ApacheAnt教程

极客学院出版

前言

Apache Ant 是由 Java 语言开发的工具，由 Apache 软件基金会所提供。Apache Ant 的配置文件写成 XML 容易维护和书写，而且结构很清晰。

本教程将以简单的方式会向你展示如何利用 Apache ANT 来自动地构建和部署过程。在完成本教程的学习以后，你将会发现你已经具备下一阶段学习 Apache Ant 中等水平的专业知识 。

适用人群

本教程有助于帮助初学者理解 Apache Ant 的基础知识，并用它来自动地构建和部署的过程。

学习前提

对于学习这个教程，我们假设读者已经具备了 Java 或者其他开发语言关于软件开发的基础知识。如果你已经接触过一些软件构建和部署过程，那么对于学习这个教程将会有一定的帮助。

目录

1

介绍

Ant 是一个 Apache 基金会下的跨平台的基于 Java 语言开发的构件工具。在我们详细了解 Apache Ant 之前，让我们来讲解为什么构建工具是需要最先了解的。

构建工具的需求

一般情况，开发人员花费大量的时间做一般性的任务，比如：构建和部署，通常包含下面的工作：

- 编译代码
- 封装二进制文件
- 在测试服务器上部署二进制文件
- 测试改变
- 从一个地点拷贝代码到另一个地点

为了自动和简化上面的工作，Apache Ant 是非常有用的。这是一个基于开放的操作系统构建和部署的工具，该工具需要从命令行执行。

Apache Ant 的历史

- Ant 是由 James Duncan Davidson 开发的（也就是 Tomcat 最初的开发者）。
- 最初是用来构建 Tomcat，被作为一个 Tomcat 的发行版的一部分。
- Apach Make 工具包的复杂性与诸多问题催生了 Apache Ant。
- 在 2000 年的时候，Ant 被作为一个独立的项目。最新的 Apache Ant 版本是 2014 年 5 月的 1.9.4 版本。

Apache Ant 的特点

- Ant 是最完整的基于 Java 语言开发的构建和部署工具。
- Ant 具有平台无关性，可以处理平台特有的属性，诸如文件分隔符。
- Ant 还可以用来执行平台特有的任务，比如使用 touch 命令修改一个文件的修改时间。
- Ant 脚本是用 XML 来完成的。如果你已经对 XML 有所了解，那么你学习 Ant，将会比较得心应手。
- Ant 擅长自动完成重复任务。
- Ant 开始需要一系列的预先定义好的任务。
- Ant 提供了开发自定义任务的接口。

- Ant 可以很容易从命令行调用，并且它能够很好地集成免费和商用的集成开发环境。

2

环境搭建

Apache Ant 是 Apache 软件许可证下发布的。Apache 是一个由一个开源机构认证的完全成熟的开源许可证。Apache Ant 最新的版本包括了以下的部分：完整的源代码， 类文件和文档，这些内容都可以在 http://ant.apache.org上面找到。

安装 Apache Ant

安装 Apache Ant 的前提是你的电脑上已经下载并且安装了 Java 开发工具包（JDK）。如果电脑上没有安装的话，请按照下列的 提示 (http://wiki.jikexueyuan.com/project/java/setup.html) 进行下载和安装。

- 确保设置 JAVA_HOME 环境变量为你安装 Java 开发工具包的文件夹。
- 从 http://ant.apache.org 下载库。
- 将文件解压到一个方便的地址，如 c:\folder 。可以使用 Winzip, winRAR, 7-zip 或者其他类似的工具进行解压缩操作。
- 创建一个新的环境变量，命名为 ANT_HOME， 该环境变量指向 Ant 的安装文件夹，在这个例子中，该文件夹为 c:\apache-ant-1.8.2-bin。
- 将 Apache Ant 的批处理文件的路径添加到 PATH 环境变量里。在这个例子中，该路径应为 c:\apache-ant-1.8.2-bin\bin 文件夹。

验证 Apache Ant 安装

为了验证你的电脑上是否已经成功安装了 Apache Ant ，你可以在命令提示符中输入 ant。你应该会看到一个与下列相似的输出：

```
> C:\>ant -version
Apache Ant(TM) version 1.8.2 compiled on December 20 2010
```

