

1. 这个 applet 来自哪里?
2. 在传输过程中代码是否被破坏?

在过去的 50 年里, 数学家和计算机科学家已经开发出各种各样成熟的算法, 用于确保数据和电子签名的完整性, 在 `java.security` 包中包含了许多这类算法的实现, 而且幸运的是, 你无须掌握相应的数学基础知识, 就可以使用 `java.security` 包中的算法。在下面几节中, 我们将要介绍消息摘要是如何检测数据文件中的变化的, 以及数字签名是如何证明签名者的身份的。

#### 10.4.1 消息摘要

消息摘要 (message digest) 是数据块的数字指纹。例如, 所谓的 SHA1 (安全散列算法 #1) 可将任何数据块, 无论其数据有多长, 都压缩为 160 位 (20 字节) 的序列。与真实的指纹一样, 人们希望任何两条不同的消息都不会有相同的 SHA1 指纹。当然, 这是不可能的一因为只存在  $2^{160}$  个 SHA1 指纹, 所以肯定会有某些消息具有相同的指纹。因为  $2^{160}$  是一个很大的数字, 所以存在重复指纹的可能性微乎其微, 那么这种重复的可能性到底小到什么程度呢? 根据 James Walsh 在他的 *True Odds: How Risks Affect Your Everyday Life* ( Merritt Publishing 出版社 1996 年出版) 一书中所叙述的, 人死于雷击的概率为三万分之一。现在, 假设有 9 个人, 比如你不喜欢的 9 个经理或者教授, 你和他们所有的人都死于雷击的概率, 比伪造的消息与原有消息具有相同的 SHA1 指纹的概率还要高。(当然, 可能有你不认识的其他 10 个以上的人会死于雷击, 但这里我们讨论的是你选择的特定的人的死亡概率。)

消息摘要具有两个基本属性:

1. 如果数据的 1 位或者几位改变了, 那么消息摘要也将改变。
2. 拥有给定消息的伪造者无法创建与原消息具有相同摘要的假消息。

当然, 第二个属性又是一个概率问题。让我们来看看下面这位亿万富翁留下的遗嘱:

“我死了之后, 我的财产将由我的孩子平分, 但是, 我的儿子 George 应该拿不到一个子。”这份遗嘱的 SHA1 指纹为:

```
12 5F 09 03 E7 31 30 19 2E A6 E7 E4 90 43 84 B4 38 99 8F 67
```

这位有疑心病的父亲将这份遗嘱交给一位律师保存, 而将指纹交给另一位律师保存。现在, 假设 George 能够贿赂那位保存遗嘱的律师, 他想修改这份遗嘱, 使得 Bill 一无所获。当然, 这需要将原指纹改为下面这样完全不同的位模式:

```
7D F6 AB 08 EB 40 EC CD AB 74 ED E9 86 F9 ED 99 D1 45 B1 57
```

那么 George 能够找到与该指纹相匹配的其他措辞吗? 如果从地球形成之时, 他就很自豪地拥有 10 亿台计算机, 每台计算机每秒钟能处理一百万条信息, 他依然无法找到一个能够替换的遗嘱。

人们已经设计出大量的算法, 用于计算这些消息摘要, 其中最著名的两种算法是 SHA1 和 MD5。SHA1 是由美国国家标准和技术学会开发的加密散列算法, MD5 是由麻省理工学院的 Ronald Rivest 发明的算法。这两种算法都使用了独特巧妙的方法对消息中的各个位进行

扰乱。如果要了解这些方法的详细信息，请参阅 William Stallings 撰写的 *Cryptography and Network Security* (第 7 版) 一书，该书由 Prentice Hall 出版社于 2017 年出版。但是，人们在这两种算法中发现了某些微妙的规律性，因此美国国家标准和技术学会建议切换到更强的加密算法上，Java 支持 SHA-2 和 SHA-3 算法集。

`MessageDigest` 类是用于创建封装了指纹算法的对象的“工厂”，它的静态方法 `getInstance` 返回继承了 `MessageDigest` 类的某个类的对象。这意味着 `MessageDigest` 类能够承担下面的双重职责：

- 作为一个工厂类。
- 作为所有消息摘要算法的超类。

例如，下面是如何获取一个能够计算 SHA 指纹的对象的方法：

```
MessageDigest alg = MessageDigest.getInstance("SHA-1");
```

在获取 `MessageDigest` 对象之后，可以通过反复调用 `update` 方法，将信息中的所有字节提供给该对象。例如，下面的代码将文件中的所有字节传给上面创建的 `alg` 对象，以执行指纹算法：

```
InputStream in = . . .;
int ch;
while ((ch = in.read()) != -1)
    alg.update((byte) ch);
```

另外，如果这些字节存放在一个数组中，那就可以一次完成整个数组的更新：

```
byte[] bytes = . . .;
alg.update(bytes);
```

当完成上述操作后，调用 `digest` 方法。该方法按照指纹算法的要求补齐输入，并且进行相应的计算，然后以字节数组的形式返回消息摘要。

```
byte[] hash = alg.digest();
```

程序清单 10-16 中的程序计算了一个消息摘要，可以在命令行中指定文件和算法：

```
java hash.Digest hash/input.txt SHA-1
```

如果没有提供命令行参数，那么就会提示你输入文件名和算法名。

### 程序清单 10-16 hash/Digest.java

```
1 package hash;
2
3 import java.io.*;
4 import java.nio.file.*;
5 import java.security.*;
6 import java.util.*;
7
8 /**
9 * This program computes the message digest of a file.
10 * @version 1.21 2018-04-10
11 * @author Cay Horstmann
```

```

12  /*
13  public class Digest
14  {
15      /**
16      * @param args args[0] is the filename, args[1] is optionally the algorithm
17      * (SHA-1, SHA-256, or MD5)
18      */
19      public static void main(String[] args) throws IOException, GeneralSecurityException
20      {
21          var in = new Scanner(System.in);
22          String filename;
23          if (args.length >= 1)
24              filename = args[0];
25          else
26          {
27              System.out.print("File name: ");
28              filename = in.nextLine();
29          }
30          String algname;
31          if (args.length >= 2)
32              algname = args[1];
33          else
34          {
35              System.out.println("Select one of the following algorithms: ");
36              for (Provider p : Security.getProviders())
37                  for (Provider.Service s : p.getServices())
38                      if (s.getType().equals("MessageDigest"))
39                          System.out.println(s.getAlgorithm());
40              System.out.print("Algorithm: ");
41              algname = in.nextLine();
42          }
43          MessageDigest alg = MessageDigest.getInstance(algname);
44          byte[] input = Files.readAllBytes(Paths.get(filename));
45          byte[] hash = alg.digest(input);
46          for (int i = 0; i < hash.length; i++)
47              System.out.printf("%02X ", hash[i] & 0xFF);
48          System.out.println();
49      }
50  }

```

### API **java.security.MessageDigest 1.1**

- static MessageDigest getInstance(String algorithmName)

返回实现指定算法的 MessageDigest 对象。如果没有提供该算法，则抛出一个 NoSuchAlgorithmException 异常。

- void update(byte input)
- void update(byte[] input)
- void update(byte[] input, int offset, int len)  
使用指定的字节来更新摘要。
- byte[] digest()

完成散列计算，返回计算所得的摘要，并复位算法对象。

- void reset()

重置摘要。

#### 10.4.2 消息签名

在上一节中，我们介绍了如何计算消息摘要，即原始消息的指纹的方法。如果消息改变了，那么改变后的消息的指纹与原消息的指纹将不匹配。如果消息和它的指纹是分开传送的，那么接收者就可以检查消息是否被篡改过。但是，如果消息和指纹同时被截获了，对消息进行修改，再重新计算指纹，就是一件很容易的事情。毕竟，消息摘要算法是公开的，不需要使用任何密钥。在这种情况下，假消息和新指纹的接收者永远不会知道消息已经被篡改。数字签名解决了这个问题。

为了了解数字签名的工作原理，我们需要解释关于公共密钥加密技术领域中的几个概念。公共密钥加密技术是基于公共密钥和私有密钥这两个基本概念的。它的设计思想是你可以在公共密钥告诉世界上的任何人，但是，只有自己才持有私有密钥，重要的是你要保护你的私有密钥，不将它泄漏给其他任何人。这些密钥之间存在一定的数学关系，但是这种关系的具体性质对于实际的编程来说并不重要。（如果你有兴趣，可以参阅 <http://www.cacr.math.uwaterloo.ca/~hac/> 站点上的 *The Handbook of Applied Cryptography* 一书。）

密钥非常长，而且很复杂。例如，下面是一对匹配的数字签名算法（DSA）的公共密钥和私有密钥。

公共密钥：

```
p: fca682ce8e12caba26efccf7110e526db078b05edecbcd1eb4a208f3ae1617ae01f35b91a47e6df63413c5e12  
ed0899bcd132acd50d99151bcd43ee737592e17
```

```
q: 962eddcc369cba8ebb260ee6b6a126d9346e38c5
```

```
g: 678471b27a9cf44ee91a49c5147db1a9aaf244f05a434d6486931d2d14271b9e35030b71fd73da179069b32e2  
935630e1c2062354d0da20a6c416e50be794ca4
```

```
y: c0b6e67b4ac098eb1a32c5f8c4c1f0e7e6fb9d832532e27d0bdab9ca2d2a8123ce5a8018b8161a760480fadd0  
40b927281ddb22cb9bc4df596d7de4d1b977d50
```

私有密钥：

```
p: fca682ce8e12caba26efccf7110e526db078b05edecbcd1eb4a208f3ae1617ae01f35b91a47e6df63413c5e12  
ed0899bcd132acd50d99151bcd43ee737592e17
```

```
q: 962eddcc369cba8ebb260ee6b6a126d9346e38c5
```

```
g: 678471b27a9cf44ee91a49c5147db1a9aaf244f05a434d6486931d2d14271b9e35030b71fd73da179069b32e2  
935630e1c2062354d0da20a6c416e50be794ca4
```

```
x: 146c09f881656cc6c51f27ea6c3a91b85ed1d70a
```

在现实中，几乎不可能用一个密钥去推算出另一个密钥。也就是说，即使每个人都知道你的公共密钥，不管他们拥有多少计算资源，他们一辈子也无法计算出你的私有密钥。

任何人都无法根据公共密钥来推算私有密钥，这似乎让人难以置信。但是时至今日，还没有人能够找到一种算法，来为现在常用的加密算法进行这种推算。如果密钥足够长，那么要是使用穷举法——也就是直接试验所有可能的密钥——所需要的计算机将比用太阳系中的所有原子来制造的计算机还要多，而且还得花费数千年的时间。当然，可能会有人提出比穷举更灵活的计算密钥的算法。例如，RSA 算法（该加密算法由 Rivest、Shamir 和 Adleman 发明）就利用了对数值巨大的数字进行因数分解的困难性。在最近 20 年里，许多优秀的数学家都在尝试提出好的因数分解算法，但是迄今为止都没有成功。据此，大多数密码学者认为，拥有 2000 位或者更多位“模数”的密钥目前是完全安全的，可以抵御任何攻击。DSA 被认为具有类似的安全性。

图 10-9 展示了实践中这种机制是如何工作的。

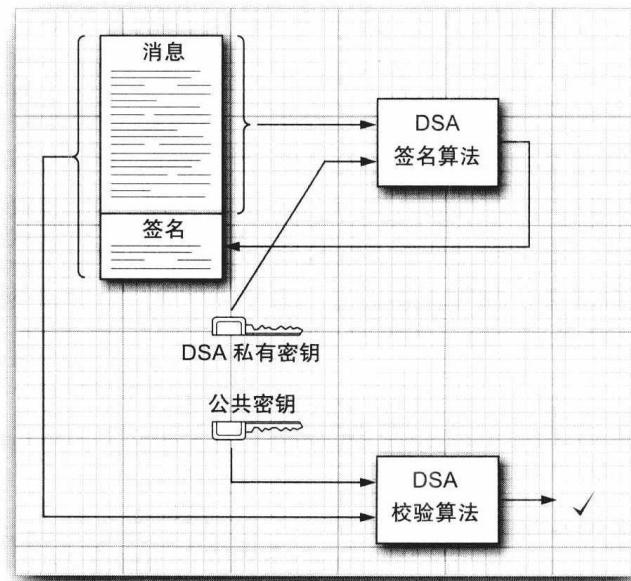


图 10-9 使用 DSA 进行公共密钥签名的交换

假设 Alice 想要给 Bob 发送一个消息，Bob 想知道该消息是否来自 Alice，而不是冒名顶替者。Alice 写好了消息，并且用她的私有密钥对该消息摘要签名。Bob 得到了她的公共密钥的拷贝，然后 Bob 用公共密钥对该签名进行校验。如果通过了校验，则 Bob 可以确认以下两个事实：

1. 原始消息没有被篡改过。
2. 该消息是由 Alice 签名的，她是私有密钥的持有者，该私有密钥就是与 Bob 用于校验的公共密钥相匹配的密钥。

你可以看到私有密钥的安全性为什么是最重要的。如果某个人偷了 Alice 的私有密钥，或者政府要求她交出私有密钥，那么她就麻烦了。小偷或者政府代表就可以假扮她的身份来发送消息，例如资金转账指令，而其他人则会相信这些消息确实来自于 Alice。

### 10.4.3 校验签名

JDK 配有一个 keytool 程序，该程序是一个命令行工具，用于生成和管理一组证书。我们期望该工具的功能最终能够被嵌入到其他更加用户友好的程序中去。但我们现在要做的是，使用 keytool 工具来展示 Alice 是如何对一个文档进行签名并且将它发送给 Bob 的，而 Bob 又是如何校验该文档确实是由 Alice 签名，而不是冒名顶替的。

keytool 程序负责管理密钥库、证书数据库和私有 / 公有密钥对。密钥库中的每一项都有一个“别名”。下面展示的是 Alice 如何创建一个密钥库 alice.certs 并且用别名生成一个密钥对。

```
keytool -genkeypair -keystore alice.certs -alias alice
```

当新建或者打开一个密钥库时，系统将提示输入密钥库口令，在下面的这个例子中，口令就使用 secret，如果要将 keytool 生成的密钥库用于重要的应用，那么需要选择一个好的口令来保护这个文件。

当生成一个密钥时，系统提示输入下面这些信息：

```
Enter keystore password: secret
Reenter new password: secret
What is your first and last name?
[Unknown]: Alice Lee
What is the name of your organizational unit?
[Unknown]: Engineering
What is the name of your organization?
[Unknown]: ACME Software
What is the name of your City or Locality?
[Unknown]: San Francisco
What is the name of your State or Province?
[Unknown]: CA
What is the two-letter country code for this unit?
[Unknown]: US
Is <CN=Alice Lee, OU=Engineering, O=ACME Software, L=San Francisco, ST=CA, C=US> correct?
[no]: yes
```

keytool 工具使用 X.500 格式的名字，它包含常用名 (CN)、机构单位 (OU)、机构 (O)、地点 (L)、州 (ST) 和国别 (C) 等成分，以确定密钥持有者和证书发行者的身份。

最后，必须设定一个密钥口令，或者按回车键，将密钥库口令作为密钥口令来使用。

假设 Alice 想把她的公共密钥提供给 Bob，她必须导出一个证书文件：

```
keytool -exportcert -keystore alice.certs -alias alice -file alice.cer
```

这时，Alice 就可以把证书发送给 Bob。当 Bob 收到该证书时，他就可以将证书打印出来：

```
keytool -printcert -file alice.cer
```

打印的结果如下：

```
Owner: CN=Alice Lee, OU=Engineering, O=ACME Software, L=San Francisco, ST=CA, C=US
Issuer: CN=Alice Lee, OU=Engineering, O=ACME Software, L=San Francisco, ST=CA, C=US
Serial number: 470835ce
Valid from: Sat Oct 06 18:26:38 PDT 2007 until: Fri Jan 04 17:26:38 PST 2008
Certificate fingerprints:
```

```

MD5: BC:18:15:27:85:69:48:B1:5A:C3:0B:1C:C6:11:B7:81
SHA1: 31:0A:A0:B8:C2:8B:3B:B6:85:7C:EF:C0:57:E5:94:95:61:47:6D:34
Signature algorithm name: SHA1withDSA
Version: 3

```

如果 Bob 想检查他是否得到了正确的证书，可以给 Alice 打电话，让她在电话里读出证书的指纹。

**注释：**有些证书发放者将证书指纹公布在他们的网站上。例如，要检查 `jre/lib/security/cacerts` 目录中的密钥库里的 DigiCert 公司的证书，可以使用 `-list` 选项：

```
keytool -list -v -keystore jre/lib/security/cacerts
```

该密钥库的口令是 `changeit`。在该密钥库中有一个证书是：

```

Owner: CN=DigiCert Assured ID Root G3, OU=www.digicert.com, O=DigiCert Inc, C=US
Issuer: CN=DigiCert Assured ID Root G3, OU=www.digicert.com, O=DigiCert Inc, C=US
Serial number: ba15afalddfa0b54944afcd24a06cec
Valid from: Thu Aug 01 14:00:00 CEST 2013 until: Fri Jan 15 13:00:00 CET 2038
Certificate fingerprints:
    SHA1: F5:17:A2:4F:9A:48:C6:C9:F8:A2:00:26:9F:DC:0F:48:2C:AB:30:89
    SHA256: 7E:37:CB:8B:4C:47:09:0C:AB:36:55:1B:A6:F4:5D:B8:40:68:0F:BA:
              16:6A:95:2D:B1:00:71:7F:43:05:3F:C2

```

通过访问网址 [www.digicert.com/digicert-root-certificates.htm](http://www.digicert.com/digicert-root-certificates.htm)，就可以核实该证书的有效性。

一旦 Bob 信任该证书，他就可以将它导入密钥库中。

```
keytool -importcert -keystore bob.certs -alias alice -file alice.cer
```

**警告：**绝对不要将你并不完全信任的证书导入到密钥库中。一旦证书添加到密钥库中，使用密钥库的任何程序都会认为这些证书可以用来对签名进行校验。

现在 Alice 就可以给 Bob 发送签过名的文档了。`jarsigner` 工具负责对 JAR 文件进行签名和校验，Alice 只需要将文档添加到要签名的 JAR 文件中。

```
jar cvf document.jar document.txt
```

然后她使用 `jarsigner` 工具将签名添加到文件中，她必须指定要使用的密钥库、JAR 文件和密钥的别名。

```
jarsigner -keystore alice.certs document.jar alice
```

当 Bob 收到 JAR 文件时，他可以使用 `jarsigner` 程序的 `-verify` 选项，对文件进行校验。

```
jarsigner -verify -keystore bob.certs document.jar
```

Bob 不需要设定密钥别名。`jarsigner` 程序会在数字签名中找到密钥所有者的 X.500 名字，并在密钥库中搜寻匹配的证书。

如果 JAR 文件没有受到破坏而且签名匹配，那么 `jarsigner` 程序将打印：

```
jar verified.
```

否则，程序将显示一个出错消息。

#### 10.4.4 认证问题

假设你从朋友 Alice 那接收到一个消息，该消息是 Alice 用她的私有密钥签名的，使用的签名方法就是我们刚刚介绍的方法。你可能已经有了她的公共密钥，或者你能够容易地获得她的公共密钥，比如问她要一个密钥拷贝，或者从她的 Web 页中获得密钥。这时，你就可以校验该消息是否是 Alice 签过的，并且有没有被破坏过。现在，假设你从一个声称代表某著名软件公司的陌生人那里获得了一个消息，他要求你运行消息附带的程序。这个陌生人甚至将他的公共密钥的拷贝发送给你，以便让你校验他是否是该消息的作者。你检查后会发现该签名是有效的，这就证明该消息是用匹配的私有密钥签名的，并且没有遭到破坏。

此时你要小心：你仍然不清楚谁写的这条消息。任何人都可以生成一对公共密钥和私有密钥，再用私有密钥对消息进行签名，然后把签名好的消息和公共密钥发送给你。这种确定发送者身份的问题称为“认证问题”。

解决这个认证问题的通常做法是比较简单的。假设陌生人和你有一个你们俩都值得信赖的共同熟人。假设陌生人亲自约见了该熟人，将包含公共密钥的磁盘交给了他。后来，你的熟人与你见面，向你担保他与该陌生人见了面，并且该陌生人确实在那家著名的软件公司工作，然后将磁盘交给你（参见图 10-10）。这样一来，你的熟人就证明了陌生人身份的真实性。

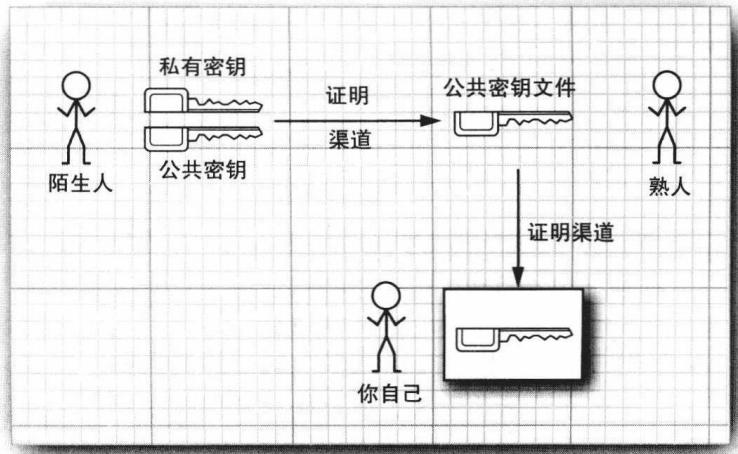


图 10-10 通过一个值得信赖的中间人进行认证

事实上，你的熟人并不需要与你见面。取而代之的是，他可以将他的私有签名应用于陌生人的公共密钥文件之上即可（参见图 10-11）。

当你拿到公共密钥文件之后，就可以检验你的熟人的签名是否真实，由于你信任他，因此你确信他在添加他的签名之前，确实核实了陌生人的身份。

然而，你们之间可能没有共同的熟人。有些信任模型假设你们之间总是存在一个“信任链”——即一个共同熟人的链路——这样你就可以信任该链中的每个成员。当然，实际

情况并不总是这样。你可能信任你的熟人 Alice，而且你知道 Alice 信任 Bob，但是你不了解 Bob，因此你没有把握究竟是不是该信任他。其他的信任模型则假设有一个我们大家都信任的慈善大佬，即一家我们大家都信任的公司。在这样的公司中，如雷贯耳的有 DigiCert、GlobalSign 和 Entrust，它们都提供认证服务。

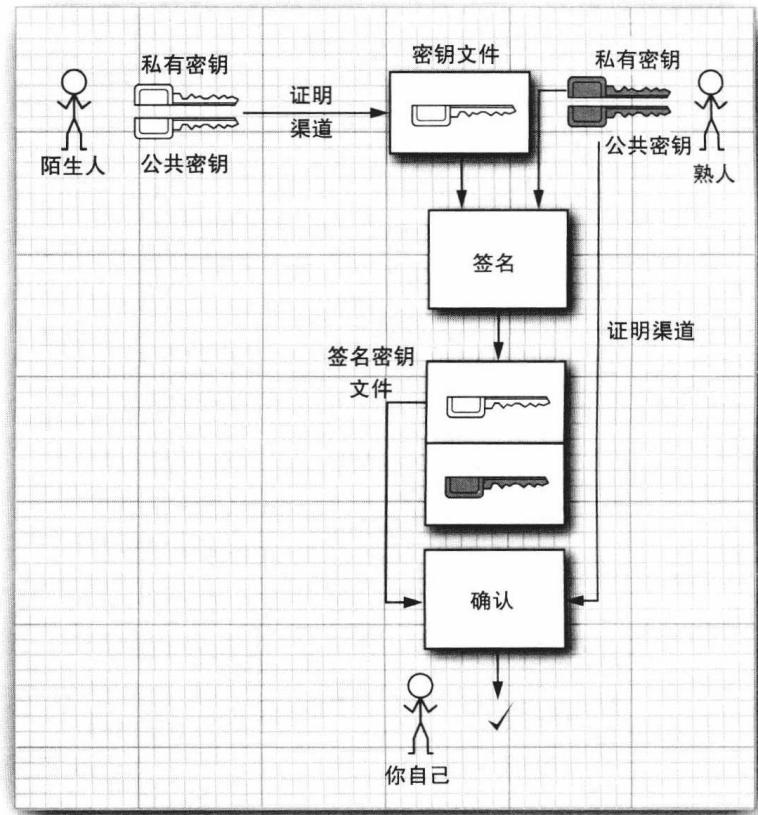


图 10-11 通过值得信赖的中间人的签名进行认证

你常常会遇到由负责担保他人身份的一个或多个实体签署的数字签名，你必须评估一下究竟能够在多大程度上信任这些身份认证人。你可能非常信赖某种特定的证书授权，因为也许你在许多网页中都看到过他们的标志，或者你曾经听说过，每当有新的万能密钥产生时，他们就会要求在一个非常保密的会议室中聚集众多揣着黑色公文包的人进行磋商。

然而，对于实际被认证的对象，你应该抱有一个符合实际的期望：直接在 Web 页面上填一份表格，并支付少量的费用，就可以获得一个“第一类”(class 1) ID，包含在证书中的密钥将被发送到指定的邮件地址。因此，你有理由相信该电子邮件是真实的，但是密钥申请人也可能填入任意名字和机构。还有其他对身份信息的检验更加严格的 ID 类别。例如，如果是“第三类”(class 3) ID，证书授权将要求密钥申请人必须进行身份公证，公证机构将要核实企业申请者的财务信用资质。其他认证机构将采用不同的认证程序。因此，当你收到一条经过认证的消息时，重要的是你应该明白它实际上认证了什么。

#### 10.4.5 证书签名

在 10.4.3 节中，你已经看到了 Alice 如何使用自签名的证书向 Bob 分发公共密钥。但是，Bob 需要通过校验 Alice 的指纹以确保这个证书是有效的。

假设 Alice 想要给同事 Cindy 发送一条经过签名的消息，但是 Cindy 并不希望因为要校验许多签名指纹而受到困扰。因此，假设有一个 Cindy 信任的实体来校验这些签名。在这个例子中，Cindy 信任 ACME 软件公司的信息资源部。

这个部门负责证书授权（CA）的运作。ACME 的每个人在其密钥库中都有 CA 的公共密钥，这是由一个专门负责详细核查密钥指纹的系统管理员安装的。CA 对 ACME 雇员的密钥进行签名，当他们在安装彼此的密钥时，密钥库将隐含地信任这些密钥，因为它们是由一个可信任的密钥签名的。

下面展示了可以如何模仿这个过程。首先需要创建一个密钥库 `acmesoft.Certs`，生成一个密钥对并导出公共密钥。

```
keytool -genkeypair -keystore acmesoft.certs -alias acmeroot
keytool -exportcert -keystore acmesoft.certs -alias acmeroot -file acmeroot.cer
```

其中的公共密钥被导入到了一个自签名的证书中，然后将其添加到每个雇员的密钥库中：

```
keytool -importcert -keystore cindy.certs -alias acmeroot -file acmeroot.cer
```

如果 Alice 要发送消息给 Cindy 以及 ACME 软件公司的其他任何人，她需要将她自己的证书签名 - 并提交给信息资源部。但是，这个功能在 `keytool` 程序中是缺失的。在本书附带的代码中，我们提供了一个 `CertificateSigner` 类来弥补这个问题。ACME 软件公司的授权机构成员将负责核实 Alice 的身份，并且生成如下的签名证书：

```
java CertificateSigner -keystore acmesoft.certs -alias acmeroot \
    -infile alice.cer -outfile alice_signedby_acmeroot.cer
```

证书签名器程序必须拥有对 ACME 软件公司密钥库的访问权限，并且该公司成员必须知道密钥库的口令，显然这是一项敏感的操作。

现在 Alice 将文件 `alice_signedby_acmeroot.cert` 交给 Cindy 和 ACME 软件公司的其他任何人。或者，ACME 软件公司直接将该文件存储在公司的目录中。请记住，该文件包含了 Alice 的公共密钥和 ACME 软件公司的声明，证明该密钥确实属于 Alice。

现在，Cindy 将签名的证书导入到她的密钥库中：

```
keytool -importcert -keystore cindy.certs -alias alice -file alice_signedby_acmeroot.cer
```

密钥库要进行校验，以确定该密钥是由密钥库中已有的受信任的根密钥签过名的。Cindy 就不必对证书的指纹进行校验了。

一旦 Cindy 添加了根证书和经常给她发送文档的人的证书后，她就再也不用担心密钥库了。

#### 10.4.6 证书请求

在前一节中，我们用密钥库和 `CertificateSigner` 工具模拟了一个 CA。但是，大多数 CA

都运行着更加复杂的软件来管理证书，并且使用的证书格式也略有不同。本节将展示与这些软件包进行交互时需要增加的处理步骤。

我们将用 OpenSSL 软件包作为实例。许多 Linux 系统和 Mac OS X 都预装了这个软件，并且用于 Windows 的 Cygwin 端口也可用这个软件，你也可以到 <http://www.openssl.org> 网站下载。

为了创建一个 CA，需要运行 CA 脚本，其确切位置依赖于你的操作系统。在 Ubuntu 上，运行

```
/usr/lib/ssl/misc/CA.pl -newca
```

这个脚本会在当前目录中创建一个 demoCA 子目录，这个目录包含了一个根密钥对，并存储了证书与证书撤销列表。

你希望将这个公共密钥导入到所有雇员的 Java 密钥库中，但是它的格式是隐私增强型邮件（PEM）格式，而不是密钥库更容易接受的 DER 格式。将文件 demoCA/cacert.pem 复制成文件 acmeroot.pem，然后在文本编辑器中打开这个文件。移除下面这行之前的所有内容：

```
-----BEGIN CERTIFICATE-----
```

以及下面这行之后的所有内容：

```
-----END CERTIFICATE-----
```

现在可以按照通常的方式将 acmeroot.pem 导入到每个密钥库中了：

```
keytool -importcert -keystore cindy.certs -alias alice -file acmeroot.pem
```

这看起来有点不可思议，keytool 竟然不能自己去执行这种编辑操作。

要对 Alice 的公共密钥签名，需要生成一个证书请求，它包含这个 PEM 格式的证书：

```
keytool -certreq -keystore alice.store -alias alice -file alice.pem
```

要签名这个证书，需要运行：

```
openssl ca -in alice.pem -out alice_signedby_acmeroot.pem
```

与前面一样，在 alice\_signedby\_acmeroot.pem 中切除 BEGIN CERTIFICATE/END CERTIFICATE 标记之外的所有内容。然后，将其导入到密钥库中：

```
keytool -importcert -keystore cindy.certs -alias alice -file alice_signedby_acmeroot.pem
```

你可以使用相同的步骤，使一个证书得到公共证书权威机构的签名。

#### 10.4.7 代码签名

认证技术最重要的一个应用是对可执行程序进行签名。如果从网上下载一个程序，自然会关心该程序可能带来的危害，例如，该程序可能已经感染了病毒。如果知道代码从何而来，并且它从离开源头后就没有被篡改过，那么放心程度会比不清楚这些信息时要高得多。

本节将展示如何对 JAR 文件签名，以及如何配置 Java 以校验这种签名。这种能力是为

applet 和 Java Web Start 应用而设计的。这些技术已经不再被广泛使用了，但是你仍旧需要在遗留产品中支持它们。

当 Java 首次发布时，applet 在加载之后就运行于具有有限权限的“沙盒”之中。如果用户想要使用可以访问本地文件系统、创建网络连接等诸如此类功能的 applet，那么就必须明确同意允许其运行。为了确保 applet 代码不会在传输过程中被篡改，必须对其进行数字签名。

下面是一个具体例子。假设当你在因特网上冲浪时，遇到了一个 Web 站点，倘若你为它授予了需要的权限，它就会运行一个来自不明提供商的 applet（参见图 10-12）。这样的程序是用由证书权威机构发放的“软件开发者”证书进行签名的。弹出的对话框用于确定软件开发者和证书发放者的身份。现在，你需要决定是否对该程序授权。

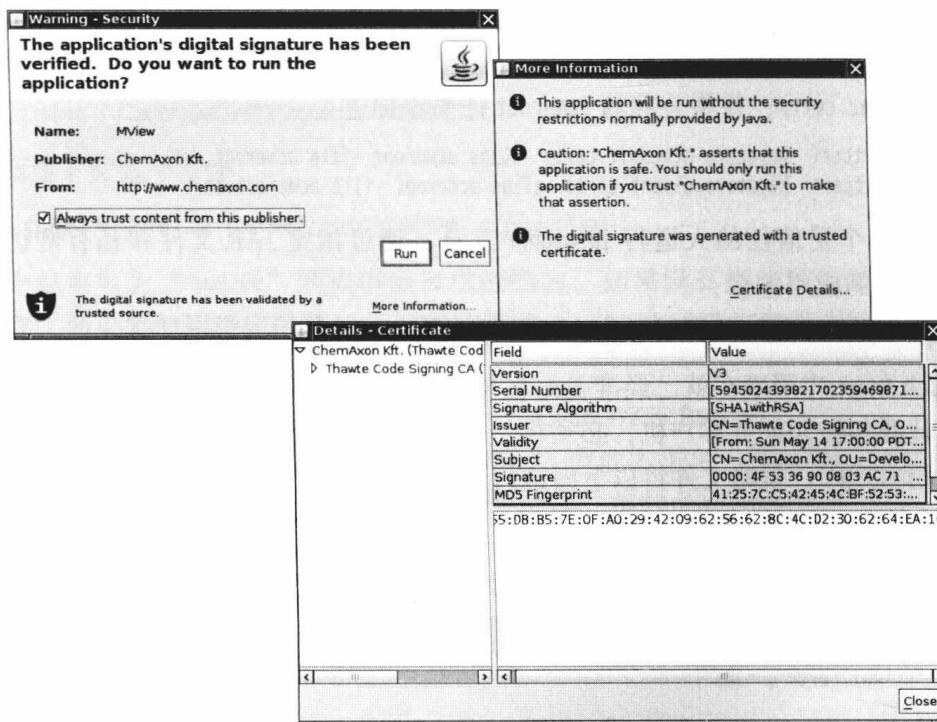


图 10-12 启动一个签过名的 applet

那么什么样的因素可能会影响你的决定呢？假设下面是你已经了解的情况：

1. Thawte 公司将一个证书卖给了软件开发人员。
2. 程序确实是用该证书签名的，并且在传输过程中没有被篡改过。
3. 该证书确实是由 Thawte 签名的，它是用本地 cacerts 文件中的公共密钥校验的。

当然，上面这些信息都不能告诉你代码是否可以安全运行。如果你只知道供应商的名字，以及 Thawte 公司卖给了他们一个软件开发者证书这个事实，那么你会信赖该供应商吗？这种方式显然没什么意义。

对内联网部署，证书更有用。管理员可以在本地机器上安装策略文件和证书，使得在启动从受信源而来的代码时可以无须任何用户交互。无论何时，只要 Java 插件工具加载了签名

的文档，它就会向策略文件索要权限，并向密钥库索要签名。

在本节的剩余部分，我们将要介绍如何建立策略文件，来为已知来源的代码赋予特定的权限。

假设 ACME 软件公司想让它的用户运行某些需要具备本地文件访问权限的程序，并且想要通过浏览器将这些程序部署为 Web Start 应用。

正如在本章前面部分看到的那样，ACME 可以根据程序的代码基来确定它们的身份，但是那将意味着每当程序移动到不同的 Web 服务器时，ACME 都需要更新策略文件。为此，ACME 决定对含有程序代码的 JAR 文件进行签名。

首先，ACME 生成根证书：

```
keytool -genkeypair -keystore acmesoft.certs -alias acmeroot
```

当然，包含私有根密钥的密钥库必须存放在一个安全的地方。因此，我们为公共证书建立第二个密钥库 Client.certs，并将公共的 acmeroot 证书添加进去。

```
keytool -exportcert -keystore acmesoft.certs -alias acmeroot -file acmeroot.cer
keytool -importcert -keystore client.certs -alias acmeroot -file acmeroot.cer
```

ACME 中某个受信任的人运行 jarsigner 工具，通过指定 JAR 文件和私有密钥的别名来对任何想要签名的应用签名：

```
jarsigner -keystore acmesoft.certs ACMEApp.jar acmeroot
```

被签名的 Web Start 应用现在就已经准备好在 Web 服务器中部署了。

接着，让我们转而配置客户机，必须将一个策略文件发布到每一台客户机上。

为了引用密钥库，策略文件将下面这行开头：

```
keystore "keystoreURL", "keystoreType";
```

其中，URL 可以是绝对的或相对的，其中相对 URL 是相对于策略文件的位置而言的。如果密钥库是由 keytool 工具生成的，则它的类型是 JKS。例如：

```
keystore "client.certs", "JKS";
```

grant 子句可以有 signedBy “alias” 后缀，例如：

```
grant signedBy "acmeroot"
{
    ...
};
```

所有可以用与别名相关联的公共密钥进行校验的签名代码现在都已经在 grant 语句中被授予了权限。

## 10.5 加密

到现在为止，我们已经介绍了一种在 Java 安全 API 中实现的重要密码技术，即通过数字签名进行的认证。安全性的第二个重要方面是加密。当信息通过认证之后，该信息本身是直

白可见的。数字签名只不过负责检验信息有没有被篡改过。相比之下，信息被加密后，是不可见的，只能用匹配的密钥进行解密。

认证对于代码签名已足够了——没必要将代码隐藏起来。但是，当 applet 或者应用程序传输机密信息时，比如信用卡号码和其他个人数据等，就有必要进行加密了。

过去，由于专利和出口控制的原因，许多公司被禁止提供高强度的加密技术。幸运的是，现在对加密技术的出口控制已经不是那么严格了，某些重要算法的专利也已到期。现在，Java 已经有了出色的加密支持，它已经成为标准类库的一部分。

### 10.5.1 对称密码

“Java 密码扩展”包含了一个 Cipher 类，该类是所有加密算法的超类。通过调用下面的 getInstance 方法可以获得一个密码对象：

```
Cipher cipher = Cipher.getInstance(algorithmName);
```

或者调用下面这个方法：

```
Cipher cipher = Cipher.getInstance(algorithmName, providerName);
```

JDK 中是由名为“SunJCE”的提供商提供密码的，如果没有指定其他提供商，则会默认为该提供商。如果要使用特定的算法，而对该算法 Oracle 公司没有提供支持，那么也可以指定其他的提供商。

算法名称是一个字符串，比如“AES”或者“DES/CBC/PKCS5Padding”。

DES，即数据加密标准，是一个密钥长度为 56 位的古老的分组密码。DES 加密算法在现在看来已经是过时了，因为可以用穷举法将它破译。更好的选择是采用它的后续版本，即高级加密标准（AES），更多详细信息，请访问网址 <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf>。我们在示例中使用了 AES。

一旦获得了一个密码对象，就可以通过设置模式和密钥来对它初始化。

```
int mode = . . .;
Key key = . . .;
cipher.init(mode, key);
```

模式有以下几种：

```
Cipher.ENCRYPT_MODE
Cipher.DECRYPT_MODE
Cipher.WRAP_MODE
Cipher.UNWRAP_MODE
```

wrap 和 unwrap 模式会用一个密钥对另一个密钥进行加密，具体例子请参见下一节。

现在可以反复调用 update 方法来对数据块进行加密。

```
int blockSize = cipher.getBlockSize();
var inBytes = new byte[blockSize];
. . . // read inBytes
int outputSize= cipher.getOutputSize(blockSize);
```

```

var outBytes = new byte[outputSize];
int outLength = cipher.update(inBytes, 0, outputSize, outBytes);
. . . // write outBytes

```

完成上述操作后，还必须调用一次 `doFinal` 方法。如果还有最后一个输入数据块（其字节数小于 `blockSize`），那么就要调用：

```
outBytes = cipher.doFinal(inBytes, 0, inLength);
```

如果所有的输入数据都已经加密，则用下面的方法调用来代替：

```
outBytes = cipher.doFinal();
```

对 `doFinal` 的调用是必需的，因为它会对最后的块进行“填充”。就拿 DES 密码来说，它的数据块的大小是 8 字节。假设输入数据的最后一个数据块少于 8 字节，当然我们可以将其余的字节全部用 0 填充，从而得到一个 8 字节的最终数据块，然后对它进行加密。但是，当对数据块进行解密时，数据块的结尾会附加若干个 0 字节，因此它与原始输入文件之间会略有不同。这肯定是个问题，我们需要一个填充方案来避免这个问题。常用的填充方案是 RSA Security 公司在公共密钥密码标准 # 5 中（Public Key Cryptography Standard, PKCS）描述的方案（该方案的网址为 <https://tools.ietf.org/html/rfc2898>）。

在该方案中，最后一个数据块不是全部用填充值 0 进行填充，而是用等于填充字节数量的值作为填充值进行填充。换句话说，如果 L 是最后一个（不完整的）数据块，那么它将按如下方式进行填充：

L 01	if length(L) = 7
L 02 02	if length(L) = 6
L 03 03 03	if length(L) = 5
. . .	
L 07 07 07 07 07 07 07	if length(L) = 1

最后，如果输入的数据长度确实能被 8 整除，那么就会将下面这个数据块：

```
08 08 08 08 08 08 08 08
```

附加到数据块后，并进行加密。在解密时，明文的最后一个字节就是要丢弃的填充字符数。

### 10.5.2 密钥生成

为了加密，我们需要生成密钥。每个密码都有不同的用于密钥的格式，我们需要确保密钥的生成是随机的。这需要遵循下面的步骤：

1. 为加密算法获取 `KeyGenerator`。

2. 用随机源来初始化密钥发生器。如果密码块的长度是可变的，还需要指定期望的密码块长度。

3. 调用 `generateKey` 方法。

例如，下面是如何生成 AES 密钥的方法：

```
KeyGenerator keygen = KeyGenerator.getInstance("AES");
```

```

var random = new SecureRandom(); // see below
keygen.init(random);
Key key = keygen.generateKey();

```

或者，可以从一组固定的原生数据（也许是由于口令或者随机击键产生的）中生成一个密钥，这时可以使用如下的 SecretKeyFactory：

```

byte[] keyData = . . .; // 16 bytes for AES
var key = new SecretKeySpec(keyData, "AES");

```

如果要生成密钥，必须使用“真正的随机”数。例如，在 Random 类中的常规的随机数发生器。是根据当前的日期和时间来产生随机数的，因此它不够随机。假设计算机时钟可以精确到 1/10 秒，那么，每天最多存在 864 000 个种子。如果攻击者知道发布密钥的日期（通常可以由消息日期或证书有效日期推算出来），那么就可以很容易地生成那一天所有可能的种子。

SecureRandom 类产生的随机数，远比由 Random 类产生的那些数字安全得多。你仍然需要提供一个种子，以便在一个随机点上开始生成数字序列。要这样做，最好的方法是从一个诸如白噪声发生器之类的硬件设备那里获取输入。另一个合理的随机输入源是请用户在键盘上进行随心所欲的盲打，但是每次敲击键盘只为随机种子提供 1 位或者 2 位。一旦你在字节数组中收集到这种随机位后，就可以将它传递给 setSeed 方法。

```

var secrand = new SecureRandom();
var b = new byte[20];
// fill with truly random bits
secrand.setSeed(b);

```

如果没有为随机数发生器提供种子，那么它将通过启动线程，使它们睡眠，然后测量它们被唤醒的准确时间，以此来计算自己的 20 个字节的种子。

**注释：**这个算法仍然未被认为是安全的。而且，在过去，依靠对诸如硬盘访问时间之类的其他的计算机组件进行计时的算法，后来也被证明并不是完全随机的。

本节结尾处的示例程序将应用 AES 密码（参见程序清单 10-17）。程序清单 10-18 中的 Crpt 工具方法将会在其他示例中被复用。如果要使用该程序，首先要生成一个密钥，运行如下命令行：

密钥就被保存在 secret.key 文件中了。

```
java aes.AESTest -genkey secret.key
```

现在可以用如下命令进行加密：

```
java aes.AESTest -encrypt plaintextFile encryptedFile secret.key
```

用如下命令进行解密：

```
java aes.AESTest -decrypt encryptedFile decryptedFile secret.key
```

该程序非常直观。使用 -genkey 选项将产生一个新的密钥，并且将其序列化到给定的文件中。该操作需要花费较长的时间，因为密钥随机生成器的初始化非常耗费时间。-encrypt

和 -decrypt 选项都调用相同的 crypt 方法，而 crypt 方法会调用密码的 update 和 doFinal 方法。请注意 update 方法和 doFinal 方法是怎样被调用的，只要输入数据块具有全长度（长度能够被 8 整除），就要调用 update 方法，而如果输入数据块不具有全长度（长度不能被 8 整除，此时需要填充），或者没有更多额外的数据（以便生成一个填充字节），那么就要调用 doFinal 方法。

### 程序清单 10-17 aes/AESTest.java

```

1 package aes;
2
3 import java.io.*;
4 import java.security.*;
5 import javax.crypto.*;
6
7 /**
8 * This program tests the AES cipher. Usage:<br>
9 * java aes.AESTest -genkey keyfile<br>
10 * java aes.AESTest -encrypt plaintext encrypted keyfile<br>
11 * java aes.AESTest -decrypt encrypted decrypted keyfile<br>
12 * @author Cay Horstmann
13 * @version 1.02 2018-05-01
14 */
15 public class AESTest
16 {
17     public static void main(String[] args)
18         throws IOException, GeneralSecurityException, ClassNotFoundException
19     {
20         if (args[0].equals("-genkey"))
21         {
22             KeyGenerator keygen = KeyGenerator.getInstance("AES");
23             var random = new SecureRandom();
24             keygen.init(random);
25             SecretKey key = keygen.generateKey();
26             try (var out = new ObjectOutputStream(new FileOutputStream(args[1])))
27             {
28                 out.writeObject(key);
29             }
30         }
31         else
32         {
33             int mode;
34             if (args[0].equals("-encrypt")) mode = Cipher.ENCRYPT_MODE;
35             else mode = Cipher.DECRYPT_MODE;
36
37             try (var keyIn = new ObjectInputStream(new FileInputStream(args[3]));
38                  var in = new FileInputStream(args[1]);
39                  var out = new FileOutputStream(args[2]))
40             {
41                 var key = (Key) keyIn.readObject();
42                 Cipher cipher = Cipher.getInstance("AES");
43                 cipher.init(mode, key);
44                 Util.crypt(in, out, cipher);
45             }
46         }
47     }
48 }
```

```

45     }
46 }
47 }
48 }
```

**程序清单 10-18 aes/Util.java**

```

1 package aes;
2
3 import java.io.*;
4 import java.security.*;
5 import javax.crypto.*;
6
7 public class Util
8 {
9     /**
10      * Uses a cipher to transform the bytes in an input stream and sends the transformed bytes
11      * to an output stream.
12      * @param in the input stream
13      * @param out the output stream
14      * @param cipher the cipher that transforms the bytes
15      */
16     public static void crypt(InputStream in, OutputStream out, Cipher cipher)
17         throws IOException, GeneralSecurityException
18     {
19         int blockSize = cipher.getBlockSize();
20         int outputSize = cipher.getOutputSize(blockSize);
21         var inBytes = new byte[blockSize];
22         var outBytes = new byte[outputSize];
23
24         int inLength = 0;
25         var done = false;
26         while (!done)
27         {
28             inLength = in.read(inBytes);
29             if (inLength == blockSize)
30             {
31                 int outLength = cipher.update(inBytes, 0, blockSize, outBytes);
32                 out.write(outBytes, 0, outLength);
33             }
34             else done = true;
35         }
36         if (inLength > 0) outBytes = cipher.doFinal(inBytes, 0, inLength);
37         else outBytes = cipher.doFinal();
38         out.write(outBytes);
39     }
40 }
```

**API javax.crypto.Cipher 1.4**

- static Cipher getInstance(String algorithmName)
- static Cipher getInstance(String algorithmName, String providerName)

返回实现了指定加密算法的 Cipher 对象。如果未提供该算法，则抛出一个 NoSuchAlgorithm-

Exception 异常。

- `int getBlockSize()`

返回密码块的大小，如果该密码不是一个分组密码，则返回 0。

- `int getOutputSize(int inputLength)`

如果下一个输入数据块拥有给定的字节数，则返回所需的输出缓冲区的大小。本方法的运行要考虑到密码对象中所有已缓冲的字节数量。

- `void init(int mode, Key key)`

对加密算法对象进行初始化。Mode 是 ENCRYPT\_MODE, DECRYPT\_MODE, WRAP\_MODE, 或者 UNWRAP\_MODE 之一。

- `byte[] update(byte[] in)`

- `byte[] update(byte[] in, int offset, int length)`

- `int update(byte[] in, int offset, int length, byte[] out)`

对输入数据块进行转换。前两个方法返回输出，第三个方法返回放入 out 的字节数。

- `byte[] doFinal()`

- `byte[] doFinal(byte[] in)`

- `byte[] doFinal(byte[] in, int offset, int length)`

- `int doFinal(byte[] in, int offset, int length, byte[] out)`

转换输入的最后一个数据块，并刷新该加密算法对象的缓冲。前三个方法返回输出，第四个方法返回放入 out 的字节数。

### javax.crypto.KeyGenerator 1.4

- `static KeyGenerator getInstance(String algorithmName)`

返回实现指定加密算法的 KeyGenerator 对象。如果未提供该加密算法，则抛出一个 NoSuchAlgorithmException 异常。

- `void init(SecureRandom random)`

- `void init(int keySize, SecureRandom random)`

对密钥生成器进行初始化。

- `SecretKey generateKey()`

生成一个新的密钥。

### javax.crypto.spec.SecretKeySpec 1.4

- `SecretKeySpec(byte[] key, String algorithmName)`

创建一个密钥描述规格说明。

### 10.5.3 密码流

JCE 库提供了一组使用便捷的流类，用于对流数据进行自动加密或解密。例如，下面是对文件数据进行加密的方法：

```
Cipher cipher = . . .;
cipher.init(Cipher.ENCRYPT_MODE, key);
var out = new CipherOutputStream(new FileOutputStream(outputFileName), cipher);
var bytes = new byte[BLOCKSIZE];
int inLength = getData(bytes); // get data from data source
while (inLength != -1)
{
    out.write(bytes, 0, inLength);
    inLength = getData(bytes); // get more data from data source
}
out.flush();
```

同样地，可以使用 `CipherInputStream`，对文件的数据进行读取和解密：

```
Cipher cipher = . . .;
cipher.init(Cipher.DECRYPT_MODE, key);
var in = new CipherInputStream(new FileInputStream(inputFileName), cipher);
var bytes = new byte[BLOCKSIZE];
int inLength = in.read(bytes);
while (inLength != -1)
{
    putData(bytes, inLength); // put data to destination
    inLength = in.read(bytes);
}
```

密码流类能够透明地调用 `update` 和 `doFinal` 方法，所以非常方便。

#### API `javax.crypto.CipherInputStream 1.4`

- `CipherInputStream(InputStream in, Cipher cipher)`

构建一个输入流，以读取 `in` 中的数据，并且使用指定的密码对数据进行解密和加密。

- `int read()`
- `int read(byte[] b, int off, int len)`

读取输入流中的数据，该数据会被自动解密和加密。

#### API `javax.crypto.CipherOutputStream 1.4`

- `CipherOutputStream(OutputStream out, Cipher cipher)`

构建一个输出流，以便将数据写入 `out`，并且使用指定的密码对数据进行加密和解密。

- `void write(int ch)`
- `void write(byte[] b, int off, int len)`

将数据写入输出流，该数据会被自动加密和解密。

- `void flush()`

刷新密码缓冲区，如果需要的话，执行填充操作。

### 10.5.4 公共密钥密码

在前面的小节中看到的 AES 密码是一种对称密码，加密和解密都使用相同的密钥。对称密码的致命缺点在于密码的分发。如果 Alice 给 Bob 发送了一个加密的方法，那么 Bob 需

要使用与 Alice 相同的密钥。如果 Alice 修改了密钥，那么她必须在给 Bob 发送信息的同时，还要通过安全信道发送新的密钥，但是也许她并没有到达 Bob 的安全信道，这也正是她必须对她发送给 Bob 的信息进行加密的原因。

公共密钥密码技术解决了这个问题。在公共密钥密码中，Bob 拥有一个密钥对，包括一个公共密钥和一个相匹配的私有密钥。Bob 可以在任何地方发布公共密钥，但是他必须严格保守他的私有密钥。Alice 只需要使用公共密钥对她发送给 Bob 的信息进行加密即可。

实际上，加密过程并没有那么简单。所有已知的公共密钥算法的操作速度都比对称密钥算法（比如 DES 或 AES 等）慢得多，使用公共密钥算法对大量的信息进行加密是不切实际的。但是，如果像下面这样，将公共密钥密码与快速的对称密码结合起来，这个问题就可以得到解决：

1. Alice 生成一个随机对称加密密钥，她用该密钥对明文进行加密。
2. Alice 用 Bob 的公共密钥给对称密钥进行加密。
3. Alice 将加密后的对称密钥和加密后的明文同时发送给 Bob。
4. Bob 用他的私有密钥给对称密钥解密。
5. Bob 用解密后的对称密钥给信息解密。

除了 Bob 之外，其他人无法给对称密钥进行解密，因为只有 Bob 拥有解密的私有密钥。这样，昂贵的公共密钥加密技术就可以只应用于少量的关键数据的加密。

最常见的公共密钥算法是 Rivest、Shamir 和 Adleman 发明的 RSA 算法。直到 2000 年 10 月，该算法一直受 RSA Security 公司授予的专利保护。该专利的转让许可证价格昂贵，通常要支付 3% 的专利权使用费，每年至少付款 50 000 美元。现在该加密算法已经公开。

如果要使用 RSA 算法，就需要一对公共 / 私有密钥。你可以按如下方法使用 Key-Pair-Generator 来获得：

```
KeyPairGenerator pairgen = KeyPairGenerator.getInstance("RSA");
var random = new SecureRandom();
pairgen.initialize(KEYSIZE, random);
KeyPair keyPair = pairgen.generateKeyPair();
Key publicKey = keyPair.getPublic();
Key privateKey = keyPair.getPrivate();
```

程序清单 10-19 中的程序有三个选项。-genkey 选项用于产生一个密钥对，-encrypt 选项用于生成 AES 密钥，并且用公共密钥对其进行包装。

```
Key key = . . .; // an AES key
Key publicKey = . . .; // a public RSA key
Cipher cipher = Cipher.getInstance("RSA");
cipher.init(Cipher.WRAP_MODE, publicKey);
byte[] wrappedKey = cipher.wrap(key);
```

然后它会生成一个包含下列内容的文件：

- 包装过的密钥的长度。
- 包装过的密钥字节。
- 用 AES 密钥加密的明文。

-decrypt 选项用于对这样的文件进行解密。请试运行该程序，首先生成 RSA 密钥：

```
java rsa.RSATest -genkey public.key private.key
```

然后对一个文件进行加密：

```
java rsa.RSATest -encrypt plaintextFile encryptedFile public.key
```

最后，对该文件进行解密，并且检验解密后的文件是否与明文相匹配：

```
java rsa.RSATest -decrypt encryptedFile decryptedFile private.key
```

### 程序清单 10-19 rsa/RSATest.java

```

1 package rsa;
2
3 import java.io.*;
4 import java.security.*;
5 import javax.crypto.*;
6
7 /**
8  * This program tests the RSA cipher. Usage:<br>
9  * java rsa.RSATest -genkey public private<br>
10 * java rsa.RSATest -encrypt plaintext encrypted public<br>
11 * java rsa.RSATest -decrypt encrypted decrypted private<br>
12 * @author Cay Horstmann
13 * @version 1.02 2018-05-01
14 */
15 public class RSATest
16 {
17     private static final int KEYSIZE = 512;
18
19     public static void main(String[] args)
20         throws IOException, GeneralSecurityException, ClassNotFoundException
21     {
22         if (args[0].equals("-genkey"))
23         {
24             KeyPairGenerator pairgen = KeyPairGenerator.getInstance("RSA");
25             var random = new SecureRandom();
26             pairgen.initialize(KEYSIZE, random);
27             KeyPair keyPair = pairgen.generateKeyPair();
28             try (var out = new ObjectOutputStream(new FileOutputStream(args[1])))
29             {
30                 out.writeObject(keyPair.getPublic());
31             }
32             try (var out = new ObjectOutputStream(new FileOutputStream(args[2])))
33             {
34                 out.writeObject(keyPair.getPrivate());
35             }
36         }
37         else if (args[0].equals("-encrypt"))
38         {
39             KeyGenerator keygen = KeyGenerator.getInstance("AES");
40             var random = new SecureRandom();
41             keygen.init(random);
42             SecretKey key = keygen.generateKey();
```

```

43
44     // wrap with RSA public key
45     try (var keyIn = new ObjectInputStream(new FileInputStream(args[3])));
46         var out = new DataOutputStream(new FileOutputStream(args[2]));
47         var in = new FileInputStream(args[1]) )
48     {
49         var publicKey = (Key) keyIn.readObject();
50         Cipher cipher = Cipher.getInstance("RSA");
51         cipher.init(Cipher.WRAP_MODE, publicKey);
52         byte[] wrappedKey = cipher.wrap(key);
53         out.writeInt(wrappedKey.length);
54         out.write(wrappedKey);
55
56         cipher = Cipher.getInstance("AES");
57         cipher.init(Cipher.ENCRYPT_MODE, key);
58         Util.crypt(in, out, cipher);
59     }
60 }
61 else
62 {
63     try (var in = new DataInputStream(new FileInputStream(args[1]));
64         var keyIn = new ObjectInputStream(new FileInputStream(args[3]));
65         var out = new FileOutputStream(args[2]))
66     {
67         int length = in.readInt();
68         var wrappedKey = new byte[length];
69         in.read(wrappedKey, 0, length);
70
71         // unwrap with RSA private key
72         var privateKey = (Key) keyIn.readObject();
73
74         Cipher cipher = Cipher.getInstance("RSA");
75         cipher.init(Cipher.UNWRAP_MODE, privateKey);
76         Key key = cipher.unwrap(wrappedKey, "AES", Cipher.SECRET_KEY);
77
78         cipher = Cipher.getInstance("AES");
79         cipher.init(Cipher.DECRYPT_MODE, key);
80
81         Util.crypt(in, out, cipher);
82     }
83 }
84 }
85 }

```

你现在已经看到了 Java 安全模型是如何允许我们去控制代码的执行的，这是 Java 平台的一个独一无二且越来越重要的方面。你也已经看到了 Java 类库提供的认证和加密服务。

下一章我们将深入讨论高级 Swing 编程和图形化。

# 第 11 章 高级 Swing 和图形化编程

- ▲ 表格
- ▲ 树
- ▲ 高级 AWT

- ▲ 像素图
- ▲ 打印

在本章中，我们继续对卷 I 的 Swing 用户界面工具包和 AWT 图形化编程进行讨论。我们聚焦于可以同时应用于客户端用户界面和服务器端图形图像生成的技术。Swing 有很多复杂的构件来绘制表格和树。通过使用 2D 图形化 API，我们可以产生具有任意复杂度的向量艺术品。ImageIO API 使我们可以操作光栅图像。最终，可以使用打印 API 来生成打印资料和 PostScript 文件。

## 11.1 表格

JTable 构件用于显示二维对象表格。当然，表格在用户界面中很常见。Swing 开发小组将大量的精力投入到了表格控件上。表格本身比较复杂，但是它可能比其他 Swing 类更为成功，因为 JTable 构件隐藏了更多的复杂性。只需编写几行代码就能够产生具有完整功能的、行为丰富的表格。当然，还可以编写更多的代码，为具体应用定制显示外观和运行特性。

在本节中，我们将着重讲解怎样产生简单表格，用户怎样与它们交互，以及怎样进行一些最常见的调整操作。与其他一些复杂的 Swing 构件一样，我们不可能覆盖所有的细节。如果想获得详细信息，请查阅 David M. Geary 撰写的 *Graphic Java* (第 3 版, Prentice Hall, 1999) 或 Kim Topley 撰写的 *Core Swing* (Prentice Hall, 1999)。

### 11.1.1 一个简单表格

JTable 并不存储它自己的数据，而是从一个表格模型中获取数据。JTable 类有一个构造器，能够将二维对象数组包装进一个默认的模型。这也正是我们第一个示例程序要用到的策略。在本章的后续部分，我们将转向介绍表格模型。

图 11-1 展示了一个典型的表格，用于描述太阳系各个行星的属性。(如果一个行星主要由氢气和氦气组成，那么它就是气态行星。对于“Color”项，你不必太当真，我们之所以将它添加为一列是因为在后面的

Planet	Radius	Moons	Gaseous	Color
Mercury	2440.0	0	false	java.awt.C...
Venus	6052.0	0	false	java.awt.C...
Earth	6378.0	1	false	java.awt.C...
Mars	3397.0	2	false	java.awt.C...
Jupiter	71492.0	16	true	java.awt.C...
Saturn	60268.0	18	true	java.awt.C...
Uranus	25559.0	17	true	java.awt.C...
Neptune	24766.0	13	true	java.awt.C...

