17.5 关系运算符

关系运算符也称为比较运算符，它反映了运算数之间关系的一类运算，因此这类运算符一般都是二元运算符，关系运算返回的值总是布尔值。

17.17.1 大小比较

基本大小关系的比较运算，以及对应运算符说明如表 E17.9 所示。

表 E17.9 基本比较运算

比较运算符	说明
<	如果第一个运算数小于第二个运算数，则比较运算的返回值为 true。否则比较运算的返回值为 false
<=	如果第一个运算数小于等于第二个运算数，则比较运算的返回值为 true。否则比较运算的返回值为 false
>=	如果第一个运算数大于等于第二个运算数，则比较运算的返回值为 true。否则比较运算的返回值为 false
>	如果第一个运算数大于第二个运算数，则比较运算的返回值为 true。否则比较运算的返回值为 false

比较运算中的运算数不局限于数值，可以是任意类型的数据。但是在执行运算时，它主要根据数值的大小，以及字符串中字符在字符编码表中的位置值来比较大小。所以对于其他类型的值，将会被转换为数字或字符串，然后再进行比较。

【示例】在比较运算中，运算数的转换操作规则说明如下。

- 如果运算数都是数字，或者都可以被转换成数字，则将根据数字大小进行比较。

```
alert(4>3);              // 返回 true，直接利用数值大小进行比较
alert("4">Infinity);     // 返回 false，无穷大比任何数字都大
```

但是对于下面代码来说，就比较特殊了。两个运算数虽然都可以被转换为数字，但是由于它们都是字符串，则不再执行数据类型转换，而是直接根据字符串进行比较。

```
alert("4">"3");          // 返回 true，以字符串进行比较，而不是数字大小进行比较
```

- 如果运算数都是字符串，或者都被转换为字符串，那么将根据字符在字符编码表中的位置大小进行比较。同时字符串是区分大小写的，因为大小写字符在表中的位置不同。一般小写字符大于大写字符。如果比较中不区分大小写，则建议使用 toLowerCase()或 toUpperCase()方法把字符串统一为小写或大写形式。

```
alert("a">"b");          // 返回 false，字符 a 编码为 61，字符 b 编码为 62
alert("ab">"cb");        // 返回 false，c 的编码为 63。从左到右对字符串中对应字
                                符逐个进行比较
alert("abd">"abc");      // 返回 true，d 的编码为 64。前面字符相同，则比较下一个字符
```

- 如果一个运算数是数字，或者被转换为数字，另一个是字符串，或者被转换为字符串，则比较运算符调用 parseInt()将字符串强制转换为数字，不过对于非数字字符串来说，将被转换为 NaN 值，最后以数字方式进行比较。运算数是 NaN，则比较结果为 false。

```
alert("a">"3");          // 返回 true，字符 a 编码为 61，字符 3 编码为 33
alert("a">3);            // 返回 false，字符 a 被强制转换为 NaN
```

- 如果运算数都无法转换为数字或字符串，则比较结果为 false。
- 如果一个运算数为 NaN，或者被转换为 NaN，则始终返回 false。
- 如果对象可以被转换为数字或字符串，则执行数字或字符串比较。

17.17.2 案例：包含检测

in 运算符能够判断左侧运算数是否为右侧运算数的成员。其中左侧运算数应该是一个字符串，或者可以转换为字符串。右侧运算数则应该是一个对象或数组。

【示例 1】下面代码演示了如何利用 in 运算符检测属性 a、b、c、valueOf 是否为对象 o 的成员。

```
var o = {                    // 定义对象结构
    a:1,                     // 定义对象的属性 a
    b:function(){}           // 定义对象的方法 b
}
alert("a" in o);             // 返回 true
alert("b" in o);             // 返回 true
alert("c" in o);             // 返回 false
alert("valueOf" in o);
//返回 true，继承 JavaScript 为所有 Object 对象定义的方法
alert("constructor" in o);
//返回 true，继承 JavaScript 为所有 Object 对象定义的属性
```

使用 instanceof 运算符检测对象实例是否属于某个类或构造函数。其中 instanceof 运算符左侧运算数是对象实例名，右侧运算数是类名或构造器名。

【示例 2】下面代码演示了如何使用 instanceof 运算符检测数组 a 是否为 Array、Object 和 Function 的实例。

```
var a = new Array();          // 定义变量 a 为构造函数 Array 的一个对象实例
alert(a instanceof Array);    // 返回 true
alert(a instanceof Object);   // 返回 true，所有对象都是 Object 类的实例
alert(a instanceof Function);       // 返回 false
```

