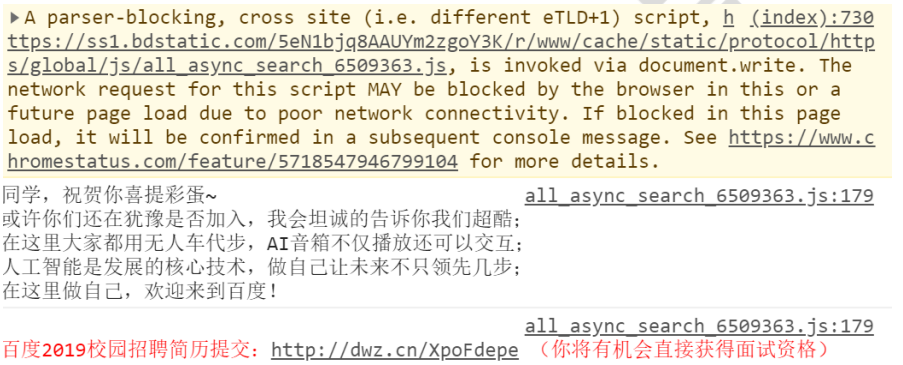
Console 详解

众所周知，调试是一个很重要的的过程，对于大家来说通过 console.log 进行调试是一个很普遍的形式。不过同学们可知道，console.log 只是用于单纯输出的么，里面有没有其他的用法。包括 console 还有没有其他方法。接下来我们来一起看一下 console 还有那些方法以及应用场景。

大家在访问百度的时候，会经常在控制台看到类似这样的内容：



有颜色，有背景。又是在控制台输出的。到底是谁做的呢，答案就是 console 对象做的。一直在说 console 对象，console 里面到底有什么，我们输出一下看一看

assert: ƒ assert()
clear: ƒ clear()
context: ƒ context()
count: ƒ count()
countReset: ƒ countReset()
debug: ƒ debug()
dir: ƒ dir()
dirxml: ƒ dirxml()
error: ƒ error()
group: ƒ group()
groupCollapsed: ƒ groupCollapsed()
groupEnd: ƒ groupEnd()
info: ƒ info()

log: ƒ log()
memory: (...)
profile: ƒ profile()
profileEnd: ƒ profileEnd()
table: ƒ table()
time: ƒ time()
timeEnd: ƒ timeEnd()
timeStamp: ƒ timeStamp()
trace: ƒ trace()
warn: ƒ warn()

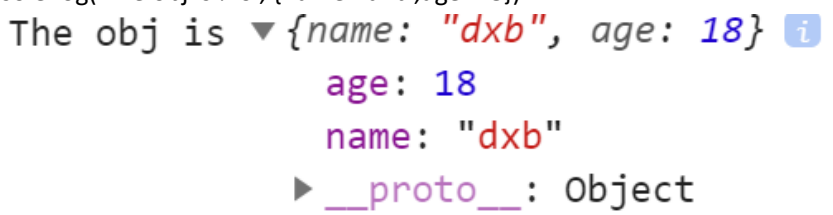
不看不知道一看吓一跳，这么多方法，然而我们只用了一个.log 确实有些屈才，接下来看看各个方法是如何使用的以及在何种场景下使用。

console.log()
很多人不知道经典的 console.log 其实有着丰富的函数特性。尽管大多数人只使用 console.log(object) 这种语法，但你仍然能写 console.log(object, otherObject, string) 并且它会将所有东西都整齐的打印出来。有时候确实很方便。
不止那些，这儿还有另一种格式：console.log(msg, values)。这个执行方式和 C 或者 C++的 printf 很相似。

```
console.log('%s love %s', 'I', 'you');
```
会准确的输出你所预期的东西。
I love you
这里的 %s 是占位符，对字符串进行占位 %s 可以理解为 sting，对于占位符还有

- %o 对对象进行占位（object）
- %d 对数字类型进行占位(digit)

```
console.log('The obj is %o', {name: 'dxb',age: 18})
```



