择工具 > 开发者工具。
- 在页面上任意元素上右键，然后选择审查元素。

DevTools窗口会在Chrome浏览器的底部打开。

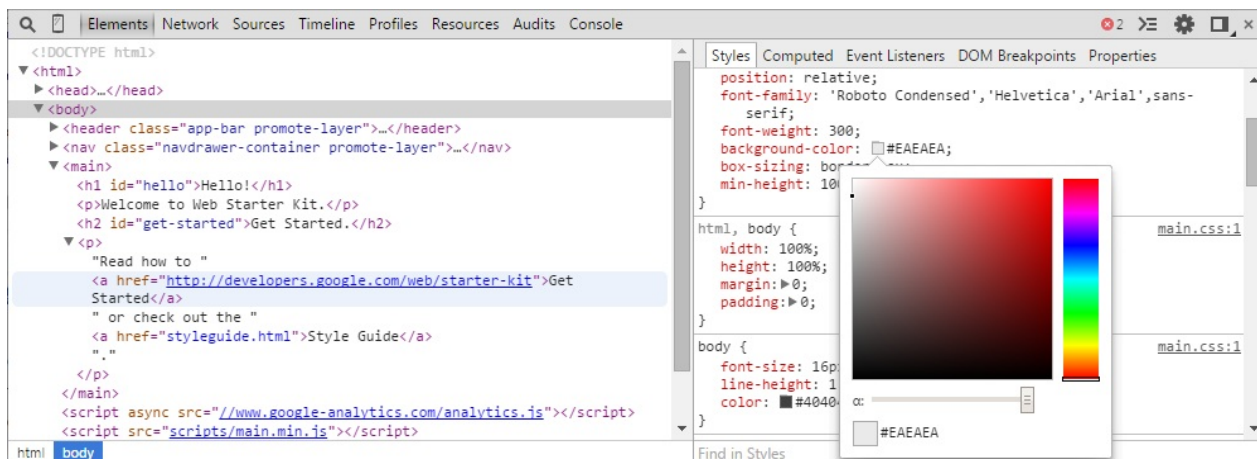
下面是一些有用的快捷键来快速的打开DevTools:

- 使用 `Ctrl + Shift + I` (Mac上为 `Cmd + Opt + I`) 打开DevTools。
- 使用 `Ctrl + Shift + J` (Mac上为 `Cmd + Opt + J`) 打开DevTools中的控制台
- 使用 `Ctrl + Shift + C` (Mac上为 `Cmd + Shift + C`) 打开DevTools的审查元素模式。

为了日常工作，[学习快捷键](#)将帮助你节省更多的时间。

DevTools 窗口

DevTools 在窗口顶部的工具栏对不同的任务功能进行分组。在每个工具栏选项卡和对应的操作面板中，你可以处理某项特殊的任务，例如DOM元素，资源和源码。



DevTools现在已经支持内置颜色选择器..

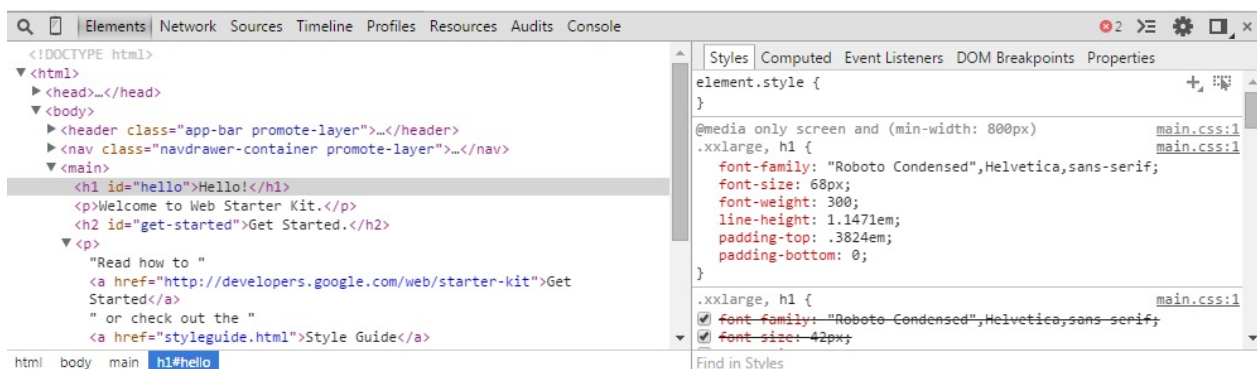
DevTools目前主要包括以下八个主要功能组：

- Elements
- Resources
- Network
- Sources
- Timeline
- Profiles
- Audits
- Console

你可以通过 `Ctrl + [` 和 `Ctrl +]` 快捷键在不同面板之间进行切换。

审查DOM元素和样式

在Elements面板中你可以通过DOM树的形式查看所有页面元素，并且可以对其进行所见即所得的编辑。当你需要确认页面某一个HTML片段的时候，可能经常需要使用Element面板，比如，你可能想知道一个图片元素是否拥有id属性以及id的具体值。



在DOM中查看一个head元素

[Read more about inspecting the DOM and styles](#)

使用 Console

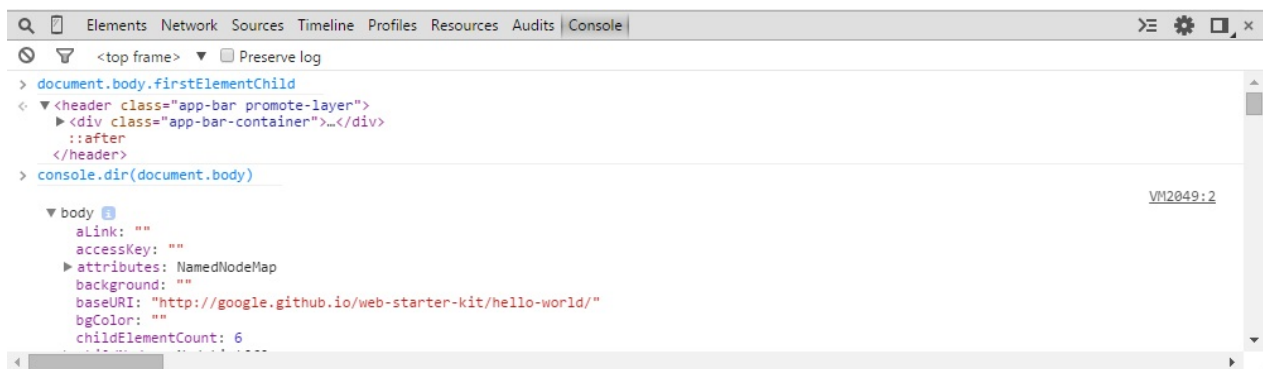
JavaScript 控制台 主要为开发者在测试web页面和应用的过程中提供两方面的功能：

- 在开发过程中，记录代码诊断信息。
- 作为shell提示窗口用来和页面文档以及DevTools进行交互。

你可以利用Console API提供的方法来记录诊断信息，例

如 `console.log()` 或 `console.profile()` 。

你可以利用命令行API提供的方法直接在控制台中计算表达式。例如，利用 `$()` 选择元素或者调用 `profile()` 来启动CPU性能监控。

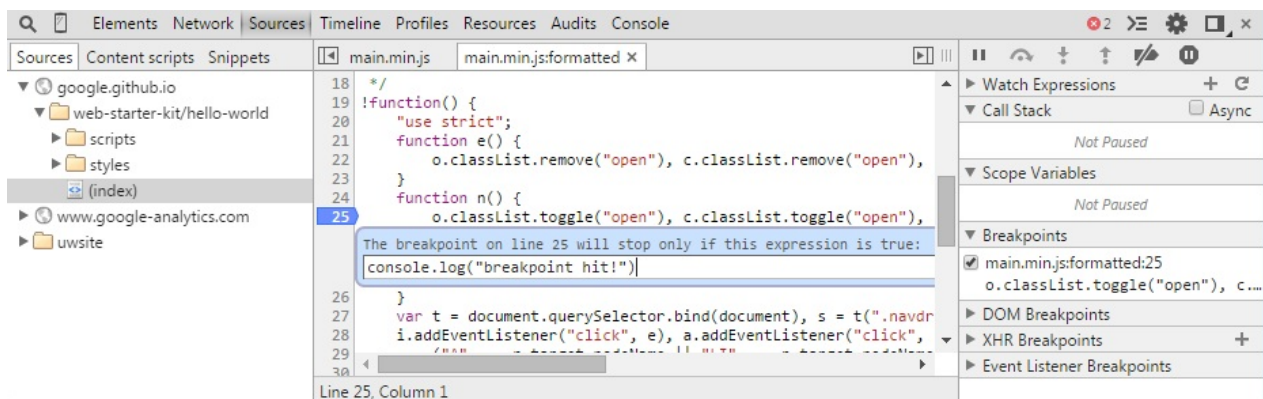


在JS控制台中执行一些命令

[了解更过关于命令行](#)

调试 JavaScript

随着JavaScript程序复杂度的增加，开发者需要更加强大的debug工具来快速的发现问题并高效的修复。DevTools包含了一系列有用的工具让调试JavaScript变得更加轻松。

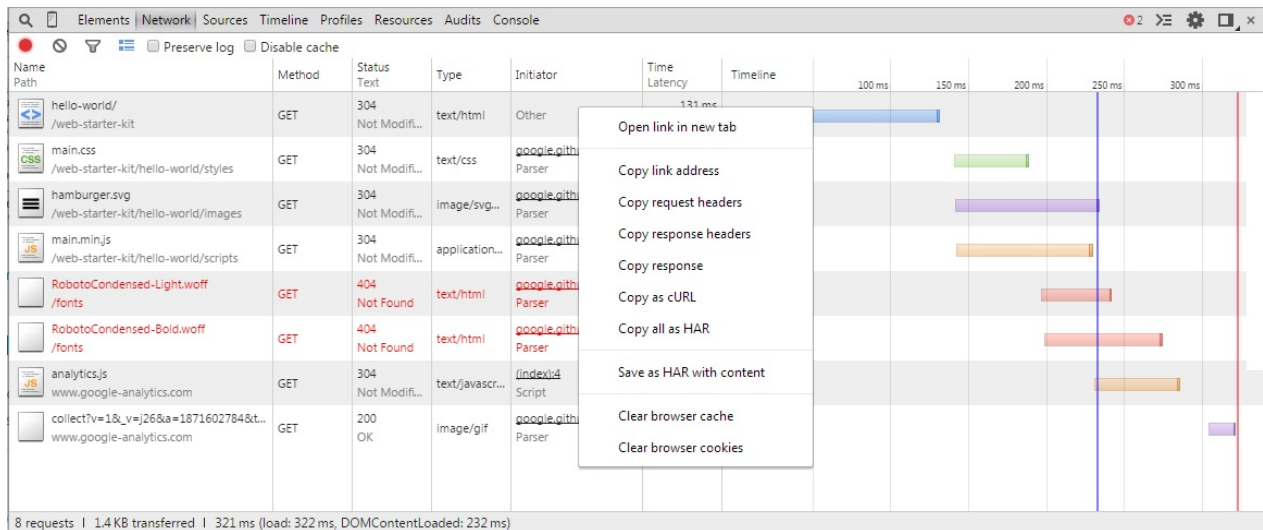


一个条件断点用于向终端输出日志

[阅读更多了解如何使用DevTools调试JavaScript代码](#)

提高网络性能

在网络面板中可以看到网络请求资源的实时信息。明确和定位那些比预期更加耗时的请求是优化web页面的关键步骤。

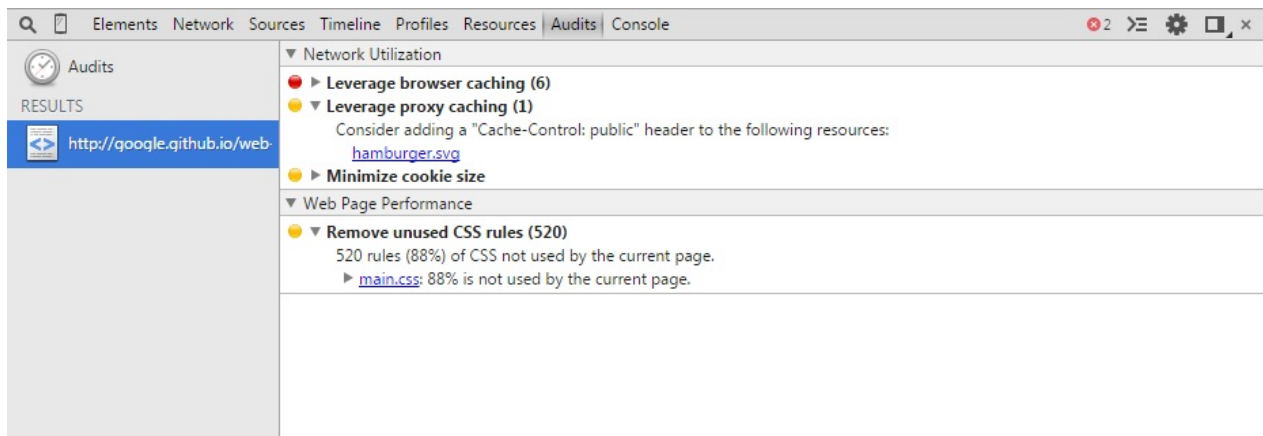


上下文目录用于查看网络请求

[阅读更多了解如何提高网络性能»](#)

监听

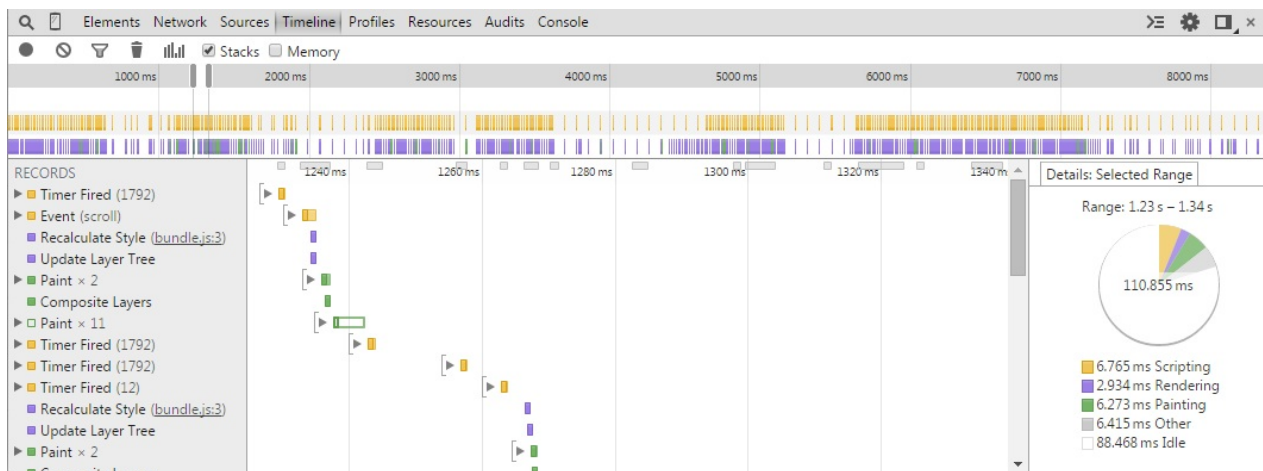
监听面板会在页面加载的时候对其进行分析，然后提供优化和建议来降低页面加载时间，并提高响应灵敏性。进一步查看，我们推荐使用[PageSpeed Insights](#)。



监听给出的建议

提高渲染性能

在Timeline面板中，你可以从整体上看到在web页面加载和被使用的过程中时间消耗在哪里了。所有的时间，从加载资源到解析JavaScript,计算样式和重绘都会被标记在时间线上。



标记着不同事件的时间线例子 [阅读更多了解如何提高渲染性能»](#)

JavaScript & CSS 性能

在分析面板中，你可以分析一个页面或app的执行时间和内存使用情况。该面板帮助你了解资源消耗在哪里，并且帮助你优化代码。目前提供的分析器包括：

- CPU分析器显示页面JavaScript函数的执行时间。
- Heap分析器页面JavaScript对象和相关联的DOM节点的内存分布情况。
- JavaScript分析器显示脚本的执行时间分布。

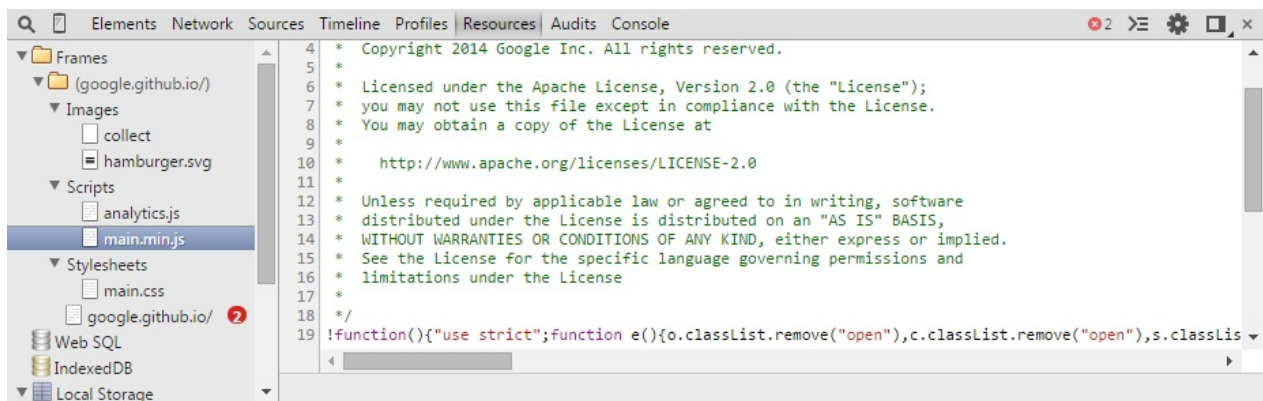
Summary	Class filter	All objects	Distance	Objects Co...	Shallow Size	Retained Size
Constructor						
▶ (compiled code)			3	4 559 8 %	1 849 176 30 %	2 614 888 43 %
▼ (array)			2	12 062 22 %	1 456 424 24 %	1 780 504 29 %
▼ (object properties)[] @39895			3		98 360 2 %	155 136 3 %
10037 :: "CanBeSafelyTreatedAsAnErrorObject" @61			4		64 0 %	64 0 %
10418 :: "GetPropertyWithoutInvokingMonkeyGetter			4		64 0 %	64 0 %
10769 :: "CALL_FUNCTION_PROXY_AS_CONSTRUCTOR" @6			4		64 0 %	64 0 %
11039 :: "ObjectInfoEnqueueExternalChangeRecord"			4		64 0 %	64 0 %
11507 :: "Uint8ClampedArrayConstructByLength" @6			4		64 0 %	64 0 %
1580 :: "ExtendStringPrototypeWithIterator" @266			4		64 0 %	64 0 %
1913 :: "TypedArraySetFromOverlappingTypedArray"			4		64 0 %	64 0 %
Retainers						
Object			Distance	Shallow Size	Retained Size	
▼ 10418 in (object properties)[] @62071			3	98 360 2 %	155 056 3 %	
▼ properties in @61191			2	520 0 %	204 976 3 %	
▼ builtins in @1231			1	56 0 %	344 448 6 %	
▼ global in @61191			2	520 0 %	204 976 3 %	
builtins @1231			1	56 0 %	344 448 6 %	
builtins in @61191			2	520 0 %	204 976 3 %	

一个Heap快照

[阅读更多了解如何提高JavaScript和CSS性能»](#)

审查存储

在资源面板中你可以查看被审查页面的资源情况。你可以对与 HTML5 Database, Local Storage, Cookies, AppCache等进行交互操作。



资源面板中显示的 [Web Starter Kit](#) 中的 *JavaScript* 文件

[阅读更多了解审查存储资源](#)

扩展阅读

DevTools 文档中还包括一些其他部分可能对你有所帮助：

- [Heap 分析](#)
- [CPU 分析](#)
- [设备模式 & 模拟设备](#)
- [远程调试](#)
- [DevTools 视频](#)

扩展资源

关注 [@ChromiumDev](#) 或去 [论坛](#) 提问。

开发 workflow

开发者 workflow 一般包含了几个步骤来实现特定的目标。使用 DevTools 可以优化你的 workflow 来节省完成任务的时间，例如定位文件或方法，连续的编辑脚本或样式，保存常用的代码片段，或者简单的重新布局来满足自己的需要。

在本章中，我们将会探索一些列建议来使你的 workflow 更加的高效。

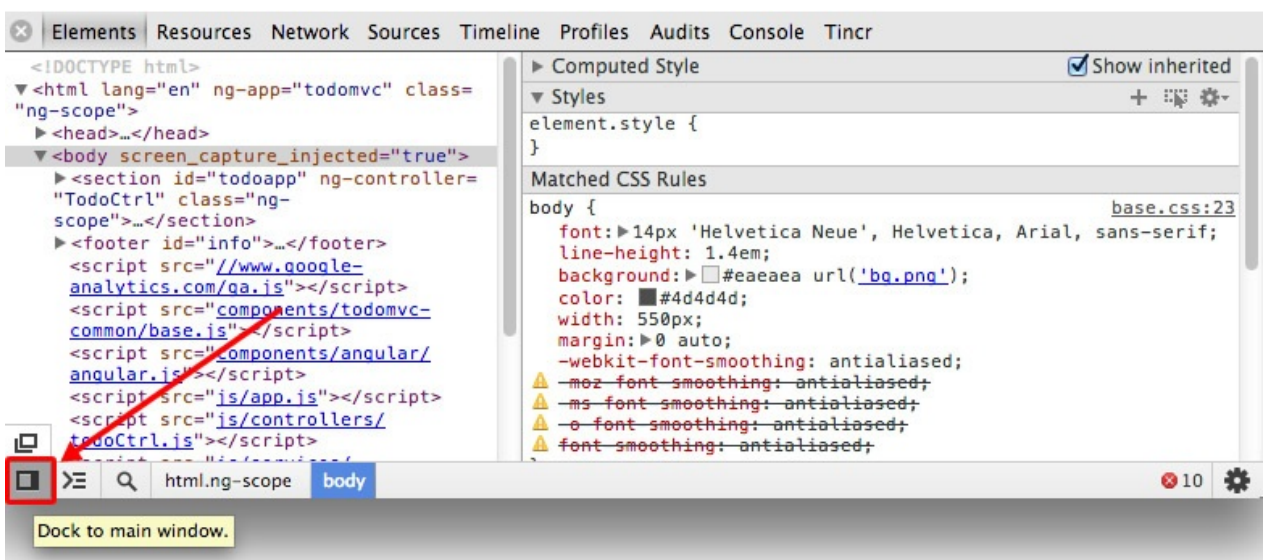
将 Dock 置于右侧进行垂直分屏编辑

你可能发现将 DevTools 放置于浏览器底部，水平空间大了，但是垂直空间比较小。你可以将 DevTools 附着在浏览器窗口的右侧，这样你可以在审查左边的页面，在右边进行调试。

右侧的 Dock 布局可能适用于下面的场景：

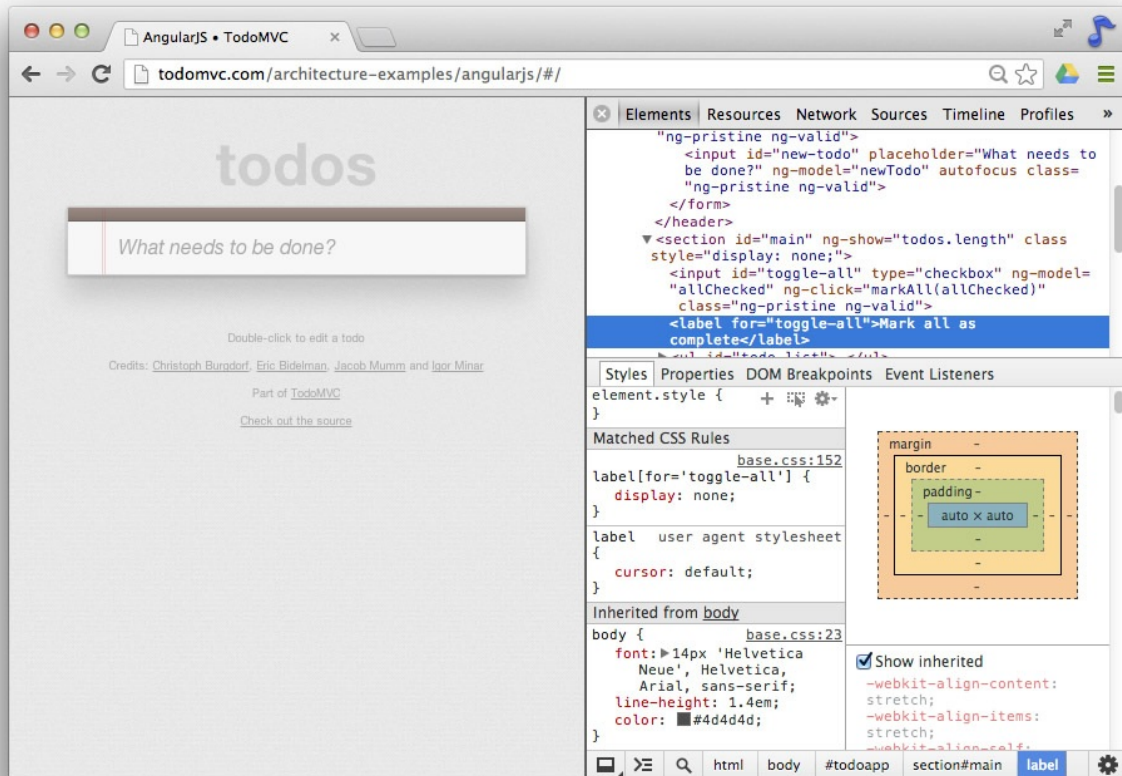
- 你可能在使用一个宽屏显示器，希望最大化的利用空间来审查和调试代码。
- 你可以将窗口拆分成小于 400px 的大小(目前 Chrome 浏览器的最小宽度)来测试布局的尺寸变化。
- 比较长的脚本在垂直空间里更容易调试

打开一个你希望调试页面的 URL，然后通过开关 `dock-to-right` 和 `dock-to-window` 来切换不同布局，或者长按下图中箭头所指示的图标，会显示所有可用的布局选项。



注意：DevTools 会记住你最后的选择，所以你可以来回的在常用选项之前进行切换。

一旦你选择了某项配置，布局变化会立刻生效。



注意: DevTools中每一个tab 都可以拥有自己的自定义布局。这意味着你可以使一个tab中的工具排在右侧, 而另一个tab的工具排在窗口底部。

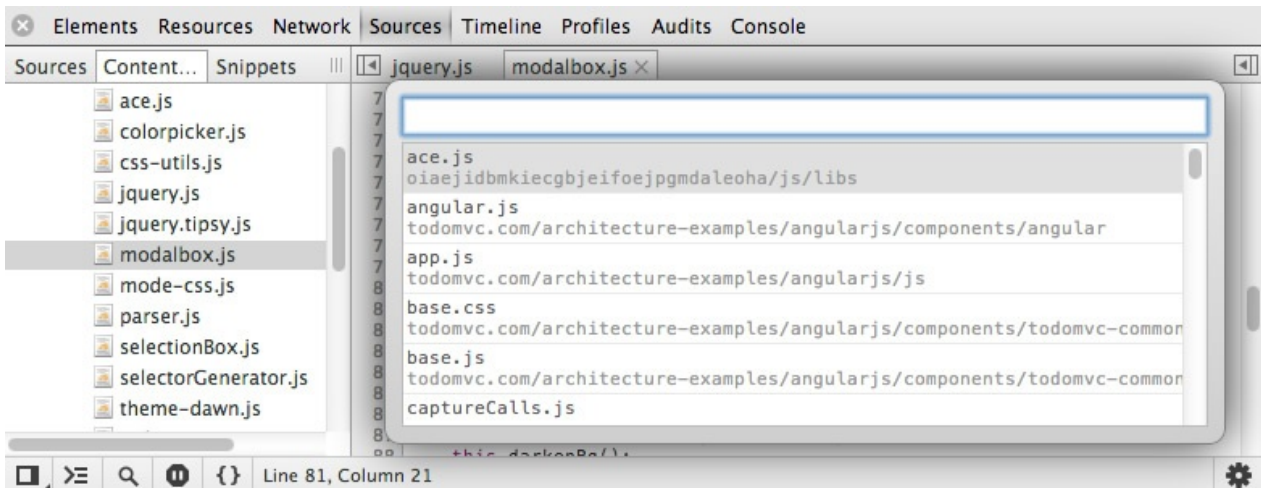
搜索, 导航和过滤器

脚本样式过滤或代码文件名检索

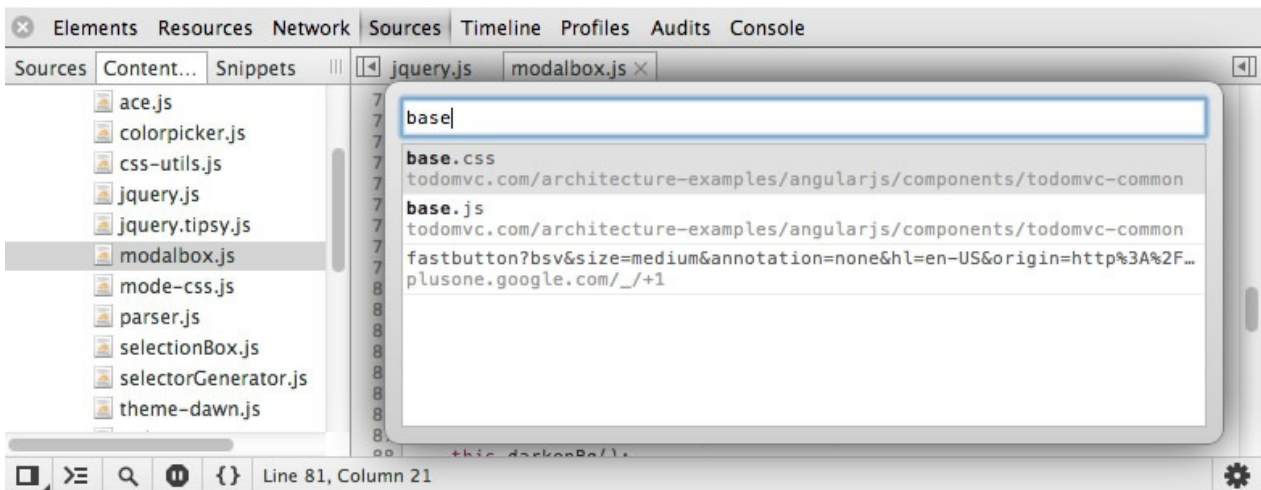
快速定位指定的文件是开发者工作流中不可获取的功能。DevTools支持在所有的脚本、样式表和代码文件中进行检索, 使用下面的快捷键:

- **Ctrl + O** (Windows, Linux)
- **Cmd + O** (Mac OS X)

无论当前在哪个面板, 都可以随时进行文件的检索。例如在这个Todo app中,使用上面的快捷键, 界面将会快速的切换到源码面板, 并且直接给我们列出所有可查看文件。



从这里，我们可以快速的向下选择特定的文件（例如名字中包含script的文件），进行查看或者编辑。



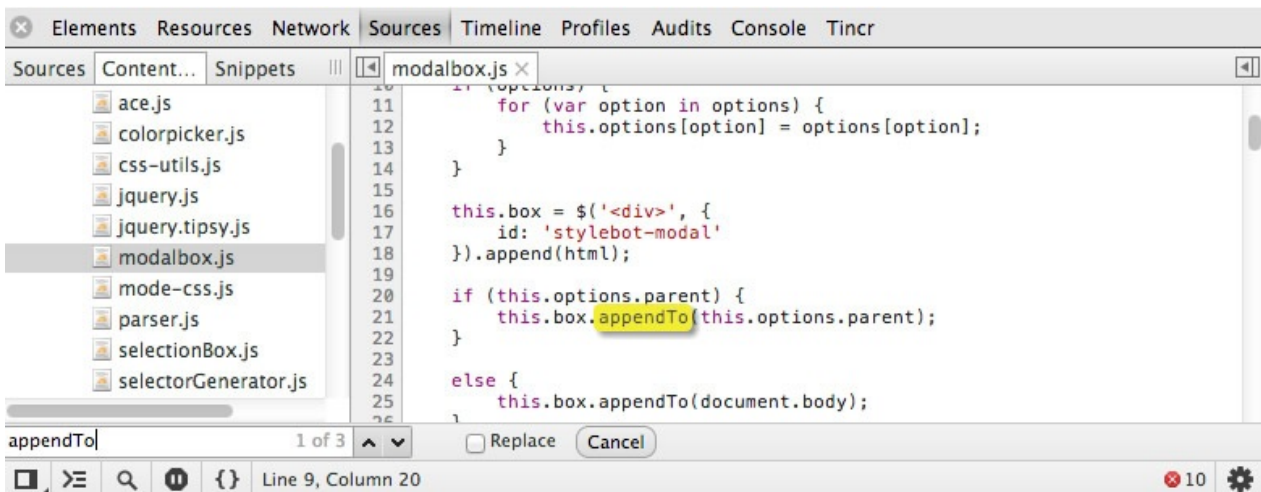
注意: 我们支持驼峰式书写的检索,例如，要打开FooBarScript.js, 你只需要输入FBaSc就可以了,这可以节约时间。

在当前文件内搜索

在指定文件内搜索一个特定的字符串，可以通过下面的快捷键：

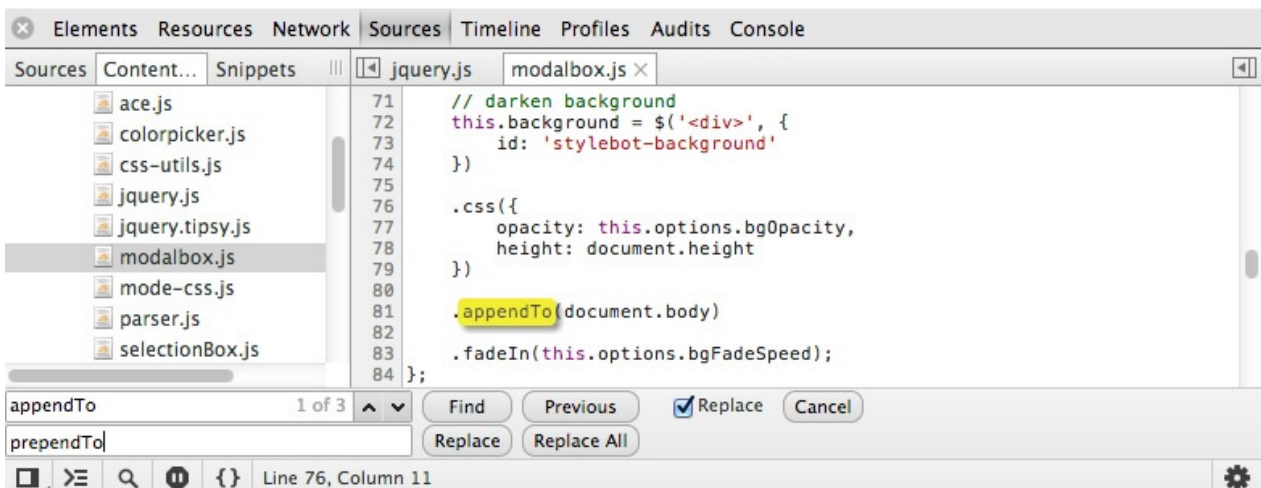
- **Ctrl + F** (Windows, Linux)
- **Cmd + F** (Mac OS X)

在搜索框输入完关键字后,直接敲回车定位到第一个匹配的结果,继续敲回车可以移动光标到下一个匹配结果。你可以通过上下的方向箭头在搜索结果之间进行移动。



在当前文件替换文本

除了支持文件内的文本定位之外，DevTools也支持单处和多处文本替换。勾选 **Replace** 后，窗口会显示第二个输入框来输入要替换内容的新数值。

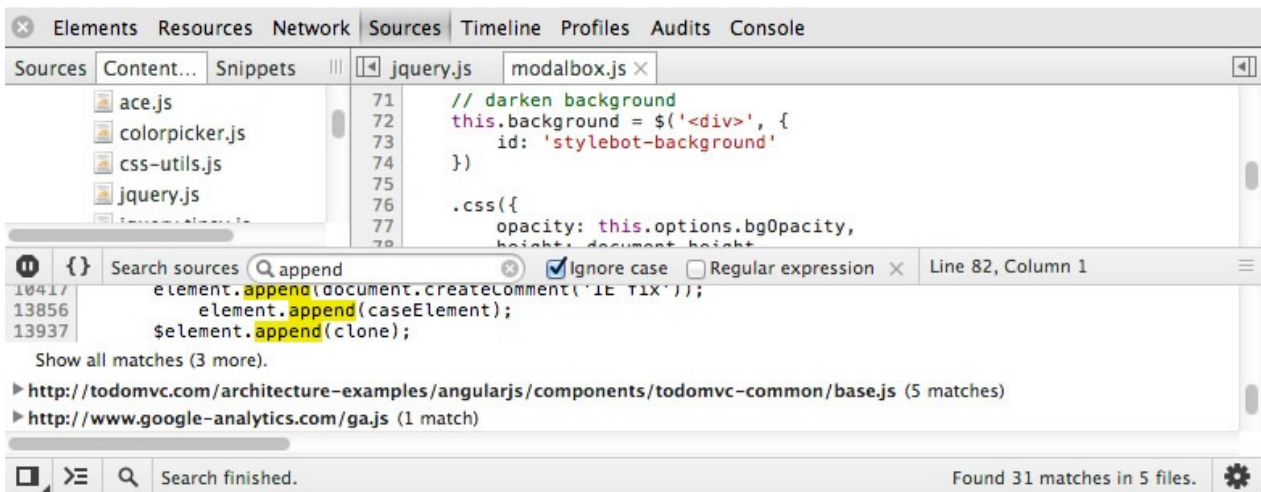


全局文本搜索

如果你希望在页面加载的所有文件中搜索一段特殊的字符串，你可以使用下面的快捷键进行全局搜索：

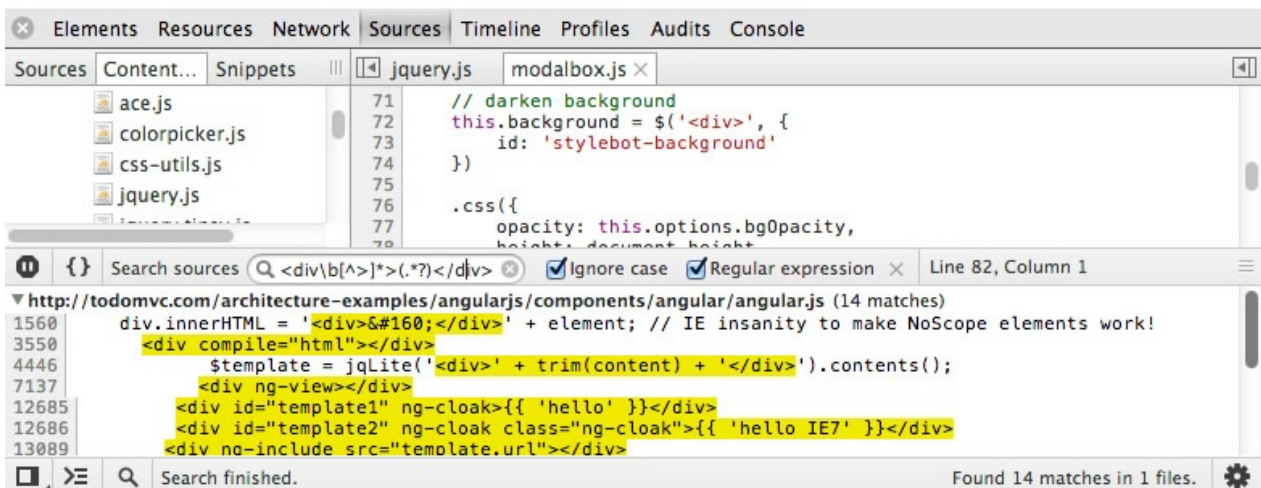
- **Ctrl + Shift + F** (Windows, Linux)
- **Cmd + Opt + F** (Max OS X)

