6.1.1 属性类型

attribute property

ECMA-262 第 5 版在定义只有内部才用的特性（attribute）时，描述了属性（property）的各种特征。ECMA-262 定义这些特性是为了实现 JavaScript 引擎用的，因此在 JavaScript 中不能直接访问它们。为了表示特性是内部值，该规范把它们放在了两对儿方括号中，例如[[Enumerable]]。尽管 ECMA-262 第 3 版的定义有些不同，但本书只参考第 5 版的描述。

ECMAScript 中有两种属性：数据属性和访问器属性。

1. 数据属性

数据属性包含一个数据值的位置。在这个位置可以读取和写入值。数据属性有 4 个描述其行为的特性。

- [[Configurable]]：表示能否通过 delete 删除属性从而重新定义属性，能否修改属性的特性，或者能否把属性修改为访问器属性。像前面例子中那样直接在对象上定义的属性，它们的这个特性默认值为 true。
- [[Enumerable]]：表示能否通过 for-in 循环返回属性。像前面例子中那样直接在对象上定义的属性，它们的这个特性默认值为 true。
- [[Writable]]：表示能否修改属性的值。像前面例子中那样直接在对象上定义的属性，它们的这个特性默认值为 true。
- [[Value]]：包含这个属性的数据值。读取属性值的时候，从这个位置读；写入属性值的时候，把新值保存在这个位置。这个特性的默认值为 undefined。

对于像前面例子中那样直接在对象上定义的属性，它们的[[Configurable]]、[[Enumerable]]和[[Writable]]特性都被设置为 true，而[[Value]]特性被设置为指定的值。例如：

```
var person = {
    name: "Nicholas"
};
```

这里创建了一个名为 name 的属性，为它指定的值是"Nicholas"。也就是说，[[Value]]特性将被设置为"Nicholas"，而对这个值的任何修改都将反映在这个位置。

要修改属性默认的特性，必须使用 ECMAScript 5 的 Object.defineProperty()方法。这个方法接收三个参数：属性所在的对象、属性的名字和一个描述符对象。其中，描述符（descriptor）对象的属性必须是： configurable 、 enumerable 、 writable 和 value 。设置其中的一或多个值，可以修改对应的特性值。例如：

```
var person = {};
Object.defineProperty(person, "name", {
    writable: false,
    value: "Nicholas"
});

alert(person.name);    //"Nicholas"
person.name = "Greg";
alert(person.name);    //"Nicholas"
```

这个例子创建了一个名为 name 的属性，它的值"Nicholas"是只读的。这个属性的值是不可修改的，如果尝试为它指定新值，则在非严格模式下，赋值操作将被忽略；在严格模式下，赋值操作将会导致抛出错误。

类似的规则也适用于不可配置的属性。例如：

```
var person = {};
Object.defineProperty(person, "name", {
    configurable: false,
    value: "Nicholas"
});

alert(person.name);    //"Nicholas"
delete person.name;
alert(person.name);    //"Nicholas"
```

DataPropertiesExample02.htm

把 configurable 设置为 false，表示不能从对象中删除属性。如果对这个属性调用 delete，则在非严格模式下什么也不会发生，而在严格模式下会导致错误。而且，一旦把属性定义为不可配置的，就不能再把它变回可配置了。此时，再调用 Object.defineProperty()方法修改除 writable 之外的特性，都会导致错误：

```
var person = {};
Object.defineProperty(person, "name", {
    configurable: false,
    value: "Nicholas"
});

//抛出错误
Object.defineProperty(person, "name", {
    configurable: true,
    value: "Nicholas"
});
```

DataPropertiesExample03.htm

也就是说，可以多次调用 Object.defineProperty()方法修改同一个属性，但在把 configurable 特性设置为 false 之后就会有限制了。

在调用 Object.defineProperty()方法时，如果不指定，configurable、enumerable 和 writable 特性的默认值都是 false。多数情况下，可能都没有必要利用 Object.defineProperty()方法提供的这些高级功能。不过，理解这些概念对理解 JavaScript 对象却非常有用。

IE8 是第一个实现 Object.defineProperty()方法的浏览器版本。然而，这个版本的实现存在诸多限制：只能在 DOM 对象上使用这个方法，而且只能创建访问器属性。由于实现不彻底，建议读者不要在 IE8 中使用 Object.defineProperty()方法。

2. 访问器属性

访问器属性不包含数据值；它们包含一对儿 getter 和 setter 函数（不过，这两个函数都不是必需的）。在读取访问器属性时，会调用 getter 函数，这个函数负责返回有效的值；在写入访问器属性时，会调用 setter 函数并传入新值，这个函数负责决定如何处理数据。访问器属性有如下 4 个特性。

- `[[Configurable]]`：表示能否通过 `delete` 删除属性从而重新定义属性，能否修改属性的特性，或者能否把属性修改为数据属性。对于直接在对象上定义的属性，这个特性的默认值为 `true`。
- `[[Enumerable]]`：表示能否通过 `for-in` 循环返回属性。对于直接在对象上定义的属性，这个特性的默认值为 `true`。
- `[[Get]]`：在读取属性时调用的函数。默认值为 `undefined`。
- `[[Set]]`：在写入属性时调用的函数。默认值为 `undefined`。

访问器属性不能直接定义，必须使用 `Object.defineProperty()` 来定义。请看下面的例子。

```
var book = {
    _year: 2004,
    edition: 1
};

Object.defineProperty(book, "year", {
    get: function(){
        return this._year;
    },
    set: function(newValue){

        if (newValue > 2004) {
            this._year = newValue;
            this.edition += newValue - 2004;
        }
    }
});

book.year = 2005;
alert(book.edition);  //2
```

AccessorPropertiesExample01.htm

以上代码创建了一个 `book` 对象，并给它定义两个默认的属性：`_year` 和 `edition`。`_year` 前面的下划线是一种常用的记号，用于表示只能通过对象方法访问的属性。而访问器属性 `year` 则包含一个 getter 函数和一个 setter 函数。getter 函数返回 `_year` 的值，setter 函数通过计算来确定正确的版本。因此，把 `year` 属性修改为 2005 会导致 `_year` 变成 2005，而 `edition` 变为 2。这是使用访问器属性的常见方式，即设置一个属性的值会导致其他属性发生变化。