



# Java程序设计

---

## 第2章 Java语言的基本语法



## 2.1 标识符和关键字

Java语言中的类名、对象名、方法名、常量名和变量名统称为标识符。标识符由程序员定义，可以由字母、数字、下划线（\_）和（\$）符号组成，但是标识符的第一个字符不允许为数字，只允许为字母、下划线（\_）或（\$）符号。

在Java语言中还定义了一些专有词汇，统称为关键字，例如public、class、int等，它们都具有一种特定的含义，只能用于特定的位置，不能作为标识符使用。



# Java 关键字

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

在定义标识符时，不允许定义为表中列出的任一关键字



# Java标识符命名规则

为了提高程序的可读性，在定义标识符时，要尽量遵循“见其名知其意”的原则。Java标识符的具体命名规则如下：

- 一个标识符可以由几个单词连接而成，以表明它的意思。
- 对于类名，每个单词的首字母都要大写，其他字母则小写，例如RecordInfo。
- 对于方法名和变量名，与类名有些相似，除了第一个单词的首字母小写外，其他单词的首字母都要大写，例如getRecordName()。



# Java标识符命名规则

---

- 对于常量名，每个单词的每个字母都要大写，如果由多个单词组成，通常情况下单词之间用下划线（-）分隔，例如MAX-VALUE。
- 对于包名，每个单词的每个字母都要小写，例如com.frame。

**注意：**Java语言是区分字母大小写的，即Java不等于java



## 2.2 常量与变量

---

常量和变量在程序代码中随处可见，下面就具体讲解常量和变量的概念及使用要点，从而达到区别常量和变量的目的。

## 2.2.1 常量的概念及使用要点

所谓常量，就是值永远不允许被改变的量。  
如果要声明一个常量，就必须用关键字final修饰，  
声明常量的具体方式如下：

**final** 常量类型 常量标识符；

例如：

```
final int YOUTH-AGE;           // 声明一个int型常量
```

```
final float PIE;               // 声明一个float型常量
```

**注意：**按照Java命名规则，常量标识符所有的字符都要大写，各个单词之间用下划线 \_ 分隔

信息科学与工程学院



# 常量

在声明常量时，通常情况下立即为其赋值，即立即对常量进行初始化，声明并初始化常量的具体方式如下：

**final 常量类型 常量标识符 = 常量值;**

例如：

```
final int YOUTH-AGE = 18; // 声明int型常量，初始化为18
```

```
final float PIE = 3.14F; // 声明float型常量，初始化为3.14
```

**说明：**为float型常量赋值时，需要在数值的后面加上一个字母“F”或“f”。





# 常量

声明多个同一类型的常量，可以采用下面的形式：

`final 常量类型 常量1= 常量值1, 常量2= 常量值2, ... ..;`

例如：

`final int NUM1 = 14, NUM2 = 25, NUM3 = 36;`

**注意：**如果在声明常量时已经对其进行了初始化，则常量的值不允许再被修改

## 2.2.2 变量的概念及使用要点

所谓变量，就是值可以被改变的量。声明变量的具体方式如下：

变量类型 变量标识符；

例如：

```
String name; // 声明String型变量
```

```
int partyMemberAge; // 声明int型变量
```

**注意：**定义变量名时，按照Java的命名规则，第一个单词的首字母小写，其他单词的首字母大写，例如 “partyMemberAge”。



# 变量

在声明变量时，可以立即为其赋值，即立即对变量进行初始化，具体语法如下：

变量类型 变量标识符 = 变量值；

例如：

```
int partyMemberAge = 26; // 声明一个int型变量  
float money = 3150; // 声明float类型变量
```



# 同类型变量

如果需要声明多个同一类型的变量，也可以采用下面的形式：

变量类型 变量1, 变量2, 变量3;

变量类型 变量4= 变量值4, 变量5= 变量值5, 变量6= 变量值6;

例如：

```
int A, B, C; // 声明3个int型变量
```

```
int D = 4, E = 5, F = 6; // 声明并分别初始化3个int型变量
```

