CHAPTER 3

第 3 章 Java 语言基础

本章将对 Java 语言的基本语法成分进行介绍，包括标识符、数据类型、表达式、语句、程序流控制与数组等。

3.1 标识符与数据类型

3.1.1 Java 基本语法

1. 语句与语句块

Java 中是以“；”为语句的分隔符。一个语句可写在连续的若干行内。例如下面的两个语句是等价的：

```
x = a + b + c + d + e;
x = a + b+
      d + e;
```

一对大括号“{”和“}”包含的一系列语句称为语句块。语句块可以嵌套，即语句块中可以嵌套子语句块。

Java 源程序中允许在变量、标识符、表达式、语句等代码元素之间出现任意数量的空白。空格、Tab 键和换行符都是空白。在程序中适当使用空白可以使程序层次清晰，增加程序的可读性。

2. 注释

程序中适当加入注释，会增加程序的可读性。程序中允许加空白的地方就可以写注释，编译器将忽略所有注释。

Java 中有如下 3 种注释。

(1) //

注释一行。表示从//开始到行尾都是注释文字。

(2) /*　　*/

注释一行或多行。表示/*和*/之间的所有内容都是注释。

(3) /** *

文档注释。表示在/** 和 */之间的文本,将自动包含在用 javadoc 命令生成的 HTML 格式的文档中。javadoc 是 JDK 中 API 文档生成器。该工具解析一组 Java 源文件中的声明与文档注释,生成一组 HTML 页面描述这些源程序中定义的类、内部类、接口、构造方法、方法与属性。JDK 的 API 文档就是用 javadoc 工具生成的。

3.1.2　标识符

1. 标识符的定义规则

在 Java 语言中,采用标识符对变量、类和方法进行命名。对标识符的定义需要遵守以下规则。

- 标识符是以字母,"_"(下划线),或"$"开始的一个字符序列。
- 数字不能作为标识符的第一个字符。
- 标识符不能是 Java 语言的关键字,但可用关键字作为标识符的一部分。
- 标识符大小写敏感,且长度没有限定。

例如: username,user_name,_sys_var,$change,thisOne 均是合法的标识符。

Java 不采用通常计算机系统采用的 ASCII 代码集,而是采用 Unicode 这样一个国际标准字符集。在这种字符集中,每个字母用 16 位表示,整个字符集中共包含 65536 个字符。其中,ASCII 代码集中的字符如英文字母 A～Z,a～z 和数字 0～9 在 Unicode 字符集中还是用十六进制的 0x0041～0x005a,0x0061～0x007a 和 0x30～0x39 来表示,以表示对 ASCII 码的兼容。另外,Unicode 字符集涵盖了像汉字、日文、朝鲜文、德文、希腊文等多国语言中的符号。这样,Java 中的"字母"和"数字"这两个术语涵盖的范围要广得多。其中,字母被定义成 A～Z,a～z 或国际语言中相当于一个字母的任何 Unicode 字符。

一般情况下,标识符中使用的字母包括下面几种。

(1) A～Z。

(2) a～z。

(3) Unicode 字符集中序号大于等于 0x00c0 的所有国际语言中相当于一个字母的任何 Unicode 字符。

为了准确起见,可以使用 Character 类中的 isJavaIdentifierStart(char ch)方法和 isJavaIdentifierPart(char ch)方法测试参数变量 ch 中的 Unicode 字符是否可以作为标识符的开始字符或后续字符。

2. 标识符风格约定

(1) 对于变量名和方法名,_和 $ 不作为标识符的第一个字符,因为这两个字符对于内部类具有特殊含义。

(2) 类名、接口名、变量名和方法名采用大小写混合的形式,即每个单词的首字母大写,其余小写。但变量名和方法名第一个单词的首字母小写,例如 anyVariableWord。而类名和接口名第一个单词的首字母大写,例如 HelloWorld。

(3) 常量名完全大写,并且用下划线_作为标识符中各个单词的分隔符,例如 MAXIMUM_SIZE。

(4) 方法名应该使用动词,类名与接口名应该使用名词。例如:

```
class Account                    //类名
interface AccountBook            //接口名
balanceAccount()                 //方法名
```