搜索支持大小敏感搜索和正则表达式。



使用正则表达式搜索

要使用正则表达式进行搜索，只需要在搜索框中输入表达式，勾选 `Regular expression` 然后敲击回车。



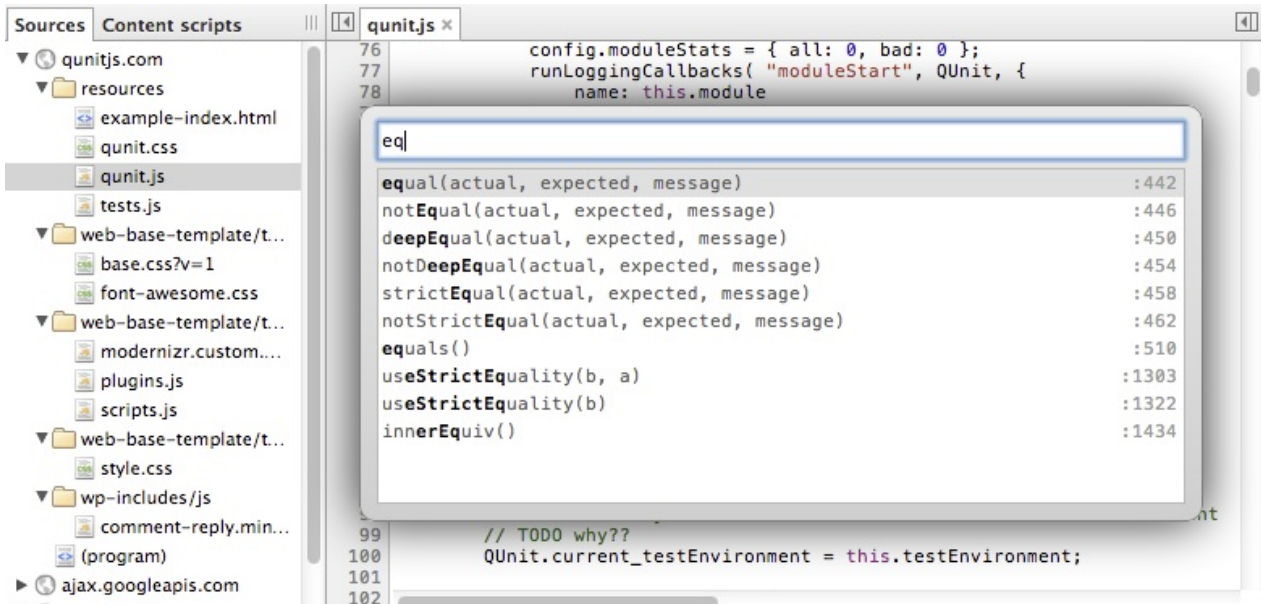
上图的例子中，我们看到如何用正则表达式找到 `<div></div>` 标签中的匹配内容。

函数过滤器或者文件内选择器

如果你需要更细的粒度，你可以定位到文件内特定的JavaScript方法或者CSS规则。

打开你选择的页面，然后打开源码面板。你可以使用下面的快捷键打开函数/选择器搜索窗口：

- `Ctrl + Shift + o` (Windows, Linux)
- `Cmd + Shift + o` (Mac OS X)

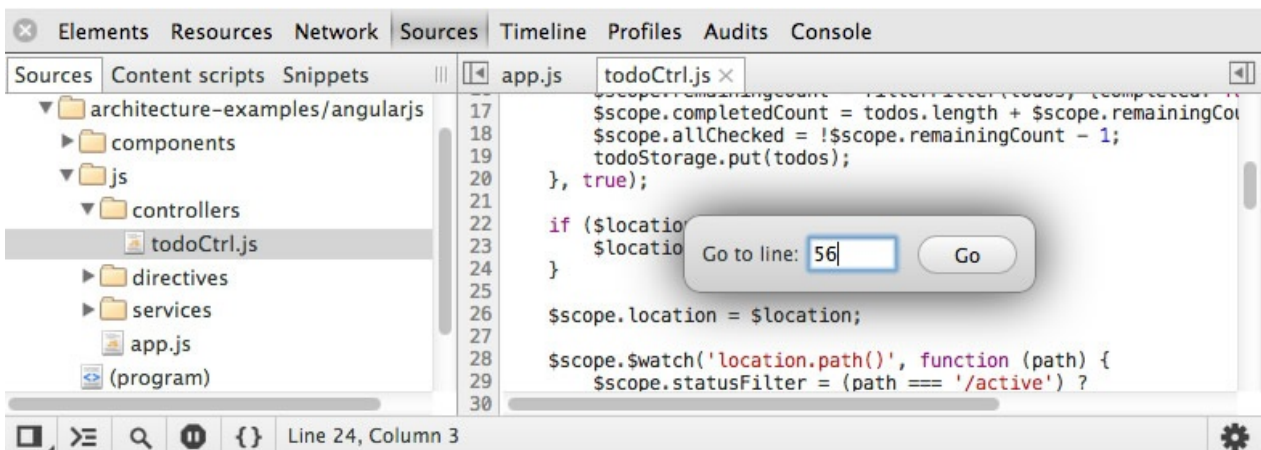


根据你选择的文件类型不同，你将看到相应的JavaScript函数或者CSS声明。直接输入一个函数或声明的名字就可以在下拉列表中看到，选择即可定位到函数或声明的定义的位置。

跳转到特定行号

DevTools也支持跳转到编辑器的特定行号中。选择一个文件进行编辑，使用下面的快捷键就可以打开跳转的对话框：

- `Ctrl + L` (Windows)
- `Cmd + L` (Mac OS X)
- `Ctrl + G` (Linux)



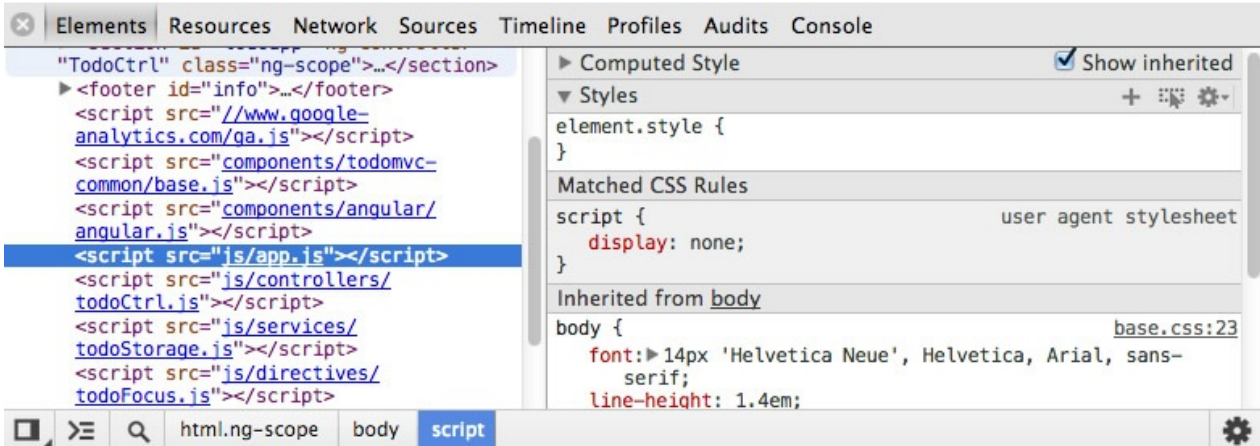
实时编辑脚本&样式

DevTools支持动态的实时编辑样式和脚本而不需要刷新整个页面。这个功能在测试设计的变更和编写脚本原型的时候比较有用。

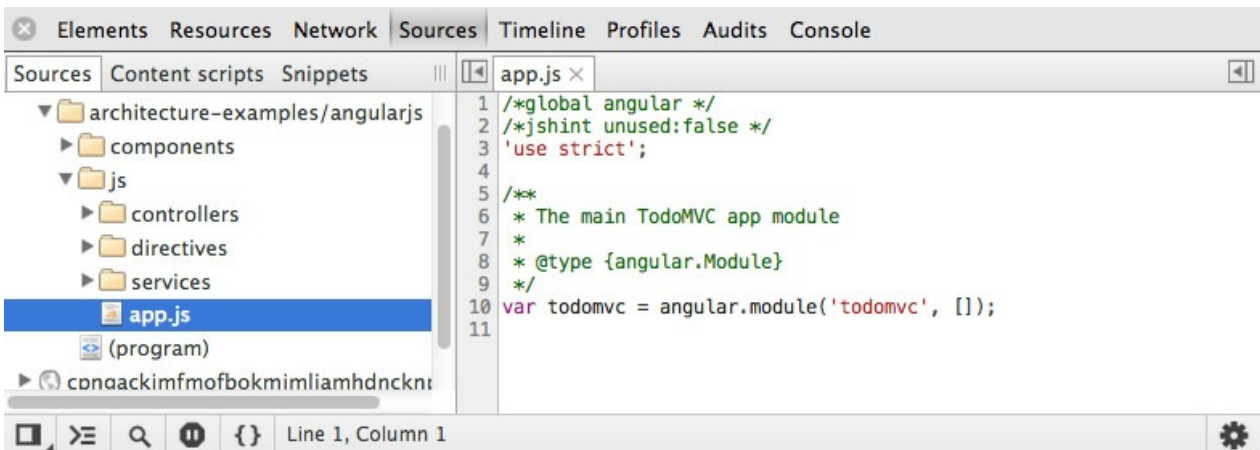
脚本

JavaScript代码可以直接在源码面板中进行编辑，要打开一个特定的脚本文件进行编辑：

1. 在元素面板中直接单击脚本的链接（例如 `<script src="app.js"></script>`）。



1. 或者在源码面板中从目录树中选择文件名。



选择文件后，在面板的右侧会在新的标签页中显示语法高亮的源码文件。

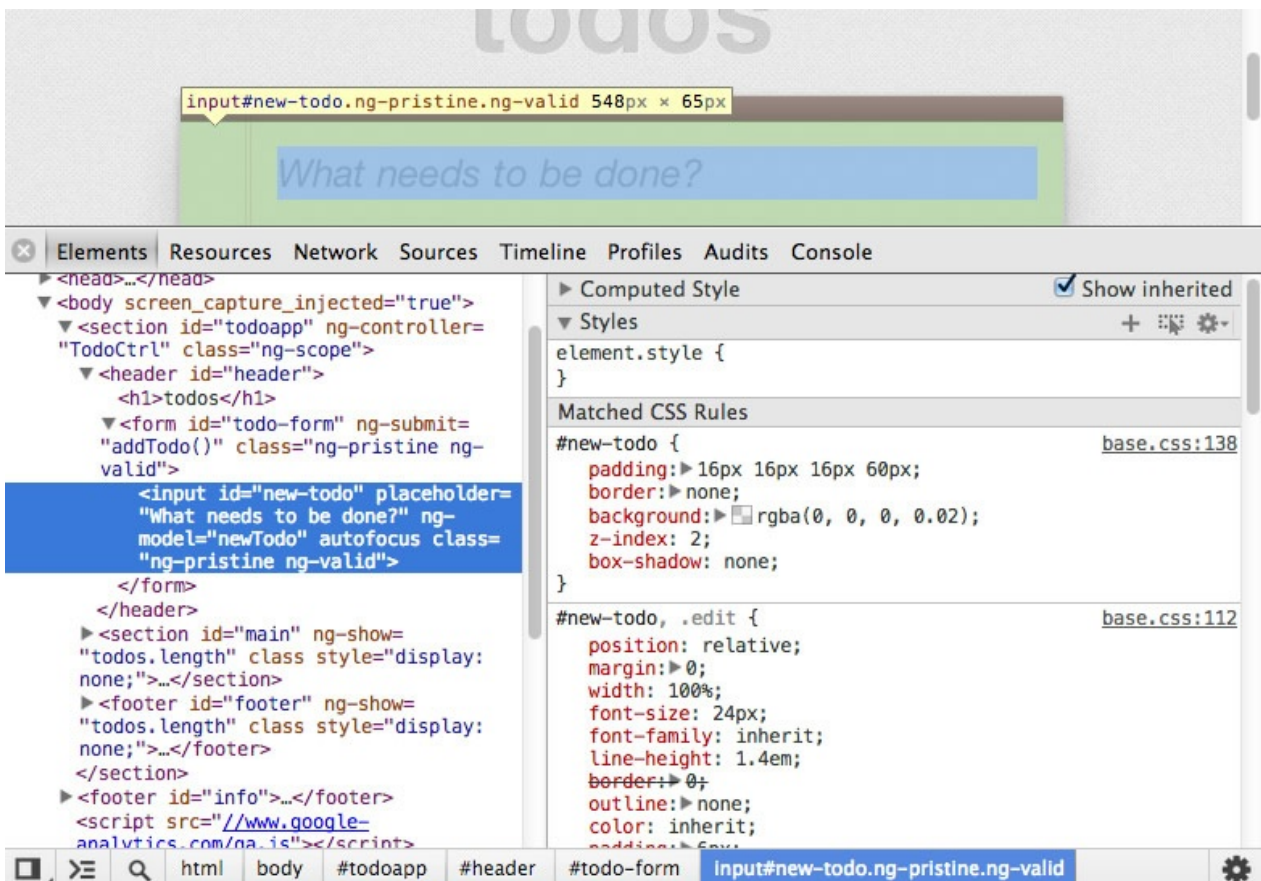
脚本的变更只会在`evaluation time`被执行，不在页面加载后执行的脚本代码发生变化是不会产生作用的。在后面会被执行的代码产生变化，例如鼠标移动处理函数或者单击回调函数是可以即可看到效果的。

要了解更过关于调试JavaScript代码的信息，可以阅读相关的[文档](#)。这里也有一段[录屏](#)来演示实时编辑和断点调试。

注意: Workspaces 特性已经支持本地文件的编辑，[了解更多](#)。

样式

DevTools也有编辑样式的工作流。打开元素面板，在界面的右侧的一些列子面板中可以看到样式。审查页面的元素时，当前节点的所有属性会被在此处列出来。

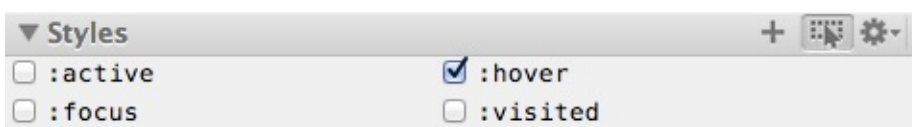


其中"element.style"部分显示了元素中style属性被预先设置的样式。

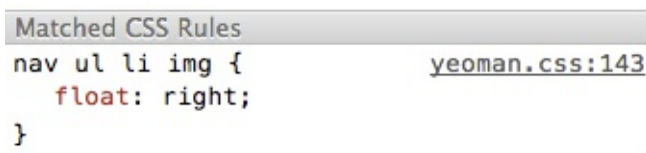
接下来的部分是"Matched CSS Rules", 这部分样式是成功匹配CSS选择器条件的样式, 对应的样式文件名以及具体的规则行号都会被列出来。匹配成功的选择器会被标为黑色, 而没有被匹配的会变灰色。这样不同的选择器就更容易区分了。

在子面板中改变CSS属性, 例如border或一个元素的大小, 都可以直接在浏览器窗口中看到实时的变化。

```
acronym, address, code, del, dfn, em, img, dl, dt, dd, c
ul, li, fieldset, form, label, legend, caption, tbody,
tfoot, thead, tr {
  margin: 0;
  padding: 0;
  border: 10px;
  font-weight: inherit;
  font-style: inherit;
  font-size: 100%;
  font-family: inherit;
  vertical-align: baseline;
}
```



回到"Matched CSS Rules" 面板, 点击CSS样式规则旁边的链接, 可以直接到源码面板。在这里显示整个的CSS文件, 并自动定位到规则所在的行号。

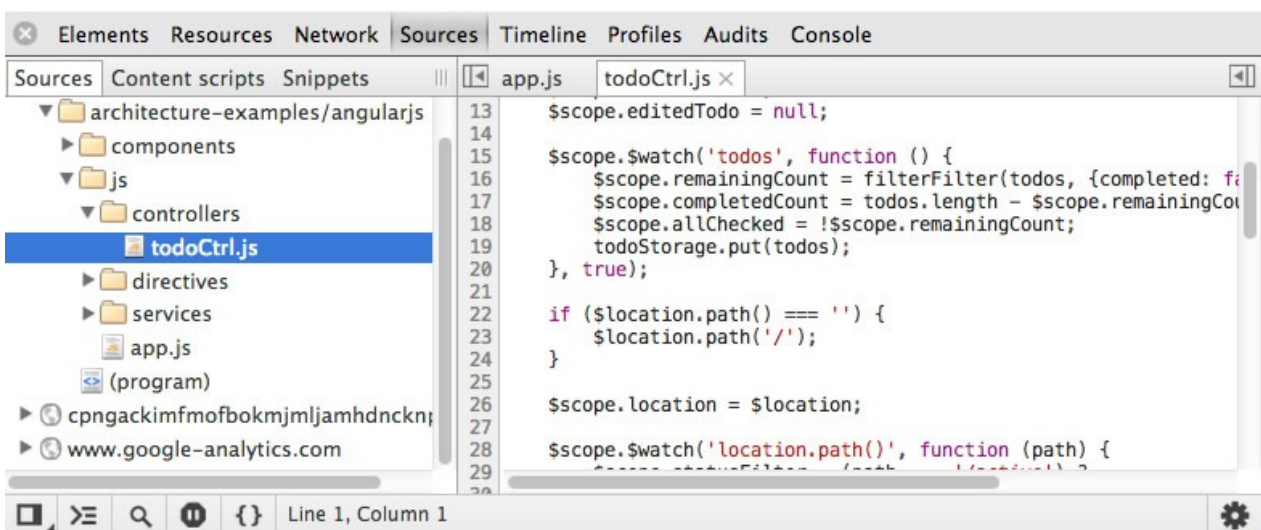


在这里，你可以像在普通编辑器里一样编辑文件，不同的是，这里可以看到实时的编辑效果。

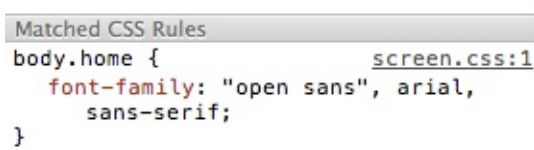
另存为

一旦你完成了修改，你可以直接将其保存。

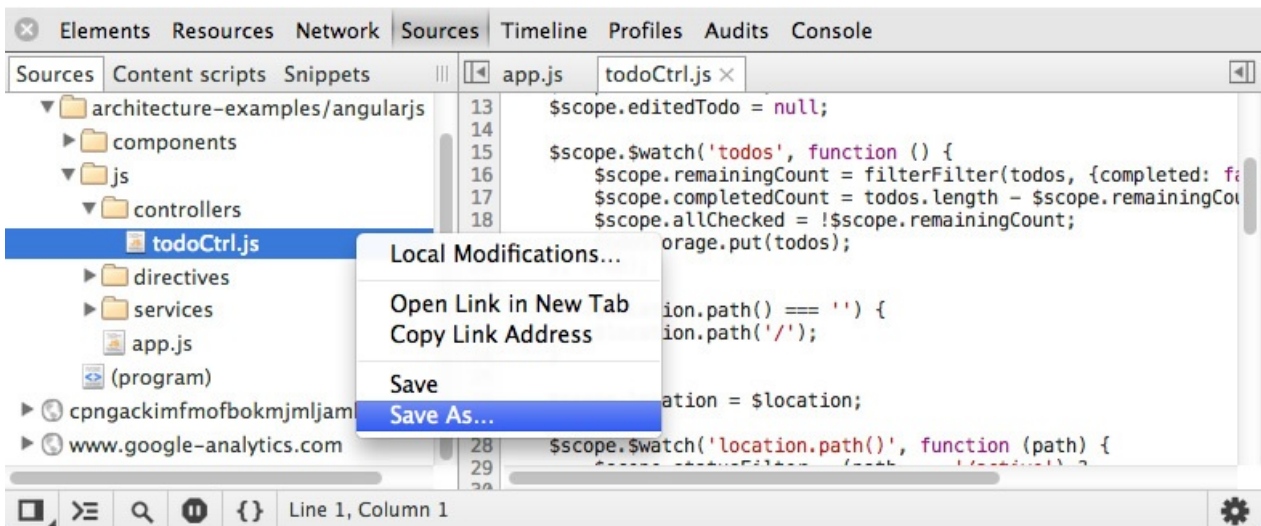
在保存文件前，确认已经打开了文件编辑的面板，无论是在源码面板的左侧目录树中打开：



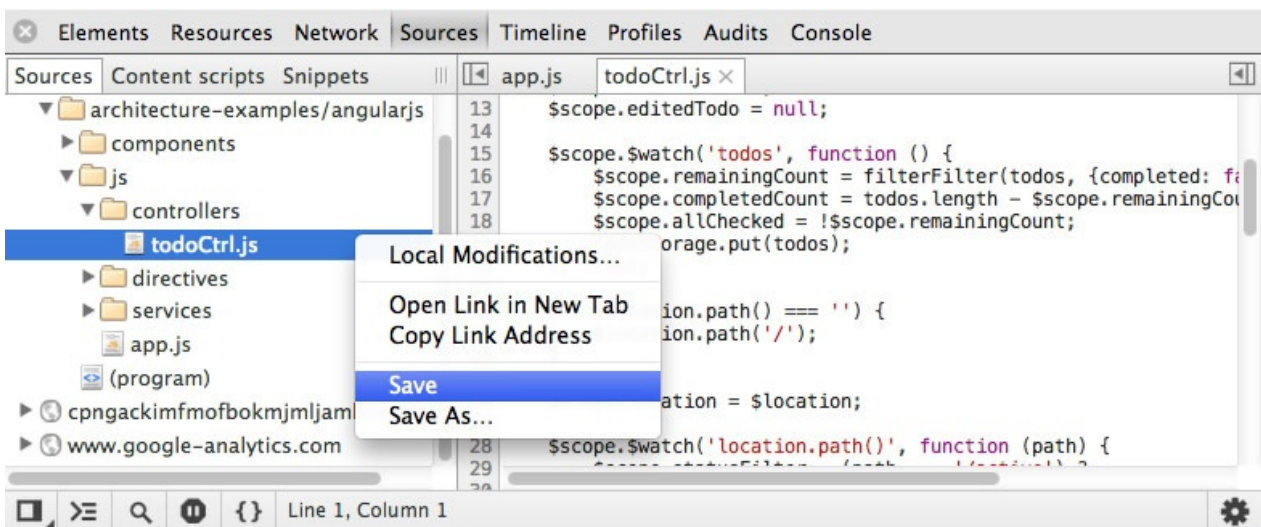
还是通过在"Elements -> Styles"面板中单击文件名：



接下来，在左侧目录树对应的文件名或者编辑器里右键选择另存为。在出现的对话框中可以选择覆盖原文件。

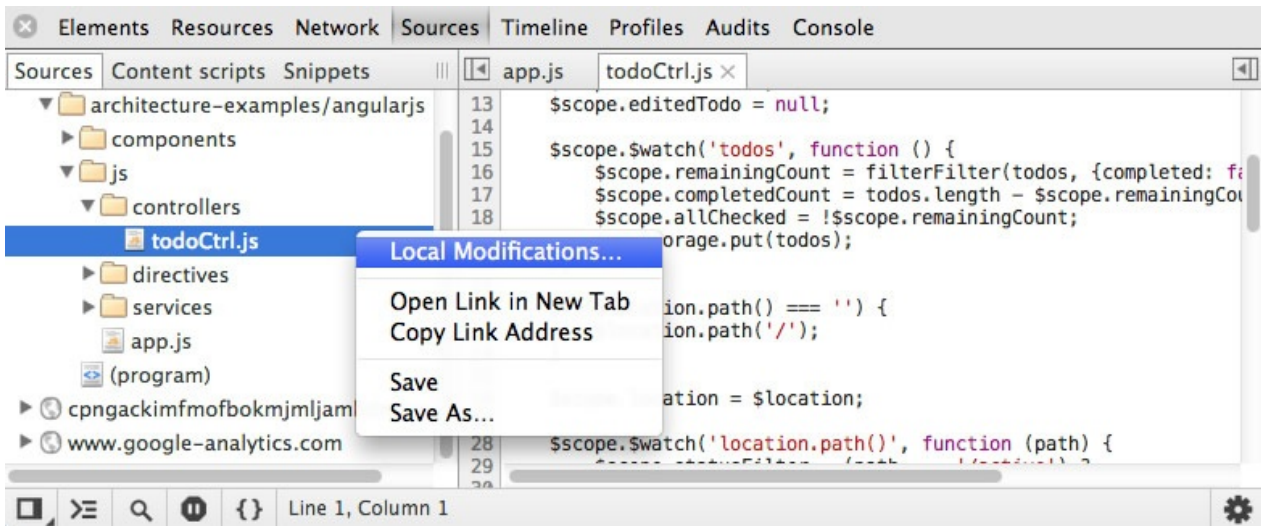


后面保存会默认写在同样的位置。



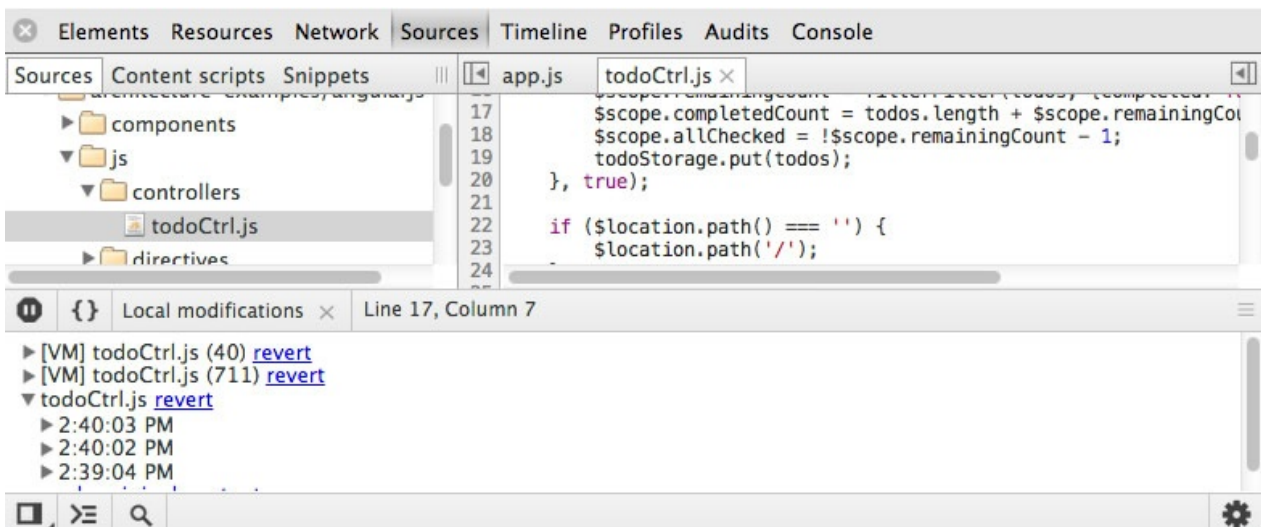
本地修改

DevTools也会保存本地代码修改的版本历史。如果你已经在Chrome中编辑了脚本或者样式，你可以在文件名右键，选择"Local modifications"来查看修改历史。

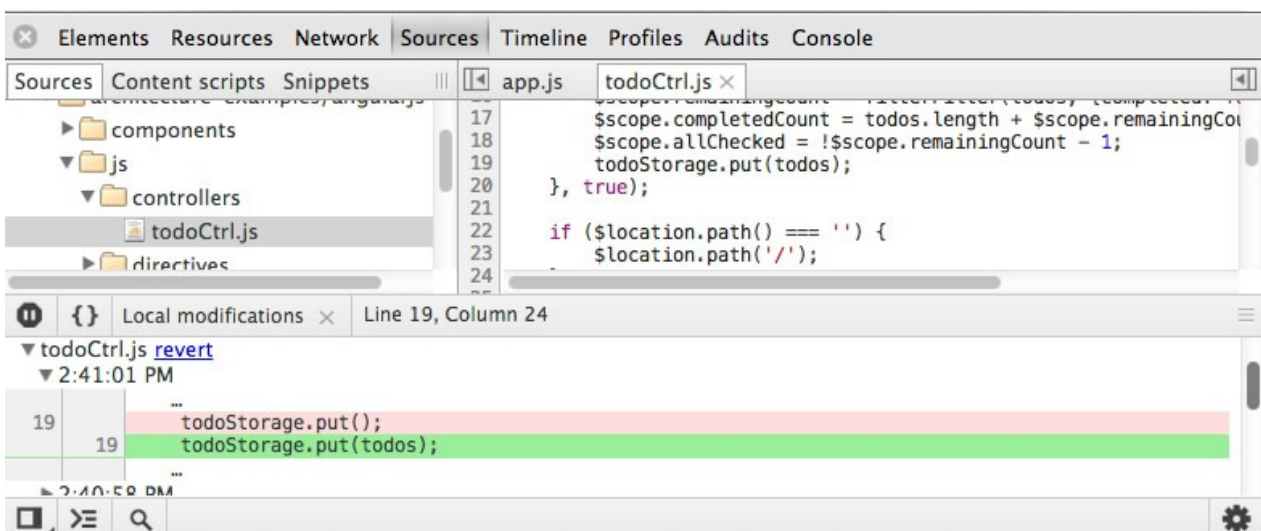


Local modifications 面板显示的信息包括:

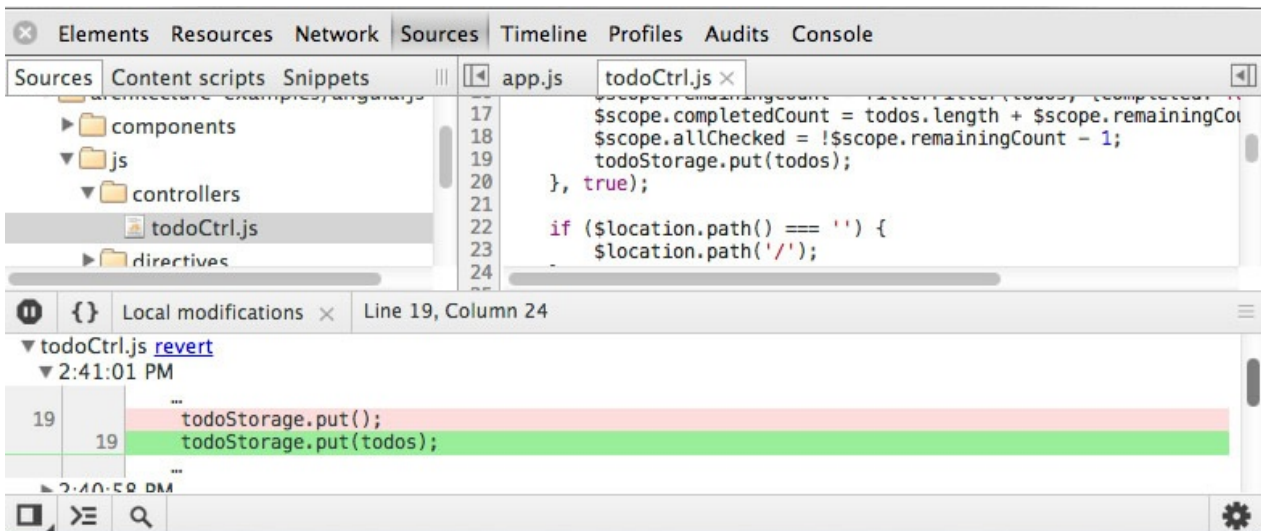
- 代码修改的差异
- 修改的时间
- 被修改文件所属的域



除此之外，点击 **revert** 链接将会把代码恢复到最初的状态，并移除修改历史。



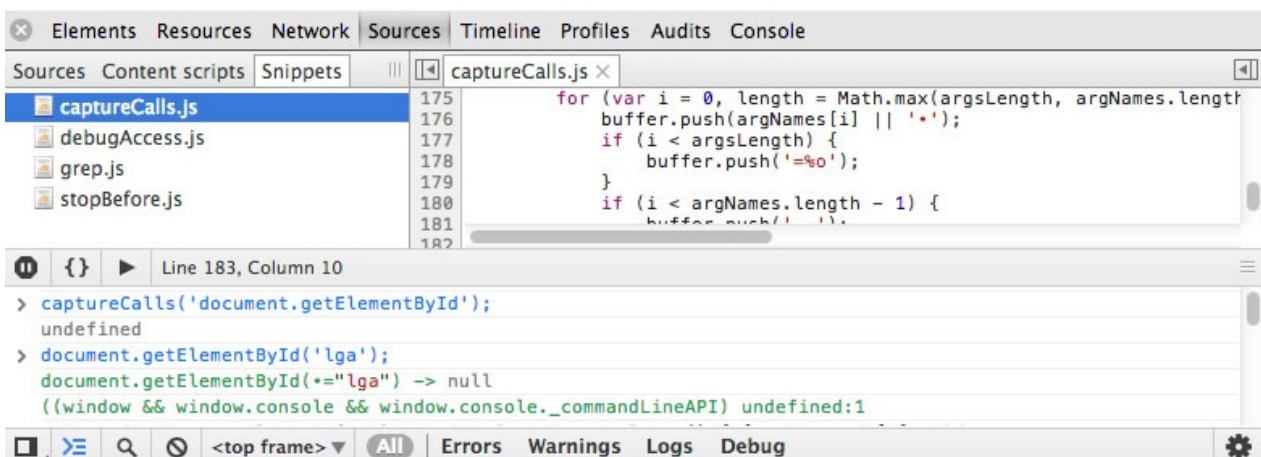
*Apply original content*可以实现同样的功能，但是有时候你可能需要回退到某个特定的版本，因此有必要维护一个代码修改历史。



最后"apply revision content"会采用某个特定时间修改的版本。

定制 JavaScript 代码片段

有时候你可能会希望保存一些小的代码片段、书签和工具以备在浏览器调试代码的时候使用。代码片段是DevTools中的新特性，允许你在源码面板中创建、保存并运行JavaScript代码。该功能目前在[Chrome Canary](#)可用。



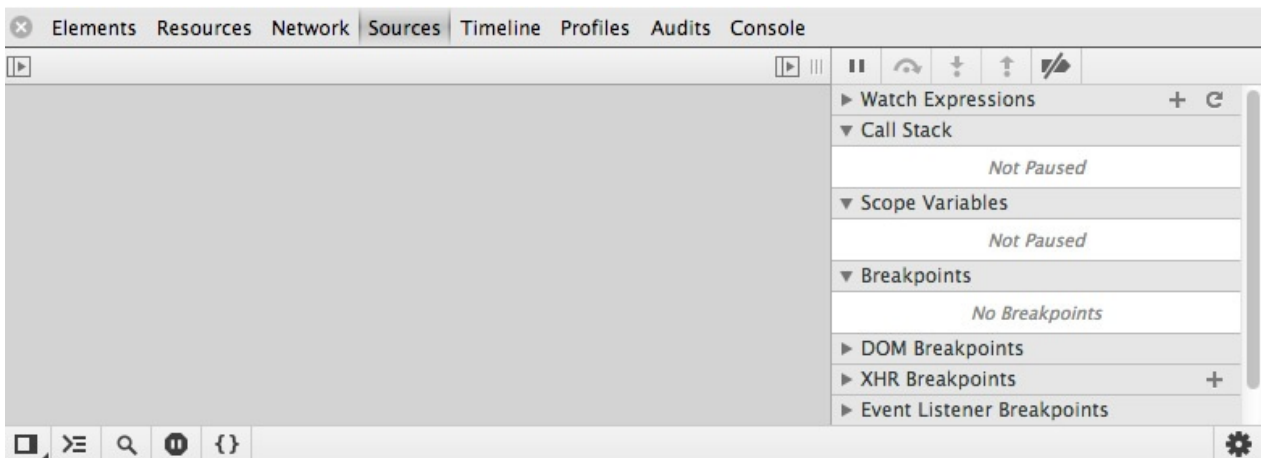
一些代码片段的应用场景包括：

- 书签 - 你所有的书签都可以存储为代码片段，特别是那些将来希望编辑的。
- 工具 - 调试页面的工具可以存储起来。目前社区维护的工具[列表](#)。
- 调试 - 代码片段提供一个多行支持语法高亮的控制台，使得调试代码更加方便。
- Monkey-patching code - 代码片段支持你在运行时添加代码,尽管大多数时间里你只需要在源码面板中实时修改代码。

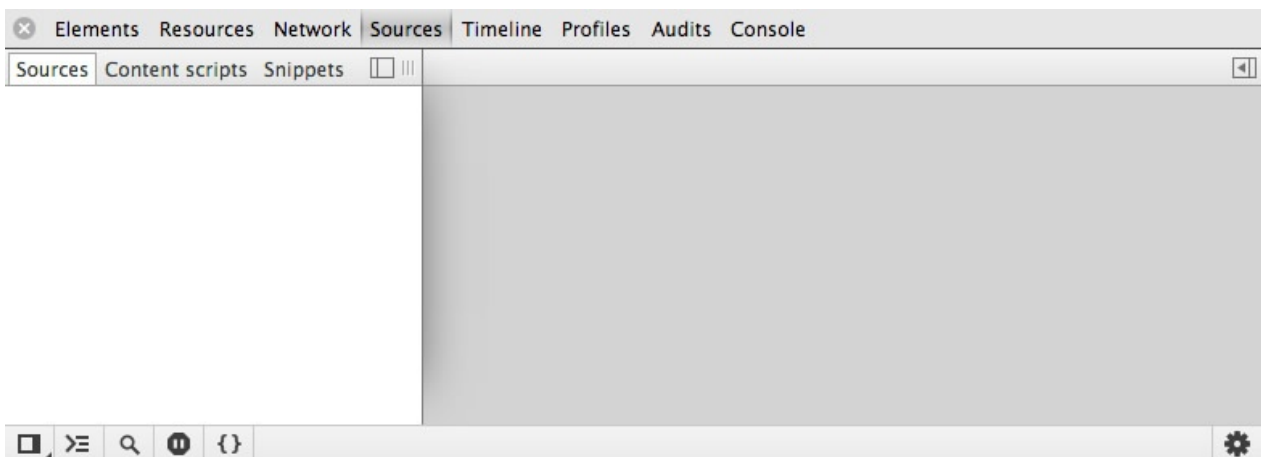
Brian Grinstead 在Github上维护了一个代码片段的项目bgrins.github.io/devtools-snippets。

开始

要开始使用代码片段，先打开源码面板。如果你还没做其他的操作，其界面应该如下图所示：

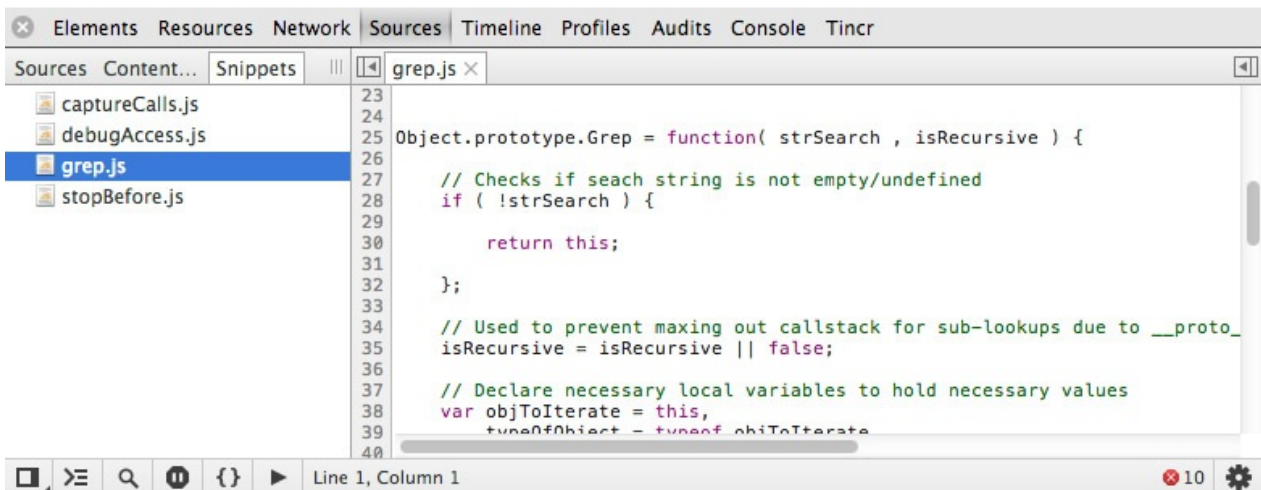


点击左上角的布局按钮打开扩展面板。在新窗口中可以看到 Sources 、 Content scripts 和一个新的标签页 Snippets 。点击 Snippets ：

