Java 高级编程：打印

Java Pro Programming: Printing

学习如何使用打印服务 API

Learn how to use the Print Service API

作者：Brett Spell
翻译：ice_x
修订：Cedar

版权声明：

可以任意转载，转载时请务必以超链接形式标明文章原始出处和作者信息及本声明

英文原文: http://www.javaworld.com/javaworld/jw-07-2005/jw-0725-print.html
中文原始翻译: http://www.matrix.org.cn/resource/article/43/43658_java_print.html
中文修订版本: http://cedar.peng.googlepages.com/Java_Print_API_ProStudy.pdf

关键词: Java Programming print PrintJob

摘要

在这篇文章里，来自 Pro Java Programming (Apress, June 2005)专家 Brett Spell 解释了如何一步一步的定位打印设备，创建打印工程，创建一个 **Doc** 接口的实例，以此来描述你想要打印的数据并且初始化打印。**（4500 字，2005 年 7 月 25 日 ice_x 原稿；6900 字，2006 年 5 月 19 日 Cedar 修订）**

自从问世以来，**Java** 在多数方面都成长迅速。但一直以来，打印输出是 **Java** 最弱的方面之一。事实上，**Java1.0** 根本不支持任何打印功能。**Java1.1** 在 **java.awt** 包里包含了一个叫做 **PrintJob** 的类，但是这个类提供的打印功能十分粗糙和不可靠。当 **Java1.2** (或者说 "***Java2***")出现后，依据 **PrinterJob** 以及在新的 **java.awt.print** 包里定义的类和接口，它提供了一个彻底分离的打印设计机制（称为 ***Java 2D printing API***）。这些改进使得基于 **PrintJob** 的机制（即 ***AWT printing***）基本上被淘汰了，但是 **PrintJob** 从未被真正废弃，而且至少在本文里仍然是一个技术上支持的类。

在 **J2SE 1.3** 里又增加了一些变化，利用在 **java.awt** 包里适合的 **JobAttributes** 和 **PageAttributes** 两个类，**PrintJob** 的功能扩展到设定任务和页面的属性。随着 J2SE 1.3 的发布，打印能力变得足够强壮，但是在关联两个完全分离的打印机制时，仍然存在一些问题。比如，两种机制都使用 **java.awt.Graphics** 类的一个实例，来展现打印内容，意味着所有要打印的东西都必须用一张图片表示。此外，更新后的 **PrintJob** 工具变得更强壮，但仅提供了很有限的任务相关的属性设置。最终，两种机制都没有提供一种选择目标打印机的可编程的途径。

Java 打印最大的改变发生于 **J2SE 1.4** 的发布，其带来的 **Java 打印服务 API**。第三代 **Java** 打印支持的诞生，借助于 **javax.print** 包的 **PrintService** 和 **DocPrintJob** 接口的实现突破了上述限制。因为新的 **API** 代表了两种旧的打印机制功能定义的父集，是你应该通常使用的并是本文的焦点。

从较高层次上来看，使用 **Java** 打印服务 **API** 的步骤是很简单的：

1. 定位打印服务（打印机），可以限制返回的列表，只要那些符合您应用程序需要的打印机。打印服务由 **PrintService** 的实例体现。

2. 通过调用 **PrintService** 接口中定义的 **createPrintJob()** 方法创建一个打印任务。打印任务由 **DocPrintJob** 的一个实例代表。

3. 创建一个 **Doc** 接口的实现，来描述你想要打印的数据。你也可以创建一个 **PrintRequestAttributeSet** 的实例，来定义你想要的打印选项。

4. 通过 **DocPrintJob** 接口定义的 **print()** 方法来初始化打印，指定你先前创建的 **Doc**，指定 **PrintRequestAttributeSet** 或者空值。

现在你可以检查每一步，并试着完成它们。

注意
在这篇文章里，我将交替使用**打印机**和**打印服务**，因为在大部分情况下，打印服务不亚于一台物理的打印机。 更一般意义上的打印服务反映了理论上可以发送到打印机以外的输出。举例来说，打印服务可能根本不打印东西，而是写入磁盘上的文件。换句话说，所有的打印机要表示为打印服务，但是并不是所有打印服务必须和一台物理的打印机关联。尽管如此，实际上你通常会把你的内容到打印机，这就是我为什么有时候使用更为简便的**打印机**这个词，来代替技术上更精确的**打印服务**。

