

ANT 使用指南

本教程来源互连网，仅供学习，
版权归原作者及其出版商所有。

第一章 入门

本教程所讲述的内容

在本教程中，您将学习 Ant 这个 Java™ 项目生成工具。由于其灵活性和易用性，Ant 很快在 Java 开发人员中流行开来，因此您有必要了解关于它的更多信息。

在继续学习本教程之前，你不需要具备先前的 Ant 经验或知识。我们将首先查看 Ant 生成文件（build file）的基本结构，并学习如何调用这个工具。我们将逐步完成为一个简单 Java 项目编写生成文件的步骤，然后考察 Ant 的其他一些有用功能，包括文件系统操作和模式匹配。最后编写一个扩展 Ant 功能的自己的 Java 类来结束本教程。

在学习本教程的过程中，我们将同时展示如何从命令行以及从其他开放源代码 Eclipse IDE 运行 Ant。试验本教程中的例子不需要同时具备这两种环境；您可以选择其一，甚至选择某种不同的开发环境，只要该环境支持 Ant。如果选择从命令行使用 Ant，并且 Ant 还没有安装到机器上，您需要遵循 Ant 主页上的安装说明。相反，如果决定仅使用 Eclipse 环境，您不需要单独安装 Ant，因为 Eclipse 已经包括了它。如果还没有 Eclipse，您可以从 Eclipse.org 下载 Eclipse。

谁应该学习本教程？

如果您正在编写 Java 代码却还没有使用 Ant，那么本教程就是为您准备的。不管您当前是否在使用某种不同的生成工具，或者根本就没有使用生成工具，了解关于 Ant 的更多知识或许会促使您转而使用它。

如果已经在使用 Ant，那么您仍然可能在本教程中发现一些有趣的东西。或许您会发现一些预料之外或无法完全理解的 Ant 行为；本教程将会帮助您。或者，也许您熟悉 Ant 的基础，但是还想知道诸如将生成文件链接起来、使用 CVS 知识库或编写自定义任务等高级主题；本教程将会介绍所有这些主题。

Ant 主要是设计用于生成 Java 项目的，但这并不是它唯一的用途。许多人发现它对其他任务也有帮助。比如以跨平台的方式执行文件系统操作。此外，还有许多可用的第三方 Ant 任务，而编写自定义的 Ant 任务也是相对简单的，因此很容易针对特定的应用程序定制 Ant。

关于作者

Matt Chapman 1996 是英国 Hursley 的 IBM Centre for Java Technology 的咨询软件工程师。他过去七年来一直致力于 Java 技术，包括 Java 虚拟机实现和各类平台、用户界面工具包 Swing 和 AWT，以及近来为 Eclipse 平台所编写的工具。Matt 拥有计算机科学方面的学位，并且还是一名 Sun 认证的 Java 程序员。可通过 mchapman@uk.ibm.com 与他联系。

第二章 **Ant** 基础

简介

本节将概述 Ant 的功能和优势，并讨论它的历史概况和日渐提高的普及性。然后通过考察一个最基础的生成文件的基本结构，直接进入对 Ant 基础的讨论。我们还会介绍 属性 和 依赖关系 的概念。

Ant 是什么？

Apache Ant 是一个基于 Java 的生成工具。据最初的创始人 James Duncan Davidson 介绍，这个工具的名称是 **another neat tool**（另一个整洁的工具）的首字母缩写。

生成工具在软件开发中用来将源代码和其他输入文件转换为可执行文件的形式（也有可能转换为可安装的产品映像形式）。随着应用程序的生成过程变得更加复杂，确保在每次生成期间都使用精确相同的生成步骤，同时实现尽可能多的自动化，以便及时产生一致的生成版本，这就变得更加重要

了。C 或 C++ 中的传统项目经常使用 **make** 工具来做这件事情，其中生成任务是通过调用 **shell** 命令来执行的，而依赖关系定义在每个生成文件之间，以便它们总是以必需的顺序执行。

Ant 与 **make** 类似，它也定义生成文件之间的依赖关系；然而，与使用特定于平台的 **shell** 命令来实现生成过程所不同的是，它使用跨平台的 **Java** 类。使用 **Ant**，您能够编写单个生成文件，这个生成文件在任何 **Java** 平台上都一致地操作（因为 **Ant** 本身也是使用 **Java** 语言来实现的）；这就是 **Ant** 最大的优势。

Ant 的其他关键优势包括其突出的简单性和无缝地使用自定义功能来扩展它的能力。但愿您在完成本教程其余内容的学习之后，会欣赏 **Ant** 的这些优势。

Ant 简史

Ant 最初是 **Tomcat** 的一个内部组件，**Tomcat** 是 **Java Servlet** 和 **JavaServer Pages (JSP)** 参考实现中使用的 **servlet** 容器。**Tomcat** 代码基被捐赠给了 **Apache** 软件基金会；在那里它又成了 **Apache Jakarta** 项目的组成部分，该项目致力于为 **Java** 平台产生开放源代码的服务器端解决方案。**Ant** 的有用性很快得到了认可，对它的使用遍布在其他 **Jakarta** 子项目中。因而，它自己也成了一个 **Jakarta** 子项目，其第一个独立版本于 2000 年 7 月发布。

从那以后，**Ant** 的普及性已经不断地提高。它赢得了无数的行业大奖，并成为用于生成开放源代码 **Java** 项目的 事实上 的标准。2002 年 11 月，这些成功得到了确认，**Ant** 被提升为顶级 **Apache** 项目。

在本文编写之际，**Ant** 的当前稳定版本是 1.5.4，它支持 1.1 以后的所有 **JDK** 版本。下一个版本（即 1.6 版）的 **beta** 版也已经可用，这些版本需要 **JDK 1.2** 或更高版本。未来的 2.0 版也正在计划之中，它将涉及一次重大的体系结构重新设计。**Ant 2.0** 将以改进的一致性和增强的功能为特色，同时仍然保持 **Ant** 的简单性、易于理解性和可扩展性等核心目标。

Ant 生成文件剖析

Ant 没有定义它自己的自定义语法；相反，它的生成文件是用 XML 编写的。存在一组 Ant 能够理解的预定义 XML 元素，而且就像您将在下一节中看到的一样，还可以定义新的元素来扩展 Ant 的功能。每个生成文件由单个 **project** 元素组成，该元素又包含一个或多个 **target** 元素。一个目标（**target**）是生成过程中已定义的一个步骤，它执行任意数量的操作，比如编译一组源文件。这些操作本身是由其他专用任务标签执行的，我们将在后面看到这一点。然后这些任务将根据需要被分组到各个 **target** 元素中。一次生成过程所必需的所有操作可以放入单个 **target** 元素中，但是那样会降低灵活性。将那些操作划分为逻辑生成步骤，每个步骤包含在它自己的 **target** 元素中，这样通常更为可取。这样可以执行整体生成过程的单独部分，却不一定要执行其他部分。例如，通过仅调用某些目标，您可以编译项目的源代码，却不必创建可安装的项目映像。

顶级 **project** 元素需要包含一个 **default** 属性，如果在 Ant 被调用时没有指定目标，这个属性将指定要执行的目标。然后需要使用 **target** 元素来定义该目标本身。下面是一个最基本的生成文件：

```
<?xml version="1.0"?>
<project default="init">
  <target name="init">
  </target>
</project>
```

注意这是一个结构良好的 XML 文档，其中一个 XML 声明指定了所使用的 XML 的版本（这不是当前的 Ant 所必需的，但是这样做是一个好习惯），而且每个元素都正确地关闭了。一次性打开和关闭一个元素也是可以做到的。因此，与其像上面那样对 **target** 元素使用单独的起始和结束标签，我们可以将它写为如下形式：

```
<target name="init"/>
```

