Broadview
www.broadview.com.cn

几十所院校、培训机构共同的选择

疯狂Java讲义 精粹

（第2版）

李刚 编著

疯狂源自梦想
技术成就辉煌

疯狂源自梦想
技术成就辉煌

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

内 容 简 介

本书是《疯狂 Java 讲义精粹》的第 2 版，本书相比《疯狂 Java 讲义》更浅显易懂，讲解更细致，本书同样介绍了 Java 8 的新特性，**本书大部分示例程序都采用 Lambda 表达式、流式 API 进行了改写，因此务必使用 Java 8 的 JDK 来编译、运行。**

本书尽量浅显、直白地介绍 Java 编程的相关方面，全书内容覆盖了 Java 的基本语法结构、Java 的面向对象特征、Java 集合框架体系、Java 泛型、异常处理、Java 注释、Java 的 IO 流体系、Java 多线程编程、Java 网络通信编程。覆盖了 java.lang、java.util、java.text、java.io 和 java.nio 包下绝大部分类和接口。本书全面介绍了 Java 8 的新的接口语法、Lambda 表达式、方法引用、构造器引用、函数式编程、流式编程、新的日期、时间 API、并行支持、改进的类型推断、重复注解、JDBC 4.2 新特性等新特性。

本书为打算认真掌握 Java 编程的读者而编写，适合各种层次的 Java 学习者和工作者阅读。本书专门针对高校课程进行过调整，尤其适合作为高校教育、培训机构的 Java 教材。

图书在版编目（CIP）数据

疯狂 Java 讲义精粹 / 李刚编著. — 2 版. —北京：电子工业出版社，2014.10
ISBN 978-7-121-24346-2

Ⅰ. ①疯… Ⅱ. ①李… Ⅲ. ①JAVA 语言－程序设计 Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字（2014）第 213893 号

策划编辑：张月萍
责任编辑：葛　娜
印　　刷：北京京科印刷有限公司
装　　订：三河市鹏成印业有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编 100036
开　　本：850×1168　1/16　　印张：28　　字数：1045 千字　　彩插：1
版　　次：2012 年 1 月第 1 版
　　2014 年 10 月第 2 版
印　　次：2014 年 10 月第 1 次印刷
印　　数：4000 册　　定价：59.90 元（含光盘 1 张）

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

CHAPTER

3

第 3 章 流程控制与数组

3.3 循环结构

循环语句可以在满足循环条件的情况下，反复执行某一段代码，这段被重复执行的代码被称为循环体。当反复执行这个循环体时，需要在合适的时候把循环条件改为假，从而结束循环，否则循环将一直执行下去，形成死循环。循环语句可能包含如下 4 个部分。

- 初始化语句（init_statement）：一条或多条语句，这些语句用于完成一些初始化工作，初始化语句在循环开始之前执行。
- 循环条件（test_expression）：这是一个 boolean 表达式，这个表达式能决定是否执行循环体。
- 循环体（body_statement）：这个部分是循环的主体，如果循环条件允许，这个代码块将被重复执行。如果这个代码块只有一行语句，则这个代码块的花括号是可以省略的。
- 迭代语句（iteration_statement）：这个部分在一次循环体执行结束后，对循环条件求值之前执行，通常用于控制循环条件中的变量，使得循环在合适的时候结束。

上面 4 个部分只是一般性的分类，并不是每个循环中都非常清晰地分出了这 4 个部分。

3.3.1 while 循环语句

while 循环的语法格式如下：

```
[init_statement]
while(test_expression)
{
    statement;
    [iteration_statement]
}
```

while 循环每次执行循环体之前，先对 test_ expression 循环条件求值，如果循环条件为 true，则运行循环体部分。从上面的语法格式来看，迭代语句 iteration_statement 总是位于循环体的最后，因此只有当循环体能成功执行完成时，while 循环才会执行 iteration_statement 迭代语句。

从这个意义上来看，while 循环也可被当成条件语句——如果 test_expression 条件一开始就为 false，则循环体部分将永远不会获得执行。

下面程序示范了一个简单的 while 循环。

程序清单：codes\03\3.3\WhileTest.java

```
public class WhileTest
{
```

```
        public static void main(String[] args)
        {
            // 循环的初始化条件
            int count = 0;
            // 当 count 小于 10 时，执行循环体
            while (count < 10)
            {
                System.out.println(count);
                // 迭代语句
                count++;
            }
            System.out.println("循环结束!");
        }
    }
```

