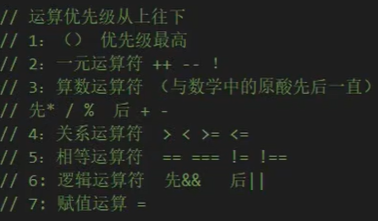
JS

一、简介

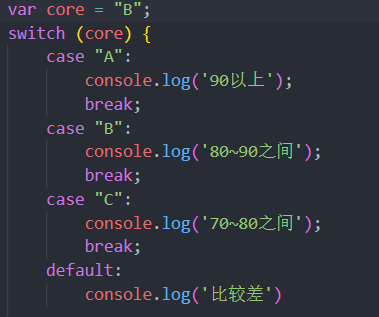
js是一种弱类型语言，即变量的数据类型存在自动转换。

1. 运算优先级



1. 判断与循环语句

Switch：





## window.getComputedStyle() 方法的使用



## 一、getComputedStyle() 用法

document.defaultView.getComputedStyle(element,pseudo-element); 或者

window.getComputedStyle(element,pseudo-element);

首先是有两个参数，元素和伪类。第二个参数不是必须的，当不查询伪类元素的时候可以忽略或者传入 null。

使用示例：

let my\_div = document.getElementById("myDiv");let style = window.getComputedStyle(my\_div, null);

关于 defaultView 引用一下 MDN 对于 defaultView 的描述:

**defaultView**

在许多在线的演示代码中, getComputedStyle 是通过 document.defaultView 对象来调用的。 大部分情况下，这是不需要的， 因为可以直接通过 window 对象调用。但有一种情况，你必需要使用 defaultView, 那是在 Firefox 3.6 上访问子框架内的样式 (iframe)。

而且除了在 IE8 浏览器中 **document.defaultView === window** 返回的是 false 外，其他的浏览器（包括 IE9 ）返回的都是 true。所以后面直接使用 window 就好，不用在输入那么长的代码了。

## 二、返回值

**getComputedStyle** 返回的对象是 **CSSStyleDeclaration** 类型的对象。取数据的时候可以直接按照属性的取法去取数据，例如 **style.backgroundColor**。需要注意的是，返回的对象的键名是 **css** 的驼峰式写法，**background-color -> backgroundColor**。

需要注意的是 **float** 属性，根据 《JavaScript 高级程序》所描述的情况 ，float 是 JS 的保留关键字。根据 DOM2 级的规范，取元素的 float 的时候应该使用 cssFloat。其实 chrome 和 Firefox 是支持 float 属性的，也就是说可以直接使用。

var float\_property = window.getComputedStyle.style; // chrome 和 Firefox支持

而在任何版本的 IE 中都不能这样使用，并且在 IE 8 中仅支持 **styleFloat** ，这个下面的兼容性问题中谈到。

## 三、和 style 的异同

**getComputedStyle** 和 **element.style** 的相同点就是二者返回的都是 CSSStyleDeclaration 对象，取相应属性值得时候都是采用的 CSS 驼峰式写法，均需要注意 float 属性。

**而不同点就是：**

* element.style 读取的只是元素的**内联样式**，即写在元素的 style 属性上的样式；而 getComputedStyle 读取的样式是最终样式，包括了**内联样式**、**嵌入样式**和**外部样式**。
* element.style 既支持读也支持写，我们通过 element.style 即可改写元素的样式。而 getComputedStyle 仅支持读并不支持写入。我们可以通过使用 getComputedStyle 读取样式，通过 element.style 修改样式

**我们可以通过使用 getComputedStyle 读取样式，通过 element.style 修改样式。**

## 四、兼容性

关于 **getComputedStyle** 的兼容性问题，在 Chrome 和 Firefox 是支持该属性的，同时 IE 9 10 11 也是支持相同的特性的，IE 8并不支持这个特性。 IE 8 支持的是 element.currentStyle 这个属性，这个属性返回的值和 **getComputedStyle** 的返回基本一致，只是在 float 的支持上，IE 8 支持的是 styleFloat,这点需要注意。

五、

getComputedStyle和getPropertyValue

getComputedStyle() 方法用于获取指定元素的 CSS 样式。

window.getComputedStyle(element, pseudoElement)

element: 必需，要获取样式的元素。

pseudoElement: 可选，伪类元素，当不查询伪类元素的时候可以忽略或者传入 null。

getPropertyValue() 方法返回指定的 CSS 属性的值。

object.getPropertyValue(propertyname)

propertyname 必需。一个字符串，表示要检测的属性名。

其中，IE8用obj.currentStyle[样式名]

