南京郵電大學
Nanjing University of Posts and Telecommunications

Java软件开发(混合式)

南京邮电大学 计算机学院
Java课程组

第2章 Java语言基础

- 2.1 常量与变量
- 2.2 数据类型
- 2.3 表达式与运算符
- 2.4 标识符、关键字及注释
- 2.5 项目实践：员工实发工资的计算

2.1.1 常量

常量有不同类型，表示不同类型的固定值。

- 数字常量：包括整数常量和浮点数常量。
- 字符常量：是单引号括起来的单个字符。
- 字符串常量：是由双引号引起来的多个字符组成的常量。
- 布尔常量：包括true或者false。
- 空常量：表示没有值的常量，即null。
- 枚举常量：是在枚举类型中定义的具体值。
- final常量：符号常量，即用一个符号来代表固定值，不能修改。

2.1.1 常量

常量有不同类型，表示不同类型的固定值。

- 数字常量：包括整数常量和浮点数常量。
- 字符常量：是单引号括起来的单个字符。
- 字符串常量：是由双引号引起来的多个字符组成的常量。
- 布尔常量：包括true或者false。
- 空常量：表示没有值的常量，即null。
- 枚举常量：是在枚举类型中定义的具体值。
- final常量：符号常量，即用一个符号来代表固定值，不能修改。

- **整数常量**有4种表示形式：十进制、二进制、八进制、十六进制。

① 十进制整数常量由数字0~9组成，如123，789。

② 二进制整数常量由数字0~1组成，书写时以“0B”或者“0b”开头，例如0b101。

③ 八进制整数常量由数字0~7组成，书写时以“0”开头，例如0123。

④ 十六进制整数常量由数字0~9和字符A~F组成，书写时以“0X”或者“0x”开头，例如0X25FA。

- **浮点数常量**表示带有小数点的数值。包括float和double类型。

① 浮点型表示：3.14或者2.5表示double类型的数据。

② 后缀表示：3.14f或者1.24F表示float类型的数据。3.14D或者3.14d或者3.14表示double类型的数据。

③ 科学记数法表示：3.0E8或者3.5e6。

2.1.1 常量

常量有不同类型，表示不同类型的固定值。

- 数字常量：包括整数常量和浮点数常量。
- 字符常量：是单引号括起来的单个字符。
- 字符串常量：是由双引号引起来的多个字符组成的常量。
- 布尔常量：包括true或者false。
- 空常量：表示没有值的常量，即null。
- 枚举常量：是在枚举类型中定义的具体值。
- final常量：符号常量，即用一个符号来代表固定值，不能修改。

- **字符型常量只能包含一个字符。**
- **字符型常量使用单引号表示。**
- 字符型常量有如下几种形式：

① **普通字符**：单引号引起来的单个字符，如‘A’，‘9’等。

② **转义字符**：如‘\n’表示换行符，‘\t’表示制表符，‘\’’表示单引号，‘\\’表示反斜杠等。

③ **Unicode字符**：使用‘\u’和4位十六进制数来表示Unicode字符，如‘\u0041’表示大写字母A。

2.1.1 常量

常量有不同类型，表示不同类型的固定值。

- 数字常量：包括整数常量和浮点数常量。
- 字符常量：是单引号括起来的单个字符。
- 字符串常量：是由双引号引起来的多个字符组成的常量。
- 布尔常量：包括true或者false。
- 空常量：表示没有值的常量，即null。
- 枚举常量：是在枚举类型中定义的具体值。
- final常量：符号常量，即用一个符号来代表固定值，不能修改。

- **字符串常量使用双引号表示。**

① **普通字符串**：如“Hello World”。

② **含转义字符**：如“Hello\nWorld”。

③ **字符串拼接**：“Hello”+“World”即为“HelloWorld”。

2.1.1 常量

常量有不同类型，表示不同类型的固定值。

- 数字常量：包括整数常量和浮点数常量。
- 字符常量：是单引号括起来的单个字符。
- 字符串常量：是由双引号引起来的多个字符组成的常量。
- 布尔常量：包括true或者false。
- 空常量：表示没有值的常量，即null。
- 枚举常量：是在枚举类型中定义的具体值。
- final常量：符号常量，即用一个符号来代表固定值，不能修改。

- 布尔常量只有true(真)和false(假)。
- 布尔常量主要应用于逻辑判断中。

- 空常量即null，常用于初始化一个变量。

- enum Weekday{ MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY};
  其中，MONDAY等为枚举常量。

- final double PI = 3.1415926;

