第10章　部署应用程序和applet

- ▲ JAR文件
- ▲ Java Web Start
- ▲ Applet
- ▲ 应用程序配置的存储

到目前为止，我们已经能够熟练地使用Java程序语言的大部分特性，并且对Java图形编程的基本知识也有所了解。现在准备创建提交给用户的应用程序，至此需要知道如何将这些应用程序进行打包，以便部署到用户的计算机上。传统的部署方式是使用applet，这应该归功于在Java出现的最初几年中对其给予的大肆吹捧。applet是一种特殊的Java程序，它允许通过网络下载，并可以在浏览器中运行。其目的在于让用户不再为安装软件烦恼，并且可以通过支持Java的计算机或者其他具有Internet连接的设备使用这些软件。

但是由于种种原因，applet并没有实现上述目标。因此，本章将首先介绍打包应用程序的指令，然后再论述Java Web Start机制，这是一种对基于互联网的应用程序发布的方式，最后介绍一下applet，并展示一下可能仍然会使用的环境。

本章还要讨论一下应用程序如何存储配置信息和用户参数。

10.1　JAR文件

在将应用程序进行打包时，使用者一定希望仅提供给其一个单独的文件，而不是一个含有大量类文件的目录，Java归档（JAR）文件就是为此目的而设计的。一个JAR文件既可以包含类文件，也可以包含诸如图像和声音这些其他类型的文件。此外，JAR文件是压缩的，它使用了大家熟悉的ZIP压缩格式。

提示：Java SE 5.0提供了一种新的压缩方案，被称为pack200，这是一种较通常的ZIP压缩算法更加有效的压缩类文件的方式。Sun声称，对类文件的压缩率接近90%。有关更加详细的信息请参看网站http://java.sun.com/javase/6/docs/technotes/guide/deployment/deployment-guide/pack200.html。

可以使用jar工具制作JAR文件（在默认的JDK安装中，位于jdk/bin目录下）。创建一个新的JAR文件应该使用的常见命令格式为：

```
jar cvf JARFileName File1 File2 . . .
```

例如：

```
jar cvf CalculatorClasses.jar *.class icon.gif
```

通常，jar命令的格式如下：

```
jar options File1 File2 . . .
```

表10-1列出了所有jar程序的可选项。它们类似于UNIX tar命令的选项。

表10-1 .jar程序选项

选 项	说 明
c	创建一个新的或者空的存档文件并加入文件。如果指定的文件名是目录，jar程序将会对它们进行递归处理
C	暂时改变目录，例如： `jar cvf JARFileName.jar -C classes *.class` 改变classes子目录，以便增加这些类文件
e	在清单文件中创建一个条目（请参看可执行的JAR文件）
f	将JAR文件名指定为第二个命令行参数。如果没有这个参数，jar命令会将结果写到标准输出上（在创建JAR文件时）或者从标准输入中读取它（在解压或者列出JAR文件内容时）
i	建立索引文件（用于加快对大型归档的查找）
m	将一个清单文件（manifest）添加到JAR文件中。清单是对存档内容和来源的说明。每个归档有一个默认的清单文件。但是，如果想验证归档文件的内容，可以提供自己的清单文件
M	不为条目创建清单文件
t	显示内容表
u	更新一个已有的JAR文件
v	生成详细的输出结果
x	解压文件。如果提供一个或多个文件名，只解压这些文件；否则，解压所有文件
0	存储，不进行ZIP压缩

10.1.1 清单文件

除了类文件、图像和其他资源外，每个JAR文件还包含一个用于描述归档特征的清单文件(manifest)。

清单文件被命名为 MANIFEST.MF，它位于JAR文件的一个特殊META-INF 子目录中。最小的符合标准的清单文件是很简单的：

```
Manifest-Version: 1.0
```

复杂的清单文件可能包含更多条目。这些清单条目被分成多个节。第一节被称为主节(main section)。它作用于整个JAR文件。随后的条目用来指定已命名条目的属性，这些已命名的条目可以是某个文件、包或者URL。它们都必须起始于名为Name的条目。节与节之间用空行分开。例如：

```
Manifest-Version: 1.0
描述这个归档文件的行

Name: Woozle.class
描述这个文件的行

Name: com/mycompany/mypkg/
描述这个包的行
```

要想编辑清单文件，需要将希望添加到清单文件中的行放到文本文件中，然后运行：

```
jar cfm JARFileName ManifestFileName . . .
```

例如，要创建一个包含清单的JAR文件，应该运行：

```
jar cfm MyArchive.jar manifest.mf com/mycompany/mypkg/*.class
```

要想更新一个已有的JAR文件的清单，则需要将增加的部分放置到一个文本文件中，然后执行下列命令：

```
jar ufm MyArchive.jar manifest-additions.mf
```

注释：请参看http://java.sun.com/javase/6/docs/technotes/guides/jar获得有关JAR文件和清单文件格式的更多信息。

10.1.2 可运行JAR文件

在Java SE 6中，可以使用jar命令中的e选项指定程序的条目点，即通常需要在调用Java程序加载器时指定的类：

```
jar cvfe MyProgram.jar com.mycompany.mypkg.MainAppClass files to add
```

用户可以简单地通过下面命令来启动应用程序：

```
java -jar MyProgram.jar
```

在旧的JDK版本中，必须指定应用程序的主类，下面是这个处理命令：

```
Main-Class: com.mycompany.mypkg.MainAppClass
```

不要将扩展名.class添加到主类名中。然后运行jar命令：

```
jar cvfm MyProgram.jar mainclass.mf files to add
```

警告：清单文件的最后一行必须以换行符结束。否则，清单文件将无法被正确地读取。常见的错误是创建了一个只包含Main-Class而没有行结束符的文本文件。

根据操作系统的配置，可以通过双击JAR文件图标来启动应用程序。下面是各种操作系统的操作方式：

- 在Windows平台中，Java运行时安装器将建立一个扩展名为.jar的文件与javaw -jar命令相关联来启动文件（与java命令不同，javaw命令不打开shell窗口）。
- 在Solaris平台中，操作系统能够识别JAR文件的“魔数”格式，并用java -jar命令启动它。
- 在Mac OS X平台中，操作系统能够识别.jar 扩展名文件。当双击JAR文件时就会执行Java程序可以运行。

无论怎样，人们对JAR文件中的Java程序与本地文件有着不同的感觉。在Windows平台中，可以使用第三方的打包器工具将JAR文件转换成Windows可执行文件。打包器是一个大家熟知的扩展名为.exe的Windows程序，它可以查找和加载Java虚拟机（JVM），或者在没有找到JVM时告诉用户应该做些什么。有许多商业的和开放的原始产品，比如，JSmooth（http://jsmooth.sourceforge.net）和Launch4J（http://launch4j.sourceforge.net）。开放的原始安装生成器IzPack（http://izpack.org）还包含了一个本地加载器。有关这个问题的更加详细的信息请参看网站http://www.javalobby.org/articles/java2exe。

在Macintosh平台中，这种情形处理起来会容易一些。应用程序打包工具MRJAppBuilder可以将JAR文件转换成一个顶级的Mac程序。有关更加详细的信息请参看http://java.sun.com/developer/technicalArticles/JavaLP/JavaToMac3。

10.1.3 资源

在applet和应用程序中使用的类通常需要使用一些相关的数据文件，例如：

- 图像和声音文件。
- 带有消息字符串和按钮标签的文本文件。
- 二进制数据文件，例如，描述地图布局的文件。

在Java中，这些关联的文件被称为资源（resource）。

注释：在Windows中，术语“资源”有着更加特殊的含义。Windows资源也是由图像、按钮标签等组成，但是它们都附属于可执行文件，并通过标准的程序设计访问。相比之下，Java资源作为单独的文件存储，并不是作为类文件的一部分存储。对资源的访问和解释由每个类自己胜任。

例如，AboutPanel类显示了一条信息，如图10-1所示。

图10-1 显示来自于JAR文件的资源

当然，在面板中的书名和版权年限将会在出版下一版图书时发生变化。为了易于追踪这个变化，希望将文本放置在一个文件中，而不是以字符串的形式硬写到代码中。

但是应该将about.txt这样的文件放在哪儿呢？显然，将它与其他程序文件一起放在JAR文件中是最方便的。

类加载器知道如何搜索类文件，直到在类路径、存档文件或web服务器上找到为止。利用资源机制，对于非类文件也可以同样方便地进行操作。下面是必要的步骤：

1）获得具有资源的Class对象，例如，AboutPanel.class。

2）如果资源是一个图像或声音文件，那么就需要调用getresource (filename)获得作为URL的资源位置，然后利用getImage或getAudioClip方法进行读取。

3）与图像或声音文件不同，其他资源可以使用getResourceAsStream方法读取文件中的数据。

重点在于类加载器可以记住如何定位类，然后在同一位置查找关联的资源。

例如，要想利用about.gif图像文件制作图标，可以使用下列代码：

```
URL url = ResourceTest.class.getResource("about.gif");
Image img = Toolkit.getDefaultToolkit().getImage(url);
```

这段代码的含义是“在找到ResourceTest类的地方定位about.gif文件”。

要想读取about.txt文件，可以使用下列命令：

```
InputStream stream = ResourceTest.class.getResourceAsStream("about.txt");
Scanner in = new Scanner(stream);
```

除了可以将资源文件与类文件放在同一个目录中外，还可以将它放在子目录中。可以使用下面所示的层级资源名：

```
data/text/about.txt
```

这是一个相对的资源名，它会被解释为相对于加载这个资源的类所在的包。注意，必须使用“/”作为分隔符，而不要理睬存储资源文件的系统实际使用哪种目录分隔符。例如，在Windows文件系统中，资源加载器会自动地将“/”转换成“\”。

一个以“/”开头的资源名被称为绝对资源名。它的定位方式与类在包中的定位方式一样。例如，资源

```
/corejava/title.txt
```

定位于corejava目录下（它可能是类路径的一个子目录，也可能位于JAR文件中，对applet来说在web服务器上）。

文件的自动装载是利用资源加载特性完成的。没有标准的方法来解释资源文件的内容。每个程序必须拥有解释资源文件的方法。

另一个经常使用资源的地方是程序的国际化。与语言相关的字符串，如消息和用户界面标签都存放在资源文件中，每种语言对应一个文件。国际化API（internationalization API）将在卷II的第5章中进行讨论。这些API提供了组织和访问本地化文件的标准方法。

例10-1显示了这个程序的原代码。这个程序演示了资源加载。编译、创建JAR文件和执行这个程序的命令是：

```
javac ResourceTest.java
jar cvfm ResourceTest.jar ResourceTest.mf *.class *.gif *.txt
java -jar ResourceTest.jar
```

将JAR文件移到另外一个不同的目录中，再运行它，以便确认程序是从JAR文件中，而不是从当前目录中读取的资源。

例10-1 ResourceTest.java

```
1. import java.awt.*;
2. import java.io.*;
3. import java.net.*;
4. import java.util.*;
5. import javax.swing.*;
6.
7. /**
8.  * @version 1.4 2007-04-30
9.  * @author Cay Horstmann
10.  */
11. public class ResourceTest
12. {
13.    public static void main(String[] args)
14.    {
15.       EventQueue.invokeLater(new Runnable()
16.          {
17.             public void run()
18.             {
19.                ResourceTestFrame frame = new ResourceTestFrame();
20.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21.                frame.setVisible(true);
22.             }
23.          });
24.    }
```

```
25. }
26.
27. /**
28.  * A frame that loads image and text resources.
29.  */
30. class ResourceTestFrame extends JFrame
31. {
32.    public ResourceTestFrame()
33.    {
34.       setTitle("ResourceTest");
35.       setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
36.       URL aboutURL = getClass().getResource("about.gif");
37.       Image img = Toolkit.getDefaultToolkit().getImage(aboutURL);
38.       setIconImage(img);
39.
40.       JTextArea textArea = new JTextArea();
41.       InputStream stream = getClass().getResourceAsStream("about.txt");
42.       Scanner in = new Scanner(stream);
43.       while (in.hasNext())
44.          textArea.append(in.nextLine() + "\n");
45.       add(textArea);
46.    }
47.
48.    public static final int DEFAULT_WIDTH = 300;
49.    public static final int DEFAULT_HEIGHT = 300;
50. }
```

