Day03 - 变量和运算符

今日学习内容

- 变量的定义和使用
- 基本数据类型的转换
- 算术运算符
- 赋值运算符
- 比较运算符
- 三元运算符
- 逻辑运算符

今日学习目标

- 必须掌握变量的定义和赋值、使用
- 了解什么是表达式概念
- 掌握基本数据类型的自动转换
- 掌握基本数据类型的自动提升
- 掌握基本数据类型的强制转换
- 掌握算术运算符的使用
- 了解什么是前置++和后置++的区别
- 掌握赋值运算符的使用，以及它的底层含义
- 掌握比较运算符的使用
- 必须掌握三元运算符的语法和使用
- 掌握逻辑运算符的使用（常用 && 、|| 、！）
- 了解与（&）和短路与（&&）的区别，记住结论使用&&即可
- 了解运算符的优先级

第二章 Java入门基础 - 变量和运算符

2、变量（重点）

通过一张不完整的房屋租赁合同，引出变量 (variable)。

案例：张三需要租赁李四的房屋，租赁合同如下：

张三、*李四*双方就下列房屋的租赁达成如下协议：
第一条 房屋押金
张三、*李四*双方自本合同签订之日起，由*张三*支付*李四*一个月房租的金额作为押金。
第二条 租赁期满。
1、租赁期满后，如*张三*要求继续租赁，*李四*则优先同意继续租赁；
2、租赁期满后，如*李四*未明确表示不续租的，则视为同意*张三*继续承租；
3、租赁期限内，如*张三*明确表示不租的，应提前一个月告知*李四*，*李四*应退还*张三*已支付的租房款及押金。

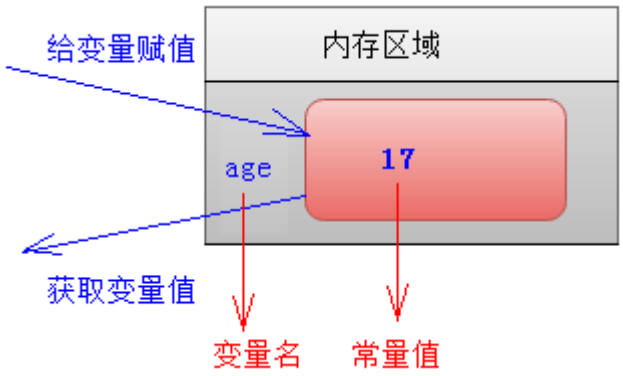
上述合同，相当不正规，因为正规的合同上，租客和房东都是有变动的，不能写死，在整个合同中应该是使用甲方来表示房东，乙方来表示租客，只会在最后的时候签名甲方是谁，乙方是谁。

甲、乙方双方就下列房屋的租赁达成如下协议：
第一条 房屋押金
甲、乙双方自本合同签订之日起，由乙方支付甲方一个月房租的金额作为押金。
第二条 租赁期满。
1、租赁期满后，如乙方要求继续租赁，甲方则优先同意继续租赁；
2、租赁期满后，如甲方未明确表示不续租的，则视为同意乙方继续承租；
3、租赁期限内，如乙方明确表示不租的，应提前一个月告知甲方，甲方应退还乙方已支付的租房款及押金。
甲方（签章）：李四　　乙方（签章）：张三

2.1 变量概述（了解）

变量指在程序运行过程中，值可以发生变化的量。

变量表示一个存储空间，可用来存放某一类型的常量，没有固定值，并可以重复使用，换句话说，变量是内存中一块区域，可以往该区域存储数据，修改里面的数据，也可以获取里面的数据。