如果 while 循环的循环体部分和迭代语句合并在一起，且只有一行代码，则可以省略 while 循环后的花括号。但这种省略花括号的做法，可能降低程序的可读性。

注意：

如果省略了循环体的花括号，那么 while 循环条件仅控制到紧跟该循环条件的第一个分号处。

使用 while 循环时，一定要保证循环条件有变成 false 的时候，否则这个循环将成为一个死循环，永远无法结束这个循环。例如如下代码（程序清单同上）：

```
    // 下面是一个死循环
    int count = 0;
    while (count < 10)
    {
        System.out.println("不停执行的死循环 " + count);
        count--;
    }
    System.out.println("永远无法跳出的循环体");
```

在上面代码中，count 的值越来越小，这将导致 count 值永远小于 10，count < 10 循环条件一直为 true，从而导致这个循环永远无法结束。

除此之外，对于许多初学者而言，使用 while 循环时还有一个陷阱：while 循环的循环条件后紧跟一个分号。比如有如下程序片段（程序清单同上）：

```
    int count = 0;
    // while 后紧跟一个分号，表明循环体是一个分号（空语句）
    while (count < 10);
    // 下面的代码块与 while 循环已经没有任何关系
    {
        System.out.println("------" + count);
        count++;
    }
```

乍一看，这段代码片段没有任何问题，但仔细看一下这个程序，不难发现 while 循环的循环条件表达式后紧跟了一个分号。在 Java 程序中，一个单独的分号表示一个空语句，不做任何事情的空语句，这意味着这个 while 循环的循环体是空语句。空语句作为循环体也不是最大的问题，问题是当 Java 反复执行这个循环体时，循环条件的返回值没有任何改变，这就成了一个死循环。分号后面的代码块则与 while 循环没有任何关系。

3.3.2 do while 循环语句

do while 循环与 while 循环的区别在于：while 循环是先判断循环条件，如果条件为真则执行循环体；而 do while 循环则先执行循环体，然后才判断循环条件，如果循环条件为真，则执行下一次循环，否则中止循环。do while 循环的语法格式如下：

```
    [init_statement]
    do
    {
        statement;
         [iteration_statement]
```

```
}while (test_expression);
```

与 while 循环不同的是，do while 循环的循环条件后必须有一个分号，这个分号表明循环结束。

下面程序示范了 do while 循环的用法。

程序清单：codes\03\3.3\DoWhileTest.java

```
public class DoWhileTest
{
    public static void main(String[] args)
    {
        // 定义变量 count
        int count = 1;
        // 执行 do while 循环
        do
        {
            System.out.println(count);
            // 循环迭代语句
            count++;
            // 循环条件紧跟 while 关键字
        }while (count < 10);
        System.out.println("循环结束!");
    }
}
```

即使 test_expression 循环条件的值开始就是假，do while 循环也会执行循环体。因此，do while 循环的循环体至少执行一次。下面的代码片段验证了这个结论（程序清单同上）。

```
// 定义变量 count2
int count2 = 20;
// 执行 do while 循环
do
    // 这行代码把循环体和迭代部分合并成了一行代码
    System.out.println(count2++);
while (count2 < 10);
System.out.println("循环结束!");
```

从上面程序来看，虽然开始 count2 的值就是 20，count2 < 10 表达式返回 false，但 do while 循环还是会把循环体执行一次。

3.3.3 for 循环

for 循环是更加简洁的循环语句，大部分情况下，for 循环可以代替 while 循环、do while 循环。for 循环的基本语法格式如下：

```
for ([init_statement]; [test_expression]; [iteration_statement])
{
    statement
}
```

程序执行 for 循环时，先执行循环的初始化语句 init_statement，初始化语句只在循环开始前执行一次。每次执行循环体之前，先计算 test_expression 循环条件的值，如果循环条件返回 true，则执行循环体，循环体执行结束后执行循环迭代语句。因此，对于 for 循环而言，循环条件总比循环体要多执行一次，因为最后一次执行循环条件返回 false，将不再执行循环体。

值得指出的是，for 循环的循环迭代语句并没有与循环体放在一起，因此即使在执行循环体时遇到 continue 语句结束本次循环，循环迭代语句也一样会得到执行。

注意：

for 循环和 while、do while 循环不一样：由于 while、do while 循环的循环迭代语句紧跟着循环体，因此如果循环体不能完全执行，如使用 continue 语句来结束本次循环，则循环迭代语句不会被执行。但 for 循环的循环迭代语句并没有与循环体放在一起，因此不管是否使用 continue 语句来结束本次循环，循环迭代语句一样会获得执行。

