HZ BOOKS

软 件 开 发 技 术 丛 书

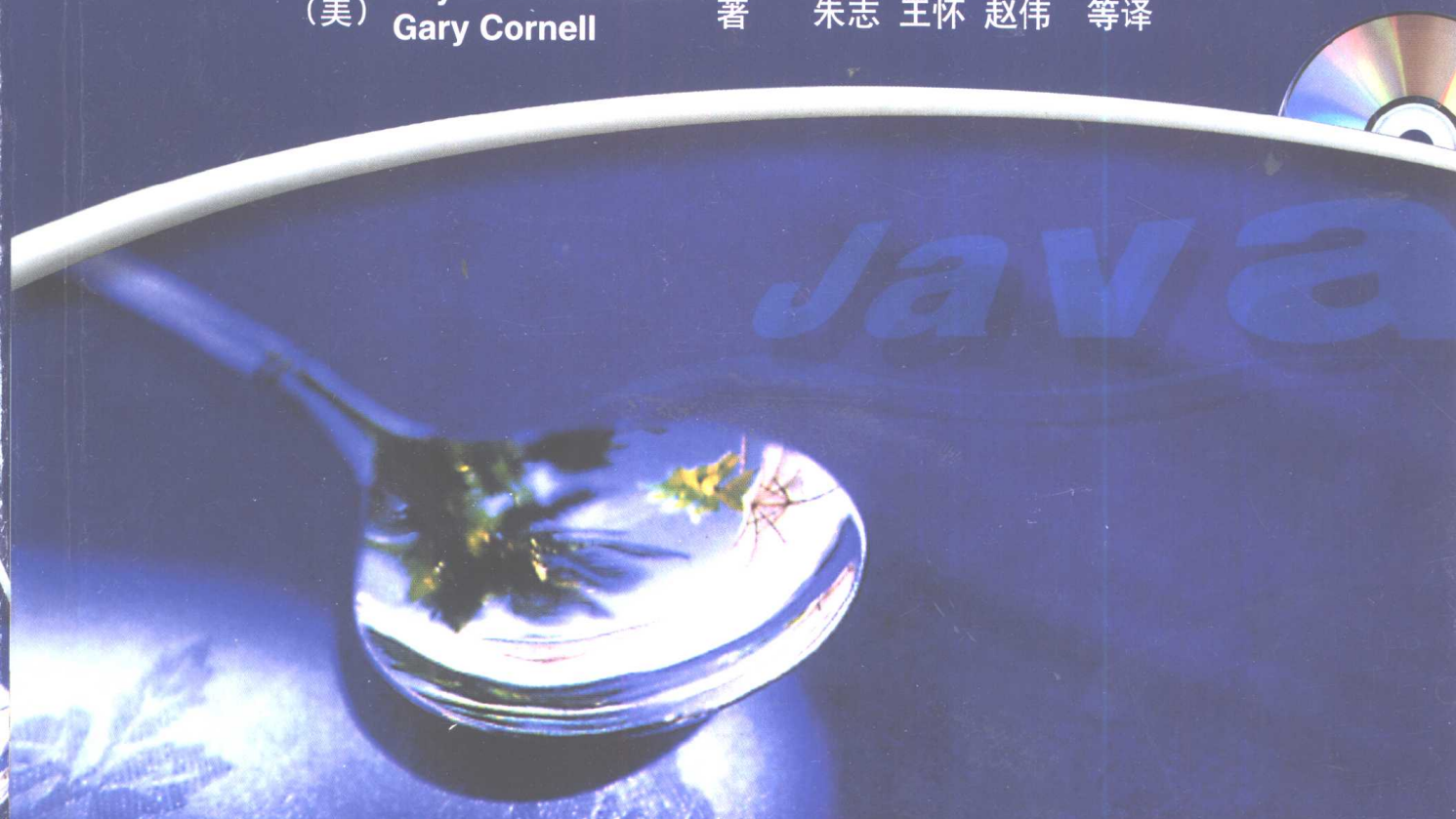
Sun microsystems

Java 2 核心技术

卷 II: 高级特性

Core Java 2 Volume II: Advanced Features

（美）Cay S. Horstmann
Gary Cornell 著　朱志 王怀 赵伟 等译



机械工业出版社
China Machine Press

Prentice Hall

软件开发技术丛书

Java 2核心技术
卷II: 高级特性

(美) Cay S. Horstmann Gary Cornell 著

朱 志 王 怀 赵 伟 等译

邓光伟 李丙午 审校

机械工业出版社
China Machine Press

本书详细介绍了程序员需要掌握的Java语言高级特性，阐述了Java语言高级编程的相关特性。本书配套光盘提供了最新Java开发包、共享软件、书中所有示例源代码等内容。本书既适合Java的编程高手阅读，也适合其他有关人员参考学习。

