## 对象

### 对象的概念

**构造器与字面量**

*字面量*是一种值的表示法。通过直接写出一个值所包含的内容，来创建这个值，这种**形式**被称为字面量。

如："Hello world" 就是一个字符串字面量，1234.5678 就是一个数字字面量。

字面量与*创建实例*相对。在 JavaScript 中，每种类型的值都有一个对应的*构造器*，原则上你可以使用构造器来构造一个值，构造器写成函数调用的形式，可以接受一些值，作为创建新值的所需信息，这种创建值的形式称为*创建实例*。如：

const a = new Number(); // a = 0  
const b = new Number(10); // b = 10  
const c = new String(); // c = ""  
const d = new String(100); // c = "100"

当然，JavaScript 中提供了方便的*字面量*方式来让我们创建值，我们不需要为值一个个创建实例。

除了我们已经了解过的布尔值、数值、字符串之外，有一大类值被称为*对象*。对象是一个包含了一些值的结构。

const o = new Object();  
o.name = "John";  
o.age = 12;

现在我们创建了一个对象，并将它赋给常量 o 。我们为 o 提供了两个信息，分别是名称和年龄。这个信息被称作*属性*，我们可以通过属性来访问对象中所存放的具体的值。

const cat = new Object();  
cat.name = "Lily";  
cat.age = 3;  
alert(cat.age); // 3

我们赋予了这个“猫对象”两个属性，一个是它的名字，一个是年龄。每个属性都对应一个值。当我们需要改变或得到某个属性的值时，我们只需要写下 <对象名>.<属性名> 即可。你可以把它当做一个对象内部的变量。

如果访问了未设置值的属性，你会得到值 undefined ，——因为“未定义”啊！

和基本的值一样，对象也可以用简便的字面量方式创建，这种形式被称为*对象字面量*。

它的特征是一对大括号，标志着这个对象字面量的起始与结束。在大括号中你可以写下一个属性和一个值，二者用冒号分隔，值后面写一个逗号，表示这一项*成员*结束了。

const dog = {name: "Peter", age: 4};  
alert(dog.name); // "Peter"

现在我们拥有了一只小狗，并给它起了名字。我们可以利用合适的换行，让对象字面量看起来舒服一些：

const dog = {  
 name: "Peter",  
 age: 4,  
};  
alert(dog.name); // "Peter"

最后一项成员的逗号是可选的，你可以自由决定是否添加。

你可以重复定义对象中的一个属性，最后一次定义的值会覆盖掉先前的值。但是我们没必要这样做，如果要改变一个属性的值，随时可以通过访问它的方式来改变。

// 不要这样做  
const object = {  
 a: 1,  
 a: 2  
};  
alert(object.a); // 2  
  
// 你可以像这样  
const object2 = {  
 a: 1  
};  
object2.a = 2;  
alert(object2.a); // 2

对象字面量中所包含的信息构成了这个对象本身，自身也是值，因此你可以在对象字面量中将一个属性的值设置为另一个对象字面量。当一个对象成为属性的值时，你可以通过属性访问这个对象，进一步访问它的属性只需要在它后面接着写 .<属性名>即可。这种访问属性的方式被称为*点号访问法。*

const person = {  
 name: "Venn",  
 age: 18,  
 foods: {  
 apple: 5,  
 pear: 3  
 }  
};  
alert(person.name); // Venn  
alert(person.foods.apple); // 5

当然，你不能直接查看一个对象，否则你会得到一个看起来有些奇怪的字符串值：

alert(person); // "[object Object]"  
alert(person.foods); // "[object Object]"

**属性名**

如前文所说，对象的属性名可以看做是一个变量名，一个合法的标识符也可以直接用做属性名。但事实上，你可以用**任何字符**作为属性名，包括空格等。你只需要用双引号或单引号将属性名括起：

const person = {  
 "first name": "John",  
 "last name": "Doe",  
};

然后，可以这样访问属性值：

alert(person["first name"]); // John  
alert(person["last name"]); // Doe

这种方法被称为*方括号访问法*。如果属性名是合法的标识符，你也可以写成完全等价的点号访问法的形式。

const o = {  
 "aaa": "bbb"  
};  
alert(o["aaa"]); // "bbb"  
alert(o.aaa); // "bbb"

练习 3.4.1

1. 编写一个程序，提示用户输入一些个人信息，并将它存储到一个对象里。
2. 在 3.4.1.1 的基础上，从对象中访问某个信息，并告诉用户某个它的值。

### 成员操作

JavaScript 提供了一些方式来操作对象中的属性和值。我们已经讨论了访问和修改成员值的方式，添加一个成员也很简单——就像修改它一样，直接赋值即可。

const cat = {  
 name: "Kitty",  
 age: 4,  
 kind: "unknown"  
};  
  
// 现在我知道它是波斯猫  
cat.kind = "Persian cat";  
alert(cat.kind); // "Persian cat"

