Scala 访问修饰符 Scala IF…ELSE 语句

Scala 运算符

一个运算符是一个符号，用于告诉编译器来执行指定的数学运算和逻辑运算。

Scala 含有丰富的内置运算符，包括以下几种类型：

- 算术运算符
- 关系运算符
- 逻辑运算符
- 位运算符
- 赋值运算符

接下来我们将为大家详细介绍以上各种运算符的应用。

算术运算符

下表列出了 Scala 支持的算术运算符。

假定变量 A 为 10，B 为 20：

运算符	描述	实例
+	加号	A + B 运算结果为 30
-	减号	A - B 运算结果为 -10
*	乘号	A * B 运算结果为 200
/	除号	B / A 运算结果为 2
%	取余	B % A 运算结果为 0

实例

```
object Test {
   def main(args: Array[String]) {
      var a = 10;
      var b = 20;
      var c = 25;
      var d = 25;
      println("a + b = " + (a + b) );
      println("a - b = " + (a - b) );
      println("a * b = " + (a * b) );
      println("b / a = " + (b / a) );
```

```
      println("b % a = " + (b % a) );
      println("c % a = " + (c % a) );

   }
}
```

运行实例 »

执行以上代码，输出结果为：

```
$ scalac Test.scala
$ scala Test
a + b = 30
a - b = -10
a * b = 200
b / a = 2
b % a = 0
c % a = 5
```

关系运算符

下表列出了 Scala 支持的关系运算符。

假定变量 A 为 10，B 为 20：

运算符	描述	实例
==	等于	(A == B) 运算结果为 false
!=	不等于	(A != B) 运算结果为 true
>	大于	(A > B) 运算结果为 false
<	小于	(A < B) 运算结果为 true
>=	大于等于	(A >= B) 运算结果为 false
<=	小于等于	(A <= B) 运算结果为 true

实例

```
object Test {
   def main(args: Array[String]) {
      var a = 10;
      var b = 20;
      println("a == b = " + (a == b) );
      println("a != b = " + (a != b) );
      println("a > b = " + (a > b) );
```

```
        println("a < b = " + (a < b) );
        println("b >= a = " + (b >= a) );
        println("b <= a = " + (b <= a) );
    }
}
```

执行以上代码，输出结果为：

```
$ scalac Test.scala
$ scala Test
a == b = false
a != b = true
a > b = false
a < b = true
b >= a = true
b <= a = false
```

逻辑运算符

下表列出了 Scala 支持的逻辑运算符。

假定变量 A 为 1，B 为 0：

运算符	描述	实例
&&	逻辑与	(A && B) 运算结果为 false
||	逻辑或	(A || B) 运算结果为 true
!	逻辑非	!(A && B) 运算结果为 true

实例

```
object Test {
   def main(args: Array[String]) {
      var a = true;
      var b = false;

      println("a && b = " + (a&&b) );

      println("a || b = " + (a||b) );

      println("!(a && b) = " + !(a && b) );
   }
}
```

执行以上代码，输出结果为：

```
$ scalac Test.scala
$ scala Test
a && b = false
a || b = true
!(a && b) = true
```

位运算符

位运算符用来对二进制位进行操作，~,&,|,^分别为取反，按位与与，按位与或，按位与异或运算，如下表实例：

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

如果指定 A = 60; 及 B = 13; 两个变量对应的二进制为：

```
A = 0011 1100

B = 0000 1101

-------位运算----------

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A  = 1100 0011
```

Scala 中的按位运算法则如下：

运算符	描述	实例
&	按位与运算符	(a & b) 输出结果 12 ，二进制解释： 0000 1100
|	按位或运算符	(a | b) 输出结果 61 ，二进制解释： 0011 1101
^	按位异或运算符	(a ^ b) 输出结果 49 ，二进制解释： 0011 0001
~	按位取反运算符	(~a) 输出结果 -61 ，二进制解释： 1100 0011， 在一个有符号二进制数的补码形式。

<<	左移动运算符	a << 2 输出结果 240 ，二进制解释： 1111 0000
>>	右移动运算符	a >> 2 输出结果 15 ，二进制解释： 0000 1111
>>>	无符号右移	A >>>2 输出结果 15, 二进制解释: 0000 1111