1．定义打印服务

Locating print services

你可以使用在 **PrintServiceLookup** 类中定义的三种静态方法中的一种来定义。最简单的一种就是 **lookupDefaultPrintService()**，正如它的名字一样，它返回一个服务指向您默认的打印机：

```
PrintService service = PrintServiceLookup.lookupDefaultPrintService();
```

虽然用这个办法简单而方便，用它来选择输出所需的打印机，意味着你默认了用户缺省的打印机的功能，总是满足正确输出您所需的程序数据。实际上，你通常想要选择的是那种可以处理您的数据类型，并可以符合您的应用所需特性，例如彩色或者两面打印。为了从列表中返回所有已定义的打印机序列，或满足您需要功能的打印机序列，您可以使用余下两个在 **PrintServiceLookup** 中定义的静态方法，即 **lookupPrintServices()**或 **lookupMultiDocPrintServices()**。

lookupPrintServices()方法接受两个参数：一个 **DocFlavor** 的实例和一个实现 **AttributeSet** 接口的对象。你马上将看到，你可以使用两者中任意一个或同时来限制返回的打印机列表，但是 **lookupPrintServices()**允许你指定这两个参数中的任意一个或同时空值。如果把两者都设为空，那么你实际要求得到的返回值将是所有可用的打印机列表。截止目前为止，你还没有真正地查看过 **PrintService** 中定义的方法，其中一个 **getName()** 方法返回了一个代表打印机的名字的字符串。你可以通过编译和执行下面的代码，来列出你的系统可用的打印机：

```
PrintService[ ] services = PrintServiceLookup.lookupPrintServices(null, null);

for (int i = 0; i < services.length; i++) {

   System.out.println(services[i].getName());

}
```

例如，你能访问到连接在名为 **PrintServer** 服务器上的 **Alpha，Beta** 和 **Gamma** 打印机，用以上代码可以得到以下输出：

\\PrintServer\Alpha

\\PrintServer\Beta

\\PrintServer\Gamma

现在让我们来查看那些你可以传给 **lookupPrintServices()**方法的参数，并观察如何返回拥有特殊功能的打印机。

2．DocFlavor

在调用 **lookupPrintService()**方法时，第一个你可以指定的参数是一个 **DocFlavor** 类的实例，它描述了将要打印的数据的类型和如何存储。在大部分情况下，并不需要您去创建一个新的实例因为 Java 包含了很多预先定义的实例，你只要简单地传递其中一个实例的引用给 **lookupPrintServices()**。尽管（事情通常都是这样地简单），我们还是来看一下 **DocFlavor** 的构造和方法，来理解打印服务如何使用这个实例。

创建 **DocFlavor** 实例需要的两个参数都是字符串，一个是 **MIME (Multipurpose Internet Mail Extensions)**类型，另一个是表现类的名字。**MIME** 类型被 **DocFlavor** 用于描述数据类型。例如，你要打印一个 **gif** 文件，你需要使用 **MIME** 类型是 **image/gif** 的 **DocFlavor**。类似地，如果打印文本，你可能要用 **text/plain**，或者是打印 **HTML** 文档，你则要用 **text/html**。

3．表现类

Representation class

MIME 类型描述将要打印数据的类型，表现类则表示这些数据如何处理并交付打印服务。**DocFlavor** 包含了七个静态的内部类，每一个对应一个表现类及不同的封装方法。

表 1 中列出了 **DocFlavor** 中定义的静态内部类的名称，及想对应的表现类。注意除了 **SERVICE_FORMATTED**（一会我会更详细地解释），每一个类都说明了和"**binary**"或"**character**"相对应。事实上，这些差别是人为的，因为"**character**"数据类型本身就是一种特殊的 **binary** 类型。这种情况下，我们说的二进制（**binary**）数据包括人们可以看懂的字符以及一些格式化的字符比如 **tabs**，回车，等等。当然，这些差别很重要，反映出面向字符的表现类并不适合存储二进制打印数据。

例如，你不会将一个表现为 **gif** 图片的东西存储到字符数组或者 **String** 对象中，同时也不会通过实现一个 **Reader** 接口来访问它。另一方面，因为字符数据也是一种特殊的二进制数据，它完全适合储存文本信息到字节数组里或者通过 **InputStream** 或者一个 **URL** 来访问它。