如果你没有看到与上述相类似的输出，请重新检查一下你是否已经正确按照之前的步骤进行了安装。

安装 Eclipse

这个教程也包括了 Ant 和 Eclipse 继承开发环境的整合。因此，如果你还没有安装好 Eclipse， 请下载并安装 Eclipse。

安装 Eclipse 请按照下述步骤：

- 从 www.eclipse.org 上面下载最新版的 Eclipse 文件。

- 解压 Eclipse 文件到一个方便的位置，比如 c:\folder 。
- 从 c:\eclipse\eclipse.exe 处运行 Eclipse 。

构建文件

一般来说，Ant 的构建文件默认为 build.xml，放在项目顶层目录中。然而，并没有限制构建文件必须命名为 build.xml，也并不限制放在项目顶层目录中。你可以将构建文件命名为其他名字，也可以将它放在项目的其他地方。

这个教程将以简单的方式向你展示如何利用 Apache Ant 来自动地构建和部署项目的过程。在完成本教程的学习以后，你将会发现你已经具备下一阶段学习 Apache Ant 中等水平的专业知识 。

对于下面的练习，创建一个文件命名为 build.xml 的文件，存储在你电脑的任意地方 ，并包含一下的内容：

```
<?xml version="1.0"?>
  <project name="Hello World Project" default="info">

  <target name="info">
    <echo>Hello World - Welcome to Apache Ant!</echo>
  </target>

</project>
```

注意到上面的练习中，在 xml 文件的声明前面没有任何空行或者空格。如果你在写 xml 文件的声明时加入了空行或者空格，执行 ant -build 操作时，将会出现下面的错误信息：

```
The processing instruction target matching "[xX][mM][lL]" is not allowed.
```

错误信息的意思是：处理指令目标匹配 `"[xX][mM][lL]"` 不被允许。所有的构建文件需要包含项目元素 (project 标签) 和至少一个目标元素 (target 标签)。

构建文件的项目元素 有 3 个属性：

属性	描述
项目名 (name)	表示项目的名称。（ 可选 ）
默认 (default)	表示构建脚本默认运行的目标，即制定默认的 target。一个项目 (project) 可以包含多个目标 (target)。（ 必须 ）
基准目录 (base dir)	表示当该属性没有指定时，使用 Ant 的构件文件的附目录作为基准目录。（ 可选 ）

一个目标 (target) 是一系列你想运行的任务 (tasks)，运行时看成一个单元。在我们的例子中，我们用一个简单的目标来为用户提供一个有信息的消息。

目标和目标之间可以有依赖关系。举个例子，一个部署 (deploy) 目标可能依赖于封装 (package) 目标，而这个封装目标可能又依赖于编译 (compile) 目标等。依赖关系被表示成依赖属性 (depends)。例如：

```
<target name="deploy" depends="package">
  ....
</target>
```

```
<target name="package" depends="clean,compile">
  ....
</target>

<target name="clean" >
  ....
</target>

<target name="compile" >
  ....
</target>
```

构建文件的目标元素有以下属性：

属性	描述
目标名 (name)	表示目标的名称。（ 必须 ）
依赖 (depends)	用于描述 target 之间的依赖关系，若与多个 target 存在依赖关系时，需要以“,”间隔。Ant 会依照 depends 属性中 target 出现的顺序依次执行每个 target。被依赖的 target 会先执行。（ 可选 ）
描述 (description)	关于 target 功能的简单描述。（ 可选 ）
如果 (if)	用于验证指定的属性是否存在，若不存在，所在 target 将不会被执行。（ 可选 ）
除非 (unless)	该属性的功能与 if 属性的功能正好相反，它也用于验证指定的属性是否存在，若不存在，所在 target 将会被执行。（ 可选 ）

在上面的例子中 echo 任务主要负责打印消息。在我们的例子中，执行 echo 任务后，打印出 “hello world” 消息。

为了运行 ant 的构建文件，打开命令提示符并导航到 build.xml 建立的文件夹。输入 ant info 命令或者 ant 命令。这两种命令都可以运行，因为 info 是构建文件的默认目标。你讲会看到下面的输出信息：

```
C:\>ant
Buildfile: C:\build.xml

info: [echo] Hello World - Welcome to Apache Ant!

BUILD SUCCESSFUL
Total time: 0 seconds

C:\>
```

