

Chrome 控制台实用指南

2017-07-13 伯乐在线 前端大全

(点击上方公众号，可快速关注)

作者：伯乐在线/秦至

[如有好文章投稿，请点击 → 这里了解详情](#)

前言

Chrome浏览器我想是每一个前端er必用工具之一吧，一部分原因是它速度快，体积不大，支持的新特性也比其它浏览器多，还有一部分我想就是因为它的控制台功能强大了吧，说它是神器一点也不过分，很方便。但其实很多开发者并没有用出控制台的精髓，只是使用简单的`console.log()`；其实控制台功能远远不止这么简单哦。

console.clear

`console.clear()`；清空控制台，这个应该和`console.log`知名度一样高吧。

console.log家族

先简单介绍一下chrome的控制台，打开chrome浏览器，按f12就可以轻松的打开控制台



如果你是一位开发者，我想`console.log`肯定是经常使用的了，我们主要看看`console.log`的几个兄弟：

- 1.console.log ('普通信息')
- 2.console.info ('提示性信息')
- 3.console.error ('错误信息')
- 4.console.warn ('警示信息')



大家都会用log，但很少有人能够很好地利用console.error， console.warn 等将输出到控制台的信息进行分类整理。他们功能区别不大，意义在于将输出到控制台的信息进行归类，或者说让它们更语义化。

如果再配合console.group 与console.groupEnd，可以将这种分类管理的思想发挥到极致。这适合于在开发一个规模很大模块很多很复杂的Web APP时，将各自的log信息分组到以各自命名空间为名称的组里面。

```
console.group("app.bundle");
console.warn("来自bundle模块的警告信息1");console.warn("来自bundle模块的警告信息2");
console.groupEnd();

console.group("app.bundle");
console.log("来自bundle模块的信息1");console.log("来自bundle模块的信息2");
console.groupEnd();
```

```
> console.group("app.bundle");
console.warn("来自bundle模块的警告信息1");console.warn("来自bundle模块的警告信息2");

console.groupEnd();
console.group("app.bundle");
console.log("来自bundle模块的信息1");console.log("来自bundle模块的信息2");
console.groupEnd();
```

▼ app.bundle

- ▲ 来自bundle模块的警告信息1
- ▲ 来自bundle模块的警告信息2

▼ app.bundle

- 来自bundle模块的信息1
- 来自bundle模块的信息2

这样的控制台信息看上去就一目了然了，就不用再为了找这是属于那一行代码输出的再翻一遍源码了。

另外，console.log家族还给我们提供了一个的API：第一个参数可以带一些格式化指令，比如%c,n;看下面这个炫酷的效果：

```
console.log('%chello world', 'background-image:-webkit-gradient( linear, left top, right top, color-stop(0, #f22), color-stop(0.15, #f2f), color-stop(0.3, #22f), color-stop(0.45, #2ff), color-stop(0.6, #2f2),color-stop(0.75, #2f2), color-stop(0.9, #ff2), color-stop(1, #f22) );color:transparent;-webkit-background-clip: text;font-size:5em;');
```

```
> console.log('%cHello world', 'background-image:-webkit-gradient( linear, left top, right top, color-stop(0, #f22), color-stop(0.15, #f2f), color-stop(0.3, #22f), color-stop(0.45, #2ff), color-stop(0.6, #2f2),color-stop(0.75, #2f2), color-stop(0.9, #ff2), color-stop(1, #f22) );color:transparent;-webkit-background-clip: text;font-size:5em;');
```

Hello world

当然，图片也是可以的，读者可以自行尝试，修改上述代码即可。

另外，console.log() 接收不定参数，参数间用逗号分隔，最终会输出会将它们以空白字符连接。

```
> console.log('%cHello', 'color:red;', 'World', 'Console.log');
Hello World Console.log
```

console.table

看着这种“黑魔法”是不是有种坑分的感觉呢，其实还不止哦！console.table可以让我们输出表格,示例：

```
var data = {code:200,content:[{'品名': '杜雷斯', '数量': 4}, {'品名': '冈本', '数量': 3}]};
console.table(data.content);
```

```
var data = {code:200,content:[{'品名':'杜蕾斯','数量':4},{'品名':'冈本','数量':3}]};  
console.table(data.content);
```