## 此外，Window getComputedStyle() 方法和 style 的异同

用document.getElementById('element').style.xxx可以获取element的xxx样式信息，可是它获取的只能是DOM元素内嵌样式style属性里的样式规则,对于使用外部样式表或内部样式设置的元素，就获取不到样式信息了。

getComputedStyle 和 element.style 的相同点就是:

二者返回的都是 CSSStyleDeclaration 对象，取相应属性值得时候都是采用的 CSS 驼峰式写法，均需要注意 float 属性。

而不同点就是：

element.style 读取的只是元素的内联样式，即写在元素的 style 属性上的样式；而 getComputedStyle 读取的样式是最终样式，包括了内联样式、嵌入样式和外部样式。

element.style 既支持读也支持写，我们通过 element.style 即可改写元素的样式。而 getComputedStyle 仅支持读并不支持写入。我们可以通过使用 getComputedStyle 读取样式，通过 element.style 修改样式

我们可以通过使用 getComputedStyle 读取样式，通过 element.style 修改样式。

## **六、**彻底理解 Node.js 中的回调(Callback)函数

## 究竟什么是回调函数(Callback)，网上有许许多多的文章，大部分看得人云里雾外，这些文章大概分成两类，第一类堆砌了太多的术语，基本上不明白术语就没法看，另一类反过来，不讲术语，完全是举一些脱离编程的生活化例子来类比，看的人更加晕头转向。

作为JS的核心，回调函数和异步执行是紧密相关的，不跨过这个门槛，很多回调代码能把人看晕！

*引用stack overflow 上大神的描述 其实callback 很简单也很纯粹:*

***A "callback" is any function that is called by another function which takes the first function as a parameter. （在一个函数中调用另外一个函数就是callback）***

以下是一个最简单的例子：

function a() {

return 1}

function b(aa) {

return 2 + aa}

//调用：var c=0

c = b(a()) //A是个函数，但它又作为一个参数在B函数中被调用

console.log(c) //结果显示3

以上例子极易理解，下面再引入另一个概念：**异步**

看以下代码：

var a = 0

function bb(x) {

console.log(x)}

function timer(time) {

setTimeout(function () {

a=6

}, time);}

//调用:

console.log(a)

timer(3000)

bb(a)

以上代码很简单，我们需要的逻辑是，全局变量a初值为0，然后过3秒后，让它为6，然后再打印出来，看上去，上面的代码没有问题，理论上符合我们的逻辑需求，但却发现结果是这样：

00

咋回事？

因为JS是一种异步执行语言，尽管timer函数内让a=6了，但是JS不会死等时间结束再跳出函数，而是马上就会执行下一步语句(即调用bb函数)，但这时候3秒钟根本就没结束，a还没有被重新赋值，所以打印出来还是为0。

用回调函数可以解决这个问题:

var a = 0

function bb(x) {

console.log(x)}

function timer(time, callback) {

setTimeout(function () {

a = 6

callback(a);

}, time);}

//调用:

console.log(a)

timer(3000,bb)

这次，在timer函数中添加了一个关键字callback，意思就是说此处不是一个普通的参数，而是一个函数名，打起精神，关键的地方来了：

*一般而言，函数的形参是指由外往内向函数体传递变量的入口，但此处加了callback后则完全相反，它是指函数体在完成某种使命后调用外部函数的出口！这时候应该明白什么叫"回调"了吧，也就是回头调用外部函数的意思。*

在本例中，当3秒钟到了后，首先a=6，然后通过关键字callback(a)调用了函数bb(x)，结果显示：

06

这个逻辑，符合我们的需求。

在写法上，也可以不需要定义函数bb, 直接在调用timer的时候写成function形式,把调用部分改成这样也可以，效果完全一样:

console.log(a)

timer(3000, function (x) {

console.log(x)})

这种写法函数名都不需要了（术语称为"匿名函数"），在nodejs代码中更为常见也更好理解，翻译成自然语言就是：定时3秒，完成后再回头调用function(x)里面的内容。

nodejs编程中大量使用了异步编程技术，这是为了高效使用硬件，同时也可以不造成同步阻塞。其实nodejs在底层还是通过多线程技术实现的异步操作，但普通用户并不需要深究它的实现方法，我们只要做好我们的异步处理即可。