4

属性任务

Ant 构建文件是用 XML 编写的，它不能像你喜欢的编程语言那样去声明变量。然而，正如你可能已经想到的，如果允许 Ant 声明变量，如项目名称，项目源目录等，这将是非常有用的。

Ant 使用**属性 (property)** 元素来让你能够具体说明属性。这就允许这些属性能够在不同的构建和不同的环境下发生改变。

默认情况下，Ant 提供以下预定义的属性，这些属性都是可以在构建文件中使用的：

属性	解释
ant.file	该构建文件的完整地址
ant.version	安装的 Apache Ant 的版本
basedir	构建文件的基目录的绝对路径，作为 **project** 元素的 **basedir** 属性
ant.java.version	Ant 使用的 JAVA 语言的软件开发工具包的版本
ant.project.name	项目的名字，具体声明为 **project** 元素的 **name** 属性
ant.project.default-target	当前项目的默认目标
ant.project.invoked-targets	在当前项目中被调用的目标的逗号分隔列表
ant.core.lib	Ant 的 jar 文件的完整的地址
ant.home	Ant 安装的主目录
ant.library.dir	Ant 库文件的主目录，特别是 ANT_HOME/lib 文件夹

Ant 也确保系统属性在构建文件中可用，如 file.separator。

除了上述内容以外，用户也可以使用 property 元素定义一些额外的属性。下面的例子就演示了怎样去定义一个叫做 sitename 的属性：

```
<?xml version="1.0"?>
<project name="Hello World Project" default="info">

  <property name="sitename" value="www.tutorialspoint.com"/>
  <target name="info">
    <echo>Apache Ant version is ${ant.version} - You are at ${sitename} </echo>
  </target>

</project>
```

在上述的构建文件下运行 Ant 可以产生以下输出：

```
C:\>ant
Buildfile: C:\build.xml

info: [echo] Apache Ant version is Apache Ant(TM) version 1.8.2
     compiled on December 20 2010 - You are at www.tutorialspoint.com
```

```
BUILD SUCCESSFUL
Total time: 0 seconds

C:\>
```

5

属性文件

当你只需要对小部分属性进行设置时，可以选择直接在构建文件中设置。然而，对于大项目，最好将设置属性的信息存储在一个独立的文件中。

存储属性信息在一个独立的文件中将会提供以下好处：

- 它可以让您重复使用相同的构建文件，该文件在不同的执行环境中使用不同的属性设置。例如，构建属性文件在 DEV , TEST , 和 PROD 环境中可以独立地被维护。
- 当你事先不知道属性的值时（ 例如，在一个实际的环境中 ），这样处理是有益的。这样允许你在知道属性值后，在其他环境中执行生成 (build) 操作。

这里没有硬性规定，但是一般情况下，属性文件都被命名为 build.properties， 并且与 build.xml 存放在同一目录层。 你可以基于部署环境 ——比如： build.properties.dev 和 build.properties.test 创建多个 build.properties 文件。

在下面的例子中展示了 build.xml 文件和与之相联系的 build.properties文件：

build.xml

```
<?xml version="1.0"?>
<project name="Hello World Project" default="info">

  <property file="build.properties"/>

  <target name="info">
    <echo>Apache Ant version is ${ant.version} - You are at ${sitename} </echo>
  </target>

</project>
```

build.properties

```
 # The Site Name
sitename=wiki.jikexueyuan.com
buildversion=3.3.2
```

注意到上面的练习中，sitename 是一个自定义属性，执行后映射到一个地址为 “wiki.jikexueyuan.com” 的网站上。你可以用这种方式声明任意数量的属性。在上面的例子中，还有一个自定义属性 buildversioin，它表明了当前构建的版本号。