变量定义的语法

```
数据类型 变量名 = 初始值;
```

变量的特点：

- 占据着内存中的某一块存储区域
- 该区域有自己的名称（变量名）和类型（数据类型）
- 可以被重复使用
- 该区域的数据可以在同一类型范围内不断变化

2.2 变量定义和赋值（重点）

需求：定义一个int类型变量，存储一个学生的年龄，并赋初始值（17岁）。

```
public class VarDemo {
    public static void main(String[] args) {
        // 方式一，先变量，后赋值，再使用
        // step 1:定义变量。语法:数据类型 变量名;
        int age;
        // step 2:给变量赋初始值。变量名 = 常量值;
        age = 17;
        // 修改age变量的值为22
        age = 22;
        // step 3:使用定义的变量
        System.out.println(age);
```

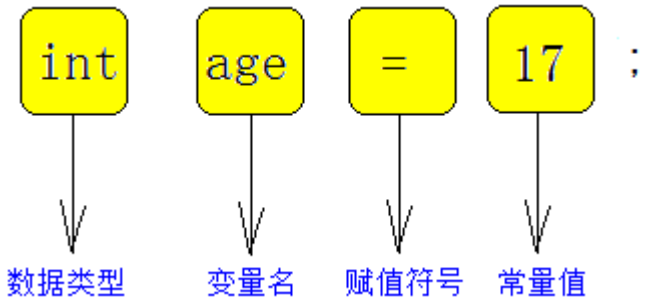
```
        // 方式二，在声明时同时赋值（推荐）
        // 数据类型 变量名 = 初始化值;
        // 定义一个String类型的变量，初始值为wolf
        String name = "wolf";
    }
}
```

使用变量注意：

- 变量必须先声明，并且初始化后才能使用
- 定义变量必须有数据类型
- 变量从开始定义到所在的花括号结束之内可以使用，离开花括号就不能使用了
- 同一{}内，变量名不能重复定义

记： 语法格式

- int：表示类型，这里可以根据需求写需要的类型
- age：变量名，和我们的姓名一样理解， 没有为什么
- =：赋值运算符，后面会讲，意思是将右边的值存入左边的变量
- 17：一个整数常量的值，也可以是其他整数常量的值，但必须保证数值在int类型的范围。

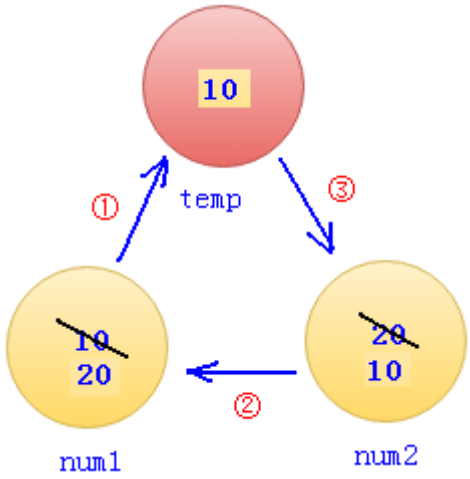


需求1：定义每一种数据类型的变量

```
public class VarDemo2 {
    public static void main(String[] args) {
        // int类型变量
        int i = 20;
        System.out.println(i);

        // long类型变量,使用L后缀
        long l = 20L;
        System.out.println(l);

        // float类型变量,使用F后缀
        float f = 3.14F;
        System.out.println(f);

        // double类型变量
        double d = 3.14;
        System.out.println(d);

        // char类型变量
        char c = 'A';
        System.out.println(c);

        // boolean类型变量
        boolean bool = true;
```

```
        System.out.println(bool);

        // String类型变量
        String str = "你好";
        System.out.println(str);
    }
}
```