说明：变量区别于常量，它的值允许被改变。



## 2.3 数据类型

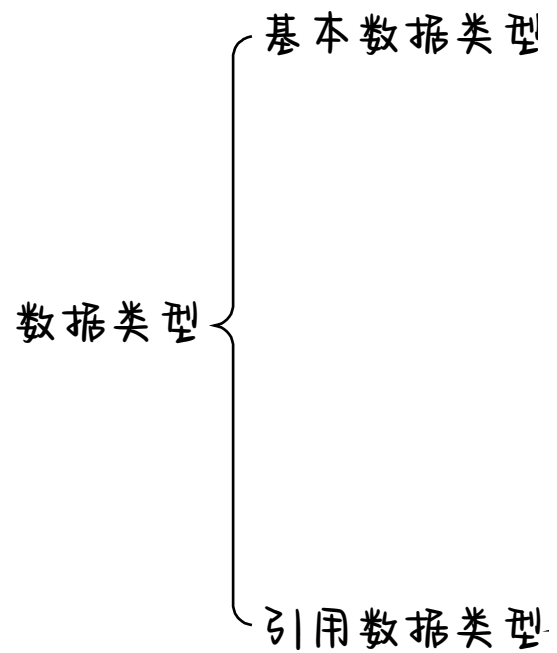
Java语言中的数据类型划分为两大类，分别是基本数据类型和引用数据类型。

其中基本数据类型由Java语言定义，不可再进行划分。基本数据类型的数据占用内存的大小固定，在内存中存入的是数值本身

引用数据类型在内存中存入的是引用数据的存放地址，并不是数据本身。

Java语言中的数据类型分类情况如下图所示：

# 数据类型





## 2.3.1 基本数据类型

基本数据类型分为：

- 整数型
- 浮点数值型
- 字符型
- 逻辑型（布尔型）

它们分别用来存储整数、小数、字符和布尔值，下面将依次讲解这4个基本数据类型的特征及使用方法。



# 整数型

声明为整数型的常量或变量用来存储整数，  
整数型包括：

- 字节型 (byte)
- 短整型 (short)
- 整型 (int)
- 长整型 (long)

这4个数据类型的区别是它们在内存中所占用的字节数不同，因此，它们所能够存储的整数的取值范围也不同。



# 整数占用内存大小以及取值范围

数据类型	关键字	内存字节	取值范围
字节型	byte	1个字节	-128~127
短整型	short	2个字节	-32768~32767
整型	int	4个字节	-2147483648~2147483647
长整型	long	8个字节	-9223372036854775808 ~ 9223372036854775807



# 长整型数值

在为long型常量或变量赋值时，需要在所赋值的后面加上一个字母“L”（或“l”），说明所赋的值为long型。如果所赋的值未超出int型的取值范围，也可以省略字母“L”（或“l”）。

例如下面的代码均是正确的。

```
long la = 9876543234L; // 超出了int取值范围，必须加“L”  
long lb = 98765432L; // 未超出int取值范围，也可以加“L”  
long lc = 98765432; // 未超出int取值范围，可以省略“L”
```



# 浮点数

声明为浮点数据型的常量或变量用来存储小数，浮点数包括单精度型 (float) 和双精度 (double) 两个基本数据类型，这两个数据类型的区别是它们在内存中所占用的字节数不同，因此，它们所能够存储的整数的取值范围也不同

数据类型	关键字	占用内存字节数	取值范围
单精度型	float	4字节	$1.4E-45 \sim 3.4028235E38$
双精度型	double	8字节	$4.9E-324$ ~ $1.7976931348623157E308$



# float型数值

在为float型常量或变量赋值时，需要在所赋值的后面加上一个字母“F”（或“f”），说明所赋的值为float型。如果所赋的值为整数，并且未超出int型的取值范围，也可以省略字母“F”（或“f”）。

例如下面的代码均是正确的。

```
float fa = 9412.75F; // 赋值为小数，必须 “F”
```

```
float fb = 9876543210F; // 赋值超出int取值范围，必须 “F”
```

```
float fc = 9412F; // 未超出int取值范围，可以 “F”
```

```
float fd = 9412; // 也可以省略 “F”
```



# double型数值

在为double型常量或变量赋值时，需要在所赋值的后面加上一个字母“D”（或“d”），说明所赋的值为double型。如果所赋的值为小数，或者所赋的值为整数，并且未超出int型的取值范围，也可以省略字母“D”（或“d”）。

