

```

JMenuItem saveMenuItem = new JMenuItem("Save");
newMenuItem.addActionListener(new NewMenuListener());
saveMenuItem.addActionListener(new SaveMenuListener());
fileMenu.add(newMenuItem);
fileMenu.add(saveMenuItem);
menuBar.add(fileMenu);
frame.setJMenuBar(menuBar);
frame.getContentPane().add(BorderLayout.CENTER, mainPanel);
frame.setSize(500, 600);
frame.setVisible(true);
}

public class NextCardListener implements ActionListener {
    public void actionPerformed(ActionEvent ev) {
        QuizCard card = new QuizCard(question.getText(), answer.getText());
        cardList.add(card);
        clearCard();
    }
}

public class SaveMenuListener implements ActionListener {
    public void actionPerformed(ActionEvent ev) {
        QuizCard card = new QuizCard(question.getText(), answer.getText());
        cardList.add(card);

        JFileChooser fileSave = new JFileChooser();
        fileSave.showSaveDialog(frame);
        saveFile(fileSave.getSelectedFile()); ←
    }
}

public class NewMenuListener implements ActionListener {
    public void actionPerformed(ActionEvent ev) {
        cardList.clear();
        clearCard();
    }
}

private void clearCard() {
    question.setText("");
    answer.setText("");
    question.requestFocus();
}

private void saveFile(File file) {
    try {
        BufferedWriter writer = new BufferedWriter(new FileWriter(file));

        for(QuizCard card:cardList) {
            writer.write(card.getQuestion() + "/");
            writer.write(card.getAnswer() + "\n");
        }
        writer.close(); ←
    } catch(IOException ex) {
        System.out.println("couldn't write the cardList out");
        ex.printStackTrace();
    }
}

```

创建菜单，把new与save项目加到File下，然后指定frame使用这个菜单，菜单项目会触发ActionEvent

调出存盘对话框 (dialog)  
等待用户决定，这都是靠JFileChooser完成的

实际编写文件的方法由SaveMenuListener的事件处理器所调用，稍后会介绍File这个类

将Buffered Writer链接到Filewriter，稍后也有说明

将ArrayList中的卡片逐个写到文件中，一行一张卡片，问题和答案由“/”分开

## java.io.File class

File这个类代表磁盘上的文件，但并不是文件中的内容。啥？你可以把File对象想象成文件的路径，而不是文件本身。例如File并没有读写文件的方法。关于File有个很有用的功能就是它提供一种比使用字符串文件名来表示文件更安全的方式。举例来说，在构造函数中取用字符串文件名的类也可以用File对象来代替该参数，以便检查路径是否合法等，然后再把对象传给FileWriter或FileInputStream。

你可以对File对象做的事情：

- ① 创建出代表现存盘文件的File对象。

```
File f = new File("MyCode.txt");
```

- ② 建立新的目录。

```
File dir = new File("Chapter7");
dir.mkdir();
```

- ③ 列出目录下的内容。

```
if (dir.isDirectory()) {
    String[] dirContents = dir.list();
    for (int i = 0; i < dirContents.length; i++) {
        System.out.println(dirContents[i]);
    }
}
```

- ④ 取得文件或目录的绝对路径。

```
System.out.println(dir.getAbsolutePath());
```

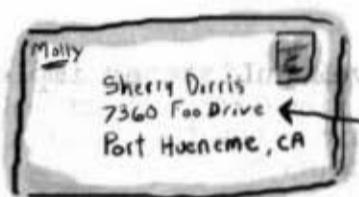
- ⑤ 删 除文件或目录（成功会返回true）。

```
boolean isDeleted = f.delete();
```

File对象代表磁盘上的文件或目录的路径名称，如：

/Users/Kathy/Data/GameFile.txt

但它并不能读取或代表文件中的数据。



地址不是房子，File对象代表特定文件的地址，但不是文件本身

File 对象代表文件名  
为 "GameFile.txt" 的文件

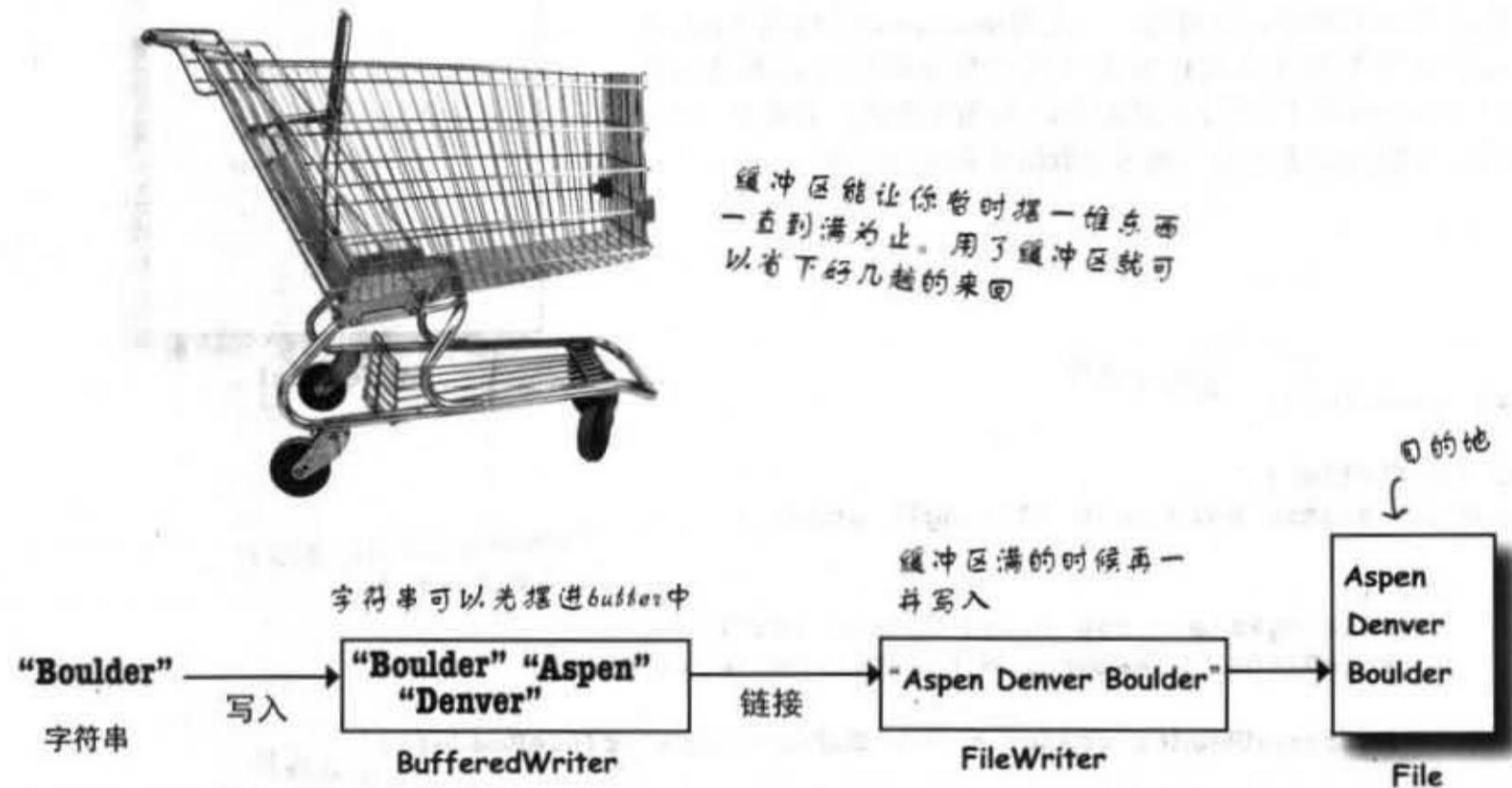
GameFile.txt

50,Elf,bow,sword,dust  
200,Troll,bare hands,big ax  
120,Magician,spells,invisibility

↑  
File 对象并不代表  
这些内容

## 缓冲区的奥妙之处

没有缓冲区，就好像逛超市没有推车一样。你只能一次拿一项东西结账。



```
BufferedWriter writer = new BufferedWriter(new FileWriter(aFile));
```

缓冲区的奥妙之处在于使用缓冲区比没有使用缓冲区的效率更好。你也可以直接使用FileWriter，调用它的write()来写文件，但它每次都会直接写下去。你应该不会喜欢这种方式额外的成本，因为每趟磁盘操作都比内存操作要花费更多时间。通过BufferedWriter和FileWriter的链接，BufferedWriter可以暂存一堆数据，然后到满的时候再实际写入磁盘，这样就可以减少对磁盘操作的次数。

如果你想要强制缓冲区立即写入，只要调用writer.flush()这个方法就可以要求缓冲区马上把内容写下去。

注意此处不需要持有对FileWriter对象的引用，我们只在乎BufferedWriter

## 读取文本文件

从文本文件读数据是很简单的，但是这次我们会使用File对象来表示文件，以FileReader来执行实际的读取，并用BufferedReader来让读取更有效率。

读取是以while循环来逐行进行，一直到readLine()的结果为null为止。这是最常见的读取数据方式（几乎非序列化对象都是这样的）：以while循环（实际上应该称为while循环测试）来读取，读到没有东西可以读的时候停止（通过读取结果为null来判断）。

### 带有两行的文本文件

What's  $2 + 27/4$   
What's  $20+22/42$

MyText.txt

```

import java.io.*; 别忘了这个

class ReadAFile {
    public static void main (String[] args) {
        try {
            File myFile = new File("MyText.txt");
            FileReader fileReader = new FileReader(myFile);

            BufferedReader reader = new BufferedReader(fileReader);
            FileReader是字符的连接到文本文件的串流
            将FileReader链接到BufferedReader以获取更高的效率。它只会在缓冲区读空的时候才会回头去磁盘读取
            用String变量来承接所读取的结果
            String line = null;

            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
            reader.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}

```

读一行就列出一行，直到没有东西可以读为止

## Quiz Card Player (程序代码大纲)

```

public class QuizCardPlayer {

    public void go() {
        // 创建并显示gui
    }

    class NextCardListener implements ActionListener {
        public void actionPerformed(ActionEvent ev) {
            // 如果是个问题，显示答案，否则显示下一个问题
            // 改一个标识表明我们已经浏览了问题或答案
        }
    }

    class OpenMenuListener implements ActionListener {
        public void actionPerformed(ActionEvent ev) {
            // 生成一个文件对话框
            // 让用户把一个卡片设置打开
        }
    }

    private void loadFile(File file) {
        // 创建卡片的ArrayList，并从文本文件中读取它们
        // 调用OpenMenuListener事件处理器，每次从文件中读取一行
        // 告诉makeCard()方法创建一个新卡片
        // (one line in the file holds both the question and answer, separated by a "/")
    }

    private void makeCard(String lineToParse) {
        // 调用LoadFile方法，从文本文件中读取一行
        // 创建一个新的QuizCard，通过调用CardList把它加入ArrayList中
    }
}

```

## Quiz Card Player代码

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

public class QuizCardPlayer {

    private JTextArea display;
    private JTextArea answer;
    private ArrayList<QuizCard> cardList;
    private QuizCard currentCard;
    private int currentCardIndex;
    private JFrame frame;
    private JButton nextButton;
    private boolean isShowAnswer;

    public static void main (String[] args) {
        QuizCardPlayer reader = new QuizCardPlayer();
        reader.go();
    }

    public void go() {
        // 创建gui

        frame = new JFrame("Quiz Card Player");
        JPanel mainPanel = new JPanel();
        Font bigFont = new Font("sanserif", Font.BOLD, 24);

        display = new JTextArea(10, 20);
        display.setFont(bigFont);

        display.setLineWrap(true);
        display.setEditable(false);

        JScrollPane qScroller = new JScrollPane(display);
        qScroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
        qScroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
        nextButton = new JButton("Show Question");
        mainPanel.add(qScroller);
        mainPanel.add(nextButton);
        nextButton.addActionListener(new NextCardListener());

        JMenuBar menuBar = new JMenuBar();
        JMenu fileMenu = new JMenu("File");
        JMenuItem loadMenuItem = new JMenuItem("Load card set");
        loadMenuItem.addActionListener(new OpenMenuListener());
        fileMenu.add(loadMenuItem);
        menuBar.add(fileMenu);
        frame.setJMenuBar(menuBar);
        frame.getContentPane().add(BorderLayout.CENTER, mainPanel);
        frame.setSize(640, 500);
        frame.setVisible(true);

    } // 关闭go
}
```

没什么，只是一般的GUI  
程序代码罢了

```

public class NextCardListener implements ActionListener {
    public void actionPerformed(ActionEvent ev) {
        if (isShowAnswer) {
            // 显示答案
            display.setText(currentCard.getAnswer());
            nextButton.setText("Next Card");
            isShowAnswer = false;
        } else {
            // 显示问题
            if (currentCardIndex < cardList.size()) {
                showNextCard();
            } else {
                // 没有更多的卡片了
                display.setText("That was last card");
                nextButton.setEnabled(false);
            }
        }
    }
}

public class OpenMenuListener implements ActionListener {
    public void actionPerformed(ActionEvent ev) {
        JFileChooser fileOpen = new JFileChooser();
        fileOpen.showOpenDialog(frame);
        loadFile(fileOpen.getSelectedFile());
    }
}

private void loadFile(File file) {
    cardList = new ArrayList<QuizCard>();
    try {
        BufferedReader reader = new BufferedReader(new FileReader(file));
        String line = null;
        while ((line = reader.readLine()) != null) {
            makeCard(line);
        }
        reader.close();
    } catch (Exception ex) {
        System.out.println("couldn't read the card file");
        ex.printStackTrace();
    }
    // 显示第一个卡片
}

private void makeCard(String lineToParse) {
    String[] result = lineToParse.split("/");
    QuizCard card = new QuizCard(result[0], result[1]);
    cardList.add(card);
    System.out.println("made a card");
}

private void showNextCard() {
    currentCard = cardList.get(currentCardIndex);
    currentCardIndex++;
    display.setText(currentCard.getQuestion());
    nextButton.setText("Show Answer");
    isShowAnswer = true;
}
} // 关闭类

```

检查 `isShowAnswer` 来判断现在看的是问题还是答案，并根据回答来执行适当的工作

打开文件的对话框让用户选择文件

读取一行数据，传给 `makeCard()` 来把字符串解析成卡片加到 `ArrayList` 中

每一行都是一张卡片，但需要分解成问题和答案，这里使用到下一页会说明的 `split()`

## 用String的split()解析

假设你的flashcard像下面这样：

问题

What is blue + yellow?

答案

green

文件中的问答像是下面这样：

What is blue + yellow?/green  
What is red + blue?/purple

要如何分开问题和答案？

当你读取文件时，问题和答案是合并在同一行，以“/”字符来分开的。

String的split()可以把字符串拆开。

split()可以将字符串拆开成String的数组。

What is blue + yellow?

单元一



green

单元二

从文件读出来的时候会像这样

```
String toTest = "What is blue + yellow?/green";
String[] result = toTest.split("/");
for (String token:result) {
    System.out.println(token);
}
```

split()会用参数所指定的字符来把这个String拆开成两个部分（此方法实际上能够做到比这个例子还要复杂的解析）

把数组中的每个元素逐一地列出来

there are no  
Dumb Questions

**问：**我查看API的时候发现java.io这个包中有500万种类，你到底是怎么知道要用哪一种？

**答：**输入/输出的API使用一种模块化的“链”概念来让你可以把连接串流与链接串流以各种可能应用到的排列组合连起来。

这个串流链并不是只能衔接两种层次，你可以连接多种链接串流来达成所需的处置。

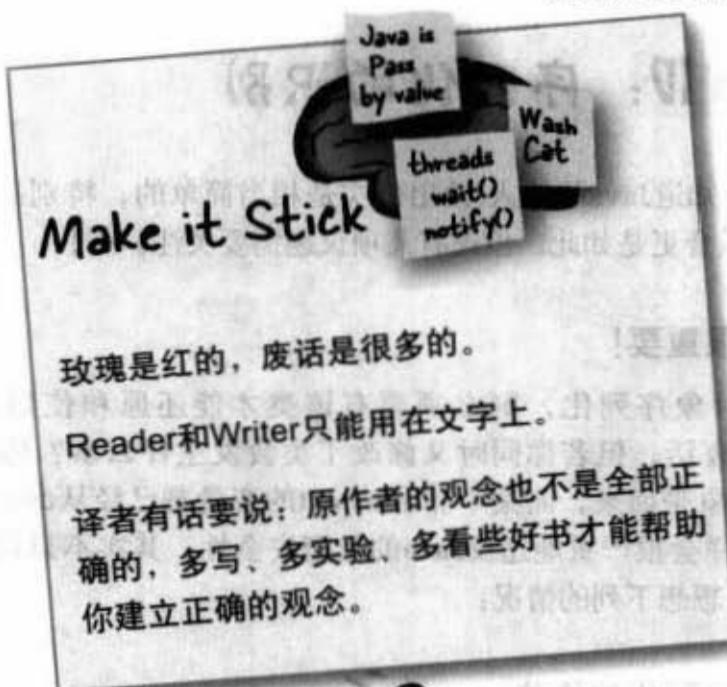
通常你只会用到少数几个类而已。如果想要读写文本文件，或许BufferedReader与BufferedWriter（链接到FileReader与FileWriter上）就够用了。

换言之，本书有涵盖到九成以上你所会用到的Java 输入/输出。

**问：**那1.4版新增的输入/输出nio这个类呢？

**答：**java.nio这个类带来重大的效能提升并可以充分利用执行程序的机器上的原始容量。nio的一项关键能力是你可以直接控制buffer。另一项能力是non-blocking的输入/输出，它能让你的输入/输出程序代码在没有东西可读取或写入时不必等在那里。某些现有的类（包括FileInputStream和FileOutputStream）会利用到其中一部分的功能。nio使用起来更为复杂，除非你真的很需要新功能，不然的话，使用我们这里所讨论的功能方法会简单得多。此外，如果没有很小心的设计，nio可能会引发效能损失。非nio的输入/输出适合九成以上的应用，特别在你还是新手的时候更是如此。

但你还是可以使用FileInputStream并通过getChannel()方法来存取channel来开始使用nio。



玫瑰是红的，废话是很多的。

Reader和Writer只能用在文字上。

译者有话要说：原作者的观念也不是全部正确的，多写、多实验、多看些好书才能帮助你建立正确的观念。

## 要点

- 用FileWriter这个连接串流来写入文本文件。
- 将FileWriter链接到BufferedWriter可以提升效率。
- File对象代表文件的路径而不是文件本身。
- 你可以用File对象来创建、浏览和删除目录。
- 用到String文件名的串流大部分都可以用File对象来代替String。
- 用FileReader来读取文本文件。
- 将FileReader链接到BufferedReader可以提升效率。
- 通常我们会使用特殊的字符来分隔文本数据中的不同元素。
- 使用split()方法可以把String拆开，其中的分隔字符不会被当作数据来看待。

## Version ID: 序列化的识别

现在你已经知道Java的输入/输出确实是相当简单的，特别是很常见的连接/链接组合更是如此。但还有几项议题需要关注。

### 版本控制很重要！

如果你将对象序列化，则必须要有该类才能还原和使用该对象。OK，这是废话。但若你同时又修改了类会发生什么事？假设你尝试要把Dog对象带回来，而某个非transient的变量却已经从double被改成String。这样会很严重地违反Java的类型安全性。其实不只是修改会伤害兼容性，想想下列的情况：

### 会损害解序列化的修改：

删除实例变量。

改变实例变量的类型。

将非瞬时的实例变量改为瞬时的。

改变类的继承层次。

将类从可序列化改成不可序列化。

将实例变量改成静态的。

### 通常不会有事的修改：

加入新的实例变量（还原时会使用默认值）。

在继承层次中加入新的类。

从继承层次中删除类。

不会影响解序列化程序设定变量值的存取层次修改。

将实例变量从瞬时改成非瞬时（会使用默认值）。

### ① 编写 Dog。

```
101101  
101101  
1011000010  
1010 10 0  
01010 1  
1010101  
10101010  
1001010101
```

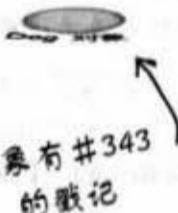
Dog.class

版本 10  
#343

### ② 将Dog对象序列化。



Dog 对象



对象有#343  
的标记

### ③ 修改Dog。

```
101101  
101101  
1011000010  
1010 10 0  
01010 1  
100001 1010  
0 00110101  
1 0 1 10 10
```

Dog.class

版本 10  
#728

### ④ 以修改过的类将Dog对象解序列化。



### ⑤ 还原失败！

俗话说得好：老狗变不出新把戏！

## 使用serialVersionUID

每当对象被序列化的同时，该对象（以及所有在其版图上的对象）都会被“盖”上一个类的版本识别ID。这个ID被称为serialVersionUID，它是根据类的结构信息计算出来的。在对象被解序列化时，如果在对象被序列化之后类有了不同的serialVersionUID，则还原操作会失败！但你还可以有控制权。

如果你认为类有可能会演化，就把版本识别ID放在类中。

当Java尝试要还原对象时，它会比对对象与Java虚拟机上的类的serialVersionUID。例如，如果Dog实例是以23这个ID来序列化的（实际的ID长得多），当Java虚拟机要还原Dog对象时，它会先比对Dog对象和Dog类的serialVersionUID。如果版本不相符，Java虚拟机就会在还原过程中抛出异常。

因此，解决方案就是把serialVersionUID放在class中，让类在演化的过程中还维持相同的ID。

这只会在你有很小心地维护类的变动时才办得到！也就是说你得要对带回旧对象的任何问题负起全责。

若想知道某个类的serialVersionUID，则可以使用Java Development Kit里面所带的serialver工具来查询。

当你的类可能会在产生序列化对象之后继续演进时……

① 使用serialver工具来取得版本ID。

```
File Edit Window Help serialKiller
% serialver Dog
Dog: static final long
serialVersionUID = -
5849794470654667210L;
```

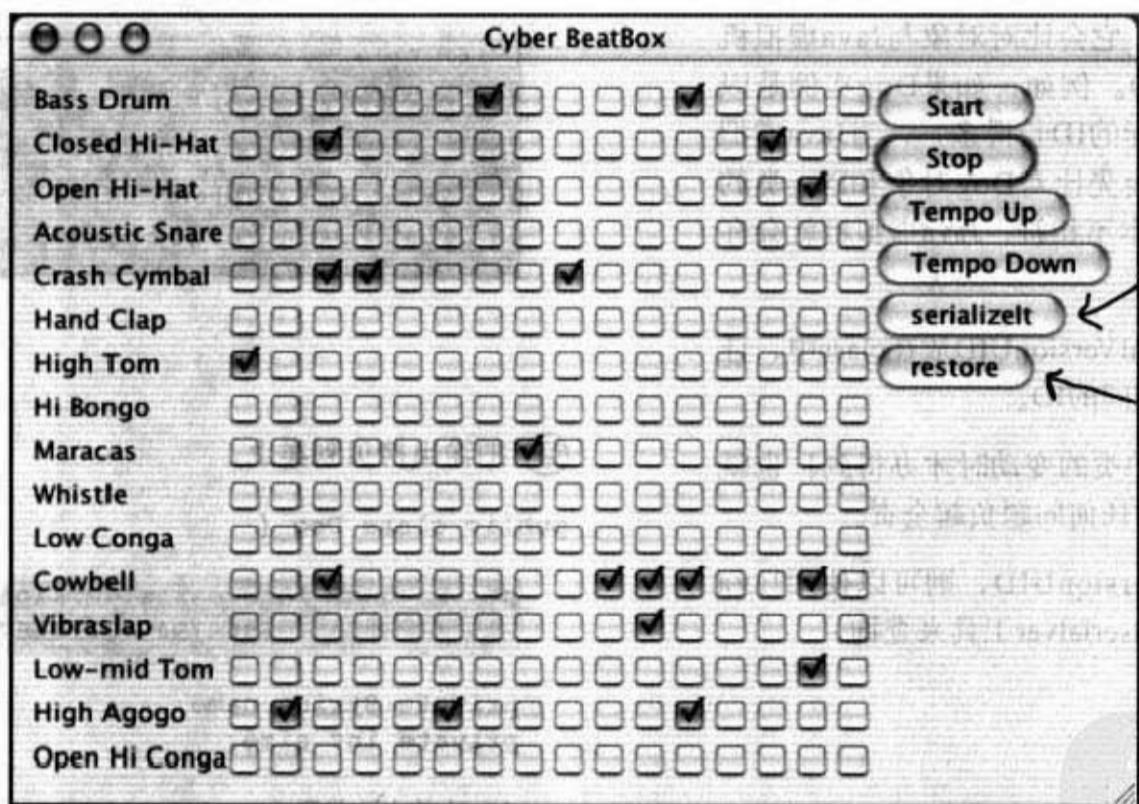
② 把输出拷贝到类上。

```
public class Dog {
    static final long serialVersionUID =
        -6849794470754667710L;
    private String name;
    private int size;
    // 这是方法代码
}
```

③ 在修改类的时候要确定修改程序的后果！  
例如，新的Dog要能够处理旧的Dog解序列化之后新加入变量的默认值。

```
File Edit Window Help serialKiller
% serialver Dog
Dog: static final long
serialVersionUID = -
5849794470654667210L;
```

# 程序料理



让BeatBox能够存储并加载节奏样式

## 存储BeatBox节奏

要记住，在BeatBox中的鼓声样式只不过是一组复选框而已。在播放sequence的同时，程序代码会逐个检查复选框以指出哪个鼓声应该在16拍之中放出。因此存储节奏对我们而言就是存储复选框的状态。

我们可以用简单的boolean数组来存储256个复选框的状态。只要其中的元素都是可被序列化的，数组对象就可以被序列化。因此存储数组是没有问题的。

要载回节奏样式时，只要读取单一的boolean数组对象（将它还原）并存回复选框就可以。你已经看过大部分的程序代码，因此这一章只会展示存储与还原的部分。

这段程序料理让我们可以为进入下一章作准备，下一章将不是写入文件，而是传送到网络上的服务器上，同时读取来源也不是文件，而是其他用户通过服务器来传送的。

这是个BeatBox程序代码中  
的内部类

```
public class MySendListener implements ActionListener {
    public void actionPerformed(ActionEvent a) { ← 这会在用户按下按钮触发
        boolean[] checkboxState = new boolean[256]; ← 此数组用来保存复选框的状态
        for (int i = 0; i < 256; i++) {
            JCheckBox check = (JCheckBox) checkboxList.get(i); ← 逐个取得状态并加到数组中
            if (check.isSelected()) {
                checkboxState[i] = true;
            }
        }
        try {
            FileOutputStream fileStream = new FileOutputStream(new File("Checkbox.ser"));
            ObjectOutputStream os = new ObjectOutputStream(fileStream);
            os.writeObject(checkboxState); ← 将boolean数组序列化
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    } // 关闭方法
} // 关闭内部类
```

## 还原 BeatBox 节奏

这几乎就是存储的反向操作，读取boolean的数组然后使用它来还原GUI上复选框的状态。这都会发生在用户按下restore的按钮之后。

这是复选框程序代码中另外  
一个内部类

```
public class MyReadInListener implements ActionListener {  
  
    public void actionPerformed(ActionEvent a) {  
        boolean[] checkboxState = null;  
        try {  
            FileInputStream fileIn = new FileInputStream(new File("Checkbox.ser"));  
            ObjectInputStream is = new ObjectInputStream(fileIn);  
            checkboxState = (boolean[]) is.readObject(); ← 读取文件中的对象并将读取回来的  
                                              Object类型转换为boolean数组  
        } catch (Exception ex) {ex.printStackTrace();}  
  
        for (int i = 0; i < 256; i++) {  
            JCheckBox check = (JCheckBox) checkboxList.get(i);  
            if (checkboxState[i]) {  
                check.setSelected(true);  
                还原每个checkbox的状态  
            } else {  
                check.setSelected(false);  
            }  
        }  
  
        sequencer.stop();  
        buildTrackAndStart();  
        停止目前播放的节奏并使用复选框状  
        态重新创建序列  
    } // 关闭方法  
} // 关闭内部类
```

### Sharpen your pencil

这是个有很大限制的版本。当你按下serializeIt按钮时，它会自动地将节奏序列化到Checkbox.ser这个文件中盖掉旧的。

通过引入JFileChooser就能够加强存储与回复的功能，让你能够为文件命名，也能自由地选择所要的文件。



## 谁能被存储？

下列哪种对象你认为应该要是可序列化的？如果不是，又为何？没有意义吗？安全风险吗？只对目前执行中的Java虚拟机有用？猜猜看，先不要偷看API文件。

### 类型 可序列化吗 如果不行，是什么原因？

Object	Yes / No	
String	Yes / No	
File	Yes / No	
Date	Yes / No	
OutputStream	Yes / No	
JFrame	Yes / No	
Integer	Yes / No	
System	Yes / No	

## 找碴！？

圈选出右边可以通过编译的程序片段。



```
FileReader fileReader = new FileReader();
BufferedReader reader = new BufferedReader(fileReader);
```

```
FileOutputStream f = new FileOutputStream(new File("Foo.ser"));
ObjectOutputStream os = new ObjectOutputStream(f);
```

```
BufferedReader reader = new BufferedReader(new FileReader(file));
String line = null;
while ((line = reader.readLine()) != null) {
    makeCard(line);
}
```

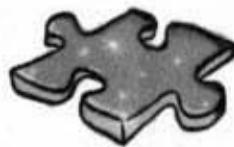
```
ObjectInputStream is = new ObjectInputStream(new FileInputStream("Game.ser"));
GameCharacter oneAgain = (GameCharacter) is.readObject();
```



这一章探索了美好的Java I/O世界。你的任务是判断下列关于I/O的陈述是否为正确的叙述。

## 是非题

- (1) 序列化适用于存储数据给非Java程序使用的情境。
- (2) 对象的状态只能用序列化来存储。
- (3) ObjectOutputStream是个用来存储序列化对象的类。
- (4) 链接串流可以单独使用或与链接串流并用。
- (5) 调用writeObject()可能会存储好几个对象。
- (6) 所有的类都是默认为可序列化的。
- (7) transient这个修饰符让你将实例变量标记为可序列化的。
- (8) 如果父类是不可序列化的，则子类就不能设定成可序列化。
- (9) 当对象被序列化时，它的读取顺序必须是相反的。
- (10) 当对象还原时，构造函数不会执行。
- (11) 序列化与使用文本文件保存的操作都可能会抛出异常。
- (12) BufferedWriter可以链接到FileWriter上。
- (13) File对象代表文件而不是目录。
- (14) 你无法强制缓冲区写出数据。
- (15) 文件的读写串流机制可以用到缓冲区机制。
- (16) String的split()会将分隔字符解析成结果的一部分。
- (17) 对类的任何修改都会破坏之前的序列化对象。



## 排排看

下面是被打乱的Java程序片段。你是否能够将它们重新排列以成为可以编译与执行并产生如同下方的输出结果？不一定每个片段都会用到或只能用一次。

```

class DungeonGame implements Serializable {
    public int x = 3;
    transient long y = 4;
    private short z = 5;
    try {
        FileOutputStream fos = new
            FileOutputStream( dg.ser );
        ObjectOutputStream oos = new
            ObjectOutputStream( fos );
        e.printStackTrace();
        oos.writeObject( d );
    } catch (Exception e) {
        e.printStackTrace();
    }
    short getZ() {
        return z;
    }
    int getX() {
        return x;
    }
    System.out.println( d.getX() + d.getY() + d.getZ() );
    long getY() {
        return y;
    }
    class DungeonTest {
        import java.io.*;
        ObjectInputStream ois = new
            ObjectInputStream( fis );
        fos.writeObject( d );
        d = (DungeonGame) ois.readObject();
    }
}

```

```

File Edit Window Help Torture
% java DungeonTest
12
8

```

```

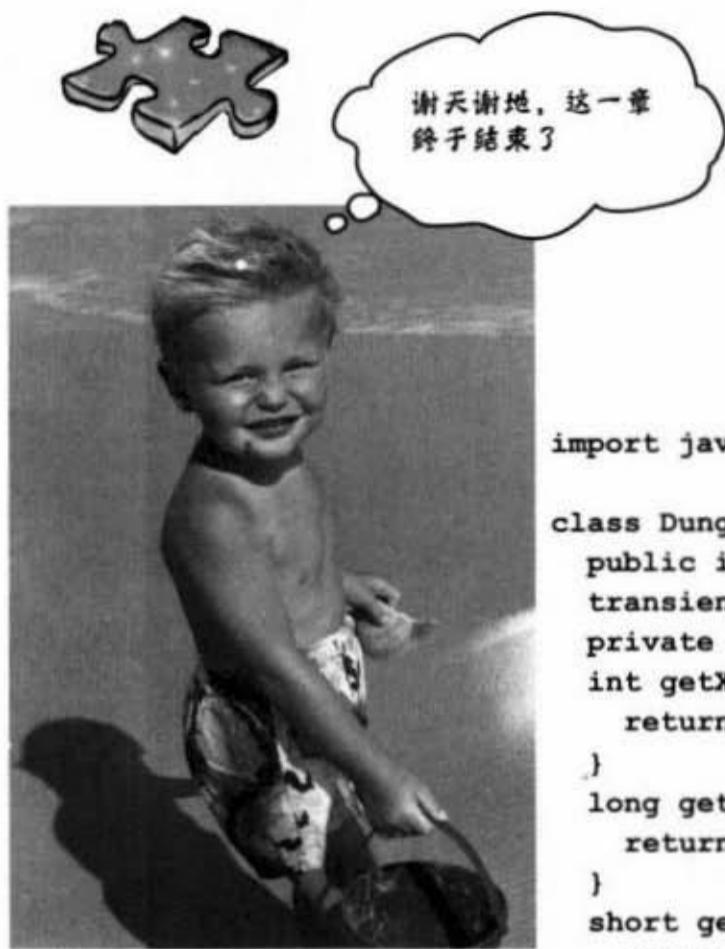
ObjectOutputStream oos = new
    ObjectOutputStream( fos );
    oos.writeObject( d );
    public static void main(String [] args) {
        DungeonGame d = new DungeonGame();
    }
}

```



## 解答

- |                                      |              |
|--------------------------------------|--------------|
| (1) 序列化适用于存储数据给非 Java 程序使用的情境。       | <b>False</b> |
| (2) 对象的状态只能用序列化来存储。                  | <b>False</b> |
| (3) ObjectOutputStream是个用来存储序列化对象的类。 | <b>True</b>  |
| (4) 链接串流可以单独使用或与链接串流并用。              | <b>False</b> |
| (5) 调用writeObject()可能会存储好几个对象。       | <b>True</b>  |
| (6) 所有的类都是默认为可序列化的。                  | <b>False</b> |
| (7) transient这个修饰符让你将实例变量标记为可序列化的。   | <b>False</b> |
| (8) 如果父类是不可序列化的，则子类就不能设定成可序列化。       | <b>False</b> |
| (9) 当对象被序列化时，它的读取顺序必须是相反的。           | <b>False</b> |
| (10) 当对象还原时，构造函数不会执行。                | <b>True</b>  |
| (11) 序列化与使用文本文件保存的操作都可能会抛出异常。        | <b>True</b>  |
| (12) BufferedWriter可以链接到FileWriter上。 | <b>True</b>  |
| (13) File对象代表文件而不是目录。                | <b>False</b> |
| (14) 你无法强制缓冲区写出数据。                   | <b>False</b> |
| (15) 文件的读写串流机制可以用到缓冲区机制。             | <b>True</b>  |
| (16) String的split()会将分隔字符解析成结果的一部分。  | <b>False</b> |
| (17) 对类的任何修改都会破坏之前的序列化对象。            | <b>False</b> |



```

import java.io.*;

class DungeonGame implements Serializable {
    public int x = 3;
    transient long y = 4;
    private short z = 5;
    int getX() {
        return x;
    }
    long getY() {
        return y;
    }
    short getZ() {
        return z;
    }
}

class DungeonTest {
    public static void main(String [] args) {
        DungeonGame d = new DungeonGame();
        System.out.println(d.getX() + d.getY() + d.getZ());
        try {
            FileOutputStream fos = new FileOutputStream("dg.ser");
            ObjectOutputStream oos = new ObjectOutputStream(fos);
            oos.writeObject(d);
            oos.close();
            FileInputStream fis = new FileInputStream("dg.ser");
            ObjectInputStream ois = new ObjectInputStream(fis);
            d = (DungeonGame) ois.readObject();
            ois.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println(d.getX() + d.getY() + d.getZ());
    }
}

```

```

File Edit Window Help Escape
$ java DungeonTest
12
8

```

### 网络联机



**连接到外面的世界。**你的Java程序可以向外扩展并触及其他计算机上的程序。这很容易，所有网络运作的低层细节都已经由java.net函数库处理掉了。Java的一项好处是传送与接收网络上的数据只不过是链接上使用不同链接串流的输入/输出而已。如果有 BufferedReader就可以读取。若数据来自文件或网络另一端，则BufferedReader不用花费很多精力去照顾。在这一章中，我们会使用socket来连接外面的世界。我们会创建客户端的socket、服务器端的socket，并且会让两端相互交谈。在完成这一章的进度之前，你会创建出功能完整、多线程的聊天程序客户端。咦？我刚刚说multithreaded吗？好吧，你还会学习到如何一边开车、一边吃盒饭、一边看报纸、一边聊天、一边修指甲……

java学习群：72030155，每天20:30-23:00都有大神视频教学，想学习的同学可以进群免费听课！

## 实时BeatBox聊天程序



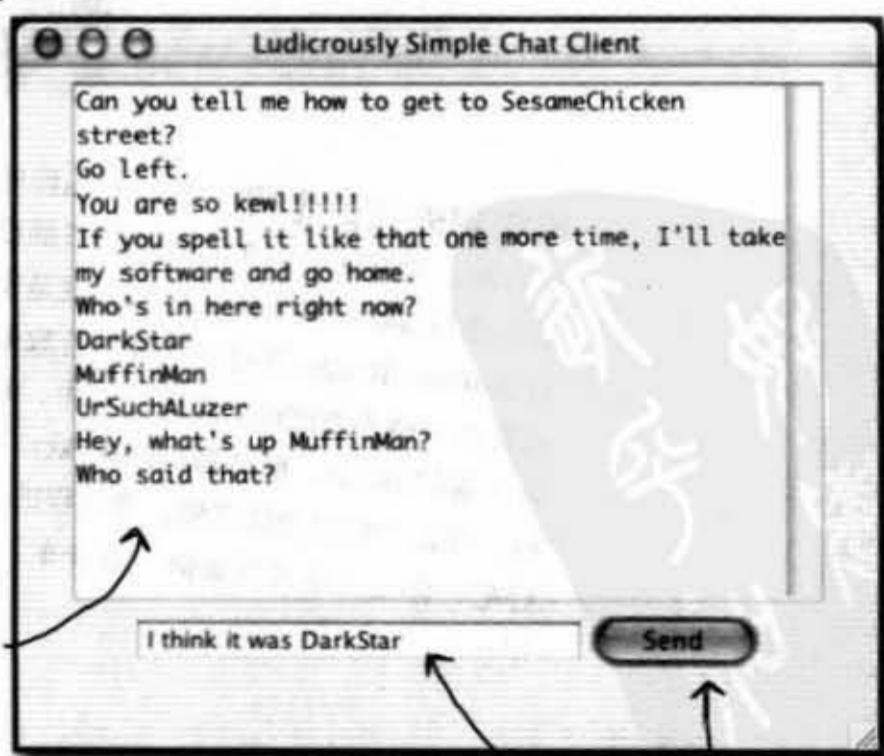
输入信息并按下  
sendit按钮就会  
把信息和节奏样  
式送出

点选接收到的信  
息就会加载它的样  
式

你正在设计计算机游戏，同事们正在进行每个关卡的音效设计。使用有“聊天”功能版的BeatBox，团队成员就可以协同工作——你可以在讨论的同时直接送出节奏样式，让上线讨论的每个人都可以收到。因此你不能读取他人的意见，还可以借着点选讨论信息区域来加载和播放节奏样式。

在这一章我们会学习到怎样作出这样的聊天程序。我们还会讨论到聊天服务器的制作。完整的BeatBox聊天程序会收录在“程序料理”一节，但前面的内容部分会来开发出“傻瓜版”客户端聊天程序与“限量发行流行时尚品味生活高级电动多功能后现代主义极简版”的聊天服务器来传送与接收文字信息。

此处通常会进行充满智慧和善意的莫名其妙对话，每个人都会看到！

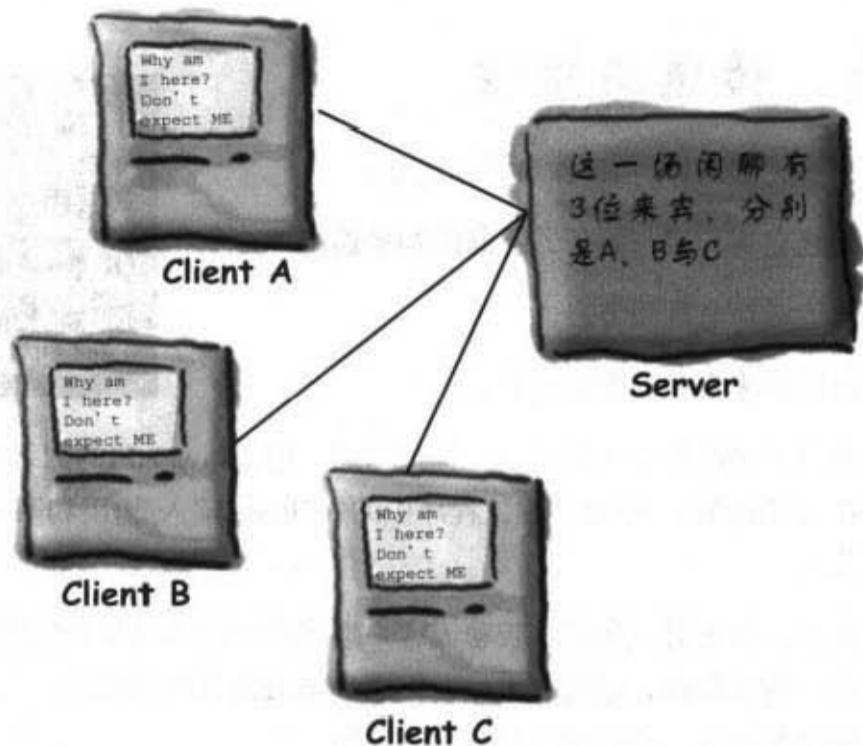


将你的信息送到服务  
器上

## 聊天程序概述

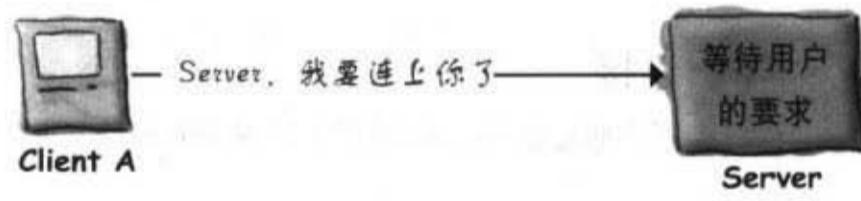
客户端必须要认识服务器。

服务器必须要认识所有的客户端。

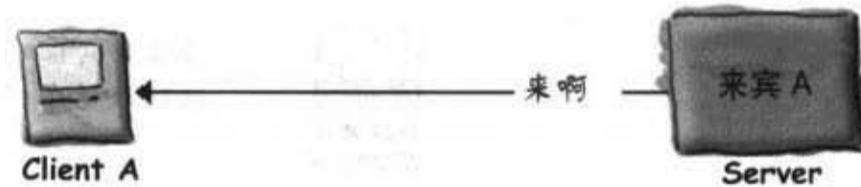


## 工作方式：

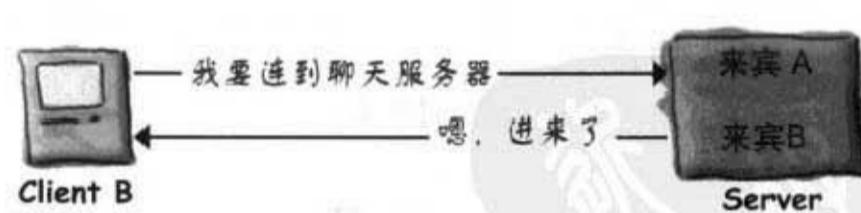
① 客户端连接到服务器。



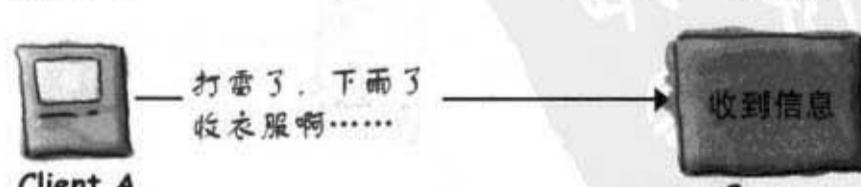
② 服务器建立连接并把客户端加到来宾清单中。



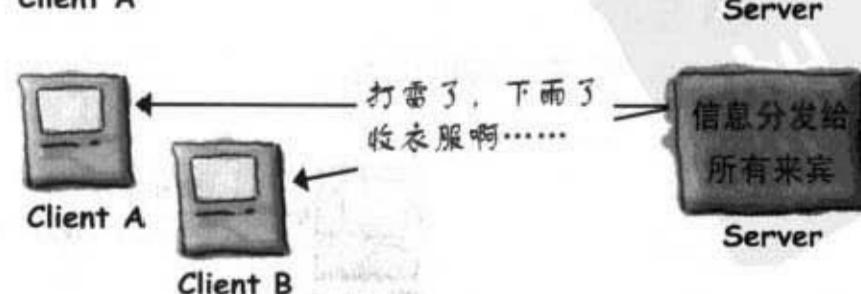
③ 另外一个用户连接上来。



④ 用户 A 送出信息到聊天服务器上。



⑤ 服务器将信息送给所有的来宾。



## 连接、传送与接收

要让客户端能够工作，有3件事必须先学：

- (1) 如何建立客户端与服务器之间的初始连接
- (2) 如何传送信息到服务器。
- (3) 如何接收来自服务器的信息。

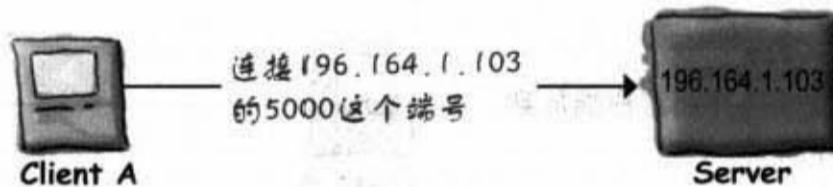
\*译者个人意见：networking的学问绝对不是三言两语说得完，你也许可以很轻松地写出有网络功能的程序，但没有打好底子（像是深入研究TCP/IP等协议的原理）就很容易出问题（例如说效率不好）同时你也不会有除错抓问题的能力

这里面有非常多的低层工作细节。但很幸运，因为Java API的网络功能包（java.net）能让程序员轻松解决这些问题\*。程序中的GUI码比输入/输出和网络码多了不少的原因就在此。

不只是这样，有个潜藏在简单版聊天客户端程序中的问题是：同时做两件事。建立联机是单次的工作。但之后聊天来宾得同时处理传送和接收信息，这得好好想个办法，我们稍后就会加以说明。

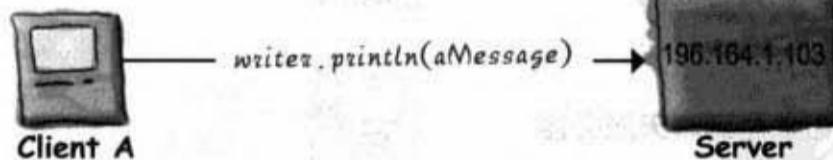
### ① 连接

用户通过建立socket连接来连接服务器。



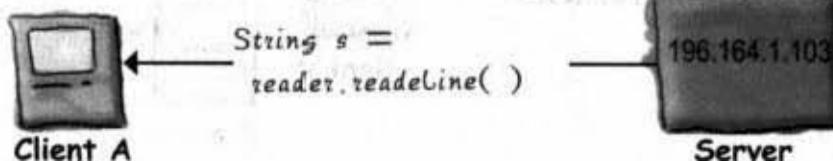
### ② 传送

用户送出信息给服务器。



### ③ 接收

用户从服务器接收信息。



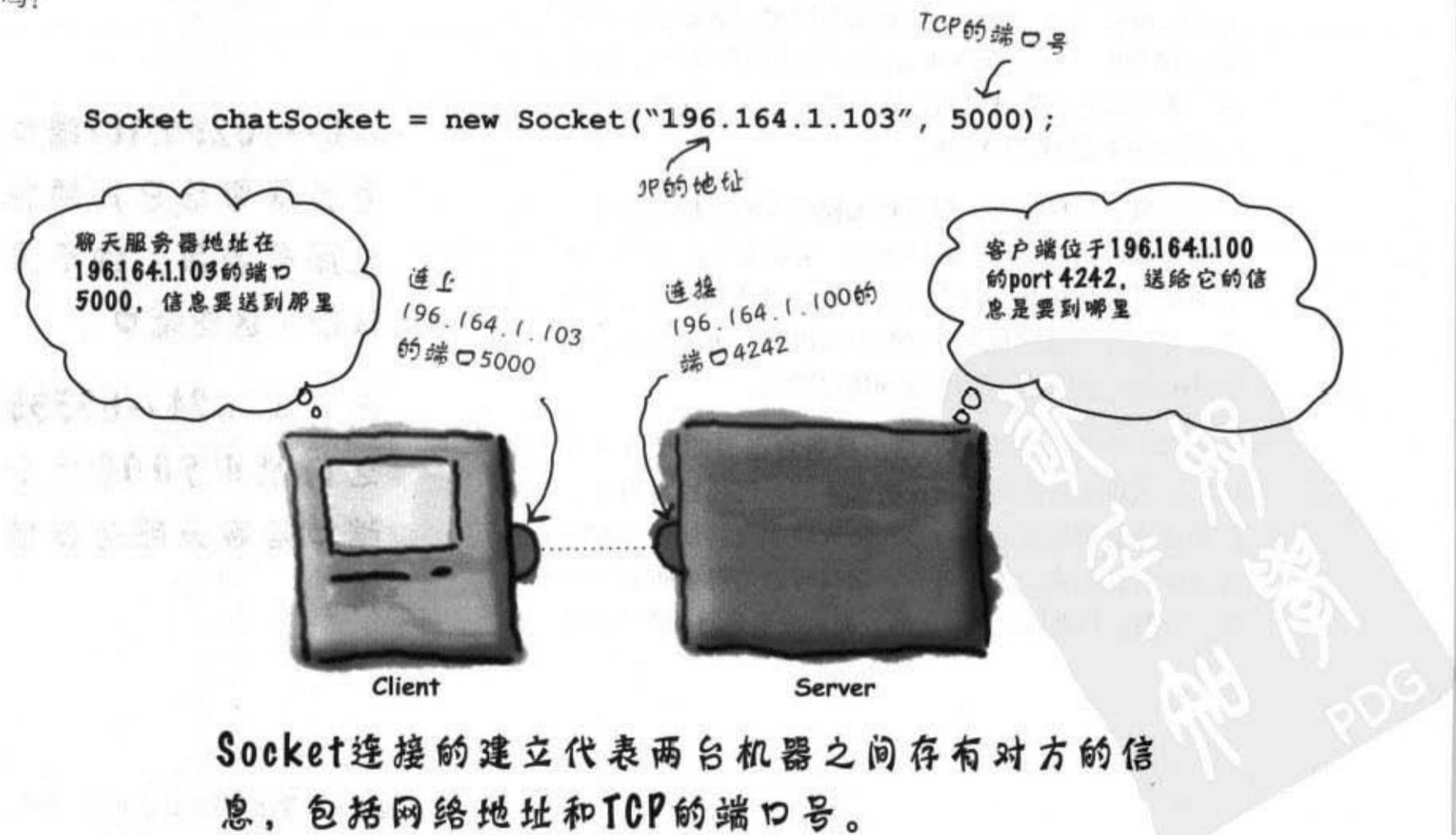
## 建立Socket连接

要连接到其他的机器上，我们会需要Socket的连接。Socket是个代表两台机器之间网络连接的对象（java.net.Socket）。什么是连接？两台机器之间的一种关系，让两个软件相互认识对方。最重要的是两者知道如何与对方通信，也就是说知道如何发送数据给对方。

还好我们不在乎低层的细节，因为这是在低层的“网络设备”中处理的。如果你不知道什么是网络设备也没关系，它只是一种让运行在Java虚拟机上的程序能够找到方法去通过实际的硬件（比如说网卡）在机器之间传送数据的机制。反正有人会负责这些低层的工作就对了。这些人是由操作系统的特定部分与Java的网络API所组成的。你所必须要处理的是高层的部分——真的很高级，且又简单到很吓人。准备好面对这一切了吗？

要创建Socket连接你得知道两项关于服务器的信息：它在哪里以及用哪个端口来收发数据。

也就是说IP地址与端口号。



## TCP端口只是一个16位宽、用 来识别服务器上特定程序的 数字

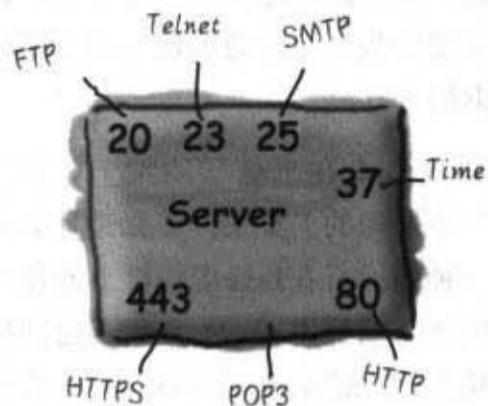
网页服务器（HTTP）的端口号是80，这是规定的标准。如果你有Telnet服务器的话，它的端口号会是23，POP3邮件服务器的是110，SMTP邮局交换服务器是25。把这些数字想成识别的代号（就像华安的终身代号是9527一样）。它们代表在服务器上执行软件的逻辑识别。注意，你在机器的背后找不到这些端口插孔。每个服务器上都有65536个端口（0~65535），很明显，哪来那么多实际的端口可以用。它只是个逻辑上用来表示应用程序的数字。

如果没有端口号，服务器就无法分辨客户端是要连到哪个应用程序的服务。每个应用程序可能都有独特的工作交谈方法，如果没有识别就发送的话会制造很大的混乱。就像是对邮件服务器发送HTTP请求。

在编写服务器程序时，你会加入程序代码来指定想要使用哪个端口号（稍后会有实际例子）。对聊天程序来说，我们选择的端口号是5000。没有特定理由，只是我们很喜欢这个数字，并且这个数字也介于1024~65535之间。为什么？因为0~1023都已经被保留给已知的特定服务。

如果你打算要在公司的网络上执行自己写的服务（服务程序），就得要先跟网管讨论有哪些端口已经被占用了。有时候网管也会把特定的端口号用防火墙或者其他特定的安全控管机制封锁起来。不管怎样，端口号总是得要挑选一个可用的。当然，如果你是在家上网，那就得要先问过你的小孩。

常见的TCP端口号。



一个地址可以有65536个不同的端口号可用。

从0~1023的TCP端口号是保留给已知的特定服务使用，你不应该使用这些端口。

我们从1024~65535之间挑出5000这个端口给聊天服务器使用。

\*也有可能你比网管大哥更厉害，此时爱用哪个端口当然是你说了算。

there are no  
Dumb Questions

**问：**你怎么知道要跟哪个端口沟通？

**答：**这要看程序是否为已知的服务。如果你尝试要连接众所周知的服务，像是HTTP、FTP、SMTP等，你可以在网络上用“已知tcp端口”来搜索引擎。或者用嘴问，别人也会叫你自己去查……

但如果程序不是这一类有共同标准的，你就得问谁开发的此服务，或者查询相关文件。这通常要问原来写程序或者是安装设定的人。例如你想要写个MSN Messenger的程序，可以上MSDN网站查相关资料。

**问：**不同程序可以共享一个端口吗？

**答：**不行，如果你想要使用（技术上叫做绑定）某个已经被占用的端口，就会收到BindException。绑定一个端口就代表程序要在特定的端口上面执行。

后面还有关于服务器的讨论会有更多相关的细节信息。

IP地址就好像是门牌号码



端口号就是该地址不同的窗口



IP地址可以指定特定的地方，  
像是“展览路1号”

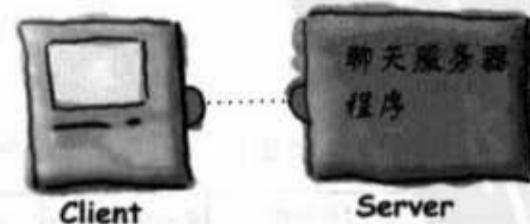
25号窗口处理邮件，80  
号窗口处理网页



### Brain Barbell

现在你已经能够建立Socket连接。客户端与服务器都已经知道对方的IP地址和端口号。接下来呢？要如何通信？如何传送数据？

要怎样交谈？

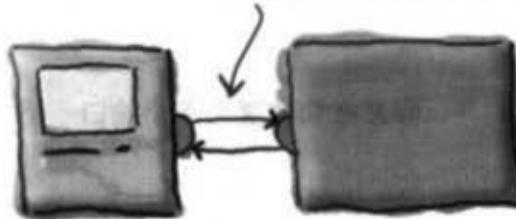


读取socket

## 使用 BufferedReader 从 Socket 上读取数据

用串流来通过Socket连接来沟通。它是跟上一章所用相同的一般串流。Java的好处就在于大部分的输入/输出工作并不在乎链接串流的上游实际上是什么。也就是说你可以使用BufferedReader而不管是串流来自文件或Socket。

在Socket上的输出入串流



### 1 建立对服务器的Socket连接

