17.3 算术运算符

算术运算符包括：加（+）、减（-）、乘（*）、除（/）、余数运算符（%）、数值取反运算符（-）。

17.3.1 加法运算

【示例 1】特殊运算数的运算结果比较特殊，需要特别留意。

```
var n = 5;                          // 定义并初始化任意一个数值
alert(NaN + n);                     // 返回 NaN。NaN 与任意运算数相加，结果都是 NaN
alert(Infinity + n);                //返回 Infinity。Infinity 与任意运算数相加，结果都是 Infinity
alert(Infinity + Infinity);         //返回 Infinity。Infinity 与  Infinity 相加，结果是 Infinity
alert(( - Infinity) + ( - Infinity));  //返回-Infinity。负 Infinity 相加，结果是负 Infinity
alert(( - Infinity) + Infinity);    //返回 NaN。正负 Infinity 相加，结果是 NaN
```

【示例 2】加运算符能够根据运算数的数据类型，尽可能的把数字转换成可以执行相加或相连运算的数值或字符串。

```
alert( 1 + 1);                      // 返回 2。如果运算数都是数值，则进行相加运算
alert( 1 + "1");                    //返回"11"。如果运算数中有一个是字符串，则把数值转换位字符串，然后进行相连运算
alert( "1" + "1");                  // 返回"11"。如果运算数都是字符串，则进行相连运算
```

【示例 3】下面两个表达式中，由于空字符串的位置不同，运算结果也是不同。在第一行代码中，3.0 和 4.3 都是数值类型，因此加号运算符就执行相加操作，由于第 3 个运算数是字符串，则把第一个加号运算结果转换为字符串并与空字符串进行相连操作。而第二行代码中则不同，第一个加号运算符首先把数值 3.0 转换为字符串，然后执行连接操作，所以结果也就不同。

```
alert(3.0 + 4.3 + "")               // 返回"7.3"
alert(3.0 + "" + 4.3)               // 返回"34.3"
```

【提示】

为了避免误解，使用加法运算符时，应先检查运算数的数据类型是否符合需要。

17.3.2 减法运算

【示例 1】特殊运算数的运算结果比较特殊，需要特别留意。

```
var n = 5;                          // 定义并初始化任意一个数值
alert(NaN - n);                     // 返回 NaN。NaN 与任意运算数相减，结果都是 NaN
alert(Infinity - n);                //返回 Infinity。Infinity 与任意运算数相减，结果都是 Infinity
alert(Infinity - Infinity);         //返回 NaN。Infinity 与 Infinity 相减，结果是 NaN
alert(( - Infinity) - ( - Infinity));  //返回 NaN。负 Infinity 相减，结果是 NaN
alert(( - Infinity) - Infinity);    //返回-Infinity。正负 Infinity 相减，结果是-Infinity
```

【示例 2】在减法运算中，如果有一个运算数不是数字，则返回值为 NaN；如果数字为字符串，则会把它转换为数值之后，再进行运算。

```
alert(2 - "1");                     // 返回 1
alert(2 - "a");                     // 返回 NaN
```

利用减法运算可快速把一个值转换为数字。例如，由于 HTTP 请求值一般都是字符串数字，可以让这些字符串减去 0 快速转换为数值。这与调用 parseFloat()方法结果相同，但减法运输符更高效、更快捷。减法运算符的隐性转换如果失败，则返回 NaN，这与使用 parseFloat()方法执行转换时返回值是不同的。

【示例 3】对于字符串来说，减法运算符能够完全匹配进行转换，如果字符串是非数字的值，则返回 NaN；而 parseFloat()方法则通过逐字符解析并努力转换为数值。

例如，对于字符串"100aaa"而言，parseFloat()方法能够解析出前面几个数字，而对于减法运算符来说，则必须是完整的数字时，可以进行完全匹配转换。

```
alert(parseFloat("100aaa"));        // 返回 100
alert(parseFloat("aaa100"));        // 返回 NaN
```

```
alert("100aaa" - 0);                    // 返回 NaN
alert("100" - 0);                       // 返回 100
```

对于布尔值来说，parseFloat()方法能够把 true 转换为 1，把 false 转换为 0，而减法运算符视其为 NaN。

对于对象来说，parseFloat()方法直接尝试调用对象的 toString()方法进行转换，而减法运算符先尝试调用对象的 valueOf()方法进行转换，失败之后再调用 toString()进行转换。