API `java.lang.Class 1.0`

- `URL getResource(String name) 1.1`
- `InputStream getResourceAsStream(String name) 1.1`

找到与类位于同一位置的资源，返回一个可以加载资源的URL或者输入流。如果没有找到资源，则返回null，而且不会抛出异常或者发生I/O错误。

10.1.4 密封

在第4章曾经提到过，可以将Java包密封（seal）以保证不会有其他的类加入到其中。如果在代码中使用了包可见的类、方法和域，就可能希望密封包。如果不密封，其他类就有可能放在这个包中，进而访问包可见的特性。

例如，如果密封了com.mycompany.util包，就不能用下面的语句顶替密封包之外的类：

```
package com.mycompany.util;
```

要想密封一个包，需要将包中的所有类放到一个JAR文件中。在默认情况下，JAR文件中的包是没有密封的。可以在清单文件的主节中加入下面一行：

```
Sealed: true
```

来改变全局的默认设定。对于每个单独的包，可以通过在JAR文件的清单中增加一节，来指定是否想要密封这个包。例如：

```
Name: com/mycompany/util/
Sealed: true
```

```
Name: com/mycompany/misc/
Sealed: false
```

要想密封一个包，需要创建一个包含清单指令的文本文件。然后用常规的方式运行jar命令：

```
jar cvfm MyArchive.jar manifest.mf files to add
```

10.2 Java Web Start

Java Web Start是一项在Internet上发布应用程序的技术。Java Web Start 应用程序包含下列主要特性：

- Java Web Start应用程序一般通过浏览器发布。只要Java Web Start应用程序下载到本地就可以启动它，而不需要浏览器。
- Java Web Start应用程序并不在浏览器窗口内。它将显示在浏览器外的一个属于自己的框架中。
- Java Web Start应用程序不使用浏览器的Java实现。浏览器只是在加载Java Web Start 应用程序描述符时启动一个外部应用程序。这与启动诸如Adobe Acrobat 或 RealAudio这样的辅助应用程序的所使用的机制一样。
- 数字签名应用程序可以被赋予访问本地机器的任意权限。未签名的应用程序只能运行在“沙箱”中，它可以阻止具有潜在危险的操作。

要想准备一个通过Java Web Start 发布的应用程序, 应该将其打包到一个或多个JAR文件中。然后创建一个Java Network Launch Protocol (JNLP)格式的描述符文件。将这些文件放置在web服务器上。

还需要确保web服务器对扩展名为.jnlp的文件报告一个application/x-java-jnlp-file 的MIME类型（浏览器利用MIME类型确定启动哪一种辅助应用程序）。有关更详细的信息请参看web服务器文档。

提示：要想体检Java Web Start，需要从jakarta.apache.org/tomcat上安装Tomcat。Tomcat是一个servlets和 JSP页面的容器，也提供网页服务。它被预配置为服务于JNLP文件所对应的MIME类型。

试着用Java Web Start发布第9章中开发的计算器应用程序。步骤如下：

1）编译Calculator.java.

2）准备一个含有下列内容的清单文件 Calculator.mf

```
Main-Class: Calculator
```

3）使用下列命令创建一个JAR 文件：

```
jar cvfm Calculator.jar Calculator.mf *.class
```

4）使用下列内容准备启动文件 Calculator.jnlp：

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+" codebase="http://localhost:8080/calculator/" href="Calculator.jnlp">
  <information>
    <title>Calculator Demo Application</title>
```

```
    <vendor>Cay S. Horstmann</vendor>
    <description>A Calculator</description>
    <offline-allowed/>
  </information>
  <resources>
    <j2se version="1.5.0+"/>
    <jar href="Calculator.jar"/>
  </resources>
  <application-desc/>
</jnlp>
```

注意，版本号必须是1.5.0，而不是5.0。在Java SE 6中，标记名字时可以用java代替j2se。启动文件格式很容易理解，无需说明。要想了解全部的规范，请参看http://java.sun.com/products/javawebstart/docs/developersguide.htm。

5）如果使用Tomcat，则在Tomcat安装的根目录上创建一个目录 tomcat/webapps/calculator。创建子目录 tomcat/webapps/calculator/WEB-INF，并且将最小的web.xml文件放置在WEB-INF子目录下：

```
<?xml version="1.0" encoding="utf-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
       http://java.sun.com/xml/ns/j2ee/web-app_2_5.xsd">
</web-app>
```

6）将JAR文件和启动文件放置在web服务器上，以便于URL与JNLP文件中的codebase条目匹配。如果使用Tomcat，把它们放置在tomcat/webapps/calculator目录中。

7）通过检查application/x-java-jnlp-file MIME类型与javaws应用程序的关联情况来确保浏览器已经为Java Web Start进行了配置。如果安装了JDK，配置将自动完成。

8）启动Tomcat。

9）将浏览器指向JNLP文件。例如，如果使用Tomcat，将指向 http://localhost:8080/calculator/Calculator.jnlp。

10）可以看到Java Web Start的启动窗口，如图10-2所示。

11）稍后，计算器就会出现，所带的边框表明这是一个Java应用程序，如图10-3所示。

图10-2 启动Java Web Start

图10-3 Java Web Start发布的计算器

12）当再次访问JNLP文件时，应用程序将从缓存中取出。可以利用Java插件控制面板查看缓存内容，如图10-4所示。在JDK 5.0中，applet和Java Web Start应用程序都使用这个控制面板。在Windows中，在Windows控制面板中可以看到Java插件控制面板。在Linux下，可以运行jdk/jre/bin/ControlPanel。

图10-4 应用程序缓存

警告：如果在测试JNLP配置期间不想运行web服务器，可以通过执行下列命令临时性地覆盖启动文件中的URL：

```
javaws -codebase file://programDirectory JNLPfile
```

例如，在UNIX中，可以从包含JNLP文件的目录中直接地发出下列命令：

```
javaws -codebase file://`pwd` WebStartCalculator.jnlp
```

当然，没有告诉用户再次运行应用程序时要启动缓存视图。可以利用安装器安装桌面和菜单的快捷键。将下列代码添加到JNLP文件中：

```
<shortcut>
   <desktop/>
   <menu submenu="Accessories"/>
</shortcut>
```

当用户首次下载应用程序时，将会显示“ desktop integration warning ”，如图10-5所示。

还应该为菜单快捷键和启动屏幕提供一个图标。Sun建议提供32×32和64×64的图标。把这些图标文件与JNLP和JAR文件一起放在web服务器上。将下列代码添加到JNLP文件的information节中：

```
<icon href="calc_icon32.png" width="32" height="32" />
<icon href="calc_icon64.png" width="64" height="64" />
```

注意，这些图标与应用程序图标无关。如果想让应用程序也有图标，需要将一个独立的图标图像添加到JAR文件中，并在框架类中调用thesetIconImage方法。请看示例10-1。

图10-5 桌面综合警告

10.2.1 沙箱

无论何时加载远程网站上代码并在本地执行，安全都是至关重要的问题。点击一个链接可以启动Java Web Start应用程序。访问一个网页时，其中的所有applet也会自动地启动。如果在点击一个链接，或者访问一个网页时，在用户的机器上能够安装任意的代码，那么犯罪分子就可能在此时窃取机密信息、读取财务数据或接管用户机器来发送广告。

为了确保Java技术不会被邪恶目的所利用，Java有一套精心设计的安全模型，具体内容在卷II中论述。安全管理器（security manager）将检查有权使用所有系统的资源。在默认情况下，只允许执行那些无害的操作，要想允许执行其他的操作，代码必须得到数字签名，用户必须通过数字认证。

在所有平台上，远程代码可以做什么呢？它可以显示图像、播放音乐、获得用户的键盘输入和鼠标点击，以及将用户的输入送回加载代码所在的主机。这些功能足以能够显示信息和图片，或者获得用户为订单所输入的信息。这种受限制的执行环境称为沙箱（sandbox）。在沙箱中运行的代码不能修改或查看用户系统。

特别是，在沙箱中的程序有下列限制：

- 不能运行任何本地的可执行程序。
- 不能从本地计算机文件系统中读取任何信息，也不能往本地计算机文件系统中写入任何信息。
- 不能查看除Java版本信息和少数几个无害的操作系统详细信息外的任何有关本地计算机的信息。特别是，在沙箱中的代码不能查看用户名、e-mail地址等信息。
- 远程加载的程序不能与除下载程序所在的服务器之外的任何主机通信，这个服务器被称为源主机（originating host）。这条规则通常称为“远程代码只能与家人通话。”这条规则将确保用户不会被代码探查到内部网络资源（在Java SE 6中，Java Web Start应用程序可以与其他网络连接，但必须得到程序用户的同意）。
- 所有弹出式窗口都会带一个警告消息。这条消息起到了安全的作用，以确保用户不会为本地应用程序弄错窗口。令人担心的是，一个可信赖的用户可以访问网页，并被蒙骗运行远程代码，然后输入密码或信用卡号，这些信息将被送回服务器。在早期的JDK版本

中，有个消息会令人害怕：“Untrusted Java Applet Window（不可信赖的Java Applet Window）”。后来的JDK版本将这个警告消息的口气缓和了一些“Unauthenticated Java Applet Window（未获认证的Java Applet Window）”，随后是“Warning：Java Applet Window（警告：Java Applet Window）”。现在只提示“Java Applet Window”或“Java Application Window”。

10.2.2 签名代码

沙箱限制在很多情况下显得过于严格。例如，在公司内部网中，可能设想让某个Web Start应用程序或某个applet访问本地文件。可以为某个特定的应用程序赋予特定的权限来进行具体的控制。有关这些内容将在卷II的第9章中讨论。当然，应用程序可以直接请求拥有一个桌面应用程序所许可的全部权限，相当多的Java Web Start应用程序就是这样做的。只要将下列标签添加到JNLP文件中就可以达到这个目的：

```
<security>
   <all-permissions/>
</security>
```

要想在沙箱外运行，java Web Start应用程序的JAR文件必须进行数字签名。签名的JAR文件将携带一个可以证明签名者身份的证书。加密技术可以确保这种证书不可被伪造，并且检测任何企图破坏签名文件的行为。

例如，假设接到一个由yWorks GmbH开发且数字签名的应用程序，它使用了由Thawte签发的证书，如图10-6所示。当收到这个应用程序时，将要确定下列内容：

图10-6 安全认证书

1）代码确实被签名，且没有被第三方破坏。

2）由yWorks签名。

3）认证由Thawte签发（Java Web Start知道应该如何核查来自Thawte和少数其他卖方的认证）。

很不幸，这些是知道的全部内容，而并不清楚代码固有的安全性。实际上，如果点击“More Information”链接，则会被告之应用程序在没有Java提供的常规安全限制下运行。应该安装和运行应用程序吗？这将取决于yWorks GmbH对你的信任。

从一个提供证书的卖方那里获得一个证书每年需要花费数百元。很多开发者直接生成自己的证书，并利用这些证书为代码签名。当然，Java Web Start无法准确地核查这些证书。在收到这样一个应用程序时，可以知道：

1）代码确实被签名，且没有被第三方破坏。

2）有些代码被签名，但Java Web Start无法确认哪个代码已经被签名。

这没有任何价值，因为任何声称自己是作者的人都可以先破坏代码，然后再签名。然而，Java Web Start完全陶醉于得到了正式批准的证书（如图10-7所示）。从理论上讲，可以用其他方式验证证书，但是，几乎没有用户能够有这样的技术头脑。

图10-7 不安全的证书

当然，很多人每天都会从Internet下载并运行应用程序。如果发现用户信任这些应用程序和web的下层链接，就可以继续使用自签名的证书（有关更加详细的内容请参看http://java.sun.com/javase/6/docs/technotes/guides/javaws/developersguide/development.html）。如果用户不信任这些应用程序，就要为用户提供安全的保证，将这些应用程序方在沙箱中。利用JNLP API（稍后将讨论）仍然允许应用程序有选择地访问资源和获得用户许可的内容。

10.2.3 JNLP API

JNLP API允许未签名的应用程序在沙箱中运行，同时通过一种安全的途径访问本地资源。例如，有一些加载和保存文件的服务。应用程序不能查看系统文件，也不能指定文件名。然而，可以弹出一个文件对话框，程序用户可以从中选择文件。在文件对话框弹出之前，应用程序不能浏览文件系统，也不能指定文件名。取而代之的是，系统将弹出文件对话框，由程序的用户从中选择文件。在文件对话框弹出之前，程序的用户会得到警告并且必须同意继续处理，如图10-8所示。而且，API并不给予程序访问File对象的权限。特别是，应用程序没有办法找到文件的位置。因此，程序员需要通过工具实现“打开文件”和“保存文件”的操作，但是对于不信任的应用程序，系统应尽可能地将信息隐藏起来。

图10-8 Java Web Start安全警告

API提供了下面的服务：

- 加载和保存文件
- 访问剪贴板
- 打印
- 下载文件
- 在默认的浏览器中显示一个文档
- 保存和恢复持久性配置信息
- 确信只运行一个应用程序的实例（JDK5.0 中新增）

要访问服务，需要使用ServiceManager，如下所示：

