17.7 对象操作运算符

对象操作运算符主要是指对对象、数组、函数执行特定任务操作的一组运算符，主要包括 in、instanceof、new、delete、.（点号）、[]（中括号）和()（小括号）运算符。

17.7.1 new 运算符

new 运算符可以根据构造函数创建一个新的对象，并初始化该对象。其语法如下：

```
new constructor( arguments)
```

constructor 必须是一个构造函数表达式，其后面应该是利用小括号包含的参数列表，参数可有可无，参数之间通过逗号进行分隔。如果函数调用时没有参数，可以省略小括号。

【示例 1】下面代码使用 new 运算符实例化 Array，并演示 3 种不同的使用方法。

```
var a = new Array;          // 创建数组结构的对象，省略了小括号
var b = new Array();        // 创建数组结构的对象
var c = new Array(1,2,3);   // 创建数组结构的对象，并初始化它的数据
alert(c[2]);                // 返回值 3。读取并显示新创建数组对象中的元素值
```

new 运算符被执行时，首先会创建一个新对象，接着 new 运算符调用指定的构造函数（类），这里是 Array 数组构造函数。并根据是否指定参数来初始化构造函数，并利用这个初始化的构造函数结构和数据（如果传递参数的话）初始化新对象。

【示例 2】下面代码自定义类，来使用它创建新的对象。

```
var a = function(){         // 自定义类 a 的数据结构
    this.x = 1;             // 类成员 x
    this.y = 2;             // 类成员 y
};
var b = new a;              // 创建自定义类 a 的对象实例
alert(b.x);                 // 返回 1，调用对象的成员
```

对于自定义类来说，只能够通过 new 运算符来进行实例化。

【示例 3】下面方法将返回 undefined。因为虽然把类的数据结构赋值给变量 b，但是由于没有实例化，所以无法访问。

```
var a = function(){         // 自定义类 a 的数据结构
    this.x = 1;             // 定义类成员 x
    this.y = 2;             // 定义类成员 y
};
var b =   a;                // 通过赋值运算符克隆自定义类的数据结构
alert(b.x);                 // 返回 undefined
```

【示例 4】对于下面这个对象结构来说，则可以使用赋值运算符进行快速引用：

```
var a ={                    // 自定义对象 a 数据结构
   x : 1,                   // 定义对象成员
   y : 2                    // 定义对象成员
};
var b = a;                  // 直接克隆对象数据结构
alert(b.x);                 // 返回 1，调用对象的成员
```

17.7.2 delete 运算符

delete 运算符能够删除指定对象的属性、数组元素或变量。

【示例 1】下面代码使用 delete 运算符删除对象 a 的属性 x。

```
var a ={                    // 定义对象 a
   x : 1,                   // 定义对象成员
```

```
    y : 2                    // 定义对象成员
};
alert(a.x);                  // 返回 1，调用对象成员
delete a.x;                  // 删除对象成员 x
alert(a.x);                  // 返回 undefined，没有找到该对象成员
```

执行 delete 运算时，如果删除操作成功，将返回 true，如果不能够被删除，则返回 false。

```
var a ={                     // 定义对象 a
    x : 1,
    y : 2
};
alert(delete a.x);           // 返回 true，说明删除成功
```

【示例 2】不是所有对象成员或变量都可以被删除的，如某些内置对象的预定义成员和客户端对象成员，使用 var 语句声明的变量也是不允许删除的。

```
a = 1;                       // 初始化变量 a，没有使用 var 语句声明
alert(delete a);             // 返回 true，说明删除成功
var b = 1;                   // 使用 var 语句声明并初始化变量
alert(delete b);             // 返回 false，说明不允许删除
alert(delete Object.constructor);
    // 返回 true，说明部分内部成员可以被删除
alert(delete Object.valueOf());
    // 返回错误，说明某些内部成员不可以被删除
```

【示例 3】如果删除不存在的对象成员，或者非对象成员、数组元素、变量时，它会返回 true，所以使用 delete 运算符时，应该注意这个问题，防止与成功删除操作相混淆。

```
var a ={                     // 定义对象 a
    x : 1,
    y : 2
};
alert(delete a);             // 返回 false，说明不允许删除
alert(delete a.z);           // 返回 true，说明不存在该属性
alert(delete Object);        // 返回 true，说明删除的不是成员、元素或变量
alert(delete b);             // 返回 true，说明不存在该变量
```

