你不知道的14种常用的javascript调试技巧

阳朝霞

阳朝霞

2 个月前

小编觉得这是一篇干货文，在此推荐给大家，减少大家js debug的时间，提高开发效率。对于里面的每一条，小编亲自试过还比较好用。

工欲善其事，必先利其器，这14个调试技巧主要针对于chrome浏览器和Firefox浏览器。会用到它们的开发者工具。

1. ‘debugger;’

一般，我们比较常用的判断代码是否执行到指定位置，一个是打断点，另外一个是通过输入console.log。这里还有一个好用的方法是-’debugger;’。 一旦你把它放在你的代码中，Chrome会在执行时自动停止。 你甚至可以用条件包装它，所以它只在你需要的时候运行。

```
if (thisThing) {// 这个thisThing可以写你自己的判断条件，
    debugger;
}
```

2.将对象显示为表格

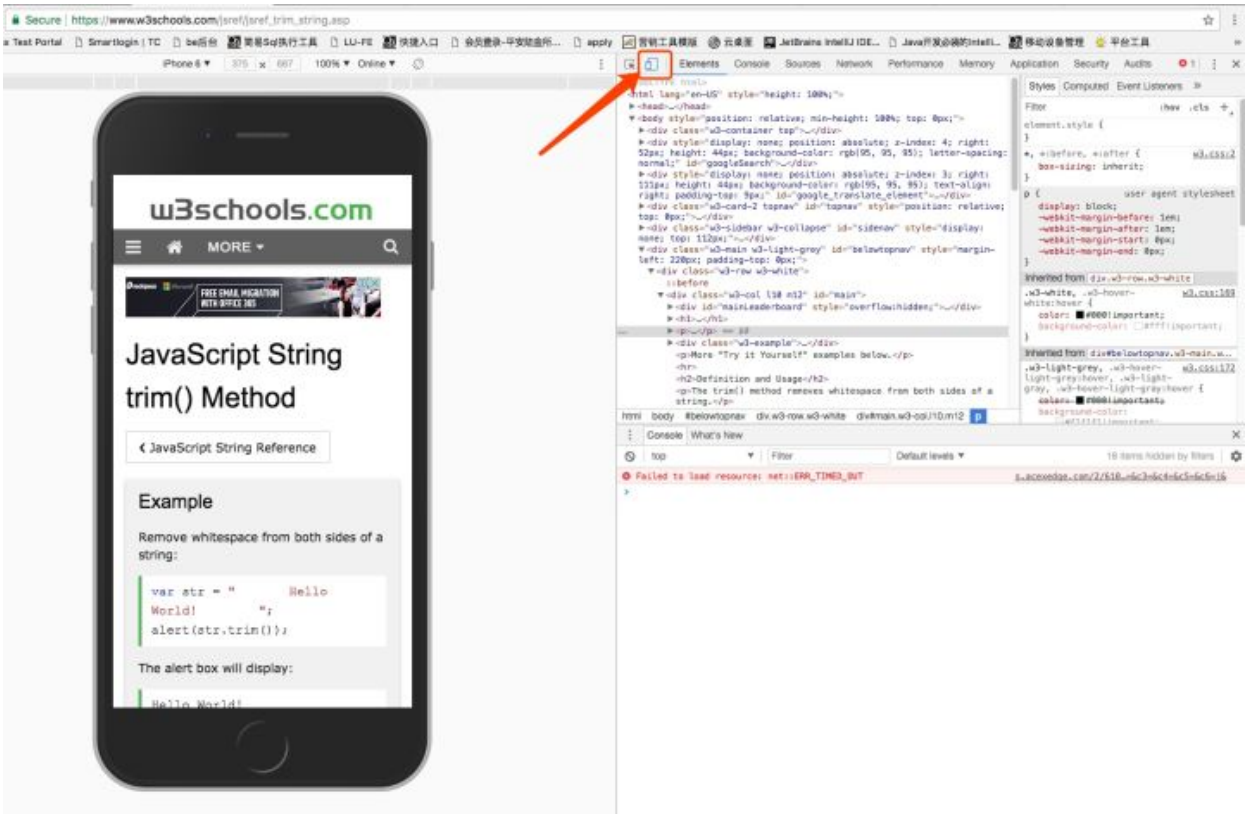
有时候，我们有一个复杂的对象，直观看起来不怎么方便。那么，我们这个时候可以用console.table()进行展示，直观方便，举个栗子：

```
var animals = [
    { animal: 'Horse', name: 'Henry', age: 43 },
    { animal: 'Dog', name: 'Fred', age: 13 },
    { animal: 'Cat', name: 'Frodo', age: 18 }
];
console.table(animals);
```