例如下面的代码均是正确的。

```
double da = 9412.75D; // 所赋值为小数，可以加上 “D”  
double db = 9412.75; // 所赋值为小数，也可以省略 “D”  
double dc = 9412D; // 未超出int取值范围，可以加上 “D”  
double dd = 9412; // 未超出int取值范围，可以省略 “D”  
double de = 9876543210D; // 超出int取值范围，必须加上 “D”
```

信息科学与工程学院





# 字符型

声明为字符型的常量或变量用来存储单个字符，它占用内存的2个字节来存储，字符型利用关键字“**char**”进行声明。

Java中的字符通过**Unicode**字符编码，以二进制的形式存储到计算机中，计算机可通过数据类型判断要输出的是一个字符还是一个整数。

Unicode编码采用无符号编码，一共可存储**65536**个字符，所以Java中的字符几乎可以处理所有国家的语言文字。



# 字符型数值

在为char型常量或变量赋值时，无论值是一个英文字母，或者是一个符号，还是一个汉字，必须将所赋的值放在英文状态下的一对单引号中。

例如下面的代码分别将字母“M”、符号“\*”和汉字“男”赋值给char型变量ca、cb和cc。

```
char ca = 'M'; // 将大写字母“M”赋值给char型变量  
char cb = '*'; // 将符号“*”赋值给char型变量  
char cc = '男'; // 将汉字“男”赋值给char型变量
```





# 逻辑型

声明为逻辑型的常量或变量用来存储逻辑值，逻辑值只有true和false，分别用来代表逻辑判断中的“真”和“假”，逻辑型利用关键字“boolean”进行声明。

例如下面的代码分别将true和false赋值给变量ba和bb。

```
boolean ba = true; // 将true赋值给变量ba  
boolean bb = false; // 将false赋值给变量bb
```







# 逻辑型

也可以将逻辑表达式赋值给boolean型变量，  
例如下面的代码分别将逻辑表达式“ $b < 8$ ”和逻辑表达式“ $b > 8$ ”赋值给boolean型变量ba和bb。

```
boolean ba = b < 8; // 将表达式 “ $b < 8$ ” 赋值给变量ba  
boolean bb = b > 8; // 将表达式 “ $b > 8$ ” 赋值给变量bb
```



## 2.3.2 引用数据类型

引用数据类型包括类引用、接口引用以及数组引用。

下面的代码分别声明一个 `java.lang.Object` 类的引用、`java.util.List` 接口的引用和一个 `int` 型数组的引用。

```
Object object = null; // 声明一个Object类的引用变量  
List list = null; // 声明一个List接口的引用变量  
int[] months = null; // 声明一个int型数组的引用变量
```

**说明：**将引用数据类型的常量或变量初始化为 `null` 时，表示引用数据类型的常量或变量不引用任何对象。

信息科学与工程学院



## 2.3.3 基本类型与引用类型的区别

---

基本数据类型与引用数据类型主要区别在以下两个方面：

- 基本数据类型与引用数据类型的组成
- Java虚拟机处理基本数据类型变量与引用数据类型变量的方式。

# 组成

基本数据类型是一个单纯的数据类型，它表示的是一个具体的数字、字符或逻辑值，例如68、  
'M' 或 true

对于引用数据类型，若一个变量引用的是一个复杂的数据结构的实例，则该变量的类型就属于引用数据类型

在引用数据类型变量所引用的实例中，不仅可以包含基本数据类型的变量，还可以包含对这些变量的具体操作行为，甚至是包含其他引用类型的变量。



# Java虚拟机的处理方式

对于基本数据类型的变量，Java虚拟机会根据变量的实际类型为其分配内存空间。

例如为int型变量分配4个字节的内存空间。

而引用类型的变量，Java虚拟机在内存空间中存放的并不是变量所引用的对象，而是对象在堆内存中存放的地址，所以引用变量最终只是指向被引用的对象，而不是存储引用对象的数据，因此两个引用变量之间的赋值，就是将一个引用变量存储的地址复制给另一个引用变量，从而使两个变量指向同一个对象。