【提示】

使用 delete 运算符应该注意几个问题：

第一，delete 运算符只能删除值类型的数据。不影响变量、属性或数组元素存储的原引用对象。例如：

```
var a ={                     // 定义对象 a
    x : 1
};
var b ={                     // 定义对象 b
    y : a
};
alert(delete(b.y));          // 删除对象 b 中 y 属性对对象 a 的引用
alert(a.x);                  // 返回 1。但是原引用对象 a 并没有被删除
```

第二，delete 运算符的删除操作不是清空值，即把变量、属性或数组元素的值设置为 undefined，而是彻底删除它们占用的存储空间。在 JavaScript 1.1 和 JavaScript 1.0 版本中仅是把变量、属性或数组元素设置为 null。

第三，除了使用 delete 运算符手动清除不用的内存外，JavaScript 主要是利用内置的一个垃圾回收程序来自动对系统进行清理，所以并不需要手动调用 delete 运算符来释放对象所占用的空间。

第四，灵活使用 delete 运算符，配合 in 运算符，可以很方便的操作对象成员、数组元素等，如检测、插入、删除或更新操作。

```
var a =[];                   // 定义空数组 a
```

```
if("x" in a)                // 如果数组 a 中存在元素 x
    delete a["x"];          // 则删除元素 x
else                        // 如果不存在元素 x
    a["x"] = true;          // 则插入数组元素 x，并为其赋值 true
alert(a.x);                 // 返回 true。查看数组元素 x 的值
if(delete a["x"])           // 如果删除数组元素 x 成功
    a["x"] = false;         // 更新数组元素 x 的值为 false
alert(a.x);                 // 返回 false。查看数组元素 x 的值
```

17.7.3 中括号和点号运算符

中括号和点号都属于存取运算符，用于访问对象或数组。使用中括号运算符（[]）可以存取数组元素值，使用点号运算符（.）可以存取对象属性值。用法如下：

```
a.b                         // 点运算符的用法
c[d]                        // 中括号运算符的用法
```

在上面代码中，运算数 a 表示对象，运算数 b 表示一个标识符，如属性名。如果属性值是函数，应在标识名后面增加小括号运算符，实现方法调用操作。注意，运算数 b 是不能使用字符串，也不能够使用值为字符串的表达式。

运算数 c 可以是数组，也可以是对象。如果左侧运算数是数组，则中括号包含的运算数应是一个值为正整数的表达式（下标值）。如果左侧运算数是对象，则中括号包含的运算数应是一个值为字符串的表达式，它与对象属性名的字符串对应。

【示例 1】 中括号运算符（[]）不仅可以存取数组元素的值，还可以存取对象属性值。

- 读取数组元素的值

```
var a =[1,"x",true,{}];     // 定义数组 a
alert(a[1]);                // 返回"x"。读取数组中第 2 个元素的值
alert(a[3]);                // 返回[object Object]。读取数组中第 4 个元素的值
```

对于数组来说，可以通过数组下标来指定元素在数组中的位置，起始位置为 0。

- 写入数组元素的值

```
var a =[1,"x",true,{}];     // 定义数组 a
a[3] = false;               // 在数组第 4 个元素中写入 false 布尔值
alert(a[3]);                // 返回 false。元素原来存储的对象被覆盖
```

- 读取对象属性值

```
var a ={                    // 定义对象 a
   x : 1,                   // 定义对象属性 x
   y : function(){          // 定义对象方法 y
    return 2;               // 返回值 2
   }
};
alert(a["x"]);              // 返回 1。读取属性 x 的值
alert(a["y"]());            // 返回 2。调用方法 y
```

对于对象来说，可以通过对象属性名称字符串来指定成员在对象中的位置。

- 重置对象属性值

```
var a ={                    // 定义对象 a
   x : 1,                   // 定义对象属性 x
   y : function(){          // 定义对象方法 y
```

```
    return 2;                // 返回值 2
  }
};
a["x"] = 3;                  // 重置属性 x 的值
alert(a["x"]);               // 返回 3。读取属性 x 的值
a["y"] = function(){         // 更新方法 y
  return 4;
}
alert(a["y"]());             // 返回 4。调用方法 y
```

【示例 2】点号运算符（.）可以存取对象属性值，它比中括号灵活、方便，因为点号运算符右侧可以直接指定属性的标识符，而不是属性名称的字符串或变量。

```
var a ={                     // 定义对象 a
  x : 1,
};
alert(a.x);                  // 返回 1。读取对象属性 a 的值
a.x = 2;                     // 重写对象属性 a 的值
alert(a.x);                  // 返回 2。再次读取对象属性 a 的值
```

