# 前言

- https://developers.google.com/web/tools/chrome-devtools/
- https://juejin.im/book/5c526902e51d4543805ef35e/section/5c5269026fb9a049f1549e4a

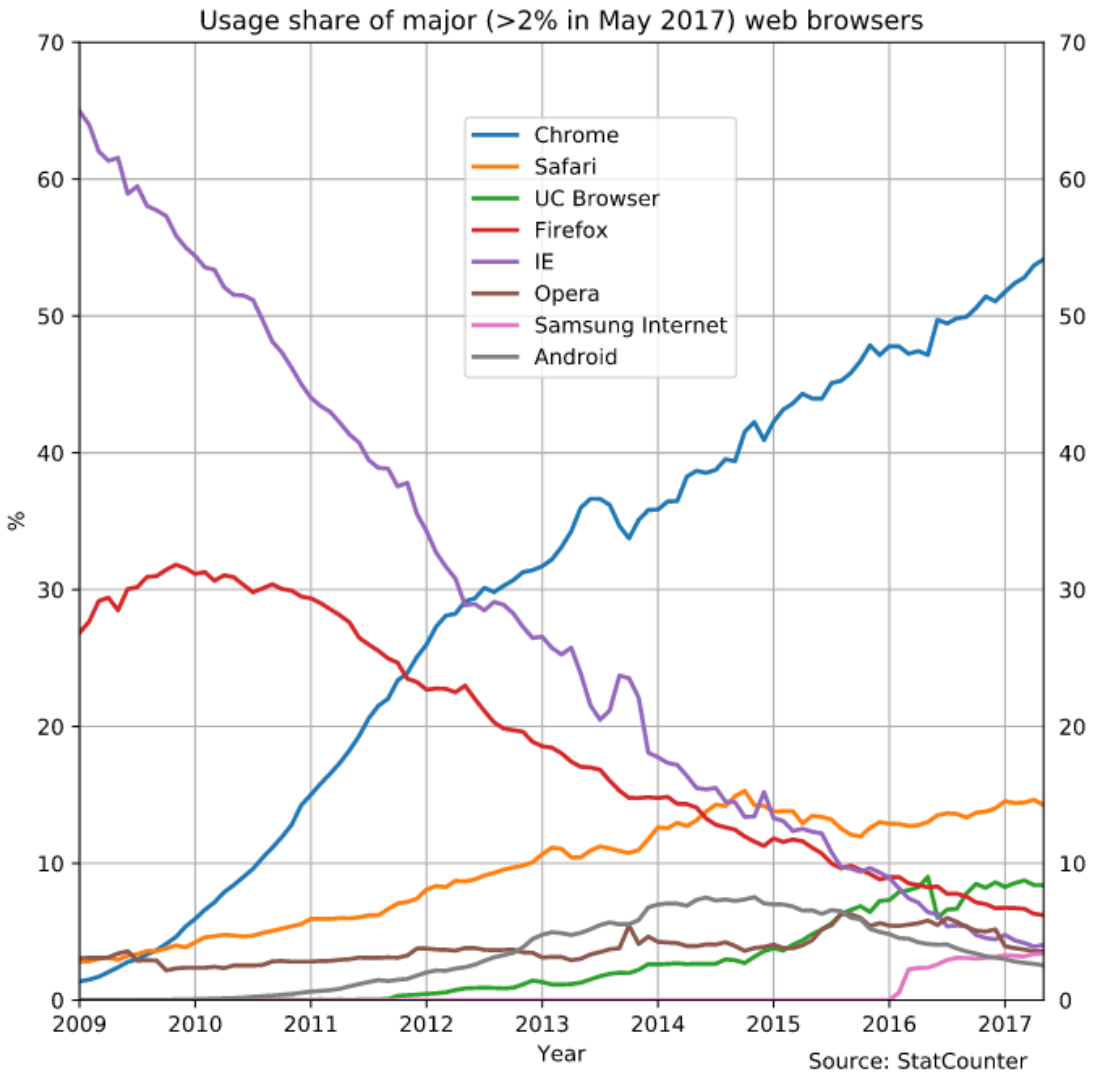
工欲善其事必先利其器。

有些时候一个技巧可以节省我们很多的时间，也会让调试的过程变得更加简单直接。本次分享从不同的情景来说明应该如何搭配使用 Chrome DevTools 中的小技巧，让大家熟练掌握 Chrome 调试技巧，直接提升工作效率。


# 关于 Chrome

谷歌浏览器（通常简称为 Chrome ）是由谷歌开发的网络浏览器。 它于 2008 年首次针对 Microsoft Windows 发布，后来移植到 Linux ， macOS ， iOS 和 Android 。 浏览器也是 Chrome OS 的主要组件，它可以作为 Web 应用的平台。Chrome-wikipedia

现如今，浏览器的市场已天下三分，Chorme，Safari 和 FireFox，从 2008 年 Chrome 横空出世以来，如今已经一家独大占据了半壁江山：




# 关于 Chrome DevTools

对于大部分人来说，Chrome 可能只是个浏览器，但是对于开发人员来说，它更是一个强大无比的工具，为了方便开发人员调试代码，主流的浏览器都内置了 DevTools， 所以无论你是前端还是后端，掌握 Chrome DevTools 的使用技巧意味着效率直接的提高。

## 打开方式

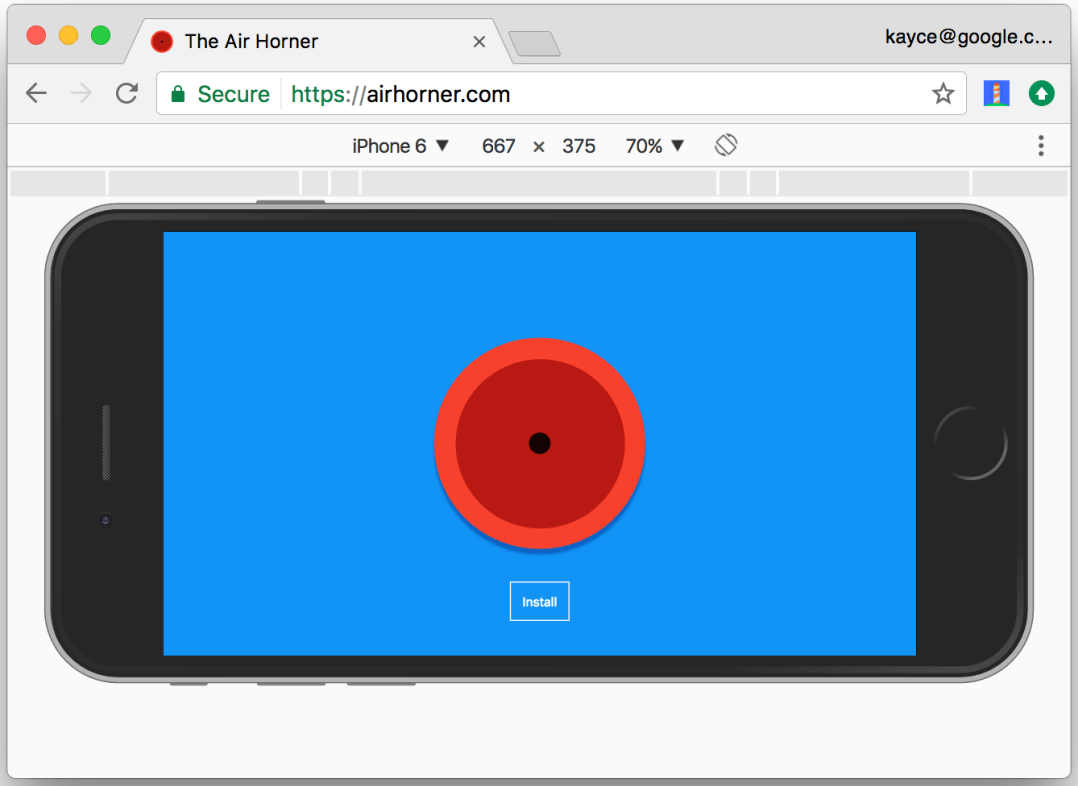
1. F12
2. Ctrl + Shift + I (Windows) 或 Cmd + Opt + I (Mac)
3. 更多工具 > 开发者工具
4. 在页面元素上右键点击，选择 “检查”

