17.4 逻辑运算符

逻辑运算与布尔值紧密相关联，故也称布尔代数。所谓布尔代数就是布尔值（true 和 false）的“算术”运算。逻辑运算常与比较运算结合使用，在条件表达式中经常应用。

逻辑运算符包括：与（&&）、或（||）和非（!）三种逻辑运算类型。

17.4.1 逻辑与运算

逻辑与运算符（&&）实际上就是两个运算数的 AND 布尔操作，只有当两个条件都为 true 时，它才返回 true，否则返回 false，详细描述如表 E17.7 所示。

表 E17.7 逻辑与运算符

第一个运算数的布尔值	第二个运算数的布尔值	逻辑与运算结果
true	true	true
true	false	false
false	true	false
false	false	false

逻辑与运算符（&&）的逻辑解析：

首先，计算第一个运算数，即左侧表达式。如果左侧的表达式的计算值可以被转换为 false（如 null、0、underfined 等），那么就会结束计算，直接返回第一个运算数的值。

然后，当第一个运算数的值为 true 时，则将计算第二个运算数的值，即位于右侧的表达式，并返回这个表达式的值。

【示例 1】下面代码利用逻辑与运算检测变量初始值。

```
var user;                          // 定义变量
( ! user && alert("没有赋值"));  // 返回提示信息“没有赋值”
```

如果变量 user 为 null，则!user 就会返回 true，如果逻辑与运算符左侧返回值为 true，则会执行右侧的表达式，否则就会忽略。也就是说逻辑与运算符右侧的表达式可以被执行，也可以不被执行。对于上面表达式，使用条件语句可以进行如下表示：

```
var user;                          // 定义变量
if( ! user){                       // 条件判断
   alert("变量没有赋值呀");
}
```

【示例 2】由于逻辑与运算符右侧的表达式将根据左侧的表达式的值来决定是否执行，在程序中常利用它来设计结构简洁的条件运算。

```
var n = 3;
(n == 1) && alert(1);
(n == 2) && alert(2);
(n == 3) && alert(3);
( ! n) && alert("null");
```

上面代码等价于下面的多条件逻辑结构：

```
var n = 3;                         // 定义变量
switch (n){                        // 指定判断的变量
   case 1 :                        // 条件 1
     alert(1);
     break;                        // 结束结构
   case 2 :                        // 条件 2
     alert(2);
     break;                        // 结束结构
   case 3 :                        // 条件 3
```

```
        alert(3);
        break;                          // 结束结构
      default :                         // 默认条件
        alert("null");
    }
```

【示例 3】利用逻辑与运算符替代条件结构，是一种便捷的设计技巧，但是在使用时应该慎重。

```
    var user = 0;                       // 定义并初始化变量
    ( ! user && alert("变量没有赋值呀")); // 返回提示信息"变量没有赋值呀"
```

上面代码设计思路：如果变量没有赋值，则其值为 null，转换为布尔值就是 false，然后利用逻辑与运算符来判断变量是否初始化。由于变量 user 值为 0 时，转换为布尔值时，则为 false。所以，当变量赋值之后，依然提示变量没有赋值。

为了安全起见，用户在设计时必须确保逻辑与左侧的表达式返回值是可以预期的，同时右侧表达式不应该包含赋值、递增、递减和函数调用等有效运算。

逻辑与运算的运算数可以是任意类型数据，如果运算数不是布尔值，则逻辑与运算并非要求必须返回布尔值，而是根据表达式的结果实事求是的进行返回。

【示例 4】下面介绍几种特殊运算数应用技巧。

- 对象被转换为布尔值时为 true。例如，一个空对象与一个布尔值进行逻辑与运算。

```
    alert(typeof({} && true))       // 返回第二个运算数 true 的类型，即返回 boolean
    alert(typeof({} && false))      // 返回第二个运算数 false 的类型，即返回 boolean
    alert(typeof(true && {}))       // 返回第二个运算数{}的类型，即返回 object
    alert(typeof(false && {}))      // 返回第一个运算数 false 的类型，即返回 boolean
```

如果运算数中包含 null，则返回值总是 null。例如，字符串"null"与 null 类型值进行逻辑与运算，不管位置如何，始终都返回 null 的类型 object。

```
    alert(typeof("null" && null))   // 返回 null 的类型，即返回 object
    alert(typeof(null && "null"))   // 返回 null 的类型，即返回 object
```