17.3.3 乘法运算

两个正数相乘，则为正数；两个负数相乘，则为正数；一正一反相乘，则为负数。

【示例】特殊运算数的运算结果比较特殊，需要特别留意。

```
var n = 5;                              //定义并初始化任意一个数值
alert(NaN * n);                         //返回 NaN。NaN 与任意运算数相乘，结果都是 NaN
alert(Infinity * n);                    //返回 Infinity。Infinity 与任意非 0 正数相乘，结果都是 Infinity
alert(Infinity * ( - n));
//返回 Infinity。Infinity 与任意非 0 负数相乘，结果都是-Infinity，换句话说结果的符号由第二个运算数的符号决定
alert(Infinity * 0);                    //返回 NaN。Infinity 与 0 相乘，结果是 NaN
alert(Infinity * Infinity);             //返回 Infinity。Infinity 与 Infinity 相乘，结果是 Infinity
```

17.3.4 除法运算

两个正数相除，则为正数；两个负数相除，则为正数；一正一反相除，则为负数。

【示例】特殊运算数的运算结果比较特殊，需要特别留意。

```
var n = 5;                              // 定义并初始化任意一个数值
alert(NaN / n);                         // 返回 NaN。如果某个运算数是 NaN，结果都是 NaN
alert(Infinity / n);
//返回 Infinity。Infinity 被任意数字除，结果都是 Infinity 或-Infinity，符号由第二个运算数的符号决定
alert(Infinity / Infinity);//返回 NaN
alert(n / 0);
//返回 Infinity。0 除一个非无穷大的数字，结果是 Infinity 或-Infinity，符号由第二个运算数的符号决定
alert(n / -0);                          //返回-Infinity。参考上一行注释说明
```

17.3.5 余数运算

也称模运算，通俗说就是求余数。例如：

```
alert(3 % 2);                           // 返回余数 1
```

模运算主要针对整数执行操作，但是它也适用浮点数，例如：

```
alert(3.1 % 2.3);                       // 返回余数 0.8000000000000003
```

【示例】特殊运算数的运算结果比较特殊，需要特别留意。

```
var n = 5;                              // 定义并初始化任意一个数值
alert(Infinity % n);                    // 返回 NaN
alert(Infinity % Infinity);             // 返回 NaN
alert(n % Infinity);                    // 返回 5
alert(0 % n);                           // 返回 0
alert(0 % Infinity);                    // 返回 0
alert(n % 0);                           // 返回 NaN
alert(Infinity % 0);                    // 返回 NaN
```

17.3.6 取反运算

取反运算符是一元运算符，或称一元减法运算符。

【示例】下面列举特殊运算数的取反运算结果。

```
alert(-5);                    // 返回-5。正常数值取负数
alert(-"5");                  // 返回-5。先转换字符串数字为数值类型
alert(-"a");                  // 返回 NaN。无法完全匹配运算，返回 NaN
alert(-Infinity);             // 返回-Infinity
alert(-(-Infinity));          // 返回 Infinity
alert(-NaN);                  // 返回 NaN
```

【提示】

与一元减法运算符相对应的还有一个一元加法运算符，在实际开发中，一元加法运算符很少使用，不过可以利用它把非数值型的数字快速的转换为数值型数值。

17.3.7 递增和递减

递增（++）和递减（--）运算就是通过不断加 1 或减 1 以实现改变自身值的一种简洁方法。递增运算符和递减运算符是一元运算符，只能够作用于变量、数组元素或对象属性，这是因为在运算过程中会执行赋值运算，赋值运算左侧必须是一个变量、数组元素或对象属性，只有这样赋值才得以实现。

【示例 1】下面代码是错误用法：

```
alert(4++);                   // 返回错误
```

下面代码是正确的用法：

```
var n = 4;
alert(n++);                   // 返回 4
```

递增运算符和递减运算符有位置讲究，位置不同所得运算结果也不同。

【示例 2】下面递增运算符是先执行赋值运算，然后再执行递加运算。即先计算表达式的返回值，最后才把自身值递加。

```
var n = 4;
alert(n++);                   // 返回 4
```

而下面的递增运算符是先执行递加运算，再返回表达式的值。

```
var n = 4;
alert(++n);                   // 返回 5
```

【示例 3】下面代码可以直观演示每个表达式与变量 n 的值并非都是同步的。

```
var n = 4;
alert(n++);                   // 返回 4
alert(++n);                   // 返回 6。在递加之前，变量 n 的值是 5，而不是 4
```

递增运算符和递减运算符是相反操作的一对。它们在运算之前都会试图转换值为数值类型，如果失败则返回 NaN。