console 对象与控制台

console 对象

`console`对象是 JavaScript 的原生对象，它有点像 Unix 系统的标准输出`stdout`和标准错误`stderr`，可以输出各种信息到控制台，并且还提供了很多有用的辅助方法。

`console`的常见用途有两个。

- 调试程序，显示网页代码运行时的错误信息。
- 提供了一个命令行接口，用来与网页代码互动。

`console`对象的浏览器实现，包含在浏览器自带的开发工具之中。以 Chrome 浏览器的“开发者工具”（Developer Tools）为例，可以使用下面三种方法的打开它。

1. 按 F12 或者`Control + Shift + i`（PC）/ `Command + Option + i`（Mac）。
2. 浏览器菜单选择“工具/开发者工具”。
3. 在一个页面元素上，打开右键菜单，选择其中的“Inspect Element”。

打开开发者工具以后，顶端有多个面板。

- **Elements**：查看网页的 HTML 源码和 CSS 代码。
- **Resources**：查看网页加载的各种资源文件（比如代码文件、字体文件 CSS 文件等），以及在硬盘上创建的各种内容（比如本地缓存、Cookie、Local Storage等）。
- **Network**：查看网页的 HTTP 通信情况。
- **Sources**：查看网页加载的脚本源码。
- **Timeline**：查看各种网页行为随时间变化的情况。
- **Performance**：查看网页的性能情况，比如 CPU 和内存消耗。
- **Console**：用来运行 JavaScript 命令。

这些面板都有各自的用途，以下只介绍`Console`面板（又称为控制台）。

`Console`面板基本上就是一个命令行窗口，你可以在提示符下，键入各种命令。

console 对象的静态方法

`console`对象提供的各种静态方法，用来与控制台窗口互动。

console.log()，console.info()，console.debug()

`console.log`方法用于在控制台输出信息。它可以接受一个或多个参数，将它们连接起来输出。

```javascript
console.log('Hello World')
```

```
// Hello World
console.log('a', 'b', 'c')
// a b c
```

`console.log`方法会自动在每次输出的结尾，添加换行符。

```javascript
console.log(1);
console.log(2);
console.log(3);
// 1
// 2
// 3
```

如果第一个参数是格式字符串（使用了格式占位符），`console.log`方法将依次用后面的参数替换占位符，然后再进行输出。

```javascript
console.log(' %s + %s = %s', 1, 1, 2)
//  1 + 1 = 2
```

上面代码中，`console.log`方法的第一个参数有三个占位符（`%s`），第二、三、四个参数会在显示时，依次替换掉这个三个占位符。

`console.log`方法支持以下占位符，不同类型的数据必须使用对应的占位符。

- `%s` 字符串
- `%d` 整数
- `%i` 整数
- `%f` 浮点数
- `%o` 对象的链接
- `%c` CSS 格式字符串

```javascript
var number = 11 * 9;
var color = 'red';

console.log('%d %s balloons', number, color);
// 99 red balloons
```

上面代码中，第二个参数是数值，对应的占位符是`%d`，第三个参数是字符串，对应的占位符是`%s`。

使用`%c`占位符时，对应的参数必须是 CSS 代码，用来对输出内容进行 CSS 渲染。

```javascript
console.log(
  '%cThis text is styled!',
  'color: red; background: yellow; font-size: 24px;'
)
```

上面代码运行后，输出的内容将显示为黄底红字。

`console.log`方法的两种参数格式，可以结合在一起使用。

```javascript
console.log(' %s + %s ', 1, 1, '= 2')
// 1 + 1  = 2
```

如果参数是一个对象，`console.log`会显示该对象的值。

```javascript
console.log({foo: 'bar'})
// Object {foo: "bar"}
console.log(Date)
// function Date() { [native code] }
```

上面代码输出`Date`对象的值，结果为一个构造函数。

`console.info`是`console.log`方法的别名，用法完全一样。只不过`console.info`方法会在输出信息的前面，加上一个蓝色图标。

`console.debug`方法与`console.log`方法类似，会在控制台输出调试信息。但是，默认情况下，`console.debug`输出的信息不会显示，只有在打开显示级别在`verbose`的情况下，才会显示。