图 11-1 简单表格

示例代码中它会很有用。)

正如在程序清单 11-1 中看到的那样，表格中的数据是以 Object 值的二维数组的形式存储的：

```
Object[][] cells =
{
    { "Mercury", 2440.0, 0, false, Color.YELLOW },
    { "Venus", 6052.0, 0, false, Color.YELLOW },
    ...
}
```

**注释：**这里，我们充分利用了自动装箱机制。第二列、第三列、第四列中的项会自动转换成类型为 Double、Integer 和 Boolean 的对象。

该表格直接调用每个对象上的 `toString` 方法来显示它们，这也正是颜色显示为 `java.awt.Color[r = ..., g = ..., b = ...]` 的原因所在。

可以用一个单独的字符串数组来提供列名：

```
String[] columnNames = { "Planet", "Radius", "Moons", "Gaseous", "Color" };
```

接着，就可以从单元格和列名数组中构建一个表格：

```
var table = new JTable(cells, columnNames);
```

最后，通过将表格包装到一个 `JScrollPane` 中这种常用方法来添加滚动条：

```
var pane = new JScrollPane(table);
```

在滚动表格时，列表头并不会滑到视图的外面。

接着，单击列表头的某一列，并且向左或向右拖拉。看看整个列是怎样移开的（参见图 11-2），可以将它放到别的位置上。这种列的重新排列只是视图上的重新排列，对数据模型没有任何影响。

如果要调整列的尺寸大小，只需将鼠标移到两列之间，直到鼠标的形状变成箭头为止，然后将列的边界拖移到你期望的位置上（参见图 11-3）。

Planet	Radius	Moons	Gaseous	Color
Mercury	2440.0	0	false	java.awt.Color[r=255,g=255,b=0]
Venus	6052.0	0	false	java.awt.Color[r=255,g=255,b=0]
Earth	6378.0	1	false	java.awt.Color[r=255,g=255,b=0]
Mars	3397.0	2	false	java.awt.Color[r=255,g=255,b=0]
Jupiter	71492.0	16	true	java.awt.Color[r=255,g=255,b=0]
Saturn	60268.0	18	true	java.awt.Color[r=255,g=255,b=0]
Uranus	25559.0	17	true	java.awt.Color[r=255,g=255,b=0]
Neptune	24766.0	12	true	java.awt.Color[r=255,g=255,b=0]

图 11-2 移动表格中的一列

Planet	Radius	Moons	Gaseous	Color
Mercury	2440.0	0	false	java.awt.Color[r=255,g=255,b=0]
Venus	6052.0	0	false	java.awt.Color[r=255,g=255,b=0]
Earth	6378.0	1	false	java.awt.Color[r=255,g=255,b=0]
Mars	3397.0	2	false	java.awt.Color[r=255,g=255,b=0]
Jupiter	71492.0	16	true	java.awt.Color[r=255,g=255,b=0]
Saturn	60268.0	18	true	java.awt.Color[r=255,g=255,b=0]
Uranus	25559.0	17	true	java.awt.Color[r=255,g=255,b=0]
Neptune	24766.0	12	true	java.awt.Color[r=255,g=255,b=0]

图 11-3 调整列的尺寸大小

用户可以通过点击行中任何一个地方来选中一行，而选中的行会高亮显示。通过单击一个单元格并键入数据，用户还可以编辑表格中的各个项。不过，在这个代码示例中，这些编辑并没有改变底层的数据。在程序中，应该要么使这些单元格不可编辑，要么处理单元格编辑事件并更新模型。我们将会在本节的后面对这些问题进行讨论。

最后，点击列的头，行就会自动排序。如果再次点击，排序顺序就会反过来。这个行为

是通过下面的调用激活的：

```
table.setAutoCreateRowSorter(true);
```

可以使用下面的调用对表格进行打印：

```
table.print();
```

**！ 警告：**如果没有将表格包装在滚动面板中，那么就需要显式地添加表头：

```
add(table.getTableHeader(), BorderLayout.NORTH);
```

### 程序清单 11-1 table/TableTest.java

```

1 package table;
2
3 import java.awt.*;
4 import java.awt.print.*;
5
6 import javax.swing.*;
7
8 /**
9  * This program demonstrates how to show a simple table.
10 * @version 1.14 2018-05-01
11 * @author Cay Horstmann
12 */
13 public class TableTest
14 {
15     public static void main(String[] args)
16     {
17         EventQueue.invokeLater(() ->
18         {
19             var frame = new PlanetTableFrame();
20             frame.setTitle("TableTest");
21             frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
22             frame.setVisible(true);
23         });
24     }
25 }
26
27 /**
28  * This frame contains a table of planet data.
29 */
30 class PlanetTableFrame extends JFrame
31 {
32     private String[] columnNames = { "Planet", "Radius", "Moons", "Gaseous", "Color" };
33     private Object[][] cells =
34     {
35         { "Mercury", 2440.0, 0, false, Color.YELLOW },
36         { "Venus", 6052.0, 0, false, Color.YELLOW },
37         { "Earth", 6378.0, 1, false, Color.BLUE },
38         { "Mars", 3397.0, 2, false, Color.RED },
39         { "Jupiter", 71492.0, 16, true, Color.ORANGE },
40         { "Saturn", 60268.0, 18, true, Color.ORANGE },
41         { "Uranus", 25559.0, 17, true, Color.BLUE },
42     };
43 }
```

```

42     { "Neptune", 24766.0, 8, true, Color.BLUE },
43     { "Pluto", 1137.0, 1, false, Color.BLACK }
44 };
45
46 public PlanetTableFrame()
47 {
48     var table = new JTable(cells, columnNames);
49     table.setAutoCreateRowSorter(true);
50     add(new JScrollPane(table), BorderLayout.CENTER);
51     var printButton = new JButton("Print");
52     printButton.addActionListener(event ->
53     {
54         try { table.print(); }
55         catch (SecurityException | PrinterException ex) { ex.printStackTrace(); }
56     });
57     var buttonPanel = new JPanel();
58     buttonPanel.add(printButton);
59     add(buttonPanel, BorderLayout.SOUTH);
60     pack();
61 }
62 }
```

### API javax.swing.JTable 1.2

- `JTable(Object[][] entries, Object[] columnNames)`

用默认的表格模型构建一个表格。

- `void print() 5.0`

显示打印对话框，并打印该表格。

- `boolean getAutoCreateRowSorter() 6`

- `void setAutoCreateRowSorter(boolean newValue) 6`

获取或设置 `autoCreateRowSorter` 属性，默认值为 `false`。如果进行了设置，只要模型发生变化，就会自动设置一个默认的行排序器。

- `boolean getFillsViewportHeight() 6`

- `void setFillsViewportHeight(boolean newValue) 6`

获取或设置 `fillsViewportHeight` 属性，默认值为 `false`。如果进行了设置，该表格就总是会填充其外围的视图。

## 11.1.2 表格模型

在上一个示例中，表格数据是存储在一个二维数组中的。不过，通常不应该在自己的代码中使用这种策略。如果你发现自己在将数据装入一个数组中，然后作为一个表格显示出来，那么就应该考虑实现自己的表格模型了。

表格模型实现起来特别简单，因为可以充分利用 `AbstractTableModel` 类，它实现了大部分必需的方法。你仅仅需要提供下面三个方法便可：

```
public int getRowCount();
```

```
public int getColumnCount();
public Object getValueAt(int row, int column);
```

实现 `getValueAt` 方法有多种途径。例如，如果想显示包含数据库查询结果的 `RowSet` 的内容，只需提供下面的方法：

```
public Object getValueAt(int r, int c)
{
    try
    {
        rowSet.absolute(r + 1);
        return rowSet.getObject(c + 1);
    }
    catch (SQLException e)
    {
        e.printStackTrace();
        return null;
    }
}
```

我们的示例程序相当简单。我们构建了一个只是用来显示某些计算结果的表格，这些计算结果也就是在不同利率条件下的投资增长额（参见图 11-4）。

`getValueAt` 方法计算出正确值，并将其格式化：

```
public Object getValueAt(int r, int c)
{
    double rate = (c + minRate) / 100.0;
    int nperiods = r;
    double futureBalance = INITIAL_BALANCE * Math.pow(1 + rate, nperiods);
    return String.format("%.2f", futureBalance);
}
```

`getRowCount` 和 `getColumnCount` 方法只是返回行数和列数。

```
public int getRowCount() { return years; }
public int getColumnCount() { return maxRate - minRate + 1; }
```

如果不提供列名，那么 `AbstractTableModel` 的 `getColumnName` 方法会将列命名为 A、B、C 等。如果要改变列名，请覆盖 `getColumnName` 方法。通常需要覆盖默认的行为。在这个示例中，我们只是将每列用利率标识了出来。

```
public String getColumnName(int c) { return (c + minRate) + "%"; }
```

程序清单 11-2 中显示了完整的源代码。

	5%	6%	7%	8%	9%	10%
100000.00	100000.00	100000.00	100000.00	100000.00	100000.00	100000.00
105000.00	106000.00	107000.00	108000.00	109000.00	110000.00	110000.00
110250.00	112360.00	114490.00	116640.00	118810.00	121000.00	121000.00
115762.50	119101.60	122504.30	125971.20	129502.90	133100.00	133100.00
121550.63	126247.70	131079.50	136048.90	141158.16	146410.00	146410.00
127628.16	133822.56	140255.17	146932.81	153862.40	161051.00	161051.00
134009.56	141851.91	150073.04	158687.43	167710.01	177156.10	177156.10
140710.04	150363.03	160578.15	171382.43	182803.91	194871.71	194871.71
147745.54	159384.81	171818.62	185093.02	199256.26	214358.88	214358.88
155132.82	168947.90	183845.92	199900.46	217189.33	235794.77	235794.77
162889.46	179084.77	196715.14	215892.50	236735.37	259374.25	259374.25
171033.94	189829.86	210485.20	233163.90	258042.64	285311.67	285311.67
179585.63	201219.65	225219.16	251817.01	281266.48	313842.84	313842.84
188564.91	213292.83	240984.50	271962.37	306580.46	345227.12	345227.12
197993.16	226090.40	257853.42	293719.36	334172.70	379749.83	379749.83
207892.82	239655.82	275903.15	317216.91	364248.25	417724.82	417724.82

图 11-4 投资增长额

## 程序清单 11-2 tableModel/InvestmentTable.java

```
1 package tableModel;
2
3 import java.awt.*;
```

```
4
5 import javax.swing.*;
6 import javax.swing.table.*;
7
8 /**
9  * This program shows how to build a table from a table model.
10 * @version 1.04 2018-05-01
11 * @author Cay Horstmann
12 */
13 public class InvestmentTable
14 {
15     public static void main(String[] args)
16     {
17         EventQueue.invokeLater(() ->
18         {
19             var frame = new InvestmentTableFrame();
20             frame.setTitle("InvestmentTable");
21             frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
22             frame.setVisible(true);
23         });
24     }
25 }
26
27 /**
28 * This frame contains the investment table.
29 */
30 class InvestmentTableFrame extends JFrame
31 {
32     public InvestmentTableFrame()
33     {
34         var model = new InvestmentTableModel(30, 5, 10);
35         var table = new JTable(model);
36         add(new JScrollPane(table));
37         pack();
38     }
39 }
40
41 /**
42 * This table model computes the cell entries each time they are requested. The table contents
43 * shows the growth of an investment for a number of years under different interest rates.
44 */
45 class InvestmentTableModel extends AbstractTableModel
46 {
47     private static double INITIAL_BALANCE = 100000.0;
48
49     private int years;
50     private int minRate;
51     private int maxRate;
52
53 /**
54 * Constructs an investment table model.
55 * @param y the number of years
56 * @param r1 the lowest interest rate to tabulate
57 * @param r2 the highest interest rate to tabulate
58 */
```

```

59     public InvestmentTableModel(int y, int r1, int r2)
60     {
61         years = y;
62         minRate = r1;
63         maxRate = r2;
64     }
65
66     public int getRowCount()
67     {
68         return years;
69     }
70
71     public int getColumnCount()
72     {
73         return maxRate - minRate + 1;
74     }
75
76     public Object getValueAt(int r, int c)
77     {
78         double rate = (c + minRate) / 100.0;
79         int nperiods = r;
80         double futureBalance = INITIAL_BALANCE * Math.pow(1 + rate, nperiods);
81         return String.format("%.2f", futureBalance);
82     }
83
84     public String getColumnName(int c)
85     {
86         return (c + minRate) + "%";
87     }
88 }
```

#### API javax.swing.table.TableModel 1.2

- `int getRowCount()`  
获取表模型中的行和列的数量。
- `int getColumnCount()`  
获取在给定的行和列所确定的位置处的值。
- `Object getValueAt(int row, int column)`  
设置在给定的行和列所确定的位置处的值。
- `void setValueAt(Object newValue, int row, int column)`  
如果在给定的行和列所确定的位置处的值是可编辑的，则返回 `true`。
- `String getColumnName(int column)`  
获取列的名字。

### 11.1.3 对行和列的操作

在本小节中，你会看到怎样操作一个表格中的行和列。在你阅读本材料的整个过程中，

要牢记 Swing 中的表格是相当不对称的，也就是你可以实施的行操作和列操作会有所不同。表格构件已经被优化过，以便能够显示具有相同结构的行信息，例如一次数据库查询的结果，而不是任意的二维对象表格。你将会看到，这种不对称性贯穿于本小节。

### 11.1.3.1 各种列类

在下一个示例中，我们将再次展示行星数据，不过这次我们会给出更多的有关表格列类型的信息。这是通过在表格模型中定义下面这个方法来实现的：

```
Class<?> getColumnClass(int columnIndex)
```

这个方法可以返回一个描述列类型的类。

JTable 类会为该类选取合适的绘制器，表 11-1 显示了默认的绘制动作。

表 11-1 默认的绘制动作

类型	绘制结果
Boolean	复选框
Icon	图像
Object	字符串

可以在图 11-5 中看到复选框和图像。(感谢 Jim Evins 提供了这些行星图像。)

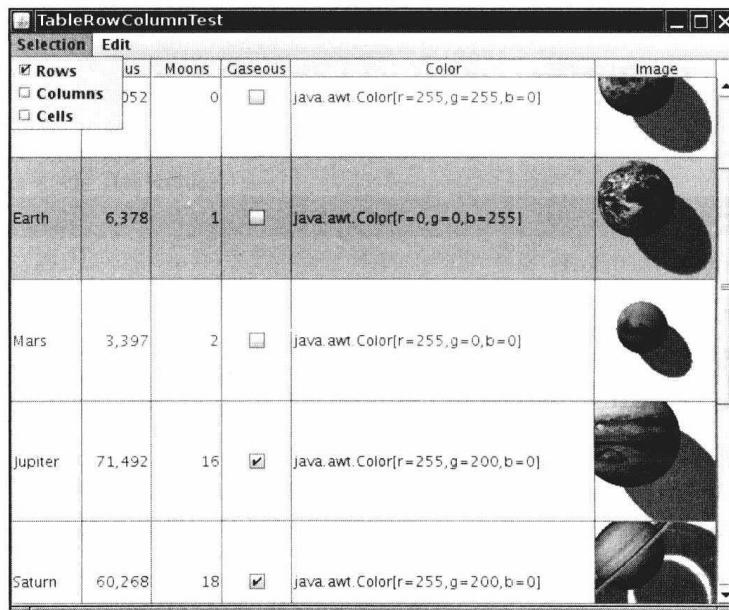


图 11-5 具有单元格绘制器的表格

要绘制其他类型，需要安装定制的绘制器，请参见 11.1.4 节。

### 11.1.3.2 访问表格列

JTable 类将有关表格列的信息存放在类型为 TableColumn 的对象中，由一个 TableColumnModel 对象负责管理这些列。(图 11-6 展示了最重要的表格类之间的关系。) 如果不想动态地

插入或删除，那么最好不要过多地使用表格列模型。列模型最常见的用法是直接获取一个 TableColumn 对象：

```
int columnIndex = . . .;
TableColumn column = table.getColumnModel().getColumn(columnIndex);
```

### 11.1.3.3 改变列的大小

TableColumn 类可以控制更改列的大小的行为。使用下面这些方法，可以设置首选的、最小的以及最大的宽度：

```
void setPreferredWidth(int width)
void setMinWidth(int width)
void setMaxWidth(int width)
```

这些信息将提供给表格构件，以便对列进行布局。

使用方法

```
void setResizable(boolean resizable)
```

可以控制是否允许用户改变列的大小。

可以使用下面这个方法在程序中改变列的大小：

```
void setWidth(int width)
```

调整一个列的大小时，默认情况下表格的总体大小会保持不变。当然，更改过大小的列的宽度的增加值或减小值会分摊到其他列上。默认方式是更改那些在被改变了大小的列右边的所有列的大小。这是一种很好的默认方式，因为这样使得用户可以通过将所有列从左到右移动，将它们调整为自己所期望的宽度。

使用下面这个方法，可以设置表 11-2 中列出的 JTable 类的其他行为：

```
void setAutoResizeMode(int mode)
```

表 11-2 变更列大小的模式

模式	行为
AUTO_RESIZE_OFF	不更改其他列的大小，而是更改整个表格的宽度
AUTO_RESIZE_NEXT_COLUMN	只更改下一列的大小
AUTO_RESIZE_SUBSEQUENT_COLUMNS	均匀地更改后续列的大小，这是默认的行为
AUTO_RESIZE_LAST_COLUMN	只更改最后一列的大小
AUTO_RESIZE_ALL_COLUMNS	更改表格中的所有列的大小，这并不是一种很明智的选择，因为这阻碍了用户只对数列而不是整个表进行调整以达到自己期望大小的行为

### 11.1.3.4 改变行的大小

行的高度是直接由 JTable 类管理的。如果单元格比默认值高，那么可以像下面这样设置行的高度：

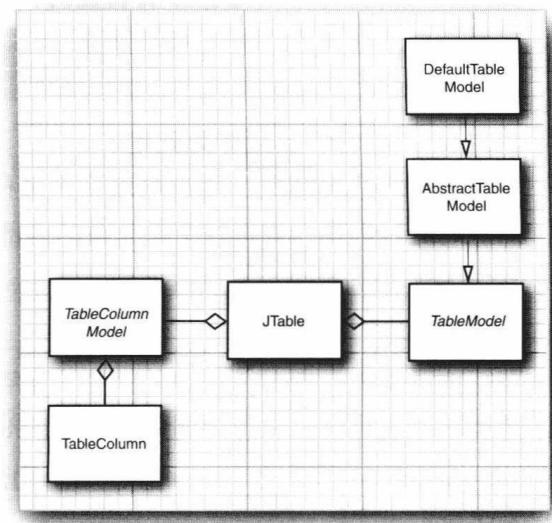


图 11-6 表格类之间的关系图

```
table.setRowHeight(height);
```

默认情况下，表格中的所有行都具有相同的高度，可以用下面的调用来为每一行单独设置高度：

```
table.setRowHeight(row, height);
```

实际的行高度等于用这些方法设置的行高度减去行边距，其中行边距的默认值是 1 个像素，但是可以通过下面的调用来修改它：

```
table.setRowMargin(margin);
```

### 11.1.3.5 选择行、列和单元格

利用不同的选择模式，用户可以分别选择表格中的行、列或者单个的单元格。默认情况下，使能的是行选择，点击一个单元格的内部就可以选择整行（参见图 11-5）。调用

```
table.setRowSelectionAllowed(false);
```

可以禁用行选择。

当行选择功能可用时，可以控制用户是否可以选择单一行、连续几行或者任意几行。此时，需要获取选择模式，然后调用它的 `setSelectionMode` 方法：

```
table.getSelectionModel().setSelectionMode(mode);
```

在这里，`mode` 是下面三个值的其中一个：

```
ListSelectionModel.SINGLE_SELECTION  
ListSelectionModel.SINGLE_INTERVAL_SELECTION  
ListSelectionModel.MULTIPLE_INTERVAL_SELECTION
```

默认情况下，列选择是禁用的。不过可以调用下面这个方法启用列选择：

```
table.setColumnSelectionAllowed(true);
```

同时启用行选择和列选择等价于启用单元格选择，这样用户就可以选择一定范围内的单元格（参见图 11-7）。也可以使用下面的调用完成这项设置：

```
table.setCellSelectionEnabled(true);
```

可以运行程序清单 11-3 中的程序，观察一下单元格选择的运行情况。启用 Selection 菜单中的行、列或单元格选项，然后观察选择行为是如何改变的。

可以通过调用 `getSelectedRows` 方法和 `getSelectedColumns` 方法来查看选中了哪些行及哪些列。这两个方法都返回一个由被选定项的索引构成的 `int[]` 数组。注意，这些索引值是表格视图中的索引值，而不是底层表格模型中的索引值。尝试着选择一些行和列，然后将列拖拽到不同的位置，并通过点击列头来对这些行进行排序。使用 Print Selection 菜单项来查看它会报告哪些行和列被选中。

如果要将表格索引值转译为表格模型索引值，可以使用 `JTable` 的 `ConvertRowIndexToModel` 和 `convertColumnIndexToModel` 方法。

### 11.1.3.6 对行排序

正如在第一个表格示例中看到的那样，向 `JTable` 中添加行排序机制是很容易的，只需调

用 `setAutoCreateRowSorter` 方法。但是，要对排序行为进行细粒度的控制，就必须向 `JTable` 中安装一个 `TableRowSorter<M>` 对象，并对其进行定制化。类型参数 `M` 表示表格模型，它必须是 `TableModel` 接口的子类型。

```
var sorter = new TableRowSorter<TableModel>(model);
table.setRowSorter(sorter);
```

Planet	Radius	Moons	Gaseous	Color	Image
Mars	3,397	2	<input type="checkbox"/>	java.awt.Color[r=255,g=0,b=0]	
Jupiter	71,492	16	<input checked="" type="checkbox"/>	java.awt.Color[r=255,g=200,b=0]	
Saturn	60,268	18	<input checked="" type="checkbox"/>	java.awt.Color[r=255,g=200,b=0]	
Uranus	25,559	17	<input checked="" type="checkbox"/>	java.awt.Color[r=0,g=0,b=255]	

图 11-7 选择一个单元格范围

某些列是不可排序的，例如，在我们的行星数据中的图像列，可以通过下面的调用来关闭排序机制：

```
sorter.setSortable(IMAGE_COLUMN, false);
```

可以对每个列都安装一个定制的比较器。在我们的示例中，将对 `Color` 列中的颜色进行排序，因为我们相对于红色来说，更喜欢蓝色和绿色。当点击 `Color` 列时，将会看到蓝色行星出现在表格底部，这是通过下面的调用完成的：

```
sorter.setComparator(COLOR_COLUMN, new Comparator<Color>()
{
    public int compare(Color c1, Color c2)
    {
        int d = c1.getBlue() - c2.getBlue();
        if (d != 0) return d;
        d = c1.getGreen() - c2.getGreen();
        if (d != 0) return d;
        return c1.getRed() - c2.getRed();
    }
});
```

如果不指定列的比较器，那么排列顺序就是按照下面的原则确定的：

1. 如果列所属的类是 String，就使用 Collator.getInstance() 方法返回的默认比较器。它按照适用于当前 locale 的方式对字符串排序。(参见第 7 章以了解 locale 和比较器的更多信息)。
2. 如果列所属的类型实现了 Comparable，则使用它的 compareTo 方法。
3. 如果已经为排序器设置过 TableStringConverter，就用默认比较器对转换器的 toString 方法返回的字符串进行排序。如果要使用该方法，可以像下面这样定义转换器：

```
sorter.setStringConverter(new TableStringConverter()
{
    public String toString(TableModel model, int row, int column)
    {
        Object value = model.getValueAt(row, column);
        convert value to a string and return it
    }
});
```

4. 否则，在单元格的值上调用 toString 方法，然后用默认比较器对它们进行比较。

#### 11.1.3.7 过滤行

除了可以对行排序之外，TableRowSorter 还可以有选择性地隐藏行，这种处理称为过滤 (filtering)。要想激活过滤机制，需要设置 RowFilter。例如，要包含所有至少有一个卫星的行星，可以调用：

```
sorter.setRowFilter(RowFilter.numberFilter(ComparisonType.NOT_EQUAL, 0, MOONS_COLUMN));
```

这里我们使用了预定义的过滤器，即数字过滤器。要构建数字过滤器，需要提供：

- 比较类型 (EQUAL、NOT\_EQUAL、AFTER 和 BEFORE 之一)。
- Number 的某个子类的一个对象 (例如 Integer 和 Double)，只有与给定的 Number 对象属于相同的类的对象才在考虑的范围内。
- 0 或多列的索引值，如果不提供任何索引值，那么所有的列都被搜索。

静态的 RowFilter.dateFilter 方法以相同的方式构建了日期过滤器，这里需要提供 Date 对象而不是 Number 对象。

最后，静态的 RowFilter.regexFilter 方法构建的过滤器可以查找匹配某个正则表达式的字符串。例如：

```
sorter.setRowFilter(RowFilter.regexFilter(".*[^\s]$", PLANET_COLUMN));
```

将只显示那些名字以“s”结尾的行星 (参见第 2 章以了解有关正则表达式的更多信息)。

还可以用 andFilter、orFilter 和 notFilter 方法来组合过滤器，例如，要过滤掉名字不是以“s”结尾，并且至少有一颗卫星的行星，可以使用下面的过滤器组合：

```
sorter.setRowFilter(RowFilter.andFilter(List.of(
    RowFilter.regexFilter(".*[^\s]$", PLANET_COLUMN),
    RowFilter.numberFilter(ComparisonType.NOT_EQUAL, 0, MOONS_COLUMN))));
```

要实现自己的过滤器，需要提供 RowFilter 的一个子类，并实现 include 方法来表示哪些行应该显示。这很容易实现，但是 RowFilter 类卓越的普适性令它有点可怕。

RowFilter<M, I> 类有两个类型参数：模型的类型和行标识符的类型。在处理表格时，模

型总是 `TableModel` 的某个子类型，而标识符类型总是 `Integer`。（在将来的某个时刻，其他构件可能也会支持行过滤机制。例如，要过滤 `JTree` 中的行，就可能可以使用 `RowFilter<TreeModel, TreePath>` 了。）

行过滤器必须实现下面的方法：

```
public boolean include(RowFilter.Entry<? extends M, ? extends I> entry)
```

`RowFilter.Entry` 类提供了获取模型、行标识符和给定索引处的值等内容的方法，因此，按照行标识符和行的内容都可以进行过滤。

例如，下面的过滤器将隔行显示：

```
var filter = new RowFilter<TableModel, Integer>()
{
    public boolean include(Entry<? extends TableModel, ? extends Integer> entry)
    {
        return entry.getIdentifier() % 2 == 0;
    }
};
```

如果想要只包含那些具有偶数个卫星的行星，可以将上面的测试条件替换为下面的内容：

```
((Integer) entry.getValue(MOONS_COLUMN)) % 2 == 0
```

在我们的示例程序中，允许用户隐藏任意多行，我们在一个 `set` 中存储了所有隐藏的行的索引。而其中的行过滤器将包含那些索引不在这个 `set` 中的所有行。

过滤机制并不是为那些过滤标准在不时地发生变化的过滤器而设计的。因此，在我们的示例程序中，只要隐藏行的 `set` 发生了变化，我们就会调用下面的语句：

```
sorter.setRowFilter(filter);
```

过滤器一旦被设置，就会立即得到应用。

### 11.1.3.8 隐藏和显示列

正如在前一节中看到的，可以根据内容或标识符来过滤表格行，而隐藏表格列使用的是完全不同的机制。

`JTable` 类的 `removeColumn` 方法可以将一列从表格视图中移除。该列的数据实际上并没有从模型中移除，它们只是在视图中被隐藏了起来。`removeColumn` 方法接受一个  `TableColumn` 参数，如果你有的是一个列号（比如来自 `getSelectedColumns` 的调用结果），那就需要向表格模型请求实际的列对象：

```
TableColumnModel columnModel = table.getColumnModel();
TableColumn column = columnModel.getColumn(i);
table.removeColumn(column);
```

如果你记得住该列，那么将来就可以再把它添加回去：

```
table.addColumn(column);
```

该方法将该列添加到表格的最后面。如果想让它出现在表格中的其他任何地方，那么可以调用 `moveColumn` 方法。

通过添加一个新的 `TableColumn` 对象，还可以添加一个对应于表格模型中的一个列索引的新列：

```
table.addColumn(new TableColumn(modelColumnIndex));
```

可以让多个表格列展示模型中的同一列。

程序清单 11-3 展示了如何选择和过滤行与列。

### 程序清单 11-3 tableRowColumn/planetTableFrame.java

```

1 package tableRowColumn;
2
3 import java.awt.*;
4 import java.util.*;
5
6 import javax.swing.*;
7 import javax.swing.table.*;
8
9 /**
10  * This frame contains a table of planet data.
11  */
12 public class PlanetTableFrame extends JFrame
13 {
14     private static final int DEFAULT_WIDTH = 600;
15     private static final int DEFAULT_HEIGHT = 500;
16
17     public static final int COLOR_COLUMN = 4;
18     public static final int IMAGE_COLUMN = 5;
19
20     private JTable table;
21     private HashSet<Integer> removedRowIndices;
22     private ArrayList<TableColumn> removedColumns;
23     private JCheckBoxMenuItem rowsItem;
24     private JCheckBoxMenuItem columnsItem;
25     private JCheckBoxMenuItem cellsItem;
26
27     private String[] columnNames = { "Planet", "Radius", "Moons", "Gaseous", "Color", "Image" };
28
29     private Object[][][] cells =
30     {
31         { "Mercury", 2440.0, 0, false, Color.YELLOW,
32             new ImageIcon(getClass().getResource("Mercury.gif")) },
33         { "Venus", 6052.0, 0, false, Color.YELLOW,
34             new ImageIcon(getClass().getResource("Venus.gif")) },
35         { "Earth", 6378.0, 1, false, Color.BLUE,
36             new ImageIcon(getClass().getResource("Earth.gif")) },
37         { "Mars", 3397.0, 2, false, Color.RED,
38             new ImageIcon(getClass().getResource("Mars.gif")) },
39         { "Jupiter", 71492.0, 16, true, Color.ORANGE,
40             new ImageIcon(getClass().getResource("Jupiter.gif")) },
41         { "Saturn", 60268.0, 18, true, Color.ORANGE,
42             new ImageIcon(getClass().getResource("Saturn.gif")) },
43         { "Uranus", 25559.0, 17, true, Color.BLUE,
44             new ImageIcon(getClass().getResource("Uranus.gif")) },

```

```
45     { "Neptune", 24766.0, 8, true, Color.BLUE,
46         new ImageIcon(getClass().getResource("Neptune.gif")) },
47     { "Pluto", 1137.0, 1, false, Color.BLACK,
48         new ImageIcon(getClass().getResource("Pluto.gif")) }
49 };
50
51 public PlanetTableFrame()
52 {
53     setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
54
55     var model = new DefaultTableModel(cells, columnNames)
56     {
57         public Class<?> getColumnClass(int c)
58         {
59             return cells[0][c].getClass();
60         }
61     };
62
63     table = new JTable(model);
64
65     table.setRowHeight(100);
66     table.getColumnModel().getColumn(COLOR_COLUMN).setMinWidth(250);
67     table.getColumnModel().getColumn(IMAGE_COLUMN).setMinWidth(100);
68
69     var sorter = new TableRowSorter<TableModel>(model);
70     table.setRowSorter(sorter);
71     sorter.setComparator(COLOR_COLUMN, Comparator.comparing(Color::getBlue)
72         .thenComparing(Color::getGreen).thenComparing(Color::getRed));
73     sorter.setSortable(IMAGE_COLUMN, false);
74     add(new JScrollPane(table), BorderLayout.CENTER);
75
76     removedRowIndices = new HashSet<>();
77     removedColumns = new ArrayList<>();
78
79     var filter = new RowFilter<TableModel, Integer>()
80     {
81         public boolean include(Entry<? extends TableModel, ? extends Integer> entry)
82         {
83             return !removedRowIndices.contains(entry.getIdentifier());
84         }
85     };
86
87     // create menu
88
89     var menuBar = new JMenuBar();
90     setJMenuBar(menuBar);
91
92     var selectionMenu = new JMenu("Selection");
93     menuBar.add(selectionMenu);
94
95     rowsItem = new JCheckBoxMenuItem("Rows");
96     columnsItem = new JCheckBoxMenuItem("Columns");
97     cellsItem = new JCheckBoxMenuItem("Cells");
98
99     rowsItem.setSelected(table.getRowSelectionAllowed());
```

```

100    columnsItem.setSelected(table.getColumnNameSelectionAllowed());
101    cellsItem.setSelected(table.getCellSelectionEnabled());
102
103    rowsItem.addActionListener(event ->
104    {
105        table.clearSelection();
106        table.setRowSelectionAllowed(rowsItem.isSelected());
107        updateCheckboxMenuItems();
108    });
109    selectionMenu.add(rowsItem);
110
111    columnsItem.addActionListener(event ->
112    {
113        table.clearSelection();
114        table.setColumnSelectionAllowed(columnsItem.isSelected());
115        updateCheckboxMenuItems();
116    });
117    selectionMenu.add(columnsItem);
118
119    cellsItem.addActionListener(event ->
120    {
121        table.clearSelection();
122        table.setCellSelectionEnabled(cellsItem.isSelected());
123        updateCheckboxMenuItems();
124    });
125    selectionMenu.add(cellsItem);
126
127    var tableMenu = new JMenu("Edit");
128    menuBar.add(tableMenu);
129
130    var hideColumnsItem = new JMenuItem("Hide Columns");
131    hideColumnsItem.addActionListener(event ->
132    {
133        int[] selected = table.getSelectedColumns();
134        TableColumnModel columnModel = table.getColumnModel();
135
136        // remove columns from view, starting at the last
137        // index so that column numbers aren't affected
138
139        for (int i = selected.length - 1; i >= 0; i--)
140        {
141            TableColumn column = columnModel.getColumn(selected[i]);
142            table.removeColumn(column);
143
144            // store removed columns for "show columns" command
145
146            removedColumns.add(column);
147        }
148    });
149    tableMenu.add(hideColumnsItem);
150
151    var showColumnsItem = new JMenuItem("Show Columns");
152    showColumnsItem.addActionListener(event ->
153    {

```

```

154         // restore all removed columns
155         for (TableColumn tc : removedColumns)
156             table.addColumn(tc);
157         removedColumns.clear();
158     });
159     tableMenu.add(showColumnsItem);
160
161     var hideRowsItem = new JMenuItem("Hide Rows");
162     hideRowsItem.addActionListener(event ->
163     {
164         int[] selected = table.getSelectedRows();
165         for (int i : selected)
166             removedRowIndices.add(table.convertRowIndexToModel(i));
167         sorter.setRowFilter(filter);
168     });
169     tableMenu.add(hideRowsItem);
170
171     var showRowsItem = new JMenuItem("Show Rows");
172     showRowsItem.addActionListener(event ->
173     {
174         removedRowIndices.clear();
175         sorter.setRowFilter(filter);
176     });
177     tableMenu.add(showRowsItem);
178
179     var printSelectionItem = new JMenuItem("Print Selection");
180     printSelectionItem.addActionListener(event ->
181     {
182         int[] selected = table.getSelectedRows();
183         System.out.println("Selected rows: " + Arrays.toString(selected));
184         selected = table.getSelectedColumns();
185         System.out.println("Selected columns: " + Arrays.toString(selected));
186     });
187     tableMenu.add(printSelectionItem);
188 }
189
190 private void updateCheckboxMenuItems()
191 {
192     rowsItem.setSelected(table.getRowSelectionAllowed());
193     columnsItem.setSelected(table.getColumnSelectionAllowed());
194     cellsItem.setSelected(table.getCellSelectionEnabled());
195 }
196 }
```

**API** ***javax.swing.table.TableModel*** 1.2

- Class getColumnClass(int columnIndex)

获取该列中的值的类。该信息用于排序或绘制。

**API** ***javax.swing.JTable*** 1.2

- TableColumnModel getColumnModel()

获取描述表格列布局安排的“列模式”。

- `void setAutoResizeMode(int mode)`

设置自动更改表格列大小的模式。

参数: mode `AUTO_RESIZE_OFF`、`AUTO_RESIZE_NEXT_COLUMN`、`AUTO_RESIZE_SUBSEQUENT_COLUMNS`、`AUTO_RESIZE_LAST_COLUMN` 以及 `AUTO_RESIZE_ALL_COLUMNS` 其中之一。

- `int getRowHeight()`

- `void setRowMargin(int margin)`

获取和设置相邻行中单元格之间的间隔大小。

- `int getRowHeight()`

- `void setRowHeight(int height)`

获取和设置表格中所有行的默认高度。

- `int getRowHeight(int row)`

- `void setRowHeight(int row, int height)`

获取和设置表格中给定行的高度。

- `ListSelectionModel getSelectionModel()`

返回列表的选择模式。你需要该模式以便在行、列以及单元格之间进行选择。

- `boolean getRowSelectionAllowed()`

- `void setRowSelectionAllowed(boolean b)`

获取和设置 `rowSelectionAllowed` 属性。如果为 `true`, 那么当用户点击单元格的时候, 可以选定行。

- `boolean getColumnSelectionAllowed()`

- `void setColumnSelectionAllowed(boolean b)`

获取和设置 `columnSelectionAllowed` 属性。如果为 `true`, 那么当用户点击单元格的时候, 可以选定列。

- `boolean getCellSelectionEnabled()`

如果既允许选定行又允许选定列, 则返回 `true`。

- `void setCellSelectionEnabled(boolean b)`

同时将 `rowSelectionAllowed` 和 `columnSelectionAllowed` 设置为 `b`。

- `voidaddColumn(TableColumn column)`

向表格视图中添加一列作为最后一列。

- `void moveColumn(int from, int to)`

移动表格 `from` 索引位置中的列, 使它的索引变成 `to`。该操作仅仅影响到视图。

- `void removeColumn(TableColumn column)`

将给定的列从视图中移除。

- `int convertRowIndexToModel(int index) 6`

- `int convertColumnIndexToModel(int index)`

返回具有给定索引的行或列的模型索引, 这个值与行被排序和过滤, 以及列被移动和

移除时的索引不同。

- void setRowSorter(TableRowSorter<? extends TableModel> sorter)  
设置行排序器。

#### API **javax.swing.table.TableColumnModel 1.2**

- TableColumn getColumn(int index)  
获取表格的列对象，用于描述给定索引的列。

#### API **javax.swing.table.TableColumn 1.2**

- TableColumn(int modelColumnIndex)  
构建一个表格列，用以显示给定索引位置上的模型列。
- void setPreferredWidth(int width)  
将表格的首选宽度设置为 width。
- void setMinWidth(int width)  
将表格的最小宽度设置为 width。
- void setMaxWidth(int width)  
将表格的最大宽度设置为 width。
- void setWidth(int width)  
设置该列的实际宽度为 width。
- void setResizable(boolean b)  
如果 b 为 true，那么该列可以更改大小。

#### API **javax.swing.ListSelectionModel 1.2**

- void setSelectionMode(int mode)  
参数： mode SINGLE\_SELECTION、SINGLE\_INTERVAL\_SELECTION 与 MULTIPLE\_INTERVAL\_SELECTION 之一。

#### API **javax.swing.DefaultRowSorter<M, I> 6**

- void setComparator(int column, Comparator<?> comparator)  
设置用于给定列的比较器。
- void setSortable(int column, boolean enabled)  
使对给定列的排序可用或禁用。
- void setRowFilter(RowFilter<? super M, ? super I> filter)  
设置行过滤器。

#### API **javax.swing.table.TableRowSorter<M extends TableModel> 6**

- void setStringConverter(TableStringConverter stringConverter)  
设置用于排序和过滤的字符串转换器。

#### API **javax.swing.table.TableStringConverter 6**

- abstract String toString(TableModel model, int row, int column)

将给定位置的模型值转换为字符串，你可以覆盖这个方法。

#### API javax.swing.RowFilter<M, I> 6

- boolean include(RowFilter.Entry<? extends M, ? extends I> entry)  
指定要保留的行，你可以覆盖这个方法。
- static <M,I> RowFilter<M,I> numberFilter(RowFilter.ComparisonType type, Number number, int... indices)  
返回一个过滤器，它包含的行是那些与给定的数字或日期进行给定比较后匹配的行。比较类型是 EQUAL、NOT\_EQUAL、AFTER 或 BEFORE 之一。如果给定了列模型索引，则只搜索这些列。否则，将搜索所有列。对于数字过滤器，单元格的值所属的类必须与给定数字的类匹配。
- static <M,I> RowFilter<M,I> regexFilter(String regex, int... indices)  
返回一个过滤器，它包含的行含有与给定的正则表达式匹配的字符串。如果给定了列模型索引，则只搜索这些列。否则，将搜索所有列。注意，RowFilter.Entry 的 getStringValue 方法返回的字符串是匹配的。
- static <M,I> RowFilter<M,I> andFilter(Iterable<? extends RowFilter<? super M, ? super I>> filters)  
返回一个过滤器，它包含的项是那些包含在所有的过滤器或至少包含在一个过滤器中的项。
- static <M,I> RowFilter<M,I> orFilter(Iterable<? extends RowFilter<? super M, ? super I>> filters)  
返回一个过滤器，它包含的项是那些不包含在给定过滤器中的项。

#### API javax.swing.RowFilter.Entry<M, I> 6

- I getIdentifier()  
返回这个行的标识符。
- M getModel()  
返回这个行的模型。
- Object getValue(int index)  
返回在这个行的给定索引处存储的值。
- int getValueCount()  
返回在这个行中存储的值的数量。
- String getStringValue()  
返回在这个行的给定索引处存储的值转换成的字符串。由 TableRowSorter 产生的项的 getStringValue 方法会调用排序器的字符串转换器。

### 11.1.4 单元格的绘制和编辑

正如在 11.1.3.2 节中看到的，列的类型确定了单元格应该如何绘制。Boolean 和 Icon 类型有默认的绘制器，它们将绘制复选框或图标，而对于其他所有类型，都需要安装定制的绘制器。

#### 11.1.4.1 绘制单元格

表格的单元格绘制器与你在前面看到的列表单元格绘制器类似。它们都实现了 TableCellRenderer 接口，并只有一个方法：

```
Component getTableCellRendererComponent(JTable table, Object value,
    boolean isSelected, boolean hasFocus, int row, int column)
```

该方法在表格需要绘制单元格的时候被调用。它会返回一个构件，接着该构件的 paint 方法会被调用，以填充单元格区域。

在图 11-8 中的表格包含类型为 Color 的单元格，绘制器直接返回一个面板，其背景颜色设置为存储在该单元格中的颜色对象，该颜色是作为 value 参数传递的。

```
class ColorTableCellRenderer extends JPanel implements TableCellRenderer
{
    public Component getTableCellRendererComponent(JTable table, Object value,
        boolean isSelected, boolean hasFocus, int row, int column)
    {
        setBackground((Color) value);
        if (hasFocus)
            setBorder(UIManager.getBorder("Table.focusCellHighlightBorder"));
        else
            setBorder(null);
        return this;
    }
}
```

Planet	Radius	Gaseous	Color	Image
Mars	3,397	2	■	
Jupiter	71,492	16	☒	
Saturn	60,268	18	☒	

图 11-8 具有单元格绘制器的表格

正如你看到的那样，当该单元格获得焦点的时候，绘制器会画出一个边框。(我们可以向

UIManager 寻求合适的边框。为了发现查找的关键所在，我们可以深入 DefaultTableCellRenderer 类的源码内部看个究竟。)

 提示：如果你的绘制器只是绘制一个文本字符串或者一个图标，那么可以继承 DefaultTableCellRenderer 这个类。该类会负责绘制焦点和选择状态。

你必须告诉表格要使用这个绘制器去绘制所有类型为 Color 的对象。JTable 类的 setDefaultRenderer 方法可以让你建立它们之间的这种联系。你需要提供一个 Class 对象和绘制器。

```
table.setDefaultRenderer(Color.class, new ColorTableCellRenderer());
```

现在这个绘制器就可以用于表格中具有给定类型的所有对象了。

如果想要基于其他标准选择绘制器，则需要从 JTable 类中扩展子类，并覆盖 getCellRenderer 方法。

#### 11.1.4.2 绘制表头

为了在表头中显示图标，需要设置表头值。

```
moonColumn.setHeaderValue(new ImageIcon("Moons.gif"));
```

然而，表头还未智能到可以为表头值选择一个合适的绘制器，因此，绘制器需要手工安装。例如，要在列头显示图像图标，可以调用：

```
moonColumn.setHeaderRenderer(table.getDefaultRenderer(ImageIcon.class));
```

#### 11.1.4.3 单元格编辑

为了使单元格可编辑，表格模型必须通过定义 isCellEditable 方法来指明哪些单元格是可编辑的。最常见的情况是，你可能想使某几列可编辑。在这个示例程序中，我们允许对表格中的四列进行编辑。

```
public boolean isCellEditable(int r, int c)
{
    return c == PLANET_COLUMN || c == MOONS_COLUMN || c == GASEOUS_COLUMN
        || c == COLOR_COLUMN;
}
```

 注释：AbstractTableModel 定义的 isCellEditable 方法总是返回 false。DefaultTableModel 覆盖了该方法以便总是返回 true。

运行一下程序清单 11-4 到程序清单 11-7 的程序就会注意到，可以点击 Gaseous 列中的复选框，并能选中或取消复选标记。如果点击 Moons 列中的某个单元格，就会出现一个组合框（参见图 11-9）。你很快就会看到怎样将这样一个组合框作为一个单元格编辑器安装到表格上。

最后，点击第一列中的某个单元格，该单元格就会获取焦点。你就可以开始键入数据，而该单元格的内容也会随之更改。

你刚刚看到的是 DefaultCellEditor 类的三种变型。DefaultCellEditor 可以用 JTextField、JCheckBox 或者 JComboBox 来构造。JTable 类会自动为 Boolean 类型的单元格安装一个复选框编辑

器，并为所有可编辑但未提供它们自己的绘制器的单元格安装一个文本编辑器。文本框可以让用户去编辑那些对表格模型 `getValueAt` 方法的返回值执行 `toString` 操作而产生的字符串。

Planet	Radius	Moons	Gaseous	Color	Image
Mercury	2,440	0			
Venus	6,052	0 1 2 3 4 5 6 7			
Earth	6,378	1			

图 11-9 单元格编辑器

一旦编辑完成，通过调用编辑器的 `getCellEditorValue` 方法就可以读取编辑过的值。该方法应该返回一个正确类型的值（也就是模型的 `getColumnType` 方法返回的类型）。

为了获得一个组合框编辑器，你需要手动设置单元格编辑器，因为 `JTable` 构件并不知道什么样的值对某一特殊类型来说是适合的。对于 Moons 列来说，我们希望可以让用户选择  $0 \sim 20$  之间的任何值。下面是对组合框进行初始化的代码。

```
var moonCombo = new JComboBox();
for (int i = 0; i <= 20; i++)
    moonCombo.addItem(i);
```

为了构造一个 `DefaultCellEditor`，需要在该构造器中提供一个组合框。

```
var moonEditor = new DefaultCellEditor(moonCombo);
```

接下来，我们需要安装这个编辑器。与颜色单元格绘制器不同，这个编辑器不依赖于对象类型，我们未必想要把它作用于类型为 `Integer` 的所有对象上。相反，我们需要把它安装到一个特定列中：

```
moonColumn.setCellEditor(moonEditor);
```

#### 11.1.4.4 定制编辑器

再次运行一下示例程序并点击一种颜色。这时会弹出一个颜色选择器让你为行星选择一种新颜色。选中一种颜色，然后点击 OK，单元格颜色就会随之更新（参见图 11-10）。

颜色单元格编辑器并不是一种标准的表格单元格编辑器，而是一种定制实现的编辑器。为了创建一个定制的单元格编辑器，需要实现 `TableCellEditor` 接口。这个接口有点拖沓冗长，从 Java SE 1.3 开始，提供了 `AbstractCellEditor` 类，用于负责事件处理的细节。

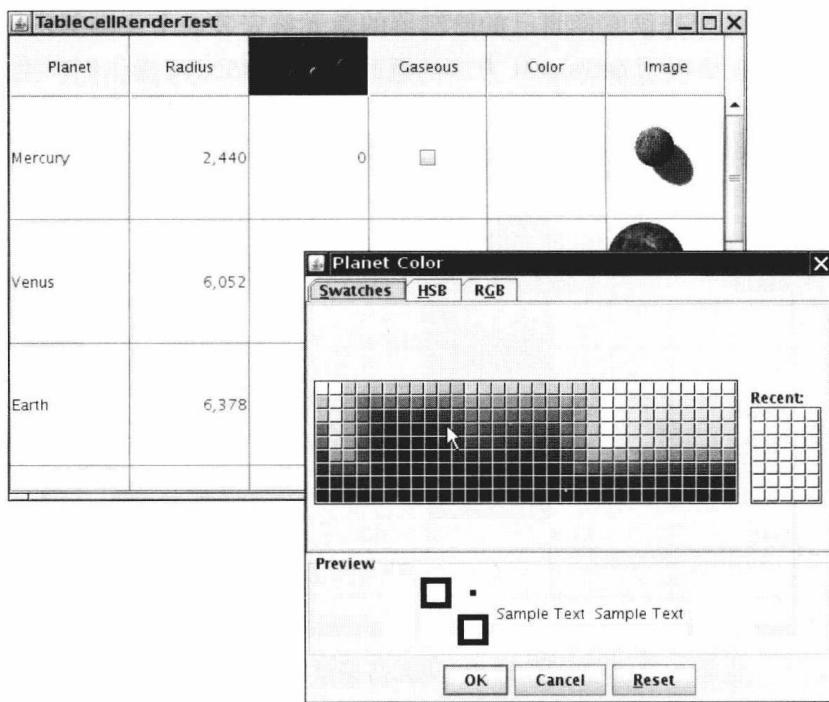


图 11-10 使用颜色选择器对单元格的颜色进行编辑

`TableCellEditor` 接口的 `getTableCellEditorComponent` 方法请求某个构件去绘制单元格。除了没有 `focus` 参数之外，它和 `TableCellRenderer` 接口的 `getTableCellRendererComponent` 方法极为相似。因为我们要编辑单元格，所以假设它获得了焦点。在编辑过程中，编辑器构件会暂时取代绘制器。在我们的示例中，返回的是一个没有颜色的空面板。这只是告诉用户该单元格正在被编辑。

接下来，当用户点击单元格时，你希望能弹出自己的编辑器。

`JTable` 类用一个事件（例如鼠标点击）去调用你的编辑器，以便确定该事件是否可以被接受去启动编辑过程。`AbstractCellEditor` 将该方法定义为能够接收所有的事件类型。

```
public boolean isCellEditable(EventObject anEvent)
{
    return true;
}
```

然而，如果你将该方法覆盖成 `false`，那么表格模型就不会遇到插入编辑器构件这样的麻烦了。

一旦安装了编辑器构件，假设我们使用的是相同的事件，那么 `shouldSelectCell` 方法就会被调用。应该在这个方法中启动编辑过程，例如，弹出一个外部的编辑对话框。

```
public boolean shouldSelectCell(EventObject anEvent)
{
    colorDialog.setVisible(true);
    return true;
}
```

如果用户取消编辑，表格会调用 `cancelCellEditing` 方法。如果用户已经点击了另一个表格单元，那么表格会调用 `stopCellEditing` 方法。在这两种情况中，都应该将对话框隐藏起来。当 `stopCellEditing` 方法被调用时，表格可能会使用被部分编辑的值。如果当前值有效，那么应该返回 `true`。在颜色选择器中，任何值都是有效的。但是如果编辑的是其他数据，那么应该保证只有有效的数据才能从编辑器中读取出来。

另外，应该调用超类的方法，以便进行事件的触发，否则，编辑事件就无法正确地取消。

```
public void cancelCellEditing()
{
    colorDialog.setVisible(false);
    super.cancelCellEditing();
}
```

最后，必须提供一个方法，以便产生用户在编辑过程中所提供的值。

```
public Object getCellEditorValue()
{
    return colorChooser.getColor();
}
```

总结一下，定制编辑器应该遵循下面几点：

1. 继承 `AbstractCellEditor` 类，并实现 `TableCellEditor` 接口。

2. 定义 `getTableCellEditorComponent` 方法以提供一个构件。它可以是一个哑构件（如果弹出一个对话框）或者是用于就地编辑的构件，例如复选框或文本框。

3. 定义 `shouldSelectCell`、`stopCellEditing` 及 `cancelCellEditing` 方法，来处理编辑过程的启动、完成以及取消。`stopCellEditing` 和 `cancelCellEditing` 方法应该调用超类方法以保证监听器能够接收到通知。

4. 定义 `getCellEditorValue` 方法返回编辑结果的值。

最后，通过调用 `stopCellEditing` 和 `cancelCellEditing` 方法，以表明用户什么时间完成了编辑操作。在构建颜色对话框的时候，我们安装了接受和取消的回调，用于触发这些事件。

```
colorDialog = JColorChooser.createDialog(null, "Planet Color", false, colorChooser,
    EventHandler.create(ActionListener.class, this, "stopCellEditing"),
    EventHandler.create(ActionListener.class, this, "cancelCellEditing"));
```

这样就完成了定制编辑器的实现过程。

你现在已经知道了怎样使一个单元格可编辑，以及怎样安装一个编辑器。还剩下一个问题，即怎样使用用户编辑过的值来更新表格模型。当编辑完成的时候，`JTable` 类会调用表格模型的下面这个方法：

```
void setValueAt(Object value, int r, int c)
```

需要将这个方法覆盖掉以便存储新值。`value` 参数是单元格编辑器返回的对象。如果实现了单元格编辑器，那么你就知道从 `getCellEditorValue` 方法返回的是什么类型的对象。在 `DefaultCellEditor` 这种情况中，这个值有三种可能：如果单元格编辑器是复选框，那么它就是

`Boolean` 值；如果是一个文本框，那么它就是一个字符串；如果这个值来源于组合框，那么就是用户选定的对象。

如果 `value` 对象不具有合适的类型，那么需要对它进行转换。例如，在一个文本框中编辑一个数字，这种情况最常发生。在我们的示例中，我们是将组合框组装成了 `Integer` 对象，所以不需要任何转换。

#### 程序清单 11-4 tableCellRender/TableCellRenderFrame.java

```

1 package tableCellRender;
2
3 import java.awt.*;
4 import javax.swing.*;
5 import javax.swing.table.*;
6
7 /**
8  * This frame contains a table of planet data.
9  */
10 public class TableCellRenderFrame extends JFrame
11 {
12     private static final int DEFAULT_WIDTH = 600;
13     private static final int DEFAULT_HEIGHT = 400;
14
15     public TableCellRenderFrame()
16     {
17         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
18
19         var model = new PlanetTableModel();
20         var table = new JTable(model);
21         table.setRowSelectionAllowed(false);
22
23         // set up renderers and editors
24
25         table.setDefaultRenderer(Color.class, new ColorTableCellRenderer());
26         table.setDefaultEditor(Color.class, new ColorTableCellEditor());
27
28         var moonCombo = new JComboBox<Integer>();
29         for (int i = 0; i <= 20; i++)
30             moonCombo.addItem(i);
31
32         TableColumnModel columnModel = table.getColumnModel();
33         TableColumn moonColumn = columnModel.getColumn(PlanetTableModel.MOONS_COLUMN);
34         moonColumn.setCellEditor(new DefaultCellEditor(moonCombo));
35         moonColumn.setHeaderRenderer(table.getDefaultRenderer(ImageIcon.class));
36         moonColumn.setHeaderValue(new ImageIcon(getClass().getResource("Moons.gif")));
37
38         // show table
39
40         table.setRowHeight(100);
41         add(new JScrollPane(table), BorderLayout.CENTER);
42     }
43 }
```

## 程序清单 11-5 tableCellRender/PlanetTableModel.java

```

1 package tableCellRender;
2
3 import java.awt.*;
4 import javax.swing.*;
5 import javax.swing.table.*;
6
7 /**
8 * The planet table model specifies the values, rendering and editing properties for the
9 * planet data.
10 */
11 public class PlanetTableModel extends AbstractTableModel
12 {
13     public static final int PLANET_COLUMN = 0;
14     public static final int MOONS_COLUMN = 2;
15     public static final int GASEOUS_COLUMN = 3;
16     public static final int COLOR_COLUMN = 4;
17
18     private Object[][] cells =
19     {
20         { "Mercury", 2440.0, 0, false, Color.YELLOW,
21             new ImageIcon(getClass().getResource("Mercury.gif")) },
22         { "Venus", 6052.0, 0, false, Color.YELLOW,
23             new ImageIcon(getClass().getResource("Venus.gif")) },
24         { "Earth", 6378.0, 1, false, Color.BLUE,
25             new ImageIcon(getClass().getResource("Earth.gif")) },
26         { "Mars", 3397.0, 2, false, Color.RED,
27             new ImageIcon(getClass().getResource("Mars.gif")) },
28         { "Jupiter", 71492.0, 16, true, Color.ORANGE,
29             new ImageIcon(getClass().getResource("Jupiter.gif")) },
30         { "Saturn", 60268.0, 18, true, Color.ORANGE,
31             new ImageIcon(getClass().getResource("Saturn.gif")) },
32         { "Uranus", 25559.0, 17, true, Color.BLUE,
33             new ImageIcon(getClass().getResource("Uranus.gif")) },
34         { "Neptune", 24766.0, 8, true, Color.BLUE,
35             new ImageIcon(getClass().getResource("Neptune.gif")) },
36         { "Pluto", 1137.0, 1, false, Color.BLACK,
37             new ImageIcon(getClass().getResource("Pluto.gif")) }
38     };
39
40     private String[] columnNames = { "Planet", "Radius", "Moons", "Gaseous",
41         "Color", "Image" };
42
43     public String getColumnName(int c)
44     {
45         return columnNames[c];
46     }
47
48     public Class<?> getColumnClass(int c)
49     {
50         return cells[0][c].getClass();
51     }
52

```

```

53     public int getColumnCount()
54     {
55         return cells[0].length;
56     }
57
58     public int getRowCount()
59     {
60         return cells.length;
61     }
62
63     public Object getValueAt(int r, int c)
64     {
65         return cells[r][c];
66     }
67
68     public void setValueAt(Object obj, int r, int c)
69     {
70         cells[r][c] = obj;
71     }
72
73     public boolean isCellEditable(int r, int c)
74     {
75         return c == PLANET_COLUMN || c == MOONS_COLUMN || c == GASEOUS_COLUMN
76             || c == COLOR_COLUMN;
77     }
78 }

```

**程序清单 11-6 tableCellRender/ColorTableCellRenderer.java**

```

1 package tableCellRender;
2
3 import java.awt.*;
4 import javax.swing.*;
5 import javax.swing.table.*;
6
7 /**
8  * This renderer renders a color value as a panel with the given color.
9  */
10 public class ColorTableCellRenderer extends JPanel implements TableCellRenderer
11 {
12     public Component getTableCellRendererComponent(JTable table, Object value,
13             boolean isSelected, boolean hasFocus, int row, int column)
14     {
15         setBackground((Color) value);
16         if (hasFocus) setBorder(UIManager.getBorder("Table.focusCellHighlightBorder"));
17         else setBorder(null);
18         return this;
19     }
20 }

```

**程序清单 11-7 tableCellRender/ColorTableCellEditor.java**

```

1 package tableCellRender;
2

```

```
3 import java.awt.*;
4 import java.awt.event.*;
5 import java.beans.*;
6 import java.util.*;
7 import javax.swing.*;
8 import javax.swing.table.*;
9
10 /**
11  * This editor pops up a color dialog to edit a cell value.
12 */
13 public class ColorTableCellEditor extends AbstractCellEditor implements TableCellEditor
14 {
15     private JColorChooser colorChooser;
16     private JPanel colorDialog;
17     private JPanel panel;
18
19     public ColorTableCellEditor()
20     {
21         panel = new JPanel();
22         // prepare color dialog
23
24         colorChooser = new JColorChooser();
25         colorDialog = JColorChooser.createDialog(null, "Planet Color", false, colorChooser,
26             EventHandler.create(ActionListener.class, this, "stopCellEditing"),
27             EventHandler.create(ActionListener.class, this, "cancelCellEditing"));
28     }
29
30     public Component getTableCellEditorComponent(JTable table, Object value,
31         boolean isSelected, int row, int column)
32     {
33         // this is where we get the current Color value. We store it in the dialog in case the
34         // user starts editing
35         colorChooser.setColor((Color) value);
36         return panel;
37     }
38
39     public boolean shouldSelectCell(EventObject anEvent)
40     {
41         // start editing
42         colorDialog.setVisible(true);
43
44         // tell caller it is ok to select this cell
45         return true;
46     }
47
48     public void cancelCellEditing()
49     {
50         // editing is canceled--hide dialog
51         colorDialog.setVisible(false);
52         super.cancelCellEditing();
53     }
54
55     public boolean stopCellEditing()
56     {
```

```

57     // editing is complete--hide dialog
58     colorDialog.setVisible(false);
59     super.stopCellEditing();
60
61     // tell caller is is ok to use color value
62     return true;
63 }
64
65 public Object getCellEditorValue()
66 {
67     return colorChooser.getColor();
68 }
69 }
```

**API** **javax.swing.JTable 1.2**

- `TableCellRenderer getDefaultRenderer(Class<?> type)`  
获取给定类型的默认绘制器。
- `TableCellEditor getDefaultEditor(Class<?> type)`  
获取给定类型的默认编辑器。

**API** **javax.swing.table.TableCellRenderer 1.2**

- `Component getTableCellRendererComponent(JTable table, Object value, boolean selected, boolean hasFocus, int row, int column)`  
返回一个构件，它的 `paint` 方法将被调用以便绘制一个表格单元格。  
 参数：  
`table` 该表格包含要绘制的单元格  
`value` 要绘制的单元格  
`selected` 如果该单元格当前已被选中，则为 `true`  
`hasFocus` 如果该单元格当前具有焦点，则为 `true`  
`row, column` 单元格的行及列

**API** **javax.swing.table.TableColumn 1.2**

- `void setCellEditor(TableCellEditor editor)`
- `void setCellRenderer(TableCellRenderer renderer)`  
为该列中的所有单元格设置单元格编辑器或绘制器。
- `void setHeaderRenderer(TableCellRenderer renderer)`  
为该列中的所有表头单元格设置单元格绘制器。
- `void setHeaderValue(Object value)`  
为该列中的表头设置用于显示的值。

**API** **javax.swing.DefaultCellEditor 1.2**

- `DefaultCellEditor(JComboBox comboBox)`  
构建一个单元格编辑器，并以一个组合框的形式显示出来，用于选择单元格的值。

**API** **`javax.swing.table.TableCellEditor 1.2`**

- `Component getTableCellEditorComponent(JTable table, Object value, boolean selected, int row, int column)`

返回一个构件，它的 `paint` 方法用于绘制表格的单元格。

参数： `table` 包含要绘制的单元格的表格

`value` 要绘制的单元格

`selected` 如果该单元格已被当前选中，则为 `true`

`row, column` 单元格的行及列

**API** **`javax.swing.CellEditor 1.2`**

- `boolean isCellEditable(EventObject event)`

如果该事件能够启动对该单元格的编辑过程，那么返回 `true`。

- `boolean shouldSelectCell(EventObject anEvent)`

启动编辑过程。如果被编辑的单元格应该被选中，则返回 `true`。通常情况下，你希望返回的是 `true`，不过，如果你不希望在编辑过程中改变单元格被选中的情况，那么你可以返回 `false`。

- `void cancelCellEditing()`

取消编辑过程。你可以放弃已进行了部分编辑的操作。

- `boolean stopCellEditing()`

出于使用编辑结果的目的，停止编辑过程。如果被编辑的值对读取来说处于适合的状态，则返回 `true`。

- `Object getCellEditorValue()`

返回编辑结果。

- `void addCellEditorListener(CellEditorListener l)`

- `void removeCellEditorListener(CellEditorListener l)`

添加或移除必需的单元格编辑器的监听器。

## 11.2 树

每个使用过分层结构的文件系统的计算机用户都见过树状显示。当然，目录和文件形式仅仅是树状组织结构中的一种。日常生活中还有很多这样的树结构，例如国家、州以及城市之间的层次结构，如图 11-11 所示。

作为一名编程人员，我们经常需要显示这些树形结构。幸运的是，Swing 类库中有一个正是用于此目的的 `JTree` 类。`JTree` 类（以及它的辅助类）负责布局树状结构，按照用户请求展开或折叠树的节点。在本节中，我们将介绍怎样使用 `JTree` 类。

与其他复杂的 Swing 构件一样，我们必须集中介绍一些常用方法，无法涉及所有的细节。如果读者想获得与众不同的效果，我们推荐你参考 David M. Geary 撰写的 *Graphic Java*