如果运算数中包含 NaN，则返回值总是 NaN。例如，字符串"NaN"与 NaN 类型值进行逻辑与运算，不管位置如何，始终都返回 NaN 的类型 number。

```
    alert(typeof("NaN" && NaN))     // 返回 NaN 的类型，即返回 number
    alert(typeof(NaN && "NaN"))     // 返回 NaN 的类型，即返回 number
```

- 对于 Infinity 特殊值来说，将被转换为 true，与普通数值一样参与逻辑与运算。

```
    alert(typeof("Infinity" && Infinity))       // 返回第二个运算数 Infinity 的类型，即返回 number
    alert(typeof(Infinity && "Infinity"))       // 返回第二个运算数"Infinity"的类型，即返回 string
```

- 如果运算数中包含 undefined，则返回错误。例如，字符串"undefined"与 undefined 类型值进行逻辑与运算，不管位置如何，始终都返回 undefined 的类型 undefined。

```
    alert(typeof("undefined" && undefined))
    alert(typeof(undefined && "undefined"))
```

17.4.2 逻辑或运算符

当逻辑或运算符（||）左右两侧运算数的值都是布尔值时，则它将执行布尔 OR 操作。如果两个运算数的值为 true，或者其中一个为 true，那么它就返回 true，否则就会返回 false。详细描述如表 E17.8 所示。

表 E17.8 逻辑或运算符

第一个运算数的布尔值	第二个运算数的布尔值	逻辑或运算结果
true	true	true
true	false	true
false	true	true
false	false	false

逻辑或运算符（||）的逻辑解析：

首先，计算第一个运算数。如果左侧的表达式的计算值可以被转换为 true，那么就直接返回第一个运算数的值，忽略第二个运算数（即不执行）。

然后，当第一个运算数的值为 false 时，则将计算第二个运算数的值，即位于右侧的表达式，并返回这个表达式的值。

【示例】针对下面三个表达式：

```
var n = 3;
(n == 1) && alert(1);
(n == 2) && alert(2);
(n == 3) && alert(3);
( ! n) && alert("null");
```

可以使用逻辑或对其进行合并：

```
var n = 2;
(n == 1) && alert(1)||(n == 2) && alert(2)||(n == 3) &&
    alert(3)||( ! n) && alert("null");
```

由于&&运算符的优先级高于||运算符的优先级，所以不用使用小括号。不过使用小括号运算符更方便阅读：

```
var n = 2;
((n == 1) && alert(1) ) || ((n == 2) && alert(2)) || ((n == 3) &&
    alert(3) ) || (( ! n) && alert("null"));
```

或者分行书写：

```
var n = 2;
((n == 1) && alert(1) ) ||      // 为 true 时，结束并返回值
((n == 2) && alert(2)) ||       // 为 true 时，结束并返回值
((n == 3) && alert(3) )||       // 为 true 时，结束并返回值
(( ! n) && alert("null"));      // 为 true 时，结束并返回值
```

即使逻辑或运算符的运算数不是布尔值，但是仍然可以将它看做布尔 OR 的操作，也不管运算数的值是什么类型，都可以被转换为布尔值。

逻辑或运算和逻辑与运算是两个互为反的操作，对于 null、NaN 特殊值都返回相应的 null 或 NaN，而对于 undefined 将返回错误。

17.4.3 逻辑非运算符

逻辑非运算符（!）是一元运算符，直接放在运算数之前，将对运算数执行布尔取反操作（NOT），并返回布尔值。

【示例 1】如果对于运算数执行两个逻辑非运算操作，实际上它相当于把运算数转换为布尔值数据类型。

```
alert(!5);              // 返回 false。把数值 5 转换为布尔值，并取反
alert(!!5);             // 返回 true。把数值 5 转换为布尔值
alert(!0);              // 返回 true。把数值 0 转换为布尔值，并取反
alert(!!0);             // 返回 false。把数值 5 转换为布尔值
```