与前面循环类似的是，如果循环体只有一行语句，那么循环体的花括号可以省略。下面使用 for 循环代替前面的 while 循环，代码如下。

程序清单：codes\03\3.3\ForTest.java

```
public class ForTest
{
    public static void main(String[] args)
    {
        // 循环的初始化条件、循环条件、循环迭代语句都在下面一行
        for (int count = 0 ; count < 10 ; count++)
        {
            System.out.println(count);
        }
        System.out.println("循环结束!");
    }
}
```

在上面的循环语句中，for 循环的初始化语句只有一个，循环条件也只是一个简单的 boolean 表达式。实际上，for 循环允许同时指定多个初始化语句，循环条件也可以是一个包含逻辑运算符的表达式。例如如下程序：

程序清单：codes\03\3.3\ForTest2.java

```
public class ForTest2
{
    public static void main(String[] args)
    {
        // 同时定义了三个初始化变量，使用&&来组合多个 boolean 表达式
        for (int b = 0, s = 0 , p = 0
            ; b < 10 && s < 4 && p < 10; p++)
        {
            System.out.println(b++);
            System.out.println(++s + p);
        }
    }
}
```

上面代码中初始化变量有三个，但是只能有一个声明语句，因此如果需要在初始化表达式中声明多个变量，那么这些变量应该具有相同的数据类型。

初学者使用 for 循环时也容易犯一个错误，他们以为只要在 for 后的括号内控制了循环迭代语句就万无一失，但实际情况则不是这样的。例如下面的程序：

程序清单：codes\03\3.3\ForErrorTest.java

```
public class ForErrorTest
{
    public static void main(String[] args)
    {
        // 循环的初始化条件、循环条件、循环迭代语句都在下面一行
        for (int count = 0 ; count < 10 ; count++)
        {
            System.out.println(count);
            // 再次修改了循环变量
            count *= 0.1;
        }
        System.out.println("循环结束!");
    }
}
```

在上面的 for 循环中，表面上看起来控制了 count 变量的自加，count < 10 有变成 false 的时候。但实际上程序中粗体字标识的代码行在循环体内修改了 count 变量的值，并且把这个变量的值乘以了 0.1，这也会导致 count 的值永远都不能超过 10，因此上面程序也是一个死循环。

注意：

建议不要在循环体内修改循环变量（也叫循环计数器）的值，否则会增加程序出错的可能性。万一程序真的需要访问、修改循环变量的值，建议重新定义一个临时变量，先将循环变量的值赋给临时变量，然后对临时变量的值进行修改。

for 循环圆括号中只有两个分号是必需的，初始化语句、循环条件、迭代语句部分都是可以省略的，如果省略了循环条件，则这个循环条件默认为 true，将会产生一个死循环。例如下面程序。

程序清单：codes\03\3.3\DeadForTest.java

```
public class DeadForTest
{
    public static void main(String[] args)
    {
        // 省略了 for 循环三个部分，循环条件将一直为 true
        for (; ; )
        {
            System.out.println("==============");
        }
    }
}
```

运行上面程序，将看到程序一直输出==============字符串，这表明此程序是一个死循环。

使用 for 循环时，还可以把初始化条件定义在循环体之外，把循环迭代语句放在循环体内，这种做法就非常类似于前面的 while 循环了。下面的程序再次使用 for 循环来代替前面的 while 循环。

程序清单：codes\03\3.3\ForInsteadWhile.java

```
public class ForInsteadWhile
{
    public static void main(String[] args)
    {
        // 把 for 循环的初始化条件提出来独立定义
        int count = 0;
        // for 循环里只放循环条件
        for( ; count < 10 ; )
        {
            System.out.println(count);
            // 把循环迭代部分放在循环体之后定义
            count++;
        }
        System.out.println("循环结束!");
        // 此处将还可以访问 count 变量
    }
}
```

上面程序的执行过程和前面的 WhileTest.java 程序的执行过程完全相同。因为把 for 循环的循环迭代部分放在循环体之后，则会出现与 while 循环类似的情形，如果循环体部分使用 continue 语句来结束本次循环，将会导致循环迭代语句得不到执行。

