目　录

无忧书库
http://ebook.51soft.com

译者序

第一部分　API 速查 …… 1

一、JAVA. APPLET …… 2

1. 类 …… 2

Applet Extends java. awt. Panel …… 2

2. 接口 …… 8

AppletContext …… 8

AppletStub …… 9

AudioClip …… 10

二、JAVA. AWT …… 11

1. 类 …… 11

BorderLayout Implements LayoutManager …… 11

Botton Extends Component …… 12

Canvas Extends Component …… 13

CardLayout Implements LayoutManager …… 13

CheckBox Extends Component …… 17

CheckboxGroup …… 18

CheckboxMenuItem Extends MenuItem …… 19

Choice Extends Component …… 19

Final Color …… 20

Abstract Component Implements ImageObserver …… 23

Abstract Container Extends Component …… 30

Dialog Extends Window …… 32

Dimension …… 33

Event …… 33

FileDialog Extends Dialog ····· 38
FlowLayout Implements LayoutManager ····· 40
Font ····· 41
FontMetrics ····· 43
Frame Extends Window Implements MenuContainer ··· 45
Abstract Graphics ····· 47
GridBagConstraints Implements java.lang.Cloneable ··· 54
GridBagLayout Implements LayoutManager ····· 56
GridLayout Implements LayoutManager ····· 62
Abstract Image ····· 63
Insets Implements Cloneable ····· 64
Label Extends Component ····· 65
List Extends Component ····· 66
MediaTracker ····· 69
Menu Extends MenuItem Implements
MenuContainer ····· 71
MenuBar Extends MenuComponent
Implements MenuContainer ····· 72
Abstract MenuComponent ····· 76
MenuItem Extends Menulomponent ····· 76
Panel Extends Coutainer ····· 77
Point ····· 78
Polygon ····· 78
Rectangle ····· 79
Scrollbar Extends Component ····· 81
TextArea Extends TextComponent ····· 85
TextComponent Extends Component ····· 87
TextField Extends TextComponent ····· 88
Toolkit ····· 89
Window Extends Container ····· 92

2. 接口 …… 92
LayoutManager …… 92
MenuContainer …… 93
三、**JAVA. AWT. IMAGE** …… 94
1. 类 …… 94
Abstract ColorModel …… 94
CropImageFilter Extends ImageFilter …… 95
DirectColorModel Extends ColorModel …… 97
FilteredImageSource Extends Object Implements
ImageProducer …… 98
ImageFilter Implements ImageConsumer, Cloneable … 99
IndexColorModel Extends ColorModel …… 104
Memory ImageSowce Implements ImageProducer …… 105
PixelGrabber Implements ImageConsumer …… 107
Abstract RGBImageFilter Extends ImageFilter …… 109
2. 接口 …… 112
ImageConsumer …… 112
ImageObserver …… 114
ImageProducer …… 115
四、**JAVA. IO** …… 116
1. 类 …… 117
BufferedInputStream Extends FilterInputStream …… 117
BufferedOutputStream Extends FilterOutputStream … 118
ByteArrayInputStream Extends InputStream …… 119
ByteArrayOutputStream Extends OutputStream …… 120
DataInputStream Extends FilterInputStream
Implements DataInput …… 122
DataOutputStream Extends FilterOutputStream
Implements DataOutput …… 123
File …… 125

FileDescriptor …… 127
FileInputStream Extends InputStream …… 128
FileOutputStream Extends OutputStream …… 129
FilterInputStream Extends InputStream …… 130
FilterOutputStream Extends OutputStream …… 132
Abstract InputStream …… 133
LineNumberInputStream Extends FilterInputStream … 134
Abstract OutputStream …… 135
PipedInputStream Extends InputStream …… 135
PipedOutputStream Extends OutputStream …… 140
PrintStream Entends FilterOutputStream …… 140
PushbackInputStream Extends FilterInputStream …… 142
RandomAccessFile Implements DataOutput
DataInput …… 143
SequenceInputStream Extends InputStream …… 146
StreamTokenizer …… 147
StringBufferInputStream Extends InputStream …… 149
2. 接口 …… 150
DataInput …… 150
DataOutput …… 151
FilenameFilter …… 152
五、JAVA.LANG …… 153
1. 类 …… 153
Boolean …… 153
Character …… 154
Class …… 156
Abstract ClassLoader …… 157
Compiler …… 158
Double Extends Number …… 158
Float Extends Number …… 160

Integer Extends Number …… 162
Long Extends Number …… 164
Math …… 166
Abstract Number …… 168
Object …… 169
Abstract Process …… 170
Runtime …… 171
Abstract SecurityManager …… 174
String …… 177
StringBuffer …… 181
System …… 186
Thread Implements Runnable …… 189
ThreadGroup …… 194
Throwable …… 197
2. 接口 …… 198
Cloneable …… 198
Runnable …… 198
六、JAVA.NET …… 203
1. 类 …… 203
Abstract ContentHandler …… 203
DatagramPacket …… 203
DatagramSocket …… 204
Final InetAddress …… 205
ServerSocket …… 206
Socket …… 207
Abstract SocketImpi …… 209
URL …… 211
Abstract URLConnection …… 213
URLEncoder …… 217
2. 接口 …… 218

ContentHandlerFactory …… 218
SocketImplFactory …… 219
URLStreamHandlerFactory …… 219
七、JAVA.UTIL …… 220
1. 类 …… 220
BitSet Implements java.lang.Cloneable …… 220
Date …… 222
Abstract Dictionary …… 225
Hashtable Extends Dictionnary
Implements java.lang.Cloneable …… 226
Observable …… 230
Properties Extends Hashable …… 231
Random …… 232
Stack Extends Vector …… 233
String Tokenizer Implements Enumeration …… 234
Vector Implements java.lang.Cloneable …… 235
2. 接口 …… 237
Enumeration …… 237
Observer …… 239
八、EXCEPTIONS …… 240
java.awt …… 241
java.io …… 241
java.lang …… 241
java.net …… 243
java.util …… 243
九、ERRORS …… 244
java.awt …… 244
java.lang …… 244
第二部分　语法速查 …… 247

Abstract 249
Break 249
Catch 249
Class 250
Continue 250
Do...While 251
Else 252
Extends 252
Final 253
Finally 253
For 254
If 255
Implements 255
Import 255
Instanceof 256
Modifiers 256
Native 258
New 258
Package 258
Public 258
Private 259
Return 259
Super 259
This 260
Throw 260
Throws 261
Static 261
Switch 262
Synchronized 262

Try ····· 263
While ····· 264
附录　行为索引 ····· 265
Animation ····· 265
Applets ····· 265
Applications ····· 265
Communication ····· 266
Image Processing ····· 266
Input and Output ····· 266
Math ····· 267
User Interfaces ····· 267

第一部分　API 速查

这一部分讲述了全部的 Java API。API 即应用程序接口，是一组类和接口，它们独立于平台，是建立所有 Java 小应用程序和应用程序的基础。尽管程序的风格与程序员的策略有关，但这些类和接口为程序员开发应用提供了开发工具。

一、JAVA. APPLET

java. applet 为用户提供了一个类和三个接口，它是最有用的一个包，因为它是所有小应用程序（Applet）的基础。Applet 类非常重要，因为在创建小应用程序时需要它。AppletContext 接口也很有用，其作用是与浏览程序相互配合。

1. 类

Applet Extends java. awt. Panel

这个类是 Java 小应用程序的核心，是从 java. awt. Panel 类导出的，Applet 类具有 java. awt. Container 类所建立的所有方法。java. awt. Container 类能够使用户建立自己所需要的用户接口。读者若要创建一个小应用程序，必须要扩充 Applet 类。

如上所述，用户需要创建一个 Java 小应用程序，须首先建立一个类，它是 java. applet. Applet 类的扩充。但若要运行小应用程序，须将其嵌入在一个 HTML 页中。这是通过标志符<APPLET>实现的，它定义小应用程序的名称、高度和宽度，还可以定义一些附加的参数——小应用程序可以获得的信息。

图 1 中所表示的 HTML 页定义了二个参数：NAME 和 KEY（参见程序 1）。

程序 1　关于一个小应用程序的 HTML 页样本

```
<HEAD>
<TITLE>Applet Example</TITLE>
<BODY>
This text is above the applet. <P>

<applet code= "examp. class" width=500 height=100>
```

<param name=NAME value= "Mike" >

<param name=KEY value= "j" >
</applet>
<P>This text is below the applet.<P>
</BODY>

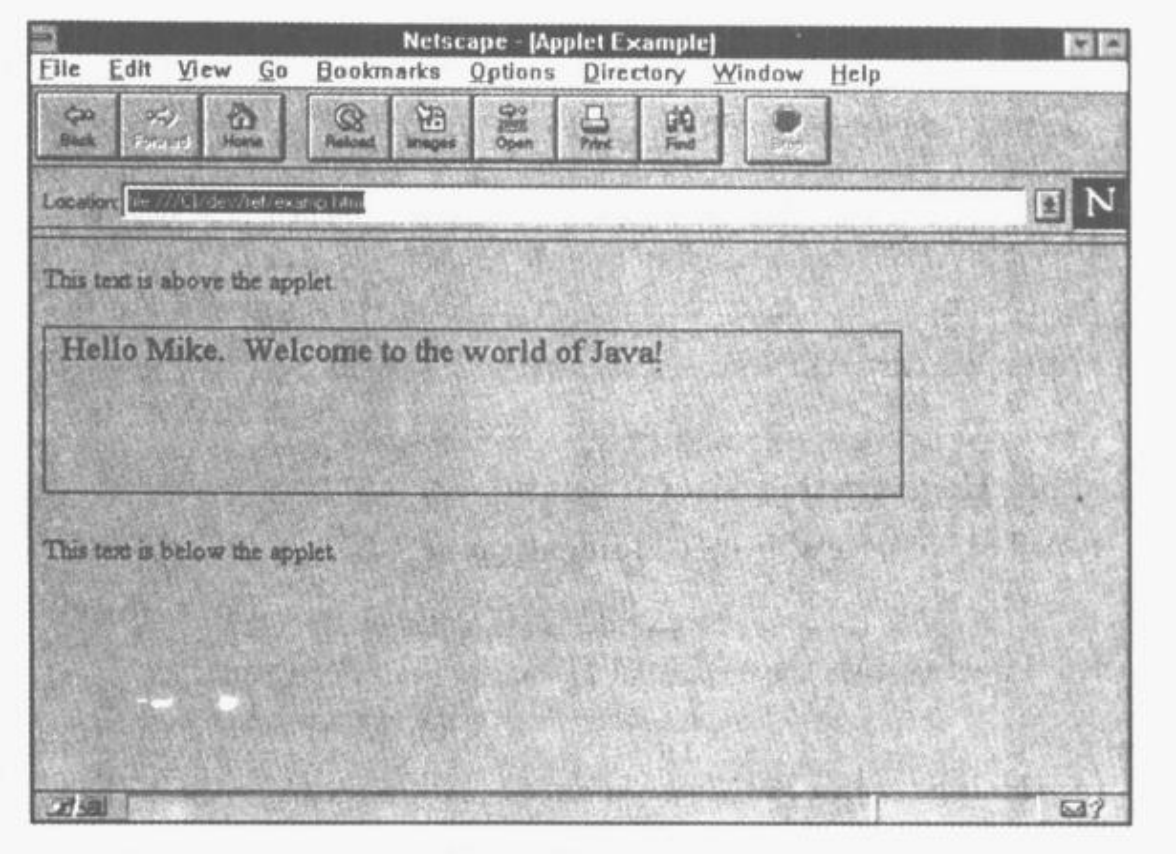


图 1　由代码和程序 1 中的 HTML 生成的简单 Applet 过程

这段程序将把类 examp 加载，examp 类被存储在名为 examp.class 的文件中。程序 2 是有关 examp 类的一段完整的源程序代码。

程序 2　Applet 样本

```
import java.applet.Applet;
import java.awt.*;

public class ExampleApplet extends Applet
```

```
{
  String userName;
  char favoriteKey;
  int appletaWidth, appletHeight,fontHeight;
  Font fontToUse;
  FontMetrics fm;

  /* This class will be called before the applet is started. As a
     result,it can be used to perform preparatory operation. */

  public void init() {
    userName=new String(); //create a new,empty string
    appletWidth=this.size().width - 2 //uses
                    //Component.size()
    appletHeigth=this.size().height - 2;
    fontToUse=new Font("TimesRoman", Font.PLAIN,
                              20); //create a new Font object
    fm=getFontMetrics(FontToUse);      //obtain the
                                  //FontMetrics for the font
    fontHeight=fm.getMaxAscent(); /*maximum size that
                                one of the font's characters may
                                      extend upwards     */
    getInfo();          //get info from HTML tag   }

  /*  This method uses the getParameter() method to obtain
        the information passed to the applet in the applet
      tag. This method assumes that there will be two
      tags, titled "NAME" and "KEY" in the HTML page. */

  void getInfo(){
```

```
    userName=getParameter("NAME");
    favoriteKey=getParameter("KEY").charAt(0);
    //we want the first character in the returned string
    }

    /* This method is automatically called whenever the
       mouse is depressed within the applet */

public boolean mouseDown(Event evt, int x, int y){
    showStatus("Mouse Click at ("+x+", "+y+")");
    return true;
}

    /* This method is automatically called whenever a key is
       depressed within the applet. */
pubic boolean keyDown(Event evt,int key){
    if(key==favoriteKey){
      showStatus("You pressed "+ favoriteKey +
                                    ",my favorite key!");
      return true;
    }
    return false;
}

/* This method is automatically called whenever the applet
   needs to be drawn. */

public void paint(Graphics g){
    g.setFont(fontToUse);
    g.drawRect(0,0,appletWidth,appletHeight);
    g.drawString("Hello"+userName+
```

```
        ". Welcome to the world of Java!",10,fontHeigth+3);
    }
}
```

在这个例子中所用的类及 Font、FontMetrics 和 Graphics 将在 AWT 包那一节介绍。但读者应注意到，在这个小应用程序中的 5 个方法（Method）中只有一个方法 getInfo（）被该小应用程序直接地调用。而其他则被浏览程序或基于 Java. applet. Applet 的某一个类调用。某些方法，例如 init（），在某些特定时刻被浏览程序自己调用。另外一些方法，包括 keyDown（）和 mouseDown（）在小应用程序内被调用来处理某些特殊的用户事件。在 java. awt. Component 类中读者可以发现更多这样的方法。

方法

※ void destroy（）

在执行了一个小应用程序之后可用这个方法进行清除操作。这个方法的缺省情况是括号为空。若要使这个方法做任何一种操作，须给予替换值。当小应用程序停止执行时，这个方法自动地被浏览程序调用。

※ AppletContext getAppletContext（）

此方法返回小应用程序的 AppletContexe，浏览程序（即 Netscape 或 Appletviewer）用它浏览小应用程序。

※ String getAppletinfo（）

这个方法用来返回诸如版本号或所有者名称这样的信息，如果没有给予一个替换值，那么将返回一个空串。

※ AudioClip getAudioClip（URL url）

在所指定的 URL 中加载一段驻留的音频剪辑信息。

※ AudioClip getAudioClip（URL url，String clipname）

根据所指定的 URL 和名字加载一段音频剪辑信息。

※ URL getCodeBase（）

返回小应用程序的 URL，通常是包含当前小应用程序页的 URL，且没有文件名。

※ URL getDocumentBase ()

返回驻留当前小应用程序页的 URL。

※ Image getImage (URL url)

加载指定的图象，可能不是立即完成处理，因为实际的加载要花费一段时间。

※ Image getImage (URL url, String imagename)

加载指定的图象，可能不是立即完成处理，因为实际的加载要花费一段时间。

※ String getParameter (String labelname)

返回 HTML 页中指定的参数。

※ String [] [] getParameterinfo ()

返回 HTML 页中所有的参数。

※ void init ()

此方法可用来执行某些准备动作。当此方法为空值时，通常用来执行准备任务，例如建立流和加载信息。这个方法总是被浏览程序调用。

※ boolean isActive ()

如果小应用程序当前正在运行，则返回一个真值。

※ void play (URL url)

播放指定的音频剪辑信息。

※ void play (URL url, String audioClipName)

播放指定的音频剪辑信息。

※ void resize (int width, int height)

重新估量小应用程序，指定其尺寸。在 Netscape 中，这个方法不起作用。若要设定小应用程序尺寸，使用 HTML 标志中的宽度和高度参数。

※ void resize (Dimension d)

重新估量小应用程序，指定其尺寸。在 Netscape 中，这个方法不起作用。若要设定小应用程序尺寸，使用 HTML 标志中的宽度和高度参数。

※ void setStub (AppletStub stub)

设定小应用程序的 AppletStub。这个任务是由浏览程序自动执行的，所以读者不必使用它。

※ void showStatus (String msg)

在浏览程序底部显示一条信息，通常这是与用户通信的一个最简便方法。

※ void start ()

当创建小应用程序后，通过给这种方法以替换值，可以立即执行某些任务。这种方法被运行系统自动调用。

※ void stop ()

当小应用程序停止运行时，通过给这种方法以替换值，可以执行某些任务。这种方法被运行系统自动调用。此方法并不常用，但可利用它执行某些清除操作。

2. 接口

AppletContext

AppletContext 接口为用户提供了与浏览程序交互的手段。通过 getAppletContext () 方法，任何一个小应用程序均可获得 AppletContext。尽管 AppletContext 内核的工作与浏览程序有关，但它能够执行多种任务。

▶请参见程序 18，“Runnable Example—HTML”

方法

※ Applet getApplet (String name)

返回给定名称的小应用程序。

※ Enumeration getApplets ()

返回给定页上的所有小应用程序。

※ AudioClip getAudioClip (URL url)

返回所指定的音频剪辑。

※ Image getImage (URL url)

返回指定的图象。

※ void showDocument (URL url)

展现当前上下文中所指定的文件。

※ void showDocument (URL url, String target)

请求浏览程序展现所指定的文件，展现的文件位于特定的位置——浏览器窗口或 Netscape 帧。下表列出了 target 的一些选项：

值	文件显示
"_self"	当前帧
"_parent"	父帧
"_top"	最顶部的帧
"_blank"	在一个新的或尚未命名的浏览程序窗口中的文件
"aNameofYourChoice"	建立一个新浏览程序窗口，具有所指定的名字。此后，使用这个名字，可以显示此窗口中的其他一些文件

※ void showStatus (String mesg)

显示浏览程序底部的一个字符串。

AppletStub

AppletStub 接口主要是在创建一个小应用程序的观察程序时使用。它还被 Applet 类用来检索信息以及与浏览程序进行交互操作。由于交互操作已由 Applet 类处理，所以一般来说用户不需要使用 AppletStub 类。

方法

※ void appletResize (int width, int height)

当小应用程序需要重新估量尺寸时调用这个方法。

※ AppletContext getAppletContext ()

返回小应用程序的上下文。

※ URL getCodeBase ()

返回无文件名的基本 URL。

※ URL getDocumentBase ()

返回当前 URL。

※ String getParameter (String mane)

返回所指定的参数。

※ boolean isActive ()

当小应用程序运行时返回一个真值。

AudioClip

AudioClip 接口是一段音频剪辑性能的高级表示。当一些接口对象不能被建立时，扩充 AudioClip 接口的对象可以通过这个接口来处理。通常当一个 applet 类中的 getAudioClip()返回一个我们知之很少的类时，就认为我们可以使用 loop()、play()和 stop()方法，来执行这个接口。

方法

※ void loop ()

开始播放一段音频剪辑，而且周而复始地播放。

※ void play ()

播放一个音频剪辑。

※ void stop ()

停止播放音频剪辑。

二、JAVA.AWT

Java.awt（Abstract Window Toolkit）可以使用户创建丰富的、具有吸引力和极有用途的接口。除了拥有诸如布局设计管理器（Layout Manager）和容器（Container）这样的管理工具外，还具有一些具体的应用工具，例如 Button 和 TextField。

1. 类

BorderLayout Implements LayoutManager

BorderLayout 类可用来设计和管理用户接口中的一些部件。BorderLayout 是一个画面边界定位类，可设置 5 种部件。当在一个布局上添加一个部件时，通过“North”、“South”、“East”、“West”和“Center”定义。

方法

※ BorderLayout（）

根据括号中的水平间距 hgap 和垂直间距 vgap 可创建一个新的 BorderLayout。这两个字段决定布局中部件之间的调整量。

※ BorderLayout（int hgap，int vgap）

根据指定的 hgap 和 vgap 创建一个新的 BorderLayout。这两个字段决定布局中两个定位部件之间的调整量。

※ void addLayoutComponent（String location，Component comp）

在布局设计中，把一个部件添加到指定的位置。有 5 个选项：“North”、“South”、“East”、“West”和“Center”。

※ void LayoutContainer（Container target）

调整目标容器中的部件，使之与 BorderLayout 相适应。

※ Dimension minimumLayoutSize（Container target）

返回在指定容器中用于部件的最小尺寸。这个值是容器中用于添加部件的最小空间尺寸，与各部件的最小尺寸有关。

※ Dimension preferredLayoutSize (Container target)

返回所选择的容器的尺寸，这是以所选的每个部件的尺寸及容器中指定的空间为基础的。

※ void removeLayoutComponent (Component comp)

去掉画面布局中指定的部件。

※ String toString ()

返回一个描述水平间距 hgap 和垂直间距 vgap 的字符串。

Botton Extends Component

Button 类可为用户接口产生一个标准按钮（参见图 2）。按钮是一个标准部件，可施加于任何一个布局管理器类。每个按钮的标志串可给用户或程序员提供按钮标识。每当一个按钮被按下，那么一个 ACTION _ EVENT 将被传递到驻留该按钮的容器中。为了区分屏幕上不同的按钮，Event 包含一个幅角字符串，串中包含被按下按钮的标记。

▶参见程序 6，“GridbagLayout 举例”



图 2　Windows NT 中的一个按钮

方法

※ Button ()

创建一个未标记的按钮。

※ Button (String label)

按照指定的标记创建一个按钮。当此按钮被按下时，标志串还可在所产生的 Event 中作为幅角值。

※ synchronized void addNotify ()

给按钮创建一个同位体。

※ String getLabel ()

返回按钮的标记。

※ protected String paramString ()

返回说明按钮标记的字符串。

※ void setLabel (String label)

设置按钮标记。

Canvas Extends Component

Canvas 类可为用户提供一个可构造的简单背景。例如 Canvas 类建立一个同位体并涂上简单的灰色背景。如果用户要扩充 Canvas 类，还可建立自己定制部件——非常有用的一些东西。

方法

※ Canvas ()

建立一个新的背景。

※ synchronized void addNotify ()

创建背景的同位体。

※ void paint (Graphics g)

给屏幕涂色。缺省情况下，这个方法涂上与背景相同的颜色。

CardLayout Implements LayoutManager

CardLayout 类可使用户在一系列部件之间进行滚动。通常卡片 (card) 指的是容器 (container)，每个都带有一个标题。卡片可通过标题来检索。所以，用户可以通过这些卡片进行滚动，滚动次序可以是顺序的，也可以通过名字来选择卡片滚动的顺序。

程序 3 用来建立一个小应用程序。如图 3 所示。如你所见，这个小应用程序使用了一个 Choice 部件，使得用户可在不同的卡片上滑动选择。

程序 3 “CardLayout 举例”

```
import java.awt.*;
import java.applet.Applet;

public class CardLayoutExample extends Applet {
```

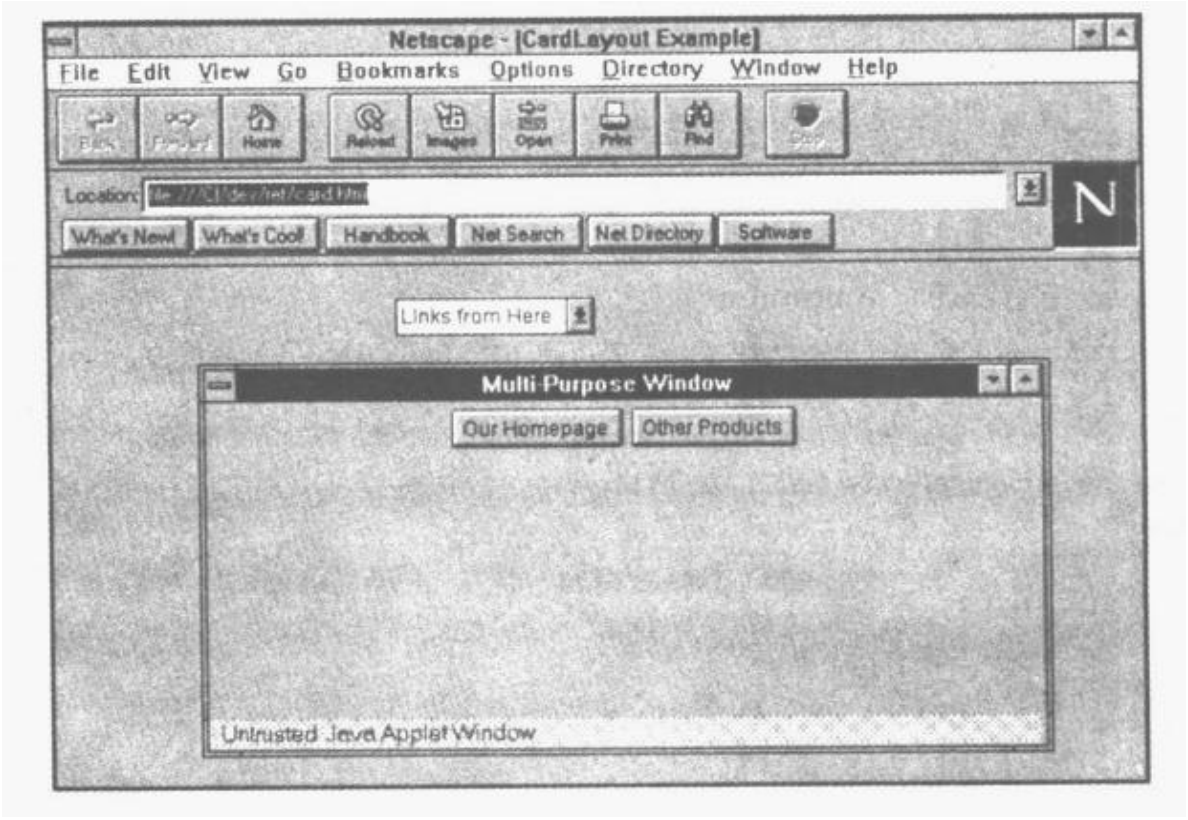


图 3　CardLayout 样本

```
Frame sidecard;
private static String LINKS="Links from Here";
private static String ORDERFORM="Order Form";

public CardLayoutExample() {
   Choice selector=new Choice();   //creates a selector
                                   //on the original page

   selector.addItem(LINKS);
   selector.addItem(ORDERFORM);
   add(selector);

   sidecard=mew Frame("Multi-Purpose Window");
                        //creates the side frame
```

```
    sidecard.setLayout(new CardLayout());

    Panel p=new Panel();//create the individual panels
    p.add(new Button("Our Homepage");
    p.add(new Button("Other Products");
    sidecard.add(LINKS,p);

    p=new Panel();
    p.setLayout(new FlowLayout());
    p.add(new TextField("Name",20));
    p.add(new TextField("Address",50));
    p.add(new TextField("Phone",20));
    p.add(new Button("Order"));
    sidecard.add(ORDERFORM,p);

    sidecard.show();   //shows the side frame
  }

  public boolean action(Event evt,Object arg) {
    CardLayout display;
    if (evt.target instanceof Choice){/* if the event
                                        occurred in a Choice */
    display=(CardLayout)sidecard.getLayout();
                        //obtains control of the frame
    display.show(sidecard.arg.toString());
                        //obtains control of the frame
    return true;
  }
  return flase;
}
```

```
public synchronized boolean handleEvent(Event evt){
  if (evt.id==Event.WINDOW_DESTORY){
    this.hide();
        return true;
    }
    return super.handleEvent(evt);
  }
}
```

方法

※ CardLayout ()

建立一个新的 CardLayout 管理程序。

※ CardLayout (int hgap, int vgap)

根据指定的水平距离和垂直距离建立一个新的 CardLayout 管理程序。

※ void addLayoutComponent (String name, Component comp)

给容器添加一个具有指定名称的部件。

※ void first (Container parent)

显示布局中的第一个卡片。

※ void last (Container prent)

显示布局中最后一个卡片。

※ void LayoutContainer (Container parent)

根据 CardLayout 的参数安排父容器。

※ Dimension minimumLayoutSize (Container parent)

返回布局所需的最小尺寸，这个值是以参数的最小尺寸为基础的。

※ void next (Container parent)

显示下一个卡片。

※ Dimension preferredLayoutSize (Container parent)

返回在父容器中所选择的布局尺寸。这是以容器中所有选择的部件尺寸为基础的。

※ void previous (Container parent)

显示前面的卡片。

※ void removeLayoutComponent (Component comp)

从布局中撤销指定的卡片。

※ void show (Container parent, String cardname)

显示在父容器中所指定的卡片。

※ String toString ()

返回一个字符串，它描述该布局中水平距离 hgup 和垂直距离 vgap 的值。

CheckBox Extends Component

Checkbox 类提供了获取用户输入的最方便的方法，如图 4 所示。它类似于一个标准的复选框：客户计算机上的一个正文标号和一个框。



图 4 复选框样本

▶参见“CheckboxGroup”

▶参见“Checkbox MenuItem 扩展 MenuItem”

▶参见程序 6，“GrdbagLayout Example”

方法

※ Checkbox ()

生成一个未标号的复选框。

※ Checkbox (String label)

建立一个复选框并带有指定的标号。

※ Chexkbox (String label, CheckboxGroup group, boolean state)

建立一个带有指定标号的复选框，并把它放在 CheckGroup (复选框组) 中。参数 state 决定是否初始校验。

※ synchronized void addNotify ()

创建复选框的同位体。

※ CheckboxGroup getCheckboxGroup ()

返回当前复选框所在的复选框组。

※ String getLabel ()

返回复选框的标号。

※ boolean getState ()

返回复选框的状态（若被复选，则为 true，否则为 false）。

※ protected String paramString ()

返回一个字符串，描述复选框的参数，例如它的状态和位置。

※ void setCheckboxGroup (CheckboxGroup g)

为当前复选框设置复选框组。

※ void setLabel (String label)

设置复选框的标号。

※ void setState (boolean state)

设置复选框的状态（校验或不校验）。

CheckboxGroup

使用 CheckboxGroups 可以创建一个选项按钮组——一组复选框，每次只能选择其中一个，这种排他性选择是很有用的。

若要使用 CheckboxGroup 类，首先要建立复选框，并根据自己的习惯安排它们的布局，然后通过 setCheckboxGroup () 方法把它们分配给复选框组。

▶参见“checkbox 扩展 Component”

方法

※ CheckboxGroup ()

创建一个新的复选框组。

※ Checkbox getCurrent ()

返回当前所选择的复选框。

※ synchronized void setCurrent (Checkbox box)

把指定的复选框设置为被选的。

※ String toString ()

返回一个字符串，描述当前选择的复选框。

CheckboxMenuItem Extends MenuItem

CheckboxMenuItem 是一个处于触发状态的 MenuItem（参见图 5）把一个 CheckboxMenuItem 放置在菜单列内，当复选框被选时，则显示一个复选标记。

▶参见“Checkbox 扩展 Component”

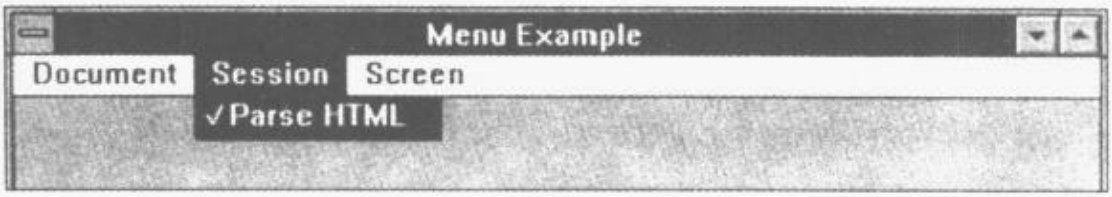


图 5　Windows NT 中的 CheckboxMenuItem

方法

※ CheckboxMenuItem（String label）

根据指定的标号产生一个新的复选框菜单选项。

※ synchronized void addNotify（）

产生复选框菜单选项的同位体。

※ boolean getState（）

返回复选框的当前状态。

※ String paramString（）

返回一个字符串，描述复选框的值。

※ void setState（boolean state）

设置复选框的状态。

Choice Extends Component

Choice 类的部件产生一个标准的弹出菜单，如图 6 所示，用户可以选择一项。

▶参见程序 6，“GridbagLagant Example”

方法

※ Choice（）

创建一个新的选择部件。

※ synchronized void addItem（String label）

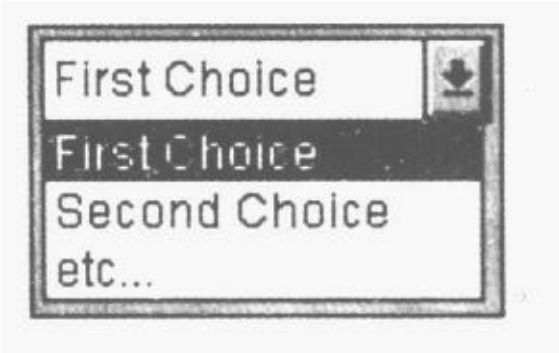


图 6　Windows NT 中的一个选项

根据指定的标号所列出的选择列表添加一个选择。

※ synchronized void addNotify ()

建立一个选择列表的同位体。

※ int countItems ()

返回当前存在于选择列表内的项号。

※ String getItem (int pos)

返回 Choice 表列中指定位置的选择列表的标号。

※ int getSelectedIndex ()

返回当前选择项的索引。

※ String getSlectedItem ()

返回当前选择项的标号。

※ protected String paramString ()

返回一个字符串，包含当前被选项的标号。

※ void select (String label)

根据指定的标号选择项目。

※ synchronized void select (int pos)

选择指定位置的项。

Final Color

Color 类可以为用户的应用程序提供各种颜色的选择，用户还可以通过这个类创建自己所需的颜色。

▶参见“抽象图形”

字段

- final static Color black
- final static Color blue

- final static Color cyan
- final static Color darkGray
- final static Color gray
- final static Color green
- final static Color lightGray
- final static Color magenta
- final static Color orange
- final static Color pink
- final static Color red
- final static Color white
- final static Color yellow

方法

※ Color (int rgb)

产生一种由一个整数所表示的颜色，0－7 位表示蓝色成分，8－15 位表示绿色成分，16－23 位表示红色成分。

※ Color (float red _ value, float green _ value, float blue _ value)

根据指定的值创建一个新的颜色，值的范围从 0 到 1.0。

※ Color (int red _ value, int green _ value, int blue _ value)

根据指定的值创建一个新的颜色，值的范围从 0 到 255。

※ Color brighter ()

返回当前颜色的更亮的版本，这是将当前颜色的每个整数成分除以 0.7 得到的。

※ Color darker ()

返回当前颜色的更暗的版本，这是将当前颜色的每个整数成分乘以 0.7 得到的。

※ boolean equals (Object otherObject)

将一种颜色与另一对象比较，如果另一个对象也是一种颜色，它们只能是相等的。

※ int getBlue ()

返回颜色的蓝色成分。

※ static Color getColor (String name)

返回一种颜色，其名称被认为是在系统特性颜色说明中所指定的。如果未找到此系统特性，则此方法返回空值 (NULL)。

※ static Color getColor (String name, int default)

返回一种颜色，其名称被认为是在系统特性颜色说明中所指定的。如果未找到此系统特性，则此方法返回由缺省值所代表的颜色。

※ static Color getColor (String name, Color default)

返回一种颜色，其名称被认为是在系统特性颜色说明中所指定的。如果未找到此系统特性，则此方法返回 default。

※ int getGreen ()

返回颜色的绿色成分。

※ static Color getHSBColor (float hue, float saturation , float brightness)

根据指定的 HSB 值（色彩，浓度和亮度）创建一个颜色，这些值的范围是 0.0 到 1.0。

※ int getRed ()

返回颜色的红色成分。

※ int getRGB ()

返回当前颜色的 RGB 值。这是个整数，0—7 位是蓝色，8—15 位是绿色，16—23 位是红色。

※ static hashCode ()

返回当前颜色的散列代码，这个值与从 getRGB () 返回的整数相同。

※ static int HSBtoRGB (float hue, float saturation, float brightness)

产生一个整数表示的颜色，它由所说明的色彩、浓度和亮度定义。在此整数中 0—7 位是蓝色，8—15 位是绿色，16—23 位是红色。

※ static float [] RGBtoHSB (int red, int green. int blue, float hsbvals [])

返回颜色的色彩、浓度和亮度值。此颜色是由红、绿和蓝色值所定义的。数组中 hsbvals [0] =hue (色彩), hsbvals [1] = saturation (浓度), hsbvals [2] =brightness (亮度)。

※ String toString ()

返回一个字符串，包含颜色的红、绿和蓝色值。

Abstract Component Implements ImageObserver

Component 类是 AWT 包中可使用的各种部件的基础，如按钮 (Button) 和滚动条 (Scrollbar)。当一些特殊部件拥有与它们功能有关的方法时，Component 类为其提供相匹配的方法，这些方法被设计用来促进某些基本功能的执行，例如事件处理。

读者还将发现，Component 类在很大程度上依赖于 java. awt. peer 类。这是因为 UNIX 机器上创建部件的系统命令与 Windows 机器上的不同。同位体 (peer) 可以起到中介作用，将基于 Java 的代码翻译为与平台相关的命令，这样可不必直接与这些方法交互对话。

方法

※ boolean action (Event evt, Object what)

当针对部件 (Component) 的动作发生时，此方法被 handleEvent () 方法调用，如果要求对成分或容器所发生的事件能够得到更多的控制，则不必使用这个方法。如果事件得到成功的控制，此方法返回一个真值。

※ void addNotify ()

告知当前部件，产生一个同位体。

※ Rectangle bounds ()

返回当前部件的界限值。

※ int checkImage (Image image, ImageObserver observer)

返回正在屏幕上显示的图象状态。这个值要么从当前部件同位体得到，要么从当前 Toolkit 获得。对指定的 ImageObserver 将

被告知图象的状态。

※ int checkImage (Image image, int width, int height, ImageObserver observer)

返回正在屏幕上显示的被定标的图象状态。这个值要么从当前部件同位体得到，要么从当前 Toolkit 获得。对指定的 ImageObserver 将被告知图象的状态。

※ Image createImage (int width, int height)

根据指定的高度和宽度建立一个图象。

※ Image createImage (ImageProducer producer)

根据给定的 ImageProducer 所提供的信息建立一个图象。

※ void deliverEvent (Event e)

把一个事件传递给当前部件。

※ synchronized void disable ()

使当前部件无效，该部件将不能响应用户的事件。

※ void enable (boolean cond)

如果条件 (cond) 为真，此方法起动当前部件，否则将使其无效。

※ syndhronized void enable ()

使能当前部件。

※ Color getBackground ()

返回当前背景颜色。

※ synchronized ColorModel getColorModel ()

在创建部件时返回所用的 ColorModel (颜色模型)。

※ Font getFont ()

返回部件字体 (font)，如果该部件无字体，这个方法将返回父容器的字体。

※ FontMetrics getFontMetrics (Font font)

返回当前部件的字体显示，如果当前部件是不可用的，那么返回一个空值。

※ Color getForeground ()

返回当前部件的前景颜色。此方法的缺省值将是返回父容器的前景颜色。

※ Graphics getGraphics ()

返回在创建该部件时所使用的图形的上下文。对于1.0.2JDK版本，这个方法只是针对Canvas部件来执行。

※ Container getParent ()

返回当前部件所驻留的父容器。

※ ComponentPeer getPeer ()

返回当前部件的同位体。

※ Toolkit getToolkit ()

返回当前部件的Toolkit（软件工具箱）。

※ boolean gotFocus (Event evt, Object what)

无论何时，只要当前部件接收到输入的焦点（通常是鼠标移动的结果），这个方法即被调用。参数what表示事件的自变量(what相当于evt.arg)。

※ boolean handleEvent (Event evt)

每当给当前部件传递一个事件时，便调用这个方法。缺省情况下，这个方法将调用一系列较为专用的方法，诸如mouseDown()和keyUp()等。

如果事件已得到成功的处理，那么这个方法将返回一个真值。如果用户忽略了这个方法，但仍希望使用一些缺省方法的处理特性，那么请调用super.handleEvent (evt)。

※ synchronized boolean inside (int x, int y)

判断指定的坐标是否在当前部件内。

※ synchronized void hide ()

屏蔽该部件将其隐藏起来，从而不再显现。

※ boolean imageUpdate (Image img, int flags, int x, int y, int width, int height)

当要修改图象时，重新绘制所指定的图象。flags是常数，诸如在java.awt.image.ImageObserver接口中出现的FRAMEBI-

TS 和 ALLBITS。

当有较多信息用来处理图象时，系统将会调用这个方法。所以若要想给此方法添加某些特性时，只能调用这个方法。

※ void invalidate ()

如果需要对一个部件进行布局设计时，那么将此部件置为无效。这个方法使当前部件无效，迫使它再次进行布局设计。

※ boolean isEnable ()

启动一个部件与用户交互。如果该部件能够启动并与用户交互，则返回一个真值，否则返回一个假值。

※ boolean isShowing ()

如果部件对用户来说是可见的，则返回一个真值。这就是说，需要可见的容器中的部件也是可见的。

※ boolean isValid ()

如果一个部件需要布局设计，则它被置为无效。如果这个部件被置为无效，返回一个假值；若该部件是有效的，则返回一个真值。

※ boolen isVisible ()

如果当前部件是可见的，返回真值。

※ boolean keyDown (Event evt, int key)

当按下一个键时，handleEvent () 调用这个方法。参数 evt 是被指定的一个完整的事件，参数 key 是被按下的键。

※ boolean keyUP (Event evt, int key)

当释放一个键时，handleEvent () 调用这个方法。参数 evt 是被指定的一个完整的事件，参数 key 是被释放的键。

※ void layout ()

在布局设计中正确的摆放当前部件。

※ void list ()

在标准输出流中打印出当前参数。

※ void list (PrintSream out)

在指定输出流中打印当前参数。

※ void list (PrintStream out, int indent)

把当前参数输出到指定的 PrintScream (输出流)。

※ Component locate (int x, int y)

返回指定位置的部件。

※ point location ()

返回当前部件在容器中的 x 和 y 坐标点。

※ boolean lostFocus (Event evt, Object what)

当前部件释放输入焦点时由 handleEvent () 调用该方法。

※ boolean mouseDown (Event evt, int x, int y)

在当前部件中按下鼠标时由 handleEvent () 调用该方法。

※ boolean mouseDrag (Event evt, int x, int y)

在当前部件中拖动鼠标时由 handleEvent () 调用该方法 (当鼠标键按下并移动鼠标时，就认为是拖动)。

※ boolean mouseEnter (Event evt, int x, int y)

当鼠标进入当前部件时由 handleEvent () 调用该方法。

※ boolean mouseExit (Event evt, int x, int y)

当鼠标离开当前部件时由 handleEvent () 调用该方法。

※ boolean mouseMove (Event evt, int x, int y)

当前部件释放输入焦点时由 handleEvent () 调用该方法。

※ boolean mouseUp (Event evt, int x, int y)

在当前部件释放输入焦点时由 handleEvent () 调用该方法 (当鼠标按键抬起并移动鼠标时，叫做移动)。

※ Dimension minimumSize ()

返回显示当前部件所需的最小尺寸。

※ void move (int x, int y)

把当前部件移到新的位置，从而使其成为无效的。

※ void nextFocus ()

把焦点向前移到下一个部件，这个方法在用户程序中从一个部件移动到另一个部件时，是很有用的。

※ void paint (Graphics g)

绘制当前部件。

※ void paintAll (Graphics g)

绘制当前部件以及它的所有子部件。

※ protected String paramString ()

返回包含当前部件有关信息的字符串。其中有部件所在的位置、是否是可见的、是否是有效的以及是否是已被启动等等。

※ boolean postEvent (Event e)

传递指定的事件给当前部件，将导致调用 handleEvent ()。如果该部件未能成功地处理此事件，那么该事件将被传送给部件的父容器。

※ Dimension preferredSize ()