本书版权登记号：图字：01-2000-1318

图书在版编目(CIP)数据

Java 2 核心技术 卷II：高级特性 / (美) 霍斯曼(Horstmann,C.S.)，(美)柯内尔(Cornell,G.)著；朱志等译. - 北京：机械工业出版社，2000.11
(软件开发技术丛书)
书名原文：Core Java 2 Volume II: Advanced Features
ISBN 7-111-08244-3

I. J… II. ①霍…②柯…③朱… III. Java语言 - 程序设计 IV. TP312

中国版本图书馆CIP数据核字(2000)第46817号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）
责任编辑：李云静
北京牛山世兴印刷厂印刷 · 新华书店北京发行所发行
2000年11月第1版 · 2001年1月第2次印刷
787mm × 1092 mm 1/16 · 43.75印张
印数：6 001-8 000册
定价：88.00元（附光盘）

译 者 序

随着计算机技术和应用的普及，Java编程语言愈来愈受到计算机开发人员的喜爱和重视。本书在《Java 2核心技术 卷I：基础知识》(机械工业出版社出版)的基础上，重点介绍了多线程、数据结构、高级Swing组件、网络编程、安全性和本地化等Java 2提供的高级特性及一些相关的组件；这些工具为开发人员提供了功能强大、界面一流的开发环境，使Java的跨平台、面向对象、可移植性及网络程序设计等特性更易实现，这必将进一步推动Java应用的普及。

本书详细介绍了一个致力于专业软件开发的程序员需要知道的Java语言高级特性，阐述了Java语言高级编程的相关特性。本书配套光盘提供了最新Java开发包、共享软件、书中所有程序源代码等内容。本书各章相互独立，你可以研究自己最感兴趣的课题，也可以以自己喜欢的任何顺序来读各个章节。《Java 2核心技术 卷I：基础知识》介绍的是Java语言的基本知识，本书介绍了程序员需要掌握的Java语言高级课题。读者对象仍是想把Java技术用于开发实际项目的程序员。如果你是一个有经验的开发者，即使不看卷I也可以从本书中获益不小。

参加本书翻译工作的人员有：朱志、王怀、赵伟、李丙午、邓光伟、刘小明。全书由朱志、赵伟审校。全书在翻译过程中得到梁晓艳、张景生等同志大力支持，负责文字录入的人员有：孙标、周学广、鲍华、张有中、彭一庆、李英、王和菊、刘有章、张和、陶玉、李金根、谭伟、张景、蒋永标、苏颖、林宏、李红、李伟、何远、何仝、何华京、刘后、刘兵、陈戈林、周丽萍、黄宁、陈亮、潘静、张建中、孙霞、葛强、王晶、洪是惠、彭庆、李晓英、王菊、刘章、张和力、孙霞、葛强、张健英等，在这里对他们的辛勤劳动表示感谢。

由于本书篇幅庞大，译者水平有限，加之时间仓促，书中错漏之处在所难免，敬请广大读者批评指正。

前　言

致读者

本书是《Java核心技术》第4版的第II卷。它的第1版发布于1996年的上半年，第2版发布于1996年的下半年，第3版于1997/1998发布。它的前两版都是包含一本书。当开始编著第3版时，我们很清楚自己不可能在一本书中覆盖一个严肃的程序员需要知道的关于Java平台的所有特性。因而，就把第3版分成两卷。在第4版中，我们仍把这些材料组织成了两卷。