【提示】

逻辑与和逻辑或运算符所执行的操作返回的未必都是布尔值，但是对于逻辑非运算符来说，它的返回值一定是布尔值。

【示例 2】下面列举一些特殊的运算数的逻辑非运算返回值。

```
alert( ! {});           // 返回 false。如果运算数是对象，则返回 false
alert( ! 0);            // 返回 true。如果运算数是 0，则返回 true
```

```
alert( ! (n = 5));          // 返回 false。如果运算数是非 0 的任何数字，则返回 false
alert( ! null);             // 返回 true。如果运算数是 null，则返回 true
alert( ! NaN);              // 返回 true。如果运算数是 NaN，则返回 false
alert( ! Infinity);         // 返回 false。如果运算数是 Infinity，则返回 false
alert( ! ( - Infinity));    //返回 false。如果运算数是-Infinity，则返回 false
alert( ! undefined);        //返回 true。如果运算数是 undefined，则返回 true，在早期浏览器中或发生错误
```

17.4.4 案例：逻辑运算训练

对于逻辑与（&&）和逻辑或（||）运算符来说，它们并不改变运算数的数据类型，同时也不会强制逻辑运算的结果是什么数据类型。它们具有如下特性：

- 在逻辑运算时，与和或运算都会把运算数视为布尔值，即使不是布尔值，也将对其进行转换，然后根据布尔值执行下一步的操作。
- 逻辑与（&&）和逻辑或（||）运算并非完整的执行所有运算数，它们可能仅执行第一个运算数，从而忽略第二个运算数。

【示例 1】在下面条件结构中，由于字符串变量 a 的逻辑值可以转换为 true，则逻辑或运算符在执行左侧的 a = "string"赋值表达式之后，就不再执行逻辑或运算符右侧的定义对象结构体。所以，最后在执行条件结构内的 alert(b.a);语句时，就会返回对象 b 没有定义的错误提示。

```
if(a = "string" || (b =     // 执行逻辑或操作
    {                       // 定义对象结构体
      a : "string"          // 定义对象的属性 a
    })
) alert(b.a);               // 调用对象 b 的属性 a
```

如果把其中的逻辑或运算符替换为逻辑与运算符，则当第一个运算数值可以转换为 true，将继续执行右侧的运算数，该运算数是一个复杂的结构体，定义了一个对象并赋值给变量 b。这样在条件结构中执行对象调用时，会显示字符串"string"。

```
if(a = "string" && (b =     // 执行逻辑与操作
    {                       // 定义对象结构体
      a : "string"          // 定义对象的属性 a
    })
) alert(b.a);               // 调用对象 b 的属性 a，返回字符串"string"
```

在下面结构中，由于 if 条件最终返回 false，所以不管对象 b 是否被定义，最后并没有执行调用 b 对象的属性 a 这个语句。

```
if(a = 0 && (b =            // 执行逻辑或操作
    {                       // 定义对象结构体
      a : "string"          // 定义对象的属性 a
    })
) alert(b.a);               // 调用对象 b 的属性 a，没有被执行
```

通过上面演示示例，可以看到逻辑与和逻辑或运算时，并没有改变运算数的数据类型，也没有改变这些表达式的值，返回值依然保持表达式的运算值，而不是被转换的布尔值。

【示例 2】逻辑与和逻辑或是两个相互补充的逻辑操作，结合它们你可以设计出很多结构复杂而又巧妙的的逻辑运算表达式。例如，下面结构是一个复杂的嵌套结构，它根据变量 a 的布尔值来判断是否执行一个循环体。

```
var a = b = 2;              // 定义并连续初始化
if(a){                      // 条件结构
  while(b ++ < 10){         // 循环结构
    alert(b ++);            // 循环执行语句
  }
}
```

对于这样一个复杂的循环结构，可以使用逻辑与和逻辑或运算符进行简化：

```
var a = b = 2;                    // 定义并连续初始化
while(a && b ++ < 10) alert(b ++ );          // 循环体。逻辑与运算符合并的多条件表达式
```

如果把上面的逻辑运算表达式转换为如下嵌套结构就不对了：

```
while(b ++ < 10){              // 先执行循环
   if(a){                      // 再判断条件
     alert(b ++);
   }
}
```

因为在 a && b ++ < 10 这个逻辑与表达式中可能会存在这样一种情况：如果逻辑与运算符左侧的运算数返回值为 false，那么就不再继续执行逻辑与运算符右侧的运算数了。