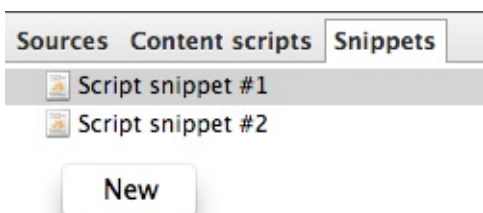


创建 Snippets

Snippets 包含两个面板，左侧的为文件目录树，右侧为snippet的具体内容。编辑snippet的体验和在源码面板中操作体验很类似。



在左侧的文件列表中，右键选择 **New** 可以创建新的snippet。

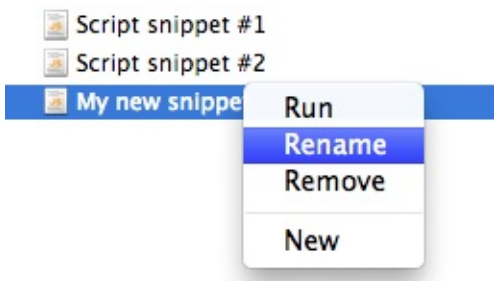


Snippet 文件名

Snippet 文件名是自动生成的，不过你也可以在其创建的时候自定义。

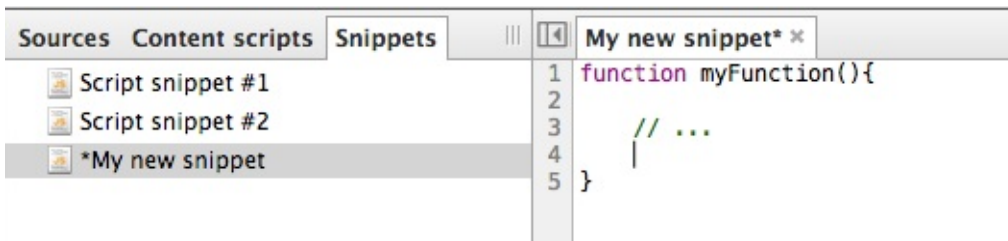


如果你希望对文件重命名，直接在文件名上右键 **Rename**。选择 **Remove** 删除snippet。



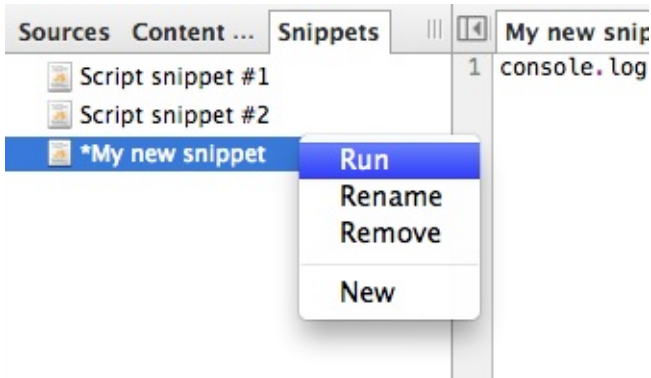
编辑和执行 Snippets

从左侧文件列表中选择一個 snippet 打开，在右侧编辑窗口中可纯编写或粘贴任何JavaScript代码，包括函数和表达式。



如果一个snippet文件名前面有一个 * 符号，表示这个文件有修改还没有保存。

要运行一个snippet，右键snippet文件名，选择 `Run` .也可以单击 `Run (>)` 按钮。

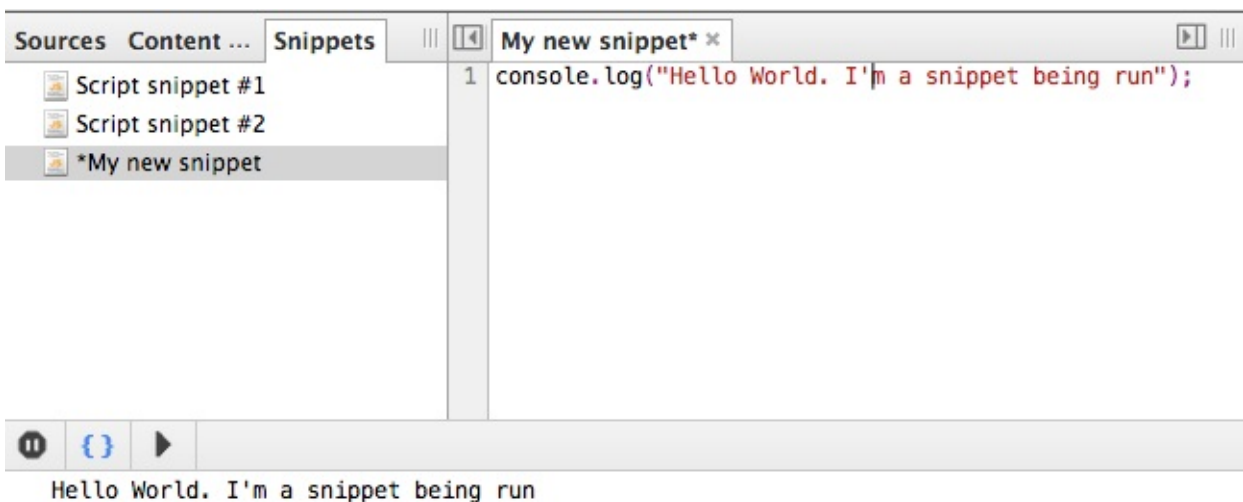


snippet的输出将会显示在编辑器下方的终端

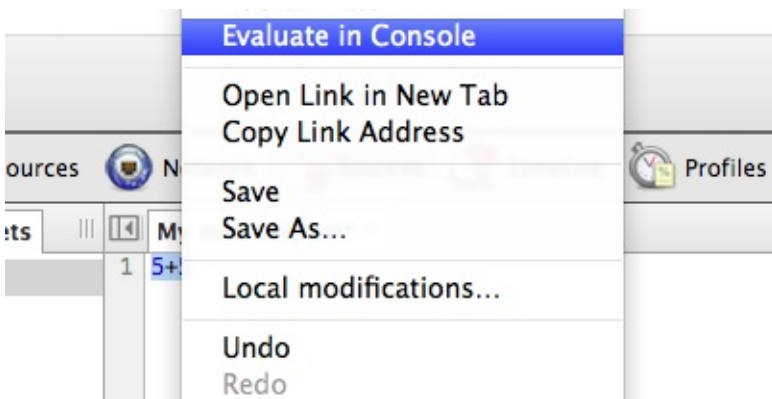
中。

注意: 你也可以使用快捷键来运行一段snippet - 只需要选择对应的snippet，然后使用

`Ctrl/Cmd + Enter` 运行. 这样可以取代运行按钮，这个按钮将会从控制台底部移到调试控制中。



如果你想在终端中执行snippet中的几行，你可以选中目标代码，右键选择 `Evaluate in Console` 来执行。其快捷键为 `Ctrl + Shift + E` 。



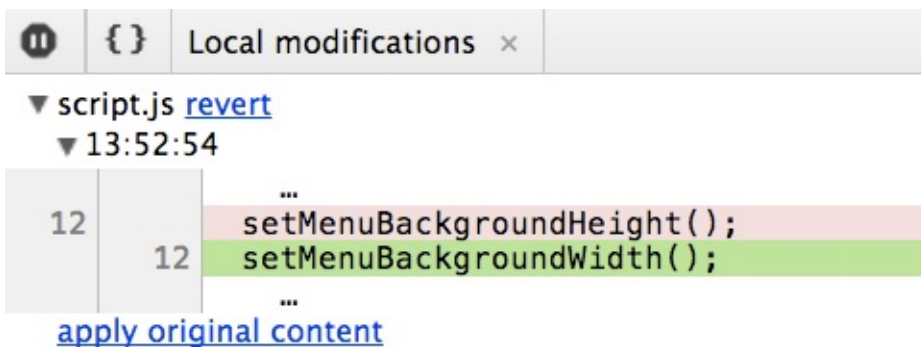
代码运行后,表达式的输出结果会显示在编辑器下面的终端窗口中。



本地修改

如同源码编辑一样，Snippets同样支持查看本地修改历史，并回退到某个历史版本。

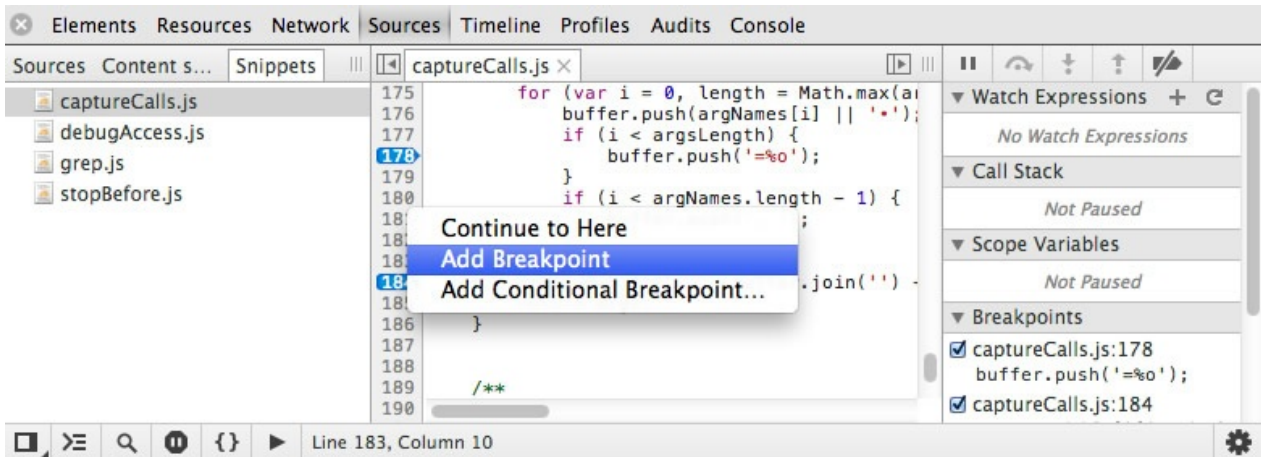
在编辑器中右键，选择 `Local modifications...`



断点调试，观察表达式以及更多

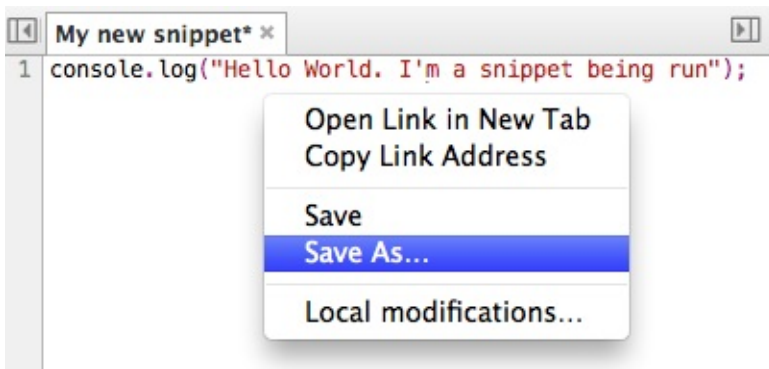
你在源码面板中使用的一些特性，例如添加观察表达式，调试断点，作用域变量，和保存文件在snippet中同样可用。

阅读源码面板的相关的文档来了解这些特性。



保存 Snippets

Snippets 可以直接在浏览器保存以便以后访问，也可以导出到文件系统中。在编辑器中右键可以查看保存选项。

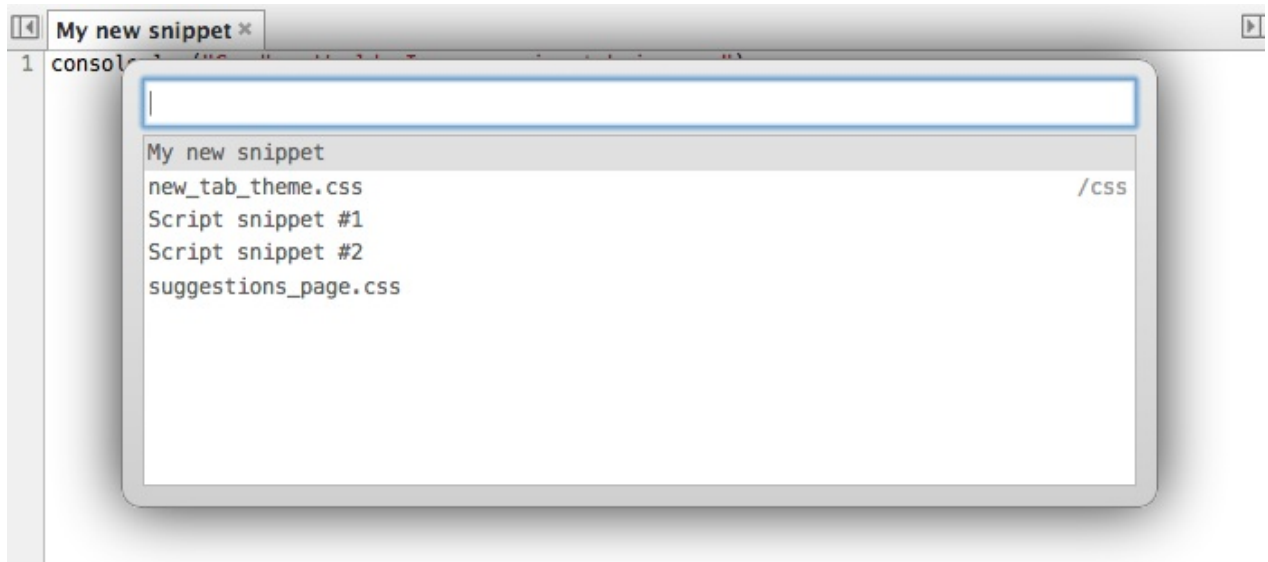


保存功能将会持续的覆盖已存在的 snippet 文件，另存为功能可以将 snippet 存储为其他位置的新文件。

注意: Snippets 存储在 DevTools 的 localStorage. 当使用另存为的时候，你可以像存储脚本一样将其存成文件。

导航 Snippets

与源码面板中的脚本和CSS类似，Snippets 也支持快速的文件定位。



资料

- [Chrome DevTools Revolutions 2013: Workspaces](#)
- [My workflow: Never having to leave the DevTools | remy sharp](#)
- [The Breakpoint With Addy Osmani And Paul Lewis - Snippets | youtube](#)
- [Chrome Dev Tools: JavaScript and Performance | nettuts](#)
- [Iterating with the Chrome DevTools | jeremey kahn](#)

使用控制台

利用控制台可以做如下的事情：

- 日志诊断信息帮助你调试web应用。
- 输入命令和页面或DevTools进行交互

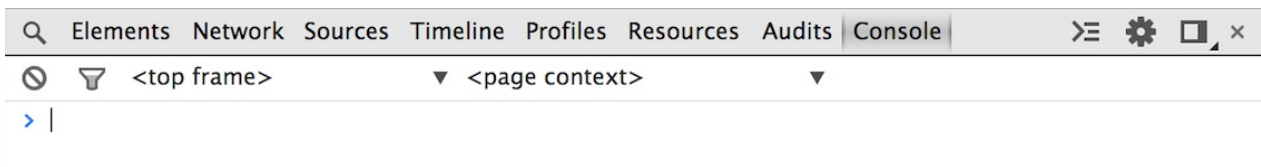
你要可以在里面执行普通的表达式。本文将介绍控制台的概要以及基本的一些使用。你可以浏览[Console API](#)和[Command line API](#)参考手册了解更多的功能。

基本操作

打开命令行

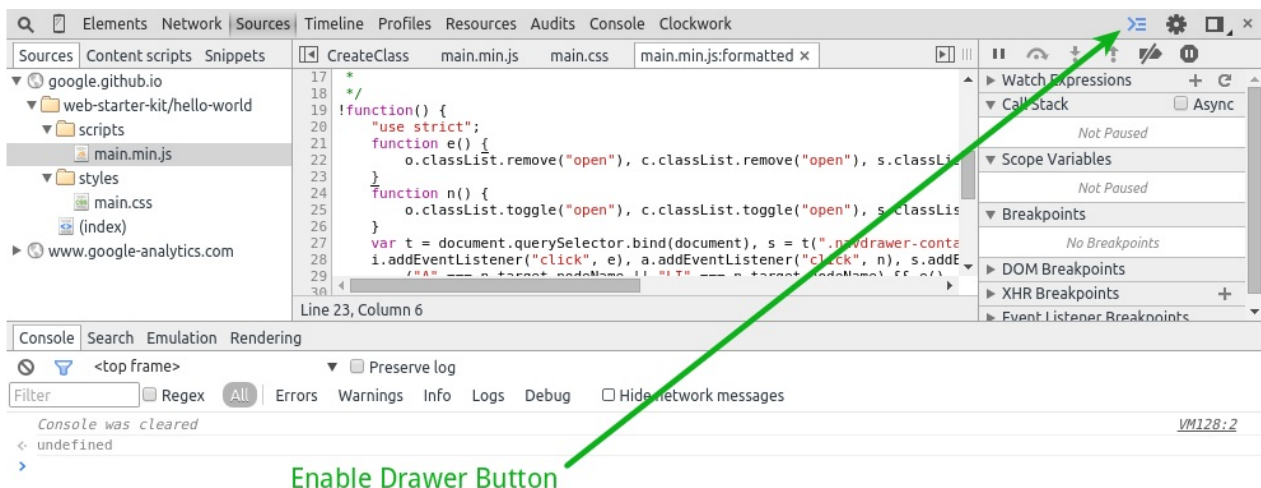
JavaScript 控制台可以通过两种方式打开。常用的方式是打开控制台面板,当然你可以在其他面板中随时点击右上角的控制台图标弹出抽屉窗口：

- 使用快捷键 `Command + Option + J` (Mac) 或者 `Control + Shift + J` (Windows/Linux)。
- 选择Chrome菜单 > 更多工具 > JavaScript 控制台。



一个清空的抽屉界面

要打开控制台，可以按 `Esc` 键或单击DevTools窗口右上角的显示控制台按钮。



在元素面板中打开的控制台界面

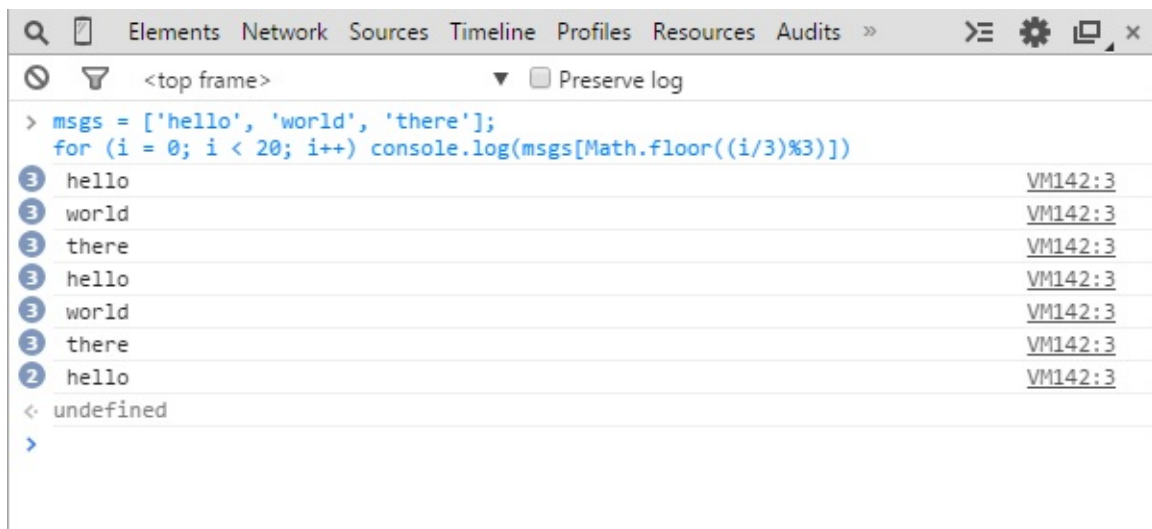
清空控制台历史记录

可以通过下面的方式清空控制台历史：

- 在控制台右键，或者按下 `Ctrl` 并单击鼠标，选择 `Clear Console`。
- 在脚本窗口输入 `clear()` 执行。
- 在JavaScript脚本中调用 `console.clear()`。
- 使用快捷键 `Cmd + K`，`⌘ + L` (Mac) `Ctrl + L` (Windows and Linux)。

消息栈

控制台会默认的折叠连续输出相同内容的消息，这样使得输出尽可能的保持简洁。



默认情况下,隐藏时间戳，折叠消息



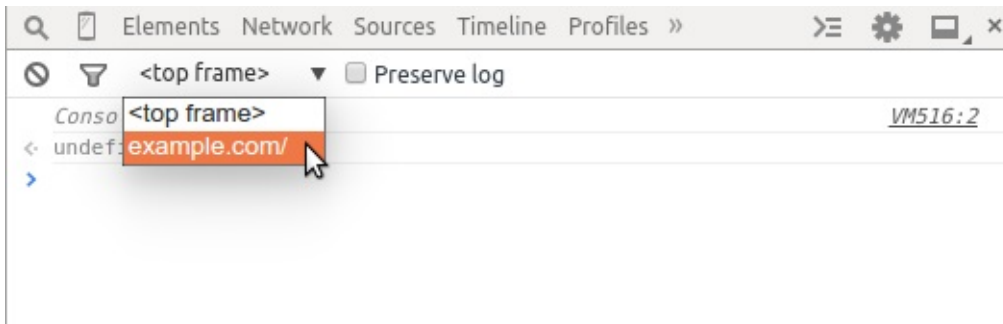
取消消息折叠(译注：该功能可以在`DevTools`设置中修改，点击右上角齿轮图标)

下面的代码可以用来测试折叠消息的效果

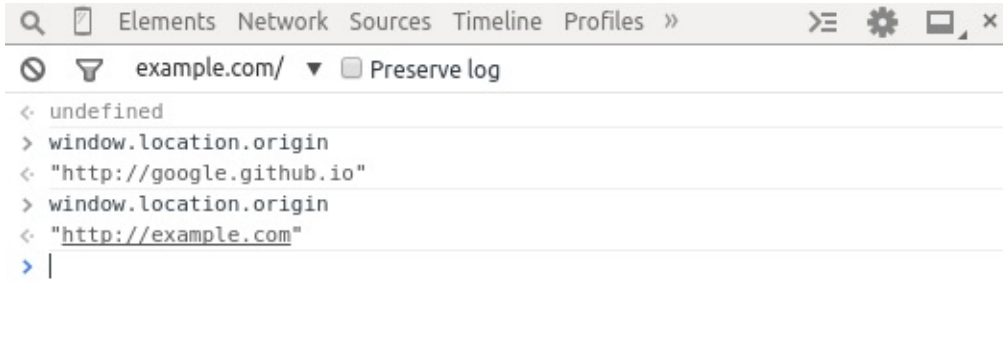
```
msgs = ['hello', 'world', 'there'];
for (i = 0; i < 20; i++) console.log(msgs[Math.floor((i/3)%3)])
```

Frame 选择

控制台支持在不同的frame中进行操作。主页面默认为top frame。一个 `iframe` 元素会创建自己的上下文环境，你可以从下拉列表中自由的选择frame。



选择一个二级 *frame*



上图显示了再切换 *frame* 时，*window origin* 属性的变化

使用 Console API

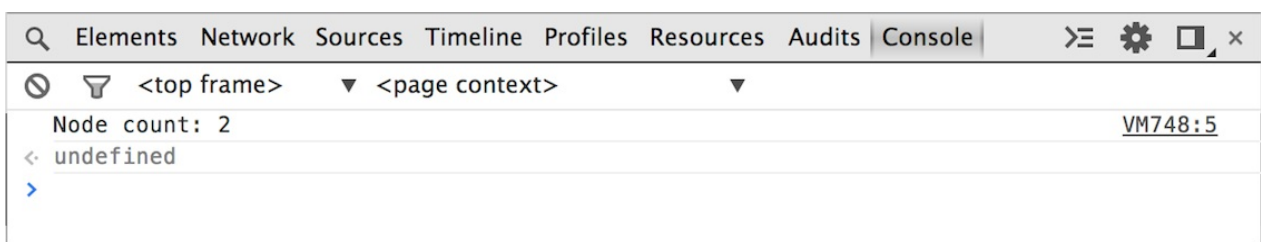
Console API 就是DevTools定义的全局变量 `console` 对象的方法集合。API的主要目的在于当你的程序运行时为你记录日志信息。你还可以对输出进行分类，使日志看起来更加清晰。

向控制台输出

`console.log()` 方法可以接受一个或多个表达式作为参数，并将他们的值打印到控制台。

下面一段代码演示了基本的使用：

```
var a = document.createElement('p');
a.appendChild(document.createTextNode('foo'));
a.appendChild(document.createTextNode('bar'));
console.log("Node count: " + a.childNodes.length);
```

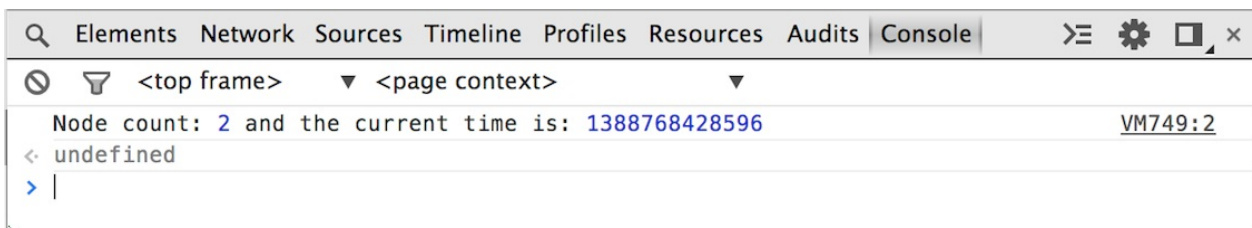


上面代码的输出结果

当有多个参数的时候，结果会拼接成连续的一行。

下面代码为 `console.log()` 接受多个参数：

```
console.log("Node count:", a.childNodes.length, "and the current time is:", Date.now());
```



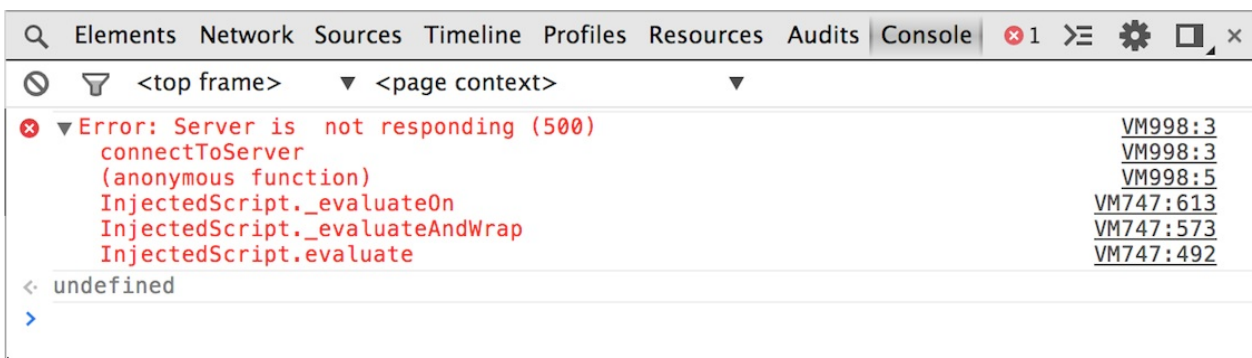
多参数的输出

错误和警告

错误和警告信息和普通输出一样使用。区别在于其输出的样式不同，`console.error()` 打印的结果是红色的，`console.warn()` 打印的信息是黄色的。

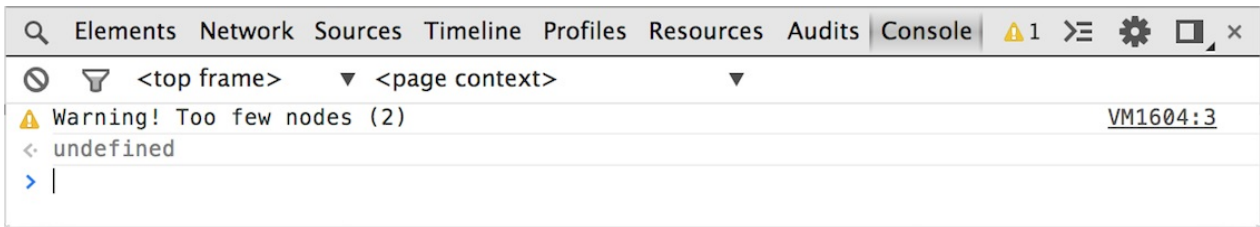
`error()`使用代码及结果

```
function connectToServer() {  
    console.error("Error: %s (%i)", "Server is not responding", 500);  
}  
connectToServer();
```



`warn()`使用代码及结果

```
if(a.childNodes.length < 3 ) {  
    console.warn('Warning! Too few nodes (%d)', a.childNodes.length);  
}
```

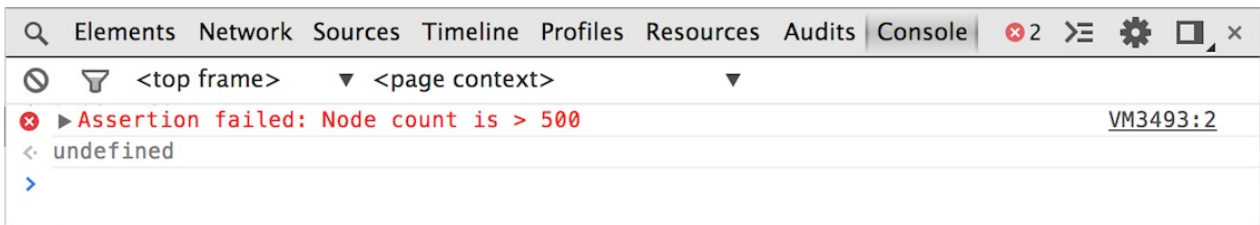


断言

`console.assert()` 接受两个参数，如果第一个参数的结果为 `false`，则该方法会将第二个参数输出到控制台。

下面的代码会判断`list`元素的子节点数目是否超过500，如果超过则打印一条错误消息：

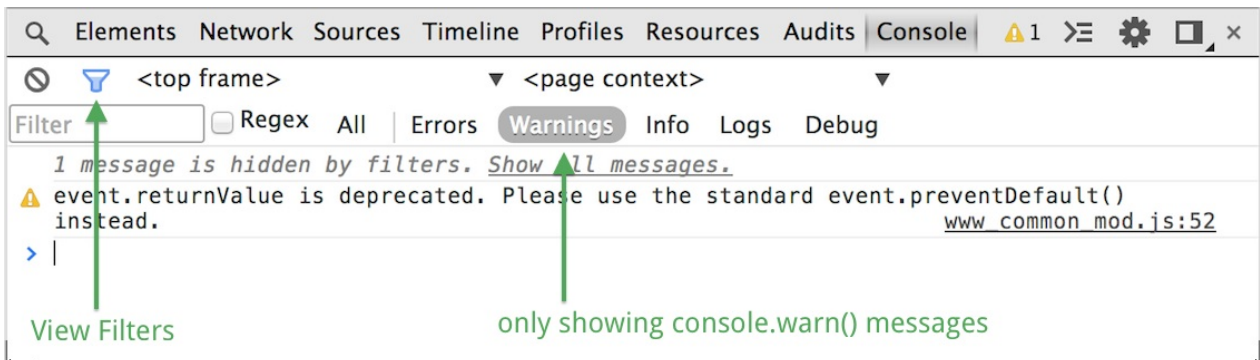
```
console.assert(list.childNodes.length < 500, "Node count is > 500");
```



过滤输出

你可以根据级别对日志输出进行过滤。点击界面左上角的漏斗形状的图标即可对日志进行过滤,目前支持的日志级别包括：

- All - 显示所有的输出
- Errors - 只显示 `console.error()` 的输出
- Warnings - 只显示 `console.warn()` 的输出
- Info - 只显示 `console.info()` 的输出
- Logs - 只显示 `console.log()` 的输出
- Debug - 只显示 `console.timeEnd()` 和 `console.debug()` 的输出

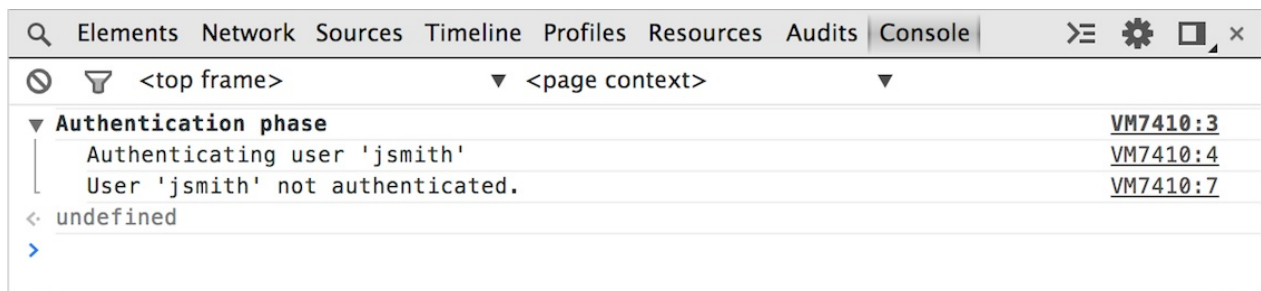


分组输出

你可以使用`group`命令对相关的输出进行分组。`group`方法只接受一个字符串参数，表示分组的名称。控制台会把接下来所有的输出组合输出，调用`groupEnd()`可以结束当前分组。

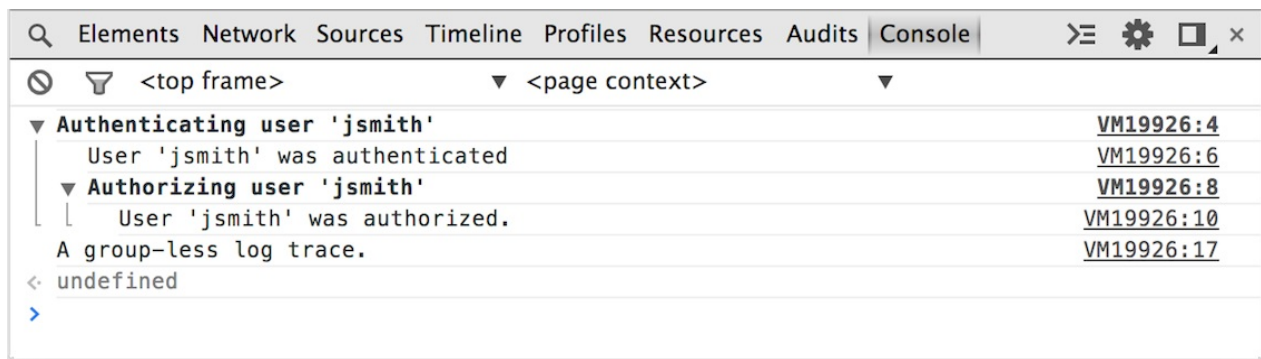
下面的代码演示了如何进行分组输出：

```
var user = "jsmith", authenticated = false;
console.group("Authentication phase");
console.log("Authenticating user '%s'", user);
// authentication code here...
if (!authenticated) {
    console.log("User '%s' not authenticated.", user)
}
console.groupEnd();
```



日志分组是可以相互嵌套的，当分组信息较多的时候比较有用，下面代码显示了分组嵌套的例子：

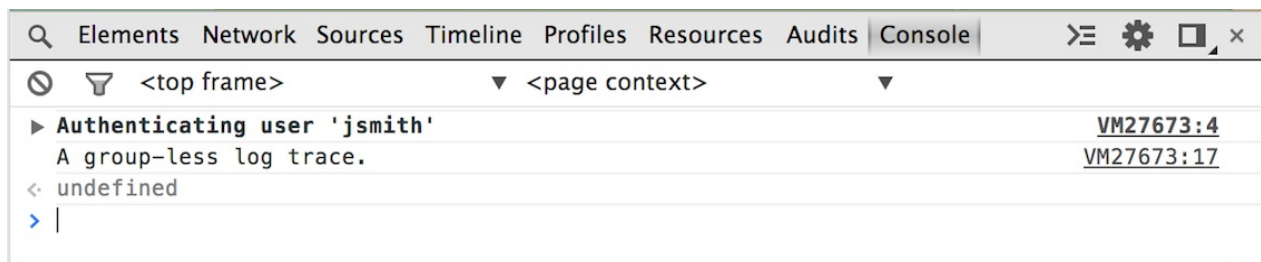
```
var user = "jsmith", authenticated = true, authorized = true;
// Top-level group
console.group("Authenticating user '%s'", user);
if (authenticated) {
    console.log("User '%s' was authenticated", user);
    // Start nested group
    console.group("Authorizing user '%s'", user);
    if (authorized) {
        console.log("User '%s' was authorized.", user);
    }
    // End nested group
    console.groupEnd();
}
// End top-level group
console.groupEnd();
console.log("A group-less log trace.");
```



当日志量较大的时候，可能并不需要看到所有的日志信息，这时推荐使用`groupCollapsed()`方法，该方法会默认折叠同组的信息。

代码示例：

```
console.groupCollapsed("Authenticating user '%s'", user);
if (authenticated) {
  ...
}
console.groupEnd();
```

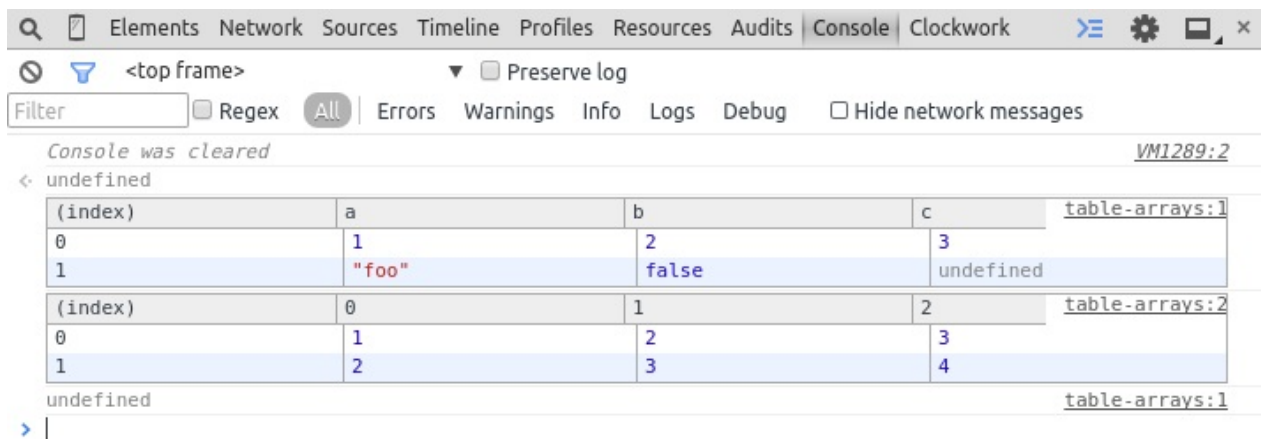


查看数据结构

`table()` 方法可以用来查看对象的内容。该方法会将对象的属性以表格的形式打印出来。

下面代码显示了打印两个数组的内容：

```
console.table([a:1, b:2, c:3], {a:"foo", b:false, c:undefined});
console.table([[1,2,3], [2,3,4]]);
```



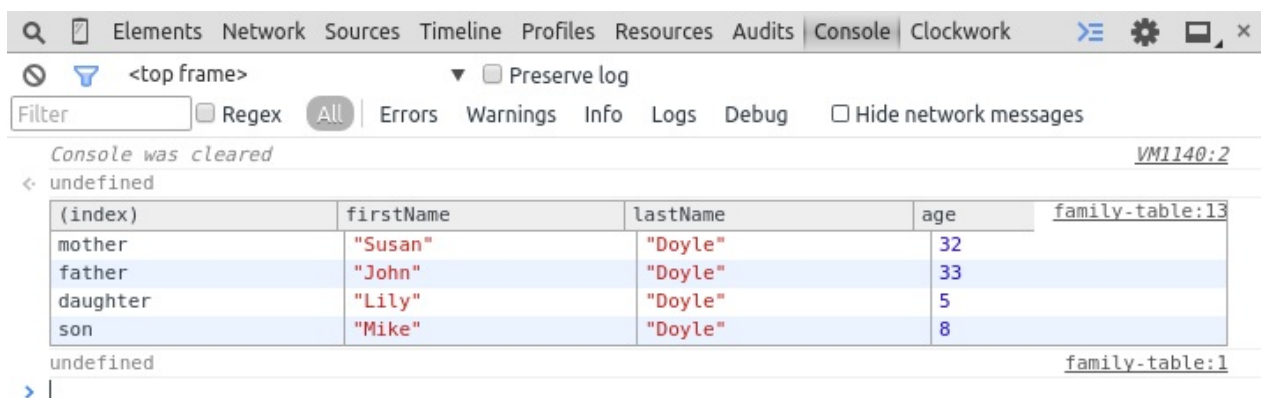
`table()` 的第二个参数是可选的，你可传入一个字符串数组，来指定想显示的属性。

示例代码如下：

```
function Person(firstName, lastName, age) {
  this.firstName = firstName;
  this.lastName = lastName;
  this.age = age;
}

var family = {};
family.mother = new Person("Susan", "Doyle", 32);
family.father = new Person("John", "Doyle", 33);
family.daughter = new Person("Lily", "Doyle", 5);
family.son = new Person("Mike", "Doyle", 8);

console.table(family, ["firstName", "lastName", "age"]);
```



字符串裁剪和格式化

日志输出一系列方法的第一个字符串参数都可以包含一个或者多个格式符。格式符由 `%` 后面跟一个字母组成，字母代表不同的格式。每个格式符与后面的参数一一对应。

下面的例子中，日志打印了结果中包含了字符串格式和数字格式内容。

```
console.log("%s has %d points", "Sam", 100);
```