卷I覆盖了Java语言的基本知识；而本书包含了程序员需要掌握的Java语言高级课题。本书的对象仍是想把Java技术用于开发实际项目的程序员。请注意：如果你是一个有经验的开发者，并熟悉新的事件模型和像内类（inner class）这样的高级语言特性，即使不看卷I也可以从本书中获益不小。（当然，在适当的时候参考了卷I的有关章节，即机械工业出版社出版的《Java 2核心技术 卷I：基础知识》一书）。

错误和不精确的地方在所难免。我们非常乐意知道这方面的问题。为此，我们建了一个主页，专门陈列本书的常见问题（FAQ）、错误修正以及解决方案等等，地址是：http://www.horstmann.com/corejava.html.。FAQ的末尾是一份表格（希望你先看完FAQ的内容），可用它提交你发现的错误，以便再版时完善本书。

本书结构

总的来说，本书的各章是相互独立的。你可以研究自己最感兴趣的课题，也可以以自己喜欢的任何顺序来读各章内容。

第1章介绍多线程，它能使你并行完成多个任务（一个线程就是程序中一个单独的控制流）。介绍如何建立线程和如何确定没有线程被阻塞/挂起。我们将通过介绍构造计时器和制作动画所需的技巧把这些知识运用到实例中。

第2章系统地介绍了Java 2的数据结构平台。当收集多个对象并在以后的工作中检索它们时，你当然不想把它们杂乱无章地存放于矢量中，而需要一种能有效检索它们的数据结构。本章介绍如何利用Java库提供的标准数据结构来满足这种需要。

第3章向大家展示了Java平台上最激动人心的API特性：网络API。Java平台大大简化了复杂的网络编程，本章详细地介绍了API的特性，同时也说明了小应用程序安全模型对网络编程的重要性。

第4章讨论了JDBC,即Java数据库连接API。介绍了如何用JDBC API的核心子集开发处理数据库事务程序。注意本章介绍的只是JDBC API的一部分。

第5章介绍了远程对象和远程方法调用（RMI）的相关知识。这个API提供了同时处理分布于不同机器上的多个对象的功能，本章也展示了“对象处处存在”的实际用途。

第6章探讨了卷I没有涉及的Swing的所有高级组件，重点介绍了重要且复杂的树和表格组件。本章介绍了编辑窗格的基本用法及“多个文档”接口的Java技术实现，把着重点放在了

实际编程中可能碰到的大多数有用的组件，因为全部介绍Swing包需很长的篇幅。

第7章讨论了Java 2D API，你可以用这个接口描绘真正的图画。本章也涉及了AWT(Abstract Windowing Toolkit，抽象窗口工具包)的某些高级特性，这些特性如在卷I介绍，似乎有凑篇幅的嫌疑，但它确实应成为程序员工具包的一部分，它包括打印及用于剪贴和拖放的API。实际上，我们对剪贴部分API的介绍比Sun公司更进了一步：介绍了如何通过系统剪贴板在Java编程语言编写的不同程序间剪贴可系列化的Java对象。

第8章展示了Java平台的组件API（即JavaBeans）。你将了解如何编写自己的bean，以便让其他程序员在集成编程环境中利用它们（但我们并不介绍各种不同的用于控制bean的编程环境）。JavaBeans组件技术对Java技术的最终成功起了不可估量的作用，因为正是这项技术提供了便于使用和操作的用户接口编程环境。当然，这些组件是用Java编程语言开发的，它们在两个方面是ActiveX 控件无法相比的：其他平台可以立即使用它们及它们也同样使用于Java平台的高级安全模型。

第9章讲述Java平台安全模型。Java平台设计的出发点之一是要保证安全，本章介绍这个安全模型是如何实现的。它讨论了如何编写自己的类装载器和安全管理器，以满足自己开发的应用程序，本章还介绍提供重要特性（如签名类）的新的安全API。

第10章讨论了Java语言的另外一个特性：国际化，我们相信这个特性会越来越重要。Java编程语言是少数几个从一开始就能处理Unicode的语言之一。但Java语言的国际化特性提供了更强大的功能。因而，你可以用它国际化自己的应用程序。例如，我们将向你展示如何编写一个退休金计算器程序，它可以使用英语、德语或中文——这取决于其浏览器所用的语言。