当元素没有包含任何内容时，更简练的形式会更清晰。

添加描述：

我们在前一小节中看到的生成文件是优雅简练的，但它并没有包含多少关于正在生成的实际项目的信息。可以通过许多方式来使它更具描述性，同时无需改变其功能。下面是一个例子：

```
<?xml version="1.0"?>
<project default="init" name="Project Argon">
    <description>
A simple project introducing the use of descriptive tags in Ant build files.
    </description>

    <!-- XML comments can also be used -->
    <target name="init" description="Initialize Argon database">
        <!-- perform initialization steps here -->
    </target>
</project>
```

可以看出，XML 注释可以使用在整个生成文件中以提高清晰性。而且，Ant 定义了自己自己的 `description` 元素和 `description` 属性，它们可用于提供更结构化的注释。

属性

Ant 中的 属性 类似编程语言中的变量，它们都具有名称和值。然而与通常的变量不同，一经设置，Ant 中的属性就不可更改；它们是不可变的，就像 Java 语言中的 `String` 对象。这起初看来似乎很有限制性，但这样是为了遵循 Ant 的简单原则：毕竟，它是一个生成工具，而不是一种编程语言。如果尝试给某个现有属性赋予一个新的值，这不会被看作是一个错误，但是该属性仍然会保留其现有值。（我们将会看到，这种行为是有用的。）

基于元素的描述性名称和到目前为止所见到的属性，在 Ant 中用于设置属性的机制看起来如下就没有什么奇怪了：

```
<property name="metal" value="beryllium" />
```

为了在生成文件的其他部分引用这个属性，您会使用以下语法：

```
${metal}
```

例如，为了使用这样一个值，它是另一个属性的值的组成部分，您会将标签写为下面这样：

```
<property name="metal-database" value="${metal}.db" />
```

Ant 中有许多预定义的属性。首先, Java 环境设置用于运行 Ant 的所有系统属性, 均可作为 Ant 属性使用, 比如 `${user.home}`。除了这些属性之外, Ant 还定义了它自己的一小组属性, 包括 `${ant.version}`, 这个属性包含 Ant 的版本; 以及 `${basedir}`, 这个属性是项目目录的绝对路径 (由包含生成文件的目录所定义, 或者由 `project` 元素的可选 `basedir` 属性所定义)。

属性经常用于引用文件系统上的文件或目录, 但是对于使用不同路径分隔符 (例如, / 与 \) 的平台来说, 这样可能在跨越不同平台时导致问题。Ant 的 `location` 属性专门设计用于以平台无关的方式包含文件系统路径。您会像下面这样使用 `location` 来代替 `value`:

```
<property name="database-file" location="archive/databases/${metal}.db"/>
```

用于 `location` 属性的路径分隔字符将被转换为当前平台的正确格式; 而且由于文件名是相对的, 它被认为是相对于项目的基目录。我们同样可以容易地写为下面这样:

```
<property name="database-file" location="archive\databases\${metal}.db"/>
```

这个标签的两个版本都会在不同的平台具有相同的行为。如果可移植性是必需的, 唯一要避免的内容就是文件名中的 DOS 风格的驱动器号。在可能的地方使用相对路径名称而不是绝对路径名称, 这样还会更加灵活。

定义依赖关系

生成一个项目一般需要许多步骤 —— 例如首先要编译源代码, 然后将它打包为 Java 归档文件 (Java Archive File, JAR)。这其中许多步骤都具有清楚定义的顺序 —— 例如, 在编译器从源代码生成类文件之前, 您不能打包类文件。与顺序指定 `target` 所不同的是, Ant 采用一种更灵活的方法来定义 依赖关系, 就像 `make` 和类似的生成工具所做的那样。每个目标的定义依据的是它在能够执行之前必须完成的其他所有目标。这是使用 `target` 元素的 `depends` 属性来实现的。例如:

```
<target name="init"/>
<target name="preprocess" depends="init"/>
<target name="compile" depends="init,preprocess"/>
<target name="package" depends="compile"/>
```

这种方法允许您执行项目任何阶段的生成过程；Ant 会首先执行已定义的先决阶段。在上面的例子中，如果让 Ant 完成 `compile` 步骤，它将判断出需要首先执行 `init` 和 `preprocess` 这两个目标。`init` 目标不依赖其他任何目标，因此它将首先被执行。然后 Ant 检查 `preprocess` 目标，发现它依赖 `init` 目标；由于已经执行了后者，Ant 不会再次执行它，因而开始执行 `preprocess` 目标。最后可以执行 `compile` 任务本身。注意目标出现在生成文件中的顺序并不重要：执行顺序是由 `depends` 属性唯一确定的。

第三章 运行 ANT

简介

Apache Ant 可通过各种不同的方式来调用。就其本身而言，Ant 是一个命令行形式的工具，通常从 UNIX 或 Linux shell 提示符或者 Windows 命令提示符调用，生成文件则使用您自己选择的文本编辑器来编写。如果要生成的项目是以这种方式开发的，那么这样调用 Ant 很好，但是许多人发现 IDE 更方便。大多数 IDE 都对 Ant 提供了某种程度的支持，因此在使用 IDE 的情况下，最起码，您不必麻烦地离开 IDE 来执行命令行操作就能调用 Ant 生成任务。

在本节中，我们将考察如何从命令行使用 Ant，并了解一些有用的命令行选项。然后简要了解一下开放源代码的 Eclipse 平台提供的 Ant 支持。（为了最充分地利用下面这些小节讲述的内容，您至少应该被动地熟悉 Eclipse。）

从命令行运行 Ant

从命令提示符调用 Ant 可以简单得只需键入单独的 `ant`。如果您这样做，Ant 将使用默认的生成文件；该生成文件中指定的默认目标就是 Ant 尝试要生成的目标。还可以指定许多命令行选项，后面跟着任意数量的生成目标，Ant 将按顺序生成这其中的每个目标，并在此过程中解决所有依赖关系。

下面是从命令行执行的 Ant 生成任务的一些典型输出：

```
Buildfile: build.xml

init:
    [mkdir] Created dir: E:\tutorials\ant\example\build
    [mkdir] Created dir: E:\tutorials\ant\example\dist

compile:
    [javac] Compiling 8 source files to E:\tutorials\ant\example\build

dist:
    [jar] Building jar: E:\tutorials\ant\example\dist\example.jar

BUILD SUCCESSFUL
Total time: 2 seconds
```

随着我们继续本教程的学习，我们将弄明白所有这些输出意味着什么。

命令行选项

就像 **make** 工具默认情况下寻找一个名为 **makefile** 的生成文件一样，Ant 寻找一个名为 **build.xml** 的文件。因此，如果您的生成文件使用这个名称，就不需要在命令行指定它。当然，有时使用具有其他名称的生成文件更方便，在那样的情况下，您需要对 Ant 使用 **-buildfile <file>** 参数（**-f <file>** 是其简写形式）。

另一个有用的选项是 **-D**，它用于设置随后可以在生成文件中使用的属性。这对于配置您想要以某种方式开始的生成过程是非常有用的。例如，为了将 **name** 属性设置为某个特定的值，您会使用一个类似下面这样的选项：

```
-Dmetal=beryllium
```

这个功能可用于覆盖生成文件中的初始属性设置。正如前面指出过的，属性的值一经设置就不能改变。**-D** 标志在读取生成文件中的任何信息之前设置某个属性；由于生成文件中的指派落在这个初始指派之后，因此它不会改变其值。

IDE 集成

由于 Ant 的普及性，大多数现代 IDE 现在都集成了对它的支持，其他许多 IDE 则在插件提供对它的支持。受支持的环境列表包括 JEdit 和 Jext 编辑器、Borland JBuilder、IntelliJ IDEA、Java Development Environment for Emacs (JDEE)、NetBeans IDE、Eclipse 以及 WebSphere® Studio Application Developer。