```
Socket chatSocket = new Socket("127.0.0.1", 5000);
```

用我们所说的5000号端口

127.0.0.1这个IP地址有特殊意义，就  
是“本机”，所以可以在自己这台计  
算机上同时测试客户端和服务端

### 2 建立连接到Socket上低层输入串流的InputStreamReader

```
InputStreamReader stream = new InputStreamReader(chatSocket.getInputStream());
```

这是个低层和高层串流间的桥梁

从Socket取得输入串流

### 3 建立BufferedReader来读取

```
BufferedReader reader = new BufferedReader(stream);  
String message = reader.readLine();
```

将BufferedReader链接到  
InputStreamReader



目的地  
缓冲区的字符  
BufferedReader

转换成字符  
InputStreamReader

来自服务器的字节  
Socket输入流



来源

## 用PrintWriter写数据到Socket上

上一章没有使用到PrintWriter，而是用BufferedWriter。现在有了别的选择，但因为每次都是写入一个String，所以PrintWriter是最标准的做法。并且PrintWriter中有print()和println()方法就跟System.out里面的两个刚好一样！

### 1 对服务器建立Socket连接

```
Socket chatSocket = new Socket("127.0.0.1", 5000);
```

这部分跟上一页相同

### 2 建立链接到Socket的PrintWriter

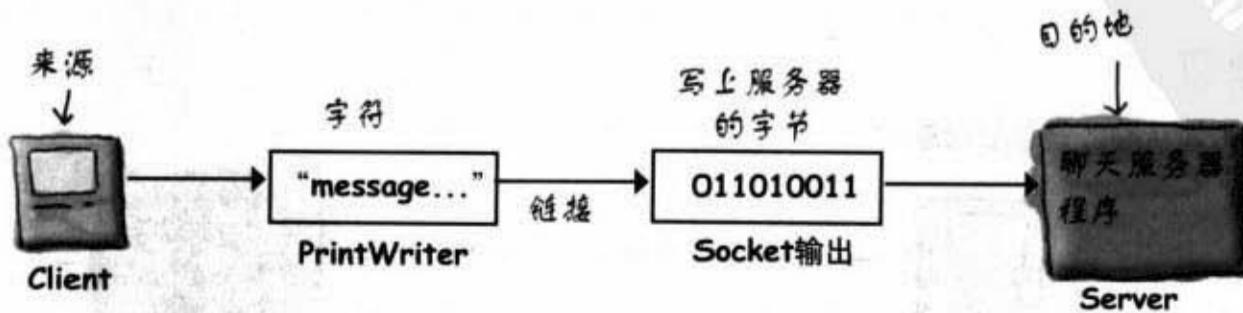
```
PrintWriter writer = new PrintWriter(chatSocket.getOutputStream());
```

↑  
字符数据和字节间的转换  
桥梁，可以衔接String和  
Socket两端

↑  
传给这个constructor的参数是来  
自于Socket

### 3 写入数据

```
writer.println("message to send"); ← println()会在送出的数据后面加上换行  
writer.print("another message"); ← print()不会加上换行
```



## 每日嘉言

在我们开始创建聊天程序之前，先做一个小程序。“叶教授”是一位充满人生智慧哲理的大师，它能在你单调孤寂的程序设计生活中指引、启发你脆弱又容易受伤的心灵。

我们会创建一个称为“The Advice Guy”的程序，它能在启动的时候从服务器上提取叶教授的嘉言精华来滋润你干枯的胡渣。

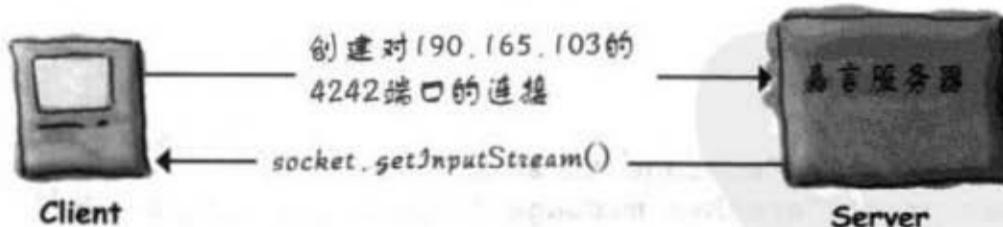
这么好的事情你还等什么？心动不如马上行动，如在通话中请稍后再拨。



叶教授

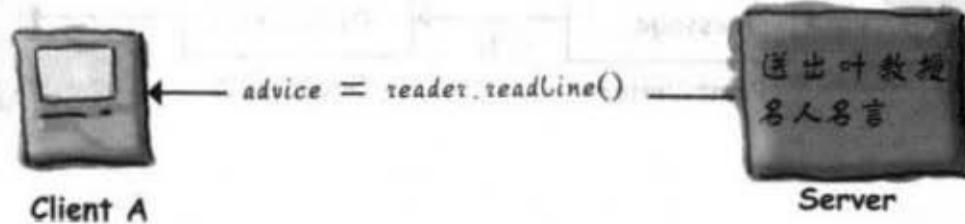
### ① 连接

客户端连上服务器并取得输入串流。



### ② 读取

客户端从服务器读取信息。



## DailyAdviceClient 客户端程序

这个程序会建立Socket，通过其他串流来制作BufferedReader，并从服务器应用程序（用4242端口服务的任何程序）上读取一行信息。

```

import java.io.*;
import java.net.*;

public class DailyAdviceClient {

    public void go() {
        try { ← 有可能出状况
            Socket s = new Socket("127.0.0.1", 4242); ← 对端口4242作连接

            InputStreamReader streamReader = new InputStreamReader(s.getInputStream());
            BufferedReader reader = new BufferedReader(streamReader); ← 链接数据串流

            String advice = reader.readLine(); ← 这个readLine()就跟读取文件
            System.out.println("Today you should: " + advice); ← 来源是一样的

            reader.close(); ← 会关闭所有的串流

        } catch(IOException ex) {
            ex.printStackTrace();
        }
    }

    public static void main(String[] args) {
        DailyAdviceClient client = new DailyAdviceClient();
        client.go();
    }
}

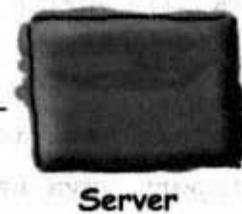
```



## Sharpen your pencil

测试一下你对Socket读写所需的串流的记忆力。不要偷看上一页。

从Socket读取文字：

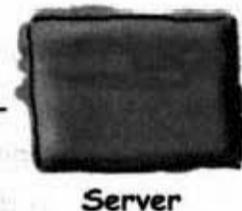


写下/画出串流的链接

Client

来源

传送文字到Socket：



写下/画出串流的链接

Client

目的地

## Sharpen your pencil

问答题

- (1) 建立Socket连接时需要知道服务器的哪两项信息？
- (2) HTTP与FTP等众所周知的服务会用到哪个端口？
- (3) 有效的TCP端口共有多少个？

## 编写简单的服务器程序

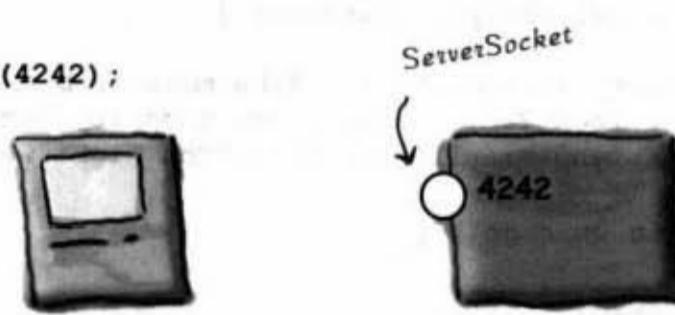
编写服务器应用程序要用到哪些东西呢？一对Socket。没错，两个称为一对。它们是一个会等待用户请求（当用户创建Socket时）的ServerSocket以及与用户通信用的Socket。

### 工作方式

- ① 服务器应用程序对特定端口创建出ServerSocket。

```
ServerSocket serverSock = new ServerSocket(4242);
```

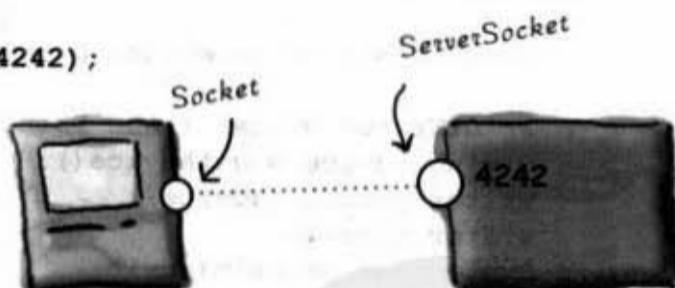
这会让服务器应用程序开始监听来自4242端口的客户端请求。



- ② 客户端对服务器应用程序建立Socket连接。

```
Socket sock = new Socket("190.165.1.103", 4242);
```

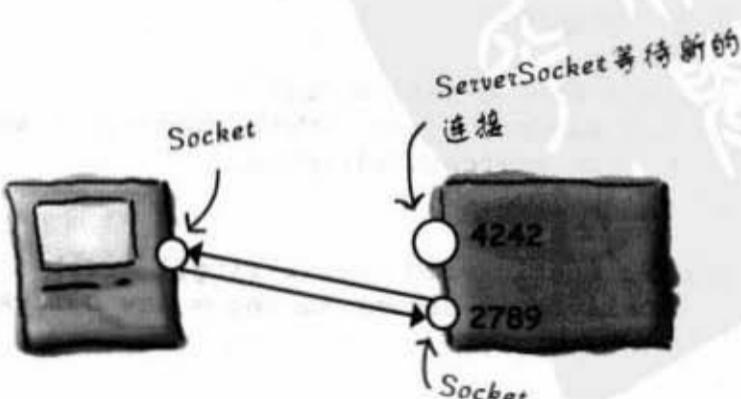
客户端得知道IP地址与端口号。



- ③ 服务器创建出与客户端通信的新Socket。

```
Socket sock = serverSock.accept();
```

accept()方法会在等待用户的Socket连接时闲置着。当用户连上来时，此方法会返回一个Socket（在不同的端口上）以便与客户端通信。Socket与ServerSocket的端口不相同，因此ServerSocket可以空出来等待其他的用户。



## DailyAdviceServer 程序代码

这个程序会创建ServerSocket并等待客户端的请求。当它收到客户端请求时，服务器会建立与客户端的Socket连接。服务器接着会建立PrintWriter来送出信息给客户端。

```

import java.io.*; 记得import
import java.net.*;

public class DailyAdviceServer { 嘉言来自这个数组
    String[] adviceList = {"Take smaller bites", "Go for the tight jeans. No they do NOT
make you look fat.", "One word: inappropriate", "Just for today, be honest. Tell your
boss what you *really* think", "You might want to rethink that haircut."};

    public void go() {
        try {
            ServerSocket serverSock = new ServerSocket(4242); ServerSocket会监听客户端对这台
机器在4242端口上的要求
            服务器进入无穷循环等待服
务客户端的请求
            while(true) {
                Socket sock = serverSock.accept(); 这个方法会停下来等待要求到达
之后才会继续
                PrintWriter writer = new PrintWriter(sock.getOutputStream());
                String advice = getAdvice();
                writer.println(advice);
                writer.close(); 使用Socket连接来送出嘉言信息，送
出后就可以把连接关闭
                System.out.println(advice);
            }
        } catch(IOException ex) {
            ex.printStackTrace();
        }
    } // 关闭go函数

    private String getAdvice() {
        int random = (int) (Math.random() * adviceList.length);
        return adviceList[random];
    }

    public static void main(String[] args) {
        DailyAdviceServer server = new DailyAdviceServer();
        server.go();
    }
}

```



服务器如何知道怎样与客户端沟通？

客户端知道服务器的IP地址和端口号，但服务器是如何知道建立和客户端沟通的Socket连接的信息呢？

回想一下服务器是在何时何处如何取得客户端的信息。

*there are no  
Dumb Questions*

**问：**上一页的嘉言服务器有个很严格的限制——看起来它每次只能服务一个用户！

**答：**没错！在没有完成目前用户的响应程序循环之前它无法回到循环的开始处来处理下一个要求（无法进入accept()的等待来建立Socket给新的用户）。

**问：**换个方式再问一次，如何才能让服务器能够同时处理多个用户？目前使用的方式根本应付不了聊天的需求。

**答：**这真的很简单，使用不同的线程并让新的客户端取得新的线程就好。我们正要开始讨论这个部分。

## 要点

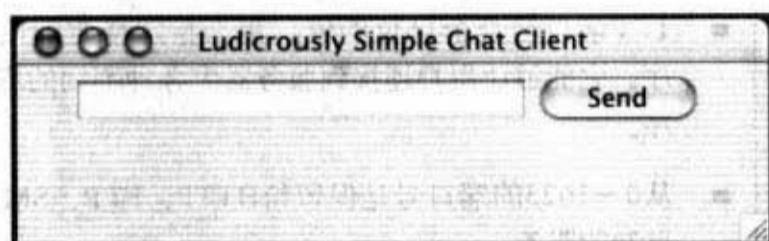
- 客户端与服务器的应用程序通过Socket连接来沟通。
  - Socket代表两个应用程序之间的连接，它们可能会是在不同的机器上执行的。
  - 客户端必须知道服务器应用程序的IP地址（或网域名称）和端口号。
  - TCP端口号是个16位的值，用来指定特定的应用程序。它能够让用户连接到服务器上各种不同的应用程序。
  - 从0~1023的端口号是保留给HTTP、FTP、SMTP等已知的服务。
  - 客户端通过建立Socket来连接服务器。
- ```
Socket s = new Socket("127.0.0.1", 4200);
```
- 一旦建立了连接，客户端可以从socket取得低层串流。
  - `sock.getInputStream();`
  - 建立BufferedReader链接InputStreamReader与来自Socket的输入串流以读取服务器的文本数据。
  - InputStreamReader是个转换字节成字符的桥梁。它主要是用来链接BufferedReader与低层的Socket输入串流。
  - 建立直接链接Socket输出串流的PrintWriter请求print()方法或println()方法来送出String给服务器。
  - 服务器可以使用ServerSocket来等待用户对特定端口的请求。
  - 当ServerSocket接到请求时，它会做一个Socket连接来接受客户端的请求。

## 编写聊天客户端程序

我们会用两阶段来编写聊天客户端应用程序。首先是只能发送信息给服务器，但不会收到其他来宾信息的版本（让聊天室整个概念废掉）。

然后我们会发展出可接收和发送信息的完整版本。

### 第一版：只能发送的版本



输入信息然后按下Send来传递给服务器。这个版本不会接收来自服务器的信息，所以也没有可滚动的文本区域。

### 大致代码

```
public class SimpleChatClientA {  
    JTextField outgoing;  
    PrintWriter writer;  
    Socket sock;  
  
    public void go() {  
        // 注册按钮的监听者  
        // 调用setUpNetworking()  
    }  
  
    private void setUpNetworking() {  
        // 建立Socket、PrintWriter  
        // 赋值PrintWriter给实例变量  
    }  
  
    public class SendButtonListener implements ActionListener {  
        public void actionPerformed(ActionEvent ev) {  
            // 取得文字字段内容  
            // 传送到服务器上  
        }  
        // 关闭SendButton Listener内部类  
    } // 关闭外部类
```

```

import java.io.*;
import java.net.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class SimpleChatClientA {
    JTextField outgoing;
    PrintWriter writer;
    Socket sock;

    public void go() {
        JFrame frame = new JFrame("Ludicrously Simple Chat Client");
        JPanel mainPanel = new JPanel();
        outgoing = new JTextField(20);
        JButton sendButton = new JButton("Send");
        sendButton.addActionListener(new SendButtonListener());
        mainPanel.add(outgoing);
        mainPanel.add(sendButton);
        frame.getContentPane().add(BorderLayout.CENTER, mainPanel);
        setUpNetworking();
        frame.setSize(400, 500);
        frame.setVisible(true);
    } // 关闭go
}

private void setUpNetworking() {
    try {
        sock = new Socket("127.0.0.1", 5000);
        writer = new PrintWriter(sock.getOutputStream());
        System.out.println("networking established");
    } catch (IOException ex) {
        ex.printStackTrace();
    }
} // 关闭setUpNetworking

public class SendButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent ev) {
        try {
            writer.println(outgoing.getText()); // 可以开始写数据，所以println()会把信息
            writer.flush(); // 送到服务器
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        outgoing.setText("");
        outgoing.requestFocus();
    }
} // 关闭SendButtonListener内部类

public static void main(String[] args) {
    new SimpleChatClientA().go();
}
} // 关闭外部类

```

如果你现在要试试看的话，把本章后面列出来的程序打出来执行。先执行服务器然后再另外启动客户端。

## 第二版：可发送和接收



服务器收到信息后转发给所有的来宾，在发给每个人之前它不会显示在收信区

### 重要问题：如何从服务器取得信息？

应该很容易：当你设定网络的同时也把输入串流（或许是BufferedReader）建立好，然后使用readLine()读取信息。

### 重要问题：何时会从服务器取得信息？

想想看，有哪些选择？

#### ① 选择一：每隔20秒查询服务器一次。

优点：可行。

缺点：服务器要知道哪些是你已经看过的？哪些是你还没有看过的？这样一来服务器就得存储信息了。为什么是20秒？这样的延迟实在不好用，越是缩小延迟时间，又显得不太有效率。

#### ② 选择二：每当用户送出信息时就顺便从服务器读回信息。

优点：可行，简单。

缺点：蠢不可及。为什么要等到那个时候才检查信息。如果用户很懒都不传送信息怎么办？

#### ③ 选择三：当信息被送到服务器上的时候就把它读回来。

优点：高效率、最好用。

缺点：程序怎么写？这样得有个循环来等待服务器的信息。但要放在哪里？GUI启动之后，除非有事件被触发，否则不会有任何一处是在运转的。



### 我们选择的是第三种方法

我们需要同时执行的能力，检查服务器信息的同时不会打断用户与GUI的交互！因此用户可以输入信息或滚动接收画面，还需要有东西在背景持续地读取服务器的数据。

这意味着我们需要新的线程（thread），一个独立的执行空间（stack）。

我们想要让只能发送的版本（第一版）可以维持原来的执行，同时有新的进程（process）并行地读取服务器的信息并将它显示在文本区域上。

除非你的计算机有多个处理器，否则Java上新的线程其实不会是运行在操作系统上独立的进程。但感觉上会像是那样。

在Java中你可以边咬口香糖边吹口哨

### Java的multithreading

Java在语言中就有内置多线程的功能。建立新的线程来执行是很简单的：

```
Thread t = new Thread();
t.start();
```

就是这么的简单。建立出新的线程对象就会启动新的线程，它是个独立的调用执行空间。

但是会有一个问题。

该线程并没有执行任何程序。因此它几乎是一出生就变成植物人了。当线程脑死时，堆栈也就消失，故事就这么结束了。

因此，我们还少了一项关键因素——线程的任务，也就是独立线程要跑的程序代码。

Java的多个线程课题意味着我们得要讨论线程与它的任务，以及java.lang中的Thread这个类（要记得java.lang是默认就有被import的，它是包括String和System等语言本身的基础）。

## Java 有多个线程但只有一种 Thread类

当你看到我们讨论线程时代表的是独立的线程，也就是独立的执行空间。当你看到Thread时，代表的是命名习惯，在Java中以大写字母开始的东西是类，所以Thread是java.lang这个包中的一个类。Thread对象代表线程，当你需要启动新的线程时就建立Thread的实例。

线程是独立的线程。它代表独立的执行空间。

Thread是Java中用来表示线程的类。

要建立线程就得创建 Thread。

### thread



主线程



由程序启动的线程

### Thread

|                     |
|---------------------|
| Thread              |
| void join()         |
| void start()        |
| static void sleep() |

java.lang.Thread 类

线程是独立的线程，它代表独立的执行空间。每个Java应用程序会启动一个主线程——将main()放在它自己执行空间的最开始处。Java虚拟机会负责主线程的启动（以及比如垃圾收集所需的系统用线程）。程序员得负责启动自己建立的线程。

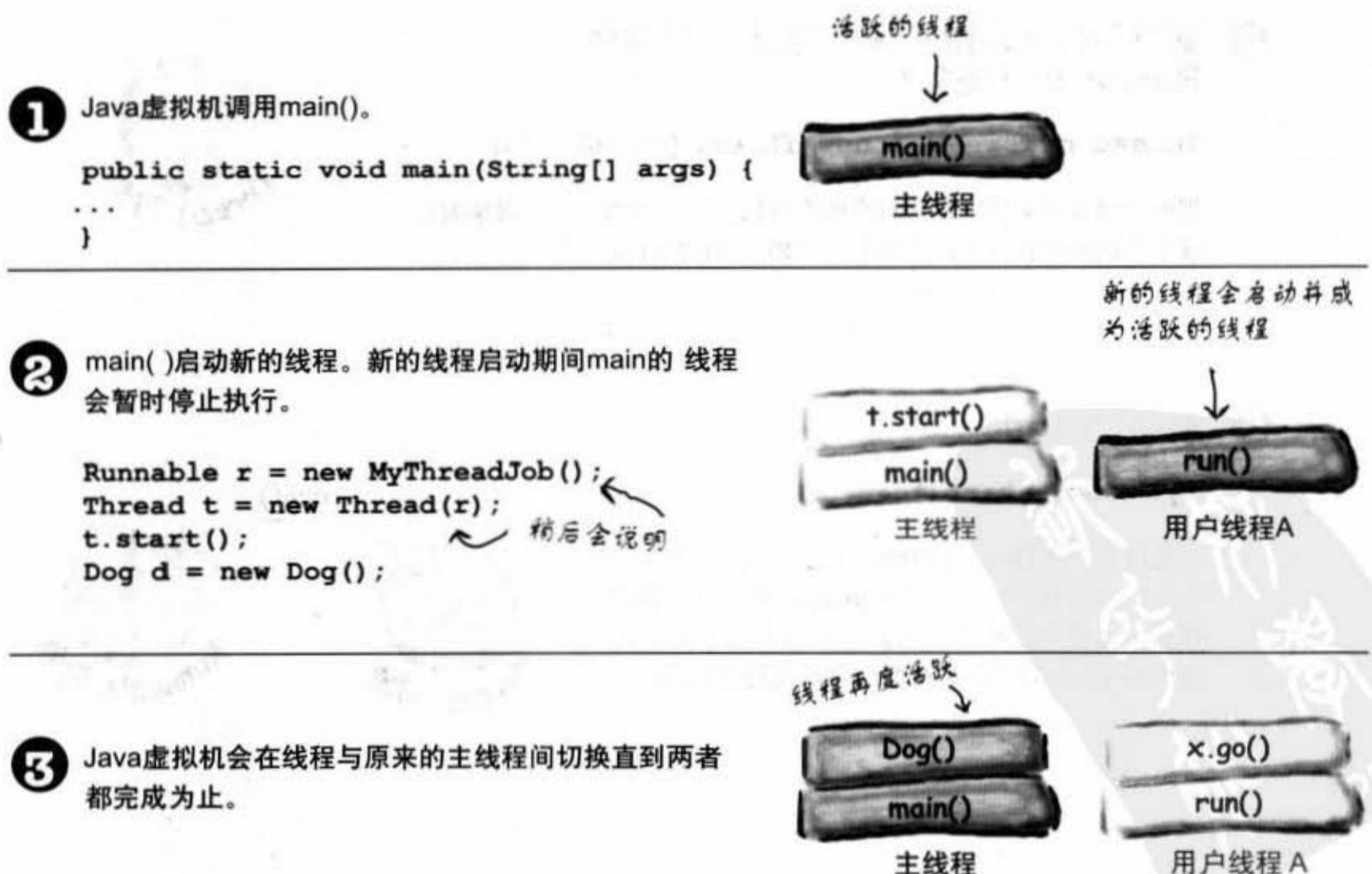
Thread是个表示线程的类。它有启动线程、连接线程和让线程闲置的方法（还有更多，这里只列出重要的几个）。

## 有一个以上的执行空间代表什么？

当有超过一个以上的执行空间时，看起来会像是有好几件事情同时发生。实际上，只有真正的多处理器系统能够同时执行好几件事，但使用Java的线程可以让它看起来好像同时都在执行中。也就是说，执行动作可以在执行空间非常快速地来回交换，因此你会感觉到每项任务都在执行。要记得，Java也只是个在低层操作系统上执行的进程。一旦轮到Java执行的时候，Java虚拟机实际上会执行什么？哪个字节码会被执行？答案是目前执行空间最上面的会被执行！在100个毫秒内，目前执行程序代码会被切换到不同空间上的不同方法。

线程要记录的一项事物是目前线程执行空间做到哪里。

它看起来会像下面这样：



## 如何启动新的线程

### ① 建立Runnable对象（线程的任务）

```
Runnable threadJob = new MyRunnable();
```

稍后会对Runnable这个接口说明。你会编写实现Runnable的类，而此类就是你对线程要执行的任务的定义。也就是说此方法会在线程的执行空间运行。



### ② 建立Thread对象（执行工人）并赋值Runnable（任务）

```
Thread myThread = new Thread(threadJob);
```

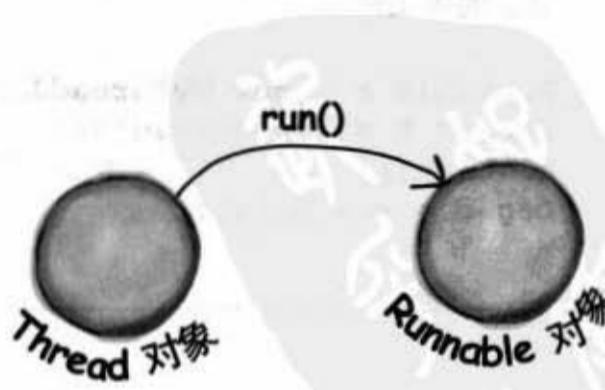
把Runnable对象传给Thread的构造函数。这会告诉Thread对象要把哪个方法放在执行空间去运行——Runnable的run()方法。



### ③ 启动Thread

```
myThread.start();
```

在还没有调用Thread的start()方法之前什么也不会发生。这是你在只有一个Thread实例来建立新的线程时会发生的事情。当新的线程启动之后，它会把Runnable对象的方法摆到新的执行空间中。



每个 Thread 需要一个任务来执行。  
一个可以放在执行空间的任务。



对 Thread 而言，  
它是个工人，而  
Runnable 就是这个工  
人的工作。

Runnable 带有会放在  
执行空间的第一项方  
法：run()。

Thread 对象需要任务。任务是线程在启动时去执行的工作。该任务是新线程空间上的第一个方法，且它一定要长得像下面这样：

```
public void run() {
    // 会被新线程执行的代码
}
```

Runnable 这个接口只有一个方法：public void run()。 (要记得它是个接口，因此不管怎么写它都会是 public 的)

线程怎么会知道要先放上哪个方法？因为 Runnable 定义了一个协议。因为 Runnable 是个接口，线程的任务可以被定义在任何实现 Runnable 的类上。线程只在乎传入给 Thread 的构造函数的参数是否为实现 Runnable 的类。

当你把 Runnable 传给 Thread 的构造函数时，实际上就是在给 Thread 取得 run() 的办法。这就等于你给了 Thread 一项任务。

## 实现Runnable接口来建立给thread运行的任务

```
public class MyRunnable implements Runnable {
    public void run() {
        go();
    }

    public void go() {
        doMore();
    }

    public void doMore() {
        System.out.println("top o' the stack");
    }
}
```

它来自java.lang所以不需要import

```
class ThreadTester {
    public static void main (String[] args) {
        Runnable threadJob = new MyRunnable();
        Thread myThread = new Thread(threadJob);

        1 myThread.start();
        System.out.println("back in main");
    }
}
```

将Runnable的实例传给Thread的构造函数

要调用start()才会让线程开始执行。在此之前，它只是个Thread的实例，并不是真正的线程

1  
myThread.start()  
main()  
主线程

2  
doMore()  
go()  
run()  
新建线程



## 新建线程的3个状态

Thread t = new Thread(r);

新建



可执行



轮到我执行



这是线程的目标

Thread t = new Thread(r);

Thread的实例已经创建，但还没有启动。也就是说，有Thread对象，没有执行中的线程。

t.start();

当你启动线程时，它会变成可执行的状态。意思是说它准备好要执行了，只要轮到它就可以开始。这时，该线程已经布置好执行空间。

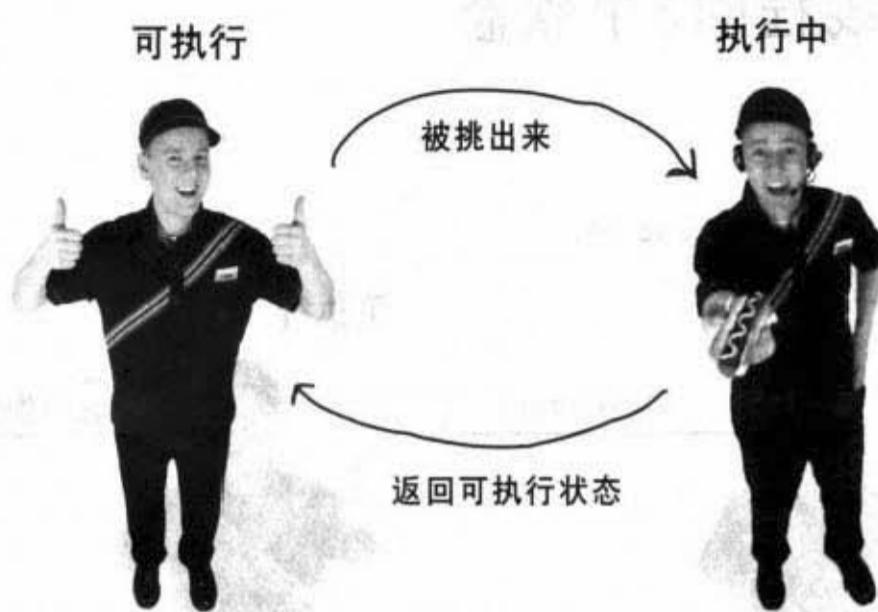
所有的线程都在等待这一刻，成为执行中的那一个！这只能靠Java虚拟机的线程调度机制来决定。你有时也能对Java虚拟机选择执行线程给点意见，但无法强迫它把线程从可执行状态移动到执行中。

不只是这样。一旦线程进入可执行状态，它会在可执行与执行中两种状态中来来去去，同时也有另外一种状态：暂时不可执行（又称为被堵塞状态）。

## 线程状态

### 典型的可执行/执行中循环

通常线程会在可执行与执行中两种状态中来回交替，因为Java虚拟机的线程调度会把线程挑出来运行又把它踢回去以让其他的线程有执行机会。



### 线程有可能会暂时被挡住

调度器（scheduler）会因为某些原因把线程送进去关一阵子。例如线程可能执行到等待Socket输入串流的程序段，但没有数据可供读取。调度器会把线程移出可执行状态，或者线程本身的程序会要求小睡一下（sleep()）。也有可能是因为线程调用某个被锁住（locked）的对象上的方法。此时线程就得等到锁住该对象的线程放开这个对象才能继续下去。

这类型的条件都会导致线程暂时失能。



## 线程调度器

线程调度器会决定哪个线程从等待状况中被挑出来运行，以及何时把哪个线程送回等待被执行的状态。它会决定某个线程要运行多久，当线程被踢出时，调度器也会指定线程要回去等待下一个机会或者是暂时地堵塞。

你无法控制调度，没有API可以调用调度器。最重要的是，调度无法确定（实际上可以做某种程度的保证，但是那也很模糊）。

至少不能让你的程序依靠调度的特定行为来保持执行的正确性！调度器在不同的Java虚拟机上面有不同的做法，就算同一个程序在同一台机器上运行也会有不同的遭遇。Java 程序设计新手会犯的最糟错误就是只在单一的机器上测试多线程程序，并假设其他机器的调度器都有相同的行为。

那写一次就可以到处跑的Java口号不是说很好玩的吗？它是表示你可以写与平台无关Java程序，不管线程调度器有怎样的行为，多线程程序一定可以运行。可是你不能假设每个线程都会被调度分配到公正平均的时间和顺序。且现今的Java虚拟机不太可能让你的线程一路执行到底。

原因在于sleep。没错，就是睡觉。让线程去睡个几毫秒才能让所有的线程都有机会被执行。线程的sleep()这个方法能够保证一件事：在指定的沉睡时间之前，昏睡中的线程一定不会被唤醒。举例来说，如果你要求线程去睡2000个毫秒，至少要等两秒过后它才会继续地执行。



线程调度器会做所有的决定。谁跑谁停都要看它。它通常是很公平的。但没有人能保证这件事，有时候某些线程很受宠，有些线程会被冷落。

## 显示调度器有多个不可预测的范例

找一台机器来运行这个程序

```
public class MyRunnable implements Runnable {  
    public void run() {  
        go();  
    }  
  
    public void go() {  
        doMore();  
    }  
  
    public void doMore() {  
        System.out.println("top o' the stack");  
    }  
  
}  
  
class ThreadTestDrive {  
  
    public static void main (String[] args) {  
  
        Runnable threadJob = new MyRunnable();  
        Thread myThread = new Thread(threadJob);  
  
        myThread.start();  
  
        System.out.println("back in main");  
    }  
}
```

注意到输出顺序是随机的，有时  
候主线程会先结束，有时候新建  
线程会先结束

产生这个输出：

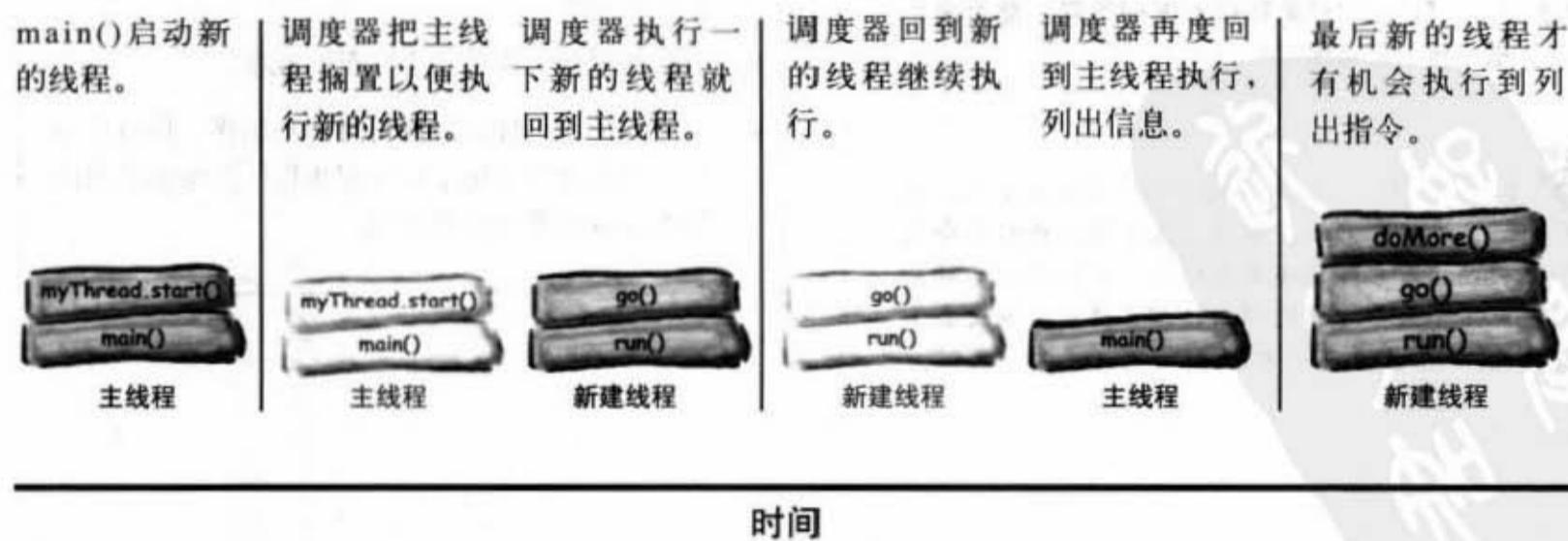
```
File Edit Window Help PickMe  
% java ThreadTestDrive  
back in main  
top o' the stack  
% java ThreadTestDrive  
top o' the stack  
back in main  
% java ThreadTestDrive  
top o' the stack  
back in main  
% java ThreadTestDrive  
top o' the stack  
back in main  
% java ThreadTestDrive  
top o' the stack  
back in main  
% java ThreadTestDrive  
top o' the stack  
back in main  
% java ThreadTestDrive  
back in main  
top o' the stack
```

## 怎么会有不同的结果？

有时它会这样运行：



有时却又是这样运行：



there are no  
Dumb Questions

**问：**我曾经看过不使用Runnable的例子，而用Thread的子类来覆盖掉run()这个方法。然后调用Thread的无参数构造函数来创建出新的线程。

```
Thread t = new Thread();
```

**答：**是的，这是另外一种创建线程的方法，却是以面向对象的观点来建立。子类的目的是什么？要记住我们现在讲的是两回事——Thread与线程的任务。从面向对象观点来看，这两者是非常不同的活动，也是不同的类。你唯一会想要子类/继承Thread的目的是要建立出更特殊的Thread。也就是说如果把Thread当作工人来看的话，除非有很特殊的工作行为，不然你不会继承Thread。如果你只是要有个工人来运行一般的任务，建立实现Runnable的独立类来给工人运行就行。

这跟设计概念有关，而不影响性能或语言用法的好坏。将Thread做个子类来覆盖掉run()是完全合法的，但通常不是个好主意。

**问：**Thread对象可以重复使用吗？能否调用start()指定新的任务给它？

**答：**不行。一旦线程的run()方法完成之后，该线程就不能再重新启动。事实上过了该点线程就会死翘翘。Thread对象可能还呆在堆上，如同活着的对象一般还能接受某些方法的调用，但已经永远地失去了线程的执行性，只剩下对象本身。

## 要点

- 以小写t描述的thread是个独立的线程。
- Java中的每个线程都有独立的执行空间。
- 大写T的Thread是java.lang.Thread这个类。它的对象是用来表示线程。
- Thread需要任务，任务是实现过Runnable的实例。
- Runnable这个接口只有一个方法。
- run()会是新线程所执行的第一项方法。
- 要把Runnable传给Thread的构造函数才能启动新的线程。
- 线程在初始化以后还没有调用start()之前处于新建立的状态。
- 调用Thread对象的start()之后，会建立出新的执行空间，它处于可执行状态等待被挑出来执行。
- 当Java虚拟机的调度器选择某个线程之后它就处于执行中的状态，单处理器的机器只能有一个执行中的线程。
- 有时线程会因为某些原因而被堵塞。
- 调度不能保证任何的执行时间和顺序，所以你不能期待它会完全地平均分配执行，你最多也只能影响sleep的最小保证时间。

## 让线程小睡一下

确保线程能够有机会执行的最好方式是让它们周期性地去睡一下\*。你只要调用sleep()这个方法，传入以毫秒指定的时间就行。

```
Thread.sleep(2000);
```

这会把线程敲昏，而保持两秒之内不会醒来进入可执行状态。

但是这个方法有可能会抛出InterruptedException异常，所有对它的调用都必须包在try/catch块中。因此真正的程序代码会像是这样：

```
try {
    Thread.sleep(2000);
} catch(InterruptedException ex) {
    ex.printStackTrace();
}
```

你写的线程或许永远也不会被中断，这个异常是API用来支持线程间通信的机制，实际上几乎没有人这样做。但良好的习惯规则会要求我们把有可能抛出异常的调用做妥善的处理。

现在你能够确定在指定时间内不会醒来，但是线程也不一定会在时间过后马上醒来直接变成执行中的状态，你只能确定它会回到可执行的状态。何时执行还是要看调度器大哥的意思。这样的运行对时间控制来说不是非常的准确，但在一般机器上没有太多线程在运行的时候还过得去。千万不要依靠这种机制来精确地控制执行时机（比如说画面与声音的同步，不然就有可能在张嘴之前就听到说话，挺恐怖的）。

如果想要确保其他的线程有机会执行的话，就把线程放进睡眠状态。

当线程醒来的时候，它会进入可执行状态等待被调度器挑出来执行。

\*译注：这一页关于sleep()的说法虽然不能算错，但概念上可能会引起误解。我们通常不必刻意使用sleep()来保证其他的线程会被执行。这个问题很复杂，需要一整本书才能讲清楚。

使用 Thread.sleep()

## 使用 sleep() 让程序更加可预测

之前不是有个范例显示出每次运行都可能有不同的结果吗？回头看一下它的程序与输出。有时主程序必须要等到线程做完，有时主程序却会先做完。我们要怎样修正这个状况？先停下来想一下这个问题：“在哪边加上sleep()可以让back……在top……之前打印出？”。

等你想出一个答案再继续下去（可行的答案有好几种）。

有了吗？（恭喜！）

```
public class MyRunnable implements Runnable {  
    public void run() {  
        go();  
    }  
  
    public void go() {  
        try {  
            Thread.sleep(2000);  
        } catch(InterruptedException ex) {  
            ex.printStackTrace();  
        }  
  
        doMore();  
    }  
  
    public void doMore() {  
        System.out.println("top o' the stack");  
    }  
  
    class ThreadTestDrive {  
        public static void main (String[] args) {  
            Runnable theJob = new MyRunnable();  
            Thread t = new Thread(theJob);  
            t.start();  
            System.out.println("back in main");  
        }  
    }  
}
```

我们想要这样的结果：一致的输出顺序：

```
File Edit Window Help SnoozeButton  
% java ThreadTestDrive  
back in main  
top o' the stack  
% java ThreadTestDrive  
back in main  
top o' the stack  
% java ThreadTestDrive  
back in main  
top o' the stack  
% java ThreadTestDrive  
back in main  
top o' the stack  
% java ThreadTestDrive  
back in main  
top o' the stack  
% java ThreadTestDrive  
back in main  
top o' the stack
```

调用 sleep() 来强迫此线程离开执行  
中的状态

主线程会变成执行中的状态，做完之后再回到新线程继续执行，因此打印输出的顺序就会一致，因此两次输出之间会有约两秒的空隙……

译注：又是我。实在不想自嘲，但是这个例子不是很好。这种方式只会浪费两秒的时间，也不是能够100%保证顺序一致。想象一下如果调用sleep()之后操作系统刚好去读硬盘驱动器而花了两秒，回头会执行哪一个线程可就不一定了！（那改成200秒会不会好一点呢？）

## 建立与启动两个线程

线程可以有名字。你可以找老师帮线程取个好听又能够走运的好名字，或是使用默认的名称。但最酷的事情还是你可以用名字来判别正在运行的是哪个线程。下面的例子有两个线程。它们都执行相同的工作：在循环中列出线程的名称。

```
public class RunThreads implements Runnable {
    public static void main(String[] args) {
        RunThreads runner = new RunThreads();
        Thread alpha = new Thread(runner); ← 创建Runnable的实例
        Thread beta = new Thread(runner); ← 创建两个线程，使用相同的Runnable
        alpha.setName("Alpha thread"); ← (相同的任务——稍后会讨论这种做法)
        beta.setName("Beta thread"); ← 帮线程取名字
        alpha.start(); ← 启动线程
        beta.start(); ←
    }
    public void run() {
        for (int i = 0; i < 25; i++) {
            String threadName = Thread.currentThread().getName();
            System.out.println(threadName + " is running");
        }
    }
}
```

## 会发生什么事？

部分的输出 →

线程会轮流执行吗？会交互出现输出吗？多久切换一次？每做完一圈就交换吗？还是要5圈后才交换？

你已经知道答案了：不知道！这都只能听命于调度器的安排。根据操作系统、使用的Java虚拟机版本、CPU等，你会运行出不一样的结果。

在OS X 10.2上面以5圈或更少的圈数来运行时，Alpha会先做完，然后Beta才会做完。结果很一致，然而不保证永远都是这样。

但如果跑上25圈时就不太一样了，Alpha可能跑不完25圈就会切换到Beta上面。



```
File Edit Window Help Centauri
Alpha thread is running
Alpha thread is running
Alpha thread is running
Beta thread is running
Alpha thread is running
Beta thread is running
Alpha thread is running
```

线程好棒啊！



## 呃……实际上有缺点的。 线程会产生并发性的问题。

并发性 (concurrency) 问题会引发竞争状态 (race condition)。竞争状态会引发数据的损毁。数据损毁会引发恐惧……最后连花儿都不开、鸟儿都不叫、人们的脸上也失去了笑容。

这一切都来自于可能发生的一种状况：两个或以上的线程存取单一对象的数据。也就是说两个不同执行空间上的方法都在堆上对同一个对象执行getter或setter。

两个线程各自认为自己是宇宙的中心，只关心自己的任务。因为线程会被打入可执行状态，此时基本上是昏迷过去的，当它回到执行中的状态时，根本也不知道自己曾经不省人事。

# 婚姻亮起了红灯。 这一对怨偶有救吗？

接下来为您播出

“鬼话连篇之马丁博士谈婚姻”

[第 42 集的对白]



欢迎收看鬼话连篇。

我们今天讲的是关于夫妻不和最主要的两个原因：金钱和睡眠。

杰纶与沛晨这一对冤家同居并且也把钱存在一起。但如果不把问题解决掉的话很快就会分手。什么问题呢？典型的贫贱夫妻百事哀。

沛晨是这么抱怨的：

“杰纶跟我说好了不会透支花费，所以规定每个人花钱之前必须先检查余额，这看起来很简单，但有一天我们就突然发现连预借现金的额度也用掉了。

我做梦也没想到会有这么一天，事情是这样的：

杰纶需要 5 万元上夜店，所以查了一下可用额度还有 10 万元，是可以把钱提出来的。但他没有先提钱，反而就不知不觉地睡了起来。

我回家时他还在睡，而我想买个价值 10 万元的新包包，因此查了一下还有 10 万元的额度（杰纶还没有提走钱，所以额度不变），真是太幸运了。然后我把钱提出来去买了包包。接着杰纶也醒了，他就去提钱上夜店。我们就是这样开始产生问题！他根本不知道自己睡了多久，所以醒来之后就去领钱，而没有再检查一次账户余额。

马丁博士，我该怎么办？”

有办法解决吗？他们这一对是不是就毁了？我们无法停止杰纶昏睡的毛病，但是我们能不能要求沛晨在杰纶睡着的时候不要碰提款卡和存折？

不要走开，广告之后我们马上回来看他们的笑话……呃，对不起，是悲剧。没关系，别回来，我们马上走开！



怨偶天成：杰纶与沛晨



杰纶有嗜睡的问题，他会在检查余额与提款的中途睡着，如果一睡不起就算了，可是他总是会醒来领钱，却没有再检查一次余额。

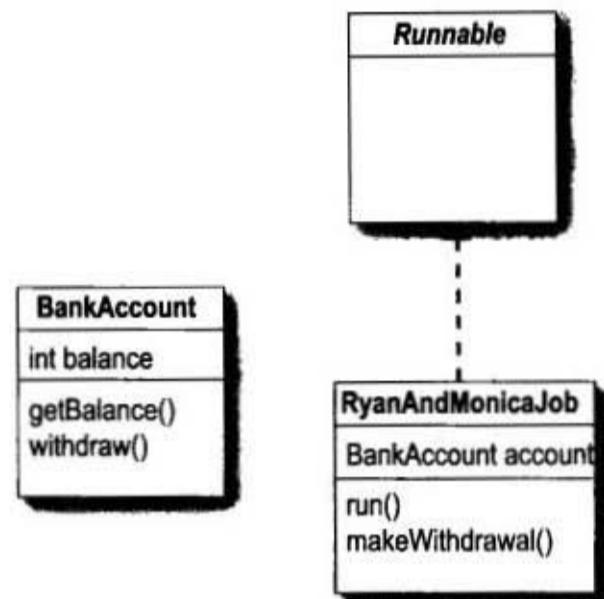
## 以程序码表示杰纶与沛晨的问题

以下的范例展示出两个线程（杰纶 = Ryan，沛晨 = Monica）间共享单一对象（银行账户）时可能会出什么状况。

这个程序代码有两个类，BankAccount与RyanAndMonicaJob。RyanAndMonicaJob这个类实现Runnable，用来代表两人都有的行为——检查余额然后花掉。当然啦，每个线程都会在检查余额与实际提款之间偷睡一下。

MonicaAndRyanJob这个类有个类型为BankAccount的实例变量，它代表共享的账户。

程序代码的工作方式是这样：



### 建立一个RyanAndMonicaJob的实例

它是个Runnable的类（要被执行的任务），因为两人的行为都一样，所以我们只需要一个实例。

```
RyanAndMonicaJob theJob = new RyanAndMonicaJob();
```

在run()这个方法中，它会执行跟故事所说一样的操作。

### 以同一个Runnable建立两个线程（RyanAndMonicaJob 的实例）

```
Thread one = new Thread(theJob);
Thread two = new Thread(theJob);
```

理论上这应该不会遇到提款过多的问题。

### 命名并启动线程

```
one.setName("Ryan");
two.setName("Monica");
one.start();
two.start();
```

除非……杰纶和沛晨都会在检查账户与提款之间睡着。

### 观察两个线程执行run()

两个线程分别代表两人。它们都会持续地检查账户，然后只会在余额足够的情况下提款！

```
if (account.getBalance() >= amount) {
    try {
        Thread.sleep(500);
    } catch(InterruptedException ex) {ex.printStackTrace(); }
}
```

## 杰伦与沛晨的范例

```

class BankAccount {
    private int balance = 100; ← 账户一开始有100元

    public int getBalance() {
        return balance;
    }

    public void withdraw(int amount) {
        balance = balance - amount;
    }
}

public class RyanAndMonicaJob implements Runnable {
    private BankAccount account = new BankAccount(); ← 只有一个RyanAndMonicaJob的实例，代表只有一个共享的账户

    public static void main (String [] args) {
        RyanAndMonicaJob theJob = new RyanAndMonicaJob(); ← 将任务初始化
        Thread one = new Thread(theJob); ← 创建出使用相同任务的两个线程，这代表
        Thread two = new Thread(theJob); ← 两个线程都会存取同一个账户
        one.setName("Ryan");
        two.setName("Monica");
        one.start();
        two.start();
    }

    public void run() {
        for (int x = 0; x < 10; x++) {
            makeWithdrawal(10); ← 检查账户余额，如果透支就列出信息，不然就去睡一会儿，然后醒来完成提款操作
            if (account.getBalance() < 0) {
                System.out.println("Overdrawn!");
            }
        }
    }

    private void makeWithdrawal(int amount) {
        if (account.getBalance() >= amount) {
            System.out.println(Thread.currentThread().getName() + " is about to withdraw"); ← 检查账户余额，如果透支就列出信息，不然就去睡一会儿，然后醒来完成提款操作
            try {
                System.out.println(Thread.currentThread().getName() + " is going to sleep");
                Thread.sleep(500);
            } catch(InterruptedException ex) {ex.printStackTrace();}
            System.out.println(Thread.currentThread().getName() + " woke up.");
            account.withdraw(amount);
            System.out.println(Thread.currentThread().getName() + " completes the withdrawal");
        }
        else {
            System.out.println("Sorry, not enough for " + Thread.currentThread().getName());
        }
    }
}

```

列出信息以便执行时进行观察

```
File Edit Window Help Visa
Ryan is about to withdraw
Ryan is going to sleep
Monica woke up.
Monica completes the withdrawl
Monica is about to withdraw
Monica is going to sleep
Ryan woke up.
Ryan completes the withdrawl
Ryan is about to withdraw
Ryan is going to sleep
Monica woke up.
Monica completes the withdrawl
Monica is about to withdraw
Monica is going to sleep
Ryan woke up.
Ryan completes the withdrawl
Ryan is about to withdraw
Ryan is going to sleep
Monica woke up.
Monica completes the withdrawl
Sorry, not enough for Monica
Ryan woke up.
Ryan completes the withdrawl
Overdrawn!
Sorry, not enough for Ryan
Overdrawn!
Sorry, not enough for Ryan
Overdrawn!
Sorry, not enough for Ryan
Overdrawn!
```

怎么会发生  
这种状况？

makeWithdrawl()这个方法会在提款之前检查账户余额，但不该发生的事情还是发生了。

以下是这个状况的说明：

杰纶检查余额，看到钱够，然后去睡。

同时间，沛晨也来检查余额，也看到有足够的余额。它不知道杰纶等下醒来就要去提款。

沛晨睡了，还流了几滴口水……

杰纶醒来，抽了一根烟，然后把钱提走。

沛晨醒来，发现口水发臭，赶忙洗了把脸，然后提款。这下问题可大了！

沛晨检查余额发现已经透支了，她先是惊慌，然后转为愤怒，最后发现这辈子第一次有想要抢银行的冲动。

沛晨必须要学习在杰纶还没有醒来完成提款之前不能去查询账户余额。反之亦然。

## 他们需要对账户存取的一道锁

这锁会像这样工作：

- ① 锁会与银行账户交易有关（检查余额与提款）。只有一把钥匙，且它会与锁摆在一起直到有人想要存取账户为止。



没有交易时，锁是开着的。

- ② 当杰纶要存取银行账户时，他会把账户锁上并收起钥匙。现在就没有其他人能够存取账户了。



杰纶想要交易，所以把账户给锁上并带走钥匙。

- ③ 杰纶会持有钥匙直到完成交易为止。唯一的钥匙在杰纶手上，因此沛晨无法存取账户，除非杰纶解锁并放回钥匙。

现在就算杰纶检查完账户就去睡觉，也能够保证醒来时账户还是维持原状，因为只有他才有钥匙。



交易完成后就解锁并归还钥匙。现在就可以换其他人存取账户。

## 我们需要让makeWithdrawal()跑起来像个原子



我们必须要确定线程一旦进入makeWithdrawal()这个方法之后，它就必须能够在其他线程进入之前把任务执行完毕。

也就是说我们需要确保线程一旦开始检查账户余额，就要能够确定它会在任何其他线程检查账户余额之前醒来把提款动作完成。

使用**synchronized**这个关键词来修饰方法使它每次只能被单一的线程存取。

这就是保护银行账户的方法！你不会把锁上到账户本身，但你会锁上执行银行交易的方法。如此一来，线程会从头到尾完成交易，就算中途昏睡过去也一样！

如果没有锁上账户，那到底锁了什么？是方法吗？还是Runnable对象？难道是线程本身？

答案在下一页揭晓。然而程序本身是很简单的——只要把synchronized这个修饰符加到方法的声明上就可以。

**synchronized**关键词代表线程需要一把钥匙来存取被同步化（synchronized）过的线程。

要保护数据，就把作用在数据上的方法给同步化。