把 for 循环的初始化语句放在循环之前定义还有一个作用：可以扩大初始化语句中所定义变量的作用域。在 for 循环里定义的变量，其作用域仅在该循环内有效，for 循环终止以后，这些变量将不可被访问。如果需要在 for 循环以外的地方使用这些变量的值，就可以采用上面的做法。除此之外，还有一种做法也可以满足这种要求：额外定义一个变量来保存这个循环变量的值。例如下面代码片段：

```
int tmp = 0;
// 循环的初始化条件、循环条件、循环迭代语句都在下面一行
for (int i = 0 ; i < 10 ; i++)
{
    System.out.println(count);
    // 使用 tmp 来保存循环变量 i 的值
    tmp = i;
}
System.out.println("循环结束!");
// 此处还可通过 tmp 变量来访问 i 变量的值
```

相比前面的代码，通常更愿意选择这种解决方案。使用一个变量 tmp 来保存循环变量 i 的值，使得程序更加清晰，变量 i 和变量 tmp 的责任更加清晰。反之，如果采用前一种方法，则变量 i 的作用域被扩大了，功能也被扩大了。作用域扩大的后果是：如果该方法还有另一个循环也需要定义循环变量，则不能再次使用 i 作为循环变量。

提示：

选择循环变量时，习惯选择 i、j、k 来作为循环变量。

3.3.4 嵌套循环

如果把一个循环放在另一个循环体内，那么就可以形成嵌套循环，嵌套循环既可以是 for 循环嵌套 while 循环，也可以是 while 循环嵌套 do while 循环……即各种类型的循环都可以作为外层循环，也可以作为内层循环。

当程序遇到嵌套循环时，如果外层循环的循环条件允许，则开始执行外层循环的循环体，而内层循环将被外层循环的循环体来执行——只是内层循环需要反复执行自己的循环体而已。当内层循环执行结束，且外层循环的循环体执行结束时，则再次计算外层循环的循环条件，决定是否再次开始执行外层循环的循环体。

根据上面分析，假设外层循环的循环次数为 *n* 次，内层循环的循环次数为 *m* 次，那么内层循环的循环体实际上需要执行 *n* × *m* 次。嵌套循环的执行流程如图 3.1 所示。

从图 3.1 来看，嵌套循环就是把内层循环当成外层循环的循环体。当只有内层循环的循环条件为 false 时，才会完全跳出内层循环，才可以结束外层循环的当次循环，开始下一次循环。下面是一个嵌套循环的示例代码。

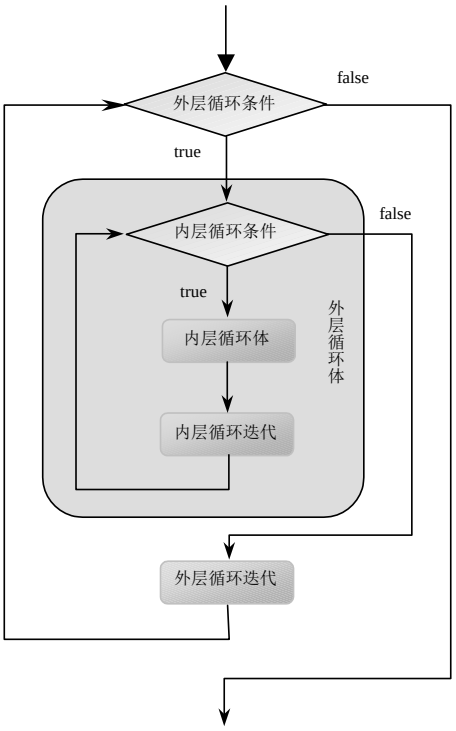


图 3.1 嵌套循环的执行流程

程序清单：codes\03\3.3\NestedLoopTest.java

```
public class NestedLoopTest
{
    public static void main(String[] args)
    {
        // 外层循环
        for (int i = 0 ; i < 5 ; i++ )
        {
            // 内层循环
            for (int j = 0; j < 3 ; j++ )
            {
                System.out.println("i 的值为:" + i + "  j 的值为:" + j);
            }
        }
    }
}
```

运行上面程序，看到如下运行结果：

```
i 的值为:0  j 的值为:0
i 的值为:0  j 的值为:1
i 的值为:0  j 的值为:2
……
```

从上面运行结果可以看出，进入嵌套循环时，循环变量 i 开始为 0，这时即进入了外层循环。进入外层循环后，内层循环把 i 当成一个普通变量，其值为 0。在外层循环的当次循环里，内层循环就是一个普通循环。

实际上，嵌套循环不仅可以是两层嵌套，而且可以是三层嵌套、四层嵌套……不论循环如何嵌套，总可以把内层循环当成外层循环的循环体来对待，区别只是这个循环体里包含了需要反复执行的代码。