(第3版), Kim Topley 编写的 *Core Swing*。

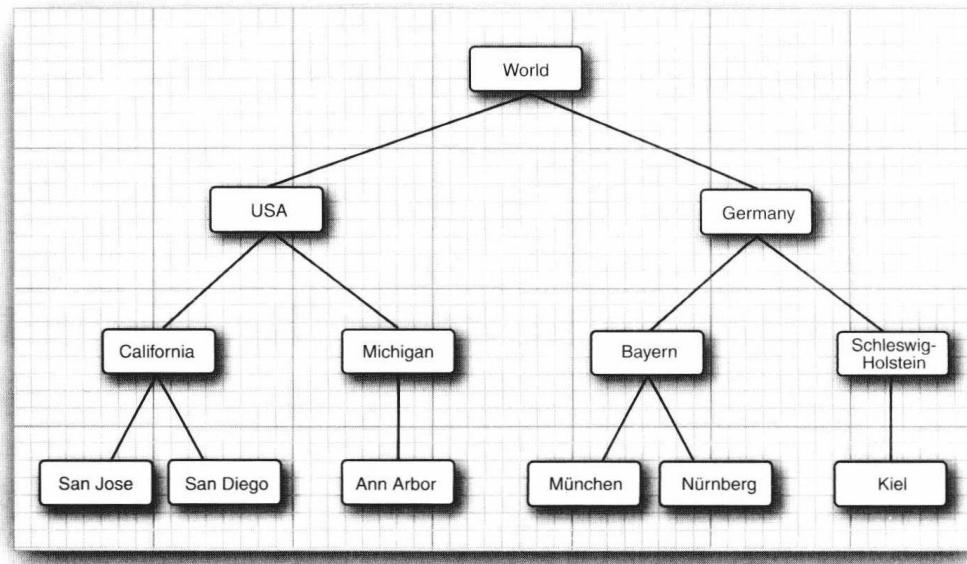


图 11-11 国家、州及城市的层次结构

在我们深入展开之前, 先介绍一些术语(参见图 11-12)。一棵树由一些节点(node)组成。每个节点要么是叶节点(leaf)要么是有孩子节点(child node)的节点。除了根节点(root node), 每一个节点都有一个唯一的父节点(parent node)。一棵树只有一个根节点。有时, 你可能有一个树的集合, 其中每棵树都有自己的根节点。这样的集合称作森林(forest)。

### 11.2.1 简单的树

在第一个示例程序中, 我们仅仅展示了一个具有几个节点的树(参见图 11-14)。

如同大多数 Swing 构件一样, 只要提供一个数据模型, 构件就可以将它显示出来。为了构建 JTree, 需要在构造器中提供这样一个树模型:

```
TreeModel model = . . .;
var tree = new JTree(model);
```

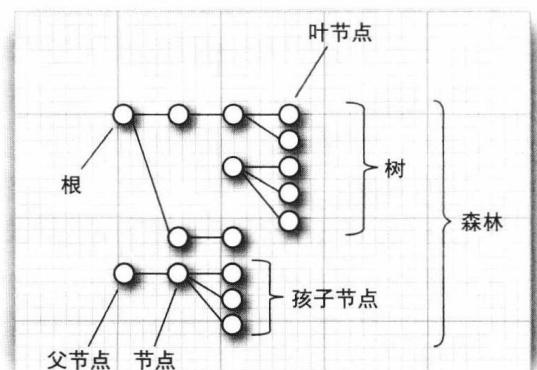


图 11-12 树中的术语

**注释:** 还有一些构造器可以用一些元素的集合来构建树。

```
JTree(Object[] nodes)
JTree(Vector<?> nodes)
JTree(Hashtable<?, ?> nodes) // the values become the nodes
```

这些构造器不是特别有用。它们仅仅是创建出一个包含了若干棵树的森林，其中每棵树只有一个节点。第三个构造器显得特别没用，因为这些节点实际的显示次序是由键的散列码确定的。

怎样才能获得一个树模型呢？可以通过创建一个实现了 TreeModel 接口的类来构建自己的树模型。在本章的后面部分，将会介绍应该如何实现。现在，我们仍坚持使用 Swing 类库提供的 DefaultTreeModel 模型。

为了构建一个默认的树模型，必须提供一个根节点。

```
TreeNode root = . . .;
var model = new DefaultTreeModel(root);
```

TreeNode 是另外一个接口。可以将任何实现了这个接口的类的对象组装到默认的树模型中。这里，我们使用的是 Swing 提供的具体节点类，叫做 DefaultMutableTreeNode。这个类实现了 MutableTreeNode 接口，该接口是 TreeNode 的一个子接口（参见图 11-13）。

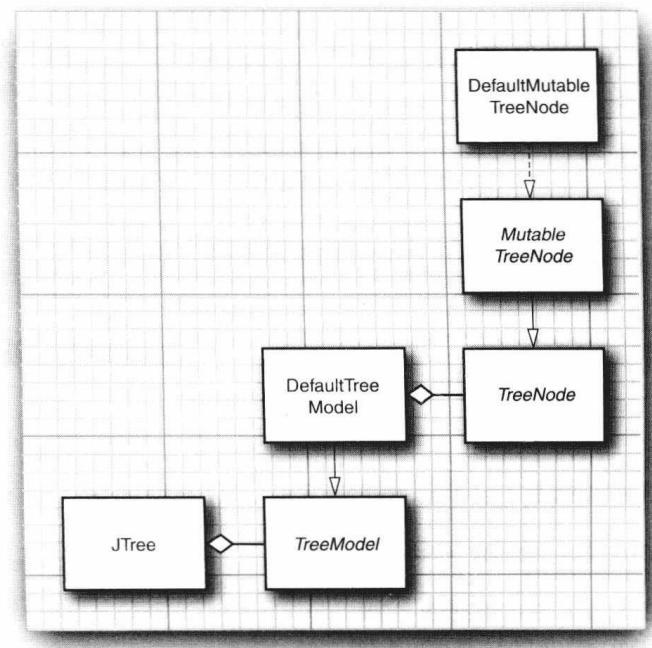


图 11-13 有关树的类

任何一个默认的可变树节点都存放着一个对象，即用户对象（user object）。树会为所有的节点绘制这些用户对象。除非指定一个绘制器，否则树将直接显示执行完 `toString` 方法之后的结果字符串。

在第一个示例程序中，我们使用了字符串作为用户对象。实际应用中，通常会在树中组装更具表现力的用户对象。例如，当显示一个目录树时，将 File 对象用于节点将具有实际意义。

可以在构造器中设定用户对象，也可以稍后在 `setUserObject` 方法中设定用户对象：

```
var node = new DefaultMutableTreeNode("Texas");
...
node.setUserObject("California");
```

接下来，可以建立节点之间的父 / 子关系。从根节点开始，使用 `add` 方法来添加子节点：

```
var root = new DefaultMutableTreeNode("World");
var country = new DefaultMutableTreeNode("USA");
root.add(country);
var state = new DefaultMutableTreeNode("California");
country.add(state);
```

图 11-14 显示了这棵树的外观。

按照这种方式将所有的节点链接起来。然后用根节点构建一个 `DefaultTreeModel`。最后，用这个树模型构建一个 `JTree`。

```
var treeModel = new DefaultTreeModel(root);
var tree = new JTree(treeModel);
```

或者，使用快捷方式，直接将根节点传递给 `JTree` 构造器。那么这棵树就会自动构建一个默认的树模型：

```
var tree = new JTree(root);
```

程序清单 11-8 给出了完整的代码。



图 11-14 一棵简单的树

#### 程序清单 11-8 tree/SimpleTreeFrame.java

```
1 package tree;
2
3 import javax.swing.*;
4 import javax.swing.tree.*;
5
6 /**
7  * This frame contains a simple tree that displays a manually constructed tree model.
8  */
9 public class SimpleTreeFrame extends JFrame
10 {
11     private static final int DEFAULT_WIDTH = 300;
12     private static final int DEFAULT_HEIGHT = 200;
13
14     public SimpleTreeFrame()
15     {
16         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
17
18         // set up tree model data
19
20         var root = new DefaultMutableTreeNode("World");
21         var country = new DefaultMutableTreeNode("USA");
22         root.add(country);
23         var state = new DefaultMutableTreeNode("California");
24         country.add(state);
```

```

25     var city = new DefaultMutableTreeNode("San Jose");
26     state.add(city);
27     city = new DefaultMutableTreeNode("Cupertino");
28     state.add(city);
29     state = new DefaultMutableTreeNode("Michigan");
30     country.add(state);
31     city = new DefaultMutableTreeNode("Ann Arbor");
32     state.add(city);
33     country = new DefaultMutableTreeNode("Germany");
34     root.add(country);
35     state = new DefaultMutableTreeNode("Schleswig-Holstein");
36     country.add(state);
37     city = new DefaultMutableTreeNode("Kiel");
38     state.add(city);
39
40     // construct tree and put it in a scroll pane
41
42     var tree = new JTree(root);
43     add(new JScrollPane(tree));
44 }
45 }
```

运行这段程序代码时，最初的树外观如图 11-15 所示。只有根节点和它的子节点可见。单击圆圈图标（把手）展开子树。当子树折叠起来时，把手图标的线伸出指向右边，当子树展开时，把手图标的线伸出指向下方（参见图 11-16）。虽然我们无法得知 Metal 外观的设计者当时是如何构想的，但是我们可以将这个图标看作一个门把手，按下把手就可以打开子树。

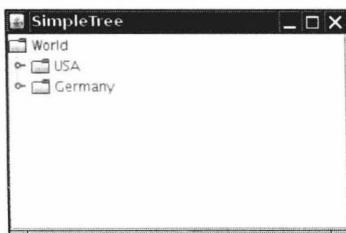


图 11-15 最初的树的显示

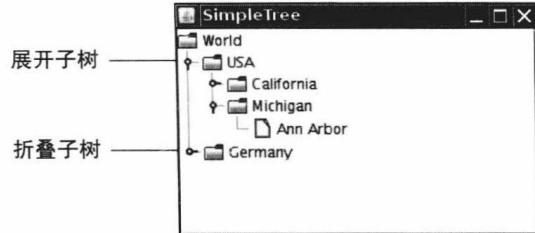


图 11-16 折叠和展开后的子树

**注释：**当然，树的显示还依赖于所选择的外观模式。我们这里只讨论 Metal 这种外观模式。在 Windows 或 Motif 外观模式中，把手则具有我们更熟悉的外观，即带有“-”或“+”的框结构（参见图 11-17）。

可以使用下面这句神奇的代码取消父子节点之间的连接线（参见图 11-18）：

```
tree.putClientProperty("JTree.lineStyle", "None");
```

相反地，如果要确保显示这些线条，则可以使用：

```
tree.putClientProperty("JTree.lineStyle", "Angled");
```

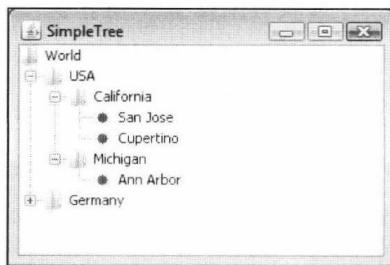


图 11-17 一棵具有 Windows 外观的树



图 11-18 不带连接线的树

另一种线条样式，“水平线”，如图 11-19 所示。这棵树显示有水平线，而这些水平线只是用来将根节点的孩子节点分离开来。我们很难说清这样做的好处。

默认情况下，这种树中的根节点没有用于折叠的手柄。如果需要的话，可以通过下面的调用添加一个把手：

```
tree.setShowsRootHandles(true);
```

图 11-20 显示了调用后的结果。现在就可以将整棵树折叠到根节点中了。



图 11-19 具有水平线样式的树

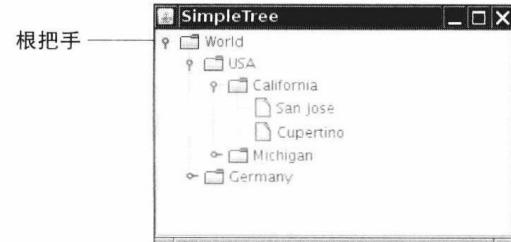


图 11-20 具有一个根把手的树

相反地，也可以将根节点完全隐藏起来。这样做只是为了显示一个森林，即一个树集，每棵树都有它自己的根节点。但是仍然必须将森林中的所有树都放到一个公共节点下。因此，可以使用下面这条指令将根节点隐藏起来。

```
tree.setRootVisible(false);
```

请观察图 11-21。它看起来似乎有两个根节点，分别用“USA”和“Germany”标识了出来，而实际上将二者合并起来的根节点是不可见的。

让我们将注意力从树的根节点转移到叶节点。注意，这些叶节点的图标和其他节点的图标是不同的（参见图 11-22）。

在显示这棵树的时候，每个节点都绘有一个图标。实际上一共有三种图标：叶节点图标、展开的非叶节点图标以及闭合的非叶节点图标。为了简化起见，我们将后面两种图标称为文件夹图标。

节点绘制器必须知道每个节点要使用什么样的图标。默认情况下，这个决策过程是这样的：如果某个节点的 `isLeaf` 方法返回的是 `true`，那么就使用叶节点图标，否则，使用文件夹图标。



图 11-21 一个森林

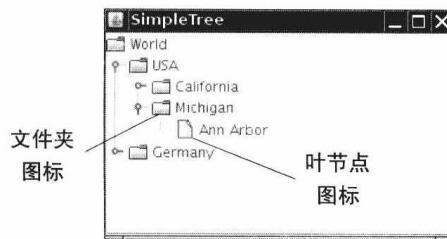


图 11-22 叶节点和折叠节点的图标

如果某个节点没有任何儿子节点，那么 `DefaultMutableTreeNode` 类的 `isLeaf` 方法将返回 `true`。因此，具有儿子节点的节点使用文件夹图标，没有儿子节点的节点使用叶节点图标。

有时，这种做法并不合适。假设我们要向我们那棵简单的树中添加一个“Montana”节点，但是我们还不知道要添加什么城市。此时，我们并不希望一个州节点使用叶节点图标，因为从概念上来讲，只有城市才使用叶节点。

`JTree` 类无法知道哪些节点是叶节点，它要询问树模型。如果一个没有任何子节点的节点不应该自动地被设置为概念上的叶节点，那么可以让树模型对这些叶节点使用一个不同的标准，即可以查询其“允许有子节点”的节点属性。

对于那些不应该有子节点的节点，调用

```
node.setAllowsChildren(false);
```

然后，告诉树模型去查询“允许有子节点”的属性值以确定一个节点是否应该显示成叶子图标。可以使用 `DefaultTreeModel` 类中的方法 `setAsksAllowsChildren` 设定此动作：

```
model.setAsksAllowsChildren(true);
```

有了这个判定规则，允许有子节点的节点就可以获得文件夹图标，而不允许有子节点的节点将获得叶子图标。

另外，如果是通过提供根节点来构建一棵树的，那么请在构造器中直接提供“询问允许有子节点”属性值的设置。

```
var tree = new JTree(root, true); // nodes that don't allow children get leaf icons
```

#### API javax.swing.JTree 1.2

- `JTree(TreeModel model)`

根据一个树模型构造一棵树。

- `JTree(TreeNode root)`

- `JTree(TreeNode root, boolean asksAllowChildren)`

使用默认的树模型构造一棵树，显示根节点和它的子节点。

参数： `root` 根节点

`asksAllowChildren` 如果设置为 `true`，则使用“允许有子节点”的节点属性来确定一个节点是否是叶节点

- `void setShowsRootHandles(boolean b)`

如果 b 为 true，则根节点具有折叠或展开它的子节点的把手图标。

- void setRootVisible(boolean b)

如果 b 为 true，则显示根节点，否则隐藏根节点。

#### API **javax.swing.tree.TreeNode** 1.2

- boolean isLeaf()

如果该节点是一个概念上的叶节点，则返回 true。

- boolean getAllowsChildren()

如果该节点可以拥有子节点，则返回 true。

#### API **javax.swing.tree.MutableTreeNode** 1.2

- void setUserObject(Object userObject)

设置树节点用于绘制的“用户对象”。

#### API **javax.swing.tree.TreeModel** 1.2

- boolean isLeaf(Object node)

如果该节点应该以叶节点的形式显示，则返回 true。

#### API **javax.swing.tree.DefaultTreeModel** 1.2

- void setAsksAllowsChildren(boolean b)

如果 b 为 true，那么当节点的 getAllowsChildren 方法返回 false 时，这些节点显示为叶节点。否则，当节点的 isLeaf 方法返回 true 时，它们显示为叶节点。

#### API **javax.swing.tree.DefaultMutableTreeNode** 1.2

- DefaultMutableTreeNode(Object userObject)

用给定的用户对象构建一个可变树节点。

- void add(MutableTreeNode child)

将一个节点添加为该节点最后一个子节点。

- void setAllowsChildren(boolean b)

如果 b 为 true，则可以向该节点添加子节点。

#### API **javax.swing.JComponent** 1.2

- void putClientProperty(Object key, Object value)

将一个键 / 值对添加到一个小表格中，每一个构件都管理着这样的一个小表格。这是一种“紧急逃生”机制，很多 Swing 构件用它来存放与外观相关的属性。

### 11.2.1.1 编辑树和树的路径

在下面的一个示例程序中，将会看到怎样编辑一棵树。图 11-23 显示了用户界面。如果点击“Add Sibling”（添加兄弟节点）或“Add Child”（添加子节点）按钮，该程序将向树中添加一个新节点（带有“New”标题）。如果你点击“Delete”（删除）按钮，该程序将删除当

前选中的节点。

为了实现这种行为，需要弄清楚当前选定的是哪个节点。JTree 类用的是一种令人惊讶的方式来标识树中的节点。它并不处理树的节点，而是处理对象路径（称为树路径）。一个树路径从根节点开始，由一个子节点序列构成，参见图 11-24。

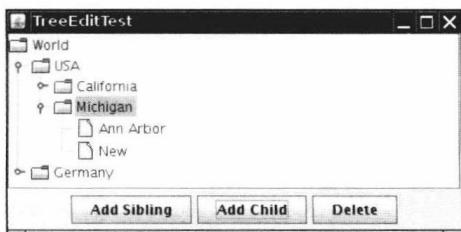


图 11-23 编辑一棵树

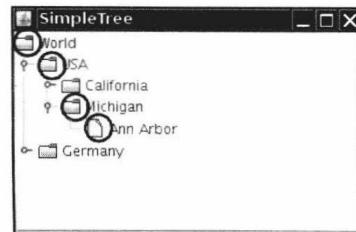


图 11-24 一个树路径

你可能要怀疑 JTree 类为什么需要整个路径。它不能只获得一个 TreeNode，然后不断调用 getParent 方法吗？实际上，JTree 类一点都不清楚 TreeNode 接口的情况。该接口从来没有被 TreeModel 接口用到过，它只被 DefaultTreeModel 的实现用到了。你完全可以拥有其他的树模型，这些树模型中的节点可能根本就没有实现 TreeNode 接口。如果你使用的是一个管理其他类型对象的树模型，那么这些对象有可能根本就没有 getParent 和 getChild 方法。它们彼此之间当然会有其他某种连接。将其他节点连接起来这是树模型的职责，JTree 类本身并没有节点之间连接属性的任何线索。因此，JTree 类总是需要用完整的路径来工作。

TreePath 类管理着一个 Object（不是 TreeNode！）引用序列。有很多 JTree 的方法都可以返回 TreePath 对象。当拥有一个树路径时，通常只需要知道其终端节点，该节点可以通过 getLastPathComponent 方法得到。例如，如果要查找一棵树中当前选定的节点，可以使用 JTree 类中的 getSelectionPath 方法。它将返回一个 TreePath 对象，根据这个对象就可以检索实际节点。

```
TreePath selectionPath = tree.getSelectionPath();
var selectedNode = (DefaultMutableTreeNode) selectionPath.getLastPathComponent();
```

实际上，由于这种特定查询经常被使用到，因此还提供了一个更方便的方法，它能够立即给出选定的节点。

```
var selectedNode = (DefaultMutableTreeNode) tree.getLastSelectedPathComponent();
```

该方法之所以没有被称为 getSelectedNode，是因为这棵树并不了解它包含的节点，它的树模型只处理对象的路径。

**注释：**树路径是 JTree 类描述节点的两种方式之一。JTree 有许多方法可以接收或返回一个整数索引——行的位置。行的位置仅仅是节点在树中显示的一个行号（从 0 开始）。只有那些可视节点才有行号，并且如果一个节点之前的其他节点展开、折叠或者被修改过，这个节点的行号也会随之改变。因此，你应该避免使用行的位置。相反地，所有使用行的 JTree 方法都有一个与之等价的使用树路径的方法。

一旦选定了某个节点，那么就可以对它进行编辑了。不过，不能直接向树节点添加子节点：

```
selectedNode.addNode(newNode); // No!
```

如果改变了节点的结构，那么改变的只是树模型，而相关的视图却没有被通知到。可以自己发送一个通知消息，但是如果使用 `DefaultTreeModel` 类的 `insertNodeInto` 方法，那么该模型类会全权负责这件事情。例如，下面的调用可以将一个新节点作为选定节点的最后子节点添加到树中，并通知树的视图。

```
model.insertNodeInto(newNode, selectedNode, selectedNode.getChildCount());
```

类似的调用 `removeNodeFromParent` 可以移除一个节点并通知树的视图：

```
model.removeNodeFromParent(selectedNode);
```

如果想保持节点结构，但是要改变用户对象，那么可以调用下面这个方法：

```
model.nodeChanged(changedNode);
```

自动通知是使用 `DefaultTreeModel` 的主要优势。如果你提供自己的树模型，那么必须自己动手实现这种自动通知。(详见 Kim Topley 撰写的 *Core Swing*)

**◆ 警告：**`DefaultTreeModel` 有一个 `reload` 方法能够将整个模型重新载入。但是，不要在进行了少数几个修改之后，只是为了更新树而调用 `reload` 方法。在重建一棵树的时候，根节点的子节点之后的所有节点将全部再次折叠起来。如果你的用户在每次修改之后都要不断地展开整棵树，这确实是一件令人烦心的事。

当视图接收到节点结构被改变的通知时，它会更新显示树的视图，但是不会自动展开某个节点以展现新添加的子节点。特别是在我们上面那个示例程序中，如果用户将一个新节点添加到其子节点正处于折叠状态的节点上，那么这个新添加的节点就被悄无声息地添加到了一个处于折叠状态的子树中，这就没有给用户提供任何反馈信息以告诉用户已经执行了该命令。在这种情况下，你可能需要特别费劲地展开所有的父节点，以便让新添加的节点成为可视节点。可以使用类 `JTree` 中的方法 `makeVisible` 实现这个目的。`makeVisible` 方法将接受一个树路径作为参数，该树路径指向应该变为可视的节点。

因此，需要构建一个从根节点到新添加节点的树路径。为了获得一个这样的树路径，首先要调用 `DefaultTreeModel` 类中的 `getPathToRoot` 方法，它返回一个包含了某一节点到根节点之间所有节点的数组 `TreeNode[]`。可以将这个数组传递给一个 `TreePath` 构造器。

例如，下面展示了怎样将一个新节点变成可见的：

```
TreeNode[] nodes = model.getPathToRoot(newNode);
var path = new TreePath(nodes);
tree.makeVisible(path);
```

**◆ 注释：**令人惊奇的是，`DefaultTreeModel` 类好像完全忽视了 `TreePath` 类，尽管它的职责是与一个 `JTree` 通信。`JTree` 类大量地使用到了树路径，而它从不使用节点对象数组。

但是，现在假设你的树是放在一个滚动面板里面，在展开树节点之后，新节点仍是不可见的，因为它落在视图之外。为了克服这个问题，请调用

```
tree.scrollPathToVisible(path);
```

而不是调用 `makeVisible`。这个调用将展开路径中的所有节点，并告诉外周的滚动面板将路径末端的节点滚动到视图中（参见图 11-25）。

默认情况下，这些树节点是不可编辑的。不过，如果调用

```
tree.setEditable(true);
```

那么，用户就可以编辑某一节点了。可以先双击该节点，然后编辑字符串，最后按下回车键。双击操作会调用默认单元格编辑器，它实现了 DefaultCellEditor 类（参见图 11-26）。也可以安装其他一些单元格编辑器，其过程与表格单元格编辑器中讨论的过程一样。

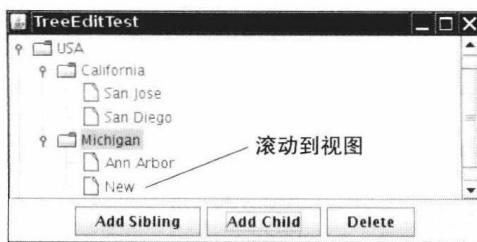


图 11-25 滚动以显示新节点的滚动面板

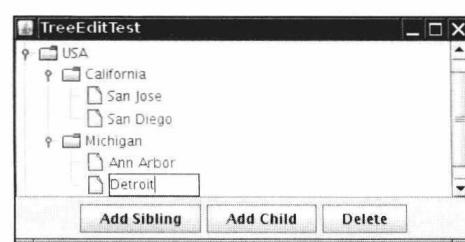


图 11-26 默认的单元格编辑器

程序清单 11-9 展示了树编辑程序的完整源代码。运行该程序，添加几个新节点，然后通过双击它们进行编辑操作。请观察折叠的节点是怎样展开以显现添加的子节点的，以及滚动面板是怎样让添加的节点保持在视图中的。

程序清单 11-9 treeEdit/TreeEditFrame.java

```
1 package treeEdit;
2
3 import java.awt.*;
4 import javax.swing.*;
5 import javax.swing.tree.*;
6
7 /**
8 * A frame with a tree and buttons to edit the tree.
9 */
10 public class TreeEditFrame extends JFrame
11 {
12     private static final int DEFAULT_WIDTH = 400;
13     private static final int DEFAULT_HEIGHT = 200;
14
15     private DefaultTreeModel model;
16     private JTree tree;
17
18     public TreeEditFrame()
19     {
20         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
21     }
22 }
```

```
21      // construct tree
22
23      TreeNode root = makeSampleTree();
24      model = new DefaultTreeModel(root);
25      tree = new JTree(model);
26      tree.setEditable(true);
27
28      // add scroll pane with tree
29
30      var scrollPane = new JScrollPane(tree);
31      add(scrollPane, BorderLayout.CENTER);
32
33      makeButtons();
34  }
35
36
37  public TreeNode makeSampleTree()
38  {
39      var root = new DefaultMutableTreeNode("World");
40      var country = new DefaultMutableTreeNode("USA");
41      root.add(country);
42      var state = new DefaultMutableTreeNode("California");
43      country.add(state);
44      var city = new DefaultMutableTreeNode("San Jose");
45      state.add(city);
46      city = new DefaultMutableTreeNode("San Diego");
47      state.add(city);
48      state = new DefaultMutableTreeNode("Michigan");
49      country.add(state);
50      city = new DefaultMutableTreeNode("Ann Arbor");
51      state.add(city);
52      country = new DefaultMutableTreeNode("Germany");
53      root.add(country);
54      state = new DefaultMutableTreeNode("Schleswig-Holstein");
55      country.add(state);
56      city = new DefaultMutableTreeNode("Kiel");
57      state.add(city);
58      return root;
59  }
60
61  /**
62   * Makes the buttons to add a sibling, add a child, and delete a node.
63   */
64  public void makeButtons()
65  {
66      var panel = new JPanel();
67      var addSiblingButton = new JButton("Add Sibling");
68      addSiblingButton.addActionListener(event ->
69      {
70          var selectedNode = (DefaultMutableTreeNode) tree.getLastSelectedPathComponent();
71
72          if (selectedNode == null) return;
73
74          var parent = (DefaultMutableTreeNode) selectedNode.getParent();
```

```

75
76     if (parent == null) return;
77
78     var newNode = new DefaultMutableTreeNode("New");
79
80     int selectedIndex = parent.getIndex(selectedNode);
81     model.insertNodeInto(newNode, parent, selectedIndex + 1);
82
83     // now display new node
84
85     TreeNode[] nodes = model.getPathToRoot(newNode);
86     var path = new TreePath(nodes);
87     tree.scrollPathToVisible(path);
88   });
89   panel.add(addSiblingButton);
90
91   var addChildButton = new JButton("Add Child");
92   addChildButton.addActionListener(event ->
93   {
94     var selectedNode = (DefaultMutableTreeNode) tree.getLastSelectedPathComponent();
95
96     if (selectedNode == null) return;
97
98     var newNode = new DefaultMutableTreeNode("New");
99     model.insertNodeInto(newNode, selectedNode, selectedNode.getChildCount());
100
101    // now display new node
102
103    TreeNode[] nodes = model.getPathToRoot(newNode);
104    var path = new TreePath(nodes);
105    tree.scrollPathToVisible(path);
106  });
107  panel.add(addChildButton);
108
109  var deleteButton = new JButton("Delete");
110  deleteButton.addActionListener(event ->
111  {
112    var selectedNode = (DefaultMutableTreeNode) tree.getLastSelectedPathComponent();
113
114    if (selectedNode != null && selectedNode.getParent() != null) model
115      .removeNodeFromParent(selectedNode);
116  });
117  panel.add(deleteButton);
118  add(panel, BorderLayout.SOUTH);
119 }
120 }
```

**API** **javax.swing.JTree 1.2**

- `TreePath getSelectionPath()`

获取到当前选定节点的路径，如果选定多个节点，则获取到第一个选定节点的路径。如果没有选定任何节点，则返回 `null`。

- `Object getLastSelectedPathComponent()`

获取表示当前选定节点的节点对象，如果选定多个节点，则获取第一个选定的节点。如果没有选定任何节点，则返回 `null`。

- `void makeVisible(TreePath path)`  
展开该路径中的所有节点。
- `void scrollPathToVisible(TreePath path)`  
展开该路径中的所有节点，如果这棵树是置于滚动面板中的，则滚动以确保该路径中的最后一个节点是可见的。

#### API `javax.swing.tree.TreePath 1.2`

- `Object getLastPathComponent()`  
获取该路径中最后一个节点，也就该路径代表的节点对象。

#### API `javax.swing.tree.TreeNode 1.2`

- `TreeNode getParent()`  
返回该节点的父节点。
- `TreeNode getChildAt(int index)`  
查找给定索引号上的子节点。该索引号必须在 0 和 `getChildCount()-1` 之间。
- `int getChildCount()`  
返回该节点的子节点个数。
- `Enumeration children()`  
返回一个枚举对象，可以迭代遍历该节点的所有子节点。

#### API `javax.swing.tree.DefaultTreeModel 1.2`

- `void insertNodeInto(MutableTreeNode newChild, MutableTreeNode parent, int index)`  
将 `newChild` 作为 `parent` 的新子节点添加到给定的索引位置上，并通知树模型的监听器。
- `void removeNodeFromParent(MutableTreeNode node)`  
将节点 `node` 从该模型中删除，并通知树模型的监听器。
- `void nodeChanged(TreeNode node)`  
通知树模型的监听器：节点 `node` 发生了改变。
- `void nodesChanged(TreeNode parent, int[] changedChildIndexes)`  
通知树模型的监听器：节点 `parent` 所有在给定索引位置上的子节点发生了改变。
- `void reload()`  
将所有节点重新载入到树模型中。这是一项动作剧烈的操作，只有当由于一些外部作用，导致树的节点完全改变时，才应该使用该方法。

### 11.2.2 节点枚举

有时为了查找树中一个节点，必须从根节点开始，遍历所有子节点直到找到相匹配的节点。`DefaultMutableTreeNode` 类有几个很方便的方法用于迭代遍历所有节点。

`breadthFirstEnumeration` 方法和 `depthFirstEnumeration` 方法分别使用广度优先或深度优先的遍历方式，返回枚举对象，它们的 `nextElement` 方法能够访问当前节点的所有子节点。图 11-27 显示了对示例树进行遍历的情况，节点标签则指示遍历节点时的先后次序。

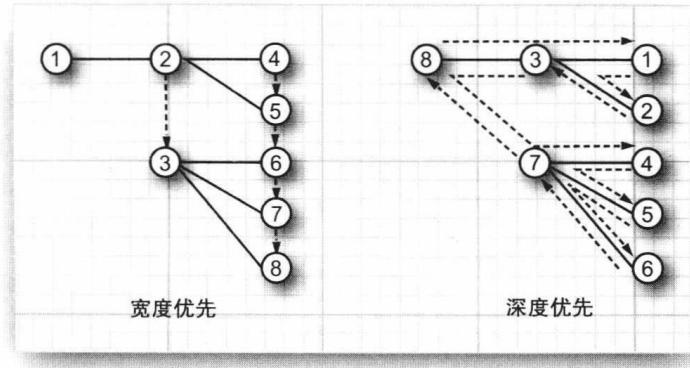


图 11-27 树的遍历顺序

按照广度优先的方式进行枚举是最容易可视化的。树是以层的形式遍历的，首先访问根节点，然后是它的所有子节点，接着是它的孙子节点，依此类推。

为了可视化深度优先的枚举，让我们想象一只老鼠陷入一个树状陷阱的情形。它沿着第一条路径迅速爬行，直到到达一个叶节点位置。然后，原路返回并转入下一条路径，依此类推。

计算机科学家也将其称为后序遍历 (postorder traversal)，因为整个查找过程是先访问到子节点，然后才访问到父节点。`postOrderTraversal` 方法是 `depthFirstTraversal` 的同义语。为了完整性，还存在一个 `preOrderTraversal` 方法，它也是一种深度优先搜索方法，但是它首先枚举父节点，然后是子节点。

下面是一种典型的使用模式：

```
Enumeration breadthFirst = node.breadthFirstEnumeration();
while (breadthFirst.hasMoreElements())
    do something with breadthFirst.nextElement();
```

最后，还有一个相关方法 `pathFromAncestorEnumeration`，用于查找一条从祖先节点到给定节点之间的路径，然后枚举出该路径中的所有节点。整个过程并不需要大量的处理操作，只需要不断调用 `getParent` 直到发现祖先节点，然后将该路径倒置过来存放即可。

在我们的下个示例程序中，将运用到节点枚举。该程序显示了类之间的继承树。向窗体最下面的文本框中输入一个类名，该类以及它的所有父类就会添加到树中（参见图 11-28）。

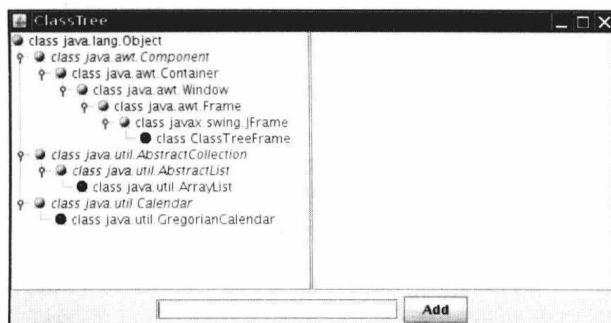


图 11-28 一棵继承树

在这个示例中，我们充分利用了这个事实，即树节点的用户对象可以是任何类型的对象。因为我们这里的节点是用来描述类的，因此我们在这些节点中存储的是 Class 对象。

当然，我们不想对同一个类对象添加两次，因此我们必须检查一个类是否已经存在于树中。如果在树中存在给定用户对象的节点，那么下面这个方法就可以用来查找该节点。

```
public DefaultMutableTreeNode findUserObject(Object obj)
{
    Enumeration e = root.breadthFirstEnumeration();
    while (e.hasMoreElements())
    {
        DefaultMutableTreeNode node = (DefaultMutableTreeNode) e.nextElement();
        if (node.getUserObject().equals(obj))
            return node;
    }
    return null;
}
```

### 11.2.3 绘制节点

在应用中可能会经常需要改变树构件绘制节点的方式，最常见的改变当然是为节点和叶节点选取不同的图标，其他一些改变可能涉及节点标签的字体或节点上的图像绘制等方面。所有这些改变都可以通过向树中安装一个新的树单元格绘制器来实现。在默认情况下，JTree 类使用 DefaultTreeCellRenderer 对象来绘制每个节点。DefaultTreeCellRenderer 类继承自 JLabel 类，该标签包含节点图标和节点标签。

 **注释：**单元格绘制器并不能绘制用于展开或折叠子树的“把手”图标。这些把手是外观模式的一部分，建议最好不要试图改变它们。

可以通过以下三种方式定制显示外观：

- 可以使用 DefaultTreeCellRenderer 改变图标、字体以及背景颜色。这些设置适用于树中所有节点。
- 可以安装一个继承了 DefaultTreeCellRenderer 类的绘制器，用于改变每个节点的图标、字体以及背景颜色。
- 可以安装一个实现了 TreeCellRenderer 接口的绘制器，为每个节点绘制自定义的图像。

让我们逐个研究这几种可能。最简单的定制方法是构建一个 DefaultTreeCellRenderer 对象，改变图标，然后将它安装到树中：

```
var renderer = new DefaultTreeCellRenderer();
renderer.setLeafIcon(new ImageIcon("blue-ball.gif")); // used for leaf nodes
renderer.setClosedIcon(new ImageIcon("red-ball.gif")); // used for collapsed nodes
renderer.setOpenIcon(new ImageIcon("yellow-ball.gif")); // used for expanded nodes
tree.setCellRenderer(renderer);
```

可以在图 11-28 中看到运行效果。我们只是使用“球”图标作为占位符，这里假设你的用户界面设计者会为你的应用提供合适的图标。

我们不建议改变整棵树中的字体或背景颜色，因为这实际上是外观设置的职责所在。

不过，改变树中个别节点的字体，以突显某些节点还是很有用的。如果仔细观察图 11-28，你会看到抽象类是设成斜体字的。

为了改变单个节点的外观，需要安装一个树单元格绘制器。树单元格绘制器与我们在本章前一节讨论的列表单元格绘制器很相似。`TreeCellRenderer` 接口只有下面这个单一方法：

```
Component getTreeCellRendererComponent(JTree tree, Object value, boolean selected,
    boolean expanded, boolean leaf, int row, boolean hasFocus)
```

`DefaultTreeCellRenderer` 类的 `getTreeCellRendererComponent` 方法返回的是 `this`，换句话说，就是一个标签 (`DefaultTreeCellRenderer` 类继承了 `JLabel` 类)。如果要定制一个构件，需要继承 `DefaultTreeCellRenderer` 类。按照以下方式覆盖 `getTreeCell RendererComponent` 方法：调用超类中的方法，以便准备标签的数据，然后定制标签属性，最后返回 `this`。

```
class MyTreeCellRenderer extends DefaultTreeCellRenderer
{
    public Component getTreeCellRendererComponent(JTree tree, Object value, boolean selected,
        boolean expanded, boolean leaf, int row, boolean hasFocus)
    {
        Component comp = super.getTreeCellRendererComponent(tree, value, selected,
            expanded, leaf, row, hasFocus);
        DefaultMutableTreeNode node = (DefaultMutableTreeNode) value;
        look at node.getUserObject();
        Font font = appropriate font;
        comp.setFont(font);
        return comp;
    }
};
```

!**警告：**`getTreeCellRendererComponent` 方法的 `value` 参数是节点对象，而不是用户对象！请记住，用户对象是 `DefaultMutableTreeNode` 的一个特性，而 `JTree` 可以包含任意类型的节点。如果树使用的是 `DefaultMutableTreeNode` 节点，那么必须在第二个步骤中获取这个用户对象，正如我们在上一个代码示例中所做的那样。

!**警告：**`DefaultTreeCellRenderer` 为所有节点使用的是相同的标签对象，仅仅是为每个节点改变标签文本而已。如果想为某个特定节点更改字体，那么必须在该方法再次调用的时候将它设置回默认值。否则，随后的所有节点都会以更改过的字体进行绘制！见程序清单 11-10 中的程序代码，看看它是怎样将字体恢复到其默认值的。

根据 `Class` 对象有无 `ABSTRACT` 修饰符，程序清单 11-10 中的 `ClassNameTreeCellRenderer` 会将类名设置为标准字体或斜体字体。我们不想设置成特殊的字体，因为我们不想改变任何通常用于显示标签的字体外观。因此，我们使用来自标签本身的字体以及从它衍生而来的一个斜体字体。请回忆一下，全部的调用只返回一个共享的单一的 `JLabel` 对象。因此，我们需要保存初始字体，并在下一次调用 `gettreeCellRendererComponent` 方法时将其恢复为初始值。

同时，注意一下我们是如何改变 `ClassTreeFrame` 构造器中的节点图标的。

**API** javax.swing.tree.DefaultMutableTreeNode 1.2

- Enumeration breadthFirstEnumeration()
- Enumeration depthFirstEnumeration()
- Enumeration preOrderEnumeration()
- Enumeration postOrderEnumeration()

返回枚举对象，用于按照某种特定顺序访问树模型中的所有节点。在广度优先遍历中，先访问离根节点更近的子节点，再访问那些离根节点远的节点。在深度优先遍历中，先访问一个节点的所有子节点，然后再访问它的兄弟节点。postOrderEnumeration 方法与 depthFirstEnumeration 基本上相似。除了先访问父节点，后访问子节点之外，先序遍历和后序遍历基本上一样。

**API** javax.swing.tree.TreeCellRenderer 1.2

- Component getTreeCellRendererComponent(JTree tree, Object value, boolean selected, boolean expanded, boolean leaf, int row, boolean hasFocus)

返回一个 paint 方法被调用的构件，以便绘制树的一个单元格。

参数：

tree	包含要绘制节点的树
value	要绘制的节点
selected	如果该节点是当前选定的节点，则为 true
expanded	如果该节点的子节点可见，则为 true
leaf	如果该节点应该显示为叶节点，则为 true
row	显示包含该节点的那行
hasFocus	如果当前选定的节点拥有输入焦点，则为 true

**API** javax.swing.tree.DefaultTreeCellRenderer 1.2

- void setLeafIcon(Icon icon)
- void setOpenIcon(Icon icon)
- void setClosedIcon(Icon icon)

设置叶节点、展开节点以及折叠节点的显示图标。

### 11.2.4 监听树事件

通常情况下，一个树构件会成对地伴随着其他某个构件。当用户选定了一些树节点时，某些信息就会在其他窗口中显示出来。参见图 11-29 的示例。当用户选定一个类时，这个类的实例及静态变量信息就会在右边的文本区显示出来。

为了获得这项功能，可以安装一个



图 11-29 一个类浏览器

树选择监听器。该监听器必须实现 TreeSelection-Listener 接口，这是一个只有下面这个单一方法的接口：

```
void valueChanged(TreeSelectionEvent event)
```

每当用户选定或者撤销选定树节点的时候，这个方法就会被调用。

可以按照下面这种通常方式向树中添加监听器：

```
tree.addTreeSelectionListener(listener);
```

可以设定是否允许用户选定一个单一的节点、连续区间内的节点或者一个任意的、可能不连续的节点集。JTree 类使用 TreeSelectionModel 来管理节点的选择。必须检索整个模型，以便将选择状态设置为 SINGLE\_TREE\_SELECTION、CONTIGUOUS\_TREE\_SELECTION 或 DISCONTIGUOUS\_TREE\_SELECTION 三种状态之一。（在默认情况下是非连续的选择模式。）例如，在我们的类浏览器中，我们希望只允许选择单个类：

```
int mode = TreeSelectionModel.SINGLE_TREE_SELECTION;
tree.getSelectionModel().setSelectionMode(mode);
```

除了设置选择模式之外，并不需要担心树的选择模型。

**注释：** 用户怎样选定多个选项则依赖于外观。在 Metal 外观中，按下 CTRL 键，同时点击一个选项将它添加到选项集中，如果当前已经选定了该选项，则将其从选项集中删除。按下 SHIFT 键，同时点击一个选项，可以选定一个选项范围，它从先前已选定的选项延伸到新选定的选项。

要找出当前的选项集，可以用 getSelectionPaths 方法来查询树：

```
TreePath[] selectedPaths = tree.getSelectionPaths();
```

如果想限制用户只能做单项选择，那么可以使用便捷的 getSelectionPath 方法，它将返回第一个被选择的路径，或者是 null（如果没有任何路径被选）。

**警告：** TreeSelectionEvent 类具有一个 getPaths 方法，它将返回一个 TreePath 对象数组，但是该数组描述的是选项集的变化，而不是当前的选项集。

程序清单 11-10 显示了类树这个程序的窗体类。该程序可以显示继承的层次结构，并且将抽象类定制显示为斜体字（参见程序清单 11-11 的单元格绘制器）。可以在窗体底部的文本框中输入任何类名，按下 Enter 键或者点击“Add”按钮，将该类及其超类添加到树中。必须输入完整的包名，例如 java.util.ArrayList。

#### 程序清单 11-10 treeRender/ClassTreeFrame.java

```
1 package treeRender;
2
3 import java.awt.*;
4 import java.awt.event.*;
5 import java.lang.reflect.*;
6 import java.util.*;
```

```

7
8 import javax.swing.*;
9 import javax.swing.tree.*;
10
11 /**
12  * This frame displays the class tree, a text field, and an "Add" button to add more classes
13  * into the tree.
14 */
15 public class ClassTreeFrame extends JFrame
16 {
17     private static final int DEFAULT_WIDTH = 400;
18     private static final int DEFAULT_HEIGHT = 300;
19
20     private DefaultMutableTreeNode root;
21     private DefaultTreeModel model;
22     private JTree tree;
23     private JTextField textField;
24     private JTextArea textArea;
25
26     public ClassTreeFrame()
27     {
28         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
29
30         // the root of the class tree is Object
31         root = new DefaultMutableTreeNode(java.lang.Object.class);
32         model = new DefaultTreeModel(root);
33         tree = new JTree(model);
34
35         // add this class to populate the tree with some data
36         addClass(getClass());
37
38         // set up node icons
39         var renderer = new ClassNameTreeCellRenderer();
40         renderer.setClosedIcon(new ImageIcon(getClass().getResource("red-ball.gif")));
41         renderer.setOpenIcon(new ImageIcon(getClass().getResource("yellow-ball.gif")));
42         renderer.setLeafIcon(new ImageIcon(getClass().getResource("blue-ball.gif")));
43         tree.setCellRenderer(renderer);
44
45         // set up selection mode
46         tree.addTreeSelectionListener(event ->
47         {
48             // the user selected a different node--update description
49             TreePath path = tree.getSelectionPath();
50             if (path == null) return;
51             var selectedNode = (DefaultMutableTreeNode) path.getLastPathComponent();
52             Class<?> c = (Class<?>) selectedNode.getUserObject();
53             String description = getFieldDescription(c);
54             textArea.setText(description);
55         });
56         int mode = TreeSelectionModel.SINGLE_TREE_SELECTION;
57         tree.getSelectionModel().setSelectionMode(mode);
58
59         // this text area holds the class description
60         textArea = new JTextArea();

```

```
61
62     // add tree and text area
63     var panel = new JPanel();
64     panel.setLayout(new GridLayout(1, 2));
65     panel.add(new JScrollPane(tree));
66     panel.add(new JScrollPane(textArea));
67
68     add(panel, BorderLayout.CENTER);
69
70     addTextField();
71 }
72
73 /**
74 * Add the text field and "Add" button to add a new class.
75 */
76 public void addTextField()
77 {
78     var panel = new JPanel();
79
80     ActionListener addListener = event ->
81     {
82         // add the class whose name is in the text field
83         try
84         {
85             String text = textField.getText();
86             addClass(Class.forName(text)); // clear text field to indicate success
87             textField.setText("");
88         }
89         catch (ClassNotFoundException e)
90         {
91             JOptionPane.showMessageDialog(null, "Class not found");
92         }
93     };
94
95     // new class names are typed into this text field
96     textField = new JTextField(20);
97     textField.addActionListener(addListener);
98     panel.add(textField);
99
100    var addButton = new JButton("Add");
101    addButton.addActionListener(addListener);
102    panel.add(addButton);
103
104    add(panel, BorderLayout.SOUTH);
105 }
106
107 /**
108 * Finds an object in the tree.
109 * @param obj the object to find
110 * @return the node containing the object or null if the object is not present in the tree
111 */
112 public DefaultMutableTreeNode findUserObject(Object obj)
113 {
114     // find the node containing a user object
```

```

115     var e = (Enumeration<TreeNode>) root.breadthFirstEnumeration();
116     while (e.hasMoreElements())
117     {
118         var node = (DefaultMutableTreeNode) e.nextElement();
119         if (node.getUserObject().equals(obj)) return node;
120     }
121     return null;
122 }
123
124 /**
125 * Adds a new class and any parent classes that aren't yet part of the tree.
126 * @param c the class to add
127 * @return the newly added node
128 */
129 public DefaultMutableTreeNode addClass(Class<?> c)
130 {
131     // add a new class to the tree
132
133     // skip non-class types
134     if (c.isInterface() || c.isPrimitive()) return null;
135
136     // if the class is already in the tree, return its node
137     DefaultMutableTreeNode node = findUserObject(c);
138     if (node != null) return node;
139
140     // class isn't present--first add class parent recursively
141
142     Class<?> s = c.getSuperclass();
143
144     DefaultMutableTreeNode parent;
145     if (s == null) parent = root;
146     else parent = addClass(s);
147
148     // add the class as a child to the parent
149     var newNode = new DefaultMutableTreeNode(c);
150     model.insertNodeInto(newNode, parent, parent.getChildCount());
151
152     // make node visible
153     var path = new TreePath(model.getPathToRoot(newNode));
154     tree.makeVisible(path);
155
156     return newNode;
157 }
158
159 /**
160 * Returns a description of the fields of a class.
161 * @param the class to be described
162 * @return a string containing all field types and names
163 */
164 public static String getFieldDescription(Class<?> c)
165 {
166     // use reflection to find types and names of fields
167     var r = new StringBuilder();
168     Field[] fields = c.getDeclaredFields();

```

```

169     for (int i = 0; i < fields.length; i++)
170     {
171         Field f = fields[i];
172         if ((f.getModifiers() & Modifier.STATIC) != 0) r.append("static ");
173         r.append(f.getType().getName());
174         r.append(" ");
175         r.append(f.getName());
176         r.append("\n");
177     }
178     return r.toString();
179 }
180 }
```

### 程序清单 11-11 treeRender/ClassNameTreeCellRenderer.java

```

1 package treeRender;
2
3 import java.awt.*;
4 import java.lang.reflect.*;
5 import javax.swing.*;
6 import javax.swing.tree.*;
7
8 /**
9  * This class renders a class name either in plain or italic. Abstract classes are italic.
10 */
11 public class ClassNameTreeCellRenderer extends DefaultTreeCellRenderer
12 {
13     private Font plainFont = null;
14     private Font italicFont = null;
15
16     public Component getTreeCellRendererComponent(JTree tree, Object value, boolean selected,
17             boolean expanded, boolean leaf, int row, boolean hasFocus)
18     {
19         super.getTreeCellRendererComponent(tree, value, selected, expanded, leaf,
20             row, hasFocus);
21         // get the user object
22         var node = (DefaultMutableTreeNode) value;
23         Class<?> c = (Class<?>) node.getUserObject();
24
25         // the first time, derive italic font from plain font
26         if (plainFont == null)
27         {
28             plainFont = getFont();
29             // the tree cell renderer is sometimes called with a label that has a null font
30             if (plainFont != null) italicFont = plainFont.deriveFont(Font.ITALIC);
31         }
32
33         // set font to italic if the class is abstract, plain otherwise
34         if ((c.getModifiers() & Modifier.ABSTRACT) == 0) setFont(plainFont);
35         else setFont(italicFont);
36         return this;
37     }
38 }
```

这个程序用到了一点小小的技巧，它是通过反射机制来构建这棵类树的。这项操作包含在 `addClass` 方法内。（细节倒不那么重要，在这个例子中，我们之所以使用类树，是因为继承树不需要怎么费劲地编码就能生成一棵丰满的树。如果想在自己的应用中显示树，那么你需要准备自己的层次结构数据的来源。）该方法使用广度优先的搜索算法，通过调用我们在前一节实现的 `findUserObject` 方法，来确定当前的类是否已经存在于树中。如果这个类还不存在于树中，那么我们将其超类添加到这棵树中，然后将新节点作为它的子节点，并使该节点成为可见的。

在选择树的一个节点时，右侧的文本域将填充为选中的类的属性。在窗体构造器中，限制用户只能进行单个选项的选择，并添加了一个树选择监听器。当调用 `valueChanged` 方法时，我们忽略它的事件参数，只向该树询问当前的选定路径。正如通常情况那样，我们必须获得路径中的最后一个节点，并且查看它的用户对象。然后调用 `getFieldDescription` 方法，该方法使用反射机制将所选类的所有属性组装成一个字符串。

#### API javax.swing.JTree 1.2

- `TreePath getSelectionPath()`
- `TreePath[] getSelectionPaths()`

返回第一个选定的路径，或者一个包含所有选定节点的数组。如果没有选定任何路径，这两个方法都返回为 `null`。

#### API javax.swing.event.TreeSelectionListener 1.2

- `void valueChanged(TreeSelectionEvent event)`

每当选定节点或撤销选定的时候，该方法就被调用。

#### API javax.swing.event.TreeSelectionEvent 1.2

- `TreePath getPath()`
- `TreePath[] getPaths()`

获取在该选择事件中已经发生更改的第一个路径或所有路径。如果你想知道当前的选择路径，而不是选择路径的更改情况，那么应该调用 `JTree.getSelectionPaths`。

### 11.2.5 定制树模型

在最后一个示例中，我们实现了一个能够查看变量内容的程序，正如调试器所做的那样（参见图 11-30）。

在继续深入之前，请先编译运行这个示例程序。其中每个节点对应于一个实例域。如果该域是一个对象，那么可以展开该节点以便查看它自己的实例域。该程序会审视窗体中的内容。如果你浏览了好几个实例域，那么你将会发现一些熟悉的类，还会对复杂的 Swing 用户界面构件有所了解。



图 11-30 对象查看树

该程序的不同之处在于它的树并没有使用 `DefaultTreeModel`。如果你已经拥有按照层次结构组织的数据，那么你可能并不想花精力去再创建一棵副本树，而且创建副本树还要担心怎样保持两棵树的一致性。这正是我们要讨论的情形：通过对象的引用，被审视的对象已经彼此连接起来了，因此在这里就不需要复制这种连接结构了。

`TreeModel` 接口只有几个方法。第一组方法使得 `JTree` 能够按照先是根节点，然后是子节点的顺序找到树中的节点。`JTree` 类只在用户真正展开一个节点的时候才会调用这些方法。

```
Object getRoot()
int getChildCount(Object parent)
Object getChild(Object parent, int index)
```

这个示例显示了为什么 `TreeModel` 接口像 `JTree` 类那样，不需要明确的用于描述节点的概念。根节点和子节点可以是任何对象，`TreeModel` 负责告知 `JTree` 它们是怎样联系起来的。

`TreeModel` 接口的下一个方法与 `getChild` 相反：

```
int getIndexofChild(Object parent, Object child)
```

实际上，这个方法可以用前面的三个方法实现，参见程序清单 11-12 中的代码。

### 程序清单 11-12 treeModel/ObjectInspectorFrame.java

```
1 package treeModel;
2
3 import java.awt.*;
4 import javax.swing.*;
5
6 /**
7  * This frame holds the object tree.
8 */
9 public class ObjectInspectorFrame extends JFrame
10 {
11     private JTree tree;
12     private static final int DEFAULT_WIDTH = 400;
13     private static final int DEFAULT_HEIGHT = 300;
14
15     public ObjectInspectorFrame()
16     {
17         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
18
19         // we inspect this frame object
20
21         var v = new Variable(getClass(), "this", this);
22         var model = new ObjectTreeModel();
23         model.setRoot(v);
24
25         // construct and show tree
26
27         tree = new JTree(model);
28         add(new JScrollPane(tree), BorderLayout.CENTER);
29     }
30 }
```

树模型会告诉 JTree 哪些节点应该显示成叶节点：

```
boolean isLeaf(Object node)
```

如果你的代码更改了树模型，那么必须告知这棵树以便它能够对自己进行重新绘制。树是将它自己作为一个 TreeModelListener 添加到模型中的，因此，模型必须支持通常的监听器管理方法：

```
void addTreeModelListener(TreeModelListener l)
void removeTreeModelListener(TreeModelListener l)
```

可以在程序清单 11-13 中看到这些方法的具体实现。

### 程序清单 11-13 treeModel/ObjectTreeModel.java

```
1 package treeModel;
2
3 import java.lang.reflect.*;
4 import java.util.*;
5 import javax.swing.event.*;
6 import javax.swing.tree.*;
7
8 /**
9  * This tree model describes the tree structure of a Java object. Children are the objects
10 * that are stored in instance variables.
11 */
12 public class ObjectTreeModel implements TreeModel
13 {
14     private Variable root;
15     private EventListenerList listenerList = new EventListenerList();
16
17     /**
18      * Constructs an empty tree.
19      */
20     public ObjectTreeModel()
21     {
22         root = null;
23     }
24
25     /**
26      * Sets the root to a given variable.
27      * @param v the variable that is being described by this tree
28      */
29     public void setRoot(Variable v)
30     {
31         Variable oldRoot = v;
32         root = v;
33         fireTreeStructureChanged(oldRoot);
34     }
35
36     public Object getRoot()
37     {
38         return root;
39     }
40 }
```

```
41 public int getChildCount(Object parent)
42 {
43     return ((Variable) parent).getFields().size();
44 }
45
46 public Object getChild(Object parent, int index)
47 {
48     ArrayList<Field> fields = ((Variable) parent).getFields();
49     var f = (Field) fields.get(index);
50     Object parentValue = ((Variable) parent).getValue();
51     try
52     {
53         return new Variable(f.getType(), f.getName(), f.get(parentValue));
54     }
55     catch (IllegalAccessException e)
56     {
57         return null;
58     }
59 }
60
61 public int getIndexofChild(Object parent, Object child)
62 {
63     int n = getChildCount(parent);
64     for (int i = 0; i < n; i++)
65         if (getChild(parent, i).equals(child)) return i;
66     return -1;
67 }
68
69 public boolean isLeaf(Object node)
70 {
71     return getChildCount(node) == 0;
72 }
73
74 public void valueForPathChanged(TreePath path, Object newValue)
75 {
76 }
77
78 public void addTreeModelListener(TreeModelListener l)
79 {
80     listenerList.add(TreeModelListener.class, l);
81 }
82
83 public void removeTreeModelListener(TreeModelListener l)
84 {
85     listenerList.remove(TreeModelListener.class, l);
86 }
87
88 protected void fireTreeStructureChanged(Object oldRoot)
89 {
90     var event = new TreeModelEvent(this, new Object[] { oldRoot });
91     for (TreeModelListener l : listenerList.getListeners(TreeModelListener.class))
92         l.treeStructureChanged(event);
93 }
94 }
```

当模型修改了树的内容时，它会调用 TreeModelListener 接口中下面 4 个方法中的某一个：

```
void treeNodesChanged(TreeModelEvent e)
void treeNodesInserted(TreeModelEvent e)
void treeNodesRemoved(TreeModelEvent e)
void treeStructureChanged(TreeModelEvent e)
```

TreeModelEvent 对象用于描述修改的位置。对描述插入或移除事件的树模型事件进行组装的细节是相当技术性的。如果树中确实有要添加或移除的节点，只需要考虑如何触发这些事件。在程序清单 11-12 中，我们展示了怎么触发一个事件：将根节点替换为一个新的对象。

 提示：为了简化事件触发的代码，我们使用了 javax.swing.EventListenerList 这个使用方便、能够收集监听器的类。程序清单 11-13 中最后 3 个方法展示了如何使用这个类。

最后，如果用户要编辑树节点，那么模型会随着这种修改而被调用：

```
void valueForPathChanged(TreePath path, Object newValue)
```

如果不允许编辑，则永远不会调用到该方法。

如果不支持编辑功能，那么构建一个树模型就变得相当容易了。我们要实现下面 3 个方法：

```
Object getRoot()
int getChildCount(Object parent)
Object getChild(Object parent, int index)
```

这 3 个方法用于描述树的结构。还要提供另外 5 个方法的常规实现，如程序清单 10-16 那样，然后就可以准备显示你的树了。

现在让我们转向示例程序的具体实现，我们的树将包含类型为 Variable 的对象。

 注释：一旦使用了 DefaultTreeModel，我们的节点就可以具有类型为 DefaultMutableTreeNode、用户对象类型为 Variable 的对象。

例如，假设我们查看下面这个变量

```
Employee joe;
```

该变量的类型为 Employee.class，名字为 joe，值为对象引用 joe 的值。在程序清单 11-14 中，我们定义了 Variable 这个类，用来描述程序中的变量：

```
var v = new Variable(Employee.class, "joe", joe);
```

#### 程序清单 11-14 treeModel/Variable.java

```
1 package treeModel;
2
3 import java.lang.reflect.*;
4 import java.util.*;
5
6 /**
7  * A variable with a type, name, and value.
8 */
```

```
9 public class Variable
10 {
11     private Class<?> type;
12     private String name;
13     private Object value;
14     private ArrayList<Field> fields;
15
16     /**
17      * Construct a variable.
18      * @param aType the type
19      * @param aName the name
20      * @param aValue the value
21      */
22     public Variable(Class<?> aType, String aName, Object aValue)
23     {
24         type = aType;
25         name = aName;
26         value = aValue;
27         fields = new ArrayList<>();
28
29         // find all fields if we have a class type except we don't expand strings and
30         // null values
31
32         if (!type.isPrimitive() && !type.isArray() && !type.equals(String.class)
33             && value != null)
34         {
35             // get fields from the class and all superclasses
36             for (Class<?> c = value.getClass(); c != null; c = c.getSuperclass())
37             {
38                 Field[] fs = c.getDeclaredFields();
39                 AccessibleObject.setAccessible(fs, true);
40
41                 // get all nonstatic fields
42                 for (Field f : fs)
43                     if ((f.getModifiers() & Modifier.STATIC) == 0) fields.add(f);
44             }
45         }
46     }
47
48     /**
49      * Gets the value of this variable.
50      * @return the value
51      */
52     public Object getValue()
53     {
54         return value;
55     }
56
57     /**
58      * Gets all nonstatic fields of this variable.
59      * @return an array list of variables describing the fields
60      */
61     public ArrayList<Field> getFields()
62     {
```

```

63     return fields;
64 }
65
66 public String toString()
67 {
68     String r = type + " " + name;
69     if (type.isPrimitive()) r += "=" + value;
70     else if (type.equals(String.class)) r += "=" + value;
71     else if (value == null) r += "=null";
72     return r;
73 }
74 }
```

如果该变量的类型为基本类型，必须为这个值使用对象包装器。

```
new Variable(double.class, "salary", new Double(salary));
```

如果变量的类型是一个类，那么该变量就会拥有一些域。使用反射机制可以将所有域枚举出来，并将它们收集存放到一个 `ArrayList` 中。因为 `Class` 类的 `getFields` 方法不返回超类的任何域，因此还必须调用超类中的 `getFields` 方法，可以在 `Variable` 构造器中找到这些代码。`Variable` 类的 `getFields` 方法将返回一个包含了各类域的一个数组。最后，`Variable` 类的 `toString` 方法将节点格式化为标签，这个标签通常包含变量的类型和名称。如果变量不是一个类，那么该标签还将包含变量的值。

**注释：**如果类型是一个数组，那么我们不会显示数组中的元素。这并不难实现，因此我们就把它留作众所周知的“读者练习”了。

让我们继续介绍树模型，头两个方法很简单。

```

public Object getRoot()
{
    return root;
}

public int getChildCount(Object parent)
{
    return ((Variable) parent).getFields().size();
}
```

`getChild` 方法返回一个新的 `Variable` 对象，用于描述给定索引位置上的域。`Field` 类的 `getType` 方法和 `getName` 方法用于产生域的类型和名称。通过使用反射机制，就可以按照 `f.get(parentValue)` 这种方式读取域的值。该方法可以抛出一个异常 `IllegalAccessException`，不过，我们可以让所有域在 `Variable` 构造器中都是可访问的，这样在实际应用中就不会发生这种抛出异常的情况。