## 面板概览

Chrome DevTools 包含 1 个设备模式和 8 个常见的面板，我们还可以通过插件来增加一些面板（如：Vue-DevTools）

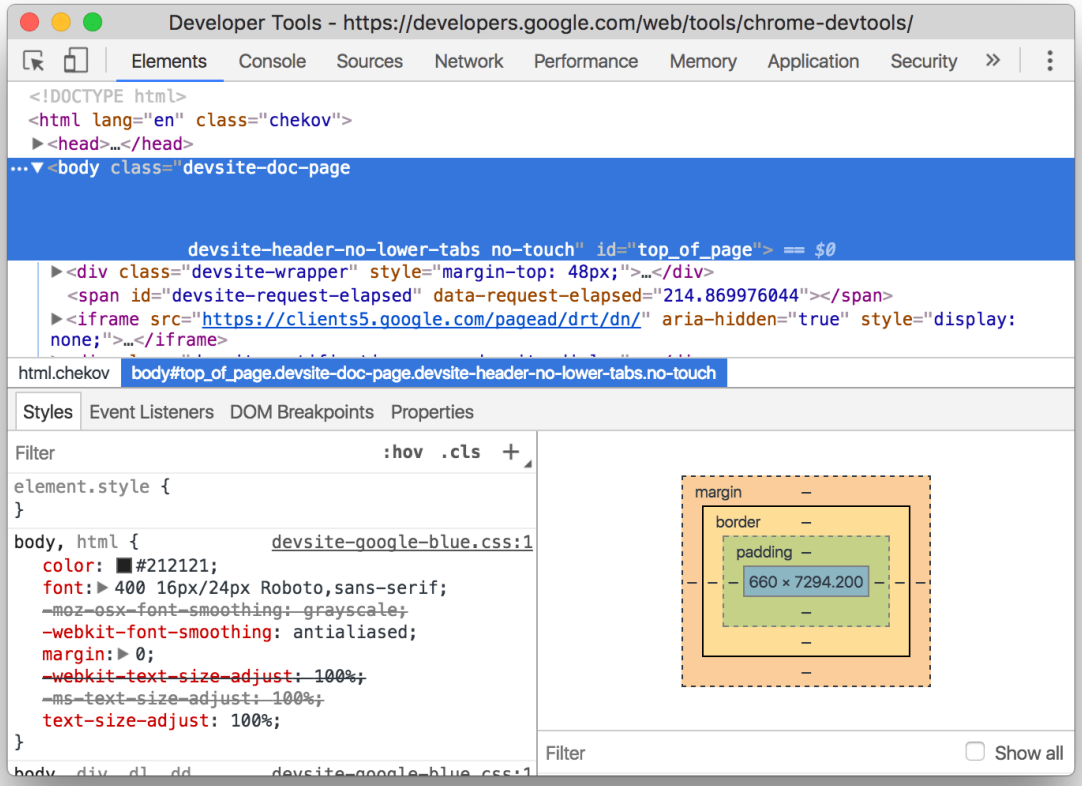
### 0. 设备模式 Device Mode

使用设备模式构建完全响应式，移动优先的网络体验。



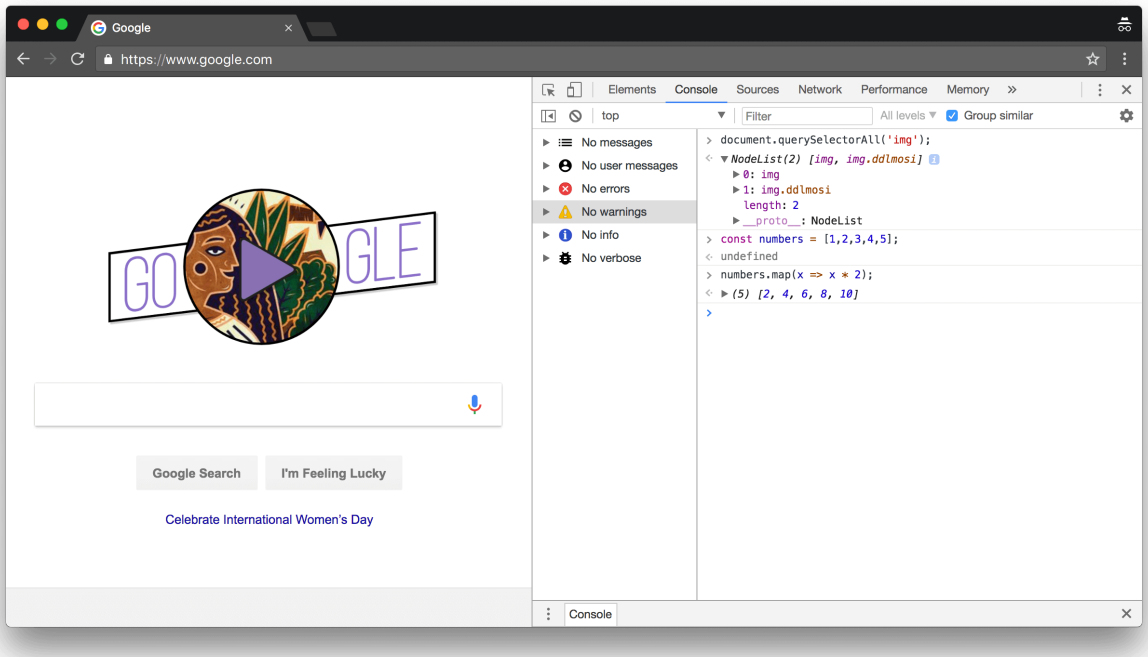

### 1. 元素面板 Elements

使用元素面板可以自由的操作 DOM 和 CSS 来迭代布局和设计页面。



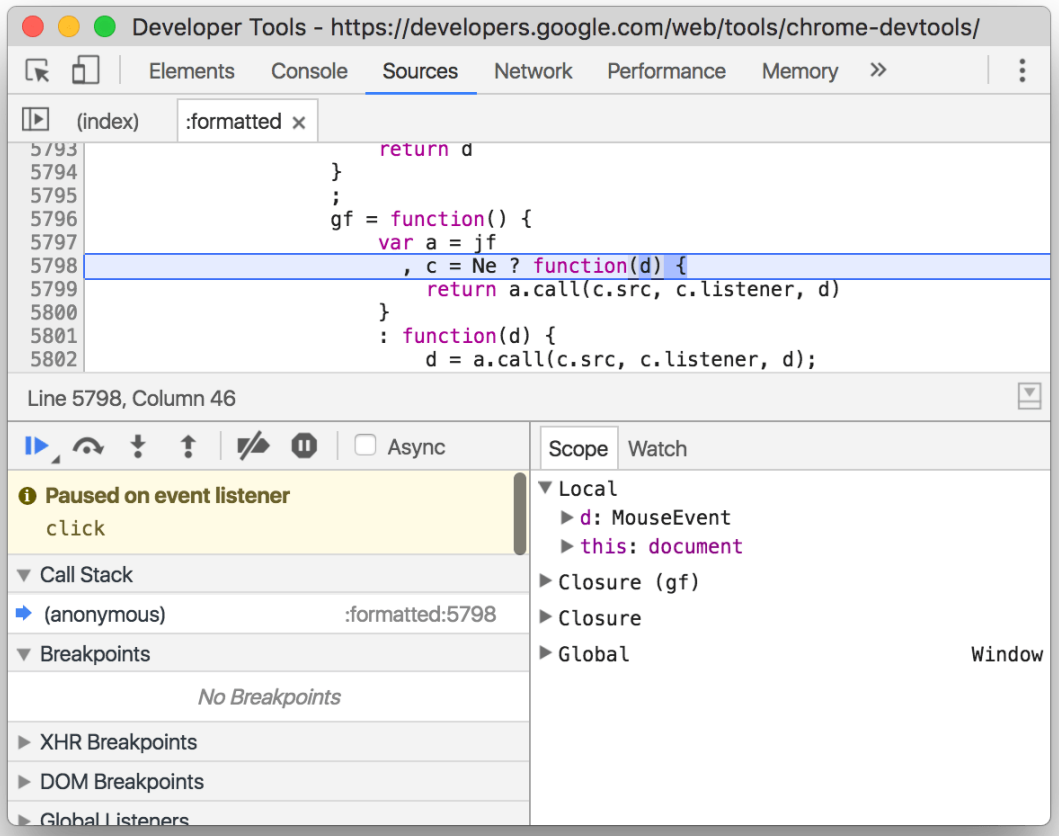

## 2. 控制台面板 Console

在开发期间，可以使用控制台面板记录诊断信息，或者使用它作为 shell 在页面上与 JavaScript 交互。



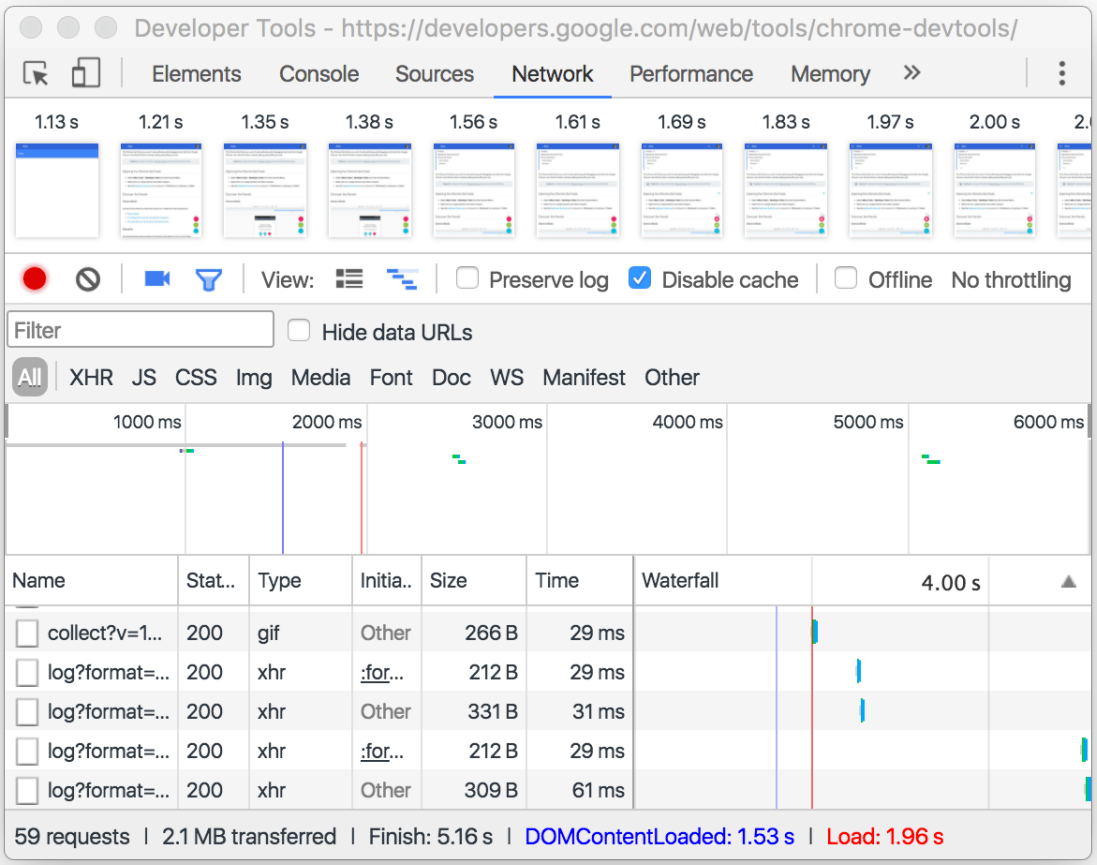

