# day03

## 运算符

基本算数运算符

自增自减运算符

赋值运算符

比较运算符

逻辑运算符

位移运算符(了解)

三元运算

### 基本算数运算符

1. + - \* / %

a : + 加法运算

1. 直接作用在数值之前, 表示正数 +3
2. 做数值之间的加法运算, 3 + 2 = 5
3. 做字符串加法, 字符串类型与任意数据类型相加, 只表示字符串拼接, 结果仍然是字符串类型

b : - 减法运算

1. 直接作用在数值之前, 表示负数 -3
2. 做数值之间的加法运算, 3 - 2 = 1

c : \* 乘法运算

1. 做数值之间的乘法运算 3 \* 2 = 6

d : / 除法运算

1. 如果/符号两端的数据都是整数类型, 那么相除的结果也是整数, 对于小数部分全部舍掉 5 / 3 = 1
2. 如果/符号两端有其中一端或者两端是浮点类型, 那么结果就可以带有小数

5.0 / 3 = 1.677777...

e : % 取模运算(取余运算) : 两数相除, 获取到余数

%取模运算使用场景, 通常可以进行数据筛选操作

7 % 3 = 7 / 3 = 2...1 取模结果就是1

10 % 5 = 10 / 5 = 2...0 获取结果就是0

%取模运算符注意 : 取模运算也带有符号位(正负), 取模运算结果的符号位与 %前面的被除数符号位一致

需求 : 定义出一个任意正整数, 判断这个数是奇数还是偶数

分析 : 偶数, 可以被2整除数, 数 / 2 除法, 没有余数,整除

数 % 2 与0数值进行比较大小, 结果就是0 ,证明数是偶数

代码

|  |
| --- |
| package com.ujiuye.variable;  public class Demo01\_基本算数运算符 {  public static void main(String[] args) {  // +  System.out.println(+3);// 3  System.out.println(3 + 2);// 5  System.out.println("6" + 5);// "65"    // -  System.out.println(-3);// -3  System.out.println(-3 - 2);// -5    // \*  System.out.println(3 \* 2);// 6    // /  System.out.println(5 / 3);// 1  System.out.println(5.0 / 3);// 1.6666666666666667    // %  // 7 % 3 = 7 / 3 = 2...1 取模结果就是1  System.out.println(7 % 3);// 1  // 10 % 5 = 10 / 5 = 2...0 获取结果就是0  System.out.println(10 % 5);// 0  // 取模运算结果的符号位与 %前面的被除数符号位一致  System.out.println(-7 % 3);// -1  System.out.println(7 % -3);// 1  System.out.println(-7 / 3);// -7 / 3 = -2...-1  System.out.println(7 / -3);// 7 / -3 = -2...1  }  } |

### 自增自减运算符

1. ++ : 自增运算符,表示每次让变量值+1

-- : 自减运算符,表示每次让变量值-1

自增自减运算符, 只能够作用在变量之上

1. ++或者--运算机制: (以下文字描述中以++为例, --运算过程与++一致)

int a = 10;

a++; ++a;

1. 如果a++或者++a, 作为独立表达式语句存在, 那么++写在变量前或者变量后,效果一致, 都是让变量a本身的值+1, 同理--
2. 如果++作用在a变量之上, a变量本身不是独立语句, a作为表达式一部分(a变量在自增同时还要与其他数据进行运算), ++作用在变量之前或者作用在变量之后有区别

