第一章　　Java 语言入门

Java 语言是一门很优秀的语言,具有面向对象,与平台无关,安全,稳定和多线程等优良特性,是目前软件设计中极为健壮的编程语言.Java 语言不仅可以用来开发大型的应用程序,而且特别适合于 Internet 的应用开发.Java 确确实实具备了"一旦写成处处可用"的特点,Java 已成为网络时代最重要的语言之一.本章将对 Java 语言做一个简单的介绍,并初步了解什么是 Java 应用程序,什么是 Java 小应用程序,有关的细节会在后续的章节中讨论.

1.1 Java 的诞生

Java 是 1995 年 6 月由 Sun 公司引进到我们这个世界的革命性的编程语言,它被美国的著名杂志 PC Magazine 评为 1995 年十大优秀科技产品.之所以称 Java 为革命性编程语言,是因为传统的软件往往与具体的实现环境有关,一旦环境有所变化就需要对软件作一番改动,耗时费力,而 Java 编写的软件能在执行码上兼容.这样,只要计算机提供了 Java 解释器,Java 编写的软件就能在其上运行.

Java 语言的出现是源于对独立于平台语言的需要,希望这种语言能编写出嵌入各种家用电器等设备的芯片上,且易于维护的程序.但是,人们发现当时的编程语言,比如 C,C++等都有一个共同的缺点,那就是只能对特定的 CPU 芯片进行编译.这样,一旦电器设备更换了芯片就不能保证程序正确运行,就可能需要修改程序并针对新的芯片重新进行编译.1990 年 Sun 公司成立了由 James Gosling 领导的开发小组,开始致力于开发一种可移植的,跨平台的语言,该语言能生成正确运行于各种操作系统,各种 CPU 芯片上的代码.他们的精心专研和努力促成了 Java 语言的诞生.Java 的快速发展得利于 Internet 和 Web 的出现,Internet 上有各种不同的计算机,它们可能使用完全不同的操作系统和 CPU 芯片,但仍希望运行相同的程序,Java 的出现标志着真正的分布式系统的到来.

注　印度尼西亚有一个重要的盛产咖啡的岛屿,中文名叫爪哇,开发人员为这种新的语言起名为 Java,其寓意是为世人端上一杯热咖啡.

1．2　Java 的特点

Java 是目前使用最为广泛的网络编程语言之一.它具有简单,面向对象,稳定,与平台无关,解释型,多线程,动态等特点.

简单　Java 语言简单是指这门语言既易学有好用.不要将简单误解为这门语言很干瘪.你可能很赞同这样的观点　英语要比阿了伯语言容易学.但这并不意味着英语就不能表达丰富的内容和深刻的思想,许多文学若贝尔奖的作品都是英文写的.如果你学习过 C++语言,你会感觉 Java 很眼熟,因为 Java 中许多基本语句的语法和 C++一样,像常用的循环语句,控制语句等和 C++几乎一样,但不要误解为 Java 是 C++的增强版,Java 和 C++是两种完全不同的语言,他们各有各的优势,将会长期并存下去,Java 语言和 C++语言已成为软件开发者应当掌握的语言.如果从语言的简单性方面看,Java 要比 C++简单,C++中许多容易混淆的概念,或

者被 Java 弃之不用了,或者以一种更清楚更容易理解的方式实现,例如,Java 不再有指针的概念.

面向对象 基于对象的编程更符合人的思维模式,使人们更容易编写程序.在实际生活中,我们每时每刻都与对象在打交道. 我们用的钢笔, 骑的自行车, 乘的公共汽车等. 而我们经常见到的卡车, 公共汽车, 轿车等都会涉及以下几个重要的物理量

可乘载的人数, 运行速度, 发动机的功率, 耗油量, 自重, 轮子数目等.

另外, 还有几个重要的功能

加速功能, 减速功能, 刹车, 转弯功能等.

我们也可以把这些功能称作是他们具有的方法,而物理量是它们的状态描述.仅仅用物理量或功能不能很好的描述它们.在现实生活中,我们用这些共有的属性和功能给出一个概念机动车类.一个具体的轿车就是机动车类的一个实例 对象 .Java 语言与其它面向对象语言一样,引入了类的概念,类是用来创建对象的模板,它包含被创建的对象的状态描述和方法的定义.我们将在第 4 章详细地讨论类,对象等概念.

与平台无关 与平台无关是 Java 语言最大的优势.其它语言编写的程序面临的一个主要问题是 操作系统的变化,处理器升级以及核心系统资源的变化,都可能导致程序出现错误或无法运行.Java 的虚拟机成功地解决了这个问题,Java 编写的程序可以在任何安装了 Java 虚拟机 JVM 的计算机上正确的运行,Sun 公司实现了自己的目标 "一次写成,处处运行".

解释型 我们知道 C,C++等语言,都是只能对特定的 CPU 芯片进行编译,生成机器代码,该代码的运行就和特定的 CUP 有关.例如,在 C 语言中,我们都碰到过类似下面的问题

int 型变量的值是 10 ,那么下面代码的输出结果是什么呢

```
printf("%d, %d", x, x=x+1)
```

如果上述语句的计算顺序是从左到右,结果是 10,11 但是,有些机器会从右到左计算,那么结果就是 11,11.

Java 不像 C++,它不针对特定的 CPU 芯片进行编译,而是把程序编译为称做字节码的一个"中间代码".字节码是很接近机器码的文件,可以在提供了 Java 虚拟机 JVM 的任何系统上被解释执行.Java 被设计成为解释执行的程序,即翻译一句,执行一句,不产生整个的机器代码程序.翻译过程如果不出现错误,就一直进行到完毕,否则将在错误处停止执行.同一个程序,如果是解释执行的,那么它的运行速度通常比编译为可执行的机器代码的运行速度慢一些.但是,对 Java 来说,二者的差别不太大,Java 的字节码经过仔细设计,很容易便能使用 JIT 即时编译方式 编译技术将字节码直接转化成高性能的本地机器码,Sun 公司在 Java 2 发行版中提供了这样一个字节码编译器---JIT(Just In Time),它是 Java 虚拟机的一部分.Java 运行系统在提供 JIT 的同时仍具有平台独立性,因而"高效且跨平台"对 Java 来说不再矛盾.

如果把 Java 的程序比做"汉语"的话, 字节码就相当于"世界语", 世界语不和具体的"国家"关, 只要这个"国家"提供了"翻译", 就可以再快速地把世界语翻译成本地语言.

多线程 Java 的特点之一就是内置对多线程的支持.多线程允许同时完成多个任务.实际上多线程使人产生多个任务在同时执行的错觉,因为,目前的计算机的处理器在同一时刻只能执行一个线程,但处理器可以在不同的线程之间快速地切换,由于处理器速度非常快,远远

超过了人接收信息的速度,所以给人的感觉好象多个任务在同时执行.C++没有内置的多线程机制,因此必须调用操作系统的多线程功能来进行多线程程序的设计.

安全 当你准备从网络上下载一个程序时,你最大的担心是程序中含有恶意的代码,比如试图读取或删除本地机上的一些重要文件,甚至该程序是一个病毒程序等.当你使用支持 Java 的浏览器时,你可以放心地运行 Java 的小应用程序 Java Applet ,不必担心病毒的感染和恶意的企图,Java 小应用程序将限制在 Java 运行环境中,不允许它访问计算机的其它部分.我们将在第 8 章详细讲述 Java 小应用程序.

动态 在学习了第 4 章之后,我们就会知道,Java 程序的基本组成单元就是类,有些类是自己编写的,有一些是从类库中引入的,而类又是运行时动态装载的,这就使得 Java 可以在分布环境中动态地维护程序及类库,而不像 C++那样,每当其类库升级之后,相应的程序都必须重新修改,编译.

1．3 安装 SUN 公司的 SDK

学习 Java 需要有一个程序开发环境.目前有许多很好的 Java 程序开发环境可用,包括来自 Sun,Borland,Sysmantec 公司的产品,例如,JBuild 等.但学习 Java 最好选用 Sun 公司的推出的软件开发工具箱 SDK 以前称作 Java 开发工具箱 JDK .目前 Sun 公司一发布了 SDK 的 1.4 版本,可以登陆到 Sun 公司的网站 http://java.sun.com,免费下载 SDK1.4 例如 j2sdk-1_4_1_02-windows-i586.exe .如果你购买某些 Java 书,也可能会得到一个含有 SDK 的光盘.如果安装 SDK 选择安装到 F \Jdk1.4 目录下,则会生成如下图 1.1 的目录结构.

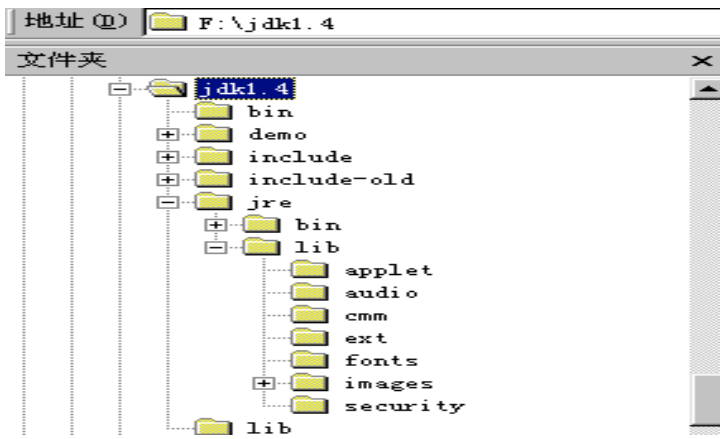


图 1.1 SDK 目录结构

现在,就可以编写 Java 程序,并进行编译,运行程序了,因为安装 SDK 的同时,我们的计算机就安装上了能运行 Java 程序的 Java 虚拟机.

SDK 的安装目录的 bin 文件夹中含有编译器 javac.exe ,解释器(java.exe)和一些其它的可执行文件,为了方便的使用编译器和解释器,应当将 bin 包含在 Path 的设置中,如果使用 Windows 操作系统可以在 Ms-dos 命令行键入下列命令后回车确定即可

```
Path  F:\jdk1.4\bin
```

你也可以在系统特性中设置 Path.对于 window2000,用鼠标右键点击”我的电脑”,弹出菜单,然后选择属性,弹出”系统特性”对话框,再单击该对话框中的高级选项,然后点击按钮”环境变量”,添加如下的系统环境变量

```
变量名  PATH,变量值  F:\jdk1.4\bin
```

如果曾经设置过环境变量 PATH,可点击该变量进行编辑操作,将需要的值加入即可.

对于 Win9x,用记事本编辑 Autoexec.bat 文件,将如下的设置语句加入即可,

```
PATH = C:\jdk1.3.1_01\bin;
```

但是,如果你的机器安装过一些商业化的 Java 开发产品或带有 Java 技术的一些产品,如 PB,Oracle 等,那么这些产品在安装后,可能会修改了 Path 的值,那么当你运行编译器或者解释器时,你可能运行这些产品所带的老版本的编译器或者解释器.因此如果想使用 SDK 的编译器或解释器,在设置 Path 的值时,将你需要的值写在前面.比如,某个软件用到的 Path 值是

```
d:\PB\jdk1.1.7\bin;
```

如果你不想删除这些产品对 Path 的设置,那么你可以重新编辑系统环境变量 Path 的值,并修改为

```
F:\jdk1.4\bin; d:\PB\jdk1.1.7\bin;
```

如果是 win9x 操作系统,就将 Autoexec.bat 文件的 path 修改为

```
path=F:\jdk1.4\bin; d:\PB\jdk1.1.7\bin;
```

也可以在 ms-dos 命令行键入下列命令后,回车确认.

```
Path  F:\jdk1.4\bin; d:\PB\jdk1.1.7\bin;
```

SDK 的安装目录的 JRE 文件夹中包含着 Java 应用程序运行时所需要的 Java 类库和虚拟机,这些类库被包含在一个 jre\lib 中的压缩文件 rt.jar 中.安装 SDK 一般不需要设置环境变量 classpath 的值,如果你的机器安装过一些商业化的 Java 开发产品或带有 Java 技术的一些产品,如 PB,Oracle 等,那么这些产品在安装后,也可能会修改了 classpath 的值,那么当你运行 java 应用程序时,你可能加载这些产品所带的老版本的类库,可能导致程序要加载的类无法找到,使你的程序出现运行错误.比如,某个软件用到的 classPath 值是

```
d:\PB\jdk1.1.7\jre\lib\classes.zip;.;
```

如果你不想删除这些产品的 classPath 设置,那么你可以重新编辑系统环境变量 classpath 的值,并修改为

```
F:\jdk1.4\jre\lib\rt.jar;.;d:\PB\jdk1.1.7\jre\lib\rt.jar;
```

如果是 win9x 操作系统,就将 Autoexec.bat 文件的 classpath 修改为

```
set classpath=F:\jdk1.4\jre\lib\rt.jar;.;d:\PB\jdk1.1.7\ jre\lib\classes.zip
```

也可以在 ms-dos 命令行键入下列命令后,回车确认.

```
set classpath=F:\jdk1.4\jre\lib\rt.jar;.;
```

注　　classpath 环境变量设置中的".;"是指可以加载应用程序当前目录中的类.

如果你的计算机安装过两个版本的 SDK,比如 SDK1.3 和 SDK1.4,若运行出现问题,可以在 ms-dos 命令行分别键入下列命令回车确认

```
Path  f:\jdk1.4\bin
Set Classpath=f:\jdk1.4\jre\lib\rt.jar;.;
```

如果你只想运行别人的 Java 程序可以只安装 Java 运行环境 JRE,JRE 由 Java 虚拟机,Java 的核心类,以及一些支持文件组成.可以登陆 Sun 的网站免费下载 Java 的 JRE,如果安装时选择了默认的安装路径就会形成如下图 1.2 所示的目录结构.

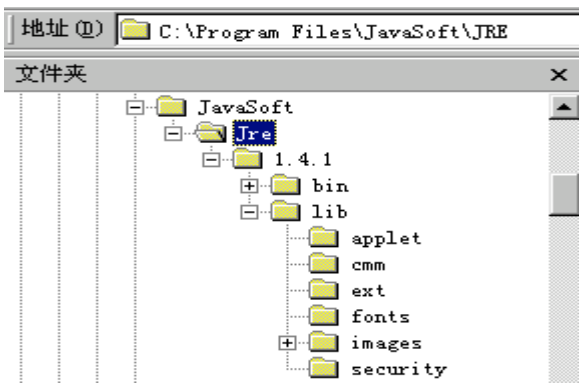


图 1.2　JRE 目录结构

注　建议下载 Sun 公司的 Java 类库文档,例如　j2sdk-1_4_1-doc.

1．4　一个 Java 程序的开发过程

Java 程序的开发过程如图 1.3 所示.

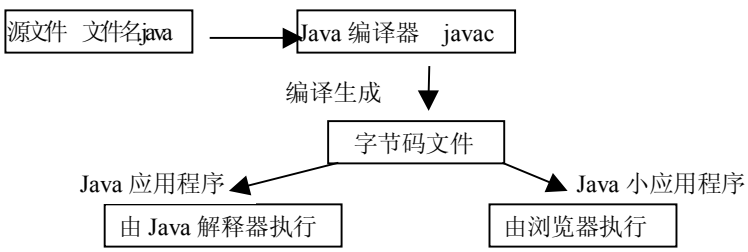


图 1.3　Java 程序的开发过程

注 字节码文件是与平台无关的二进制码,执行时由解释器解释成本地机器码,解释一句,执行一句.

1 编写源文件 使用一个文字编辑器,如 Edit 或记事本,来编写源文件.不可使用 Word 编辑器,因它含有不可见字符.将编好的源文件保存起来,源文件的扩展名必须是 java.

2 编译 Java 源程序 使用 Java 编译器 javac.exe 编译源文件得到字节码文件.

3 运行 Java 程序 Java 程序分为两类—Java 应用程序和 Java 小应用程序,Java 应用程序必须通过 java 解释器 java.exe 来解释执行其字节码文件 Java 小应用程序必须通过支持 Java 标准的浏览器来解释执行.你马上就会知道怎样使用解释器和浏览器来运行程序,普遍使用的 Netscape Navigator 和 Microsoft Explorer 都完全支持 Java.

1．5 一个简单的 Java 应用程序的开发过程

1 编写源文件

```
public class Hello
{
    public static void main (String args[ ])
    {
     System.out.println("你好,很高兴学习 Java");
    }
}
```

注 Java 源程序中语句所涉及到的小括号及标点符号都是英文状态下输入的括号和标点符号,比如"你好,很高兴学习 Java"中的引号必须是英文状态下的引号,而字符串里面的符号不受汉或英的限制.

- 一个 Java 源程序是由若干个类组成的.如果你学过 C 语言,你就会知道一个 C 源程序是由若干个函数组成的.上面的这个 Java 应用程序简单到只有一个类,类的名字是由我们起的,叫 Hello.
- class 是 Java 的关键字,用来定义类的.public 也是关键字,说明 Hello 是一个 public 类 我们将会系统的学习类的定义和使用 .第一个大括号和最后一个大括号以及它们之间的内容叫做类体.
- public static void main String args[ ] 是类体中的一个方法,之后的两个大括号以及之间的内容叫做方法体.一个 Java 应用程序必须有一个类且只有一个类含有这样的 main 方法,这个类称为应用程序的主类.public,static 和 void 分别是对 main 方法的说明.在一个 Java 应用程序中 main 方法必须被说明为 public static void.
- String args[]声明一个字符串类型的数组 args[] 注意 String 的第一个字母是大写的 ,它是 main 方法的参数 以后会学习怎样使用这个参数,见 5.9 .main 方法是程序开始

执行的位置.

现在将源文件保存到 C:\1000\ 中,并命名为 Hello.java.注意不可写成 hello.java,因为 Java 语言是区分大小写的.

源文件的命名规则是这样的,如果源文件中有多个类,那么只能有一个类是 public 类.如果有一个类是 public 类,那么源文件的名字必须与这个类的名字完全相同,扩展名是.java.如果源文件没有 public 类,那么源文件的名字只要和某个类的名字相同,并且扩展名是.java 就可以了.

2 编译

当创建了 Hello.java 这个源文件后,就要使用 Java 编译器 javac.exe 对其进行编译.

```
C:\1000\>javac Hello.java
```

编译完成后生成一个 Hello.class 文件,该文件称为字节码文件.这个字节码文件 Hello.class 将被存放在与源文件相同的目录中.

如果 Java 源程序中包含了多个类,那么用编译器 javac 编译完源文件后将生成多个扩展名为.class 的文件,每个扩展名是.class 的文件中只存放一个类的字节码,其文件名与该类的名字相同.这些字节码文件将被存放在与源文件相同的目录中.

如果你对源文件进行了修改,那么你必须重新编译,再生成新的字节码文件.

注 如果你在安装 SDK1.4 时没有另外指定目录,javac.exe 和 java.exe 被存放在 C:\jdk1.4\bin 下,如果你想在任何目录下能使用编译器和解释器,应在 dos 提示符下键入下列命令 C:\>path c:\jdk1.4\bin.

3 运行

使用 Java 解释器 java.exe 运行这个应用程序程序.

```
C:\1000\>java Hello
```

屏幕将显示如下信息

```
你好,很高兴学习 Java
```

注 当 Java 应用程序中有多个类时,java 后的类名必须是包含了 main 方法的那个类的名字,即主类的名字.

我们再看一个简单的 Java 应用程序.也许你现在还看不懂这个程序,但你必须知道怎样命名,保存源程序 怎样使用编译器编译源程序 怎样使用解释器运行程序.

- 源程序

```
public class people
{
    float hight,weight;
```

```
C:\1000>javac people.java

C:\1000>java A
重量200.0身高1.7
大头一只大嘴两只大耳朵
师傅，咱们别去西天了，改去月宫吧
```

```
    String head, ear, mouth;
    void speak(String s)
    {
      System.out.println(s);
    }
}
class A
{ public static void main(String args[])
   {people zhubajie;
    zhubajie=new people();
    zhubajie.weight=200f;
    zhubajie.hight=1.70F;
    zhubajie.head="大头";
    zhubajie.ear="两只大耳朵";
    zhubajie.mouth="一只大嘴";
    System.out.println("重量"+zhubajie.weight+"身高" +zhubajie.hight);
    System.out.println(zhubajie.head+zhubajie.mouth+zhubajie.ear);
    zhubajie.speak("师傅,咱们别去西天了,改去月宫吧");
    }
}
```

我们必须把源文件保存起来并命名为 **people.java**　回忆一下源文件起名的规定　.假设保存 **people.java** 在 **C:\1000** 下.

- 编译源文件

```
c:\1000\>javac  people.java
```

如果编译成功,你的目录 **1000** 下就会有 **people.class** 和 **A.class** 这两个字节码文件了.

- 执行

```
c:\1000\>java A
```

java 命令后的名字必须是含有 **main** 方法的那个类的名字,运行效果如图 **1.1.**

1．6　一个简单的 Java 小应用程序　Java Applet

- **编写源程序**

```
import java.applet.*;
import java.awt.*;
public class boy extends Applet
{
```

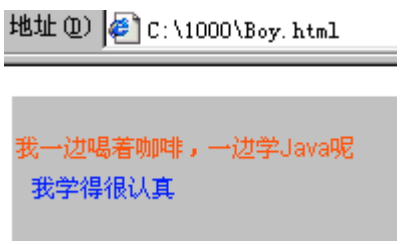


图 1.2　通过 web 页运行小应用程序

```
    public void paint(Graphics g)
     {
      g.setColor(Color.red);
      g.drawString("我一边喝着咖啡,一边学 Java 呢",2,30);
      g.setColor(Color.blue);
      g.drawString("我学得很认真",10,50);
     }
}
```

一个 Java Applet 也是由若干个类组成的,一个 Java Applet 不再需要 main 方法,但必须有且只有一个类扩展了 Applet 类,即它是 Applet 类的子类.我们把这个类叫做这个 Java Applet 的主类,Java Applet 的主类必须是 public 的 我们将会系统学习类和子类 .Applet 类是系统提供的类.当我们保存上面的源文件时,必须命名为 boy.java.假设我们保存 boy.java 在 c:\1000 目录下.

注 上述源程序中我们使用了 import 语句,这是因为我们要使用系统提供给我们的 Applet 类.Applet 类在包 java.applet 中.包 java.applet 中有很多类,Java 语言把一些类放在一起叫做一个包,这里 java.applet 是一个包的包名,关于包以后还会讲解.如果我们不使用 import 语句,主类必须写成 public boy extends java.applet.Applet.Graphics 是包 java.awt 中的一个类.

- **编译**

```
c:\1000\>javac  boy.java
```

编译成功后,文件夹 1000 下会生成一个 boy.class 文件.如果源文件有多个类,将生成多个 class 文件,都和源文件在同一文件夹里.

如果你对源文件进行了修改,那么你必须重新编译,再生成新的字节码文件.

- **运行**

Java Applet 必须由浏览器来运行,因此我们必须编写一个超文本文件 含有 applet 标记的 web 页 通知浏览器来运行这个 Java Applet.

下面是一个最简单的一个 html 文件,通知浏览器运行我们的 Java Apple.我们使用记事本编辑如下

```
<applet  code=boy.class  height=100  width=300>
</applet>
```

超文本中的标记 <apple… > 和</applet>通知浏览器运行一个 Java Applet,code 通知浏览器运行哪个 Java Applet.code 的"="后面是主类的字节码文件,当然这个字节码文件的扩展名是.class,而它的名字和源文件的名字是相同的.width,height 规定了这个 Java Apple 的宽度和高度,单位是象素.要想让浏览器运行一个 Java Applet,<applet…> </applet> 标记中的 code,height,width 都是必须的.另外还有一些可选的项,如 vspace,设置小程序与

其周围对象的垂直距离,hspace,设置水平距离,等等.

现在我们把上面编辑的文件命名为 Boy.html 扩展名必须是 html,名字不必是 Boy,你可以起一个你喜欢的名字 .把 Boy.html 保存在 C:\1000,即和 boy.class 在同一目录里.如果不是这样,你必须在文件 Boy.html 中增加选项 codebase,来指定你的小程序中的.class 文件所在的目录.

现在可以使用浏览器打开文件 Boy.html 来运行小程序了,效果如图 1.2.

注 也可以使用 JDK 提供的 appletviewer 来调试小程序,如,在 DOS 命令行执行

```
c:\1000\appletviewer  Boy.html.
```

g.drawString "我一边喝着咖啡,一边学 Java 呢",5,30 的作用是在程序中画字符串,数字 5 和 30 的意思是 从距小程序左面 5 个像素,距上面 30 像素的位置开始从左到右的方向画字符串 "我一边喝着咖啡,一边学 Java 呢".

1．7 什么是 JSP

JSP 是 Java Server Pages 的缩写,是由 Sun 公司 1999 于年推出的一种动态网页技术标准.JSP 是基于 Java Servlet 以及整个 Java 体系的 Web 开发技术,利用这一技术可以建立安全,跨平台的先进动态网站,这项技术还在不断的更新和优化中.你可能对 Microsoft 的 ASP 比较熟悉,ASP 的全名是 Active Server Pages,也是一个 Web 服务器端的开发环境,可以开发出动态的,高性能的 Web 服务应用程序.JSP 和 ASP 技术非常相似,ASP 的编程语言是 VBScript,JSP 使用的是 Java.与 ASP 相比,JSP 以 Java 技术为基础,又在许多方面做了改进,具有动态页面与静态页面分离,能够脱离硬件平台的束缚,以及编译后运行等优点,完全克服了 ASP 的脚本级执行的缺点.我们相信 JSP 会逐渐成为 Internet 上的主流开发工具.

需要强调的一点是 要想真正地掌握 JSP 技术,必须有较好的 Java 基础,但学习 Java 的目的并不只是为了掌握 JSP.

可以访问 Sun 公司的站点 http://java.sun.com详细了解 JSP.

习 题一

1 开发与运行 Java 程序需要经过哪些主要步骤和过程

2 怎样区分应用程序和小应用程序 应用程序的主类或小应用程序的主类必须用 public 修饰吗

3 Java 程序是由什么组成的 一个程序中必须要有 public 类吗 Java 源文件的命名规则是怎样的

4 在运行小程序的 html 文件中可以使用 codebase 属性指定小程序的字节码所驻留的目录.如果不使用 codebase 属性,小程序的字节码文件必须和运行它的 html 在同一目录中.编写一个小程序并将小程序的字节码存放在某个目录中,比如 c:\Boy 把运行该小程序的 html 文件 注意其中的 codebase 属性 :

```
<applet code=你的小程序的字节码  wdith=200  height=300  codebase=c:\Boy>
```

```
</applet>
```

存放在另一个目录中　查阅有关编写网页方面的书籍,哪里有更详细的关于怎样在一个网页中嵌入一个小应用程序的讲解　.

第二章　　标识符,关键字和数据类型

1．8　标识符和关键字

1. 标识符

用来标识类名,变量名,方法名,类型名,数组名,文件名的有效字符序列称为标识符.简单地说,标识符就是一个名字.

Java 语言规定标识符由字母,下划线,美元符号和数字组成,并且第一个字符不能是数字字符.下列都是合法的标识符

```
Girl_$, www_12$, $23boy.
```

标识符中的字母是区分大小写的,Boy 和 boy 是不同的标识符

Java 语言使用 unicode 标准字符集,最多可以识别 65535 个字符,unicode 字符表的前 128 个字符刚好是 ASCII 表.每个国家的"字母表"的字母都是 unicode 表中的一个字符,比如汉字中的"你"字就是 unicode 表中的第 20320 个字符.

Java 所谓的字母包括了世界上任何语言中的"字母表",因此,Java 所使用的字母不仅包括通常的拉丁字母 a,,b,c 等,也包括汉语中的汉字,日文里的片假名,平假名,朝鲜文以及其他许多语言中的文字.

2. 关键字

关键字就是 Java 语言中已经被赋予特定意义的一些单词.不可以把这类词作为名字来用.Java 的关键字有

abstract boolean break byte case catch char class continue do double else extends false find finally float for implements import instanceof int interface long nativenew null package private public return short static super switch synchronized this throw true try void while.

1．9　Java 语言基本数据类型

1. 逻辑类型

- 常量　true ,false.
- 变量的定义

使用关键字 boolean 来定义逻辑变量

```
boolean  x;  boolean tom_12;
```

也可以一次定义几个

```
boolean  x, tom, jiafei, 漂亮
```

x,tom,jiafei,漂亮都是变量的名字.定义时也可以赋给初值

```
boolean x=true,tom=false,漂亮=true,jiafei
```

2.整数类型

- 常量　123,6000　十进制　,077(八进制),0x3ABC(十六进制).
- 整型变量的定义分为 4 种

1　int 型

使用关键字 int 来定义 int 型整型变量

```
int x
int tom_12
```

也可以一次定义几个

```
int x,tom,jiafei,漂亮
```

x,tom,jafei,漂亮都是名字.定义时也可以赋给初值

```
int x= 12,tom=-1230,漂亮=9898,jiafei
```

对于 int 型变量,内存分配给 4 个字节　byte　,一个字节由 8 位(bit)组成,4 个字节占 32 位(bit).bit 有两种状态,分别用来表示 0,1.这样计算机就可以使用 2 进制数来存储信息了.内存是一种特殊的电子元件,如果把内存条放大到摩天大楼那么大,那么它的基本单位　字节,就好比是大楼的房间,每个房间的结构都是完全相同的,一个字节由 8 个能显示两种状态的 bit 组成,就好比每个房间里有 8 个灯泡,每个灯泡有两种状态　亮灯灭灯.

对于

```
int x=7;
```

内存存储状态如下

```
00000000 00000000 00000000 00000111.
```

最高位　左边的第一位　是符号位,用来区分正数或负数,正数使用原码表示,最高位是 0.负数用补码表示,最高位是 1,
例如,

```
int x=-8
```

内存的存储状态的如下

```
11111111 11111111 11111111 11111000.
```

要得到-8 的补码,首先得到 7 的原码,然后将 7 的原码中的 0 变成 1,1 变成 0 就是-8 的补码.因此,int 型变量的取值范围是　-2^{31}~2^{31}-1.

2 byte 型

使用关键字 byte 来定义 byte 型整型变量

```
byte  x    byte tom_12
```

也可以一次定义几个

byte x,tom,jiafei,漂亮

x,tom,jafei,漂亮都是名字.定义时也可以赋给初值

```
byte x= -12,tom=28,漂亮=98,jiafei
```

注 对于 byte 型变量,内存分配给 1 个字节,占 8 位,因此 byte 型变量的取值范围是: -2^7 2^7-1.

3 short 型

使用关键字 short 来定义 short 型整型变量

```
short x    short tom_12
```

也可以一次定义几个

```
short x,tom,jafei,漂亮
```

x,tom,jafei,漂亮都是名字.定义时也可以赋给初值

```
short x= 12,tom=1234,漂亮=9876,jiafei
```

注 对于 short 型变量,内存分配给 2 个字节,占 16 位,因此 short 型变量的取值范围是 -2^{15} $2^{15}-1$.

4 long 型

使用关键字 long 来定义 long 型整型变量

```
long x    long tom_12
```

也可以一次定义几个

```
long x,tom,jiafei,漂亮
```

x,tom,jiafei,漂亮都是名字.定义时也可以赋给初值

```
long x= 12,tom=1234,漂亮=9876,jiafei
```

注 对于 long 型变量,内存分配给 8 个字节,占 64 位,因此 long 型变量的取值范围是 -2^{63} $2^{63}-1$.

3. 字符类型

- **常量 A b ? ! 9 好 \t ,**

Java 使用 unicode 字符集,所以常量共有 65535 个.

- 变量的定义

使用关键字 char 来定义字符变量

```
char  x,  char tom_12
```

也可以一次定义几个

```
char  x,tom,jafei,漂亮
```

x,tom,jafei,漂亮都是变量名字.定义时也可以赋给初值

```
char x=  A  ,tom=  家  ,漂亮=  假  ,jiafei
```

char 型变量,内存分配给 2 个字节,占 16 位,最高为不用来表示符号,没有负数的 char.char 型变量的取值范围是 0~65536.对于

```
char x='a';
```

那么内存 x 中存储的是 97,97 是字符 a 在 unicode 表中的排序位置.因此,允许将上面的语句写成

```
char x=97;
```

要观察一个字符在 unicode 表中的顺序位置,必须使用 int 类型显示转换,如(int)'a'.不可以使用 short 型转换,因为 char 的最高位不是符号位.如果要得到一个 0~65536 之间的数所代表的 unicode 表中相应位置上的字符也必须使用 char 型显示转换.

在下面的例子中,分别用显示转换来显示一些字符在 unicode 表中的位置,以及某些位置上的字符.

例子 1(效果如图 2.1)

```
public class  Example2_1
{
   public static void main (String args[ ])
    {
     char chinaWord='你',japanWord='  ';
     int  p1=20328,p2=12358;
     System.out.println("汉字\'你\'字在 unicode 表中的顺序位置:"+(int)chinaWord);
     System.out.println("日语\'  \'字在 unicode 表中的顺序位置:"+(int)japanWord);
     System.out.println("unicode 表中第 20328 位置上的字符是:"+(char)p1);
     System.out.println("unicode 表中第 12358 位置上的字符是:"+(char)p2);
    }
}
```

```
F:\8000>java Example2_1
汉字'你'字在unicode表中的顺序位置:20320
日语'ぁ'字在unicode表中的顺序位置:12353
unicode表中第20328位置上的字符是:佈
unicode表中第12358位置上的字符是:ぁ
```

图 2.1　字符的位置

4. 浮点类型　实型

浮点型分两种

1 float 型

- 常量 123.5439f,12389.987F,123.0f,2e40f 2 乘 10 的 40 次方,科学计数法
- 变量的定义

使用关键字 float 来定义 float 型变量

```
float x    float tom_12
```

也可以一次定义几个

```
float x,tom,jiafei,漂亮
```

x,tom,jiafei,漂亮都是名字.定义时也可以赋给初值

```
float x= 12.76f,tom=1234.987f,漂亮=9876.0f,jiafei
```

注 对于 float 型变量,内存分配给 4 个字节,占 32 位,float 型变量的取值范围大约是 10^{-38} 10^{38} ,-10^{38} 10^{-38}.

2 double 型

- 常量 12389.5439d d 可以省略 ,12389908.987,123.0,6e-140 6 乘 10 的-140 次方,科学计数法
- 变量的定义

使用关键字 double 来定义 double 型变量

```
double x    double tom_12
```

也可以一次定义几个

```
double x,tom,jiafei,漂亮
```

x,tom,jiafei,漂亮都是名字.定义时也可以赋给初值

```
double  x= 12.76,tom=1234098.987,漂亮=9876.098d,jiafei
```

注 double 型变量,内存分配给 8 个字节,占 64 位,double 型变量的取值范围大约是(10^{-308} 10^{308})

5. 基本数据类型的转换

当我们把一种基本数据类型变量的值赋给另一种基本类型变量时,就涉及到数据转换.基本类型数据的下列类型会涉及到数据转换,不包括逻辑类型和字符类型.我们将这些类型按精度从"底"到"高"排列了顺序.

```
byte  short  int  long  float  double
```

当把在级别低的变量的值赋给级别高的变量时,系统自动完成数据类型的转换.例如,

```
float x=100;
```

如果输出 x 的值,结果将是 100.0

例如

```
int x=50;
float y;
y=x;
```

如果输出 y 的值,结果将是 50.0.

当把在级别高的变量的值赋给级别底变量时,必须使用显示类型转换运算.显示转换的格式

类型名 要转换的值

例如,

```
int x=(int)23.89;
long y=(long)34.98F;
```

如果输出 x,y 的值将是 23 和 34,强制转换运算可能导致精度的损失.

例子 2

```
public class Example2_2
{ public static void main (String args[ ])
    { byte  a=120;   short b=255;
      int c=2200;    long d=8000;
      float f;
      double g=123456789.123456789;
     b=a;
     c=(int)d;
     f=(float)g;   //导致精度的损失.
     System.out.print("a=  "+a);    System.out.println("  b=  "+b);
     System.out.print("c=  "+c);    System.out.println("  d=  "+d);
     System.out.println("f=  "+f);  System.out.println("g=  "+g);
    }
}
```

```
F:\8000>java Example2_2
a=  120  b=  120
c=  8000  d=  8000
f=  1.23456792E8
g=  1.2345678912345679E8
```

图 2.2 基本数据的转换与输出

习 题 二

1 上机运行下列程序,注意观察输出的结果.

```
public class E
{ public static void main (String args[ ])
    { for(int i=20302;i<=20322;i++)
```

```
        { System.out.println((char)i);
        }
      }
  }
```

2.System.out.println(" 你 好 ");可 输 出 串 值 , 也 可 以 使 用 System.out.println()输出变量或表达式的值,只需使用并置符号:"+"将变量,表达式或一个常数值与一个字符串并置即可,如 **System.out.println(" "+x); System.out.println(":"+123+ "大于"+122);**等.上机调试下列程序,注意观察结果,特别注意 **System.out.print()**和 **System.out.println()**的区别.

```
public class OutputData
{  public static void main(String args[])
   {  int x=234,y=432;
      System.out.println(":"+x+"<"+2*x);
      System.out.print("我输出结果后不回车");
      System.out.println("我输出结果后自动回车到下一行");
      System.out.println("x+y= "+(x+y));
      System.out.println("?"+x+y+"=234432");
   }
}
```

第三章　　运算符,表达式和语句

3．1　运算符与表达式

Java 提供了丰富的运算符,如算术运算符,关系运算符符,逻辑运算符,位运算符等.本节将介绍大部分运算符,某些运算符,比如 instanceof 归类运算符将在后续的章节中介绍.

1. 算术运算符与算术表达式

1　加减运算符　+,-.例如　2+39,908.98-23 等.

加减运算符是双目运算符,即连接两个操作元的运算符.加减运算符的结合方向是从左到右.例如: 2+3-8,先计算 2+3,然后再将得到的结果减 8.加减运算符的操作元是整型或浮点型数据,加减运算符的优先级是 4 级.

2　乘,除和求余运算符　*,/,%.例如　2*39　908.98/23 等.

,/,%运算符是双目运算符,即连接两个操作元的运算符.,/,%运算符的结合方向是从左到右,例如　2*3/8,先计算 2*3,然后再将得到的结果除以 8.乘除运算符的操作元是整型或浮点型数据.*,/,%运算符的优先级是 3 级.

用算术符号和括号连接起来的符合 java 语法规则的式子,如 x+2*y-30+3*(y+5).

2　自增, 自减运算符　++, --

自增,自减运算符是单目运算符,可以放在操作元之前,也可以放在操作元之后.操作元必须是一个整型或浮点型变量.作用是使变量的值增 1 或减 1,如

++x,--x 表示在使用 x 之前,先使 x 的值加　减　1　.

x++,x—表示在使用 x 之后,使 x 的值加　减　1.

粗略的看,++x 和 x++的作用相当于 x=x+1.但++x 和 x++的不同之处在于,++x 是先执行 x=x+1 再使用 x 的值,而 x++是先使用 x 的值再执行 x=x+1.如果 x 的原值是 5,则

对于 y=++x　y 的值为 6.

对于 y=x++　y 的值为 5,然后 x 的值变为 6.

3　算术混合运算的精度

精度从"底"到"高"排列的顺序是

```
byte  short int long float double
```

Java 将按运算符两边的操作元的最高精度保留结果的精度,例如

5/2 的结果是 2,要想得到 2.5,必须写成 5.0/2 或 5.0f/2.

char 型数据和整型数据运算结果的精度是 int.例如

```
byte x=7;
```

那么

```
'B'+x;
```

的结果是 int 型,因此下列写法是不正确的,

```
char  ch='B'+x;
```

应当写成

```
char ch=(char)('B'+x);
```

4　关系运算符与关系表达式

关系运算符用来比较两个值的关系.关系运算符的运算结果是 boolean 型,当运算符对应的关系成立时,运算结果是 true,否则是 false.例如,10<9 的结果是 false,5>1 的结果是 true,3!=5 的结果是 true,10>20-17 的结果为 true,因为算术运算符的级别高于关系运算符,10>20-17 相当于 10>　20-17　,结果当然是 true.

结果为数值型的变量或表达式可以通过关系运算符　如表 3.1 所示　形成关系表达式.如,4>8,(x+y)>80.

表 3.1　　关系运算符

运算符	优先级	用法	含义	结合方向
>	6	op1>op2	大于	左到右
<	6	op1<op2	小于	左到右
>=	6	op1>=op2	大于等于	左到右
<=	6	op1<=op2	小于等于	左到右
==	7	op1==op2	等于	左到右
!=	7	op1!=op2	不等于	左到右

5　逻辑运算符与逻辑表达式

逻辑运算符包括　&&,||,!.其中&&,||为二目运算符,实现逻辑与,逻辑或　为单目运算符,实现逻辑非.逻辑运算符的操作元必须是 boolean 型数据 ,逻辑运算符可以用来连接关系表达式.

表 3.2 给出了逻辑运算符的用法和含义

表 3.2　　逻辑运算符

运算符	优先级	用法	含义	结合方向
&&	11	op1&&op2	逻辑与	左到右
||	12	op1||op2	逻辑或	左到右
!	2	!op	逻辑非	右到左

结果为 boolean 型的变量或表达式可以通过逻辑运算符合成为逻辑表达式.

用逻辑运算符进行逻辑运算,结果如表 3.3 所示.

表 3.3　　用逻辑运算符进行逻辑运算

op1	op2	op1&&op2	op1||op2	!op1
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

例如,2>8&&9>2 的结果为 false,2>8||9>2 的结果为 true.由于关系运算符的级别高于&&,||的级别,2>8&&8>2 相当于 2>8 && 9>2 .

逻辑运算符"&&"和"||"也称做短路逻辑运算符,这是因为当 op1 的值是 false 时,"&&"运算符在运算时不再去计算 op2 的值,直接就得出 op1&&op2 的结果是 false.当 op1 的值是 true 时,"||"运算符号在运算时不再去计算 op2 的值,直接就得出 op1|op2 的结果是 true.比如,x 的初值是 1,那么经过下列逻辑比较运算后,

```
((y=1)==0))&&((x=6)==6));
```

x 的值仍然是 1.经过下列逻辑比较运算后,

```
((y=1)==1))&&((x=6)==6));
```

x 的值将变为 6.

6 赋值运算符与赋值表达式

赋值运算符 =.

赋值运算符是双目运算符,左面的操作元必须是变量,不能是常量或表达式.设 x 是一个整型变量,y 是一个 boolean 型变量,x=20 和 y = true 都是正确的赋值表达式,赋值运算符的优先级较低,是 14 级,结合方向右到左.赋值表达式的值就是"="左面变量的值.注意不要将赋值运算符"="与等号运算符"=="混淆.

7 位运算符

我们知道整型数据在内存中以 2 进制的形式表示,比如一个 int 型的变量在内存中占 4 个字节共 32 位,int 型数据 7 的 2 进制表示是

```
00000000 00000000 00000000  00000111
```

左面最高位是符号位,最高位是 0 表示正数,1 表示负数.负数采用补码表示,比如-8 的进制是

```
111111111 111111111 1111111 11111000
```

这样我们就可以对整型数据进行按位的运算,比如,对两个整型数据对应的位进行运算得到一个新的整型数据

1 "按位与"运算符

"&"是双目运算符,对两个整型数据 a,b 按位进行运算,运算结果是一个整型数据 c.运算法则是 如果 a,b 两个数据对应位都是 1,则 c 的该位是 1,否则是 0.如果 b 的精度高于 a,那么结果 c 的精度和 b 相同.

例如

```
    a   00000000  00000000  00000000  00000111
    b   10000001  10100101  11110011  10101011
&
    c   00000000  00000000  00000000  00000011
```

2 ”按位或”运算符

“|”是双目运算符.对两个整型数据 a,b 按位进行运算,运算结果是一个整型数据 c.运算法则是 如果 a,b 两个数据对应位都是 0,则 c 的该位是 0,否则是 1.如果 b 的精度高于与 a,那么结果 c 的精度和 b 相同.

3 ”按位非”运算符

“~”是单目运算符.对一个整型数据 a 按位进行运算,运算结果是一个整型数据 c.运算法则是 如果 a 对应位都是 0,则 c 的该位是 1,否则是 1.

“按位异或”运算符

4 ”^”是双目运算符.对两个整型数据 a,b 按位进行运算,运算结果是一个整型数据 c.运算法则是 如果 a,b 两个数据对应位相同,则 c 的该位是 0,否则是 1.如果 b 的精度高于与 a,那么结果 c 的精度和 b 相同.

由异或运算法则可知

a^a=0,

a^0=a.

因此,如果 c=a^b,那么 a=c^b,即用同一个数对数 a 进行两次”异或”运算的结果又是数 a.

在下面的例子 1 中,利用”异或”运算的性质,对几个字符进行加密并输出密文,然后再解密.

例子 1 效果如图 3.1

```
class Example3_1
{  public static void main(String args[])
    {char a1='十',a2='点',a3='进',a4='攻';
     char secret='8';
     a1=(char)(a1^secret);   a2=(char)(a2^secret);
     a3=(char)(a3^secret);   a4=(char)(a4^secret);
     System.out.println("密文:"+a1+a2+a3+a4);
     a1=(char)(a1^secret);   a2=(char)(a2^secret);
     a3=(char)(a3^secret);  a4=(char)(a4^secret);
     System.out.println("原文:"+a1+a2+a3+a4);
    }
}
```

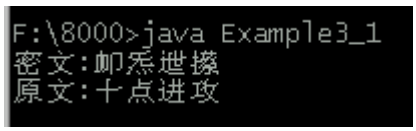


图 3.1 用”异或”加密,解密

位运算符也可以操作逻辑型数据,法则是

当 a,b 都是 true 时,a&b 是 true,否则 a&b 是 false.
当 a,b 都是 false 时,a|b 是 false,否则 a|b 是 true.
当 a 是 true 时,~a 是 false　当 a 是 false 时,~a 是 true.

位运算符在操作逻辑型数据时,与逻辑运算符&&,||,, 不同的是 位运算符要计算完 a 和 b 的之后再给出运算的结果.比如,x 的初值是 1,那么经过下列逻辑比较运算后,

```
((y=1)==0))&&((x=6)==6));
```

x 的值仍然是 1,但是如果经过下列位运算之后,

```
((y=1)==0))&((x=6)==6));
```

x 的值将是 6.

位运算符也可以操作字符数据,但运算结果是 int 型数据,例如

```
char x='a',y='b';
```

那么

```
x^y,x&y,x^y,  x
```

的结果是 int 型.

通过下面的例子 可以比较出短路逻辑运算和位运算的不同.

例子 2 效果如图 3.2

```
class Example3_2
{  public static void main(String args[])
   { int x,y=10;
    if(((x=0)==0)||((y=20)==20))
     { System.out.println("现在 y 的值是:"+y);
     }
     int a,b=10;
     if(((a=0)==0)|((b=20)==20))
     { System.out.println("现在 b 的值是:"+b);
     }
   }
}
```

```
F:\8000>java Example3_2
现在y的值是:10
现在b的值是:20
```

图 3.2　逻辑运算与位运算的不同

8　instanceof 运算符

该运算符是双目运算符,左面的操作元是一个对象 右面是一个类.当左面的对象是右面的类创建的对象时,该运算符运算的结果是 true ,否则是 false.

9　运算符综述

Java 的表达式就是用运算符连接起来的符合 Java 规则的式子.运算符的优先级决定了表达式中运算执行的先后顺序.例如,x<y&&!z 相当于(x<y)&&(!z),没有必要去记忆运算符号的优先级别,在编写程序时可尽量的使用括号　　运算符号来实现你想要的运算次序,以免产生难以阅读或含糊不清的计算顺序.运算符的结合性决定了并列相同级别的运算符的先后顺序,例如,加减的结合性是从左到右,8-5+3 相当于　8-5　+3.逻辑否运算符　的结合性是右到左,　x 相当于!(!x).表 3.4 是 Java 所有运算符的优先级和结合性,有些运算符我们没有介绍,可参见相关书籍.

表 3.4　　运算符的优先级和结合性

优先级	描述	运算符	结合性
1	分隔符	[ ]　( ) .　,　;	
2	对象归类,自增自减运算,逻辑非	instanceof　++ --	右到左
3	算术乘除运算	*　/　%	左到右
4	算术加减运算	+　-	左到右
5	移位运算	>>　<<　>>>	左到右
6	大小关系运算	<　<=　>　>=	左到右
7	相等关系运算	==　!=	左到右
8	按位与运算	&	左到右
9	按位异或运算	^	左到右
10	按位或	|	左到右
11	逻辑与运算	&&	左到右
12	逻辑或运算	||	左到右
13	三目条件运算	? :	左到右
14	赋值运算	=	右到左

3．2　语句

1 语句概述

Java 里的语句可分为以下五类

1　方法调用语句,如

```
System.out.println(" Hello")
```

2　表达式语句　由一个表达式构成一个语句,最典型的是赋值语句.

```
x=23
```

一个表达式的最后加上一个分号就构成了一个语句.分号是语句不可缺少的部分.

3　复合语句　可以用{　}把一些语句括起来构成复合语句.

```
{
    z=23+x;
    System.out.println("hello");
}
```

4　控制语句.

4　package 语句和 import 语句.

package 语句和 import 语句和类,对象有关,将在第 4 章讲解.

2　Java 语言的控制语句

Java 语言的控制语句有 2 种类型,即条件语句,和 switch 开关语句.

1　条件控制语句

条件语句可分为 3 种.

a　if 语句

if 语句的一般形式

```
if(表达式)
  {
    若干语句
  }
```

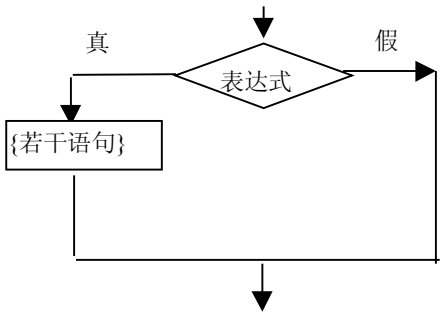


图 3.3　if 条件语句

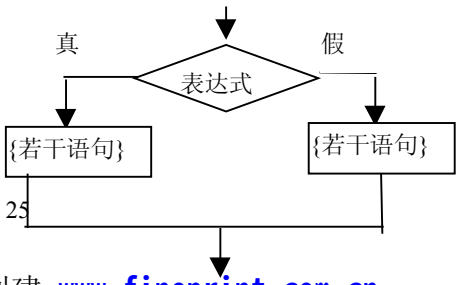
if 后面　　内的表达式的值必须是 boolean 类型,当值为 true 时,则执行紧跟着的复合语句　如果表达式的值为 false,则执行程序中 if 语句后面的其他语句.复合语句如果只有一个语句,{ }可以省略不写,但为了增强程序的可读性最好不要省略.在下面的例子中,我们将变量 a,b,c 内存中的数值按大小顺序进行互换.

例子 3

```
public class Example3_3
{  public static void main(String args[])
    {  int a=9,b=5,c=7,t;
       if(a>b)
       { t=a; a=b; b=t;
       }
      if(a>c)
      { t=a; a=c; c=t;
      }
      if(b>c)
      { t=b; b=c; c=t;
      }
     System.out.println("a="+a+",b="+b+",c="+c);
    }
}
```

b　if-else 语句.

if-else 语句的一般形式



```
if(表达式)
 {  若干语句
 }
else
 { 若干语句
 }
```

if 后面( )内的表达式的值必须是 **boolean** 型的.如果表达式的值为 **true**,则执行紧跟着的复合语句　如果表达式的值为 **false**,则执行 **else** 后面的复合语句.复合语句是由{ }括起来的若干个语句,如图 **3.4** 所示.

下列是有语法错误的 **if-else** 语句,

```
if(x>0)
  y=10;
  z=20;
else
  y=-100;
```

正确的写法是

```
if(x>0)
  {y=10;
   z=20;
  }
else
  y=100;
```

注　if 和 else 后面的复合句里如果只有一个语句,{ }可以省略不写,但为了增强程序的可读性最好不要省略.

有时为了编程的需要,else 或 if 后面的大括号里可以没有语句.

例子 3

```
public class Example3_4
{  public static void main(String args[])
    {int math=65 ,english=85;
     if(math>60)
        { System.out.println("数学及格了");
        }
     else
        { System.out.println("数学不及格");
        }
     if(english>90)
```

```
        { System.out.println("英语是优");
        }
      else
        {  System.out.println("英语不是优");
        }
      if(math>60&&english>90)
        {  System.out.println("英语是优,数学也及格了");
        }
      System.out.println("我在学习控制语句");
    }
}
```

c　if 语句的扩充形式

```
if(表达式 1)
  语句 1
else if(表达式 2)
 语句 2
.....
else if(表达式 n)
语句 n
```

其功能如图 3.5 所示.

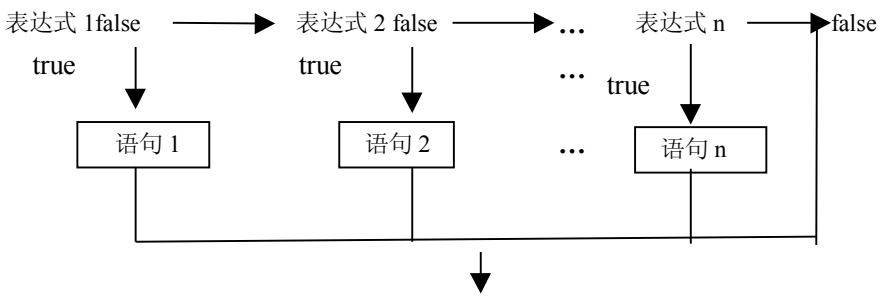


图 3.5　if 语句的扩充形式

2　switch 开关语句

switch　语句是多分支的开关语句,它的一般格式定义如下

```
switch(表达式)
  {
case  常量值 1
     若干个语句
     break
case  常量值 2
```

```
        若干个语句
        break
      …  …   …
case  常量值 n
        若干个语句
        break
default
      若干语句
}
```

switch 语句中表达式的值必须是整型或字符型　常量值 **1** 到常量值 **n** 必须也是整型或字符型.**switch** 语句首先计算表达式的值,如果表达式的值和某个 **case** 后面的常量值相同,就执行该 **case** 里的若干个语句直到碰到 **break** 语句为止.若没有一个常量与表达式的值相同,则执行 **default** 后面的若干个语句.其中 **default** 是可有可无的,如果它不存在,并且所有的常量值都和表达式的值不相同,那么 **switch** 语句就不会进行任何处理.

需要注意的是　在同一个 switch 语句中,case 后的常量值必须互不相同. 下面的例子 5 出现了 switch 语句.

例子 5

```
import java.applet.*;import java.awt.*;
public class Example3_5 extends Applet
{ public void paint(Graphics g)
    { int x=2,y=1;
      switch(x+y)
        {case 1 :
             g.setColor(Color.red);g.drawString("i am 1",5,10);
             break;
         case 2:
             g.setColor(Color.blue); g.drawString("i am 2",5,10);
             break;
         case 3:
            g.setColor(Color.green); g.drawString("i am 3",5,10);
            break;
         default:  g.drawString("没有般配的",5,10);
        }
    }
}
```

3. 循环语句

1　for 循环语句

for 语句是 java 程序设计中最有用的循环语句,**for** 语句的格式如下

```
for (表达式 1  表达式 2  表达式 3)
  { 若干语句
  }
```

- ✧ **for** 语句中的复合语句 {若干语句},称为循环体.
- ✧ 表达式 1 负责完成变量的初始化.
- ✧ 表达式式 2 是值为 boolean 型的表达式,称为循环条件.
- ✧ 表达式 3 用来修整变量,改变循环条件.

for 语句的执行过程是这样的 首先计算表达式 1,完成必要的初始化工作 再判断表达式 2 的值,若表达式 2 的值为 true,则执行循环体 执行完循环体之后紧接着计算表达式 3,以便改变循环条件,这样一轮循环就结束了.第二轮循环从计算表达式 2 开始,若表达式 2 的值仍为 true 则继续循环,否则跳出整个 **for** 语句执行后面的语句,如图 3.6 所示.

例子 6 求从 1 加到 100 的和

```
import java.applet.*;import java.awt.*;
public class Example3_6  extends Applet
{  public void paint(Graphics g)
    {  int sum=0;
       for(int i=1;i<=100;i++)
        { sum=sum+i;
        }
      g.drawString("sum= "+sum,10,20);
    }
}
```

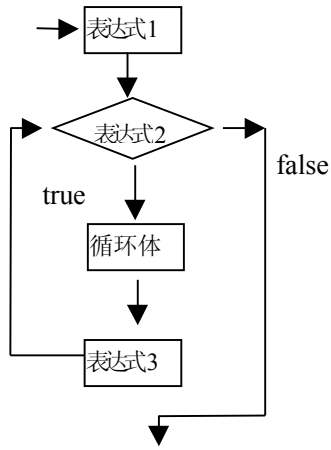


图 3.6 for 循环语句

例子 7 求 10 的阶乘

```
import java.applet.*;import java.awt.*;
public class Example3_7  extends Applet
{ public void paint(Graphics g)
    { long jiecheng=1;
      for(int i=10;i>=1;i--)
       { jiecheng=jiecheng*i;
       }
      g.drawString("10 的阶乘是 "+jiecheng,10,20);
    }
}
```

2 while 循环

一般格式

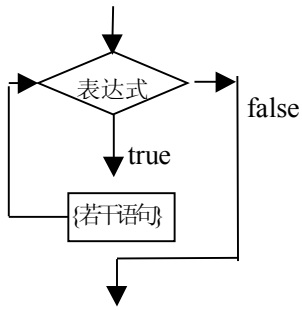


图 3.7 while 循环语句

```
while(表达式)
      { 若干语句
      }
```

3 do-while 循环.

一般格式

```
do
   { 若干语句
   }
while(表达式);
```

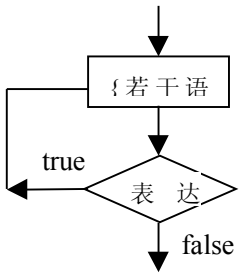


图 3.8 do-while 循环

当{}中只有一条语句时,大括号{}可以省略,但最好不要省略,以便增加程序的可读性.

注 do-while 循环和 while 循环的区别是 do-while 的循环体至少被执行一次,如图 3.7,3.8 所示.

下面的例子 8 用 while 语句计算 1+1/2!+1/3!+1/4! … 的前 20 项和.

例子 8

```
class Example3_8
{   public static void main(String args[])
    { double sum=0,a=1;int i=1;
      while(i<=20)
        { a=a*(1.0/i);
          sum=sum+a;
          i=i+1;
        }
      System.out.println("sum="+sum);
    }
}
```

4 在循环体中使用语句 break 和语句 continue

在一个循环中,比如循环 50 次的循环语句中,如果在某次循环体的执行中执行了 break 语句,那么整个循环语句就结束.如果在某次循环体的执行中执行了 continue 语句,那么本次循环就结束,即不再执行本次循环中循环体中 continue 语句后面的语句,而转入进行下一次循环.

例子 9

```
class Example3_9
{  public static void main(String args[])
   {  int sum=0,i,j;
```

```
        for( i=1;i<=10;i++)                    //计算 1+3+5+7+9.
         {  if(i%2==0)
              continue;
           sum=sum+i;
         }
        System.out.println("sum="+sum);
        for( j=2;j<=50;j++)                    //求 50 以内的素数
        { for( i=2;i<=j/2;i++)
            {if(j%i==0)
                 break;
            }
          if(i>j/2)
            {System.out.println(""+j+"是素数");
            }
         }
    }
}
```

习 题 三

1 分别编写一个应用程序和小应用程序求 1!+2!+…+20!.

2 编写一个小应用程序求 100 以内的全部素数.

3 分别用 do-while 和 for 循环计算 1+1/2!+1/3!+1/4!… … 的前 20 项和.

3 一个数如果恰好等于它的因子之和,这个数就称为"完数".分别编写一个应用程序和小应用程序求 1000 之内的所有完数.

第四章　　类,对象,和接口

4．1　编程语言的几个发展阶段

4.1.1.　机器语言　如汇编语言

在第一台计算机诞生之后,虽然它的计算速度并不比当时的一些计算工具有太大的优势,但人们注意到这种新的"计算工具"和传统的计算工具有了一个本质的区别,就是它能存储指令,并可以不断地重新执行这些指令.以往的传统计算工具,比如中国的"算盘",当人们按着一定的口诀　即指令　计算出结果后,如果想再计算一次,就必须重新在算盘上"拨弄口诀",因为算盘不能存储"口诀".电子计算机则不同,它是由电子元件组成的,这种电子元件有两种稳定的状态,可以用 0,1 来表示这两种状态,这样电子计算机就可以使用二进制数来存储和处理信息了.

计算机处理信息的早期语言是所谓的机器语言,这种语言中的指令都是由 0,1 组成的序列,称这样的序列为一条机器指令.比如,某种型号的的计算机用 8 位二进制信息 10001010 表示一次加法,以 0001 0011 表示一次减法等等.这些指令的执行由计算机的线路来保证,计算机在设计之初,事先就要确定好每一条指令对应的线路逻辑操作.用机器语言进行程序设计是一项累人的工作,同样的任务,人们要针对不同型号的计算机分别进行编写指令,因为一种型号的计算机用 10001010 表示加法操作,而另一种型号的计算机可能用 11110000 来表示加法操作.因此,使用机器语言编程也称做面向机器编程.20 世纪 50 年代出现了汇编语言,在编写指令时,它用一些简单的容易记忆的符号来代替二进制指令,但汇编语言仍是面向机器语言,需针对不同的机器来编写不同的代码.习惯上称机器语言,汇编语言是低级语言.

4.1.2. 过程语言　如 c 语言,Fortrans 语言等

随着计算机硬件功能的提高,在 20 世纪 60 年代出现了过程设计语言,如 C 语言,Fortans 语言等.用这些语言编程也称做面向过程编程,语言把代码组成叫做过程或函数的块.每个块的目标是完成某个任务,例如,一个 C 的源程序就是由若干个书写形式互相独立的函数组成.使用这些语言编写代码指令时,不必再去考虑机器指令的细节,人们只要按着具体语言的语法要求去编写"源文件",所谓源文件,就是根据这门语言的语法编写具有一定扩展名的文本文件,比如 C 语言编写的源文件的扩展名是.c,Fortans 语言编写的源文件的扩展名是.for 等等.过程语言的源文件的一个特点是更接近人的"自然语言",比如, C 语言源程序中的一个函数

```
int max(int a,int b)
 {if(a>b)
   return a;
```

```
    else
     return b
  }
```

该函数负责计算两个整数的最大值.使用面向过程语言,人们只需按着自己的意图来编写各个函数,语言的语法更接近人们的自然语言,所以,习惯上也称过程语言是高级语言.但是,无论那种高级语言编写的源文件,计算机都不能直接执行,因为计算机只能直接识别,执行机器指令.因此,必须把源文件转换成机器指令,然后计算机去执行相应的机器指令.

将高级语言编写的源程序转化成机器指令,经常使用下列两种方式.

1 编译方式 Complilation 针对当前的机器处理器芯片,将源程序全部翻译成机器指令,称做目标程序,再将目标程序交给计算机执行.

2 解释方式 Interpretation :这种方式不产生整个的目标程序,而是根据当前的机器处理器芯片,边翻译边执行,翻译一句执行一句.

无论那种高级语言都必须提供相应的编译器或解释器,例如,C 语言采用的是上述的编译方式,即针对特定的 CPU 芯片对源文件进行编译,生成机器代码.采用上述 1 的编译方式生成的目标代码就和特定的 CUP 芯片有关,一旦环境有所变化就需要对软件作一番改动,耗时费力.

Java 语言的处理方式既不是第 1 种也不是第 2 种,Java 语言的创造发明之处在于,它不针对特定的 CPU 芯片进行编译,Java 提供的编译器把源程序编译成称做字节码的一个中间代码.字节码是很接近机器码的文件,可以在提供了 Java 虚拟机 JVM 的任何系统上被解释执行.因此,Java 本质上是解释执行的程序,当字节码加载到内存之后,再由 Java 的解释器对字节码按上述 2 的解释方式执行,即翻译一句,执行一句,不产生整个的机器代码程序,翻译过程如果不出现错误,就一直进行到全部执行完毕,否则将在错误处停止执行.同一个程序,如果是解释执行的,那么它的运行速度通常会比被编译成可执行的机器代码的运行速度慢一些.但是,对 Java 来说,二者的差别不太大,Java 的字节码经过仔细设计,很容易便能使用 JIT 即时编译方式 编译技术将字节码直接转化成高性能的本地机器码,Sun 公司在 Java 2 发行版中提供了这样一个字节码编译器---JIT(Just In Time),它是 Java 虚拟机的一部分.Java 运行系统在提供 JIT 的同时仍具有平台独立性,因而"高效且跨平台"对 java 来说不再矛盾.

如果把 Java 的源文件比做"汉语"的话,字节码就相当于"世界语",世界语不和具体的"国家"有关,只要这个"国家"提供了"翻译",就可以再快速地把世界语翻译成本地语言.

4.1.3. 面向对象编程

随着计算机硬件设备功能的进一步提高,使得基于对象的编程成为可能.基于对象的编程更加符合人的思维模式,编写的程序更加健壮和强大,更重要的是,面向对象编程鼓励创造性的程序设计.

在实际生活中,我们每时每刻都与"对象"在打交道.我们用的钢笔,骑的自行车,乘的公共汽车等.而我们经常见到的卡车,公共汽车,轿车等都会涉及以下几个重要的物理量

可乘载的人数,运行速度,发动机的功率,耗油量,自重,轮子数目等.

另外,还有几个重要的功能

加速功能,减速功能,刹车,转弯功能等.

我们也可以把这些功能称作是他们具有的方法,而物理量是它们的状态描述.仅仅用物理量或功能不能很好的描述它们.在现实生活中,我们用这些共有的属性和功能给出一个概念机动车类.一个具体的轿车就是机动车类的一个实例 对象 .

Java 语言与其它面向对象语言一样,引入了类的概念,类是用来创建对象的模板,它包含被创建的对象的状态描述和方法的定义. **Java** 是面向对象语言,它的源程序是由若干个类组成,源文件是扩展名为.java 的文本文件.

因此,要学习 **Java** 编程就必须学会怎样去写类,即怎样用 **Java** 的语法去描述一类事物共有的属性和功能.属性通过变量来刻画,功能通过方法来体现,即方法操作属性形成一定的算法来实现一个具体的功能.类把数据和对数据的操作封装成一个整体.

4．2 类

类是组成 Java 程序的基本要素. 类封装了一类对象的状态和方法. 类是用来定义对象的模板.

类的实现包括两部分:类声明和类体. 基本格式为

```
class 类名
 { ...
   类体的内容.....
 }
```

class 是关键字,用来定义类. "class 类名"是类的声明部分,类名必须是合法的 Java 标识符. 两个大括号以及之间的内容是类体.

4.2.1. 类声明

以下是两个类声明的例子.

```
class People
 { ...
 }
class 植物
 {...
 }
```

"class People"和"class 植物"叫做类声明 "People"和"植物"分别是类名习惯上类名的第一个字母大写,但这不是必须的. 类的名字不能是 Java 中的关键字,要符合标识符规定,即名字可以由字母,下划线,数字或美圆符号组成,并且第一个字符不能是数字. 但给类命名时,最好遵守下列习惯

1 如果类名使用拉丁字母,那么名字的首写字母使用大写字母,如

```
Hello,Time,People
```

等.

2 类名最好见名得意,当类名由几个"单词"复合而成时,每个单词的首写字母使用大写,如

BeijingTime,AmericanGame,HelloChina

等.

4.2.2. 类体

写类的目的是为了描述一类事物共有的属性和功能,描述过程由类体来实现.类声明之后的一对大括号"{","}"以及它们之间的内容称做类体,大括号之间的内容称做类体的内容.

类体的内容由两部分构 一部分是变量的定义,用来刻画属性 另一部分是方法的定义,用来刻画功能.

下面是一个类名为"梯形"的类,类体内容的变量定义部分定义了 4 个 float 类型的变量 "上底","下底","高"和"laderArea" 方法定义部分定义了两个方法 "计算面积"和"修改高".

```
class 梯形
{
  float 上底,下底,高,laderArea;          //变量定义部分.
  float 计算面积()
        {     //方法定义
          laderArea=(上底+下底)*高/2.0f;
          return laderArea;
        }
  void 修改高(float h)
        {  //方法定义
          高=h;
        }
}
```

4.2.3. 成员变量和局部变量

我们已经知道类体分为两部分.变量定义部分所定义的变量被称为类的成员变量.在方法体中定义的变量和方法的参数被称为局部变量.

1 成员变量和局部变量的类型可以是 Java 中的任何一种数据类型,包括基本类型整型,浮点型,字符型 引用类型 数组类型,对象.对象也称作类类型变量.

```
class People
  {int boy;
    float a[];
```

```
  ...void f(){
    boolean cool;  Workman zhangboy; ...
     ... ...
    }
 }
class Workman
 {  double x; People zhiwei;
   ......
 }
```

People 类的成员变量 a 是浮点数组型变量,cool 是局部变量,zhangboy 是类类型 对象 局部变量,类 Workman 中的 zhiwei 是类类型变量,即对象.

2 成员变量在整个类内都有效,局部变量只在定义它的方法内有效.

```
 class  Sun
  {  int distance;
     int  find()
     { int a=12;;
       distance=a;        //合法,distance 在整个类内有效.
       return distance;
      }
     void g()
      { int y;
        y=a;             //非法,因为 a 以失效,而方法 g 内又没有定义变量 a
      }
   }
```

成员变量与它在类体中书写的先后位置无关,例如,前述的梯形类也可以写成

```
class 梯形
{ float 上底,laderArea;   //成员变量的定义.
  float 计算面积()
   { laderArea=(上底+下底)*高/2.0f;
     return laderArea;
    }
  float 下底;              //成员变量的定义.
  void 修改高(float h)     //方法定义.
    {   高=h;
    }
float  高;                 //成员变量的定义.
}
```

但不提倡把成员变量的定义分散地写在方法之间或类体的最后,人们习惯先介绍属性再介绍功能.

3 成员变量又分为实例成员变量 简称实例变量 和类成员变量 简称类变量 例如

```
class dog
{ float x;
  static int y;
   … …
}
```

x 是实例变量,而 y 是类变量.如果成员变量的类型前面加上关键字 static,这样的成员变量称做是类变量或静态成员变量.再学习过对象之后,就会知道实例变量和类变量的区别.

4 如果局部变量的名字与成员变量的名字相同,则成员变量被隐藏,即这个成员变量在这个方法内暂时失效.

```
class  Tom{
  int x=98,y;
  void f()
  {
   int x=3;
   y=x;//y 得到的值是 3,不是 98.如果方法 f 中没有"int x=3;"语句,y 的值将是 98.
  }
}
```

5 我们已经知道,如果局部变量的名字与成员变量的名字相同,则成员变量被隐藏.这时如果想在该方法内使用成员变量,必须使用关键字 this.

```
class 三角形
{  float sideA,sideB,sideC,lengthSum;
   void setSide(float sideA,float sideB,float sideC)
    { this.sideA=sideA;  this.sideB=sideB; this.sideC=sideC;
    }
}
```

this.sideA, this. sideB, this. sideC 就分别表示成员变量 sideA, sideB, sideC.

4.2.4. 方法

我们已经知道一个类的类体由两部分组成 变量的定义和方法的定义.方法的定义包括两部分 方法声明和方法体.一般格式为

方法声明部分

{

方法体的内容

}

1 方法声明.

最基本的方法声明包括方法名和方法的返回类型,如

float area()

{....

}

方法返回的数据的类型可以是任意的 Java 数据类型,当一个方法不需要返回数据时,返回类型必须是 void.很多的方法声明中都给出方法的参数,参数是用逗号隔开的一些变量声明.方法的参数可以是任意的 Java 数据类型.

方法的名字必须符合标识符规定.在给方法起名字时应遵守良好的习惯 名字如果使用拉丁字母,首写字母使用小写.如果由多个单词组成,从第 2 个单词开始的其它单词的首写字母使用大写.例如

```
float getTrangleArea()
void setCircleRadius(double radius)
```

等.

下面的 **Trangle** 类中有个 5 个方法.

```
class Trangle
{  double sizdA,siddB,siddC;
   void setSide(double a,double b,double c)
     { sideA=a;
       sideB=b;
       sideC=c;
      }
   double getSideA()
     { return sideA;
     }
   double getSideB()
     { return sideB;
     }
   double getSideC()
    { return sideC;
    }
boolean isOrNotTrangle()
   {  if(sideA+sideB>sideC&&sideA+sideC>sideB&&sideB+sideC>sideA)
```

```
        { return true;
        }
      else{ return flase;
          }
    }
}
```

(2)方法体.

方法声明之后的一对大括号"{" ,"}"以及之间的内容称做方法的方法体.方法体的内容包括局部变量的定义和合法的 Java 语句,如

```
int getPrimNumberSum(int n)
{  int sum=0;
   for(int i=1;i<=n;i++)
     { int j;
       for(j=2;j<i;j++)
         { if(i%j==0)
                break;
         }
       if(j>=i)
         {  sum=sum+i;
         }
     }
   return  sum;
}
```

方法参数在整个方法内有效,方法内定义的局部变量从它定义的位置之后开始有效.写一个方法和 C 语言中写一个函数完全类似,只不过在这里称做方法罢了.局部变量的名字必须符合标识符规定,遵守习惯　名字如果使用拉丁字母,首写字母使用小写.如果由多个单词组成,从第 2 个单词开始的其它单词的首写字母使用大写.

4.2.5.　方法重载

方法重载的意思是　一个类中可以有多个方法具有相同的名字,但这些方法的参数必须不同,即或者是参数的个数不同,或者是参数的类型不同.下面的 Area 类中 getArea 方法是一个重载方法.

```
class Area
{   float getArea(float r)
    { return 3.14f*r*r;
    }
    double getArea(float x,int y)
```

```
    {  return x*y
    }
    float getArea(int x,float y);
    {   return x*y;
    }
    double getArea(float x,float y,float z)
    {   return (x*x+y*y+z*z)*2.0;
    }
}
```

注:方法的返回类型和参数的名字不参与比较,也就是说如果两个方法的名字相同,即使类型不同,也必须保证参数不同.

4.2.6. 构造方法

构造方法是一种特殊方法,它的名字必须与它所在的类的名字完全相同,并且不返回任何数据类型,即它是 void 型 void 可以省略不写 .例如

```
class 梯形
{   float 上底,下底,高;
   梯形()
   { 上底=60;
     下底=100;
     高=20;
   }
  梯形(float x,int y,float h)
   { 上底=x;
     下底=y;
      高=h;
   }
}
```

注: 当用类创建对象时,使用构造方法,见 4.3.

4.2.7. 类方法和实例方法

我们已经知道,成员变量可分为实例变量和类变量.同样,类中的方法也可分为实例方法和类方法,如

```
class A
{  int a;
   float max(float x,float y)
```

```
    { … …
    }
     static float jerry()
     {… …
    }
    static void String speak(String s)
    {… …
    }
}
```

类 A 中的方法 jerry 和 speak 是类方法,max 是实例方法,即方法声明时,方法类型前面不加关键字 static 的是实例方法,加 static 的是类方法.注意 static 需放在方法的类型的前面.

4.2.8. 两个值得注意的问题

1 对成员变量的操作只能放在方法中,方法可以对成员变量和方法体中自己定义的局部变量进行操作.在定义类的成员变量时可以同时赋予初值,如

```
class A
{  int a=12;
  float b=12.56f;
}
```

但是不可以这样做

```
class A
{ int a;
  float b;
  a=12;    //非法
  b=12.56f; //非法
}
```

因为类体的内容由成员变量的定义和方法的定义两部分组成.如

```
class A
{ int a;
  float b;
  void f()
  { int x,y;
    x=34;  y=-23;   a=12; b=12.56f;
  }
}
```

2　但需要注意的是　实例方法既能对类变量操作也能对实例变量操作,而类方法只能对类变量进行操作.如

```
class A
{ int a; static int b;
  void f(int x,int y)
     {  a=x;  //合法.
        b=y;  // 合法.
     }
  static void g(int z)
   {   b=23; // 合法.
       a=z;  //非法.
   }
}
```

3　一个类中的方法可以互相调用,实例方法可以调用该类中的其它方法　类中的类方法只能调用该类的类方法,不能调用实例方法.如

```
class A
{  float a,b;
   void sum(float x,float y)
   {  a=max(x,y);
      b=min(x,y);
   }
  static  float  getMaxSqrt(float x,float y)
  {  float c;
     c=max(x,y)*max(x,y);
     return c;
  }
  static  float max(float x,float y)
  {   if(x<=y)
        { return y;
        }
     else
        { return x;
        }
  }
  float min(float x,float y)
  {  if(x<=y)
       { return x;
       }
```

```
        else
          { return y;
          }
    }
}
```

4．3　对象

我们已经说过类是创建对象的模板.当使用一个类创建了一个对象时,我们也说我们给出了这个类的一个实例.

4.3.1.　创建对象

创建一个对象包括对象的声明和为对象分配内存两个步骤.

1　对象的声明.

一般格式为

类的名字　对象名字;

如

People zhangPing

这里 People 是一个类的名字,zhangPing 是我们声明的对象的名字.

2　为声明的对象分配内存.

使用 new 运算符和类的构造方法为声明的对象分配内存,如果类中没有构造方法,系统会调用默认的构造方法　默认的构造方法是无参数的,你一定还记得构造方法的名字必须和类名相同这一规定　.如

```
zhangPing=new People();
```

以下是两个详细的例子.

例子 1

```
class XiyoujiRenwu
{   float height,weight;
    String head, ear,hand,foot, mouth;
    void speak(String s)
    {  System.out.println(s);
    }
}
class A
{   public static void main(String args[])
```

```
    { XiyoujiRenwu  zhubajie;        //声明对象.
      zhubajie=new  XiyoujiRenwu(); //为对象分配内存,使用 new 运算符和默认的构造方法.
      … …. ….
     }
}
```

例子 2

```
class Point
{ int x,y;
  Point(int a,int b)
  { x=a;
    y=b;
  }
}
public class A
{ public static void main(String args[])
  { Point p1,p2;                  //声明对象 p1 和 p2.
    p1=new Point(10,10);          //为对象分配内存,使用 new 和类中的构造方法.
    p2=new Point(23,35);          //为对象分配内存,使用 new 和类中的构造方法.
     … …   …
 }
}
```

注　如果你的类里定义了一个或多个构造方法,那么 Java 不提供默认的构造方法.上述例子 2 提供了构造方法,下列创建对象是非法的

```
p1=new Point();
```

3　对象的内存模型

我们使用前面的例子 1 来说明对象的内存模型.

1　声明对象时的内存模型.

当用 XiyoujiRenwu 类声明一个变量 zhubajie,即对象 zhubajie 时,如例子 1 中

```
XiyoujiRenwu  zhubajie;
```

图 4.1　未分配实体的对象

内存模型如图 4.1 所示.

声明对象变量 zhubajie 后,zhubajie 的内存中还没有任何数据,我们称这时的 zhubajie 是一个空对象,空对象不能使用,因为它还没有得到任何"实体".必须再进行为对象分配内存的步骤,即为对象分配实体.

2　对象分配内存后的内存模型

当系统见到

```
zhubajie=new XiyoujiRenwu();
```

时,就会做两件事

为 height,weight,head,ear,mouth,hand,foot 各个变量分配内存. 即 XiyoujiRenwu 类的成员变量被分配内存空间.如果成员变量在声明时没有指定初值,那么,对于整型变量,默认初值是 0　对于浮点型,默认初值是 0.0　对于 booean 型,默认初值是 false　对于引用型,默认初值是 null.

给出一个信息,已确保这些变量是属于对象 zhubajie 的,即这些内存单元将由 zhubajie 操作管理. 为了作到这一点,new 运算符在为变量 height,weight,head,ear,mouth,hand,foot 分配内存后,将返回一个引用给对象变量 zhubajie.也就是返回一个"号码"　地址号,即代表这些成员变量内存位置的首地址号码给 zhubajie,你不妨就认为这个引用就是 zhubajie 在内存里的名字,而且这个名字　引用是 Java 系统确保分配给 height,weight,head,ear,mouth,hand,foot 的内存单元将由 zhubajie 操作管理.称 height,weight,head,ear,mouth,hand,foot 分配的内存单元是属于对象 zhubajie 的实体,即这些变量是属于 zhubajie 的.所谓为对象分配内存就是指为它分配变量,并获得一个引用,以确保这些变量由它来"操作管理".为对象分配内存后,内存模型由声明对象时的模型　图 4.1,变成如图 4.2 所示意.

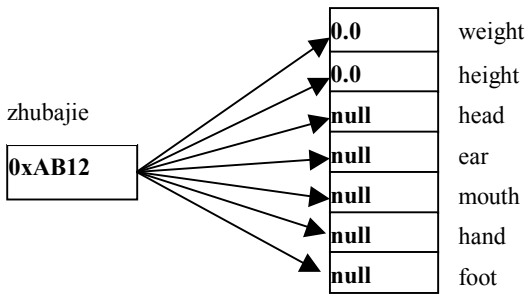


图 4.2　分配实体后的对象

如果你的想象力丰富的话,那么你可以想象在这个计算机内存中的"现实世界"里就有了一个"活生生"的 zhubajie. 这个 zhubajie 就可以对自己的 height,weight,head,ear,mouth,hand,foot 进行操作,改变自己的状态,高兴的话就改变成一个浓眉大眼的壮汉,到高老庄去.

对象的声明和分配内存两个步骤可以用一个等价的步骤完成,如

```
XiyoujiRenwu zhubajie=new XiyoujiRenwu();
```

3　创建多个不同的对象

一个类通过使用 new 运算符可以创建多个不同的对象,这些对象将被分配不同的内存空间,因此,改变其中一个对象的状态不会影响其它对象的状态.例如,我们可以在上述例子 1

中创建两个对象 zhubajie,sunwukong.

```
zhubajie=new XiyoujiRenwu();
sunwukong =new XiyoujiRenwu();
```

当创建对象 zhubajie 时,XiyoujiRenwu 类中的成员变量 height,weight,head,ear,mouth,hand,foot 被分配内存空间,并返回一个引用给 zhubajie 当再创建一个对象 sunwukong 时,XiyoujiRenwu 类中的成员变量 height,weight,head,ear,mouth,hand,foot 再一次被分配内存空间,并返回一个引用给 sunwukong. sunwukong 的变量所占据的内存空间和 zhubajie 的变量所占据的内存空间是互不相同的位置. 内存模型如下图 4.3 所示

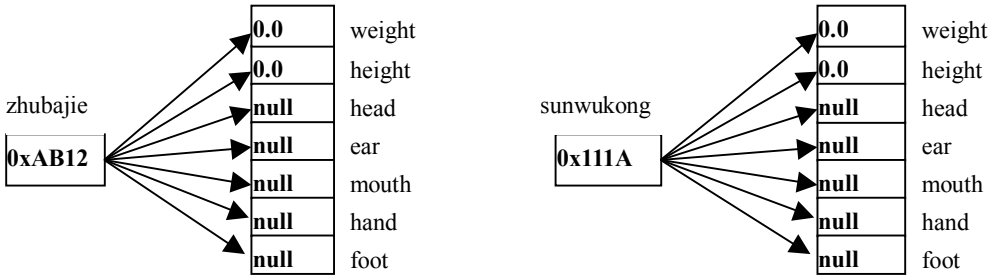


图 4.3 创建多个对象

4.3.2. 使用对象

对象不仅可以操作自己的变量改变状态,而且还拥有了使用创建它的那个类中的方法的能力,对象通过使用这些方法可以产生一定的行为.

通过使用运算符". ",对象可以实现对自己的变量访问和方法的调用.

1 对象操作自己的变量 对象的属性 .

对象创建之后,就有了自己的变量,即对象的实体. 通过使用运算符". ",对象可以实现对自己的变量的访问.

2 对象调用类中的方法 对象的功能

对象创建之后,可以使用运算符". "调用创建它的类中的方法,从而产生一定的行为功能. 当对象调用类中的一个方法时,方法中的局部变量被分配内存空间. 方法执行完毕,局部变量即刻释放内存.

在学习类的时候我们讲过 类中的方法可以操作成员变量. 当对象调用方法时,方法中出现的成员变量就是指该对象的成员变量

下面的例子 3 中,有两个对象 zhubajie,sunwukong.

例子 3 效果如图 4.5

```
class XiyoujiRenwu
```

```
F:\8000>java Example4_3
zhubajie的身高：1.8
zhubajie的头：大头
sunwukong的重量：1000.0
sunwukong的头：绣发飘飘
俺老猪我想娶媳妇
zhubajie现在的头：歪着头
老孙我重1000斤，我想骗八戒背我
sunwukong现在的头：歪着头
```

```
 { float height,weight;
   String head, ear,hand,foot, mouth;
   void speak(String s)
   { head="歪着头";
     System.out.println(s);
   }
}
class Example4_3
{  public static void main(String args[])
   { XiyoujiRenwu  zhubajie,sunwukong;//声明对象.
     zhubajie=new  XiyoujiRenwu();  //为对象分配内存,使用 new 运算符和默认的构造方法.
     sunwukong=new  XiyoujiRenwu();
     zhubajie.height=1.80f;      //对象给自己的变量赋值.
     zhubajie.weight=160f;       zhubajie.hand="两只黑手";
     zhubajie.foot="两只大脚";   zhubajie.head="大头";
     zhubajie.ear="一双大耳朵";  zhubajie.mouth="一只大嘴";
     sunwukong.height=1.62f;     //对象给自己的变量赋值.
     sunwukong.weight=1000f;     sunwukong.hand="白嫩小手";
     sunwukong.foot="两只绣脚";  sunwukong.head="绣发飘飘";
     sunwukong.ear="一对小耳";   sunwukong.mouth="樱头小嘴";
     System.out.println("zhubajie 的身高  "+zhubajie.height);
     System.out.println("zhubajie 的头  "+zhubajie.head);
     System.out.println("sunwukong 的重量  "+sunwukong.weight);
     System.out.println("sunwukong 的头  "+sunwukong.head);
     zhubajie.speak("俺老猪我想娶媳妇");       //对象调用方法.
     System.out.println("zhubajie 现在的头  "+zhubajie.head);
    sunwukong.speak("老孙我重 1000 斤,我想骗八戒背我"); //对象调用方法.
    System.out.println("sunwukong 现在的头  "+sunwukong.head);
   }
}
```

我们知道 类中的方法可以操作成员变量,当对象调用该方法时,方法中出现的成员变量就是指该对象的成员变量.在上述例子中,当对象 zhubajie 调用过方法 speak 之后,就将自己的头修改成 "外着头".同样,对象 sunwukong 调用过方法 speak 之后,也就将自己的头修改成 "外着头".

下面的例子 4 中,有两个对象 wanghong,lihong.

例子 4 效果如图 4.6

```
import java.applet.*;
import java.awt.*;
class Student
{  float math,english,sum;
   float f(float k1,float k2)
   { sum=k1*math+k2*english;
     return sum;
   }
}
public class Example4_4 extends Applet
{  Student wanghong,lihong;
   public void init()
   { wanghong=new Student();  lihong  =new Student();
     wanghong.math=60.0f;     wanghong.english=80f;
     lihong.math=70.0f;       lihong.english=90.0f;
     wanghong.sum=wanghong.f(2.0f,2.0f);
     lihong.sum=lihong.f(2.0f,2.0f);
   }
   public void paint(Graphics g)
   { g.drawString("lihong sum= "+lihong.sum,12,45);
     g.drawString("wanghong sum="+wanghong.sum,12,60);
   }
}
```

在上述例子 4 中,当 lihong 调用方法 f 时,将值 2.0f 和 2.0 传递给方法的参数 x,y.方法 f 将计算 2.0*math+2.0*english,这个计算式子中的变量 math,english 当然是指被 liong 操纵的 math,english,因此这个式子中的 math 的值是 70f,english 的值是 90f.

在下面的例子 5 中,我们用梯形类创建对象,该对象有计算自身面积的功能.

例子 5

```
class 梯形
{  float 上底,下底,高;
    梯形()
    { 上底=60;
     下底=40;
      高=20;
    }
    梯形(float x,float y,float h)
    { 上底=x;
     下底=y;
       高=h;
```

```
    }
    float 计算面积()
    { float 面积;
      面积=(上底+下底)*高/2.0f;
      return 面积;
    }
    void 修改高(float height)
    {  高=height;
    }
    float 获取高()
    { return 高;
    }
}
class Example4_5
{    public static void main(String args[])
    { 梯形 laderOne=new 梯形(),laderTwo=new 梯形(2.0f,3.0f,10);
      System.out.println("laderOne的高是:"+laderOne.获取高());
      System.out.println("laderTwo的高是:"+laderTwo.获取高());
      System.out.println("laderOne的面积是:"+laderOne.计算面积());
      System.out.println("laderTwo的面积是:"+laderTwo.计算面积());
      laderOne.修改高(10);
      float h=laderOne.获取高();
      laderTwo.修改高(h*2);
      System.out.println("laderOne现在的高是:"+laderOne.获取高());
      System.out.println("laderTwo现在的高是:"+laderTwo.获取高());
      System.out.println("laderOne现在的面积是:"+laderOne.计算面积());
      System.out.println("laderTwo现在的面积是:"+laderTwo.计算面积());
    }
}
```

4.3.3. 对象的引用和实体

我们已经知道,当用类创建一个对象时,类中的成员变量被分配内存空间,这些内存空间称做该对象的实体,而对象中存放着引用,以确保实体由该对象操作使用.

再以例 2 中的 Point 类为例,假如我们分别使用类的构造方法 Point(int x,int y)创建了两个对象 p1,p2.

```
Point p1=new Point(12,16)  Point p2=new Point(6,18)
```

那么内存模型如图 4.7 所示.

假如你在程序中使用了下述赋值语句

```
p1=p2;
```

把 p2 的引用 p2 在内存中的名字 赋给了 p1,因此 p1 和 p2 本质上是一样的了.虽然在你的源程序中 p1,p2 是两个名字,但在系统看来他们的名字是一个 0xDD12.系统将取消原来分配给 p1 的内存.这时如果你输出 p1.x 的结果将是 6,而不是 12.即 p1,p2 有相同的实体.

内存模式变成如图 4.8 所示.

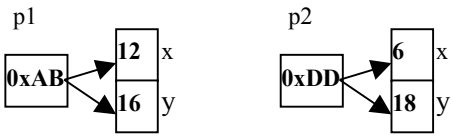


图 4.7 对象内存

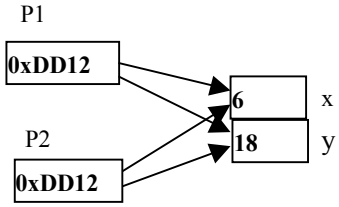


图 4.8 对象内存模式

因此,一个类创建的两个对象,如果具有相同的引用,那么就具有完全相同的实体.

注 实际上,Java 有所谓"垃圾收集"机制,这种机制周期地检测某个实体是否已不再被任何对象所拥有,如果发现这样的实体,就释放实体占有的内存.因此,Java 编程人员不必象 C++程序员那样,要时刻自己检查哪些对象应该释放内存.在上述程序中,当将 p2 的引用赋给 p1 后,最初分配给对象 p1 的变量 实体 所占有的内存就会被释放.

在下面的例子 6 中,"圆锥"类在创建对象时,将一个圆的对象的引用传递给圆锥对象的底圆.

例子 6 效果如图 4.9

```
class 圆
{  double 半径;
   圆(double r)
   { 半径=r;
   }
  double 计算面积()
  {  return 3.14*半径*半径;
  }
  void 修改半径(double 新半径)
  {  半径=新半径;
  }
  double 获取半径()
  { return 半径;
  }
}
```

```
F:\8000>java Example4_6
圆锥底圆半径:10.0
圆锥的体积:2093.3333333333335
圆锥底圆半径:100.0
圆锥的体积:209333.33333333334
```

图 4.9 圆堆对象

```
class 圆锥
{  圆 底圆;
  double 高;
  圆锥(圆 circle,double h)
  {  this.底圆=circle;
    this.高=h;
  }
  double 计算体积()
  {  double volume;
    volume=底圆.计算面积()*高/3.0;
    return  volume;
  }
  void 修改底圆半径(double r)
  {  底圆.修改半径(r);
  }
  double 获取底圆半径()
  {  return 底圆.获取半径();
  }
}
class Example4_6
{  public static void main(String args[])
   { 圆 circle=new 圆(10);
     圆锥 circular=new 圆锥(circle,20);
     System.out.println("圆锥底圆半径:"+circular.获取底圆半径());
     System.out.println("圆锥的体积:"+circular.计算体积());
     circular.修改底圆半径(100);
     System.out.println("圆锥底圆半径:"+circular.获取底圆半径());
     System.out.println("圆锥的体积:"+circular.计算体积());
   }
}
```

4.4 static 关键字

4.4.1. 实例变量和类变量的区别

在讲述类的时候我们讲过 类体的定义包括成员变量的定义和方法的定义,并且成员变量又分为实例变量和类变量,用 static 修饰的变量是类变量.那么类变量和实例变量有什么区别呢

我们已经知道 一个类通过使用 new 运算符可以创建多个不同的对象,这些对象将被分配不同的内存空间,现在再说得准确些就是 不同的对象的实例变量将被分配不同的内存空

间,如果类中的成员变量有类变量,那么所有的对象的这个类变量都分配给相同的一处内存,改变其中一个对象的这个类变量会影响其它对象的这个类变量.也就是说对象共享类变量.如

```
class  Takecare
{   static int x;
   int y;
... ....
 }
class  A
{  Takecare A1=new Takecare();
   Takecare A2=new Takecare();
}
```

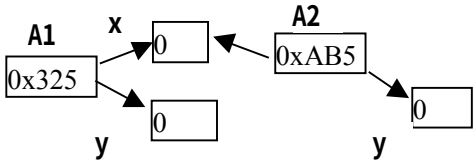


图 **4.10**　内存模式

内存模式如图 4.10 所示.如果程序中使用了

```
A1.x=18
```

这时如果输出 A1.x,A2.x 的值,结果相同都是 18.

下面例子 7 中的梯形对象共享一个底.