## 3. 源代码面板 Sources

在源代码面板中设置断点来调试 JavaScript ， 或者通过Workspaces（工作区）连接本地文件来使用开发者工具的实时编辑器。



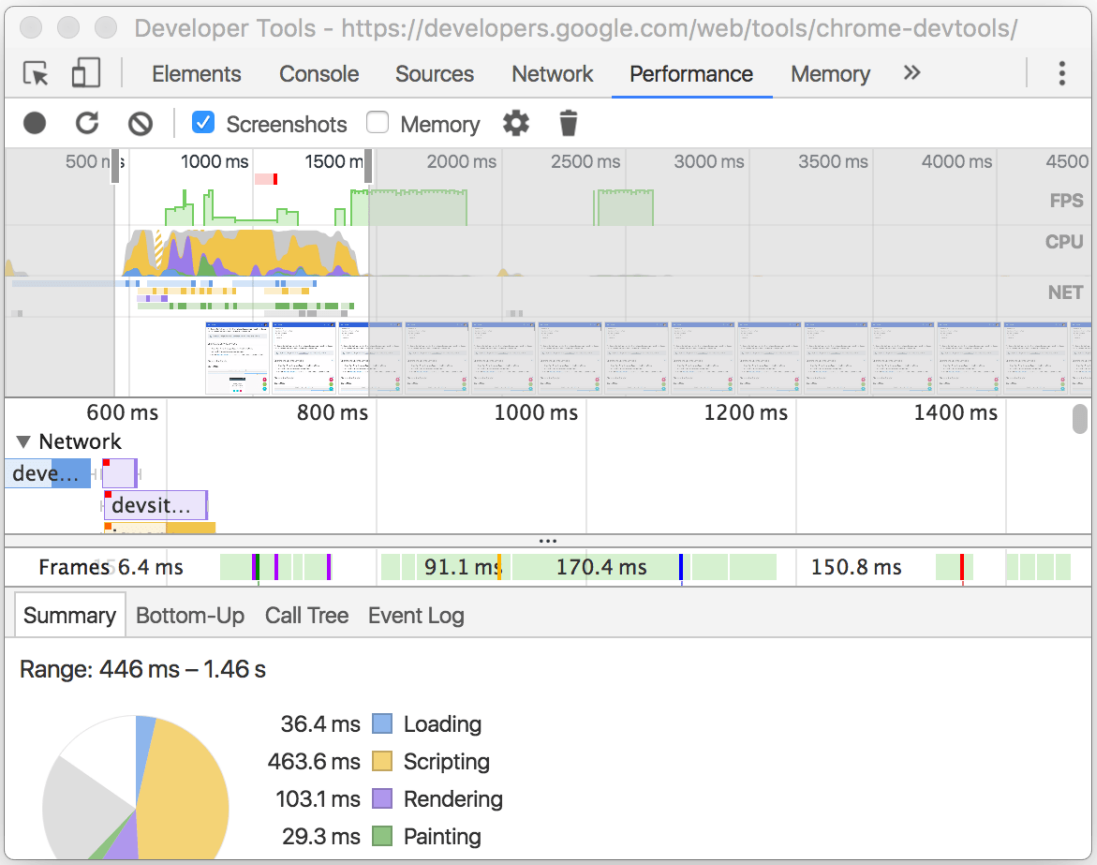

### 4. 网络面板 Network

使用网络面板了解请求和下载的资源文件并优化网页加载性能。



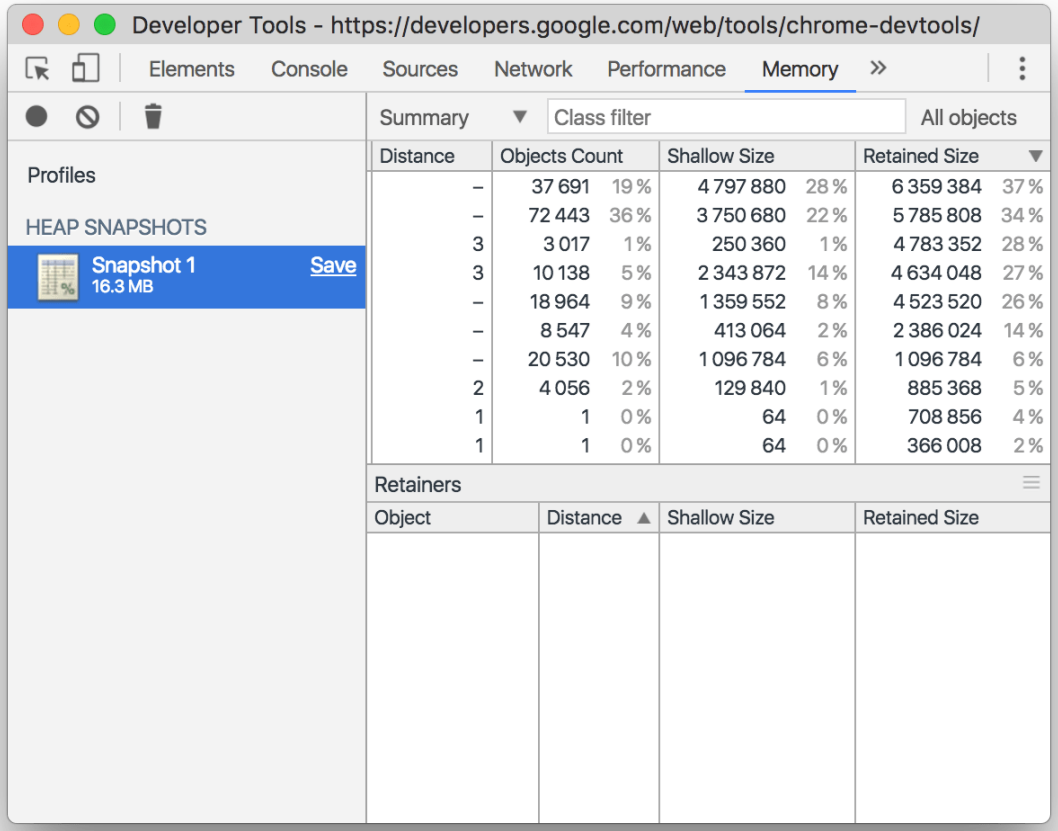

### 5. 性能面板 Performance

使用性能面板可以通过记录和查看网站生命周期内发生的各种事件来提高页面的运行时性能。



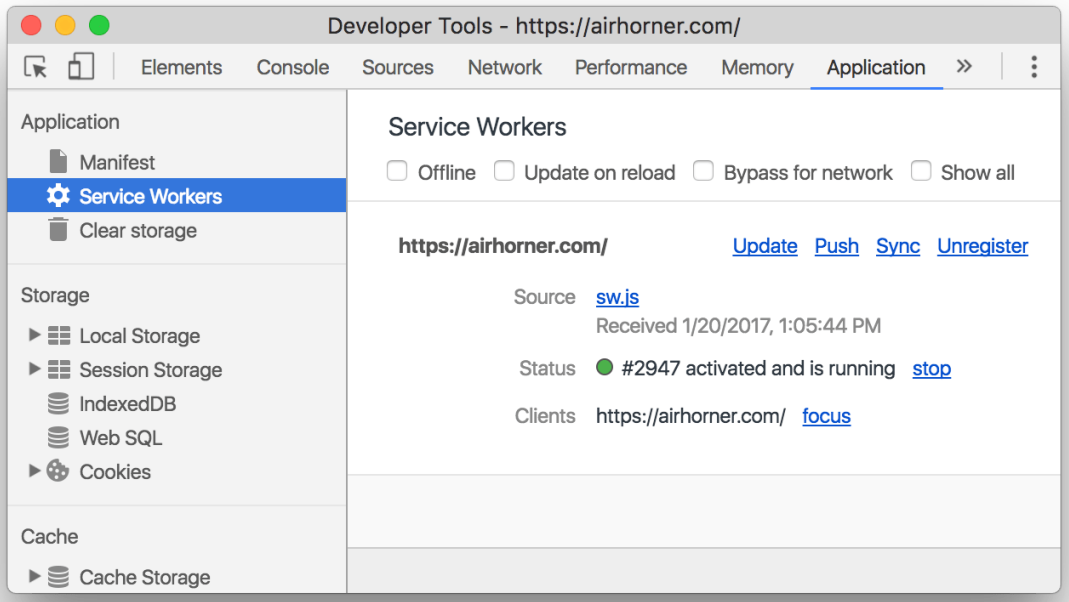

### 6. 内存面板 Memory

如果需要比内存面板提供的更多信息，可以使用内存面板，例如跟踪内存泄漏。



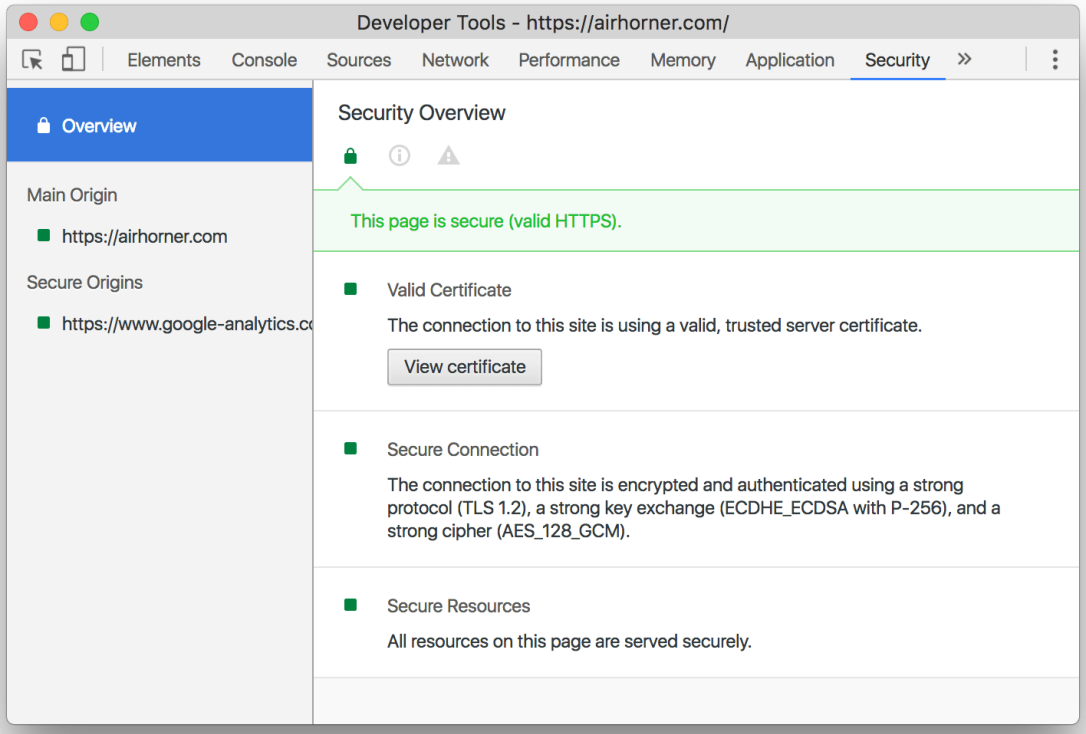