`console`对象的所有方法，都可以被覆盖。因此，可以按照自己的需要，定义`console.log`方法。

```javascript
['log', 'info', 'warn', 'error'].forEach(function(method) {
  console[method] = console[method].bind(
    console,
    new Date().toISOString()
  );
});

console.log("出错了！ ");
// 2014-05-18T09:00.000Z 出错了！
```

上面代码表示，使用自定义的`console.log`方法，可以在显示结果添加当前时间。

console.warn()，console.error()

`warn`方法和`error`方法也是在控制台输出信息，它们与`log`方法的不同之处在于，`warn`方法输出信息时，在最前面加一个黄色三角，表示警告；`error`方法输出信息时，在最前面加一个红色的叉，表示出错。同时，还会高亮显示输出文字和错误发生的堆栈。其他方面都一样。

```javascript
console.error('Error: %s (%i)', 'Server is not responding', 500)
// Error: Server is not responding (500)
console.warn('Warning! Too few nodes (%d)', document.childNodes.length)
// Warning! Too few nodes (1)
```

可以这样理解，`log`方法是写入标准输出（`stdout`），`warn`方法和`error`方法是写入标准错误（`stderr`）。

console.table()

对于某些复合类型的数据，`console.table`方法可以将其转为表格显示。

```javascript
var languages = [
  { name: "JavaScript", fileExtension: ".js" },
  { name: "TypeScript", fileExtension: ".ts" },
  { name: "CoffeeScript", fileExtension: ".coffee" }
];

console.table(languages);
```

上面代码的`language`变量，转为表格显示如下。

```
(index)|name|fileExtension
-------|----|-------------
0|"JavaScript"|".js"
1|"TypeScript"|".ts"
2|"CoffeeScript"|".coffee"
```

下面是显示表格内容的例子。

```javascript
var languages = {
  csharp: { name: "C#", paradigm: "object-oriented" },
  fsharp: { name: "F#", paradigm: "functional" }
};

console.table(languages);
```

上面代码的`language`，转为表格显示如下。

```
(index)|name|paradigm
-------|----|--------
csharp|"C#"|"object-oriented"
fsharp|"F#"|"functional"
```

console.count()

`count`方法用于计数，输出它被调用了多少次。

```javascript
function greet(user) {
  console.count();
  return 'hi ' + user;
}

greet('bob')
//  : 1
// "hi bob"

greet('alice')
//  : 2
// "hi alice"

greet('bob')
//  : 3
// "hi bob"
```

上面代码每次调用`greet`函数，内部的`console.count`方法就输出执行次数。

该方法可以接受一个字符串作为参数，作为标签，对执行次数进行分类。

```javascript
function greet(user) {
  console.count(user);
  return "hi " + user;
}

greet('bob')
// bob: 1
// "hi bob"

greet('alice')
// alice: 1
// "hi alice"

greet('bob')
// bob: 2
// "hi bob"
```

上面代码根据参数的不同，显示`bob`执行了两次，`alice`执行了一次。

console.dir()，console.dirxml()

`dir`方法用来对一个对象进行检查（inspect），并以易于阅读和打印的格式显示。

```javascript
console.log({f1: 'foo', f2: 'bar'})
// Object {f1: "foo", f2: "bar"}

console.dir({f1: 'foo', f2: 'bar'})
// Object
//   f1: "foo"
//   f2: "bar"
//   __proto__: Object
```

上面代码显示`dir`方法的输出结果，比`log`方法更易读，信息也更丰富。

该方法对于输出 DOM 对象非常有用，因为会显示 DOM 对象的所有属性。

```javascript
console.dir(document.body)
```

Node 环境之中，还可以指定以代码高亮的形式输出。

```javascript
console.dir(obj, {colors: true})
```

`dirxml`方法主要用于以目录树的形式，显示 DOM 节点。

```javascript
console.dirxml(document.body)
```