例子 7　效果如图 4.11

```
class 梯形
{    float 上底,高;
    static float 下底;
    梯形(float x,float y,float h)
    { 上底=x; 下底=y; 高=h;
    }
    float 获取下底()
    {   return 下底;
    }
    void 修改下底(float b)
    {  下底=b;
    }
}
class Example4_7
{    public static void main(String args[])
     { 梯形 laderOne=new 梯形(3.0f,10.0f,20),laderTwo=new 梯形(2.0f,3.0f,10);
       System.out.println("laderOne 的下底:"+laderOne.获取下底());
       System.out.println("laderTwo 的下底:"+laderTwo.获取下底());
       laderTwo.修改下底(60);
       System.out.println("laderOne 的下底:"+laderOne.获取下底());
```

```
F:\8000>java Example4_7
laderOne的下底:3.0
laderTwo的下底:3.0
laderOne的下底:60.0
laderTwo的下底:60.0
```

图 **4.11**　共享一个底的两个梯形

```
        System.out.println("laderTwo的下底:"+laderTwo.获取下底());
    }
}
```

我们知道,当 Java 程序执行时,类的字节码文件被加载到内存,如果该类没有创建对象,类的实例成员变量不会被分配内存.但是,类中的类变量,在该类被加载到内存时,就分配了相应的内存空间.如果该类创建对象,那么不同对象的实例变量互不相同,即分配不同的内存空间,而类变量不再重新分配内存,所有的对象共享类变量,即所有的对象的类变量是相同的一处内存空间,类变量的内存空间直到程序退出运行,才释放所占有的内存.

4.4.2. 通过类名直接访问类变量

类变量在类的字节码加载到内存时就分配了内存空间,并且被所有的对象共享,因此,Java 语言允许通过类名直接访问类变量.

下面例子 8,通过类名直接访问 static 成员变量.

例子 8

```
import java.applet.*;
import java.awt.*;
class Family
{  static String familyname;
       String name;
  int age;
}
public class Example4_8 extends Applet
{  Family father,son1,son2;
   public void init()
   { father=new Family();
     son1=new Family();son2=new Family();
     Family.familyname="打"; father.name="鬼子";
     son1.name="汉奸"; son2.name="恶霸";
   }
   public void paint(Graphics g)
   {  g.drawString("father: "+father.familyname+father.name,5,10);
      g.drawString("son1: "+son1.familyname+son1.name,5,20);
      g.drawString("son2: "+son2.familyname+son2.name,5,30);
      Family.familyname="杀";
      g.drawString("father:"+father.familyname+father.name,5,40);
      g.drawString("son1: "+son1.familyname+son1.name,5,50);
      g.drawString("son2: "+son2.familyname+son2.name,5,65);
```

```
    }
}
```

4.4.3. 实例方法和类方法的区别

我们已经知道类体中的方法分为实例方法和类方法两种,用 static 修饰的是类方法.二者有什么区别呢 我们已经知道,当一个类创建了一个对象后,这个对象就可以调用该类的方法.那么实例方法和类方法有什么区别呢

当类的字节码文件被加载到内存时,类的实例方法不会被分配入口地址,当该类创建对象后,类中的实例方法才分配入口地址,从而实例方法可以被类创建的任何对象调用执行.需要注意的是,当我们创建第一个对象时,类中的实例方法就分配了入口地址,当再创建对象时,不再分配入口地址,也就是说,方法的入口地址被所有的对象共享,当所有的对象都不存在时,方法的入口地址才被取消.

对于类中的类方法,在该类被加载到内存时,就分配了相应的入口地址.从而类方法不仅可以被类创建的任何对象调用执行,也可以直接通过类名调用.类方法的入口地址直到程序退出才被取消.

无论是类方法或实例方法,当被调用执行时,方法中的局部变量才被分配内存空间,方法调用完毕,局部变量即刻释放所占的内存.在一个方法被调用执行完毕之前,如果该方法又被调用,那么,方法的局部变量会再次被分配新的内存空间,比如,方法在递归调用时,方法中的局部变量会再次被分配新的内存空间.在下面的例子 9 中,我们通过递归地调用类中的方法,计算出 Fibinacii 序列的前十项,Fibinacii 序列的前两项是 1,后续每项的值都是该项的前两项之和.

例子 9

```
class Fibi
{  public long  fibinacii(int n)
   {        long c=0;
            if(n==1||n==2)
                 c=1;
            else
                 c=fibinacii(n-1)+fibinacii(n-2);
            return c;
    }
}
public class Example4_9
{  public static void main(String args[])
   { Fibi a=new Fibi();
     for(int i=1;i<=10;i++)
       { System.out.print(" "+a.fibinacii(i));
       }
```

```
    }
}
```

4.4.4. 通过类名直接调用类方法

类方法在类的字节码加载到内存时就分配了入口地址,因此,Java 语言允许通过类名直接调用类方法,而实例方法不能通过类名调用.在讲述类的时候我们强调过,在 Java 语言中,类中的类方法不可以操作实例变量,也不可以调用实例方法,这是因为,在类创建对象之前,实例成员变量还没有分配内存,实例方法也没有入口地址,如

```
class A
{     int x,y;
      static float f(int a){}
      float g(int x1,int x2){}
}
class B
{   public static void main(String args[])
    {  A a1=new A();
       A.f(2,3);          //合法.
       a1.f(2,4);         //合法.
       a1.g(2,5);        //合法.
       A.g(3,2);        //非法.
    }
}
```

4．5 this 关键字

this 关键字可以出现在类的实例方法中,代表使用该方法的当前对象.

为了说明 this 的用法,下面例子 10 中的"三角形"的构造方法中,有意使用了 this."三角形"类中的构造方法中,出现了 this,当一个对象使用构造方法来创建自己时,构造方法中的 this 就代表当前对象.

例子 10

```
class 三角形
{  double a,b,c;
   三角形(double a,double b,double c)
   { setABC(this,a,b,c);
   }
  void setABC(三角形 trangle,double a,double b,double c)
   { trangle.a=a;
     trangle.b=b;
     trangle.c=c;
```

```
    }
}
class Example4_10
{  public static void main(String args[])
    {三角形 tra=new 三角形(3,4,5);
     System.out.print("三角形型的三边是:"+tra.a+","+tra.b+","+tra.c+",");
     }
}
```

我们已经知道,实例方法可以操作类的成员变量.实际上,当成员变量在实例方法中出现时,默认的格式是

this.成员变量.

如

```
class  A
{   int x;
    void f()
   {  this.x=100;
   }
}
```

在上述 A 类中的实例方法 f 中出现了 this,this 就代表使用 f 的当前对象.所以,"this.x"就表示当前对象的变量 x,当对象调用方法 f 时,将 100 赋给该对象的变量 x.因此,当一个对象调用方法时,方法中的成员变量就是指分配给该对象的成员变量.因此,通常情况下,可以省略成员变量名字前面的"this.".

如

```
class  A
{      int x;
     void f()
     {   x=100;         //省略 x 前面的 this..
     }
}
```

但是,当成员变量的名字和局部变量的名字相同时.成员变量前面的"this."就不可以省略 见 4.2.3 .

我们知道类的实例方法可以调用类的其它方法,调用的默认格式是

this.方法.

如

```
class  B
{   void f()
    {  this.g();
    }
    void g()
    {  System.out.println(“ok”);
    }
}
```

在上述 B 类中的方法 f 中出现了 this,this 代表使用方法 f 的当前对象,所以,方法 f 的方法体中 this.g()就是当前对象调用方法 g,也就是说,当某个对象调用方法 f 过程中,又调用了方法 g.由于这种逻辑关系非常明确,一个方法调用另一个方法时可以省略方法名字前面的”this.”.

如

```
class  B
{    void f()
    {   g();          //省略 g 前面的 this..
    }
    void g()
    {   System.out.println(“ok”);
    }
}
```

注 this 不能出现在类方法中,这是因为,类方法可以通过类名直接调用,这时,可能还没有任何对象诞生.

4．6 包

包是 Java 语言中有效地管理类的一个机制.

4.6.1. 包语句

通过关键字 package 声明包语句. **package** 语句作为 **Java** 源文件的第一条语句,指明该源文件定义的类所在的包.**package** 语句的一般格式为

```
package  包名
```

如果源程序中省略了 **package** 语句,源文件中你定义命名的类被隐含地认为是无名包的一部分,即源文件中您定义命名的类在同一个包中,但该包没有名字.

包名可以是一个合法的标识符,也可以是若干个标识符加”.”分割而成,如

```
package sunrise;
package sun.com.cn;
```

程序如果使用了包语句,例如

```
package tom.jiafei;
```

那么你的目录结构必须包含有如下的结构

```
\tom\jiafei,
```

比如

```
c:\1000\tom\jiafei.
```

并且要将源文件保存在目录 **c:\1000\tom\jiafei** 中,然后编译源文件

```
c:\1000\tom\jiafei\javac  源文件
```

或

```
javac  c:\1000\tom\jiafei\源文件
```

例子 11　效果如图 4.12

```
package tom.jiafei;
public class  PrimNumber
{    public static void main(String args[])
     {  int sum=0,i,j;
        for( i=1;i<=10;i++)  //找出 10 以内的素数.
         {  for(j=2;j<=i/2;j++)
             {   if(i%j==0)
                   break;
             }
           if(j>i/2) System.out.print(" 素数  "+i);
         }
    }
}
```

我们保存上述源文件到 **c:\1000\tom\jiafei** 中,然后编译原文件

```
c:\1000\tom\jiafei\javac Primnumber.java
```

运行程序时必须到 **tom\jiafei** 的上一层目录 **1000** 中来运行,如

```
c:\1000\java tom.jiafei.PrimNumber
```

因为起了包名,类 **PrimNumber** 的全名已经是　**tom.jiafei.PrimNumber**　就好比大连的全名是　中国.辽宁.大连　.

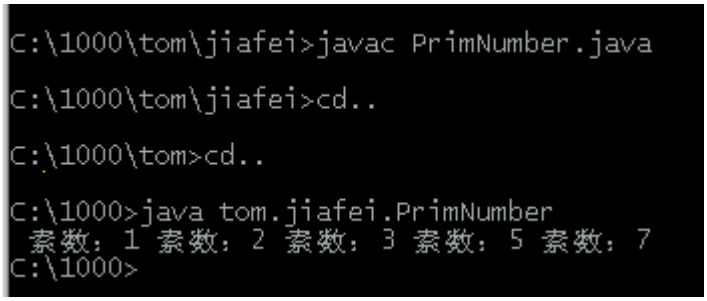


图 **4.12** 编译,运行带包名的类

4.6.2. import 语句

使用 import 语句可以引入包中的类.在编写源文件时,除了自己编写类外,我们经常需要使用 Java 提供的许多类,这些类可能在不同的包中.在学习 Java 语言时,使用已经存在的类,避免一切从头做起,这是面向对象编程的一个重要方面

为了能使用 Java 提供给我们的类,我们可以使用 import 语句来引入包中类.在一个 Java 源程序中可以有多个 import 语句,它们必须写在 package 语句 假如有 package 语句的话 和源文件中类的定义之间..Java 为我们提供了大约 130 多个包,如

java.applet	包含所有的实现 Java applet 的类
java.awt	包含抽象窗口工具集中的图形,文本,窗口 GUI 类
java.awt.image	包含抽象窗口工具集中的图像处理类
java.lang	包含所有的基本语言类
java.io	包含所有的输入输出类
java.net	包含所有实现网络功能的类
java.until	包含有用的数据类型类

如果要引入一个包中的全部类,则可以用星号来代替,如

```
import java.awt.*
```

表示引入包 java.awt 中所有的类,而

```
import java.until.Date;
```

只是引入包 **java.until** 中的 Date 类.

例如,如果我们想建立一个 java applet 程序,并想使用 java.awt 中的 Button 类和 Graphics,那么就可以使用 import 语句引入包 java.applet 中的 Applet 类和包 java.awt 中的 Button 类和 Graphics 类.

例子 12 效果如图 4.13

```
import java.applet.Applet;import java.awt.*;
public class Example4_12 extends Applet
{  Button redbutton;
    public void init()
```

```
    {   redbutton=new Button("我是一个红色的按钮");
        redbutton.setBackground(Color.red);
        add(redbutton);
    }
  public void  paint(Graphics g)
    {  g.drawString("it is a button",30,50);
    }
}
```

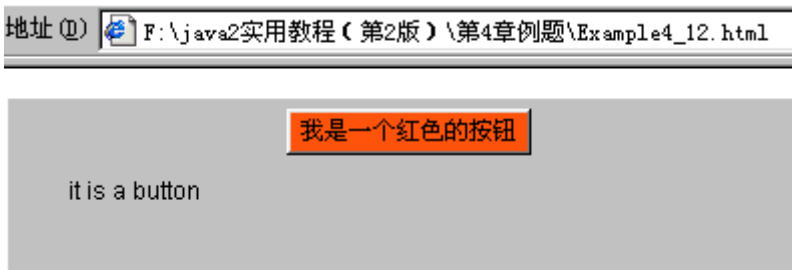


图 4.13　带按钮的小应用程序

注　系统自动为我们引入 java.lang 这个包,因此不需要再使用 import 语句引入该包. java.lang 包是 Java 语言的核心类库,它包含了运行 Java 程序必不可少的系统类.

如果使用 import 语句引入了整个包中的类,那么可能会增加编译时间.但绝对不会影响程序运行的性能,因为当程序执行时,只是将你真正使用的类的字节码文件加载到内存.

我们也可以使用 import 语句引入自己的包,如

```
import tom.jiafei.*;
```

为了能使程序使用 tom.jiafei 包中的类,我们必须在 classpath 中指明我们包的位置,例子 11 中的包 tom.jafei 的位置是 c:\1000.因此必须更新 classpath 的设置,在命令行执行如下命令

```
set classpath=c:\jdk1.2.2\jre\lib\rt.jar;.;c:\1000
```

也可以将上述命令添加到 classpath 值中.window2000,用鼠标右键点击"我的电脑",弹出菜单,然后选择属性,弹出"系统特性"对话框,再单击该对话框中的高级选项,然后点击按钮"环境变量".对于 win9x 可以将上述命令写到 autoexec.bat 文件中.

下面的例子 13 引入 tom.jiafei 包中的所有类.

例子 13　效果如图 4.14

```
import tom.jiafei.*; //引入包 tom.jiafei 中的类.
public class Example4_13
```

```
{    public static void main(String args[])
     {  PrimNumber num=new PrimNumber();//用包 tom.jiafei 中的类创建对象.
        String a[]={"ok"};
        System.out.println(a[0]);
        num.main(a);
     }
}
```

将上述源文件保存到任何一个目录下,例如

C \1000

编译

C \1000\javac Example4_13.java

运行

C \1000\java Example4_13

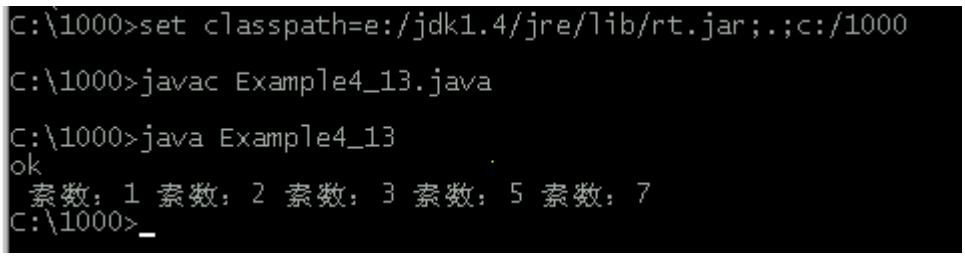


图 4.14 引如自己的包

我们也可以使用无名包中的类.假如例子 11 中没有使用包语句,如果一个程序使用 PrimNumber 类,可以将该类存放在当前程序所在的目录中.

例子 14

```
public class Example4_14
{  public static void main(String args[])
   { PrimNumber num=new PrimNumber();//要保证 PrimNuber 类和 Example4_14 类在同一目录中
     String a[]={"ok"};
     System.out.println(a[0]);
     num.main(a);
  }
}
```

4.6.3. 将类打包

我们也可以对单独的一个类进行编译,生成字节码文件,然后供其它类使用.

以下的源文件只有一个类组成,这个类可以被其它的程序引入使用.

Trangel.java

```
package tom.jiafei;
public class Trangle
{  double sideA,sideB,sideC;
   boolean boo;
   public  Trangle(double a,double b,double c)
   { sideA=a;sideB=b;sideC=c;
    if(a+b>c&&a+c>b&&c+b>a)
     {  System.out.println("我是一个三角形");
       boo=true;
     }
    else
     { System.out.println("我不是一个三角形");
       boo=false;
     }
   }
 public void  计算面积()
   {    if(boo)
        { double p=(sideA+sideB+sideC)/2.0;
          double area=Math.sqrt(p*(p-sideA)*(p-sideB)*(p-sideC)) ;
          System.out.println("面积是:"+area);
        }
      else
        { System.out.println("不是一个三角形,不能计算面积");
        }
   }
 public void 修改三边(double a,double b,double c)
   { sideA=a;sideB=b;sideC=c;
     if(a+b>c&&a+c>b&&c+b>a)
     { boo=true;
     }
    else
     { boo=false;
     }
   }
}
```

将上述源文件编译通过得到字节码文件.

在下面的例子 15 中,我们用 import 语句引如包 tom.jiafei 中的 Trangle 类.

例子 15 效果如图 4.15

```
import tom.jiafei.*;
class Example4_15
{  public static void main(String args[])
  {  Trangle trangle=new Trangle(12,3,1);
           trangle.计算面积();
           trangle.修改三边(3,4,5);
           trangle.计算面积();
  }
}
```

```
C:\1000>java Example4_15
我不是一个三角形
不是一个三角形,不能计算面积
面积是:6.0
```

图 4.15 引如包中的类

注 编写一个有价值的类是令人高兴事情,你可以将这样的类打包,形成有价值的"软件产品",供其他软件开发者使用.

4．7 访问权限

我们已经知道 当用一个类创建了一个对象之后,该对象可以通过"."运算符访问自己的变量,并使用类中的方法.但访问自己的变量和使用类中的方法是有一定限制的,通过修饰符 private,protected 和 public 来说明使用权限.

4.7.1. 私有变量和私有方法

用关键字 private 修饰的成员变量和方法称为私有变量和私有方法.如

```
class Tom
{    private float weight;              //weight被修饰为私有的float型变量.
    private float f(float a,float b)  //方法 f 是私有方法.
     {  … …
     }
… …
}
```

当在另外一个类中用类 Tom 创建了一个对象后,该对象不能访问自己的私有变量和私有方法.如

```
class Jerry
{
   void g()
   {  Tom cat=new Tom();
     cat.weight=23f; //非法.
```

```
        cat.f(3f,4f);  //非法.
    ... }
    }
```

如果 Tom 类中的某个成员是私有类变量　静态成员变量　,那么在另外一个类中,也不能通过类名 Tom 来操作这个私有类变量.如果 Tom 类中的某个方法是私有的类方法,那么在另外一个类中,也不能通过类名 Tom 来调用这个私有的类方法.

对于私有成员变量或方法,只有在本类中创建该类的对象时,这个对象才能访问自己的私有成员变量和类中的私有方法,如例 16 所示.

例子 16

```
class Example4_16
{    private int money;
    Example4_16()
    {  money=2000;
    }
    private int getMoney()
    {  return money;
    }
    public static void main(String args[])
    { Example4_16 exa=new Example4_16();
      exa.money=3000;int m=exa.getMoney();
      System.out.println("money="+m);
    }
}
```

4.7.2.　共有变量和共有方法

用 public 修饰的成员变量和方法称为共有变量和共有方法.如

```
class Tom
{  public float weight;              //weight被修饰为public的float型变量.
  public float f(float a,float b)  //方法 f 是public方法.
     {  ... ...
     }
}
```

当我们在任何一个类中用类 Tom 创建了一个对象后,该对象能访问自己的 public 变量和类中的 public 方法.如

```
class Jerry
{  void g()
```

```
  { Tom cat=new Tom();
    cat.weight=23f;  //合法.
    cat.f(3,4);     //合法.
  }
}
```

如果 Tom 类中的某个成员是 public 类变量,那么在任何一个类中,也可以通过类名 **Tom** 来操作 **Tom** 的这个成员变量.如果 Tom 类中的某个方法是 public 类方法,那么我们在任何一个类中,也可以通过类名 **Tom** 来调用 **Tom** 类中的这个 **public** 类方法.

4.7.3. 友好变量和友好方法

不用 private,public ,protected 修饰符的成员变量和方法被称为友好变量和友好方法.如

```
class Tom
{  float weight;                //weight 是友好的 float 型变量.
   float f(float a,float b)     //方法 f 是友好方法.
   {
   }
}
```

当在另外一个类中用类 Tom 创建了一个对象后,如果这个类与 Tom 类在同一个包中,那么该对象能访问自己的友好变量和友好方法.在任何一个与 Tom 同一包中的类中,也可以通过 Tom 类的类名访问 Tom 类的类友好成员变量和类友好方法.

假如 Jerry 与 Tom 是同一个包中的类,那么,下述 Jerry 类中的**"cat.weight","cat.f(3,4)"** 都是合法的.

```
class Jerry
{  void g()
   {   Tom cat=new Tom();
       cat.weight=23f;   //合法.
       cat.f(3,4);       //合法.
   }
}
```

您在源文件中编写命名的类总是在同一包中的.如果你在源文件中用 import 语句引入了另外一个包中的类,并用该类创建了一个对象,那么该类的这个对象将不能访问自己的友好变量和友好方法.

4.7.4. 受保护的成员变量和方法

用 protected 修饰的成员变量和方法被称为受保护的成员变量和受保护的方法.如

```
class Tom
{   protected  float weight; //weight被修饰为public的float型变量.
   protected float f(float a,float b) //方法 f是public方法.
   {
   }
}
```

当在另外一个类中用类 Tom 创建了一个对象后,如果这个类与类 Tom 在同一个包中,那么该对象能访问自己的 protected 变量和 protected 方法.在任何一个与 Tom 同一包中的类中,也可以通过 Tom 类的类名访问 Tom 类的 protected 类变量和 protected 类方法.

假如 Jerry 与 Tom 是同一个包中的类,那么,Jerry 类中的**"cat.weight","cat.f(3,4)"**都是合法的.

```
class Jerry
{   void g()
   { Tom cat=new Tom();
     cat.weight=23f;   //合法.
     cat.f(3,4);       //合法.
   }
}
```

注 在后面讲述子类时,将讲述"受保护 protected "和"友好的"之间的区别.

4.7.5.public 类与友好类

类声明时,如果关键字 class 前面加上 public 关键字,就称这样的类是一个 public 类如

```
public class A
{……
}
```

可以在任何另外一个类中,使用 **public** 类创建对象.如果一个类不加 **public** 修饰,如

```
class A
{……
}
```

这样的类称做友好类,那么另外一个类中使用友好类创建对象时,要保证它们是在同一包中.

注 不能用 protected 和 private 修饰类.

访问权限的级别排列.访问限制修饰符,按访问权限从高到低的排列顺序是 public,protected,友好的,private.

4．8　类的继承

继承是一种由已有的类创建新类的机制.利用继承,我们可以先创建一个共有属性的一般类,根据该一般类再创建具有特殊属性的新类,新类继承一般类的状态和行为,并根据需要增加它自己的新的状态和行为.由继承而得到的类称为子类,被继承的类称为父类　超类　.Java 不支持多重继承　子类只能有一个父类　.

4.8.1.　创建子类

在类的声明中,通过使用关键字 extends 来创建一个类的子类,格式如下:

```
class 子类名 extends  父类名
{... ..
}
```

例如

```
class Students extends People
{... ..
}
```

把 Students 声明为 People 类的子类,People 是 Students 的父类.

4.8.2.　子类的继承性

1　子类和父类在同一包中的继承性

如果子类和父类在同一个包中,那么,子类自然地继承了其父类中不是 private 的成员变量作为自己的成员变量,并且也自然地继承了父类中不是 private 的方法作为自己的方法.

例子 17

```
import java.applet.*;
import java.awt.*;
class Father
{  private int money;
   float weight,height;
   String head;
   String speak(String s)
   { return s ;
   }
}
class Son extends Father
{   String hand ,foot;
```

```
}
public class Example4_17 extends Applet
{   Son boy;
    public void init()
   { boy=new Son();
    boy.weight=1.80f;  boy.height=120f;
    boy.head="一个头"; boy.hand="两只手 ";
    boy.foot="两只脚";
   }
   public void paint(Graphics g)
   {  g.drawString(boy.speak("我是儿子"),5,20);
     g.drawString(boy.hand+boy.foot+boy.head+boy.weight+boy.height,5,40);
   }
}
```

2　子类和父类不在同一包中的继承性

如果子类和父类不在同一个包中,那么,子类继承了父类的 protected,public 成员变量做为子类的成员变量,并且继承了父类的 protected,public 方法为子类的方法.如果子类和父类不在同一个包里,子类不能继承父类的友好变量和友好方法.

下面的例子 18 中,Father 和 Jerry 分别隶属不同的包.

例子 18

Father.java:

```
package tom.jiafei;
public class  Father
{   int  height;
    protected int money;
   public   int weight;
   public Father(int m) {
   {  money=m;
   }
   protected int getMoney()
   {  return money;
   }
    void setMoney(int newMoney)
   {  money=newMoney;
   }
}
```

Jerry.java:

```
package sun.com;
import tom.jiafei.Father;
public class Jerry extends Father   //Jerry和Father在不同的包中.
{  public Jerry()
   {   super(20);
   }
   public static void main(String args[])
   {  Jerry  jerry=new Jerry();
      jerry.height=12;          //非法,因为Jerry没有继承友好的height.
      jerry.weight=200;         //合法.
      jerry.money=800;          //合法.
      int m=jerry.getMoney(); //合法..
      jerry.setMoney(300);     //非法,因为Jerry没有继承友好的方法setMoney.
      System.out.println("m="+m);
   }
}
```

注　如果一个类的声明中没有使用 extends 关键字,这个类被系统默认为是 Object 的子类.Object 是包 java.lang 中的类.

4.8.3.　成员变量的隐藏和方法的重写

当我们在子类中定义的成员变量和父类中的成员变量同名时,则父类中的成员变量不能被继承,此时称子类的成员变量隐藏了父类的成员变量.当子类中定义了一个方法,并且这个方法的名字,返回类型,及参数个数和类型和父类的某个方法完全相同时,父类的这个方法将被隐藏.我们说　我们重写了父类的方法或置换了父类的方法.

子类通过成员变量的隐藏和方法的重写可以把父类的状态和行为改变为自身的状态和行为.在下面的例子中,子类重写了父类的方法 f.

例子 19　效果如图 4.16

```
import java.applet.*;
import java.awt.*;
class Chengji
{  float f(float x,float y)
   {  return x*y;
   }
}
class Xiangjia  extends Chengji
{  float f(float x,float y)
```

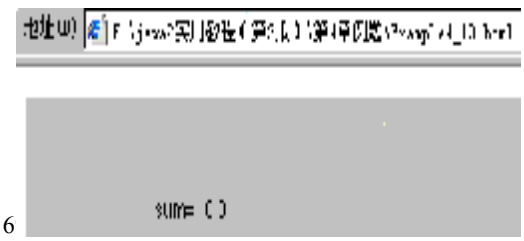


图 4.16　方法重写

```
    { return x+y ;
    }
}
public class Example4_19 extends Applet
{  Xiangjia sum;
   public void init()
   {  sum=new Xiangjia();
   }
   public void paint(Graphics g)
   { g.drawString("sum="+sum.f(4,6),100,40);
   }
}
```

对于子类创建的一个对象,如果子类重写了父类的方法,则运行时系统调用子类重写的方法,如果子类继承了父类的方法　未重写　,那么子类创建的对象也可以调用这个方法,只不过方法产生的行为和父类的相同而已.如下述例子 20 所示.

例子 20

```
import java.applet.*;
import java.awt.*;
class Area
{  float f(float r )
   {  return 3.14159f*r*r;
   }
   float g(float x,float y)
   {  return x+y;
   }
}
class Circle extends Area
{   float f(float r)
   {  return 3.14159f*2.0f*r;
   }
}
public class Example4_20 extends Applet
{  Circle yuan;
   public void init()
   {  yuan=new Circle();
   }
   public void paint(Graphics g)
   { g.drawString("yuan= "+yuan.f(5.0f),5,20);  //调用子类重写的方法 f.
```

```
        g.drawString(" "+yuan.g(2.0f,8.0f),5,40);  //调用继承父类的方法 g.
      }
   }
```

注 重写父类的方法时,不可以降低方法的访问权限.下面的代码中,子类重写父类的方法 f,该方法在父类中的访问权限是 protected 级别,子类重写时不允许级别低于 protected 级别.

如

```
import java.applet.*;
import java.awt.*;
class Chengji
{  protected  float f(float x,float y)
   {  return x*y;
   }
}
class Xiangjia  extends Chengji
{  float f(float x,float y) //非法,因为降低了访问级别.
   {  return x+y ;
   }
}
class Xiangjian  extends Chengji
{  public  float f(float x,float y) //合法,没有降低访问级别.
   {  return x-y ;
   }
}
```

4.8.4. final 类 final 方法

final 类不能被继承,即不能有子类.如

```
final class A
{ ...
}
```

A 就是一个 **final** 类.有时候是出于安全性的考虑,将一些类修饰为 **final** 类.例如 **Java** 提供的 **String** 类,它对于编译器和解释器的正常运行有很重要的作用,对它不能轻易改变,因此它被修饰为 **final** 类.

如果一个方法被修饰为 **final** 方法,则这个方法不能被重写.如果一个成员变量被修饰为 **final** 的,就是常量.

4．9　对象的上转型对象

我们经常说”老虎是哺乳动物”,”狗是哺乳动物”等.若哺乳类是老虎类的父类,这样说当然正确,但当你说老虎是哺乳动物时,老虎将失掉老虎独有的属性和功能.下面我们就介绍对象的上转型对象.

假设,A 类是 B 类的父类,当我们用子类创建一个对象,并把这个对象的引用放到父类的对象中时,比如

```
A a;
A=new B();
```

或

```
A a;
B b=new B();
a=b;
```

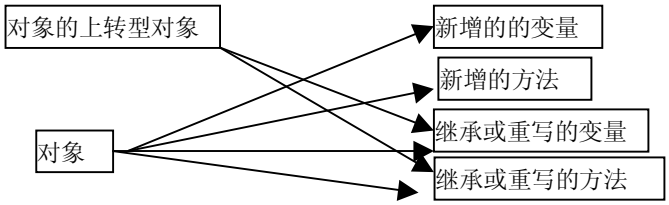


图 4.17　上转型对象示意图

称这个父类对象　a,是子类对象 b 的上转型对象　好比说　”老虎是哺乳动物”　.

对象的上转型对象的实体是子类负责创建的,但上转型对象会失去原对象的一些属性和功能.上转型对象具有如下特点　如图 4.17 所示意

1　上转对象不能操作子类新增的成员变量　失掉了这部分属性　不能使用子类新增的方法　失掉了一些功能　.

2　上转型对象可以操作子类继承或重写的成员变量,也可以使用子类继承的或重写的方法.

3　如果子类重写了父类的某个方法后,当对象的上转对象调用这个方法时一定是调用了这个重写的方法,因为程序在运行时知道,这个上转对象的实体是子类创建的,只不过损失了一些功能而已.

注　　不要将父类创建的对象和子类对象的上转对象混淆.

可以将对象的上转型对象再强制转换到一个子类对象,这时,该子类对象又具备了子类所给的所有属性和功能.

不可以将父类创建的对象的引用赋值给子类声明的对象　不能说　”哺乳动物是老虎”　.

下面的例子 21 中,monkey 是 People 类型对象的上转型对象.

例子 21　效果如图 4.18

```
class  类人猿
{  private int n=100;
   void crySpeak(String s)
   {  System.out.println(s);
   }
}
class People extends 类人猿
```

```
F:\8000>java Example4_21
**I love this game**
100
```

图 4.18　上转型对象

```
{  void computer(int a,int b)
   {  int c=a*b;
      System.out.println(c);
   }
 void crySpeak(String s)
   { System.out.println("**"+s+"**");
   }
}
class Example4_21
{  public static void main(String args[])
   {  类人猿 monkey=new People();   //monkey是People对象的上转型对象.
      monkey.crySpeak("I love this game");
      People people=(People)monkey; //把上转型对象强制转化为子类的对象.
      people.computer(10,10);
   }
}
```

在上述例子中,上转对象 **monkey** 调用方法:

```
monkey.crySpeak("I love this game");
```

得到的结果是"**I love this game**".而不是"I love this game".
因为 **monkey** 调用的是子类重写的方法 **crySpeak.**

注 不可以

```
monkey.n=1000;
```

因为子类本来就没有继承 **n**,也不可以

```
monkey.computer(10,10);
```

因为 **computer** 是子类新增的方法.

4.10 多态性

我们经常说 "哺乳动物有很多种叫声",比如,"吼""嚎""汪汪""喵喵"等,这就是叫声的多态.

当一个类有很多子类时,并且这些子类都重写了父类中的某个方法.那么当我们把子类创建的对象的引用放到一个父类的对象中时,就得到了该对象的一个上转型对象,那么这个上转的对象在调用这个方法时就可能具有多种形态,因为不同的子类在重写父类的方法时可能产生不同的行为,比如,狗类的上转型对象调用"叫声"方法时产生的行为是"旺旺",而猫类的上转型对象调用"叫声"方法时,产生的行为是"喵喵"等等.

多态性就是指父类的某个方法被其子类重写时,可以各自产生自己的功能行为.

下面的例子 22 展示了多态.

例子 22

```
class  动物
{  void cry()
   {
   }
}
class 狗 extends 动物 {
{  void cry()
   {  System.out.println("汪汪.....");
   }
}
class 猫 extends 动物
{  void cry()
   {  System.out.println("喵喵.....");
   }
}
class Example4_22
{  public static void main(String args[])
   {  动物 dongwu;
      if(Math.random()>=0.5)
         {
           dongwu=new 狗();
           dongwu.cry();
         }
       else
         {
           dongwu=new 猫();
           ongwu.cry();
         }
   }
}
```

4．11　abstract 类和 abstract 方法

用关键字 abstract 修饰类称为 abstract 类　抽象类　.如

```
abstract class A
{ ......
}
```

abstract 类不能用 **new** 运算创建对象,必须产生其子类,由子类创建对象.

对于 **abstract** 方法,只允许声明,而不允许实现.如

```
abstract  void draw()
```

如果一个类是一个 **abstract** 类的子类,它必须具体实现父类的 **abstract** 方法.

如果一个类中含有 **abstract** 方法,那么这个类必须用 **abstract** 来修饰(**abstract** 类也可以没有 **abstract** 方法).

一个 **abstract** 类只关心它的子类是否具有某种功能,并不关心功能的具体行为,功能的具体行为由子类负责实现.

在下面的例子 **23** 中,有一个 **abstract** 的"图形"类,图形类要求其子类都必须有具体计算面积的功能.

例子 23　效果如图 4.19

```
abstract class 图形
{  public abstract double 求面积();
}
class 梯形 extends 图形
{  double a,b,h;
   梯形(double a,double b,double h)
   {  this.a=a;this.b=b;this.h=h;
   }
   public double 求面积()
   {  return((1/2.0)*(a+b)*h);
   }
}
class 圆形 extends 图形
{  double r;
   圆形(double r)
   {  his.r=r;
   }
   public double 求面积()
   {  return(3.14*r*r);
   }
}
class 堆
   {  图形 底;
      double 高;
    堆(图形 底,double 高)
   {  this.底=底;
```

```
F:\8000>java Example4_23
梯形的面积48.15
梯形底的堆的体积481.5
半径是10的圆的面积314.0
圆形底的堆的体积3140.0
```

图 4.19　使用 abstract 类

```
        this.高=高;
    }
     void 换底(图形 底)
    {  this.底=底;
    }
     public double 求体积()
    {   return (底.求面积()*高)/3.0;
    }
}
public class Example4_23 {
{  public static void main(String args[])
    {  堆 zui;
       图形 tuxing;
       tuxing=new 梯形(2.0,7.0,10.7);
       System.out.println("梯形的面积"+tuxing.求面积());
       zui=new  堆(tuxing,30);
       System.out.println("梯形底的堆的体积"+zui.求体积());
       tuxing=new 圆形(10);
       System.out.println("半径是10的圆的面积"+tuxing.求面积());
       zui.换底(tuxing);
       System.out.println("圆形底的堆的体积"+zui.求体积());
    }
}
```

4．12　super 关键字

我们已经知道,如果子类中定义的成员变量和父类中的成员变量同名时,则父类中的成员变量不能被继承,此时称子类的成员变量隐藏了父类的成员变量.当子类中定义了一个方法,并且这个方法的名字,返回类型,及参数个数和类型和父类的某个方法完全相同时,父类的这个方法将被隐藏,既不能被子类继承下来.如果我们在子类中想使用被子类隐藏的父类的成员变量或方法就可以使用关键字 super.

1　使用 super 调用父类的构造方法

子类不继承父类的构造方法,因此,子类如果想使用父类的构造方法,必须在子类的构造方法中使用,并且必须使用关键字 super 来表示,而且 super 必须是子类构造方法中的头一条语句.如例 24 所示.

例子 24

```
class Student
{  int number;String name;
   Student(int number,String name)
```

```
    {  this.number=number;this.name=name;
       System.out.println("I am "+name+ "my number is "+number);
    }
 }
class Univer_Student extends Student
{  boolean 婚否;
   Univer_Student(int number,String name,boolean b)
   {  super(number,name);
      婚否=b;
      System.out.println("婚否="+婚否);
   }
 }
public class Example4_24
{  public static void main(String args[])
   {  Univer_Student zhang=new Univer_Student(9901,"和晓林",false);
   }
}
```

运行结果

```
I am 和晓林 my number is 9901
婚否=false.
```

需要注意的是　如果在子类的构造方法中,没有显示地使用 **super** 关键字调用父类的某个构造方法,那么默认地有

```
super();
```

语句,即调用父类的不带参数的构造方法.如果父类没有提供不带参数的构造方法,就会出现错误.

2　使用 **super** 操作被隐藏的成员变量和方法

如果我们在子类中想使用被子类隐藏了的父类的成员变量或方法就可以使用关键字 **super**.比如　**super.x,super.play()**,就是被子类隐藏的父类的成员变量 **x** 和方法 **play()**.

例子 25

```
class Sum
{  int n;
   float f()
   {  float sum=0;
      for(int i=1;i<=n;i++)
          sum=sum+i;
```

```
        return sum;
    }
 }
class Average extends Sum
{  int n;
   float f()
   {  float c;
      super.n=n;
      c=super.f();
      return c/n;
   }
   float g()
   {  float c;
     c=super.f();
      return c/2;
   }
 }
public class Example4_25
{  public static void main(String args[])
   {  Average aver=new Average();
      aver.n=100;
      float result_1=aver.f();
      float result_2=aver.g();
      System.out.println("result_1="+result_1);
      System.out.println("result_2="+result_2);
   }
}
```

运行结果

```
result_1=50.50
result_2=2525.0
```

4.13 接口

Java 不支持多继承性,即一个类只能有一个父类.单继承性使得 Java 简单,易于管理程序.为了克服单继承的缺点,Java 使用了接口,一个类可以实现多个接口.

使用关键字 interface 来定义一个接口.接口的定义和类的定义很相似,分为接口的声明和接口体.

4.13.1. 接口的声明与使用

1 接口声明

我们曾使用 class 关键字来声明类,接口通过使用关键自 interface 来声明. 格式

```
interface  接口的名字
```

2 接口体

接口体中包含常量定义和方法定义两部分.接口体中只进行方法的声明,不许提供方法的实现,所以,方法的定义没有方法体,且用分号” ”结尾.如

```
interface  Printable
{  final  int MAX=100;
   void add();
   float sum(float x ,float y);
}
```

3 接口的使用

一个类通过使用关键字 **implements** 声明自己使用一个或多个接口.如果使用多个接口,用逗号隔开接口名.如

```
class A implements Printable,Addable
```

类 **A** 使用接口 **Printable** 和接口 **Addable.**

```
class Dog extends Animal implements Eatable,Sleepable
```

类 **Dog** 使用接口 **Eatable** 和接口 **Sleepable.**

如果一个类使用了某个接口,那么这个类必须实现该接口的所有方法,即为这些方法提供方法体.需要注意的是 在类中实现接口的方法时,方法的名字,返回类型,参数个数及类型必须与接口中的完全一致.特别要注意的是 接口中的方法被默认是 **public** 的,所以类在实现接口方法时,一定要用 **public** 来修饰.另外,如果接口的方法的返回类型如果不是 **void** 的,那么在类中实现该接口方法时,方法体至少要有一个 **return** 语句 如果是 **void** 型,类体除了两个大括号外,也可以没有任何语句.

Java 为我们提供的接口都在相应的包中,通过引入包可以使用 **Java** 提供的接口.也可以自己定义接口,一个 **java** 源文件就是由类和接口组成的.

下面的例子 **26** 实用了接口.

例子 26

```
import java.applet.*;import java.awt.*;
interface Computable
{  final int MAX=100;
   void speak(String s);
```

```
   int f(int x);
    float g(float x,float y);
}
class China implements Computable
{  int xuehao;
   public  int f(int x)   //不要忘记public关键字.
   {  int sum=0;
      for(int i=1;i<=x;i++)
         { sum=sum+i;
         }
       return sum;
   }
   public float g(float x,float y)
   {  return 6;                    //至少有return语句.
   }
   public void speak(String s)
   {
   }
}
class Japan implements Computable
{  int xuehao;
   public int f(int x)
   {  return 68;
   }
    public  float g(float x,float y)
   {  return x+y;
   }
 public void speak(String s)
   {                              //必须有方法体,但体内可以没有任何语句.
   }
}
public class Example4_26 extends Applet
{  China Li; Japan Henlu;
   public void init()
   {  Li=new China();   Henlu=new Japan();
      Li.xuehao=991898; Henlu.xuehao=941448;
   }
   public void paint(Graphics g)
   {  g.drawString("xuehao:"+Li.MAX+Li.xuehao+"从1到100求和"+Li.f(100),10,20);
```

```
        g.drawString("xuehao:"+Henlu.MAX+Henlu.xuehao+"加法"+Henlu.g(2.0f,3.0f),10,40);
      }
   }
```

注 如果一个类声明实现一个接口,但没有实现接口中的所有方法,那么这个类必须是 abstract 类,例如

```
interface Computable
{  final int MAX=100;
   void speak(String s);
   int f(int x);
   float g(float x,float y);
}
abstract class A implements Computable
{  public  int f(int x)
   {  int sum=0;
      for(int i=1;i<=x;i++)
       { sum=sum+i;
       }
     return sum;
   }
}
```

4.13.2. 理解接口

接口的语法规则很容易记住,但真正理解接口更重要.你可能注意到,在上述例子 26 中如果去掉接口,并把程序中的 Li.MAX,Henlu.MAX 去掉,上述程序的运行没有任何问题.那为什么要用接口呢

假如轿车,卡车,拖拉机,摩托车,客车都是机动车的子类,其中机动车是一个抽象类.如果机动车中有一个抽象方法 "收取费用",那么所有的子类都要实现这个方法,即给出方法体,产生各自的收费行为.这显然不符合人们的思维方法,因为拖拉机可能不需要有"收取费用"的功能,而其他的一些类,比如飞机,轮船等可能也需要具体实现"收取费用".接口可以增加很多类都需要实现的功能,不同的类可以使用相同的接口,同一个类也可以实现多个接口.接口只关心功能,并不关心功能的具体实现,比如"客车类"实现一个接口,该接口中有一个"收取费用"的方法,那么这个"客车类"必须具体给出怎样收取费用的操作,即给出方法的方法体,不同车类都可以实现"收取费用",但"收取费用"的手段可能不相同.接口的思想在于它可以增加很多类都需要实现的功能,使用相同的接口类不一定有继承关系,就象各式各样的商品,它们可能隶属不同的公司,工商部门要求都必须具有显示商标的功能 实现同一接口 ,但商标的具体制作由各个公司自己去实现.

再比如,你是一个项目主管,你需要管理许多部门,这些部门要开发一些软件所需要的类,你可能要求某个类实现一个接口,也就是说你对一些类是否具有这个功能非常关心,但不关心

功能的具体体现,比如,这个功能是 speakLove ,但你不关心是用汉语实现功能 speakLove 或用英语实现 speakLove.在某些时候,你也许打一个电话就可以了,告诉远方的一个开发部门实现你所规定的接口,并建议他们用汉语来实现 speakLove.如果没有这个接口,你可能要化很多的口舌来让你的部门找到那个表达爱的方法,也许他们给表达爱的那个方法起的名字是完全不同的名字.

例子 27 效果如图 4.20

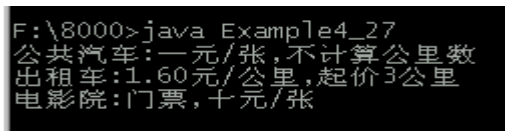


图 4.20 使用接口

```
interface  收费
{ public void  收取费用();
}
class 公共汽车 implements 收费
{  public  void 收取费用()
   {  System.out.println("公共汽车:一元/张,不计算公里数");
   }
 }
class 出租车 implements 收费
{  public void 收取费用()
   {  System.out.println("出租车:1.60元/公里,起价3公里");
   }
 }
class 电影院 implements 收费
{  public void 收取费用()
   {  System.out.println("电影院:门票,十元/张");
   }
}
class Example4_27
{  public static void main(String args[])
   {  公共汽车 七路=new 公共汽车();
      出租车   天宇=new 出租车();
      电影院   红星=new 电影院();
      七路.收取费用();天宇.收取费用();
      红星.收取费用();
   }
}
```

注 接口声明时,如果关键字 interface 前面加上 public 关键字,就称这样的接口是一个 public 接口 public 接口可以被任何一个类使用.如果一个接口不加 public 修饰,就称做友好接口类,友好接口可以被同一包中的类.

4.13.3. 接口回调

接口回调是指　可以把实现某一接口的类创建的对象的引用赋给该接口声明的接口变量中.那么该接口变量就可以调用被类实现的接口中的方法.实际上,当接口变量调用被类实现的接口中的方法时,就是通知相应的对象调用接口的方法.

下面的例子 28 使用了接口的回调技术.

例子 28

```
interface  ShowMessage
{   void 显示商标(String s);
}
class TV implements ShowMessage
{  public void 显示商标(String s)
   {  System.out.println(s);
   }
}
class PC implements ShowMessage
{  public void 显示商标(String s)
   {  System.out.println(s);
   }
}
public class Example4_28
{  public static void main(String args[])
   {  ShowMessage sm;                    //声明接口变量.
      sm=new TV();                       //接口变量中存放对象的引用.
      sm.显示商标("长城牌电视机");       //接口回调.
      sm=new PC();                       //接口变量中存放对象的引用.
      sm.显示商标("联想奔月 5008PC 机"); //接口回调.
   }
}
```

在下面的例子 29 中,梯形,圆形都必须有具体计算面积的功能.请读者比较例子 29 和例子 23 的不同之处.

例子 29　效果如图 4.21

```
interface  Computerable
{  public  double 求面积();
}
class 梯形 implements Computerable
```

```
F:\8000>java Example4_29
梯形的面积48.15
梯形底的堆的体积481.5
半径是10的圆的面积314.0
圆形底的堆的体积3140.0
```

图 4.21　接口回调

```
{  double a,b,h;
    梯形(double a,double b,double h)
   {  this.a=a;this.b=b;this.h=h;
   }
    public double 求面积()
   {  return((1/2.0)*(a+b)*h);
   }
}
class 圆形 implements Computerable
{  double r;
   圆形(double r)
   {  this.r=r;
   }
   public double 求面积()
   { eturn(3.14*r*r);
   }
}
class 堆
{  Computerable 底;            //声明一个接口变量,可以回调"求面积"方法.
   double 高;
   堆(Computerable 底,double 高)
   {  this.底=底;
      this.高=高;
   }
    void 换底(Computerable 底)
   {  this.底=底;
   }
    public double 求体积()
   {  return (底.求面积()*高)/3.0;
   }
}
public class Example4_29
{  public static void main(String args[])
   {  堆 zui;
      Computerable bottom;
      bottom=new 梯形(2.0,7.0,10.7); //接口变量中存放对象的引用.
      System.out.println("梯形的面积"+bottom.求面积()); //bottom 接口回调,求面积.
      zui=new  堆(bottom,30);
      System.out.println("梯形底的堆的体积"+zui.求体积());
```

```
        bottom=new 圆形(10);  //接口变量中存放对象的引用.
        System.out.println("半径是10的圆的面积"+bottom.求面积());
        zui.换底(bottom);
        System.out.println("圆形底的堆的体积"+zui.求体积());
    }
}
```

4．14　jar 文件

4.14.1. 将应用程序压缩为 jar 文件

可以使用 jar.exe 把一些文件压缩成一个 JAR 文件,来发布我们的应用程序.我们可以把 java 应用程序中涉及到的类压缩成一个 JAR 文件,比如 Tom.jar,然后使用 java 解释器　使用参数-jar　执行这个压缩文件,或用鼠标双击该文件,执行这个压缩文件

```
java  -jar  Tom.jar
```

假设应用程序中有两个类 A,B,其中 A 是主类.
生成一个 Jar 文件的步骤如下

1　首先用文本编辑器　比如 Windows 下的记事本　编写一个清单文件

Mymoon.mf

```
Manifest-Version: 1.0
Main-Class: A
Created-By: 1.2.2(Sun Microsystems Inc.)
```

比如,保存 Mymoon.mf 到 D:\test.需要注意的是在编写清单文件时,在”Manifest-Version　”和 “1.0”之间,”Main-Class　”和主类”A”之间,以及”Created-By　”和”1.2.2”之间必须有且只有一个空格.

2　生成 JAR 文件

```
D:\test\jar  cfm  Tom.jar  Mymoon.mf  A.class  B.class
```

其中参数 c 表示要生成一个新的 JAR 文件　f 表示要生成的 JAR 文件的名字　m 表示文件清单文件的名字.

现在就可以将 Tom.jar 文件拷贝到任何一个安装了 java 运行环境 版本号需高于 1.2.2 的计算机上,只要用鼠标双击该文件就可以运行该 java 应用程序了.

需要注意的是如果机器上没有安装过中文版 WinRAR 解压缩软件,那么 Tom.jar 的文件类型是　Executable Jar File.如果机器上安装过中文版 WinRAR 解压缩软件,并将.jar 文件与该解压缩软件做了关联,那么 Tom.jar 的文件类型是　WinRAR,在这种情况下,当用鼠标双击该文件时 WinRAR 解压缩软件会运行起来准备进行解压缩操作,使得我们的 java 程序无法运行.因此,我们在发布我们的软件时,还应该再写一个有如下内容的 bat 文件 Tom.bat

用文本编辑器 .

```
javaw  -jar  Tom.jar
```

另外再写一个帮助文件 help.txt

您可以用鼠标双击 A.bat 或 Tom.jar 来运行本软件,但是如果您的计算机上安装了中文版 WinRAR 解压缩软件,并将.jar 文件与该解压缩软件做了关联,那么当用鼠标双击 Tom.jar 文件时 WinRAR 解压缩软件会运行起来准备进行压缩操作,这时您可以双击 Tom.bat 来运行我们的软件.

将该.bat 文件,.jar 文件,帮助文件一同发布.

4.14.2. 将类压缩成 jar 文件

我们可以使用 jar.exe 把一些类的字节码文件压缩成一个 JAR 文件,然后将这个 jar 文件存放到 Java 运行环境的宽展中,即将该 jar 文件存放在 JDK 安装目录的 jre\lib\ext 文件夹中.这样,其他的程序就可以使用这个 jar 文件中的类来创建对象了.

现在,我们就将 D:\test 中的 Test1.class 和 Test2.class 压缩成一个 jar 文件 Jerry.jar.

Test1.java

```
public class  Test1
{  public void fTest1()
   {  System.out.println("I am a method In Test1 class");
   }
}
```

Test2.java

```
public class Test2
{  public void fTest2()
   {  System.out.println("I am a method In Test2 class");
   }
}
```

1 首先编写一个清单文件 Manifestfiles .

moon.mf

```
Manifest-Version: 1.0
Class: Test1  Test2
Created-By: 1.2.2 (Sun Microsystems Inc.)
```

保存 moon.mf 到 D:\test.

2 生成 JAR 文件.

```
D:\test\jar  cfm  Jerry.jar  moon.mf  Test1.class  Test2.class
```

下面的 Use 类中.使用了 Jerry.jar 中的 Test1,Test2 类.

```
class Use
{  public static void main(String args[])
   {  Test1 a=new Test1(); Test2 b=new Test2();
      a.fTest1(); b.fTest2();
   }
 }
```

注　应当保证使用 bin 下的解释器来运行例子　path c:/jdk1.4/bin　.

4.14.3. 更新,查看 jar 文件

可以使用参数 t 和 f 查看一个 JAR 文件中的内容

```
D:\test\jar tf  Tom.jar
```

使用参数 x 和 f 解压 JAR 文件

```
D:\test\jar xf Tom.jar
```

使用参数-u 和 f 更新一个 JAR 文件,下面的命令将一个新的文件增加到 Tom.jar 中

```
D:\test\ jar –uf  Tom.jar  oranger.class
```

习 题 四

1.举例说明 protected 方法和友好方法的区别.
2.举例说明类方法和实例方法以及类变量和实例变量的区别.
3.子类将继承父类的那些成员变量和方法　子类在什么情况下隐藏父类的成员变量和方法　在子类中是否允许有一个方法和父类的方法名字相同,而类型不同　说明你的理由
.下列程序有什么错误

```
public class  Takecare
{  int a=90;
   static float b=10.98f;
   public static void main(String args[])
   {  float c=a+b;
     System.out.println("="+c);
   }
}
```

5.使用接口有哪些注意事项　模仿例子 28,编写一个类实现两个接口的程序.
6.上机编译运行下列程序.

```
class 学生
```

```
{   String 书,笔;int 学号,年级;
   学生(int number,int grade )
   {  学号=number;年级=grade;
   }
  void 去教室()
   {  System.out.println("我带着"+书+"和"+笔+"来到了教室,准备听课");
   }
}
class Univ
{  public static void main(String args[])
   {  学生 张小林=new 学生(9901,2);
     张小林.书="英语书";  张小林.笔="钢笔"; 张小林.去教室();
   }
}
```

第五章　　数组与字符串

数组是相同类型的数据按顺序组成的一种复合数据类型.通过数组名加数组下标,来使用数组中的数据.下标从 0 开始排序.

5.1. 声明数组

声名数组包括数组的名字,数组包含的元素的数据类型.

声明一维数组有下列两种格式

```
数组元素类型　数组名字[ ]
数组元素类型[ ]　数组名字
```

声明 2 维数组有下列两种格式

```
数组元素类型　数组名字[ ][ ]
数组元素类型[ ] [ ] 数组名字
```

例如

```
float  boy[ ];  double  girl [ ]; char cat[ ]
float  a[ ][];  double  b[ ] [ ]; char  d[ ][ ]
```

将来数组 boy 的元素可以存放 float 型数据.

数组的元素的类型可以是 Java 的任何一种类型.假如你已经定义了一个 People 类,那么你可以声明一个数组

```
People china[ ];
```

将来数组 china 元素可以存放 People 类型数据,即 People 类创建的对象.

5.2. 创建数组

声明数组仅仅是给出了数组名字和元素的数据类型,要想真正的使用数组必须为它分配内存空间,即创建数组.在为数组分配内存空间时必须指明数组的长度.为数组分配内存空间的格式如下

```
数组名字 = new  数组元素的类型[数组元素的个数];
```

例如

```
boy= new  float[7];
```

声明数组和创建数组可以一起完成,

例如

```
float boy[]=new float[7]
```

内存示意如图 5.1 所示.

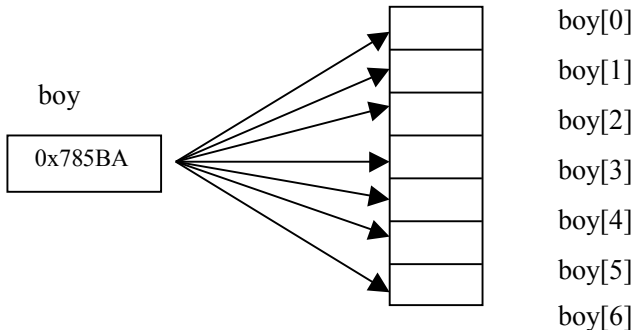


图 5.1 内存模式

二维数组和一维数组一样,在定义之后必须用 new 运算符分配内存空间,例如

```
int  mytwo[][];
mytwo=new int [3][5];
```

或

```
int mytwo[][]=new int[3][4];
```

注　和 C 语言不同的是,Java 允许使用 int 型变量指定数组的大小,例如

```
int  size=30;
double  number[]=new double[size];
```

5.3. 数组元素的使用

一维数组通过下标符访问自己的元素,如 boy[0],boy[1]等.需要注意的是下标从 0 开始,因此,数组若是 7 个元素,下标到 6 为止,如果你将来使用了如下语句将发生异常.

```
boy[7]=384.98f;
```

二维数组也通过下标符访问自己的元素,如 a[0][1],a[1][2]等　需要注意的是下标从 0 开始,比如声明创建了一个二维数组 a

```
int a[][] =new  int[2][3]
```

那么第一个下标的变化范围从 0 到 1,第二个下标变化范围从 0 到 2.如果你将来使用了如下语句将发生异常.

```
a[2][1]=38;a[0][3]=90;
```

例子 1

```
import java.applet.*; import java.awt.*;
public class Example5_1 extends Applet
{  float a[];
```

```
    public void init()
    {  a=new float[5];
       a[0]=23.9f;a[1]=34.9f;a[2]=45f;a[3]=56.98f;a[4]=100f;
    }
    public void paint(Graphics g)
    {
g.drawString("a[0]="+a[0]+"a[1]="+a[1]+"a[2]"+a[2]+"a[3]="+a[3]+"a[4]="+ a[4],12,12);
    }
}
```

注 有一个重要的表示数组长度的 即元素的个数 格式.如,如果创建了 float a[]=new float[6],则 a.length 的值 6.

5.4. 数组的初始化

创建数组后,系统会给每个数组元素一个默认的值,如,**float** 型是 **0.0.**

我们在声明数组时同时也还可以给数组的元素一个初始值,如

```
float boy[ ]={ 12.3f;23.89f,2.0f,23f,578.98f}
```

上述语句相当于

```
float boy[]=new float[5]
```

然后

```
boy[0]=12.3f;boy[1]=23.89f;boy[2]=2.0f;boy[3]=23f;boy[4]=578.98f
```

例如

```
String s[ ]={ "we", "are", "hello", "123", "who? "}
```

5.5. 字符串

Java 使用 java.lang 包中的 String 类来创建一个字符串变量,因此字符串变量是类类型变量,是一个对象.

1 字符串常量 如,"你好","1234.987","weqweo".
2 声明字符串 String s
3 创建字符串 使用 String 类的构造方法.

例如

```
s=new String("we are students")
```

也可写成

```
s= "we are students"
```

声明和创建可用一步完成

```
String  s=new String("we are students");
```

或

```
String  s= "we are students";
```

也可以用一个已创建的字符串创建另一个字符串,如

```
String  tom=String(s);
```

相当于

```
String  tom= "we are students"
```

String 类还有两个较常用构造方法

1　String (char　a[])　用一个字符数组 a 创建一个字符串对象,如

```
char  a[3]={'b','o','y'};
String s=new String(a);
```

上述过程相当于

```
String s= "boy";
```

2　String(char a[],int startIndex,int count)　提取字符数组 a 中的一部分字符创建一个字符串对象,参数 startIndex 和 count 分别指定在 a 中提取字符的起始位置和从该位置开始截取的字符个数,例如

```
char a[]={'s','t','b','u','s','n'};
String s=new String(a,2,3);
```

上述过程相当于

```
String s= "bus";
```

5.1 怎样获取字符串的长度

使用 String 类中的 length()方法可以获取一个字符串的长度,如

```
String s= "we are students",tom= "我们是学生";
int n1,n2;
n1=s.length();
n2=tom.length();
```

那么 n1 的值是 15,n2 的值 5.

字符串常量也可以使用 length()获得长度,如　"你的爱好".length()的值是 4.

5.2 字符串比较

1　equals 方法

字符串对象调用 String 类中的 public boolean equals(String s)方法比较当前字符串对象的实体是否与参数指定的字符串 s 的实体相同.如

```
String tom=new String( "we are students");
String boy=new String( "We are students");
String jerry= new String("we are students");
```

tom.equals(boy)的值是 false,tom.equals(jerry)的值是 true.

注 tom==jerry 的值是 false. 因为字符串是对象, tom, jerry 是引用. 内存示意如图 5.2 所示.

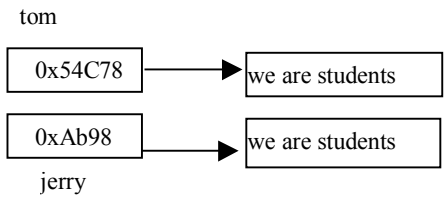


图 5.2 内存示意图

2 equalsIgnoreCase 方法

字符串对象调用 public boolean equalsIgnoreCase(String s) 比较当前字符串对象是否与参数指定的字符串 s 是否相同,比较时忽略大小写.如

```
String  tom =new String("ABC"),
        Jerry=new String("abc");
```

tom.equalsIgnoreCase(boy)的值是 true.

3 startsWith,endsWith 方法

字符串对象调用 public boolean srartsWith(String s)方法,判断当前字符串对象的前缀是否是参数指定的字符串 s,如

```
String tom= "220302620629021",jerry= "21079670924022";
```

tom.startsWith("220")的值是 true jerry.startsWith("220")的值是 false.

可以使用 public boolean endsWith(String s) 方法,判断一个字符串的后缀是否是字符串 s,如

```
String tom= "220302620629021",jerry= "21079670924022";
```

tom.endsWith("021")的值是 true jerry.endsWith("021")的值是 false.

例子 2

```
import java.applet.*;import java.awt.*;
public class Example5_2 extends Applet
{  String tom;
   public void init()
   {  tom="220302620629021";
```

```
    }
    public void paint(Graphics g)
    {  if((tom.startsWith("220302"))&&(tom.endsWith("1")||tom.endsWith("3")))
       g.drawString("tom是吉林人,男性",10,10);
    }
}
```

4　regionMatches 方法

字符串调用

public boolean regionMatches(int firstStart,String other,int ortherStart,int length) 方法,从当前字符串参数 **firstStart** 指定的位置开始处,取长度为 **length** 的一个子串,并将这个子串和参数 **other** 指定的一个子串进行比较,其中,**other** 指定的子串是从参数 **othertStart** 指定的位置开始,从 **other** 中取长度为 **length** 的一个子串.如果两个子串相同该方法就返回 **true**,否则返回 **false.**

使用该方法的重载方法

public boolean regionMatches(boolean b,int firstStart,String other,int ortherStart,int length)

可以通过参数 **b** 决定是否忽略大小写,当 **b** 取 **true** 时,忽略大小写.

在下面的例子 **3** 中,我们判断一个字符串中共出现了几个 **“en”**

例子 3

```
class Example5_3
{   public static void main(String args[])
    {  int number=0;
       String s="student;entropy;engage,english,client";
       for(int k=0;k<s.length();k++)
        {  if(s.regionMatches(k,"en",0,2))
             {  number++;
             }
        }
       System.out.println("number="+number);
    }
}
```

5　compareTo,compareToIgnoreCase 方法

字符串对象可以使用 **String** 类中的 **public int compareTo　String s**　方法,按辞典序与参数 **s** 指定的字符串比较大小.如果当前字符串与 **s** 相同,该方法返回值 **0**　如果当前字符串对象大于 **s**,该方法返回正值　如果小于 **s**,该方法返回负值.例如

```
String str= "abcde"
```

str.compareTo("boy")小于 0　str.compareTo("aba")大于 0　str.compareTo("abcde")等于 0.

按辞典序比较两个字符串还可以使用 public int compareToIgnoreCase(String s)方法,该方法忽略大小写.

下面的例子 4 将一个字符串数组按字典序重新排列.

例子 4

```
class Example5_4
{  public static void main(String args[])
   {  String a[]={"boy","apple","Applet","girl","Hat"};
      for(int i=0;i<a.length-1;i++)
         {for(int j=i+1;j<a.length;j++)
           {  if(a[j].compareTo(a[i])<0)
               {  String temp=a[i];
                  a[i]=a[j];
                  a[j]=temp;
               }
           }
         }
     for(int i=0;i<a.length;i++)
        {  System.out.print("  "+a[i]);
        }
   }
}
```

5.3 字符串检索

1　搜索指定串出现的位置

- **public int indexOf (String s)　字符串调用该方法从当前字符串的头开始检索字符串 s,并返回首次出现 s 的位置.如果没有检索到字符串 s,该方法返回的值是-1.**
- **public int　indexOf(String s ,int startpoint)　字符串调用该方法从当前字符串的 startpoint 位置处开始检索字符串 s,并返回首次出现 s 的位置.如果没有检索到字符串 s,该方法返回的值是-1.**
- **public int lastIndexOf (String s)　字符串调用该方法从当前字符串的头开始检索字符串 s,并返回最后出现 s 的位置.如果没有检索到字符串 s,该方法返回的值是-1.**
- **public int　lastIndexOf(String s ,int startpoint)　字符串调用该方法从当前字符串的 startpoint 位置处开始检索字符串 s,并返回最后出现 s 的位置.如果没有检索到字符串 s,该方法返回的值是-1.**

例如

```
String  tom="I am a good cat"
tom.indexOf("a");//值是 2.
tom.indexOf("good",2);//值是 7.
tom.indexOf("a",7);//值是 13.
tom.indexOf("w",2);//值是-1
```

2 搜索指定字符出现的位置

- **public int indexOf (int char)** 字符串调用该方法从当前字符串的头开始检索字符 char,并返回首次出现 char 的位置.如果没有检索到字符串 s,该方法返回的值是-1.
- **public int indexOf(int char ,int startpoint)** 字符串调用该方法从当前字符串的 startpoint 位置处开始检索字符 char,并返回首次出现 char 的位置.如果没有检索到字符串 s,该方法返回的值是-1.
- **public int lastIndexOf (int char)** 字符串调用该方法从当前字符串的头开始检索字符 char,并返回最后出现 char 的位置.如果没有检索到字符 char,该方法返回的值是-1.
- **public int lastIndexOf(int char ,int startpoint)** 字符串调用该方法从当前字符串的 startpoint 位置处开始检索字符 char,并返回最后出现 char 的位置.如果没有检索到字符 char,该方法返回的值是-1.

5.4 字符串的截取

- **public String substring(int startpoint)** 字符串对象调用该方法获得一个当前字符串的子串,该子串是从当前字符串的 startpoint 处截取到最后所得到的字符串.例如,
- **public String substring(int start ,int end)** 字符串对象调用该方法获得一个当前字符串的子串,该子串是从当前字符串的 start 处截取到 end 处所得到的字符串,但不包括 end 处所对应的字符.

例如

```
String tom="I love tom";
String s=tom.substring(2,5);
s 是 lov
```

5.5 替换

- **public String replace(char oldChar,char newChar)** 字符串对象 s 调用该方法可以获得一个串对象,这个串对象是用参数 newChar 指定的字符替换 s 中由 oldChar 指定的所有字符而得到的字符串.
- **public String replaceAll(String old ,String new)** 字符串对象 s 调用该方法可以获得一个串对象,这个串对象是通过用参数 new 指定的字符串替换 s 中由 old 指定的所有字符串而得到的字符串.
- **Public String trim()** 一个字符串 s 通过调用方法 trim()得到一个字符串对象,该字符

串对象是 s 去掉前后空格后的字符串.

例如

```
String s= "I mist  theep "
Strong temp=s.replace(  t  ,  s  )
```

Temp 是 “I miss sheep".

```
String  s="  I am a student    "
String temp=s.trim();
```

那么 temp 是"I am a student".

5.6 字符串转化为相应的数值

1 转化为整型

java.lang 包中的 Integer 类调用其类方法 public static int parseInt(String s) 可以将 “数字”格式的字符串,如"12387",转化为 int 型数据.例如

```
int x;
String s="6542";
x=Integer.parseInt("6542");
```

类似地,使用 java.lang 包中的 Byte,Short,Long 类调相应的类方法

public static byte parseByte(String s)

public static short parseShort(String s)

public static long parseLong(String s)

可以将 ““数字”格式的字符串,转化为相应的基本数据类型.例如

```
int x;
String s="6542";
x=Integer.parseInt("6542");
```

2 转化为 float 型或 double 型

java.lang 包中的 Float 类调用其类方法

public static int parse Float (String s)

可以将 “数字”格式的字符串,如"12387.8976",转化为 float 型数据.例如

```
float n=Float.parseFloat("12387.8976")
```

或

```
String s= new String(“12387.8976”);
float n=Float.parseFloat(s)
```

类似地,使用 java.lang 包中的 Double 类调相应的类方法

public static double parseDouble(String s)

可以将 “数字”格式的字符串,转化为 double 型数据.

另外,还可以使用 Float 类的类方法 **public static Float valueOf(String s)** 将形如"12334.870"字符串转化为 float 型数据,例如

```
flaot n=Float.valueOf("12334.870").floatValue()
```

使用 Double 类的类方法 **public static Double valueOf(String s)** 将形如"12334.870"字符串转化为 double 型数据,例如

```
double n=Double.valueOf("12334.870").doubleValue()
```

现在我们举一个求若干个数的平均数的例子,若干个数从键盘输入.

例子 5

```
public class Example5_5
{  public static void main(String args[])
   {  double n,sum=0.0 ;
      for(int i=0;i<args.length;i++)
       {  sum=sum+Double.valueOf(args[i]).doubleValue();
       }
     n=sum/args.length;
     System.out.println("平均数:"+n);
   }
}
```

这是一个应用程序,应用程序中的 main 方法中的参数 args 能接受你从键盘键入的字符串.你首先应编译上述源程序

```
c:\2000\ >javac Example5_5.java
```

编译通过后,你要使用解释器 java.exe 来执行你的字节码文件

```
C:\2000\>java Example5_5 "123.78" 23324.9" "213214" (回车)
```

这时,程序中的 args[0],arg[1],arg[2]分别得到字符串"123.78","23324.9" 和"213214". 在源程序中我们再将这些字符串转化为数值进行运算,得到所需的结果.

5.7 数值转化为字符串

可以使用 String 类的下列类方法

public String valueOf byte n

public String valueOf int n

public String valueOf long n

public String valueOf float n

public String valueOf double n

将形如 123,1232.98 等数值转化为字符串,如

```
String str=String.valueOf(12313.9876);
float  x=123.987f;
String temp=String.valueOf(x);
```

5.13 对象的字符串表示

在子类的讲述中我们讲过,所有的类都默认地是 java.lang 包中 Object 类的子类或间接子类.Object 类有一个 public 方法 toString(),一个对象通过调用该方法可以获得该对象的字符串表示.

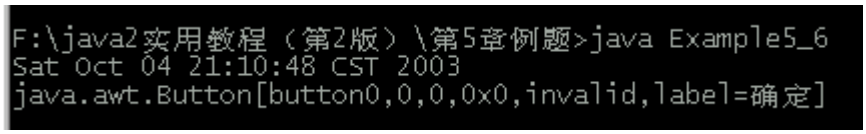


图 5.3 对象的字符串表示

例子 6 效果如图 5.3

```
import java.util.Date;
import java.awt.*;
public class Example5_6
{   public static void main(String args[])
   {  Date date=new Date();
      Button button=new Button("确定");
      System.out.println(date.toString());
      System.out.println(button.toString());
   }
}
```

5.14 使用 StringTokenizer 类分析字符串

有时我们需要分析字符串并将字符串分解成可被独立使用的单词,这些单词叫做语言符号.例如,对于字符串"We are Students ",如果我们把空格作为该字符串的分隔符,那么该字符串有三个单词 语言符号 .而对于字符串"WE,are,Student",如果我们把逗号作为了该字符串的分隔符,那么该字符串有三个单词 语言符号 .

当我们分析一个字符串并将字符串分解成可被独立使用的单词时,可以使用 java.util 包中的 StringTokenizer 类,该类有两个常用的构造方法

- StringTokenizer String s 为字符串 s 构造一个分析器.使用默认的分隔符集合,即空格符 若干个空格被看做一个空格 ,换行符,回车符,Tab 符,进纸符.
- StringTokenizer(String s, String delim) 为字符串 s 构造一个分析器.参数 dilim

中的字符被作为分隔符.

例如

```
StringTokenizer   fenxi=new  StringTokenizer("we  are  student");
StringTokenizer   fenxi=new  StringTokenizer("we ,are  ; student",  ", ; ");
```

我们把一个 StringTokenizer 对象称作一个字符串分析器.一个分析器可以使用 nextToken()方法逐个获取字符串中的语言符号 单词 ,每当调用 nextToken()时,都将在字符串中获得下一个语言符号.通常用 while 循环来逐个获取语言符号,为了控制循环,我们可以使用 StringTokenizer 类中的 hasMoreTokens()方法,只要字符串中还有语言符号,该方法就返回 true,否则返回 false.另外我们还可以调用 countTokens()方法得到字符串一共有多少个语言符号.

下面是一个应用程序,分析字符串,分别输出字符串的单词,并统计出单词个数.

例子 7

```
import java.util.*;
public class Example5_7
{  public static void main(String args[])
   {  String s="I am Geng.X.y,she is my girlfriend";
      StringTokenizer fenxi=new StringTokenizer(s," ,"); //空格和逗号做分
      int number=fenxi.countTokens();
      while(fenxi.hasMoreTokens())
       {  String str=fenxi.nextToken();
          System.out.println(str);
          System.out.println("还剩"+fenxi.countTokens()+"个单词");
       }
     System.out.println("s共有单词  "+number+"个");
   }
}
```

5.15 Character 类

当处理字符串时,Character 类中的一些类方法是很有用的,这些方法可以用来进行字符分类,比如判断一个字符是否是数字字符或改变一个字符大小写等.

- public static boolean isDigit(char ch) 如果 ch 是数字字符方法返回 true,否则返回 false.
- public static boolean isLetter(char ch) 如果 ch 是字母方法返回 true,否则返回 false.
- public static boolean isLetterOrDigit(char ch) 如果 ch 是数字字符或字母方法返回 true,否则返回 false.
- public static boolean isLowerCase(char ch) 如果 ch 是小写字母方法返回 true,否则

返回 false.

- public static boolean isUpperCase(char ch)　如果 ch 是大写字母方法返回 true,否则返回 false.
- public static char toLowerCase(char ch)　返回 ch 的小写形式.
- public static char toUpperCase(char ch)　返回 ch 的大写形式.
- public static boolean isSpaceChar(char ch) 如果 ch 是空格返回 true.

在下面的例子 8 中,我们将一个字符串中的小写字母变成大写字母,并将大写字母变成小写字母.

例子 8

```
import java.util.*;
public class Example5_8
{  public static void main(String args[])
   {  String s=new String("abcABC123");
      System.out.println(s);
      char a[]=s.toCharArray();
      for(int i=0;i<a.length;i++)
       { if(Character.isLowerCase(a[i]))
          { a[i]=Character.toUpperCase(a[i]);
          }
        else if(Character.isUpperCase(a[i]))
          { a[i]=Character.toLowerCase(a[i]);
          }
       }
     s=new String(a);
     System.out.println(s);
   }
}
```

5.16 字符串与字符,字节数组

5.16.1.　字符串与字符数组

1　用字符数组创建字符串对象

String 类中有两个用字符数组创建字符串对象的构造方法

- String(char[]) 该构造方法用指定的字符数组构造一个字符串对象.
- String(char[],int offset,int length) 用指定的字符数组的一部分,即从数组起始位置 offset 开始取 length 个字符构造一个字符串对象.

2　将字符串中的字符拷贝到字符数组

- **public void getChars(int start,int end,char c[],int offset)**　字符串调用 getChars 方法将当前字符串中的一部分字符拷贝到参数 c 指定的数组中,将字符串中从位置 start 到 end-1 位置上的字符拷贝的数组 c 中,并从数组 c 的 offset 处开始存放这些字符.需要注意的是,必须保证数组 c 能容纳下要被拷贝的字符.
  下面的例子 9 具体地说明了该方法的使用.

例子 9

```
class Example5_9
{   public static void main(String args[])
   {  char c[],d[];
      String s="巴西足球队击败德国足球队";
      c=new char[2];
      s.getChars(5,7,c,0);
      System.out.println(c);
      d=new char[s.length()];
      s.getChars(7,12,d,0);
      s.getChars(5,7,d,5);
      s.getChars(0,5,d,7);
      System.out.println(d);
   }
}
```

运行结果

```
击败
德国足球队击败巴西足球队
```

- **public char[] toCharArray()**　字符串对象调用该方法可以初始化一个字符数组,该数组的长度与字符串的长度相等,并将字符串对象的全部字符拷贝到该数组中.
  下面的例子 10 对字符串加密,解密,使用了 toCharArray 方法.

例子 10　效果如图 5.4

```
class Example5_10
{  public static void main(String args[])
   {  String s="清华大学出版社";
      char a[]=s.toCharArray();
      for(int i=0;i<a.length;i++)
        {  a[i]=(char)(a[i]^'t');
        }
    String secret=new String(a); System.out.println("密文:"+secret);
```

图 5.4　加密与解密

```
        for(int i=0;i<a.length;i++)
           {  a[i]=(char)(a[i]^'t');
           }
        String code=new String(a);   System.out.println("原文:"+code);
      }
   }
```

5.16.2. 字符串与字节数组

1 用字节数组创建字符串对象

- **String(byte[])** 该构造方法使用平台默认的字符编码,用指定的字节数组构造一个字符串对象.
- **String(byte[],int offset,int length)** 该构造方法使用平台默认的字符编码,用指定的字节数组的一部分,即从数组起始位置 **offset** 开始取 **length** 个字节构造一个字符串对象.

2 将字符串转化为字节数组

- **public byte[] getBytes()** 使用平台默认的字符编码,将当前字符串转化为一个字节数组.

例子 11

```
public class Example5_11
{  public static void main(String args[])
   {  byte d[]="你我他".getBytes();
      System.out.println("数组 d 的长度是(一个汉字占两个字节):"+d.length);
      String s=new String(d,0,2);
      System.out.println(s);
   }
}
```

上述程序的输出结果

```
数组 d 的长度是(一个汉字占两个字节)  6
你
```

习 题 五

1.使用 String 类的 public String toUpperCase()方法可以将一个字符串中的小写字母变成大写字母 使用 public String toLowerCase()方法可以将一个字符串中的大写字母变成小写字母.编写一个程序,使用这个两个方法实现大小写的转换.

2.使用 String 类的 public String concat(String str)方法可以把调用该方法

的字符串与参数指定的字符串连接,把 **str** 指定的串连接到当前串的尾部获得一个新的串.编写一个程序通过连接两个串得到一个新串,并输出这个新串.

3 String 类的 **public char charAt(int index)** 方法可以得到当前字符串 **index** 位置上的一个字符.说出下列程序的输出结果.

```
class E4
{  public static void main(String args[])
   {  String s="中国科学技术大学";
      char a=s.charAt(2),b=s.charAt(6);
      System.out.print(a);
     System.out.println(b);
   }
}
```

4.

(1) 使用 java.util 包中的 Arrays 类的静态方法:public static void sort(double a[])可以把参数 a 指定的 double 型数组按升序排序.

(2) 使用 java.util 包中的 Arrays 类的静态方法:public static void sort(double a[],int start,int end)可以把参数 a 指定的 double 型数组中从位置 statr 到 end 位置的数按升序排序.说出下列程序的输出结果.

```
import java.util.*;
class E5
{  public static void main(String args[])
   {  int a[]={23,67,89,90,-987}; double b[]={12.89,90.87,34,678.987,-98.78,0.89};
     Arrays.sort(a);Arrays.sort(b,1,4);
     for(int i=0;i<=4;i++)
      {  System.out.print(a[i]+",");
      }
     for(int i=0;i<b.length;i++)
      {  System.out.print(b[i]+",");
      }
   }
}
```

5. 使用 java.lang 包中 System 类的静态方法 arraycopy 可以实现数组的快速拷贝,上机实习下列程序,并总结出 arraycopy 方法参数的使用规则.

```
class ArrayCopy
{  public static void main(String args[])
   {  char a1[]={'a','b','c','d','e','f'},b1[]={'1','2','3','4','5','6'};
      System.arraycopy(a1,0,b1,1,a1.length-1);
      System.out.println(new String(a1)); System.out.println(new String(b1));
```

```
        byte a2[]={97,98,99,100,101,102},b2[]={65,67,68, 69, 70, 71};
        System.arraycopy(b2,0,a2,3,b2.length-3);
        System.out.println(new String(a2)); System.out.println(new String(b2));
    }
}
```

第六章　　时间,日期和数字

6.1.Date 类

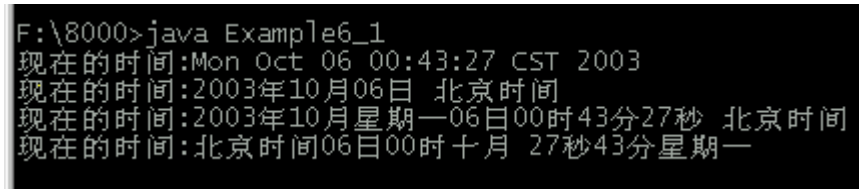


图 6.1　格式化时间

Date 类在 **java.util** 包中.使用 **Date** 类的无参数构造方法创建的对象可以获取本地当前时间.**Date** 对象表示时间的默认顺序是　星期,月,日,小时,分,秒,年.例如

```
Sat Apr 28 21:59:38 CST 2001.
```

我们可能希望按着某种习惯来输出时间,比如时间的顺序

```
年 月 星期 日
```

或

```
年 月 星期 日 小时 分 秒.
```

这时可以使用 **DataFormat** 的子类 **SimpleDateFormat** 来实现时期的格式化.**SimpleDateFormat** 有一个常用构造方法

```
public SimpleDateFormat(String pattern).
```

该构造方法可以用参数 **pattern** 指定的格式创建一个对象,该对象调用

```
format(Data  date)
```

方法格式化时间对象 **date**.需要注意的是,**pattern** 中应当含有一些有效的字符序列.例如

- **y** 或 **yy**　　表示用 **2** 位数字输出年份　**yyyy** 表示用 **4** 为数字输出年份.
- **M** 或 **MM**　表示用 **2** 为数字或文本输出月份,如果想用汉字输出月份,**pattern** 中应连续包含至少 **3** 个 **M**,如　**MMM.**
- **d** 或 **dd**　　表示用 **2** 为数字输出日.
- **H** 或 **HH**　　表示用两位数字输出小时.
- **m** 或 **mm**　　表示用两位数字输出分.
- **s** 或 **ss**　　表示用两位数字输出秒.

- **E** 表示用字符串输出星期.

在下面的例子 1 中,我们用三种格式输出时间.

例子 1 效果如图 6.1

```
import java.util.Date;
import java.text.SimpleDateFormat;
class Example6_1
{  public static void main(String args[])
   {  Date nowTime=new Date();
      System.out.println("现在的时间:"+nowTime);
      SimpleDateFormat matter1=new SimpleDateFormat("yyyy 年 MM 月 dd 日 北京时间");
      System.out.println("现在的时间:"+matter1.format(nowTime));
      SimpleDateFormat matter2=
      new SimpleDateFormat("yyyy 年 MM 月 Edd 日 HH 时 mm 分 ss 秒 北京时间");
      System.out.println("现在的时间:"+matter2.format(nowTime));
      SimpleDateFormat matter3=
      new SimpleDateFormat("北京时间 dd 日 HH 时 MMM ss 秒 mm 分 EE");
      System.out.println("现在的时间:"+matter3.format(nowTime));
   }
}
```

可以用 System 类的静态方法 public long currentTimeMillis()获取系统当前时间,这个时间是从 1970.年 1 月 1 日 0 点到目前时刻所走过的毫秒数 这是一个不小的数 .下面的例子 2 测试一个递归调用所消耗的毫秒数.我们用字符串技术,截取时间的后 8 位.

另外,也可以根据 currentTimeMillis()方法得到的数字,用 Date 的构造方法

```
Date(long time)
```

来创建一个 Date 对象.

例子 2 (效果如图 6.2

```
import java.util.Date;
class Example6_2
{  public static void main(String args[])
   {  long time1=System.currentTimeMillis();
      Date date=new Date(time1);
      System.out.println(date);
      String s=String.valueOf(time1);
      int length=s.length(); s=s.substring(length-8);
      System.out.println(s);
```

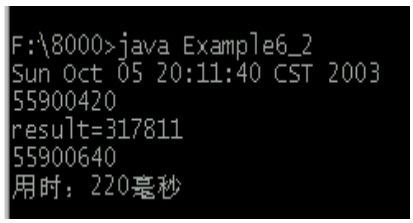


图 6.2 用毫秒表示时间

```
        long result=f(28);
        System.out.println("result="+result);
        long time2=System.currentTimeMillis();//计算 f(28)之后的时间.
        s=String.valueOf(time2);
        length=s.length();  s=s.substring(length-8);
        System.out.println(s);
        System.out.println("用时  "+(time2-time1)+"毫秒");
    }
    public static long f(long n)
    {  long c=0;
      if(n==1||n==2) c=1;
      else if(n>=3) c=f(n-1)+f(n-2);
      return c;
    }
}
```

6.2. Calendar 类

Calendar 类在 java.util 包中.使用 Calendar 类的 static 方法 getInstance()可以初始化一个日历对象,如

```
Calendar  calendar= Calendar.getInstance();
```

然后,calendar 对象可以调用方法

```
public final void set(int year,int month,int date)
public final void set(int year,int month,int date,int hour,int minute)
public final void set(int year,int month, int date, int hour, int minute,int second)
```

将日历翻到任何一个时间,当参数 year 取负数时表示公元前.

calendar 对象调用方法

```
public int get(int field)
```

可以获取有关年份,月份,小时,星期等信息,参数 field 的有效值由 Calendar 的静态常量指定,例如

```
calendar.get(Calendar.MONTH);
```

返回一个整数,如果该整数是 0 表示当前日历是在一月,该整数是 1 表示当前日历是在二月等.

日历对象调用

```
public long getTimeInMillis()
```

可以将时间表示为毫秒.

下面的例子使用 Calendar 来表示时间.并计算了 2003 年和 1962 年之间相隔的天数.

例子 3（效果如图 6.3

```
import java.util.*;
class Example6_3
{  public static void main(String args[])
   {  Calendar calendar=Calendar.getInstance(); //创建一个日历对象.
      calendar.setTime(new Date());              //用当前时间初始化日历时间.
      String 年=String.valueOf(calendar.get(Calendar.YEAR)),
             月=String.valueOf(calendar.get(Calendar.MONTH)+1),
             日=String.valueOf(calendar.get(Calendar.DAY_OF_MONTH)),
             星期=String.valueOf(calendar.get(Calendar.DAY_OF_WEEK)-1);
      int hour=calendar.get(Calendar.HOUR_OF_DAY),
          minute=calendar.get(Calendar.MINUTE),
          second=calendar.get(Calendar.SECOND);
      System.out.println("现在的时间是 ");
      System.out.println(""+年+"年"+月+"月"+日+"日 "+ "星期"+星期);
      System.out.println(""+hour+"时"+minute+"分"+second+"秒");
      calendar.set(1962,5,29);  //将日历翻到 1962 年 6 月 29 日,注意 5 表示六月.
      long time1962=calendar.getTimeInMillis();
      calendar.set(2003,9,5);  //将日历翻到 2003 年 10 月 5 日.9 表示十月.
      long time2003=calendar.getTimeInMillis();
      long 相隔天数=(time2003-time1962)/(1000*60*60*24);
      System.out.println("2003 年 10 月 5 日和 1962 年 6 月 29 日相隔"+相隔天数+"天");
   }
}
```

```
F:\8000>java Example6_3
现在的时间是:
2003年10月5日 星期0
21时58分56秒
2003年10月5日和1962年6月29日相隔15074天
```

图 6.3 使用 Calendar 表示时间

```
F:\8000>java Example6_4
 日 一 二 三 四 五 六
 ** ** ** ** 1  2  3
 4  5  6  7  8  9  10
 11 12 13 14 15 16 17
 18 19 20 21 22 23 24
 25 26 27 28 29 30 31
```

图 6.4 2004 年 1 月日历页

下面的例子 4 输出 2004 年 1 月的日历页.

例子 4（效果如图 6.4

```
import java.util.*;
class Example6_4
```

```
{  public static void main(String args[])
   {  System.out.println(" 日 一 二 三 四 五 六");
      Calendar 日历=Calendar.getInstance(); //创建一个日历对象.
      日历.set(2004,0,1);  //将日历翻到2004年1月1日,注意0表示一月.
      //获取1日是星期几(get方法返回的值是1表示星期日,星期六返回的值是7):
      int 星期几=日历.get(Calendar.DAY_OF_WEEK)-1;
      String a[]=new String[星期几+31];              //存放号码的一维数组
      for(int i=0;i<星期几;i++)
             { a[i]="**";
             }
      for(int i=星期几,n=1;i<星期几+31;i++)
             { if(n<=9)
                  a[i]=String.valueOf(n)+" ";
               else
                  a[i]=String.valueOf(n) ;
               n++;
             }
      //打印数组
     for(int i=0;i<a.length;i++)
      { if(i%7==0)
          {  System.out.println("");       //换行.
          }
        System.out.print(" "+a[i]);
      }
   }
}
```

6.3.Math 类

在编写程序时,可能需要计算一个数的平方根,绝对值,获取一个随机数等等.java.lang 包中的类包含许多用来进行科学计算的类方法,这些方法可以直接通过类名调用.另外,Math 类还有两个静态常量,E 和 PI,它们的值分别是

2.7182828284590452354

和

3.14159265358979323846.

以下是 Math 类常用方法

- public static long abs(double a) 返回 a 的绝对值.
- public static double max(double a,double b) 返回 a,b 的最大值.

- public static double min(double a,double b) 返回 a,b 的最小值.
- public static double random() 产生一个 0 到 1 之间的随机数 不包括 0 和 1 .
- public static double pow(double a,double b) 返回 a 的 b 次幂.
- public static double sqrt(double a) 返回 a 的平方根.
- public static double log(double a) 返回 a 的对数.
- public static double sin(double a) 返回正弦值.
- public static double asin(double a) 返回反正弦值.

有时我们可能需要对输出的数字结果进行必要的格式化,例如,对于 3.14356789,我们希望保留小数位为 3 位,整数部分至少要显示 3 位,即将 3.14356789 格式化为 003.144.

可以使用 java.text 包中的 NumberFormat 类,该类调用类方法

```
public static final NumberFormat getInstance()
```

实例化一个 NumberFormat 对象,该对象调用

```
public final String format(double number)
```

方法可以格式化数字 number.

NumberFormat 类有如下常用方法

- public void setMaximumFractionDigits(int newValue)
- public void setMinimumFractionDigits(int newValue)
- public void setMaximumIntegerDigits(int newValue)
- public void setMinimumIntegerDigits(int newValue)

在下面的例子中我们用一定的格式输出 5 的平方根,通过一个 20 次的循环,每次获取 1 到 8 之间的一个随机数.

例子 5 (效果如图 6.5

```
import java.text.NumberFormat;
class Example6_5
{ public static void main(String args[])
   {  double a=Math.sqrt(5);
      System.out.println("格式化前 "+a);
      NumberFormat f=NumberFormat.getInstance();
      f.setMaximumFractionDigits(5);f.setMinimumIntegerDigits(3);
      String s=f.format(a);
      System.out.println("格式化后 "+s);System.out.println("得到的随机数 ");
      int number=8;
      for(int i=1;i<=20;i++)
       { int randomNumber=(int)(Math.random()*number)+1;//产生 1 到 8 之间的随机数.
         System.out.print(" "+randomNumber);
```

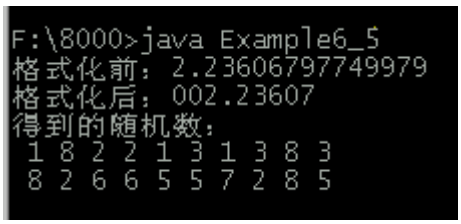


图 6.5 数字计算

```
            if(i%10==0)
                System.out.println("");
        }
    }
}
```

习题六

1 输出某年某月的日历页,通过 main 方法的参数将年份和月份时间传递到程序中.

2 计算某年,某月,某日和某年,某月,某日之间的天数间隔.要求年,月,日通过 main 方法的参数传递到程序中.

3 编程练习 Math 类的常用方法.

第七章　　AWT 工具集简介

通过图形用户界面 GUI raphics User Interface ,用户和程序之间可以方便地进行交互.Java 的抽象窗口工具包 AWT Abstrac Window Toolkit 中包含了许多类来支持 GUI 设计.AWT 由 Java 的 java.awt 提供,该包中有许多用来设计 GUI 的组件类,如 按钮,菜单,列表,文本框等组件类,同时它还包含窗口,面板等容器类.

在学习 GUI 编程时,必须很好地理解掌握两个概念 容器类(Container)和组件类(Component).Java.awt 包中一部分类的层次关系如图 7.1 所示.

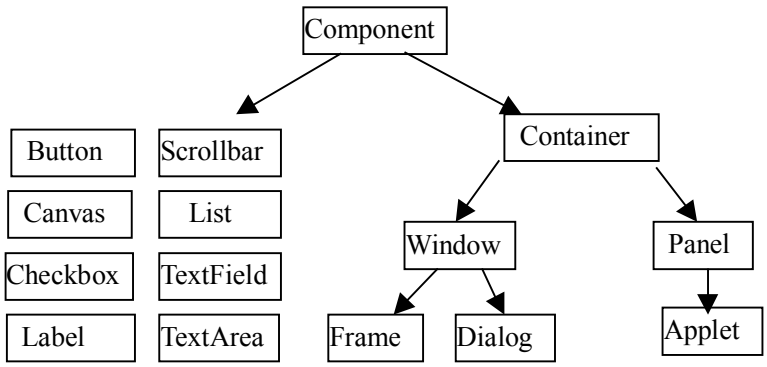


图 7.1　Component 类的部分子类

- Button,Scrollbar,Canvas,List,Checkbox,TextField,TextArea ,Label 类是包 java.awt 中的类,并且是 java.awt 包中的 Component 组件 的子类.Java 称由 Component 类的子类或间接子类创建的对象为一个组件.
- Java 称由 Container 的子类或间接子类创建的对象为一个容器.
- 可以向容器添加组件.Component 类提供了一个 public 方法 add( ),一个容器可以调用这个方法将组件添加到该容器中.
- 容器调用 removeAll()方法可以移掉容器中的全部组件 调用 remove(Component c)方法可以移掉容器中参数指定的组件.
- 每当容器添加新的组件或移掉组件时,应当让容器调用 validate()方法,以保证容器中的组件能正确显示出来.
- 注意到容器本身也是一个组件,因此你可以把一个容器添加到另一个容器中实现容器的嵌套.
- 在上图中需要注意的是 Applet 类不是包 java.awt 中的类,上图只是说明它是 Panel 的子类,是 Container 的间接子类.它是包 java.applet 中的类,不同包中的类可以有继承关系.

现在我们举一个向容器添加组件的应用程序的例子.