第11章论述了本地方法，它们可让你调用为某个指定类型的机器（如Microsoft Windows API）编写的方法。很明显，这个特性会引起争议：使用本地方法，就丧失了Java平台交互特性所带来的优点。但不管怎么说，每个为指定平台开发Java应用程序的认真负责的程序员需了解这些技术。因为很可能出现这样的情况：编写应用程序时，需参阅你目标平台的操作系统的API。我们通过展示如何访问Windows的登录功能来说明这一点。

约定

正文中的“C++注释”解释了Java和C++的差异。如果读者对C++语言不感兴趣，或者本来就没有这方面的背景，跳过不读便是。

“注意”和“提示”指出了一些技巧和参考信息。

如果要指出一项潜在的危险，“警告”栏会提醒你注意。

Java平台拥有一个大型的编程库或应用程序接口（API）。如果首次用到一个API调用，我们便对其进行简要注释。

本书配套光盘包含了作为例子列出的程序的源代码，例如，例5-7: Warehouse.java可在光盘上找到相应的代码。你也可以从网上下载这些例程文件。

定义

一个Java对象 (Java object) 就是一个用Java编程语言编写的程序创建的对象。

一个Java应用程序 (Java application) 就是一个用Java编程语言编写的程序，并能在Java虚拟机运行（Java虚拟机是基于Java平台的虚拟机）。

目　录

第1章 多 线 程

- 线程的属性
- 线程的优先权
- 合作和利己线程
- 同步
- 动画
- 计时器
- 线程和Swing组件
- 利用管道实现线程间的通信

你可能对多任务很熟悉，多任务能运行两个或两个以上的程序，且这些程序似乎在同时运行。例如，在编辑或发传真的同时还能打印文件。当然，除非你的机器拥有多个处理机，否则是不能同时运行两个或两个以上的程序的，真正发生的是操作系统把资源分配给各个程序，给人留下并发活动的印象。当一个计算机用户认为他的活动（例如，输入数据）使计算机处于忙碌状态时，实际上CPU的大部分时间处于空闲状态（毕竟，一个快的打字员打一个字符大约需1/20秒），所以，这种资源分配是可能的。

根据操作系统中断程序时是否先和程序协商，或是否只有程序自愿放弃控制时它们才被中断，可以用两种方法实现多任务。前者叫抢占式多任务 (preemptive multitasking)；后者叫合作式（或就叫非抢占式）多任务 (cooperative multitasking)。Windows 3.1 是一个合作式多任务系统，而Windows NT(和用于32位编程的Windows 95)是抢占式的。（虽然较难实现，但抢占式任务有效得多。对于合作式多任务来说，一个运行不好的程序会占有整个系统。）

多线程通过把多任务的原理用到程序的更低一层中进一步发展了这一原理。单个程序似乎能同时执行多个任务。每个任务通常被称为一个线程（thread）——它是线程控制流的简称。能同时运行一个以上线程的程序被称为多线程程序。设想每个线程在各自的环境 (context) 中运行：似乎使每个线程都有自己的CPU，同时还有自己的寄存器、内存和代码。

那么，多进程和多线程之间的区别是什么？基本的区别是每个进程都有一组完整的属于自己的变量，而线程共享这些数据。这好象有点不安全，事实上确实如此，这一点你可以在本章的后面看到。但创建和注销单个线程比运行新进程所需的开销少得多，这就是为什么现在所有的操作系统都支持多线程的原因。而且，和线程间的通信相比，进程间的通信慢得多，要求也较高。

实际上多线程非常有用：举个例子来说，一个浏览器应能浏览多个主机或能打开一个e-mail窗口或在下载数据的同时浏览其他网页。Java语言本身用一个线程在后台收集无用单元——这样就减少了你管理内存的麻烦！GUI程序有一个单独的线程用于收集来自于主机运行环境的用户接口事件。本章将向你说明如何在Java应用程序(application)和 小应用程序（Applet）中添加多线程功能。