The full list of format specifiers are as follows:

- `%s` - 字符串格式
- `%i` 或 `%d` - 整型格式
- `%f` - 浮点格式
- `%o` - DOM节点
- `%O` - JavaScript 对象
- `%c` - 对输出的字符串使用css样式，样式由第二个参数指定。

下面的例子使用整型来打印 `document.childNodes.length` 的值，使用浮点格式来打印 `Date.now()` 的值。

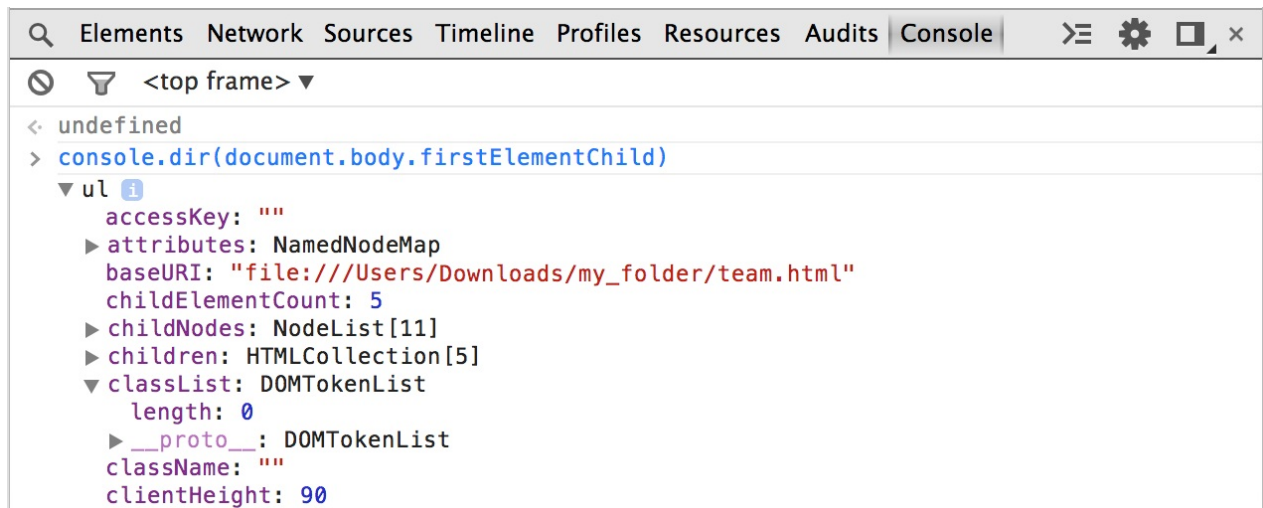
```
console.log("Node count: %d, and the time is %f.", document.childNodes.length, Date.now());
```

将DOM与元素格式化为JavaScript对象

当你打印一个DOM元素的时候，它会以XML的格式显示，正如在元素面板中看到的那样。如果要以JavaScript对象的形式显示，你可以使用 `dir()` 方法或者在 `log()` 中制定格式符。



默认情况打印DOM元素



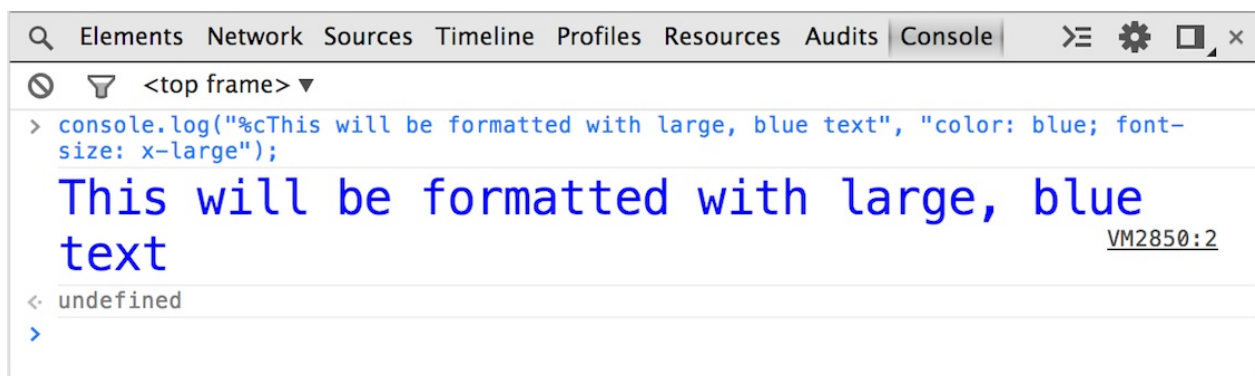
使用dir打印

使用CSS增加控制台样式

CSS格式符允许你自定义控制台输出的样式。第二个参数可以制定具体的样式。

代码演示：

```
console.log("%cThis will be formatted with large, blue text", "color: blue; font-size: x-large");
```

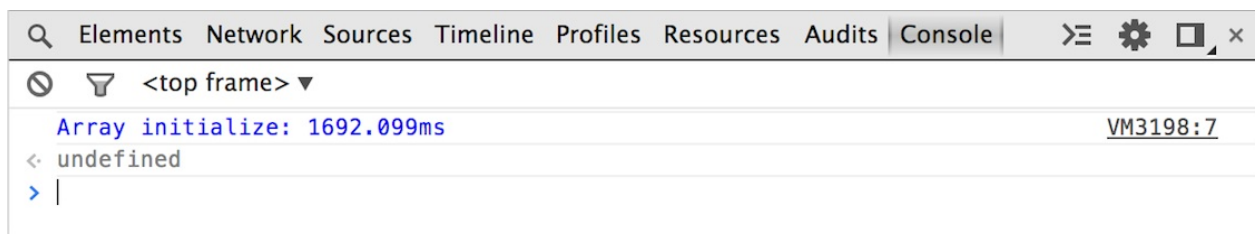



耗时监控

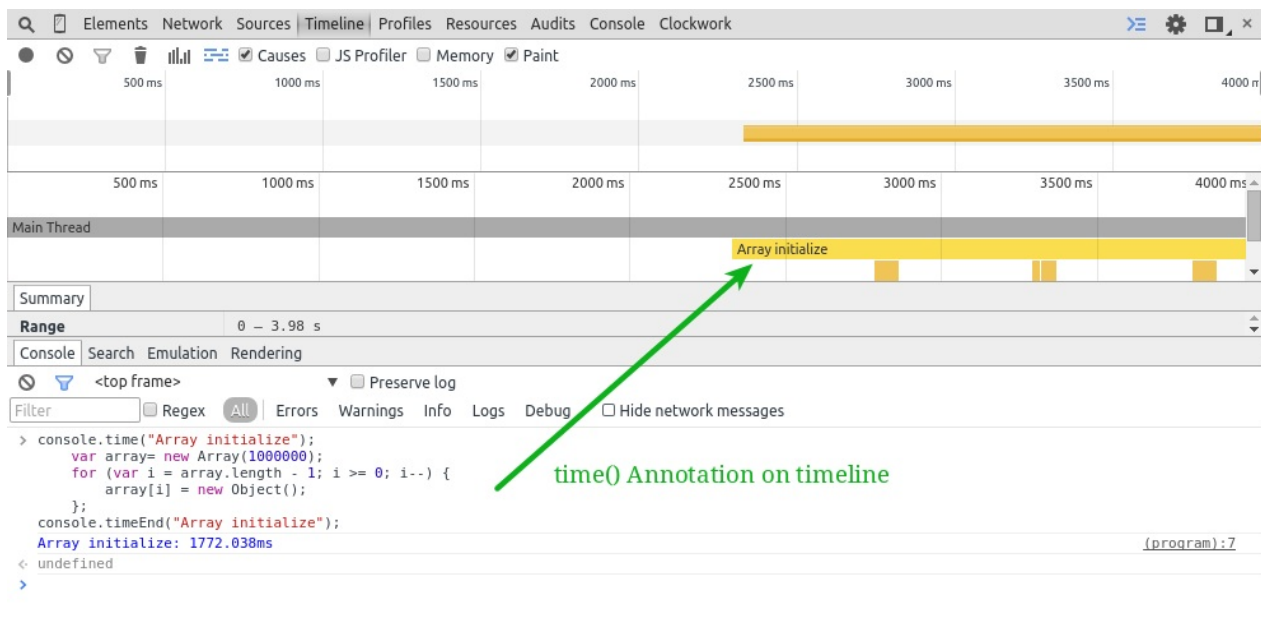
通过调用 `time()` 可以开启计时器。你必须传入一个字符串参数来唯一标记这个计时器的ID。当你要结束计时的时候可以调用 `timeEnd()`，并且传入指定的名字。计时结束后控制台会打印计时器的名字和具体的时间。

代码示例：

```
console.time("Array initialize");
var array= new Array(1000000);
for (var i = array.length - 1; i >= 0; i--) {
    array[i] = new Object();
};
console.timeEnd("Array initialize");
```



如果在计时期间正好有 `timeline` 操作。`time()` 则也会显示在 `timeline` 上。这可以帮你更加清晰的看到耗时到底在哪里。



标注 Timeline

在 Timeline 面板，你可以看到浏览器引擎的耗时分布。你可以通过调用 `timeStamp()` 方法在 Timeline 中打上一个标记。通过这样的方式你可以看到不同事件之间的联系。

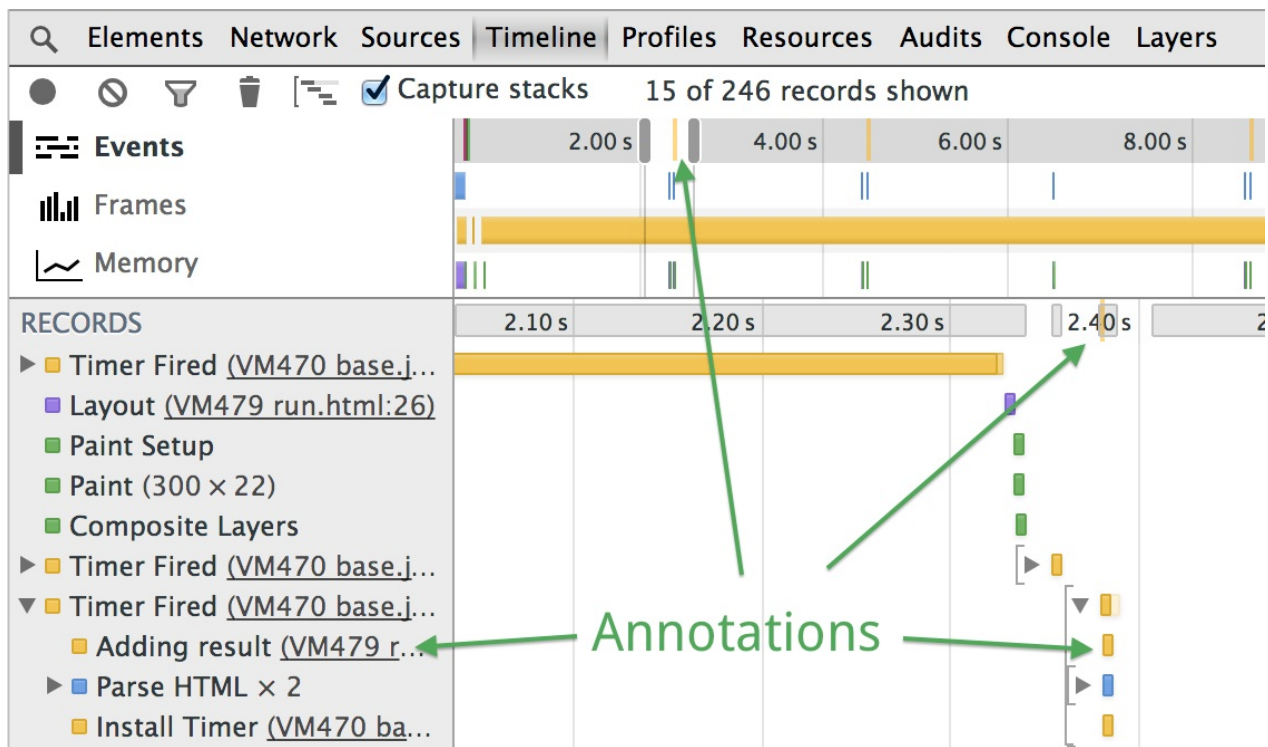
注意：`timeStamp()` 方法只有在 timeline 记录期间生效。

Timeline 中标记的结果会通过两种方式显示：

- Timeline 中的显示一个黄色的竖线
- 事件列表中新增一个事件

代码示例：

```
function AddResult(name, result) {
  console.timeStamp("Adding result");
  var text = name + ': ' + result;
  var results = document.getElementById("results");
  results.innerHTML += (text + "<br>");
}
```

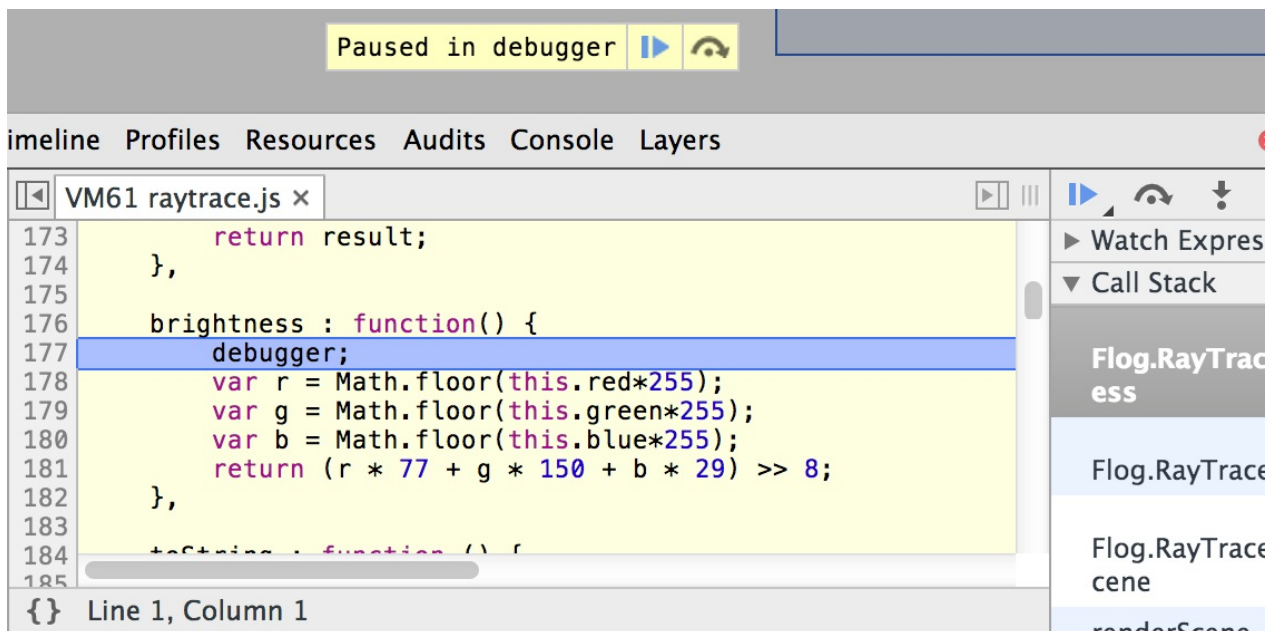


在JavaScript代码中设置断点

在代码中插入 `debugger;` 语句相当于在相应的代码行中插入一个断点。

代码示例：

```
brightness : function() {  
  debugger;  
  var r = Math.floor(this.red*255);  
  var g = Math.floor(this.green*255);  
  var b = Math.floor(this.blue*255);  
  return (r * 77 + g * 150 + b * 29) >> 8;  
}
```



统计表达式执行次数

`count()` 方法用于统计表达式被执行的次数，它接受一个字符串参数用于标记不同的记号。如果两次传入相同的字符串，该方法就会累积计数。

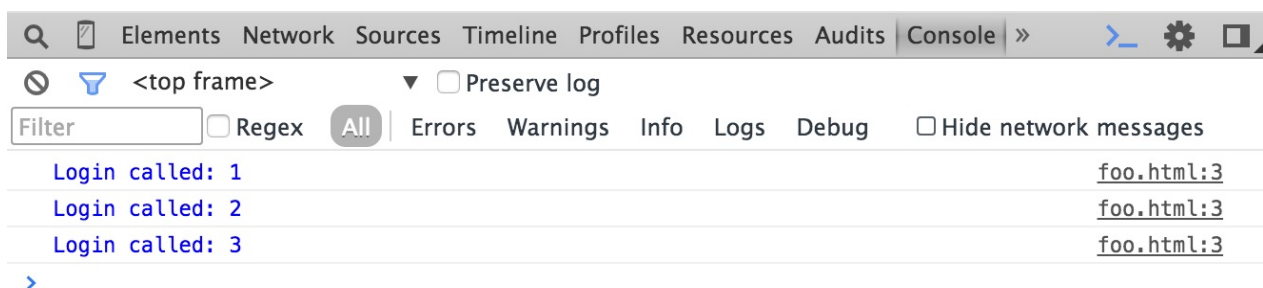
下面的代码示例显示了统计用户登录次数：

```
function login(user) {
  console.count("Login called for user " + user);
}

users = [ // by last name since we have too many Pauls.
  'Irish',
  'Bakaus',
  'Kinlan'
];

users.forEach(function(element, index, array) {
  login(element);
});

login(users[0]);
```



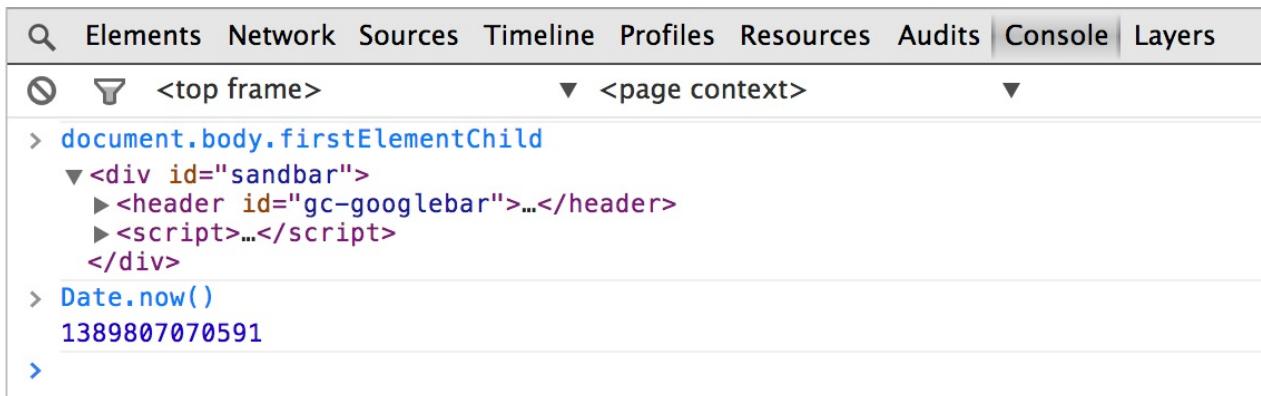
使用 Command Line API

控制台绝对不仅仅是简单的日志输出。它是一个功能完整的终端页面交互窗口，Command Line API 包含下面的特性：

- 快捷的DOM元素选择函数
- 提供检测CPU性能的方法
- 提供一系列Console API 的别名
- 事件监控
- 查看对象事件绑定情况

计算表达式

控制台会自动计算你输入的所有JavaScript表达式，它还支持命令补全。当你输入一个对象的时候，其属性会自动提示，你可以使用 `↑` 和 `↓` 进行选择。使用 `→` 选择当前提示。



选择元素

选择元素可以使用快捷方式，与传统方式比起来，这大大的节省了时间：`arts`。

- `$()` - 返回满足指定CSS规则的第一个元素，此方法为 `document.querySelector()` 的简化。
- `$$()` - 返回满足指定CSS规则的所有元素，此方法为 `querySelectorAll()` 的简化。
- `$x()` - 返回满足指定XPath的所有元素。

代码示例：

```
$('code') // Returns the first code element in the document.  
$$('figure') // Returns an array of all figure elements in the document.  
$x('html/body/p') // Returns an array of all paragraphs in the document body.
```

审查DOM元素和JavaScript内存对象

`inspect()` 接受DOM元素或者js对象引用作为参数。如果传入的是一个DOM元素，则DevTools会自动转到元素面板并显示该元素的结构。如果传入的是一个JS对象引用，则会打开分析器面板。

下面的代码首先选择页面中的一个元素，并且将其显示在元素面板中。`$_` 表示上一个表达式的输出结果

```
$('#[data-target="inspecting-dom-elements-example"]')
inspect($_)
```

访问最近选择的元素和对象

控制台会存储最近5个被选择的元素和对象。当你在元素面板选择一个元素或在分析器面板选择一个对象，记录都会存储在栈中。可以使用 `$x` 来操作历史栈，`x`是从0开始计数的，所以 `$0` 表示最近选择的元素，`$4` 表示最后选择的元素。

监控事件

`monitorEvents()` 会输出指定监控目标的事件信息。该方法的第一个参数表示被监控的对象，第二个参数表示要监控的时间，如果不填则会默认返回该对象上所有事件的监控信息。你可以将数组传入第二个参数，来同时监控多个事件。

代码，监听页面的点击事件：

```
monitorEvents(document.body, "click");
```

如果某个事件类型对应了一组标准的JavaScript事件，则指定该类型监听，会对这些事件都生效。[Command Line API](#)包含了事件类型的说明。

要结束事件监控只需要调用 `unmonitorEvents()`，将指定对象的监控移除。

控制CPU分析器

调用 `profile()` 方法可以开启CPU分析，你可以传入一个字符串参数对分析器进行命名。结束的时候可以直接调用 `profileEnd()`。

```
profile()
profileEnd()
```


Self	Total	Function
1934.1 ms	1934.1 ms	(idle)
0 ms	3.0 ms	▶ InjectedScript.evaluate VM24:490

如果你创建了多个名字相同的分析器，则会自动组合在一起：

```
profile("init")
profileEnd("init")
profile("init")
profileEnd("init")
```

Self	Total	Function
99.98%	99.98%	(idle)
0.00%	0.00%	(program)
0%	0.01%	▶ InjectedScript.evaluate VM24:490

多个分析器可以一起执行，而且并不需要按照创建的顺序去关闭。

按照创建顺序关闭

```
profile("A")
profile("B")
profileEnd("A")
profileEnd("B")
```

不按照创建顺序关闭

```
profile("A")
profile("B")
profileEnd("B")
profileEnd("A")
```

设置

在设置-通用中，你可以进行四项设置：

- 隐藏网络消息 - 隐藏关于网络情况的日志，例如屏蔽404或500的日志。
- 记录 XMLHttpRequests - 设置是否记录 XMLHttpRequest。
- 切换页面时保存日志 - 在页面切换的时候是否保存日志。
- 显示时间戳 - 每条语句被调用的时候显示时间戳，开启本功能会关闭 [message stacking](#)。

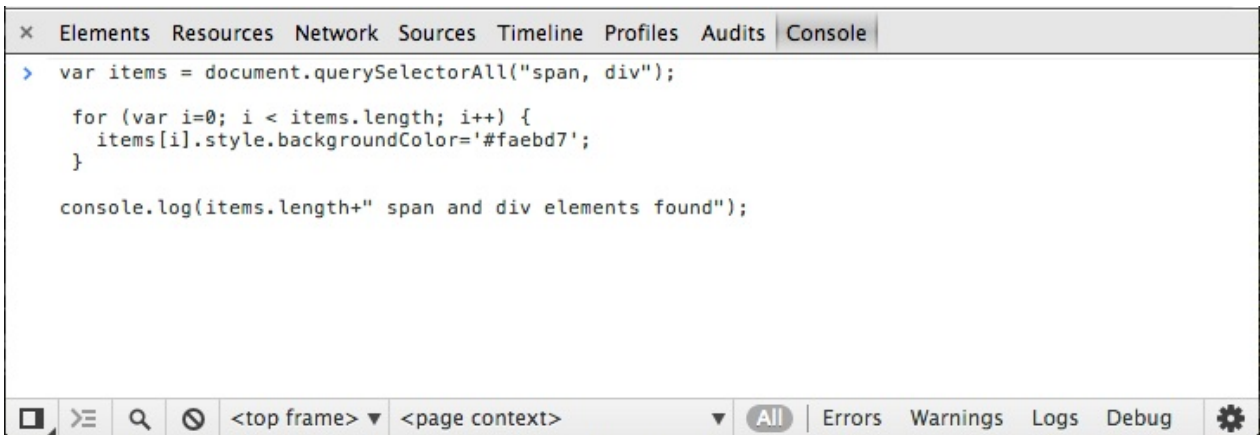
总结

Chrome DevTools提供了一个给力的控制台。你现在应该已经了解如何使用它存储信息和分析元素数据。它的功能还在不断的扩展。如果说web是你的花园，那么他们就是你的砖瓦:)

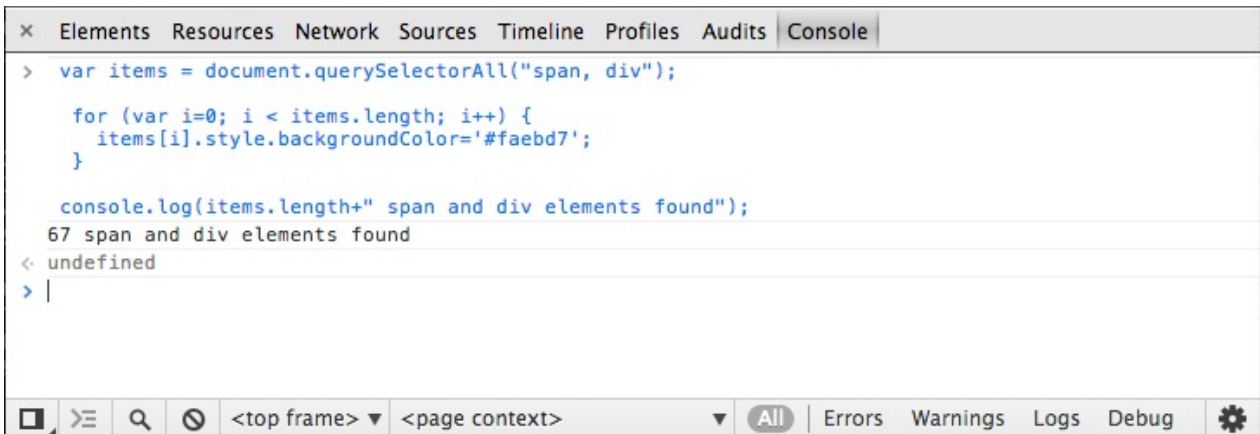
建议和技巧——控制台篇

编写多行命令

控制台在多行编辑的模式下，你可以像在普通的文本编辑器中那样输入命令。使用 `Shift + Enter` 可以输入普通的换行符，而实现多行的编辑。



当你需要写比较的复杂的JavaScript代码的时候，多行编辑就显得很给力了。当你写完一段代码的时候，直接敲回车就可以运行代码了。



关于支持保存代码的多行控制台，阅读[snippet](#)

快捷键打开审查元素模式

使用 `Ctrl + Shift + C` 或 `Cmd + Shift + C` 可以在DevTools中快速打开审查元素模式。(译注：可能会与系统快捷键冲突)

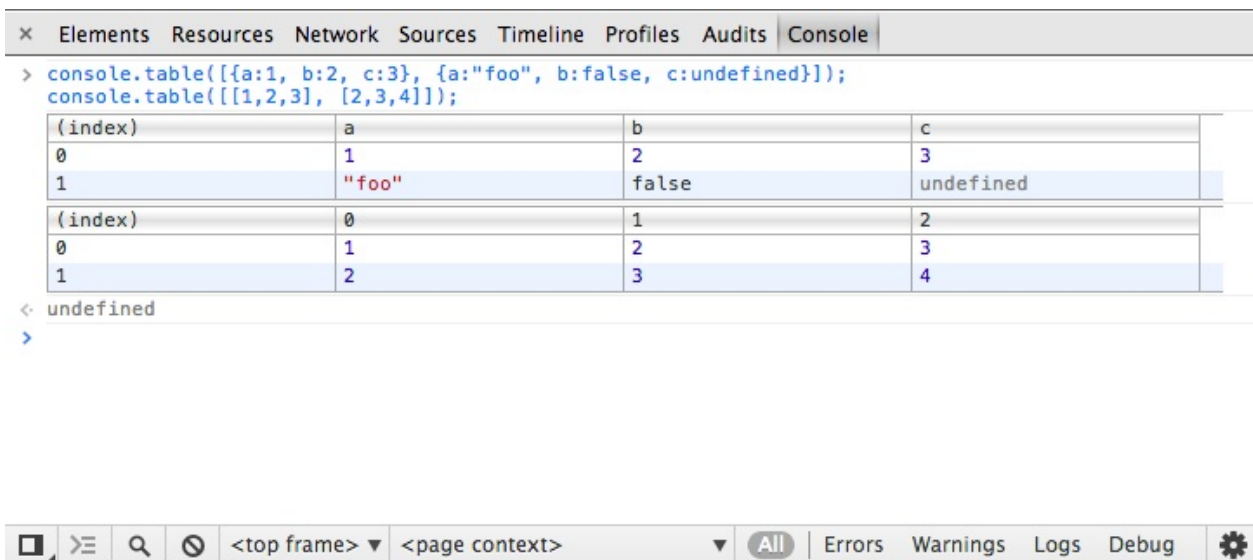


更多：使用控制台

支持 `console.table` 命令

该命令支持以表格的形式输出日志信息。下面为代码示例：

```
console.table([{a:1, b:2, c:3}, {a:"foo", b:false, c:undefined}]);
console.table([[1,2,3], [2,3,4]]);
```

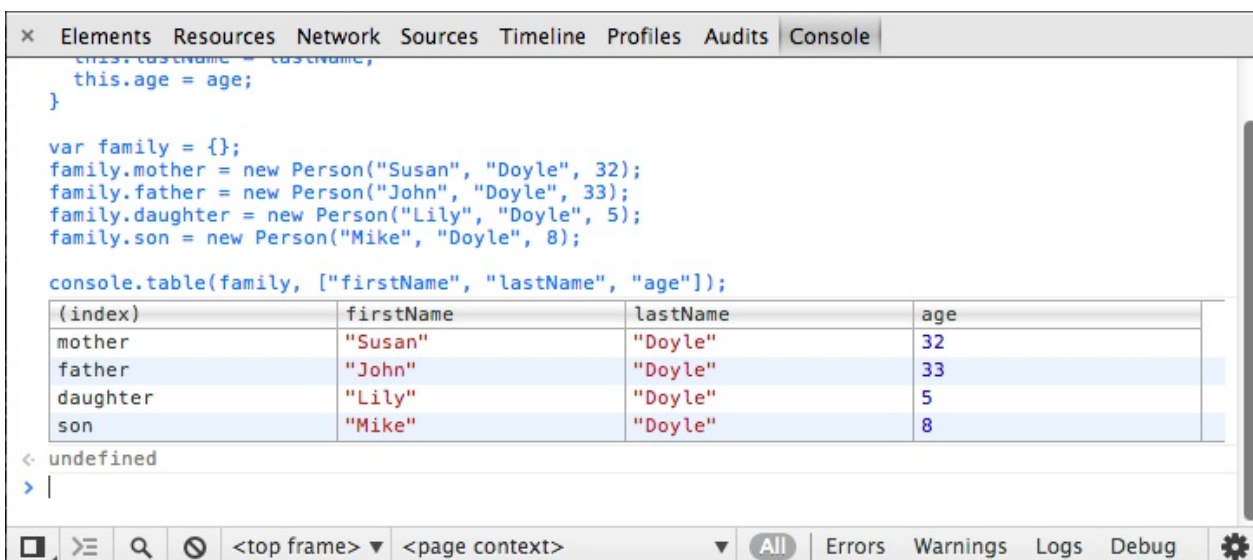


`table` 方法还可以接收一个 `columns` 参数用来设置表格列的标题。列参数的格式是一个字符串数组，每个数组元素一次对应着列的标题，如果对应列标题为空，则认为不显示该列。下面的代码定义了列的标题：

```
function Person(firstName, lastName, age) {
  this.firstName = firstName;
  this.lastName = lastName;
  this.age = age;
}

var family = {};
family.mother = new Person("Susan", "Doyle", 32);
family.father = new Person("John", "Doyle", 33);
family.daughter = new Person("Lily", "Doyle", 5);
family.son = new Person("Mike", "Doyle", 8);

console.table(family, ["firstName", "lastName", "age"]);
```



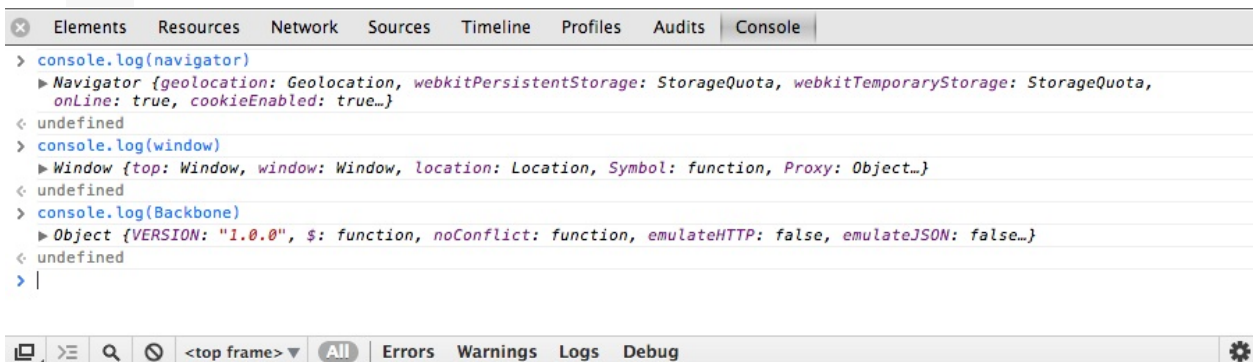
如果你只想显示 `firstName` 和 `lastName`，可以通过下面的方式调用：

```
console.table(family, ["firstName", "lastName"]);
```

了解`table`方法更多的使用功能| [G+](#).(译注：需要翻墙)

预览对象

通过 `log` 打印的对象可以直接在控制台中预览。



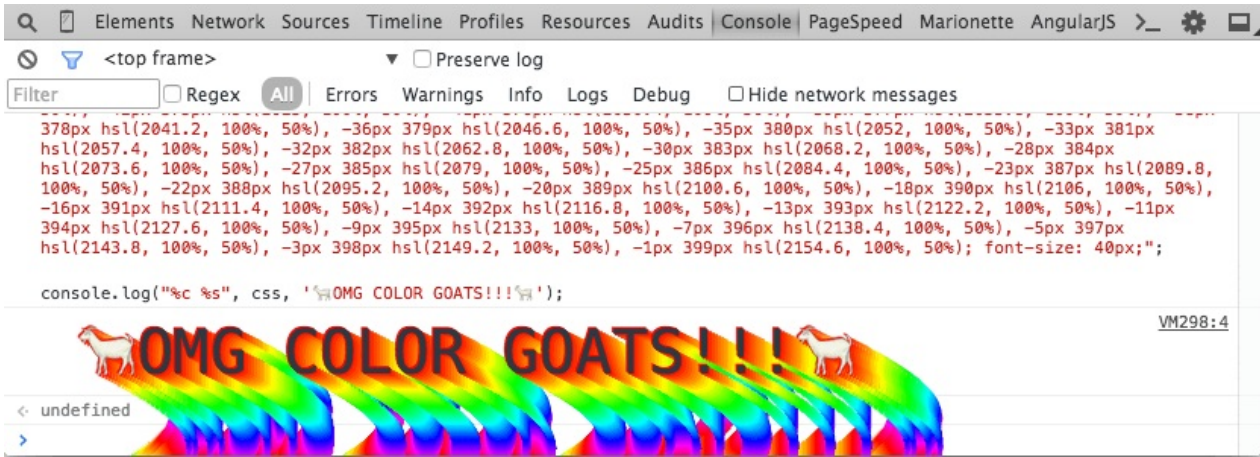
给控制台设置样式

在上一章我们已经提到过，你可以使用 `%c` 格式符，来为控制台增加样式：

```
console.log("%cBlue!", "color: blue;");
```

还可以为每一块输出设置样式：

```
console.log('%cBlue! %cRed!', 'color: blue;', 'color: red;');
```

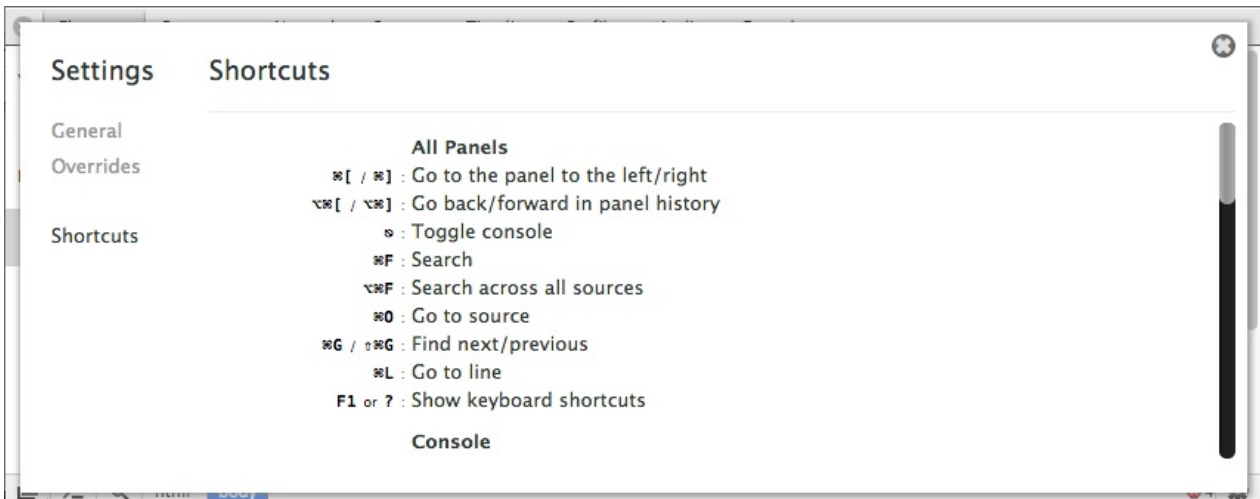
不同风格的控制台|G+

快速清空控制台历史

在DevTools窗口中，使用 `Ctrl + L` 或者 `Cmd + L` 快捷键可以快速清空命令行历史记录。当然你也可以在控制台直接调用 `clear()` 方法。

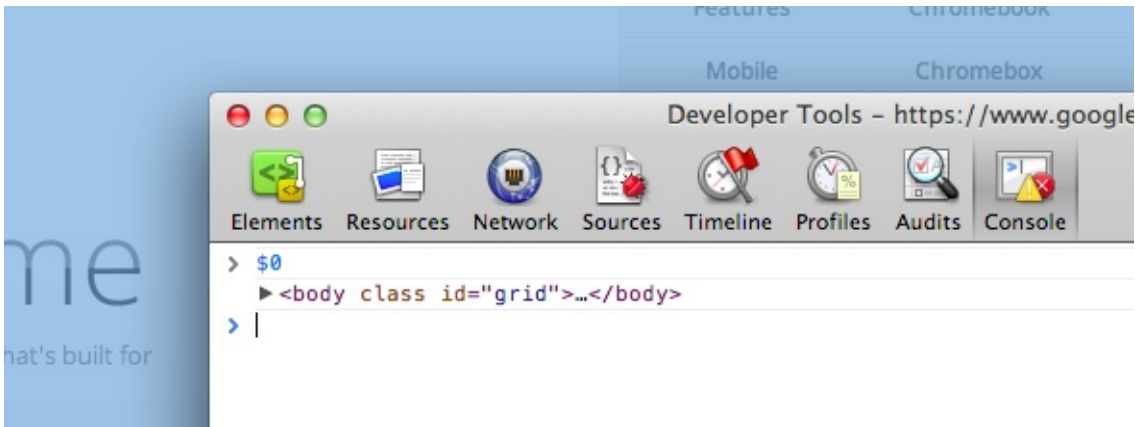
成为键盘党

在DevTools窗口中，使用 `?` 可以打开通用设置面板，在快捷键一项中可以看到DevTools当前支持的所有快捷键。

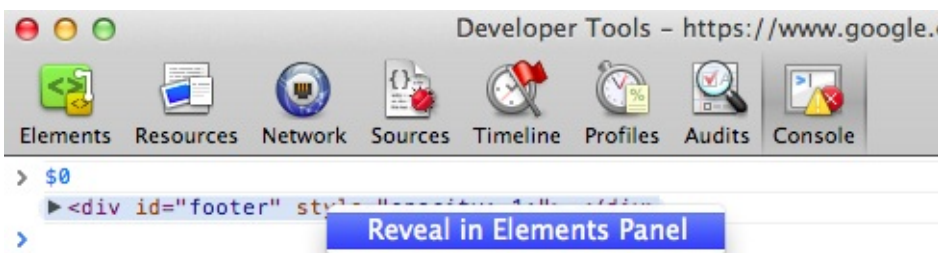


在控制台中直接访问页面元素

在元素面板选择一个元素，然后在控制台输入 `$0`，就会在控制台中得到刚才选中的元素。如果页面中已经包含了jQuery，你也可以使用 `$($0)` 来进行选择。