2.1.2 变量

变量定义

在计算机编程中，变量是一种用于存储数据和信息的工具，其值可以在程序执行期间改变。

变量分类

变量可以根据其数据类型、作用域和生命周期分为不同类型，如整型、浮点型、字符型等。

存储方式

变量通常被存储在内存中，以便在程序执行期间快速访问和操作。

特点

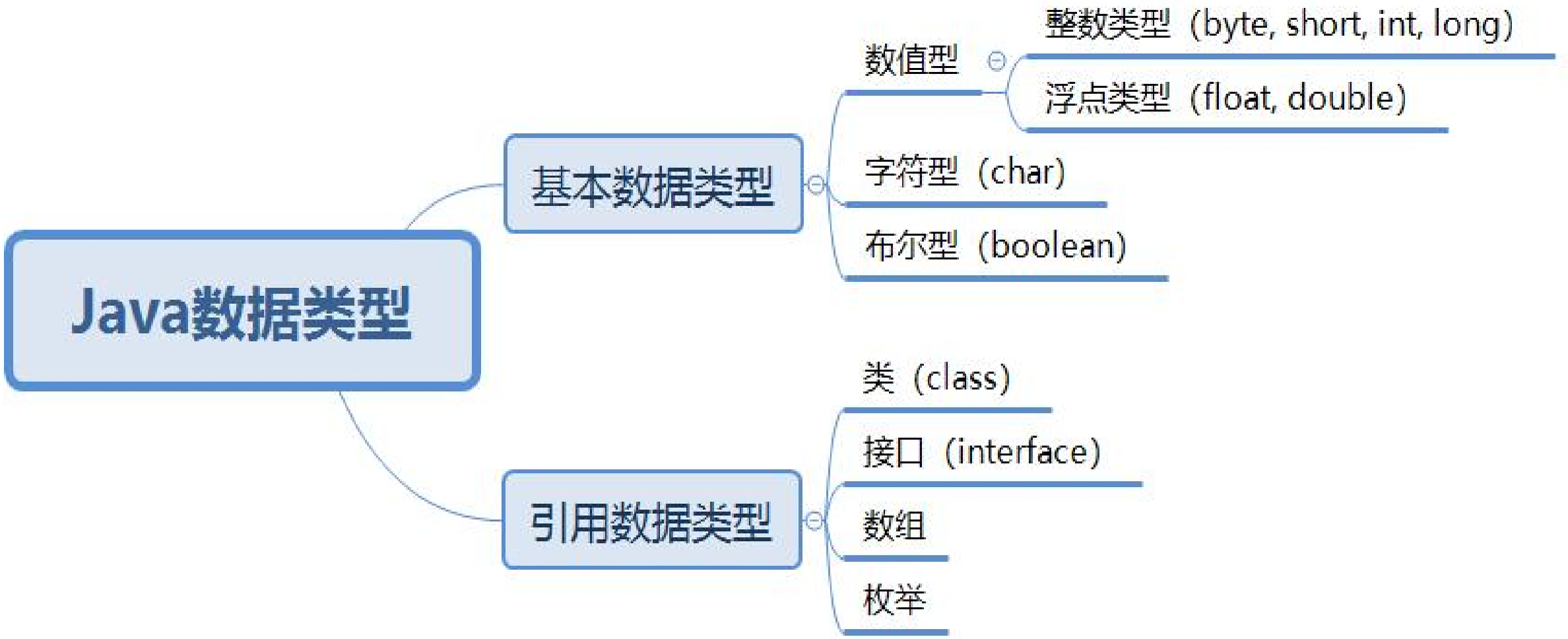
变量具有可变性、作用域和生命周期，可以用于存储各种数据和信息，包括数字、字符串、布尔值等。

- 变量是一种存储数据的容器，可以存储各种不同的数据类型。
- 定义语法：

  数据类型 变量名 = 初始化值;

  例如： int num = 10;
- 变量名在同一作用域内必须是唯一的。同时需要遵循标识符的命名规则，见2.4.1。

2..2 数据类型


Java数据类型
基本数据类型
数值型
整数类型（byte, short, int, long）
浮点类型（float, double）
字符型（char）
布尔型（boolean）
引用数据类型
类（class）
接口（interface）
数组
枚举

2.2.1基本数据类型

基本数据类型也称作简单数据类型。Java语言有8种简单数据类型，分别是：boolean、byte 、short、int、long、 float、double、char。

这8种数据类型习惯上可分为4大类型：

- 整数类型：byte、short、int、long
- 浮点类型：float、double
- 字符类型：char
- 布尔类型：boolean

数据类型	关键字	占用位数	取值范围
布尔型	boolean	8	true, false
字符型	char	16	‘\u 0000’ ~ ‘\u FFFF’
字节型	byte	8	-128~127 ($-2^7 \sim 2^7-1$)
短整型	short	16	-32768~32767($-2^{15} \sim 2^{15}-1$)
整型	int	32	$-2^{31} \sim 2^{31}-1$
长整型	long	64	$-2^{63} \sim 2^{63}-1$
浮点型	float	32	1.4012984643248170e-45~3.4028234663852886e+38
双精度型	double	64	4.9406564584124654e-324~1.7976931348623157e+308d

2.2.1基本数据类型

数据类型 变量名 = 初始化值;

- long l1 = 235L;
- float f1 = 3.14f;
- double d1 = 3.14;
- char c1 = ‘\n’;
- boolean b1 = true;
- boolean b1 = 5>2;

数据类型	关键字	占用位数	取值范围
布尔型	boolean	8	true, false
字符型	char	16	‘\u0000’ ~ ‘\uFFFF’
字节型	byte	8	-128~127 ($-2^7 \sim 2^7-1$)
短整型	short	16	-32768~32767($-2^{15} \sim 2^{15}-1$)
整型	int	32	$-2^{31} \sim 2^{31}-1$
长整型	long	64	$-2^{63} \sim 2^{63}-1$
浮点型	float	32	1.40129846432481707e-45 ~ 3.40282346638528860e+38
双精度型	double	64	4.94065645841246544e-324 ~ 1.79769313486231570e+308d

