

在 Chrome DevTools 中调试 JavaScript 入门



By Kayce Basques

(<https://developers.google.com/web/resources/contributors/kaycebasques?hl=zh-cn>)

Technical Writer for Chrome DevTools

本交互式教程循序渐进地教您在 Chrome DevTools 中调试 JavaScript 的基本工作流程。虽然教程介绍的是如何调试一种具体问题，但您学到的一般工作流程对调试各种类型的 JavaScript 错误均有帮助。

如果您使用 `console.log()` 来查找和修正代码中的错误，可以考虑改用本教程介绍的工作流程。其速度快得多，也更有效。

第 1 步：重现错误

重现错误始终是调试的第一步。“重现错误”是指找到一系列总是能导致错误出现的操作。

您可能需要多次重现错误，因此要尽量避免任何多余的步骤。

请按照以下说明重现您要在本教程中修正的错误。

1. 点击 **Open Demo**。演示页面在新标签中打开。

OPEN DEMO

(<https://googlechrome.github.io/devtools-samples/debug-js/get-started>)

2. 在演示页面上，输入 5 作为 **Number 1**。
3. 输入 1 作为 **Number 2**。
4. 点击 **Add Number 1 and Number 2**。
5. 查看输入和按钮下方的标签。上面显示的是 `5 + 1 = 51`。

啊呜。这个结果是错误的。正确结果应为 6。这就是您要修正的错误。

第 2 步：使用断点暂停代码

DevTools 让您可以暂停执行中的代码，并对暂停时刻的*所有*变量值进行检查。用于暂停代码的工具称为**断点**。立即试一试：

1. 按 **Command+Option+I** (Mac) 或 **Ctrl+Shift+I** (Windows、Linux) 在演示页面上打开 DevTools。
2. 点击 **Sources** 标签。
 1. 点击 **Event Listener Breakpoints** 将该部分展开。DevTools 显示一个包含 **Animation** 和 **Clipboard** 等可展开事件类别的列表。
 1. 在 **Mouse** 事件类别旁，点击 **Expand** ►。DevTools 显示一个包含 **click** 等 Mouse 事件的列表，事件旁有相应的复选框。
 2. 选中 **click** 复选框。