如果参数不是 DOM 节点，而是普通的 JavaScript 对象，`console.dirxml`等同于`console.dir`。

```javascript
console.dirxml([1, 2, 3])
// 等同于
console.dir([1, 2, 3])
```

console.assert()

`console.assert`方法主要用于程序运行过程中，进行条件判断，如果不满足条件，就显示一个错误，但不会中断程序执行。这样就相当于提示用户，内部状态不正确。

它接受两个参数，第一个参数是表达式，第二个参数是字符串。只有当第一个参数为`false`，才会提示有错误，在控制台输出第二个参数，否则不会有任何结果。

```javascript
console.assert(false, '判断条件不成立')
// Assertion failed: 判断条件不成立

// 相当于
try {
  if (!false) {
    throw new Error('判断条件不成立');
  }
} catch(e) {
  console.error(e);
}
```

下面是一个例子，判断子节点的个数是否大于等于500。

```javascript
console.assert(list.childNodes.length < 500, '节点个数大于等于500')
```

上面代码中，如果符合条件的节点小于500个，不会有任何输出；只有大于等于500时，才会在控制台提示错误，并且显示指定文本。

console.time()，console.timeEnd()

这两个方法用于计时，可以算出一个操作所花费的准确时间。

```javascript
console.time('Array initialize');

var array= new Array(1000000);
for (var i = array.length - 1; i >= 0; i--) {
  array[i] = new Object();
};

console.timeEnd('Array initialize');
// Array initialize: 1914.481ms
```

`time`方法表示计时开始，`timeEnd`方法表示计时结束。它们的参数是计时器的名称。调用`timeEnd`方法之后，控制台会显示“计时器名称: 所耗费的时间”。

console.group()，console.groupEnd()，console.groupCollapsed()

`console.group`和`console.groupEnd`这两个方法用于将显示的信息分组。它只在输出大量信息时有用，分在一组的信息，可以用鼠标折叠/展开。

```javascript
console.group('一级分组');
console.log('一级分组的内容');

console.group('二级分组');
console.log('二级分组的内容');

console.groupEnd(); // 二级分组结束
console.groupEnd(); // 一级分组结束
```

上面代码会将“二级分组”显示在“一级分组”内部，并且“一级分组”和“二级分组”前面都有一个折叠符号，可以用来折叠本级的内容。

`console.groupCollapsed`方法与`console.group`方法很类似，唯一的区别是该组的内容，在第一次显示时是收起的（collapsed），而不是展开的。

```javascript
console.groupCollapsed('Fetching Data');

console.log('Request Sent');
console.error('Error: Server not responding (500)');

console.groupEnd();
```

上面代码只显示一行”Fetching Data“，点击后才会展开，显示其中包含的两行。

console.trace()，console.clear()

`console.trace`方法显示当前执行的代码在堆栈中的调用路径。

```javascript
console.trace()
// console.trace()
//   (anonymous function)
//   InjectedScript._evaluateOn
//   InjectedScript._evaluateAndWrap
//   InjectedScript.evaluate
```

`console.clear`方法用于清除当前控制台的所有输出，将光标回置到第一行。如果用户选中了控制台的“Preserve log”选项，`console.clear`方法将不起作用。

控制台命令行 API

浏览器控制台中，除了使用`console`对象，还可以使用一些控制台自带的命令行方法。

（1）`$_`

`$_`属性返回上一个表达式的值。

```javascript
2 + 2
// 4
$_
// 4
```

（2）`$0` - `$4`

控制台保存了最近5个在 Elements 面板选中的 DOM 元素，`$0`代表倒数第一个（最近一个），`$1`代表倒数第二个，以此类推直到`$4`。

（3）`$(selector)`

`$(selector)`返回第一个匹配的元素，等同于`document.querySelector()`。注意，如果页面脚本对`$`有定义，则会覆盖原始的定义。比如，页面里面有 jQuery，控制台执行`$(selector)`就会采用 jQuery 的实现，返回一个数组。