还有一个比较有意思的占位符是 %c ,被称为 CSS 占位符，意味着，利用 console.log 输出的内容是可以携带样式的！

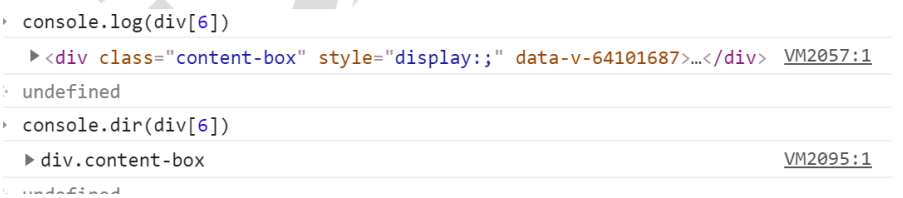
console.log('I am a %cbutton', 'color: white; background-color: orange; padding: 2px 5px; border-radius: 2px');

当然了，两个%c 中间的内容是 css 样式作用的区域，like this：
console.log('I am a %cbutton%c not a div', 'color: white; background-color: orange; padding: 2px 5px; border-radius: 2px','color: "auto"');

在这里面也是，通过两个占位符，对应着两个样式实现样式截断。当然这也不是真实的按钮。当大家输出内容醒目的时候可以使用。
我们先实现 %c 怎么办呢，既然转义一次，在转一次就好了。
Console.log（’%%c’）就可以了

console.dir()

通常来看，console.dir() 功能和 log() 非常相似，尽管看起来有略微不同。



下拉小箭头展示的对象信息和 console.log 的视图里一样。但是在你观察元素节点的时候，两者结果会非常有趣并且截然不同。

这是 log 输入 element 的输出：

```
▼<div class="content-box" style="display:;" data-v-64101687>
    <div class="close ion-close-round" data-v-64101687></div>
  ▶<div class="header" data-v-64101687>…</div>
    <div class="desc" data-v-64101687>凡未购买过小册的用户，均可领取三张 5 折新
    人专享券，购买小册时自动使用专享券，最高可节省 45 元。</div>
  ▶<div class="tickets" data-v-64101687>…</div>
    <div class="remark" data-v-64101687>注：专享券的使用期限在领券的七天内。
    </div>
    <div class="submit-btn" data-v-64101687>一键领取</div>
  </div>
```

我打开了一些元素节点。清晰的展示了 DOM 节点，一览无余，而且我们还可以跳转到子 DOM 节点。但是 console.dir(div[6]) 给我们一个意外不同的输出。

```
                                                                VM2095:1
▼div.content-box
    accessKey: ""
    align: ""
    assignedSlot: null
  ▶attributeStyleMap: StylePropertyMap {size: 0}
  ▶attributes: NamedNodeMap {0: class, 1: style, 2: data-v-64101687, class: …
    autocapitalize: ""
    baseURI: "https://juejin.im/post/5bf64218e51d45194266acb7"
    childElementCount: 6
  ▶childNodes: NodeList(6) [div.close.ion-close-round, div.header, div.desc…
  ▶children: HTMLCollection(6) [div.close.ion-close-round, div.header, div.…
  ▶classList: DOMTokenList ["content-box", value: "content-box"]
    className: "content-box"
    clientHeight: 0
    clientLeft: 0
    clientTop: 0
    clientWidth: 0
    contentEditable: "inherit"
  ▶dataset: DOMStringMap {v-64101687: ""}
```

这是一种更对象化的方式去观察元素节点。也许在某些像监测元素节点的时候，这样的结果才是你所想要的。

console.warn()

可能是 log() 最直接明显的替换，你可以用相同的方式使用 console.warn()。唯一的区别在于输出是一抹黄色。确切的说，输出是一个 warn 级别而不是一个 info 级别的信息，因此浏览器的处理稍稍有些不同。在一堆杂乱的输出中高亮你的输出是很有效果的。

不过，这还有一个更大的优点。因为输出是一个 warn 级别而不是一个 info 级别，你可以将所有的 console.log 过滤掉只留下 console.warn。这有时在那些不停输出一堆无用和无意义的东西到浏览器的随意的应用程序中是非常有用。屏蔽干扰能更容易的看到你自己的输出。

Vue 中的部分源码中也是用的 console.warn()

```
// vue.js
tip = function (msg, vm) {
      if (hasConsole && (!config.silent)) {
         console.warn("[Vue tip]: " + msg + (
            vm ? generateComponentTrace(vm) : ''
         ));
      }
   };
```

console.table()