CHAPTER

3

第 3 章 流程控制与数组

3.5 数组类型

数组是编程语言中最常见的一种数据结构，可用于存储多个数据，每个数组元素存放一个数据，通常可通过数组元素的索引来访问数组元素，包括为数组元素赋值和取出数组元素的值。Java 语言的数组则具有其特有的特征，下面将详细介绍 Java 语言的数组。

3.5.1 理解数组：数组也是一种类型

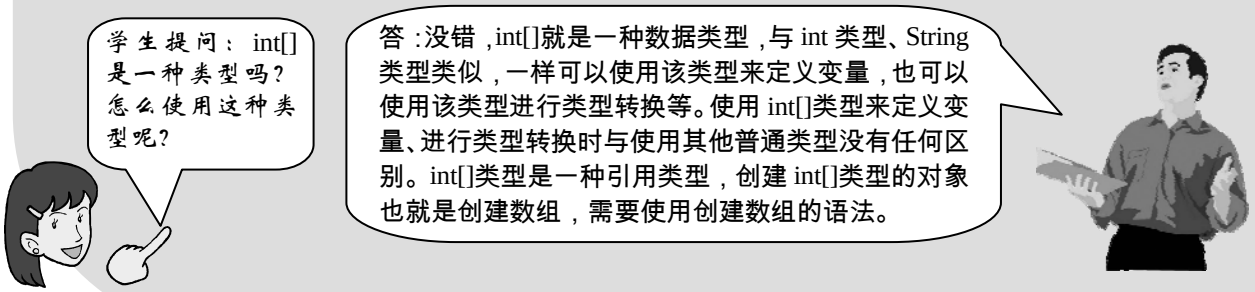
Java 的数组要求所有的数组元素具有相同的数据类型。因此，在一个数组中，数组元素的类型是唯一的，即一个数组里只能存储一种数据类型的数据，而不能存储多种数据类型的数据。

因为 Java 语言是面向对象的语言，而类与类之间可以支持继承关系，这样可能产生一个数组里可以存放多种数据类型的假象。例如有一个水果数组，要求每个数组元素都是水果，实际上数组元素既可以是苹果，也可以是香蕉（苹果、香蕉都继承了水果，都是一种特殊的水果），但这个数组的数组元素的类型还是唯一的，只能是水果类型。

一旦数组的初始化完成，数组在内存中所占的空间将被固定下来，因此数组的长度将不可改变。即使把某个数组元素的数据清空，但它所占的空间依然被保留，依然属于该数组，数组的长度依然不变。

Java 的数组既可以存储基本类型的数据，也可以存储引用类型的数据，只要所有的数组元素具有相同的类型即可。

值得指出的是，数组也是一种数据类型，它本身是一种引用类型。例如 int 是一个基本类型，但 int[]（这是定义数组的一种方式）就是一种引用类型了。



3.5.2 定义数组

Java 语言支持两种语法格式来定义数组：

```
type[] arrayName;
type arrayName[];
```

对这两种语法格式而言，通常推荐使用第一种格式。因为第一种格式不仅具有更好的语意，而且具有更好的可读性。对于 type[] arrayName;方式，很容易理解这是定义一个变量，其中变量名是 arrayName，而变量类型是 type[]。前面已经指出：type[]确实是一种新类型，与 type 类型完全不同（例如 int 类型是基本类型，但 int[]是引用类型）。因此，这种方式既容易理解，也符合定义变量的语法。但第二种格式 type arrayName[]的可读性就差了，看起来好像定义了一个类型为 type 的变量，而变量名是 arrayName[]，这与真实的含义相去甚远。

可能有些读者非常喜欢 type arrayName[];这种定义数组的方式，这可能是因为早期某些计算机读物的误导，从现在开始就不要再使用这种糟糕的方式了。

提示：

Java 的模仿者 C#就不再支持 type arrayName[]这种语法，它只支持第一种定义数组的语法。越来越多的语言不再支持 type arrayName[]这种数组定义语法。