```
FileSaveService service = (FileSaveService) ServiceManager.lookup("javax.jnlp.FileSaveService");
```

如果服务不可用，调用将抛出UnavailableServiceException。

注释：如果想要编译使用了JNLP API的程序，那就必须在类路径中包含javaws.jar文件。这个文件在JDK的jre/lib子目录下。

现在，讨论一些最常用的JNLP服务。要保存文件，需要为文件对话框提供文件的初始路径名、文件名和扩展类型。例如：

```
service.saveFileDialog(".", new String[] { "txt" }, data, "calc.txt");
```

数据必须由InputStream 传递。这里有一些小窍门。在例10-2中，程序使用下面的策略：

1）建立ByteArrayOutputStream用于存放需要保存的字节。

2）建立PrintStream用于将数据传递给ByteArrayOutputStream。

3）将要保存的数据打印到PrintStream。

4）建立ByteArrayInputStream用于读取保存的字节。

5）将上面的流传递给saveFileDialog方法。

卷II的第1章将阐述更多有关流的知识。现在，可以忽略示例中关于流的细节。

要想从文件中读取数据，需要使用FileOpenService。它的openFileDialog对话框接收应用程序提供的初始路径名和文件扩展名，并返回一个FileContents对象。然后调用getInputStream和getOuputStream方法来读写文件数据。如果用户没有选择文件， openFileDialog方法将返回null。

```
FileOpenService service = (FileOpenService) ServiceManager.lookup("javax.jnlp.FileOpenService");
FileContents contents =  service.openFileDialog(".", new String[] { "txt" });
if (contents != null)
{
   InputStream in = contents.getInputStream();
   . . .
}
```

注意，应用程序不知道文件名或文件所放置的位置。相反地，如果想要打开一个特定的文件，需要使用ExtendedService：

```
ExtendedService service = (ExtendedService) ServiceManager.lookup("javax.jnlp.ExtendedService");
FileContents contents =  service.openFile(new File("c:\\autoexec.bat"));
if (contents != null)
{
   OutputStream out = contents.getOutputStream();
   . . .
}
```

程序的用户必须同意文件访问。如图10-9所示。

图10-9 文件访问警告

要想在默认浏览器中显示一个文档，就需要使用BasicService 接口。注意，有些系统（特别是UNIX和Linux系统）可能没有默认的浏览器。

```
BasicService service = (BasicService) ServiceManager.lookup("javax.jnlp.BasicService");
if (service.isWebBrowserSupported())
   service.showDocument(url);
```

```
else . . .
```

处于起步阶段的PersistenceService允许应用程序保存少量的配置信息，并且在应用程序下次运行时恢复。这种机制类似于HTTP的cookie，使用URL键进行持久性存储。URL并不需要指向一个真正的web资源，服务仅仅使用它们来构造一个方便的具有层次结构的名称空间。对于任何给定的URL键，应用程序可以存储任意的二进制数据（存储可能会限制数据块的大小）。

因为应用程序是彼此分离的，一个特定的应用程序只能使用从codebase开始的URL键值（codebase在JNLP文件中指定）。例如，如果一个应用程序是从http://myserver.com/apps下载的，那么它只能使用http://myserver.com/apps/subkey1/subkey2/...形式的键。访问其他键值的企图终将会失败。

应用程序可以调用BasicService类的getCodeBase方法查看它的codebase。可以调用PersistenceService中的create方法建立一个新的键：

```
URL url = new URL(codeBase, "mykey");
service.create(url, maxSize);
```

要想访问与某个特定键关联的信息，需要调用get方法。这个方法将返回一个FileContents对象，通过这个对象可以对键数据进行读写。例如：

```
FileContents contents = service.get(url);
InputStream in = contents.getInputStream();
OutputStream out = contents.getOutputStream(true); // true = overwrite
```

遗憾的是，无法轻而易举地知道这个键是已经存在还是需要创建。可以假定这个键已经存在，并调用get方法。如果抛出FileNotFoundException，就说明需要创建。

注释：从JDK 5.0开始，Java Web Start 应用程序和applet都可以使用常规的打印API进行打印。这时会弹出一个安全对话框来询问用户是否允许访问打印机。有关更多的打印API的信息,请参看卷II的第7章。

例10-2对计算器应用程序的功能做了一点改进。这个计算器有一个虚拟的纸带来记录所有的计算过程。用户可以保存或读取计算历史信息。为了演示持久性存储功能，应用程序允许设置框架的标题。如果再次运行程序，应用程序会从持久性存储中得到标题（如图10-10所示）。

图10-10 WebStartCalculator应用程序

例10-2 WebStartCalculator.java

```
1. import java.awt.EventQueue;
2. import java.awt.event.*;
3. import java.io.*;
4. import java.net.*;
5. import javax.swing.*;
6. import javax.jnlp.*;
7.
8. /**
9.  * A calculator with a calculation history that can be deployed as a Java Web Start application.
10. * @version 1.02 2007-06-12
11. * @author Cay Horstmann
12. */
13. public class WebStartCalculator
14. {
15.    public static void main(String[] args)
16.    {
17.       EventQueue.invokeLater(new Runnable()
18.          {
19.             public void run()
20.             {
21.                CalculatorFrame frame = new CalculatorFrame();
22.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23.                frame.setVisible(true);
24.             }
25.          });
26.    }
27. }
28.
29. /**
30.  * A frame with a calculator panel and a menu to load and save the calculator history.
31.  */
32. class CalculatorFrame extends JFrame
33. {
34.    public CalculatorFrame()
35.    {
36.       setTitle();
37.       panel = new CalculatorPanel();
38.       add(panel);
39.
40.       JMenu fileMenu = new JMenu("File");
41.
42.       JMenuItem openItem = fileMenu.add("Open");
43.       openItem.addActionListener(new ActionListener()
44.          {
45.             public void actionPerformed(ActionEvent event)
46.             {
47.                open();
48.             }
49.          });
50.
51.       JMenuItem saveItem = fileMenu.add("Save");
52.       saveItem.addActionListener(new ActionListener()
53.          {
54.             public void actionPerformed(ActionEvent event)
55.             {
```

```
56.                save();
57.             }
58.          });
59.       JMenuBar menuBar = new JMenuBar();
60.       menuBar.add(fileMenu);
61.       setJMenuBar(menuBar);
62.
63.       pack();
64.    }
65.
66.    /**
67.     * Gets the title from the persistent store or asks the user for the title if there is no
68.     * prior entry.
69.     */
70.    public void setTitle()
71.    {
72.       try
73.       {
74.          String title = null;
75.
76.          BasicService basic = (BasicService) ServiceManager.lookup("javax.jnlp.BasicService");
77.          URL codeBase = basic.getCodeBase();
78.
79.          PersistenceService service = (PersistenceService) ServiceManager
80.                .lookup("javax.jnlp.PersistenceService");
81.          URL key = new URL(codeBase, "title");
82.
83.          try
84.          {
85.             FileContents contents = service.get(key);
86.             InputStream in = contents.getInputStream();
87.             BufferedReader reader = new BufferedReader(new InputStreamReader(in));
88.             title = reader.readLine();
89.          }
90.          catch (FileNotFoundException e)
91.          {
92.             title = JOptionPane.showInputDialog("Please supply a frame title:");
93.             if (title == null) return;
94.
95.             service.create(key, 100);
96.             FileContents contents = service.get(key);
97.             OutputStream out = contents.getOutputStream(true);
98.             PrintStream printOut = new PrintStream(out);
99.             printOut.print(title);
100.         }
101.         setTitle(title);
102.      }
103.      catch (UnavailableServiceException e)
104.      {
105.         JOptionPane.showMessageDialog(this, e);
106.      }
107.      catch (MalformedURLException e)
108.      {
109.         JOptionPane.showMessageDialog(this, e);
110.      }
111.      catch (IOException e)
112.      {
```

```
113.          JOptionPane.showMessageDialog(this, e);
114.       }
115.    }
116.
117.    /**
118.     * Opens a history file and updates the display.
119.     */
120.    public void open()
121.    {
122.       try
123.       {
124.          FileOpenService service = (FileOpenService) ServiceManager
125.                .lookup("javax.jnlp.FileOpenService");
126.          FileContents contents = service.openFileDialog(".", new String[] { "txt" });
127.
128.          JOptionPane.showMessageDialog(this, contents.getName());
129.          if (contents != null)
130.          {
131.             InputStream in = contents.getInputStream();
132.             BufferedReader reader = new BufferedReader(new InputStreamReader(in));
133.             String line;
134.             while ((line = reader.readLine()) != null)
135.             {
136.                panel.append(line);
137.                panel.append("\n");
138.             }
139.          }
140.       }
141.       catch (UnavailableServiceException e)
142.       {
143.          JOptionPane.showMessageDialog(this, e);
144.       }
145.       catch (IOException e)
146.       {
147.          JOptionPane.showMessageDialog(this, e);
148.       }
149.    }
150.
151.    /**
152.     * Saves the calculator history to a file.
153.     */
154.    public void save()
155.    {
156.       try
157.       {
158.          ByteArrayOutputStream out = new ByteArrayOutputStream();
159.          PrintStream printOut = new PrintStream(out);
160.          printOut.print(panel.getText());
161.          InputStream data = new ByteArrayInputStream(out.toByteArray());
162.          FileSaveService service = (FileSaveService) ServiceManager
163.                .lookup("javax.jnlp.FileSaveService");
164.          service.saveFileDialog(".", new String[] { "txt" }, data, "calc.txt");
165.       }
166.       catch (UnavailableServiceException e)
167.       {
168.          JOptionPane.showMessageDialog(this, e);
169.       }
```