2.2.2 引用数据类型

概念

引用数据类型是指在内存中保存对象引用的变量类型。它们由程序员定义，并在运行时分配内存空间。使用这些类型时，实际是在使用一个引用，该引用指向实际的数据对象。

作用

引用数据类型在面向对象编程中发挥着重要作用，是构成程序的基本单元，承担着不同角色。

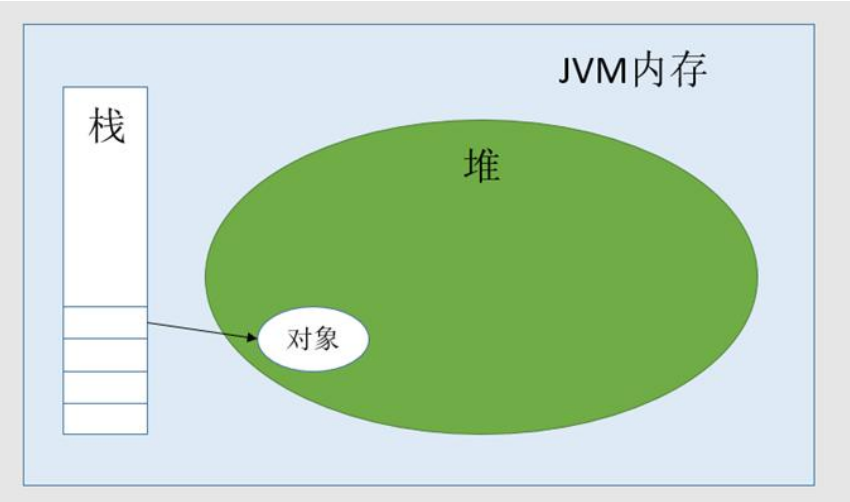
类型

包括：类、数组、接口、枚举等。

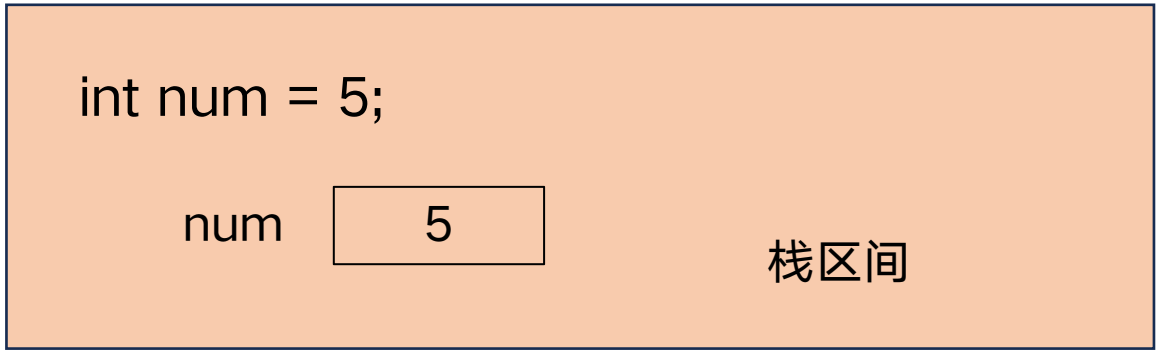
优势

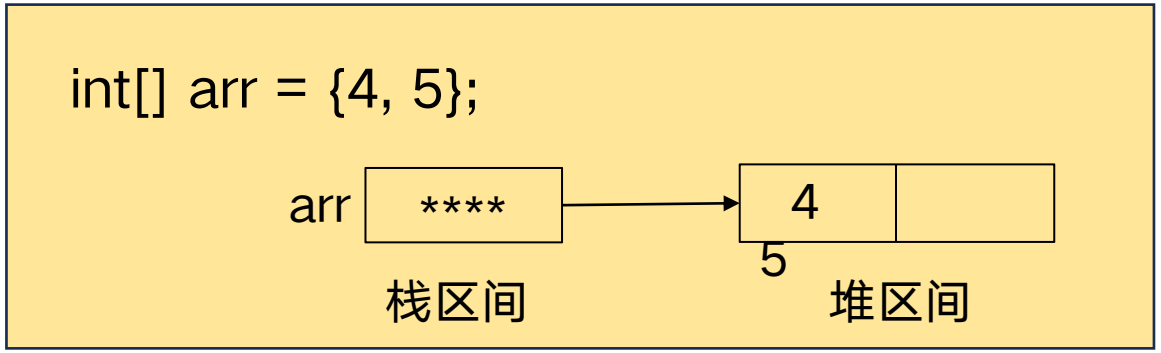
使用引用数据类型可以让我们更加灵活地定义和管理程序中的对象和类，从而更好地组织和管理程序逻辑。

2.2.2 引用数据类型



- **基本数据类型**的变量存储在栈中。
- **引用数据类型**的变量存储在堆中。





2.2.3 数据类型转换

基本数据类型的转换就是把一种基本数据类型变量转变成另一种基本类型变量。**下列基本类型会涉及数据转换，不包括逻辑类型和字符类型**。我们将这些类型按精度从“低”到“高”排列了顺序：

byte short int long float double

当把级别低的变量的值赋给级别高的变量时，系统**自动**完成数据类型的转换，如int型转换成long型，称之为**自动类型转换（隐式类型转换）**。

当把级别高的变量的值赋给级别低的变量时，必须使用**强制/显示类型转换**，格式为：**（类型名）要转换的值；**

```
int x = 34.89;
long y = 56.98F;
byte a = 128;
float b = 3.14;
```

```
int x = (int)34.89;
long y = (long)56.98F;
byte a = (byte)128;
float b = 3.14f;
```

```
34
56
-128
3.14
```

【例2-1】自动类型转换的应用。

例2-1中，定义了一个int类型的变量num1，并将其赋值为100。然后将num1赋值给一个double类型的变量num2，这个操作是一种自动类型转换。由于double类型的精度比int类型的精度高，因此自动将int类型的值转换成double类型的值参与运算，但变量num1和num2本身类型不变。

```
package ch02;
public class TestAutoType {
    public static void main(String[] args) {
        int num1 = 100;           // 定义一个int类型的变量
        double num2 = num1;  // 将int类型的变量自动转换成了double类型的变量
        System.out.println("num1 = " + num1);   // 输出：num1 = 100
        System.out.println("num2 = " + num2);   // 输出：num2 = 100.0
    }
}
```

程序运行结果如下。
num1 = 100
num2 = 100.0

【例2-2】强制类型转换应用。

例2-2定义了一个double类型的变量num1，并将其赋值为3.14159。然后将3.14159强制转换成了int类型的数据赋值给变量num2，这是一种强制类型转换。由于int类型的精度比double类型的精度低，因此在进行强制类型转换时可能会丢失精度。

```
package ch02;

public class TestCastType {
    public static void main(String[] args) {
        double num1 = 3.14159;    // 定义一个double类型的变量
        int num2 = (int)num1;  // 将double类型的数据强制转换成了int类型的数据
        System.out.println("num1 = " + num1);  // 输出：num1 = 3.14159
        System.out.println("num2 = " + num2);  // 输出：num2 = 3
    }
}
```