(index)	品名	数量
0	"杜蕾斯"	4
1	"冈本"	3

有的时候后端传回来一大串数据，是不是觉得直接console.log或是通过抓包工具查看都会让人晕头转向呢，这个时候正是console.table发挥作用的时候了，以表格的形式呈现数据，自然一目了然。

console.assert

```
var isDebug=false;  
console.assert(isDebug,'开发中的log信息。。。');
```

当你想代码满足某些条件时才输出信息到控制台，那么你大可不必写if或者三元表达式来达到目的，console.assert便是这样场景下一种很好的工具，它会先对传入的表达式进行断言，只有表达式为假时才输出相应信息到控制台。

```
> var isDebug=false;  
    console.assert(isDebug,'开发中的log信息。。。');
```

✖ ▶ Assertion failed: 开发中的log信息。。。

console.count

除了条件输出的场景，还有常见的场景是计数。

当你想统计某段代码执行了多少次时也大可不必自己去写相关逻辑，内置的console.count可以很地胜任这样的任务。

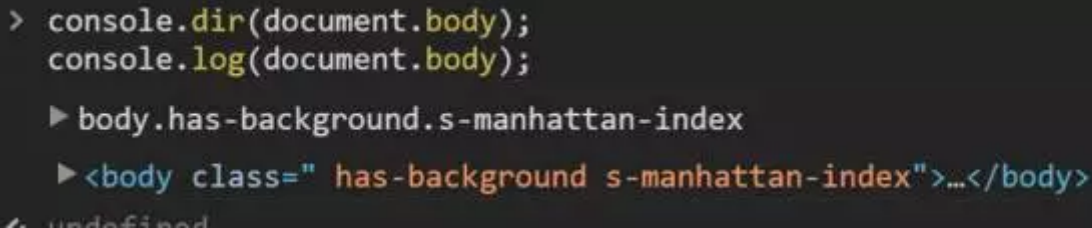
```
> function fun(){  
    //函数逻辑  
    console.count('fun被执行的次数');  
}  
fun();  
fun();  
fun();
```

fun被执行的次数: 1
fun被执行的次数: 2
fun被执行的次数: 3

console.dir

将DOM结点以JavaScript对象的形式输出到控制台，而console.log是直接将该DOM结点以DOM树的结构进行输出，与在元素审查时看到的结构是一致的。不同的展现形式，同样的优雅，各种体位任君选择反正就是方便与体贴。

```
console.dir(document.body);  
console.log(document.body);
```

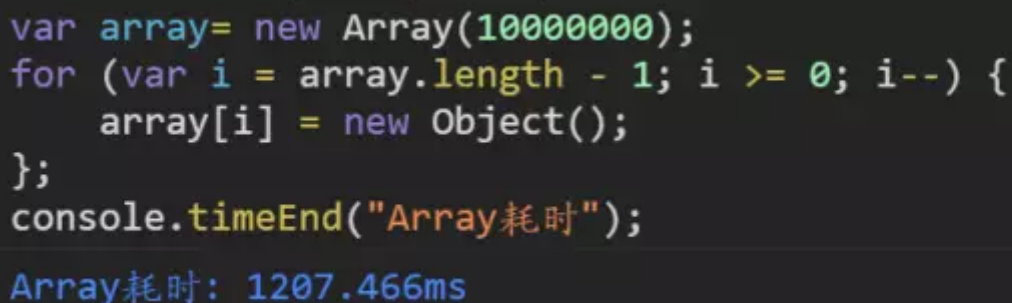


```
> console.dir(document.body);  
console.log(document.body);  
  
▶ body.has-background.s-manchattan-index  
▶ <body class=" has-background s-manchattan-index">...</body>
```