Table 1. DocFlavor 的表现类

Inner class name	Representation class	Data type
BYTE_ARRAY	[B (byte[])	Binary
CHAR_ARRAY	[C (char[])	Character
INPUT_STREAM	java.io.InputStream	Binary
READER	java.io.Reader	Character
SERVICE_FORMATTED	java.awt.print.Pageable or java.awt.print.Printable	Other
STRING	java.lang.String	Character
URL	java.net.URL	Binary

在 **DocFlavor** 中定义的每一个静态内部类对应一个表现类，但是请记住我说过，每一个 **DocFlavor** 的实例封装了一个表现类和一个 **MIME** 来确认要打印的数据的类型。要访问这样一个 **DocFlavor** 实例，其不仅与表现类，并且与你想要打印的内容的 **MIME** 类型相关，你需要参考表 1 列出的一个内部类。例如，我们假设你要打印一个能在网上通过 **URL** 访问的 **gif** 文件。这里，显然的表现类选择是 **java.net.URL**，在 **DocFlavor** 中对应的静态类就是 **URL** 类。如果你查看那个内部类的文档，你会发现其实它定义了一系列静态的内部类，每一个对应一种打印机普遍支持的 **MIME** 类型。表 2 描述了在 **DocFlavor.URL** 里的内部类极其对应的 **MIME**。

Static inner classes	MIME type
AUTOSENSE	application/octet-stream
GIF	image/gif
JPEG	image/jpeg
PCL	PCL application/vnd-hp.PCL
PDF	application/pdf
PNG	image/png
POSTSCRIPT	application/postscript
TEXT_HTML_HOST	text/html
TEXT_HTML_US_ASCII	text/html;charset=us-ascii
TEXT_HTML_UTF_16	text/html;charset=utf-16
TEXT_HTML_UTF_16BE	text/html;charset=utf-16be
TEXT_HTML_UTF_16LE	text/html;charset=utf-16le
TEXT_HTML_UTF_8	TEXT_HTML_UTF_8 text/html;charset=utf-8
TEXT_PLAIN_HOST	text/plain
TEXT_PLAIN_US_ASCII	text/plain;charset=us-ascii
TEXT_PLAIN_UTF_16	text/plain;charset=utf-16
TEXT_PLAIN_UTF_16BE	text/plain;charset=utf-16be
TEXT_PLAIN_UTF_16LE	text/plain;charset=utf-16le
TEXT_PLAIN_UTF_8	text/plain;charset=utf-8

因为要通过 **URL** 打印 **gif** 图片，你可以用以下代码来访问获得一个的 **DocFlavor** 实例

```
DocFlavor flavor = DocFlavor.URL.GIF;
```

该代码创建了一个 **DocFlavor** 静态实例的引用，其代表类是 **java.net.URL**，**MIME** 是 **image/gif**。

表 2 列出的类在 **DocFlavor.URL** 的类中定义，那么其他六个在 **DocFlavor** 内定义的内部类呢？我们依然会等一下再来讨论 **SERVICE_FORMATTED**，这之前，看看与二进制数据相关的所有三种类型（**BYTE_ARRAY, INPUT_STREAM, 和 URL**）相关的内部类，它们的名字和表 2 中列出的一样。例如，如果你把 **gif** 数据储存到了一个字节数组里，那么你可以用以下代码：

```
DocFlavor flavor = DocFlavor.BYTE_ARRAY.GIF;
```

正如有三个与二进制类型关联的 **DocFlavor** 有它们自己的内部类一样，与字符类型相关的另外三个类，也包含另一种类型的内部类，表 3 中列出。

Table 3. CHAR_ARRAY, READER, and STRING

Static inner class	MIME type
TEXT_HTML	text/html;charset=utf-16
TEXT_PLAIN	text/plain;charset=utf-16

所以，如果你想打印储存在字符串中的文本数据，可用以下代码：

```
DocFlavor flavor = DocFlavor.STRING.TEXT_PLAIN;
```

类似，如果文本来自于网页上的 HTML 文档，并且你希望打印出和在浏览器中看到的一样的效果，就用以下代码：

```
DocFlavor flavor = DocFlavor.STRING.TEXT_HTML;
```