下面是 `getChild` 方法的完整代码。

```

public Object getChild(Object parent, int index)
{
    ArrayList fields = ((Variable) parent).getFields();
    var f = (Field) fields.get(index);
```

```

Object parentValue = ((Variable) parent).getValue();
try
{
    return new Variable(f.getType(), f.getName(), f.get(parentValue));
}
catch (IllegalAccessException e)
{
    return null;
}
}
}

```

这 3 个方法展示了对象树到 JTree 构件之间的结构，其余的方法是一些常规方法，源代码请见程序清单 11-13。

关于该树模型，有一个不同寻常之处：它实际上描述的是一棵无限树。可以通过追踪 WeakReference 对象来证实这一点。当你点击名字为 referent 的变量时，它会引导你回到初始的对象。你将获得一棵相同的子树，并且可以再次展开它的 WeakReference 对象，周而复始，无穷无尽。当然，你无法存储一个无限的节点集合。树模型只是在用户展开父节点时，按照需要来产生这些节点。

程序清单 11-12 展示了样例程序的框体类。

#### API javax.swing.tree.TreeModel 1.2

- `Object getRoot()`  
返回根节点。
- `int getChildCount(Object parent)`  
获取 parent 节点的子节点个数。
- `Object getChild(Object parent, int index)`  
获取给定索引位置上 parent 节点的子节点。
- `int getIndexOfChild(Object parent, Object child)`  
获取 parent 节点的子节点 child 的索引位置。如果在树模型中 child 节点不是 parent 的一个子节点，则返回 -1。
- `boolean isLeaf(Object node)`  
如果节点 node 从概念上讲是一个叶节点，则返回 true。
- `void addTreeModelListener(TreeModelListener l)`
- `void removeTreeModelListener(TreeModelListener l)`  
当模型中的信息发生变化时，告知添加和移除监听器。
- `void valueForPathChanged(TreePath path, Object newValue)`  
当一个单元格编辑器修改了节点值的时候，该方法被调用。  
参数：path 到被编辑节点的树路径  
newValue 编辑器返回的修改值

#### API javax.swing.event.TreeModelListener 1.2

- `void treeNodesChanged(TreeModelEvent e)`

- void treeNodesInserted(TreeModelEvent e)
- void treeNodesRemoved(TreeModelEvent e)
- void treeStructureChanged(TreeModelEvent e)

如果树被修改过，树模型将调用该方法。

#### **API** javax.swing.event.TreeModelEvent 1.2

- TreeModelEvent(Object eventSource, TreePath node)

构建一个树模型事件。

参数：eventSource 产生该事件的树模型

node 到达要修改节点的树路径

## 11.3 高级 AWT

Graphics 类有多种方法可以用来创建简单的图形。这些方法对于简单的 applet 和应用来说已经绰绰有余了，但是当你创建复杂的图形或者需要全面控制图形的外观时，它们就显得力不从心了。Java 2D API 是一个更加成熟的类库，可以用它产生高质量的图形。下面我们将概要地介绍一下该 API。

### 11.3.1 绘图操作流程

在最初的 JDK 1.0 中，用来绘制形状的是一种非常简单的机制，即选择颜色和画图的模式，并调用 Graphics 类的各种方法，比如 drawRect 或者 fillOval。而 Java 2D API 支持更多的功能：

- 可以很容易地绘制各式各样的形状。
- 可以控制绘制形状的笔画，即控制跟踪形状边界的绘图笔。
- 可以用单色、变化的色调和重复的模式来填充各种形状。
- 可以使用变换法，对各种形状进行移动、缩放、旋转和拉伸。
- 可以对形状进行剪切，将其限制在任意的区域内。
- 可以选择各种组合规则，来描述如何将新形状的像素与现有的像素组合起来。
- 可以提供绘制图形提示，以便在速度与绘图质量之间实现平衡。

如果要绘制一个形状，可以按照如下步骤操作：

1. 获得一个 Graphics2D 类的对象，该类是 Graphics 类的子类。自 Java SE 1.2 以来，paint 和 paintComponent 等方法就能够自动地接收一个 Graphics2D 类的对象，这时可以直接使用如下的转型：

```
public void paintComponent(Graphics g)
{
    var g2 = (Graphics2D) g;
    ...
}
```

2. 使用 setRenderingHints 方法来设置绘图提示，它提供了速度与绘图质量之间的一种平衡。

```
RenderingHints hints = . . .;
g2.setRenderingHints(hints);
```

3. 使用 `setStroke` 方法来设置笔画，笔画用于绘制形状的边框。可以选择边框的粗细和线段的虚实。

```
Stroke stroke = . . .;
g2.setStroke(stroke);
```

4. 使用 `setPaint` 方法来设置着色法，着色法用于填充诸如笔画路径或者形状内部等区域的颜色。可以创建单色、渐变色或者平铺的填充模式。

```
Paint paint = . . .;
g2.setPaint(paint);
```

5. 使用 `clip` 方法来设置剪切区域。

```
Shape clip = . . .;
g2.clip(clip);
```

6. 使用 `transform` 方法设置一个从用户空间到设备空间的变换方式。如果使用变换方式比使用像素坐标更容易定义在定制坐标系统中的形状，那么就可以使用变换方式。

```
AffineTransform transform = . . .;
g2.transform(transform);
```

7. 使用 `setComposite` 方法设置一个组合规则，用来描述如何将新像素与现有的像素组合起来。

```
Composite composite = . . .;
g2.setComposite(composite);
```

8. 建立一个形状，Java 2D API 提供了用来组合各种形状的许多形状对象和方法。

```
Shape shape = . . .;
```

9. 绘制或者填充该形状。如果要绘制该形状，那么它的边框就会用笔画画出来。如果要填充该形状，那么它的内部就会被着色。

```
g2.draw(shape);
g2.fill(shape);
```

当然，在许多实际的环境中，并不需要采用所有这些操作步骤。Java 2D 图形上下文中有关合理的默认设置。只有当你确实想要改变设置时，再去修改这些默认设置。

在下面的几节中，我们将要介绍如何描绘形状、笔画、着色、变换及组合的规则。

各种不同的 `set` 方法只是用于设置 2D 图形上下文的状态，它们并不进行任何实际的绘图操作。同样，在构建 `shape` 对象时，也不进行任何绘图操作。只有在调用 `draw` 或者 `fill` 方法时，才会绘制出图形的形状，而就在此刻，这个新的图形由绘图操作流程计算出来（参见图 11-31）。

在绘图流程中，需要以下这些操作步骤来绘制一个形状：

1. 用笔画画出形状的线条；
2. 对形状进行变换操作；

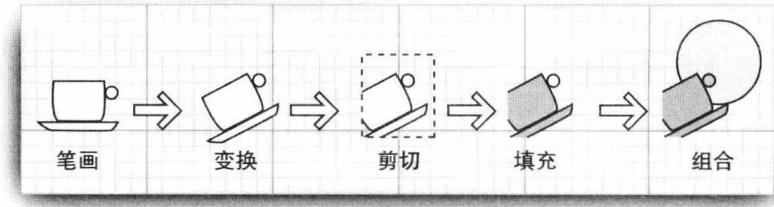


图 11-31 绘图操作流程

3. 对形状进行剪切。如果形状与剪切区域之间没有任何相交的地方，那么就不用执行该操作；

4. 对剪切后的形状进行填充；
5. 把填充后的形状与已有的形状进行组合（在图 11-31 中，圆形是已有像素部分，杯子的形状加在了它的上面）。

在下一节中，将会讲述如何对形状进行定义。然后，我们将转而对 2D 图形上下文设置进行介绍。

#### **API** java.awt.Graphics2D 1.2

- void draw(Shape s)

用当前的笔画来绘制给定形状的边框。

- void fill(Shape s)

用当前的着色方案来填充给定形状的内部。

### 11.3.2 形状

下面是 Graphics 类中绘制形状的若干方法：

```
drawLine
drawRectangle
drawRoundRect
draw3DRect
drawPolygon
drawPolyline
drawOval
drawArc
```

它们还有对应的 fill 方法，这些方法从 JDK 1.0 起就被纳入到 Graphics 类中了。Java 2D API 使用了一套完全不同的面向对象的处理方法，即不再使用方法，而是使用下面的这些类：

```
Line2D
Rectangle2D
RoundRectangle2D
Ellipse2D
Arc2D
QuadCurve2D
```

CubicCurve2D  
GeneralPath

这些类全部都实现了 Shape 接口，我们将在下面各小节中一一审视它们。

### 11.3.2.1 形状类层次结构

Line2D、Rectangle2D、RoundRectangle2D、Ellipse2D 和 Arc2D 等这些类对应于 drawLine、drawRectangle、drawRoundRect、drawOval 和 drawArc 等方法。（“3D 矩形”的概念已经理所当然地过时了，因而没有与 draw3DRect 方法相对应的类。）Java 2D API 提供了两个补充类，即二次曲线类和三次曲线类。我们将在本节的后面部分阐释这些形状。Java 2D API 中没有任何 Polygon2D 类。相反，它用 GeneralPath 类来描述由线条、二次曲线、三次曲线构成的线条路径。可以使用 GeneralPath 来描述一个多边形；我们将在本节的后面部分对它进行介绍。

如果要绘制一个形状，首先要创建一个实现了 Shape 接口的类的对象，然后调用 Graphics2D 类的 draw 方法。

下面这些类：

Rectangle2D  
RoundRectangle2D  
Ellipse2D  
Arc2D

都是从一个公共超类 RectangularShape 继承而来的。诚然，椭圆形和弧形都不是矩形，但是它们都有一个矩形的边界框（参见图 11-32）。

名字以“2D”结尾的每个类都有两个子类，用于指定坐标是 float 类型的还是 double 类型的。在本书的卷 I 中，我们已经介绍了 Rectangle2D.Float 和 Rectangle2D.Double。

其他类也使用了相同的模式，比如 Arc2D.Float 和 Arc2D.Double。

从内部来讲，所有的图形类使用的都是 float 类型的坐标，因为 float 类型的数占用较少的存储空间，而且它们有足够高的几何计算精度。然而，Java 编程语言使得对 float 类型的数的操作要稍微复杂些。由于这个原因，图形类的大多数方法使用的都是 double 类型的参数和返回值。只有在创建一个 2D 对象的时候，才需要选择究竟是使用带有 float 类型坐标的构造器，还是使用带有 double 类型坐标的构造器。例如：

```
var floatRect = new Rectangle2D.Float(5F, 10F, 7.5F, 15F);
var doubleRect = new Rectangle2D.Double(5, 10, 7.5, 15);
```

Xxx2D.Float 和 Xxx2D.Double 两个类都是 Xxx2D 类的子类，在对象被构建之后，再记住其确切的子类型实质上已经没有任何额外的好处了，因此可以将刚被构建的对象存储为一个超类变量，正如上面代码示例中所阐释的那样。

从这些类古怪的名字中就可以判断出，Xxx2D.Float 和 Xxx2D.Double 两个类同时也是 Xxx2D

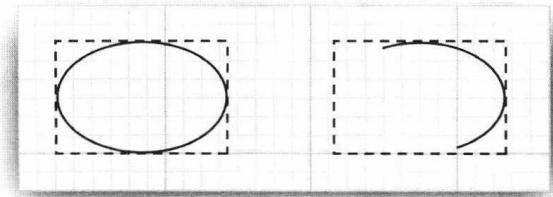


图 11-32 椭圆形和弧形的矩形边界框

类的内部类。这只是为了在语法上比较方便，以避免外部类的名字变得太长。

最后，还有一个 Point2D 类，它用 x 和 y 坐标来描述一个点。点对于定义形状非常有用，不过它们本身并不是形状。

图 11-33 显示了各个形状类之间的关系。不过图中省略了 Double 和 Float 子类，并且来自以前的 2D 类库的遗留类用灰色的填充色标识。

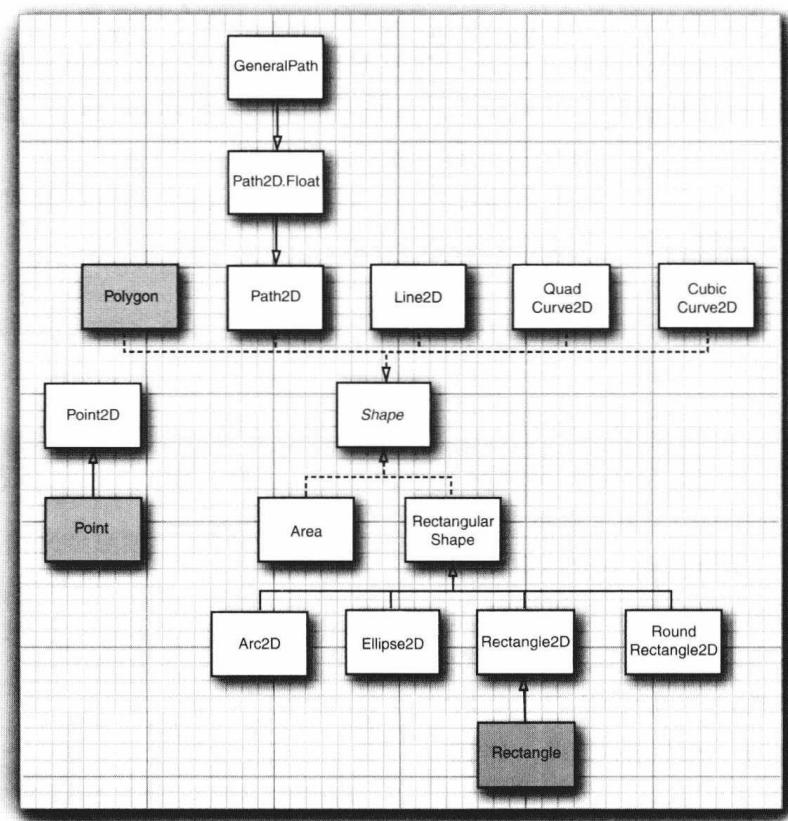


图 11-33 形状类之间的关系

### 11.3.2.2 使用形状类

我们在本书的卷 I 第 10 章中介绍了如何使用 Rectangle2D、Ellipse2D 和 Line2D 类的方法。本节将介绍如何建立其他的 2D 形状。

如果要建立一个 RoundRectangle2D 形状，应该设定左上角、宽度、高度及应该变成圆角的边角区的 x 和 y 的坐标尺寸（参见图 11-34）。例如，调用下面的方法：

```
var r = new RoundRectangle2D.Double(150, 200, 100, 50, 20, 20);
```

便产生了一个带圆角的矩形，每个角的圆半径为 20。

如果要建立一个弧形，首先应该设定边界框，接着设定它的起始角度和弧形跨越的角度（见图 11-35），并且设定弧形闭合的类型，即 Arc2D.OPEN、Arc2D.PIE 或者 Arc2D.CHORD 这几种类型中的一个。

```
var a = new Arc2D(x, y, width, height, startAngle, arcAngle, closureType);
```

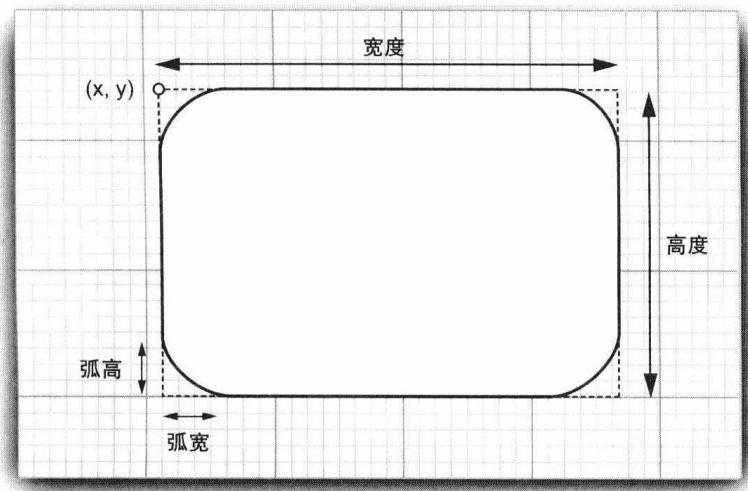


图 11-34 构建一个 RoundRectangle2D

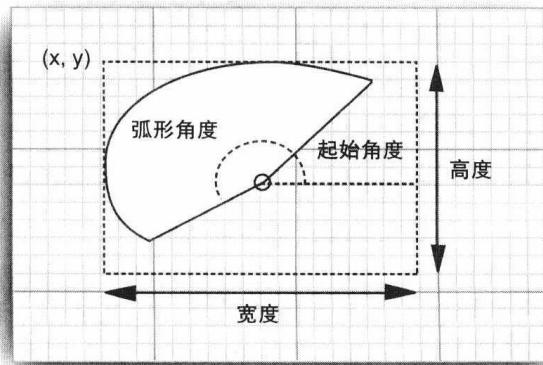


图 11-35 构建一个椭圆弧形

图 11-36 显示了几种弧形的类型。

**！ 警告：**如果弧形是椭圆的，那么弧形角的计算就不是很直接了。API 文档中描述到：“角是相对于非正方形的矩形边框指定的，以使得 45 度总是落到了从椭圆中心指向矩形边框右上角的方向上。因此，如果矩形边框的一条轴比另一条轴明显长许多，那么弧形段的起始点和终止点就会与边框中的长轴斜交。”但是，文档中并没有说明如何计算这种“斜交”。下面是其细节：

假设弧形的中心是原点，而且点  $(x, y)$  在弧形上。那么我们可以用下面的公式来获得这个斜交角：

```
skewedAngle = Math.toDegrees(Math.atan2(-y * height, x * width));
```

这个值介于 -180 到 180 之间。按照这种方式计算斜交的起始角和终止角，然后，计

算两个斜交角之间的差，如果起始角或角的差是负数，则加上 360。之后，将起始角和角的差提供给弧形的构造器。如果运行本节末尾的程序，你用肉眼就能观察到这种计算所产生的用于弧形构造器的值是正确的。可参见本章图 11-39。

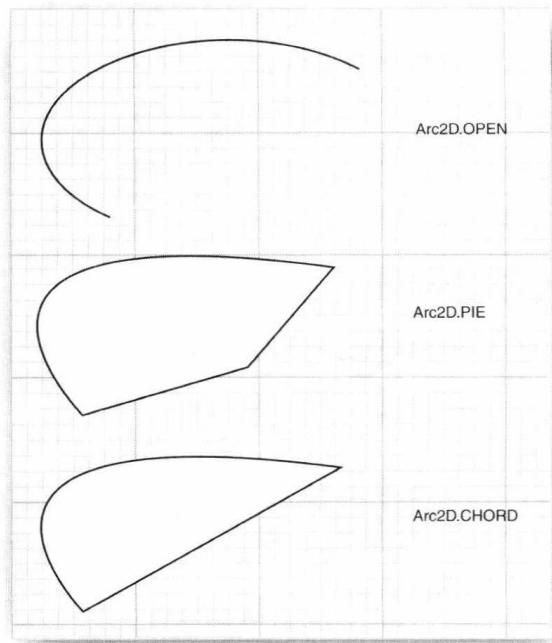


图 11-36 弧形的类型

Java 2D API 提供了对二次曲线和三次曲线的支持。在本章中，我们并不会深入介绍这些曲线的数学特征。我们建议你通过运行程序清单 11-1 的代码，对曲线的形状有一个感性的认识。正如在图 11-37 和图 11-38 中看到的那样，二次曲线和三次曲线是由两个端点和一个或两个控制点来设定的。移动控制点，曲线的形状就会改变。

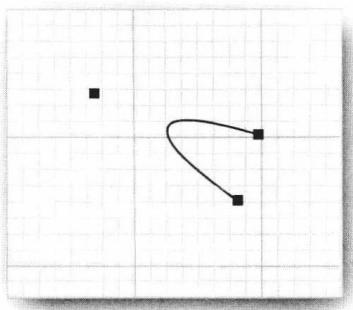


图 11-37 二次曲线

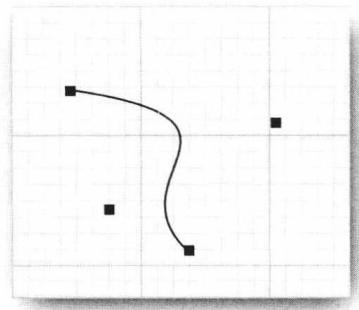


图 11-38 三次曲线

如果要构建二次曲线和三次曲线，需要给出两个端点和控制点的坐标。例如，

```
var q = new QuadCurve2D.Double(startX, startY, controlX, controlY, endX, endY);
var c = new CubicCurve2D.Double(startX, startY, control1X, control1Y,
    control2X, control2Y, endX, endY);
```

二次曲线不是非常灵活，所以实际上它并不常用。三次曲线（比如用 `CubicCurve2D` 类绘制的贝塞尔（Bézier）曲线）却是非常常用的。通过将三次曲线组合起来，使得连接点的各个斜率互相匹配，就能够创建复杂的、外观平滑的曲线形状。如果要了解这方面的详细信息，请参阅 James D. Foley、Andries van Dam 和 Steven K. Feiner 等人合作撰写的 *Computer Graphics: Principles and Practice*（第 3 版），Addison Wesley 出版社 2013 年出版。

可以建立线段、二次曲线和三次曲线的任意序列，并把它们存放到一个 `GeneralPath` 对象中去。可以用 `moveTo` 方法来指定路径的第一个坐标，例如，

```
var path = new GeneralPath();
path.moveTo(10, 20);
```

然后，可以通过调用 `lineTo`、`quadTo` 或 `curveTo` 三种方法之一来扩展路径，这些方法分别用线条、二次曲线或者三次曲线来扩展路径。如果要调用 `lineTo` 方法，需要提供它的端点。而对两个曲线方法的调用，应该先提供控制点，然后提供端点。例如，

```
path.lineTo(20, 30);
path.curveTo(control1X, control1Y, control2X, control2Y, endX, endY);
```

可以调用 `closePath` 方法来闭合路径，它能够绘制一条回到路径起始点的线条。

如果要绘制一个多边形，只需调用 `moveTo` 方法，以到达第一个拐角点，然后反复调用 `lineTo` 方法，以便到达其他的拐角点。最后调用 `closePath` 方法来闭合多边形。程序清单 11-15 更加详细地展示了构建多边形的方法。

### 程序清单 11-15 shape/ShapeTest.java

```
1 package shape;
2
3 import java.awt.*;
4 import java.awt.event.*;
5 import java.awt.geom.*;
6 import java.util.*;
7 import javax.swing.*;
8
9 /**
10  * This program demonstrates the various 2D shapes.
11  * @version 1.04 2018-05-01
12  * @author Cay Horstmann
13  */
14 public class ShapeTest
15 {
16     public static void main(String[] args)
17     {
18         EventQueue.invokeLater(() ->
19         {
20             var frame = new ShapeTestFrame();
21             frame.setTitle("ShapeTest");
```

```

22         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23         frame.setVisible(true);
24     });
25 }
26 }
27 */
28 /**
29 * This frame contains a combo box to select a shape and a component to draw it.
30 */
31 class ShapeTestFrame extends JFrame
32 {
33     public ShapeTestFrame()
34     {
35         var comp = new ShapeComponent();
36         add(comp, BorderLayout.CENTER);
37         var comboBox = new JComboBox<ShapeMaker>();
38         comboBox.addItem(new LineMaker());
39         comboBox.addItem(new RectangleMaker());
40         comboBox.addItem(new RoundRectangleMaker());
41         comboBox.addItem(new EllipseMaker());
42         comboBox.addItem(new ArcMaker());
43         comboBox.addItem(new PolygonMaker());
44         comboBox.addItem(new QuadCurveMaker());
45         comboBox.addItem(new CubicCurveMaker());
46         comboBox.addActionListener(event ->
47         {
48             ShapeMaker shapeMaker = comboBox.getItemAt(comboBox.getSelectedIndex());
49             comp.setShapeMaker(shapeMaker);
50         });
51         add(comboBox, BorderLayout.NORTH);
52         comp.setShapeMaker((ShapeMaker) comboBox.getItemAt(0));
53         pack();
54     }
55 }
56 */
57 /**
58 * This component draws a shape and allows the user to move the points that define it.
59 */
60 class ShapeComponent extends JComponent
61 {
62     private static final Dimension PREFERRED_SIZE = new Dimension(300, 200);
63     private Point2D[] points;
64     private static Random generator = new Random();
65     private static int SIZE = 10;
66     private int current;
67     private ShapeMaker shapeMaker;
68
69     public ShapeComponent()
70     {
71         addMouseListener(new MouseAdapter()
72         {
73             public void mousePressed(MouseEvent event)
74             {
75                 Point p = event.getPoint();

```

```
76     for (int i = 0; i < points.length; i++)
77     {
78         double x = points[i].getX() - SIZE / 2;
79         double y = points[i].getY() - SIZE / 2;
80         var r = new Rectangle2D.Double(x, y, SIZE, SIZE);
81         if (r.contains(p))
82         {
83             current = i;
84             return;
85         }
86     }
87 }
88
89     public void mouseReleased(MouseEvent event)
90     {
91         current = -1;
92     }
93 });
94 addMouseMotionListener(new MouseMotionAdapter()
95 {
96     public void mouseDragged(MouseEvent event)
97     {
98         if (current == -1) return;
99         points[current] = event.getPoint();
100        repaint();
101    }
102 });
103 current = -1;
104 }
105
106 /**
107 * Set a shape maker and initialize it with a random point set.
108 * @param aShapeMaker a shape maker that defines a shape from a point set
109 */
110 public void setShapeMaker(ShapeMaker aShapeMaker)
111 {
112     shapeMaker = aShapeMaker;
113     int n = shapeMaker.getPointCount();
114     points = new Point2D[n];
115     for (int i = 0; i < n; i++)
116     {
117         double x = generator.nextDouble() * getWidth();
118         double y = generator.nextDouble() * getHeight();
119         points[i] = new Point2D.Double(x, y);
120     }
121     repaint();
122 }
123
124 public void paintComponent(Graphics g)
125 {
126     if (points == null) return;
127     var g2 = (Graphics2D) g;
128     for (int i = 0; i < points.length; i++)
129     {
```

```
130         double x = points[i].getX() - SIZE / 2;
131         double y = points[i].getY() - SIZE / 2;
132         g2.fill(new Rectangle2D.Double(x, y, SIZE, SIZE));
133     }
134
135     g2.draw(shapeMaker.makeShape(points));
136 }
137
138 public Dimension getPreferredSize() { return PREFERRED_SIZE; }
139 }
140
141 /**
142 * A shape maker can make a shape from a point set. Concrete subclasses must return a shape in
143 * the makeShape method.
144 */
145 abstract class ShapeMaker
146 {
147     private int pointCount;
148
149     /**
150      * Constructs a shape maker.
151      * @param pointCount the number of points needed to define this shape
152      */
153     public ShapeMaker(int pointCount)
154     {
155         this.pointCount = pointCount;
156     }
157
158     /**
159      * Gets the number of points needed to define this shape.
160      * @return the point count
161      */
162     public int getPointCount()
163     {
164         return pointCount;
165     }
166
167     /**
168      * Makes a shape out of the given point set.
169      * @param p the points that define the shape
170      * @return the shape defined by the points
171      */
172     public abstract Shape makeShape(Point2D[] p);
173
174     public String toString()
175     {
176         return getClass().getName();
177     }
178 }
179
180 /**
181 * Makes a line that joins two given points.
182 */
183 class LineMaker extends ShapeMaker
```

```
184 {
185     public LineMaker()
186     {
187         super(2);
188     }
189
190     public Shape makeShape(Point2D[] p)
191     {
192         return new Line2D.Double(p[0], p[1]);
193     }
194 }
195
196 /**
197 * Makes a rectangle that joins two given corner points.
198 */
199 class RectangleMaker extends ShapeMaker
200 {
201     public RectangleMaker()
202     {
203         super(2);
204     }
205
206     public Shape makeShape(Point2D[] p)
207     {
208         var s = new Rectangle2D.Double();
209         s setFrameFromDiagonal(p[0], p[1]);
210         return s;
211     }
212 }
213
214 /**
215 * Makes a round rectangle that joins two given corner points.
216 */
217 class RoundRectangleMaker extends ShapeMaker
218 {
219     public RoundRectangleMaker()
220     {
221         super(2);
222     }
223
224     public Shape makeShape(Point2D[] p)
225     {
226         var s = new RoundRectangle2D.Double(0, 0, 0, 0, 20, 20);
227         s setFrameFromDiagonal(p[0], p[1]);
228         return s;
229     }
230 }
231
232 /**
233 * Makes an ellipse contained in a bounding box with two given corner points.
234 */
235 class EllipseMaker extends ShapeMaker
236 {
237     public EllipseMaker()
```

```

238     {
239         super(2);
240     }
241
242     public Shape makeShape(Point2D[] p)
243     {
244         var s = new Ellipse2D.Double();
245         s setFrameFromDiagonal(p[0], p[1]);
246         return s;
247     }
248 }
249
250 /**
251 * Makes an arc contained in a bounding box with two given corner points, and with starting
252 * and ending angles given by lines emanating from the center of the bounding box and ending
253 * in two given points. To show the correctness of the angle computation, the returned shape
254 * contains the arc, the bounding box, and the lines.
255 */
256 class ArcMaker extends ShapeMaker
257 {
258     public ArcMaker()
259     {
260         super(4);
261     }
262
263     public Shape makeShape(Point2D[] p)
264     {
265         double centerX = (p[0].getX() + p[1].getX()) / 2;
266         double centerY = (p[0].getY() + p[1].getY()) / 2;
267         double width = Math.abs(p[1].getX() - p[0].getX());
268         double height = Math.abs(p[1].getY() - p[0].getY());
269
270         double skewedStartAngle = Math.toDegrees(Math.atan2(-(p[2].getY() - centerY) * width,
271             (p[2].getX() - centerX) * height));
272         double skewedEndAngle = Math.toDegrees(Math.atan2(-(p[3].getY() - centerY) * width,
273             (p[3].getX() - centerX) * height));
274         double skewedAngleDifference = skewedEndAngle - skewedStartAngle;
275         if (skewedStartAngle < 0) skewedStartAngle += 360;
276         if (skewedAngleDifference < 0) skewedAngleDifference += 360;
277
278         var s = new Arc2D.Double(0, 0, 0, 0,
279             skewedStartAngle, skewedAngleDifference, Arc2D.OPEN);
280         s setFrameFromDiagonal(p[0], p[1]);
281
282         var g = new GeneralPath();
283         g.append(s, false);
284         var r = new Rectangle2D.Double();
285         r setFrameFromDiagonal(p[0], p[1]);
286         g.append(r, false);
287         var center = new Point2D.Double(centerX, centerY);
288         g.append(new Line2D.Double(center, p[2]), false);
289         g.append(new Line2D.Double(center, p[3]), false);
290         return g;
291     }

```

```
292 }
293 /**
294 * Makes a polygon defined by six corner points.
295 */
296 class PolygonMaker extends ShapeMaker
297 {
298     public PolygonMaker()
299     {
300         super(6);
301     }
302
303     public Shape makeShape(Point2D[] p)
304     {
305         var s = new GeneralPath();
306         s.moveTo((float) p[0].getX(), (float) p[0].getY());
307         for (int i = 1; i < p.length; i++)
308             s.lineTo((float) p[i].getX(), (float) p[i].getY());
309         s.closePath();
310         return s;
311     }
312 }
313 }
314 /**
315 * Makes a quad curve defined by two end points and a control point.
316 */
317 class QuadCurveMaker extends ShapeMaker
318 {
319     public QuadCurveMaker()
320     {
321         super(3);
322     }
323
324     public Shape makeShape(Point2D[] p)
325     {
326         return new QuadCurve2D.Double(p[0].getX(), p[0].getY(), p[1].getX(), p[1].getY(),
327             p[2].getX(), p[2].getY());
328     }
329 }
330 }
331 /**
332 * Makes a cubic curve defined by two end points and two control points.
333 */
334 class CubicCurveMaker extends ShapeMaker
335 {
336     public CubicCurveMaker()
337     {
338         super(4);
339     }
340
341     public Shape makeShape(Point2D[] p)
342     {
343         return new CubicCurve2D.Double(p[0].getX(), p[0].getY(), p[1].getX(), p[1].getY(),
344             p[2].getX(), p[2].getY(), p[3].getX(), p[3].getY());
345     }
346 }
```

```
346     }
347 }
```

普通路径没有必要一定要连接在一起，我们随时可以调用 `moveTo` 方法来建立一个新的路径段。

最后，可以使用 `append` 方法，向普通路径添加任意个 `Shape` 对象。如果新建的形状应该连接到路径的最后一个端点，那么 `append` 方法的第二个参数值就是 `true`，如果不应该连接，那么该参数值就是 `false`。例如，调用下面的方法：

```
Rectangle2D r = . . .;
path.append(r, false);
```

可以把矩形的边框添加到该路径中，但并不与现有的路径连接在一起。而下面的方法调用：

```
path.append(r, true);
```

则是在路径的终点和矩形的起点之间添加了一条直线，然后将矩形的边框添加到该路径中。

程序清单 11-15 中的程序使你能够构建许多示例路径。图 11-37 和图 11-38 显示了运行该程序的示例结果。你可以从组合框中选择一个形状绘制器，该程序包含的形状绘制器可以用来绘制：

- 直线；
- 矩形、圆角矩形和椭圆形；
- 弧形（除了显示弧形本身外，还可以显示矩形边框的线条和起始角度及结束角度）；
- 多边形（使用 `GeneralPath` 方法）；
- 二次曲线和三次曲线。

可以用鼠标来调整控制点。当移动控制点时，形状会连续地重绘。

该程序有些复杂，因为它可以用来处理多种不同的形状，并且支持对控制点的拖拽操作。

抽象超类 `ShapeMaker` 封装了形状绘制器类的共性特征。每个形状都拥有固定数量的控制点，用户可以在控制点周围随意移动，而 `getPointCount` 方法用于返回控制点的数量。下面这个抽象方法：

```
Shape makeShape(Point2D[] points)
```

将在给定控制点的当前位置的情况下，计算实际的形状。`toString` 方法用于返回类的名字，这样，`ShapeMaker` 对象就能够放置到一个 `JComboBox` 中。

为了激活控制点的拖拽特征，`ShapePanel` 类要同时处理鼠标事件和鼠标移动事件。当鼠标在一个矩形上面被按下时，那么拖拽鼠标就可以移动该矩形了。

大部分形状绘制器类都很简单，它们的 `makeShape` 方法只是用于构建和返回需要的形状。然而，当使用 `ArcMaker` 类的时候，需要计算弧形的变形起始角度和结束角度。此外，为了说明这些计算确实是正确的，返回的形状应该是包含该弧本身、矩形边框和从弧形中心到角度控制点之间的线条等的 `GeneralPath`（参见图 11-39）。

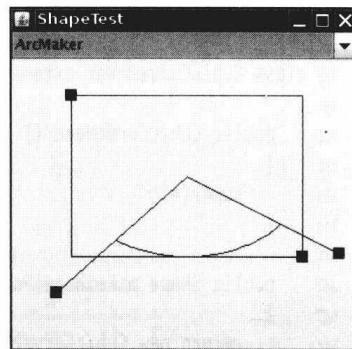


图 11-39 ShapeTest 程序的运行结果

**API** **java.awt.geom.RoundRectangle2D.Double 1.2**

- RoundRectangle2D.Double(double x, double y, double width, double height, double arcWidth, double arcHeight)

用给定的矩形边框和弧形尺寸构建一个圆角矩形。参见图 11-34 有关 arcWidth 和 arcHeight 参数的解释。

**API** **java.awt.geom.Arc2D.Double 1.2**

- Arc2D.Double(double x, double y, double w, double h, double startAngle, double arcAngle, int type)

用给定的矩形边框、起始角度、弧形角度和弧形类型构建一个弧形。startAngle 和 arcAngle 在图 11-35 中已做介绍, type 是 Arc2D.OPEN、Arc2D.PIE 和 Arc2D.CHORD 之一。

**API** **java.awt.geom.QuadCurve2D.Double 1.2**

- QuadCurve2D.Double(double x1, double y1, double ctrlx, double ctrly, double x2, double y2)  
用起始点、控制点和结束点构建一条二次曲线。

**API** **java.awt.geom.CubicCurve2D.Double 1.2**

- CubicCurve2D.Double(double x1, double y1, double ctrlx1, double ctrly1, double ctrlx2, double ctrly2, double x2, double y2)

用起始点、两个控制点和结束点构建一条三次曲线。

**API** **java.awt.geom.GeneralPath 1.2**

- GeneralPath()

构建一条空的普通路径。

**API** **java.awt.geom.Path2D.Float 6**

- void moveTo(float x, float y)

使 (x, y) 成为当前点, 也就是下一个线段的起始点。

- void lineTo(float x, float y)

- void quadTo(float ctrlx, float ctrly, float x, float y)

- void curveTo(float ctrlx1, float ctrly1, float ctrlx2, float ctrly2, float x, float y)

从当前点绘制一个线条、二次曲线或者三次曲线到达结束点 (x, y), 并且使该结束点成为当前点。

**API** **java.awt.geom.Path2D 6**

- void append(Shape s, boolean connect)

将给定形状的边框添加到普通路径中去。如果布尔型变量 connect 的值是 true, 那么该普通路径的当前点与添加进来的形状的起始点之间用一条直线连接起来。

- `void closePath()`

从当前点到路径的第一点之间绘制一条直线，从而使路径闭合。

### 11.3.3 区域

在上一节中，我们介绍了如何通过建立由线条和曲线构成的普通路径来绘制复杂的形状。通过使用足够数量的线条和曲线可以绘制出任何一种形状，例如，在屏幕上和打印文件上看到的字符的各种字体形状，都是由线条和三次曲线构成的。

有时候，使用各种不同形状的区域，比如矩形、多边形和椭圆形来建立形状，可能会更加容易描述。Java 2D API 支持四种区域几何作图（constructive area geometry）操作，用于将两个区域组合成一个区域。

- `add`: 组合区域包含了所有位于第一个区域或第二个区域内的点。
- `subtract`: 组合区域包含了所有位于第一个区域内的点，但是不包括任何位于第二个区域内的点。
- `intersect`: 组合区域包含了所有既位于第一个区域内，又位于第二个区域内的点。
- `exclusiveOr`: 组合区域包含了所有位于第一个区域内，或者是位于第二个区域内的所有点，但是这些点不能同时位于两个区域内。

图 11-40 显示了这些操作的结果。

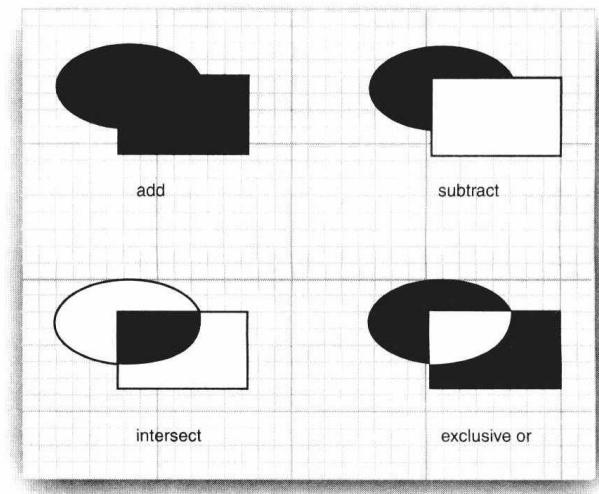


图 11-40 区域几何作图操作

如果要构建一个复杂的区域，可以使用下面的方法先创建一个默认的区域对象。

```
var a = new Area();
```

然后，将该区域和其他的形状组合起来：

```
a.add(new Rectangle2D.Double(. . .));
a.subtract(path);
. . .
```

Area 类实现了 Shape 接口。可以用 draw 方法勾勒出该区域的边界，或者使用 Graphics2D 类的 fill 方法给区域的内部着色。

#### API java.awt.geom.Area

- void add(Area other)
- void subtract(Area other)
- void intersect(Area other)
- void exclusiveOr(Area other)

对该区域和 other 所代表的另一个区域执行区域几何作图操作，并且将该区域设置为执行后的结果。

#### 11.3.4 笔画

Graphics2D 类的 draw 操作通过使用当前选定的笔画来绘制一个形状的边界。在默认的情况下，笔画是一条宽度为一个像素的实线。可以通过调用 setStroke 方法来选定不同的笔画，此时要提供一个实现了 Stroke 接口的类的对象。Java 2D API 只定义了一个这样的类，即 BasicStroke 类。在本节中，我们将介绍 BasicStroke 类的功能。

你可以构建任意粗细的笔画。例如，下面的方法就绘制了一条粗细为 10 个像素的线条。

```
g2.setStroke(new BasicStroke(10.0F));
g2.draw(new Line2D.Double(. . .));
```

当一个笔画的粗细大于一个像素的宽度时，笔画的末端可采用不同的样式。图 11-41 显示了这些所谓的端头样式。端头样式有下面三种：

- 平头样式 (butt cap) 在笔画的末端处就结束了；
- 圆头样式 (round cap) 在笔画的末端处加了一个半圆；
- 方头样式 (square cap) 在笔画的末端处加了半个方块。

当两个较粗的笔画相遇时，有三种笔画的连接样式可供选择（参见图 11-42）：

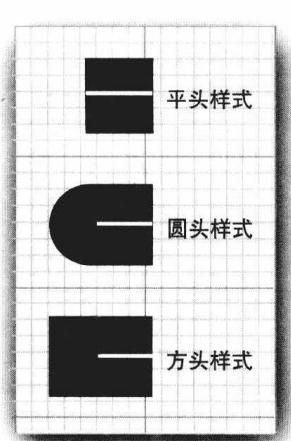


图 11-41 笔画的端头样式

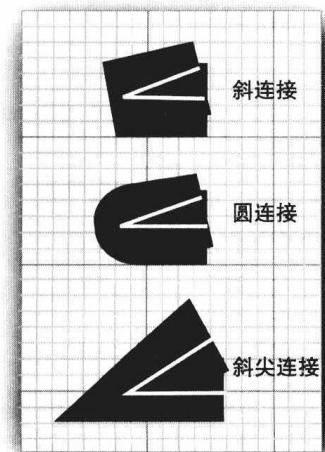


图 11-42 笔画的连接样式

- 斜连接 (bevel join)，用一条直线将两个笔画连接起来，该直线与两个笔画之间的夹角的平分线相垂直。
- 圆连接 (round join)，延长了每个笔画，并使其带有一个圆头。
- 斜尖连接 (miter join)，通过增加一个尖峰，从而同时延长了两个笔画。

斜尖连接不适合小角度连接的线条。如果两条线连接后的角度小于斜尖连接的最小角度，那么应该使用斜连接。斜连接的使用，可以防止出现太长的尖峰。默认情况下，斜尖连接的最小角度是 10 度。

可以在 BasicStroke 构造器中设定这些选择，例如：

```
g2.setStroke(new BasicStroke(10.0F, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND));
g2.setStroke(new BasicStroke(10.0F, BasicStroke.CAP_BUTT, BasicStroke.JOIN_MITER,
    15.0F /* miter limit */));
```

最后，通过设置一个虚线模式，可以指定需要使用的虚线。在程序清单 11-16 的程序中，可以选择一个虚线模式，拼出摩斯电码中的 SOS 代码。虚线模式是一个 float[] 类型的数组，它包含了笔画中“连接”(on) 和“断开”(off) 的长度 (见图 11-43)。

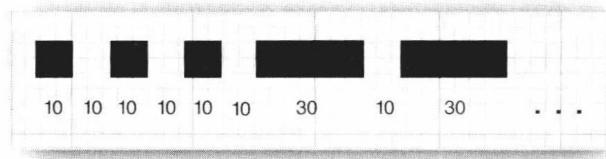


图 11-43 一种虚线图案

#### 程序清单 11-16 stroke/StrokeTest.java

```
1 package stroke;
2
3 import java.awt.*;
4 import java.awt.event.*;
5 import java.awt.geom.*;
6 import javax.swing.*;
7
8 /**
9  * This program demonstrates different stroke types.
10 * @version 1.05 2018-05-01
11 * @author Cay Horstmann
12 */
13 public class StrokeTest
14 {
15     public static void main(String[] args)
16     {
17         EventQueue.invokeLater(() ->
18         {
19             var frame = new StrokeTestFrame();
20             frame.setTitle("StrokeTest");
21             frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
22             frame.setVisible(true);
23         });
24     }
25 }
```

```
23         });
24     }
25 }
26 /**
27 * This frame lets the user choose the cap, join, and line style, and shows the resulting
28 * stroke.
29 */
30
31 class StrokeTestFrame extends JFrame
32 {
33     private StrokeComponent canvas;
34     private JPanel buttonPanel;
35
36     public StrokeTestFrame()
37     {
38         canvas = new StrokeComponent();
39         add(canvas, BorderLayout.CENTER);
40
41         buttonPanel = new JPanel();
42         buttonPanel.setLayout(new GridLayout(3, 3));
43         add(buttonPanel, BorderLayout.NORTH);
44
45         var group1 = new ButtonGroup();
46         makeCapButton("Butt Cap", BasicStroke.CAP_BUTT, group1);
47         makeCapButton("Round Cap", BasicStroke.CAP_ROUND, group1);
48         makeCapButton("Square Cap", BasicStroke.CAP_SQUARE, group1);
49
50         var group2 = new ButtonGroup();
51         makeJoinButton("Miter Join", BasicStroke.JOIN_MITER, group2);
52         makeJoinButton("Bevel Join", BasicStroke.JOIN_BEVEL, group2);
53         makeJoinButton("Round Join", BasicStroke.JOIN_ROUND, group2);
54
55         var group3 = new ButtonGroup();
56         makeDashButton("Solid Line", false, group3);
57         makeDashButton("Dashed Line", true, group3);
58     }
59
60 /**
61 * Makes a radio button to change the cap style.
62 * @param label the button label
63 * @param style the cap style
64 * @param group the radio button group
65 */
66 private void makeCapButton(String label, final int style, ButtonGroup group)
67 {
68     // select first button in group
69     boolean selected = group.getButtonCount() == 0;
70     var button = new JRadioButton(label, selected);
71     buttonPanel.add(button);
72     group.add(button);
73     button.addActionListener(event -> canvas.setCap(style));
74     pack();
75 }
76 }
```

```

77  /**
78  * Makes a radio button to change the join style.
79  * @param label the button label
80  * @param style the join style
81  * @param group the radio button group
82  */
83 private void makeJoinButton(String label, final int style, ButtonGroup group)
84 {
85     // select first button in group
86     boolean selected = group.getButtonCount() == 0;
87     var button = new JRadioButton(label, selected);
88     buttonPanel.add(button);
89     group.add(button);
90     button.addActionListener(event -> canvas.setJoin(style));
91 }
92
93 /**
94 * Makes a radio button to set solid or dashed lines.
95 * @param label the button label
96 * @param style false for solid, true for dashed lines
97 * @param group the radio button group
98 */
99 private void makeDashButton(String label, final boolean style, ButtonGroup group)
100 {
101     // select first button in group
102     boolean selected = group.getButtonCount() == 0;
103     var button = new JRadioButton(label, selected);
104     buttonPanel.add(button);
105     group.add(button);
106     button.addActionListener(event -> canvas.setDash(style));
107 }
108 }
109 /**
110 * This component draws two joined lines, using different stroke objects, and allows the user
111 * to drag the three points defining the lines.
112 */
113
114 class StrokeComponent extends JComponent
115 {
116     private static final Dimension PREFERRED_SIZE = new Dimension(400, 400);
117     private static int SIZE = 10;
118
119     private Point2D[] points;
120     private int current;
121     private float width;
122     private int cap;
123     private int join;
124     private boolean dash;
125
126     public StrokeComponent()
127     {
128         addMouseListener(new MouseAdapter()
129         {
130             public void mousePressed(MouseEvent event)

```

```

131     {
132         Point p = event.getPoint();
133         for (int i = 0; i < points.length; i++)
134         {
135             double x = points[i].getX() - SIZE / 2;
136             double y = points[i].getY() - SIZE / 2;
137             var r = new Rectangle2D.Double(x, y, SIZE, SIZE);
138             if (r.contains(p))
139             {
140                 current = i;
141                 return;
142             }
143         }
144     }
145
146     public void mouseReleased(MouseEvent event)
147     {
148         current = -1;
149     }
150 });
151
152 addMouseMotionListener(new MouseMotionAdapter()
153 {
154     public void mouseDragged(MouseEvent event)
155     {
156         if (current == -1) return;
157         points[current] = event.getPoint();
158         repaint();
159     }
160 });
161
162 points = new Point2D[3];
163 points[0] = new Point2D.Double(200, 100);
164 points[1] = new Point2D.Double(100, 200);
165 points[2] = new Point2D.Double(200, 200);
166 current = -1;
167 width = 8.0F;
168 }
169
170 public void paintComponent(Graphics g)
171 {
172     var g2 = (Graphics2D) g;
173     var path = new GeneralPath();
174     path.moveTo((float) points[0].getX(), (float) points[0].getY());
175     for (int i = 1; i < points.length; i++)
176         path.lineTo((float) points[i].getX(), (float) points[i].getY());
177     BasicStroke stroke;
178     if (dash)
179     {
180         float miterLimit = 10.0F;
181         float[] dashPattern = { 10F, 10F, 10F, 10F, 10F, 10F, 30F, 10F, 30F, 10F, 30F, 10F,
182             10F, 10F, 10F, 10F, 10F, 30F };
183         float dashPhase = 0;
184         stroke = new BasicStroke(width, cap, join, miterLimit, dashPattern, dashPhase);

```

```

185     }
186     else stroke = new BasicStroke(width, cap, join);
187     g2.setStroke(stroke);
188     g2.draw(path);
189 }
190
191 /**
192 * Sets the join style.
193 * @param j the join style
194 */
195 public void setJoin(int j)
196 {
197     join = j;
198     repaint();
199 }
200
201 /**
202 * Sets the cap style.
203 * @param c the cap style
204 */
205 public void setCap(int c)
206 {
207     cap = c;
208     repaint();
209 }
210
211 /**
212 * Sets solid or dashed lines.
213 * @param d false for solid, true for dashed lines
214 */
215 public void setDash(boolean d)
216 {
217     dash = d;
218     repaint();
219 }
220
221 public Dimension getPreferredSize() { return PREFERRED_SIZE; }
222 }

```

当构建 BasicStroke 时，可以指定虚线模式和虚线相位（dash phase）。虚线相位用来表示每条线应该从虚线模式的何处开始。通常情况下，应该把它的值设置为 0。

```

float[] dashPattern = { 10, 10, 10, 10, 10, 10, 30, 10, 30, ... };
g2.setStroke(new BasicStroke(10.0F, BasicStroke.CAP_BUTT, BasicStroke.JOIN_MITER,
    10.0F /* miter limit */, dashPattern, 0 /* dash phase */));

```

**注释：**在虚线模式中，每一条虚线的末端都可以应用端头样式。

程序清单 11-16 中的程序可以设定端头样式、连接样式和虚线（见图 11-44）。可以移动线段的端头，用以测试斜尖连接的最小角度：首先选定斜尖连接；然后，移动线段末端形成一个非常尖的锐角。可以看到斜尖连接变成了一个斜连接。

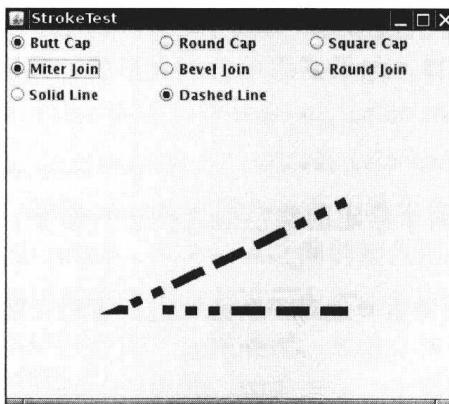


图 11-44 StrokeTest 程序

这个程序类似于程序清单 11-15 的程序。当点击一个线段的末端时，鼠标监听器就会记下操作，而鼠标动作监听器则监听对端点的拖曳操作。一组单选按钮用以表示用户选择的端头样式、连接样式以及实线或虚线。StrokePanel 类的 paintComponent 方法构建了一个 GeneralPath，它由连接着用户可以用鼠标移动的三个点的两条线段构成。然后，它根据用户的选择构建一个 BasicStroke，最后绘制出这个路径。

**API** **java.awt.Graphics2D 1.2**

- void setStroke(Stroke s)

将该图形上下文的笔画设置为实现了 Stroke 接口的给定对象。

**API** **java.awt.BasicStroke 1.2**

- BasicStroke(float width)
- BasicStroke(float width, int cap, int join)
- BasicStroke(float width, int cap, int join, float miterlimit)
- BasicStroke(float width, int cap, int join, float miterlimit, float[] dash, float dashPhase)

用给定的属性构建一个笔画对象。

参数: width 画笔的宽度

cap	端头样式，它是 CAP_BUTT、CAP_ROUND 和 CAP_SQUARE 三种样式中的一个
join	连接样式，它是 JOIN_BEVEL、JOIN_MITER 和 JOIN_ROUND 三种样式中的一个
miterlimit	用度数表示的角度，如果小于这个角度，斜尖连接将呈现为斜连接
dash	虚线笔画的填充部分和空白部分交替出现的一组长度
dashPhase	虚线模式的“相位”；位于笔画起始点前面的这段长度被假设为已经应用了该虚线模式

### 11.3.5 着色

当填充一个形状时，该形状的内部就上了颜色。使用 setPaint 方法，可以把颜色的样式

设定为一个实现了 Paint 接口的类的对象。Java 2D API 提供了三个这样的类：

- Color 类实现了 Paint 接口。如果要用单色填充形状，只需要用 Color 对象调用 setPaint 方法即可，例如：  

```
g2.setPaint(Color.red);
```
- GradientPaint 类通过在两个给定的颜色值之间进行渐变，从而改变使用的颜色（参见图 11-45）。
- TexturePaint 类用一个图像重复地对一个区域进行着色（见图 11-46）。

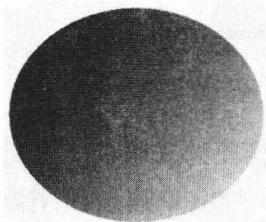


图 11-45 渐变着色

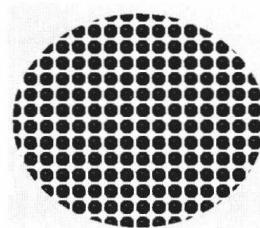


图 11-46 纹理着色

可以通过指定两个点以及在这两个点上想使用的颜色来构建一个 GradientPaint 对象，即：

```
g2.setPaint(new GradientPaint(p1, Color.RED, p2, Color.YELLOW));
```

上面语句将沿着连接两个点之间的直线的方向对颜色进行渐变，而沿着与该连接线垂直方向上的线条颜色则是不变的。超过线条端点的各个点被赋予端点上的颜色。

另外，如果调用 GradientPaint 构造器时 cyclic 参数的值为 true，即：

```
g2.setPaint(new GradientPaint(p1, Color.RED, p2, Color.YELLOW, true));
```

那么颜色将循环变换，并且在端点之外仍然保持这种变换。

如果要构建一个 TexturePaint 对象，需要指定一个 BufferedImage 和一个锚位矩形。

```
g2.setPaint(new TexturePaint(bufferedImage, anchorRectangle));
```

在本章后面部分详细讨论图像时，我们再介绍 BufferedImage 类。获取缓冲图像最简单的方式就是读入图像文件：

```
bufferedImage = ImageIO.read(new File("blue-ball.gif"));
```

锚位矩形在 x 和 y 方向上将不断地重复延伸，使之平铺到整个坐标平面。图像可以伸缩，以便纳入该锚位，然后复制到每一个平铺显示区中。

#### API **java.awt.Graphics2D 1.2**

- void setPaint(Paint s)

将图形上下文的着色设置为实现了 Paint 接口的给定对象。

#### API **java.awt.GradientPaint 1.2**

- GradientPaint(float x1, float y1, Color color1, float x2, float y2, Color color2)

- GradientPaint(float x1, float y1, Color color1, float x2, float y2, Color color2, boolean cyclic)

- GradientPaint(Point2D p1, Color color1, Point2D p2, Color color2)

- GradientPaint(Point2D p1, Color color1, Point2D p2, Color color2, boolean cyclic)

构建一个渐变着色的对象，以便用颜色来填充各个形状，其中，起始点的颜色为 color1，结束点的颜色为 color2，而两个点之间的颜色则是以线性的方式渐变。沿着连接起始点和结束点之间的线条相垂直的方向上的线条颜色是恒定不变的。在默认的情况下，渐变着色不是循环变换的。也就是说，起始点和结束点之外的各个点的颜色是分别与起始点和结束点的颜色相同的。如果渐变着色是循环的，那么颜色是连续变换的，首先返回到起始点的颜色，然后在两个方向上无限地重复。

#### API **java.awt.TexturePaint 1.2**

- TexturePaint(BufferedImage texture, Rectangle2D anchor)

建立纹理着色对象。锚位矩形定义了色的平铺空间，该矩形在 x 和 y 方向上不断地重复延伸，纹理图像则被缩放，以便填充每个平铺空间。

### 11.3.6 坐标变换

假设我们要绘制一个对象，比如汽车。从制造商的规格说明书中可以了解到汽车的高度、轴距和整个车身的长度。如果设定了每米的像素个数，当然就可以计算出所有像素的位置。但是，可以使用更加容易的方法：让图形上下文来执行这种转换。

```
g2.scale(pixelsPerMeter, pixelsPerMeter);
g2.draw(new Line2D.Double(coordinates in meters)); // converts to pixels and
// draws scaled line
```

Graphics2D 类的 scale 方法可以将图形上下文中的坐标变换设置为一个比例变换。这种变换能够将用户坐标（用户设定的单元）转换成设备坐标（pixel，即像素）。图 11-47 显示了如何进行这种变换的方法。

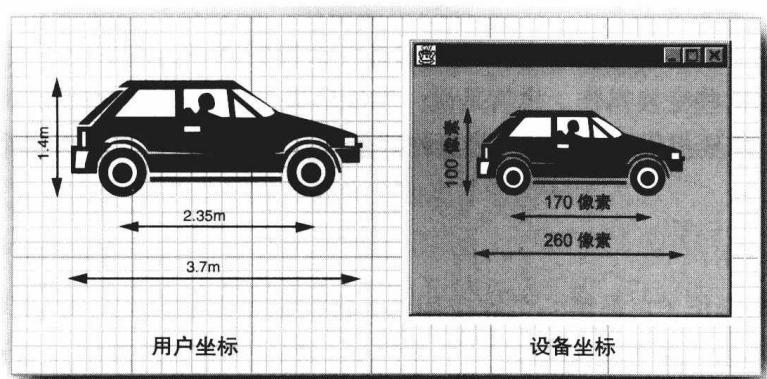


图 11-47 用户坐标与设备坐标

坐标变换在实际应用中非常有用，程序员可以使用方便的坐标值进行各种操作，图形上下文则负责执行将坐标值转换成像素的复杂工作。

这里有四种基本的变换：

- 比例缩放：放大和缩小从一个固定点出发的所有距离。
- 旋转：环绕着一个固定中心旋转所有点。
- 平移：将所有的点移动一个固定量。
- 切变：使一个线条固定不变，再按照与该固定线条之间的距离，成比例地将与该线条平行的各个线条“滑动”一个距离量。

图 11-48 显示了对一个单位的正方形进行这四种基本变换操作的效果。

`Graphics2D` 类的 `scale`、`rotate`、`translate` 和 `shear` 等方法用以将图形上下文中的坐标变换设置成为以上这些基本变换中的一种。

可以组合不同的变换操作。例如，你可能想对图形进行旋转和两倍尺寸放大的操作，这时，可以同时提供旋转和比例缩放的变换：

```
g2.rotate(angle);
g2.scale(2, 2);
g2.draw(. . .);
```

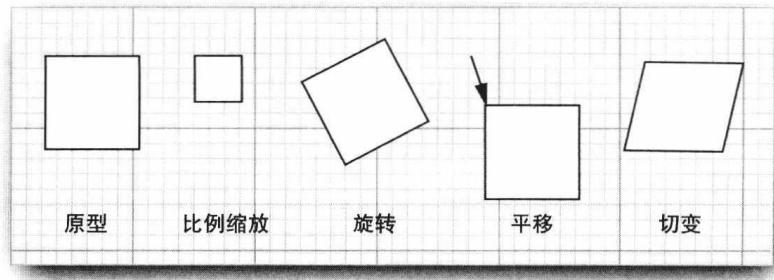


图 11-48 基本的变换

在这种情况下，变换方法的顺序是无关紧要的。然而，在大多数变换操作中，顺序却是很重要的。例如，如果想对形状进行旋转和切变操作，那么两种变换操作的不同执行序列，将会产生不同的图形。你必须明确想要得到的是什么样的图形，图形上下文将按照你所提供的相反顺序来应用这些变换操作。也就是说，你最后提供的方法会被最先应用。

可以根据你的需要提供任意多的变换操作。例如，假设你提供了下面这个变换操作序列：

```
g2.translate(x, y);
g2.rotate(a);
g2.translate(-x, -y);
```

最后一个变换操作（它是第一个被应用的）将把某个形状从点  $(x, y)$  移动到原点，第二个变换将使该形状围绕着原点旋转一个角度  $a$ ，最后一个变换方法又重新把该形状从原点移动到点  $(x, y)$  处。总体效果就是该形状围绕着中心点  $(x, y)$  进行了一次旋转（参见图 11-49）。围绕着原点之外的任意点进行旋转是一个很常见的操作，所以我们采用下面的快

捷方法：

```
g2.rotate(a, x, y);
```

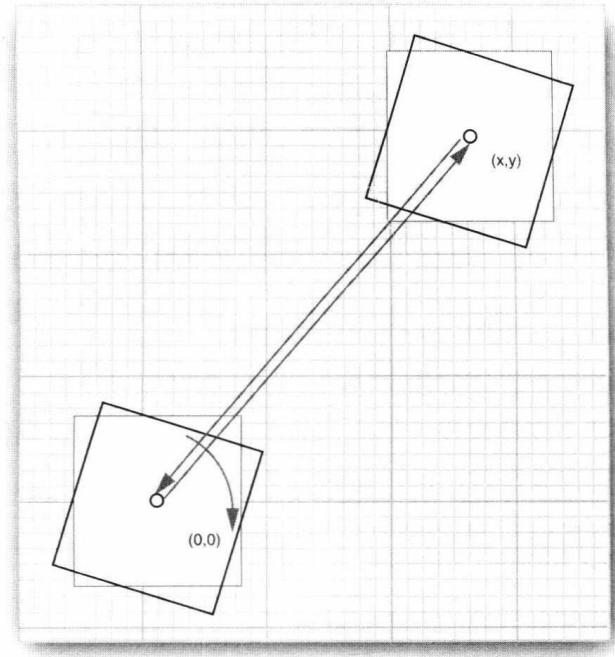


图 11-49 组合变换操作的应用

如果对矩阵论有所了解，那么就会知道所有操作（诸如旋转、平移、缩放、切变）和由这些操作组合起来的操作都能够以如下矩阵变换的形式表示出来：

$$\begin{bmatrix} x_{new} \\ y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