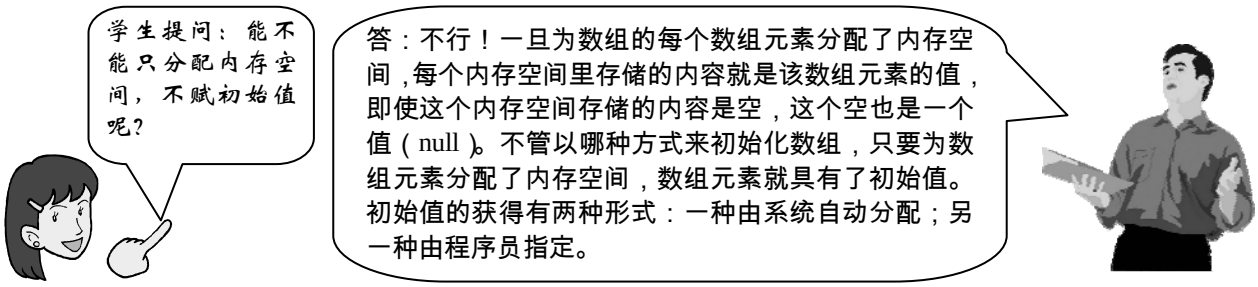
数组是一种引用类型的变量，因此使用它定义一个变量时，仅仅表示定义了一个引用变量（也就是定义了一个指针），这个引用变量还未指向任何有效的内存，因此定义数组时不能指定数组的长度。而且由于定义数组只是定义了一个引用变量，并未指向任何有效的内存空间，所以还没有内存空间来存储数组元素，因此这个数组也不能使用，只有对数组进行初始化后才可以使用。

注意：

定义数组时不能指定数组的长度。

3.5.3 数组的初始化

Java 语言中数组必须先初始化，然后才可以使用。所谓初始化，就是为数组的数组元素分配内存空间，并为每个数组元素赋初始值。



数组的初始化有如下两种方式。

- 静态初始化：初始化时由程序员显式指定每个数组元素的初始值，由系统决定数组长度。
- 动态初始化：初始化时程序员只指定数组长度，由系统为数组元素分配初始值。

1. 静态初始化

静态初始化的语法格式如下：

```
arrayName = new type[]{element1, element2 , element3 , element4 ...}
```

在上面的语法格式中，前面的 type 就是数组元素的数据类型，此处的 type 必须与定义数组变量时所使用的 type 相同，也可以是定义数组时所指定的 type 的子类，并使用花括号把所有的数组元素括起来，多个数组元素之间以英文逗号（,）隔开，定义初始化值的花括号紧跟[]之后。值得指出的是，执行静态初始化时，显式指定的数组元素值的类型必须与 new 关键字后的 type 类型相同，或者是其子类的实例。下面代码定义了使用这三种形式来进行静态初始化。

程序清单：codes\03\3.5\ArrayTest.java

```
// 定义一个 int 数组类型的变量，变量名为 intArr
int[] intArr;
// 使用静态初始化，初始化数组时只指定数组元素的初始值，不指定数组长度
intArr = new int[]{5, 6, 8, 20};
//定义一个 Object 数组类型的变量，变量名为 objArr
Object[] objArr;
// 使用静态初始化，初始化数组时数组元素的类型是
// 定义数组时所指定的数组元素类型的子类
objArr = new String[] {"Java" , "李刚"};
Object[] objArr2;
// 使用静态初始化
objArr2 = new Object[] {"Java" , "李刚"};
```

因为 Java 语言是面向对象的编程语言，能很好地支持子类和父类的继承关系：子类实例是一种特殊的父类实例。在上面程序中，String 类型是 Object 类型的子类，即字符串是一种特殊的 Object 实例。关于继承更详细的介绍，请参考本书第 4 章。

除此之外，静态初始化还有如下简化的语法格式：

```
type[] arrayName = {element1, element2, element3, element4 ...}
```

在这种语法格式中，直接使用花括号来定义一个数组，花括号把所有的数组元素括起来形成一个数组。只有在定义数组的同时执行数组初始化才支持使用简化的静态初始化。

在实际开发过程中，可能更习惯将数组定义和数组初始化同时完成，代码如下（程序清单同上）：

```
// 数组的定义和初始化同时完成，使用简化的静态初始化写法
int[] a = {5, 6, 7, 9};
```

2．动态初始化

动态初始化只指定数组的长度，由系统为每个数组元素指定初始值。动态初始化的语法格式如下：

```
arrayName = new type[length];
```

在上面语法中，需要指定一个 int 类型的 length 参数，这个参数指定了数组的长度，也就是可以容纳数组元素的个数。与静态初始化相似的是，此处的 type 必须与定义数组时使用的 type 类型相同，或者是定义数组时使用的 type 类型的子类。下面代码示范了如何进行动态初始化（程序清单同上）。

```
// 数组的定义和初始化同时完成，使用动态初始化语法
int[] prices = new int[5];
// 数组的定义和初始化同时完成，初始化数组时元素的类型是定义数组时元素类型的子类
Object[] books = new String[4];
```

