17.8 其他运算符

下面介绍其他没有分类的运算符。这些运算符在程序中经常使用，也是非常重要的。

17.8.1 条件运算符

条件运算符与条件语句在逻辑上是相同。但条件运算符侧重于连续运算，它自身可以作为表达式，也可以作为子表达式使用；而条件语句侧重于逻辑结构，在结构中执行不同的运算。条件运算符拥有函数式特性，而条件语句具有面向对象的编程结构。

条件运算符是 JavaScript 唯一的三元运算符，语法形式如下：

```
a ? x : y
```

其中 a、x、y 是它的三个运算数。a 运算数必须是一个布尔型的表达式，即返回值必须是一个布尔值，一般使用比较表达式来表示。x 和 y 是任意类型的值。如果运算数 a 返回值为 true 时，将执行 x 运算数，并返回该表达式的值。如果运算数 a 返回值为 false 时，将执行 y 运算数，并返回该表达式的值。

【示例】定义变量 a，然后检测 a 是否被赋值，如果赋值则使用该值，否则使用默认值给它赋值。

```
var a;                                  // 定义变量 a
a ? (a = a) : (a = "Default Value");    // 检测变量 a 是否赋值
alert(a);                               // 显示变量 a 的值
```

条件运算符可以转换为条件语句：

```
var a;
if(a)                                   // 赋值
    a=a;
else                                    // 没有赋值
    a = "default value";
alert(a);
```

条件运算符也可以转换为逻辑表达式：

```
var a;
a && (a=a) || (a = "default value");    // 逻辑表达式
alert(a);
```

在上面表达式中，如果 a 为 true，则执行(a=a)表达式，执行完毕就不再执行逻辑或运算符后面的(a = "default value")表达式。如果 a 为 false，则不再执行逻辑与运算符后面的(a=a)表达式，同时将不再继续执行逻辑或运算符前面的表达式 a && (a=a)，转而执行逻辑或运算符后面的表达式(a = "default value")。

【提示】

上面代码仅是演示，在实战中用户需要考虑假值的影响。因为，当变量赋值 0、null、undefined、NaN 等假值时，它们被转换为逻辑值也是 false。

17.8.2 逗号运算符

逗号运算符是二元运算符，它能够先执行运算符左侧的运算数，然后再执行右侧的运算数，最后仅把右侧运算数的值作为结果返回。

【示例 1】逗号运算符可以实现连续声明多个变量并赋值。

```
var a = 1, b = 2, c = 3, d = 4;
```

它等于：

```
var a = 1;
var b = 2;
var c = 3;
var d = 4;
```

多个逗号运算符可以联排使用，从而设计一种多重计算的功效。

```
var a =   b = 2, c =   d = 4;
```

与条件运算符或逻辑运算符根据条件来决定是否执行所有或特定运算数不同，逗号运算符会执行所有的运算数，但并非返回所有运算数的结果，它只返回最后一个运算数的值。

【示例 2】下面代码中，变量 a 的值是逗号运算之后，通过第二个运算数 c=2 的执行结果赋值得到。第一个运算数的执行结果没有返回，但是这个表达式被执行了。

```
a = (b=1,c=2);                    // 连续执行和赋值
alert(a);                         // 返回 2
alert(b);                         // 返回 1
alert(c);                         // 返回 2
```

逗号运算符可以作为仅需执行表达式的工具，这些表达式不需要返回值，但必须要运算。在特定环境中，可以在一个表达式中包含多个子表达式，通过逗号运算符仅让它们全部执行，而不用返回全部结果。

【示例 3】下面代码中，for 语句的条件表达式中仅能够包含三个表达式，第一个表达式是初始化循环值，第二个表达式是布尔值，第三个表达式是循环变量的递增值。为了能够在三个表达式中完成各种计算任务，这里把逗号运算符发挥到极致。但是，要确保在第二个循环条件的第二个表达式，最后一个表达式返回一个逻辑值，否则会导出循环出现错误。

```
for(a = 1, b = 10, c = 100; ++ c, a < b; a ++ , c -- ){
   alert(a * c);
}
```

