《JavaScript 闯关记》之单体内置对象

javascript 劼哥stone 11月1日发布

ECMA-262 对内置对象的定义是「由 JavaScript 实现提供的、不依赖于宿主环境的对象，这些对象在 JavaScript 程序执行之前就已经存在了」。意思就是说，开发人员不必显式地实例化内置对象，因为它们已经实例化了。前面我们已经介绍了大多数内置对象，例如 `Object`、`Array` 和 `String`。ECMA-262 还定义了两个单体内置对象：`Global` 和 `Math`。

`Global` 对象

`Global` 对象可以说是 JavaScript 中最特别的一个对象了，因为不管你从什么角度上看，这个对象都是不存在的。`Global` 对象在某种意义上是作为一个终极的「兜底儿对象」来定义的。换句话说，不属于任何其他对象的属性和方法，最终都是它的属性和方法。所有在全局作用域中定义的属性和函数，都是 `Global` 对象的属性。本书前面介绍过的那些函数，诸如 `isNaN()`、`isFinite()`、`parseInt()` 以及 `parseFloat()`，实际上全都是 `Global` 对象的方法。除此之外，`Global` 对象还包含其他一些方法。

URI 编码方法

`Global` 对象的 `encodeURI()` 和 `encodeURIComponent()` 方法可以对 URI（Uniform Resource Identifiers，通用资源标识符）进行编码，以便发送给浏览器。有效的 URI 中不能包含某些字符，例如空格。而这两个 URI 编码方法就可以对 URI 进行编码，它们用特殊的 UTF-8 编码替换所有无效的字符，从而让浏览器能够接受和理解。

其中，`encodeURI()` 主要用于整个 URI，而 `encodeURIComponent()` 主要用于对 URI 中的某一段进行编码。它们的主要区别在于，`encodeURI()` 不会对本身属于 URI 的特殊字符进行编码，例如冒号、正斜杠、问号和井字号；而 `encodeURIComponent()` 则会对它发现的任何非标准字符进行编码。来看下面的例子。

```
var uri = "http://shijiajie.com/illegal value.htm#start";

console.log(encodeURI(uri));
// "http://shijiajie.com/illegal%20value.htm#start"

console.log(encodeURIComponent(uri));
// "http%3A%2F%2Fshijiajie.com%2Fillegal%20value.htm%23start"
```

使用 `encodeURI()` 编码后的结果是除了空格之外的其他字符都原封不动，只有空格被替换成了 `%20`。而 `encodeURIComponent()` 方法则会使用对应的编码替换所有非字母数字字符。这也正是可以对整个 URI 使用 `encodeURI()`，而只能对附加在现有 URI 后面的字符串使用 `encodeURIComponent()` 的原因所在。

一般来说，我们使用 `encodeURIComponent()` 方法的时候要比使用 `encodeURI()` 更多，因为在实践中更常见的是对查询字符串参数而不是对基础 URI 进行编码。

与 `encodeURI()` 和 `encodeURIComponent()` 方法对应的两个方法分别是 `decodeURI()` 和 `decodeURIComponent()`。其中，`decodeURI()` 只能对使用 `encodeURI()` 替换的字符进行解码。例如，它可将 `%20` 替换成一个空格，但不会对 `%23` 作任何处理，因为 `%23` 表示井字号 `#`，而井字号不是使用 `encodeURI()` 替换的。同样地，`decodeURIComponent()` 能够解码使用 `encodeURIComponent()` 编码的所有字符，即它可以解码任何特殊字符的编码。来看下面的例子：

```
var uri = "http%3A%2F%2Fshijiajie.com%2Fillegal%20value.htm%23start";

console.log(decodeURI(uri));
// http%3A%2F%2Fshijiajie.com%2Fillegal value.htm%23start

console.log(decodeURIComponent(uri));
// http://shijiajie.com/illegal value.htm#start
```