（类型名）要转换的值；

程序运行结果如下。
num1 = 3.14159
num2 = 3

【例2-3】混合类型运算的应用。

例2-3中，定义了一个int类型的变量num1，并将其赋值为10，同时定义一个double类型的变量num2，并将其赋值为3.14。然后将num1和num2相加，并将结果赋值给一个double类型的变量result。由于支持混合类型运算，此时，会自动将int类型的数据转换成double类型的数据，然后才进行相加运算，并得出结果。最后，输出了result的值。

```
package ch02;
public class TestAutoCase {
    public static void main(String[] args) {
        int num1 = 10;        // 定义一个int类型的变量
        double num2 = 3.14;    // 定义一个double类型的变量
        double result = num1 + num2;// 将一个int类型的变量和一个double类型的变量相加，并将结果赋值给一个double类型的变量
        System.out.println("result = " + result);  // 输出：result = 13.14
    }
}
```

程序运行结果如下。
result = 13.14

2.3 运算符与表达式

操作数可以是变量、常量、方法调用或者其他表达式。

- 运算符: 指明对操作数的运算方式 。
- 表达式： 由运算符和操作数组成的。
- 按功能分：
  - 算术运算符： +，—，*，/，%，++，——
  - 关系运算符： >，<，>=，<=，==，!=
  - 逻辑运算符： !，&&，||
  - 位运算符： >>，<<，>>>，&，|，^，~
  - 赋值运算符： =，+=，—=，*=，/=等。
  - 三元运算符： ?：(如e1?e2:e3)

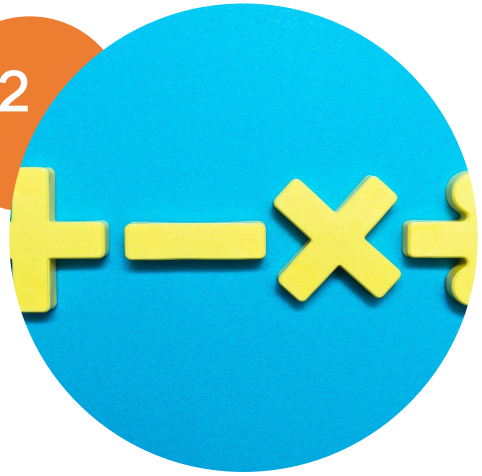
2.3.1 表达式

01

表达式定义

数学和编程中，表达式是由**标点符号**、**数字**、**变量**和**运算符**组成的组合，可以用来计算和操作数据。

02



表达式的作用

表达式可以用来建立数学和编程中的各种计算和操作，如加法、减法、乘法和除法等。

03

表达式的种类

根据表达式的不同，表达式可以是简单的算术表达式、复杂的关系表达式或逻辑表达式等。

2.3.2 算术运算符

运算符	用法	描述	示例（x=5,y=3）
+	x + y	用于执行加法操作。 如果两个操作数都是数值，则执行数值相加； 如果其中一个或两个操作数是字符串，则执行字符串连接操作。	int a = x + y;　// 8 String s = "hello" + "world"; // "helloworld"
-	x - y	用于执行减法操作	int a =x - y;　// 2
*	x * y	用于执行乘法操作	int a =x * y;　// 15
/	x / y	用于执行除法操作	int a =x / y;　// 1
%	x % y	用于执行模运算，即求除法的余数	int a =x % y;　// 2
++	x++	用于变量的值自增1	int a= x++; //a值为5，x为6
--	x--	用于变量的值自减1	int a= x--; //a值为5，x为4

2.3.2 算术运算符

- **注意：**

(1)算术运算的结果的精度和操作数中精度高者一致。因此两个整型的数据做除法时，结果是截取商的整数部分，而小数部分被截断。

例如：2/4=0

2.0/4=0.5

13%5=3

(2)整型(int, long, short)和浮点型(float，double)都可以进行算术运算，包括%。

例如：23.6%12=11.6　　23.6%12.2 = 11.4

2.3.2 算术运算符

- 前缀：如++x、--x，在使用x之前，先使x的值加（减）1。
- 后缀：如**x++**、**x--**，在使用x之后，再使x的值加（减）1。
- 自增和自减的操作对象只能是**变量，不能是常量**。

例如：

```
int   x = -1;
      x = -x;
int   y = ( x++  ) * 3;   //x为2，y为3
```

填空题 10分

设置

假设：

```
int  x = 5;
int  y = (-- x ) * 3;
```

则x= [填空1] ， y= [填空2] 。

现在继续定义：

```
int z=(y++)-(-- x);
```

则x= [填空3] , y= [填空4] , z= [填空5] 。

作答

【例2-4】递增、递减运算符的应用。

```
package ch02;

public class TestIncDec {
    public static void main(String [] args) {
        int x = 10;
        int y = 20;
        int i1 = x++;
        int i2 = ++x;
        int i3 = y--;
        int i4 = --y;
        System.out.println("x++ = " + i1);
        System.out.println("++x = " + i2);
        System.out.println("y-- = " + i3);
        System.out.println("--y = " + i4);
    }
}
```

程序运行结果如下：

```
x++ = 10
++x = 12
y-- = 20
--y = 18
```

案例

自增、自减拓展

```
int c = 10;
int d = 5;
int rs3 = c++ + ++c - --d - ++d + 1 + c--;
System.out.println(rs3);
System.out.println(c);
System.out.println(d);
```

```
"C:\Progra
26
11
5
```

c++: 后缀递增，先使用c的值然后递增。c的值是10，然后c变为11。
++c: 前缀递增，先递增然后使用。c变为12，然后使用12。
--d: 前缀递减，先递减然后使用。d变为4，然后使用4。
++d: 前缀递增，先递增然后使用。d变为5，然后使用5。
c--: 后缀递减，先使用c的值然后递减。c的值是12，然后c变为11。
将这些值代入表达式中：
rs3 = (10) + (12) - (4) - (5) + (1) + (12)