返回这个部件所选择的尺寸。这个值由此部件的同位体提供。

※ boolean prepareImage(Image image,int width,int height,ImageObserver observer)

这个方法“准备”一个要显示的已被标度的图象，其过程包括装填这个图象和准备显示的屏幕。在指定的 ImageObserver 中的 imageUpdate () 方法将被告知这个过程的状态。

※ boolean prepareImage (Image image, ImageObserver observer)

这个方法“准备”要显示一个图象，其过程包括装填这个图象和准备显示的屏幕。通过 ImageUpdate () 方法，将把有关此过程的信息告知图象观察器 (ImageObserver)。

※ void print (Graphics g)

打印此部件。缺省情况下，这个方法简单地调用 paint () 方法。

※ void printAll (Graphics g)

打印此部件和它的子部件。缺省情况下，这个方法打印该部件和它的子部件。

※ synchronized void removeNotify ()

清除当前部件的同位体。

※ Void repaint (int x, int y, int width, int height)

重新绘制该部件的一部分。

※ void repaint (long delay, int x, int y, int width, int heigth)

在等待所指定数目的微秒后，重新绘制该部件的一部分。

※ void repaint (long time)

在等到所指定数目的微秒后重新绘制整个部件。

※ void repaint ()

重新绘制整个部件。

※ void requestFocus ()

请求输入焦点。

※ synchronized void reshape (int x, int y, int width, int height)

移动此部件到所指定的坐标并且将其重新修整到所指定的尺寸。这将使该部件成为无效。

※ void resize (Dimension d)

根据所指定的维数——宽度和高度，重新设定部件的尺寸。这将使该部件成为无效。

※ void resize (int width, int height)

根据所指定的宽度和高度，重新设定部件的尺寸。这将使该部件成为无效。

※ synchronized void setBackground (Color c)

设置该部件的背景颜色。

※ synchronized void setFont (Font f)

设置该部件的字体。

※ synchronized void setForeground (Color c)

设置该部件的前景颜色。

※ void show (boolean cond)

如果 cond 参数为真，那么显示该部件；如果 cond 参数为假，则此部件被隐藏。

※ synchronized void show ()

显示此部件。

※ Dimension size ()

返回该部件的尺寸。

※ String toString ()

返回包含当前部件参数的一个字符串。其中有部件名称以及由 paramString () 所返回的一些值。

※ void update (Graphics g)

这个方法是因调用 repaint () 而引起的调用。此方法可用来完成与修改部件有关的一些管理任务。

※ void validate ()

使此部件成为有效，这意味着不必对其进行布局设计。

Abstract Container Extends Component

Container（容器）类是一种基本类，可容纳其他许多部件。Container 类是诸如 Frame、Panel 和 FileDialog 等其他容器型类的基础。须注意的是，不能显现地创建一个 Container，而是须使用一个导出的类或创建自己的 Container 类。

方法

※ synchronized Component add (String info, Component comp)

给容器增加一个部件。对字符串中所提供的信息的应用同 Container 的 LayoutManager 有关。

※ synchronized Component add (Component comp, int pos)

在指定的位置给容器添加一个部件。

※ Component add (Component comp)

把一个部件添加到该容器的尾部。

※ synchronized void addNotify ()

创建该容器的同位体。

※ int countComponents ()

返回容器中部件数。

※ void deliverEvent (Event evt)

把所指定的事件传递给相应的部件。如果这个事件没有在当前容器中的一个部件内发生，那么该事件将被移交给 java.awt.Component.postEvent（）方法。

※ synchronized Component getComponent(int num) throws ArrayIndexOutBoundsException

返回指定索引处的部件。如果容器中的部件数少于 num，那么将启动 ArrayIndexOutBoundsException。

※ synchronized Component [] getComponents ()

返回一个阵列，包含驻留在容器中的部件。

※ LayoutManager getLayout ()

返回在创建容器时所使用的布局管理器。

※ Synchronized void layaut ()

根据布局管理器的约定，构造容器的部件。

※ void list (PrintStream out, int indent)

显示容器中的部件及其标准的部件信息（名称和参数）。在指定标识处开始打印。

※ Component locate (int x, int y)

返回位于指定坐标处的部件。

※ synchronized Dimension minimumSize ()

返回这个部件所需的最小尺寸。这个值与每个单一的部件的最小尺寸有关。

※ void paintComponents (Graphics g)

绘制容器中的部件。这个方法等同于这个类中的 printComponents（）方法。

※ protected String paramString ()

返回由 java.awt.Component.paramString（）方法所提供的除部件布局管理器外的信息。

※ synchronized Dimension preferredSize ()

返回所选择的容器的尺寸。这个值是基于对每个部件的 preferredSize（）方法返回的值。

※ void printComponents (Graphics g)

显示容器中的部件。这个方法等同于同类中 printComponent () 方法。

※ synchronized void remove (Component comp)

从容器中撤销指定的部件。

※ synchronized void removeNotify ()

撤销容器的同位体。

※ void setLayout (LayoutManager mgr)

为该部件设置布局管理器。

※ synchronized void validate ()

有效的部件不必进行布局设计。这个方法使容器和它的所有部件有效。

Dialog Extends Window

对话框 (Dialog) 是一个用来接收用户输入的窗口。一个对话框必须存在于一个 Frame 对象中。模态对话框是一种捕捉所有用户输入的对话框。

方法

※ Dialog (Frame parent, String title, boolean modal)

在一个指定的 Frame 中创建一个对话框，并有一个指定的标题。如果 modal 参数为真，则创建的是一个模态式对话框。

※ Dialog (Frame parent, boolean modal)

在一个指定的 Frame 中创建一个无标题的对话框。如果 modal 参数为真，则创建的是一个模态式对话框。

※ synchronized void addNotify ()

创建一个对话框的同位体。

※ boolean isModal ()

如果对话框是模态的，则返回真值。

※ boolean isResizable ()

如果对话框是可缩放的，则返回真值。

※ protected String paramString ()

返回一个描述了对话框参数的字符串，包括标题以及是否是模态对话框。

※ void setTitle (String title)

为对话框设置标题。

Dimension

Dimension 类提供用户一个简单的方法来控制图形工具的大小。通常，Dimension 类提供一个简单的方法来返回两个值：宽和高。

字段

- int height
- int width

方法

※ Dimension ()

创建一个新的 Dimension 对象。

※ Dimension (Dimension d)

创建一个新的 Dimension 对象，它与一个已经被创建的 Dimension 对象具有相同的值。

※ Dimension (int width, int height)

创建一个新的有指定的宽和高的 Dimension 对象。

※ String toString ()

以一个字符串的形式返回 Dimension 的宽和高。

Event

Event 类被用做为响应用户交互作用的方法。它在一个单一的对象中传送有关用户事件的大量信息。通过它的常量和实例变量，每个 Event 对象在处理事件时可以提供给用户一些具体细节。

所有 final 字段被用做常量。通过它们可以区分所接收到的 Event 对象中包含的信息。通常，这些字段被分为两类：一类代表某些键(如 F1 和 HOME)，一类代表某些事件(如 GOT _ FOCUS 和 KEY _ PRESS)

▶参见程序 3 “CardLayout Example Source Code”

▶参见程序 7 “Menu Example”

字段

• final static int ACTION _ EVENT

它是各组件产生事件的标志。一个 ACTION _ EVENT 将产生一个对 action （） 的调用。

• final static int ALT _ MASK

如果 ALT 键被按下，则它将被放入 modifiers 字段中。且通过使用 AND 操作符而被检测。

• Object arg

它是一个依赖于事件类型的可选变量。大多数部件置 arg 等于它们的标题。

• int clickCount

跟踪鼠标连续按下的次数。

• final static int CTRL _ MASK

如果 CTRL 键被按下，则它将被放入 modifiers 字段中。且通过使用 AND 操作符而被检测。

• final static int DOWN

• Event evt

作为另一个 Event 对象的参考。这样可以用来建立一个 Event 对象的连接表。

• final static int F1

key 字段的一个可能的值

• final static int F2

• final static int F3

• final static int F4

• final static int F5

• final static int F6

• final static int F7

• final static int F8

• final static int F9

• final static int F10

• final static int F11

• final static int F12

• final static int GOT _ FOCUS

当 Component 获得输入焦点时产生该事件。它调用 gotFocus ()。

• final static int HOME

key 字段的可能的值。

• int id

事件的标识。它可以区别于像 ACTION _ EVENT 或者 MOUSE _ DOWN 这样的静态常量。

• int key

包含了被按下的键。

• final static int KEY _ ACTION

当用户按下一个功能键时产生这个事件。它将调用 keyDown ()。

• final static int KEY _ ACTION _ RELEASE

当用户释放一个功能键时产生这个事件。它将调用 keyUp ()。

• final static int KEY _ PRESS

当用户按下一个标准的 ASCII 键时产生这个事件。它将调用 keyDown ()。

• final static int KEY _ RELEASE

当用户释放一个标准的 ASCII 键时产生这个事件。它将调用 keyUp ()。

• final static LEFT

key 字段的一个可能的值。它代表了左箭头键。

• final static int LIST _ DESELECT

当用户取消一个 List 中被选中的项时调用它。这个 Event 将包含一个 Integer 作为它的 arg 字段，代表被取消的项。

• final static int LIST _ SELECT

当用户选中 List 中的一项时调用它。这个 Event 将包含一个 Integer 作为它的 arg 字段，代表被选中的项。

• final static int LOAD _ FILE

当一个文件被装入时产生这个事件。

• final static int LOST _ FOCUS

当 Component 释放输入焦点时产生这个事件，它向 lostFocus（）发出请求。

• final static int META _ MASK

如果按下 META 键，那么它被放入 modifiers 字段中，且在使用 AND 操作符时对其进行测试。

• int modifiers

代表了修饰键的状态：Alt，Ctrl，Meta 和 Shift。它能通过使用 AND 操作符与在类中定义的静态常量比较。

• final static int MOUSE _ DOWN

当按下鼠标时产生这个事件。它调用 mouseDown（）。

• final static int MOUSE _ DRAG

• final static int MOUSE _ ENTER

• final static int MOUSE _ EXIT

• final static int MOUSE _ MOVE

• final static int MOUSE _ UP

• final static int PGDN

• final static int PGUP

• final static int RIGHT

• final static int SAVE _ FILE

• final static int SCROLL _ ABSOLUTE

当用户通过在滚动条内移动滚动条标记到一个具体的位置时产生这个事件。

• final static int SCROLL _ LINE _ DOWN

当用户在滚动条上单击下箭头时产生这个事件。

• final static int SCROLL _ LINE _ UP

• final static int SCROLL _ PAGE _ DOWN

• final static int SCROLL _ PAGE _ UP

• final static int SHIFT _ MASK

如果 shift 键被按下或者 Caps Lock 打开时，则放入 modifiers 字段中，它将通过使用 AND 操作符进行检测。

• Object target

事件所指向的对象。

• final static int UP

• long when

事件发生的时间。

• fianl static int WINDOW _ DEICONIFY

id 字段一个可能的值。

• final static int WINDOW _ DESTROY

id 字段一个可能的值，当用户试图关闭 Windows 时使用。

• final static int WINDOW _ EXPOSE

id 字段一个可能的值。

• final static int WINDOW _ ICONIFY

id 字段一个可能的值。

• final static int WINDOW _ MOVED

id 字段一个可能的值。

• int x

鼠标单击时的横坐标。

• int y

鼠标单击时的纵坐标。

方法

※ Event (Object target, int id, Object arg)

根据指定的目标、标识和 arg 值创建一个新的事件。

※ Event (Object target, long when, int id, int x, int y, int key, int modifiers)

根据指定的目标、发生时间、识别号、横坐标、纵坐标、key值和 modifiers 值创建一个新的事件。

※ Event (Object target, long when, int id, int x, int y, int key, int modifiers, Object arg)

用指定的目标、发生时间、识别号、横坐标、纵坐标、key 值、modifiers 值和 arg 值创建一个新的事件。

※ boolean controlDown ()

如果 Ctrl 键被按下则返回 true。

※ boolean metaDown ()

如果 Ctrl 键被按下则返回 meta。

※ protected String paramString ()

返回包含所有事件实例变量值的一个字符串。

※ boolean shiftDown ()

如果 Shift 键被按下则返回 true.

※ String toString ()

返回事件对象的名字以及从 paramString () 方法中返回的信息。

※ void translate (int x, int y)

FileDialog Extends Dialog

一个 FileDialog 创建一个窗口以提供用户选择文件的功能。FileDialog 本身集成了客户机上的文件对话框的表面特征。

图 7 所示的 FileDialog 的源代码见程序 4。这些代码虽然很简单以至于它不能作为一个完整的 applet，但它仍然表明了一种创建 FileDialog 的方法。

程序 4　FileDialog Example

```
import java.applet.Applet;
import java.awt.*;

public class FDialogExample extends Applet{
```

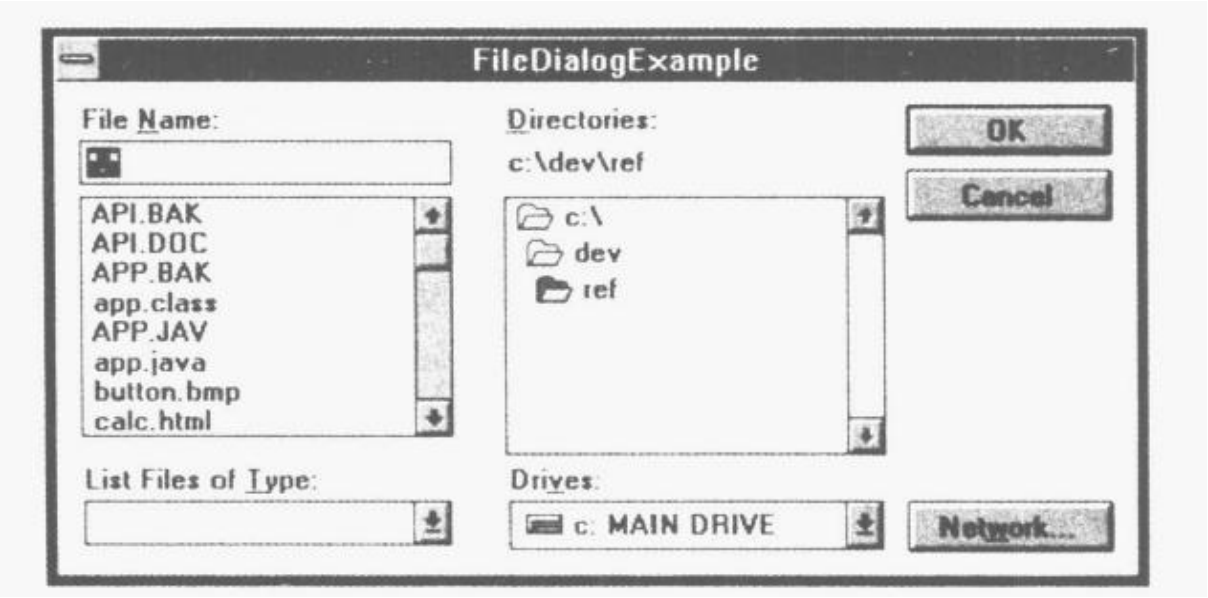


图 7　Windows NT 中的一个文件对话框

```
  public void init(){
    Frame f;
    f=new Frame();

    Dialog d;
    d=new File Dialog(f,"FileDialogExample");

    f.show();
    d.show();
  }
}
```

字段

• final staic LOAD

这是一个定义文件对话框模式的常量。如果模式为 LOAD，则文件对话框是面向文件装入而设定的。

• final staic SAVE

这是一个定义文件对话框模式的常量。如果模式为 SAVE，则文件对话框是面向保存文件而设定的。

方法

※ FileDialog (Frame parent, String title, int mode)

创建一个有指定模式的文件对话框。

※ FileDialog (Frame parent, String title)

创建一个文件对话框，设置模式为 LOAD。

※ synchronized void addNotify ()

创建一个文件对话框的同位体。

※ String getDirectory ()

返回当前目录。

※ String getFile ()

返回被用户选中的文件名称。

※ FilenameFilter getFilenameFieter ()

返回用户选择文件时所使用的文件名称过滤器 FilenameFilter。

※ int getMode ()

返回文件对话框的模式是 LOAD 还是 SAVE。

※ protected String paramString ()

返回包括目录和模式在内的部件的属性。

※ void setDirectory (String dir)

设置文件对话框的目录。

※ setFile (String file)

在文件对话框中高亮指定文件。

※ public void setFilenameFilter (FilenameFilter filter)

用户在选择文件时，为要使用的文件对话框设置一个过滤器。

FlowLayout Implements LayoutManager

FlowLayout 类是一个相对简单的布局管理器。它按行放置所有部件并且在当前行满时另起一行。

字段

• final static int CENTER

部件的对齐方式是中央对齐。

• final static int LEFT

部件的对齐方式是左对齐。

方法

※ FlowLayout ()

以中央对齐方式创建一个新的 FlowLayout。

※ FlowLayout (int align)

以指定的对齐方式创建一个新的 FlowLayout。

※ FlowLayout (int align, int hgap, int vgap)

以指定的对齐方式创建一个新的 FlowLayout，横向间隙 hgap 和纵向间隙 vgap 确定了围绕部件的空间大小。

※ void addLayoutComponent (String name, Component comp)

在类提供一个布局管理器 LayoutManager 接口时需要这个方法，但是它不做任何事情。

※ void layoutContainer (Container target)

根据 FlowLayout 参数安排具体的容器。

※ Dimension minimumLayoutSize (Container target)

返回在指定的容器中部件所需的最小空间。这个值等于部件所需的最小空间加上容器所占的空间大小。

※ Dimension preferredLayoutSize (Container target)

返回在指定的容器中部件所需的最佳空间。这个值等于部件所需的最佳空间加上容器所占的空间大小。

※ void removeLayoutComponent (Componenet comp)

从布局管理器中移走一个指定的部件。

※ String toString ()

返回一个包括了管理器名称、对齐方式、横向间隙和纵向间隙的字符串。

Font

Font 类是作为客户机和你的源代码之间的中介而提供服务的，它为你提供文本的具体风格。由于它不影响输出流上的输出的表面特征（如：System.out），当前字体将决定在显示一个组件

——特别是 Applet 时所生成的任何文本的表面特征。

▶参见程序 2 “Sample Applet”

注意：如果要改变显示的字体，你必须首先创建一个新的 Font 类并设置当前字体。这些可以通过 java. awt. Component. setFont （） 或者 java. awt. Graphics. setFont （） 来完成。

字段

- final static int BOLD

  style 字段的一个可能的值，指明字体是粗体。

- final static int ITALIC

  style 字段的一个可能的值，指明字体是斜体。

- protected String name

  字体名称。

- final static int PLAIN

  style 字段的一个可能的值，指明字体是标准体。

- protected int size

  字体大小。

- protected int style

  字体类型。

方法

※ Font （String name，int style，int size）

创建一个有指定名称、类型和大小的新字体。

※ boolean equals （Object anotherobject）

比较两种字体是否等同。如果第二个对象不是一个 Font 类，则返回 flase。

※ String getFamily （）

返回字体所属系列的名称。这个值取决于系统本身。

※ static Font getFont （String name）

返回系统支持的字体中的某字体。

※ static Font getFont （String name，Font defaultfont）

返回系统支持的字体中的某字体。如果字体不存在，则返回

缺省字体 defaultfont。

※ String getName ()

返回字体名称。

※ int getSize ()

返回字体大小。

※ int getStyle ()

返回字体类型。

※ int hashCode ()

返回字体的 hashcode，这个值依赖于字体名称、类型和大小。

※ boolean isBold ()

如果字体是粗体则返回 true。

※ boolean isItalic ()

如果字体是斜体则返回 true。

※ boolean isPlain ()

如果字体是标准体则返回 true。

※ String toString ()

返回字体的名称、类型和大小等信息的字符串。

FontMetrics

FontMetrics 类提供了给定字体的信息。这个类对于像增强文本的美观性这样的工作是很有用的，因为我们经常需要确切地知道每个字符所在位置。为某字体创建一个 FontMetrics 并且使用存取方法来获得信息，这样可帮助用户决定字符和字符串的确切大小。

▶参见“Font”

注意：若要在你的程序中使用 FontMetrics，你应先定义一个 FontMetrics 句柄，然后使用 java.awt.getFontMetrics () 方法提供 FontMetrics 信息。

字段

• protected Font font

FontMetrics 所基于的字体。

方法

※ protected FontMetrics (Font font)

创建一个新的 FontMetrics。

※ int bytesWidth (byte data [], int offset, int len)

返回一组字节的宽度。offset 参数指定了度量开始时的变址，len 参数指定了需要度量的项数。

※ int charsWidth (char data [], int offset, int len)

返回一组字符的宽度。offset 参数指定了度量开始时的变址，len 参数指定了需要度量的项数。

※ int charWidth (char ch)

返回单一字符的宽度。

※ int charWidth (int ch)

返回单一字符的宽度。

※ int getAscent ()

返回字体中 x-height(小写字母主体的高度)和字符顶端之间的空间大小。

※ int getDescent ()

返回字体中基线（某类字体所位于的一条虚构的线）和字符底端的空间大小。

※ Font getFont ()

返回字体。

※ int getHeight ()

返回字体中字符顶端到底端的空间大小。

※ int getMaxAdvance ()

推进值被定义为两个字符间的空间大小（实际上是第一个字符底端到第二个字符顶端之间的空间大小）。这个方法返回某字体中任意字符的最大的预留值。

※ int getMaxAscent ()

升部是 x-height 到字母顶端的距离。这个方法返回某字体中任意字符的最大的升部。

※ int getMaxDescent ()

降部是基线到字母底端的距离。这个方法返回某字体中任意字符的最大的降部

※ int [] getWidhths ()

返回包含了字体前 256 个字符宽度的数组。

※ int stringWidth (String str)

返回代表指定字符串宽度的一个整数。

※ String toString ()

返回包含了类名、字体名、升部、降部和高度这些有关类的信息的字符串。

Frame Extends Window Implements MenuContainer

Frame 是一个加入了一些功能的窗口。对于其表面特征，一个 Frame 有边界和标题。对于其功能，你能够在 Frame 中加入一个菜单并且指定 Frame 最小化时所要显示的图象。

▶参见程序“MenuBar Extends MenuComponent Implements MenuContainer”

▶参见程序“Window Extends Container”

注意：当使用一个 Frame 时，记住一定要使用 show () 方法，否则你的 Frame 是不可见的。

字段

- final static int CROSSHAIR _ CURSOR
- final static int DEFAULT _ CURSOR
- final static int E _ RESIZE _ CURSOR
- final static int HAND _ CURSOR
- final static int MOVE _ CURSOR
- final static int NE _ RESIZE _ CURSOR
- final static int NW _ RESIZE _ CURSOR
- final static int N _ RESIZE _ CURSOR
- final static int SE _ RESIZE _ CURSOR
- final static int SW _ RESIZE _ CURSOR

- final static int S _ RESIZE _ CURSOR
- final static int TEXT _ CURSOR
- final static int WAIT _ CURSOR
- final static int W _ RESIZE _ CURSOR

方法

※ Frame ()

创建一个新的无标题的 Frame。

※ Frame (String title)

创建一个新的有指定标题的 Frame。

※synchronized void addNotify ()

创建一个 Frame 的同位体。

※ synchronized void dispose ()

分配 Frame 所使用的资源。

※ int getCursorType ()

返回 Frame 中光标的类型。

※ Image getIconImage ()

返回 Frame 的图标 icon，icon 是一个不可访问的变量，它定义了 Frame 最小化时应显示的图象。

※ MenuBar getMenuBar ()

返回某 Frame 的 MenuBar（菜单状态条）。

※ String getTitle ()

返回 Frame 的标题。

※ boolean isResizable ()

如果 Frame 是可缩放的，则返回 true。

※ protected String paramString ()

返回一个字符串，它包括了 Frame 的标题，如果 Frame 是可缩放的，还包括字缩放能力的信息。

※ synchronized void remove (MenuComponent m)

从 Frame 中移走指定的 MenuBar。

※ void setCursor (int cursorType)

设置 Frame 的光标。这个值可以是上述常量表中的任何一个。

※ void setIconImage (Image image)

设置 Frame 的图标。这个图标在 Frame 最小化时显示。

※ synchronized void setMenuBar (MenuBar mb)

设置 Frame 的 MenuBar。

※ void setResizable (boolean resizable)

设置 Frame 是可缩放的 (true) 还是不可缩放的 (false)。所有的 Frame 初始化为可缩放的。

※ void setTitle (String title)

设置 Frame 的标题。

Abstract Graphics

Graphics 类提供了从绘图工具 (drawline ()) 到编辑工具 (clipRect ()) 这样大量的图形工具。

如果你要生成一个应用，通过浏览器环境可以产生一个图形对象。因此，你可以使用 Graphics 对象向 paint () 和 update () 方法传递参数而不用担心 drawline () 是如何工作的。

▶参见程序 2 "Sample Applet"

Graphics 类为用户提供大量的图形工具去生成非常好的接口 (参看图 8)。程序 5 给出了一个使用这些工具的简明的例子并说明了创建 Graphics 的方式。

程序 5　Graphics Example

```
import java.applet.Applet;
import java.awt.*;

public class GraphicsExample extends Applet
{

  public void paint(Graphics g){
```

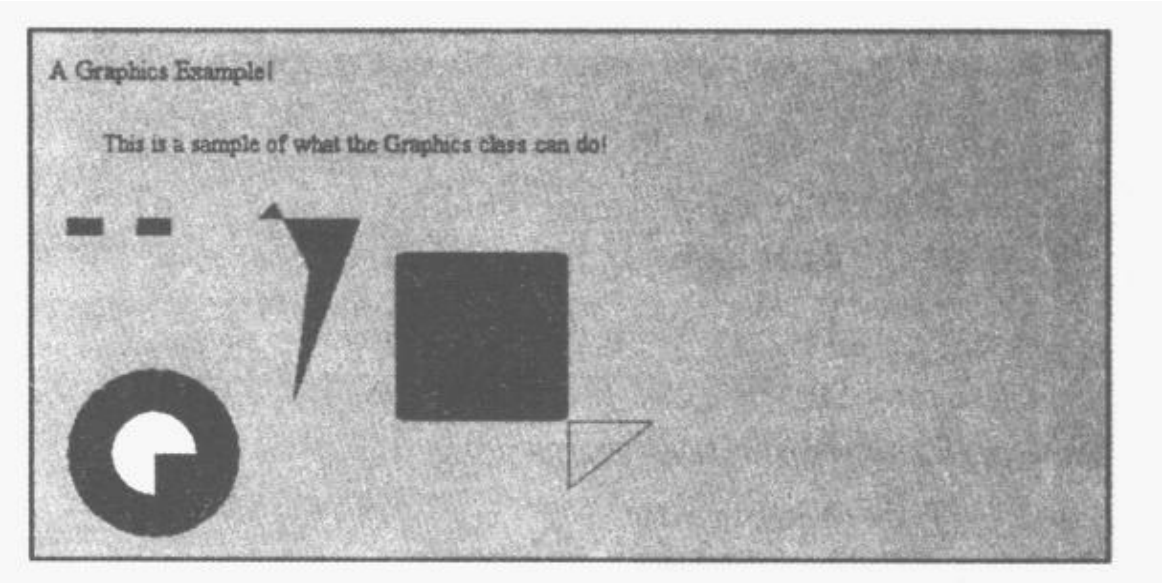


图 8　图形类样例

```
g.setFont(new Font("TimesRoman",Font.PLAIN,14));
                    // sets the font for the drawString() statement
g.drawString("This is a sample of what the Graphics class can
do!"30,20);

g.setColor(Color.red);

g.fill3DRect(10,60,20,10,true);
g.fill3DRect(50,60,20,10,false);
int xpoints[]={120,130,150,140,180};
                          // x-coordinate for polygon
int ypoints[]={60,50,90,170,60};
g.fillPolygon(xpoints,ypoints,5);

g.fillRound Rect(200,80,100,100,10,5);
g.setColor(Color.blue);
g.fillArc(10,150,100,100,0,360);
```

```
    g.setColor(Color.white);
    g.fillArc(35,175,50,50,0,270);

    g.setColor(Color.magenta);
    g.drawLine(300,180,350,180);
    g.drawLine(300,180,300,220);
    g.drawLine(300,220,350,180);
    }
}
```

方法

※ protected Graphics ()

创建一个新的 Graphics 对象。

※ abstract void clearRect (int x, int y, int width, int height)

清除带有背景颜色的矩形框。

※ abstract void clipRect (int x, int y, int width, int height)

为图形操作设置一个剪贴框。只有在指定区域的图形才能显示出来。

注意：在设置剪贴区时要小心。一旦设置了剪贴区，它将不能被恢复。所以，我们建议通过 create () 方法创建一个临时的 Graphics 对象。这样就可以建立剪贴区并且用这个新对象进行绘画操作。当你想要再整屏显示时，你可以切换到原始对象去。

※ abstract void copyArea (int x, int y, int width, int height, int dx, int dy)

拷贝由横坐标、纵坐标以及宽和高定义的一块区域。这块区域是水平方向上 dx 个点，垂直方向上 dy 个点。

※ Graphics create (int x, int y, int width, int height)

创建一个新的 Grahics 对象。这个对象除了以指定的坐标和大小所确定的剪贴区外，它和当前的 Graphics 对象是等同的。这样做是很有用的，因为一旦建立了剪贴区，它将不能被恢复。所以，通过建立一个新的 Graphics 对象，你可以不破坏原始 Graph-

ics 对象而使用剪贴区。

※ abstract Graphics create ()

建立一个新的与当前目标等同的 Graphics 对象。由于原始对象和新建立的对象都对同一区域进行绘画操作，这对于像设置剪贴区这样的进行临时不可恢复的修改操作是很有用的。

※ abstract void dispose ()

分配图形内容所使用的资源。

※ void draw3DRect (int x, int y, int width, int height, boolean raised)

在指定的位置以指定的大小画一个三维矩形框。将参数 raised 的值设置为 true，就会给矩形框一个高亮度的颜色，这样就可以以三维的形式显示出来。

※ abstract void drawArc (int x, int y, int width, int height, int startAngle, int arcAngle)

以指定的坐标和大小画一个弧。参数 startAngle 和 arcAngle 代表了弧开始时的角度和弧结束时的角度（从 0 到 360 度的弧是一个圆）。

※ void drawBytes (byte data [], int offset, int lenght, int x, int y)

从数组 data [] 中显示各字节。参数 offset 代表了第一个要显示的字节的索引值，length 代表了要显示的字节数。x，y 代表了在屏幕上显示字节的坐标。

※ void drawChars (char data [], int offset, int lenght, int x, int y)

从数组 data [] 中显示各字符。参数 offset 代表了第一个要显示的字符的索引值，length 代表了要显示的字符数。x，y 代表了在屏幕上显示字符的坐标。

※ abstract boolean drawImage (Image img, int x, int y, int width, int height, Color bgcolor, ImageObserver observer)

在指定的位置以指定的大小画一幅图象。参数 bgcolor 是图

象背后的颜色，如果图象中有透明的点，这个颜色就会显示出来。observer 就是监视图象装入过程的 ImageObserver。

注意：java.awtComponent 类提供了 ImageObserver 接口。因此，在大多数应用中你可以使用 this 关键字作为 ImageObserver。

※ abstract boolean drawImage (Image img, int x, int y, Color bgcolor, ImageObserver observer)

在指定的位置以正常大小画一幅图象，参数 bgcolor 是画在 applet 图象背后的颜色。如果图象中有透明的点，这个颜色就会显示出来。observer 就是监视图象装入过程的 ImageObserver。

※ abstract boolean drawImage (Image img, int x, int y, int width, int height, ImageObserver observer)

在指定的位置以指定的大小画一幅图象。observer 就是监视图象装入过程的 ImageObserver。

※ abstract boolean drawImage(Image img,int x,int y,ImageObserver observer)

在指定的位置以正常大小画一幅图象。observer 就是监视图象装入过程的 ImageObserver。

※ abstract void drawLine (int x1, int y1, int x2, int y2)

在两点间画一条直线。

※ abstract void drawOval (int x, int y, int width, int height)

在指定的位置以指定的大小画椭圆。

※ void drawRect (int x, int y, int width, int height)

在指定的位置以指定的大小画矩形框。

※ abstract void drawRoundRect (int x, int y, int width, int height, int arcWidth, int arcHeight)

在指定位置以指定的大小画矩形框。参数 arcWidth 和 arcHeight 决定了矩形框拐角的形状和大小。

※ void drawPolygon (Polygon p)

画一个由指定的 java.awt.polygon 对象所定义的多边形。

※ abstract void drawPolygon (int xPoints [], int yPoints [], int

npoints)

画一个由一组指定的点所决定的多边形。参数 nPoints 代表了多边形锐角的个数。Java 通过数组的前 n 对坐标生成符合锐角个数的多边形。

※ abstract void drawString (String str, int x, int y)

在指定的位置用当前的字体写字符串。

※ void fill3DRect (int x, int y, int width, int height, boolean raised)

填充在指定位置有指定大小的矩形框。参数 raised 将改变矩形框的明暗度，如果它的值为 true，则矩形框以三维的形式出现。

※ abstract void fillOval (int x, int y, int width, int height)

填充由 x 横坐标和 y 纵坐标定义的有指定大小的椭圆。

※ abstract void fillRect (int x, int y, int width, int height)

在给定的坐标点填充指定大小的矩形框。

※ abstract void fillRoundRect (int x, int y, int width, int height, int arcWidth, int arcHeight)

在给定的坐标点填充指定大小的矩形框。参数 arcWidth 和 arcHeight 决定了矩形框拐角的形状和大小。

※ abstract void fillArc (int x, int y, int width, int height, int atartAngle, int arcAngle)

在给定的坐标点填充指定大小的弧。弧的大小由参数 startAngle 和 arcAngle 决定，单位是度。

※ void fillPolygon (Polygon p)

填充由具体的 Polygon 目标所定义的多边形。

※ abstract void fillPolygon (int cPoints [], int yPoints [], int nPoints)

填充由一组指定的点所定义的多边形。参数 nPoints 代表了多边形锐角的个数。Java 通过数组的前 n 对坐标生成符合锐角个数的多边形。

※ void finalize ()

为指定的 Graphics 对象分配资源。

※ abstract Rectangle getClipRect ()

返回由 clipRect () 方法所设置的剪贴区。

※ abstract Color getColor ()

返回当前图形操作所使用的颜色。

※ abstract Font getFont ()

返回有关文本操作所使用的字体。

※ abstract FontMetrics getFontMetrics (Font f)

返回指定 Font 的 FontMetrics。

※ FontMetrics getFontMetrics ()

返回当前字体的 FontMetrics。

※ abstract void setColor (Color c)

设置所有图形操作使用的颜色。

※ absract void setFont (Font foat)

设置所有图形操作的字体。

※ abstract void setPaintMode ()

用随后的图形命令覆盖当前颜色所处的位置。

※ abstract void setXORMode (Color otherColor)

这个方法使 Graphics 对象对当前颜色和指定的颜色 otherColor 进行异或。当画图形时，当前颜色的所有图形均用 otherColor 去画，反之亦然。不具有二者中任何一种颜色的图形将以不同的、随机的颜色显示。

※ abstract void translate (int x, int y)

将指定的坐标作为图形操作的新的起始点。这样，所有的操作均与这个起始点相关。

※ String toString ()

以一个字符串的形式返回包括当前颜色和字体在内的 Graphics 对象的信息。

GridBagConstraints Implements java. lang. Cloneable

GridBagConstraints 类可以使你定制 GridBagLayout 类中每

个部件的外观属性。通过首先建立一个 GridBagConstraints 类并设置其中的字段，一旦将其加入了外布局管理器，你就可以指定部件的形状、大小和性能。

▶参见程序“GridBadLayout Implements LayoutManager”

字段

- int anchor

  描述了部件所处的位置。

- final static int BOTH

  fill 字段的一个值。部件以水平和垂直的方式填充所允许的空间。

- final static int CENTER

  anchor 字段的一个值。

- final static int EAST

  anchor 字段的一个值。

- int fill

  确定如果包容类 container 是可缩放时组件应该如何动作。部件可在水平、垂直或两个方向上移动。

- int gridheight

  设置部件的高度。

- int gridwidth

  设置部件的宽度。

- int gridx

  设置部件的横坐标。

- int gridy

  设置部件的纵坐标。

- final static int HORIZONTAL

  fill 字段的一个值。当允许的区域空间超过了部件的大小时，部件以水平方式展开。

- insets insets

  定义在布局管理器边缘的部件周围的空间大小。

• int ipadx

部件之间水平方向上的空间大小。

• int ipady

部件之间垂直方向上的空间大小。

• final static int NONE

fill 字段的一个值。表明组件不向任何方向扩展。

• final static int NORTH

anchor 字段的一个值。

• final static int NORTHEAST

anchor 字段的一个值。

• final static int RELATIVE

字段 gridx 或 gridy 的值。一个部件将紧接着前一个部件放置。对于 gridx 意味着在左边。对于 gridy 意味着在下边。

• final static int REMAINDER

字段 gridwidth 或 gridheight 的值。部件将各自处理行或列的多余部分。

• final static int SOUTH

字段 anchor 的一个值。

• final static int SOUTHEAST

字段 anchor 的一个值。

• final static int SOUTHWEST

字段 anchor 的一个值。

• final static int VERTICAL

字段 fill 的一个值。部件将在允许的区域内在垂直方向上填充。

• double weightx

定义了部件的相对重要性。当在某部件所在位置缩放容器时会用到这个属性。

• double weighty

定义了部件的相对重要性。当在某部件所在位置缩放容器时

会用到这个属性。

• final static int WEST

字段 anchor 的一个值。

方法

※ GridBagConstraints ()

创建一个新的 GridBagConstraints 对象。

※Objiect clone ()

返回某 GridBagConstraints 对象的一个拷贝。

GridBagLayout Implements LayoutManager

GridBagLayout 类具有很多的用途，并且经常做为部件的外观管理器来使用。一个 GridBagLayout 类可以使你以所能想到的任何安排方式放置任何数目的部件（参看图 9）。

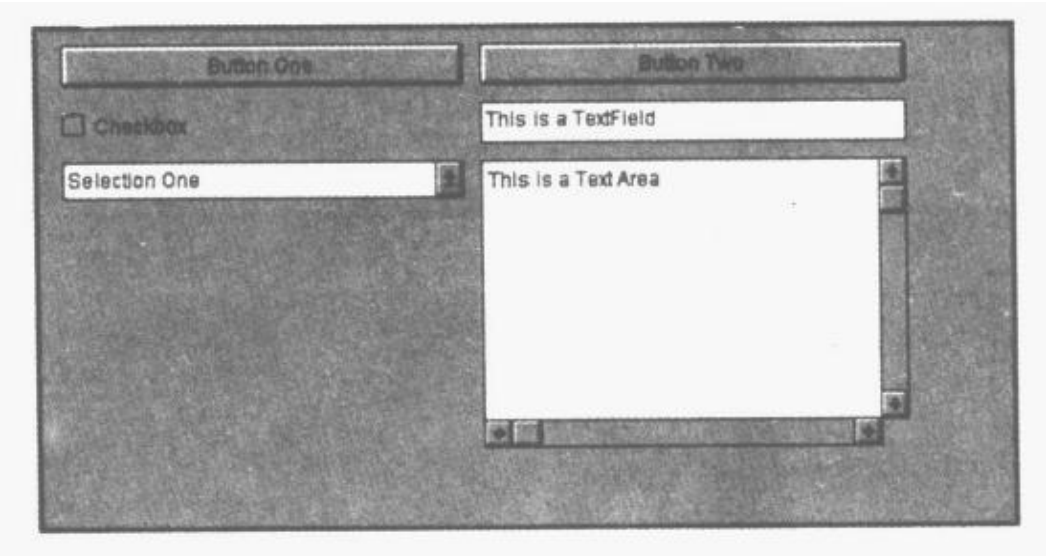


图 9　程序 6 的输出

为了控制 GridBagLayout 中部件的安排，你必须使用 GridBagConstraints 类。对 GridBagConstraints 设置了初值后，你就可以通过使用约束在管理器中添加部件。这样你就可以灵活地确定有关部件安排的大量信息。

▶参见程序“GridBagConstraints Implements java. lang. Cloneable”

GridBagLayou 类中的约束是通过 GridBagLayout 中的字段

来设置的，这些字段描述如下：

字段	用　　途
anchor	用来在容器中的指定区域放置部件（如：SOUTHWEST）
fill	如果容器为一个部件提供可增长的空间，那么这个字段将判定该部件如何动作。合法的值有 NONE、HORIZONTAL、VERTICAL 和 BOTH。
insets	说明包容类器一侧的部件与容器边缘之间的空间大小
ipadx，ipady	指定部件之间的空间大小
gridx，gridy	定义部件左上角所处的位置
gridwidth，gridheight	定义部件的大小
weightx，weighty	定义部件的相对重要性，只有在容器类 container 缩放时才使用

程序 6　GridBagLayout 举例

```
import java.applet.Applet;
import java.awt.*;

public class GridBagExample extends Applet
{
  GridBagLayout gridbag;
  GridBagConstraints c;

  public void init(){
    Button button;
    CheckBox checkbox;
    TextArea textarea;
    TextField textfield;
    Choice choice;

    gridbag=new GridBagLayout();
    c=new GridBagConstraints();
    setLayout(gridbag);
```

```
c. ipadx=2;c. ipady=2; // padding between components
c. insets=new Insets(5,5,5,5); /* padding
between the components and the edge of the container */

button=new Button("Botton one");
gridbag. setConstraints(botton,c);
add(button);

c. gridwidth=GridBagonstraints. REMAINOER;
/* Button will occupy the remainder of this row.
   Forces the next component onto next row. */

button=new Button("Botton two");
gridbag. setConstraints(botton,c);
add(button);

c. gridwidth=1;
//back to defaults-components continue on same row

checkbox=new Checkbox("Checkbox");
gridbag. setConstraints(checkbox,c);
add(checkbox);

c. gridwidth=GridBagConstraints. REMAINDER;
textfield=new TextField("This is a TextField");
gridbag. setConstraints(textfield,c);
add(textfield);

c. gridwidth=1;
```

```
        c.gridheight=GridBagConstraints.REMAINDER;
        choice=new Choice();
        choice.addItem("Selection one");
        choice.addItem("Selection two");
        gridbag.setConstraints(choice,c);
        add(choice);

        textarea=new TextArea("This is a Text Area",10,3);
        gridbag.setConstraints(textarea,c);
        add(textarea);
    }
}
```

字段

- double columweights []

  存储每列中部件的水平加权。

- int columWidths []

  存储每列中部件的水平尺寸。

- protected Hashtable comptable

  存储部件的散列表。

- protected GridBagConstraints defaultConstraints

  针对部件的缺省约束。

- protected GridBagLayoutInfo layoutinfo

GridBagLayoutInfo 类是一个被约束类，它提供了针对 GridBagLayout 类的布局管理，它追踪布局的尺寸、部件的最小空间和权重。

该类只是由下列字段和构造器方法组成。除非要创建 java.awt 包中的一个新类来扩充 GridBagLayout 类，否则用户是不需要这个类的。

```
int width, height;        //每个方向中的单元数
int startx, starty;       //布局的开始
```

int minWidth []; //每列最大的 minWidth 最小宽度
int minHeight []; //每行最大的 minHeight 最小高度
double weightx []; //每列的最大权重
double weighty []; //每行的最大权重

• protected final static int MAXGRIDSIZE

最大行数或列数，最大可设置为 128。

• protected final static int MINSIZE

行与列的最小数目，设置为 1。

• protected final static int PREFERREDSIZE

优选的部件尺寸，设置为 2。

• int rowHeight []

跟踪每行中部件的高度。

• double rowWeight []

跟踪每行中部件的权重。

方法

※ GridBagLayout ()

创建一个新的 GridBagLayout。

※ void addLayoutComponent (String name, Component comp)

当 GridBagLayout 类执行 LayoutManager 接口时需要这个功能，但 GridBagLayout 类本身没有这个功能。

※ protected void AdjustForGravity (GridBagConstraints constraints, Rectangle r)

根据指定的 Rectangle 调整部件。这个方法被 ArrangeGrid () 调用。

※ protected void ArrangeGrid (Container parent)