4．选择正确的打印机

Choosing the right printer

还记得我们在开始讨论 **DocFlavor** 之时，关于确认您实际使用的打印机，支持需要打印的数据类型，以及你期望使用的传送机制（即表现类）。这步看起来似乎没有必要，但实际上，你会对给定打印机所支持的文档类型感到吃惊。例如，刚提到文本类型看起来似乎是最容易支持的，所以，如果你的程序要打印一个普通文本或者 **HTML** 文本，你可能会简单地选择第一个有效的打印服务，并将输出送到那台打印机去。然而大部分打印机不支持基于文本的表现类，如果你试图向打印机发送你选择的 **DocFlavor**，但是它却不支持，就会抛出下面的异常：

```
Exception in thread "main" sun.print.PrintJobFlavorException: invalid flavor
        at sun.print.Win32PrintJob.print(Win32PrintJob.java:290)
        at PrintTest.main(PrintTest.java:11)
```

现在你已经知道了如何得到一个 **DocFlavor** 的引用，并且我们也讨论了选择支持这个 **DocFlavor** 的打印机重要性，接下来我来将告诉你，如何确定你使用的打印机支持所需特性。我先前说过 **lookupPrintServices()**允许你指定一个 **DocFlavor** 作为第一个参数，如果你指定的参数非空，那么方法会返回支持指定 **DocFlavor** 的打印服务实例。例如，以下代码将返回可以通过 **URL** 来打印 **gif** 文件的打印机的列表：

```
DocFlavor flavor = DocFlavor.URL.GIF;
PrintService[ ] services = PrintServiceLookup.lookupPrintServices(flavor, null);
```

另外，如果你的程序已经获得了打印服务的实例，而你想知道它是否支持一种特定的属性，你可以调用 **isDocFlavorSupported()**方法。在下面的代码里，将得到一个默认打印机的引用，如果不支持打印出通过 **URL** 获得的 **gif**，就会出现错误信息：

```
PrintService service = PrintServiceLookup.lookupDefaultPrintService();
DocFlavor flavor = DocFlavor.URL.GIF;
if (!service.isDocFlavorSupported(flavor)) {
    System.err.println("The printer does not support the appropriate DocFlavor") ;
}
```

5．AttributeSet (属性集)

如你见，一个 **DocFlavor** 描述了要打印的数据，并且可以用来确定 **PrintService**(打印服务)是否支持该数据类型。然而，您的应用程序也可能需要一种基于打印机特性的选择机制。例如，你要打印的图片需要用不同的颜色来传递信息，你想知道给定的（打印）服务是否支持彩色打印，如果不是，那么要么不使用该打印机，或者转换成不依赖颜色的图片演示。

类似彩色打印，两面打印，以及不同的打印制方向选择（垂直肖像式或水平风景式）等特性被称为打印机属性，而 **javax.print.attribute** 包中包含了许多你可以用于描述这些属性的类和接口。其中的一个接口是 **AttributeSet**，可以作为前面提到的 **lookupPrintServices()** 中第二个参数。正如你预计的那样，**AttributeSet** 的一个实现代表了一组属性的集合，在调用 **lookupPrintServices()**时指定一个非空的值，将只返回支持这些属性的打印服务。换句话说，如果 **DocFlavor** 和 **AttributeSet** 都不为空，那么方法将返回那些这两种属性都支持的打印机

6．Attribute

给定的一个 **AttributeSet** 是一组属性的集合，一个显而易见的问题是，如何指定组成该集合的属性值呢？ **javax.print.attribute** 包里同时含有一个叫 **Attribute** 的接口，你马上可以看到通过调用 **add()**方法，来给 **AttributeSet** 添加若干个 **Attribute** 实例来获得这个集合。查阅 Attribute 接口的文档后，发现在 **javax.print.attribute.standard** 包里定义了大量你将要用到的实现。在你了解这些之前，你可以先查看 **javax.print.attribute** 这个包里的其他接口及其实现，将非常有帮助！

7．属性角色

目前为止，我们把属性描述成打印服务的能力，这在大部分上是正确的，至少对于 **Java** 是如何支持属性来说，它是某种意义上的单纯概括。对应每个不同属性，**java** 都将其关联到不同的角色上，属性仅在相关的角色上下文中才有效。换而言之，在不同的位置要使用不同的 **Java** 打印服务属性，不是每个属性在任何地方都适用。

为了更好的理解这个，来看一下 **javax.print.attribute.standard** 包里定义的 **OrientationRequested** 和 **ColorSupported** 实现。创建一个新的打印文档时，应该通过设定 **OrientationRequested** 属性来确定打印纸的方向（例如垂直肖像式或水平风景式）。与此相反，**ColorSupported** 是你在调用 **PrintService** 接口的 **getAttributes()**方法时返回的属性。换而言之，**OrientationRequested** 是一个你用来将信息传递给打印机的属性，而 **ColorSupported** 是打印服务用来提供给你关于打印机能力信息的属性。你不能在创建打印文档时把 **ColorSupported** 作为指定属性，因为打印机是否支持彩色打印是你的程序不能控制的。