综合案例：交换两个相同类型变量的值



1、把num1的值存储到临时变量temp中去

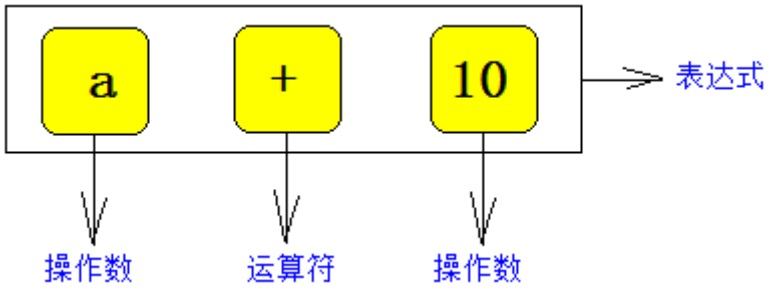
2、把num2的值赋给num1变量

3、把temp存储的值赋给num2变量

```
public class ChangVarDemo {
    public static void main(String[] args) {
      int num1 = 10;
      int num2 = 20;
      System.out.println(num1);
      System.out.println(num2);

      // --------------------------------
      // 交换过程
      int temp = num1;
      num1 = num2;
      num2 = temp;

      //--------------------------------
      System.out.println(num1);
      System.out.println(num2);
    }
}
```

3、表达式（先知道概念）

表达式（expression），是由数字、常量、变量、运算符、括号等组合以能求得结果的式子，表达式在开发过程中用于计算结果。



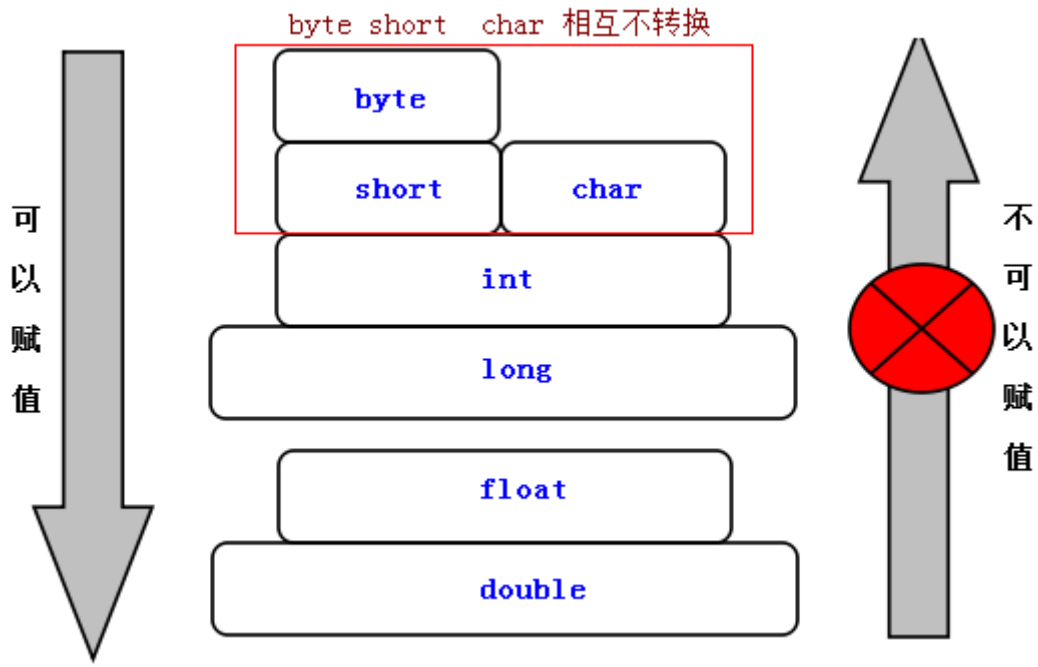
表达式举例（下列a、b、x、y、z都表示变量）。

- a + b
- 3.14 + a
- (x + y) * z + 100

4、基本数据类型转换（掌握）

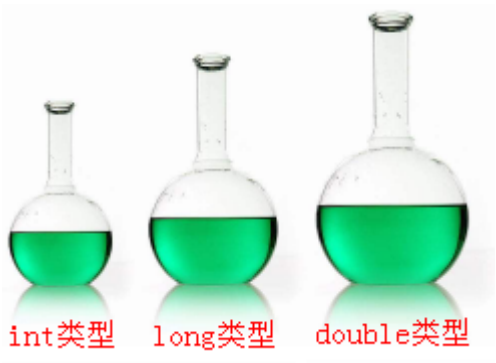
在8大基本数据类型中，boolean不属于数值类型 (主要用于逻辑判断)，所以不参与转换。

其他类型的转换规则如下图。一般的，byte、short、char三种类型相互之间一般不参与。



按照转换方式，有两种（注意：boolean类型不参与类型转换）：

- 自动类型转换：范围小的数据类型**直接转换**成范围大的数据类型 (小 => 大)。
- 强制类型转换：范围大的数据类型**强制转换**成范围小的数据类型 (大 => 小)。



问题：三个大小不同容器，能相互把盛装的水倒给对方吗?