例子 (效果如图 7.1)

```
import java.awt.*;
class Example7_1
{ public static void main(String args[])
   { Frame fr=new Frame("媒体新闻");    // 一个容器对象.
     fr.setLayout(new FlowLayout());
     Button button1=new Button("确定 ");
     Button button2=new Button("取消");
     fr.add(button1);
     fr.add(button2);
     fr.setSize(200,300);              //调用方法 setSize(int,int)设置容器的大小.
     fr.setBackground(Color.cyan);
     fr.setVisible(true);
     fr.validate();
   }
}
```

图　7.1 容器,组件

注　Java 小应用程序(Java applet)的主类是 Applet 类的子类,因此 Java applet 本身是一个容器.另外,它通过浏览器运行,具有更加丰富的图形功能.Java applet 是 Java 的重要内容之一.下一章讲授 Java applet.

习 题 七

下列程序中,一共有多少个组件,哪些组件既是组件又是容器

```
import java.awt.*;
class E6
{
  public static void main(String args[])
   {
     Frame fra=new Frame("?");
     fra.setVisible(true);
     fra.setBounds(120,100,200,180);
```

```
        Panel p=new Panel();
        Button b=new Button("java");
        TextField text=new TextField(10);
        Label label=new Label("how are you");
        Checkbox box=new Checkbox("Wa");
        p.add(box);
        p.add(b);
        fra.add(label,"North");
        fra.add(p,"Center");
        fra.add(text,"South");
        fra.validate();
    }
}
```

第八章　　Java Applet 基础

在第一章中我们已经知道,一个 Java Applet 程序中必须有一个类是 Applet 类的子类.称该子类是 Java Applet 的主类,并且主类必须修饰为 public 的.Applet 类是包 java.applet 中的一个类,同时它还是包 java.awt 中 Container 容器 类的子类,因此 Java applet 的主类的实例是一个容器.

我们已经知道,Java Applet 程序通过浏览器来执行,因此它和 Java 应用程序有许多不同之处.

下面我们通过一个例子来说明一个 Java applet 的全过程.

1. 用记事本编写下列源文件

例子 1

```
import java.applet.*;
import java.awt.*;
public class Example8_1 extends Applet
{  Button button1; Button button2;
   int sum;
   public void init()
   {  button1=new Button("yes");
      button2=new Button("No");
      add(button1);
      add(button2);
   }
   public void start()
   {  sum=0;
      for(int i=1;i<=100;i++)
         { sum=sum+i;
         }
   }
   public void stop() { }
   public void destroy(){ }
   public void paint(Graphics g)
   {  g.setColor(Color.blue);
      g.drawString("程序设计方法",20,60);
      g.setColor(Color.red);
      g.drawString("sum="+sum,20,100);
   }
}
```

一个 Java Applet 也是由若干个类组成的,但必须有一个类扩展了 Applet 类,即它是Applet类的子类,Applet类是系统提供的类.我们把这个类叫做这个Java Applet的主类,Java Applet 的主类必须是 public 的.一个 Java Applet 不再需要 main 方法,但必须有且只有一个类扩展了 Applet 类.当我们保存上面的源文件时,必须命名为 Example8_1.java.假设保存Example8_1.java 在 f:\8000 目录下.

2. 编译

```
f:\8000>\javac Example8_1.java
```

编译成功后,文件夹 8000 下会生成一个 Example8_1.class 文件.如果源文件有多个类,将生成多个 class 文件,都和源文件在同一文件夹里.

3. 运行

Java Applet 必须由浏览器来运行,因此我们必须编写一个超文本文件 含有 applet 标记的 web 页 告诉浏览器来运行这个 Java Applet.

下面是一个最简单的一个 html 文件,告诉浏览器运行我们的 Java Apple.我们使用记事本编辑如下一个超文本文件,并保存在 f:\8000 目录下,命名为 Example8_1.html 扩展名必须是 html,主文件名只要符合 Java 标识符规定即可 .

```
<applet code=Example8_1.class height=180 width=300>
</applet>
```

超文本中的标记 <apple …> 和</applet> 告诉浏览器将运行一个 Java Applet,code 告诉浏览器运行哪个 Java Applet.code"="后面是主类的字节码文件.

一个 Java Applet 的执行过程称为这个 Java Applet 的生命周期.一个 Java Applet 的生命周期内涉及如下方法,这些方法也正是一个完整的 Java Applet 所包含的,它们是 init () ,start(),stop(),destroy ,paint(Graphics g) 方法.

我们已经知道类是对象的模板,那么上述 Java Applet 的主类的对象是由谁创建的呢?这些方法又是怎样被调用执行的呢 当浏览器打开超文本文件 Example8_1.html,发现有 applet 标记时,将创建主类 Example8_1 的一个对象,图 8.1 中的灰色部分.它的大小由超文本文件"Example8_1.html"中的 width 和 height 来确定.由于 Applet 类也是 Container 的间接子类,因此,主类的实例也是一个容器,容器有相应的坐标系统,单位是像素,原点是容器的左上角.容器可以使用 add()方法放置组件.

图 8.1 主类的对象

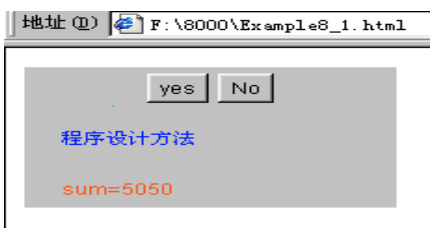


图 8.2 调用 init,start,paint 之后

1 初始化 init()

这个对象首先自动调用 init()方法完成必要的初始化工作.初始化的主要任务是创建所需要的对象,设置初始状态,装载图像,设置参数等.init()方法格式如下

```
public void init()
{ ……
}
```

init()方法只被调用执行一次.

该方法是父类 Applet 中的方法,Example8_1.java 重写了这个方法.

2 启动 :start().

初始化之后,仅接着自动调用 start()方法.在程序的执行过程中,init()方法只被调用执行一次.但 start()方法将多次被自动调用执行.除了进入执行过程时调用方法 start()外,当用户从 applet 所在的 Web 页面转到其它页面,然后又返回时,start()将再次被调用,但不再调用 init()方法.start()方法的格式如下

```
public void start()
{ ……
}
```

该方法是父类 Applet 中的方法,Example8_1.java 重写了这个方法.

3 停止 :stop()

当浏览器离开 Java Applet 所在的页面转到其它页面时,stop()方法被调用.如果浏览器又回到此页,则 start()又被调用来启动 Java Applet.在 Java Applet 的生命周期中,stop()方法也可以被调用多次.如果你在小程序中设计了播放音乐的功能,那么,如果你没有在 stop()方法中给出停止播放它的有关语句,那么当离开此页去浏览其他页时,音乐将不能停止.如果没有定义 stop()方法,当用户离开 Java Applet 所在的页面时,Java Applet 将继续使用系统的资源.若定义了 stop()方法,则可以挂起 applet 的执行.stop()方法的格式为

```
public void stop()
{ ……
}
```

该方法是父类 Applet 中的方法,Example8_1.java 重写了这个方法.

4 删除 destroy()

当浏览器结束浏览时,执行 destroy()方法,结束 applet 的生命.

该方法是父类 Applet 中的方法,不必重写这个方法,直接继承即可.

5 描绘 paint(Graphics g)

paint(Graphics g)方法可以使一个 applet 在屏幕上显示某些信息,如文字,色彩,背景或

图像等.在 applet 的生命周期内可能多次地被随机的调用.例如,当 applet 被其它页面遮挡,然后又重新放到最前面,改变浏览器窗口的大小,以及 applet 本身需要显示信息时,paint()方法都会被自动调用.

与上述 4 种方法不同的是,paint()方法有一个参数 g,浏览器的 Java 运行环境产生一个 Graphics 类的实例,并传递给方法 paint 中的参数 g ,因此,你不妨就把 g 理解为一个画笔.

该方法是 Component 中的方法,Example8_1.java 重写了这个方法.

主类创建的容器对象调用 init,start,paint 方法之后,出现如图 8.2 的效果.

6 关于 repaint()方法

当你使用 repaint()方法时,将导致下列事情发生

程序首先清除 paint()方法以前所画的内容,然后再调用 paint()方法.

在下面的例子中,我们在 paint()方法中,使用了 repaint()方法,因此每当小程序调用 paint()方法时,将会导致 paint()方法以前所画的内容消失,并紧接着再调用 paint()方法.小程序中的字符串不断地往下走.

该方法是 Component 中的方法,Example8_1.java 继承了这个方法

例子 2

```
import java.applet.*;import java.awt.*;
public class Example8_3 extends Applet
{  int x;
   public void init()
   {  x=5;
   }
   public void paint(Graphics g)
   {  x=x+1;
      if(x>=200)
        x=5;
      g.drawString("我们在学习 repaint 方法",20,x);
      repaint();
   }
}
```

习 题 八

1.查阅编写网页的有关书籍,总结出在网页中加入小应用程序的更多技术细节.

2 我们可以在超文本中使用若干个<Param...>标志把值传递到小程序中,例如

```
<param name=名字串  value=值串>
```

写出下列小程序的输出结果,小程序和运行该小程序的超文本文件如下

E2.html

```
<applet code=E2.class width=200 height=200>
<Param name="girl"  value ="160">
<Param name="boy"  value ="175">
</applet>
```

E2.java

```
import java.awt.*;
import java.applet.*;
public class E2 extends Applet
{  int x=8,y=9;
   public void init()
   {  String s1=getParameter("girl");//从html得到"girl"的值  字符串类型  .
      String s2=getParameter("boy");//从html得到"boy"的值  字符串类型  .
      x=Integer.parseInt(s1);
      y=Integer.parseInt(s2);
   }
   public void paint(Graphics g)
   { g.drawString("x="+x+","+"y="+y,90,120);
   }
}
```

第九章　　文本框和文本区

在第六章介绍了组件和容器的概念,第七章介绍了 Java Applet 的基本概念.我们已经知道,Java Applet 本身也是一个容器　准确地说　Java Applet 的主类的实例是一个容器　,因此 Java applet 可以添加交互组件,如文本框,文本区,按钮,滚动列表等.本章学习文本框和文本区.

9.1. 文本框

文本框可以输入单行的文本,你肯定对它不陌生.java.awt 包中的类 TextField 类是专门用来建立文本框的,即 TextField 创建的一个对象就是一个文本框.

TextField 类有下列主要方法

1. TextField()　如果使用这个构造方法创建文本框对象,则文本框的长度为一个字符长,可以在文本框中输入若干个字符.
2. TextField(int x)　如果使用这个构造方法创建文本框对象,则文本框的长度为 x 个字符长,可以在文本框中输入若干个字符.
3. TextField(String s)　如果使用这个构造方法创建文本框对象,则文本框的初始字符串为 s,可以在文本框中输入若干个字符.
4. TextField(String s, int x)　如果使用这个构造方法创建文本框对象,则文本框的初始字符串为 s,文本框的长为 x,可以在文本框中输入若干个字符.
5. public void setText(String s)　文本框对象调用该方法可以设置文本框中的文本为参数 s 指定的文本,文本框中先前的文本将被清除.
6. public String getText()　文本框对象调用该方法可以获取文本框中的文本.
7. public void setEchoChar(char c)　文本框对象调用该方法可以设置文本框的回显字符,这样当用户在文本框中进行文字输入时,在文本框中只显示参数 c 指定的字符.
8. public void setEditable(boolean b)　文本框对象调用该方法可以指定文本框的可编辑性.创建的文本框本默认是为可编辑的.
9. public void addActionListener(ActionListener)　文本框对象调用该方法可以向文本框增加动作监视器　将监视器注册到文本框　.
10. public void removeActionListener(ActionListener)　文本框对象调用该方法可以移去文本框上的动作监视器.

下面的例子 1 中有三个文本框.

例子 1 (效果如图 9.1)

```
import java.applet.*;import java.awt.*;
public class Boy extends Applet
```

```
{  TextField text1,text2,text3;
   public void init()
   {  text1=new TextField("输入密码  ",10);
      text1.setEditable(false);
      text2=new TextField(10);
      text2.setEchoChar('*');
      text3=new TextField("我是一个文本框",20);
      add(text1);add(text2);add(text3);
      text3.setText("重新设置了文本");
   }
}
```

图 9.1　文本框组件

9.2. 文本框上的 ActionEvent 事件

当用户在文本框中键入文本后按回车键,单击按钮,在一个下拉式列表表中选择一个条目等都发生界面事件.我们有时需对发生的事件作出反应,来实现特定的任务,例如,用户单击一个名字叫"确定"或名字叫"取消"的按钮,程序可能将作出不同的处理.在学习处理事件时,必须很好地掌握事件源,监视器,处理事件的接口这三个概念.在这一节,通过处理文本框这个具体的组件上的事件,来掌握处理事件的基本原理.

1　事件源

能够产生事件的对象都可以成为事件源,如文本框,按钮,下拉式列表等.也就是说,事件源必须是一个对象,而且这个对象必须是 Java 认为能够发生事件的对象.

2　监视器

我们需要一个对象对事件源进行监视,以便对发生的事件作出处理.事件源通过调用相应的方法将某个对象作为自己的监视器.例如,对于文本框,这个方法是

addActionListener(监视器)

对于获取了监视器的文本框对象,在文本框获得输入焦点之后,如果用户按回车键,Java 运行系统就自动用 ActionEvent 类创建了一个对象,即发生了 ActionEvent 事件.也就是说,事件源获得监视器之后,相应的操作就会导致事件的发生,并通知监视器,监视器就会作出相应的处理.

3　处理事件的接口

监视器负责处理事件源发生的事件.监视器是一个对象,为了处理事件源发生的事件,监视器这个对象会自动调用一个方法来处理事件.那么监视器去调用哪个方法呢　我们已经知道,对象可以使用创建它的那个类中的方法,那么它到底调用该类中的哪个方法呢　Java 规定　为了让监视器这个对象能对事件源发生的事件进行处理,创建该监视器对象的类必须声明实现相应的接口,即必须在类体中给出该接口中所有方法的方法体,那么当事件源发生事件

时,监视器就自动调用执行被类实现的某个接口方法.

java.awt.event 包中提供了许多事件类和处理各种事件的接口.对于文本框这个接口的名字是 ActionListener,这个接口中只有一个方法

```
public void  actionPerformed(ActinEvent  e)
```

当在文本框中输入字符并回车时,java.awt.envent 包中的 ActionEvent 类自动创建了一个事件对象,并将它传递给方法 actionPerformed(ActionEvent e)中的参数 e,监视器将自动调用方法

```
actionPerformed(ActinEvent e)
```

对发生的事件作出处理.

所以,我们称文本框这个事件源可以发生 ActionEvent 类型事件.为了能监视到这种类型的事件,事件源必须使用 addActionListener 方法获得监视器 创建监视器的类必须实现接口 ActionListener.只要学会了处理文本框这个组件上的事件,其它事件源上的事件的处理也就很容易学会,所不同的是事件源能发生的事件类型的不同,所使用的接口不同而已.

现在,我们将文本框对象上的事件处理总结如下

a 对于文本框事件源,可以发生 ActionEvent 事件

文本框获得监视器之后,在文本框获得输入焦点之后按回车,java.awt.envent 包中的 ActionEvent 类自动创建了一个事件对象.

b 文本框获得监视器

发生 ActionEvent 事件的事件源获得监视器的方法是

```
addActionListener(监视器).
```

由于文本框可以发生 ActionEvent 事件,所以 TextField 类提供了 addActionListener 方法.

c 处理事件的接口

处理发生 ActionEvent 事件的接口是 ActionListener,该接口中只有一个的方法

```
actionPerformer(ActinEvent e).
```

文本框可以发生 ActionEvent 事件,因此,创建文本框的监视器的类必须要实现 ActionListener 接口.当在文本框中回车时,java.awt.envent 包中的 ActionEvent 类自动创建了一个事件对象,这个对象将自动传递给方法 actionPerformed(ActionEvent e)中的参数 e,监视器将自动调用方法 actionPerformed(ActinEvent e)对发生的事件作出处理.

(d) ActionEvent 类中的方法

public Object getSource() ActionEvent 对象调用方法 getSource()可以获取发生 ActionEvent 事件的事件源对象的引用.

public String getActionCommand() ActionEvent 对象调用方法该方法可以获取发生

ActionEvent 事件时,和该事件相关的一个命令字符串,对于文本框,当发生 ActionEvent 事件时,文本框中的文本字符串就是和该事件相关的一个命令字符串.

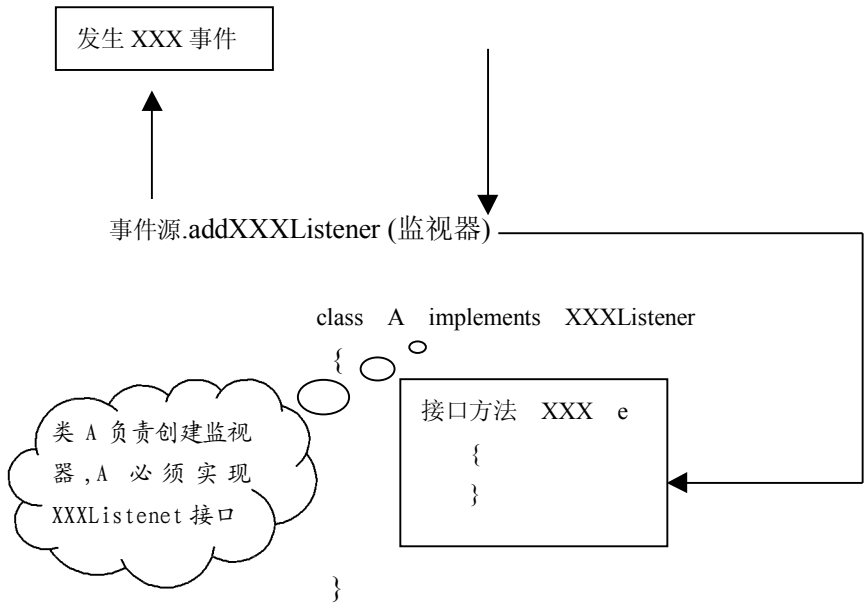


图 9.2 处理事件示意图

Java 事件处理就是基于这种授权模式,即发生相应事件的事件源对象,比如 sourceObjcet,通过调用相应的方法

sourceObjcet.addXXXListener(监视器);

将某个对象作为自己的监视器.

创建监视器对象的类必须实现相应的事件接口

XXXListener

当事件源发生事件时,监视器将调用接口中相应的方法作出处理.事件的处理过程如图 9.2 所示.

现在我们看一个例子,例子想达到的目的是在文本框 text1 中输入英文单词并回车 发生 ActionEvent 事件 ,监视器负责在另一个文本框 text3 中立刻显示汉语意思 事件处理结果 在文本框 text2 中输入汉语单词并回车 发生 ActionEvent 事件 ,监视器负责在另一个文本框 text3 中立刻显示英文意思 事件处理结果 .

例子 2 (效果如图 9.3)

```
import java.applet.*;import java.awt.*;import java.awt.event.*;
public class Example9_2 extends Applet implements ActionListener
{  TextField text1,text2,text3;
```

```
public void init()
 {  text1=new TextField(10);
    text2=new TextField(10);
    text3=new TextField(20);
    add(text1);add(text2);add(text3);
    text1.addActionListener(this); //将主类的实例作为 text1 的监视器,
                                   //因此主类必须实现接口 ActionListener .
     text2.addActionListener(this);
 }
 public void actionPerformed(ActionEvent e)
 {  if(e.getSource()==text1)
    {  String word=text1.getText();
       if(word.equals("boy"))
         {  text3.setText("男孩");
         }
       else if (word.equals("girl"))
         {  text3.setText("女孩");
         }
       else if (word.equals("sun"))
        {  text3.setText("太阳");
        }
      else
        {text3.setText("没有该单词");
        }
    }
  else if(e.getSource()==text2)
    {  String word=text2.getText();
       if(word.equals("男孩"))
       {  text3.setText("boy");
       }
       else if (word.equals("女孩"))
        {  text3.setText("girl");
        }
      else if (word.equals("太阳"))
       {  text3.setText("sun");
       }
     else
      {  text3.setText("没有该单词");
      }
```

图 9.3　处理文本框事件

```
        }
    }
}
```

代码分析

事件源发生的事件传递到监视对象,这意味着你要把监视器连接到文本框.当事件发生时,监视器对象将"监视"它.在上述例子 2 中,通过把主类的实例　这个实例是浏览器创建的　连接到文本框,使它成为监视器.我们使用文本框的 addActionListener 方法完成这个任务.但要说明的是　你想让主类的实例成为文本框的监视器,需要把这个实例作为参数传递到 addActionListener.如何完成这个任务呢　利用 this 关键字,能够完成这个任务.this 代表目前所用的对象,这句话是什么意思呢?

```
text1.addActionListener(this);
```

是在 init()方法中,那么,哪个对象调用 init()这个方法,上述的 this 就表示那个对象.主类的对象自动调用 init()方法,因此 this 就代表主类的对象.

在上述例子 2 中,语句

```
text1.addActionListener(this)
```

将小程序,即 Example9_2 类的实例做为事件源 text1 的监视器,因此 Example9_2 必须实现相应的接口,例子 2 中,因为事件源 text1 发生的事件是 ActionEvent 类型,所以要实现接口 ActionListener

```
public class Example9_2 extends Applet implements ActionListener
```

在 text1 中输入文本回车之后,java.awt.envent 中的 ActionEvent 类创建一个事件对象,并将它传递给方法 public void actionPerformed(ActionEvent e)中的参数 e.监视器就会知道所发生的事件,这时接口中的方法

```
public void actionPerformed(ActionEvent e)
```

会被自动地执行,对所发生的事件作出处理.

在下面的例子 3 中,text1 有两个监视器.当在 text1 中输入一个数字字符串之后,一个监视器负责计算这个数的平方,并将结果放入 text2 中　另一个监视器负责计算这个数的立方,并将结果放入 text3 中,

例子 3 (效果如图 9.4)

```
import java.applet.*;import java.awt.*;import java.awt.event.*;
public class Example9_3 extends Applet implements ActionListener
{  TextField text1,text2,text3;
   PoliceMan police;
   public void init()
   {  text1=new TextField(10);
```

```
        text2=new TextField(10);
        text3=new TextField(10);
        police=new PoliceMan(this);
        add(text1);add(text2);add(text3);
        text1.addActionListener(this);
        text1.addActionListener(police);
    }
    public void actionPerformed(ActionEvent e)
    {  String number=e.getActionCommand();
        int n=Integer.parseInt(number);
        int m=n*n;text2.setText(n+"的平方是:"+m);
    }
}
class PoliceMan implements ActionListener
{  Example9_3 a=null;
    PoliceMan(Example9_3 a)
    {  this.a=a;
    }
    public void actionPerformed(ActionEvent e)
    {  String number=e.getActionCommand();
        int n=Integer.parseInt(number);
        int m=n*n*n;a.text3.setText(n+"的立方是:"+m);
    }
}
```

注 如果用户没有在文本框中输入任何字符,文本框调用 getText 方法将返回一个长度为 0 的字符串,即不含有任何字符的字符串 "".可以通过 getText 方法返回的字符串的长度来判断用户是否输入了字符.

Java 中处理事件就是基于这种授权模式,因此领会了上述两个例子,对学习事件处理就不会有太大的困难了.所不同的,仅仅是处理相应的事件类型使用相应的接口和相应的注册监视器的方法.在今后几章的学习中会自然的掌握.

9.3. 文本区

文本区框可以输入多行的文本,你肯定对它不陌生.**java.awt** 包中的类 **TextArea** 类是专门用来建立文本区的,即 **TextArea** 创建的一个对象称做一个文本区.

TextArea 类有下列主要方法

1 TextArea() 使用这个构造方法创建文本区对象,则文本区的列数和行数取默认值.文本区有水平和垂直滚动条.

2　TextArea(String s)　使用这个构造方法创建文本区对象,则文本区的初始字符串为 s.文本区有水平和垂直滚动条.

3　TextArea(int x,int y)　使用这个构造方法创建文本区对象,文本框行数为 y,列数为 x.文本区有水平和垂直滚动条.

4　TextArea(String s, int x,int y)　如果使用这个构造方法创建文本区对象,则文区的初始字符串为 s,文本框行数为 y,列数为 x.文本区有水平和垂直滚动条.

5　TextArea(String s, int x,int y,int scrollbar)　如果使用这个构造方法创建文本区对象,则文区的初始字符串为 s,文本框行数为 y,列数为 x.scrollbar 取值

```
TextArea.SCROLLBARS_BOTH
TextArea.SCROLLBARS_VERTICAL_ONLY
TextArea.SCROLLBARS_HORIZONTAL_ONLY
TextArea.SCROLLBARS_NONE
```

控制文本区滚动条的显示状态.

6　public void setText(String s)　文本区对象调用该方法可以将文本区中的文本设置为参数 s 指定的文本,文本区中先前的文本将被清除.

7　public String getText()　文本区对象调用该方法可以获取文本区中的文本.

8　public void setEditable(boolean b)　文本区对象调用该方法可以指定文本区的可编辑性.文本区默认是为可编辑的.

9　public boolean　isEditable(boolean b)　文本区对象调用该方法可以获取文本区是否是可编辑的,当文本区是可编辑时,该方法返回 true,否则返回 false.

10　public void insert(String s int x)　文本区对象调用该方法可以在指定位置 x 处,插入指定文本 s.x 是指距文本区开始处字符的个数,x 不能大于文本区中字符的个数.

11　public void replaceRange(String　s,int start,int end)　文本区对象调用该方法可以用给定新文本 s 替换从指定位置 start 开始到指定位置 end 结束的文本,start 和 end 不能大于文本区中字符的个数.

12　public void append(String s)　文本区对象调用该方法可以在文本区中尾加文本

13　int getCaretPosition　文本区对象调用该方法可以获取文本区中输入光标的位置.

14　public void setCaretPosition(int position)　文本区对象调用该方法可以设置文本区中输入光标的位置,其中 position 不能大于文本区中字符的个数.

15　String getSelectedText　文本区对象调用该方法可以获取文本区中选中的文本,例如,通过拖动鼠标选中的文本.

16　public int getSelectionStart　文本区对象调用该方法可以获取被选中的文本的开始位置.

17　public int getSelectionEnd　文本区对象调用该方法可以获取被选中的文本的结束位置.

18　public void setSelectionStart(int n)　文本区对象调用该方法可以设置文本区中被选中的文本的开始位置,其中 n 不能大于文本区中字符的个数.

20　public void setSelectionStart(int n)　文本区对象调用该方法可以设置文本区中被选中的文本的结束位置,其中 n 不能大于文本区中字符的个数.

21　public void selectAll()　文本区对象调用该方法选中文本中的全部文本.

22　addTextListener(TextListener)　文本区对象调用该方法可以向文本框增加文本监视器.

23　removeTextListener(TextListener)　文本框对象调用该方法可以移去文本框上的文本监视器.

下面的例子 4 显示了文本区的一些常用方法.

例子 4 (效果如图 9.5)

```
import java.applet.*;import java.awt.*;
public class Example9_4 extends Applet
{  TextArea text1,text2;
   public void init()
   {  text1=new TextArea("我是学生",6,16);
      text2=new TextArea(6,16);
      add(text1);add(text2);
      text2.append("我在学习 java 语言");
      text1.insert("们",1);
      text1.selectAll();
      int length=text2.getText().length();
      text2.setSelectionStart(2);
      text2.setSelectionEnd(length);
   }
}
```

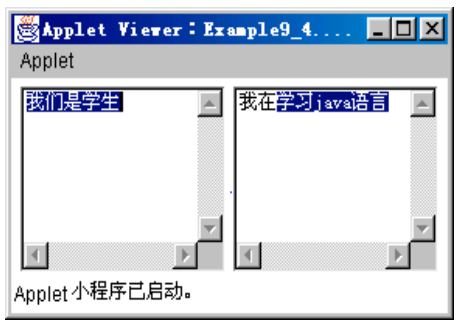


图 9.4　文本区组件

9.4. 文本区上的 TextEvent 事件

文本区可以发生 TextEvent 事件.当文本区中的内容发生变化时,例如　键入字符,删除字符时,都会导致文本区中的内容发生变化,这时,TextEvent 类将自动创建一个事件对象.

现在,我们将文本区对象上的事件处理总结如下

a　TextEvent 事件

对于文本区事件源,可以发生 TextEvent 事件.当文本区获得监视器之后,在文本区域中改变文本的内容,如键入字符,删除字符时,TextEvent 类将自动为创建一个事件对象.

b　文本区获得监视器

发生 TextEvent 事件的事件源获得监视器的方法是　addTextListener(监视器).由于文

本区可以发生 TextEvent 事件,所以 TextArea 类提供了 addTextListener 方法.

c　处理事件的接口

处理发生 TextEvent 事件的接口是 TextListener,该接口中只有一个的方法

textValueChanged (TextEvent e).

创建文本区的监视器的类必须要实现 TextListener 接口.当文本区发生 TextEvent 事件时,监视器将自动调用方法

textValueChanged (TextEvent e)

对发生的事件作出处理.

d　TextEvent 类中的方法

public Object getSource()　获取发生 TextEvent 事件的事件源对象的引用.

在下面的例子 6 中,有两个文本区.当我们在一个文本区中输入若干英文单词时　用空格,逗号或回车做为单词之间的分隔符　,另一个文本区同时对你输入的英文单词按字典序排序,也就是说随着你输入的变化,另一个文本区不断地更新排序.这里我们要用到第五章讲过的 StringTokenizer 类　字符串解析器　.

例子 5 (效果如图 9.6)

```
import java.util.*;import java.applet.*;
import java.awt.*;import java.awt.event.*;
public class Example9_5 extends Applet implements TextListener
{  TextArea text1,text2;
   public void init()
   {  text1=new TextArea(6,15);
      text2=new TextArea(6,15);
      add(text1);add(text2);
      text2.setEditable(false);
      text1.addTextListener(this) ;
   }
 public void textValueChanged(TextEvent e)
   {  if(e.getSource()==text1)
      {  String s=text1.getText();
         StringTokenizer fenxi=new StringTokenizer(s," ,'\n'");
         int n=fenxi.countTokens();
         String a[]=new String[n];
         for(int i=0;i<=n-1;i++)
           {  String temp=fenxi.nextToken();
              a[i]=temp;
```

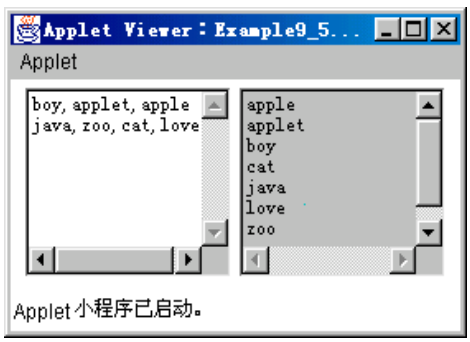


图 9.4　处理文本区事件

```
            }
          for(int i=0;i<=n-1;i++)                    //按字典序从小到大排序.
            {  for(int j=i+1;j<=n-1;j++)
              {  if(a[j].compareTo(a[i])<0)
                   {  String t=a[j]; a[j]=a[i]; a[i]=t;
                   }
              }
            }
          text2.setText(null); //刷新显示.
          for(int i=0;i<n;i++)
             {  text2.append(a[i]+"\n");
             }
        }
    }
}
```

习题九

1　编写有两个文本区的小应用程序.当我们在一个文本区中输入若干个数时,另一个文本区同时对你输入的数进行求和运算并求出平均值,也就是说随着你输入的变化,另一个文本区不断地更新求和及平均值.

2　在下列程序中,当在文本框 **text1** 中输入单词 **Glad** 后按一次回车键,程序的执行会出现怎样的结果　按两次回车键会出现怎样的结果

```
import java.applet.*;
import java.awt.*;import java.awt.event.*;
public class E3 extends Applet implements ActionListener,TextListener
{  TextField text1; TextArea text2;
   public void init()
   {  text1=new TextField(12);text2=new TextArea(6,16);
      add(text1);add(text2);
      text1.addActionListener(this) ;text2.addTextListener(this) ;
   }
public void actionPerformed(ActionEvent e)
   {  if(e.getSource()==text1)
      { text2.append("\n"+text1.getText());
      }
   }
public void textValueChanged(TextEvent e)
```

```
    {  if(e.getSource()==text2)
        {  text1.setText("你好  ");
        }
    }
}
```

4 所谓异常就是程序运行时可能出现一些错误,比如试图打开一个根本不存在的文件等.我们没有单独列出一章来讲述异常处理,而是将 Java 程序中出现的异常问题分散到相应的章节中.当在文本框读取数字时,使用了如下语句

```
double a=Double.valueOf(text_1.getText()).doubleValue();
```

那么客户如果在文本框 text_1 中输入了非数字字母,就无法转化成数字.这时程序的运行就出现 NumberFormatException 异常.Java 规定,应当把可能出现异常的操作放在

```
try{}
catch(){}
```

块语句中.比如

```
try{a=Double.valueOf(text_1.getText()).doubleValue();
   }
catch(NumberFormatException event) {}
```

当出现异常时,就会执行 catch 块中的语句,如果此块中无语句,就会执行相应的默认处理语句.实习下列程序,注意观察在 text_1 中输入 25 按回车键与输入 abc 按回车键程序的运行结果.

```
import java.applet.*;import java.awt.*;import java.awt.event.*;
public class E4 extends Applet implements ActionListener
{  TextField text_1=null,text_2=null;
   public void init()
   {  text_1=new TextField(18);text_2=new TextField(18);
      text_1.addActionListener(this);
      add(new Label("输入一个数字后,按 Enter"));
      add(text_1);add(text_2);
   }
   public void actionPerformed(ActionEvent e)
   {  double a=0.0;
      text_2.setText(null);
      try
      {  a=Double.valueOf(text_1.getText()).doubleValue();
         text_1.setText(""+Math.sqrt(a));
```

```
        }
    catch(NumberFormatException event)
        { text_2.setText("请输入数字字符");
        }
    }
}
```

5.文本区可以使用 **getSelectedText()**方法获取该文本区通过拖动鼠标选中的文本.上机练习习题 **4**,要求在 **text1** 中输入一篇英文短文,然后用鼠标把英文短文中的动词选出来放入另一个文本区.

```
import java.applet.*;
import java.awt.*;import java.awt.event.*;
public class SelectText extends Applet implements ActionListener
{  TextArea text1,text2;
   Button b=new Button("确定");
   public void init()
   {  text1=new TextArea(10,10);
      text2=new TextArea(10,10);
      add(text1);add(text2);
      add(b);
      b.addActionListener(this);
      text2.setEditable(false);
   }
   public void actionPerformed(ActionEvent e)
   {  text2.append("\n"+text1.getSelectedText());
   }
}
```

第十章　　按钮与标签

10.1 按钮

Java.awt 包中的 Button 类是专门用来建立按钮的.即 Button 类创建的一个对象就是一个按钮.

Button 类有下列常用的方法

1　Button()　使用这个构造方法创建按钮,按钮没有名称.

2　Button(String s)　使用这个构造方法创建按钮对象,则按钮上的名称是字符串 s.

3　public void　setLabel(String s)　按钮对象调用该方法可以设置按钮上的名称.

4　public String　getLabel()　按钮对象调用该方法可以获取按钮上的名称.

5　public void　addActionListener(ActionListener)　按钮对象调用该方法可以向按钮增加动作监视器.

6　public void removeActionListener(ActionListener)　按钮对象调用该方法可以移去按钮上的动作监视器.

7　public setActionCommand(String command)　按钮对象调用该方法可以设置按钮发生 ActionEvent 事件相关的字符串命令,默认地字符串命令是按钮上的名称.

ActionEvent 事件

按钮可以发生 ActionEvent 事件,当按钮获得监视器之后,用鼠标单击按钮,就发生 ActionEven 事件,即 java.awt.envent 包中的 ActionEvent 类自动创建了一个事件对象.按钮上的事件处理和文本框类似,所不同是,当发生 ActionEvent 事件时,该事件对象调用 getActionCommand 方法返回的命令名默认地是按钮上的名称,对于文本框,发生 ActionEvent 事件时,事件调用 getActionCommand 方法返回的是文本框中的文本.

下面的例子 1 中有一个文本框 text 和两个按钮 buttonEnter,buttonQuit.在文本框中输入数字回车或单击按钮 buttonEnter,文本框 text 将显示这个数的平方根,如果单击按钮 buttonQuit,就将 text 中的数字设置为 0.

例子 1　效果如图 10.1

```
import java.applet.*;import java.awt.*;import java.awt.event.*;
public class Example10_1 extends Applet implements ActionListener
{  TextField text;
   Button buttonEnter,buttonQuit;
   public void init(){
   {  text=new TextField("0",10); add(text);
      buttonEnter=new Button("确定"); buttonQuit =new Button("清除");
      add(buttonEnter); add(buttonQuit);
```

```
        buttonEnter.addActionListener(this);
        buttonQuit.addActionListener(this);
        text.addActionListener(this);
    }
  public void paint(Graphics g)
    {  g.drawString("在文本框输入数字字符回车或单击按钮",10,100);
       g.drawString("第文本框显示该数的平方根",10,120);
    }
    public void actionPerformed(ActionEvent e)
    {  if(e.getSource()==buttonEnter||e.getSource()==text)
          {  double number=0;
             try {  number=Double.valueOf(text.getText()).doubleValue();
                    text.setText(""+Math.sqrt(number));
                 }
             catch(NumberFormatException event)
                 {  text.setText("请输入数字字符");
                 }
          }
       else if(e.getSource()==buttonQuit)
          {  text.setText("0");
          }
    }
}
```

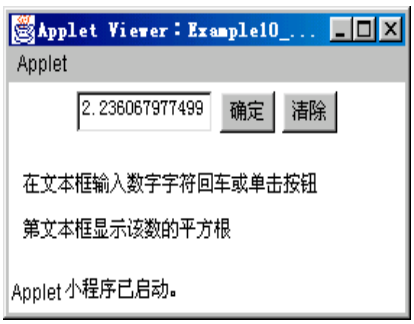


图 10.1　处理 ActionEvent 事件

10.2 扩展按钮

编写一个 Button 类的子类,以便增加一些新的属性和功能,在子类中增加新的成员变量,比如 Button 子类有文本框,文本区等成员变量.

在下面的例子 2 中,MyButton 类是 Button 的子类,MyButton 创建的对象有文本区 text1,text2 等成员变量.MyButton 类实现 ActionListener 和 TextListener 接口,以便监视按钮对象以及 text1 对象.当单击程序中的按钮对象时,将 text1 中的文本用"异或"加密,然后放置到 text2 中.当在 text1 中编辑文本时,text2 中显示 text1 中文本的倒置文本.

例子 2　效果如图 10.2

```
import java.awt.*;import java.applet.*;import java.awt.event.*;
//写一个按扭类的子类,增加一些新的功能:
class MyButton extends Button implements ActionListener,TextListener
{  TextArea text1,text2;       //类的成员变量.
   char save[];
   MyButton(String s,Container con)
```

```
    {  super(s);
       text1=new TextArea(6,12);  text2=new TextArea(6,12);
       text2.setEditable(false);
       text1.addTextListener(this);  //创建的按扭监视其中一个文本区.
       this.addActionListener(this); //创建的按扭自己监视自己.
       con.add(text1);con.add(text2);con.add(this);
    }
    public void textValueChanged(TextEvent e) //实现接口.
    {
    String s=text1.getText();
    StringBuffer strbuffer=new StringBuffer(s);
    String  temp=new String(strbuffer.reverse());
    text2.setText(temp);
    }
    public void actionPerformed(ActionEvent e) //实现接口.
    {  text2.setText(null);
       String s=text1.getText();
      int length=s.length();
      save=new char[length];
      //将字符串拷贝到数组 save
      s.getChars(0,length,save,0);
      for(int i=0;i<save.length;i++)
        {  save[i]=(char)(save[i]^'你');
        }
      String temp=new String(save);
      text2.setText(temp);
    }
}
public class Example10_2 extends Applet implements ActionListener
{  MyButton mybutton;
   Button button;
   public void init()
   {  mybutton=new MyButton("加密",this);
      button=new Button("解密");
      button.addActionListener(this);
      add(button);
   }
   public void actionPerformed(ActionEvent e) //实现接口.
   {  for(int i=0;i<mybutton.save.length;i++)
```

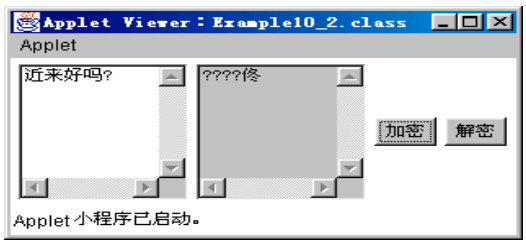


图 10.2　能加密解密的按钮

```
            {  mybutton.save[i]=(char)(mybutton.save[i]^'你');
            }
           String temp=new String(mybutton.save);
           mybutton.text2.setText(temp);
       }
    }
```

注 由于“异或”运算可能出现空字符 空字符无法显示在文本组件中 ,所以我们将加密后的文本存放到 mybutton 对象的数组 save 中.

Component 类有一个方法 **public void paint(Graphics g)**,我们可以在其子类中重写这个方法.当重写这个方法时,参数 **g** 是自动实例化的,这样我们就可以在子类中使用 **g** 调用相应方法,比如画串,画图形 见 **17** 章 ,图象 见 **22** 章 等.在下面的例子 **3** 中通过扩展 **Botton** 类自制一个竖状的按扭和一个模仿交通信号灯的按扭.

例子 3 效果如图 **10.3**

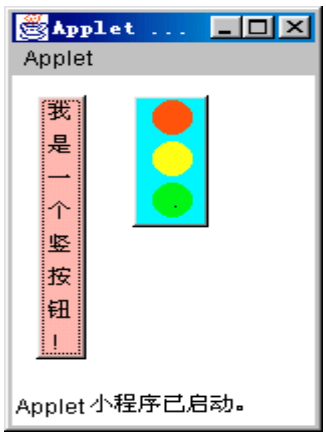


图 10.3 自制的竖状按扭

```
import java.awt.*;import java.applet.*;
import java.awt.event.*;
public class Example10_3 extends Applet
{   public void init()
   {  MyButton1 button1=new MyButton1();
      MyButton2 button2=new MyButton2();
      setLayout(null);
      add(button1);add(button2);
      button1.setLocation(12,12);
      button2.setLocation(60,12);
   }
}
class MyButton1 extends Button implements ActionListener
{  int n=-1;
   MyButton1()
   { setSize(25,160); addActionListener(this);
   }
   public void paint(Graphics g)
   {  g.drawString("我",6,14);  g.drawString("是",6,34);
      g.drawString("一",6,54);  g.drawString("个",6,74);
      g.drawString("竖",6,94);  g.drawString("按",6,114);
      g.drawString("钮",6,134); g.drawString("!",8,154);
   }
   public void actionPerformed(ActionEvent e)
```

```
    {  n=(n+1)%3;
       if(n==0)
          this.setBackground(Color.cyan);
       else if(n==1)
          this.setBackground(Color.orange);
       else if(n==2)
          this.setBackground(Color.pink);
    }
}
class MyButton2 extends Button
{  MyButton2()
    {  setSize(38,80); setBackground(Color.cyan);
    }
    public void paint(Graphics g)
    {  g.setColor(Color.red);
       g.fillOval(10,3,20,20);          //在按扭上画圆,见 17 章.
       g.setColor(Color.yellow);    g.fillOval(10,28,20,20);
       g.setColor(Color.green);     g.fillOval(10,53,20,20);
    }
}
```

10.3 标签

标签的功能是只显示文本,不能动态地编辑文本.Lalel 类的实例就是一个标签

Label 类的常用方法

1 Label() 使用这个构造方法创建标签对象,则标签上没有名称.

2 Label(String s) 使用这个构造方法创建标签对象,则标签上的名称是字符串 s,名称靠左对齐.

3 Label(String s,int alignment) 使用这个构造方法创建标签对象,则标签上的名称是字符串 s,名称的对齐方式由参数 alignment 决定,alignment 取值 Label.LEFT,Label.RIGHT,Label.CENTER.

4 public void setText(String s) 标签对象调用该方法可以设置标签上的名称.

5 public String getTextl() 标签对象调用该方法可以获取标签上的名称.

6 public void setAlignment(int alignment) 标签对象调用该方法可以设置标签上名称的对齐方式,alignment 取值 Label.LEFT,Label.RIGHT,Label.CENTE.

7 public int getAlignment() 标签对象调用该方法可以获取标签上名称的对齐方式,返回的值是 Label.LEFT,Label.RIGHT 或 Label.CENTE.

10.4 扩展标签

编写一个 Label 类的子类,以便增加一些新的属性和功能,在子类中增加新的成员变量,比如,子类可以有文本框,文本区等成员变量.

在下面的例子 4 中,MyLabel 类是 Label 的子类,MyLabel 创建的对象有文本框,文本区,按钮等成员变量..MyLabe 类实现了 ActionListener 接口,以便监视文本框 inputNumber 上的事件.当在文本框 inputNumber 中输入数字回车后,文本区 showResult 显示这个数的全部因子,单击按钮 button,文本区 showResult 显示不超过这个数的全部素数.

例子 4 效果如图 10.4

```
import java.awt.*;import java.awt.event.*;import java.applet.*;
class MyLabel extends Label implements ActionListener
{  String 标签上的初始名称;
   TextField inputNumber;TextArea showResult;Button button;
   MyLabel(String s,Container con)
   {  super(s);
      标签上的初始名称=s;
      inputNumber=new TextField(10); showResult =new TextArea(10,10);
      button=new Button("Enter");
      button.addActionListener(this);inputNumber.addActionListener(this);
      con.add(this);con.add(inputNumber);con.add(showResult);con.add(button);
   }
   public void actionPerformed(ActionEvent e)
   {  long n=0;
      showResult.setText(null);
      try{  n=Long.valueOf(inputNumber.getText()).longValue();
            this.setText(标签上的初始名称);
         }
      catch(NumberFormatException e1)
         {  this.setText("请输入数字字符");
         }
      if(e.getSource()==inputNumber)
        {  求因子(n);
        }
      if(e.getSource()==button)
          {  求素数(n);
          }
   }
  public void 求因子(long n)
```

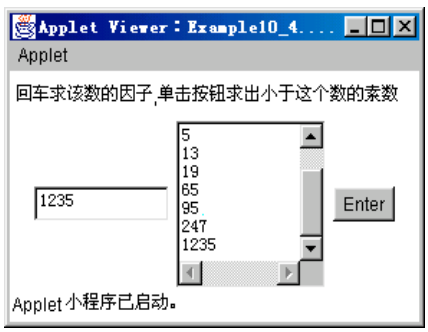


图 10.4 扩展标签

```
    {  for(int i=1;i<=n;i++)
        {  if(n%i==0)
             showResult.append("\n"+i);
        }
    }
    public void 求素数(long n)
    {  showResult.append("小于"+n+"的素数有:");
       for(int i=1;i<=n;i++)
          {  int j=0;
             for(j=2;j<i;j++)
                {  if(i%j==0)  break;
                }
             if(j>=i)
               { showResult.append("\n"+i);
               }
          }
     }
}
public class Example10_4  extends Applet
{  MyLabel lab;
   public void init()
   {  lab=new MyLabel("回车求该数的因子,单击按钮求出小于这个数的素数",this);
   }
}
```

习题十

1　编写一个小应用程序,在小应用程序的容器中有一个按钮和一个文本框.当点击按钮时,文本框显示按钮的名字.

2　编写一个有两个文本框和一个按钮的小应用程序,在一个文本框输入单词 girl 之后按回车键或点击按钮,另一个文本框都能显示"男孩"二字.

3　编写一个小应用程序,设计四个按钮,分别命名为"加","差","积,","除".有三个文本框.单击相应的按钮,将两个文本框的数字做运算,在第三个文本框中显示结果.要求处理 NumberFormatException.

4.编写一个竖状的标签.

第十一章　面板和画布

11.1 面板

1　Panel 类

java.awt 包的类 Panel 是用来建立面板的.因为 Panel 类是 Container　容器　的子类,因此 Panel 类及它的子类的实例也是一个容器　容器同时也是一个组件　.那么一个容器里添加若干个组件后,再放到另一个容器里.这叫容器的嵌套.

下面是一个例子,有三个面板,每个面板里添加三个按钮,再把三个面板添加到我们的小程序中.

例子 1　效果如图 11.1

```
import java.applet.*;import java.awt.*;import java.awt.event.*;
class Mypanel extends Panel implements ActionListener
{  Button button1,button2,button3;
   Color backColor;
   Mypanel()   //构造方法.当创建面板对象时,面板被初始化为有三个按钮.
   {  button1=new Button("确定");button2=new Button("取消");button3=new Button("保存");
      add(button1);add(button2);add(button3);
      setBackground(Color.pink);  //设置面板的底色.
      backColor=getBackground();  //获取底色.
      button1.addActionListener(this);button2.addActionListener(this);
   }
   public void actionPerformed(ActionEvent e)
   {  if(e.getSource()==button1)
        { setBackground(Color.cyan);
        }
      else if(e.getSource()==button2)
        { setBackground(backColor);
        }
   }
}
public class Example11_1 extends Applet
{  Mypanel panel1,panel2,panel3;
   Button button;
   public void init()
```

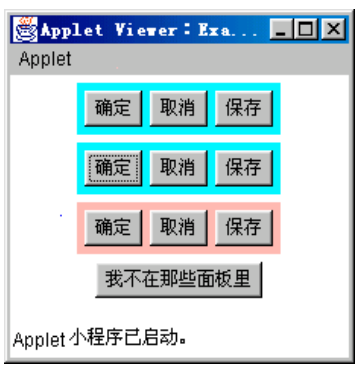


图 11.1　面板容器

```
    {  panel1=new Mypanel();panel2=new Mypanel();panel3=new Mypanel();
       button=new Button("我不在那些面板里");
       add(panel1);add(panel2);add(panel3);
       add(button);
    }
}
```

2　ScrollPane 类

java.awt 包中的 **ScrollPane** 类也是 **Container** 类的子类,因此该类创建的对象也是一个容器,称为滚动窗口.我们可以把一个组件放到一个滚动窗口中,然后通过滚动条来观察这个组件.与 **Panel** 创建的容器所不同的是,**ScrollPane** 带有滚动条,而且只能向滚动窗口添加一个组件.所以,经常将一些组件添加到一个面板容器中,然后再把这个面板添加到滚动窗口中.**ScrollPane** 有两个构造方法

1　ScrollPane()　创建滚动窗口,滚动条初始不可见,当添加的组件的可见范围大于滚动窗口时,滚动条自动出现.

2　ScrollPane(int a)　创建滚动窗口,参数 **a** 指定滚动条的初始状态.

a 取下列三个值之一

```
ScrollPane.SCROLLBARS_ALWAYS
ScrollPane.SCROLLBARS_AS_NEEDED
ScrollPane.SCROLLBARS_NEVER
```

例子 2　效果如图 **11.2** 所示

```
import java.awt.*;import java.applet.*;
public class Example11_2 extends Applet
{  Panel p ;
   ScrollPane scrollpane;
   public void init()
   {  p=new Panel();
      scrollpane=new ScrollPane(ScrollPane.SCROLLBARS_ALWAYS);
      p.add(new Button("one"));p.add(new Button("two"));
      p.add(new Button("three"));p.add(new Button("four"));
      scrollpane.add(p);//scrollpane添加一个面板.
      add(scrollpane);//小程序添加滚动窗口.
   }
}
```

图 11.2 滚动窗口

11.2 画布

在以前的例子学习中你已经知道小程序本身是一个容器,而且我们还可以在小程序中画

字符串,将来你还能学会画更复杂的图形和图像(见第十七章,二十二章).另外,java.awt 包还提供我们一个组件 画布 不是容器 ,它是一个可以在上面绘画的简单组件.java.awt 包中的类 Canvas 负责创建画布对象.创建画布对象的常用办法是用 Canvas 的子类来创建画布对象,并在子类中重写父类的方法 paint.需要注意的是,一定要在创建画布的类的构造方法中给出画布的尺寸 单位是像素 .

在例子 3 中,Mycanvas 类中的构造方法必须通过 setSize()方法设置它的大小.面板是一个容器,因此我们可以在面板中放置画布.小程序中有一个面板,面板里添加了画布组件.另一个画布添加到小程序中.

例子 3 效果如图 11.3 所示

```
import java.awt.*;import java.applet.*;
class Mycanvas extends Canvas
{  String s;
   Mycanvas(String s)
   {  this.s=s;
      setSize(90,80);
      setBackground(Color.cyan);
   }
   public void paint(Graphics g)
   {  if(s.equals("circle"))
         g.drawOval(20,25,30,30);
      else if(s.equals("rect"))
         g.drawRect(30,35,20,20);
   }
}
public class Example11_3 extends Applet
{  Mycanvas canvas1,canvas2;
   public void init()
   {  canvas1=new Mycanvas("circle");canvas2=new Mycanvas("rect");
      add(canvas1);
      Panel p=new Panel();p.setBackground(Color.pink);
      p.add(canvas2) ;
      add(p);
   }
}
```

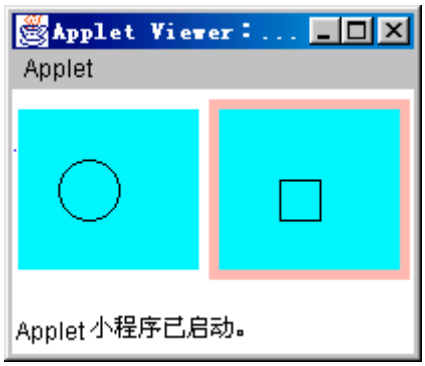


图 11.3 画布组件

在下面的例子 4 中,画布上绘制一个圆,用户通过文本条输入圆的半径以及圆的左上角的位置坐标.

例子 4 效果如图 11.4 所示

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
class Mycanvas extends Canvas
{  int x,y,r;
   int red,green,blue;
   Mycanvas()
   {  setSize(100,100);
      setBackground(Color.cyan);
   }
   public void setX(int x)
   {  this.x=x;
   }
   public void setY(int y)
   { this.y=y;
   }
   public void setR(int r)
   { this.r=r;
   }
   public void paint(Graphics g)
   {  g.drawOval(x,y,2*r,2*r);
   }
}
public class Example11_4 extends Applet implements ActionListener
{  Mycanvas canvas;
   TextField inputR,inputX,inputY;
   Button b;
   public void init()
   {  canvas=new Mycanvas();
      inputR=new TextField(6);
      inputX=new TextField(6);
      inputY=new TextField(6);
      add(new Label("输入圆的位置坐标  "));
      add(inputX);
      add(inputY);
      add(new Label("输入圆的半径  "));
      add(inputR);
      b=new Button("确定");
      b.addActionListener(this);
```

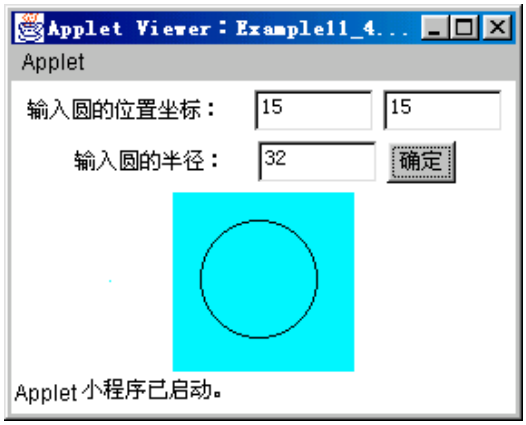


图 11.4　画布上绘制圆

```
        add(b);
        add(canvas);
    }
    public void actionPerformed(ActionEvent e)
    {  int x,y,r;
        try {
                x=Integer.parseInt(inputX.getText());
                y=Integer.parseInt(inputY.getText());
                r=Integer.parseInt(inputR.getText());
                canvas.setX(x);
                canvas.setY(y);
                canvas.setR(r);
                canvas.repaint();
              }
        catch(NumberFormatException ee)
              {
                x=0;y=0;r=0;
              }
    }
}
```

习题十一

1　写一个小应用程序,要求有一个面板,该面板是 Panle 类的子类的一个对象,面板中有 2 个画布组件.

2　写一个小应用程序,要求有一个画布,在画布上绘制一个矩形圆,用户通过文本条输入矩形的宽和高以及矩形左上角的位置坐标.

第十二章　布局设计

当我们把组件添加到容器中时,希望控制组件在容器中的位置,这就需要学习布局设计的知识.我们将分别介绍java.awt包中的FlowLayout,BorderLayout,CardLayout,GridLayout布局类和**java.swing.border**包中的BoxLayout布局类.

容器可以使用方法

```
setLayout(布局对象)
```

来设置自己的布局.

12.1 FlowLayout 布局

FlowLayout类创建的对象称做**FlowLayout**型布局.FlowLayout型布局是Panel型容器的默认布局,即Panel及其子类创建的容器对象,如果不专门为其指定布局,则它们的布局就是FlowLayout型布局.

FlowLayout类的常用方法

1　FlowLayout()　这个构造方法可以创建一个居中对齐的布局对象.例如

```
FlowLayout flow=new FlowLayout();
```

如果一个容器con使用这个布局布局对象,

```
con.setLayout(flow);
```

那么,con可以使用Container类提供的add方法将组件顺序地添加到容器中,组件按照加入的先后顺序从左向右排列,一行排满之后就转到下一行继续从左至右排列,每一行中的组件都居中排列,组件之间的默认水平和垂直间隙是5个像素.

2　FlowLayout(int aligin,int hgap,int vgap)　使用这个构造方法可以创建一个布局对象,其中对齐方式align可取值

```
FlowLayout.LEFT,FlowLayout.CENTER,FlowLayout.RIGHT
```

3　public void setAlignment(int align)　FlowLayout布局对象调用该方法可以设置布局的对齐方式.

4　public void setHgap(int hgap)　FlowLayout布局对象调用该方法可以设置布局的水平间隙.

5　public void setVgap(int vgap)　FlowLayout布局对象调用该方法可以设置布局的垂直间隙.

FlowLayout 对应的布局非常简单,遵循这种布局的容器将其中的组件按照加入的先后顺序从左向右排列,一行排满之后就转到下一行继续从左至右排列,每一行中的组件都按着布局指定的对齐方式和垂直间隙排列　当形成多行组件时,行与行之间的间隙是就是布局的水平间隙.尽管这种布局非常方便,但是当容器内的组件素增加时,就显得高低参差不齐.有时会采用容器嵌套的方法,即把一个容器当作一个组件加入另一个容器,使整个容器的布局达到应用的需求.

在下面例子 1 使用 FlowLayout 布局放置 12 个组件.

例子 1

```
import java.applet.*;import java.awt.*;
public class Example12_1 extends Applet
{  public void init()
   {  FlowLayout flow=new FlowLayout();
      flow.setAlignment(FlowLayout.LEFT);
      flow.setHgap(20);flow.setVgap(40);
      setLayout(flow);
      setBackground(Color.cyan);
      for(int i=1;i<=12;i++)
         { add(new Button("i am "+i));
         }
   }
}
```

12.2 BorderLayout 布局

BorderLayout 也是一种简单的布局策略,如果一个容器使用这种布局,那么容器空间简单地划分为东,西,南,北,中五个区域,中间的区域最大.每加入一个组件都应该指明把这个组件加在哪个区域中,区域由 BorderLayout 中的静态常量 CENTER,NORTH,SOUTH,WEST,EAST 表示,例如,一个使用 BorderLayout 布局的容器 con,可以使用 add 方法将一个组件 b 添加到中心区域

```
con.add(b,BorderLayout.CENTER);
```

或

```
con.add(BorderLayour.CENTER,b);
```

添加到某个区域的组件将占据整个这个区域.每个区域只能放置一个组件,如果向某个已放置了组件的区域再放置一个组件,那么先前的组件将被后者替换掉.使用 BorderLayout 布局的容器最多能添加 5 个组件,如果容器中需要加入超过 5 个组件,就必须使用容器的嵌套或改用其他的布局策略.

例子 2　效果如图 12.1 所示

```
import java.awt.*;import java.applet.*;
public class Example12_2 extends Applet
{  Button button1,button2;
   Label label1,label2;
   TextArea text;
   public void init()
   {  setLayout(new BorderLayout());
      text=new TextArea(10,10);
      button1=new Button("东");
      button2=new Button("西");
      label1=new Label("上北",Label.CENTER);
      label2=new Label("下南",Label.CENTER);
      add(BorderLayout.NORTH,label1);
      add(label2,BorderLayout.SOUTH);
      add(button1,BorderLayout.EAST);
      add(BorderLayout.WEST,button2);
      add(BorderLayout.CENTER,text);
   }
}
```

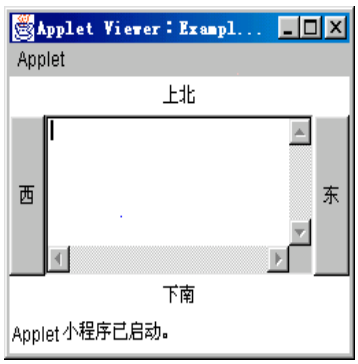


图 12.1　Borderlayout 布局

12.3 CardLayout 布局

使用 CardLayout 的容器可以容纳多个组件,但是实际上同一时刻容器只能从这些组件中选出一个来显示,就像一叠"扑克牌"每次只能显示最上面一张一样,这个被显示的组件将占据所有的容器空间,依次排序.假设有一个容器 con,那么,使用 CardLayout 的一般步骤如下

1　创建 CardLayout 对象作为布局　如,CardLayout card=new CardLayout()

2　使用容器的 setLayout()方法为容器设置布局　如,con.setLayout(card)

3　调用容器的方法 add(String s,Componnemt b)将组件 b 加入容器,并给出了显示该组件的代号 s.组件的代号是你另外给的,和组件的名字没有必然联系.不同的组件代号互不相同.最先加入 con 的是第一张,依次排序.

4　创建的布局 card 用 CardLayout 类提供的 show(方法),根据容器名字 con 和其中的组件的代号 s 显示这一组件

```
mycard.show(con,s);
```

也可以按组件加入容器的顺序显示组件,如:

```
card.first(con);
```

显示 con 中的第一个组件.

```
card.last(con);
```

显示 con 中最后一个组件.

```
card.next(con);
```

显示当前正在被显示的组件的下一个组件.

```
card.previous(con);
```

显示当前正在被显示的组件的前一个组件.

以下的例子 3 中,有一个面板容器 p,在 p 中放置了 20 个画布组件,p 使用 CardLayout 布局策略布局这 20 个画布组件.然后将这个面板添加到小程序中.在小程序中有 3 个按钮,一个负责看 p 中的第一个组件,一个负责看 p 中最后一个组件,一个负责循环看 p 中的组件.这些画布上画的是逐渐增大的实心圆.

例子 3 效果如图 12.2 所示

```
import java.applet.*;import java.awt.*;import java.awt.event.*;
class Mycanvas extends Canvas
{  int x,y;
   Mycanvas(int a,int b)
   {  x=a;y=b;
      setSize(100,160);
   }
   public void paint(Graphics g)
   {  g.setColor(Color.red);
      g.fillOval(50,50,4*x,4*y);//画实心椭圆
      g.drawString("我是第 "+x,10,150);
   }
}
public class Example12_3 extends Applet implements ActionListener
{  CardLayout mycard;
   Button button1,button2,button3;
   Mycanvas mycanvas[];
   Panel p;
   public void init()
   {  setLayout(new BorderLayout()); //小容器的布局是边界布局.
      mycard=new CardLayout();
      p=new Panel();
      p.setLayout(mycard);          //p 的布局设置为 mycard  卡片式布局
      button1=new Button("first");
      button2=new Button("next");
```

```
        button3=new Button("last one");
        mycanvas=new Mycanvas[21];
        for(int i=1;i<=20;i++)
           { mycanvas[i]=new Mycanvas(i,i);
             p.add("i am"+i,mycanvas[i]);
           }
        button1.addActionListener(this);
        button2.addActionListener(this);
        button3.addActionListener(this);
        Panel p2=new Panel();
        p2.add(button1);p2.add(button2);p2.add(button3);
        add(p,BorderLayout.CENTER);add(p2,BorderLayout.SOUTH);
    }
    public void actionPerformed(ActionEvent e)
    {  if(e.getSource()==button1)
         {  mycard.first(p);
         }
       else if(e.getSource()==button2)
        {  mycard.next(p);
        }
       else if(e.getSource()==button3)
        {  mycard.last(p);
        }
    }
}
```

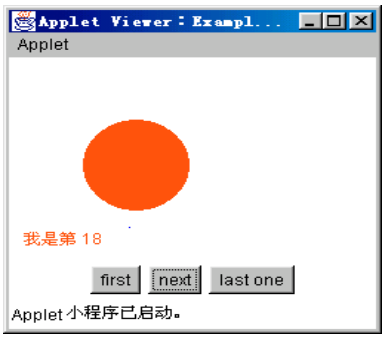


图 12.2　CardderLayout 布局

12.4 GridLayout 布局

GridLayout 是使用较多的布局编辑器,其基本布局策略是把容器划分成若干行乘若干列的网格区域,组件就位于这些划分出来的小格中.GridLayout 比较灵活,划分多少网格由程序自由控制,而且组件定位也比较精确,使用 GridLayout 布局编辑器的一般步骤如下

1　使用 GridLayout 的构造方法 GridLayout(int m,int n)创建布局对象,指定划分网格的行数 m 和列数 n,例如

```
GridLayout grid=new new GridLayout(10,8)
```

2　使用 GridLayout 布局的容器调用方法 add 将组件加入容器,组件进入容器的顺序将按照第一行第一个,第一行第二个,...第一行最后一个,第二行第一个,...最后一行第一个,...最后一行最后一个.

使用 **GridLayout** 布局的容器最多可添加 **m*n** 个组件.GridLayout 布局中每个网格都是相同大小并且强制组件与网格的大小相同.

下面例子 4,利用 GriderLayou 布局模拟的国际象棋棋盘.

例子 4 效果如图 12.3 所示

```
import java.applet.*; import java.awt.*;
import java.awt.event.*;
public class Example12_4 extends Applet
{  GridLayout grid;
   public void init()
   {  grid=new GridLayout(12,12);
      setLayout(grid);
      Label label[][]=new Label[12][12];
      for(int i=0;i<12;i++)
        {  for(int j=0;j<12;j++)
            {  label[i][j]=new Label();
               if((i+j)%2==0)
                 label[i][j].setBackground(Color.black);
               else
                 label[i][j].setBackground(Color.white);
               add(label[i][j]);
            }
         }
   }
}
```

图 12.3 GriderLayout 布局

由于 GridLayout 布局中每个网格都是相同大小并且强制组件与网格的大小相同,使得容器中的每个组件也都是相同的大小,显得很不自然. 为了克服这个缺点,你可以使用容器嵌套. 如,一个容器使用 GridLayout 布局,将容器分为三行一列的网格,那么你可以把另一个容器添加到某个网格中,而添加的这个容器又可以设置为 GridLayout 布局,FlowLayout 布局,CarderLayout 布局或 BorderLayout 布局等. 利用这种嵌套方法,可以设计出符合一定需要的布局.

12.5 BoxLayout 布局

用 BoxLayout 类可以创建一个布局对象,称为盒式布局.BoxLayout 在 java.swing.border 包中.java swing 包提供了 Box 类,该类创建的容器称作一个盒式容器,盒式容器的的默认布局就是盒式布局,而且不允许更改盒式容器的布局.因此,在策划程序的布局时,可以利用容器的嵌套,将某个容器嵌入几个盒式容器,达到你的布局目的.

使用盒式布局的容器将组件排列在一行或一列,这取决于创建盒式布局对象时,是否指定了是行排列还是列排列.使用 BoxLayou 的构造方法 BoxLayout(Container con,,int axis) 可以创建一个盒式布局对象,并指定容器 con 使用该布局对象,参数 axis 的有效值是

BoxLayout.X_AXIS,BoxLayout.Y_AXIS

该参数 axis 的取值决定盒式布局是行型盒式布局或列型盒式布局.使用行　列　型盒式布局的容器将组件排列在一行　列　,组件按加入的先后顺序从左　上　向右　下　排列,容器的两端是剩余的空间.和 FlowLayou 布局不同的是,使用行型盒式布局的容器只有一行　列　,即使组件再多,也不会延伸到下一行　列　,这些组件可能会被缩小大小,紧缩在这一行　列　中.

行型盒式布局容器中添加的组件的上沿在同一水平线上.列型盒式布局容器中添加的组件的左沿在同一垂直线上.

使用 Box 类的类　静态　方法 createHorizontalBox()可以获得一个具有行型盒式布局的盒式容器　使用 Box 类的类　静态　方法 createVerticalBox()可以获得一个具有列型盒式布局的盒式容器.

在下面的例子 5 中,有三个 Box 容器　baseBox,boxH,boxV,其中 baseBox 和 boxH 是行型盒式布局,boxV 是列型盒式布局.我们将 boxH 和 boxV 中分别添加了若干个按钮后,再将 boxH,boxV 添加到 baseBox 中.

例子 5　效果如图 12.4 所示

```
import javax.swing.*;import java.awt.*;import java.awt.event.*;
import javax.swing.border.*;
public class Example12_5 extends java.applet.Applet
{  Box baseBox ,boxH,boxV;
   public  void init()
   {  baseBox=Box.createHorizontalBox();
      boxH=Box.createHorizontalBox();
      boxV=Box.createVerticalBox();
      for(int i=1;i<=5;i++)
         { boxH.add(new JButton("按钮 "+i));
           boxV.add(new JButton("按钮 "+i));
         }
      baseBox.add(boxH);baseBox.add(boxV);
      add(baseBox);
   }
}
```

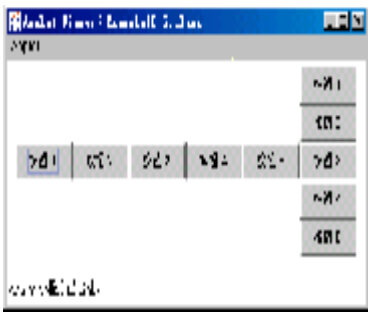


图 12.4　BoxLayout 布

1　支撑

如果想控制盒式布局容器中组件之间的距离,就需要使用水平支撑组件或垂直支撑组件.

Box 类调用静态方法 createHorizontalStrut(int width)可以得到一个不可见的水平 Struct 类型对象,称做水平支撑.该水平支撑的高度为 0,宽度是 width.

Box 类调用静态方法 createVertialStrut(int height)可以得到一个不可见的垂直 Struct 类型对象,称做垂直支撑.参数 height 决定垂直支撑的高度,垂直支撑的宽度为 0.

一个行型盒式布局的容器,可以通过在添加的组件之间插入水平支撑来控制组件之间的距离.一个列型盒式布局的容器,可以通过在添加的组件之间插入垂直支撑来控制组件之间的距离.

下面的例子 6 中,有两个列型盒式容器 boxV1,boxV2 和一个行行盒式容器 baseBox.在列型盒式容器的组件之间添加垂直支撑,控制组件之间的距离,将 boxV1,boxV2 添加到 baseBox 中,并在它俩之间添加了水平支撑.

例子 6 效果如图 12.5 所示

```
import javax.swing.*; import java.awt.*;
import javax.swing.border.*;
public class Example12_6 extends java.applet.Applet
{  Box baseBox ,boxV1,boxV2;
   public  void init()
   {  boxV1=Box.createVerticalBox();
      boxV1.add(new Label("输入您的姓名"));
      boxV1.add(Box.createVerticalStrut(8));
      boxV1.add(new Label("输入 email"));
      boxV1.add(Box.createVerticalStrut(8));
      boxV1.add(new Label("输入您的职业"));
      boxV2=Box.createVerticalBox();
      boxV2.add(new TextField(16));
      boxV2.add(Box.createVerticalStrut(8));
      boxV2.add(new TextField(16));
      boxV2.add(Box.createVerticalStrut(8));
      boxV2.add(new TextField(16));
      baseBox=Box.createHorizontalBox();
      baseBox.add(boxV1);
      baseBox.add(Box.createHorizontalStrut(10));
      aseBox.add(boxV2);
      add(baseBox);
   }
}
```

图 12.5 添加支撑

2 胶水

如果想处理盒式布局容器的剩余空间 容器的两端是剩余的空间 就需要胶水组件.

胶水也是不可见的组件,Box 类调用静态方法 createHorizontalGlue()可以得到一个不可见的水平 Glue 类型对象,称做水平胶水.对于行型盒式布局的容器,添加若干个组件后,容器的两端也许会有剩余空间.如果容器最后添加一个水平胶水组件,那么这个水平胶水的大小就和整个剩余空间的大小相同,这样就达到了容器中的组件靠左对齐的目的 如果容器的最先

添加一个水平胶水组件,那么这个水平胶水的大小就和整个剩余空间的大小相同,这样就达到了容器中的组件靠右对齐的目的　如果在容器的组件之间添加一个水平胶水组件,那么这个水平胶水前面的组件靠左对齐,后面的组件靠右对齐.

Box 类调用静态方法 createVerticalGlue()可以得到一个不可见的垂直 Glue 类型对象,称做垂直胶水.对于列型盒式布局的容器,添加若干个组件后,容器的两端会有剩余空间.如果容器最后添加一个垂直胶水组件,那么这个垂直胶水和整个剩余空间的大小相同,这样就达到了容器中的组件靠上对齐的目的　如果容器最先添加一个垂直胶水组件,那么这个垂直胶水和整个剩余空间的大小相同,就达到了容器中的组件靠下对齐的目的　如果在容器的组件之间添加一个垂直胶水组件,那么这个垂直胶水前面的组件靠上对齐,后面的组件靠下对齐.

下面的例子 7 使用胶水组件调整盒式布局容器中组件的对齐方式.

例子 7　效果如图 12.6 所示

```
import javax.swing.*;import java.awt.*;
import javax.swing.border.*;
public class Example12_7 extends java.applet.Applet
{  Box baseBox ,boxV1,boxV2,boxV3;
   public void init()
   {  boxV1=Box.createVerticalBox();
      for(int i=1;i<=3;i++)
       {  boxV1.add(new JButton("按钮"+i));
       }
      boxV1.add(Box.createVerticalGlue());
      boxV2=Box.createVerticalBox();
      boxV2.add(Box.createVerticalGlue());
      for(int i=1;i<=8;i++)
       {  boxV2.add(new JButton("按钮"+i));
       }
      boxV3=Box.createVerticalBox();
      for(int i=1;i<=6;i++)
       {  boxV3.add(new JButton("按钮"+i));
          if(i==3)
             boxV3.add(Box.createVerticalGlue());
       }
      baseBox=Box.createHorizontalBox();
      baseBox.add(boxV1);baseBox.add(boxV2);
      baseBox.add(boxV3); add(baseBox);
   }
}
```

图 12.6　添加胶水

12.6 null 布局

我们可以把一个容器的布局设置为 null 布局　空布局　.

setBounds　int a,int b,int width,int height　方法是所有组件都拥有的一个方法,组件调用该方法可以设置本身的大小和在容器中的位置.

例如,p 是某个容器,

```
p.setLayout(null);
```

把 p 的布局设置为空布局.

向空布局的容器 p 添加一个组件 c 需要两个步骤,首先使用 add(c)方法向容器添加组件,然后组件 c 再调用 **setBounds　int a,int b,int width,int height**　方法设置该组件在容器中的位置和本身的大小,组件都是一个矩形结构,方法中的参数 a,b 是被添加的组件 c 的左上角在容器中的位置坐标即该组件距容器左面 a 个像素,距容器上访 b 个像素　weidth,height 是组件 c 的宽和高.

习 题 十二

1　在一个 BorderLayout 布局的东,西,南,北区域分别添加 4 个使用盒式布局的容器,要求每个容器中有若干个组件,东,西区域的容器使用列式盒布局,南,北区域的容器使用行式盒布局.

2　FlowLayout,GridLayout,BorderLayout 布局都可以调用 setHgap(int hgap)和 setVgap(int vgap)设置组件的垂直间距和水平间距.编写程序,通过调整组件的间距达到改变组件大小的目的.

3　编写一个小应用程序.小应用程序中有一个面板和按钮,将小应用程序和面板的布局设为 null,小应用程序添加面板,面板中添加按钮.

第十三章　选择型组件

这一章介绍经常使用的选择框,下拉式列表,滚动列表组件.

13.1 选择框

选择框提供两种状态,一种是选中,另一种是未选中.java.awt 包中的 Checkbox 用来建立选择框,即 Checkbox 创建的一个对象就是一个选择框.

1　Checkbox 类的常用方法

1　Checkbox()　使用该构造方法创建选择框,选择框右面没有名称,初始状态是未选中.

2　Checkbox(String　s) 使用该构造方法创建选择框,选择框右面的名称由参数 s 指定.

3　Checkbox(String　s,boolean b) 使用该构造方法创建选择框,选择框右面的名称由参数 s 指定,初始状态有参数 b 决定,b 取值 true,选择框是选中状态,否则,选择框是未选中状态.

4　Checkbox(String　s,boolean b,CheckboxGroup g)　如果使用该构造方法创建选择框,选择框右面的名称由参数 s 指定,初始状态有参数 b 决定,b 取值 true,选择框是选中状态,否则,选择框是未选中状态,参数 g 决定该选择狂所在的组,属于同一个组中的选择框,每一时刻只能有一个处于选中状态.

5　public void addItemListener(ItemListener)　选择框调用该方法可以向选择框增加监视 ItemEvent 事件的监视器.

6　public void removeItemListener(ItemListener)　选择框调用该方法可以移去选择框上的 ItemEvent 事件的监视器.

7　public boolean getSate()　选择框调用该方法可以返回选择框的布尔状态,当选择框是选中状态,该方法返回 true,否则返回 false.

8　setState(boolean b)　选择框调用该方法可以设置选择框的状态.

9　getLabel()　选择框调用该方法可以获取选择框的上的名称.

10　setLabel(String　s)　选择框调用该方法可以将选择框上的名称设置为指定的字符串 s.

在下面的例子 1 中一部分选择框被归组,那么在同一组中的选择框,同一时刻只能有一个被选中.

例子 1　效果如图 13.1

```
import java.applet.*;import java.awt.*;
class Mypanel1 extends Panel
```

```
{  Checkbox box1,box2,box3;CheckboxGroup sex;
   Mypanel1()
   {  sex=new CheckboxGroup();
      box1=new Checkbox("男",true,sex);
      box2=new Checkbox("女",false,sex);
      add(box1);add(box2);
   }
}
class Mypanel2 extends Panel
{  Checkbox box1,box2,box3;
   Mypanel2()
   {  box1=new Checkbox("读书");
      box2=new Checkbox("电脑");
      box3=new Checkbox("电影");
      add(box1);add(box2);add(box3);
   }
}
public class Example13_1 extends Applet
{  Mypanel1 panel1;Mypanel2 panel2;
   public void init()
   {  setLayout(new GridLayout(1,2));
      panel1=new Mypanel1();panel2=new Mypanel2();
      add(panel1);add(panel2);
   }
}
```

图 13.1　选择框组件

2　选择框上的 ItemEvent 事件

a　选择框可以发生 ItemEvent 事件

当选择框获得监视器之后,选择框从未选中状态变成选中状态或从选中状态变成未选中状态时就发生 ItemEvent 事件,ItemEvent 类将自动创建一个事件对象.

b　选择框获得监视器

发生 ItemEvent 事件的事件源获得监视器的方法是　addItemListener(监视器).由于选择框可以发生 ItemEvent 事件, Checkbox 类提供了 addItemListener 方法.

c　处理事件的接口

处理 ItemEvent 事件的接口是 ItemListener,创建监视器的类必须实现 ItemListener 接口,该接口中只有一个的方法.当在选择框发生 ItemEvent 事件时,监视器将自动调用接口方法

itemStateChanged(ItemEvent e)

对发生的事件作出处理.

5 ItemEvent 类中的方法

getItemSelectable()方法它返回 **Itemevent** 事件的事件源.也就是说,对于 **ItemEvent** 事件类型的事件,如果想寻找事件源,需使用 **getItemSelectable()**方法.

下面的例子处理了选择框上的 **ItemEvent** 事件,当选择框发生 **ItemEvent** 事件后,将选择框上的名称放入一个文本区.

例子 2 效果如图 13.2

```
import java.awt.*;import java.awt.event.*;import java.applet.*;
public class Example13_2 extends Applet implements ItemListener
{  Checkbox box1,box2,box3,box4;
   TextArea text;
   public void init()
   {  box1=new Checkbox("  ");box2=new Checkbox("  ");
      box3=new Checkbox("  ");box4=new Checkbox("  ");
      box1.addItemListener(this);
      box2.addItemListener(this);
      box3.addItemListener(this);
      box4.addItemListener(this);
      text=new TextArea(16,18);
      add(box1);add(box2);
      add(box3);add(box4);
      add(text);
   }
   public void itemStateChanged(ItemEvent e)
   {  Checkbox box=(Checkbox)e.getItemSelectable(); //获取事件源.
      if(box.getState())
      {  int n=text.getCaretPosition();       //获取文本区活动光标位置.
         text.insert(box.getLabel(),n);       //插入字符.
         box.setState(false);
      }
   }
}
```

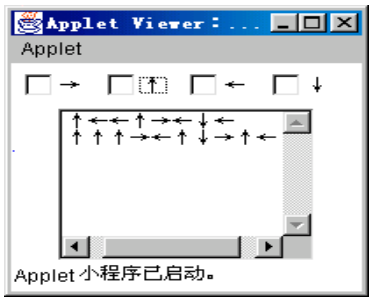


图 13.2 处理 ItemEvent 事件

下面是一个统计选票的小程序,仔细分析该例子有利于对前面内容的理解.在这个例子中,候选人的名字从一个文本框里获取,使用了第 5 章讲述的 StringTokenizer 类 见 5.14 .当选择了候选人后,程序使用选择框代表这些候选人,即选择框的名字就是候选人的名字.然后根据选择框的状态的变化统计出候选人的最后得票数,并可按着得票的多少对候选人进行

排序.在下面的例子 3 中,规定最多从候选人中选取 3 人,如果一张选票选取多于 3 人,就做废票处理.程序能显示一共统计了多少选票,并能统计出废票和弃权票的票数.

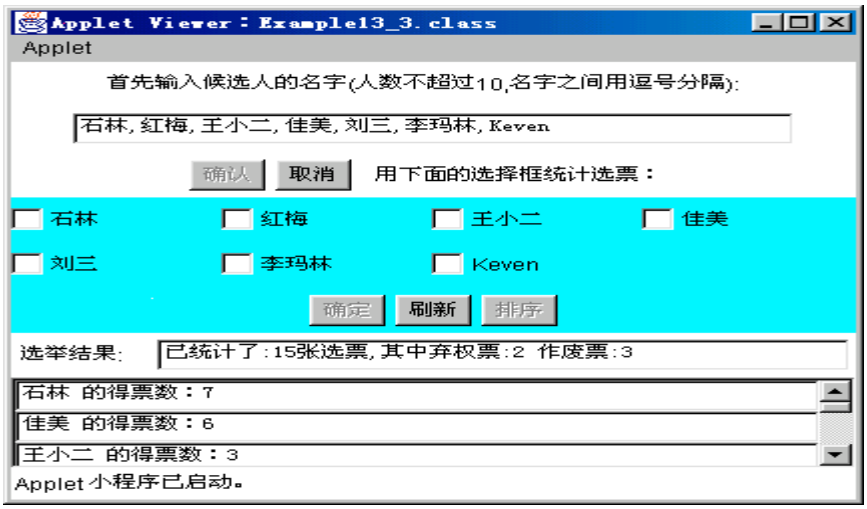


图 **13.3** 选票统计程序

例子 3 效果如图 13.3

```
import java.applet.*;import java.awt.*;import java.awt.event.*;
import java.util.StringTokenizer;
public class Example13_3 extends Applet implements ActionListener
{  Label 候名字=new Label("首先输入候选人的名字(人数不超过 10,名字之间用逗号分隔):"),
         统计选票=new Label("用下面的选择框统计选票  ",Label.CENTER),
         结果=new Label("选举结果:");
   Button 确认=new Button("确认"),取消=new Button("取消"),
          确定=new Button("确定"),刷新=new Button("刷新"),
          排序=new Button("排序");
   TextField name=new TextField(48);          //输入候选人.
   TextField voteMessage=new TextField(46);   //显示选举信息.
   Checkbox checkbox[]=new Checkbox[10];      //选择框数组,代表候选人.
   TextField personVote[]=new TextField[10];  //文本条数组,显示每个人的得票情况.
  int  count[]=new int[10],                   //记录每个人的得票数.
       totalVote=0,                           //总票数.
       peopleNumber=0;                        //候选人个数.
   Panel p2_1=new Panel();                    //添加候选人的面板.
   int  有效人数=3,                           //可选举的最多人数.
        废票数=0,弃权票数=0;
   public void init()
   {  setLayout(new GridLayout(3,1));
```

```
    Panel p1=new Panel(),          //添加到小程序上部的面板和添加到该面板中的面板.
    p1_1=new Panel(),p1_2=new Panel(),p1_3=new Panel();
    p1.setLayout(new BorderLayout());
    p1_1.add(候名字); p1_2.add(name);
    p1_3.add(确认);p1_3.add(取消);p1_3.add(统计选票);
    p1.add(p1_1,BorderLayout.NORTH);p1.add(p1_2,BorderLayout.CENTER);
    p1.add(p1_3,BorderLayout.SOUTH);
    Panel p2=new Panel(),  //添加到小程序中间的面板 p2,它里面的一个面板 p2_1,
                           //将用来添加代表候选人的选择框.
    p2_2=new Panel();
    p2.setLayout(new BorderLayout());p2.setBackground(Color.cyan);
    p2_1.setLayout(new GridLayout(2,5));
    p2_2.add(确定); p2_2.add(刷新); p2_2.add(排序);
    p2.add(p2_1,BorderLayout.CENTER);p2.add(p2_2,BorderLayout.SOUTH);
    for(int i=0;i<=9;i++)
      {  checkbox[i]=new Checkbox();
         p2_1.add(checkbox[i]);
      }
    Panel p3=new Panel(),      //添加到小程序底部的面板 p3,及它里面的面板.
    p3_1=new Panel(),p3_2=new Panel();
    p3.setLayout(new BorderLayout());
    p3_1.add(结果);p3_1.add(voteMessage);
    p3_2.setLayout(new GridLayout(10,1));
    for(int i=0;i<=9;i++)
      {  personVote[i]=new TextField();
         p3_2.add(personVote[i]);
      }
    ScrollPane scroll=new ScrollPane();
    scroll.add(p3_2);                    //把 p3_2 添加到一个滚动窗体中.
    p3.add(p3_1,BorderLayout.NORTH);
    p3.add(scroll,BorderLayout.CENTER);
    add(p1);add(p2); add(p3);                          //小程序添加这三个面板.
    确认.addActionListener(this); 取消.addActionListener(this);
    确定.addActionListener(this); 刷新.addActionListener(this);
    排序.addActionListener(this);
}
public void actionPerformed(ActionEvent e)
{  String s[]=new String[10];
   if(e.getSource()==确认)
```

```
  {  p2_1.removeAll();
     String s_name=name.getText();
    //提取候选人的名字,名字用逗号  英文逗号或汉文逗号  分隔:
     StringTokenizer fenxi=new StringTokenizer(s_name,",,");
     peopleNumber=fenxi.countTokens();   //获取候选人的个数.
     int i=0;
     while(fenxi.hasMoreTokens())  //用单选框代表候选人,并添加到面板 p2_1.
      {
         s[i]=fenxi.nextToken();
         p2_1.add(checkbox[i]);
         checkbox[i].setLabel(s[i]);
         i++;
      }
     for(int k=0;k<peopleNumber;k++)
      {  personVote[k].setText(null);
      }
  }
else if(e.getSource()==取消)
  {  name.setText(null);
     确认.setEnabled(true);
     for(int k=0;k<peopleNumber;k++)
        {  personVote[k].setText(null);
        }
   }
else if(e.getSource()==确定)       //统计候选人的得票数目.
  {  totalVote=totalVote+1;           //记录下统计的票数.
     确认.setEnabled(false);
     //检查选票是否有效
     int number=0;
     for(int k=0;k<peopleNumber;k++)
       {  if(checkbox[k].getState())
              {  number++;
              }
       }
     if(number>有效人数)
       {  废票数++;
          for(int k=0;k<peopleNumber;k++)
              {  checkbox[k].setState(false);
              }
```

```
        }
        else if(number==0)
        {  弃权票数++;
        }
        else if(number>0&&number<=有效人数)
        {  for(int k=0;k<peopleNumber;k++)
             {  if(checkbox[k].getState())
                  {  count[k]=count[k]+1;
                     checkbox[k].setState(false);
                     personVote[k].setText(checkbox[k].getLabel()+
                                    " 的得票数  "+count[k]);
                  }
                else
                  {  personVote[k].setText(checkbox[k].getLabel()+
                                    " 的得票数  "+count[k]);
                  }
             }
        }
 voteMessage.setText("已统计了:"+totalVote+"张选票,其中弃权票:"+
                                    弃权票数+" 作废票:"+废票数);
 }
else if(e.getSource()==排序)  //对选举人按得票数,从大到小排序.
 {  for(int i=0;i<peopleNumber;i++)
     {  for(int j=i+1;j<peopleNumber;j++)
         {   if(count[j]>count[i])
            {  String str_temp=personVote[i].getText();
               personVote[i].setText(personVote[j].getText());
               personVote[j].setText(str_temp);
               int nnn=count[i];count[i]=count[j];count[j]=nnn;
             }
         }
     }
    排序.setEnabled(false);确定.setEnabled(false);
 }
else if(e.getSource()==刷新)
 {  totalVote=0;
    voteMessage.setText("已统计了  "+totalVote+"张选票");
    name.setText(null);
    确认.setEnabled(true);  确定.setEnabled(true); 排序.setEnabled(true);
```

```
            for(int i=0;i<=4;i++)
              {  count[i]=0;
                 personVote[i].setText(null);
                 p2_1.removeAll();
               }
          }
      }
  }
```

13.2 下拉列表

下拉列表是用户十分熟悉的一个组件,用户可以在下拉列表看到第一个选项和它旁边的箭头按钮,当用户单击箭头按钮时,选项列表打开. java.awt 包中的 Choice 类是专门用来建立下拉列表的,即 Choice 创建的一个对象就是一个下拉列表组件.

1 常用方法

1 Choice() 使用该构造方法创建下拉列表.

2 public void add(String name) 下拉列表调用该方法可以增加一个名字是 name 的选项.

3 public int getSelectedIndex() 下拉列表调用该方法可以返回当前下拉列表中被选中的选项的索引,索引的起始值是 0.

4 public String getSelectedItem() 下拉列表调用该方法可以返回当前下拉列表中被选中的选项的名字..

5 public void insert(String name ,int index) 下拉列表调用该方法可以将名字是 name 的选项插入下拉列表的指定位置. index 值非负,并且小于下拉列表的选项总数,否则会发生 ArrayIndexOutOfBoundsException 异常.

6 public void remove(String name) 下拉列表调用该方法可以删除名字是 name,索引值最小的选项. 如果该名字不存在,会发生 IllegalArgumentException 异常

7 public int getItemCount() 下拉列表调用该方法可以返回当前下拉列表中选项总数.

8 public void select(int index) 下拉列表调用该方法可以把下拉列表中索引值是 index 的选项设置成选中状态.index 值非负,并且小于下拉列表的选项总数.

9 public void select(String name) 下拉列表调用该方法可以把下拉列表中名字是 name 的选项设置成选中状态,如果有多个选项的名字相同,就将其中索引值最小的设置成选中状态.

10 public void remove(int index) 下拉列表调用该方法可以从下拉列表的选项中删除索引值是 index 选项. index 值非负,并且小于下拉列表的选项总数,否则会发生 ArrayIndexOutOfBoundsException 异常.

11 public void removeAll() 下拉列表调用该方法可以删除全部选项

12　public void addItemListener(ItemListener)　下拉列表调用该方法可以向下拉列表增加 ItemEvent 事件的监视器.

13　public void removeItemListener(ItemListener)　下拉列表调用该方法可以移去下拉列表上的监视器.

2　下拉式列表上的 ItemEvent 事件

对于下拉式列表事件源,可以发生 ItemEvent 事件.当下拉式列表获得监视器之后,用户在下拉列表选项列表中选中某个选项时就发生 ItemEvent 事件,ItemEvent 类将自动创建一个事件对象.

在下面的例子中,当下拉列表发生 ItemEvent 事件时,文本区显示被选中的选项的索引值和选项的名字.程序中还有一个文本框,在这个文本框中输入文本回车或单击名字为"添加"的按钮,下拉列表将文本框的字符串作为一个选项添加到下拉列表中,同时将新添加的选项设置成选中状态.如果单击名字为"删除"的按钮,就删除下拉列表中选中的选项.

例子 4　效果如图 13.4

```
import java.awt.*;import java.awt.event.*;
public class Example13_4 extends java.applet.Applet implements
ItemListener,ActionListener
{  Choice choice;
   TextField text;
   TextArea area;
   Button add,del;
   public void init()
   {  choice=new Choice();
      text=new TextField(8);
      area=new TextArea(6,15);
      choice.add("音乐天地");
      choice.add("武术天地");
      choice.add("象棋乐园");
      choice.add("交友聊天");
      add=new Button("添加");
      del=new Button("删除");
      add.addActionListener(this);del.addActionListener(this);
      choice.addItemListener(this);
      add(choice);
      add(del);add(text);add(add);add(area);
   }
   public void itemStateChanged(ItemEvent e)
   {  String name=choice.getSelectedItem();
```

图 13.4　处理事件