```
private synchronized void makeWithdrawal(int amount) {
    if (account.getBalance() >= amount) {
        System.out.println(Thread.currentThread().getName() + " is about to withdraw");
        try {
            System.out.println(Thread.currentThread().getName() + " is going to sleep");
            Thread.sleep(500);
        } catch(InterruptedException ex) {ex.printStackTrace();}
        System.out.println(Thread.currentThread().getName() + " woke up.");
        account.withdraw(amount);
        System.out.println(Thread.currentThread().getName() + " completes the withdrawl");
    } else {
        System.out.println("Sorry, not enough for " + Thread.currentThread().getName());
    }
}
```

（物理系的同学请注意：原文所说的原子是“atomic”，这是一种“古典”的说法，古人认为原子是不可分割的最小物质单位，拿来描述我们所说的不可切割工作是很方便的，但这种用法不是我们发明的，不然我们会引用海森堡测不准原理来卖弄一下我们在现代量子物理学上的功力。）

## 使用对象的锁

每个对象都有个锁。大部分时间都没有锁上，并且你可以假设有个虚拟的钥匙随侍在旁。对象的锁只会在有同步化的方法上起作用。当对象有一个或多个同步化的方法时，线程只有在取得对象锁的钥匙时才能进入同步化的方法。

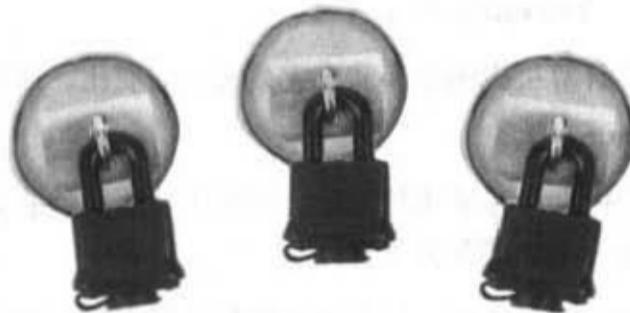
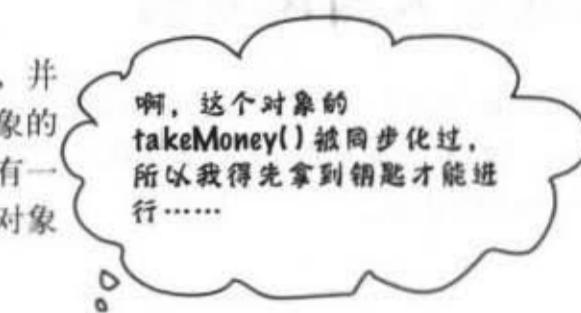
锁不是配在方法上的，而是配在对象上。如果对象有两个同步化的方法，就表示两个线程无法进入同一个方法，也表示两个线程无法进入不同的方法。

想想看，如果你有多个方法可能会操作对象的实例变量，则这些方法都应该要有同步化的保护。

同步化的目标是要保护重要的数据。但要记住，你锁住的不是数据而是存取数据的方法。

所以线程在开始执行并遇上有同步化的方法时候会发生什么事？线程会认知到它需要对象的钥匙才能进入该方法。它会取得钥匙（这是由Java虚拟机来处理，没有存取对象锁的API可用），如果可以拿到钥匙才会进入方法。

从这一点开始，线程会全力照顾好这个钥匙，除非完成同步化的方法，否则它不会放开钥匙。因此当线程持有钥匙时，没有其他的线程可以进入该对象的同步化方法，因为每个对象只有一个钥匙。



每个Java对象都有一个锁，每个锁  
只有一把钥匙。

通常对象都没上锁，也没有人在乎  
这件事。

但如果对象有同步化的方法，则线  
程只能在取得钥匙的情况下进入线  
程。也就是说并没有其他线程已经  
进入的情况下才能进入。

## “丢失更新”问题令人闻风丧胆

下面是另一个典型的并行性（concurrency）问题，这是数据库领域的说法。它跟杰纶与沛晨这个悲剧结合的故事类似，但我们另外使用下面这个例子来展示出几项重点。

丢失更新（lost update）有一种特定的过程。

(1) 取得账户余额。

```
int i = balance;
```

(2) 将账户余额加1。

```
balance = i + 1;
```

这会让计算机以两个步骤来完成账户的变化。通常你会以单一的命令来做这件事：

```
balance++;
```

但强行以两个步骤来处理就会浮现出非原子性的问题。因此你可以想象到如果步骤复杂到无法以单一命令来完成时会怎样。

下面我们用两个都想把余额递增的线程来展示丢失更新。

```
class TestSync implements Runnable {
    private int balance;
    public void run() {
        for(int i = 0; i < 50; i++) { ← 每个线程都把账户递增50次
            increment();
            System.out.println("balance is " + balance);
        }
    }
    public void increment() {
        int i = balance;
        balance = i + 1; ← 问题出在这里，我们用的是读取时的值而不是目前的值
    }
}

public class TestSyncTest {
    public static void main (String[] args) {
        TestSync job = new TestSync();
        Thread a = new Thread(job);
        Thread b = new Thread(job);
        a.start();
        b.start();
    }
}
```

## 执行程序

### ① thread A 运行了几下



把账户余额读进变量  $i$ ，账户为 0，所以  $i$  为 0。

设定账户余额为  $i + 1$ ，账户余额变成 1。

把账户余额读进变量  $i$ ，账户为 1，所以  $i$  为 1。

设定账户余额为  $i + 1$ ，账户余额变成 2。

### ② 换 thread B 运行几下



把账户余额读进变量  $i$ ，账户为 2，所以  $i$  为 2。

设定账户余额为  $i + 1$ ，账户余额变成 3。

把账户余额读进变量  $i$ ，账户为 3，所以  $i$  为 3。

然后 B 睡着了，还没有把账户余额设为 4。 ←

### ③ 再换 thread A 从上次做完的位置开始运行几下。



把账户余额读进变量  $i$ ，账户为 3，所以  $i$  为 3。

设定账户余额为  $i + 1$ ，账户余额变成 4。

把账户余额读进变量  $i$ ，账户为 4，所以  $i$  为 4。

设定账户余额为  $i + 1$ ，账户余额变成 5。

### ④ 轮到 thread B 运行。



设定账户余额为  $i + 1$ ，账户余额变成 4。 ←

哎呀！

B 把 A 所做过动作覆盖掉。  
让 A 的更新看起来好像从未发  
生一般

thread A 的更新就这么消失了！

thread B 之前读取过数据，等到醒  
来后，它就继续写入的操作，完  
全不知道中间丧失过意识

## 用同步机制让 `increment()` 方法原子化！



将 `increment()` 方法同步化可以解决丢失更新问题，因为它会让方法中的两个步骤组成不可分割的单元。

```
public synchronized void increment() {  
    int i = balance;  
    balance = i + 1;  
}
```

一旦线程进入了方法，我们必须确保在其他线程可以进入该方法之前所有的步骤都会完成（如同原子不可分割一样）。

there are no  
Dumb Questions

**问：** 听起来把所有东西都同步化是个不错的主意，如此一来全部都会具有多线程执行的安全性。

**答：** 镶主意。同步化不是没有代价的。首先，同步化的方法有些额外的成本。也就是说进入同步化方法的程序会查询钥匙等性能上的损耗（虽然你不太会注意到）。

其次，同步化的方法会让你的程序因为要同步并行的问题而慢下来。换句话说，同步化会强制线程排队等着执行方法，也许听起来没什么，但你得要想想一开始为什么要写多线程并行的程序。

最后，最可怕的是同步化可能会导致死锁现象（见516页）。

原则上最好只做最少量的同步化。事实上同步化的规模可以小于方法全部。本书不会深入这个部分，但你可用 `synchronized` 来修饰一行或数行的指令而不必整个方法都同步化。

```
public void go() {  
    doStuff();  
  
    synchronized(this) {  
        criticalStuff();  
        moreCriticalStuff();  
    }  
}
```

不需要整个都同步化  
只有这两个调用要被组合成原子单位，同步化的这种用法不会将整个方法设定成需要同步化，只会使用参数所指定的对象的锁来做同步化。

虽然有别的方法可以达到同样的效果，但你通常会以当前对象 (`this`) 来同步化。

## ① thread A运行了几下



尝试进入increment()方法，因为是同步化的方法，所以要取得钥匙。

把账户余额读进变量i；账户为0，所以i为0。

设定账户余额为i + 1；账户余额变成1。

归还钥匙。

再重新进入，取得钥匙，余额为1，所以i为1。

[进入睡眠状态，但因为还没有完成同步化的方法，所以钥匙还在手上]

## ② 换thread B运行几下



尝试进入increment()方法，因为是同步化的方法，所以要取得钥匙。

拿不到钥匙。

[B只好进入等待对象锁钥的状态]

## ③ 再换thread A从上次做完的位置开始运行几下

(注意到钥匙还在手上)



设定账户余额为i + 1；账户余额变成2。

归还钥匙。

[A进入可执行状态]

## ④ 轮到thread B运行



尝试进入increment()方法，因为是同步化的方法，所以要取得钥匙。

这一次拿到钥匙了……

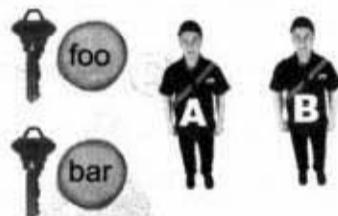
## 同步化的死亡阴影

使用同步化的程序代码要小心，因为没有其他的东西能够像线程的死锁（deadlock）这样伤害你的程序。死锁会发生是因为两个线程互相持有对方正在等待的东西。没有办法可以脱离这个情况，所以两个线程只好停下来等，一直等，一直等，海枯石烂还在继续等。

如果你对数据库或其他的应用程序服务器很熟，那你就应该知道这个问题：数据库有与同步化非常相似的上锁机制。但像样的数据库交易管理系统有时能处理掉死锁。例如它可能会把等待太久的交易视为死锁。但与Java不同的地方在于它们有事务回滚机制来复原不能全部完成的交易。

Java没有处理死锁的机制。它甚至不会知道死锁的发生。所以你得小心设计程序。如果你经常编写多线程的程序，建议阅读O'Reilly出版的“Java Thread”，上面有一些观念的澄清和设计的提示可以帮助避免死锁（译注：有中文译本，译文精确优雅，译者帅气逼人，是本不可多得的好书）。

只要两个线程和两个对象  
就可以引起死锁



## 要点

- `Thread.sleep()`这个静态方法可以强制线程进入等待状态到过了设定时间为止，例如`Thread.sleep(200)`会睡上200个毫秒。
- 可以调用`sleep()`让所有的线程都有机会运行。
- `sleep()`方法可能会抛出`InterruptedException`异常，所以要包在`try/catch`块，或者把它也声明出来。
- 你可以用`setName()`方法来帮线程命名，通常是用来除错的。
- 如果两个或以上的线程存取堆上相同的对象可能会出现严重的问题。
- 如果两个或两个以上的线程存取相同的对象可能会引发数据的损毁。
- 要让对象在线程上有足够的安全性，就要判断出哪些指令不能被分割执行。
- 使用`synchronized`这个关键词修饰符可以防止两个线程同时进入同一对象的同一方法。
- 每个对象都有单一的锁，单一的钥匙。这只会在对象带有同步化方法时才有实际的用途。
- 线程尝试要进入同步化过的方法时必须要取得对象的钥匙，如果因为已经被别的线程拿走了，那就得等。
- 对象就算是有多个同步化过的方法，也还是只有一个锁。一旦某个线程进入该对象的同步化方法，其他线程就无法进入该对象上的任何同步化线程。

## 全新配方的 SimpleChatClient

回到本章开始的主题，我们创建出 SimpleChatClient 来发送信息给服务器，但无法接收。这就是为什么要讨论线程的原因，因为我们需要能够同时做两件事的方法：送出信息给服务器的同时读取来自服务器的信息，并显示在可滚动的区域。

```

import java.io.*;
import java.net.*;
import java.util.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class SimpleChatClient {

    JTextArea incoming;
    JTextField outgoing;
    BufferedReader reader;
    PrintWriter writer;
    Socket sock;

    public static void main(String[] args) {
        SimpleChatClient client = new SimpleChatClient();
        client.go();
    }

    public void go() {

        JFrame frame = new JFrame("Ludicrously Simple Chat Client");
        JPanel mainPanel = new JPanel();
        incoming = new JTextArea(15, 50);
        incoming.setLineWrap(true);
        incoming.setWrapStyleWord(true);
        incoming.setEditable(false);
        JScrollPane qScroller = new JScrollPane(incoming);
        qScroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
        qScroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
        outgoing = new JTextField(20);
        JButton sendButton = new JButton("Send");
        sendButton.addActionListener(new SendButtonListener());
        mainPanel.add(qScroller);
        mainPanel.add(outgoing);
        mainPanel.add(sendButton);
        setUpNetworking();

        Thread readerThread = new Thread(new IncomingReader());
        readerThread.start();

        frame.getContentPane().add(BorderLayout.CENTER, mainPanel);
        frame.setSize(400, 500);
        frame.setVisible(true);
    } // 关闭 go
}

```

是的，本章要结束了，  
但是……

除了标记出来的段落之外，这边  
都是之前看过的GUI程序代码

启动新的线程，以内部类  
作为任务，此任务是读取  
服务器的socket串流显示在  
文本区域

```

private void setUpNetworking() {
    try {
        sock = new Socket("127.0.0.1", 5000);
        InputStreamReader streamReader = new InputStreamReader(sock.getInputStream());
        reader = new BufferedReader(streamReader);
        writer = new PrintWriter(sock.getOutputStream());
        System.out.println("networking established");
    } catch(IOException ex) { 使用socket取得输入/输出的串流
        ex.printStackTrace();
    }
} // 关闭setUpNetworking

public class SendButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent ev) {
        try {
            writer.println(outgoing.getText());
            writer.flush(); 用户按下send按钮时送出文本字
           段的内容到服务器上
        } catch(Exception ex) {
            ex.printStackTrace();
        }
        outgoing.setText("");
        outgoing.requestFocus();
    }
} // 关闭内部类

public class IncomingReader implements Runnable {
    public void run() {
        String message;
        try {
            while ((message = reader.readLine()) != null) {
                System.out.println("read " + message);
                incoming.append(message + "\n");
            }
        } // while结束
        catch(Exception ex) {ex.printStackTrace();}
    } // 关闭run()
} // 关闭内部类
} // 关闭外部类

```

thread的任务!

run()会持续地读取服务器信息并把它加到可滚动的文本区域上



## 现成码

## 非常非常简单的聊天服务器程序

你可以用这个服务器程序来服务先后两版的聊天客户端程序。我们去掉很多功能来让它精简。也就是说，这个程序可用，但至少有100种方法可以让它死掉。如果你真的很想磨练你的功夫，可以在这本书结束之后回头加强这个程序。

另外一种现在就可以进行锻炼的方法是自己帮程序加注释。如果你自己搞定程序的来龙去脉会比我们跟你说要好。再说一次，这边列出的是现成可用的程序，所以不一定要全部看懂。它的目的只是为了要让客户端可以测试。

你需要用一个命令列来启动服务器，然  
后再用另外一个命令列来启动客户端。

```
import java.io.*;
import java.net.*;
import java.util.*;

public class VerySimpleChatServer {

    ArrayList clientOutputStreams;

    public class ClientHandler implements Runnable {
        BufferedReader reader;
        Socket sock;

        public ClientHandler(Socket clientSocket) {
            try {
                sock = clientSocket;
                InputStreamReader isReader = new InputStreamReader(sock.getInputStream());
                reader = new BufferedReader(isReader);
            } catch (Exception ex) {ex.printStackTrace();}
        } // 关闭构造函数

        public void run() {
            String message;
            try {
                while ((message = reader.readLine()) != null) {
                    System.out.println("read " + message);
                    tellEveryone(message);

                } // 结束while循环
            } catch (Exception ex) {ex.printStackTrace();}
        } // 关闭run()
    } // 关闭内部类
}
```

```

public static void main (String[] args) {
    new VerySimpleChatServer().go();
}

public void go() {
    clientOutputStreams = new ArrayList();
    try {
        ServerSocket serverSock = new ServerSocket(5000);

        while(true) {
            Socket clientSocket = serverSock.accept();
            PrintWriter writer = new PrintWriter(clientSocket.getOutputStream());
            clientOutputStreams.add(writer);

            Thread t = new Thread(new ClientHandler(clientSocket));
            t.start();
            System.out.println("got a connection");
        }
    } catch(Exception ex) {
        ex.printStackTrace();
    }
} // 关闭go

public void tellEveryone(String message) {

    Iterator it = clientOutputStreams.iterator();
    while(it.hasNext()) {
        try {
            PrintWriter writer = (PrintWriter) it.next();
            writer.println(message);
            writer.flush();
        } catch(Exception ex) {
            ex.printStackTrace();
        }
    }
} // 结束while

} // 关闭tellEveryone
} // 关闭类

```

there are no  
Dumb Questions

**问：**那么静态变量状态的保护呢？如果有静态的方法可以对静态变量的状态作更新，还能够用同步化吗？

**答：**可以！记得静态的方法是运行在类而不是每个实例上的吗？所以你可能会猜想要用哪个对象的锁。毕竟有可能完全没有该类的实例存在。幸好对象有锁，每个被载入的类也有个锁。这表示说如果有3个Dog对象在堆上，则总共有4个与Dog有关的锁。3个是Dog实例的，1个是类的。当你要对静态的方法做同步化时，Java会使用类本身的锁。因此如果同一个类有两个被同步化过的静态方法，线程需要取得类的锁才能进入这些方法。

**问：**什么是线程优先级？我听说它可以用来自控制调度。

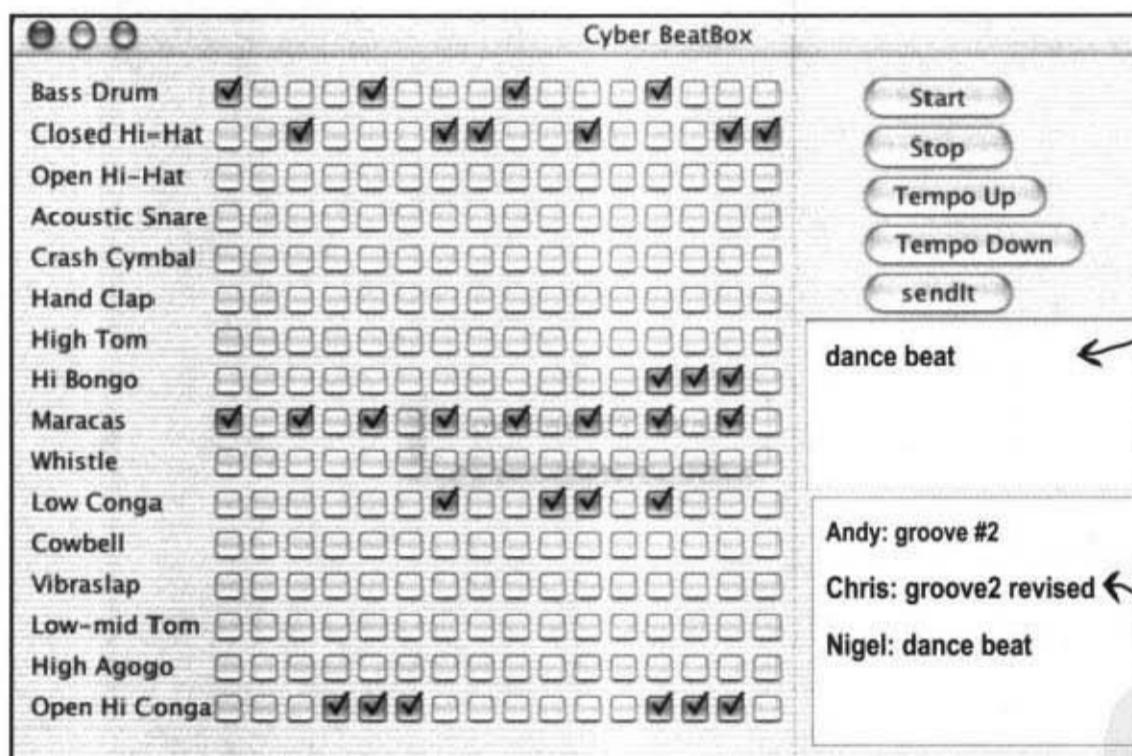
**答：**线程的优先级可以对调度器产生影响，但也是没有绝对的保证。优先权的级别会告诉调度器某个线程的重要性，低优先级的线程也许会把机会让给高优先级的线程，也许……建议你可以用优先级来影响执行性能，但绝不能依靠优先级来维持程序的正确性。

**问：**为什么不把要被保护的数据的getter与setter都给同步化？例如把账户类上检查余额与提款的两个方法都同步化，而不是把Runnable任务类上的检查余额加上提款操作的这个方法同步化？

**答：**事实上，我们应该要把这些方法都同步化，以防止其他的线程以别种方式存取它们。我们没有这么做是因为没有别的程序代码会存取账户余额。但只是将getter与setter同步化是不够的。要记得同步化的意义是指定某段工作要在不能分割的状态下执行。也就是说单独的操作不重要，重要的是有多个步骤的方法。想想看，如果只是把检查余额单一操作的方法给同步化，杰纶与沛晨的问题同样还是会发生的。因为此时检查余额与提款还是有被分割的可能。

因此把所有存取的方法都同步化是个好主意，但还是得要把不可分割的原子单元作同步化。

# 程序料理



这是BeatBox的最终决定版！

它能连接到MusicServer上以便发送与接收其他客户端的  
节拍样式。

程序代码相当长，所以完整的列表是放在附录A。



## 排排看

下一页有被打散的程序代码。你是否能够重组这些程序代码来产生下面的输出？你可以自己加入括号来保持程序的正确性。

```
public class TestThreads {
```

```
class Accum {
```

```
class ThreadOne
```

```
class ThreadTwo
```

```
File Edit Window Help Sewing
* java TestThreads
one 98098
two 98099
```

## 排排看 (续)

```

Thread one = new Thread(t1);
} catch(InterruptedException ex) { }

Thread two = new Thread(t2);
Accum a = Accum.getAccum();
public static Accum getAccum() {
private int counter = 0;
a.updateCounter(1);
for(int x=0; x < 99; x++) {
public int getCount() {
public void updateCounter(int add) {
for(int x=0; x < 98; x++) {
try {
public void run() {
private Accum() { }
Accum a = Accum.getAccum();
System.out.println("two "+a.getCount());
ThreadTwo t2 = new ThreadTwo();
try {
return counter; counter += add;
implements Runnable {
one.start();
Thread.sleep(50);
} catch(InterruptedException ex) { }

private static Accum a = new Accum();
public void run() {
Thread.sleep(50);
implements Runnable {
a.updateCounter(1000);
return a;
System.out.println("one "+a.getCount());
public static void main(String [] args) {
ThreadOne t1 = new ThreadOne();

```

## 练习解答

```
public class TestThreads {
    public static void main(String [] args) {
        ThreadOne t1 = new ThreadOne();
        ThreadTwo t2 = new ThreadTwo();
        Thread one = new Thread(t1);
        Thread two = new Thread(t2);
        one.start();
        two.start();
    }
}

class Accum {
    private static Accum a = new Accum();
    private int counter = 0;

    private Accum() {} 私用的构造函数
    public static Accum getAccum() {
        return a;
    }

    public void updateCounter(int add) {
        counter += add;
    }

    public int getCount() {
        return counter;
    }
}

class ThreadOne implements Runnable {
    Accum a = Accum.getAccum();
    public void run() {
        for(int x=0; x < 98; x++) {
            a.updateCounter(1000);
            try {
                Thread.sleep(50);
            } catch(InterruptedException ex) {}
        }
        System.out.println("one "+a.getCount());
    }
}
```

## 练习解答

两个不同的类会对另一类的同对象作更新，因为两个线程会去存取Accum唯一的实例。

```
private static Accum a = new Accum();
```

上面这行程序会创建Accum的静态实例，而私用的构造函数代表其他人无法创建它的对象。运用这两项技术能够做出称为Singleton的模式，它能限制应用程序上某对象实例的数量（通常会跟名字一样限制一个）。但你也可以使用相同的模式来做出想要的限制。

```
class ThreadTwo implements Runnable {
    Accum a = Accum.getAccum();
    public void run() {
        for(int x=0; x < 99; x++) {
            a.updateCounter(1);
            try {
                Thread.sleep(50);
            } catch(InterruptedException ex) {}
        }
        System.out.println("two "+a.getCount());
    }
}
```



## 差点出错的气闸舱

### 5分钟 短剧



温迪正在与开发小组进行设计会议，她看了看窗外的印度洋日出，虽然船上会议室的豪华设备无法掩饰狭小空间带来的封闭感，但破晓时分黑夜与白昼交会刹那的美景还是能缓和她的不安。

这一天早上的会议主题是太空站的气闸舱（airlock）控制系统。现在已经接近最后完工的阶段，太空漫游的行程也快要排满了，航天员出入气闸舱的交通将会非常的忙碌。

“早啊，”彼特打了招呼“来得正好，我们马上开始讨论设计细节”。

彼特直接切入主题，一点时间都不浪费。“大家都知道，每个气闸舱内外都有 GUI 操作终端机，航天员可以通过终端机操作气闸程序。”温迪点了点头说道：“彼特，你能不能帮大家说明进入和离开气闸的程序？”彼特转身向白板过去开始画出方法调用顺序。

```
orbiterAirlockExitSequence() //出舱程序
    verifyPortalStatus(); //检查入口状态
    pressurizeAirlock(); //气闸舱加压
    openInnerHatch(); //打开内侧闸门
    confirmAirlockOccupied(); //确认气闸舱有人
    closeInnerHatch(); //关闭内侧闸门
    decompressAirlock(); //气闸舱泄压
    openOuterHatch(); //打开外侧闸门
    confirmAirlockVacated(); //确认气闸舱无人
    closeOuterHatch(); //关闭外侧闸门
```

“为了要确保此过程不会被中断，我们已经把所有被 `orbiterAirlockExitSequence()` 所调用的方法都做了同步化，”彼特边画边说明“我实在很讨厌看到航天员把航天服的裤子脱到一半的样子”。

每个人都同意彼特的说法，但温迪隐隐约约觉得有什么事情不太对劲。“等一下！”温迪知道是什么了“这个程序会出人命的！”

温迪发现了什么？彼特是不是犯了什么错？



### 温迪发现什么问题？

她发现为了要确保整个离舱程序不会被打断，`orbiterAirlockExitSequence()`这个方法必须要被同步化。现有的设计会在离舱过程进行到一半的时候被进舱的航天员给打断！个别的操作虽然做同步化，但动作间仍有被插断的可能。整个程序应该要以不可分割的方式进行才能确保外层空间不会有没穿裤子的航天员尸体。

# 数据结构

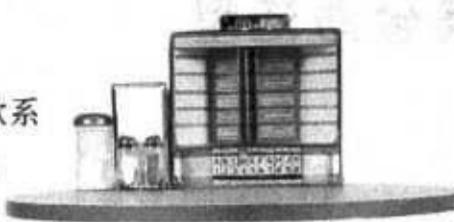


唉……我连排序都不会，将来  
怎么卖鸡排呢？还是专攻应用  
数学好了，听说数学老师常常  
请假……

**排序在Java中只是雕虫小技。**你不必自己编写排序算法就有一大堆现成的工具可供收集与操作数据（除非你正在上电工课，老师铁定会叫你写排序程序的作业）。Java集合框架（Collections Framework）能够支持绝大多数你会用到的数据结构。想要很容易加入元素的列表吗？想要根据名称来搜索吗？打算创建可以自动排除重复项目的列表吗？需要将同事暗算你的次数排个复仇黑名单吗？这里全都有……

## 记录KTV最常点的歌

欢迎你到大富豪 KTV 担任点歌系统管理员一职。虽然点歌系统没有内置的Java，但是只要有人点歌，数据就会自动记录到一个文本文件中。



你的工作是要管理点播记录、产生报表和管理歌本。你并不需要写出整个程序——很多同事都是找不到程序设计工作才来当服务生的，所以大家可以分工合作，你只需要负责用 Java 程序来把数据排序，并且偶尔帮忙打扫包厢就行。因为老板很抠，所以公司没买数据库应用程序，你只能靠内存上的数据集合，还有记录用的文本文件。

你已经知道如何读取与解析文件，并且也用 ArrayList 来排序。

### **SongList.txt**

```
Communication/The Cardigans
Black Dog/Led Zeppelin
Dreams/Van Halen
Comfortably Numb/Pink Floyd
Beth/Kiss
倒退噜/黄克林
```

这是点歌系统写出的文件，你的程序必须读取文件才能操作数据

### **挑战一**

#### **以歌名来排序**

文件上列出歌曲，每行代表一首歌，歌名与歌星用斜线分开，所以应该很容易解析并放到 ArrayList 上。

消费者通常是看歌名点歌，所以目前只要记歌名就行。

但你会注意到歌曲没有依据字母排序，要怎么办呢？

你知道使用 ArrayList 的时候，元素会维持被加入 ArrayList 的顺序，所以它们不会依照字母排序，或许 ArrayList 这个类有个 sort() 方法可以用吧？

## 下面是目前的程序，没有排序功能

```

import java.util.*;
import java.io.*;

public class Jukebox1 {
    ArrayList<String> songList = new ArrayList<String>();

    public static void main(String[] args) {
        new Jukebox1().go();
    }

    public void go() {
        getSongs();
        System.out.println(songList);
    }

    void getSong() {
        try {
            File file = new File("SongList.txt");
            BufferedReader reader = new BufferedReader(new FileReader(file));
            String line = null;
            while ((line = reader.readLine()) != null) {
                addSong(line);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    void addSong(String lineToParse) {
        String[] tokens = lineToParse.split("/");
        songList.add(tokens[0]);
    }
}

```

歌曲名称会存在String的ArrayList上

这个方法会载入文件并列出内容

读取文件的程序

split()方法会用反斜线来拆开歌曲内容

因为只需要歌名，所以只取第一项加入Songlist

```

File Edit Window Help Dance
%java Jukebox1
[Communication, Black Dog,
Dreams, Comfortably Numb,
Beth, 倒退噜]

```

依照加入的顺序列出，与原始的文本文件顺序相同  
这铁定不是歌本的排列方式！

## 但是ArrayList没有sort()这个方法

当你查询ArrayList的说明时，看起来是没有任何方法和排序有关。

查询整个继承结构也没有帮助，很明显ArrayList就是不能排序。

ArrayList (Java 2 Platform SE 5.0)

Method Summary

|           |                                                                                                                                                                                                                      |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| boolean   | <code>add(E e)</code><br>Appends the specified element to the end of this list.                                                                                                                                      |
| void      | <code>add(int index, E element)</code><br>Inserts the specified element at the specified position in this list.                                                                                                      |
| boolean   | <code>addAll(Collection&lt;? extends E&gt; c)</code><br>Appends all of the elements in the specified Collection to the end of this list, in the order that they are returned by the specified Collection's iterator. |
| boolean   | <code>addAll(int index, Collection&lt;? extends E&gt; c)</code><br>Inserts all of the elements in the specified Collection into this list, starting at the specified position.                                       |
| void      | <code>clear()</code><br>Removes all of the elements from this list.                                                                                                                                                  |
| Object    | <code>clone()</code><br>Returns a shallow copy of this ArrayList instance.                                                                                                                                           |
| boolean   | <code>contains(Object elem)</code><br>Returns <code>true</code> if this list contains the specified element.                                                                                                         |
| ++14      | <code>ensureCapacity(int minCapacity)</code><br>Increases the capacity of this ArrayList instance specified by the minimum capacity argument.                                                                        |
| E         | <code>get(int index)</code><br>Returns the element at the specified position in this list.                                                                                                                           |
| int       | <code>indexOf(Object elem)</code><br>Searches for the first occurrence of the given element in this list.                                                                                                            |
| boolean   | <code>isEmpty()</code><br>Tests if this list has no elements.                                                                                                                                                        |
| int       | <code>lastIndexOf(Object elem)</code><br>Returns the index of the last occurrence of the given element in this list.                                                                                                 |
| E         | <code>remove(int index)</code><br>Removes the element at the specified position in this list.                                                                                                                        |
| boolean   | <code>remove(Object o)</code><br>Removes a single instance of the specified element from this list.                                                                                                                  |
| protected | <code>removeRange(int fromIndex, int toIndex)</code><br>Removes from this List all of the elements between the specified indices.                                                                                    |
| E         | <code>set(int index, E element)</code><br>Replaces the element at the specified position in this list with the specified element.                                                                                    |
| int       | <code>size()</code><br>Returns the number of elements in this list.                                                                                                                                                  |
| Object[]  | <code>toArray()</code><br>Returns an array containing all of the elements in this list in the correct order.                                                                                                         |
| <T> T[]   | <code>toArray(T[] a)</code><br>Returns an array containing all of the elements in this list in the correct order; the runtime type of the returned array is that of the specified array.                             |
| ++14      | <code>trimToSize()</code><br>Trims the capacity of this ArrayList instance to be the list's current size.                                                                                                            |

Methods inherited from class `java.util.AbstractList`

`equals, hashCode, iterator, ListIterator, ListIterator, subList`

ArrayList 有一大堆方法，但就是没有可以排序的.....



## ArrayList不是唯一的集合

虽然ArrayList会是最常用的, 但偶尔还是会有特殊情况。  
下面列出几个较为重要的。

稍后还会有更深入的说明

### ► TreeSet

以有序状态保持并可防止重复。

### ► HashMap

可用成对的name/value来保存与取出。

### ► LinkedList

针对经常插入或删除中间元素所设计的高效率集合。  
(实际上ArrayList还是比较实用)

### ► HashSet

防止重复的集合, 可快速地找寻相符的元素。

### ► LinkedHashMap

类似HashMap, 但可记住元素插入的顺序, 也可以  
设定成依照元素上次存取的先后来排序。

## 你可以使用TreeSet或Collections.sort()方法

如果你把字符串放进TreeSet而不是ArrayList，这些String会自动地按照字母顺序排在正确的位置。每当你想要列出清单时，元素总是会以字母顺序出现。

当你需要set集合（稍后讨论）或总是会依照字母排列的清单时，它会很好用。

另外一方面，如果你没有需要让清单保持有序的状态，TreeSet的成本会比你想付出的还多——每当插入新项目时，它都必须花时间找出适当的位置。而ArrayList只要把项目放在最后面就好。

**问：** 但你可以用指定的索引来添加新项目到ArrayList中，而不是放到最后面——add()有个重载的版本可以指定int值。这样会比较慢吗？

**答：** 是的，插到指定位置会比直接加到最后面要慢。所以add(index, element)不会像add(element)这么快。但通常你不会对ArrayList加上指定的索引。

**问：** 使用LinkedList这个类会不会比较好？我记得以前上数据结构的课时是这样说的。

**答：** 没错，LinkedList对于在中间的插入或删除会比较快，但对大多数的应用程序而言ArrayList与LinkedList的差异有限，除非元素量真的很大。稍后会讨论LinkedList。

**java.util.Collections**

```
public static void copy(List destination, List source)
public static List emptyList()
public static void fill(List listToFill, Object objToFillItWith)
public static int frequency(Collection c, Object o)
public static void reverse(List list)
public static void rotate(List list, int distance)
public static void shuffle(List list)
public static void sort(List list)
public static boolean swapAll(List list, Object oldVal, Object newVal)
// many more methods...
```

嗯……Collections这个类有个sort()方法。它会用到List，而ArrayList有实现List这个接口，所以ArrayList是一个List。感谢多态机制，你确实可以把ArrayList传给用到List的方法。

注意：这不是完整的API说明，我们把稍后会讨论的generic信息拿掉来加以简化。

## 对点歌系统加上 Collections.sort()

```

import java.util.*;
import java.io.*;

public class Jukebox1 {
    ArrayList<String> songList = new ArrayList<String>();

    public static void main(String[] args) {
        new Jukebox1().go();
    }

    public void go() {
        getSong();
        System.out.println(songList);
        Collections.sort(songList);
        System.out.println(songList);
    }

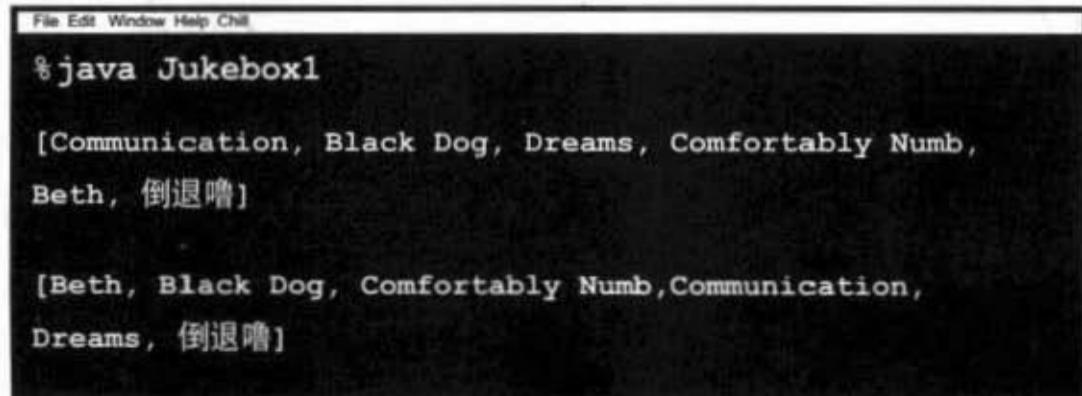
    void getSong() {
        try {
            File file = new File("SongList.txt");
            BufferedReader reader = new BufferedReader(new FileReader(file));
            String line = null;
            while ((line = reader.readLine()) != null) {
                addSong(line);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    void addSong(String lineToParse) {
        String[] tokens = lineToParse.split("/");
        songList.add(tokens[0]);
    }
}

```

**Collections.sort()**会把 list 中的 String 依照字母排序

调用 Collection 静态的 sort() 然后再列出清单。第二次的输出会依照字母排序！



```

File Edit Window Help Chill
%java Jukebox1
[Communication, Black Dog, Dreams, Comfortably Numb,
Beth, 倒退噜]

[Beth, Black Dog, Comfortably Numb, Communication,
Dreams, 倒退噜]

```



## 但是现在要用Song对象而不只是String

老板说list里面要摆的是Song这个类的实例，这样新的点歌系统才会有更细节的资料可以输出。所以文件内也会从两种数据增加到4种。

Song这个类是很单纯的，但有一项很有意思的功能：被覆盖过的toString()。要知道toString()是定义在Object这个类中，所以Java中的每个类都有继承到，且因为对象被System.out.println(anObject)列出来时会被调用toString()，所以当你要把list列出时，每个对象的toString()都会被调用一次。

```
class Song {  
    String title;  
    String artist;  
    String rating;  
    String bpm;  
  
    Song(String t, String a, String r, String b) {  
        title = t;  
        artist = a;    变量都会在创建时从构造  
        rating = r;  函数中设定  
        bpm = b;  
    }  
  
    public String getTitle() {  
        return title;  
    }  
  
    public String getArtist() {  
        return artist;  
    }  
  
    public String getRating() {  
        return rating;  
    }  
  
    public String getBpm() {  
        return bpm;  
    }  
  
    public String toString() { ← 将toString()覆盖过，让它返回  
        return title;  
    }  
}
```

### SongListMore.txt

```
Communication/The  
Cardigans/5/80  
Black Dog/Led Zeppelin/4/84  
Dreams/Van Halen/6/120  
Comfortably Numb/Pink  
Floyd/5/110  
Beth/Kiss/4/100  
倒退噜/黄克林/5/90
```

新的歌曲文件带有4项属性，  
所以我们需要创建出Song  
的实例变量来带这些属性。

## 修改点歌系统程序

程序代码只有改一点点，文件输入/输出的部分不变，除了属性变成4个之外，解析的部分也一样使用String.split()。当然ArrayList的类型要从<String>改成<Song>。

```

import java.util.*;
import java.io.*;
public class Jukebox3 {
    ArrayList<Song> songList = new ArrayList<Song>();
    public static void main(String[] args) {
        new Jukebox3().go();
    }
    public void go() {
        getSongs();
        System.out.println(songList);
        Collections.sort(songList);
        System.out.println(songList);
    }
    void getSong() {
        try {
            File file = new File("SongList.txt");
            BufferedReader reader = new BufferedReader(new FileReader(file));
            String line = null;
            while ((line = reader.readLine()) != null) {
                addSong(line);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
    void addSong(String lineToParse) {
        String[] tokens = lineToParse.split("/");
        Song nextSong = new Song(tokens[0], tokens[1], tokens[2], tokens[3]);
        songList.add(nextSong);
    }
}

```

将String改成Song类型

使用解析出来的4项属性来创建  
Song对象并加入到list中

Collections.sort()

## 无法通过编译！

有点问题，Collections类很明显地说明sort()这个方法会取用List。

ArrayList是一个List，因为ArrayList有实现List这个接口，所以应该要没问题才对。

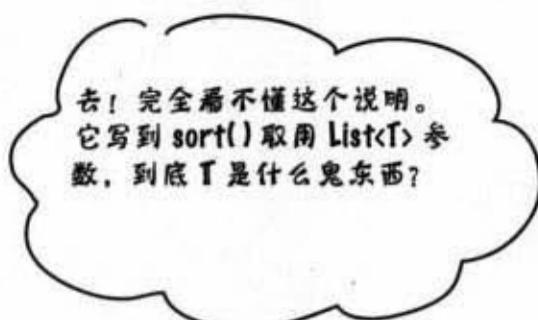
但就是不行！

编译器表示它找不到取用ArrayList<Song>参数的sort()方法，所以ArrayList<String>与ArrayList<Song>之间到底有什么差异？为什么编译器不会让它过关？

```
File Edit Window Help Bummer
%javac Jukebox3.java
Jukebox3.java:15: cannot find symbol
symbol  : method sort(java.util.ArrayList<Song>)
location: class java.util.Collections
        Collections.sort(songList);
                           ^
1 error
```

或许你已经在想：“那它要靠什么东西来排序？”sort()要怎么判断某首歌应该在另外一首歌之前？很明显的，如果你想要让歌曲依照曲名字母排列，就得要有一种方法可以告诉sort()它依靠的不是曲名长度来排列。

接下来我们会有几页进一步的讨论，但首先要解决无法通过编译器的问题。



## sort()的声明

Collections (Java 2 Platform SE 5.0)

file:///Users/kathy/Public/docs/api/index.html

Method Detail

**sort**

```
public static <T extends Comparable<? super T>> void sort(List<T> list)
```

Sorts the specified list into ascending order, according to the *natural ordering* of its elements. All elements in the list must implement the Comparable interface. Furthermore, all elements in the list must be *mutually comparable* (that is, `e1.compareTo(e2)` must not throw a `ClassCastException` for any elements `e1` and `e2` in the list).

从API说明文件找java.util.Collections下面的sort()，你会发现它的声明有点怪怪的。至少跟我们之前所看过的有些不一样。

这是因为sort()很大量地运用到泛型(generic)功能。只要你在Java的程序或文件中看到<>这一组符号，就代表泛型正在作用——它是一种从Java 5.0开始加入的特质。看起来我们得先学会如何解读说明文件才能看得出来为何ArrayList可应付String对象，但不吃Song对象这一套。

## 泛型意味着更好的类型安全性

我们就这么说吧，几乎所有你会以泛型写的程序都与处理集合有关。虽然泛型可以用在其他地方，但它主要目的还是让你能够写出有类型安全性的集合。也就是说，让编译器能够帮忙防止你把Dog加到一群Cat中。

在泛型功能出现前，编译器无法注意到你加入集合中的东西是什么，因为所有的集合都写成处理Object类型。你可以把任何东西放进ArrayList中，有点像是ArrayList<Object>。

### 没有泛型

各种对象以引用的形式加入到ArrayList中。

在泛型出现前，没有办法声明ArrayList的类型，所以只能用Object来操作



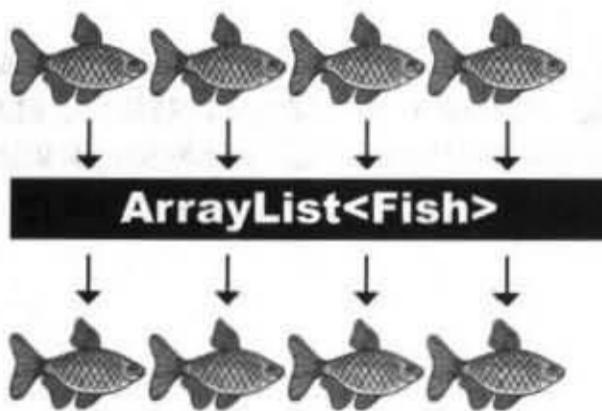
出来时会是Object类型的引用。

运用泛型你就可以创建类型安全更好的集合，让问题尽可能在编译期就能抓到，而不会等到执行期才冒出来。

如果没有泛型，编译器会很愉快地接受你把绵羊对象送到老虎集合中。

### 使用泛型

仅有Fish对象能以引用形式加入到ArrayList中。



出来的还是Fish对象的引用。

加上泛型之后，你不能把非Fish放进ArrayList<Fish>中，因此取出的也会是Fish引用。

## 关于泛型

泛型有好几样东西需要知道，但对大部分的程序员来说，其实只有3件事情是重要的：

### 创建被泛型化类（例如ArrayList）的实例。

创建ArrayList时你必须要指定它所容许的对象，就像单纯的数组那样。

```
new ArrayList<Song>()
```

### 声明与指定泛型类型的变量。

多态遇到泛型类型会怎样？如果你有个ArrayList<Animal>引用变量，能够赋给ArrayList<Dog>吗？如果是List<Animal>呢？可以赋给ArrayList<Animal>吗？稍后分晓……

```
List<Song> songList =  
    new ArrayList<Song>()
```

### 声明（与调用）取用泛型类型的方法。

如果你有个方法取用Animal对象ArrayList的参数，那代表什么？是否也可以传入Dog对象的ArrayList呢？接着我们会讨论到某些多态问题的细节内容。

```
void foo(List<Song> list)  
x.foo(songList)
```

（其实这跟第二项一样，只是让你知道我们认为这有多重要。）

**问：**但我不是还需要学习如何自己创建泛型的类吗？如果我想设计出让人们在初始化同时要决定类型的类要怎么办？

**答：**你或许不会经常做这件事。想想看，API的设计团队已经涵盖了大部分你会遇到的数据结构，而几乎只有集合才会真的需要泛型。也就是说这些类是设计来保存其他元素，并要让程序员在声明与初始化类的时候指定元素的类型。

没错，你可能会想要创建出泛型的类，但那是很少见的情况，所以我们就不多做讨论了（还是可以从这些内容看出大概）。

## 使用泛型的类

因为ArrayList是最常用的泛型化类型，我们会从查看它的文件看起。有两个关键的部分：

- (1) 类的声明。
- (2) 新增元素的方法的声明。

把E想做是“集合所要维护和返回的元素类型”  
(E代表Element)

ArrayList的说明文件（又称“E是什么？”）

```
public class ArrayList<E> extends AbstractList<E> implements List<E> ... {  
    public boolean add(E o)  
    ...  
}
```

E部分会用你所声明与创建的真正类型来取代

ArrayList是AbstractList的子类，所以指定给ArrayList的类型会自动地用在AbstractList上

这边最重要！E用来指示可以加入ArrayList的元素类型

此类型也会用在List这个接口上

E代表用来创建与初始ArrayList的类型。当你看到ArrayList文件上的E时，就可以把它换成实际上的类型。

所以ArrayList<Song>就会在所有方法与变量的声明中把E换成Song。

## ArrayList 的类型参数

下面这行程序：

```
ArrayList<String> thisList = new ArrayList<String>
```

代表这个ArrayList：

```
public class ArrayList<E> extends AbstractList<E> ... {
    public boolean add(E o)
    // 更多代码
}
```

会被编译器这样看待：

```
public class ArrayList<String> extends AbstractList<String>... {
    public boolean add(String o)
    // 更多代码
}
```

也就是说，E会被所指定的真正类型所取代（又被称为类型参数）。这也是为何add()这个方法不会让你加入与E所指定类型不兼容的引用的原因。若你创建出ArrayList<String>，则add()会变成add(String o)。若你创建出ArrayList<Dog>，则add会变成add(Dog o)。

**问：** 只能用E吗？因为排序的文件上面用的是T……

**答：** 你可以使用任何合法的Java标识字符串。这代表不管用什么都会被当作是类型参数。但习惯用法是以单一的字母表示（你也应该这么做），除非与集合有关，否则都是用T，因为E很清楚地指明是元素。

## 运用泛型的方法

泛型的类代表类的声明用到类型参数。泛型的方法代表方法的声明特征用到类型参数。

在方法中的类型参数有几种不同的运用方式。

### ● 使用定义在类声明的类型参数。

```
public class ArrayList<E> extends AbstractList<E> ... {  
    public boolean add(E o) {  
        // 只能在此使用E, 因为它已经被  
        // 定义成类的一部分  
    }  
}
```

当你声明类的类型参数时，你就可以直接把该类或接口类型用在任何地方。参数的类型声明基本上会以用来初始化类的类型来取代。

### ● 使用未定义在类声明的类型参数。

```
public <T extends Animal> void takeThing(ArrayList<T> list)
```

如果类本身没有使用类型参数，你还是可以通过在一个不寻常但可行的位置上指定给方法——在返回类型之前。这个方法意味着T可以是“任何一种Animal”。

因为在前面声明T所以这里  
就可以使用<T>



## 这就是怪异之处……

这行程序：

```
public <T extends Animal> void takeThing(ArrayList<T> list)
```

跟这个是不一样的：

```
public void takeThing(ArrayList<Animal> list)
```

两者都合法，但意义不同！

首先，`<T extends Animal>`是方法声明的一部分，表示任何被声明为Animal或Animal的子型（像是Cat或Dog）的ArrayList是合法的。因此你可以使用ArrayList<Dog>、ArrayList<Cat>或ArrayList<Animal>来调用上面的方法。

但是，下面方法的参数是ArrayList<Animal> list，代表只有ArrayList<Animal>是合法的。也就是说第一个可以使用任何一种Animal的ArrayList，而第二个方法只能使用Animal的ArrayList。

没错，这看起来已经违反动态绑定的精神，但在这一章最后的回顾时就会很清楚这是怎么一回事。现在只要记得我们还在想办法对SongList排序就行。

现在只要知道上面的语法是合法的就够了，它代表你可以传入以Animal或子型来初始化的ArrayList对象。

接着回头看sort()方法……



出错的地方……

```
File Edit Window Help Bummer
% javac Jukebox3.java
Jukebox3.java:15: cannot find symbol
  symbol : method sort(java.util.ArrayList<Song>)
  location: class java.util.Collections
          Collections.sort(songList);
   ^
1 error
```

```
import java.util.*;
import java.io.*;

public class Jukebox3 {
    ArrayList<Song> songList = new ArrayList<Song>();
    public static void main(String[] args) {
        new Jukebox3().go();
    }
    public void go() {
        getSongs();
        System.out.println(songList);
        Collections.sort(songList);
        System.out.println(songList);
    }
    void getSongs() {
        try {
            File file = new File("SongList.txt");
            BufferedReader reader = new BufferedReader(new FileReader(file));
            String line = null;
            while ((line= reader.readLine()) != null) {
                addSong(line);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
    void addSong(String lineToParse) {
        String[] tokens = lineToParse.split("/");
        Song nextSong = new Song(tokens[0], tokens[1], tokens[2], tokens[3]);
        songList.add(nextSong);
    }
}
```

就是这里有问题！传入ArrayList<String>可以  
过关，但ArrayList<Song>就不行

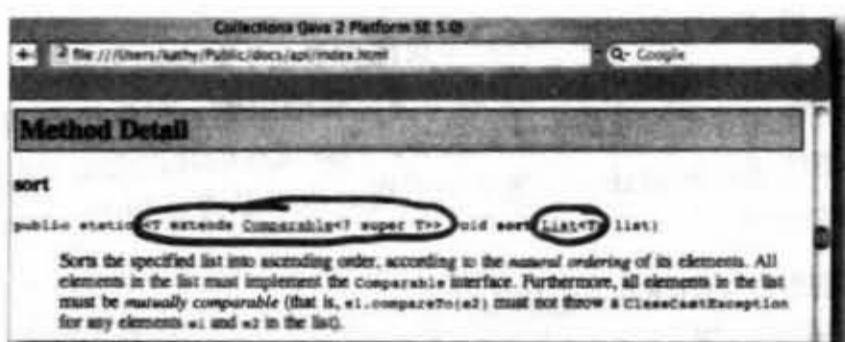
## 再看一下sort()方法

查找文件上关于为何对String的list排序可行，但Song对象就不行的线索。看起来答案应该是：

看起来sort()方法只能接受Comparable对象的list。

Song不是Comparable的子型，所以你无法对Song的list排序。

至少还不行……



public static <T extends Comparable<? super T>> void sort(List<T> list)

这表示它必须是 Comparable

光不管它，它代表Comparable 的类型参数必须是T或T的父型

只能传入继承 Comparable的参数化类型的list

“……我刚查过String的说明，它没有继承过Comparable，只有实现，因为Comparable是个接口，所以上面的extends不太合理。”



public final class String extends Object implements Serializable, Comparable<String>, CharSequence

sort()方法

## 以泛型的观点来说，extend代表 extend或implement

Java设计团队有给你一种对参数化类型加上限制的方法，因此你可以加上只能使用Animal的子类之类的限制。但你也会需要限制只允许有实现某特定接口的类。因此现在的状况需要对两种情形都能适用的语法——继承和实现。也就是说适用于extends和implements。

答案揭晓：extends。它确实代表“是一个……”，且不管是接口或类都能适用。

对泛型来说，extends这个关键词代表“是一个……”，且适用于类和接口。

Comparable是个接口，所以这可以读作：“T必须是有实现Comparable的类型”

↓  
public static <T extends Comparable<? super T>> void sort(List<T> list)  
↑

这是类或接口都没关系，还是可以说这是extends过的

**问：**为什么不弄个“is”关键词来用？

**答：**对语言加入新的关键词是很重大的事件，因为这样可能会有破坏较早写作程序的风险。想想看，假如你有用过名称为is的变量（我们就有用is来代表输入流）会怎样。且因为关键词是不能用来当作标识符，所以本来可以过关的程序也会因为用了保留字而被退回。因此只要有可能的话，Sun的工程师就会像extends这样反复利用现有的关键词。但有时候连他们也没选择的余地……

有少数几个关键词被加入语言中，像是Java 1.4出现的**assert**和Java 5.0的**enum**（见附录）。这确实会破坏某些程序，但有时还是可以对新版Java加上选项让它仿真旧版的行为。通过指定特殊标记给编译器或Java虚拟机就可以让它们如此执行。

（在命令栏上输入javac或java且不加任何指令就会看到可用的选项，讨论部署的章节会有更多的细节。）

知道哪里有问题了……

## Song类必须实现 Comparable

我们只有在Song类实现Comparable的情况下才能把ArrayList<Song>传给sort()方法，因为这个方法就是如此声明的。稍微看过一下说明文件就会知道Comparable其实很单纯，只有一个方法需要实现。

`java.lang.Comparable`

```
public interface Comparable<T> {
    int compareTo(T o);
}
```

而compareTo()的文件说明是这样的：

**Returns:**  
a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

看起来compareTo()方法是会从某个Song的对象来调用，然后传入其他Song对象的引用。执行compareTo()方法的Song必须要判别在排序位置上它自己是高于、低于或相等于所传入的Song。

你的主要任务就是决定如何判断Song的先后，然后以compareTo()方法的实现来反映出这个逻辑。返回负数值表示传入的Song大于执行的Song，正数刚好相反，而返回0代表相等（以排序的目的来说，并不代表两对象真的相等）。

重点在于：两个Song是如何比较出大小的？

要先确定如何比较才能有办法实现Comparable这个接口。

 Sharpen your pencil

写出实现compareTo()方法的程序代码以让Song对象能够依据歌名来排序。

提示：如果你没有搞错的话，大约在3行之内就可以搞定。

## 更新、更好、更comparable的Song类

我们决定要靠歌名来排序，所以把compareTo()方法实现成用执行方法的Song歌曲名称和所传入的Song歌曲名称来比较。也就是说，执行方法的Song会判断它的歌名和参数歌名的比较结果。

我们知道String一定有办法比较字母先后顺序，因为sort()方法就可以比较String的list。我们也知道String有个compareTo()方法，因此只要让曲名String相互比较就好，不必另行编写比较字母的算法！

```

class Song implements Comparable<Song> {
    String title;
    String artist;
    String rating;
    String bpm;

    public int compareTo(Song s) {
        return title.compareTo(s.getTitle());
    }