忠告 多线程可能会非常复杂。本章中将介绍Java编程语言提供的所有用于线程编程

的工具，解释它们的用法和使用范围并提供一些简单典型的例子。对于它们更进一步的用法，我们建议你参考其他更详细的资料。

注意 在许多编程语言中，如果你想执行多线程的程序，就不得不使用一个外部线程包。而Java本身有内置的多线程包，这使得你的工作变得容易多了。

1.1 什么是线程

让我们从研究一个不使用多线程的程序开始，当然，这样的程序也不便于用户用它执行多个任务。当我们解剖这个程序后，将说明用它运行单独的线程会变得多么容易。它将激活一个弹球，并让这个小球不停地移动，当小球碰到边框时，它将重画这个小球（参见图1-1）。

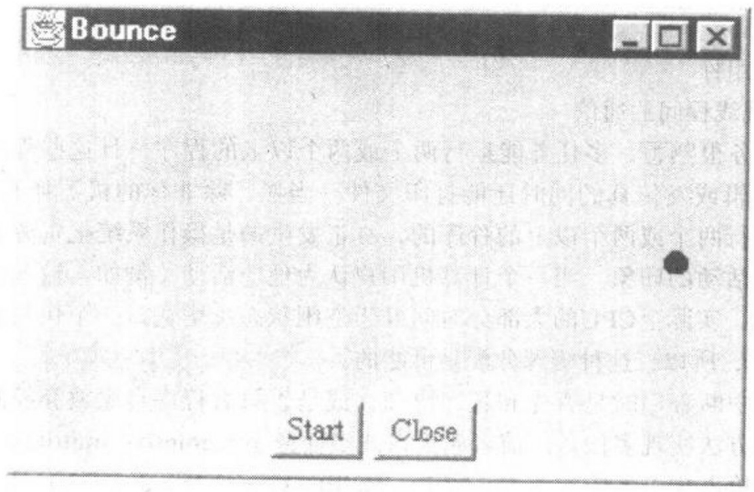


图1-1 用线程激活一个弹球

当你一用鼠标点击Start按钮，从屏幕的左上角就会出现一个小球并开始弹跳。Start 按钮的控制器调用了类Ball中的bounce()方法，这个方法包含了一个跳动1000次的循环。在每次跳动后，我们调用类Thread中的静态方法sleep()让小球睡眠5毫秒。

```
class Ball
{  . . .
   public void bounce()
   {  draw();
      for (int i = 1; i <= 1000; i++)
      {  move();
         try
         {  Thread.sleep(5);
         }
         catch(InterruptedException e)
         {
         }
      }
   }
}
```

调用Thread.sleep并不能创建一个新线程——sleep是类Thread中的静态方法，它使当前线

程进入睡眠状态。

sleep方法会抛出一个异常（或违例）InterruptedException。我们将在后面讨论这个异常和对它的正确处理方法。

如果运行这个程序，你会看到这个小球在屏幕上来回弹跳，似乎不错，但它已完全接管了这个应用程序。假如你想在小球弹1000次之前终止它的这个动作，并用鼠标点击Close按钮，但小球还是继续弹跳。在小球完成1000次弹跳之前，用户和程序之间是不能发生交互作用的。

这种情况不论在理论上还是在实际中都不太好，而且随着网络的越来越普及，这个问题会变得越来越突出。毕竟，当你从网络上读取数据被阻塞时（这会经常发生，因为它是个费时的任务），会希望立即中断它。举个例子来说，假设你正在下载一个大的图像文件，但在看了它的一部分后，觉得不再需要剩余的部分，这时可以单击Stop或Back按钮来中断这个下载过程。在下一节，我们将说明如何通过运行一个独立线程代码的关键部分来使用户始终控制这个程序。

例1-1是这个程序的全部源代码。

例1-1 Bounce.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Bounce
{  public static void main(String[] args)
   {  JFrame frame = new BounceFrame();
      frame.show();
   }
}

class BounceFrame extends JFrame
{  public BounceFrame()
   {  setSize(300, 200);
      setTitle("Bounce");

      addWindowListener(new WindowAdapter()
         {  public void windowClosing(WindowEvent e)
            {  System.exit(0);
            }
         } );

      Container contentPane = getContentPane();
      canvas = new JPanel();
      contentPane.add(canvas, "Center");
      JPanel p = new JPanel();
      addButton(p, "Start",
         new ActionListener()
         {  public void actionPerformed(ActionEvent evt)
            {  Ball b = new Ball(canvas);
               b.bounce();
            }
         });

      addButton(p, "Close",
         new ActionListener()
         {  public void actionPerformed(ActionEvent evt)
```