```
            int index=choice.getSelectedIndex();
            area.setText("\n"+index+":"+name);
        }
        public void actionPerformed(ActionEvent e)
        {  if(e.getSource()==add||e.getSource()==text)
             {  String name=text.getText();
                if(name.length()>0)
                   {  choice.add(name);
                      choice.select(name);
                      area.append("\n添加"+name);
                   }
             }
           else if(e.getSource()==del)
             {  choice.remove(choice.getSelectedIndex());
                area.append("\n删除"+choice.getSelectedItem());
             }
        }
    }
```

13.3 滚动列表

滚动列表是用户十分熟悉的组件.用户使用滚动列表中的上下箭头选择选项. java.awt 包中的类 List 类是专门用来建立滚动列表的,即 List 创建的一个对象就是一个滚动列表.滚动列表的大部分方法和下拉式列表相同 见下拉式列表方法的 2 ~ 13 ,需要注意的是,滚动列表可以允许选择多个选项,那么,如果滚动列表有多个选项处于选中状态,getSelectedIndex()方法和返回-l,getSelectedItem()方法返回 null.如果滚动列表允许多选,那么,public int[] getSelectedIndexes()方法返回当前滚动列表中被选中的所有选项的索引值的数组 public String[] getSelectedItems()方法返回选中的所有选项名字的字符串数组.滚动列表和下拉列表的另一不同之处是 滚动列表除了可以发生 ItemEvent 事件外,还可以发生 ActionEvent 事件.当用鼠标单击滚动列表的某个选项后,发生 ItemEvent 事件 当用鼠标双击某个选项后,发生 ActionEvent 事件.由于滚动列表可以发生 ItemEvent 事件和 ActionEvent 事件,所滚动列表类提供了 addItemListener 方法和 addActionListener 方法.

滚动列表的构造方法

1 List() 使用该构造方法创建的滚动列表有默认的可见行.

2 List(int n) 使用该构造方法可以创建一个滚动列表,起可见行数由参数 n 设置.

3 List(int n,boolean b) 使用该构造方法可以创建一个滚动列表,参数 n 设置可见行数,b 设置是否允许多项选择.

在下面例子 5 中有 2 个滚动列表.一个滚动列表 list1 添加了

"计算 1+2+3…","计算 1*1+2*2+3*3…","计算 1*1+2*2+3*3…",

3 个选项.另一个滚动列表 list2 添加了一百个选项.当单击 list1 上某个选项时,text1 显示该选项的名称.当在 lis2 上双击某个选项,程序计算和,例如 单击 list1 列表中的"计算 1*1+2*2+3*3…",然后双击 list2 列表的"前 68 项和",程序将计算连续 68 项的平方和.

例子 5

```
import java.applet.*;import java.awt.*;import java.awt.event.*;
public class Example13_6 extends Applet implements ItemListener,ActionListener
{  List list1,list2;
   TextArea text1,text2; int index=0;
   public void init()
   {  list1=new List(3,false); list2=new List(3,false);
      text1=new TextArea(6,15); text2=new TextArea(6,15);
      list1.add("计算 1+2+...");  list1.add("计算 1*1+2*2+3*3...");
      list1.add("计算 1*1*1+2*2*2+3*3*3...");
      for(int i=1;i<=100;i++)
       {  list2.add("前"+i+"项和");
       }
      add(list1);add(list2);add(text1);add(text2);
      list1.addItemListener(this); list2.addActionListener(this);
   }
   public void itemStateChanged(ItemEvent e)
   {  if(e.getItemSelectable()==list1)
      {  text1.setText(list1.getSelectedItem());
         index=list1.getSelectedIndex();
      }
   }
  public void actionPerformed(ActionEvent e)
   {  int n=list2.getSelectedIndex(),sum=0;
      String name=list2.getSelectedItem();
      switch(index)
        {  case 0:
                for(int i=1;i<=n+1;i++)
                { sum=sum+i;
                }
                break;
           case 1:
                for(int i=1;i<=n+1;i++)
                { sum=sum+i*i;
```

```
                }
            case 2:
                for(int i=1;i<=n+1;i++)
                {  sum=sum+i*i*i;
                }
                break;
             default :
                sum=-100;
          }
        text2.setText(name+"等于"+sum);
    }
}
```

习 题 十三

1　编写一个小应用程序,布局为 BorderLayout 布局,北面添加一个 Choice 组件,该组件有四个课程名称的选项.中心添加一个文本区,当选择 Choice 组件中的某个选项后,文本区显示对该课程的介绍.

2　编写应用程序,有一个窗口对象,该窗口取它的默认布局　BorderLayout 布局,北面添加一个 List 组件,该组件有四个商品名称的选项.中心添加一个文本区,当选择 List 组件中的某个选项后,文本区显示对该商品的价格和产地　当用鼠标双击 List 组件中的某个选项后,文本区显示该商品的明细.

第十四章　Component 类的常用方法

Component 类是所有组件的父类,这一节介绍 Component 类的常用方法.

组件都是矩形形状,组件本身有一个默认的坐标系,组件的左上角的坐标值是(0,0).如果一个组件的宽是 20,高是 10,那么,该坐标系中,x 坐标的最大值是 20　y 坐标的最大值是 10.如图 14.1 所示.

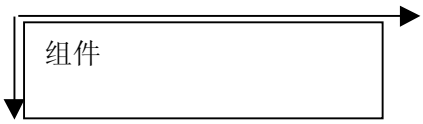


图 14.1　组件上的坐标系

14.1 组件的颜色

组件使用下列方法设置颜色

1　public void setBackground(Coloe c)　设置组件的背景色.

2　**public void setForeground(Coloe c)**　设置组件的前景色.

3　public Color getBackground(Color c) 获取组件的背景色.

4　public Color getForeground(Color c)　获取组件的前景色.

上述方法中都涉及到 Color 类,Color 类是 java.awt 包中的类,该类创建的对象称为颜色对象.

用 Color 类的构造方法　public Color(int red,int green,ing blue)可以创建一个颜色对象,其中 red, green,blue 的取值在 0 到 255 之间.另外,Color 类中还有 red,blue,green,orange,cyan,yellow,pink 等静态常量,都是颜色对象.

14.2 组件的字体

1　public void setFont(Font f)　组件调用该方法设置组件上的字体.例如,文本组件调用该方法可以设置文本组件中的字体.

2　public Font getFont(Font f)　组件调用该方法获取组件上的字体.

上述方法中用到了 java.awt 包中的 Font 类,该类创建的对象称为字体对象.Font 类的构造方法是

```
public Font(String name,int style,int size)
```

使用该构造方法可以创建字体对象,其中 name 是字体的名字,如果系统不支持字体的名字,将取默认的名字创建字体对象.style 决定字体的样式,取值是一个整数,有效取值是

Font.BOLD,Font.PLAIN,Font.ITALIC,

Font.ROMAN_BASELINE,Font.CENTER_BASELINE

Font.HANGING_BASELINE,Font.TRUETYPE_FONT.

例如,取值是 Font.BOLD 时,字体的样式是粗体.size 参数决定字体的大小,单位是磅,例如取值 12,就是我们熟悉的 5 号大小.

在创建字体对象时,应当给出一个合理的字体名字,也就是说,程序所在的计算机系统上有这样的字体名字.如果在创建字体对象时,没有给出一个合理的字体名字,那么该字体在特定平台的字体系统名称为默认名称.

如果想知道,计算机上有哪些字体名字可使用,可以使用 GraphicsEnvironment 对象调用

```
String [] getAvailableFontFamilyNames()
```

方法,该方法获取计算机上所有可用的字体名称,并存放到字符串数组中.

GraphicsEnviroment 类是 java.awt 包中的抽象类,不能用构造方法创建对象,Java 运行环境准备好了这个对象,只需让 GraphicsEnvironment 类调用它的类方法

```
public GraphicsEnvironment  static getLocalGraphicsEnvironment()
```

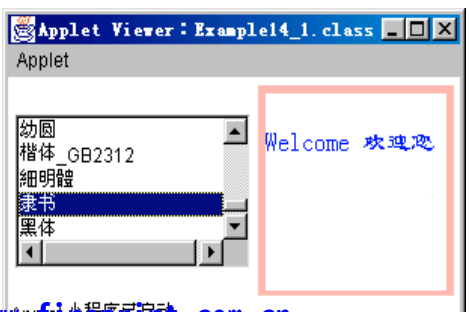
获取这个对象的引用即可,如下列代码所示

```
GraphicsEnvironment ge= GraphicsEnvironment getLocalGraphicsEnvironment();
String fontName[]=ge.getAvailableFontFamilyNames();
```

下面的例子 1 中,我们在一个滚动列表中列出全部可用字体名字,然后在滚动列表中选择字体名字,文本区用这种字体显示特定的文本"Welcome 欢迎您".

例子 1 效果如图 14.2

```
import java.applet.*;import java.awt.*;
import java.awt.event.*;
import javax.swing.JTextArea;
public class Example14_1 extends Applet implements ItemListener
{  List list  ;
   JTextArea text;
   public void init()
   {  list=new List(6,false);
     text=new JTextArea(6,15);text.setForeground(Color.blue);
     GraphicsEnvironment ge=GraphicsEnvironment.getLocalGraphicsEnvironment();
     String
fontName[]=ge.getAvailableFontFamilyNames();
      for(int i=0;i<fontName.length;i++)
        {  list.add(fontName[i]);
        }
```



```
        add(list);
        Panel p=new Panel();
        p.setBackground(Color.pink);
        p.add(text);
        add(p);
        list.addItemListener(this);
    }
    public void itemStateChanged(ItemEvent e)
    {  String name=list.getSelectedItem();
        Font f=new Font(name,Font.BOLD,16);
        text.setFont(f);
        text.setText("\nWelcome 欢迎您");
    }
}
```

注 字体名称只对 javax.swing 包中的 轻 组件有效 见二十五章 ,对于 java.awt 包中的组件,系统将取默认的字体名称.

14.3 组件的大小与位置

1 public void setSize(int width,int height) 组件调用该方法设置组件的大小,参数 width 指定组件的宽度,height 指定组件的高度.

2 public void setLocation(int x,int y) 组件调用该方法设置组件在容器中的位置,包含该组件的容器都有默认的坐标系,容器的坐标系的左上角的坐标是(0,0),参数 x,y 指定该组件的左上角在容器的坐标系中的坐标,即组件距容器的左边界 x 个像素,距容器的上边界 y 个像素.

3 public Dimension getSize() 组件调用该方法返回一个 Dimension 对象的引用,该对像实体中含有名字是 width 和 height 的成员变量,方法返回的 Dimmension 对象的 width 的值就是组件的宽度,height 的值就是当前组件的高度.

4 public Point getLocation(int x,int y) 组件调用该方法返回一个 Point 对象的引用,该对像实体中含有名字是 x 和 y 的成员变量,方法返回的 Point 对象的 x,y 的值就是组件的左上角在容器的坐标系中的 x 坐标和 y 坐标.

5 public void setBounds(int x,int y,int width,int height) 组件调用该方法设置组件在容器中的位置,和组件的大小.该方法相当于 setSize 方法和 setLocation 方法的组合.

6 public Rectangle getBounds() 组件调用该方法返回一个 Rectangle 对象的引用,该对像实体中含有名字是 x,y,width 和 height 的成员变量,方法返回的 Rectangle 对象的 x,y 的值就是组件的左上角在容器的坐标系中的 x 坐标和 y 坐标,width 和 height 的值就是当前组件的宽度和高度.

Rectangle 对象是一个很有用的对象.下面是 Rectangle 对象的常用方法.

- **Rectangle(int x,int y,int width.int height)** 创建一个左上角坐标是(x,y),宽是 width,高是 height 的矩形.
- **public boolean intersects(Rectangle rect)** 判断当前矩形是否和 rect 相交.
- **public boolean contains(int x,int y)** 判断点(x,y)是否在当前矩形内.
- **public boolean contains(int x,int y,int width,int height)** 判断当前矩形是否包含参数所指定的矩形.
- **public boolean contains(Rectangle rect)** 判断当前矩形是否包含参数所指定的矩形.

- **public Rectangle intersection(Rctangle rect)** 得到当前矩形与 rect 相交部分所构成的矩形,如果当前矩形和 rect 不相交,就返回 null.
- **public Rectangle union(Rectangle rect)** 得到同时包含当前矩形和 rect 的最小矩形.

14.4 组件的激活与可见性

1 public void setEnabled(boolean b) 组件调用该方法可以设置组件是否可被激活,当参数 b 取值 true 时,组件可以被激活,当参数 b 取值 false 时,组件不可激活.默认情况下,组件是可以被激活的.

2 public boolean isEnabled() 判断组件是否是可激活的,当组件是可激活状态时,该方法返回 true.

3 public void setVisible(boolean) 设置组件在该容器中的可见性,当参数 b 取值 true 时,组件在容器中可见,当参数 b 取值 false 时,组件在容器中不可见.除了 Window 型组件外,其它类型组件默认是可见的.

4 public boolean isVisible() 判断组件是否是可见的,当组件是可见时,该方法返回 true.

14.5 组件上的光标

1 public void setCursor(Cursor c) 设置鼠标指向组件时的光标形状.

使用 Cursor 类可以创建光标对象.Cousor 类中有许多类常量,它们是

HAND_CURSOR,CROSSHAIR_CURSOR,TEXT_CURSOR,WAIT_CURSOR,SW_RESIZE_CURSOR,
SE_RESIZE_CURSOR,NW_RESIZE_CURSOR,NE_RESIZE_CURSOR,N_RESIZE_CURSOR,
S_RESIZE_CURSOR,W_RESIZE_CURSOR,E_RESIZE_CURSOR,HAND_CURSOR,
MOVE_CURSOR,CUSTOM_CURSOR.

用这些类常量和类的构造方法可以创建标准的光标形状,例如

```
Cursor c=new Cursor(Cursor.HAND_CURSOR);
```

创建了一个"手"型的光标对象.

另外,我们也可以使用 Cursor 类的类方法直接获得一个光标对象,例如

```
Cursor c=Cursor.getPredefinedCursor(Cursor.HAND_CURSOR);
```

在下面的例子 2 中,有一个按钮 button 和一个标签.当单击按钮 button 时,该按钮横向移动,如果该按钮在移动过程中和标签相交,就将标签设置为不可见状态,然后按钮改为纵向移动.鼠标在容器区域时光标是 HAND 形状,当鼠标指向按钮时光标是 MOVE 形状.

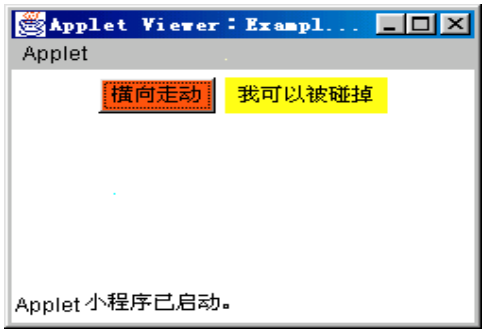


图 **14.3(a)** 按钮移动之前

图 **14.3(b)** 按钮移动之后

例子 2 效果如图 14.3(a),14.3(b)

```
import java.awt.*;import java.awt.event.*;
public class Example14_2 extends java.applet.Applet implements ActionListener
{  Button button;
   Label label;
   public void init()
   {  button=new Button("横向走动"); button.setBackground(Color.red);
      button.addActionListener(this);
      label=new Label("我可以被碰掉",Label.CENTER);
      label.setBackground(Color.yellow);
      add(button); add(label);
   }
 public void actionPerformed(ActionEvent e)
   {  Rectangle rect=button.getBounds();
      if(rect.intersects(label.getBounds()))
         {  label.setVisible(false);
         }
      if(label.isVisible())
         {  button.setLocation(rect.x+3,rect.y);
         }
      else
         {  button.setLocation(rect.x,rect.y+3);
            button.setLabel("纵向走动");
         }
```

```
    }
}
```

14.6 paint 方法与 repaint 方法

Component 类有一个方法 public void paint(Graphics g),我们可以在其子类中重写这个方法.当重写这个方法时,相应的 java 运行环境将参数 g 实例化,这样我们就可以在子类中使用 g 调用相应方法,比如画串,画图形 见 17 章 ,图象 见 22 章 等.

调用 repaint 方法时,程序首先清除 paint()方法以前所画的内容,然后再调用 paint()方法.实际上当我们调用 repaint()方法时,程序自动地去调用 update(Graphics g)方法,该方法也是 Component 类中的一个方法,这个方法的功能是 清除 paint()方法以前所画的内容,然后再调用 paint 方法.因此我们可以在子类中重写 update 方法 即隐藏父类的方法 ,根据需要来清除哪些部分或保留哪些部分.

在下面的例子 3 中,当单击"全部清除"按钮时,程序清除 paint 方法所绘制的全部内容,不再调用 paint 方法 当单击"部分清除"按钮时,程序清除 paint 方法所绘制的一部内容,不再调用 paint 方法.

例子 3

```
import java.awt.*;import java.awt.event.*;
class MyCanvas extends Canvas
{  int n=-1;
   MyCanvas()
   {  setSize(150,120);setBackground(Color.pink);
   }
   public void paint(Graphics g)
   {  g.setColor(Color.red);
      g.drawString("部分清除时我将消失",10,12);
      g.drawString("我们正在学习 repaint 方法",10,80);
   }
   public void setN(int n)
   {  this.n=n;
   }
   public void update(Graphics g)
   {  int width=0, height=0;
      width=getSize().width;height=getSize().height;
      if(n==0)
        {  g.clearRect(0,0,width,height);
           //paint(g); //如果取消该注释,update 的功能就与父类相同.
        }
      else if(n==1)
```

```
        {  g.clearRect(2,2,width,40);
        }
   }
}
public class Example14_3 extends java.applet.Applet implements ActionListener
{  Button b1,b2;MyCanvas canvas;
   public void init()
   {  canvas=new MyCanvas();
      b1=new Button("全部清除"); b1.addActionListener(this);
      b2=new Button("部分清除"); b2.addActionListener(this);
      add(b1); add(b2); add(canvas);
   }
   public void actionPerformed(ActionEvent e)
   {  if(e.getSource()==b1)
       {  canvas.setN(0);canvas.repaint();
       }
      if(e.getSource()==b2)
       {  canvas.setN(1);canvas.repaint();
       }
   }
}
```

习题十四

1　编写程序,观察各种组件设置背景色,和前景色的情况.

2　编写小应用程序,其布局为 null,在容器中有 2 个按钮,单击一个按钮让另一个按钮移动.

3　编写小应用程序,其布局为 null,在容器中有 3 个按钮和一个画布,3 个按钮的颜色分别是红,绿,蓝.单击相应的按钮,画布绘制相应颜色的圆.

4　编程程序,测试 Cursor 类中表示鼠标形状的静态常量.

第十五章　建立窗口和菜单

15.1 Java 窗口

Frame 类是 Container 类的间接子类.当需要一个窗口时,可使用 Frame 或其子类创建一个对象.窗口也是一个容器,可以向窗口添加组件.需要注意的是,窗口默认地被系统添加到显示器屏幕上,因此窗口不能和其它窗口嵌套,即不能将一个窗口添加到另一个窗口中.

Frame 有下列常用方法

1　Frame()　该构造方法可以创建一个无标题的窗口,窗口的默认布局的 BorderLayout 布局.

2　Frame(String s)　该构造方法可以创建一个标题为 s 的窗口,窗口的默认布局的 BorderLayout 布局.

3　public void setBounds(int a,int b,int width,int height)　窗口调用该方法可以设置出现在屏幕上时的的初始位置是(a,b),即距屏幕左面 a 个像素,距屏幕上方 b 个像素窗口的宽是 width,高是 height.

4　public void setSize(int width,int height)　设置窗口的大小,窗口在屏幕出现是默认位置是(0,0).

5　public void setVisible(boolean b)　设置窗口是可见还是不可见,窗口默认是不可见的.

6　setTitle(String s)　设置窗口的标题.

7　public void setResizable(boolean b) 设置窗口是否可调整大小,窗口默认是可调整大小的.

8　String getTitle()　获取窗口的标题

9　boolean isResizable()　获取窗口是否可调整大小的信息,当窗口可调整大小,该方法返回 true,否则返回 false.

10　public void dispose()　窗口调用该方法可以撤消当前窗口,并释放当前窗口所使用的资源.

11　public void　validate()　窗口调用该方法可以确保当前窗口中添加的组件能显示出来.窗口初始出现时有可能看不到窗口中的组件,当用户调整窗口大小时才能看到这些组件.如果窗口调用了该方法就不会发生这种情况.另外,当窗口调用方法 setSize 或 setBounds 调整大小后,都应调用方法 vaidate, 以确保当前窗口中添加的组件能显示出来.

下面的例子 1 中,我们写了一个 Frame 的子类,该子类创建的窗口中有按钮,文本区和选择框组件,可以通过按钮事件关闭窗口,通过选择框事件设置窗口是否可调整大小.

例子 1　效果如图 15.1

```
import java.awt.*;import java.awt.event.*;
class MyFrame extends Frame implements ItemListener,ActionListener
{  Checkbox box; TextArea text; Button button;
   MyFrame(String s)
   {  super(s);
      box=new Checkbox("设置窗口是否可调整大小");
      text=new TextArea(12,12);
      button=new Button("关闭窗口");
      button.addActionListener(this);
      box.addItemListener(this);
      setBounds(100,100,200,300);
      setVisible(true);
      add(text,BorderLayout.CENTER);
      add(box,BorderLayout.SOUTH);
      add(button,BorderLayout.NORTH);
      setResizable(false);
      validate();
   }
   public void itemStateChanged(ItemEvent e)
   {  if(box.getState()==true)
        {  setResizable(true);
        }
       else
        {  setResizable(false);
        }
   }
   public void actionPerformed(ActionEvent e)
   {  dispose();
   }
}
class Example15_1
{  public static void main(String args[])
   {  new MyFrame("窗口");
   }
}
```

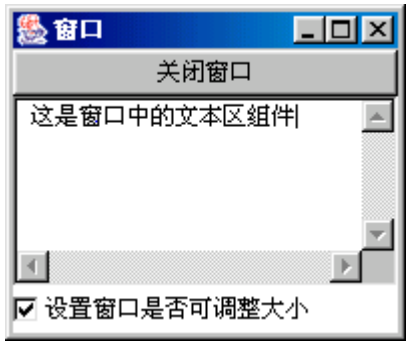


图 15.1　窗口对象

注　在运行上述程序时,你还无法通过单击窗口上的关闭图标使窗口消失或退出程序的运行,在本章讲述窗口事件时会解决这一问题.

15.2 窗口与屏幕

我们已经知道,当窗口可见时,它被自动出现在屏幕上,有时可能希望窗口和计算机的屏幕的大小相同或窗口的宽和屏幕的宽相同.那么,怎样知道屏幕的宽和高呢 这就需要 Toolkit 类帮忙.Toolkit 类是一个抽象类,不能用构造方法直接创建这样对象,但 Java 运行环境提供了一个 Toolkit 对象,任何一个组件调用 getToolkit 方法可以返回这个对象的引用.Tollkit 类中有一个方法

```
Dimension getScreenSize()
```

可以返回一个 Dimesion 对象,这个对象中有名字是 width,height 的 int 型属性,其中 width 的值就是屏幕的宽, height 的值就是屏幕的高.

在下面的例子 2 中,通过选择框事件将窗口的大小设置为屏幕的大小.

例子 2

```
import java.awt.*;import java.awt.event.*;
class MyFrame extends Frame implements ItemListener,ActionListener
{  Checkbox box; Button button;
   Toolkit tool; Dimension dim;
   MyFrame(String s)
   {  super(s);
      box=new Checkbox("设置窗口和屏幕同样大小");
      add(box,BorderLayout.SOUTH);
      button=new Button("关闭窗口"); button.addActionListener(this);
      box.addItemListener(this);
      setBounds(100,100,200,300); setVisible(true);
      add(box,BorderLayout.SOUTH); add(button,BorderLayout.NORTH);
      tool=getToolkit();
      validate();
   }
   public void itemStateChanged(ItemEvent e)
   {  if(box.getState()==true)
        {  dim=tool.getScreenSize();
           setBounds(0,0,dim.width,dim.height);
           validate();
        }
      else
        {  setBounds(0,0,dim.width,80);
           validate();
        }
```

```
    }
    public void actionPerformed(ActionEvent e)
    {  dispose();
    }
}
class Example15_2
{  public static void main(String args[])
   {new MyFrame("窗口");
    }
}
```

15.3 菜单条,菜单,菜单项

窗口中的菜单条,菜单,菜单项是我们所熟悉的界面,菜单放在菜单条里,菜单选项放在菜单里.

1 MenuBar 菜单条

java.awt 包中的 Menubar 类是负责创建菜单条的,即 Menubar 的一个实例就是一个菜单条.Frame 类有一个将菜单条放置到窗口中的方法

```
setMenuBar(MenuBar bar);
```

该方法将菜单条添加到窗口的顶端,需要注意的是,只能向窗口添加一个菜单条.

2 Menu 菜单

java.awt 包中的 Menu 类是负责创建菜单的,即 Menu 的一个实例就是一个菜单.Menu 类的主要方法

- Menu() 建立一个空标题的菜单.
- Menu(String s) 建立一个指定标题菜单,标题由参数 s 确定.
- public void add(MenuItem item) 向菜单增加由参数 item 指定的菜单选项对象.
- public void add(String s) 向菜单增加指定的选项.
- public MenuItem getItem(int n) 得到指定索引处的菜单选项.
- public int getItemCount() 得到菜单选项数目.
- public void insert(MenuItem item ,int n) 在菜单的指定位置插入菜单选项.
- public void insert(String s,int n) 在菜单指定位置插入菜单选项.
- public void remove(int n) 删除菜单的指定位置的菜单选项.
- public void removeAll() 删除菜单的所有选项.

3 MenuItem 菜单项

java.awt 包的 MunuItem 是 Menu 的父类,该类是负责创建菜单项的,即 MenuItem 的一个实例就是一个菜单项.菜单项将被放在菜单里.MenuItem 类的主要方法

- **MenuItem()** 构造无标题菜单项.
- **MenuItem(String s)** 构造有标题的菜单项.
- **public void setEnabled(boolean b)** 设置当前菜单项是否可被选择.
- **public String getLabel()** 得到菜单选项的名字.
- **public void addActionListener(ActionListener)** 向菜单项增加监视器,从菜单项接受行动事件(单击某个菜单项).

4 菜单项上的 ActionEvent 事件

单击某个菜单项可以发生 **ActionEvent** 事件.

下面的例子 3 中处理了菜单项事件,程序有一个窗口对象,用来计算常见的几何图形的面积.该窗口有一个菜单 "选择",其中有 2 个菜单项,当我们选择某个选项时,窗口的中心就会出现计算相应几何图形面积的用户界面.

例子 3 效果如图 15.2

```
import java.awt.*;import java.awt.event.*;
class 圆 extends Panel implements ActionListener//负责计算圆面积的类.
{  double r,area;
   TextField  半径=null,
              结果=null;
   Button b=null;
   圆()
   {  半径=new TextField(10);
      结果=new TextField(10);
      b=new Button("确定");
      add(new Label("输入半径"));
      add(半径);
      add(new Label("面积是:"));
      add(结果); add(b);
      b.addActionListener(this);
   }
   public void actionPerformed(ActionEvent e)
   {  try
         {  r=Double.parseDouble(半径.getText());
            area=Math.PI*r*r;
            结果.setText(""+area);
         }
      catch(Exception ee)
         {  半径.setText("请输入数字字符");
         }
```

图 15.2 处理菜单事件

```
   }
}
class 三角形 extends Panel implements ActionListener//负责计算三角形面积的类.
{  double a=0,b=0,c=0,area;
   TextField 边_a=new TextField(6),
             边_b=new TextField(6),
             边_c=new TextField(6),
             结果=new TextField(24);
   Button button=new Button("确定");
   三角形()
   {  add(new Label("输入三边的长度:"));
      add(边_a); add(边_b); add(边_c);
      add(new Label("面积是:"));
      add(结果); add(button);
      button.addActionListener(this);
   }
   public void actionPerformed(ActionEvent e)//获取三边的长度.
   {  try{ a=Double.parseDouble(边_a.getText());
           b=Double.parseDouble(边_b.getText());
           c=Double.parseDouble(边_c.getText());
           if(a+b>c&&a+c>b&&c+b>a)
            {  double p=(a+b+c)/2;
               area=Math.sqrt(p*(p-a)*(p-b)*(p-c));//计算三角形的面积.
               结果.setText(""+area);
            }
           else
            {  结果.setText("您输入的数字不能形成三角形");
            }
          }
      catch(Exception ee)
          { 结果.setText("请输入数字字符");
          }
   }
}
class Win extends Frame implements ActionListener
{  MenuBar bar=null; Menu menu=null;
   MenuItem item1, item2;
   圆  circle ;
   三角形 trangle;
```

```
    Win()
    {  bar=new MenuBar(); menu=new Menu("选择");
       item1=new MenuItem("圆面积计算"); item2=new MenuItem("三角形面积计算");
       menu.add(item1); menu.add(item2);
       bar.add(menu);
       setMenuBar(bar);
       circle=new 圆();
       trangle=new 三角形(); //创建一个圆和一个三角形.
       item1.addActionListener(this); item2.addActionListener(this);
       setVisible(true); setBounds(100,120,100,90);
    }
    public void actionPerformed(ActionEvent e)
    {  if(e.getSource()==item1)
          {  removeAll();
             add(circle,"Center");//添加圆面积计算的界面.
             validate();
          }
        else if(e.getSource()==item2)
          {  removeAll();
             add(trangle,"Center");//添加三角形面积计算的界面.
             validate();
          }
    }
}
public class Example15_3
{  public static void main(String args[])
    {  Win win=new Win();win.setBounds(100,100,200,100);win.setVisible(true);
       win.addWindowListener(new WindowAdapter()
            { public void windowClosing(WindowEvent e)
                { System.exit(0);
                }
            });
    }
}
```

15.4 有关菜单的几个技巧

1 增加菜单分割线

要增加菜单分割线,只需使用 Menu 类中的 addSeparator()方法,假设 menu1 是 Menu

的一个实例,”文件”,”新建”,”保存”,”打印”是它的三个菜单项的名字.如果你想在菜单项之间分隔线,只需在源文件中有这样的代码行

```
menu1.add(“新建”);
menu1.addSeparator();//分隔线
menu1.add(“保存”);
menu1.add(“打印”);
```

2 复选框菜单项

我们用 MenuItem 创建菜单项.如果想在选择这个菜单项时出现一个对号标记,你可以用 CheckboxMenuItem 类创建这个菜单项.如:

```
item1=new CheckboxMenuItem(“新建”);
```

3 嵌入子菜单

Menu 是 MenuItem 的子类,因此菜单项本身还可以是一个菜单,称这样的菜单项为子菜单.

4 设置菜单项的快捷键

可以使用 **MenuShotcut** 类为菜单项设置快捷键,该类的一个构造方法是

```
MenuShotcut (int key);
```

其中 key 可以取值 KeyEvent.VK_A~ KeyEvent.VK_Z.也可以取:’a’到’z’.关于键码值可查看 18.5 节中的键码表.一个菜单项使用 setShortcut(MenuShortcut k)方法来设置快捷键.

例子 4 中的 item 菜单项设置的快捷键是 E,进行 Ctrl+E 操作等同于单击该菜单项.

例子 4

```
import java.awt.*;import java.awt.event.*;
class Herwindow extends Frame  implements ActionListener
{  MenuBar menubar;Menu menu;MenuItem  item;
   MenuShortcut shortcut=new MenuShortcut(KeyEvent.VK_E);
   Herwindow(String s)
   {  super(s);
      setSize(160,170);setVisible(true);
      menubar=new MenuBar(); menu=new Menu("文件");
      item=new MenuItem("退出");
      item.setShortcut(shortcut);            //设置菜单选项的键盘快捷键.
      item.addActionListener(this);
      menu.add(item);
      menubar.add(menu);menubar.add(menu);
       setMenuBar(menubar);
```

```
    }
    public void actionPerformed(ActionEvent e)
    {  if(e.getSource()==item)
        {  System.exit(0);
        }
    }
}
public class Example15_4
{  public static void main(String args[])
    {  Herwindow window=new Herwindow("法制之窗");
    }
}
```

15.5 窗口事件

Frame 是 Window 的子类,凡是 Window 子类创建的对象都可以发生 WindowEvent 类型事件,即窗口事件.

1 WindowListener 接口

当一个 Frame 窗口被激活,撤消激活,打开,关闭,图标化或撤消图标化时,就发生了窗口事件,即 WindowEvent 创建一个窗口事件对象.WindowEvent 创建的事件对象调用 getWindow 方法可以获取发生窗口事件的窗口.窗口使用 addWindowlistener 方法获得监视器,创建监视器对象的类必须实现 WindowListener 接口,该接口中有 7 个不同的方法,分别是

- **public void windowActivated(WindowEvent e) 当窗口从非激活状态到激活时,窗口的监视器调用该方法.**
- **public void windowDeactivated(WindowEvent e) 当窗口激活状态到非激活状态时,窗口的监视器调用该方法.**
- **public void windowClosing(WindowEvent e) 当窗口正在被关闭时,窗口的监视器调用该方法.**
- **public void windowClosed(WindowEvent e) 当窗口关闭时,窗口的监视器调用该方法.**
- **public void windowIconified(WindowEvent e) 当窗口图标化时,窗口的监视器调用该方法.**
- **public void windowDeiconified(WindowEvent e) 当窗口撤消图标化时,窗口的监视器调用该方法.**
- **public void windowOpened(WindowEvent e) 当窗口打开时,窗口的监视器调用该方法.**

注 当单击窗口上的关闭图标时,监视器首先调用 windowClosing 方法,如果在该方法中使用

```
System.exit(0);
```

推出程序的运行,那么监视器就没有机会再调用 windowClosed 方法.

当单击窗口的图标化按钮时,监视器调用 windowIconified 后,还将调用 windowDeactivated.当撤消窗口图标化时,监视器调用 windowDeiconified 方法后还会调用 windowActivated 方法.

下面的例子 5 处理窗口事件,在窗口的文本区中记录发生的事件.

例子 5 效果如图 15.3

```
import java.awt.*;import java.awt.event.*;
class MyFrame extends Frame implements WindowListener
{  TextArea text;
   MyFrame(String s)
   {  super(s);
      setBounds(100,100,200,300);
      setVisible(true);
      text=new TextArea();
      add(text,BorderLayout.CENTER);
      addWindowListener(this);
      validate();
   }
   public void windowActivated(WindowEvent e)
   {  text.append("\n我被激活");
      validate();
   }
   public void windowDeactivated(WindowEvent e)
   {  text.append("\n我不是激活状态了");
      setBounds(0,0,400,400); validate();
   }
   public void windowClosing(WindowEvent e)
   {  text.append("\n窗口正在关闭呢"); dispose();
   }
   public void windowClosed(WindowEvent e)
   {  System.out.println("程序结束运行");
      System.exit(0);
   }
   public void windowIconified(WindowEvent e)
   {  text.append("\n我图标化了");
   }
```

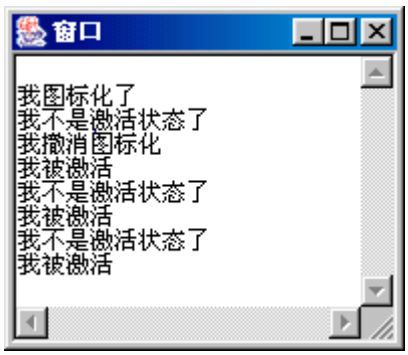


图 15.3 处理窗口事件

```
    public void windowDeiconified(WindowEvent e)
    {  text.append("\n我撤消图标化");
       setBounds(0,0,400,400);validate();
    }
    public void windowOpened(WindowEvent e){ }
}
class Example15_5
{ ublic static void main(String args[])
    {  new MyFrame("窗口");
    }
}
```

15.6 WindowAdapter 适配器

我们知道,当一个类实现一个接口时,即使不准备处理某个方法,也必须给出接口中所有方法的实现.适配器可以代替接口来处理事件,当 Java 提供处理事件的接口中多于一个方法时,Java 相应地就提供一个适配器类,比如 WindowAdapter 类.适配器已经实现了相应的接口,例如 WindowAdapter 实现了 WindowListener 接口.因此,可以使用 WindowAdapte 的子类创建的对象做监视器,在子类中重写所需要的接口方法即可.

在下面的例子 6 中,只处理窗口关闭事件和激活事件,我们使用适配器做监视器,只需重写 windowColsing 方法和 windowActivated 既可.

例子 6

```
import java.awt.*;import java.awt.event.*;
class MyFrame extends Frame
{  TextArea text;  Boy police;
   MyFrame(String s)
   {  super(s);
      police=new Boy(this);
      setBounds(100,100,200,300); setVisible(true);
      text=new TextArea(); add(text,BorderLayout.CENTER);
      addWindowListener(police); validate();
   }
}
class Boy extends WindowAdapter
{  MyFrame f;
   public Boy(MyFrame f)
   {  this.f=f;
   }
   public void windowActivated(WindowEvent e)
```

```
   {  f.text.append("\n我被激活");
   }
   public void windowClosing(WindowEvent e)
   {  System.exit(0);
   }
}
class Example15_6
{  public static void main(String args[])
   {  new MyFrame("窗口");
   }
}
```

适配器经常象在下面例子 7 中那样使用.

例子 7

```
import java.awt.*;import java.awt.event.*;
class MyFrame extends Frame
{  TextArea text;
   MyFrame(String s)
   {  super(s);
      setBounds(100,100,200,300);setVisible(true);
      text=new TextArea(); add(text,BorderLayout.CENTER);
      addWindowListener(new WindowAdapter()
                     {   public void windowActivated(WindowEvent e)
                          { text.append("\n我被激活");
                          }
                          public void windowClosing(WindowEvent e)
                          {  System.exit(0);
                          }
                     }
                  );
      validate();
   }
}
class Example15_7
{  public static void main(String args[])
   {  new MyFrame("窗口");
   }
}
```

在下面的小应用程序中,有两个窗口,用关闭图标钮关闭"音乐之窗"时,"体育之窗"被打

开　用关闭图标钮关闭”体育之窗”时,” 音乐之窗”又被打开.

例子 8

```
import java.applet.*;import java.awt.*;import java.awt.event.*;
public class Example15_8 extends Applet
{  Frame f,g;
   public void init()
   {  f=new Frame("音乐之窗");g=new Frame("体育之窗");
      f.setSize(150,150); f.setVisible(true);
      g.setSize(200,200); g.setVisible(false);
      f.addWindowListener(new WindowAdapter()
                  {  public void windowClosing(WindowEvent e)
                     { f.setVisible(false);
                       g.setVisible(true);
                     }
                  } );          //适配器
      g.addWindowListener(new WindowAdapter()
                    {public void windowClosing(WindowEvent e)
                       {  g.setVisible(false);
                          f.setVisible(true);
                       }
                    } );
   }
}
```

15.7 打印

我们可以在应用程序中打印,,首先获得一个 **PrintJob** 对象.假如获得了一个 **PrintJob** 对象 **p**,那么 **p** 可以使用 **getGraphics()**方法获得一个 **Graphics** 对象 **g.**

这时如果一个容器调用方法　从 **Component** 类继承下来的方法

```
paintAll(g);
```

将打印出该容器及其子组件　添加到该容器中的组件　.如果调用方法

```
pain(g);
```

将打印出该容器本身,但不打印子组件.

如果一个非容器组件调用方法

```
paintAll(g);
```

将打印出组件及其组件中的内容,比如按扭的名字,文本框中的文字等等.如果调用方法

```
pain(g);
```

将打印出该组件本身的形状,但不打组件上的其它信息,比如文字图形等.

打印机在打印你的组件时总是从打印页的左上角开始打印你程序中的组件,如果程序中有两个如下的语句

```
按扭_1.paintAll(g); 按扭_2.paintAll(g);
```

按扭_2 就会被打印在按扭_1 的上面,盖住按扭_1.为了避免这种情况的发生,g 可以使用 Graphics 类中的方法

```
translate(int x,int y);
```

改变组件在打印页中打印的位置,比如

```
按扭_1.paintAll(g);
g.translate(78,0);  //在打印页的(78,0)坐标处开始打印按扭_2.
按扭_2.paintAll(g);
```

当运行应用程序选择打印时,系统会打开我们熟悉的打印对话框.需要注意的是 PrintJob 是 java.awt 包中的一个 abstract 类,我们不能用它直接创建对象.Java.awt 包中有个抽象类 Toolkit,可以帮助我们获得一个 PringJob 对象.Toolkit 类有一个获得 PrintJob 对象的方法 getPrintJob(Frame f,Striong s,null),而任何一个组件都可以使用 getToolkit()方法获得一个 Toolkit 对象.

在下面的应用程序窗口中有一个文本区和三个按扭 打印窗口,打印文本区,打印按扭.点击相应的按扭会产生不同的打印操作,图 15.4 是打印出的 3 个按钮.

图 15.4 打印出的三个按扭

例子 9 (效果如图 15.4)

```
import java.awt.*;import java.awt.event.*;
public class Example15_9
{  public static void main(String args[])
   {  MyFrame f=new MyFrame();
      f.setBounds(70,70,70,89);f.setVisible(true);f.pack();
   }
}
class MyFrame extends Frame implements ActionListener
{  PrintJob p=null; //声明一个 PrintJob 对象.
   Graphics g=null;
   TextArea text=new TextArea(10,10);
```

```
    Button 打印文本框=new Button("打印文本框"),
           打印窗口=new Button("打印窗口"),
           打印按扭=new Button("打印按扭");
    MyFrame()
    {  super("在应用程序中打印");
       打印文本框.addActionListener(this);
       打印窗口.addActionListener(this);
       打印按扭.addActionListener(this);
       add(text,"Center");
       Panel panel=new Panel();
       panel.add(打印文本框); panel.add(打印窗口); panel.add(打印按扭);
       add(panel,"South");
       addWindowListener(new WindowAdapter()
            {public void windowClosing(WindowEvent e)
               {System.exit(0); }
            });
    }
public void actionPerformed(ActionEvent e)
    {  if(e.getSource()==打印文本框)
        {  p=getToolkit().getPrintJob(this,"ok",null);
                                    //创建一个 PrintJob 对象 p .
           g=p.getGraphics();    //p 获取一个用于打印的 Graphics 对象.
           g.translate(120,200);
           text.printAll(g);
           g.dispose();          //释放对象 g.
           p.end();
        }
       else if(e.getSource()==打印窗口)
        {  p=getToolkit().getPrintJob(this,"ok",null);
           g=p.getGraphics();    //p 获取一个用于打印的 Graphics 对象.
           g.translate(120,200);
           this.printAll(g);      //打印当前窗口及其子组件.
           g.dispose();          //释放对象 g.
           p.end();
        }
      else if(e.getSource()==打印按扭)
         {  p=getToolkit().getPrintJob(this,"ok",null);
            g=p.getGraphics();
            g.translate(120,200); //在打印页的坐标(120,200)处打印第一个"按扭".
```

```
                打印文本框.printAll(g);
                g.translate(78,0);    //在打印页的坐标(198,200)处打印第二个"按扭".
                打印窗口.printAll(g);
                g.translate(66,0);    //在打印页的坐标(264,200)处打印第三个"按扭".
                打印按扭.printAll(g);
                g.dispose();
                p.end();
            }
        }
    }
```

15.8 使用剪贴板

剪贴板是我们很熟悉的概念,我们经常将一个程序中的某些数据先剪切或拷贝到系统剪贴板,然后再将剪贴板中的数据粘贴本程序或其它的程序中,例如我们可以将程序 Word 中的某些文本数据复制或剪切到剪贴板然后再粘贴到程序 Word 中或记事本中.

java.awt.datatransfer 包提供的类只能使我们实现将程序中的字符串数据复制或剪切到系统剪贴板.

1 java.awt.datatransfer 包中的 Clipboard 类

如果准备将文本数据复制或剪切到系统剪贴板,必须首先使用 Clipboard 声明一个剪贴板对象

```
Clipboard clipboad;
```

然后让某个组件调用方法 getToolkit()获取一个 Toolkit 对象,该对象再调用 getSystemClipboad()方法获取系统的剪贴板

```
clipboad=getToolkit().getSystemClipboad();
```

现在就可以将文本数据复制或剪切到 clipboard 中了,首先用 StringSelection 创建一个对象,把待复制或剪切的字符串 temp 传递给该对象,如下所示

```
StringSelection text=new StringSelection(temp);
```

然后将对象 text 放入剪贴板

```
clipboard.setContents(text,null);
```

上述语句中 setContenets 方法的第 2 个参数应当设置为剪贴板的拥有者,但对于文本的复制或剪切可以不考虑剪贴板的拥有者,这个参数可取 null.

2 从剪贴板取数据到 Java 程序中

clipboard 可以使用 getContents(Component b)方法获取剪贴板中的数据,把取回的数据看作是一个 Transferable 类型的数据,因此必须使用如下语句

```
Transferable contents=clipboard.getContents(new Button());
```

Transferable 对象可以告诉我们那些风格的剪贴板信息是 **Java** 可用的,对于系统剪贴板只能使用标准风格

```
DataFlavor  flavor= DataFlavor.stringFlavor;
```

对象 **contents** 可以使用方法 **isDataFlavorSupported(Flavor flavor)**来判断目前所选风格是否可用

```
contents.isDataFlavorSupported( flavor);
```

现在 **contents** 就可以用方法 **getTransferData(Flavor flavor)**,根据指定的风格获取剪贴板中的数据了.方法 **getTransferData(Flavor flavor)**可能产生 **Exception** 异常,如下所示

```
try{  String text=(String)contents.getTransferData(flavor);
   }
catch(Exception e){}
```

在下面的例子 **10** 中我们将一个文本区中选中的文本复制到剪贴板并再粘贴到另一个文本区.

例子 10

```
import java.awt.*;import java.awt.event.*;
import java.awt.datatransfer.*;
public class Example15_10 extends Frame implements ActionListener
{  MenuBar menubar; Menu menu;
   MenuItem copy,cut,paste;
   TextArea text1,text2;
   Clipboard clipboard=null;
   Example15_10()
   {  clipboard=getToolkit().getSystemClipboard();//获取系统剪贴板.
      menubar=new MenuBar();
      menu=new Menu("Edit");  copy=new MenuItem("copy");
      cut=new MenuItem ("cut");  paste=new MenuItem ("paste");
      text1=new TextArea(20,20); text2=new TextArea(20,20);
      copy.addActionListener(this); cut.addActionListener(this);
      paste.addActionListener(this);
      setLayout(new FlowLayout());
      menubar.add(menu);
      menu.add(copy);  menu.add(cut); menu.add(paste);
      setMenuBar(menubar);
      add(text1);add(text2);
```

```
        setBounds(100,100,200,250); setVisible(true);pack();
        addWindowListener(new WindowAdapter()
                {public void windowClosing(WindowEvent e)
                   {System.exit(0);
                   }
                }) ;
    }
    public void actionPerformed(ActionEvent e)
    {  if(e.getSource()==copy)                    //拷贝到剪贴板.
         {  String temp=text1.getSelectedText();  //拖动鼠标选取文本.
            StringSelection text=new StringSelection(temp);
            clipboard.setContents(text,null);
         }
       else if(e.getSource()==cut)                //剪贴到剪贴板.
        {  String temp=text1.getSelectedText();   //拖动鼠标选取文本.
           StringSelection text=new StringSelection(temp);
           clipboard.setContents(text,null);
           int start=text1.getSelectionStart();
           int end  =text1.getSelectionEnd();
           text1.replaceRange("",start,end) ; //从 Text1 中删除被选取的文本.
        }
       else if(e.getSource()==paste)         //从剪贴板粘贴数据.
       {  Transferable contents=clipboard.getContents(this);
          DataFlavor  flavor= DataFlavor.stringFlavor;
          if( contents.isDataFlavorSupported(flavor))
            try{  String str;
                  str=(String)contents.getTransferData(flavor);
                  text2.append(str);
               }
            catch(Exception ee){}
        }
    }
    public static void main(String args[])
    {  Example15_10 win=new Example15_10 ();
    }
}
```

习题十五

1　Frame 类的对象的默认布局是什么布局　和 Panel 类对象的默认布局相同吗

2　创建 Frame 窗口对象时可以不设置该窗口的大小吗

3　一个容器对象是否可以使用 add 方法添加一个 Frame 窗口　窗口可以嵌套吗

4　编写一个应用程序,有一个 Frame 窗口,窗口的中心添加了一个文本区.该窗口中有四个菜单.每个菜单里都有菜单项,每个菜单项都对应有快捷键,选择某个菜单项时窗口中心的文本区显示某些信息.

5　上机实习下列程序,学会弹出式菜单的使用.

```
import java.awt.*;import java.awt.event.*;
public class PopupMenuTest extends Frame implements ActionListener
{  Button b=null;PopupMenu menu;
   PopupMenuTest()
   {  menu =new PopupMenu("我是弹出式菜单,是 Menu 的子类");
      menu.add(new MenuItem("copy"));menu.add(new MenuItem("paste"));
      menu.add(new MenuItem("save"));add(menu);
      b=new Button("ok");add(b,"North");b.addActionListener(this);
      setSize(200,200);setVisible(true);
      addWindowListener(new WindowAdapter()
               {public void windowClosing(WindowEvent e)
                   {System.exit(0);
                   }
               }) ;
   }
   public void actionPerformed(ActionEvent e)
   {  menu.show(b,20,5);  //show()显示时,第一个参数必须是一个组件,使得菜单从该
                          //组件的坐标系统的(20,5)位置弹出.
    }
   public static void main(String args[])
   {  new PopupMenuTest();
   }
}
```

第十六章　建立对话框

16.1 Dialog 类

Dialog 类和 Frame 都是 Window 的子类,二者有相似之处也有不同的地方,比如 Dialog 没有添加菜单的功能,而且对话框必须要依赖于某个窗口或组件,当它所依赖的窗口或组件消失,对话框也将消失　而当它所依赖的窗口或组件可见时,对话框又会自动恢复.

创建对话框与创建窗口类似,通过建立 Dialog 的子类来建立一个对话框类,然后这个类的一个实例,即这个子类创建的一个对象,就是一个对话框.对话框也是一个容器,它的默认布局是 BorderLayout,对话框可以添加组件,实现与用户的交互操作.

1　Dialod 类的主要方法

1　Dialod(Frame f,String s)　构造一个具有标题的初始不可见的对话框,参数 s 是标题的名字,f 是对话框所依赖的窗口.

2　Dialod(Frame f,String s,boolean b)　构造一个具有标题 s 的初始不可见的对话框.参数 b 决定对话框是否为有模式或无模式,参数 f 是对话框所依赖的窗口.

3　getTitle()　获取对话框的标题.

4　setTitle()　设置对话框的标题.

5　setModal(boolean)　设置对话框的模式.

6　setSize()　设置对话框的大小.

7　setVisible(boolean b)　显示或隐藏对话框.

2　对话框的模式

对话框分为无模式和有模式两种.

如果一个对话框是有模式的对话框,那么当这个对话框处于激活状态时,只让程序响应对话框内部的事件,程序不能再激活它所依赖的窗口或组件,而且它将堵塞其它线程的执行,直到该对话框消失不可见.

无模式对话框处于激活状态时,程序仍能激活它所依赖的窗口或组件,它也不堵塞线程的执行.

在下面的例子 1 中,当对话框处于激活状态时,文本区 text 中无法显示信息,当对话框消失时,再根据对话框消失的原因,文本区 text 分别显示信息　"你单击了对话框的 Yes 按钮"或"你单击了对话框的 No 按钮"

例子 1　效果如图 16.1

```
import java.awt.event.*; import java.awt.*;
class MyDialog extends Dialog implements ActionListener  //建立对话框类.
{  static final int YES=1,NO=0;
```

```
    int message=-1; Button yes,no;
    MyDialog(Frame f,String s,boolean b) //构造方法.
    {  super(f,s,b);
       yes=new Button("Yes"); yes.addActionListener(this);
       no=new Button("No");   no.addActionListener(this);
       setLayout(new FlowLayout());
       add(yes); add(no);
       setBounds(60,60,100,100);
       addWindowListener(new WindowAdapter()
                         {   public void windowClosing(WindowEvent e)
                             { message=-1;setVisible(false);
                             }
                         }
                        );
    }
    public void actionPerformed(ActionEvent e)
    {  if(e.getSource()==yes)
       { message=YES;setVisible(false);
       }
      else if(e.getSource()==no)
       { message=NO;setVisible(false);
       }
    }
    public int getMessage()
    {  return message;
    }
}
class Dwindow extends Frame implements ActionListener
{  TextArea text; Button button; MyDialog dialog;
   Dwindow(String s)
   {  super(s);
      text=new TextArea(5,22); button=new Button("打开对话框");
      button.addActionListener(this);
      setLayout(new FlowLayout());
      add(button); add(text);
      dialog=new MyDialog(this,"我有模式",true);
      setBounds(60,60,300,300); setVisible(true);
      validate();
      addWindowListener(new WindowAdapter()
```

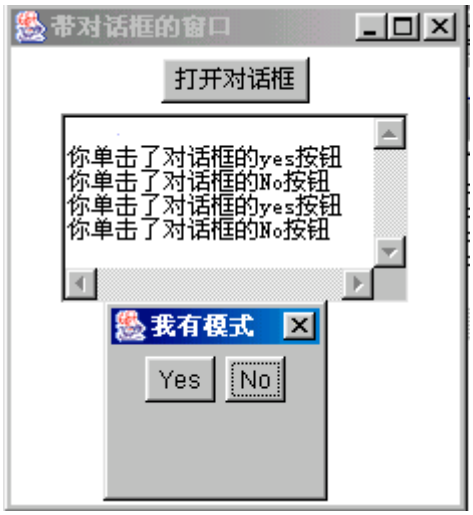


图 16.1　有模式的对话框

```
                    {   public void windowClosing(WindowEvent e)
                        { System.exit(0);
                        }
                    }
                );
    }
    public void actionPerformed(ActionEvent e)
    {  if(e.getSource()==button)
        {  dialog.setVisible(true); //对话框激活状态时,堵塞下面的语句.
          //对话框消失后下面的语句继续执行
           if(dialog.getMessage()==MyDialog.YES)   //如果单击了对话框的"yes"按钮.
             {  text.append("\n你单击了对话框的yes按钮");
             }
           else if(dialog.getMessage()==MyDialog.NO)   //如果单击了对话框的"no"按钮.
            {  text.append("\n你单击了对话框的No按钮");
            }
        }
    }
}
public class Example16_1
{  public static void main(String args[])
    {  new Dwindow("带对话框的窗口");
    }
}
```

注 在进行一个重要的操作动作之前,最好能弹出一个有模式的对话框.6.14.2 文件对话框.

16.2 文件对话框

FileDialog 是 Dialog 类的子类,它创建的对象称为文件对话框.文件对话框是一个打开文件和保存文件的有模式对话框.文件对话框必须依附一个 Frame 对象.

FileDialog 类有下列主要方法

1 FileDialog(Frame f,String s, int mode) 构造方法,参数 f 是所创建的对话框所依赖的窗口对象,s 是对话框的名字,mode 的取值为 FileDialog.LOAD 或 FileDialod.SAVE,决定对话是打开文件模型或保存文件模型,如图 16.2 所示.

2 public String getDirectory() 获取当前文件对话框中显示的文件的所属目录.

3 public String getFile() 获取当前文件对话框中显示的文件的字符串表示,如果不存在就得到 null.

当文件对话框处于激活状态时,在”文件名”输入栏中输入文件名之后,无论点击对话框中的”打开”按钮或”取消”按钮,对话框都自动消失,不能实现对文件的打开或保存操作,因为文件对话框仅仅提供了一个文件操作的界面,要真正实现对文件的操作,应学习文件的输入输出流　见二十章　.需要注意的是,只有单击了文件对话框上的批准按钮后　”打开”,”保存”　,getFile()才能得到非空的字符串对象

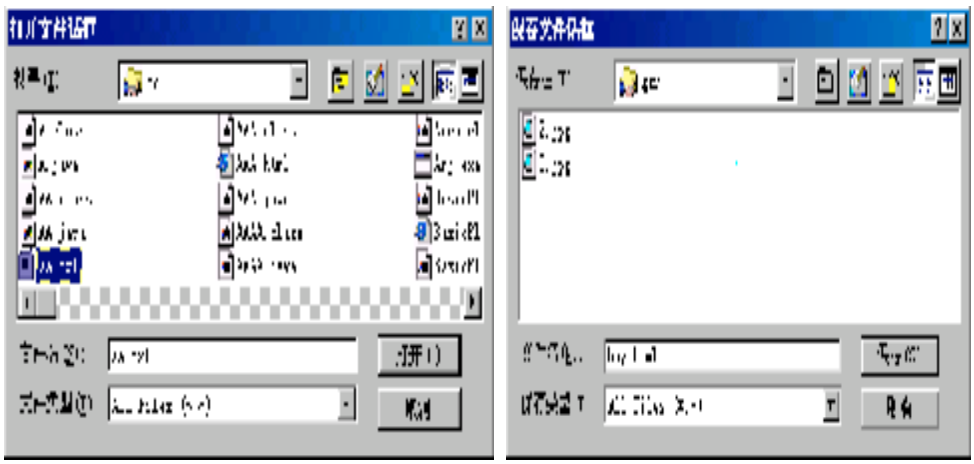


图 16.2 文件对话框

在下面的例子 2 中,一个窗口带有”保存文件对话框”和”打开文件对话框”.窗口还有一个菜单,当我们选择菜单中的”打开文件”选项时,出现”打开文件对话框”　选择菜单中的”保存文件”选项时,出现”保存文件对话框”.

例子 2

```
import java.awt.*;import java.awt.event.*;
public class Example16_2
{  public static void main(String args[])
   {  FWindow f=new FWindow("窗口");
   }
}
class  FWindow extends Frame implements ActionListener
{  FileDialog filedialog_save,
              filedialog_load;//声明 2 个文件对话筐
   MenuBar menubar;
   Menu menu;
   MenuItem itemSave,itemLoad;
   TextArea text;
   FWindow(String s)
   {  super(s);
      setSize(300,400);setVisible(true);
```

```
    text=new TextArea(10,10);
    add(text,"Center"); validate();
    menubar=new MenuBar();menu=new Menu("文件");
    itemSave=new MenuItem("保存文件"); itemLoad=new MenuItem("打开文件");
    itemSave.addActionListener(this); itemLoad.addActionListener(this);
    menu.add(itemSave); menu.add(itemLoad);
    menubar.add(menu);
    setMenuBar(menubar);
    filedialog_save=new FileDialog(this,"保存文件话框",FileDialog.SAVE);
    filedialog_save.setVisible(false);
    filedialog_load=new FileDialog(this,"打开文件话框",FileDialog.LOAD);
    filedialog_load.setVisible(false);
    filedialog_save.addWindowListener(new WindowAdapter()//对话框增加适配器.
            {  public void windowClosing(WindowEvent e)
                 {  filedialog_save.setVisible(false);
                 }
            });
    filedialog_load.addWindowListener(new WindowAdapter()//对话框增加适配器.
           {public void windowClosing(WindowEvent e)
                 { filedialog_load.setVisible(false);
                 }
           });
    addWindowListener(new WindowAdapter()  //窗口增加适配器.
          {public void windowClosing(WindowEvent e)
                { setVisible(false);System.exit(0);
                }
          });
}
public void actionPerformed(ActionEvent e) //实现接口中的方法.
{  if(e.getSource()==itemSave)
      {  filedialog_save.setVisible(true);
         String name=filedialog_save.getFile();
         if(name!=null)
           {  text.setText("你选择了保存文件,名字是"+name);
           }
         else
          {  text.setText("没有保存文件");
          }
      }
```

```
        else if(e.getSource()==itemLoad)
         {  filedialog_load.setVisible(true);
             String name=filedialog_load.getFile();
            if(name!=null)
              {  text.setText("你选择了打开文件,名字是"+name);
              }
            else
             { text.setText("没有打开文件");
             }
         }
    }
}
```

16.3 消息对话框

消息对话框是有模式对话框,进行一个重要的操作动作之前,最好能弹出一个消息对话框.可以用 javax.swing 包中的 JOptionPane 类的静态方法:

```
public static void showMessageDialog(Component parentComponent,
                                      String message,
                                      String title,
                                      int messageType)
```

创建一个消息对话框,其中参数 parentComponent 指定消息对话框所依赖的组件,消息对话框会在该组件的正前方显示出来 message 指定对话框上显示的消息 title 指定对话框的标题 messageType 取下列有效值

```
JOptionPane.INFORMATION_MESSAGE
JOptionPane.WARNING_MESSAGE
JOptionPane.ERROR_MESSAGE
JOptionPane.QUESTION_MESSAGE
JOptionPane.PLAIN_MESSAGE
```

这些值可以确定对话框的外观,例如,取值 JOptionPane.WARNING_MESSAGE 时,对话框的外观上会有一个明显的" "符号.

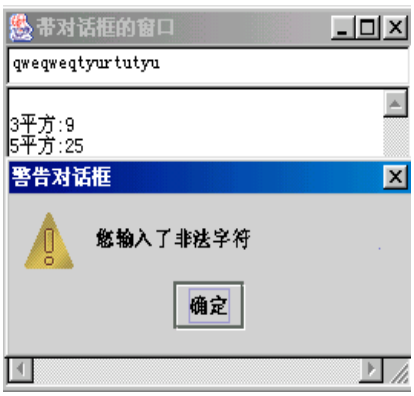


图 16.3 消息对话框

在下面的例子 3 中,要求用户在文本框中只能输入数字字符,当输入非数字字符时,弹出"警告"消息对话框.

例子 3 效果如图 16.3

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

```
class Dwindow extends Frame implements ActionListener
{  TextField inputNumber;
   TextArea  show;
   Dwindow(String s)
   {  super(s);
      inputNumber=new TextField(22); inputNumber.addActionListener(this);
      show=new TextArea();
      add(inputNumber,BorderLayout.NORTH); add(show,BorderLayout.CENTER);
      setBounds(60,60,300,300); setVisible(true);
      validate();
      addWindowListener(new WindowAdapter()
                     {   public void windowClosing(WindowEvent e)
                          { System.exit(0);
                          }
                     }
                  );
   }
   public void actionPerformed(ActionEvent e)
   {  boolean boo=false;
      if(e.getSource()==inputNumber)
        {  String s=inputNumber.getText();
           char a[]=s.toCharArray();
           for(int i=0;i<a.length;i++)
              {  if(!(Character.isDigit(a[i])))
                     boo=true;
              }
           if(boo==true)    //弹出"警告"消息对话框.
              {  JOptionPane.showMessageDialog(this,"您输入了非法字符","警告对话框",
                                               JOptionPane.WARNING_MESSAGE);
                 inputNumber.setText(null);
              }
           else if(boo==false)
              {  int number=Integer.parseInt(s);
                 show.append("\n"+number+"平方:"+(number*number));
              }
        }
   }
}
public class Example16_3
```

```
{  public static void main(String args[])
   {  new Dwindow("带对话框的窗口");
   }
}
```

16.4 确认对话框

确认对话框是有模式对话框,可以用 javax.swing 包中的 JOptionPane 类的静态方法:

```
public static int showConfirmDialog(Component parentComponent,Object message,
                                    String title,int optionType)
```

创建一个确认对话框,其中参数 parentComponent 指定消息对话框所依赖的组件,确认对话框会在该组件的正前方显示出来 message 指定对话框上显示的消息 title 指定对话框的标题 messageType 取下列有效值

JOptionPane.YES_NO_OPTION
JOptionPane.YES_NO_CANCEL_OPTION
JOptionPane.OK_CANCEL_OPTION

这些值可以确定对话框的外观,例如,取值 JOptionPane.YES_NO_OPTION 时,对话框的外观上会有"Yes","No"2 个按钮.当对话框消失后,showConfirmDialog 方法会返回下列整数值之一

JOptionPane.YES_OPTION
JOptionPane.NO_OPTION
JOptionPane.CANCEL_OPTION
JOptionPane.OK_OPTION
JOptionPane.CLOSED_OPTION

返回的具体值依赖于用户单击了对话框上的哪个按钮以及对话框上的关闭图标.
在下面的例子 4 中,用户在文本框中输入姓名,按回车后,将弹出一个"确认"对话框.
如果单击对话框上的"Y"按钮,就将名字放入文本区.

例子 4 效果如图 16.4

```
import java.awt.event.*;import java.awt.*;
import javax.swing.JOptionPane;
class Dwindow extends Frame implements ActionListener
{  TextField inputName;
   TextArea  save;
   Dwindow(String s)
   {  super(s);
      inputName=new TextField(22);inputName.addActionListener(this);
```

```
        save=new TextArea();
        add(inputName,BorderLayout.NORTH); add(save,BorderLayout.CENTER);
        setBounds(60,60,300,300); setVisible(true);
        validate();
        addWindowListener(new WindowAdapter()
                        {   public void windowClosing(WindowEvent e)
                             { System.exit(0);
                             }
                        }
                      );
    }
    public void actionPerformed(ActionEvent e)
    {  String s=inputName.getText();
        int n=JOptionPane.showConfirmDialog(this,"确认正确吗  ","确认对话框",
                                            JOptionPane.YES_NO_OPTION );
        if(n==JOptionPane.YES_OPTION)
          {  save.append("\n"+s);
          }
        else if(n==JOptionPane.NO_OPTION)
          { inputName.setText(null);
          }
    }
}
public class Example16_4
{  public static void main(String args[])
    {  new Dwindow("带对话框的窗口");
    }
}
```

图 16.4　确认对话框

16.5 颜色对话框

可以用 javax.swing 包中的 JColorChooser 类的静态方法:

```
public static Color showDialog(Component component,String title,Color initialColor)
```

创建一个颜色对话框,其中参数 component 指定对话框所依赖的组件,title 指定对话框的标题　initialColor 指定对话框返回的初始颜色,即对话框消失后,返回的默认值.颜色对话框可根据用户在颜色对话框中选择的颜色返回一个颜色对象.

在下面的例子 5 中,当用户单击按钮时,弹出一个颜色对话框,然后根据用户选择的颜色来改变按钮的颜色.

图 16.5　颜色对话框

例子 5　效果如图 16.5

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.JColorChooser;
class Dwindow extends Frame implements ActionListener
{  Button button;
   Dwindow(String s)
   {  super(s);
      button=new Button("打开颜色对话框");
      button.addActionListener(this);
      setLayout(new FlowLayout());
      add(button);
      setBounds(60,60,300,300);
      setVisible(true);
      validate();
      addWindowListener(new WindowAdapter()
                        {   public void windowClosing(WindowEvent e)
                            {
                              System.exit(0);
                            }
                        }
                       );
```

```
    }
    public void actionPerformed(ActionEvent e)
    {
      Color newColor=JColorChooser.showDialog(this,"调色板",button.getBackground());
      button.setBackground(newColor);
    }
}
public class Example16_5
{  public static void main(String args[])
    {
       new Dwindow("带颜色对话框的窗口");
    }
}
```

习题

1　编写一个应用程序,用户可以在一个文本框里输入数字字符,按回车后将数字放入一个文本区.当输入的数字大于 1000 时,弹出一个有模式的对话框,提示用户数字已经大于 1000,是否继续将该数字放入文本区.

2　参考 Windows 平台的 NotePad,编写一个简单的"记事本"程序.

第十七章　Java 与图形

前面过了许多 Java 小程序了,当你用浏览器打开含有小程序的 web 页时,会首先看到浏览器为小程序创建的一个实例,如图 17.1 所示.

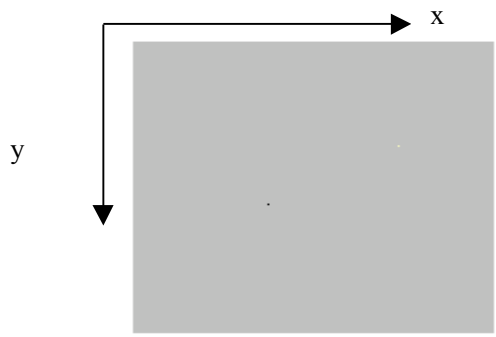


图 17.1　Java Applet 实例

它的大小由超文本文件:

```
<applet code=h.class width=200 height=300>
</applet>
```

中的 width　和 height 指定的.原点是上图的左上角,坐标的单位是像素.

Java 的 java.awt 中的 Graphics 类有很多处理图形的方法,供该类的对象使用,本章选择主要方法进行介绍.

17.1 绘制文本

可以使用 drawString 方法在屏幕上显示串对象.

drawString(String s, int x, int y)　从参数 x,y 指定的坐标位置处,从左向向右绘制参数 s 指定的字符串.

drawChars(char data[],int offset, int length, int x, int y)　绘制 data 数组中的部分字符,length 指定数组中要连续绘制的字符的个数,offset 是首字符在数组中的位置.

例子 1(效果如图 17.2)

```
import java.applet.*;import java.awt.*;
public class Example17_1 extends Applet
{  public void paint(Graphics g)
   {  int y,x=120;
      g.drawString("计算机科学技术",10,20);
      g.drawString("I Love This Game",20,40);
```

图 17.2　绘制文本

```
        char a[]="中国科学技术大学".toCharArray();
        for(int i=0;i<a.length;i++)
          {  y=2*x-200;
             g.drawChars(a,i,1,x,y);
             x=x+6;
          }
     }
}
```

17.2 绘制基本图形

1　绘制直线

drawLine(int x1,int y1,int x2,int y2)　绘制从起点(x1,y1)到终点(x2,y2)的直线段.

例子 2(效果如图 17.3)

```
import java.applet.*;import java.awt.*;
public class Example17_2 extends Applet
{  public void paint(Graphics g)
   {  int gap=0;
      for(int i=1;i<=10;i++)
        {  g.drawLine(10,10+5*i,180,10+5*i);
        }
      for(int i=1;i<=8;i++)
       {  g.drawLine(40+3*i,70,40+3*i,150);
       }
      g.drawLine(64,70,150,156);
   }
}
```

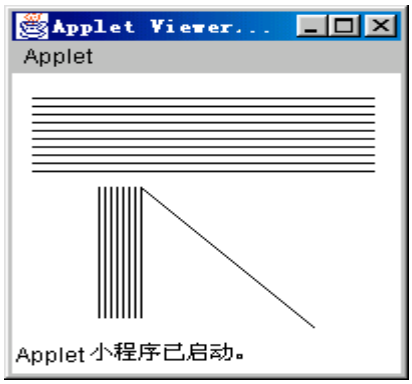


图 17.3　绘制直线

2　绘制矩形

drawRect (int x,int y,int w,int h)　绘制矩形,矩形的左上角的坐标由参数 x,y 指定,矩形的宽和高由参数 w,h 指定.

fillRect (int x,int y,int w,int h)　绘制填充矩形.

3　绘制圆角矩形

drawRoundRect(int x,int y,int w,int h,int arcW,int arcH)　绘制圆角矩形.参数 arcW,arcH 指定圆角的尺寸,见图 17.4 中的 4 个黑角部分.

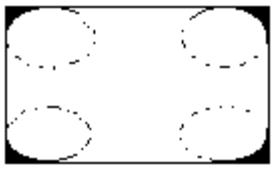


图 17.4　圆角的尺寸

4　drawRoundRect(int x,int y,int w,int h,int arcW,int arcH)　绘制填充圆角矩形.

5　drawOval(int x,int y,int w,int h)　绘制椭圆. x,y 给出椭圆距 x 轴和 y 轴的距离,参数 w,h 给出椭圆的宽和高.

6　drawOval(int x,int y,int w,int h)　绘制填充椭圆.

例子 3(效果如图 17.5)

```
import java.applet.*;import java.awt.*;
 public class Example17_3 extends Applet
 {  public void paint(Graphics g)
    {  g.drawRect(24,22,60,34);
       g.drawRoundRect(10,10,90,60,50,30);
       g.setColor(Color.cyan);
       g.fillOval(10,80,120,60);
       int k=0;
       for(int i=1;i<=8;i++)
         {  Color c=new Color(i*32-1,0,0);
            g.setColor(c); k=k+5;
            g.drawOval(160+k,77+k,120-2*k,80-2*k);
         }
    }
}
```

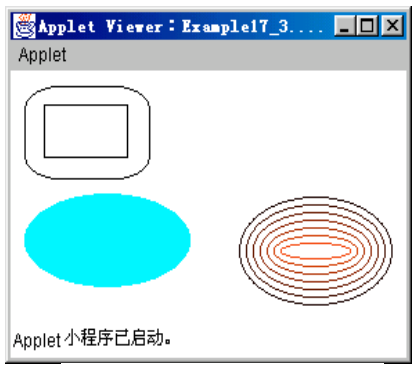


图 17.5　绘制矩形,椭圆

7　绘制圆弧

drawArc(int x,int y,int width,int height,int starAngle, int arcAngle)　该方法的 width　heigth starAngle 和 arcAngle 参数如何控制圆弧的形状. 弧的中心就是它的外接矩形的中心,该外接矩形的左上角的坐标是(x,y),宽度是 width,高度是 height. 参数 starAngle 和 arcAngle 的单位都是"度". 而起始角度的 0 度是 3 点钟的方位. 参数 starAngle,arcAngle 表示从 starAngle 的角度开始,逆时针方向画 arcAngle 度的弧 starAngle 的值可以是负值,例如-90 度是 6 点的方位.

fillArc(int x,int y,int width,int height,int starAngle,int arcAngle)　绘制填充圆弧.

实际上,所画的圆弧就是某个椭圆的一部分.

例子 4(效果如图 17.6)

```
import java.applet.*;import java.awt.*;
public class Example17_4 extends Applet
{  public void paint(Graphics g)
   {  g.drawArc(0,40,100,50,0,180);
      g.drawArc(100,40,100,50,180,180);
      g.setColor(Color.blue);
      g.fillArc(0,100,40,40,0,270);
      g.setColor(Color.green);
      g.drawArc(55,120,120,60,-90,270);
   }
}
```

图 17.5　绘制圆弧

8　画多边形

drawPolygon(int xPoints[], int yPoints[], int nPoints)　绘制多边形.参数数组 xPoint 和 yPoint 组成多边形的顶点坐标,nPoints 是顶点的数目.

fillPolygon(int xPoints[], int yPoints[], int nPoints)　绘制填充多边形.

注　Java 自动闭合多边形,程序总是把最后的顶点和第一个顶点连接起来.

例子 5(效果如图 17.6)

```
import java.applet.*;
import java.awt.*;
public class Example17_5 extends Applet
{  int px1[]={40,80,0,40};
   int py1[]={5,  45,45,5};
   int px2[]={140,180,180,140,100,100,};
   int py2[]={5,   25,   45,  65,  45,  25,};
   public void paint(Graphics g)
   {  g.drawPolygon(px1,py1,4);//从点(40,5)画到点(80,45),再从点(80,45)画到点(0,45).
      g.drawPolygon(px2,py2,6);
   }
}
```

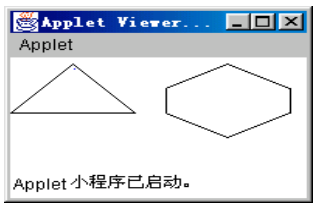


图 17.6　绘制多边形

17.3 建立字体

setFont(Font f)　设置字体.当绘制文本时,Graphics 对象用参数 f 指定的字体.

例子 6(效果如图 17.7)

```
import java.applet.*; import java.awt.*;
public class Example17_6  extends Applet
{  Font f1=new Font("Helvetica",Font.PLAIN,28);
   Font f2=new Font("Helvetica",Font.BOLD,15);
   Font f3=new Font("Courier",Font.PLAIN,12);
   Font f4=
   new Font("TimesRoman",Font.BOLD+Font.ITALIC,18);
   public void paint(Graphics g)
   {  g.setFont(f1);
      g.drawString("28pt plain Helvetica",5,20);
      g.setFont(f2);
      g.drawString("15pt bold Helvetica",5,43);
      g.setFont(f3);
      g.drawString("12pt plain courier",5,75);
      g.setFont(f4);
      g.drawString("18pt bold & italic Times Roman",5,92);
   }
}
```

图 17.7　用指定字体绘制文本

17.4 清除

clearRect(int x,int y,int width,int height)　用背景色填充指定矩形以达到清除该矩形的效果,也就是说当一个Graphics对象使用该方法时,相当于在使用一个"橡皮擦".参数 x,y 是被清除矩形的左上角的坐标　另外两个参数是被清除矩形的宽和高.
在下面的例子中,我们在一个红色圆中清除一个矩形,设置小程序的颜色是黄色　那么背景色就是黄色,因为我们是在小容器这个容器背景上绘画　.

例子 7(效果如图 17.8)

```
import java.applet.*;import java.awt.*;
public class Example17_7 extends Applet
{  public void init()
   {  setBackground(Color.yellow);
   }
   public void paint(Graphics g)
   {  g.setColor(Color.red);
      g.fillOval(10,10,100,100);
      g.clearRect(40,40,40,40);
   }
}
```

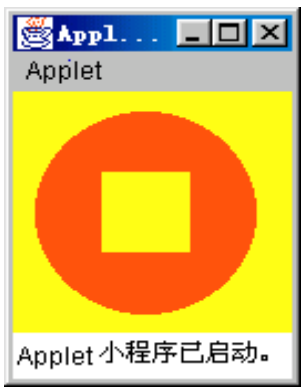


图 17.8　使用 clearRect 方法

我们已经知道,在小程序中调用 repaint 方法时,程序首先清除 paint()方法以前所画的内容,然后再调用 paint()方法.但有时我们不想让程序清除 paint()方法以前所画的所有内容.实际上当我们调用 repaint()方法时,程序自动地去调用 update(Graphics g)方法 从父类 Component 继承下来的 清除 paint()方法以前所画的内容,然后再调用 paint 方法.因此我们可以在我们的小程序中重写这个方法 即隐藏父类的方法 ,根据需要来清除哪些部分或保留哪些部分.

下面的例子 8 重写了 update 方法.

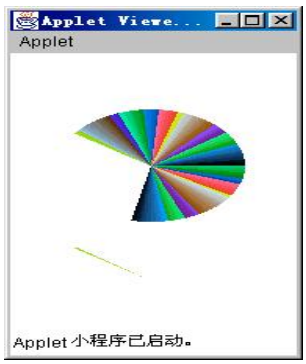


图 17.9 update 方法

例子 8(效果如图 17.9)

```
import java.applet.*;import java.awt.*;
import java.awt.event.*;
public class Example17_8 extends Applet
{  int i=0;
   public void paint(Graphics g)
   {  i=(i+2)%360;
      Color c=new Color((3*i)%255,(7*i)%255,(11*i)%255);
      g.setColor(c); g.fillArc(30,50,120,100,i,2);
      g.fillArc(30,152,120,100,i,2);
      try{  //程序暂停 300 毫秒,再执行 repaint()(见 19 章):
            Thread.sleep(300);
         }
      catch(InterruptedException e){}
      repaint();
   }
   public void update(Graphics g)
   {  g.clearRect(30,152,120,100);
      paint(g);
   }
}
```

17.5 Java 2D

java 1.2 拥有了更为强大的二维图形处理能力.在前面学习中,我们绘制图形的手段是使用 paint(Graphics g)方法,通过 Graphics 对象 g 调用各种方法绘制图形.Java1.2 给出了一个新类 Graphics2D,它是 Graphics 类的子类.一个 Graphics2D 对象把直线,圆等作为一个对象来绘制 也就是说,如果想用一个 graphics2D 类型的"画笔"来画一个圆的话,就必须先创建一个圆的对象.我们仍需使用 paint(Graphics g)方法来绘制,只需将 Graphics 对象强制转化为 Graphics2D 对象即可.

1 画直线

使用 java.awt.geom 包中的 Line2D 的子类 Line2D.Double 创建一个直线对象，Double(double x1,double y1,double x2, double y2) 创建一个 x1,y1 到 x2,y2 Line2D 对象.

例子 9(效果如图 17.10)

```
import java.awt.*; import java.applet.*;
import java.awt.geom.*;
public class Example17_9 extends Applet
{  public void paint(Graphics g)
   {  g.setColor(Color.red);
      Graphics2D g_2d=(Graphics2D)g;
      Line2D line=new Line2D.Double(2,2,300,300);
      g_2d.draw(line);
      for(int i=1,k=1;i<=10;i++)
         {  line.setLine(10,10+k,80,10+k);
            g_2d.draw(line); k=k+3;
         }
   }
}
```

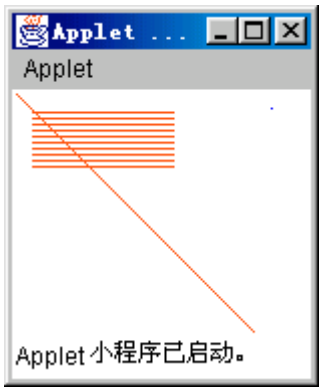


图 **17.10** 绘制 2D 直线

注 目前的浏览器(IE5.0)可能还不支持 java2D,你可以使用 appletviewer 来调试你的小程序.

2 绘制矩形

使用 java.awt.geom 包中的 Rectangle2D. Double 类来创建一个矩形对象,如

```
Rectangle2D rect=new Rectangle2D. Double (50,50,300,50.897);
```

上述语句创建了一个左上角坐标是(50,50),宽是 300,高是 50.987 的一个矩形对象.

例子 10(效果如图 17.11)

```
import java.awt.*; import java.applet.*;
import java.awt.geom.*;
public class Example17_10 extends Applet
{  public void paint(Graphics g)
   {  g.setColor(Color.blue) ;
      Graphics2D g_2d=(Graphics2D)g;
      Rectangle2D rect=
      new Rectangle2D. Double (20,30,30,30);
      g_2d.draw(rect);
      for(int i=1;i<=5;i++)
```

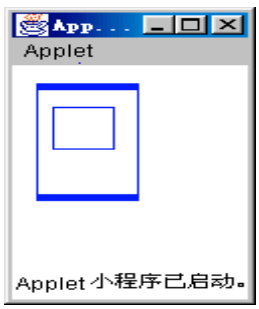


图 **17.11** 绘制 2D 矩形

```
        {  rect.setRect(12,12+i,50,80);
           g_2d.draw(rect);
        }
    }
}
```

3　　绘制圆角矩形

使用 java.awt.geom 包中的 RoundRectangle2D.Double 类来创建一个圆角矩形对象

```
RoundRectangle2D rect_round=new RoundRectangle2D. Double (50,50,300,50,8,5);
```

上述语句创建了一个左上角坐标是(50,50),宽是 300,高是 50,圆角的长轴和短轴分别为 8 和 5 的圆角矩形对象.

4　　绘制椭圆

使用 java.awt.geom 包中的 Ellipse2D.Double 类来创建一个椭圆对象

```
Ellipse2D ellipse=new Ellipse2D. Double (50,30,300,50);
```

上述语句创建了一个外接矩形的左上角坐标是(50,30),宽是 300,高是 50 的椭圆对象.

例子 11(效果如图 17.12)

```
import java.awt.*;import java.applet.*;
import java.awt.geom.*;
public class Example17_11 extends Applet
{  public void paint(Graphics g)
   {  g.setColor(Color.blue) ;
      Graphics2D g_2d=(Graphics2D)g;
      Ellipse2D ellipse=
      new Ellipse2D. Double (20,30,100,50);
      g_2d.draw(ellipse);
      for(int i=1,k=0;i<=6;i++)
        {  ellipse.setFrame(20+k,30,100-2*k,50);
           g_2d.draw(ellipse); k=k+5;
        }
   }
}
```

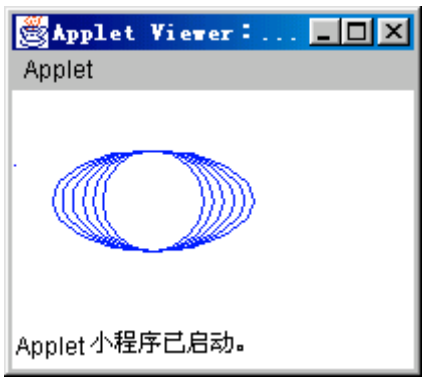


图 **17.12** 绘制 2D 椭圆

5　　绘制圆弧

使用 java.awt.geom 包中的 Arc2D.Double 类创建一个圆弧对象

```
Arc2D ellipse=new Arc2D. Double (50,30,300,50,0,100,Arc.PIE);
```

上述语句创建了一个外接矩形的左上角坐标是(50,30),宽是 300,高是 50 的起始角是 0 度终止角是 100 度的饼弧对象.其中,最后一个参数取值 Arc.OPEN,Arc.CHORD,Arc.PIE 决定弧是开弧,弓弧和饼弧.

例子 12(效果如图 17.13)

```
import java.awt.*; import java.applet.*;
import java.awt.geom.*;
public class Example17_12 extends Applet
{  public void paint(Graphics g)
   {  g.setColor(Color.red) ;
      Graphics2D g_2d=(Graphics2D)g;
      Arc2D arc=
      new Arc2D.Double (2,30,80,55,180,-90,Arc2D.OPEN);
      g_2d.draw(arc);
      arc.setArc(90,30,90,70,0,180,Arc2D.CHORD);
      g_2d.draw(arc);
      arc.setArc(190,30,50,90,0,270,Arc2D.PIE);
      g_2d.draw(arc);
   }
}
```

图 **17.13** 绘制 2D 弧

6. 绘制二次曲线

二次曲线可用二阶多项式

$y(x)=ax^2+bx+c$

来表示.一条二次曲线需要三个点来确定.使用 java.awt.geom 包中的 QuadCurve2D.Double 类来创建一个二次曲线

```
QuadCurve2D  curve=new QuadCurve2D.Double (50,30,10,10,50,100);
```

上述代码片段使用端点 50,30 和 50,100 及控制点 10,10 创建了一条二次曲线.

例子 13(效果如图 17.14)

```
import java.awt.*;import java.applet.*;
import java.awt.geom.*;
public class Example17_13 extends Applet
{  public void paint(Graphics g)
   {  g.setColor(Color.red) ;
      Graphics2D g_2d=(Graphics2D)g;
      QuadCurve2D quadCurve=
      new  QuadCurve2D.Double(2,10,51,90,100,10);
      g_2d.draw(quadCurve);
```

```
        quadCurve.setCurve(2,100,51,10,100,100);
        g_2d.draw(quadCurve);
      }
   }
```

7.　　绘制三次曲线

三次曲线可用三阶多项式

$y(x)=ax^3+bx^2+cx+d$

来表示.一条三次曲线需要四个点来确定该曲线.使用 **java.awt.geom** 包中的 **QuadCurve2D.Double** 类来创建一个二次曲线

```
CubicCurve2D  curve=new QuadCurve2D.Double (50,30,10,10,100,100,50,100);
```

上述代码片段使用端点 **50,30** 和 **50,100** 及控制点 **10,10** 和**(100,100)**创建了一条三次曲线.

例子 14

```
import java.awt.*;import java.applet.*;
import java.awt.geom.*;
public class Example17_14  extends Applet
{  public void paint(Graphics g)
   {  g.setColor(Color.red) ;
      Graphics2D g_2d=(Graphics2D)g;
      CubicCurve2D curve_1=new  CubicCurve2D.Double(2,30,80,55,10,10,20,90);
      CubicCurve2D curve_2=new  CubicCurve2D.Double(2,30,5,67,20,30,20,90);
      CubicCurve2D curve_3=new  CubicCurve2D.Double(30,35,54,67,20,90,100,190);
      g_2d.draw(curve_1);g_2d.draw(curve_2);g_2d.draw(curve_3);
   }
}
```

8　　控制图形线条的粗细

Graphics 类创建的”画笔”的粗细是默认的,我们不能改变它.在 **Java2D** 中,我们可以改变画笔的粗细.

首先使用 **BasicStroke** 类创建一个供画笔选择线条粗细的对象.**BasicStroke** 类的一个常用的构造方法

```
BasicStroke (float width  int cap,int join),
```

其中 **width** 参数决定画笔线条的粗细,缺省值是 **1.cap** 参数决定线条两端的形状,对于与周围不再有连接的直线或曲线该参数是有用的.**cap** 参数的取值是

```
BasicStroke.CAP_BUTT, BisicStroke.CAP_ROUND, BisicStroke.CAP_SQUARE.
```

参数 join 决定线条中的角应如何处理,join 取值是

BasicStroke.JOIN_BEVEL BisicStroke.JOIN_MITER, BisicStroke.JOIN_ROUND.

然后 Graphics2D 对象调用方法 setStroke(BasicStroke a)设置线条形状.

例子 15(效果如图 17.15)

图 **17.15**　设置线条

```
import java.awt.*;import java.applet.*;
import java.awt.geom.*;
public class Example17_15 extends Applet
{  public void paint(Graphics g)
   {  Graphics2D g_2d=(Graphics2D)g;
      BasicStroke bs_1
      =new BasicStroke(16,BasicStroke.CAP_BUTT,BasicStroke.JOIN_BEVEL);
      BasicStroke bs_2
      =new BasicStroke(16f,BasicStroke.CAP_ROUND,BasicStroke.JOIN_MITER);
      BasicStroke bs_3
      =new BasicStroke(16f,BasicStroke.CAP_SQUARE,BasicStroke.JOIN_ROUND);
      Line2D line_1=new  Line2D.Double(20,20,100,20);
      Line2D line_2=new  Line2D.Double(20,50,100,50);
      Line2D line_3=new  Line2D.Double(20,80,100,80);
      g_2d.setStroke(bs_1); //设置线条.
      g_2d.draw(line_1);
      g_2d.setStroke(bs_2); g_2d.draw(line_2);
      g_2d.setStroke(bs_3); g_2d.draw(line_3);
   }
}
```

9　　填充图形

Graphics2D 对象使用 fill 方法填充图形.

下面的 16 使用 fill 方法绘制"太极图"

例子 16(效果如图 17.16)

图 **17.16**　太极图

```
import java.awt.*;import java.applet.*;
import java.awt.geom.*;
public class Example17_16 extends Applet
{  public void paint(Graphics g)
   {  Graphics2D g_2d=(Graphics2D)g;
      g_2d.setColor(Color.cyan);
      Rectangle2D rect=new Rectangle2D.Double(0,0,200,200);
```

```
        g_2d.fill(rect);
        Arc2D arc1=new Arc2D.Double(0,0,200,200,0,180,Arc2D.CHORD),
              arc2=new Arc2D.Double(0,0,200,200,0,-180,Arc2D.CHORD);
        RoundRectangle2D roundR1=new RoundRectangle2D.Double
                                 (0,50,100,100,100,100),
                        roundR2=new RoundRectangle2D.Double
                                 (100,50,100,100,100,100),
                        roundR3=new RoundRectangle2D.Double
                                 (37.5,87.8,25,25,25,25),
                        roundR4=new RoundRectangle2D.Double
                                 (137.5,87.8,25,25,25,25);
        g_2d.setColor(Color.white);  g_2d.fill(arc1);
        g_2d.setColor(Color.black);  g_2d.fill(arc2);
        g_2d.fill(roundR1);
        g_2d.setColor(Color.white);
        g_2d.fill(roundR2);  g_2d.fill(roundR3);
        g_2d.setColor(Color.black);
        g_2d.fill(roundR4);
    }
}
```

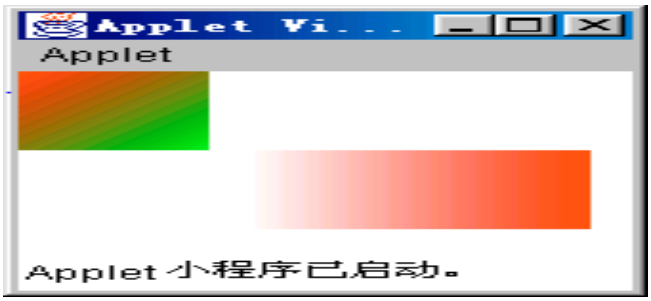


图 **17.17**　颜色渐变

Java2D 还容许我们使用渐变的颜色填充一个图形.使用 GradientPaint 类可以定义一个渐变的颜色对象.GradientPaint 类的构造方法

```
GradientPaint  float x1,float y1,Color color1,float  x2,float y2,Color2 ,boolean cyclic
```

其中参数 color1,color2 决定这个渐变色是从颜色 color1 渐变到颜色 color2 参数 x1,y1,x2,y2 决定了渐变的强弱,即要求颜色 color1 从点 x1,y1 出发到达点 x2,y2 变成 color2.如果希望当渐变到达终点时循环起点的颜色,就将 cyclic 设置为 true.

Graphics2D 对象使用 setPaint()方法来使用渐变颜色.

例子 17(效果如图 17.17)

```
import java.awt.*;import java.applet.*;
import java.awt.geom.*;
public class Example17_17  extends Applet
{  public void paint(Graphics g)
   {  Graphics2D g_2d=(Graphics2D)g;
      GradientPaint gradient_1
      =new GradientPaint(0,0,Color.red,50,50,Color.green,false);
      g_2d.setPaint(gradient_1);
      Rectangle2D rect_1=new Rectangle2D.Double (0,0,50,50);
      g_2d.fill(rect_1);
      GradientPaint gradient_2
      =new GradientPaint(60,50,Color.white,150,50,Color.red,false);
      g_2d.setPaint(gradient_2);
      Rectangle2D rect_2=new Rectangle2D.Double (60,50,90,50);
      g_2d.fill(rect_2);
   }
}
```

10　　旋转图形

我们有时需要平移,缩放或旋转一个图形.我们可以使用 AffineTransform 类来实现对图形的这些操作.

第一步　首先使用 AffineTransform 类创建一个对象

```
AffineTransform   trans=new AffineTransform();
```

对象 **trans** 具有最常用的三个方法来实现对图形变换操作

- translate(double a,double b)　将图形在 x 轴方向移动 a 个单位像素,y 轴方向移动 b 个像素单位.x 是正值时向右移动,负值是向左移动　y 是正值时是向下移动,负值向上移动.
- scale(double a,double b)　将图形在 x 轴方向缩放 a 倍,y 轴方向缩放 b 倍.
- rotate(double number,double x,double y)　将图形沿顺时针或逆时针以　x,y 为轴点旋转 number 个弧度.

第二步　进行需要的变换,比如要把一个矩形绕　100,100　点顺时针旋转 60 度,那么就要在第二步作好准备

```
trans.rotate(60.0*3.1415927/180,100,100);
```

第三步　把 **Graphics** 对象,比如 **g_2d** 设置为具有 **trans** 这种功能的"画笔",

```
g_2d.setTransform(trans);
```

假如 rect 是一个矩形对象,那么 g_2d.draw(rect)画的就是旋转后的矩形的样子.
注意不要把第二步和第三步颠倒.

例子 18(效果如图 17.18)

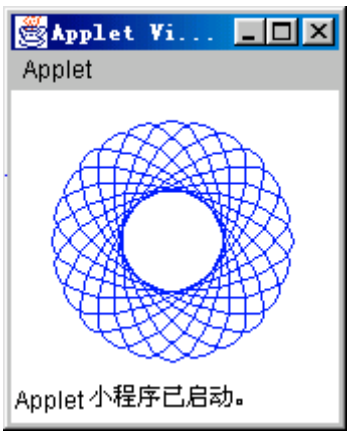


图 **17.18** 旋转椭圆

```
import java.awt.*;import java.applet.*;
import java.awt.geom.*;
public class Example17_18 extends Applet
{  public void paint(Graphics g)
   {  Graphics2D g_2d=(Graphics2D)g;
      Ellipse2D ellipse=
      new Ellipse2D. Double (20,50,120,50);
      g_2d.setColor(Color.blue);

      AffineTransform trans=new  AffineTransform();
      for(int i=1;i<=24;i++)
        {  trans.rotate(15.0*Math.PI/180,80,75);
           g_2d.setTransform(trans);
           //现在画的就是旋转后的椭圆样子
           g_2d.draw(ellipse);
        }
   }
}
```

以下的例子 19 结合 2 次曲线和 3 次曲线绘制一个花朵.