安排父容器。

※ protected void DumpConstraints (GridBagConstraints constraints)

打印当前约束。

※ protected void DumpLayoutInfo (GridBagLayoutInfo string)

打印当前管理器的信息。

※ GridBagConstraints getConstraints (Component comp)

返回当前的约束。

※ int [] [] getLayoutDimensions ()

返回 LayoutInfo 对象中的一个整型数组。如果返回的数组名为 arr，则 arr [0] 指定了每列的最小宽度，arr [1] 指定了每行的最小高度。

※ protected GridBagLayoutInfo getLayoutInfo (Container parent, int sizeflag)

返回 GridBagLayout 中的 GridBagLayoutInfo 对象。

※ Point getLayoutOrigin ()

返回布局的起始点，并不总是 (0, 0)。

※ double [] [] getLayoutWeights ()

返回每行和每列中的部件的最大权重，arr [0] 指定了每列中的最大权重，arr [1] 指定了每行中的最大权重。

※ protected Dimension GetMinSize (Container parent, GridBagLayoutInfo info)

返回某管理器所需的最小空间。这个值是基于部件的大小和父容器 insets 字段的值。这个方法通过 preferredLayoutSize () 和 minimumLayoutSize () 以不同的 info 参数调用。

※ void layoutContainer (Container parent)

根据 GridBagLayout 的参数来安排包容类 container。

※ Point location (int x, int y)

返回点 (x, y) 的上方和左边的所有部件所占矩形框的大小。这个值等于每个部件的最小宽度和最小高度之和，并且每个部件的横坐标不大于 x，纵坐标不大于 y。

※ protected GridBagConstraints lookupConstraints (Component comp)

返回特定的部件的容器。

※ Dimension minimumLayoutSize (Container parent)

返回容器所需的最小的布局尺寸。这个值是基于该布局中部件所占的最小空间。

※ Dimension preferredLayoutSize (Container parent)

返回容器所占的最佳空间大小。这个值是基于布局中部件所占的最佳空间。

※ void removeLayoutComponent (Component comp)

从布局中移走指定的部件。

※ void setConstraints (Component comp, GridBagConstraints constraints)

对给定的部件分配新的一组约束。

※ String toString ()

返回 java.awt.GridBagLayout 字符串。

GridLayout Implements LayoutManager

一个 GridLayout 类提供不亚于 GridBagLayout 类的布局控制功能。一个 GridLayout 通过指定的行数和列数产生，随后用 add () 方法通过调整部件的大小在布局中加入指定的部件。这样，在 GridLayout 中的所有部件均具有同样的大小。

方法

※ GridLayout (int rows, int cols)

创建一个有指定行数和列数的 GridLayout。行间距 hgap 和列间距 vgap 将置为 0。

※ GridLayout (int rows, int cols, int hgap, int vgap)

创建一个有指定行数、列数、行间距和列间距的 GridLayout。

※ void addLayoutComponent (String name, Component comp)

这个方法无任何功能。

※ void layoutContainer (Container parent)

根据 GridLayout 模型安排容器。

※ Dimension minimumLayoutSize (Container parent)

返回容器所需的最小的布局尺寸。这个值是基于布局中部件所占的最小空间。

※ Dimension preferredLayoutSize (Container parent)

返回容器所占的最佳空间大小。这个值是基于布局中部件所占的最佳空间。

※ void removeLayoutComponent (Component comp)

从管理器中移走指定的部件。

※ String toString ()

返回包括了布局名称、行间距、列间距、行数和列数的字符串。

Abstract Image

Image 类提供了一个控制图象的独立的操作平台。下面的方法适用于所有的图象而不用考虑图象的来源和平台。

在图象操作中所遇到的一个问题就是我们经常要花费一些时间去装入这些图象。所以，在使用存在这个问题的方法时，需要指定一个 ImageCbserver(提供 ImageCbserver 接口的一个对象)。如果某方法不能返回所要的信息，它将返回-1 或空串。一旦产生了正确的信息，它将通过指定的 ImageCbserver 中的 ImageUpdate () 方法来传送。

▶参见“java. awt. Image”

字段

• final static Object UndefinedProperty

对于某图象，当你需要一个未被定义的特性时返回这个字段。

方法

※ Image ()

创建一个新的图象。

※ abstract Graphics getGraphics ()

返回 Image 图象的 Graphics 对象。

※ abstract int getHeight (ImageObsever observer)

返回 Imge 图象的高度。如果当前不知道图象的高度，这个方法将返回-1 并且通过 imageUpdate ()向指定的 ImageObserver 传递信息。

※ abstract Object getProperty (String propertyname, ImageObserver observer)

返回所需要的某图象的属性。如果这个属性没定义，则返回UndefinedProperty 对象。如果当前不知道这个属性，则返回 null 并且向 ImageObserver 传送信息。

※ abstract void flush ()

分配图象所使用的内存，这其中包括了快速缓存信息。如果要重新显示图象，将再进行一次这个操作。如果存在快速缓存，装入会快一些。

※ abstract ImageProcducer getSource ()

以类所提供的 ImageProducer 接口的形式返回图象的来源。当使用 ImageFilters 时，这个方法是很有用的。当生成一个新的 Java. awt. image. FilteredImageSoutce 时，通过首先引用这个方法，就可获得有要使用的 Image 源。

※ abstract int getWidth (ImageObserver observer)

返回图象的宽度。如果当前不知道图象的宽度，这个方法将返回-1 并且通过 imageUpdate () 向指定的 ImageObserver 传递信息。

Insets Implements Cloneable

Insets 类是外观管理器特别是 GridBagLayoutManager 使用的一个工具。Insets 类可以设置并且跟踪管理器和包容类 container 边缘上的部件之间所要插入的空间大小。

▶参见程序 6 “BridBagLayout Example”

字段

- int bottom
- int left
- int right
- int top

方法

※ Insets (int top, int left, int bottom, int right)

在指定的位置创建一个 Insets 对象。

※ Object clone ()

产生一个 Insets 对象的拷贝。

※ String toString ()

返回包含了类名和四个空间位置值的字符串。

Label Extends Component

在屏幕上显示一行文本使用标签 Label 是很方便的（参看图 10）。由于它是一个组件，它应该放在外观管理器中。



图 10　作为一个 Label 的文本

字段

• final static int CENTER

在 Label 中文本对齐方式的一个值。

• final static int LEFT

在 Label 中文本对齐方式的一个值。

• final static int RIGHT

在 Label 中文本对齐方式的一个值。

方法

※ Label ()

创建一个新的空的 Label，它的文本置为左对齐。

※ Label (String label)

创建一个新的有指定内容的 Label，它的文本置为左对齐。

※ Label (String label, int alignment)

创建一个新的有指定内容和对齐方式的 Label，对齐方式的值有 LEFT、CENTER 或 RIGHT。

※ synchronized void addNotify ()

创建一个 Label 的同位体。

※ int getAlignment ()

返回 Label 的对齐方式。

※ String getText ()

返回 Label 当前的文本。

※ protected String paramString ()

返回包含了 Label 参数的字符串，其中包括对齐方式和文本。

※ void setAlignment (int alignment)

设置显示时文本的对齐方式。参数值可以是 LEFT、CENTER 或 RIGHT。

※ void setText (String label)

设置 Label 的文本。

List Extends Component

List 是一种非常有用的原文标题的滚动式索引（参看图 11）。当使用 List 时，你不仅能在表列尾部添加项目，还可以在表列中插入。在运行期间，你可以根据用户的需求对表列中的信息进行排序或者使用 makeVisible () 滚动表列。

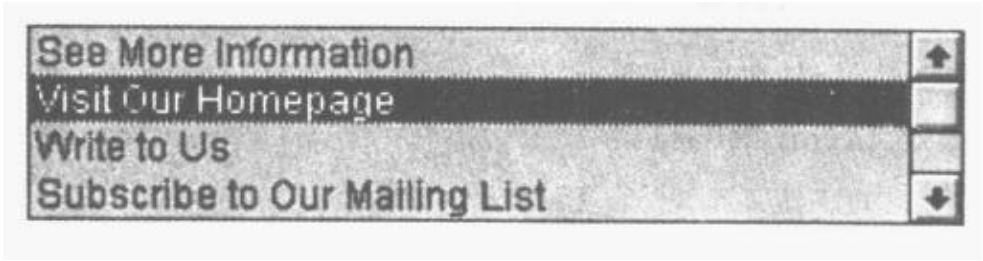


图 11　滚动列表

方法

※ List ()

创建一个新的没有行的 List 并且不允许多重选择。

※ List (int rows, boolean multipleSelection)

创建一个有指定行数的 List。如果 multipleSelection 自变量的值为 true，则 List 在给定的时间内允许进行多重选择。

※ synchronized void addItem (String itemTitle, int index)

在指定的索引处给表列加入一项。itemTitle 指定了要显示的标题。如果 index 值是－1 或大于表列中的项目数，则新的一项将加在表列的尾部。

※ synchronized void addItem (String itemTitle)

根据指定的标题，在表列尾部添加一项。

※ synchronized void addNotify ()

产生一个表列的同位体。

※ boolean allowsMultipleSelections ()

如果用户在表列中进行多重选择，则返回 true。

※ synchronzied void clear ()

清除表列，所有的项内容均被清除。

※ int countItems ()

返回表列中所有项的数目。

※ String getItem (int index)

返回指定索引位置处项的标题。

※ Synchronized void delItem (int index)

删除指定的索引项。

※ synchronized void deleteItems (int star, int end)

删除 start 和 end 之间的（包括 start 和 end）所有项。

※ synchronized void deselect (int index)

取消已经选中的索引。没有任何项被选择。

※ int getRows ()

返回可以被用户看到的项的数目。

※ synchronzied int getSelectedIndex ()

返回当前被选中的项的索引。如果返回值为－1，则没有一条被选中。

※ synchronzied int [] getSelectedIndexs ()

返回包含了所有被选中项的索引的数组。只有允许多重选择时才可以使用。

※ synchronzied String getSelectedIndex ()

返回当前被选中条目的标题。如果返回值为－1，则没有一条被选中。

※ synchronzied String [] getSelectedIndexs ()

返回包含了所有被选中项的标题的数组。只有允许多重选择时才可以使用。

※ int getVisibleIndex ()

返回项的索引，这个项是由最近使用的 makeVisible () 置为可视的。

※ synchronized boolean isSelected (int index)

如果指定的条目是可视的，则返回 true。

※ void makeVisible (int index)

滚动表列以使指定的索引可视。

※ Dimension minimumSize ()

返回正确显示整个表列所需最小空间尺寸。这个值基于表列同位体的返回值或者基于从 Component 类返回的值。

※ Dimension minimumSize (int rows)

返回正确显示整个表列中指定的行数所需最小空间的大小。这个值是基于 List 同位体的返回值或者从 Component 类返回的值。

※ protected String paramString ()

返回包含有关表列信息的字符串，其中有被选择项的标题。

※ Dimension preferredSize ()

返回正确显示整个表列所需最佳空间的大小。这个值是基于表列同位体的返回值或者基于从 Component 类返回的值。

※ Dimension preferredSize (int rows)

返回正确显示表列中指定的行数所需最佳空间的大小。这个值是基于表列同位体的返回值或者基于从 Component 类返回的值。

※ synchronzied void removeNotify ()

移走表列的同位体。

※ synchronized void replaceItem (String newValue, int index)

改变在指定索引位置处项的值。首先删除指定索引位置的项，然后再创建一个有指定标题和索引的项。

※ synchronzied void select（int index）

选择指定的项。

※ void setMultipleSelections（boolean multipleSelections）

如果参数 multipleSelections 的值为 true，则同一时刻在表列中可以进行多重选择。如果它的值为 false，则同一时刻在表列中只能进行单一的选择。

MediaTracker

MediaTracker 类用来管理多媒体（如图象和音频）对象（但是，在 JDK1.0.2 版本中只支持图象）。就像一个数组一样，MediaTracker 类分配了每一个参考数目。但是，它也对每个图象的状态进行跟踪，允许你决定每个图象的状态或者锁定跟踪器直到装入所有图象。

字段

• final static int ABORTED

一个标志，指明媒体的装入已被中止。

• final static int COMPETE

一个标志，表明媒体已经正确地装入。

• final static int ERRORED

一个标志，表明媒体装入时出现错误。

方法

※ MediaTracker（Component comp）

创建一个新的 MediaTracker 报告，以记录指定组件中的媒体。

※ void addImage（Image image，int id）

在 MediaTracker 中添加一个有指定识别号的项。这个识别号在以后会用作对一组媒体项的参考。

※ synchronzied void addImage（Image image，int id，int width，int height）

在 MediaTracker 中添加一个有指定识别号和指定大小的图象。这个识别号在以后会用作对一组媒体项的参考。

※ synchronzied boolean checkAll (boolean load)

如果所有媒体成功地装入，则返回 true，否则返回 false。如果参数 load 的值是 true，则这个方法将装入任何未被装入的图象。

※ boolean checkAll ()

如果所有媒体成功地装入，则返回 true。这个方法对未被装入的图象不起作用。

※ synchronzied boolean checkId (int id boolean load)

如果所有指定了识别号的媒体成功地装入，则返回 true，否则返回 false。如果参数 load 的值是 true，则这个方法将装入任何未被装入的图象。

※ boolean checkID (int id)

如果所有指定了识别号的媒体成功地装入，则返回 true，否则这回 faese。这个方法对未被装入的图象不起作用。

※ synchronzied Object [] getErrorsAny ()

返回一个数组，这个数组包含了那些出现错误的媒体项。

※ synchronzied Object [] getErrorsID (int id)

返回一个数组，这个数组包含了那些出现错误而且指定了识别号的媒体项。

※ synchronzied boolean isErrorAny ()

如果任何媒体项出现了错误，则返回 true。

※ synchronzied boolean isErrorID (int id)

如果任何指定了识别号的媒体项目出现了错误，则返回 true。

※ int statusAll (boolean load)

返回 MediaTracker 中所有媒体项的状态。如果参数 load 的值为 true，将装入任何未被装入的图象。

※ int statusID (int id, boolean load)

返回 MediaTracker 中所有指定了识别号的媒体项目的状态。如果参数 load 的值为 true，将装入任何未被装入的图象。

※ void waitForAll () throws InterruptedException

开始装入所有的媒体项并且直到整个过程结束才返回。记住，这个方法并不认为这些项会被成功地装入，所以你要仔细地检查任何错误。

※ synchronzied boolean waitForAll (long ms) throws InterruptedException

检查 MediaTracker 对象并装入所有的媒体项。只有当装入完成或者超过了指定的毫秒数，这个方法才放弃对 MediaTracker 的控制。注意，这个方法并不认为这些项都会被成功地装入，所以你要仔细地检查任何错误。

※ void waitForID (int id) throws InterruptedException

开始装入所有指定了识别号的媒体项并且直到整个过程结束才返回。记住，这个方法并不认为这些项都会被成功地装入，所以你要仔细检查任何错误。

※ synchronzied boolean waitForID (int id, long ms) throws InterruptedException

开始装入所有指定了识别号的媒体项目。只有当装入完成或者超过了指定的毫秒数，这个方法才放弃对 MediaTracker 的控制。注意，这个方法并不认为这些项目都会被成功地装入，所以你要仔细地检查任何错误。

Menu Extends MenuItem Implements MenuContainer

Menu 只是 MenuBar 的一列。就像大多数应用中的“文件”和“帮助”题头一样，一个 Menu 定义了 MenuBar 上的标题以及所处列的头信息。当创建一个菜单时，你首先要生成一个 Menu 条目并把它加到你的 MenuBar 中去。

▶参见 “MenuBar Extends MenuComponent Impements MenuContainer”

▶参见 “Frame Extens Window Implements MenuContainer”

方法

※ Menu (String title)

创建一个新的有指定标题的 Menu。

※ Menu (String title, boolean tearOff)

创建一个新的有指定标题的 Menu。参数 tearOff 是独立于操作平台的。一个 tear-off 型的菜单就是保持一个独立的窗口直到用户从中选择一个条目。

※ void add (String label)

在 Menu 表中添加一个有指定标签的选项。

※ synchronzied MenuItem add (MenuItem mitem)

在 Menu 表中添加一个 MenuItem。

※ synchronzied void addNotify ()

创建 Menu 的同位体。

※ void addSeparator ()

在 Menu 中，在已经存在的条目和后来添加的条目之间插入一行。

※ int countItems ()

返回 Menu 表中条目的数目。

※ MenuItem getItem (int index)

返回在指定索引处的条目。

※ boolean isTearOff ()

如果菜单是 tear-off 型的，则返回 true。

※ synchronzied void remove (MenuComponent item)

在菜单表中移走指定的条目。

※ synchronzied void remove (int index)

在菜单表中指定的索引处移走指定的条目。

※ synchronzied void removeNotify ()

移走 Menu 的同位体。

MenuBar Extends MenuComponent Implements MenuContainer

MenuBar 是 Menu 项目的容器并且当菜单显示时做为菜单的头。若要在程序中生成一个菜单，必须首先创建一个 Menu 项目，然后把它加到 MenuBar 中去。通过将 MenuBar 加到 Frame

中，就可以使菜单变为可视的。

▶参见“Menu Extends MenuItem Implements MenuContainer”

▶参见“Frame Extends Window Implements MenuContainer”

图 12 显示的菜单的源代码见程序 7。还应注意，使用 Display 类可使 applet 对用户的选择进行监视。

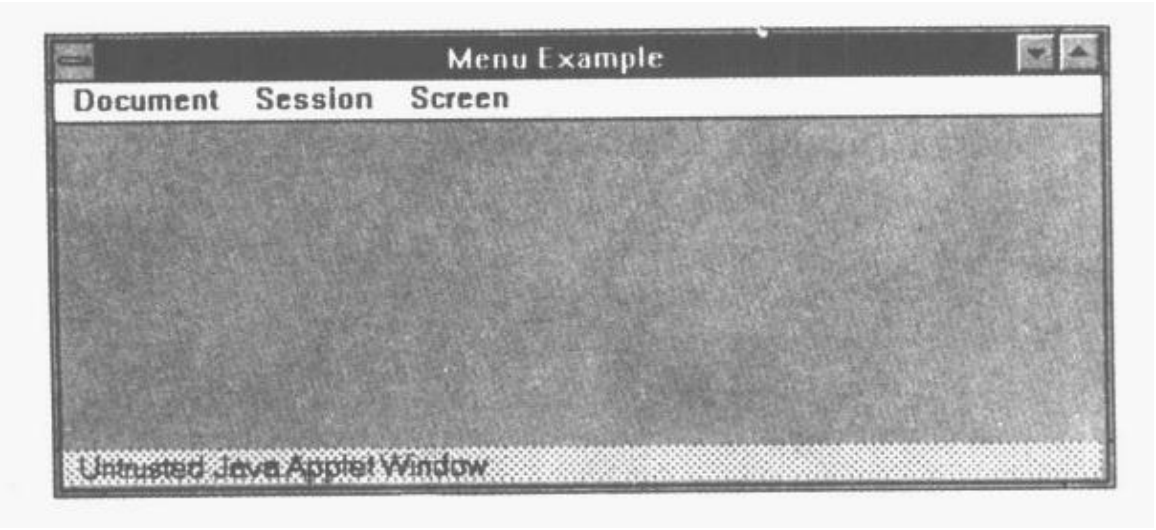


图 12　程序 7 的输出

程序 7　Menu Example

```
import java.applet.Applet;
import java.awt.*;

public class MenuExample extends Applet
{
   private Menu category;
   private MenuBar bar;
   private Display picture;

   public void init(){
      bar=new MenuBar();
      category=new MenuBar("Document");
      category.add("Option One");
```

```
        category.add("Option Two");
        category.add("Option Three");
        bar.add(category);

        category=new Menu("Session");
        category.add("Reset");
        category.add("Restart");
        bar.add(category);

        category=new Menu("Screen");
        category.add("Change colors");
        category.add("Display Title");
        bar.add(category);

        picture=new Display();
        picture.setMenuBar(bar);
        picture.setTitle("Menu Example");
        picture.show();
    }
}

class Display extends Frame{
    public boolean action(Event evt,Object arg){
        if (arg.equals("Option Two")){
            System.out.println("Option Two was Selected");
        return true;
        }
    return super.handleEvent(evt);
    }
```

```
  public boolean handleEvent(Event evt){
    if (evt.id=Event.WINDOW_DESTROY){
      hide();//closes the frame when appropriate
    return true;
    }
  return super.handleEvent(evt);
  }
}
```

方法

※ MenuBar ()

创建一个新的 MenuBar。

※ synchronzied Menu add (Menu m)

在 MenuBar 中添加一个 Menu。

※ synchronzied void addNotify ()

创建一个 MenuBar 的同位体。

※ int countMenus ()

返回 MenuBar 中 Menu 的个数。

※ Menu getHelpMenu ()

返回某 MenuBar 的帮助菜单。

※ Menu getMenu (int index)

返回在指定的索引位置的 Menu。

※ synchronzied void remove (MenuComponent m)

移走指定的 MenuComponent。

※ synchronzied void remove (int index)

在指定的索引位置移走 MenuComponent。

※ void removeNotify ()

移走 MenuBar 的同位体。

※ synchronzied void setHelpMenu (Menu m)

为 MenuBar 设置指定的帮助菜单。

Abstract MenuComponent

MenuComponent 是处理菜单有关部件的一种超类（superclass）。

▶参见“Menu Extends MenuItem Implements MenuContainer”

方法

※ MenuComponent ()

创建一个新的 MenuComponent。

※ MenuContainer getParent ()

返回某 MenuComponent 的容器。通常是 MenuComponent 所处的 Menu。

※ MenuComponentPeer getPeer ()

返回 MenuComponent 的同位体。

※ Font getFont ()

返回 MenuComponent 的字体。

※ protected String paramString ()

返回包括了 MenuComponent 信息的一个字符串。即使是在子类中被替换，这个方法也将返回一个空串。

※ boolean postEvent (Event evt)

向 MenuComponent 传送一个指定的事件 Event。

※ void removeNotify ()

移走 MenuComponent 的同位体。

※ void setFont (Font f)

设置 MenuComponent 的字体。

※ String toString ()

返回对象名称以及 paramString () 所返回的内容。

MenuItem Extends MenuComponent

MenuItem 是一个容器，它包容了表示菜单可选项的字符串。它通常被更有用的 menu 类来使用和继承。

▶参见“CheckboxMenuItem Extends MenuItem”

方法

※ MenuItem（String label）

创建一个新的有指定 label 的 MenuItem。

※ synchronzied void addNotify（）

生成 MenuItem 的同位体。

※ void disable（）

禁止用户选择 MenuItem。

※ void enable（boolean cond）

如果参数 cond 的值为 true，则允许用户选择 MenuItem，否则，用户将不能进行选择。

※ void enable（）

允许用户选择 MenuItem。

※ String getLabel（）

返回这个 MenuItem 的正文标签。

※ boolean isEnable（）

如果允许用户选择 MenuItem，则返回 true。

※ String paramString（）

返回包含项目标号的字符串。

※ setLabel（String label）

设置 MenuItem 的标签。

Panel Extends Container

Panel 类是一个简单的容器，它可以被扩充从而产生内容更丰富的子类。java. applet. Applet 就是 Panel 的子类。

▶参见程序 3 “CardLayout Example Source Code”

方法

※ Panel（）

创建一个新的 Panel。

※ synchronzied void addNotify（）

生成 Panel 的同位体。

Point

Point 是一个简单的类，用来对有序偶（ordered pair）进行跟

踪和控制。

字段

• int x

• int y

方法

※ Point (int x, int y)

创建一个有指定横坐标和纵坐标的 Point。

※ boolean equals (Object anotherobject)

用一个 Point 对象去和另一个对象进行比较。如果横坐标和纵坐标相等，则这两个对象就相等。

※ int hashCode ()

返回 Point 对象的 hashcode。这个值依赖于横坐标和纵坐标。

※ void move (int x, int y)

用指定的 x 和 y 值设置 Point 的坐标。

※ String toString ()

返回包含了类名、横坐标和纵坐标的一个字符串。

※ void translate (int x, int y)

用指定的数增大 Point 的横坐标和纵坐标。

Polygon

Polygon 类能够定义多边形 polygon 的属性。这个类不能在屏幕上画多边形，但它可以指定由 Graphics 类中的方法所使用的坐标。

为了跟踪顶点的坐标，这个类维护两个并行的数组 xpoints [] 和 ypoints []。例如：如果 #2 点是 (4, 5)，则 xpoints [2] 等于 4，ypoints [2] 等于 5。

▶参见程序 5 “Graphics Example”

字段

• int npoints

多边形的顶点数。这个值将在定义了的 polygon 中使用，而不用考虑 xpoints [] 和 ypoints [] 数组值的个数，这个值也被用在

定义 polygon 上。

• int xpoints []

多边形顶点的横坐标，这些值与 ypoints [] 数组中相应的值配对生成多边形的角。

• intypoints []

多边形顶点的纵坐标，这些值与 xpoints [] 数组中相应的值配对生成多边形的角。

方法

※ Polygon ()

创建一个新的 Polygon。

※ Polygon (int xpoints [], int ypoints [], int npoints)

产生一个指定顶点数的 Polygn，npoints 指定了应从数组中读取的点的个数。

※ void addPoint (int x, int y)

在多边形中加一个顶点。

※ Rectangle getBoundingBox ()

返回包围多边形的一个矩形框。

※ boolean inside (int x, int y)

如果给定的点在多边形 polygon 中，则返回 true。

Rectangle

Rectangle 类定义了一个矩形框的角。它不能在屏幕上实际地画一个矩形框，而是让用户创建和跟踪矩形的对象。

字段

• int height

• int width

• int x

• int y

方法

※ Rectangle ()

创建一个新的 Rectangle。

※ Rectangle (Dimension d)

创建一个新的指定大小的 Rectangle。

※ Rectangle (int x, int y, int width, int height)

创建一个指定了坐标和大小的 Rectangle。

※ Rectangle (int width, int height)

创建一个具有指定坐标的新 Rectangle。

※ Rectangle (Point p)

创建一个指定了坐标的 Rectangle。

※ Rectangle (Point p, Dimension d)

创建一个指定了坐标和大小的 Rectangle。

※ void add (Rectangle r)

扩大当前的 Rectangle 使之包围指定的 Rectangle。

※ void add (Point pt)、

扩大当前的 Rectangle 使之包围指定的 Point。

※ void add (int newx, int newy)

扩大当前的 Rectangle 使之包围指定的坐标。

※ boolean equals (Object anotherobject)

使 Rectanglet 和另一个对象进行比较。如果横坐标和纵坐标及尺寸相等，则这两个 Rectangle 就相等。

※ void grow (int h, int v)

用特定的量来增大矩形的尺寸。

※ int hashCode ()

返回 Rectangle 的 hashcode。这个值依赖于 Rectangle 的坐标和大小。

※ boolean inside (intx, inty)

如果指定的坐标在矩形框 rectangle 中，则返回 true。

※ Rectangle intersection (Rectangle r)

返回代表某矩形框和另外一个指定的矩形框的交叉点的 Rectangle。

※ boolean inersects (Rectangle r)

如果某矩形框和指定的矩形框交叉，则返回 true。

※ booleab isEmpty ()

如果 Rectangle 的高和宽均为 0，则返回 true。

※ void move (int x, int y)

移动矩形框到指定的位置。

※ void reshape (int x, int y, int width, int height)

指定某矩形框新的坐标和大小。

※ void resize (int width, int height)

改变矩形框 Rectangle 的大小。

※ String toString ()

返回包含有类名、矩形框的坐标和大小的一个字符串。

※ void translate (int x, int y)

用指定的 x 和 y 值来增大矩形框的坐标。

※ Rectangle union (Rectangle r)

创建一个新的 Rectangle 对象，它包含了当前的和另一个 Rectangle 对象。

Scrollbar Extends Component

Scrollbar 类用途很广泛，它可以用来在用户程序中生成滚动条。Scrollbar 类可以生成竖直和水平的滚动条并且指定滚动条的变化范围。同其他的 Component 对象一样，Scrollbar 对象可以放在外观管理器中。

程序 8 的代码生成了一个简单的计算器来简化输入。尽管滚动条可以自动地自我调整，但还应注意 Calculate 按钮是如何修改 TextField 的。

程序 8 Scrollbar Example

```
import java.applet.Applet;
import java.awt.*;

public class ScrollbarExample extends Applet
{
```

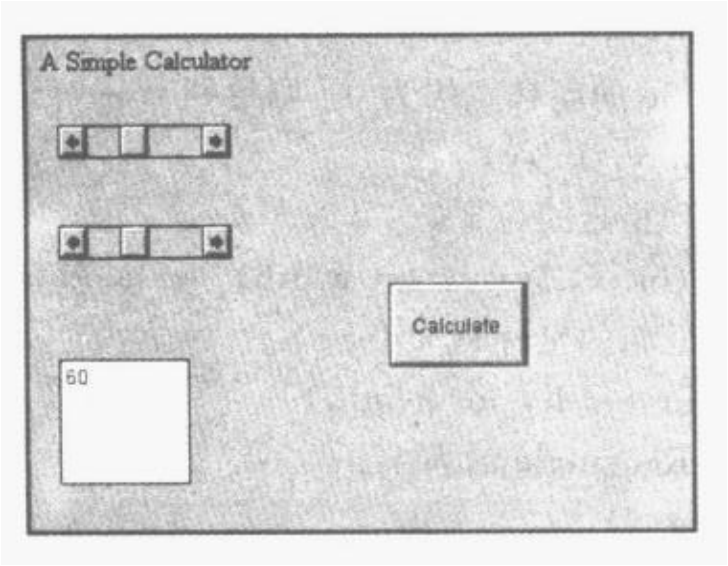


图 13　滚动条计算器

```
Scrollbar sb1,sb2;
TextField info;
Button trigger;
public void init(){

    setLayout(null);//won't use one of the layout managers
    add(sb1);
    sb1.reshape(10,10,100,20);

    sb2 = new  Scrollbar (Scrollbar.HORIZONTAL, 30, 20, 0,
100);
    add(sb2);
    sb2.reshape(10,70,100,20);

    trigger=new Button("Calculate");
    add(trigger);
    trigger.reshape(200,105,80,50);
```

```
    info=new TextField("0");
    add(info);
    info.reshape(10,150,75,75);

    refresh();
  }

  void refresh(){
    int sum=sb1.getValue()+sb2.getValue();
    info.setText(""+sum);
  }

  public boolean action(Event evt,Object arg){
    if (arg.equals("Calculate")){
      refresh();
      return true;
    }
    return false;
  }
}
```

字段

- final static int HORIZONTAL

Scrollbar 定位时的一个可能值。产生一个水平的滚动条 Scrollbar。

- final static int VERTICAL

Scrollbar 定位时的一个可能值。产生一个竖直的滚动条 Scrollbar。

方法

※ Scrollbar ()

根据对 VERTICAL 的定位设置，创建一个新的 Scrollbar。

※ Scrollbar (int orientation)

以指定的定位参数 orientation 创建一个 Scrollbar 对象。

※ Scrollbar (int orientation, int value, int visible, int minimum, int maximum)

以指定的初始化参数 orientation 创建一个 Scrollbar。参数 value 是 Scrollbar 的初始值。参数 visible 是一个属性，它表明了当应用窗口滚动时可以看到的条目的数目。参数 minimum 和 maximum 指定了 Scrollbar 的最大和最小值。

※ synchronzied void addNotify ()

生成一个 Scrollbar 的同位体。

※ int getLineIncrement ()

返回当滚动条移动一行时滚动条的值的变化情况。当用鼠标单击滚动条两端的箭头时，滚动条就会移动一行。

※ int getMaximum ()

返回滚动条的最大值。

※ int getMinimum ()

返回滚动条的最小值。

※ int getOrientation ()

返回滚动条初始化的值是 HORIZONTAL 还是 VERTICAL。

※ int getPageIncrement ()

返回当滚动条移动一页时滚动条的值的变化情况。当用鼠标单击滚动条浮标的一端时，滚动条就会移动一页。

※ int getValue ()

返回滚动条的值。

※ int getVisible ()

返回滚动条中可视的数目。

※ protected String paramString ()

返回包含了 Scrollbar 信息的字符串，其中包括它当前的值、可视的个数、最大值和最小值。

※ void setLineIncrement (int l)

设置当滚动条移动一行时滚动条的变化值。当用鼠标单击滚动条两端的箭头时，滚动条就会移动一行。

※ void setPageIncrement (int l)

设置当滚动条移动一页时滚动条的变化值。当用鼠标单击滚动条浮标的一端时，滚动条就会移动一页。

※ void setValue (int value)

设置滚动条的值。如果参数 value 的值超出了范围，则它将被置为滚动条的最大值或者最小值。

※ void setValues (int value, int visible, int minnmum, int maximum)

设置滚动条的参数。参数 orientation (HORIZONTAL 或者 VERTICAL) 决定了滚动条的表面特征。参数 value 是滚动条 Scrollbar 的初始值。参数 visible 是一个属性，它表明了当应用窗口滚动时可以看到的条目的数目。参数 minimum 和 maximum 指定了滚动条的最大和最小值。

TextArea Extends TextComponent

TextArea 是用来显示和编辑文本的。所以，生成的 TextArea 必须是可以滚动的 (如图 14)。

▶参见程序 "GridBagLayout Example"

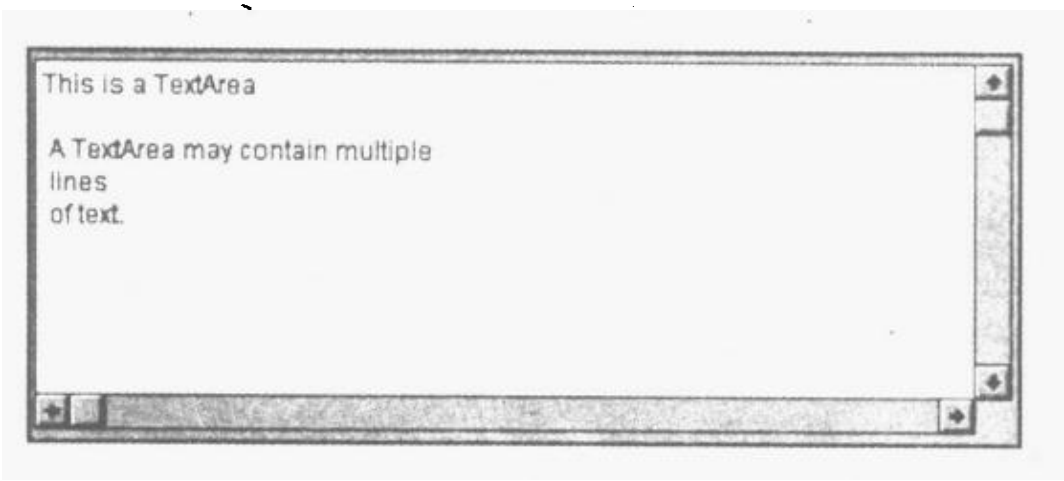


图 14　Windows NT 中的 TextArea

方法

※ TextArea ()

创建一个空的 TextArea。

※ TextArea (int rows, int cols)

创建一个有指定行数和列数的 TextArea。

※ TextArea (String text)

创建一个有指定文本的 TextArea。

※ TextArea (String text, int rows, int cols)

创建一个有指定文本、行数和列数的 TextArea。

※ synchronzied void addNotify ()

生成 TextArea 的同位体。

※ void appendText (String newText)

在当前显示的文本后加入指定的文本。

※ int getColumns ()

返回 TextArea 中的列数。

※ int getRows ()

返回 TextArea 中的行数。

※ void insertText (String str, int pos)

在 TextArea 中，在指定的位置插入文本。

※ Dimension minimumSize ()

返回整个组件所需的最小空间。

※ Dimension minimumSize (int rows, int cols)

返回指定了行数和列数的 TextArea 所需的最小空间。

※ protected String paramString ()

返回包含了文本和 TextArea 尺寸的字符串。

※ Dimension preferredSize ()

返回整个部件所需的最佳空间。

※ Dimension preferredSize (int rows, int cols)

返回指定了行数和列数的 TextArea 所需的最小的优化空间。

※ void replaceText (String newText, int start, int end)

在参数 start 和 end 所规定范围内用指定的文本 newText 进行替换。参数 newText 的长度不一定等于被替换文本的长度。

TextComponenet Extends Component

TextComponent 是文本显示部件 TextArea 和 TextField 的基础结构块(图 15)。但是,它具有存取等级的限制,除非你在 java.awt 包中进行开发,否则你不能使用它。所以,尽管建立了若干有用的方法,但这个类通常并不直接地被使用。



图 15 Windows NT 中的 TextArea

方法

※ String getSelectedText ()

返回当前被用户加亮的文本。

※ int getSelectionEnd ()

返回加亮部分中最后一个字符的索引。

※ int getSelectionStart ()

返回加亮部分中第一个字符的索引。

※ boolean isEditable ()

如果显示的文本可以被修改,则返回 true。

※ protected String paramString ()

返回一个字符串,它包含了显示的文本、文本加亮部分的头索引和尾索引、文本是否可以被修改等信息。

※ void select (int start, int end)

选择指定索引之间的所有文本。

※ void selectAll ()

选择所有的文本。

※ void setEditable (boolean editable)

如果参数 editable 的值为 true,则文本可以被用户修改,否则

不能。

TextField Extends TextComponent

TextField 是一个简单的部件，它用来显示和编辑一行文本。

方法

※ TextField ()

创建一个新的 TextField。

※ TextField (int cols)

创建一个新的有指定列数的 TextField 对象。

※ TextField (String text)

创建一个有指定文本内容的 TextField 对象。

※ TextField (String text, int cols)

创建一个新的有指列行数和有指定文本内容的 TextField。

※ synchronzied void addNotify ()

生成一个 TextField 的同位体。

※ boolean echoCharIsSet ()

如果 TextField 中设置了回应字符 (echo)，则返回 true。回应字符就是显示的字符并不是用户所输入的实际字符。这种特性通常被用来在 TextField 中输入口令。

※ int getColumns ()

返回 TextField 中列的数目。

※ char getEchoChar ()

返回回应字符。回应字符是做为用户输入的一种替代而被显示的。

※ Dimension minimumSize ()

返回组件所需的最小空间。

※ Dimension minimumSize (int cols)

返回指定了列数的 TextField 所需的最小空间。

※ protected String paramString ()

返回包含了当前文本和回应字符的字符串。

※ Dimension preferredSize ()

返回组件所需的最佳空间大小。

※ Dimension preferredSize (int cols)

返回指定了列数的 TextField 所需的最佳空间大小。

※ void setEchoCharater (char c)

设置回应字符。显示的字符并不是用户实际输入的。

Toolkit

Toolkit 类是作为 AWT 的抽象方法与执行这些方法的特定平台之间的一种联系。通常，用户不必和 Toolkit 直接发生联系，但偶尔也可以使用一些方便的 Toolkit 方法（如 getImage ()）。在这种情况下，须运用 Component 类中的 getToolkit () 方法来获得当前的 Toolkit。注意，getToolkit () 方法返回驻留部件内所驻留的 Frame 的 Toolkit。

方法

※ Toolkit ()

创建一个新的 Toolkit 类。

※ protected abstract ButtonPeer createButton (Button target)

生成指定的 Button 并返回它的同位体。

※ protected abstract CanvasPeer createCanvas (Canvas target)

生成指定的 Canvas 并返回它的同位体。

※ abstract int checkImage (Image image, int width, int height, ImageObserver observer)

返回指定 image 图象的状态，这个状态可以通过在指定的 ImageObserver 中的 imageUpdate () 方法进行修改。

※ abstract checkboxPeer createCheckbox (Checkbox target)

生成指定的 Checkbox 并返回它的同位体。

※ abstract Image createImage (ImageProducer producer)

返回一个由 ImageProducer 提供信息而生成的 Image。

※ Protected abstract CheckboxMenuItemPeer createCheckboxMenuItem (CheckboxMenuItem target)

生成指定的 CheckboxMenuItem 并返回它的同位体。

※ protected abstract ChoicePeer createChoice (Choice target)

生成指定的 Choice 并返回它的同位体。

※ protected abstract DialogPeer createDialog (Dialog target)

生成指定的 Dialog 并返回它的同位体。

※ protected abstract FileDialogPeer createFileDialog (FileDialog target)

生成指定的 FileDialog 并返回它的同位体。

※ protected abstract FramePeer createFrame (Frame target)

生成指定的 Frame 并返回它的同位体。

※ protected abstract LabelPeer createLabel (Label target)

生成指定的 Label 并返回它的同位体。

※ protected abstract ListPeer createList (List target)

生成指定的 List 并返回它的同位体。

※ protected abstract MenuPeer createMenu (Menu target)

生成指定的 Menu 并返回它的同位体。

※ protected abstract MenuBarPeer createMenuBar (MenuBar target)

生成指定的 MenuBar 并返回它的同位体。

※ protected abstract MenuItemPeer createMenuItem (MenuItem target)

生成指定的 MenuItem 并返回它的同位体。

※ protected abstract PanelPeer createPanel (Panel target)

生成指定的 Panel 并返回它的同位体。

※ protected abstract ScrollbarPeer createScrollbar (Scrollbar target)

生成指定的 Scrollbar 并返回它的同位体。

※ protected abstract TextAreaPeer createTextArea (TextArea target)

生成指定的 TextArea 并返回它的同位体。

※ protected abstract TextFieldPeer createTextField (TextField

target)

生成指定的 TextField 并返回它的同位体。

※ protected abstract WindowPeer createWindow (Window target)

生成指定的 Window 并返回它的同位体。

※ abstract ColorModel getColorModel ()

返回当前屏幕显示所使用的颜色模式 ColorModel。

※ static synchronzied Toolkit getDefaultToolkit ()

返回这次执行中使用的标准的 Toolkit。

※ abstract String [] getFontList ()

返回一个数组，它包含了某应用中使用的所有可能字体。

※ abstract FontMetrics getFontMetrics (Font font)

根据内部信息，返回当前字体的 FontMetrics。

※ abstract Image getImage (URL url)

返回指定 Image。

※ abstract Image getImage (String filename)

返回指定的 Image。

※ abstract int getScreenResolution ()

以每英寸点数的形式返回屏幕的分辨率。

※ abstract Dimension getScreenSize ()

以象素的形式返回屏幕的大小。

※ abstract boolean prepareImage (Image image, int width, int height, ImageObserver observer)

以指定的宽度和高度准备显示一个图象。ImageObserver 将监视这个方法的成功与否。

※ abstract void sync ()

同步一些图形操作。

Window Extends Container

Window 类只是一个简单的容器，它用于从一个 Frame 中产生的弹出式窗口。由于 Frame 类扩充了 Window 类并建立于它之

上，从而可以只使用 Frame 而不用 Window。

方法

※ Window（Frame parent）

创建一个属于指定 parent 的 Window。

※ synchronzied void addNotify（）

生成一个 Window 的同位体。

※ synchronzied void dispose（）

清除 Window 的同位体。

※ Toolkit getToolkit（）

返回某 Window 的 Toolkit。

※ final String getWarningString（）

返回警告用户的一个字符串。在 Netscape 中，这个字符串是“Untrusted Java Window”。

※ synchronzied void pack（）

将 Window 缩小到可以显示所有部件的最小尺寸。

※ synchronzied void show（）

使某 Window 可视。

※ void toback（）

将某 Window 放在它的父 Frame 之后。

※ void toFront（）

将某 Window 放在它的父 Frame 之前。

2. 接口

LayoutManager

LayoutManager 类定义了一个所有布局管理器要使用的动作。

方法

※ void addLayoutComponent（String name，Component comp）

在布局中添加指定的 Component 参数，name 用来提供给管

理器一些有关部件的信息。这个参数的应用视不同的管理器而不同。

※ void layoutContainer (Container parent)

安排指定的 container。

※ Dimension minimumLayoutSize (Container parent)

对于指定的 container，根据它的部件返回它所需的最小空间。

※ Dimension preferredLayoutSize (Container parent)

对于指定的 container，根据它的部件返回它所需的最佳空间。

※ void removeLayoutComponent (Component comp)

从管理器中移走指定的部件。

MenuContainer

MenuContainer 接口被所有的包容类 container 使用去处理各种菜单。

方法

※ Font getFont ()

返回显示 MenuContainer 时用到的字体。

※ boolean postEvent (Event evt)