那么将会输出

3.调试移动端响应式web项目

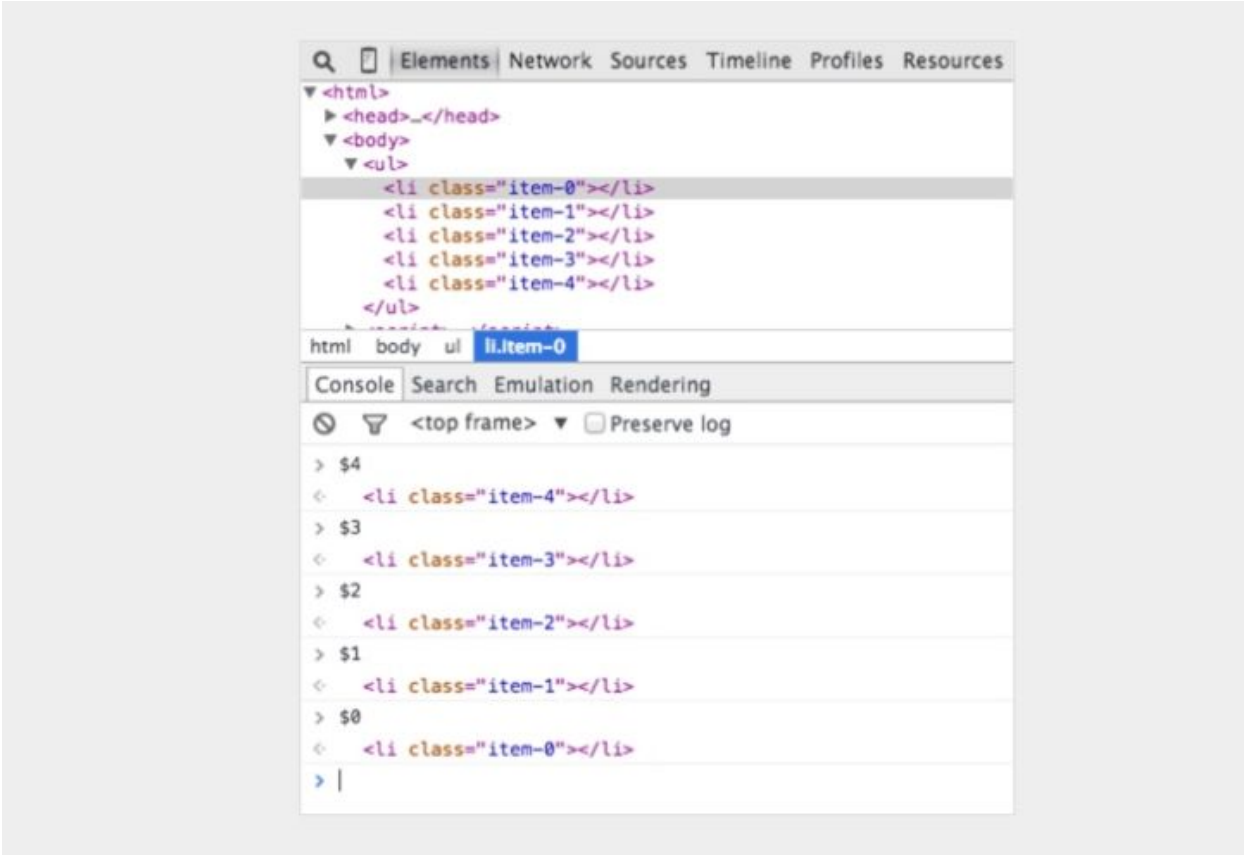
调出开发者工具，在右上角有个小手机的按钮，点击它可以任意切换视口，还可以调出酷炫的手机壳。各种分辨率也可以自己调整。