a : ++作用在变量之前, 先进行变量自增运算, 然后参与其他运算

b : ++作用在变量之后, 先使用变量的原值进行运算, 运算完毕, 再进行变量本身自增运算

代码

|  |
| --- |
| package com.ujiuye.cal;  public class Demo01\_自增自减运算符 {  public static void main(String[] args) {  // 1. ++或者--作用在某个变量之上, 这个变量就是一个独立语句, 变量不参与其他运算  int a = 10;  a++;  System.out.println(a);// 11  ++a;  System.out.println(a);// 12    int b = 5;  b--;  System.out.println(b);// 4  --b;  System.out.println(b);// 3    // 2. ++作用在变量之前, 且变量参与运算, 先进行变量自增运算, 然后参与其他数据运算  // 运算过程 : 1) ++a a = 13 2) a + b = 13 + 3 = 16  int w = ++a + b;  System.out.println(w);// 16  System.out.println(a);// 13    // --作用在变量之后, 且变量参与运算, 先使用变量a原值参与运算, 再进行a值--操作  // 运算过程 : 1) 13 + 3 = 16(赋值给y) 2) a-- a = 12  int y = a-- + b;  System.out.println(y);// 16  System.out.println(a);// 12    System.out.println("---------------");    // a = 12 b = 3  // 1) 12 + 3 = 15(赋值result) 2) a++ a = 13  int result = a++ + b;// a++ + b;  System.out.println(result);// 15  System.out.println(a);// 13  System.out.println(b);// 3  }  } |

### 赋值运算符

1. 基本赋值运算符 = : 表示赋值, 将等号右边的数据,赋值给等号左边的变量
2. 扩展赋值运算符: += -= \*= /= %=

所有扩展赋值运算符运算原理一样

a : 先进行等号两边数据运算 b : 将a步骤的运算结果赋值给等号左边变量

举例 : int i = 10;

i += 5; // 1) i + 5 = 15 2) i = (int)15 相当于 : i = (int)(i + 5);

代码

|  |
| --- |
| package com.ujiuye.cal;  public class Demo02\_赋值运算符 {  public static void main(String[] args) {  int i = 10;  // 1) i + 5 = 15 2) i = 15 类似于 : i = i + 5;  i += 5;  System.out.println(i);// 15    // 1) i % 6 = 15 / 6 = 2...3 结果为3  // 2) i = 3;  i %= 6;  System.out.println(i);// 3    /\*short s = 5;  s = s + 1;\*/    short ss = 5;  // ss = (short)(ss + 1);  ss += 1;  System.out.println(ss);// 6    int i2 = 15;  // i2 + 6.2 = 15 + 6.2 = 21.2(double)  // i2 = (int)(i2 + 6.2);  i2 += 6.2;  System.out.println(i2);// 21  }  } |

### 比较运算符

1. 比较运算符 : 用于比较数据是否相等以及数据之间的大小关系
2. 比较运算符 : 运算符比较结果是boolean类型真假值
3. 比较运算符:
4. == : 比较符号两端数据是否相等, 如果相等, 返回true, 如果不相等,返回false
5. != : ! 在运算符中, 表示取反概念, != 与==结果正好相反, 比较符号两端数据是否不相等, 不相等返回true , 相等返回false

注意 : == 和 != 可以比较任意数据类型

1. > >= < <= : 只能进行数值的比较
2. 注意 : 比较运算符不能做连续比较判断

定义出一个整数变量i , 判断i是不是5到10之间的整数(分析 : 验证i值大于等于5同时小于等于10数据范围即可)

代码

|  |
| --- |
| package com.ujiuye.cal;  public class Demo03\_比较运算符 {  public static void main(String[] args) {  int i = 10;  int j = 20;  boolean boo = i == j;  System.out.println(boo);// false    boolean boo1 = i != j;  System.out.println(boo1);// true    System.out.println(i >= j);// false    // 定义出一个整数变量i , 判断i是不是5到10之间的整数(分析 : 验证i值大于等于5同时小于等于10数据范围即可)  //The operator <= is undefined for the argument type(s) boolean, int  // 运算符 <= 没有定义过 boolean类型与int类型之间比较方式  // 1) 5 <= i true  // 2) true <= 10  // boolean boo2 = 5 <= i <= 10;  }  } |

### 逻辑运算符

1. 逻辑运算符 : 在代码场景中, 例如 : 与(同时,并且), 或(或者), 非(取反), 属于逻辑运算

举例 :