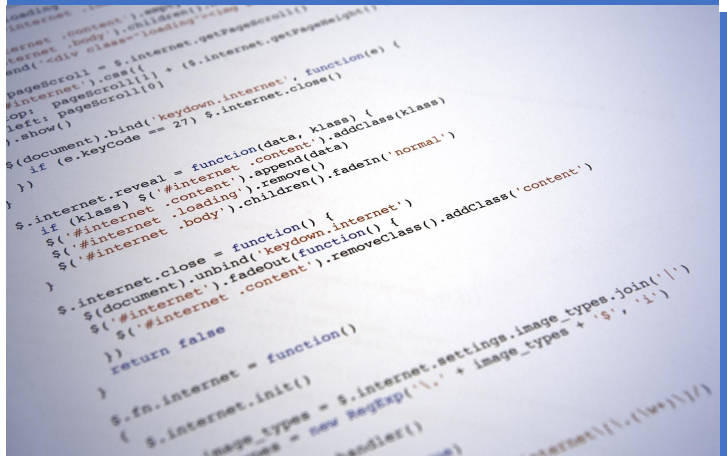
2.3.3 关系运算符

关系运算符的定义

关系运算符是用于数学和编程中测试两个值之间关系的符号，例如大于、小于、等于等。

逻辑运算规则

关系运算符的优先级和结合性各不相同，需要根据具体情况进行分析和运算。



关系运算符的种类

关系运算符通常由一个或多个字符组成，每个运算符代表一个关系，例如“>”表示大于。

2.3.3 关系运算符

- 关系运算符：用于比较两个**数据**之间的大小关系的运算符。
- 关系运算的结果：**布尔值**（true或false）。
- 关系表达式：由关系运算符和操作数构成的表达式。

例如：

```
int  x = 5, y = 7;
boolean   b = ( x == y );  // x==y的结果为false
```

2.3.3 关系运算符

运算符	用法	描述	示例(x=5, y=3)
<	x < y	用于比较操作数左边的值是否小于右边的值	boolean result = x < y; //true
<=	x <= y	用于比较操作数左边的值是否小于或等于右边的值	boolean result = x <= y; // true
>	x > y	用于比较操作数左边的值是否大于右边的值	boolean result = x > y; // false
>=	x >= y	用于比较操作数左边的值是否大于或等于右边的值	boolean result = x >= y; // false
==	x == y	用于比较操作数左右两边的值是否相等	boolean result = x == y; // false
!=	x != y	用于比较操作数左右两边的值是否不相等	boolean result = x != y; // true

注意点：

- 浮点数相等的比较

比较两个浮点数时会出现精度误差，所以判断相等如下：

if(Math.abs(a-b)<1e-9)

```
Double d1 = 0.000000001;
Double d2 = 0.000000001;
if(Math.abs(d1-d2)<1e-9)
    System.out.println("equals");
else
    System.out.println("not equals");
```

```
equals
```

2.3.3 关系运算符

注意点：

- 浮点数相等的比较

  比较两个浮点数时会出现精度误差，所以判断相等如下：

  if(**Math.abs(a-b)<1e-9**)

```
Double a = 0.1 + 0.2; // 实际可能是 0.30000000000000004
Double b = 0.3;
System.out.println(a == b); // 输出 false，因为不完全相等
System.out.println(Math.abs(a – b) < 1e–9); // 输出 true，因为差值足够小
```

Math.abs(a – b) < 1e–9 是**浮点数**比较的“行业惯例”，“容忍误差”，非常常用，务必掌握。

2.3.3 关系运算符

注意点：

- 字符串相等的比较

在 Java 中，比较字符串内容一定要用 .equals()，== 只能判断是不是同一个对象。

```
String s1 = "hello";
String s2 = new String("hello");
if (s1 == s2)
    System.out.println("s1==s2: equals");
else
    System.out.println("s1==s2: not equals");

if (s1.equals(s2))
    System.out.println("s1和s2的内容相同");
```

```
s1==s2: not equals
s1和s2的内容相同
```

【例2-5】关系运算符综合应用。

```
package ch02;

public class TestRelational {
    public static void main(String[] args) {
        int i = 10;
        int j = 20;
        char c1 = 'A';
        char c2 = 'B';
        System.out.println("10 == 20 : " + (i == j));
        System.out.println("10 != 20 : " + (i != j));
        System.out.println("A == B : " + (c1 == c2));
        System.out.println("A >= B : " + (c1 >= c2));
    }
}
```

程序运行结果如下。

```
10 == 20 : false
10 != 20 : true
A == B : false
A >= B : false
```

填空题 8分

设置

int a=24;

1. 表达式 *a%8==0* 的值是 [填空1] 。
2. 表达式 *(a-4)>20* 的值是 [填空2] 。
3. 表达式 *a/5 ==4* 的值是 [填空3] 。
4. 表达式 *a++!=24* 的值是 [填空4] 。

作答

2.3.4 逻辑运算符

- **逻辑运算**是对**布尔型数据**进行的运算，运算的结果仍然是布尔型。

记下来

运算符	运算	示例	运算规则
&&/&	逻辑与	x && y x & y	x,y都真时结果才为真(短路) 用于判断多个条件是否同时满足
||/|	逻辑或	x || y x | y	x,y都假时结果才为假(短路) 用于判断多个条件是否至少满足一个
!	逻辑非	! x	x真时为假, x假时为真

2.3.4 逻辑运算符

1. 逻辑与运算符（&&和&）

- **逻辑与运算符(&&)**：用于判断多个条件是否同时满足。如果第一个条件为假(false)，则不会再判断后面的条件，直接返回假；只有在第一个条件为真(true)的情况下，才会继续判断后面的条件。**(短路)**
- **逻辑与运算符(&)**：也可以用于操作布尔类型的变量。对于布尔类型变量的运算，&也表示逻辑与操作。在进行逻辑与运算时，&会对两个操作数进行判断，并返回一个布尔类型的结果，并**无短路**情况。