### 7. 应用面板 Application

使用应用面板可以检查加载的所有资源，包括 IndexedDB 与 Web SQL 数据库，本地和会话存储，cookie，应用程序缓存，图像，字体和样式表。




### 8. 安全面板 Security

使用安全面板调试混合内容问题，证书问题等等。




# 实用快捷键

`Ctrl + shift + D`：把 DevTools 窗口切换到右边、下边

`Ctrl + shift + M`：切换设备模式

`Ctrl + [` 和 `Ctrl + ]`：切换面板

上下箭头：递增/递减，可配合 Alt、Ctrl、Shift 实现不同的步长（对调整样式特别有用）

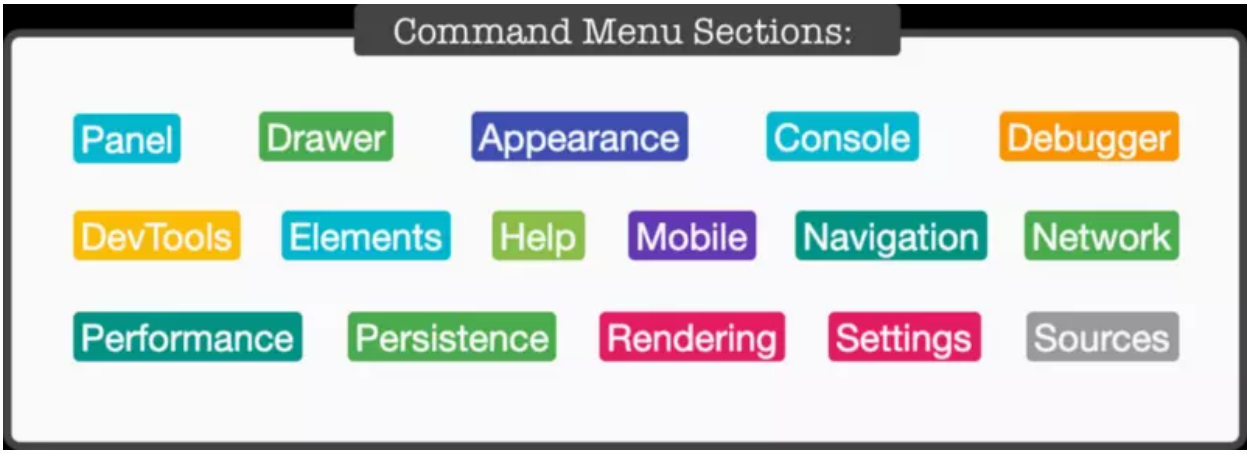
`Ctrl + F`：在 elements， logs， sources 和 network 中查找

# 使用 Command

我们直接可以直接看到的 DevTools 的功能，其实只是有限的一部分，怎么去探索更多的功能呢?

