Global（全局）对象可以说是 ECMAScript 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。ECMAScript 中的 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。事实上，没有全局变量或全局函数；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。

1. URI 编码方法

encodeURI() 和 encodeURIComponent()

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（例如，http://www.wrox.com/illegal value.htm），而 encodeURIComponent()主要用于对 URI 中的某一段（例如前面 URI 中的 illegal value.htm）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://www.wrox.com/illegal value.htm#start";

//"http://www.wrox.com/illegal%20value.htm#start"
alert(encodeURI(uri));

//"http%3A%2F%2Fwww.wrox.com%2Fillegal%20value.htm%23start"
alert(encodeURIComponent(uri));
```

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。

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。

decodeURI() decodeURIComponent()

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

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

//http%3A%2F%2Fwww.wrox.com%2Fillegal value.htm%23start
alert(decodeURI(uri));

//http://www.wrox.com/illegal value.htm#start
alert(decodeURIComponent(uri));
```

个有效的 URI）。

URI 方法 encodeURI()、encodeURIComponent()、decodeURI()和 decodeURIComponent()用于替代已经被 ECMA-262 第 3 版废弃的 escape()和 unescape()方法。URI 方法能够编码所有 Unicode 字符，而原来的方法只能正确地编码 ASCII 字符。因此在开发实践中，特别是在产品级的代码中，一定要使用 URI 方法，不要使用 escape()和 unescape()方法。

2. eval()方法

eval() 最牛逼的js方法

现在，我们介绍最后一个——大概也是整个 ECMAScript 语言中最强大的一个方法：eval()。eval()方法就像是一个完整的 ECMAScript 解析器，它只接受一个参数，即要执行的 ECMAScript（或 JavaScript）字符串。看下面的例子：

```
eval("alert('hi')");
```

这行代码的作用等价于下面这行代码：

```
alert("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 ECMAScript 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("alert(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 alert()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() { alert('hi'); }");
sayHi();
```

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'; ");
alert(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**）。

3. Global 对象的属性

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。

4. window 对象

ECMAScript 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(){
    alert(window.color);
}

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

JavaScript 中的 window 对象除了扮演 ECMAScript 规定的 Global 对象的角色外，还承担了很多别的任务。第 8 章在讨论浏览器对象模型时将详细介绍 window 对象。

另一种取得 Global 对象的方法是使用以下代码：

```
var global = function(){
    return this;
}();
```

以上代码创建了一个立即调用的函数表达式，返回 this 的值。如前所述，在没有给函数明确指定 this 值的情况下（无论是通过将函数添加为对象的方法，还是通过调用 call()或 apply()），this 值等于 Global 对象。而像这样通过简单地返回 this 来取得 Global 对象，在任何执行环境下都是可行的。第 7 章将深入讨论函数表达式。