```
170.       catch (IOException e)
171.       {
172.          JOptionPane.showMessageDialog(this, e);
173.       }
174.    }
175.
176.    private CalculatorPanel panel;
177. }
```

API javax.jnlp.ServiceManager

- `static String[] getServiceNames()`
  返回所有可用的服务名称。
- `static Object lookup(String name)`
  返回给定名称的服务。

API javax.jnlp.BasicService

- `URL getCodeBase()`
  返回应用程序的codebase。
- `boolean isWebBrowserSupported()`
  如果Web Start环境可以启动浏览器，返回true。
- `boolean showDocument(URL url)`
  尝试在浏览器中显示一个给定的URL。如果请求成功，返回true。

API javax.jnlp.FileContents

- `InputStream getInputStream()`
  返回一个读取文件内容的输入流。
- `OutputStream getOutputStream(boolean overwrite)`
  返回一个对文件进行写操作的输出流。如果overwrite为true，文件中已有的内容将被覆盖。
- `String getName()`
  返回文件名（但不是完整的目录路径）。
- `boolean canRead()`
- `boolean canWrite()`
  如果后台文件可以读写，返回true。

API javax.jnlp.FileOpenService

- `FileContents openFileDialog(String pathHint, String[] extensions)`
- `FileContents[] openMultiFileDialog(String pathHint, String[] extensions)`
  显示警告信息并打开文件选择器。返回文件的内容描述符或者用户选择的文件。如果用户没有选择文件，返回null。

API javax.jnlp.FileSaveService

- FileContents saveFileDialog(String pathHint, String[] extensions, InputStream data, String nameHint)
- FileContents saveAsFileDialog(String pathHint, String[] extensions, FileContents data)

显示警告信息并打开文件选择器。写入数据，并返回文件的内容描述符或者用户选择的文件。如果用户没有选择文件，返回null。

API javax.jnlp.PersistenceService

- long create(URL key, long maxsize)
  对于给定的键创建一个持久性存储条目。返回由持久性存储授予这个条目的最大存储空间。
- void delete(URL key)
  删除给定键的条目。
- String[] getNames(URL url)
  返回由给定的URL开始的全部键的相对键名。
- FileContents get(URL key)
  返回用来修改与给定键相关的数据的内容描述符。如果没有与键相关的条目，抛出FileNotFoundException。

10.3 Applet

Applet是一种包含在HTML网页中的Java应用程序。HTML网页必须告诉浏览器要加载哪个applet以及每个applet放置在页面的哪个位置。正如所预计的那样，使用applet的标记需要告诉浏览器从哪里获得类文件，以及在网页上如何定位applet（大小、位置），然后，浏览器从Internet上（或者从用户机器的目录上）获得类文件，并自动地运行applet。

在applet开发早期，必须使用Sun的HotJava 浏览器来查看包含applet的网页。自然地，很少有用户愿意专门使用一个浏览器来享受一个新的web特性。当Netscape在其浏览器中包含了Java虚拟机之后，applet才真正地流行起来。微软的IE浏览器也紧随其后。可惜的是产生了两个问题。Netscape没有跟上Java的发展，而微软在对旧版Java是否提供支持的问题上举棋不定。

为了解决这个问题，Sun发布了一个名为Java Plug-in的工具。通过使用各种扩展机制，可以将插件平滑地嵌入到各种浏览器中，使这些浏览器能够利用Sun提供的外部Java运行时环境执行Java applet。通过更新插件可以保持使用最新、最棒的Java特性。

注释：只有安装当前版本的Java Plug-in，并且保证浏览器也正确连接了Java Plug-in，才能在浏览器中运行本章介绍的applet。可以登录http://java.sun.com/getjava 寻找下载信息和配置信息。

10.3.1 一个简单的applet

根据传统习惯，首先编写一个名为NotHelloWorld 的applet程序。一个applet就是一个扩展于java.applet.Applet类的Java类。在本书中，将使用Swing实现applet.注意，所有的applet都扩展于JApplet类，它是Swing applets的超类。如图10-11所示，JApplet是Applet类的直接子类。

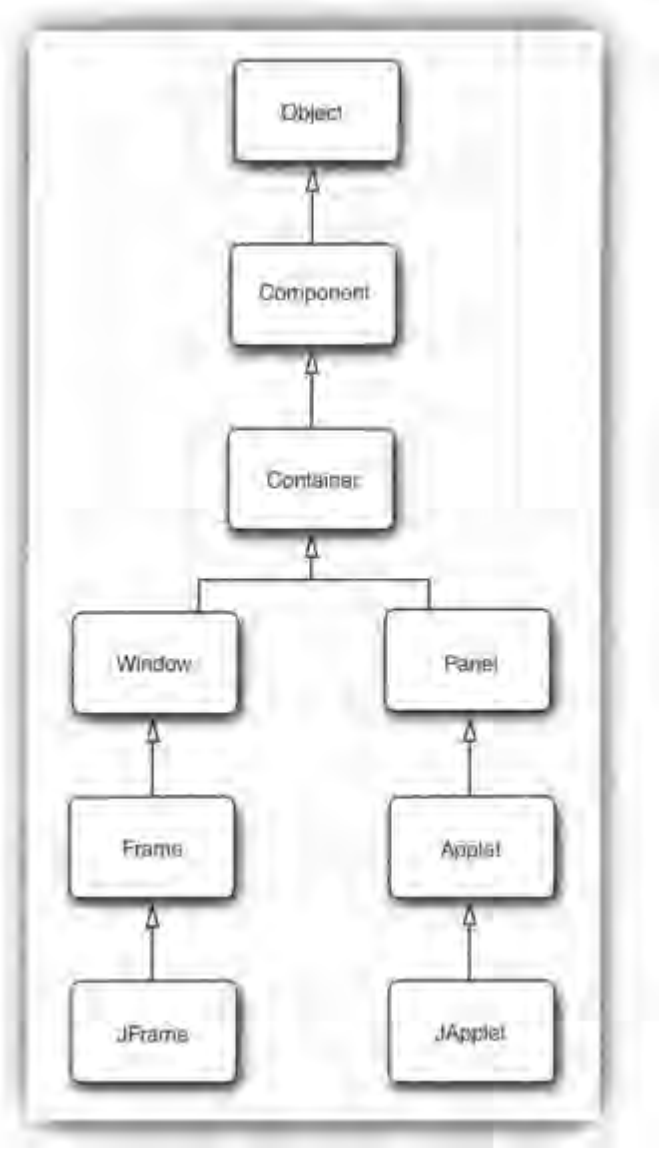

图10-11 Applet继承层次结构

注释：如果applet包含Swing组件，就必须扩展于JApplet类。在Applet类中的Swing组件不能正确地显示出来。

例10-3是“ Not Hello World ”的applet版本的代码。

注意，这个示例与第7章对应的程序非常相似。但是，由于applet在网页中运行，所以没有必要定义退出applet的方法。

例10-3 NotHelloWorldApplet.java

```
1. /*
2.  * The following HTML tags are required to display this applet in a browser: <applet
3.  * code="NotHelloWorldApplet.class" width="300" height="100"> </applet>
4.  */
5.
6. import java.awt.*;
7. import javax.swing.*;
8.
9. /**
10.  * @version 1.22 2007-06-12
11.  * @author Cay Horstmann
12.  */
13. public class NotHelloWorldApplet extends JApplet
14. {
15.    public void init()
16.    {
17.       EventQueue.invokeLater(new Runnable()
18.          {
19.             public void run()
20.             {
21.                JLabel label = new JLabel("Not a Hello, World applet", SwingConstants.CENTER);
22.                add(label);
23.             }
24.          });
25.    }
26. }
```

想执行applet，需要执行下面两个步骤：

1）将Java源文件编译成类文件。

2）创建一个HTML文件，告诉浏览器首先加载哪个类文件以及如何设定applet的大小。

习惯上（但不是必须的），将HTML文件命名为要加载的applet类文件名。因此，根据这个习惯，这个文件名为NotHelloWorldApplet.html。下面是文件的内容：

```
<applet code="NotHelloWorldApplet.class" width="300" height="300">
</applet>
```

在浏览器中查看applet之前，最好使用JDK自带的applet查看器（applet viewer）对applet测试一下。在本示例中，要想使用applet浏览器，需要键入下列命令：

```
appletviewer NotHelloWorldApplet.html
```

Applet查看器的命令行参数是HTML文件名，而不是class文件。图10-12展示了用applet查看器显示这个applet的效果。

图10-12 在applet查看器中查看applet

提示：这里有个用来避免创建其他HTML文件的技巧，即将applet标记直接作为注释添加在源文件中：

```
/*
```

```
  <applet code="MyApplet.class" width="300" height="300">
  </applet>
*/
public class MyApplet extends JApplet
. . .
```

然后将源文件作为命令行参数运行applet查看器：

```
appletviewer NotHelloWorldApplet.java
```

我们并不推荐使用这样方法，但是，如果想在测试阶段减少文件数目，这样方法还是很方便的。

提示：也可以在集成环境中运行applet。在Eclipse中，从菜单中选择Run→Run as→Java Applet。

Applet查看器很适合用于第一阶段的测试，但有时需要在浏览器中运行applet，以用户使用的方式查看。特别是，applet查看器程序只显示applet，而没有显示周围的HTML文本。如果HTML文件包含多个applet标签，applet查看器就会弹出多个窗口。

为了正确地查看applet，应该直接地在浏览器中调用HTML文件，如图10-13所示。如果applet没有显示出来，就需要安装Java Plug-in。

图10-13 在applet查看器中查看applet

提示：如果修改了applet并重新进行了编译，就需要重新启动浏览器以便加载新的类文件。只刷新HTML页面是不会加载新代码的。在调试applet时显得非常麻烦。通过启动Java控制台并发出x命令清理类加载器缓存可以避免烦人的浏览器重启。然后可以重新加载HTML页面，使用新的applet代码。在Windows下，打开Windows控制面板中的Java Plug-in 控制。在Linux下，运行jcontrol并请求Java控制台显示。当加载applet时，控制台就会弹开。

10.3.2 将应用程序转换为applet

将一个图形的Java应用程序转换为能够嵌入在网页中运行的applet非常容易。从本质上说，所有用户界面编码都是相同的。下面是将应用程序转化为applet的基本步骤：

1）创建一个HTML页面，并用适当的标记加载applet代码。

2）创建一个JApplet类的子类。将这个子类标记为公有。否则，不能加载apple。

3）删去应用程序中的main方法。不要为应用程序构造框架窗口。应用程序将显示在浏览

器中。

4）将框架窗口构造器中的初始化代码移到applet的init方法中。不需要明确地构造applet对象，浏览器负责实例化并调用init方法。

5）删除对setSize的调用。在applet中，大小由HTML中的width和height参数确定定义。

6）删除对setDefaultCloseOperation的调用。不要关闭applet，退出浏览器时它将会终止运行。

7）如果应用程序调用setTitle，要删除这个调用。Applet没有标题栏。（当然，可以使用HTML的title标记为网页设置标题。）

8）不要调用setVisible(true)。Applet会被自动显示出来。


注释：稍后，将会介绍如何编写程序，使其既是一个applet，又是一个应用程序。


`java.applet.Applet 1.0`

- `void init()`
  首次加载applet时调用这个方法。覆盖这个方法，并且将所有的初始化代码放在这里。
- `void start()`
  覆盖这个方法，将用户每次访问包含applet的网页时需要执行的代码放入其中。典型的操作是重新激活线程。
- `void stop()`
  覆盖这个方法，将用户每次离开包含applet网页时需要执行的代码放入其中。典型的操作是撤销线程。
- `void destroy()`
  覆盖这个方法，将用户退出浏览器时需要执行的代码放入其中。
- `void resize(int width, int height)`
  请求调整applet的尺寸。如果这个方法能够在网页上使用就太好了。遗憾的是，由于与当前的布局机制冲突，因此，在当前的浏览器中不起作用。

10.3.3 Applet的HTML 标记和属性

下面是一个以最简单的形式使用applet标记的例子：

```
<applet code="NotHelloWorldApplet.class" width="300" height="100">
```

可以看出，code属性指出了类名，必须要包括.class扩展名；width和height属性确定容纳applet窗口的大小。两者都以像素为单位。还需要一个匹配的</applet>标记，它标志applet的HTML标记的结束。在<applet>和</applet>标记之间的的文字只有在浏览器不支持applet时才会显示出来。code、width和height属性是必需的。如果缺少任何一个，浏览器将不能加载applet。

所有这些信息应该嵌入在HTML页面中。至少，应该如下所示：

```
<html>
   <head>
      <title>NotHelloWorldApplet</title>
   </head>
   <body>
```

```
      <p>The next line of text is displayed under the auspices of Java:</p>
      <applet code="NotHelloWorldApplet.class" width="100" height="100">
         If your browser could show Java, you would see an applet here.
      </applet>
   </body>
</html>
```

在applet标记中可以使用下列属性：

- `width, height`

  这些属性是必要的，它以像素为单位，设定applet的宽度和高度。在applet查看器中，这是applet的初始大小。在applet查看器中可以调整创建的窗口大小。在浏览器中，不能调整applet的大小，因此，需要预测一下applet所需要的空间，让用户能够看到一个显示效果良好的结果。

- `align`

  这个属性指定applet的对齐方式。属性值与HTMLimg标记的align属性相同。

- `vspace, hspace`

  这些可选属性指定applet上下（vspace）及两侧（hspace）的像素值。

- `code`

  这个属性指定applet的类文件名称。这个名称不是相对于codebase （稍后介绍）的， 就是在没有指定codebase时相对于当前页面的。

  路径名必须与applet类的包相匹配。例如，如果applet类在包com.mycompany中，这个属性就应该是code=“com/mycompany/MyApplet.class”。也可以定义为code=“com.mycompany.MyApplet.class”。但是，在这里不能使用绝对路径名。如果类文件存放在其他地方，就应该使用codebase属性。

  code属性只用于指定包含applet类的类名。当然，applet还可以包含其他类文件。当浏览器的类加载器加载包含applet的类时，它会意识到需要更多的类文件，并自动地加载这些类文件。

  code和object 属性（稍后介绍）都是必需的。

- `codebase`

  这个可选属性指出用于定位类文件的URL。可以使用绝对URL，甚至是不同的服务器。最常见的情况是，使用指向子目录的相对URL。例如，如果文件布局如下：