Command 菜单可以帮助我们快速找到那些被隐藏起来的功能，你可以使用 `Ctrl + Shift + P` 或 DevTools 的 `dropdown` 按钮打开 Command。

Command 包含下列几类命令：



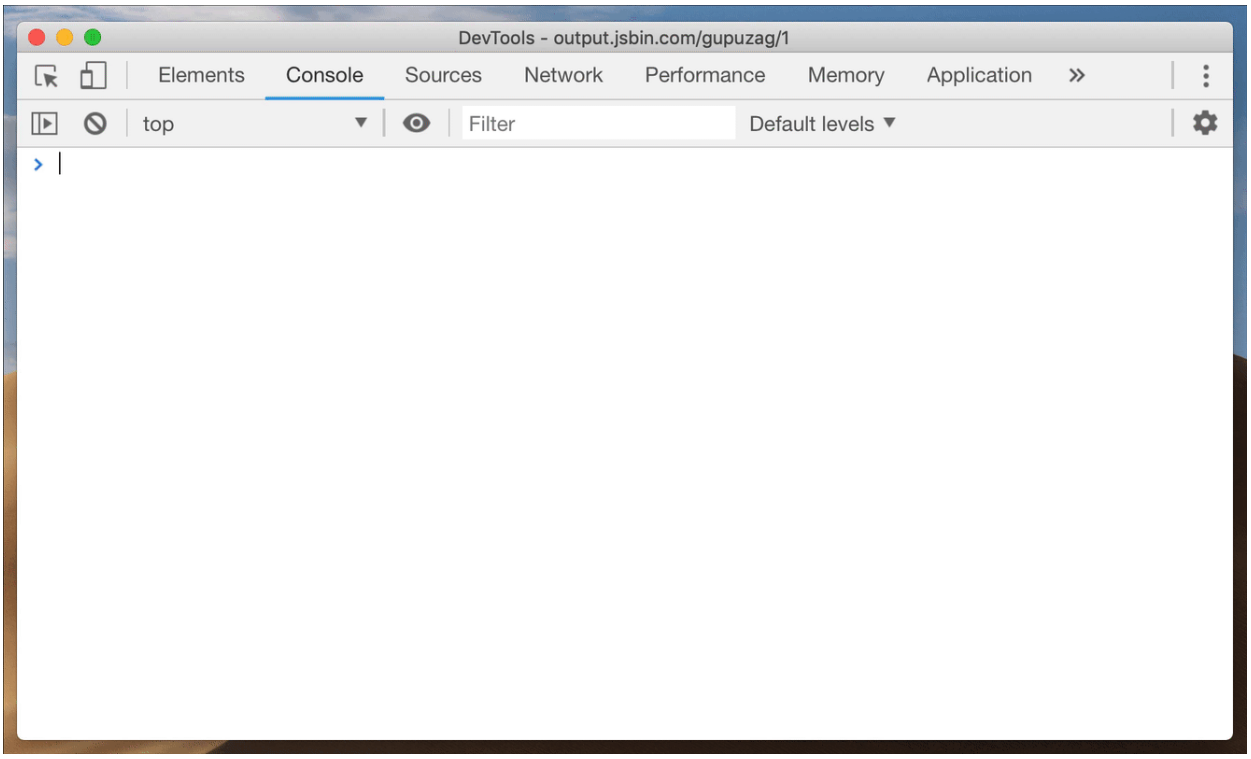

常用的 Command：

- 截屏：`capture`
- 切换主题：`theme`

# copying

你可以通过全局的方法 `copy()` 在 console 里复制任何你能拿到的资源

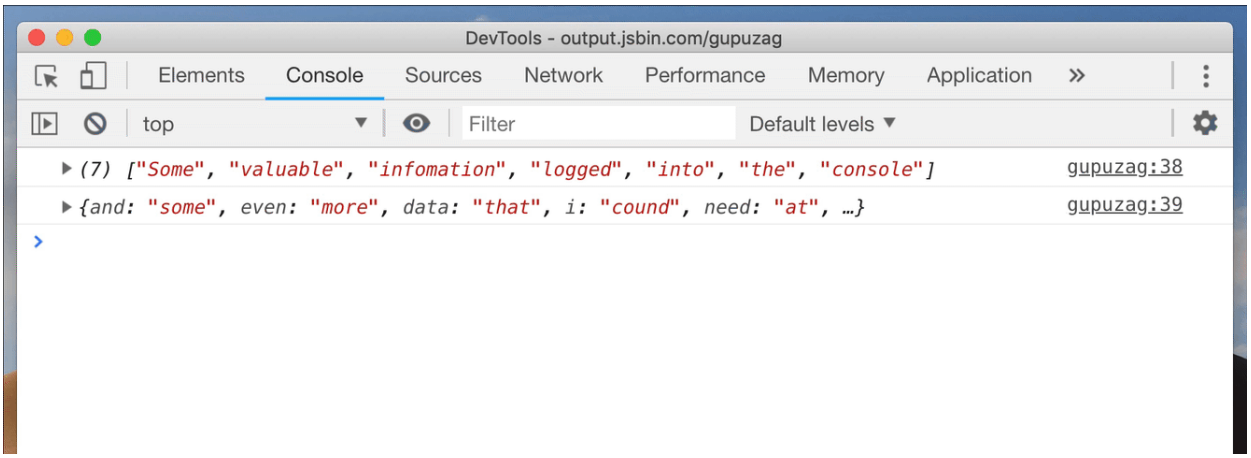
例如：`copy(location)`




# saving

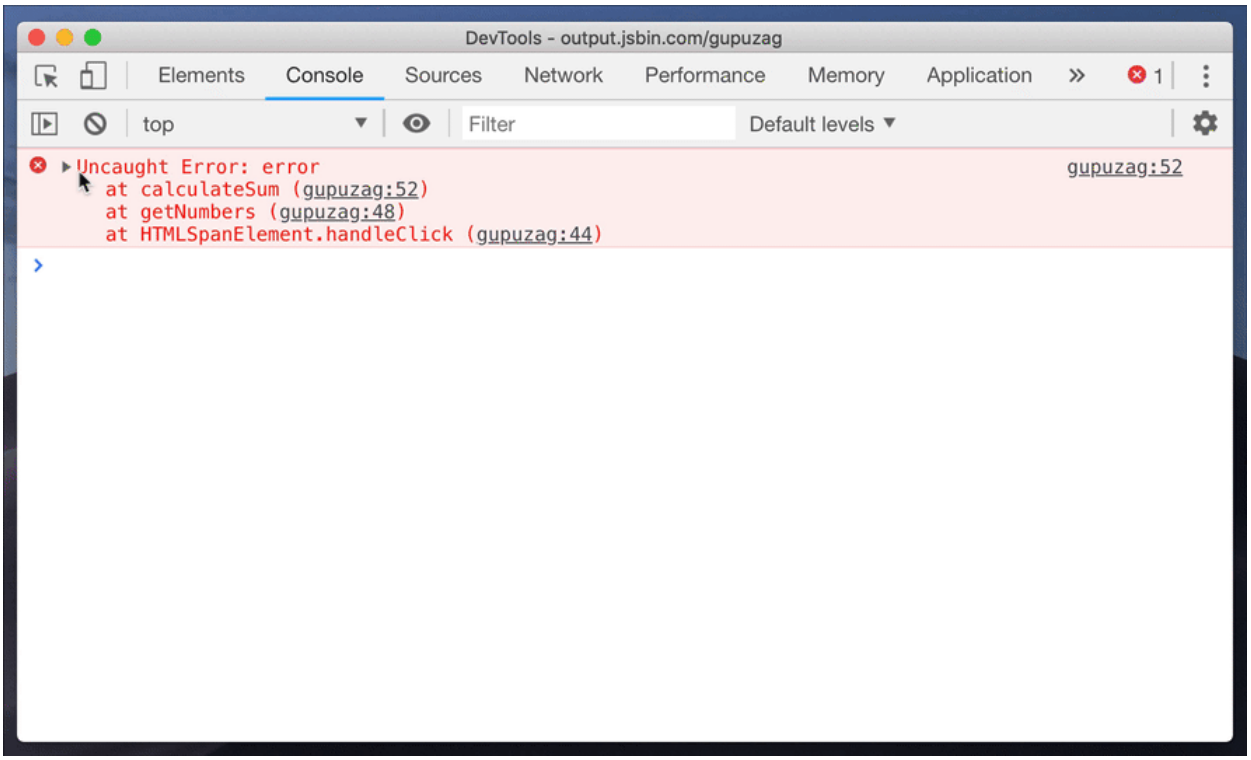
如果你在 console 中打印了一堆数据（例如一个大型数组），然后你想对这些数据做一些额外的操作，那就可以将它转换成一个全局变量，只需要右击它，并选择 “**Store as global variable**” (保存为全局变量) 选项。第一次使用的话，它会创建一个名为 temp1 的变量，第二次创建 temp2，以此类推。