执行动态初始化时，程序员只需指定数组的长度，即为每个数组元素指定所需的内存空间，系统将负责为这些数组元素分配初始值。指定初始值时，系统按如下规则分配初始值。

- 数组元素的类型是基本类型中的整数类型（byte、short、int 和 long），则数组元素的值是 0。
- 数组元素的类型是基本类型中的浮点类型（float、double），则数组元素的值是 0.0。
- 数组元素的类型是基本类型中的字符类型（char），则数组元素的值是'\u0000'。
- 数组元素的类型是基本类型中的布尔类型（boolean），则数组元素的值是 false。
- 数组元素的类型是引用类型（类、接口和数组），则数组元素的值是 null。

注意：

不要同时使用静态初始化和动态初始化，也就是说，不要在进行数组初始化时，既指定数组的长度，也为每个数组元素分配初始值。

数组初始化完成后，就可以使用数组了，包括为数组元素赋值、访问数组元素值和获得数组长度等。

3.5.4 使用数组

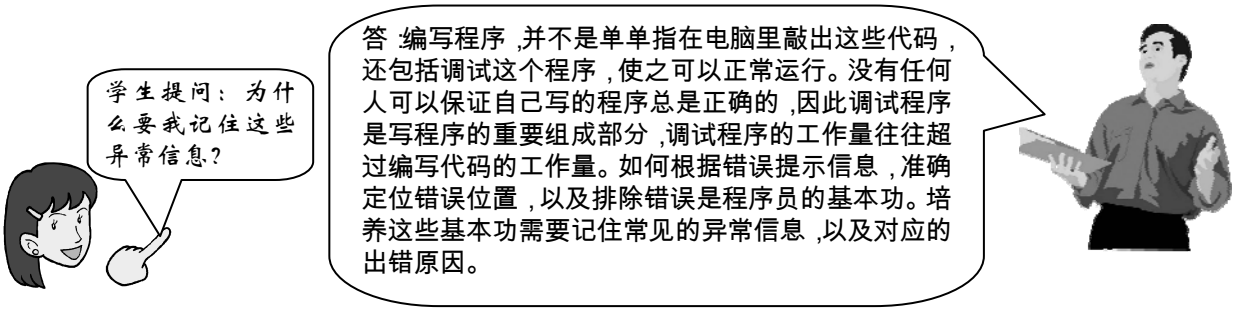
数组最常见的用法就是访问数组元素，包括对数组元素进行赋值和取出数组元素的值。访问数组元素都

是通过在数组引用变量后紧跟一个方括号（[]），方括号里是数组元素的索引值，这样就可以访问数组元素了。访问到数组元素后，就可以把一个数组元素当成一个普通变量使用了，包括为该变量赋值和取出该变量的值，这个变量的类型就是定义数组时使用的类型。

Java 语言的数组索引是从 0 开始的，也就是说，第一个数组元素的索引值为 0，最后一个数组元素的索引值为数组长度减 1。下面代码示范了输出数组元素的值，以及为指定数组元素赋值（程序清单同上）。

```
// 输出 objArr 数组的第二个元素，将输出字符串"李刚"
System.out.println(objArr[1]);
// 为 objArr2 的第一个数组元素赋值
objArr2[0] = "Spring";
```

如果访问数组元素时指定的索引值小于 0，或者大于等于数组的长度，编译程序不会出现任何错误，但运行时出现异常：java.lang.ArrayIndexOutOfBoundsException: N（数组索引越界异常），异常信息后的 N 就是程序员试图访问的数组索引。



下面代码试图访问的数组元素索引值等于数组长度，将引发数组索引越界异常（程序清单同上）。

```
// 访问数组元素指定的索引值等于数组长度，所以下面代码将在运行时出现异常
System.out.println(objArr2[2]) ;
```

所有的数组都提供了一个 length 属性，通过这个属性可以访问到数组的长度，一旦获得了数组的长度，就可以通过循环来遍历该数组的每个数组元素。下面代码示范了输出 prices 数组（动态初始化的 int[]数组）的每个数组元素的值（程序清单同上）。

```
// 使用循环输出 prices 数组的每个数组元素的值
for (int i = 0; i < prices.length ; i ++ )
{
    System.out.println(prices[i]);
}
```