```
aDirectory/
   ├─ MyPage.html
   └─ myApplets/
         └─ MyApplet.class
```

  可以在MyPage.html中使用下列标记：

```
<applet code="MyApplet.class" codebase="myApplets" width="100" height="150">
```

- `archive`

  这个可选项列出JRA文件，或包含类的文件和applet需要的其他资源。这些文件将在加载applet前就从服务器端获得而来。这种技术明显地加快了加载过程的速度。这是因为只需

要一个HTTP的请求来加载包含多个小文件的JAR文件。JAR文件之间用逗号分隔。例如：

```
<applet code="MyApplet.class"
   archive="MyClasses.jar,corejava/CoreJavaClasses.jar"
   width="100" height="150">
```

- object

这个标签用来指定包含序列化（serialized）的applet对象的文件名（当把所有的实例域写到一个文件中时，对象就被序列化了。在卷II的第1章中将讨论序列化问题）。在显示applet时，对象将从文件中反序列化，从而还原到原先的状态。当使用这个属性时，不会调用init方法，而是调用applet的start方法。在序列化applet对象之前，应该调用stop方法。这个特性对于实现持久性浏览器是非常有用的。持久性浏览器可以自动重新加载applet并且将其恢复到上一次浏览器关闭时的状态。这是一个特殊的特性，网页设计人员通常不会遇到这个特性。

在每个applet标记中，必须有code和object属性，例如：

```
<applet object="MyApplet.ser" width="100" height="150">
```

- name

脚本编写人员希望为applet指定 name属性，这样在编写脚本时，可以使用这个属性来代表相应的applet。Netscape 和 Internet Explorer都允许利用JavaScript调用页面中的applet方法。本书不是专门讲述JavaScript的书籍，所以只利用下列代码简述一下如何用JavaScript调用Java代码。

注释：JavaScript是一种可以在网页内部使用的脚本语言，由Netscape开发，最初被称为LiveScript。除了语法与Java有些类似外，它基本上与Java没有什么关系。鉴于销售的目的，将它称为JavaScript。它的子集（称为ECMAScript）被标准化为ECMA-262。但是，Netscape和Microsoft在各自的浏览器中对这个标准作出了互不兼容的扩展，并没有人对此感到意外。如果想了解有关JavaScript的更多信息，我们一本书籍：《The Definitive Guide》，由O'Reilly & Associates出版，作者是David Flanagan。

要从JavaScript中访问applet，首先需要为其指定一个名字：

```
<applet code="MyApplet.class" width="100" height="150" name="mine">
</applet>
```

使用document.applets.appletname引用这个对象。例如：

```
var myApplet = document.applets.mine;
```

通过使用Netscape 和Internet Explorer提供的集成Java和JavaScript的能力，可以调用applet方法：

```
myApplet.init();
```

要使同一个页面上的两个applet之间可以直接地通信，name属性也是一个要素，必须为每个当前的applet实例指定一个名字。然后将这个名字字符串传递给AppletContext 类的getApplet方法。在本章稍后会讨论被称为applet间通信（inter-applet communication）的机制。

注释：在http://www.javaworld.com/javatips/jw-javatip80.html上，Francis Lu通过JavaScript与Java的通信解决了一个遗留长时间的问题：调整applet的大小而不再受width和height属性的限制。这是一个Java与JavaScript集成的很好示例。

- alt

Java有可能在浏览器中被系统管理员禁用。此时可以利用alt属性显示信息。

```
<applet code="MyApplet.class" width="100" height="150"
   alt="If you activated Java, you would see my applet here">
```

如果浏览器不能处理applet，那么就会忽略不识别的applet和param标记。位于<applet>与</applet>标记之间的所有文本会由浏览器显示出来。相反，支持Java的浏览器不会显示<applet>与</applet>标记之间的文本。对于仍旧使用旧版本浏览器的用户，可以在这些标记中显示提示信息。例如：

```
<applet code="MyApplet.class" width="100" height="150">
   If your browser could show Java, you would see my applet here.
</applet>
```

10.3.4 Object标记

Object标记是HTML 4.0标准的一部分，W3 协会建议用它取代applet标记。Object标记有35个不同的属性，大部分属性（如onkeydown）只适用于编写动态HTML的用户。各种定位属性（如align和height）的用法与applet标记中所对应的属性一样。对于Java applet而言，object标记中的关键属性是classid。这个属性指定对象所在的位置。当然，object标记可以加载不同类型的对象，如：Java applets或ActiveX组件（如JavaPlug-in本身）。在codetype属性中，可以指定对象的类别。如Java applet的代码类型是application/java。下面是一个能够加载Java applet的object标记的示例：

```
<object
   codetype="application/java"
   classid="java:MyApplet.class"
   width="100" height="150">
```

注意，classid属性后面可以跟codebase属性，并与在applet标记中的用法一样。

10.3.5 使用参数向applet传递信息

与应用程序可以使用命令行信息一样，applet也可以使用嵌入在HTML中的参数。这是由使用被称为param的HTML标记连同自定义属性完成的。例如，假定想让网页决定文字在applet中的样式。可以使用下面的HTML标记：

```
<applet code="FontParamApplet.class" width="200" height="200">
   <param name="font" value="Helvetica"/>
</applet>
```

可以使用Applet类中的getParameter方法获得参数值。如下例所示：

```
public class FontParamApplet extends JApplet
{
```

```
   public void init()
   {
      String fontName = getParameter("font");
      . . .
   }
   . . .
}
```

注释：只能在applet的init方法中调用getParameter方法，而不是在构造器中调用。当applet构造器被执行时，参数还没有准备好。由于许多重要的applet布局都是由参数决定的，所以建议用户不要为applet提供构造器，而是将所有的初始化代码放到init方法中。

参数的返回值通常是字符串类型。如果需要调用数字类型的值，那就需要将字符串类型转换为数字类型。可以调用适当的方法采用标准方式进行转换，如使用Integer类中parseInt方法。

例如，如果想为字体增加一个size参数，HTML代码可以是：

```
<applet code="FontParamApplet.class" width="200" height="200">
   <param name="font" value="Helvetica"/>
   <param name="size" value="24"/>
</applet>
```

下面的源代码显示了读取整型参数的方式：

```
public class FontParamApplet extends JApplet
{
   public void init()
   {
      String fontName = getParameter("font");
      int fontSize = Integer.parseInt(getParameter("size"));
      . . .
    }
}
```

注释：在param标记中定义的参数字符串应与getParameter方法中使用的字符串完全匹配。尤其是，两者都区分大小写。

除了保证参数与代码中的参数完全匹配外，还应该检查是否遗漏了size参数。通过测试字符串是否为null就可以达到这个目的。例如：

```
int fontsize;
String sizeString = getParameter("size");
if (sizeString == null) fontSize = 12;
else fontSize = Integer.parseInt(sizeString);
```

这里有一个很有用的applet，它使用了大量的参数。这个applet用于绘制如图10-14所示的柱状图。

```
<applet code="Chart.class" width="400" height="300">
   <param name="title" value="Diameters of the Planets"/>
   <param name="values" value="9"/>
   <param name="name.1" value="Mercury"/>
   <param name="name.2" value="Venus"/>
   <param name="name.3" value="Earth"/>
   <param name="name.4" value="Mars"/>
```

```
    <param name="name.5" value="Jupiter"/>
    <param name="name.6" value="Saturn"/>
    <param name="name.7" value="Uranus"/>
    <param name="name.8" value="Neptune"/>
    <param name="name.9" value="Pluto"/>
    <param name="value.1" value="3100"/>
    <param name="value.2" value="7500"/>
    <param name="value.3" value="8000"/>
    <param name="value.4" value="4200"/>
    <param name="value.5" value="88000"/>
    <param name="value.6" value="71000"/>
    <param name="value.7" value="32000"/>
    <param name="value.8" value="30600"/>
    <param name="value.9" value="1430"/>
</applet>
```

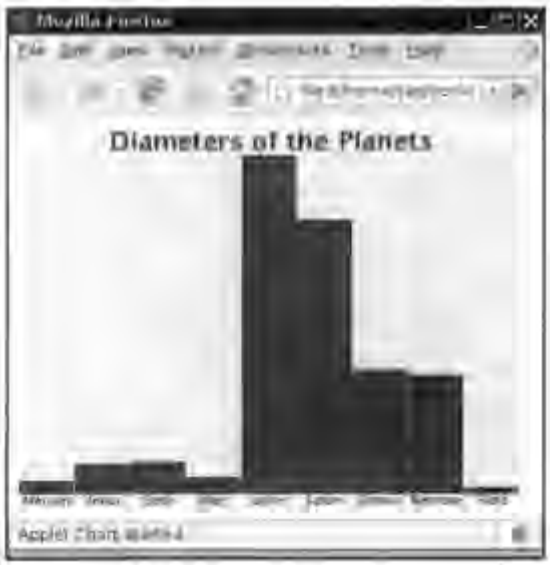

图10-14 图表applet

可以在applet中使用字符串数组和数字数组，然而使用参数机制将会有两个优势。可以在一个网页上放置同一个applet的多个拷贝以显示不同的柱状图，即只需要在页面中放置两个不同参数集的applet标记。而且还可以改变图表显示的数据。当然，像行星直径这样的数据在相当长的时间内都不会发生改变，但是，假如网页中包含的是每周销售量呢？由于使用纯文本，所以更新网页非常容易，而每周编辑和重新编译Java文件是很烦人的一件事情。

事实上，已经有很多商业JavaBeans组件（beans），它们可以显示比applet更加精美的图形。如果购买了商业JavaBean，就可以将其用到网页中，并且只需要设定参数，而不必知道applet绘制这些图形的具体方式。

例10-4是图表applet的源代码。注意，init方法读取参数， paintComponent方法绘制图表。

例10-4 Chart.java

```
1. import java.awt.*;
2. import java.awt.font.*;
3. import java.awt.geom.*;
```

```
4. import javax.swing.*;
5.
6. /**
7.  * @version 1.33 2007-06-12
8.  * @author Cay Horstmann
9.  */
10. public class Chart extends JApplet
11. {
12.    public void init()
13.    {
14.       EventQueue.invokeLater(new Runnable()
15.          {
16.             public void run()
17.             {
18.                String v = getParameter("values");
19.                if (v == null) return;
20.                int n = Integer.parseInt(v);
21.                double[] values = new double[n];
22.                String[] names = new String[n];
23.                for (int i = 0; i < n; i++)
24.                {
25.                   values[i] = Double.parseDouble(getParameter("value." + (i + 1)));
26.                   names[i] = getParameter("name." + (i + 1));
27.                }
28.
29.                add(new ChartComponent(values, names, getParameter("title")));
30.             }
31.          });
32.    }
33. }
34.
35. /**
36.  * A component that draws a bar chart.
37.  */
38. class ChartComponent extends JComponent
39. {
40.    /**
41.     * Constructs a ChartComponent.
42.     * @param v the array of values for the chart
43.     * @param n the array of names for the values
44.     * @param t the title of the chart
45.     */
46.    public ChartComponent(double[] v, String[] n, String t)
47.    {
48.       values = v;
49.       names = n;
50.       title = t;
51.    }
52.
53.    public void paintComponent(Graphics g)
54.    {
55.       Graphics2D g2 = (Graphics2D) g;
56.
57.       // compute the minimum and maximum values
58.       if (values == null) return;
59.       double minValue = 0;
60.       double maxValue = 0;
61.       for (double v : values)
```