如果左侧运算数不是对象，或右侧运算数不是类或构造函数，则将返回 false。如果右侧运算数不是对象，则将返回错误。

17.17.3 案例：等值检测

JavaScript 提供了四个等值检测运算符：全等（===）和不全等（!==）、相等（==）和不相等（!=）。详细说明如表 E17.10 所示。

表 E17.10 等值运算

比较运算符	说明
==（相等）	比较两个运算数的返回值，看是否相等
!= （不相等）	比较两个运算数的返回值，看是否不相等
===（全等）	比较两个运算数的返回值，看是否相等，同时检测它们的数据类型是否相等
!== （不全等）	比较两个运算数的返回值，看是否不相等，同时检测它们的数据类型是否不相等

在相等运算中，一般遵照如下基本规则进行比较：

- 如果运算数是布尔值，在比较之前先转换为数值。其中 false 转为 0，true 转换为 1。
- 如果一个运算数是字符串，另一个运算数是数字，在比较之前先尝试把字符串转换为数字。
- 如果一个运算数是字符串，另一个运算数是对象，在比较之前先尝试把对象转换为字符串。
- 如果一个运算数是数字，另一个运算数是对象，在比较之前先尝试把对象转换为数字。
- 如果两个运算数都是对象，那么比较它们的引用值（引用地址）。如果指向同一个引用对象，则相等，否则不等。

【示例 1】下面是一些特殊运算数的比较。

```
alert("1" == 1)              // 返回 true。字符串被转换为数字
alert(true == 1)             // 返回 true。true 被转换为 1
alert(false == 0)            // 返回 true。false 被转换为 0
alert(null == 0)             // 返回 false
alert(undefined == 0)        // 返回 false
alert(undefined == null)     // 返回 true
alert(NaN == "NaN")          // 返回 false
```

```
alert(NaN == 1)              // 返回 false
alert(NaN == NaN)            // 返回 false
alert(NaN != NaN)            // 返回 true
```

NaN 与任何值都不相等，包括它自己。null 和 undefined 值相等，但是它们是不同类型的数据。在相等比较中，null 和 undefined 是不允许被转换为其他类型的值。

【示例 2】下面两个变量的值虽然是通过计算得到，但是它们的值是相等的。

```
var a = "abc" + "d";
var b = "a" + "bcd";
alert(a == b);               // 返回 true
```

对于值类型的数据而言，数值和布尔值的相等比较运算效果比较高，但是字符串需要消耗大量资源，因为字符串需要逐个字符进行比较，才能够确定它们是否相等。

在全等运算中，一般遵照如下基本规则进行比较：

- 如果运算数都是值类型，则只有数据类型相同，且数值相等时才能够相同。
- 如果一个运算数是数字、字符串或布尔值（值类型），另一个运算数是对象等引用类型，则它们肯定不相同。
- 如果两个对象（引用类型）比较，则比较它们的引用地址。

【示例 3】下面是特殊运算数的全等比较。

```
alert(null === undefined)    // 返回 false
alert(0 === "0")             // 返回 false
alert(0 === false)           // 返回 false
```

【示例 4】下面是两个对象的比较，由于它们都引用相同的地址，所以返回 true。

```
var a = {};
var b = a;
alert(a === b) ;             // 返回 true
```

但是对于下面两个对象来说，虽然它们的结构相同，由于地址不同，所以也不全等。

```
var a = {};
var b = {};
alert(a === b) ;             // 返回 false
```

【示例 5】对于引用类型的值进行比较，主要比较引用的地址是否相同，而不是比较它们的值。

```
var a = new String("abcd")   // 定义字符串"abcd"对象
var b = new String("abcd")   // 定义字符串"abcd"对象
alert(a === b);              // 返回 false
alert(a == b);               // 返回 false
```

在上面示例中，两个对象的值相等，但是它们的引用地址不同，所以它们即不相等，也不全等。事实上，对于引用类型的值来说，相等（==）和全等（===）运算符操作的结果是相同的，没有本质区别。

【示例 6】对于值类型而言，只要类型相同，值相等，它们就应该完全全等，这里不需要考虑比较运算数的表达式数据类型变化，也不用考虑变量的引用地址。

```
var a = "1" + 1;
var b = "11" ;
alert(a === b);              // 返回 true
```

【示例 7】表达式(a > b || a == b)与表达式(a >= b)并不完全相等。

```
var a = 1;
var b = 2;
alert((a > b || a == b) == (a >= b))    // 返回 true，此时似乎相等
```

如果为变量 a 和 b 分别赋值为 null 和 undefined，则返回值为 false，说明这两个表达式并非完全等价。

```
var a = null;
var b = undefined;
alert((a > b || a == b) == (a >= b))    // 返回 false，表达式的值并非相等
```

因为 null==undefined 等于 true，所以(a > b || a == b)表达式返回值就为 true。但是表达式 null>=undefined 返回值为 false。