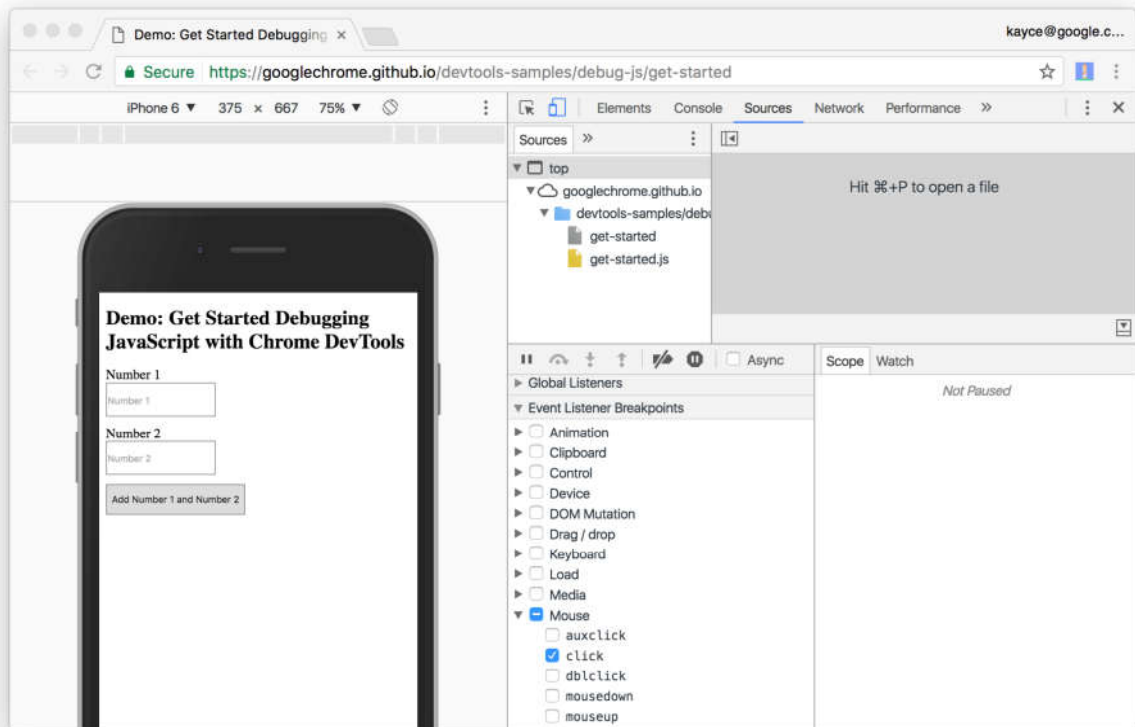


图 1： DevTools 在演示页面上打开，Sources 面板获得焦点，click 事件侦听器断点处于启用状态。如果 DevTools 窗口较大，则 **Event Listener Breakpoints** 窗格位于右侧，而不是像屏幕截图中那样位于左下方。

3. 返回至演示页面，再次点击 **Add Number 1 and Number 2**。DevTools 暂停演示并在 **Sources** 面板中突出显示一行代码。DevTools 突出显示的是下面这行代码：

```
function onClick() {
```

当您选中 **click** 复选框时，就是在所有 click 事件上设置了一个基于事件的断点。点击了*任何*节点，并且该节点具有 click 处理程序时，DevTools 会自动暂停在该节点 click 处理程序的第一行。


注：这不过是 DevTools 提供的众多断点类型中的一种。应使用的断点类型取决于您要调试的问题类型。

第 3 步：单步调试代码

一个常见的错误原因是脚本执行顺序有误。可以通过单步调试代码一次一行地检查代码执行情况，准确找到执行顺序异常之处。立即试一试：

1. 在 DevTools 的 **Sources** 面板上，点击 **Step into next function call** ，一次一行地单步调试 `onClick()` 函数的执行。DevTools 突出显示下面这行代码：

```
if (inputsAreEmpty()) {
```

2. 点击 **Step over next function call** 。DevTools 执行 `inputsAreEmpty()` 但不进入它。请注意 DevTools 是如何跳过几行代码的。这是因为 `inputsAreEmpty()` 求值结果为 `false`，所以 `if` 语句的代码块未执行。


这就是单步调试代码的基本思路。如果您看一下 `get-started.js` 中的代码，就能发现错误多半出在 `updateLabel()` 函数的某处。您可以不必单步调试每一行代码，而是使用另一种断点在靠近错误位置的地方暂停代码。

第 4 步：设置另一个断点

代码行断点是最常见的断点类型。如果您想在执行到某一行代码时暂停，请使用代码行断点。立即试一试：

1. 看一下 `updateLabel()` 中的最后一行代码，其内容类似于：

```
label.textContent = addend1 + ' + ' + addend2 + ' = ' + sum;
```

2. 在这行代码的左侧，可以看到这行代码的行号：**32**。点击 **32**。DevTools 会在 **32** 上放置一个蓝色图标。这意味着这行代码上有一个代码行断点。DevTools 现在总是会在执行这行代码之前暂停。
3. 点击 **Resume script execution** 。脚本继续执行，直至到达您设置了断点的代码行。
4. 看一下 `updateLabel()` 中已执行的代码行。


DevTools 打印输出 `addend1`、`addend2` 和 `sum` 的值。

`sum` 的值疑似有问题。其求值结果本应是数字，而实际结果却是字符串。这可能就是造成错误的原因。

第 5 步：检查变量值

另一种常见的错误原因是，变量或函数产生的值异常。许多开发者都利用 `console.log()` 来了解值随时间变化的情况，但 `console.log()` 可能单调乏味而又效率低下，原因有两个。其一，您可能需要手动编辑大量调用 `console.log()` 的代码。其二，由于您不一定知晓究竟哪一个变量与错误有关，因此可能需要对许多变量进行记录。

DevTools 为 `console.log()` 提供的其中一个替代工具是监视表达式。可以使用监视表达式来监视变量值随时间变化的情况。顾名思义，监视表达式的监视对象不仅限于变量。您可以将任何有效的 JavaScript 表达式存储在监视表达式中。立即试一试：

1. 在 DevTools 的 **Sources** 面板上，点击 **Watch**。该部分随即展开。
2. 点击 **Add Expression** .
3. 键入 `typeof sum`。
4. 按 **Enter**。DevTools 显示 `typeof sum: "string"`。冒号右侧的值就是监视表达式的结果。