4.如何快速找到你的DOM元素

在元素面板中标记DOM元素并在控制台中使用它。 Chrome检查器会保留历史记录中的最后五个元素。

如果您按照“item-4”，“item-3”，“item-2”，“item-1”，“item-0”的顺序标记下列项目，则可以在控制台中像这样访问DOM节点（暂时还没想到用的场景）：



5.使用console.time（）和console.timeEnd（）来标记循环时间。

确切地知道循环需要执行多长时间是非常有用的，尤其是在调试慢循环时。 您甚至可以通过为该方法分配一个标签来设置多个定时器。 让我们看看它是如何工作的（这个很方便）：

```
console.time('Timer1');

var items = [];

for(var i = 0; i < 100000; i++){
   items.push({index: i});
}

console.timeEnd('Timer1');
```

6.获取函数的堆栈跟踪

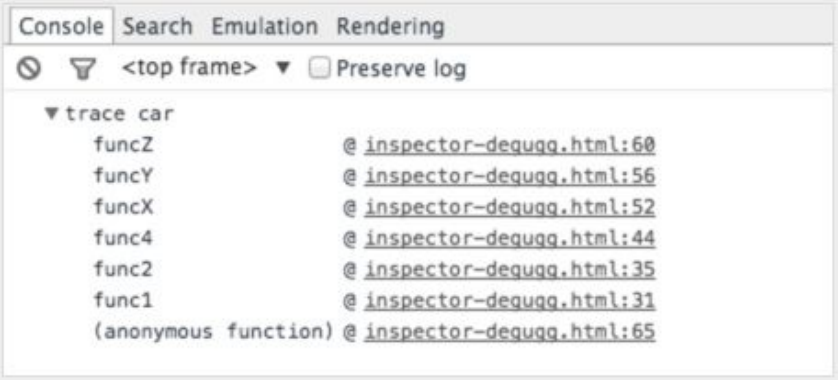
这个技巧我看了很久，比较抽象难懂。小编的理解是，有时候你创建一个函数，但是迭代生成了另外的好几个函数，而这些函数可能有调用或者依赖关系，

当我们在触发事件时，会想知道是什么导致了函数调用。而JavaScript不是一个非常结构化的语言，因此有时候很难知道发生了什么事情。 所以这个时候，trace就可以进行方便的跟踪了。

举个下面的栗子，可以看到第33行car实例中的函数调用funcZ的整个堆栈跟踪：

```
var car;
var func1 = function() {
 func2();
}

var func2 = function() {
 func4();
}
var func3 = function() {
}

var func4 = function() {
 car = new Car();
 car.funcX();
}
var Car = function() {
 this.brand = 'volvo';
 this.color = 'red';
 this.funcX = function() {
  this.funcY();
 }

 this.funcY = function() {
  this.funcZ();
 }

 this.funcZ = function() {
  console.trace('trace car')
 }
}
func1();
var car;
var func1 = function() {
 func2();
}
```

```
var func2 = function() {
 func4();
}
var func3 = function() {
}
var func4 = function() {
 car = new Car();
 car.funcX();
}
var Car = function() {
 this.brand = 'volvo';
 this.color = 'red';
 this.funcX = function() {
  this.funcY();
 }
 this.funcY = function() {
  this.funcZ();
 }
  this.funcZ = function() {
  console.trace('trace car')
 }
}
func1();
```