请参阅参考资料以了解关于 Ant 的广泛 IDE 支持的更多信息。在本节的其余部分，我们将探讨开放源代码的 Eclipse 环境所包括的 Ant 支持。

Eclipse 对 Ant 的支持

开放源代码的 Eclipse 项目提供了对 Ant 的大量支持。这些支持的核心是 Eclipse 的 Ant 编辑器，它以语法高亮显示为特色。该编辑器图示如下：

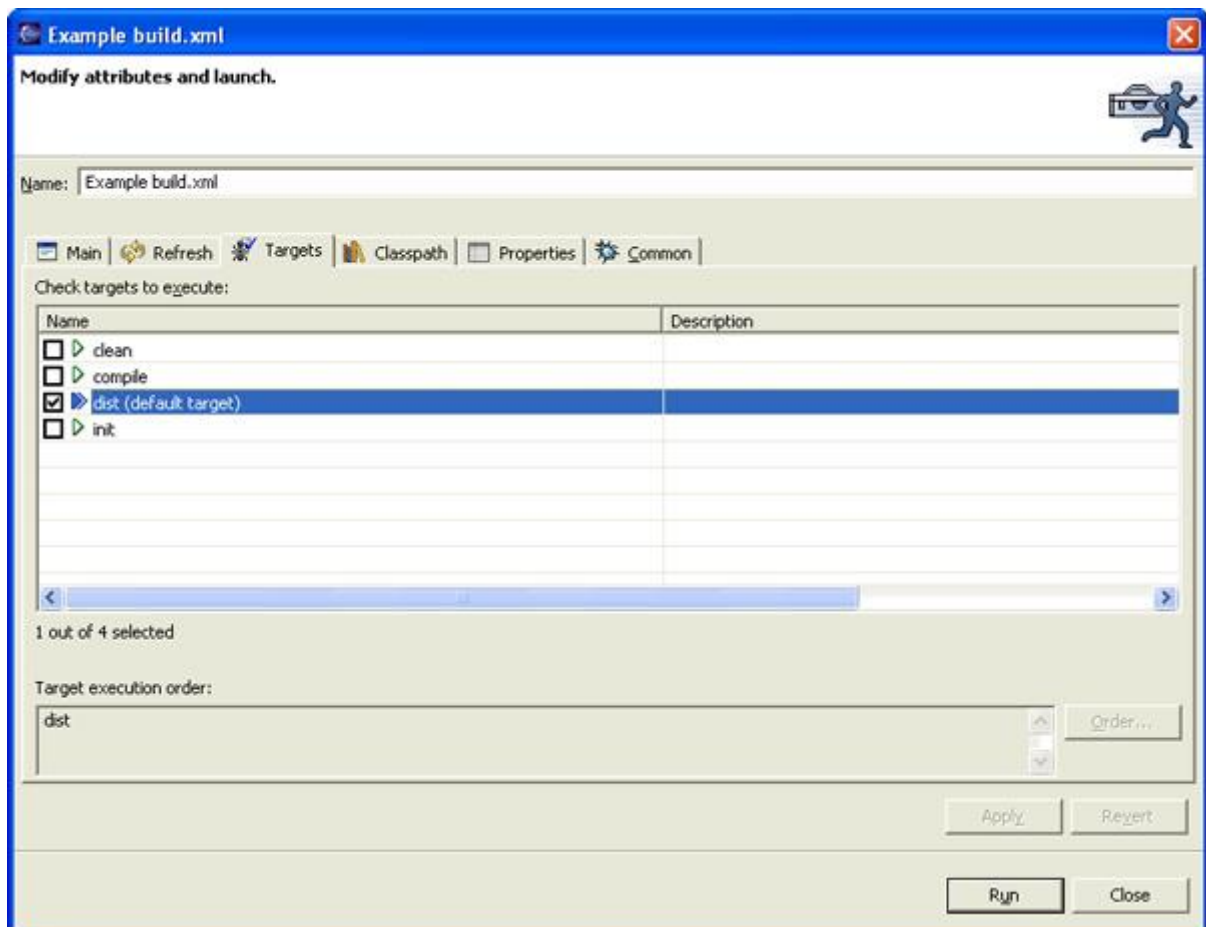


这个编辑器提供内容辅助 —— 例如，键入 `<pro` 然后按 `Ctrl-Space`，这样将会显示一个自动完成列表，其中包含 `<property>` 标签，以及对该任务的简要说明。

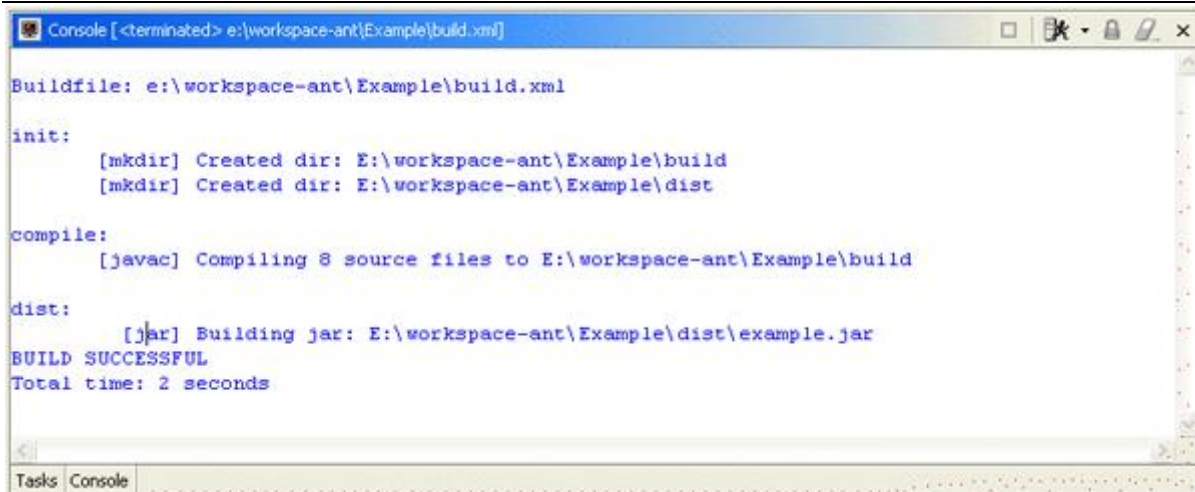
您还可以看到一个大图视图，它显示生成文件的结构，并提供在该文件中导航的便捷方式。此外还有一个 Ant 视图，它允许根据许多不同的 Ant 文件生成目标。

从 Eclipse 内运行 Ant 生成任务

名为 build.xml 的文件在 Eclipse 的导航程序视图中使用一个 Ant 图标来标识和装饰。右键单击这些文件会提供一个 Run Ant... 菜单选项，选择这个菜单选项将打开一个类似如下的对话框：