例子 19(效果如图 17.19)

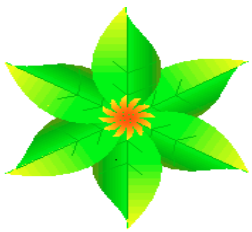


图 17.19 绘制花朵

```
import java.awt.*;import java.awt.event.*;
import java.awt.geom.*;
import java.applet.*;
public class Flower extends Applet
{  public void paint(Graphics g)
   {  Graphics2D  g_2d=(Graphics2D)g;
      //花叶两边的曲线:
      QuadCurve2D
      curve_1=new  QuadCurve2D.Double(200,200,150,160,200,100);
      CubicCurve2D curve_2=
      new  CubicCurve2D.Double(200,200,260,145,190,120,200,100);
      //花叶中的纹线:
      Line2D line=new Line2D.Double(200,200,200,110);
      QuadCurve2D leaf_line1=
```

```
new  QuadCurve2D.Double(200,180,195,175,190,170);
QuadCurve2D leaf_line2=
new  QuadCurve2D.Double(200,180,210,175,220,170);
QuadCurve2D leaf_line3=
new  QuadCurve2D.Double(200,160,195,155,190,150);
QuadCurve2D leaf_line4=
new  QuadCurve2D.Double(200,160,214,155,220,150);
//利用旋转来绘制花朵:
AffineTransform trans=new AffineTransform();
for(int i=0;i<6;i++)
 {  trans.rotate(60*Math.PI/180,200,200);
    g_2d.setTransform(trans);
    GradientPaint gradient_1=
     new GradientPaint(200,200,Color.green,200,100,Color.yellow);
    g_2d.setPaint(gradient_1);
    g_2d.fill(curve_1);
    GradientPaint gradient_2=new
     GradientPaint(200,145,Color.green,260,145,Color.red,true);
    g_2d.setPaint(gradient_2);
    g_2d.fill(curve_2);
    Color c3=new Color(0,200,0); g_2d.setColor(c3);
    g_2d.draw(line);
    g_2d.draw(leaf_line1); g_2d.draw(leaf_line2);
    g_2d.draw(leaf_line3); g_2d.draw(leaf_line4);
  }
//花瓣中间的花蕾曲线:
  QuadCurve2D center_curve_1=
  new   QuadCurve2D.Double(200,200,190,185,200,180);
  AffineTransform trans_1=new AffineTransform();
  for(int i=0;i<12;i++)
    {  trans_1.rotate(30*Math.PI/180,200,200);
       g_2d.setTransform(trans_1);
       GradientPaint gradient_3=
       new GradientPaint(200,200,Color.red,200,180,Color.yellow);
         g_2d.setPaint(gradient_3);
         g_2d.fill(center_curve_1);
     }
   //再绘制一个 0.4 倍的花朵:
   AffineTransform trans_2=new AffineTransform();
```

```
            trans_2.scale(0.4,0.4);
            for(int i=0;i<6;i++)
              {  trans_2.rotate(60*Math.PI/180,200,200);
                 g_2d.setTransform(trans_2);g_2d.setColor(Color.pink);
                 g_2d.fill(curve_1);g_2d.setColor(Color.green);
                 g_2d.fill(curve_2);
              }
        }
    }
```

17.6 图形的布尔运算

通过基本图形的布尔运算可以得到更为复杂的图形.假设 T1,T2 是两个图形.

T1 和 T2 的布尔"与"(AND)运算的结果是两个图形的重叠部分.

T1 和 T2 的布尔"或"(OR)运算的结果是两个图形的合并.

T1 和 T2 的布尔"差"(NOT)运算的结果是 T1 去掉 T1 和 T2 的重叠部分.

T1 和 T2 的布尔"异或"(XOR)运算的结果是两个图形的非重叠部分.

两个图形进行布尔运算运算之前,必须分别用这两个图形创建两个 Area 区域对象,例如

```
Area a1=new Area(T1);
Area a2=new Area(T2);
```

a1 就是图形 T1 所围成的区域　a2 就是 T2 所围成的区域.那么,a1 调用 add 方法

```
a1.add(a2);
```

之后,a1 就变成 a1 和 a2 经过布尔"或"运算后的图形区域.可以用 Graphics2D 对象 g 来填充一个 Area 对象　区域

```
g.draw(a1);
```

Area 类的常用方法

```
public void add(Area r)  与参数 r 布尔"或".
public void intersect(Area r)  与参数 r 布尔"与".
public void exclusiveOr(Area rhs)  与参数 r 布尔"异或"
public void subtract(Area rhs) 与参数 r 布尔"差"
```

下面的例子 20 显示的图形的布尔运算.

例子 20(效果如图 17.20)

```
import java.awt.*;import java.applet.*;
import java.awt.geom.*;
public class Example17_20 extends Applet
```

```
{  public void paint(Graphics g)
   {  Graphics2D g_2d=(Graphics2D)g;
      Ellipse2D ellipse1=
      new Ellipse2D. Double (20,50,120,120);
      Ellipse2D ellipse2=
      new Ellipse2D. Double (80,50,120,120);
      Area a1=new Area(ellipse1); Area a2=new Area(ellipse2);
      a1.intersect(a2);              //"与"
      g_2d.fill(a1);
      ellipse1.setFrame(150,50,120,120); ellipse2.setFrame(260,50,50,100);
      a1=new Area(ellipse1); a2=new Area(ellipse2);
      a1.add(a2);              //"或"
      g_2d.fill(a1);
      ellipse1.setFrame(20,170,120,120);ellipse2.setFrame(80,170,160,160);
      a1=new Area(ellipse1);  a2=new Area(ellipse2);
      a1.subtract(a2);              //"差"
      g_2d.fill(a1);
      ellipse1.setFrame(150,170,120,120); ellipse2.setFrame(260,170,50,100);
      a1=new Area(ellipse1);  a2=new Area(ellipse2);
      a1.exclusiveOr(a2);          //"异或"
      g_2d.fill(a1);
   }
}
```

下面的例子 21 旋转两个图形经过布尔运算得到区域.

例子 21(效果如图 17.21)

```
import java.awt.*;import java.applet.*;
import java.awt.geom.*;
public class Example17_21 extends Applet
{  public void paint(Graphics g)
   {  Graphics2D g_2d=(Graphics2D)g;
      Ellipse2D ellipse1=
      new Ellipse2D. Double (20,80,60,60),
      ellipse2=
      new Ellipse2D. Double (40,80,80,80);
      g_2d.setColor(Color.blue);
      Area a1=new Area(ellipse1),
           a2=new Area(ellipse2);
      a1.subtract(a2);              //"差"
```

```
        AffineTransform trans=new  AffineTransform();
        for(int i=1;i<=10;i++)
           {  trans.rotate(36.0*Math.PI/180,80,75);
              g_2d.setTransform(trans);
              g_2d.fill(a1);
           }
     }
}
```

注 Java2D 本身就是 Java 中很丰富的一部分,这里我们只作了初步介绍.

17.7 XOR 绘图模式

Graphics 类有一个方法 setXORMode(Color color),Graphics 对象 g 可以使用该方法将绘图模式设置为 XOR(异或)模式,例如

```
g.setXORMode(Color.red);
```

g 使用 XOR 模式绘制图形时,如果 g 本身的颜色是 c(c 不可取 red 颜色),那么 g 绘制图形时,g 实际上使用颜色 c 与其相遇的所有颜色(包括背景色)做 XOR 运算后的颜色来绘制图形.

XOR 运算法则

假如

```
g.setXORMode(Color.red);
```

那么

背景色+背景色=red

c+c=背景色(c 是非背景色)

c+d=c 和 d 的混合色(c,d 不相同).

图 17.22 XOR 绘图效果

因此,g 使用颜色 c 绘制了一个图形后,如果在同一位置用颜色 c 重复绘制该图形,将使得该图形的某些部分颜色变成当前背景色,相当于清除该图形的某些部分.

例子 22(效果如图 17.22)

```
import java.awt.*;import java.applet.*;
public class PaintTest extends Applet
{  public void init()
   {  setBackground(Color.yellow);
   }
   public void paint(Graphics g)
   {  g.setXORMode(Color.red);//设置 XOR 绘图模式.
      g.setColor(Color.green);
```

```
        g.fillRect(20,20,80,40);//矩形的实际颜色是 green+yellow 的混合色  灰色.
        g.setColor(Color.yellow);
        g.fillRect(60,20,80,40);//该矩形的前一半是 yellow+yellow+灰色=red+灰色,
                                //后一半是红色.
        g.setColor(Color.green);
        g.fillRect(20,70,80,40);//矩形的实际颜色是 green+yellow 的混合色  灰色.
        g.fillRect(60,70,80,40);// 该矩形的前一半是 green+yellow+灰色=背景色,
                                // 后一半是 green+yellow  灰色.
        g.setColor(Color.green);
        g.drawLine(100,100,200,200);//该直线是 green+yellow  灰色.
        //下面,在同一位置再绘制该直线,因此,该直线前半段是 green+yellow+灰色=
         //灰色+灰色=背景色,该直线后半段是 green+yellow=灰色
        g.drawLine(100,100,220,220);
        //仔细分析下列直线颜色的变化:
        g.setColor(Color.yellow);
        g.drawLine(20,30,160,30); g.drawLine(20,75,160,75);
    }
}
```

注 g 调用 setPaintMode()方法可以恢复到默认的 paint 绘图模式.

17.8 打印图形

我们可以在应用程序中打印图形.为了打印图形,我们必须首先获得一个 **PrintJob** 对象.假如我们获得了一个 **PrintJob** 对象 **p**,那么 **p** 可以使用 **getGraphics()**方法获得一个 **Graphics** 对象 **g**.这时使用 **g** 所画得的图形将被发送到打印机打印出来.当我们运行应用程序时,系统会打开我们熟悉的打印对话框.但 **PrintJob** 是 **java.awt** 包中的一个 **abstract** 类,我们不能用它直接创建对象.**Java.awt** 包中有个抽象类 **Toolkit**,可以帮助我们获得一个 **PringJob** 对象.**Toolkit** 类有一个获得 **PrintJob** 对象的方法 **getPrintJob(Frame f,Striong s,null)**.任何一个组件都可以使用 **getToolkit()**方法获得一个 **Toolkit** 对象.下面的例子打印一个矩形.我们也可以把我们的图形先画到一个画布上,然后再打印.

例子 23

```
import java.awt.*;import java.awt.event.*;
import java.awt.geom.*;
public class Example17_23
{  public static void main(String args[])
   {  Frame f=new Frame();
      f.setSize(70,70); f.setVisible(true);
      Mycanvas canvas=new Mycanvas();
```

```
        f.add(canvas,"Center");f.pack();
        f.addWindowListener(new WindowAdapter()
               {public void windowClosing(WindowEvent e)
                    {System.exit(0);
                    }
               });
        PrintJob p=f.getToolkit().getPrintJob(f,"ok",null);
        Graphics g=p.getGraphics();
        g.drawRect(30,30,40,40);
        g.dispose();
        p.end();
    }
}
class Mycanvas extends Canvas
 {  Mycanvas()
      {  setSize(200,200);
      }
    public void paint(Graphics g)
      {  g.drawRect(30,30,40,40);
      }
  }
```

习题十七

1　编写一个应用程序,绘制五角形,并打印出来.

2　用 java2D 绘制一条抛物线的一部分.

3. 用 java2D 绘制双曲线的一部分.

4. 利用 java2D 的平移,缩放,旋转功能绘制一个你喜欢的图形.

5　利用图形的布尔运算绘制各种样式的"月牙".

第十八章　Java 中的鼠标事件和键盘事件

任何组件上都可以发生鼠标事件,如　鼠标进入组件,退出组件,在组件上方点击鼠标,按下鼠,运动鼠标,拖动鼠标等都发生了鼠标事件,也就是说,组件可以成为发生鼠标事件的事件源.

18.1 使用 MouseListener 接口处理鼠标事件

使用 MouseListener 接口可以处理 5 种操作发生的鼠标事件

1　在事件源上按下鼠标键.
2　在事件源上释放鼠标键.
3　在事件源上击鼠标键.
4　鼠标进入事件源.
5　鼠标退出事件源.

鼠标事件的类型是 **MouseEvent**,即当发生鼠标事件时,**MouseEvent** 类自动创建一个事件对象.

MouseEvent 类中有下列几个重要的方法

1　getX()　获取鼠标在事件源的坐标系中的 x-坐标.
2　getY()　获取鼠标在事件源的坐标系中的 y-坐标.
3　getModifiers() 获取鼠标的左或右键.鼠标的左键和右键分别使用 InputEvent 类中的常量 BUTTON1_MASK 和 BUTTON3_MASK 来表示.
4　getClickCount()　获取鼠标被点击的次数.
5　getSource()　获取发生鼠标事件的事件源.

事件源获得监视器的方法是 addMouseListener(监视器).

MouseListener 接口中有如下方法

1　mousePressed(MouseEvent)　负责处理鼠标按下事件.即,当你在事件源按下鼠标时,监视器发现这个事件后将自动调用接口中的这个方法对事件作出处理.
2　mouseReleased(MouseEvent)　负责处理鼠标释放事件.即,当你在事件源释放鼠标时,监视器发现这个事件后将自动调用接口中的这个方法对事件作出处理.
3　mouseEntered(MouseEvent)　负责处理鼠进入容器事件.即,当鼠标进入时,监视器发现这个事件后将自动调用接口中的这个方法对事件作出处理.
4　mouseExited(MouseEvent)　负责处理鼠标离开.即,当鼠标离开容器时,监视器发现这个事件后将自动调用接口中的这个方法对事件作出处理.
5　mouseClicked(MouseEvent)　负责处理点击鼠标事件.即,当鼠标被击时,监视器发现这个事件后将自动调用接口中的这个方法对事件作出处理.

现在我们给出一个小程序的例子,在这个小程序中有一个文本框,它负责记录鼠标事件.

当鼠标进入小程序时,文本区显示”鼠标进入” 当鼠标离开时,文本区显示”鼠标离开” 当鼠标被按下时,文本区显示”鼠标被按下”并显示鼠标的坐标.

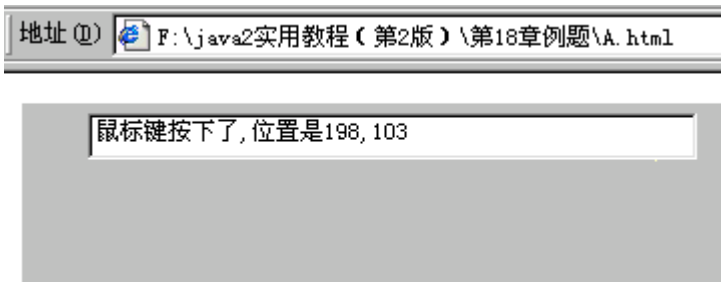


图 18.1 处理鼠标事件

例子 1 效果如图 18.1

```
import java.applet.*;import java.awt.*;
import java.awt.event.*;
public class Example18_1 extends Applet implements MouseListener
{  TextField text;
   public void init()
   {  text=new TextField(40); add(text);
      addMouseListener(this) ;//向小程序增加鼠标事件监视器.
   }
   public void mousePressed(MouseEvent e)
   {  text.setText("鼠标键按下了,位置是"+e.getX()+","+e.getY() );
   }
   public void mouseReleased(MouseEvent e)
   {  text.setText(" 鼠标松开了,位置是"+e.getX()+","+e.getY() );
   }
   public void mouseEntered(MouseEvent e)
   {  text.setText(" 鼠标进来了,位置是"+e.getX()+","+e.getY() );
   }
   public void mouseExited(MouseEvent e)
   {  text.setText(" 鼠标走开了");
   }
   public void mouseClicked(MouseEvent e)
   {  if(e.getClickCount()==2)
       {  text.setText("鼠标键双击,位置:"+e.getX()+","+e.getY());
       }
      else {}
   }
```

}

在下面例子 2 中,当在画布上按下鼠标左键时,在鼠标位置处绘制一个圆　当按下鼠标右键时,在鼠标位置处绘制一个矩形　当鼠标退出画布时,清除绘制的全部图形.

例子 2　效果如图 18.2

```
import java.awt.*;import java.awt.event.*;
class MyCanvas extends Canvas implements MouseListener
{  int left=-1,right=-1; //记录左,右键用的变量.
   int x=-1,y=-1;        //记录鼠标位置用的变量.
   MyCanvas()
   {  setSize(100,100);
      setBackground(Color.cyan) ;
      addMouseListener(this);
    }
   public void paint(Graphics g)
   {  if(left==1)
       {  g.drawOval(x-10,y-10,20,20);
       }
     else if(right==1)
       {  g.drawRect(x-8,y-8,16,16);
       }
   }
   public void mousePressed(MouseEvent e)
   {  x=e.getX(); y=e.getY();
      if(e.getModifiers()==InputEvent.BUTTON1_MASK)
        {  left=1;right=-1;
           repaint();
        }
      else if(e.getModifiers()==InputEvent.BUTTON3_MASK)
       {  right=1; left=-1;
          repaint();
        }
   }
   public void mouseReleased(MouseEvent e){}
   public void mouseEntered(MouseEvent e){}
   public void mouseExited(MouseEvent e)
   {  left=-1;right=-1;
      repaint();
   }
```

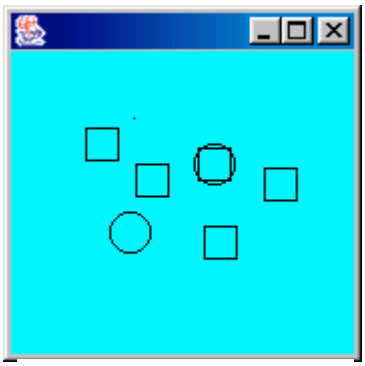


图 18.2　处理鼠标左,右键

图 18.3　获取鼠标在事件源中的坐标

```
    public void mouseClicked(MouseEvent e){}
    public void update(Graphics g)
    {  if(left==1||right==1)
         { paint(g);
         }
       else
         { super.update(g);
         }
    }
}
public class Example18_2
{  public static void main(String args[])
    {  Frame f=new Frame();
       f.setBounds(100,100,200,200);f.setVisible(true);
       f.addWindowListener(new WindowAdapter()  //适配器
                   {public void windowClosing(WindowEvent e)
                     {System.exit(0);
                     }
                   });
       f.add(new MyCanvas(),BorderLayout.CENTER);//添加画布.
       f.validate();
    }
 }
```

下面的例子 3 分别监视按钮,文本框和容器上的鼠标事件,当发生鼠标事件时,获取鼠标的坐标值,注意,事件源的坐标系的左上角是原点.

例子 3　效果如图 18.3

```
import java.awt.*;import java.awt.event.*;
import java.applet.*;
public class Example18_3 extends Applet implements MouseListener
{  TextField text; Button button;
   TextArea textArea;
   public void init()
   {  text=new TextField(10); text.addMouseListener(this);
      button=new Button("按钮"); button.addMouseListener(this);
      addMouseListener(this);
      textArea=new TextArea(8,28);
      add(button);add(text);add(textArea);
   }
```

```
    public void mousePressed(MouseEvent e)
    {  if(e.getSource()==button)
       {textArea.append("\n在按钮上鼠标按下,位置:"+"("+e.getX()+","+e.getY()+")");
       }
      else if(e.getSource()==text)
       {textArea.append("\n在文本框上鼠标按下,位置:"+"("+e.getX()+","+e.getY()+")");
       }
     else if(e.getSource()==this)
      {textArea.append("\n在容器上鼠标按下,位置:"+"("+e.getX()+","+e.getY()+")");
      }
    }
    public void mouseReleased(MouseEvent e) {}
    public void mouseEntered(MouseEvent e)  {}
    public void mouseExited(MouseEvent e) {}
    public void mouseClicked(MouseEvent e)
    {  if(e.getClickCount()>=2)
       textArea.setText("鼠标连击,位置:"+"("+e.getX()+","+e.getY()+")");
    }
}
```

18.2 使用 MouseMotionListener 接口处理鼠标事件

使用 MouseMotionListener 接口可以处理 2 种操作发生的鼠标事件

1 在事件源上拖动鼠标,

2 在事件源上运动鼠标.

鼠标事件的类型是 **MouseEvent**,即当发生鼠标事件时,**MouseEvent** 类自动创建一个事件对象.

事件源获得监视器的方法是 addMouseMotionListener(监视器).

MouseMotionListener 接口中有如下方法

1 mouseDragged(MouseEvent) 负责处理鼠标按下事件.即,当你在事件源按下鼠标时,监视器发现这个事件后将自动调用接口中的这个方法对事件作出处理.

2 mouseMoved(MouseEvent) 负责处理鼠标释放事件.即,当你在事件源释
监视器发现这个事件后将自动调用接口中的这个方法对事件作出处理.

你可能在其它的 Java 书籍里看过用鼠标作画的小程序,这些程序的代码都很长,也比较复杂.如果你感兴趣就可以在你安装的 jdk1.2 中找到这样一个例子 在 jdk1.2\demo\applets\DrawTest 下 .但愿你能有耐心研究这个例子.在这里我们也提供一个非常简单的用鼠标作画的小程序.我们的想法很简单,只要我们能用鼠标画点,那么就可以

用鼠标自由作画了.我们已经会用 drawLine(int x1,int y1,int x2, int y2)画从点 x1,y1 到点 x2,y2 的直线,那么当直线的起点和终点相同时,就画出了一个点.

例子 4 (效果如图 18.4 所示)

```
import java.applet.*;import java.awt.*;import java.awt.event.*;
public class Example18_4 extends Applet implements MouseMotionListener
{  int x=-1,y=-1;
   public void init()
   {  setBackground(Color.green);
      addMouseMotionListener(this);
   }
   public void paint(Graphics g)
   {  if(x!=-1&&y!=-1)
       {  g.setColor(Color.red);
          g.drawLine(x,y,x,y);
       }
   }
   public void mouseDragged(MouseEvent e)
   {  x=(int)e.getX();y=(int)e.getY();
      repaint();
    }
   public void mouseMoved(MouseEvent e){}
   public void update(Graphics g)
   {  paint(g);
   }
}
```

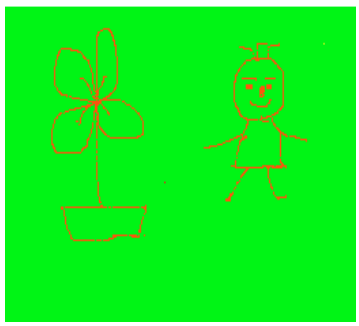


图 18.4 鼠标作画

现在让我们来分析一下例子 9 的代码.当我们拖动鼠标时 按着鼠标左键 ,我们就获得了鼠标拖动时的坐标,同时调用 repaint()方法画下这个坐标点.我们已经知道,在小程序中调用 repaint 方法时,程序首先清除 paint()方法以前所画的内容,然后再调用 paint()方法.但有时我们不想让程序清除 paint()方法以前所画的所有内容.实际上当我们调用 repaint()方法时,程序自动地去调用 update(Graphics g)方法 从父类 Applet 继承下来的 清除 paint()方法以前所画的内容,然后在调用 paint 方法.因此我们可以在我们的小程序中重写这个 update(Graphics g)方法 即隐藏父类的方法 ,根据需要来清除哪些部分或保留哪些部分.在我们这个例子中,我们在 update(Graphics g)方法体内没有清除以前所画的内容 如果准备清除可使用 clearRect 方法,见 17.4 而是继续调用 paint 方法.

受上面例子的启发,你可能已经想到,我们可以通过画实心圆来改变线条的粗细.下面是一个略微复杂的例子.在这个例子中我们可以控制线条的颜色,而且我们还可以使用橡皮擦掉所画的图形.您可以稍加改进就可以控制线条的粗细,另外你还可以使用上一节的内容控制鼠标的形状.

例子 5 (效果如图 18.5 所示)

```
import java.applet.*;import java.awt.*;
import java.awt.event.*;
public class Example18_5 extends Applet
implements ActionListener,MouseMotionListener
{  int x=-1,y=-1,橡皮擦通知=0,清除通知=0;
   Color c=new Color(255,0,0);int con=3;
   Button b_red,b_blue,b_green,
   清除,b_quit;
   public void init()
   {  addMouseMotionListener(this);
      b_red=new Button("画红色图形");
      b_blue=new Button("兰色图形");
      b_green=new Button("画绿色图形");
      b_quit=new Button("橡皮");
      清除=new Button("清除");
      add(b_red); add(b_green); add(b_blue); add(b_quit);add(清除);
      b_red.addActionListener(this); b_green.addActionListener(this);
      b_blue.addActionListener(this); b_quit.addActionListener(this);
      清除.addActionListener(this);
   }
   public void paint(Graphics g)
   {  if(x!=-1&&y!=-1&&橡皮擦通知==0&&清除通知==0)
         {  g.setColor(c); g.fillOval(x,y,con,con);
         }
       else if(橡皮擦通知==1&&清除通知==0)
        {  g.clearRect(x,y,10,10);
        }
       else if(清除通知==1&&橡皮擦通知==0)
        {  g.clearRect(0,0,getSize().width,getSize().height);
        }
   }
   public void mouseDragged(MouseEvent e)
   {  x=(int)e.getX();y=(int)e.getY(); repaint();
   }
    public void mouseMoved(MouseEvent e){ }
   public void update(Graphics g)
   {  paint(g);
```

图 **18.5**　鼠标自由作画

```
    }
    public void actionPerformed(ActionEvent e)
    {  if(e.getSource()==b_red)
         {  橡皮擦通知=0;清除通知=0; c=new Color(255,0,0);
         }
       else if(e.getSource()==b_green)
         {  橡皮擦通知=0;清除通知=0;  c=new Color(0,255,0);
         }
       else if(e.getSource()==b_blue)
         {  橡皮擦通知=0;清除通知=0; c=new Color(0,0,255);
         }
       if(e.getSource()==b_quit)
        {   橡皮擦通知=1;清除通知=0 ;
        }
       if(e.getSource()==清除)
        {   清除通知=1; 橡皮擦通知=0;repaint();
        }
    }
}
```

注:对鼠标作画程序的改进可参见二十六章例子 13.

18.3 鼠标事件的转移

假如我们监视一个容器上的鼠标事件,而容器中添加了一些组件,当在组件上进行单击鼠标,运动鼠标等操作时,容器将不知道这些操作的发生.

可以使用鼠标事件的转移将一个事件源发生的鼠标事件转移到另一个事件源上,也就是说,当用户的在某个事件源上单击鼠标时,可以通过鼠标事件的转移导致另一个事件源上发生鼠标事件　声东击西　.使用 **javax.swing** 包中的 **SwingUtilities** 类的静态方法

MouseEvent convertMouseEvent(Component source, MouseEvent sourceEvent, Component destination)

可以将 **source** 组件上发生的鼠标事件转移到组件 **destination**,该方法返回转移后的鼠标事件.

在下面的例子 **6** 中,用户单击一个按钮,然后拖动鼠标移动按钮的位置.

例子 6

```
import java.awt.*;import java.awt.event.*;import java.applet.*;
import javax.swing.SwingUtilities;
public class Example18_6 extends Frame implements MouseListener,MouseMotionListener
```

```
{  Button button;
   int x,y;
   boolean move=false;
   Example18_6()
   {  button=new Button("按钮");
      button.addMouseListener(this);
      button.addMouseMotionListener(this);
      addMouseMotionListener(this);
      setLayout(new FlowLayout());
      add(button);
      addWindowListener(new WindowAdapter()
              {   public void windowClosing(WindowEvent e)
                  { System.exit(0);
                  }
              });
      setBounds(10,10,200,180);
      setVisible(true); validate();
   }
   public void mousePressed(MouseEvent e) {}
   public void mouseReleased(MouseEvent e)
   {  move=false;
   }
   public void mouseEntered(MouseEvent e)  {}
   public void mouseExited(MouseEvent e) {}
   public void mouseClicked(MouseEvent e){}
   public void mouseMoved(MouseEvent e){}
   public void mouseDragged(MouseEvent e)
   {  Button b=null;
      if(e.getSource() instanceof Button) //在按钮上拖动鼠标导致按钮上发生鼠标事件.
         { b=(Button)e.getSource();
           move=true;
          //将鼠标拖动事件转移到棋盘,导致棋盘上发生鼠标拖动事件:
          e=SwingUtilities.convertMouseEvent(button,e,this);
          //上述语句导致窗口发生鼠标事件,再次导致 mouseDragged 执行,
          //并将从按钮转移得到的鼠标事件 e 传递给 mouseDragged 方法的参数.
         }
      if(e.getSource()==this)
         {  if(move&&b!=null)
              {  x=e.getX(); y=e.getY();
```

```
                int w=b.getSize().width,h=b.getSize().height;
                 b.setLocation(x-w/2,y-h/2);
              }
           }
      }
      public static void main(String args[])
      {  new Example18_6();
      }
   }
```

18.4 键盘事件

当按下,释放或敲击键盘上一个键时就发生了键盘事件,在 Java1.2 事件模式中,必须要有发生事件的事件源,当一个组件处于激活状态时,敲击键盘上一个键就导致这个组件上发生了键盘事件.

事件源使用 addKeyListener 方法获得监视器.

监视器是一个对象,创建该对象的类必须实现接口 KeyListener.接口 KeyListener 中有三个方法

```
public void keyPressed(KeyEvent e),
public void keyTyped(KeyEvent e),
public void KeyReleased(KeyEvent e).
```

当你按下键盘上某个键时,监视器就会发现,然后方法 keyPressed 会自动执行,并且 KeyEvent 类自动创建一个对象传递给方法 keyPressed 中的参数 e.方法 keyTyped 是 Pressedkey 和 keyReleased 方法的组合,当键被按下又释放时,keyTyped 方法被调用.

用 KeyEvent 类的 public int getKeyCode()方法,可以判断哪个键被按下,敲击或释放,getKeyCode 方法返回一个键码值 见表 18.1 .也可以用 KeyEvent 类的 public char getKeyChar()判断哪个键被按下,敲击或释放,getKeyCahr 方法返回键的字符.

表 18.1 键码表

键码	键
VK_F1-VK_F12	功能键 F1-F12
VK_LEFT	向左箭头键
VK_RIGHT	向右箭头键
VK_UP	向上箭头键
VK_DOWN	向下箭头键
VK_KP_UP	小键盘的向上箭头键
VK_KP_DOWN	小键盘的向下箭头键
VK_KP_LEFT	小键盘的向左箭头键
VK_KP_RIGHT	小键盘的向右箭头键
VK_END	END 键
VK_HOME	HOME 键

VK_PAGE_DOWN	向后翻页键
VK_PAGE_UP	向前翻页键
VK_PRINTSCREEN	打印屏幕键
VK_SCROLL_LOCK	滚动锁定键
VK_CAPS_LOCK	大写锁定键
VK_NUM_LOCK	数字锁定键
PAUSE	暂停键
VK_INSERT	插入键
VK_DELETE	删除键
VK_ENTER	回车键
VK_TAB	制表符键
VK_BACK_SPACE	退格键
VK_ESCAPE	Esc 键
VK_CANCEL	取消键
VK_CLEAR	清除键
VK_SHIFT	Shift 键
VK_CONTROL	Ctrl 键
VK_ALT	Alt 键
VK_PAUSE	暂停键
VK_SPACE	空格键
VK_COMMA	逗号键
VK_SEMICOLON	分号键
VK_PERIOD	.键
VK_SLASH	/键
VK_BACK_SLASH	\键
VK_0~VK_9	0~9 键
VK_A~VK_Z	a~z 键
VK_OPEN_BRACKET	[键
VK_CLOSE_BRACKET	]键
VK_UNMPAD0-VK_NUMPAD9	小键盘上的 0 至 9 键
VK_QUOTE	单引号　键
VK_BACK_QUOTE	但引号　键

下面例子 7 中有 10 个按钮,用户通过按动键盘上的方向键移动这些按钮.

例子 7(效果如图 18.6 所示)

```
import java.applet.*;import java.awt.*;
import java.awt.event.*;
public class Example18_7 extends Applet implements
KeyListener
{  Button b[]=new Button[10];
   int x,y;
   public void init()
   {  for(int i=0;i<=9;i++)
         {  b[i]=new Button(""+i);
```

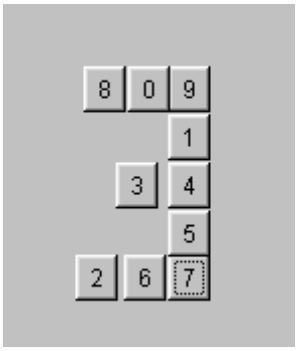


图 **18.6**　处理键盘事件

```
            b[i].addKeyListener(this);
            add(b[i]);
          }
      }
      public void keyPressed(KeyEvent e)
      {  Button button=(Button)e.getSource();
         x=button.getBounds().x;
         y=button.getBounds().y;
         if(e.getKeyCode()==KeyEvent.VK_UP)
           {  y=y-2;
              if(y<=0) y=0;
              button.setLocation(x,y);
           }
         else if(e.getKeyCode()==KeyEvent.VK_DOWN)
          {  y=y+2;
             if(y>=300) y=300;
             button.setLocation(x,y);
          }
         else if(e.getKeyCode()==KeyEvent.VK_LEFT)
          {  x=x-2;
             if(x<=0) x=0;
             button.setLocation(x,y);
          }
         else if(e.getKeyCode()==KeyEvent.VK_RIGHT)
          {  x=x+2;
             if(x>=300) x=300;
             button.setLocation(x,y);
          }
      }
      public void keyTyped(KeyEvent e) {}
      public void keyReleased(KeyEvent e) {}
  }
```

18.5 围棋对弈,迷宫程序及华容道

在这一小节我们综合这章的知识练习三个略有难度的例子.

1.围棋对弈

下面的例子 8 是本章一个具有一定难度的例子.双方通过点击鼠标进行围棋对弈.效果如图 18.7.在这个例子中,我们需要一个创建棋盘的类,我们通过一个面板的子类来创建这

个棋盘.通过点击鼠标左键实现在棋盘上布棋子.另外我们还需要创建棋子的类,我们分别用实现鼠标接口的画布子类来创建黑,白棋子,用鼠标双击棋子从当前棋盘上去掉棋子,用鼠标右键单击棋子实现悔棋.仔细阅读下面的例子 15,特别注意棋子类的技巧.

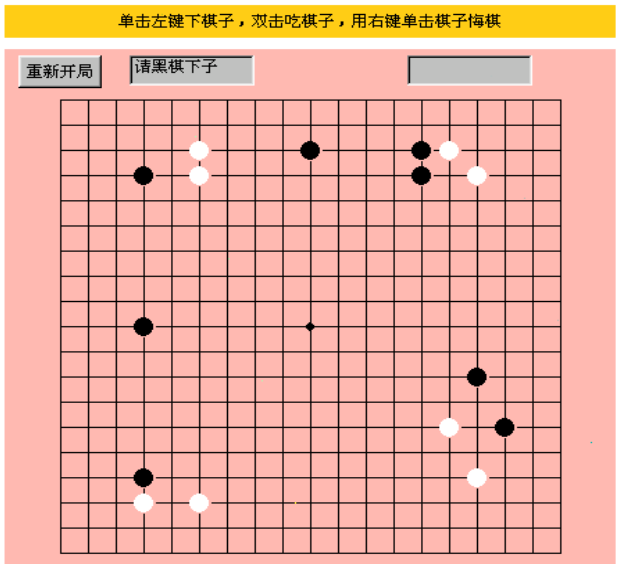


图 18.7　围棋对弈程序

例子 8　效果如图 18.7

```
import java.awt.*;import java.awt.event.*;
//创建棋盘的类
class ChessPad extends Panel  implements MouseListener,ActionListener
{  int x=-1,y=-1, 棋子颜色=1;                    //控制棋子颜色的成员变量.
   Button button=new Button("重新开局");     //控制重新开局的按扭.
   TextField text_1=new TextField("请黑棋下子"),
             text_2=new TextField();              //提示下棋的两个文本框.
   ChessPad()
   {  setSize(440,440);
      setLayout(null);setBackground(Color.pink);
      addMouseListener(this);add(button);button.setBounds(10,5,60,26);
      button.addActionListener(this);
      add(text_1);text_1.setBounds(90,5,90,24);
      add(text_2);text_2.setBounds(290,5,90,24);
      text_1.setEditable(false);text_2.setEditable(false);
   }
   public void paint(Graphics g)                    //绘制围棋棋盘的外观.
   {  for(int i=40;i<=380;i=i+20)
```

```
    {  g.drawLine(40,i,400,i);
    }
   g.drawLine(40,400,400,400);
   for(int  j=40;j<=380;j=j+20)
    { g.drawLine(j,40,j,400);
    }
    g.drawLine(400,40,400,400);
    g.fillOval(97,97,6,6); g.fillOval(337,97,6,6);
    g.fillOval(97,337,6,6);g.fillOval(337,337,6,6);
    g.fillOval(217,217,6,6);
}
public void mousePressed(MouseEvent e)      //当按下鼠标左键时下棋子.
{  if(e.getModifiers()==InputEvent.BUTTON1_MASK)
     {  x=(int)e.getX();y=(int)e.getY();       //获取按下鼠标时的坐标位置.
        ChessPoint_black chesspoint_black=new ChessPoint_black(this);
        ChessPoint_white chesspoint_white=new ChessPoint_white(this);
        int a=(x+10)/20,b=(y+10)/20;
        if(x/20<2||y/20<2||x/20>19||y/20>19)     //棋盘以外不下棋子.
         {}
        else
         {
           if(棋子颜色==1)                          //当棋子颜色是 1 时下黑棋子.
            {  this.add(chesspoint_black);
               chesspoint_black.setBounds(a*20-7,b*20-7,16,16);
               棋子颜色=棋子颜色*(-1);
               text_2.setText("请白棋下子");
               text_1.setText("");
            }
           else if(棋子颜色==-1)                    //当棋子颜色是-1 时下白棋子.
            {  this.add(chesspoint_white);
               chesspoint_white.setBounds(a*20-7,b*20-7,16,16);
                棋子颜色=棋子颜色*(-1);
               text_1.setText("请黑棋下子");
               text_2.setText("");
            }
         }
     }
}
public void mouseReleased(MouseEvent e){}
```

```
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}
    public void mouseClicked(MouseEvent e){}
    public void actionPerformed(ActionEvent e)
    {  this.removeAll();棋子颜色=1;
       add(button);button.setBounds(10,5,60,26);
       add(text_1);text_1.setBounds(90,5,90,24);
       text_2.setText("");text_1.setText("请黑棋下子");
       add(text_2);text_2.setBounds(290,5,90,24);
    }
}
//负责创建黑色棋子的类
class ChessPoint_black extends Canvas implements MouseListener
{  ChessPad chesspad=null;                          //棋子所在的棋盘.
   ChessPoint_black(ChessPad p)
   {  setSize(20,20);chesspad=p; addMouseListener(this);
   }
   public void paint(Graphics g)                     //绘制棋子的大小.
   {  g.setColor(Color.black);g.fillOval(0,0,14,14);
   }
   public void mousePressed(MouseEvent e)
   {  if(e.getModifiers()==InputEvent.BUTTON3_MASK)
       {  chesspad.remove(this);//当用鼠标右键点击棋子时,从棋盘中去掉该棋子  悔棋  .
          chesspad.棋子颜色=1;
          chesspad.text_2.setText("");chesspad.text_1.setText("请黑棋下子");
       }
   }
   public void mouseReleased(MouseEvent e){}
   public void mouseEntered(MouseEvent e) {}
   public void mouseExited(MouseEvent e) {}
   public void mouseClicked(MouseEvent e)
   {  if(e.getClickCount()>=2)
         chesspad.remove(this);              //当用左键双击该棋子时,吃掉该棋子.
   }
}
//负责创建白色棋子的类
class ChessPoint_white extends Canvas implements MouseListener
{  ChessPad chesspad=null;
   ChessPoint_white(ChessPad p)
```

```
    {  setSize(20,20);addMouseListener(this);
       chesspad=p;
    }
    public void paint(Graphics g)
    {  g.setColor(Color.white);g.fillOval(0,0,14,14);
    }
    public void mousePressed(MouseEvent e)
    {  if(e.getModifiers()==InputEvent.BUTTON3_MASK)
         {  chesspad.remove(this);chesspad.棋子颜色=-1;
            chesspad.text_2.setText("请白棋下子"); chesspad.text_1.setText("");
         }
    }
    public void mouseReleased(MouseEvent e){}
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}
    public void mouseClicked(MouseEvent e)
    {  if(e.getClickCount()>=2)
          chesspad.remove(this);
    }
}
public class Chess extends Frame  //添加棋盘的窗口.
{  ChessPad chesspad=new ChessPad();
   Chess()
   {  setVisible(true);
      setLayout(null);
      Label label=
      new Label("单击左键下棋子,双击吃棋子,用右键单击棋子悔棋",Label.CENTER);
      add(label);label.setBounds(70,55,440,26);
      label.setBackground(Color.orange);
      add(chesspad);chesspad.setBounds(70,90,440,440);
      addWindowListener(new WindowAdapter()
                {public void windowClosing(WindowEvent e)
                        {System.exit(0);
                        }
                });
       pack();setSize(600,550);
   }
public static void main(String args[])
   {  Chess chess=new Chess();
```

```
    }
}
```

2　走迷宫

下面是用键盘实现的走迷宫程序.走迷宫需要将一个物体　用一个按扭表示　限制在一定的区域内行走,这里要用到十四章讲过的 **Rectangle**　矩形　类　见 **14.3**　.

我们可以使用若干个矩形对象进行互相交叉形成迷宫　为了简化代码,例子中的迷宫是很简单的　.我们通过 **Grapghics** 对象的 **fillRect** 方法画出这个迷宫.用一个按扭代表走迷宫者,由于按扭是个矩形形状的组件,因此我们可以根据按扭的形状创建一个和按扭相关的矩形对象,当这个矩形对象和代表迷宫的矩形对象满足相交条件时,按扭根据键盘事件在迷宫里走来走去.

例子 9　效果如图 **18.8**

```
import java.awt.event.*;import java.applet.*;
import java.awt.*;
public class Move extends.Applet implements KeyListener,ActionListener
{  Button b_go=new Button("go"),
   b_replay=new Button("replay");
   Rectangle rect1,rect2,rect3;
   int b_x=0,b_y=0;
   public void init()
   {  b_go.addKeyListener(this);
      b_replay.addActionListener(this);
      setLayout(null);
      //代表迷宫的矩形
      rect1=new Rectangle(20,40,200,40);
      rect2=new Rectangle(200,40,24,240);
      rect3=new Rectangle(200,220,100,40);
      b_go.setBackground(Color.red);        //代表走迷宫者的按扭.
      add(b_go);b_go.setBounds(22,45,20,20);
      b_x=b_go.getBounds().x;b_y=b_go.getBounds().y;
      b_go.requestFocus() ;
      add(b_replay);b_replay.setBounds(2,2,45,16);//点击重新开始的按扭.
   }
   public void paint(Graphics g)
   {  //画出迷宫
      g.setColor(Color.green);
      g.fillRect(20,40,200,40);
      g.setColor(Color.yellow);
```

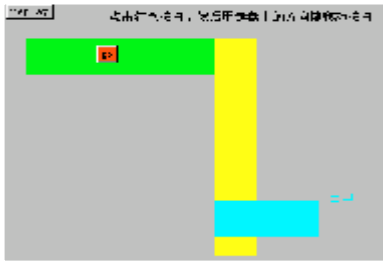


图 18.8　走迷宫

```
    g.fillRect(200,40,40,240);
    g.setColor(Color.cyan);
    g.fillRect(200,220,100,40);
    g.drawString("出口",310,220);
    g.setColor(Color.black);
    g.drawString("点击红色按扭,然后用键盘上的方向键移动按扭",100,20);
}
public void keyPressed(KeyEvent e)
{  //按键盘上的上下,左右键在迷宫中行走
   if((e.getKeyCode()==KeyEvent.VK_UP))
     { //创建一个和按扭  b_go 同样大小的矩形
       Rectangle rect=new Rectangle(b_x,b_y,20,20);
      //要求必须在迷宫内行走
        if(rect.intersects(rect1)||rect.intersects(rect2)||
                                    rect.intersects(rect3))
          {  b_y=b_y-2;b_go.setLocation(b_x,b_y);
          }
      }
    else if(e.getKeyCode()==KeyEvent.VK_DOWN)
      {  Rectangle rect=new Rectangle(b_x,b_y,20,20);
         if(rect.intersects(rect1)||rect.intersects(rect2)||
                                       rect.intersects(rect3))
          { b_y=b_y+2;b_go.setLocation(b_x,b_y);
          }
      }
    else if(e.getKeyCode()==KeyEvent.VK_LEFT)
      {  Rectangle rect=new Rectangle(b_x,b_y,20,20);
         if(rect.intersects(rect1)||rect.intersects(rect2)
                                    ||rect.intersects(rect3))
          {  b_x=b_x-2;b_go.setLocation(b_x,b_y);
          }
       }
   else if(e.getKeyCode()==KeyEvent.VK_RIGHT)
      {  Rectangle rect=new Rectangle(b_x,b_y,20,20);
        if(rect.intersects(rect1)||rect.intersects(rect2)||
                                     rect.intersects(rect3))
           { b_x=b_x+2;b_go.setLocation(b_x,b_y);
           }
   }
```

```
    }
    public void keyReleased(KeyEvent e){}
    public void keyTyped(KeyEvent e){}
    public void actionPerformed(ActionEvent e)
    {  b_go.setBounds(22,45,20,20);
       b_x=b_go.getBounds().x;b_y=b_go.getBounds().y;
       b_go.requestFocus() ;
    }
}
```

3 华容道

华容道是我们很熟悉的一个传统智力游戏.我们通过键盘事件来实现曹操,关羽等人物的移动,如图 18.9.

例子 10 效果如图 18.9

```
import java.awt.*;import java.applet.*;import java.awt.event.*;
class People extends Button implements FocusListener  //代表华容道人物的类.
{  Rectangle rect=null;
   int left_x,left_y;//按扭的左上角坐标.
   int width,height; //按扭的宽和高.
   String name; int number;
   People(int number,String s,int x,int y,int w,int h,Hua_Rong_Road road)
   {  super(s);
      name=s;this.number=number;
      left_x=x;left_y=y;
      width=w;height=h;setBackground(Color.orange);
      road.add(this);     addKeyListener(road);
      setBounds(x,y,w,h);addFocusListener(this);
      rect=new Rectangle(x,y,w,h);
   }
   public void focusGained(FocusEvent e)
   {  setBackground(Color.red);
   }
   public void focusLost(FocusEvent e)
   {   setBackground(Color.orange);
   }
}
public class Hua_Rong_Road extends Applet implements KeyListener,ActionListener
{  People people[]=new People[10];
```

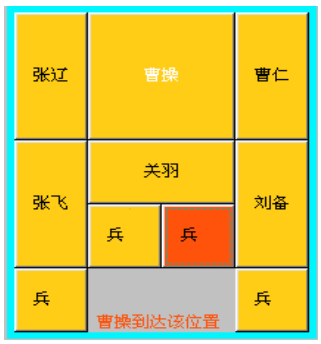


图 18.8 华容道

```
Rectangle left,right,above ,below;//华容道的边界 .
Button restart=new Button("重新开始");
public void init()
{  setLayout(null); add(restart);
   restart.setBounds(5,5,80,25);
   restart.addActionListener(this);
   people[0]=new People(0,"曹操",104,54,100,100,this);
   people[1]=new People(1,"关羽",104,154,100,50,this);
   people[2]=new People(2,"张飞",54,  154,50,100,this);
   people[3]=new People(3,"刘备",204,154,50,100,this);
   people[4]=new People(4,"张辽",54,  54,  50,100,this);
   people[5]=new People(5,"曹仁",204,  54,  50,100,this);
   people[6]=new People(6,"兵  ",54,254,50,50,this);
   people[7]=new People(7,"兵  ",204,254,50,50,this);
   people[8]=new People(8,"兵  ",104,204,50,50,this);
   people[9]=new People(9,"兵  ",154,204,50,50,this);
   people[9].requestFocus();
   left=new Rectangle(49,49,5,260);
   people[0].setForeground(Color.white) ;
   right=new Rectangle(254,49,5,260);
   above=new Rectangle(49,49,210,5);
   below=new Rectangle(49,304,210,5);
}
public void paint(Graphics g)
{  //画出华容道的边界:
   g.setColor(Color.cyan);
   g.fillRect(49,49,5,260);//left.
   g.fillRect(254,49,5,260);//right.
   g.fillRect(49,49,210,5); //above.
   g.fillRect(49,304,210,5);//below.
   //提示曹操逃出位置和按键规则:
   g.drawString("点击相应的人物,然后按键盘上的上下左右箭头移动",100,20);
   g.setColor(Color.red);
   g.drawString("曹操到达该位置",110,300);
}
public void keyPressed(KeyEvent e)
{  People man=(People)e.getSource();//获取事件源.
    man.rect.setLocation(man.getBounds().x,  man.getBounds().y);
    if(e.getKeyCode()==KeyEvent.VK_DOWN)
```

```
    {  man.left_y=man.left_y+50;      //向下前进 50 个单位.
       man.setLocation(man.left_x,man.left_y);
       man.rect.setLocation(man.left_x,man.left_y);
         //判断是否和其它人物或下边界出现重叠,如果出现重叠就退回 50 个单位距离.
       for(int i=0;i<10;i++)
           {if((man.rect.intersects(people[i].rect))&&(man.number!=i))
               {  man.left_y=man.left_y-50;
                  man.setLocation(man.left_x,man.left_y);
                  man.rect.setLocation(man.left_x,man.left_y);
               }
           }
        if(man.rect.intersects(below))
            {  man.left_y=man.left_y-50;
               man.setLocation(man.left_x,man.left_y);
               man.rect.setLocation(man.left_x,man.left_y);
            }
    }
  if(e.getKeyCode()==KeyEvent.VK_UP)
    {  man.left_y=man.left_y-50;      //向上前进 50 个单位.
       man.setLocation(man.left_x,man.left_y);
       man.rect.setLocation(man.left_x,man.left_y);
       //判断是否和其它人物或上边界出现重叠,如果出现重叠就退回 50 个单位距离.
       for(int i=0;i<10;i++)
           {  if((man.rect.intersects(people[i].rect))&&(man.number!=i))
                  {  man.left_y=man.left_y+50;
                     man.setLocation(man.left_x,man.left_y);
                      man.rect.setLocation(man.left_x,man.left_y);
                   }
           }
        if(man.rect.intersects(above))
           {  man.left_y=man.left_y+50;
              man.setLocation(man.left_x,man.left_y);
              man.rect.setLocation(man.left_x,man.left_y);
           }
    }
 if(e.getKeyCode()==KeyEvent.VK_LEFT)
    {  man.left_x=man.left_x-50;      //向左前进 50 个单位.
       man.setLocation(man.left_x,man.left_y);
       man.rect.setLocation(man.left_x,man.left_y);
```

```
        //判断是否和其它人物或左边界出现重叠,如果出现重叠就退回 50 个单位距离.
        for(int i=0;i<10;i++)
             {  if((man.rect.intersects(people[i].rect))&&(man.number!=i))
                  { man.left_x=man.left_x+50;
                    man.setLocation(man.left_x,man.left_y);
                    man.rect.setLocation(man.left_x,man.left_y);
                   }
             }
        if(man.rect.intersects(left))
            {  man.left_x=man.left_x+50;
               man.setLocation(man.left_x,man.left_y);
               man.rect.setLocation(man.left_x,man.left_y);
             }
      }
    if(e.getKeyCode()==KeyEvent.VK_RIGHT)
     {  man.left_x=man.left_x+50;      //向右前进 50 个单位.
        man.setLocation(man.left_x,man.left_y);
        man.rect.setLocation(man.left_x,man.left_y);
       //判断是否和其它人物或右边界出现重叠,如果出现重叠就退回 50 个单位距离.
        for(int i=0;i<10;i++)
            {  if((man.rect.intersects(people[i].rect))&&(man.number!=i))
                  { man.left_x=man.left_x-50;
                    man.setLocation(man.left_x,man.left_y);
                    man.rect.setLocation(man.left_x,man.left_y);
                   }
             }
         if(man.rect.intersects(right))
             {  man.left_x=man.left_x-50;
                man.setLocation(man.left_x,man.left_y);
                man.rect.setLocation(man.left_x,man.left_y);
              }
       }
}
public void keyTyped(KeyEvent e){}
public void keyReleased(KeyEvent e){}
public void actionPerformed(ActionEvent e)
{  this.removeAll();
   this.init();
}
```

```
}
```

习题十八

1 完善例子 10,要求能改变"画笔"的粗细及橡皮擦的大小.

2 进一步改进例子 7,要求一个按钮在移动时,不允许和其它按钮相交.

3 上机实习下列程序,掌握复合键的使用.

```
import java.awt.*;import java.awt.event.*;
public class ComKey extends Frame implements KeyListener
{  Button b=new Button("学习复合键的使用");
   ComKey()
   {   setSize(300,300);setVisible(true);
       setLayout(new FlowLayout());
       add(b);b.addKeyListener(this);
       pack();setSize(300,300);
   }
   public void keyPressed(KeyEvent e)
   {  if((e.getKeyCode()==KeyEvent.VK_X))
      {  if((e.getModifiers()&InputEvent.CTRL_MASK)!=0)//判断是否按下 Ctrl+X
         b.setLocation(150,40);
      }
      if((e.getKeyCode()==KeyEvent.VK_Y))
      {  if((e.getModifiers()&InputEvent.CTRL_MASK)!=0)//判断是否按下 Ctrl+Y
         b.setLocation(40,150);
      }
   }
   public void keyTyped(KeyEvent e) {}
   public void keyReleased(KeyEvent e) {}
   public static void main(String args[])
   {  ComKey win=new ComKey();
   }
}
```

第十九章　Java 多线程机制

以往我们开发的程序大多是单线程的,即一个程序只有一条从头至尾的执行线索.然而现实世界中的很多过程都具有多条线索同时动作的特性　例如,我们可以一边看电视,一边活动我们的胳膊,如果不容许这样做,我们会感觉很难受.再如一个网络服务器可能需要同时处理多个客户机的请求等.

Java 语言的一大特性点就是内置对多线程的支持.多线程是指同时存在几个执行体,按几条不同的执行线索共同工作的情况,它使得编程人员可以很方便地开发出具有多线程功能,能同时处理多个任务的功能强大的应用程序.虽然执行线程给人一种几个事件同时发生的感觉,但这只是一种错觉,因为我们的计算机在任何给定的时刻只能执行那些线程中的一个.为了建立这些线程正在同步执行的感觉,Java 快速地把控制从一个线程切换到另一个线程.

19.1 Java 中的线程

1　程序,进程与线程

程序是一段静态的代码,它是应用软件执行的蓝本.

进程是程序的一次动态执行过程,它对应了从代码加载,执行至执行完毕的一个完整过程,这个过程也是进程本身从产生,发展至消亡的过程.如果把银行一天的工作比作一个进程,那么早上打铃上班是进程的开始,晚上打下班铃是进程的结束.

线程是比进程更小的执行单位.一个进程在其执行过程中,可以产生多个线程,形成多条执行线索,每条线索,即每个线程也有它自身的产生,存在和消亡的过程,也是一个动态的概念.就像银行一天的工作开始后,可以有多个不同的"线程"为客户服务,如财会部门,出纳部门,保安部门等.我们知道,每个进程都有一段专用的内存区域,与此不同的是,线程间可以共享相同的内存单元　包括代码与数据　,并利用这些共享单元来实现数据交换,实时通信与必要的同步操作.比如在银行一天的工作开始后,财会部门,出纳部门,保安部门这三个线程共享银行的电力资源,财会部门,出纳部门可能共享银行的帐目数据等.多线程的程序能更好地表达和解决现实世界的具体问题,是计算机应用开发和程序设计的一个必然发展趋势.

2　线程的状态与生命周期

每个 Java 程序都有一个缺省的主线程.对于应用程序,主线程是 main()方法执行的线索　之所以称为主线程,而不称做进程,是因为我们的程序在执行时,我们的计算机仍可以浏览网页,也可以后台打印等　.对于 Applet,主线程指挥浏览器加载并执行 Java 小程序.要想实现多线程,必须在主线程中创建新的线程对象.Java 语言使用 Thread 类及其子类的对象来表示线程,新建的线程在它的一个完整的生命周期中通常要经历如下的四种状态

1　新建

当一个 Thread 类或其子类的对象被声明并创建时,新生的线程对象处于新建状态.此

时它已经有了相应的内存空间和其他资源.

2 运行

线程创建之后就具备了运行的条件,一旦轮到它来享用 CPU 资源时,就可以脱离创建它的主线程独立开始自己的生命周期了.

3 中断

一个正在执行的线程可能被人为地中断,让出 CPU 的使用权,暂时中止自己的执行,进入阻塞状态.阻塞时它不能进入排队队列,只有当引起阻塞的原因被消除时,线程才可以转入就绪状态,重新进到线程队列中排队等待 CPU 资源,以便从原来终止处开始继续运行.

4 死亡

处于死亡状态的线程不具有继续运行的能力.线程死亡的原因有二,一个是正常运行的线程完成了它的全部工作,另一个是线程被提前强制性地终止.所谓死亡状态就是线程释放了实体,即释放分配给线程对象的内存

3 线程调度与优先级

处于就绪状态的线程首先进入就绪队列排队等候处理器资源,同一时刻在就绪队列中的线程可能有多个.多线程系统会给每个线程自动分配一个线程的优先级,任务较紧急重要的线程,其优先级就较高 相反则较低.在线程排队时,优先级高的线程可以排在较前的位置,能优先享用到处理器资源,而优先级较低的线程则只能等到排在它前面的高优先级线程执行完毕之后才能获得处理器资源.对于优先级相同的线程,则遵循队列的"先进先出"的原则,即先进入就绪状态排队的线程被优先分配到处理器资源,随后才为后进入队列的线程服务.

当一个在就绪队列中排队的线程被分配到处理器资源而进入运行状态之后,这个线程就称为是被 "调度"或被线程调度管理器选中了.线程调度管理器负责管理线程排队和处理器在线程间的分配,一般都配有一个精心设计的线程调度算法.在 Java 系统中,线程调度依据优先级基础上的"先到先服务"原则.

Thread 类的方法 setPriority(int a)可以设置线程优先级,使之符合程序的特定需要.a 取值是

Thread.MIN_PRIORITY,Thread.MAX_PRIORITY,Thread.NORM_PRIORITY.

线程的默认级别是 Thread.NORM_PRIORITY.

19.2 Thread 类与 Runnable 接口

Java 中编程实现多线程应用有两种途径 一种是用 Thread 类的子类创建线程,另一种是用 Thread 类创建线程.

1 Thread 类

Thread 类综合了 Java 程序中一个线程需要拥有的属性和方法,主要有

1　构造函数

public Thread(Runnable target)　创建线程对象,参数 target 称为被创建线程的目标对象.创建目标对象 target 的类负责实现 Runnable 接口,给出该接口中 run()方法的方法体.

利用构造函数创建新线程对象之后,这个对象中的有关数据被初始化,从而进入线程的生命周期的第一个状态—新建状态.

(2)其他主要方法

start()　线程调用该方法将启动线程,使之从新建状态转入就绪状态并进入就绪队列排队,一旦轮到它来享用 CPU 资源时,就可以脱离创建它的主线程独立开始自己的生命周期了.

run()　Thread 类的 run()方法与 Runnable 接口中的 run()方法的功能和作用相同,都用来定义线程对象被调度之后所执行的操作,都是系统自动调用而用户程序不得引用的方法.系统的 Thread 类中,run()方法没有具体内容,所以用户程序需要创建自己的 Thread 类的子类,并重写 run()方法来覆盖原来的 run()方法.当 run 方法执行完毕,线程就变成死亡状态,所谓死亡状态就是线程释放了实体,即释放分配给线程对象的内存.在线程没有结束 run 方法之前,不赞成让线程再调用 start 方法,否则将发生 **ILLegalThreadStateException** 异常.

sleep(int millsecond)　线程的调度执行是按照其优先级的高低顺序进行的,当高级线程不完成,即未死亡时,低级线程没有机会获得处理器.有时,优先级高的线程需要优先级低的线程做一些工作来配合它,或者优先级高的线程需要完成一些费时的操作,此时优先级高的线程应该让出处理器,使优先级低的线程有机会执行.为达到这个目的,优先级高的线程可以在它的 run()方法中调用 sleep 方法来使自己放弃处理器资源,休眠一段时间.休眠时间的长短由 sleep 方法的参数决定,millsecond 是毫秒为单位的休眠时间

isAlive()方法　检查线程是否仍然存活的方法.

currentThread()　判断当前正在占有 CPU 的线程.

2　Runnable 接口

Runnable 接口只有一个方法 run(),所有实现 Runnable 接口的用户类都必须具体实现这个 run()方法,为它提供方法体并定义具体操作.当用 Thread 类的构造方法 Thread(Runnable target)创建线程对象时,构造方法中的参数必须是一个具体的对象,该对象称作线程的目标对象,创建目标对象的类必须要实现 Runnable 接口.当线程调用 start 方法时,一旦轮到它来享用 CPU,目标对象就会自动调用接口中的 run 方法.Runnable 接口中的这个 run()方法是一个较特殊的方法,它可以被运行系统自动识别和执行,也就是说,当线程被调度并转入运行状态时,它所执行的就是 run()方法中规定的操作.所以,一个实现了 Runnable 接口的类实际上定义了一个主线程之外的新线程的操作.

19.3 如何在程序中实现多线程

如前所述,在程序中实现多线程有两个途径　一种是用 Thread 类的子类创建线程,另一种是用 Thread 类创建线程.

1　用 Thread 类的子类创建线程

当编写 Thread 类的子类时,可以在子类中重写父类的 run 方法,该方法中包含了线程的操作.这样程序需要建立自己的线程时,只需要创建一个已定义好的 Thread 子类的实例就可以了.当创建的线程调用 start()方法开始运行时,run()方法将被自动执行.

下面的例子 1 模拟左右手轮流写字.

例子 1

```
public class Example19_1
{  public  static void main(String args[])
   {  Lefthand left;
      Righthand right;
      left=new Lefthand() ;//创建线程.
      right=new Righthand();
      left.start();   //线程开始运行后,Lefthand 类中的 run 方法将被执行.
      right.start();
   }
}
class Lefthand extends Thread
{  public void run()
   {  for(int i=1;i<=5;i++)
       {  System.out.print("A");
          try {
               sleep(500);
              }
         catch(InterruptedException e){}
       }
   }
}
class Righthand extends Thread
{  public void run()
   {  for(int i=1;i<=5;i++)
         {  System.out.print("B");
           try{ sleep(300);
              }
```

```
            catch(InterruptedException e){}
        }
    }
}
```

在上述例子中,我们在 **main** 主线程中创建了两个新的线程 **lethand** 和 **righthand**.当 **lefthand** 调用 **start()**开始运行时,类 **Lefthand** 中的 **run()**将自动被执行.

我们来分析一下上面程序的输出结果.Left 线程首先开始执行,这时 Lefthand 类中的 run 方法开始执行,输出 A 后,left 主动"休息"500 毫秒,让出了 CPU.这时正在排队等待 CPU 的 right 线程的 run 方法马上被执行,输出 B,再主动让出 CPU 300 毫秒后又来排队等待 CPU 服务,这时 right 发现 left 还没有"醒来",即没有来排队抢占 CPU,因此 left 的 run 方法被继续执行,又输出 A.程序的输出结果是 ABBABBABAA.

2 使用 Thread 类创建线程

用 Thread 类的构造方法 Thread(Runnable target)创建线程对象时,构造方法中的参数必须是一个具体的对象,该对象称作线程的目标对象,创建目标对象的类必须要实现 Runnable 接口.当线程调用 start 方法时,一旦轮到它来享用 CPU,目标对象就会自动调用接口中的 run 方法.也就是说,当线程被调度并转入运行状态时,它所执行的就是 run()方法中所规定的操作.

下面的例子 2 是一个应用程序,这个应用程序在创建窗口的同时又使用 Thread 类创建了一个新的线程,该线程的目标对象就是当前窗口.该线程负责让窗口中的一个按钮变化它的大小.

例子 2

```
import java.awt.*;import java.awt.event.*;
public class Example19_3
{  public static void main(String args[])
   {  Mywin win=new Mywin();win.pack();
   }
}
class Mywin extends Frame implements Runnable
{  Button b=new Button("ok");int x=5;
   Thread bird=null;
   Mywin()
   {  setBounds(100,100,120,120);setLayout(new FlowLayout());
      setVisible(true);
      add(b);b.setBackground(Color.green);
      addWindowListener(new WindowAdapter()
           {  public void windowClosing(WindowEvent e)
```

```
            {  System.exit(0);
            }
         });
      bird=new Thread(this);//创建一个新的线程,窗口做目标对象,
                            //替线程 bird 实现接口 Runnable.
      bird.start();          //在创建窗口时又开始了线程 bird.
   }
   public void run()        //实现 bird 的操作.
   {  while(true)
       {  x=x+1;
          if(x>100) x=5;
          b.setBounds(40,40,x,x);
          try{  bird.sleep(200);
             }
          catch(InterruptedException e){}
       }
   }
}
```

下面的例子 3 是带滚动字幕的小词典,当用户在一个文本框中输入英文单词回车时,另一个文本框显示汉语解释.程序中的一个线程对象负责滚动地显示"欢迎使用本字典".用户通过在文本框中输入 moon 单词,来杀死这个线程,即让线程结束 run 方法,进入死亡状态.

例子 3(效果如图 19.1)

```
import java.applet.*;import java.awt.*;import java.awt.event.*;
public class Example19_3 extends Appletimplements ActionListener,Runnable
{  TextField  text1,text2;
   boolean boo;  int x=0;
   Thread Scrollwords=null;
   public void init()
   {  Scrollwords=new Thread(this);
      text1=new TextField(10);
      text2=new TextField(10);
      add(text1); add(text2);
      text1.addActionListener(this);
   }
   public void start()
   {  boo=false;
      try{  Scrollwords.start();
         }
```

图 19.1　滚动字幕线程

```
        catch(Exception ee) { }
    }
    public void run ()
    {  while(true)
         {  x=x+5;
            if(x>200)
               x=0;
            epaint();
            try{  Scrollwords .sleep(80);
               }
            catch(InterruptedException e){}
            if(boo)
              {  return;            //结束 run 方法,导致线程死亡.
              }
         }
    }
    public void paint(Graphics g)
    {  g.drawString("欢 迎 使 用 本 字 典",x,70);
    }
    public void actionPerformed(ActionEvent e)
    {  if(text1.getText().equals("boy"))
       {  text2.setText("男孩");
       }
      else if(text1.getText().equals("moon"))
       {  boo=true;          //将 boo 的值设置为 true,以便杀死线程 Scrollwords.
       }
      else
       {  text2.setText("没有该单词");
       }
    }
}
```

下面是一个左手画圆右手画方的例子.我们在主线程中又创建了两个线程 left,right,其中一个负责画圆,另一个负责画方.在这个例子中我们使用了容器类的一个方法 getGraphics(),来获取一个 Graphics 对象 可以理解为一个画笔 .

例子 4(效果如图 19.2)

```
import java.applet.*;import java.awt.*;import java.awt.event.*;
public class Example19_4 extends Applet implements Runnable
{  Thread left ,right; Graphics mypen; int x,y;
```

```
    public void init()
    {  left=new Thread(this);right=new Thread(this);
       x=10;y=10;mypen=getGraphics();
    }
    public void start()
    {  left.start();right.start();
    }
    public void run()
    {  while(true)
       { if(Thread.currentThread()==left)
          { x=x+1;
            if(x>240) x=10;
            mypen.setColor(Color.blue);
            mypen.clearRect(10,10,300,40);
            mypen.drawRect(10+x,10,40,40);
            try{  left.sleep(60);
               }
            catch(InterruptedException e){}
           }
         else if(Thread.currentThread()==right)
           {  y=y+1;
              if(y>240) y=10; mypen.setColor(Color.red);
               mypen.clearRect(10,90,300,40);
               mypen.drawOval(10+y,90,40,40);
               try{  right.sleep(60);
                  }
               catch(InterruptedException e){}
            }
       }
    }
    public void stop()
    {  left=null;right=null;
    }
}
```

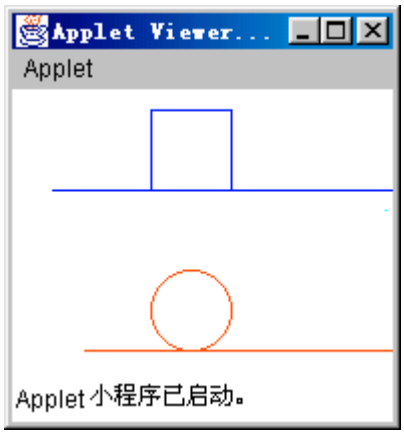
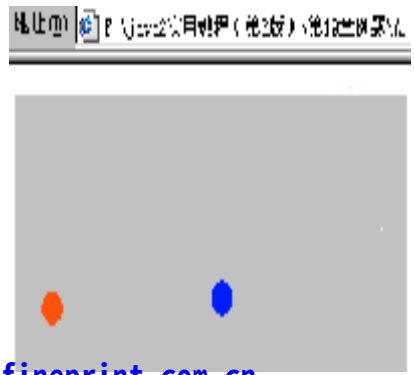


图 19.2　双线程绘画程序

例子 5 是模拟平抛运动,一个球自由落下,一个球同时水平抛出,二者同落地.



例子 5(效果如图 19.3 所示)

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

```
import java.applet.*;
public class Example19_5 extends Applet
                 implements Runnable
{  Thread 红色球,兰色球;
   Graphics redPen,bluePen;
   double t=0;
   public void init()
   {  红色球=new Thread(this);
      兰色球=new Thread(this);
      redPen=getGraphics();bluePen=getGraphics();
      redPen.setColor(Color.red); bluePen.setColor(Color.blue);
   }
   public void start()
   {  红色球.start();兰色球.start();
   }
   public void run()
   {  while(true)
      {  t=t+0.2;
         if(Thread.currentThread()==红色球)
            {  if(t>20)  t=0;
               redPen.clearRect(0,0,38,300);
               redPen.fillOval(20,(int)(1.0/2*t*t*3.8),16,16);
               try{  红色球.sleep(50);
                  }
               catch(InterruptedException e){}
             }
          else if(Thread.currentThread()==兰色球)
            {  bluePen.clearRect(38,0,500,300);
               bluePen.fillOval(38+(int)(16*t),(int)(1.0/2*t*t*3.8),16,16);
               try{  兰色球.sleep(50);
                  }
               catch(InterruptedException e){}
            }
      }
   }
}
```

19.4 Thread 类的静态方法 sleep()

sleep()方法可以被类名直接调用,因此当需要延时程序的执行时可以使用如下语句

```
Thread.sleep(int times);
```

在下面例子 6 中点击按钮后,每隔半秒程序发出"嘟"声.

例子 6

```
import java.applet.*;import java.awt.*;
import java.awt.event.*;
public class Example19_6 extends Applet implements ActionListener
{  Toolkit toolkit;Button button;
   public void init()
   {  toolkit=getToolkit();//获得一个工具包对象
      button=new  Button("确定");add(button);
      button.addActionListener(this);
   }
   public void actionPerformed(ActionEvent e)
   {  if(e.getSource()==button)
       {  for(int i=0;i<=9;i++)
             {  toolkit.beep();
                try {  Thread.sleep(500);
                    }       //每隔半秒钟发出嘟声
               catch(InterruptedException e1){}
             }
       }
   }
}
```

19.5 线程同步

Java 使我们可以创建多个线程,在处理多线程问题时,我们必须注意这样一个问题　当两个或多个线程同时访问同一个变量,并且一个线程需要修改这个变量.我们应对这样的问题作出处理,否则可能发生混乱,比如一个工资管理负责人正在修改雇员的工资表,而一些雇员也正在领取工资,如果容许这样做必然出现混乱.因此,工资管理负责人正在修改工资表时　包括他喝杯茶休息一会　,将不容许任何雇员领取工资,也就是说这些雇员必须等待.

在处理线程同步时,要做的第一件事就是要把修改数据的方法用关键字　synchronized 来修饰.一个方法使用关键字 synchronized 修饰后,当一个线程 A 使用这个方法时,其他线程想使用这个方法时就必须等待,直到线程 A 使用完该方法.

在下面的这个例子中有两个线程　会计和出纳,他俩共同拥有一个帐本.他俩都可以使用

存取方法对帐本进行访问,会计使用存取方法时,向帐本上写入存钱记录　出纳使用存取方法时,向帐本写入取钱记录.因此,当会计正在使用帐本时,出纳被禁止使用,反之也是这样.比如,会计每次使用帐本时,在帐本上存入 90 万元,但在存入这比钱时.每写入 30 万元,就喝口茶,那么他喝茶休息时　注意,这时存钱这件事还没结束,即会计还没有使用完存取方法　,出纳仍不能使用帐本.从周一到周三会计和出纳都要使用帐本,我们要保证其中一人使用帐本时,另一个人将必须等待.

例子 7

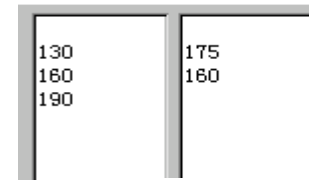
```
import java.applet.*;import java.awt.*;
import java.awt.event.*;
public class Example19_7 extends Applet implements Runnable
{  int money=100;TextArea text1,text2;
   Thread 会计,出纳;
   public void init()
   {  会计=new Thread(this); 出纳=new Thread(this);//创建两个线程:会计,出纳.
      text1=new TextArea(20,8); text2=new TextArea(20,8);
      add(text1);add(text2);
   }
   public void start()
   {  会计.start();出纳.start();                    //线程开始.
   }
   public synchronized void 存取(int number) //存取方法.
   {  if(Thread.currentThread()==会计)
       {  for(int i=1;i<=3;i++) //会计使用存取方法存入 90 元,存入 30 元,稍歇一下,
            {  money=money+number;          //这时出纳仍不能使用存取方法,
               try {  Thread.sleep(1000);  //因为会计还没使用完存取方法.
                   }
               catch(InterruptedException e){}
               text1.append("\n"+money);
            }
       }
      else if(Thread.currentThread()==出纳)
      {  for(int i=1;i<=2;i++) //出纳使用存取方法取出 30 元,取出 15 元,稍歇一下,
            {  money=money-number/2;        //这时会计仍不能使用存取方法,
               try { Thread.sleep(1000); //因为出纳还没使用完存取方法.
                   }
               catch(InterruptedException e){}
               text2.append("\n"+money);
            }
```

```
        }
    }
    public void run()
    {  if(Thread.currentThread()==会计||Thread.currentThread()==出纳)
        {  for(int i=1;i<=3;i++) //从周一到周三会计和出纳都要使用帐本.
            {  存取(30);
            }
        }
    }
}
```

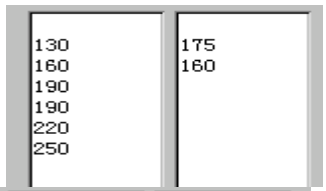
上述程序中,"会计"这个线程先开始,然后"出纳"线程开始,两个线程都去自动执行 run 方法,在这个 run 方法中,调用了存取方法,对帐本操作.程序执行时,程序中的左面文本区首先出现了会计使用存取方法时,帐本上钱的数目变化,周一分 3 次存入了 90 元,如图 19.4(a)所示,左面的文本区显示会计的存入钱后,帐本上钱的数目,右边文本区显示出纳取出钱后帐本钱的数目.当周一会计使用完后,出纳马上使用存取方法分 2 次取走 30 元 因为周一出纳也正在等待使用帐本 ,这时程序的执行出现图 19.4(b)所示效果.周二会计又存入 90 元,程序执行出现下图 19.4(c),当周二会计使用完后,出纳马上使用存取方法分 2 次取走 30 元,这时程序的执行出现图 19.4(d).周三会计又存入 90 元,程序执行出现图 19.4(e).会计使用完帐本后,出纳马上使用存取方法分 2 次取走 30 元,这时程序的执行出现图 19.4(f).

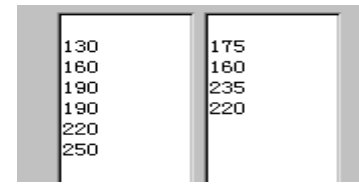
如果存取方法不用 synchronized 限制,程序运行结果是图 19.4(g).

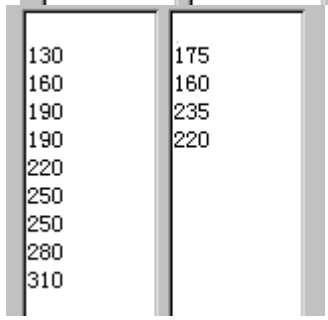
(a)



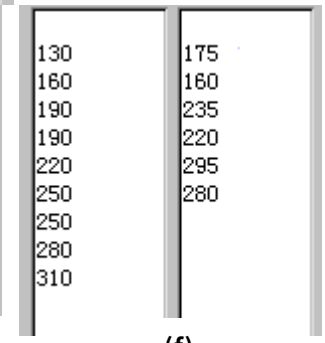
(b)



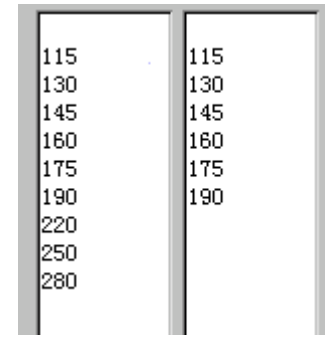




(e)



(f)



(g)

19.6 在同步方法中使用 wait(),notify 和 notifyAll()方法

在上一节我们已经知道,当一个线程正在使用一个同步方法时　用 synchronized 修饰的方法　,其它线程就不能使用这个同步方法.对于同步方法,有时涉及到某些特殊情况,比如当你在一个售票窗口排队购买电影票时,如果你给售票员的钱不是零钱,而售票员又没有零钱找给你,那么你就必须等待,并允许你后面的人买票,以便售票员获得零钱给你.如果第 2 个人仍没有零钱,那么你俩必须等待,并允许后面的人买票.

当一个线程使用的同步方法中用到某个变量,而此变量又需要其它线程修改后才能符合本线程的需要,那么可以在同步方法中使用 wait()方法.使用 wait 方法可以中断方法的执行,使本线程等待,暂时让出 CPU 的使用权,并允许其它线程使用这个同步方法.其它线程如果在使用这个同步方法时不需要等待,那么它使用完这个同步方法的同时,应当用 notifyAll()方法通知所有的由于使用这个同步方法而处于等待的线程结束等待.曾中断的线程就会从刚才的中断处继续执行这个同步方法,并遵循"先中断先继续"的原则　注意,如果使用 notify(), 那么只是通知第一个处于等待的线程结束等待　.

在下面的例子中,为了我们避免复杂数学算法,我们模拟两个人,张某和李某买电影票,售票员只有两张五元的钱,电影票 5 元钱一张.张某拿二十元一张的新人民币排在李的前面买票,李某拿一张 5 元的人民币买票.因此张某必须等待.

例子 8(效果如图 19.5 所示)

```
import java.applet.*;import java.awt.*;
import java.awt.event.*;
class 售票员
{  int 五元钱的个数=2,十元钱的个数=0,二十元钱的个数=0; String s=null;
   public synchronized void  售票规则(int money)
   {  if(money==5)      //如果使用该方法的线程传递的参数是 5,就不用等待.
        {  五元钱的个数=五元钱的个数+1;
           s= "给您入场卷您的钱正好";
           Example19_8.text.append("\n"+s);
        }
      else if(money==20)
```

```
        {  while(五元钱的个数<3)
           {  try { wait();     //如果使用该方法的线程传递的参数是 20 须等待.
                  }
              catch(InterruptedException e){}
           }
           五元钱的个数=五元钱的个数-3;
           二十元钱的个数=二十元钱的个数+1;
           s="给您入场卷"+"  您给我 20,找您 15 元";
           Example19_8.text.append("\n"+s);
        }
        notifyAll();
    }
}
public class Example19_8 extends Applet implements Runnable
{  售票员 王小姐;
   Thread 张平,李明;  //创建两个线程.
   static TextArea text;
   public void init()
   {  张平=new Thread(this);李明=new Thread(this);
      text=new TextArea(10,30);add(text);
      王小姐=new 售票员();
   }
   public void start()
   {  张平.start();李明.start();
   }
   public void run()
   {  if(Thread.currentThread()==张平)
        {  王小姐.售票规则(20);
        }
      else if(Thread.currentThread()==李明)
        {  王小姐.售票规则(5);
        }
   }
}
```

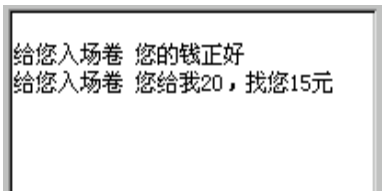


图19.5 售票小程序

19.7 线程的 interrupt()方法

一个线程在执行完 run 方法后就自动地消灭了,如果想在 run 方法执行完之间就消灭该线程,可以让线程调用 interrupt()方法,这时该线程会捕获 InteruptedException 异常,在处理

该异常的语句里告诉线程结束 run 方法的执行,比如使用 return 语句　方法体中一旦执行了 return 语句,则该方法立刻结束执行　.在下面的例子中点击 start 按扭线程开始工作　每隔一秒钟显示一次当前时间　点击 stop 按扭后,线程就结束了生命,释放了实体,即释放线程对象的内存,再点击 start 按扭,线程已经不能再开始工作了.在下面的程序中,每当单击 start 按钮时,我们都让线程调用 isAlive 方法,判断线程是否还有实体,如果线程是死亡状态就再分配实体给线程.

当把一个线程委派给一个组件事件时要格外小心,比如点击一个按扭让线程开始运行,那么当这个线程在执行完 run 方法之前,客户可能会随时再次点击该按扭,这时就会发生 ILLegalThreadStateException 异常.

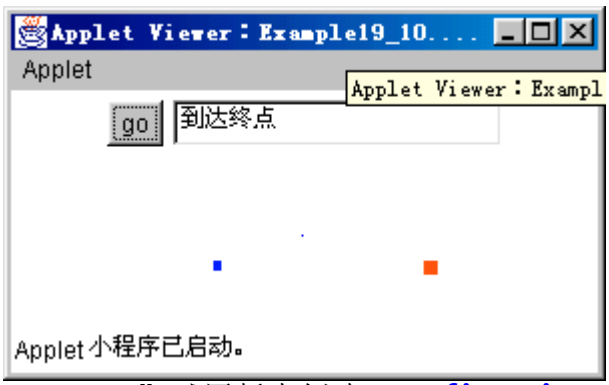
例子 9

```
import java.awt.event.*;
import java.awt.*;import java.util.Date;
class Example19_9 extends Frame implements Runnable,ActionListener
{  Thread thread=null; TextArea text=null;
   Button b_start=new Button("Start"),b_stop=new Button("Stop");
   Example19_9()
   {  thread=new Thread(this);
      text=new TextArea();add(text,"Center");
      Panel p=new Panel();p.add(b_start);p.add(b_stop);
      b_start.addActionListener(this);
      b_stop.addActionListener(this) ;
     add(p,"North");setVisible(true);
     setSize(500,500);pack();setSize(500,500);
     setResizable(false);                    //让窗口的大小不能被调整.
     addWindowListener(new WindowAdapter()
                  {  public void windowClosing(WindowEvent e)
                    {System.exit(0);
                    }
                  });
   }
   public void actionPerformed(ActionEvent e)
   {  if(e.getSource()==b_start)
        {
          if(!(thread.isAlive()))
             {  thread=new Thread(this);
             }
         try {  thread.start();
             }
```

```
            catch(Exception e1)
                {  text.setText("线程没有结束 run 方法之前,不要再调用 start 方法");
                }
        }
        else if(e.getSource()==b_stop)
        {  thread.interrupt();
        }
    }
    public void run()
    {  while(true)
            {  text.append("\n"+new Date());
               try{  thread.sleep(1000);
                  }
               catch(InterruptedException ee)
                  {  text.setText("我被消灭");return;//结束 run 语句,消灭该线程.
                  }
            }
    }
    public static void main(String args[])
    {  Example19_9 tt=new Example19_9();
    }
}
```

注 在上面的例子 9 中,要格外注意的事情是 当一个线程没有进入死亡状态时,不要再给线程分配实体,由于线程只能引用最后分配的实体,先前的实体就会成为"垃圾",并且不会被垃圾收集机收集掉.

在下面的例子中我们用线程模拟运动员接力跑赛.在这个例子中有一个代表发令员的线程 发令员,该线程最先开始运行,不断地发出预备的命令 如图 19.5(a) ,我们通过一个按扭控制该线程的生命,即当点击该按扭时,发令员发出"跑"的命令,并结束自己的工作.然后,代表第一个运动员的线程开始运行,当跑到 100 米处时,结束生命,并通知代表第二个运动员的线程开始跑向终点(如图 19.5(b)和图 19.5 (c)).



例子 10 (效果如图 19.6)

```
import java.awt.*;import java.awt.event.*;import java.applet.*;
public class Example19_10 extends Applet implements Runnable,ActionListener
{  Button b=new Button("go");TextField text=null;
   Thread 发令员,运动员_1,运动员_2;
   int x=10;//线程运动的起始位置.
   Graphics mypen=null;
   public void init()
   {  b.addActionListener(this);text=new TextField(20);
      发令员=new Thread(this);运动员_1=new Thread(this);运动员_2=new Thread(this);
      add(b);add(text);
      mypen=getGraphics();
   }
   public void start()
   {  发令员.start();
   }
   public void actionPerformed(ActionEvent e)
   {  发令员.interrupt();//点击按扭结束发令员的生命.
   }
   public void run()
   {  if(Thread.currentThread()==发令员)
      { while(true)
        {  text.setText("准备跑... ...");text.setText(".....");
           try {  发令员.sleep(30);
               }
           catch(InterruptedException e)
               { //点击按扭结束生命,并让运动员_1开始跑.
                   text.setText("跑");
                   运动员_1.start(); break;
               }
        }
     }
    if(Thread.currentThread()==运动员_1)
     {  while(true)
```

```
            {   x=x+1;
                mypen.setColor(Color.blue);
                mypen.clearRect(10,80,99,100);//显示线程运动的动画.
                mypen.fillRect(x,85,5,5);
                try  {  运动员_1.sleep(10);
                     }
                catch(InterruptedException e)
                     {          //通知运动员_2 开始跑,运动员_1 结束生命
                         运动员_2.start(); return;
                      }
                if(x>=100)
                    {  运动员_1.interrupt();//运动员_1 当跑到 100 米处时结束生命.
                    }
            }
        }
        if(Thread.currentThread()==运动员_2)
        {  while(true)
            {   x=x+1;
                mypen.setColor(Color.red);
                mypen.clearRect(105,80,150,100);//显示线程运动的动画.
                mypen.fillRect(x+5,85,7,7);
                try  {  运动员_2.sleep(10);
                     }
                catch(InterruptedException e)
                     {text.setText("到达终点"); return;
                     }
                if(x>=200)              //运动员_2 跑到 200 米处时结束生命.
                    {  运动员_2.interrupt();
                    }
            }
        }
      }
}
```

19.8 用线程显示本地时间

为了显示本地时间,我们用三个线程来分别地显示时间的时,分和秒.

例子 11

```
import java.awt.*;import java.util.*;import java.awt.event.*;
import java.awt.geom.*;import java.applet.*;
public class Example19_11 extends Applet implements Runnable
{   Thread 时针=null,分针=null,秒针=null;//用来表示时针,分针和秒针的线程.
     //表示时针,分针,秒针端点的整型变量:
     int hour_a,hour_b,munite_a,munite_b,second_a,second_b;
     //用来获取当前时间的整型变量:
     int hour=0,munite=0,second=0;
     //用来绘制时针,分针和秒针的Grapghics对象:
     Graphics g_second=null,g_munite=null,g_hour=null;
     //用来存放表盘刻度的数组,供指针走动时使用:
     double point_x[]=new double[61], point_y[]=new double[61] ;
     //用来存放表盘刻度的数组,供绘制表盘使用:
     double scaled_x[]=new double[61], scaled_y[]=new double[61] ;
    //用来判断小程序是否重新开始的变量:
      int start_count=0;
   public void init()
   {g_hour=this.getGraphics();   g_hour.setColor(Color.cyan);
    g_second=this.getGraphics(); g_second.setColor(Color.red);
    g_munite=this.getGraphics(); g_munite.setColor(Color.blue);
    g_second.translate(200,200);//进行坐标系变换,将新坐标系原点设在(200,200)处.
    g_munite.translate(200,200);
    g_hour.translate(200,200);
    point_x[0]=0;point_y[0]=-120; //各个时针十二点处的位置坐标(按新坐标系的坐标).
    scaled_x[0]=0;scaled_y[0]=-140; //十二点处的刻度位置坐标(按新坐标系的坐标).
    double jiaodu=6*Math.PI/180;
     //表盘分割成60分,将分割点处的坐标存放在数组中
     for(int i=0;i<60;i++)
               { point_x[i+1]=point_x[i]*Math.cos(jiaodu)-
                          Math.sin(jiaodu)*point_y[i];
                 point_y[i+1]=point_y[i]*Math.cos(jiaodu)+
                                point_x[i]*Math.sin(jiaodu);
              }
      point_x[60]=0;point_y[60]=-120;//十二点各个时针的位置坐标(按新坐标系的坐标).
    //表盘分割成60分,将分割点处的坐标存放在数组中
     for(int i=0;i<60;i++)
            { scaled_x[i+1]=scaled_x[i]*Math.cos(jiaodu)-
                              Math.sin(jiaodu)*scaled_y[i];
              scaled_y[i+1]=scaled_y[i]*Math.cos(jiaodu)+
```

```
                                scaled_x[i]*Math.sin(jiaodu);
          }
  scaled_x[60]=0; scaled_y[60]=-140;//十二点处刻度位置坐标(按新坐标系的坐标).
}
 public void start()
 { //每当小程序重新开始时,首先消灭线程,然后重新开始创建线程.
   if(start_count>=1)
   {秒针.interrupt();分针.interrupt();时针.interrupt();
   }
   秒针=new Thread(this);分针=new Thread(this);
   时针=new Thread(this);
   秒针.start();分针.start();时针.start();
   start_count++;if(start_count>=2) start_count=1;
 }
 public void stop()
 {秒针.interrupt();分针.interrupt();时针.interrupt();
 }
 public void paint(Graphics g)
 {  //每当小程序重新绘制自己时,重新开始创建线程
        this.start();
       //绘制表盘的外观:
       g.drawOval(50,50,300,300);//表盘的外圈.
       g.translate(200,200);
      //绘制表盘上的小刻度和大刻度
      for(int i=0;i<60;i++)
         { if(i%5==0)
              { g.setColor(Color.red);
                g.fillOval((int) scaled_x[i],(int) scaled_y[i],8,8);
              }
           else
             g.fillOval((int) scaled_x[i],(int) scaled_y[i],3,3);
         }
}
public void run()
{ //获取本地时间:
  Date date=new Date();String s=date.toString();
  hour=Integer.parseInt(s.substring(11,13));
  munite=Integer.parseInt(s.substring(14,16));
  second=Integer.parseInt(s.substring(17,19));
```

```
if(Thread.currentThread()==秒针)
  {  second_a=(int)point_x[second];second_b=(int)point_y[second];
     g_second.drawLine(0,0,second_a,second_b); //秒针的初始位置.
     g_second.drawString("秒",second_a,second_b);
     int i=second;
     while(true)  //秒针开始行走,每一秒走6度.
     {try{秒针.sleep(1000);
      Color c=getBackground();g_second.setColor(c);
      g_second.drawLine(0,0,second_a,second_b);//用背景色清除前一秒时的秒针.
      g_second.drawString("秒",second_a,second_b);
              //如果这时秒针与分针重合,恢复分针的显示:
              if((second_a==munite_a)&&(second_b==munite_b))
               { g_munite.drawLine(0,0,munite_a,munite_b);
                 g_munite.drawString("分",munite_a,munite_b);
               }
              //如果这时秒针与时针重合,恢复时针的显示:
              if((second_a==hour_a)&&(second_b==hour_b))
               { g_hour.drawLine(0,0,hour_a,hour_b);
                 g_hour.drawString("时",hour_a,hour_b);
               }
             }
         catch(InterruptedException e)
             { Color c=getBackground();g_second.setColor(c);
              g_second.drawLine(0,0,second_a,second_b);//用背景色清除秒针.
              g_second.drawString("秒",second_a,second_b);
              return;
             }
         //秒针向前走一个单位:
 second_a=(int)point_x[(i+1)%60];
 second_b=(int)point_y[(i+1)%60];//每一秒走6度(一个单位格).
 g_second.setColor(Color.red);
 g_second.drawLine(0,0,second_a,second_b);    //绘制新的秒针.
 g_second.drawString("秒",second_a,second_b);
         i++;
       }
  }
if(Thread.currentThread()==分针)
  {
      munite_a=(int)point_x[munite];munite_b=(int)point_y[munite];
```

```
        g_munite.drawLine(0,0,munite_a,munite_b);//分针的初始位置.
        g_munite.drawString("分",munite_a,munite_b);
        int i=munite;
        while(true)
         {      //第一次,过 60-second 秒就前进一分钟,以后每过 60 秒前进一分钟.
                try{分针.sleep(1000*60-second*1000);second=0;
                 Color c=getBackground();g_munite.setColor(c);
                 //用背景色清除前一分钟的分针
                 g_munite.drawLine(0,0,munite_a,munite_b);
                  g_munite.drawString("分",munite_a,munite_b);
                 //如果这时分针与时针重合,恢复时针的显示:
                  if((hour_a==munite_a)&&(hour_b==munite_b))
                  { g_hour.drawLine(0,0,hour_a,hour_b);
                    g_hour.drawString("时",hour_a,hour_b);
                  }
                }
          catch(InterruptedException e)
                {return;
                }
               //分针向前走一个单位:
               munite_a=(int)point_x[(i+1)%60];
               munite_b=(int)point_y[(i+1)%60];//分针每分钟走 6 度(一个单位格).
               g_munite.setColor(Color.blue);
               g_munite.drawLine(0,0,munite_a,munite_b);//绘制新的分针.
               g_munite.drawString("分",munite_a,munite_b);
               i++;   second=0;
         }
    }
if(Thread.currentThread()==时针)
   { int h=hour%12;
     hour_a=(int)point_x[h*5+munite/12];
      hour_b=(int)point_y[h*5+munite/12];
     int i= h*5+munite/12;
     g_hour.drawLine(0,0,hour_a,hour_b);
     g_hour.drawString("时",hour_a,hour_b);
   while(true)
   {//第一次,过 12-munite%12 分钟就前进一个刻度,以后每过 12 分钟前进一个刻度.
      try{
        时针.sleep(1000*60*12-1000*60*(munite%12)-second*1000);munite=0;
```

```
            Color c=getBackground();g_hour.setColor(c);
            g_hour.drawLine(0,0,hour_a,hour_b);// 用背景色清除前 12 分钟时的时针.
            g_hour.drawString("时",hour_a,hour_b);
            }
          catch(InterruptedException e)
                 {return;
                 }
        hour_a=(int)point_x[(i+1)%60];
        hour_b=(int)point_y[(i+1)%60];//时针每 12 分走 6 度(一个单位格)
        g_hour.setColor(Color.cyan);
        g_hour.drawLine(0,0,hour_a,hour_b);//绘制新的时针.
        g_hour.drawString("时",hour_a,hour_b);
        i++; munite=0;
        }
      }
     }
}
```

习题十九

1　建立线程有几种方法

2.怎样设置线程的优先级

3　在多线程中,为什么要引入同步机制

4　在什么地方 wait()方法,notify()及 notifyAll()方法可以被使用

5　将例子 8 中的循环条件

```
while(五元钱的个数<3)
```

该写成

```
if(五元钱的个数<3)
```

是否合理.