    Song(String t, String a, String r, String b) {
        title = t;
        artist = a;
        rating = r;
        bpm = b;
    }

    public String getTitle() {
        return title;
    }

    public String getArtist() {
        return artist;
    }

    public String getRating() {
        return rating;
    }

    public String getBpm() {
        return bpm;
    }

    public String toString() {
        return title;
    }
}

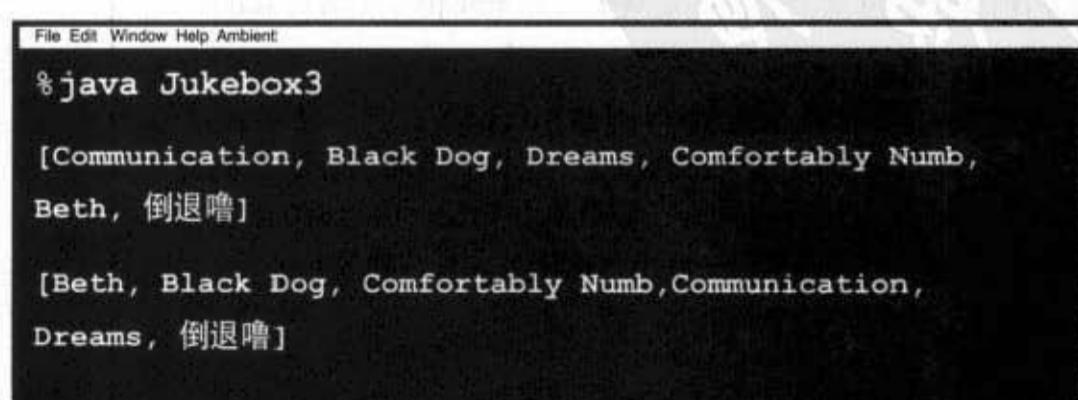
```

通常箭头所指处会是一样的……为了排序的目的，大象只会跟大象比较，而不是跟鸡腿比大。)

要比较的对象

就是这么简单！返回String比较的结果就行

这次成功了，调用sort()方法之后会把Song依照字母作排序



list排好了，但是……

又有问题了，大厨说他只会唱陈雷的歌，所以除了依照歌名排序之外，也要能依照歌星名来排。

但是集合元素的排序就只能实现一个`compartoTo()`，那要怎么办？

有一种糟糕的做法是在Song的类中加上一个旗标，然后在compareTo()中再加上if判断来依照标识决定用哪个项目做比较。

这样很不好，厨房管到外场，简直是造反……事实上还有更好的方法。API中已经设计出一种方法来解决这种问题——以不同方式来比大小。

继续查询API，有另一种sort()方法——它取用Comparator参数。



| Collections (Java 2 Platform SE 5.0)                                                                                                                                                                               |                                                                                                                                                                 |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="file:///Users/kathy/Public/docs/api/index.html">file:///Users/kathy/Public/docs/api/index.html</a>                                                                                                        | <input type="text"/> Google                                                                                                                                     |
| <a href="#">Jellyvision, Inc.</a> <a href="#">Collections</a> <a href="#">...form SE 5.0</a> <a href="#">Caffeinated</a> <a href="#">...d Brain Day</a> <a href="#">Brand Noise</a> <a href="#">Diva Marketing</a> | <a href="#">»</a>                                                                                                                                               |
| <pre>static &lt;K,V&gt; Map&lt;K,V&gt;</pre>                                                                                                                                                                       | <pre>singletonMap(K key, V value)     Returns an immutable map, mapping only the     specified key to the specified value.</pre>                                |
| <pre>static super T&gt; void</pre>                                                                                                                                                                                 | <pre>sort(List&lt;T&gt; list)     Sorts the specified list into ascending order,     according to the <i>natural ordering</i> of its elements.</pre>            |
| <pre>static &lt;T&gt; void</pre>                                                                                                                                                                                   | <pre>sort(List&lt;T&gt; list, Comparator&lt;? super T&gt; c)     Sorts the specified list according to the order     induced by the specified comparator.</pre> |

## 使用自制的Comparator

使用`compareTo()`方法时，list中的元素只能有一种将自己与同类型的另一个元素做比较的方法。但Comparator是独立于所比较元素类型之外的——它是独立的类。因此你可以有各种不同的比较方法！想要依照歌星排序吗？做个`ArtistComparator`，想要依照恶搞程度排序吗？没问题，做个`KusoComparator`。

然后只要调用重载版，取用List与Comparator参数的`sort()`方法来处理就可以。

取用Comparator版的`sort()`方法会用Comparator而不是元素内置的`compareTo()`方法来比较顺序。也就是说，如果`sort()`方法带有Comparator，它就不会调用元素的`compareTo()`方法，而会去调用Comparator的`compare()`方法。

所以规则是这样的：

- u 调用单一参数的`sort(List o)`方法代表由list元素上的`compareTo()`方法来决定顺序。因此元素必须要实现`Comparable`这个接口。
- u 调用`sort(List o, Comparator c)`方法代表不会调用list元素的`compareTo()`方法，而会使用Comparator的`compare()`方法。这意味着list元素不需要实现`Comparable`。

java.util.Comparator

```
public interface Comparator<T> {  
    int compare(T o1, T o2);  
}
```

如果传Comparator给sort()方法，则排序是由Comparator而不是元素的`compareTo()`方法来决定。

**问：**是否这代表如果类没有实现`Comparable`，且你也没拿到原始码，还是能够通过Comparator来排序？

**答：**没错。另外一种方法就是子类化该元素并实现`Comparable`。

**问：**为什么不是每个类都有实现`Comparable`？

**答：**你真的认为每种东西都可以排序吗？如果不能排序的东西硬加上`Comparable`只会使人产生误解。

## 用Comparator更新点歌系统

我们在新版本做了3件事：

- (1) 创建并实现Comparator的内部类，以compare()方法取代compareTo()方法。
- (2) 制作该类的实例。
- (3) 调用重载版的sort()，传入歌曲的list以及Comparator的实例。

附注：我们也有更新Song的toString()以列出歌名与歌星。

```

import java.util.*;
import java.io.*;

public class Jukebox5 {
    ArrayList<Song> songList = new ArrayList<Song>();
    public static void main(String[] args) {
        new Jukebox5().go();
    }

    class ArtistCompare implements Comparator<Song> {
        public int compare(Song one, Song two) {
            return one.getArtist().compareTo(two.getArtist());
        }
    }
}

public void go() {
    getSongs();
    System.out.println(songList);
    Collections.sort(songList);
    System.out.println(songList);

    ArtistCompare artistCompare = new ArtistCompare();
    Collections.sort(songList, artistCompare);    // 调用sort(), 传入list与Comparator对象
    System.out.println(songList);
}

void getSongs() {
    // I/O程序代码
}

void addSong(String lineToParse) {
    // 解析歌曲清单
}

```

创建并实现Comparator的内部类，注意到类型参数和要比较的类型是相符的

这会返回String

by String来比较

创建Comparator的实例

(译注：这就是所谓的垃圾注释：好注释不需要重复程序代码就已经明白表达出的东西。)

注意：另外一种设计哲学是拿掉Song的compareTo()，以两个Comparator来实现歌名排序和歌星名排序。这代表我们只能使用两个参数版的Collections.sort()。

## 习题

```
import _____;
public class SortMountains {
    LinkedList _____ mtn = new LinkedList _____();
    class NameCompare _____ {
        public int compare(Mountain one, Mountain two) {
            return _____;
        }
    }
    class HeightCompare _____ {
        public int compare(Mountain one, Mountain two) {
            return (_____);
        }
    }
    public static void main(String [] args) {
        new SortMountains().go();
    }
    public void go() {
        mtn.add(new Mountain("Longs", 14255));
        mtn.add(new Mountain("Elbert", 14433));
        mtn.add(new Mountain("Maroon", 14156));
        mtn.add(new Mountain("Castle", 14265));
        System.out.println("as entered:\n" + mtn);
        NameCompare nc = new NameCompare();
        _____;
        System.out.println("by name:\n" + mtn);
        HeightCompare hc = new HeightCompare();
        _____;
        System.out.println("by height:\n" + mtn);
    }
}
class Mountain {
    _____;
    _____;
    _____ {
        _____;
        _____;
    }
    _____ {
        _____;
    }
    _____;
}
```



## 逆向工程

假设程序代码都在同一个文件。你的任务是要填满空格让程序产生下面的输出。

答案在本章最后面。

输出：

```
File Edit Window Help ThisOne'sForBob
%java SortMountains
as entered:
[Longs 14255, Elbert 14433, Maroon 14156, Castle 14265]
by name:
[Castle 14265, Elbert 14433, Longs 14255, Maroon 14156]
by height:
[Elbert 14433, Castle 14265, Longs 14255, Maroon 14156]
```



## 填空题

回答下面的问题，答案只能从可用的答案中挑出。标准答案在章末。

可用的答案：

Comparator,  
Comparable,  
compareTo(),  
compare(),  
yes,  
no

对下面这行程序来说：

`Collections.sort(myArrayList);`

- (1) 存储在myArrayList中对象的class一定要实现什么？\_\_\_\_\_
- (2) 存储在myArrayList中对象的class一定要实现什么方法？\_\_\_\_\_
- (3) 存储在myArrayList中对象的 class 能够同时实现Comparator和Comparable吗？\_\_\_\_\_

对下面这行程序来说：

`Collection.sort(myArrayList, myCompare);`

- (4) 存储在myArrayList中对象的class必须实现 Comparable吗？\_\_\_\_\_
- (5) 存储在myArrayList中对象的class必须实现Comparator 吗？\_\_\_\_\_
- (6) 存储在myArrayList中对象的class必须实现Comparable 吗？\_\_\_\_\_
- (7) 存储在myArrayList中对象的class必须实现Comparator 吗？\_\_\_\_\_
- (8) myCompare必须实现什么？\_\_\_\_\_
- (9) myCompare必须实现什么方法？\_\_\_\_\_

## 老爷不好了！数据有重复……

排序功能工作得很好，现在我们知道要如何对歌名和歌星名来排序。但是在测试点歌系统的文本文件时发现到新的问题——排序过的列表带有重复的数据。

这可能是因为点歌系统持续写入文件而不管同样的歌曲是否早就被写入到文本文件中。

`SongListMore.txt` 是完整的点歌记录，所以带有重复的歌曲记录是很正常的，就好像汽车维修员身上带有扳手也是很自然的一样。

```
File Edit Window Help TooManyNotes
%java Jukebox4

[Pink Moon: Nick Drake, Somersault: Zero 7, Shiva Moon: Prem
Joshua, Circles: BT, Deep Channel: Afro Celts, Passenger: Headmix,
Listen: Tahiti 80, Listen: Tahiti 80, Listen: Tahiti 80, Circles:
BT]                                     排序前

[Circles: BT, Circles: BT, Deep Channel: Afro Celts, Listen:
Tahiti 80, Listen: Tahiti 80, Listen: Tahiti 80, Passenger:
Headmix, Pink Moon: Nick Drake, Shiva Moon: Prem Joshua,
Somersault: Zero 7]                      依照歌名排序

[Deep Channel: Afro Celts, Circles: BT, Circles: BT, Passenger:
Headmix, Pink Moon: Nick Drake, Shiva Moon: Prem Joshua, Listen:
Tahiti 80, Listen: Tahiti 80, Listen: Tahiti 80, Somersault: Zero
7]                                     依照歌星名排序
```

### `SongListMore.txt`

```
Pink Moon/Nick Drake/5/80
Somersault/Zero 7/4/84
Shiva Moon/Prem Joshua/6/120
Circles/BT/5/110
Deep Channel/Afro Celts/4/120
Passenger/Headmix/4/100
Listen/Tahiti 80/5/90
Listen/Tahiti 80/5/90
Listen/Tahiti 80/5/90
Circles/BT/5/110
```

这个文件带有重复的歌曲，因为点歌系统会依次地写入记录。有人不停地重复点同一首歌来练，所以就会连续重复的记录。因为要保持记录完整，所以这个部分不能修改，只好改我们自己的程序代码。

## 我们需要 Set 集合

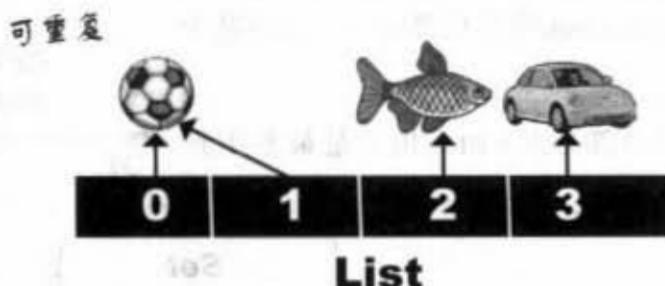
从 Collection 的 API 说明文件中我们发现3个主要的接口：List、Set和Map。

ArrayList 是个 List，但看起来 Set 才是我们所需要的。

### ► LIST：对付顺序的好帮手。

是一种知道索引位置的集合。

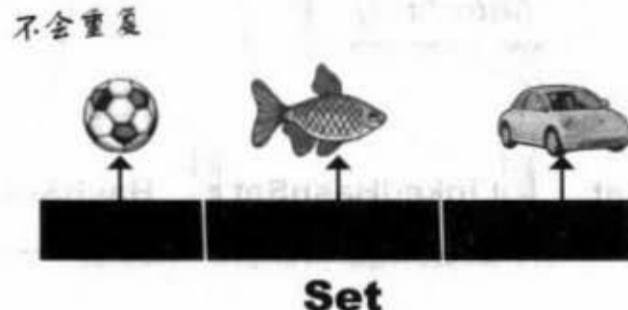
List 知道某物在系列集合中的位置。可以有多个元素引用相同的对象。



### ► SET：注重独一无二的性质。

不允许重复的集合。

它知道某物是否已经存在于集合中。不会有多个元素引用相同的对象（被认为相等的两个对象也不行，稍后有更多的说明）。



### ► MAP：用 key 来搜索的专家。

使用成对的键值和数据值。

Map 会维护与 key 有关联的值。两个 key 可以引用相同的对象，但 key 不能重复，典型的 key 会是 String，但也可以是任何对象。

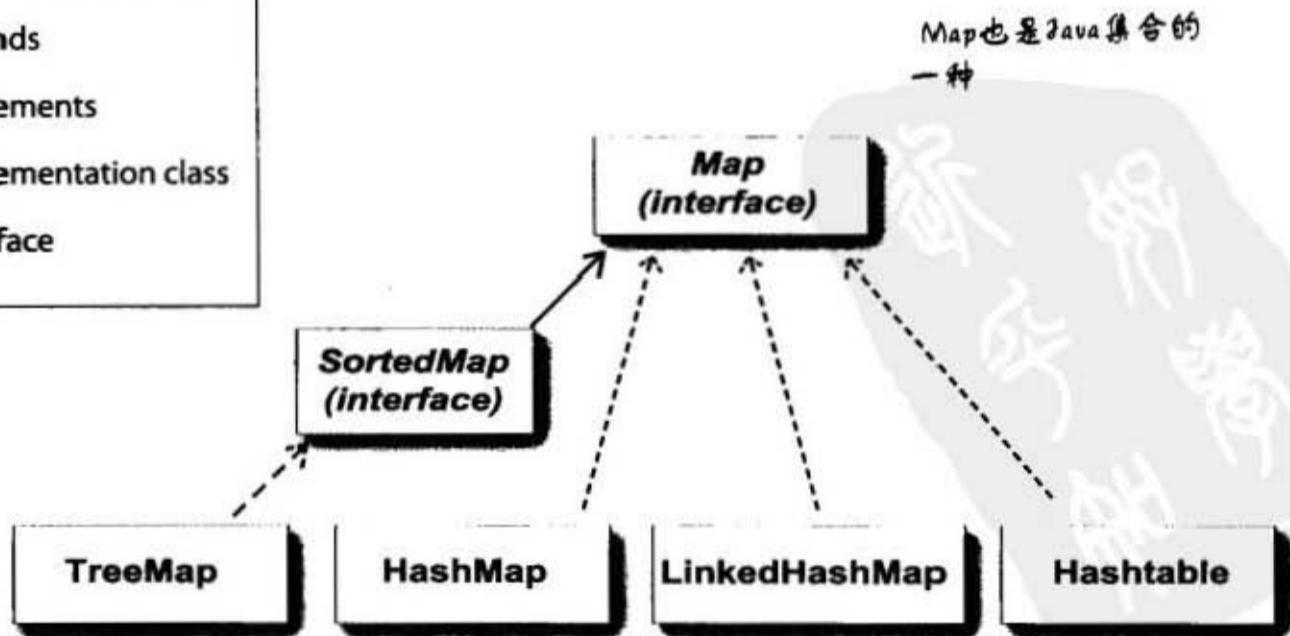
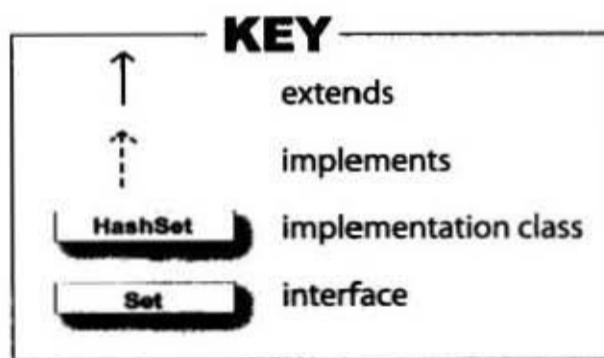
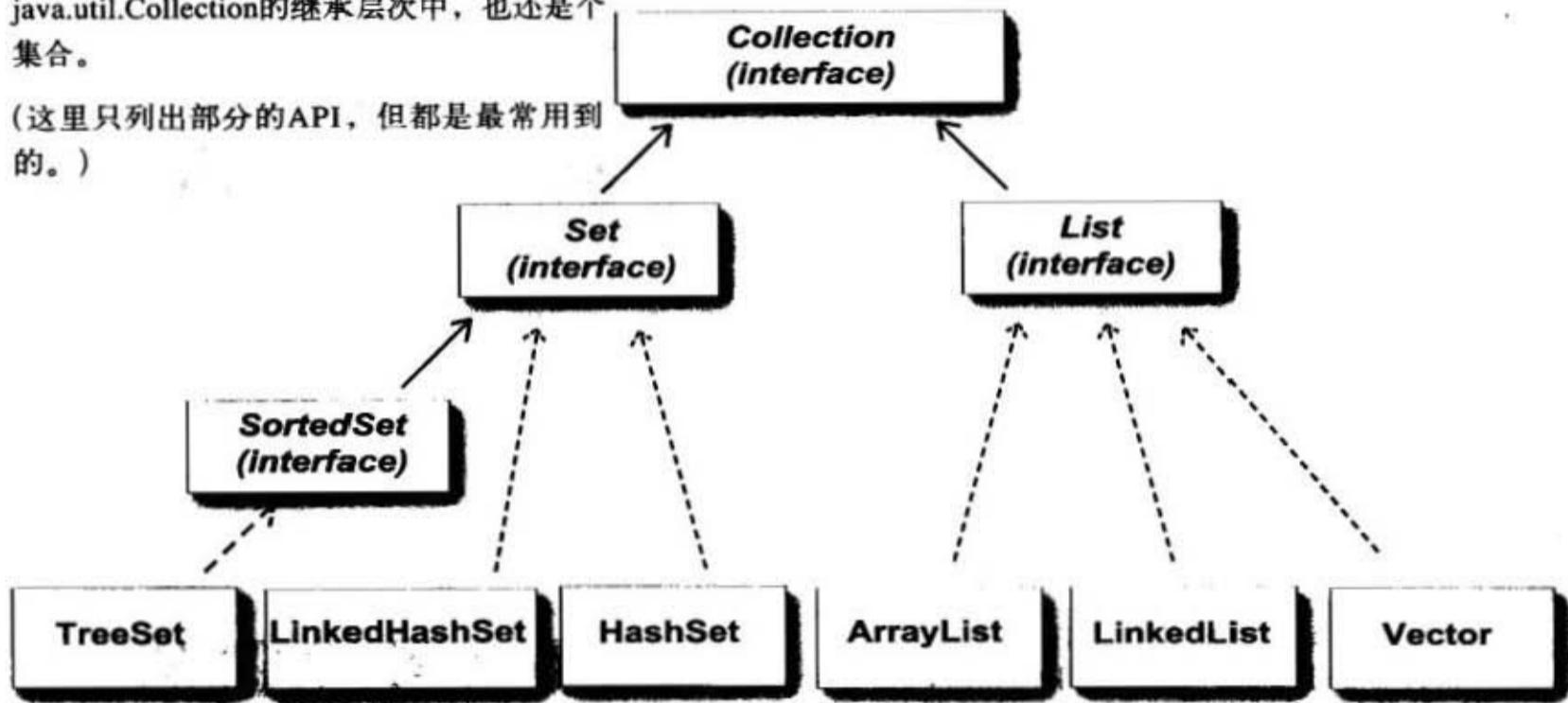
值可以重复，但 key 不行。



## Collection API (摘列)

注意Map事实上并没有继承Collection这个接口，但Map也应该被当作是Collection Framework中的一分子。因此就算它不在java.util.Collection的继承层次中，也还是个集合。

(这里只列出部分的API，但都是最常用到的。)



## 以 HashSet 取代 ArrayList

我们把点歌系统改成使用 HashSet。这里列出一部分的程序代码，其余的部分可以沿用上一版。为了易于阅读，输出的部分直接使用 `toString()`。

```

import java.util.*;
import java.io.*;

public class Jukebox6 {
    ArrayList<Song> songList = new ArrayList<Song>(); ←
    // main method etc.

    public void go() { ←
        getSongs(); ←
        System.out.println(songList); ←
        Collections.sort(songList); ←
        System.out.println(songList); ←

        HashSet<Song> songSet = new HashSet<Song>(); ←
        songSet.addAll(songList); ←
        System.out.println(songSet); ←
    }
    // getSong() and addSong() methods
}

```

```

File Edit Window Help GetBetterMusic
%java Jukebox6
[Pink Moon, Somersault, Shiva Moon, Circles, Deep Channel,
Passenger, Listen, Listen, Listen, Circles] ←
[Circles, Circles, Deep Channel, Listen, Listen, Listen, ←
Passenger, Pink Moon, Shiva Moon, Somersault] ←
[Pink Moon, Listen, Shiva Moon, Circles, Listen, Deep Channel, ←
Passenger, Circles, Listen, Somersault] ←

```

Set还是会重复

已经设定的顺序在Set上消失了

放进 HashSet  
之后列出

## 对象要怎样才算相等？

首先我们得问一个问题：两个Song的引用怎样才算重复？它们必须被认为是相等的。这是说引用到完全相同的对象，还是有相同歌名的不同对象也算？

这带出一个很关键的议题：引用相等性和对象相等性。

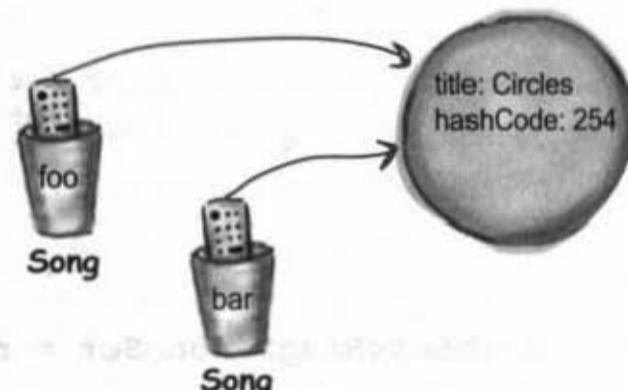
### ▶ 引用相等性

#### 堆上同一对象的两个引用

引用到堆上同一个对象的两个引用是相等的。就这样。如果对两个引用调用hashCode()，你会得到相同的结果。如果没有被覆盖的话，hashCode()默认的行为会返回每个对象特有的序号（大部分的Java版本是依据内存位置计算此序号，所以不会有相同的hashcode）。

如果想要知道两个引用是否相等，可以使用==来比较变量上的字节组合。如果引用到相同的对象，字节组合也会一样。

如果foo与bar两对象相等，则foo.equals(bar)会返回true，且两者的hashCode()也会返回相同的值。要让Set能把对象视为重复的，就必须让它们符合上面的条件被视为是相同的。



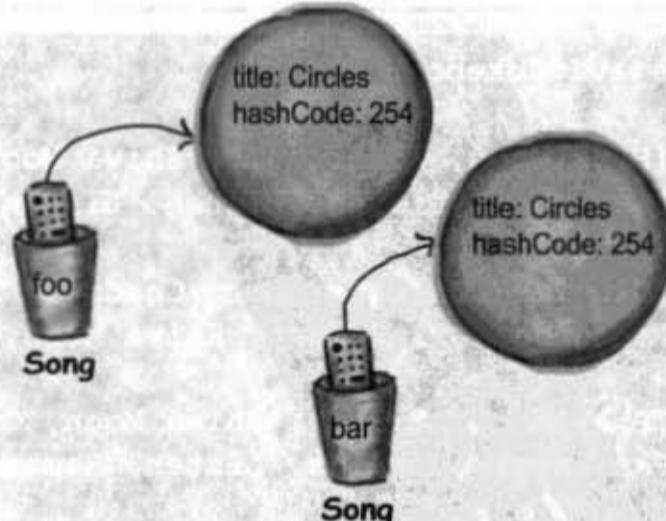
```
if (foo == bar) {  
    // 两个引用都指向同一个对象  
}
```

### ▶ 对象相等性

#### 堆上的两个不同对象在意义上是相同的

如果你想要把两个不同的Song对象视为相等的，就必须覆盖过从Object继承下来的hashCode()方法与equals()方法。

就因为上面所说的内存计算问题，所以你必须覆盖过 hashCode() 才能确保两个对象有相同的hashcode，也要确保以另一个对象为参数的equals()调用会返回true。



```
if (foo.equals(bar) && foo.hashCode() == bar.hashCode()) {  
    // 两个引用指向同一个对象，或者两个对象是相等的  
}
```

## HashSet 如何检查重复： hashCode() 与 equals()

当你把对象加入 HashSet 时，它会使用对象的 hashCode 值来判断对象加入的位置。但同时也会与其他已经加入的对象的 hashCode 作比对，如果没有相符的 hashCode，HashSet 就会假设新对象没有重复出现。

也就是说，如果 hashCode 是相异的，则 HashSet 会假设对象不可能是相同的。

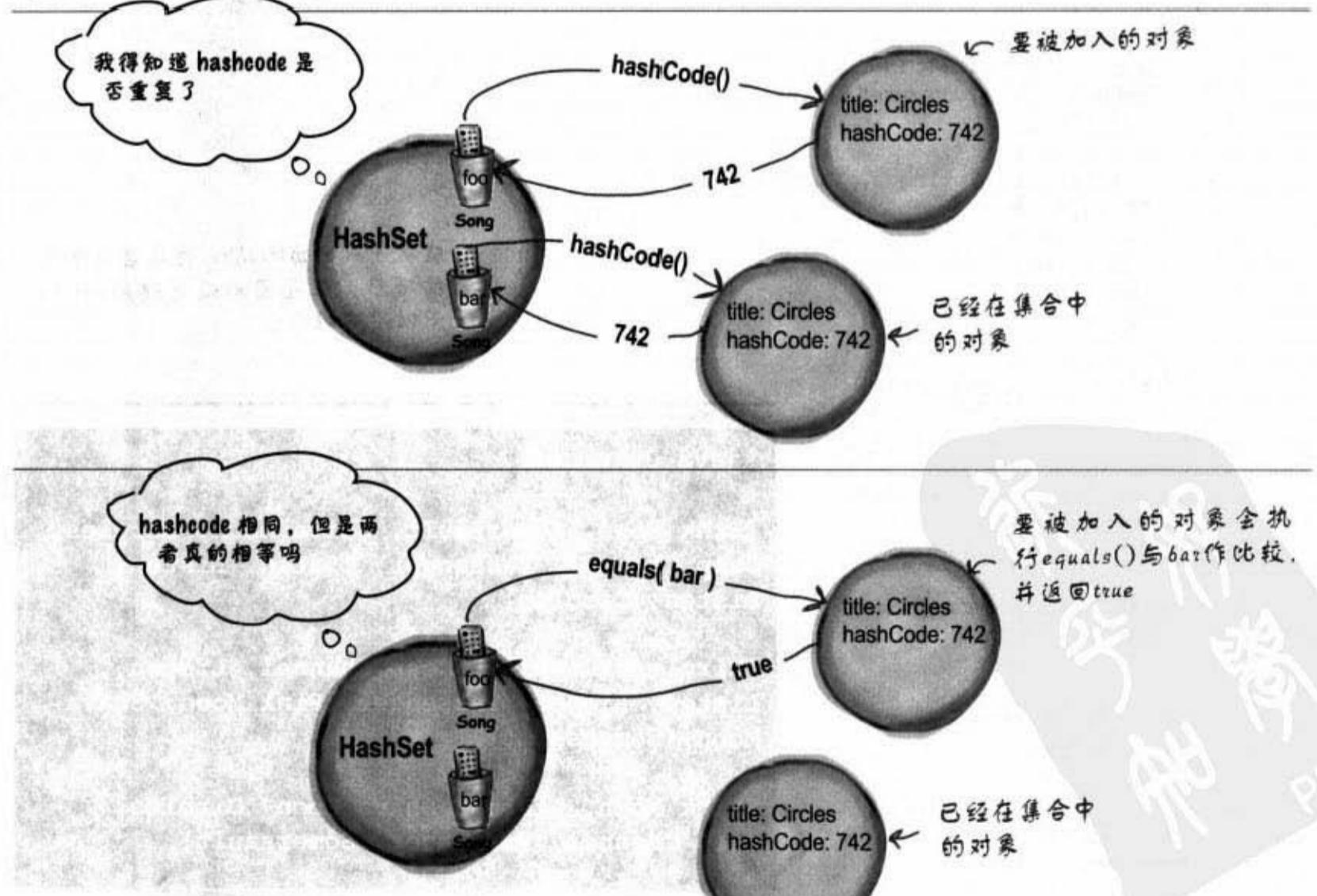
因此你必须 override 过 hashCode() 来确保对象有相同的值。

但有相同 hashCode() 的对象也不一定相等（下一页会

讨论），所以如果 hashCode 找到相同 hashCode 的两个对象：新加入与本来存在的，它会调用其中一个的 equals() 来检查 hashCode 相等的对象是否真的相同。

如果两者相同，HashSet 就会知道要加入的项目已经重复了，所以加入的操作就不会发生。

此时不会传回例外，但 add() 会返回以让你判别对象是否成功的加入。因此若 add() 返回 false，就表示新对象与集合中的某项目被认为是重复的。



hashCode() 与 equals()

## 有覆盖过 hashCode() 与 equals() 的 Song 类

```
class Song implements Comparable<Song> {
    String title;
    String artist;
    String rating;
    String bpm;
}

public boolean equals(Object aSong) {
    Song s = (Song) aSong;
    return getTitle().equals(s.getTitle());
}

public int hashCode() {
    return title.hashCode();
}

public int compareTo(Song s) {
    return title.compareTo(s.getTitle());
}

Song(String t, String a, String r, String b) {
    title = t;
    artist = a;
    rating = r;
    bpm = b;
}

public String getTitle() {
    return title;
}

public String getArtist() {
    return artist;
}

public String getRating() {
    return rating;
}

public String getBpm() {
    return bpm;
}

public String toString() {
    return title;
}
}
```

要被比较的对象

因为歌名是 String, 且 String 本来就覆盖过的 equals(), 所以我们可以调用它

String 也有覆盖过的 hashCode(), 注意到 hashCode() 与 equals() 使用相同的实例变量

成功了! 列出 HashSet 时不会有重复的项目, 但是因为没有调用 sort() 所以没有排序

```
File Edit Window Help RebootWindows
%java Jukebox6
[Pink Moon, Somersault, Shiva Moon, Circles,
Deep Channel, Passenger, Listen, Listen,
Listen, Circles]

[Circles, Circles, Deep Channel, Listen,
Listen, Listen, Passenger, Pink Moon, Shiva
Moon, Somersault]

[Pink Moon, Listen, Shiva Moon, Circles,
Deep Channel, Passenger, Somersault]
```

### hashCode()与equals()的相关规定

API文件有对对象的状态制定出必须遵守的规则：

- (1) 如果两个对象相等，则 hashCode() 必须也是相等的。
- (2) 如果两个对象相等，对其中一个对象调用 equals() 必须返回 true。也就是说，若 a.equals(b) 则 b.equals(a)。
- (3) 如果两个对象有相同的 hashCode 值，它们也不一定是相等的。但若两个对象相等，则 hashCode 值一定是相等的。
- (4) 因此若 equals() 被覆盖过，则 hashCode() 也必须被覆盖。
- (5) hashCode() 的默认行为是对在 heap 上的对象产生独特的值。如果你没有 override 过 hashCode()，则该 class 的两个对象怎样都不会被认为相等的。
- (6) equals() 的默认行为是执行 == 的比较。也就是说会去测试两个引用是否对上 heap 上同一个对象。如果 equals() 没有被覆盖过，两个对象永远都不会被视为相同的，因为不同的对象有不同的字节组合。

a.equals(b) 必须与

a.hashCode() == b.hashCode() 等值。

但 a.hashCode() == b.hashCode()

不一定要与 a.equals() 等值。

### there are no Dumb Questions

**问：**为什么不同对象会有相同 hashCode 的可能？

**答：** HashSet 使用 hashCode 来达成存取速度较快的存储方法。如果你尝试用对象来寻找 ArrayList 中相同的对象（也就是不用索引来找），ArrayList 会从头开始找起。但 HashSet 这样找对象的速度就快多了，因为它使用 hashCode 来寻找符合条件的元素。因此当你想要寻找某个对象时，通过 hashCode 就可以很快地算出该对象所在的位置，而不必从头一个一个找起。

数据结构的课程绝对会告诉你更完整的理论，但这样说明也能让你知道如何有效率地运用 HashSet。事实上，如何开发出有效率的杂凑算法一直都是博士论文的热门题目。

重点在于 hashCode 相同并不一定保证对象是相等的，因为 hashCode() 所使用的杂凑算法也许刚好会让多个对象返回相同的杂凑值。越糟糕的杂凑算法越容易碰撞，但这也与数据值域分布的特性有关。总而言之，把数据结构这科目搞定就对了。

如果 HashSet 发现在比对的时候，同样的 hashCode 有多个对象，它会使用 equals() 来判断是否有完全的符合。也就是说， hashCode 是用来缩小寻找成本，但最后还是要用 equals() 才能认定是否真的找到相同的项目。

## 如果想要保持有序，使用TreeSet

TreeSet在防止重复上面与HashSet是一样的。但它还会一直保持集合处于有序状态。如果使用TreeSet默认的构造函数，它工作起来就会像sort()一样使用对象的compareTo()方法来排序。但也可以选择传入Comparator给TreeSet的构造函数。缺点是如果不需要排序时就会浪费处理能力。但你可能会发现这点损耗实在很小。

```

import java.util.*;
import java.io.*;
public class Jukebox8 {
    ArrayList<Song> songList = new ArrayList<Song>();
    int val;

    public static void main(String[] args) {
        new Jukebox8().go();
    }

    public void go() {
        getSong();
        System.out.println(songList);
        Collections.sort(songList);
        System.out.println(songList);
        TreeSet<Song> songSet = new TreeSet<Song>();
        songSet.addAll(songList); ←
        System.out.println(songSet);    使用addAll()可以把对象全部加入
    }

    void getSong() {
        try {
            File file = new File("SongListMore.txt");
            BufferedReader reader = new BufferedReader(new FileReader(file));
            String line = null;
            while ((line = reader.readLine()) != null) {
                addSong(line);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    void addSong(String lineToParse) {
        String[] tokens = lineToParse.split("/");
        Song nextSong = new Song(tokens[0], tokens[1], tokens[2], tokens[3]);
        songList.add(nextSong);
    }
}

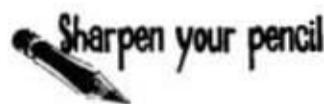
```

调用没有参数的构造函数来用  
TreeSet取代HashSet意味着以对象的  
compareTo()方法来进行排序

使用addAll()可以把对象全部加入

## 使用 TreeSet 一定要知道的事情……

它很容易使用，但你得先确定已经知道所有必须了解的事情。我们认为它太重要了，所以要让你通过习题来思考一下。先别翻到下一页，说真的！



仔细阅读这份程序代码，然后回答下面的问题（程序没有语法错误）。

```
import java.util.*;

public class TestTree {
    public static void main (String[] args) {
        new TestTree().go();
    }

    public void go() {
        Book b1 = new Book("How Cats Work");
        Book b2 = new Book("Remix your Body");
        Book b3 = new Book("Finding Emo");

        TreeSet<Book> tree = new TreeSet<Book>();
        tree.add(b1);
        tree.add(b2);
        tree.add(b3);
        System.out.println(tree);
    }
}

class Book {
    String title;
    public Book(String t) {
        title = t;
    }
}
```

(1) 编译会有什么结果？

---

(2) 如果能编译，执行TestTree会有什么结果？

---

(3) 如果有问题，要怎么更正？

---

TreeSets是如何排序的

## TreeSet的元素必须是Comparable

TreeSet无法猜到程序员的想法，你必须指出对象应该如何排序。

要使用TreeSet，下列其中一项必须为真。

- ▶ 集合中的元素必须是有实现Comparable的类型。

上一页的Book并没有实现Comparable，所以执行时会有问题。想想看，可怜的TreeSet唯一的目的就是要保持元素有序，但是它根本不知道要如何排列Book对象！这在编译时不会有问題，因为TreeSet的add()并未要求取用Comparable类型的参数。如果你以TreeSet<Book>来创建，基本上add()只会要求元素得是Book，并没有检查Book是否有实现出Comparable！但加入第二个元素时，会因为无法调用对象的compareTo()而失败。

```
class Book implements Comparable {  
    String title;  
    public Book(String t) {  
        title = t;  
    }  
    public int compareTo(Object b) {  
        Book book = (Book) b;  
        return (title.compareTo(book.title));  
    }  
}
```

或

- ▶ 使用重载、取用Comparator参数的构造函数来创建TreeSet。

就像sort()，你可以选择使用元素的compareTo()，假设元素的类型都有实现Comparable，或者自定义Comparator。要使用自订的Comparator，你可以调用取用Comparator的构造函数。

```
public class BookCompare implements Comparator<Book> {  
    public int compare(Book one, Book two) {  
        return (one.title.compareTo(two.title));  
    }  
}  
  
class Test {  
    public void go() {  
        Book b1 = new Book("How Cats Work");  
        Book b2 = new Book("Remix your Body");  
        Book b3 = new Book("Finding Emo");  
        BookCompare bCompare = new BookCompare();  
        TreeSet<Book> tree = new TreeSet<Book>(bCompare);  
        tree.add(new Book("How Cats Work"));  
        tree.add(new Book("Finding Emo"));  
        tree.add(new Book("Remix your Body"));  
        System.out.println(tree);  
    }  
}
```

## 现在来看 Map

List和Set很好用，但有时Map才是最好的选择。

想象一下如果你需要用名称来取得值的情况。虽然key通常都是String，但也可以用任何Java的对象（或通过autoboxing的primitive）。



### Map example

```
import java.util.*;
public class TestMap {
    public static void main(String[] args) {
        HashMap<String, Integer> scores = new HashMap<String, Integer>();
        scores.put("Kathy", 42);
        scores.put("Bert", 343);
        scores.put("Skyler", 420);
        System.out.println(scores);
        System.out.println(scores.get("Bert"));
    }
}
```

HashMap 需要两个类型参数：关键字和值

使用 put() 取代 add()，它需要两个参数

get() 取用关键字参数，返回它的值

```
File Edit Window Help WhereAmI
%java TestMap
{Skyler=420, Bert=343, Kathy=42}
343
```

当你列出 Map 时，它会以 `key=value` 的形式打印出，外面用 {} 包起来

## 终于回到泛型

还记得本章较早之前讨论到取用泛型参数的方法会有多古怪吗？这是以多态的观点来说的。如果越看越觉得奇怪，就继续看下去，整个故事要好几页才说得完。

我们会先回顾一下数组参数是如何多态化运行的，然后再看同样的工作如何以泛型集合来完成。下面的程序代码在编译与执行时都不会有问题。

普通数组工作的方式：

```
import java.util.*;

public class TestGenerics1 {
    public static void main(String[] args) {
        new TestGenerics1().go();
    }

    public void go() {
        Animal[] animals = {new Dog(), new Cat(), new Dog()};
        Dog[] dogs = {new Dog(), new Dog(), new Dog()};
        takeAnimals(animals);
        takeAnimals(dogs);
    }
}

public void takeAnimals(Animal[] animals) {
    for(Animal a: animals) {
        a.eat();
    }
}

记得吗？我们只能调用声明在Animal中的方法，因为它的参数是Animal数组，且无需任何类型转换（怎么转换？数组可能同时带有Dog与Cat）
```

声明并创建Animal数组，它带有Dog与Cat

↑ 声明并创建Dog数组，仅能带Dog

↑ 对两种数组调用

↑ takeAnimal()

重点在于takeAnimal()能够取用Animal[]或Dog[]参数，因为Dog是一个Animal，多态在此发挥作用

```
abstract class Animal {
    void eat() {
        System.out.println("animal eating");
    }
}
class Dog extends Animal {
    void bark() { }
}
class Cat extends Animal {
    void meow() { }
}
```

简单版的Animal继承层次

如果方法的参数是Animal的数组，它也能够取用Animal次类型的数组。

也就是说，如果方法是这样声明的：

```
void foo(Animal[] a) { }
```

若Dog有extend过Animal，你就可以用下列的两种方式调用：

```
foo(anAnimalArray);
foo(aDogArray);
```

## 使用多态参数与泛型

我们已经看过数组的整个工作状况，但当我们把array换成ArrayList时还会一样吗？听起来很合理，不是吗？

先来看只有Animal的ArrayList的状况。我们仅对go()作修改。

只传入ArrayList<Animal>

从Animal[ ]改成ArrayList<Animal>

```
public void go() {
    ArrayList<Animal> animals = new ArrayList<Animal>();
    animals.add(new Dog());
    animals.add(new Cat()); ← 只能一个一个加进去，没有像
    animals.add(new Dog()); 数组的那种语法可用

    takeAnimals(animals); ← 除了animals这个变量引用ArrayList而不是
}                               数组之外，程序代码都相同

public void takeAnimals(ArrayList<Animal> animals) {
    for(Animal a: animals) {    此方法改成取用ArrayList，其余都
        a.eat();                一样，for循环可以操作集合与数组
    }
}
```

可编写可运行，没有问题

```
File Edit Window Help CatFoodIsBetter
%java TestGenerics2

animal eating
animal eating
animal eating
animal eating
animal eating
animal eating
```

## 但 `ArrayList<Dog>` 可用吗？

因为多态的关系，编译器会让 `Dog` 数组通过取用 `Animal` 数组参数的方法。没问题，且 `ArrayList<Animal>` 也可以通过取用 `ArrayList<Animal>` 的方法。问题是，`ArrayList<Animal>` 参数能够接受 `ArrayList<Dog>` 吗？如果在数组上是可以的，这应该要也可以吧？

只传入 `ArrayList<Dog>`

```
public void go() {
    ArrayList<Animal> animals = new ArrayList<Animal>();
    animals.add(new Dog());
    animals.add(new Cat());
    animals.add(new Dog());
    takeAnimals(animals); ← 我们知道这里不会有问题是
}

ArrayList<Dog> dogs = new ArrayList<Dog>();
dogs.add(new Dog());
dogs.add(new Dog());           创建Dog的ArrayList
takeAnimals(dogs); ← 改成ArrayList之后还会过关吗?
}
```

```
public void takeAnimals(ArrayList<Animal> animals) {
    for(Animal a: animals) {
        a.eat();
    }
}
```

编译时：

```
File Edit Window Help CatsAreSmarter
%java TestGenerics3

TestGenerics3.java:21: takeAnimals(java.util.
ArrayList<Animal>) in TestGenerics3 cannot be applied to
(java.util.ArrayList<Dog>)
    takeAnimals(dogs);
    ^
1 error
```

想不到啊，看起来都  
没有问题……



## 如果可以会怎样？

假设编译器让你过关。它允许对下面这个方法传入ArrayList<Dog>作为参数：

```
public void takeAnimals(ArrayList<Animal> animals) {
    for(Animal a: animals) {
        a.eat();
    }
}
```

程序看起来不太像是会有问题的样子，对吧？毕竟多态的意义在于Animal可以做的事情（此例中的eat()），Dog也都能做。所以对Dog引用调用eat()会出什么状况呢？

不会，当然不会。

上面的程序代码没问题，但下面这段程序代码：

```
public void takeAnimals(ArrayList<Animal> animals) {
    animals.add(new Cat()); ← 加入一只猫
}
```

这就会是问题了。理论上把Cat加到ArrayList<Animal>是合法的，这也是使用Animal的本意——让各种Animal都可以加入到此ArrayList中。

但如果你传入一个Dog的ArrayList给该方法，那么结果就是一个被Cat给混进去的Dog集合。原本不可能出现Cat的ArrayList<Dog>在这种情况下还是出问题了，这就是为什么编译器不会让它通过的原因。

**如果把方法声明成取用ArrayList<Animal>，它就只会取用ArrayList<Animal>参数，ArrayList<Dog>与ArrayList<Cat>都不行。**



等一下，如果这就是为什么不能过关的原因，为何数组就可以？同样的问题不也会发生在数组上吗？难道你不能把Cat加到Dog[]中？

数组的类型是在运行期间检查的，但集合的类型检查只会发生在编译期间

假设你对Dog[]加入一个Cat（把Dog[]传入声明为Animal[]的参数中是完全合法的操作）

```
public void go() {  
    Dog[] dogs = {new Dog(), new Dog(), new Dog()};  
    takeAnimals(dogs);  
}  
  
public void takeAnimals(Animal[] animals) {  
    animals[0] = new Cat();  
}
```

把Cat放到Dog数组中，这能  
骗过编译器

可以编译，但运行的时候：

哇！被Java虚拟机发现了！

```
File Edit Window Help CatsAreSmarter  
%java TestGenerics1  
Exception in thread "main" java.lang.ArrayStoreException:  
Cat  
    at TestGenerics1.takeAnimals(TestGenerics1.java:16)  
    at TestGenerics1.go(TestGenerics1.java:12)  
    at TestGenerics1.main(TestGenerics1.java:5)
```



## 万用字符

这听起来很怪，但却还有一种能够创建出接受Animal子型参数的方法，就是使用万用字符（wildcard），这是它被加入Java语言中的原因。

```
public void takeAnimals(ArrayList<? extends Animal> animals) {  
    for(Animal a: animals) {  
        a.eat();  
    }  
}
```



要记住此处的**extends**同时代表继承和实现。如果要取用有实现Pet这个接口类型的ArrayList，也是这样声明：

```
ArrayList<? extends Pet>
```

你可能会怀疑“有什么差别？问题还不是一样？Dog群还不是有可能被加入Cat”。

你的疑问没错，但实际上在使用带有<?>的声明时，编译器不会让你加入任何东西到集合中！

在方法参数中使用万用字符时，编译器会阻止任何可能破坏引用参数所指集合的行为。

你能够调用list中任何元素的方法，但不能加入元素。

也就是说，你可以操作集合元素，但不能新增集合元素。如此才能保障执行期间的安全性，因为编译器会阻止执行期的恐怖行动。

所以下面这个程序是可以的：

```
for(Animal a: animals) {  
    a.eat();  
}
```

但这个就过不了编译：

```
animals.add(new Cat());
```

## 相同功能的另一种语法

你或许还记得当讨论到sort()方法时，它使用到泛型类型，但以一个不寻常的格式在返回类型前声明类型参数。这是另外一种类型参数的声明方式，但结果是一样的。

这一行：

```
public <T extends Animal> void takeThing(ArrayList<T> list)
```

跟这一行执行一样：

```
public void takeThing(ArrayList<? extends Animal> list)
```

there are no  
Dumb Questions

**问：** 如果都一样，为什么要用有问号的那个？

**答：** 这要看你是否会使用到T来决定。举例来说，如果方法有两个参数——都是继承Animal的集合会怎样？此时，只声明一次会比较有效率。

```
public <T extends Animal> void takeThing(ArrayList<T> one, ArrayList<T> two)
```

而不必这样：

```
public void takeThing(ArrayList<? extends Animal> one,
                      ArrayList<? extends Animal> two)
```



第16章 Java泛型

## 我是编译器

你的任务是扮演编译器角色并判断下列哪些程序是可以通过编译的。有些程序代码并没有被讨论过，所以要根据你已知的规则来回答。也有可能需要用猜的，但要猜得有根据。



可以通过编译吗？

- `ArrayList<Dog> dogs1 = new ArrayList<Animal>();`
- `ArrayList<Animal> animals1 = new ArrayList<Dog>();`
- `List<Animal> list = new ArrayList<Animal>();`
- `ArrayList<Dog> dogs = new ArrayList<Dog>();`
- `ArrayList<Animal> animals = dogs;`
- `List<Dog> dogList = dogs;`
- `ArrayList<Object> objects = new ArrayList<Object>();`
- `List<Object> objList = objects;`
- `ArrayList<Object> objs = new ArrayList<Dog>();`

## 逆向工程解答

```

import java.util.*;
public class SortMountains {
    LinkedList<Mountain> mtn = new LinkedList<Mountain>();
    class NameCompare implements Comparator<Mountain> {
        public int compare(Mountain one, Mountain two) {
            return one.name.compareTo(two.name);
        }
    }
    class HeightCompare implements Comparator<Mountain> {
        public int compare(Mountain one, Mountain two) {
            return (two.height - one.height);
        }
    }
    public static void main(String [] args) {
        new SortMountain().go();
    }
    public void go() {
        mtn.add(new Mountain("Longs", 14255));
        mtn.add(new Mountain("Elbert", 14433));
        mtn.add(new Mountain("Maroon", 14156));
        mtn.add(new Mountain("Castle", 14265));
        System.out.println("as entered:\n" + mtn);
        NameCompare nc = new NameCompare();
        Collections.sort(mtn, nc);
        System.out.println("by name:\n" + mtn);
        HeightCompare hc = new HeightCompare();
        Collections.sort(mtn, hc);
        System.out.println("by height:\n" + mtn);
    }
}
class Mountain {
    String name;
    int height;
    Mountain(String n, int h) {
        name = n;
        height = h;
    }
    public String toString() {
        return name + " " + height;
    }
}

```

你注意到这是降幂排序吗？

输出：

```

File Edit Window Help ThisOne'sForBob
%java SortMountains
as entered:
[Longs 14255, Elbert 14433, Maroon 14156, Castle 14265]
by name:
[Castle 14265, Elbert 14433, Longs 14255, Maroon 14156]
by height:
[Elbert 14433, Castle 14265, Longs 14255, Maroon 14156]

```

## 填空题解答

可用的答案：

Comparator,

Comparable,

compareTo( ),

compare( ),

yes,

no

对下面这行程序来说：

```
Collections.sort(myArrayList);
```

- (1) 存储在myArrayList中对象的类一定要实现什么？
- (2) 存储在myArrayList中对象的类一定要实现什么方法？
- (3) 存储在myArrayList中对象的类能够同时实现Comparator与Comparable吗？

Comparable

compareTo()

yes

对下面这行程序来说：

```
Collection.sort(myArrayList, myCompare);
```

- (4) 存储在myArrayList中对象的类能够实现Comparable吗？
- (5) 存储在myArrayList中对象的类能够实现Comparator吗？
- (6) 存储在myArrayList中对象的类必须实现Comparable吗？
- (7) 存储在myArrayList中对象的类必须实现Comparator吗？
- (8) myCompare对象的类必须实现什么？
- (9) myCompare对象的类必须实现什么方法？

yes

yes

no

no

Comparator

compare()



可以通过编译吗？

- `ArrayList<Dog> dogs1 = new ArrayList<Animal>();`
- `ArrayList<Animal> animals1 = new ArrayList<Dog>();`
- ☒ `List<Animal> list = new ArrayList<Animal>();`
- ☒ `ArrayList<Dog> dogs = new ArrayList<Dog>();`
- `ArrayList<Animal> animals = dogs;`
- ☒ `List<Dog> dogList = dogs;`
- ☒ `ArrayList<Object> objects = new ArrayList<Object>();`
- ☒ `List<Object> objList = objects;`
- `ArrayList<Object> objs = new ArrayList<Dog>();`

java学习群：72030155，每天20:30-23:00都有大神视频教学，想学习的同学可以进群免费听课！

## 17 包、jar存档文件和部署

# 发布程序



**该是放手的时候了。**你写程序、测试、修改。你告诉每个人你想要转行去开出租车。最后，你还是写出可以出国比赛的好程序——它居然可以运行！那现在呢？如何交到用户的手上？到底要给客户什么东西？如果不知道谁会用呢？最后这两章会来讨论如何组织、包装与部署Java程序。我们会触及包括可执行的jar、Java Web Start、RMI与Servlets等本机、半本机与远程部署的选项。这一章大部分的内容会叙述程序代码的组织与包装——不管最后标的是什么都知道的事情。放心，发布程序并不是分手，而是没完没了的维护工作的开始……

## 部署应用程序

到底什么才是Java应用程序？也就是说开发完成之后，要交付的项目是什么？用户的系统很可能跟你的不一样。更重要的是，他们并没有拿到应用程序。所以现在该把你的程序塑造成可部署给外人使用的形式。在这一章中，我们会讨论本机部署，包括Executable Jar与称为Java Web Start的半本机半远程技术。下一章会讨论较远程的部署选择，包括RMI和Servlet在内。

### 部署的选择



完全在本机

介于两者之间

完全在远程

#### ① 本机。

整个程序都在用户的计算机上以独立、可携的GUI执行，并以可执行的Jar来部署（稍后讨论JAR）。

#### ② 两者之间的组合。

应用程序被分散成在用户本地系统运行的客户端，连接到执行应用程序服务的服务器部分。

#### ③ 远程。

整个应用程序都在服务器端执行，客户端通过非Java形式、可能是浏览器的装置来存取。

但在我们开始进入关于部署的内容之前，先回头来看看当你完成程序时，你只把类文件交给用户会发生什么事。工作目录中到底有什么东西？

Java程序是由一组类所组成。那就是开发过程的输出。

真正的问题是完成之后要拿这些类怎么办？



把应用程序部署成在客户端计算机上独立执行的程序有什么好处和坏处？

把应用程序部署成在独立在服务器上执行的Web程序，并让用户通过浏览器交互有什么好处和坏处？



## 想象这个场景……

大傻正在愉快地进行Java程序最后的部分，经历好几个礼拜“就快写好了”的状态，这次真地就要写完了。这只程序是相当复杂的GUI程序，但因为有Swing的帮助，他实际上只需要写出9个类。

最后，交付给客户的时间到了。他发现只要把那9个类文件拷贝给客户就行，因为客户端已经安装好了Java API。他切换到放文件的目录开始ls……

哇……

啊！目录下面不只有18个文件（9个源代码、9个编译出的类），实际上有31个文件，好几个文件的名字怪怪的。

Account\$FileListener.class

Char\$SaveListener.class

类似这样。他完全忘记编译器还必须对内部类的GUI事件监听程序产生出类文件，就是那些怪怪的文件。

现在他小心地把文件抽出来，少了一个文件程序就不能运行。但他又不想不小心把源代码交给客户，并且整个目录看起来也很乱。

## 将源代码与类文件分离

带有一堆源代码和类文件的目录是一团混乱的。大傻应该要好好的整理一下文件，让源代码与编译出的文件分开。也就是说，确保编译过的类文件不会放在源代码的目录中。

关键在于结合 `-d` 这个编译选项和目录组织的结构。

有好几种方法可以组织文件，你们公司可能有规定要怎么做。然而我们会建议一种几乎已成为标准的组织化纲要。

使用这种纲要时，你会创建出项目目录，下面有 `source` 和 `classes` 目录。把源代码 (`.java`) 存储在 `source` 目录下。在编译时动点手脚让输出 (`.class`) 产生在 `classes` 目录。

有个编译器选项能够这么搞。

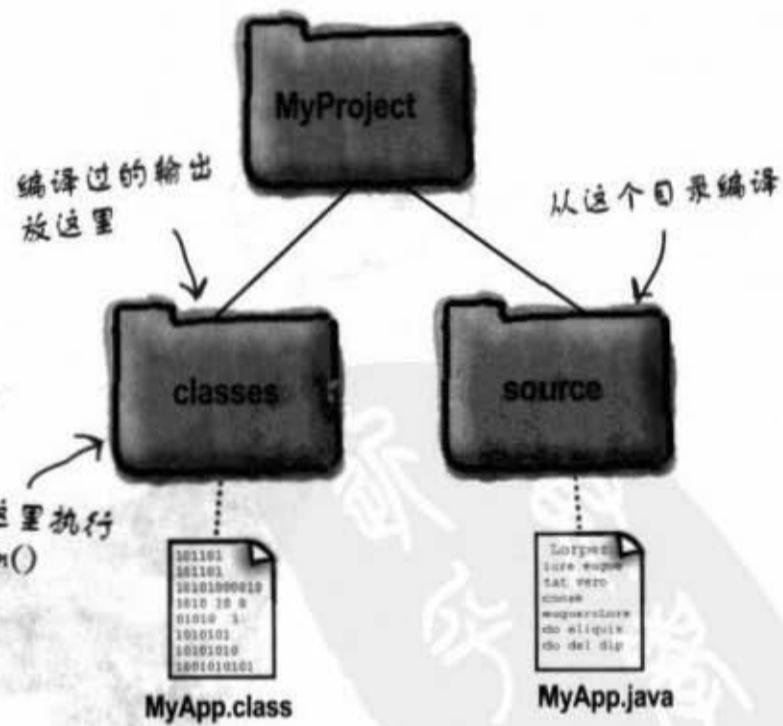
编译时加上 `-d` (directory) 选项。