来自该生成文件的所有目标都显示出来了，而默认的目标则处于选中状态。在您决定是否要改变默认目标之后，请按 Run 按钮来运行 Ant。Eclipse 将切换到 Console 视图，如下图所示。错误将以不同的颜色显示出来，可以单击输出中的任务名称来跳到生成文件中的对应调用点。



```
Console [<terminated> e:\workspace-ant\Example\build.xml]

Buildfile: e:\workspace-ant\Example\build.xml

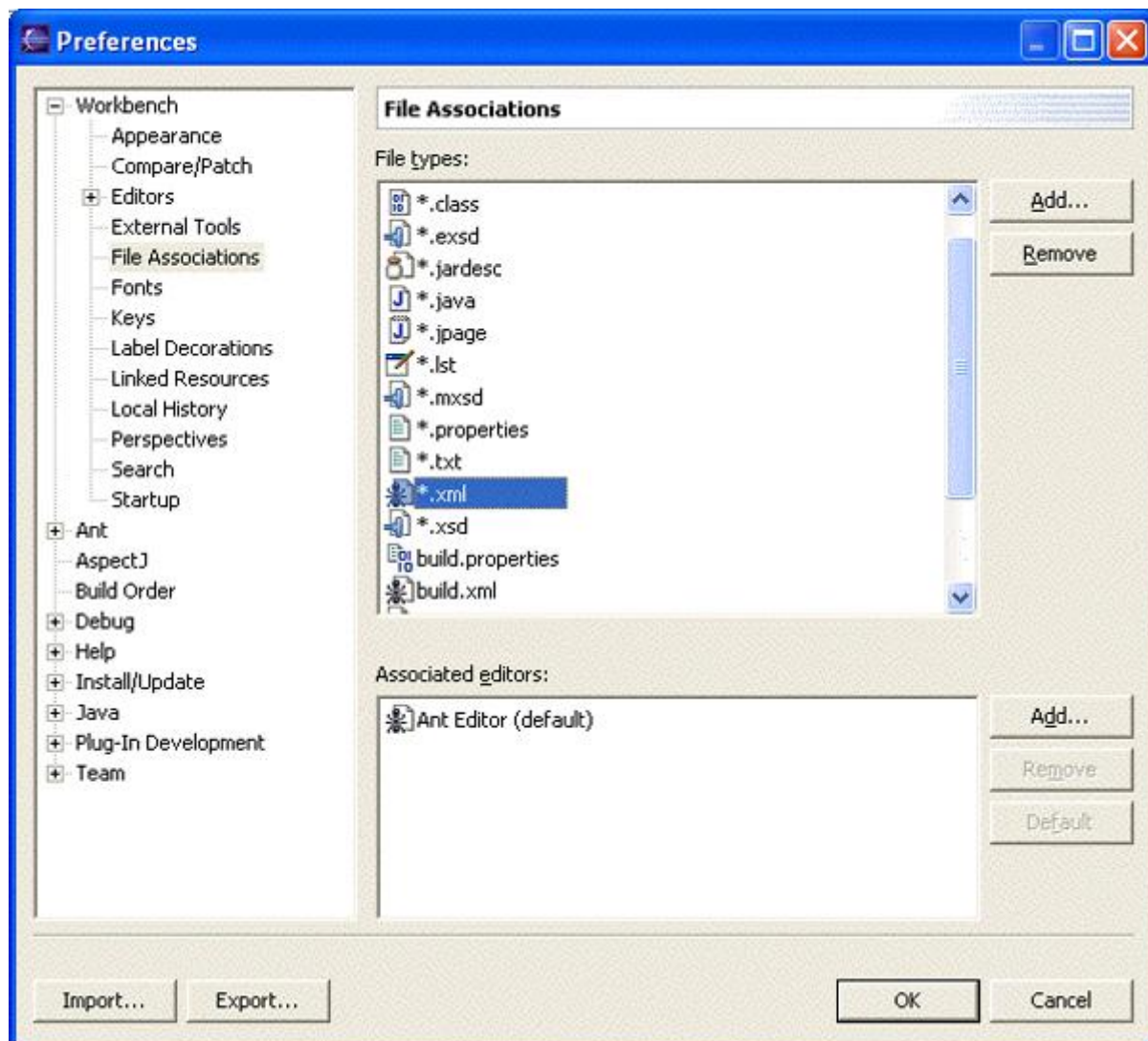
init:
    [mkdir] Created dir: E:\workspace-ant\Example\build
    [mkdir] Created dir: E:\workspace-ant\Example\dist

compile:
    [javac] Compiling 8 source files to E:\workspace-ant\Example\build

dist:
    [jar] Building jar: E:\workspace-ant\Example\dist\example.jar
BUILD SUCCESSFUL
Total time: 2 seconds
```

Eclipse 中的文件关联

默认情况下，Eclipse 仅对名为 `build.xml` 的文件使用 Ant 编辑器，不过可以容易地配置该编辑器，使其识别具有其他名称的文件。从菜单上选择 `Window=>Preferences`，然后展开 `Workbench` 组，再选择 `File Associations` 参数设置页面。然后为预期的文件名添加一种新的文件类型。例如，可以为名为 `mybuild.xml` 的所有文件添加一种新的文件类型。如果想对具有 `.xml` 后缀的所有文件（特殊文件名除外，比如 `plugin.xml`，它在 Eclipse 中覆盖通配符指定）做同样的事情，您甚至可以使用 `*.xml`。最后为这种新的文件类型添加一个关联的编辑器，然后从编辑器列表上选择 `Ant editor`，如下所示：



第四章 生成一个简单的 **JAVA** 项目

简介

现在我们已经清楚了 Ant 生成文件的格式，并了解了如何定义属性和依赖关系以及如何运行 Ant，下面可以开始为一个基本的 Java 项目构建一个生成环境了。这将包括学习用于编译源代码和组合 JAR 文件的 Ant 任务。

编译源代码

由于 Ant 的主要目标是生成 Java 应用程序，它能够内在地、出色地支持调用 `javac` 编译器以及其他 Java 相关任务就毫不奇怪了。下面是编译 Java 代码的任务的编写方式：

```
<javac srcdir="src"/>
```

这个标签寻找 `src` 目录中以 `.java` 为扩展名的所有文件，并对它们调用 `javac` 编译器，从而在相同的目录中生成类文件。当然，将类文件放在一个单独的目录结构中通常会更清晰；可以通过添加 `destdir` 属性来让 Ant 做到这点。其他有用的属性包括：

- `classpath`：等价于 `javac` 的 `-classpath` 选项。
- `debug="true"`：指示编译器应该带调试信息编译源文件。

`javac` 任务的一个重要特点在于，它仅编译那些它认为需要编译的源文件。如果某个类文件已经存在，并且对应的源文件自从该类文件生成以来还没有改变过，那么该源文件就不会被重新编译。`javac` 任务的输出显示了实际被编译的源文件的数目。编写一个 `clean` 目标来从目标目录移除生成的任何类文件是个很好的习惯。如果想要确保所有源文件都已编译，就可以使用这个任务。这种行为刻画了 Ant 的许多任务的特点：如果某个任务能够确定所请求的操作不需要执行，那么该操作就会被跳过。

像 Ant 一样，`javac` 编译器本身也是用 Java 语言实现的。这对 Ant 中的 `javac` 任务的使用来说非常有利，因为它通常调用 Ant 运行所在的相同 Java 虚拟机（JVM）中的编译器类。在每次需要编译 Java 代码时，其他生成工具通常需要运行一个新的 `javac` 进程，从而需要一个新的 JVM 实例。但是在使用 Ant 的情况下，只需要单个 JVM 实例，它既用于运行 Ant 本身，也用于执行所有必需的编译任务（以及其他相关任务，比如处理 JAR 文件）。这是一种高效得多的资源使用方式，能够极大地缩短项目生成时间。

编译器选项

正如我们从前一小节看到的，Ant 的 `javac` 任务的默认行为是调用运行 Ant 本身的任何 JVM 的标准编译器。然而，有时您可能想要单独地调用编译器——例如当你希望指定编译器的某些内存选

项，或者需要使用一种不同级别的编译器的时候。为实现这个目的，只需将 `javac` 的 `fork` 属性设置为 `true`，比如像下面这样：

```
<javac srcdir="src" fork="true"/>
```