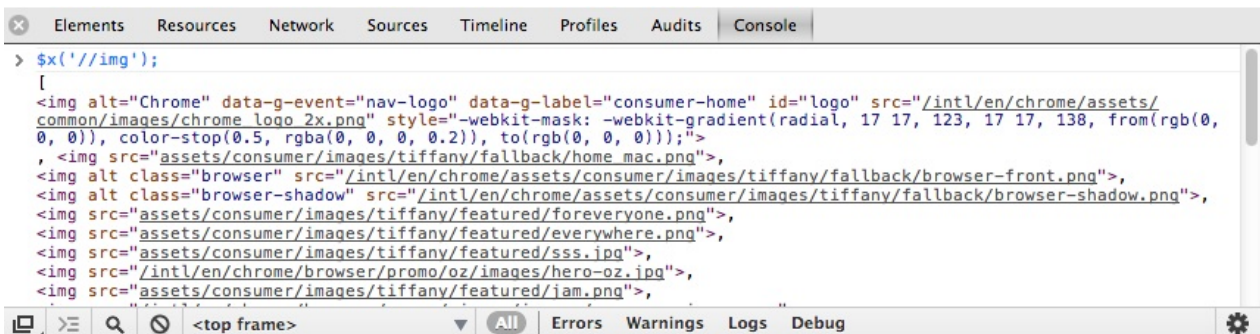
你也可以反过来，在控制台输出的DOM元素上右键选择 `Reveal in Elements Panel` 来直接在DOM树中查看。



使用XPath查询DOM

XPath是一种查询语言，用于从DOM文档中查找node，返回结果可能是一组node，字符串，布尔值或者数字。你可以直接在DevTools的JavaScript控制台中直接使用XPath查询语句。

你可以通过 `$x(xpath)` 的形式执行一个查询语句，下面的示例展示了如何在页面查询图片元素：



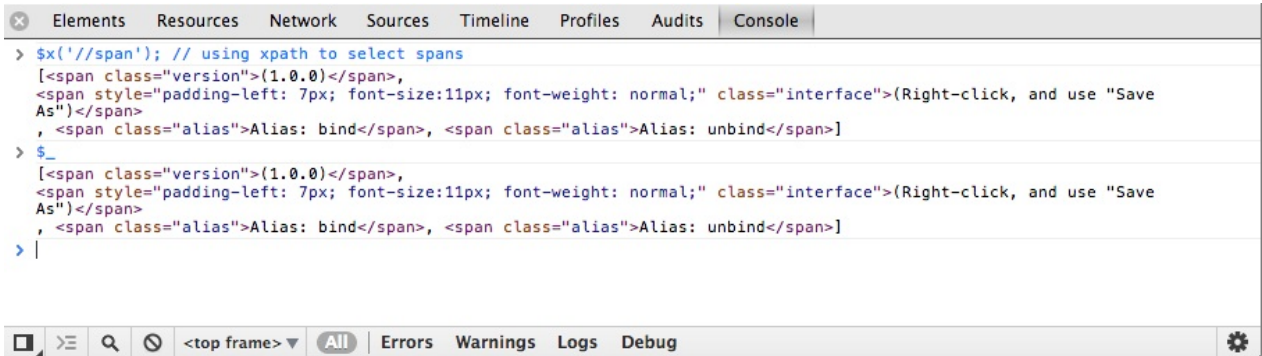
此外，这个方法还支持扩展的参数，你可以在第二个参数中传入查询的上下文，例如 `$x(xpath, context)`。通过扩展参数，你可以选择特定上下文里的DOM元素。

```
var frame = document.getElementsByTagName('iframe')[0].contentWindow.document.body;
$x('//*[@img]', frame);
```

在指定的`frame`中查找`img`元素

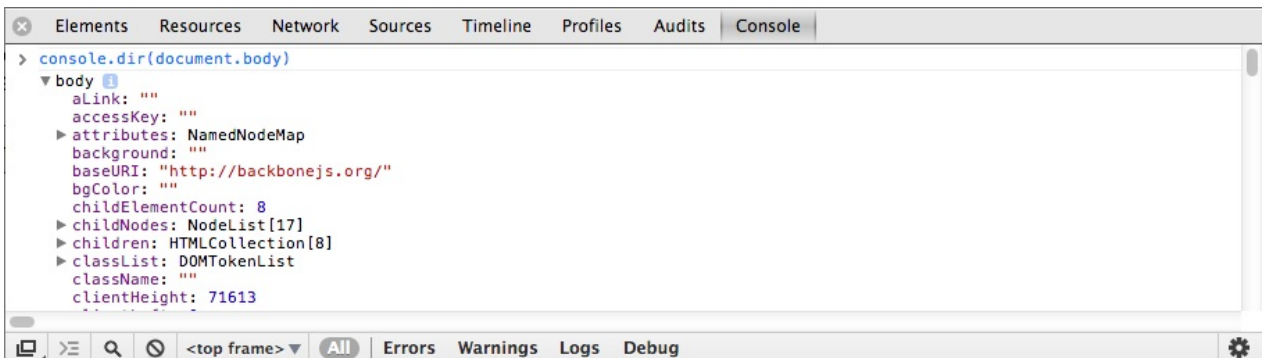
访问最近的控制台结果。

在控制台输入 `$_` 可以获控制台最近一次的输出结果。继续以XPath表达式查询为例：



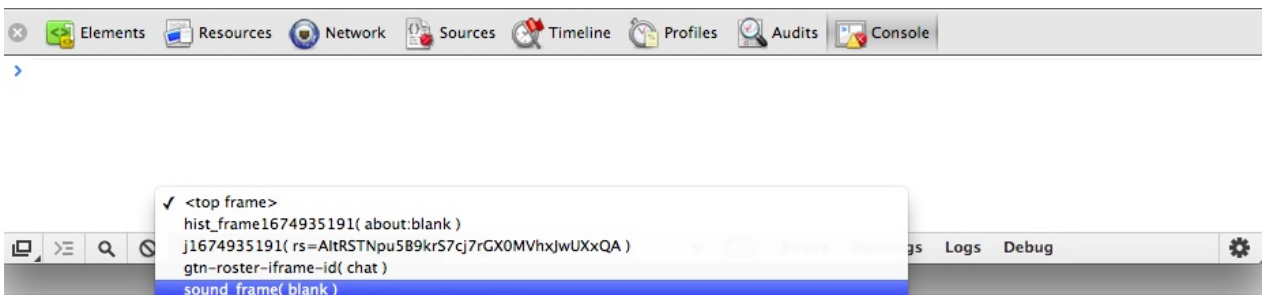
使用 `console.dir`

`console.dir(object)` 命令可以列出参数`object`的所有对象属性。下面的例子展示了`body`对象的属性：



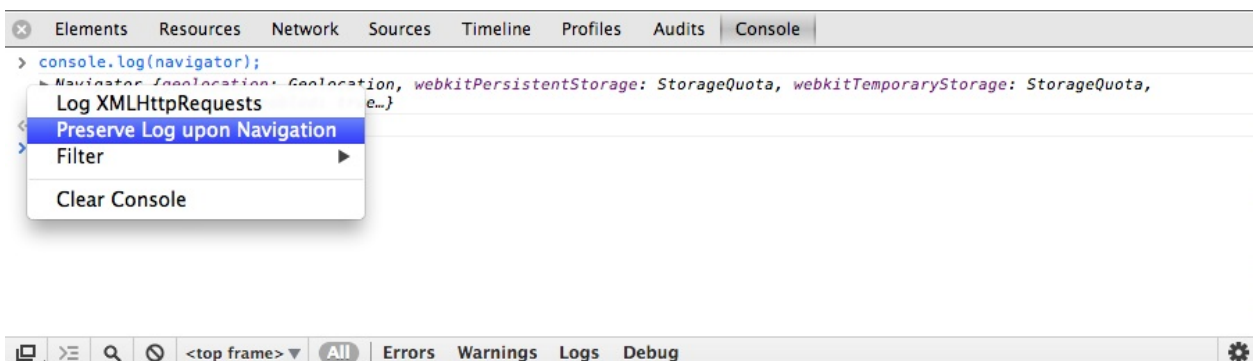
在特定`iframe`中执行JavaScript

在DevTools底部工具栏中有一个下拉列表用来改变当前控制台的上下文环境。当你打开控制台面板的时候，你可以直接在下拉列表中选择`frame`来作为JavaScript的执行环境。



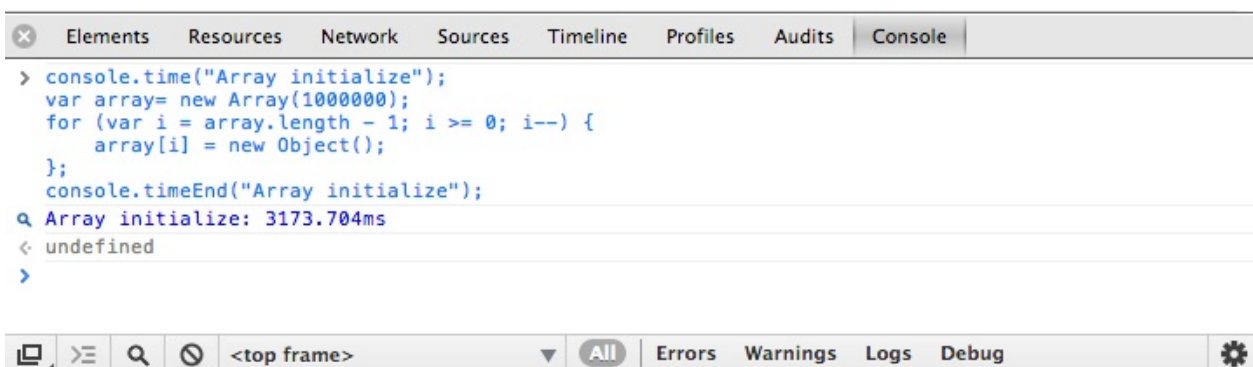
阻止控制台记录在新页面中被清空

在打开一个新的web页面的时候，你可能希望继续保留之前控制台的操作记录，这时候只需要在控制台右键选择"Preserve Log upon Navigation"，这样在打开新的页面的时候，之前的控制台记录还会保存。（译注：在最新版本的Chrome DevTools中，右键已经去掉该选项，可以直接在控制台上方标签中勾选Preserve log）



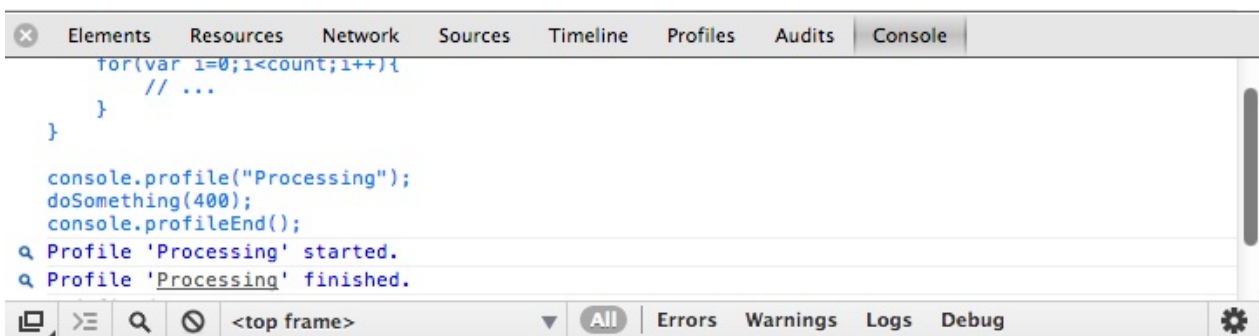
使用 `console.time()` 和 `console.timeEnd()` 分析循环的性能

调用 `console.time()` 开启一个计时器，调用 `console.timeEnd()` 关闭计时器，并在终端输出计时器消耗的时间。计时器在分析循环这样的非函数内操作的时候还是蛮有用的。

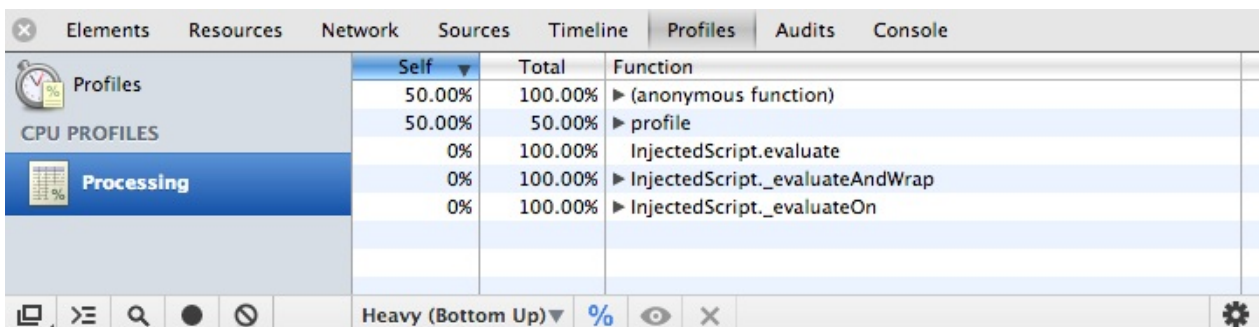


使用 `console.profile()` 和 `console.profileEnd()` 分析程序性能

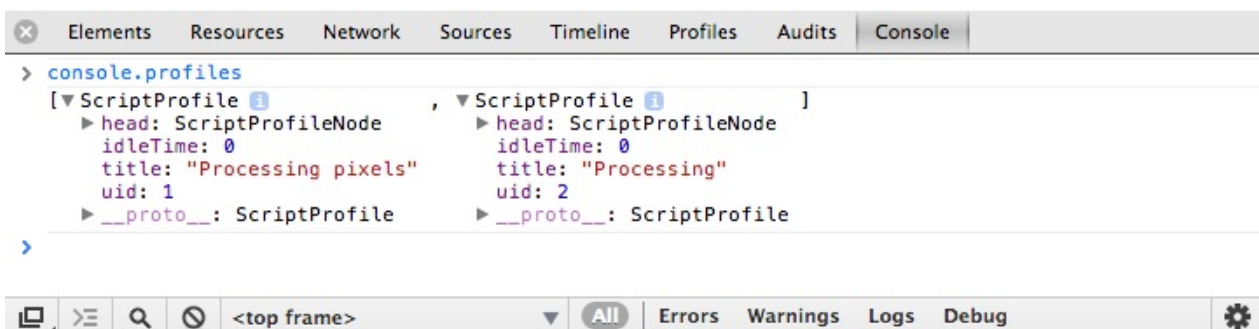
在DevTools窗口控制台中，调用 `console.profile()` 开启一个JavaScript CPU 分析器.结束分析器直接调用 `console.profileEnd()` .



具体的性能分析会在分析器面板中



分析结束后，也可以在命令行中调用`console.profiles[]`得到。

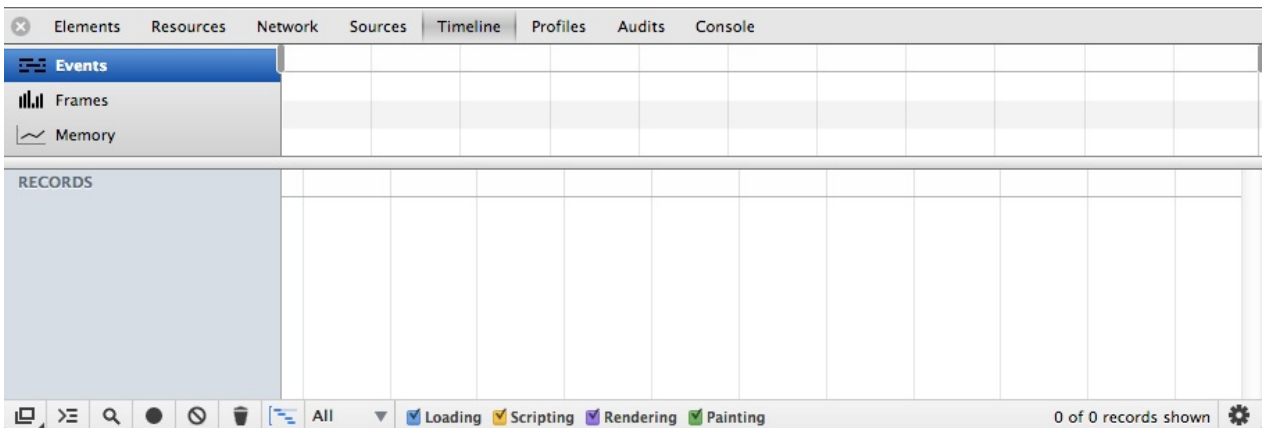


想了解更多关于控制台的使用技巧，请自觉前往[使用控制台](#)一章。

建议和技巧——Timeline篇

使用帧模式分析页面

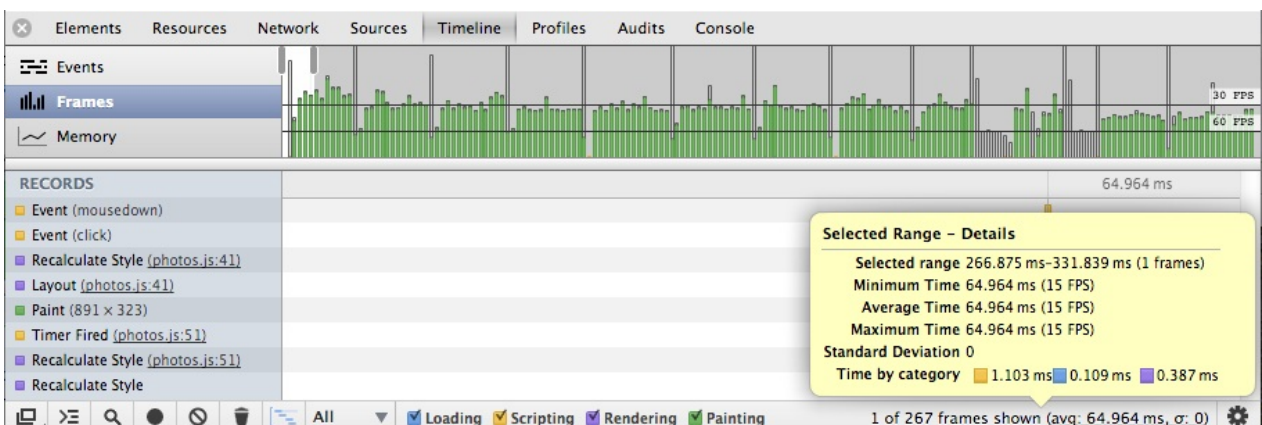
在Timeline面板中，你可以看到web应用内的耗时分布，例如执行DOM事件、渲染页面布局或者绘制页面元素等。你可以通过：页面帧、事件和实际内存使用三个方面发现程序的问题：



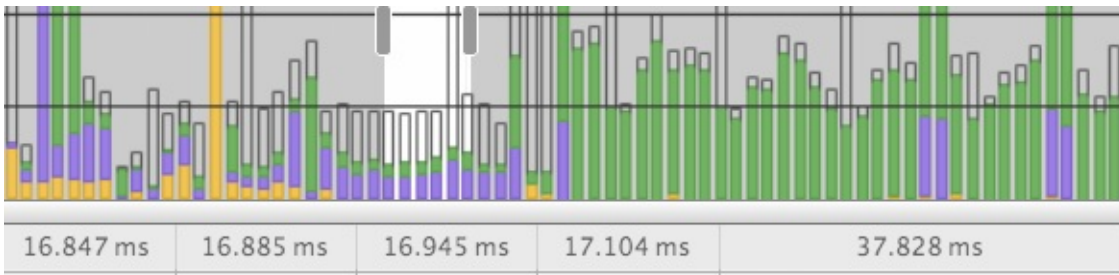
默认情况下 Timeline 不会记录任何数据，你可以通过点击Timeline面板下面的圆点图标来开启一个Timeline监控会话。开启Timeline快捷键是 `Ctrl + E` 或 `Cmd + E`。



Timeline开启时，圆点会从灰色变为红色。开启监控后，在页面内执行一些交互操作，然后可以单击按钮停止记录。

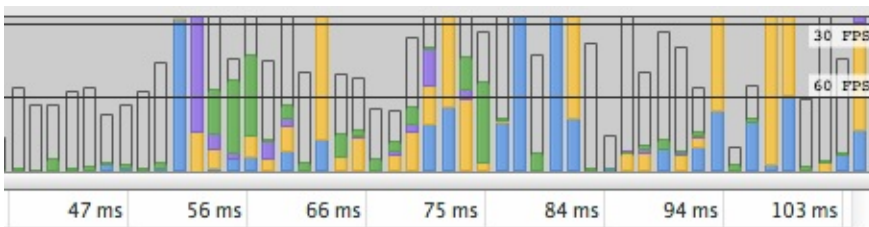


网页在浏览器打开的时候，Chrome会生成每一帧将页面渲染在屏幕上。Timeline的帧模式能够帮助你分析每一帧里发生的情况。在帧模式里，阴影部分的竖条对应的是重新计算样式、组合页面等操作所消耗的时间。透明的区域则表示空闲时间，至少在你当前页面是空闲的。



通常，每一帧都是根据刷新频率同步来执行的。在上面例子中，第二帧的耗时稍微超过了15ms,第三帧的任务错过了真正的硬件帧，只能等到下一帧再渲染。因而实际上第二帧消耗的时间加倍了，超过了30ms。

即使你的app中不涉及大量的动画，帧的概念也很重要。因为浏览器要不停的处理一系列输入事件，当你在一个帧里留下了充足的时间来处理这些事件，你的app就能更好的响应，这也意味着更好的用户体验。



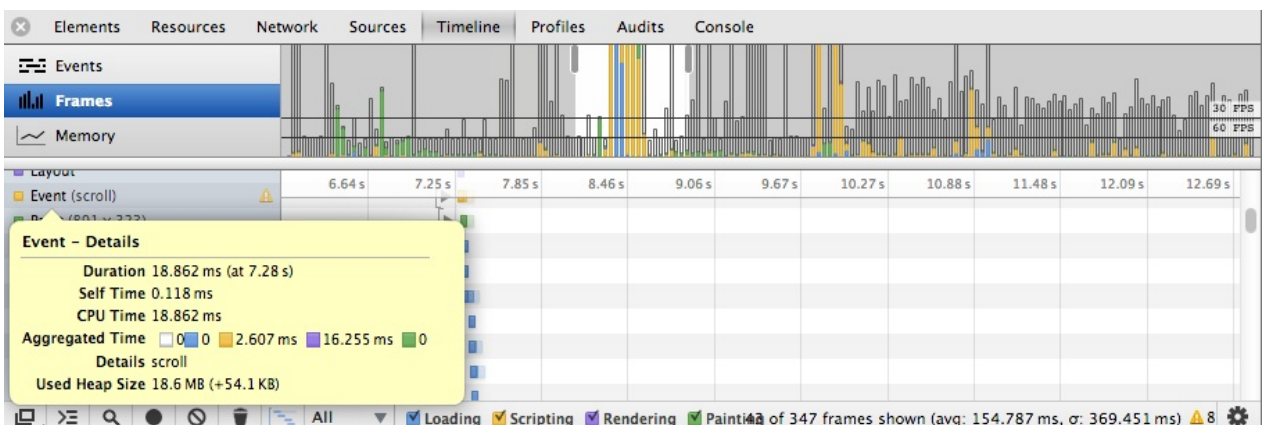
一般我们把目标帧频率定在60fps,我们在每帧里最多有16.66ms来处理任务。这个时间并不多，所以尽可能的从动画中节省性能消耗是很关键的一种优化方式。

更多：[利用DevTools Timeline 提升程序性能](#)

使用warning找出强制布局事件

在Timeline面板中，如果你看到一个黄色的三角警告符号，这表示你的某些代码会触发强制/同步布局事件。

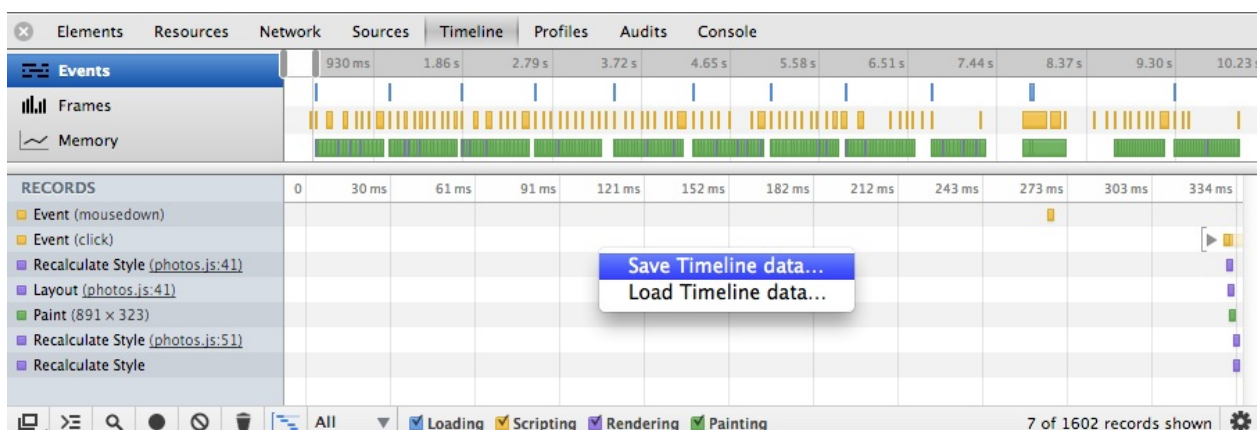
理想情况下，我们通常避免不必要的布局事件被触发，因为这些事件对页面的性能影响很大。



更多：[Timeline Warnings](#) 嗅探性能

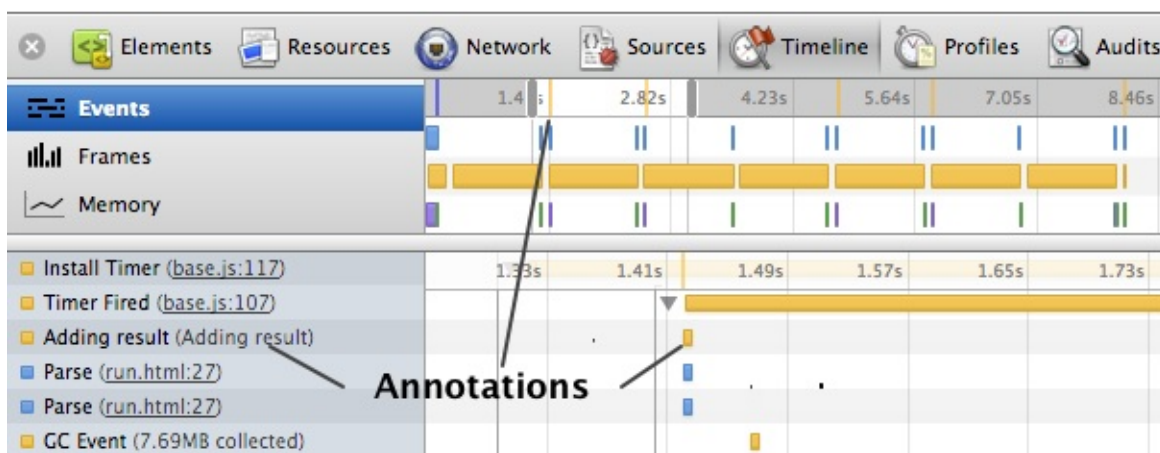
分享或分析他人的Timeline

使用Timeline的导入/导出功能，你可以分享自己的Timeline数据或者分析其他人的数据。使用 `Ctrl + E` 或者 `Cmd + E` 记录一段Timeline数据，在Timeline内右键选择 `Save Timeline data` 可以导出自己的Timeline数据，右键选择 `Load Timeline data` 可以导入其他的数据。



标注Timeline

你可以通过在代码中调用 `console.timeStamp()` 方法来在Timeline中打上标注。这样更容易找到你自己app中的代码与其他活动代码或浏览器自身实践的关联。



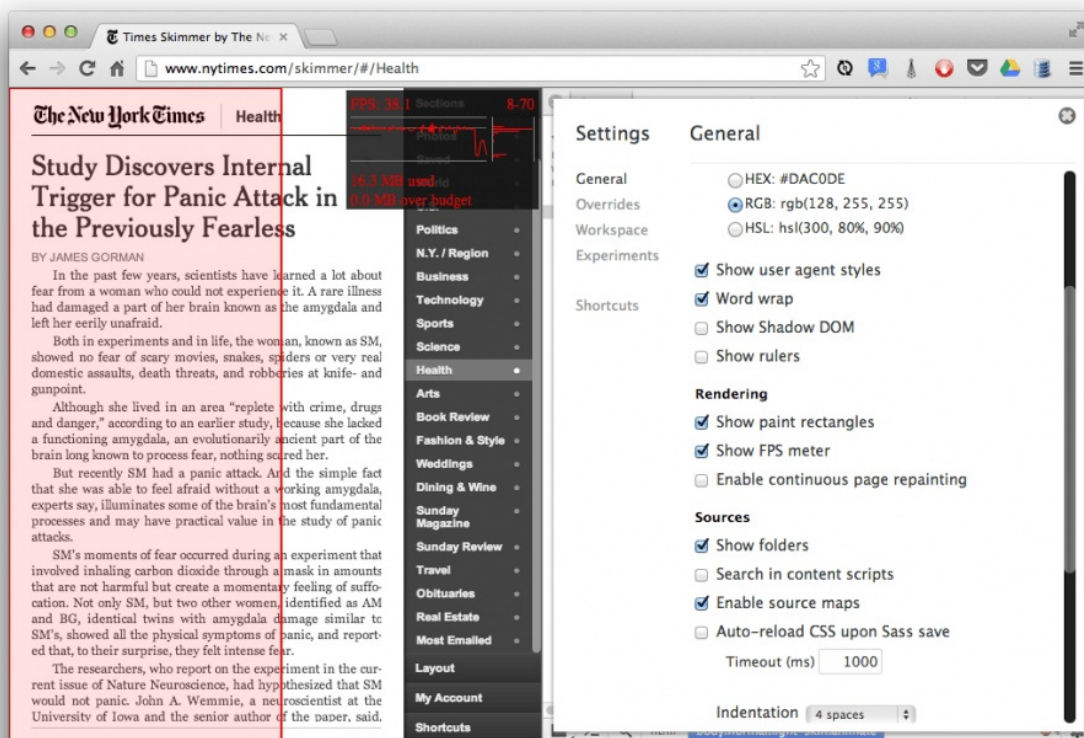
更多：[DevTools Console API - 标注Timeline](#)

FPS 计数器/HUD平视显示器

DevTools 提供了一个FPS的监控计数器，用于显示可视化的帧频率。可以在设置-选中 `Show FPS meter` 来开启。



激活计数器后，你可以在页面的右上角看到一个黑色的区域，里面包含了画面帧的实时数据。有了这个工具，你就可以在实时操作页面的过程中直接看到什么导致了页面帧刷新频率下降，而不用麻烦去打开Timeline。



更多：使用DevTools 连续绘制模式分析性能|Paul Irish大牛博文

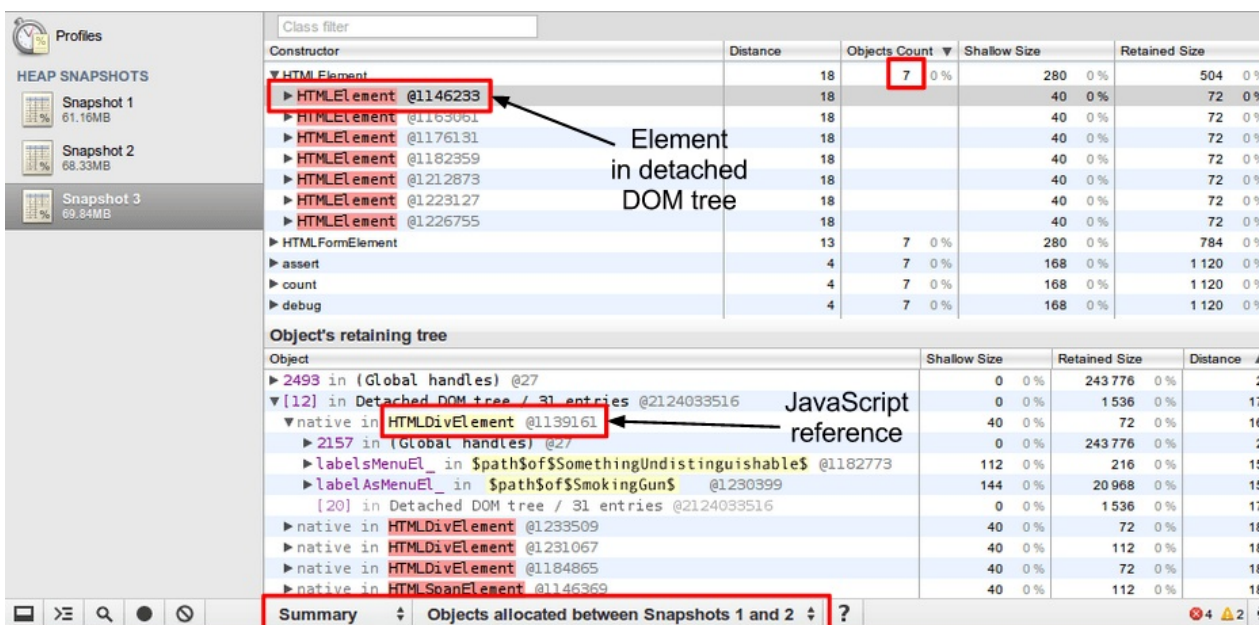
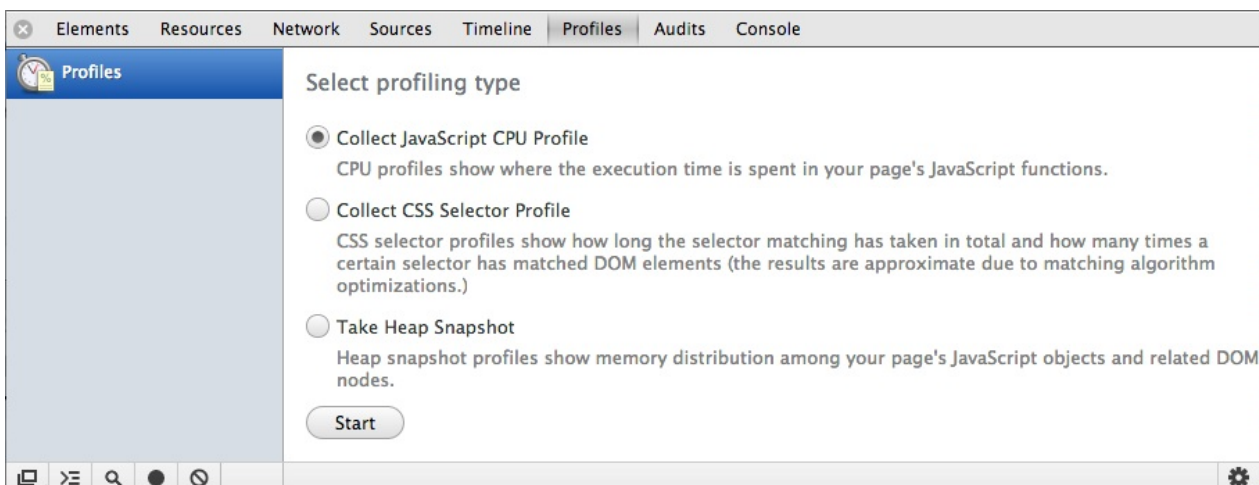
记住只关注FPS计数器可能会让你忽视间歇性的闪烁(译注：frames with intermittent jank，猜测指代的是那些错失物理帧的页面帧)，谨慎的分析这些内容。还要注意的是桌面的FPS并不代表移动设备上的情况，移动设备上的性能分析也十分值得关心。

想了解更多关于控制台的使用技巧，请自觉前往[Timeline](#)一章。

建议和技巧——Profiles篇

利用 三次快照 发现JavaScript内存泄露

1. 打开DevTools 并切换到 Profiles 面板
2. 执行一个会导致内存泄露的操作
3. 创建一个新的heap快照
4. 重复步骤2和步骤3三次
5. 选择最近的heap快照
6. 下边的默认结果过滤为 All Object ,将其改为 Objects between Snapshot 1 and 2
7. 这时候你就可以看到一组未被释放的对象，选择其中一个，可以在 Object's retaining tree 中查看什么导致了内存泄露。



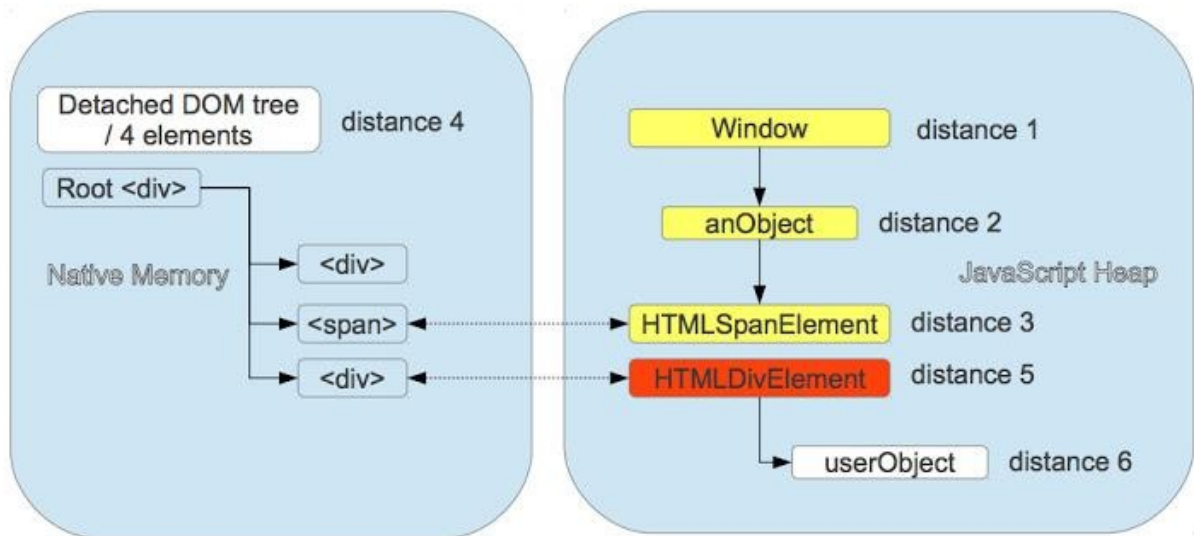
PPT：排除GMail中的内存泄露

了解Heap 分析器中的不同节点含义

红色的节点表示仍然存在的分离DOM树的一部分，并且DOM树种的某个节点仍然在被JavaScript引用（可能是一个闭包或者某些属性）

黄色的节点表示一个分离DOM树的引用，可能是某个对象的属性或者是一个数组元素，在元素和 `window` 之间可能存在着一属性链（例如 `window.foo.bar[2].baz`）

Detached DOM tree color schema



扩展阅读：理解Heap分析器中的节点

Heap 分析器的其他视图

一个常见的疑惑是，Profile面板中快照结果里的 Comparison, Dominator, Containment 和 Summary 四种视图之间有什么区别。四个视图分别从不同角度分析快照数据：

Comparison 视图可以显示哪些对象已经被垃圾回收正确释放了。一般在该视图中比较一次操作前后的内存快照数据。通过检查空闲内存中的变量增量和引用数来确定内存泄露的存在和原因。

Dominators 视图用来确认对象已经没有其他未知的引用，并且垃圾回收可以正常工作。(译注：新版本Chrome中，该面板已经去掉，新增了Statistics，统计不同类型数据所占的内存)

Summary 视图可以按照类型追踪对象及其内存使用情况，对象会以构造器名分组显示，主要用于寻找DOM内存泄露的场景。

Containment 视图可以更清晰的了解到对象的结构，借此可以分析出在全局作用域中对该对象的引用情况(例如window)，可以用来分析闭包，以更低的层次去查看对象情况。

The screenshot shows the Chrome DevTools Profiles tab with the 'Profiles' section selected. The 'CPU PROFILES' section is active, showing a profile named 'Profile 1'. The 'HEAP SNAPSHOTS' section shows three snapshots: 'Snapshot 1' (8.3 MB), 'Snapshot 2' (8.3 MB), and 'Snapshot 3'. The 'Profiles' section is expanded, showing a table of objects. The table has columns: 'Constructor', 'Distance', 'Objects Count', 'Shallow Size', 'Retained Size', and 'Distance'. The table shows the following data:

Constructor	Distance	Objects Count	Shallow Size	Retained Size	Distance
HTMLDocument	4	4	0%	64	0%
HTMLDocument	5	16	0%	132	0%
HTMLDocument	4	36	0%	1616	0%
HTMLDocument	5	16	0%	524	0%
HTMLDocument	4	36	0%	260	0%
HTMLDocument	4	36	0%	260	0%
HTMLDocument	4	36	0%	260	0%

Below the table, the 'Object's retaining tree' is shown. It lists objects and their retaining relationships:

- Object: `HTMLScriptElement @42501`, Shallow Size: 16, Retained Size: 84, Distance: 3
- Object: `HTMLDivElement @221561`, Shallow Size: 16, Retained Size: 84, Distance: 3
- Object: `HTMLBodyElement @42611`, Shallow Size: 16, Retained Size: 84, Distance: 3
- Object: `HTMLSpanElement @227459`, Shallow Size: 16, Retained Size: 84, Distance: 3
- Object: `function HTMLScriptElement() @`, Shallow Size: 36, Retained Size: 1616, Distance: 4
- Object: `HTMLAnchorElement @98371`, Shallow Size: 16, Retained Size: 96, Distance: 5

At the bottom, a context menu is open, showing options: 'Summary', 'Comparison', 'Containment', and 'Dominators'. The 'Containment' option is selected.