这里，变量 `uri` 包含着一个由 `encodeURIComponent()` 编码的字符串。在第一次调用 `decodeURI()` 输出的结果中，只有 `%20` 被替换成了空格。而在第二次调用 `decodeURIComponent()` 输出的结果中，所有特殊字符的编码都被替换成了原来的字符，得到了一个未经转义的字符串（但这个字符串并不是一个有效的 URI）。

`eval()` 方法

`eval()` 方法就像是一个完整的 JavaScript 解析器，它只接受一个参数，即要执行的 JavaScript 字符串。看下面的例子：

```
eval("console.log('hi')");
```

这行代码的作用等价于下面这行代码：

```
console.log("hi");
```

当解析器发现代码中调用 `eval()` 方法时，它会将传入的参数当作实际的 JavaScript 语句来解析，然后把执行结果插入到原位置。通过 `eval()` 执行的代码被认为是包含该次调用的执行环境的一部分，因此被执行的代码具有与该执行环境相同的作用域链。这意味着通过 `eval()` 执行的代码可以引用在包含环境中定义的变量，举个例子：

```
var msg = "hello world";
eval("console.log(msg)");    // "hello world"
```

可见，变量 `msg` 是在 `eval()` 调用的环境之外定义的，但其中调用的 `console.log()` 仍然能够显示 `"hello world"`。这是因为上面第二行代码最终被替换成了一行真正的代码。同样地，我们也可以在 `eval()` 调用中定义一个函数，然后再在该调用的外部代码中引用这个函数：

```
eval("function sayHi() { console.log('hi'); }");
sayHi();    // "hi"
```

显然，函数 `sayHi()` 是在 `eval()` 内部定义的。但由于对 `eval()` 的调用最终会被替换成定义函数的实际代码，因此可以在下一行调用 `sayHi()`。对于变量也一样：

```
eval("var msg = 'hello world';");
console.log(msg);     // "hello world"
```

在 `eval()` 中创建的任何变量或函数都不会被提升，因为在解析代码的时候，它们被包含在一个字符串中；它们只在 `eval()` 执行的时候创建。

严格模式下，在外部访问不到 `eval()` 中创建的任何变量或函数，因此前面两个例子都会导致错误。同样，在严格模式下，为 `eval` 赋值也会导致错误：

```
"use strict";
eval = "hi";   // causes error
```

能够解释代码字符串的能力非常强大，但也非常危险。因此在使用 `eval()` 时必须极为谨慎，特别是在用它执行用户输入数据的情况下。否则，可能会有恶意用户输入威胁你的站点或应用程序安全的代码（即所谓的代码注入）。

`Global` 对象的属性

`Global` 对象还包含一些属性，其中一部分属性已经在本书前面介绍过了。例如，特殊的值 `undefined`、`NaN` 以及 `Infinity` 都是 `Global` 对象的属性。此外，所有原生引用类型的构造函数，像 `Object` 和 `Function`，也都是 `Global` 对象的属性。下表列出了 `Global` 对象的所有属性。

属性	说明	属性	说明
undefined	特殊值undefined	Date	构造函数Date
NaN	特殊值NaN	RegExp	构造函数RegExp
Infinity	特殊值Infinity	Error	构造函数Error
Object	构造函数Object	EvalError	构造函数EvalError
Array	构造函数Array	RangeError	构造函数RangeError
Function	构造函数Function	ReferenceError	构造函数ReferenceError
Boolean	构造函数Boolean	SyntaxError	构造函数SyntaxError

String	构造函数String	TypeError	构造函数TypeError
Number	构造函数Number	URIError	构造函数URIError

ECMAScript 5 明确禁止给 `undefined`、`NaN` 和 `Infinity` 赋值，这样做即使在非严格模式下也会导致错误。

`window` 对象

JavaScript 虽然没有指出如何直接访问 `Global` 对象，但 Web 浏览器都是将这个全局对象作为 `window` 对象的一部分加以实现的。因此，在全局作用域中声明的所有变量和函数，就都成为了 `window` 对象的属性。来看下面的例子。

```
var color = "red";

function sayColor(){
    console.log(window.color);
}

window.sayColor();  // "red"
```