令人惊讶的是这个并没有广为人知，但是 console.table() 方法更偏向于一种方式展示列表形式的数据，这比只扔下原始的对象数组要更加整洁。举一个例子，下面是数据的列表。

```
const transactions = [{
   id: "7cb1-e041b126-f3b8",
   seller: "WAL0412",
   buyer: "WAL3023",
   price: 203450,
   time: 1539688433
},
{
   id: "1d4c-31f8f14b-1571",
   seller: "WAL0452",
   buyer: "WAL3023",
   price: 348299,
   time: 1539688433
},
{
   id: "b12c-b3adf58f-809f",
   seller: "WAL0012",
   buyer: "WAL2025",
   price: 59240,
   time: 1539688433
}];
```

如果使用 console.log 去列出以上信息，我们能得到一些中看不中用的输出：

>(3) [{…}, {…}, {…}]

这小箭头允许你点击并会展开这个数组，但这并不是我们想要的「一目了然」。

而 console.table(data) 的输出则对我们更为有帮助。

(index)	id	seller	buyer	price	time
0	"7cb1-e041b126-f…	"WAL0412"	"WAL3023"	203450	1539688433
1	"1d4c-31f8f14b-1…	"WAL0452"	"WAL3023"	348299	1539688433
2	"b12c-b3adf58f-8…	"WAL0012"	"WAL2025"	59240	1539688433

▸ Array(3)

第二个可选参数是你想要显示列表的某列。默认是整个列表，但是我们也能这样做。

> console.table(data, ["id", "price"]);

我们得到这样的输出，仅仅只展示 id 和 price。在有着大量不相关信息的庞杂对象中非常有用。index 列是自动生成的并且据我所知是不会消失。

(index)	id	price ▲
2	"b12c-b3adf58f-809f"	59240
0	"7cb1-e041b126-f3b8"	203450
1	"1d4c-31f8f14b-1571"	348299

▸ Array(3)

值得一提的是在最右一列头部的右上角有个箭头可以颠倒次序。点击了它，会排序整个列。非常方便的找出一列的最大或者最小值，或者只是得到不同的数据展示形式。这个功能特性并没有做什么，只是对列的展示。但总会是有用的。

console.table() 只有处理最多 1000 行的数据的能力，所以它可能并不适用于所有的数据集合。

console.assert()

一个经常被忽视的实用的函数，assert() 在第一个参数是 falsey 时和 log() 一样。当第一个参数为真值时也什么都不做。

这个在你需要循环（或者不同的函数调用）并且只有一个要显示特殊的行为的场景下特别有用。本质上和这个是一样的。

```
if (object.whatever === 'value') {
   console.log(object);
}
```

澄清一下，当我说「一样」的时候，我本应该说是做相反的事。所以你需要变换一下场合。

所以，假设我们上面的值在时间戳里有一个 null 或者 0，这会破坏我们代码日期格式。

```
console.assert(tx.timestamp, tx);
```

当和任何有效的事物对象一起使用时会跳过。但是有一个触发了我们的日志记录，因为时间戳在 0 和 null 时为假值。

有时我们想要更加复杂的场景。举个例子，我们看到了关于用户 WAL0412

的数据问题并且想要只展示来自它们的事务。这将会是一个非常简便的方案。

```
console.assert(tx.buyer === 'WAL0412', tx);
```

看起来正确，但是并不奏效。牢记，场景必须是为否定态,我门想要的是断言，而不是过滤。

```
console.assert(tx.buyer !== 'WAL0412', tx);
```

我们想做的就是这样。在那种情况下，所有不是 WAL0412 号顾客的事务都为真值，只留下那些符合的事务。或者，也不完全是。

诸如此类，console.assert() 并不是一直都很管用。但是在特定的场景下会是最优雅的的解决方法。

console.count()

另外一个合适的用法是，将 console 作为一个计数器使用。

```
for(let i = 0; i < 10000; i++) {
   if(i % 2) {
      console.count('odds');
   }
   if(!(i % 5)) {
      console.count('multiplesOfFive');
   }
   if(isPrime(i)) {
      console.count('prime');
   }
}
```