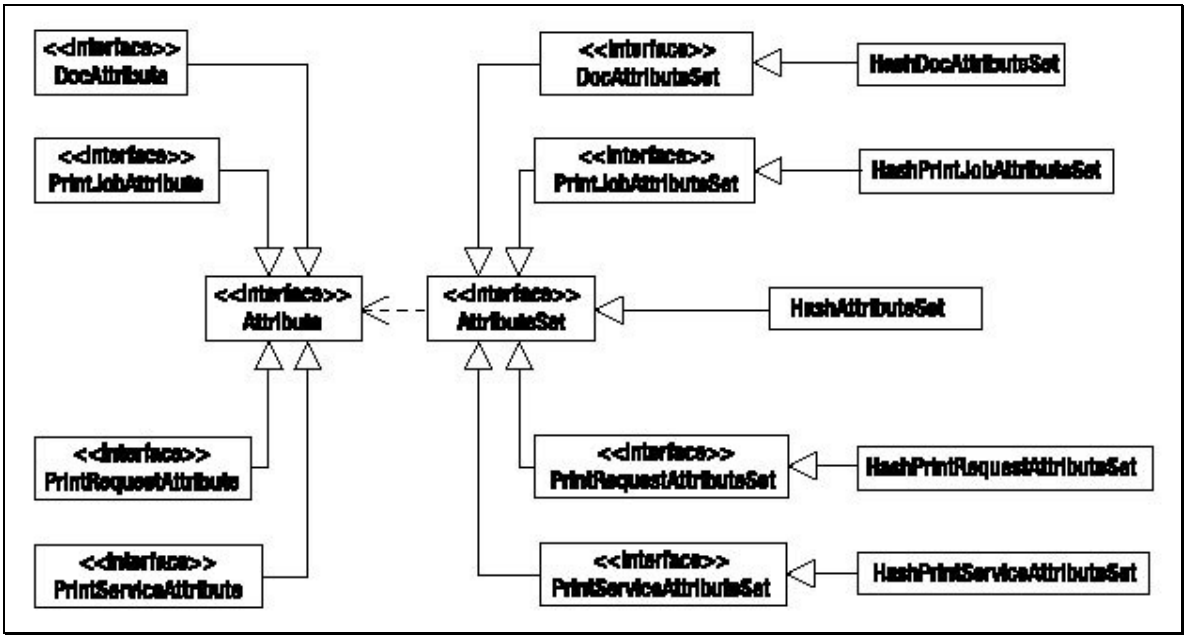
8．接口和实现

你第一次查看 **javax.print.attribute** 包里的接口和类时，选择列表里的接口和类，看起来很令人困惑。除了 **Attribute** 和 **AttributeSet** 接口，以及实现 **AttributeSet** 的 **HashAttributeSet** 类，**javax.print.attribute** 包里有 4 个子接口和类，列出在表 4 和图 1 中。

Table 4. javax.print.attribute 里定义的接口和类

Attribute subinterface	AttributeSet subinterface	AttributeSet subclass
DocAttribute	DocAttributeSet	HashDocAttributeSet
PrintJobAttribute	PrintJobAttributeSet	HashPrintJobAttributeSet
PrintRequestAttribute	PrintRequestAttributeSet	HashPrintRequestAttributeSet
PrintServiceAttribute	PrintServiceAttribute PrintServiceAttributeSet	HashPrintServiceAttributeSet

图-1，javax.print.attribute 包的一部分类的层次结构.



为什么你需要所有这些各式各样的接口和继承类呢，特别是已经有了 **Attribute, AttributeSet**, 和 **HashAttributeSet**？是因为这些特殊的定制是为了确保在角色中使用合适的有效属性。比方说，我提到过当你创建打印文档的位置，可以使用属性；但根据上下文，一些属性，例如 **ColorSupported** 在那里不能使用。当创建这样的文档时，你可以使用 **DocAttributeSet** 接口（或者更明确一点，**HashDocAttributeSet** 这个实现），这个实现只允许你添加继承 DocAttribute 这个接口的属性。这四种不同的角色如下：

- **Doc:** 在创建打印文档时，描述文档如何打印
- **PrintJob:** 从打印任务返回的属性，描述任务的状态
- **PrintRequest:** 请求初始化打印时，传给任务的属性
- **PrintService:** 由打印服务返回，用于描述打印机的能力