除了以上提到的两个属性， Ant 还提供了其他内置属性，在前一章节中已经提到，但是下面我们再一次给出相关属性。

属性	描述
ant.file	表示 buildfile 的绝对路径。
ant.version	表示 Ant 的版本。
basedir	表示 project 基目录的绝对路径。
ant.jave.version	表示 Ant 检测到的 JDK 的版本。
ant.project.name	表示当前指定的 project 的名字。
ant.project.default-target	表示当前项目的默认目标。
ant.project.invoked-targets	表示被当前项目调用的一系列用逗号分隔开的目标。
ant.core.lib	表示 Ant jar 文件的绝对路径。
ant.home	表示 Ant 安装的根目录。
ant.library.dir	表示 Ant 函数库，一般情况下为 ANT_HOME/lib 文件的根目录。

在这一章节的例子中，我们用到的 Ant 内置属性是 ant.version 属性。

6

数据类型

Ant 提供一些预定义的数据类型。不要将术语“数据类型”和那些在编程语言中可用的数据类型相混淆，而是将他们视作一组已经在产品中配置好的服务。

下述的数据类型是由 Apache Ant 提供的。

文件集

文件集的数据类型代表了一个文件集合。它被当作一个过滤器，用来包括或移除匹配某种模式的文件。

例如，参考下面的代码。这里，src 属性指向项目的源文件夹。

文件集选择源文件夹中所有的 .java 文件，除了那些包含有 'Stub' 单词的文件。能区分大小写的过滤器被应用到文件集上，这意味着名为 Samplestub.java 的文件将不会被排除在文件集之外。

```
<fileset dir="${src}" casesensitive="yes">
  <include name="**/*.java"/>
  <exclude name="**/*Stub*"/>
</fileset>
```

模式集合

一个模式集合指的是一种模式，基于这种模式，能够很容易地过滤文件或者文件夹。模式可以使用下述的元字符进行创建。

- ？– 仅匹配一个字符
- * – 匹配零个或者多个字符
- ** – 递归地匹配零个或者多个目录

下面的例子演示了模式集合的使用。

```
<patternset id="java.files.without.stubs">
  <include name="src/**/*.java"/>
  <exclude name="src/**/*Stub*"/>
</patternset>
```

该模式集合能够通过一个类似于下述的文件集进行重用：

```
<filelist id="config.files" dir="${webapp.src.folder}">
  <file name="applicationConfig.xml"/>
  <file name="faces-config.xml"/>
  <file name="web.xml"/>
```

```
  <file name="portlet.xml"/>
</filelist>
```

文件列表

文件列表数据类型与文件集相类似，除了以下几处不同：

- 文件列表包含明确命名的文件的列表，同时其不支持通配符。
- 文件列表数据类型能够被应用于现有的或者还不存在的文件中。

让我们来看一个下述的关于文件列表数据类型的例子。在这个例子中，属性 webapp.src.folder 指向该项目中的 Web 应用的源文件夹。

```
<fileset dir="${src}" casesensitive="yes">
  <patternset refid="java.files.without.stubs"/>
</fileset>
```

过滤器集合

使用一个过滤器集合数据类型与拷贝任务，你可以在所有文件中使用一个替换值来替换掉一些与模式相匹配的文本。

一个常见的例子就是对一个已经发行的说明文件追加版本号，代码如下：

```
<copy todir="${output.dir}">
  <fileset dir="${releasenotes.dir}" includes="**/*.txt"/>
  <filterset>
    <filter token="VERSION" value="${current.version}"/>
  </filterset>
</copy>
```

在这段代码中：

- 属性 output.dir 指向项目的输出文件夹。
- 属性 releasenotes.dir 指向项目的发行说明文件夹。
- 属性 current.version 指向项目的当前版本文件夹。
- 拷贝任务，顾名思义，是用来将文件从一个地址拷贝到另一个地址。

路径

path 数据类型通常被用来表示一个类路径。各个路径之间用分号或者冒号隔开。然而，这些字符在运行时被替代为执行系统的路径分隔符。

类路径被设置为项目中 jar 文件和类文件的列表，如下面例子所示：

```
<path id="build.classpath.jar">
  <pathelement path="${env.J2EE_HOME}/${j2ee.jar}"/>
  <fileset dir="lib">
    <include name="**/*.jar"/>
  </fileset>
</path>
```

在这段代码中：

- 属性 env.J2EE_HOME 指向环境变量 J2EE_HOME 。
- 属性 j2ee.jar 指向在 J2EE 基础文件夹下面的名为 J2EE jar 的文件。