可以看到， func1 调用了 func2, func2调用了 func4，以此往上有着调用关系。这样，对于理清函数之间的调用关系，是非常方便的做法。

7.解压缩代码

有时候浏览器代码是压缩过的格式，可读性比较差，这时，我们可以点击开发者工具中代码右下角的大括号，这个具有美化代码的功能，类似于格式化。

8. 快速找到一个函数来调试

假设你想在一个函数中设置一个断点。

最常见的两种方法是：

* 在您的检查器中找到该行并添加一个断点。

* 在脚本中添加一个调试器。

在这两种解决方案中，您都必须在文件中单击以查找要调试的特定行。

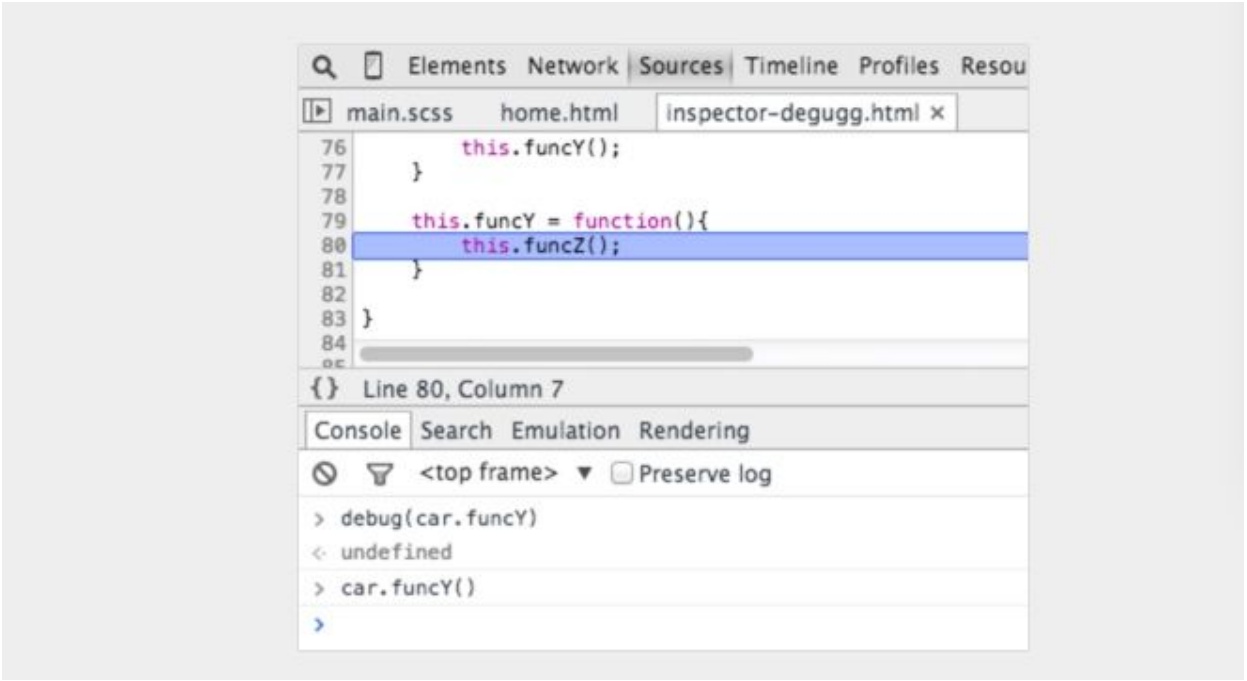
可能不太常见的是使用控制台。 在控制台中使用debug（funcName），脚本到达你传入的函数时会停止。

这种方式能很快的debug，但缺点是它不适用于私有或匿名函数。

```
var func1 = function() {
 func2();
};

var Car = function() {
 this.funcX = function() {
  this.funcY();
 }

 this.funcY = function() {
  this.funcZ();
 }
}
```

```
var car = new Car();
```