实例

```
object Test {
   def main(args: Array[String]) {
      var a = 60;           /* 60 = 0011 1100 */
      var b = 13;           /* 13 = 0000 1101 */
      var c = 0;

      c = a & b;            /* 12 = 0000 1100 */
      println("a & b = " + c );

      c = a | b;            /* 61 = 0011 1101 */
      println("a | b = " + c );

      c = a ^ b;            /* 49 = 0011 0001 */
      println("a ^ b = " + c );

      c = ~a;               /* -61 = 1100 0011 */
      println("~a = " + c );

      c = a << 2;           /* 240 = 1111 0000 */
      println("a << 2 = " + c );

      c = a >> 2;           /* 15 = 1111 */
      println("a >> 2  = " + c );

      c = a >>> 2;          /* 15 = 0000 1111 */
      println("a >>> 2 = " + c );
   }
}
```

执行以上代码，输出结果为：

```
$ scalac Test.scala
$ scala Test
a & b = 12
a | b = 61
a ^ b = 49
~a = -61
a << 2 = 240
```

```
a >> 2  = 15
a >>> 2 = 15
```

赋值运算符

以下列出了 Scala 语言支持的赋值运算符:

运算符	描述	实例
=	简单的赋值运算，指定右边操作数赋值给左边的操作数。	C = A + B 将 A + B 的运算结果赋值给 C
+=	相加后再赋值，将左右两边的操作数相加后再赋值给左边的操作数。	C += A 相当于 C = C + A
-=	相减后再赋值，将左右两边的操作数相减后再赋值给左边的操作数。	C -= A 相当于 C = C - A
*=	相乘后再赋值，将左右两边的操作数相乘后再赋值给左边的操作数。	C *= A 相当于 C = C * A
/=	相除后再赋值，将左右两边的操作数相除后再赋值给左边的操作数。	C /= A 相当于 C = C / A
%=	求余后再赋值，将左右两边的操作数求余后再赋值给左边的操作数。	C %= A is equivalent to C = C % A
<<=	按位左移后再赋值	C <<= 2 相当于 C = C << 2
>>=	按位右移后再赋值	C >>= 2 相当于 C = C >> 2
&=	按位与运算后赋值	C &= 2 相当于 C = C & 2
^=	按位异或运算符后再赋值	C ^= 2 相当于 C = C ^ 2
|=	按位或运算后再赋值	C |= 2 相当于 C = C | 2

实例

```
object Test {
   def main(args: Array[String]) {
      var a = 10;
      var b = 20;
      var c = 0;

      c = a + b;
      println("c = a + b  = " + c );

      c += a ;
      println("c += a  = " + c );
```

```
      c -= a ;
      println("c -= a = " + c );

      c *= a ;
      println("c *= a = " + c );

      a = 10;
      c = 15;
      c /= a ;
      println("c /= a  = " + c );

      a = 10;
      c = 15;
      c %= a ;
      println("c %= a  = " + c );

      c <<= 2 ;
      println("c <<= 2  = " + c );

      c >>= 2 ;
      println("c >>= 2  = " + c );

      c >>= 2 ;
      println("c >>= a  = " + c );

      c &= a ;
      println("c &= 2  = " + c );

      c ^= a ;
      println("c ^= a  = " + c );

      c |= a ;
      println("c |= a  = " + c );
   }
}
```

执行以上代码，输出结果为：

```
$ scalac Test.scala
$ scala Test
c = a + b  = 30
c += a  = 40
c -= a = 30
c *= a = 300
c /= a  = 1
c %= a  = 5
c <<= 2  = 20
c >>= 2  = 5
```

```
c >>= a  = 1
c &= 2  = 0
c ^= a  = 10
c |= a  = 10
```

运算符优先级取决于所属的运算符组，它会影响算式的的计算。

实例： x = 7 + 3 * 2; 这里， x 计算结果为 13, 而不是 20，因为乘法（ * ）高于加法（ + ），所以它先计算 3*2 再加上 7。

查看以下表格，优先级从上到下依次递减，最上面具有最高的优先级，逗号操作符具有最低的优先级。

类别	运算符	关联性
1	() []	左到右
2	! ~	右到左
3	* / %	左到右
4	+ -	左到右
5	>> >>> <<	左到右
6	> >= < <=	左到右
7	== !=	左到右
8	&	左到右
9	^	左到右
10	|	左到右
11	&&	左到右
12	||	左到右
13	= += -= *= /= %= >>= <<= &= ^= |=	右到左
14	,	左到右

← Scala 访问修饰符 Scala IF…ELSE 语句 →