第 5 章

CHAPTER 5

数 据 类 型

在声明变量或常量时会用到数据类型，在前面已经用到一些数据类型，如 int、double 和 String 等。Java 语言的数据类型分为基本数据类型和引用数据类型。

5.1 基本数据类型

基本数据类型表示简单的数据，基本数据类型分为 4 大类，共 8 种数据类型。

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

基本数据类型如图 5-1 所示，其中整数类型、浮点类型和字符类型都属于数值类型，它们之间可以互相转换。

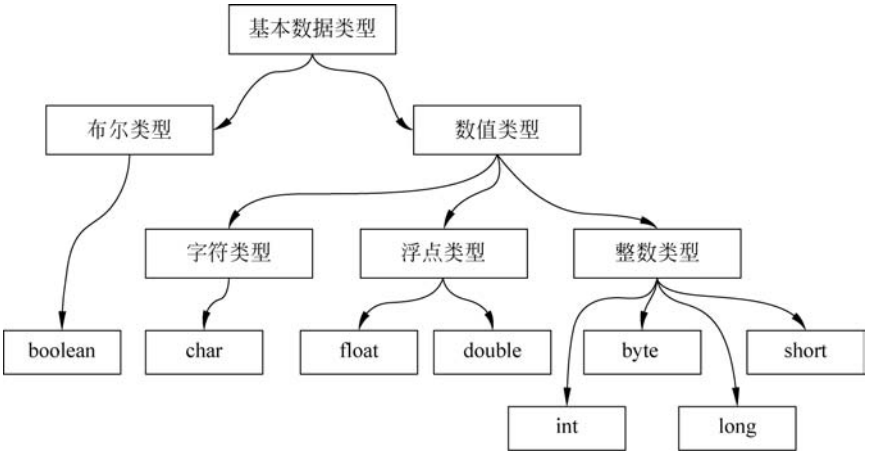

图 5-1 基本数据类型

5.2 整数类型

从图 5-1 可见，Java 中整数类型包括 byte、short、int 和 long，它们之间的区别仅是宽度和范围的不同。Java 中整数都有符号，与 C 语言不同，没有无符号的整数类型。

Java的数据类型是跨平台的(与平台无关),无论计算机系统是32位还是64位,byte类型整数都是1字节(8位)。这些整数类型的宽度和范围如表5-1所示。

表5-1 整数类型

整数类型	宽 度	取值范围
byte	1字节(8位)	$-128\sim127$
short	2字节(16位)	$-2^{15}\sim2^{15}-1$
int	4字节(32位)	$-2^{31}\sim2^{31}-1$
long	8字节(64位)	$-2^{63}\sim2^{63}-1$

Java语言的整数类型默认是int类型,例如16表示为int类型常量,而不是short或byte,更不是long,long类型需要在数值后面加l(小写英文字母)或L(大写英文字母)。示例代码如下:

```
public class HelloWorld {

      public static void main(String[] args) {
        // 声明整数变量
        // 输出一个默认整数常量
        System.out.println("默认整数常量   =   " + 16);  ①
        byte a = 16;                                    ②
        short b = 16;                                   ③
        int c = 16;                                     ④
        long d = 16L;                                   ⑤
        long e = 16l;                                   ⑥

        System.out.println("byte整数      =   " + a);
        System.out.println("short整数     =   " + b);
        System.out.println("int整数       =   " + c);
        System.out.println("long整数      =   " + d);
        System.out.println("long整数      =   " + e);

      }
}
```

上述代码多次用到了16整数,但它们是有所区别的。其中,代码第①行中的16是默认整数类型,即int类型常量;代码第②行中的16是byte整数类型;代码第③行中的16是short类型;代码第④行中的16是int类型;代码第⑤行中的16后面加了L,这说明是long类型整数;代码第⑥行中的16后面加了l(小写英文字母),这也是long类型整数。

提示 在程序代码中,尽量不用小写英文字母l,因为它容易与数字1混淆,特别是在Java中表示long类型整数时应尽量少使用小写英文字母l,而是使用大写的英文字母L。例如,16L要比16l可读性更好。