向 MenuContainer 发送一个事件。

※ void remove (MenuComponent comp)

从 MenuContainer 中移走指定的组件。

三、JAVA.AWT.IMAGE

java.awt.image 包中有一些工具，这些工具是针对跨越网络来管理图象而设计的。因为这个包中的所有类和接口是紧密相关的，所以读者可能看到一些方法会多次出现。

1. 类

Abstract ColorModel

这个抽象类描述了任何ColorModel 类所需要的一些功能，这个类将颜色标识（red，green，blue 或 alpha）转换为实际显示的颜色。

字段

protected int pixel _ bits

每个像素所占位数。

方法

※ ColorModel (int bits)

根据指定的每个像素所占位数创建一个新的 ColorModel。

※ abstract int getAlpha (int pixel)

返回指定象素的 alpha 值。

※ abstract int getBlue (int pixel)

返回指定象素的蓝色值。

※ abstract int getGreen (int pixel)

返回指定象素的绿色值。

※ int getPixelSize()

返回某 ColorModel 中每个象素所占位数。

※ abstract int getRed (int pixel)

返回指定像素的红色值。

※ int getRGB (int pixel)

返回在缺省的 RGB ColorModel 中某象素的颜色值。

※ static ColorModel getRGBdefault ()

返回显示由 RGB 值定义的某象素时所用到的缺省 ColorModel。

CropImageFilter Extends ImageFilter

CropImageFilter 类是一个图象过滤器，它可以使你在其他图象的基础之上创建一个新的图象。图 16 中左上角的图象是从一个更大的图象中抽取出来的，这是程序 9 运用 CropImageFilter 所实现的。尽管每个图象过滤器各不相同，但这里所讲的步骤是使用任何过滤器时都要执行的。

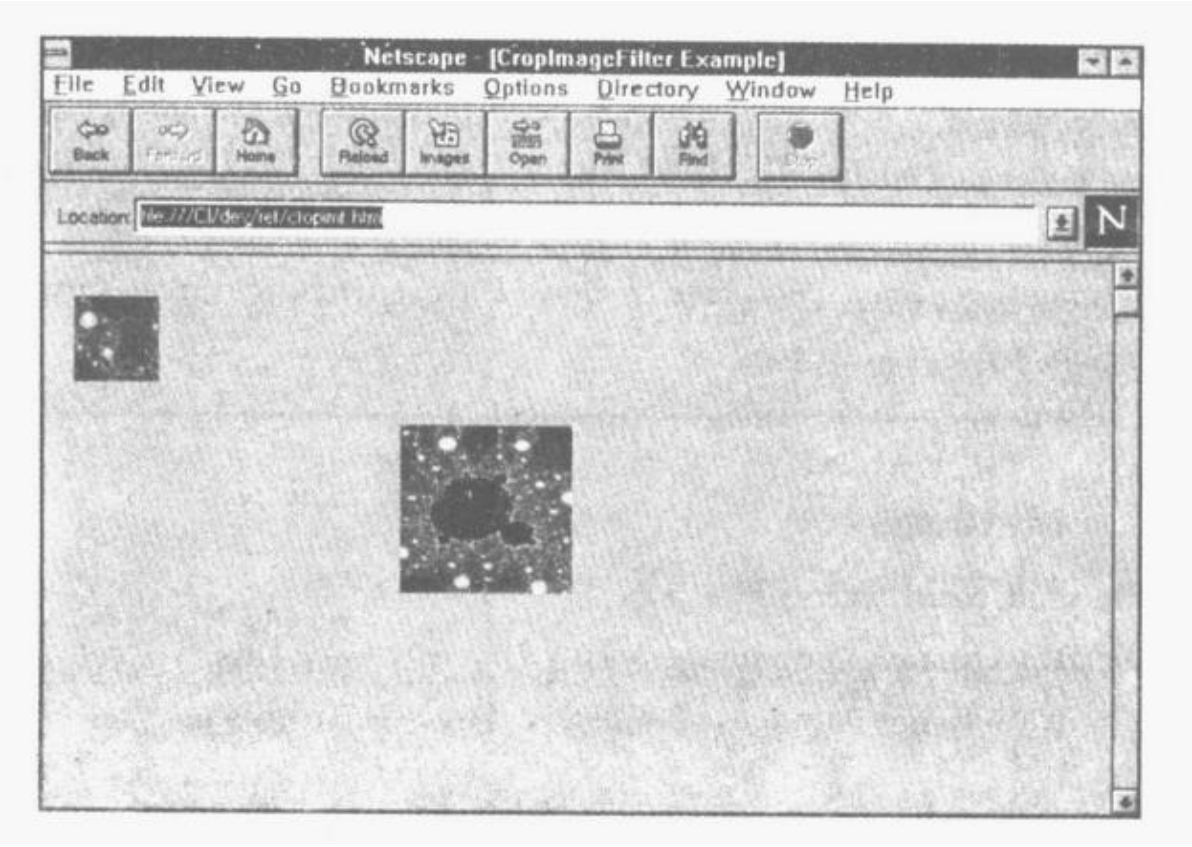


图 16 CropImageFilter 的结果

程序 9 CropImageFilter Example

```
import java.applet.Applet.*;
```

```
import java.awt. * ;
import java.awt.image. * ;

public class CropImageFilterExample extends Applet {

    Image newimage, image;

    public void init () {

        image=getImage (getCodeBase (),"fract.jpg");
        ImageFilter filter = new CropImageFilter (0, 0, 50,
50);          // set the properties for the filter

    /* The following statement creates an ImageProducer .
producer will be able to produce a new image using the source of
the old image and the specified filter */

    ImageProducer producer = new FilteredImageSource (im-
age.getSource (), filter);
    newimage=createImage (producer);
}

public void paint (Graphics g) {
    g.drawImage (newimage, 10, 10, 50, 50, this);
    g.drawImage (image, 200, 85, 100, 100, this);
    }
}
```

方法

※ CropImageFilter (int x, int y, int w, int h)

创建一个新的 CropImageFilter。指定的参数值与指定的图象

无关，但以后用过滤器裁剪图象时，这些值将被用到。

※ void setDimensions (int width, int height)

设置过滤器的大小，此过滤器是生成裁剪时所使用的。

※ void setPixels (int x, int y, int w, int h, ColorModel model, int pixels [], int offset, int scansize)

生成一个所需的图象内部所使用的方法。

※ void setPixels (int x, int y, int w, int h, ColorModel model, byte pixels [], int offset, int scansize)

生成一个所需的图象内部所使用的方法。

※ void setProperties (Hashtable props)

用来提供 ImageFilter.setProperties ( ) 一些有关裁剪区域的信息。

DirectColorModel Extends ColorModel

这个类将与机器有关的象素值转换为它们的 alpha 值，红色值、绿色值和蓝色值部件。

方法

※ DirectColorModel (int bits, int rmask, int gmask, int bmask)

创建一个新的 DirectColorModel 对象，它通过参数 bits 的值控制象素。参数 rmask、bmask 和 gmask 分别代表了 bits 指定的红色值、蓝色值和绿色值部件的位置。

※ DirectColorModel (int bits, int rmask, int gmask, int bmask, int amask)

创建一个新的 DirectColorModel 对象，它通过参数 bits 的值控制象素，参数 rmask、bmask、gmask 和 amask 分别代表了 bits 指定的红色值、蓝色值、绿色值和 alpha 值部件各自的 final int getAlpha (int pixel)

返回指定象素的 alpha 部件。

※ final int getAlphaMask ()

返回 DirectColorModel 的 alpha 屏蔽。

※ final int getBlue (int pixel)

返回指定象素的蓝色部件。

※ final int getBlueMask ()

返回 DirectColorModel 的蓝色屏蔽。

※ final int getGreen (int pixel)

返回指定象素的绿色值。

※ final int getGreenMask ()

返回 DirectColorModel 的绿色屏蔽。

※ final int getRed (int pixel)

返回指定象素的红色值。

※ final int getRedMask ()

返回 DirectlolorModel 的红色屏蔽。

※ final int getRGB (int pixel)

根据缺省的 RGB 模式获得象素的颜色值。

FilteredImageSource Extends Object Implements ImageProducer

FilterImageSource 类可以传送通过过滤器所定义的图象信息，大多数情况下，在此类中所使用的方法仅是与 java. awt. Component 类中 createImage () 方法结合而成的构造器。

▶参见程序 9 “CropImageFilter Example”

▶参见程序 10 “MyImageFilter Example”

方法

※ FilteredImageSource (ImageProducer source, ImageFilter filter)

创建一个新的 FilteredImageSource 对象。创建时由 ImageProducer 提供信息，由指定的 ImageFilt 进行过滤。

※ synchronzied void addConsumer (ImageConsumer ic)

在对象中添加一个使用者 consumer 并从 ImageProducer 得到信息。

※ synchronzied boolean isConsumer (ImageConsumer ic)

如果 ImageConsumer 是由 ImageProducer 提供信息的使用

者，则返回 true。

※ void startProduction (ImageConsumer ic)

开始向使用者传送信息。使用者 consumer 包含了指定的 ImageConsumer，但不只限于它。

※Synchronized void removeConsumer (ImageConsumer ic)

从对象列表中清除指定使用者，该对象列表是从 ImageProducer 中获得信息。

※ void requestTopDownLeftRightResend (ImageConsumer ic)

控制一个请求。而这个请求是将图象的象素以从上到下、从左到右的顺序传送给使用者 consumer。

ImageFilter Implements ImageConsumer, Cloneable

ImageFilter 类实际上不做任何事情。它从 ImageProducer 接收信息并将同样的信息发送给 ImageConsumer。如果仅仅是装入和显示图象，你就不需要用这个类。但是，ImageFilter 类为你创建自己的图象过滤器提供了一个框架：在 ImageProducer 创建图象之后以及 ImageConsumer 接收图象之前，ImageFilter 对象将对图象进行变换。

为了创建自己的图象过滤器，你必须扩充 ImageFilter 类并且至少替换其中的一个方法。六个可以让你访问图象数据的方法是：sfetColorModel ()、setDimensions ()、setHints ()、setPixels () (两个版本)、setProperties () 和 ImageComplete ()。

参见程序 9 “CropImageFilter Example”

图 17 中的两个图象来自于同一个文件。但是就像你所看到的，左上角的图象被变换并发生了移位。实现这种变换的源代码见程序 10。

程序 10　MyImageFilter Example

```
import java.applet.*;
import java.awt.*;
import java.awt.image.*;
```

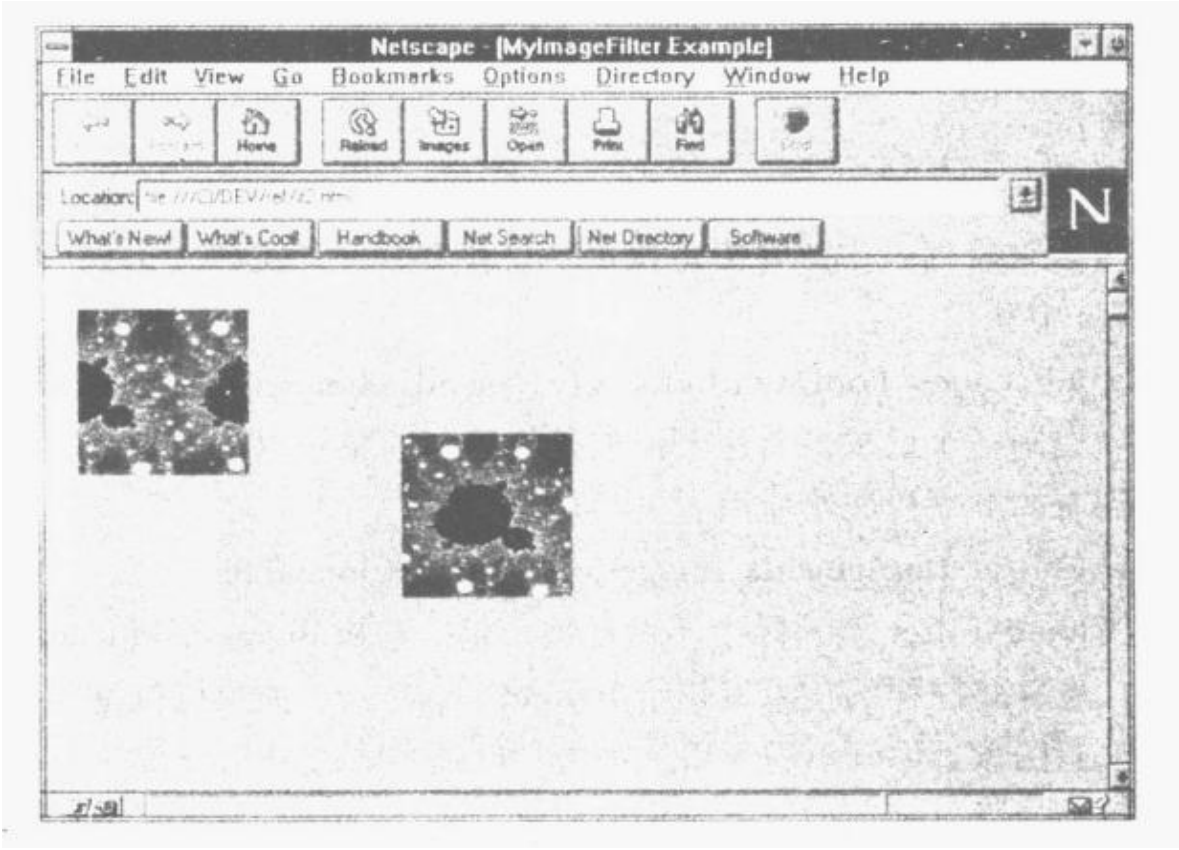


图 17　来自于一个文件的两个图像

```
public class MyFilterExample extends Applet {
    Image newimage, image;

    public void init () {
        image=getImage (getCodeBase (), "fract.jpg");
        ImageFilter filter=new MyImageFilter (100);
        ImageProducer producer=new FilteredImageSource (im-
age.getSource (), filter);
    newImage=createImage (producer);
}

public void paint (Graphics g) {
        g.drawimage (newimage, 10, 10, 100, 100, this);
        g.drawimage (image, 200, 85, 100, 100, this);
```

```
    }
}
    public class MyImageFilter extends ImageFilter {
        private int width;
        private int height;

        public MyImageFilter (int translation) {
          shift=translation;
        }

    public void setDimensions (int width, int height) {
         consumer.setDimensions ( width, height);
         this.width=width;
    }
```

/ * This implementation of setPixels () shifts all colums over by the value of the shilft field . All colums that are pushed off the screen (to the left) are displayed on the right.

To do this , the method scrolls through each of colum on the screen . If the colum still fits once shifted , it is displayed in its new position . If it is pushed off the screen , it is wrapped around relative to its original position. * /

```
    public void setPixels (int x, int y, int w, int h, ColorModel
model, byte pixels [], int off, int scansize) {
        for (int line=x; line< (x+w); line++) {
          if ( (line+shift) <=width)
            consumer.setPixels (line, y, 1, h, model, pixels,
off+shift, scansize);
          else
```

```
            consumer.setPixels (line, y, 1, h, model, pixels,
off-width+shift, scansize);
        }
    }
    /* Shifts all colums over by the value of the shift field . All
colums that are pushed off the screen (to the left) are displayed
on the right */
    public void setPixels (int x, int y, int w, int h, ColorModel
model, int pixels [], int off, int scansize) {
    for (int line=x; line< (x+w); line++) {
        if ( (line+shift) <=width)
            consumer.setPixels (line, y, 1, h, model, pixels,
off+shift, scansize);
        else
            consumer.setPixels (line, y, 1, h, model, pixels,
off-width+shift, scansize);
        }
    }
}
```

读者将注意到这个图象过滤器只用到了六个方法中的三个。通常情况下，最重要的方法是 setPixels ()，它分配给每个象素一个位置。其他的方法也很有用，只是不像 setPixels () 重要而已。

注意：象素位移的变换只是把它们放在包含图象数据的数组 pixels [] 中不同的位置上。变换的规则是：象素 (u, v) 放在 pixels [v * scansize+u+off] 中。如果你想将 (u, v) 变换为 (v, u)，只需在表 10 中替换下面的语句：

```
consumer.setPixels (line, y, 1, h, model, pixels, off,
scansize);
```

在这个例子中，是将象素在 x 轴上进行变换。如果想使每个象素左移，就使用下面的语句：

consumer.setPixels(line, y, l, h, model, pixels, off +shift, scansize);

字段

• protected ImageComsumer consumer

ImageConsumer 接收从过滤器来的信息。

方法

※ ImageFilter ()

创建一个新的 ImageFilter。

※ Object clone ()

生成一个 ImageFilter 的拷贝。

※ ImageFilter getFilterInstance (ImageConsumer ic)

返回一个 ImageComFilter 的拷贝，它将向 ImageConsumer 提供信息。

※ void ImageComplete (int status)

在向 ImageConsumer 传送信息之前从 ImageProducer 接收信息。缺省情况下，这个方法仅仅是传递信息，但它可以被修改去做更多的事情。

※ void resendTopDownLeftRight (ImageProducer ip)

请求 ImagProducer 以从上到下，从左到右的顺序发送有关图象的象素信息。

※ void setColorModel (ColorModel model)

从 ImageConsumer 接口的 setColorModel ( ) 方法中接收信息，并让用户使用这些信息。

※ void setDimensions (int width, int height)

从 ImageConsumer 接口的 setDimensions () 中接收信息。

※ void setHints (int hints)

从 ImageConsumer 接口的 setDimensions () 中接收信息。

※ void setPixels (int x, int y, int width, int height, ColorModel model, int pixels [], int offset, int scansize)

用来控制图象的象素。它是这个类中很有用的一个方法。

※ void setPixels (int x, int y, int w, int h, ColorModel model, int pixels [], int off, int scansize)

用来控制图象的象素。它是这个类中有用的一个方法。

※ void setProperties (Hashtable props)

对于某过滤器，在向 ImageConsumer 传递它的属性信息之前，再加入一些信息。

IndexColorModel Extends ColorModel

这个类用来为一组颜色建立一个检索表。这个表包含了每个颜色部件的信息。

方法

※ IndexColorModel (int bits, int size, byte r [], byte g [], byte b[])

建立一个新的指定了大小(参数 size)的 IndexColorModel。参数 bits 代表了每个象素的位数。r []、g [] 和 b [] 是红色值、绿色值和蓝色值构成的数组。

※ IndexColorModel (int bits, int size, byte r [], byte g [], byte b [], int trans)

建立一个新的指定了大小(参数 size)的 IndexColorModel。参数 bits 代表了每个象素的位数。r []、g [] 和 b [] 是红色值、绿色值和蓝色值构成的数组。参数 trans 代表了透明颜色存储的索引。

※ IndexColorModel (int bits, int size, byte r [], byte g [], byte b [], byte a [])

建立一个新的指定了大小(参数 size)的 IndexColorModel。参数 bits 代表了每个象素的位数。r []、g []、b [] 和 a [] 是红色值、绿色值、蓝色值和 alpha 值构成的数组。

※ final int getAlpha (int index)

返回有指定索引的象素的 alpha 值。

※ final void getAlphas (byte a [])

返回存储 alpha 值的数组。

※ final int getBlue (int index)

返回有指定索引点的象素的蓝色值。

※ final void getblues (byte b [])

返回存储蓝色值的数组。

※ final int getGreen (int index)

返回有指定索引点的象素的绿色值。

※ final void getGreens (byte g [])

返回存储绿色值的数组。

※ final int getMapSize ( )

返回每个数组中值的数目。

※ final int getRed (int index)

返回有指定索引点的象色素的红色值。

※ final int getMapSize ()

返回存储红色值的数组

※ final int getRGB (int index)

基于当前 RGB 调色板返回一个整数，它代表了指定索引点的象素值。

※ final int getTransparentPixel ()

返回透明象素的索引点。如果没有定义，则返回－1。

MemoryImageSource Implements ImageProducer

Memory ImageProducer 可以让用户建立自己的图象。把“图片”以整数或者字节的形式存在数组中之后，用户就可以从数组中装入图象，而不是从文件中。

方法

※ MemoryImageSource (int w, int h, ColorModel cm, byte pix [], int offset, int scansize)

创建一个指定了大小和 ColorModel 的 MemoryImageSource 对象。参数 pix [] 存储了图象数据。参数 offset 代表了图象数据开始处的索引。参数 scansize 代表了每行在数组中所占的字节数。

※ MemoryImageSource (int w, int h, ColorModel cm, byte pix

[], int offset, int scan, Hashtable props)

创建一个指定了大小和 ColorModel 的 MemoryImageSource 对象。参数 pix [] 存储了图象数据。参数 offset 代表了图象数据开始处的索引。参数 scansize 代表了每行在数组中所占的字节数。参数 props 是所生成的图象的一个 Hashtable 特性。

※ MemoryImageSource (int w, int h, ColorModel cm, int pix [], int offset, int scan)

创建一个指定了大小和 ColorModel 的 MemoryImageSourced 对象。参数 pix [] 存储了图象数据。参数 offset 代表了图象数据开始处的索引。参数 scan 代表了每行在数组中所占的字节数。

※ MemoryImageSource (int w, int h, ColorModel cm, int pix [], int offset, int scan, Hashtable props)

创建一个指定了大小和 ColorModel 的 MemoryImageSource 对象。参数 pix [] 存储了图象数据。参数 offset 代表了图象数据开始处的索引。参数 scansize 代表了每行在数组中所占的字节数。参数 props 是所生成的图象的一个 Hashtable 特性。

※ MemoryImageSource (int w, int h, int pix [], int offset, int scan)

在缺省的 RGB 调色板下，创建一个指定了大小的 MemoryImageSource 对象。参数 pix [] 存储了图象数据。参数 offset 代表了图象数据开始处的索引。参数 scan 代表了每行在数组中所占的字节数。

※ MemoryImageSource (int w, int h, int pix [], int offset, int scan, Hashtable props)

根据缺省的 RGB 调色板，创建一个指定了大小的 MemoryImageSource 对象。参数 pix [] 存储了图象数据。参数 offset 代表了图象数据开始处的索引。参数 scan 代表了每行在数组中所占的字节数。参数 props 是所生成的图象的一个 Hashtable 特性。

※ synchronzied void addConsumer (ImageConsumer ic)

在检索 ImageProducer 信息的对象表列中添加一个使用者。

※ synchronzied boolean isConsumer (ImageConsumer ic)

如果 ImageConsumer 从当前 ImageProducer 中接收信息，则返回 true。

※ void startProduction (ImageConsumer ic)

开始向使用者 consumer 传递信息的过程。使用者 consumer 不仅仅限于指定的 ImageConsumer 本身。

※ synchronzied void removeConsumer (ImageConsumer ic)

在一个从 ImageProducer 中得到信息的对象表中移走指定的一个使用者 consumer。

※ void requestTopDownLeftRightResend (ImageConsumer ic)

处理以从上到下、从左到右的顺序发送图象的象素信息的使用者请求。

PixelGrabber Implements ImageConsumer

PixelGrabber 类是 ImageConsumer 的一个特殊的类型。它的设计是用来获取图象的一个矩形框中的象素。注意，可以从一个已装入的图象也可以从一个 ImageProducer 中获取象素。

为了获取象素，首先要创建一个象素的获取工具，同时还要确定一个矩形框和一个存储象素的数组。获取象素时，要调用 grabPixels () 方法。

方法

※ PixelGrabber (Image img, int x. int y, int w, int h, int pix [], int offset, int scansize)

创建一个新的 PixelGraber 对象，从图象中获取指定的象素。获取的象素根据参数 offset 和 scansize 的指定值将被放在数组 pix [] 中。

※ PixelGrabber (ImageProducer ip, int x, int y, int w, int h, int pix [], int offset, int scansize)

创建一个新的 PixelGrabber 对象，以从图象中获取指定的象素，获取的象素通过指定的参数 offset 值和 scansize 的值被放在

数组 pix [] 中。

※ synchronzied boolean grabPixels (long ms) throws InterruptedException

在请求获取象素时锁住 PixelGrabber 对象，如果锁定成功或者超过指定时间后，该方法才返回。

※ boolean grabPixels () throws InterruptedException

请求处理预先指定的图象的象素。

※ void setColorModel (ColorModel model)

请求执行 ImageConsumer 接口。但是在缺省状态下，不做任何事情。

※ void setDimension (int width, int height)

请求执行 ImageConsumer 接口。但是在缺省状态下，不做任何事情。

※ void setHints (int hints)

请求执行 ImageConsumer 接口。但是在缺省状态下，不做任何事情。

※ void setPixels (int srcX, int srcY, int srcW, int srcH, ColorModel model, byte pixels [], int srcOff, int srcScan)

PixelGrabber 类使用这个方法把预期的象素放在数组中，这个数组是在构造器 (constructor) 方法或 getPixels () 方法中指定的。

※ void setPixels (int srcX, int srcY, int srcW, int srcH, ColorModel model, int pixels [], int srcOff, int srcScan)

PixelGrabber 类使用这个方法把预期的象素放在数组中，这个数组是在构造器方法中指定的。

※ void setProperties (Hashtable props)

请求执行 ImageConsumer 接口。但是在缺省状态下，不做任何事情。

※ synchronzied int status ()

返回在 PixelGrabber 生存期中所产生的任何标志。

※ synchronzied void imageComplete (int status)

修改内部标志 flags 字段，以反映指定的状态 status。这些标志可能是从 status () 方法返回的。

Abstract RGBImageFilter Extends ImageFilter

RGBImageFilter 类提供一个很简单的方法来操纵图象的显示，这是通过改变象素颜色实现的。若要使用 RGBImageFilter 类，必须放弃旧类并创建一个新类并且定义 filterRGB () 方法。这个方法可以通过控制单个部件的颜色来调整象素的显示。如图 18。

正如你所看到的，图中左上角的图象的表面特征被改变了。这个工作通过程序 11 中的代码完成。在 RGBImageFilter 类中，最重要的方法是 filterRGB () 方法。通过在 MyRGBImageFilter 类中使用这个方法，不但使 MyRGBImageFilter 类不再是抽象类，而且还实际执行了改变象素值的具体操作。

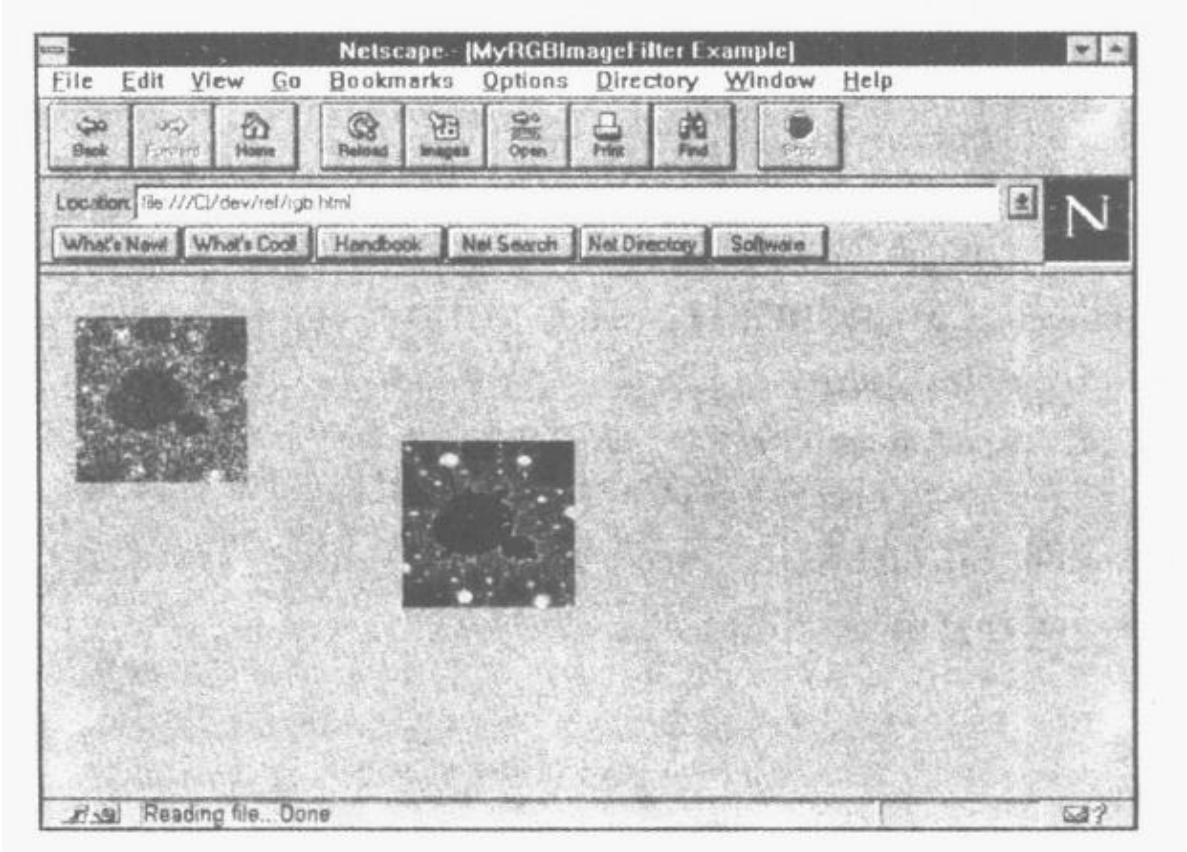


图 18　由 MyRGBImageFilter 生成的图象

程序 11　MyRGBImageFilter Example

```
public class ImageFilter extends Applet {

  Image newimage, image;

  public void init () {

      image=getImage (getCodeBase (), "frag.jpg");
      ImageFilter filter=new MYRGBFilter ();
      ImageProducer producer=new FilterImageSource (image.
fetSource (), filter);
  newimage=createImage (producer);
}

public void paint (Graphics g) {
   g.drawImage (newimage, 10, 10, 100, 100, this);
   g.drawimage (image, 200, 85, 100, 100, this);
}
}
public class MYRGBFilter extends RGBImageFilter {
   private int width;
   private int height;

   public int filterRGB (int x, int y, int rgb) {
    return (rgb>>1);
   }
}
```

字段

· protected boolean canFilterIndexColorModel

如果返回值为 true，那么滤波器将从 IndexColorModel 表中

处理象素值。如果你的 filterRGB() 方法不依赖于象素的坐标，则这个字段应为 true。

• protected ColorModel newmodel

通过 substituteColorModel () 方法来设置。

• protected ColorModel origmodel

通过 substituteColorModel () 方法来设置。

方法

※ RGBImageFilter ()

创建一个新的 RGBImageFilter。

※ indexColorModel filterIndexColorModel (indexColorMolel icm)

过滤指定的 IndexColorModel 表，返回模板 model 修改后的类型。

※ void filterRGBPixels (int x, int y, int w, int h, int pixels [], int offset, int scansize)

通过 filterRGB () 方法传递每个象素，从而过滤数组 pixels [] 中的一部分数值。参数 offset 定义了被过滤的数值开始处的变址。参数 x 和 y 定义了象素的坐标。参数 w 和 h 定义了被过滤区域的大小。参数 scansize 定义了每行象素的数目。

※ void setColorModel (ColorMoldel model)

如果过滤器能够控制 IndexColorModels 并且指定的 ColorModel 就是一个 IndexColorModel，那么这个方法无论何时遇到 setPixels () 方法时，都将用过滤后的类型代替指定的模板 model。

※ void substituteColorModel (ColorModel oldcm, ColorModel newcm)

无论何时遇到 setPixels () 方法时，都将用新的类型 newcm 代替旧的类型 oldcm。

※ void setPixels (int x, int y, int width, int height, ColorModel model, int pixels [], int offset, int scansize)

在首次用 filterRGB() 方法进行过滤之后，将象素传递给 ImageConsumer。

※ void setPixels (int x, int y, int width, int height, ColorModel model, byte pixels [], int offset, int scansize)

在首次用 filterRGB() 方法进行过滤之后，将象素传递给 ImageConsumer。

※ abstract int filterRGB (int x, int y, int rgbpixel)

这个方法是这个类中最重要的。它必须被超越从而创建一个有具体功能的 RGBColorModel。为了改变在（x，y）处显示的指定的 rgbpixel 的表面特征，要创建和返回一个新的象素值。如果参数 x 和 y 的值为－1，则给定的象素将从 IndexColorModel 中得到。

2. 接口

ImageConsumer

ImageConsumer 接口定义了通过一些类装入图象时所要做的动作。除了定义这些类的动作以外，它还被用作一个参考型变量，以引用下面所列的字段和方法，这些字段和方法须在 ImageConsumer 类中定义。

字段

- final static int COMPLETESCANLINES

  传送给 setHints()的一个值，它表明了象素将整行地被传送。

- final static int IMAGEABORTED

  传送给 imageComplete () 的一个状态值，它表明退出图象的生成过程。

- final static int IMAGEERROR

  传送给 imageComplete () 的一个值，它表明生成图象时发生错误。

- final static int RANDOMPIXELEXPORDER

  传送给 setHints () 的一个值，它表明了象素将以随机的顺序传送。

- final static int SINGLEFRAME

  传送给 setHints () 的一个值，它表明了图象包含了一单个的

frame。

• final static int SINGLEFRAMEDONE

传送给 imageComplete () 的一个状态值，它表明了一个图象的 frame 已经被完全传送出去，但有更多的 frame 将要传送。

• final static int SINGLEPASS

传送给 setHints () 的一个值，它表明了图象已经被完全传送出去。

• final static int STATICIMAGEDONE

传送给 imageComplete () 的一个值，它表明了整个图象已经被完全传送。

• fianl static int TOPDOWNLEFTRIGHT

传送给 setHints () 的一个值，它表明了图象将以从上到下、从左到右的顺序传送。

方法

※ abstract void imageComplete (int status)

被 ImageProducer 所调用，它表明了当前的图象操作已经完成。参数 status 的值将提供 ImageConsumer 一些更详细的信息。

※ abstract void setColorModel (ColorModel model)

对于传送给某 ImageConsumer 的象素，由 ImageProducer 调用这个方法来设置缺省的调色板 ColorModel。

※ abstract void setDimensions (int width, int height)

由 ImageProducer 调用这个方法来设置被传送图象的大小。

※ abstract void setHints (int hintflags)

由 ImageProducer 调用这个方法向 ImageConsumer 提供有关被传送图象属性的信息。

※ abstract void setPixels (int x, int y, int w, int h, ColorModel model, byte pixels [], int offset, int scansize)

由 ImageProducer 调用这个方法来设置象素的值。

※ abstract void setPixels (int x, int y, int w, int h, ColorModel model, int pixels [], int offset, int scansize)

由 ImageProducer 调用这个方法来设置象素的值。

ImageObserver

Imageobserve 接口定义了通过一些类处理图象时所要做的动作，并且提供了对图象的异步修改。这样是非常必要的，因为第一次使用图象时，它的信息不一定都有用。在像 Graphics.drawImage () 这样的方法中说明 ImageObserver，将使得图象到达后，对象 object 才接收它的修改信息。

字段

• final static int ABORT

传送给 imageUpdate () 的一个值，它表明了退出图象的装入过程。

• final static int ALLBITS

传送给 imageUpdate () 的一个值，它表明了前面所画的一个图象现在已经完成。

• final static int ERROR

传送给 imageUpdate () 的一个值，它表明了装入图象时发生错误。

• final static int FRAMEBITS

传送给 imageUpdate () 的一个值，它表明了图象的一个完整的 frame 已经被接收了。

• final static int HEIGHT

传送给 imageUpdate () 的一个值，它表明了从 imageUpdate () 方法中的高度参数内不能得到图象的高度值。

• final static int PROPERTIES

传送给 imageUpdate () 的一个值，它表明了从 imageUpdate () 方法中不能得到图象的属性值。

• final static int SOMEBITS

传送给 imageUpdate () 的一个值，它表明了图象的许多象素是可利用的。

• final static int WIDTH

传送给 imageUpdate () 的一个值，它表明了从 imageUpdate () 方法中的宽度参数里不能得到图象的宽度值。

方法

※ abstract boolean imageUpdate (Image img, int infoflags, int x, int y, int width)

对于一个图象，当它的大多数信息可以利用时，将调用这个方法。参数 x，y，width 和 height 的值依赖于参数 infoflags 的值。如果需要更多地修改，则返回 true；如果接收了所有需要的信息，则返回 false。

ImageProducer

ImageProducer 类定义了通过一些类（如：FilteredImageSource 或 MemoryImageSource）在创建图象时所要做的一般动作。

▶参见程序 9 "CropImageFilter Example"

▶参见程序 11 "MyRGBImageFilter Example"

方法

※ abstract void addConsumer (ImageConsumer ic)

当使用者对 ImageProducer 所提供的信息感兴趣时，记录指定的 ImageConsumer。

※ abstract boolean isConsumer (ImageConsumer ic)

当指定的 ImageConsumer 是这个 ImageProducer 对象所登记的一个使用者时，返回 true。

※ abstract void removeConsumer (ImageConsumer ic)

从使用者表中移走指定的 ImageConsumer。一旦接收到的所需信息阻止使用者接收更多的信息拷贝时，应该调用这个方法。

※ abstract void requestTopDownLeftRightResend (ImageConsumer ic)

请求 ImageProducer 以从上到下、从左到右的顺序传送象素。

※ abstract void startProduction (ImageConsumer ic)

在使用者表中加入指定的 ImageConsumer 并且向所有登记的使用者传送象素信息。

四、JAVA.IO

java.io 是 Java 语言的标准输入/输出库。它提供了一系列的类型(types)，从简单的字符串输入缓冲 StringBufferInputStream 到复杂的随机访问文件 RandomAccessFile，这个软件包使一个直观的无限数目的通信成为可能。

java.io 软件包主要由两种类型的类组成：建立流和管理流的类。下表简要地说明了这些类在模块中的位置。

建立流的类	操纵流的类
ByteArrayInputStream	BufferedInputStream
ByteArrayOutputStream	BufferedOutputStream
FileInputStream	DataInputStream
FileOutputStream	DataOutputStream
PipedInputStream	FilterInputStream
PipedOutputStream	FilterOutputStream
StringBufferInputStream	LineNumberInputStream
	PrintStream
	PushbackInputStream
	RandomAccessFile
	SequenceInputStream
	StreamTokenizer
	StringBufferInputStream

一般来讲，一个流的管理器的实例要以这样的语法来建立：

StreamMaanagerType instanceofManager = new StreamManagerType (StreamCreatorType)；

这里 StreamManagerType 和 StreamCreatorType 都是类的名字。

1. 类

BufferedInputStream Extends FilterInputStream

这个缓冲流型的类提供给用户一种从 InputStream（输入流）读取信息的更有效的途径。不是让信息堆积在流中等待用户去读取它，而是在每次调用 read（）方法时，BufferedInputStream 会读入流中所有的数据并将它放在缓冲区内。也就是说，后面的信息会来自这个缓冲区，而不是输入流，从而节省了用户的时间。

字段

- protected byte buf []

  数据存储的缓冲区。

- protected int count

  缓冲区中的字节数目。

- protected int marklimit

  在 BufferedInputStream 不能跟踪用户的标记之前可以访问的字节数目。

- protected int markpos

  标记位置。

- protected int pos

  缓冲区中当前的位置。

方法

※ BufferedInputStream（InputStream in）

建立一个基于指定的输入流的新的 BufferedInputStream。

※ BufferedInputStream（InputStream in，int size）

建立一个基于指定的输入流的新的 BufferedInputStream，缓冲区大小已指定。

※ synchronized int available（）throws IOException

返回当前可被读取的有效字节数。该值包括缓冲区中未读取的字节数及 InputStream 输入流中的有效字节数。

※ synchronized void mark（int readlimit）

标记流中的当前位置。通过调用方法 reset（），可以返回到这一点再次读取同样的数据。readlimit 设置了调用 reset（）之前可被读取的字节数的边界。虽然 readlimit 越大就有越多的灵活性，但也会造成存储缓冲区的增大。

※ boolean markSupported（）

由于这是一个 FilterInputStream，这种方法返回一个布尔值指示该类是否支持标记。对于该类，永远返回 true。

※ synchronized int read（byte b []，int offset，int len）throws IOException

填充存储缓冲区，这个方法将数组 b[]中从 offset 开始的 len 个字节放入缓冲区。

※ synchronized int read（）throws IOException

读一个字节的数据。

※ synchronized void reset（）throws IOExcepton

返回最后一个标记位置，使用户能够从这一位置读取。注意这将失去当前的位置。如果超过了 marklimit（由 mark（）指定）或者没有指定标记，将出现一个 IOException。

※ synchronized long skip（long num）throws IOException

跳过流中指定数目的字节。

BufferedOutputStream Extends FilterOutputStream

BufferedOutputStream 是输出流 OutputStream 的管理者，它使用户能够更有效地往流中写数据。在调用 write（）方法时，不是每次写入一点，BufferedOutputStream 将信息写入一个临时的缓冲区，当缓冲区已满或者调用 flush（）方法时，这个信息才写入输出流中。

注意：在关闭 OutputStream 之前，用户要确认已调用了 flush（）方法！否则，信息将被冻结——置于缓冲区中永不被发送出去。

字段

- protected byte buf []

  临时存储缓冲区。

• protected int count

缓冲区中当前的字节数。

方法

※ BufferOutputStream (OutputStream out)

建立一个使用指定的 OutputStream、新的 BufferedOutputStream。

※ BufferOutputStream (OutputStream out, int size)

建立一个使用指定的 OutputStream、新的 BufferedOutputStream。size 指定了缓冲区的大小。

※ synchronized void flush () throws IOException

清空缓冲区，将缓冲区中所有的字节写入 OutputStream。

※ synchronized void write (byte b [], int offset, int len) throws IOException

将指定的字节写入缓冲区。被写入的数据从 b [offset] 开始包括 len 个字节。如果缓冲区实际不能容纳指定数目的字节，则缓冲区和指定的字节都将被写入 OutputStream。

※ synchronized void write (int b) throws IOException

将一个字节写入缓冲区。如果缓冲区没有空间，缓冲区的内容将先被送入输出流。指定的字节将被放置在缓冲区中。

ByteArrayInputStream Extends InputStream

ByteArrayInputStream 是 InputStream 中的一种类，可以同其他的 InputStream 类一样来操作。不同的是，ByteArrayInputStream 不是从标准流中接收数据，而是从已经建立好的数组中接收。

当建立起了一个 ByteArrayInputStream，必须指定用户打算用到的缓冲区内容。当调用读命令时，InputStream 将从这一缓冲区读取。

字段

• protected byte buf []

被读取数据所在的缓冲区。注意这不是在构造方法中指定的

数组的拷贝，而是指定数组的一个参照(refrence 很像 C 语言的指针)。

- protected int count

  缓冲区中可以被读取的最大字节数目。

- protected int pos

  缓冲区中当前的位置。

方法

※ ByteArrayInputStream (byte buf [])

建立一个将从指定缓冲区读取数据、新的 ByteArrayInputStream。当前的位置 (pos) 置为 0，可被读取的最大字节数(count)被置为数组的长度。

※ ByteArrayInputStream (byte buf [], int offset, int length)

建立一个将从指定数组读取数据、新的 ByteArrayInputStream。pos 和 count 分别被设定为 offset 和 length。

※ synchronized int available ()

返回可被读取的字节数，等于字节总数减掉当前位置(count-pos)。

※ synchronized int read (bute b [], int offset, int len)

从数组输入流中读取 len 个字节，从 offset 位置开始写入数组 b []。

※ synchronized int read ()

从缓冲区中读取一个字节。

※ synchronized int reset ()

将当前位置设为输入缓冲区的开始。

※ synchronized long skip (long num)

在缓冲区中跳过指定的字节数。

ByteArrayOutputStream Extends OutputStream

ByteArrayOutputStream 是一种特定的可被用来聚集数据的输出流 OutputStream。ByteArrayOutputStream 不是直接写进流中，而是写入临时缓冲区，这个缓冲区将在必要时自动增长以足

够容纳用户的数据。

将数据写入缓冲区之后，可以：

• 用 WriteTo () 方法将整个缓冲区内容写入 OutputStream；

• 检索缓冲区中的所有数据，用 toByteArray () 方法将它存入一个新数组；

• 用 toString () 方法将缓冲区内容转化为字符串。

字段

• protected byte buf []

用来存储数据的临时缓冲区。

• protected int count

缓冲区中的字节数。

方法

※ ByteArrayOutputStream ()