1. 模拟登录界面, 输入用户名和密码, 验证 : 用户名输入正确 同时 密码输入正确, 登录成功
2. 有功能界面要求输入客户信息, 有选项要求客户输入的电话号码, 手机号码, 家庭座机, 紧急联系人号码..., 有时候,这些电话号码中, 只需要输入一个或者多个即可
3. 取反通常在做数据验证方面比较多, 如果输入用户名错的取反 对应正确用户名
4. 逻辑运算符的使用: 逻辑运算符操作的数据都是boolean类型数据, 逻辑运算符结果也是boolean类型

&& : 与运算, 短路与, 如果需要多个条件同时满足(符合要求), 可以使用&&运算连接多个条件, 只有&&符号两端的数据结果都为true, 那么&&结果才为true

true && true true

true && false false

false && true false

false && false false

定义出一个整数变量i , 判断i是不是5到10之间的整数(分析 : 验证i值大于等于5同时小于等于10数据范围即可)

|| : 或运算, 短路或, 如果多个条件中, 有一个或者多个满足,就可以成立的场景, 可以使用||运算连接多个条件, 只要有一个条件为true,结果就是true;

true || true true

true || false true

false || true true

false || false false

! : 取反

!true false

!false true

1. 注意 :

&& 短路与, || 短路或, 两种逻辑运算符在实际运算中, 具有高效运行原理

1. && : &&运算符左边的表达式(第一个表达式)结果为false , 那么右边表达式不管是什么, 一律不再运行, 结果返回false假结果, 因为可能有部分表达式不运行, 因此执行效率相对比较高
2. || : ||运算符左边表达式(第一个表达式)结果为true , 那么右边表达式不管是什么, 一律不再运行, 结果返回true真结果, 因为可能有部分表达式不运行, 因此执行效率相对比较高

结论 : 如果符号第一个表达式可以解决&&和||的结果, 那么第二个表达式就不会运行,这就是短路的效果

代码

|  |
| --- |
| package com.ujiuye.cal;  public class Demo04\_逻辑运算符 {  public static void main(String[] args) {  int i = 12;  boolean boo = i >= 5 && i <= 10;  System.out.println(boo);// false    boolean boo1 = i >= 5 || i <= 10;  System.out.println(boo1);// true    System.out.println(!true);//false  System.out.println(!boo);// true    System.out.println("----------------");    // && 运算高效原理  // 因为第一个条件 i <= 10 为false , 因此第二个表达式条件 ++i >= 10 根本没有计算和比较  // 直接返回false结果  boolean boo2 = i <= 10 && ++i >= 10;  System.out.println(boo2);// false  System.out.println(i);// 12    // ||运算高效原理  // 第一个条件表达式 true , 第二个表达式, 不需要运算个比较, 直接返回true结果  boolean boo3 = i >= 5 || --i >= 5;  System.out.println(boo3);// true  System.out.println(i);// 12  }  } |

### 位移运算(了解)

1. 位移运算 : 数据在内存中都是以二进制方式进行存储, 计算机底层在进行运算, 就是以二进制方式进行运算, 位移运算通过移动数据在内存中二进制位数, 从而达到数据运算目的
2. 位移运算符 : 1) 位移运算只能使用在整数上 2) 每次计算数据都只能是2的倍数 3) 运算效率比较高, 直接操作数据底层对应的二进制位

<< : 左位移运算, 每移动一次,数值就会变成原来的2倍, 移动几位以运算符后面的数据为准

>> : 右位移运算, 每移动一次,数值就会变成原来的1/2, 移动几位以运算符后面的数据为准

>>> : 无符号右移, 只能针对于正整数进行操作, 效率比>>效率更高, 不考虑符号位

代码

|  |
| --- |
| package com.ujiuye.cal;  public class Demo05\_位移运算 {  public static void main(String[] args) {  int i = 20;  // 01001000  System.out.println(i << 1);// 40  System.out.println(i << 2);// 80    int y = 2;// 10  // 00000010  // 00000000  System.out.println(y >> 1);// 1  System.out.println(y >> 2);// 0    System.out.println(7 >>> 1);// 3  }  } |

### 三元运算

1. 元 : 表达式
2. 三元 : 可以直接操作三个表达式的运算符
3. 三元运算语法结构:

表达式1 ? 表达式2 : 表达式3;

1. 解释 :
2. 表达式1 : 必须是boolean类型结果的表达式
3. 表达式2和表达式3 : 可以是任意结果表达式
4. 三元运算运行过程:
5. 先计算出表达式1的boolean结果, true, false
6. ? : 表示询问表达式1结果是真还是假
7. 如果表达式1结果为true, 那么执行表达式2, 如果表达1结果为false, 那么执行表达式3

注意 : 三元运算使用场景就是二选一场景

需求1 : 定义出一个任意正整数, 判断这个数是奇数还是偶数

需求2 : 定义出两个浮点类型数据, 获取到两数中的较大值

代码

|  |
| --- |
| package com.ujiuye.cal;  public class Demo06\_三元运算符 {  public static void main(String[] args) {  // 需求1 : 定义出一个任意正整数, 判断这个数是奇数还是偶数  int i = 11;  // 表达式1 ? 表达式2 : 表达式3;  String s = i % 2 == 0 ? i + "是偶数" : i + "是奇数";  System.out.println(s);    // 需求2 : 定义出两个浮点类型数据, 获取到两数中的较大值  double d1 = 5.77;  double d2 = 5.77;  double max = d1 >= d2 ? d1 : d2;  System.out.println("两数中的较大值为: "+max);  }  } |

## **键盘录入和生成随机数**

### 键盘录入

1. 键盘录入 : JDK核心类库中, 已经给开发人员准备好了一个类(class), 类名Scanner, 来自于java.util包中, Scanner类型的功能就是可以接收通过键盘录入数据,将录入的数据赋值给某一个变量, 在代码中, 可以使用这个变量所表示的数据
2. Scanner类型使用步骤:
3. 导包 : 将Scanner类型所在的具体路径进行导入, 目的就是为了定位到Scanner类型所在的具体位置, 类的声明之上使用关键字 import,表示导入的概念

import java.util.Scanner; (.表示文件夹和文件之间级别)

1. 创建出一个Scanner类型对象

Scanner sc = new Scanner(System.in); // System.in默认通过键盘录入数据

1. 使用sc对象名(变量名) .调用Scanner类型中的方法功能

a : nextInt(), 表示可以接收通过键盘录入的一个整数, 功能有一个整数类型结果返回

需求1 : 通过键盘录入一个整数, 获取到这个整数的2倍

需求2 : 通过键盘录入三个整数, 求三个整数中的最大值

需求1代码

|  |
| --- |
| package com.ujiuye.scanner;  // 1. 导入Scanner类型所在的路径  import java.util.Scanner;  public class Demo01\_Scanner {  public static void main(String[] args) {  /\*  \* 需求 : 通过键盘录入一个整数, 获取到这个整数的2倍  \*/    // 2. 创建出一个Scanner类型对象  Scanner sc = new Scanner(System.in);  // 3. 调用Scanner中方法功能nextInt()  System.out.println("请输入一个整数: ");  int count = sc.nextInt();  System.out.println(count + "数值的2倍为: " + count \* 2);  }  } |

需求2代码

|  |
| --- |
| package com.ujiuye.scanner;  import java.util.Scanner;  public class Demo02\_Scanner练习 {  public static void main(String[] args) {  // 需求 : 通过键盘录入三个整数, 求三个整数中的最大值  Scanner sc = new Scanner(System.in);  // 多次调用nextInt方法功能, 需要三个整数  System.out.println("请输入第一个整数:");  int first = sc.nextInt();  System.out.println("请输入第二个整数:");  int second = sc.nextInt();  System.out.println("请输入第三个整数:");  int third = sc.nextInt();  // 三个数比较最大值 : 思路就是两两比较  // 1) 让first和second比较出一个较大值  // 2) 让较大值与third比出最大值  // max表示最大值  int max = first >= second ? first : second;  max = max >= third ? max : third;  System.out.println("三个数中的最大值为: " + max);  }  } |