> 注意：得到的 temp$ 的是原数据的引用，操作会影响原数据。




# 保存堆栈信息( Stack trace )

大多数情况下都不是一个人开发一个项目，而是一个团队协作，那么 如何准确的描述问题，就成为了沟通的关键 ， 这时候 console 打印出来的堆栈跟踪的信息对你和同事来说就起大作用了，可以省去很多沟通成本，所以你可以直接把堆栈跟踪的信息保存为一个文件，而不只是截图发给对方：




# Snippets 代码段

> 如果你需要频繁地在 console 中输入某个代码段，Snippets 功能可能会帮助你。

比如，我想统计网页标签使用数量，评价网页是否语义化，我会把这段脚本赋值 Console 面板：

```
/**
 * 统计网页标签使用数量，评价网页是否语义化的重要指标
 */
function getTagsMap() {
  return [...document.querySelectorAll('*')].reduce((a, c) => {
    let tagName = c.tagName.toLowerCase()
    a[tagName] = a[tagName] ? a[tagName] + 1 : 1
    return a
  }, {})
}
getTagsMap()
```

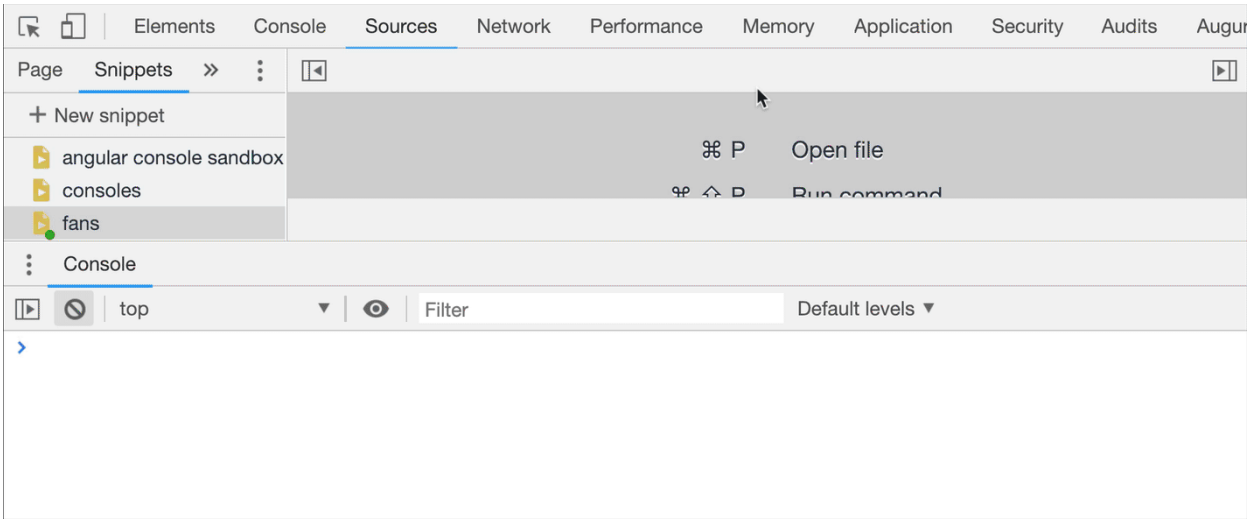
现在看来，即使这个脚本并没有花费我太多的精力来编写，但也只是偶尔运行一下，所以对于我来说，记住一段这样的脚本会很麻烦。

那怎么解决这个问题呢?

这就是 Snippets 的用武之地：它允许你存放 JavaScript 代码到 DevTools 中，方便你复用这些 JavaScript 代码段：

进入到 Sources 面板，在导航栏里选中 `Snippets` 这栏，点击 `New snippet` 新建一个代码段，然后输入你的代码之后保存就可以 `Run` 了!

> 当我在 DevTools 中预设了一组很棒的代段块以后，甚至都不必再通过 Sources 来运行它们。使用 Command Menu 才是最快的方式。只需在它的输入框中输入 `!`，就可以根据名字来筛选预设代码段。




# Elements 篇

H 隐藏元素

Delete 删除元素

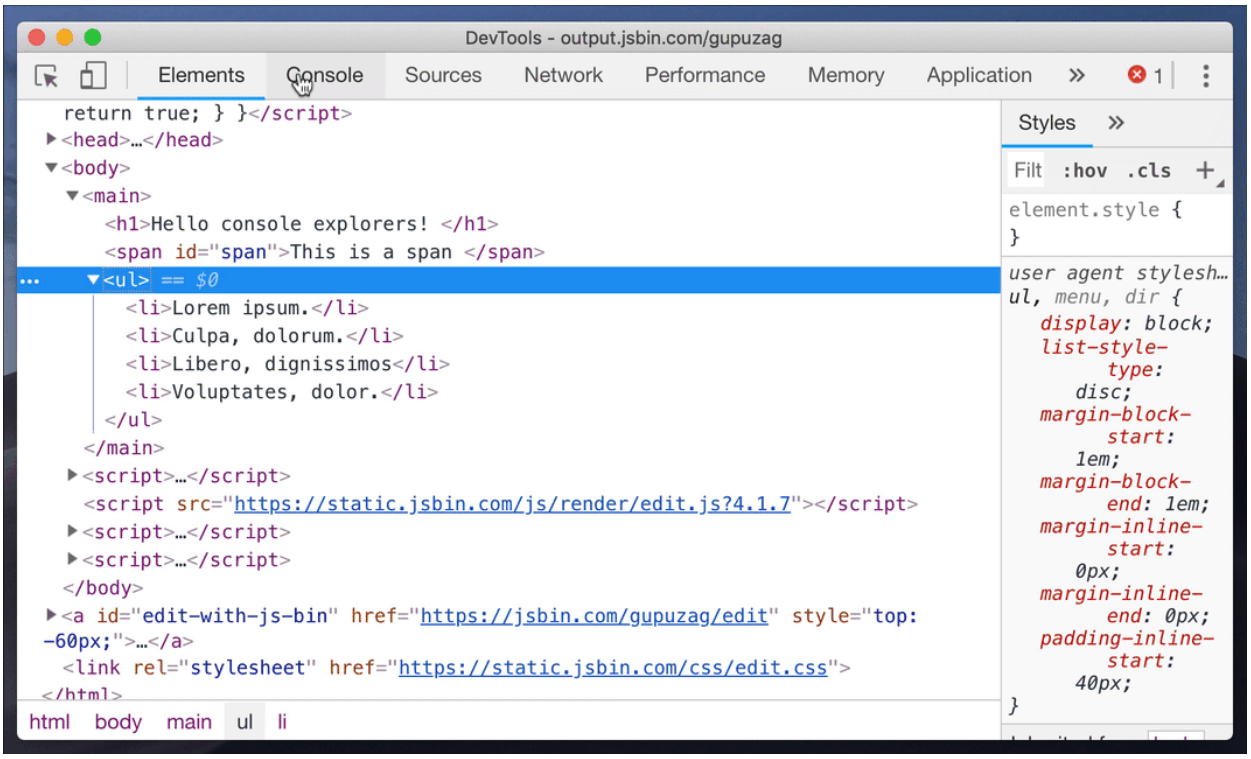
Ctrl + C 复制元素

Ctrl + Z 撤销改动

拖动 & 放置 元素

使用 Ctrl 移动元素

更多操作请在元素上点击右键




# console 篇

## 多种多样的 console

console.log 普通打印

console.table 表格打印（适用于数组、类数组、对象）

console.dir 查看节点属性

console.assert 断言

console.time / console.timeEnd() 计时器

## console 中的 '$'

**1. $0**

在 Chrome 的 Elements 面板中， `$0` 是对我们当前选中的 html 节点的引用。

同理， `$1` 是对上一次我们选择的节点的引用，一直到 `$4` 。

你可以尝试一些相关操作，例如: `$1.appendChild($0)`

**2. $ 和 $$**

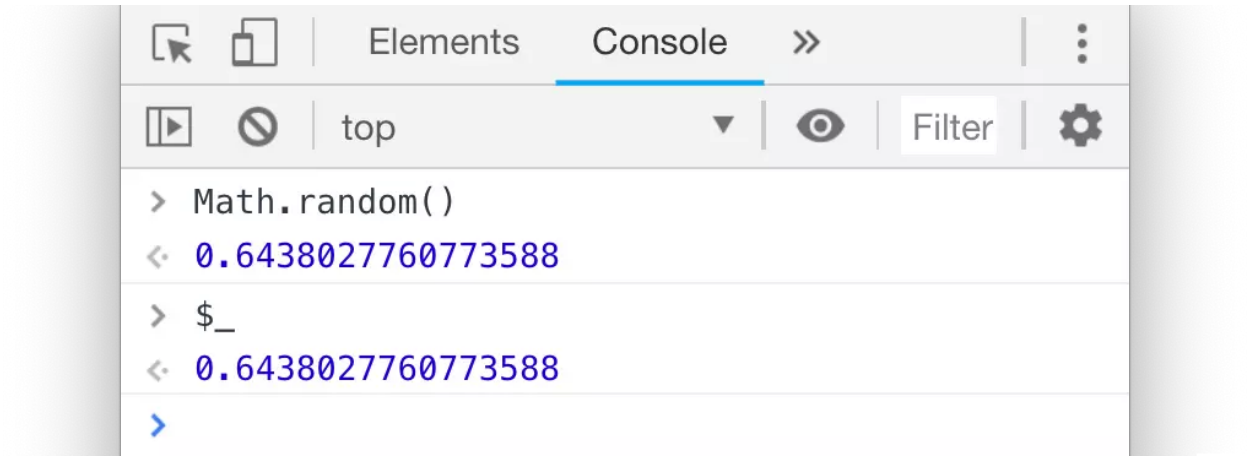
console 面板中 $ 变量如果未曾被定义过就是 `document.querySelector` 的别名

$$ 相当于执行 `document.QuerySelectorAll` 并且它返回的是一个节点的数组。

```
$$('div') === Array.from(document.querySelectorAll('div'))
```