```
%cd MyProject/source
%javac -d ../classes MyApp.java
```

↑  
要求编译器将编译输出放到目前目录上一层下面的  
`classes` 目录

↑  
这里放的还是需要被  
编译的文件



使用 `-d` 选项，你就可以指定编译过的程序要摆在哪里，而不会放到默认的同一个目录下。若要编译全部的 `.java` 文件：

```
%javac -d ../classes *.java
```

↑  
代表目前目录所有  
的源文件

执行程序

(除错信息：这一章假设目前目录是在 `classpath` 中，如果你修改过环境变量，要确定 “.” 包含在 `classpath` 中。)

## 把程序包进 JAR



JAR就是Java ARchive。这种文件是个pkzip格式的文件，它能让你把一组类文件包装起来，所以交付时只需要一个JAR文件。如果你很熟悉UNIX上的tar命令的话，你就会知道jar这个工具要怎么使用（注意：当我们提到全大写的 JAR 时是说集合起来的文件，全小写的jar是用来整理文件的工具）。

问题是用户要拿JAR怎么办？

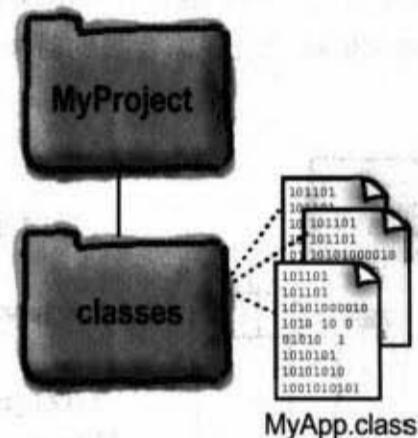
你会创建出可执行的JAR。

可执行的JAR代表用户不需把文件抽出来就能运行。程序可以在类文件保存在JAR的情况下执行。秘诀在于创建出manifest文件，它会带有JAR的信息，告诉Java虚拟机哪个类含有main()这个方法！

### 创建可执行的JAR

#### ① 确定所有的类文件都在classes目录下

稍后还有进一步的讨论，但现在先把所有的类文件放在classes目录下。

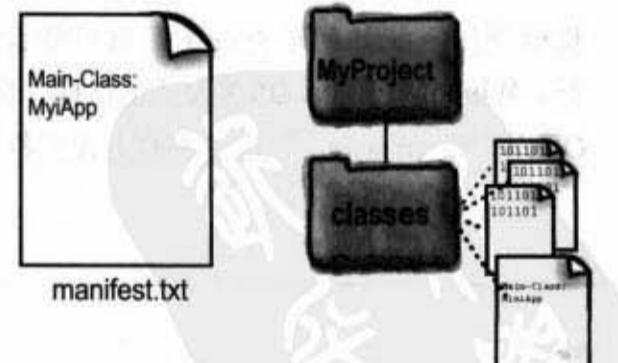


#### ② 创建manifest.txt来描述哪个类带有main()方法

该文件带有下面这一行：

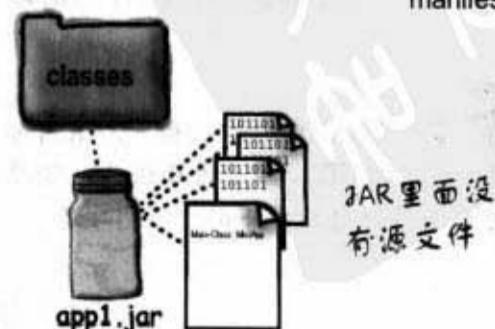
Main-Class: MyApp ← 后面没有.class

在此行后面要有换行，否则有可能出错。将此文件放在classes目录下。



#### ③ 执行jar工具来创建带有所有类以及manifest的JAR文件

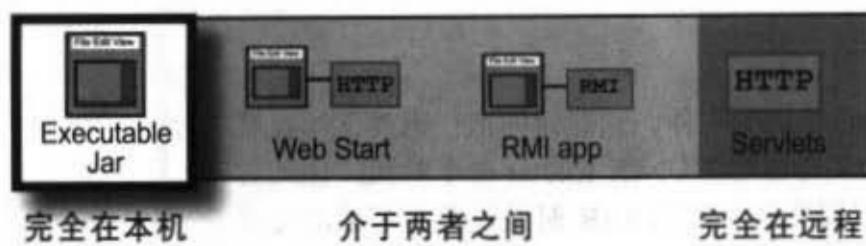
```
%cd MiniProject/classes
%jar -cvmf manifest.txt appl.jar *.class
或
%jar -cvmf manifest.txt appl.jar MyApp.class
```



你现在的位置

585

## 可执行的JAR

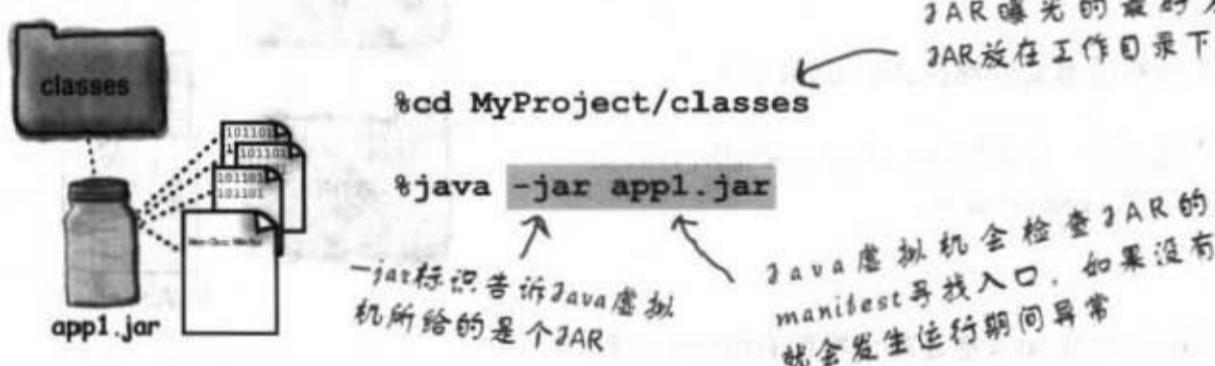


大部分完全在本机的 Java 应用程序都是以可执行的 JAR 来部署的。

## 执行 JAR

Java 虚拟机能够从 JAR 中载入类，并调用该类的 main() 方法。事实上，整个应用程序都可以包在 JAR 中。一旦 main() 方法开始执行，Java 虚拟机就不会在乎类是从哪里来的，只要能够找到就行。其中一个来源就是 classpath 指定位置的所有 JAR 文件。如果看到某个 JAR，则 Java 虚拟机就会在需要类的时候查询此 JAR。

Java 虚拟机必须能够找到 JAR，所以它必须在 classpath 下。让 JAR 曝光的最好方式就是把 JAR 放在工作目录下。



根据操作系统如何动态设定，有可能直接双击 JAR 就可以开始执行，Windows 与 Mac OS X 大致是这样。你可以通过点选 JAR 并要求 OS 以“Open with...”这一类的方式来打开。

there are no  
Dumb Questions

**问：**为什么不干脆把整个目录都 JAR 掉？

**问：**你说什么？

**答：**Java 虚拟机会检查 JAR 内部并预期找到所需的东西。它不会深入其他的目录找，除非类是包的一部分，或者目录符合包指令下的 ? 万用字符。

**答：**你不能把类文件放进某个目录后就这样包起来。但如果类属于某个包，那么你就可以把整个包结构给 JAR 起来。事实上是必须这么做的。稍后会对这做出解释。

## 把类包进包中！

写出可重复使用的类时，你会把它们放到内部的函数库给其他的程序员使用。就在你谦虚地让大家使用这些面向对象完美经典范例时，电话响了。一种不祥的电话。大傻也在函数库里面加进了相同名称的类，因此该出的状况一样也没少，你遇上了命理学所谓的姓名相克状况。

这都要怪你没有使用包！其实你使用包，把Java API包起来的那些包。但你没有把自己写的类包进包中，实际上来说这实在很糟糕。

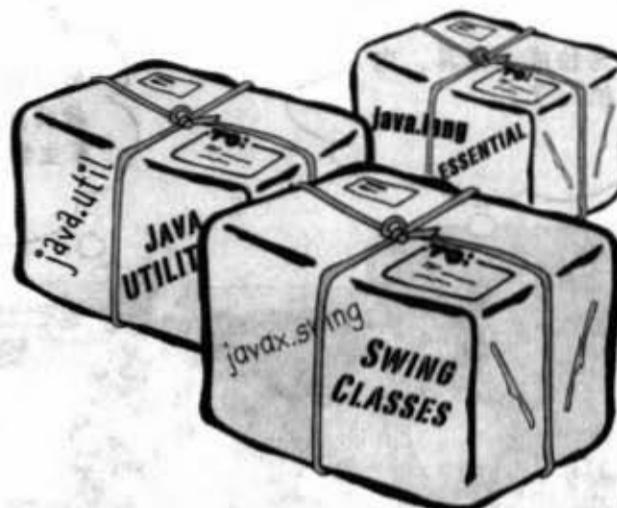
我们要把前几页展示的组织结构做个修改，让类包进包中，然后再把整个包给JAR起来。

接下来的内容要专心注意，细微的细节差异就可能让程序无法编译或执行。

## 用包防止类名称的冲突

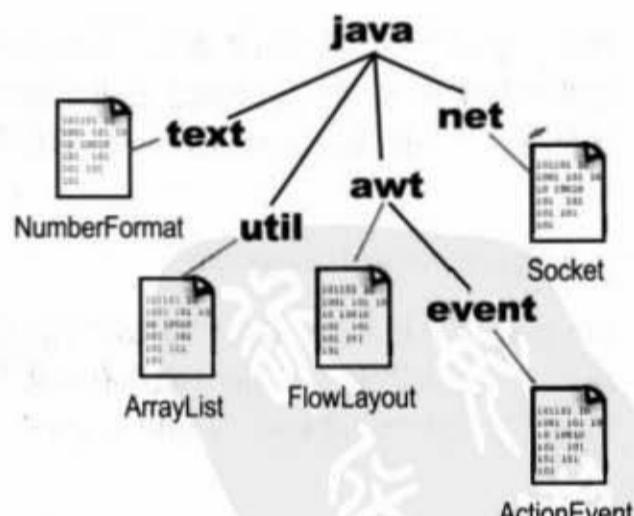
虽然包不只是用来防止名称冲突，但这也是主要的目的之一。你可能会写出称为Customer、Account与ShoppingCart的类。当然啦，超过一半以上写电子商务应用程序的开发者也会对class取这种菜市场名。这对面向对象来说是很危险的。如果面向对象的重点之一是写出可以重用的组件，开发者就必须能够组合各种来源的组件产生新的应用。你的组件也要能够跟其他组件和平相处，包括那些从你根本没想到的来源所写的。

回忆一下第6章我们讨论过包的名称就像是类的全名，技术上称为fully-qualified name。ArrayList其实是java.util.ArrayList，JButton其实叫做javax.swing.JButton，而Socket全名是java.net.Socket。注意到其中两个都是以java开头的。当你在处理包结构时要想到继承层次，并把你的类也做同样的安排。



Java API有这些包结构：

java.text.NumberFormat  
java.util.ArrayList  
java.awt.FlowLayout  
java.awt.event.ActionEvent  
java.net.Socket



这看起来像什么？是不是很像文件目录结构？



包可以防止名称冲突，但这只会在包名称保证不会重复的情况下起作用。最好的方式是在前面加上domain名称。

## 防止包命名冲突

把类包进包中可以减少与其他类产生命名冲突的机会，但要如何防止两个程序员做出同名的包呢？也就是说如何避免大家都把Account这个类放进名称为shopping.customers的包呢？如此一来最后的名称也会一样：

`shopping.customers.Account`

Sun建议的命名规则能够大幅降低冲突的可能性——加上你所取得的域名称。它会是独一无二的。也许有好几个人都会叫做“淑芬”，但是不会有两个网域都叫“oreilly.com.cn”。

`com.headfirstbooks.Book`

包名称

类名称

### 反向使用domain的包名称

将domain名称反过来放在前面

`com.headfirstjava.projects.Chart`

↑这个名字也许很常见，  
加上domain之后就只需  
担心同公司的人

类的名称第一个字母  
是大写的

## 把类包进包中

### ① 选择包名称

我们以com.headfirstjava为例。此类的名称为PackageExercise，因此完整的名称会是：com.headfirstjava.PackageExercise。

### ② 在类中加入包指令

这必须是程序源文件的第一个语句，比import语句还要靠上。每个原始文件只能有一个包指令，因此同一文件中的类都会在同一个包中。当然也包括内部。

```
package com.headfirstjava;

import javax.swing.*;

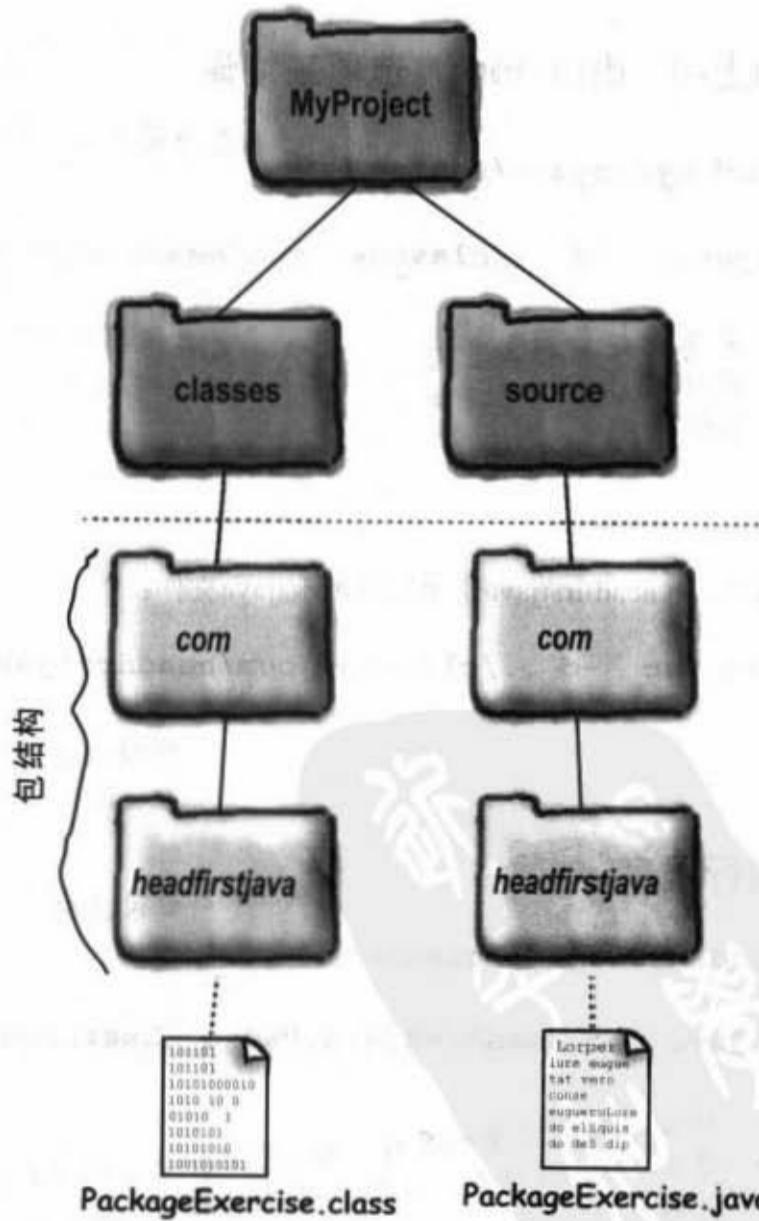
public class PackageExercise {
    // life-altering code here
}
```

### ③ 设定相对应的目录结构

只是把包指令加入源文件是不够的。类不会真的被加入包中，除非类也在相对应的目录结构中。因此，如果完整名称是com.headfirstjava.PackageExercise，则你必须把PackageExercise源文件放在名为headfirstjava的目录下，此目录必须在com目录下。

不这样做也可以编译，但相信我们，你不会想找自己的麻烦。让源代码摆在相对应的目录下你就可以避免令人头痛的问题。

你必须把类放在与包层次结构相对应的目录结构下。



将源文件和类都做出相对应的结构

## 编译与执行包

当类包在包中，编译与执行都要有点技巧。主要的问题来自于编译器和Java虚拟机都得要能够找到你的类以及所用到的其他类。对于核心API的类来说这不是问题。Java一定会知道它们在哪里。但对你的类而言，从单一目录来编译源文件是不可能的（至少是不可靠的）。然而我们保证如果能够照着这一页所描述的方法进行，你就一定会成功。还有其他的方法也可行，但我们发现这是最可靠也做容易遵循的方式。

### 加上-d (directory) 选项来编译

%cd MyProject/source ← 告在当前目录，别切换到.java文件的目录

%javac -d ../classes com/headfirstjava/PackageExercise.java

要求编译器将输出放到 class 目录下正确的包位置上

现在得指定取得源文件的路径

编译com.headfirstjava这个包的所有.java文件：

%javac -d ../classes com/headfirstjava/\*.java

↑ 编译此目录下所有的.java文件

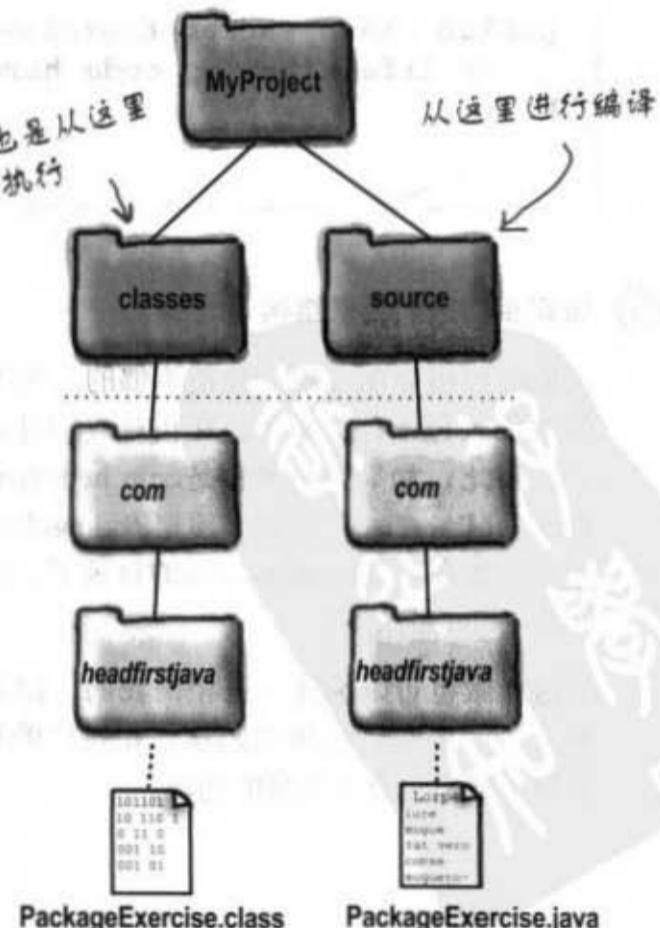
### 执行程序

从 classes 目录执行

%cd MyProject/classes

%java com.headfirstjava.PackageExercise

你必须指定完整的名称！Java虚拟机会看得懂并找寻当前目录下的com目录，其下应该有headfirstjava目录，那里应该能找到class。class在其他位置都无法运行！



## 其实-d选项真地很酷

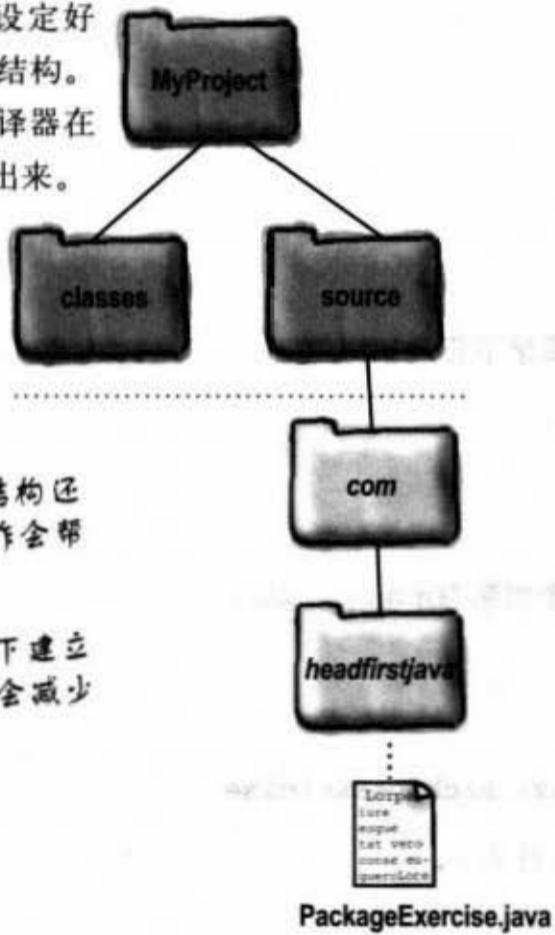
加上-d来编译是很棒的事情，因为它不仅让你把编译结果输出到别的地方，它还可以把类依照包的组织放到正确的目录上。

更棒的还在后头！

假设说你已经把源代码的目录结构设定好了，但还没有设定相对应的输出目录结构。没问题！加上-d的编译操作也会让编译器在遇到目录结构尚未建立时主动帮你做出来。

如果classes目录下包的目录结构还没有建好，使用-d的编译操作会帮你照顾好

因此你无需事先在classes之下建立目录结构，实际上这样反而会减少打字错误的可能



-d选项会要求编译器将编译结果根据包的结构来建立目录并输出，如果目录还没有建好，编译器会自动地处理这些工作。

there are no  
Dumb Questions

**问：** 我切换到主文件的目录下面，结果Java虚拟机就说找不到类！可是文件明明就在当前目录啊！

**答：** 一旦类被包进包中，你就不能用“简写”的名称来调用它。你必须在命令栏指定要执行main()的类的完整名称，这包括了包结构，Java会坚持类必须要待在相对应的目录结构下。若命令栏的指令像这样：

%java com.foo.Book

则Java虚拟机会从当前目录寻找名称为com的目录。在还没有找到带有foo子目录的com目录之前它不会去找寻名称为Book的类。只有在该目录下找到的Book才会被Java虚拟机接受。Java虚拟机也不会往上观察目录名称刚好是com然后决定就在这里找。

## 以包创建可执行的JAR



当你把类包进包中，包目录结构必须在JAR中！你不能只是把类装到JAR里面，还必须确定目录结构没有多往上走。包的第一层目录（通常是com）必须是JAR的第一层目录！如果你不小心从上面的目录包下来（例如从classes开始包），JAR就无法正确运行。

### 我创建可执行的JAR

- ① 确定所有的类文件都放在class目录下正确相对应的包结构中

- ② 创建manifest.txt文件来描述哪个类带有main()，以及确认有使用完整的类名称

在 manifest.txt 写入一行：

```
Main-Class: com.headfirstjava.PackageExercise
```

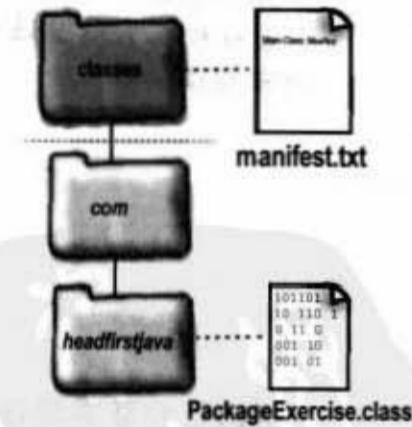
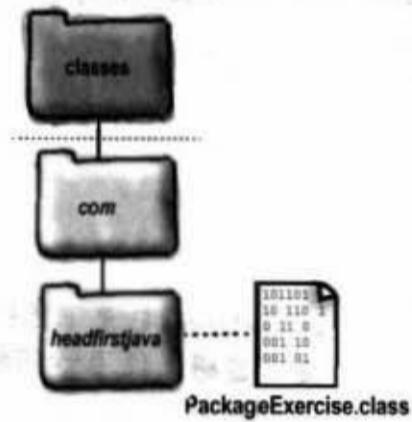
然后把 manifest 文件放到 classes 目录下。

- ③ 执行jar工具来创建带有目录结构与manifest的JAR文件

只要从com开始就行，其下整个包的类都会被包进去JAR。

```
%cd MyProject/classes
%jar -cvfm manifest.txt packEx.jar com
```

只要指定 com 目录  
就行，剩下都不  
会有问题



## 那manifest文件会跑到哪里？

看进去JAR就会知道。jar工具不只可以从命令栏中创建和执行JAR而已，你也可以把JAR的内容物解压出来（就像unzip或untar一样）。

假设说你已经把packEx.jar放到Skyler这个目录下。

### 条列和解压的jar命令

- ① 将JAR内容列出。

```
% jar -tf packEx.jar
```

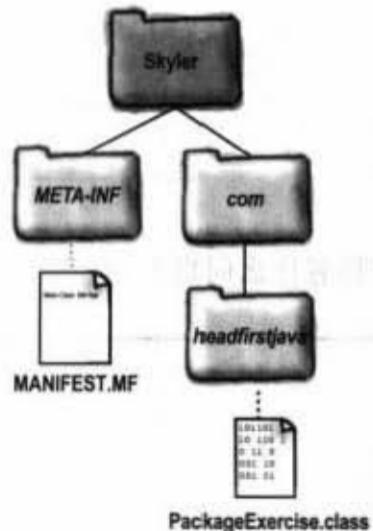
↑ tf代表Table File，也就是列出文件的列表。

```
File Edit Window Help Pickle
% cd Skyler
% jar -tf packEx.jar
META-INF/
META-INF/MANIFEST.MF
com/
com/headfirstjava/
com/headfirstjava/PackageExercise.class
```

- ② Extract the contents of a JAR (i.e. unjar)

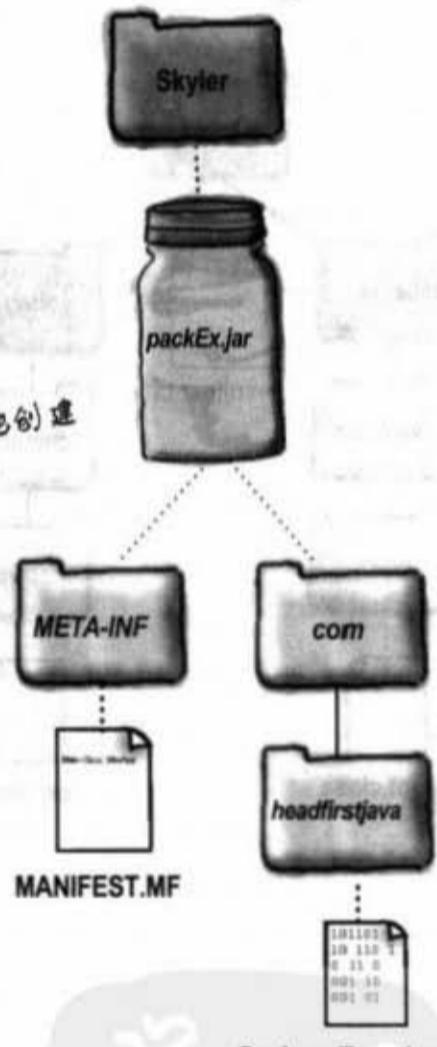
```
% cd Skyler
% jar -xf packEx.jar
```

↑ xf代表eXtract File，就像unzip一样。如果把packEx.jar解开，你会在当前目录之下看到META-INF和com目录



把JAR文件放到  
Skyler目录下

jar工具会自动地创建  
META-INF目录



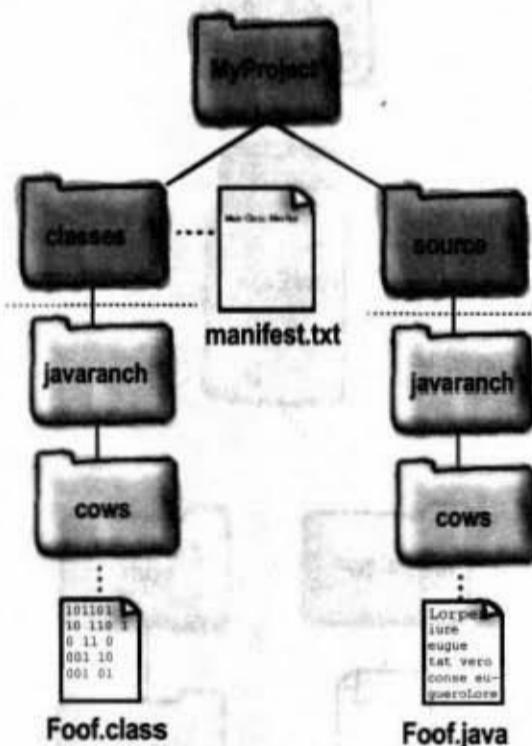
MANIFEST.MF



PackageExercise.class

META-INF代表META INFomation，jar工具会自动创建出这个目录和MANIFEST.MF文件，你的manifest.txt不会被带进JAR中，但它的内容会放进真正的manifest中。

## Sharpen your pencil



观察左图表示的包目录结构，写出应该在命令栏输入什么样的指令才能编译、执行、创建JAR、执行JAR。假设包目录结构是以标准的source与classes布局方式设定的。

### 编译：

```
%cd source  
%javac _____
```

### 执行：

```
%cd _____  
%java _____
```

### 创建JAR：

```
%cd _____  
% _____
```

### 执行JAR：

```
%cd _____  
% _____
```

作者免费招待的额外题目：这个包名称有什么问题？

there are no  
Dumb Questions

## 要点

**问：**如果用户尝试执行JAR但没有安装Java会怎样？

**答：**如果没有下雨撑伞会怎样？当然不会怎样，因为没有Java虚拟机所以Java程序就不会执行。

**问：**如何让用户安装Java？

**答：**理想情况是自定义安装程序和应用程序一并发布。有很多厂商提供简单或高等的工具来创建installer。有些安装程序能够检测用户的计算机是否有安装合适版本的Java，如果没有的话就会帮用户安装并设定Java。Installshield、InstallAnywhere与DeployDirector等都提供安装程序工具。

有些工具还能够制作安装光盘给各种主要的Java平台使用，因此一张CD-ROM就能搞定。比如在Solaris上它就会安装Solaris版的Java，在Windows上就会安装Windows版的Java。如果预算足够，你还可以来上一段耗资千万的动画片头。

- 将项目组织一下以让源代码和类文件分开在不同的目录下。
- 标准的组织化结构是创建出项目目录，然后在其下建立source和classes目录。
- 将类以包来组织，并在前面加上域名称以防止命名冲突。
- 在程序源文件最前面加上包指令可以把类包进包中：

```
package com.wickedlysmart;
```

- 类必须呆在完全相对应于包结构的目录中才能包进包中。以com.wickedlysmart.Foo来说，Foo这个类必须放在com目录下wickedlysmart这个目录中。
- 要让编译过的类可以放在正确的包目录结构中，使用-d编译标识：

```
%cd source
%javac -d ..\classes com/wickedlysmart/Foo.java
```

- 切换到classes目录然后指定完整的类名称来执行程序：

```
%java com.wickedlysmart.Foo
```

- 你可以把类包进JAR中，它的格式是根据pkzip制作的。
- 将描述哪个类带有main()的manifest包进JAR中可以制作出可执行的AR文件。manifest文件是个带有像下面这样设定的文本文件，记得最后要换行才能保证正确：

```
Main-Class: com.wickedlysmart.Foo
```

- 用下面的命令格式来创建JAR文件：

```
jar -cvfm manifest.txt MyJar.jar com
```

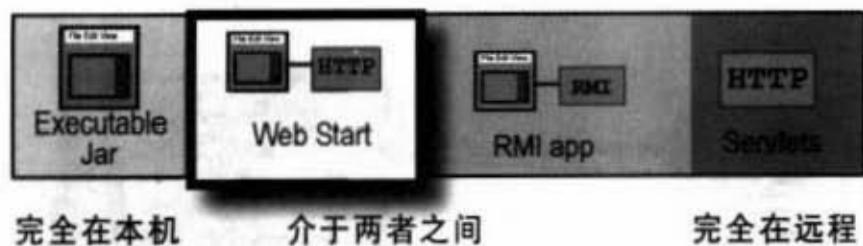
- JAR中的结构必须完全的符合包的目录结构。
- 以下面的命令格式来执行JAR：

```
java -jar MyJar.jar
```

你真的想太多了！



可执行的JAR是很不错，但如果办法能够制作出标准的独立的GUI在网络上发布会更好，这样就不必压缩发布CD-ROM了。如果程序还能自动地更新、替换掉有更新的部分那就更棒了，这样用户永远都会保持使用最新版……天啊！像我这种文艺美少女干嘛想这么多？



完全在本机 介于两者之间 完全在远程

用户能通过点选网页上的某个连接来启动Java Web Start的应用程序。一旦程序下载后，它就能独立于浏览器之外来执行。事实上，Java Web Start应用程序只不过是通过网络来发布的应用程序而已。

## Java Web Start

运用Java Web Start (JWS)，你的应用程序可以从浏览器上执行首次启动（从web来start，懂了吗？）但它运行起来几乎像是个独立的应用程序而不受浏览器的束缚。一旦它被下载到使用者的计算机之后（首次浏览网址来下载），它就会被保存下来。

Java Web Start是个工作上如同浏览器plug-in的小Java程序（就像ActiveX组件或用浏览器打开.pdf文件出现的Acrobat Reader）。这个程序被称为Java Web Start的hepler app，主要目的是用来管理下载、更新和启动JWS程序。

当JWS下载你的程序（可执行的JAR）时，它会调用程序的main()。然后用户就可以通过JWS helper app启动应用程序而不需回到当初的网页。

这还不是最棒的，JWS还能够检测服务器上应用程序局部（比如说某个类文件）的更新——在不需要用户介入的情况下，下载与整合更新过的程序。

当然这还有点问题，比如用户要如何取得Java以及JWS。但这个问题也可以解决：从Sun下载JWS。如果装了JWS但Java版本不是最新的，Java 2 Standard Edition也会被下载到用户计算机上。

最棒的是这一切都很简单。你可以把JWS应用程序当作HTML网页或.jpg图文件一样的网络资源，设置一个链接到你的JWS应用程序上，然后就可以工作了。

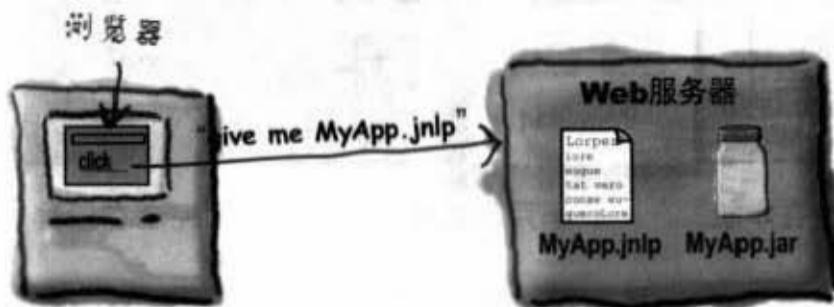
反正JWS应用程序就跟从网络上下载的可执行JAR一样。

## Java Web Start的工作方式

- ① 客户端点击某个网页上JWS应用程序的链接 (.jnlp 文件)。

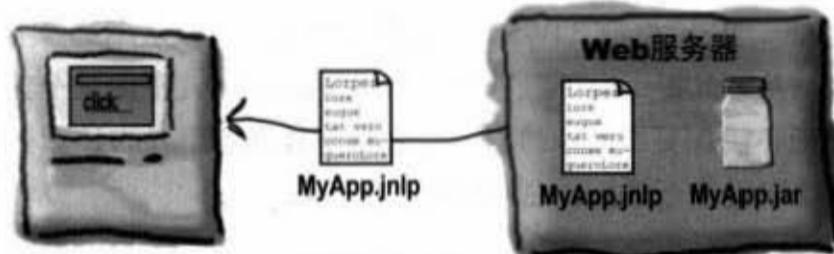
The Web page link

```
<a href="MyApp.jnlp">Click</a>
```



- ② Web服务器收到请求发出.jnlp文件 (不是 JAR) 给客户端的浏览器。

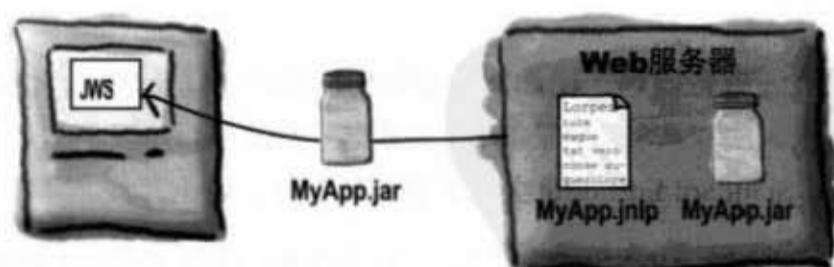
.jnlp文件是个描述应用程序可执行JAR文件的XML文件。



- ③ 浏览器启动Java Web Start, JWS的 helper app读取.jnlp文件, 然后向服务器请求MyApp.jar。



- ④ Web服务器发送.jar文件。



- ⑤ JWS取得JAR并调用指定的main()来启动应用程序。

然后用户就可以在离线的情况下通过JWS来启动应用程序。



## .jnlp 文件

你需要.jnlp文件（Java Network Lanuch Protocol）来制作Java Web Start的应用程序。JWS会读取这个文件来寻找JAR并启动应用程序（调用JAR里面的main()）。.jnlp文件是个简单的XML文件，里面可以做许多设置，但至少会有下面几项：

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="0.2 1.0"
      codebase="http://127.0.0.1/~kathy"
      href="MyApp.jnlp">
```

codebase用来指定相关文件的起始目录，我们使用127.0.0.1来测试本机上的程序

相对于codebase的位置路径，它也可以放在某个目录下

```
<information>
  <title>kathy App</title>
  <vendor>Wickedly Smart</vendor>
  <homepage href="index.html"/>
  <description>Head First WebStart demo</description>
  <icon href="kathys.gif"/>
  <offline-allowed/>
```

这些tag都一定要加进去，不然程序可能会有问题。information是给helper app用的，可供显示程序的信息

← 设定成离线时也可执行

</information>

```
<resources>
  <j2se version="1.3+/"> ← 指定需要1.3或之后版本的Java
  <jar href="MyApp.jar"/> ← 可执行JAR的名称
</resources>
```

<application-desc main-class="HelloWebStart"/>

</jnlp> ↑ 和manifest一样描述哪个类带有main()

## 创建与部署Java Web Start的步骤

- ① 将程序制作成可执行的JAR



- ② 编写.jnlp文件



- ③ 把.jnlp与JAR文件放到Web服务器



- ④ 对Web服务器设定新的mime类型

application/x-java-jnlp-file

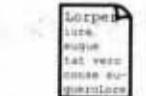
这会让Web服务器以正确的header送出.jnlp数据，如此才能让浏览器知道所接收的是什么。

Web服务器

设定mime

- ⑤ 设定网页连接到.jnlp文件

```
<HTML>
  <BODY>
    <a href="MyApp2.jnlp">Launch My Application</a>
  </BODY>
</HTML>
```





先有鸡  
还是  
先有蛋？



包、jar存档文件和部署

(1)

(2)

(3)

(4)

(5)

(6)

(7)

下面有一些事件，请排列出在JWS应用  
程序上面发生的正确顺序。



there are no  
Dumb Questions

**问：** Java Web Start与applet有什么不同？

**答：** applet无法独立于浏览器之外。applet是网页的一部分而不是单独的。浏览器会使用Java的plug-in来执行applet。applet没有类似程度的自动更新等功能，且一定得从浏览器上面执行。对JWS应用程序而言，一旦从网站上面下载后，用户不必通过浏览器就可以离线执行程序。

**问：** JWS有什么安全性的限制？

**答：** JWS有包括用户硬盘的读写等好几项限制。但JWS自有一套API可操作特殊的对话框来打开和存储文件，因此应用程序可以在用户同意的情况下存取硬盘上特定受限区域的文件。

## 要点

- Java Web Start技术让你能够从网站来部署独立的客户端程序。
- Java Web Start有个必须要安装在客户端的 helper app (当然也需要Java)。
- JWS程序由两个部分组成：可执行的JAR与.jnlp文件。
- .jnlp文件是用来描述JWS应用程序的XML文件。它有tag以指定JAR的名称和位置，以及带有main( )的类名称。
- 当浏览器从服务器上取得.jnlp文件时，浏览器就会启动JWS的helper app。
- JWS的helper app会读取.jnlp来判断要从服务器上下载的可执行 JAR。
- 取得 JAR 之后它就会调用.jnlp指定的 main()。



这一章讨论包、部署和JWS。你的任务是判断下列陈述的真假。

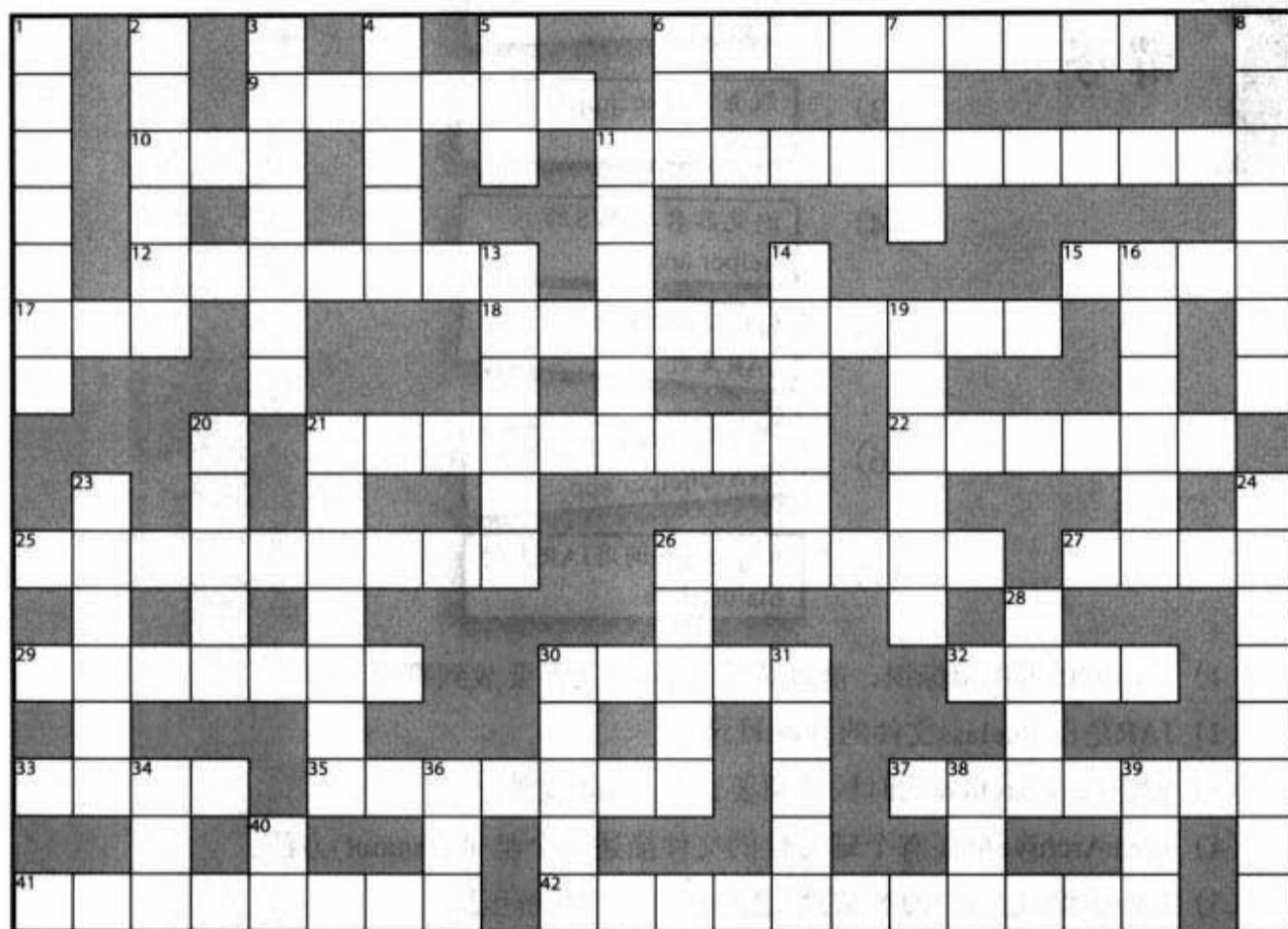
## 是非题

- (1) Java编译器的-d标识能让你决定.class文件要放到哪里。
- (2) JAR是保存.class文件的标准目录。
- (3) 创建Java Archive的时候必须要设定jar.mf文件。
- (4) Java Archive里面有个辅助性的文件描述哪个类带有main()方法。
- (5) Java虚拟机在使用JAR的class之前必须先将其解压缩。
- (6) Java Archive从命令行调用时必须加上-arch标识。
- (7) 包结构是以层次来表示的。
- (8) 对包的命名不建议使用公司的域名称。
- (9) 同一源文件中的不同类可以放到不同的包中。
- (10) 建议加上-p标识来编译包中的类。
- (11) 编译包中的类时，完整名称必须和目录结构一致。
- (12) 使用-d标识能够预防打错类名称。
- (13) 将JAR解压缩时会产生meta-inf目录。
- (14) 将JAR解压缩时会产生manifest.mf文件。
- (15) JWS的helper app必须要和浏览器一起执行。
- (16) JWS程序需要.html文件才能操作。
- (17) JWS的main()方法是由JAR文件所指定的。



练习

## 字谜 7.0



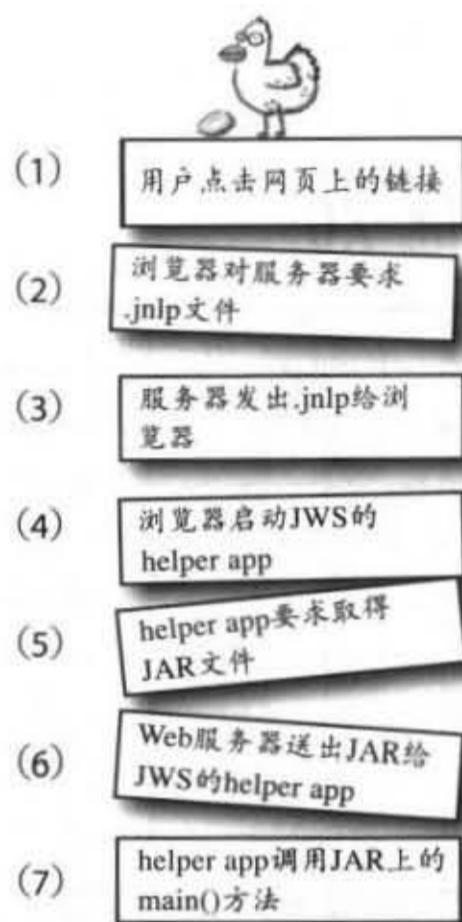
书中提到的东西都  
有可能出现

### 横排提示：

- 6. Won't travel
- 9. Don't split me
- 10. Release-able
- 11. Got the key
- 12. I/O gang
- 15. Flatten
- 17. Encapsulated returner
- 18. Ship this one
- 21. Make it so
- 22. I/O sieve
- 25. Disk leaf
- 26. Mine is unique
- 27. GUI's target
- 29. Java team
- 30. Factory
- 32. For a while
- 33. Atomic \* 8
- 35. Good as new
- 37. Pairs event
- 41. Where do I start
- 42. A little firewall

### 竖排提示：

- 1. Pushy widgets
- 2. \_\_\_ of my desire
- 3. 'Abandoned' moniker
- 4. A chunk
- 5. Math not trig
- 6. Be brave
- 7. Arrange well
- 8. Swing slang
- 11. I/O canals
- 13. Organized release
- 14. Not for an instance
- 16. Who's allowed
- 19. Efficiency expert
- 20. Early exit
- 21. Common wrapper
- 23. Yes or no
- 24. Java jackets
- 26. Not behavior
- 28. Socket's suite
- 30. I/O cleanup
- 31. Milli-nap
- 34. Trig method
- 36. Encaps method
- 38. JNLP format
- 39. VB's final
- 40. Java branch



**True** (1) Java编译器的-d标识，能让你决定.class文件要放到哪里。

**False** (2) JAR是保存.class文件的标准目录。

**False** (3) 创建Java Archive的时候必须要设定jar.mf文件。

**True** (4) Java Archive里面有个辅助性的文件描述哪个类带有main()方法。

**False** (5) Java虚拟机在使用JAR的类之前必须先将其解压缩。

**False** (6) Java Archive从命令行调用时必须加上-arch标识。

**True** (7) 包结构是以层次来表示的。

**False** (8) 对包的命名不建议使用公司的域名称。

**False** (9) 同一源文件中的不同类可以放到不同的包中。

**False** (10) 建议加上-p标识来编译包中的类。

**True** (11) 编译包中的类时，完整名称必须和目录结构一致。

**True** (12) 使用-d标识能够预防打错类名称。

**True** (13) 将JAR解压缩时会产生meta-inf目录。

**True** (14) 将JAR解压缩时会产生manifest.mf文件。

**False** (15) JWS的helper app必须要和浏览器一起执行。

**False** (16) JWS程序需要.html文件才能操作。

**False** (17) JWS的main()方法是由JAR文件所指定的。



## 字谜7.0



java学习群：72030155，每天20:30-23:00都有大神视频教学，想学习的同学可以进群免费听课！

## 18 远程部署的RMI

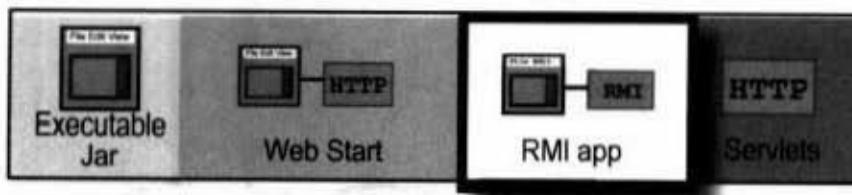
# 分布式计算



每个人都说远距离的恋爱是很辛苦的，但我俩有RMI，所以一点也不会。RMI让我们感觉在一起。

**距离不是问题。**当然啦，如果所有组件都在同一台计算机的同一个Java虚拟机的同一个堆空间上执行是最简单的，但有时候就是办不到。如果用户端只是个能够执行Java的装置怎么办？如果为了安全性的理由只能让服务器上的程序存取数据库怎么办？想象一下电子商务的情境。有时候，程序的某些部分就是得在服务器上执行，而客户端会在不同用户的计算机上执行。这一章会介绍Java的远程程序调用（Remote Method Invocation, RMI）技术。我们也会很快地看过Servlet、Enterprise Java Bean（EJB）、Jini以及EJB与Jini是如何运用RMI。最后你还会以Java创建出一个通用服务浏览器。

有多少个堆



完全在本机

介于两者之间

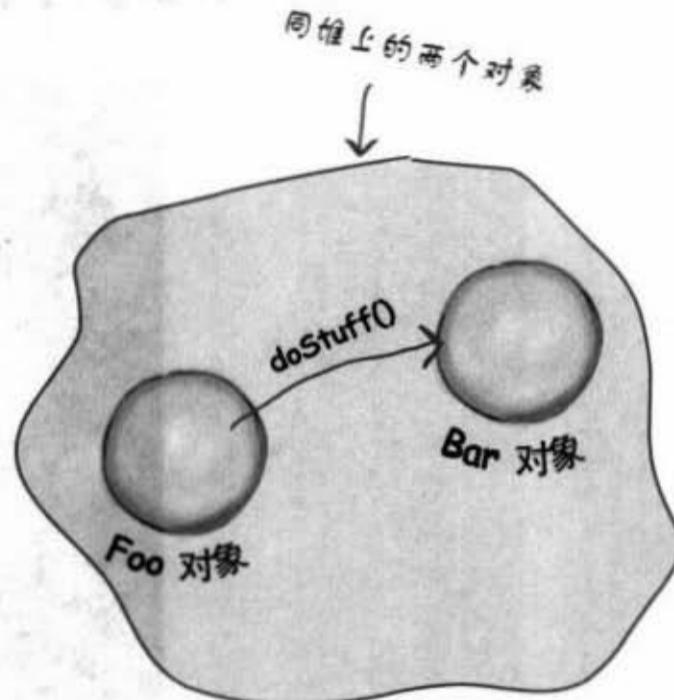
完全在远程

方法的调用都是发生在相同堆上的两个对象之间

本书所描述的方法调用到目前为止都是对运行在相同Java虚拟机上的对象所进行的。也就是说调用方与被调用方都是在同一个堆上。

```
class Foo {  
    void go() {  
        Bar b = new Bar();  
        b.doStuff();  
    }  
    public static void main (String[] args) {  
        Foo f = new Foo();  
        f.go();  
    }  
}
```

在上面的程序中，我们知道Foo实例是被f引用，而Bar对象是被b所引用，两者都在同一个Java虚拟机上的同一个堆中。要记得Java虚拟机会负责把表示“如何取得堆上的对象”的字节组合填入引用变量中。Java虚拟机一定会知道对象，以及如何取得对象。但Java虚拟机只会知道自有堆的引用！例如你无法让Java虚拟机知道不同机器上Java虚拟机的信息。这就使Java虚拟机是不是在同一台机器上运行没有什么区别，只是使两个Java虚拟机的调用不同。



一般来说，对象的方法调用都是在相同的Java虚拟机上面进行的。

## 如果要调用不同机器上的对象的方法呢？

我们要知道如何从某一台计算机上面取得另一台计算机上的信息——通过Socket和输入/输出。打开另一台计算机的Socket连接，然后取得OutputStream来写入数据。

但如果要调用另一台计算机上，另一个Java虚拟机上面的对象的方法呢？我们当然可以自己定义和设计通信协议来调用，然后通过Socket把执行的结果再传回去。想想就知道这有多麻烦。如果能够取得另一台计算机上对象的引用该有多好。