```
62.       {
63.          if (minValue > v) minValue = v;
64.          if (maxValue < v) maxValue = v;
65.       }
66.       if (maxValue == minValue) return;
67.
68.       int panelWidth = getWidth();
69.       int panelHeight = getHeight();
70.
71.       Font titleFont = new Font("SansSerif", Font.BOLD, 20);
72.       Font labelFont = new Font("SansSerif", Font.PLAIN, 10);
73.
74.       // compute the extent of the title
75.       FontRenderContext context = g2.getFontRenderContext();
76.       Rectangle2D titleBounds = titleFont.getStringBounds(title, context);
77.       double titleWidth = titleBounds.getWidth();
78.       double top = titleBounds.getHeight();
79.
80.       // draw the title
81.       double y = -titleBounds.getY(); // ascent
82.       double x = (panelWidth - titleWidth) / 2;
83.       g2.setFont(titleFont);
84.       g2.drawString(title, (float) x, (float) y);
85.
86.       // compute the extent of the bar labels
87.       LineMetrics labelMetrics = labelFont.getLineMetrics("", context);
88.       double bottom = labelMetrics.getHeight();
89.
90.       y = panelHeight - labelMetrics.getDescent();
91.       g2.setFont(labelFont);
92.
93.       // get the scale factor and width for the bars
94.       double scale = (panelHeight - top - bottom) / (maxValue - minValue);
95.       int barWidth = panelWidth / values.length;
96.
97.       // draw the bars
98.       for (int i = 0; i < values.length; i++)
99.       {
100.         // get the coordinates of the bar rectangle
101.         double x1 = i * barWidth + 1;
102.         double y1 = top;
103.         double height = values[i] * scale;
104.         if (values[i] >= 0) y1 += (maxValue - values[i]) * scale;
105.         else
106.         {
107.            y1 += maxValue * scale;
108.            height = -height;
109.         }
110.
111.         // fill the bar and draw the bar outline
112.         Rectangle2D rect = new Rectangle2D.Double(x1, y1, barWidth - 2, height);
113.         g2.setPaint(Color.RED);
114.         g2.fill(rect);
115.         g2.setPaint(Color.BLACK);
116.         g2.draw(rect);
117.
118.         // draw the centered label below the bar
119.         Rectangle2D labelBounds = labelFont.getStringBounds(names[i], context);
```

```
120.
121.          double labelWidth = labelBounds.getWidth();
122.          x = x1 + (barWidth - labelWidth) / 2;
123.          g2.drawString(names[i], (float) x, (float) y);
124.       }
125.    }
126.
127.    private double[] values;
128.    private String[] names;
129.    private String title;
130. }
```

API `java.applet.Applet 1.0`

- `public String getParameter(String name)`
  获得正在加载的applet网页中由param标记定义的参数值。字符串名区分大小写。
- `public String getAppletInfo()`
  很多applet作者覆盖这个方法。它用于返回一个包含当前applet作者、版本号和版权信息的字符串。通过在applet类中覆盖这个方法，就可以创建这些信息。
- `public String[][] getParameterInfo()`
  可以覆盖这个方法，用于返回一个applet支持的param标记选项的数组。这个数组每行有三项：名字、类型和有关参数的描述。例如：

```
"fps", "1-10", "frames per second"
"repeat", "boolean", "repeat image loop?"
"images", "url", "directory containing images"
```

10.3.6 访问图像和音频文件

Applet可以处理图像和音频。在编写这本书时，图像必须是GIF、PNG或 JPEG 格式。音频文件必须是AU、AIFF、WAV或MIDI格式。动画支持GIF，并且能显示动画效果。

需要利用相对的URL指定图像和音频文件的位置。通常，由调用getDocumentBase或getCodeBase方法获得基URL。前者可以获得包含applet的HTML网页的URL；后者可以获得applet的codebase的URL。

注释：在早期的Java SE版本中，这些方法相当混乱，请参看Java bug parade中的错误报告#4456393 (http://bugs.sun.com/bugdatabase/index.jsp)。这些最终在JDK 5.0中得到了澄清。

可以通过getImage和getAudioClip方法获得基URL和文件的存储位置。例如：

```
Image cat = getImage(getCodeBase(), "images/cat.gif");
AudioClip meow = getAudioClip(getCodeBase(), "audio/meow.au");
```

第7章已经讲过如何显示图像。要想播放音频剪辑，只需要调用play方法即可。还可以调用Applet类中的play方法而不用加载音频剪辑。

```
play(getCodeBase(), "audio/meow.au");
```

API java.applet.Applet 1.0

- URL getDocumentBase()
  获得包含applet的网页的URL。
- URL getCodeBase()
  获得codebase目录的URL，applet是由这个目录加载的。返回的URL既可以是由codebase属性指定的引用目录的绝对URL，在没有指定codebase的情况下，也可以是HTML文件的目录。
- void play(URL url)
- void play(URL url, String name)
  前一个方法播放由URL指定的音频文件。第二个方法利用字符串提供相对于第一个参数URL的路径。如果没有找到音频剪辑，则不做任何操作。
- AudioClip getAudioClip(URL url)
- AudioClip getAudioClip(URL url, String name)
  前一个方法获得URL指定的音频剪辑。第二个方法使用字符串来提供相对于第一个参数URL的路径。如果没有找到音频剪辑，则返回null。
- Image getImage(URL url)
- Image getImage(URL url, String name)
  返回一个由URL指定的图像对象，这个对象封装一个图像。如果图像不存在，则立即返回null。否则，将启动一个独立的线程来装载图像。

10.3.7 Applet上下文

Applet在浏览器或者applet查看器中运行。Applet可以请求浏览器为它提供服务，例如，获得一段音频剪辑、在状态栏中显示简单的消息或者显示另一个网页。环境浏览器可以响应上述请求，也可以采取忽视的态度。例如，如果一个运行在applet查看器中的applet请求applet查看器显示一个网页，就不会给予响应。

如果想在浏览器之间进行通信，那么需要applet调用getAppletContext方法。这个方法将返回一个实现了AppletContext接口的对象。可以将AppletContext接口的具体实现认为是打开了一条applet与环境浏览器之间的通信道路。除了getAudioClip和getImage方法外，AppletContext接口还包含了几个很有用的方法，将在稍后介绍。

1. Applet间的通信

一个网页可以包含多个applet。如果网页中包含多个来自于同一个codebase的applet，则它们之间就可以互相通信。当然，这是一种高级技巧，也许很少用到。

如果为HTML文件中的每个applet都指定一个name属性，就可以使用AppletContext 接口中的getApplet方法来得到对applet的引用。例如，若HTML文件包含下列标记：

```
<applet code="Chart.class" width="100" height="100" name="Chart1">
```

则调用

```
Applet chart1 = getAppletContext().getApplet("Chart1");
```

就会得到对applet的引用。如何使用这个引用呢？如果Chart类包含一个利用获得的新数据重新绘制图表的方法，那么就可以在进行相应的类型转换后调用这个方法。

```
((Chart) chart1).setData(3, "Earth", 9000);
```

还可以把所有的applet都列在网页上，而不管它们是否有name属性。getApplets 方法返回一个枚举对象（enumeration object）。有关枚举更加详细的信息将在第13章中讨论。下面的循环可以打印出当前页面中包含的全部applet的类名。

```
Enumeration<Applet> e = getAppletContext().getApplets();
while (e.hasMoreElements())
{
   Applet a = e.nextElement();
   System.out.println(a.getClass().getName());
}
```

Applet不能与其他网页上的applet进行通信。

2. 在浏览器中显示信息

通过使用AppletContext 类的方法可以访问环境浏览器的两个区域：状态行和网页显示区域。

可以使用showStatus消息在浏览器底部的状态行中显示一个字符串，例如：

```
showStatus("Loading data . . . please wait");
```

提示：经验证明，showStatus方法有一定的局限性。这是因为浏览器也经常使用状态行，并会用Applet running之类的话覆盖住applet在状态行上显示的有用信息，所以，应该在状态行中显示不太重要的信息，例如，“ Loading data . . . please wait, ”，而不要显示对用户来说非常重要的信息。

利用showDocument 方法，可以请求浏览器显示不同的网页。有几种方法可以达到这个目的。最简单的一种方法是调用只需要提供一个参数的showDocument 方法，其参数就是要显示的URL：

```
URL u = new URL("http://java.sun.com/index.html");
getAppletContext().showDocument(u);
```

这种调用方式的问题在于新打开的页面将会出现在当前网页的窗口中。因此，applet会被替换掉。要想返回applet，用户必须点击浏览器的Back按钮。

如果在调用showDocument方法时提供第二个参数，就可以告诉浏览器在另一个窗口中显示文档，请参看10-2。如果提供的特殊字符串是“ _blank ”，浏览器就会打开一个新窗口，而不会替换原来的窗口。更重要的是，如果利用HTML的框架特性，可以将浏览器窗口分成多个框架，每个框架都有一个名字，可以将applet放到一个框架中，并让它在另一个框架中显示文档。在下一节中，将给出这样一个示例。

注释：Sun的applet查看器不能显示网页。showDocument 方法被applet查看器忽视。

表10-2 showDocument 方法

目标参数	位置
“_self”或者none	在当前框架中显示文档
“_parent”	在父框架中显示文档
“_top”	在最顶层框架中显示文档
“_blank”	在新的、未命名的、顶层窗口中显示文档
其他字符串	在指定名称的框架中显示文档。如果没有相应名称的框架，就打开一个新窗口，并给这个窗口指定名称

3. 既是applet，又是应用程序

几年前，在一则名为“周六晚直播”的幽默电视广告中，一对夫妇正在讨论一种白色胶状物质。丈夫说:“这是一流的高级甜食。”妻子说:“这是地板蜡。”而播音员得意地宣布：“都对！”

在这一节中，将介绍如何将一个applet转换成一个Java程序，使它既是一个applet又是一个应用程序。也就是说，既可以在applet查看器和浏览器中加载这个程序，也可以通过命令行使用Java程序启动器启动这个程序。我们并不知道这种情况发生的几率，只是觉得很有趣，希望读者也是这么看的。

图10-15和图10-16显示了同一个程序。在applet查看器中运行的是applet。由命令行启动的是应用程序。

图10-15 Applet程序

图10-16 应用程序

GUI应用程序会构造一个框架对象，并调用setVisible(true)方法。由于不能在光秃秃的applet上调用setVisible，所以必须将appet放在框架中，并在AppletFrame类提供的构造器中将applet添加到内容窗格中。

```
public class AppletFrame extends JFrame
{
   public AppletFrame(Applet anApplet)
   {
      applet = anApplet;
      add(applet);
      . . .
   }
   . . .
}
```

在applet/应用程序的main方法中，构造并显示AppletFrame。

```
class MyAppletApplication extends MyApplet
{
   public static void main(String args[])
   {
      AppletFrame frame = new AppletFrame(new MyApplet());
      frame.setTitle("MyAppletApplication");
      frame.setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
   }
}
```

注意，直接扩展于已存在的applet。

在applet启动时，必须调用它的init方法和start方法。通过覆盖AppletFrame 类中的setVisible方法可以达到这个目的：

```
public void setVisible(boolean b)
{
   if (b)
   {
      applet.init();
      super.setVisible(true);
      applet.start();
   }
   else
   {
      applet.stop();
      super.setVisible(false);
      applet.destroy();
   }
}
```

但有一个问题。如果程序以应用程序的方式启动，并且调用getAppletContext方法，就会得到一个null指针，其原因是这个程序不能在浏览器中启动。如果有下列代码，就会引发运行时错误

```
getAppletContext().showStatus(message);
```

我们并不想编写一个功能齐全的浏览器，但是需要对上述调用提供最低限度的支持。这个调用可以不显示任何信息，但至少不能导致程序崩溃。这就需要实现两个接口： AppletStub 和AppletContext。AppletStub接口的主要用途是定位applet的上下文。每个applet都有一个applet桩（用Applet类的setStub方法设置）。下面提供了实现这两个接口的最小的基本功能。

```
public class AppletFrame extends JFrame
   implements AppletStub, AppletContext
{
   . . .
   // AppletStub methods
   public boolean isActive() { return true; }
   public URL getDocumentBase() { return null; }
   public URL getCodeBase() { return null; }
   public String getParameter(String name) { return ""; }
   public AppletContext getAppletContext() { return this; }
```