1. Array对象

## 数组属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| [constructor](https://www.runoob.com/jsref/jsref-constructor-array.html) | 返回创建数组对象的原型函数。 |
| [length](https://www.runoob.com/jsref/jsref-length-array.html) | 设置或返回数组元素的个数。 |
| [prototype](https://www.runoob.com/jsref/jsref-prototype-array.html) | 允许你向数组对象添加属性或方法。 |

## Array 对象方法

|  |  |
| --- | --- |
| **方法** | **描述** |
| [concat()](https://www.runoob.com/jsref/jsref-concat-array.html) | 连接两个或更多的数组，并返回结果。 |
| [copyWithin()](https://www.runoob.com/jsref/jsref-copywithin.html) | 从数组的指定位置拷贝元素到数组的另一个指定位置中。 |
| [entries()](https://www.runoob.com/jsref/jsref-entries.html) | 返回数组的可迭代对象。 |
| [every()](https://www.runoob.com/jsref/jsref-every.html) | 检测数值元素的每个元素是否都符合条件。 |
| [fill()](https://www.runoob.com/jsref/jsref-fill.html) | 使用一个固定值来填充数组。 |
| [filter()](https://www.runoob.com/jsref/jsref-filter.html) | 检测数值元素，并返回符合条件所有元素的数组。 |
| [find()](https://www.runoob.com/jsref/jsref-find.html) | 返回符合传入测试（函数）条件的数组元素。 |
|  |  |
| [findIndex()](https://www.runoob.com/jsref/jsref-findindex.html) | 返回符合传入测试（函数）条件的数组元素索引。 |
|  |  |
| [forEach()](https://www.runoob.com/jsref/jsref-foreach.html) | 数组每个元素都执行一次回调函数。 |
| [from()](https://www.runoob.com/jsref/jsref-from.html) | 通过给定的对象中创建一个数组。 |
|  |  |
| [includes()](https://www.runoob.com/jsref/jsref-includes.html) | 判断一个数组是否包含一个指定的值。 |
| [indexOf()](https://www.runoob.com/jsref/jsref-indexof-array.html) | 搜索数组中的元素，并返回它所在的位置。 |
| [isArray()](https://www.runoob.com/jsref/jsref-isarray.html) | 判断对象是否为数组。 |
| [join()](https://www.runoob.com/jsref/jsref-join.html) | 把数组的所有元素放入一个字符串。 |
| [keys()](https://www.runoob.com/jsref/jsref-keys.html) | 返回数组的可迭代对象，包含原始数组的键(key)。 |
| [lastIndexOf()](https://www.runoob.com/jsref/jsref-lastindexof-array.html) | 搜索数组中的元素，并返回它最后出现的位置。 |
| [map()](https://www.runoob.com/jsref/jsref-map.html) | 通过指定函数处理数组的每个元素，并返回处理后的数组。 |
| [pop()](https://www.runoob.com/jsref/jsref-pop.html) | 删除数组的最后一个元素并返回删除的元素。 |
| [push()](https://www.runoob.com/jsref/jsref-push.html) | 向数组的末尾添加一个或更多元素，并返回新的长度。 |
|  |  |
| [reduce()](https://www.runoob.com/jsref/jsref-reduce.html) | 将数组元素计算为一个值（从左到右）。 |
|  |  |
| [reduceRight()](https://www.runoob.com/jsref/jsref-reduceright.html) | 将数组元素计算为一个值（从右到左）。 |
| [reverse()](https://www.runoob.com/jsref/jsref-reverse.html) | 反转数组的元素顺序。 |
| [shift()](https://www.runoob.com/jsref/jsref-shift.html) | 删除并返回数组的第一个元素。 |
| [slice()](https://www.runoob.com/jsref/jsref-slice-array.html) | 选取数组的的一部分，并返回一个新数组。 |
| [some()](https://www.runoob.com/jsref/jsref-some.html) | 检测数组元素中是否有元素符合指定条件。 |
| [sort()](https://www.runoob.com/jsref/jsref-sort.html) | 对数组的元素进行排序。 |
| [splice()](https://www.runoob.com/jsref/jsref-splice.html) | 从数组中添加或删除元素。 |
| [toString()](https://www.runoob.com/jsref/jsref-tostring-array.html) | 把数组转换为字符串，并返回结果。 |
| [unshift()](https://www.runoob.com/jsref/jsref-unshift.html) | 向数组的开头添加一个或更多元素，并返回新的长度。 |
| [valueOf()](https://www.runoob.com/jsref/jsref-valueof-array.html) | 返回数组对象的原始值。 |

1. Boolean对象