(5) 变量名应该能够表示一定的含义,因此应尽量不使用单个字符作为变量名。但临时性变量如循环控制变量可以采用 i,j,k 等。

3.1.3 关键字

在表 3-1 中列出了 Java 的关键字,这些单词是 Java 语言的保留字。Java 编译器在词法扫描时,需要区分关键字和一般的标识符,因此,用户自定义的标识符不能与这些关键字重名,否则会产生编译错误。另外,true,false 和 null 虽然不是关键字,但也被 Java 保留,同样不能用来定义标识符。

表 3-1 Java 语言的关键字

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

3.1.4 基本数据类型

Java 语言定义了 4 类共 8 种基本类型。

- 逻辑型: boolean。
- 文本型: char。
- 整型: byte,short,int 和 long。
- 浮点型: double 和 float。

1. 逻辑型——boolean

boolean 类型数据有两种取值 true 和 false,在机器中只占 1 位。boolean 型变量的默认初始值为 false。例如:

```
boolean truth = true;            //定义 truth 为 boolean 类型,且初始值为 true
```

注意:与其他高级语言不同,Java 中的布尔值和数字之间不能来回转换,即 false 和 true 不对应于任何零或非零的整数值。

2. 文字型——char 和 String

char 是文字型的基本数据类型,而 String 是类不是基本类型,但很常用,所以在此一并介绍。

(1) char

是一个 16 位的 Unicode(国际码)字符,用单引号引上。例如:

```
char mychar = 'Q';                    //mychar 变量的初值被置为 Q 字符对应的 16 位 Unicode 值
```

字符型变量的默认初始值是\u0000。

Unicode 字符集可以支持各类文字的字符,总数达 34168 个字符。通过将国际标准的 Unicode 字符集作为字符变量的取值范围,使 Java 能极为方便地处理各种语言,例如可以将汉字作为字符型变量的值,这为程序的国际化提供方便。

一些控制字符不能直接显示,Java 与 C/C++一样,利用转义序列来表示这些字符。还有一种直接以八进制或十六进制代表字符值的表示方法,在反斜杠后跟 3 位八进制数字或在反斜杠后跟 u,后面再跟 4 位十六进制数字都可代表一个字符常量,如"\141"和"\u0061"都代表字符常量"a"。表 3-2 是 Java 中的转义字符。

表 3-2　Java 中的转义字符序列

转义字符	描　　述	转义字符	描　　述
\ddd	1～3 位八进制数所表示的字符	\r	回车
\uxxxx	1～4 位十六进制数所表示的字符	\n	换行
\'	单引号字符	\f	走纸换页
\"	双引号字符	\b	退格
\\	反斜杠字符	\t	水平制表(Tab 键)

一般情况下,char 类型的十六进制 Unicode 编码值可自动转换成等值的 int 类型值,并可与 int 类型数值进行运算。而 int 类型到 char 类型需要通过强制类型转换,如例 3-1 所示。

例 3-1　char 类型的值到 int 类型的转换。

```
public class CharToInt{
    public static void main(String args[]){
        int intResult,intVar = 10;
        char charVar = '语';
        intResult = intVar + charVar;
        System.out.println("The char is :  " + charVar);
        System.out.println("The char's Unicode is :  \\u" + Integer.toHexString(charVar));
        System.out.println ("The int value corresponding to the char is :  "
                            + new Integer(charVar).toString());
        System.out.println("Int " + intVar + " adds the char, the result is :  " + intResult);
    }
}
```

例 3-1 中,字符变量 charVar 的值是"语"字。程序中输出了该字符的 Unicode 值以及 Unicode 对应的十进制数值,并打印输出了 charVar 与一个 int 型变量做加法运算后的值。例 3-1 的运行结果如图 3-1 所示。

图 3-1　例 3-1 的运行结果

(2) String

String 不是基本类型而是一个类。字符串在 Java 中是对象,在 Java 中有两个类可以表达字符串: String 和 StringBuffer。一个 String 的对象表示一个字符串,字符串要放在双引号("")中。字符串中的字符也是 Unicode。与 C 和 C++不同,Java 中的字符串不以'\0'为结束符。例如:

```
//声明了两个字符串变量并进行初始化
String greeting = "good morning! \n";
String anotherGreeting = "How are you?";
```

注意: String 对象表示的字符串是不能修改的。如果需要对字符串修改,应该使用 StringBuffer 类。

3. 整数类型: byte,short,int 和 long

Java 提供了 4 种整数类型: byte,short,int,long。由于 char 类型的值可以转换为 int 型,所以表 3-3 将这 4 种类型和 char 类型的长度与取值范围一起列出。