2. 逻辑或运算符（||和|）

- **逻辑或运算符(||)**：用于判断多个条件是否至少满足一个。它的特点是，如果第一个条件为真（true），则不会再判断后面的条件，直接返回真；只有在第一个条件为假（false）的情况下，才会继续判断后面的条件。**(短路)**
- **逻辑或运算符(|)**：用于判断两个表达式中是否至少有一个为 true，如果是则返回 true，否则返回 false，并且**无短路**情况。

【例2-6】逻辑运算符的应用

```
public class TestLogicalAnd {
    public static void main(String[] args) {
        int age = 25;
        boolean hasExperience = true;
        System.out.println("能否参加面试："+ (age >= 18 && hasExperience));
    }
}
```

程序运行结果如下。
能否参加面试：true

```
public class TestShortCircuit {
    public static void main(String[] args) {
        int  x = 3,  y = 5 ;
        boolean  b= x > y && x++ == y-- ;
        System.out.println("x="+x+ ", y="+y +", b="+b);
    }
}
```

程序运行结果如下。
x=3, y=5, b=false

填空题 8分

设置

int a = 24， b = 10;

1. 表达式 (a%3==0) && (a%4==0)的值是 [填空1] 。
2. 表达式 a++<2*b && b++>10的值是 [填空2] ，a的值是 [填空3] ，b的值是 [填空4] 。

作答

符号	介绍	说明
&	逻辑与	**必须都是true，结果才是true;** 只要有一个是false，结果一定是false。
|	逻辑或	只要有一个为true、结果就是true
!	逻辑非	你真我假、你假我真。 !true=false 、 !false= true
^	**逻辑异或**	如果两个条件都是false或者都是true则结果是false。**两个条件不同结果是true。**

符号	介绍	说明
&&	短路与	判断结果与“&”一样。过程是左边为 false，右边则不执行。
||	短路或	判断结果与“|”一样。过程是左边为 true， 右边则不执行。

2.3.5 位运算符

位运算符的定义

位运算符是用于计算机内存中的位级别操作的符号，例如与、或、取反等。

位运算符的种类

位运算符通常由一个或多个字符组成，每个运算符代表一种位运算，例如“&”表示与。

位运算的规则

位运算符的优先级和结合性各不相同，需要根据具体情况进行分析和运算。

2.3.5 位运算符

运算符	用法	描述	示例(x=5, y=3)
&	x & y	按位与，两个操作数的对应位都为1才为1，否则为0	int a = x & y; // 1
|	x | y	按位或，两个操作数的对应位只要有一个为1就为1，否则为0	int a = x | y; // 7
^	x ^ y	按位异或，两个操作数的对应位不同则为1，否则为0	int a = x ^ y; // 6
~	~x	按位取反，操作数的每一位都取反	int a = ~x ; // -6
<<	x<< y	左移位，把操作数的二进制位全部左移，高位丢弃，低位补0，相当于乘以2的n次方	int a = x << y; // 40
>>	x >> y	右移位，把操作数的二进制位全部右移，低位丢弃，高位的值由原来的最高位决定。对于正数，高位补0；对于负数，高位补1，相当于除以2的n次方取整	int a = x >> y; // 0

```
    5:    0 0 0 0  0 1 0 1
    3: (&) 0 0 0 0  0 0 1 1
5 & 3:    0 0 0 0  0 0 0 1
```

与0与为0，与1与保留原值

```
    5:    0 0 0 0  0 1 0 1
    3: (|) 0 0 0 0  0 0 1 1
5 | 3:    0 0 0 0  0 1 1 1
```

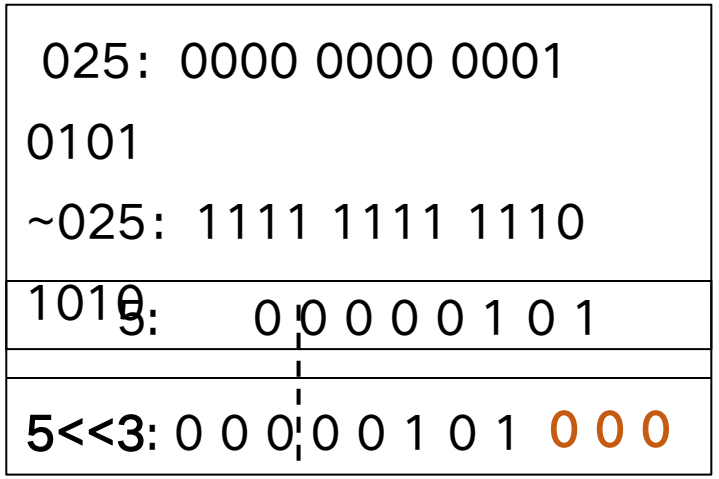
与1或为1，与0或保留原值

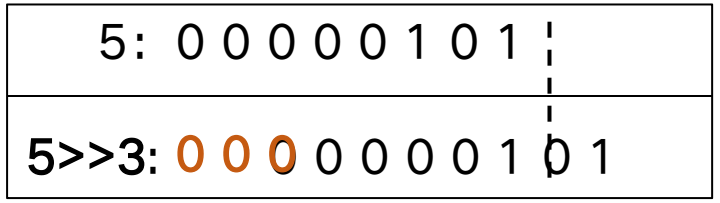
```
    5:    0 0 0 0  0 1 0 1
    3: (^) 0 0 0 0  0 0 1 1
5 ^ 3:    0 0 0 0  0 1 1 0
```