要知道这些如何工作，我们来创建一个 **DocAttributeSet** 的实例，然后尝试为 **AttributeSet** 设置 **DocAttributeSet** 和 **OrientationRequested** 属性。**HashDocAttributeSet** 定义了一个无参数的创建结构，所以你可以很容易地创建实例：

```
DocAttributeSet attrs = new HashDocAttributeSet();
```

现在你已经创建了 **AttributeSet**，你可以调用 **add()**方法，并把 **Attribute** 的实现传递给它。如果你看了 **OrientationRequested** 这个类的文档，你会发现它包含了一系列静态的 **OrientationRequest** 实例，每一个对应一种纸张打印方向，例如垂直人像或水平风景。要指定你想要的方向，你所要做的只是利用 **add()**方法传递一个静态实例的引用：

```
DocAttributeSet attrs = new HashDocAttributeSet();

attrs.add(OrientationRequested.PORTRAIT);
```

ColorSupported 类有一点不同，但一样很简单，它定义了两种静态实例：一个表示支持彩色打印，另一个表示不支持。你可以试着增加一个 **ColorSupported** 属性到 **DocAttributeSet** 去，代码如下：

```
DocAttributeSet attrs = new HashDocAttributeSet();

attrs.add(OrientationRequested.PORTRAIT);
```

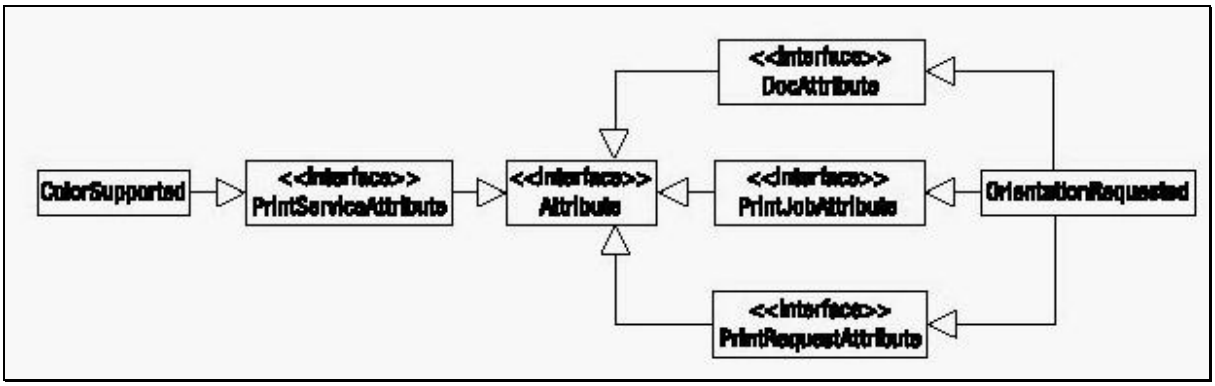
```
attrs.add(ColorSupported.SUPPORTED);
```

如前所述，去指定是否支持彩色打印不恰当的，因为这不是程序所能控制的内容。换句话说，**ColorSupported** 这个属性放到一系列文档属性上下文中并不合适，所以，运行先前的代码，当添加 **ColorSupported** 属性时会抛出一个 **ClassCastException** 异常。

要这是如何工作的，记住每一个 **AttributeSet** 子接口（这个例子是 **DocAttributeSet**）都有一个相应 **Attribute** 子接口（**DocAttribute**）和实现类（**HashDocAttributeSet**）。当添加一个属性时，实现子类试图把 **Attribute** 参数转换为相应的子接口类型，这样来确保只有当前上下文合适的属性会添加成功。

这个例子中，**HashDocAttributeSet** 的 **add()**方法第一次调用，是配合一个 **OrientationRequested** 的实例，并成功将该对象转换为 **DocAttribute**。因为，如图 2 所示，**OrientationRequested** 继承了那个接口。与之相反，传递 **ColorSupported** 实例的时候失败了，因为其没有继承 **DocAttribute**。