这种变换称为仿射变换（affine transformation）。Java 2D API 中的 `AffineTransform` 类就是用于描述这种变换的。如果你知道某个特定变换矩阵的组成元素，就可以用下面的方法直接构造它：

```
var t = new AffineTransform(a, b, c, d, e, f);
```

另外，工厂方法 `getRotateInstance`、`getScaleInstance`、`getTranslateInstance` 和 `getShearInstance` 能够构建出表示相应变换类型的矩阵。例如，调用下面的方法：

```
t = AffineTransform.getScaleInstance(2.0F, 0.5F);
```

将返回一个与下面这个矩阵相一致的变换。

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

最后，实例方法 `setToRotation`、`setToScale`、`setToTranslation` 和 `setToShear` 用于将变换对象设置为一个新的类型。下面是一个例子：

```
t.setToRotation(angle); // sets t to a rotation
```

可以把图形上下文的坐标变换设置为一个 `AffineTransform` 对象：

```
g2.setTransform(t); // replaces current transformation
```

不过，在实际运用中，不要调用 `setTransform` 操作，因为它会取代图形上下文中可能存在的任何现有的变换。例如，一个用以横向打印的图形上下文已经有了一个  $90^\circ$  的旋转变换，如果调用方法 `setTransfrom`，就会删除这样的旋转操作。可以调用 `transform` 方法作为替代方案：

```
g2.transform(t); // composes current transformation with t
```

它会把现有的变换操作和新的 `AffineTransform` 对象组合起来。

如果只想临时应用某个变换操作，那么应该首先获得旧的变换操作，然后和新的变换操作组合起来，最后当你完成操作时，再还原旧的变换操作：

```
AffineTransform oldTransform = g2.getTransform(); // save old transform
g2.transform(t); // apply temporary transform
draw on g2
g2.setTransform(oldTransform); // restore old transform
```

### API `java.awt.geom.AffineTransform` 1.2

- `AffineTransform(double a, double b, double c, double d, double e, double f)`
- `AffineTransform(float a, float b, float c, float d, float e, float f)`

用下面的矩阵构建该仿射变换。

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

- `AffineTransform(double[] m)`
- `AffineTransform(float[] m)`

用下面的矩阵构建该仿射变换。

$$\begin{bmatrix} m[0] & m[2] & m[4] \\ m[1] & [3] & m[5] \\ 0 & 0 & 1 \end{bmatrix}$$

- `static AffineTransform getRotateInstance(double a)`

创建一个围绕原点、旋转角度为  $a$  (弧度) 的旋转变换。其变换矩阵是：

$$\begin{bmatrix} \cos(a) & -\sin(a) & 0 \\ \sin(a) & \cos(a) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

如果  $a$  在  $0$  到  $\pi/2$  之间，那么图形将沿着  $x$  轴正半轴向  $y$  轴正半轴的方向旋转。

- static `AffineTransform getRotateInstance(double a, double x, double y)`

创建一个围绕点  $(x, y)$ 、旋转角度为  $a$  (弧度) 的旋转变换。

- static `AffineTransform getScaleInstance(double sx, double sy)`

创建一个比例缩放变换。 $x$  轴缩放幅度为  $sx$ ;  $y$  轴缩放幅度为  $sy$ 。其变换矩阵是：

$$\begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- static `AffineTransform getShearInstance(double shx, double shy)`

创建一个切变变换。 $x$  轴切变  $shx$ ;  $y$  轴切变  $shy$ 。其变换矩阵是：

$$\begin{bmatrix} 1 & shx & 0 \\ shy & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- static `AffineTransform getTranslateInstance(double tx, double ty)`

创建一个平移变换。 $x$  轴平移  $tx$ ;  $y$  轴平移  $ty$ 。其变换矩阵是：

$$\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix}$$

- void `setToRotation(double a)`
- void `setToRotation(double a, double x, double y)`
- void `setToScale(double sx, double sy)`
- void `setToShear(double sx, double sy)`
- void `setToTranslation(double tx, double ty)`

用给定的参数将该变换设置为一个的基本变换。如果要了解基本变换和它们的参数说明，请参见 `getXxxInstance` 方法。

#### API java.awt.Graphics2D 1.2

- void `setTransform(AffineTransform t)`

以  $t$  来取代该图形上下文中现有的坐标变换。

- void `transform(AffineTransform t)`

将该图形上下文的现有坐标变换和  $t$  组合起来。

- void `rotate(double a)`
- void `rotate(double a, double x, double y)`
- void `scale(double sx, double sy)`
- void `shear(double sx, double sy)`
- void `translate(double tx, double ty)`

将该图形上下文中现有的坐标变换和一个带有给定参数的基本变换组合起来。如果要了解基本变换和它们的参数说明，请参见 `AffineTransform.getXXXInstance` 方法。

### 11.3.7 剪切

通过在图形上下文中设置一个剪切形状，就可以将所有的绘图操作限制在该剪切形状内部来进行。

```
g2.setClip(clipShape); // but see below
g2.draw(shape); // draws only the part that falls inside the clipping shape
```

但是，在实际应用中，不应该调用这个 `setClip` 操作，因为它会取代图形上下文中可能存在的任何剪切形状。例如，正如在本章的后面部分所看到的那样，用于打印操作的图形上下文就具有一个剪切矩形，以确保你不会在页边距上绘图。相反，你应该调用 `clip` 方法。

```
g2.clip(clipShape); // better
```

`clip` 方法将你所提供的新的剪切形状同现有的剪切形状相交。

如果只想临时地使用一个剪切区域的话，那么应该首先获得旧的剪切形状，然后添加新的剪切形状，最后，在完成操作时，再还原旧的剪切形状：

```
Shape oldClip = g2.getClip(); // save old clip
g2.clip(clipShape); // apply temporary clip
draw on g2
g2.setClip(oldClip); // restore old clip
```

在图 11-50 的例子中，我们炫耀了一下剪切的功能，它绘制了一个按照复杂形状进行剪切的相当出色的线条图案，即一组字符的轮廓。

如果要获得字符的外形，需要一个字体渲染上下文（`font render context`）。请使用 `Graphics2D` 类的 `getFontRenderContext` 方法：

```
FontRenderContext context = g2.getFontRenderContext();
```

接着，使用某个字符串、某种字体和字体渲染上下文来创建一个 `TextLayout` 对象：

```
var layout = new TextLayout("Hello", font, context);
```

这个文本布局对象用于描述由特定字体渲染上下文所渲染的一个字符序列的布局。这种布局依赖于字体渲染上下文，相同的字符在屏幕上或者打印机上看起来会有不同的显示。

对我们当前的应用来说，更重要的是，`getOutline` 方法将会返回一个 `Shape` 对象，这个 `Shape` 对象用以描述在文本布局中的各个字符轮廓的形状。字符轮廓的形状从原点（0, 0）开始，这并不适合大多数的绘图操作。因此，必须为 `getOutline` 操作提供一个仿射变换操作，以便设定想要的字体轮廓所显示的位置：

```
AffineTransform transform = AffineTransform.getTranslateInstance(0, 100);
Shape outline = layout.getOutline(transform);
```



图 11-50 按照字母形状剪切出的线条图案

接着，我们把字体的轮廓附加给剪切的形状：

```
var clipShape = new GeneralPath();
clipShape.append(outline, false);
```

最后，我们设置剪切形状，并且绘制一组线条。线条仅仅在字符边界的内部显示：

```
g2.setClip(clipShape);
var p = new Point2D.Double(0, 0);
for (int i = 0; i < NINES; i++)
{
    double x = . . .;
    double y = . . .;
    var q = new Point2D.Double(x, y);
    g2.draw(new Line2D.Double(p, q)); // lines are clipped
}
```

#### API **java.awt.Graphics 1.0**

- **void setClip(Shape s) 1.2**  
将当前的剪切形状设置为形状 s。
- **Shape getClip() 1.2**  
返回当前的剪切形状。

#### API **java.awt.Graphics2D 1.2**

- **void clip(Shape s)**  
将当前的剪切形状和形状 s 相交。
- **FontRenderContext getFontRenderContext()**  
返回一个构建 TextLayout 对象所必需的字体渲染上下文。

#### API **java.awt.font.TextLayout 1.2**

- **TextLayout(String s, Font f, FontRenderContext context)**  
根据给定的字符串和字体来构建文本布局对象。方法中使用字体渲染上下文来获取特定设备的字体属性。
- **float getAdvance()**  
返回该文本布局的宽度。
- **float getAscent()**
- **float getDescent()**  
返回基准线上方和下方该文本布局的高度。
- **float getLeading()**  
返回该文本布局使用的字体中相邻两行之间的距离。

### 11.3.8 透明与组合

在标准的 RGB 颜色模型中，每种颜色都是由它的红、绿和蓝这三种成分来描述的。但

是，用它来描述透明或者部分透明的图像区域也是非常方便的。当你将一个图像置于现有图像的上面时，透明的像素完全不会遮挡它们下面的像素，而部分透明的像素则与它们下面的像素相混合。图 11-51 显示了一个部分透明的矩形和一个图像相重叠时所产生的效果，我们仍然可以透过矩形看到该图像的细节。

在 Java 2D API 中，透明是由一个透明度通道 (alpha channel) 来描述的。每个像素，除了它的红、绿和蓝色部分外，还有一个介于 0 (完全透明) 和 1 (部分透明) 之间的透明度 (alpha) 值。例如，图 11-51 中的矩形填充了一种淡黄色，透明度为 50%：

```
new Color(0.7F, 0.7F, 0.0F, 0.5F);
```

现在让我们看一看如果将两个形状重叠在一起时将会出现什么情况。必须把源像素和目标像素的颜色和透明度值混合或者组合起来。从事计算机图形学研究的 Porter 和 Duff 已经阐明了在这个混合过程中的 12 种可能的组合原则，Java 2D API 实现了所有的这些原则。在继续介绍这个问题之前，需要指出的是，这些原则中只有两个原则有实际的意义。如果你发现这些原则晦涩难懂或者难以搞清楚，那么只使用 SRC\_OVER 原则就可以了。它是 Graphics2D 对象的默认原则，并且它产生的结果最直接。

下面是这些规则的原理。假设你有了一个透明度值为  $a_s$  的源像素，在该图像中，已经存在了一个透明度值为  $a_d$  的目标像素，你想把两个像素组合起来。图 11-52 的示意图显示了如何设计一个像素的组合原则。

Porter 和 Duff 将透明度值作为像素颜色将被使用的概率。从源像素的角度来看，存在一个概率  $a_s$ ，它是源像素颜色被使用的概率；还存在一个概率  $1 - a_s$ ，它是不在乎是否使用该像素颜色的概率。同样的原则也适用于目标像素。当组合颜色时，我们假设源像素的概率和目标像素的概率是不相关的。那么正如图 11-52 所示，有四种组合情况。如果源像素想要使用它的颜色，而目标像素也不在乎，那么很自然的，我们就只使用源像素的颜色。这也是为什么右上角的矩形框用“S”来标志的原因了，这种情况的概率为  $a_s \cdot (1 - a_d)$ 。同理，左下角的矩形框用“D”来标志。如果源像素和目标像素都想选择自己的颜色，那该怎么办才好呢？这里就要应用 Porter-Duff 原则了。如果我们认为源像素比较重要，那么我们在右下角的矩形框内也标志上一个“S”。这个规则被称为 SRC\_OVER。在这个规则中，我们赋予源像素颜色的权值  $a_s$ ，目标像素颜色的权值为  $(1 - a_s) \cdot a_d$ ，然后将它们组合起来。

这样产生的视觉效果是源像素与目标像素相混合的结果，并且优先选择给定的源像素的

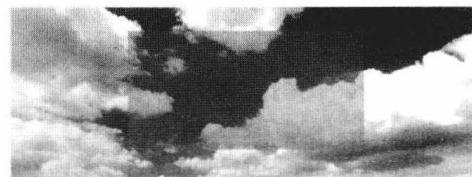


图 11-51 一个部分透明的矩形和一个图像重叠时所显示的效果

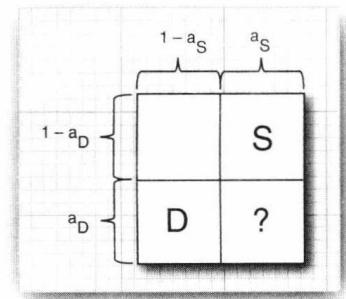


图 11-52 设计一个像素组合的原则

颜色。特别是，如果  $a_s$  为 1，那么根本就不用考虑目标像素的颜色。如果  $a_s$  为 0，那么源像素将是完全透明的，而目标像素颜色则是不变的。

还有其他的规则，可以根据置于概率示意图各个框中的字母来理解这些规则的概念。表 11-3 和图 11-53 显示了 Java 2D API 支持的所有这些规则。图 11-53 中的各个图像显示了当你使用透明度值为 0.75 的矩形源区域和透明度值为 1.0 的椭圆目标区域组合时，所显示的各种组合效果。

表 11-3 Porter-Duff 组合规则

规则	解释
CLEAR	源像素清除目标像素
SRC	源像素覆盖目标像素和空像素
DST	源像素不影响目标像素
SRC_OVER	源像素和目标像素混合，并且覆盖空像素
DST_OVER	源像素不影响目标像素，并且不覆盖空像素
SRC_IN	源像素覆盖目标像素
SRC_OUT	源像素清除目标像素，并且覆盖空像素
DST_IN	源像素的透明度值修改目标像素的透明度值
DST_OUT	源像素的透明度值取反修改目标像素的透明度值
SRC_ATOP	源像素和目标像素相混合
DST_ATOP	源像素的透明度值修改目标像素的透明度值。源像素覆盖空像素
XOR	源像素的透明度值取反修改目标像素的透明度值。源像素覆盖空像素

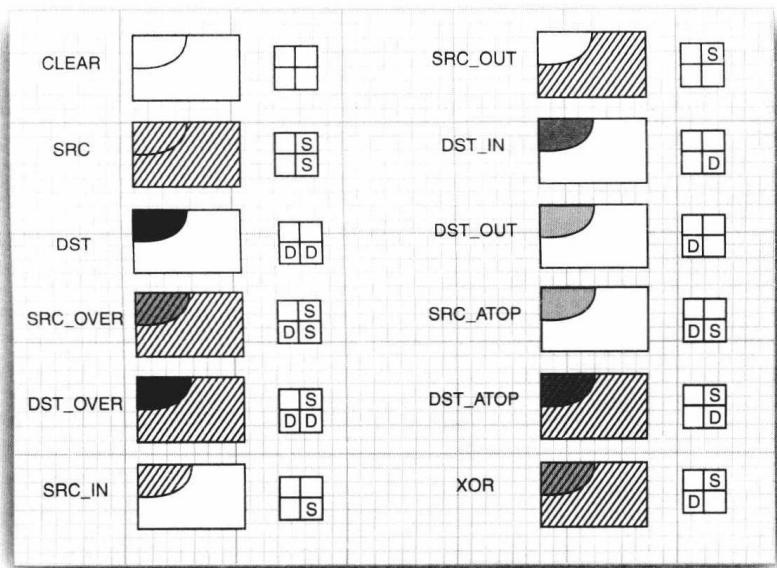


图 11-53 Porter-Duff 组合规则

如你所见，大多数规则并不是非常有用。例如，`DST_IN` 规则就是一个极端的例子。它根

本不考虑源像素颜色，但是却使用了源像素的透明度值来影响目标像素。SRC 规则可能是有用的，它强制使用源像素颜色，而且关闭了与目标像素相混合的特性。

如果要了解更多的关于 Porter-Duff 规则的信息，请参阅 Foley、Dam 和 Feiner 等撰写的 *Computer Graphics: Principles and Practice, Second Edition*。

你可以使用 `Graphics2D` 类的 `setComposite` 方法安装一个实现了 `Composite` 接口的类的对象。Java 2D API 提供了这样的一个类，即 `AlphaComposite` 它实现了图 11-53 中的所有的 Porter-Duff 规则。

`AlphaComposite` 类的工厂方法 `getInstance` 用来产生 `AlphaComposite` 对象，此时需要提供用于源像素的规则和透明度值。例如，可以考虑使用下面的代码：

```
int rule = AlphaComposite.SRC_OVER;
float alpha = 0.5f;
g2.setComposite(AlphaComposite.getInstance(rule, alpha));
g2.setPaint(Color.blue);
g2.fill(rectangle);
```

这时，矩形将使用蓝色和值为 0.5 的透明度进行着色。因为该组合规则是 `SRC_OVER`，所以它透明地置于现有图像的上面。

程序清单 11-17 中的程序深入地研究了这些组合规则。可以从组合框中选择一个规则，调节滑动条来设置 `AlphaComposite` 对象的透明度值。

### 程序清单 11-17 composite/CompositeTestFrame.java

```
1 package composite;
2
3 import java.awt.*;
4 import javax.swing.*;
5
6 /**
7 * This frame contains a combo box to choose a composition rule, a slider to change the
8 * source alpha channel, and a component that shows the composition.
9 */
10 class CompositeTestFrame extends JFrame
11 {
12     private static final int DEFAULT_WIDTH = 400;
13     private static final int DEFAULT_HEIGHT = 400;
14
15     private CompositeComponent canvas;
16     private JComboBox<Rule> ruleCombo;
17     private JSlider alphaSlider;
18     private JTextField explanation;
19
20     public CompositeTestFrame()
21     {
22         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
23
24         canvas = new CompositeComponent();
25         add(canvas, BorderLayout.CENTER);
26
27         ruleCombo = new JComboBox<>(new Rule[] { new Rule("CLEAR", " ", " "),
```

```

28     new Rule("SRC", "S", "S"), new Rule("DST", " ", "DD"),
29     new Rule("SRC_OVER", "S", "DS"), new Rule("DST_OVER", "S", "DD"),
30     new Rule("SRC_IN", " ", "S"), new Rule("SRC_OUT", "S", " "),
31     new Rule("DST_IN", " ", "D"), new Rule("DST_OUT", " ", "D"),
32     new Rule("SRC_ATOP", " ", "DS"), new Rule("DST_ATOP", "S", "D"),
33     new Rule("XOR", "S", "D"), });
34 ruleCombo.addActionListener(event ->
35 {
36     var r = (Rule) ruleCombo.getSelectedItem();
37     canvas.setRule(r.getValue());
38     explanation.setText(r.getExplanation());
39 });
40
41 alphaSlider = new JSlider(0, 100, 75);
42 alphaSlider.addChangeListener(event -> canvas.setAlpha(alphaSlider.getValue()));
43 var panel = new JPanel();
44 panel.add(ruleCombo);
45 panel.add(new JLabel("Alpha"));
46 panel.add(alphaSlider);
47 add(panel, BorderLayout.NORTH);
48
49 explanation = new JTextField();
50 add(explanation, BorderLayout.SOUTH);
51
52 canvas.setAlpha(alphaSlider.getValue());
53 Rule r = ruleCombo.getItemAt(ruleCombo.getSelectedIndex());
54 canvas.setRule(r.getValue());
55 explanation.setText(r.getExplanation());
56 }
57 }

```

此外，对每一条规则该程序都显示了一条文字描述。请注意，描述是根据组合规则表计算而来的。例如，第二行中的“DS”表示的就是“与目标像素相混合”。

该程序有一个重要的缺陷：它不能保证和屏幕相对应的图形上下文一定具有透明通道。（实际上，它通常没有这个透明通道）。当像素被放到没有透明通道的目标像素之上的时候，这些像素的颜色会与目标像素的透明度值相乘，而其透明度值却被弃用了。因为许多 Porter-Duff 规则都使用目标像素的透明度值，因此目标像素的透明通道是很重要的。由于这个原因，我们使用了一个采用 ARGB 颜色模型的缓存图像来组合各种形状。在图像被组合后，我们就将产生的图像在屏幕上绘制出来：

```

var image = new BufferedImage(getWidth(), getHeight(), BufferedImage.TYPE_INT_ARGB);
Graphics2D gImage = image.createGraphics();
// now draw to gImage
g2.drawImage(image, null, 0, 0);

```

程序清单 11-17 和程序清单 11-18 展示了框体和构件类，程序清单 11-19 中的 Rule 类提供了对每条规则的简要解释，如图 11-54 所示。在运行这个程序的时候，从左到右地移动 Alpha 滑动条，就可以观察到所产生的组合形状的效果。特别是，请注意 DST\_IN 与 DST\_OUT 规则之间唯一的差别，那就是，当你改变源像素的透明度值时，目标（！）颜色将会发生什么

样的变化。

### 程序清单 11-18 composite/CompositeComponent.java

```
1 package composite;
2
3 import java.awt.*;
4 import java.awt.geom.*;
5 import java.awt.image.*;
6 import javax.swing.*;
7
8 /**
9  * This component draws two shapes, composed with a composition rule.
10 */
11 class CompositeComponent extends JComponent
12 {
13     private int rule;
14     private Shape shape1;
15     private Shape shape2;
16     private float alpha;
17
18     public CompositeComponent()
19     {
20         shape1 = new Ellipse2D.Double(100, 100, 150, 100);
21         shape2 = new Rectangle2D.Double(150, 150, 150, 100);
22     }
23
24     public void paintComponent(Graphics g)
25     {
26         var g2 = (Graphics2D) g;
27
28         var image = new BufferedImage(getWidth(), getHeight(), BufferedImage.TYPE_INT_ARGB);
29         Graphics2D gImage = image.createGraphics();
30         gImage.setPaint(Color.red);
31         gImage.fill(shape1);
32         AlphaComposite composite = AlphaComposite.getInstance(rule, alpha);
33         gImage.setComposite(composite);
34         gImage.setPaint(Color.blue);
35         gImage.fill(shape2);
36         g2.drawImage(image, null, 0, 0);
37     }
38
39 /**
40  * Sets the composition rule.
41  * @param r the rule (as an AlphaComposite constant)
42  */
43     public void setRule(int r)
44     {
45         rule = r;
46         repaint();
47     }
48
49 /**
50  * Sets the alpha of the source.
```

```

51     * @param a the alpha value between 0 and 100
52     */
53     public void setAlpha(int a)
54     {
55         alpha = (float) a / 100.0F;
56         repaint();
57     }
58 }
```

**程序清单 11-19 composite/Rule.java**

```

1 package composite;
2
3 import java.awt.*;
4
5 /**
6  * This class describes a Porter-Duff rule.
7  */
8 class Rule
9 {
10     private String name;
11     private String porterDuff1;
12     private String porterDuff2;
13
14     /**
15      * Constructs a Porter-Duff rule.
16      * @param n the rule name
17      * @param pd1 the first row of the Porter-Duff square
18      * @param pd2 the second row of the Porter-Duff square
19      */
20     public Rule(String n, String pd1, String pd2)
21     {
22         name = n;
23         porterDuff1 = pd1;
24         porterDuff2 = pd2;
25     }
26
27     /**
28      * Gets an explanation of the behavior of this rule.
29      * @return the explanation
30      */
31     public String getExplanation()
32     {
33         var r = new StringBuilder("Source ");
34         if (porterDuff2.equals(" ")) r.append("clears");
35         if (porterDuff2.equals("S")) r.append("overwrites");
36         if (porterDuff2.equals("DS")) r.append("blends with");
37         if (porterDuff2.equals("D")) r.append("alpha modifies");
38         if (porterDuff2.equals("D ")) r.append("alpha complement modifies");
39         if (porterDuff2.equals("DD")) r.append("does not affect");
40         r.append(" destination");
41         if (porterDuff1.equals("S")) r.append(" and overwrites empty pixels");
42         r.append(".");
43         return r.toString();
44     }
45 }
```

```

44 }
45
46 public String toString()
47 {
48     return name;
49 }
50
51 /**
52 * Gets the value of this rule in the AlphaComposite class.
53 * @return the AlphaComposite constant value, or -1 if there is no matching constant
54 */
55 public int getValue()
56 {
57     try
58     {
59         return (Integer) AlphaComposite.class.getField(name).get(null);
60     }
61     catch (Exception e)
62     {
63         return -1;
64     }
65 }
66 }

```

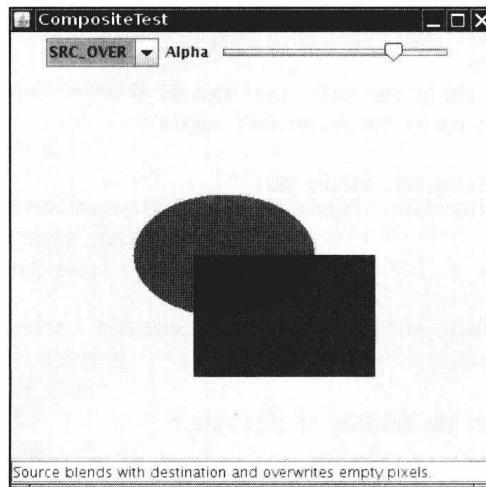


图 11-54 CompositeTest 程序运行的结果

**API** **java.awt.Graphics2D 1.2**

- void setComposite(Composite s)

把图形上下文的组合方式设置为实现了 Composite 接口的给定对象。

**API** **java.awt.AlphaComposite 1.2**

- static AlphaComposite getInstance(int rule)
- static AlphaComposite getInstance(int rule, float sourceAlpha)

构建一个透明度 (alpha) 值的组合对象。规则是 CLEAR, SRC, SRC\_OVER, DST\_OVER, SRC\_IN, SRC\_OUT, DST\_IN, DST\_OUT, DST, DST\_ATOP, SRC\_ATOP, XOR 等值之一。

## 11.4 像素图

Java2D API 使得我们可以创建由直线、曲线和区域构成的图。它是一个“向量” API，因为我们需要指定各种形状的数学属性。但是，对于处理由像素构成的图像，我们希望能够操作由颜色数据构成的“栅格”。下面将展示 Java 中的像素图。

### 11.4.1 图像的读取器和写入器

`javax.imageio` 包包含了对读取和写入数种常用文件格式进行支持的“非常方便的”特性。同时还包含了一个框架，使得第三方能够为其他图像格式的文件添加读取器和写入器。GIF、JPEG、PNG、BMP (Windows 位图) 和 WBMP (无线位图) 等文件格式都得到了支持。

该类库的基本应用是极其直接的。要想装载一个图像，可以使用 `ImageIO` 类的静态 `read` 方法。

```
File f = . . .;
BufferedImage image = ImageIO.read(f);
```

`ImageIO` 类会根据文件的类型，选择一个合适的读取器。它可以参考文件的扩展名和文件开头的专用于此目的的“幻数” (magic number) 来选择读取器。如果没有找到合适的读取器或者读取器不能解码文件的内容，那么 `read` 方法将返回 `null`。

把图像写入到文件中也是一样地简单。

```
File f = . . .;
String format = . . .;
ImageIO.write(image, format, f);
```

这里，`format` 字符串用来标识图像的格式，比如“JPEG”或者“PNG”。`ImageIO` 类将选择一个合适的写入器以存储文件。

#### 11.4.1.1 获得适合图像文件类型的读取器和写入器

对于那些超出 `ImageIO` 类的静态 `read` 和 `write` 方法能力范围的高级图像读取和写入操作来说，首先需要获得合适的 `ImageReader` 和 `ImageWriter` 对象。`ImageIO` 类枚举了匹配下列条件之一的读取器和写入器。

- 图像格式（比如“JPEG”）
- 文件后缀（比如“jpg”）
- MIME 类型（比如“image/jpeg”）

**注释：**MIME (Multipurpose Internet Mail Extensions standard) 是“多用途因特网邮件扩展标准”的英文缩写。MIME 标准定义了常用的数据格式，比如“image/jpeg”和“application/pdf”等。

例如，可以用下面的代码来获取一个 JPEG 格式文件的读取器。

```
ImageReader reader = null;
Iterator<ImageReader> iter = ImageIO.getImageReadersByFormatName("JPEG");
if (iter.hasNext()) reader = iter.next();
```

`getImageReadersBySuffix` 和 `getImageReadersByMIMEType` 这两个方法用于枚举与文件扩展名或 MIME 类型相匹配的读取器。

`ImageIO` 类可能会找到多个读取器，而它们都能够读取某一特殊类型的图像文件。在这种情况下，必须从中选择一个，但是也许你不清楚怎样才能选择一个最好的。如果要了解更多的关于读取器的信息，就要获取它的服务提供者接口：

```
ImageReaderSpi spi = reader.getOriginatingProvider();
```

然后，可以获得供应商的名字和版本号：

```
String vendor = spi.getVendor();
String version = spi.getVersion();
```

也许该信息能够帮助你决定选择哪一种读取器，或者你可以为你的程序用户提供一个读取器的列表，让他们做出选择。然而，目前来说，我们假定第一个列出来的读取器就能够满足用户的需求。

在程序清单 11-20 中，我们想查找所有可获得的读取器能够处理的文件的所有后缀，这样我们就可以在文件过滤器中使用它们。我们可以使用静态的 `ImageIO.getWriterFileSuffixes` 方法来达到此目的：

```
String[] extensions = ImageIO.getWriterFileSuffixes();
chooser.setFileFilter(new FileNameExtensionFilter("Image files", extensions));
```

对于保存文件，相对来说更麻烦一些：我们希望为用户展示一个支持所有图像类型的菜单。可惜，`IOImage` 类的 `getWriterFormatNames` 方法返回了一个相当奇怪的列表，里边包含了許多冗余的名字，比如：

```
jpg, BMP, bmp, JPEG, jpeg, wbmp, png, JPEG, PNG, WBMP, GIF, gif
```

这些并不是人们想要在菜单中显示的东西，我们所需要的是“首选”格式名列表。我们提供了一个用于此目的的助手方法 `getWriterFormats`（参见程序清单 11-7）。我们查找与每一种格式名相关的第一个写入器，然后，询问该写入器它支持的格式名是什么，从而希望它能够将最流行的一个格式名列在首位。实际上，对 JPEG 写入器来说，这种方法确实很有效：它将“JPEG”列在其他选项的前面。（另一方面，PNG 写入器把小写字母的“png”列在“PNG”的前面。我们希望这种行为能够在将来的某个时候得以解决。同时，我们强制将全小写名字转换为大写）。一旦挑选了首选名，我们就会将所有其他的候选名从最初的名字集中移除。之后，我们会继续执行直至所有的格式名都得到处理。

#### 11.4.1.2 读取和写入带多个图像的文件

有些文件，特别是 GIF 动画文件，都包含了多个图像。`ImageIO` 类的 `read` 方法只能读取单个图像。为了读取多个图像，应该将输入源（例如，输入流或者输入文件）转换成一个

`ImageInputStream`。

```
InputStream in = . . .;
ImageInputStream imageIn = ImageIO.createImageInputStream(in);
```

接着把图像输入流作为参数传递给读取器的 `setInput` 方法：

```
reader.setInput(imageIn, true);
```

方法中的第二个参数值表示输入的方式是“只向前搜索”，否则，就采用随机访问的方式，要么是在读取时缓冲输入流，要么是使用随机文件访问。对于某些操作来说，必须使用随机访问的方法。例如，为了在一个 GIF 文件中查寻图像的个数，就需要读入整个文件。这时，如果想获取某一图像的话，必须再次读入该输入文件。

只有当从一个流中读取图像，并且输入流中包含多个图像，而且在文件头中的图像格式部分没有所需要的信息（比如图像的个数）时，考虑使用上面的方法才是合适的。如果要从一个文件中读取图像信息的话，可直接使用下面的方法：

```
File f = . . .;
ImageInputStream imageIn = ImageIO.createImageInputStream(f);
reader.setInput(imageIn);
```

一旦拥有了一个读取器后，就可以通过调用下面的方法来读取输入流中的图像。

```
BufferedImage image = reader.read(index);
```

其中 `index` 是图像的索引，其值从 0 开始。

如果输入流采用“只向前搜索”的方式，那么应该持续不断地读取图像，直到 `read` 方法抛出一个 `IndexOutOfBoundsException` 为止。否则，可以调用 `getNumImages` 方法：

```
int n = reader.getNumImages(true);
```

在该方法中，它的参数表示允许搜索输入流以确定图像的数目。如果输入流采用“只向前搜索”的方式，那么该方法将抛出一个 `IllegalStateException` 异常。要不然，可以把是否“允许搜索”参数设置为 `false`。如果 `getNumImages` 方法在不搜索输入流的情况下无法确定图像的数目，那么它将返回 -1。在这种情况下，必须转换到 B 方案，那就是持续不断地读取图像，直到获得一个 `IndexOutOfBoundsException` 异常为止。

有些文件包含一些缩略图，也就是图像用来预览的小版本。可以通过调用下面的方法来获得某个图像的缩略图数量。

```
int count = reader.getNumThumbnails(index);
```

然后可以按如下方式得到一个特定索引：

```
BufferedImage thumbnail = reader.getThumbnail(index, thumbnailIndex);
```

另一个问题是，有时你想在实际获得图像之前，了解该图像的大小。特别是，当图像很大，或者是从一个较慢的网络连接中获取的时候，你更加希望能够事先了解到该图像的大小。那么请使用下面的方法：

```
int width = reader.getWidth(index);
int height = reader.getHeight(index);
```

通过上面两个方法可以获得具有给定索引的图像的大小。

如果要将多个图像写入到一个文件中，首先需要一个 `ImageWriter`。`ImageIO` 类能够枚举可以写入某种特定图像格式的所有写入器。

```
String format = . . .;
ImageWriter writer = null;
Iterator<ImageWriter> iter = ImageIO.getImageWritersByFormatName(format);
if (iter.hasNext()) writer = iter.next();
```

接着，将一个输出流或者输出文件转换成 `ImageOutputStream`，并且将其作为参数传给写入器。例如，

```
File f = . . .;
ImageOutputStream imageOut = ImageIO.createImageOutputStream(f);
writer.setOutput(imageOut);
```

必须将每一个图像都包装到 `IIOImage` 对象中。可以根据情况提供一个缩略图和图像元数据（比如，图像的压缩算法和颜色信息）的列表。在本例中，我们把两者都设置为 `null`；如果要了解详细信息，请参阅 API 文档。

```
var iioImage = new IIOImage(images[i], null, null);
```

使用 `write` 方法，可以写出第一个图像：

```
writer.write(new IIOImage(images[0], null, null));
```

对于后续的图像，使用下面的方法：

```
if (writer.canInsertImage(i))
    writer.writeInsert(i, iioImage, null);
```

上面方法中的第三个参数可以包含一个 `ImageWriteParam` 对象，用以设置图像写入的详细信息，比如是平铺还是压缩；可以用 `null` 作为其默认值。

并不是所有的图像格式都能够处理多个图像。在这种情况下，如果 `i > 0`，`canInsertImage` 方法将返回 `false` 值，而且只保存单一图像。

程序清单 11-20 中的程序使用 Java 类库所提供的读取器和写入器支持的格式来加载和保持文件。该程序显示了多个图像（见图 11-55），但是没有缩略图。

#### 程序清单 11-20 imageIO/ImageIOFrame.java

```
1 package imageIO;
2
3 import java.awt.image.*;
4 import java.io.*;
5 import java.util.*;
```

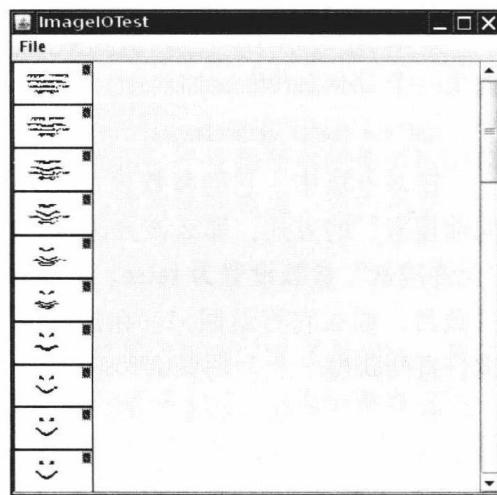


图 11-55 一个 GIF 动画图像

```
7 import javax.imageio.*;
8 import javax.imageio.stream.*;
9 import javax.swing.*;
10 import javax.swing.filechooser.*;
11 /**
12  * This frame displays the loaded images. The menu has items for loading and saving files.
13 */
14 public class ImageIOFrame extends JFrame
15 {
16     private static final int DEFAULT_WIDTH = 400;
17     private static final int DEFAULT_HEIGHT = 400;
18
19     private static Set<String> writerFormats = getWriterFormats();
20
21     private BufferedImage[] images;
22
23     public ImageIOFrame()
24     {
25         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
26
27         var fileMenu = new JMenu("File");
28         var openItem = new JMenuItem("Open");
29         openItem.addActionListener(event -> openFile());
30         fileMenu.add(openItem);
31
32         var saveMenu = new JMenu("Save");
33         fileMenu.add(saveMenu);
34         Iterator<String> iter = writerFormats.iterator();
35         while (iter.hasNext())
36         {
37             final String formatName = iter.next();
38             var formatItem = new JMenuItem(formatName);
39             saveMenu.add(formatItem);
40             formatItem.addActionListener(event -> saveFile(formatName));
41         }
42
43         var exitItem = new JMenuItem("Exit");
44         exitItem.addActionListener(event -> System.exit(0));
45         fileMenu.add(exitItem);
46
47         var menuBar = new JMenuBar();
48         menuBar.add(fileMenu);
49         setJMenuBar(menuBar);
50     }
51
52 /**
53  * Open a file and load the images.
54  */
55 public void openFile()
56 {
57     var chooser = new JFileChooser();
58     chooser.setCurrentDirectory(new File("."));
59     String[] extensions = ImageIO.getReaderFileSuffixes();
60     chooser.setFileFilter(new FileNameExtensionFilter("Image files", extensions));
```

```
int r = chooser.showOpenDialog(this);
if (r != JFileChooser.APPROVE_OPTION) return;
File f = chooser.getSelectedFile();
Box box = Box.createVerticalBox();
try
{
    String name = f.getName();
    String suffix = name.substring(name.lastIndexOf('.') + 1);
    Iterator<ImageReader> iter = ImageIO.getImageReadersBySuffix(suffix);
    ImageReader reader = iter.next();
    ImageInputStream imageIn = ImageIO.createImageInputStream(f);
    reader.setInput(imageIn);
    int count = reader.getNumImages(true);
    images = new BufferedImage[count];
    for (int i = 0; i < count; i++)
    {
        images[i] = reader.read(i);
        box.add(new JLabel(new ImageIcon(images[i])));
    }
}
catch (IOException e)
{
    JOptionPane.showMessageDialog(this, e);
}
setContentPane(new JScrollPane(box));
validate();
}

/** 
 * Save the current image in a file.
 * @param formatName the file format
 */
public void saveFile(final String formatName)
{
    if (images == null) return;
    Iterator<ImageWriter> iter = ImageIO.getImageWritersByFormatName(formatName);
    ImageWriter writer = iter.next();
    var chooser = new JFileChooser();
    chooser.setCurrentDirectory(new File("."));
    String[] extensions = writer.getOriginatingProvider().getFileSuffixes();
    chooser.setFileFilter(new FileNameExtensionFilter("Image files", extensions));
    int r = chooser.showSaveDialog(this);
    if (r != JFileChooser.APPROVE_OPTION) return;
    File f = chooser.getSelectedFile();
    try
    {
        ImageOutputStream imageOut = ImageIO.createImageOutputStream(f);
        writer.setOutput(imageOut);

        writer.write(new IIOImage(images[0], null, null));
        for (int i = 1; i < images.length; i++)
        {
            var iioImage = new IIOImage(images[i], null, null);
            if (writer.canInsertImage(i)) writer.writeInsert(i, iioImage, null);
        }
    }
}
```

```

117         }
118     }
119     catch (IOException e)
120     {
121         JOptionPane.showMessageDialog(this, e);
122     }
123 }
124
125 /**
126 * Gets a set of "preferred" format names of all image writers. The preferred format name
127 * is the first format name that a writer specifies.
128 * @return the format name set
129 */
130 public static Set<String> getWriterFormats()
131 {
132     var writerFormats = new TreeSet<String>();
133     var formatNames = List.of(ImageIO.getWriterFormatNames());
134     while (formatNames.size() > 0)
135     {
136         String name = formatNames.iterator().next();
137         Iterator<ImageWriter> iter = ImageIO.getImageWritersByFormatName(name);
138         ImageWriter writer = iter.next();
139         String[] names = writer.getOriginatingProvider().getFormatNames();
140         String format = names[0];
141         if (format.equals(format.toLowerCase())) format = format.toUpperCase();
142         writerFormats.add(format);
143         formatNames.removeAll(List.of(names));
144     }
145     return writerFormats;
146 }
147 }

```

**API** **javax.imageio.ImageIO 1.4**

- static BufferedImage read(File input)
  - static BufferedImage read(InputStream input)
  - static BufferedImage read(URL input)
- 从 input 中读取一个图像。
- static boolean write(RenderedImage image, String formatName, File output)
  - static boolean write(RenderedImage image, String formatName, OutputStream output)
- 将给定格式的图像写入 output 中。如果没有找到合适的写入器，则返回 false。
- static Iterator<ImageReader> getImageReadersByFormatName(String formatName)
  - static Iterator<ImageReader> getImageReadersBySuffix(String fileSuffix)
  - static Iterator<ImageReader> getImageReadersByMIMEType(String mimeType)
  - static Iterator<ImageWriter> getImageWritersByFormatName(String formatName)
  - static Iterator<ImageWriter> getImageWritersBySuffix(String fileSuffix)
  - static Iterator<ImageWriter> getImageWritersByMIMEType(String mimeType)

获得能够处理给定格式（例如 "JPEG"）、文件后缀（例如 "jpg"）或者 MIME 类型（例如 "image/jpeg"）的所有读取器和写入器。

- static String[] getReaderFormatNames()
- static String[] getReaderMIMETypes()
- static String[] getWriterFormatNames()
- static String[] getWriterMIMETypes()
- static String[] getReaderFileSuffixes() 6
- static String[] getWriterFileSuffixes() 6

获取读取器和写入器所支持的所有格式名、MIME 类型名和文件后缀。

- ImageInputStream createImageInputStream(Object input)
- ImageOutputStream createImageOutputStream(Object output)

根据给定的对象来创建一个图像输入流或者图像输出流。该对象可能是一个文件、一个流、一个 RandomAccessFile 或者某个服务提供商能够处理的其他类型的对象。如果没有任何注册过的服务提供器能够处理这个对象，那么返回 null 值。

#### **API javax.imageio.ImageReader 1.4**

- void setInput(Object input)
- void setInput(Object input, boolean seekForwardOnly)

设置读取器的输入源。

参数：input 一个 ImageInputStream 对象或者是这个读取器能够接受的其他对象  
seekForwardOnly 如果读取器只应该向前读取，则返回 true。默认地，读取器会采用随机访问的方式，如果有必要，将会缓存图像数据

- BufferedImage read(int index)

读取消给定索引的图像（索引从 0 开始）。如果没有这个图像，则抛出一个 IndexOutOfBoundsException 异常。

- int getNumImages(boolean allowSearch)

获取读取器中图像的数目。如果 allowSearch 值为 false，并且不向前阅读就无法确定图像的数目，那么它将返回 -1。如果 allowSearch 值是 true，并且读取器采用了“只向前搜索”方式，那么就会抛出 IllegalStateException 异常。

- int getNumThumbnails(int index)

读取消给定索引的图像的缩略图的数量。

- BufferedImage readThumbnail(int index, int thumbnailIndex)

读取消给定索引的图像的索引号为 thumbnailIndex 的缩略图。

- int getWidth(int index)

- int getHeight(int index)

获取图像的宽度和高度。如果没有这样的图像，就抛出一个 IndexOutOfBoundsException 异常。

异常。

- `ImageReaderSpi getOriginatingProvider()`

获取构建该读取器的服务提供者。

#### **API** `javax.imageio.spi.IIOServiceProvider 1.4`

- `String getVendorName()`
- `String getVersion()`

获取该服务提供者的提供商的名字和版本。

#### **API** `javax.imageio.spi.ImageReaderWriterSpi 1.4`

- `String[] getFormatNames()`
- `String[] getFileSuffixes()`
- `String[] getMIMETypes()`

获取由该服务提供者创建的读取器或者写入器所支持的图像格式名、文件的后缀和 MIME 类型。

#### **API** `javax.imageio.ImageWriter 1.4`

- `void setOutput(Object output)`

设置该写入器的输出目标。

参数： `output` 一个 `ImageOutputStream` 对象或者这个写入器能够接受的其他对象。

- `void write(II0Image image)`
- `void write(RenderedImage image)`

把单一的图像写入到输出流中。

- `void writeInsert(int index, II0Image image, ImageWriteParam param)`

把一个图像写入到一个包含多个图像的文件中。

- `boolean canInsertImage(int index)`

如果在给定的索引处可以插入一个图像的话，则返回 `true` 值。

- `ImageWriterSpi getOriginatingProvider()`

获取构建该写入器的服务提供者。

#### **API** `javax.imageio.II0Image 1.4`

- `II0Image(RenderedImage image, List thumbnails, II0Metadata metadata)`

根据一个图像、可选的缩略图和可选的元数据来构建一个 `II0Image` 对象。

### 11.4.2 图像处理

假设你有一个图像，并且希望改善图像的外观。这时需要访问该图像的每一个像素，并用其他的像素来取代这些像素。或者，你也许想要从头计算某个图像的像素，例如，你想显示一下物理测量或者数学计算的结果。`BufferedImage` 类提供了对图像中像素的控制能力，而

实现了 `BufferedImageOP` 接口的类都可以对图像进行变换操作。

**注释:** JDK1.0 有一个完全不同且复杂得多的图像框架, 它得到了优化, 以支持对从 Web 下载的图像进行增量渲染 (incremental rendering), 即一次绘制一个扫描行。但是, 操作这些图像很困难。我们在本书中不讨论这个框架。

#### 11.4.2.1 构建像素图

你处理的大多数图像都是直接从图像文件中读入的。这些图像有的可能是数码相机产生的, 有的是扫描仪扫描而产生的, 还有一些图像是绘图程序产生的。在本节中, 我们将介绍一种不同的构建图像技术, 也就是每次为图像增加一个像素。

为了创建一个图像, 需要以通常的方法构建一个 `BufferedImage` 对象:

```
image = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);
```

现在, 调用 `getRaster` 方法来获得一个类型为 `WritableRaster` 的对象, 后面将使用这个对象来访问和修改该图像的各个像素:

```
WritableRaster raster = image.getRaster();
```

使用 `setPixel` 方法可以设置一个单独的像素。这项操作的复杂性在于不能只是为该像素设置一个 `Color` 值, 还必须知道存放在缓冲中的图像是如何设定颜色的, 这依赖于图像的类型。如果图像有一个 `TYPE_INT_ARGB` 类型, 那么每一个像素都用四个值来描述, 即: 红、绿、蓝和透明度 (alpha), 每个值的取值范围都介于 0 和 255 之间, 这需要以包含四个整数值的一个数组的形式给出:

```
int[] black = { 0, 0, 0, 255 };
raster.setPixel(i, j, black);
```

用 Java 2D API 的行话来说, 这些值被称为像素的样本值。

**警告:** 还有一些参数值是 `float[]` 和 `double[]` 类型的 `setPixel` 方法。然而, 需要在这些数组中放置的值并不是介于 0.0 和 1.0 之间的规格化的颜色值:

```
float[] red = { 1.0F, 0.0F, 0.0F, 1.0F };
raster.setPixel(i, j, red); // ERROR
```

无论数组属于什么类型, 都必须提供介于 0 和 255 之间的某个值。

可以使用 `setPixels` 方法提供批量的像素。需要设置矩形的起始像素的位置和矩形的宽度和高度。接着, 提供一个包含所有像素的样本值的一个数组。例如, 如果你缓冲的图像有一个 `TYPE_INT_ARGB` 类型, 那么就应该提供第一个像素的红、绿、蓝和透明度的值 (alpha), 然后, 提供第二个像素的红、绿、蓝和透明度的值, 以此类推:

```
var pixels = new int[4 * width * height];
pixels[0] = . . .; // red value for first pixel
pixels[1] = . . .; // green value for first pixel
pixels[2] = . . .; // blue value for first pixel
pixels[3] = . . .; // alpha value for first pixel
. .
raster.setPixels(x, y, width, height, pixels);
```

反过来，如果要读入一个像素，可以使用 `getPixel` 方法。这需要提供一个含有四个整数的数组，用以存放各个样本值：

```
var sample = new int[4];
raster.getPixel(x, y, sample);
var color = new Color(sample[0], sample[1], sample[2], sample[3]);
```

可以使用 `getPixels` 方法来读取多个像素：

```
raster.getPixels(x, y, width, height, samples);
```

如果使用的图像类型不是 `TYPE_INT_ARGB`，并且已知该类型是如何表示像素值的，那么仍旧可以使用 `getPixel/setPixel` 方法。不过，必须要知道该特定图像类型的样本值是如何进行编码的。

如果需要对任意未知类型的图像进行处理，那么你就要费神了。每一个图像类型都有一个颜色模型，它能够在样本值数组和标准的 RGB 颜色模型之间进行转换。

**注释：**RGB 颜色模型并不像你想象中的那么标准。颜色值的确切样子依赖于成像设备的特性。数码相机、扫描仪、控制器和 LCD 显示器等都有它们独有的特性。结果是，同样的 RGB 值在不同的设备上看上去就存在很大的差别。国际配色联盟 (<http://www.color.org>) 推荐，所有的颜色数据都应该配有一个 ICC 配置特性，它用以设定各种颜色是如何映射到标准格式的，比如 1931 CIE XYZ 颜色技术规范。该规范是由国际照明委员会即 CIE (Commission Internationale de l'Eclairage，其网址为：<http://www.cie.co.at>) 制定的。该委员会是负责提供涉及照明和颜色等相关领域事务的技术指导的国际性机构。该规范是显示肉眼能够察觉到的所有颜色的一个标准化方法。它采用称为 X、Y、Z 三元组坐标的方式来显示颜色。(关于 1931 CIE XYZ 规范的详尽信息，可以参阅 Foley、van Dam 和 Feiner 等人所撰写的 *Computer Graphics: Principles and Practice* 一书的第 13 章。)

ICC 配置特性非常复杂。然而，我们建议使用一个相对简单的标准，称为 sRGB (请访问其网址 <http://www.w3.org/Color/sRGB.html>)。它设定了 RGB 值与 1931 CIE XYZ 值之间的具体转换方法，它可以非常出色地在通用的彩色监视器上应用。当需要在 RGB 与其他颜色空间之间进行转换的时候，Java 2D API 就使用这种转换方式。

`getColorModel` 方法返回一个颜色模型：

```
ColorModel model = image.getColorModel();
```

为了了解一个像素的颜色值，可以调用 `Raster` 类的 `getDataElements` 方法。这个方法返回了一个 `Object`，它包含了有关该颜色值的与特定颜色模型相关的描述：

```
Object data = raster.getDataElements(x, y, null);
```

**注释：**`getDataElements` 方法返回的对象实际上是一个样本值的数组。在处理这个对象时，不必要了解到这些。但是，它却解释了为什么这个方法名叫做 `getDataElements` 的原因。

颜色模型能够将该对象转换成标准的 ARGB 的值。getRGB 方法返回一个 int 类型的值，它把透明度 (alpha)、红、绿和蓝的值打包成四个块，每块包含 8 位。也可以使用 Color(int argb, boolean hasAlpha) 构造器来构建一个颜色的值：

```
int argb = model.getRGB(data);
var color = new Color(argb, true);
```

如果要把一个像素设置为某个特定的颜色值，需要按与上述相反的步骤进行操作。Color 类的 getRGB 方法会产生一个包含透明度、红、绿和蓝值的 int 型值。把这个值提供给 ColorModel 类的 getDataElements 方法，其返回值是一个包含了该颜色值的特定颜色模型描述的 Object。再将这个对象传递给 WritableRaster 类的 setDataElements 方法：

```
int argb = color.getRGB();
Object data = model.getDataElements(argb, null);
raster.setDataElements(x, y, data);
```

为了阐明如何使用这些方法来用各个像素构建图像，我们按照传统，绘制了一个 Mandelbrot 集，如图 11-56 所示。

Mandelbrot 集的思想就是把平面上的每一点和一个数字序列关联在一起。如果数字序列是收敛的，该点就被着色。如果数字序列是发散的，该点就处于透明状态。

下面就是构建简单 Mandelbrot 集的方法。对于每一个点 (a, b)，你都能按照如下的公式得到一个点集序列，其开始于点 (x, y) = (0, 0)，反复进行迭代：

$$\begin{aligned}x_{\text{new}} &= x^2 - y^2 + a \\y_{\text{new}} &= 2 \cdot x \cdot y + b\end{aligned}$$

结果证明，如果 x 或者 y 的值大于 2，那么序列就是发散的。仅有那些与导致数字序列收敛的点 (a, b) 相对应的像素才会被着色。(该数字序列的计算公式基本上是从复杂的数学概念中推导出来的。我们只使用现成的公式。

程序清单 11-21 显示了该代码。在此程序中，我们展示了如何使用 ColorModel 类将 Color 值转换成像素数据。这个过程和图像的类型是不相关的。为了增加些趣味，你可以把缓冲图像的颜色类型改变为 TYPE\_BYTE\_GRAY。不必改变程序中的任何代码，该图像的颜色模型会自动地负责把颜色转换为样本值。

### 程序清单 11-21 rasterImage/RasterImageFrame.java

```
1 package rasterImage;
2
3 import java.awt.*;
4 import java.awt.image.*;
5 import javax.swing.*;
6
7 /**
```

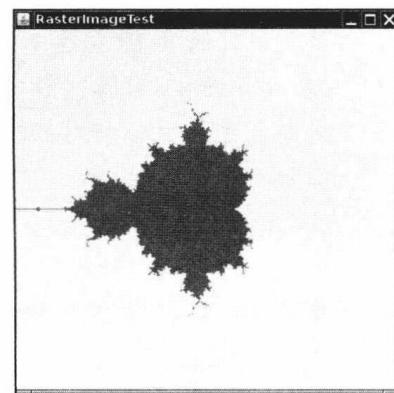


图 11-56 Mandelbrot 集

```
8 * This frame shows an image with a Mandelbrot set.
9 */
10 public class RasterImageFrame extends JFrame
11 {
12     private static final double XMIN = -2;
13     private static final double XMAX = 2;
14     private static final double YMIN = -2;
15     private static final double YMAX = 2;
16     private static final int MAX_ITERATIONS = 16;
17     private static final int IMAGE_WIDTH = 400;
18     private static final int IMAGE_HEIGHT = 400;
19
20     public RasterImageFrame()
21     {
22         BufferedImage image = makeMandelbrot(IMAGE_WIDTH, IMAGE_HEIGHT);
23         add(new JLabel(new ImageIcon(image)));
24         pack();
25     }
26
27 /**
28 * Makes the Mandelbrot image.
29 * @param width the width
30 * @param height the height
31 * @return the image
32 */
33 public BufferedImage makeMandelbrot(int width, int height)
34 {
35     var image = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);
36     WritableRaster raster = image.getRaster();
37     ColorModel model = image.getColorModel();
38
39     Color fractalColor = Color.RED;
40     int argb = fractalColor.getRGB();
41     Object colorData = model.getDataElements(argb, null);
42
43     for (int i = 0; i < width; i++)
44         for (int j = 0; j < height; j++)
45         {
46             double a = XMIN + i * (XMAX - XMIN) / width;
47             double b = YMIN + j * (YMAX - YMIN) / height;
48             if (!escapesToInfinity(a, b)) raster.setDataElements(i, j, colorData);
49         }
50     return image;
51 }
52
53 private boolean escapesToInfinity(double a, double b)
54 {
55     double x = 0.0;
56     double y = 0.0;
57     int iterations = 0;
58     while (x <= 2 && y <= 2 && iterations < MAX_ITERATIONS)
59     {
60         double xnew = x * x - y * y + a;
61         double ynew = 2 * x * y + b;
```

```

62         x = xnew;
63         y = ynew;
64         iterations++;
65     }
66     return x > 2 || y > 2;
67 }
68 }
```

**API** **java.awt.image.BufferedImage 1.2**

- `BufferedImage(int width, int height, int imageType)`

构建一个被缓存的图像对象。

参数: `width, height` 图像的尺寸

`imageType` 图像的类型, 最常用的类型是 `TYPE_INT_RGB`、`TYPE_INT_ARGB`、`TYPE_BYTE_GRAY` 和 `TYPE_BYTE_INDEXED`

- `ColorModel getColorModel()`

返回被缓存图像的颜色模型。

- `WritableRaster getRaster()`

获得访问和修改该缓存图像的像素栅格。

**API** **java.awt.image.Raster 1.2**

- `Object getDataElements(int x, int y, Object data)`

返回某个栅格点的样本数据, 该数据位于一个数组中, 而该数组的长度和类型依赖于颜色模型。如果 `data` 不为 `null`, 那么它将被视为是适合于存放样本数据的数组, 从而被充填。如果 `data` 为 `null`, 那么将分配一个新的数组, 其元素的类型和长度依赖于颜色模型。

- `int[] getPixel(int x, int y, int[] sampleValues)`
- `float[] getPixel(int x, int y, float[] sampleValues)`
- `double[] getPixel(int x, int y, double[] sampleValues)`
- `int[] getPixels(int x, int y, int width, int height, int[] sampleValues)`
- `float[] getPixels(int x, int y, int width, int height, float[] sampleValues)`
- `double[] getPixels(int x, int y, int width, int height, double[] sampleValues)`

返回某个栅格点或者是由栅格点组成的某个矩形的样本值, 该数据位于一个数组中, 数组的长度依赖于颜色模型。如果 `sampleValues` 不为 `null`, 那么该数组被视为长度足够存放样本值, 从而该数组被填充。如果 `sampleValues` 为 `null`, 就要分配一个新数组。仅当你知道某一颜色模型的样本值的具体含义的时候, 这些方法才会有用。

**API** **java.awt.image.WritableRaster 1.2**

- `void setDataElements(int x, int y, Object data)`

设置栅格点的样本数据。`data` 是一个已经填入了某一像素样本值的数组。数组元素的

类型和长度依赖于颜色模型。

- void setPixel(int x, int y, int[] sampleValues)
- void setPixel(int x, int y, float[] sampleValues)
- void setPixel(int x, int y, double[] sampleValues)
- void setPixels(int x, int y, int width, int height, int[] sampleValues)
- void setPixels(int x, int y, int width, int height, float[] sampleValues)
- void setPixels(int x, int y, int width, int height, double[] sampleValues)

设置某个栅格点或由多个栅格点组成的矩形的样本值。只有当你知道颜色模型样本值的编码规则时，这些方法才会有用。

#### **API** `java.awt.image.ColorModel 1.2`

- int getRGB(Object data)

返回对应于 `data` 数组中传递的样本数据的 ARGB 值。其元素的类型和长度依赖于颜色模型。

- Object getDataElements(int argb, Object data);

返回某个颜色值的样本数据。如果 `data` 不为 `null`，那么该数组被视为非常适合于存放样本值，进而该数组被填充。如果 `data` 为 `null`，那么将分配一个新的数组。`data` 是一个填充了用于某个像素的样本数据的数组，其元素的类型和长度依赖于该颜色模型。

#### **API** `java.awt.Color 1.0`

- Color(int argb, boolean hasAlpha) 1.2

如果 `hasAlpha` 的值是 `true`，则用指定的 ARGB 组合值创建一种颜色。如果 `hasAlpha` 的值是 `false`，则用指定的 RGB 值创建一种颜色。

- int getRGB()

返回和该颜色相对应的 ARGB 颜色值。

### 11.4.2.2 图像过滤

在前面的章节中，我们介绍了从头开始构建图像的方法。然而，你常常是因为另一个原因去访问图像数据的：你已经拥有了一个图像，并且想从某些方面对图像进行改进。

当然，可以使用前一节中的 `getPixel/getDataElements` 方法来读取和处理图像数据，然后把图像数据写回到文件中。不过，幸运的是，Java 2D API 已经提供了许多过滤器，它们能够执行常用的图像处理操作。

图像处理都实现了 `BufferedImageOp` 接口。构建了图像处理的操作之后，只需调用 `filter` 方法，就可以把该图像转换成另一个图像。

```
BufferedImageOp op = . . .;
BufferedImage filteredImage
    = new BufferedImage(image.getWidth(), image.getHeight(), image.getType());
op.filter(image, filteredImage);
```

有些图像操作可以恰当地（通过 `op.filter(image,image)` 方法）转换一个图像，但是大多数

的图像操作都做不到这一点。

以下五个类实现了 `BufferedImageOp` 接口。

```
AffineTransformOp
RescaleOp
LookupOp
ColorConvertOp
ConvolveOp
```

`AffineTransformOp` 类用于对各个像素执行仿射变换。例如，下面的代码就说明了如何使一个图像围绕着它的中心旋转。

```
AffineTransform transform = AffineTransform.getRotateInstance(Math.toRadians(angle),
    image.getWidth() / 2, image.getHeight() / 2);
var op = new AffineTransformOp(transform, interpolation);
op.filter(image, filteredImage);
```

`AffineTransformOp` 构造器需要一个仿射变换和一个渐变变换策略。如果源像素在目标像素之间的某处会发生变换的话，那么就必须使用渐变变换策略来确定目标图像的像素。例如，如果旋转源像素，那么通常它们不会精确地落在目标像素上。有两种渐变变换策略：`AffineTransformOp.TYPE_BILINEAR` 和 `AffineTransformOp.TYPE_NEAREST_NEIGHBOR`。双线性（Bilinear）渐变变换需要的时间较长，但是变换的效果却更好。

使用程序清单 11-22 的程序，可以把一个图像旋转 5°（参见图 11-57）。

`RescaleOp` 用于为图像中的所有的颜色构件执行一个调整其大小的变换操作（透明度构件不受影响）：

$$x_{\text{new}} = a \cdot x + b$$

用  $a > 1$  进行调整，那么调整后的效果是使图像变亮。可以通过设定调整大小的参数和可选的绘图提示来构建 `RescaleOp`。在程序清单 11-22 中，我们使用下面的设置：

```
float a = 1.1f;
float b = 20.0f;
var op = new RescaleOp(a, b, null);
```

也可以为每个颜色构件提供单独的缩放值，参见 API 说明。

使用 `LookupOp` 操作，可以为样本值设定任意的映射操作。你提供一张表格，用于设定每一个样本值应该如何进行映射操作。在示例程序中，我们计算了所有颜色的反，即将颜色  $c$  变成  $255 - c$ 。

`LookupOp` 构造器需要一个类型是 `LookupTable` 的对象和一个选项提示映射表。`LookupTable` 是抽象类，其有两个实体子类：`ByteLookupTable` 和 `ShortLookupTable`。因为 RGB 颜色值是由字节组成的，所以 `ByteLookupTable` 类应该就够用了。但是，考虑到在 [http://bugs.sun.com/bugdatabase/view\\_bug.do?bug\\_id=6183251](http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6183251) 中描述的缺陷，我们将使用 `ShortLookupTable`。下面的代码说明了我们在程序清单中是如何构建一个 `LookupOp` 类的：

```
var negative = new short[256];
for (int i = 0; i < 256; i++) negative[i] = (short) (255 - i);
```

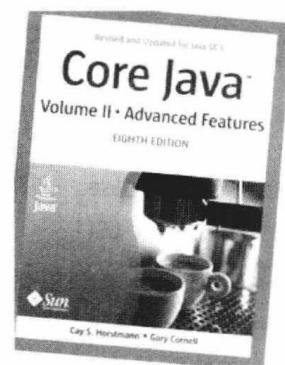


图 11-57 一个旋转的图像

```
var table = new ShortLookupTable(0, negative);
var op = new LookupOp(table, null);
```