在console控制台中，输入debug(car.funcY)，当执行到这个函数时，那么脚本会自动停止。



9. 黑箱调试

小编的理解是，将无用的脚本屏蔽，只调试作者需要的脚本，这样程序就不会进入到无用的脚本中浪费时间。具体可以参考黑箱调试。小编会仔细阅读再跟大家慢慢分享的。

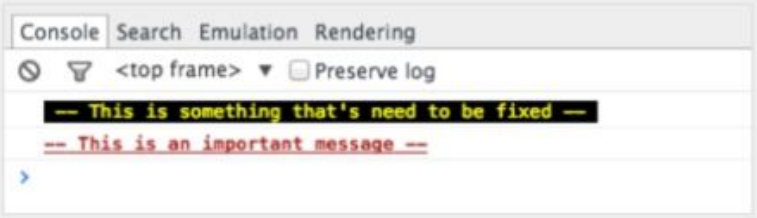
10. 在复杂的调试中找到重要的东西

在复杂的调试中，我们有时需要输出很多行。 你可以做的一件事情是保持更好的输出结构，就是使用更多的控制台功能，例如Console.log，console.debug，console.warn，console.info，console.error等等。 然后，您可以在您的检查器中对其进行过滤。 当您要调试JavaScript时，可以使用CSS制作您自己的结构化控制台消息（第一次知道console.log还能加样式，果然村里来的）：

```
console.todo = function(msg) {
 console.log(' % c % s % s % s', 'color: yellow; background -
color: black;', '-', msg, '-');
}

console.important = function(msg) {
 console.log(' % c % s % s % s', 'color: brown; font - weight:
bold; text - decoration: underline;', '-', msg, '-');
```

```
}

console.todo("This is something that' s need to be fixed");
console.important('This is an important message');
```

输出



例如：

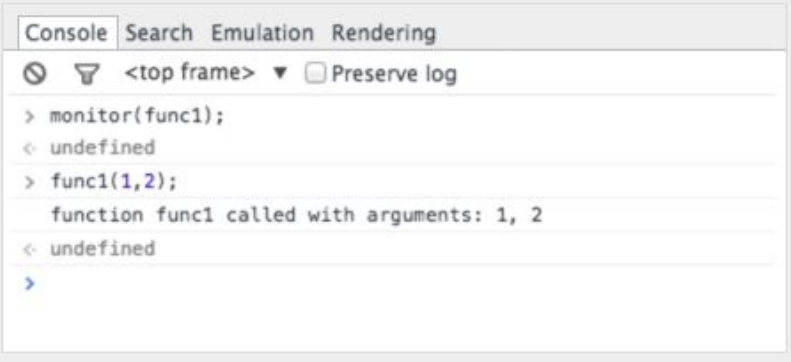
在console.log（）中，可以为字符串设置%s，为整数设置%i，为自定义样式设置%c。 你可以找到更好的方法来使用它。 如果您使用单个页面框架，则可能希望为视图消息创建一个样式，为模型，集合，控制器等创建另一个样式。也许还可以像wlog，clog和mlog一样使用你的想象力！

11.观察特定的函数调用及其参数

在Chrome控制台中，您可以关注特定的函数。 每次调用该函数时，都会使用传入的值进行记录。

```
var func1 = function(x, y, z) {
//....
};
```