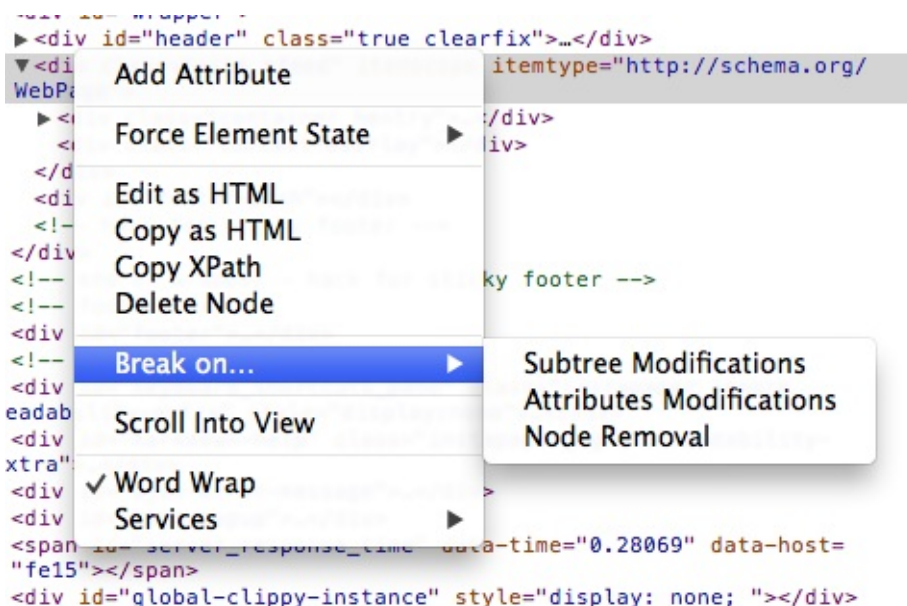
扩展阅读：更加轻松的在Chrome中分析JavaScript内存

想了解更多关于性能分析的使用技巧，请自觉前往[分析内存性能](#)一章。

建议和技巧——代码篇

调试&DOM编辑

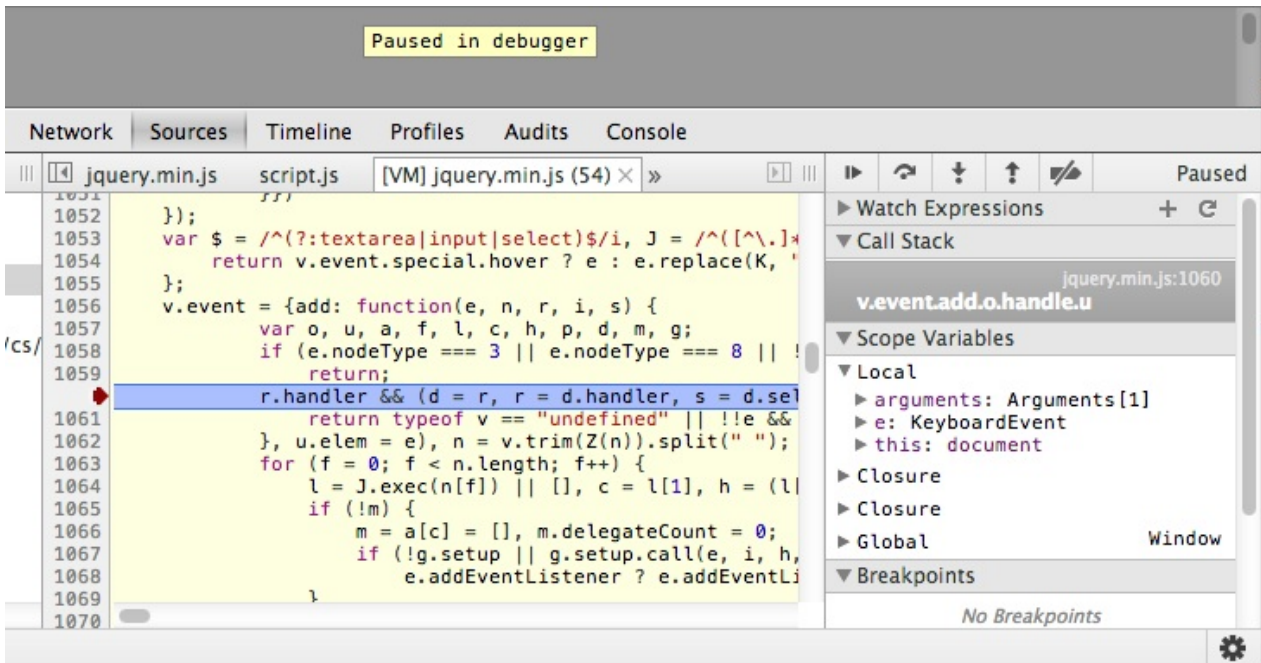
在元素上右键，选择‘Break on Subtree Modifications’: 当脚本遍历到该元素子节点并且进行修改操作时，断点调试器就会自动被触发。



同样值得注意的是设置“Attribute modifications”选项可以在元素的inline style变化时触发调试器，这在调试DOM动画的时候很有用。

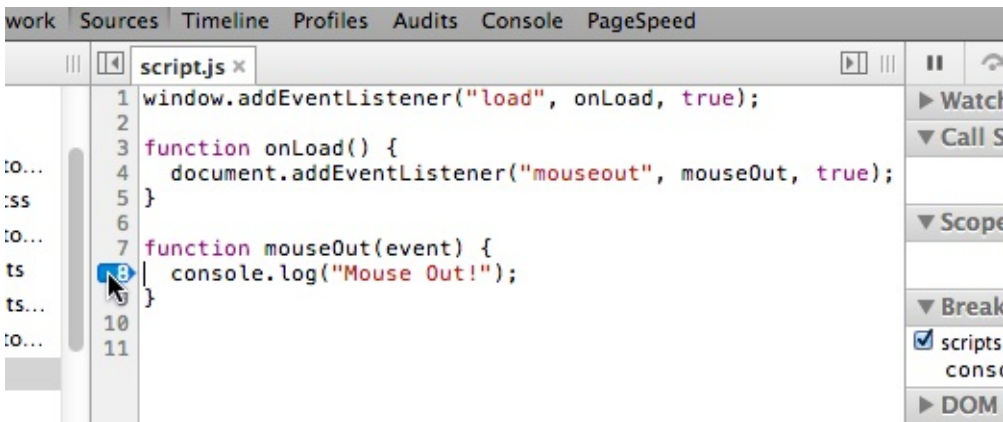
追踪未捕获异常

在 Source 面板，双击 暂停执行脚本 按钮 (||), 在未捕获的异常出现时，调试器就会开启，保存调用栈和程序当前的状态信息。

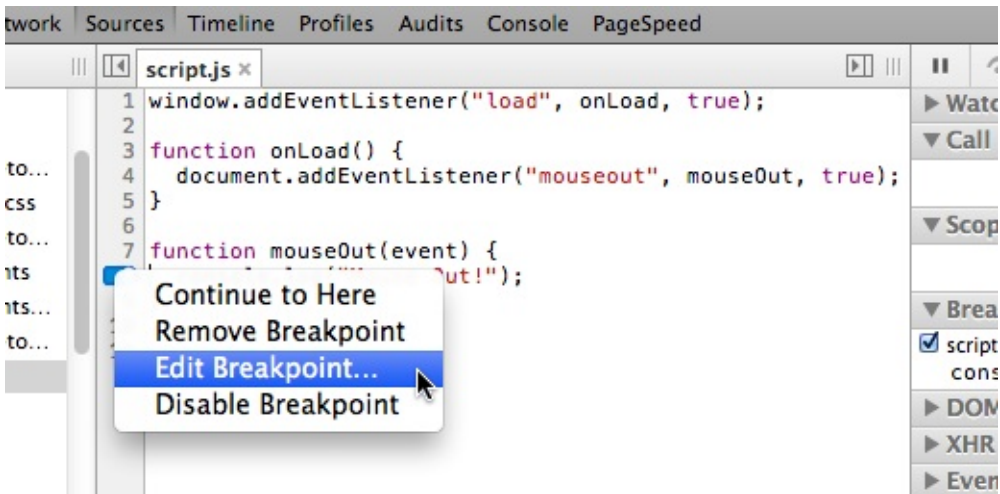


条件断点

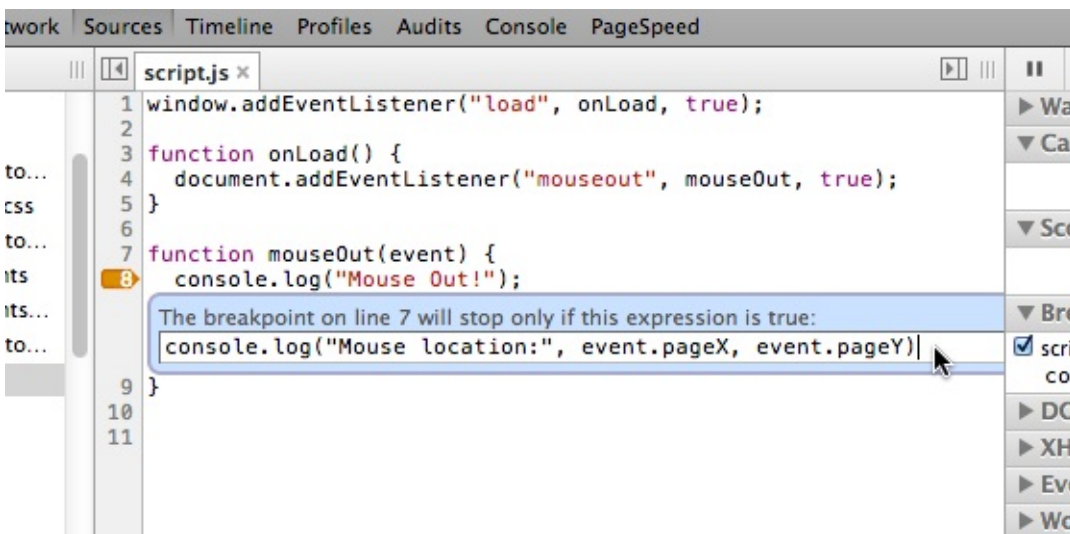
DevTools支持条件断点，我们都知道在代码的行号上单击鼠标可以在当前行设置一个普通断点，程序执行到这里就会暂停。



接着，你可以在断点上右键然后选择 "Edit Breakpoint"，这样就可以看到一个表达式输入框。在里面可以定义条件，如果条件为 `True`，断点就会生效。



一个通常的表达式可能是 `x === 5` 这种，然而在表达式写 `console.log` 语句也是完全OK的。



这个条件表达式可以正常的工作，并且我们可以很明显看到 `console.log` 语句在代码经过断点的时候执行了：



因为 `console.log` 并没有真正的返回值，所以相当于返回了 `undefined`。这样对应的条件断点相当于不满足条件而不会被触发，程序在打印表达式信息后会继续的执行。所以这种做法感觉就相当于硬编码进去一个调试语句而不需要真的修改自己的代码。

扩展阅读：[JavaScript断点调试](#)

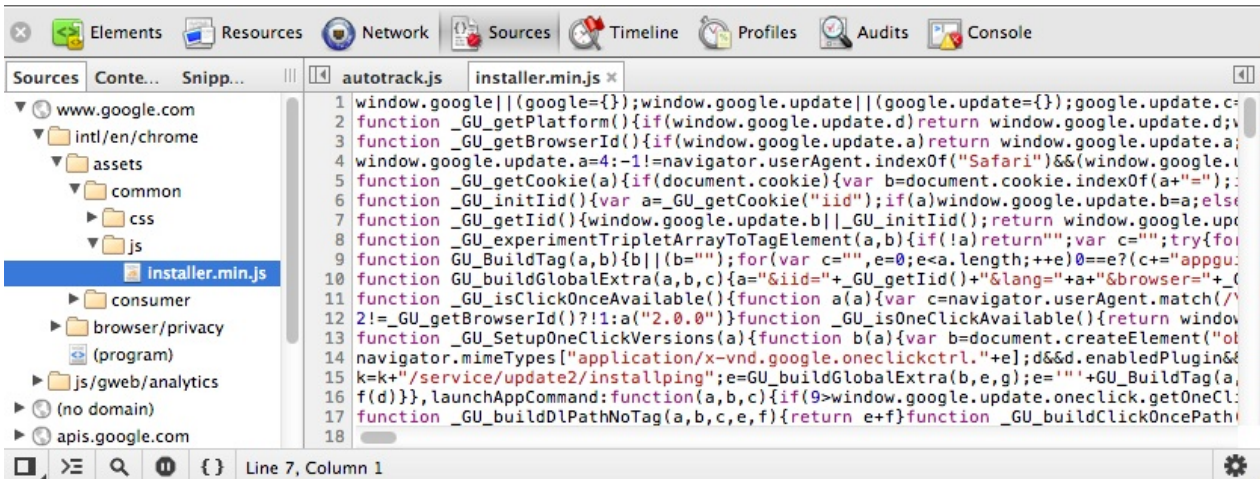
清晰显示JavaScript代码

一般web页面的代码都是minified压缩的。DevTools支持将代码格式化显示，使代码可读：

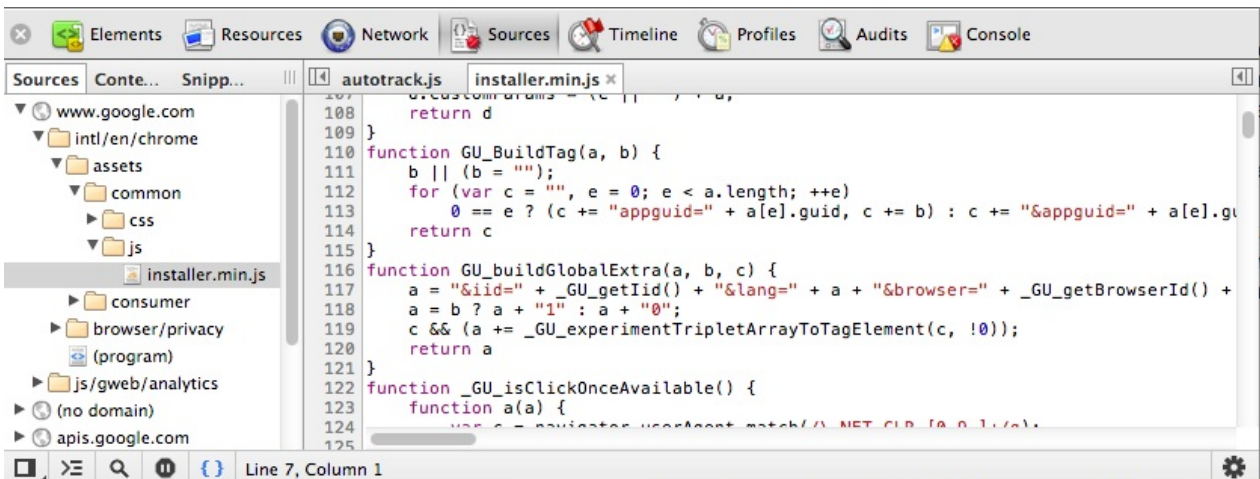
- 前往Source面板选择你想要查看的JavaScript代码
- 接下来，点击"Pretty print" 按钮 (大概这个样子 `{ }`)

你的代码现在看起来一定清晰漂亮多啦！

之前

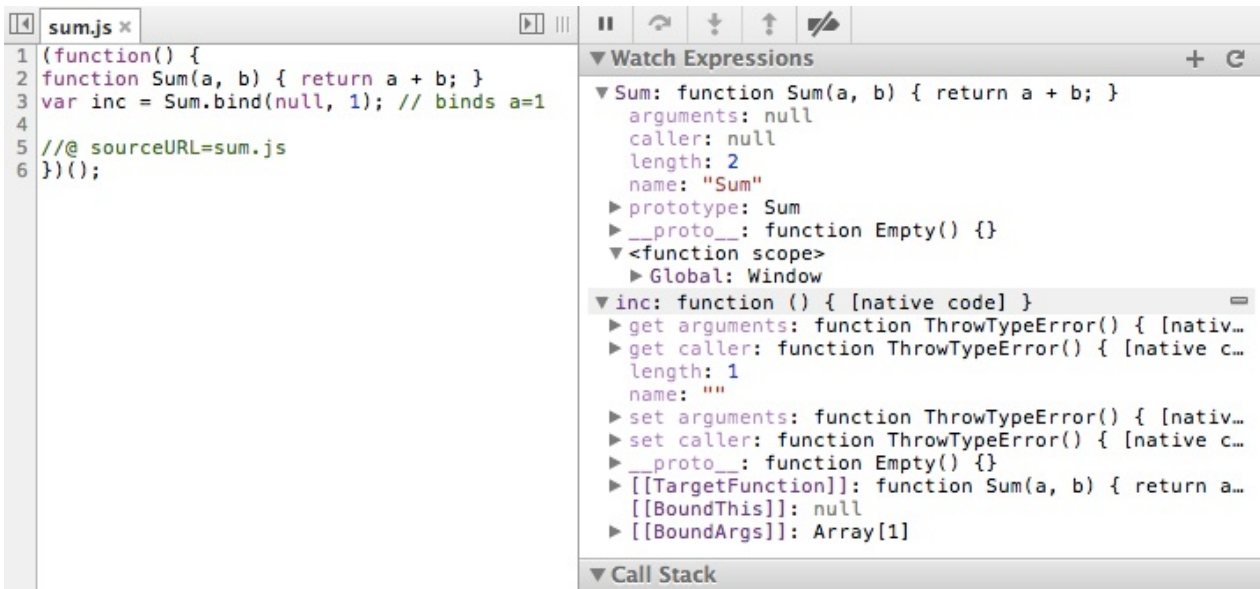


之后



收藏表达式或者变量值

在调试过程中，你可能需要一遍又一遍的输入相同的变量名或者表达式，有了DevTools你可以将这些常用的变量和表达式添加到 Watch Expression 中。你可以直接对这些内容进行修改，或者只是用于在代码运行的过程中观察这些指的变化。



查看内部属性

假设你定义了一个变量 `s`，它可以进行下面的操作：

```
s.substring(1, 4) // returns 'ell'
```

`s` 一定是一个 `string` 类型吗？它也可能是一个包装 `string` 对象。在 `watch expressions` 添加下面的表达式：

```
"hello"
Object("hello")
```

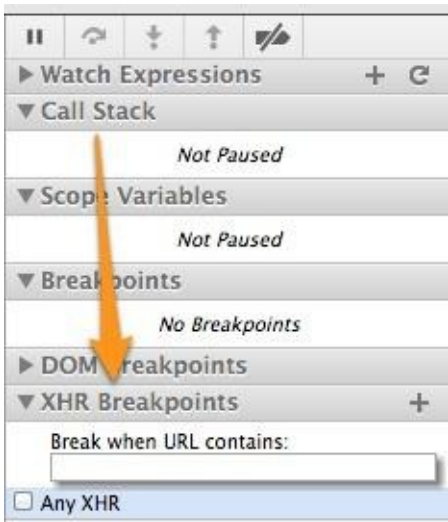
第一个是一个普通的字符串，第二个是一个完整的对象。有些让人迷惑的是，二者看起来几乎是一样的，但是后者拥有属性，并且你可以为其添加属性。

展开它的属性列表，你就会发现它并非一个普通对象，它拥有一个内部的属性 `[[PrimitiveValue]]` 存储着字符串的值。你无法在代码中访问这个属性，但是可以在调试窗口里面看到。

扩展阅读：[通过DevTools学习JavaScript](#)

轻松调试 XHRs

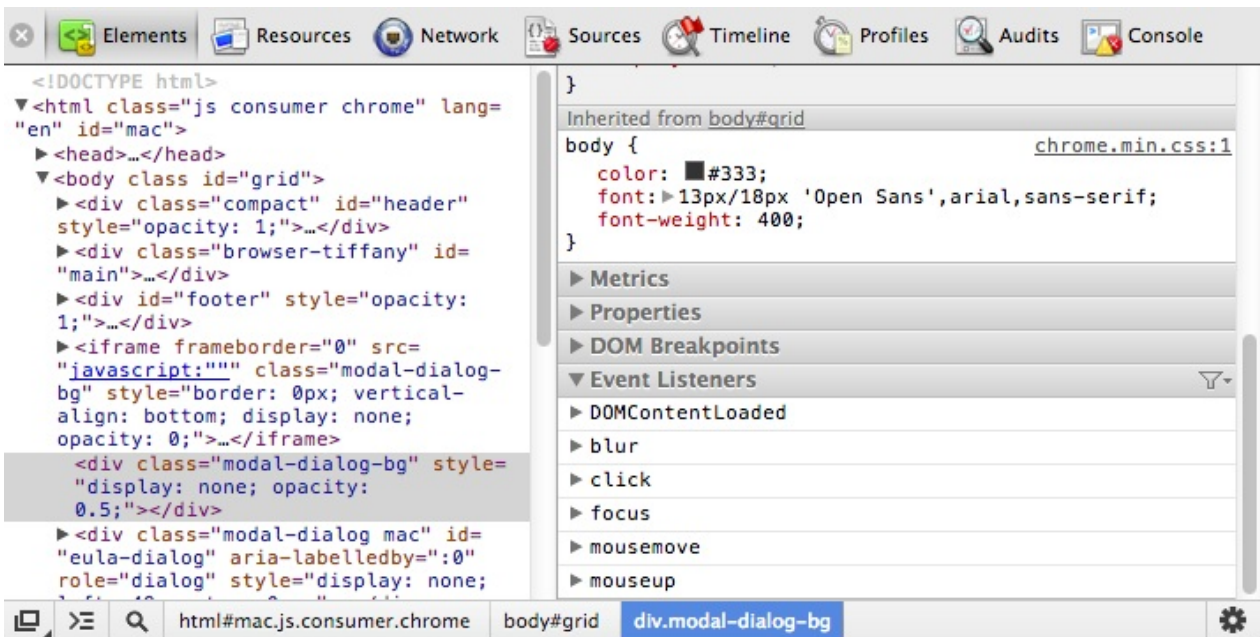
在调试窗口中打开“XHR Breakpoints”部分，你可以添加一个URL或者一个字符串来，指定发起某种XMLHttpRequest时开启程序调试。你也可以在任何XHR发起时都进入调试：



检索元素上注册的事件

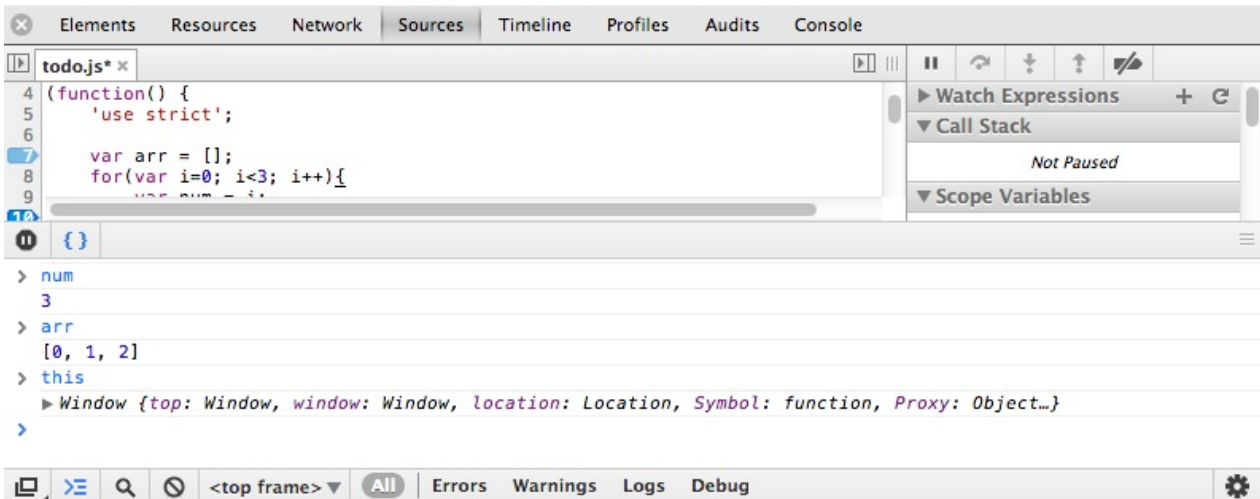
打开 Elements 面板，在 DOM 树中用鼠标选中一个元素。注意：你也可以在控制台中使用 `getEventListeners(targetNode)` 选择。

接下来在右侧窗口，展开“Event Listeners”选项。在这里会看到，当前元素所绑定事件列表。



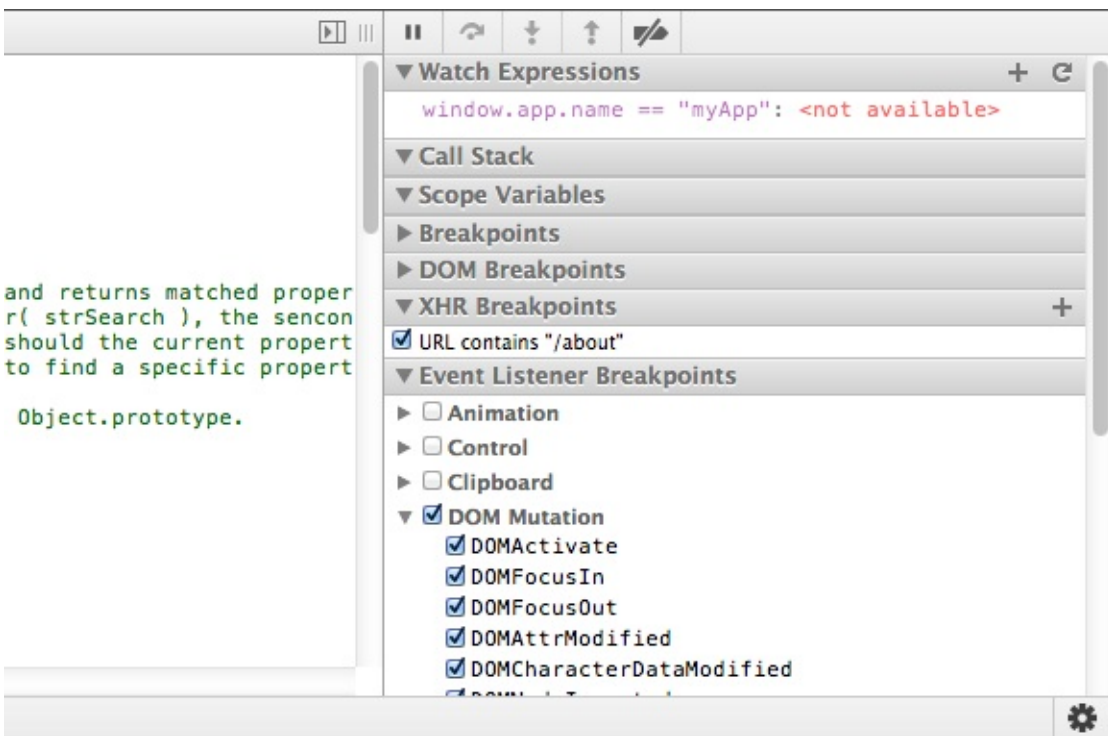
Esc 快速打开控制台

在 Source 面板中调试的时候，你有时需要同时访问控制台。这时只需要按 `Esc` 键，就可以打开控制台。你可以在控制台中执行 JavaScript 代码，不过更加给力的地方在于，在代码的调试断点执行 JavaScript 的环境就是当前暂停的程序上下文。（译注：因此你可以在此时编写代码查看一些特定的数据）



更加高效的处理断点

当你的程序在调试断点暂停的时候，你可以进行更多的操作：



你或许知道你可以通过 "Continue", "Step Over", "Step Into" 和 "Step Out" 控制程序执行，其实每一项功能都有对应的快捷键。学习这些快捷键操作可以在调代码的过程中更加高效。

Watch Expressions (在窗口的右侧)会实时的现实表达式的值，所以你无须切换到控制台去计算表达式(例如 `X === Y`)。

Call Stack (在Watch EXpressions下方)会列出当前系统已经执行函数调用到哪里了。

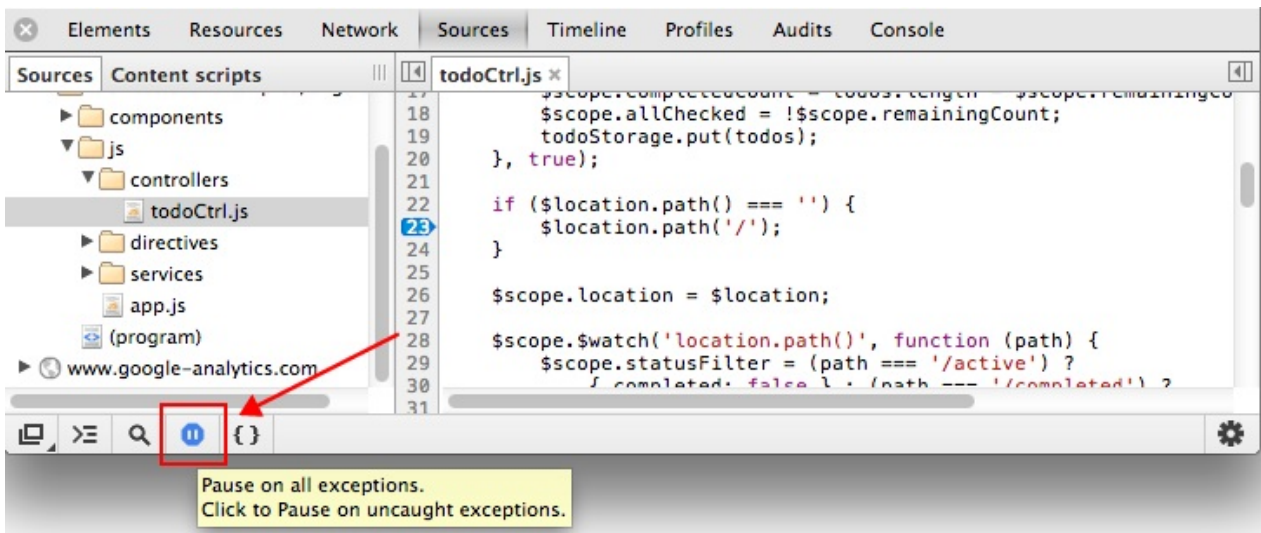
在 Scope Variables 部分，你可以在任何函数上右键然后选择 "Jump to definition" 跳转到函数定义的位置。

DOM Breakpoints 会显示那些在 Elements 面板被添加调试断点的元素。这个部分将帮助你观察那些事件是否已经成功的绑定到了元素上,以及事件被触发的时候执行了哪些操作。

XHR Breakpoints 的作用在于为XMLHttpRequest请求设置了断点,指定URL可以观察具体的异步请求情况。

异常时暂停

你可能希望在有异常被抛出的时候暂停JavaScript的执行,然后去检查函数调用栈,作用域变量和app的状态。



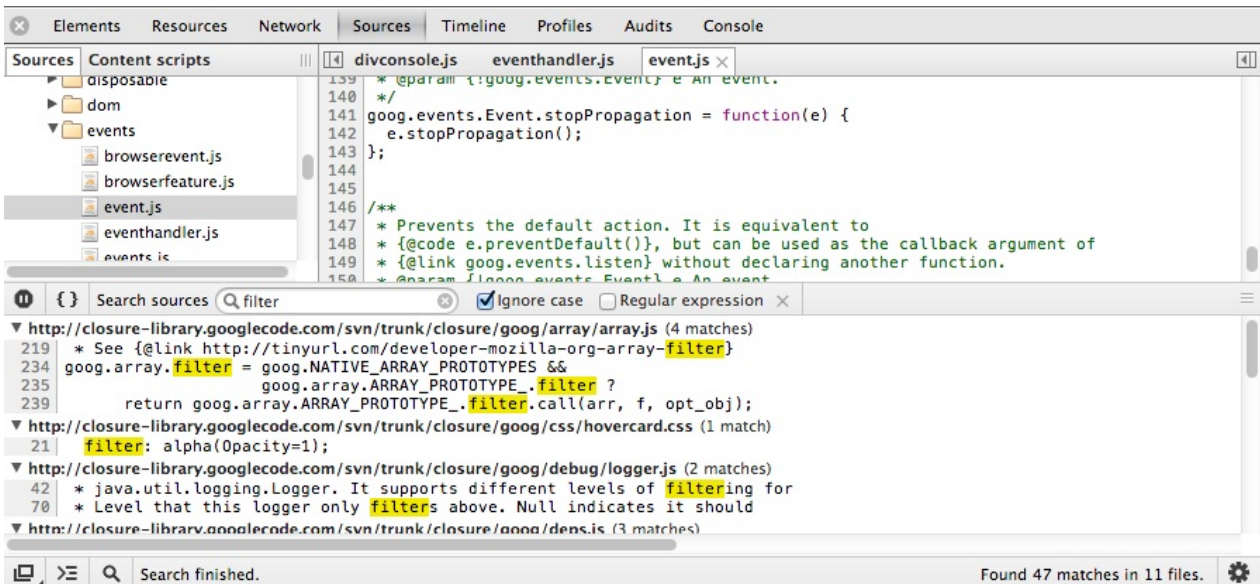
点击Scripts面板顶部的黑色暂停按钮可以切换不同的异常处理模式。很可能你并不希望在所有的异常抛出时都去暂停程序,除非你调试的代码都有try/catch语句。

全文检索

如果你想在所有文件中检索一个字符串,可以使用下面的快捷键:

- **Ctrl + Shift + F** (Windows, Linux)
- **Cmd + Opt + F** (Max OSX)

检索支持正则表达式和大小写敏感。



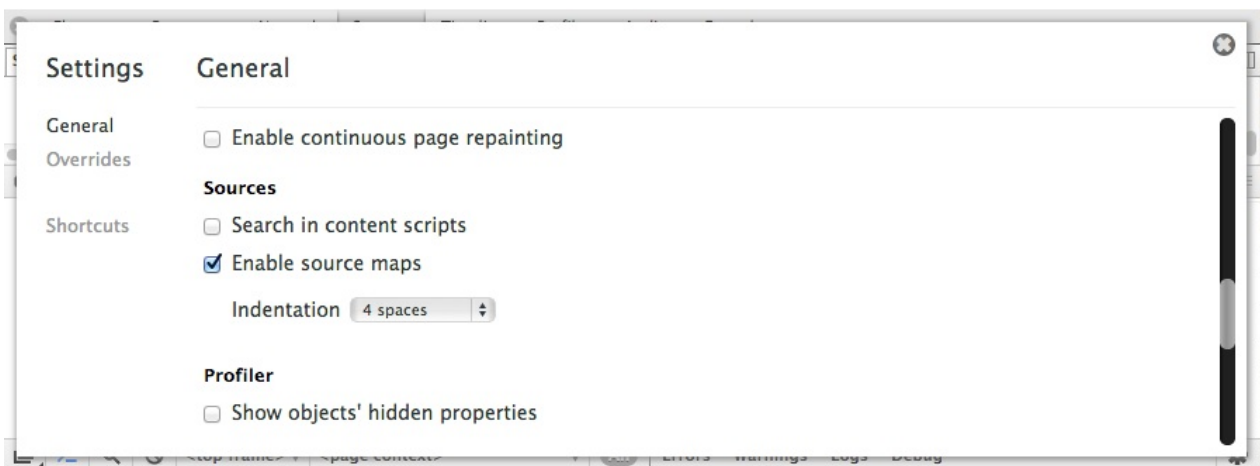
使用DevTools和源码地图调试 CoffeeScript

源码地图是一种将编译好的生产环境代码还原到开发环境的原始代码的方式，并且是语言无关的。

生产环境中生成的代码一般是被压缩过的，这使得很难定位到生成环境的代码对应到源码中的哪一行。

打开源码地图支持：

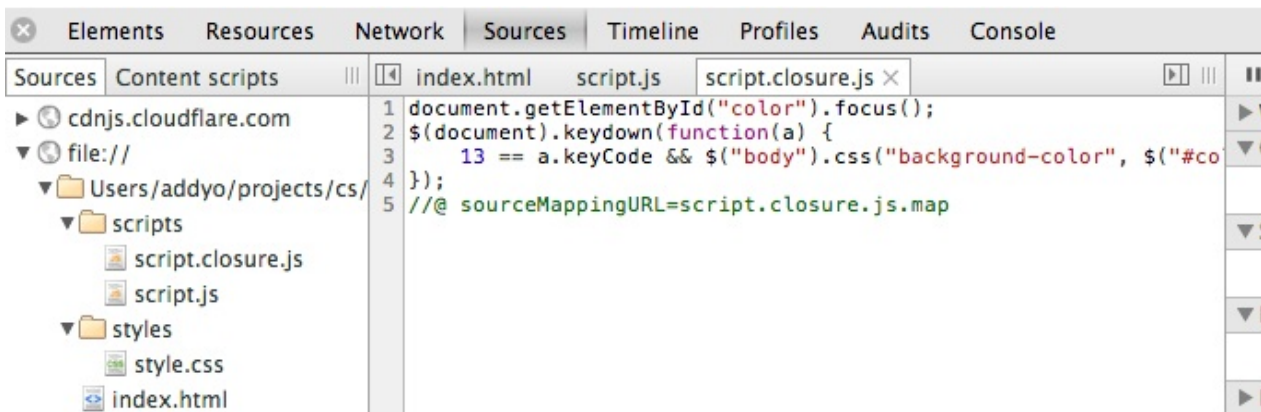
- 打开设置 cog > General
- 选择 "Enable source maps"



接下来：

- 将你的CoffeeScript 转换成 JavaScript，运行: `coffee -c myexample.coffee`
- 安装 [CoffeeScript Redux](#)
- 创建一个源码地图文件 `example.js.map` 用来存储所有的代码映射信息: `$ coffee-redux/bin/coffee --source-map -i example.coffee > example.js.map`

- 确认生成的 JavaScript 文件, `example.js`, 在文件的末尾有下面的url: `//# sourceMappingURL=example.js.map`



现在你可以调试CoffeeScript代码了, 通过上面的声明语句, DevTools就知道源码在哪里了。

你可以继续使用这个源码地图, 在代码压缩的阶段, 使用像UglifyJS2的工具来引用该文件, 将压缩后的JS代码映射回到CoffeeScript代码而不是JavaScript编译后的输出。

这样你就可以直接调试生产环境的代码然后定位到CoffeeScript源码。

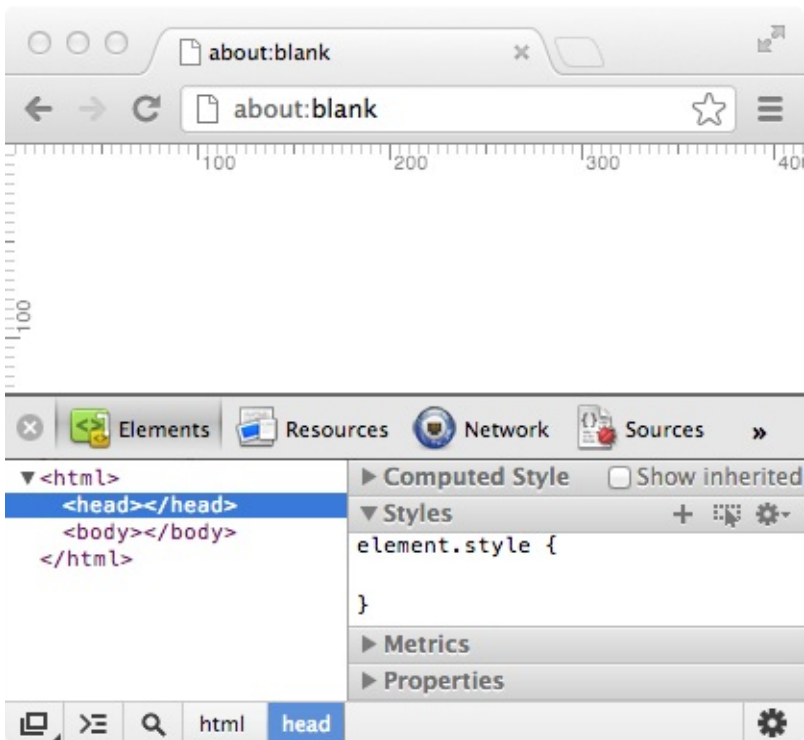
[扩展阅读:Source Maps 101](#)

