
脚下留心：

```
String s1="hello";
String s2="hello";
```

上面的两个等号“=”说明 s1 与 s2 是同一个对象，而下面这两句代码是创建两个对象。

```
String s1=new String("hello");
String s2=new String("hello");
```

有了前面讲的知识，读者就不难理解：尽管它们内容相同，但却是两个不同的对象。

---

indexOf(int ch)方法是用来返回一个字符在该字符串中的首次出现的位置，如果没有这个字符则返回“-1”。它的另一种形式 indexOf(int ch,int fromIndex)返回的是从 fromIndex 指定的数值开始，ch 字符首次出现的位置。该方法可以应用于文件和数据库内容的查找等功能，如 Word 等字处理软件。如：

```
String str="hello world";
System.out.println(str.indexOf('o'));
```

打印结果是 4。

```
String str="hello world";
System.out.println(str.indexOf('o',6));
```

打印结果是 7。

indexOf 还有其他几种重载形式，如：

```
indexOf(String str)
indexOf(String str,int fromIndex)
```

这两个函数返回一个子字符串在该字符串中的首次出现的位置，读者掌握了上面对单个字符进行操作的方法，自然也就明白了如何使用这两个函数。

subString(int beginIndex)方法返回的是在一个字符串中从beginIndex指定的数值到末尾的一个字符串，如果beginIndex指定的数值超过了当前字符串的长度，则返回一个空字符串。这个方法也有另外一种形式substring(int beginIndex,int endIndex)，它返回的是当前字

符串中从beginIndex开始到endIndex-1结束的一个字符串。如：

```
String str="hello world";
System.out.println(str.substring(6));
```

打印结果是world。

```
String str="hello world";
System.out.println(str.substring(6,8));
```

打印结果是wo。

关于这两个类的其他方法，读者要自己查阅JDK文档资料了。看完文档中所有String类的方法后，你也许会问："String类中的replace和toUpperCase方法不都能改变字符串的内容吗？这与你先讲的'String类对象的内容一旦被初始化就不能再改变'不是自相矛盾吗？"，请你再仔细看一下这两个函数的帮助，他们的返回类型都是String类，即生成一个新的字符串，而不是改变原来的字符串内容。

args【】参数的传递

另外，包装类对象在进行基本数据类型的类型转换时也有所有用，如整数与字符串的转换。下面的程序用于在屏幕上打印出一个星号（*）组成的矩形，矩形的宽度和高度通过运行时为程序传递的参数指定。

程序清单：TestInteger.java

```
class TestInteger
{
    public static void main(String[] args)
    {
        int w = Integer.parseInt(args[0]);  //第一种方法
        int h = new  Integer(args[1]).intValue(); //第二种方法
    //int h = Integer.valueOf(args[1]).intValue(); //第三种方法




    for(int i=0;i<h;i++)
    {
        StringBuffer sb-new StringBuffer();
        for(int j=0 ;j<w;j++)
        {
            sb.append('*');
        }
        System.out.println(sb.toString());
    }
    }
}
```

在上面的程序中，我们想打印出一个由*号组成的矩形，这时需要指定矩形的长度和宽度两个值，可以在运行这个程序时，将矩形的长度和宽度作为程序参数传递进去。在程序中，可以直接从 args 数组中取出传递进的程序参数，但这些参数是字符串型的，不能直接使用，所以我们需要实现字符串到整数之间的转换。这时，就需要用到整数包装类 Integer，从 JDK 文档中，很容易发现有好几种方式都可以实现字符串与整数间的转换，上面的程序给出了作者从 JDK 文档中了解到的 3 种方法，供读者参考。注意，我们在打印的这个过程中用到了 StringBuffer 类的 append 方法，这也是复习前面讲的内容。

6.5 集 合 类

在 Java 编程中，经常会用到 Vector、Enumeration、ArrayList、Collection、Iterator、Set、List 等集合类和接口。

1．Vector 类与 Enumeration 接口

Vector 类是 Java 语言提供的一种高级数据结构，可用于保存一系列对象，Java 不支持动态数组，Vector 类提供了一种与“动态数组”相近的功能。如果我们不能预先确定要保存的对象的数目，或是需要方便获得某个对象的存放位置时，Vector 类都是一种不错的选择。