科普了解：
float占4个字节为什么比long占8个字节大？
---> 因为底层的实现方式不同
float类型的32位并不是简单直接表示大小，而是按照一定标准分配的。
第1位，符号位，即S
接下来8位，指数域，即E。
剩下23位，小数域，即M，取值范围为[1 ,2) 或[0 , 1)
然后按照公式：V = (-1) ^ s * M * 2 ^ E
也就是说小数float在内存中的32位不是简单地转换为十进制，而是通过公式来计算而来，通过这个公式，虽然只有4个字节，但浮点数最大值要比长整型的范围要大。

4.1 自动类型转换（掌握）

自动类型转换，也称为"隐式类型转换"，就是把范围小的数据类型直接转换成范围大的数据类型。

转换规则：byte、short、char—>int—>long—>float—>double

语法格式

```
范围大的数据类型 变量 = 范围小的数据类型常量值/变量;
```

语法举例：

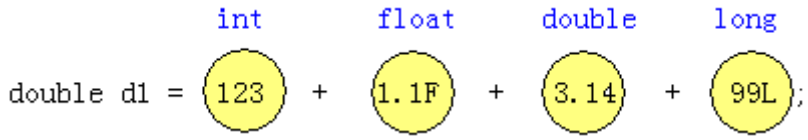
```
public class TypeConvertDemo {
    public static void main(String[] args){
        // 自动类型转换 ：范围小的数据类型可以直接转换为范围大的数据类型
        int intNum1 = 10;
        long longNum2 = intNum1;

        long longNum3 = 100L;
        float floatNum4 = longNum3;

        float floatNum5 = 3.14f;
        double doubleNum6 = floatNum5;
    }
}
```

4.2 自动类型提升（掌握）

当一个算术表达式中，包含多个基本数据类型的常量或变量（boolean除外）时，整个算术表达式的结果类型将在出现自动提升，其规则是：

- **所有的byte、short、char类型被自动提升到int类型，再参与运算**
- **整个表达式的最终结果类型，被提升到表达式中范围最大的那个类型**

```
System.out.println('a' + 1); // 98

byte b = 22;
b = b + 11;// 编译出错，此时结果类型应该是int

double d1 = 123 + 1.1F + 3.14 + 99L ;
```



结论：算数表达式结果的类型就是其中范围最大的数据类型。

```
public class TypeConvertDemo2 {
    public static void main(String[] args){

        // 规则1:所有的byte、short、char类型被自动提升到int类型，再参与运算
        byte byteNum1 = 10;
        short shortNum2 = 20;
        int r = byteNum1 + shortNum2;

        // 规则2:整个表达式的最终结果类型，被提升到表达式中类型最高的类型
        float floatNum3 = 3.14f;
        double doubleNum4 = 1.0;
        double r2 = byteNum1 + shortNum2 + floatNum3 + doubleNum4;
    }
}
```

4.3 强制类型转换（掌握）

强制类型转换，也称为“显式类型转换”，就是把范围大的数据类型强制转换成范围小的数据类型。

语法格式：

```
范围小的数据类型  变量 = (范围小的数据类型)范围大的数据类型值;
```

注意：一般情况下不建议使用强转，因为强转有可能损失精度

```
public class TypeConvertDemo3 {
    public static void main(String[] args) {
        // 强制类型转换
        float floatNum1 = 3.14f;
        int intNum = (int)floatNum1;
        System.out.println(intNum);

        // 应用:根据消费金额计算vip积分(规则：一块钱积2分)
        float price = 998.88f;
        int vipScore = (int)price * 2;
        System.out.println(vipScore);

        // 思考题
        char c = 'a';
        int num = 1;
        char r = (char)(c + num);
        System.out.println(r);
    }
}
```