```
   public void appletResize(int width, int height) {}

   // AppletContext methods
   public AudioClip getAudioClip(URL url) { return null; }
   public Image getImage(URL url) { return Toolkit.getDefaultToolkit().getImage(url); }
   public Applet getApplet(String name) { return null; }
   public Enumeration<Applet> getApplets() { return null; }
   public void showDocument(URL url) {}
   public void showDocument(URL url, String target) {}
   public void showStatus(String status) {}
   public void setStream(String key, InputStream stream) {}
   public InputStream getStream(String key) { return null; }
   public Iterator<String> getStreamKeys() { return null; }
}
```

接下来，框架类的构造器调用applet的setStub方法，以便将applet设定为自己的桩。

```
public AppletFrame(Applet anApplet)
{
   applet = anApplet
   Container contentPane = getContentPane();
   contentPane.add(applet);
   applet.setStub(this);
}
```

使用前面提到的技巧，将applet标记以注释的方式添加到源文件中。这样，可以直接地在applet查看器中调用源程序，而不需要额外的HTML文件。

例10-5和图10-6列出了源代码。试着运行applet和应用程序：

```
appletviewer NotHelloAppletApplication.java
java NotHelloAppletApplication
```

例10-5 AppletFrame.java

```
1. import java.awt.*;
2. import java.applet.*;
3. import java.io.*;
4. import java.net.*;
5. import java.util.*;
6. import javax.swing.*;
7.
8. /**
9.  * @version 1.32 2007-06-12
10.  * @author Cay Horstmann
11.  */
12. public class AppletFrame extends JFrame implements AppletStub, AppletContext
13. {
14.    public AppletFrame(Applet anApplet)
15.    {
16.       applet = anApplet;
17.       add(applet);
18.       applet.setStub(this);
19.    }
20.
21.    public void setVisible(boolean b)
22.    {
23.       if (b)
```

```
24.       {
25.          applet.init();
26.          super.setVisible(true);
27.          applet.start();
28.       }
29.       else
30.       {
31.          applet.stop();
32.          super.setVisible(false);
33.          applet.destroy();
34.       }
35.    }
36.
37.    // AppletStub methods
38.    public boolean isActive()
39.    {
40.       return true;
41.    }
42.
43.    public URL getDocumentBase()
44.    {
45.       return null;
46.    }
47.
48.    public URL getCodeBase()
49.    {
50.       return null;
51.    }
52.
53.    public String getParameter(String name)
54.    {
55.       return "";
56.    }
57.
58.    public AppletContext getAppletContext()
59.    {
60.       return this;
61.    }
62.
63.    public void appletResize(int width, int height)
64.    {
65.    }
66.
67.    // AppletContext methods
68.    public AudioClip getAudioClip(URL url)
69.    {
70.       return null;
71.    }
72.
73.    public Image getImage(URL url)
74.    {
75.       return Toolkit.getDefaultToolkit().getImage(url);
76.    }
77.
78.    public Applet getApplet(String name)
79.    {
80.       return null;
81.    }
```

```
82.
83.    public Enumeration<Applet> getApplets()
84.    {
85.       return null;
86.    }
87.
88.    public void showDocument(URL url)
89.    {
90.    }
91.
92.    public void showDocument(URL url, String target)
93.    {
94.    }
95.
96.    public void showStatus(String status)
97.    {
98.    }
99.
100.   public void setStream(String key, InputStream stream)
101.   {
102.   }
103.
104.   public InputStream getStream(String key)
105.   {
106.      return null;
107.   }
108.
109.   public Iterator<String> getStreamKeys()
110.   {
111.      return null;
112.   }
113.
114.   private Applet applet;
115. }
```

例10-6 AppletApplication.java

```
1. /*
2.  * The applet viewer reads the tags below if you call it with appletviewer AppletApplication.java
3.  * No separate HTML file is required. <applet code="AppletApplication.class"
4.  * width="200" height="200">
5.  * </applet>
6.  */
7.
8. import java.awt.EventQueue;
9.
10. import javax.swing.*;
11.
12. /**
13.  * It's an applet. It's an application. It's BOTH!
14.  * @version 1.32 2007-04-28
15.  * @author Cay Horstmann
16.  */
17. public class AppletApplication extends NotHelloWorldApplet
18. {
19.    public static void main(String[] args)
20.    {
```

```
21.       EventQueue.invokeLater(new Runnable()
22.          {
23.             public void run()
24.             {
25.                AppletFrame frame = new AppletFrame(new NotHelloWorldApplet());
26.                frame.setTitle("NotHelloWorldApplet");
27.                frame.setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
28.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
29.                frame.setVisible(true);
30.             }
31.          });
32.    }
33.
34.    public static final int DEFAULT_WIDTH = 200;
35.    public static final int DEFAULT_HEIGHT = 200;
36. }
```

API `java.applet.Applet 1.2`

- `public AppletContext getAppletContext()`
  获得applet浏览器环境的一个句柄。在大多数浏览器中，可以使用这个信息控制运行applet的浏览器。
- `void showStatus(String msg)`
  显示在浏览器状态栏中指定的字符串。

API `java.applet.AppletContext 1.0`

- `Enumeration <Applet> getApplets()`
  返回在同一个上下文中（也就是在同一个网页中）所有applet的枚举值（参看第13章）。
- `Applet getApplet(String name)`
  返回当前上下文中给定名称的applet。如果不存在返回null。只在当前网页中进行查找。
- `void showDocument(URL url)`
- `void showDocument(URL url, String target)`
  在浏览器的框架中显示一个新网页。第一个形式在当前页中显示新页面。第二种形式用target参数指定目标框架。请参看表10-2。

10.4 应用程序存储的配置

应用程序的用户通常期待能够自行对应用程序进行配置，并能够将其保存起来。日后再次运行这个应用程序时将能够读取这些配置。下面首先介绍一种直接将配置信息存储在属性文件中的传统方式。然后，再介绍Java SE 1.4中的一种功能更加强大的机制。

10.4.1 属性映射

属性映射（property map）是一种存储键/值对的数据结构。属性映射经常被用来存放配置信息。它有三个特性：

- 键和值都是字符串
- 键/值对可以很容易地写入文件或从文件读出。
- 用二级表存放默认值。

实现属性映射的Java类被称为Properties。

属性映射对指定程序的配置选项非常有用。例如：

```
Properties settings = new Properties();
settings.put("width", "200");
settings.put("title", "Hello, World!");
```

可以使用store方法将这个属性映射列表保存到文件中。在这里，将属性映射保存在Myprog.properties文件中。第二个参数是包含到这个文件的注释。

```
FileOutputStream out = new FileOutputStream("program.properties");
settings.store(out, "Program Properties");
```

上述示例将会输出如下结果。

```
#Program Properties
#Mon Apr 30 07:22:52  2007
width=200
title=Hello, World!
```

要想从文件中加载这些属性，可以使用：

```
FileInputStream in = new FileInputStream("program.properties");
settings.load(in);
```

习惯上，将程序属性存储在用户主目录的某个子目录下。目录名通常选择由一个（在UNIX系统中）圆点开始。这个约定表示来自用户的一个隐藏的系统目录。在示例程序中将遵守这个约定。

要想查看用户的主目录，可以调用System.getProperties方法。很巧，还可以使用Properties对象描述系统信息。主目录包含键user.home。还有一个很有用的方法，它用于读取单键：

```
String userDir = System.getProperty("user.home");
```

一旦用户手工地编辑文件，那么为应用程序提供默认值就是一种很好的想法。Properties类有两种提供默认值的机制。第一种是在试图获得字符串值时指定默认值。当键值不存在时，就会自动地使用它。

```
String title = settings.getProperty("title", "Default title");
```

如果在属性映射中有title属性，则title将设置成字符串。否则，title将设置成Default title。

如果觉得在每次调用getProperty方法时指定默认值太麻烦，那么就可以将所有的默认值放在一个二级属性映射中，并在主属性映射的构造器中提供映射。且用它来构造查询表。

```
Properties defaultSettings = new Properties();
defaultSettings.put("width", "300");
defaultSettings.put("height", "200");
defaultSettings.put("title", "Default title");
. . .
Properties settings = new Properties(defaultSettings);
```

甚至，如果为defaultSettings的构造器提供另一个属性映射参数，可以为默认值指定默认值。

但是一般不会这样做。例10-7展示了利用这些属性存储和加载程序状态的方式。程序将记住框架的位置、大小和标题。还可以手工地编辑主目录中的文件.corejava/program.properties，用以将程序的外观变成所希望的那样。

注释：属性映射是一个没有层次结构的简单表。通过引入window.main.color、window.main.title等键名可以伪装层次结构。但是Properties类没有提供组织这种层次结构的方法。如果存储复杂的配置信息，就应该使用Preferences 类，请看下一节。

例10-7 PropertiesTest.java

```
1. import java.awt.EventQueue;
2. import java.awt.event.*;
3. import java.io.*;
4. import java.util.Properties;
5.
6. import javax.swing.*;
7.
8. /**
9.  * A program to test properties. The program remembers the frame position, size, and title.
10.  * @version 1.00 2007-04-29
11.  * @author Cay Horstmann
12.  */
13. public class PropertiesTest
14. {
15.    public static void main(String[] args)
16.    {
17.       EventQueue.invokeLater(new Runnable()
18.          {
19.             public void run()
20.             {
21.                PropertiesFrame frame = new PropertiesFrame();
22.                frame.setVisible(true);
23.             }
24.          });
25.    }
26. }
27.
28. /**
29.  * A frame that restores position and size from a properties file and updates the properties
30.  * upon exit.
31.  */
32. class PropertiesFrame extends JFrame
33. {
34.    public PropertiesFrame()
35.    {
36.       // get position, size, title from properties
37.
38.       String userDir = System.getProperty("user.home");
39.       File propertiesDir = new File(userDir, ".corejava");
40.       if (!propertiesDir.exists()) propertiesDir.mkdir();
41.       propertiesFile = new File(propertiesDir, "program.properties");
42.
43.       Properties defaultSettings = new Properties();
44.       defaultSettings.put("left", "0");
```

```
45.       defaultSettings.put("top", "0");
46.       defaultSettings.put("width", "" + DEFAULT_WIDTH);
47.       defaultSettings.put("height", "" + DEFAULT_HEIGHT);
48.       defaultSettings.put("title", "");
49.
50.       settings = new Properties(defaultSettings);
51.
52.       if (propertiesFile.exists()) try
53.       {
54.          FileInputStream in = new FileInputStream(propertiesFile);
55.          settings.load(in);
56.       }
57.       catch (IOException ex)
58.       {
59.          ex.printStackTrace();
60.       }
61.
62.       int left = Integer.parseInt(settings.getProperty("left"));
63.       int top = Integer.parseInt(settings.getProperty("top"));
64.       int width = Integer.parseInt(settings.getProperty("width"));
65.       int height = Integer.parseInt(settings.getProperty("height"));
66.       setBounds(left, top, width, height);
67.
68.       // if no title given, ask user
69.
70.       String title = settings.getProperty("title");
71.       if (title.equals("")) title = JOptionPane.showInputDialog("Please supply a frame title:");
72.       if (title == null) title = "";
73.       setTitle(title);
74.
75.       addWindowListener(new WindowAdapter()
76.          {
77.             public void windowClosing(WindowEvent event)
78.             {
79.                settings.put("left", "" + getX());
80.                settings.put("top", "" + getY());
81.                settings.put("width", "" + getWidth());
82.                settings.put("height", "" + getHeight());
83.                settings.put("title", getTitle());
84.                try
85.                {
86.                   FileOutputStream out = new FileOutputStream(propertiesFile);
87.                   settings.store(out, "Program Properties");
88.                }
89.                catch (IOException ex)
90.                {
91.                   ex.printStackTrace();
92.                }
93.                System.exit(0);
94.             }
95.          });
96.    }
97.
98.    private File propertiesFile;
99.    private Properties settings;
100.
```