5.3 浮点类型

浮点类型主要用来存储小数数值,也可以用来存储范围较大的整数。它分为浮点数(float)和双精度浮点数(double)两种,双精度浮点数所使用的内存空间比浮点数多,可表示

的数值范围与精确度也比较大。浮点类型说明如表 5-2 所示。

表 5-2 浮点类型

浮 点 类 型	宽 度
float	4 字节(32 位)
double	8 字节(64 位)

Java 语言的浮点类型默认是 double 类型,例如 0.0 表示 double 类型常量,而不是 float 类型。如果想要表示 float 类型,则需要在数值后面加 f 或 F。示例代码如下:

```
public class HelloWorld {

      public static void main(String[] args) {
        // 声明浮点数
        // 输出一个默认浮点常量
        System.out.println("默认浮点常量   =   " + 360.66);  ①
        float myMoney = 360.66f;                              ②
        double yourMoney = 360.66;                            ③
        final double PI = 3.14159d;                           ④

        System.out.println("float 整数    =   " + myMoney);
        System.out.println("double 整数   =   " + yourMoney);
        System.out.println("PI     =   " + PI);

      }
}
```

上述代码第①行中的 360.66 是默认浮点类型 double。代码第②行中的 360.66f 是 float 浮点类型,float 浮点类型以常量表示时,数值后面需要加 f 或 F。代码第③行中的 360.66 表示是 double 浮点类型。事实上 double 浮点数值后面也可以加字母 d 或 D,以表示是 double 浮点数。代码第④行是声明一个 double 类型常量,数值后面加了 d 字母。

5.4 数值表示方式

整数类型和浮点类型都表示数值类型,那么在给这些类型的变量或常量赋值时,应该如何表示这些数值呢?下面介绍数字和指数等的表示方式。

5.4.1 进制数字表示

如果为一个整数变量赋值,使用二进制数、八进制数和十六进制数表示,它们的表示方式分别如下:

- 二进制数:以 0b 或 0B 为前缀,注意 0 是阿拉伯数字,不要误认为是英文字母 o。
- 八进制数:以 0 为前缀,注意 0 是阿拉伯数字。
- 十六进制数:以 0x 或 0X 为前缀,注意 0 是阿拉伯数字。

例如,下面几条语句都是表示 int 整数 28。

```
int decimalInt = 28;
int binaryInt1 = 0b11100;
int binaryInt2 = 0B11100;
int octalInt = 034;
int hexadecimalInt1 = 0x1C;
int hexadecimalInt2 = 0X1C;
```

5.4.2 指数表示

进行数学计算时往往会用到指数表示的数值。如果采用十进制表示指数,则需要使用大写或小写的 e 表示幂,e2 表示 10^2。

采用十进制指数表示的浮点数示例代码如下:

```
double myMoney = 3.36e2;
double interestRate = 1.56e-2;
```

其中 3.36e2 表示的是 3.36×10^2,1.56e-2 表示的是 1.56×10^{-2}。

5.5 字符类型

字符类型表示单个字符,Java 中 char 关键字用于声明字符类型,Java 中的字符常量必须是用单引号括起来的单个字符,示例代码如下:

```
char c = 'A';
```

Java 字符采用双字节 Unicode 编码,占 2 字节(16 位),因而可用十六进制(无符号的)编码形式表示,它们的表现形式是\un,其中 n 为 16 位十六进制数,所以'A'字符也可以用 Unicode 编码'\u0041'表示。如果对字符编码感兴趣,可以到维基百科(https://zh.wikipedia.org/wiki/Unicode 字符列表)查询。

示例代码如下:

```
public class HelloWorld {

      public static void main(String[] args) {

        char c1 = 'A';
        char c2 = '\u0041';
        char c3 = '花';

        System.out.println(c1);
        System.out.println(c2);
        System.out.println(c3);
      }
}
```

上述代码变量 c1 和 c2 都是保存的'A',所以输出结果如下:

```
A
A
花
```

提示 字符类型也属于数值类型，可以与 int 等数值类型进行数学计算或进行转换。这是因为字符类型在计算机中保存的是 Unicode 编码，双字节 Unicode 的存储范围为 \u0000～\uFFFF，所以 char 类型取值范围为 $0 \sim 2^{16}-1$。