想了解更多关于工作流, 请自觉前往[开发工作流](#)一章。

建议和技巧——Elements篇

使用标尺

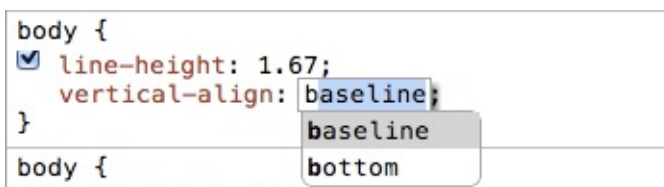
在 Settings > General > Show rulers a ruler 中可以开启标尺，当鼠标悬停在元素上面或者将其选中，标尺就会显示自动显示出来。



CSS属性自动补全

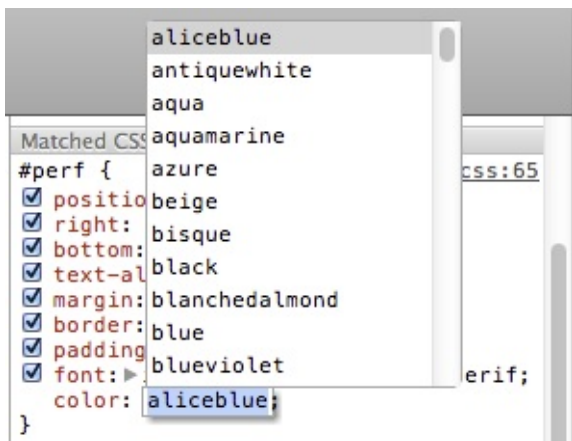
DevTools支持CSS属性名和值的自动补全（包括带前缀的），在补全列表中你就可以看到当前元素是可以设置哪些属性的。

当你输入属性名或者其值的时候，DevTools会自动给出提示，使用上下箭头可以在提示列表中进行选择。给元素添加属性后，页面就会立刻生效。

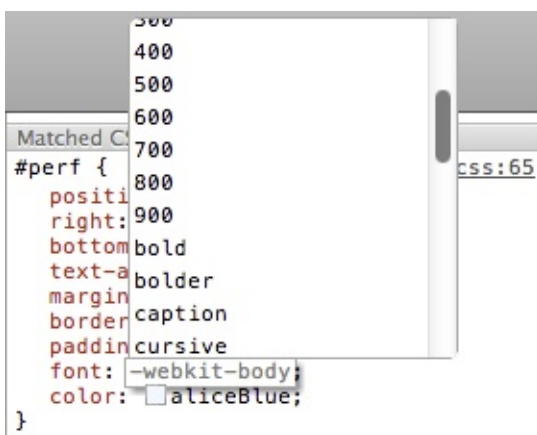


在 Styles 面板中，可以选择CSS颜色值，格式包括 名字(例如'red'), HSL, HEX或者RGB。按住 shift 键，在颜色上单击鼠标可以进行不同格式之间的切换。

如果你希望显示一个属性所有支持的值，使用 `Ctrl + space`。

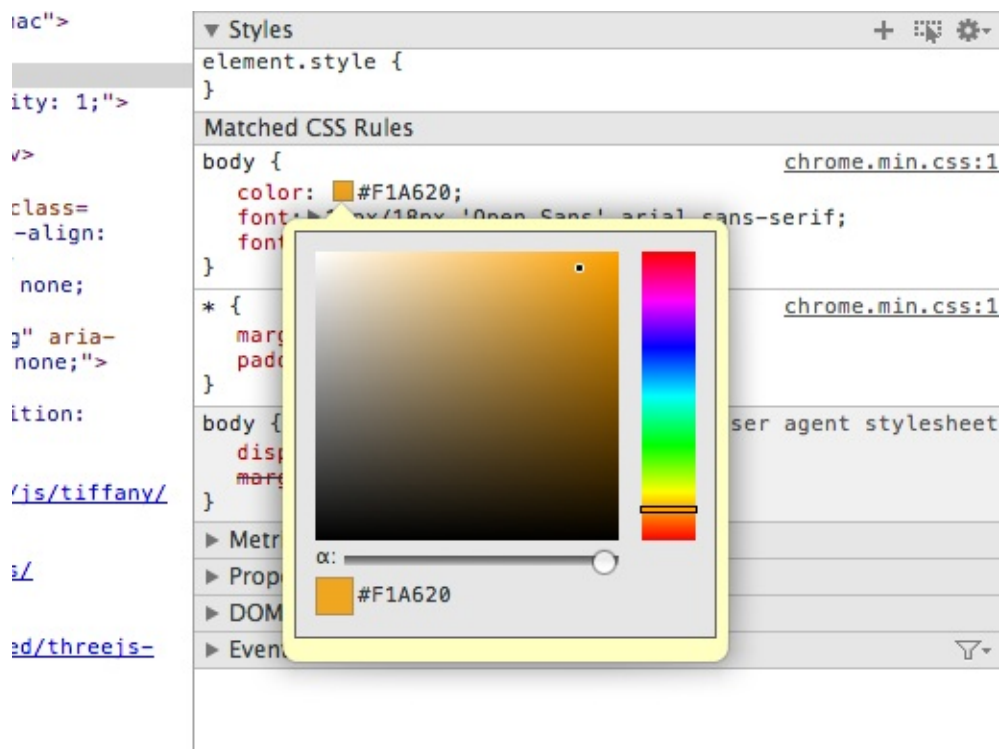


这个提示列表与具体的属性类型相关（例如font），数字和带前缀的数值也支持。



颜色选择器

DevTools内置了一个颜色选择器，用鼠标单击颜色的小方块，就会开启选择器。使用选择器可以选择页面中的任何颜色。（译注：这个非常给力，尤其是参考某个页面的时候）



按住 **shift** 并单击颜色值，可以改变颜色的格式。

增加 CSS 样式

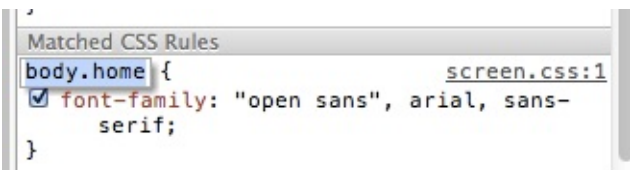
在一个css规则中大括号中间的任何位置单击鼠标都可以为当前元素添加一个新的CSS属性，添加后属性会立刻生效。

```
div {
  border: solid black 10px;
  line-height: 1.5;
  font-family: "Arial", sans-serif;
  color: #F31A1A;
  font-size: 81.25%;
}
```

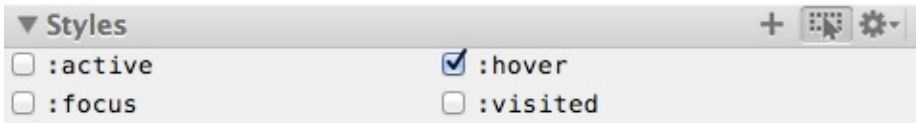
添加完一个属性后，按 **tab** 键可以继续输入下一个属性。

在Style面板的右侧点击加号按钮，可以添加新的选择器。

注意：其实可以直接编辑选择器，在CSS选择器上单击，就可以直接修改选择器名字，一旦完成修改，之前选择器对应的属性就会应用到新的选择器上。



新的伪类选择器同样使用类似的方式添加。注意，单击右侧的"toggle element states"按钮（加号的旁边），可以开启"Force element state"功能。这个功能非常给力，勾选对应状态，就可以强制的给当前选中的元素加特技（伪类）了。

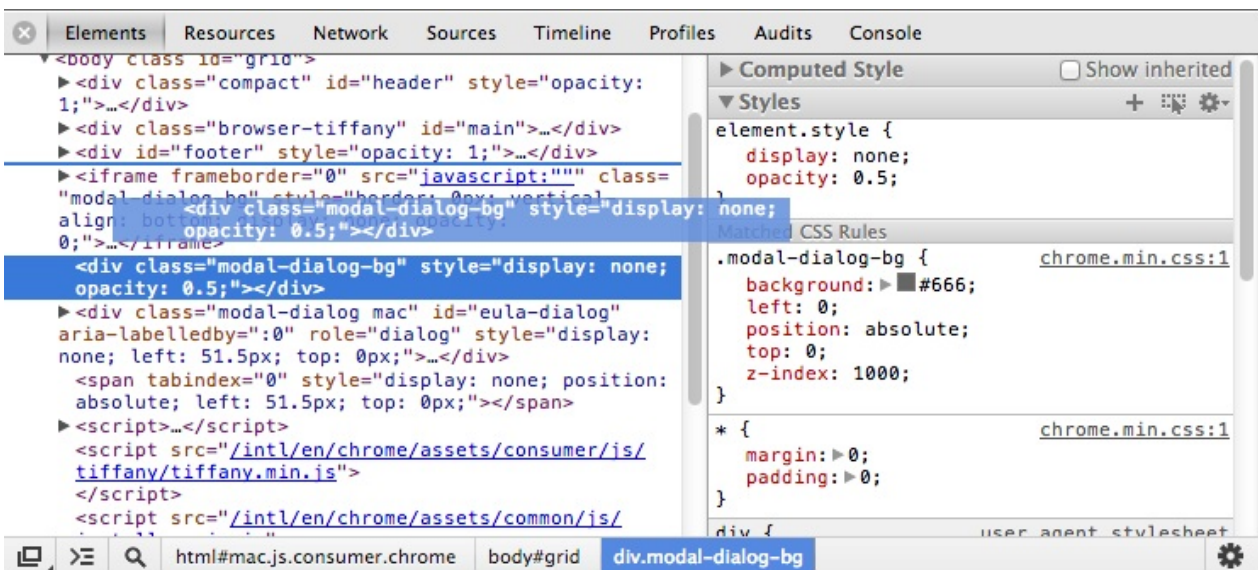


回到“Matched CSS Rules”面板，点击CSS规则旁边的链接，可以直接定位到具体CSS文件的某一行。



拖拽页面元素

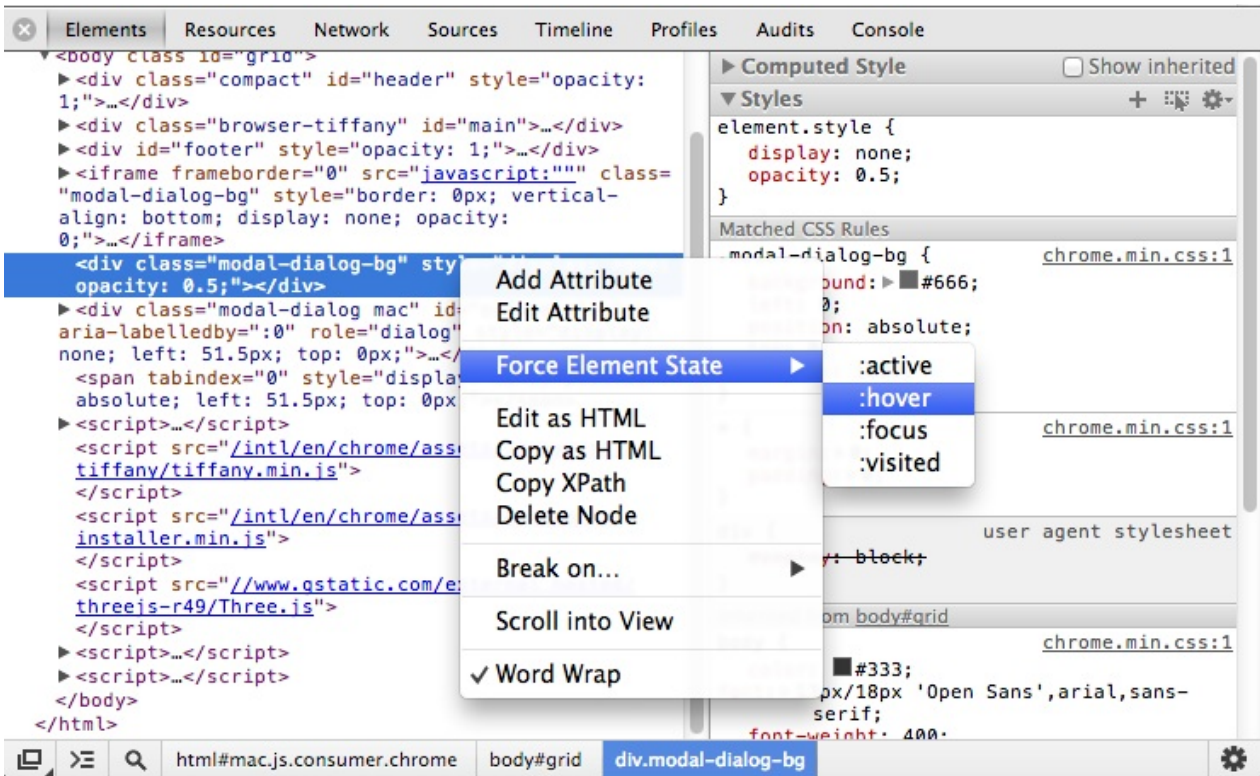
在 Elements 面板里，可以拖拽一个元素来随意调整它的位置。



强制元素状态

是不是有时希望强行的改变一个元素的状态？

- 在一个子元素上右键，选择‘审查元素’
- Elements面板中，在父元素上右键选择"Force Element State"
- 你可以任意选择 `:active`，`:hover`，`:focus` 或者 `:visited`



编写调试Sass

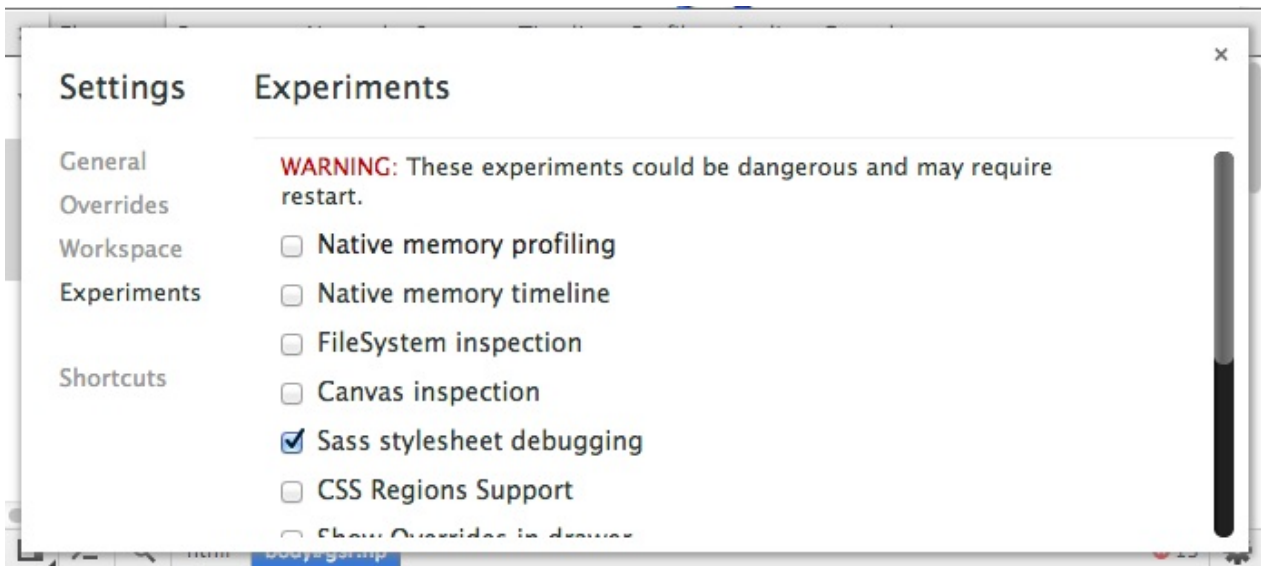
注意: 要在Chrome中调试Sass 需要3.3.0 (pre-release) 以上的 Sass 编译器, 支持source map生成。

处理一个带有预处理CSS的页面比较棘手, 以为在DevTools中对CSS样式的修改一般不会更新Sass源码文件。这意味着如果想即时的改变样式, 你需要手动的去在其他编辑器中修改源码文件。

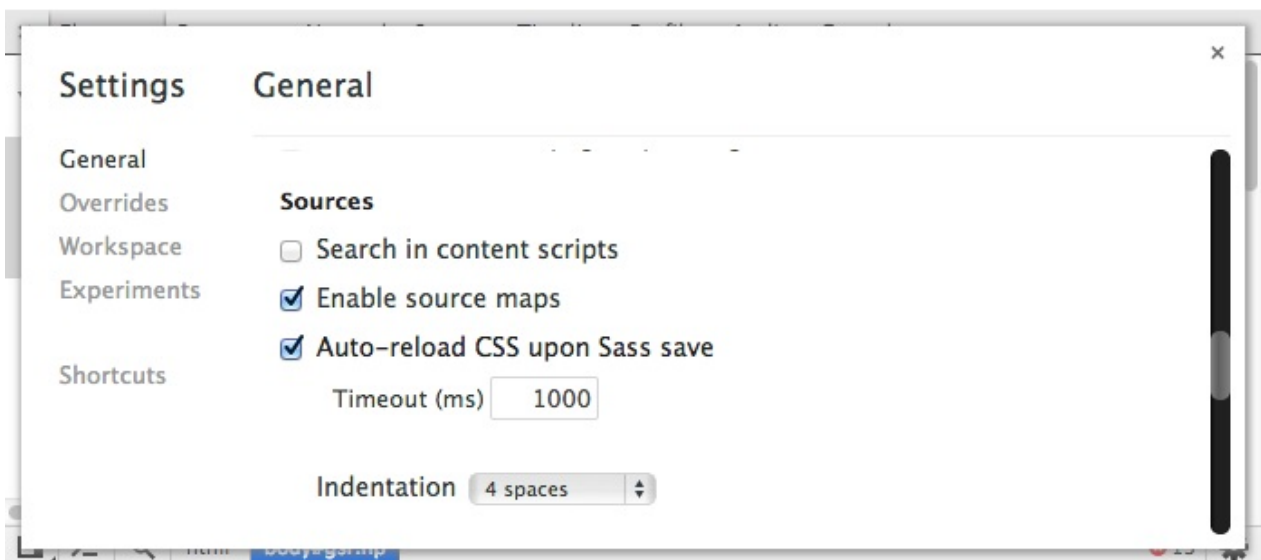
最新的Sass工作流中这已经不再是一个问题, 如果想支持Sass:

[1] 确认你的DevTools已经开启了实验室功能 (译注: 在地址栏输入 `chrome://flags/` 开启工具实验室功能)

[2] 接着, 前往设置页面, 选择 Settings cog > Experiments 并勾选 “Sass stylesheet debugging” (译注: 最新版本的Chrome已经将该功能移除实验室, 可以忽略本步骤)



[3] 前往 General menu > Settings > 勾选“Enable source maps”和“Auto-reload CSS upon Sass save”



timeout参数可以使用默认值。这取决于Sass编译器多久可以完成编译，你也可以禁用自动加载，而在需要的时候手动去刷新。

[4] 打开你想调试的页面

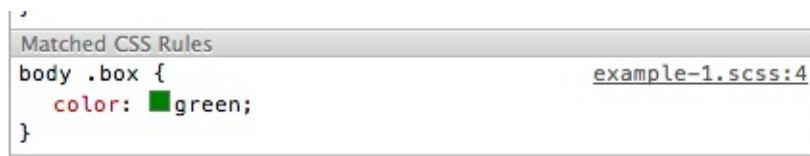
[5] 接下来，开启Sass编译器，使用如下命令 `sass --watch --sourcemap`

`sass/styles.scss:styles.css`，编译器会监测源码文件是否发生变化，并为每个生成的CSS文件创建source map，如下面的控制台所示：

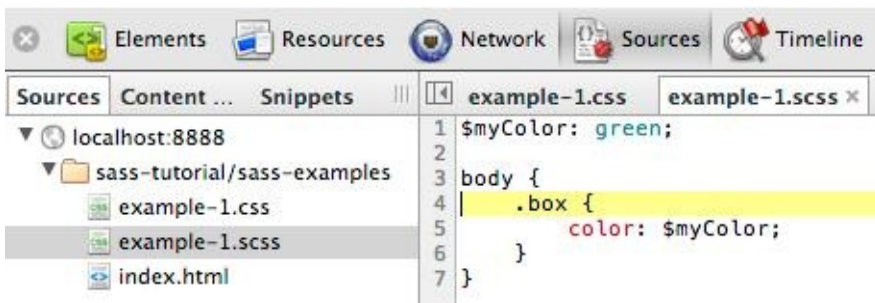
```
>>> Sass is watching for changes. Press Ctrl-C to stop.  
  overwrite example-1.css  
  overwrite example-1.css.map
```

确认Sass调试正确工作

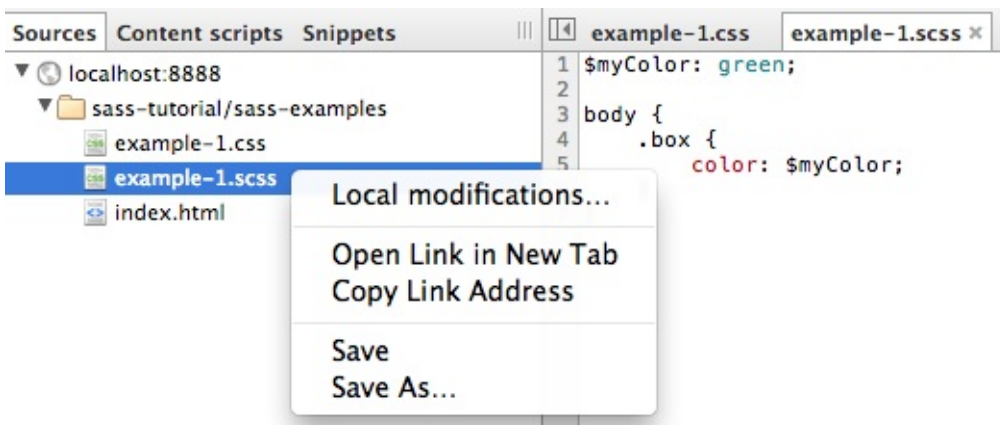
[6] 如果正确配置完成，可以在Elements面板看到，每个样式后面对应的链接已经是 `.scss` 文件以及具体的代码行号了。



[7] 点击文件名，就可以直接到Source面板中对应的源码行中，现在就可以直接在DevTools中的语法高亮编辑器中工作了。



[8] 如果你希望在Source面板中编辑Sass文件，只要确认DevTools能够找到源文件的磁盘位置。在编辑器中右键选择"Save as",可以用现在编辑的文件去覆盖源文件。DevTools支持文件自动加载，这样修改可以在Chrome中立刻生效了。

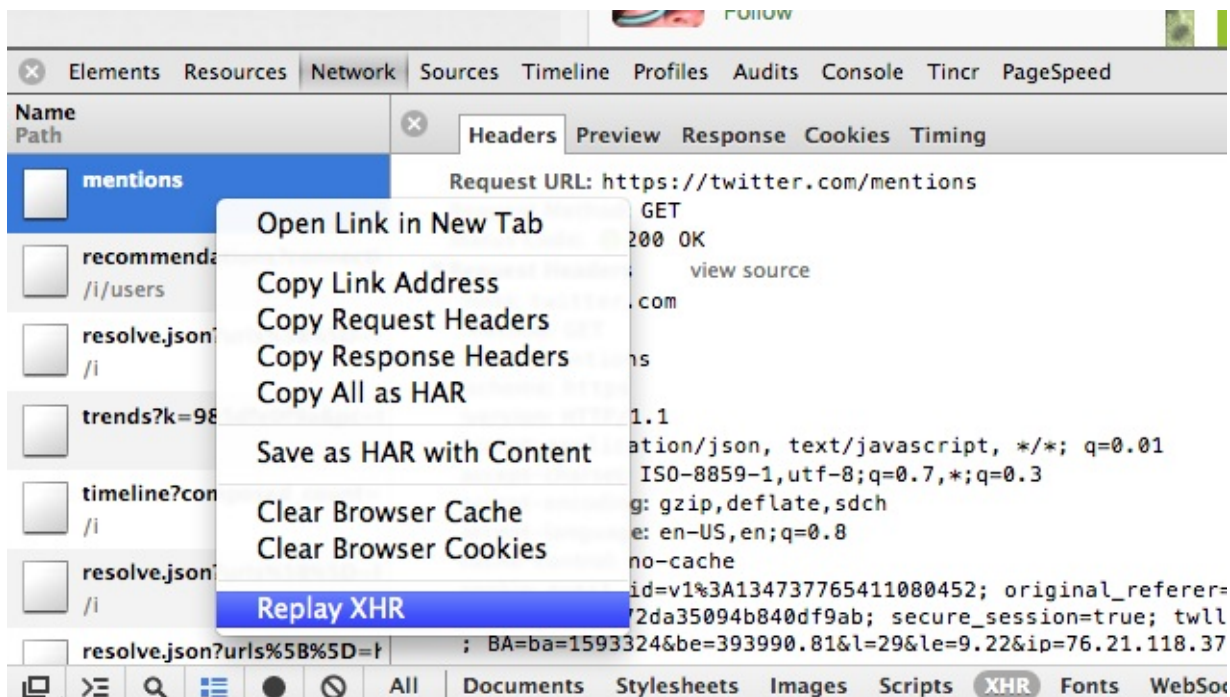


想了解更多关于页面元素和样式的使用技巧，请自觉前往[修改DOM样式](#)一章。

建议和技巧——Network&Setting篇

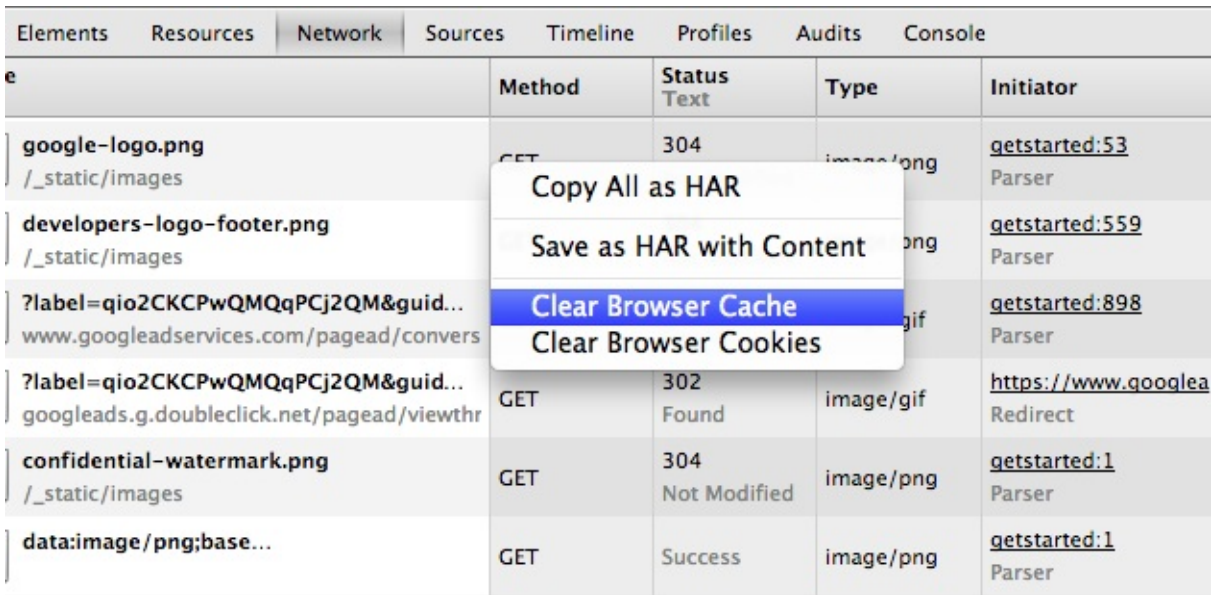
重发异步请求

Network面板中会列出页面中所有的异步请求，在任何一条GET或POST请求上右键选择“Replay XHR”可以重新发送对应的异步请求。



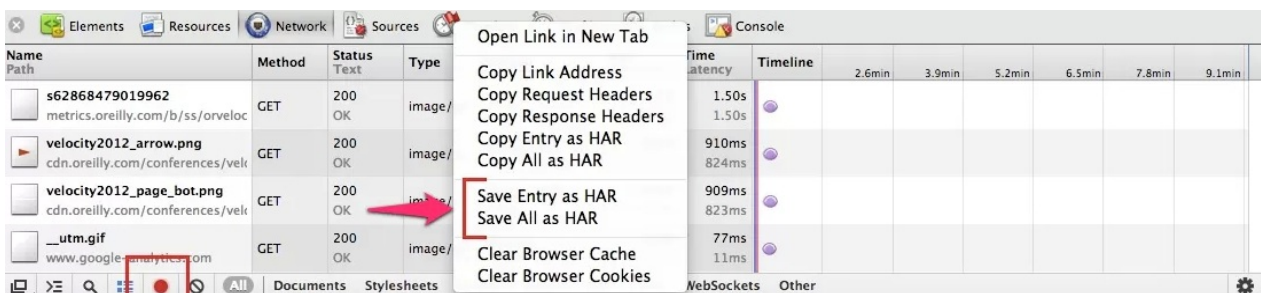
清空网络缓存或cookies

在Network面板中右键，选择“Clear Browser Cache/Network Cache”。



记录轨迹 & 导出瀑布流

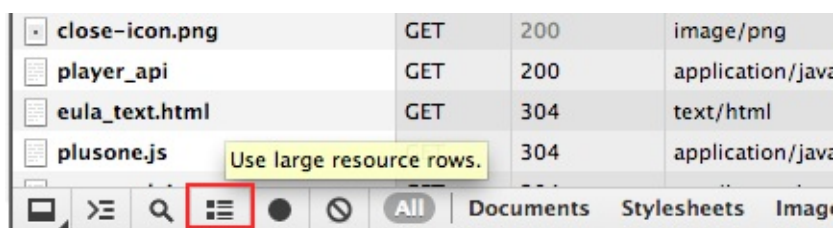
- 点击 "record" 开始记录多页面请求
- 导出请求数据: 右键选择 "Copy Entry as HAR"
- 导出整个瀑布流: 右键, "Copy All as HAR"



扩展阅读：等等，DevTools还能做这个？| lgvita.com

使用大视图查看更多网络细节

点击Network面板下方的 "Use large resource rows" 可以开启大的视图来查看更多网络请求信息。



小视图：

Name	Method	Status	Type	Initiator	Size	Time	Timeline
chrome/	GET	200	text/html	Other	6.6 KB	151 ms	
chrome.min.css	GET	200	text/css	www.google.c...	0 B	135 ms	
css?family=Open+Sans:300...	GET	200	text/css	www.google.c...	885 B	193 ms	
tiffany.css	GET	200	text/css	www.google.c...	0 B	141 ms	
tiffany-view.js	GET	200	text/javascript	www.google.c...	0 B	120 ms	

大视图：

Name Path	Method	Status Text	Type	Initiator	Size Content	Time Latency	Timeline
chrome/ /intl/en	GET	200 OK	text/html	Other	6.6 KB 35.9 KB	151 ms 149 ms	
chrome.min.css /intl/en/chrome/assets/comi	GET	200 OK	text/css	www.google.c... Parser	0 B 106 KB	135 ms 125 ms	
css?family=Open+Sans:3... fonts.googleapis.com	GET	200 OK	text/css	www.google.c... Parser	885 B 1.1 KB	193 ms 191 ms	
tiffany.css /intl/en/chrome/assets/cons	GET	200 OK	text/css	www.google.c... Parser	0 B 14.1 KB	141 ms 121 ms	
tiffany-view.js /intl/en/chrome/assets/cons	GET	200 OK	text/javascript	www.google.c... Parser	0 B 120 KB	120 ms 72 ms	

获取更多信息的技巧

在Network面板中 Timeline 一列，单击标题可以在切换不同的数据视角：

- Timeline（时间线）
- Start Time（开始时间）
- Response Time（响应时间）
- End Time（结束时间）
- Duration（耗时）
- Latency（延迟）

x Elements Resources Network Sources Timeline Profiles Audits Console							
Name Path	Method	Status Text	Type	Initiator	Size Content	Time Latency	Timeline
	GET	Success	Image...	Parser	1.1 KB	8 ms	
modernizr.custom.824... 1-ps.googleusercontent.c...	GET	200 OK	applic...	www.html5r...	5.5 KB	395 ms	
jquery.min.js ajax.googleapis.com/aja...	GET	200 OK	text/j...	www.html5r...	34.0 KB	395 ms	
handlebars-1.min.js.p... 1-ps.googleusercontent.c...	GET	200 OK	applic...	www.html5r...	10.5 KB	394 ms	
persona.min.js.pagesp... 1-ps.googleusercontent.c...	GET	200 OK	applic...	www.html5r...	1.5 KB	388 ms	
app.min.js.pagespeed... 1-ps.googleusercontent.c...	GET	200 OK	applic...	www.html5r...	2.4 KB	389 ms	
				Parser	3.8 KB	385 ms	

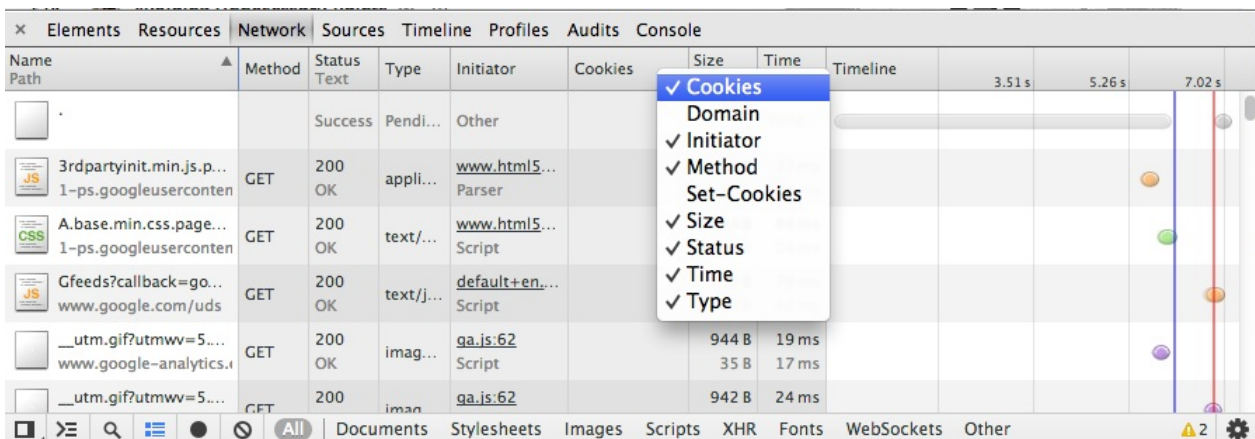
查看灰色背景的文本，了解下面的信息：

- 每个请求的HTTP头是什么？
- 每个请求的首字节响应时间是多少？
- 哪个资源响应时间最慢？

- 哪个资源耗时最久？

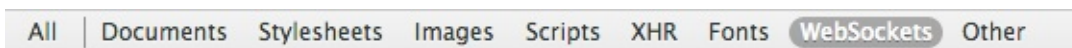
在每一列的标题栏右键，可以选择是否显示该列，其中有三个是默认不显示的：

- Cookies
- Domain
- Set-Cookies

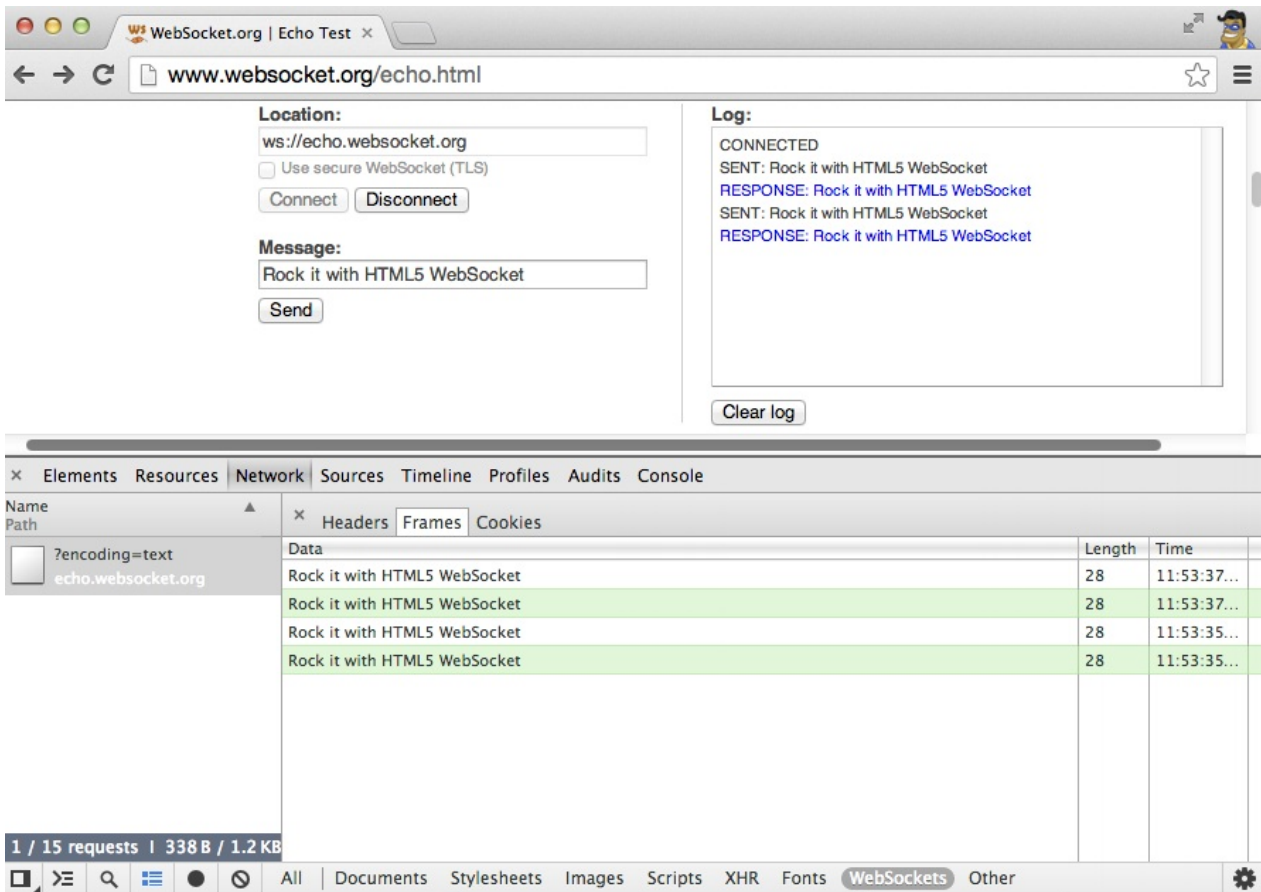


WebSocket 审查

在Network 面板, 你可以在窗口底部的过滤器选择审查 WebSocket 报文。



例如，打开 [Echo demo](#)，在Network面板底部选择"WebSockets"，点击"Connect"按钮。任何通过点击"Send"按钮发送的消息都可以在"Frames"子面板中检测到。