**3. $_**

`$_` 是对上次执行的结果的引用

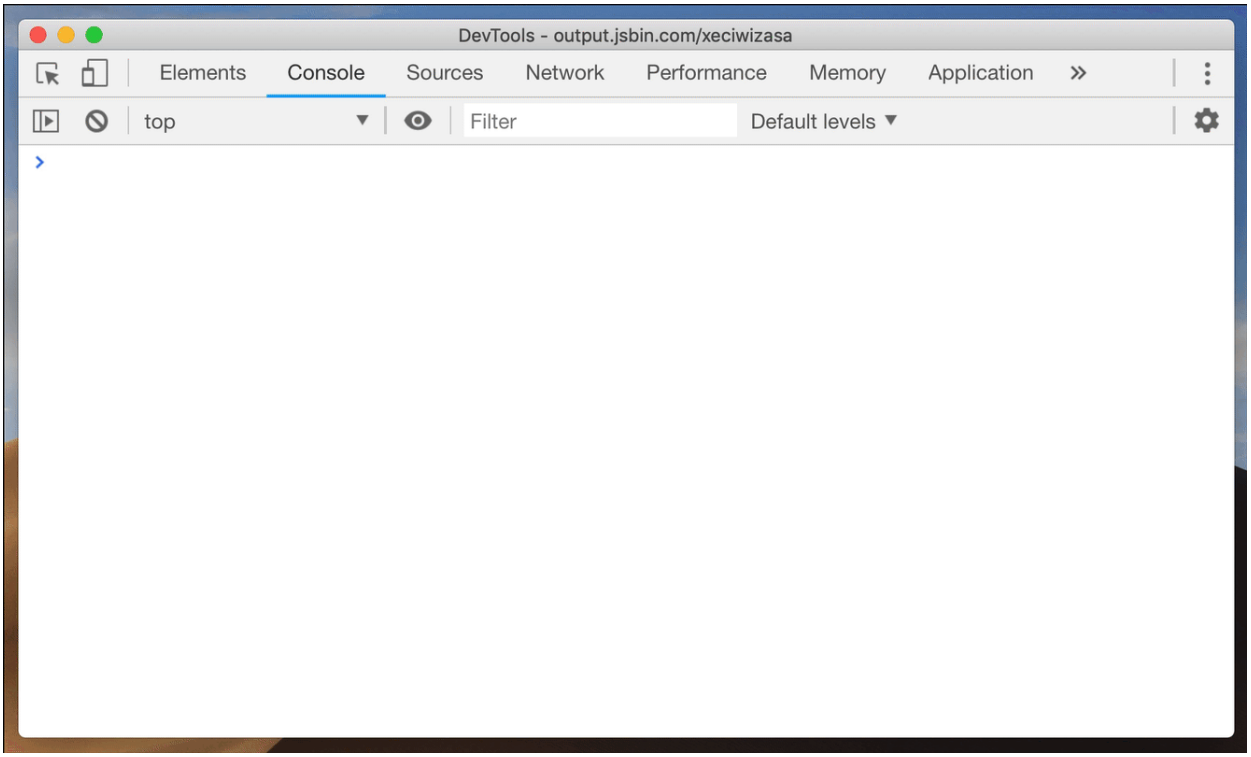



**4. $i**

在 DevTools 中使用 npm 插件!

有时你只是想尝试下新出的 npm 包，现在不用再大费周章去建一个项目测试了，只需要在 Chrome 扩展插件: Console Importer 的帮助之下，快速的在 console 中引入和测试一些 npm 库。

运行 `$i('lodash')` 或者 `$i('moment')` 后，你就可以获取到 `lodash` / `momentjs` 了:



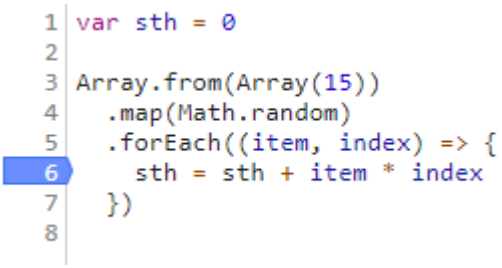

# 断点调试 breakpoints

# 基本用法

你还在用 `console.log` 和 `alert` 吗？使用断点（breakpoint）可以非常方便地调试 JavaScript 代码。

在行号上单击就设置了一个断点，然后你就可以调试了。当然这也是最最基本的打断点的方式了，当然了，相较于调试全靠 `console.log` 和 `alert` 已经高端很多了。

同时也可以通过在行号上右键点击 `Add breakpoint` 来设置断点。设置断点的行号上会显示一个蓝色的矩形来告诉你这里有一个断点。




当断点触发时，整个页面会处于暂停状态，并会切换到 Source 页断点处方便调试，直到终止该断点调试后页面才会继续运行。

快捷键：

- F8
- F9
- F10
- F11

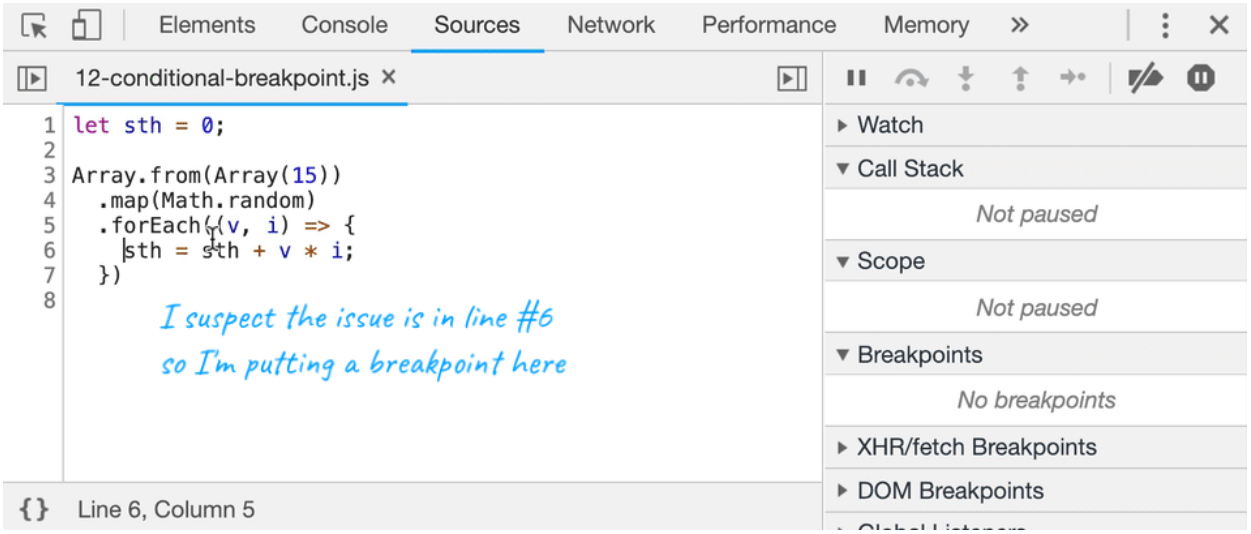
# 条件断点

比如说你写了一个循环，该循环会执行 100 次，但是你只对第 50 次循环的结果感兴趣，又或者你只对一些满足某些条件的结果感兴趣。于是，你一手托腮，另一只手放在 F8 键上，狂按数十下下后正襟危坐，开始调试。