执行上面代码将输出 5 个 0，因为 prices 数组执行的是默认初始化，数组元素是 int 类型，系统为 int 类型的数组元素赋值为 0。

下面代码示范了为动态初始化的数组元素进行赋值，并通过循环方式输出每个数组元素（程序清单同上）。

```
// 对动态初始化后的数组元素进行赋值
books[0] = "疯狂 Java 讲义";
books[1] = "轻量级 Java EE 企业应用实战";
// 使用循环输出 books 数组的每个数组元素的值
for (int i = 0 ; i < books.length ; i++ )
{
    System.out.println(books[i]);
}
```

上面代码将先输出字符串"疯狂 Java 讲义"和"轻量级 Java EE 企业应用实战"，然后输出两个 null，因为 books 使用了动态初始化，系统为所有数组元素都分配一个 null 作为初始值，后来程序又为前两个元素赋值，所以看到了这样的程序输出结果。

从上面代码中不难看出，初始化一个数组后，相当于同时初始化了多个相同类型的变量，通过数组元素的索引就可以自由访问这些变量（实际上都是数组元素）。使用数组元素与使用普通变量并没有什么不同，一样可以对数组元素进行赋值，或者取出数组元素的值。

3.5.5 foreach 循环

从 Java 5 之后，Java 提供了一种更简单的循环：foreach 循环，这种循环遍历数组和集合（关于集合的介绍请参考本书第 7 章）更加简洁。使用 foreach 循环遍历数组和集合元素时，无须获得数组和集合长度，无须根据索引来访问数组元素和集合元素，foreach 循环自动遍历数组和集合的每个元素。

foreach 循环的语法格式如下：

```
for(type variableName : array | collection)
{
    // variableName 自动迭代访问每个元素...
}
```

在上面语法格式中，type 是数组元素或集合元素的类型，variableName 是一个形参名，foreach 循环自动将数组元素、集合元素依次赋给该变量。下面程序示范了如何使用 foreach 循环来遍历数组元素。

程序清单：codes\03\3.5\ForEachTest.java

```
public class ForEachTest
{
    public static void main(String[] args)
    {
        String[] books = {"轻量级 Java EE 企业应用实战" ,
        "疯狂 Java 讲义",
        "疯狂 Android 讲义"};
        // 使用 foreach 循环来遍历数组元素
        // 其中 book 将会自动迭代每个数组元素
        for (String book : books)
        {
            System.out.println(book);
        }
    }
}
```

从上面程序可以看出，使用 foreach 循环遍历数组元素时无须获得数组长度，也无须根据索引来访问数组元素。foreach 循环和普通循环不同的是，它无须循环条件，无须循环迭代语句，这些部分都由系统来完成，foreach 循环自动迭代数组的每个元素，当每个元素都被迭代一次后，foreach 循环自动结束。

当使用 foreach 循环来迭代输出数组元素或集合元素时，通常不要对循环变量进行赋值，虽然这种赋值在语法上是允许的，但没有太大的实际意义，而且极易引起错误。例如下面程序。

程序清单：codes\03\3.5\ForEachErrorTest.java

```
public class ForEachErrorTest
{
    public static void main(String[] args)
    {
        String[] books = {"轻量级 Java EE 企业应用实战" ,
        "疯狂 Java 讲义",
        "疯狂 Android 讲义"};
        // 使用 foreach 循环来遍历数组元素，其中 book 将会自动迭代每个数组元素
        for (String book : books)
        {
            book = "疯狂 Ajax 讲义";
            System.out.println(book);
        }
        System.out.println(books[0]);
    }
}
```

运行上面程序，将看到如下运行结果：

```
疯狂 Ajax 讲义
疯狂 Ajax 讲义
疯狂 Ajax 讲义
轻量级 Java EE 企业应用实战
```

从上面运行结果来看，由于在 foreach 循环中对数组元素进行赋值，结果导致不能正确遍历数组元素，不能正确地取出每个数组元素的值。而且当再次访问第一个数组元素时，发现数组元素的值依然没有改变。不难看出，当使用 foreach 来迭代访问数组元素时，foreach 中的循环变量相当于一个临时变量，系统会把数

组元素依次赋给这个临时变量，而这个临时变量并不是数组元素，它只是保存了数组元素的值。因此，如果希望改变数组元素的值，则不能使用这种 foreach 循环。

注意：

使用 foreach 循环迭代数组元素时，并不能改变数组元素的值，因此不要对 foreach 的循环变量进行赋值。