【示例 4】逗号运算符的优先级是最低的。在下面代码中，赋值运算符优先于逗号运算符，也就是说数值 1 被赋值给变量 b 之后，继续赋值给变量 a，最后才执行逗号运算符。

```
a = b=1,c=2;                      // 连续执行和赋值
alert(a);                         // 返回 1
alert(b);                         // 返回 1
alert(c);                         // 返回 2
```

17.8.3 void 运算符

void 是一元运算符，它可以出现在任意类型的运算数之前，执行运算数，却忽略运算数的返回值，结果总返回一个 undefined。void 多用于 URL 中执行 JavaScript 表达式，但不需要表达式的计算结果。

【示例 1】在下面代码中，使用 void 运算符让表达式返回 undefined。

```
var a = b = c = 2;                // 定义并初始化变量的值
d = void (a -= (b *= (c += 5)));  // 执行 void 运算符，并把返回值赋予给变量 d
alert(a);                         // 返回-12
alert(b);                         // 返回 14
alert(c);                         // 返回 7
alert(d);                         // 返回 undefined
```

由于 void 运算符的优先级比较高（14），高于普通运算符的优先级，所以在使用时应该使用小括号明确 void 运算符操作的运算数，避免发生错误。

【示例 2】在下面两行代码中，由于第一行代码没有使用小括号运算符，则 void 运算符优先执行，返回值 undefined 再与 1 执行减法运算，所以返回值为 NaN。在第二行代码中由于使用小括号运算符明确 void 的运算数，减法运算符先被执行，然后再执行 void 运算，最后返回值是 undefined。

```
alert(void 2 - 1);                // 返回 NaN
alert(void (2 - 1));              // 返回 undefined
```

【示例 3】在下面代码中，undefined 是一个变量，由于 void 运算符返回值是 undefined，所以该变量的值就等于 undefined。由于早期 IE 浏览器对 undefined 数据类型支持不是很好，如果直接调用 undefined 就会出错，但是如果使用变量 undefined 来代替直接量 undefined 就能够避开这个 Bug。

```
var undefined = void null;
```

也可以调用一个空函数，其返回值为 undefined，来定义变量 undefined：

```
var undefined = function(){}();
```

使用下面代码来定义 undefined 变量：

```
var undefined = void 0;
```

【示例 4】void 运算符也能像函数一样使用，如 void(0)也是合法的。在特殊环境下一些复杂的语句可能不方便使用 void 关键字形式，而必须要使用 void 的函数形式。

```
void(i=0);                    // 返回 undefined
void(i=0, i++);               // 返回 undefined
```

17.8.4 typeof 运算符

typeof 是一元运算符，其后可以跟随任意类型的运算数，计算之后将返回一个字符串，描述运算数的数据类型。针对不同类型的数据所返回的字符串如表 E17.12 所示。

表 E17.12 数据类型的 typeof 运算返回值

数据类型	typeof 返回的字符串	数据类型	typeof 返回的字符串
数值	"number"	字符串	"string"
布尔型	"boolean"	对象	"object"
数组	"object"	null	"object"
函数	"function"	未定义或初始化	"undefined"
NaN	"number"	undefined	"undefined"

【示例】在下面代码中，使用 typeof 运算符获取变量的类型，以此判断它是否被定义或初始化，如果没有定义或初始化，则重新为其赋值。

```
var a;                        // 定义变量
(typeof a == "undefined") && (a = "string")
                              // 通过类型来检测变量 a 的状态
alert(a);                     // 返回"string"
```

这种方法比下面这种使用逻辑运算符来检测会安全很多：

```
var a;
a && (a=a) || (a = " string ");      // 逻辑表达式
alert(a);
```

因为不管 a 的值是什么，只要赋值，它的类型都不会是 undefined。不过为其赋值为 undefined 例外，因为赋值 undefined 将返回 undefined 类型：

```
var a=undefined;
alert(typeof a);              // 返回 undefined
(typeof a == "undefined") && (a = "string")
alert(a);                     // 返回"string"
```