绿色的部分表示你的客户端发送的消息。WebSocket审查中既可以审查WebSocket握手，也可以看到独立的WebSocket数据帧。

扩展阅读：等等，DevTools还能做这个？

扩展阅读：使用DevTools审查WebSocket

查找过滤异步请求

在Network面板中，你经常需要用键盘上下寻找特定请求。这时可以通过快捷键 `Ctrl + F` 或者 `Cmd + F`。

在输入框里，输入想要匹配文件名或url的关键字，结果就会被高亮出来。你可以使用上下箭头在结果间进行切换。

×

Elements

Resources

Network









Sources

Timeline

Profiles

Audits

Console

Name	Path	Met...	Status	Text	Type	Initiator	Size	Conter	Time	Latenc	Timeline	1.1 min	1.6 min	2.1 min
	cnn.com	GET	301	Moved	text...	Other	261 B	0 B	247 ...	247 ms				
	www.cnn.com	GET	200	OK	text...	http://cn... Redirect	27.2...	106 KB	462 ...	260 ms				
	hplib-min.css z.cdn.turner.com/cn	GET	200	OK	text...	www.cnn... Parser	32.4...	188 KB	1.23 s	952 ms	 282 ms			
	jquery-1.7.2.min.js z.cdn.turner.com/cn	GET	200	OK	appl...	www.cnn... Parser	33.2...	92.6 Ki	1.26 s	966 ms				

jquery| 1 of 3

Filter Cancel

>

All

Documents

Stylesheets

Images

Scripts

XHR

Fonts

WebSockets







Ot

16

1


选中“Filter”可以只显示那些匹配到的请求，这样可以排除多余信息的干扰。

ElementsResourcesNetworkSourcesTimelineProfilesAuditsConsole

Name Path	Met...	Status Text	Type	Initiator	Size Conter	Time Latenc	Timeline	1.1 min	1.6 min	2.1 min
 jquery-1.7.2.min.js z.cdn.turner.com/cn	GET	200 OK	appl...	www.cnn... Parser	33.2... 92.6 Kf	1.26 s 966 ms				
 jquery.timeago.m... z.cdn.turner.com/cn	GET	200 OK	appl...	www.cnn... Parser	1.5 KB 3.0 KB	974 ... 966 ms				
 topics?callback=j... trends.cnn.com	GET	200 OK	appl...	jquery-1... Script	23.9... 85.6 Kf	724 ... 599 ms				

3 / 195 requests | 58.6 KB / 1.3 MB transferred | 2.1 min (onload: 7.28 s, DOMContentLoaded: 5.71 s)

jquery☒ Filter

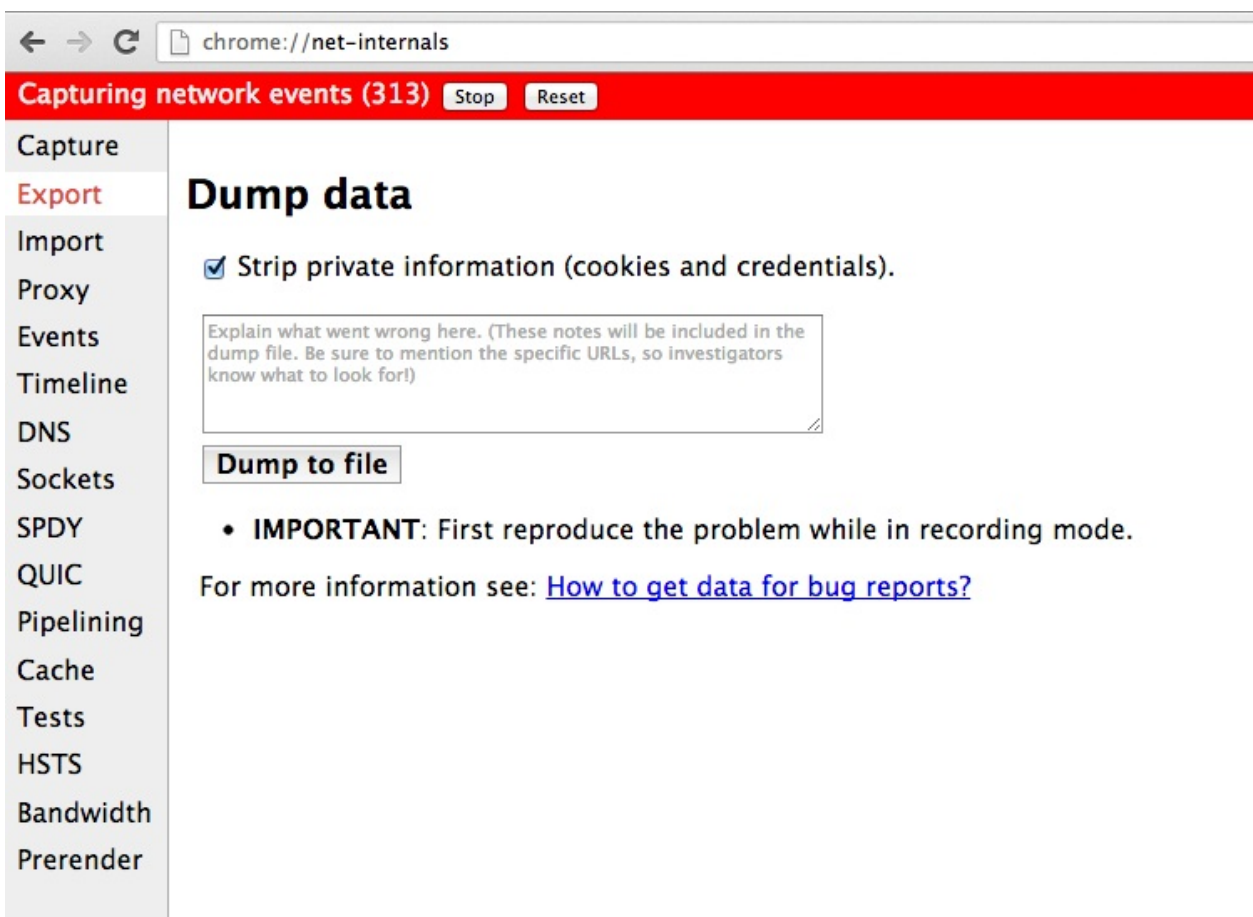


AllDocumentsStylesheetsImagesScriptsXHRFontsWebSocketsOt161

扩展阅读：计算网络性能

获取网络栈内部状态数据

"about:net-internals" 页面是一个特殊的URL，在该页面中可以收集网络栈内部状态数据。这些数据可以用来处理网络性能和连接问题，包括了请求性能信息，代理设置和DNS缓存。



注意，`about:net-internals/#tests` 也支持指定URL的测试。

关于**Network**更加详细的内容，请参考[评估网络性能](#)

设置

模拟触摸事件

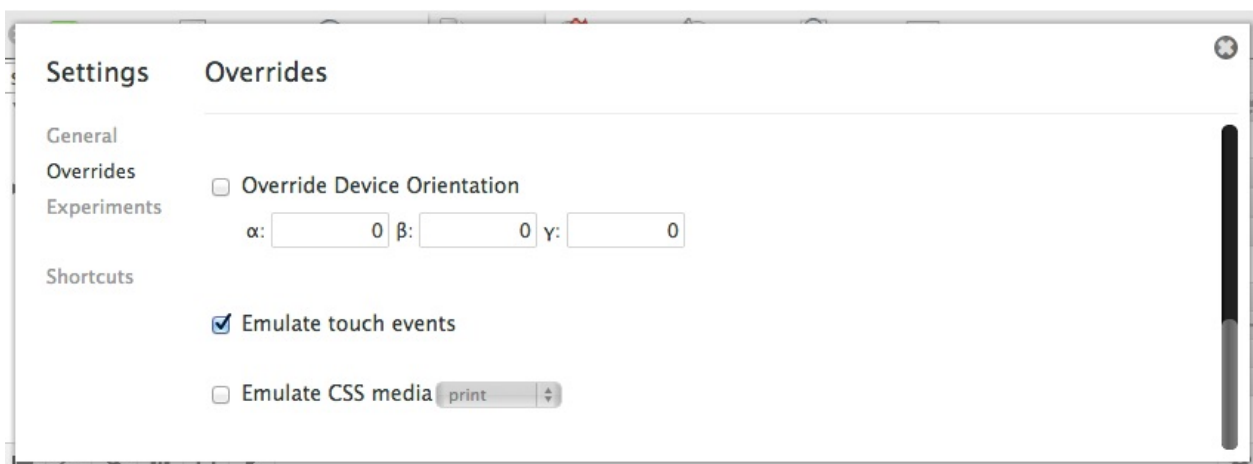
触摸这种输入方式在桌面上很难测试，因为大部分桌面系统都不支持触摸。然而在移动端测试又会导致开发周期变长，因为每次变更都需要将修改推送到服务端，然后再到设备上看到效果。

解决的方案是在开发机上模拟触摸时间。Chrome DevTools支持单点触控，来使得调试移动应用更加容易。

要支持触摸模拟：

- 打开 `overrides` 设置选项卡
- 勾选“Enable touch events”

（译注：新版本调整了设备模拟选项，在任一面板，按Esc键调起控制台，让后选择Emulation选项）



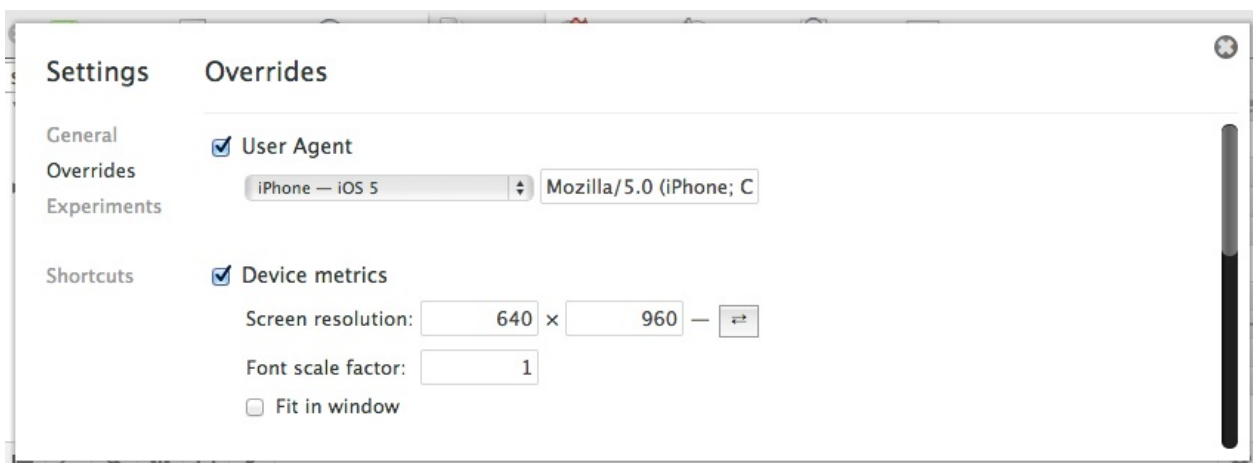
现在你可以在代码中设置断点，通过标准的桌面事件来调试触摸事件了。

扩展阅读：[DevTools设备模式](#)

模拟设备UA以及视图

在开发移动页面时，通常我们会现在桌面上制作原型，然后去处理那些需要在移动设备上特殊显示的部分。现在通过设备的模拟，工作就可以变得更加简单了。

DevTools设备模拟支持不同设备类型和屏幕尺寸。（译注：新版本的Chrome可以直接点击手机图标，更加简单的模拟设备型号、屏幕尺寸以及网络）



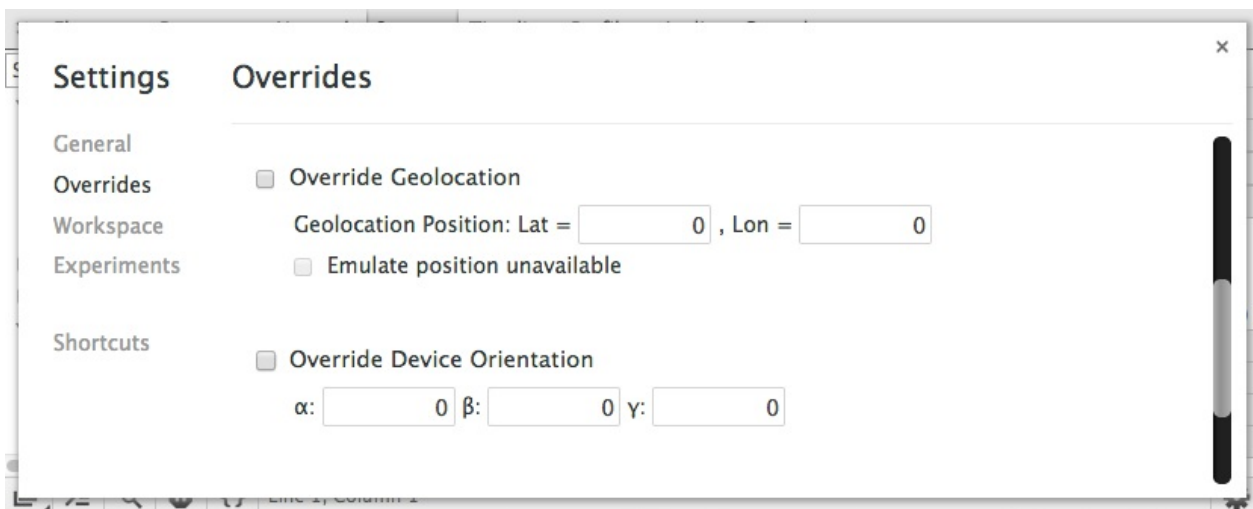
现在你可以模拟在 Galaxy Nexus 和iPhone这样的设备上进行测试了。

扩展阅读：[DevTools设备模式](#)

模拟地理位置

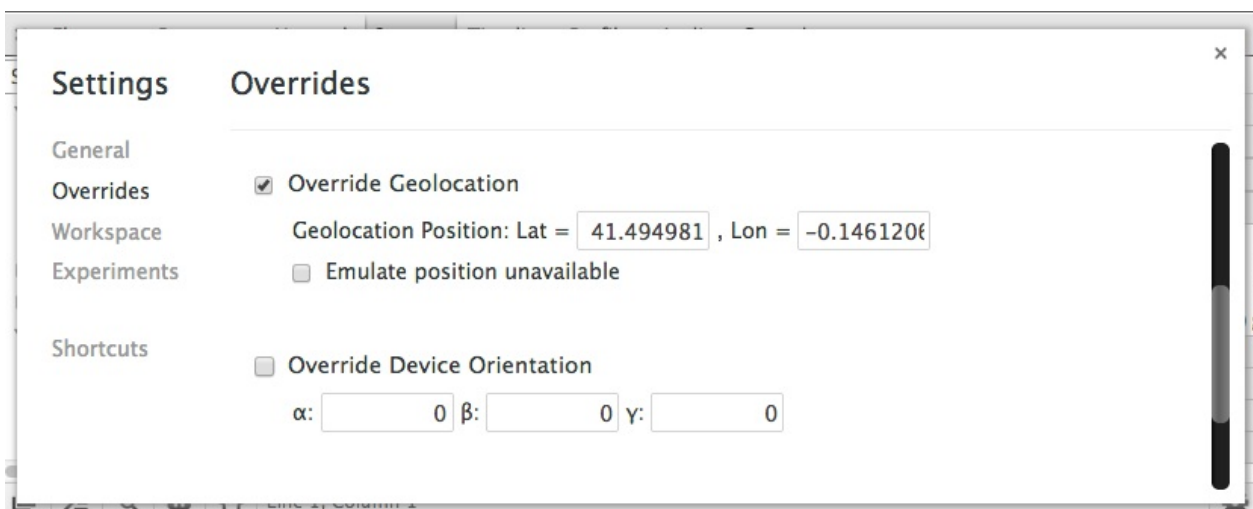
在调试使用HTML5 geolocation的应用时，常常需要模拟不同的经度和纬度的输入。

DevTools可以模拟 `navigator.geolocation` 的数值，也可以模拟位置不可用的情形。



覆盖地理位置：

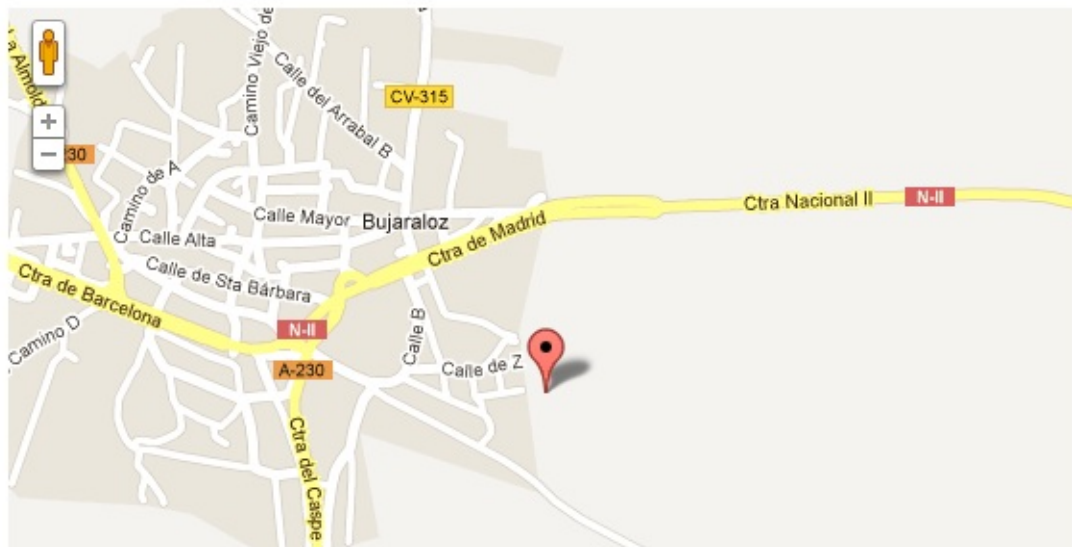
- 打开 [Geolocation demo](#)
- 允许页面使用地理位置
- 打开设置菜单中的 overrides部分（译注：新的位置在控制台窗口中，通过Esc打开）
- 勾选“Override Geolocation” 然后输入 Lat = 41.4949819 and Lat = -0.1461206



刷新页面后，demo会使用你输入的新的地理位置。

geolocation

Finding your location: **found you!**



1.再勾选“Emulate position unavailable”选项 2.刷新页面，demo会通知你寻找你的位置失败了。

geolocation

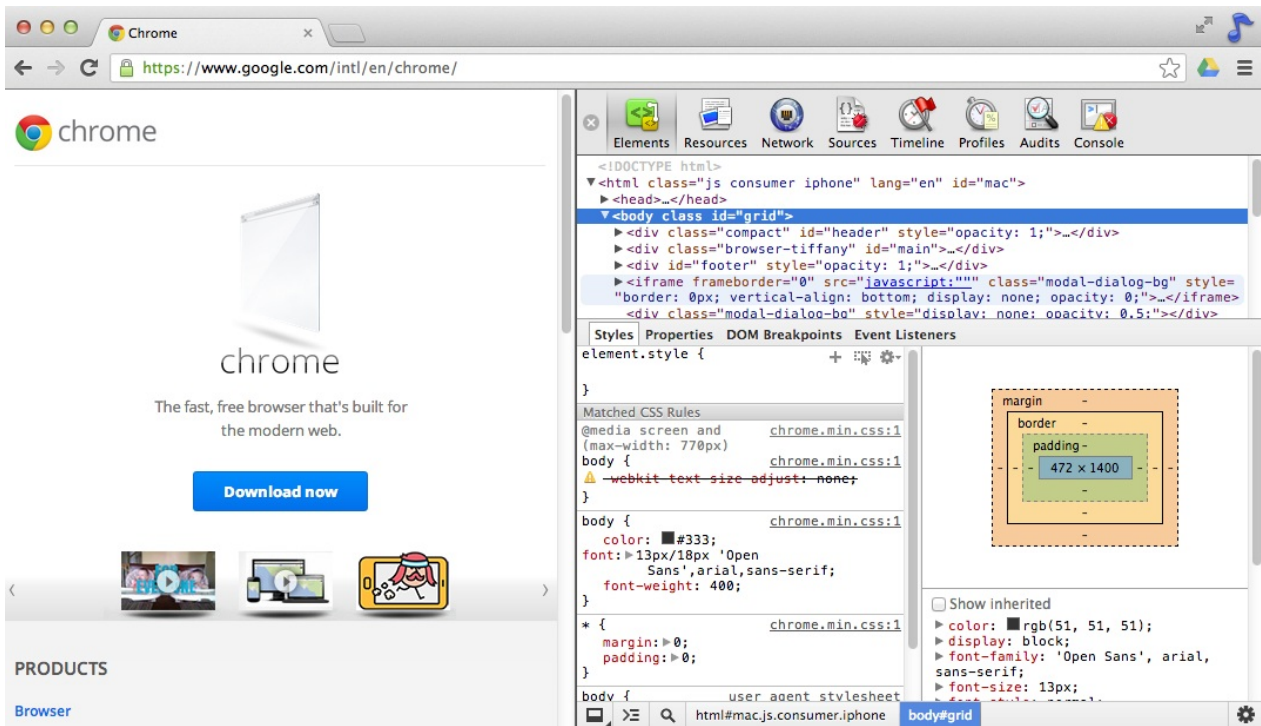
Finding your location: **failed**

扩展阅读：[DevTools设备模拟](#)

使用Dock-to-right模式调试视图

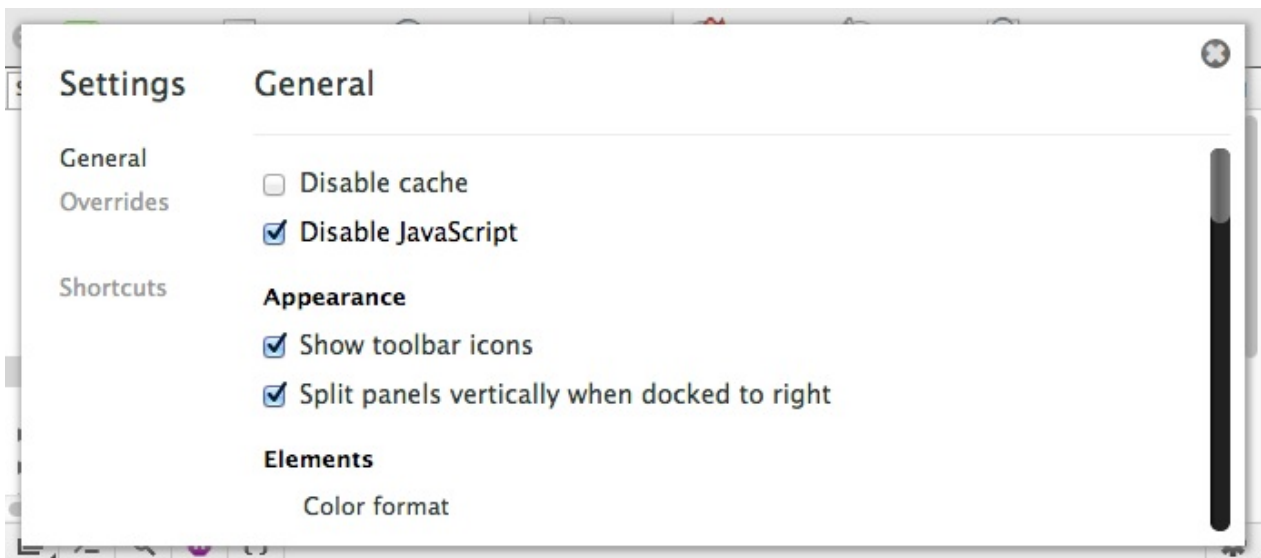
Dock-to-right 模式在调试小屏幕页面时非常有用：

- 长按布局切换按钮
- 左右分栏后，可以拖拽分栏位置来调整视图的宽度。



禁用JavaScript

进入 Settings -> General 勾选 “Disable JavaScript” 可以禁用JavaScript。当DevTools打开并且该选项被选中，当前页面的JavaScript就会被禁用。



常用

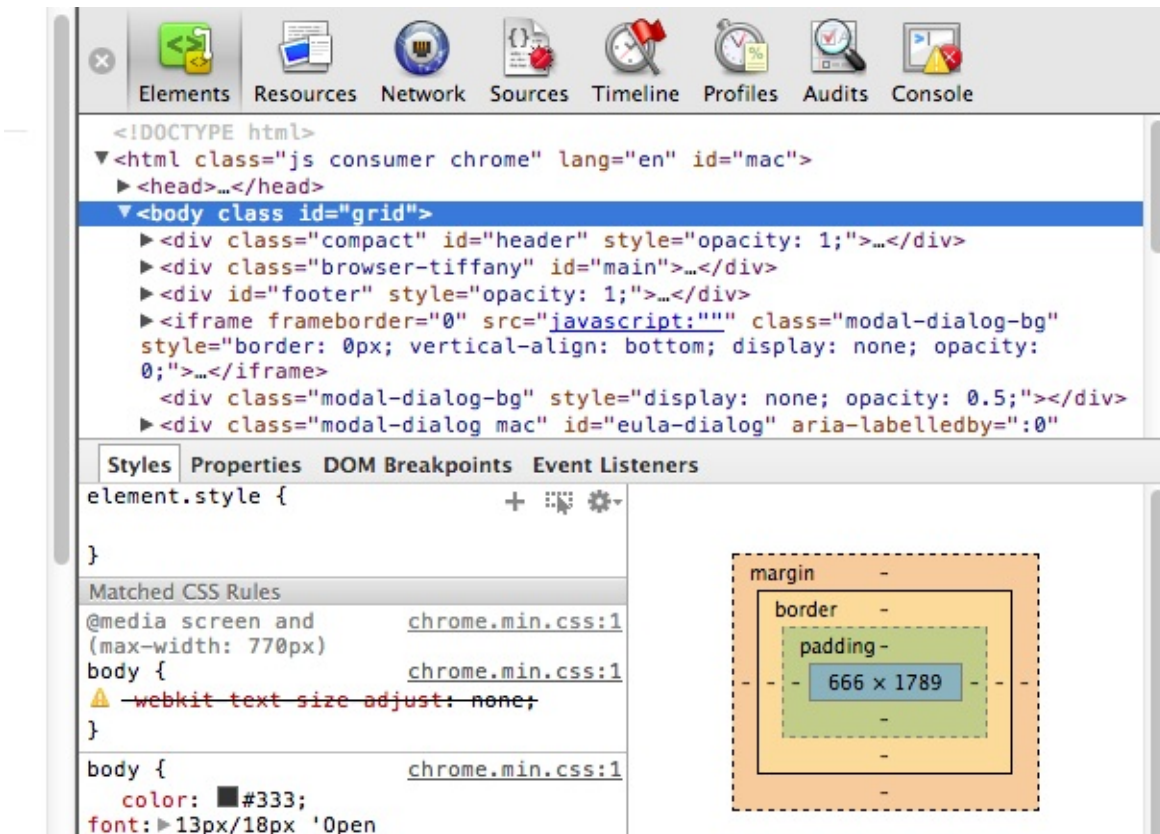
快速在**tab**间切换

使用 `Cmd +]` 和 `Cmd + [` (或 `Ctrl +]` 和 `Ctrl + [`) 可以快速的在DevTools中不同的面板之间进行切换，省时省力。

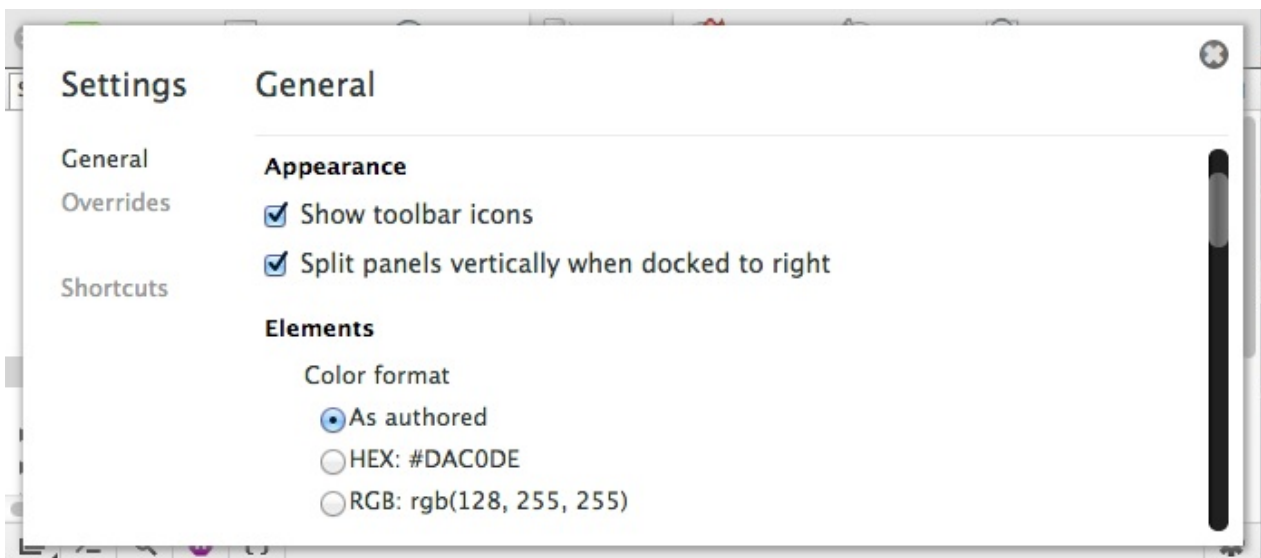


获得更好的左右分栏体验

新版本的Chrome在左右分栏的时候，会对DevTools窗口进行上下的拆分，便于查看信息。



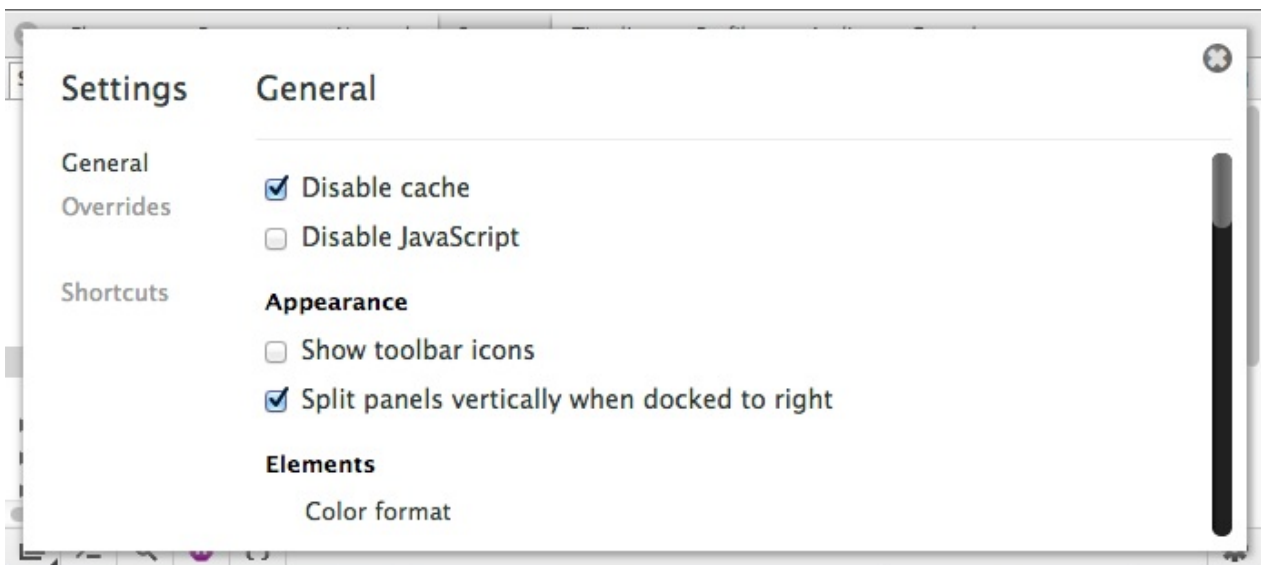
当然，如果你的屏幕很宽，直接在设置中取消勾选“Split panels vertically when docked to right”就可以了。



[扩展阅读：3步获取更好的分栏界面](#)

禁用Cache

在设置界面, 你可以勾选'Disable cache'从而禁用Cache.这在调试的过程中太有用了, 但是要注意只有在DevTools打开状态下才会有效的。



审查 Shadow DOM

拥有Shadow DOM 的元素会在Elements面板中显示

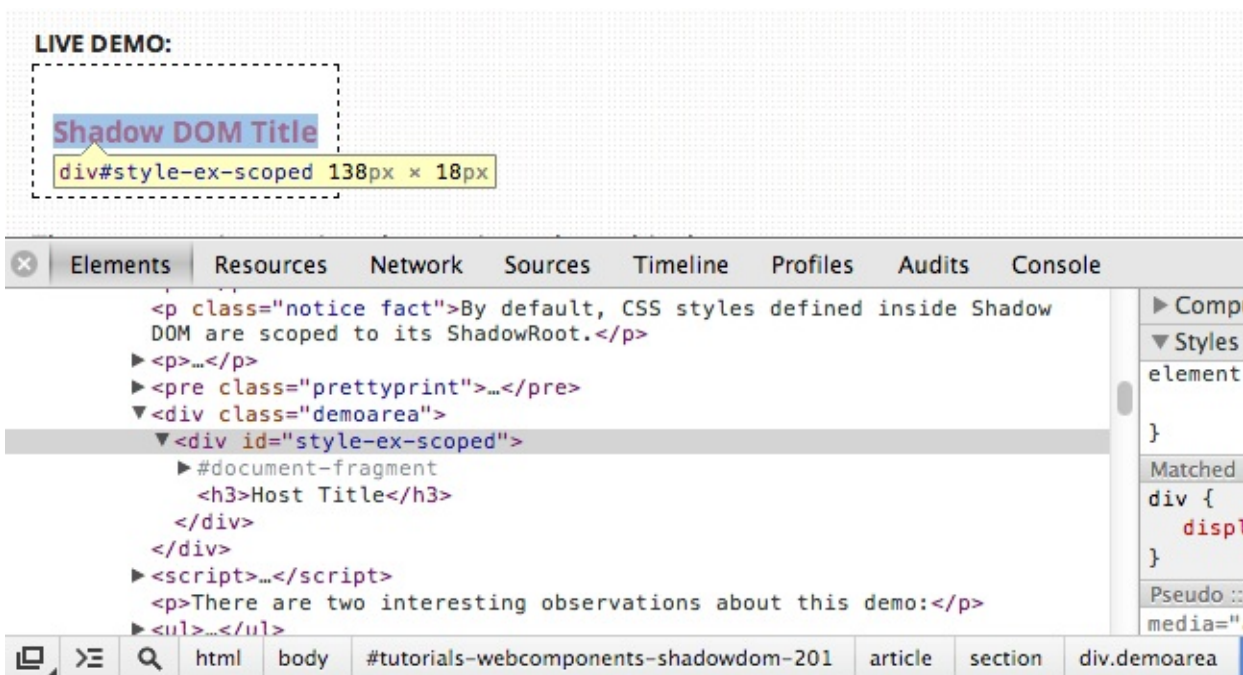
- 在设置中勾选 'Show Shadow DOM'
- 重启 DevTools

General

☒ Show Shadow DOM

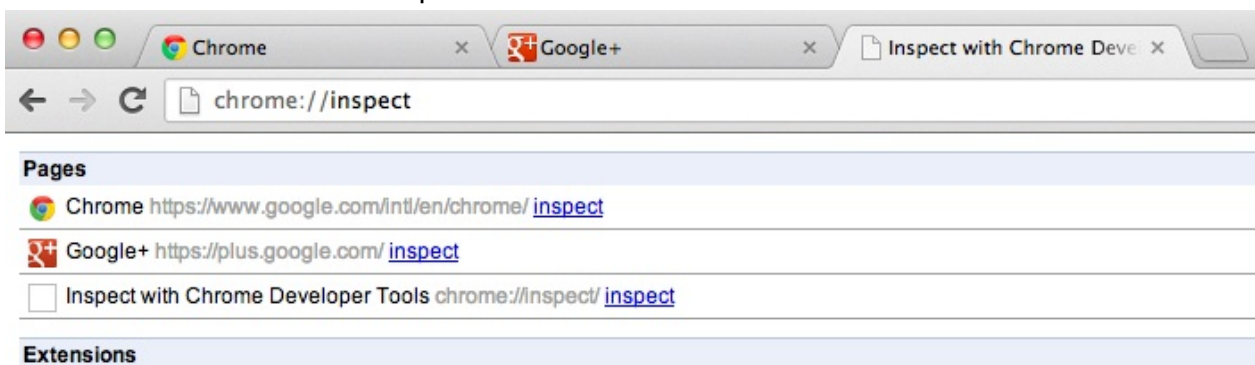
☐ Show rulers

你可以查看 Shadow DOM了，例如 [html5 Rocks](#)这个 [Title](#)



预览所有可调式页面

如果你经常使用远程调试，你可以在地址栏打开“`about:inspect`”，这样可以看到所有可调式的标签页或者扩展程序。点击'inspect'可以打开对应的页面并启动DevTools。



查看哪些站点有应用缓存

在地址栏访问“about:appcache-internals”，可以查看站点的应用缓存信息。你可以查看哪些站点有缓存，这些站点的修改时间以及所占的空间大小，也可以在该页面中移除某些站点缓存。

chrome://appcache-internals

Manifest: https://docs.google.com/a/google.com/document/offline/manifest?uc=en_GB.i&jobset=canary&ftrack=1

[Remove](#) [View Entries](#)

- Size: 3.6 MB
- Creation Time: Friday, February 22, 2013 12:02:53 PM
- Last Update Time: Wednesday, March 13, 2013 4:15:52 PM
- Last Access Time: Friday, March 15, 2013 10:24:17 AM

Manifest: https://docs.google.com/a/google.com/document/offline/manifest?uc=en_GB.i&jobset=prod

[Remove](#) [View Entries](#)

- Size: 3.6 MB
- Creation Time: Friday, February 22, 2013 12:21:12 PM
- Last Update Time: Tuesday, March 12, 2013 9:00:41 PM
- Last Access Time: Friday, March 15, 2013 10:23:54 AM

Manifest: https://docs.google.com/a/google.com/document/offline/manifest?uc=en_GB.i&jobset=prod&ftrack=1

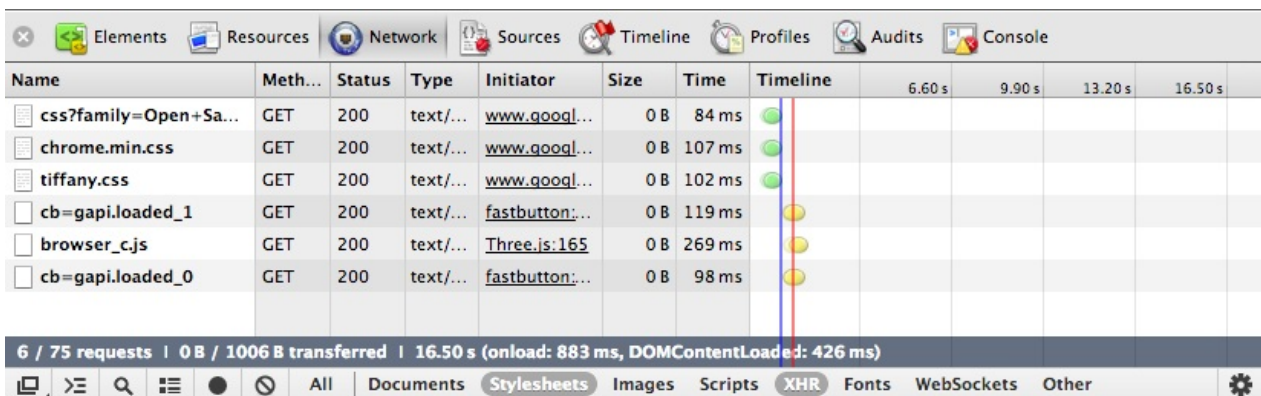
[Remove](#) [View Entries](#)

- Size: 3.6 MB
- Creation Time: Friday, February 22, 2013 12:12:55 PM
- Last Update Time: Tuesday, March 12, 2013 9:00:40 PM
- Last Access Time: Friday, March 15, 2013 10:23:54 AM

Network/Console 面板多过滤器

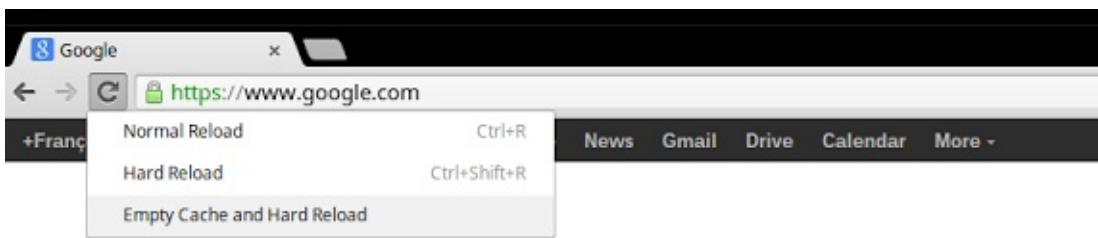
在Network面板和Console面板中有多个过滤器供你根据不同的数据维度进行选择。

其实按住 **Cmd** 或 **Ctrl** 并单击鼠标可以同时选择多个过滤器。效果如下图所示：



清空缓存并执行硬性重新加载

如果你需要执行页面硬性重新加载，在DevTools开启的状态下，长按浏览器刷新按钮。接下来可以看到一个下拉菜单，在菜单里可以同时执行清空缓存并重新加载。



使用**Chrome**任务管理器

Chrome任务管理器允许你查看GPU，CPU，JavaScript内存使用，CSS和脚本缓存等信息。

执行下面的操作打开任务管理器：

1. 单击Chrome工具栏中的菜单按钮
2. 选择工具
3. 选择任务管理器
4. 在这里你可以通过在某列鼠标右键查看任何一个进程附加信息或者结束一个进程。