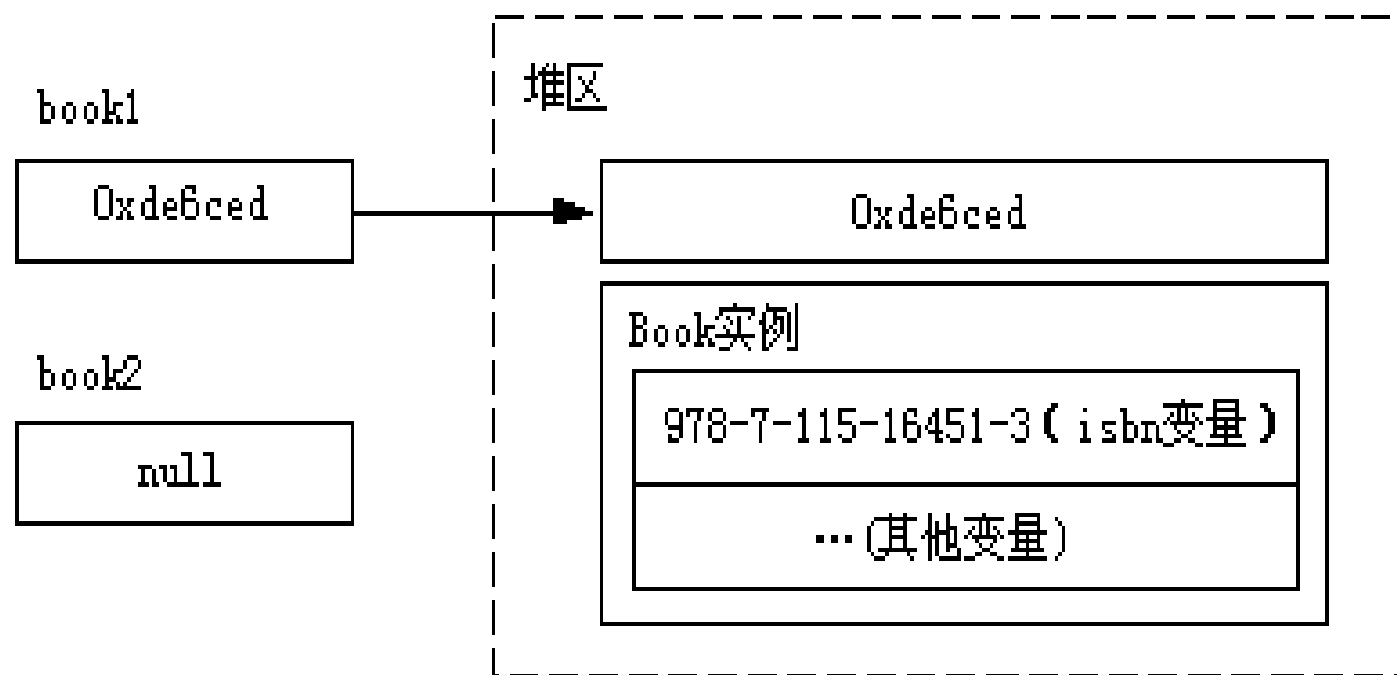
例如创建一个图书类Book:

```
public class Book {  
    String isbn = "978-7-115-16451-3";  
    String name = "×××应用开发完全手册";  
    String author = "××科技";  
    float price = 59.00F;  
}
```

声明两个Book类的实例，分别通过变量book1和book2进行引用，对book1进行具体的初始化，而将book2初始化为null，具体代码如下。

```
Book book1 = new Book();  
Book book2 = null;
```