5　编写一个小应用程序,在小应用程序的主线程中有两个线程,一个负责模仿垂直上抛运动,另一个模仿 45 度的抛体运动.

6. 模拟三个人排队买票,张某,李某和赵某买电影票,售票员只有三张五元的钱,电影票 5 元钱一张.张某拿二十元一张的新人民币排在李的前面买票,李某排在赵的前面拿一张 10 元的人民币买票,赵某拿一张 5 元的人民币买票.

7　在下列程序的主线程 main 中,又开始运行了几个线程

```
import java.awt.*;import java.awt.event.*;
class Gxy extends Thread implements Runnable
{  Frame f=new Frame("ok");
```

```
   TextField text1=new TextField(20),text2=new TextField(20),
             text3=new TextField(20);
   double n=0,正面=0,反面=0,正立=0;
   Gxy()
   {  f.setLayout(new FlowLayout());
      f.setSize(45,69); f.setVisible(true);
      f.add(text1);f.add(text2);f.add(text3);
      f.addWindowListener(new WindowAdapter()
                {  public void windowClosing(WindowEvent e)
                   {  System.exit(0);
                   }
                });
   }
   public void run()
   {  while(true)
       {  n++;
          double i=Math.random();
          if(i<0.5)       {正面++;text1.setText("正面出现的频率  "+正面/n);}
          else if(i==0.5) {正立++; text2.setText("正立出现的频率  "+正立/n);}
          else            {反面++;  text3.setText("反面出现的频率  "+反面/n);}
       }
   }
}
public class B
{  public static void main(String args[])
   {  Thread t=new Thread(new Gxy());
      t.start();
   }
}
```

第二十章　输入输出流

I/O 流提供一条通道程序,可以使用这条通道把源中的字节序列送给目的地.把输入流的指向称做源,程序从指向源的输入流中读取源中的数据.而输出流的指向是字节要去的一个目的地(或用户),程序通过向输出流中写入数据把信息传递到目的地.虽然 I/O 流经常与磁盘文件存取有关,但是程序的源和目的地也可以是键盘,鼠标,内存或显示器窗口.

Java 的 I/O 流库提供大量的流类(在包 java.io 中 .但是,所有输入流类都是 InputStream 字节输入流 抽象类或抽象类 Reader 字符输入流 的子类,而所有输出流都是 OutputStream(字节输出流)抽象类或抽象类 Writer(字符输出流)的子类.

把输入流的指向称做源,程序从指向源的输入流中读取源中的数据.而输出流的指向是数据要去的一个目的地,程序通过向输出流中写入数据把信息传递到目的地,如图 20.1,20.2 所示.

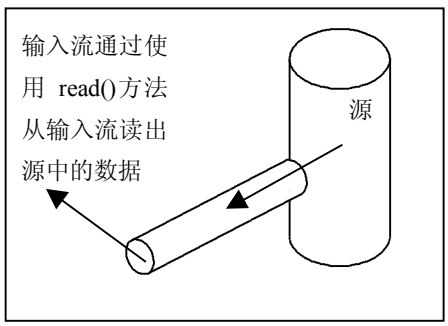


图 20.1　输入流示意图

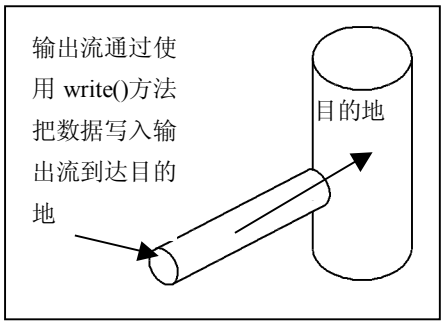


图 20.2　输出流示意图

20.1 File 类

File 类的对象主要用来获取文件本身的一些信息,例如文件所在的目录,文件的长度,文件读写权限等,不涉及对文件的读写操作.

创建一个 **File** 对象的构造方法有 **3** 个

```
File(String  filename);
File(String directoryPath,String filename);
File(File f, String filename);
```

其中,**filename** 是文件名字,**directoryPath** 是文件的路径,**f** 是指定成一个目录的文件.

使用 **File(String filename)**创建文件时,该文件被认为是与当前应用程序在同一目录中.

1　文件的属性

经常使用 **File** 类的下列方法获取文件本身的一些信息

1 public String getName() 获取文件的名字.
2 public boolean canRead() 判断文件是否是可读的.
3 public boolean canWrite() 判断文件是否可被写入.
4 public boolean exits() 判断文件是否存在.
5 public long length() 获取文件的长度 单位是字节 .
6 public String getAbsolutePath() 获取文件的绝对路径.
7 public String getParent() 获取文件的父目录.
8 public boolean isFile() 判断文件是否是一个正常文件,而不是目录.
9 public boolean isDirectroy() 判断文件是否是一个目录.
10 public boolean isHidden() 判断文件是否是隐藏文件.
11 public long lastModified() 获取文件最后修改的时间 时间是从 1970 年午夜至文件最后修改时刻的毫秒数 .

在下面的例子 1 中,我们使用上述的一些方法,获取某些文件的信息.

例子 1

```
import java.io.*;
class Example20_1
{  public static void main(String args[])
   {  File f1=new  File("F:\\8000","Example20_1.java");
      File f2=new File("F:\\8000");
      System.out.println("文件 Example20_1 是可读的吗:"+f1.canRead());
      System.out.println("文件 Example20_1 的长度:"+f1.length());
      System.out.println("文件 Example20_1 的绝对路径:"+f1.getAbsolutePath());
      System.out.println("F:\\8000:是目录吗  "+f2.isDirectory());
   }
}
```

2 目录

1 创建目录

File 对象调用方法 public boolean mkdir()创建一个目录,如果创建成功返回 true,否则返回 false(如果该目录已经存在将返回 false).

2 列出目录中的文件

如果 File 对象是一个目录,那么该对象可以调用下述方法列出该目录下的文件和子目录

public String[] list() 用字符串形式返回目录下的全部文件,
public File [] listFiles() 用 File 对象形式返回目录下的全部文件.

我们有时需要列出目录下指定类型的文件,比如.java,.txt 等扩展名的文件.可以使用 File

类的下述两个方法,列出指定类型的文件,

public String[] list(FilenameFilter obj) 该方法用字符串形式返回目录下的指定类型的所有文件.

public File [] listFiles(FilenameFilter obj) 该方法用 File 对象返回目录下的指定类型所有文件.

FilenameFile 是一个接口,该接口有一个方法

```
public boolean  accept(File dir,String name)
```

当向 list 方法传递一个实现该接口的对象时,dir 调用 list 方法在列出文件时,将调用 accept 方法检查该文件 name 是否符合 accept 方法指定的目录和文件名字要求.

在下面的例子 2 中,列出 f:\8000 目录下的部分.java 文件的名字.

例子 2

```
import java.io.*;
class FileAccept implements FilenameFilter
{  String str=null;
   FileAccept(String s)
   {  str="."+s;
   }
   public  boolean accept(File dir,String name)
   { return name.endsWith(str);
   }
}
public class Example20_2
{  public static void main(String args[])
   {  File dir=new File("F:/8000");
      FileAccept acceptCondition=new FileAccept("java");
      String fileName[]=dir.list(acceptCondition);
      for(int i=0;i<5;i++)
           {  System.out.println(fileName[i]);
           }
   }
}
```

20.2 FileInputStream 类

如果用户的文件读取需求比较简单,那么用户可以使用 FileInputStream 类,该类是从 InputStream 中派生出来的简单输入类.该类的所有方法都是从 InputStream 类继承来的.为了创建 FileInputStream 类的对象,用户可以调用它的构造器.下面显示了两个构造器

```
FileInputStream  String name
FileInputStream(File file)
```

第一个构造器使用给定的文件名 **name** 创建一个 **FileInputStream** 对象.第二个构造器使用 **File** 对象创建 **FileInputStream** 对象.

1 使用文件输入流读取文件

我们将要建立的许多程序都需要从文件中检索信息.文件输入流 输入流的子类 提供对文件的存取.为了读取文件,使用文件输入流构造器来打开一个到达该文件的输入流 源就是这个文件,输入流指向这个文件 ,文件输入流的格式如下所示

```
FileInputStream(String  name);
```

例如,为了读取一个名为 **myfile.dat** 的文件,建立一个文件输入流对象,如下所示

```
FileInputStream  istream = new  FileInputStream("myfile.dat");
```

文件输入流构造器的另一种格式是允许使用文件对象来指定要打开哪个文件,如下

```
FileInputStream  File file  ;
```

例如,下面这段代码使用文件输入流构造器来建立一个文件输入流,以检索文件,如下代码所示

```
File f = new File("myfile.dat");
FileInputStream  istream = new  FileInputStream(f);
```

2 处理 **I/O** 异常

当您使用文件输入流构造器建立通往文件的输入流时,可能会出现错误 也被称为异常 .例如,您试图要打开的文件可能不存在,就当出现 **I/O** 错误,**Java** 生成一个出错信号,它使用一个 **IOException** 对象来表示这个出错信号.程序必须使用一个 **try-catch** 块检测并处理这个异常.例如,为了把一个文件输入流对象与一个文件关联起来,使用类似于下面所示的代码

```
try  {  FileInputStream  ins = new  FileInputStream("myfile.dat"); //读取输入流
     }
catch (IOException e )
    {     //文件 I/O 错误
       System.out.println("File read error:  " +e );
    }
```

由于 **I/O** 操作对于错误特别敏感,所以许多其它的流类构造器和方法也抛出 **I/O** 异常.您必须按上述程序段所示的相同方式捕获处理这些异常.

3 从输入流中读取字节

输入流的唯一目的是提供通往数据的通道,程序可以通过这个通道读取数据,如图 20.1 所示.read 方法给程序提供一个从输入流中读取数据的基本方法.

read 方法的格式如下

```
int read();
```

read 方法从输入流中读取单个字节的数据.该方法返回字节值 0~255 之间的一个整数 ,如果该方法到达输入流的末尾,则返回-1.你可以这样直观的来理解从流中读取数据,一个输入流好比接在一个水罐上的水管流,如果把水比做字节源的数据,那么可以从水管一口一口的吸水,当没水时,我们就称达到了流的末尾.

read 方法还有其它一些形式.这些形式能使程序把多个字节读到一个字节数组中

```
int read(byte b[]);
int read(byte b[], int off, int len);
```

其中,off 参数指定 read 方法把数据存放在字节数组 b 中的什么地方样,len 参数指定该方法将读取的最大字节数.上面所示的这两个 read 方法都返回实际读取的字节个数,如果它们到达输入流的末尾,则返回-1.

FileInputStream 流顺序地读取文件,只要不关闭流,每次调用 read 方法就顺序地读取源其余的内容,直到流的末尾或流被关闭.

4 关闭流

虽然 Java 在程序结束时自动关闭所有打开的流,但是当我们使用完流后,显式地关闭任何打开的流仍是一个良好的习惯.一个被打开的流可能会用尽系统资源,这取决于平台和实现.如果没有关闭那些被打开的流,那么在这个或另一个程序试图打开另一个流时,这些资源可能得不到.关闭输出流的另一个原因是把该流缓冲区的内容冲洗掉 通常冲洗到磁盘文件上 .正如我们将要学到的,在操作系统把程序所写到输出流上的那些字节保存到磁盘上之前,有时被存放在内存缓冲区中,通过调用 close 方法,可以保证操作系统把流缓冲区的内容写到它的目的地.

下列程序是一个应用程序,该应用程序从磁盘上读取文件,将代码显示在屏幕上和文本区中.

例子 3

```
import java.io.*;import java.awt.*;import java.awt.event.*;
class Example20_3
{  public static void main(String args[])
   {  int b;
      TextArea text;
      Frame window=new Frame();
      byte tom[]=new byte[25];
      window.setSize(100,100);text=new TextArea(10,16);
```

```
        window.setVisible(true);window.add(text,BorderLayout.CENTER);
        window.pack();
        window.addWindowListener(new WindowAdapter()
                          {  public void windowClosing(WindowEvent e)
                            {  System.exit(0);
                            }
                          });

        try{  File f=new File("F:\\8000","Example20_1.java");
              FileInputStream readfile=new FileInputStream(f);
              while((b=readfile.read(tom,0,25))!=-1)
                  {
                     String s=new String (tom,0,b);
                     System.out.println(s);
                     text.append(s);
                  }
              readfile.close();
           }
        catch(IOException e)
           {  System.out.println("File read Error");
           }
     }
}
```

20.3 FileOutputStream 类

与 FileInputStream 类相对应的类是 FileOutputStream 类.FileOutputStream 提供了基本的文件写入能力.除了从 FileOutputStream 类继承来的方法以外,FileOutputStream 类还有三个构造器,这 2 个构造方法

```
FileOutputStream  String name
FileOutputStream(File file)
```

第一个构造器使用给定的文件名 name 创建一个 FileOutputStream 对象.第二个构造器使用 File 对象创建 FileOutputStream 对象.

可以使用 write 方法把字节发送给输出流,如图 20.2 所示.

write 的格式如下

```
public void write(byte b[])
```

其功能是写 b.length 个字节到输出流.

public void.write(byte b[],int off,int len)

其功能是从给定字节数组中起始于偏移量 off 处写 len 个字节到输出流,参数 b 是存放了数据的字节数组　off 是数据的起始偏移量　len 是要输出的字节数.

FileOutStream 流顺序地写文件,只要不关闭流,每次调用 writer 方法就顺序地向文件写入内容,直到流被关闭.

例子 4 是一个 Java 应用程序,该应用程序从键盘读取一行文本并将其存储到文件 line.txt 中.当用户运行该应用程序时,需要输入一行文本并按下 Enter 键.你可以使用记事本或其他文本编辑器查看该文件,这个文件 line.txt 被保存在和 Example20_3.class 的同一目录下.

例子 4

```
import java.io.*;
class Example20_4
{  public static void main(String args[])
   {  int b;
      byte buffer[]=new byte[100];
      try{  System.out.println("输入一行文本,并存入磁盘  ");
            b=System.in.read(buffer);   //把从键盘敲入的字符存入 buffer.
            FileOutputStream writefile=new FileOutputStream("line.txt");
            writefile.write(buffer,0,b);   // 通过流把 buffer 写入到文件 line.txt.
         }
      catch(IOException e)
         {  System.out.println("Error ");
         }
  }
}
```

20.4　FileReader 类和 FileWriter 类

与 FileInputStream 类和 FileOutputStream 类等价的读取器是 FileReader 类和 FileWriter 类,他们分别是 Reader 和 Writer 的子类,其构造方法分别是

```
FileReader  String  filename  ,
FileWriter (String  filename).
```

上面的 FileInputStream 使用字节读取文件,字节流不能直接操作 Unicode 字符,所以 Java 提供了字符流.由于汉字在文件中占用 2 个字节,如果使用字节流,读取不当会出现乱码现象,采用字符流就可以避免这个现象,因为,在 Unicode 字符中,一个汉字被看作一个字符.

下面是一段创建 FileReader 对象的代码,可以使用该对象读取名为 Student.txt 的文件.

```
FileReader in=new FileReader("Example20_1.java");
```

下面的例子 5 使用字符输入,输出流读写文件.将一段文字加密后存入文件,然后再读取.

例子 5 效果如图 20.3

```
import java.io.*;import java.awt.*;import java.awt.event.*;
class Example20_5
{  public static void main(String args[])
   {  char a[]="今晚 10 点发起总攻".toCharArray();
      int n=0,m=0;
      try{  File f=new File("F:\\8000","secret.txt");
            for(int i=0;i<a.length;i++)
               {  a[i]=(char)(a[i]^'R');
               }
            FileWriter out=new FileWriter(f);
            out.write(a,0,a.length);
            out.close();
            FileReader in=new FileReader(f);
            int length=(int)f.length();
            char tom[]=new char[length];
            while((n=in.read(tom,0,length))!=-1)
                {  String s=new String (tom,0,n);
                   m=n;
                   System.out.println("密文:"+s);
                }
            in.close();
            for(int i=0;i<m;i++)
               {  tom[i]=(char)(tom[i]^'R');
               }
            String 明文=new String(tom,0,m);
            System.out.println("明文:"+明文);
         }
      catch(IOException e)
         {  System.out.println("File read Error");
         }
   }
}
```

```
F:\8000>java Example20_5
密文:亘盱cb菽产败恩敉
明文:今晚10点发起总攻
```

图 20.3　用输入,输出流加密信息

假如 **Student.txt** 是一个学生名单,每个姓名占一行.如果我们想读取名字,那么每次必须读取一行,但 **FileReader** 类没有提供这种方法,所以我们必须把这个流　对象　再接到另外

一个流上,从后一个流中读取名单.Java 提供了名为 BufferedReader 的类,构造方法是

```
BufferedReader  Reader in
```

BufferedReader 流能够读取文本行,方法是 readLine().

通过向 BufferedReader 传递一个 Reader 对象 如 FileReader 的实例 ,来创建一个 BufferedReader 对象,如

```
BufferedReader in= BufferedReader(new FileReader("Student.txt"));
```

然后再从流 in 中读取 Student.txt.

类似地,可以将 BufferedWriter 流和 FileWriter 流连接在一起,然后使用 BufferedWriter 流将数据写到目的地,例如

```
FileWriter  tofile=new FileWriter("hello.txt");
BufferedWriter out= BufferedWriter(tofile);
```

然后使用 BufferedReader 类的方法

```
write(String s,int off,int len)
```

把字符串 s 写到 hello.txt 中,参数 off 是 s 开始处的偏移量,len 是写入的字符数量.

下面例子中使用了 BufferedReader 和 BufferedWriter,可以把文本区中的文本存入文件,也可以把读入的文件内容显示在文本区中.

例子 6

```
import  java.io.*;
import java.awt.*;
import java.awt.event.*;
class EWindow extends Frame implements ActionListener
{  TextArea text;
   Button buttonRead,buttonWrite;
   BufferedReader bufferIn;
   FileReader in;
   BufferedWriter bufferOut;
   FileWriter out;
   EWindow()
   {  super("流的读取");
      text=new TextArea(10,10);text.setBackground(Color.cyan);
      buttonRead =new Button("读取");
      buttonRead.addActionListener(this);
      buttonWrite =new Button("写出");
      buttonWrite.addActionListener(this);
      setLayout(new BorderLayout());
```

```
    setSize(340,340);
    setVisible(true);
    add(text,BorderLayout.CENTER);
    Panel pNorth=new Panel();
    pNorth.add(buttonRead);pNorth.add(buttonWrite);
    pNorth.validate();
    add(BorderLayout.NORTH,pNorth);
    addWindowListener(new WindowAdapter()
                    {  public void windowClosing(WindowEvent e)
                        {  System.exit(0);
                        }
                    });
}
public void actionPerformed(ActionEvent e)
{  String s;
    if(e.getSource()==buttonRead)
     {  try{  text.setText(null);
              File f=new File("F:\\8000\\","E.java");
              in=new FileReader(f);
              bufferIn=new BufferedReader(in);
              while((s=bufferIn.readLine())!=null)
               {  text.append(s+'\n');
               }
              bufferIn.close();
              in.close();
           }
        catch(IOException exp){System.out.println(exp);}
     }
    if(e.getSource()==buttonWrite)
     {  try {  File f=new File("F:\\8000\\","E.java");
               FileWriter out=new FileWriter(f);
               BufferedWriter bufferOut=new BufferedWriter(out);
               bufferOut.write(text.getText(),0,(text.getText()).length());
               bufferOut.flush();
               bufferOut.close();
               out.close();
            }
        catch(IOException exp){  System.out.println(exp);}
     }
```

```
    }
}
public class Example20_6
{  public static void main(String args[])
   {  EWindow w=new EWindow();
      w.validate();
   }
}
```

下面是一个英语单词填空小程序.在该程序中我们要用到第五章中我们学习过的字符串分析器　见 5.14　.一个文本文件 English.txt 作为要练习的英语句子表,如图 20.4 所示

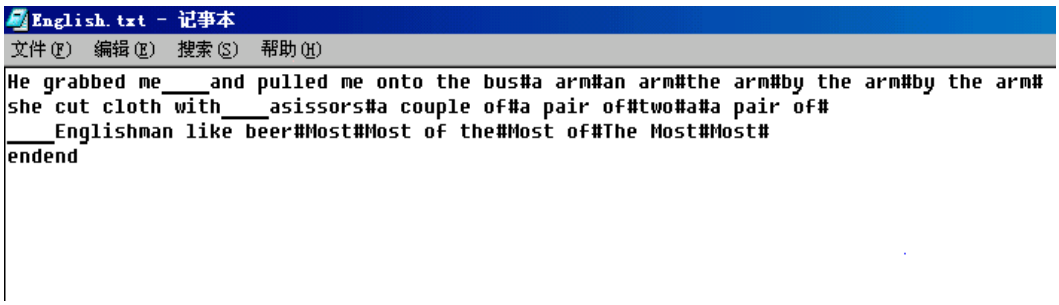


图 **20.4**　待读入的文件

图 20.5　句子填空程序

文本文件 English.txt 的每一行字符串的分隔符是符号"#".第一个语言符号是一个练习填空的英语句子,第 2 到第 5 个语言符号是供选择的答案,第 6 个是正确的答案.图 20.5 是例子 7 的效果图.在这个例子中,每次读取文本文件的一行,然后使用字符串解析器把语言符号分解出来.第一个语言符号放在文本框里,其余 4 个语言符号作为选择框的名字,供选择答案用.最后一个语言符号是答案,根据选择的情况来给出分数.每当作完一个题目之后　即选择框发生事件之后　,就判定分数.单击"下一题目"按钮,就再读取一行.当读取到最后一行字母 endend 时,就关闭流,通知用户练习已完成.当点击"重新练习"按钮时,程序重新将流指向文件 English.txt.

例子 7

```
import java.util.*;import java.io.*;
import java.awt.*;import java.awt.event.*;
class EWindow extends Frame implements ActionListener,ItemListener
{  String str[]=new String[7];String s;
   FileReader file;
   BufferedReader in;
   Button start,next;
   Checkbox box[];
   TextField 题目,分数;
   int score=0;
   CheckboxGroup age=new CheckboxGroup();
   EWindow()
   {  super("英语单词学习");
      分数=new TextField(10);题目=new TextField(70);
      start=new Button("重新练习");start.addActionListener(this);
      next=new Button("下一题目");next.addActionListener(this);
      box=new Checkbox[4];
      for(int i=0;i<=3;i++)
       {  box[i]=new Checkbox("",false,age);
          box[i].addItemListener(this);
       }
      try {  file=new FileReader("English.txt");
             in=new BufferedReader(file);
          }
      catch(IOException e){}
      setBounds(100,100,400,320); setVisible(true);
      setLayout(new GridLayout(4,1));
      setBackground(Color.pink);
      Panel p1=new Panel(),p2=new Panel(),
      p3=new Panel() ,p4=new Panel(),p5=new Panel();
      p1.add(new Label("题目:"));p1.add(题目);
      p2.add(new Label("选择答案  "));
      for(int i=0;i<=3;i++)
           {  p2.add(box[i]);
           }
      p3.add(new Label("您的得分:"));p3.add(分数);
      p4.add(start); p4.add(next);
```

```
   add(p1); add(p2);add(p3); add(p4);
   addWindowListener(new WindowAdapter()
                     {public void windowClosing(WindowEvent e)
                        {  System.exit(0);
                        }
                     });
  reading();
}
public void reading()
{   int i=0;
    try { s=in.readLine();
          if(!(s.startsWith("endend")))
              {  StringTokenizer tokenizer=new StringTokenizer(s,"#");
                 while(tokenizer.hasMoreTokens())
                     {  str[i]=tokenizer.nextToken();
                        i++;
                     }
                  题目.setText(str[0]);
                  for(int j=1;j<=4;j++)
                    {  box[j-1].setLabel(str[j]);
                    }
              }
           else if(s.startsWith("endend"))
               {  题目.setText("学习完毕");
                  for(int j=0;j<4;j++)
                    {  box[j].setLabel("end");
                       in.close();file.close();
                    }
               }
         }
      catch(Exception exp){ 题目.setText("无试题文件") ; }
}
public void actionPerformed(ActionEvent event)
{  if(event.getSource()==start)
      {  score=0;
         分数.setText("得分   "+score);
         try {  file=new FileReader("English.txt");
                in=new BufferedReader(file);
             }
```

```
            catch(IOException e){}
            reading();
         }
      if(event.getSource()==next)
         {  reading();
            for(int j=0;j<4;j++)
             {
               box[j].setEnabled(true);
             }
         }
   }
   public void itemStateChanged(ItemEvent e)
   {  for(int j=0;j<4;j++)
        { if(box[j].getLabel().equals(str[5])&&box[j].getState())
             {  score++;
                分数.setText("得分   "+score);
             }
          box[j].setEnabled(false);
        }
   }
}
public class Example20_7
{   public static void main(String args[])
    {  EWindow w=new EWindow();
       w.pack();
    }
}
```

20.5 使用文件对话框打开和保存文件

在十六章我们学过文件对话框,但那时没有真正实现对文件的读写操作.在学习了上述流之后,我们可以使用文件对话框方便地打开和保存文件,因为该对话框可以使用户很方便的选择文件所在的目录以及文件的名字.

在下面的例子 8 使用文件对话框打开和保存文件.

例子 8

```
import java.awt.*;import java.io.*;
import java.awt.event.*;
public class Example20_8
{  public static void main(String args[])
```

```
    {  FileWindows win=new FileWindows();
    }
}
class  FileWindows extends Frame implements ActionListener
{   FileDialog filedialog_save,filedialog_load;//声明2个文件对话筐
    MenuBar menubar;
    Menu menu;
    MenuItem itemOpen,itemSave;
    TextArea text;
    BufferedReader in;
    FileReader file_reader;
    BufferedWriter out;
    FileWriter tofile;
    FileWindows()
    {  super("带文件对话框的窗口");
       setSize(260,270);
       setVisible(true);
       menubar=new MenuBar();
       menu=new Menu("文件");
       itemOpen=new MenuItem("打开文件");
       itemSave=new MenuItem("保存文件");
       itemOpen.addActionListener(this);
       itemSave.addActionListener(this);
       menu.add(itemOpen);
       menu.add(itemSave);
       menubar.add(menu);
       setMenuBar(menubar);
       filedialog_save=new FileDialog(this,"保存文件话框",FileDialog.SAVE);
       filedialog_load=new FileDialog(this,"打开文件话框",FileDialog.LOAD);
       filedialog_save.addWindowListener(new WindowAdapter()
                                  {public void windowClosing(WindowEvent e)
                                       {  filedialog_save.setVisible(false);
                                       }
                                  });
     filedialog_load.addWindowListener(new WindowAdapter()//对话框增加适配器
                                  {public void windowClosing(WindowEvent e)
                                       {  filedialog_load.setVisible(false);
                                       }
                                  });
```

```
   addWindowListener(new WindowAdapter()
                                {public void windowClosing(WindowEvent e)
                                     {  System.exit(0);}
                                     });
  text=new TextArea(10,10);
  add(text,BorderLayout.CENTER);
}
public void actionPerformed(ActionEvent e)
{  if(e.getSource()==itemOpen)
     {  filedialog_load.setVisible(true);
        text.setText(null);
        String s;
        if(filedialog_load.getFile()!=null)
           {
              try{  File file= new
                    File(filedialog_load.getDirectory(),filedialog_load.getFile());
                    file_reader=new FileReader(file);
                    in=new BufferedReader(file_reader);
                    while((s=in.readLine())!=null)
                         text.append(s+'\n');
                    in.close();
                    file_reader.close();
                 }
             catch(IOException e2){}
           }
     }
   else if(e.getSource()==itemSave)
      {  filedialog_save.setVisible(true);
         if(filedialog_save.getFile()!=null)
           {
             try {
                    File file=new
                    File(filedialog_save.getDirectory(),filedialog_save.getFile());
                    tofile=new FileWriter(file);
                    out=new BufferedWriter(tofile);
                    out.write(text.getText(),0,(text.getText()).length());
                    out.flush();
                    out.close();
                    tofile.close();
```

```
                    }
                catch(IOException e2){}
            }
        }
    }
}
```

20.6 运行可执行文件

当要执行一个本地机上的可执行文件时,可以使用 **java.lang** 包中的 **Runtime** 类.首先使用 **Runtime** 类声明一个对象,如

```
Runtime  ec;
```

然后使用该类的静态 **getRuntime()**方法创建这个对象

```
ec=Runtime.getRuntime();
```

ec 可以调用 **exec(String command)**方法打开本地机的可执行文件或执行一个操作.

下面的例子 9 中,**Runtime** 对象执行"**java Example20_8**"操作,并打开 **windows** 平台上的记事本.

例子 9

```
import java.awt.*;import java.io.*;
import java.awt.event.*;
public class Example20_9
{  public static void main(String args[])
   {  try{
            Runtime ce=Runtime.getRuntime();
            ce.exec("java Example20_8");
            File file=new File("c:/windows","Notepad.exe");
            ce.exec(file.getAbsolutePath());
         }
      catch(Exception e){}
   }
}
```

20.7 RandomAccessFile 类

前面我们学习了用来处理文件的几个文件输入,文件输出流,而且通过一些例子,已经了解了那些流的功能.Java 还提供了专门用来处理文件输入输出操作的功能更完善的 RandomAccessFile 流.当用户真正需要严格地处理文件时,就可以使用 RandomAccessFile

类来创建一个对象,称做 RandomAccessFile 流.

RandomAccessFile 类创建的流与前面的输入,输出流不同,RandomAccessFile 类既不是输入流类 InputStream 类的子类,也不是输出流类 OutputStram 类的子类流.但是 RandomAccessFile 类创建的流的指向既可以作为源也可以作为目的地,换句话说,当我们想对一个文件进行读写操作时,我们可以创建一个指向该文件的 RandomAccessFile 流即可,这样我们既可以从这个流中读取文件的数据,也可以通过这个流写入数据到文件.

RandomAccessFile 类的两个构造方法.

1 RandomAccessFile(String name,String mode) 参数 name 用来确定一个文件名,给出创建的流的源,也是流目的地.参数 mode 取 r 只读 或 rw 可读写 ,决定创建的流对文件的访问权利.

2 RandomAccessFile(File file,String mode): 参数 file 是一个 File 对象,给出创建的流的源,也是流目的地.参数 mode 取 r 只读 或 rw 可读写 ,决定创建的流对文件的访问权利.

RandomAccessFile 类中有一个方法 seek(long a),用来移动 RandomAccessFile 流指向的文件的指针,其中参数 a 确定文件指针距离文件开头的字节位置.另外流还可以调用 getFilePointer()方法获取当前文件的指针的位置.RandomAccessFile 流对文件的读写比顺序读写更为灵活.

我们先来看一个简单的例子,在这个例子中我们把几个 int 型整数写入到一个名字为 tom.dat 文件中,然后按相反顺序读出这些数据.

例子 10

```
import java.io.*;
public class Example20_10
{   public static void main(String args[])
    {  RandomAccessFile in_and_out=null;
       int data[]={1,2,3,4,5,6,7,8,9,10};
       try{  in_and_out=new RandomAccessFile("tom.dat","rw");
          }
       catch(Exception e){}
       try{  for(int i=0;i<data.length;i++)
               {  in_and_out.writeInt(data[i]);
               }
             for(long i=data.length-1;i>=0;i—) //一个 int 型数据占 4 个字节,我们从
               {  in_and_out.seek(i*4); //文件的第 36 个字节读取最后面的一个整数,
                  //每隔 4 个字节往前读取一个整数
                  System.out.print(","+in_and_out.readInt());
               }
             in_and_out.close();
```

```
            }
        catch(IOException e){}
    }
}
```

表 20.1 RandomAccessFile 类的方法

方法	描述
close()	关闭文件
getFilePointer()	获取文件指针的位置
length()	获取文件的长度
read()	从文件中读取一个字节的数据
readBoolean()	从文件中读取一个布尔值,0 代表 false;其他值代表 true
readByte()	从文件中读取一个字节
readChar()	从文件中读取一个字符(2 个字节)
readDouble()	从文件中读取一个双精度浮点值 8 个字节
readFloat()	从文件中读取一个单精度浮点值 4 个字节
readFully(byte b[])	读 b.length 字节放入数组 b,完全填满该数组
readInt()	从文件中读取一个 int 值 4 个字节
readLine()	从文件中读取一个文本行
readlong()	从文件中读取一个长型值 8 个字节
readShort()	从文件中读取一个短型值 2 个字节
readUnsignedByte()	从文件中读取一个无符号字节 1 个字节
readUnsignedShort()	从文件中读取一个无符号短型值 2 个字节
readUTF()	从文件中读取一个 UTF 字符串
seek()	定位文件指针在文件中的位置
setLength(long newlength)	设置文件的长度
skipBytes(int n)	在文件中跳过给定数量的字节
write(byte b[])	写 b.length 个字节到文件
writeBoolean(boolean v)	把一个布尔值作为单字节值写入文件
writeByte(int v)	向文件写入一个字节
writeBytes(String s)	向文件写入一个字符串
writeChar(char c)	向文件写入一个字符
writeChars(String s)	向文件写入一个作为字符数据的字符串
writeDouble(double v)	向文件写入一个双精度浮点值
writeFloat(float v)	向文件写入一个单精度浮点值
writeInt(int v)	向文件写入一个 int 值
writeLong(long v)	向文件写入一个长型 int 值
writeShort(int v)	向文件写入一个短型 int 值
writeUTF(String s)	写入一个 UTF 字符串

下面的例子 11 显示了自己的源代码.

例子 11

```
import  java.io.*;
class Example20_11
{  public static void main(String args[])
   {  try{  RandomAccessFile in=new RandomAccessFile("Example20_11.java","rw");
            long filePoint=0;
```

```
            long fileLength=in.length();
            while(filePoint<fileLength)
                 {  String s=in.readLine();
                    System.out.println(s);
                    filePoint=in.getFilePointer();
                 }
            in.close();
          }
       catch(Exception e){}
   }
}
```

下面的例子 12 使用 RandomAccessFile 流实现一个通讯薄的录入与显示系统.

例子 12 效果如图 20.6

```
import java.io.*;
import javax.swing.*;
import java.awt.*;import
java.awt.event.*;
import javax.swing.border.*;
class InputArea extends Panel
implements ActionListener
{  File f=null;
   RandomAccessFile out;
   Box baseBox ,boxV1,boxV2;
   TextField name,email,phone;
   Button button;
   InputArea(File f)
   {   setBackground(Color.cyan);
       this.f=f;
       name=new TextField(12);
       email=new TextField(12);
       phone=new TextField(12);
       button=new Button("录入");
       button.addActionListener(this);
       boxV1=Box.createVerticalBox();
       boxV1.add(new Label("输入姓名"));
       boxV1.add(Box.createVerticalStrut(8));
       boxV1.add(new Label("输入 email"));
```

图 20.6　通讯录

```
        boxV1.add(Box.createVerticalStrut(8));
        boxV1.add(new Label("输入电话"));
        boxV1.add(Box.createVerticalStrut(8));
        boxV1.add(new Label("单击录入"));
        boxV2=Box.createVerticalBox();
        boxV2.add(name);
        boxV2.add(Box.createVerticalStrut(8));
        boxV2.add(email);
        boxV2.add(Box.createVerticalStrut(8));
        boxV2.add(phone);
        boxV2.add(Box.createVerticalStrut(8));
        boxV2.add(button);
        baseBox=Box.createHorizontalBox();
        baseBox.add(boxV1);
        baseBox.add(Box.createHorizontalStrut(10));
        baseBox.add(boxV2);
        add(baseBox);
    }
    public void actionPerformed(ActionEvent e)
    {  try{
            RandomAccessFile out=new RandomAccessFile(f,"rw");
            if(f.exists())
               {  long length=f.length();
                  out.seek(length);
               }
            out.writeUTF("姓名:"+name.getText());
            out.writeUTF("eamil:"+email.getText());
            out.writeUTF("电话:"+phone.getText());
            out.close();
          }
        catch(IOException ee){}
    }
}

public class Example20_12 extends Frame implements ActionListener
{  File file=null;
   MenuBar bar;
   Menu fileMenu;
   MenuItem 录入,显示;
```

```
TextArea show;
InputArea inputMessage;
CardLayout card=null; //卡片式布局.
Panel pCenter;
Example20_12()
{
    file=new File("通讯录.txt");
    录入=new MenuItem("录入");
    显示=new MenuItem("显示");
    bar=new MenuBar();
    fileMenu=new Menu("菜单选项");
    fileMenu.add(录入);
    fileMenu.add(显示);
    bar.add(fileMenu);
    setMenuBar(bar);
    录入.addActionListener(this);
    显示.addActionListener(this);
    inputMessage=new InputArea(file);
    show=new TextArea(12,20);
    card=new CardLayout();
    pCenter=new Panel();
    pCenter.setLayout(card);
    pCenter.add("录入",inputMessage);
    pCenter.add("显示",show);

    add(pCenter,BorderLayout.CENTER);
    addWindowListener(new WindowAdapter()
                { public void windowClosing(WindowEvent e)
                   {
                      System.exit(0);
                    }
                });
   setVisible(true);
   setBounds(100,50,420,380);
   validate();
}
public void actionPerformed(ActionEvent e)
{
  if(e.getSource()==录入)
```

```
        {
          card.show(pCenter,"录入");
        }
      else if(e.getSource()==显示)
        { int number=1;
          card.show(pCenter,"显示");
          try{  RandomAccessFile in=new RandomAccessFile(file,"r");
                String 姓名=null;
                while((姓名=in.readUTF())!=null)
                   {  show.append("\n"+number+" "+姓名);
                      show.append(in.readUTF());  //读取 email.
                      show.append(in.readUTF());  //读取 phone
                      show.append("\n------------------------ ");
                      number++;
                   }
                in.close();
              }
          catch(Exception ee){}
        }
    }
    public static void main(String args[])
    {  new Example20_12();
    }
}
```

20.8 数据流

1.DataInputStream 类和 DataOutputStream 类

DataInputStream 类和 DataOutputStream 类创建的对象被称为数据输入流和数据输出流.这两个流是很有用的两个流,它们允许程序按着机器无关的风格读取 Java 原始数据.也就是说,当我们读取一个数值时,不必再关心这个数值应当是多少个字节.

2.DataInputStream 类和 DataOutputStream 的构造方法

1 DataInputStream InputStream in 将创建的数据输入流指向一个由参数 in 指定的输入流,以便从后者读取数据 按着机器无关的风格读取 .

2 DataOutputStream OutnputStream out 将创建的数据输出流指向一个由参数 out 指定的输出流,然后通过这个数据输出流把 Java 数据类型的数据写到输出流 out.

表 20.2 DataInputStream 类及 DataOutputSteam 的部分方法

方法	描述
close()	关闭流
readBoolean()	读取一个布尔值
readByte()	读取一个字节
readChar()	读取一个字符
readDouble()	读取一个双精度浮点值
readFloat()	读取一个单精度浮点值
readInt()	从文件中读取一个 int 值
readlong()	读取一个长型值
readShort()	读取一个短型值
readUnsignedByte()	读取一个无符号字节
readUnsignedShort()	读取一个无符号短型值
readUTF()	读取一个 UTF 字符串
skipBytes(int n)	跳过给定数量的字节
writeBoolean(boolean v)	把一个布尔值作为单字节值写入
writeBytes(String s)	写入一个字符串
writeChars(String s)	写入字符串
writeDouble(double v)	写入一个双精度浮点值
writeFloat(float v)	写入一个单精度浮点值
writeInt(int v)	一个 int 值
writeLong(long v)	一个长型值
writeShort(int v)	一个短型值
writeUTF(String s)	写入一个 UTF 字符串

在下面的这个应用程序中,通过写几个 Java 类型的数据到一个文件,并再读出来.

例子 13 效果如图 **20.7**

```
import java.io.*;
public class Example20_13
{  public static void main(String args[])
   {  try
        {  FileOutputStream fos=new FileOutputStream("jerry.dat");
           DataOutputStream out_data=new DataOutputStream(fos);
           out_data.writeInt(100);out_data.writeInt(10012);
           out_data.writeLong(123456);
           out_data.writeFloat(3.1415926f); out_data.writeFloat(2.789f);
           out_data.writeDouble(987654321.1234);
           out_data.writeBoolean(true);out_data.writeBoolean(false);
           out_data.writeChars("i am ookk");
         }
       catch(IOException e){}
       try
         {  FileInputStream fis=new FileInputStream("jerry.dat");
            DataInputStream in_data=new DataInputStream(fis);
            System.out.println(":"+in_data.readInt());//读取第 1 个 int 整数.
```

```
            System.out.println(":"+in_data.readInt());//读取第 2 个 int 整数.
            System.out.println(":"+in_data.readLong()); //读取 long 整数 .
            System.out.println(":"+in_data.readFloat());//读取第 1 个 float 数.
            System.out.println(":"+in_data.readFloat());//读取第 2 个 float 数.
            System.out.println(":"+in_data.readDouble());
            System.out.println(":"+in_data.readBoolean());//读取第 1 个 boolean.
            System.out.println(":"+in_data.readBoolean());//读取第 2 个 boolean.
            char c;
            while((c=in_data.readChar())!='\0')      //'\0'表示空字符.
                  System.out.print(c);
          }
        catch(IOException e){}
    }
}
```

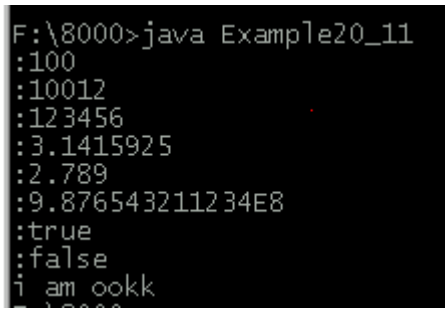


图 20.7　数据流读写数据

20.9 对象流

ObjectInputStream 类和 ObjectOutputStream 类分别是 InputStream 类和 OutputStream 类的子类.ObjectInputStream 类和 ObjectOutputStream 类创建的对象被称为对象输入流和对象输出流.对象输出流使用 writeObject(Object obj)方法将一个对象 obj 写入到一个文件,对象输入流使用 readObject()读取一个对象到程序中.

ObjectInputStream 类和 ObjectOutputStream 类的构造方法分别是

```
ObjectInputStream(InputStream  in),
ObjectOutputStream(OutputStream  out).
```

ObjectOutputStream 的指向应当是一个输出流对象,因此当准备将一个对象写入到文件时,首先用 FileOutputStream 创建一个文件输出流,如下列代码所示

```
FileOutputStream file_out=new FileOutputStream("tom.txt");
ObjectOutputStream object_out=new  ObjectOutputStream(file_out);
```

同样 ObjectInputStream 的指向应当是一个输入流对象,因此当准备从文件中读入一个对象到程序中时,首先用 FileInputStream 创建一个文件输入流,如下列代码所示

```
FileInputStream file_in=new FileInputStream("tom.txt");
ObjectInputStream object_in=new  ObjectInputStream(file_in);
```

在下面的例子中,我们在程序中的文本区输入若干个字符后,点击"写出对象"按扭将这个文本区对象写入到一个文件:tom.txt.然后点击"读入对象"按扭再将这个文本区对象读如到程序中,并改变该文本区对象的背景颜色,再添加该文本区对象到窗口,如图 20.8.

图 20.8 对象的读入与写出

例子 14(效果如图 20.9)

```
import java.awt.*;import java.awt.event.*;
import java.io.*;
public class Example20_14 extends Frame implements ActionListener
{  TextArea text=null; Button 读入=null,写出=null;
   FileInputStream file_in=null; FileOutputStream file_out=null;
   ObjectInputStream object_in=null;                    //对象输入流.
   ObjectOutputStream object_out=null;                  //对象输出流.
   Example20_14()
   {  setLayout(new FlowLayout()); text=new TextArea(6,10);
      读入=new Button("读入对象");  写出=new Button("写出对象");
      读入.addActionListener(this);写出.addActionListener(this);
      setVisible(true); add(text);add(读入);add(写出);
      addWindowListener(new WindowAdapter()
                  {  public void windowClosing(WindowEvent e)
                           {  System.exit(0);
                           }
                  });
       pack();setSize(300,300);
   }
   public void actionPerformed(ActionEvent e)
   {  if(e.getSource()==写出)
         {  try{  file_out=new FileOutputStream("tom.txt");
                  object_out=new ObjectOutputStream(file_out);//创建对象输出流.
```

```
                object_out.writeObject(text);                    //写对象到文件中.
                object_out.close();
              }
            catch(IOException event){}
          }
        else if(e.getSource()==读入)
          {  try{  file_in=new FileInputStream("tom.txt");
                   object_in=new ObjectInputStream(file_in);     //创建对象输入流.
                   TextArea temp=(TextArea)object_in.readObject();//从文件中读入对象.
                   temp.setBackground(Color.pink); this.add(temp);//添加该对象到窗口.
                   this.pack();this.setSize(600,600);
                   object_in.close();
                 }
             catch(ClassNotFoundException event)
               {  System.out.println("不能读出对象");
               }
             catch(IOException event)
               {  System.out.println("can not read file");
               }
          }
    }
    public static void main(String args[])
    {  Example20_14 win=new Example20_14();
    }
}
```

注 当我们使用对象流写入或读入对象时,要保证对象是序列化的.这是为了保证能把对象写入到文件,并能再把对象正确读回到程序中的缘故.Java 提供给我们的绝大多数对象都是所谓序列化的,比如组件等.一个类如果实现了 **Serializable** 接口,那么这个类创建的对象就是所谓序列化的对象.**Serializable** 接口中没有方法,因此实现该接口的类不需要实现额外的方法.一个序列化类的子类创建的对象也是序列化的.

在下面的例子 15 中有一个实现接口 Serializable 的 Student 类.

例子 15

```
import java.io.*;
class Student implements Serializable//实现接口 Serializable 的 Student 类.
{  String name=null;double height;
   Student(String name,double height)
   {  this.name=name;this.height=height;
```

```
    }
    public void setHeight (double c)
    {  this.height=c;
    }
}
public class Example20_15
{  public static void main(String args[])
    {  Student zhang=new Student("zhang ping",1.65);
       try{  FileOutputStream    file_out=new FileOutputStream("s.txt");
             ObjectOutputStream object_out=new ObjectOutputStream(file_out);
             object_out.writeObject(zhang);
             System.out.println(zhang.name+"的身高是  "+zhang.height);
             FileInputStream file_in=new FileInputStream("s.txt");
             ObjectInputStream object_in=new ObjectInputStream(file_in);
             zhang=(Student)object_in.readObject();
             zhang.setHeight(1.78); //修改身高.
             System.out.println(zhang.name+"现在的身高是  "+zhang.height);
           }
        catch(ClassNotFoundException event)
           {  System.out.println("不能读出对象");
           }
        catch(IOException event)
           {  System.out.println("can not read file"+event);
           }
    }
}
```

20.10 Process 类中的流

Process 是 java.lang 包中的一个类,可以使用该包中的 Runtime 类调用其静态方法 exec 得到 Process 的一个实例,exec 方法可以运行一个可执行文件,即启动一个进程,exec 方法返回一个 Process 对象 见 20.6 .一个 Process 对象可以使用方法 getErrorStream()获取该进程错误信息的输入流 使用方法 getInputStream()获取该进程的输入流.方法 getOutputStream()获取该进程的输出流.利用 Process 中的流,我们可以获取该进程的某些信息.

下面的例子 16 是一个用于编写 java 程序的小软件,可以用来编译,运行 Java 应用程序 如图 20.9 .

例子 16　效果如图 **20.9**

```
import java.awt.*;import java.io.*;import java.awt.event.*;
public class Example20_16
{  public static void main(String args[])
   {  JDK f=new JDK();
      f.pack();
      f.addWindowListener(new WindowAdapter()  //窗口增加适配器.
                  {public void windowClosing(WindowEvent e)
                        {  System.exit(0);
                        }
                   });
      f.setBounds(100,120,700,360);
      f.setVisible(true);
   }
}
class  JDK extends Frame implements ActionListener,Runnable
{  Thread compiler=null;  //负责编译的线程.
   Thread run_prom=null; //负责运行程序的线程.
   boolean bn=true;
   CardLayout mycard;
   Panel p=new Panel();
   File file_saved=null;
   TextArea input_text=new TextArea(),//程序输入区.
   compiler_text=new TextArea(),     //编译出错显示区.
```

```
dos_out_text=new TextArea();       //程序运行时,负责显示在 dos 窗口的输出信息.
Button button_input_text,button_compiler_text,
button_compiler,button_run_prom,button_see_doswin;
TextField input_flie_name_text=new TextField("输入被编译的文件名字.java");
TextField run_file_name_text=new TextField("输入应用程序主类的名字");
JDK()
{  super("Java 编程小软件");
   mycard=new CardLayout();
   compiler=new Thread(this);
   run_prom=new Thread(this);
   button_input_text=new Button("程序输入区(白色)");
   button_compiler_text=new Button("编译结果区(粉色)");
   button_compiler=new Button("编译程序");
   button_run_prom=new Button("运行应用程序");
   button_see_doswin=new  Button("查看应用程序运行时在 dos 窗口输出的信息");
   p.setLayout(mycard);
   p.add("input",input_text);p.add("compiler",compiler_text);
   p.add("dos",dos_out_text);
   add(p,"Center");
   add( button_see_doswin,"South");
   compiler_text.setBackground(Color.pink);
   dos_out_text.setBackground(Color.blue);
   Panel p1=new Panel();p1.setLayout(new GridLayout(4,2));
   p1.add(new Label("按扭输入源程序  "));p1.add(button_input_text);
   p1.add(new Label("按扭看编译结果  "));p1.add(button_compiler_text);
   p1.add(input_flie_name_text);  p1.add(button_compiler);
   p1.add(run_file_name_text); p1.add(button_run_prom);
   add(p1,BorderLayout.NORTH);
   button_input_text.addActionListener(this);
   button_compiler_text.addActionListener(this);
   button_compiler.addActionListener(this);
   button_run_prom.addActionListener(this);
   button_see_doswin.addActionListener(this);
}
public void actionPerformed(ActionEvent e)
{  if(e.getSource()==button_input_text)
     {  mycard.show(p,"input");
     }
   else if(e.getSource()==button_compiler_text)
```

```
    {  mycard.show(p,"compiler");
    }
   else if(e.getSource()==button_see_doswin)
    {  mycard.show(p,"dos");
    }
   else if(e.getSource()==button_compiler)
   {   if(!(compiler.isAlive()))
          {  compiler=new Thread(this);
          }
       try{  compiler.start();
          }
       catch(Exception eee){}
       mycard.show(p,"compiler");
   }
  else if(e.getSource()==button_run_prom)
  {  if(!(run_prom.isAlive()))
          {  run_prom =new Thread(this);
          }
       try{  run_prom.start();
          }
       catch(Exception eee){}
     mycard.show(p,"dos");
  }
}
public void run()
{  if(Thread.currentThread()==compiler)
     {  compiler_text.setText(null);
        String temp=input_text.getText().trim();
        byte buffer[]=temp.getBytes();
        int b=buffer.length;
        String flie_name=input_flie_name_text.getText().trim();
        try{  file_saved=new File(flie_name);
              FileOutputStream writefile=new FileOutputStream(file_saved);
              writefile.write(buffer,0,b);writefile.close();
         }
        catch(IOException e5)
          {  System.out.println("Error ");
          }
       try{  Runtime ce=Runtime.getRuntime();
```

```
                InputStream in=ce.exec("javac "+flie_name).getErrorStream();
                BufferedInputStream bin=new BufferedInputStream(in);
                byte shuzu[]=new byte[100];
                int n;boolean bn=true;
                while((n=bin.read(shuzu,0,100))!=-1)
                   {  String s=null;
                      s=new String(shuzu,0,n);
                      compiler_text.append(s);
                      if(s!=null) bn=false;
                   }
                if(bn)
                    {  compiler_text.append("编译正确");
                    }
              }
            catch(IOException e1){}
         }
        else if(Thread.currentThread()==run_prom)
         {   dos_out_text.setText(null);
            try{  Runtime ce=Runtime.getRuntime();
                  String path=run_file_name_text.getText().trim();
                  InputStream in=ce.exec("java  "+path).getInputStream();
                  BufferedInputStream bin=new BufferedInputStream(in);
                  byte zu[]=new byte[150];
                  int m;String s=null;
                  while((m=bin.read(zu,0,150))!=-1)
                     {  s=new String(zu,0,m);
                        dos_out_text.append(s);
                     }
                }
             catch(IOException e1){}
          }
      }
  }
```

习题二十

1.了解打印流.我们已经学习了数据流,其特点是用 Java 的数据类型读写文件,但使用数据

流写成的文件用其它文件阅读器无法进行阅读　看上去是乱码　.PrintStream 类提供了一个过滤输出流,该输出流能以文本格式显示 Java 的数据类型.上机实习下列程序

```
import java.awt.*;import java.io.*;
public class E1
{  public static void main(String args[])
   {  double x=0.987;boolean b=true;  long g=12345; Button button=new Button("ok");
     try
        {  PrintStream ps=new PrintStream(new FileOutputStream("p.txt"));
           ps.print(12345); ps.println("how are you"); ps.println(234); ps.println(x);
           ps.println(button); ps.close();
        }
     catch(IOException e){}
   }
}
```

2.了解字节输入,输出流.我们已经知道,流的源和目标除了可以是文件外,还可以是计算机内存.字节输入流　ByteArrayInputStream 和字节输出流　ByteArrayOutputStream 分别使用字节数组作为流的源和目标.上机练习下列程序

```
import java.io.*;
public class E2
{  public static void main(String args[])
   {  byte b[]={12,56,30,48,4,8,-9};
      try { int i=0;
            ByteArrayInputStream bais=new ByteArrayInputStream(b);
            DataInputStream data=new DataInputStream(bais);
            while(i<b.length)
             {  System.out.println(data.readByte());i++;
            }
          }
      catch(IOException e){}
   }
}
```

3　参照例子 7 编写一个标准化考试小软件,要求每做完一个题目都把该题目的正确答案显示给用户.

4　参照本章例子 12 编写一个货物管理小软件.

6　实习下列程序,注意 File 类中 delete 方法的使用.

```
import java.io.*;
public class File_Del_Test
{  public static void main(String args[])
```

```
    {  File f=null;boolean b=false;
       f=new File("Tom.doc");
       b=f.delete();
       if(b)
        {  System.out.print("文件被删除了");
        }
       else
        {  System.out.print("文件未能删除");
        }
    }
}
```

7　将例子 16 增加运行 Java Applet 的功能.

第二十一章 Java 网络的基本知识

很少有人在接触过 Internet 后,能拒绝它的诱惑,大量和多样的信息太吸引人了.能与其他人交流和共享信息,其重要性已是无可争议的了.本章重点介绍了 4 个重要的类 URL,Socket,InetAddress,DatagramSocket,揭示了它们在网络编程中的重要作用.

21.1 使用 URL

1. Internet 寻址

因为 **Internet** 上的每台计算机必须能够唯一地标识出来,因此标准化的第一个部分就是 IP 地址.IP 地址是用于唯一标识连接到 Iternet 的计算机的数字地址.理解 IP 地址的最简单的方式就是把它与您家的邮政地址做比较.想想就能明白,邮政地址与 IP 地址的工作方式极其相似,它们唯一地标识了您住址.一个邮政地址保证仅标识一个地理位置.同样一个 IP 地址唯一地标识了 Internet 上的一台计算机.顺便说一下,IP 地址中的"IP"代表 **Internet Protocol(Internet** 协议).IP 地址实际上是由 32 位二进制数组成,如 202.199.28.6.您也许对符号形式的 IP 地址更为熟悉,如 **dlrin.edu.cn,cctv.com.** 请记住,没 IP 地址就不能区分连在 **Internet** 上不同的计算机.

2 使用 URL 定位资源

如果 IP 地址唯一标识了 **Internet** 上的计算机,则 URL 标识了这些计算机上的资源.更具体地说,**URL(uniform resource locators** 统一资源定位符)充当一个指针,指向 Web 上的 Web 页,二进制文件以及其他的信息对象.当您手工输入例如 **http://www.dlrin.edu.cn/hotlink.html** 的网址时,实际上就提供了该站点主页的 URL.

可能您没有注意到,IP 地址在 URL 中扮演了重要的角色.因为 Web 页最终是驻留在连接到 Internet 的某台计算机上,所以当引用该页面,就需要确定计算机.因此,URL 自然包含给定资源的计算机的 IP 地址,包含资源的计算机就是资源的主机.我们分析一下面的 **http://www.dlrin.edu.cn/hotlink.html**,看看它到底包含了哪些信息

http 服务使用的协议 **HTTP** ,

dlrin.edu.cn 存储资源的计算机的域名地址,

hotlink.html 资源.

3 客户与服务器

在学习 Java 的网络知识之前,还要介绍一下网络中客户/服务器体系结构.您肯定听说过这一名词,但您可能并未完全理解它们在网络中的重要性.Web 的客户/服务器体系结构的含义就是,客户需要某些类型的信息,而服务器提供客户所需要的信息.客户需要连接到服务器上,并向服务器请求信息 而服务器则向客户发送信息,两者协同工作,各得其所.

4 一个有趣的例子

在下面的学习中,您将高兴地看到,Java对网络的支持将与您前面学过的概念密切相关.下面是一些由Java的网络API所提供的基本网络类,它们都包含在java.net包中.其中的URL类提供了许多构造方法,您可以利用它们创建 URL.其中最简单的构造方法只要求您提供表示URL资源的字符串

```
public URL(String s)
```

下面的例子利用这种构造方法创建一个URL对象url

```
try {
      url=new URL("http//www.dlrin.edu.cn");
    }
catch(MalformedURLException e)
   {
      System.out.println ("Bad URL:"+url);
   }
```

由于这个 URL 构造方法在运行出错时会抛出异常,所以您必须在 try-catch 结构中创建URL对象.当然,正如上面代码所示的那样,做起来并不麻烦.

成功创建了 URL 对象后,您就可以努力练习其他有趣的 Java 网络功能了.例如,您可以在applet中链向另外的Web页面,下面的代码就能实现这个功能

```
getAppletContext().showDocument(url);
```

在小程序中可以放心的使用这个方法,这是因为 **getAppletContext()**方法是在 **Applet**类中定义的.**showDocument()**实际完成定位到另一个 Web 页面的工作,程序只需提供 URL,其他的工作将自动完成.

下面的例子1在一个文本框中输入网址,然后点击确定按钮链接到指定的页面.

例子 1

```
import java.applet.*;import java.awt.*;
import java.awt.event.*;import java.net.*;
public class Example21_1 extends Applet implements ActionListener
{  Button button;
   URL url;
   TextField text;
   public void init()
   {  text=new TextField(18);
      button=new Button("确定");
      add(new Label("输入网址:"));add(text); add(button);
     button.addActionListener(this);
```

```
    }
    public void actionPerformed(ActionEvent e)
    {  if(e.getSource()==button)
        {  try {  url=new URL(text.getText().trim());
                  getAppletContext().showDocument(url);
            }
           catch(MalformedURLException g)
            {  text.setText("不正确的URL:"+url);
            }
        }
    }
}
```

21.2 套接字

1.套接字 Socket

IP 地址标识 Internet 上的计算机,端口号标识正在计算机上运行的进程 程序 .端口号与 IP 地址的组合得出一个网络套接字.端口号被规定为一个 16 位的整数 0~65535.其中,0~1023 被预先定义的服务通信占用 如 telnet 占用端口 23,http 占用端口 80 等 .除非我们需要访问这些特定服务,否则,就应该使用 1024~65535 这些端口中的某一个进行通信,以免发生端口冲突.当两个程序需要通信时,它们可以通过使用 Socket 类建立套接字连接.你可以把套接字连接想象为一个电话呼叫,当呼叫完成后,谈话的任何一方都可以随时讲话.但是在最初建立呼叫时,必须有一方呼叫,而另一方则监听铃声.这样,呼叫的一方为"客户",负责监听的一方是"服务器".

2 客户建立连接到服务器的套接字对象

客户端的程序使用 Socket 类建立到服务器的套接字连接.
Socket 的构造方法是

```
Socket(String  host,int port)
```

参数 host 是服务器的 IP 地址,port 是一个端口号.
当建立时可能发生 IOException 异常,因此应象下面那样建立到服务器的套接字连接

```
try{  Socket mysocket=new Socket("http://192.168.0.78",1880);
   }
catch(IOException e){}
```

当套接字连接 mysocket 建立后,你可以想象一条通信"线路"已经建立起来.mysocket 可以使用方法 getInputStream()获得一个输入流,然后用这个输入流读取服务器放入"线路"的

信息　但不能读取自己放入”线路”的信息,就象打电话时,我们只能听到对方放入线路里的声音一样　.mysocket 还可以使用方法 getOutputStream()获得一个输出流,然后用这个输出流将信息写入”线路”.

在实际编写程序时,把 mysocket 使用方法 getInputStream()获得的输入流接到另一个数据流上　回忆我们在文件输入输出流所进行的连接,道理是一样的　,然后就可以从这个数据流读取服务器来的信息,之所以这样做是因为后面 DataInputStream 流有更好的从流中读取信息的方法.同样把 mysocket 使用方法 getOutputStream()得到的输出流接到另一个 DataOutputStream 数据流上,然后向这个数据流写入信息,发送给服务器端,之所以这样做是因为后面的 DataOutputStream 流有更好的向流中写入信息的方法.

3　ServerSocket 类

我们已经知道客户负责建立客户到服务器的套接字连接,即客户负责呼叫.因此服务器必须建立一个等待接收客户的套接字的 ServerSocket 对象.ServerSocket 的构造方法是

```
ServerSocket(int port)
```

port 是一个端口号.port 必须和客户呼叫的端口号相同.

当建立服务器套接字时可能发生 IOException 异常,因此应象下面那样建立接收客户的服务器套接字.

```
try{  ServerSocket  server_socket=new ServerSocket(1880);
   }
catch(IOException e){}
```

当服务器的 ServerSocket 对象 server_socket 建立后,就可以使用方法 accept()接收客户的套接字连接呼叫,代码如下所示

```
server_socket.accept();
```

接收客户的套接字也可能发生 IOException 异常,
因此应象下面那样建立接受客户的套接字.

```
try{  Socket  sc=server_socket.accept();
   }
catch(IOException e){}
```

所谓”接收”客户的套接字连接就是 accept()会返回一个和客户端 Socket 对象相连接的 Socket 对象,服务器端的这个 Socket 对象 sc 使用方法 getOutputStream()获得的输出流将指向客户端 Socket 对象 mysocket 使用方法 getInputStream()获得的那个输入流　同样,服务器端的这个 Socket 对象 sc 使用方法 getInputStream()获得的输入流将指向客户端 Socket 对象 mysocket 使用方法 getOutputStream()获得的那个输出流.因此,当服务器向这个输出流写入信息时,客户端通过相应的输入流就能读取,反之亦然.如图 21.1 所示意.需要注意的是,从套接字连接中读取数据与从文件中读取数据有着很大的不同,尽管二者都是输入流.从文件中读取数据时,所有的数据都已经在文件中了.而使用套接字连接时,可能在另一端数据发送出来之前,就已经开始试着读取了,这时,就会堵塞本线程,直到该读取方法成功读取到信息,本

线程才继续执行后续的操作.

另外,需要注意的是 accept 方法也会堵塞线程的继续执行,直到接收到客户的呼叫.也就是说,如果没有客户呼叫服务器,那么下述代码中的"System.out.println("ok") "总也不会被执行

```
try{  Socket  sc=server_socket.accept();
      System.out.println("ok")
   }
catch(IOException e){}
```

连接建立后,服务器端的套接字对象调用 getInetAddress()方法可以获取一个 InetAddess 对象,该对象含有客户端的 IP 地址和域名,同样,客户端的套接字对象调用 getInetAddress()方法可以获取一个 InetAddess 对象,该对象含有服务器端的 IP 地址和域名,有关 InetAddess 类的讲解见后面的 21.3.

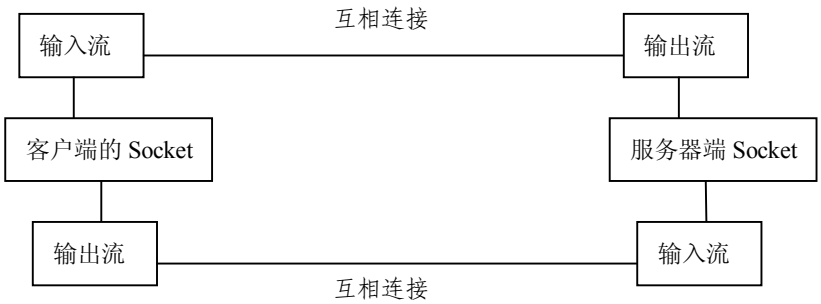


图 20.1　套接字连接示意图

双方通信完毕后,应友好地关闭套接字连接

```
sc.close();
```

下面我们通过一个简单的例子说明上面讲的概念.在例子 2 中,客户端向服务器发出信息 "你好",然后每隔 500 毫秒,客户端向服务器发送一个随机数.服务器将回答 "你好,我是服务器",并将客户发来的数据返回给客户.你首先将例子 2 中服务器端的 Server.java 编译通过,并运行起来,等待客户的呼叫.然后运行客户端程序.

例子 2

1　客户端程序

```
import java.io.*;
import java.net.*;
public class Client
{  public static void main(String args[])
   {  String s=null;
      Socket mysocket;
```

```
        DataInputStream in=null;
        DataOutputStream out=null;
        try{
            mysocket=new Socket("localhost",4331);
            in=new DataInputStream(mysocket.getInputStream());
            out=new DataOutputStream(mysocket.getOutputStream());
            out.writeUTF("你好  ");//通过 out 向"线路"写入信息.
            while(true)
              {
                 s=in.readUTF();//通过使用 in 读取服务器放入"线路"里的信息.堵塞状态,
                                //除非读取到信息.
                 out.writeUTF(":"+Math.random());
                 System.out.println("客户收到:"+s);
                 Thread.sleep(500);
              }
           }
         catch(IOException e)
           {  System.out.println("无法连接");
           }
         catch(InterruptedException e){}
    }
}
```

2 服务器端程序

```
import java.io.*;import java.net.*;
public class Server
{ public static void main(String args[])
  {  ServerSocket server=null;
     Socket you=null;String s=null;
     DataOutputStream out=null;DataInputStream  in=null;
     try{ server=new ServerSocket(4331);}
      catch(IOException e1){System.out.println("ERRO:"+e1);}
     try{  you=server.accept();
           in=new DataInputStream(you.getInputStream());
           out=new DataOutputStream(you.getOutputStream());
           while(true)
           {
             s=in.readUTF();// 通过使用 in 读取客户放入"线路"里的信息.堵塞状态,
                            //除非读取到信息.
```

```
                out.writeUTF("你好:我是服务器");//通过 out 向"线路"写入信息.
                out.writeUTF("你说的数是:"+s);
                System.out.println("服务器收到:"+s);
                Thread.sleep(500);
              }
            }
        catch(IOException e)
            {  System.out.println(""+e);
            }
          catch(InterruptedException e){}
      }
  }
```

4　把套接字连接放在一个线程中

套接字连接中涉及到输入流和输出流操作,为了不影响我们做其它的事情,我们应把套接字连接放在一个单独的线程中去进行.另外,服务器端收到一个客户的套接字后,就应该启动一个专门为该客户服务的线程.

我们用我们学过的组件,设计一个略微复杂的套接字连接.

在下面的例子 6 中,客户输入三角形三边的长度并发送给服务器,服务器把计算出的三角形面积返回给客户.因此你可以将计算量大的工作放在服务器端,客户负责计算量小的工作,实现客户-服务器交互计算,来完成某项任务.你首先将例子 3 中服务器端的程序编译通过,并运行起来,等待客户的呼叫.客户端通过使用浏览器访问服务器上一个含有 **Java Applet** 的小应用程序.

虽然小应用程序驻留在服务器端,但它需要下载到客户端的浏览器来运行,因此小应用程序还是称做客户端程序,小应用程序和应用程序的一个不同之处是　小应用程序只能和它所驻留的服务器建立套接字连接.

例子 3

(1) 客户端　效果如图 21.2

```
import java.net.*;import java.io.*;
import java.awt.*;import java.awt.event.*;
import java.applet.*;
public class Computer_client extends Applet implements Runnable,ActionListener
{  Button 计算;TextField 输入三边长度文本框,计算结果文本框;
   Socket socket=null;
   DataInputStream in=null; DataOutputStream out=null;
   Thread thread;
   public void init()
```

```
{  setLayout(new GridLayout(2,2));
   Panel p1=new Panel(),p2=new Panel();
   计算=new Button(" 计算");
   输入三边长度文本框=new TextField(12);计算结果文本框=new TextField(12);
   p1.add(new Label("输入三角形三边的长度,用逗号或空格分隔:"));
   p1.add( 输入三边长度文本框);
   p2.add(new Label("计算结果  "));p2.add(计算结果文本框);p2.add(计算);
   计算.addActionListener(this);
   add(p1);add(p2);
}
public void start()
{  try
      {  //和小程序所驻留的服务器建立套接字连接
         socket = new Socket(this.getCodeBase().getHost(), 4331);
         in =new DataInputStream(socket.getInputStream());
         out = new DataOutputStream(socket.getOutputStream());
      }
   catch (IOException e){}
   if(thread == null)
     {  thread = new Thread(this);
        thread.start();
     }
}
public void run()
{  String s=null;
   while(true)
    {    try{  s=in.readUTF();//堵塞状态,除非读取到信息.

               计算结果文本框.setText(s);
            }
         catch(IOException e)
            {  计算结果文本框.setText("与服务器已断开");
               break;
            }
    }
}
public void actionPerformed(ActionEvent e)
{  if(e.getSource()==计算)
   {  String s=输入三边长度文本框.getText();
```

```
            if(s!=null)
              {  try {  out.writeUTF(s);
                     }
                 catch(IOException e1){}
              }
        }
    }
}
```

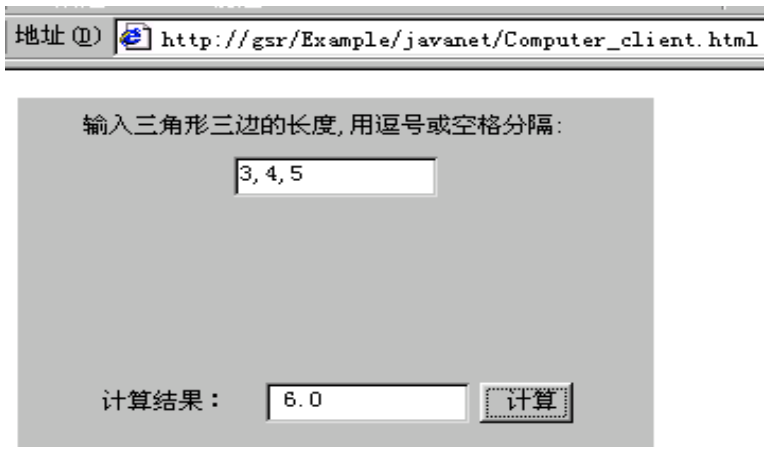


图 21.2　用套接字实现的客户-服务器交互计算

(2) 服务器端

```
import java.io.*;import java.net.*;
import java.util.*;import java.sql.*;
public class Computer_server
{  public static void main(String args[])
   {  ServerSocket server=null;Server_thread thread;
      Socket you=null;
      while(true)
       {  try{  server=new ServerSocket(4331);
             }
          catch(IOException e1)
             {  System.out.println("正在监听"); //ServerSocket 对象不能重复创建.
             }
          try{  you=server.accept();
                System.out.println("客户的地址:"+you.getInetAddress());
             }
          catch (IOException e)
             {  System.out.println("正在等待客户");
             }
```

```
            if(you!=null)
                {  new Server_thread(you).start(); //为每个客户启动一个专门的线程.
                }
            else
                {  continue;
                }
        }
    }
}
class Server_thread extends Thread
{  Socket socket;Connection Con=null;Statement Stmt=null;
   DataOutputStream out=null;DataInputStream  in=null;int n=0;
   String s=null;
   Server_thread(Socket t)
   {  socket=t;
      try {  in=new DataInputStream(socket.getInputStream());
             out=new DataOutputStream(socket.getOutputStream());
          }
      catch (IOException e)
          {}
   }
   public void run()
   {  while(true)
      {  double a[]=new double[3] ;int i=0;
         try{  s=in.readUTF();堵塞状态,除非读取到信息.

                StringTokenizer fenxi=new StringTokenizer(s," ,");
                while(fenxi.hasMoreTokens())
                  {  String temp=fenxi.nextToken();
                     try{  a[i]=Double.valueOf(temp).doubleValue();i++;
                        }
                     catch(NumberFormatException e)
                        {  out.writeUTF("请输入数字字符");
                        }
                  }
               double p=(a[0]+a[1]+a[2])/2.0;
               out.writeUTF(" "+Math.sqrt(p*(p-a[0])*(p-a[1])*(p-a[2])));
               sleep(2);
           }
```

```
            catch(InterruptedException e){}
            catch (IOException e)
               {  System.out.println("客户离开");
                  break;
               }
         }
      }
   }
```

本程序为了你实习的方便,我们在建立套接字连接时,使用的服务器地址是 localhost,它代表本地机的 IP 地址.你可以用个人 Web 发布管理器　用 win98 安装盘可以安装个人 Web 管理器　或 IIS 将服务器端含有 Java Applet 小程序在的文件夹设成 Web 共享.

注　套接字对象在网络编程中扮演着重要的角色,比如你可以练习使用套接字技术编写一个聊天室,服务器为每个客户启动一个线程,在该线程中通过套接字和客户交流信息,当你向服务器发送一条聊天信息　"大家好"时,服务器要让所有的这些线程中的输出流写入信息　"大家好",这样所有的客户的套接字的输入流就都读取到了这一条信息.如果你想把信息　"你好"送给特定的用户,服务器就让特定的线程中的输出流写入信息　"你好",那么只有特定的客户的套接字的输入流可以读取到这条信息.有关聊天室的详细设计和代码请参考作者为本系列教材编写的　Java 课程设计　一书.

21.3 InetAdress 类

我们已经知道 Internet 上的主机有两种方式表示地址

1　域名

例如,www.tsinghua.edu.cn

2　IP 地址

例如,202.108.35.210

java.net 包中的 InetAddress 类对象含有一个 Internet 主机地址的域名和 IP 地址

www.sina.com.cn/202.108.35.210.

域名容易记忆,当你在连接网络时输入一个主机的域名后,域名服务器　DNS　负责将域名转化成 IP 地址,这样我们才能和主机建立连接.

1. 获取 Internet 上主机的地址

我们可以使用 InetAddress 类的静态方法

```
getByName(String s);
```

将一个域名或 IP 地址传递给该方法的参数 s,获得一个 InetAddress 对象,该对象含有主机地

址的域名和 IP 地址,该对象用如下格式表示它包含的信息

www.sina.com.cn/202.108.37.40

下面的例子分别获取域名是 www.sina.com.cn 的主机域名及 IP 地址,及 IP 地址是 166.111.222.3 的主机域名及 IP 地址.

例子 4

```
import java.net.*;
public class DomainName
{  public static void main(String args[])
  {  try{  InetAddress address_1=InetAddress.getByName("www.sina.com.cn");
           System.out.println(address_1.toString());
           InetAddress address_2=InetAddress.getByName("166.111.222.3");
           System.out.println(address_2.toString());
        }
    catch(UnknownHostException e)
       {  System.out.println("无法找到 www.sina.com.cn");
       }
   }
}
```

当你运行上述程序时应保证你已经连接到 Internet 上 通过拨号或局域网连接到 internet 上 .上述程序的运行结果

```
www.sina.com.cn/202.108.37.40
maix.tup.tsinghua.edu.cn/166.111.222.3
```

另外,InetAddress 类中含有两个实例方法

public String getHostName() 取 InetAddress 对象所含的域名

public String getHostAddress() 获取 InetAddress 对象所含的 IP 地址.

下面的例子分别显示了主机 www.yahoo.com 的域名和 IP 地址.

例子 5

```
import java.net.*;
public class DomainName
{  public static void main(String args[])
   {  try{  InetAddress address=InetAddress.getByName("www.yahoo.com");
            String domain_name=address.getHostName();//获取 address 所含的域名.
            String IP_name=address.getHostAddress(); //获取 address 所含的 IP 地址.
            System.out.println(domain_name);
            System.out.println(IP_name);
```

```
        }
    catch(UnknownHostException e)
        {  System.out.println("无法找到 www.yahoo.com");
        }
   }
}
```

运行结果

```
www.yahoo.com
64.58.76.227
```

2. 获取本地机的地址

我们可以使用 **InetAddress** 类的静态方法

```
getLocalHost();
```

获得一个 **InetAddress** 对象,该对象含有本地机的域名和 **IP** 地址.

下面的例子分别显示了本机的域名和 **IP** 地址.

例子 6

```
import java.net.*;
public class DomainName
{  public static void main(String args[])
   {  try{  InetAddress address=InetAddress.getLocalHost();
            String domain_name=address.getHostName();//获取 address 所含的域名.
            String IP_name=address.getHostAddress();//获取 address 所含的 IP 地址.
            System.out.println(domain_name);
            System.out.println(IP_name);
        }
    catch(UnknownHostException e){}
   }
}
```

21.4 UDP 数 据 报

前面学习了基于 **TCP** 协议的网络套接字(**socket**),我们把套接字形象地比喻为打电话,一方呼叫,另一方负责监听,一旦建立了套接字连接,双方就可以进行通信了.这一节我们将介绍 **Java** 中基于 **UDP**(用户数据报协议)协议的网络信息传输方式.基于 **UDP** 的通信和基于 **TCP** 的通信不同,基于 **UDP** 的信息传递更快,但不提供可靠性保证.也就是说,数据在传输时,用户无法知道数据能否正确到达目的地主机,也不能确定数据到达目的地的顺序是否和发送的顺序相同.可以把 **UDP** 通信比作邮递信件,我们不能肯定所发的信件就一定能够到达目的地,也

不能肯定到达的顺序是发出时的顺序,可能因为某种原因导致后发出的先到达,另外,也不能确定对方收到信就一定会回信.既然 UDP 是一种不可靠的协议,为什么还要使用它呢 如果要求数据必须绝对准确的到达目的地,显然不能选择 UDP 协议来通信.但有时候人们需要较快速地传输信息,并能容忍小的错误,就可以考虑使用 UDP 协议.

基于 UDP 通信的基本模式是

1 将数据打包,称为数据包 好比将信件装入信封一样 ,然后将数据包发往目的地.

2 接受别人发来的数据包 好比接收信封一样 ,然后查看数据包中的内容.

1 发送数据包

1 首先用 DatagramPacket 类将数据打包,即用 DatagramPacket 类创建一个对象,称为数据包.用 DatagramPacket 的以下两个构造方法创建待发送的数据包

DatagramPacket(byte data[],int length,InetAddtress address,int port)

使用该构造方法创建的数据报对象具有下列两个性质

- 含有 data 数组指定的数据.
- 该数据包将发送到地址是 address,端口号是 port 的主机上.

我们称 address 是它的目标地址址,port 是这个数据包的目标端口号.

```
DatagramPack(byte data[],int offset,int length,InetAddtress address,int port)
```

使用该构造方法创建的数据报对象含有数组 data 从 offset 开始指定长度的数据,该数据包将发送到地址是 address,端口号是 port 的主机上.例如

```
byte data[]="近来好吗".getByte();
InetAddtress address=InetAddtress.getName("www.sian.com.cn");
DatagramPacket  data_pack=
new  DatagramPacket(data,data.length, address,980);
```

注 用上述方法创建的用于发送的数据包 data_pack 如果调用方法 public int getPort()可以获取该数据包目标端口号 调用方法 public InetAddress getAddres()可获取这个数据包的目标地址 调用方法 public byet[] getData()可以获取数据包中的数据.

2 然后用 DatagramSocket 类的不带参数的构造方法 DatagramSocket()创建一个对象,该对象负责发送数据包.

```
DatagramSocket  mail_out=new DatagramSocket();
mail_out.send(data_pack);
```

2 接收数据包

用 DatagramSocket 类另一个构造方法 DatagramSocket(int port) 创建一个对象,其中的参数必须和待接收的数据包的端口号相同.例如,如果发送方发送的数据包的端口号是 5666

```
DatagramSocket  mail_in=new DatagramSocket(5666);
```

该对象 mail_in 使用方法 receive(DatagramPacket pack)接受数据包.该方法有一个数据包参数 pack,方法 receive 把收到的数据包传递给该参数.因此我们必须预备一个数据包以便收取数据包.这时需使用 DatagramPack 类的另外一个构造方法 DatagramPack(byte data[],int length)创建一个数据包,用于接收数据包,例如

```
byte data[]=new byte[100];int length=90;
DatagramPacket pack=new  DatagramPacket(data,length);
mail_in.receive(pack);
```

该数据包 pack 将接收长度是 length 的数据放入 data.

注　　receive 方法可能会堵塞,直到收到数据报.

如果 pack 调用方法 getPort()可以获取所收数据包是从远程主机上的哪个端口发出的,即可以获取包的始发端口号　调用方法 InetAddress getAddres()可获取这个数据包来自哪个主机,即可以获取包的始发地址.我们称主机发出数据包使用的端口号为该包的始发端口号,发送数据包的主机地址称为数据包的始发地址.

数据包数据的长度不要超过 8192k.

在下面的例子 7 中两个主机(可用本地机模拟)互相发送和接收数据报.

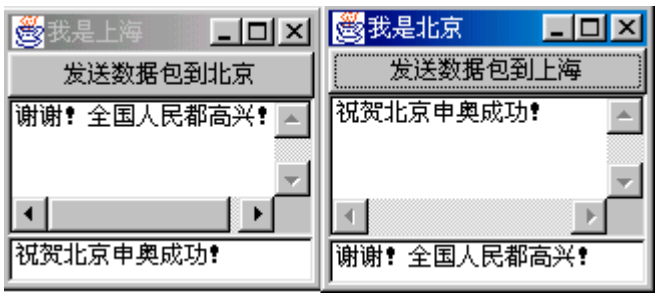


图 21.3　UDP 通信

例子 7　效果如图 21.3

主机 1

```
import java.net.*;import java.awt.*; import java.awt.event.*;
class Shanghai_Frame extends Frame implements Runnable,ActionListener
{  TextField out_message=new TextField("发送数据到北京  ");
   TextArea in_message=new TextArea();
   Button b=new Button("发送数据包到北京");
   Shanghai_Frame()
   {  super("我是上海");
      setSize(200,200);setVisible(true);
      b.addActionListener(this);
```

```
        add(out_message,"South");add(in_message,"Center");add(b,"North");
        Thread thread=new Thread(this);
        thread.start();//线程负责接收数据包
    }
  //点击按扭发送数据包
    public void actionPerformed(ActionEvent event)
    {  byte buffer[]=out_message.getText().trim().getBytes();
       try{  InetAddress address=InetAddress.getByName("localhost");
             //数据包的目标端口是888(那么收方(北京)需在这个端口接收):
     DatagramPacket data_pack=
new DatagramPacket(buffer,buffer.length, address,888);
             DatagramSocket mail_data=new DatagramSocket();
             in_message.append("数据报目标主机地址:"+data_pack.getAddress()+"\n");
             in_message.append("数据报目标端口是:"+data_pack.getPort()+"\n");
             in_message.append("数据报长度:"+data_pack.getLength()+"\n");
             mail_data.send(data_pack);
          }
       catch(Exception e){}
    }
    //接收数据包
    public void run()
    {  DatagramPacket pack=null;
       DatagramSocket mail_data=null;
       byte data[]=new byte[8192];
       try{
             pack=new DatagramPacket(data,data.length);
            //使用端口666来接收数据包(因为北京发来的数据报的目标端口是666).
             mail_data=new DatagramSocket(666);
          }
       catch(Exception e){}
       while(true)
          {  if(mail_data==null) break;
             else
                try{  mail_data.receive(pack);
                      int length=pack.getLength(); //获取收到的数据的实际长度.
                      InetAddress adress=pack.getAddress();//获取收到的数据包的始发地址.
                      int port=pack.getPort();//获取收到的数据包的始发端口.
                      String message=new String(pack.getData(),0,length);
                      in_message.append("收到数据长度  "+length+"\n");
```

```
                    in_message.append("收到数据来自  "+adress+"端口  "+port+"\n");
                    in_message.append("收到数据是  "+message+"\n");
                  }
               catch(Exception e){}
         }
   }
}
public class Shanghai
{  public static void main(String args[])
   {  Shanghai_Frame shanghai_win=new Shanghai_Frame();
      shanghai_win.addWindowListener(new WindowAdapter()
       {  public void windowClosing(WindowEvent e)
          {System.exit(0);
          }
        });
      shanghai_win.pack();
   }
}
```

主机 2

```
import java.net.*;import java.awt.*; import java.awt.event.*;
class Beijing_Frame extends Frame implements Runnable,ActionListener
{  TextField out_message=new TextField("发送数据到上海  ");
   TextArea in_message=new TextArea();
   Button b=new Button("发送数据包到上海");
   Beijing_Frame()
   {  super("我是北京");
      setSize(200,200);setVisible(true);
      b.addActionListener(this);
      add(out_message,"South");add(in_message,"Center");add(b,"North");
      Thread thread=new Thread(this);
      thread.start();//线程负责接收数据包
   }
//点击按扭发送数据包
   public void actionPerformed(ActionEvent event)
   {  byte buffer[]=out_message.getText().trim().getBytes();
      try{  InetAddress address=InetAddress.getByName("localhost");
            //数据包的目标端口是 666(那么收方(上海)需在这个端口接收):
            DatagramPacket data_pack=
```

```
                    new DatagramPacket(buffer,buffer.length, address,666);
            DatagramSocket mail_data=new DatagramSocket();
            in_message.append("数据报目标主机地址:"+data_pack.getAddress()+"\n");
            in_message.append("数据报目标端口是:"+data_pack.getPort()+"\n");
            in_message.append("数据报长度:"+data_pack.getLength()+"\n");
            mail_data.send(data_pack);
          }
        catch(Exception e){}
   }
   public void run()
   {  DatagramSocket mail_data=null;
      byte data[]=new byte[8192];
      DatagramPacket pack=null;
     try{
           pack=new DatagramPacket(data,data.length);
           //使用端口 888 来接收数据包(因为上海发来的数据报的目标端口是 888).
           mail_data=new DatagramSocket(888);
         }
     catch(Exception e){}
     while(true)
       {  if(mail_data==null) break;
          else
           try{  mail_data.receive(pack);
                 int length=pack.getLength(); //获取收到的数据的实际长度.
               InetAddress adress=pack.getAddress();//获取收到的数据包的始发地址.
               int port=pack.getPort();//获取收到的数据包的始发端口.
               String message=new String(pack.getData(),0,length);
               in_message.append("收到数据长度  "+length+"\n");
               in_message.append("收到数据来自  "+adress+"端口  "+port+"\n");
               in_message.append("收到数据是  "+message+"\n");
             }
           catch(Exception e){}
       }
   }
}
public class Beijing
{   public static void main(String args[])
   {  Beijing_Frame beijing_win=new Beijing_Frame();
      beijing_win.addWindowListener(new WindowAdapter()
```

```
        {  public void windowClosing(WindowEvent e)
           {System.exit(0);
            }
        });
       beijing_win.pack();
   }
}
```

21.5 广播数据报

广播数据报类似于电台广播,进行广播的电台需在指定的波段和频率上广播信息,接收者只有将收音机调到指定的波段,频率上才能收听到广播的内容.

广播数据报涉及到地址和端口.我们知道,Internet 的地址是 a.b.c.d 的形式.该地址的一部分代表用户自己主机,而另一部分代表用户所在的网络.当 a 小于 128,那么 b.c.d 就用来表示主机,这类地址称做 A 类地址.如果 a 大于等于 128 并且小于 192,则 a.b 表示网络地址,而 c.d 表示主机地址,这类地址称做 B 类地址.如果 a 大于等于 192,则网络地址是 a.b.c,d 表示主机地址,这类地址称做 C 类地址.224.0.0.0 与 224.255.255.255 是保留地址,称做 D 类地址.

广播数据报是一种较新的技术,要广播或接收广播的主机都必须加入到同一个 D 类地址.一个 D 类地址也称做一个广播组,加入到同一个广播组的主机可以在某个端口上广播信息,也可以在某个端口号上接收信息.

在下面的例子中,一个主机不断地重复广播天气预报 如图 21.4 a ,加入到同一组的主机都可以随时接收广播的信息.接收者将正在接收的信息放入一个文本区,把已接收到的全部信息放入另一个文本区 如图 21.4 b .

1 广播信息的主机 效果如图 21.4 a

BroadCast.java

```
import java.net.*;
public class BroadCast extends Thread
{   String s="天气预报,最高温度32度,最低温度25度";
    int port=5858;                                    //组播的端口.
    InetAddress group=null;                          //组播组的地址.
    MulticastSocket socket=null;                     //多点广播套接字.
   BroadCast()
   {try
     {
    group=InetAddress.getByName("239.255.8.0");  //设置广播组的地址为239.255.8.0.
    socket=new MulticastSocket(port);          //多点广播套接字将在port端口广播.
```

```
        socket.setTimeToLive(1);                    //多点广播套接字发送数据报范围为本地网络.
        socket.joinGroup(group);                   //加入广播组,加入group后,socket发送的数据报,
                                                    //可以被加入到group中的成员接收到.
          }
        catch(Exception e)
          { System.out.println("Error: "+ e);
          }
      }
      public void run()
      { while(true)
          { try
              {  DatagramPacket packet=null;                //待广播的数据包.
                 byte data[]=s.getBytes();
                 packet=new DatagramPacket(data,data.length,group,port);
                 System.out.println(new String(data));
                 socket.send(packet);                       //广播数据包.
                 sleep(2000);
              }
            catch(Exception e)
              {  System.out.println("Error: "+ e);
              }
          }
      }
      public static void main(String args[])
      {
        new BroadCast().start();
      }
    }
```

2 接收广播的主机 效果如图 21.4 b

Receive.java

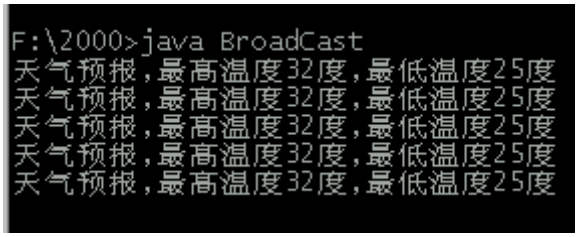
```
import java.net.*;import java.awt.*;
import java.awt.event.*;
public class Receive extends Frame implements Runnable,ActionListener
{ int port;                                          //组播的端口.
  InetAddress group=null;                            //组播组的地址.
  MulticastSocket socket=null;                       //多点广播套接字.
  Button 开始接收,停止接收;
  TextArea 显示正在接收内容,显示已接收的内容;
```

```
Thread thread;                                          //负责接收信息的线程.
boolean 停止=false;
public Receive()
 { super("定时接收信息");
   thread=new Thread(this);
   开始接收=new Button("开始接收");
   停止接收=new Button("停止接收");
   停止接收.addActionListener(this);
   开始接收.addActionListener(this);
   显示正在接收内容=new TextArea(10,10);
   显示正在接收内容.setForeground(Color.blue);
   显示已接收的内容=new TextArea(10,10);
   Panel north=new Panel();
   north.add(开始接收);
   north.add(停止接收);
   add(north,BorderLayout.NORTH);
   Panel center=new Panel();
   center.setLayout(new GridLayout(1,2));
   center.add(显示正在接收内容);
   center.add(显示已接收的内容);
   add(center,BorderLayout.CENTER);
   validate();
   port=5858;                                           //设置组播组的监听端口.
   try{
   group=InetAddress.getByName("239.255.8.0");  /设置广播组的地址为 239.255.8.0.
   socket=new MulticastSocket(port);          //多点广播套接字将在 port 端口广播.
   socket.joinGroup(group);        //加入广播组,加入 group 后,socket 发送的数据报,
                                   //可以被加入到 group 中的成员接收到.
      }
  catch(Exception e)
     { }
 setBounds(100,50,360,380);
 setVisible(true);
 addWindowListener(new WindowAdapter()
                  { public void windowClosing(WindowEvent e)
                    { System.exit(0);
                    }
                  });
```

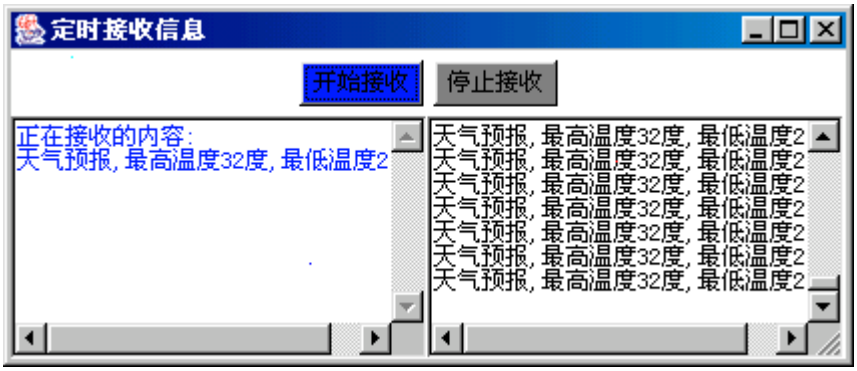
```
}
public void actionPerformed(ActionEvent e)
 {  if(e.getSource()==开始接收)
    { 开始接收.setBackground(Color.blue);
      停止接收.setBackground(Color.gray);
      if(!(thread.isAlive()))
         {  thread=new Thread(this);
         }
      try{  thread.start();
            停止=false;
         }
      catch(Exception ee) {}
    }
  if(e.getSource()==停止接收)
    { 开始接收.setBackground(Color.gray);
      停止接收.setBackground(Color.blue);
      thread.interrupt();
      停止=true;
    }
 }
 public void run()
 {  while(true)
    {byte data[]=new byte[8192];
     DatagramPacket packet=null;
     packet=new DatagramPacket(data,data.length,group,port);  //待接收的数据包.
     try { socket.receive(packet);
           String message=new String(packet.getData(),0,packet.getLength());
           显示正在接收内容.setText("正在接收的内容:\n"+message);
           显示已接收的内容.append(message+"\n");
         }
    catch(Exception e) {}
    if(停止==true)
         {  break;
         }
    }
 }

 public static void main(String args[])
 {
```

```
        new Receive();
    }
}
```



(a)



(b)

图 21.4　广播与接收

习题二十一

1　什么叫 socket?怎样建立 socket 连接　建立连接时,客户端和服务器端有什么不同

2　利用 URL 对象读取网络资源.URL 类中有一个方法 openStream(),一个 URL 对象可以使用这个方法获得一个输入流,然后用这个输入流读取 URL 对象处的资源.上机实习下列程序

```
import java.net.*;
import java.io.*;
public class E2
{  public static void main(String args[])
    {try{  URL url=new URL("http://www.tsinghua.edu.cn/welcome.html");
           InputStream in=url.openStream();
           int b,j=1000;byte tom[]=new byte[2500];
            while((b=in.read(tom,0,j))!=-1)
             {  String s=new String (tom,0,j);
                System.out.println(s);
             }
```

```
                in.close();
              }
            catch(Exception e){}
        }
    }
```

3　改进 21.5 中的 BroadCast.java,使得能通过窗口中的菜单选择要广播的文件或停止广播.广播文件时,每次广播文件的一行,并且可以重复广播一个文件.