JavaScript 中的 `window` 对象除了扮演规定的 `Global` 对象的角色外，还承担了很多别的任务。

`Math` 对象

JavaScript 还为保存数学公式和信息提供了一个公共位置，即 `Math` 对象。与我们在 `JavaScript` 直接编写的计算功能相比，`Math` 对象提供的计算功能执行起来要快得多。`Math` 对象中还提供了辅助完成这些计算的属性和方法。

`Math` 对象的属性

`Math` 对象包含的属性大都是数学计算中可能会用到的一些特殊值。下表列出了这些属性。

属性	说明
Math.E	自然对数的底数，即常量e的值
Math.LN10	10的自然对数
Math.LN2	2的自然对数
Math.LOG2E	以2为底e的对数
Math.LOG10E	以10为底e的对数
Math.PI	π的值
Math.SQRT1_2	1/2的平方根（即2的平方根的倒数）
Math.SQRT2	2的平方根

`min()` 和 `max()` 方法

`Math` 对象还包含许多方法，用于辅助完成简单和复杂的数学计算。其中，`min()` 和 `max()` 方法用于确定一组数值中的最小值和最大值。这两个方法都可以接收任意多个数值参数，如下面的例子所示。

```
var max = Math.max(3, 54, 32, 16);
console.log(max);    // 54

var min = Math.min(3, 54, 32, 16);
console.log(min);    // 3
```

要找到数组中的最大或最小值，可以像下面这样使用 `apply()` 方法。

```
var values = [1, 2, 3, 4, 5, 6, 7, 8];
var max = Math.max.apply(Math, values);
console.log(max);   // 8
```

这个技巧的关键是把 `Math` 对象作为 `apply()` 的第一个参数，从而正确地设置 `this` 值。然后，可以将任何数组作为第二个参数。

舍入方法

下面来介绍将小数值舍入为整数的几个方法：`Math.ceil()`、`Math.floor()` 和 `Math.round()`。这三个方法分别遵循下列舍入规则：

- `Math.ceil()` 执行向上舍入，即它总是将数值向上舍入为最接近的整数；
- `Math.floor()` 执行向下舍入，即它总是将数值向下舍入为最接近的整数；
- `Math.round()` 执行标准舍入，即它总是将数值四舍五入为最接近的整数。

下面是使用这些方法的示例：

```
console.log(Math.ceil(25.9));     // 26
console.log(Math.ceil(25.5));     // 26
console.log(Math.ceil(25.1));     // 26

console.log(Math.round(25.9));    // 26
console.log(Math.round(25.5));    // 26
console.log(Math.round(25.1));    // 25

console.log(Math.floor(25.9));    // 25
console.log(Math.floor(25.5));    // 25
console.log(Math.floor(25.1));    // 25
```

`random()` 方法

`Math.random()` 方法返回介于0和1之间一个随机数，不包括0和1。对于某些站点来说，这个方法非常实用，因为可以利用它来随机显示一些名人名言和新闻事件。套用下面的公式，就可以利用 `Math.random()` 从某个整数范围内随机选择一个值。

```
值 = Math.floor(Math.random() * 可能值的总数 + 第一个可能的值)
```

公式中用到了 `Math.floor()` 方法，这是因为 `Math.random()` 总返回一个小数值。而用这个小数值乘以一个整数，然后再加上一个整数，最终结果仍然还是一个小数。举例来说，如果你想选择一个1到10之间的数值，可以像下面这样编写代码：

```
var num = Math.floor(Math.random() * 10 + 1);
```

总共有10个可能的值（1到10），而第一个可能的值是1。而如果想要选择一个介于2到10之间的值，就应该将上面的代码改成这样：

```
var num = Math.floor(Math.random() * 9 + 2);
```