console.time & console.timeEnd

输出一些调试信息是控制台最常用的功能，当然，它的功能远不止于此。当做一些性能测试时，同样可以在这里很方便地进行。比如需要考量一段代码执行的耗时情况时，可以用console.time与console.timeEnd来做此事。

```
console.time("Array耗时");  
var array= new Array(10000000);  
for (var i = array.length - 1; i >= 0; i--) {  
    array[i] = new Object();  
};  
console.timeEnd("Array耗时");
```



```
var array= new Array(10000000);  
for (var i = array.length - 1; i >= 0; i--) {  
    array[i] = new Object();  
};  
console.timeEnd("Array耗时");  
  
Array耗时: 1207.466ms
```

当想要查看CPU使用相关的信息时，可以使用console.profile配合console.profileEnd来完成这个需求。

这一功能可以通过UI界面来完成，Chrome 开发者工具里面有个tab便是Profile。使用方法和console.time基本一样，其实time开发者工具里也有个tab就是timeline。关于console.prefile博

主就不做多余的介绍。想要做更多了解的读者可以看[这里](#)。

\$

讲真，米国程序员们真的很喜欢money啊（谁又不是呢），看看PHP就知道了，满屏的\$。而在Chrome的控制台里，\$用处同样是蛮多且方便的。

```
2+2//回车, 再
```

```
$._+1//回车得5
```

上面的\$_需要领悟其奥义才能使用得当，而\$0~\$4则代表了最近5个你选择过的DOM节点。

什么意思呢？在页面右击选择审查元素，然后在弹出来的DOM结点树上面随便点选，这些被点过的节点会被记录下来，而\$0会返回最近一次点选的DOM结点，以此类推，\$1返回的是上上次点选的DOM节点，最多保存了5个，如果不够5个，则返回undefined。



另外值得一赞的是，Chrome 控制台中原生支持类jQuery的选择器，也就是说你可以用\$加上熟悉的css选择器来选择DOM节点，多么滴熟悉。

```
$('body');
```

```
$$('div');
```



```
> $('body');
  $$('div');
< ▶ Array[314]
```

\$(selector)返回的是满足选择条件的首个DOM元素。

剥去她伪善的外衣，其实\$(selector)是原生JavaScript document.querySelector() 的封装。同时另一个命令\$\$ (selector)返回的是所有满足选择条件的元素的一个集合，是对 document.querySelectorAll() 的封装。

\$x(path)

将所匹配的节点放在一个数组里返回

```
$x("//p");
$x("//p[a]");
```

```
> $x("//p")
< [ <p class="s-skin-lm s-isindex-wrap"></p>, <p></p>, <p></p>, <p></p>, <p></p>, <p class="lh"></p>, <p>歌曲加载失败</p>,
  <p class="title"></p>, <p class="mylike"></p>]
> $x("//p[a]")
< [ <p class="lh"></p>, <p class="mylike"></p>]
```

\$x("//p")匹配所有的p节点，\$x("//p[a]");匹配所有子节点包含a的p节点

copy

```
copy(document.body)
```

然后你就可以Ctrl+v了。

注意：他不依附于任何全局变量比如window，所以其实在JS代码里是访问不了这个copy方法的，所以从代码层面来调用复制功能也就无从谈起。但愿有天浏览器会提供相应的JS实现吧~这样我们就可以通过js代码进行复制操作而不用再依赖Flash插件了。

keys & values

这是一对基友。前者返回传入对象所有属性名组成的数据，后者返回所有属性值组成的数组。具体请看下面的例子：

```
var tfboy={name:'wayou',gender:'unknown',hobby:'opposite to the gender'};
```

```
keys(tfboy);  
values(tfboy);
```

```
> var tfboy={name:'wayou',gender:'unknown',hobby:'opposite to the gender'};  
keys(tfboy);  
◀ ["name", "gender", "hobby"]  
> var tfboy={name:'wayou',gender:'unknown',hobby:'opposite to the gender'};  
values(tfboy);  
◀ ["wayou", "unknown", "opposite to the gender"]
```