如果想要指定一个不同的 `javac` 可执行文件，并向它传递一个最大内存设置，您可以像下面这样做：

```
<javac srcdir="src" fork="true" executable="d:\sdk141\bin\javac"
memoryMaximumSize="128m"/>
```

甚至可以将 Ant 配置为使用某种不同的编译器。受支持的编译器包括开放源代码的 `Jikes` 编译器和来自 GNU 编译器集 (GNU Compiler Collection, GCC) 的 `GCC` 编译器。（请参阅参考资料以了解关于这两种编译器的更多信息。）可以通过两种方式指定这些编译器：可以设置 `build.compiler` 属性，这将应用于使用 `javac` 任务的所有场合；或根据需要设置每个 `javac` 任务中的 `compiler` 属性。

`javac` 任务还支持其他许多选项。请参考 Ant 手册以了解更多细节（请参阅参考资料）。

创建 JAR 文件

在编译 Java 源文件之后，结果类文件通常被打包到一个 JAR 文件中，这个文件类似 zip 归档文件。每个 JAR 文件都包含一个清单文件，它可以指定该 JAR 文件的属性。

下面是 Ant 中 `jar` 任务的一个简单使用例子：

```
<jar destfile="package.jar" basedir="classes"/>
```

这将创建一个名为 `package.jar` 的 JAR 文件，并把 `classes` 目录中的所有文件添加到其中（JAR 文件能够包含任意类型的文件，而不只是类文件）。此处没有指定清单文件，因此 Ant 将提供一个基本的清单文件。

`manifest` 属性允许指定一个用作该 JAR 文件的清单的文件。清单文件的内容还可以使用 `manifest` 任务在生成文件中指定。这个任务能够像文件系统写入一个清单文件，或者能够实际嵌套在 `jar` 之内，以便一次性地创建清单文件和 JAR 文件。例如：

```
<jar destfile="package.jar" basedir="classes">
  <manifest>
    <attribute name="Built-By" value="${user.name}" />
    <attribute name="Main-class" value="package.Main" />
  </manifest>
</jar>
```

时间戳生成

在生成环境中使用当前时间和日期，以某种方式标记某个生成任务的输出，以便记录它是何时生成的，这经常是可取的。这可能涉及编辑一个文件，以便插入一个字符串来指定日期和时间，或将这个信息合并到 JAR 或 zip 文件的文件名中。

这种需要是通过简单但是非常有用的 `tstamp` 任务来解决的。这个任务通常在某次生成过程开始时调用，比如在一个 `init` 目标中。这个任务不需要属性，许多情况下只需 `<tstamp/>` 就足够了。

`tstamp` 不产生任何输出；相反，它根据当前系统时间和日期设置 Ant 属性。下面是 `tstamp` 设置的一些属性、对每个属性的说明，以及这些属性可被设置到的值的例子：

属性	说明	例子
DSTAMP	设置为当前日期，默认格式为 <code>yyyymmdd</code>	20031217
TSTAMP	设置为当前时间，默认格式为 <code>hhmm</code>	1603
TODAY	设置为当前日期，带完整的月份	2003 年 12 月 17 日

例如，在前一小节中，我们按如下方式创建了一个 JAR 文件：

```
<jar destfile="package.jar" basedir="classes"/>
```

在调用 `tstamp` 任务之后，我们能够根据日期命名该 JAR 文件，如下所示：

```
<jar destfile="package-${DSTAMP}.jar" basedir="classes"/>
```

因此，如果这个任务在 2003 年 12 月 17 日调用，该 JAR 文件将被命名为 `package-20031217.jar`。

还可以配置 `tstamp` 任务来设置不同的属性，应用一个当前时间之前或之后的时间偏移，或以不同的方式格式化该字符串。所有这些都是使用一个嵌套的 `format` 元素来完成的，如下所示：

```
<tstamp>
  <format property="OFFSET_TIME"
    pattern="HH:mm:ss"
    offset="10" unit="minute"/>
</tstamp>
```

上面的清单将 `OFFSET_TIME` 属性设置为距离当前时间 10 分钟之后的小时数、分钟数和秒数。

用于定义格式字符串的字符与 `java.text.SimpleDateFormat` 类所定义的那些格式字符相同。

综合

前面几小节为我们提供了生成简单 Java 项目所需的足够知识。下面将把这些代码片断组合成一个完整的生成文件，它将编译 `src` 目录下的所有源代码，将结果类文件放在 `build` 目录下，然后把所有类文件打包到 `dist` 目录中的一个 JAR 文件中。要自己试验这个生成文件，您所需要的就是包含一个或多个 Java 源代码文件的 `src` 目录——这个目录可以包含从简单的“Hello World”程序到来自某个现有项目的大量源文件的任何内容。如果需要向 Java classpath 添加 JAR 文件或其他任何内容，以便成功地编译源代码，您只需在 `javac` 任务中为其添加一个 `classpath` 属性。

该生成文件看起来如下：

```
<?xml version="1.0"?>
<project default="dist" name="Project Argon">
  <description>A simple Java project</description>

  <property name="srcDir" location="src"/>
  <property name="buildDir" location="build"/>
  <property name="distDir" location="dist"/>

  <target name="init">
    <tstamp/>
    <mkdir dir="${buildDir}"/>
    <mkdir dir="${distDir}"/>
  </target>
```

```
<target name="compile" depends="init">
    <javac srcdir="${srcDir}" destdir="${buildDir}"/>
</target>

<target name="dist" depends="compile">
    <jar destfile="${distDir}/package-${DSTAMP}.jar" basedir="${buildDir}">
        <manifest>
            <attribute name="Built-By" value="${user.name}"/>
            <attribute name="Main-Class" value="package.Main"/>
        </manifest>
    </jar>
    <jar destfile="${distDir}/package-src-${DSTAMP}.jar" basedir="${srcDir}"/>
</target>

<target name="clean">
    <delete dir="${buildDir}"/>
    <delete dir="${distDir}"/>
</target>
</project>
```

下面是使用该文件执行的某次生成过程的示例输出（您得到的输出可能不一样，具体取决于 **src** 目录的内容）：

Buildfile: build.xml

init:

[mkdir] Created dir: E:\tutorial\javaexample\build

[mkdir] Created dir: E:\tutorial\javaexample\dist

compile:

[javac] Compiling 10 source files to E:\tutorial\javaexample\build

dist:

[jar] Building jar: E:\tutorial\javaexample\dist\package-20031217.jar

[jar] Building jar: E:\tutorial\javaexample\dist\package-src-20031217.jar

BUILD SUCCESSFUL

Total time: 5 seconds

注意 JAR 文件是根据当前日期来命名的，并且为应用程序的主类设置了一个清单条目，以便主类能够通过一个简单的命令 `java -jar package-20031217.jar` 来直接运行。我们还创建了一个 JAR 文件，它仅包含项目的源代码。

第五章 文件系统操作

简介

我们了解了关于 Ant 的足够多的知识，现在能够生成一个基本的 Java 项目了，不过现实中的项目当然很少像我们的例子那样简单。在下面几节中，我们将考察 Ant 的许多附加功能中的一部分，以及能够使用它们的场合。

在本节中，我们将考察如何执行常见文件操作，比如创建目录和解压缩文件。Ant 的优秀特性之一在于，执行这些操作的任务一般在所有平台上都是相同的。

创建和删除目录

最基本的文件系统操作之一就是创建目录或文件夹。做这项工作的任务名为 `mkdir`，毫不奇怪，它非常类似于具有相同名称的 Windows 和 UNIX/Linux 命令。

```
<mkdir dir="archive/metals/zinc"/>
```