大人物拥有小不点想要的东西。

就是运算能力。

小不点想要发出数据给大人物，让大人物来计算。

小不点只想要调用方法……

`double doCalcUsingDatabase(CalcNumbers numbers)`

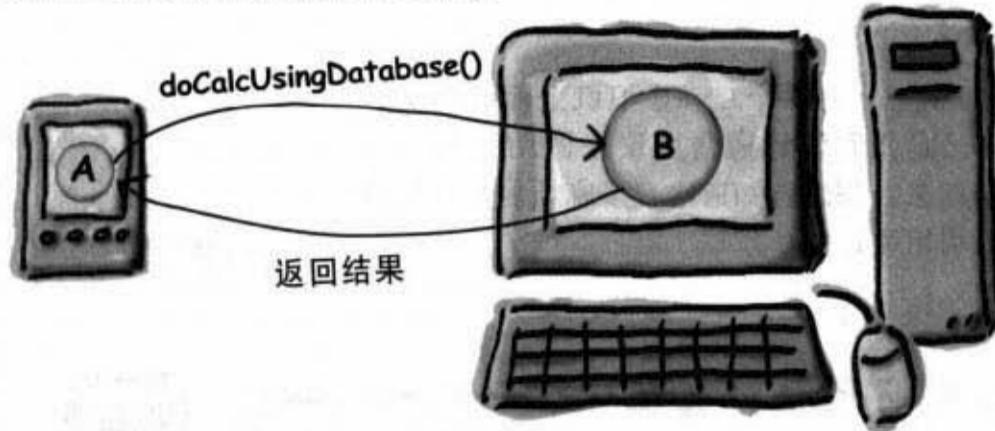
然后取得结果。

但小不点怎样才能取得大人物的对象引用呢？

两个对象，两个堆

## 小不点的对象A想要调用大人物上对象B的方法

问题是如何取得不同机器上的对象来调用某个方法呢？



办不到

嗯……无法直接调用。你无法取得别的堆上的引用。如果像下面这样写：

```
Dog d = ???
```

无论如何d都只能引用到执行程序的堆。

如果你想要设计出某种机制能够使用Socket和输入/输出来表达你的意图（对另一台机器上面运行的对象调用方法），并且还能够像是对本机的方法调用一样，也就是说你想要调用远程的对象（像是别的堆上的），却又要像是一般的调用。

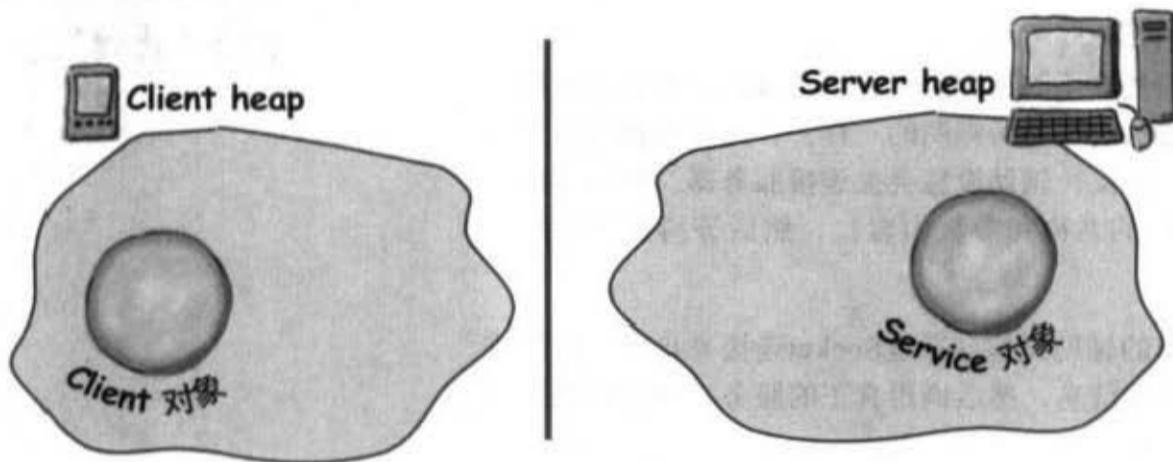
这就是RMI能够带给你的功能！

先让我们回头假设你要自己设计RMI功能。了解要如何自行创建能够帮助你学习RMI是如何工作的。

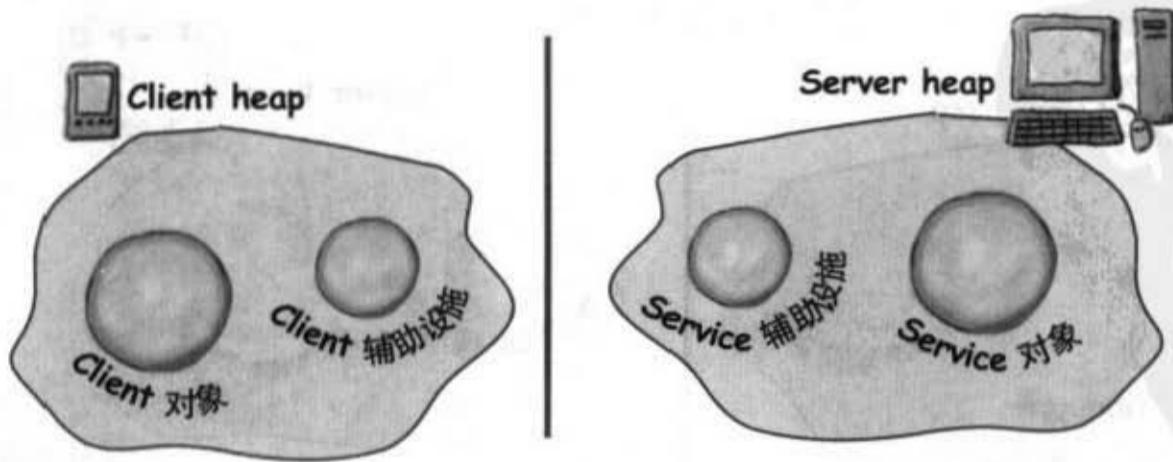
# 远程过程调用的设计

要创建出4种东西：服务器、客户端、服务  
器辅助设施和客户端辅助设施

- ① 创建客户端和服务器应用程序。服务器应用程序是个远程服务，是个带有客户端会调用的方法的对象。



- ② 创建客户端和服务器端的辅助设施（helper）。它们会处理所有客户端和服务器的低层网络输入/输出细节，让你的客户端和程序好像在处理本机调用一样。



## 辅助设施的任务

辅助设施是个在实际上执行通信的对象。它们会让客户端感觉上好像是在调用本机的对象。事实上正是这样。客户端调用辅助设施的方法，就好像客户端就是服务器一样。客户端是真正服务的代理（proxy）。

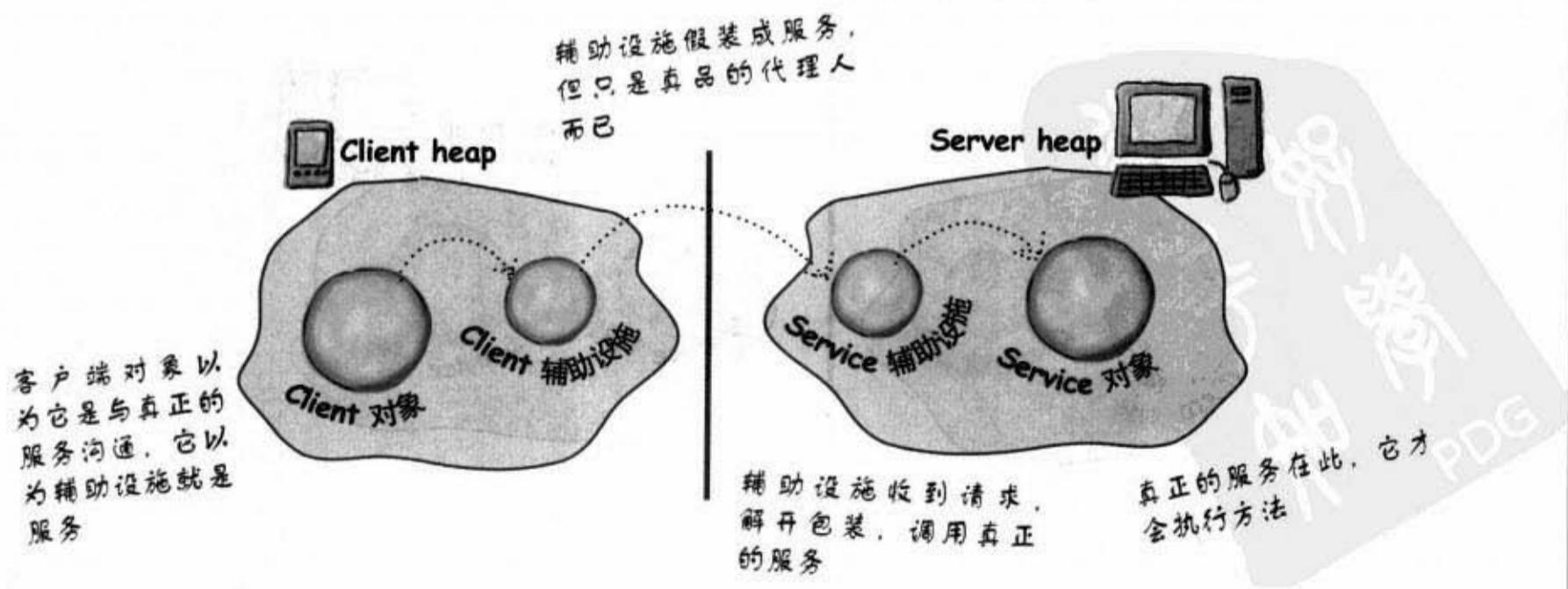
也就是说，客户端以为它调用的是远程的服务，因为辅助设施假装成该服务对象。

但客户端辅助设施并不是真正的远程服务，虽然辅助设施的举止跟它很像（因为它提供的方法跟服务所声明的一样），却没有任何真正客户端需要的方法逻辑。相反，辅助设施会去连接服务器，将调用的信息传送过去（像是方法的名称和参数内容），然后等待服务器的响应。

在服务器这端，服务器的辅助设施会通过Socket连接来自客户端设施的要求，解析打包送来的信息，然后调用真正的服务。因此对服务对象来说此调用来自于本地。

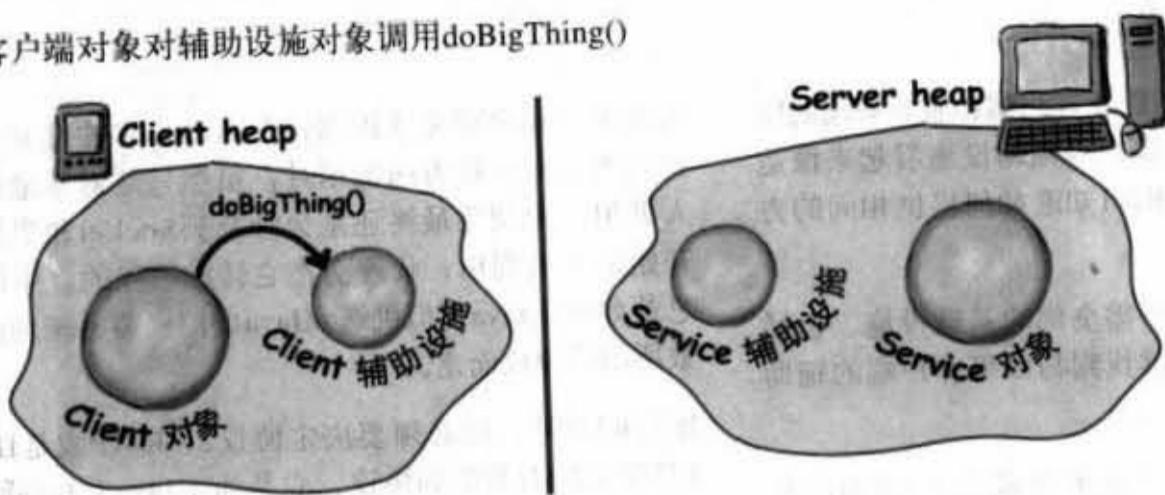
服务的辅助设施取得返回值之后就把它包装然后送回去（通过Socket的输出串流）给客户端的辅助设施。客户端的辅助设施会解开这些信息传给客户端的对象。

客户端对象看起来像是在调用远程的方法。但实际上它只是在调用本地处理Socket和串流细节的“代理”。

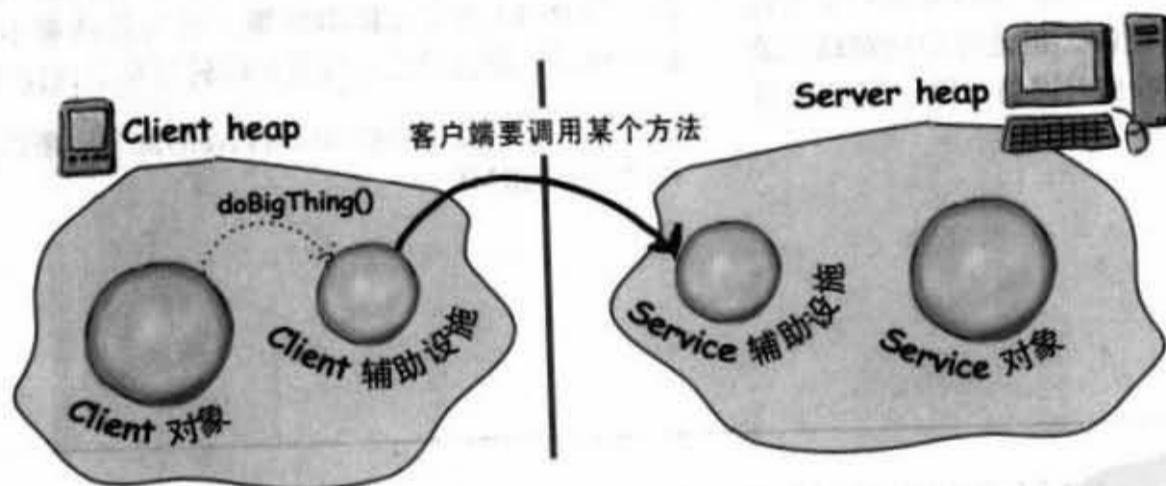


## 调用方法的过程

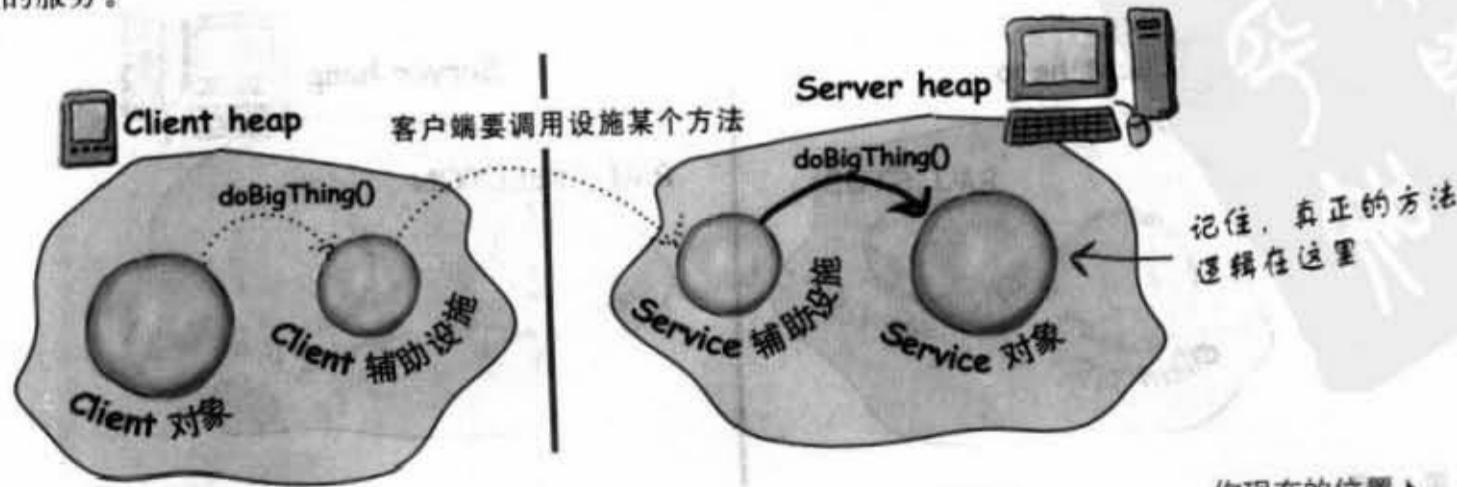
- ① 客户端对象对辅助设施对象调用doBigThing()



- ② 客户端辅助设施把调用信息打包通过网络送到服务器的辅助设施



- ③ 服务端的辅助设施解开来自客户端辅助设施的信息，并以此调用真正的服务。



你现在的位置：

613

## Java RMI 提供客户端和服务器端的辅助设施对象！

在Java中，RMI已经帮你创建好客户端和服务器端的辅助设施，它也知道如何让客户端辅助设施看起来像是真正的服务。也就是说，RMI知道如何提供相同的方法给客户端调用。

此外，RMI有提供执行期所需全部的基础设施，包括服务的查询以让客户端能够找到与取得客户端的辅助设施（真正服务的代理人）。

使用RMI时，你无需编写任何网络或输入/输出的程序。客户端对远程方法的调用就跟对同一个Java虚拟机上的方法调用是一样的。

一般调用和RMI调用有一点不同。虽然对客户端来说此方法调用看起来像是本地的，但是客户端辅助设施会通过网络发出调用。那我们对网络与输入/输出有什么感觉呢？

它们都会有风险！

它们是会抛出异常的。

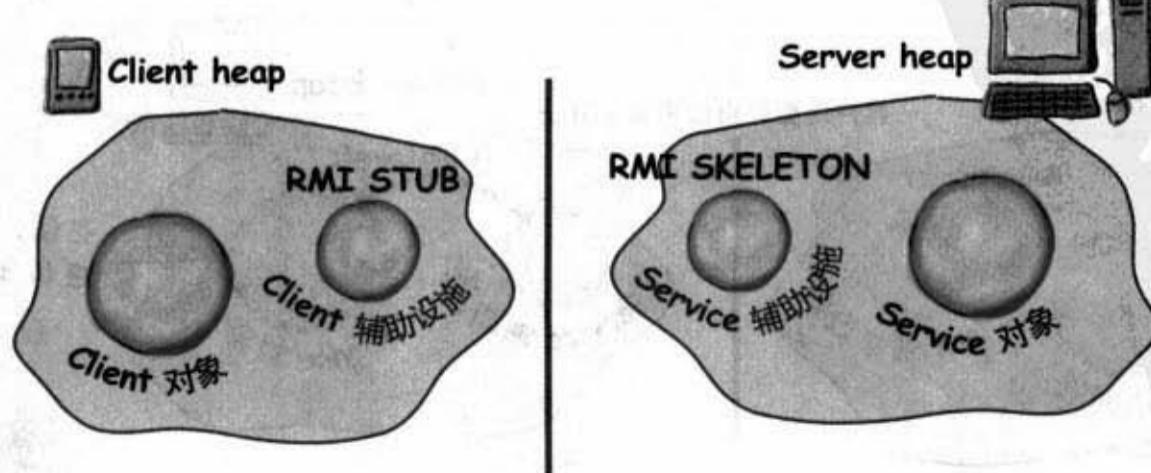
因此客户端必须要认识到这个风险。客户端必须要认识到当它对远程方法调用时，虽然只是对本地的代理人调用，此调用最终还是会涉及到Socket和串流。一开始是本机调用，代理会把它转成远程的。中间的信息是如何从Java虚拟机送到Java虚拟机要看辅助设施对象所用的协议而定。

使用RMI时，你必须要决定协议：JRMP或是IIOP。JRMP是RMI原生的协议，它是为了Java对Java间的远程调用而设计的。另外一方面，IIOP是为了CORBA（Common Object Request Broker Architecture）而产生的，它让你能够调用Java对象或其他类型的远程方法。CORBA通常比RMI麻烦，因为若两端不全都是Java的话，就会发生一堆可怕的转译和交谈操作。

幸好，我们只需要关心Java对Java的部分，所以会使用相当简易的RMI。

---

在RMI中，客户端的辅助设施称为 stub，而服务器端的辅助设施称为skeleton。



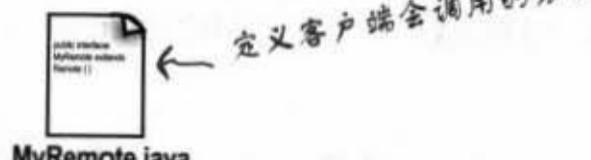
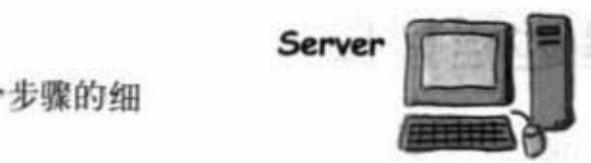
# 创建远程服务

这是个创建远程服务的5个步骤，接下来会有每个步骤的细节。

## 步骤1：

### 创建 Remote 接口

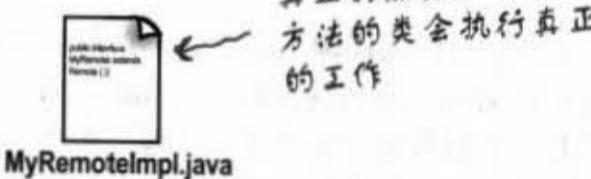
远程的接口定义了客户端可以远程调用的方法。它是个作为服务的多态化类。stub 和服务都会实现此接口！



## 步骤2：

### 实现Remote

这是真正执行的类。它实现出定义在该接口上的方法。它是客户端会调用的对象。



## 步骤3：

### 用rmic产生stub与skeleton

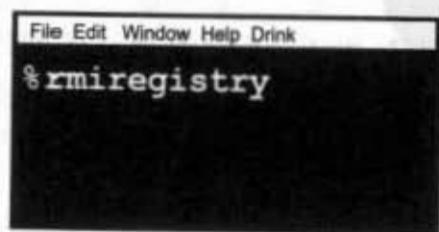
客户端和服务器都有 helper。你无需创建这些类或产生这些类的源代码。这都会在你执行 JDK 所附的 rmic 工具时自动地处理掉。



## 步骤4：

### 启动 RMI registry (rmiregistry)

rmiregistry 就像是电话簿。用户会从此处取得代理（客户端的 stub/helper 对象）。

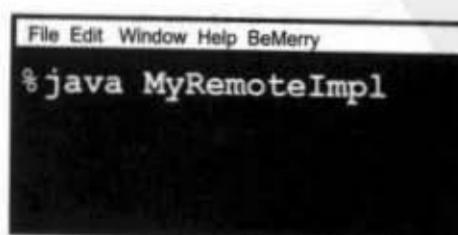


在另一个命令行上执行

## 步骤5：

### 启动远程服务

你必须让服务对象开始执行。实现服务的类会起始服务的实例并向 RMI registry 注册。要有注册后才能对用户提供服务。



## 步骤1：创建远程接口

### ① 继承java.rmi.Remote

Remote是个标记性的接口，意味着没有方法。然而它对RMI有特殊的意义，所以你必须遵守这项规则。注意这里用的是extend，接口是可以继承其他的接口。



MyRemote.java

外你自己的接口声明出它和远程方法调用有关

```
public interface MyRemote extends Remote {
```

### ② 声明所有的方法都会抛出RemoteException

远程的接口定义了客户端可以远程调用的方法。它是个作为服务的多态化类。也就是说，客户端会调用有实现此接口的stub，而此stub因为会执行网络和输入/输出工作，所以可能会发生各种问题。客户端必须处理或声明异常来认知这一类风险。如果方法在接口中声明异常，调用该方法的所有程序都必须处理或再声明此异常。

```
import java.rmi.*; ← Remote在java.rmi中
public interface MyRemote extends Remote {
    public String sayHello() throws RemoteException;
}
```

每个远程调用都会被认为是有风险的，这样声明会强迫客户端要注意到这件事

### ③ 确定参数和返回值都是primitive主数据类型或Serializable

远程方法的参数和返回值必须是primitive或Serializable的。任何远程方法的参数都会被打包通过网络传送，而这是通过序列化来完成的。返回值也是一样。如果使用的是API中像是String、primitive主数据类型等主要的类型（包括数组和集合），那就没问题。如果是自定义的类型，那你就得要实现Serializable。

```
public String sayHello() throws RemoteException;
```

返回值会通过网络传送，所以必须是Serializable。参数和返回值都是这样打包传送的

## 步骤2：实现远程接口

### ① 实现Remote这个接口

你的服务必须要实现Remote——就是客户端会调用的方法。

```
public class MyRemoteImpl extends UnicastRemoteObject implements MyRemote {
    public String sayHello() { ← 编译器会确保你实现出所有
        return "Server says, 'Hey'"; 接口定义的方法，此例中只有一个
    }
    // more code in class
}
```



### ② 继承UnicastRemoteObject

为了要成为远程服务对象，你的对象必须要有与远程有关的功能。其中最简单的方式就是继承UnicastRemoteObject（来自java.rmi.server）以让这个父类处理这些工作。

```
public class MyRemoteImpl extends UnicastRemoteObject implements MyRemote {
```

### ③ 编写声明 RemoteException 的无参数构造函数

UnicastRemoteObject有个小问题：它的构造函数会抛出RemoteException。处理它的唯一方式就是对你的实现声明一个构造函数，如此才会有地方可以声明出RemoteException。要记得当类被初始化的时候，父类的构造函数一定会被调用，如果父类的构造函数抛出异常，你也得声明你的构造函数会抛出异常。

```
public MyRemoteImpl() throws RemoteException {}
```

真的不需写出什么，只是需要一种方式声明父类的构造函数会抛出异常

### ④ 向 RMI registry 注册服务

现在你已经有了远程服务，还必须要让远程用户存取。这可以通过将它初始化并加进RMI registry（它一定得要运行起来，不然此行程序就会失败）。当你注册对象时，RMI系统会把stub加到registry中，因为这是客户端所需要的。使用java.rmi.Naming的rebind()来注册服务。

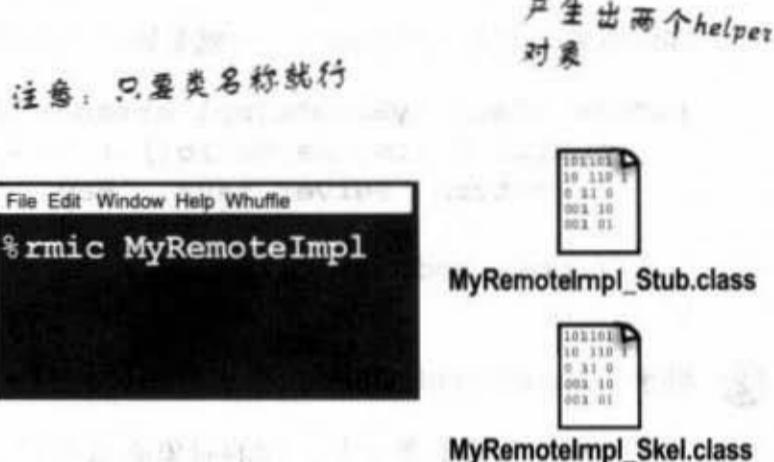
```
try {
    MyRemote service = new MyRemoteImpl();
    Naming.rebind("Remote Hello", service);
} catch(Exception ex) {...}
```

帮服务命名（客户端会靠名字查询registry），并向RMI registry注册，RMI会将stub做交换并把stub加入 registry

## 步骤3：产生stub和skeleton

### ① 对实现出的类（不是remote接口）执行rmic

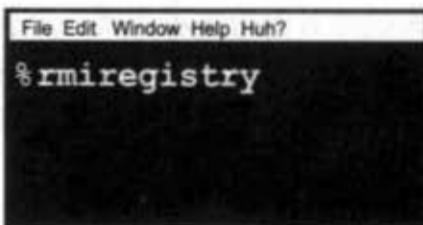
伴随Java Software Development Kit而来的rmic工具会以服务的实现产生出两个新的类：stub和skeleton。它会按照命名规则在你的远程实现名称后面加上\_Stub或\_Skeleton。rmic有几个选项，包括了不产生skeleton，观察产出类的源代码或使用IIOP作为通信协议等。我们在此使用的是一般的方式。产出的类会放在当前目录下。要记住rmic必须能够找到你所实现的类，因此你或许要从实现所在的目录执行rmic（为了简化起见，我们并没有运用到包。实际上你可能需要考虑到包目录结构和完整名称）。



## 步骤4：执行 rmiregistry

### ① 调出命令行来启动 rmiregistry

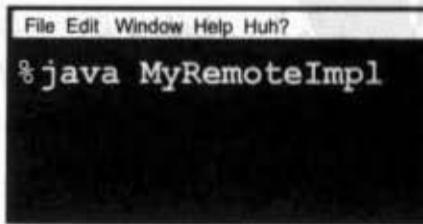
要确定你是从可以存取到该类的目录来启动。最简单的方法就是从类这个目录来运行。



## 步骤5：启动服务

### ① 调用另一个命令行来启动服务

这可能是从你所实现的类上的main()或独立的启动用类来进行。在这个简单的范例中，我们把启动程序代码放在类的实现中，它的main()会将对象初始化并把它注册给RMI registry。



# 完整的程序代码



## Remote interface:

```

import java.rmi.*;      // RemoteException 和 Remote 接口都在 java.rmi 中
public interface MyRemote extends Remote {
    public String sayHello() throws RemoteException; // 所有接口中的方法都必须声明 RemoteException
}

```

## Remote service (实现) :

```

import java.rmi.*;      // UnicastRemoteObject 在 java.rmi.server 中
import java.rmi.server.*; // 这是创建远程对象最简单的方式
public class MyRemoteImpl extends UnicastRemoteObject implements MyRemote {
    public String sayHello() { // 你得实现出接口所有的方法，但注意到你无需声明 RemoteException
        return "Server says, 'Hey'";
    }
    public MyRemoteImpl() throws RemoteException { // 父类的构造函数声明了异常，所以你必须写出构造函数，因为它代表你的构造函数会调用有风险的程序代码
    }
    public static void main (String[] args) {
        try {
            MyRemote service = new MyRemoteImpl(); // 创建出远程对象，然后使用静态的 Naming.rebind() 来产生关联，所注册的名称会供客户端查询
            Naming.rebind("Remote Hello", service);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}

```

取得 stub

## 客户端如何取得 stub 对象？

客户端必须取得 stub 对象，因为客户端必须要调用它的方法。这就得靠 RMI registry 了。客户端会像查询电话簿一样地搜索，找出上面有相符名称的服务。

MyRemote service = (MyRemote) Naming.lookup("rmi://127.0.0.1/Remote Hello");

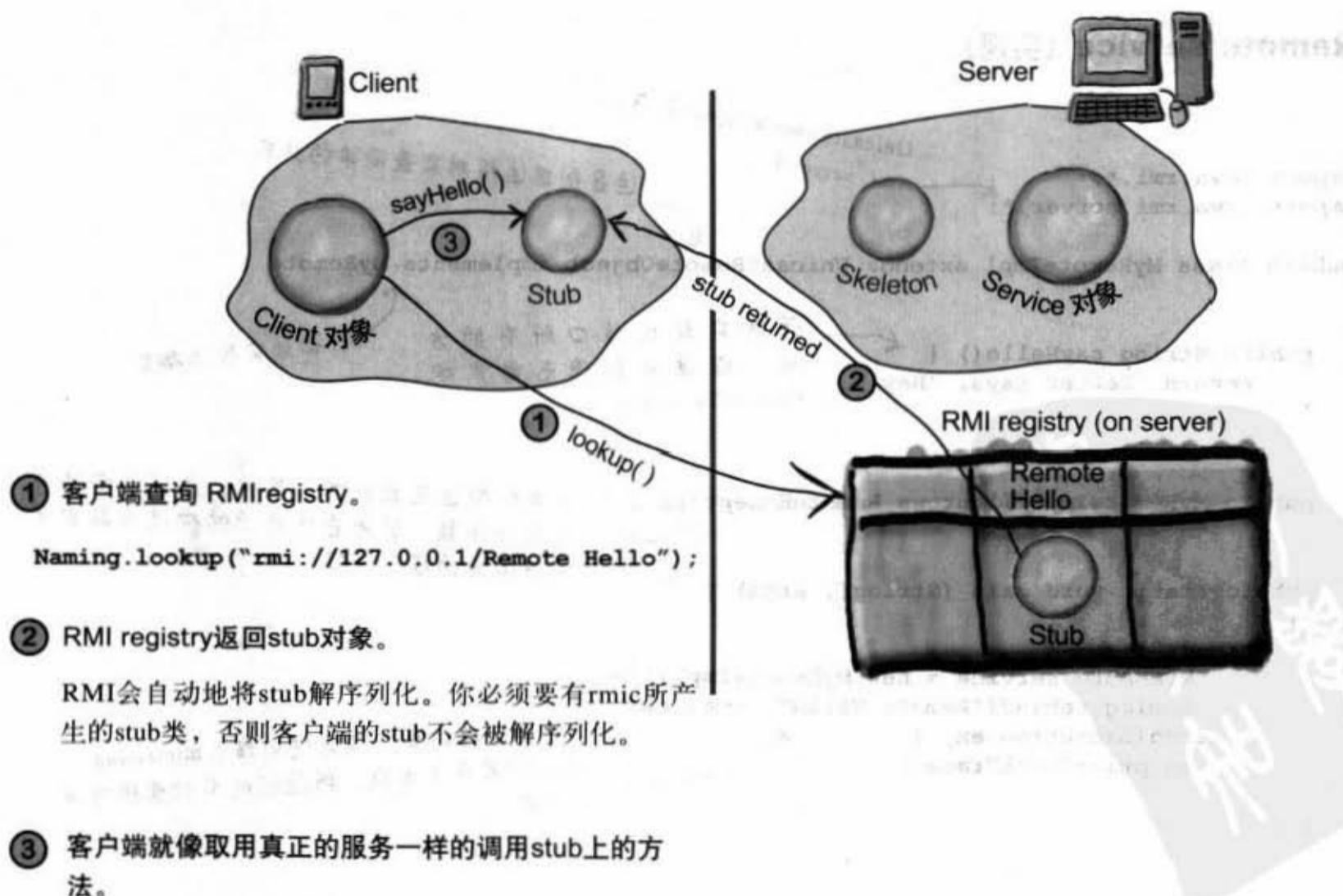
lookup()是个静态的方法

必须要跟注册的名称一样

客户端必须要使用与服务相同的类型，事实上客户端不需要知道服务实际上的类名称

必须要转换成接口的类型，因为查询结果会是 Object 类型

主机名称 (host name) 或 IP 地址



## 用户如何取得stub的类？

现在有个很有意思的问题。不管怎样做，客户端在查询服务时一定要有stub类（之前用rmic产生出来的），不然就无法在客户端解序列化。在最简单的情况下，你只要把stub的类直接交给用户就行。

但是还有更酷的方式，然而那已经超出本书的范围了。不过我们还是稍微地说明一下：最简单的方式称为“dynamic class downloading”。使用动态类下载时，stub对象会被加上注明RMI可以去哪里找到该对象的类文件的URL标记。之后在解序列化的过程中，RMI会在本机上找不到类，所以就使用HTTP的Get来从该URL取得类文件。因此你会需要一个Web服务器来提供类文件，并且也得改变客户端的某些安全性设定。这里面还有些特别的问题，不过我们就先看过动态类下载的概念就行。

## 完整的客户端程序代码

```

import java.rmi.*;
Naming类是在java.rmi中

public class MyRemoteClient {
    public static void main (String[] args) {
        new MyRemoteClient().go();
    }

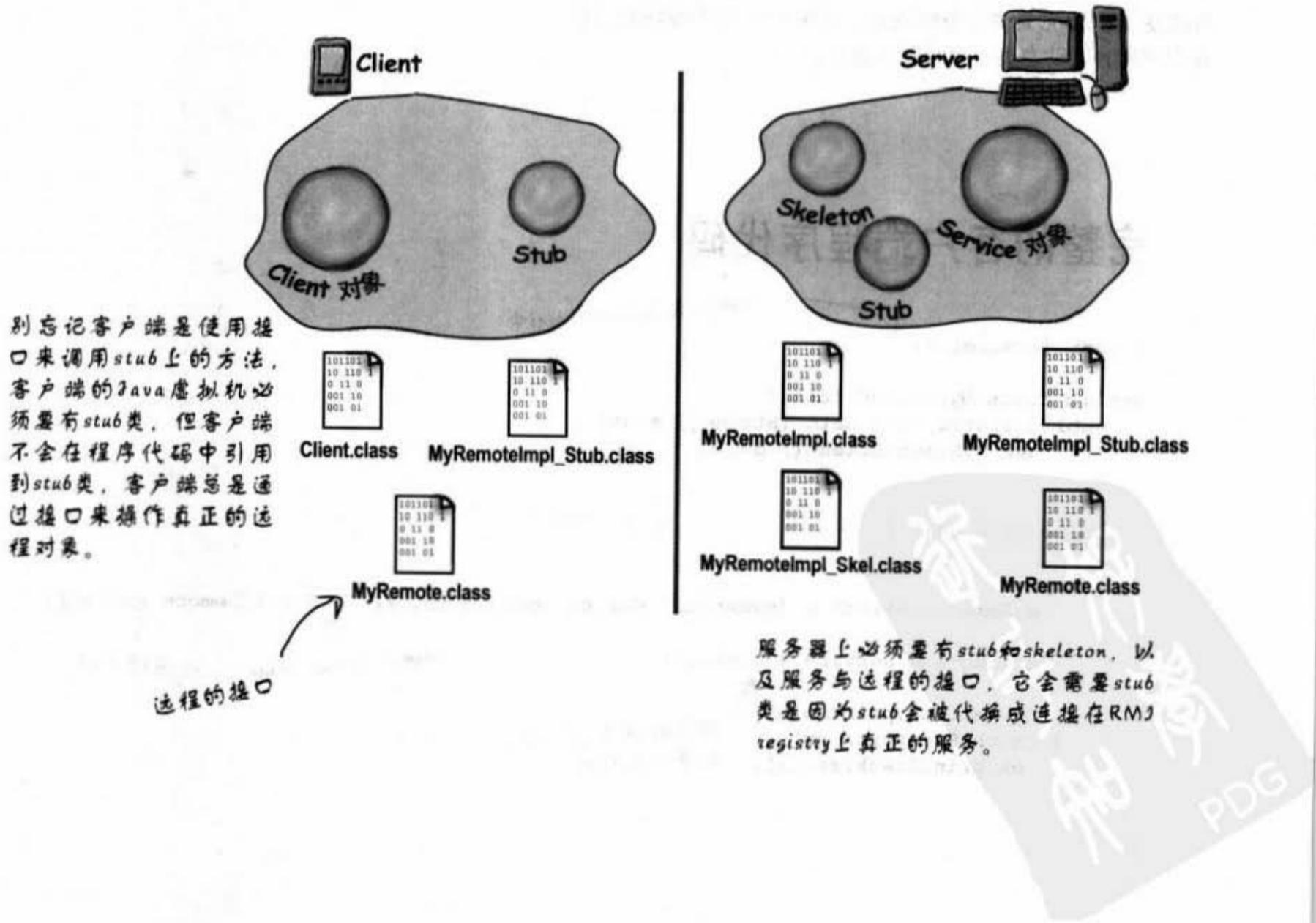
    public void go() {
        try {
            MyRemote service = (MyRemote) Naming.lookup("rmi://127.0.0.1/Remote Hello");
别忘了做类型转换
            String s = service.sayHello();
IP地址或host name 注册的名称
            System.out.println(s);
        } catch (Exception ex) {
就像是调用本机的方法，但
            ex.printStackTrace();
        }
    }
}

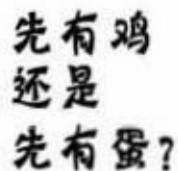
```

## 确保每台机器都有所需的类文件

使用 RMI 时程序员最常犯的3个错误：

- (1) 忘记在启动远程服务前启动rmiregistry (使用Naming.rebind()注册服务前rmiregistry必须启动)。
- (2) 忘记把参数和返回类型做成可序列化 (编译器不会检测到, 执行时才会发现)。
- (3) 忘记将stub类交给客户端。





(1)

下面有一些事件，请排列出在 Java RMI 应用程序上面发生的正确顺序。

(2)

(3)

(4)

(5)

(6)

68

(7)

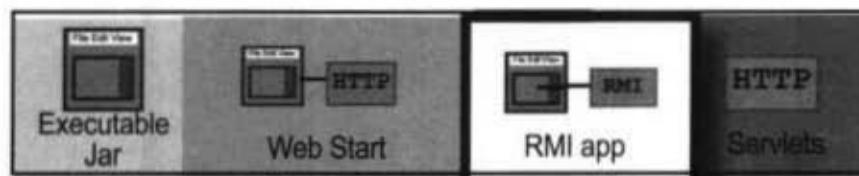


## 要点

- 在某堆上的对象无法进行另外堆上的对象引用。
  - Java Remote Method Invocation (RMI) 让你感觉上像是调用远程对象的方法，但其实不是。
  - 当客户端调用远程对象的方法时，其实是调用代理上的方法，此代理被称为stub。
  - stub是个处理低层网络细节的辅助性对象，它会把方法的调用包装起来送到服务器上。
  - 要创建远程服务的话，你就必须要以远程接口来启动。
  - 远程接口必须要extend过java.rmi.Remote这个接口，且所有的方法都必须声明RemoteException。
  - 你的远程服务会实现远程接口。
  - 远程服务应该要继承UnicastRemoteObject（技术上也有其他方法可以创建远程对象，但这是最简单的方式）。
  - 远程服务必须要声明RemoteException的构造函数（因为父类的构造函数声明了）。
  - 远程服务的对象必须要向 RMI registry 注册。
  - 使用静态的Naming.rebind( )来注册远程服务。
  - RMI registry必须在同一台机器上与远程服务一块执行，且必须在对象的注册之前启动。
  - 客户端会以 Naming.lookup()查询远程服务。
  - 几乎所有与RMI有关的都会抛出RemoteException（由编译器检查）。

## 是啊，有谁真的会用到RMI？





远程部署的RMI

完全在本机 介于两者之间 完全在远程

## 关于servlet

servlet是放在HTTP Web服务器上面运行的Java程序。当用户通过浏览器和网页交互时，请求（request）会送给网页服务器。如果请求需要Java的servlet时，服务器会执行或调用已经执行的servlet程序代码。servlet只是在服务器上运行的程序代码，执行出用户发出请求所要的结果（比如说将数据存进数据库）。如果你本来就熟悉使用Perl来编写的CGI，那你就会知道我们说的是什么。网页开发者使用CGI或servlet来操作用户提交（submit）给服务器的信息，像是发表在讨论版上的文章。

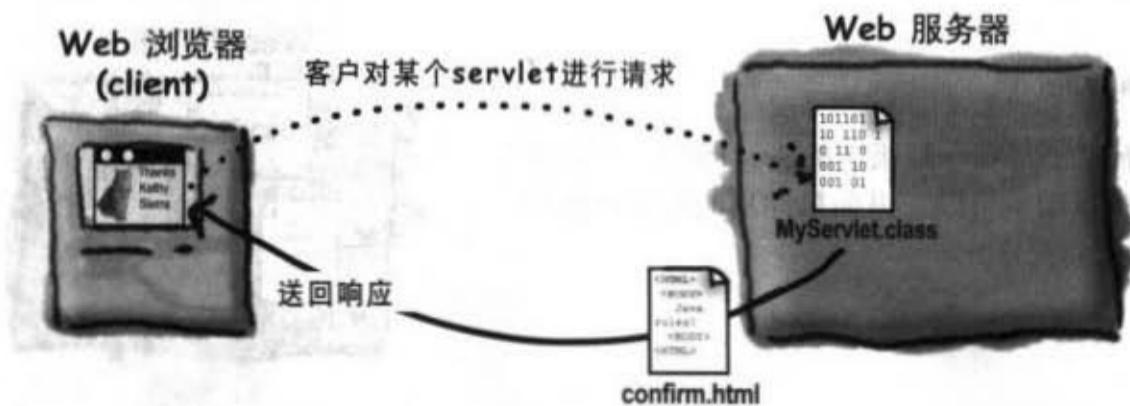
而servlet也可以使用RMI！

目前最常见的J2EE技术混合了servlet和EJB，前者是后者的用户。此时，servlet是通过RMI来与EJB通信的（但这与我们之前看到的程序有点不同）。

- ① 用户填写网页上的表格并提交。HTTP服务器收到请求，判断出是要给servlet的，就将请求传过去。



- ② servlet开始执行，把数据存到数据库中，然后组合出返回给浏览器的网页。



## 创建并执行servlet的步骤

### ① 找出可以存放servlet的地方

我们假设你已经有网页服务器可以运行servlet。最重要的事情是找到哪里可以存放servlet文件让服务器可以存取。如果服务器是向ISP租借的，技术支持的人应该会告诉你可以放在哪里，就像他们也会跟你说哪里可以放CGI一样。



### ② 取得servlets.jar并添加到classpath上

servlet并不是Java标准函数库的一部分，你需要包装成servlets.jar的文件。这可从java.sun.com下载，或者从默认好可执行Java的网页服务器（例如在apache.org网站的Apache Tomcat）。没有这些类你将无法编译servlet。



### ③ 通过extend过HttpServlet来编写servlet的类

servlet是个extend过HttpServlet（javax.servlet.http）的类。还有其他类型的servlet可以创建，但通常你只会使用HttpServlet。



```
public class MyServletA extends HttpServlet { ... }
```

### ④ 编写HTML来调用servlet

当用户点击引用到servlet的网页链接时，服务器会找到servlet并根据HTTP的GET、POST等命令调用适当的方法。



```
<a href="servlets/MyServletA">This is the most amazing servlet.</a>
```

### ⑤ 给服务器设定HTML网页和servlet

这就要看你的网页服务器而定了（哪一种版本的Servlets）。ISP可能会跟你说放到Servlets目录下面就可以。但如果你用的是最新版的Tomcat，可能要多花一点时间才搞定。



## 很简单的 servlet

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

除了io之外，还有两个包需要import，记住这两个
是需要另行下载的，它们不是Java标准库的一部分

public class MyServletA extends HttpServlet {
    // 一般的servlet会继承
    // HttpServlet

    // Override the doGet for simple HTTP
    // GET messages.
    // 服务器会调用这个方法来处理请求，然后以响
    // 应对象来响应
    public void doGet (HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        // 告诉服务器和浏览器响应结果的形式
        response.setContentType("text/html");

        // 此对象会给我们可写回结果的输出串流
        PrintWriter out = response.getWriter();

        String message = "If you're reading this, it worked!";
        out.println("<HTML><BODY>");
        out.println("<H1>" + message + "</H1>");
        out.println("</BODY></HTML>");
        out.close();
    }
}

```

我们print的对象是个网页！就像是  
HTML网页一样，只是内容是由程序  
的输出组成的

网页看起来会像是这样：

## 连接到 servlet 的 HTML 网页

```

<HTML>
<BODY>
    <a href="servlets/MyServletA">This is an amazing servlet.</a>
</BODY>
</HTML>

```

点击链接来启动  
servlet → This an amazing servlet.

**要点**

- servlet是完全在HTTP服务器上运行的Java程序。
- servlet用来处理与用户交互的网页程序。例如用户提交一些信息给服务器, servlet就可以处理信息并把特定的结果以网页形式返回给用户。
- 你需要servlets.jar文件中的servlet相关包才能编译出servlet。它不是标准函数库的一部分, 所以需要从java.sun.com或Web服务器供货商处取得(事实上Java 2 Enterprise Edition, 也就是J2EE就带有Servlet函数库)。
- 你必须要有支持servlet的Web服务器才能运行servlet, 比如apache.org的Tomcat。
- servlet必须放在特定的位置才能执行, 如果Web服务器是向ISP租借的, 它会告诉你应该放在哪个目录。
- 一般的servlet是继承HttpServlet并覆盖doGet()和doPost()来创建的。
- Web服务器会根据用户的请求来启动并调用servlet上对应的方法。
- servlet可以通过doGet()的响应参数取得输出串流来组成响应的网页。
- servlet要输出带有完整标识的HTML网页。

*there are no  
Dumb Questions*

**问:** 什么是JSP? 它跟servlet有什么关系?

**答:** JSP代表Java Server Pages。实际上Web服务器最终会把JSP转换成servlet, 但差别在于你所写出的是JSP。servlet是让你写出带有HTML输出的类, 而JSP刚好相反——你会写出带有Java程序的网页!

这样就能让你在编写一般HTML网页的时候还能够同时掌握动态内容的能力, 内嵌的程序代码(还有其他能够触发Java程序代码的标识)可以于执行时处理。也就是说网页内容有些部分可以在执行时由程序制作的。

JSP的主要好处在于能更容易地编写HTML部分而不会像servlet一样出现一堆难以阅读的print命令。想象一下如果要使用多个print命令才能响应很复杂的HTML网页是多么麻烦。

但对许多应用程序来说, 是无需使用到JSP的, 因为servlet不需要处理动态的响应, HTML内容其实很简单。或者是因为还有一些Web服务器并不支持JSP, 所以你只能使用servlet。

使用JSP的另一个好处是你可以将网页设计师与Java程序员的工作分离开来。不过这是宣传上的说法, 实际上还要突破学习特定Java标签的关卡, 以为这样就能天下太平是很不切实际的想法。至少不靠工具就想办到的想法是不正确的。好消息是开发工具已经出现了, 它能帮助设计师设计JSP而不用从头开始编写程序代码。

**问:** 花很多页讨论了RMI之后, 关于servlet的讨论就这样吗?

**答:** 没错。RMI是Java语言的一部分, 所有的RMI相关类也在标准函数库中。而servlet和JSP则不是Java语言的一部分, 也不被认为是标准的扩充套件。RMI可以在任何新版的Java虚拟机上面执行, 但servlet和JSP需要正确设定好的Web服务器和servlet的容器。这就是为什么说这个部分已经超出本书的范围。但你还是可以从《Head First Servlets & JSP》这本书寻求更多帮助。

## 闲着没事，来把专家术语学习机改成servlet

我们无法抗拒把第1章的专家术语学习机改成网络化的想法。servlet也是程序，Java程序可以调用其他程序，因此servlet能够调用学习机的方法。只要把学习机的类放到servlet的目录就行（学习机的程序代码在下一页）。



```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class KathyServlet extends HttpServlet {
    public void doGet (HttpServletRequest request, HttpServletResponse response)
                      throws ServletException, IOException {
        String title = "PhraseOMatic has generated the following phrase.';

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.println("<HTML><HEAD><TITLE>");
        out.println("PhraseOMatic");
        out.println("</TITLE></HEAD><BODY>");
        out.println("<H1>" + title + "</H1>"); ↓
        out.println("<P>" + PhraseOMatic.makePhrase());
        out.println("<P><a href=\"KathyServlet\">make another phrase</a></p>");
        out.println("</BODY></HTML>");

        out.close();
    }
}

```

看到没？servlet可以调用其他类的方法

## 适用于 servlet 的专家术语学习机

这跟第1章的版本稍有不同。原来是在 `main()` 中运行整个程序，且必须在命令行上重新执行程序才能收到新的组合。新版本会在调用静态的 `makePhrase()` 方法时返回 `String`。如此一来，你可以在任何程序中来调用此方法取得随机组合的字词。

注意：不要键入 `String[]` 中的破折号！实际上的程序并没有在两个双引号之间换行。

```
public class PhraseOMatic {
    public static String makePhrase() {

        // make three sets of words to choose from
        String[] wordListOne = {"24/7", "multi-Tier", "30,000 foot", "B-to-B", "win-win", "front-end", "web-based", "pervasive", "smart", "six-sigma", "critical-path", "dynamic"};

        String[] wordListTwo = {"empowered", "sticky", "valued-added", "oriented", "centric", "distributed", "clustered", "branded", "outside-the-box", "positioned", "networked", "focused", "leveraged", "aligned", "targeted", "shared", "cooperative", "accelerated"};

        String[] wordListThree = {"process", "tipping point", "solution", "architecture", "core competency", "strategy", "mindshare", "portal", "space", "vision", "paradigm", "mission"};

        // find out how many words are in each list
        int oneLength = wordListOne.length;
        int twoLength = wordListTwo.length;
        int threeLength = wordListThree.length;

        // generate three random numbers, to pull random words from each list
        int rand1 = (int) (Math.random() * oneLength);
        int rand2 = (int) (Math.random() * twoLength);
        int rand3 = (int) (Math.random() * threeLength);

        // now build a phrase
        String phrase = wordListOne[rand1] + " " + wordListTwo[rand2] + " " + wordListThree[rand3];