图-2. javax.print.attribute 包的部分类层次图示。



这个例子举例表明，表 4 里的四个接口和类组确保了只有合适的属性在合适的上下文中使用。注意各种角色和不同的属性之间有大量的重叠部分，所以很多属性与不止一个角色相关联。例如，许多属性继承了 **PrintJobAttribute** 和 **PrintRequestAttribute**，因为大部分维护和通过打印任务提供给你的属性，与你可以在初始化打印时能指定的属性是相关的。举例来说，你可以通过添加名称到 **PrintRequestAttributeSet** 中来指定打印任务名，并且在打印的时候，通过查询 **PrintJobAttributeSet** 来获得它。因此，**JobName** 属性类同时实现了 **PrintRequestAttribute** 和 **PrintJobAttribute**。

9．AttributeSet 和 HashAttributeSet

你已经知道了为什么会有四个子类，但是 **AttributeSet** 接口和 **HashAttributeSet** 父类又是什么呢？ **AttributeSet / HashAttributeSet** 在你不能确定要存储在这个集合中的属性仅仅和一个角色相关时使用。记得我以前提到的 **lookupPrintServices()**方法允许你指定 **AttributeSet** 参数来限制返回的打印服务。表面上看来最好指定 **PrintServiceAttributeSet** 的实例，但是很多你可能用到的属性并不继承 **PrintServiceAttribute**。

我们假设你想要让 **lookupPrintServices()** 方法返回，支持彩色打印和水平方向打印的打印机。这些属性与 **ColorSupported** 和 **OrientationRequested** 属性关联，但请注意这些类并不共享角色：前者是一个 **PrintServiceAttribute**，而 **OrientationRequested** 与另外三个角色 (**Doc**，**PrintRequest** 和 **PrintJob**)关联，如图-2 所示。这意味着不存在单个特定角色的 **AttributeSet** 接口或类来同时包含 **ColorSupported** 和 **Sides** 属性。

创建一个 AttributeSet，并使其同时包含 **OrientationRequested** 和 **ColorSupported** 实例的简单方法是使用一个 **HashAttributeSet**。与它的子类不同，它并不限制你往上加特殊角色的属性，所以以下代码可以成功执行：

```
AttributeSet attrs = new HashAttributeSet();

attrs.add(ColorSupported.SUPPORTED);

attrs.add(OrientationRequested.LANDSCAPE);

PrintService[ ] services = PrintServiceLookup.lookupPrintServices(null, attrs);
```

10．通过用户界面的打印机选择

到当前为止，我假设所需要使用的打印机，应该能够由应用程序编程选择。事实上，尽管如此，更为普遍的过程是显示一个对话框，并允许客户在输出时，选择使用哪个打印机。幸运的是，**Java** 通过使用在 **javax.print** 包中定义的 **ServiceUI** 类，中的静态方法 **printDialog()** 来使得这些操作非常简单。

除了对话框显示的位置外，在调用 **printDialog()**时必须指定的唯一参数是这些：

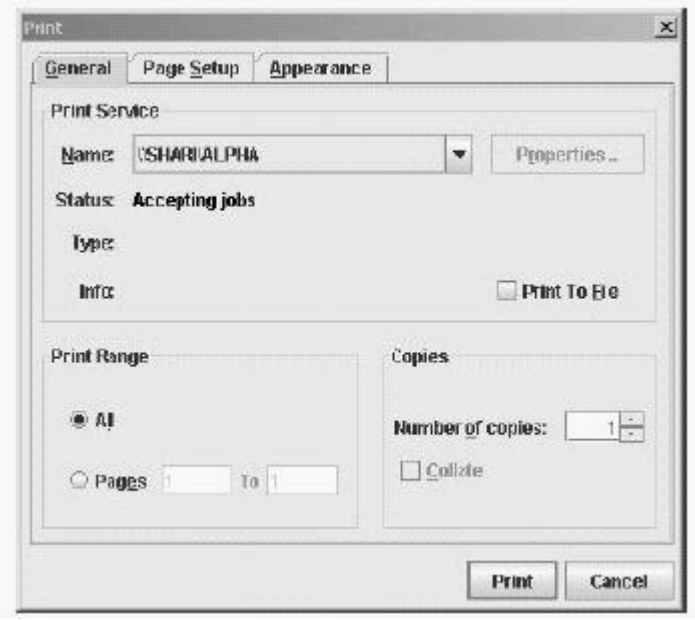
- 一个用户可选用的 **PrintService** 实例的数组；
- 默认的 **PrintService**；
- 一个 **PrintRequestAttributeSet** 实例。这用来弹出显示的对话框，并在对话框消失之前返回用户所作的任何更改。