异或：相同为0，相异为1

2.3.5 位运算符

运算符	用法	描述	示例(x=5, y=3)
&	x & y	按位与，两个操作数的对应位都为1才为1，否则为0	int a = x & y; // 1
|	x | y	按位或，两个操作数的对应位只要有一个为1就为1，否则为0	int a = x | y; // 7
^	x ^ y	按位异或，两个操作数的对应位不同则为1，否则为0	int a = x ^ y; // 6
~	~x	**按位取反，操作数的每一位都取反**	**int a = ~x ; // -6**
<<	x<< y	**左移位**，把操作数的二进制位全部左移，**高位丢弃，低位补0**，相当于乘以2的n次方	int a = x << y; // 40
>>	x >> y	**右移位**，把操作数的二进制位全部右移，**低位丢弃**，高位的值由原来的最高位决定。**对于正数，高位补0；对于负数，高位补1**，相当于除以2的n次方取整	int a = x >> y; // 0





【例2-11】位运算符的综合应用。

```
package ch02;

public class TestBit {
        public static void main(String[] args) {
                int x = 10;
                int y = 20;

                System.out.println("x << 2 : " + (x << 2));
                System.out.println("y >> 2 : " + (y >> 2));
                System.out.println("x & y : " + (x & y));
        }
}
```

```
x:    0 0 0 0   1 0 1 0
y:    0 0 0 1   0 1 0 0
```

程序运行结果如下。

```
x << 2 : 40
y >> 2 : 5
x & y : 0
```

2.3.6 赋值运算符

- 赋值运算符是用于给变量赋值的符号，用“=”表示。
- Java中还提供了复合赋值运算符，这些运算符允许在赋值前先进行其他运算。
- 赋值运算符的优先级是最低的。

运算符	用法	描述	示例x=5，y=3
+=	x +=y	将右侧表达式的值加到左侧变量上	x + = y; // 8
-=	x -=y	将右侧表达式的值从左侧变量中减去	x - = y; // 2
*=	x *=y	将右侧表达式的值乘到左侧变量上	x *= y; // 15
/=	x /=y	将左侧变量的值除以右侧表达式的值	x /= y; // 1
%=	x %=y	将左侧变量的值取模（求余）右侧表达式的值	x %= y; // 2

【例2-12】 赋值运算符的综合应用。

```
package ch02;

public class TestAssignment {
    public static void main(String[] args) {
        int b1 = 2;
        int b2 = 4;
        System.out.println("b1 += b2 = " + (b1 += b2));
        System.out.println("b1 -= b2 = " +  (b1 -= b2));
        System.out.println("b1 *= b2 = " +  (b1 *= b2));
        System.out.println("b1 /= b2 = " +  (b1 /= b2));
        System.out.println("b1 %= b2 = " + (b1 %= b2));
    }
}
```

程序运行结果如下。

```
b1 += b2 = 6
b1 -= b2 = 2
b1 *= b2 = 8
b1 /= b2 = 2
b1 %= b2 = 2
```

基本赋值运算符

- 就是“=”。

```
int a = 10;  // 先看“=”右边，把数据10赋值给左边的变量a存储。
```

扩展赋值运算符

符号	作用	说明
+=	加后赋值	a+=b 等价于 a = (a的数据类型)(a+b); 将a + b的值给a
-=	减后赋值	a-=b 等价于 a = (a的数据类型)(a-b); 将a - b的值给a
*=	乘后赋值	a*=b 等价于 a = (a的数据类型)(a*b); 将a * b的值给a
/=	除后赋值	a/=b 等价于 a = (a的数据类型)(a/b); 将a / b的商给a
%=	取余后赋值	a%=b 等价于 a = (a的数据类型)(a%b); 将a % b的商给a

注意：扩展的赋值运算符隐含了强制类型转换。

扩展赋值运算符

```
byte a = 10;
a += 5;  // 等价于 a = (byte)(a + 5)
System.out.println(a);  // 输出: 15
```

尽管 a + 5 的结果是 int 类型，但是由于使用了 +=，结果被自动转换回 byte 类型。

+=还可以实现数据的累加，把别人的数据加给自己。

```
// 计步器累加步数
int steps = 0;
steps += 2000; // 上午走了2000步
steps += 6000; // 下午走了6000步
steps += 3000; // 晚上走了3000步
System.out.println("今天总共走了:" + steps + "步"); // 输出11000步
```

单选题 1分

设置

南京邮电大学 Nanjing University of Posts and Telecommunications

```
byte b = 100;
b += 130;
System.out.println(b); 输出结果是:
```

A 230

B −26

C 编译错误

提交

注意：扩展的赋值运算符隐含了强制类型转换。

可能出现的问题

```
byte b = 100;
b += 130;  // 等价于 b = (byte)(b + 130)
System.out.println(b);  // 输出: -26
```

b + 130 的结果超出了 byte 的范围，但是由于隐含的类型转换，结果被截断为 byte 类型，导致了溢出。

填空题 1分

设置

南京邮电大学 Nanjing University of Posts and Telecommunications

```
int x = 1 , y = 2 , z = 3;
System.out.println(y+=z--/++x);//输出为：
```
[填空1]

作答

填空题 2分

设 x = 1 , y = 2 , z = 3,

则表达式 y+=z--/++x 的值是 [填空1] 。

作答

2.3.7 三元运算符

- 三元运算符是一个包含3个操作数的运算符，语法形式如下：

表达式1 ？ 表达式2 ： 表达式3

其中：表达式1是判断条件，其取值为true或者false。

如果表达式1的值为true，则运算结果为表达式2的值；

如果表达式1的值为false，则运算结果为表达式3的值。

```
例如：  y= x>=0  ? x: -x          //求|x|
        max= x>y ? x: y           //求x,y的较大者
```

【例2-13】 三元运算符的综合应用。

```
package ch02;

public class TestTernary {
    public static void main(String[] args) {
        int b1 = 2;
        int b2 = 4;
        int b = b1 > b2 ? b1 : b2;
        System.out.println("b1和b2中比较大的是" + b);
    }
}
```