此项操作可以分别应用于每个颜色构件，但是不能应用于透明度值。也可以为每个颜色构件提供单独的查找表，参见 API 说明。

**注释：**不能将 `LookupOp` 用于带有索引颜色模型的图像。(在这些图像中，每个样本值都是调色板中的一个偏移量。)

`ColorConvertOp` 对于颜色空间的转换非常有用。我们不准备在这里讨论这个问题了。

`ConvolveOp` 是功能最强大的转换操作，它用于执行卷积变换。我们不想过分深入地介绍卷积变换的详尽细节。不过，其基本概念还是比较简单的。我们不妨看一下模糊过滤器的例子(见图 11-58)。

这种模糊的效果是通过用像素和该像素临近的 8 个像素的平均值来取代每一个像素值而达到的。凭借直观感觉，就可以知道为什么这种变换操作能使得图像变模糊了。从数学理论上来说，这种平均法可以表示为一个以下面这个矩阵为内核的卷积变换操作：

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

卷积变换操作的内核是一个矩阵，用以说明在临近的像素点上应用的加权值。应用上面的内核进行卷积变换，就会产生一个模糊图像。下面这个不同的内核用以进行图像的边缘检测，查找图像颜色变化的区域：

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

边缘检测是在分析摄影图片时使用的一项非常重要的技术(参见图 11-59)。

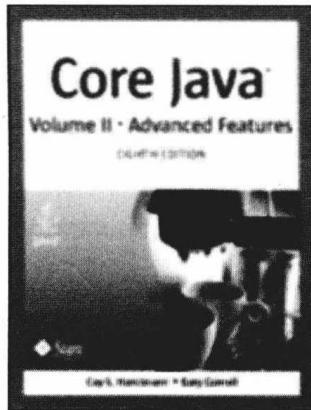


图 11-58 对图像进行模糊处理

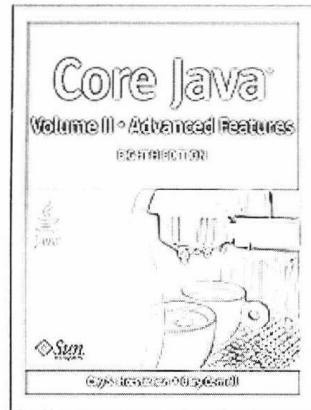


图 11-59 边缘检测

如果要构建一个卷积变换操作，首先应为矩阵内核建立一个含有内核值的数组，并且构

建一个 Kernel 对象。接着，根据内核对象建立一个 ConvolveOp 对象，进而执行过滤操作。

```
float[] elements =
{
    0.0f, -1.0f, 0.0f,
    -1.0f, 4.0f, -1.0f,
    0.0f, -1.0f, 0.0f
};
var kernel = new Kernel(3, 3, elements);
var op = new ConvolveOp(kernel);
op.filter(image, filteredImage);
```

使用程序清单 11-22 的程序，用户可以装载一个 GIF 或者 JPEG 图像，并且执行我们已经介绍过的各种图像处理的操作。由于 Java 2D API 的图像处理的功能很强大，下面的程序非常简单。

### 程序清单 11-22 imageProcessing/ImageProcessingFrame.java

```
1 package imageProcessing;
2
3 import java.awt.*;
4 import java.awt.geom.*;
5 import java.awt.image.*;
6 import java.io.*;
7
8 import javax.imageio.*;
9 import javax.swing.*;
10 import javax.swing.filechooser.*;
11
12 /**
13  * This frame has a menu to load an image and to specify various transformations, and a
14  * component to show the resulting image.
15 */
16 public class ImageProcessingFrame extends JFrame
17 {
18     private static final int DEFAULT_WIDTH = 400;
19     private static final int DEFAULT_HEIGHT = 400;
20
21     private BufferedImage image;
22
23     public ImageProcessingFrame()
24     {
25         setTitle("ImageProcessingTest");
26         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
27
28         add(new JComponent()
29         {
30             public void paintComponent(Graphics g)
31             {
32                 if (image != null) g.drawImage(image, 0, 0, null);
33             }
34         });
35
36     var fileMenu = new JMenu("File");
```

```
37 var openItem = new JMenuItem("Open");
38 openItem.addActionListener(event -> openFile());
39 fileMenu.add(openItem);
40
41 var exitItem = new JMenuItem("Exit");
42 exitItem.addActionListener(event -> System.exit(0));
43 fileMenu.add(exitItem);
44
45 var editMenu = new JMenu("Edit");
46 var blurItem = new JMenuItem("Blur");
47 blurItem.addActionListener(event ->
48 {
49     float weight = 1.0f / 9.0f;
50     float[] elements = new float[9];
51     for (int i = 0; i < 9; i++)
52         elements[i] = weight;
53     convolve(elements);
54 });
55 editMenu.add(blurItem);
56
57 var sharpenItem = new JMenuItem("Sharpen");
58 sharpenItem.addActionListener(event ->
59 {
60     float[] elements = { 0.0f, -1.0f, 0.0f, -1.0f, 5.f, -1.0f, 0.0f, -1.0f, 0.0f };
61     convolve(elements);
62 });
63 editMenu.add(sharpenItem);
64
65 var brightenItem = new JMenuItem("Brighten");
66 brightenItem.addActionListener(event ->
67 {
68     float a = 1.1f;
69     float b = 20.0f;
70     var op = new RescaleOp(a, b, null);
71     filter(op);
72 });
73 editMenu.add(brightenItem);
74
75 var edgeDetectItem = new JMenuItem("Edge detect");
76 edgeDetectItem.addActionListener(event ->
77 {
78     float[] elements = { 0.0f, -1.0f, 0.0f, -1.0f, 4.f, -1.0f, 0.0f, -1.0f, 0.0f };
79     convolve(elements);
80 });
81 editMenu.add(edgeDetectItem);
82
83 var negativeItem = new JMenuItem("Negative");
84 negativeItem.addActionListener(event ->
85 {
86     short[] negative = new short[256 * 1];
87     for (int i = 0; i < 256; i++)
88         negative[i] = (short) (255 - i);
89     var table = new ShortLookupTable(0, negative);
90     var op = new LookupOp(table, null);
```

```

91         filter(op);
92     });
93     editMenu.add(negativeItem);
94
95     var rotateItem = new JMenuItem("Rotate");
96     rotateItem.addActionListener(event ->
97     {
98         if (image == null) return;
99         var transform = AffineTransform.getRotateInstance(Math.toRadians(5),
100             image.getWidth() / 2, image.getHeight() / 2);
101         var op = new AffineTransformOp(transform,
102             AffineTransformOp.TYPE_BICUBIC);
103         filter(op);
104     });
105     editMenu.add(rotateItem);
106
107     var menuBar = new JMenuBar();
108     menuBar.add(fileMenu);
109     menuBar.add(editMenu);
110     setJMenuBar(menuBar);
111 }
112 /**
113 * Open a file and load the image.
114 */
115 public void openFile()
116 {
117     var chooser = new JFileChooser(".");
118     chooser.setCurrentDirectory(new File(getClass().getPackage().getName()));
119     String[] extensions = ImageIO.getReaderFileSuffixes();
120     chooser.setFileFilter(new FileNameExtensionFilter("Image files", extensions));
121     int r = chooser.showOpenDialog(this);
122     if (r != JFileChooser.APPROVE_OPTION) return;
123
124     try
125     {
126         Image img = ImageIO.read(chooser.getSelectedFile());
127         image = new BufferedImage(img.getWidth(null), img.getHeight(null),
128             BufferedImage.TYPE_INT_RGB);
129         image.getGraphics().drawImage(img, 0, 0, null);
130     }
131     catch (IOException e)
132     {
133         JOptionPane.showMessageDialog(this, e);
134     }
135     repaint();
136 }
137 /**
138 * Apply a filter and repaint.
139 * @param op the image operation to apply
140 */
141 private void filter(BufferedImageOp op)
142 {

```

```

145     if (image == null) return;
146     image = op.filter(image, null);
147     repaint();
148 }
149
150 /**
151 * Apply a convolution and repaint.
152 * @param elements the convolution kernel (an array of 9 matrix elements)
153 */
154 private void convolve(float[] elements)
155 {
156     var kernel = new Kernel(3, 3, elements);
157     var op = new ConvolveOp(kernel);
158     filter(op);
159 }
160 }
```

**API** **java.awt.image.BufferedImageOp 1.2**

- `BufferedImage filter(BufferedImage source, BufferedImage dest)`

将图像操作应用于源图像，并且将操作的结果存放在目标图像中。如果 `dest` 为 `null`，一个新的目标图像将被创建。该目标图像将被返回。

**API** **java.awt.image.AffineTransformOp 1.2**

- `AffineTransformOp(AffineTransform t, int interpolationType)`

构建一个仿射变换操作符。渐变变换的类型是 `TYPE_BILINEAR`、`TYPE_BICUBIC` 或者 `TYPE_NEAREST_NEIGHBOR` 中的一个。

**API** **java.awt.image.RescaleOp 1.2**

- `RescaleOp(float a, float b, RenderingHints hints)`
- `RescaleOp(float[] as, float[] bs, RenderingHints hints)`

构建一个进行尺寸调整的操作符，它会执行缩放操作  $x_{\text{new}} = a \cdot x + b$ 。当使用第一个构造器时，所有的颜色构件（但不包括透明度构件）都将按照相同的系数进行缩放。当使用第二个构造器时，可以为每个颜色构件提供单独的值，在这种情况下，透明度构件不受影响，或者为每个颜色构件和透明度构件都提供单独的值。

**API** **java.awt.image.LookupOp 1.2**

- `LookupOp(LookupTable table, RenderingHints hints)`

为给定的查找表构建一个查找操作符。

**API** **java.awt.image.ByteLookupTable 1.2**

- `ByteLookupTable(int offset, byte[] data)`
- `ByteLookupTable(int offset, byte[][] data)`

为转化 `byte` 值构建一个字节查找表。在查找之前，从输入中减去偏移量。在第一个

构造器中的值将提供给所有的颜色构件，但不包括透明度构件。当使用第二个构造器时，可以为每个颜色构件提供单独的值，在这种情况下，透明度构件不受影响，或者为每个颜色构件和透明度构件都提供单独的值。

#### API `java.awt.image.ShortLookupTable 1.2`

- `ShortLookupTable(int offset, short[] data)`
- `ShortLookupTable(int offset, short[][] data)`

为转化 short 值构建一个字节查找表。在查找之前，从输入中减去偏移量。在第一个构造器中的值将提供给所有的颜色构件，但不包括透明度构件。当使用第二个构造器时，可以为每个颜色构件提供单独的值，在这种情况下，透明度构件不受影响，或者为每个颜色构件和透明度构件都提供单独的值。

#### API `java.awt.image.ConvolveOp 1.2`

- `ConvolveOp(Kernel kernel)`
- `ConvolveOp(Kernel kernel, int edgeCondition, RenderingHints hints)`

构建一个卷积变换操作符。边界条件是 `EDGE_NO_OP` 和 `EDGE_ZERO_FILL` 两种方式之一。由于边界值没有足够的临近值来进行卷积变换的计算，所以边界值必须被特殊处理，其默认值是 `EDGE_ZERO_FILL`。

#### API `java.awt.image.Kernel 1.2`

- `Kernel(int width, int height, float[] matrixElements)`

为指定的矩阵构建一个内核。

## 11.5 打印

在本节中，我们将介绍如何在单页纸上轻松地打印出一幅图画，如何来管理多页打印输出，以及如何将打印内容存储为 PostScript 文件。

### 11.5.1 图形打印

在本节中，我们将处理最常用的打印情景，即打印一个 2D 图形，当然该图形可以含有不同字体组成的文本，甚至可能完全由文本构成。

如果要生成打印输出，必须完成下面这两个任务：

- 提供一个实现了 `Printable` 接口的对象。
- 启动一个打印作业。

`Printable` 接口只有下面一个方法：

```
int print(Graphics g, PageFormat format, int page)
```

每当打印引擎需要对某一页面进行排版以便打印时，都要调用这个方法。你的代码绘制

了准备在图形上下文上打印的文本和图像，页面排版显示了纸张的大小和页边距，页号显示了将要打印的页。

如果要启动一个打印作业，需要使用 `PrinterJob` 类。首先，应该调用静态方法 `getPrinterJob` 来获取一个打印作业对象。然后，设置要打印的 `Printable` 对象。

```
Printable canvas = . . .;
PrinterJob job = PrinterJob.getPrinterJob();
job.setPrintable(canvas);
```

**◆ 警告：** `PrintJob` 这个类处理的是 JDK1.1 风格的打印操作，这个类已经被弃用了。请不要把 `PrinterJob` 类同其混淆在一起。

在开始打印作业之前，应该调用 `printDialog` 方法来显示一个打印对话框（见图 11-60）。这个对话框为用户提供了机会去选择要使用的打印机（在有多个打印机可用的情况下），选择将要打印的页的范围，以及选择打印机的各种设置。

可以在一个实现了 `PrintRequestAttributeSet` 接口的类的对象中收集到各种打印机的设置，例如 `HashPrintRequestAttributeSet` 类：

```
var attributes = new HashPrintRequestAttributeSet();
```

你可以添加属性设置，并且把 `attributes` 对象传递给 `printDialog` 方法。

如果用户点击 `OK`，那么 `printDialog` 方法将返回 `true`；如果用户关掉对话框，那么该方法将返回 `false`。如果用户接受了设置，那么就可以调用 `PrinterJob` 类的 `print` 方法来启动打印进程。`print` 方法可能会抛出一个 `PrinterException` 异常。下面是打印代码的基本框架：

```
if (job.printDialog(attributes))
{
    try
    {
        job.print(attributes);
    }
    catch (PrinterException exception)
    {
        ...
    }
}
```

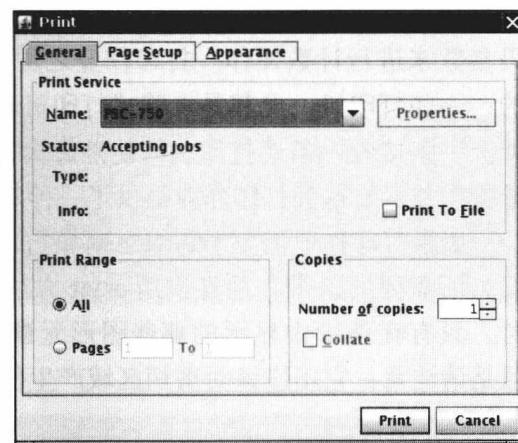


图 11-60 一个跨平台的打印对话框

**注释：** 在 JDK1.4 之前，打印系统使用的都是宿主平台本地的打印和页面设置对话框。要展示本地打印对话框，可以调用没有任何参数的 `printDialog` 方法。（不存在任何方式可以用来将用户的设置收集到一个属性集中。）

在执行打印操作时，PrinterJob 类的 print 方法不断地调用和此项打印作业相关的 Printable 对象的 print 方法。

由于打印作业不知道用户想要打印的页数，所以它只是不断地调用 print 方法。只要该 print 方法的返回值是 Printable.PAGE\_EXISTS，打印作业就不断地产生输出页。当 print 方法返回 Printable.NO\_SUCH\_PAGE 时，打印作业就停止。

**！ 警告：** 打印作业传递到 print 方法的打印页号是从 0 开始的。

因此，在打印操作完成之前，打印作业并不知道准确的打印页数。为此，打印对话框无法显示正确的页码范围，而只能显示“Pages 1 to 1”（从第一页到第一页）。在下一节中，我们将介绍如何通过为打印作业提供一个 Book 对象来避免这个缺陷。

在打印的过程中，打印作业反复地调用 Printable 对象的 print 方法。打印作业可以对同一页多次调用 print 方法，因此不应该在 print 方法内对页进行计数，而是应始终依赖于页码参数来进行计数操作。打印作业之所以能够对某一页反复地调用 print 方法是有一定道理的：一些打印机，尤其是点阵式打印机和喷墨式打印机，都使用条带打印技术，它们在打印纸上一条接着一条地打印。即使是每次打印一整页的激光打印机，打印作业都有可能使用条带打印技术。这为打印作业提供了一种对假脱机文件的大小进行管理的方法。

如果打印作业需要 printable 对象打印一个条带，那么它可以将图形上下文的剪切区域设置为所需要的条带，并且调用 print 方法。它的绘图操作将按照条带矩形区域进行剪切，同时，只有在条带中显示的那些图形元素才会被绘制出来。你的 print 方法不必晓得该过程，但是请注意：它不应该对剪切区域产生任何干扰。

**！ 警告：** 你的 print 方法获得的 Graphics 对象也是按照页边距进行剪切的。如果替换了剪切区域，那么就可以在边距外面进行绘图操作。尤其是在打印机的绘图上下文中，剪切区域是被严格遵守的。如果想进一步地限制剪切区域，可以调用 clip 方法，而不是 setClip 方法。如果必须要移除一个剪切区域，那么请务必在你的 print 方法开始处调用 getClip 方法，并还原该剪切区域。

print 方法的 PageFormat 参数包含有关被打印页的信息。getWidth 方法和 getHeight 方法返回该纸张的大小，它以磅为计量单位。1 磅等于 1/72 英吋<sup>①</sup>。例如，A4 纸的大小大约是 595 × 842 磅，美国人使用的信纸大小为 612 × 792 磅。

磅是美国印刷业中通用的计量单位，让世界上其他地方的人感到苦恼的是，打印软件包使用的是磅这种计量单位。使用磅有两个原因，即纸张的大小和纸张的页边距都是用磅来计量的。对所有的图形上下文来说，默认的计量单位就是 1 磅。你可以在本节后面的示例程序中证明这一点。该程序打印了两行文本，这两行文本之间的距离为 72 磅。运行一下示例程序，并且测量一下基准线之间的距离。它们之间的距离恰好是 1 英吋或是 25.4 毫米。

PageFormat 类的 getWidth 和 getHeight 方法给你的信息是完整的页面大小，但并不是所有的

<sup>①</sup> 1 英吋 = 0.0254 米。——编辑注

纸张区域都会被用来打印。通常的情况是，用户会选择页边距，即使他们没有选择页边距，打印机也需要用某种方法来夹住纸张，因此在纸张的周围就出现了一个不能打印的区域。

`getImageableWidth` 和 `getImageableHeight` 方法可以告诉你能够真正用来打印的区域的大小。然而，页边距没有必要是对称的，所以还必须知道可打印区域的左上角，见图 11-61，它们可以通过调用 `getImageableX` 和 `getImageableY` 方法来获得。

提示：在 `print` 方法中接收到的图形上下文是经过剪切后的图形上下文，它不包括页边距。但是，坐标系统的原点仍然是纸张的左上角。应该将该坐标系统转换成可打印区域的左上角，并以其为起点。这只需让 `print` 方法以下面的代码开始即可：

```
g.translate(pageFormat.getImageableX(), pageFormat.getImageableY());
```

如果想让用户来设定页边距，或者让用户在纵向和横向打印方式之间切换，同时并不涉及设置其他打印属性，那么就应该调用 `PrinterJob` 类的 `pageDialog` 方法。

```
PageFormat format = job.pageDialog(attributes);
```

注释：打印对话框中有一个选项卡包含了页面设置对话框（参见图 11-62）。在打印前，你仍然可以为用户提供选项来设置页面格式。特别是，如果你的程序给出了一个待打印页面的“所见即所得”的显示屏幕，那么就更应该提供这样的选项。`pageDialog` 方法返回了一个含有用户设置的 `PageFormat` 对象。

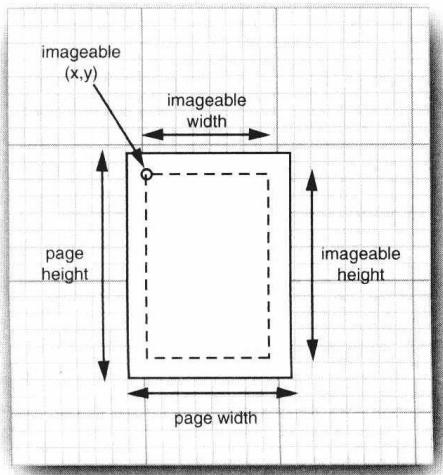


图 11-61 页面格式计量

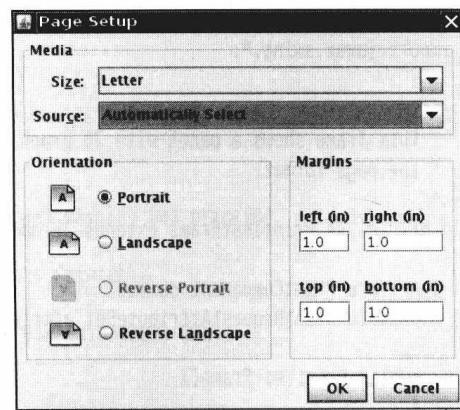


图 11-62 一个跨平台的页面设置对话框

程序清单 11-23 和程序清单 11-24 显示了如何在屏幕和打印页面上绘制相同的一组形状的方法。`Jpanel` 类的一个子类实现了 `Printable` 接口，该类中的 `paintComponent` 和 `print` 方法都调用了相同的方法来执行实际的绘图操作。

```
class PrintPanel extends JPanel implements Printable
{
```

```

public void paintComponent(Graphics g)
{
    super.paintComponent(g);
    var g2 = (Graphics2D) g;
    drawPage(g2);
}

public int print(Graphics g, PageFormat pf, int page) throws PrinterException
{
    if (page >= 1) return Printable.NO_SUCH_PAGE;
    var g2 = (Graphics2D) g;
    g2.translate(pf.getImageableX(), pf.getImageableY());
    drawPage(g2);
    return Printable.PAGE_EXISTS;
}

public void drawPage(Graphics2D g2)
{
    // shared drawing code goes here
    . .
}
. .
}

```

程序清单 11-23 print/PrintTestFrame.java

```

1 package print;
2
3 import java.awt.*;
4 import java.awt.print.*;
5
6 import javax.print.attribute.*;
7 import javax.swing.*;
8
9 /**
10  * This frame shows a panel with 2D graphics and buttons to print the graphics and to set up
11  * the page format.
12 */
13 public class PrintTestFrame extends JFrame
14 {
15     private PrintComponent canvas;
16     private PrintRequestAttributeSet attributes;
17
18     public PrintTestFrame()
19     {
20         canvas = new PrintComponent();
21         add(canvas, BorderLayout.CENTER);
22
23         attributes = new HashPrintRequestAttributeSet();
24
25         var buttonPanel = new JPanel();
26         var printButton = new JButton("Print");
27         buttonPanel.add(printButton);
28         printButton.addActionListener(event ->

```

```

29         {
30             try
31             {
32                 PrinterJob job = PrinterJob.getPrinterJob();
33                 job.setPrintable(canvas);
34                 if (job.printDialog(attributes)) job.print(attributes);
35             }
36             catch (PrinterException ex)
37             {
38                 JOptionPane.showMessageDialog(PrintTestFrame.this, ex);
39             }
40         });
41
42     var pageSetupButton = new JButton("Page setup");
43     buttonPanel.add(pageSetupButton);
44     pageSetupButton.addActionListener(event ->
45     {
46         PrinterJob job = PrinterJob.getPrinterJob();
47         job.pageDialog(attributes);
48     });
49
50     add(buttonPanel, BorderLayout.NORTH);
51     pack();
52 }
53 }
```

#### 程序清单 11-24 print/PrintComponent.java

```

1 package print;
2
3 import java.awt.*;
4 import java.awt.font.*;
5 import java.awt.geom.*;
6 import java.awt.print.*;
7 import javax.swing.*;
8
9 /**
10  * This component generates a 2D graphics image for screen display and printing.
11 */
12 public class PrintComponent extends JComponent implements Printable
13 {
14     private static final Dimension PREFERRED_SIZE = new Dimension(300, 300);
15
16     public void paintComponent(Graphics g)
17     {
18         var g2 = (Graphics2D) g;
19         drawPage(g2);
20     }
21
22     public int print(Graphics g, PageFormat pf, int page) throws PrinterException
23     {
24         if (page >= 1) return Printable.NO_SUCH_PAGE;
25         var g2 = (Graphics2D) g;
26         g2.translate(pf.getImageableX(), pf.getImageableY());
```

```

27     g2.draw(new Rectangle2D.Double(0, 0, pf.getImageableWidth(), pf.getImageableHeight()));
28
29     drawPage(g2);
30     return Printable.PAGE_EXISTS;
31 }
32
33 /**
34 * This method draws the page both on the screen and the printer graphics context.
35 * @param g2 the graphics context
36 */
37 public void drawPage(Graphics2D g2)
38 {
39     FontRenderContext context = g2.getFontRenderContext();
40     var f = new Font("Serif", Font.PLAIN, 72);
41     var clipShape = new GeneralPath();
42
43     var layout = new TextLayout("Hello", f, context);
44     AffineTransform transform = AffineTransform.getTranslateInstance(0, 72);
45     Shape outline = layout.getOutline(transform);
46     clipShape.append(outline, false);
47
48     layout = new TextLayout("World", f, context);
49     transform = AffineTransform.getTranslateInstance(0, 144);
50     outline = layout.getOutline(transform);
51     clipShape.append(outline, false);
52
53     g2.draw(clipShape);
54     g2.clip(clipShape);
55
56     final int NLINES = 50;
57     var p = new Point2D.Double(0, 0);
58     for (int i = 0; i < NLINES; i++)
59     {
60         double x = (2 * getWidth() * i) / NLINES;
61         double y = (2 * getHeight() * (NLINES - 1 - i)) / NLINES;
62         var q = new Point2D.Double(x, y);
63         g2.draw(new Line2D.Double(p, q));
64     }
65 }
66
67 public Dimension getPreferredSize() { return PREFERRED_SIZE; }
68 }

```

该示例代码显示并且打印了图 11-50，即被用作线条模式的剪切区域的消息“Hello, World”的边框。

可以点击 Print 按钮来启动打印，或者点击页面设置按钮来打开页面设置对话框。程序清单 11-23 显示了它的代码。

**注释：**为了显示本地页面设置对话框，需要将默认的 PageFormat 对象传递给 `pageDialog` 方法。该方法会克隆这个对象，并根据用户在对话框中的选择来修改它，然后返回这个克隆的对象。

```
PageFormat defaultFormat = printJob.defaultPage();
PageFormat selectedFormat = printJob.pageDialog(defaultFormat);
```

**API** **java.awt.print.Printable 1.2**

- int print(Graphics g, PageFormat format, int pageNumber)

绘制一个页面，并且返回 PAGE\_EXISTS，或者返回 NO\_SUCH\_PAGE。

参数：g 在上面绘制页面的图形上下文

format 要绘制的页面的格式

pageNumber 所请求页面的页码

**API** **java.awt.print.PrinterJob 1.2**

- static PrinterJob getPrinterJob()

返回一个打印机作业对象。

- PageFormat defaultPage()

为该打印机返回默认的页面格式。

- boolean printDialog(PrintRequestAttributeSet attributes)

- boolean printDialog()

打开打印对话框，允许用户选择将要打印的页面，并且改变打印设置。第一个方法将显示一个跨平台的打印对话框，第二个方法将显示一个本地的打印对话框。第一个方法修改了 attributes 对象来反映用户的设置。如果用户接受默认的设置，两种方法都返回 true。

- PageFormat pageDialog(PrintRequestAttributeSet attributes)

- PageFormat pageDialog(PageFormat defaults)

显示页面设置对话框。第一个方法将显示一个跨平台的对话框，第二个方法将显示一个本地的页面设置对话框。两种方法都返回了一个 PageFormat 对象，对象的格式是用户在对话框中所请求的格式。第一个方法修改了 attributes 对象以反映用户的设置。第二个对象不修改 defaults 对象。

- void setPrintable(Printable p)

- void setPrintable(Printable p, PageFormat format)

设置该打印作业的 Printable 和可选的页面格式。

- void print()

- void print(PrintRequestAttributeSet attributes)

反复地调用 print 方法，以打印当前的 Printable，并将绘制的页面发送给打印机，直到没有更多的页面需要打印为止。

**API** **java.awt.print.PageFormat 1.2**

- double getWidth()

- double getHeight()

返回页面的宽度和高度。

- `double getImageableWidth()`
- `double getImageableHeight()`

返回可打印区域的页面宽度和高度。

- `double getImageableX()`
- `double getImageableY()`

返回可打印区域的左上角的位置。

- `int getOrientation()`

返回 `PORTRAIT`、`LANDSCAPE` 和 `REVERSE_LANDSCAPE` 三者之一。页面打印的方向对程序员来说是透明的，因为打印格式和图形上下文自动地反映了页面的打印方向。

### 11.5.2 打印多页文件

在实际的打印操作中，通常不应该将原生的 `Printable` 对象传递给打印作业。相反，应该获取一个实现了 `Pageable` 接口的类的对象。Java 平台提供了这样的一个被称为 `Book` 的类。一本书是由很多章节组成的，而每个章节都是一个 `Printable` 对象。可以通过添加 `Printable` 对象和相应的页数来构建一个 `Book` 对象。

```
var book = new Book();
Printable coverPage = . . . ;
Printable bodyPages = . . . ;
book.append(coverPage, pageFormat); // append 1 page
book.append(bodyPages, pageFormat, pageCount);
```

然后，可以使用 `setPageable` 方法把 `Book` 对象传递给打印作业。

```
printJob.setPageable(book);
```

现在，打印作业就知道将要打印的确切页数了。然后，打印对话框显示一个准确的页面范围，用户可以选择整个页面范围或可选择它的一个子范围。

**!** **警告：**当打印作业调用 `Printable` 章节的 `print` 方法时，它传递的是该书的当前页码，而不是每个章节的页码。这让人非常痛苦，因为每个章节必须知道它之前所有章节的页数，这样才能使得页码参数有意义。

从程序员的视角来看，使用 `Book` 类最大的挑战就是，当你打印它时，必须知道每一个章节究竟有多少页。你的 `Printable` 类需要一个布局算法，以便用来计算在打印页面上的素材的布局。在打印开始前，要调用这个算法来计算出分页符的位置和页数。可以保留此布局信息，从而可以在打印的过程中方便地使用它。

必须警惕“用户已经修改过页面格式”这种情况的发生。如果用户修改了页面格式，即使是所打印的信息没有发生任何改变，也必须要重新计算布局。

程序清单 11-26 中显示了如何产生一个多页打印输出。该程序用很大的字符在多个页面上打印了一条消息（见图 11-63）。然后，可以剪裁掉页边缘，并将这些页面粘连起来，形成

一个标语。

Banner 类的 `layoutPages` 方法用以计算页面的布局。我们首先展示了一个字体为 72 磅的消息字符串。然后，我们计算产生的字符串的高度，并且将其与该页面的可打印高度进行比较。我们根据这两个高度值得出一个比例因子，当打印该字符串时，我们按照比例因子来放大此字符串。

**!** **警告：**如果要准确地布局打印信息，通常需要访问打印机的图形上下文。遗憾的是，只有当打印真正开始时，才能获得打印机的图形上下文。在我们的示例程序中使用的是屏幕的图形上下文，并且希望屏幕的字体度量单位与打印机的相匹配。

`Banner` 类的 `getPageCount` 方法首先调用布局方法。然后，扩展字符串的宽度，并且将该宽度除以每一页的可打印宽度。得到的商向上取整，就是要打印的页数。

由于字符可以断开分布到多个页面上，所以上面打印标语的操作好像会有困难。然而，感谢 Java 2D API 提供的强大功能，这个问题现在不过是小菜一碟。当需要打印某一页时，我们只需要调用 `Graphics2D` 类的 `translate` 方法，将字符串的左上角向左平移。接着，设置一个大小是当前页面的剪切矩形（参见图 11-64）。最后，我们用布局方法计算出的比例因子来扩展该图形上下文。

这个例子显示了图形变换操作的强大功能。绘图代码很简单，而图形变换操作负责执行将图形放到恰当位置上的所有操作。最后，剪切操作负责将落在页面外面的图像剪切掉。这个程序展示了另一种必须使用变换操作的情况，即显示页面的打印预览。

#### 程序清单 11-25 book/BookTestFrame.java

```

1 package book;
2
3 import java.awt.*;
4 import java.awt.print.*;
5
6 import javax.print.attribute.*;
7 import javax.swing.*;
8
9 /**
10 * This frame has a text field for the banner text and buttons for printing, page setup, and
11 * print preview.

```

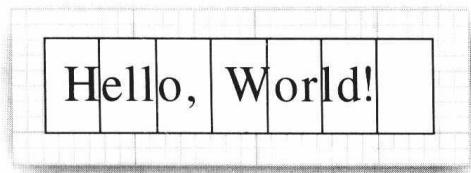


图 11-63 一幅标语

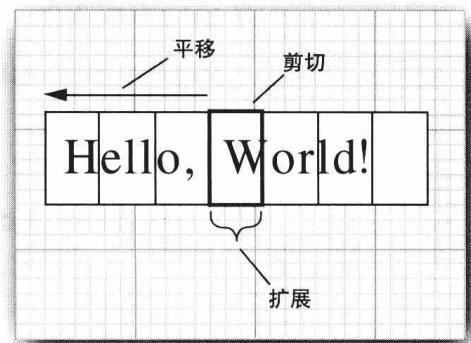


图 11-64 打印一个标语页面

```
12  */
13 public class BookTestFrame extends JFrame
14 {
15     private JTextField text;
16     private PageFormat pageFormat;
17     private PrintRequestAttributeSet attributes;
18
19     public BookTestFrame()
20     {
21         text = new JTextField();
22         add(text, BorderLayout.NORTH);
23
24         attributes = new HashPrintRequestAttributeSet();
25
26         var buttonPanel = new JPanel();
27
28         var printButton = new JButton("Print");
29         buttonPanel.add(printButton);
30         printButton.addActionListener(event ->
31             {
32                 try
33                 {
34                     PrinterJob job = PrinterJob.getPrinterJob();
35                     job.setPageable(makeBook());
36                     if (job.printDialog(attributes))
37                     {
38                         job.print(attributes);
39                     }
40                 }
41                 catch (PrinterException e)
42                 {
43                     JOptionPane.showMessageDialog(BookTestFrame.this, e);
44                 }
45             });
46
47         var pageSetupButton = new JButton("Page setup");
48         buttonPanel.add(pageSetupButton);
49         pageSetupButton.addActionListener(event ->
50             {
51                 PrinterJob job = PrinterJob.getPrinterJob();
52                 pageFormat = job.pageDialog(attributes);
53             });
54
55         var printPreviewButton = new JButton("Print preview");
56         buttonPanel.add(printPreviewButton);
57         printPreviewButton.addActionListener(event ->
58             {
59                 var dialog = new PrintPreviewDialog(makeBook());
60                 dialog.setVisible(true);
61             });
62
63         add(buttonPanel, BorderLayout.SOUTH);
64         pack();
65     }
```

```

66
67 /**
68 * Makes a book that contains a cover page and the pages for the banner.
69 */
70 public Book makeBook()
71 {
72     if (pageFormat == null)
73     {
74         PrinterJob job = PrinterJob.getPrinterJob();
75         pageFormat = job.defaultPage();
76     }
77     var book = new Book();
78     String message = text.getText();
79     var banner = new Banner(message);
80     int pageCount = banner.getPageCount((Graphics2D) getGraphics(), pageFormat);
81     book.append(new CoverPage(message + " (" + pageCount + " pages)", pageFormat));
82     book.append(banner, pageFormat, pageCount);
83     return book;
84 }
85 }
```

### 程序清单 11-26 book/Banner.java

```

1 package book;
2
3 import java.awt.*;
4 import java.awt.font.*;
5 import java.awt.geom.*;
6 import java.awt.print.*;
7
8 /**
9  * A banner that prints a text string on multiple pages.
10 */
11 public class Banner implements Printable
12 {
13     private String message;
14     private double scale;
15
16     /**
17      * Constructs a banner.
18      * @param m the message string
19      */
20     public Banner(String m)
21     {
22         message = m;
23     }
24
25     /**
26      * Gets the page count of this section.
27      * @param g2 the graphics context
28      * @param pf the page format
29      * @return the number of pages needed
30      */
31     public int getPageCount(Graphics2D g2, PageFormat pf)
```

```

32  {
33      if (message.equals("")) return 0;
34      FontRenderContext context = g2.getFontRenderContext();
35      var f = new Font("Serif", Font.PLAIN, 72);
36      Rectangle2D bounds = f.getStringBounds(message, context);
37      scale = pf.getImageableHeight() / bounds.getHeight();
38      double width = scale * bounds.getWidth();
39      int pages = (int) Math.ceil(width / pf.getImageableWidth());
40      return pages;
41  }
42
43  public int print(Graphics g, PageFormat pf, int page) throws PrinterException
44  {
45      var g2 = (Graphics2D) g;
46      if (page > getPageCount(g2, pf)) return Printable.NO_SUCH_PAGE;
47      g2.translate(pf.getImageableX(), pf.getImageableY());
48
49      drawPage(g2, pf, page);
50      return Printable.PAGE_EXISTS;
51  }
52
53  public void drawPage(Graphics2D g2, PageFormat pf, int page)
54  {
55      if (message.equals("")) return;
56      page--; // account for cover page
57
58      drawCropMarks(g2, pf);
59      g2.clip(new Rectangle2D.Double(0, 0, pf.getImageableWidth(), pf.getImageableHeight()));
60      g2.translate(-page * pf.getImageableWidth(), 0);
61      g2.scale(scale, scale);
62      FontRenderContext context = g2.getFontRenderContext();
63      var f = new Font("Serif", Font.PLAIN, 72);
64      var layout = new TextLayout(message, f, context);
65      AffineTransform transform = AffineTransform.getTranslateInstance(0, layout.getAscent());
66      Shape outline = layout.getOutline(transform);
67      g2.draw(outline);
68  }
69
70 /**
71  * Draws 1/2" crop marks in the corners of the page.
72  * @param g2 the graphics context
73  * @param pf the page format
74  */
75  public void drawCropMarks(Graphics2D g2, PageFormat pf)
76  {
77      final double C = 36; // crop mark length = 1/2 inch
78      double w = pf.getImageableWidth();
79      double h = pf.getImageableHeight();
80      g2.draw(new Line2D.Double(0, 0, 0, C));
81      g2.draw(new Line2D.Double(0, 0, C, 0));
82      g2.draw(new Line2D.Double(w, 0, w, C));
83      g2.draw(new Line2D.Double(w, 0, w - C, 0));
84      g2.draw(new Line2D.Double(0, h, 0, h - C));
85      g2.draw(new Line2D.Double(0, h, C, h));

```

```

86     g2.draw(new Line2D.Double(w, h, w, h - C));
87     g2.draw(new Line2D.Double(w, h, w - C, h));
88 }
89 }
90
91 /**
92  * This class prints a cover page with a title.
93 */
94 class CoverPage implements Printable
95 {
96     private String title;
97
98     /**
99      * Constructs a cover page.
100     * @param t the title
101     */
102    public CoverPage(String t)
103    {
104        title = t;
105    }
106
107    public int print(Graphics g, PageFormat pf, int page) throws PrinterException
108    {
109        if (page >= 1) return Printable.NO_SUCH_PAGE;
110        var g2 = (Graphics2D) g;
111        g2.setPaint(Color.black);
112        g2.translate(pf.getImageableX(), pf.getImageableY());
113        FontRenderContext context = g2.getFontRenderContext();
114        Font f = g2.getFont();
115        var layout = new TextLayout(title, f, context);
116        float ascent = layout.getAscent();
117        g2.drawString(title, 0, ascent);
118        return Printable.PAGE_EXISTS;
119    }
120 }

```

### 程序清单 11-27 book/PrintPreviewDialog.java

```

1 package book;
2
3 import java.awt.*;
4 import java.awt.print.*;
5
6 import javax.swing.*;
7
8 /**
9  * This class implements a generic print preview dialog.
10 */
11 public class PrintPreviewDialog extends JDialog
12 {
13     private static final int DEFAULT_WIDTH = 300;
14     private static final int DEFAULT_HEIGHT = 300;
15
16     private PrintPreviewCanvas canvas;

```

```

17
18 /**
19 * Constructs a print preview dialog.
20 * @param p a Printable
21 * @param pf the page format
22 * @param pages the number of pages in p
23 */
24 public PrintPreviewDialog(Printable p, PageFormat pf, int pages)
25 {
26     var book = new Book();
27     book.append(p, pf, pages);
28     layoutUI(book);
29 }
30
31 /**
32 * Constructs a print preview dialog.
33 * @param b a Book
34 */
35 public PrintPreviewDialog(Book b)
36 {
37     layoutUI(b);
38 }
39
40 /**
41 * Lays out the UI of the dialog.
42 * @param book the book to be previewed
43 */
44 public void layoutUI(Book book)
45 {
46     setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
47
48     canvas = new PrintPreviewCanvas(book);
49     add(canvas, BorderLayout.CENTER);
50
51     var buttonPanel = new JPanel();
52
53     var nextButton = new JButton("Next");
54     buttonPanel.add(nextButton);
55     nextButton.addActionListener(event -> canvas.flipPage(1));
56
57     var previousButton = new JButton("Previous");
58     buttonPanel.add(previousButton);
59     previousButton.addActionListener(event -> canvas.flipPage(-1));
60
61     var closeButton = new JButton("Close");
62     buttonPanel.add(closeButton);
63     closeButton.addActionListener(event -> setVisible(false));
64
65     add(buttonPanel, BorderLayout.SOUTH);
66 }
67 }

```

**程序清单 11-28 book/PrintPreviewCanvas.java**

```

1 package book;

```

```
2
3 import java.awt.*;
4 import java.awt.geom.*;
5 import java.awt.print.*;
6 import javax.swing.*;
7
8 /**
9  * The canvas for displaying the print preview.
10 */
11 class PrintPreviewCanvas extends JComponent
12 {
13     private Book book;
14     private int currentPage;
15
16     /**
17      * Constructs a print preview canvas.
18      * @param b the book to be previewed
19      */
20     public PrintPreviewCanvas(Book b)
21     {
22         book = b;
23         currentPage = 0;
24     }
25
26     public void paintComponent(Graphics g)
27     {
28         var g2 = (Graphics2D) g;
29         PageFormat pageFormat = book.getPageFormat(currentPage);
30
31         double xoff; // x offset of page start in window
32         double yoff; // y offset of page start in window
33         double scale; // scale factor to fit page in window
34         double px = pageFormat.getWidth();
35         double py = pageFormat.getHeight();
36         double sx = getWidth() - 1;
37         double sy = getHeight() - 1;
38         if (px / py < sx / sy) // center horizontally
39         {
40             scale = sy / py;
41             xoff = 0.5 * (sx - scale * px);
42             yoff = 0;
43         }
44         else
45             // center vertically
46         {
47             scale = sx / px;
48             xoff = 0;
49             yoff = 0.5 * (sy - scale * py);
50         }
51         g2.translate((float) xoff, (float) yoff);
52         g2.scale((float) scale, (float) scale);
53
54         // draw page outline (ignoring margins)
55         var page = new Rectangle2D.Double(0, 0, px, py);
```

```

56     g2.setPaint(Color.white);
57     g2.fill(page);
58     g2.setPaint(Color.black);
59     g2.draw(page);
60
61     Printable printable = book.getPrintable(currentPage);
62     try
63     {
64         printable.print(g2, pageFormat, currentPage);
65     }
66     catch (PrinterException e)
67     {
68         g2.draw(new Line2D.Double(0, 0, px, py));
69         g2.draw(new Line2D.Double(px, 0, 0, py));
70     }
71 }
72
73 /**
74 * Flip the book by the given number of pages.
75 * @param by the number of pages to flip by. Negative values flip backwards.
76 */
77 public void flipPage(int by)
78 {
79     int nextPage = currentPage + by;
80     if (0 <= nextPage && nextPage < book.getNumberOfPages())
81     {
82         currentPage = nextPage;
83         repaint();
84     }
85 }
86 }

```

### 11.5.3 打印服务程序

到目前为止，我们已经介绍了如何打印 2D 图形。然而，Java SE 1.4 中的打印 API 提供了更大的灵活性。该 API 定义了大量的数据类型，并且可以让你找到能够打印这些数据类型的打印服务程序。这些类型有：

- GIF、JPEG 或者 PNG 格式的图像。
- 纯文本、HTML、PostScript 或者 PDF 格式的文档。
- 原始的打印机代码数据。
- 实现了 `Printable`、`Pageable` 或 `RenderableImage` 的某个类的对象。

数据本身可以存放在一个字节源或字符源中，比如一个输入流、一个 URL 或者一个数组中。文档风格（document flavor）描述了一个数据源和一个数据类型的组合。`DocFlavor` 类为不同的数据源定义了许多内部类，每一个内部类都定义了指定风格的常量。例如，常量

`DocFlavor.INPUT_STREAM.GIF`

描述了从输入流中读入一个 GIF 格式的图像。表 11-4 中列出了数据源和数据类型的各种组合。

表 11-4 打印服务的文档风格

数据源	数据类型	MIME 类型
INPUT_STREAM	GIF	image/gif
URL	JPEG	image/jpeg
BYTE_ARRAY	PNG	image/png
	POSTSCRIPT	application/postscript
	PDF	application/pdf
	TEXT_HTML_HOST	text/html (使用主机编码)
	TEXT_HTML_US_ASCII	text/html; charset=us-ascii
	TEXT_HTML_UTF_8	text/html; charset=utf-8
	TEXT_HTML_UTF_16	text/html; charset=utf-16
	TEXT_HTML_UTF_16LE	text/html; charset=utf-16le (小尾数法)
	TEXT_HTML_UTF_16BE	text/html; charset=utf-16be (大尾数法)
	TEXT_PLAIN_HOST	text/plain (使用主机编码)
	TEXT_PLAIN_US_ASCII	text/plain; charset=us-ascii
	TEXT_PLAIN_UTF_8	text/plain; charset=utf-8
	TEXT_PLAIN_UTF_16	text/plain; charset=utf-16
	TEXT_PLAIN_UTF_16LE	text/plain; charset=utf-16le (小尾数法)
	TEXT_PLAIN_UTF_16BE	text/plain; charset=utf-16be (大尾数法)
	PCL	application/vnd.hp-PCL (惠普公司打印机控制语言)
	AUTONSENSE	application/octet-stream (原始打印数据)
READER	TEXT_HTML	text/html; charset=utf-16
STRING	TEXT_PLAIN	text/plain; charset=utf-16
CHAR_ARRAY		
SERVICE_FORMATTED	PRINTABLE	无
	PAGEABLE	无
	RENDERABLE_IMAGE	无

假设我们想打印一个位于文件中的 GIF 格式的图像。首先，确认是否有能够处理该打印任务的打印服务程序。PrintServiceLookup 类的静态 lookupPrintServices 方法返回一个能够处理给定文档风格的 PrintService 对象的数组。

```
DocFlavor flavor = DocFlavor.INPUT_STREAM.GIF;
PrintService[] services = PrintServiceLookup.lookupPrintServices(flavor, null);
```

当 lookupPrintServices 方法的第二个参数值为 null 时，表示我们不想通过设定打印机属性来限制对文档的搜索。我们在下一节中介绍打印机的属性。

如果对打印服务程序的查找返回的数组带有多个元素的话，那就需要从打印服务程序列表中选择所需的打印服务程序。通过调用 PrintService 类的 getName 方法，可以获得打印机的名称，然后让用户进行选择。

接着，从该打印服务获取一个文档打印作业：

```
DocPrintJob job = services[i].createPrintJob();
```

如果要执行打印操作，需要一个实现了 Doc 接口的类的对象。Java 为此提供了一个 SimpleDoc 类。SimpleDoc 类的构造器必须包含数据源对象、文档风格和一个可选的属性集。例如，

```
var in = new FileInputStream(fileName);
var doc = new SimpleDoc(in, flavor, null);
```

最后，就可以执行打印输出了。

```
job.print(doc, null);
```

与前面一样，null 参数可以被一个属性集取代。

请注意，这个打印进程和上一节的打印进程之间有很大的差异。这里不需要用户通过打印对话框来进行交互式操作。例如，可以实现一个服务器端的打印机制，这样，用户就可以通过 Web 表单提交打印作业了。

#### **API** javax.print.PrintServiceLookup 1.4

- PrintService[] lookupPrintServices(DocFlavor flavor, AttributeSet attributes)

查找能够处理给定文档风格和属性的打印服务程序。

参数： flavor 文档风格

attributes 需要的打印属性，如果不考虑打印属性的话，其值应该为 null

#### **API** javax.print.PrintService 1.4

- DocPrintJob createPrintJob()

为了打印实现了 Doc 接口（如 SimpleDoc）的对象而创建一个打印作业。

#### **API** javax.print.DocPrintJob 1.4

- void print(Doc doc, PrintRequestAttributeSet attributes)

打印带有给定属性的给定文档。

参数： doc 要打印的 Doc

attributes 需要的打印属性，如果不考虑任何打印属性的话，其值为 null

#### **API** javax.print.SimpleDoc 1.4

- SimpleDoc(Object data, DocFlavor flavor, DocAttributeSet attributes)

构建一个能够用 DocPrintJob 打印的 SimpleDoc 对象。

参数： data 带有打印数据的对象，比如一个输入流或者一个 Printable

flavor 打印数据的文档风格

attributes 文档属性，如果不考虑文档属性，其值为 null

### 11.5.4 流打印服务程序

打印服务程序将打印数据发送给打印机。流打印服务程序产生同样的打印数据，但是并不把数据发送给打印机，而是发给流。这么做的目的也许是延迟打印或者因为打印数据

格式可以由其他程序来进行解释。尤其是，如果打印数据格式是 PostScript 时，那么可将打印数据保存到一个文件中，因为有许多程序都能够处理 PostScript 文件。Java 平台引入了一个流打印服务程序，它能够从图像和 2D 图形中产生 PostScript 输出。可以在任何系统中使用这种服务程序，即使这些系统中没有本地打印机，也可以使用该服务程序。

枚举流打印服务程序要比定位普通的打印服务程序复杂一些。既需要打印对象的 DocFlavor 又需要流输出的 MIME 类型，接着获得一个 StreamPrintServiceFactory 类型的数组，如下所示：

```
DocFlavor flavor = DocFlavor.SERVICE_FORMATTED.PRINTABLE;
String mimeType = "application/postscript";
StreamPrintServiceFactory[] factories
    = StreamPrintServiceFactory.lookupStreamPrintServiceFactories(flavor, mimeType);
```

StreamPrintServiceFactory 类没有任何方法能够帮助我们区分不同的 factory，所以我们只提取 factories[0]。我们调用带有输出流参数的 getPrintService 方法来获得一个 StreamPrintService 对象。

```
var out = new FileOutputStream(fileName);
StreamPrintService service = factories[0].getPrintService(out);
```

StreamPrintService 类是 PrintService 的子类。如果要产生一个打印输出，只要按照上一节介绍的步骤进行操作即可。

#### API **javax.print.StreamPrintServiceFactory 1.4**

- StreamPrintServiceFactory[] lookupStreamPrintServiceFactories(DocFlavor flavor, String mimeType)

查找所需的流打印服务程序工厂，它能够打印给定文档风格，并且产生一个给定 MIME 类型的输出流。

- StreamPrintService getPrintService(OutputStream out)

获得一个打印服务程序，以便将打印输出发送到指定的输出流中。

程序清单 11-29 展示了如何使用流打印服务程序将 Java 2D 的形状打印到 PostScript 文件中。你可以用任何生成 Java 2D 形状的代码替换其中的样例绘图代码，然后将这些形状转换为 PostScript。然后，通过使用外部工具，你可以很容易地将得到的结果转换成 PDF 或 EPS。(遗憾的是，Java 不支持直接打印成 PDF)。

**注释：**在这个示例中，我们在 Graphics2D 对象上调用了绘制 Java 2D 形状的 draw 方法。如果想要绘制一个构件的表层（例如表格或树），那么使用下面的代码：

```
private static int IMAGE_WIDTH = component.getWidth();
private static int IMAGE_HEIGHT = component.getHeight();
public static void draw(Graphics2D g2) { component.paint(g2); }
```

#### 程序清单 11-29 printService/PrintServiceTest.java

```
1 package printService;
2
```

```
3 import java.awt.*;
4 import java.awt.font.*;
5 import java.awt.geom.*;
6 import java.awt.print.*;
7 import java.io.*;
8 import javax.print.*;
9 import javax.print.attribute.*;

10 /**
11 * This program demonstrates the use of stream print services. The program prints
12 * Java 2D shapes to a PostScript file. If you don't supply a file name on the command
13 * line, the output is saved to out.ps.
14 * @version 1.0 2018-06-01
15 * @author Cay Horstmann
16 */
17

18 public class PrintServiceTest
19 {
20     // Set your image dimensions here
21     private static int IMAGE_WIDTH = 300;
22     private static int IMAGE_HEIGHT = 300;
23
24     public static void draw(Graphics2D g2)
25     {
26         // Your drawing instructions go here
27         FontRenderContext context = g2.getFontRenderContext();
28         var f = new Font("Serif", Font.PLAIN, 72);
29         var clipShape = new GeneralPath();

30         var layout = new TextLayout("Hello", f, context);
31         AffineTransform transform = AffineTransform.getTranslateInstance(0, 72);
32         Shape outline = layout.getOutline(transform);
33         clipShape.append(outline, false);

34         layout = new TextLayout("World", f, context);
35         transform = AffineTransform.getTranslateInstance(0, 144);
36         outline = layout.getOutline(transform);
37         clipShape.append(outline, false);

38         g2.draw(clipShape);
39         g2.clip(clipShape);

40
41         final int NLINES = 50;
42         var p = new Point2D.Double(0, 0);
43         for (int i = 0; i < NLINES; i++)
44         {
45             double x = (2 * IMAGE_WIDTH * i) / NLINES;
46             double y = (2 * IMAGE_HEIGHT * (NLINES - 1 - i)) / NLINES;
47             var q = new Point2D.Double(x, y);
48             g2.draw(new Line2D.Double(p, q));
49         }
50     }

51     public static void main(String[] args) throws IOException, PrintException
52     {
53 }
```

```

57     String fileName = args.length > 0 ? args[0] : "out.ps";
58     DocFlavor flavor = DocFlavor.SERVICE_FORMATTED.PRINTABLE;
59     var mimeType = "application/postscript";
60     StreamPrintServiceFactory[] factories
61         = StreamPrintServiceFactory.lookupStreamPrintServiceFactories(flavor, mimeType);
62     var out = new FileOutputStream(fileName);
63     if (factories.length > 0)
64     {
65         PrintService service = factories[0].getPrintService(out);
66         var doc = new SimpleDoc(new Printable()
67         {
68             public int print(Graphics g, PageFormat pf, int page)
69             {
70                 if (page >= 1) return Printable.NO_SUCH_PAGE;
71                 else
72                 {
73                     double sf1 = pf.getImageableWidth() / (IMAGE_WIDTH + 1);
74                     double sf2 = pf.getImageableHeight() / (IMAGE_HEIGHT + 1);
75                     double s = Math.min(sf1, sf2);
76                     var g2 = (Graphics2D) g;
77                     g2.translate((pf.getWidth() - pf.getImageableWidth()) / 2,
78                         (pf.getHeight() - pf.getImageableHeight()) / 2);
79                     g2.scale(s, s);
80
81                     draw(g2);
82                     return Printable.PAGE_EXISTS;
83                 }
84             }
85         }, flavor, null);
86         DocPrintJob job = service.createPrintJob();
87         var attributes = new HashPrintRequestAttributeSet();
88         job.print(doc, attributes);
89     }
90     else
91         System.out.println("No factories for " + mimeType);
92 }
93 }
```

### 11.5.5 打印属性

打印服务程序 API 包含了一组复杂的接口和类，用以设定不同种类的属性。重要的属性共有四组，前两组属性用于设定对打印机的访问请求。

- 打印请求属性（Print request attribute）为一个打印作业中的所有 doc 对象请求特定的打印属性，例如，双面打印或者纸张的大小。
  - Doc 属性（Doc attribute）是仅作用在单个 doc 对象上的请求属性。
- 另外两组属性包含关于打印机和作业状态的信息。
- 打印服务属性（Print service attribute）提供了关于打印服务程序的信息，比如打印机的种类和型号，或者打印机当前是否接受打印作业。
  - 打印作业属性（Print job attribute）提供了关于某个特定打印作业状态的信息，比如该

打印作业是否已经完成。

如果要描述各种不同的打印属性，可以使用带有如下子接口的 Attribute 接口。

```
PrintRequestAttribute
DocAttribute
PrintServiceAttribute
PrintJobAttribute
SupportedValuesAttribute
```

各个属性类都实现了上面的一个或几个接口。例如，Copies 类的对象描述了一个打印输出的拷贝数量，该类就实现了 PrintRequestAttribute 和 PrintJobAttribute 两个接口。显然，一个打印请求可以包含一个需要多个拷贝的请求。反过来，打印作业的某个属性可能表示的是实际上打印出来的拷贝数量。这个拷贝数量可能很小，也许是因为打印机的限制或者是因为打印机的纸张已经用完了。

SupportedValuesAttribute 接口表示某个属性值反映的不是实际的打印请求或状态数据，而是某个服务程序的能力。例如，实现了 SupportedValuesAttribute 接口的 CopiesSupported 类，该类的对象可以用来描述某个打印机能够支持 1 ~ 99 份拷贝的打印输出。

图 11-65 显示了属性分层结构的类图。

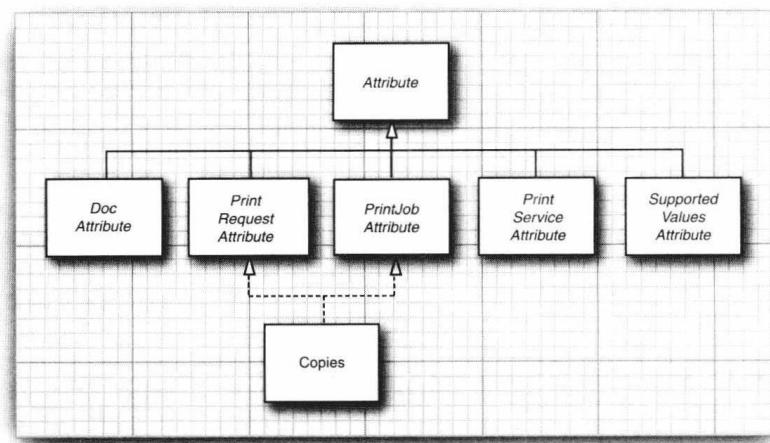


图 11-65 属性分层结构图

除了为各个属性定义的接口和类以外，打印服务程序 API 还为属性集定义了接口和类。父接口 AttributeSet 有四个子接口：

```
PrintRequestAttributeSet
DocAttributeSet
PrintServiceAttributeSet
PrintJobAttributeSet
```

对于每个这样的接口，都有一个实现类，因此会产生下面 5 个类：

```
HashSetAttributeSet
HashPrintRequestAttributeSet
HashDocAttributeSet
HashPrintServiceAttributeSet
HashPrintJobAttributeSet
```

图 11-66 显示了属性集分层结构的类图。

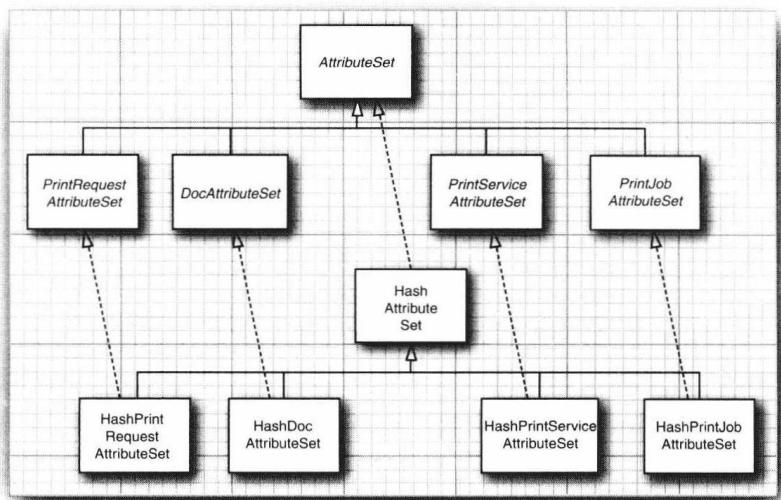


图 11-66 属性集的分层结构

例如，可以用如下方式构建一个打印请求属性集。

```
var attributes = new HashPrintRequest.AttributeSet();
```

当构建完属性集后，就不用担心使用 Hash 前缀的问题了。

为什么要配所有这些接口呢？因为，它们使得“检查属性是否被正确使用”成为可能。例如，Doc.AttributeSet 只接受实现了 DocAttribute 接口的对象，添加其他属性的任何尝试都会导致运行期错误的产生。

属性集是一个特殊的映射表，其键是 Class 类型的，而值是一个实现了 Attribute 接口的类。例如，如果要插入一个对象

```
new Copies(10)
```

到属性集中，那么它的键就是 Class 对象 Copies.class。该键被称为属性的类别。Attribute 接口声明了下面这样一个方法：

```
Class getCategory()
```

该方法就可以返回属性的类别。Copies 类定义了用以返回 Copies.class 对象的方法。但是，属性的类别和属性的类没有必要是相同的。

当将一个属性添加到属性集中时，属性的类别就会被自动地获取。你只需添加该属性的值：

```
attributes.add(new Copies(10));
```

如果后来添加了一个具有相同类别的另一个属性，那么新属性就会覆盖第一个属性。如果要检索一个属性，需要使用它的类别作为键，例如，

```
AttributeSet attributes = job.getAttributes();
var copies = (Copies) attribute.get(Copies.class);
```

最后，属性是按照它们拥有的值来进行组织的。Copies 属性能够拥有任何整数值。Copies 类继承了 IntegerSyntax 类，该类负责处理所有带有整数值的属性。getValue 方法将返回属性的整数值，例如，

```
int n = copies.getValue();
```

下面这些类：

```
TextSyntax  
DateTimeSyntax  
URISyntax
```

用于封装一个字符串、日期与时间，或者 URI（通用资源标识符）。

最后要说明的是，许多属性都能够接受数量有限的值。例如，PrintQuality 属性有三个设置值：draft（草稿质量）、normal（正常质量）和 high（高质量），它们用三个常量来表示：

```
PrintQuality.DRAFT  
PrintQuality.NORMAL  
PrintQuality.HIGH
```

拥有有限数量值的属性类继承了 EnumSyntax 类，该类提供了许多便利的方法，用来以类型安全的方式设置这些枚举。当使用这样的属性时，不必担心该机制，只需要将带有名字的值添加给属性集即可：

```
attributes.add(PrintQuality.HIGH);
```

下面的代码说明了如何来检查一个属性的值：

```
if (attributes.get(PrintQuality.class) == PrintQuality.HIGH)
```

```
...
```

表 11-5 列出了各个打印属性。表中的第二列列出了属性类的超类（例如，Copies 属性的 IntegerSyntax 类）或者是具有一组有限值属性的枚举值。最后四列表示该属性是否实现了 DocAttribute (DA)、PrintJobAttribute (PJA)、PrintRequestAttribute (PRA) 和 PrintServiceAttribute (PSA) 几个接口。