#14.40

5、运算符

对常量和变量进行操作的符号称为运算符(operator)。

常用运算符：

- 算术运算符
- 赋值运算符
- 比较运算符
- 逻辑运算符
- 三元运算符

5.1 算术运算符（掌握）

运算符	运算规则	范例	结果
+	正号	+3	3
+	加	2+3	5
+	连接字符串	“中”+“国”	“中国”
-	负号	int a=3;-a	-3
-	减	3-1	2
*	乘	2*3	6
/	除	5/2	2
%	取模	5%2	1
++	自增	int a=1;a++ / ++a	2
--	自减	int b=3; b-- / --b	2

用来四则运算的符号，和小学学习的加减乘除无异。

5.1.1 加减乘除余（了解）

```
public class ArithOperatorDemo {
    public static void main(String[] args) {
        // 算术运算符 +  -  * /
        int num1 = 10;
        int num2 = 2;
        int r1 = num1 + num2;
        System.out.println(r1);

        // 整除操作 /
        int num3 = 5;
        System.out.println(num3 / 2);
        System.out.println(num3 * 1.0 / 2);

        // + 作为字符串连接符
        int num4 = 40;
        String str = "num4=" + intNum3;
```

```
        System.out.println(str);

        // % 取模(理解为求余数)
        System.out.println(5 % 2);
        System.out.println(4 % 5);

        // 需求: 给定47天，问47天中有__月(30)__天
        int days = 47;
        // step 1:求47天中有几个月
        int month = days / 30;
        // step 2:求剩余的天数
        int day = days % 30;  // 47除以30后剩余的天数
    }
}
```

注意:

- 如果/两边的操作数都是整数,/ 的结果是整数；如果/两边的操作数有一个是小数,/ 的结果是小数
- 对于字符串而言，+符号表示连接操作，任何类型的数据和字符串相连接，结果都是字符串。

5.1.2 自增和自减（掌握）

自增：++，递增操作符，使变量值增加1，有前置和后置之分，只能操作变量。

自减：-- ，递减操作符，使变量值减去1，有前置和后置之分，只能操作变量。

自增和自减具体操作是一样的，仅仅是一个是加1，一个是减1而已，现在单讲++。

代码 result++和 ++result，结果都是result变量的值加1。

唯一的区别是:

- 前置（++result）：表示对result加1之后的结果进行运算
- 后置（result++）：表示对result变量加1之前的值（原始值）进行运算。

如果仅仅执行简单的递增操作（只写result++或++result），那么选用任意一个都可以。

```
public class ArithOperatorDemo2 {
    public static void main(String[] args) {
        // 自加
        // i++ / i-- : 遇到i++/i--,i先参与运算，运算后自增1/自减1
        // 情况1（必须掌握）:
        /*
        int i = 10;
        i++;
        System.out.println("i = " + i);
        */

        // 情况2（必须掌握）:
        int i = 10;
        int j;
        j = i++;
        System.out.println("i = " + i);
        System.out.println("j = " + j);
    }
}
```