Java虚拟机为引用变量book1、book2及book1所引用对象的成员变量分配的内存空间如下图所示。

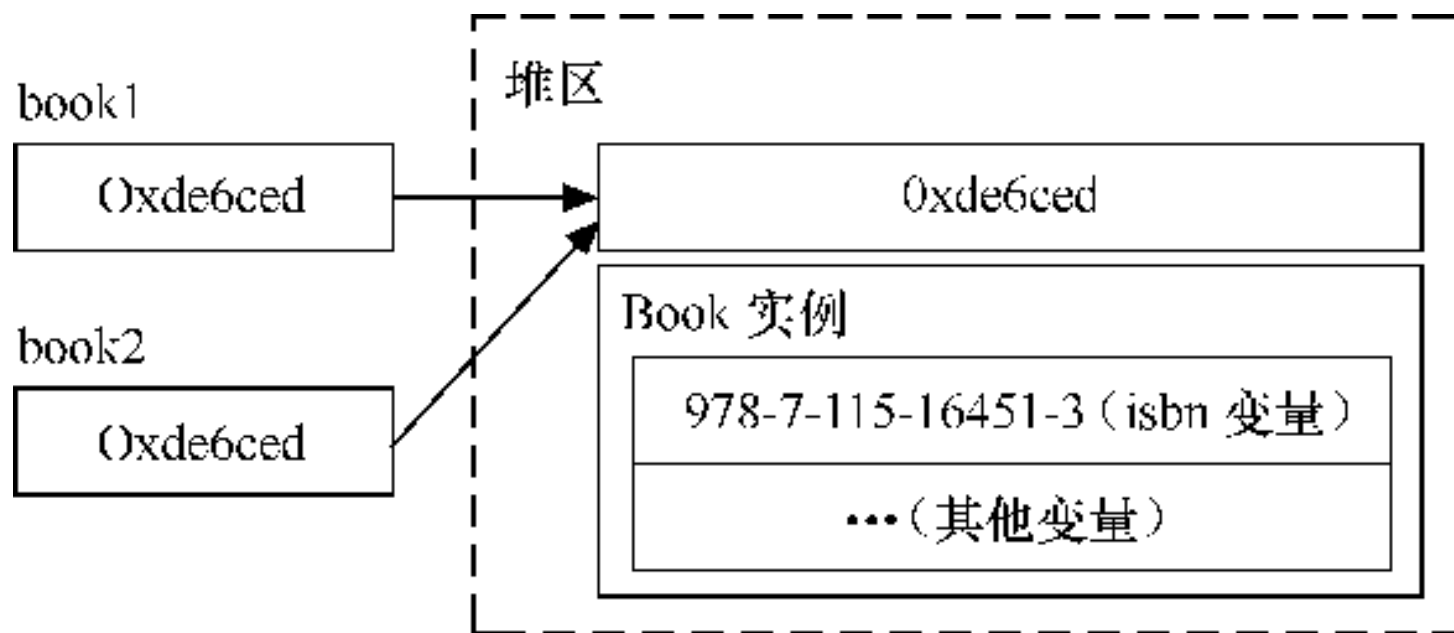


从图中可以看出，变量`book1`引用了`Book`类的实例，`book2`没有引用任何实例。

下面对变量book2进行具体的初始化，将book1引用实例的地址复制给book2变量，即book2与book1引用同一个Book类的实例，具体代码如下：

```
book2 = book1;
```

此时Java虚拟机的内存空间分配情况如下图所示。





## 2.3.4 数据类型之间的相互转换

所谓数据类型之间的相互转换，就是将变量从当前的数据类型转换为其他数据类型。

在Java中数据类型之间的相互转换可以分为以下3种情况：

- 基本数据类型之间的相互转换；
- 字符串与其他数据类型之间的相互转换；
- 引用数据类型之间的相互转换。

**说明：**这里只介绍基本数据类型之间的相互转换，其他两种情况将在相关的章节中介绍。

## 2.3.4 数据类型之间的相互转换

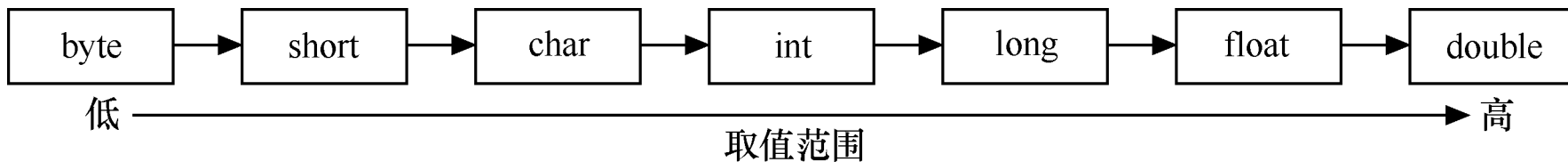
在对多个基本数据类型的数据进行混合运算时，如果这几个数据并不属于同一基本数据类型，需要先将它们转换为统一的数据类型，然后才能进行运算。