程序运行结果如下。
b1和b2中比较大的是4

2.3.8 运算符的优先级

高

优先级	描述	运算符	结合性
1	最高优先级	. [ ] ( )	左→右
2	单目运算	+(正号) –(负号) ++ -- ~ ！ 强制类型转换符	右→左
3	算术乘除运算	* / %	左→右
4	算术加减运算	+ -	左→右
5	移位运算	>> << >>>	左→右
6	关系运算	< <= > >=	左→右
7	相等关系运算	== !=	左→右
8	按位与,布尔逻辑与	&	左→右
9	按位异或	^	左→右
10	按位或,布尔逻辑或	|	左→右
11	逻辑与	&&	左→右
12	逻辑或	||	左→右
13	三目条件运算	? :	右→左
14	赋值运算	= += -= *= /= %= <<= >>=	右→左

低

- 运算符的**优先级**决定了运算执行的先后顺序。
- 运算符的**结合性**决定了并列级别的运算符的先后顺序。

【例2-14】多种运算符的优先级的应用。

```
package ch02;
public class TestMix {
    public static void main(String[] args) {
        int result = 10 + 2 * 6 / (4 - 2) % 3;
        System.out.println("Result: " + result);
        boolean isEqual = 5 < 7 && 8 >= 10 || !(4 != 4);
        // isEqual = (5 < 7) && (8 >= 10) || (!(4 != 4));
        System.out.println("isEqual: " + isEqual);

        int x = 5;
        int y = 10;
        int z = 15;
        boolean result2 = (x > y) ^ (y <= z) || (z > y) || !(x < z);
        //  result2 = (x > y) ^ (y <= z) || (z > y) || (!(x < z));
        System.out.println("Result 2: " + result2);
    }
}
```

程序运行结果如下。
Result: 10
isEqual: true
Result 2: true

2.4.1 标识符

标识符的概念

标识符是用于识别程序中各种元素，如**变量**、**方法**、**类**或**接口**等的名称。

标识符的作用

标识符的作用是让开发者能够清晰地识别和命名各种元素，从而更好地管理和理解代码。

标识符的命名规则

标识符的命名规则：

1）标识符可以由字母、数字、下划线(_)或美元符号($)组成，并且不能以数字开头。

2）标识符区分大小写。

3）标识符不能是**关键字**。

标识符的命名规范：

1）标识符应该具有描述性，便于理解和维护程序。

2）类名的首字母应该大写，采用驼峰命名法，如ExampleClass。

3）方法名和变量名的首字母应该小写，采用驼峰命名法，如calculateArea。

4）常量的命名应该全部大写，单词之间用下划线分隔，如PI，MAX_VALUE。

2.4.2 关键字

关键字是预先定义的、具有特殊用途的保留字，用于表示语言的特定功能或特性。

类别	关键字
访问修饰符的关键字（3个）	public、protected、private
定义类、接口、抽象类和实现接口、继承类的关键字、实例化对象（6个）	class、interface、abstract、implements、extends、new
包的关键字（2个）	import、package
数据类型的关键字（9个）	byte、char、boolean、short、int、float、long、double、void
条件循环（流程控制）（12个）	if、else、switch、case、default、while、for、do、break、continue、return、instanceof
修饰方法、类、属性和变量（9个）	static、final、super、this、native、strictfp、synchronized、transient、volatile
错误处理（5个）	catch、try、finally、throw、throws
其他（2个）	enum、assert

2.4.3 注释

注释的概念

注释是用于解释或描述代码片段的文字，可以帮助其他开发者更好地理解代码。

注释的分类

注释可分为单行注释、多行注释和文档注释。

注释的作用

注释在程序编写中是非常有用的，可以帮助其他开发者更好地理解代码，维护代码的未来可读性。

注释的编写规则

编写注释时，应使用适当的文字和格式，使注释易于阅读和理解，同时避免使用过于复杂或晦涩的文字。

```
/**
 * 这是文档注释，可以使用javadoc工具将此注释中的内容整合到API文档中
 */
public static void cal(){
    //定义变量sum并初始化为0,用于求和。
    int sum = 0;

    /*
     下面代码用于计算1~10的累加和，并打印结果。
   */
     for(int i=1;i<=10;i++)
         sum+=i;
     System.out.println("1~10的和是:"+sum);
 }
```

2.5 项目实践：员工实发工资的计算

员工管理系统中实发工资根据设定的的基本工资、奖金、补贴和本月的请假天数进行计算。

实发工资=基本工资+奖金+补贴-请假的扣款

系统中有三类员工：

1）经理：奖金=基本工资*50%，交通补贴为200，请假按天扣除。

2）董事：奖金=基本工资*8%，交通补贴为2000，请假按天扣除。

3）普通员工：奖金=基本工资*10%，交通补贴为1000，请假按天扣除。

```
package ch02;
public class Salary {
    public static void main(String[] args) {
        double salary1 = 0.0;    //经理实发工资
        double salary2 = 0.0;    //董事实发工资
        double salary3 = 0.0;    //普通员工实发工资
        // 请假天数
        int leaveDays = 2;
        // 基本工资
        double basicSalary1 = 9000;      //经理基本工资
        double basicSalary2 = 15000;     //董事基本工资
        double basicSalary3 = 5000;      //普通员工基本工资
        // 经理的工资计算方式
        salary1 = basicSalary1  + basicSalary1 *0.5 + 200 - leaveDays*(basicSalary1/30);
        System.out.println("经理实发工资为："+salary1);

        // 董事的工资计算方式
        salary2 = basicSalary2  + basicSalary2 *0.08 + 2000 -
leaveDays*(basicSalary2/30);
        System.out.println("董事实发工资为："+salary2);

        // 普通员工的工资计算方式
        salary3 = basicSalary3 + basicSalary3 *0.1 + 1000 -
leaveDays*(basicSalary3/30);
        System.out.println("普通员工实发工资为："+salary3);
    }
}
```

THANKS