7

构建项目

现在我们已经学习了 Ant 的数据类型，是时候在实际过程中运用所学知识了。在这一章节中，我们将会构建一个项目。这一章节的目的是创建一个 Ant build 文件，该文件能够编译 Java 源文件和将这些类文件存储在 WEB-INF\classes 文件夹下。

考虑接下来构建项目的结构：

- 数据脚本存储在 db 文件夹中。
- java 源文件存储在 src 文件夹中。
- images (图像)，js (JavaScript 脚本)，style (css 层叠样式表)存储在 war 文件夹中。
- JSPs 文件存储在 jsp 文件夹中。
- 第三方的 jar 文件存储在 lib 文件夹中。
- java 类文件存储在 WEB-INF\classes 文件夹中。

学习完本教程的剩余部分后，就能知道这个项目是一个 Hello World 传真应用。

```
C:\work\FaxWebApplication>tree
Folder PATH listing
Volume serial number is 00740061 EC1C:ADB1
C:.
+---db
+---src
.  +---faxapp
.  +---dao
.  +---entity
.  +---util
.  +---web
+---war
   +---images
   +---js
   +---META-INF
   +---styles
   +---WEB-INF
      +---classes
      +---jsp
      +---lib
```

下面给出上述项目的 build.xml 文件的内容。让我们来一条语句接一条语句地来分析它。

```
<?xml version="1.0"?>
<project name="fax" basedir="." default="build">
   <property name="src.dir" value="src"/>
   <property name="web.dir" value="war"/>
```

```
  <property name="build.dir" value="${web.dir}/WEB-INF/classes"/>
  <property name="name" value="fax"/>

  <path id="master-classpath">
    <fileset dir="${web.dir}/WEB-INF/lib">
      <include name="*.jar"/>
    </fileset>
    <pathelement path="${build.dir}"/>
  </path>

  <target name="build" description="Compile source tree java files">
    <mkdir dir="${build.dir}"/>
    <javac destdir="${build.dir}" source="1.5" target="1.5">
      <src path="${src.dir}"/>
      <classpath refid="master-classpath"/>
    </javac>
  </target>

  <target name="clean" description="Clean output directories">
    <delete>
      <fileset dir="${build.dir}">
        <include name="**/*.class"/>
      </fileset>
    </delete>
  </target>
</project>
```

首先，让我们来声明一些源文件，web 文件和构建文件的一些属性信息。

```
<property name="src.dir" value="src"/>
<property name="web.dir" value="war"/>
<property name="build.dir" value="${web.dir}/WEB-INF/classes"/>
```

在上面的例子中：

- src.dir 表示这个项目的源文件目录，也就是存储 java 文件的地方。
- web.dir 表示这个项目的 web 文件目录，也就是存储 JSPs 文件，web.xml，css，javascript 以及其它与 web 相关的文件的地方。
- build.dir 表示该项目的输出文件。

属性也可以引用其它属性。在上面的例子中，build.dir 属性引用了 web.dir 属性。

在上面的例子中，src.dir 就是项目源文件存放的地方。

我们项目的默认目标是编译目标。但是首先让我们来看一下 clean 目标。

clean 目标，就像它的名字所表明的意思一样，删除构建文件夹中的所有文件。

```
<target name="clean" description="Clean output directories">
  <delete>
    <fileset dir="${build.dir}">
      <include name="**/*.class"/>
    </fileset>
  </delete>
</target>
```

控制类路径 (master-classpath) 保存类路径的相关信息。在这种情况下，它包含了构建文件夹和 jar 文件夹中的所有的类文件。

```
<path id="master-classpath">
  <fileset dir="${web.dir}/WEB-INF/lib">
    <include name="*.jar"/>
  </fileset>
  <pathelement path="${build.dir}"/>
</path>
```

最后，构建目标构建这些文件。首先，我们创建一个构建目录，如果该目录不存在，我们就执行 javac 命令（具体以 jdk 1.5 作为我们目标的编译环境）。 我们对 javac 任务提供源文件夹和类路径，并且通过执行 javac 任务将类文件存放在构建文件夹中。

```
<target name="build" description="Compile main source tree java files">
  <mkdir dir="${build.dir}"/>
  <javac destdir="${build.dir}" source="1.5" target="1.5" debug="true"
    deprecation="false" optimize="false" failonerror="true">
    <src path="${src.dir}"/>
    <classpath refid="master-classpath"/>
  </javac>
</target>
```