monitor & unmonitor

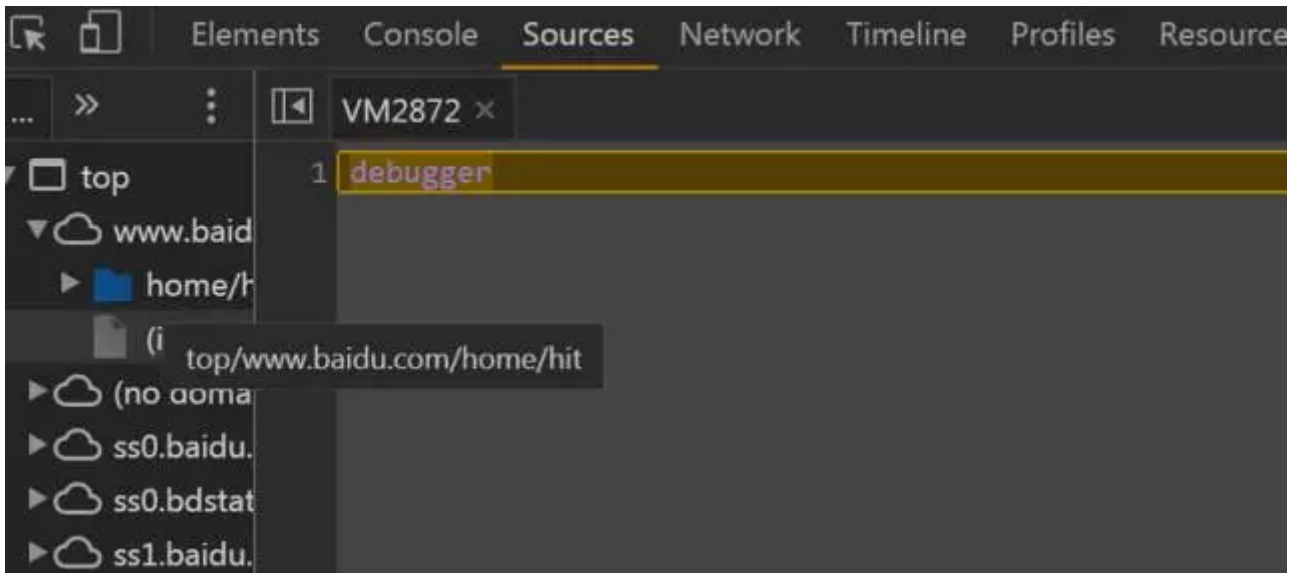
monitor(function), 它接收一个函数名作为参数, 比如function a, 每次a被执行了, 都会在控制台输出一条信息, 里面包含了函数的名称a及执行时所传入的参数。而unmonitor(function)便是用来停止这一监听。

```
function sayHello(name){  
    alert('hello,'+name);  
}  
monitor(sayHello);  
sayHello('damonare');  
sayHello('tjz');  
unmonitor(sayHello);
```

```
> function sayHello(name){  
    alert('hello,'+name);  
}  
monitor(sayHello);  
sayHello('damonare');  
  
sayHello('tjz');unmonitor(sayHello);  
function sayHello called with arguments: damonare  
function sayHello called with arguments: tjz
```

debug & undebug

debug同样也是接收一个函数名作为参数。当该函数执行时自动断下来以供调试, 类似于在该函数的入口处打了个断点, 可以通过debugger来做到, 同时也可以通过在Chrome开发者工具里找到相应源码然后手动打断点。而undebug 则是解除该断点。而其他还有好些命令则让人没有说的欲望, 因为好些都可以通过Chrome开发者工具的UI界面来操作并且比用在控制台输入要方便。



参考

- Console API文档;
- Command API;
- Chrome 控制台不完全指南 – 刘哇勇

觉得本文对你有帮助？请分享给更多人
关注「前端大全」，提升前端技能