在 Java 中，为了表示一些特殊字符，前面要加上反斜杠(\)，这称为字符转义。常见的转义符含义如表 5-3 所示。

表 5-3 常见的转义符含义

字符表示	Unicode 编码	说　明
\t	\u0009	水平制表符 tab
\n	\u000a	换行
\r	\u000d	回车
\"	\u0022	双引号
\'	\u0027	单引号
\\	\u005c	反斜杠

示例代码如下：

```
//在 Hello 和 World 插入制表符
String specialCharTab1 = "Hello\tWorld.";
//在 Hello 和 World 插入制表符，制表符采用 Unicode 编码\u0009 表示
String specialCharTab2 = "Hello\u0009World.";
//在 Hello 和 World 插入换行符
String specialCharNewLine = "Hello\nWorld.";
//在 Hello 和 World 插入回车符
String specialCharReturn = "Hello\rWorld.";
//在 Hello 和 World 插入双引号
String specialCharQuotationMark = "Hello\"World\".";
//在 Hello 和 World 插入单引号
String specialCharApostrophe = "Hello\'World\'.";
//在 Hello 和 World 插入反斜杠
String specialCharReverseSolidus = "Hello\\World.";

System.out.println("水平制表符 tab1: " + specialCharTab1);
System.out.println("水平制表符 tab2: " + specialCharTab2);
System.out.println("换行: " + specialCharNewLine);
System.out.println("回车: " + specialCharReturn);
System.out.println("双引号: " + specialCharQuotationMark);
System.out.println("单引号: " + specialCharApostrophe);
System.out.println("反斜杠: " + specialCharReverseSolidus);
```

输出结果如下：

```
水平制表符 tab1: HelloWorld.
水平制表符 tab2: HelloWorld.
换行: Hello
World.
回车: Hello
```

```
World.
双引号: Hello"World".
单引号: Hello'World'.
反斜杠: Hello\World.
```

5.6 布尔类型

在 Java 语言中声明布尔类型的关键字是 boolean,它只有两个值:true 和 false。

提示 在 C 语言中布尔类型是数值类型,它有两个取值,即 1 和 0。而在 Java 中的布尔类型取值不能用 1 和 0 替代,也不属于数值类型,不能与 int 等数值类型之间进行数学计算或类型转化。

示例代码如下:

```
boolean isMan = true;
boolean isWoman = false;
```

如果试图给它们赋值 true 和 false 之外的常量,示例代码如下:

```
boolean isMan = 1;
boolean isWoman = 'A';
```

则发生类型不匹配编译错误。

5.7 数值类型相互转换

学习了前面的数据类型后,大家会思考一个问题,数据类型之间是否可以转换呢?数据类型的转换情况比较复杂。基本数据类型中数值类型之间可以互相转换,布尔类型不能与它们之间进行转换。但有些不兼容类型之间,如 String(字符串)转换为 int 整数等,可以借助于一些类的方法实现。本节只讨论数值类型的互相转换。

从图 5-1 中可见,数值类型包括 byte、short、char、int、long、float 和 double,这些数值类型之间的转换有两个方向:自动类型转换和强制类型转换。

5.7.1 自动类型转换

自动类型转换就是需要类型之间转换是自动的,不需要采取其他手段,总的原则是小范围数据类型可以自动转换为大范围数据类型。数据类型转换顺序如图 5-2 所示,从左到右是自动的。

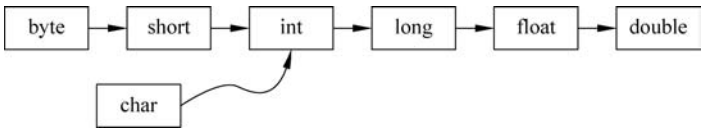

图 5-2 数据类型转换顺序

注意 如图 5-2 所示，char 类型比较特殊，char 自动转换为 int、long、float 和 double，但 byte 和 short 不能自动转换为 char，而且 char 也不能自动转换为 byte 或 short。