表 11-5 打印属性一览表

属性	超类或枚举常量	DA	PJA	PRA	PSA
Chromaticity	MONOCHROME, COLOR	✓	✓	✓	
ColorSupported	SUPPORTED, NOT_SUPPORTED				✓
Compression	COMPRESS, DEFLATE, GZIP, NONE	✓			
Copies	IntegerSyntax		✓	✓	
DateTimeAtCompleted	DateTimeSyntax	✓			
DateTimeAtCreation	DateTimeSyntax	✓			
DateTimeAtProcessing	DateTimeSyntax	✓			
Destination	URISyntax		✓	✓	
DocumentName	TextSyntax	✓			

( 续 )

属性	超类或枚举常量	DA	PJA	PRA	PSA
Fidelity	FIDELITY_TRUE, FIDELITY_FALSE		✓	✓	
Finishings	NONE, STAPLE, EDGE_STITCH, BIND, SADDLE_STITCH, COVER, . . .	✓	✓	✓	
JobHoldUntil	DateTimeSyntax		✓	✓	
JobImpressions	IntegerSyntax		✓	✓	
JobImpressionsCompleted	IntegerSyntax		✓		
JobKOctets	IntegerSyntax		✓	✓	
JobKOctetsProcessed	IntegerSyntax		✓		
JobMediaSheets	IntegerSyntax		✓	✓	
JobMediaSheetsCompleted	IntegerSyntax		✓		
JobMessageFromOperator	TextSyntax		✓		
JobName	TextSyntax		✓	✓	
JobOriginatingUserName	TextSyntax		✓		
JobPriority	IntegerSyntax		✓	✓	
JobSheets	STANDARD, NONE		✓	✓	
JobState	ABORTED, CANCELED, COMPLETED, PENDING, PENDING_HELD, PROCESSING, PROCESSING_STOPPED		✓		
JobStateReason	ABORTED_BY_SYSTEM, DOCUMENT_FORMAT_ERROR, 其他				
JobStateReasons	HashSet		✓		
MediaName	ISO_A4_WHITE, ISO_A4_TRANSPARENT, NA LETTER WHITE, NA LETTER TRANSPARENT	✓	✓	✓	
MediaSize	ISO.A0-ISO.A10, ISO.B0-ISO.B10, ISO.C0-ISO.C10, NA.LETTER, NA.LEGAL, 各种其他纸张和信封尺寸				
MediaSizeName	ISO_A0-ISO_A10, ISO_B0-ISO_B10, ISO_C0-ISO_C10, NA LETTER, NA LEGAL, 各种其他纸张和信封尺寸名称	✓	✓	✓	
MediaTray	TOP, MIDDLE, BOTTOM, SIDE, ENVELOPE, LARGE_CAPACITY, MAIN, MANUAL	✓	✓	✓	
MultipleDocumentHandling	SINGLE_DOCUMENT, SINGLE_DOCUMENT_NEW_SHEET, SEPARATE_DOCUMENTS_COLLATED_COPIES, SEPARATE_DOCUMENTS_UNCOLLATED_COPIES		✓	✓	
NumberOfDocuments	IntegerSyntax		✓		
NumberOfInterveningJobs	IntegerSyntax		✓		
NumberUp	IntegerSyntax	✓	✓	✓	
OrientationRequested	PORTRAIT, LANDSCAPE, REVERSE_PORTRAIT, REVERSE_LANDSCAPE	✓	✓	✓	
OutputDeviceAssigned	TextSyntax		✓		
PageRanges	SetOfInteger	✓	✓	✓	
PagesPerMinute	IntegerSyntax				✓
PagesPerMinuteColor	IntegerSyntax				✓
PDLOVERRIDE_SUPPORTED	ATTEMPTED, NOT_ATTEMPTED				✓
PresentationDirection	TORIGHT_TOBOTTOM, TORIGHT_TOTOP, TOBOTTOM_TORIGHT, TOBOTTOM_TOLEFT, TOLEFT_TOBOTTOM, TOLEFT_TOTOP, TOTOP_TORIGHT, TOTOP_TOLEFT		✓	✓	
PrinterInfo	TextSyntax				✓

(续)

属性	超类或枚举常量	DA	PJA	PRA	PSA
PrinterIsAcceptingJobs	ACCEPTING_JOBS, NOT_ACCEPTING_JOBS				✓
PrinterLocation	TextSyntax				✓
PrinterMakeAndModel	TextSyntax				✓
PrinterMessageFromOperator	TextSyntax				✓
PrinterMoreInfo	URISyntax				✓
PrinterMoreInfoManufacturer	URISyntax				✓
PrinterName	TextSyntax				✓
PrinterResolution	ResolutionSyntax	✓	✓	✓	
PrinterState	PROCESSING, IDLE, STOPPED, UNKNOWN				✓
PrinterStateReason	COVER_OPEN, FUSER_OVER_TEMP, MEDIA_JAM, 其他				
PrinterStateReasons	HashMap				
PrinterURI	URISyntax				✓
PrintQuality	DRAFT, NORMAL, HIGH	✓	✓	✓	
QueuedJobCount	IntegerSyntax				✓
ReferenceUriSchemesSupported	FILE, FTP, GOPHER, HTTP, HTTPS, NEWS, NNTP, WAIS				
RequestingUserName	TextSyntax				✓
Severity	ERROR, REPORT, WARNING				
SheetCollate	COLLATED, UNCOLLATED	✓	✓	✓	
Sides	ONE_SIDED, DUPLEX (= TWO_SIDED_LONG_EDGE), TUMBLE (= TWO_SIDED_SHORT_EDGE)	✓	✓	✓	

**注释：**可以看到，属性的数量很多，其中许多属性都是专用的。大多数属性都来源于因特网打印协议 1.1 版 (RFC 2911)。

**注释：**打印 API 的早期版本引入了 `JobAttributes` 和 `PageAttributes` 类，其目的与本节所介绍的打印属性类似。这些类现在已经弃用了。

#### API `javax.print.attribute.Attribute` 1.4

- `Class getCategory()`  
获取该属性的类别。
- `String getName()`  
获取该属性的名字。

#### API `javax.print.attribute.AttributeSet` 1.4

- `boolean add(Attribute attr)`  
向属性集中添加一个属性。如果集中有另一个属性和此属性有相同的类别，那么集中的属性被新添加的属性所取代。如果由于添加属性的操作改变了属性集，则返回 `true`。

- `Attribute get(Class category)`

检索带有指定属性类别键的属性，如果该属性不存在，则返回 `null`。

- `boolean remove(Attribute attr)`

- `boolean remove(Class category)`

从属性集中删除给定属性，或者删除具有指定类别的属性。如果由于这个操作改变了属性集，则返回 `true`。

- `Attribute[] toArray()`

返回一个带有该属性集中所有属性的数组。

**API `javax.print.PrintService 1.4`**

- `PrintServiceAttributeSet getAttributes()`

获取打印服务程序的属性。

**API `javax.print.DocPrintJob 1.4`**

- `PrintJobAttributeSet getAttributes()`

获取打印作业的属性。

现在，我们来到了本章的尾声，这长长的一章涵盖了高级 Swing 和 AWT 特性。在最后一章，我们将转而研究 Java 编程的另一个完全不同的方面：在同一台机器上与用其他编程语言编写的“本地”代码交互。

# 第 12 章 本地方法

- ▲ 从 Java 程序中调用 C 函数
- ▲ 数值参数与返回值
- ▲ 字符串参数
- ▲ 访问域
- ▲ 编码签名
- ▲ 调用 Java 方法
- ▲ 访问数组元素
- ▲ 错误处理
- ▲ 使用调用 API
- ▲ 完整的示例：访问 Windows 注册表

原则上说，“100% 纯 Java”的解决方案是非常好的，但有时你也会想要编写或使用其他语言的代码（这种代码通常称为本地代码）。

特别是在 Java 的早期阶段，许多人都认为使用 C 或 C++ 来加速 Java 应用中关键部分是个好主意。但是，实际上，这基本上是徒劳的。1996 年 JavaOne 会议上有一个演讲很明确地说明了这一点，来自 Sun Microsystems 的密码库的实现者报告说他们的加密函数的纯 Java 平台实现已臻化境。他们的代码确实没有已有的 C 实现快，但是事实证明这无关紧要。Java 平台实现比网络 I/O 要快得多，而后者是真正的瓶颈。

当然，求助于本地代码是有缺陷的。如果应用的某个部分是用其他语言编写的，那么就必须为需要支持的每个平台都提供一个单独的本地类库。用 C 或 C++ 编写的代码没有对通过使用无效指针所造成的内存覆写提供任何保护。编写本地代码很容易破坏你的程序，并感染操作系统。

因此，我们建议只有在必需的时候才使用本地代码。特别是在以下三种情况下，也许可以使用本地代码：

- 你的应用需要访问的系统特性和设备通过 Java 平台是无法实现的。
- 你已经有了大量的测试过和调试过的用另一种语言编写的代码，并且知道如何将其导出到所有的目标平台上。
- 通过基准测试，你发现所编写的 Java 代码比用其他语言编写的等价代码要慢得多。

Java 平台有一个用于和本地 C 代码进行互操作的 API，称为 Java 本地接口（JNI）。我们将在本章讨论 JNI 编程。

**C++** **C++ 注释：**你可以使用 C++ 代替 C 来编写本地方法。这样会有一些好处：类型检查会更严格一些，访问 JNI 函数会更便捷一些。然而，JNI 并不支持 Java 类和 C++ 类之间的任何映射机制。

## 12.1 从 Java 程序中调用 C 函数

假设你有一个 C 函数，它能为你实现某个功能，因为某种原因，你不想费事使用 Java 编程语言重新实现它。为了方便说明问题，我们从一个很简单的打印问候语的 C 函数入手。

Java 编程语言使用关键字 `native` 表示本地方法，而且很显然，你还需要在类中放置一个方法。其结果显示在程序清单 12-1 中。

关键字 `native` 提醒编译器该方法将在外部定义。当然，本地方法不包含任何 Java 编程语言编写的代码，而且方法头后面直接跟着一个表示终结的分号。因此，本地方法声明看上去和抽象方法声明类似。

**程序清单 12-1** helloNative/HelloNative.java

```

1 /**
2  * @version 1.11 2007-10-26
3  * @author Cay Horstmann
4 */
5 class HelloNative
6 {
7     public static native void greeting();
8 }
```

**注释：**与前一章一样，为了保持样例的简单性，我们在这里也不使用包。

在这个特定示例中，本地方法也被声明为 `static`。本地方法既可以是静态的也可以是非静态的，使用静态方法是因为我们此刻还不想处理参数传递。

你实际上可以编译这个类，但是在程序中使用它时，虚拟机就会告诉你它不知道如何找到 `greeting`，它会报告一个 `UnsatisfiedLinkError` 异常。为了实现本地代码，需要编写一个相应的 C 函数，你必须完全按照 Java 虚拟机预期的那样来命名这个函数。其规则是：

1. 使用完整的 Java 方法名，比如：`HelloNative.greeting`。如果该类属于某个包，那么在前面添加包名，比如：`com.horstmann.HelloNative.greeting`。
2. 用下划线替换掉所有的句号，并加上 `Java_` 前缀，例如，`Java_HelloNative_greeting` 或 `Java_com_horstmann_HelloNative_greeting`。
3. 如果类名含有非 ASCII 字母或数字，如：'\_'，'\$' 或是大于 '\u007F' 的 Unicode 字符，用 `_0xxxx` 来替代它们，`xxxx` 是该字符的 Unicode 值的 4 个十六进制数序列。

**注释：**如果你重载了本地方法，也就是说，你用相同的名字提供了多个本地方法，那么你必须在名称后附加两个下划线，后面再加上已编码的参数类型。在本章后面，我们将描述参数类型的编码方法。例如，如果你有一个本地方法 `greeting` 和另一个本地方法 `greeting(int repeat)`，那么，第一个称为 `Java_HelloNative_greeting_`，第二个称为 `Java_HelloNative_greeting_I`。

实际上，没人会手工完成这些操作。相反，你应该用 `-h` 标志运行 `javac`，并提供头文件

放置的目录：

```
javac -h . HelloNative.java
```

这条命令在当前目录中创建了一个名为 `HelloNative.h` 的头文件，正如程序清单 12-2 所示。

### 程序清单 12-2 helloNative/HelloNative.h

```
1 /* DO NOT EDIT THIS FILE - it is machine generated */
2 #include <jni.h>
3 /* Header for class HelloNative */
4
5 #ifndef _Included_HelloNative
6 #define _Included_HelloNative
7 #ifdef __cplusplus
8 extern "C" {
9 #endif
10 /*
11  * Class:      HelloNative
12  * Method:     greeting
13  * Signature:  ()V
14  */
15 JNIEXPORT void JNICALL Java_HelloNative_greeting
16     (JNIEnv *, jclass);
17
18 #ifdef __cplusplus
19 }
20 #endif
21 #endif
```

如你所见，这个文件包含了函数 `Java_HelloNative_greeting` 的声明（宏 `JNIEXPORT` 和 `JNICALL` 是在头文件 `jni.h` 中定义的，它们为那些来自动态装载库的导出函数标明了依赖于编译器的说明符）。

现在，需要将函数原型从头文件中复制到源文件中，并且给出函数的实现代码，如程序清单 12-3 所示。

### 程序清单 12-3 helloNative/HelloNative.c

```
1 /*
2  * @version 1.10 1997-07-01
3  * @author Cay Horstmann
4 */
5
6 #include "HelloNative.h"
7 #include <stdio.h>
8
9 JNIEXPORT void JNICALL Java_HelloNative_greeting(JNIEnv* env, jclass cl)
10 {
11     printf("Hello Native World!\n");
12 }
```

在这个简单的函数中，我们忽略了 `env` 和 `cl` 参数。后面你会看到它们的用处。

**C++ 注释：**你可以使用 C++ 实现本地方法。然而，那样你必须将实现本地方法的函数声明为 `extern"C"`（这可以阻止 C++ 编译器混编方法名）。例如：

```
extern "C"
JNIEXPORT void JNICALL Java_HelloNative_greeting(JNIEnv* env, jclass cl)
{
    cout << "Hello, Native World!" << endl;
}
```

将本地 C 代码编译到一个动态装载库中，具体方法依赖于编译器。

例如，Linux 下的 Gnu C 编译器，使用如下命令：

```
gcc -fPIC -I jdk/include -I jdk/include/linux -shared -o libHelloNative.so HelloNative.c
```

用 Windows 下的微软编译器，命令是：

```
cl -I jdk\include -I jdk\include\win32 -LD HelloNative.c -FeHelloNative.dll
```

这里 `jdk` 是含有 JDK 的目录。

**提示：**如果你要从命令 shell 中使用微软的编译器，首先要运行批处理文件 `vcvars32.bat` 或 `vcvarsall.bat`。这个批处理文件设置了编译器需要的路径和环境变量。你可以在目录 `c:\Program Files\Microsoft Visual Studio .14.0\Common7\Tools`，或类似位置找到该文件，细节请查看 Visual Studio 的文档。

也可以使用可从 <http://www.cygwin.com> 处免费获取的 Cygwin 编程环境。它包含了 GNU C 编译器和 Windows 下的 UNIX 风格编程的库。使用 Cygwin 时，用以下命令：

```
gcc -mno-cygwin -D __int64="long long" -I jdk/include/ -I jdk/include/win32 \
-shared -Wl,--add-stdcall-alias -o HelloNative.dll HelloNative.c
```

整个命令应该键入在同一行中。

**注释：**Windows 版本的头文件 `jni_md.h` 含有如下类型声明：

```
typedef __int64 jlong;
```

它是专门用于微软编译器的。如果你使用的是 GNU 编译器，那么你可能需要编辑这个文件，例如：

```
#ifdef __GNUC__
    typedef long long jlong;
#else
    typedef __int64 jlong;
#endif
```

或者，如编译器调用的示例那样，使用 `-D __int64="long long"` 进行编译。

最后，我们要在程序中添加一个对 `System.loadLibrary` 方法的调用。为了确保虚拟机在第一次使用该类之前就会装载这个库，需要使用静态初始化代码块，如程序清单 12-4 所示。

## 程序清单 12-4 helloNative/HelloNativeTest.java

```

1  /**
2   * @version 1.11 2007-10-26
3   * @author Cay Horstmann
4   */
5  class HelloNativeTest
6  {
7      public static void main(String[] args)
8      {
9          HelloNative.greeting();
10     }
11
12     static
13     {
14         System.loadLibrary("HelloNative");
15     }
16 }

```

图 12-1 给出了对本地代码处理的总结。

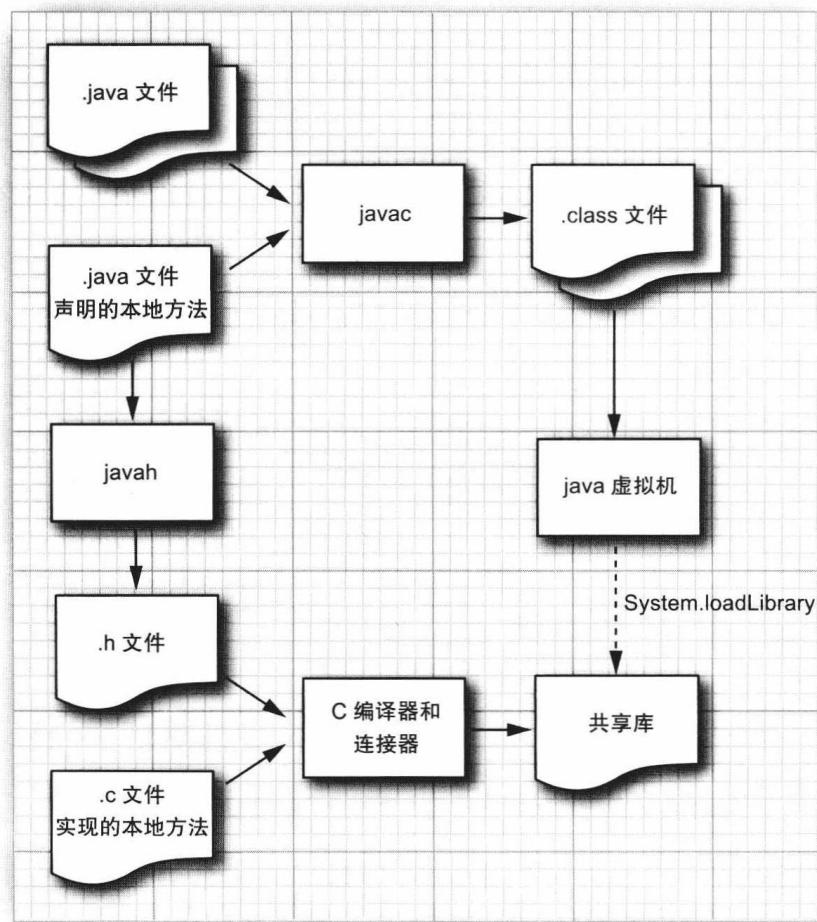


图 12-1 处理本地代码

如果编译并运行该程序，终端窗口会显示消息“Hello, Native World!”。

**注释：**如果运行在 Linux 下，必须把当前目录添加到库路径中。实现方式可以是通过设置 `LD_LIBRARY_PATH` 环境变量：

```
export LD_LIBRARY_PATH=.:${LD_LIBRARY_PATH}
```

或者是设置 `java.library.path` 系统属性：

```
java -Djava.library.path=. HelloNativeTest
```

当然，这个消息本身并不会给人留下深刻印象。然而，如果你记得这个信息是由 C 的 `printf` 命令产生而不是由任何 Java 编程语言代码产生的话，你就会明白我们已经在连接两种语言上走出了第一步。

总之，遵循下面的步骤就可以将一个本地方法链接到 Java 程序中：

1. 在 Java 类中声明一个本地方法。
2. 运行 `javadoc` 以获得包含该方法的 C 声明的头文件。
3. 用 C 实现该本地方法。
4. 将代码置于共享类库中。
5. 在 Java 程序中加载该类库。

#### API `java.lang.System` 1.0

- `void loadLibrary(String libname)`

装载指定名字的库，该库位于库搜索路径中。定位该库的确切方法依赖于操作系统。

**注释：**一些本地代码的共享库必须先运行初始化代码。你可以把初始化代码放到 `JNI_OnLoad` 方法中。类似地，如果你提供该方法，当虚拟机关闭时，将会调用 `JNI_OnUnload` 方法。它们的原型是：

```
jint JNI_OnLoad(JavaVM* vm, void* reserved);
void JNI_OnUnload(JavaVM* vm, void* reserved);
```

`JNI_OnLoad` 方法要返回它所需的虚拟机的最低版本，例如：`JNI_VERSION_1_2`。

## 12.2 数值参数与返回值

当在 C 和 Java 之间传递数字时，应该知道它们彼此之间的对应类型。例如，C 也有 `int` 和 `long` 的数据类型，但是它们的实现却是取决于平台的。在一些平台上，`int` 类型是 16 位的，在另外一些平台上是 32 位的。然而，在 Java 平台上 `int` 类型总是 32 位的整数。基于这个原因，Java 本地接口定义了 `jint`、`jlong` 等类型。

表 12-1 显示了 Java 数据类型和 C 数据类型的对应关系。

表 12-1 Java 数据类型和 C 数据类型

Java 编程语言	C 编程语言	字节	Java 编程语言	C 编程语言	字节
boolean	jboolean	1	int long float double	jint	4
byte	jbyte	1		jlong	8
char	jchar	2		jfloat	4
short	jshort	2		jdouble	8

在头文件 jni.h 中，这些类型被 `typedef` 语句声明为在目标平台上等价的类型。该头文件还定义了常量 `JNI_FALSE = 0` 和 `JNI_TRUE = 1`。

直到 Java 5.0，Java 才有了与 C 语言的 `printf` 函数相类似的方法。在下面的示例中，我们假设你依然坚持使用古老版本的 JDK，并且决定通过调用本地方法中的 C 的 `printf` 函数来实现同样的功能。

程序清单 12-5 给出了一个名为 `Printf1` 的类，它使用本地方法来打印给定域宽度和精度的浮点数。

#### 程序清单 12-5 printf1/Printf1.java

```

1 /**
2  * @version 1.10 1997-07-01
3  * @author Cay Horstmann
4 */
5 class Printf1
6 {
7     public static native int print(int width, int precision, double x);
8
9     static
10    {
11        System.loadLibrary("Printf1");
12    }
13 }
```

注意，用 C 实现该方法时，所有的 `int` 和 `double` 参数都要转换成 `jint` 和 `jdouble`，如程序清单 12-6 所示。

#### 程序清单 12-6 printf1/Printf1.c

```

1 /**
2  * @version 1.10 1997-07-01
3  * @author Cay Horstmann
4 */
5
6 #include "Printf1.h"
7 #include <stdio.h>
8
9 JNIEXPORT jint JNICALL Java_Printf1_print(JNIEnv* env, jclass cl,
10     jint width, jint precision, jdouble x)
11 {
12     char fmt[30];
```

```

13     jint ret;
14     sprintf(fmt, "%%d.%df", width, precision);
15     ret = printf(fmt, x);
16     fflush(stdout);
17     return ret;
18 }
```

该函数只是装配了变量 fmt 中的格式字符串 "%w.pf"，然后调用 printf 函数，接着返回打印出的字符的个数。

程序清单 12-7 给出了验证 Printf1 类的测试程序。

**程序清单 12-7 printf1/Printf1Test.java**

```

1 /**
2  * @version 1.10 1997-07-01
3  * @author Cay Horstmann
4 */
5 class Printf1Test
6 {
7     public static void main(String[] args)
8     {
9         int count = Printf1.print(8, 4, 3.14);
10        count += Printf1.print(8, 4, count);
11        System.out.println();
12        for (int i = 0; i < count; i++)
13            System.out.print("-");
14        System.out.println();
15    }
16 }
```

### 12.3 字符串参数

接着，我们要考虑怎样把字符串传入、传出本地方法。字符串在这两种语言中很不一样，Java 编程语言中的字符串是 UTF-16 编码点的序列，而 C 的字符串则是以 null 结尾的字节序列。JNI 有两组操作字符串的函数，一组把 Java 字符串转换成“modified UTF-8”字节序列，另一组将它们转换成 UTF-16 数值的数组，也就是说转换成 jchar 数组。（UTF-8、“modified UTF-8”和 UTF-16 格式都已经在第 2 章中讨论过了，请回忆一下，“modified UTF-8”编码保持 ASCII 字符不变，但是其他所有 Unicode 字符被编码为多字节序列。）

**注释：**标准 UTF-8 编码和“modified UTF-8”编码的差别仅在于编码大于 0xFFFF 的增补字符。在标准 UTF-8 编码中，这些字符编码为 4 字节序列；然而，在 modified UTF-8 编码中，这些字符首先被编码为一对 UTF-16 编码的“替代品”，然后再对每个替代品用 UTF-8 编码，总共产生 6 字节编码。这有点笨拙，但这是个由历史原因造成的意外，编写 Java 虚拟机规范的时候 Unicode 还局限在 16 位。

如果你的 C 代码已经使用了 Unicode，那么你可以使用第二组转换函数。另一方面，如果你的字符串都仅限于使用 ASCII 字符，那么就可以使用“modified UTF-8”转换函数。

带有字符串参数的本地方法实际上都要接受一个 `jstring` 类型的值，而带有字符串参数返回值的本地方法必须返回一个 `jstring` 类型的值。JNI 函数将读入并构造出这些 `jstring` 对象。例如，`NewStringUTF` 函数会从包含 ASCII 字符的字符数组，或者是更一般的“modified UTF-8”编码的字节序列中，创建一个新的 `jstring` 对象。

JNI 函数有一个有些古怪的调用惯例。下面是对 `NewStringUTF` 函数的一个调用：

```
JNIEXPORT jstring JNICALL Java_HelloNative_getGreeting(JNIEnv* env, jclass cl)
{
    jstring jstr;
    char greeting[] = "Hello, Native World\n";
    jstr = (*env)->NewStringUTF(env, greeting);
    return jstr;
}
```

 **注释：**本章中的所有代码都是 C 代码，除了指明为别的代码。

所有对 JNI 函数的调用都使用到了 `env` 指针，该指针是每一个本地方法的第一个参数。`env` 指针是指向函数指针表的指针（参见图 12-2）。所以，你必须在每个 JNI 调用前面加上 `(*env)->`，以便解析对函数指针的引用。而且，`env` 是每个 JNI 函数的第一个参数。

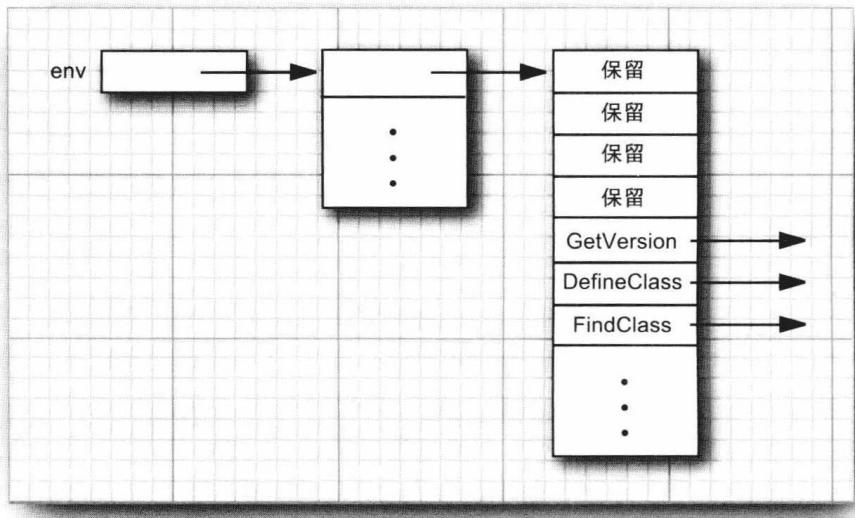


图 12-2 `env` 指针

 **C++ 注释：**C++ 中对 JNI 函数的访问要简单一些。`JNIEnv` 类的 C++ 版本有一个内联成员函数，它负责帮你查找函数指针。例如，你可以这样调用 `NewStringUTF` 函数：

```
jstr = env->NewStringUTF(greeting);
```

注意，这里删除了该调用的参数列表里的 `JNIEnv` 指针。

NewStringUTF 函数可以用来构造一个新的 `jstring`, 而读取现有 `jstring` 对象的内容, 需要使用 `GetStringUTFChars` 函数。该函数返回指向描述字符串的“modified UTF-8”字符的 `const jbyte*` 指针。注意, 具体的虚拟机可以为其内部的字符串表示方法自由地选择编码机制。所以, 你可以得到实际的 Java 字符串的字符指针。因为 Java 字符串是不可变的, 所以慎重处理 `const` 就显得非常重要, 不要试图将数据写到该字符数组中。另一方面, 如果虚拟机使用 UTF-16 或 UTF-32 字符作为其内部字符串的表示, 那么该函数会分配一个新的内存块来存储等价的“modified UTF-8”编码字符。

虚拟机必须知道你何时使用完字符串, 这样它就能进行垃圾回收(垃圾回收器是在一个独立线程中运行的, 它能够中断本地方法的执行)。基于这个原因, 你必须调用 `ReleaseStringUTFChars` 函数。

另外, 可以通过调用 `GetStringRegion` 或 `GetStringUTFRegion` 方法来提供你自己的缓存, 以存放字符串的字符。

最后 `GetStringUTFLength` 函数返回字符串的“modified UTF-8”编码所需的字符个数。

 **注释:** 你可以在 <http://docs.oracle.com/javase/7/docs/technotes/guides/jni> 处找到 JNI API。

## API 从 C 代码访问 Java 字符串

- `jstring NewStringUTF(JNIEnv* env, const char bytes[])`

根据以全 0 字节结尾的“modified UTF-8”字节序列, 返回一个新的 Java 字符串对象, 或者当字符串无法构建时, 返回 `NULL`。

- `jsize GetStringUTFLength(JNIEnv* env, jstring string)`

返回进行 UTF-8 编码所需的字节个数(作为终止符的全 0 字节不计入内)。

- `const jbyte* GetStringUTFChars(JNIEnv* env, jstring string, jboolean* isCopy)`

返回指向字符串的“modified UTF-8”编码的指针, 或者当不能构建字符数组时返回 `NULL`。直到 `ReleaseStringUTFChars` 函数调用前, 该指针一直有效。`isCopy` 指向一个 `jboolean`, 如果进行了复制, 则填入 `JNI_TRUE`, 否则填入 `JNI_FALSE`。

- `void ReleaseStringUTFChars(JNIEnv* env, jstring string, const jbyte bytes[])`

通知虚拟机本地代码不再需要通过 `bytes`(`GetStringUTFChars` 返回的指针)访问 Java 字符串。

- `void GetStringRegion(JNIEnv *env, jstring string, jsize start, jsize length, jchar *buffer)`

将一个 UTF-16 双字节序列从字符串复制到用户提供的尺寸至少大于  $2 \times \text{length}$  的缓存中。

- `void GetStringUTFRegion(JNIEnv *env, jstring string, jsize start, jsize length, jbyte *buffer)`

将一个“modified UTF-8”字符序列从字符串复制到用户提供的缓存中。为了存放要复制的字节, 该缓存必须足够长。最坏情况下, 要复制  $3 \times \text{length}$  个字节。

- `jstring NewString(JNIEnv* env, const jchar chars[], jsize length)`

根据 Unicode 字符串返回一个新的 Java 字符串对象，或者在不能构建时返回 NULL。

- `jsize GetStringLength(JNIEnv* env, jstring string)`

返回字符串中字符的个数。

- `const jchar* GetStringChars(JNIEnv* env, jstring string, jboolean* isCopy)`

返回指向字符串的 Unicode 编码的指针，或者当不能构建字符数组时返回 NULL。直到 `ReleaseStringChars` 函数调用前，该指针一直有效。`isCopy` 要么为 NULL；要么在进行了复制时，指向用 `JNI_TRUE` 填充的 `jboolean`，否则指向用 `JNI_FALSE` 填充的 `jboolean`。

- `void ReleaseStringChars(JNIEnv* env, jstring string, const jchar chars[])`

通知虚拟机本地代码不再需要通过 `chars` (`GetStringChars` 返回的指针) 访问 Java 字符串。

让我们使用这些函数来编写一个调用 C 函数 `sprintf` 的类，我们要像程序清单 12-8 所示那样调用这个函数。

#### 程序清单 12-8 printf2/Printf2Test.java

```

1 /**
2  * @version 1.10 1997-07-01
3  * @author Cay Horstmann
4 */
5 class Printf2Test
6 {
7     public static void main(String[] args)
8     {
9         double price = 44.95;
10        double tax = 7.75;
11        double amountDue = price * (1 + tax / 100);
12
13        String s = Printf2.sprintf("Amount due = %8.2f", amountDue);
14        System.out.println(s);
15    }
16 }
```

程序清单 12-9 给出了带有本地 `sprint` 方法的类。

因此，格式化浮点数的 C 函数原型如下：

```
JNICALL jstring Java_Printf2_sprint(JNIEnv* env, jclass cl,
                                         jstring format, jdouble x)
```

#### 程序清单 12-9 printf2/Printf2.java

```

1 /**
2  * @version 1.10 1997-07-01
3  * @author Cay Horstmann
4 */
5 class Printf2
6 {
7     public static native String sprint(String format, double x);
8
9     static
10    {
```

```

11     System.loadLibrary("Printf2");
12 }
13 }
```

程序清单 12-10 给出了 C 的实现代码。注意，我们通过调用 GetStringUTFChars 来读取格式参数，通过调用 NewStringUTF 来产生返回值，通过调用 ReleaseStringUTFChars 来通知虚拟机不再需要访问该字符串。

### 程序清单 12-10 printf2/Printf2.c

```

1  /**
2   * @version 1.10 1997-07-01
3   * @author Cay Horstmann
4  */
5
6 #include "Printf2.h"
7 #include <string.h>
8 #include <stdlib.h>
9 #include <float.h>
10
11 /**
12  * @param format a string containing a printf format specifier
13  * (such as "%8.2f"). Substrings "%%" are skipped.
14  * @return a pointer to the format specifier (skipping the '%')
15  * or NULL if there wasn't a unique format specifier
16 */
17 char* find_format(const char format[])
18 {
19     char* p;
20     char* q;
21
22     p = strchr(format, '%');
23     while (p != NULL && *(p + 1) == '%') /* skip %% */
24         p = strchr(p + 2, '%');
25     if (p == NULL) return NULL;
26     /* now check that % is unique */
27     p++;
28     q = strchr(p, '%');
29     while (q != NULL && *(q + 1) == '%') /* skip %% */
30         q = strchr(q + 2, '%');
31     if (q != NULL) return NULL; /* % not unique */
32     q = p + strspn(p, " -0+#"); /* skip past flags */
33     q += strspn(q, "0123456789"); /* skip past field width */
34     if (*q == '.') { q++; q += strspn(q, "0123456789"); }
35     /* skip past precision */
36     if (strchr("eEfFgG", *q) == NULL) return NULL;
37     /* not a floating-point format */
38     return p;
39 }
40
41 JNIEXPORT jstring JNICALL Java_Printf2_sprint(JNIEnv* env, jclass cl,
42     jstring format, jdouble x)
43 {
```

```

44     const char* cformat;
45     char* fmt;
46     jstring ret;
47
48     cformat = (*env)->GetStringUTFChars(env, format, NULL);
49     fmt = find_format(cformat);
50     if (fmt == NULL)
51         ret = format;
52     else
53     {
54         char* cret;
55         int width = atoi(fmt);
56         if (width == 0) width = DBL_DIG + 10;
57         cret = (char*) malloc(strlen(cformat) + width);
58         sprintf(cret, cformat, x);
59         ret = (*env)->NewStringUTF(env, cret);
60         free(cret);
61     }
62     (*env)->ReleaseStringUTFChars(env, format, cformat);
63     return ret;
64 }
```

在本函数中，我们选择简化错误处理。如果打印浮点数的格式代码不是 %w.pc 形式的（其中 c 是 e、E、f、g 或 G 中的一个），那么我们将不对数字进行格式化。后面我们会介绍如何让本地方法抛出异常。

## 12.4 访问域

目前为止你看到的所有本地方法都是带有数字或字符串参数的静态方法。下面，我们考虑在对象上进行操作的本地方法。作为一个练习，我们用本地方法实现卷 I 第 4 章中的 Employee 类的一个方法。通常情况下并不需要这么做，但是这里演示了当你需要的时候可以怎样从本地方法访问对象域。

### 12.4.1 访问实例域

为了了解怎样从本地方法访问实例域，我们用 Java 重新实现了 raiseSalary 方法。其代码很简单：

```

public void raiseSalary(double byPercent)
{
    salary *= 1 + byPercent / 100;
}
```

让我们重写代码，使其成为一个本地方法。与此前的本地方法不同，它并不是一个静态方法。运行 javac-h 给出以下原型：

```
JNIEXPORT void JNICALL Java_Employee_raiseSalary(JNIEnv *, jobject, jdouble);
```

注意，第二个参数不再是 jclass 类型而是 jobject 类型。实际上，它和 this 引用等价。静态方法得到的是类的引用，而非静态方法得到的是对隐式的 this 参数对象的引用。

现在，我们访问隐式参数的 salary 域。在 Java1.0 中“原生的”Java 到 C 的绑定中，这很简单，程序员可以直接访问对象数据域。然而，直接访问要求虚拟机暴露它们的内部数据布局。基于这个原因，JNI 要求程序员通过调用特殊的 JNI 函数来获取和设置数据的值。

在我们的例子里，要使用 GetdoubleField 和 SetDoubleField 函数，因为 salary 是 double 类型的。对于其他类型，可以使用的函数有：GetIntField/SetIntField、GetObjectField/SetObjectField 等等。其通用语法是：

```
x = (*env)->GetXxxField(env, this_obj, fieldID);
(*env)->SetXxxField(env, this_obj, fieldID, x);
```

这里，fieldID 是一个特殊类型 jfieldID 的值，jfieldID 标识结构中的一个域，而 Xxx 代表 Java 数据类型（Object、Boolean、Byte 或其他）。为了获得 fieldID，必须先获得一个表示类的值，有两种方法可以实现此目的。GetObjectClass 函数可以返回任意对象的类。例如：

```
jclass class_Employee = (*env)->GetObjectClass(env, this_obj);
```

FindClass 函数可以让你以字符串形式来指定类名（有点奇怪的是，要以 / 代替句号作为包名之间的分隔符）。

```
jclass class_String = (*env)->FindClass(env, "java/lang/String");
```

之后，可以使用 GetFieldID 函数来获得 fieldID。必须提供域的名字、它的签名以及它的类型的编码。例如，下面是从 salary 域得到域 ID 的代码：

```
jfieldID id_salary = (*env)->GetFieldID(env, class_Employee, "salary", "D");
```

字符串 "D" 表示类型是 double。你将在下一节中学习到编码签名的全部规则。

你可能会认为访问数据域相当令人费解。JNI 的设计者不想把数据域直接暴露在外，所以他们不得不提供获取和设置数据域值的函数。为了使这些函数的开销最小化，从域名计算域 ID（代价最大的一个步骤）被分解出来作为单独的一步操作。也就是说，如果你反复地获取和设置一个特定的域，你计算域标识符的开销就只有一次。

让我们把各部分汇总起来，下面的代码以本地方法形式重新实现了 raiseSalary 方法。

```
JNIEXPORT void JNICALL Java_Employee_raiseSalary(JNIEnv* env, jobject this_obj,
    jdouble byPercent)
{
    /* get the class */
    jclass class_Employee = (*env)->GetObjectClass(env, this_obj);

    /* get the field ID */
    jfieldID id_salary = (*env)->GetFieldID(env, class_Employee, "salary", "D");

    /* get the field value */
    jdouble salary = (*env)->GetDoubleField(env, this_obj, id_salary);

    salary *= 1 + byPercent / 100;

    /* set the field value */
    (*env)->SetDoubleField(env, this_obj, id_salary, salary);
}
```

**!** 警告：类引用只在本地方法返回之前有效。因此，不能在你的代码中缓存 GetObjectClass 的返回值。不要将类引用保存下来以供以后的方法调用重复使用。必须在每次执行本地方法时都调用 GetObjectClass。如果你无法忍受这一点，必须调用 NewGlobalRef 来锁定该引用：

```
static jclass class_X = 0;
static jfieldID id_a;

.

if (class_X == 0)
{
    jclass cx = (*env)->GetObjectClass(env, obj);
    class_X = (*env)->NewGlobalRef(env, cx);
    id_a = (*env)->GetFieldID(env, cls, "a", "...");
}
```

现在，你可以在后面的调用中使用类引用和域 ID 了。当你结束对类的使用时，务必调用：  
(\*env)->DeleteGlobalRef(env, class\_X);

程序清单 12-11 和程序清单 12-12 给出了测试程序和 Employee 类的 Java 代码。程序清单 12-13 包含了本地 raiseSalary 方法的 C 代码。

#### 程序清单 12-11 employee/EmployeeTest.java

```
1 /**
2  * @version 1.11 2018-05-01
3  * @author Cay Horstmann
4 */
5
6 public class EmployeeTest
7 {
8     public static void main(String[] args)
9     {
10         var staff = new Employee[3];
11
12         staff[0] = new Employee("Harry Hacker", 35000);
13         staff[1] = new Employee("Carl Cracker", 75000);
14         staff[2] = new Employee("Tony Tester", 38000);
15
16         for (Employee e : staff)
17             e.raiseSalary(5);
18         for (Employee e : staff)
19             e.print();
20     }
21 }
```

#### 程序清单 12-12 employee/Employee.java

```
1 /**
2  * @version 1.10 1999-11-13
3  * @author Cay Horstmann
4 */
5
```

```

6 public class Employee
7 {
8     private String name;
9     private double salary;
10
11    public native void raiseSalary(double byPercent);
12
13    public Employee(String n, double s)
14    {
15        name = n;
16        salary = s;
17    }
18
19    public void print()
20    {
21        System.out.println(name + " " + salary);
22    }
23
24    static
25    {
26        System.loadLibrary("Employee");
27    }
28 }

```

### 程序清单 12-13 employee/Employee.c

```

1 /**
2  * @version 1.10 1999-11-13
3  * @author Cay Horstmann
4 */
5
6 #include "Employee.h"
7
8 #include <stdio.h>
9
10 JNIEXPORT void JNICALL Java_Employee_raiseSalary(
11     JNIEnv* env, jobject this_obj, jdouble byPercent)
12 {
13     /* get the class */
14     jclass class_Employee = (*env)->GetObjectClass(env, this_obj);
15
16     /* get the field ID */
17     jfieldID id_salary = (*env)->GetFieldID(env, class_Employee, "salary", "D");
18
19     /* get the field value */
20     jdouble salary = (*env)->GetDoubleField(env, this_obj, id_salary);
21
22     salary *= 1 + byPercent / 100;
23
24     /* set the field value */
25     (*env)->SetDoubleField(env, this_obj, id_salary, salary);
26 }
27

```

### 12.4.2 访问静态域

访问静态域和访问非静态域类似，要使用 `GetStaticFieldID` 和 `GetStaticXxxField`/ `SetStaticXxxField` 函数。它们几乎与非静态的情形一样，只有两个区别：

- 由于没有对象，所以必须使用 `FindClass` 代替 `GetObjectClass` 来获得类引用。
- 访问域时，要提供类而非实例对象。

例如，下面给出的是怎样得到 `System.out` 的引用的代码：

```
/* get the class */
jclass class_System = (*env)->FindClass(env, "java/lang/System");

/* get the field ID */
jfieldID id_out = (*env)->GetStaticFieldID(env, class_System, "out",
    "Ljava/io/PrintStream;");

/* get the field value */
jobject obj_out = (*env)->GetStaticObjectField(env, class_System, id_out);
```

### API 访问实例域

- `jfieldID GetFieldID(JNIEnv *env, jclass cl, const char name[], const char fieldSignature[])`  
返回类中一个域的标识符。
- `Xxx GetXxxField(JNIEnv *env, jobject obj, jfieldID id)`  
返回域的值。域类型 `Xxx` 是 `Object`、`Boolean`、`Byte`、`Char`、`Short`、`Int`、`Long`、`Float` 或 `Double` 之一。
- `void SetXxxField(JNIEnv *env, jobject obj, jfieldID id, Xxx value)`  
把某个域设置为一个新值。域类型 `Xxx` 是 `Object`、`Boolean`、`Byte`、`Char`、`Short`、`Int`、`Long`、`Float` 或 `Double` 之一。
- `jfieldID GetStaticFieldID(JNIEnv *env, jclass cl, const char name[], const char fieldSignature[])`  
返回某类型的一个静态域的标识符。
- `Xxx GetStaticXxxField(JNIEnv *env, jclass cl, jfieldID id)`  
返回某静态域的值。域类型 `Xxx` 是 `Object`、`Boolean`、`Byte`、`Char`、`Short`、`Int`、`Long`、`Float` 或 `Double` 之一。
- `void SetStaticXxxField(JNIEnv *env, jclass cl, jfieldID id, Xxx value)`  
把某个静态域设置为一个新值。域类型 `Xxx` 是 `Object`、`Boolean`、`Byte`、`Char`、`Short`、`Int`、`Long`、`Float` 或 `Double` 之一。

## 12.5 编码签名

为了访问实例域和调用用 Java 编程语言定义的方法，你必须学习将数据类型的名称和方法签名进行“混编”的规则（方法签名描述了参数和该方法返回值的类型）。下面是编码方案：

B	byte
C	char
D	double
F	float
I	int
J	long
Lclassname;	类的类型
S	short
V	void
Z	boolean

为了描述数组类型，要使用 [。例如，一个字符串数组如下：

[Ljava/lang/String;

一个 float[][] 可以描述为：

[[F

要建立一个方法的完整签名，需要把括号内的参数类型都列出来，然后列出返回值类型。例如，一个接收两个整型参数并返回一个整数的方法编码为：

(II)I

12.3 节中的 Sprint 方法有下面的混编签名：

(Ljava/lang/String;D)Ljava/lang/String;

也就是说，该方法接收一个 String 和一个 double，返回值是一个 String。

注意，在 L 表达式结尾处的分号是类型表达式的终止符，而不是参数之间的分隔符。例如，构造器：

Employee(java.lang.String, double, java.util.Date)

具有如下签名：

"(Ljava/lang/String;D)V"

注意，在 D 和 Ljava/util/Date; 之间没有分隔符。另外要注意在这个编码方案中，必须用 / 代替 . 来分隔包和类名。结尾的 V 表示返回类型为 void。即使对 Java 的构造器没有指定返回类型，也需要将 V 添加到虚拟机签名中。

 提示：可以使用带有选项 -s 的 javap 命令来从类文件中产生方法签名。例如，运行

```
javap -s -private Employee
```

可以得到以下显示所有域和方法的输出：

```
Compiled from "Employee.java"
public class Employee extends java.lang.Object{
    private java.lang.String name;
        Signature: Ljava/lang/String;
    private double salary;
        Signature: D
    public Employee(java.lang.String, double);
```

```

Signature: (Ljava/lang/String;D)V
public native void raiseSalary(double);
    Signature: (D)V
public void print();
    Signature: ()V
static {};
    Signature: ()V
}

```

**注释：**没有任何理由强迫程序员使用这种混编方案来描述签名。本地调用机制的设计者可以非常容易地编写一个函数来读取 Java 编程语言风格的签名，比如 void (int,java.lang.String)，并且将它们编码为他们喜欢的某种内部表示法。再者，使用混编签名使你能够分享接近虚拟机的编程奥秘。

## 12.6 调用 Java 方法

当然，Java 编程语言的函数可以调用 C 函数，这正是本地方法要做的。我们能不能换一种方式呢？为什么我们要这么做？答案是，本地方法常常需要从传递给它的对象那里得到某种服务。我们首先介绍非静态方法如何进行这种操作，然后介绍静态方法如何进行这种操作。

### 12.6.1 实例方法

作为从本地代码调用 Java 方法的一个例子，我们先增强 `Printf` 类，给它增加一个与 C 函数 `fprintf` 类似的方法。也就是说，它能够在任意 `PrintWriter` 对象上打印一个字符串。下面是用 Java 编写的该方法的定义：

```

class Printf3
{
    public native static void fprintf(PrintWriter out, String s, double x);
    ...
}

```

我们首先把要打印的字符串组装成一个 `String` 对象 `str`，就像我们在 `sprint` 方法中已经实现的那样。然后，我们从实现本地方法的 C 函数中调用 `PrintWriter` 类的 `print` 方法。

使用如下函数调用，可以从 C 中调用任何 Java 方法：

```
(*env)->CallXxxMethod(env, implicit parameter, methodID, explicit parameters)
```

根据方法的返回类型，用 `Void`、`Int`、`Object` 等来替换 `Xxx`。就像需要一个 `fieldID` 来访问某个对象的一个域一样，还需要一个方法的 ID 来调用方法。可以通过调用 JNI 函数 `GetMethodID`，并且提供该类、方法的名字和方法签名来获得方法 ID。

在我们的例子中，我们想要获得 `PrintWriter` 类的 `print` 方法的 ID。`PrintWriter` 类有几个名为 `print` 的重载方法。基于这个原因，还必须提供一个字符串，描述想要使用的特定函数的

参数和返回值。例如，我们想要使用 void print(java.lang.String)，正如前一节讲到的那样，我们必须把签名“混编”为字符串 "(Ljava/lang/String;)V"。

下面是进行方法调用的完整代码：

```
/* get the class of the implicit parameter */
class_PrintWriter = (*env)->GetObjectClass(env, out);

/* get the method ID */
id_print = (*env)->GetMethodID(env, class_PrintWriter, "print", "(Ljava/lang/String;)V");

/* call the method */
(*env)->CallVoidMethod(env, out, id_print, str);
```

程序清单 12-14 和程序清单 12-15 给出了测试程序和 Printf3 类的 Java 代码。程序清单 12-16 包含了本地 fprintf 方法的 C 代码。

 **注释：**数值型的方法 ID 和域 ID 在概念上和反射 API 中的 Method 和 Field 对象相似。

可以使用以下函数在两者间进行转换：

```
 jobject ToReflectedMethod(JNIEnv* env, jclass class, jmethodID methodID);
    // returns Method object
 methodID FromReflectedMethod(JNIEnv* env, jobject method);
 jobject ToReflectedField(JNIEnv* env, jclass class, jfieldID fieldID);
    // returns Field object
 fieldID FromReflectedField(JNIEnv* env, jobject field);
```

#### 程序清单 12-14 printf3/Printf3Test.java

```
1 import java.io.*;
2
3 /**
4 * @version 1.11 2018-05-01
5 * @author Cay Horstmann
6 */
7 class Printf3Test
8 {
9     public static void main(String[] args)
10    {
11        double price = 44.95;
12        double tax = 7.75;
13        double amountDue = price * (1 + tax / 100);
14        var out = new PrintWriter(System.out);
15        Printf3.fprintf(out, "Amount due = %8.2f\n", amountDue);
16        out.flush();
17    }
18 }
```

#### 程序清单 12-15 printf3/Printf3.java

```
1 import java.io.*;
2
3 /**
4 * @version 1.10 1997-07-01
```

```

5  * @author Cay Horstmann
6  */
7 class Printf3
8 {
9     public static native void fprintf(PrintWriter out, String format, double x);
10
11    static
12    {
13        System.loadLibrary("Printf3");
14    }
15 }

```

### 程序清单 12-16 printf3/Printf3.c

```

1 /**
2  * @version 1.10 1997-07-01
3  * @author Cay Horstmann
4 */
5
6 #include "Printf3.h"
7 #include <string.h>
8 #include <stdlib.h>
9 #include <float.h>
10
11 /**
12  * @param format a string containing a printf format specifier
13  * (such as "%8.2f"). Substrings "%%" are skipped.
14  * @return a pointer to the format specifier (skipping the '%')
15  * or NULL if there wasn't a unique format specifier
16 */
17 char* find_format(const char format[])
18 {
19     char* p;
20     char* q;
21
22     p = strchr(format, '%');
23     while (p != NULL && *(p + 1) == '%') /* skip %% */
24         p = strchr(p + 2, '%');
25     if (p == NULL) return NULL;
26     /* now check that % is unique */
27     p++;
28     q = strchr(p, '%');
29     while (q != NULL && *(q + 1) == '%') /* skip %% */
30         q = strchr(q + 2, '%');
31     if (q != NULL) return NULL; /* % not unique */
32     q = p + strspn(p, " -0+#"); /* skip past flags */
33     q += strspn(q, "0123456789"); /* skip past field width */
34     if (*q == '.') { q++; q += strspn(q, "0123456789"); }
35     /* skip past precision */
36     if (strchr("eEfFgG", *q) == NULL) return NULL;
37     /* not a floating-point format */
38     return p;
39 }
40

```

```

41 JNIEXPORT void JNICALL Java_Printf3_fprint(JNIEnv* env, jclass cl,
42         jobject out, jstring format, jdouble x)
43 {
44     const char* cformat;
45     char* fmt;
46     jstring str;
47     jclass class_PrintWriter;
48     jmethodID id_print;
49
50     cformat = (*env)->GetStringUTFChars(env, format, NULL);
51     fmt = find_format(cformat);
52     if (fmt == NULL)
53         str = format;
54     else
55     {
56         char* cstr;
57         int width = atoi(fmt);
58         if (width == 0) width = DBL_DIG + 10;
59         cstr = (char*) malloc(strlen(cformat) + width);
60         sprintf(cstr, cformat, x);
61         str = (*env)->NewStringUTF(env, cstr);
62         free(cstr);
63     }
64     (*env)->ReleaseStringUTFChars(env, format, cformat);
65
66     /* now call ps.print(str) */
67
68     /* get the class */
69     jclass class_PrintWriter = (*env)->GetObjectClass(env, out);
70
71     /* get the method ID */
72     id_print = (*env)->GetMethodID(env, class_PrintWriter, "print", "(Ljava/lang/String;)V");
73
74     /* call the method */
75     (*env)->CallVoidMethod(env, out, id_print, str);
76 }

```

## 12.6.2 静态方法

从本地方法调用静态方法与调用非静态方法类似。两者的差别是：

- 要用 GetStaticMethodID 和 CallStaticXxxMethod 函数。
- 当调用方法时，要提供类对象，而不是隐式的参数对象。

作为一个例子，让我们从本地方法调用以下静态方法：

```
System.getProperty("java.class.path")
```

这个调用的返回值是给出了当前类路径的字符串。

首先，我们必须找到要用的类。因为我们没有 System 类的对象可供使用，所以我们使用 FindClass 而非 GetObjectClass：

```
jclass class_System = (*env)->FindClass(env, "java/lang/System");
```

接着，我们需要静态 getProperty 方法的 ID。该方法的编码签名是：

```
"(Ljava/lang/String;)Ljava/lang/String;"
```

既然参数和返回值都是字符串。因此，我们这样获取方法 ID：

```
jmethodID id_getProperty = (*env)->GetStaticMethodID(env, class_System, "getProperty",
"(Ljava/lang/String;)Ljava/lang/String;");
```

最后，我们进行调用。注意，类对象被传递给了 CallStaticObjectMethod 函数。

```
jobject obj_ret = (*env)->CallStaticObjectMethod(env, class_System, id_getProperty,
(*env)->NewStringUTF(env, "java.class.path"));
```

该方法的返回值是 jobject 类型的。如果我们想要把它当作字符串操作，必须把它转型为 jstring：

```
jstring str_ret = (jstring) obj_ret;
```

**C++ 注释：**在 C 中，jstring 和 jclass 类型同后面将要介绍的数组类型一样，都是与 jobject 等价的类型。因此，在 C 语言中，前面例子中的转型并不是严格必需的。但是在 C++ 中，这些类型被定义为指向拥有正确继承层次关系的“哑类”的指针。例如，将一个 jstring 不经过转型便赋给 jobject 在 C++ 中是合法的，但是将 jobject 赋给 jstring 必须先转型。

### 12.6.3 构造器

本地方法可以通过调用构造器来创建新的 Java 对象。可以调用 NewObject 函数来调用构造器。

```
jobject obj_new = (*env)->NewObject(env, class, methodID, construction parameters);
```

可以通过指定方法名为 "<init>"，并指定构造器（返回值为 void）的编码签名，从 GetMethodID 函数中获取该调用必需的方法 ID。例如，下面是本地方法创建 FileOutputStream 对象的情形：

```
const char[] fileName = "...";
jstring str_fileName = (*env)->NewStringUTF(env, fileName);
jclass class_FileOutputStream = (*env)->FindClass(env, "java/io/FileOutputStream");
jmethodID id_FileOutputStream
= (*env)->GetMethodID(env, class_FileOutputStream, "<init>", "(Ljava/lang/String;)V");
jobject obj_stream
= (*env)->NewObject(env, class_FileOutputStream, id_FileOutputStream, str_fileName);
```

注意，构造器的签名接受一个 java.lang.String 类型的参数，返回类型为 void。

### 12.6.4 另一种方法调用

有若干种 JNI 函数的变体都可以从本地代码调用 Java 方法。它们没有我们已经讨论过的那些函数那么重要，但偶尔也会很有用。

CallNonvirtualXxxMethod 函数接收一个隐式参数、一个方法 ID、一个类对象（必须对应于隐式参数的超类）和一个显式参数。这个函数将调用指定的类中的指定版本的方法，而不使

用常规的动态调度机制。

所有调用函数都有后缀 "A" 和 "V" 的版本，用于接收数组中或 `va_list` 中的显式参数（就像在 C 头文件 `stdarg.h` 中所定义的那样）。

### API 执行 Java 方法

- `jmethodID GetMethodID(JNIEnv *env, jclass cl, const char name[], const char methodSignature[])`  
返回类中某个方法的标识符。
- `Xxx CallXxxMethod(JNIEnv *env, jobject obj, jmethodID id, args)`
- `Xxx CallXxxMethodA(JNIEnv *env, jobject obj, jmethodID id, jvalue args[])`
- `Xxx CallXxxMethodV(JNIEnv *env, jobject obj, jmethodID id, va_list args)`

调用一个方法。返回类型 `Xxx` 是 `Object`、`Boolean`、`Byte`、`Char`、`Short`、`Int`、`Long`、`Float` 或 `Double` 之一。第一个函数有可变数量参数，只要把方法参数附加到方法 ID 之后即可。第二个函数接受 `jvalue` 数组中的方法参数，其中 `jvalue` 是一个联合体，定义如下：

```
typedef union jvalue
{
    jboolean z;
    jbyte b;
    jchar c;
    jshort s;
    jint i;
    jlong j;
    jfloat f;
    jdouble d;
    jobject l;
} jvalue;
```

第三个函数接收 C 头文件 `stdarg.h` 中定义的 `va_list` 中的方法参数。

- `Xxx CallNonvirtualXxxMethod(JNIEnv *env, jobject obj, jclass cl, jmethodID id, args)`
- `Xxx CallNonvirtualXxxMethodA(JNIEnv *env, jobject obj, jclass cl, jmethodID id, jvalue args[])`
- `Xxx CallNonvirtualXxxMethodV(JNIEnv *env, jobject obj, jclass cl, jmethodID id, va_list args)`

调用一个方法，并绕过动态调度。返回类型 `Xxx` 是 `Object`、`Boolean`、`Byte`、`Char`、`Short`、`Int`、`Long`、`Float` 或 `Double` 之一。第一个函数有可变数量参数，只要把方法参数附加到方法 ID 之后即可。第二个函数接受 `jvalue` 数组中的方法参数。第三个函数接受 C 头文件 `stdarg.h` 中定义的 `va_list` 中的方法参数。

- `jmethodID GetStaticMethodID(JNIEnv *env, jclass cl, const char name[], const char methodSignature[])`

返回类的某个静态方法的标识符。

- `Xxx CallStaticXxxMethod(JNIEnv *env, jclass cl, jmethodID id, args)`
- `Xxx CallStaticXxxMethodA(JNIEnv *env, jclass cl, jmethodID id, jvalue args[])`

- `Xxx CallStaticXxxMethodV(JNIEnv *env, jclass cl, jmethodID id, va_list args)`  
调用一个静态方法。返回类型 `Xxx` 是 `Object`、`Boolean`、`Byte`、`Char`、`Short`、`Int`、`Long`、`Float` 或 `Double` 之一。第一个函数有可变数量参数，只要把方法参数附加到方法 ID 之后即可。第二个函数接受 `jvalue` 数组中的方法参数。第三个函数接受 C 头文件 `stdarg.h` 中定义的 `va_list` 中的方法参数。
- `jobject NewObject(JNIEnv *env, jclass cl, jmethodID id, args)`
- `jobject NewObjectA(JNIEnv *env, jclass cl, jmethodID id, jvalue args[])`
- `jobject NewObjectV(JNIEnv *env, jclass cl, jmethodID id, va_list args)`  
调用构造器。函数 ID 从带有函数名为 "`<init>`" 和返回类型为 `void` 的 `GetMethodID` 获取。第一个函数有可变数量参数，只要把方法参数附加到方法 ID 之后即可。第二个函数接收 `jvalue` 数组中的方法参数。第三个函数接收 C 头文件 `stdarg.h` 中定义的 `va_list` 中的方法参数。

## 12.7 访问数组元素

Java 编程语言的所有数组类型都有相对应的 C 语言类型，见表 12-2。

表 12-2 Java 数组类型和 C 数组类型之间的对应关系

Java 数组类型	C 数组类型	Java 数组类型	C 数组类型
<code>boolean[]</code>	<code>jbooleanArray</code>	<code>long[]</code>	<code>jlongArray</code>
<code>byte[]</code>	<code>jbyteArray</code>	<code>float[]</code>	<code>jfloatArray</code>
<code>char[]</code>	<code>jcharArray</code>	<code>double[]</code>	<code>jdoubleArray</code>
<code>int[]</code>	<code>jintArray</code>	<code>Object[]</code>	<code>jobjectArray</code>
<code>short[]</code>	<code>jshortArray</code>		

**C++ 注释：**在 C 中，所有这些数组类型实际上都是 `jobject` 的同义类型。然而，在 C++ 中它们被安排在如图 12-3 所示的继承层次结构中。`jarray` 类型表示一个泛型数组。

`GetArrayLength` 函数返回数组的长度。

```
jarray array = . . .;
jsize length = (*env)->GetArrayLength(env, array);
```

怎样访问数组元素取决于数组中存储的是对象还是基本类型的数据（如 `bool`、`char` 或数值类型）。可以通过 `GetObjectArrayElement` 和 `SetObjectArrayElement` 方法访问对象数组的元素。

```
jobjectArray array = . . .;
int i, j;
jobject x = (*env)->GetObjectArrayElement(env, array, i);
(*env)->SetObjectArrayElement(env, array, j, x);
```