建立一个 32 字节容量的、新的 ByteArrayOutputStream。

※ ByteArrayOutputStream (int size)

建立一个指定容量的、新的 ByteArrayOutputStream。

※ synchronized byte [] toByteArray ()

返回一份缓冲区的备份。

※ synchronized void reset ()（原文无注解，译者注）

※ int size ()

返回存储在缓冲区的字节数目。

※ String toString ()

以字符串形式返回缓冲区内容（下一个方法 hibyte 设为 0 时，与此等价）。

※ String toString (int hibyte)

以字符串形式返回缓冲区内容，每个字符的高八位设为 hibyte。

※ synchronized void write (int b)

将指定的字节写入临时缓冲区，必要时增加缓冲区的大小。

※ synchronized void write (byte b [], int offset, int len)

将指定的字节数组写入临时缓冲区，必要时增加缓冲区的大小。

※ synchronized void writeTo (OutputStream out) throws IOException

将整个缓冲区写到指定的 OutputStream。

DataInputStream Extends FilterInputStream Implements DataInput

DataInputStream 使用户能够从流中读取基本类型的数据。例如，读取字符串时，不是从流中以字节为单位来读取，而是运用 readLine () 读取，直到遇到新行的字符为止。

▶参见“DataOutputStream Extends FilterOutputStream Implements DataOutput”

方法

※ DataInputStream (InputStream in)

建立一个新的 DataInputStream 操作指定的 InputStream。

※ final int read(byte b[],int offset,int len) throws IOException

从流中读取 len 个字节，从 b [offset] 位置开始写入 b []。

※ final int read (byte b []) throws IOException

从流中读取数据，写到指定的缓冲区中，返回读取的字节数。

※ final boolean readBoolean () throws IOException

从流中读取下一个字节，假定该字节为布尔型值。

※ final byte readByte () throws IOException

从流中读取并返回一个单字节。

※ final char readChar () throws IOException

从流中读取并返回下两个字节组成的 16 位字符 chur。

※ final double readDouble () throws IOException

从流中读取并返回下八个字节组成的 64 位双精度数。

※ final float readFloat () throws IOException

从流中读取并返回下四个字节组成的 32 位浮点数。

※ final void readFully (byte b [], int offset, int len) throws

IOException

从流中读取数据，直到从指定数组的 offset 开始 len 长度的区间填满新数据时返回。

※ final void readFully (byte b []) throws IOException

从流中读取数据，直到整个缓冲区内填满新数据时返回。

※ final int readInt () throws IOException

从流中读取并返回下四个字节组成的 32 位整数。

※ final String readLine () throws IOException

从流中读取字符直到遇到\n，\r，\r\n 或 EOF。然后返回整个字符串。

※ final long readLong () throws IOException

从流中读取并返回下八个字节组成的 64 位长整数。

※ final short readShort () throws IOException

从流中读取并返回下四个字节组成的 32 位短整数。

※ final int readUnsignedByte () throws IOException

从流中读取并返回一个字节。

※ final int readUnsignedShort () throws IOException

从流中读取并返回下两个字节组成的 16 位无符号短整数。

※ final String readUTF () throws IOException

从流中读取并返回一个 UTF 字符串。

※ final static String readUTF (DataInput in) throws IOException

从指定的 DataInput 中读取并返回一个 UTF 字符串。

※ final int skipBytes (int num) throws IOException

跳过指定数目的字节。

DataOutputStream Extends FilterOutputStream Implements DataOutput

这是和 DataInputStream 相对应的类，DataOutputStream 类可以使用户以基本数据类型的形式往流中写数据，如字符或双精度数，而不是一系列的字节。

▶参见“DataInputStream Extends FilterInputStream Implements DataInput”

字段

- protected int written

  写到流中去的字节数。

方法

※ DataOutputStream (OutputStream out)

建立一个新的 DataOutputStream 操作指定的 OutputStream。

※ void flush () throws IOException

刷新流。这是处理缓冲区溢出的唯一方法。

※ final int size() 返回写入缓冲区的字节数——即 written 字段。

※ synchronized void write (byte b[], int offset, int len) throws IOException

将由 len 指定数目的字节写入输出流。这些字节将从指定数组 b [] 的 offset 索引开始。

※ synchronized void write (int b) throw IOException

将一个字节写入输出流。

※ final void writeBoolean (boolean v) throw IOException

将一个布尔数写入输出流。

※ final void writeByte (int v) throw IOException

将一个布尔数写入输出流——调用 write ()。

※ final void writeBytes (String s) throw IOException

将一个字符串以字节为单位写入输出流。

※ final void writeChar (int v) throw IOException

将一个字符写入输出流。

※ final void writeChars (String s) throw IOException

将一个字符串以字符为单位写入输出流。

※ final void writeDouble (double v) throw IOException

将一个双精度数写入输出流。

※ final void writeFloat (float v) throw IOException

将一个浮点数写入输出流。

※ final void writeInt (int v) throws IOException

将一个整型数写入输出流。

※ final void writeLong (long v) throws IOException

将一个长整型数写入输出流。

※ final void writeShort (int v) throws IOException

将一个短整型数写入输出流。

※ final void writeVTF (String str) throws IOException

将一个字符串以 UTF 格式写入输出流。

File

File 类提供一种对文件的基本管理和操纵手段。虽然它不能使用户直接访问包含在文件中的数据信息，但它的确能使用户进行“housekeeping”操作，比如建立目录、查询文件长度或者比较两个独立文件的版本。由于它所提供操作的性质，该类很大程度上依赖于各自的方法。

▶参见“DataInputStream Extends FilterInputStream Implements DataInput”

▶参见“DataOutputStream Extends FilterOutputStream Implements DataOutput”

▶参见“FileDialog Extends Dialog”

注意：虽然 Java 关于该类的测试表明没有问题，但该类中方法的执行涉及到安全问题。也就是说，当前用户编译的程序，也许不能同时在以 applet 的方式在 applet 浏览器和 Netscape 中运行代码来完成同样的工作。

字段

- final static String pathSeparator

依赖于系统的路径分界符（在基于 Windows 的机器中为“;”）。

- final static char pathSeparatorChar

依赖于系统的路径分界符，以单字符方式表示。

- final static String separator

依赖于系统的文件分界符（在基于 Windows 的机器中为“\”）。

- final static char separatorChar

依赖于系统的文件分界符，以单字符方式表示。

方法

※ File（String path）

建立一个新的基于指定路径的 File 对象。

※ File（File dir，String name）

建立一个新的在指定路径中有特定文件名的 File 对象。路径由另一个 File 对象指定。

※ File（String path，String name）

建立一个新的在指定路径 path 中有特定文件名的 File 对象。

※ boolean canRead（）

如果程序可以从指定文件中读，返回 true。

※ boolean canWrite（）

如果程序可以往指定文件中写，返回 true。

※ boolean delete（）

删除文件。成功返回 true。

※ boolean equals（Object anotherObject）

将 File 和另外一个对象相比较，如果另一对象是一个有相同的路径的 File 对象，返回 true。

※ boolean exists（）

如果文件存在返回 true。

※ String getAbsolutePath（）

返回文件的绝对路径。

※ String getName（）

返回文件名——最后一个分界符后面的字符串。

※ String getParent（）

返回父目录的名字——除文件名以外的所有东西。

※ String getPath ()

返回整个路径，包括父目录及文件名。

※ int hashCode ()

返回 File 对象的散列码。这个值基于路径字符串。

※ boolean isAbsolute ()

如果路径为绝对路径返回 true。

※ boolean isDirectory ()

如果目录文件存在返回 true。

※ boolean isFile ()

如果路径中存在文件返回 true。

※ long lastModified ()

返回最后修改日期。返回值没有意义或仅用于比较两个文件。

※ long length ()

返回文件的长度。

※ String [] list ()

返回当前路径中的文件名数组。

※ String [] list (FilenameFilter filter)

返回当前路径中符合过滤器要求的文件名数组。

※ boolean mkdir ()

建立当前的目录，成功返回 true。

※ boolean mkdirs ()

建立当前的目录路径，成功返回 true。

※ boolean renameTo (File newName)

修改当前文件名。

※ String toString ()

返回文件的路径。

FileDescriptor

FileDescriptor 是用于 Java 类内部处理文件的类。这个类封装基于机器的文件视图——把每一个文件描述为一个整数。

FileDescriptor 对象的用处可以在 FileInputStream 和 FiltOutputStream 中得以体现。由于该类的特性所致，所有的字段和方法都是基于本地属性的。

字段

- final static FileDescriptor err
  标准错误输出文件的描述符。
- final static FileDescriptor in
  标准输入文件的描述符。
- final static FileDescriptor out
  标准输出文件的描述符。

方法

※ FileDescriptor ()
  建立一个新的文件描述符 FileDescriptor。

※ boolean valid ()
  如果文件描述符 FileDescriptor 有效则返回 true。

FileInputStream Extends InputStream

FileInputStream，顾名思义，是一个特定的 InputStream，被设计为从文件中获取输入。与标准的 InputStream 相比，FileInputStream 除了将输入流指向一个文件外，没有任何更多的操作，保持了二者的兼容性。

鉴于这个有限的兼容性，FileInputStream 一般由一个健壮性更好的输入流管理者如 DataInputStream 来操纵。DataInputStream 可以简单地根据 FileInputStream 来创建，这是通过在 DataInputStream 构造器中把 FileInputStream 说明为参数而实现的。

▶参见“DataInputStream Extends FilterInputStream Implements DataInput”

方法

※ FileInputStream (File file) throws FileNotFoundException
  建立一个文件输入流 FileInputStream，它从由指定的 File 对

象操纵的物理文件中读取信息。

※ FileInputStream(String name)throws FileNotFoundException

建立一个从文件名为 name 的文件中读取信息的文件输入流 FileInputStream。

※ FileInputStream (FileDescriptor descriptorObject)

建立一个从指定的文件描述符 FileDescriptor 表示的文件中读取信息的文件输入流 FileInputStream。

※ int available () throws IOException

返回可访问的字节数。

※ void close () throws IOException

关闭文件。

※ protected void finalize () throws IOException

在文件输入流 FileInputStream 结束前自动调用。它关闭文件。

※ final FileDescriptor getFD () throws IOException

返回文件输入流 FileInputStream 正在读取的文件的文件描述符 FileDescriptor。

※ int read () throws IOException

从流中读取一个字节并返回。如果到达文件末尾，返回－1。

※ int read (byte b []) throws IOException

读入并写入到指定的数组。返回读取的字节数，遇到文件尾时返回－1。

※ int read (byte b [], int offset, int len) throws IOException

最多读取 len 个字节，写到指定缓冲区，从 offset 位置开始。返回实际读取的字节数，遇到文件尾时返回－1。

※ long skip (long num) throws IOException

在流中跳过指定数目的字节。

FileOutputStream Extends OutputStream

FileOutputStream 为 OutputStream 类的简单扩展，能够向文件中写数据。和 FileInputStream 一样，FileOutputStream 一般由

健壮性更好的流管理器来管理，比如 DataOutputStream。

▶参见“DataOutputStream Extends FilterOutputStream Implements DataOutput”

方法

※ FileOutputStream (String name) throws IOException

建立一个 FileOutputStream，它向指定文件名的文件中写数据。

※ FileOutputStream (FileDescriptor fdObj)

建立一个往指定文件中进行写操作的 FileOutputStream，这个文件由 FileDescriptor 引用。

※ FileOutputStream (File file) throws IOException

建立一个往指定物理文件中进行写操作的 FileOutputStream，这个文件由指定的 File 对象管理。

※ void close () throws IOException

关闭文件。

※ protected void finalize () throws IOException

这个方法仅在 FileOutputStream 被破坏前调用，用于关闭文件。

※ final FileDescriptor getFD () throws IOException

返回输出文件的 FileDescriptor 对象。

※ void write (int b) throws IOException

写一个字节。

※ void write (byte b []) throws IOException

写缓冲区中所有的字节。

※ void write (byte b [], int offset, int len) throws IOException

从指定数组的 offset 位置开始写 len 个字节。

FilterInputStream Extends InputStream

FilterInputStream 是 InputStream 的一个非抽象版本，它可以用一种基本的同时也是很有用的方式来操作输入流。不幸的是，FilterInputStream 要求以字节序列的方式读取信息——这在读取

类似于双精度型数甚至字符串这样的复杂一些的数据时，都是很繁琐的。

然而，FilterInputStreams 可以由功能更强的输入流管理器如 DataInputStream 来操作。FilterInputStream 同时又是所有其他 InputStream 管理器的基础，这些管理器包括 DataInputStream 和 PushBackInputStream 等。

FilterInputStream 与 InputStream 的区别

为什么可以建立一个 FilterInputStream 的实例，而不是 InputStream 的实例？不能建立 InputStream 的实例是因为它是抽象的——它的 read()方法是未定义的。然而，如果建立了一个 FilterInputStream，必须指定一个准备操作的输入流（InputStream），例如 FilterInputStream 或 ByteArrayInputStream。因为这些类已经定义了 read () 方法，FilterInputStream 管理器运用 InputStream 类的这些方法完成自己的工作。

字段

- protected InputStream in

被读入数据的输入流 (InputStream)。

方法

※ FilterInputStream (InputStream in)

建立一个 FilterInputStream 来管理指定的 InputStream。

※ int available () throws IOException

返回流中有效字节数。

※ void close () throws IOException

关闭流并且调用被管理的 InputStream 中的 close () 方法。

※ synchronized void mark (int readlimit)

在输入流 InputStream 的当前位置放置一个标记。通过调用 reset () 方法，用户可以返回到流当前的位置。readlimit 指定在标记丢失以前的标记位置后可以被读取的字节数。

※ boolean markSupported ()

如果当前的输入流 InputStream 支持作标记，返回 true。

※ int read () throws IOException

从流中读取并返回一个单字节。到文件尾返回－1。

※ int read (byte b []) throws IOException

读到指定的数组中。返回被读取的字节数，到文件尾返回－1。

※ int read (byte b [], int offset, int len) throws IOException

从指定数组的 offset 开始，最多读取 len 个字节到缓冲区中。返回被读取的字节数，到文件尾返回－1。

※ synchronized void reset () throws IOException

返回到流中标记位置。

※ long skip (long num) throws IOException

跳过流中指定数目的字节。

FilterOutputStream Extends OutputStream

FilterOutputStream 提供了一种管理 OutputStream 的简单而有效的操作方式。该类是所有其他被用来管理输出流 OutputStream 类的基础，例如 BufferedOutputStream 和 DataOutputStream。

▶参见 “FilterInputStream Extends InputStream”

字段

• protected OutputStream out

被管理的输出流 OutputStream。

方法

※ FilterOutputStream (OutputStream out)

建立一个新的 FilterOutputStream 来管理指定的输出流 OutputStream。

※ void close () throws IOException

关闭 OutputStream。

※ void flush () throws IOException

将缓冲区信息写入流并刷新缓冲区。这仅影响 OutputStream 中将要写入流中的缓冲区内的信息。

※ void write (int b) throws IOException

写一个单字节。

※ void write (byte b []) throws IOException

写缓冲区中所有的字节。

※ void write (byte b [], int offset, int len) throws IOException

从指定数组的 offset 位置开始写 len 个字节。

Abstract InputStream

InputStream 是操作输入流的基本类。它是所有输入流类的基础，然而是抽象的，这就意味着不能建立 InputStream 的实例。但是，因为所有输入流类都是从InputStream类继承来的，因而可以用在InputStream中建立的所有方法来操作所有的输入流类。

可以看到输入流的操作器（比如 DataInputStream）将 InputStream 作为参数。虽然不可能以一个单纯的 InputStream 提供给这些操作者，但这个流可以从 InputStream 派生然后被作为一种 InputStream 类型的输入流对待。

方法

※ InputStream ()

建立一个新的输入流 InputStream。

※ int available () throws IOException

返回流中有效的字节数。

※ void close () throws IOException

关闭流。它调用被管理的输入流 InputStream 中的 close () 方法。

※ synchronized void mark (int readlimit)

在输入流 InputStream 的当前位置放置一个标记。通过调用 reset () 方法，可以返回到流当前的位置。在标记丢失以前，readlimit 指定标记位置之后可以读取的字节数。

※ boolean markSupported ()

如果当前的输入流 InputStream 可以作标记，返回 true。

※ int read () throws IOException

从流中读取并返回一个单字节，若到达文件尾则返回－1。

※ int read (byte b []) throws IOException

读到指定的数组中。返回被读取的字节数，到文件尾返回－1。

※ int read (byte b [], int offset, int len) throws IOException

最多读取 len 个字节到指定缓冲区中，从 offset 开始。返回被读取的字节数，到文件尾返回－1。

※ synchronized void reset () throws IOException

返回到流中标记位置。

※ long skip (long num) throws IOException

跳过流中指定数目的字节。

LineNumberInputStream Extends FilterInputStream

LineNumberInputStream 是一个输入流管理器。它比 FilterInputStream 增加的功能是：使用户不仅能从流中读取信息，而且能跟踪记录用户从这个流中已经读取的行数。

方法

※ LineNumberInputStream (InputStream in)

建立一个新的 LineNumberInputStream，管理指定的 InputStream 输入流。

※ int available () throws IOException

返回有效字节数。

※ int getLineNumber ()

返回当前的行号。

※ void mark (int readlimit)

在输入流 InputStream 的当前位置放置一个标记。通过调用 reset () 方法，可以返回到流当前的位置。在标记丢失以前，readlimit 指定标记位置之后可以被读取的字节数。

※ int read () throws IOException

从流中读取并返回一个单字节。

※ int read (byte b []，int offset，int len) throws IOException

从指定数组的 offset 开始，最多读取 len 个字节到该数组中。返回被读取的字节数。

※ void reset () throws IOException

返回到流中标记位置。

※ void setLineNumber (int lineNumber)

将当前的行号设置为指定值。

※ long skip (long num) throws IOException

跳过流中指定数目的字节。返回实际跳过的字节数。

Abstract OutputStream

OutputStream 是一个抽象类，是其他所有类型输出流类建立的基础，虽然不能建立 OutputStream 的实例，但由于所有输出流类都是基于该类的——这就提供了一种操作所有类型输出流类的方便的手段。

方法

※ OutputStream ()

建立一个新的输出流。

※ void close () throws IOException

关闭输出流 OutputStream。

※ void flush () throws IOException

在继承类中，该方法用以将缓冲区的字节写入输出流中。而在这里，该方法为空。

※ abstract void write (int b) throws IOException

写一个单字节。这是这个类的本质所在，在任何非抽象子类中它必须被超越。

※ void write (byte b []) throws IOException

将数组写入输出流中。

※ void write (byte b []，int offset，int len) throws IOException

将数组从 offset 开始的 len 字节写入输出流中。

PipedInputStream Extends InputStream

类 PipedInputStream 与类 PipedOutputStream 协同工作。通过将二者连接，能够建立两个独立的线程——相互独立运行——能够以非常简捷的方式互相传送信息。

▶参见“PipedOutputStream Extends OutputStream”

图 19 的输出是通过连接 PipedInputStream 的实例与 PipedOutputStream 的实例产生的——各自被一个独立的线程操作（参见程序 12）。根据线程管理特性，为了将控制权移交给 Reader 类，Writer 类中的 sleep () 语句是必要的。没有 sleep () 语句，Writer 类将首先写它的所有数据，将控制权交给 Reader 类，然后再允许在所有的数据中进行读操作。

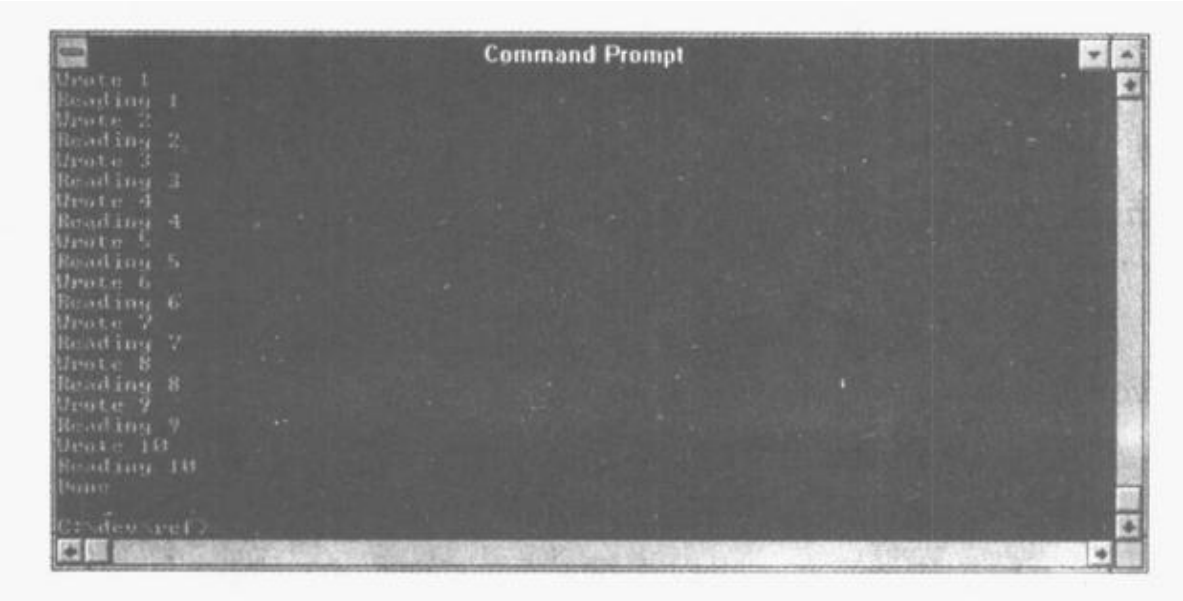


图 19　程序 12 的运行结果

还应注意的是，在这个例子中，程序用了 Reader 类的错误句柄来退出程序。一旦 Write 类完成自己的工作，输出管道将被关闭。这将引起 Reader 类中发生异常——终止整个程序。

程序 12　PipedInputStream 和 PipedOutputStream 的例子

```
class Reader extends Thread{

    PipedInputStream in;
    boolean running;
```

```
    Reader(PipedOutputStream out) {
        try{
            in=newPipedInputStream(out);
            running=true;
        }
        catch(Exception e)
            System.out.println(e.tostring());
    }

    PipedInputStream getStream() {
        return(in)
    }
    public void run(){
        while(running){
            try{
                System.out.println("Reading " + in.read
()));
            }
            catch(Exception e) {
                System.out.println("Done");
                System.exit(1);
            }
        }
    }
}

class Writer extends Thread{
    public PipeOutputStream out;
    boolean running;
```

```
Write() {
        out=new PipedOutputStream();
        running=true;
}

PipedOutputStream getStream() {
        return (out);
}

void connectTo(PipedInputStream in) {
        try{
            out.connect(in);
        }
        catch(Exception e) {
            System.out.println(e.toString());
          running=false;
        }
}

public void run() {
        try{
            for(int i=1;i<=10;i++) {
                out.write(i);
                System.out.println("Wrote "+i+" ");
                yield();
                sleep(2000);
            }
        catch(Exception e){
            System.out.println(e.toString());
        }
```

```
    }
}


public class PipedExample{
    public static void main(String args[]){
        Writer w=new Writer();
        Reader r=new Reader(w.getStream());
        w.connectTo(r.getStream());
        r.start();
        w.start();
    }
}
```

方法

※ PipedInputStream ()

建立一个PipedInputStream。在使用之前，必须先将这个流与PipedOutputStream 连接。

※ PipedInputStream (PipedOutputStream src) throws IOException

建立一个 PipedInputStream 并将它和指定的 PipedOutputStream 相连接。

※ void close () throws IOException

关闭一个流并命令它不再从 PipedOutputStream 中读。

※ void connect (PipedOutputStream src) throws IOException

将 PipedInputStream 和指定的 PipedOutputStream 相连接。

※ synchronized int read () throws IOException

从流中读取一个字节并返回。

※ synchronized int read (byte b [], int offset, int len) throws IOException

从流中读取 len 个字节，将它存储在指定的数组中，从位置

offset 开始。

PipedOutputStream Extends OutputStream

与类 PipedInputStream 一同工作，PipedOutputStream 能够借助一个流使当前正在运行的两个线程通信。

▶参见 “PipedInputStream Extends InputStream”

方法

※ PipedOutputStream ()

建立一个新的 PipedOutputStream，这必须和 PipedInputStream 相连接才能进行工作。

※ PipedOutputStream (PipedInputStream dest) throws IOException

建立一个新的 PipedOutputStream，并将它和指定的 PipedInputStream 相连接。

※ void close () throws IOException

通知 PipedInputStream 已经到达它的最后一个字节，以便有效地关闭流。

※ void connect (PipedInputStream dest) throws IOException

将 PipedOutputStream 与指定的 PipedIntputStream 相连接。

※ void write (int b) throws IOException

将一个单字节写入流中。

※ void write (byte b [], int offset, int len) throws IOException

从指定的缓冲区 offset 位置开始写入 len 个字节到流中。

PrintStream Extends FilterOutputStream

PrintStream 是一个非常直接的流管理器，它将所有类型的数据以字符串输出。

注意：由于 PrintStream 将任何数据均以字符串形式输出，所有的数据类型——甚至整型数和双精度型数——也将以字符串方式输出。当后来接着读取这些信息时应小心：所有的数据类型将先以字符串或字节形式读取，然后才将它们转化为实际的数据类型。

方法

※ PrintStream（OutputStream out）

建立一个新的 PrintStream 来管理指定的 OutputStream。

※ PrintStream（OutputStream out，boolean autoflush）

建立一个新的 PrintStream 来管理指定的 OutputStream。如果 autoflush 为 true，则每次写入一个换行符（'\n'）时流将被刷新。

※ boolean chechError（）

在写入流时若遇到错误则返回 true。一旦错误发生，本方法返回 true。

※ void close（）

关闭流。

※ void flush（）

刷新由 OutputStream 缓冲的任何信息。

※ void print（boolean b）

打印一个布尔数 boolean。

※ void print（double d）

打印一个双精度数 double。

※ void print（float f）

打印一个浮点数 float。

※ void print（long l）

打印一个长整型数 long。

※ void print（int i）

打印一个整型数 int。

※ void print（char c）

打印一个字符型 char。

※ synchronized void print（char s []）

打印一个字符数组。

※ synchronized void print（String s）

打印一个字符串。

※ void print (Object obj)

打印由 String.valueOf (obj) 返回的值。

※ synchronized void println (boolean b)

打印一个 boolean 型值，之后跟一个换行符。

※ synchronized void println (double d)

打印一个 double 型值，之后跟一个换行符。

※ synchronized void println (float f)

打印一个 float 型值，之后跟一个换行符。

※ synchronized void println (long l)

打印一个 long 型值，之后跟一个换行符。

※ synchronized void println (int i)

打印一个 int 型值，之后跟一个换行符。

※ synchronized void println (char c)

打印一个 char 型值，之后跟一个换行符。

※ synchronized void println (char s [])

打印一个字符数组，之后跟一个换行符。

※ synchronized void println (String s)

打印一个字符串，之后跟一个换行符。

※ synchronized void println (Object obj)

打印由 String.valueOf (obj) 返回的值，之后跟一个换行符。

※ void println ()

打印一个换行符。

※ void write (int b)

写一个单字节。

※ void write (byte b [], int offset, int len)

写指定数组中从 len 位置开始到 len+offset 位置结束的所有字节。

PushbackInputStream Extends FilterInputStream

PushbackInputStream 是一个在解析流时很有用的流管理器。它能够使你“看一看”流中的下一个字节，而不需要将这个

字节从流中完全移走。如果用户确定该字节不是想要的字节，还可以通过 unread () 方法再送回流中。这将会把这个字节“推”回流中，任何下一个 read () 语句将从这一字节开始读取。

如何将一个字符送回流中?

不可。确切地讲，PushbackInputStream 实际上不能将一个字符送回流中。它只是将字符保存在名为 pushBack 的 protected 字段中，下一次从流中读取，读到的是 pushBack 字节——不是流的下一字节。

字段

- protectd int pushBack

  已被放入 pushBack 字段的字节。

方法

※ PushbackInputStream (InputStream in)

建立一个新的 PushbackInputStream 来管理指定的流。

※ int available () throws IOException

返回流中有效的字节数目。

※ boolean markSupported ()

由于该 InputStream 句柄不支持 mark/reset 过程，因此返回 false。

※ int read () throws IOException

读取并返回一个单字节。

※ int read (byte b [], int offset, int len) throws IOException

从数组中读取 len 个字节并且将它们放置在从 b[offset]开始的指定缓冲区。

※ void unread (int ch) throws IOException

将指定的字节回放入流中。

RandomAccessFile Implements DataOutput DataInput

与 C 语言中随机访问文件很相近，RandomAccessFile 类提供了一个很灵活的文件读写工具。它还允许用户指定访问权限——只读或者读写——该类提供较高的安全性。

方法

※ RandomAccessFile (File file, String mode) throws IOException

建立一个新的 RandomAccessFile 对象，用来操作由 File 对象管理的物理文件。mode 指定访问文件的权限类型。“r”为只读，“rw”允许读和写。

※ RandomAccessFile (String name, String mode) throws IOException

建立一个新的 RandomAccessFile 对象，用来操作具有指定文件名的文件。mode 指定访问文件的权限类型。“r”为只读，“rw”允许读和写。

※ void close () throws IOException

关闭文件。

※ final FileDescriptor getFD () throws IOException

返回被访问文件的文件描述符 FileDescriptor。

※ long getFilePointer () throws IOException

返回文件中当前的指针位置。

※ long length () throws IOException

返回文件长度。

※ int read () throws IOException

读取并返回一个单字节。

※ int read (byte b []) throws IOException

读取到指定的字节数组，返回读到的字节数。

※ int read (byte b [], int offset, int len) throws IOException

读取 len 个字节，并将它们放置在从 offset 开始的数组。返回实际读到的字节数。

※ final boolean readBoolean () throws IOException

从文件中读一个 boolean 型值。

※ final byte readByte () throws IOException

从文件中读一个 byte 型值。

※ final char readChar () throws IOException

从文件中读一个 char 型值。

※ final double readDouble () throws IOException

从文件中读一个 double 型值。

※ final float readFloat () throws IOException

从文件中读一个 float 型值。

※ final void readFully (byte b [], int offset, int len) throws IOException

从文件中读 len 个字节，并将它们放置在数组中，从数组的 offset 位置开始。

※ final void readFully (byte b []) throws IOException

从文件中读字节，并将它们放置在数组中，直到数组满为止。

※ final int readInt () throws IOException

从文件中读取并返回一个 int 型值。

※ final String readLine () throws IOException

从流中读取一个字符串，直到遇到 '\n' 或 EOF 为止。

※ final long readLong () throws IOException

从流中读取并返回一个 long 型值。

※ final short readShort () throws IOException

从流中读取并返回一个 short 型值。

※ final int readUnsignedByte () throws IOException

从流中读取并返回一个 unsigned byte 无符号字节型值。

※ final int readUnsignedShort () throws IOException

从流中读取并返回一个 unsigned short 型值。

※ final String readUTF () throws IOException

从流中以 UTF 格式读取并返回一个字符串。

※ void seek (long pos) throws IOException

将文件指针放置在文件中的指定位置。

※ int skipBytes (int num) throws IOException

将文件指针跳过指定字节数。

※ void write (byte b []) throws IOException
将指定缓冲区中内容写入文件。
※ void write (byte b [], int offset, int len) throws IOException
将指定缓冲区中内容从 offset 开始将 len 个字节写入文件。
※ final void writeBoolean (boolean v) throws IOException
将一个 boolean 型值写入文件。
※ final void writeByte (int v) throws IOException
将一个 byte 型值写入文件。
※ final void wtiteBytes (String s) throws IOException
将一个字符串以系列字节的形式写入文件。
※ final void writeChar (int v) throws IOException
将一个 char 型值写入文件。
※ final void writeChars (String s) throws IOException
将一个字符串以系列字符的形式写入文件。
※ final void writeDouble (double v) throws IOException
将一个 double 型值写入文件。
※ final void writeFloat (float v) throws IOException
将一个 float 型值写入文件。
※ final void writeInt (int v) throws IOException
将一个 int 型值写入文件。
※ final void writeLong (long v) throws IOException
将一个 long 型值写入文件。
※ final void writeShort (int v) throws IOException
将一个 short 型值写入文件。
※ final void writeUTF (String str) throws IOException
将一个字符串以 UTF 格式写入文件。

SequenceInputStream Extends InputStream

SequenceInputStream 能够将多个输入流 (InputStreams) 链接成一个伪流。SequenceInputStream 从一个流读取完数据后，将立即从下一个流开始读。

方法

※ SequenceInputStream (InputStream s1, InputStream s2)

建立一个新的 SequenceInputStream，连接指定的输入流 InputStream。

※ SequenceInputStream (Enumeration e)

建立一个新的 SequenceInputStream，连接指定的 Enumeration 中所有输入流 InputStream。

※ void close () throws IOException

通过遍历所有的流来有效的关闭它们，直到最后一个流被标志。

※ int read () throws IOException

从当前流读取并返回一个字节。

※ int read (byte b [], int offset, int len) throws IOException

从流序列中读取 len 个字节，放在数组 (b []) 中，从指定的 offset 位置开始。

StreamTokenizer

在解析流的过程中，StreamTokenizer 是一个任务繁重的工具。运用 nextToken () 方法，可以搜索流中的任何标记 (tokens)——即用户定义的重要字符。遇到这个标记，StreamTokenizer 将返回一个标志。标志可能是字符本身，也可能是像 TT_EOF (标志流结束) 一类的特定标志。

字段

- double nval

  如果返回标志为 TT_NUMBER，该值为最近读到的数字。

- String sval

  如果返回标志为 TT_WORD，该值为最近读到的字符串。

- final static int TT_EOF

  文件结尾标志。

- final static int TT_EOL

行结尾标志。

- final static int TT_NUMBER

用来指示刚被读取的数字的标志。

- final static int TT_WORD

用来指示刚被读取的词（字符串）的标志。

方法

※ StreamTokenizer（InputStream in）

建立一个新的 StreamTokenizer 用来管理指定的输入流 InputStream。

※ void commentChar（int ch）

设置注解字符。

※ void eollsSignificant（boolean flag）

如果 flag 为 true，则行结尾标志符将以标志形式返回。

※ int lineno（）

返回当前的行号。这在行尾标志符无效时很有用处。

※ void lowerCaseMode（boolean flag）

如果 flag 为 true，则 sval 中的字符串被转化为小写字母。

※ int nextToken（）throws IOException

在流中向前移动位置直到遇到下一个标志符，并返回这个标志符。

※ void ordinary（int ch）

强制一个字符为“一般”字符。换句话说，这个字符将不被看作是标志符。

※ void ordinaryChars（int low，int hi）

强制从 low 到 hi 的字符都为“一般”字符。这些字符将不被看作是标志符。

※ void parseNumbers（）

强制数字解析。这个方法在 StreamTokenizer 建立时被缺省调用。但如果用户重置句法，必须重新再次调用这个方法。这个任务通过将数字（0—9）、句号（.）、连字符（—）作为特定字符

而被 tokenizer 进行不同的操作来完成的。

※ void pushBack ()

有效地将当前标志符返回至流中。其结果，下次调用 nextToken()时将返回同样的标志，如果可能，同时返回带有同样的 nval 或 sval 的标志。

※ void quoteChar (int ch)

设置引用字符。

※ void resetSyntax ()

重置 tokenizer 的句法。所有的字符置为普通状态。

※ void slashSlashComments (boolean flag)

如果 flag 为 true，则双斜杠 (//) 标志表示注解。

※ void slashStarComments (boolean flag)

如果 flag 为 true，则斜杠一星 (/*到*/) 标志表示注解。

※ String toString ()

返回包含当前标志符的字符串，可能的话返回 sval 或 nval。

※ void whitespaceChars (int low, int hi)

所有在 low 到 hi 位之间的字符被认为是空白。缺省情况下字符 0－32 被认为是空白。

※ void wordChars (int low, int hi)

所有在 low 到 hi 之间的字符被认为是词的一部分。

StringBufferInputStream Extends InputStream

StringBufferInputStream 能够把字符串当作输入流来读取信息。

▶参见 “PushbackInputStream Extends FilterInputStream”

字段

- protected String buffer 对在构造器方法中指定的字符串的引用。
- protected int count

  缓冲区的长度。
- protected int pos

缓冲区中当前的位置。

方法

※ StringBufferInputStream (String s)

建立一个新的 StringBufferInputStream，读指定的字符串。

※ synchronized int available ()

返回字符串缓冲区中的有效字符数。

※ synchronized int read ()

读取并返回一个字节。如果不再有字节则返回－1。

※ synchronized int read (byte b [], int offset , int len)

从指定字串缓冲区的 offset 开始，最多读取 len 个字节到该数组中。返回被读取的字节数，到达文件尾则返回－1。

※ synchronized void reset ()

返回到缓冲区的开始。

※ synchronized long skip (long num)

在缓冲区中跳过最多 num 个字节，返回实际跳过的字节数。

2. 接口

DataInput

这一接口声明了一系列的方法，定义了不依赖于硬件的输入流的行为。它由 DataInputStream 类实现。

方法

※ abstract boolen readBoolean () throws IOException

从流中读取并返回一个 boolean 型值。

※ abstract byte readByte () throws IOException

从流中读取并返回一个 byte 型值。

※ abstract char readChar () throws IOException

从流中读取并返回一个 char 型值。

※ abstract double readDouble () throws IOException

从流中读取并返回一个 double 型值。

※ abstract float readFloat () throws IOException

从流中读取并返回一个 float 型值。

※ abstract void readFully (byte b []) throws IOException

从流中读取，直到整个数组被填满。

※ abstract void readFully (byte b [], int offset, int len) throws IOException

从流中读取 len 个字节。这些字节将从 offset 位置开始被放置在数组中。

※ abstract int readInt () throws IOException

从流中读取并返回一个 int 型值。

※ abstract long readLong () throws IOException

从流中读取并返回一个 long 型值。

※ abstract short readShort () throws IOException

从流中读取并返回一个 short 型值。

※ abstract int readUnsignedByte () throws IOException

从流中读取并返回一个 unsigned byte 型值。

※ abstract int readUnsignedShort () throws IOException

从流中读取并返回一个 unsigned short 型值。

※ abstract String readUTF () throws IOException

从流中以 UTF 格式读取并返回一个字符串型值。

※ abstract int skipBytes (int num) throws IOException

在缓冲区中跳过最多 num 个字节，返回实际跳过的字节数。

DataOutput

这一接口声明了一系列的方法，定义了不依赖于硬件的输出流的行为。它由 DataOutputStream 类实现。

方法

※ abstract void write(byte b[],int offset,int len) throws IOException

将数组中从 b [offset] 到 b [offset+len] 的字节写入流中。

※ abstract void write (byte b) throws IOException

将一个 byte 型值写入流中。

※ abstract void write (int b) throws IOException

将一个 int 型值写入流中。

※ abstract void writeBoolean (boolean v) throws IOException

将一个 boolean 型值写入流中。

※ abstract void writeByte (int v) throws IOException

将一个字节写入流中。

※ abstract void writeBytes (String s) throws IOException

将一个字符串以一串 byte 的形式写入流中。

※ abstract void writeChar (int v) throws IOException

将一个 char 型值写入流中。

※ abstract void writeChars (String s) throws IOException

将一个字符串以一系列字符的形式写入流中。

※ abstract void writeDouble (double v) throws IOException

将一个 double 型值写入流中。

※ abstract void writeFloat (float v) throws IOException

将一个 float 型值写入流中。

※ abstract void writeInt (int v) throws IOException

将一个 int 型值写入流中。

※ abstract void writeLong (long v) throws IOException

将一个 long 型值写入流中。

※ abstract void writeShort (int v) throws IOException

将一个 short 型值写入流中。

※ abstract void writeUTF (String str) throws IOException

以 UTF 格式将一个字符串写入流中。

FilenameFilter

FilenameFilter 是一个用来在屏幕上显示目录的工具。它用在 File 类和 java.awt.FileDialog 类。

方法

※ abstract boolean accept (File dir, String name)

如果指定的文件满足过滤器的要求，返回 true。

五、JAVA.LANG

这些类实际上是 java 语言的核心。这部分除了包括基本数据类型的封装器（wrapper），比如 Integer 和 String 以外，还提供通过 Throwable 和 Error 类处理错误的手段。而且，不论是命令提示符还是在 Netscape 中的 java 控制台，SecurityManager 和 System 类提供了对客户端一定程度上的控制。

1. 类

Boolean

尽管 boolean 是一种基本的数据类型，但 Boolean 类作为一个软件包，提供了处理 boolean 型值更有效的手段。基本数据类型的封装器在使用散列表（hashtables）中很有用，散列表只接受对象，而不接受基本数据类型。

字段

- final static Boolean FALSE

  建立一个值为 false 的 Boolean 对象。

- final static char MAX _ VALUE

  字符的最大值：65535。这些值偶然被存在于 Boolean 类中。

- fianl static char MIN _ VALUE

  字符的最小值：0。这些值偶然被存在于 Boolean 类中。

- final static Boolean TRUE

  建立一个值为 true 的 Boolean 对象。

方法

※ Boolean（String s）

以一个特定的值建立一个 Boolean 型对象：如果 s 等于 true，该对象为 true，否则为 false。

※ Boolean（boolean value）

建立一个 Boolean 型对象，其值为 Boolean 参数指定的值。

※ boolean booleanValue（）

返回 Boolean 对象的值。

※ boolean equals（Object anotherObject）

如果另一个布尔对象 anotherBoolean 是一个具有同样值的 Boolean 型值，返回 true。

※ static boolean getBoolean（String s）

如果 s 等于 true，返回 true。否则返回 false。

※ int hashCode（）

返回 Boolean 对象的散列码———一个依赖于对象值的数值。

※ String toString（）

依据对象的布尔值返回 true 或 false。

※ static Boolean valueOf（String s）

返回一个 Boolean 对象，该对象的值是指定字符串的值。

Character

类 Character 用作操作字符（char）型数据的封装器类。

字段

• final static int MAX _ RADIX

该最大基数（Radix）用来将一个用字符表示的数字转换为整型数。这里的基数（Radix）即是一般数学意义上的基数，指示每个位置的值。MAX _ RADIX 是 36。

• final static int MIN _ RADIX

该最小基数（Radix）用来将一个用字符表示的数字转换为整型数。这里的基数（Radix）即是一般数学意义上的基数，指示每个位置的值。MIN _ RADIX 是 2。

方法

※ Character（char value）

以一个指定的值建立一个 Character 对象。

※ char charValue（char ch）

返回 Character 对象的值。

※ static int digit (char ch, int base)

把一个用 ch 表示的数字转换为具有指定基数的整数。

※ boolean equals (Object anotherObject)

如果 anotherObject 是一个具有同样 char 值的 Character 对象，返回 true。

※ static char forDigit (int digit, int base)

返回用字符 char 表示的指定基 base 中的 digit 值。

※ int'hashCode ()

返回 Character 对象的散列码——char 转化为 int。

※ static boolean isDigit (char ch)

如果 char 的值在 0 到 9 之间，返回 true。

※ static boolean isJavaLetter (char ch)

如果 ch 是一个 Java 字母——即可以用作标志符的第一个字母——返回 true。包括所有的字母，美元符（'$'）和下划线（'_'）。

※ static boolean isJavaLetterOrDigit (char ch)

如果 ch 是一个 Java 标志符中的有效字符，返回 true。Java 标志符包括所有的字母和数字以及美元符（'$'）和下划线（'_'）。如果等于'$'，'_'，或者 isLetterOrDigit (ch) 为真，返回 true。

※ static boolean isLetter (char ch)

如果是字母返回 true。

※ static boolean isLetterOrDigit (char ch)

如果是字母或数字返回 true。

※ static boolean isLowerCase (char ch)

如果 ch 值为小写返回 true。

※ static boolean isSpace (char ch)

如果 cher 值为 ' '，'\t'，'\f'，'\n' 或 '\r' 返回 true。

※ static boolean isSpace (char ch)（译者注：原文如此）

如果 ch 为五个空白符' ','\t','\f','\n'或'\r'之一则返回

true。

※ static boolean isTitleCase (char ch)