下面的程序实现了在键盘上输入一个数字序列并存储在某种数据结构中，最后在屏幕上打印出每位数字相加的结果。

程序清单：TestVector.java

```
import java.util.*;  //下面用到的 Vector 类和 Enumeration 接口都在此包中
public class TestVector
{
    public static void main(String [] args)
    {
        int b=0;
        Vector v=new Vector();
        System.out.println("Please Enter Number:");
        while(true)
        {
            try
            {
```

```
                b= System.in.read();
            }
            catch(Exception e)
            {
                System.out.println(e.getMessage());
            }
            if(b=='\r' || b== '\n')
                break;
            else
            {
                int num=b-'0';
                v.addElement(new Integer(num));
            }
        }
        int sum=0;
        Enumeration e=v.elements();
        while(e.hasMoreElements())
        {
            Integer intObj=(Integer)e.nextElement();
            sum += intObj.intValue();
        }
        System.out.println(sum);

    }
}
```

运行结果：

输入 32　　打印 5
输入 1234　　打印 10

在上面的例子中，因为不能预先确定输入数字序列的位数，所以不能使用数组来存储每一个数值。正因为如此，我们选择了 Vector 类来保存数据。Vector.addElements 只能接受对象类型的数据，先用 Integer 类包装了整数后，再用 Vector.addElements 方法向 Vector 对象中加入这个整数对象。最后，我们要取出保存在 Vector 对象中的所有整数进行相加，首先必须通过 Vector.elements 方法返回一个 Enumeration 接口对象，再用 Enumeration.nextElement 方法逐一取出保存的每个整数对象，Enumeration 对象内部有一个指示器指向调用 nextElement 方法时要返回的对象的位置。

2. Collection 接口与 Iterator 接口

接口的类来创建对象，ArrayList 类就是一个实现了 Collection 接口的类，我们将上面使用 Vector 和 Enumeration 的例子改为用 ArrayList 和 Iterator 编写，读者就应该基本明白了这些类之间的关系和用法。

程序清单：TestCollection.java

```
import java.util.*;  //ArrayList 类和 Iterator 接口都在此包中
public class TestCollection
{
    public static void main(String [] args)
    {
        int b=0;
        ArrayList al=new ArrayList();
        System.out.println("Please Enter Number:");
        while(true)
        {
            try
            {
              b= System.in.read();
            }
            catch(Exception e)
            {
                System.out.println(e.getMessage());
            }
  if(b=='\r' || b== '\n')
      break;
  else
```



```
      {
          int num=b-'0';
          al.add(new Integer(num));
      }
  }
  int sum=0;
  Iterator itr=al.iterator();
  while(itr.hasNext())
  {
      Integer intObj=(Integer)itr.next();
      sum += intObj.intValue();
```

```
                sum += intObj.intValue();
            }
            System.out.println(sum);
        }
}
```

运行结果与前面的 TestVector.java 相同：

```
输入 32        打印 5
输入 1234      打印 10
```

Vector 和 arraylist 的区别

爱思考的读者也许要问：我什么时候用 Vector，什么时候用 ArrayList 呢？Vector 类中的所有方法都是线程同步的，两个线程并发访问 Vector 对象将是安全的，但只有一个线程访问 Vector 对象时，因为源程序仍调用了同步方法，需要额外的监视器检查，运行效率要低些。

ArrayList 类中的所有方法是非同步的，所以在没有多线程安全问题时，最好用 ArrayList，程序的效率会高些。在有线程安全问题，且我们的程序又没有自己处理（自己处理是指对调用 ArrayList 的代码或方法加上同步代码同步处理）时，只能用 Vector。

集合类接口的比较

3．集合类接口的比较

另外还有几个集合类接口 Set、List，下面是 Collection 和它们的比较。

Collection——对象之间没有指定的顺序，允许重复元素。

Set——对象之间没有指定的顺序，不允许重复元素。

List——对象之间有指定的顺序，允许重复元素。

这三个接口的继承关系如图 6.3 所示。

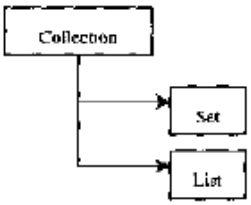

图 6.3

多学两招：

由于在 List 接口中，对象之间有指定的顺序。因此我们可以对 List 接口的对象进行排序。

程序清单：TestSort.java

```
import java.util.*;
public class TestSort
{
    public static void main(String[] args)
    {
        ArrayList al=new ArrayList();
        al.add(new Integer(1));
        al.add(new Integer(3));
        al.add(new Integer(2));
        System.out.println(al.toString()); //排序前
        Collections.sort(al);
        System.out.println(al.toString()); //排序后
    }
}
```