```
            {  System.exit(0);
            }
         });
      contentPane.add(p, "South");
   }

   public void addButton(Container c, String title,
     ActionListener a)
   {  JButton b = new JButton(title);
      c.add(b);
      b.addActionListener(a);
   }

   private JPanel canvas;
}

class Ball
{  public Ball(JPanel b) { box = b; }

   public void draw()
   {  Graphics g = box.getGraphics();
      g.fillOval(x, y, XSIZE, YSIZE);
      g.dispose();
   }

   public void move()
   {  Graphics g = box.getGraphics();
      g.setXORMode(box.getBackground());
      g.fillOval(x, y, XSIZE, YSIZE);
      x += dx;
      y += dy;
      Dimension d = box.getSize();
      if (x < 0)
      { x = 0; dx = -dx; }
      if (x + XSIZE >= d.width)
      { x = d.width - XSIZE; dx = -dx; }
      if (y < 0)
      { y = 0; dy = -dy; }
      if (y + YSIZE >= d.height)
      { y = d.height - YSIZE; dy = -dy; }
      g.fillOval(x, y, XSIZE, YSIZE);
      g.dispose();
   }

   public void bounce()
   {  draw();
      for (int i = 1; i <= 1000; i++)
      {  move();
         try { Thread.sleep(5); }
         catch(InterruptedException e) {}
      }
   }

   private JPanel box;
   private static final int XSIZE = 10;
   private static final int YSIZE = 10;
   private int x = 0;
   private int y = 0;
   private int dx = 2;
   private int dy = 2;
}
```

注意 在这个例子中，我们使用了XOR绘画模式来模拟小球的运动。这使程序本身非常简单，这样当我们创建这个例子时，你可以把注意力放在线程的细节上。然而，这种方法的显示质量不太好，因为相重叠的小球没有被正确地显示出来。下面介绍的是一种更好的方法：把所有的小球放在BouncePanel的一个向量中。它的PaintComponent方法会使所有的小球自己画自己。在Move方法调用画板的repaint方法。我们把这个改善任务留给读者自己去做。

1.1.1 利用线程给其他任务提供一个机会

下面我们将通过运行一个独立线程的代码使弹球程序的响应性更好。

注意 因为大部分计算机都没有多个处理器，Java虚拟机采用了这样的机制：每个线程都有机会运行一段时间，然后Java虚拟机唤醒其他线程并调度运行它们。一般来说，虚拟机主机的操作系统为它提供线程调度包。

在下一个程序中，我们使用了两个线程：一个是控制弹球的线程，另一个是用于管理用户接口的主线程。因为每个线程都有执行的机会，当你在小球弹跳时点击Close按钮，主线程就得到了执行的机会。然后它就开始执行Close的动作。

这里有一个运行独立线程代码的简单方法：把这些代码放在从类Thread派生出的一个类的run方法中。

我们只需从Thread类衍生出Ball类，然后把bounce更名为run，就可以把我们的弹球程序变成一个独立线程，代码如下：

```
class Ball extends Thread
{  . . .
   public void run()
    {  try
       {  draw();
          for (int i = 1; i <= 1000; i++)
          {  move();
             sleep(5);
          }
       }
       catch(InterruptedException e) {}
    }
}
```

你可能已注意到我们将捕获一个称为InterruptedException的异常。当你的线程因为其他线程调用interrupt方法而中断时，像sleep和wait这类方法就会产生这个异常。即使一个线程不处于运行状态，中断它也是引起它注意的突然方式，特别是中断它以使它终止时更是如此。相应地，当一个InterruptedException异常发生时，我们的run方法会退出。

1.1.2 运行和启动线程

当你构造一个从Thread类衍生出的对象时，run方法不会被自动调用。