首先要注意 `/` 被用作目录分隔符，这是 UNIX 和 Linux 的惯例。您可能认为这不是很平台无关的，但是 Ant 知道如何处理它，并针对它运行所在的平台做恰当的事情，这与我们在前面定义基于位置的属性时所看到的方式相同。我们能够同样容易地使用 `\`，而不管平台是什么——Ant 能够处理任一种形式，甚至能够处理两种形式的混合。

`mkdir` 任务的另一个有用特性是它的如下能力：在父目录还不存在时创建它们。考虑一下上面的清单，设想 `archive` 目录存在，但是 `metals` 目录不存在。如果使用底层平台的 `mkdir` 命令，您需要首先显式地创建 `metals` 目录，然后第二次调用 `mkdir` 命令来创建 `zinc` 目录。但是 Ant 任务

比这更加智能，它能够一次性创建这两个目录。类似地，如果目标目录已经存在，`mkdir` 任务不会发出错误消息，而只是假设它的工作已经完成，从而什么也不做。

删除目录同样也很容易：

```
<delete dir="archive/metals/zinc"/>
```

这将删除指定的目录连同它包含的所有文件以及子目录。使用 `file` 属性而不是 `dir` 属性可以指定要删除的单个文件。

复制和移动文件及目录

在 Ant 中制作文件的一份拷贝很简单。例如：

```
<copy file="src/Test.java" tofile="src/TestCopy.java"/>
```

您还可以使用 `move` 来执行重命名操作而不是拷贝文件：

```
<move file="src/Test.java" tofile="src/TestCopy.java"/>
```

另一个常用的文件系统操作是将文件复制或移动到另一个目录。做这项工作的 Ant 语法同样也很简单：

```
<copy file="src/Test.java" todir="archive"/>
<move file="src/Test.java" todir="archive"/>
```

默认情况下，Ant 仅输出它执行的移动和复制操作的摘要，包括诸如已移动或复制的文件的数量等信息。如果想看到更详细的信息，包括涉及的文件名称等，您可以将 `verbose` 属性设置为 `true`。

创建和解压缩 zip 及 tar 文件

在前一节中，我们看到了如何创建 JAR 文件。创建其他归档文件的过程几乎完全相同。下面是创建 zip 文件的 Ant 任务：

```
<zip destfile="output.zip" basedir="output"/>
```

相同的语法也可用于创建 **tar** 文件。 还可以使用 **GZip** 和 **BZip** 任务来压缩文件。例如：

```
<gzip src="output.tar" zipfile="output.tar.gz"/>
```

解压缩和提取文件同样也很简单：

```
<unzip src="output.tar.gz" dest="extractDir"/>
```

还可以包括 **overwrite** 属性来控制覆盖行为。默认设置是覆盖与正在被提取的归档文件中的条目相匹配的所有现有文件。相关的任务名称是 **untar**、**unjar**、**gunzip** 和 **bunzip2**。

替换文件中的标记

我们将在本节考察的最后一个文件系统操作是 **replace** 任务，它执行文件中的查找和替换操作。**token** 属性指定要查找的字符串，**value** 属性指定一个新的字符串，查找到的标记字符串的所有实例都被替换为这个新的字符串。例如：

```
<replace file="input.txt" token="old" value="new"/>
```

替换操作将在文件本身之内的适当位置进行。为了提供更详细的输出，可把 **summary** 属性设置为 **true**。这将导致该任务输出找到和替换的标记字符串实例的数目。

第六章 其它有用的任务和技术

简介

在考察自定义的任务之前，我们首先介绍一些还没遇到过的有用功能。**Ant** 标准地附带了大量的功能，因此这里仅经挑选其中几个最有用的功能。模式匹配和文件选择器是功能强大的机制，它们极大地增强了我们已看到过的一些任务的功能；将生成任务链接起来以及与 **CVS** 知识库协同工作，是已发现的这些机制的两个主要实际应用领域。

模式匹配

在前面考察文件系统任务时，我们仅使用了单独地命名的文件和目录。然而，一次对一组文件执行那些操作经常是有用的——例如对给定目录中以 `.java` 结尾的所有文件执行操作。正如等价的 DOS 和 UNIX 命令提供了这样的功能一样，Ant 也提供了这样的功能。这是使用通配符字符来完成的：`*`，它匹配零个或多个字符；以及 `?`，它仅匹配一个字符。因而匹配以 `.java` 结尾的所有文件的模式不过就是 `*.java`。

也可以对目录执行模式匹配。例如，模式 `src/*.java` 将匹配带 `src` 前缀的任何目录中的所有 Java 文件。还有另一种模式结构：`**`，它匹配任意数量的目录。例如，模式 `**/*.java` 将匹配当前目录结构下的所有 Java 文件。

您能够以相当一致的方式对文件系统任务使用模式，比如嵌套的 `fileset` 元素。先前，我们使用这个任务来复制单个文件：

```
<copy file="src/Test.java" todir="archive"/>
```

如果我们想要使用一个模式，可以将 `file` 属性替换为一个 `fileset` 元素，如下所示：

```
<copy todir="archive">
  <fileset dir="src">
    <include name="*.java"/>
  </fileset>
</copy>
```

`fileset` 默认情况下包含指定 `src` 目录下的所有文件，因此为了仅选择 Java 文件，我们对模式使用一个 `include` 元素。类似地，我们可以对另一个模式添加一个 `exclude` 元素，从而潜在地排除 `include` 指定的匹配项。甚至可以指定多个 `include` 和 `exclude` 元素；这样将得到一组文件和目录，它们包含 `include` 模式的所有匹配项的并集，但排除了 `exclude` 模式的所有匹配项。

注意还有一个通常很有用的文件集特性，但是对于没有意识到它的人来说，这个特性偶尔会产生混淆。这个特性称为 **默认排除**：即自动从文件集内容中排除的内置模式列表。该列表包括与名为 CVS 的目录相匹配的条目，以及以 `~` 字符结尾的文件，它们可能是备份文件。您通常不想在文件系统操作中包括这类文件和目录，因此排除这些文件是默认行为。然而，如果确实想无例外地选择所有文件和目录，可以将文件集的 `defaultexcludes` 属性设置为 `no`。

使用选择器

正如我们已经看到的，文件集用于指定一组文件，并且这个组的内容可以使用 `include` 和 `exclude` 模式来指定。也可以结合称为 选择器 的特殊元素使用 `include` 和 `exclude` 来选择文件。下面是对 Ant 可用的核心选择器的列表：

- **size:** 这个选择器用于根据文件的字节大小选择文件（除非使用 `units` 属性来指定了不同的单位）。`when` 属性用于设置比较的性质（`less`、`more` 或者 `equal`），`value` 属性定义每个文件将与之作比较的目标大小。
- **contains:** 只有包含给定文本字符串（由 `text` 属性指定）的文件才匹配这个选择器。默认情况下，查找操作是大小写敏感的；添加 `casesensitive="no"` 可以改变默认设置。
- **filename:** `name` 属性指定文件名要与之匹配的模式。它本质上与 `include` 元素相同，以及与指定了 `negate="yes"` 时的 `exclude` 元素相同。
- **present:** 从当前目录结构中选择如下文件：它们与指定的 `targetdir` 目录中的文件具有相同的名称和相对目录结构。
- **depend:** 这个选择器与 `present` 选择器具有相同的效果，只不过匹配的文件被限制到相对于 `targetdir` 位置中的对应文件来说，最近已修改过的那些文件。
- **date:** 这个选择器基于其最后修改日期选择文件。`when` 属性指定作比较的性质是 `before`、`after` 还是 `equal`，`datetime` 属性指定与之作比较的日期和时间，这个日期和时间具有给定的固定格式 `MM/DD/YYYY HH:MM AM_or_PM`。注意 Windows 平台上有一个内置的 2 秒偏移，以允许底层文件系统的不精确性——这可能导致匹配的文件数量超过预期。允许的回旋时间量可以使用 `granularity` 属性来更改（以毫秒为单位来指定）。
- **depth:** 这个选择器检查每个文件的目录结构层数。 `min` 和/或 `max` 属性用于选择具有想要的目录层数目的文件。