第二十二章 Java 与图像

22.1 图像类型

我们不准备深入讨论不同图像的细节,但必须知道图像是矩形内的一组像素.需要指出的是,Java 支持 2 种主要的图像格式 GIF (Graphics Interchang Format),JPEG(Join Phtographic Expert Group).插图和图标经常使用 GIF 图像格式,相片图像经常使用 JPEG 这种格式.

22.2 Image 类

为了显示一幅图像,必须首先将其加载到计算机内存中.就好比放映电影前,应首先把胶片放入放映机一样.

Applet 类提供了一个重要的方法,用于从文件加载图像

```
public Image getImage(URL url,String name)
```

这个方法返回可以被显示在屏幕上的图像对象.即它的返回值是一个 Image 对象.

该方法经常使用的一种格式是

```
getImage(getCodeBase(),"family.jpg")
```

这个方法从你的 applet 小应用程序 文件所在目录中加载了名为 family.jpg 的图像.getCodeBase()方法返回小应用程序的 URL.

为了处理 getImage(URL url,String name)方法加载的图像,我们还必须使用 Image 类声明一个对象,以便存放该图象对象的引用,如

```
Image img= getImage( getCodeBase(),"family.jpg");
```

Image 类在 java.awt 包中.

图像被加载之后,就可以在 paint()方法中绘制它了,那么怎样来绘制呢 Graphics 类提供了几个名为 drawImage()的方法用于输出图像.它们的功能相似,都是在指定位置绘制一幅图像.不同之处在于确定图像大小方式,解释图像中透明部分的方式,以及是否支持图像的剪辑和拉伸.学会使用下面的最基本的 drawImage()方法,您可以很容易地使用另外的几个方法.

```
public boolean drawImage(Image img,int x,int y,ImageObserver observer)
```

参数 img 是被绘制的 Image 对象 x,y 是要绘制指定图像的矩形的左上角所处的位置,observer 是加载图像时的图像观察器.

由于 Applet 类已经实现了 ImageObserver 接口,因此它可以作为加载图像时的图像观察器.将 this 作为最后一个参数传递给 drawImage()便可将 Applet 对象传递过去,如

```
public void paint(Graphics g)
```

```
{g.drawImage(img,0,3,this);
}
```

下面我们看一个完整的例子

例子 1

```
import java.applet.*;
import java.awt.*;
public class Example22_1 extends Applet
{  Image img;
   public void start()
   {  img=getImage(getCodeBase(),"vintdev.jpg");
   }
   public void paint(Graphics g)
   {  g.drawImage(img,2,2,this);
   }
}
```

当我们使用 **drawImage(img,2,2,this)**来绘制图像时,如果小程序的宽或高设计的不合理,可能就会出现"**intdev.jpg** 图像的某些部分未能绘制到小程序中.为了克服这个缺点,可以使用 **drawImage()**的另一个方法

```
Public boolean drawImage(Image img,int x,int y,int width ,int height ,
                         ImageObserver observer)
```

在矩形内绘制加载的图像.参数 **img** 是被绘制的 **Image** 对象 **x,y** 是要绘制指定图像的矩形的左上角所处的位置,**width** 和 **height** 指定矩形的宽和高,**observer** 是加载图像时的图像观察器.

使用该方法时,不管原始图像的高和宽是多少,该图像会自动按比例调整自身大小以便适应目标区域的尺寸.

如果你不想图像有比例上的变化,在绘制之前可以通过 **Image** 类提供的方法获取被加载的图像的宽和高.如

```
img.getHeight(this);
img.getWidth(this);
```

这两个方法的参数是实现 **ImageObserver** 接口类创建的对象,**Java** 所有组件已经实现了该接口,因此任何一个组件都可以作为图象观察器.

让我们再看一个例子.

例子 2

```
import java.applet.*;import java.awt.*;
public class Example22_2 extends Applet
```

```
{  Image img;int height,width;
   public void start()
   {  img=getImage(getCodeBase(),"vintdev.jpg");
      height=img.getHeight(this);width=img.getWidth(this);
  }
   public void paint(Graphics g)
   { g.drawImage(img,22,72,width,height,this);
     g.drawImage(img,2+width,2+height,width,height,this);
   }
}
```

下面的例子 3 在画布上绘制一幅图象.

例子 3

```
import java.applet.*;import java.awt.*;
public class Wuqiong extends Applet
{  static Image img; Canvas canvas; static int width,height;
   public void init()
   {  setLayout(new GridLayout(3,1));add(new Button("祝好"));
      add(new Button("进步"));
      canvas=new Mycanvas(); add(canvas);
      width=getSize().width;height=getSize().height;
   }
   public void start()
   {  img=getImage(getCodeBase(),"Tom1.jpg");
   }
}
class Mycanvas extends Canvas
{  public void paint(Graphics g)
   { g.drawImage(Wuqiong.img,0,0,Wuqiong.width,(Wuqiong.height)/3,this);
   }
}
```

22.3 播放幻灯片和动画

HTML 也可以将图片贴到网页上去,但在 **HTML** 中想播放幻灯片确实是一件不现实的事情,因为为了看幻灯片必须在多个网页间切换,这是件痛苦的事情.在下面这个 **Java** 小程序中,用户只需单击鼠标就可变换幻灯片.

例子 4

```
import java.applet.*;import java.awt.*;import java.awt.event.*;
public class Example22_4 extends Applet implements MouseListener
{  final int number=38; int count=0;
   Image[] card=new Image[number];
   public void init()
   {  addMouseListener(this);
      for(int i=0;i<number;i++)
          {  card[i]=getImage(getCodeBase(),"jiafei"+i+".jpg");
          }
   }
   public void paint(Graphics g)
   {  if((card[count])!=null)
     g.drawImage(card[count],10,10,
                card[count].getWidth(this),card[count].getHeight(this),this);
   }
   public void mousePressed(MouseEvent e)
   {  count++;
      if(count>number)
        count=0;
     repaint();
   }
   public void mouseReleased(MouseEvent e){}
   public void mouseEntered(MouseEvent e){}
   public void mouseExited(MouseEvent e){}
   public void mouseClicked(MouseEvent e){}
}
```

在上面的例子中,你应把要播放的图片和小程序放在相同的目录中.

下面是用本章的知识和多线程编写的一个动画程序.

动画是一种错觉,运动的错觉是通过快速显示一组图片造成的,而这些图片在内容上只有微小的变化.

例子 5

```
import java.applet.*;import java.awt.*;import java.awt.event.*;
public class Example22_5 extends Applet implements Runnable
{  final int number=59; int count=0;
   Thread mythread;
   Image[]  pic=new Image[number];
   public void init()
   {  for(int i=0;i<number;i++)
```

```
        {  pic[i]=getImage(getCodeBase(),"tom"+i+".jpg");
        }
    }
    public void start()
    {  mythread=new Thread(this);
       mythread.start();
    }
    public void stop()
    {  mythread=null;
    }
    public void run()
    {  while(true)
         {  if(count>59)
              count=0;
            repaint();
            count++;
           try{  mythread.sleep(200);
              }
           catch(InterruptedException e){}
         }
    }
    public void paint(Graphics g)
    {  if((pic[count])!=null)
       g.drawImage(pic[count],10,10,
                   pic[count].getWidth(this),pic[count].getHeight(this),this);
    }
}
```

22.4 怎样在应用程序中绘制图象

Applet 类有一个下载图象的方法 **getImage**,因此我们在小程序中较容易绘制图像.我们编写应用程序时经常要使用 **Frame** 类,但 **Frame** 类没有获取图像的方法.Java.awt 包中有个抽象类 Toolkit,该类有一个获取图像的方法 getImage(String s).每个组件都从它的父类继承了一个得到 Toolkit 对象的方法 getToolkit(),该方法返回给调用者一个 Toolkit 对象.

要想在应用程序的一个 Frame 对象中绘制图像,应当首先创建一个画布组件对象,在这个画布上绘制图像,然后把画布添加到窗口中.

例子 6

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

```
import java.awt.event.*;
class Imagecanvas extends Canvas
{  Toolkit tool;  Image myimage;
   Imagecanvas()
   { setSize(200,200);
     tool=getToolkit();//得到一个 Toolkit 对象.
     myimage=tool.getImage("apple.jpg");//由 tool 负责获取图像.
   }
   public void paint(Graphics g)
   {
     g.drawImage(myimage,10,10,myimage.getWidth(this),myimage.getHeight(this),this);
   }
}
public class Example22_6
{ public static void main(String args[])
  {Imagecanvas canvas=new Imagecanvas();
   Frame frame=new Frame();
   frame.setLayout(new BorderLayout());
   frame.add(canvas,"Center");   frame.add("South",new Label());
   frame.add("West",new Label()); frame.add("North",new Label());
   frame.setSize(400,300);frame.setVisible(true);
   frame.pack();
   frame.addWindowListener(new WindowAdapter()
     {public void windowClosing(WindowEvent e)
        {System.exit(0);}
     });
   }
}
```

现在我们把例子 5 改成应用程序.

例子 7

```
import java.applet.*;import java.awt.*;
import java.awt.event.*;
class Imagecanvas extends Canvas implements MouseListener
{final int number=59; int count=0; Toolkit tool;
 Image[] card=new Image[number];
 Imagecanvas()
 { getSize(); tool=getToolkit(); addMouseListener(this);
   for(int i=0;i<number;i++)
```

```
    {card[i]=tool.getImage("tom"+i+".jpg");
    }
  }
  public void paint(Graphics g)
  {if((card[count])!=null)
   g.drawImage(card[count],10,10,
   card[count].getWidth(this),card[count].getHeight(this),this);
  }
  public Dimension getPreferredSize()
   {return new Dimension(160,100);
   }
  public void mousePressed(MouseEvent e)
  { count++;
     if(count>number-1)
     count=0;
     repaint();
  }
  public void mouseReleased(MouseEvent e){}
  public void mouseEntered(MouseEvent e){}
  public void mouseExited(MouseEvent e){}
  public void mouseClicked(MouseEvent e){}
}
public class Example226
{ public static void main(String args[])
   {Imagecanvas canvas=new Imagecanvas();
    Frame frame=new Frame(); frame.setLayout(new BorderLayout());
    frame.add(canvas,"Center");
    frame.add("South",new Label());frame.add("West",new Label());
    frame.add("North",new Label());
    frame.setSize(400,300);frame.setVisible(true); frame.pack();
    frame.addWindowListener(new WindowAdapter()
      {public void windowClosing(WindowEvent e)
         {System.exit(0);}
      });
    }
}
```

22.5 怎样设置 Java 窗口的图标

Frame 对象可以使用 **setIconImage(Image image)**方法设置窗口左上角的图标,**Java** 窗口的默认图标是一个咖啡杯.

例子 8

```
import java.awt.*;import java.awt.event.*;
public class Frame_Icon
{ public static void main(String args[])
  { Frame frame=new Frame();
    Toolkit tool= frame.getToolkit();//得到一个 Toolkit 对象.
    Image myimage=tool.getImage("apple.jpg");//由 tool 负责获取图像.
   //设置窗口的图标是 myimage 指定的图象 apple.jpg
   frame.setIconImage(myimage);
   frame.setSize(400,300);frame.setVisible(true);
   frame.addWindowListener(new WindowAdapter()
     {public void windowClosing(WindowEvent e)
        {  System.exit(0);
        }
     });
  }
}
```

习题二十二

1.Image 是一个抽象类,不能直接使用该类创建对象,那么小应用程序或应用程序用怎样的方法获得一个 **Image** 对象呢

2 结合多线程方法,编写一个小应用程序,在观察图象时实现图象的一些特殊效果,比如,图象逐渐推进放大等.

3 改进十二章的例子 6,在该例子中的画布中画一幅图象,比如照片等.

第二十三章 Java 数据库连接(JDBC)

JDBC(Java DataBase Connectivity)是 Java 数据库连接 API.简单地说,JDBC 能完成 3 件事

1　　与一个数据库建立连接,

2　　向数据库发送 SQL 语句,

3　　处理数据库返回的结果.

下面我们就结合一个例子来实现这 3 步.

JDBC 在设计上和 ODBC 很相似.JDBC 和数据库建立连接的一种方式是首先建立起一个 JDBC—ODBC 桥接器.由于 ODBC 驱动程序被广泛的使用,建立这种桥接器后,使得 JDBC 有能力访问几乎所有类型的数据库.

假设我们有一个用 Access 设计的数据库 student.mdb,该库中有一个表,表的名字是 chengjibiao,如图 23.1 所示.

chengjibiao : Table

	学号	姓名	出生日期	数学	物理	英语
▶	1	王德彪	67-9-9	68	70	89
	2	团庆见	88-1-12	50	68	98
	3	加正号	60-5-24	45	49	45
	4	枣小红	58-9-10	98	60	73
	5	黄客仁	77-9-12	90	89	80
	6	客人好	72-8-8	30	34	58

图 23.1　待连接的数据库

设置数据源

为了同这个数据库建立连接,首先配置一个 ODBC 数据源.打开 Windows 中的控制面

图 23.2　控制面板

板,出现下列界面,图 23.2 所示.双击 ODBC DATA Source 图标.出现下列界面,如图 23.3 所示.图中显示了已有的数据源的名称.点击 Add 按钮,增加新的数据源.选择 Mirosoft Aceess Driver(*.mdb)之后,点击完成按钮　为数据源选择了驱动程序　,如图如图 23.4,23.5 所示.

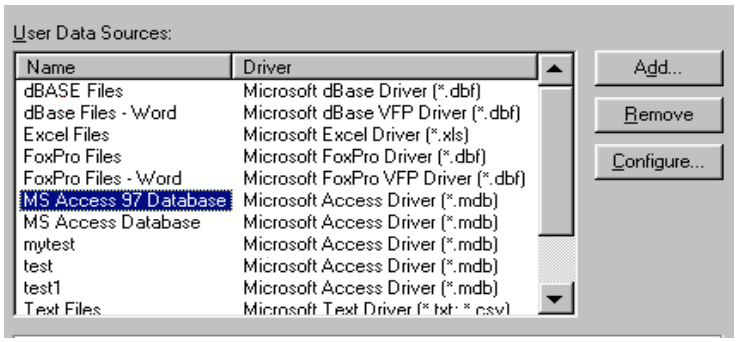


图 23.3　用户数据源对话框

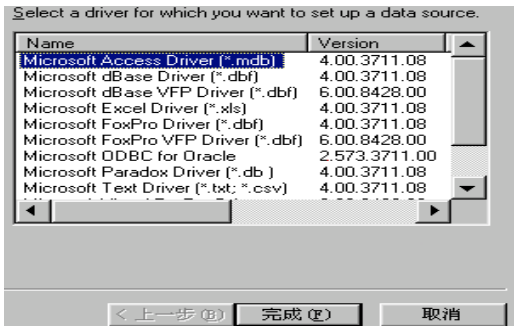


图 23.4　添加新数据源

在 Data Source Name 文本框里为数据源起一个你自己喜欢的名字,这里我们起的名字是 redsun 如图 23.5 所示,.我们的这个数据源就是指某个数据库.再单击 Select 按钮,把数据源 redsun 设成是我们要连接的数据库 student.mdb,见图 23.6 所示.

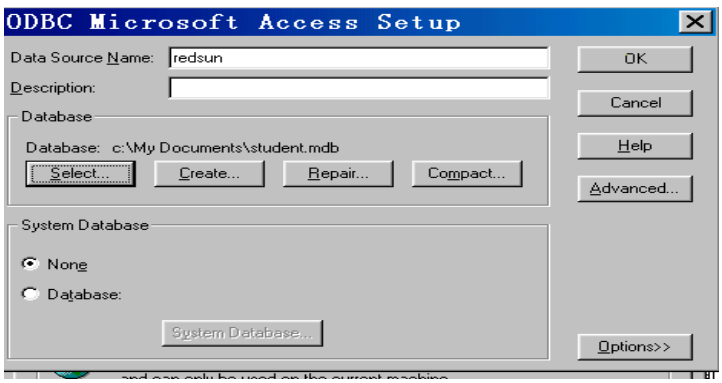


图 23.5 输入数据源名称

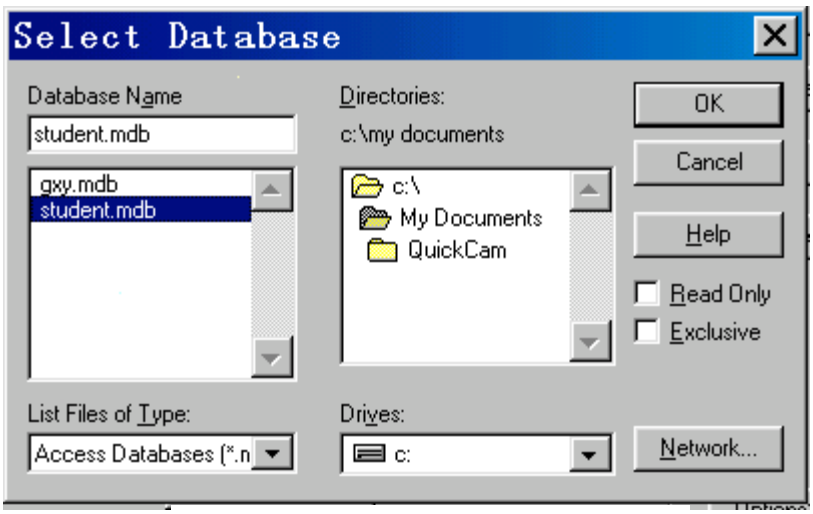


图 23.6　指定数据库

选择 student.mdb 后单击 ok 按钮.单击 ok 按钮后,又出现了和图 23.5 一样的窗口.

如果我们要为数据源 redsun 设置一个 login name 和 password 的话,就再单击 Advance 按钮,否则单击 ok 按钮就完成了数据源设置的全部步骤.现在,假设我们要给数据源一个 login name 和 password,那么单击 Advance 按钮.打开如图 23.7 所示的对话框.

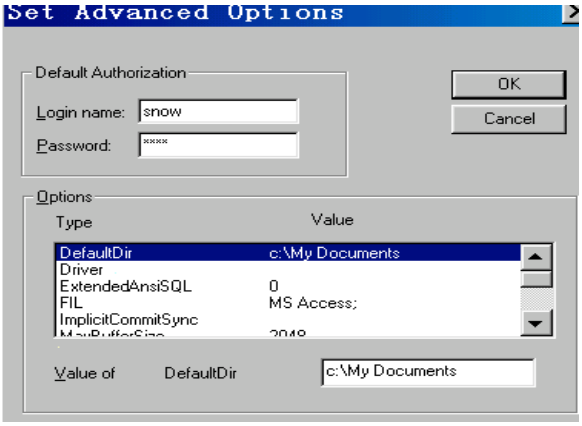


图 23.7　设置密码

在 login name 里输入一个你喜欢的名字,这里我们给的是 snow,在 password 文本框里输入口令 这里我们输入的是 ookk .(注意这里的 password 和你的数据库本身的 password 没有关系).然后单击 ok 按钮,又回到图 23.5,再单击 ok 按钮就完成了全部过程.关闭控制面板即可.

23.1 JDBC-ODBC 桥接器

1　建立 JDBC—ODBC 桥接器

现在我们有了一个数据源,这个数据源就是一个数据库.

为了要连接到这个数据库,我们首先要建立一个 JDBC-ODBC 桥接器

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

这里,Class 是包 java.lang 中的一个类,该类通过调用它的静态方法 forName 就可以建立 JDBC-ODBC 桥接器.

建立桥接器时可能发生异常,因此捕获这个异常.所以建立桥接器的标准是

```
try { Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }
catch(ClassNotFoundException e)
   {}
```

2 连接到数据库

首先使用包 java.sql 中的 Connection 类声明一个对象,然后再使用类 DriverManager 调用它的静态方法 getConnection 创建这个连接对象

```
Connection  con = DriverManager.getConnection("jdbc:odbc:数据源名字",
                        "数据源的 login name",  "数据源的 password ");
```

假如没有为数据源设置 login name 和 password,那么连接形式是

```
Connection  con = DriverManager. getConnection("jdbc:odbc:数据源名字", "",  "");
```

为了能和数据源 redsun 交换数据,建立 Connection 对象如下

```
Connection  con = DriverManager. getConnection("jdbc:odbc:redsun", "snow", "ookk ");
```

建立连接时应捕获 SQLException 异常,

```
try{ Connection  con = DriverManager. getConnection("jdbc:odbc:redsun", "snow", "ookk
")
   }
catch(SQLException e){}
```

这样就建立了到数据库 student.mdb 的连接.

3 向数据库发送 SQL 语句

首先使用 Statement 声明一个 SQL 语句对象,然后通过刚才创建的连接数据库的对象 con 调用方法 createStatment()创建这个 SQL 语句对象.

```
try {Statement   sql=con.createStatement();}
catch(SQLException e ){}
```

4 处理查询结果

有了 SQL 对象后,这个对象就可以调用相应的方法实现对数据库的查询和修改.并将查

询结果存放在一个 ResultSet 类声明的对象中,也就是说 SQL 语句对数据库的查询操作将返回一个 ResultSet 对象.

```
ResultSet  rs=sql.executeQuery("SELECT * FROM  成绩表")
```

ResultSet 对象实际上是一个管式数据集,即它是以统一形式的列组织的数据行组成.

ResultSet 对象一次只能看到一个数据行,使用 next()方法走到下一数据行,获得一行数据后,ResultSet 对象可以使用位置索引 第一列使用 1,第二列使用 2 等等 或使用列名称,以便使用 getxxxx()方法获得字段值.表 23.1 给了出了 ResultSet 对象的若干方法.

表 23.1 ResultSet 类的若干方法

返回类型	方法名称
boolean	next()
byte	getByte(int columnIndex)
Date	getDate(int columnIndex)
double	getDouble(int columnIndex)
float	getFloat(int columnIndex)
int	getInt(int columnIndex)
long	getLong(int columnIndex)
String	getString(int columnIndex)
byte	getByte(String columnName)
Date	getDate(String columnName)
double	getDouble(String columnName)
float	getFloat(String columnName)
int	getInt(String columnName)
long	getLong(String columnName)
String	getString(String columnName)

23.2 顺序查询

使用结果集 Result 的 next()方法,可以顺序的查询.一个结果集将游标最初定位在第一行的前面,第一次调用 next()方法使游标移动到第一行.next()方法返回一个 boolean 型数据,当游标移动到最后一行之后返回 false.

在下面的例子 1 中,我们查询数据库 student.mdb 中 chengjibaio 表里包含全部字段值的记录.

例子 1

```
import java.sql.*;
public class Example23_1
{  public static void main(String args[])
   {  Connection con;Statement sql; ResultSet rs;
      try {  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
          }
```

```
        catch(ClassNotFoundException e)
             {  System.out.println(""+e);
             }
         try
             {
                con=DriverManager.getConnection("jdbc:odbc:redsun","snow","ookk");
                sql=con.createStatement();
                rs=sql.executeQuery("SELECT * FROM chengjibiao");
                while(rs.next())
                { String  number=rs.getString(1);
                  String  name=rs.getString(2);
                  Date    date=rs.getDate(3);
                  int     math=rs.getInt("数学");
                  int  physics=rs.getInt("物理");
                  int  english=rs.getInt("英语");
                  System.out.println("学号  "+number);
                  System.out.print("  姓名  "+name);
                  System.out.print("  出生  "+date);
                  System.out.print("  数学  "+math);
                  System.out.print("  物理  "+physics);
                  System.out.print("  英语  "+english);
                 }
                con.close();
               }
         catch(SQLException e1) {}

      }
   }
```

在下面的例子 2 中查询"英语"字段值大于 80 的记录,只显示"姓名"字段 第 2 个字段和"英语"字段.

例子 2

```
   import java.sql.*;
   public class Example23_2
   {  public static void main(String args[])
      {  Connection con;Statement sql; ResultSet rs;
         try {  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
             }
         catch(ClassNotFoundException e)
```

```
            {  System.out.println(""+e);
            }
        try
            {  con=DriverManager.getConnection("jdbc:odbc:redsun","snow","ookk");
               sql=con.createStatement();
               rs=sql.executeQuery("SELECT 姓名,英语 FROM chengjibiao WHERE 英语 >= 80 ");
               while(rs.next())
               { String  name=rs.getString(1);
                 int  english=rs.getInt(2);
                 System.out.println("  姓名  "+name);
                 System.out.print("  英语  "+english);
               }
              con.close();
             }
        catch(SQLException e1) {}
    }
 }
```

注 当使用 ResultSet 的 getXXX 方法查看一行记录时,不可以颠倒字段的顺序,例如,不可以

```
rs.getInt(5);
rs.getInt(3)
```

下面是一个英汉小词典,词典的内容放在一个 Access 数据库 English.mdb 的表 1 中,如图 23.8 所示.

单词	解释
arm	臂
cloud	云
ear	耳
eight	八
eye	眼
face	脸
five	五
foot	脚
four	四
game	游戏
glad	高兴
hand	手
head	头

图 23.8 单词表

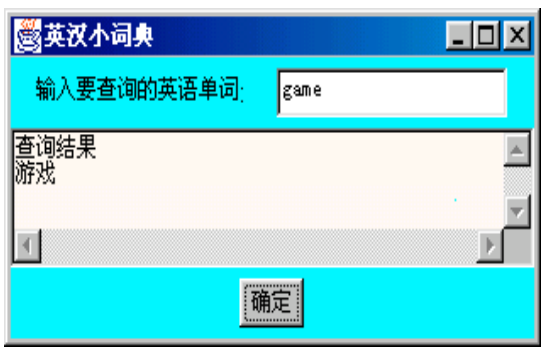


图 23.9 运行界面

例子 3 效果如图 23.9

```
import java.awt.*;import java.net.*;
import java.sql.*;import java.awt.event.*;
```

```
class DataWindow extends Frame implements ActionListener
{  TextField englishtext;TextArea chinesetext; Button button;
   DataWindow()
   {  super("英汉小词典");
      setBounds(150,150,300,120);
      setVisible(true);
      englishtext=new TextField(16);
      chinesetext=new TextArea(5,10);
      button=new Button("确定");
      Panel p1=new Panel(),p2=new Panel();
      p1.add(new Label("输入要查询的英语单词:"));
      p1.add(englishtext);
      p2.add(button);
      add(p1,"North");add(p2,"South");add(chinesetext,"Center");
      button.addActionListener(this);
     addWindowListener(new WindowAdapter()
                     {  public void windowClosing(WindowEvent e)
                           {  System.exit(0);
                           }
                     });
   }
   public void actionPerformed(ActionEvent e)
   {  if(e.getSource()==button)
        {  chinesetext.setText("查询结果");
           try{  Liststudent();
              }
           catch(SQLException ee) {}
        }
   }
   public  void Liststudent() throws SQLException
   { String cname,ename;
     try{  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        }
     catch(ClassNotFoundException e){}
     Connection Ex1Con=DriverManager.getConnection("jdbc:odbc:test","gxy","ookk");
     Statement Ex1Stmt=Ex1Con.createStatement();
     ResultSet rs=Ex1Stmt.executeQuery("SELECT *  FROM 表1 ");
     boolean boo=false;
     while((boo=rs.next())==true)
```

```
        {  ename=rs.getString("单词");
           cname=rs.getString("解释");
           if(ename.equals(englishtext.getText()))
              {
                chinesetext.append('\n'+cname);
                break;
              }
         }
      Ex1Con.close();
     if(boo==false)
        {  chinesetext.append('\n'+"没有该单词");
        }
    }
}
public class Example23_3
{  public static void main(String args[])
     {  DataWindow window=new DataWindow();window.pack();
     }
}
```

23.3 可滚动结果集

前面我们学习了使用 Result 的 next()方法顺序地查询数据,但有时候我们需要在结果集中前后移动,或显示结果集指定的一条记录等等.这时,我们必须要返回一个可滚动的结果集.为了得到一个可滚动的结果集,和上一节不同的是,我们必须使用下述方法先获得一个 Statement 对象

```
Statement stmt=con.createStatement(int type ,int concurrency);
```

然后,根据参数的 type,concurrency 的取值情况,stmt 返回相应类型的结果集

```
ResultSet  re=stmt.executeQuery(SQL语句);
```

type 的取值决定滚动方式,取值可以是

ResultSet.TYPE_FORWORD_ONLY　结果集的游标只能向下滚动.

ResultSet.TYPE_SCROLL_INSENSITIVE　结果集的游标可以上下移动,当数据库变化时,当前结果集不变.

ResultSet.TYPE_SCROLL_SENSITIVE　返回可滚动的结果集,当数据库变化时,当前结果集同步改变.

Concurrency　取值决定是否可以用结果集更新数据库,Concurrency 取值

ResultSet.CONCUR_READ_ONLY　不能用结果集更新数据库中的表.

ResultSet.CONCUR_UPDATETABLE　能用结果集更新数据库中的表.

滚动查询经常用到 ResultSet 的下述方法

- **public boolean previous()** 将游标向上移动,该方法返回 boolean 型数据,当移到结果集第一行之前时返回 false.
- **public void beforeFirst** 将游标移动到结果集的初始位置,即在第一行之前.
- **public void afterLast()** 将游标移到结果集最后一行之后.
- **public void first()** 将游标移到结果集的第一行.
- **public void last()** 将游标移到结果集的最后一行.
- **public boolean isAfterLast()** 判断游标是否在最后一行之后.
- **public boolean isBeforeFirst()** 判断游标是否在第一行之前
- **public boolean ifFirst()** 判断游标是否指向结果集的第一行.
- **public boolean isLast()** 判断游标是否指向结果集的最后一行.
- **public int getRow()** 得到当前游标所指行的行号,行号从 1 开始,如果结果集没有行,返回 0
- **public boolean absolute(int row)** 将游标移到参数 row 指定的行号.

注意,如果 row 取负值,就是倒数的行数,absolute(-1)表示移到最后一行,absolute(-2)表示移到倒数第 2 行.当移动到第一行前面或最后一行的后面时,该方法返回 false.

在下面的例子 4 中,首先将游标移动到最后一行,然后再获取行号,这样就获得表中的记录数目.然后我们倒序输出结果集中的记录,即首先输出最后一行.最后单独输出第 5 条记录.

例子 4

```
import java.sql.*;
public class Example23_4
{  public static void main(String args[])
   {  Connection con;Statement sql; ResultSet rs;
      try {  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
          }
     catch(ClassNotFoundException e)
          {  System.out.println(""+e);
          }
      try
          {  con=DriverManager.getConnection("jdbc:odbc:redsun","snow","ookk");
             sql=con.createStatement();
             sql=
  con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
             //返回可滚动的结果集
             rs=sql.executeQuery("SELECT 姓名,英语 FROM chengjibiao WHERE 英语 >= 80 ");
             //将游标移动到最后一行
             rs.last();
             //获取最后一行的行号
```

```
                int number=rs.getRow();
                System.out.println("该表共有"+number+"条记录");
                //为了逆序输出记录,需将游标移动到最后一行之后
                rs.afterLast();
                while(rs.previous())
                { String  name=rs.getString(1);
                  int  english=rs.getInt("英语");
                  System.out.println("  姓名  "+name);
                  System.out.print("  英语  "+english);
                }
               System.out.println("单独输出第5条记录:");
               rs.absolute(5);
                  String  name=rs.getString(1);
                  int  english=rs.getInt("英语");
                  System.out.println("  姓名  "+name);
                  System.out.print("  英语  "+english);
               con.close();
              }
        catch(SQLException e1) {}
    }
}
```

23.4 排序查询

可以在 SQL 语句中使用 ORDER BY 子语句,对记录排序.使用 SQL 语句的 ORDER BY 子语句查询所同学的成绩,可以选择按 3 科的总分从低到高排列记录,按姓氏拼音排序或英语成绩排序等等.例如,按总成绩排序查询的 SQL 语句

```
SELECT * FROM student ORDER BY 数学成绩+英语成绩+物理成绩.
```

下面的例子 5 按英语成绩排序.

例子 5

```
import java.sql.*;
public class Example23_5
{  public static void main(String args[])
   {  Connection con;Statement sql; ResultSet rs;
      try {  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
          }
```

```
        catch(ClassNotFoundException e)
             {  System.out.println(""+e);
             }
         try
             {  con=DriverManager.getConnection("jdbc:odbc:redsun","snow","ookk");
                sql=con.createStatement();
                String condition="SELECT 姓名,英语 FROM chengjibiao ORDER BY 英语";
                rs=sql.executeQuery(condition);
                while(rs.next())
                { String  name=rs.getString(1);
                  int  english=rs.getInt(2);
                  System.out.println("  姓名  "+name);
                  System.out.print("  英语  "+english);
                }
               con.close();
              }
         catch(SQLException e1) { System.out.print(e1);}
      }
   }
```

23.5 模糊查询

可以用 SQL 语句操作符 LIKE 进行模式般配,使用”%”代替一个或多个字符,用一个下划线”_”代替一个字符.比如,下述语句查询姓氏是”王”的记录

```
rs=sql.executeQuery("SELECT * FROM students WHERE 姓名  LIKE  '王%'  ");
```

例子 6 查询所有名字中含有”小”的记录.

例子 6

```
import java.sql.*;
public class Example23_6
{  public static void main(String args[])
   {  Connection con;Statement sql; ResultSet rs;
      try { Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
          }
     catch(ClassNotFoundException e)
          {  System.out.println(""+e);
          }
      try
          {  con=DriverManager.getConnection("jdbc:odbc:redsun","snow","ookk");
```

```
            sql=con.createStatement();
        rs=sql.executeQuery("SELECT 姓名,数学 FROM chengjibiao WHERE 姓名 LIKE '%小%' ");
            while(rs.next())
            { String  name=rs.getString(1);
              int  math=rs.getInt(2);
              System.out.println(" 姓名 "+name);
              System.out.print(" 数学 "+math);
            }
           con.close();
          }
       catch(SQLException e1) { System.out.print(e1);}
    }
}
```

23.6 更新,添加,删除记录

我们可以使用 SQL 语句更新记录中字段的值

Statement 对象调用方法

```
public int executeUpdate  String sqlStatement  ;
```

通过参数 sqlStatement 指定的方式实现对数据库表中记录的字段值的更新,例如,下述语句将表 students 中王名同学的数学字段的值更新 88

```
executeUpdate("UPDATE chengjibiao SET 数学 =  88 WHERE 姓名='王名'");
```

Statement 对象调用方法

```
public int executeUpdate  String sqlStatement  ;
```

通过参数 sqlStatement 指定的方式实现向数据库表中添加新记录,例如,下述语句将向表 students 中添加一条新的记录 '199911','刘明明',70,95,96 .

```
executeUpdate("INSERT INTO chengjibiao VALUES  '199911','刘,明明',70,95,96  ")
```

Statement 对象调用方法

```
public int executeUpdate  String sqlStatement  ;
```

通过参数 sqlStatement 指定的方式删除数据库表中的记录,例如,下述语句将删除学号是 199904 的记录

```
executeUpdate("DELETE  FROM chengjibiao WHERE 学号 = '199904' ");
```

例子 6

下面的例子 6 实现了对数据库 English.mdb 的表 1 中记录的更新,填加.

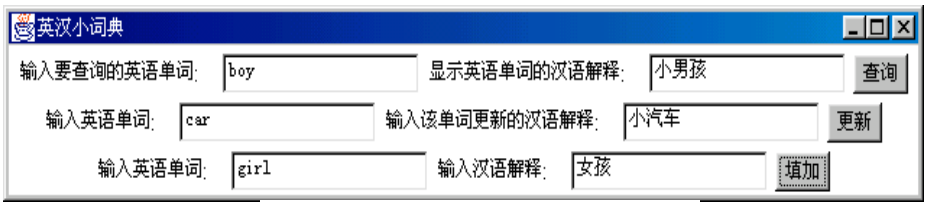


图 23.10　数据库查询,更新和填加

例子 6　效果如图 23.10

```
import java.awt.*;
import java.sql.*;import java.awt.event.*;
class DataWindow extends Frame implements ActionListener
 { TextField   待查英文单词_文本条,汉语解释_文本条,
                更新英文单词_文本条,更新汉语解释_文本条,
                填加英文单词_文本条,填加汉语解释_文本条;
   Button       查询按扭,更新按扭,填加按扭;
   int 查询记录=0;
   Connection Con=null;Statement Stmt=null;
   DataWindow()
   {  super("英汉小词典");
     setBounds(150,150,300,120);
     setVisible(true);setLayout(new GridLayout(3,1));
     try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");}
     catch(ClassNotFoundException e){}
      try{
Con=DriverManager.getConnection("jdbc:odbc:test","gxy","ookk");
          Stmt=Con.createStatement();
        }
      catch(SQLException ee) {}
      待查英文单词_文本条=new TextField(16);
      汉语解释_文本条=new TextField(16);
      更新英文单词_文本条=new TextField(16);
      更新汉语解释_文本条=new TextField(16);
      填加英文单词_文本条=new TextField(16);
      填加汉语解释_文本条=new TextField(16);
      查询按扭=new Button("查询");
      更新按扭=new Button("更新");
      填加按扭=new Button("填加");
      Panel p1=new Panel(),p2=new Panel(),p3=new Panel();
      p1.add(new Label("输入要查询的英语单词:"));
```

```
    p1.add( 待查英文单词_文本条);
    p1.add(new Label("显示英语单词的汉语解释:"));
    p1.add(汉语解释_文本条);
    p1.add(查询按扭);
    p2.add(new Label("输入英语单词:"));p2.add( 更新英文单词_文本条);
    p2.add(new Label("输入该单词更新的汉语解释:"));
    p2.add(更新汉语解释_文本条);
    p2.add(更新按扭);
    p3.add(new Label("输入英语单词:"));p3.add(  填加英文单词_文本条);
    p3.add(new Label("输入汉语解释:"));p3.add(填加汉语解释_文本条);
    p3.add(填加按扭);
    add(p1);add(p2);add(p3);
    查询按扭.addActionListener(this);更新按扭.addActionListener(this);
    填加按扭.addActionListener(this);
    addWindowListener(new WindowAdapter()
   {public void windowClosing(WindowEvent e)
              {setVisible(false);System.exit(0);  } } );
}
public void actionPerformed(ActionEvent e)
{if(e.getSource()==查询按扭)
  {  查询记录=0;
     try{ 查询();}
     catch(SQLException ee) {}
  }
else if(e.getSource()==更新按扭)
 {  try{ 更新();}
    catch(SQLException ee) {}
 }
else if(e.getSource()==填加按扭)
 {  try{ 填加();}
    catch(SQLException ee) {}
  }
}
public  void 查询() throws SQLException
{  String cname,ename;
    Con=DriverManager.getConnection("jdbc:odbc:test","gxy","ookk");
    ResultSet rs=Stmt.executeQuery("SELECT * FROM 表 1 ");
    while (rs.next())
     {  ename=rs.getString("单词"); cname=rs.getString("解释");
```

```
            if(ename.equals( 待查英文单词_文本条.getText().trim()))
            {  汉语解释_文本条.setText(cname);查询记录=1; break;
            }
       }
        Con.close();
      if(查询记录==0)
         {  汉语解释_文本条.setText("没有该单词");
         }
  }
  public void 更新() throws SQLException
    {  String s1="'"+更新英文单词_文本条.getText().trim()+"'",
            s2="'"+更新汉语解释_文本条.getText().trim()+"'";
       String temp="UPDATE 表1 SET 解释 ="+s2+" WHERE 单词 = "+s1 ;
       Con=DriverManager.getConnection("jdbc:odbc:test","gxy","ookk");
       Stmt.executeUpdate(temp);  Con.close();
    }
    public void 填加() throws SQLException
    {  String s1="'"+填加英文单词_文本条.getText().trim()+"'",
            s2="'"+填加汉语解释_文本条.getText().trim()+"'";
     String temp="INSERT INTO 表1 VALUES ("+s1+","+s2+")";
     Con=DriverManager.getConnection("jdbc:odbc:test","gxy","ookk");
     Stmt.executeUpdate(temp);
     Con.close();
    }
}
public class Database
{ public static void main(String args[])
    {DataWindow window=new DataWindow();window.pack();
   }
}
```

23.7 数据库访问中的套接字技术

由于小应用程序安全性的限制,所以前面的例子都是应用程序,因为对数据库的访问需要调用客户端相应的数据库驱动程序,而小应用程序不允许对本地机的文件进行读写操作.在这一节我们利用套接字技术实现小应用程序中对数据库的访问.小应用程序只是利用套接字连接向服务器发送一个查询的条件,而服务器负责对数据库的查询,然后服务器再将查询的结果利用建立的套接字返回给客户端.如果用这种方式访问数据库,客户端不需要任何数据库驱动程序.

图 23.12 用套接字实现的数据

例子 7

(1) 客户端程序　效果如图 23.11

```
import java.net.*;import java.io.*;
import java.awt.*;import java.awt.event.*;
import java.applet.*;
public class Database_client extends Applet implements Runnable,ActionListener
{ Button 查询;TextField 英文单词_文本框,汉语解释_文本框;
  Socket socket=null;
  DataInputStream in=null;
  DataOutputStream out=null;
  Thread thread;
 public void init()
 {查询=new Button("查询");
  英文单词_文本框=new TextField(10);汉语解释_文本框=new TextField(10);
  add(new Label("输入要查询的英文单词"));add(英文单词_文本框);
  add(new Label("汉语解释  "));add(汉语解释_文本框);add(查询);
  查询.addActionListener(this);
 }
 public void start()
 { try
   {socket = new Socket(this.getCodeBase().getHost(), 4331);
   in = new DataInputStream(socket.getInputStream());
   out = new DataOutputStream(socket.getOutputStream());
   }
   catch (IOException e){}
  if (thread == null)
   {thread = new Thread(this);
    thread.setPriority(Thread.MIN_PRIORITY);
```

```
        thread.start();
      }
    }
    public void stop()
    {try{out.writeUTF("客户离开");}
     catch(IOException e1){}
    }
    public void destroy()
    {try{out.writeUTF("客户离开");}
     catch(IOException e1){}
    }
    public void run()
    {String s=null;
       while(true)
        { try{s=in.readUTF();
             }
          catch (IOException e)
            {汉语解释_文本框.setText("与服务器已断开");break;
             }
         汉语解释_文本框.setText(s);
       }
    }
    public void actionPerformed(ActionEvent e)
    {if (e.getSource()==查询)
        { String s=英文单词_文本框.getText();
          if(s!=null)
           { try{out.writeUTF(s);}
             catch(IOException e1){}
           }

        }
    }
   }
```

(2) 服务器端程序

```
import java.io.*;import java.net.*;
import java.util.*;import java.sql.*;
public class Database_server
{   public static void main(String args[])
    {   ServerSocket server=null;Server_thread thread;
```

```
        Socket you=null;
        while(true)
        { try{ server=new ServerSocket(4331);
              }
            catch(IOException e1) {System.out.println("正在监听");}
            try{ you=server.accept();
               }
            catch (IOException e)
               {System.out.println("正在等待客户");
               }
            if(you!=null)
               {new Server_thread(you).start();
               }
            else {continue;}
        }
    }
}
class Server_thread extends Thread
{  Socket socket;Connection Con=null;Statement Stmt=null;
   DataOutputStream out=null;DataInputStream  in=null;
   String s=null;
   Server_thread(Socket t)
      { socket=t;
        try {in=new DataInputStream(socket.getInputStream());
             out=new DataOutputStream(socket.getOutputStream());
            }
        catch (IOException e) {}
        try { Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            }
        catch(ClassNotFoundException e)
            {
            }
        try { Con=DriverManager.getConnection("jdbc:odbc:test","gxy","ookk");
              Stmt=Con.createStatement();
            }
        catch(SQLException ee) {}
      }
   public void run()
   {  while(true)
```

```
        {try{ s=in.readUTF();
              int n=0;
              ResultSet rs=
              Stmt.executeQuery("SELECT * FROM 表1 WHERE 单词 ="+"'"+s+"'" );
              while (rs.next())
                  { String 英语单词=rs.getString("单词");
                    if(s.equals(英语单词))
                        { out.writeUTF(rs.getString("解释")); n=1;break;
                        }
                     if(n==0) {out.writeUTF("没有此单词");}
                  }
               sleep(45);
             }
         catch(Exception ee)
             { break;
             }

       }
   }
}
```

习 题 二十三

1 编写一个简单图书查询程序

2 编写一个具有英汉,汉英查询功能的电子词典.

第二十四章 Java 与多媒体

24.1 在小程序中播放声音

用 java 可以编写播放.au,.aiff,.Wav,.Midi,.rfm 格式的音频.Au 格式是 Java 早期唯一支持的音频格式.要在小程序中播放声音,可以使用 Applet 的一个静态的方法 类方法

```
newAudioClip(URL url,String name)
```

或 Applet 类的实例方法

```
getAudioClip(Url url,String name)
```

根据参数 url 提供的地址,以及该处的声音文件 name.,获得一个可用于播放的音频对象 AudioClip 类型对象 .这个音频对象可以使用下列方法来处理声音文件.

play() 播放声音文件 name,

loop() 循环播放 name,

stop() 停止播放 name.

例子 1

```
import java.applet.*;import java.awt.*;
import java.awt.event.*;
public class Example24_1 extends Applet implements ActionListener
{  AudioClip clip;//声明一个音频对象
   Button button_play,button_loop,button_stop;
    public void init()
   {   clip=getAudioClip(getCodeBase(),"1.au");
      //根据程序所在的地址处的声音文件 1.au 创建音频对象,Applet 类的
     // getCodeBase() 方法可以获得小程序所在的 html 页面的 URL 地址.
     button_play=new Button("开始播放");button_loop=new Button("循环播放");
     button_stop=new Button("停止播放");button_play.addActionListener(this);
     button_stop.addActionListener(this);button_loop.addActionListener(this);
     add(button_play);add(button_loop);add(button_stop);
    }
    public void stop()
    {  clip.stop();//当离开此页面时停止播放.
    }
    public void actionPerformed(ActionEvent e)
```

```
    {  if(e.getSource()==button_play)
        { clip.play();}
       else if(e.getSource()==button_loop)
       { clip.loop();}
      if(e.getSource()==button_stop)
       { clip.stop();}
   }
}
```

24.2 在另一个线程中创建音频对象

在上述例子中我们是在小程序的初始化方法中创建音频对象的,如果声音文件较大或网络速度慢将影响小程序完成其它的初始化工作,因此我们应当在另外一个级别较底的线程中完成音频对象的创建工作,也就是后台载入声音文件.

如果在网络上有多个音频文件想让客户选择播放,我们可以用 **Choice** 类创建一个选择控件组件,将声音文件放在控件的选择列表中,当客户选择列表中一个项目后,我们就启动一个创建音频对象的线程.

另外,我们可以在超文本中使用若干个<**Param…**>标志把值传递到小程序中

下面是为运行下述例子 **2** 程序的 **html** 文件

```
<applet code=Example24_2.class width=200 height=200>
<Param name="1"  value ="祝酒歌:1.au">
<Param name="2"  value ="云的思念:2.au">
<Param name="3"  value ="祝你平安:3.au">
<Param name="4"  value ="难忘今宵:4.au">
<Param name="总数" value="4">
</applet>
```

例子 2 效果如图 24.1

```
import java.applet.*;import java.awt.*;import java.awt.event.*;
public class Example24_2 extends Applet implements ActionListener,Runnable,ItemListener
{  AudioClip clip;//声明一个音频对象.
   Choice choice;
   TextField text;
   Thread thread;
   String item=null;
   Button button_play,button_loop,button_stop;
   public void init()
  {  choice=new Choice();
```

```
    thread=new Thread(this);
    int N=Integer.parseInt(getParameter("总数"));
    for(int i=1;i<=N;i++)
      {  choice.add(getParameter(String.valueOf(i)));
      }
   button_play=new Button("开始播放");
   button_loop=new Button("循环播放");
   button_stop=new Button("停止播放");
   text=new TextField(12);
   button_play.addActionListener(this);
   button_stop.addActionListener(this);
   button_loop.addActionListener(this);
   choice.addItemListener(this);
   add(choice);
   add(button_play);add(button_loop);add(button_stop); add(text);
   button_play.setEnabled(false);
   button_loop.setEnabled(false);
 }
 public void itemStateChanged(ItemEvent e)
 { item=choice.getSelectedItem();
   int index=item.indexOf(":");
   item=item.substring(index+1).trim();
   if(!(thread.isAlive()))
    {  thread=new Thread(this);
    }
   try {
        thread.start();
       }
   catch(Exception exp)
       { text.setText("正在下载音频文件");
       }
 }
public void stop()
{  clip.stop();
}
public void actionPerformed(ActionEvent e)
{  if(e.getSource()==button_play)
     { clip.play();
     }
```

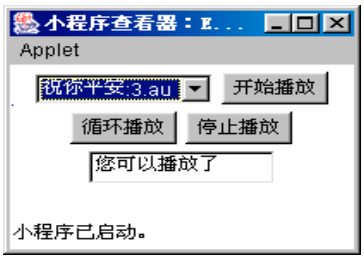


图 24.1　播放音

```
    else if(e.getSource()==button_loop)
      { clip.loop();
      }
    else if(e.getSource()==button_stop)
      { clip.stop();
        button_play.setEnabled(false);
        button_loop.setEnabled(false);
      }
  }
  public void run()
  {
    clip=getAudioClip(getCodeBase(),item);
    text.setText("请稍等...");
     if(clip!=null)
    {  button_play.setEnabled(true);
       button_loop.setEnabled(true);
       text.setText("您可以播放了");
    }
  }
}
```

24.3 Java 媒体框架(JMF)

Java 推出 JMF(java media Framework)之后标志着 Java 进入了多媒体时代.如果你想编写能播放视频的 Java 程序程序,必须到 Sun 公司的网站下载并安装 Jmf2.1(或更高版本).JMF 为我们提供编写多媒体程序所必须的包 javax.media.

建立一个播放视频的多媒体程序需经过下列主要步骤.

假设小程序所在的目录有一个视频文件 Bird.mpg.

1 创建一个播放器

```
try{  URL  url=new URL(getDocumentBase(),Bird.mpg);
        Player player=Manager.createPlayer(url);
    }
catch(IOException e){}
```

在上述代码中,首先根据小程序所在的目录下的视频文件 Bird.mpg 创建一个 URL 对象,然后用 javax.media 包中的 Player 类声明一个被称作播放器的对象 player,并使用 javax.media 包中的 Manager 类调用它的静态方法,并向这个方法传递参数 url,为媒体文件创建这个播放器 player.

2　向播放器注册控制监视器

```
player.addControllerListener(监视器)
```

因此创建监视器的类必须接口 ControllerListener,该接口中有一个方法

```
public void controllerUpdate(ControllerEvent  e).
```

3　让播放器对媒体进行预提取

```
player.prefetch();
```

播放器进行媒体预提取时,将不断获得媒体文件的有关信息,每当得到一个新的信息将导致 ControllerEvent 事件的发生　ControllerEven 类有几个重要的子类,如 RealizeCompleteEvent,PrefetchCompleteEvent 等　,并且监视器将调用方法 controllerUpdate(ControllerEvent　e)作出处理　比如,获得用于播放媒体文件的可视组件,控制组件等.方法 controllerUpdate(ControllerEvent　e)可能被调用多次.当在预提取中已获得足够的关于媒体文件的信息后将导致 PrefetchComlementEvent 事件发生,这时我们就可以通知播放器开始播放媒体文件.

4　启动播放器

```
player.start();
```

5　停止播放器

```
player.stop();
```

当停止播放器后,播放器应释放占有的资源

```
player.deallocate();
```

和播放音频文件的想法一样,我们应在另一个线程中进行媒体的预提取.

例子 3　效果如图 24.2

```
import java.applet.*;import java.awt.*;
import java.net.*;import java.awt.event.*;
import java.io.*;import javax.media.*;
public class Example24_3 extends Applet
         implements ControllerListener,Runnable,ItemListener
{  Player player;
   String str;
   Thread mythread;
   Choice choice;
   Component visualComponent,controlComponent,progressBar;
   String mediaFile;
   URL mediaURL,codeBase;
```

```
Frame frame;
public void init()
{ str="Music01.MPG";
  mythread=new Thread(this);
  choice=new Choice();
  choice.add("Music01.MPG");
  choice.add("Music02.avi");
  choice.add("Music03.avi");
  choice.addItemListener(this);
  codeBase=getDocumentBase();
  frame=new Frame("视频系统");
  frame.setSize(660,580);
  frame.addWindowListener(new WindowAdapter()
         {  public void windowClosing(WindowEvent e)
            {  if(player!=null)
                 { player.stop();player.deallocate();
                 }
              frame.setVisible(false);
              System.exit(0);
            }
         });
  add(choice);
}
public void stop()
{  if(player!=null)
   { player.stop();
   }
}
public synchronized void controllerUpdate(ControllerEvent event)
{   player.getDuration();
   if(event  instanceof  RealizeCompleteEvent)
     { if((visualComponent=player.getVisualComponent())!=null)
              frame.add("Center",visualComponent);
       if((controlComponent=player.getControlPanelComponent())!=null)
            if(visualComponent!=null)
                 frame.add("South",controlComponent);
            else
                 frame.add( "Center",controlComponent);
       frame.validate();
```

图 24.2　播放视频

```
        frame.pack();
      }
   else if(event  instanceof  PrefetchCompleteEvent)
      {  player.start();
      }
}
public void itemStateChanged(ItemEvent e)
{
   str=choice.getSelectedItem();
   if(player==null)
      {
      }
   else
      { player.stop();player.deallocate();
      }
   frame.removeAll();
   frame.setVisible(true);
   frame.setBounds(300,100,150,100);
   frame.validate();
   if(!(mythread.isAlive()))
      {  mythread=new Thread(this);
      }
   try{
         mythread.start();
      }
   catch(Exception ee)
      {
      }
}
public synchronized void run()
{  try{  mediaURL=new URL(codeBase,str);
         player=Manager.createPlayer(mediaURL);player.getDuration();
         if(player!=null)
          {  player.addControllerListener(this);
          }
        else
          System.out.println("failed to creat player for"+mediaURL);
    }
  catch(MalformedURLException e)
```

```
        {  System.out.println("URL for"+mediaFile+"is invalid");
        }
       catch(IOException e)
        {  System.out.println("URL for"+mediaFile+"is invalid");
        }
      catch(NoPlayerException e)
        { System.out.println("canot find a player for"+mediaURL);
        }
      if(player!=null)
        { player.prefetch();
        }
    }
  }
```

习题二十四

1．编写一个播放音乐的小应用程序,当客户选择某个音乐之后,程序在适当的位置显示一幅图象.

2　编写一个播放视频的小应用程序,要求视频文件的名字通过 html 文件传送给程序　见例子 5

第二十五章 Java Swing 基础

我们已经学习过了 java.awt 包中的 TextField,TextArea ,Button,Label,Choice,List 等类.这些类创建的对象被称为组件.当我们用这些类创建一个组件时,都有一个相应的本地组件在为它工作 称为它的同位体 .AWT 组件的设计原理是把与显示组件有关的许多工和处理组件事件的工作交给相应的本地组件.因此我们把有同位体的组件称为重量组件,基于重量组件的 GUI 设计有很多不足之处,比如我们的程序的外观在不同的平台上可能有所不同,而且重量组件的类型也不能满足 GUI 设计的需要,例如,不可能把一副图象添加到 AWT 按钮上或 AWT 标签上,因为 AWT 按钮或标签外观绘制是由本地的对等组件,即同位体来完成的,而同同位体可能是用 c++编写的,它的行为是不能被 java 扩展的.另外,使用 AWT 进行 GUI 设计可能会消耗大量的系统资源,这是因为每个建立一个 AWT 组件,就会产生一个对等的同位体组件,如果你在一个 Frame 窗体中填加 100 个 Button 组件,就大量的消耗的系统的资源,可能导致 java 程序无法正确运行.如果把程序比做一个剧本,那么程序中的一个"按钮"组件就好比剧本中的一个人物 "王子",如果是重组件,就有同位体,基于 AWT 的 GUI 设计就好比用舞台话剧的形式来演出剧本,"王子"就有"同位体",也就是说,当在北京演出时,就在北京找一个人来扮演王子,当在伦敦演出时,就在伦敦找一个人来扮演王子,虽然二者表达的台词剧情完全相同,但二者的外观不尽相同,如果你的剧本需要一万个士兵,那就很消耗本地的人力资源,而且用话剧形式来演出剧本,人物的行为也受到很多限制.

javax.swing 包为我们提供了更加丰富的,功能强大的组件,称为 swing 组件,其中大部分组件是轻量组件,没有同位体.swing 组件的轻组件在设计上和 AWT 完全不同,轻组件把与显示组件有关的许多工和处理组件事件的工作交给相应的 UI 代表来完成,这些 UI 代表是用 Java 语言编写的类,这些类被增加到 java 的运行环境中,因此组件的外观不依赖平台,不仅在不同平台上的外观是相同的,而且较重量组件而言有更高的性能.如果你的 Java 编程环境或 java 运行环境低于 1.2 版本,就不能使用 swing 组件或运行含有 swing 组件的程序,例如,IE 5.5 就不能运行含有 swing 组件的 java applet 小应用程序,因为 IE5.5 内置的 java 运行环境版本是 1.1.

javax.swing 包中有四个最重要的类:JApplet,JFrame,JDialog,JComponent.

JComponent 类的子类都是轻组件,而 JFrame,JApplet,Jdialog 都是重组件,即有同位体的组件,这样,窗口(JFrame),对话框 JDialog ,小应用程序 JApplet 可以和操作系统交互信息,轻组件必须在这些重量容器中绘制自己.javax.swing 包中有四个最重要的类 Japplet,JFrame,Jdialog,JComponent.

25.1 几个重要的类

1 JComponent 类

JComponent 类是所有轻量组件的父类 就象 Component 类是所有重量组件的父类 .JComonent 子类如表 25.1.

表 25.1 Jcomonent 类的一些重要子类

JButton	负责创建按钮对象.而且可以创建带图标按钮
JComboBox	负责创建组合框对象,和重量组件 Choice 类似 能在一个显示区域内显示出多选项.
JCheckBox	负责创建复选框对象,和重组件 CheckBox 类似
JFileChooser	负责创建文件选择器
JInternalFrame	负责创建内部窗体
JLable	负责创建标签,比重组件的功能更强大
JMenu	负责创建菜单对象
JMenuBar	负责创建菜单条对象
JMenuItem	负责创建菜单项对象
JPanel	负责创建面板对象
JPasswordField	负责创建口令文本框对象
JPopupMenu	负责创建弹出式菜单
JProgressBar	负责创建进程条
JRadioButton	负责创建单选按钮
JScrollBar	负责创建滚动条
JScrollPane	负责创建滚动窗格
JSlider	负责创建滑动条
JSplitPane	负责创建拆分窗格
JTable	负责创建表格
JTextArea	负责创建文本区
JTexPane	负责创建文本窗格
JToolBar	负责创建工具条
JToolTip	负责创建工具提示对象
JTree	负责创建树对象

注 JComponent 类是 java.awt 包中容器类 Container 类的子类,因此所有的轻量组件也都是容器.

2 JFrame 类

javax.swing 包中的 JFrame 类是 java.awt 包中 Frame 类的子类.因此 JFrame 类其子类创建的对象是窗体.由于 Frame 是重量容器,因此 JFrame 类或子类创建的对象 窗体也是重量容器.JFrame 类除了父类提供的功能外,还具有许多新的特性

1 称 JFrame 类或它的子类创建的窗体是 swing 窗体.

2 不可以把组件直接填加到 swing 窗体中.

3 swing 窗体含有一个称为内容面板的容器,应当把组件填加到内容面板中 内容面板也是重量容器 .

4 不能为 swing 窗体设置布局,而应当为 swing 窗体的内容面板设置布局.内容面板的默认布局是 BorderLayout 布局.

5 swing 窗体通过调用方法 getContentPane(),得到它的内容面板.

例子 1

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

```
public class Example25_1
{  public static void main(String args[])
    {  JButton button=new JButton("轻组件按钮");
       JTextArea text=new JTextArea("轻组件",20,20);
       JFrame  jframe=new JFrame("根窗体");
       jframe.setSize(200,300);jframe.setBackground(Color.blue);
       jframe.setVisible(true);jframe.pack();
       jframe.addWindowListener(new WindowAdapter()
             {public void windowClosing(WindowEvent e)
                 {System.exit(0);}
             });
        Container contentpane=jframe.getContentPane();//获得内容面板.
        contentpane.add(button,BorderLayout.SOUTH); //向内容面板加入组件.
        contentpane.add(text,BorderLayout.CENTER);
        jframe.pack();
  }
}
```

注 我们可以这样来理解 swing 窗体 swing 窗体与普通的窗体相比是一类特殊的窗体.另外,在 swing 窗体的内容面板中尽量不要既有重量组件又有轻量组件,最好只使用轻量组件.否则可能会出现预想不到的问题.

例子 2

```
import javax.swing.*;import java.awt.*;import java.awt.event.*;
class Mywindow extends JFrame
{  JButton button;JTextArea text;
   Mywindow()
   { setSize(200,400);setVisible(true);
     Container con=getContentPane(); con.setLayout(new FlowLayout());
     button=new JButton("ok");text=new JTextArea(10,20);
     con.add(button);con.add(text);pack();
     addWindowListener(new WindowAdapter()
          {public void windowClosing(WindowEvent e)
             {System.exit(0);}
          });
   }
}
public class Example25_2
{  public static void main(String args[])
   { Mywindow win=new Mywindow();win.pack();
```

```
    }
}
```

3 JApplet 类

JApplet 类也是用来建立 java 小应用程序用的.JApplet 是 Javax.swing 包中的类,它还是 java.applet 包中的 Applet 类的子类,因此 JApplet 对象也是一个重量容器.用 Applet 和 JApplet 建立的小应用程序有许多不同之处.

当用 JApplet 来创建小应用程序时,有以下规定

1　不可以把组件直接添加到小程序容器中.小程序容器也含有一个称为内容面板的容器,应当把组件添加到内容面板中　这个内容面板是重量容器　.

2　不能为小程序容器设置布局,而应当为小程序容器的内容面板设置布局.内容面板的默认布局是 BorderLayout 布局.

3　小程序容器通过调用方法　getContentPane(),得到内容面板.

例子 3

```
import javax.swing.*;import java.awt.BorderLayout;
public class Example25_3 extends JApplet
{   JButton  button;   JTextArea  text;
    public void init()
   { button=new JButton("确定");text=new JTextArea();
     getContentPane().add(text,BorderLayout.CENTER); //小程序容器得到内容面板.
     getContentPane().add(button,BorderLayout.WEST);//并向内容面板中添加组件.
   }
}
```

4 JDialog 类

JDialog 是 java.awt 包中 Dialog 类的子类.JDialog 类或子类创建的对象是也是重量容器,该对象必须依附一个 JFrame 对象.JDialog 的一个构造方法是

JDialog(JFram　f,String s)

对这种类型的对话框　JDialog 对象　也有以下规定

1　不可以把组件直接添加到对话框中.JDialog 型对话框也含有一个称为内容面板的容器,应当把组件添加到内容面板中　这个内容面板是重量容器　.

2　不能为对话框设置布局,而应当为对话框的内容面板设置布局.内容面板的默认布局是 BorderLayout 布局.

3　对话框通过调用方法　getContentPane(),得到内容面板.

例子 4

```
import javax.swing.*;import java.awt.*;import java.awt.event.*;
class Dwindow extends JFrame //建立根窗体用的类.
{  JButton button1,button2;
```

```
   Dwindow(String s)
   {  super(s);
      Container con=getContentPane();
      button1=new JButton("打开"); button2=new JButton("关闭");
      con.add(button1);con.add(button2);pack();
      setVisible(true);
      addWindowListener(new WindowAdapter()
        {  public void windowClosing(WindowEvent e)
            {System.exit(0);}
        });
   }
}
class Mydialog extends JDialog //建立对话框类.
{  JButton  button1,button2;
   Mydialog(JFrame F,String s)   //构造方法.
   { super(F,s);
     button1=new JButton("open");     button2=new JButton("close");
     setSize(90,90);setVisible(true);setModal(false);
     Container con=getContentPane();con.setLayout(new FlowLayout());
     con.add(button1);con.add(button2);
     addWindowListener(new WindowAdapter()
       {  public void windowClosing(WindowEvent e)
          {System.exit(0);}});
   }
}
public class Example25_4 extends JApplet
{  Dwindow window;  Mydialog dialog;  JButton button;
   public void init()
   {  window=new Dwindow("带对话框窗口");//创建窗口.
      dialog=new Mydialog(window,"我是对话框");  //创建依赖于窗口 window 的对话框.
      button=new JButton("ok");  getContentPane().add(button);
   }
}
```

5　事件处理

重量组件的事件处理模式　授权处理模式,仍然适合于 轻组件的事件处理,即事件源,监视器和接口.事件由事件源产生,为了能处理相应类型的事件,事件源必须使用相应的方法获得监视器.当事件发生时,监视器使用接口中的方法对事件进行处理.

例子 5

```
import javax.swing.*;import java.awt.*;import java.awt.event.*;
class Myframe extends JFrame implements ActionListener
{  JButton button;JTextArea text;
   Myframe()
   {  setSize(200,400);setVisible(true);
      Container con=getContentPane();
      con.setLayout(new FlowLayout());
      button=new JButton("ok");text=new JTextArea(10,20);
      con.add(button);con.add(text);
      button.addActionListener(this);
      addWindowListener(new WindowAdapter()
       { public void windowClosing(WindowEvent e)
         {System.exit(0);}});
   }
   public void actionPerformed(ActionEvent e)
   {  if(e.getSource()==button)
      text.setText("i am a boy,and you?");
   }
}
public class Example25_5
{  public static void main(String args[])
   {  Myframe fr=new Myframe();fr.pack();
   }
}
```

总结　称 JFrame, JApplet, Jdialog,是 swing 的底层容器,都是重量容器.我们不能把组件直接添加到底层容器中,而应当添加到它们的内容面板中　我们也不能为底层容器设置布局,而应当为它的内容面板设置布局,内容面板的默认布局是 BorderLayout 布局.

25.2 中间容器

我们已经知道轻组件都是容器,但仍有一些经常用来添加组件的轻容器,相对于底层重量容器而言,我们习惯上称这些轻容器为中间容器.

1　JPanel 面板

我们会经常使用 JPanel 创建一个面板,再向这个面板添加组件,然后把这个面板添加到底层容器或其他中间容器中.JPpanel 面板的默认布局是 FlowLayout 布局.JPanel 类的两个构造方法

JPanel(),JPanel(布局对象),

另外 JPanel 还能实现画布的功能.

在下面的例子中有两个面板,其中一个具有画布的功能.

例子 6

```
 import javax.swing.*; import java.awt.*;
class Mycanvas extends JPanel
{  public void paintComponent(Graphics g)
   { super.paintComponent(g);
     g.setColor(Color.red);  g.drawString("a Jpanel used as canvas",50,50);
   }
}
public class Example25_6 extends JApplet
{  Mycanvas canvas; JPanel panel;JButton button;
   public void init()
   {  canvas=new Mycanvas();panel=new JPanel();button=new JButton("ok");
      panel.add(button); Container con=getContentPane();
      con.add(panel,BorderLayout.NORTH); con.add(canvas,BorderLayout.CENTER);
   }
}
```

2　滚动窗口 JScrollPane

我们可以把一个组件放到一个滚动窗口中,然后通过滚动条来观察这些组件.例如,**JTextArea** 不自带滚动条　这一点与重量组件 **TextArea** 不同　,因此我们就需要把文本区放到一个滚动窗口中.

JScorollPane 两个构造方法　**JScorollPane(),JScorollPane(component　c).**

例子 7

```
import javax.swing.*;import java.awt.*;import java.awt.event.*;
class Mywindow extends JFrame
{  JButton button;JTextArea text;JScrollPane scroll;
   Mywindow()
   {  setSize(200,400);setVisible(true);
      Container con=getContentPane();
      button=new JButton("ok");text=new JTextArea(10,20);
      scroll=new JScrollPane(text);
      con.add(button,BorderLayout.SOUTH);con.add(scroll,BorderLayout.CENTER);
      addWindowListener(new WindowAdapter()
      {public void windowClosing(WindowEvent e)
        {System.exit(0);}});
   }
}
public class Example25_7
```

```
{  public static void main(String args[])
   {  Mywindow win=new Mywindow();win.pack();
   }
}
```

3. 拆分窗口 JSplitPane

顾名思义,拆分窗口就是被分成两部分的窗口.拆分窗口有两种类型　水平拆分和垂直拆分.水平拆分窗口用一条拆分线把窗口分成左右两部分,左面放一个组件,右面放一个组件,拆分线可以水平移动.垂直拆分窗口用一条拆分线把窗口分成上下两部分,上面放一个组件,下面放一个组件,拆分线可以垂直移动.

JSplitPane 的两个常用的构造方法

```
JSplitPane(int  a, Component b ,Component  c)
```

参数 a 取 JSplitPane 的静态常量 HORIZONTAL_SPLIT 或 VERTICAL_SPLIT,以决定是水平还是垂直拆分.后两个参数决定要放置的组件.当拆分线移动时,组件不是连续变化的.

```
JSplitPane(int  a, boolean  b,Component c ,Component  d)
```

参数 a 取 JSplitPane 的静态常量 HORIZONTAL_SPLIT 或 VERTICAL_SPLIT,以决定是水平还是垂直拆分.参数 b 决定当拆分线移动时,组件是否连续变化　true 是连续　,后两个参数决定要放置的组件.

例子 8　效果如图 25.1

```
import javax.swing.*;import java.awt.*;import java.awt.event.*;
class Mywindow extends JFrame
{  JButton button1,button2;JTextArea text;JSplitPane split_one,split_two;
   Mywindow()
   {  setSize(200,400);setVisible(true); Container con=getContentPane();
      button1=new JButton("ok"); button2=new JButton("No");
      text=new JTextArea("I love you,java",10,20);
      split_one=new JSplitPane(JSplitPane.VERTICAL_SPLIT,button1,button2);
      split_two=new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,true,split_one,text);
      con.add(split_two,BorderLayout.CENTER);
      addWindowListener(new WindowAdapter()
        {public void windowClosing(WindowEvent e)
        {System.exit(0);}});
   }
}
public class Example25_8
{  public static void main(String args[])
```

```
    { Mywindow win=new Mywindow();win.pack();
    }
}
```

4　内部窗体 JInternalFrame

内部窗体的一个常用的构造方法是

```
JInternalFrame(String title,boolean resizable,boolean closable,boolean max ,boolean min),
```

第一个参数是窗体的名字,接下来的参数分别决定窗体能否调整大小,能否关闭,能否最大化,能否最小化.对于内部窗体需注意下列两点

1　内部窗体和前面讲的中间容器有所不同,我们不能直接把组件加到内部窗体中,而只能加到它的内容面板中.内部窗体和 JFrame 窗体一样,可以通过 getContentPane()得到它的内容面板,内容面板的默认布局也是 BorderLayout 布局.

2　为了能显示内部窗体,必须把内部窗体先添加到一个容器中,这个容器是 JDesktopPane,该容器是专门为内部窗体服务的.

3　需调用 setVisible 为内部窗体设置可见性,默认是不可见的.内部窗体仍然需要调用 setSize()或 setBounds 设置大小.

例子 9　效果如图 25.2

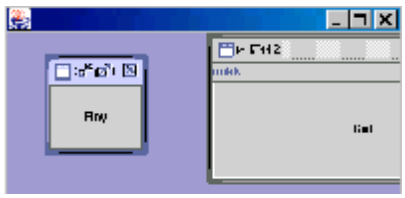


图 25.2　　内部窗

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class Mywindow extends JFrame
{  JButton button1,button2;
   JInternalFrame interframe_1,
                  interframe_2;
   Mywindow()
   { setSize(200,200); setVisible(true);
     Container con=getContentPane();
     con.setLayout(new GridLayout(1,2));
     button1=new JButton("Boy");  button2=new JButton("Girl");
     interframe_1=
     new JInternalFrame("内窗体 1",true,true,true,true);
     interframe_1.setSize(100,100);interframe_1.setVisible(true);
     interframe_1.getContentPane().add(button1);
     JDesktopPane  desk1=new JDesktopPane();
     desk1.add(interframe_1);
     interframe_2=new JInternalFrame("内窗体 2",true,true,true,true);
     interframe_2.setSize(300,150);interframe_2.setVisible(true);
     interframe_2.getContentPane().add(button2,BorderLayout.CENTER);
```

```
        interframe_2.getContentPane().add(new JLabel("ookk"),BorderLayout.NORTH);
        JDesktopPane  desk2=new JDesktopPane();
        desk2.add(interframe_2);
        con.add(desk1);con.add(desk2);
        addWindowListener(new WindowAdapter()
         {public void windowClosing(WindowEvent e)
           {System.exit(0);}});
    }
}
public class Exam25_9
{ public static void main(String args[])
  { Mywindow win=new Mywindow();win.pack();
  }
}
```

25.3 各种组件

1. 按钮(JButton)

JButton 类负责创建按钮对象,与重量组件按钮 **Button** 相比,**JBtton** 按钮具有更加丰富的外观,例如我们可以创建带图标的按钮.

1 构造方法

public JButton() 创建没有图标,也没有名字的按钮.

public JButton(Icon icon) 创建带图标的按钮,图标由参数 **icon** 指定,该图标是按钮在各种状态下的默认图标.

public JButton(String text)创建有名字的按钮,名字由参数 **text** 指定.

public JButton(String text,Icon icon) 创建一个既有名字又有图标的按钮,该图标是按钮在各种状态下的默认图标,名字和图标分别由参数 **text** 和 **icon** 指定.

2 实例方法

下面列出的常用实例方法,大部分都是从 **AbstractButton** 类继承下来的.

public void setText(String text) 按钮对象调用该方法可以重新设置当前按钮的名字,名字由参数 **text** 指定.

public String getText() 按钮对象调用该方法可以获取当前按钮上的名字.

public void setIcon(Icon icon) 按钮对象调用该方法可以重新设置当前按钮上的图标,称为按钮的默认图标,图标由参数 **icon** 指定.如果按钮没有调用其它方法设置某种状态下的图标,该图标就是按钮在该状态下的图标.

public Icon getIcon() 按钮对象调用该方法可以获取当前按钮上的默认图标.

public void setPressedIcon(Icon pressedIcon) 按钮对象调用该方法设置按钮在被按下状态时的图标.

public Icon getPressedIcon() 按钮对象调用该方法获取按钮曾用 setPressedIcon 方法设置过的图标图标.

public void setRolloverIcon(Icon rolloverIcon) 按钮对象调用该方法设置鼠标指向按钮上方时的图标.

public Icon getRolloverIcon() 按钮对象调用该方法获取按钮曾用 setRolloverIcon t 方法设置过的图标.

public void setDisabledIcon (Icon disabledIcon) 按钮对象调用该方法设置按钮不能激活状态下的图标

public Icon getDisabledIcon() 按钮对象调用该方法获取按钮曾用 setDisabledIcon 方法设置过的图标.

public void setHorizontalTextPosition(int textPosition) 设置按钮上的名字相对按钮上图标的水平位置,参数 textPosition 的有效值是

AbstractButton.LEFT,AbstractButton.CENTERT,AbstractButton.RIGHT

public void setVerticalTextPosition(int textPosition) 设置按钮上的名字相对按钮上图标的垂直位置,参数 textPosition 的有效值是

AbstractButton.TOP,AbstractButton.CENTERT,AbstractButton.BOTTOM

如果 setHorizontalTextPosition 方法中参数的值是 LEFT,setVerticalTextPosition 方法中参数的取值是 TOP,那么,按钮的名字就在图标的左上角,如果 setVerticalTextPosition 方法中参数的取值是 CENTER,那么按钮的名字就在图标的左面,当两个方法的参数的值都是 CENTER 时,按钮的名字在图标的正中间.

public void setMnemonic(char mnemonic) 按钮对象调用该方法可以设置按钮的键盘激活方式,mnemonic 的有效值是 'a'~'z'

如果按钮用此方法设置了键盘激活方式,比如,参数 mnemonic 取值'x',那么当在键盘操作"ALT+o"就可激活按钮.

public void setMnemonic(int mnemonic) 按钮对象调用该方法可以设置按钮的键盘激活方式,mnemonic 的取值是 KeyEvent 类中的静态常量,有效值是

KeyEvent.VK_A~KeyEvent.VK_Z

public void addActionListener(ActionListener) 按钮对象调用该方法可以获得一个监视 ActionEvent 类型事件的监视器.

下面的例子 10 中,设置了按钮的图标,并设置的按钮名字和图标的相对位置.

例子 10 效果如图 25.3

```
import javax.swing.*; import java.awt.*;
import java.awt.event.*;
class MyWin extends JFrame
{ JButton b1,b2,b3;
  public  MyWin()
  { setBounds(100,100,300,200); setVisible(true);
```

```
    addWindowListener(new WindowAdapter()
      {  public void windowClosing(WindowEvent e)
         {  System.exit(0);
         }
      });
    b1=new JButton("按钮 1",new ImageIcon("f:/2000/a1.gif"));
    b2=new JButton("按钮 2",new ImageIcon("f:/2000/a2.gif"));
    b3=new JButton("按钮 3",new ImageIcon("f:/2000/a3.gif"));
    b1.setRolloverIcon(b2.getIcon());
    b2.setRolloverIcon(b3.getIcon());
    b3.setRolloverIcon(b1.getIcon());
    b1.setHorizontalTextPosition(AbstractButton.LEFT);
    b1.setVerticalTextPosition(AbstractButton.TOP);
    b2.setHorizontalTextPosition(AbstractButton.RIGHT);
    b2.setVerticalTextPosition(AbstractButton.BOTTOM);
    b3.setHorizontalTextPosition(AbstractButton.CENTER);
    b3.setVerticalTextPosition(AbstractButton.CENTER);
    Container con=getContentPane();
    con.setLayout(new FlowLayout());
    con.add(b1);con.add(b2); con.add(b3);
    con.validate();
  }
}
public class Example25_10
{  public static void main(String args[])
  {  new MyWin();
  }
}
```

注　为了创建一个图标,我们可以用图标类 Icon 声明一个图标,然后使用 ImageIcon 类创建一个图标,如

```
Icon  icon=new ImageIcon("dog.gif");
```

2 .标签(JLabel)

JLabel 类负责创建标签对象,与重量组件标签 Label 相比,JLabel 标签具有更加丰富的外观,例如我们可以创建带图标的标签.

1　JLabel 类的构造方法.

public　JLabel ()　　创建没有名字的标签.

public JLabel (String　s)　　创建名字是 s 的标签,s 在标签中靠左对齐.

public JLabel (String　s, int aligment)　　参数 aligment 决定标签中的文字在标签中

的水平对齐方式.Aligment 的取值是 JLabel.CENTER, JLabel.LEFT,JLabel.RIGHT.

public JLabel (Icon icon)　　创建具有图标 icon 的标签,icon 在标签中靠左对齐.

public JLabel (String s,Icon icon,int aligment)　　创建名字是 s,具有图标 icon 的标签,参数 aligment 决定标签中的文字和图标做为一个整体在标签中的水平对齐方式　名字总是在图标的右面　.

2　JLabel 类的实例方法　很丰富,大约 30 个实例方法,我们只列出常用的几个　.

String　getText()　获取标签的名字.

void　setText(String s)　设置标签的名字是 s.

Icon　getIcon()　获取标签的图标.

void　setIcon(Icon icon)　设置标签的图标是 icon.

void　setHorizontalTextPosition(int　a)　参数 a 确定名字相对与标签上的图标的位置,a 的取值是　JLabel.LEFT, JLabel.RIGHT.

void　setVerticalTextPosition(int　a)　参数 a 确定名字相对与 JLabel 上的图标的位置, 参数 a 取值是　JLabel.BOTTOM,JLabel.TOP.

例子 11

```
import javax.swing.*;import java.awt.BorderLayout;
import java.awt.event.*;import java.awt.*;
public class Example25_11 extends JApplet implements ActionListener
{  JLabel label_1,label_2;JButton button;JTextArea text;
   public void init()
   { button=new JButton("确定");text=new JTextArea();
     Icon icon=new ImageIcon("tom.jpg");
     label_1=new JLabel("标签 1",icon,JLabel.CENTER);
     label_2=new JLabel("标签 2");
     getContentPane().add(text,BorderLayout.CENTER);
     getContentPane().add(button,BorderLayout.WEST);
     getContentPane().add(label_1,BorderLayout.NORTH);
     getContentPane().add(label_2,BorderLayout.SOUTH);
     button.addActionListener(this);
   }
   public void actionPerformed(ActionEvent e)
   { button.setIcon(label_1.getIcon());
     label_1.setHorizontalTextPosition(JLabel.LEFT);
   }
 }
```

3　复选框(JCheckBox)

JCheckBox 类负责创建复选框对象.复选框的样子我们已经很熟悉,通常是一个小框,当

你选择某个选择框后,里面就有了个小对号.与重量组件复选框 CheckBox 相比,JCheckBox 复选框的名字不仅可以是字符串,而且它的样子可以是一个图标.

1 JCheckBox 类的构造方法.

public JCheckBox () 创建没有名字的复选框.

public JCheckBox (String s) 创建名字是 s,s 在复选框的右面.

public JCheckBox (String s ,boolean b) 名字是 s,b 决定选择框的初始状态,true 是选中.默认值是 false.

public JCheckBox (Icon icon) 创建形状是图标 icon 的复选框.

public JCheckBox (Icon icon,boolean b) 创建形状是图标 icon 的复选框, b 决定选择框的初始状态.

public JCheckBox (String s,Icon icon) 创建名字是 s,具有形状 icon 的复选框.

public JCheckBox (String s,Icon icon,boolean b) 创建名字是 s,具有形状 icon 的复选框,b 决定选择框的初始状态.

2 JCheckBox 类的实例方法.

复选框的常用实例方法,都是从 AbstractButton 类继承下来的,其中,setSelectedIcon 方法可能经常被使用,因为如果不明显的设置选中状态时的图标,用户很难知道复选框是否被选中.另外,isSelected 方法也经常被使用,因为程序可能需要根据复选框的状态来作出反应.

在下面的例子 12 中,我们用 JCheckBox 组件来代表一个候选人.当单击按钮时,如果复选框组件是选中状态时,给候选人增加一票,同时,文本区显示每个候选人的得票数.

例子 12 效果如图 25.4

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;
class 候选人 extends JCheckBox
{  int 得票数=0;
   候选人(String name,Icon icon)
   {  super(name,icon);
   }
  public int 获取得票数()
   {  return 得票数;
   }
  public void 增加票数()
   {  得票数++;
   }
}

class MyWin extends JFrame implements ActionListener
```

图 25.4 复选框

```
{    Box baseBox,boxH,boxV;
  JTextArea text;
  JButton button;
  候选人 候选人 1, 候选人 2, 候选人 3;
  public   MyWin()
  {   setBounds(100,100,300,200);
     setVisible(true);
     addWindowListener(new WindowAdapter()
        { public void windowClosing(WindowEvent e)
            {   System.exit(0);
            }
        });
     baseBox=Box.createHorizontalBox();
     boxH=Box.createHorizontalBox();
     boxV=Box.createVerticalBox();
     候选人 1=new 候选人("张小兵",new ImageIcon("a1.gif"));
     候选人 2=new 候选人("李大营",new ImageIcon("a2.gif"));
     候选人 3=new 候选人("王中堂",new ImageIcon("a3.gif"));
     候选人 1.setSelectedIcon(new ImageIcon("b1.gif"));
     候选人 2.setSelectedIcon(new ImageIcon("b2.gif"));
     候选人 3.setSelectedIcon(new ImageIcon("b3.gif"));
     boxH.add(候选人 1);   boxH.add(候选人 2);   boxH.add(候选人 3);
     text=new JTextArea();
     button=new JButton("显示得票数");
     button.addActionListener(this);
     boxV.add(text);   boxV.add(button);   baseBox.add(boxH);
     baseBox.add(boxV);
     Container con=getContentPane();
     con.setLayout(new FlowLayout());
     con.add(baseBox);
     con.validate();
  }
 public void actionPerformed(ActionEvent e)
  {    text.setText(null);
   if(候选人 1.isSelected())
      {      候选人 1.增加票数();
      }
   if(候选人 2.isSelected())
      {   候选人 2.增加票数();
```

```
        }
    if(候选人 3.isSelected())
        {  候选人 3.增加票数();
        }
    text.append("\n"+候选人 1.getText()+":"+候选人 1.获取得票数());
    text.append("\n"+候选人 2.getText()+":"+候选人 2.获取得票数());
    text.append("\n"+候选人 3.getText()+":"+候选人 3.获取得票数());
    候选人 1.setSelected(false);    候选人 2.setSelected(false);
    候选人 3.setSelected(false);
  }
}
public class Example25_12
{  public static void main(String args[])
  {  new MyWin();
  }
}
```

4. 单选按钮(JRadioButton)

单选按钮和复选框很类似,所不同的是　在若干个复选框中我们可以同时选中多个,而一组单选按钮同一时刻只能有一个被选中.当创建了若干个单选按钮后,应使用 ButtonGroup 再创建一个对象,然后利用这个对象把这若干个单选按钮归组.归到同一组的单选按钮每一时刻只能选一.单选按钮上可发生 ItemEvent 事件.

例子 13　效果如图 25.5

```
import javax.swing.*;
import java.awt.*;import java.awt.event.*;
class Mywindow extends JFrame implements ItemListener
{  JRadioButton button1,button2,button3;ButtonGroup fruit;
   JLabel label ;JScrollPane scroll;JPanel panel;JSplitPane split;
   Mywindow()
   {  setSize(200,400);setVisible(true);
      Container con=getContentPane();
      fruit=new ButtonGroup();
      button1=new JRadioButton("苹果");
      fruit.add(button1);
      button2=new JRadioButton("香蕉");
      fruit.add(button2);
      button3=new JRadioButton("西瓜");
      fruit.add(button3);
```

图 25.5　单选按钮

```
        label=new JLabel();panel=new JPanel();
        scroll=new JScrollPane(label);
        panel.add(button1);panel.add(button2);panel.add(button3);
        split=new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,true,panel, scroll);
        con.add(split);
        button1.addItemListener(this);button2.addItemListener(this);
        button3.addItemListener(this);
        addWindowListener(new WindowAdapter()
         {public void windowClosing(WindowEvent e)
           {System.exit(0);}});
    }
    public void itemStateChanged(ItemEvent e)
    {if(e.getItemSelectable()==button1)
     {label.setIcon(new ImageIcon("a.jpg")); }
    else if(e.getItemSelectable()==button2)
     {label.setIcon(new ImageIcon("b.jpg")); }
    else if(e.getItemSelectable()==button3)
     {label.setIcon(new ImageIcon("c.jpg")); }
    }
}
public class Example25_13
{ public static void main(String args[])
  { Mywindow win=new Mywindow();win.pack();
  }
}
```

5　选择框(JComboBox)

我们对选择框的形状很熟悉,是一个条状的显示区,并且在显示区的右端有一个小按钮,当单击该按钮时,可以打开一个列表.我们可以选择列表中的项目,并使之显示在显示区中.当列表中的选项较多时,下拉列表会自动出现滚动条.JComboBox 和重量组件 Choice 很类似,所不同的是 JComboBox 选择框可以被设置成可编辑的　默认是不可编辑的　,既用户可以在选择框的显示区里输入文本,然后按回车键,该文本就会被加入到下拉列表中,该类的很多方法和 java.awt 包中的 Choice 类相同.

下面这个例子有两个选择框,第一个是不可编辑的,第二个是可编辑的.当我们在第二个选择框的列表中输入某个网址之后,就会连接到另外一个网页　参见二十一章的例子 1　.

例子 14

```
import javax.swing.*;import java.awt.*;import java.awt.event.*; import java.net.*;
public class Example25_14 extends JApplet implements ItemListener
```

```
{  JComboBox choice1,choice2; JSplitPane split1,split2;
   JLabel label; URL url;
   public void init()
   { Container con=getContentPane(); String[] s={"苹果", "香蕉" ,"西瓜"};
    choice1=new JComboBox(s);choice2=new JComboBox();
    label=new JLabel();choice2.setEditable(true);
    split1=newJSplitPane(JSplitPane.HORIZONTAL_SPLIT,true,choice1,choice2);
    split2=new JSplitPane(JSplitPane.VERTICAL_SPLIT,true,split1,label);
    choice1.addItemListener(this);choice2.addItemListener(this);con.add(split2);
   }
   public void itemStateChanged(ItemEvent e)
   { if(e.getItemSelectable()==choice1)
        { if(choice1.getSelectedIndex()==0)
         { label.setIcon(new ImageIcon("a.jpg"));}
         else if(choice1.getSelectedIndex()==1)
         { label.setIcon(new ImageIcon("b.jpg")); }
         else if(choice1.getSelectedIndex()==2)
         {label.setIcon(new ImageIcon("c.jpg"));}
       }
    else if(e.getItemSelectable()==choice2)
       { try{url=new URL((String)choice2.getSelectedItem());
                label.setText("你正在连接到  "+choice2.getSelectedItem());
            }
         catch(MalformedURLException g)
           { label.setText("不正确的URL:"+url); }
        getAppletContext().showDocument(url);
       }
   }
}
```

6. 文本框　JTextField　,密码框 JPasswordField,文本区　JTextArea

JTextField 文本框,JTextArea 文本区和重组件的文本框,文本区类似,分别用于显示单行文本和多行文本.它们的实例方法也和重量组件的文本框和文本区基本相同.例如,可以使用 setText(String)设置文本,getText()获取文本,文本区还可以使用 append(String)向文本区追加文本,口令框可以使用 setEchoChar(char c)设置回显字符等,需注意的是 JPasswordField 有一个特殊的方法　char[]　getPassword().另外,与重量组件的文本区不同的是,JTextArea 文本区没有滚动条,所以应将文本区放入一个滚动窗口中　JScorllPane　.JPasswordField 和 JTextField 组件上都可以发生 ActionEvent 事件　输入文本后按 Enter 回车　.与重组件 TextArea 不同的是,JTextArea 上不再有 TextEvent 事

件.另外,JTextArea 可以使用 setLineWrap(boolean b)方法决定输入的文本能否在文本区的右边界自动换行 还可以使用方法 setWrapStyleWord(boolean b)决定是以单词为界 b 取 true 时 或以字符为界(b 取 false 时)进行换行.

与重组件 TextField 不同的是,JTextField 有一个如下的构造方法

```
JTextField(Document  document ,String s,int columns )
```

该构造方法用指定的文档 document 创建一个文本框.其它构造方法创建的文本框的默认文档类型是 PlainDocumnet 的一个实例,这种模型允许在文本框里任意地输入,删除字符.PlainDocumnet 类有如下两个重要的方法

```
insertString(int offset  String s , AttributeSet a)
```

在位置 offset 处插入一个具有指定属性的字符串 s,当在文本框里进行字符输入操作时,insertString 方法会自动被调用执行,offset 取值为文本框中输入光标的位置 s 为键盘键入的字符 a 被初始化为文本框的字符属性 文本框字符的默认属性是黑色 5 号字 .

```
remove(int offset  int length)
```

从位置 offset 处移去文本框中一段指定长度的内容 s.当在文本框里进行字符删除时,remove 方法会自动被调用执行,offset 取值文本框中输入光标的位置 length 初始化为 1.

我们可以通过上述构造方法或 JTextField 的实例方法

```
setDocutment(PlainDocumnet  d);
```

改变文本框的文档类型.例如,有时可能要求在文本框里只能输入数字等,这时,我们可以扩展 PlainDocumnet,重写父类 PlainDocumnet 中的方法,得到符合要求的子类.

在下面的例子中,我们在子类 DigitDucumnet 中重写了父类的 insertString 方法,使得该文档类型的文本框只能输入数字 包括小数点

例子 15

```
import javax.swing.*;
import javax.swing.text.*;
import java.awt.*;
class DigitDocumnet extends PlainDocument
{ public void insertString(int offset ,String s,AttributeSet a)
     {  char c=s.charAt(0);
      if ((c<='9'&&c>='0')||(c=='.'))
        { try {super.insertString(offset,s,a);}
          catch(BadLocationException e){}
        }
     }
}
public class DigitText extends JApplet
```

```
{ JTextField text=null;
  DigitDocumnet document=new DigitDocumnet();
   public void init()
    { text=new JTextField(30);
      Container con= getContentPane();
      con.setLayout(new FlowLayout());
      text.setDocument(document);
      con.add(text);
    }
}
```

与 JTextField 一样,JTextArea 文本区文档的缺省模型也是 PlainDocumnet 的一个实例.

7　文本窗格　JTextPane

与文本区相比,文本窗格的功能更加强大,它不仅可以显示各种风格段落的文本,而且还可以显示图标,图像.

下面是简单的一个含有文本窗格的小程序

例子 16

```
import javax.swing.*;
import javax.swing.text.*;
import java.awt.*;
public class Example25_16 extends JApplet
{ JTextPane textpane;
   public void init()
  { textpane=new JTextPane();//创建文本窗格.
    getContentPane().add(textpane);
  }
}
```

当你运行上面的程序在文本窗格里输入文本时,感觉和以前的文本区　JTextArea　没什么不同.这是因为我们还没有为文本窗格设置属性.文本窗格的属性包括　段落属性,字符属性.

现在,我们想在我们创建的文本窗格中输入文本时,文本自然是居中对齐,字符的颜色是红色,字体是 Serif 字体,字体的大小是 70 磅　例子 17　.