基本数据类型之间的相互转换又分为两种情况：

- 自动类型转换
- 强制类型转换。

# 1. 自动类型转换

当需要从低级类型向高级类型转换时，编程人员无需进行任何操作，Java会**自动完成**类型转换。低级类型是指取值范围相对较小的数据类型，高级类型则指取值范围相对较大的数据类型，例如long型相对于float型是低级数据类型，但是相对于int型则是高级数据类型。在基本数据类型中，除了boolean类型外均可参与算术运算，这些数据类型从低到高的排序如下图所示。





# 自动类型转换

在不同数据类型间的算术运算中，自动类型转换可以分为两种情况进行考虑：

- 第一种情况

含有int、long、float或double型的数据

- 第二种情况

含有byte、short或char型的数据。

# 自动类型转换

## 第一种情况

如果在算术表达式中含有int、long、float或double型的数据，Java首先会将所有数据类型较低的变量自动转换为表达式中最高的数据类型，然后再进行计算，并且计算结果的数据类型是表达式中级别最高的数据类型。

例如下面这段代码：

```
byte b = 75;  
char c = 'c';  
int i = 794215;  
long l = 9876543210L;  
long result = b * c - i + l;
```

Java首先会自动将表达式“ $b * c - i + l$ ”中的变量 $b$ 、 $c$ 和 $i$ 的数据类型转换为 $long$ 型。然后再进行计算，并且计算结果的数据类型为 $long$ 型。

所以将表达式“ $b * c - i + l$ ”直接赋值给数据类型相对小于 $long$ 型（例如 $int$ 型）的变量是不允许的，但是可以直接赋值给数据类型相对大于 $long$ 型（例如 $float$ 型）的变量。

再看下面这段代码：

```
byte b = 75;  
char c = 'c';  
int i = 794215;  
double d = 11.17;  
double result = b * c - i + d;
```

Java首先会自动将表达式“ $b * c - i + d$ ”中的变量 $b$ 、 $c$ 和 $i$ 的数据类型转换为 $double$ 型，然后再进行计算，并且计算结果的数据类型为 $double$ 型。

所以将表达式“ $b * c - i + d$ ”直接赋值给数据类型相对小于 $double$ 型（例如 $long$ 型）的变量是不允许的。

# 自动类型转换

## 第二种情况

---

如果在算术表达式中只含有byte、short或char型的数据，Java首先会将所有变量的类型自动转换为int型，然后再进行计算，并且计算结果的数据类型是int型。



例如下面这段代码：

```
byte b = 75;  
short s = 9412;  
char c = 'c';  
int result = b + s * c;
```

Java首先会自动将表达式“ $b + s * c$ ”中的变量 $b$ 、 $s$ 和 $c$ 的数据类型转换为 $int$ 型，然后再进行计算，并且计算结果的数据类型为 $int$ 型。

所以将表达式“ $b + s * c$ ”直接赋值给数据类型小于 $int$ 型（例如 $char$ 型）的变量是不允许的，但是可以直接赋值给数据类型相对大于 $int$ 型（例如 $long$ 型）的变量。

再看下面这段代码：

```
short s1 = 75;  
short s2 = 9412;  
int result = s1 * s2;
```

即使是在这段代码中，Java也会自动将表达式“ $s1 * s2$ ”中的变量 $s1$ 和 $s2$ 的数据类型转换为 $int$ 型，然后再进行计算，并且计算结果的数据类型也为 $int$ 型。

对于数据类型为 $byte$ 、 $short$ 、 $int$ 、 $long$ 、 $float$ 和 $double$ 的变量，可以将数据类型较小的数据或变量，直接赋值给数据类型较大的变量，但是相反的条件则不成立。



## 2. 强制类型转换

如果需要把数据类型较高的数据或变量赋值给数据类型相对较低的变量，就必须进行强制类型转换。

例如将Java默认为double型的数据“7.5”，赋值给数据类型为int型变量的方式如下：

```
int i = (int) 7.5;
```

这句代码在数据“7.5”的前方添加了代码“(int)”，意思就是将数据“7.5”的类型强制转换为int型。

在执行强制类型转换时，可能会导致数据溢出或精度降低。例如上面语句中变量i的值最终为7，导致数据精度降低。

信息科学与工程学院



## 2. 强制类型转换

如果将Java默认为int型的数据“774”赋值给数据类型为byte型变量，方法如下：

```
byte b = (byte) 774;
```

最终变量b的值为6，原因是整数774超出了byte型的取值范围，在进行强制类型转换时，整数774的二进制数据的前24位将被舍弃，变量b的数值是后8位的二进制数据，如下图所示。