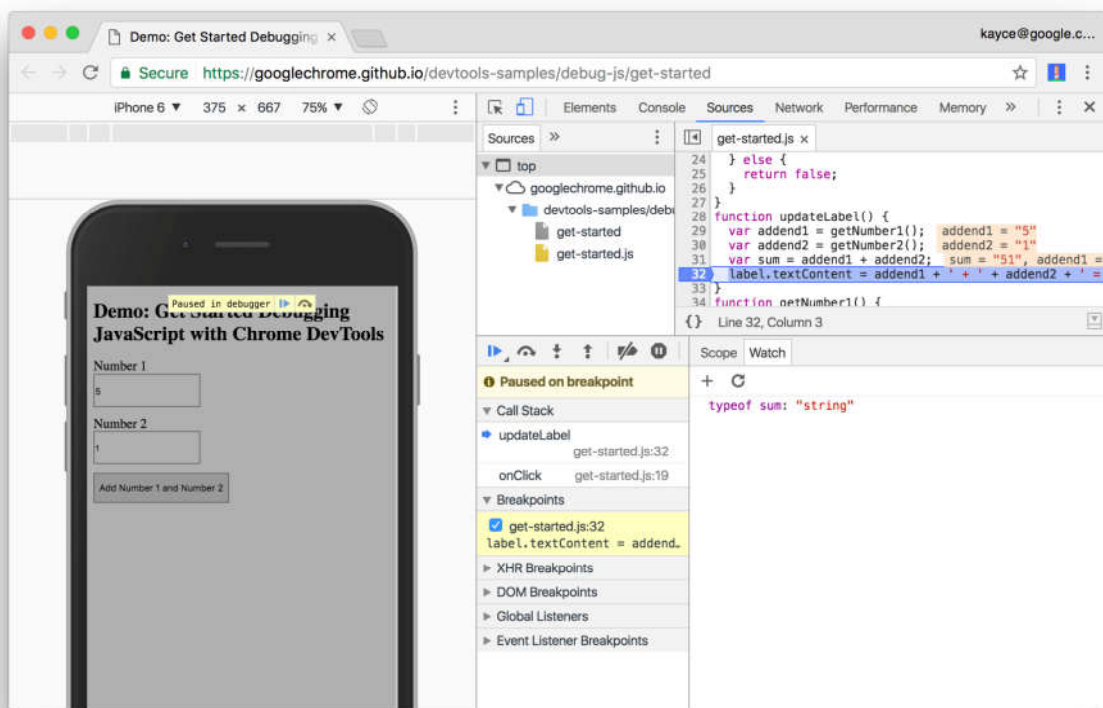


图 1：创建 `typeof sum` 监视表达式后的“监视表达式”窗格（右下方）。如果 DevTools 窗口较大，则“监视表达式”窗格位于右侧，**Event Listener Breakpoints** 窗格的上方。

正如猜想的那样，`sum` 的求值结果本应是数字，而实际结果却是字符串。这就是演示页面错误的原因。

DevTools 为 `console.log()` 提供的另一个替代工具是 Console。可以使用 Console 对任意 JavaScript 语句求值。开发者通常利用 Console 在调试时覆盖变量值。在您所处的情况下，Console 可帮助您测试刚发现的错误的潜在解决方法。立即试一试：

1. 如果您尚未打开 Console 抽屉，请按 `Escape` 将其打开。它会在 DevTools 窗口底部打开。
2. 在 Console 中，键入 `parseInt(addend1) + parseInt(addend2)`。
3. 按 `Enter`。DevTools 对语句求值并打印输出 6，即您预料演示页面会产生的结果。