还可以通过在一个选择器 容器 内嵌套一个或多个选择器来组合选择器。最常用的选择器容器 `and` 仅选择它包含的所有选择器都选择了的文件。其他选择器容器包括 `or`、`not`、`none` 和 `majority`。

下面是一个文件集的例子，它仅选择那些大于 512 字节并且包含字符串“hello”的文件。

```
<fileset dir="dir">
  <and>
    <contains text="hello"/>
    <size value="512" when="more"/>
  </and>
</fileset>
```

将生成文件链接起来

有两种生成大型项目的不同方法。一种是让一个单一的生成文件做所有事情；另一种是让高级别的生成文件调用其它生成文件以执行特定任务，从而将生成过程划分为许多较小的部分。

使用 `ant` 任务来从一个 `Ant` 生成中调用另一个 `Ant` 生成是很容易的。在简单的情况下，您可以使用 `antfile` 属性，仅指定那些要使用的生成文件，`Ant` 将生成该生成文件中的默认目标。例如：

```
<ant antfile="sub-build.xml"/>
```

在父生成文件中定义的任何属性默认将传递给子生成文件，虽然这可以通过指定

`inheritAll="false"`来避免。通过使用 `property` 元素来传入显式的属性也是可以做到的 —— 即使将 `inheritAll` 设置为 `false`，这些属性也仍然适用于子生成文件。这个功能很适合用于给子生成文件传入参数。

让我们来考虑一个例子。下面是我们想要调用的一个生成文件：

```
<?xml version="1.0"?>
<project default="showMessage">
  <target name="showMessage">
    <echo message="Message=${message}"/>
  </target>
</project>
```


（我们在前面还没有遇到过 **echo** 任务 —— 它简单地输出给定的消息。）

下面是调用第一个生成文件的第二生成文件，它还给第一个生成文件传入 **message** 属性：

```
<?xml version="1.0"?>
<project default="callSub">
  <target name="callSub">
    <ant antfile="sub.xml" target="showMessage" inheritAll="false">
      <property name="message" value="Hello from parent build"/>
    </ant>
  </target>
</project>
```

运行第二个生成文件所得到的输出如下：

```
Buildfile: build.xml

callSub:

showMessage:
    [echo] Message=Hello from parent build

BUILD SUCCESSFUL
Total time: 0 seconds
```

使用 CVS 知识库

CVS 是 **concurrent versions system**（并发版本控制系统）的缩写。它是一个源代码控制系统，设计用于跟踪许多不同开发人员做出的更改。它非常流行，在开放源代码项目中特别受欢迎。Ant 提供了与 CVS 的紧密集成。这对于自动化生成环境是非常有用的，因为单个生成文件也可以从源代码知识库中提取出一个或多个模块，生成项目，甚至基于自从前次执行生成以来所作的变更生成批处理文件。

注意，为了利用 Ant 中的 **cv**s 任务，您需要在机器上安装 **cv**s 命令，并使其从命令行可用。这个命令包括在大多数 **Linux** 发行套件中；它也以多种形式对 **Windows** 可用 —— 例如作为宝贵的 **Cygwin** 环境的一部分。（请参阅参考资料以了解关于 **Cygwin** 的更多信息。）

下面是从 CVS 知识库提取模块的一个例子生成文件:

```
<?xml version="1.0"?>
<project name="CVS Extract" default="extract" basedir=".">
<property name="cvsRoot" value=":pserver:anonymous@dev.eclipse.org:/home/eclipse" />
  <target name="extract">
    <cvs cvsRoot="${cvsRoot}"
      package="org.eclipse.swt.examples"
      dest="${basedir}" />
  </target>
</project>
```

cvs 任务的主要属性是 **cvsRoot**，它是对 CVS 知识库的完整引用，包括连接方法和用户详细信息。这个参数的格式如下:

```
[[:method:]][:[user][:password]@]hostname[:[port]]/path/to/repository
```

在上面的例子中，我们作为匿名用户连接到 **Eclipse** 项目的中央知识库。然后其他属性指定了我们希望提取的模块以及放置提取文件的目的地。提取是 CVS 任务的默认操作；其他操作可通过使用 **command** 属性来指定。

请参阅参考资料以了解关于 CVS 的更多信息。

第七章 使用自定义任务来扩展 **ANT**

简介

正如我们从前述几节中所看到的，**Ant** 非常强大，具有涵盖广泛功能集的许多核心任务。它还有许多这里没有介绍的附加任务，再加上提供广泛附加功能的许多可选任务，以及作为 **Ant-Contrib** 项目的一部分来提供的其他任务；最后，**Apache Ant** 主页上还列出了外部可用的更多任务。面对 **Ant** 提供的所有这些任务，您似乎再也不需要其他任务了，但是 **Ant** 的真正力量在于它的易于扩展性。事实上，恰恰正是这种可扩展性促使人们开发了如此多的附加任务。

可能会存在这样的场合，在那样的场合下创建自定义的任务更为合适。例如，假设您创建了一个命令行工具来执行某个特定操作；这个工具可能是将对 Ant 可用的任务的恰当候选者（当该工具是用 Java 语言编写的时更是这样，虽然该工具不一定是用 Java 语言编写的。）与其让 Ant 使用 `exec` 任务外部地调用该工具（这样将引入依赖关系，并使得生成文件在跨越不同平台时更难于使用），您可以将它直接合并到生成文件中。还可以使得 Ant 的常规文件集和通配符匹配功能对自定义的任务可用。

在本节中，我们将考察一个简单自定义任务的构造过程。这个任务将对文件中的行执行排序操作，并将排序后的行集写到一个新文件中。

创建自定义的任务

为实现一个简单的自定义任务，我们所需做的就是扩展 `org.apache.tools.ant.Task` 类，并重写 `execute()` 方法。因此，作为这个文件排序自定义任务的框架，我们将编写如下代码：

```
import org.apache.tools.ant.BuildException;
import org.apache.tools.ant.Task;

public class FileSorter extends Task {
    // The method executing the task
    public void execute() throws BuildException {}
}
```

注意我们声明 `execute()` 方法抛出一个 `BuildException` 异常。如果这个任务出了任何错误，我们将抛出这个异常以便向 Ant 指出故障。

大多数任务，不管是核心任务还是自定义任务，都利用属性来控制它们的行为。对于这个简单任务，我们需要一个属性来指定要排序的文件，需要另一个属性来指定排序内容的输出。我们把这两个属性分别叫做 `file` 和 `tofile`。

Ant 使得支持自定义任务中的属性非常容易。为此，我们只需实现一个具有特别格式化的名称的方法，Ant 能够使用生成文件中指定的对应属性的值来调用这个方法。这个方法的名称需要是 `set` 加

上属性的名称，因此在这个例子中，我们需要名为 `setFile()` 和 `setToFile()` 的方法。当 Ant 遇到生成文件中的一个属性设置时，它会寻找相关任务中具有适当名称的方法（称为 **setter** 方法）。

生成文件中的属性是作为字符串来指定的，因此我们的 **setter** 方法的参数可以是一个字符串。在这样的情况下，Ant 将在展开值所引用的任何属性之后，使用该属性的字符串值来调用我们的方法。但有时我们想把属性的值看作是一种不同的类型。这里的示例任务就是这种情况，其中的属性值引用文件系统上的文件，而不只是引用任意的字符串。可以通过将方法参数声明为 `java.io.File` 类型来容易地做到这点。Ant 将接受属性的字符串值，并把它解释为一个文件，然后传递给我们的方法。如果文件是使用相对路径名称来指定的，则会被转换为相对于项目基目录的绝对路径。Ant 能够对其他类型执行类似的转换，比如 `boolean` 和 `int` 类型。如果您提供具有相同名称但是具有不同参数的两个方法，Ant 将使用更明确的那一个方法，因此文件类型将优先于字符串类型。