从2数到10要数9个数，因此可能值的总数就是9，而第一个可能的值就是2。多数情况下，其实都可以通过一个函数来计算可能值的总数和第一个可能的值，例如：

```
function selectFrom(lowerValue, upperValue) {
    var choices = upperValue - lowerValue + 1;
    return Math.floor(Math.random() * choices + lowerValue);
}

var num = selectFrom(2, 10);
console.log(num);   // 介于2和10之间（包括2和10）的一个数值
```

函数 `selectFrom()` 接受两个参数：应该返回的最小值和最大值。而用最大值减最小值再加1得到了可能值的总数，然后它又把这些数值套用到了前面的公式中。这样，通过调用 `selectFrom(2,10)` 就可以得到一个介于2和10之间（包括2和10）的数值了。利用这个函数，可以方便地从数组中随机取出一项，例如：

```
var colors = ["red", "green", "blue", "yellow", "black", "purple", "brown"];
var color = colors[selectFrom(0, colors.length-1)];
console.log(color);  // 可能是数组中包含的任何一个字符串
```

其他方法

`Math` 对象中还包含其他一些与完成各种简单或复杂计算有关的方法，但详细讨论其中每一个方法的细节及适用情形超出了本书的范围。下面我们就给出一个表格，其中列出了这些没有介绍到的 `Math` 对象的方法。

方法	说明
Math.abs(num)	返回num的绝对值
Math.asin(x)	返回x的反正弦值
Math.exp(num)	返回Math.E的num次幂
Math.atan(x)	返回x的反正切值
Math.log(num)	返回num的自然对数
Math.atan2(y,x)	返回y/x的反正切值
Math.pow(num,power)	返回num的power次幂
Math.cos(x)	返回x的余弦值
Math.sqrt(num)	返回num的平方根
Math.sin(x)	返回x的正弦值
Math.acos(x)	返回x的反余弦值
Math.tan(x)	返回x的正切值

虽然 ECMA-262 规定了这些方法，但不同实现可能会对这些方法采用不同的算法。毕竟，计算某个值的正弦、余弦和正切的方式多种多样。也正因为如此，这些方法在不同的实现中可能会有不同的精度。

关卡

```
// 如何高效产生m个n范围内的不重复随机数（m<=n）
var getRandomNumber = function(n, m){
    // 待实现方法体
}
console.log(getRandomNumber(20, 3));  // 8,4,19
```

更多

> 关注微信公众号「劼哥舍」回复「答案」，获取关卡详解。
> 关注 https://github.com/stone0090/javascript-lessons，获取最新动态。

讨论区

使用评论询问更多信息或提出修改意见，请不要在评论里回答问题

提交评论

评论支持部分 Markdown 语法： `**bold**` `_italic_` `[link](http://example.com)` `> 引用` `` `code` `` `- 列表` 。
同时，被你 @ 的用户也会收到通知

本文隶属于专栏

劼哥舍

欢迎来到「劼哥舍」，您非要念成「劫个色」也行。这里坚持原创，分享随笔和学习心得，主要涉及 前端 / .NET / Java 等方面的内容。欢迎交流，欢迎提问，欢迎转载，但需注明出处。

劼哥stone
作者 关注作者

关注专栏

系列文章

《JavaScript 闯关记》之对象 6 收藏， 164 浏览
《JavaScript 闯关记》之数组 6 收藏， 232 浏览
《JavaScript 闯关记》之函数 3 收藏， 161 浏览
《JavaScript 闯关记》之正则表达式 30 收藏， 809 浏览
《JavaScript 闯关记》之基本包装类型 18 收藏， 862 浏览
《JavaScript 闯关记》之 BOM 14 收藏， 525 浏览
《JavaScript 闯关记》之 DOM（上） 3 收藏， 232 浏览

相关收藏夹

换一组

基础知识
9 个条目 | 0 人关注

前端技术
4 个条目 | 1 人关注

webpack
5 个条目 | 3 人关注

分享扩散：