比较权威的解释：

- ++a表示取a的地址，增加它的内容，然后把值放在寄存器中；
- a++表示取a的地址，把它的值装入寄存器，然后增加内存中的a的值；

如果不理解什么是寄存器，简单记住，都可以表示当前变量自身加1，区别是：

- **前置++：先增加后使用**
- **后置++：先使用后增加**

5.2 赋值运算符（掌握）

```
变量 = 表达式的值或者常量值
```

运算符	运算规则	范例	结果
=	赋值	int a=2	2
+=	加后赋值	int a=2，a+=2	4
-=	减后赋值	int a=2，a-=2	0
*=	乘后赋值	int a=2，a*=2	4
/=	整除后赋值	int a=2，a/=2	1
%=	取模后赋值	int a=2，a%=2	0

```
public class AssignOperatorDemo {
    public static void main(String[] args) {
        // 把常量17赋值给int类型的变量a
        int a = 17;
        System.out.println("a=" + a);

        // += 把左边和右边的数据进行运算，最后赋值左边变量
        a += 10;// 相当于a = a + 10
        System.out.println("a=" + a);

        // 注意情况: += 操作已经包含了强制类型转换
        short s = 5;
        s += 2; //底层相当于 s = (short) (s + 2)
        System.out.println("s=" + s);
    }
}
```

5.3 比较运算符（掌握）

用于比较变量或常量、表达式之间的大小关系，其结果是boolean类型（要么为true，要么为false）。

其操作格式为：

```
boolean  result  = 表达式A  比较运算符  表达式B;
```

运算符	运算规则	范例	结果
==	相等于	4==3	false
!=	不等于	4!=3	true
<	小于	4<3	false
>	大于	4>3	true
<=	小于等于	4<=3	false
>=	大于等于	4>=3	true

注意：>=符号，表示大于或者等于。

```
public class CompareOperatorDemo {
    public static void main(String[] args) {
        // 直接操作常量
        System.out.println(10 > 5);//true
        System.out.println(10 >= 5);//true
        System.out.println(10 >= 10);//true
        System.out.println(10 < 5);//false
        System.out.println(10 <= 5);//false
        System.out.println(10 <= 10);//true
        System.out.println(10 == 10);//true
        System.out.println(10 != 10);//false

        // 使用变量操作
        int a = 10;
        int b = 5;
        boolean result = a > b;
        System.out.println(result);//true
    }
}
```

5.4 三元运算符（掌握）

三元运算符，表示有三个元素参与的运算符，所以又称为三目运算符，其语义表示if-else（如果什么情况就做什么，否则做什么）。 如果...那么...否则...

语法格式：

```
数据类型 变量 = boolean表达式 ?  结果A : 结果B;
```

- 如果boolean表达式结果：
  - 为true，则三元运算符的结果是结果A；
  - 为false，则三元运算符的结果是结果B；

注：三元运算符**必须定义变量接受**运算的结果，否则报错

三元运算符结果的类型由结果A和结果B来决定的，结果A和结果B的类型是相同的。

需求1：判断一个数99是不是偶数

```
public class TernaryOperatorDemo1 {
    public static void main(String[] args) {
        int a = 99;
        String result = a % 2 == 0 ? "偶数" : "奇数";
        System.out.println(result);
    }
}
```

需求2：求99和20两个数中的最大值

```
public class TernaryOperatorDemo2{
    public static void main(String[] args) {
        int a = 99;
        int b = 20;
        int result = a >= b ? a : b;
        System.out.println("最大值："+result);
    }
}
```

需求3：一共55条数据，每页10条数据，一共分多少页


```
public class TernaryOperatorDemo3{
    public static void main(String[] args) {
        int totalCount = 55;
        int pageSize = 10;
        int totalPage = totalCount % pageSize == 0
                    ? totalCount / pageSize
                    : totalCount / pageSize + 1;
        System.out.println(totalPage);
    }
}
```

#16.50

5.5 逻辑运算符（掌握）

逻辑运算符用于连接两个boolean表达式，结果也是boolean类型的。

语法格式为：

```
boolean  result = boolean表达式A  逻辑运算符  boolean表达式B;
```

运算规则如下：

运算符	运算规则	范例	结果
&	与	false&true	false
|	或	false|true	true
^	异或	true^flase	true
!	非	!true	false
&&	短路与	false&&true	false
||	短路或	false||true	true