运行结果：

```
[1, 3, 2]
[1, 2, 3]
```

在上面的程序中，我们向实现了 List 接口的 ArrayList 类对象中添加了 3 个成员。然后用 Collections 类的 sort 静态方法对其进行排序。

Hashtable 的用法

6.6 Hashtable 与 Properties 类

Hashtable 也是一种高级数据结构，用以快速检索数据。Hashtable 不仅可以像 Vector 一样动态存储一系列的对象，而且对存储的每一个对象（称为值）都要安排另一个对象（称为关键字）与之相关联。例如，我们可以在 Hashtable 中存储若干国家的中文和英文名称，并且能够通过英文检索出对应的中文名称，这里中文就是值，英文就是关键字。

向 Hashtable 对象中存储数据，使用的是 Hashtable.put(Object key,Object value)方法，从 Hashtable 中检索数据，使用 Hashtable.get(Object key)方法。值和关键字都可以是任何类型的非空的对象。

Hashtable.put(Object key, Object value);

Hashtable.get(Object key);

的非空的对象。

下面的代码产生一个存储数字的 Hashtable，用英文数字作为关键字。

```
Hashtable numbers = new Hashtable();
numbers.put("one", new Integer(1));
numbers.put("two", new Integer(2));
numbers.put("three", new Integer(3));
```

要检索其中"two"关键字对应的数据，看下面的代码就能明白。

```
Integer n = (Integer)numbers.get("two");
if (n != null)
{
    System.out.println("two = " + n);
}
```

下面是我们编写的一个类用于关键字时，是如何实现 equals 和 hashCode 这两个方法的。

```
class MyKey
{
    private String name;
```

```
    private int age;
    public MyKey(String name,int age)
    {
        this.name = name;
        this.age = age;
    }
    public String toString()
    {
        return new String(name + "," + age);
    }
    public boolean equals(Object obj)
    {
        if(name.equals(obj.name) && age==obj.age)
            return true;
        else
            return false;
    }
    public int hashCode()
    {
        return name.hashCode() + age;
    }
}
```

这个类实现了我们的目的：如果两个人名字和年龄都相同，我们就认为他们是同一个人。

6.7 System 类与 Runtime 类

6.7.1 System 类

Java 不支持全局函数和变量，Java 设计者将一些系统相关的重要函数和变量收集到了一个统一的类中，这就是 System 类。System 类中的所有成员都是静态的，当我们要引用这些变量和方法时，直接使用 System 类名作前缀，我们前面已经使用到了标准输入和输出的 in 和 out 变量，关于这两个变量的详细介绍，请看第 7 章。下面再介绍 System 类中几个方法，其他的方法读者还是参看 JDK 文档资料。

exit(int status)方法，提前终止虚拟机的运行。对于发生了异常情况而想终止虚拟机的运行，传递一个非零值作为参数。对于在用户正常操作下，终止虚拟机的运行，传递零值作为参数。

CurrentTimeMillis 方法返回自 1970 年 1 月 1 日 0 点 0 分 0 秒起至今的以毫秒为单位的时间，这是一个 long 类型的大数值。在计算机内部，只有数值，没有真正的日期类型及其他各种类型，也就是说，我们平常用到的日期本质上就是一个数值，但通过这个数值，能够推算出其对应的具体日期时间。

多学两招：

我们还可用 CurrentTimeMillis 方法检测一段程序代码运行时所花费的时间：

```
    Long startTime=System.currentTimeMillis();
    ......  //代码段
    Long endTime= System.currentTimeMillis();
System.out.println("total time exponded is "  + (starttime - endTime) +
" milliseconds");
```

getProperties 方法与 Java 的环境属性。

getProperties 方法获得当前虚拟机的环境属性。如果大家明白 Windows 的环境属性，如我们在第 1 章中讲到的 path 和 classpath 就是其中的两个环境变量，每一个属性都是变量与值成对的形式出现的。

同样的道理，Java 作为一个虚拟的操作系统，它也有自己的环境属性，Properties 是 Hashtable 的子类，正好可以用于存储环境属性中的多个变量/值对格式的数据，getProperties