如果 ch 是四个特殊唯一码字符之一则返回 true。之所以说这些字符是特殊的，因为它们很像两个 ASCII 字符在一起，比如 *Lj*。这个方法将返回 true 仅当 ch 为这四个字符之一并且第一个字母为大写。

※ static boolean isUpperCase (char ch)

如果 char 值为大写返回 true。

※ static char toLowerCase (char ch)

返回指定 char 的小写值。

※ String toString ()

以字符串形式返回 char 值。

※ static char toTitleCase (char ch)

返回 ch 的格式。除了四个字符，其他仅简单为它们的大写格式。对于四个类似于二个 ASCII 字符的特殊字符，第一个字母大写，其余仍小写。

※ static char toUpperCase (char ch)

返回指定 char 的大写值。

Class

对虚拟机来讲，每一个类都是一个对象。然而。在虚拟机中，每一个类（和接口）都是由 Class 类来操纵。当作为对象的类实例化后（实际的对象），这个类使用在建立该类时使用的代码。由于不能将 Class 类实例化，因而必须运用 forName () 方法来返回一个 Class 对象。

方法

※ static Class forName (String className) throws ClassNotFoundException

返回指定类的 Class 对象描述符。

※ ClassLoader getClassLoader ()

返回指定类的类载入器 ClassLoader。

※ Class [] getInterfaces ()

返回一个同由该类实现的接口数组。

※ String getName ()

返回该类的名字。

※ Class getSuperclass ()

返回该类的超类的 Class 对象。

※ boolean isInterface ()

如果这个“类”是接口返回 true。

※ Object newInstance () throws InstantiationException, IllegalAccessException

建立一个由该 Class 对象描述的类实例。

※ String toString ()

返回一个字符串，该字符串含有单词 class 或 interface 以及该类/接口的名字。

Abstract ClassLoader

类 ClassLoader 可以用来从当前服务器以外的地方比如网络上装入一个类。由于这个过程很大程度上依赖于用户自己的环境，所以具体的操作由用户来确定。

由于 ClassLoader 是一个抽象类，所以，如果打算运用它，必须先扩展它，然后用方法 loadClass () 装载。一旦完成，用户必须再调用 defineClass () 方法来建立一个类对象来描述这个类，另外这个类对象还能够使用户建立类的实例。

方法

※ protected ClassLoader ()

建立一个新的 ClassLoader。

※ abstract Class loadClass (String name, boolean resolve) throws ClassNotFoundException

这一方法必须由用户自己的类实现重载。resolve 可供解析这个类：装入所有与新装入的类有关联的类。

※ final Class defineClass (byte data [], int offset, int length)

建立一个 Class 对象，用来描述从指定的一系列字节建立的类。这个类将由从 data [offset] 开始直到 data [offset+length] 结束的的字节建立。

※ final Class findSystemClass (String name) throws ClassNotFoundException

从系统中装入一个类。这将使用缺省的类载入器。

※ final void resolveClass (Class c)

通过装入与指定的类相关的所有类来解析这个类。这必须在类被使用之前完成。

Compiler

该类提供一种与未来的 Java 技术的接口，使用户能更有效地控制 Java 编译器。然而并不像 1.0.2 JDK 一样，它没有一套完备的库或手段建立 Compiler 对象。

方法

※ static Object command (Object any)

运行指定的命令。

※ static boolean compileClass (Class class)

编译指定的类。

※ static boolean compileClasses (String string)

编译指定的类。

※ static void disable ()

使编译器无效。

※ static void enable ()

使编译器有效。

Double Extends Number

类 Double 是一个操作双精度 (double) 型数据的类软件包。

字段

- final static double MAX _ VALUE

  double 值的最大值：$1.79769313486231570 * 10^{308}$。

- final static double MIN _ VALUE

double 值的最小值：$-4.94065645841246544e*10^{324}$。

- final static double NaN

这个特殊值代表一个不是数字的 double。被设为等于 0.0/0.0。它不等于任何值——甚至它自己。

- final static double MEGATIVE _ INFINITY

负的无穷大：-1.0/0.0。

- final static double POSITIVE _ INFINITY

正的无穷大：1.0/0.0。

方法

※ Double (double value)

根据指定的 double 值建立新的 Double 对象。

※ Double (String s) throws NumberFormatException

建立 Double 对象，指定 s 用来代表一个有效的 double 值。

※ static long doubleToLongBits (double value)

返回位表示的指定值。

※ double doubleValue ()

返回 Double 对象的值。

※ boolean equals (Object anotherObject)

如果与另一个 Double 对象的值相等，返回 true。

※ float floatValue ()

以 float 类型返回对象的 double 值。

※ int hashCode ()

返回 Double 对象的散列码——用 int 型表示的 double 值。

※ int intValue ()

返回表示为一个 int 型的 double 值。

※ boolean isInfinite ()

如果 double 值等于 POSITIVE _ INFINITY 或 NEGATIVE _ INFINITY 则返回 true。

※ static boolean isInfinite (double v)

如果指定的 double 等于 POSITIVE _ INFINITY 或 NEGA-

TIVE _ INFINITY 则返回 true。

※ boolean isNaN ()

如果 double 等于指定的 NaN 则返回 true。

※ static boolean isNaN (double v)

如果指定的双精度值等于指定的 NaN 则返回真。

※ static double longBitsToDouble (long bits)

运用给定 bit 表示形式，返回 double 的表示形式。

※ long longValue ()

以长整型形式返回 double 数。

※ String toString ()

以字符串形式返回 double 数。

※ static String toString (double d)

将指定的 double 数转化为字符串。

※ static Double valueOf (String s) throws NumberFormatException

将指定的字符串转化为 Double 对象。

Float Extends Number

类 Float 是一个操作浮点（float）型数据的封装器（wrapper）类。

字段

• final static float MAX _ VALUE

浮点数的最大值：$3.40282346638528860 * 10^{38}$。

• final static float MIN _ VALUE

浮点数的最小值：$1.40129846432481707 * 10^{-45}$。

• final static float NaN

这个特殊值代表一个不是数字的 float 数（译者注：原文为 double 数）。被设为等于 0.0/0.0。它不等于任何值——甚至它自己。

• final static float MEGATIVE _ INFINITY

负的无穷大：－1.0/0.0。

• final static float POSITIVE _ INFINITY

正的无穷大：1.0/0.0。

方法

※ Float (String s) throws NumberFormatException

建立一个新的 Float 对象，值等于指定的字符串。

※ Float (double value)

建立一个新的 Float 对象，初始值等于指定的 double 值。

※ Float (float value)

建立一个新的 Float 对象，初始值等于指定的 float 值。

※ double doubleValue ()

返回作为 double 的 float 值。

※ boolean equals (Object anotherObject)

如果另一个对象是具有同样值的 Float，返回 true。

※ static int floatToIntBits (float value)

返回指定浮点数的位表达值。

※ float floatValue ()

返回 Float 对象的浮点值。

※ int hashCode ()

返回 Float 对象的散列码：浮点数被演算成整数。

※ static float intBitsToFloat (int bits)

返回位表达式的浮点数值。

※ int intValue ()

以整型数返回 Float 对象的值。

※ boolean isInfinite ()

如果浮点数值等于 POSITIVE _ INFINITY 或 NEGATIVE _ INFINITY，返回 true。

※ static boolean isInfinite (float v)

如果指定的 float 值等于负无穷大或正无穷大，返回 true。

※ boolean isNaN ()

如果 float 值不是数字，返回 true。

※ static boolean isNaN (float v)

如果指定的 float 值不是数字，返回 true。

※ long longValue ()

以 long 型返回 float 值。

※ String toString ()

以字符串形式返回 float 值。

※ static String toString (float f)

将指定的 float 转化为字符串。

※ static Float valueOf (String s) throws NumberFormatException

返回一个 Float 对象，该对象的初始值等于字符串中所含的 float 值。

Integer Extends Number

类 Integer 是一个操作整型 (int) 数值的封装器类。通常，由 API (应用程序接口) 中的方法返回 Integer 对象，这些方法提供数值的同时又提供了一种操作数值的手段。

字段

- final static int MAX_VALUE

  Integer 对象的最大值：2147483647。

- final static int MIN_VALUE

  Integer 对象的最小值：－2147483648。

方法

※ Integer (int value)

根据给定的值建立一个新的 Integer 对象。

※ Integer (String s) throws NumberFormatException

建立一个新的 Integer 对象，其初始值等于指定字符串中的整型数值。

※ double doubleValue ()

以双精度形式返回整型数值。

※ boolean equals (Object anotherObject)

如果 anotherObject 是 Integer 对象并具有同样的数值，返回 true。

※ float floatValue ()

以浮点数形式返回整型数值。

※ static Integer getInteger (String propName)

返回一个系统属性，其值为以十六进制或八进制格式保存的整型数。

※ static Integer getInteger (String propName, int val)

返回一个系统属性，其值为以十六进制或八进制格式保存的整型数。如果这个过程失败，返回一个值等于 val 的 Integer 对象。

※ static Integer getInteger (String propName, Integer val)

返回一个系统属性，其值为以十六进制或八进制格式保存的整型数。如果这个过程失败，返回 val。

※ int hashCode ()

返回 Integer 对象的散列码：它的整型值。

※ int intValue ()

返回整型数值。

※ long longValue ()

以长整型数形式返回 Interger 对象的数值。

※ static int parseInt (String s) throws numberFormatException

返回一个由指定字符串表达的整型数的数值。

※ static int parseInt (String s, int base) throws numberFormat Exception

返回一个整数，该整数在字符串中以 base 为基所指定而构成。例如：parseInt ("101", 2) =5 而 parseInt ("101", 10) = 10。(原书如此——译者注)

※ static String toBinaryString (int)

将整数用字符串表示，以无符号二进制数形式表达。

※ static String toHexString (int)

将整数用字符串表示，以无符号十六进制数形式表达。

※ static String toOctalString (int)

将整数用字符串表示，以无符号八进制数形式表达。

※ String toString ()

以字符串形式返回整数值。

※ Static String toString (int i)

以字符串形式返回指定的整数值。

※ static String toString (int i, int base)

用字符串以指定的基 base 表示整型数 i。例如：toString (5, 2) =101, toString (5, 10) =5。

※ static Integer valueOf (String s) throws NumberFormatException

返回一个 Integer 对象，该对象的值等于字符串所表示的整型数值。

※ static Integer valueOf (String s, int base) throws NumberFormatException

以指定的基返回字符串所表示的数字值。

Long Extends Number

类 Long 是一个操作长整型 (Long) 数据的类封装器。

字段

- final static long MAX _ VALUE

  Long 的最大值：9223372036854775807

- final static long MIN _ VALUE

  Long 的最小值：－9223372036854775808

方法

※ Long (String s) throws NumberFormatException

建立一个新的 Long 对象，值等于以字符串表示的长整型数值。

※ Long (long value)

建立一个新的 Long 对象，值等于给定的初值。

※ double doubleValue ()

以双精度形式返回整型数值。

※ boolean equals (Object anotherObject)

如果另一个对象是 Long 对象并具有同样的数值，返回 true。

※ float floatValue ()

以浮点数形式返回长整型数值。

※ staic Long getLong (String name)

返回 name 表示的系统属性。

※ static Long getLong (String name, long val)

返回 name 表示的系统属性。如果操作失败，返回一个值为 val 的 Long 对象。

※ static Long getLong (String name, Long val)

返回 name 表示的系统属性。如果不能操作，返回 val。

※ int hashCode ()

返回 Long 对象的散列码：它的整型值。

※ int intValue ()

以整型数形式返回长整型数值。

※ long longValue ()

返回对象的长整型值。

※ static long parseLong (String s) throws NumberFormatException

返回字符串 s 所表示的长整型值。

※ static long parseLong (String s, int base) throws NumberFormatException

返回的长整型数等于字符串 s 所表达的数值，base 指定数字的基。也就是说，如果用户想把长整数解析成一个基为 10 的字符串表达式，base 置为 10。

※ static String toBinaryString (long num)

返回 String 用二进制无符号数表示 num。

※ static String toHexString (long num)

返回 String 用十六进制无符号数表示 num。

※ static String toOctalString (long num)

返回 String 用八进制无符号数表示 num。

※ String toString ()

以字符串形式返回长整数的值。

※ static String toString (long val)

以字符串形式返回长整数 val 的值。

※ static String toString (long val, int base)

以字符串形式返回基为 base 的长整数 val 的值。

※ static Long valueOf (String s) throws NumberFormatException

返回一个 Long 对象，其值由指定的字符串决定。

※ static Long valueOf(String s,int base)throws NumberFormatException

返回一个 Long 对象，其值由指定的字符串以 base 为基决定。

Math

Math 类提供了包括范围很广的数学方法，以及诸如 e 和 pi 这样的常数。

字段

- final static double E

  常量 e：2.7182818284590452354

- final static double PI

  常量 pi：3.14159265358979323846。

方法

※ static double abs (double val)

返回双精度型数的绝对值。

※ static float abs (float val)

返回浮点型数的绝对值。

※ static long abs (long val)

返回长整型数的绝对值。

※ static int abs (int val)

返回整型数的绝对值。

※ static double acos (double val)

返回指定数值的反余弦值。

※ static double asin (double val)

以弧度返回指定数值的反正弦值。

※ static double atan (double val)

返回指定数值的反正切值。

※ static double atan2 (double u, double v)

返回由直角坐标 (u, v) 定义的角度值。

※ static double ceil (double val)

返回大于等于 val 的最小整数 (即“最小整数方法”[val])。

※ static double cos (double val)

返回指定弧度数值的余弦值。

※ static double exp (double x)

返回 e^x (译者注：原文为 e`x)。

※ static double floor (double val)

返回小于等于 val 的最大整数 (“即最大整数方法”[val])。

※ static double IEEEremainder (double v1, double v2)

返回运算 v1/v2 的余数。

※ static double log (double val) throws ArithmeticException

返回指定值的数学对数。

※ static double max (double v1, double v2)

返回 v1 和 v2 中的最大数值。

※ static float max (float v1, float v2)

返回 v1 和 v2 中的最大数值。

※ static long max (long v1, long v2)

返回 v1 和 v2 中的最大数值。

※ static int max (int v1, int v2)

返回 v1 和 v2 中的最大数值。

※ static double min (double v1, double v2)

返回 v1 和 v2 中的最小数值。

※ static float min (float v1, float v2)

返回 v1 和 v2 中的最小数值。

※ static long min (long v1, long v2)

返回 v1 和 v2 中的最小数值。

※ static int min (int v1, int v2)

返回 v1 和 v2 中的最小数值。

※ static double pow (double base, double exp) throws ArithmetiException

返回 $base^{exp}$ 的值。

※ static synchronized double random ()

返回运用 java.util.Random 类产生的随机数。

※ static double rint (double val)

返回最靠近的四舍五入整数值，以双精度型数返回（例如：rint (5.3) =5.0)。

※ static long round (double val)

对双精度数进行四舍五入，以长整型数返回（译者注：原文为“浮点数返回”)。

※ static int round (float val)

对指定的数进行四舍五入。

※ static double sin (double val)

返回指定弧度数值的正弦值。

※ static double sqrt (double val) throws ArithmeticException

返回指定数的平方根。

※ static double tan (double val)

返回指定弧度角的正切值。

Abstract Number

Number 类是所有用来操作类似 Integer，Float 和 Double 数据类型值的封装器的基类。虽然 int，double 和 float 是各具优点的数据类型，由于与 Java 中所使用的更丰富的对象相比，它们更

基础一些，因此被认为是“原始数据类型”。虽然这些原始的类型能产生更有效的代码，但在一定的环境中，它们缺少足够的健壮性，此如在 hashtables 中即是如此，这时需要以一个完整的对象对待每一项。更重要的是，每一 Number 类型的类提供一种依赖于类的自然的操作方法。这给用户从诸如 String 等更为复杂的对象中获得原始值以很简捷的手段。

方法

※ Number ()

建一个新的 Number。

※ abstract double doubleValue ()

以双精度值返回 Number 的值。

※ abstract float floatValue ()

以浮点值返回 Number 的值。

※ abstract int intValue ()

以整型值返回 Number 的值。

※ abstract long longValue ()

以长整型值返回 Number 的值。

Object

Object 类是 Java 程序语言中所有类的基类。即使没有声明一个超类，每个类也都将从 Object 类继承。也就是说，该类中的所有方法都可以访问所有的类。但这并不是说用户在实际工作中必须真正使用这些方法。

方法

※ Object ()

建立一个新的 Object。

※ protected Object clone ()throws CloneNotSupportedException

建立一个 Object 的拷贝。

※ boolean equals (Object anotherObject)

如果两个 Object 相等，返回真。

※ protected void finalize () throws Throwable

这个方法将在 Object 被 Java 环境集中起来之前调用。缺省为空，任何类都可以重载这一方法来增加功能。该方法出现异常将终止垃圾收集。

※ final Class getClass ()

以 Class 对象形式返回 Object 的类。

※ int hashCode ()

返回 Object 的散列码。这个值依赖于特定的对象。

※ final void notify ()

通告 (notify) 一个可能重新获得控制的线程 (该线程由 wait () 暂停)。

※ final void notifyAll ()

通告所有可能重新获得控制的线程 (这些线程由 wait () 暂停)。

※ String toString ()

返回类名和它的唯一标志：一个用于区分同一类的不同实例的特定的整数标记。

※ final void wait () throws InterruptedException

通知一个线程一直等待直至被通告。

※ final void wait (long millisec) throws InterruptedException

通知一个线程等待指定的毫秒数，或者到被通告。

※ final void wait (long millisec, int nanos) throws InterruptedException

通知一个线程等待指定的毫秒和毫微秒，或者到被通告。

Abstract Process

Process 类提供一个由 java 程序开始的管理操作的句柄。由 Runtime 类中的 exec () 方法返回，Process 类通常用于监视——而不是控制——正在进行的操作。

▶参见 “Runtime”

方法

※ Process ()

建立一个新的 Process。

※ abstract void destroy ()

终止一个子进程。

※ abstract int exitValue ()

返回 Process 的出口值。如果 Process 未完成则发出非法线程状态异常 IllegalThreadStateException。

※ abstract InputStream getErrorStream ()

返回标准错误输出进程 Process 所使用的流。

※ abstract InputStream getInputStream ()

返回标准输入进程 Process 所使用的流。

※ abstract OutputStream getOutputStream ()

返回标准输出进程 Process 所使用的流。

※ abstract int waitFor () throws InterruptedException

等待直到 Process 完成并且返回出口值。

Runtime

Runtime 类为用户提供了与其程序运行所在系统以及 Java 虚拟机的接口。

注意：虽然 exec () 方法可以在 appletviewer 下运行，但在 Netscape 下会因安全而受到限制。

Runtime 类使用户能够执行一个与提示 prompt 有关的普通任务，比如开始一个文件。在图 20 中，Runtime 类用来启动一个纸牌游戏（程序 13）。

▶参见“Abstract Process”

程序 13　Runtime 例子

```
public class RuntimeExample {
    public static void main(String argv()) {
        Runtime r=java.lang.Runtime.getRuntime();
        try{
            r.exec("\\winnt\\system32\\sol.exe");
        }
```

```
            catch (Exception e) {
                {System.out.println(e.toString());
            }
        }
}
```

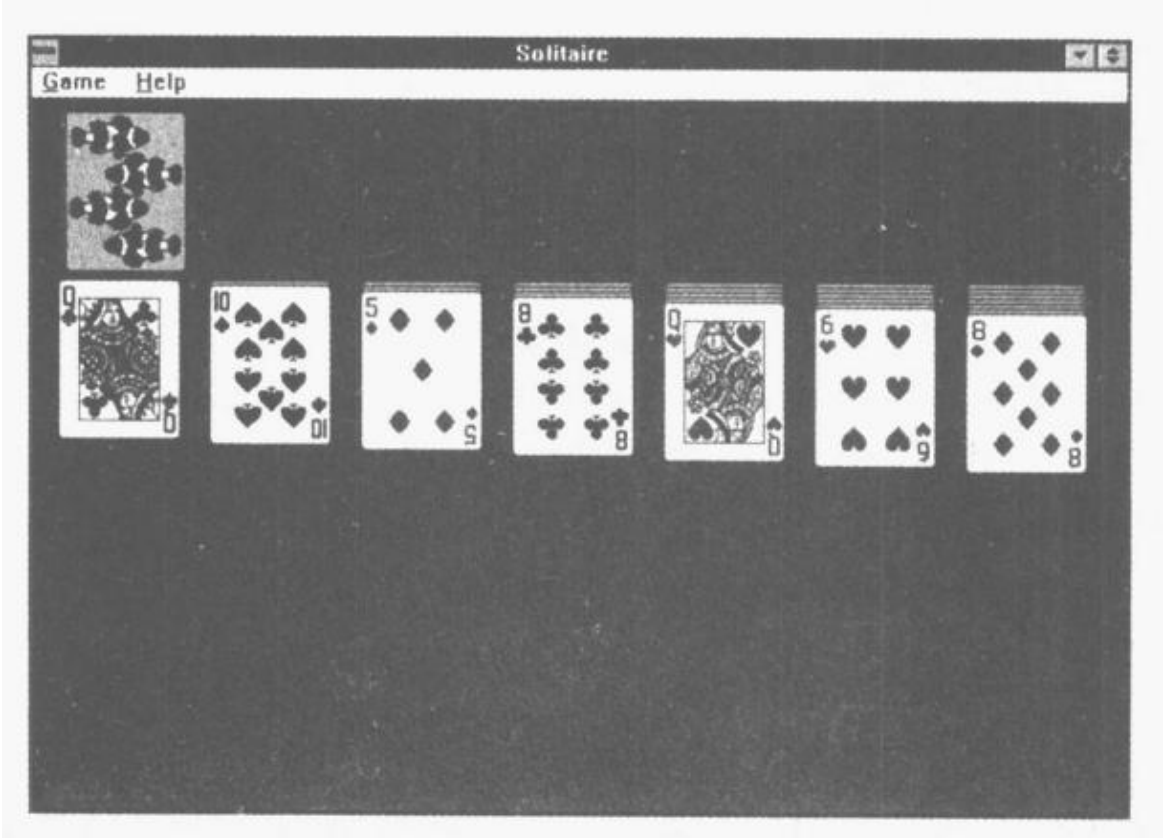


图 20　程序 13 的输出

方法

※ Process exec (String command) throws IOException

运行一个指定的命令。

※ Process exec (String cmdarray []) throws IOException

运行在 cndarray 中建立的命令。

※ Process exec (String cmdarray [], String envp []) throws IOException

使用在 envp 数组中指定的环境参数运行 cmdarray 中的命令。参数的格式为：parameter = value。

※ Process exec (String command, string envp []) throws IOException

使用在envp 数组中指定的环境参数运行给定的命令。参数的格式为：parameter = value。

※ void exit (int status)

终止程序运行并退出 Java 虚拟机。status 为整个操作的退出码。

※ long freeMemory ()

返回内存中的未用字节数。

※ void gc ()

调用垃圾收集器。这个方法释放内存中不再被使用的所有对象——不留对象的索引。垃圾收集工作在不活动期间或内存不够时由系统自动执行。

※ InputStream getLocalizedInputStream (InputStream in)

这个方法的目的是返回一个输入流 InputStream，这个输入流将被翻译成 UNICODE 格式。然而，这一方法仅仅返回 in 参数。

※ OutputStream getLocalizedOutputStream (OutputStream out)

这个方法的目的是返回一个输出流 OutputStream，这个输出流将被翻译成 UNICODE 格式。然而，这一方法仅仅返回 out 参数。

※ static Runtime getRuntime ()

返回当前的 Runtime 对象。当类的构造方法不是公共的时候，必须用这一方法来获得 Runtime。

※ synchronized void load (String filename)

装入一个动态库。

※ synchronized void loadLibrary (String libname)

装入一个动态库。

※ void runFinalization ()

在任何搜集到的对象中运行 finalize () 方法。在垃圾收集期间 finalize 方法自动被调用。

※ long totalMemory ()

返回内存所有字节数。

※ void traceInstructions (boolean on)

如果 on 是 true，将跟踪指令。

※ void traceMethodCalls (boolean on)

如果 on 是 true，将跟踪方法调用。

Abstract SecurityManager

SecurityManager 提供了一个为了监视程序能力创立和配置多种安全检查的手段。当用户认为 Java 小应用程序已经可以在 intranet 和 internet 上运行时，这些检测是必要的。因为在 intranet 和 internet 这样的网络环境下，安全和保密是一个重要的问题。

建立用户自己的安全限定，使这个类子类化并且定义它们的方法(这些方法对用户的程序来讲是重要的)是必要的。可以看到，这个类的大多数方法用来检测一个行为是否有效但没有返回值。然而，当这个行为不被允许时，将产生安全性异常 SecurityException。

字段

• protected boolean inCheck

如果 SecurityManager 正在安全检测中，该值为 true。

方法

※ protected SecurityManager ()

建立一个新的 SecurityManager。

※ void checkAccept (String host, int port)

判定试图在给定端口 port 与指定 host 的连接能否成功。

※ void checkAccess (ThreadGroup g)

检测当前的线程 Thread 能否修改指定的线程组 ThreadGroup。

※ void checkAccess (Thread t)

检测当前的线程 Thread 能否修改指定的线程组 ThreadGroup。如果不能则发出异常处理。

※ void checkConnect (String host, int port)

检测插座是否已经与指定的 host 上的指定的 port 连通，如果没有则出错。

※ void chechConnect(String host,int port,Object executionContext)

检测当前运行的代码和指定的 executionContext 能否与指定的 host 上的指定的 port 连通。如果没有则出错。

※ void checkCreateClassLoader ()

检测 ClassLoader 是否已被建立，如果没有则出错。

※ void checkDelete (String file)

检测指定的文件是否能被删除，如果不能则出错。

※ void checkExec (String cmd)

检测指定的命令是否能被所依托的代码运行，如果不能则出错。

※ void checkExit (int status)

检测 Runtime 系统是否以一定的退出码退出，如果没有则出错。

※ void checkLink (String lib)

检测库是否存在。如果没有则出错。

※ void checkListen (int port)

检测服务器插座是否在指定的端口监听，如果没有则出错。

※ void checkPackageAccess (String packageName)

检测小应用程序能否访问指定的包，如果不能则出错。

※ void checkPackageDefinition (String packageName)

检测小应用程序能否在指定的包中建立类，如果不能则出错。

※ void checkPropertiesAccess ()

检测程序是否已经访问了一系列的系统属性，如果没有则出错。

※ void checkPropertiesAccess (String property)

检测程序是否能够访问一系列的系统属性，如果不能则出错。

※ void checkPropertiesAccess (String property, String default)

检测小应用程序能否访问指定的系统属性,如果不能则出错。当使该类子类化时，如果得不到 property 的值，则使用 default 值。

※ void checkRead (String file)

检测一个小应用程序能否从指定的文件中读，如果不能则出错。

※ void checkRead (FileDescriptor fd)

检测一个小应用程序能否从指定的文件中读，如果不能则出错。

※ void checkRead (String file, Object executionContext)

检测一个小应用程序和指定的 executionContext 能否从指定的文件中读，如果不能则出错。

※ void checkSetFactory ()

检测一个小应用程序能否设置一个对象工厂，如果不能则出错。

※ boolean checkTopLevelWindow (ObjectWindow)

检测一个小应用程序能否建立一个顶层窗口。返回值为 false 表明伴随窗口将显示一个警告信息。

※ void checkWrite (String file)

检测一个小应用程序能否往指定的文件中写，如果不能则出错。

※ void checkWrite (FileDescriptor fd)

检测一个小应用程序能否往指定的文件中写，如果不能则出错。

※ protected int classDepth (String name)

返回指定类名的栈框 (stack frame) 的第一个位置。

※ protected int classLoaderDepth ()

返回指定的类载入器的栈的第一个位置。如果不能则出错。

※ protected ClassLoader currentClassLoader ()

返回在运行栈顶端的当前类的载入器。

※ protected Class [] getClassContext ()

返回类的上下文。

※ boolean getInCheck ()

如果正在进行安全检测，返回 true。

※ Object getSecurityContext ()

返回一个 Object，这个 Object 包含一些关于将来可能被安全检测用到的当前的上下文。一般来讲，该方法返回空串。

※ protected boolean inClass (String className)

如果能在运行栈中找到指定的类 ClassName，则返回 true。

※ protected boolean inClassLoader ()

如果当前的 ClassLoader 非空则返回 true。

String

String 类提供了一个字符串的基本的自然属性和功能。由于 String 的值一旦建立就不能改变，因此该类主要用于分析、操作及取值——而不能重建它们，虽然有一些方法能够在某种程序上修改或增加字符串的值，但这些方法不是修改 String 本身的值，而是建立并返回一个新的程序所要求得到的 String。

方法

※ String ()

建立一个新的 String。

※ String (StringBuffer buffer)

建立一个新的 String，拷贝 buffer 中的内容。

※ String (byte ascii [], int hibyte)

建立一个基于指定数组的新的 String，hibyte 为每个字符的高位字节。

※ String (byte ascii [], int hibyte, int offset, int count)

建立一个基于指定数组的新的 String，offset 指定加入字符串中的第一个元素，count 指定加入的个数，hibyte 指定每个字符的高位字节。

※ String (char value [], int offset, int count)

建立一个基于指定数组的新的String,offset 指定加入字符串中的第一个元素，count 指定加入的个数。

※ String (char value [])

建立一个基于指定数组的新的 String。

※ String (String value)

建立一个新的 String，值已指定。

※ char charAt (int index)

返回在指定位置的字符。

※ int compareTo (String anotherString)

返回一个表示该字符串小于、等于还是大于另一个字符串的整数，如果返回值小于 0，这个字符串小于另一个字符串。如果返回值等于 0，这个字符串等于另一个字符串。如果返回值大于 0，这个字符串大于另一个字符串。

※ String concat (String str)

返回该字符串与另一个 str 的连接值。

※ boolean equals (Object anotherObject)

如果 anotherObject 是一个具有相同值的 String，返回 true。

※ boolean equalsIgnoreCase (Object anotherObject)

不考虑大小写因素比较该 String 与 anotherObject。如果相等返回 true。

※ void getBytes (int srcBegin, int srcEnd, byte dst [], int dstBegin)

将该 String 中从 srcBegin 开始到 srcEnd 结束的字符拷贝到指定的数组中去，从这一数组的 dstBegin 位置开始。

※ void getChars (int srcBegin, int srcEnd, char dst [], int dstBegin)

将该 String 中从 srcBegin 开始到 srcEnd 结束的字符拷贝到指定的数组中去，从这一数组的 dstBegin 位置开始。

※ int length ()

返回 String 的长度。

※ static String copyValueOf (char data [])

返回一个值等于指定字符的 String。

※ static String copyValueOf (char data [], int offset, int count)

返回一个 String，其值等于数组 data [] 中在 offset 和 offset +count 之间的字符部分。

※ boolean endsWith (String suffix)

如果 String 以指定的 suffix 结尾，返回 true。

※ int hashCode ()

返回 String 的散列码———一个基于 String 中的一些（不一定为全部）字符的值。

※ int indexOf (String str)

返回 String 中 str 的第一个索引。

※ int indexOf (String str, int fromIndex)

返回 String 中 fromIndex 之后 str 的第一个索引。

※ int indexOf (int ch)

返回 String 中 ch 的第一个索引。

※ int indexOf (int ch, int fromIndex)

返回 String 中 fromIndex 之后 ch 的第一个索引。

※ String intern ()

返回一个值等于该 String 的 String，新的 String 将从 Hashtahle 中基于该 String 的值获得。

※ int lastIndexOf (String str)

返回 String 中 str 的最后一个索引。

※ int lastIndexOf (String str, int fromIndex)

返回 String 中在 fromIndex 之前 str 的最后一个索引。

※ int lastIndexOf (int ch)

返回 String 中 ch 的最后一个索引。

※ int lastIndex (int ch, int fromIndex)

返回 String 中在 fromIndex 之前 ch 的最后一个索引。

※ boolean regionMatches (boolean ignoreCase, int toffset, String another, int offset, int len)

比较该 String 中与 another 长度相等的一部分的值。如果 ignoreCase 为 true，忽略大小写。len 为要比较的字符数目；toffset 为这个 String 中被比较的第一个字符位置；offset 为 another 中被比较的第一个字符位置。

※ boolean regionMatches (int toffset, String other, int offset, int len)

比较该 String 中与 another 长度相等的一部分的值，忽略大小写。len 为要比较的字符数目；toffset 为这个 String 中被比较的第一个字符位置；offset 为 another 中被比较的第一个字符位置。

※ String replace (char oldChar, char newChar)

将 String 中所有实例的 oldChar 替换成 newChar。

※ boolean startsWith (String prefix)

如果 String 以指定的 prefix 开始，返回 true。

※ boolean startsWith (String prefix, int toffset)

如果从 toffset 开始的 String 的子串等于 prefix，返回 true。

※ String substring (int beginIndex, int endIndex)

返回从 String 的 beginIndex 开始到 endIndex 结束的子串。

※ String substring (int beginIndex)

返回从 String 的 beginIndex 开始到结尾的子串。

※ char [] toCharArray ()

以字符数组的形式返回 String 的值。

※ String toLowerCase ()

返回 String 的一份拷贝，拷贝中所有的字符为小写。

※ String toString ()

返回 String。

※ String toUpperCase ()

返回 String 的一份拷贝，拷贝中所有的字符为大写。

※ String trim ()

返回去掉首尾空白的 String 的拷贝。

※ static String valueOf (double d)

返回一个含有 double 的 String。

※ static String valueOf (float f)

返回一个含有 float 的 String。

※ static String valueOf (long l)

返回一个含有 long 的 String。

※ static String valueOf (int i)

返回一个含有 int 的 String。

※ static String valueOf (char c)

返回一个含有 char 的 String。

※ static String valueOf (boolean b)

返回一个含有 boolean 的 String。

※ static String valueOf (char data [], int offset, int num)

返回一个 String，其值从 data [] 数组的 offset 位置开始，包括 num 个字符。

※ static String valueOf (char data [])

返回一个 String，其值为 data [] 中的字符。

※ static String valueOf (Object obj)

返回一个 String，其值为 Object 的字符串表示。该值通过调用 obj.toString () 获得。

StringBuffer

StringBuffer 是一个动态的字符串类，使用户能够修改和增加字符串的值。它的强有力的功能表现为：用来建立字符串的值——尤其那些从一个孤立的源获得的字符串，例如程序 14 和程序 15 所示范的一个插座（socket）连接或小应用程序的标记参数。

程序 14　运用小应用程序标记的参数—— HTML StringBuffer 示例

<HEAD>

```
<TITLE>StringBuffer Example</TITLE>

<BODY>

<applet code="sbufe.class" width=300 height=300>
<param name=linenum0="This is the">
<param name=linenum1="message I'm">
<param name=linenum2="tring to show.">
</applet>
</BODY>
```

程序 15　StringBuffer 示例

```
import java.applet.Applet;
import java.awt.Graphics;

public class StringBufferExample extends Applet{
    String message;

    /* This method parses the applet parameters. and returns
       them in one string separated by spaces */
    public void init() {
        StringBuffer mes;
        String val;
        int i=0;

        mes=new StringBuffer();

        do{
            String paramName="linenum"+i++;
            val=getParameter(paramName);
```

```
            if(val! =null) {          //continues until end
                mes.append(val)  //adds the message
                mes.append(' ');//adds the space
            }
        }while(val! =null)
        message=mes.toString();//saves the message
                                //as a String
    }

    public void paint(Graphics g) {
        g.drawString(message,10,10);
    }
}
```

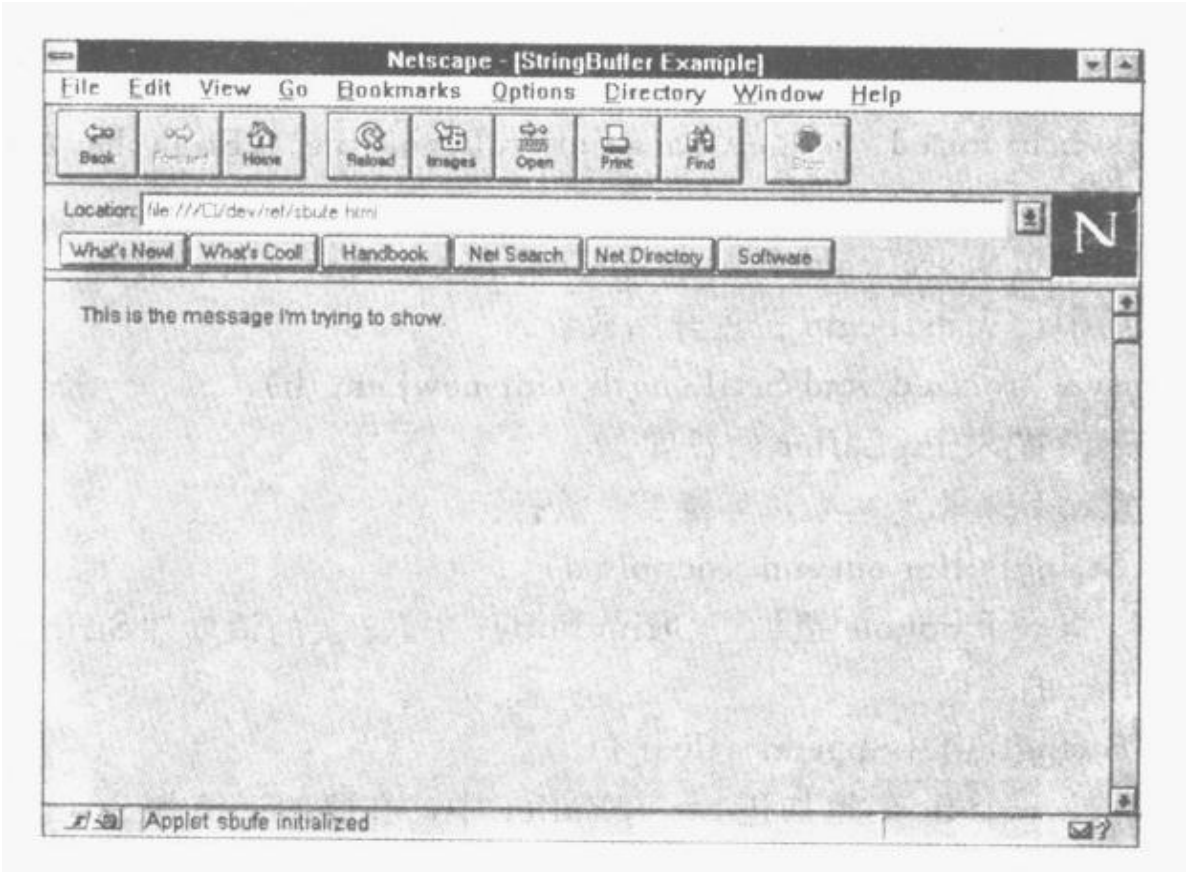


图 21　当使用程序 14 时程序 15 的运行输出

方法

※ StringBuffer (String str)

基于指定的 String 建立一个 StringBuffer。

※ StringBuffer (int length)

建立一个具有指定长度的 StringBuffer。

※ StringBuffer ()

建立一个初始长度为 16 个字符的 StringBuffer。

※ int capacity ()

返回不需申请附加内存即可以加到 StringBuffer 中的字符数目。

※ synchronized char charAt (int index)

返回指定索引处的字符。

※ synchronized void ensureCapacity (int minimumCapacity)

如果必要的话增加 StringBuffer 的长度，以确使 minimumCapacity 个字符可以被加入到 StringBuffer 中。

※ synchronized void getChars (int srcBegin, int srcEnd, char dst[], int dstBegin)

拷贝从 srcBegin 开始到 srcEnd 结束的所有字符，放在 dest 数组中，从 dstBegin 位置开始放置。

※ synchronized void SetlLength (int newLength)

设置 StringBuffer 的长度。

注意：如果长度减小，可能丢失数据。

※ StringBuffer append (double d)

将一个 double 追加进 StringBuffer 中，必要的话增加 StringBuffer 的长度。

※ StringBuffer append (float f)

将一个 float 追加进 StringBuffer 中，必要的话增加 StringBuffer 的长度。

※ StringBuffer append (long l)

将一个 long 追加进 StringBuffer 中，必要的话增加 String-

Buffer 的长度。

※ StringBuffer append (int i)

将一个 int 追加进 StringBuffer 中，必要的话增加 StringBuffer 的长度。

※ synchronized StringBuffer append (char c)

将一个 char 追加进 StringBuffer 中，必要的话增加 StringBuffer 的长度。

※ StringBuffer append (boolean b)

将一个 boolean 追加进 StringBuffer 中，必要的话增加 StringBuffer 的长度。

※ synchronized StringBuffer append (char str [], int offset, int len)

将一个字符序列追加入 StringBuffer 中，必要时增加 StringBuffer 的长度。

※ synchronized StringBuffer append (char str [])

将一个字符数组追加入 StringBuffer 中，必要时增加 StringBuffer 的长度。

※ synchronized StringBuffer append (String str)

将一个 String 追加入 StringBuffer 中，必要时增加 StringBuffer 的长度。

※ synchronized StringBuffer append (Object obj)

将一个 Object 追加入 StringBuffer 中，必要时增加 StringBuffer 的长度。被加入的值等于用 String. valueOf (obj) 方法的返回值。

※ StringBuffer insert (int offset, double d)

在 StringBuffer 的指定的 offset 位置插入一个 double，必要时增加 StringBuffer 的长度。

※ StringBuffer insert (int offset, float f)

在 StringBuffer 的指定的 offset 位置插入一个 float，必要时增加 StringBuffer 的长度。

※ StringBuffer insert (int offset, long l)

在 StringBuffer 的指定的 offset 位置插入一个 long，必要时增加 StringBuffer 的长度。

※ StringBuffer insert (int offset, int i)

在 StringBuffer 的指定的 offset 位置插入一个 int，必要时增加 StringBuffer 的长度。

※ synchronized StringBuffer insert (int offset, char c)

在 StringBuffer 的指定的 offset 位置插入一个 char，必要时增加 StringBuffer 的长度。

※ StringBuffer insert (int offset, boolean b)

在 StringBuffer 的指定的 offset 位置插入一个 boolean，必要时增加 StringBuffer 的长度。

※ synchronized StringBuffer insert (int offset, char str [])

在 StringBuffer 的指定的 offset 位置插入一个字符数组，必要时增加 StringBuffer 的长度。

※ synchronized StringBuffer insert (int offset, Object obj)

在 StringBuffer 的指定的 offset 位置插入一个 Object，必要时增加 StringBuffer 的长度。被加入的值等于由 String. valueOf (obj) 方法返回的值。

※ synchronized StringBuffer insert (int offset, String str)

在 StringBuffer 的指定的 offset 位置插入一个 String，必要时增加 StringBuffer 的长度。

※ int length ()

返回 StringBuffer 的长度。

※ synchronized void setCharAt (int index, char ch)

将指定的 index 位置的字符设置为 ch。

※ String toString ()

以 String 的形式返回 StringBuffer 的值。

System

Java 程序，尤其是小应用程序，由 Java 虚拟机以一种方式操

作，而这种方式对程序员隐藏了大量的机能。这主要是由于 Java 的目的在于完全的平台无关性。然而，通过结合使用 Runtime 类和 System 类，能够获得并控制一些与系统相关的功能——比如收集垃圾——都以一种系统无关的方式进行。