### 生成随机数(扩展)

1. 随机数 : JDK中给开发人员提供核心类库文件, 有一个类型Random类, 功能就是可以生成指定范围的随机数, 来自于java.util包
2. 随机数Random使用步骤:
3. 导包 : 使用import关键字在类声明之上将Random类型所在的具体路径进行导入

import java.util.Random;

1. 创建出一个Random类型对象

Random ran = new Random();

1. 使用对象名ran.调用Random类型中方法工能:

nextInt(int count) : 方法功能表示生成指定范围随机整数, 参数count表示生成的随机数的范围, [0,count), 左闭右开, 闭区间包含, 开区间不包含

nextInt(5) : 0-4

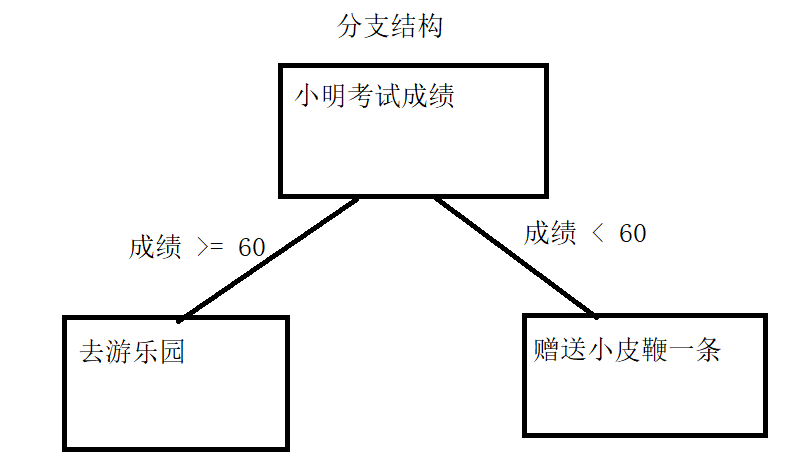
需求 : 生成1-100之间的任意一个随机数

代码

|  |
| --- |
| package com.ujiuye.scanner;  import java.util.Random;  public class Demo03\_Random {  public static void main(String[] args) {  // 需求 : 生成1-100之间的任意一个随机数  Random ran = new Random();  int number = ran.nextInt(100)+1;// (0-99) + 1 ---> 1-100  System.out.println(number);  }  } |

## 分支结构

分支结构: 限定代码的执行顺序, 表示当代码执行到某种情况下, 接下来代码执行选择可以有很多种, 需要从多种选择中挑选出一种最符合当下场景的逻辑运行, 类比于道路的分叉口



分支结构需要语法才能实现:

1. if...else系列语句
2. switch...case语句

### if第一种语法结构

1. if语法结构:

if(boolean类型表达式){

if代码逻辑;

}

1. 解释 :
2. if : 关键字, 表示如果
3. if小括号中必须给boolean类型表达式结果
4. if大括号中可以写任意逻辑
5. if语句的执行过程:
6. 先计算出if小括号中boolean类型表达式结果的值, true, false
7. 如果表达式结果为true, 那么执行大括号中的逻辑代码
8. 如果表达式结果为fasle , 那么不执行大括号中的逻辑代码, if 语句结束
9. if语句使用场景 : 满足条件就需要完成某一个动作, 不满足条件就不做, 可以使用if
10. 实际开发中的if语句使用场景 :
11. 外卖, 电商平台, 有一些场景, 可以根据购买用户的实际地址的距离, 决定是否要加运费(距离过远, 加运费)
12. 销售产品, 产品来自不同的公司, 有些公司产品销售, 自动加价5元(可能这些公司有境外产品,费用较高)

需求 : 某公司现在要给员工进行薪资调整, 工作年龄大于等于5年, 加薪3000元, 工龄不够5年的, 没有加薪

代码

|  |
| --- |
| package com.ujiuye.ifdemo;  public class Demo01\_If第一种语法 {  public static void main(String[] args) {  // workAge表示员工的工龄  int workAge = 14;  // salary表示员工基本工资  double salary = 5000;  // 如果工龄大于等于5年, 加薪  if(workAge >= 5) {  salary += 3000;// salary = salary + 3000;  }  System.out.println("员工工作了" + workAge + "年,目前薪资为" + salary);  }  } |