首先使用 javax.swing.text 包中的类 MutableAttributeSet 声明两个文本属性对象

```
MutableAttributeSet  para_align,char_style;
```

再用 SimpleAttributeSet 类创建这两个对象

```
para_align=new SimpleAttributeSet();
```

```
char_style=new SimpleAttributeSet();
```

这时的文本属性对象还比较笼统,必须进一步初始化.再使用 StyleConstants 类调用相应的方法,并将上述的文本属性对象传递给相应方法的参数,实现对文本属性对象的初始化.

```
StyleConstants.setAlignment(para_align,StyleConstants.ALIGN_CENTER);
```

上述语句将 para_align 传递给 setAlignment 方法,把 para_align 初始化为段落属性 居中对齐 .

```
StyleConstants.setFontFamily( char_style,"Serif");
StyleConstants.setFontSize(char_style,70);
StyleConstants.setForeground(char_style,Color.red);
```

上述 3 个语句通过将 char_style 分别传递给 StyleConstants 类的 setFontFamily,setFontSize 和 setForeground 方法中的参数,把 para_align 文本属性对象初始化为 字体是 Serif 字体,字体的大小是 70 磅,字体的颜色是红色.

使用下列方法

```
StyleConstants.setBold( 属性对象,boolean  a);
StyleConstants.setItalic(属性对象, boolean  a );
StyleConstants.setUndrline(属性对象, boolean  a);
```

还可以把属性对象初始化为是否粗体,斜体或带下划线.

现在文本窗格就可以设置自己的属性了

```
textpane.setParagraphAttributes(para_align,true)
textpane.setCharacterAttributes(char_style,true);
```

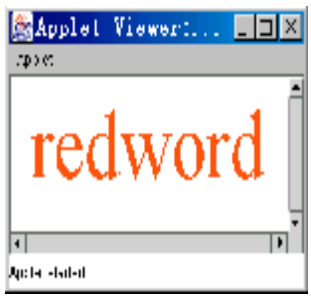


图 25.6 带格式的文本区

例子 17 效果如图 25.6

```
import javax.swing.*;import javax.swing.text.*;
import java.awt.*;
public class Example25_17 extends JApplet
 { JTextPane textpane;
   MutableAttributeSet center_align,char_style;
   public void init()
   { textpane=new JTextPane();//创建文本窗格.
     JScrollPane scroll=
new JScrollPane(textpane,JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
                                        JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
        center_align=new SimpleAttributeSet();
        char_style=new SimpleAttributeSet();  //创建属性对象.
        StyleConstants.setAlignment(center_align,StyleConstants.ALIGN_CENTER);
        StyleConstants.setFontFamily( char_style,"Serif");
```

```
            StyleConstants.setFontSize(char_style,70);
            StyleConstants.setForeground(char_style,Color.red);//为属性对象指定值
            textpane.setParagraphAttributes(center_align,true);//文本窗格设置文本的属性
            textpane.setCharacterAttributes(char_style,true);
            getContentPane().add(scroll);
        }
    }
```

使用 JtextPane 中的 insert(Icon a) 向文本窗格中插入图标.

使用 insertString(textpane.getDocument.getLength(),String s,文本属性对象)向文 本窗格中插入字符串 s.

图 25.7 文本窗格插入图标

例子 18 效果如图 25.7

```
import javax.swing.*;import javax.swing.text.*;
import java.awt.*;
public class Example25_18 extends JApplet
{  JTextPane textpane;
   MutableAttributeSet center_align,char_style_1,char_style_2;
   public void init()
   { textpane=new JTextPane();//创建文本窗口
     JScrollPane scroll=new
     JScrollPane(textpane,JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
     JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
     Document mydocument=textpane.getDocument();//初始化一个文档.
     center_align=new SimpleAttributeSet();
     char_style_1=new SimpleAttributeSet();
     char_style_2=new SimpleAttributeSet();
     StyleConstants.setAlignment(center_align,StyleConstants.ALIGN_CENTER);
     StyleConstants.setFontFamily( char_style_1,"Courier");
     StyleConstants.setFontSize(char_style_1,20);
     StyleConstants.setForeground(char_style_1,Color.red);
```

```
        StyleConstants.setFontFamily( char_style_2,"Serif");
        StyleConstants.setFontSize(char_style_2,14);
        StyleConstants.setForeground(char_style_2,Color.blue);
        textpane.setParagraphAttributes(center_align,true);
        textpane.setCharacterAttributes(char_style_1,true);
        try{  textpane.insertIcon(new ImageIcon("a.jpg"));
        mydocument.insertString(mydocument.getLength(),"Lovely Apple\n",char_style_1);
           }
       catch(BadLocationException e) {}
       textpane.setParagraphAttributes(center_align,true);
       textpane.setCharacterAttributes(char_style_2,true);
       try{ mydocument.insertString(mydocument.getLength(),
                                  "I Want It\n",char_style_2);
           }
       catch(BadLocationException e)  {}
       getContentPane().add(scroll);
     }
}
```

JTextPane 中还有一个读取流的方法　read(InputStream in　Object b).
下面这个例子是把一个文件读入到 JtextPane.

例子 19

```
import javax.swing.*;
import javax.swing.text.*;
import java.awt.*;import java.io.*;
public class Example25_19 extends JApplet
{ JTextPane textpane;FileInputStream readfile;
  public void init()
  { textpane=new JTextPane();//创建文本窗口
    JScrollPane scroll=new JScrollPane(textpane);
    try{ readfile=new FileInputStream("Example25_19.java");
       }
    catch(IOException ee){}
     try{textpane.read(readfile,this);
        }
     catch(Exception e)
      {}
     getContentPane().add(scroll);
  }
```

}

9　文件选择器(JFileChooser)

文件选择器是一个从文件系统中进行文件选择的界面.文件选择器事实上并不能打开或保存文件,它们只能替你得到要打开或保存的文件对象,要想真正实现打开或保存,必须还得使用输入,输出流.

使用构造方法 JFileChooser()创建文件选择器.选择器不是初始可见的,文件选取器是有模式的对话框.下述三个方法

```
showDialog(Component a,String s),
showSaveDialog(Component a),
showOpenDialog(Component a),
```

都可以使得文件选取器可见,三个方法中的参数 a 给出文件选择器的位置,当 a 是 null 时,选择器在屏幕的中央显示.如果组件 a 不空,则在 a 的前面居中显示.当文件选取器对话框消失后,这些方法返回下面的整型常量之一,返回的值依赖于单击了选取器的"批准"按钮还是"取消"按钮.

```
JFileChooser.APPROVE_OPTION
JFileChooser.CANCEL_OPTION
```

当上面的某个方法返回 JFileChooser.APPROVE_OPTION 时,我们可以使用 JFileChooser 类的 getSelecedFile()得到被选择的文件　如果文件选取器中的 file name 文本框是 null,就得不到文件　.下面我们就通过文件的输入流实现对文件的只读打开.

例子 20

```
import javax.swing.*;import java.awt.*;
import java.awt.event.*;import java.io.*;
class FileWin extends JFrame implements ActionListener
{ JButton button; JTextArea text;JTextPane textpane;FileInputStream readfile;
  JScrollPane scroll;Container con;
  JFileChooser chooser=new JFileChooser();
 FileWin()
   { super("有文件选择器的窗口");
     button=new JButton("打开文件选取器");
     button.addActionListener(this);
     textpane=new JTextPane();
     scroll=new JScrollPane(textpane);
     setSize(200,200); setVisible(true);
     addWindowListener(new WindowAdapter()
            {public void windowClosing(WindowEvent e)
                 { System.exit(0);}} );
```

```
        con=getContentPane();con.add(button,BorderLayout.NORTH);
        con.add(scroll,BorderLayout.CENTER);
    }
  public void actionPerformed(ActionEvent e)
   {if(e.getSource()==button)
     {String s;
      int state=chooser.showOpenDialog(null);
      File file=chooser.getSelectedFile();
      if(file!=null&&state==JFileChooser.APPROVE_OPTION)
      { try{ readfile=new FileInputStream(file); //建立到文件的输入流.
           }
        catch(IOException ee){}
        try{  textpane.read(readfile,this);//从流中读取数据.
           }
       catch(IOException e1){}
      }
     }
   }
}
public class Example25_20
{public static void main(String args[])
 {FileWin Win=new FileWin(); Win.pack();  }
}
```

10. 计时器(Timer)与进度条(JProgressBar)

当我们的程序在执行某项耗时的任务时,最好能有个显示任务进度的组件,那么使用进度条(JProgressBar)是一个好办法.在使用进度条时,经常会用到计时器类,该类在 javax.swing 包中.

1 Timer 类.

当某些操作需要周期性地执行,就可以使用计时器.我们可以使用 Timer 类的构造方法 Timer(int a, Object b)创建一个计时器,其中的参数 a 的单位是豪秒,确定计时器每隔 a 毫秒"震铃"一次,参数 b 是计时器的监视器.计时器发生的震铃事件是 ActinEvent 类型事件.当震铃事件发生时,监视器就会监视到这个事件,就会执行接口 ActionListener 中的方法 actionPerformed.因此当震铃每隔 a 毫秒发生一次时,方法 actionPerformed 就被执行一次.当我们想让计时器只震铃一次时,可以让计时器调用 setReapeats(boolean b)方法,参数 b 的值取 false 即可.当我们使用 Timer(int a, Object b)创建计时器时,对象 b 就自动地成了计时器的监视器,不必象其它组件那样,比如按钮,使用特定的方法获得监视器,但负责创建监视器的类必须实现接口 Actionlistener.

计时器创建后,使用 Timer 类的方法 start()启动计时器,使用 Timer 类的方法 stop()停止计时器.

下面的程序中,一个文本区里区每隔 1 秒显示一句话　欢迎光临　,另一个文本区每隔 2 秒钟显示　再见　.

例子 21　效果如图 25.8

```
import javax.swing.*;
import java.awt.*;import java.awt.event.*;
class TimeWin extends JFrame implements ActionListener
{ static JTextArea text1,text2; Boy boy=new Boy();
   JScrollPane scroll_1,scroll_2;Container con;
   Timer time_1,time_2 ;   //声明 2 个计时器对象.
   JSplitPane splitpane;
 TimeWin()
   {super("有计时器窗口");
    time_1=new Timer(1000,this);//TimeWin 对象做计时器的监视器.
    time_2=new Timer(2000,boy);//Boy 对象做计时器的监视器.
    text1=new JTextArea(); text2=new JTextArea();
    scroll_1=new JScrollPane(text1);
    scroll_2=new JScrollPane(text2);
    splitpane=new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,scroll_1,
                             scroll_2);
    setSize(200,200);
    setVisible(true);
    con=getContentPane();con.add(splitpane);
    time_1.start();time_2.start();//启动计时器.
    addWindowListener(new WindowAdapter()
     {public void windowClosing(WindowEvent e)
       { System.exit(0);}} );
   }
  public void actionPerformed(ActionEvent e)
  {text1.append("欢迎光临　"+"\n"); }
}
class Boy implements ActionListener
{ public void actionPerformed(ActionEvent e)
  { TimeWin.text2.append("再见　"+"\n");  }
}
public class Example25_21
{public static void main(String args[])
```

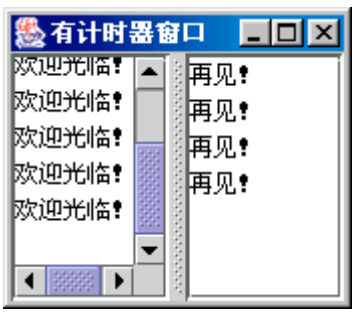


图 25.8　使用计时

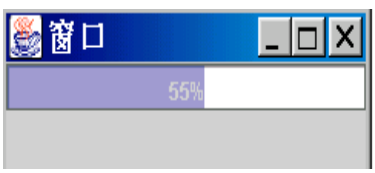


图 25.9　进度条

```
 {TimeWin Win=new TimeWin(); Win.pack();  }
}
```

2 进度条

进度条是一个简单的组件,根据需要用一种颜色动态地填充一个长条矩形,以便显示某任务完成的百分比,如图 25.9 所示.

进度条 JProgressBar 有 3 个常用的构造函数

```
JProgessBar(),
JProgressBar(int min,int max),
JProgressBar(int orient ,int min,int max).
```

JPprogressBar()创建一个水平进度条,它的最大值和最小值分别是缺省值 100 和 0.可以调用 setMinimum(int min),setMaximum(int max)改变这两个值.进度条的最大值并不是进度条的长度,进度条的长度依赖于放置它的布局和本身是否使用了 setSize()设置了大小.进度条的最大值 max 是指将进度条平均分成 max 份.如果你使用 JProgressBar() 创建了一个进度条 p_bar,那么 p_bar 默认地被平均分成 100 份.当 p_bar 根据需要调用了方法 setValue(int a)后,比如 p_bar.setValue(20),那么这时进度条的颜色条就填充了整个长条矩形的 20/100,即 20%,如果进度条的最大值被设置成 1000,那么这时进度条的颜色条就填充了整个长条矩形的 20/1000,即 2%(a 的值不能超过 max).如果进度条的最小值是 min, 那么使用 setValue(int a)方法时,a 的值不能小于 min.

使用 JProgressBar(int min,int max),JProgressBar(int orient ,int min,int max)可以创建进度条,并给出进度条的最大值和最小值,参数 orient 取

```
JProgressBar.HORIZONTAL 和 JProgressBar.VERTICAL,
```

决定进度条是水平放置还是垂直放置.

进度条使用方法 setStringPainted(boolean a)来设置是否使用百分数或字符串来表示进度条的进度情况,使用 int getValue()来获取进度值.

下面看一个简单的例子,假设有 55 个苹果 第一天吃一个,第二天吃 2 个,第 3 天吃 3 个…我们在程序中通过一个进度条显示被吃掉的苹果的进度.

例子 22

```
import javax.swing.*;import java.awt.*;
import java.awt.event.*;
class BarWin extends JFrame implements ActionListener
{  Timer time_1;  int sum=0,i=1;
   JProgressBar p_bar;Container con;
 BarWin()
   {super("窗口");
    time_1=new Timer(1000,this);//TimeWin对象做计时器的监视器,每
```

```
                                        //1000 毫秒震铃一次.
      p_bar=new JProgressBar(0,55);
      p_bar.setBackground(Color.white);
      p_bar.setStringPainted(true);
      setSize(200,200);
      setVisible(true);
      con=getContentPane();con.add(p_bar,BorderLayout.NORTH);
      time_1.start();
      addWindowListener(new WindowAdapter()
       {public void windowClosing(WindowEvent e)
         { System.exit(0);}} );
       }
    public void actionPerformed(ActionEvent e)
    {  sum=sum+i;
       p_bar.setValue(sum);//吃掉 sum/55
       i=i+1;
       if(sum>=55)
       time_1.stop();
    }
  }
  public class Example25_22
  {public static void main(String args[])
   {BarWin Win=new BarWin(); Win.pack();  }
  }
```

我们可以使用带进度条的输入流读取文件.如果读取文件时希望看见文件的读取进度可以使用 javax swing 包提供的输入流类 ProgressMonitorInputStream.它的构造方法是

```
ProgressMonitorInputStream(Conmponent c,String s,InputStream);
```

该类创建的输入流在读取文件时会弹出一个显示读取速度的进度条,进度条在参数 c 指定的组件的正前方显示,若该参数取 null,则在屏幕的正前方显示.

例子 23 效果如图 25.10

```
import javax.swing.*;
import java.io.*;import java.awt.*;import java.awt.event.*;
public class Example25_23
{ public static void main(String args[])
  { byte b[]=new byte[30];
    JTextArea text=new JTextArea(20,20);
    JFrame  jframe=new JFrame();
```

```
        jframe.setSize(200,300);jframe.setBackground(Color.blue);
        jframe.setVisible(true);
        jframe.addWindowListener(new WindowAdapter()
             {public void windowClosing(WindowEvent e)
                {System.exit(0);} });
        Container contentpane=jframe.getContentPane();
        contentpane.add(text,BorderLayout.CENTER);

    try{  FileInputStream input=new FileInputStream("Example25_23.java");
          ProgressMonitorInputStream input_progress=
          new ProgressMonitorInputStream(contentpane,"读取 java 文件",input);
          ProgressMonitor p=input_progress.getProgressMonitor();//获得进度条.
          while(input_progress.read(b)!=-1)
             { String s=new String(b);
               text.append(s);
               Thread.sleep(1000);//由于文件较小,为了看清进度条这里有意延缓 1 秒.
             }
        }
        catch(InterruptedException e){}
        catch(IOException e){}
    }
}
```

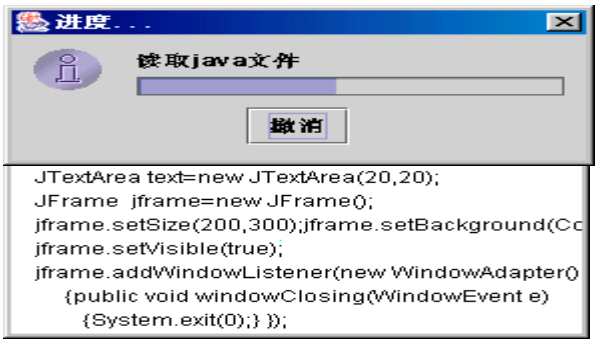


图 25.10　　带进度条的输入流

11　表格(JTable)

表格组件以行和列的形式显示数据,允许对表格中的数据进行编辑.表格的模型功能强大,灵活并易于执行.表格是最复杂的组件,对于初学者,我们这里只介绍默认的表格模型.

JTable 有 7 个构造函数,这里我们介绍常用的 3 个.

JTable()　创建默认的表格模型.

JTable(int a,int b)　　创建 a 行,b 列的默认模型表格

JTable (Object data[][],Object　columnName[])　创建默认表格模型对象,并且显示由 data 指定的二维数组的值,其列名由数组 columnName 指定.

通过对表格中的数据进行编辑,可以修改表格中 2 数组 data 中对应的数据数据.在表格中输入或修改数据后,需按回车或用鼠标单击表格的单元格确定所输入或修改的结果.当表格需要刷新显示时,让表格调用 repaint 方法.

下面的例子是一个成绩单录入程序,客户通过一个表格的单元格输入每个人的数学和英语成绩.单击按钮后,将总成绩放入相应的表格单元中.因为 Object 类是 java 中所有类的默认父类,所以当我们在表格中输入一个数值时被认为是一个 Object 对象,Object 类有一个很有用的方法 toString(),它可以得到对象的字符串表示.

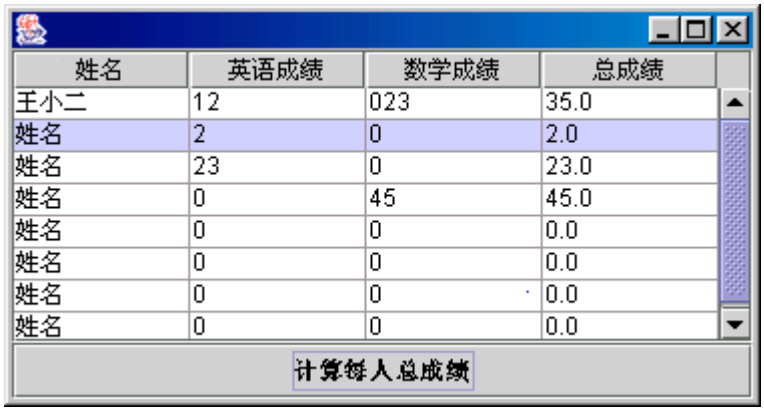


图 25.11 使用表格统计成

例子 24 效果如图 25.11

```
import javax.swing.*;import java.awt.*;
import java.awt.event.*;
public class Example25_24 extends JFrame implements ActionListener
{  JTable table;Object a[][];
   Object name[]={"姓名","英语成绩","数学成绩","总成绩"};
   JButton button;
   Example25_24()
   { a=new Object[8][4];
     for(int i=0;i<8;i++)
      { for(int j=0;j<4;j++)
          {if(j!=0)
             a[i][j]="0";
           else
             a[i][j]="姓名";
          }
      }
```

```
    button=new JButton("计算每人总成绩");
    table=new JTable(a,name);
    button.addActionListener(this);
    getContentPane().add(new JScrollPane(table),BorderLayout.CENTER);
getContentPane().add(new JLabel("修改或录入数据后,需回车确认"),BorderLayout.SOUTH);
    getContentPane().add(button,BorderLayout.SOUTH);
    setSize(200,200);
    setVisible(true);
    validate();
    addWindowListener(new WindowAdapter()
           {public void windowClosing(WindowEvent e)
                  { System.exit(0);
                  }
           });
  }
  public void actionPerformed(ActionEvent e)
  {  for(int i=0;i<8;i++)
     { double sum=0;
       boolean boo=true;
       for(int j=1;j<=2;j++)
         {  try{
                sum=sum+Double.parseDouble(a[i][j].toString());
             }
           catch(Exception ee)
             {
               boo=false;
               table.repaint();
             }
           if(boo==true)
            {
              a[i][3]=""+sum;
              table.repaint();
            }
         }
     }
   }
   public static void main(String args[])
   {  Example25_24 Win=new Example25_24();
   }
```

}

下面的例子 25 是把数据库的查询结果放到一个表格里.

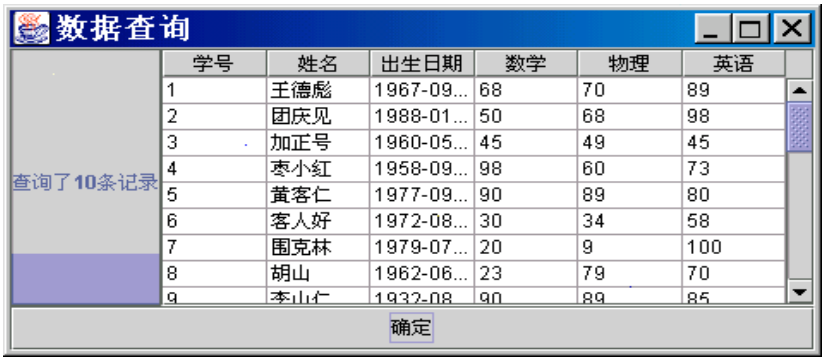


图 25.12　使用表格显示查询记

例子 25　效果如图 25.12

```
import javax.swing.*;
import java.awt.event.*;
import java.sql.*;
import java.awt.*;
 class ResultWin extends JFrame implements ActionListener
{  Object a[][];
   Object columnName[]={"学号","姓名","出生日期","数学","物理","英语"};
   JTable table;JButton button;
   Container container;
   String name,xuehao;Date date; int math,physics,english;
   Connection con;Statement sql; ResultSet rs;
   JProgressBar p_bar;
  ResultWin()
  {  super("数据查询");
     a=new Object[30][6];
     table=new JTable(a,columnName);
     setSize(300,300);setVisible(true);
     button=new JButton("确定");
     addWindowListener(new WindowAdapter()
     {public void windowClosing(WindowEvent e)
       { System.exit(0);}} );
     button.addActionListener(this);
     p_bar=new JProgressBar(JProgressBar.VERTICAL,0,50);
```

```
        p_bar.setStringPainted(true) ;
        container=getContentPane();
        container.add(button,BorderLayout.SOUTH);
        container.add(new JScrollPane(table),BorderLayout.CENTER);
        container.add(p_bar,BorderLayout.WEST);
      }
   public void actionPerformed(ActionEvent evt)
    {if(evt.getSource()==button)
     {int i=0;
      try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  }
      catch(ClassNotFoundException e){}
      try
{con=DriverManager.getConnection("jdbc:odbc:redsun","snow","ookk");
         sql=con.createStatement();
         rs=sql.executeQuery("SELECT * FROM chengjibiao");
         while(rs.next())
         { xuehao=rs.getString(1); name=rs.getString(2);date=rs.getDate(3);
           math=rs.getInt("数学"); physics=rs.getInt("物理");english=rs.getInt("英语");
          a[i][0]=xuehao;a[i][1]=name;a[i][2]=date;a[i][3]=String.valueOf(math);
          a[i][4]=String.valueOf(physics);a[i][5]=String.valueOf(english);
          i++;
          p_bar.setValue(i);p_bar.setString("查询了"+i+"条记录");
         }
        con.close();
       }
      catch(SQLException e1) {}
     }
    }
}
public class Example25_25
{ public static void main(String args[])
   {ResultWin win=new ResultWin(); win.pack(); }
}
```

12　菜单条,菜单,菜单项(JMenuBar,JMenu,JMenuItem)

在 Java swing 的小应用程序和应用程序中,使用 JMenubar()构造函数创建菜单条,再用 Jmenu 创建一些菜单并将它们放进菜单条里,最后使用 add(JMenubar)把菜单条放到 JApplet 或 JFrame 框架中.这部分内容和 15 章有很多相似之处,故不再赘述.

例子 26 效果如图 25.13

图 25.213 菜单

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Example24_26 extends JApplet
{ Container con;
  public void init()
 {con=getContentPane();
  JMenuBar menubar=new JMenuBar();
  con.add(menubar,BorderLayout.NORTH);
  JMenu fileMenu=new JMenu("文件");
  JMenu editMenu=new JMenu("编辑");
  JMenu helpMenu=new JMenu("帮助");
  JMenuItem item1=new JMenuItem("打开");
  JMenuItem  item2=new JMenuItem("保存"); //创建 6 个菜单项.
  fileMenu.add(item1); fileMenu.add(item2);
  menubar.add(fileMenu);  menubar.add(editMenu);
  menubar.add(helpMenu);
 }
}
```

13 工具条 JToolBar 与工具提示

工具条就是一个显示一组动作,命令或功能控件的组件.一般说来,工具条中的组件都是带图标的按钮.工具条对频繁使用的组件非常有用.使用 **JToolBar()**来创建一个工具条对象,然后使用 **add()**方法将带图标的按钮加到工具条中.为了能使客户明白每个组件的功能,我们可以使用组件通有的方法 **setToolTipText(String s)**设置组件的提示文字,当鼠标指针在组件上停止时,就会显示组件的文字提示.

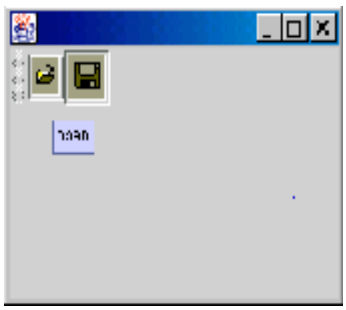


图 25.14 工具

例子 27 效果如图 25.14

```
import javax.swing.*;import java.awt.*;
import java.awt.event.*;
class ToolWin extends JFrame implements ActionListener
{ JButton button1,button2; JToolBar bar; Container con;
 ToolWin()
 {con=getContentPane();
  setSize(300,250);setVisible(true);
  Icon open_icon =new ImageIcon("open.gif");
```

```
  Icon save_icon =new ImageIcon("save.gif");
  button1=new JButton(open_icon); button2=new JButton(save_icon);
  bar=new JToolBar();//工具条对象
  bar.add(button1);bar.add(button2);
  con.add(bar,BorderLayout.NORTH);
  button1.addActionListener(this);
  button1.setToolTipText("open");//设置组件的提示文字
  button2.setToolTipText("save");
 }
 public void actionPerformed(ActionEvent e)
 {if(e.getSource()==button1)
  {JFileChooser c=new JFileChooser();
    c.showOpenDialog(null);
  }
 }
}
public class Example25_27
{static void main(String args[])
 {ToolWin win=new ToolWin() ;win.pack();}
}
```

14　树(JTree)

一个 JTree 类对象提供了一个用树型结构分层显示数据的视图,如图 25.24 所示.树中最基本的对象叫做节点,它表示在给定层次结构中的数据项.从图中可以看出,树以垂直方式显示数据,每行显示一个节点.树中只有一个根节点,所有其它节点从这里引出.除根节点外,其它节点分为两类　一类是带子节点的分支节点,另一类是不带子节点的叶节点.每一个节点关联着一个描述该节点的文本标签和图象图标.文本标签是节点的字符串表示,图标指明该节点是否是叶节点.初始状态的树型视图,在默认情形下,只显示根节点和它的直接子节点.用户可以用鼠标双击分节点的图标或单击图标前的"开关"使该节点扩展或收缩　使它的子节点显示或不显示　,见图 25.15.

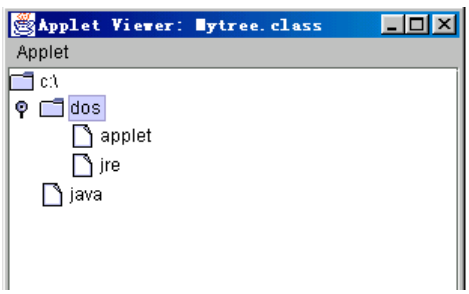


图 25.15　　显示树结构的小程序

树节点由 javax.swing.tree 包中的接口 TreeNode 定义,该接口又被 MutableTreeNode 接口扩展,而 MutableTreeNode 接口又由 DefaultMutableTreeNode 类来实现.为了创建一个树,使用 DefaultMutableTreeNode 类为该树创建节点,它的两个常用的构造方法是

DefaultMutableTreeNode Object userObject ,

DefaultMutableTreeNode Object userObject,boolean allowChildren .

第一个构造方法创建的节点默认可以有子节点,即它可以使用 add 方法添加其它节点作为它的子节点.如果需要,一个节点可以使用方法 setAllowsChildren(boolean b)来设置是否允许有子节点.节点创建完毕后,再使用 JTree 的构造方法 JTree(TreeNode root)创建根节点是 root 的树.在下面的例子中我们创建了一个如图 25.24 的树.

例子 28 效果如图 25.15

```
import javax.swing.*;import javax.swing.tree.*;
import java.awt.*;
public class Mytree extends JApplet
{ public void init()
  {Container con=getContentPane();
   DefaultMutableTreeNode root=new DefaultMutableTreeNode("c:\\");//树的根节点.
   DefaultMutableTreeNode t1=new DefaultMutableTreeNode("dos");//节点.
   DefaultMutableTreeNode t2=new DefaultMutableTreeNode("java");//节点.
   DefaultMutableTreeNode t1_1=new DefaultMutableTreeNode("applet");
   DefaultMutableTreeNode t1_2=new DefaultMutableTreeNode("jre");
   root.add(t1);root.add(t2);
   t1.add(t1_1);t1.add(t1_2);//t1_1,t1_2成为t1的子节点.
   JTree tree =new JTree(root); //创建根为root的树.
   JScrollPane scrollpane=new JScrollPane(tree);
   con.add(scrollpane);
  }
}
```

树中的节点可以发生选择事件,即用鼠标单击节点.一个树可以使用 addTreeSelectionListener TreeSelectionListener 方法获得一个监视器,处理事件的接口是 TreeSelectionListener .树通过使用方法 getLastSelectedPathComponent()获取选中的节点,使用方法 getUserObject()得到与节点相关的信息.下面的例子是通过树来显示一个同学通讯录,当点击树的节点时,文本区显示某些信息.如图 25.16.

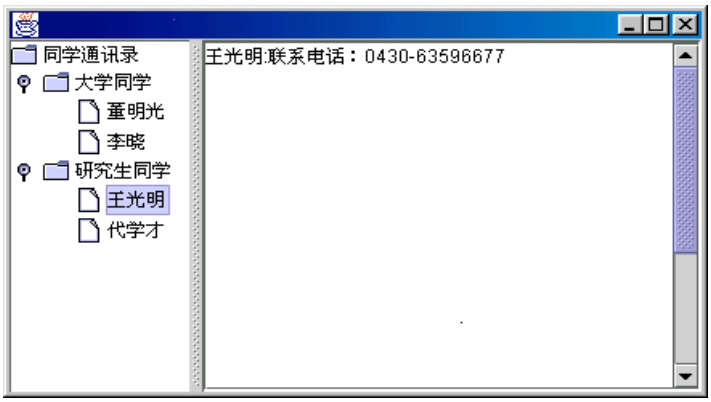


图 25.16　　处理节点事件的树

例子 29(效果如图 25.16 所示)

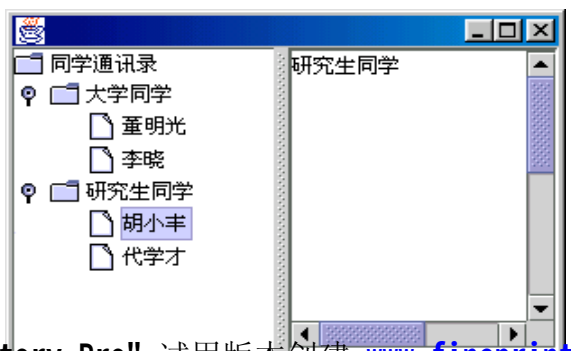
```
import javax.swing.*;
import javax.swing.tree.*;import java.awt.*;
import java.awt.event.*;import javax.swing.event.*;
public class Mytree2 extends JFrame implements TreeSelectionListener
{  JTree tree=null;JTextArea text=new JTextArea(20,20);
  Mytree2()
  {Container con=getContentPane();
   DefaultMutableTreeNode root=new DefaultMutableTreeNode("同学通讯录");
   DefaultMutableTreeNode t1=new DefaultMutableTreeNode("大学同学");
   DefaultMutableTreeNode t2=new DefaultMutableTreeNode("研究生同学");
   DefaultMutableTreeNode t1_1=new DefaultMutableTreeNode("董明光");
   DefaultMutableTreeNode t1_2=new DefaultMutableTreeNode("李晓");
   DefaultMutableTreeNode t2_1=new DefaultMutableTreeNode("王光明");
   DefaultMutableTreeNode t2_2=new DefaultMutableTreeNode("代学才");
   root.add(t1);root.add(t2);
   t1.add(t1_1);t1.add(t1_2);  t2.add(t2_1);t2.add(t2_2);
   tree =new JTree(root);
   JScrollPane scrollpane=new JScrollPane(text);
   JSplitPane splitpane=new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                              true,tree,scrollpane);
   tree.addTreeSelectionListener(this);
   con.add(splitpane);
   addWindowListener(new WindowAdapter()
    { public void windowClosing(WindowEvent e)
      {System.exit(0);} });
```

```
    setVisible(true);setBounds(70,80,200,300);
     }
   public void valueChanged(TreeSelectionEvent e)
    { if(e.getSource()==tree)
       {DefaultMutableTreeNode node=
        (DefaultMutableTreeNode)tree.getLastSelectedPathComponent();
        if(node.isLeaf())
          { String str=node.toString();
            if(str.equals("董明光"))
              {text.setText(str+":联系电话  0411-4209876");}
            else if(str.equals("李晓"))
             {text.setText(str+":联系电话  010-62789876");}
            else if(str.equals("王光明"))
             {text.setText(str+":联系电话  0430-63596677");}
            else if(str.equals("代学才"))
             {text.setText(str+":联系电话  020-85192789");}
          }
        else
         {text.setText(node.getUserObject().toString());
         }
       }
  }
}
class Example25_29
{public static void main(String args[])
  { Mytree2 win=new Mytree2();win.pack();}
}
```

一个树可以使用 **setEditable(true)**方法使得树节点成为可编辑的,这时,可以在节点单元上连击三次鼠标,或者中间加一暂停的双击鼠标激活节点的文本字段实现编辑操作.完成编辑后,按回车键确认.图 **25.17** 是"例子 **29**"加入代码

```
tree.setEditable(true);
```

之后运行的效果图,其中我们激活了节点 **t2_1** 并将该节点的字符串王光明改成胡小丰.



一个树对象还对应一个用于描述节点显示形式的 DefaultTreeCellRenderer 类.用该类的 setXXX 等方法,可以方便地定制树的显示.比如将叶节点的图标由文件 leaf.gif 指定,代码如下

```
DefaultTreeCellRenderer render=new  DefaultTreeCellRenderer();
render.setLeafIcon(new ImageIcon("leaf.gif"));
tree.setCellRenderer(render);
```

DefaultTreeCellRenderer 类的一些常用方法如下

```
   setLeafIcon(Icon  newIcon);
setBackground(Color newColor ;
setClosedIcon(Icon,newIcon);
setOpenIcon(Icon,newIcon);
setTextSelectionColor(Color,newColor);   setTextNonSelectionColor(Color,newColor);
setFont(Font,f);
```

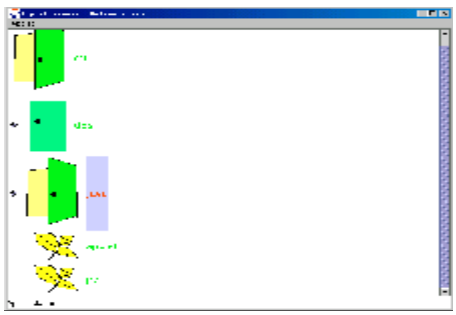


图 25.18　　用自制图标和字体显示的

例子 30 用自制的图标和字体来显示一个树.

例子 30(效果如图 25.18)

```
import javax.swing.*;import javax.swing.tree.*;
import java.awt.*;
public class Mytree3 extends JApplet
{ public void init()
  {Container con=getContentPane();
```

```
        DefaultMutableTreeNode root=new DefaultMutableTreeNode("c:\\");//树的根节点.
        DefaultMutableTreeNode t1=new DefaultMutableTreeNode("dos");//节点.
        DefaultMutableTreeNode t2=new DefaultMutableTreeNode("java");//节点.
        DefaultMutableTreeNode t1_1=new DefaultMutableTreeNode("wps");
        DefaultMutableTreeNode t1_2=new DefaultMutableTreeNode("epg");
        DefaultMutableTreeNode t2_1=new DefaultMutableTreeNode("applet");
        DefaultMutableTreeNode t2_2=new DefaultMutableTreeNode("jre");
        root.add(t1);root.add(t2);
        t1.add(t1_1);t1.add(t1_2);
        t2.add(t2_1);t2.add(t2_2);
        JTree tree =new JTree(root); //创建根为 root 的树.
        DefaultTreeCellRenderer render=new  DefaultTreeCellRenderer();
        render.setLeafIcon(new ImageIcon("leaf.gif"));
        render.setBackground(Color.yellow);
        render.setClosedIcon(new ImageIcon("close.gif"));
        render.setOpenIcon(new ImageIcon("open.gif"));
        render.setTextSelectionColor(Color.red);
        render.setTextNonSelectionColor(Color.green);
        render.setFont(new Font("TimeRoman",Font.BOLD,16));
        tree.setCellRenderer(render);
        JScrollPane scrollpane=new JScrollPane(tree);
        con.add(scrollpane);
      }
  }
```

例子 31 用树来显示一个文本格式的通讯录内容.如果按着文本格式通讯录的书写规则增加新的通讯记录,程序运行时,树将增加相应的节点.如图 **25.19**,图 **25.20** 所示.

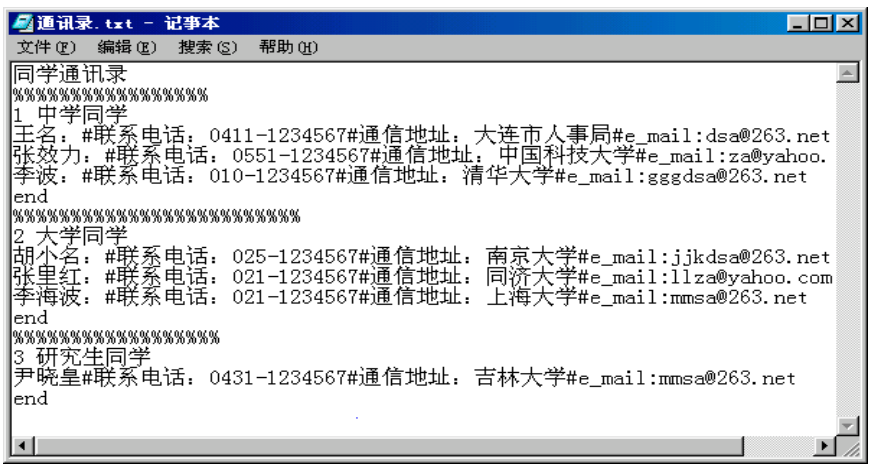


图 25.19　　文本格式通讯录

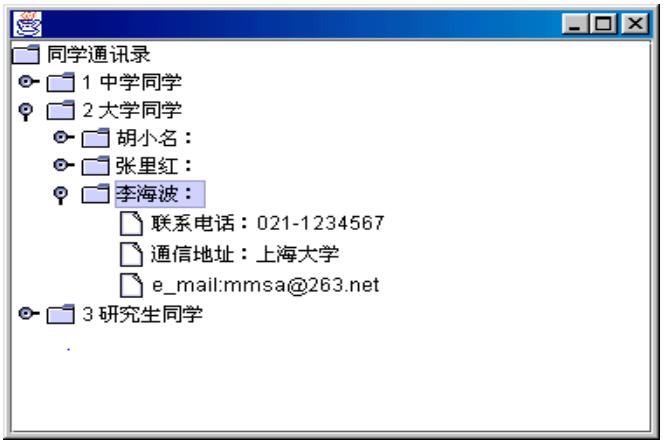


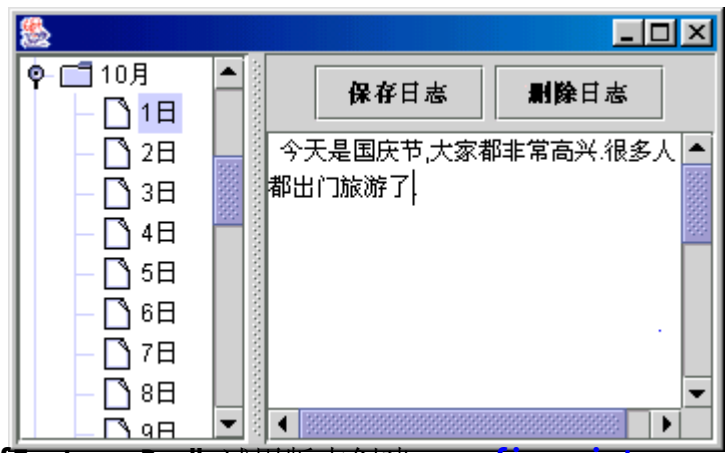
图 25.20　用树实现的通讯录

例子 31　效果如图 25.20 示

```
import javax.swing.*;
import javax.swing.tree.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;import java.util.*;
class Classmate extends JFrame
{  JTree tree=null; DefaultMutableTreeNode root;
   BufferedReader in; FileReader file;
   Classmate()
   {  Container con=getContentPane();
      String s=null;
      try { File f=new File("通讯录.txt");
            file=new FileReader(f);
            in=new BufferedReader(file);
           }
      catch(FileNotFoundException e){}
      try { s=in.readLine();                    //读取第一行并用它创建根节点.
            root=new DefaultMutableTreeNode(s);
           }
      catch(IOException exp){}
      try
         {  while((s=in.readLine())!=null&&(s.startsWith("%")))
               { s=in.readLine();
                 DefaultMutableTreeNode 同学种类=new DefaultMutableTreeNode(s);
```

```
                root.add(同学种类);
                while((s=in.readLine())!=null&&!(s.startsWith("end")))
                     {  StringTokenizer tokenizer=new StringTokenizer(s,"#");
                        String temp=tokenizer.nextToken();
                        DefaultMutableTreeNode 同学种类_姓名
                        =new DefaultMutableTreeNode(temp);
                        同学种类.add(同学种类_姓名);
                         while(tokenizer.hasMoreTokens())
                         {
            同学种类_姓名.add(new DefaultMutableTreeNode(tokenizer.nextToken()));
                         }
                      }
                }
             }
          catch(IOException exp){}
          tree =new JTree(root);
          JScrollPane scrollpane=new JScrollPane(tree);
          con.add(scrollpane);
          addWindowListener(new WindowAdapter()
                  { public void windowClosing(WindowEvent e)
                  {System.exit(0);} });
          setVisible(true);setBounds(70,80,200,300);
     }
  }
public class Example31
{ public static void main(String args[])
    { Classmate win=new Classmate();win.pack();}
}
```

下面的例子 32 结合树和输入输出流实现了一个日历记事本,用户可以记录某月某日的事情,也可以查阅或删除过去记录的内容.



例子 32　效果如图 25.21

```
import javax.swing.*;
import javax.swing.tree.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
import java.io.*;
class Remember extends JFrame implements TreeSelectionListener,ActionListener
{  JTree tree=null;JTextArea text=new JTextArea(" ",20,20);int i=0;
   DefaultMutableTreeNode root;JButton b_save=new JButton("保存日志"),
   b_del=new JButton("删除日志");
   DefaultMutableTreeNode month[]=new DefaultMutableTreeNode[13];
   Remember()
   {  Container con=getContentPane();
      DefaultMutableTreeNode root=new DefaultMutableTreeNode("日历记事本");
      for(i=1;i<=12;i++)
          {  month[i]=new DefaultMutableTreeNode(""+i+"月");
             root.add(month[i]);
          }
      for(i=1;i<=12;i++)
      {  if(i==1||i==3||i==5||i==7||i==8||i==10||i==12)
          {  for(int j=1;j<=31;j++)
             month[i].add(new DefaultMutableTreeNode(j+"日"));
          }
         else if(i==4||i==6||i==9||i==11)
          {  for(int j=1;j<=30;j++)
                month[i].add(new DefaultMutableTreeNode(j+"日"));
          }
         else
         {  for(int j=1;j<=28;j++)
                month[i].add(new DefaultMutableTreeNode(j+"日"));
         }
      }
     b_save.addActionListener(this); b_del.addActionListener(this);
```

```
  tree =new JTree(root);
  JPanel p=new JPanel();p.setLayout(new BorderLayout());
  JScrollPane scrollpane_1=new JScrollPane(text);
  p.add(scrollpane_1,BorderLayout.CENTER);
  JPanel p_1=new JPanel();p_1.add(b_save);p_1.add(b_del);
  p.add(p_1,BorderLayout.NORTH);
  JScrollPane scrollpane_2=new JScrollPane(tree);
  JSplitPane splitpane=
  new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,true,scrollpane_2,p);
  tree.addTreeSelectionListener(this);
  con.add(splitpane);
  addWindowListener(new WindowAdapter()
         {  public void windowClosing(WindowEvent e)
                     {System.exit(0);
                     }
         });
  setVisible(true);setBounds(70,80,200,300);
}
public void valueChanged(TreeSelectionEvent e)
{  text.setText(" ");
   if(e.getSource()==tree)
     {  DefaultMutableTreeNode node=
        (DefaultMutableTreeNode)tree.getLastSelectedPathComponent();
        if(node.isLeaf())
           {  String str=node.toString();
              for(int i=0;i<=12;i++)
                {  if(node.getParent()==month[i])
                   { try
                        {  String temp=null;
                           File f=new File(node.getParent().toString()+str+".txt");
                           FileReader file=new FileReader(f);
                           BufferedReader in=new BufferedReader(file);
                           while((temp=in.readLine())!=null)
                                  text.append(temp+'\n');
                           file.close();in.close();
                         }
                      catch(FileNotFoundException e1){}
                      catch(IOException e1){}
                    }
```

```
                }
            }
        }
    }
    public void actionPerformed(ActionEvent e)
    {  if(e.getSource()==b_save)
        {  DefaultMutableTreeNode node=
          (DefaultMutableTreeNode)tree.getLastSelectedPathComponent();
          String str=node.toString();
          if(node.isLeaf())
            { try
                { File f=new File(node.getParent().toString()+str+".txt");
                  FileWriter tofile=new FileWriter(f);
                  BufferedWriter out=new BufferedWriter(tofile);
                  out.write(text.getText(),0,(text.getText()).length());
                  out.flush();
                  tofile.close();out.close();
                }
              catch(FileNotFoundException e1){}
               catch(IOException e1){}
             }
        }
       else if(e.getSource()==b_del)
        {  DefaultMutableTreeNode node=
          (DefaultMutableTreeNode)tree.getLastSelectedPathComponent();
          String str=node.toString();
          if(node.isLeaf())
            {
              File f=new File(node.getParent().toString()+str+".txt");
              f.delete();
            }
        }
    }
}
public class Example25_32
{  public static void main(String args[])
    {  Remember win=new Remember();win.pack();
    }
}
```

习题二十五

1　改进第二十章的例子 12,将通讯录显示到一个表格中.
2　用 java.swing 中的 JButton 组件改进第十八章的例子 10,使得华容道中的人物带有图标.
3　用 java.swing 包中的组件改写十八章的例子 8.
4　用表格编写一个计算三阶行列式的程序.
5　使用记时器　Timer　编写一个动画程序.

第二十六章 常见数据结构的 Java 实现

我们在编写程序时经常要和各种数据打交道,为处理这些数据所选的数据结构对于我们的程序的运行效率是非常重要的.这章讲述几种常见的数据结构的 Java 实现.在学习数据结构这门课程的时候,人们要用具体的算法去实现相应的数据结构,例如,为了实现链表这种数据结构,需要实现往链表中插入节点或从链表中删除节点的算法,感觉有些烦琐.在 jdk1.2 之后,Java 提供了实现常见数据结构的类,创建链表等数据结构和创建数组一样简单,不再需要你去写具体的算法.我们主要讲述这些类的用法,但对这些数据结构的原理的掌握对于我们熟练地用好这些类还是很有帮助的.如果需要了解这些数据结构的原理,可以参考本系列教材 数据结构实用教程 徐孝凯 编著 .

26.1 链表

如果需要处理一些类型相同的数据,人们习惯上使用数组这种数据结构,但数组在使用之前必须定义大小,而且不能动态定义大小.有时可能给数组分配了太多的单元而浪费了宝贵的内存资源,糟糕的一方面是,程序运行时需要处理的数据可能多于数组的单元.当需要动态地减少或增加数据项时,可以使用链表这种数据结构.

链表是由若干个称作节点的对象组成的一种数据结构,每个节点含有一个数据和下一个节点对象的引用 单链表 ,或含有一个数据并含有上一个节点对象的引用和下一个节点对象的引用 双链表 .

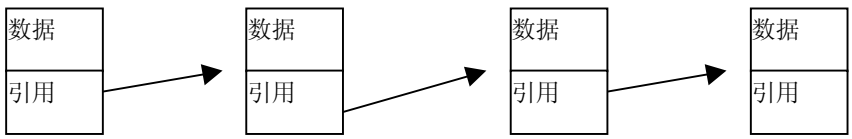


图 26.1 单链表示意图

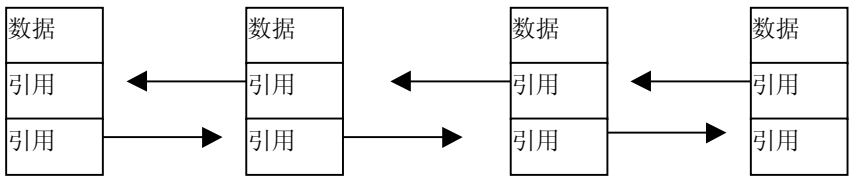


图 26.2 双链表示意图

1 .创建链表

使用 java.util 包中的 LinkedList 类创建一个链表.例如,

```
LinkedList mylist=new LinkedList();
```

创建了一个空双链表.然后 mylist 链表可以使用 add()方法向这个链表依次增加节点.例如

```
mylist.add("It");mylist.add("is");
mylist.add("a");mylist.add("door");
```

这时,就形成了如图 26.3 所示的链表, 个节点是自动连接在一起的,不需要我们去做连接,也就是说,不需要我们去操作安排节点中所存放的下一个或上一个节点的引用.

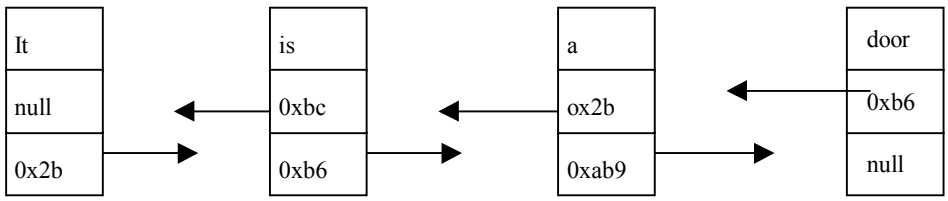


图 26.3 用 LinkedList 创建的一个链表

mylist 可以使用方法 public Object get(index i)获取第 i 个节点中存储的数据.存放在节点中的数据都被看作是一个 Object 对象.下面的例子构造一个含有 个节点链表,并输出节点中的数据 效果如图 26.4 .

例子 1 效果如图 26.4

```
import java.util.*;
public class LinkListOne
{public  static void main(String args[])
 { LinkedList mylist=new LinkedList();
   mylist.add("It");          //链表中的第一个节点.
   mylist.add("is");      //链表中的第二个节点.
   mylist.add("a");            //链表中的第三个节点.
   mylist.add("door");        //链表中的第四个节点.
   int number=mylist.size();    //获取链表的长度.
   for(int i=0;i<number;i++)
   {String temp=(String)mylist.get(i);
    System.out.println("第"+i+"节点中的数据:"+temp);
   }
 }
}
```

```
F:\Example\javadata>java LinkListOne
第0节点中的数据:It
第1节点中的数据:is
第2节点中的数据:a
第3节点中的数据:door
```

图 26.4 输出节点中的数据

注 由于任何类都是 Object 类的子类,因此可以把任何一个对象作为链表的节点对

象.需要注意的是当使用 get()获取一个节点对象后,要用类型转换运算符转化回原来的类型.

2 LinkedList 类中的常用方法

public boolean add(Object element) 向链表的末尾填加一个新的节点对象 elememt.

public void add(int index ,Object element) 向链表的指定位置尾填加一个新的节点对象 elememt.

public void addFirst(Object element) 把节点对象 elememt 填加到链表的表头.

public void addLast(Object element) 把节点对象 elememt 填加到链表的末尾.

public void clear() 删除链表的所有节点对象.

public Obiect remove(int index) 删除指定位置上的节点对象.

public boolean remove(Object element) 将首次出现的节点对象 elemen 删除.

public Obiect removeFirst() 删除第一个节点对象,并返回这个节点对象.

public Obiect removeLast() 删除最后一个节点对象.

public Object get(int index) 得到链表中指定位置处的节点对象.

public Object getFirst() 得到链表中第一个节点对象.

public Object getLast() 得到链表中最后一个节点对象.

public int indexOf(Object element) 返回节点对象 element 在链表中首次出现的位置,如果链表中无此节点对象则返回-1.

public int lastIndexOf(Object element) 返回节点对象 element 在链表中最后出现的位置,如果链表中无此节点对象则返回-1.

public Object set(int index ,Object element) 用节点对象 element 替换链表中指定位置处的节点对象.并返回链表中先前位置处的节点对象.

public int size() 返回链表的长度,即节点的个数.

public boolean contains(Object element) 判断链表节点对象中是否含有 element.

下面的例子 中包含了 LinkedList 类中的一些常用方法.

例子

```
import java.util.*;
public class LinkListTwo
{public  static void main(String args[])
 { LinkedList mylist=new LinkedList();
   mylist.add("is"); mylist.add("a");
   int number=mylist.size();
   System.out.println("现在链表中有"+number+"个节点:");
   for(int i=0;i<number;i++)
   {String temp=(String)mylist.get(i);
    System.out.println("第"+i+"节点中的数据:"+temp);
   }
```

```
   mylist.addFirst("It");mylist.addLast("door");
   number=mylist.size();
  System.out.println("现在链表中有"+number+"个节点:");
   for(int i=0;i<number;i++)
   {String temp=(String)mylist.get(i);
    System.out.println("第"+i+"节点中的数据:"+temp);
   }
   mylist.remove(0);mylist.remove(1);
   mylist.set(0,"open");
   number=mylist.size();
    System.out.println("现在链表中有"+number+"个节点:");
     for(int i=0;i<number;i++)
     {String temp=(String)mylist.get(i);
      System.out.println("第"+i+"节点中的数据:"+temp);
     }
   }
  }
```

3　使用 Iterator 类遍历链表

在例子　和例子　中我们借助 get 方法实现了遍历链表.我们可以借助 Iterator 对象实现遍历链表,一个链表对象可以使用 iterator()方法获取一个 Iterator 对象,后者使用 next()方法遍历链表.在下面的例子　中,我们把学生的成绩存放在一个链表中,并实现了遍历链表.

例子

```
import java.util.*;
class Student
{String name ;int number;float score;
  Student(String name,int number,float score)
  {this.name=name;this.number=number;this.score=score;
  }
}
public class LinkListThree
{public  static void main(String args[])
 { LinkedList mylist=new LinkedList();
   Student stu_1=new Student("赵好民" ,9012,80.0f),
           stu_2=new Student("钱小青" ,9013,90.0f),
           stu_3=new Student("孙力枚" ,9014,78.0f),
           stu_4=new Student("周左右" ,9015,55.0f);
    mylist.add(stu_1); mylist.add(stu_2);
```

```
        mylist.add(stu_3); mylist.add(stu_4);
        Iterator iter=mylist.iterator();
       while(iter.hasNext())
       {  Student te=(Student)iter.next();
          System.out.println(te.name+" "+te.number+"  "+te.score);
         }
       }
      }
```

下面是一个较复杂的例子,用链表来管理商品的库存情况.通过节点存放一个商品对象,该对象具有代号,名称,库存量和单价等属性.

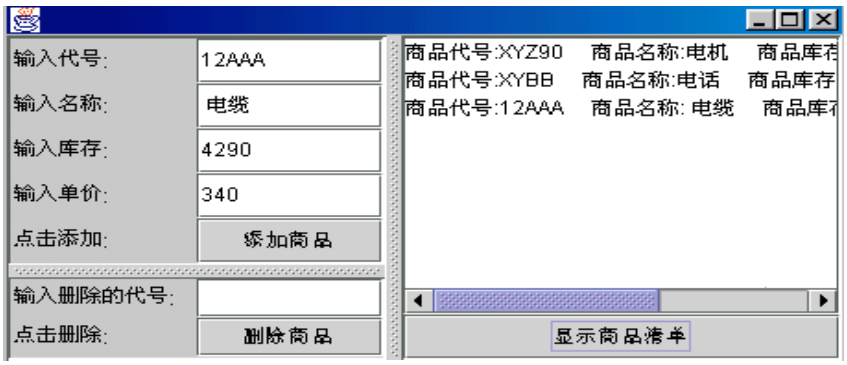


图 26.5 用链表实现的库存管理

例子 效果如图 26.5

```
import java.util.*;import java.awt.event.*;import java.awt.*;
import javax.swing.*;import java.io.*;
class 商品 extends Panel
{String 代号,名称;int 库存;float 单价;
  商品(String 代号,String 名称,int 库存,float 单价)
  {this.代号=代号;this.名称=名称;this.库存=库存;this.单价=单价;
  }
}

class ShowWin extends JFrame implements ActionListener
{  LinkedList goods_list=null;
  JTextField 代号文本框=new JTextField(),
名称文本框=new JTextField(),
           库存文本框=new JTextField(),
单价文本框=new JTextField(),
           删除文本框=new JTextField();
  JButton    b_add=new JButton("添加商品"),
```

```
b_del=new JButton("删除商品"),
             b_show =new JButton("显示商品清单");
  JTextArea   显示区=new JTextArea();
  ShowWin()
  {goods_list=new LinkedList();
   Container con=getContentPane();
   JScrollPane pane=new JScrollPane(显示区);
显示区.setEditable(false);
   JPanel save=new JPanel();save.setLayout(new GridLayout(5,2));
   save.add(new Label("输入代号:"));save.add(代号文本框);
   save.add(new Label("输入名称:"));save.add(名称文本框);
   save.add(new Label("输入库存:"));save.add(库存文本框);
   save.add(new Label("输入单价:"));save.add(单价文本框);
   save.add(new Label("点击添加:"));save.add(b_add);
   JPanel del=new JPanel();del.setLayout(new GridLayout(2,2));
   del.add(new Label("输入删除的代号:"));del.add(删除文本框);
   del.add(new Label("点击删除:"));del.add(b_del);
   JPanel show=new JPanel();show.setLayout(new BorderLayout());
   show.add(pane,BorderLayout.CENTER);show.add(b_show,BorderLayout.SOUTH);
   JSplitPane split_one,split_two;
   split_one=new JSplitPane(JSplitPane.VERTICAL_SPLIT,save,del);
   split_two=new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,true,split_one,show);
   con.add(split_two,BorderLayout.CENTER);
   b_add.addActionListener(this);b_del.addActionListener(this);
   b_show.addActionListener(this);
  }
 public void actionPerformed(ActionEvent e)
 {if(e.getSource()==b_add)
  {String daihao=null,mingcheng=null;int kucun=0;float danjia=0.0f;
   daihao=代号文本框.getText();mingcheng=名称文本框.getText();
   kucun=Integer.parseInt(库存文本框.getText());
   danjia=Float.valueOf(单价文本框.getText()).floatValue();
   商品 goods=new 商品(daihao,mingcheng,kucun,danjia);
   goods_list.add(goods);
   try {FileOutputStream file=new FileOutputStream("goods.txt");
        ObjectOutputStream out=new  ObjectOutputStream(file);
        out.writeObject(goods_list);out.close();
       }
   catch(IOException event){}
```

```
 }
 else if(e.getSource()==b_del)
 {String daihao=删除文本框.getText();
  try {FileInputStream come_in=new FileInputStream("goods.txt");
       ObjectInputStream in=new  ObjectInputStream(come_in);
       goods_list=(LinkedList)in.readObject();in.close();
      }
  catch(ClassNotFoundException event){}
  catch(IOException event){}
  for(int i=0;i<goods_list.size();i++)
   {商品 temp=(商品)goods_list.get(i);
     if(temp.代号.equals(daihao)) {goods_list.remove(i);}
     try  {FileOutputStream file=new FileOutputStream("goods.txt");
           ObjectOutputStream out=new  ObjectOutputStream(file);
            out.writeObject(goods_list);
            out.close();
          }
      catch(IOException event){}
     }
 }
else if(e.getSource()==b_show)
 { 显示区.setText(null);
   try {FileInputStream come_in=new FileInputStream("goods.txt");
        ObjectInputStream in=new  ObjectInputStream(come_in);
        goods_list=(LinkedList)in.readObject();
       }
  catch(ClassNotFoundException event){}
  catch(IOException event){}
  Iterator iter=goods_list.iterator();
  while(iter.hasNext())
   {  商品 te=(商品)iter.next();
      显示区.append("商品代号:"+te.代号+"     ");
      显示区.append("商品名称:"+te.名称+"     ");
      显示区.append("商品库存:"+te.库存+"     ");
      显示区.append("商品单价:"+te.单价+"     ");
      显示区.append("\n");
   }
 }
}
```

```
}
public class LinkListFour
{public  static void main(String args[])
 { ShowWin win=new ShowWin();
   win.setSize(100,100);
   win.setVisible(true);
   win.addWindowListener(new WindowAdapter()
      {public void windowClosing(WindowEvent e)
        {System.exit(0);}});
 }
}
```

注 在上面的例子中,商品类故意扩展了 java.awt 包中的 Panel,因为 Java 提供给我们的绝大多数对象都是所谓序列化的,比如组件等,这是为了保证把对象写入到文件中后,能再把对象正确读回到程序中的缘故.商品类是我们自己写的的类,没有序列化,只要随便扩展一个序列化的类即可.

26.2. 堆栈

堆栈是一种"后进先出"的数据结构,只能在一端进行输入或输出数据的操作.堆栈把第一个放入该堆栈的数据放在最底下,而把后续放入的数据放在已有数据的顶上,如图 26.6 所示.

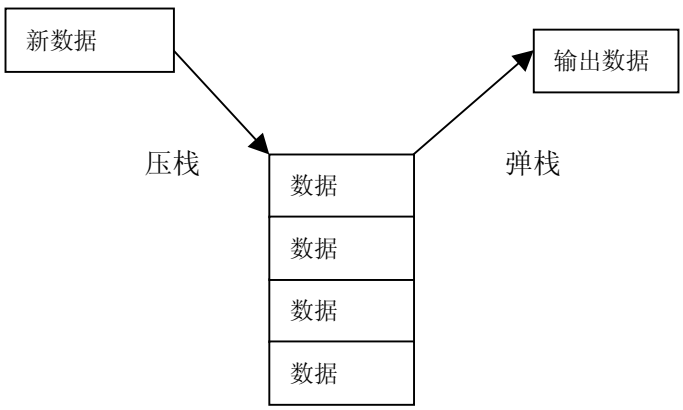


图 26.6 堆栈结构示意图

向堆栈中输入数据的操作称为"压栈",从堆栈中输出数据的操作称为"弹栈".由于堆栈总是在顶端进行数据数据的输入输出操作,所以弹栈是总是输出 删除 最后压入堆栈中的数据,这就是"后进先出"的来历.

使用 java.util 包中的 Stack 类创建一个堆栈对象,堆栈对象可以使用

```
public Object push(Object data);
```

输入数据,实现压栈操作,使用

```
public Object pop();
```

输出数据,实现弹栈操作.使用

```
public boolean empty();
```

判断堆栈是否还有数据,有数据返回 **false** ,否则返回 **true**.使用

```
public Object peek();
```

查看堆栈顶端的数据,但不删除该数据.使用

```
public int search(Object data);
```

获取数据在堆栈中的位置,最顶端的位置是　,向下依次增加,如果堆栈不含此数据,则返回**-1**.

注　由于任何类都是 Object 类的子类,因此可以把任何一个对象压入堆栈.

下面的例子　将数字　,　,　,　,　,**6** 压入堆栈,然后弹出.

例子 5

```
import java.util.*;
class StackOne
{public static void main(String args[])
 {Stack mystack=new Stack();
  mystack.push(new Integer(1)); mystack.push(new Integer(2));
  mystack.push(new Integer(3)); mystack.push(new Integer(4));
  mystack.push(new Integer(5)); mystack.push(new Integer(6));
  while(!(mystack.empty()))
  {Integer temp=(Integer)mystack.pop();
   System.out.print("   "+temp.toString());}
 }
}
```

输出结果是　　.

堆栈是很灵活的数据结构,使用堆栈可以节省内存的开销.比如,递归是一种很消耗内存的算法,我们可以借助堆栈消除大部分递归,达到和递归算法同样的目的.**Fibonacii** 整数序列是我们熟悉的一个递归序列,它的第 **n** 项是前两项的和,第一项和第二项是　.下面的例子用堆栈输出该递归序列的若干项.

例子

```
import java.util.*;
class StackTwo
{public static void main(String args[])
 {Stack mystack=new Stack();
  mystack.push(new Integer(1)); mystack.push(new Integer(1));
  int k=1;
  while(k<=10)
```

```
  for(int i=1;i<=2;i++)
   {Integer F1=(Integer)mystack.pop();int f1=F1.intValue();
    Integer F2=(Integer)mystack.pop();int f2=F2.intValue();
    Integer temp=new Integer(f1+f2);
    System.out.println(""+temp.toString());
    mystack.push(temp);mystack.push(F2);k++;
   }
 }
}
```

注 如果将上述程序中

```
mystack.push(temp);
mystack.push(F2);
```

两个语句的顺序颠倒,输出的结果如何

26.3 树集

树集是有一些节点对象组成的数据结构,节点按着树形一层一层的排列,如下图所示.

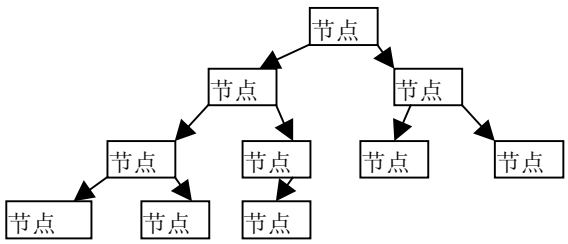


图 26.7 树集结构示意图

用构造方法 TreeSet()创建一个树集

可以使用 java.util 包中的 TreeSet 来创建一个树集,如

```
TreeSet mytree=new TreeSet();
```

然后使用 add 方法为树集添加节点

```
mytree.add("boy");
mytree.add("zoo");
mytree.add("apple");
mytree.add("girl");
```

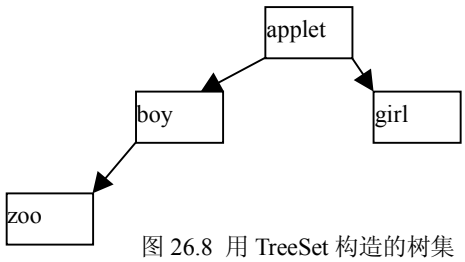


图 26.8 用 TreeSet 构造的树集

和链表不同的是,当使用构造方法 TreeSet()创建树集后,再用 add 方法增加节点时,节点会按其存放的数据的字典序一层一层地依次排列,在同一层中的节点从左到右按字典序递增

排列,下一层的都比上一层的小.mytree 的示意图如下图.

两个字符串对象 s1,s2 可以使用

```
s1.compare(s2);
```

按字典序比较大小,即 s1.compare(s2) 时二者相同,s1.compare(s2)> 时,称 s1 大于 s2,s1.compare(s2)< 时,称 s1 小于 s2.下面的例子使用构造方法 TreeSet()创建了一个树集,并填加了 4 个节点.

例子

```
import java.util.*;
class TreeOne
{public static void main(String args[])
 { TreeSet mytree=new TreeSet();
   mytree.add("boy");mytree.add("zoo");
   mytree.add("apple"); mytree.add("girl");
   Iterator te=mytree.iterator();
   while(te.hasNext())
    System.out.println(""+te.next());
    }
  }
```

输出结果是

```
apple
boy
girl
zoo
```

用构造方法 TreeSet(Comparator c)创建一个树集

但很多对象不适合用字典序进行比较,这时我们在创建树集时可自己规定节点按着什么样的"大小"顺序排列.假如我们有四个学生对象,他们有姓名和成绩,我们想把这四个对象做为一个树集的节点,并按着成绩的高低排列节点,而不是按着姓名的字典序排列节点.

首先创建学生的 Student 类必须实现接口 Comparable.Comparable 接口中有一个方法

```
public int compareTo(Object b)
```

Student 类通过实现这个接口来规定它创建的对象的大小关系,如下所示.

```
class Student  implements Comparable
{ int english=0; String name;
 Student(int e,String n)
```

```
 {english=e;name=n;}
 public int compareTo(Object b)
 { Student st=(Student)b;
   return (this.english-st.english);//按成绩大小排列对象.
 }
}
```

Java 规定　当 a.compareTo(b)　　时,称二者相等　s1.compare(s2)>　时,称 a 大于 b s1.compare(s2)<　时,称 a 小于 b.

然后用带 Comparator 参数的构造方法

```
TreeSet(Comparator c);
```

创建一个树集,通过参数 c 规定树集的节点按怎样的顺序排列,如下所示

```
TreeSet mytree=new TreeSet(new Comparator()
   {public int compare(Object a,Object b)
    {Student stu1=(Student)a;
     Student stu2=(Student)b;
return stu1.compareTo(stu2);
}});
```

注　Comparator 是 java.util 包中的一个接口,compare 是接口中的方法,因此必须给出方法体.

下面的例子　中的树集按着英语成绩从底到高存放四个 Student 对象.

例子

```
import java.util.*;import java.awt.*;
class TreeTwo
{public static void main(String args[])
 { TreeSet mytree=new TreeSet(new Comparator()
   {public int compare(Object a,Object b)
    {Student stu1=(Student)a;Student stu2=(Student)b;
     return stu1.compareTo(stu2);}
   });
   Student st1,st2,st3,st4;
   st1=new Student(90,"zhan ying");st2=new Student(66,"wang heng");
   st3=new Student(86,"Liuh qing");st4=new Student(76,"yage ming");
   mytree.add(st1);mytree.add(st2);mytree.add(st3);mytree.add(st4);
   Iterator te=mytree.iterator();
   while(te.hasNext())
    {Student stu=(Student)te.next();
```

```
        System.out.println(""+stu.name+"  "+stu.english);}
    }
}
class Student  implements Comparable
{ int english=0;String name;
 Student(int e,String n)
 {english=e;name=n;
 }
 public int compareTo(Object b)
 { Student st=(Student)b;
   return (this.english-st.english);
 }
}
```

输出结果是

```
wang  heng  66
yage  ming   76
liuhe  qing   86
zhan  ying    90
```

注 树集中不容许出现大小相等的两个节点,例如,在上述例子中如果再添加语句

```
st5=new Student(76,"keng wenyi"); mytree.add(st5);
```

是无效的.如果允许成绩相同,可把上述例子中 Student 类中的 compareTo 方法更改为

```
public int compareTo(Object b)
 { Student st=(Student)b;
   if((this.english-st.English)==0)
        return 1;
   else
        return (this.english-st.english);
 }
```

注 理论上已经知道,把一个元素插入树集的合适位置要比插入数组或链表中的合适位置效率高.

TreeSet 类的一些常用方法

public boolean add(Object o) 向树集添加加节点,添加成功返回 **true**,否则返回 **false.**
public void clear() 删除所有的节点.
public void contains(Object o) 如果包含节点 o 返回 **true .**
public Object first() 返回根节点,即第一个节点 最小的节点 .
public Object last() 返回最后一个节点 最大的节点 .

public isEmpty()　判断是否是空树集,如果树集不含节点返回 true .
public boolean remove(Object o)　删除节点 o.
public int size()　返回节点数目.

下面的例子用树集实现了某次演出的节目单的安排与管理,按演出时间的先后对节目单排序.

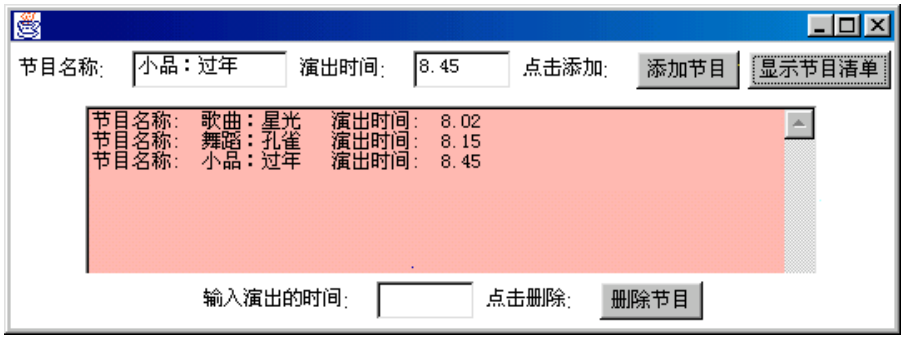


图 26.9　用树集实现的节目单

例子 9 (效果如图 26.9)

```
import java.util.*;import java.awt.event.*;
import java.awt.*;
class 节目  implements Comparable
{String name;double time;
  节目(String 名称,double 演出时间)
  {name=名称;time=演出时间;
  }
  public int compareTo(Object b)
   {节目 item=(节目)b;
    return (int)((this.time-item.time)*1000);
   }
}

 class Win extends Frame implements ActionListener
 {  TreeSet 节目清单=null;
   TextField  名称文本框=new TextField(10),
              时间文本框=new TextField(5),
              删除文本框=new TextField(5);
   Button   b_add=new Button("添加节目"),
            b_del=new Button("删除节目"),
            b_show =new Button("显示节目清单");
   TextArea 显示区=new TextArea();
```

```
 Win()
 {  节目清单=new TreeSet(new Comparator()
            {public int compare(Object a,Object b)
               {节目  item_1=(节目)a;
                节目  item_2=(节目)b;
                return item_1.compareTo(item_2);
               }
             });
  Panel  节目单输入区=new Panel();
         节目单输入区.add(new Label("节目名称:"));
         节目单输入区.add(名称文本框);
         节目单输入区.add(new Label("演出时间:"));
         节目单输入区.add(时间文本框);
         节目单输入区.add(new Label("点击添加:"));
         节目单输入区.add(b_add);
         节目单输入区.add(b_show);
  Panel 节目单删除区=new Panel();
         节目单删除区.add(new Label("输入演出的时间:"));
         节目单删除区.add(删除文本框);
         节目单删除区.add(new Label("点击删除:"));
         节目单删除区.add(b_del);
  Panel  节目单显示区=new Panel();
         节目单显示区.add(显示区);
  显示区.setBackground(Color.pink);
  b_add.addActionListener(this);b_del.addActionListener(this);
  b_show.addActionListener(this);
  add(节目单输入区,"North");add(节目单显示区,"Center");
  add(节目单删除区,"South");
}
public void actionPerformed(ActionEvent e)
{if(e.getSource()==b_add)
 {String 名称=null;double 时间=0.0;
  名称=名称文本框.getText();
  try{时间=Double.valueOf(时间文本框.getText()).doubleValue();
     }
  catch(NumberFormatException ee)
     {时间文本框.setText("请输入代表时间的实数");
     }
  节目 programme=new 节目(名称,时间);
```

```
    节目清单.add(programme);
    showing();
   }
   else if(e.getSource()==b_del)
   {节目 待删除节目=null;
     double time=Double.valueOf(删除文本框.getText()).doubleValue();
     Iterator te=节目清单.iterator();
     while(te.hasNext())
     {节目 item=(节目)te.next();
       if(Math.abs(item.time-time)<=0.000001d)
       {待删除节目=item; }
     }
    if(待删除节目!=null) 节目清单.remove(待删除节目);
    showing();
   }
  else if(e.getSource()==b_show)
   { showing();
   }
  }
  void showing()
  { 显示区.setText(null);
     Iterator iter=节目清单.iterator();
     while(iter.hasNext())
     {节目 item=(节目)iter.next();
      显示区.append("节目名称:"+item.name+"演出时间: "+item.time);
      显示区.append("\n");
     }
  }
}
public class Tree_3
{public  static void main(String args[])
 { Win win=new Win();
   win.setSize(500,250);win.setVisible(true);
   win.addWindowListener(new WindowAdapter()
      {public void windowClosing(WindowEvent e)
        {System.exit(0);}});
 }
}
```

26.4 散列表

散列表是使用相关关键字查找被存储的数据项的一种数据结构,关键字不可以发生逻辑冲突,即不要两个数据项使用相同的关键字,如果出现两个数据项对应相同的关键字,那么,先前散列表中的数据项将被替换.散列表在它需要更多的存储空间时会自动增大容量.例如,如果散列表的装载因子是 0.75,那么当散列表的容量被使用了 75%时,它就把容量增加到原始容量的 倍.对于数组和链表这两种数据结构,如果要查找它们存储的某个特定的元素却不知道它的位置,就需要从头开始访问元素直到找到匹配的为止 如果数据结构中包含很多的元素,就会浪费时间.这时最好使用散列表来存储要查找的数据.

使用 java.util 包中的 Hashtable 类来创建散列表对象,该类的常用方法如下

public Hashtable() 创建具有默认容量和装载因子为 0.75 的散列表.

public Hashtable(int itialCapacity) 创建具有指定容量和装载因子为 0.75 的散列表.

public Hashtable(int initialCapacity,float loadFactor) 创建具有默认容量和指定装载因子散列表.

public void clear() 清空散列表.

public boolean contains(Object o) 判断散列表是否有含有元素 o.

public Object get(Object key) 获取散列表中具有关键字 key 的数据项.

public boolean isEmpty() 判断散列表是否为空.

public Object put(Object key,Object value) 向散列表添加数据项 value 并把关键字 key 关联到数据项 value.

public Object remove(Object key) 删除关键字是 key 的数据项.

public int size() 获取散列表中关键字的数目.

public Enumeration keys() 返回散列表所用的关键字的一个枚举对象.

使用上述的 get 方法可以从散列表中检索某个数据.我们还可以借助 Enumeration 对象实现遍历散列表,一个散列表可以使用 elements()方法获取一个 Enumeration 对象,后者使用 nextElement()方法遍历散列表.

下面的例子 10 使用 Hashtable()创建一个散列表,存放 Student 对象,每个 Student 对象用该对象的学号作为关键字.

例子 10

```
import java.util.*;
class Student
{ int english=0; String  name,number;
 Student(String na,String nu,int e)
 {english=e;name=na;number =nu;}
}
public class HT
{ public static void main(String args[])
  { Hashtable hashtable=new Hashtable();
```

```
    hashtable.put("199901",new Student("199901","王小林",98));
    hashtable.put("199902",new Student("199902","能林茂",70));
    hashtable.put("199903",new Student("199903","多种林",93));
    hashtable.put("199904",new Student("199904","围林蛤",46));
    hashtable.put("199905",new Student("199905","夹贸林",77));
    hashtable.put("199906",new Student("199906","噔林可",55));
    hashtable.put("199907",new Student("199907","降王林",68));
    hashtable.put("199908",new Student("199908","纠林咯",76));
    Student stu=(Student)hashtable.get("199902");//检索一个元素.
    System.out.println(stu.number+"  "+stu.name+"  "+stu.english);
    hashtable.remove("199906"); //删除一个元素
   System.out.println("散列表中现在含有  "+hashtable.size()+"个元素");
     Enumeration enum=hashtable.elements();
     while(enum.hasMoreElements())    //遍历当前散列表.
     {Student s=(Student)enum.nextElement();
      System.out.println(s.number+"  "+s.name+"  "+s.english);
     }
   }
}
```

下面的例子 11 使用散列表实现学生成绩单的输入和查询.

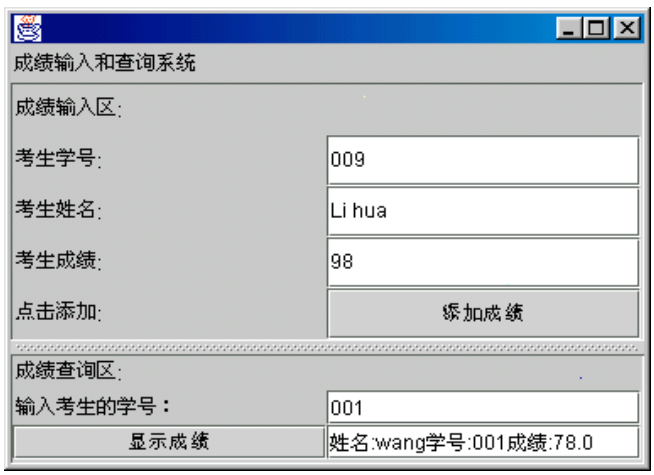


图 26.10 用散列表实现成绩输入和查

例子 11 (效果如图 26.10)

```
import java.util.*;import java.awt.event.*;import java.awt.*;
 import javax.swing.*;import java.io.*;
 class  学生 extends JPanel
```

```
{String 学号,姓名;float 分数;
   学生(String 学号,String 姓名,float 分数)
  {this.学号=学号;this.姓名=姓名;this.分数=分数;
  }
}
class ShowWin extends JFrame implements ActionListener
{ Hashtable hashtable=new Hashtable();
  JTextField 学号文本框=new JTextField(),
             姓名文本框=new JTextField(),
            分数文本框=new JTextField(),
            查询文本框=new JTextField();
  JButton  b_add=new JButton("添加成绩"),
          b_show =new JButton("显示成绩");
  JTextField 成绩显示条=new JTextField();
  ShowWin()
  {Container con=getContentPane();
   JPanel 成绩输入区=new JPanel();
          成绩输入区.setLayout(new GridLayout(5,2));
          成绩输入区.add(new Label("成绩输入区:"));
          成绩输入区.add(new Label());
          成绩输入区.add(new Label("考生学号:"));
          成绩输入区.add(学号文本框);
          成绩输入区.add(new JLabel("考生姓名:"));
          成绩输入区.add(姓名文本框);
          成绩输入区.add(new Label("考生成绩:"));
          成绩输入区.add(分数文本框);
          成绩输入区.add(new Label("点击添加:"));
          成绩输入区.add(b_add);
   JPanel  查询显示区=new JPanel();
          查询显示区.setLayout(new GridLayout(3,2));
          查询显示区.add(new Label("成绩查询区:"));
          查询显示区.add(new Label());
          查询显示区.add(new Label("输入考生的学号  "));
          查询显示区.add(查询文本框);
          查询显示区.add(b_show);
          查询显示区.add(成绩显示条);
  JSplitPane split;
split=new JSplitPane(JSplitPane.VERTICAL_SPLIT,成绩输入区,查询显示区);
   con.add(split,BorderLayout.CENTER);
```

```
    con.add(new Label("成绩输入和查询系统"),BorderLayout.NORTH);
    b_add.addActionListener(this);b_show.addActionListener(this);
   }
  public void actionPerformed(ActionEvent e)
  {if(e.getSource()==b_add)
   {String 学号=null,姓名=null;float 分数=0.0f;
     try {学号=学号文本框.getText();
          姓名=姓名文本框.getText();
         }
     catch(NullPointerException ee)
        { 学号文本框.setText("请输入学号");
          姓名文本框.setText("请输入姓名");
        }
     try{分数=Float.valueOf(分数文本框.getText()).floatValue();}
     catch(NumberFormatException ee)
            {分数文本框.setText("请输入数字字符");}
     学生 stu=new 学生(学号,姓名,分数);
     hashtable.put(学号,stu);
     try {FileOutputStream file=new FileOutputStream("score.txt");
          ObjectOutputStream out=new  ObjectOutputStream(file);
          out.writeObject(hashtable);  out.close();
         }
         catch(IOException event){}
    }
  else if(e.getSource()==b_show)
   { String temp=null;
     temp=查询文本框.getText();
     成绩显示条.setText(null);
     try {FileInputStream come_in=new FileInputStream("score.txt");
          ObjectInputStream in=new  ObjectInputStream(come_in);
          hashtable=(Hashtable)in.readObject();in.close();
         }
    catch(ClassNotFoundException event){}
    catch(IOException event){System.out.println("文件无法读出");}
   学生 s=(学生)hashtable.get(temp);
   成绩显示条.setText("姓名:"+s.姓名+"学号:"+s.学号+"成绩:"+s.分数);
   }
  }
}
```

```
public class HT_2
{public  static void main(String args[])
 { ShowWin win=new ShowWin();
   win.setSize(100,100); win.setVisible(true);
   win.addWindowListener(new WindowAdapter()
      {public void windowClosing(WindowEvent e)
        {System.exit(0);}});
 }
}
```

26.5 向量

Java 的 java.util 包中的 Vector 类负责创建一个向量对象.如果你已经学会使用数组,那么很容易就会使用向量.当我们创建一个向量时不用象数组那样必须要给出数组的大小.向量创建后,例如,Vector a=new Vector()　a 可以使用 add(Object o)把任何对象添加到向量的末尾,向量的大小会自动的增加.可以使用 add(int index ,Object o)把一个对象追加到该向量的指定位置.向量 a 可以使用 elementAt(int index)获取指定索引处的向量的元素　索引初始位置是 0　a 可以使用方法 size()获取向量所含有的元素的个数.另外,与数组不同的是向量的元素类型不要求一致.需要注意的是,当你把某一种类型的对象放入一个向量后,数据被默认为是 Object 对象,因此当向量中取出一个元素时应用强制类型转化运算符转化为原来的类型.

Vector 类的常用方法

public void add(Object o) 将对象 o 添加到向量的末尾.

public void add(int index Object o) 将对象 o 插入到向量的指定位置.

public void addElements(Object o)　将对象 o 添加到向量的末尾.

public boolean contains(Object o)　判断对象 o 是否为向量的成员.

public Object elementAt(int index)　获取指定位置处的成员.

public Object get(int index)　获取此向量指定位置处的成员.

public Object firstElement()　获取此向量的第一个成员.

public Object lastElement()　获取此向量的最后一个成员.

public int indexOf(Obkect o)　获取对象 o 在此向量中首次出现的位置.

public int indexOf(Obkect o,int index)　从指定位置查找对象 o 在此向量中首次现的位置.

public int lastIndexOf(Object o)　获取对象 o 在此向量中最后出现的位置.

public int lastIndexOf(Object o,int index)　对象 o 在此向量位置 index 之前最后出现的位置.

public Object remove(int index)　从此向量中删除指定位置处的成员,并返回这个成员.

public void removeAllElements()　删除向量的所有成员.

public boolean removeElement(Object o)　删除第一次出现的成员 o.

public boolean removeElementAt(int index)　删除指定位置处的成员.

public void set(int index,Object o)　把指定位置处的成员用 o 替换掉.
public void setElementAt(Object o,int index)　把指定位置处的成员用 o 替换掉.
public Enumeration elements()　获取一个枚举对象.

在下面的例子 12 中将几个不同的对象添加到向量.

例子 12

```
import java.util.*;
class Example26_12
 {public static void main(String args[])
  { Vector vector=new Vector(); Date date=new Date();
    vector.add(new Integer(1));vector.add(new Float(3.45f));
    vector.add(new Double(7.75));vector.add(new Boolean(true));
    vector.add(date);
    System.out.println(vector.size());
    Integer number1=(Integer)vector.get(0);
    System.out.println("向量的第 1 个元素　"+number1.intValue());
    Float number2=(Float)vector.get(1);
    System.out.println("向量的第 2 个元素　"+number2.floatValue());
    Double number3=(Double)vector.get(2);
    System.out.println("向量的第 3 个元素　"+number3.doubleValue());
    Boolean number4=(Boolean)vector.get(3);
    System.out.println("向量的第 4 个元素　"+number4.booleanValue());
    date=(Date)vector.lastElement();
    System.out.println("向量的第 5 个元素　"+date.toString());
    if(vector.contains(date))
      System.out.println("ok");
  }
 }
```

下面的例子 13 使用向量技术改进了十八章 18.4 中用鼠标自由作画的程序,在下面的例子中无论鼠标拖动的怎样快,所画出的点都是很连续的.每当拖动鼠标时,我们把获得的鼠标位置坐标存入一个向量,然后依次把向量的点连成直线.

例子 13

```
import java.applet.*;
import java.awt.*;import java.util.*;
import java.awt.event.*;
class Point
{int x,y;
```

```
 Point(int x,int y)
 {this.x=x;this.y=y;
 }
}
public class Example26_13 extends Applet
                        implements MouseMotionListener,MouseListener
{ int x=-1,y=-1;
   Vector v=null;int n=1;
  public void init()
 { setBackground(Color.green);
   addMouseMotionListener(this);  addMouseListener(this);
    v=new Vector();
 }
 public void paint(Graphics g)
 {if(x!=-1&&y!=-1)
  {  n=v.size();
    for(int i=0;i<n-1;i++)
       {Point p1=(Point)v.elementAt(i);
        Point p2=(Point)v.elementAt(i+1);
        g.drawLine(p1.x,p1.y,p2.x,p2.y);
       }
   }

 }
public void mouseDragged(MouseEvent e)
 { x=(int)e.getX();y=(int)e.getY();
   Point p=new Point(x,y);
   v.addElement(p);
   repaint();
 }
 public void mouseMoved(MouseEvent e)
  {}
  public void mousePressed(MouseEvent e){}
  public void mouseReleased(MouseEvent e)
    {v.removeAllElements();}
  public void mouseEntered(MouseEvent e){}
  public void mouseExited(MouseEvent e){}
  public void mouseClicked(MouseEvent e){}
  public void update(Graphics g)
```

```
    { paint(g);
    }
}
```

习题二十六

1　改进例子 4,增加查询商品和修改库存量的功能.

2　使用堆栈结构输出 a_n 的若干项,其中 $a_n=2a_{n-1}+2a_{n-2}$, $a_1=3$, $a_2=8$.

3　编写一个程序,用散列表实现学生成绩单的存储与查询,并将若干个查询结果存放到一个树集中,通过树集实现对查询结果的自动排序,并将排序结果用表格显示出来.

附录　JDK 常见命令

如果你在安装 JDK　例如,JDK1.2.2　时没有改变默认的安装路径的话,JDK 命令被安装在　c:\jdk1.2.2\bin 目录下.

1　编译器　javac.exe

用来编译源文件,生成字节码文件.例如,

```
javac MyFile.java
```

javac 可以使用参数-d 指定生成的字节吗文件所在的目录,否则 javac 在当前目录生成字节码文件.

```
javac -d  F:\Tom\1000  MyFile.java
```

2　解释器　java.exe,javaw.exe

用来运行 java 应用程序,其中 javaw 是后台运行应用程序.例如,

```
java  Program
javaw  Program
```

3. 小应用程序浏览器　appletviewer.exe

用来调试小应用程序, 例如,

```
appletviewer  Boy.html
```

4. 文挡生成器　javadoc.exe

javadoc 用 html 格式生成原文件中类的结构信息.

假设有如下源文件

```
import java.awt.*;
public class MyButton extends Button
 {
 }
```

将上述文件保存在某一文件夹中,比如:　D:\test.然后用 javadoc 生成文挡

```
javadoc MyButton.java
```

这时在文件夹 test 中将生成有关 public 类 MyButton 的结构信息,通过该文挡,可以查看 MyButton 类,Button 类以及它们父类中的方法的名字,如图 A.1,A.2 所示.

使用 javadoc 时,也可以使用参数-d 指定生成文挡所在的目录,例如,

```
javadoc  -d  F:\gxy\book  MyButton.java
```

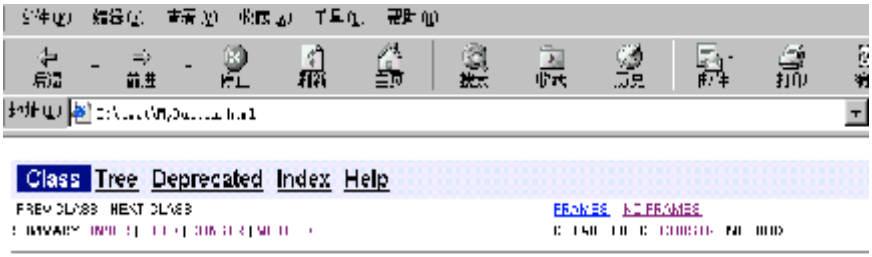


图 A.1　Javadoc 生成的文挡

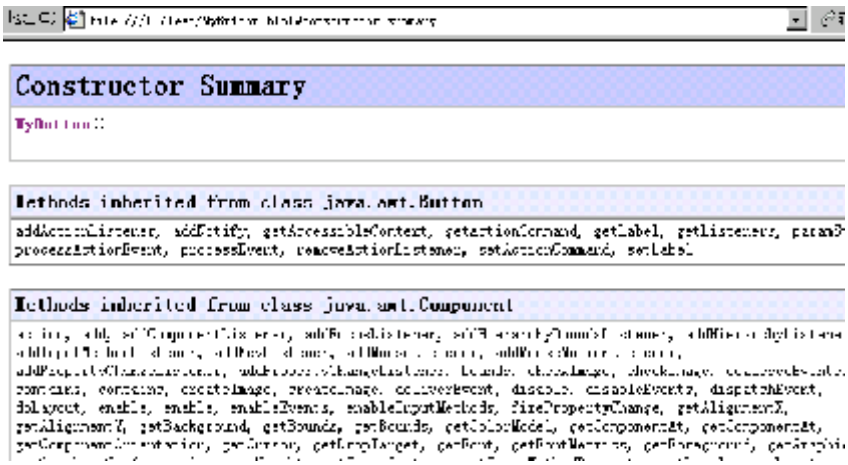


图 A.2　Javadoc 生成的文挡

5　反编译器　javap.exe

反编译器可以用来分析字节吗文件,例如

javap　Example226

```
javap java.awt.Button
```

6 环境变量

安装 **jdk1.2** 及之后的版本,不需用户设置环境变量.如果你的应用程序编译正确,并包含正确的 **main** 方法,但程序执行时总是提示找不到类,你就要重新设置环境变量

```
set  classpath=c:\jkd1.2.2\jre\lib\rt.jar; .;
```