十进制数 774 的二进制数据流的表现形式

00000000 00000000 00000011 00000110

被舍弃的二进制数据流的前 24 位 截取二进制数据流的后 8 位(表示十进制数 6)赋值给变量 b



## 2.4 数 组

数组是一种最为常见的数据结构，通过数组可以保存一组相同数据类型的数据，数组一旦创建，它的长度就固定了。

数组的类型可以为基本数据类型，也可以为引用数据类型，可以是一维数据，二维数据，甚至是多维数据。

## 2.4.1 数组的声明

声明一维数组的方式如下：

```
数组类型[] 数组标识符;
```

```
数组类型 数组标识符[];
```

这两种声明数组格式的作用是相同的。

Java语言中的二维数组是一种特殊的一维数组，即数组的每个元素又是一个一维数组，Java语言并不直接支持二维数组。声明二维数组的方式如下：

```
数组类型[][] 数组标识符;
```

或

```
数组类型 数组标识符[][];
```

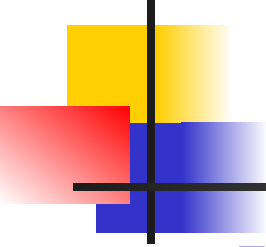


## 2.4.2 创建数组

创建数组实质上就是在内存中为数组分配相应的存储空间，有两种方式可以创建一维数组，一种是通过new关键字创建，另一种是通过“{ }”创建，例如：

```
int[] months = new int[12]; //months的长度为12
```

```
boolean[] members = { false, true, true, false }; //members的长度为4
```



## ■ 二维数组的创建方式 如下：

```
int[][] days = new int[2][3];
```

```
boolean holidays[][] = { { true, false, true }, { false, true, false } };
```

二维数组可以看做一个表格。数组days看成一个2行3列的表格，数组holidays看成2行3列的表格

	列索引0	列索引1	列索引2
行索引0	days[0][0]	days[0][1]	days[0][2]
行索引1	days[1][0]	days[1][1]	days[1][2]





## 2.4.3 初始化数组

- 在声明数组的同时也可以给数组元素一个初始值，一维数组初始化如下：

```
int boy [] = {2,45,36,7,69};
```

或

```
int boy [] = new int [5]
```

- 二维数组初始化如下：

```
boolean holidays[][] = { { true, false, true }, { false, true, false } };
```

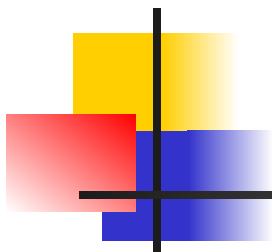


## 2.4.4 数组长度

如果需要获得一维数组的长度，可以通过下面的方式：

```
System.out.println(months.length) ;// 输出值为12
```

```
System.out.println(members.length) ;// 输出值为4
```



如果是通过下面的方式获得二维数组的长度，得到的是二维数组的行数：

```
System.out.println(days.length); // 输出值为2
```

```
System.out.println(holidays.length); // 输出值为2
```

如果需要获得二维数组的列数，可以通过下面的方式：

```
System.out.println(days[0].length); // 输出值为3
```

```
System.out.println(holidays[0].length); // 输出值为3
```

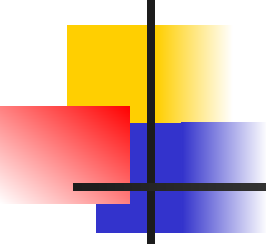




## 2.4.5 使用数组元素

- 一维数组在访问数组中的元素时，需要同时指定数组标识符和元素在数组中的索引，例如访问上面代码中创建的数组，具体代码如下：

```
System.out.println(months[2]);  
System.out.println(members[2]);
```



二维数组也是通过索引符访问自己的元素，在访问数组中的元素时，需要同时指定数组标识符和元素在数组中的索引，例如访问2.4.2节代码中创建的二维数组，输出位于第2行、第3列的元素，具体代码如下：

```
System.out.println(days[1][2]);  
System.out.println(holidays[1][2]);
```