### if...else第二种语法结构

1. if...else语法结构:

if(boolean类型表达式){

if语句逻辑

}else{

else语句逻辑

}

1. 解释:

else : 关键字, 表示其他, 除了if条件之外的所有其他

1. if...else语句执行过程:
2. 先计算出if小括号中的boolean类型表达式结果, true, false
3. 如果boolean结果为true, 那么执行if语句逻辑
4. 如果boolean结果为false, 那么执行else语句逻辑
5. if...else使用场景: 使用在二选一场景之下, 如果想if和else中逻辑全部执行或者全部不执行, 做不到

需求 : 某公司现在要给员工进行薪资调整, 工作年龄大于等于5年, 加薪3000元, 工龄不够5年的, 加薪1000元

代码

|  |
| --- |
| package com.ujiuye.ifdemo;  public class Demo02\_IfElse第二种语法结构 {  public static void main(String[] args) {  // workAge表示员工工龄  int workAge = 5;  // salary表示员工基本工资  double salary = 4688.88;  // 根据员工工龄判断员工涨薪的级别  if(workAge >= 5) {  salary += 3000;  }else {  salary += 1000;  }  System.out.println("员工工作了" + workAge + "年,目前薪资为" + salary);  }  } |

### if...else if...else第三种语法结构

1. 语法结构:(多选1)

if(boolean类型表达式1){

if逻辑;

}else if(boolean类型表达式2){

逻辑2;

}else if(boolean类型表达式3){

逻辑3;

}...else{

else逻辑;

}

1. 执行过程:
2. 从if语句开始, 从上到下依次进行boolean类型表达式结果的计算, 如果某一个if条件结果为true, 那么执行对应的大括号中的逻辑, 整个if...else if...else语句结束;
3. 如果所有if语句条件全部false , 不成立, 那么执行else中的逻辑
4. if...else if...else使用的注意事项:
5. 第三种语法结构中, else语句可以没有; 如果没有else语句, 那么有可能所有if语句条件全部失败, 导致整个语法结构中, 没有可以执行语句

需求 : 某公司现在要给员工进行薪资调整,根据工龄决定涨薪的级别

1. [0-5) : 涨薪2000元
2. [5-10) : 涨薪3000元
3. [10-15) : 涨薪4000元
4. [15,以上) : 涨薪5000元

代码

|  |
| --- |
| package com.ujiuye.ifdemo;  public class Demo03\_IfElseIf第三种语法结构 {  public static void main(String[] args) {  /\*需求 : 某公司现在要给员工进行薪资调整,根据工龄决定涨薪的级别  1)[0-5) : 涨薪2000元  2)[5-10) : 涨薪3000元  3)[10-15) : 涨薪4000元  4)[15,以上) : 涨薪5000元\*/    // workAge表示员工工龄  int workAge = -15;  // salary表示员工基本工资  double salary = 4688.88;  if(workAge < 0) {  System.out.println("工龄有误,无法进行薪资调整");  }else if(workAge >= 0 && workAge < 5) {  salary += 2000;  }else if(workAge >= 5 && workAge < 10) {  salary += 3000;  }else if(workAge >= 10 && workAge < 15) {  salary += 4000;  }else {  salary += 5000;  }  System.out.println("员工工作了" + workAge + "年,目前薪资为" + salary);  }  } |

优化后的涨薪代码

|  |
| --- |
| package com.ujiuye.ifdemo;  public class Demo04\_IfElseIf注意事项 {  public static void main(String[] args) {  // workAge表示员工工龄  int workAge = 3;  // salary表示员工基本工资  double salary = 4688.88;  if(workAge < 0) {  System.out.println("工龄有误,无法进行薪资调整");  }else if(workAge < 5) {  salary += 2000;  }else if(workAge < 10) {  salary += 3000;  }else if(workAge < 15) {  salary += 4000;  }else if(workAge >= 15){  salary += 5000;  }    System.out.println("员工工作了" + workAge + "年,目前薪资为" + salary);  }  } |