要演示这些如何运作，可使用下列简单的代码段来显示一个简单的打印对话框：

```
PrintService[ ] services = PrintServiceLookup.lookupPrintServices(null, null);

PrintService svc = PrintServiceLookup.lookupDefaultPrintService( );

PrintRequestAttributeSet attrs = new HashPrintRequestAttributeSet( );

PrintService selection = ServiceUI.printDialog(

        null, 100, 100, services, svc, null, attrs);
```

运行时，代码产生如图-3 中所示的对话框

图-3 打印对话框



正如代码所示，从 **printDialog()**方法返回的值是一个 **PrintService** 实例，识别用户所选的打印机，或在用户取消打印机对话时标识为空值。此外，**PrintRequestAttributeSet** 已更新了，包含用户通过对话框所做出的更改，例如要打印的份数等。

通过使用 **printDialog()**方法，可让用户选择其输出要发往的打印机，提供用户对于专业应用程序的所期望功能。

11．创建打印任务

这是打印中的最简单的一个步骤；因为一旦获得 **PrintService** 的一个引用，所有你需要做的就是调用 **createPrintJob()**方法，象这样：

```
PrintService service;

.

.

.

DocPrintJob job = service.createPrintJob();
```

如代码所示，从 **createPrintJob()**返回的值是一个 **DocPrintJob** 实例，该对象可以让让您控制并监视打印操作的状态。要初始化打印，您应该调用 **DocPrintJob** 对象的 **print()**方法，但是，在这之前，您需要定义待打印的文档，并 **PrintRequestAttributeSet**。您已经 知道如何构造并使用 **AttributeSet**，这些步骤不再重复；接下来，您将了解如何定义待打印的文档。

12．定义要打印的文档

打印过程中接下的一步是定义要打印的文档，就是创建一个实例，其实现了在 **javax.print** 包里定义的 **Doc** 接口。每一个 **Doc** 的实例有两个必须定义的属性和一个可选择的属性：

- 一个 Object 代表要打印的内容；
- DocFlavor 的一个实例描述数据类型；
- 一个可选的 DocAttributeSet 包含打印时所需属性。

查阅 **Doc** 接口的文档，可以看出 **javax.print** 包里包含了一个叫 **SimpleDoc** 接口的实现，它的构造函数包含了与上面三个属性对应的三个参数。要了解如何构建 **SimpleDoc** 的实例，我们假设你要打印两份存在 http://www.apress.com/ApressCorporate/supplement/1/421/bcm.gif 的 **gif** 文件拷贝。

构建一个描述所要打印文件的 **SimpleDoc** 实例，我们所有要做的是，创建一个指向图片的 **URL**，并获得一个合适的 **DocFlavor** 引用，并把这两个传给 **SimpleDoc** 构造函数：

```
URL url = new

        URL("http://www.apress.com/ApressCorporate/supplement/1/421/bcm.gif");

DocFlavor flavor = DocFlavor.URL.GIF;

SimpleDoc doc = new SimpleDoc(url, flavor, null);
```

13．启动打印

打印的最后一个步骤就是调用 **DocPrintJob** 的 **print()**方法，传递给其，待打印数据的 **Doc** 对象，以及可选的 **PrintRequestAttributeSet** 实例。为简单起见，假设默认打印机支持你所需要的 **flavor** 和属性，在此情况下要使用下列代码将上一个例子提及的 **gif** 文件打印两份：

```
PrintService service = PrintServiceLookup.lookupDefaultPrintService();

DocPrintJob job = service.createPrintJob();

URL url = new
```

```
            URL("http://www.apress.com/ApressCorporate/supplement/1/421/bcm.gif ");

        DocFlavor flavor = DocFlavor.URL.GIF;

        Doc doc = new SimpleDoc(url, flavor, null);

        PrintRequestAttributeSet attrs = new HashPrintRequestAttributeSet( );

        attrs.add(new Copies(2));

        job.print(doc, attrs)
```

注意，某些情况下，打印是异步执行的，这可能会在实际打印完成之前，返回对 **print()** 的调用。

关于作者

Brett Spell 是一个 **Frito-Lay** 的资深开发者/高级员，并是著名的 **Pro Java Programming** 原始版本的作者。

资源

- 这是 Pro Java Programming, Second Edition, Brett Spell (Apress, June 2005; ISBN: 1590594746)第十章，“打印”的部分节选 http://www.apress.com/book/bookDisplay.html?bID=421
- 要得到更多关于 Java API 的文章，点击以下链接 http://www.javaworld.com/channel_content/jw-apis-index.shtml

-END-