表 3-3　Java 整数类型和 char 类型长度与取值范围

类　型	长　度	取 值 范 围
byte	8 位	$-2^7 \sim 2^7-1$,即 −128～127
short	16 位	$-2^{15} \sim 2^{15}-1$,即 −32 768～32 767
int	32 位	$-2^{31} \sim 2^{31}-1$,即 −2 147 483 648～2 147 483 647
long	64 位	$-2^{63} \sim 2^{63}-1$,即 −9 223 372 036 854 775 808～9 223 372 036 854 775 807
char	16 位	'\u0000' ～ '\uffff',即 0～65 535

注意: Java 中所有的整数类型都是有符号的整数类型,Java 没有无符号整数类型。

所有整型变量的默认初始值为 0。

int 类型是最常使用的一种整数类型。它所表示的数据范围足够大,而且适合于 32 位和 64 位处理器。但对于大型计算,常会遇到很大的整数,超出 int 类型所表示的范围,这时要使用 long 类型。

由于不同的机器对于多字节数据的存储方式不同,可能是从低字节向高字节存储,也可能是从高字节向低字节存储,这样,在分析网络协议或文件格式时,为了解决不同机器上的字节存储顺序问题,用 byte 类型来表示数据是比较合适的。而通常情况下,由于其表示的数据范围很小,容易造成溢出,因此要尽量少使用。

如果一个数超出了计算机的表达范围,称为溢出;如果超出最大值,称为上溢;如果超过最小值,称为下溢。将一个整型类型数的最大值加 1 后,产生上溢而变成了同类型的最小值;最小值减 1 后,产生下溢而变成了同类型的最大值。

整型常量可以有 3 种形式：十进制、八进制和十六进制。八进制整数以 0 为前导，十六进制整数以 0X 或 0x 为前导。整型常量的默认类型是 int。对于 long 型常量，则要在数值后加 L 或 l，建议使用 L，因为小写的 L 看起来与数字 1 很相像。表 3-4 是整型常量示例。

表 3-4　整型常量示例

类　型	十进制	八进制	十六进制
int	24	030	0x18
long	24L	030L	0x18L

4. 浮点型：float 和 double

Java 提供了两种浮点类型：float 和 double，如表 3-5 所示。

表 3-5　Java 浮点类型长度与取值范围

类　型	长　度	取 值 范 围
float	32 位	1.4e－45～3.402 823 5e＋38
double	64 位	4.9e－324～1.797 693 134 862 315 7e＋308

双精度类型 double 比单精度类型 float 具有更高的精度和更大的表示范围，但 float 类型具有速度快、占用内存小的优点。

浮点型变量的默认初值是 0.0。浮点数在运算过程中不会因溢出而导致异常处理。如果出现下溢，则结果为 0.0；如果上溢，则结果为正或负无穷大。此外，如果出现数学上没有定义的值，如 0.0/0.0，则结果将被视为非法数，表示为 NaN(Not-a-Number)。

浮点类型的常量默认是 double 类型。如 3.14 是 double 型，在机器内存中占 64 位。浮点型常量还可以用科学记数法表达，用 E 或 e，如 6.02×10^{23} 可表达为 6.02e23。另外可以用 F 或 f 表示 float 类型的常量，如 6.02e23F；用 D 或 d 表示 double 型的常量，如 2.718D。

下面的例 3-2 说明 Java 基本数据类型的使用，例 3-3 显示了 Java 定义基本数据类型相关的常量值，包括各种类型的最大值、最小值以及浮点型中的无穷大与非法数的定义等。

例 3-2　基本数据类型的声明与赋值。

```
public class Assign {
    public static void main (String args[]) {
        int  x, y;                          //声明整型变量
        float  z = 3.414f ;                 //声明并赋值 float 型变量
        double  w = 3.1415;                 //声明并赋值 double 型变量
        boolean  truth = true;              //声明并赋值 boolean 型变量
        char  c;                            //声明字符变量
        String  str;                        //声明 String 类变量
        String  str1 = "bye";               //声明并赋值 String 类变量
        c = 'A';                            //给字符变量赋值
        str = "Hi out there";               //Java 中所有字符串都作为 String 类的对象实现
                                            //所以可采用这种方式给 String 变量赋值
        x = 6;
        y = 1000;                           //给 int 型变量赋值
        System.out.println("x = " + x);
        System.out.println("y = " + y);
```

```
        System.out.println("z = " + z);
        System.out.println("w = " + w);
        System.out.println("truth = " + truth);
        System.out.println("c = " + c);
        System.out.println("str = " + str);
        System.out.println("str1 = " + str1);
    }
}
```