### switch...case语法结构

1. 语法结构:

switch(表达式){

case 常量1:

逻辑1;

break;

case 常量2:

逻辑2;

break;

...

default:

default逻辑;

}

1. 解释:
2. switch : 关键字, 表示开关
3. switch小括号中, 表达式结果

JDK1.5版本之前 小于等于int类型基本数据类型 byte short char int

JDK1.5版本当前 枚举类型

JDK1.7版本 字符串类型 String

1. case : 关键字, 表示情况
2. break : 关键字,表示破坏,结束, switch语法执行到break关键字, 表示switch语句结束
3. default : 关键字, 表示默认, 相当于if...else中else角色, 如果所有的case对应的常量数据都无法匹配switch的计算结果, 那么执行default默认逻辑语句
4. switch...case执行过程
5. 先计算switch小括号中的表达式结果
6. 将结果依次(从上到下)与每一个case后面的常量进行数值匹配,如果某一个case常量匹配成功, 执行对应case后面的逻辑语句, 执行break结束整个switch
7. 如果所有的case常量都无法匹配switch的计算结果, 那么执行default中的默认逻辑

需求 : 定义出一个整数, 用于进行星期匹配, 数据范围1-7,对应匹配星期一到星期天, 如果不是1-7范围, 那么报出数据错误提示信息

代码

|  |
| --- |
| package com.ujiuye.switchdemo;  public class Demo01\_SwitchCase语句 {  public static void main(String[] args) {  // 需求 : 定义出一个整数, 用于进行星期匹配, 数据范围1-7,  // 对应匹配星期一到星期天, 如果不是1-7范围, 那么报出数据错误提示信息    // weekday表示对应星期数值  int weekday = 12;  switch(weekday) {  case 1:  System.out.println("星期一");  break;  case 2:  System.out.println("星期二");  break;  case 3:  System.out.println("星期三");  break;  case 4:  System.out.println("星期四");  break;  case 5:  System.out.println("星期五");  break;  case 6:  System.out.println("星期六");  break;  case 7:  System.out.println("星期日");  break;  default :  System.out.println("数据输入有误,需要1-7之间数据");  }  }  } |

### switch...case使用的注意事项

1. case 语句必须匹配常量
2. case 语句逻辑执行完毕之后, 通常每个case语句后面都会带有break, break功能就是为了结束switch语句; 但是如果多个case常量对于逻辑处理是相同的, 就没有必要在每一个case后面添加break, 如果case语句匹配之后, 没有break, 那么switch就不会结束(称为case的穿透性), switch会继续向下寻找一切可执行语句运行, 直到整个switch语句全部执行完毕或者遇到break
3. default : 默认执行语句, 在switch和case语句中, 可以没有, default就相当于if语句中else

需求 : 定义出一个整数, 用于进行工作日与休息日匹配, 数据范围1-5,输出工作日; 数据范围6-7输出休息日; 其余数据报出”数据有误”提示信息

代码

|  |
| --- |
| package com.ujiuye.switchdemo;  public class Demo02\_SwitchCase穿透性 {  public static void main(String[] args) {  // weekday表示星期数据  int weekday = 13;  /\*switch(weekday) {  case 1 :  System.out.println("工作日");  break;  case 2 :  System.out.println("工作日");  break;  case 3 :  System.out.println("工作日");  break;  case 4 :  System.out.println("工作日");  break;  case 5 :  System.out.println("工作日");  break;  case 6 :  System.out.println("休息日");  break;  case 7 :  System.out.println("休息日");  break;  default :  System.out.println("数据输入有误,需要1-7之间数据");  }\*/    switch(weekday) {  case 1 :  case 2 :  case 3 :  case 4 :  case 5 :  System.out.println("工作日");  break;  case 6 :  case 7 :  System.out.println("休息日");  break;  default :  System.out.println("数据输入有误,需要1-7之间数据");  }  }  } |