自动类型转换不仅发生在赋值过程中，在进行数学计算时也会发生自动类型转换，在运算中往往是先将数据类型转换为同一类型，然后再进行计算，计算规则如表 5-4 所示。

表 5-4 计算过程中自动类型转换规则

操作数 1 类型	操作数 2 类型	转换后的类型
byte、short、char	int	int
byte、short、char、int	long	long
byte、short、char、int、long	float	float
byte、short、char、int、long、float	double	double

示例代码如下：

```
// 声明整数变量
byte byteNum = 16;
short shortNum = 16;
int intNum = 16;
long longNum = 16L;

// byte 类型转换为 int 类型
intNum = byteNum;
// 声明 char 变量
char charNum = '花';
// char 类型转换为 int 类型
intNum = charNum;

// 声明浮点变量
// long 类型转换为 float 类型
float floatNum = longNum;
// float 类型转换为 double 类型
double doubleNum = floatNum;

//表达式计算后类型是 double
double result = floatNum * intNum + doubleNum / shortNum;  ①
```

上述代码第①行中的表达式 floatNum * intNum + doubleNum / shortNum 进行数学计算，该表达式是由 4 个完全不同的数据类型组成，范围最大的是 double，所以在计算过程中它们先转换成 double，最后的结果类型是 double。

5.7.2 强制类型转换

在数值类型转换过程中，除了需要自动类型转换外，有时还需要强制类型转换，强制类型转换是在变量或常量之前加上“(目标类型)”实现。示例代码如下：

```
//int 型变量
```

```
int i = 10;
//把 int 变量 i 强制转换为 byte
byte b = (byte) i;
```

上述代码(byte) i 表达式实现强制类型转换。强制类型转换主要用于大宽度类型转换为小宽度类型情况,如把 int 转换为 byte。示例代码如下:

```
//int 型变量
int i = 10;
//把 int 变量 i 强制转换为 byte
byte b = (byte) i;
int i2 = (int)i;        ①
int i3 = (int)b;        ②
```

上述代码第①行是将 int 类型的 i 变量强制转换为 int 类型,这显然没有必要,但是语法也是允许的。代码第②行是将 byte 类型的 b 变量强制转换为 int 类型,从图 5-2 中可见这个转换是自动的,不需要强制转换。本例中这个转换没有实际意义,但有时为了提高精度需要这种转换。示例代码如下:

```
//int 类型变量
int i = 10;
float c1 = i / 3;          ①
System.out.println(c1);   ②
//把 int 变量 i 强制转换为 float
float c2 = (float)i / 3;  ③
System.out.println(c2);   ④
```

输出结果如下:

```
3.0
3.3333333
```

比较上述代码输出结果发现 c1 和 c2 变量小数部分差别是比较大的,这种差别在一些金融系统中是不允许的。在代码第①行 i 除以 3 中结果是小数,但由于两个操作数都是整数 int 类型,小数部分被截掉了,结果是 3,然后再赋值给 float 类型的 c1 变量,最后 c1 保存的是 3.0。为了防止两个整数进行除法等运算导致小数位被截掉问题,可以将其中一个操作数强制类型转换为 float,见代码第③行,这样计算过程中操作数是 float 类型,结果是 float 不会截掉小数部分。

再看一个强制类型转换与精度丢失的示例。

```
long yourNumber = 6666666666L;
System.out.println(yourNumber);
int myNuber = (int)yourNumber;
System.out.println(myNuber);
```

输出结果如下:

```
6666666666
-1923267926
```

从上述代码输出结果可见，经过强制类型转换后，原本的 6666666666L 变成了负数。当大宽度数值转换为小宽度数值时，大宽度数值的高位被截掉，这样就会导致数据精度丢失。除非大宽度数值的高位没有数据，就是这个数比较小的情况，例如将 6666666666L 换为 6L 就不会丢失精度。

5.8 引用数据类型

在 Java 中除了 8 种基本数据类型外，其他数据类型全部都是引用(reference)数据类型，引用数据类型用来表示复杂数据类型，如图 5-3 所示，包含类、接口、枚举和数组声明的数据类型。