        // now return it
        return ("What we need is a " + phrase);
    }
}
```

## Enterprise JavaBeans: 打了类 固醇的RMI

RMI很适合编写并运行远程服务。但你不会单独使用RMI来执行Amazon或eBay网站服务。对大型的企业级应用程序来说，你需要更多更好的功能。你需要交易管理、大量并发处理（全世界的人一起上来抢购“哈利·波特——消失的蜜饯”）、安全性和数据库管理等。为此，你会需要enterprise application server。

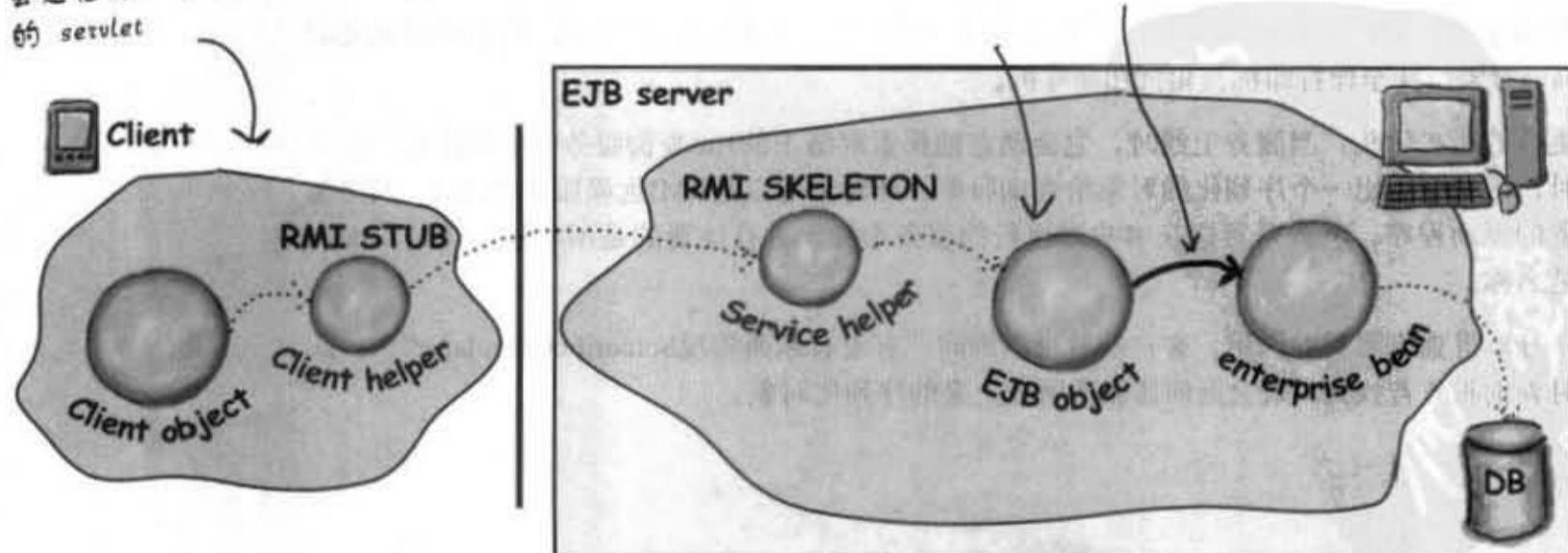
用Java术语来说，这就是Java 2 Enterprise Edition (J2EE) 服务器。J2EE服务器包括了Web服务器和Enterprise JavaBeans (EJB) 服务器。就跟servlet一样，EJB已经超过了本书的范围，且EJB也无法用简短的范例展示，但我们会对它的工作方式稍加说明。

（译注：业界已经出现一股反对EJB的声音，理由就跟你不会开十八轮大货车去市场买菜一样，工具和技术并不是越大越重就越合适。）

EJB服务器具有一组你光靠RMI不会有服务，比如交易管理、安全性、并发性、数据库和网络功能等。

EJB服务器作用于RMI调用和服务层之间。

客户端可以是任何装置，但通常会是在同一个J2EE服务器上执行的servlet



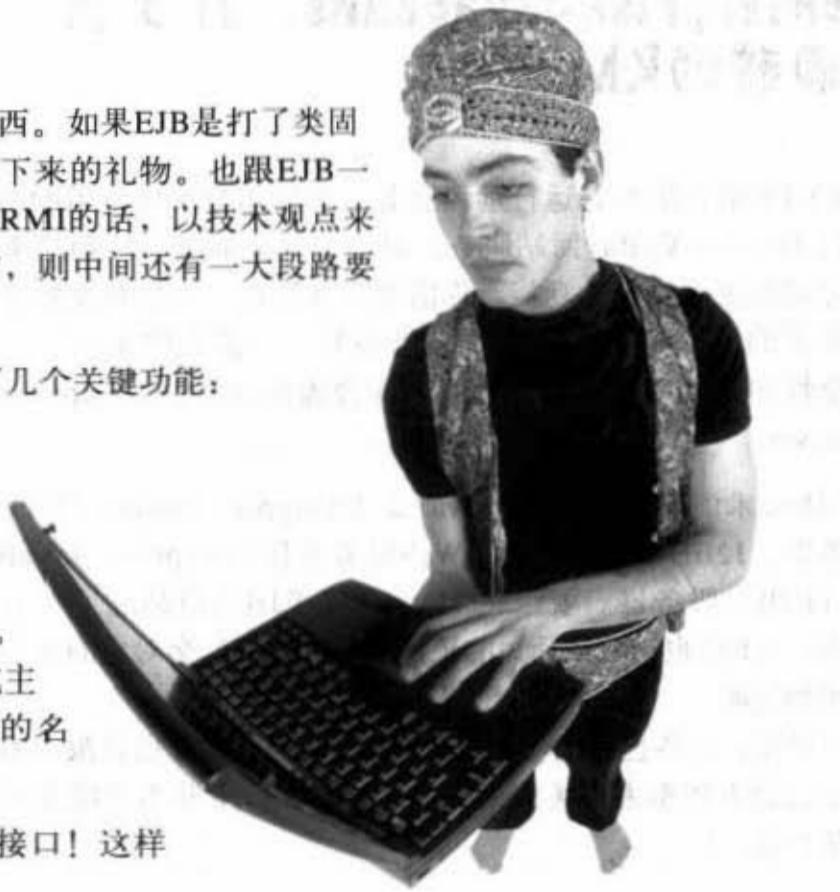
这只是完整EJB架构的一部分

## 最后，来点Jini魔法

我们是超爱Jini的。我们认为Jini应该是Java最棒的东西。如果EJB是打了类固醇的RMI，Jini就是长出翅膀的RMI。真是个天上掉下来的礼物。也跟EJB一样，我们无法在这本书讨论Jini的细节，但若你搞定RMI的话，以技术观点来说那就已经接近一半真相的一半。以观念的理解而言，则中间还有一大段路要走。

Jini也是使用RMI（虽然也可以用别的协议），但多了几个关键功能：

- (1) adaptive discovery（自适应探索）。
- (2) self-healing networks（自恢复网络）。



要知道RMI的客户端得先取得远程服务的地址和名称。

客户端的查询程序代码就要带有远程服务的IP地址或主机名（因为RMI registry就在上面）以及服务所注册的名称。

但使用Jini时，用户只需知道一件事：服务所实现的接口！这样就行。

哪怎么找到的呢？秘诀就在于Jini的lookup service。Jini的查询服务比RMI registry更强更有适应性。因为Jini会在网络上自动的广告。当查询服务上线时，它会使用IP组播技术送出信息给整个网络说：“大爷我在这里，想找东西就问我”。

不只是这样。如果客户端在查询服务已经广播之后上线，客户端也可以发出信息给整个网络说：“那个谁在不在啊？”。

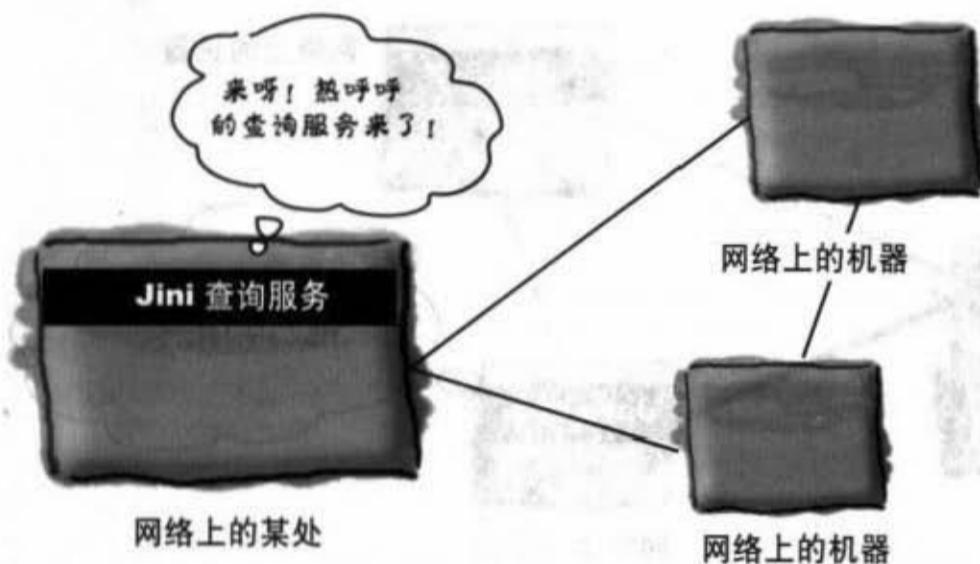
其实你感兴趣的不是查询服务，而是已经注册的服务。像RMI远程服务、其他可序列化的Java对象，甚至像打印机、相机和咖啡机。

更棒的还在后头：当服务上线时，它会动态地探索网络上的Jini查询服务并申请注册。注册时，服务会送出一个序列化的对象给查询服务。此对象可以是RMI远程服务的stub、网络装置的驱动程序，甚或是可以在客户端执行的服务本身，并且注册的是所实现的接口，而不是名称。

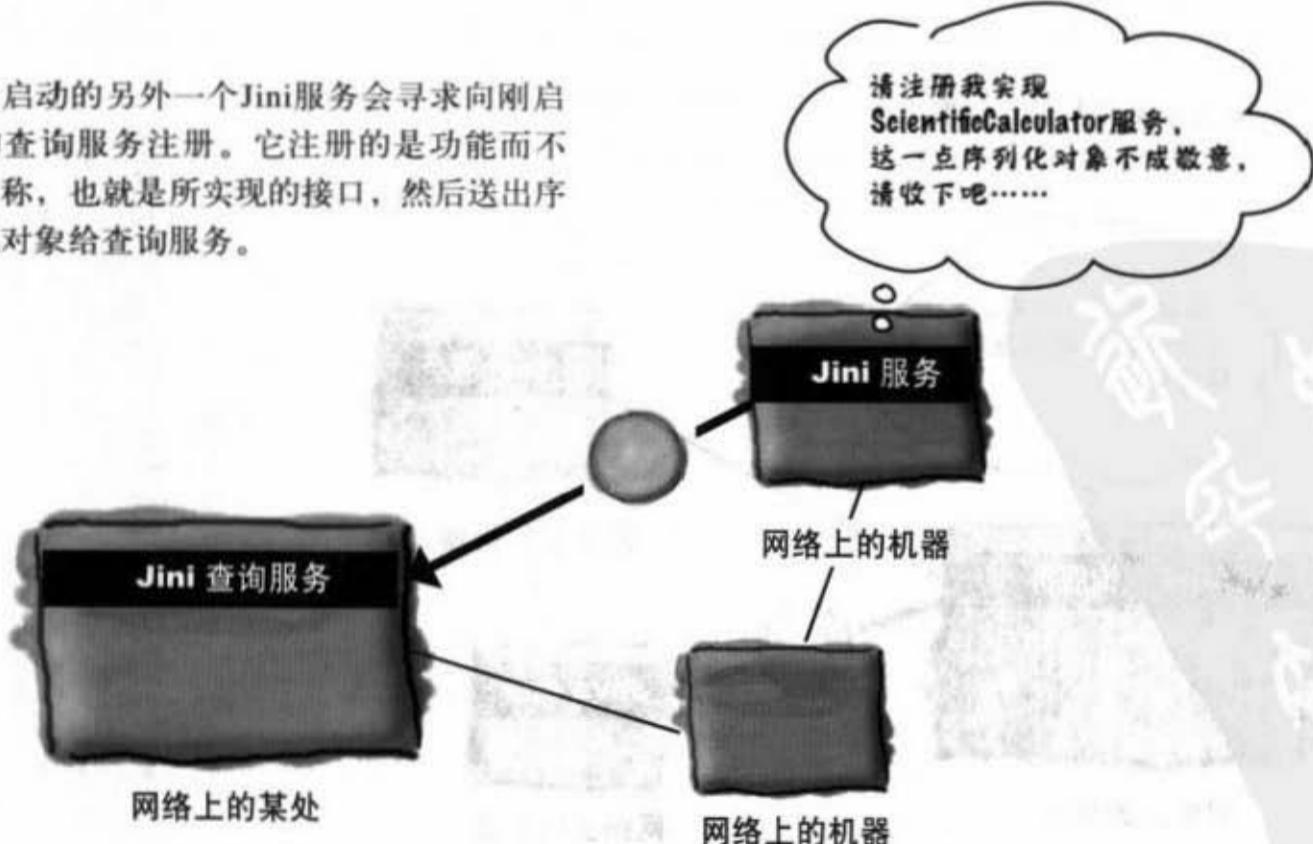
一旦取得查询服务的引用，客户端就可以询问“有没有东西实现ScientificCalculator?”。此时查询服务若找到，就会返回该服务所放上来的序列化对象。

## 自适应探索的运作

- ① Jini查询服务在网络上启动，并使用IP组播技术为自己做宣传。

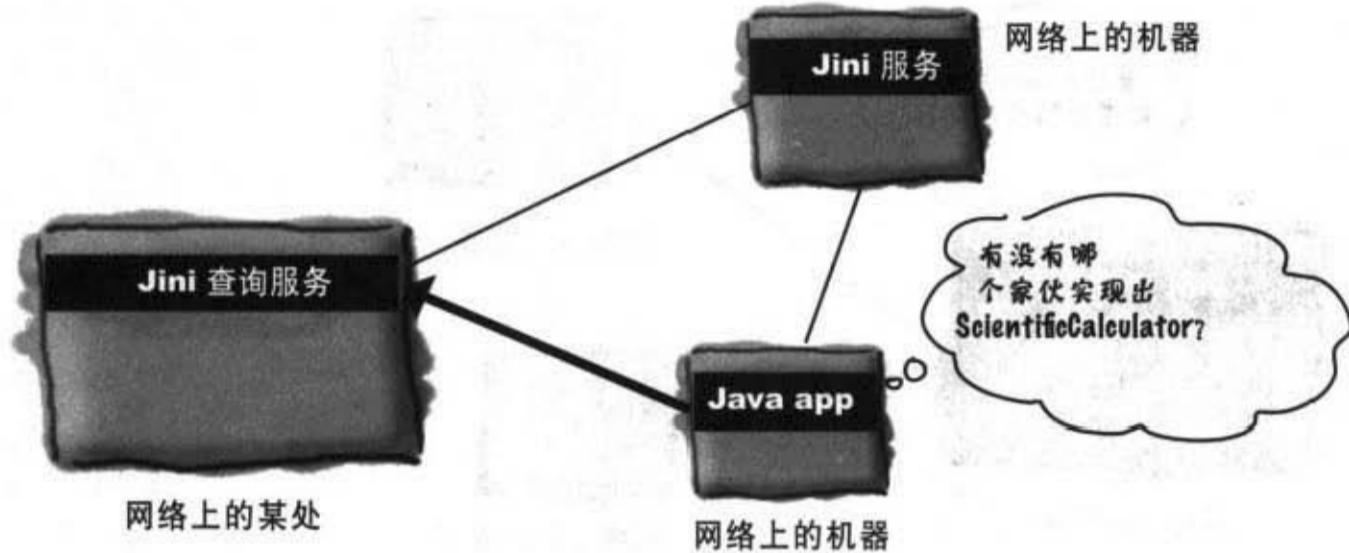


- ② 已经启动的另外一个Jini服务会寻求向刚启动的查询服务注册。它注册的是功能而不是名称，也就是所实现的接口，然后送出序列化对象给查询服务。

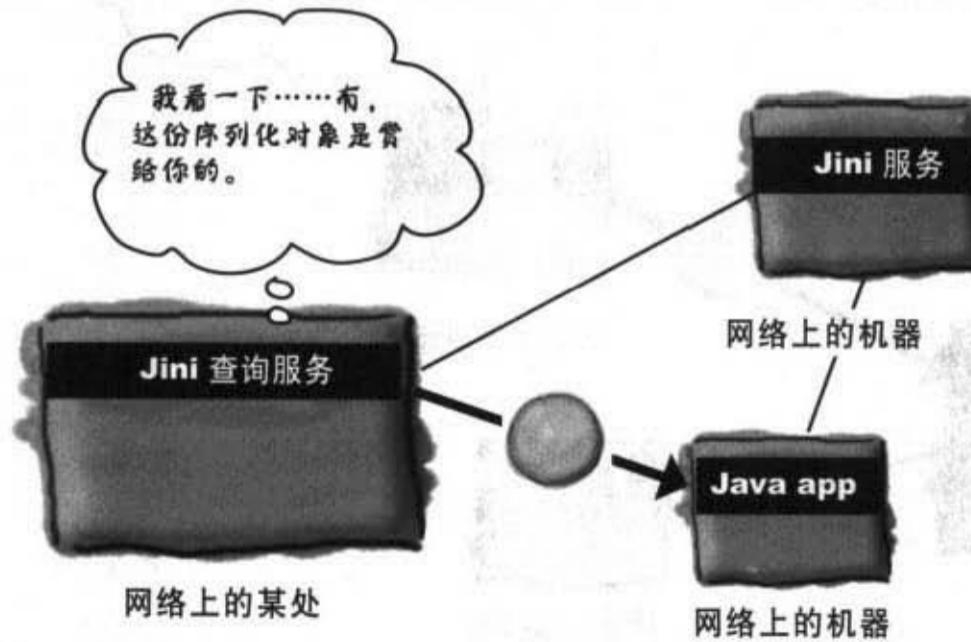


## 自适应探索的运作（续）

- ③ 网络客户想要取得实现ScientificCalculator的东西，可是不知道哪里有，所以就问查询服务。

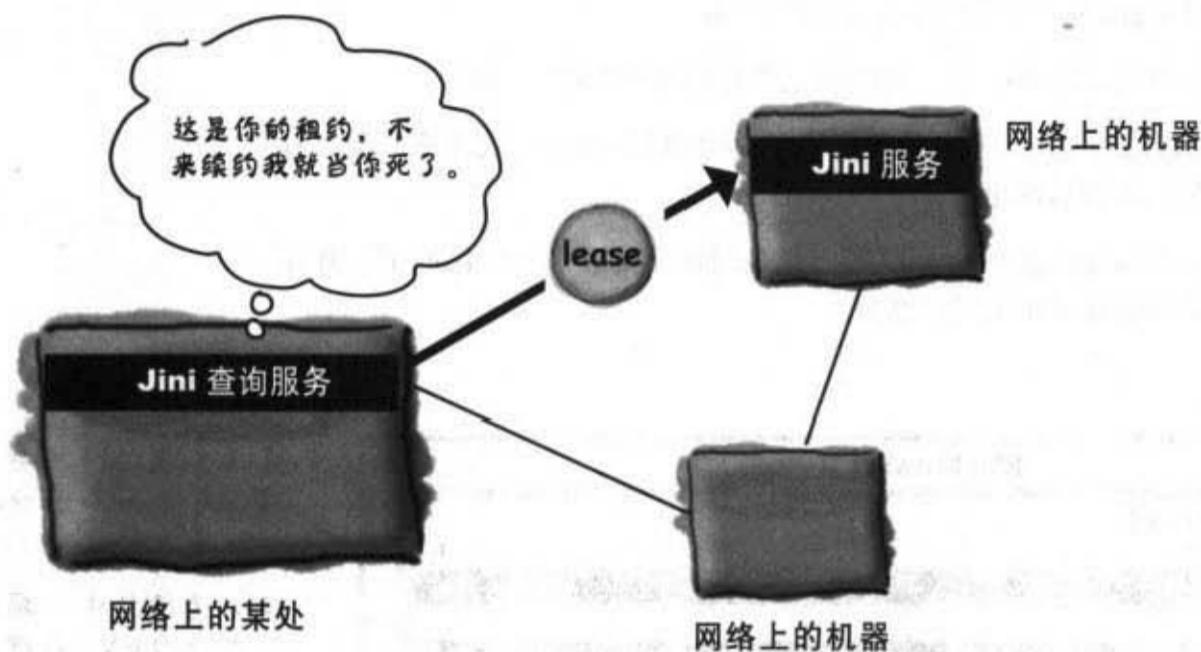


- ④ 查询服务响应查询的结果。

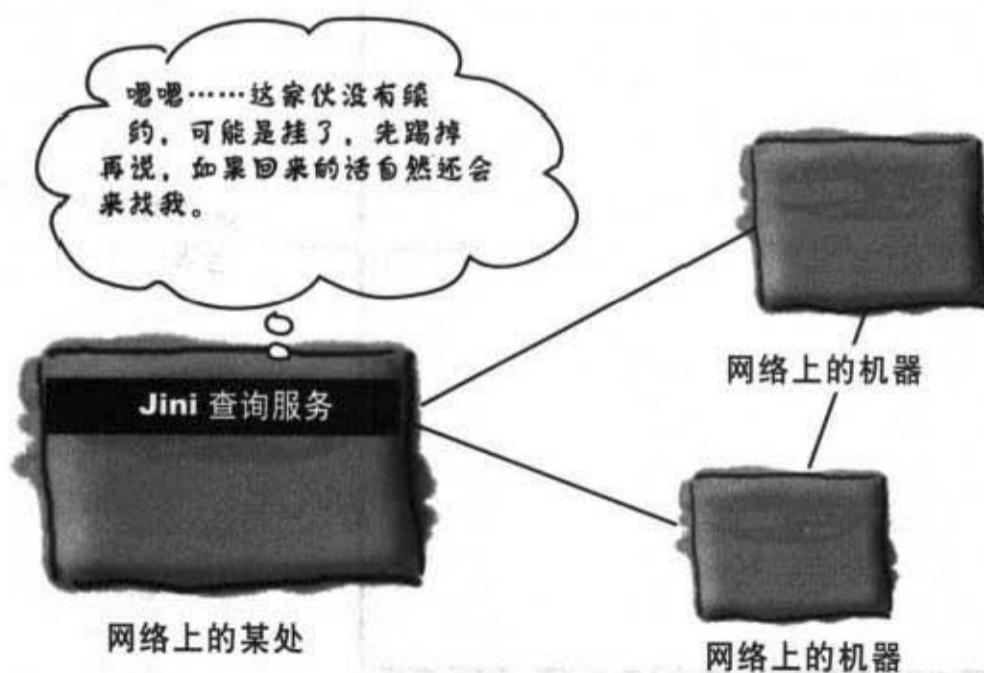


## 自恢复网络的运作

- ① 某个Jini服务要求注册，查询服务回给一份租约。新注册的服务必须要定期更新租约，不然查询服务会假设此服务已经离线了。查询服务会力求呈现精确完整的可用服务网络状态。



- ② 因为关机所以服务离线，因此也没有更新租约，查询服务就把它踢掉。



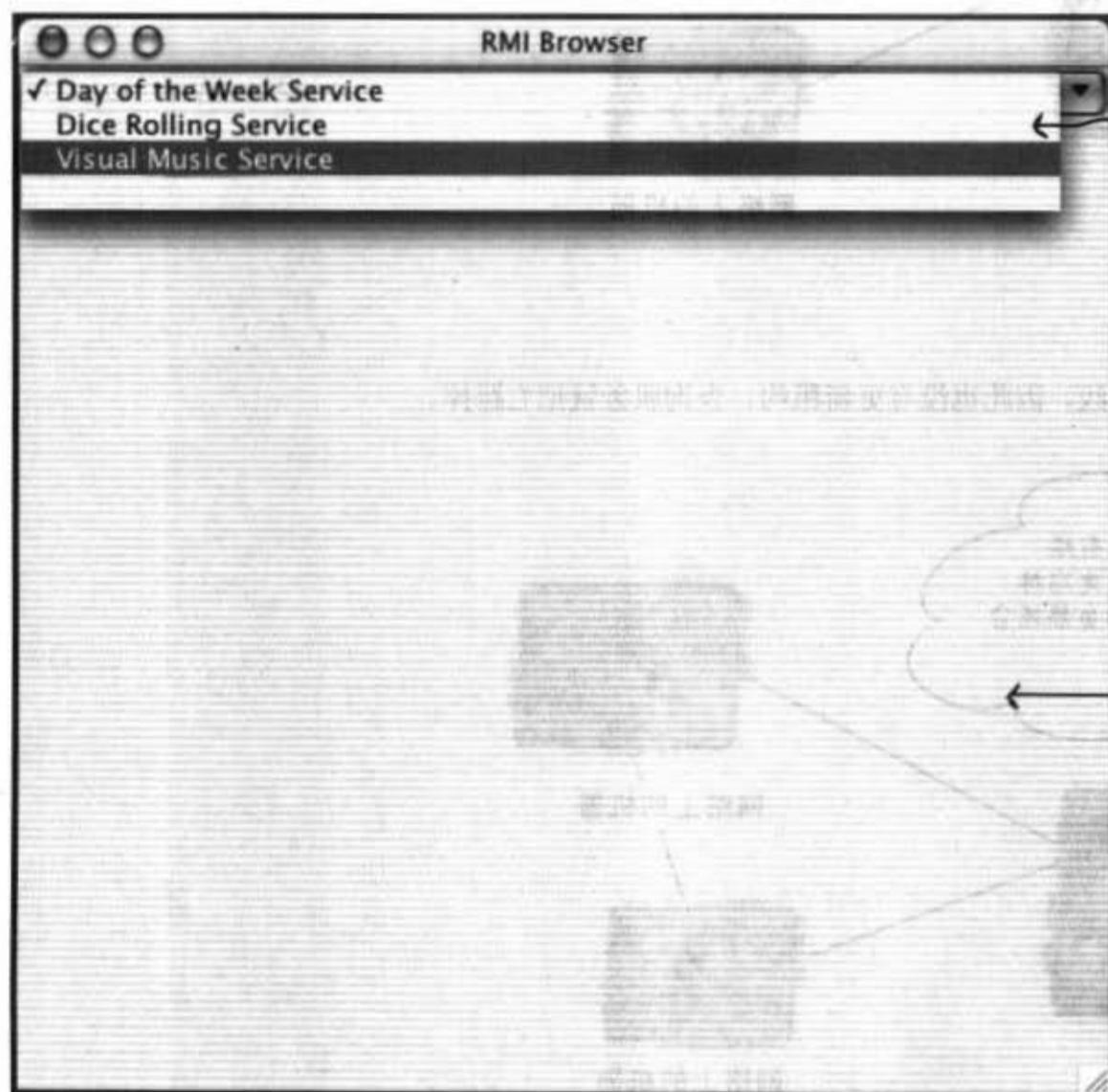
## 终极任务：通用服务浏览器

我们要做一个没有Jini功能的程序，但很容易实现。它会让你体验Jini，却只有用到RMI。事实上我们的程序与Jini应用程序的主要差别只在于服务是如何探索的。相对于Jini的查询服务会自动地对所处的网络做广告，我们使用的是必须与远程服务在同一台机器上执行的RMI registry，这当然不会自动的声明。

且服务也不会自动地向查询服务做注册，我们必须将它注册给RMI registry。

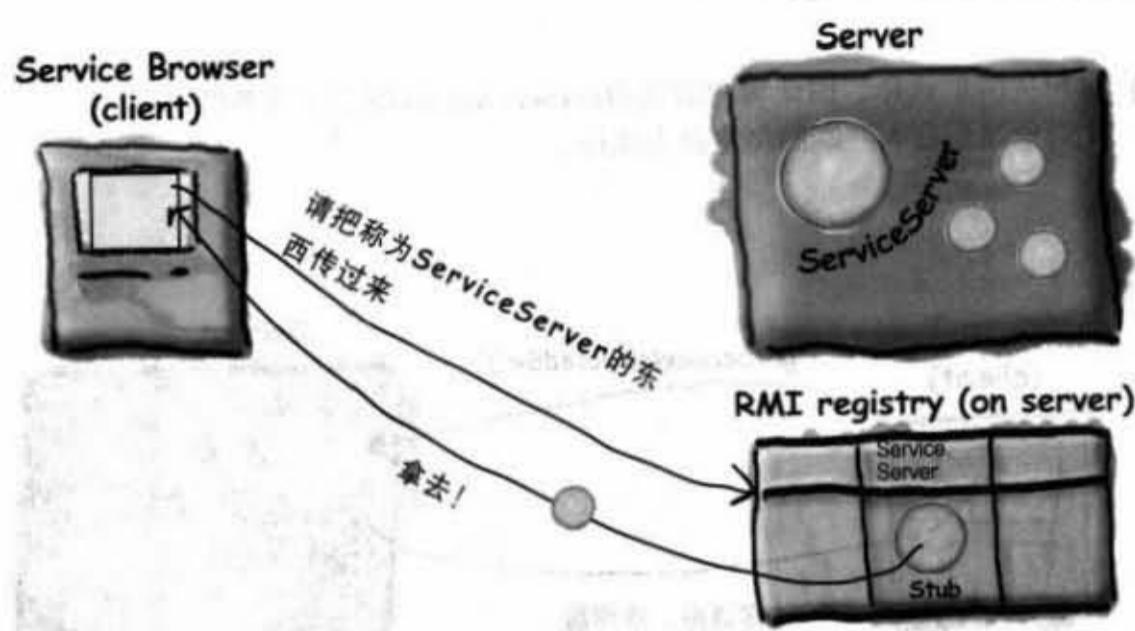
一旦用户在RMI registry找到服务，应用程序其余的部分就跟Jini的方式几乎一模一样（当然还少了租约和续约机制这回事）。

通用服务浏览器就像是个特殊化的网页浏览器，只是所展示的并非HTML网页。此服务浏览器会下载并显示出交互的Java图形界面。



## 它的运作方式：

- ① 用户启动并查询在 RMI registry 上注册为 ServiceServer 的服务，然后取回 stub。



- ② 客户端对 stub 调用 `getServiceList()`。它会返回服务的数组。

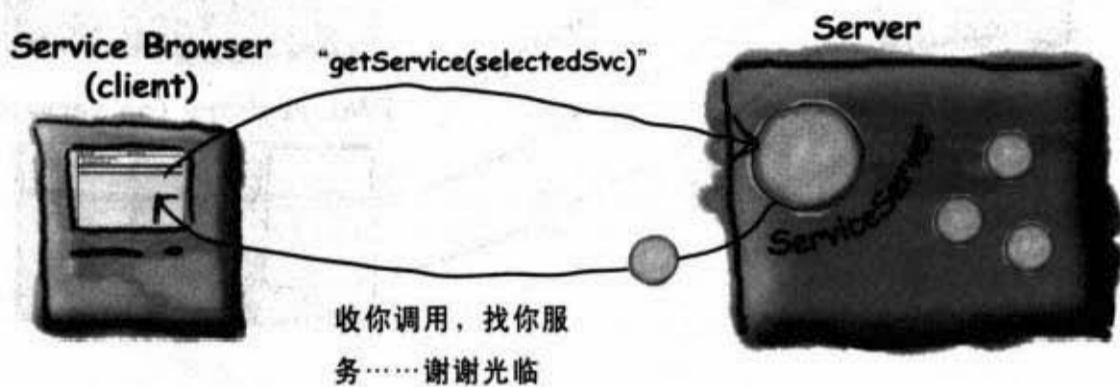


- ③ 客户端以 GUI 显示出服务对象的清单。



## 运作方式（续）：

- ④ 用户从清单选择，因此客户端调用getService()取得实际服务的序列化对象然后在客户端的浏览器上执行。



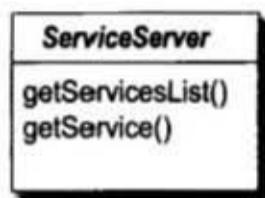
- ⑤ 客户端调用刚取得序列化对象的getGuiPanel()。此服务的GUI会显示在浏览器中，且用户可与它在本机上交互。此时就不需要远程服务了。



## 类与接口

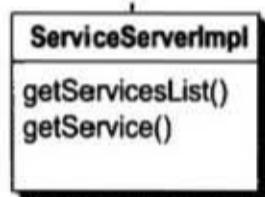
- ServiceServer这个接口实现了Remote。

那是给远程服务用的RMI普通接口（让远程服务写出取得服务清单的方法）。



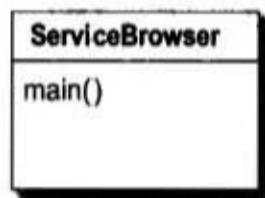
- ServiceServerImpl这个类实现了ServiceServer。

这是实际的RMI远程服务（有继承UnicastRemoteObject），它的任务是初始化并存储全部的服务（会被传送给客户端的东西），并把自己登记给RMI registry。



- ServiceBrowser。

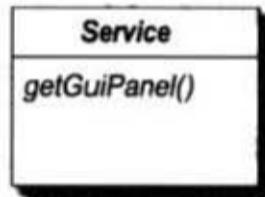
这是客户端的类，它创建出很简单的GUI，在RMI registry中查询取得ServiceServer的stub，然后调用它的远程服务取得服务清单并显示在GUI上面。



- Service。

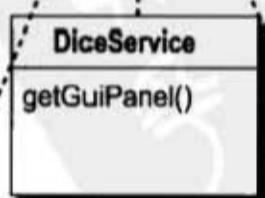
它是最关键的部分。这个简单的接口只有一个getGuiPanel()方法，每个要传送给客户端的服务都得实现这个接口。这样才会让浏览器“通用”！实现这个接口之后，服务才能克服客户端完全不知道服务实际类的问题。至少来的服务会有getGuiPanel()方法。

客户端调用getService(selectedSvc)后取得序列化对象，然后在不知道是什么类的情况下调用一定会有的getGuiPanel()来取得JPanel，然后放到浏览器上开始与用户交互。



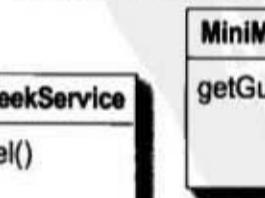
- DiceService实现Service。

需要骰子吗？没问题，取用这个服务就会出现虚拟的骰子。



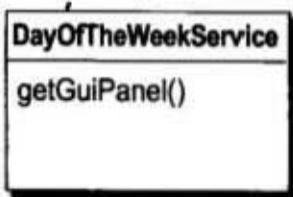
- 实现Service的MiniMusicService。

还记得我们之前做过音乐录像带吗？我们把它改装成服务，你就可以不停地播放听了想打人的音乐。



- DayOfTheWeekService实现Service。

输入你的生日就可以知道今天是星期几。



## interface ServiceServer (远程接口)

```
import java.rmi.*;
public interface ServiceServer extends Remote {
    Object[] getServiceList() throws RemoteException;
    Service getService(Object serviceKey) throws RemoteException;
}
```

一般的RMI接口，定义出两个远程服务要实现的方法

## interface Service (GUI 服务要实现的部分)

```
import javax.swing.*;
import java.io.*;
public interface Service extends Serializable {
    public JPanel getGuiPanel();
}
```

定义任何通用服务都得要实现的  
getGuiPanel()这个方法，因为继承  
Serializable，所以能够自动地序列化。  
而这也是又通过网络传递服务所必须的  
机制

## class ServiceServerImpl (远程的实现)

```

import java.rmi.*;
import java.util.*;
import java.rmi.server.*;

```

一般的RMI实现

```

public class ServiceServerImpl extends UnicastRemoteObject implements ServiceServer {
    HashMap serviceList;  服务会被存储在HashMap集合中
}

```

```

public ServiceServerImpl() throws RemoteException {
    setUpServices();
}

```

```

private void setUpServices() {
    serviceList = new HashMap();
    serviceList.put("Dice Rolling Service", new DiceService());
    serviceList.put("Day of the Week Service", new DayOfTheWeekService());
    serviceList.put("Visual Music Service", new MiniMusicService());
}

```

构造函数被调用时会将实际的通用服务初始化

创建服务并将String名称放进HashMap中

```

public Object[] getServiceList() {
    System.out.println("in remote");
    return serviceList.keySet().toArray();
}

```

客户端会调用它以取得服务的清单，我们会送出Object的数组，只带有HashMap的key，实际服务会到用户要求时才通过getService()送出

```

public Service getService(Object serviceKey) throws RemoteException {
    Service theService = (Service) serviceList.get(serviceKey);
    return theService;
}

```

这个方法会通过key名称来返回HashMap中相对应的服务

```

public static void main (String[] args) {
    try {
        Naming.rebind("ServiceServer", new ServiceServerImpl());
    } catch (Exception ex) {
        ex.printStackTrace();
    }
    System.out.println("Remote service is running");
}

```

你现在的位置

641

## ServiceBrowser代码

```
class ServiceBrowser (用户端)

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

public class ServiceBrowser {

    JPanel mainPanel;
    JComboBox serviceList;
    ServiceServer server;

    public void buildGUI() {
        JFrame frame = new JFrame("RMI Browser");
        mainPanel = new JPanel();
        frame.getContentPane().add(BorderLayout.CENTER, mainPanel);

        Object[] services = getServiceList(); ← 此方法执行RMI registry查询，取得stub并调用getServiceList()

        serviceList = new JComboBox(services); ← 把此服务添加到JComboBox里面，JComboBox知道如何显示数组中的字符串

        frame.getContentPane().add(BorderLayout.NORTH, serviceList);

        serviceList.addActionListener(new MyListListener());

        frame.setSize(500,500);
        frame.setVisible(true);
    }

    void loadService(Object serviceSelection) {
        try {
            Service svc = server.getService(serviceSelection);

            mainPanel.removeAll(); ← 把实际的服务添加到GUI上的mainPanel中
            mainPanel.add(svc.getGuiPanel());
            mainPanel.validate();
            mainPanel.repaint();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

```

Object[] getServicesList() {
    Object obj = null;
    Object[] services = null;
    try {
        obj = Naming.lookup("rmi://127.0.0.1/ServiceServer");
    }
    catch(Exception ex) {
        ex.printStackTrace();
    }
    server = (ServiceServer) obj;
    try {
        services = server.getServiceList();
    } catch(Exception ex) {
        ex.printStackTrace();
    }
    return services;
}

class MyListListener implements ActionListener {
    public void actionPerformed(ActionEvent ev) {
        Object selection = serviceList.getSelectedItem();
        loadService(selection);
    }
}

public static void main(String[] args) {
    new ServiceBrowser().buildGUI();
}

```

执行RMI查询，取得stub

将stub转换成remote interface的类型，如此才能调用它的getServiceList()

它会返回Object的数组

用户点击ComboBox的项目后会到这里来，因此就会把相对应的服务加载

## class DiceService (实现Service的通用服务)

```

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

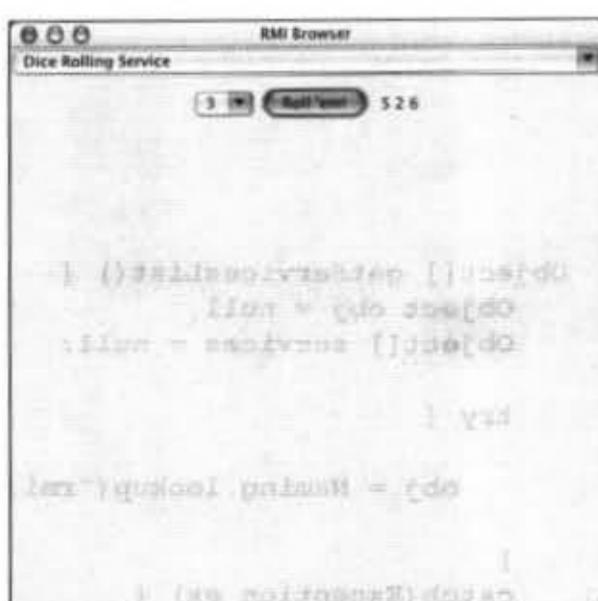
public class DiceService implements Service {
    JLabel label;
    JComboBox numOfDice;

    public JPanel getGuiPanel() {
        JPanel panel = new JPanel();
        JButton button = new JButton("Roll 'em!");
        String[] choices = {"1", "2", "3", "4", "5"};
        numOfDice = new JComboBox(choices);
        label = new JLabel("dice values here");
        button.addActionListener(new RollEmListener());
        panel.add(numOfDice);
        panel.add(button);
        panel.add(label);
        return panel;
    }

    public class RollEmListener implements ActionListener {
        public void actionPerformed(ActionEvent ev) {
            // roll the dice
            String diceOutput = "";
            String selection = (String) numOfDice.getSelectedItem();
            int numOfDiceToRoll = Integer.parseInt(selection);
            for (int i = 0; i < numOfDiceToRoll; i++) {
                int r = (int) ((Math.random() * 6) + 1);
                diceOutput += (" " + r);
            }
            label.setText(diceOutput);
        }
    }
}

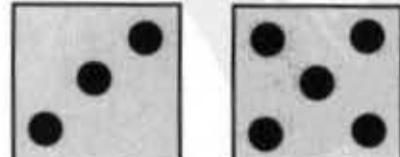
```

这是很重要的方法！客户端会调用这个定义在Service中的方法来创建实际的骰子



### Sharpen your pencil

想想看要如何改善DiceService。一个贴心的小建议：做成图形化的骰子。



## class MiniMusicService (实现Service的通用服务)

```

import javax.sound.midi.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class MiniMusicService implements Service {
    MyDrawPanel myPanel;
    public JPanel getGuiPanel() {
        JPanel mainPanel = new JPanel();
        myPanel = new MyDrawPanel();
        JButton playItButton = new JButton("Play it");
        playItButton.addActionListener(new PlayItListener());
        mainPanel.add(myPanel);
        mainPanel.add(playItButton);
        return mainPanel;
    }

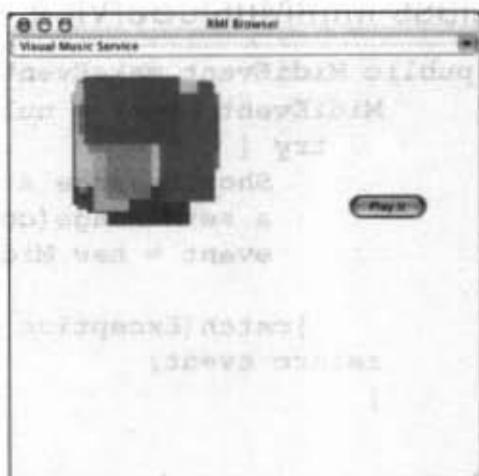
    public class PlayItListener implements ActionListener { 程序代码来自第12章
        public void actionPerformed(ActionEvent ev) {
            try {
                Sequencer sequencer = MidiSystem.getSequencer();
                sequencer.open();

                sequencer.addControllerEventListener(myPanel, new int[] {127});
                Sequence seq = new Sequence(Sequence.PPQ, 4);
                Track track = seq.createTrack();

                for (int i = 0; i < 100; i += 4) {
                    int rNum = (int) ((Math.random() * 50) + 1);
                    if (rNum < 38) { // so now only do it if num < 38 (75% of the time)
                        track.add(makeEvent(144, 1, rNum, 100, i));
                        track.add(makeEvent(176, 1, 127, 0, i));
                        track.add(makeEvent(128, 1, rNum, 100, i + 2));
                    }
                } // 结束循环

                sequencer.setSequence(seq);
                sequencer.start();
                sequencer.setTempoInBPM(220);
            } catch (Exception ex) {ex.printStackTrace();}
        }
    } // 关闭actionperformed
} // 关闭内部类

```



就是它会显示出button并  
画出一堆方块

MiniMusicService

class MiniMusicService (续)

```
public MidiEvent makeEvent(int comd, int chan, int one, int two, int tick) {  
    MidiEvent event = null;  
    try {  
        ShortMessage a = new ShortMessage();  
        a.setMessage(comd, chan, one, two);  
        event = new MidiEvent(a, tick);  
  
    } catch (Exception e) {}  
    return event;  
}
```

```
class MyDrawPanel extends JPanel implements ControllerEventListener {
```

```
// only if we got an event do we want to paint  
boolean msg = false;
```

```
public void controlChange(ShortMessage event) {  
    msg = true;  
    repaint();  
}
```

```
public Dimension getPreferredSize() {  
    return new Dimension(300,300);  
}
```

```
public void paintComponent(Graphics g) {  
    if (msg) {  
  
        Graphics2D g2 = (Graphics2D) g;  
  
        int r = (int) (Math.random() * 250);  
        int gr = (int) (Math.random() * 250);  
        int b = (int) (Math.random() * 250);  
  
        g.setColor(new Color(r, gr, b));  
  
        int ht = (int) ((Math.random() * 120) + 10);  
        int width = (int) ((Math.random() * 120) + 10);  
  
        int x = (int) ((Math.random() * 40) + 10);  
        int y = (int) ((Math.random() * 40) + 10);  
  
        g.fillRect(x, y, ht, width);  
        msg = false;  
    } // close if  
} // close method  
} // close inner class  
} // close class
```

整页都跟以前是一样的

## class DayOfTheWeekService (实现Service的通用服务)

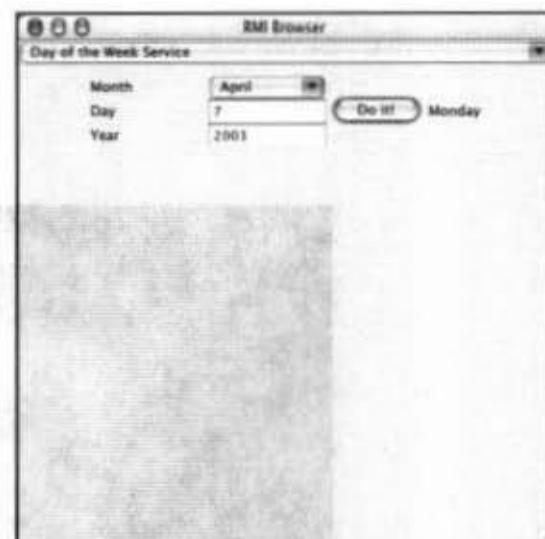
```

import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.io.*;
import java.util.*;
import java.text.*;

public class DayOfTheWeekService implements Service {
    JLabel outputLabel;
    JComboBox month;
    JTextField day;
    JTextField year;
    public JPanel getGuiPanel() {
        JPanel panel = new JPanel();
        JButton button = new JButton("Do it!");
        button.addActionListener(new DoItListener());
        outputLabel = new JLabel("date appears here");
        DateFormatSymbols dateStuff = new DateFormatSymbols();
        month = new JComboBox(dateStuff.getMonths());
        day = new JTextField(8);
        year = new JTextField(8);
        JPanel inputPanel = new JPanel(new GridLayout(3,2));
        inputPanel.add(new JLabel("Month"));
        inputPanel.add(month);
        inputPanel.add(new JLabel("Day"));
        inputPanel.add(day);
        inputPanel.add(new JLabel("Year"));
        inputPanel.add(year);
        panel.add(inputPanel);
        panel.add(button);
        panel.add(outputLabel);
        return panel;
    }

    public class DoItListener implements ActionListener {
        public void actionPerformed(ActionEvent ev) {
            int monthNum = month.getSelectedIndex();
            int dayNum = Integer.parseInt(day.getText());
            int yearNum = Integer.parseInt(year.getText());
            Calendar c = Calendar.getInstance();
            c.set(Calendar.MONTH, monthNum);
            c.set(Calendar.DAY_OF_MONTH, dayNum);
            c.set(Calendar.YEAR, yearNum);
            Date date = c.getTime();
            String dayOfWeek = (new SimpleDateFormat("EEEE")).format(date);
            outputLabel.setText(dayOfWeek);
        }
    }
}

```



创建GUI的方法

日期格式化的工作见第10章



恭喜你！

你办到了！

当然，后面还有两个附录。

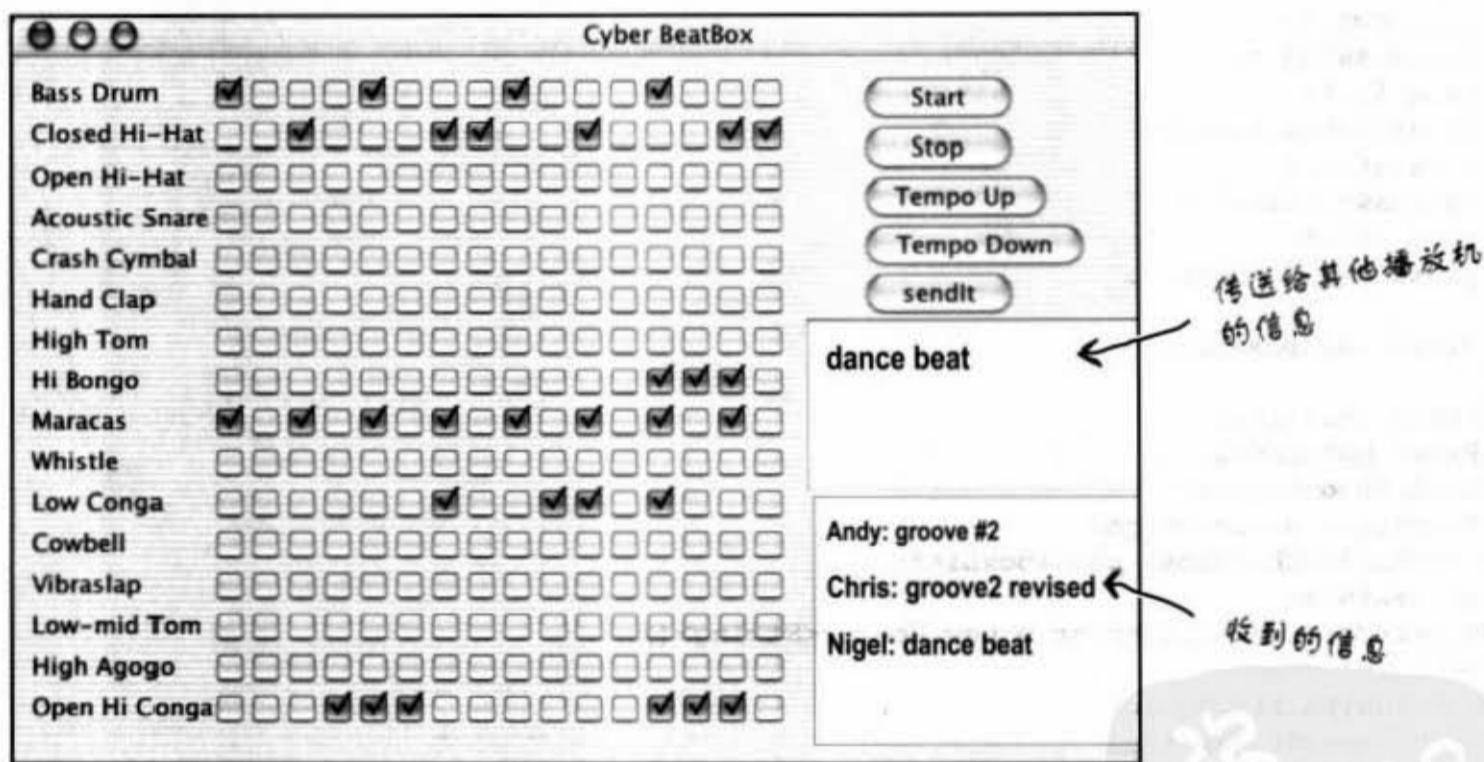
一个索引。

一个网站……

别想偷懒。



# 附录A： 程序料理决定版



这是 BeatBox 的完成版！

它会连接到 MusicServer 来让你传送并接收其他用户的信  
息以及节奏样式。

## BeatBox客户端决定版

这些程序大部分都跟章节内容的程序一样，所以我们没有重复加上注释。

新增加的部分包括了：

- (1) GUI：加入两个新的组件来显示收到的信息。
- (2) 网络：BeatBox会连接到服务器取得输出入串流。
- (3) 线程：reader这个类会持续地从服务器读取信息。

```
import java.awt.*;
import javax.swing.*;
import java.io.*;
import javax.sound.midi.*;
import java.util.*;
import java.awt.event.*;
import java.net.*;
import javax.swing.event.*;
```

```
public class BeatBoxFinal {

    JFrame theFrame;
    JPanel mainPanel;
    JList incomingList;
    JTextField userMessage;
    ArrayList<JCheckBox> checkboxList;
    int nextNum;
    Vector<String> listVector = new Vector<String>();
    String userName;
    ObjectOutputStream out;
    ObjectInputStream in;
    HashMap<String, boolean[]> otherSeqsMap = new HashMap<String, boolean[]>();

    Sequencer sequencer;
    Sequence sequence;
    Sequence mySequence = null;
    Track track;

    String[] instrumentNames = {"Bass Drum", "Closed Hi-Hat", "Open Hi-Hat", "Acoustic Snare", "Crash Cymbal", "Hand Clap", "High Tom", "Hi Bongo", "Maracas", "Whistle", "Low Conga", "Cowbell", "Vibraslap", "Low-mid Tom", "High Agogo", "Open Hi Conga"};
    int[] instruments = {35, 42, 46, 38, 49, 39, 50, 60, 70, 72, 64, 56, 58, 47, 67, 63};
```

```

public static void main (String[] args) {
    new BeatBoxFinal().startUp(args[0]); // args[0] is your user ID/screen name
}

public void startUp(String name) {
    userName = name;
    // open connection to the server
    try {
        Socket sock = new Socket("127.0.0.1", 4242);           // 作为显示名称的命令栏参数, 例如:
        out = new ObjectOutputStream(sock.getOutputStream());    %java BeatBoxFinal theFlash
        in = new ObjectInputStream(sock.getInputStream());
        Thread remote = new Thread(new RemoteReader());
        remote.start();
    } catch (Exception ex) {
        System.out.println("couldn't connect - you'll have to play alone.");
    }
    setUpMidi();
    buildGUI();
} // close startUp

public void buildGUI() {                                     GUI 程序

    theFrame = new JFrame("Cyber BeatBox");
    BorderLayout layout = new BorderLayout();
    JPanel background = new JPanel(layout);
    background.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));

    checkboxList = new ArrayList<JCheckBox>();

    Box buttonBox = new Box(BoxLayout.Y_AXIS);
    JButton start = new JButton("Start");
    start.addActionListener(new MyStartListener());
    buttonBox.add(start);

    JButton stop = new JButton("Stop");
    stop.addActionListener(new MyStopListener());
    buttonBox.add(stop);

    JButton upTempo = new JButton("Tempo Up");
    upTempo.addActionListener(new MyUpTempoListener());
    buttonBox.add(upTempo);

    JButton downTempo = new JButton("Tempo Down");
    downTempo.addActionListener(new MyDownTempoListener());
    buttonBox.add(downTempo);

    JButton sendIt = new JButton("sendIt");
    sendIt.addActionListener(new MySendListener());
    buttonBox.add(sendIt);

    userMessage = new JTextField();
}

```

```

buttonBox.add(userMessage);

incomingList = new JList();
incomingList.addListSelectionListener(new MyListSelectionListener());
incomingList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
JScrollPane theList = new JScrollPane(incomingList);
buttonBox.add(theList);
incomingList.setListData(listVector); // no data to start with

Box nameBox = new Box(BoxLayout.Y_AXIS);
for (int i = 0; i < 16; i++) {
    nameBox.add(new Label(instrumentNames[i]));
}

background.add(BorderLayout.EAST, buttonBox);
background.add(BorderLayout.WEST, nameBox);

theFrame.getContentPane().add(background);
GridLayout grid = new GridLayout(16,16);
grid.setVgap(1);
grid.setHgap(2);
mainPanel = new JPanel(grid);
background.add(BorderLayout.CENTER, mainPanel);

for (int i = 0; i < 256; i++) {
    JCheckBox c = new JCheckBox();
    c.setSelected(false);
    checkboxList.add(c);
    mainPanel.add(c);
} // end loop

theFrame.setBounds(50,50,300,300);
theFrame.pack();
theFrame.setVisible(true);
} // close buildGUI

public void setUpMidi() {
    try {
        sequencer = MidiSystem.getSequencer();
        sequencer.open();
        sequence = new Sequence(Sequence.PPQ, 4);
        track = sequence.createTrack(); // 创建Track
        sequencer.setTempoInBPM(120);
    } catch(Exception e) {e.printStackTrace();}
} // close setUpMidi

```

```

public void buildTrackAndStart() {
    ArrayList<Integer> trackList = null; // this will hold the instruments for each
    sequence.deleteTrack(track);
    track = sequence.createTrack();
    for (int i = 0; i < 16; i++) {
        trackList = new ArrayList<Integer>();
        for (int j = 0; j < 16; j++) {
            JCheckBox jc = (JCheckBox) checkboxList.get(j + (16*i));
            if (jc.isSelected()) {
                int key = instruments[i];
                trackList.add(new Integer(key));
            } else {
                trackList.add(null); // because this slot should be empty in the track
            }
        } // close inner loop
        makeTracks(trackList);
    } // close outer loop
    track.add(makeEvent(192, 9, 1, 0, 15)); // - so we always go to full 16 beats
    try {
        sequencer.setSequence(sequence);
        sequencer.setLoopCount(sequencer.LOOP_CONTINUOUSLY);
        sequencer.start();
        sequencer.setTempoInBPM(120);
    } catch (Exception e) {e.printStackTrace();}
    } // close method

    public class MyStartListener implements ActionListener {
        public void actionPerformed(ActionEvent a) {
            buildTrackAndStart();
        } // close actionPerformed
    } // close inner class

    public class MyStopListener implements ActionListener {
        public void actionPerformed(ActionEvent a) {
            sequencer.stop();
        } // close actionPerformed
    } // close inner class

    public class MyUpTempoListener implements ActionListener {
        public void actionPerformed(ActionEvent a) {
            float tempoFactor = sequencer.getTempoFactor();
            sequencer.setTempoFactor((float)(tempoFactor * 1.03));
        } // close actionPerformed
    } // close inner class
}

```

跟以前的章节内容一样

GUI的监听者

```

public class MyDownTempoListener implements ActionListener {
    public void actionPerformed(ActionEvent a) {
        float tempoFactor = sequencer.getTempoFactor();
        sequencer.setTempoFactor((float)(tempoFactor * .97));
    }
}

public class MySendListener implements ActionListener {
    public void actionPerformed(ActionEvent a) {
        // make an arraylist of just the STATE of the checkboxes
        boolean[] checkboxState = new boolean[256];
        for (int i = 0; i < 256; i++) {
            JCheckBox check = (JCheckBox) checkboxList.get(i);
            if (check.isSelected()) {
                checkboxState[i] = true;
            }
        } // close loop
        String messageToSend = null;
        try {
            out.writeObject(userName + nextNum++ + ":" + userMessage.getText());
            out.writeObject(checkboxState);
        } catch (Exception ex) {
            System.out.println("Sorry dude. Could not send it to the server.");
        }
        userMessage.setText("");
    } // close actionPerformed
} // close inner class

public class MyListSelectionListener implements ListSelectionListener {
    public void valueChanged(ListSelectionEvent le) {
        if (!le.getValueIsAdjusting()) {
            String selected = (String) incomingList.getSelectedValue();
            if (selected != null) {
                // now go to the map, and change the sequence
                boolean[] selectedState = (boolean[]) otherSeqsMap.get(selected);
                changeSequence(selectedState);
                sequencer.stop();
                buildTrackAndStart();
            }
        }
    } // close valueChanged
} // close inner class

```

文本信息和节拍样式会被序列化送到输出串流

用户点选信息时会马上加载节拍样式并开始播放

```

public class RemoteReader implements Runnable {
    boolean[] checkboxState = null;
    String nameToShow = null;
    Object obj = null;
    public void run() {
        try {
            while((obj=in.readObject()) != null) {
                System.out.println("got an object from server");
                System.out.println(obj.getClass());
                String nameToShow = (String) obj;
                checkboxState = (boolean[]) in.readObject();
                otherSeqsMap.put(nameToShow, checkboxState);
                listVector.add(nameToShow);
                incomingList.setListData(listVector);
            } // close while
        } catch(Exception ex) {ex.printStackTrace();}
    } // close run
} // close inner class

public class MyPlayMineListener implements ActionListener {
    public void actionPerformed(ActionEvent a) {
        if (mySequence != null) {
            sequence = mySequence; // restore to my original
        }
    } // close actionPerformed
} // close inner class

public void changeSequence(boolean[] checkboxState) {
    for (int i = 0; i < 256; i++) {
        JCheckBox check = (JCheckBox) checkboxList.get(i);
        if (checkboxState[i]) {
            check.setSelected(true);
        } else {
            check.setSelected(false);
        }
    } // close loop
} // close changeSequence

public void makeTracks(ArrayList list) {
    Iterator it = list.iterator();
    for (int i = 0; i < 16; i++) {
        Integer num = (Integer) it.next();
        if (num != null) {
            int numKey = num.intValue();
            track.add(makeEvent(144, 9, numKey, 100, i));
            track.add(makeEvent(128, 9, numKey, 100, i + 1));
        }
    } // close loop
} // close makeTracks()