这不是一段有用的代码，并且有点抽象。我也不打算去证明 isPrime 函数，只假设可以运行。

我们将得到应该是这样的列表

```
odds: 1
odds: 2
prime: 1
odds: 3
multiplesOfFive: 1
prime: 2
odds: 4
prime: 3
odds: 5
multiplesOfFive: 2
...
```

以及剩下的。在你只想列出索引，或者想保留一次（或多次）计数的情况下非常有用。

你也能像那样使用 console.count()，不需要参数。使用 default 调用。

这还有关联函数 console.countReset()，如果你希望重置计数器可以使用它。

console.trace()

这在简单的数据中演示更加困难。在你试图找出有问题的内部类或者库的调用这一块是它最擅长。

举个例子，这儿可能有 12 个不同的组件正在调用一个服务，但是其中一个没有正确配置依赖。

```
export default class CupcakeService {

   constructor(dataLib) {
      this.dataLib = dataLib;
      if(typeof dataLib !== 'object') {
         console.log(dataLib);
         console.trace();
      }
   }
   ...
}
```

这里单独使用 console.log() 我们只能知道执行了哪一个基础库，并不知道执行的具体位置。但是，堆栈轨迹会清楚的告诉我们问题在于 Dashboard.js，我们从中发现 new CupcakeService(false) 是造成出错的罪魁祸首。

console.time()

console.time() 是专门用于监测操作的时间开销的函数，也是监测 JavaScript 细微时间的更好的方式。

```
function slowFunction(number) {
   var functionTimerStart = new Date().getTime();
   // something slow or complex with the numbers.
   // Factorials, or whatever.
   var functionTime = new Date().getTime() - functionTimerStart;
   console.log(`Function time: ${ functionTime }`);
}
var start = new Date().getTime();

for (i = 0; i < 100000; ++i) {
```

```
  slowFunction(i);
}

var time = new Date().getTime() - start;
console.log(`Execution time: ${ time }`);
```

这是一个过时的方法。我指的同样还有上面的 console.log。大多数人没有意识到这里你本可以使用模版字符串和插值法。它时不时的会帮助到你。那么让我们更新一下上面的代码。

```
const slowFunction = number =>    {
  console.time('slowFunction');
  // something slow or complex with the numbers.
  // Factorials, or whatever.
  console.timeEnd('slowFunction');
}
console.time();

for (i = 0; i < 100000; ++i) {
  slowFunction(i);
}
console.timeEnd();
```

我现在不需要去做任何算术或者设置临时变量。

console.group()

如今我们可能在大多数 console 中要输出高级和复杂的东西。分组可以让你归纳这些。尤其是让你能使用嵌套。它擅长展示代码中存在的结构关系。

```
// this is the global scope
let number = 1;
console.group('OutsideLoop');
console.log(number);
console.group('Loop');
for (let i = 0; i < 5; i++) {
  number = i + number;
  console.log(number);
}
console.groupEnd();
console.log(number);
console.groupEnd();
console.log('All done now');
```

这又有一点难以理解。你可以看看这里的输出。

这并不是很有用，但是你能看到其中一些是如何组合的。

```
class MyClass {
   constructor(dataAccess) {
      console.group('Constructor');
      console.log('Constructor executed');
      console.assert(typeof dataAccess === 'object',
         'Potentially incorrect dataAccess object');
      this.initializeEvents();
      console.groupEnd();
   }
   initializeEvents() {
      console.group('events');
      console.log('Initialising events');
      console.groupEnd();
   }
}
let myClass = new MyClass(false);
```

很多工作和代码在调试信息上可能并不是那么有用。但是仍然是一个有意思的办法，同时你可以看到它使你打印的上下文是多么的清晰。

关于这个，还有最后一点需要说明，那就是 console.groupCollapsed。功能上和 console.group 一样，但是分组块一开始是折叠的。它没有得到很好的支持，但是如果你有一个无意义的庞大的分组并想默认隐藏它，可以试试这个。

结语

这里真的没有过多的总结。在你可能只想得到比 console.log(pet) 的信息更多一点，并且不太需要调试器的时候，上面这些工具都可能帮到你。

也许最有用的是 console.table，但是其他方法也都有其适用的场景。在我们想要调试一些东西时，我热衷于使用 console.assert，但那也只在某种特殊情况下。