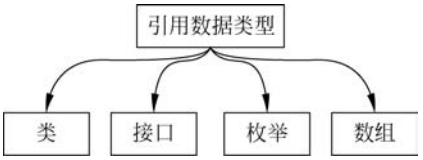

图 5-3 引用数据类型

提示 Java 中的引用类型，相当于 C 等语言中指针(pointer)类型，引用事实上就是指针，是指向一个对象的内存地址。引用类型变量中保持的是指向对象的内存地址。很多资料上提到 Java 不支持指针，事实上是不支持指针计算，而指针类型还是保留了下来，只是在 Java 中称为引用类型。

引用数据类型示例代码如下：

```
int x = 7;                ①
int y = x;                ②

String str1 = "Hello";    ③
String str2 = str1;       ④
str2 = "World";           ⑤
```

上述代码声明了两个基本数据类型(int)和两个引用数据类型(String)。当程序执行完第①和第②行代码后，x 值为 7，x 赋值给 y，这时 y 的值也是 7，它们的保存方式如图 5-4 所示，x 和 y 两个变量值都是 7，但是它们之间是独立的，任何一个变化都不会影响另一个。

当程序执行完第③行时，字符串 Hello 对象被创建，保存到内存地址 0x12345678 中，str1 是引用类型变量，它保存的是内存地址 0x12345678，这个地址指向 Hello 对象。

当程序执行完第④行时，str1 变量内容(0x12345678)被赋值给 str2(引用类型变量)，这样 str1 和 str2 保存了相同的内存地址，都指向 Hello 对象。如图 5-4 所示，此时 str1 和 str2 本质上是引用一个对象，通过任何一个引用都可以修改对象本身。

当程序执行完第⑤行时，字符串 World 对象被创建，保存到内存地址 0x23455678 中，地址保存到 str2 变量中，此时，str1 和 str2 不再指向相同内存地址，如图 5-5 所示。

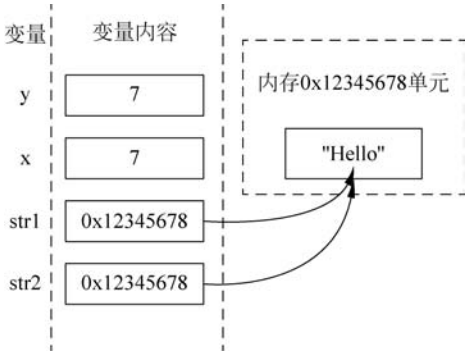

图 5-4 引用数据类型赋值过程 1

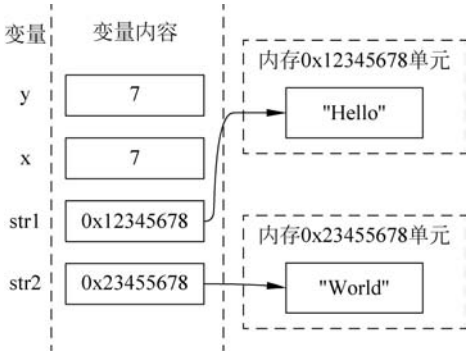

图 5-5 引用数据类型赋值过程 2

5.9 本章小结

本章主要介绍了 Java 中的数据类型，读者需要重点掌握基本数据类型，理解基本数据类型与引用数据类型的区别，熟悉数值类型如何互相转换。

5.10 同步练习

选择题

1. 下列哪些行代码在编译时不会出现警告或错误信息？（ ）

   A. float f = 1.3；　　B. char c = "a"；

   C. byte b = 257；　　D. Boolean b = null；

   E. Int I = 10；

2. 下列选项中 byte 的取值范围有哪些？（ ）

   A. −128～127　　B. −256～256

   C. −255～256　　D. 依赖于计算机本身硬件

3. 下列选项中正确的表达式有哪些？（ ）

   A. byte = 128；　　B. Boolean = null；

   C. long l = 0xfffL；　　D. double = 0.9239d；

4. 下列选项中哪些不是 Java 的基本数据类型？（ ）

   A. short　　B. Boolean　　C. Int　　D. float