例 3-2 的运行结果如下：

```
x = 6
y = 1000
z = 3.414
w = 3.1415
truth = true
c = A
str = Hi out there
str1 = bye
```

例 3-3 输出 Java 基本数据类型相关的一些常量。

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

        //输出 byte 型的最大值与最小值
        System.out.println("Byte.MAX_VALUE = " + Byte.MAX_VALUE);
        System.out.println("Byte.MIN_VALUE = " + Byte.MIN_VALUE);
        System.out.println();

        //输出 short 型的最大值与最小值
        System.out.println("Short.MAX_VALUE = " + Short.MAX_VALUE);
        System.out.println("Short.MIN_VALUE = " + Short.MIN_VALUE);
        System.out.println();

        //输出 int 型的最大值与最小值
        System.out.println("Integer.MAX_VALUE = " + Integer.MAX_VALUE);
        System.out.println("Integer.MIN_VALUE = " + Integer.MIN_VALUE);
        System.out.println();

        //输出 long 型的最大值与最小值
        System.out.println("Long.MAX_VALUE = " + Long.MAX_VALUE);
        System.out.println("Long.MIN_VALUE = " + Long.MIN_VALUE);
        System.out.println();

        //输出 float 型的最大值与最小值
        System.out.println("Float.MAX_VALUE = " + Float.MAX_VALUE);
        System.out.println("Float.MIN_VALUE = " + Float.MIN_VALUE);
        System.out.println();
```

```
        //输出 double 型的最大值与最小值
        System.out.println("Double.MAX_VALUE = " + Double.MAX_VALUE);
        System.out.println("Double.MIN_VALUE = " + Double.MIN_VALUE);
        System.out.println();

        //输出 float 型的正无穷大与负无穷大
        System.out.println("Float.POSITIVE_INFINITY = " + Float.POSITIVE_INFINITY);
        System.out.println("Float.NEGATIVE_INFINITY = " + Float.NEGATIVE_INFINITY);
        System.out.println();

        //输出 double 型的正无穷大与负无穷大
        System.out.println("Double.POSITIVE_INFINITY = " + Double.POSITIVE_INFINITY);
        System.out.println("Double.NEGATIVE_INFINITY = " + Double.NEGATIVE_INFINITY);
        System.out.println();

        //输出 float 型 0/0
        System.out.println("Float.NaN = " + Float.NaN);
        System.out.println();

        //输出 double 型 0/0
        System.out.println("Double.NaN = " + Double.NaN);
        System.out.println();
    }
}
```

例 3-3 的运行结果如图 3-2 所示。

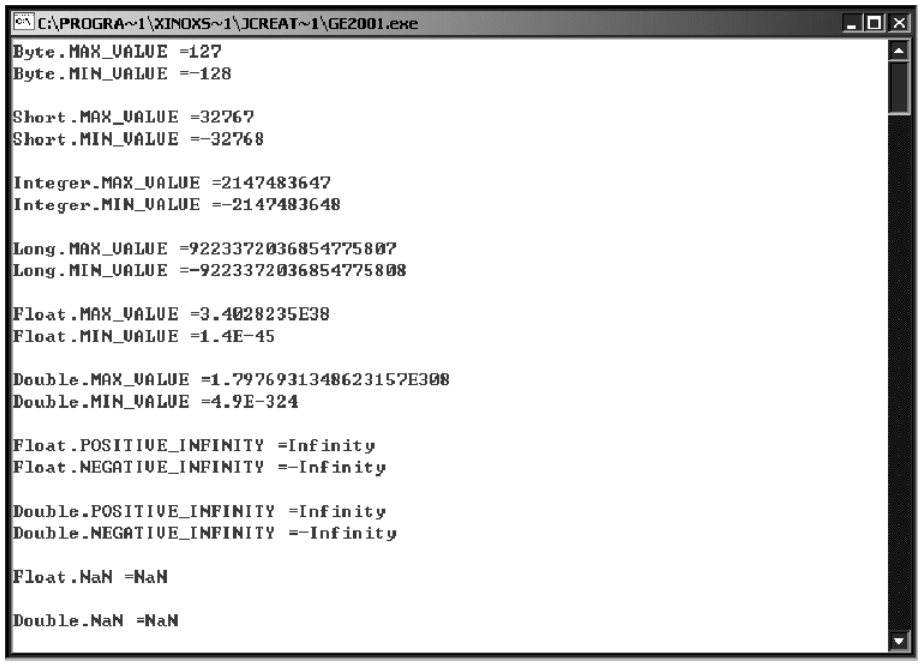

图 3-2　例 3-3 的运行结果

3.1.5 复合数据类型

3.1.4 节介绍的 4 类 8 种基本数据类型,是 Java 的内置类型。在很多应用程序的开发中,仅使用这几种类型是远远不够的。例如,如果要处理日期,则要独立声明 3 个整数,分别代表日、月、年:

```
int  day, month, year ;
```

该语句有个含义:表明 day,month,year 的类型是整数类型,另外还为这些整数分配了存储空间。用这种方式表示日期,虽然容易理解,但存在明显不足。首先如果程序要处理多个日期,则需声明很多变量。例如要保存两个日期,则需如下定义:

```
int  day1, month1, year1;
int  day2, month2, year2;
```

这种方法因使用了多个变量而使程序显得混乱,并且容易出错。另外,这种定义方法忽略了日期的年、月、日之间的联系,把这些变量孤立起来,它们之间将毫无联系,各自的取值范围将只受整数类型取值范围的限制。而在概念上,日、月、年之间是有联系的,它们是同一事物"日期"的各个组成部分,3 部分的取值是有约束的。例如,日的取值范围为 0～31,月的取值范围是 1～12,并且对于一般的月份,日的取值上限又有 30 和 31 的差别,而 2 月份的天数又与年有关系。如果要在程序中维护日期的 3 个元素之间的约束,则需要编写程序,并注意在使用日期的这些变量时启动约束检查。这样,不仅增加了程序编写的复杂度,还可能由于程序员的疏忽而造成错误。

如果程序设计语言能够允许用户定义新的数据类型,则上述问题就可以得到很好的解决。而实际上目前很多种语言都具有这种扩展能力,有些语言提供结构或记录来定义新的类型。一般地将用户定义的新类型称为复合数据类型。Java 是一种面向对象的语言,基于面向对象概念,以类和接口的形式定义新类型。因此在 Java 中类和接口是两种用户定义的复合数据类型。

在上述日期的例子中,将日期相关的 3 个变量进行封装,用 class 关键字创建了一个日期类。这个用 class 定义的日期类在 Java 语言中是一种新创建的数据类型。日期类的定义如下:

```
class MyDate{
    int day ;
    int month;
    int year;
}
```

使用语言内置类型定义变量时,因为每种类型都是预定义的,所以无须程序员指定变量的存储结构。例如通过整型变量 day 的定义,Java 运行系统就可以知道要分配多大的空间,并能够解释所存储的内容。对于新的数据类型,需要指定该类型所需的存储空间以及如何解释这些空间中的内容。新类型不是通过字节数指定空间大小,也不是通过位的顺序和含义定义该存储空间的含义,而是通过包含在类定义中的已有数据类型来提供这些信息的。例如,上述类 MyDate 的定义,表明为了表示一个日期,需要保存 3 个整数所需的空间,并且这些整数分别表示了一个日期中的日、月、年。

复合数据类型由程序员在源程序中定义。一旦有了定义，该类型就可像其他类型一样使用。可以声明 MyDate 类的变量，并且日期的年、月、日 3 部分也都由该变量表示。例如：