你也可以删除对象中的属性，JavaScript 提供了 delete 操作符。

const cat = {  
 name: "Kitty",  
 age: 4,  
 kind: "unknown"  
};  
  
// 现在我不需要关心它的品种  
delete cat.kind;  
alert(cat.kind); // undefined

这时候我们或许会有疑问：当一个属性被删除之后，再访问它就会得到 undefined。那么这跟直接把属性的值设为 undefined 有什么区别吗？

当然有！当属性值为 undefined 时，它只是一个值为 undefined 的属性，依然存在于对象中。但是，当你用 delete 删除一个属性的时候，这个对象中就不存在这个属性了，只不过是访问了不存在的属性会得到 undefined 而已。

你可以使用 in 操作符来查看对象中究竟是否存在某一个属性：

const cat = {  
 name: "Kitty",  
 age: 4,  
 kind: "unknown"  
};  
  
// 现在我不需要关心它的品种  
delete cat.kind;  
alert("kind" in cat); // false  
alert("name" in cat); // true

为什么当我们查看某个属性是否存在时，这个属性名要写成字符串？这是因为在 JavaScript 的语法中，如果直接写成 kind in cat 这样，kind 会被理解为一个标识符，然而我们并没有定义这个标识符，因此会得到一个错误。

事实上，JavaScript 中的对象的属性名都是以字符串方式存储的，只不过在对象字面量中，如果是合法的标识符，就不需要写成字符串的形式，JavaScript 会理解它。如果是点号标记法，由于它**只能用于访问写成合法标识符的属性名称**，因此属性名也不需要写成字符串的形式。但是在中括号标记法中，就应该写成字符串的形式。因为 JavaScript 会**将中括号中的内容当做表达式进行求值**，如果是一个字符串，那么就是按这个字符串对应的属性名进行访问，而如果认为是一个标识符，就会查找这个标识符本身的值所对应的属性名。

下面用一个示例来说明。

let a = "123";  
const object = {  
 a: 456,  
 "123": 123  
};  
alert(object.a); // 456  
alert(object[a]); // 123（标识符 a 的值是 "123"，那么访问了名为 "123" 的属性）  
alert(object["a"]); // 456（访问名为"a"的属性）

练习 3.4.2

1. 在 3.4.1.1 的基础上，提示用户输入一个名称，告诉用户这个对象中是否能找到名称对应的值。

* （提示：使用前面学习的逻辑相关知识）

### 枚举

上文所提供的访问对象成员的方式，都是通过属性名进行的。假如我们不知道某个属性的名称，或者说想一次性访问到所有的值，JavaScript 也提供了一些方便的函数来进行这类操作。

第一个是 Object.values 函数。我们可以从它得到一个列表，列表中会依次存放它所有的值。由于我们可以直接显示数组中的内容，因此它的所有值也就一目了然。

const cat = {  
 name: "Kitty",  
 age: 4,  
 kind: "Cheshire Cat"  
};  
  
const values = Object.values(cat);  
alert(values); // Kitty,4,Cheshire Cat

第二个是 Object.keys 函数，它得到一个对象中所有属性名称的列表。

const cat = {  
 name: "Kitty",  
 age: 4,  
 kind: "Cheshire Cat"  
};  
  
const keys = Object.keys(cat);  
alert(keys); // name,age,kind

此外，我们还可以使用 Object.entries 函数，它同样是得到一个列表，列表中交替保存属性名和值。

const cat = {  
 name: "Kitty",  
 age: 4,  
 kind: "Cheshire Cat"  
};  
  
const entries = Object.entries(cat);  
alert(entries); // name,Kitty,age,4,kind,Cheshire Cat

我们会在下一节中详细了解如何使用列表中的值。

### 全局对象

我们在这里需要明确一个之前没有提及的概念：

在 JavaScript 中，所有变量和常量都是一个*全局对象*的属性。

这个全局对象叫 window 。它在语义上指浏览器的窗口对象，同时也是 JavaScript 运行环境的一个“兜底”对象。当我们访问 window 对象的属性的时候，我们实际上是在访问这样一个变量/常量。

let hello = "Hello world";  
alert(window.hello); // "Hello world"  
alert(window.hello == hello); // true

由于 window 自身也是一个变量，因此它也是它自己的属性。

alert(window === window.window); // true  
alert(window.window === window.window.window); // true

一般情况下我们不需要直接使用 window 对象，不过我们可以使用它来判断一个标识符是否存在于当前环境中——换句话说，只要知道 window 对象是否有这个 A 属性，就可以是否声明过一个叫 A 的标识符。

练习 3.4.3

1. 编写一个程序，判断当前环境中是否存在一个变量或常量。
2. 编写一个程序，在 3.4.3.1 的基础上判断某个标识符是否为变量或常量。

* （提示：window 对象的属性值可改变）