```

线程执行的任务  
收到的信息会加入list组件  
MIDI部分与以前的内容完全一样

```
public MidiEvent makeEvent(int comd, int chan, int one, int two, int tick) {  
    MidiEvent event = null;  
    try {  
        ShortMessage a = new ShortMessage();  
        a.setMessage(comd, chan, one, two);  
        event = new MidiEvent(a, tick);  
    } catch (Exception e) {}  
    return event; 也跟以前一样  
} // close makeEvent  
  
} // close class
```



有哪些方法可以改善这个程序？

你可以从以下的方向考虑：

- (1) 加载新的样式时，目前的样式就会消失。如果正在设计新样式，则辛苦都白费了。你也许会希望有个提示对话框确认是否要先存盘。
- (2) 输入不正确的命令栏参数会导致异常！把程序改成在这种情况下能够使用默认值或给使用者一个提示，尽可能地让程序优雅的运行而不会粗鲁得直接挂掉。
- (3) 随机产生节奏样式（译注：这是个烂主意，相信我，计算机只能随机产生噪音）。

## BeatBox服务器端最终版

大部分的程序与讨论网络以及线程的章节内容一样。唯一的差别是服务器接收并传送两个对象而不是一个对象。

```

import java.io.*;
import java.net.*;
import java.util.*;

public class MusicServer {

    ArrayList<ObjectOutputStream> clientOutputStreams;

    public static void main (String[] args) {
        new MusicServer().go();
    }

    public class ClientHandler implements Runnable {

        ObjectInputStream in;
        Socket clientSocket;

        public ClientHandler(Socket socket) {
            try {
                clientSocket = socket;
                in = new ObjectInputStream(clientSocket.getInputStream());
            } catch (Exception ex) {ex.printStackTrace();}
        } // close constructor

        public void run() {
            Object o2 = null;
            Object o1 = null;
            try {
                while ((o1 = in.readObject()) != null) {
                    o2 = in.readObject();

                    System.out.println("read two objects");
                    tellEveryone(o1, o2);
                } // close while
            } catch (Exception ex) {ex.printStackTrace();}
        } // close run
    } // close inner class
}

```

```
public void go() {
    clientOutputStreams = new ArrayList<ObjectOutputStream>();

    try {
        ServerSocket serverSock = new ServerSocket(4242);

        while(true) {
            Socket clientSocket = serverSock.accept();
            ObjectOutputStream out = new ObjectOutputStream(clientSocket.getOutputStream());
            clientOutputStreams.add(out);

            Thread t = new Thread(new ClientHandler(clientSocket));
            t.start();

            System.out.println("got a connection");
        }
    } catch(Exception ex) {
        ex.printStackTrace();
    }
} // close go

public void tellEveryone(Object one, Object two) {
    Iterator it = clientOutputStreams.iterator();
    while(it.hasNext()) {
        try {
            ObjectOutputStream out = (ObjectOutputStream) it.next();
            out.writeObject(one);
            out.writeObject(two);
        } catch(Exception ex) {ex.printStackTrace();}
    }
} // close tellEveryone

} // close class
```



## 附录B



我们已经说了很多，你也差不多快看完了。我们会很想念你的，但在让你上场前如果漏掉一些东西没有告诉你，我们会很不放心。可是附录里面放不了这么多东西，实际上我们曾经尝试要把所有东西挤到这本书里面。最后结果是字体要缩小到0.00003英寸才办得到，这根本不能读，所以我们决定放弃这个念头。最后我们留下了这10项最重要的主题。

## #10 位操作

### 你在乎吗？

我们之前讨论过byte有8位，short有16位。也许你会需要个别的操作某个位的值，也许有一天你会遇到Java烤面包机并发现它只有1KB的内存，为了节省空间只好用个别字节来存储设定。

### 按位非运算符：~

这个运算符会将primitive主数据类型的字节组合值取反。

```
int x = 10;           // 00001010
x = ~x;              // 变成 11110101
```

接下来的3个运算符会逐位地比较primitive主数据类型的值，然后返回比较结果，下面是要比较的范例。

```
int x = 10;           // 位组合是00001010
int y = 6;            // 位组合是00000110
```

### 按位与运算符：&

如果两个位都是1才会返回1，否则返回0。

```
int a = x & y; // 位组合是00000010
```

### 按位或运算符：|

只要其中有一个是1就会返回1。

```
int a = x | y; // 位组合是00001110
```

### 按位异或运算符：^

位相同时返回0，否则返回1。

```
int a = x ^ y; // 位组合是00001100
```

### 移位运算符

此运算符取用单一primitive主数据类型并向某个方向执行移位，如果你的二进制运算技巧够好的话，你就会发现左移的效果与乘以2是一样的，而右移跟除以2一样。

接下来的3个运算符使用下面这个值。

```
int x = -11; // 位组合是11110101
```

以下将全世界最短的负数表示成补码。整数最左方的字节称为符号位。Java中的负整数此位永远是1。正数此位永远是0。Java使用补码来存储负值，改变正负号时要把位取反然后加1。

### 右移运算符：>>

以指定量右移字节合，左方补0，正负号不变。

```
int y = x >> 2; // 位组合是11111101
```

### 无符号右移运算符：>>>

与>>一样，但第一个位也会补0，正负号可能会改变。

```
int y = x >>> 2; // 位组合是00111101
```

### 左移运算符：<<

与>>>一样，但方向相反：右方补0，正负号可能改变。

```
int y = x << 2; // 位组合是11010100
```

## #9 不变性

### 为什么要在乎 String 的不变性？

当程序越来越大时，不可避免地会有很多 String 对象。为了安全性和节省空间（例如在手机上执行）的原因，String 是不变的。意思是下面的程序：

```
String s = "0";
for (int x = 1; x < 10; x++) {
    s = s + x;
}
```

实际上会创建出10个String对象（“0”，“01”，“012”……“0123456789”）。最后会引用到“0123456789”这个值，但此时会存在10个String。

创建新的String时，Java虚拟机会把它放到称为“String Pool”的特殊存储区中。如果已经出现同值的String，Java虚拟机不会重复建立String，只会引用已存在者。这是因为String是不变的，引用变量无法改变其它参考变量引用到的同一个String值。

String pool不受Garbage Collector管理，因此我们在for循环中建立的10个String有9个是在浪费空间。

### 为什么要在乎包装类的不变性？

之前我们有讨论到包装类的两个主要用途：

- (1) 将 primitive 主数据类型包装成对象。
- (2) 使用静态的工具方法（例如说 Integer.parseInt()）。

当你创建像下面这个包装对象时：

```
Integer iWrap = new Integer(42);
```

它的值永远会是 42。包装对象没有 setter。你当然还是能够让 iWrap 引用别的包装对象，但是会产生两个对象。包装对象创建后就无法改变该对象的值！

### 这要如何节省内存？

如果你不注意的话就不会！但如果你知道String的不变性，就可以利用它来节省空间。如果要执行一堆String操作，则StringBuilder这个稍后会介绍的class更为合适。



## #8 断言

我们并没有对开发过程的除错进行很多的讨论。我们认为你应该从命令行来学习Java，就像这本书的设计方式一样。一旦你成为Java大师之后，或许你会使用IDE\*的debug工具。以前的Java程序设计师得在程序中加入一大堆System.out.println()命令来显示除错的信息，列出变量值，还有执行到那里的信息以观察流程控制的走向。等到程序可以正确执行的时候，还要看过所有的程序把println()拿掉。这很麻烦且容易出错。从Java 1.4之后除错就变得容易多了。为什么？

### 断言

它就像是喝了3瓶蛮牛加上两粒克补的println()一样。使用的方式跟加入println()差不多，Java 5.0的编译器会假设所编译的源文件与5.0兼容，此时预设断言是打开的。

执行时，如果没有特别设定的话，被加入到程序中的assert命令会被Java虚拟机忽略。但若你指定Java虚拟机要打开断言的话，它就能够在不变动任何一行程序的情况下帮助你对程序除错。

有些人会抱怨最终版本的程序上还有assert命令，然而留着assert对于已经部署安装的程序代码是很有价值的。如果用户遇到问题，你就可以指示客户打开断言来执行程序，并取得输出结果。如果没有留下assert，你就很难知道发生了什么事。这样做没什么坏处；未打开时，Java虚拟机会忽略掉它们，所以不会影响性能。

### 如何使用断言

在你认为一定是true的地方加上assert命令，例如：

```
assert (height > 0);
// 若true，则继续执行
// 若false，抛出AssertionError
```

你也可以加上一点信息：

```
assert (height > 0) : "height = " + height +
" weight = " + weight;
```

在冒号后面的指令可以是任何解出非null值的合法Java语句。无论如何千万别在assert中改变对象的状态！不然的话，打开assertion执行时可能会改变程序的行为。

### 带有断言的编译和执行

编译：

```
javac TestDriveGame.java
```

（注意并不需要特殊选项）

执行：

```
java -ea TestDriveGame
```

\*IDE代表Integegered Development Environment（集成开发环境），例如Eclipse、Borland的JBuilder或开放源码的NetBeans。

## #7 块区域

我们在第9章讨论局部变量的生命周期范围只有在声明它的方法还待在栈上的期间内。但某些变量的生命周期会更短。我们经常会在方法中建立程序区段，但是还没有讨论过块的议题。通常区段程序代码会在方法中，并以{}字符符号来区分。常见的例子有循环（for与while）以及条件测试运算（如if）。

以下面的程序为例：

```

void doStuff() { ← 方法区段的起点
    int x = 0; ← 生命周期跟整个方法一样的局部变量
    for(int y = 0; y < 5; y++) { ← for循环的block, y的范围只限于此
        x = x + y; ← 使用x和y不是问题
    } ← 区段终点
    x = x * y; ← 啊！不能编译，y已经超出范围了（注意：某些其他
} ← language容许此状况)
    method区段的终点，x也不能超出此范围

```

在上面的范例中，y是个块方法，被宣告在块中，一离开循环就超出范围。使用局部变量比实体变量更方便除错且扩充性更好，而能用块变量时就把局部变量换掉。编译器会确保你不会用到超出范围的变量，因此不用担心执行时会出错。

## #6 链接的调用

本书中很少这么使用，因为我们尝试让语法尽可能的干净和方便阅读。然而在Java中有很多合法的快捷方式，毫无疑问，你会看到很多如此编写的程序代码。你最有可能遇到的结构之一就是链接调用。像下面这样：

```
StringBuffer sb = new StringBuffer("spring");
sb = sb.delete(3, 6).insert(2, "umme").deleteCharAt(1);
System.out.println("sb = " + sb);
// 结果是 sb = summer
```

第二行程序是怎么回事？好吧，这是很刻意弄出来的例子，但你需要知道如何解释它们。

- (1) 从左开始看到右边。
- (2) 找出最左方的方法，此例为sb.delete(3,6)。如果查询StringBuffer的文件，你会看到delete()返回StringBuffer对象。结果就是delete()返回一个值为“spr”的StringBuffer对象。
- (3) 接着是对新创建出来的StringBuffer对象调用insert()，此调用也会返回StringBuffer对象，然后如此这般的下去直到最右边。理论上，你在同一行程序可以无限地链接下去（实际上很少会超过3个）。如果不这么做，上面的第二行程序可以改写成下面这种更易于阅读的形式：

```
sb = sb.delete(3, 6);
sb = sb.insert(2, "umme");
sb = sb.deleteCharAt(1);
```

下面有个更常见、更有用的例子，你之前就已经看到我们用过，现在又被带出来。这是个调用方法又不需维持一个引用的方法。

```
class Foo {
    public static void main(String [] args) [
        new Foo().go();
    }
    void go() {
        // 真正要执行的程序
    }
}
```

← 我们需要调用go()，但又不需要维持一个对Foo的引用

## #5 Anonymous和Static Nested Classes

### 嵌套的类有很多种

以前提到GUI事件处理时，我们有使用内部（套迭）的类来解决接听的实现。这是内部类很常见、实用，且易于阅读的形式，类是包在另一个类的括号中。要记住这代表你需要外部类的实例才能取得内部类的实例，因为内部是外部的一个成员。

但还有包括了static与anonymous等类型的内部类。我们不会详谈细节，但还是要能让你看别人的程序时懂这种语法。因为Java几乎能做任何事情，所以也许没有其它东西能够产生出更可怕的不具名内部类。我们先从比较简单的静态嵌套类开始。

### 静态嵌套类

你已经知道static是什么意思——跟在类而不是特定实例的东西。静态嵌套的类看起来跟我们用在事件监听者的非静态类很像，但被标示为static的。

```
public class FooOuter {  
    static class BarInner {  
        void sayIt() {  
            System.out.println("method of a static inner class");  
        }  
    }  
}  
  
class Test {  
    public static void main(String[] args) {  
        FooOuter.BarInner foo = new FooOuter.BarInner();  
        foo.sayIt();  
    }  
}
```

静态嵌套类就是被包在另外一个类里面标记为static的类

因为是静态的，所以不需外层的实例

静态嵌套很像一般非嵌套的，他们并未与外层对象产生特殊关联。但因为还被认为是外层的一个成员，所以能够存取任何外层的私用成员。然而只限于也是静态的，这是因为静态并没有实例。

## #5 Anonymous和Static Nested Classes (续)

### nested与inner的差别

任何被定义在另一个类范围内的类被称为嵌套的类，不管是否为匿名、静态、正常或其他类型。只要是在另一个类中，技术上它就被认定是嵌套的类。但非静态的嵌套的类通常被称做内部的类，我们之前也是这么写的。基本上所有内部类都是嵌套的，但不是所有嵌套类都是内部的。

### 匿名的内部类

假设你在编写GUI程序，突然发现你需要一个有实现ActionListner的类的实例，你也发现你没有任何该实例，其实你根本没有写过这样的类（译注：听起来很白痴，实际上很常见），此时你有两种选择：

(1) 在程序中编写内部类，就像我们之前的GUI程序一样。

或者

(2) 当场创建出匿名的内部类。就在现场开始写一个类出来。没错，就在需要传入一个实例的地方创建出这个类。这代表你把整个类当作参数传进去。

```
import java.awt.event.*;
import javax.swing.*;
public class TestAnon {
    public static void main (String[] args) {
        JFrame frame = new JFrame();
        JButton button = new JButton("click");
        frame.getContentPane().add(button);
        // button.addActionListener(quitListener);
    }
}
```

这个语句  
一直到这才算完整

注意此处的ActionListner其实是个接口，而我们不能创建接口的实例！但这种情况的语法就是要这样

创建框架并加入按钮，此时需要注册按钮的  
监听者  
通常是传入一个内部类的实例  
但是我们也可以把整个类的定义传进去，此语法也会自动地创建出该类的实例

## #4 存取权限和存取修饰符 (谁可以看到什么)

Java有4种存取权限等级与3种存取修饰符。只有3种修饰符的原因是因为还有一个缺省的（不加任何的修饰字）的权限等级。

存取权限（从限制最少的开始列出）

**public** ← 代表任何程序代码都可以存取的公开事物（类、变量、方法、构造函数等）

**protected** ← 受保护的部分运行起来像是**default**，但也能允许不在相同包的子类继承受保护的部分

**default** ← 只有在同一包中的默认事物能够存取

**private** ← 只有同一类中的程序代码才能存取，它是对类而不是对象设限，所以**Dog**可以看到别的**Dog**的私用部分，但**Cat**就不能看到**Dog**的私用部分

存取修饰符

**public**

**protected**

**private**

通常你只会用到 **public** 和 **private** 两种等级。

### **public**

用**public**来设定打算要开放给其他程序代码的类、常数（**static final** 变量）、方法，以及大部分的构造函数。

### **private**

设定给大部分的实例变量，以及不想开放给外部程序调用的方法。

虽然你可能不会用到其余两个，你还是必须知道它们的作用。

## #4 存取权限和存取修饰符（续）

### default与protected

#### **default**

protected与default权限等级都是作用在包上。默认的等级是最简单的，它代表只有同一包也是默认等级的程序可以存取。例如没有被明确声明为public的类只能被同一包中的类存取。

存取又是什么意思呢？不能存取某类的程序代码是不能想到该类的。这个想字代表使用。例如说你不能初始化或声明此受限制类型的变量、参数、返回值。反正就是不能在程序中提到这个类！不然编译器会跟你解释概念。

想象一下这样隐含的意义——在默认的类中的public方法意味着此方法并不全然是public的。如果你无法存取该类就无法取用此方法。

为什么要限制同一包的存取？通常包是一群在运作上有关联的类。所以互相存取程序代码是很合理的，而只有少部分的类与方法会开放给外部。

这就是default。它是如此的单纯——只有同一包可以存取。

#### **protected**

它与默认的等级几乎一样，只有一处不同：允许不同包中的子类继承它的成员。这就是protected所带来的功能——不在同一个包的子类。

大部分的开发者很少有机会动用到protected，也许有一天你会遇到适用的设计和时机。protected 只能应用在继承上。如果不同包的子类有对父类实例的引用，子类无法透过此引用存取父类的protected方法！唯一的办法只能通过继承取得此方法。

### #3 String和StringBuffer/StringBuilder的方法

Java API中最常用到的类包括了String和StringBuffer（因为String是不变的，使用StringBuffer/StringBuilder来操作String比较有效率）。从Java 5.0起，你应该以StringBuilder取代StringBuffer，除非这些操作必须在不常见的thread安全环境中进行。下面列出一些比较重要的方法：

char charAt(int index);	// 字符出现的位置
int length();	// 字符串有多长
String substring(int start, int end);	// 抽出一部分
String toString();	// 此物的String表示值

连接字符串：

String concat(string);	// String使用
String append(String);	// StringBuffer和StringBuilder使用

String的方法：

String replace(char old, char new);	// 替换
String substring(int begin, int end);	// 抽出一部分
char [] toCharArray();	// 转换成char数组
String toLowerCase();	// 转成小写
String toUpperCase();	// 转成大写
String trim();	// 删除后端空格符
String valueOf(char [])	// 转换成String
String valueOf(int i)	// 转换成String
	// 也有其他primitive主数据类型版本

StringBuffer和StringBuilder的方法：

StringBxxxx delete(int start, int end);	// 删除部分
StringBxxxx insert(int offset, any primitive or a char []);	// 插入
StringBxxxx replace(int start, int end, String s);	// 取代
StringBxxxx reverse();	// 翻转
void setCharAt(int index, char ch);	// 替换字符

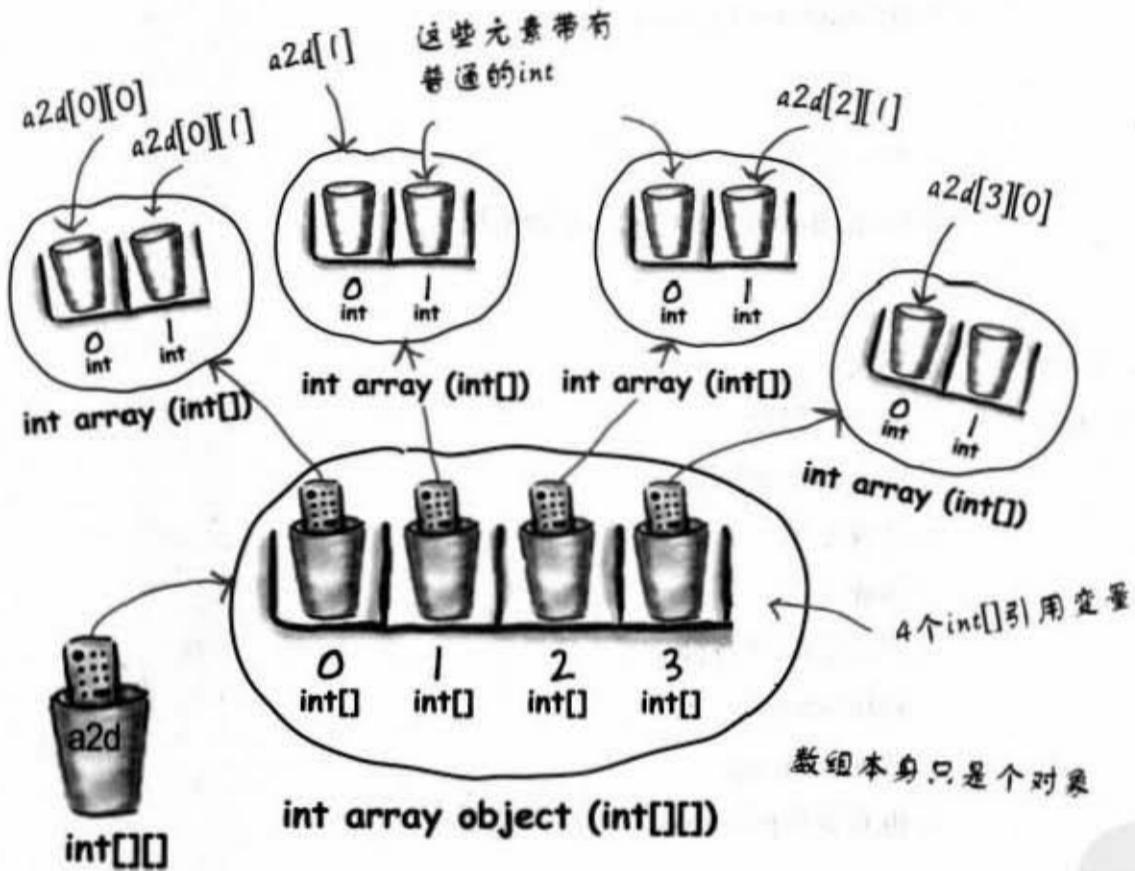
注意：Bxxxx 不是在骂人

## #2 多维数组

在大部分的语言中，如果你创建了 $4 \times 2$ 的二维数组，看起来会像是个有8个元素的 $4 \times 2$ 方阵。但在Java中，此数组实际上是由5个数组连接而成！在Java中，二维数组只是个数组的数组（三维数组是数组的数组的数组）：

```
int[][] a2d = new int[4][2];
```

上面程序会让Java虚拟机创建出有4个元素的数组，这些元素实际上是对2个元素数组的引用变量。



操作多维数组：

- (1) 存取第三个数组的第二个元素： `int x = a2d[2][1];`
- (2) 对某个子数组创建引用： `int[] copy = a2d[1];`
- (3) 初始化 $2 \times 3$ 数组： `int[][] x = {{2,3,4}, {7,8,9}};`
- (4) 创建非常规二维数组：

```
int[][] y = new int[2][]; // 创建长度为2的第一层
y[0] = new int[3]; // 创建三元素的子数组
y[1] = new int[5]; // 创建五元素的子数组
```

## 我们还没有讨论到的第一大主题是……

### #1 枚举 (又称为枚举类型或Enum)

我们以前讨论过定义在API中的常量，例如JFrame.EXIT\_ON\_CLOSE。你也可以用标记为static final的变量来设定常量。有时你会需要一组常量来代表可用值的集合。这样的一组可用值被称为枚举。在Java 5.0中你可以创建让旧版程序员羡慕的要死的全功能货真价实枚举。

团员有谁？

假设你在帮你最喜欢的乐团架构网站，且想要确保所有的讨论都会指向特定的团员。

古老的方法：

```
public static final int JERRY = 1;
public static final int BOBBY = 2;
public static final int PHIL = 3;

// 稍后
if (selectedBandMember == JERRY) {
    // 执行与JERRY有关的工作
}
```

我们祈祷到时候这个值是合理有效的！

好消息是这样的，程序有很高的可读性。另外一项好消息是你无法变更已经创建出来的常数值：JERRY永远会是1。坏消息是你没有办法能够简单地确保selectedBandMember的值一定是1, 2, 3。如果有个程序不小心把selectedBandMember设成9527，你的程序很可能就因此无法运行了……

## #1 枚举还没完……

同样的情况套在Java 5.0的enum上面就很简单了，因为这是很基本的枚举情况，所以会很简单。

全新正式的enum是这样的：

```
public enum Members { JERRY, BOBBY, PHIL };
public Members selectedBandMember;
// 稍后
if (selectedBandMember == Members.JERRY) {
    // 执行与JERRY有关的工作
}
无需担心此变量的值
```

这很像类的定义不是吗？enum就是一种特殊的类

Members类型的selectedBandMember只能有JERRY, BOBBY, PHIL这3种值

引用到enum的实例

### enum会继承java.lang.Enum

在创建enum时，其实是隐含地继承java.lang.Enum来创建新的类。你可以在独立的源文件中把enum宣告成独立的类，或是作为其他类的一员。

### 使用if或switch

我们可以在if或switch中使用enum。注意到我们能够以==或.equals()来比较enum的实例。一般认为==是比较好的风格。

```
Members n = Members.BOBBY; ← 赋enum值
if (n.equals(Members.JERRY)) System.out.println("Jerrrry!");
if (n == Members.BOBBY) System.out.println("Rat Dog"); ← 这两种方式都行

Members ifName = Members.PHIL;
switch (ifName) {
    case JERRY: System.out.print("make it sing ");
    case PHIL: System.out.print("go deep "); ← 猜猜看会有什么输出
    case BOBBY: System.out.println("Cassidy! ");
}
```

答案： go deep Cassidy!

## #1 枚举，说完了

## 相同enum的特殊版本

你可以在enum中加入构造函数、方法、变量和特定常量的内容（class body）。不是很常见，但有时你还是会遇到：

```
public class HfjEnum {
    enum Names {
        JERRY("lead guitar") { public String sings() { return "plaintively"; } },
        BOBBY("rhythm guitar") { public String sings() { return "hoarsely"; } },
        PHIL("bass");
    }
    private String instrument;
    Names(String instrument) {
        this.instrument = instrument;
    }
    public String getInstrument() {
        return this.instrument;
    }
    public String sings() {
        return "occasionally";
    }
}
public static void main(String [] args) {
    for (Names n : Names.values()) {
        System.out.print(n);
        System.out.print(", instrument: " + n.getInstrument());
        System.out.println(", sings: " + n.sings());
    }
}
```

```
File Edit Window Help Bootleg
%java HfjEnum
JERRY, instrument: lead guitar, sings: plaintively
BOBBY, instrument: rhythm guitar, sings: hoarsely
PHIL, instrument: bass, sings: occasionally
%
```

传给下方定义构造函数的参数

特定常量的内容。把它们当作是  
基本enum方法的覆盖（此例中是  
sings()）

这是enum的构造函数，会对每个被声  
明的enum值执行一次（此例中运行  
3次）

全被main()调用

每个enum都有内置的  
values()，通常会用在  
for循环

基本的sings()只会在enum  
值没有特定的内容时才会  
被调用



## 迷你推理 短篇

### 归乡路



星际战舰“Traverser”号的魏元祖船长收到一份总部传来的紧急最高机密信息。信息带有30个高度加密编码过的导航码，只有这些导航码指出的路径能带他们通过危机四伏的敌方控制区。

来自邻近星系的骇克人有着高度进化的科技，能够发送捣蛋射线潜入“Traverser”号上面唯一的导航计算机，在计算机的堆空间埋进伪造的对象。此外，他们还有种技术能够偷偷地将引用变量改指向伪造对象。“Traverser”号的船员手上只有一种工具能够抵抗骇克人对导航系统的Java 5.0程序代码做邪恶的攻击，就是程序代码内嵌的病毒检测器。

魏船长以下面列出的命令要求史正浩处理关键程序代码：

“将前五个导航码放进ParsecKey类型的数组中。其余25个放进 $5 \times 5$ 的QuadrantKey类型二维数组中。将这两个数组传入public final的ShipNavigation这个class的plotCourse()中。一旦取回路线对象就对所有的引用变量执行病毒检查以及NavSim仿真程序，然后向我报告结果”。

几分钟后史正浩拿到了模拟输出。“模拟结果出来了”。“很好”，船长回应，“请说”。“是，长官”，史正浩继续说明，“我先以ParsecKey[] p = new ParsecKey[5]; 声明并创建ParsecKey类型的数组，接着是QuadrantKey[][] q = new QuadrantKey[5][5]; 再来使用for循环加载5个导航码给ParsecKey数组，同样使用for循环加载25个导航码给QuadrantKey数组。此后我用病毒检测器对32个引用变量、ParsecKey数组和它的5个元素，QuadrantKey数组和它的25个元素做检查。等待检测器报告没有发现病毒时，我就运行了仿真程序并重新执行病毒检测”。

魏元祖船长瞪着史正浩看着，冷冷地说：“小史，因为你危害这艘船，我要关你禁闭，我不想再看到你进到指挥舱，除非你把Java学好！布尔中尉，你接手这项任务”。

为什么船长要关他禁闭呢？



## 迷你推理短篇解答



### 归乡路

魏船长知道在Java中，多维数组实际上是数组的数组。 $5 \times 5$ 的QuadrantKey数组实际上总共有31个引用变量能够存取所有的元素：

1个引用变量： q

5个引用变量： q[0] 到 q[4]

25个引用变量： q[0][0] 到 q[4][4]

共计31个。

史正浩忘记算到埋在q数组的5个一维数组。如果骇客人通过这5个引用变量来潜入，则病毒检测不会捕获到这个危机。

java学习群：72030155，每天20:30-23:00都有大神视频教学，想学习的同学可以进群免费听课！

# 索引

## 符号和标识

&, &&, !, || (布尔运算符) 151, 660  
 &, <<, >>, >>>, ^, !, ~ (位运算符) 660  
 ++ -- (递增/递减) 105, 115  
 + (String连接运算符) 17  
 . (圆点运算符) 36  
     引用 54  
 <, <=, ==, !=, >, >= (比较运算符) 86, 114, 151  
 <, <=, ==, >, >= (比较运算符) 11

## A

abandoned objects (见垃圾收集器)  
 abstract (抽象)  
     class (类) 200–210  
     class modifier (类修饰符) 200  
 abstract methods (抽象方法)  
     declaring (声明) 203  
 access (存取)  
     and inheritance (继承) 180  
     class modifiers (类修饰符) 667  
     method modifiers (方法修饰符) 81, 667  
     variable modifiers (变量修饰符) 81, 667  
 accessors and mutators (见getters和setters)  
 ActionListener interface (ActionListener接口) 358, 358–361  
 addActionListener() 359–361  
 advice guy (叶教授) 480, 484  
 Aeron™ 28  
 animation (动画效果) 382–385  
 API (应用编程接口) 154–155, 158–160  
     ArrayList 532  
 collections (集合) 558

appendix A (附录A) 649–658  
     beat box final client (beat box客户端最终版) 650  
     beat box final server (beat box服务器端最终版) 657  
 appendix B (附录B)  
     access levels and modifiers (存取权限和存取修饰符) 667  
     assertions (断言) 662  
     bit manipulation (位操作) 660  
     block scope (区块域) 663  
     immutability (不变性) 661  
     linked invocations (链接的调用) 664  
     multidimensional arrays (多维数组) 670  
     String and StringBuffer methods (String和StringBuffer方法) 669  
     apples and oranges (苹果和桔子) 137  
     arguments (参数)  
         method (方法) 74, 76, 78  
         polymorphic (多态) 187  
     ArrayList 132, 133–138, 156, 208, 558  
         API (应用编程接口) 532  
         ArrayList<Object> 211–213  
         autoboxing (autoboxing) 288–289  
         casting (转换) 229  
     arrays (数组)  
         about (介绍) 17, 59, 135  
         assigning (赋值) 59  
         compared to ArrayList (比较ArrayList和一般数组) 134–137  
         creation (创建) 60  
         declaring (声明) 59  
         length attribute (长度) 17  
         multidimensional (多维) 670  
         objects, of (对象) 60, 83  
         primitives, of (primitives主数据类型) 59  
     assertions (断言)  
         assertions (断言) 662  
     assignments, primitive (赋值, primitive主数据类型) 52

## A - C

assignments, reference variables (赋值, 引用变量) 55, 57, 83

atomic code blocks (原子块) 510–512 (见线程)

audio. (见 midi)

autoboxing (autoboxing) 288–291

and operators (运算符) 291

assignments (赋值) 291

## B

bark different (不同的吠声) 73

bathtub (澡盆) 177

beat box (beat box) 316, 347, 472 (见附录A)

beer (啤酒) 14

behavior (行为) 73

Bela Fleck (Bela Fleck) 30

bitwise operators (位运算符) 660

bit shifting (移位) 660

block scope (块区域) 663

boolean (布尔) 51

boolean expressions (布尔表达式) 11, 114

logical (逻辑) 151

BorderLayout manager (BorderLayout管理器) 370–371, 401, 407

BoxLayout manager (BoxLayout管理器) 411

brain barbell (brain barbell) 33, 167, 188

break statement (break语句) 105

BufferedReader 454, 478

BufferedWriter 453

buffers (缓冲区) 453, 454

byte (字节) 51

bytecode (字节码) 2

## C

Calendar 303–305

methods (方法) 305

678 索引

casting (转换)

explicit primitive (显primitive主数据类型) 117

explicit reference (显引用) 216

implicit primitive (隐primitive主数据类型) 117

catching exceptions (捕获异常) 326

catch (catch) 338

catching multiple exceptions (捕获多重异常) 329, 330, 332

try (try) 321

catch blocks (catch块) 326, 338

catching multiple exceptions (捕获多重异常) 329, 330, 332

chair wars (椅子大战) 28, 166

char (char) 51

chat client (聊天室客户端程序) 486

with threads (线程) 518

chat server (simple) (聊天服务器程序) 520

checked exceptions (检查异常)

runtime vs. (运行期间) 324

checking account (见Ryan和Monica)

check box (JCheckBox) (复选框) 416

class (类)

abstract (抽象) 200–210

concrete (创建) 200–210

designing (设计) 34, 41, 79

final (final) 283

fully qualified names (完整名称) 154–155, 157

client/server (客户端/服务器端) 473

code kitchen (程序料理)

beat box save and restore (beat box保存和恢复) 462

final beat box. (beat box最终版, 见附录A)

making the GUI (创建GUI) 418

music with graphics (带有图形的音乐) 386

playing sound (播放声音) 339

coffee cups (咖啡杯) 51

collections (集合) 137, 533

API (应用程序接口) 558

ArrayList (ArrayList) 137

ArrayList<Object> (ArrayList<Object>) 211–213

Collections.sort() 534, 539

**H**  
 HashMap (HashMap) 533  
 HashSet (HashSet) 533  
 LinkedHashMap (LinkedHashMap) 533  
 LinkedList (LinkedList) 533  
 List (List) 557  
 Map (Map) 557, 567  
 parameterized types (参数化类型) 137  
 Set (Set) 557  
 TreeSet (TreeSet) 533  
 Collections.sort() 534, 539  
 Comparator 551  
 compare() 553  
**C**  
 Comparable 547, 566  
 and TreeSet 566  
 compareTo() method (compareTo()方法) 549  
 Comparator 551, 566  
 and TreeSet 566  
 compare() 553  
 compareTo() 549  
 comparing with == (用==号来比较) 86  
 compiler (编译器) 2  
 about (介绍) 18  
 java -d (java -d) 590  
 concatenate (连接) 17  
 concrete classes (创建类) 200–210  
 conditional expressions (条件表达式) 10, 11, 13  
 constants (常量) 282  
 constructors (构造函数)  
 about (介绍) 240  
 chaining (链接) 250–256  
 overloaded (重载) 256  
 superclass (父类) 250–256  
 contracts (合约) 190–191, 218  
 cups (杯子) 51  
 curly braces (curly braces) 10

**D**

DailyAdviceClient (DailyAdviceClient程序代码) 480

**D**  
 DailyAdviceServer (DailyAdviceServer程序代码) 484  
 dancing girl (跳舞女孩) 316  
 dates (日期)  
 Calendar (Calendar) 303  
 methods (方法) 305  
 formatting (格式) 301  
 GregorianCalendar (GregorianCalendar) 303  
 java.util.Date 303  
 deadlock (死锁) 516  
 deadly diamond of death (致命方块) 223  
 declarations (声明)  
 about (介绍) 50  
 exceptions (异常) 335–336  
 instance variables (实例变量) 50  
 default access (默认存取) 668  
 default value (默认值) 84  
 deployment options (部署的选择) 582, 608  
 deserialized objects 441 (见序列化)  
 directory structures (目录结构)  
 packages (包) 589  
 servlets (servlets) 626  
 doctor (医生) 169  
 dot operator (圆点运算符)  
 reference (引用) 54  
 double (double) 51  
 duck (鸭子) 277  
 construct (构造) 242  
 garbage collect (垃圾收集器) 261  
 ducking exceptions (ducking异常) 335

**E**

EJB 631

encapsulation (封装)  
 about (介绍) 79–82  
 benefits (优点) 80

end of book (结束) 648

E - F

enumerations (枚举) 671–672  
enums (枚举) 671–672  
equality (相等) 560  
    and hashCode() (hashCode()方法) 561  
equals() (equals()方法) 561  
equals() (equals()方法)  
    about (介绍) 209  
    Object class (Object类) 209  
event handling (事件处理) 357–361  
    event object (事件对象) 361  
    listener interface (监听接口) 358–361  
    using inner classes (使用内部类) 379  
event source (事件源) 359–361  
exceptions (异常)  
    about (介绍) 320, 325, 338  
    catch (catch块) 321, 338  
    catching multiple exceptions (捕获多重异常) 329, 332  
    checked vs. runtime (检查和运行期) 324  
    declaring (声明) 335–336  
    ducking (ducking掉) 335–336  
    finally (finally块) 327  
    flow control (流程控制) 326  
    handle or declare law (处理或声明) 337  
    propagating (传播) 335–336  
    remote exceptions (远程异常) 616  
    throwing (throw块) 323–326  
    try (try块) 321, 338  
executable JAR (可执行的JAR) 585–586, 586  
    with packages (包) 592, 592–593  
exercises (练习)  
    be the... (是……) 88, 118, 266, 310, 395  
    code magnets (排排看) 20, 43, 64, 119, 312, 349, 467, 524–525  
    honeypot 267  
    true or false (是非题) 311, 348, 466, 602  
    what's the declaration (Java声明呢?) 231  
    what's the picture (图呢?) 230  
    which layout manager? (哪一个layout管理器) 424  
    who am I (我是谁?) 45, 89, 394

F

File (文件) 452  
FileInputStream (文件输入流) 441 (见输入/输出)  
FileOutputStream (文件输出流) 432  
FileReader 454 (见输入/输出)  
files  
    File class (File类) 452  
    reading from (读取) 441, 454  
    source file structure (源文件结构) 7  
    writing to (写入) 432, 447  
FileWriter 447  
File class (File类) 452  
final (final标识)  
    class (类) 189, 283  
    methods (方法) 189, 283  
    static variables (静态变量) 282  
    variables (变量) 282, 283  
finally block (finally块) 327  
fireside chats  
    about (介绍) 18  
five minute mystery (见puzzles)  
float (float类型) 51  
FlowLayout (FlowLayout布局) 403, 408–410  
flow control (流程控制)  
    exceptions (异常) 326  
font (字体) 406  
formatting (格式化)  
    dates (日期) 301–302  
    format specifiers (格式化声明) 295–296  
    argument (参数) 300  
    numbers (数字) 294–295  
    printf() (printf()方法) 294  
    String.format() (String.format()) 294  
for loops (for循环) 105  
fully qualified name (完整名称) 154, 157  
    packages (包) 587

## G

garbage collection (垃圾收集器)  
 about (介绍) 40  
 eligible objects (合适对象) 260–263  
 heap (堆) 57, 58  
 nulling references (空引用) 58  
 reassigning references (再分配引用) 58  
**generics** (泛型) 540, 542, 568–574  
 methods (方法) 544  
 wildcards (万用字符) 574  
**getters and setters** (getters和setters) 79  
**ghost town** (荒源小镇) 109  
**giraffe** (长颈鹿) 50  
**girl dreaming** (女孩的梦想)  
 inner classes (内部类) 375  
 Java Web Start (JWS) 596  
**girl in a tub** (澡盆中的女孩) 177  
**girl who isn't getting it** 182–188  
**graphics** (图形) 364–366 (见**GUI**)  
 Graphics2D class (Graphics2D类) 366  
 Graphics object (Graphics对象) 364  
**GregorianCalendar** (GregorianCalendar) 303  
**guessing game** (猜猜看游戏) 38  
**GUI** (图形用户接口) 406  
 about (介绍) 354, 400  
 animation (动画效果) 382–385  
 BorderLayout (BorderLayout管理器) 370–371, 401, 407  
 BoxLayout (BoxLayout管理器) 403, 411  
 buttons (按钮) 405  
 components (元件) 354, 363–368, 400  
 event handling (事件处理) 357–361, 379  
 FlowLayout (FlowLayout管理器) 403, 408  
 frames (框架) 400  
 graphics (图形) 363–367  
 ImageIcon class (ImageIcon类) 365  
 JButton (JButton组件) 400  
 JLabel (JLabel组件) 400  
 JPanel ( JPanel组件) 400, 401  
 JTextArea (JTextArea) 414

JTextField (JTextField) 413  
**layout managers** (布局管理器) 401–412  
**listener interface** (监听接口) 358–361  
**scrolling ( JScrollPane)** (滚动条) 414  
**Swing** (Swing) 354

## GUI Constants (GUI常量)

ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER 415  
 ScrollPaneConstants.VERTICAL\_SCROLLBAR\_ALWAYS 415

## GUI methods (GUI方法)

drawImage() 365  
 fillOval() 365  
 fillRect() 364  
 gradientPaint(). *See also GUI*  
 paintComponent() 364  
 setColor() 364  
 setFont() 406

## GUI Widgets (GUI部件) 354

JButton 354, 405  
 JCheckBox 416  
 JFrame 354, 400, 401  
 JList 417  
 JPanel 400, 401  
 JScrollPane 414, 417  
 JTextArea 414  
 JTextField 413

## H

HAS-A 177–181  
 hashCode() 561  
 HashMap 533, 558  
 HashSet 533, 558  
 Hashtable 558  
**heap** (堆)  
 about (介绍) 40, 57, 236–238  
 garbage collection (垃圾收集器) 40, 57, 58

## I

I/O (输入/输出)

你现在的位置 ▶

681

BufferedReader 454, 478  
 BufferedWriter 453  
 buffers (缓冲区) 453  
 deserialization (解序列化) 441  
 FileInputStream 441  
 FileOutputStream 432  
 FileWriter 447  
 InputStreamReader 478  
 ObjectInputStream 441  
 ObjectOutputStream 432, 437  
 serialization (序列化) 432, 434–439, 437, 446, 460  
 streams (流) 433, 437  
 with sockets (socket) 478  
 if -else 13  
 if statement (if语句) 13  
 immutability, Strings  
 immutability (不变性) 661  
 implements (实现) 224  
 imports  
 static imports (静态imports) 307  
 import statement (import语句) 155, 157  
 increment (递增) 105  
 inheritance (继承)  
 about (介绍) 31, 166–192  
 and abstract classes (抽象类) 201  
 animals (动物) 170–175  
 IS-A (IS-A) 214, 251  
 super (super的使用) 228  
 initializing (初始化)  
 instance variables (实例变量) 84  
 primitives (primitive主数据类型) 84  
 static variables (静态变量) 281  
 inner classes (内部类)  
 about (介绍) 376–386  
 events (事件) 379  
 inner class threesome (内部类) 381  
 InputStreamReader 478  
 instance variables (实例变量)  
 about (介绍) 34, 73  
 declaring (声明) 84  
 default values (默认值) 84

initializing (初始化) 84  
 life and scope (生命周期和范围) 258–263  
 local variables vs. (局部变量) 236–238, 239  
 static vs. (静态) 277  
 instantiation (实例化, 见对象)  
 int (int类型) 50  
 primitive (primitive主数据类型) 51  
 Integer. See wrapper  
 interfaces (接口)  
 about (介绍) 219–227  
 for serialization (序列化) 437  
 implementing (实现) 224, 437  
 implementing multiple (实现多个接口) 226  
 java.io.Serializable 437  
 IP address (IP地址, 见网络章节)  
 IS-A 177–181, 251

## J

J2EE 631  
 JAR files  
 basic commands (基本命令) 593  
 executable (可执行的) 585–586, 592  
 manifest 585  
 running executable (运行期可执行) 586, 592  
 tool (工具) 593  
 with Java Web Start (JWS) 598  
 Java, about (Java介绍) 5, 6  
 javac (见编译器)  
 Java in a Nutshell 158–159  
 java sound 317, 340  
 Java Web Start (JWS) 597–601  
 jnlp file (jnlp文件) 598, 599  
 Jini 632–635  
 JNLP 598  
 jnlp file (jnlp文件) 599  
 JPEG 365  
 JVM (Java虚拟机)  
 about (介绍) 2, 18

JWS (JWS, 见Java Web Start)

# K

keywords (关键字) 53

# L

I 264

layout managers (布局管理器) 401–412

- BorderLayout 370–371, 403, 407
- BoxLayout 403, 411
- FlowLayout 403, 408–410

lingerie, exceptions (异常) 329

LinkedHashMap 533, 558

LinkedHashSet 558

LinkedList 533, 558

linked invocations (链接的调用) 664

List 557

listeners (监听)

- listener interface (监听接口) 358–361

literals, assigning values (赋值)

- primitive (primitive主数据类型) 52

local (局部)

- variables (变量) 85, 236, 236–238, 258–263

locks (锁)

- object (对象) 509
- threads (线程) 509

long 51

loops (循环)

- about (介绍) 10
- break (break语句) 105
- for (for语句) 105
- while (while语句) 115

lost update problem (丢失的更新问题, 见线程)

# M

main() 9, 38

make it stick 53, 87, 157, 179, 227, 278

manifest file (manifest文件) 585

Map 557, 567

Math class (Math类)

- methods (方法) 274–278, 286
- random() 111

memory

- garbage collection (垃圾收集器) 260–263

metacognitive tip (学习密诀) 33, 108, 325

methods (方法)

- about (介绍) 34, 78
- abstract (抽象) 203
- arguments (参数) 74, 76, 78
- final 283
- generic arguments (泛型参数) 544
- on the stack (栈) 237
- overloading (重载) 191
- overriding (覆盖) 32, 167–192
- return (返回值) 75, 78
- static (静态) 274–278

midi 317, 340–346, 387–390

midi sequencer 340–346

MINI Cooper 504

modifiers (修饰符)

- class (类) 200
- method (方法) 203

multidimensional arrays (多维数组) 670

multiple inheritance (多重继承) 223

multiple threads (见线程)

music (见midi)

mystery (见puzzles)

# N

naming (命名, 见RMI)

- classes and interfaces (类和接口) 154–155, 157

- collisions (冲突) 587

- packages (包) 587

networking (网络)

- about (介绍) 473

- ports (端口) 475

你现在的位置 ▶

683

## N - P

    sockets (socket) 475  
    new 55  
    null  
        reference (引用) 262  
    numbers  
        formatting (格式化) 294-295

## O

    ObjectOutputStream 432, 437  
    objects (对象)  
        about (介绍) 55  
        arrays (数组) 59, 60, 83  
        comparing (比较) 209  
        creation (创建) 55, 240-256  
        eligible for garbage collection (可垃圾回收的)  
            260-263  
        equality (相等) 560  
        equals() 209, 561  
        life (生命周期) 258-263  
        locks (锁) 509

    Object class (Object类)  
        about (介绍) 208-216  
        equals() 561  
        hashCode() 561  
        overriding methods (覆盖方法) 563  
    object graph (对象图) 436, 438  
    object references (对象引用) 54, 56  
        assignment (赋值) 55, 262  
        casting (转换) 216  
        comparing (比较) 86  
        equality (相等) 560  
        nulling 262  
        polymorphism (多态) 185-186

    OO (面向对象)  
        contracts (合约) 190-191, 218  
        deadly diamond of death (致命方块) 223  
        design (设计) 34, 41, 79, 166-191  
        HAS-A 177-181  
        inheritance (继承) 166-192  
        interfaces (接口) 219-227  
        IS-A 177-181, 251

    overload (重载) 191  
    override (覆盖) 167-192  
    polymorphism (多态) 183, 183-191, 206-217  
    superclass (父类) 251-256

    operators (运算符)  
        and autoboxing 291  
        bitwise (位) 660  
        comparison (比较) 151  
        conditional (条件) 11  
        decrement (递减) 115  
        increment (递增) 105, 115  
        logical (逻辑) 151  
        shift (移位) 660  
    overload (重载) 191  
    constructors (构造函数) 256  
    override (覆盖)  
        about (介绍) 32, 167-192  
        polymorphism (多态, 见多态)

## P

    packages (包) 154-155, 157, 587-593  
        directory structure (目录结构) 589  
        organizing code (代码组织) 589  
    paintComponent() 364-368  
    parameter (参数, 见arguments)  
    parameterized types (参数化类型) 137  
    parsing an int (见wrapper)  
    parsing text with String.split() 458  
    pass-by-copy (见pass-by-value)  
    pass-by-value 77  
    phrase-o-matic 16  
    polymorphism (多态) 183-191  
        abstract classe (抽象类) 206-217  
        and exceptions (异常) 330  
        arguments and return types (参数和返回类型) 187  
        references of type Object (Object引用类型)  
            211-213  
    pool puzzle (见puzzles)  
    ports (端口) 475

prep code (伪码) 99–102  
 primitives (primitive主数据类型) 53  
   == operator (==运算符) 86  
   autoboxing 288–289  
   boolean 51  
   byte 51  
   char 51  
   double 51  
   float 51  
   int 51  
   ranges (域) 51  
   short 51  
   type 51  
 primitive casting (转换primitive主数据类型)  
   explicit primitive 117  
 printf() 294  
 PrintWriter 479  
 private  
   access modifier (存取修饰符) 81  
 protected 668  
 public  
   access modifier (存取修饰符) 81, 668  
 puzzles  
   five minute mystery (5分钟短剧) 92, 527, 674  
   Java cross 22, 120, 162, 350, 426, 603  
   pool puzzle (泳池迷宫) 24, 44, 65, 91, 194, 232, 396

## Q

quiz card builder 448, 448–451

## R

rabbit 50  
 random() 111  
 ready-bake code (现成码) 112, 152–153, 520  
 reference variables (引用变量, 见对象引用转换) 216  
 registry, RMI 615, 617, 620  
 remote control (远程控制) 54, 57

remote interface (远程接口, 见RMI)  
 reserved words (保留字) 53  
 return types (返回类型)  
   about (介绍) 75  
   polymorphic (多态) 187  
   values (值) 78  
 risky code (风险码) 319–336  
 RMI  
   about (介绍) 614–622  
   client (客户端) 620, 622  
   compiler (编译器) 618  
   Jini (见Jini)  
   Naming.lookup() 620  
   Naming.rebind(). See also RMI  
   registry 615, 617, 620  
   remote exceptions 616  
   remote implementation 615, 617  
   remote interface 615, 616  
   rmic 618  
   skeleton 618  
   stub 618  
   UnicastRemoteObject 617  
   universal service browser (通用服务浏览器) 636–648  
 rmic (见RMI)  
 run()  
   overriding in Runnable interface (覆盖Runnable接口) 494  
 Runnable interface (Runnable接口) 492  
   about (介绍) 493  
   run() 493, 494  
   threads (线程) 493  
 runnable thread state (运行中的线程状态) 495  
 Ryan and Monica (杰纶与沛晨) 505–506  
   introduction (介绍) 505–506

## S

scary objects (怪物对象) 200  
 scheduling threads (线程调度)  
   scheduling (调度) 496–498

## S - T

scope  
variables (变量) 236–238, 258–263

scrolling (JScrollPane) (滚动) 414

serialization (序列化) 434–439, 446  
deserialization (解序列化) 460  
interface (接口) 437  
ObjectInputStream (见输入/输出)  
objectOutputStream 432  
objects (对象) 460  
object graph (对象图) 436  
reading (见输入/输出)  
restoring 460 (见输入/输出)  
saving 432  
serialVersionUID 461  
transient 439  
versioning 460, 461  
writing 432

server (服务器)  
socket 483. *See also* socket

servlet 625–627

Set 557  
importance of equals() (equals()的重要性) 561  
importance of hashCode() (hashCode()的重要性) 561

short 51

short circuit logical operators 151

sink a dot com 96–112, 139–150

skeleton. *See* RMI

sleep() 501–503

sleeping threads (sleep线程) 501–503

snowboard 214

socket  
about (介绍) 475  
addresses (地址) 475  
creating (创建) 478  
I/O (输入/输出) 478  
ports (端口) 475  
reading from (读数据) 478  
server (服务器) 483  
TCP/IP 475

writing to (写数据) 479

sorting (排序)  
Collections.sort() 534, 539, 547  
Comparable interface (Comparable接口) 547, 549  
Comparator 551, 553  
TreeSet 564–566

source files (源文件)  
structure of (结构) 7

specifiers (说明)  
format specifiers (格式化说明) 295, 298  
argument specifier (参数说明) 300

stack (堆栈)  
heap vs. (堆) 236  
methods on (方法) 237  
scope (范围) 236  
threads (线程) 490  
trace 323

static  
enumerated types (枚举类型) 671  
initializer (初始化) 282  
Math class methods (Math类的方法) 274–278  
methods (方法) 274–278  
static imports (307)  
variables (变量) 282

streams (流) 433. (见输入/输出)

String  
arrays (数组) 17  
concatenating (连接) 17  
methods (方法) 669  
parsing (解析) 458  
String.format() 294–297  
String.split() 458

StringBuffer/StringBuilder  
methods (方法) 669

stub (见RMI)

subclass (子类)  
about (介绍) 31, 166–192

super (父类) 228  
about (介绍) 31

superclass (父类)

about (介绍) 166–192, 214–217, 228  
 super constructor (父类构造函数) 250–256  
 Swing. See GUI  
 synchronized  
     methods (方法) 510 (见线程)  
 syntax (语法)  
     about (介绍) 10, 12  
 System.out.print() 13  
 System.out.println() 13

## T

talking head (关于head的谈论) 203  
 TCP ports (TCP端口) 475

Telluride 30

testing

    extreme programming (极限编程) 101

text

    parsing with String.split() (用String.split()解析) 458  
     read from a file (读取文件, 见输入/输出)  
     write to a file (写入文件) 447

text area (JTextArea) 414

text field (JTextField) 413

Thread.sleep() 501–503

threads (线程)

    about (介绍) 489–515  
     deadlock (死锁) 516  
     locks (锁) 509  
     lost update problem (丢失更新问题) 512–514  
     run() 493, 494  
     Runnable 492, 493, 494  
     Ryan and Monica problem (杰伦和沛晨的问题) 505–507  
     scheduling (调度) 496, 496–498  
     sleep() 501–503  
     stack (堆栈) 490–491  
     start() 492  
     starting 492  
     states (状态) 495, 496

summary (摘要) 500, 517  
 synchronized (同步化) 510–512  
 unpredictability (不同预测性) 498–499

throw

    exceptions (异常) 323–326  
     throws 323–326

transient 439

TreeMap 558

TreeSet 533, 558, 564–566, 566

try

    blocks (块) 321, 326

type (类型) 50

    parameter (参数) 137, 542, 544

type-safety (类型安全) 540

    and generics (泛化) 540

## U

universal service browser (通用服务浏览器) 636–648

## V

variables (变量)

    assigning (赋值) 52, 262  
     declaring (声明) 50, 54, 84, 236–238  
     local (局部) 85, 236–238  
     nulling 262  
     primitive (primitive主数据类型) 51, 52  
     references (引用) 54, 55, 56, 185–186  
     scope (范围) 236–238  
     static (见static)

variable declarations (声明变量) 50

    instance (实例) 84

    primitive (primitive主数据类型) 51

    reference (引用) 54

virtual method invocation (虚拟方法调用) 175

## W

web start. See Java Web Start

## W

while loops (while循环) 11, 115

wildcard (万用字符) 574

wine (酒) 202

wrapper (包装) 287

autoboxing 288–289

conversion utilities 292

Integer.parseInt() 104, 106, 117

writing (见输入/输出)

## 这不是结束 带着你的大脑来参观

[wickedlysmart.com](http://wickedlysmart.com)

你知道我们的网站吗？上面有些  
精采的习题解答，以及已经输入  
好的程序代码，还有作者每日更新  
的 blog！



你现在的位置 ▶

689