显然，Chrome 已经帮我们想到了这种场景，我们可以通过添加一些条件断点来避免一些无意义的断点：

1. 右击行号，选择 `Add conditional breakpoint...` (添加条件断点)
2. 或者右击一个已经设置的断点并且选择 `Edit breakpoint` (编辑断点)

然后输入一个执行结果为 `true` 或者 `false` 的表达式（它的值其实不需要完全为 `true` 或者 `false` 尽管那个弹出框的描述是这样说的）。在这个表达式中你可以使用任何这段代码可以获取到的值（当前行的作用域）。如果条件成立，这个断点就会生效。



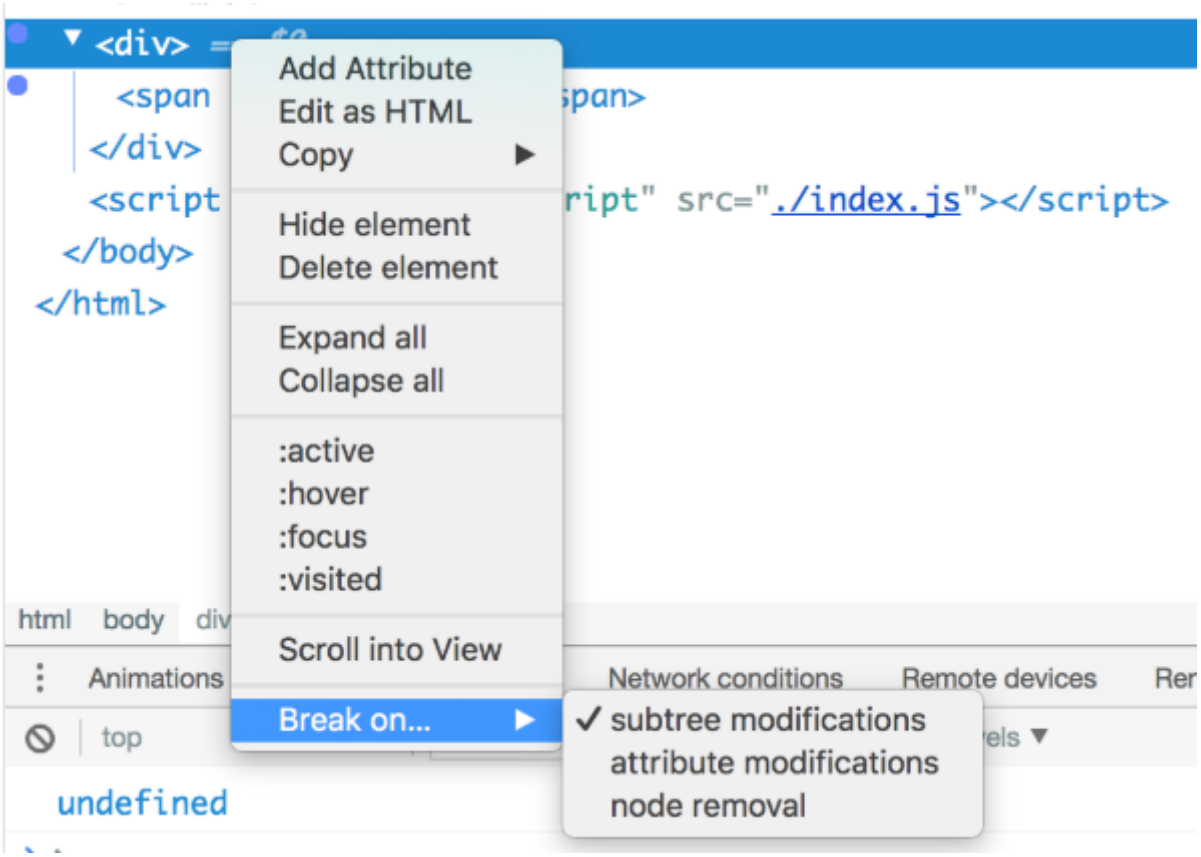

## DOM 元素断点

断点不仅仅可以设置在 JS 代码上，还可以在 DOM 元素上设置断点。

我们可以设置三种 DOM 元素断点

1. `subtree modifications` 子节点内容的的改变（子节点的属性修改不会触发，当前节点的修改不会触发）
2. `attribute modifications` 当前节点的属性改变
3. `node removal` 当前节点被移除

当我们的脚本触发了 DOM 的修改时，devtools 会直接跳转到 Source 页并定位到修改 DOM 的那行代码上。

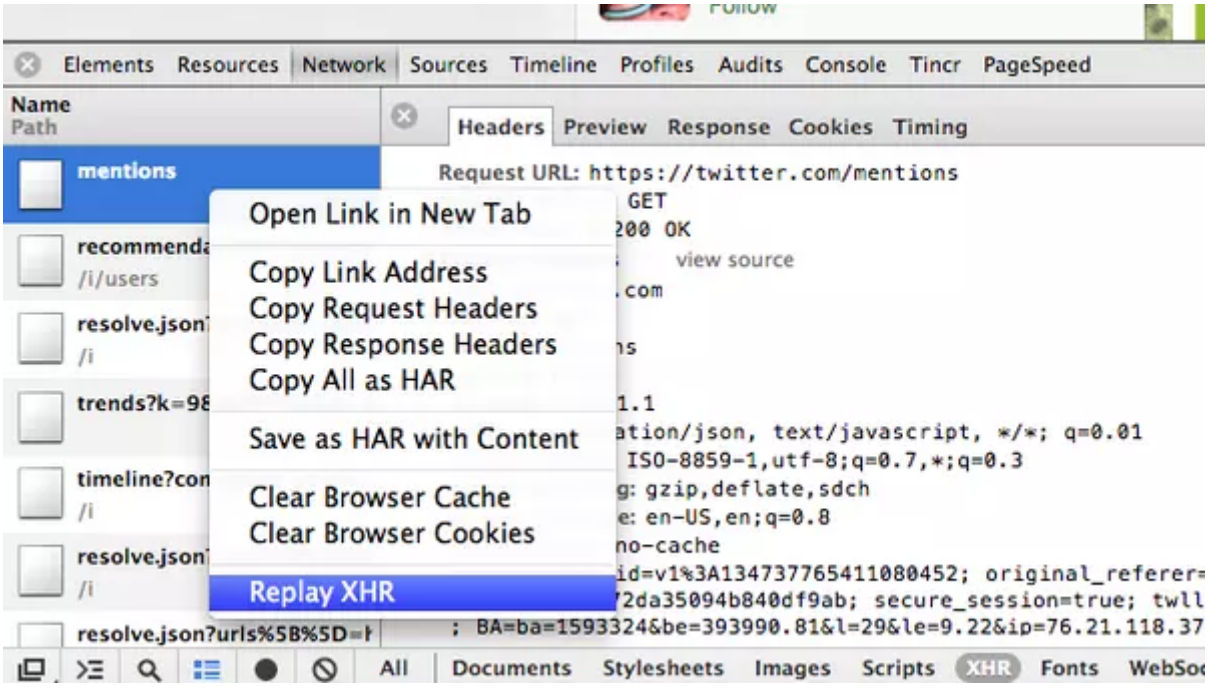



# 你还一层层展开 DOM?

Alt + Click，快速展开所有 DOM

## Network - 重新发送 XHR 的请求

如何重新发送 XHR 的请求？刷新页面？太老套了，试试这么做：




## 恐龙小游戏(Chrome Dino)

相信在座的各位对 Google Chrome 中的一个复活节彩蛋 —— 恐龙小游戏(Chrome Dino)不会感到陌生。在与互联网断开连接时，当用户尝试访问网站，并在按下空格键后即可开始这个游戏。

Chrome 中的恐龙小游戏是一个简单的无限跑步游戏，它会让用户跳过仙人掌，并闪避障碍物，游戏为用户提供基本控制，按空格键跳跃（并开始游戏），向下箭头(↓)俯身奔跑以躲避障碍物。目标是在互联网重新开始工作之前让用户打发时间。

其实用户也可以在不打开飞行模式的情况下玩 Chrome dino 游戏。只需在 Chrome 浏览器的地址栏中输入 `chrome://dino`，用户就可以进入“街机模式”，在那里用户可以在全窗口环境中畅玩这款游戏。


# 推荐一些好用的 Chrome 扩展

启动页：掘金、Infinity新标签页、Momentum

翻译：Google 翻译、有道云翻译、百度翻译、有道词典Chrome划词插件

广告屏蔽：Adblock Plus

OneTab

Clipboard History 2

WEB前端助手(FeHelper)

JSONView

Stylish-为任意网站自定义主题

tampermonkey 油猴