运用系统类，不需要（事实上也不可能）建立 System 对象。然而由于该类中所有的方法都是静态的，所以可以使用程序 16（图 22）所示的方法。

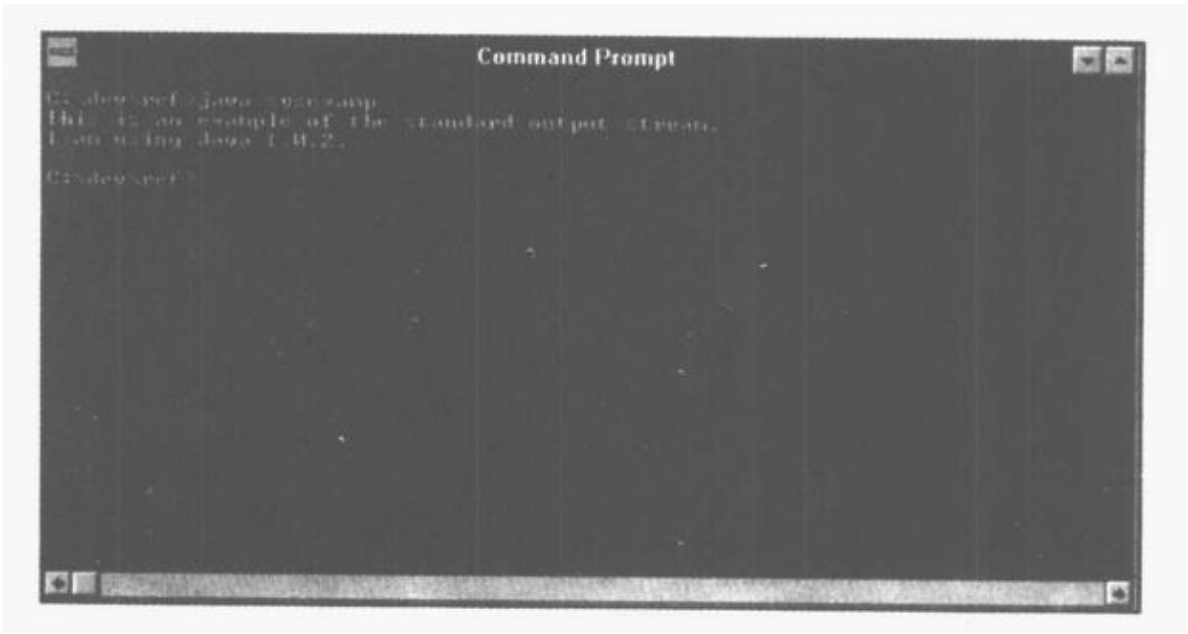


图 22　程序 16 的输出

程序 16　System 示例

```
public class SystemExample {

        public static void main(String argv()) {

            System.out.print("This is an example of the
standard");
            System.out.println("output stream");

             String s=System.getProperty("Java.version");
            System.out.println("I am using Java "+s +
```

```
".");
        }
    }
```

字段

- static PrintStream err

  打印错误信息的标准流。通常与 out 相同。

- static InputStream in

  标准输入流。

- static PrintStream out

  标准输出流。一般是提示或 Netscape 的 Java 控制台。

方法

※ static void arraycopy(Object src,int src _position,Object dst, int dst _posistion, int length)

从 src [src _position] 处开始，拷贝 length 个元素到 dst [dst _position]。

※ static SecurityManager getSecurityManager ()

返回该上下文的安全管理对象。这个对象可以借助 SecurityManager 接口来处理。

※ static void setSecurityManager (SecurityManager s)

设置安全管理者。这个方法只能被调用一次。

※ static void exit (int status)

使虚拟机器用指定的退出码退出。

※ static void gc ()

开始收集垃圾进程。这不会造成马上进行收集垃圾。

※ static String getenv (String name)

这个方法不再被支持。

※ static Properties getProperties ()

返回一个包含系统属性的 Properties 对象。

※ static String getProperty (String propName, String def)

返回由 propName 指定的属性。如果这个属性未定义，返回字

符串 def。

※ static String getProperty (String key)

返回由 key 指定（译者注：原文为“由 propName 指定”）的属性。如果这个属性未定义，返回空串。

※ static void load (String filename)

装入指定的库。filename 提供库的完整的路径和文件名。

※ static void loadLibrary (String libname)

装入指定的库。libname 代表库名。

※ static void runFinalization ()

在 Object 以任何方式收集垃圾时调用 finalize()方法。在 Object 被搜集前 finalize () 方法被自动调用。本方法强制它们立刻被调用。

※ static void setProperties (Properties props)

设置系统属性为指定的一组属性。

Thread Implements Runnable

线程是 Java 语言中一个重要的组成部分。线程是可以同时运行的独立进程。通过建立多线程，用户就可以同时进行多任务。

线程特别有效的用途是处理多个输入源。例如，用户可以建立一个小应用程序，同时从用户和另一个输入流接收数据，比如这个输入流可以是网络插座 (networked socket)。这样一来，就可以像平常一样建立一个小应用程序处理用户输入，然后建立另一个线程读取和解析来自流的信息。

程序 17 和图 23 示范了一个简单线程类 Counter 的应用。当 threadex 类等待用户输入时，Counter 类在后台进行计算。因而，当用户按了回车键，getCount () 方法返回值的产生非常迅速。

尤其请注意用来建立和开始 Counter 类的进程。开始运行一个线程，用户必须调用 start () 方法。甚至，只有线程类的 run () 方法才有能力同时运行。

▶参见“Runnable”

```
public class ThreadEx{

    public static void main(String argv[]){
        Counter c=new Counter();
        c.start(); //开始计数

        for(int i=1;i<=5;i++){
            try{
                System.in.read();
            }
            catch(Exception e){
                System.out.println(e.toString());
            }
            System.out.println(c.getCount());
        }           //循环结束

        c.stop();
        System.exit(1);
    }
}

class Counter extends Thread {
    private int count;

    public void run(){
        count=0;
        while(true)     //建立一个连续的循环
            count++;
    }
    public int getCount(){
```

```
            return count;
        }
    }
```

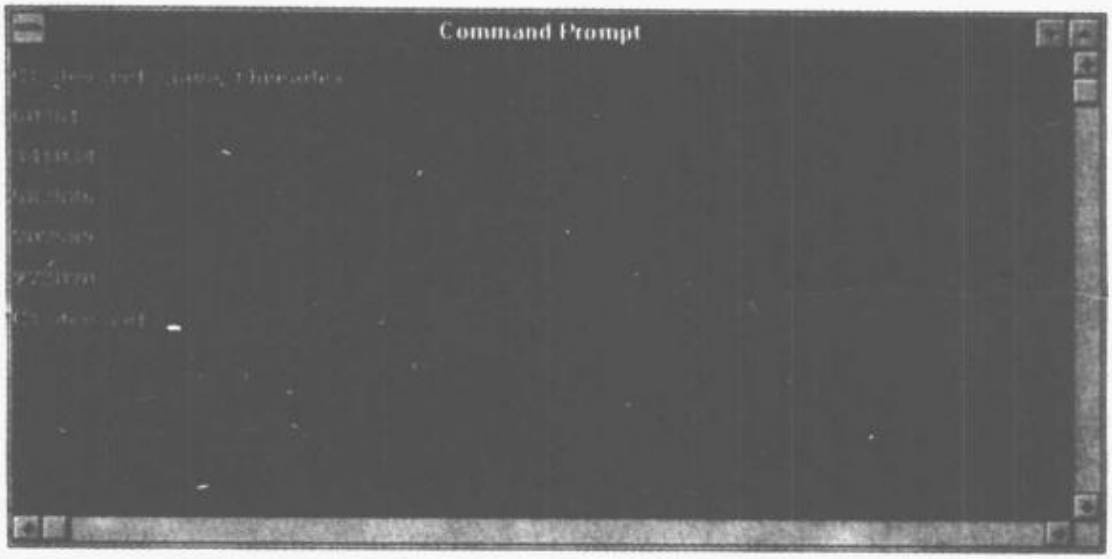


图 23　程序 17 的输出

字段

- final static int MAX _ PRIORITY

  线程的最大优先权，这将影响接收并运行程序的时间。

- final static int MIN _ PRIORITY

  线程的最小优先权，这将影响接收并运行程序的时间。

- final static int NORM _ PRIORITY

  线程的一般优先权，这将影响接收并运行程序的时间。

方法

※ Thread ()

建立一个新的线程。

※ Thread (Runnable target)

建立一个新的线程，该线程将使用指定（target）类的 run () 方法。该类必须执行 Runnable 接口。

※ Thread (Runnable target, String name)

建立一个新的线程，它具有指定的名字，并使用指定

(target) 类的 run () 方法。这个类必须执行 Runnable 接口。

※ Thread (ThreadGroup group, Runnable target)

建立一个新的线程，该线程将使用指定 (target) 类的 run () 方法。该类必须执行 Runnable 接口。新的线程类将放在指定的线程组中。

※ Thread (ThreadGroup group, Runnable target, String name)

建立一个新的线程，它具有指定的名字，并使用指定 (target) 类的 run () 方法。这个类必须执行 Runnable 接口。新的线程类将放在指定的线程组中。

※ Thread (ThreadGroup group, String name)

建立一个新的线程，它具有指定的名字，并将放在指定的线程组中。

※ Thread (String name)

建立一个具有指定名字的线程。

※ static int activeCount ()

返回当前线程组中活动线程数目。

※ void checkAccess ()

如果用户能够修改线程，返回 true。依赖于 SecurityManager 的返回值。

※ int countStackFrames ()

返回线程中的栈框数目。

※ static Thread currentThread ()

返回一个运行 Thread 对象的参考。

※ void destroy ()

不运行任何清除程序而破坏 Thread。

※ static void dumpStack ()

(译者注：原文无注解)

※ static int enumerate (Thread tarray [])

将当前线程所在线程组中所有活动线程放在数组中。返回实

际放入数组中线程数。

※ final String getName ()

返回当前线程的名字。

※ final int getPriority ()

返回当前线程的优先权。

※ final ThreadGroup getThreadGroup ()

返回当前线程所在的线程组。

※ void interrupt ()

中断当前线程。这个方法必须被重载，否则将返回一个异常NoSuchMethodException。

※ static boolean interrupted ()

用来确定当前的线程是否被中断。这个方法必须被重载，否则将返回一个异常 NoSuchMethodException。

※ boolean isinterrupted ()

用来确定是否有另一个线程被中断。这个方法必须被重载，否则将返回一个异常 NoSuchMethodException。

※ final boolean isDaemon ()

如果当前线程是一个守护程序 (daemon)，返回 true。

※ final void join () throws InterruptedException

无限等待直到线程结束。

※ final synchronized void join (long millis, int nanos) throws InterruptedException

等待到指定时间线程结束。它运用 java. long. wait ()。

※ final synchronzed void join (long millis) throws InterruptedException

等待指定时间到线程结束。它运用 java. long. wait ()。

※ final void resume ()

重新开始一个被挂起的线程。

※ void run ()

线程运行的主体。虽然 run () 方法可以使用其他方法，但它

是可以和其他操作同时运行的唯一方法。当另一个操作发生时，该线程中任何要运行的方法都要从 run（）方法调用。

※ final void setDaemon（boolean on）

如果 on 为 true，这个线程为“守护程序化的”，允许从 Java 操作中在后台独立运行。

※ final void setName（String name）

设置线程的名字。

※ final void serPriority（int newPriority）

设置线程的优先级。

※ static void sleep（long millis，int nanos）throws InterruptedException

使一个线程暂停一定的时间。

※ static void sleep（long millis）throws interruptedException

使一个线程暂停一定的时间。

※ synchronized void start（）

使 run（）方法开始。

※ final synchronized void stop（Throwable obj）

终止线程，指定一个 Throwable 对象作为终止原因。

※ final void stop（）

以 ThreadDeath 实例作为参数调用 stop（throwable）方法。

※ final void suspend（）

挂起当前线程。唤醒它用 resume（）方法。

※ static void yield（）

终止任何等待线程。

※ String toString（）

返回一个包含线程名字、属性和线程组的字符串。

ThreadGroup

由于线程一定程度上为独立进程，为了更好地管理它们常常要加一些限制。ThreadGroup 类使用户能够建立线程集合体。进一步讲，它能使用户建立层次结构的 ThreadGroup，通过这种层

次结构，一个线程可以访问在同一组中所有的线程，而其他线程不会在当前线程所处的层次之上。

方法

※ ThreadGroup (String name)

建立一个具有指定名字的 ThreadGroup。当前的 ThreadGroup 将成为新线程组的父线程组。

※ ThreadGroup (ThreadGroup parent, String name)

建立一个具有指定名字和父线程组的新线程组。

※ synchronized int activeCount ()

返回当前线程组中活动线程数目。

※ synchronized int activeGroupCount ()

返回当前线程组中活动线程组数目。

※ final void checkAccess ()

询问当前线程能否修改该线程组。使用当前 SecurityManager 中的 checkAccess () 方法。

※ final synchronized void destroy ()

破坏掉线程组，但不影响各个线程。

※ int enumerate (Thread list [])

将线程组中的每一个活动线程的索引放在 list [] 数组中。返回数组中的元素个数。

※ int enumerate (Thread list [], boolean recurse)

将线程组中的每一个活动线程的索引放在 list [] 数组中。如果 recurse 为 true，本方法将把该线程组中的子线程组中的线程也转化并加入索引数组中。返回数组中的元素个数。

※ int enumerate (ThreadGroup list [])

将该线程组中的每一个活动的 threadGroups 的索引放在 list [] 数组中。返回数组中的元素个数。

※ int enumerate (ThreadGroup list [], boolean recurse)

将该线程组中的每一个活动的 threadGroups 的索引放在 list [] 数组中。如果 recurse 为 true，本方法将把该线程组中的子线

程组中的线程组也转化并加入索引数组中。返回数组中的元素个数。

※ final int getMaxPriority ()

返回该组中所有方法的最大优先权。

※ final String getName ()

返回组的名字。

※ final ThreadGroup getParent ()

返回组的父线程组。

※ final boolean isDaemon ()

返回该线程组中 daemon 标志值，由该组中的一个线程创建的标志，组中的所有线程继承相同的标志。

※ synchronized void list ()

将该线程组中的所有线程和线程组打印输出至 System. out。

※ final boolean parentOf (ThreadGroup anotherGroup)

如果该线程组是 anotherGroup 的父线程组或二者相同，返回 true。

※ final synchronized void resume ()

调用属于该线程组及它的子线程组的所有线程的 resume () 方法。

※ final void setDaemon (boolean daemon)

为该线程组设置 daemon 标志。

※ final synchronized void setMaxPriority (int level)

为在该组中建立的所有新线程设置最大优先级。

※ final synchronized void stop ()

调用属于该线程组及它的子线程组的所有线程的 stop () 方法。

※ final synchronized void suspend ()

调用属于该线程组及它的子线程组的所有线程的 suspend () 方法。

※ String toString ()

返回包含该组名字和它的最大优先级的字符串。

※ void uncaughtException (Thread t , Throwable e)

由系统调用处理一个非捕捉错误。

Throwable

Throwable 对象可以由一个方法抛出。Throwable 类为由方法抛出的所有对象定义了一组行为。所有的异常和错误都是从 Throwable 类继承的。

▶参见 “Exception”

▶参见 “Errors”

方法

※ Throwable ()

建立一个新的 Throwable 对象。

※ Throwable (String message)

建立一个具有指定错误信息的 Throwable 对象。

※ Throwable fillInStackTrace ()

为该 Throwable 对象填充栈框。

※ String getMessage ()

返回这一对象的详细信息。这只能在构造方法中设置。

※ void printStackTrace ()

将栈的路径输出到标准错误输出流 (System. err)。当用户未捕捉到错误时，这将在浏览器上显示。这对发现错误源是非常有用的，类似于下面所示：

```
java. lang. ArithmeticException: /by zero
        at appletname. init (appletname. java: 6)
        at sun. applet. AppletPanel. run (AppletPanel. java: 243)
        at java. lang. Thread. run (Thread. java: 289)
```

※ void printStackTrace (PrintStream s)

将本地路径打印输出到指定的打印流 PrintStream 上。

※ String toString ()

返回一个 String，包括 Throwable 对象的名字和它的错误信息（如果有的话）。

2. 接口

Cloneable

虽然这个接口不包括任何方法，但它的确提供了很大的灵活性。它作为一个 Object 类的标志存在。如果所用的一个类执行了 Cloneable 接口，用户便可建立这一对象的拷贝。

Runnable

Runnable 接口是表现接口功能强大的一个主要例子。通过执行 Runnable 接口，用户可以建立一个类——通常是小应用程序——它可以作为一个线程。

程序 18 和程序 19 中，建立了一个用来作消息卷滚器的简单的小应用程序类（参见图 24）。使用在浏览窗口底部的状态条，用

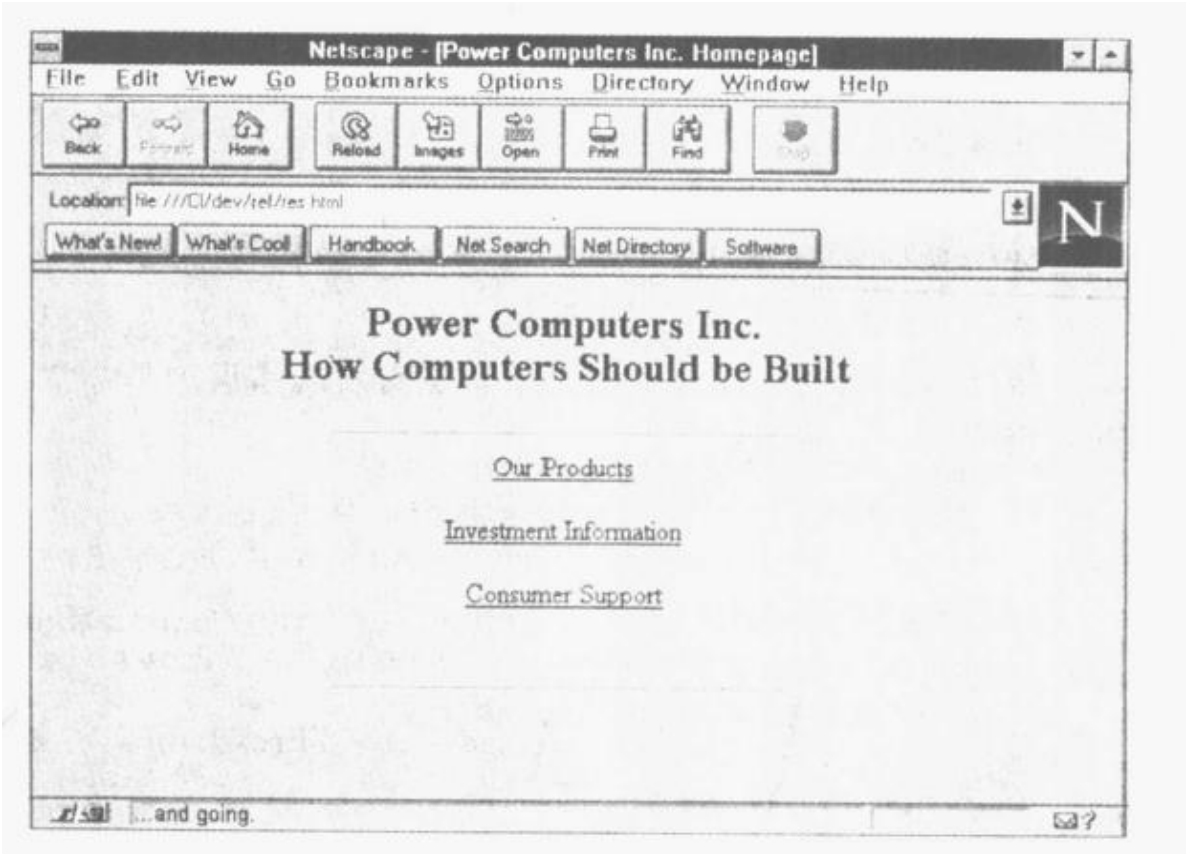


图 24　程序 18 与程序 19 的输出屏幕

户可以在一系列的消息中滚动，这些消息从小应用程序标记参数获得。

▶参见“Thread Implements Runnable”

程序 18　可运行的例程——HTML

```
<HTML>
<HEAD>

<TITLE>Power Computers Inc. Homepage</TITLE>
</HEAD>

<center>
<h2>Power Computers Inc. <br>
How Computers Should be Built</h2>

<hr width=45%>

<a href="\products. html">Our Products</a><p>
<a href="\stocks. html">Investment Information</a><p>
<a href="\operators. html">Consumer Support</a><p>
</center>
<applet code="runne. class" width=0 height=0 >
<param name=linenum0 value="This is the message. ">
<param name=linenum1 value="It keeps going,">
<param name=linenum2 value="and going. . . ">
<param name=linenum3 value=". . . and going. ">
</applet>
<hr width=45%>
</BODY>
</HTML>
```

程序 19　可运行的例程——java Code

```
import java.applet.Applet;

public class RunnableEx extends Applet implements Runnable
{
    Thread display;
    int pos,max_mes;
    String message[];

    /* This method obtains the applet parameters and stores
them in message[]. It also sets the max_mes value */

    public void init(){
        String val;
        int i=0;

        message=new String[30];
                //can accept up to 30 messages
        do{
            String paramName="linenum"+i;
            val=getParameter(ParamName);
            if(val != null){
                message[i]=val;
                i++;
            }
        }while (val! =null);
        max_mes=i-1;/* we don't want to display the null
message that caused the do loop to terninate */
        pos=0;

        for(int j=0;j<=max_mes;j++)
```

```
        System.out.println(j+message[j]);
        }

        /* This method creates and begins the thread. */
        public void start( ){
            if (display==null)
                display=new Thread(this);
                display.start( );
        }

        public void stop( ){
            if(display! ==null){
                display.stop( );
                display=null;
            }
        }

        public void run( ){
        /* We don't want the scroller to receive too much attention. */
         Thread.currentThread( ).setPriority(Thread.MIN_PRIORITY);

        while (display! =null){
            getAppletContext( ).showStatus(message[pos++]);
            if(pos>max_mes)
                pos=0;

                try{
```

```
                    Tread.sleep(1000);//pauses
                }
                catch(Exception e){
                    System.out.println(e.toString( ));
                }//end catch
            }//end while
        }//end method
    }//end class
```

由于简单的原因，String 数组通常用来存储文本行。如果要创建更加有效和灵活的模式，那么可取的方法是采用更具适应性的结构比如 Vector。

方法

※ abstract void run ( )

六、JAVA.NET

因为 Java 是一种基于网络的语言，这个相对较小的软件包是非常有用的。最重要的是，它通过创建一些接口或与一些接口的连接，或者通过利用 URL 和 Internet 地址，具有与其他信息源通信的能力。

1. 类

Abstract ContentHandler

此类用于自 URL 返回一个特定类型的对象。当在 URL 中找到一个适当的对象时，它被 ContentHandlerFactory 调用。

方法

※ContentHandler（ ）

创建一个新 ContentHandler 对象。

※abstract Object getContent（URLConnection contentSource）throws IOException

如果 contentSource 包含一个特殊的 MIME 类型，此方法则解析 contentSource 中信息，并取得一个特殊对象。

DatagramPacket

DatagramPacket 是一个能够在网络上传输的信息束，这样一个对象封装了许多关于此信息束的信息，包括内含信息、本身长度、它自何处发送。

下面是类中的私有字段：

```
private byte [] buf;
private int length;
private InetAddress address;
private int port;
```

这些字段不能被外部代码访问；然而，了解 DatagramPacket 包含与不包含哪些字段是有益的。

▶参见“DatagramPacket”。

方法

※DatagramPacket (byte buf [], int length)

创建一个接收信息的 DatagramPacket。包中数据将被存储在 buf [] 数组。length 指定欲接收的字节数，地址被设为空，端口被设为－1。

※DatagramPacket(byte buf[],intlength,inetAddress destAddr, int destPort)

创建一个发送信息的 DatagramPacket 。此信息包包括 buf []数组的最初字节长度。destAddr 和 destPort 指定此数据包的接收者。

※inetAddress getAddress ()

取得地址字段——发送信息的目的地址或接收信息的源地址。

※byte [] getData ()

取得数据包中的信息——已被接收的信息或将要发送的信息。

※int getLength ()

取得在构造器方法中设定的 length 字段。

※int getPort ()

取得数据包的目的 port 字段或源 port 字段

DatagramSocket

DatagramSocket 是一个用于传输 DatagramPacket 的插座。DatagramSocket 不同于标准的 Socket，它没有错误检查，不能保证所有数据包被完整接收或发送。因此，它被认为是不可靠的。

▶参见“DatagramePacket”

方法

※ DatagramSocket (int port) throws SocketException

创建一个 DatagramSocket 来监听约定端口。

※DatagramSocket （） throws SocketException

创建一个由系统指定端口号的 DatagramSocket。

※protected synchronized viod finalize （）

用于执行清除，换句话说即关闭 socket。

※int getLocalPort （ ）

返回系统中与 socket 锁定的端口。

※sychronized void receive (DatagramPacket p) throws IOException

等待直到从流中读取一个可用的 DatagramPacket。

※void send （DatagramPacket p） throws IOException

发送一个穿过流的 DatagramPacket。

Final InetAddress

InetAddress 是处理 Internet 地址的一种简便方式。它能够像存储更为友好的主机名字一样存储原始 IP 地址。

方法

※static sychronized InetAddress [] getAllByName (String hostName) throws UnknownHostException

返回所有指定 hostname 地址。

※statec InetAddtess getLocalHost ( ) throws UnknownHostException

返回一个代表本地主机的 InetAddress 对象。

※boolean equals （Object anotherObject）

如果 anotherObject 与 InetAddress 具有同样的地址，则返回真值。

※byte [] getAddress （ ）

返回此 InetAddress 的原始 IP 地址。返回的数组包括 4 个字节。

※ static synchronized InetAddress getByName （String hostName) throws UnknownHostExction

返回指定 hostName 的单一地址。

※String GetHostName（）

返回此 InetAddress 的主机名字。

※int hashCode（）

返回此 InetAddressd 的 hashcode 值：IP 地址。

※String toString（）

返回一个包含主机名字和 InetAddress 的 IP 地址的字符串。

ServerSocket

ServerSocket 创建一个服务器端的插座，用于通信。缺省情况下基于 PlainSocket SocketImpl 创建，但它没有安全检查。然而，可为 ServerSocket 指定 socketImpl 来加强安全限制。

方法

※ServerSocket（int port）throws IOException

利用当前的 SocketImplFactory 基于指定端口创建一个新 Socket。

※ServerSocket（int prot，int count）throws IOException

利用当前的 SocketImplFactory 基于指定端口创建一个新 Socket。count 指 Socket 等待连接的时间总和。

※Socket accept（）throws IOException

它等待一个请求，然后适当处理连接请求，返回包含实际连接的 Socket。

※void close（）throws IOException

关闭插座连接。

※inetAddress getinetAddress（）

返回连接插座的 InetAddress。

※int getLocalPort（）

返回插座正在侦听的端口号。

※String toString（）

和取得本地端口号一样，取得一个包含约定插座的地址和端口。

※static synchronized void setSocketFactory (SocketImplFactory fac) throws IOException

为 ServerSocket 对象设置 SocketImplFactory，所有将来用此 ServerSocket 对象创建的 socket 将用到指定的 SocketImplFactory。此方法对于一个 ServerSocket 对象只许用一次。

Socket

Socket 创建一个客户端的插座，用于通信。若缺省则基于 PlainSocket SocketImpl 创建，但它没有安全检查。然而，可为 Socket 指定 socketImpl 来加强安全性。使用一个 Socket，一定要考虑正在和谁通话。一般而言，要与特定端口的 socket 服务器通信。在这种情况下，须注意是否要向服务器写东西？用什么协议？主机名字和端口是否不变？

此外，必须记住使用方法 getInputStream ( ) 和 getOutputStream ( ) 与 socket 通信，因为这些流在 sun 程序包之外是不可访问的。

程序 20 中的代码举例说明了一个标准 socket 通信的建立。如果连接已经成功，DataInputStream 的输入与 OutputStream 的输出能够使你自由地通过 socket 进行通信。

程序 20　Socket 例子

```
import java.net.Socket;
import java.io.*;
public class Client {
    String host= "host.name.com";
    int port=1500;

    try {
        me=new Socket (host, port);
        in=new DataInputStream (me.getInputStream ());
        out=me.getOutputStream ();
    }
```

```
    catch (Exception e) {
        System.out.prinln (e.toString ( ));
    }
    .
    .
    .
}
```

方法

※Socket(String host,int port) throws UnknownHostException, IOException

创建一个 Socket 并试图将它连接到指定主机的指定端口。

※Socket (inetAddress host, int port) throws IOException

创建一个 Socket 并试图将它连接到指定主机的指定端口。

※Socket(String host,int port,boolean stream) throws IOException

创建一个 Socket 并试图将它连接到指定主机的指定端口。若 stream 为真，那么信息将作为流被读取。若 stream 为假，那么它将像 DatagramSocket 一样在程序包中被读取。

※Socket (inetAddress address, int port, boolean stream) throws IOException

创建一个 Socket 并试图将它连接到指定主机的指定端口。若 stream 为真，那么信息将作为流被读取。若 stream 为假，那么它将像 DatagramSocket 一样在程序包中被读取。

※synchronized viod close ( ) throws IOException

关闭 socket。

※inetAddress getinetAddress ( )

返回连接 socket 的主机的 InetAddress。

※inputStream getinputStream ( ) throws IOException

返回此 socket 的 InputStream。如果想从流中读取信息，此方

法是必要的。

※int getLocalPort（）

返回连接 socket 的本地端口。

※OutputStream getOutputStream（）throws IOException

返回 Socket 的 OutputStream。若想在流中写信息，此方法是必要的。

※int getPort（）

返回连接 socket 的远程机器端口。

※ static synchronized void setSocketImplFactory（SocketImplFactory fac）throws IOException

为对象设置 SocketImplFactory。此对象创建的 Socket 将基于此 Factory。此方法只能被调用一次。

※String toString（）

返回一个字符串，已包含本地端口，插座所连接的主机和远程端口信息。

Abstract SocketImpl

SocketImpl 类定义了 socket 之间一组必要的通信行为。这些功能对于插座来说通信是整体性的，并且在类 Socket 和 ServerSocket 中应用。然而多数程序员将会发现不需显式地用它们。

字段

- protected inetAddress address

连接插座的地址。

- protected FileDescriptor fd

当读写一个插座时被内部使用，FileDescriptor 对象被内在应用。因为插座接口是基于 FileInputStream 和 FileOutputStream 类。

- protected int localport

被插座使用的本地端口。

- protected int port

连接插座的远程系统端口。

方法

※SocketImpl（）

创建一个新 SocketImpl。

※protected abstract void accept (SocketImpl connedtion) throws IOException

接收指定的连接。

※protected abstract int available（）throws IOException

返回流中可用字节数。

※protected abstract void bind (inetAddress hosr, int port) throws IOException

把插座约束到指定主机和端口。

※protected abstract void close（）throws IOException

关闭插座。

※protected abstract void connect (inetAddress host, int port) throws IOException

将插座与指定主机和端口连接。

※protected abstract void connect (String host, int port) throws IOException

将插座与指定主机和端口连接。

※protected abstract void create (boolean stream) throws IOException

创建一个插座，若 stream 为真，插座将像读取流一样读取信息；若 stream 为假，那么将在程序包中读取信息。

※FileDescriptor getFileDescriptor（）

返回插座的 FileDescriptor 对象，此描述器用于创建流的交互作用。

※protected InetAddtess getInetAddress（）

返回连接插座的 InetAddress。

※protected abstract InputStream getInputStream（）throws IOException

返回将从插座流中获取信息的 InputStream。

※protected int getLocalPort （ ）

返回连接到插座上的本地端口号。

※protected abstract OutputStream getOutputStream （ ） throws IOException

返回将向插座流中写信息的 Outputstream。

※protected int getPort （ ）

取得连接到 socket 上的端口号。

※protected abstract void listen （int time） throws IOException

在指定时间段内侦听插座，等待连接请求。

※String toString （ ）

像返回本地端口号一样，返回一个包含连接插座的地址和端口的字符串。

URL

URL 类具有双重功能。最简单地，它能处理 URL 运算的语义，例如它能剖析 URL 的文件或主机名。而较为重要的功能是它也可作为检索存储在指定 URL 中信息的句柄。

方法

※URL （String link） throws MalformedURLException

根据指定的包含一个完整的 URL 的字符串创建一个 URL 对象。

※ URL （URL context， String fileName） throws MalformedURLException

创建一个 URL 对象，与 Context 具有同样的主机名和协议，但要用指定的文件名 （FileName）。例如下面的代码：

```
URL u1=new
URL ( "http", "ww.xyz.com", 1500, "/foo.html//abc");
URL u=new URL （ul, "funny.html");
System.out.println(ul.toString （ ));
System.out.println(u.toString （ ));
```

输出结果：

http：//www.xyz.com:1500/foo.html//abc

http：//www.xyz.com:1500/funny.html

※URL（String protocol，String host，String file）throws MalformedURLException

创建一个将利用指定协议与指定主机交互作用的URL。file是URL依附的文件名，比如HTML文档或用于FTP接收的文件。

※URL(String protocol,String host,int port,String file) throws MalformedURLException

创建一个将利用指定协议与指定主机交互作用的URL。file是URL依附文件名，比如HTML文档或通过FTP接收的文件。

※int getPort（）

返回URL将由之获取信息的端口。

※String getProtocol（）

返回URL将由之获取信息的端口。

※boolean equals（Object anotherObject）

如果另一个对象（anotherObject）是一个以同样方式用同一文件表达的URL对象，则返回真值。这意味着anotherObject必定有相同的协议、主机、端口和文件。但是，任何URL结尾处的“#ref”字段被忽略，(这是一个由方法sameFile（）返回的值)。

※final Object getContent（）throws IOException

※String getFile（）

返回URL感兴趣的文件名。

※String getHost（）

返回URL将由之获取信息的主机。

※String getRef（）

返回URL的ref——标磅符（#）后的任何东西。

※int hashCode（）

返回URL对象的hashcode——一个与协议、主机和文件名

有关的散列码。

※URLConnection openConnection（）throws IOExcerption

打开一个连接，获取由 URL 指定的文件中包含的信息。它返回一个基于当前 URLStreamHandler 的 URLConnection。

※final inputStream openStream（）throws IO Exception

打开一个 URL 的 InputStream。这可以通过首先创建一个 URLConnection 对象然后用方法 URLConnection. getInputStream（）获取 InputStream 来做到。

※boolean sameFile（URL oterURL）

如果 otherURL 以同样方式参阅同一文件，则取得真值。这意味 otherURL 必定有相同的协议、主机、端口和文件。然而，任何 URL 结尾处的“#ref”字段将被忽略。

※protected void set（String protocol，String host，ing port，String file，String ref）

设置 URL 字段，此方法设为 protected 以便只允许 URLStreamHandler 改变这些字段。

※static synchronized void setURLStreamHandlerFactory（URLStreamHandlerFactory fac）

为 URL 设置 URLStreamHandlerFaetory，URLStreamHandlerFactory 将决定如何处理 URL 的文档内容。

※String toExternalForm（）

结合各种字段以标准格式创建一个 URL。

※String toString（）

结合各种字段以标准格式创建一个 URL。

Abstract URLConnection

URLConnection 类是一个抽象类，用于协助管理基于 URL 对象的连接。

字段

• protected boolean allowUserInteraction

一些 URL 需要用户输入——比如填写一个口令对话框。它

初始被设为False，意味着任何欲获取用户交互操作的请求将会导致异常。

• protected boolean connected

跟踪URLConnection的状态——连接或没有连接。

• protected boolean doInput

如果URLConnection被用来作为输入源则为真值。初始置为true。

• protected boolean doOutput

如果URLConnection被用来作为输出源（原书为输入源）则为真值。初始置为false。

• protected long ifModifiedSince

此字段通过不装载旧信息可使连接更为有效，初始值设为0，更多的要求可以在任何一个子类中实现。

• protected URL url

已建立连接的URL。

• protected boolean useCaches

如果useCaches为真，连接将在任何可能的情况下利用高速缓存信息。此字段初始值为true。

方法

※protected URLConnection（URL url）

创建一个连接到指定URL的URLConnection。

※abstract void connect（）throws IOException

创建与该对象指定URL的连接。

※boolean getAllowUserinteraction（）

返回allowUserInteraction标志。

※Object getContent（）throws IOException

返回URL参阅的对象。

※String getContentEncoding（）

返回URL题头中的“context encoding”字段。

※int getContentLength（）

从 URL 的题头返回“context length”字段。

※String getContentType（）

从 URL 题头表列中返回“context type”字段。

※long getDate（）

从 URL 题头表列中返回“date”字段。

※boolean getDoInput（）

返回此对象的 doInput 字段。

※boolean getDoOutput（）

返回此对象的 doOutput 字段。

※long getExpiration（）

从 URL 的题头返回“expires”字段。

※String getHeaderField（int n）

从题头列表中返回第 n 个字段的标题。若少于 n 个字段名，则返回空。当前返回一个空串。

※String getHeaderField（String name）

返回指定名字的标题值。

※long getHeaderFieldDate（String name，long Default）

返回指定名字为一个日期的标题。

※int getHeaderFieldInt（String name，int Default）

返回指定名字标题字段为一个整数的值。

※String getHeaderFieldKey（int n）

返回列表标题的第 n 个字段值。若少于 n 个标题则返回 NULL。

※inputStream getInputStream（）throws IOException

返回此 URLConnection 的 InputStream。

※long getLastModified（）

返回“last modified”字段值，若未定义则返回 NULL。

※OutputStream getOutputStream（）THROWS IOException

返回 URLConnection 的 OutputStre am。

※URL getURL（）

返回 URLConn ection 由之获取信息的 URL。

※viod setAllowUserinteraction（boolean allow）

将 allowUserInteraction 字段设为指定允许 allow 值。

※static void setDefaultAllowUserInteraction（boolean default）

为 allowUserInteraction 字段设置缺省值。

※void setDoinput（boolean doinput）

设置 doInput 字段值。

※void setDoOutput（boolean dooutput）

设置 doOutput 字段值。

※String toString（）

返回包含连接 URLConnection 的类名与 URL 名的字符串。

※static boolean getDefaultAllowUserinteraction（）

返回 allowUserInteraction 字段缺省值。

※static String getDefaultRequestProperty（String key）

返回 allowUserInteraction 字段缺省值。缺省情况下返回一个空串，因此必须给予替换值。

※boolean getDegaultUseCaches（）

返回 useCaches 字段的缺省值。

※long getifModefiedSince（）

返回 isModefiedSince 字段。

※String getRequestProperty（String key）

返回名字是 key 的一般请求属性。

※boolean getUserCaches（）

返回 useCaches 字段。

※ static synchronized void setContentHandlerFactory（ContentHandlerFactory fac）

说明这个对象的 ContentHandlerFactory。

※ static void setDefaultRequestProperty（String key，String value）

此方法用于为 URLConnection 的请求属性设置缺省值。除非

重载，此方法将不做任何事情。

※void setDefaultUseCaches（boolean defaultusecaches）

针对 useCaches 字段设置缺省值

※void setIfModifiedSince（long ifmodifiedsince）

设定 ifModifiedSince 字段值。

※viod setRequestProrerty（String key，String value）

此方法用于设置一个由 URLConnection 应用的请求属性。当前，此方法不做任何事情。

※void setUseCaches（boolean usecaches）

设置 useCaches 字段值。

※static String guessContentTypeFromName（String fileName）

返回一个包含 MIME 类型的字符串，它最可能包含在指定文件名的文件中。此方法已能实现识别总共 39 个指定内容类型的扩展名，包括 .exe、.zip、.gif 和 .html。

※static String guessContentTypeFromStream（InputStream is）throws IOException

此方法能够基于流的前六个字节推测上下文类型，能够识别 image/gif，image/jpg，image/x-bitmap 和 text/html 数据 MIME 类型。

URLEncoder

此类能够把标准字符串翻译成 x-www-form-urlencode 格式。这个过程涉及一些转换，比如将空格变为符号‘+’，将非字母数字的字符转换为十六进制数。当向 CGI 脚本传递信息时，将会看到这是用于标准 URL 的格式。

方法

※static String encode（String s）

把一个标准格式字符串翻译成 x-www-form-urlencode 格式。所有空格转换为符号‘+’，除数字字符和下划线‘_’以外的所有字符被翻译成为十六进制的数字表示。

※Abstract URLStreamHandler

URLStreamHandler 是一相当简单的类，它用于把 URL 转换为 URLConnection 方法。

※URLStreamHandler（）

创建一个 URLStreamHandler。

※abstract URLConnection openConnection（URL url）throws IOException

此方法创建一个从指定 URL 接收信息的 InputStream。它必须被重载以完成类的“非抽象化”。

※void parseURL（URL url，String spec，int start，int limit）

解析字母 start 和 limit 之间（且包括）的 spec 串，并且将它们的值存入 URL 对象。注意尽管没有新 URL 返回，指定 URL 中的字段也将由于此方法设置了 URL 的内部字段而发生变化。

※void setURL（URL url，String protocol，String host，int port，String file，String ref）

为指定 URL 对象设置字段，注意尽管没有取得新 URL，指定 URL 中的字段也将由于此方法设置了 URL 的内部字段而发生变化。

※String toExternalForm（URL url）

返回一个代表指定 URL 的字符串，这个对象正在从指定 URL 中检索信息。

2. 接口

ContentHandlerFactory

ContentHandlerFactory 能够用于创建一个管理器，此管理器能够为指定 MIME 类型取得适当的 ContentHandler。这样的类被 URLConnection 类用来正确处理接收的数据。

方法

※abstract ContentHandler creatContentHandler（String mimetype）

针对不同的 MIME 类型返回 ContentHandler 的属性。

SocketImplFactory

SocketImplFactory 能够让用户使用 SocketImpl 类。SocketImplFactory 类将根据创建插座的请求返回响应中的一个 SocketImpl。通过在 java.net.serverSocket 类中应用 SocketImplFactory，用户能够说明要用哪一个 SocketImplFactory。这样，某个 socketImpl 便得以应用。

方法

※abstract SocketImpl createSocketImpl（）

创建一个用于创建插座的新 SocketImpl 对象。

URLStreamHandlerFactory

通过应用方法 CreateURLStreamHandler（），URLStreamHandlerFactory 类为指定类型的数据流类型创建一个 URLStreamHandler。此类通常不是显现地用于程序中。

方法

※abstract ULRStreamHandler CreatURLStreamHandler (String protocol)

根据指定的协议创建一个 URLStreamHandler，用来处理 URL。

七、JAVA. UTIL

此软件包实际上包含了一组手边常用的类，这些类不适于归到其他任何的包。它们之中有 Date 类，用来管理和处理日期操作；还有 Hashtable 类和 ADT，比如 Stack 与 Vector。

1. 类

BitSet Implements java. lang. Cloneable

Bitset 是一个位的动态集合，它具有附加的基本逻辑运算的能力。尽管可以在 BitSet 上执行一些逻辑操作，但此类是修改当前的 BitSet 而不返回一个新的 BitSet。

演示程序 21 并将其展示在图 25，其逻辑操作将导致原 BitSet 的改变。

程序 21　BitSet 示例

```
import java. util. BitSet;

public class BitsetExample {
    public static void main (String argv []) {

        BitSet a=new BitSet ( );
        BitSet b=new BitSet ( );

        a. set(0);
        a. set(1);
        a. set(2);
        b. set(0);
        b. set(1);
```

```
        b.set(8);
        b.set(9);

        System.out.println("This is the first set:"+a);
        System.out.println("This is the second set:"+b);
        a.and(b);//places all elements in both sets into a
         System.out.print ("The following elements are in
both");
        System.out.println("sets:"+a);
    }
}
```

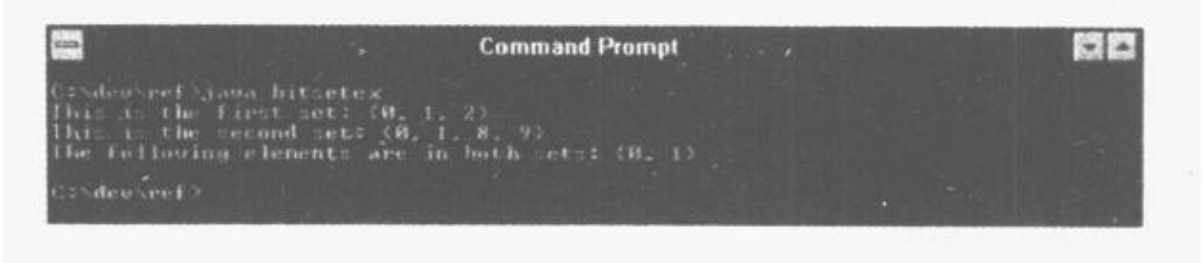


图 25　程序 21 的输出

方法

※BitSet（）

创建一个大小为 64 的新 BitSet。

※BitSet（int nbits）

用指定大小创建一个新 BitSet。

※void and（BitSet set）

把两个集合的所有元素放入原始集合——两者的逻辑交集。

※void clear（int bit）

从集合中删除指定位的所有实例。

※Object clone（）

产生当前集合的一个拷贝。

※boolean equals（Object anotherObject）

如果 antherObject 是一个具有同样元素的 BitSet 则返回

true。

※boolean get (int bit)