```
MyDate  a,b;
```

对于一个日期的年、月、日各组成部分的使用，都是通过 MyDate 类型的 a，b 变量进行，例如：

```
a.day = 30;
a.month = 12;
a.year = 1999;
```

在定义了 MyDate 类后，对于一个日期的定义只需要声明一个变量，并且日期中的 3 个组成部分被封装为一个有机的整体，它们之间的取值约束关系可以通过在 MyDate 类内部定义方法实现，操纵 MyDate 变量的程序员不需关心这些问题。因此在 Java 中使用类或接口这样的复合数据类型，不仅反映了现实世界事物的本质形态，还使程序简练并且更可靠。

3.1.6　基本类型变量与引用类型变量

Java 中数据类型分为两大类：基本数据类型与复合类型。相应地，变量也有两种类型：基本类型与引用类型。Java 的 8 种基本类型的变量称为基本类型变量，而类、接口和数组变量是引用类型变量。这两种类型变量的结构和含义不同，系统对它们的处理也不相同。

1. 基本类型与引用类型变量

- 基本类型(primitive type)

基本数据类型的变量包含了单个值，这个值的长度和格式符合变量所属数据类型的要求，可以是一个数字、一个字符或一个布尔值，如图 3-3(a)所示。例如一个整型值是 32 位的二进制补码格式的数据，而一个字符型的值是 16 位的 Unicode 字符格式的数据等。

- 引用类型(reference type)

引用类型变量的值与基本类型变量不同，变量值是指向内存空间的引用(地址)。所指向的内存中保存着变量所表示的一个值或一组值，如图 3-3(b)所示。

引用在其他语言中称为指针或内存地址。Java 语言与其他程序设计语言不同，不支持显式使用内存地址，而必须通过变量名对某个内存地址进行访问。

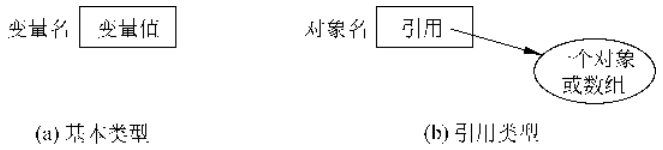

图 3-3　基本类型与引用类型的内存结构

2. 两种类型变量的不同处理

在 Java 中基本类型变量声明时，系统直接给该变量分配空间，因此程序中可以直接操作。例如：

```
int a;                                    //声明变量 a 的同时，系统给 a 分配了空间
a = 12;
```

引用类型(或称为引用型)变量声明时,只是给该变量分配引用空间,数据空间未分配。因此引用型变量声明后不能直接引用,下列第二条语句是错误的。

```
MyDate  today;
today.day = 14;                          //错误!因为 today 对象的数据空间未分配
```

引用型变量在声明后必须通过实例化开辟数据空间,才能对变量所指向的对象进行访问。通过对引用型变量声明与实例化语句的执行过程分析,可以理解系统对引用型变量的上述处理。例如有如下语句:

```
1       MyDate  today;
2       today = new Date();
```

第 1 条语句的执行,将给 today 变量分配一个保存引用的空间,如图 3-4(a)所示。第 2 条语句分两个步骤执行,首先执行 new Date(),给 today 变量开辟数据空间,如图 3-4(b)所示,然后再执行第 2 条语句中的赋值操作,结果如图 3-4(c)所示。

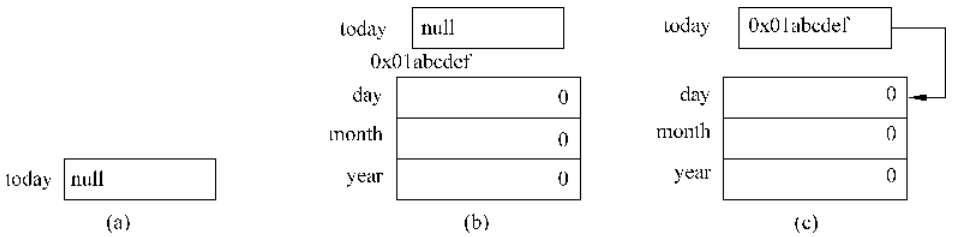

图 3-4　引用型变量的声明与实例化过程

3. 引用型变量的赋值

Java 中引用型变量之间的赋值是引用赋值。例如,下列语句执行后,内存的布局如图 3-5 所示。

```
MyDate  a, b;                    //在内存开辟两个引用空间
a = new MyDate();                //开辟 MyDate 对象的数据空间,并把该空间的首地址赋给 a
b = a;                           //将 a 存储空间中的地址写到 b 的存储空间中
```

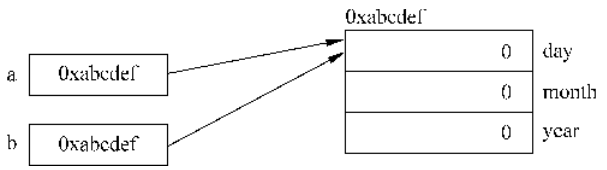

图 3-5　引用型变量之间赋值示例

3.2　表达式与语句

3.2.1　变量

1. 变量及作用域

变量有基本类型与引用类型。同时变量按照作用域来分,有局部变量、类成员变量、方