这个方法虽然简单，但是效率明显低下，当想要直接访问数组元素，特别是在进行向量

或矩阵计算时更是如此。

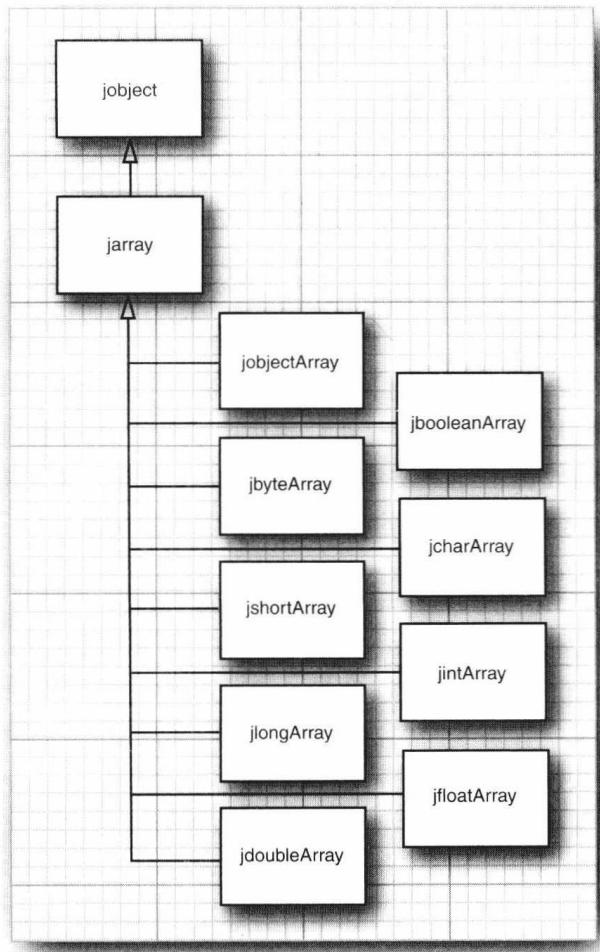


图 12-3 数组类型的继承层次结构

`GetXxxArrayElements` 函数返回一个指向数组起始元素的 C 指针。与普通的字符串一样，当不再需要该指针时，必须记得要调用 `ReleaseXxxArrayElements` 函数通知虚拟机。这里，类型 `Xxx` 必须是基本类型，也就是说，不能是 `Object`。这样就可以直接读写数组元素了。另一方面，由于指针可能会指向一个副本，只有调用相应的 `ReleaseXxx-ArrayElements` 函数时，所做的改变才能保证在源数组里得到反映。

**注释：**通过把一个指向 `jboolean` 变量的指针作为第三个参数传递给 `GetXxxArrayElements` 方法，就可以发现一个数组是否是副本了。如果是副本，则该变量被 `JNI_TRUE` 填充。如果对这个信息不感兴趣，传一个空指针即可。

下面是对 `double` 类型数组中的所有元素乘以一个常量的示例代码。我们获取一个 Java 数组的 C 指针 `a`，并用 `a[i]` 访问各个元素。

```
jdoubleArray array_a = . . .;
double scaleFactor = . . .;
double* a = (*env)->GetDoubleArrayElements(env, array_a, NULL);
for (i = 0; i < (*env)->GetArrayLength(env, array_a); i++)
    a[i] = a[i] * scaleFactor;
(*env)->ReleaseDoubleArrayElements(env, array_a, a, 0);
```

虚拟机是否确实需要对数组进行拷贝取决于它是如何分配数组和如何进行垃圾回收的。有些“拷贝”型的垃圾回收器例行地移动对象，并更新对象引用。该策略与将数组锁定在特定位置是不兼容的，因为回收器不能更新本地代码中的指针值。

**注释：**Oracle 的 JVM 实现中，boolean 数组是用打包的 32 位字数组表示的。GetBoolean-ArrayElements 方法能将它们复制到拆包的 jboolean 值的数组中。

如果要访问一个大数组的多个元素，可以用 GetXxxArrayRegion 和 SetXxxArray-Region 方法，它能把一定范围内的元素从 Java 数组复制到 C 数组中或从 C 数组复制到 Java 数组中。

可以用 NewXxxArray 函数在本地方法中创建新的 Java 数组。要创建新的对象数组，需要指定长度、数组元素的类型和所有元素的初始值（典型的是 NULL）。下面是一个例子。

```
jclass class_Employee = (*env)->FindClass(env, "Employee");
jobjectArray array_e = (*env)->NewObjectArray(env, 100, class_Employee, NULL);
```

基本类型的数组要简单一些。只需提供数组长度。

```
jdoubleArray array_d = (*env)->NewDoubleArray(env, 100);
```

该数组被 0 填充。

**注释：**下面的方法用来操作“直接缓存”：

```
jobject NewDirectByteBuffer(JNIEnv* env, void* address, jlong capacity)
void* GetDirectBufferAddress(JNIEnv* env, jobject buf)
jlong GetDirectBufferCapacity(JNIEnv* env, jobject buf)
```

java.nio 包中使用了直接缓存来支持更高效的输入输出操作，并尽可能减少本地和 Java 数组之间的复制操作。

### API 操作 Java 数组

- `jsize GetArrayLength(JNIEnv *env, jarray array)`  
返回数组中的元素个数。
- `jobject GetObjectArrayElement(JNIEnv *env, jobjectArray array, jsize index)`  
返回数组元素的值。
- `void SetObjectArrayElement(JNIEnv *env, jobjectArray array, jsize index, jobject value)`  
将数组元素设为新值。
- `Xxx* GetXxxArrayElements(JNIEnv *env, jarray array, jboolean* isCopy)`  
产生一个指向 Java 数组元素的 C 指针。域类型 Xxx 是 Boolean、Byte、Char、Short、Int、Long、Float 或 Double 之一。指针不再使用时，该指针必须传递给 `ReleaseXxxArrayElements`。

*iscopy* 可能是 NULL，或者在进行了复制时，指向用 `JNI_TRUE` 填充的 `jboolean`；否则，指向用 `JNI_FALSE` 填充的 `jboolean`。

- `void ReleaseXxxArrayElements(JNIEnv *env, jarray array, Xxx elems[], jint mode)`

通知虚拟机通过 `GetXxxArrayElements` 获得的一个指针已经不再需要了。Mode 是 0（更新数组元素后释放 `elems` 缓存）、`JNI_COMMIT`（更新数组元素后不释放 `elems` 缓存）或 `JNI_ABORT`（不更新数组元素便释放 `elems` 缓存）之一。

- `void GetXxxArrayRegion(JNIEnv *env, jarray array, jint start, jint length, Xxx elems[])`

将 Java 数组的元素复制到 C 数组中。域类型 `Xxx` 是 `Boolean`、`Byte`、`Char`、`Short`、`Int`、`Long`、`Float` 或 `Double` 之一。

- `void SetXxxArrayRegion(JNIEnv *env, jarray array, jint start, jint length, Xxx elems[])`

将 C 数组的元素复制到 Java 数组中。域类型 `Xxx` 是 `Boolean`、`Byte`、`Char`、`Short`、`Int`、`Long`、`Float` 或 `Double` 之一。

## 12.8 错误处理

在 Java 编程语言中，使用本地方法对于程序来说是要冒很大的安全风险的。C 的运行期系统对数组越界错误、不良指针造成的间接错误等不提供任何防护。所以，对于本地方法的程序员来说，处理所有的出错条件以保持 Java 平台的完整性显得格外重要。尤其是，当本地方法诊断出一个它无法解决的问题时，那么它应该将此问题报告给 Java 虚拟机。然后，在这种情况下，很自然地会抛出一个异常。然而，C 语言没有异常，必须调用 `Throw` 或 `ThrowNew` 函数来创建一个新的异常对象。当本地方法退出时，Java 虚拟机就会抛出该异常。

要使用 `Throw` 函数，需要调用 `NewObject` 来创建一个 `Throwable` 子类的对象。例如，下面我们分配了一个 `EOFException` 对象，然后将它抛出。

```
jclass class_EOFException = (*env)->FindClass(env, "java/io/EOFException");
jmethodID id_EOFException = (*env)->GetMethodID(env, class_EOFException, "<init>", "()V");
/* ID of no-argument constructor */
jthrowable obj_exc = (*env)->NewObject(env, class_EOFException, id_EOFException);
(*env)->Throw(env, obj_exc);
```

通常调用 `ThrowNew` 会更加方便，因为只需提供一个类和一个“modified UTF-8”字节序列，该函数就会构建一个异常对象。

```
(*env)->ThrowNew(env, (*env)->FindClass(env, "java/io/EOFException"),
"Unexpected end of file");
```

`Throw` 和 `ThrowNew` 都只是发布异常，它们不会中断本地方法的控制流。只有当该方法返回时，Java 虚拟机才会抛出异常。所以，每一个对 `Throw` 和 `ThrowNew` 的调用语句之后总是紧跟着 `return` 语句。

**C++** **C++ 注释：**如果用 C++ 实现本地方法，那么就无法用 C++ 代码抛出 Java 异常。在 C++ 绑定中，是可以实现一个在 C++ 异常和 Java 异常之间的转换的。然而，到目前

为止还没有实现这个功能。需要在本地方法中使用 Throw 或 ThrowNew 函数来抛出 Java 异常，并且要确保你的本地方法不抛出 C++ 异常。

通常，本地代码不需要考虑捕获 Java 异常。但是，当本地方法调用 Java 方法时，该方法可能会抛出异常。而且，一些 JNI 函数也会抛出异常。例如，如果索引越界，`SetObject-ArrayElement` 方法会抛出一个 `ArrayIndexOutOfBoundsException` 异常，如果所存储的对象的类不是数组元素类的子类，该方法会抛出一个 `ArrayStoreException` 异常。在这类情况下，本地方法应该调用 `ExceptionOccurred` 方法来确认是否有异常抛出。如果没有任何异常等待处理，则下面的调用：

```
jthrowable obj_exc = (*env)->ExceptionOccurred(env);
```

将返回 `NULL`。否则，返回一个当前异常对象的引用。如果只要检查是否有异常抛出，而不需要获得异常对象的引用，那么应使用：

```
jboolean occurred = (*env)->ExceptionCheck(env);
```

通常，有异常出现时，本地方法应该直接返回。那样，虚拟机就会将该异常传送给 Java 代码。但是，本地方法也可以分析异常对象，确定它是否能够处理该异常。如果能够处理，那么必须调用下面的函数来关闭该异常：

```
(*env)->ExceptionClear(env);
```

在我们的例子中，我们实现了 `fprint` 本地方法，这是基于该方法适合编写为本地方法的假设而实现的。下面是我们抛出的异常：

- 如果格式字符串是 `NULL`，则抛出 `NullPointerException` 异常。
- 如果格式字符串不含适合打印 `double` 所需的 `%` 说明符，则抛出 `IllegalArgumentException` 异常。
- 如果调用 `malloc` 失败，则抛出 `OutOfMemoryError` 异常。

最后，为了说明本地方法调用 Java 方法时，怎样检查异常，我们将一个字符串发送给数据流，一次一个字符，并且在每次调用 Java 方法后调用 `ExceptionOccurred`。程序清单 12-17 给出了本地方法的代码，程序清单 12-18 展示了含有本地方法的类的定义。注意，在调用 `PrintWriter.print` 出现异常时，本地方法并不会立即终止执行，它会首先释放 `cstr` 缓存。当本地方法返回时，虚拟机再次抛出异常。程序清单 12-19 的测试程序说明了当格式字符串无效时，本地方法是如何抛出异常的。

#### 程序清单 12-17 printf4/Printf4.c

```
1 /**
2  * @version 1.10 1997-07-01
3  * @author Cay Horstmann
4 */
5
6 #include "Printf4.h"
7 #include <string.h>
```

```

8 #include <stdlib.h>
9 #include <float.h>
10
11 /**
12  * @param format a string containing a printf format specifier
13  * (such as "%8.2f"). Substrings "%%" are skipped.
14  * @return a pointer to the format specifier (skipping the '%')
15  * or NULL if there wasn't a unique format specifier
16 */
17 char* find_format(const char format[])
18 {
19     char* p;
20     char* q;
21
22     p = strchr(format, '%');
23     while (p != NULL && *(p + 1) == '%') /* skip %% */
24         p = strchr(p + 2, '%');
25     if (p == NULL) return NULL;
26     /* now check that % is unique */
27     p++;
28     q = strchr(p, '%');
29     while (q != NULL && *(q + 1) == '%') /* skip %% */
30         q = strchr(q + 2, '%');
31     if (q != NULL) return NULL; /* % not unique */
32     q = p + strspn(p, " -0+#"); /* skip past flags */
33     q += strspn(q, "0123456789"); /* skip past field width */
34     if (*q == '.') { q++; q += strspn(q, "0123456789"); }
35     /* skip past precision */
36     if (strchr("eEfFgG", *q) == NULL) return NULL;
37     /* not a floating-point format */
38     return p;
39 }
40
41 JNIEXPORT void JNICALL Java_Printf4_fprint(JNIEnv* env, jclass cl,
42                                         jobject out, jstring format, jdouble x)
43 {
44     const char* cformat;
45     char* fmt;
46     jclass class_PrintWriter;
47     jmethodID id_print;
48     char* cstr;
49     int width;
50     int i;
51
52     if (format == NULL)
53     {
54         (*env)->ThrowNew(env,
55                             (*env)->FindClass(env,
56                                     "java/lang/NullPointerException"),
57                                     "Printf4.fprint: format is null");
58         return;
59     }
60
61     cformat = (*env)->GetStringUTFChars(env, format, NULL);

```

```

62     fmt = find_format(cformat);
63
64     if (fmt == NULL)
65     {
66         (*env)->ThrowNew(env,
67             (*env)->FindClass(env,
68                 "java/lang/IllegalArgumentException"),
69                 "Printf4.fprintf: format is invalid");
70     return;
71 }
72
73     width = atoi(fmt);
74     if (width == 0) width = DBL_DIG + 10;
75     cstr = (char*)malloc(strlen(cformat) + width);
76
77     if (cstr == NULL)
78     {
79         (*env)->ThrowNew(env,
80             (*env)->FindClass(env, "java/lang/OutOfMemoryError"),
81             "Printf4.fprintf: malloc failed");
82     return;
83 }
84
85     sprintf(cstr, cformat, x);
86
87     (*env)->ReleaseStringUTFChars(env, format, cformat);
88
89     /* now call ps.print(str) */
90
91     /* get the class */
92     class_PrintWriter = (*env)->GetObjectClass(env, out);
93
94     /* get the method ID */
95     id_print = (*env)->GetMethodID(env, class_PrintWriter, "print", "(C)V");
96
97     /* call the method */
98     for (i = 0; cstr[i] != 0 && !(*env)->ExceptionOccurred(env); i++)
99         (*env)->CallVoidMethod(env, out, id_print, cstr[i]);
100
101     free(cstr);
102 }

```

### 程序清单 12-18 printf4/Printf4.java

```

1 import java.io.*;
2
3 /**
4  * @version 1.10 1997-07-01
5  * @author Cay Horstmann
6  */
7 class Printf4
8 {
9     public static native void fprintf(PrintWriter ps, String format, double x);
10

```

```

11     static
12     {
13         System.loadLibrary("Printf4");
14     }
15 }
```

### 程序清单 12-19 printf4/Printf4Test.java

```

1 import java.io.*;
2
3 /**
4  * @version 1.11 2018-05-01
5  * @author Cay Horstmann
6  */
7 class Printf4Test
8 {
9     public static void main(String[] args)
10    {
11        double price = 44.95;
12        double tax = 7.75;
13        double amountDue = price * (1 + tax / 100);
14        var out = new PrintWriter(System.out);
15        /* This call will throw an exception--note the %% */
16        Printf4.fprintf(out, "Amount due = %%8.2f\n", amountDue);
17        out.flush();
18    }
19 }
```

### API 处理 Java 异常

- jint Throw(JNIEnv \*env, jthrowable obj)

准备一个在本地代码退出时抛出的异常。成功时返回 0，失败时返回一个负值。

- jint ThrowNew(JNIEnv \*env, jclass cl, const char msg[])

准备一个在本地代码退出时抛出的类型为 cl 的异常。成功时返回 0，失败时返回一个负值。msg 是表示异常对象的 String 构造参数的“modified UTF-8”字节序列。

- jthrowable ExceptionOccurred(JNIEnv \*env)

如果有异常挂起，则返回该异常对象，否则返回 NULL。

- jboolean ExceptionCheck(JNIEnv \*env)

如果有异常挂起，则返回 true。

- void ExceptionClear(JNIEnv \*env)

清除挂起的异常。

## 12.9 使用调用 API

到现在为止，我们主要讨论的都是进行了一些 C 调用的用 Java 编程语言编写的程序，

这大概是因为 C 的运行速度更快一些，或者允许访问一些 Java 平台无法访问的功能。假设在相反的情况下，你有一个 C 或者 C++ 的程序，并且想要调用一些 Java 代码。调用 API (invocation API) 使你能够把 Java 虚拟机嵌入到 C 或者 C++ 程序中。下面是初始化虚拟机所需的基本代码。

```
JavaVMOption options[1];
JavaVMInitArgs vm_args;
JavaVM *jvm;
JNIEnv *env;

options[0].optionString = "-Djava.class.path=.";
memset(&vm_args, 0, sizeof(vm_args));
vm_args.version = JNI_VERSION_1_2;
vm_args.nOptions = 1;
vm_args.options = options;

JNI_CreateJavaVM(&jvm, (void**) &env, &vm_args);
```

对 `JNI_CreateJavaVM` 的调用将创建虚拟机，并且使指针 `jvm` 指向虚拟机，使指针 `env` 指向执行环境。

可以给虚拟机提供任意数目的选项，这只需增加选项数组的大小和 `vm_args.nOptions` 的值。例如，

```
options[i].optionString = "-Djava.compiler=NONE";
```

可以钝化即时编译器。

 提示：当你陷入麻烦导致程序崩溃，从而不能初始化 JVM 或者不能装载你的类时，请打开 JNI 调试模式。设置一个选项如下：

```
options[i].optionString = "-verbose:jni";
```

你会看到一系列说明 JVM 初始化进程的消息。如果看不到你装载的类，请检查你的路径和类路径的设置。

一旦设置完虚拟机，就可以如前面小节介绍的那样调用 Java 方法了。只要按常规方法使用 `env` 指针即可。

只有在调用 API 中的其他函数时，才需要 `jvm` 指针。目前，只有四个这样的函数。最重要的一个是终止虚拟机的函数：

```
(*jvm)->DestroyJavaVM(jvm);
```

遗憾的是，在 Windows 下，动态链接到 `jre/bin/client/jvm.dll` 中的 `JNI_CreateJavaVM` 函数变得非常困难，因为 Vista 改变了链接规则，而 Oracle 的类库仍旧依赖于旧版本的 C 运行时类库。我们的示例程序通过手工加载该类库解决了这个问题，这种方式与 Java 程序所使用的方式一样，请参阅 JDK 中的 `src.jar` 文件里的 `launcher/java_md.c` 文件。

程序清单 12-20 的 C 程序设置了虚拟机，然后调用了 `Welcome` 类的 `main` 方法，这个类在卷 I 第 2 章中讨论过了（在开始启用测试程序之前，务必编译 `Welcome.java` 文件）。

## 程序清单 12-20 invocation/InvocationTest.c

```
1  /**
2   * @version 1.20 2007-10-26
3   * @author Cay Horstmann
4  */
5
6 #include <jni.h>
7 #include <stdlib.h>
8
9 #ifdef _WINDOWS
10
11 #include <windows.h>
12 static HINSTANCE loadJVMLibrary(void);
13 typedef jint (JNICALL *CreateJavaVM_t)(JavaVM **, void **, JavaVMInitArgs *);
14
15 #endif
16
17 int main()
18 {
19     JavaVMOption options[2];
20     JavaVMInitArgs vm_args;
21     JavaVM *jvm;
22     JNIEnv *env;
23     long status;
24
25     jclass class_Welcome;
26     jclass class_String;
27     jobjectArray args;
28     jmethodID id_main;
29
30 #ifdef _WINDOWS
31     HINSTANCE hjvmlib;
32     CreateJavaVM_t createJavaVM;
33 #endif
34
35     options[0].optionString = "-Djava.class.path=.";
36
37     memset(&vm_args, 0, sizeof(vm_args));
38     vm_args.version = JNI_VERSION_1_2;
39     vm_args.nOptions = 1;
40     vm_args.options = options;
41
42 #ifdef _WINDOWS
43     hjvmlib = loadJVMLibrary();
44     createJavaVM = (CreateJavaVM_t) GetProcAddress(hjvmlib, "JNI_CreateJavaVM");
45     status = (*createJavaVM)(&jvm, (void **) &env, &vm_args);
46 #else
47     status = JNI_CreateJavaVM(&jvm, (void **) &env, &vm_args);
48 #endif
49
50     if (status == JNI_ERR)
51     {
52         fprintf(stderr, "Error creating VM\n");
```

```

53     return 1;
54 }
55
56 class_Welcome = (*env)->FindClass(env, "Welcome");
57 id_main = (*env)->GetStaticMethodID(env, class_Welcome, "main", "([Ljava/lang/String;)V");
58
59 class_String = (*env)->FindClass(env, "java/lang/String");
60 args = (*env)->NewObjectArray(env, 0, class_String, NULL);
61 (*env)->CallStaticVoidMethod(env, class_Welcome, id_main, args);
62
63 (*jvm)->DestroyJavaVM(jvm);
64
65 return 0;
66 }
67
68 #ifdef _WINDOWS
69
70 static int GetStringFromRegistry(HKEY key, const char *name, char *buf, jint bufsize)
71 {
72     DWORD type, size;
73
74     return RegQueryValueEx(key, name, 0, &type, 0, &size) == 0
75         && type == REG_SZ
76         && size < (unsigned int) bufsize
77         && RegQueryValueEx(key, name, 0, 0, buf, &size) == 0;
78 }
79
80 static void GetPublicJREHome(char *buf, jint bufsize)
81 {
82     HKEY key, subkey;
83     char version[MAX_PATH];
84
85     /* Find the current version of the JRE */
86     char *JRE_KEY = "Software\\JavaSoft\\Java Runtime Environment";
87     if (RegOpenKeyEx(HKEY_LOCAL_MACHINE, JRE_KEY, 0, KEY_READ, &key) != 0)
88     {
89         fprintf(stderr, "Error opening registry key '%s'\n", JRE_KEY);
90         exit(1);
91     }
92
93     if (!GetStringFromRegistry(key, "CurrentVersion", version, sizeof(version)))
94     {
95         fprintf(stderr, "Failed reading value of registry key:\n\t%s\\CurrentVersion\n",
96                 JRE_KEY);
97         RegCloseKey(key);
98         exit(1);
99     }
100
101    /* Find directory where the current version is installed. */
102    if (RegOpenKeyEx(key, version, 0, KEY_READ, &subkey) != 0)
103    {
104        fprintf(stderr, "Error opening registry key '%s\\%s'\n", JRE_KEY, version);
105        RegCloseKey(key);
106        exit(1);

```

```

107 }
108
109 if (!GetStringFromRegistry(subkey, "JavaHome", buf, bufsize))
110 {
111     fprintf(stderr, "Failed reading value of registry key:\n\t%s\\%s\\JavaHome\n",
112             JRE_KEY, version);
113     RegCloseKey(key);
114     RegCloseKey(subkey);
115     exit(1);
116 }
117
118 RegCloseKey(key);
119 RegCloseKey(subkey);
120 }
121
122 static HINSTANCE loadJVMLibrary(void)
123 {
124     HINSTANCE h1, h2;
125     char msวดll[MAX_PATH];
126     char javadll[MAX_PATH];
127     GetPublicJREHome(msวดll, MAX_PATH);
128     strcpy(javadll, msวดll);
129     strncat(msวดll, "\\bin\\msvcr71.dll", MAX_PATH - strlen(msวดll));
130     msวดll[MAX_PATH - 1] = '\0';
131     strncat(javadll, "\\bin\\client\\jvm.dll", MAX_PATH - strlen(javadll));
132     javadll[MAX_PATH - 1] = '\0';
133
134     h1 = LoadLibrary(msวดll);
135     if (h1 == NULL)
136     {
137         fprintf(stderr, "Can't load library msvcr71.dll\n");
138         exit(1);
139     }
140
141     h2 = LoadLibrary(javadll);
142     if (h2 == NULL)
143     {
144         fprintf(stderr, "Can't load library jvm.dll\n");
145         exit(1);
146     }
147     return h2;
148 }
149
150 #endif

```

要在 Linux 下编译该程序，请用：

```
gcc -I jdk/include -I jdk/include/linux -o InvocationTest \
-L jdk/jre/lib/i386/client -ljvm InvocationTest.c
```

在 Windows 下用微软的 C 编译器时，请用下面的命令行：

```
cl -D_WINDOWS -I jdk/include -I jdk/include/win32 InvocationTest.c \
jdk\lib\jvm.lib advapi32.lib
```

需要确保 INCLUDE 和 LIB 环境变量包含了 Windows API 头文件和库文件的路径。

用 Cygwin 时，用下面的语句进行编译：

```
gcc -D_WINDOWS -mno-cygwin -I jdk\include -I jdk\include\win32 -D_int64="long long" \
-I c:\cygwin\usr\include\w32api -o InvocationTest
```

在 Linux/UNIX 下运行该程序之前，需要确保 LD\_LIBRARY\_PATH 包含了共享类库的目录。例如，如果使用 Linux 上的 bash 命令行，则需要执行下面的命令：

```
export LD_LIBRARY_PATH=jdk/jre/lib/i386/client:$LD_LIBRARY_PATH
```

### API 调用 API 函数

- `jint JNI_CreateJavaVM(JavaVM** p_jvm, void** p_env, JavaVMInitArgs* vm_args)`

初始化 Java 虚拟机。如果成功，则返回 0，否则返回 `JNI_ERR`。

- `jint DestroyJavaVM(JavaVM* jvm)`

销毁虚拟机。如果成功，则返回 0，否则返回一个负值。该函数必须通过一个虚拟机指针调用。例如，`(*jvm)->DestroyJavaVM(jvm)`。

## 12.10 完整的示例：访问 Windows 注册表

在本节中，我们介绍一个完整的可运行的例子，涵盖了我们在本章讨论的所有内容：使用带有字符串、数组和对象的本地方法，构造器调用和错误处理。我们将展示如何用 Java 平台包装器来包装普通的基于 C 的 API 子集，用于进行 Windows 注册表操作。当然，由于 Windows 的具体特性，使用 Windows 注册表的程序天生就不可移植。基于这个原因，标准的 Java 库不支持注册表，所以使用本地方法访问注册表是有意义的。

### 12.10.1 Windows 注册表概述

Windows 注册表是一个存放 Windows 操作系统和应用程序的配置信息的数据仓库。它提供了对系统和应用程序参数的单点管理和备份。其不足的方面是，注册表的错误也是单点的。如果你弄乱了注册表，你的电脑就会出故障，甚至无法启动。

我们不建议你使用注册表来存储 Java 程序的配置参数。Java 配置 API (preferences API) 是一个更好的解决方案（更多信息请参见卷 I 第 10 章）。我们使用注册表只是为了说明怎样把重要的本地 API 包装成 Java 类。

检查注册表的主要工具是注册表编辑器。由于可能存在幼稚而狂热的用户，所以 Windows 没有配备任何图标来启动注册表编辑器。你必须启动 DOS shell (或打开“开始”→“运行”对话框) 然后键入 `regedit`。图 12-4 给出了一个运行中的注册表编辑器。

左边是树形结构排列的注册表键。请注意，每个键都以 `HKEY` 节点开始，如：

```
HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
...
```

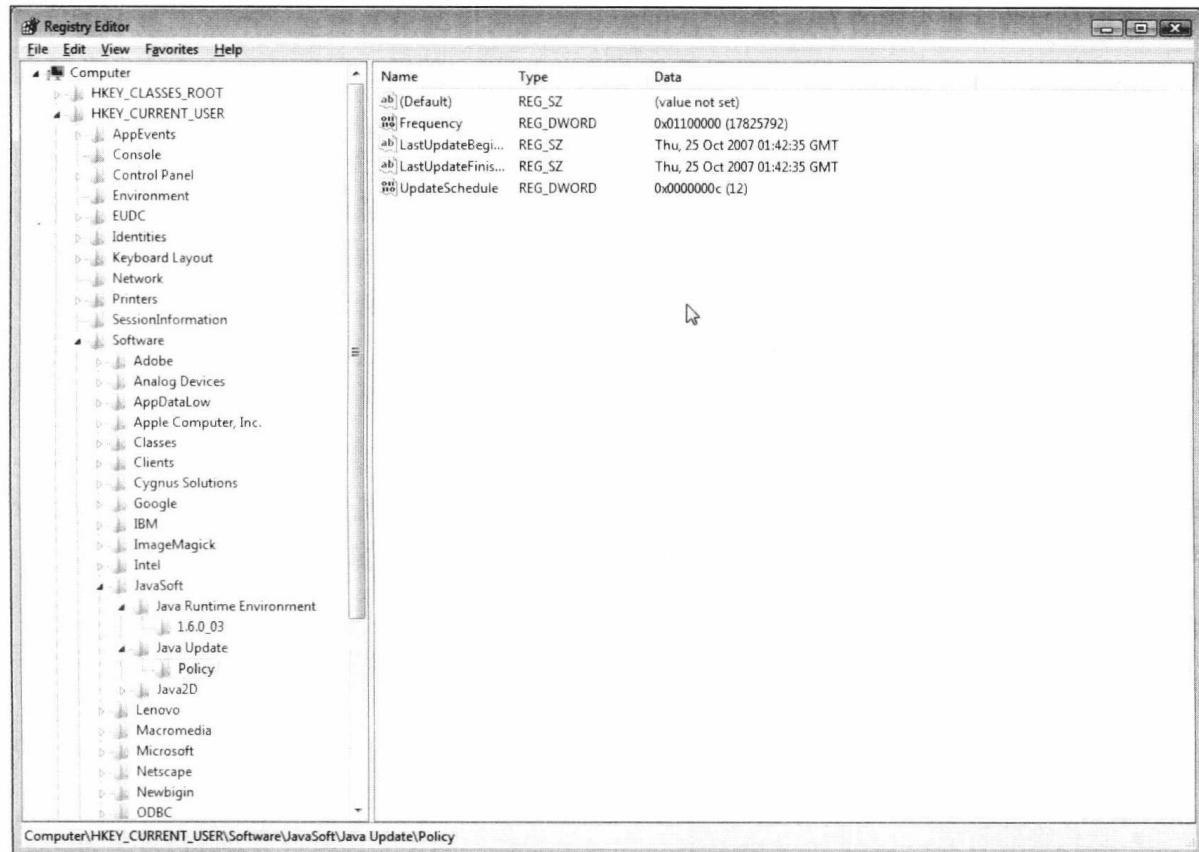


图 12-4 注册表编辑器

右边是与特定键关联的名 / 值对。例如，如果你安装了 Java 11，那么键：

HKEY\_LOCAL\_MACHINE\Software\JavaSoft\Java Runtime Environment

就包含下面这样的名值对：

```
CurrentVersion="11.0_10"
```

在本例中，值是字符串。值也可以是整数或字节数组。

### 12.10.2 访问注册表的 Java 平台接口

我们创建了一个从 Java 代码访问注册表的简单接口，然后用本地代码实现了这个接口。我们的接口只允许几个注册表操作，为了保持较小的代码规模，我们省略了其他重要的操作，如：添加、删除和枚举注册表键（添加剩余的这些注册表 API 函数是很容易的）。

即使使用我们提供的受限的子集，你也可以：

- 枚举某个键中存储的所有名字。
- 读出用某个名字存储的值。
- 设置用某个名字存储的值。

下面是封装注册表键的 Java 类：

```
public class Win32RegKey
{
    public Win32RegKey(int theRoot, String thePath) { . . . }
    public Enumeration names() { . . . }
    public native Object getValue(String name);
    public native void setValue(String name, Object value);

    public static final int HKEY_CLASSES_ROOT = 0x80000000;
    public static final int HKEY_CURRENT_USER = 0x80000001;
    public static final int HKEY_LOCAL_MACHINE = 0x80000002;
    .
}
```

`names` 方法返回与该键存放在一起的所有名字的一个枚举，你可以用你熟悉的 `hasMoreElements/nextElement` 方法获取它们。`getValue` 方法返回一个对象，该对象可以是字符串、`Integer` 对象或字节数组。`setValue` 方法的 `value` 参数也必须是上述三种类型之一。

### 12.10.3 以本地方法实现注册表访问函数

我们需要实现三个操作：

- 获取某个键的值。
- 设置某个键的值。
- 迭代键的名字。

在本章中，你基本上已经看到了所有必需的工具，如 Java 字符串和数组到 C 的字符串和数组的转换，还了解了如何在出错时抛出异常。

有两个问题使得这些本地方法比之前的例子更加复杂。`getValue` 和 `setValue` 方法处理的是 `Object` 类型，它可以是 `String`、`Integer` 或 `byte[]` 之一。枚举对象需要用来存放连续的对象。`hasMoreElements` 和 `nextElement` 的调用之间的状态。

让我们先看一下 `getValue` 方法，该方法（见程序清单 12-22）经历了以下几个步骤：

1. 打开注册表键。为了读取它们的值，注册表 API 要求这些键是开放的。
2. 查询与名字关联的值的类型和大小。
3. 把数据读到缓存。
4. 如果类型是 `REG_SZ`（字符串），调用 `NewStringUTF`，用该值来创建一个新的字符串。
5. 如果类型是 `REG_DWORD`（32 位整数），调用 `Integer` 构造器。
6. 如果类型是 `REG_BINARY`，调用 `NewByteArray` 来创建一个新的字节数组，并调用 `SetByteArrayRegion`，把值数据复制到该字节数组中。
7. 如果不是以上类型或调用 API 函数时出现错误，那就抛出异常，并小心地释放到此为止所获得的所有资源。
8. 关闭键，并返回创建的对象（`String`、`Integer` 或 `byte[]`）。

如你所见，这个例子很好地说明了怎样产生不同类型的 Java 对象。

在本地方法中，处理泛化的返回类型并不困难，`jstring`、`jobject` 或 `jarray` 引用都可以直接作为一个 `jobject` 返回。但是，`setValue` 方法接受的是一个对 `Object` 的引用，并且，为了把该 `Object` 保存为字符串、整数或字节数组，必须确定该 `Object` 的确切类型。我们可以通过查询 `value` 对象的类，找出对 `java.lang.String`、`java.lang.Integer` 和 `byte[]` 的引用，将其与 `IsAssignableFrom` 函数进行比较，从而确定它的确切类型。

如果 `class1` 和 `class2` 是两个类引用，那么调用：

```
(*env)->IsAssignableFrom(env, class1, class2)
```

当 `class1` 和 `class2` 是同一个类或 `class1` 是 `class2` 的子类时，返回 `JNI_TRUE`。在这两种情况下，`class1` 对象的引用都可以转型到 `class2`。例如，当：

```
(*env)->IsAssignableFrom(env, (*env)->GetObjectClass(env, value),
    (*env)->FindClass(env, "[B"))
```

为 `true` 时，那么我们就知道该值是一个字节数组。

下面是对 `setValue` 方法中的步骤的概述：

1. 打开注册表键以便写入。
2. 找出要写入的值的类型。
3. 如果类型是 `String`，调用 `GetStringUTFChars` 获取一个指向这些字符的指针。
4. 如果类型是 `Integer`，调用 `intValue` 方法获取该包装器对象中存储的整数。
5. 如果类型是 `byte[]`，调用 `GetByteArrayElements` 获取指向这些字节的指针。
6. 把数据和长度传递给注册表。
7. 关闭键。
8. 如果类型是 `String` 或 `byte[]`，那么还要释放指向数据的指针。

最后，我们介绍枚举键的本地方法。这些方法属于 `Win32RegKeyNameEnumeration` 类（参见程序清单 12-21）。当枚举过程开始时，我们必须打开键。在枚举过程中，我们必须保持该键的句柄。也就是说，该键的句柄必须与枚举对象存放在一起。键的句柄是 `DWORD` 类型的，它是一个 32 位数，所以可以存放在一个 Java 的整数中。它被存放在枚举类的 `hkey` 域中，当枚举开始时，`SetIntField` 初始化该域，而后续的调用用 `GetIntField` 来读取其值。

#### 程序清单 12-21 win32reg/Win32RegKey.java

```

1 import java.util.*;
2
3 /**
4  * A Win32RegKey object can be used to get and set values of a registry key in the Windows
5  * registry.
6  * @version 1.00 1997-07-01
7  * @author Cay Horstmann
8 */
9 public class Win32RegKey
10 {
11     public static final int HKEY_CLASSES_ROOT = 0x80000000;
12     public static final int HKEY_CURRENT_USER = 0x80000001;
```

```
13 public static final int HKEY_LOCAL_MACHINE = 0x80000002;
14 public static final int HKEY_USERS = 0x80000003;
15 public static final int HKEY_CURRENT_CONFIG = 0x80000005;
16 public static final int HKEY_DYN_DATA = 0x80000006;
17
18 private int root;
19 private String path;
20
21 /**
22 * Gets the value of a registry entry.
23 * @param name the entry name
24 * @return the associated value
25 */
26 public native Object getValue(String name);
27
28 /**
29 * Sets the value of a registry entry.
30 * @param name the entry name
31 * @param value the new value
32 */
33 public native void setValue(String name, Object value);
34
35 /**
36 * Construct a registry key object.
37 * @param theRoot one of HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE,
38 * HKEY_USERS, HKEY_CURRENT_CONFIG, HKEY_DYN_DATA
39 * @param thePath the registry key path
40 */
41 public Win32RegKey(int theRoot, String thePath)
42 {
43     root = theRoot;
44     path = thePath;
45 }
46
47 /**
48 * Enumerates all names of registry entries under the path that this object describes.
49 * @return an enumeration listing all entry names
50 */
51 public Enumeration<String> names()
52 {
53     return new Win32RegKeyNameEnumeration(root, path);
54 }
55
56 static
57 {
58     System.loadLibrary("Win32RegKey");
59 }
60 }
61
62 class Win32RegKeyNameEnumeration implements Enumeration<String>
63 {
64     public native String nextElement();
65     public native boolean hasMoreElements();
66     private int root;
```

```

67     private String path;
68     private int index = -1;
69     private int hkey = 0;
70     private int maxsize;
71     private int count;
72
73     Win32RegKeyNameEnumeration(int theRoot, String thePath)
74     {
75         root = theRoot;
76         path = thePath;
77     }
78 }
79
80 class Win32RegKeyException extends RuntimeException
81 {
82     public Win32RegKeyException()
83     {
84     }
85
86     public Win32RegKeyException(String why)
87     {
88         super(why);
89     }
90 }

```

在这个例子里，我们用枚举对象存放了另外三个数据项。当枚举一开始，我们可以从注册表中查询到名 / 值对的个数和最长名字的长度，我们需要这些信息，因此我们分配 C 字符数组以保存这些名字。这些值存放在枚举对象的 count 和 maxsize 域中。最后，index 域被初始化为 -1，表示枚举的开始。一旦其他实例域被初始化，index 域就被置为 0，在完成每个枚举步骤之后，都会进行递增。

让我们简要介绍一下支持枚举的本地方法。`hasMoreElements` 方法很简单：

1. 获取 `index` 和 `count` 域。
2. 如果 `index` 是 -1，调用 `startNameEnumeration` 函数打开键，查询数量和最大长度，初始化 `hkey`、`count`、`maxsize` 和 `index` 域。

3. 如果 `index` 小于 `count`，则返回 `JNI_TRUE`，否则返回 `JNI_FALSE`。

`nextElement` 方法要复杂一些。

1. 获取 `index` 和 `count` 域。
2. 如果 `index` 是 -1，调用 `startNameEnumeration` 函数打开键，查询数量和最大长度，初始化 `hkey`、`count`、`maxsize` 和 `index` 域。

3. 如果 `index` 等于 `count`，抛出一个 `NoSuchElementException` 异常。

4. 从注册表中读入下一个名字。

5. 递增 `index`。

6. 如果 `index` 等于 `count`，则关闭键。

在编译之前，记得在 `Win32RegKey` 和 `Win32RegKeyNameEnumeration` 上都要运行 `javac-h`。微软编

译器的完整命令行如下：

```
cl -I jdk\include -I jdk\include\win32 -LD Win32RegKey.c advapi32.lib -FeWin32RegKey.dll
```

Cygwin 系统上，请使用：

```
gcc -mno-cygwin -D __int64="long long" -I jdk\include -I jdk\include\win32 \
-I c:\cygwin\usr\include\w32api -shared -Wl,--add-stdcall-alias -o Win32RegKey.dll
Win32RegKey.c
```

因为注册表 API 是针对 Windows 的，所以这个程序不能在其他操作系统上运行。

程序清单 12-23 给出了测试我们新的注册表函数的程序。我们在键中添加了三个名值对：一个字符串、一个整数和一个字节数组。

```
HKEY_CURRENT_USER\Software\JavaSoft\Java Runtime Environment
```

然后，我们枚举该键的所有名字并获取它们的值。该程序应该打印如下信息：

```
Default user=Harry Hacker
Lucky number=13
Small primes=2 3 5 7 11 13
```

虽然在该键中添加这些名值对不会有什么危害，但是在运行该程序后，你可能还是想使用注册表编辑器去移除它们。

### 程序清单 12-22 win32reg/Win32RegKey.c

```
1  /**
2   * @version 1.00 1997-07-01
3   * @author Cay Horstmann
4  */
5
6 #include "Win32RegKey.h"
7 #include "Win32RegKeyNameEnumeration.h"
8 #include <string.h>
9 #include <stdlib.h>
10 #include <windows.h>
11
12 JNIEXPORT jobject JNICALL Java_Win32RegKey_getValue(
13     JNIEnv* env, jobject this_obj, jobject name)
14 {
15     const char* cname;
16     jstring path;
17     const char* cpath;
18     HKEY hkey;
19     DWORD type;
20     DWORD size;
21     jclass this_class;
22     jfieldID id_root;
23     jfieldID id_path;
24     HKEY root;
25     jobject ret;
26     char* cret;
27
28     /* get the class */
```

```

29     this_class = (*env)->GetObjectClass(env, this_obj);
30
31     /* get the field IDs */
32     id_root = (*env)->GetFieldID(env, this_class, "root", "I");
33     id_path = (*env)->GetFieldID(env, this_class, "path", "Ljava/lang/String;");
34
35     /* get the fields */
36     root = (HKEY) (*env)->GetIntField(env, this_obj, id_root);
37     path = (jstring)(*env)->GetObjectField(env, this_obj, id_path);
38     cpath = (*env)->GetStringUTFChars(env, path, NULL);
39
40     /* open the registry key */
41     if (RegOpenKeyEx(root, cpath, 0, KEY_READ, &hkey) != ERROR_SUCCESS)
42     {
43         (*env)->ThrowNew(env, (*env)->FindClass(env, "Win32RegKeyException"),
44                           "Open key failed");
45         (*env)->ReleaseStringUTFChars(env, path, cpath);
46         return NULL;
47     }
48
49     (*env)->ReleaseStringUTFChars(env, path, cpath);
50     cname = (*env)->GetStringUTFChars(env, name, NULL);
51
52     /* find the type and size of the value */
53     if (RegQueryValueEx(hkey, cname, NULL, &type, NULL, &size) != ERROR_SUCCESS)
54     {
55         (*env)->ThrowNew(env, (*env)->FindClass(env, "Win32RegKeyException"),
56                           "Query value key failed");
57         RegCloseKey(hkey);
58         (*env)->ReleaseStringUTFChars(env, name, cname);
59         return NULL;
60     }
61
62     /* get memory to hold the value */
63     cret = (char*)malloc(size);
64
65     /* read the value */
66     if (RegQueryValueEx(hkey, cname, NULL, &type, cret, &size) != ERROR_SUCCESS)
67     {
68         (*env)->ThrowNew(env, (*env)->FindClass(env, "Win32RegKeyException"),
69                           "Query value key failed");
70         free(cret);
71         RegCloseKey(hkey);
72         (*env)->ReleaseStringUTFChars(env, name, cname);
73         return NULL;
74     }
75
76     /* depending on the type, store the value in a string,
77      integer, or byte array */
78     if (type == REG_SZ)
79     {
80         ret = (*env)->NewStringUTF(env, cret);
81     }
82     else if (type == REG_DWORD)

```

```

83  {
84      jclass class_Integer = (*env)->FindClass(env, "java/lang/Integer");
85      /* get the method ID of the constructor */
86      jmethodID id_Integer = (*env)->GetMethodID(env, class_Integer, "<init>", "(I)V");
87      int value = *(int*) cret;
88      /* invoke the constructor */
89      ret = (*env)->NewObject(env, class_Integer, id_Integer, value);
90  }
91  else if (type == REG_BINARY)
92  {
93      ret = (*env)->NewByteArray(env, size);
94      (*env)->SetByteArrayRegion(env, (jarray) ret, 0, size, cret);
95  }
96  else
97  {
98      (*env)->ThrowNew(env, (*env)->FindClass(env, "Win32RegKeyException"),
99                          "Unsupported value type");
100     ret = NULL;
101 }
102 free(cret);
103 RegCloseKey(hkey);
104 (*env)->ReleaseStringUTFChars(env, name, cname);
105
106 return ret;
107 }
108 }

110 JNIEXPORT void JNICALL Java_Win32RegKey_setValue(JNIEnv* env, jobject this_obj,
111                                                 jstring name, jobject value)
112 {
113     const char* cname;
114     jstring path;
115     const char* cpath;
116     HKEY hkey;
117     DWORD type;
118     DWORD size;
119     jclass this_class;
120     jclass class_value;
121     jclass class_Integer;
122     jfieldID id_root;
123     jfieldID id_path;
124     HKEY root;
125     const char* cvalue;
126     int ivalue;
127
128     /* get the class */
129     this_class = (*env)->GetObjectClass(env, this_obj);
130
131     /* get the field IDs */
132     id_root = (*env)->GetFieldID(env, this_class, "root", "I");
133     id_path = (*env)->GetFieldID(env, this_class, "path", "Ljava/lang/String;");
134
135     /* get the fields */
136     root = (HKEY)(*env)->GetIntField(env, this_obj, id_root);

```

```

137 path = (jstring)(*env)->GetObjectField(env, this_obj, id_path);
138 cpath = (*env)->GetStringUTFChars(env, path, NULL);
139
140 /* open the registry key */
141 if (RegOpenKeyEx(root, cpath, 0, KEY_WRITE, &hkey) != ERROR_SUCCESS)
142 {
143     (*env)->ThrowNew(env, (*env)->FindClass(env, "Win32RegKeyException"),
144         "Open key failed");
145     (*env)->ReleaseStringUTFChars(env, path, cpath);
146     return;
147 }
148
149 (*env)->ReleaseStringUTFChars(env, path, cpath);
150 cname = (*env)->GetStringUTFChars(env, name, NULL);
151
152 class_value = (*env)->GetObjectClass(env, value);
153 class_Integer = (*env)->FindClass(env, "java/lang/Integer");
154 /* determine the type of the value object */
155 if ((*env)->IsAssignableFrom(env, class_value, (*env)->FindClass(env, "java/lang/String")))
156 {
157     /* it is a string--get a pointer to the characters */
158     cvalue = (*env)->GetStringUTFChars(env, (jstring) value, NULL);
159     type = REG_SZ;
160     size = (*env)->GetStringLength(env, (jstring) value) + 1;
161 }
162 else if ((*env)->IsAssignableFrom(env, class_value, class_Integer))
163 {
164     /* it is an integer--call intValue to get the value */
165     jmethodID id_intValue = (*env)->GetMethodID(env, class_Integer, "intValue", "()I");
166     ivalue = (*env)->CallIntMethod(env, value, id_intValue);
167     type = REG_DWORD;
168     cvalue = (char*)&ivalue;
169     size = 4;
170 }
171 else if ((*env)->IsAssignableFrom(env, class_value, (*env)->FindClass(env, "[B")))
172 {
173     /* it is a byte array--get a pointer to the bytes */
174     type = REG_BINARY;
175     cvalue = (char*)(*env)->GetByteArrayElements(env, (jarray) value, NULL);
176     size = (*env)->GetArrayLength(env, (jarray) value);
177 }
178 else
179 {
180     /* we don't know how to handle this type */
181     (*env)->ThrowNew(env, (*env)->FindClass(env, "Win32RegKeyException"),
182         "Unsupported value type");
183     RegCloseKey(hkey);
184     (*env)->ReleaseStringUTFChars(env, name, cname);
185     return;
186 }
187
188 /* set the value */
189 if (RegSetValueEx(hkey, cname, 0, type, cvalue, size) != ERROR_SUCCESS)
190 {

```

```

191     (*env)->ThrowNew(env, (*env)->FindClass(env, "Win32RegKeyException"),
192                         "Set value failed");
193 }
194
195 RegCloseKey(hkey);
196 (*env)->ReleaseStringUTFChars(env, name, cname);
197
198 /* if the value was a string or byte array, release the pointer */
199 if (type == REG_SZ)
200 {
201     (*env)->ReleaseStringUTFChars(env, (jstring) value, cvalue);
202 }
203 else if (type == REG_BINARY)
204 {
205     (*env)->ReleaseByteArrayElements(env, (jarray) value, (jbyte*) cvalue, 0);
206 }
207 }
208
209 /* helper function to start enumeration of names */
210 static int startNameEnumeration(JNIEnv* env, jobject this_obj, jclass this_class)
211 {
212     jfieldID id_index;
213     jfieldID id_count;
214     jfieldID id_root;
215     jfieldID id_path;
216     jfieldID id_hkey;
217     jfieldID id_maxsize;
218
219     HKEY root;
220     jstring path;
221     const char* cpath;
222     HKEY hkey;
223     DWORD maxsize = 0;
224     DWORD count = 0;
225
226     /* get the field IDs */
227     id_root = (*env)->GetFieldID(env, this_class, "root", "I");
228     id_path = (*env)->GetFieldID(env, this_class, "path", "Ljava/lang/String;");
229     id_hkey = (*env)->GetFieldID(env, this_class, "hkey", "I");
230     id_maxsize = (*env)->GetFieldID(env, this_class, "maxsize", "I");
231     id_index = (*env)->GetFieldID(env, this_class, "index", "I");
232     id_count = (*env)->GetFieldID(env, this_class, "count", "I");
233
234     /* get the field values */
235     root = (HKEY)(*env)->GetIntField(env, this_obj, id_root);
236     path = (jstring)(*env)->GetObjectField(env, this_obj, id_path);
237     cpath = (*env)->GetStringUTFChars(env, path, NULL);
238
239     /* open the registry key */
240     if (RegOpenKeyEx(root, cpath, 0, KEY_READ, &hkey) != ERROR_SUCCESS)
241     {
242         (*env)->ThrowNew(env, (*env)->FindClass(env, "Win32RegKeyException"),
243                           "Open key failed");
244         (*env)->ReleaseStringUTFChars(env, path, cpath);

```

```
245     return -1;
246 }
247 (*env)->ReleaseStringUTFChars(env, path, cpath);
248
249 /* query count and max length of names */
250 if (RegQueryInfoKey(hkey, NULL, NULL, NULL, NULL, NULL, NULL, &count, &maxsize,
251     NULL, NULL, NULL) != ERROR_SUCCESS)
252 {
253     (*env)->ThrowNew(env, (*env)->FindClass(env, "Win32RegKeyException"),
254         "Query info key failed");
255     RegCloseKey(hkey);
256     return -1;
257 }
258
259 /* set the field values */
260 (*env)->SetIntField(env, this_obj, id_hkey, (DWORD) hkey);
261 (*env)->SetIntField(env, this_obj, id_maxsize, maxsize + 1);
262 (*env)->SetIntField(env, this_obj, id_index, 0);
263 (*env)->SetIntField(env, this_obj, id_count, count);
264 return count;
265 }
266
267 JNIEXPORT jboolean JNICALL Java_Win32RegKeyNameEnumeration_hasMoreElements(JNIEnv* env,
268     jobject this_obj)
269 {
270     jclass this_class;
271     jfieldID id_index;
272     jfieldID id_count;
273     int index;
274     int count;
275     /* get the class */
276     this_class = (*env)->GetObjectClass(env, this_obj);
277
278     /* get the field IDs */
279     id_index = (*env)->GetFieldID(env, this_class, "index", "I");
280     id_count = (*env)->GetFieldID(env, this_class, "count", "I");
281
282     index = (*env)->GetIntField(env, this_obj, id_index);
283     if (index == -1) /* first time */
284     {
285         count = startNameEnumeration(env, this_obj, this_class);
286         index = 0;
287     }
288     else
289         count = (*env)->GetIntField(env, this_obj, id_count);
290     return index < count;
291 }
292
293 JNIEXPORT jobject JNICALL Java_Win32RegKeyNameEnumeration.nextElement(JNIEnv* env,
294     jobject this_obj)
295 {
296     jclass this_class;
297     jfieldID id_index;
298     jfieldID id_hkey;
```

```

299 jfieldID id_count;
300 jfieldID id_maxsize;
301
302 HKEY hkey;
303 int index;
304 int count;
305 DWORD maxsize;
306
307 char* cret;
308 jstring ret;
309
310 /* get the class */
311 this_class = (*env)->GetObjectClass(env, this_obj);
312
313 /* get the field IDs */
314 id_index = (*env)->GetFieldID(env, this_class, "index", "I");
315 id_count = (*env)->GetFieldID(env, this_class, "count", "I");
316 id_hkey = (*env)->GetFieldID(env, this_class, "hkey", "I");
317 id_maxsize = (*env)->GetFieldID(env, this_class, "maxsize", "I");
318
319 index = (*env)->GetIntField(env, this_obj, id_index);
320 if (index == -1) /* first time */
321 {
322     count = startNameEnumeration(env, this_obj, this_class);
323     index = 0;
324 }
325 else
326     count = (*env)->GetIntField(env, this_obj, id_count);
327
328 if (index >= count) /* already at end */
329 {
330     (*env)->ThrowNew(env, (*env)->FindClass(env, "java/util/NoSuchElementException"),
331                     "past end of enumeration");
332     return NULL;
333 }
334
335 maxsize = (*env)->GetIntField(env, this_obj, id_maxsize);
336 hkey = (HKEY)(*env)->GetIntField(env, this_obj, id_hkey);
337 cret = (char*)malloc(maxsize);
338
339 /* find the next name */
340 if (RegEnumValue(hkey, index, cret, &maxsize, NULL, NULL, NULL, NULL) != ERROR_SUCCESS)
341 {
342     (*env)->ThrowNew(env, (*env)->FindClass(env, "Win32RegKeyException"),
343                     "Enum value failed");
344     free(cret);
345     RegCloseKey(hkey);
346     (*env)->SetIntField(env, this_obj, id_index, count);
347     return NULL;
348 }
349
350 ret = (*env)->NewStringUTF(env, cret);
351 free(cret);
352

```

```

353     /* increment index */
354     index++;
355     (*env)->SetIntField(env, this_obj, id_index, index);
356
357     if (index == count) /* at end */
358     {
359         RegCloseKey(hkey);
360     }
361
362     return ret;
363 }
```

### 程序清单 12-23 win32reg/Win32RegKeyTest.java

```

1 import java.util.*;
2
3 /**
4  * @version 1.03 2018-05-01
5  * @author Cay Horstmann
6 */
7 public class Win32RegKeyTest
8 {
9     public static void main(String[] args)
10    {
11        var key = new Win32RegKey(
12            Win32RegKey.HKEY_CURRENT_USER, "Software\\JavaSoft\\Java Runtime Environment");
13
14        key.setValue("Default user", "Harry Hacker");
15        key.setValue("Lucky number", new Integer(13));
16        key.setValue("Small primes", new byte[] { 2, 3, 5, 7, 11 });
17
18        Enumeration<String> e = key.names();
19
20        while (e.hasMoreElements())
21        {
22            String name = e.nextElement();
23            System.out.print(name + "=");
24
25            Object value = key.getValue(name);
26
27            if (value instanceof byte[])
28                for (byte b : (byte[]) value) System.out.print((b & 0xFF) + " ");
29            else
30                System.out.print(value);
31
32            System.out.println();
33        }
34    }
35 }
```

### API 类型质询函数

- `jboolean IsAssignableFrom(JNIEnv *env, jclass cl1, jclass cl2)`

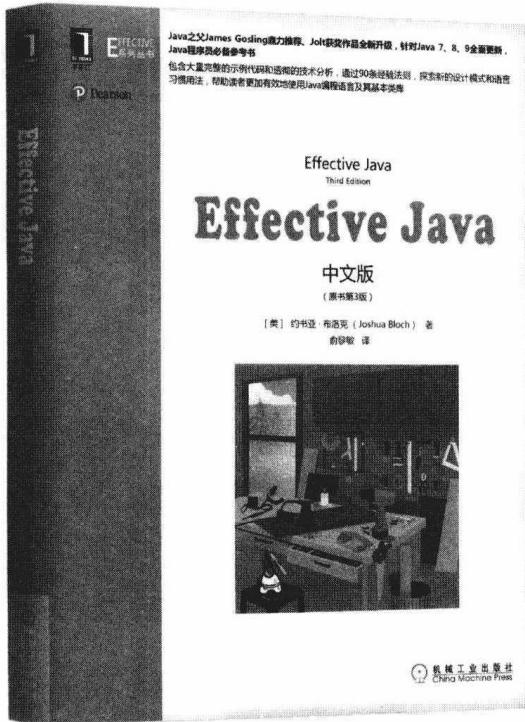
如果第一个类的对象可以赋给第二个类的对象，则返回 `JNI_TRUE`，否则返回 `JNI_FALSE`。这个函数可以测试：两个类是否相同，`cl1` 是否是 `cl2` 的子类，`cl2` 是否表示一个由 `cl1` 或它的一个超类实现的接口。

- `jclass GetSuperclass(JNIEnv *env, jclass cl)`

返回某个类的超类。如果 `cl` 表示 `Object` 类或一个接口，则返回 `NULL`。

一路走来，大家已经学习了许多高级 API，现在，终于结束了。我们从每位 Java 程序员都应该了解的主题开始，即：流、XML、网络、数据库和国际化，又用了非常技术性的几章结尾，即安全、注解处理、高级图形化编程和本地方法。我们希望你能够真正享受这个旅程，掌握这些涉及领域广泛的 Java API，并能够将这些新知识应用到你的项目中。

# 推荐阅读



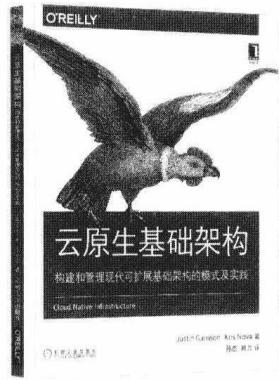
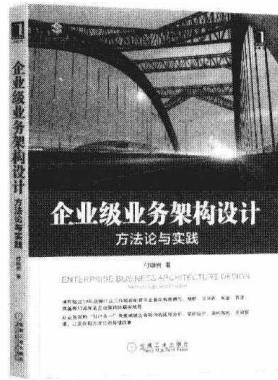
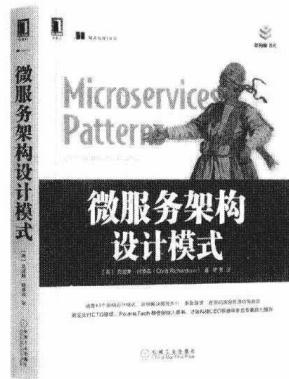
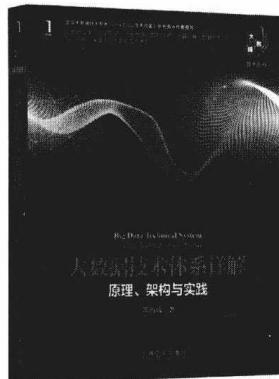
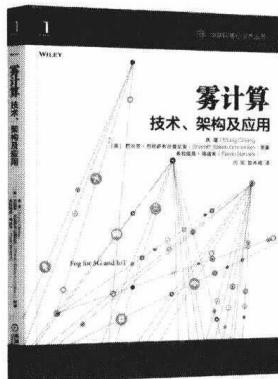
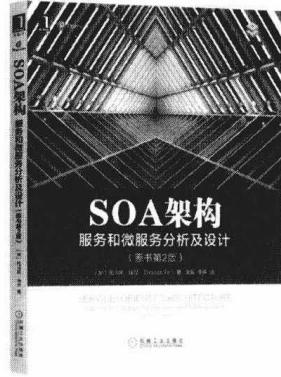
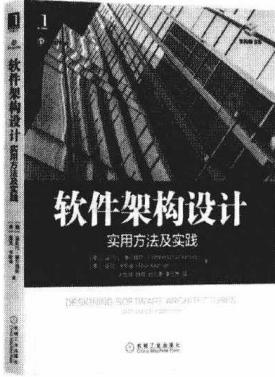
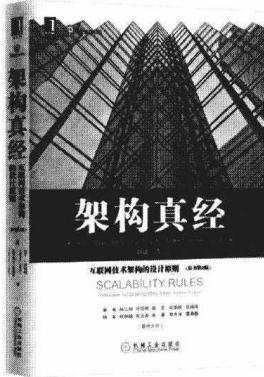
## Effective Java中文版（原书第3版）

作者：[美] 约书亚·布洛克（Joshua Bloch） ISBN：978-7-111-61272-8 定价：119.00元

Java之父James Gosling鼎力推荐、Jolt获奖作品全新升级，针对Java 7、8、9全面更新，Java程序员必备参考书

包含大量完整的示例代码和透彻的技术分析，通过90条经验法则，探索新的设计模式和语言习惯用法，帮助读者更加有效地使用Java编程语言及其基本类库

# 推荐阅读



# Java 核心技术 卷II

高级特性 (原书第11版)

对程序员来说，如果希望为实际应用编写健壮的代码，那么《Java核心技术》绝对是一本业内领先的、言简意赅的经典教程和参考文献。如今，《Java核心技术 卷II 高级特性（原书第11版）》针对Java 11进行修订，全面讨论了高级UI特性、企业编程、网络、安全和Java强大的模块系统等内容。

凯·S·霍斯特曼对Java复杂的新特性进行了深入而全面的研究，展示了如何使用它们来构建具有专业品质的应用程序。霍斯特曼所设计的经过全面完整测试的示例反映了当今的Java风格和最佳实践，这些示例设计精心、易于理解且实践价值极高，读者可以以这些示例为基础来编写自己的代码。

## 通过阅读本书，你将：

- 掌握用于编写优质Java代码所需的高级技术、惯用法和最佳实践
- 充分利用新的Java I/O API、对象序列化和正则表达式
- 高效地连接到网络服务、实现客户端和服务器程序以及获取Web数据
- 通过使用脚本API、编译器API和注解处理机制来处理代码
- 加深对Java平台模块系统的理解，并将代码迁移到其上
- 利用对编写应用程序的程序员来说颇有价值的新的Java安全特性
- 对高级的客户端用户界面编程，并在服务器端生成图像

《Java核心技术 卷I 基础知识（原书第11版）》中有对Java语言和UI编程的基础知识的专业级详解，包括对象、泛型、集合、lambda表达式、Swing设计、并发和函数式编程等。



上架指导：计算机\程序设计\Java

ISBN 978-7-111-64343-2

9 787111 643432 >

定价：149.00元

Pearson  
www.pearson.com

投稿热线：(010) 88379604  
读者信箱：hzit@hzbook.com  
客服电话：(010) 88361066 88379833 68326294

华章网站：[www.hzbook.com](http://www.hzbook.com)  
网上购书：[www.china-pub.com](http://www.china-pub.com)  
数字阅读：[www.hzmedia.com.cn](http://www.hzmedia.com.cn)