如果在 BitSet 中发现指定位则返回 true。

※int hashCode ( )

返回集合的 hashcode ——一个基于集合元素的值。

※void or (BitSet set)

把此 BitSet 或指定集合的所有元素放入这个集合——逻辑并。

※void set (int bit)

在集合中放置指定位。

※int size ( )

返回集合的大小。

※String toString ( )

返回一个包含集合的字符串,此集合用标准集合表示法表示,例如,{1, 2, 3}。

※void xor (BitSet set)

把一个(不是二个)集合的所有元素置入当前集合。它包括并集而不是交集的所有元素。

Date

Data 类提供管理日期和时间的便利方法。此类用法相当灵活,它可以用许多方法来设置和操作日期,并且能够处理时区和夏令时间转换。

方法

※Date ( )

创建一个基于当前时间的日期。

※Data (long date)

创建一个基于指定日期的时间,代表从 1970 年 1 月 1 日 00:00:00 时起逝去的百万秒数。尽管这种指定时间的方式看起来相当抽象,但这种压缩格式极为有效。要获得从 1970 年 1 月 1 日 00:00:00 时起逝去的百万秒数须用函数 System.currentTime-

Millis（）。

※Date（int year，int month，int date）

以指定年，月，日创建一个日期。年数必须晚于1900年。

※Date（int year，int month，int date，int hours，int min）

以指定年，月，日，时，分值创建一个日期。年数必须晚于1900年。

※Date（String dateString）

用日期字符串（dateString）中的指定值创建一个日期。这依赖于parse（），它以“Tue，26 Mar 1996 04：05：00 GMT＋1900”的格式接收日期。

※Date（int year，int month，int date，int date，int hrs，int min，int sec）

用指定值创建一个日期。

※boolean after（Date when）

如果日期在when之后是按年编的，则返回true。

※boolean before（Date when）

如果日期在when之前是按年编的，则返回true。

※boolean equals（Object anotherObject）

如果antherObject具有同样的日期值（自1/1/70的百万秒数）则返回true。

※int getDate（）

返回日期：一个1和31之间的整数。

※int getDay（）

返回日期为星期几：一个0到6之间的整数值。0代表周日。

※int getHours（）

返回日期的小时数：一个0到23之间的整数。

※int getMinutes（）

返回日期的分钟数：一个0到59之间的整数。

※int getMonth（）

返回日期的月数：一个1到12之间的整数。

※int getSeconds（）

返回日期的秒数：一个 0 到 59 之间的整数。

※long getTime（）

返回 Date 对象所指定的日期，表示为自 1970 第 0 时 0 分 0 秒之后的百万秒数。

※int getTimezoneOffset（）

返回此时区与格林威治平均时间相差的分钟数。

※int getYear（）

返回自 1900 年起的年数（这样 100 将代表 2000 年）。

※int hashCode（）

返回日期的散列码：基于自 1/1/70 00：00：00GMT 逝去秒数的一个值。

※statec long parse（String s）

解析一个代表日期的字符串，返回与 1/1/70 00：00：00GMT 相差的秒数。

※void setDate（int date）

设置 Date 对象的日期（月日）。

※viod setHours（int hours）

设置 Date 对象的小时值。

※void setMinutes（int minutes）

设置 Date 对象的分钟值。

※void setMonth（int month）

设置 Date 对象的月数值。

※void setSeconds（int seconds）

设置 Date 对象的秒数值。

※void setTime（long time）

通过指定相对于 1/1/70 00：00：00GMT 的百万秒数，设置 Date 对象的值。

※void setYear（int year）

设置 Date 的年数值。

※String toGMTString（）

返回一个包含GMT格式日期的字符串，它已对指定时区进行调整（例：25 Mar5 1996 09：05：00 GMT）。

※String toLocaleString（）

返回一个包含当地日期格式的字符串，不做时区调整（例：03/25/96 04:05:00）。

※String toString（）

返回一个以系统所依赖格式表述时间的字符串。

※statec long UTC（int year，int month，int date，int hrs，int min，int sec）

返回在1/1/70 00：00：GMT和指定时间之间的百万秒数。

Abstract Dictionary

此类定义散列表的行为。然而在散列表中所有方法服务于一个目的，在此类中，它们都是抽象的。

方法

※Dictionary（）

创建一个新字典。

※abstract Enumeration elements（）

返回一个可以顺序查阅的元素的Enumeration。

※abstract Object get（Object key）

返回具有指定关键词的元素。

※abstracy boolean isEmpty（）

如果字典为空则返回真。

※abstract Enumeration keys（）

返回一个可以顺序查阅的关键词细目。

※abstract Object put（Object key，Object element）

把指定元素放在字典中指定关键词的下面。

※abstract Object remove（Object key）

去掉关键词以及与它相应的元素。

※abstract int size（）

返回字典中的元素数。

Hashtable Extends Dictionary Implements java. lang. Cloneable

Hashtable 类是 hashtable ADT 的一个简单实现。散列表以简便有效的方法存储基于不同关系的信息。和数组每个元素与特定下标相配类似，散列表中每个元素与特定的关键词相配。通过使用此关键词可以检索表中信息。

使用 Hashtable 类时，需要注意每一个关键词和元素必须是一个对象。正如图 26 中所看到，处理字符串不成问题。

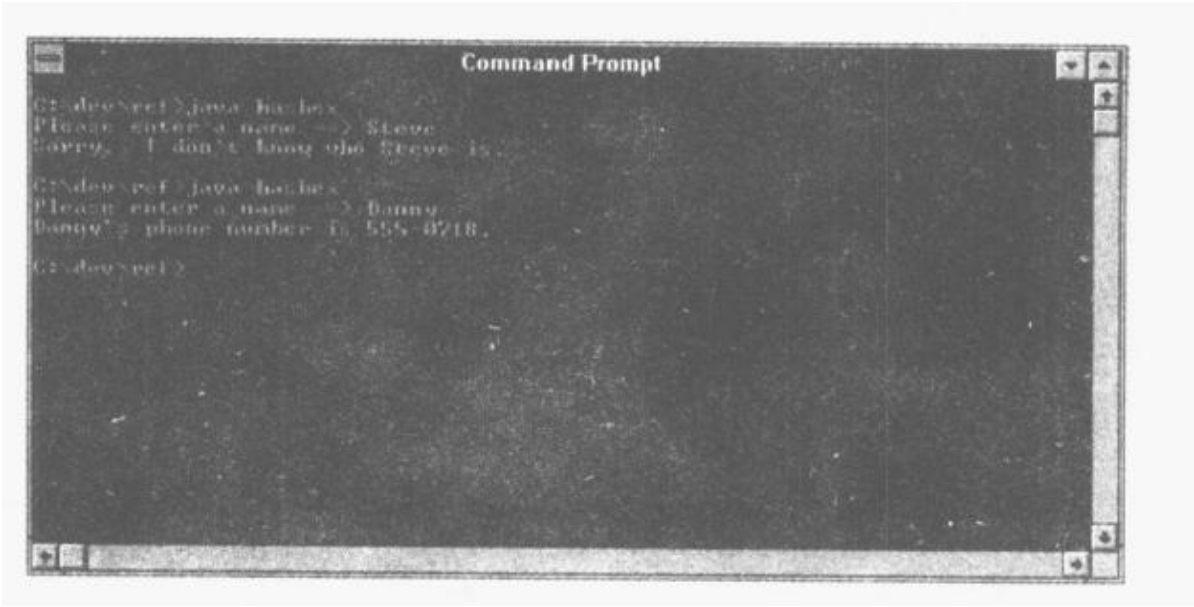


图 26　程序 26 的运行结果

程序 22　使用 String Objects 的 Hashtable 示例

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

public class HashtableExample {

    public statec void main (String argv []) {
        Hashtable phones=new Hashtable ( );
        String name=null;
        String phone=null;
```

```
        phones.put ( "Danny", "555-0718");
        phones.put ( "Jason", "555-0031");
        phones.put ( "jaime", "555-2191");
        phones.put ( "Jeff", "555-1391");
        System.out.print ( "Please enter a name==>");
        System.out.flush ( ); //forces the prompt to be dis-
played
        DataInputStream din = new DataInputStream (Sys-
tem.in); try {
        name=din.readLine ( );
    }
    catch (Exception e) {
        System.out.println (e.toString ( ));
    }
    if (phones.containsKey (name)) {/* checks to see if the
name is valid */
        phone= (String) phones.get (name);
        System.out.println (name+ " 's phone number is " +
phone+ ".");
    }
    else {
        System.out.print { "Sorry. I don't know who ");
        System.out.print {name+ "is.");
    }
}
```

当处理某些基础数据类型如整数和浮点数时，有必要建立一种封装器（wrapper）Object。这种封装器初始时用来存储一些数值，在检索信息时又会再次用它。请参看程序 23，其输出展示在图 27。

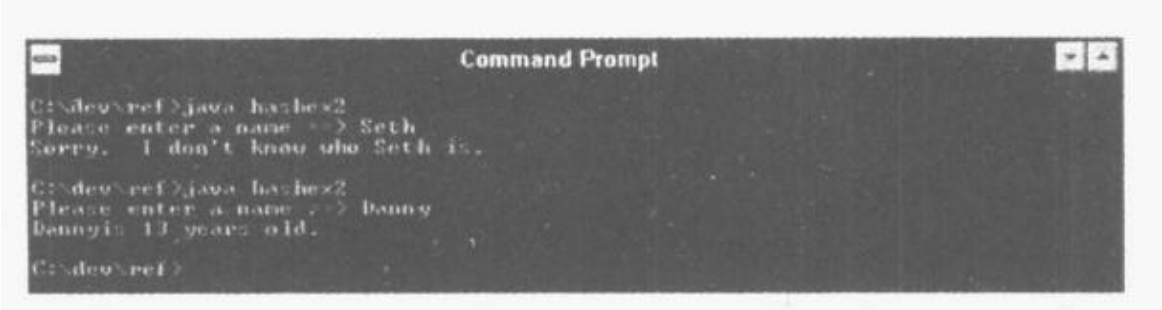


图 27　程序 23 的运行结果

程序 23　Hashtable Example Using Integer Objects

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

public class HashableExample2 {

    public static void main (String argv []) {
        Hashtable ages=new Hashtable ();
        String name=null;
        Integer age=null;

        ages.put (" Danny", ew Integer (13));
        ages.put (" Jason", new Integer (7));
        ages.put (" Jaime", new Integer (39));
        ages.put (" Jeff", new Integer (54));

        System.out.print (" Please enter a name==>");
        System.out.flush ();
        DataInputStream din = new DataInputStream (System.in);

        try {
            name=din.readLine ();
```

```
        }
        catch (Exception e) {
            System.out.println (e.toString ( ));
        }
        if (ages.containsKey (name)) {
            age= (Integer) ages.get (name)) {
            System.out.print (name+ "is" +age.intvalue
( ));
            System.out.println (" years old.");
        }
        else {
            System.out.print ( "Sorry. I don't know who");
            System.out.println (name+ "is.");
            }
        }
    }
```

方法

※Hashtable ( )

创建一个散列表。容量设为 101，装填因子 loadFactor 设为 0.75。

※Hashtable (int initialCapacity)

根据指定的容量和 0.75 的 loadFactor，创建一个新的 Hashtable。

※Hashtable (int initialCapacity, float loadFactor)

创建一个新的散列表，initialCapacity 是能包括的条目数，loadFactor 是一个 0.0 到 1.0 之间的数，它代表散列表的百分比，是 hashtable 通过 rehash () 方法重新设定大小前可以填入条目数所占的百分比。

※synchronized void clear ( )

清除散列表，删除所有关键词和元素。

※synchronized Object clone ( )

创建一个散列表的拷贝。然而，新表的内容将是旧对象的参照，并不是一个新对象。

※synchronized boolean contains (Object value)

如果参数 value 是散列表中的一个元素则返回真值。

※synchronized boolean containsKey (Object key)

如果参数 key 是散列表中的关键词则返回真值。

※synchronized Enumeration elements ( )

返回可以顺序检索的元素计数。

※synchronized Object get (Object key)

用指定关键词取得元素。

※boolean isEmpty ( )

若散列表为空则返回真值。

※synchronized Enumeration keys ( )

返回可以顺序查阅的关键词计数。

※synchronized Object put (Object key, Object element)

根据特定关键词，将指定元素放入 hashtable。

※protected void rehash ( )

重新设定散列表的大小，当散列表的大小超过在构造方法中指定的界限值，此过程自动触发。新散列表尺寸大于当前表的两倍。

※synchronized Object remove (Object key)

去掉关键词和相应的元素。若正确删除则返回关键词 key，若没有发现则取得 0。

※int size ( )

返回散列表的元素数。

※synchronized String toString ( )

返回包括所有关键词-元素对的字符串。

Observable

Observable 类可以创建一个对象，这个对象包含当该对象和

其他对象紧密连接时被操作的数据。要应用此类，必须将其扩充，来创建用户自己的“observable”类。此类可以通过方法 notifyObservers（）与许多Observer 类通信。因此，Observable 类的任何变化都能在所有 Observer 中触发相应的反应。

▶参见 “Observer”

方法

※Observable（）

创建一个新 Observable 对象。

※synchronizzed void addObserver（Observer o）

在对象的 observer 列表中增加一个 observer。

※protected synchronized void clearChanged（）

把 Changed 标志置为假。

※synchronized int countObservers（）

返回 observer 的数目。

※synchronized void deleteObserver（Observer o）

从 observer 列表中删除指定的 Observer。

※synchronized void deleteObservers（）

从 observer 列表中删除所有的 Observer。

※synchronized boolean hasChanged（）

返回 Changed 标志的值。

※synchronized void notifyObservers（Object information）

将指定的 infomation 对象发送到所有的 observers。

※void notifyObservers（）

向所有的 observers 发送一个 null 对象。

※protected synchronized void setChanged（）

将 Changed 标志置为 true。

Properties Extends Hashable

Properties 类是一个可存储和（或）装载的散列表。它通常用来处理系统属性，因为它通常需要检索这些属性，经常能够方便地将它们存储起来。

字段

• lprotected Properties defaults

当在新 Properties 集中没有找到某个属性时，缺省的属性集将被应用。

方法

※Properties ( )

创建一个没有设缺省值的 Properties 对象。

※Properties (Properties defaults)

创建一个指定缺省集的 Properties 对象。

※String getProperty (String key, String defaultValue)

返回指定关键词的属性元素。若没有找到，此方法将检查缺省属性。若仍不能找到指定关键词，它将返回缺省值 defaultValue。

※String getProperty (String key)

返回指定关键词的属性元素。若没有找到，此方法将检查缺省属性。

※void list (PrintStream out)

将属性输出到指定 PrintSteam。

※synchronized void load (InputStream in) throws IOException

从指定 InputStream 检索一个属性集。

※Enumeration propertyNames ( )

返回属性表中所有关键词的计数。

※synchronized void save (OutputStream out, String header)

运用 header 参数将属性表保存到指定的 Outputstream。

Random

Random 类的作用是封装创建随机数所必须的行为。

方法

※Random ( )

用当前的百万秒时间作为种子 (seed) 创建一个 Random 对象。

※Random（long seed）

用指定种子 seed 创建一个 Random 对象。

※double nextDouble（ ）

返回一个 0.0 到 1.0 之间的随机数。

※float nextFloat（ ）

返回一个 0.0 到 1.0 之间的随机数。

※synchronized double nextGaussian（ ）

返回一随机产生的 double，它的中心在 0 附近，而且具有标准偏移为 1.0。这意味着会得到大于 1.0 的某些数。

※int nextint（ ）

返回一个随机整数值。

※long nextLong（ ）

返回一个随机产生的 long 型值。

※synchronized void setSeed（long seed）

设置用于产生随机数的种子。

Stack Extends Vector

Stack 类实现了一个标准的 ADT 堆栈。堆栈遵循后进先出(LIFO)算法，新数据置于堆栈的顶端——因此它是首先要被删除的数据项。堆栈中的每一项须是 fullblown 对象。

方法

※Stack（ ）

创建一个空堆栈 Stack。

※boolean empty（ ）

如果堆栈为空返回 true。

※Object peek（ ）

返回堆栈的顶端项，并不删除它。

※Object pop（ ）

返回堆栈的顶端项，并从栈中删除它。

※Object push（Object item）

把指定项置于堆栈顶部。

※int search (Object item)

返回 item 项相对于堆栈顶端的位置。如果堆栈中没有此项则返回－1。

StringTokenizer Implements Enumeration

StringTokenizer 类能够解析由若干子串组成的字符串。

对于 StringTokenizer 的应用，有两个内部字段可以替换：delimiters 串和 retTokens 值。delimiters 串包括所有重要标志。当做串解析时，如果 delimiters 中有当前字符，StringTokenizer 将返回一个字符串。retToken 是一个标志，此标志能够指明是否返回作为独立标志的定界字符。

方法

※StringTokenizer (String str)

创建一个 StringTokenizer。若缺省，“\t\n\r”将作为定界符串（用‘ ’，‘\t’，‘\n’和‘\r’表征定界符），并且 retTokens 标志将被置为 false。

※StringTokenizer (String str, String delim)

根据指定的定界字符集创建一个新的 StringTokenizer。

※StringTokenizer (String str, String delim, boolean returnTokens)

根据指定的定界限字符集创建一个 StringTokenizer。如果 returnTokens 为真，所有标志将像独立字符串一样被返回。

※int countTokens ( )

返回串中标志数目。

※boolean hasMoreElements ( )

若元素保留在串中则返回 true。缺省情况下，它只返回 hasMoreTokens ( ) 的值。此方法对于满足实现 Enumeration 接口的需求是必要的。

※boolean hasMoreTokens ( )

若有更多的标志存在于串中则返回 true。

※Object nextElement ( )

返回由 nextToken（）返回的串。此方法对于满足实现 Enumeration 接口的要求是必要的。

※String nextToken（String delim）

使 delim 成为新的定界符集，然后返回下一个标志。

※String nextToken（）

返回串中下一个标志。

Vector Implements java. lang. Cloneable

Vector 是一个动态数组，当处理未知数量并且经常改动的数据时最为有用。Vector 尽管基于数组结构，但当需要时，它可以自动增长提供新内存。而且，这种能力能够使用户执行更多的处理工作，比如从 Vector 删除一个元素或将一个元素插入到 Vector 中。当应用 Vector 时，须注意：像一个散列表一样，所有的元素都必须是一个对象。因此，如果想存储简单数据，比如整型或浮点型，则必须应用 java. lang 软件包中的 wrapper 类。

字段

• protected int capacityIncrement

由于扩展需要而引起 Vector 数组每次增长的总数。如果 capacityIncrement 是 0，那么数组大小增加一倍。

• protected int elementCount

Vector 中的元素数。

• protected Object elementData []

存储元素的数组。

方法

※Vector（）

创建一个初始容量为 10 的 Vector 对象。当 Vector 需要增长时，它的大小增加一倍。

※Vector（int initialCapacity）

根据指定的初始容量 initialCapacity 创建一个新 Vector 对象。当 Vector 需要增长时，它的大小增加一倍。

※Vector（int initialCapacity，int capacityincrement）

用指定初始容量创建一个 Vector 对象。当 Vector 需要增长时，它应按指定大小增长。

※final synchronized void addElement (Object element)

向 Vector 数组尾部添加指定元素 element，相应按需要增大数组长度。

※final int capacity ( )

返回 Vector 的总容量：elementData [] 数组的长度。

※synchronized Object clone ( )

建立 Vector 对象本身的一个拷贝，不包括它的元素。这意味着新 Vector 对象和旧 Vector 对象中的元素将被相同对象参阅。

※final boolean contains (Object element)

若 Vector 包含指定元素则返回 true。

※final synchronized void copyInto (Object anArray [])

把 Vector 中的元素拷贝到一个静态（标准）数组。

※final synchronized Object elementAt (int index)

返回指定索引的元素。

※final synchronized Enumeration elements ( )

返回 Vector 中元素计数。

※final synchronezed void ensureCapacity (int elements)

检查 Vector 是否能至少包含指定元素数。

※final synchronezed Object firstElement ( )

返回 Vector 数组的第一个元素。

※final int indexOf (Object element)

返回指定元素的第一个索引。若没有找到该元素则返回－1。

※final synchronized int indexOf (Object elem, int start)

返回指定元素的第一个索引。它在 Start 指定的起始处开始查找，若没有找到则返回－1。

※final synchronized void insertElementAt (Object object, int index)

在指定索引处加入指定对象。

※final boolean isEmpty（）

若 Vector 数组为空则返回真值。

※final synchronized Object lastElement（）

返回数组中最后一个索引处的最后一个元素。

※final int lastindexOf（Object elem）

返回指定元素的最后一个索引。若没有找到则返回－1。

※final synchronized int lastindexOf（Object elem，int start）

返回指定元素的第一个索引。它从 Start 指定的起始位置向后查找，若没有找到则返回－1。

※final synchronized void removeAllElements（）

删除 Vector 中所有元素。

※final synchronized boolean removeElement（Object element）

从 Vector 数组删除指定元素的第一个实例。

※final synchronized void removeElementAt（int index）

从 Vector 数组删除指定索引的元素。

※final synchronized void setElementAt（Object obj，int index）

设置指定索引的元素，覆盖原来存储在索引中的任何值。

※final synchronized void setSize（int newSize）

设置 Vector 数组的大小。如果 newSize 比当前尺寸小，数组将缩小到 newSize。如果 newSize 比当前尺寸大，数组将扩大到 newSize。

※final int size（）

返回 Vector 数组的大小。

※final synchronized String toString（）

返回一个包含在 Vactor 数组中所有元素的字符串。

※final synchronezed void trimToSize（）

将数组缩小到能够包含数组中所有元素的最小长度。

2. 接口

Enumeration

Enumeration 是一种处理条目集合的简单方法，运用此方法可使用户不必急于知道包含在条目集中的条目数。当处理许多组数据时，这种方法是必要的。Enumeration 接口并不提供诸如 Hashtable 和 AppletContext 中的 applet 那样的按顺序处理每一个元素的简单方法（参见图 28）。运用 Enumeration，能够以简单方式穿过条目进行滚动，如在 properties 集合的关键词中滚动，又如在 AppletContext 的程序中滚动或者在 StringTokenizer 的子串中间滚动。

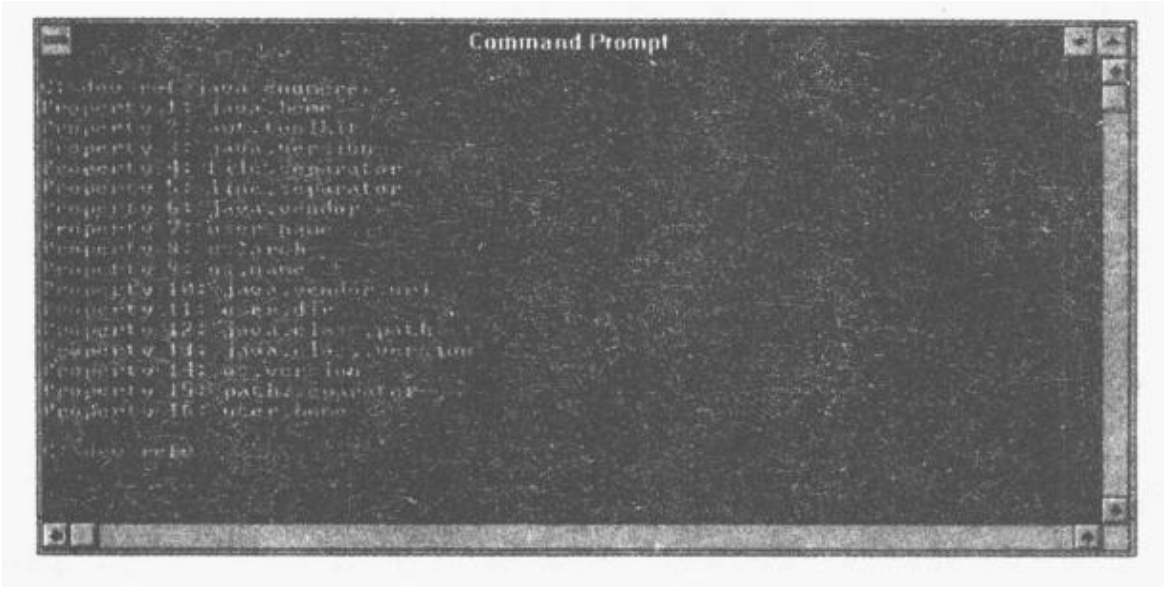


图 28　为程序 24 的输出屏幕

程序 24　Enumeration 示例

```
import java.util.Enumeration;
import java.util.Properties;

public class EnumerationExample {

    public static void main (String args []) {

        Enumeration propNames=System.getProperties ( ).
propertyNames ( );
```

```
        int index=1;
        while (propNames.hasMoreElements ())
            System.out.print ("Property" + (index++) +
":");
            System.out.println(propNames.nextElement());
    }
}
```

方法

※abstract boolean hasMoreElements ()

如果 Enumeration 包含更多的元素则返回真。

※abstracy Object nextElemet ()

返回 Enumeration 中的下一个元素。

Observer

Observer 接口用于和 Observable 类连接。通过向 Observable 类的 observer 列表加入一个 Observer，Observable 类将被通知 Observable 中的重要变化(无论何时调用 Observable 类中的 notifyObservers () 方法，都将发出这种通信)。

方法

※abstract void update (Observable o, Object arg)

▶参见 "Observable"

八、EXCEPTIONS

EXCEPTIONS 提供处理一般运行问题，这些问题可能会在程序运行期间遇到。所有异常源自 java.lang.Exception 类并且绝大多数仅包含两个构造方法。因此，仅把 java.lang.Exception 的构造方法列在这里。除非另加注释，所有异常处理可以在没有参数或带有作为参数的描述性字符串的情况下创建。异常用 try...catch 结构处理，一旦遇到一个异常，它会被直接处理或被抛向调用方法。程序 25 是一个简单除法方法的例子，如果被 0 除则将结果设置为 0。

程序 25　Catching an Exception

```
int divide (int a, int b) {
    int val;
    try {
        val=a/b;
    }
    catch (ArithmeticException e) {
        System.out.println ("Invalid data.val set to 0.");
        val=0;
    }
    return (val);
}
```

为了抛出 (throw) 一个异常，必须应用 throw 语法，如程序 26 和程序 27 中所示。

程序 26　Throwing a Caught Exception

```
void connect (String host, int port) throws IOException {
    try {
```

```
        s=new Socket (host, port);
    }
    catch (IOException e) {
        throw e;
    }
}
```

程序 27　throwing aNew Eceptiǒn

```
static void connect (String host, int port) throws IOException {
/ * Make sure the port satisfies our system policy * /
if (port<1600)
    throw new SocketException ( "Port under 1600");

try {
    s=new Socket (host, port);
}
catch (IOException e) {
    throw e;
    }
}
```

java. awt

AWTException

AWTException（字符串消息）。

java. io

EOFException Extends IOException

FileNotFoundException Extends IOException

IOException Extends java. lang. Exception

interruptedIOException Extends IOException

UTFDataFormatException Extends IOException

java. lang

ArithmeticException Extends RuntimeException
ArrayindexOutOfBoundsException Extends
indexOutOfBoundsException
constructors
    ArrayindexOutOfBoundsException ( )
    ArrayindexOutOfBoundsException (Int invalid _index)
    ArrayindexOutOfBuondsException (String message)
ArrayStoreException Extends RuntimeException
ClassCastException Extends RuntimeException
ClassNotFoundException Extends Exception
CloneNotSupportedException Extends Exception
Exception Extends Throwable
Methods
    Exception (String message)
    Exception ( )
IllegalAccessException Extends Exception
IllegalArgumentException Extends RuntimeException
IllegalMonitorStateException Extends RuntimeException
Illegalthread StateException Extends RuntimeException
IndexOutOfBoundsException Extends RuntimeException
InstantiationException Extends Exception
InterruptedException Extends ExCeption RuntimeException
NegativeArraySizeException Extends RutimeException
NoSuchMethodException Extends Exception
NullPointerException Extends RuntimeException
NumberFormatException Extends
IllegalArgumentException
RuntimeException Extends Exception
SecurityException Extends Exception
StringIndexOutOfBoundsException Extends

IndexOutOfBoundsException

Constructors

StringIndexOutOfBoundsException ( )

StringIndexOutOfBoundsException (int invalid _ index)

StringIndexOutOfBoundsException (String)

java. net

MalformedURLException Extends

java. lang. IOException

ProtocolException Extends java. lang. IOException

SocketException Extends java. lang. IOException

UnknownHostException Extends

java. lang. IOException

UnknownServiceException Extends

java. lang. IOException

java. util

EmptyStackException Extends

java. lang. RuntimeException

Constructios

EmptyStackException ( )

NoSuchElementException EXtends java. lang. RuntimeException

九、ERRORS

ERRORS

尽管错误处理和异常处理都基于 java. lang. Throwable 类，但错误处理可以处理更为严重的运行错误。处理错误的方式与异常相似。然而，除非清楚地理解问题并且已经设计了适当的解决办法，否则错误处理在某种程度上并不适于用在应用程序中。

与异常类似，所有的错误都源自 java lang Error，所有的错误仅会有构造方法并且大多数仅有两个构造方法，ErrorName ( )和 ErrorName (S tring message)，虽然模型一致的才被详细列出。

java. awt

AWTError Extends java.lang. Error

Constructors

AWTError( )

java. lang

AbstractMethodError Extends incompatibleClassChangeError

ClassCircularityError Extends LinkageError

ClassFormatError Extends LinkageError

Error Extends Throwable

Methods

Error (String message)

Error ( )

IlegalAcce SsError Extends IncompatibleClassChangeError

IncompatibleClassChangeError Extends LinkageError

InstantiationError Extends incompatibleClassChangeError

InternalError Extends VirtualMachineError

LinkageError Extends Error

NoClassDefFoundError Extends LinkageError

NoSuchFieldError Extends IncompatibleClassChangeError

NoSuchMethodError Extends IncompatibleClassChangeE-
rror

OutOfMemoryError Extends VirtualMachineError

StackOverflowError Extends Virtua. MachineError

ThreadDeath Extends Error

Constructors

ThreadDeath ( )

UnknownError Extends VirtualMachineError

UnsatisfiedLinkError Extends LinkageError

VerifyError Extends LinkageError

VirtualMachineError Extends Error

第二部分　语法速查

这部分是Java语言本身的一个参考。语言中所有的关键词和操作符以字母顺序列出，每个术语后面是它的完整解释、语法、和一个用于实际代码中的例子。为了更易于看清，术语在代码范列中以黑体字出现。

Abstract

抽象类或方法是不完整的，Interface 本身就是抽象类。

语法：

```
abstract class className {
    abstract returnType
methodName (optionalParameters);
}
```

举例：

```
abstract class Grapher
abstract void displayImage (Image im);
```

Break

它被用来退出一个循环。

语法：

```
break;
```

举例：

```
while (true) {
    if (connection. isClosed ( ))
        break;
    else
    //此处写代码
}
```

Catch

catch 语句用于处理 try 代码块发出的异常。

语法：

```
try {
    statement (s)
}
catch (Exception list) {
    statement (s)
}
```

举例：

```
InputStream in;
int val;
...
try {
    val=in.read ( ) /in.read ( );
}
catch (ArithmeticException e) {
    System.out.println ( "Invalid data. val set to 0.");
    val=0;
    }
catch (Exception e) {
    System.out.rintln ( " Exception encointered, but not
handle.");
    }
```

Class

用于类定义，表示后面代码定义一个类。

语法：

修饰语 **class** 类名 extends 被继承类名 implements 接口名

举例：

```
classMyClass
public class GraphAnimator implements Runnale
public class Calculator implents Runnable Cloneable
```

Continue

返回程序循环的顶部。

语法：

```
continue;
```

举例：

```
Enumeration enum;
Object value;
```

```
...
while (enum.hasMoreElements ( )) {
    value=enum.nextElement ( );
    if (value.equals ( "Invalid"))
        continue;
    else
        System.out.println (value);
}
```

Do...While

当一个条件满足时，用来执行一个操作。循环体至少被执行一次。

语法：

```
do
    statement (s)
while (布尔变量);
do
    statement (s)
while (布尔表达式);
```

举例：

```
do {
    val=in.readByte ( );
    system.out.println (val);
    } while (val! = '\n');
boolean valid=true;
do {
    val=in.readByte ( );
    if (val== '\n')
        valid=false;
    else
        System.out.println (val);
```

```
    } while (valid);
```

Else

else 用来与 if 语句连接，当 if 语句的陈述为假时执行相应操作。

语法：

```
if (布尔变量)
    statement (s)
else
    statement (s)
if (布尔表达式)
    statement (s)
else
    statement (s)
```

举例：

```
if (stillRunning) {
    System.out.println ("Still Running");
    advanceCounter ();
}
else {
    System.out.println ("We're all done.");
    closeConnection ();
}
if (size>=5)
    System.out.println ("Too big");
else
    System.out.println ("Just Right");
```

Extends

它可以使当前类或接口成为另一个类或接口的子类。

语法：

修饰语 class 类名 **extends** 超类名

interface interfaceName **extends** superInterfaceName

举例：

public class Clock **extends** Applet

public interface carefulobserver **extends** observer

Final

final 修饰语使一个类或方法成为终结，意味着它不能在子类中被改变。Interface 不能被终结。

语法：

final class 类名

final 返回类型 方法名（可选参数）

举例：

final class LogoSnimator

final Color getCurrentColor （）

Finally

finally 语句用于错误处理，保证一部分代码段的执行。不管 try 语句是否发出一个异常，finally 块中代码都将被执行。

语法：

```
try {
    statement (s)
    }
finally {
    清除语句
    }
try {
    statement (s)
    }
catch (Exception) {
    异常处理语句
    }
finally {
    清除语句
```

```
    }
```

举例：

```
public static void testMath (int numerator, int divisor) throws
ArithmeticException {
try {
    if (divisor==0)
        throw new ArirhmeticException ("Division by
Zero.");
}
finally {
    System.out.println ("The fraction was " +
numerator+ "/" +divisor);
    }
}
try {
    percent_over =quantity/number_ordered *
100;              //could cause dividion by 0
    }
catch (ArithmeticException e) {
    percent_over=0;
    }
finally {//regardless of rhe success of try,
we still need to ptint the info
    System.out.println ("Quantity=" +quantity);
    System.out.println ("ordered=" +ordered);
    System.out.println ("Percent Over=" + percent_over);
    }
```

For

用于按指定次数执行代码段。

语法：

for（计数初始化；计数检验；计数修改）

语句

举例：

```
String name;
...
for (pos=0; pos<name.length ( ); I++)
System.out.println(name.charAt (i));
```

If

执行满足某种特定条件的操作。

语法：

if（布尔变量）

statement（s）

if（布尔表达式）

statement（s）

举例：

```
if (ValidNumbersOnly)
    checkInput (Answer);
if (area>=2*PI) {
    System.out.println ("The size of the loop is still too big.")
    reduceSize (area);
}
```

Implements

用于强制一个类执行一个在接口中定义的方法。

语法：

修饰语 class 类名 **implements** 接口名

举例：

public class Clock **implements** Runnable，Cloneable

Import

用于包含其他库。

语法：

import 包名；

import 类名；

import 接口名；

举例：

import java. io. * ;

import java. applet. Applet;

import java. applet. AppletContext;

Instanceof

如果表达式左边的对象是右边类的一个实例，则操作符 instanceof 返回 true。

语法：

对象 **instanceof** 类名

举例：

```
void testType (object instance) {
    if (instance instanceof Sting) {
        System. out. println ( "This is a string. ")
        System. out. println ( "It is" + ( (String) i) . length
()); //casts the Object to a String first
```

Modifiers

访问修饰语用于控制类、接口、方法和字段的访问能力与行为。

修饰语	影响类	影响方法	影响字段
none (friendly)	对同一软件包中的类和子类可视。	能够被属于同一软件包类的方法调用。	仅能被同一软件包中的类仿问。

public	对子类和其它类可视(不管它们属于哪一个软件包)。	能够被子类和所有类中的方法调用（不管它们属于哪一个软件包)。	仅能被子类和所有类仿问(不管它们属于哪一个软件包)。
pvivate	类不能为private。	仅能被当前类中的方法调用。	仅能被当前类中的方法记问。
static	对于类不适用。	方法被当前类中所有实例共享。	字段被当前类中所有实例共享。
abstract	一些方法没有被定义,这些方法必须在子类中实现。	不包含代码，必须在子类中被替换。	对于字段不适用。
final	此类不能用作一个超类。	方法不能在任何一个子类中被替换。	变量的值不能被修改。
native	对于类不适用。	方法的实现将被用另一种语言编写的代码所定义。	对于字段不适用。
synchro-nized	对于类不适用。	此方法运行时将掌握类的控制。如果另一个方法已经掌握控制。它将等待，直到第一个方法完成。	对于字段不适用。

Native

native 方法将被用另一种语言写的代码定义。

语法：

native 返回类型 methodName（可选参数），

举例：

native long sumSeries（）；

New

操作符 new 为一个对象分配内存，比如 String，Socket，array 或任何其他类的实例。

语法：

数据类型 数组名［］＝**new** 数据类型［数字］；

数据类型 字段名＝**new** 数据类型（构造参数），

举例：

int sizes［］＝**new** int［9］；

String name＝new String（“Hello”）；

Package

用于把当前类放入指定软件包。

语法：

package 包名；

举例：

package java. lang；

package mytools；

Public

Public 使类、方法或字段能够被所有的类访问。

语法：

public class 类名；

public interface 接口名；

public 返回类型方法名（可选参数）；

public 数据类型字段名；

举例：

public class GraphicsExample;

public interface Graph;

public boolean checkStatus (int x, int y)

public int size;

Private

修饰语 private 使方法或字段仅能被当前类的方法访问。

语法:

private 返回类型 方法名 (可选参数),

private 数据类型 字段名;

举例:

private int changeStatus (int index);

private int count;

Return

return 语句用于从一个方法返回一个值。返回的数据类型必须和方法定义中指定的数据类型一致。

语法:

return 值;

举例:

```
float calculateArea (float circumference) {
    float radius, area;
    radius=circumference/ (2 * PI);
    area=radius * radius * PI;
    return (area);
}
```

Super

用于参阅此类的超类。

语法:

super

super. 方法名 ()

super. 字段名

举例：

```
class FloorManager extends Manager {
    FloorManager () {
        type=floor;
        super ( ); //calls the Manager
constructor
        }
    void organize () {
        size=name. getSize ();
        super. organize (size); //calls the organize method in
the Manager method
            ......}
    }
```

This

This 用于参考当前类。

语法：

this

this. 方法名（）

this. 字段名

举例：

```
ticker=new Thread (this);
```

Throw

Throw 语句用于抛出在方法内部的一个异常。该异常必须是方法定义中和 throw 语句定义在一起的一个异常的子类。

语法：

throw 异常对象

举例：

```
float calculateArea (float radius) throws
IllegalArgumentException {
    if (radius<0)
```

```
        throw (new
IllegalArgumentException (" Radius less than 0.");
    else
        return (radius * radius * PI);
}
```

Throws

关键词 Throws 能够指定在一个方法中发出的异常类型。

语法：

修饰语 返回类型 方法名（可选参数）**throws** ExceptionNames

举例：

String getName (InputStream in) **throws** IOException

Static

Static 修饰语使一个类或字段成为静态。不管被一个指定类所创建的实例的个数，仅有一个静态方法或字段的拷贝被创建。

语法：

static 返回类型 方法名（可选参数）

static 数据类型 方法名；

举例：

static void haltChanges (optionalParameters)

static Color backgroundColor;

Static

static 块是一个在对象创建后立即执行的代码集。它只处理静态方法和静态字段。

语法：

```
static
    statement (s)
```

举例：

```
static {
        type=prepare ();
        size=25;
```

}

Switch

switch 语句是具有多种选择的条件语句。

语法：

switch（变量名）{

case（值表达式 1）：statement（s）

case（值表达式 2）：statement（s）

default：statement（s）

}

举例：

```
char ans;
...
switch (ans) {
case 'Y': startOver ();
          break;
case 'n';
case 'N': cleanUp ();
default: System.out.println ("Invalid response.");
}
```

Synchronized

每个对象都有“锁”，它被一种操作掌握。任何同步操作都需把握此锁，以防止其他同步进程在它完成之前开始运行。

语法：

Synchronized 方法；

synchronized 返回类型

methodName（可选参数）

synchronized（对象名）

statement（s）

举例：

synchronized void changeValues（int size，int shape，String

```
name)
synchronized (runningThread) {
runningThread.name=newName;
}
```

Try

try 语句用于封装能够发出异常的代码。它应与 catch ( ) 语句一起使用也可以和 finally 语句一起使用。

语法：

```
try
    statement (s)
catch (常量列表)
    statement (s)
finally
    statement (s)
```

举例：

```
InputStream in;
int val;
...
try
    val=in.read () /in.read ();
catch (ArithmeticException e) {
    System.out.println ( "Invalid data.val set to 0.");
    val=0;
    }
catch (Exception e)
    System.out.println ( "Exception encountered, but not handled.");
finally {
    in.close ( );
    System.out.println ( "Stream closed.");
```

}

While

While 用于满足某种条件时执行循环操作。

语法：

while（布尔表达式）

statement（s）

while（布尔表达式）

statement（s）

举例：

```
FileInputStream din;
byte info;
while (info=din.read ()! =-1//End of File
    System.out.println (info);
while (stillValidData) {
    info=din.read ();
    stillValidData=checkData (ingo); //returns false if data is
not valid
```

附录　行为索引

行为索引

这里列出了一张问题解决表。第一列提出一般问题，解答的关键词放在第二列。可根据关键词查找每个条目的详细解释。

Animation

问题	解答
如何显示图形?	Graphics
如何使 applet 自己运行?	Runnable
如何开始 animation?	Thread
此手册有否 animation 例子?	Runnable

Applets

问题	解答
如何创建一个 applet?	Applet
如何交互使用 browser?	AppletContext
如何处理用户事件?	Event
哪一个方法捕捉用户事件?	Component

Applications

问题	解答
如何在屏幕上显示文本?	System. out
如何在客户端运行其他程序?	Runtime
如何处理文件?	File

如何读文件?	FileInputStream
如何写文件?	FileOutputStream

Communication

问题	解答
如何显示一个 URL?	AppletContext. show
如何处理 URLs?	URL
如何从一个 URL 装载信息?	URLconnection
如何与一个服务器通信?	Socket
如何创建一个 TCP socket?	ServerSocket
如何连续使用一个 socket 通信?	Thread

Image Processing

问题	解答
如何在 applet 中装载一个图象?	Appplet. getImage ( )
如何在应用中装载一个图象?	Toolkit. getImage ( )
如何操作一个图象?	ImageFiler
如何剪切一个图象?	CropImageFilter
如何由象素数据创建图象?	MemoryImageSource

Input and Output

问题	解答
是否有一个象 C 语言中 printf( )的类?	PrintStream
是否有一个容易的方法来解析流?	StreamTokenizer
如何从一个流中读取 doubles, strings, 等等?	DataInputStream

如何向一个流写 doubles，strings 等等?	DataOutputStream
如何把输入流连接在一起?	SequenceInputStream
如何交互使用环境（Java 控制台提示或一个浏览器）?	System

Math

问题	解答
如何执行标准的数学函数?	Math
如何创建随机数?	Random
如何把字符当作对象处理?	Character
如何把一个整数当作对象处理?	Integer

User Interfaces

问题	解答
从哪里寻找窗口组件?	java. awtpackage
如何创建一个弹出菜单?	Frame
如何安排屏幕组件的位置?	GridBagLayout
如何处理用户事件?	Event
哪一个方法捕捉用户事件?	Component
如何滚动序列屏幕?	Cardlayout
如何创建一个工具条菜单?	Menu

无忧书库书籍版权申明

无忧书库提供的所有书籍、资料版权归原作者和出版商所有。

本站提供下载的书籍仅供个人学习、参考之用，任何集体、个人不得用于商业用途。

请您在下载书籍 **24** 小时之后删除书籍，购买正版书籍！

本站书籍如有侵犯您的版权，请与我们联系，我们将尽快删除。联系信箱：**pcbook@51soft.com**。

无忧书库
http://pcbook.51soft.com