这个自定义任务需要的两个 **setter** 方法类似如下：

```
// The setter for the "file" attribute
public void setFile(File file) {}

// The setter for the "tofile" attribute
public void setToFile(File tofile) {}
```

实现自定义的任务

使用前一小节开发的框架，现在我们可以完成这个简单的文件排序任务的实现：

```
import java.io.*;
import java.util.*;
import org.apache.tools.ant.BuildException;
import org.apache.tools.ant.Task;

/**
 * A simple example task to sort a file
 */
public class FileSorter extends Task {
    private File file, tofile;

    // The method executing the task
```

```
public void execute() throws BuildException {
    System.out.println("Sorting file="+file);
    try {
        BufferedReader from =
            new BufferedReader(new FileReader(file));
        BufferedWriter to =
            new BufferedWriter(new FileWriter(tofile));
        List allLines = new ArrayList();
        // read in the input file
        String line = from.readLine();
        while (line != null) {
            allLines.add(line);
            line = from.readLine();
        }
        from.close();
        // sort the list
        Collections.sort(allLines);
        // write out the sorted list
        for (ListIterator i=allLines.listIterator(); i.hasNext(); ) {
            String s = (String)i.next();
            to.write(s); to.newLine();
        }
        to.close();
    } catch (FileNotFoundException e) {
        throw new BuildException(e);
    } catch (IOException e) {
        throw new BuildException(e);
    }
}

// The setter for the "file" attribute
public void setFile(File file) {
    this.file = file;
}

// The setter for the "tofile" attribute
public void setTofile(File tofile) {
    this.tofile = tofile;
}
}
```

两个 **setter** 方法简单地对属性的值排序，以便这些值能够在 **execute()** 方法中使用。这里，输入文件被逐行地读入一个列表中，然后被排序并逐行地输出到输出文件。注意，为简单起见，我们很

少执行错误检查——例如，我们甚至没有检查生成文件是否设置了必需的属性。不过我们的确至少捕捉了所执行的操作抛出的 I/O 异常，并将这些异常作为 `BuildExceptions` 重新抛出。

现在可以使用 `javac` 编译器或从某个 IDE 内编译这个自定义的任务。为了解决所使用的 Ant 类的引用问题，您需要把 `ant.jar` 文件的位置添加到 `classpath` 中。这个文件应该在 Ant 安装路径下的 `lib` 目录。

使用自定义的任务

现在我们已经开发和编译了这个自定义的任务，下面可以从生成文件中利用它了。

在能够调用自定义的任务之前，我们需要给它指定一个名称来定义它，并告诉 Ant 关于实现这个任务的类文件的信息，以及定位该类文件所必需的任何 `classpath` 设置。这是使用 `taskdef` 任务来完成的，如下所示：

```
<taskdef name="filesorter"
  classname="FileSorter"
  classpath="." />
```

大功告成！现在可以像使用 Ant 的核心任务一样使用这个自定义的任务了。下面是一个完整的生成文件，它显示了这个自定义任务的定义和用法：

```
<?xml version="1.0"?>
<project name="CustomTaskExample" default="main" basedir=".">
  <taskdef name="filesorter"
    classname="FileSorter"
    classpath="." />
  <target name="main">
    <filesorter file="input.txt" tofile="output.txt" />
  </target>
</project>
```

现在在当前工作目录中创建一个 `input.txt` 文件来测试这个自定义的任务。例如：

```
Hello there
This is a line
```

```
And here is another one
```

下面是运行上面的生成文件之后产生的控制台输出：

```
Buildfile: build.xml

main:
[filesorter] Sorting file=E:\tutorial\custom\input.txt

BUILD SUCCESSFUL
Total time: 0 seconds
```

注意 `input.txt` 的相对路径名称被转换成了当前目录中的一个绝对路径名称。这是因为我们将 `setter` 方法的参数指定为 `java.io.File` 类型而不是 `java.lang.String` 类型。

现在看一下这个任务实际是否能工作。这时应该已经在同一目录中创建了名为 `output.txt` 的文件，它包含以下内容：

```
And here is another one
Hello there
This is a line
```

您可以尝试指定一个不存在的输入文件，以确定该任务是如何向 Ant 报告 “file not found” 异常的。

祝贺您：您现在已经开发和使用了一个自定义的 Ant 任务！创建更复杂的任务还会涉及其他许多方面，参考资料包含了指向此主题的进一步信息源的链接。

第八章 结束语和参考资料

结束语

我们希望您会发现这次 Ant 之旅很有帮助。同时，Ant 的目标是保持尽可能简单，它通过大量的任务提供了大量的功能，每个任务都有许多选项，有时这可能是令人难以招架的。我们无法在单个教程中探索 Ant 的所有功能，但愿我们已经介绍了所有的基本概念和足够的基本功能，以使您步入开始在现实项目中使 Ant 的轨道。下面总结一下本教程介绍过的内容：

- Ant 生成文件是如何构造的
- 如何从命令行以及从 Eclipse 内运行 Ant
- 如果通过编译源代码、创建 JAR 文件以及时间戳文件（以识别每次生成过程的输出）来生成简单的 Java 项目
- 如何在 Ant 中执行基本的文件系统操作
- 模式匹配和选择器的基本概念，再加上如何从一个生成文件调用另一个生成文件，以及如何执行 CVS 操作
- 如何通过编写 Java 类来扩展 Ant 的标准功能

要继续 Ant 之旅，请访问下一小节中的某些链接，同时熟悉 Ant 手册中的内容，该手册是编写 Ant 生成文件时的权威参考源。祝您好运！

参考资料

- 访问 [Apache Ant 主页](#)。这是一个极好的信息源，它包括下载、Ant 手册以及指向其他参考资料的链接。
- 查找关于 Ant 的 [IDE 和编辑器集成](#) 的更多信息。
- 了解关于 [Eclipse](#) 这个通用工具平台的更多信息。
- [developerWorks 开放源代码项目专区](#) 包含丰富的基于 Eclipse 的内容。
- 如果您是 XML 新手，请访问 [developerWorksXML 专区](#)。
- [Cygwin](#) 是一个用于 Windows 的类 Linux 环境。
- [CVS](#) 版本控制的开放标准。
- 了解关于 [WebSphere Studio Application Developer](#) 的更多信息，它基于 Eclipse 平台。
- 查找关于 WebSphere Application ServerAnt 对 [Ant 任务](#) 的支持的更多信息。
- 在 “[JAR files revealed](#)” 中探索 JAR 文件格式的强大能力（developerWorks, 2003 年 10 月）。
- developerWorks 特别贡献了许多关于 Ant 的文章：
 - “[Incremental development with Ant and JUnit](#)”，Malcolm Davis 撰稿，（2000 年 11 月）。
 - “[Extending Ant to support interactive builds](#)”，Anthony Young-Garner 撰稿，（2001 年 11 月）。
 - “[Enhance Ant with XSL transformations](#)”，Jim Creasman 撰稿（2003 年 9 月）。
- Ant 还支持开放源代码的 [Jikes 编译器](#) 和 [GNU Compiler for Java](#)（GCJ）。
- 在 [developerWorks Java 技术专区](#) 可以找到关于 Java 编程的方方面面的数百篇文章。
- 还可以参阅 [Java 技术专区教程主页](#)，从 developerWorks 获得免费的 Java 相关教程的完整列表。

[完]