```
Ball   b = new Ball(. . .) ; // 不会运行
```

你应该在对象中调用start方法来确实启动一个线程。

```
b.start ();
```

注意 不要直接调用run方法——当一个线程已被创建并准备运行时start方法将调用它。直接调用run方法只执行同一线程中的内容——却不能启动新线程。

在Java语言中,当一个线程处于空闲状态时，它必须通知其他线程，以使它们抓住机会执行它们run方法中的代码（见图1-2）。完成这个任务一般是通过静态的sleep方法。Ball类中的run方法调用sleep(5)以表示这个线程在随后的5毫秒处于空闲状态，它会再次执行，但同时其他线程也获得了执行的机会。

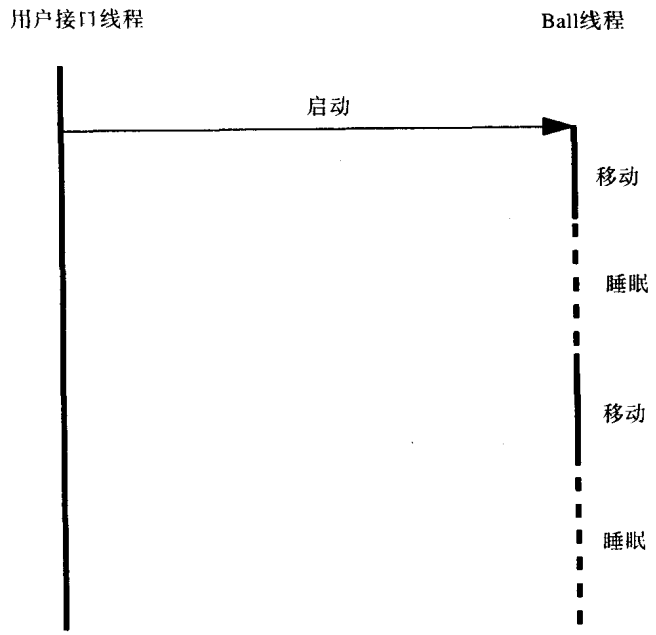


图1-2 UI(用户接口)和Ball线程

提示 Thread类中有许多静态方法。它们都是在当前线程（就是执行这个方法的线程）上运行的。例如，静态的sleep方法使调用sleep的线程进入空闲状态。

从设计的角度看，让Ball类继承Thread类似乎有点怪。小球是一个在屏幕上移动并在碰到屏幕角后反弹回的对象。难道继承性的“是一个（is-a）”规则适用于这里？难道一个小球是一个线程？并不真是这样。因为技术原因我们在这里必须精确地使用继承性。为了得到一个你能控制的线程，需要拥有一个包含run方法的线程对象。我们可以把run方法添加到这样的类中：run方法能使用它的方法和实例域。因而，我们把Ball类做成Thread类的子类.而且，这样也使本例容易理解：你构造的每个Ball类对象在自己的线程中启动和执行自己的run方法。在本章的后面你将学会用Runnable接口以避免继承Thread类。

提示 实际上使用Runnable接口来代替从Thread类派生出子类总是一个不错的想法。然而，在本章中，我们使用从Thread类派生出子类的方法，这样能使读者对什么线程正在执行一目了然。

例1-2将给出这个程序完整的代码。

例1-2 BounceThread.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class BounceThread
{  public static void main(String[] args)
      JFrame frame = new BounceThreadFrame();
      frame.show();
   }


class BounceThreadFrame extends JFrame
{  public BounceThreadFrame()
   {  setSize(300, 200);
      setTitle("Bounce");

      addWindowListener(new WindowAdapter()
         {  public void windowClosing(WindowEvent e)
            {  System.exit(0);
            }
         } );

      Container contentPane = getContentPane();
      canvas = new JPanel();
      contentPane.add(canvas, "Center");
      JPanel p = new JPanel();
      addButton(p, "Start",
         new ActionListener()
         {  public void actionPerformed(ActionEvent evt)
            {  Ball b = new Ball(canvas);
               b.start();
            }
         });

      addButton(p, "Close",
         new ActionListener()
         {  public void actionPerformed(ActionEvent evt)
            {  canvas.setVisible(false);
               System.exit(0);
            }
         });
      contentPane.add(p, "South");
   }

   public void addButton(Container c, String title,
      ActionListener a)
   {  JButton b = new JButton(title);
      c.add(b);
      b.addActionListener(a);
   }

   private JPanel canvas;
}

class Ball extends Thread
```