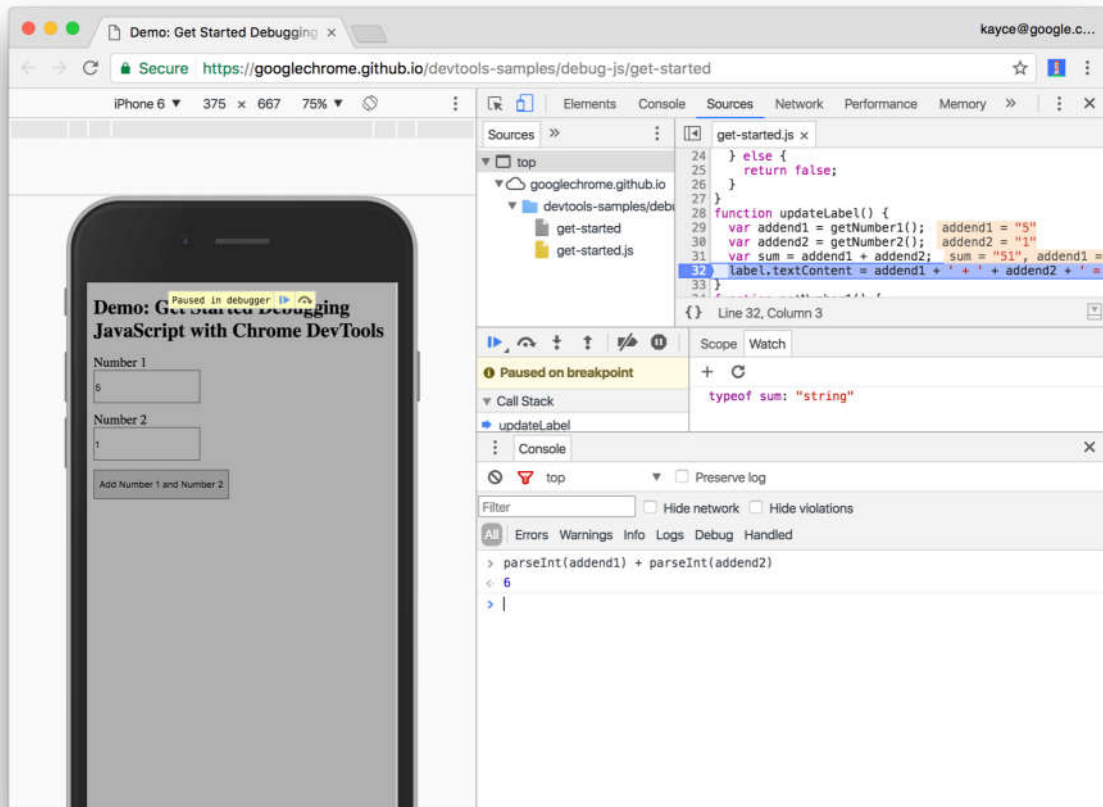




图 1：对 `parseInt(addend1) + parseInt(addend2)` 求值后的 Console 抽屉。

第 6 步：应用修正

您已找到错误的潜在解决方法。剩下的工作就是编辑代码后重新运行演示页面来测试修正效果。您不必离开 DevTools 就能应用修正。您可以直接在 DevTools UI 内编辑 JavaScript 代码。立即试一试：

1. 在 DevTools 的 **Sources** 面板上的代码编辑器中，将 `var sum = addend1 + addend2` 替换为 `var sum = parseInt(addend1) + parseInt(addend2);`。它就是您当前暂停位置上面那行代码。

2. 按 **Command+S** (Mac) 或 **Ctrl+S** (Windows、Linux) 保存更改。代码的背景色变为红色，这表示在 DevTools 内更改了脚本。
3. 点击 **Deactivate breakpoints** 。它变为蓝色，表示处于活动状态。如果进行了此设置，DevTools 会忽略您已设置的任何断点。
4. 点击 **Resume script execution** 。
5. 使用不同的值测试演示页面。现在演示页面应能正确计算求和。

切记，此工作流程只对运行在浏览器中的代码应用修正。它不会为所有运行您的页面的用户修正代码。要实现该目的，您需要修正运行在提供页面的服务器上的代码。

后续步骤

恭喜！现在您已掌握了在 DevTools 中调试 JavaScript 的基础知识。

本教程只向您介绍了两种设置断点的方法。DevTools 还提供了许多其他方法，其中包括：

- 仅在满足您指定的条件时触发的条件断点。
- 发生已捕获或未捕获异常时触发的断点。
- 当请求的网址与您提供的子字符串匹配时触发的 XHR 断点。

[为我演示所有断点](https://developers.google.com/web/tools/chrome-devtools/javascript/add-breakpoints?hl=zh-cn)

(<https://developers.google.com/web/tools/chrome-devtools/javascript/add-breakpoints?hl=zh-cn>)

有几个代码单步执行控件在本教程中未予说明。请点击以下链接，了解有关它们的更多信息。

[我想要掌握代码单步调试知识](https://developers.google.com/web/tools/chrome-devtools/javascript/step-code?hl=zh-cn#stepping_in_action)

(https://developers.google.com/web/tools/chrome-devtools/javascript/step-code?hl=zh-cn#stepping_in_action)

反馈

请通过回答下列问题帮助我们改进本教程。