方法返回值是，包含了当前虚拟机的所有环境属性的 Properties 类型的对象。下面的例子打印出当前虚拟机的所有环境属性的变量和值。

6.7.2 Runtime 类

Runtime 类封装了 Java 命令本身的运行进程，其中的许多方法与 System 中的方法相重复。我们不能直接创建 Runtime 实例，但可以通过静态方法 Runtime.getRuntime 获得正在运行的 Runtime 对象的引用。

Exec 方法，Java 命令运行后，本身是多任务操作系统上的一个进程，在这个进程中启动一个新的进程，即执行其他程序时使用 exec 方法。exec 方法返回一个代表子进程的 Process 类对象，通过这个对象，Java 进程可以与子进程交互：

程序清单：TestRuntime.java

```
public class TestRuntime
{
    public static void main(String[] args)
    {
        Process p=null;
       try
        {
            p=Runtime.getRuntime().exec("notepad.exe TestRuntime.java");
            Thread.sleep(5000);
        }
```

运行后程序启动一个子进程，用 Windows 的记事本程序打开了我们的源程序，并在 5 秒钟后销毁该子进程，记事本程序被关掉。

由于程序不能直接创建类 Runtime 的实例，所以可以保证我们只会产生一个 Runtime 的实例对象，而不能产生多个实例对象，这种情况就是我们前面曾经讲过的单态设计模式。读者可以按照单态设计模式思想，来设想一下 Runtime 类在内部是如何构造 Runtime 类的对象实例的。

6.8 Date 与 Calendar、DateFormat 类

Date 类用于表示日期和时间，最简单的构造函数是 Date()，它以当前的日期和时间初始化一个 Date 对象。由于开始设计 Date 时没有考虑到国际化，所以后来又设计了两个新的类来解决 Date 类中的问题，一个是 Calendar 类，一个是 DateFormat 类。

Calendar 类是一个抽象基类，主要用于完成日期字段之间相互操作的功能，如 Calendar.add 方法可以实现在某一日期的基础上增加若干天（或年、月、小时、分、秒等日期字段）后的新日期，Calendar.get 方法可以取出日期对象中的年、月、日、小时、分、秒等日期字段的值，Calendar.set 方法修改日期对象中的年、月、日、小时、分、秒等日期字段的值。Calendar.getInstance 方法可以返回一个 Calendar 类型（更确切地说是它的某个子类）的对象实例，GregorianCalendar 类是 JDK 目前提供的一个惟一的 Calendar 子类，Calendar.getInstance 方法返回的就是预设了当前时间的 GregorianCalendar 类对象。

下面的例子将“2002-03-15”格式的日期字符串转换成“2002年03月15日”。

程序清单：TestDateFormat.java

```
import java.util.*;
import java.text.*;
public class TestDateFormat
{
    public static void main(String[] args)
    {
        SimpleDateFormat sdf1=new SimpleDateFormat("yyyy-MM-dd");
SimpleDateFormat sdf2=new SimpleDateFormat("yyyy年MM月dd日");


            try
            {
                Date d=sdf1.parse("2003-03-15");
                System.out.println(sdf2.format(d));
            }
            catch(Exception e)
            {
                System.out.println(e.getMessage());
            }
        }
    }
```

运行结果：

2003年03月15日

SimpleDateFormat类就相当于一个模板，其中yyyy对应的是年，MM对应的是月，dd对应的是日，更详细的细节查阅JDK文档，关于这些参数，JDK中写得非常清楚。

在上面程序中，我们定义了一个SimpleDateFormat类的对象sdf1来接收和转换源格式字符串“2003-03-15”，随后又定义了该类的另一个对象sdf2来接收sdf1转换成的Date类的对象，并按sdf2所定义的格式转换成字符串。

在这个过程中，我们已经实现了利用SimpleDateFormat类来把一个字符串转换成Date类对象，和把一个Date对象按我们指定的格式输出的两个功能。

6.9 Math 与 Random 类

Math 类包含了所有用于几何和三角的浮点运算函数，这些函数都是静态的，每个方法的使用都非常简单，读者一看 JDK 文档就能明白。

Random 类是一个伪随机数产生器，随机数是按照某种算法产生的，一旦用一个初值创建 Random 对象，就可以得到一系列的随机数，但如果用相同的初值创建 Random 对象，

得到的随机数序列是相同的，也就是说，在程序中我们看到的“随机数”是固定的那些数，起不到“随机”的作用，针对这个问题，Java 设计者们在 Random 类的 Random()构造方法中使用当前的时间来初始化 Random 对象，因为没有任何时刻的时间是相同的，所以就可以减少随机数序列相同的可能性。