将会输出



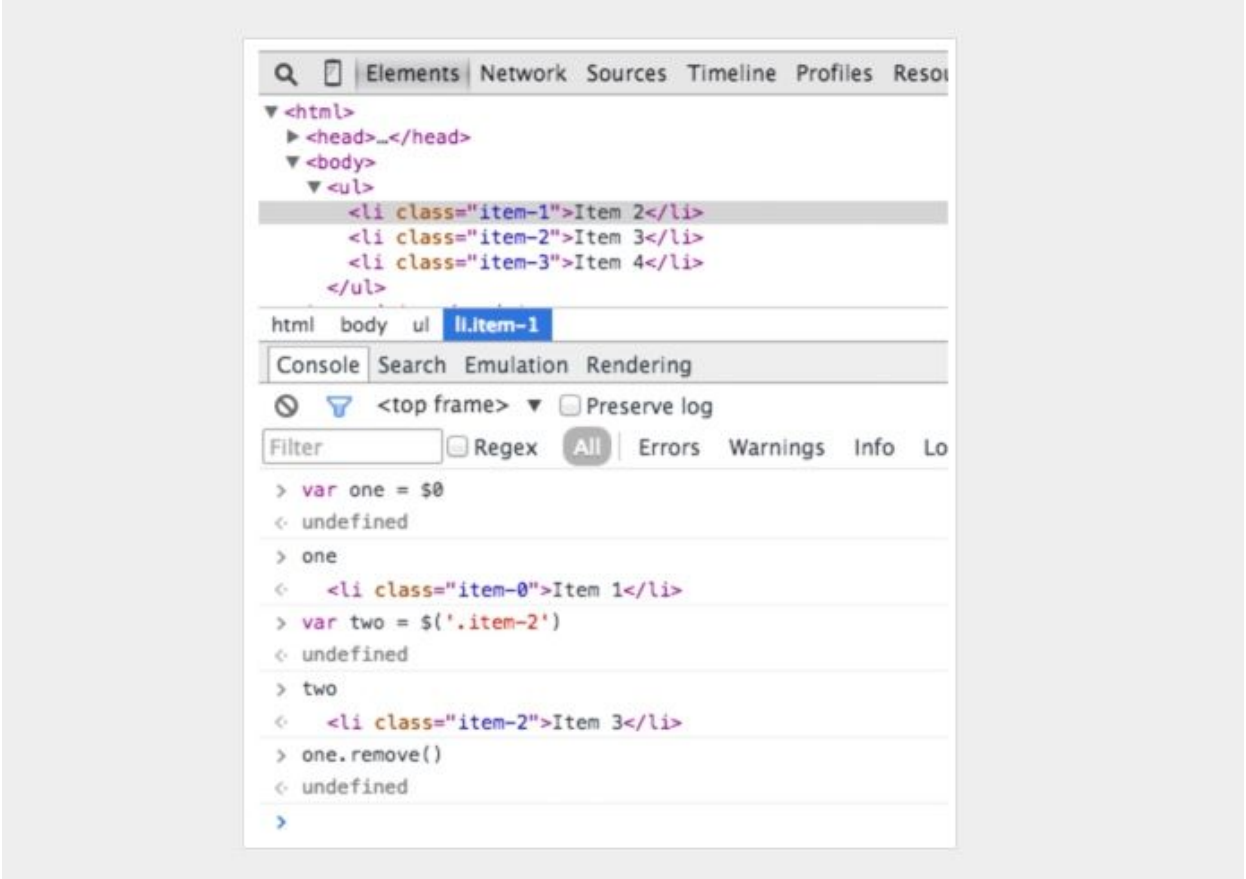
这是查看哪些参数传递给函数的好方法。 在上面的例子中，func1期望3个参数，但是只有2个参数被传入。如果在代码中没有处理这个参数，它可能导致一个可能的错误。

12.快速访问控制台中的元素

在控制台中执行querySelector的更快方法是使用美元符号。

- $('css-selector') 将返回CSS选择器的第一个匹配项。
- $('css-selector') 将返回所有这些。

如果你多次使用一个元素，值得把它保存为一个变量。



13.postman很好，但Firefox更快。

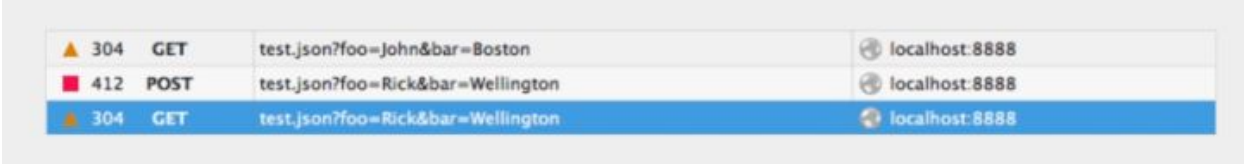
许多开发人员正在使用postman玩ajax请求。 postman很好，但打开一个新的浏览器窗口，写入新的请求对象，然后测试它们可能有点小麻烦。

有时使用浏览器更容易。

当你这样做时，如果你发送到一个密码安全的页面，你不再需要担心身份验证cookie。

打开Firefox开发者工具并转到网络选项卡。 右键单击所需的请求，然后选择编辑并重新发送。 现在你可以改变任何你想要的。 更改标题并编辑您的参数并点击重新发送。

下面我用不同的属性提出两次请求：



14.中断节点更改

DOM非常有趣。 有时候节点会改变，但你不知道发生了啥。 所以，当你需要调试JavaScript时，Chrome会在DOM元素发生更改时暂停。 你甚至可以监视它的属性。 在Chrome Inspector中，右键单击该元素，然后在设置中选择一个中断即可使用：