**JavaScript学习笔记**

何清 2016-10-12

概览

JavaScript 是一种面向对象的动态语言，它包含类型、运算符、标准内置（ built-in）对象和方法。它的语法来源于 Java 和 C，所以这两种语言的许多语法特性同样适用于 JavaScript。需要注意的一个主要区别是 **JavaScript 不支持类**，类这一概念在 JavaScript 通过对象原型（object prototype）得到延续（有关 ES6 类的内容参考这里Classes）。另一个主要区别是 **JavaScript 中的函数也是对象**，JavaScript 允许函数在包含可执行代码的同时，能像其他对象一样被传递。

1. 相等比较== 与 ===

由两个“=（等号）”组成的相等运算符有类型自适应的功能，具体例子如下：

123 == "123" // true

1 == true; // true

如果在比较前不需要自动类型转换，应该使用由三个“=（等号）”组成的相等运算符：

1 === true; //false

123 === "123"; // false

JavaScript 还支持 != 和 !== 两种不等运算符，具体区别与两种相等运算符的区别类似。

1. 多重分支switch

在需要多重分支时可以使用 基于一个**数字或字符串**的switch 语句：

default 语句是可选的。switch 和 case 都可以使用需要运算才能得到结果的表达式；在 switch 的表达式和 case 的表达式是使用 **=== 严格相等运算符进行比较的**。

1. 创建一个空对象

有两种简单方法可以创建一个空对象：

var obj = new Object();

和：

var obj = {};

这两种方法在语义上是相同的。第二种更方便的方法叫作“对象字面量（object literal）”法。这种也是 JSON 格式的核心语法，一般我们优先选择第二种方法。

1. JavaScript数组

JavaScript 中的数组是一种特殊的对象。它的工作原理与普通对象类似（以数字为属性名，但**只能通过[] 来访问**），但数组还有一个特殊的属性——length（长度）属性。这个属性的值通常比数组最大索引大 1。

创建数组的传统方法是：

var a = new Array();

a[0] = "dog";

a[1] = "cat";

a[2] = "hen";

a.length; // 3

使用数组字面量（array literal）法更加方便：

var a = ["dog", "cat", "hen"];

a.length; // 3

注意，Array.length **并不总是等于数组中元素的个数**，如下所示：

var a = ["dog", "cat", "hen"];

a[100] = "fox";

a.length; // 101

记住：**数组的长度是比数组最大索引值多一的数**。

如果试图访问一个不存在的数组索引，会得到 undefined：

typeof(a[90]); // undefined

可以通过如下方式遍历一个数组：

for (var i = 0; i < a.length; i++) {

// Do something with a[i]

}

遍历数组的另一种方法是使用 **for...in** 循环。注意，如果有人向 Array.prototype 添加了新的属性，使用这样的循环这些属性也同样会被遍历。所以并不推荐这种方法：

for (var i in a) {

// Do something with a[i]

}

ECMAScript 5 增加了遍历数组的另一个方法 **forEach()**：

["dog", "cat", "hen"].forEach(function(currentValue, index, array) {

// Do something with currentValue or array[index]

});

如果想在数组后**追加元素**，只需要：

a.push(item);

Array（数组）类自带了许多方法。查看<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array>。

| **方法名称** | **描述** |
| --- | --- |
| a.toString() | 返回一个包含数组中所有元素的字符串，每个元素通过逗号分隔。 |
| a.toLocaleString() | 根据宿主环境的区域设置，返回一个包含数组中所有元素的字符串，每个元素通过逗号分隔。 |
| a.concat(item1[, item2[, ...[, itemN]]]) | 返回一个数组，这个数组包含原先 a 和 item1、item2、……、itemN 中的所有元素。 |
| a.join(sep) | 返回一个包含数组中所有元素的字符串，每个元素通过指定的 sep 分隔。 |
| a.pop() | 删除并返回数组中的最后一个元素。 |
| a.push(item1, ..., itemN) | 将 item1、item2、……、itemN 追加至数组 a。 |
| a.reverse() | 数组逆序（会更改原数组 a）。 |
| a.shift() | 删除并返回数组中第一个元素。 |
| a.slice(start, end) | 返回子数组，以 a[start] 开头，以 a[end] 前一个元素结尾。 |
| a.sort([cmpfn]) | 依据 cmpfn 返回的结果进行排序，如果未指定比较函数则按字符顺序比较（即使元素是数字）。 |
| a.splice(start, delcount[, item1[, ...[, itemN]]]) | 从 start 开始，删除 delcount 个元素，然后插入所有的item。 |
| a.unshift([item]) | 将 item 插入数组头部，返回数组新长度（考虑undefined）。 |

1. 如何创建对象

在创建新的对象或者类时，方法通常应该关联于对象的原型，而不是定义到对象的构造器中。原因是这将导致每次构造器被调用，方法都会被重新赋值一次（也就是说，为每一个对象的创建）。

考虑以下虽然不切实际但却说明问题的示例：

function MyObject(name, message) {

this.name = name.toString();

this.message = message.toString();

this.getName = function() {

return this.name;

};

this.getMessage = function() {

return this.message;

};

}

上面的代码并未利用到闭包的益处，因此，应该修改为如下常规形式：

function MyObject(name, message) {

this.name = name.toString();

this.message = message.toString();

}

MyObject.prototype = {

getName: function() {

return this.name;

},

getMessage: function() {

return this.message;

}

};

或者改成：

function MyObject(name, message) {

this.name = name.toString();

this.message = message.toString();

}

MyObject.prototype.getName = function() {

return this.name;

};

MyObject.prototype.getMessage = function() {

return this.message;

};

在前面的两个示例中，继承的原型可以为所有对象共享，且不必在每一次创建对象时定义方法。