对于中括号运算符可以通过变量或字符串表达式来传递特定值。

```
var b = "x";                 // 把属性 x 的标识符名作为字符串存储在变量 b 中
var a ={                     // 定义对象 a
  x : 1                      // 定义属性 x
};
alert(a[b]);                 // 返回 1。通过变量间接获取对象 a 的属性 x 的值
alert(a.b);                  // 返回 undefined。点运算符无法识别这种变量引用法
```

中括号运算符能够对第二个运算数执行运算，并对返回值的类型进行转换。这种类型转换与关系运算符的类型转换规则类似。

【示例 3】对于下面两种方法都可以读取数组 a 中第二个元素的值。虽然说 a["1"]中参数是一个字符串，但是中括号运算符能够把它转换为数字。

```
var a = ["x",true,{}];       // 定义数组
alert(a[1]);                 // 返回 true
alert(a["1"]);               // 返回 true
```

与关系运算符不同，如果中括号运算符中第二个运算数为对象时，会使用 toString()方法进行转换，如果失败，则会调用 valueOf()方法转换。同时对于布尔值 true 和 false 将被转换为字符串"true"和"false"，而不是 1 和 0。；

```
var a = {                    // 定义对象
  "true":1,                  // 定义属性"true"。为了避免与系统标识符冲突，这里加了引号，以表示它是一个字符串
  "false":0                  // 定义属性"false"。为了避免与系统标识符冲突，这里加了引号，以表示它是一个字符串
}
alert(a[true]);              //返回 1。此时中括号运算符会先把布尔值 true 转换为字符串"true"，而不是数值 1
alert(a[false]);             //返回 0。此时中括号运算符会先把布尔值 false 转换为字符串"false"，而不是数值 0
```

当对象被用做关联数组时，由于对象的属性名是动态生成的，所以不能够使用点号运算符来准确操作对象属性。但是如果使用中括号运算符来操作对象属性时，反而更方便，借助 for 循环语句可以实现自动化读写操作。

【示例 4】下面代码能够遍历客户端 window 对象的所有属性以及属性值。这里主要使用了中括号运算符来操作 document 对象的属性，这种批量读取属性及其值的操作，如果使用点号运算符来实现是非常困难的，甚至是不可能的。

```
for(o in window){            // 遍历 window 对象成员，此时对象被看做关联数组
  document.write("window." + o + " = " + window[o] + "<br />");
}
```

如果点号运算符右侧的标识符不存在，在读取该成员时返回 undefined 值，而不是返回错误。例如：

```
var a = {                    // 定义对象
  x:1                        // 定义属性
}
```

```
alert(a.y);                    // 返回 undefined。读取不出来的成员
```

如果点号运算符右侧的标识符不存在，而为该标识符写入值时，会创建新的对象成员。例如：

```
var a = {                      // 定义对象
    x:1                        // 定义属性
}
alert(a.y);                    // 返回 undefined。说明不存在该成员
a.y = 2;                       // 新建属性，并写入值
alert(a.y);                    // 返回 2。说明存在该属性，且值为 2
a.y = function(){              // 重写该成员，设置该成员为一个方法，返回值为 3
   return 3;
};
alert(a.y());                  // 返回 3。再次调用该方法
```

17.7.4 小括号运算符

小括号是一个特殊的运算符，它没有固定数目的运算数。其中第一个运算数必须是一个函数名或者引用函数的表达式，其后附加小括号运算符，小括号中可以包含数量没有限制的运算数，它们之间通过逗号进行分隔。语法如下：

```
f (a,b,c……)
```

其中运算数 f 是一个函数名或者引用函数的表达式，a、b、c……是数目不详的参数，这些参数可以是任意类型的表达式。

【示例】下面代码演示了如何使用小括号运算符调用函数的过程。

```
function a(){                  // 定义函数 a
    alert("Hello,World");      // 函数体包含的语句
}
a;                             // 直接引用函数名，没有反应
a();                           // 返回提示信息"Hello,World"。没有传递参数，直接调用
a(1,"string",{},true);         // 返回提示信息"Hello,World"。传递 4 个不同类型的参数
```

小括号运算符在执行时是这样的：先对每个运算数进行计算，然后调用第一个运算数所指的函数，同时把余下的运算数的值传递给函数作为它的参数。