规律：

- 非：取反，！true则false,！false则true
- 与：有false则false
- 或：有true则true
- 异或：^ 相同则false,不同则true

```
& 与运算，可以理解为 "并，并且"
true  & true  => true
true  & false => false
false & true  => false
false & false => false
总结 ： & 运算，只要两边的表达式有一个为false，结果就为false

&& 短路与
&& 运算，只要两边的表达式有一个为false，结果就为false，如果第一个表达式为false，后续表达式不再运算；

| 或运算，可以理解为 "或，或者"
true  | true  => true
true  | false => true
false | true  => true
false | false => false
总结 ： | 运算，只要两边的表达式有一个为true，结果就为true

|| 短路或
|| 短路或运算，只要两边的表达式有一个为true，结果就为true，如果第一个表达式为true，后续表达式不再运算；

! 非运算，可以理解为 取反
!true = false
!false = true
```

5.5.1 基本使用（掌握）

```
public class LogicalOperatorDemo1 {
    public static void main(String[] args) {
        int a = 10;
        int b = 20;
        int c = 30;
        //与操作
        System.out.println((a > b) & (a > c));// false  &   false
        System.out.println((a > b) & (a < c)); // false &   true
        System.out.println((a < b) & (a > c)); // true  &   false
        System.out.println((a < b) & (a < c)); // true  &   true

        //或操作
        System.out.println((a > b) | (a > c));// false  |   false
        System.out.println((a > b) | (a < c)); // false |   true
        System.out.println((a < b) | (a > c)); // true  |   false
        System.out.println((a < b) | (a < c)); // true  |   true

        //相反操作
        System.out.println((a > b)); // false
        System.out.println(!(a > b)); // !false
        System.out.println(!!(a > b)); // !!false
    }
}
```

5.5.2 &和&&的区别（掌握）

&： &左边表达式无论真假，&右边表达式都进行运算；

&& ： 如果&&左边表达式为真，&&右边表达式参与运算；如果&&左边表达式为假，&&右边表达式不参与运算，故称短路与。

| 和 || 的区别同理，对于||，左边为真，右边不参与运算。

```
public class LogicOperatorDemo2 {
    public static void main(String[] args) {
        // 逻辑运算符
        // 短路与 &&
        int m = 10;
        int n = 20;
        // boolean r1 = false && true;
        boolean r1 = (m > n) && (++m > 1);
        System.out.println("r1 = " + r1);
        System.out.println("m = " + m);

        // 短路或 ||
        boolean r2 = (m > n) | (m < 1);
        System.out.println("r2 = " + r2);
    }
}
```

上述代码，一行一行的测试，测试完，注释该行代码。

5.6 运算优先级（了解）

表达式的运算都是有优先级的，基本上和数学中的优先级类似，

```
public class PriorityDemo {
    public static void main(String[] args) {
        int m = 10;
        int n = 20;
        // 给定一个表达式,如果让你计算结果，你首先想到的是？
        m++ > n * 2 && m + n * 2 > 100;

        // 求m+n的和乘以3的结果
        int r = (m + n) * 3;
    }
}
```

这里需要注意的是，赋值符号。注意：赋值符号最后运算的，并且是从右向左运算的。

优先级	运算符	类	结合性
1	()	括号运算符	由左至右
1	[]	方括号运算符	由左至右
2	!、+（正号）、-（负号）	一元运算符	由右至左
2	~	位逻辑运算符	由右至左
2	++、--	递增与递减运算符	由右至左
3	*、/、%	算术运算符	由左至右
4	+、-	算术运算符	由左至右
5	<<、>>	位左移、右移运算符	由左至右
6	>、>=、<、<=	关系运算符	由左至右
7	==、!=	关系运算符	由左至右
8	&（位运算符 AND）	位逻辑运算符	由左至右
9	^（位运算符号 XOR）	位逻辑运算符	由左至右
10	|（位运算符号 OR）	位逻辑运算符	由左至右
11	&&	逻辑运算符	由左至右
12	||	逻辑运算符	由左至右
13	?:	条件运算符	由右至左
14	=	赋值运算符	由右至左

结论：

- ( ) 的优先级最高
- 赋值运算符优先级最低