（4）`$$(selector)`

`$$(selector)`返回选中的 DOM 对象，等同于`document.querySelectorAll`。

（5）`$x(path)`

`$x(path)`方法返回一个数组，包含匹配特定 XPath 表达式的所有 DOM 元素。

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

上面代码返回所有包含`a`元素的`p`元素。

（6）`inspect(object)`

`inspect(object)`方法打开相关面板，并选中相应的元素，显示它的细节。DOM 元素在`Elements`面板中显示，比如`inspect(document)`会在 Elements 面板显示`document`元素。JavaScript 对象在控制台面板`Profiles`面板中显示，比如`inspect(window)`。

（7）`getEventListeners(object)`

`getEventListeners(object)`方法返回一个对象，该对象的成员为`object`登记了回调函数的各种事件（比如`click`或`keydown`），每个事件对应一个数组，数组的成员为该事件的回调函数。

（8）`keys(object)`，`values(object)`

`keys(object)`方法返回一个数组，包含`object`的所有键名。

`values(object)`方法返回一个数组，包含`object`的所有键值。

```javascript
var o = {'p1': 'a', 'p2': 'b'};

keys(o)
// ["p1", "p2"]
values(o)
// ["a", "b"]
```

（9）`monitorEvents(object[, events])，unmonitorEvents(object[, events])`

`monitorEvents(object[, events])`方法监听特定对象上发生的特定事件。事件发生时，会返回一个`Event`对象，包含该事件的相关信息。`unmonitorEvents`方法用于停止监听。

```javascript
monitorEvents(window, "resize");
monitorEvents(window, ["resize", "scroll"])
```

上面代码分别表示单个事件和多个事件的监听方法。

```javascript
monitorEvents($0, 'mouse');
unmonitorEvents($0, 'mousemove');
```

上面代码表示如何停止监听。

`monitorEvents`允许监听同一大类的事件。所有事件可以分成四个大类。

- mouse："mousedown", "mouseup", "click", "dblclick", "mousemove", "mouseover", "mouseout", "mousewheel"
- key："keydown", "keyup", "keypress", "textInput"
- touch："touchstart", "touchmove", "touchend", "touchcancel"
- control："resize", "scroll", "zoom", "focus", "blur", "select", "change", "submit", "reset"

```javascript
```

```
monitorEvents($("#msg"), "key");
```

上面代码表示监听所有`key`大类的事件。

（10）其他方法

命令行 API 还提供以下方法。

- `clear()`：清除控制台的历史。
- `copy(object)`：复制特定 DOM 元素到剪贴板。
- `dir(object)`：显示特定对象的所有属性，是`console.dir`方法的别名。
- `dirxml(object)`：显示特定对象的 XML 形式，是`console.dirxml`方法的别名。

debugger 语句

`debugger`语句主要用于除错，作用是设置断点。如果有正在运行的除错工具，程序运行到`debugger`语句时会自动停下。如果没有除错工具，`debugger`语句不会产生任何结果，JavaScript 引擎自动跳过这一句。

Chrome 浏览器中，当代码运行到`debugger`语句时，就会暂停运行，自动打开脚本源码界面。

```javascript
for(var i = 0; i < 5; i++){
  console.log(i);
  if (i === 2) debugger;
}
```

上面代码打印出0，1，2以后，就会暂停，自动打开源码界面，等待进一步处理。

参考链接

- Chrome Developer Tools, [Using the Console](https://developers.google.com/chrome-developer-tools/docs/console)
- Matt West, [Mastering The Developer Tools Console](http://blog.teamtreehouse.com/mastering-developer-tools-console)
- Firebug Wiki, [Console API](https://getfirebug.com/wiki/index.php/Console_API)
- Axel Rauschmayer, [The JavaScript console API](http://www.2ality.com/2013/10/console-api.html)
- Marius Schulz, [Advanced JavaScript Debugging with console.table()](http://blog.mariusschulz.com/2013/11/13/advanced-javascript-debugging-with-consoletable)
- Google Developer, [Command Line API Reference](https://developers.google.com/chrome-developer-tools/docs/commandline-api)