```
101.    public static final int DEFAULT_WIDTH = 300;
102.    public static final int DEFAULT_HEIGHT = 200;
103. }
```

API java.util.Properties 1.0

- `Properties()`
  创建一个空的属性映射。
- `Properties(Properties defaults)`
  用一组默认值创建空的属性映射。
  参数：defaults　用于查找的默认值集合
- `String getProperty(String key)`
  获得一个属性映射。返回对应键的字符串。如果键没有在表中出现，返回在默认值表中与键对应的字符串。如果键也没有出现在默认值表中，则返回null。
  参数：key　要获得的相关字符串的键
- `String getProperty(String key, String defaultValue)`
  如果没有找到键，返回具有默认值的属性。如果键不没有在表中出现，则返回对应键的字符串，或默认字符串。
  参数：key　要获得的相关字符串的键
  defaultValue　如果键不存在，则返回这个字符串
- `void load(InputStream in) throws IOException`
  从输入流中加载一个属性映射。
  参数：in　输入流
- `void store(OutputStream out, String header) 1.2`
  将属性映射存放到一个输出流中。
  参数：out　输出流
  header　存储文件中的第一行标题

API java.lang.System 1.0

- `Properties getProperties()`
  获得全部系统属性。应用程序必须有访问全部系统属性的权限，否则将抛出安全异常。
- `String getProperty(String key)`
  返回具有给定键名的系统属性。应用程序必须有访问全部系统属性的权限，否则将抛出安全异常。下列特性无论怎样都可以得到：

```
java.version
java.vendor
java.vendor.url
java.class.version
os.name
os.version
os.arch
```

```
file.separator
path.separator
line.separator
java.specification.version
java.vm.specification.version
java.vm.specification.vendor
java.vm.specification.name
java.vm.version
java.vm.vendor
java.vm.name
```

注释：可以在Java运行时的目录下的security/java. policy文件中找到免费访问系统特性的名字。

10.4.2 Preferences API

正如读者所看到的，Properties类能够简化读取和保存配置信息的过程。但是，使用属性文件存在下列不足：

- 配置文件不能存放在用户的主目录中。这是因为某些操作系统（如Windows 9x） 没有主目录的概念。
- 没有标准的为配置文件命名的规则。当用户安装了多个Java应用程序时，会增加配置文件名冲突的可能性。

有些操作系统提供一个存储配置信息的中心知识库。其中广为人知的就是Microsoft Windows。Java SE 1.4中的Preferences类提供了一个与平台无关的中心知识库。在Windows中，Preferences类用来注册存储信息；在Linux中，这些信息被存放在本地文件系统中。当然，对于使用Preferences类的程序员而言，中心知识库的实现是透明的。

Preferences的中心知识库具有树状结构，每个节点的路径类似于/com/mycompany/myapp。因为有了包名，所以只要程序员用保留的域名作为路径的开始，就可以避免命名冲突。实际上，API的设计者建议配置节点路径与程序中的包名匹配。

中心知识库中的每个节点都有一个用来存放键/值的独立表。用户可以用这张表存放数字、字符串或字节数组。但不适于存放序列化的对象。API的设计者们认为序列化格式不适宜长期存放数据。当然，如果不适宜的话，可以将序列化对象存放在字节数组中。

为了增加灵活性，系统中有多棵并行的树。每个程序使用者拥有一棵树，同时，系统中还存在一棵树，称为系统树，用于存放全体用户的公共信息。Preferences类使用操作系统中“当前用户”的概念确保用户访问恰当的用户树。

为了访问树中节点，要从用户或系统的根开始：

```
Preferences root = Preferences.userRoot();
```

或者

```
Preferences root = Preferences.systemRoot();
```

然后访问这个节点。需要直接提供节点的路径名：

```
Preferences node = root.node("/com/mycompany/myapp");
```

如果节点的路径名和类的包名一样，则有一种简单、快捷的方法获取这个节点。只需要获得这个类的对象，然后调用：

```
Preferences node = Preferences.userNodeForPackage(obj.getClass());
```

或者

```
Preferences node = Preferences.systemNodeForPackage(obj.getClass());
```

通常情况下，obj就是this引用。

一旦获得了节点，就可以使用下列方法访问键/值表。

```
String get(String key, String defval)
int getInt(String key, int defval)
long getLong(String key, long defval)
float getFloat(String key, float defval)
double getDouble(String key, double defval)
boolean getBoolean(String key, boolean defval)
byte[] getByteArray(String key, byte[] defval)
```

注意，在读取信息时，必须指定默认值，以防止中心知识库中的值不可用。有几种情况需要默认值。由于用户始终没有指定配置信息，所以数据可能找不到。在某些对资源受限的平台上可能没有中心知识库，并且移动设备可能暂时与中心知识库断开连接。

与之对应，可以利用put方法将数据写入中心知识库。

```
put(String key, String value)
putInt(String key, int value)
```

使用下列方法可以枚举出节点中全部的键：

```
String[] keys
```

但是目前无法查看给定键值的类型。

类似于Windows注册信息表这样的中心知识库通常会遇到下面两个问题。

- 其中添满了陈旧的信息，成为了“垃圾场”。
- 配置数据与中心知识库相关，使得数据不易迁移至新平台上。

Preferences类解决了第二个问题。可以调用下列方法将子树（或者一个节点）的全部值显示输出。

```
void exportSubtree(OutputStream out)
void exportNode(OutputStream out)
```

数据以XML格式存储。可以调用下列方法将它们导入到另一个中心知识库中

```
void importPreferences(InputStream in)
```

下面是一个示例文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE preferences SYSTEM "http://java.sun.com/dtd/preferences.dtd">
<preferences EXTERNAL_XML_VERSION="1.0">
  <root type="user">
    <map/>
    <node name="com">
      <map/>
      <node name="horstmann">
```

```
      <map/>
      <node name="corejava">
        <map>
          <entry key="left" value="11"/>
          <entry key="top" value="9"/>
          <entry key="width" value="453"/>
          <entry key="height" value="365"/>
          <entry key="title" value="Hello, World!"/>
        </map>
      </node>
    </node>
  </node>
 </root>
</preferences>
```

如果程序使用配置信息，那么应该允许用户导入和导出，这样就可以将这些信息从一台机器上迁移到另一台上。例10-8程序演示了这个技术。程序只保存主窗口的位置、大小和标题。然后改变窗口的大小，退出并重启应用程序。窗口的大小和位置将与关闭窗口时一样。

例10-8 PreferencesTest.java

```
1. import java.awt.EventQueue;
2. import java.awt.event.*;
3. import java.io.*;
4. import java.util.prefs.*;
5. import javax.swing.*;
6.
7. /**
8.  * A program to test preference settings. The program remembers the frame position, size,
9.  * and title.
10.  * @version 1.02 2007-06-12
11.  * @author Cay Horstmann
12.  */
13. public class PreferencesTest
14. {
15.    public static void main(String[] args)
16.    {
17.       EventQueue.invokeLater(new Runnable()
18.          {
19.             public void run()
20.             {
21.                PreferencesFrame frame = new PreferencesFrame();
22.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23.                frame.setVisible(true);
24.             }
25.          });
26.    }
27. }
28.
29. /**
30.  * A frame that restores position and size from user preferences and updates the preferences
31.  * upon exit.
32.  */
33. class PreferencesFrame extends JFrame
34. {
35.    public PreferencesFrame()
```

```
36.    {
37.       // get position, size, title from preferences
38.
39.       Preferences root = Preferences.userRoot();
40.       final Preferences node = root.node("/com/horstmann/corejava");
41.       int left = node.getInt("left", 0);
42.       int top = node.getInt("top", 0);
43.       int width = node.getInt("width", DEFAULT_WIDTH);
44.       int height = node.getInt("height", DEFAULT_HEIGHT);
45.       setBounds(left, top, width, height);
46.
47.       // if no title given, ask user
48.
49.       String title = node.get("title", "");
50.       if (title.equals("")) title = JOptionPane.showInputDialog("Please supply a frame title:");
51.       if (title == null) title = "";
52.       setTitle(title);
53.
54.       // set up file chooser that shows XML files
55.
56.       final JFileChooser chooser = new JFileChooser();
57.       chooser.setCurrentDirectory(new File("."));
58.
59.       // accept all files ending with .xml
60.       chooser.setFileFilter(new javax.swing.filechooser.FileFilter()
61.          {
62.             public boolean accept(File f)
63.             {
64.                return f.getName().toLowerCase().endsWith(".xml") || f.isDirectory();
65.             }
66.
67.             public String getDescription()
68.             {
69.                return "XML files";
70.             }
71.          });
72.
73.       // set up menus
74.       JMenuBar menuBar = new JMenuBar();
75.       setJMenuBar(menuBar);
76.       JMenu menu = new JMenu("File");
77.       menuBar.add(menu);
78.
79.       JMenuItem exportItem = new JMenuItem("Export preferences");
80.       menu.add(exportItem);
81.       exportItem.addActionListener(new ActionListener()
82.          {
83.             public void actionPerformed(ActionEvent event)
84.             {
85.                if (chooser.showSaveDialog(PreferencesFrame.this) == JFileChooser.APPROVE_OPTION)
86.                {
87.                   try
88.                   {
89.                      OutputStream out = new FileOutputStream(chooser.getSelectedFile());
90.                      node.exportSubtree(out);
91.                      out.close();
92.                   }
```

```
93.                  catch (Exception e)
94.                  {
95.                     e.printStackTrace();
96.                  }
97.               }
98.            }
99.         });
100.
101.     JMenuItem importItem = new JMenuItem("Import preferences");
102.     menu.add(importItem);
103.     importItem.addActionListener(new ActionListener()
104.        {
105.           public void actionPerformed(ActionEvent event)
106.           {
107.              if (chooser.showOpenDialog(PreferencesFrame.this) == JFileChooser.APPROVE_OPTION)
108.              {
109.                 try
110.                 {
111.                    InputStream in = new FileInputStream(chooser.getSelectedFile());
112.                    Preferences.importPreferences(in);
113.                    in.close();
114.                 }
115.                 catch (Exception e)
116.                 {
117.                    e.printStackTrace();
118.                 }
119.              }
120.           }
121.        });
122.
123.     JMenuItem exitItem = new JMenuItem("Exit");
124.     menu.add(exitItem);
125.     exitItem.addActionListener(new ActionListener()
126.        {
127.           public void actionPerformed(ActionEvent event)
128.           {
129.              node.putInt("left", getX());
130.              node.putInt("top", getY());
131.              node.putInt("width", getWidth());
132.              node.putInt("height", getHeight());
133.              node.put("title", getTitle());
134.              System.exit(0);
135.           }
136.        });
137.  }
138.
139.  public static final int DEFAULT_WIDTH = 300;
140.  public static final int DEFAULT_HEIGHT = 200;
141. }
```

API java.util.prefs.Preferences 1.4

- `Preferences userRoot()`

  返回调用应用程序的用户配置信息的根节点。

- `Preferences systemRoot()`
  返回系统范围的配置信息的根节点。
- `Preferences node(String path)`
  返回从当前节点经过给定路径可以到达的节点。如果path是绝对路径（以 / 开始），则从包含配置信息的树根开始查找这个节点。如果给定的路径不存在这样的节点，则创建它。
- `Preferences userNodeForPackage(Class cl)`
- `Preferences systemNodeForPackage(Class cl)`
  返回当前用户树或者系统树中的节点。这个节点的绝对路径符合类cl的包名。
- `String[] keys()`
  返回属于这个节点的所有键。
- `String get(String key, String defval)`
- `int getInt(String key, int defval)`
- `long getLong(String key, long defval)`
- `float getFloat(String key, float defval)`
- `double getDouble(String key, double defval)`
- `boolean getBoolean(String key, boolean defval)`
- `byte[] getByteArray(String key, byte[] defval)`
  返回与给定键关联的值。如果没有与之关联的值，或者关联值的类型不符合要求，或者配置存储不可用，则返回默认值。
- `void put(String key, String value)`
- `void putInt(String key, int value)`
- `void putLong(String key, long value)`
- `void putFloat(String key, float value)`
- `void putDouble(String key, double value)`
- `void putBoolean(String key, boolean value)`
- `void putByteArray(String key, byte[] value)`
  将键/值存放在节点中。
- `void exportSubtree(OutputStream out)`
  将这个节点及子节点的配置信息写到指定流中。
- `void exportNode(OutputStream out)`
  将这个节点（不包括子节点）配置信息写到指定流中。
- `void importPreferences(InputStream in)`
  导入指定流中的配置信息。

至此，结束了对Java软件部署的讨论。下一章将介绍如何利用异常告诉程序在出现运行问题时如何进行处理。另外，还将介绍一些测试和调试的技巧，以保证程序运行时不会出现过多的错误。