在这个文件上执行 Ant，编译 java 源文件，并将编译后的类文件存放在构建文件夹的地方。

运行 Ant 文件后，能看到以下输出：

```
C:\>ant
Buildfile: C:\build.xml

BUILD SUCCESSFUL
Total time: 6.3 seconds
```

文件被编译后，将存储在 build.dir 文件夹中。

8

生成文档

文档在任何项目中都是必须的。文档对一个项目的维护起了至关重要的作用。 通过使用内置的 Javadoc 工具，使用 Java 生成文档变得更加容易。Ant 通过按需生成文档使得这个步骤甚至变得更简单。

如你所知，javadoc 工具具有高度的灵活性，而且其还允许进行一些配置。Ant 通过使用 javadoc 任务的方式来公开这些配置选项。如果你对 javadoc 不熟悉的话，我们建议你先看一下 Java 文档教程 (http://wiki.jikexueyuan.com/project/java) 。

下述的章节列出了在 Ant 中最常使用的 javadoc 的选项。

属性

源包括源路径，源路径引用或者源文件三个属性。

- 源路径 (sourcepath) 指向源文件所在的文件夹，例如： src 文件夹。
- 源路径引用 (sourcepathref) 指向由该路径属性引用的路径，例如：delegates.src.dir 。
- 源文件 (sourcefiles) 在你想指定单独的文件时使用，比如指定一个逗号分隔列表。

目标路径是通过使用 destdir 文件夹来指定的，例如 build.dir 。

你能够通过指定应被包括的包的名字来过滤 javadoc 任务。这可以通过使用 packagenames 属性实现，即一个以逗号分隔的包文件列表。

你可以过滤 javadoc 过程以只显示公有的，私有的，包或者被保护的类和成员。这些可以通过使用 private，public，package 和 protected 属性实现。

你也可以通过使用相应的属性来告诉 javadoc 任务去包含作者和版本信息。

你也可以使用 group 属性将所有的包组织在一起，以使得他们更易被操作。

将上述内容集中到一起

让我们继续我们的主题，Hello world 传真应用程序。让我们给我们的传真应用项目添加一个文档目标。

下面给出的例子是我们在项目中使用的 javadoc 任务。在这个例子中，我们指定 javadoc 去使用 src.dir 作为源目录，doc 作为目标。

我们还定制窗口标题，标题，以及显示在 java 文档页上的页脚信息。

此外，我们还创建了三个组：

- 为源文件夹中的实用工具类创建了一个组。
- 为用户接口的类创建了一个组。
- 为数据库相关的类创建了一个组。

您可能会注意到，数据包组含有两个包 -- faxapp.entity 和 faxapp.dao 。

```
<target name = "generate-javadoc">
  <javadoc packagenames="faxapp.*" sourcepath="${src.dir}"
    destdir = "doc" version = "true" windowtitle = "Fax Application">

    <doctitle><![CDATA[= Fax Application =]]></doctitle>

    <bottom>
      <![CDATA[Copyright © 2011. All Rights Reserved.]]>
    </bottom>

    <group title = "util packages" packages = "faxapp.util.*"/>
    <group title = "web packages" packages = "faxapp.web.*"/>
    <group title = "data packages" packages = "faxapp.entity.*:faxapp.dao.*"/>
  </javadoc>

  <echo message = "java doc has been generated!" />
</target>
```

让我们运行 javadoc Ant 任务。它将生成 java 文档文件，并将这些文件放置于 doc 文件夹中。

当执行 javadoc 目标时，其产生以下的输出：

```
>C:\>ant generate-javadoc
>Buildfile: C:\build.xml

>java doc has been generated!

>BUILD SUCCESSFUL
>Total time: 10.63 second
```

java 文档文件现在出现在 doc 文件夹中。

通常情况下，javadoc 文件作为发行版或者包目标的一部分。

9

生成 JAR 文件

编译完你的 java 源文件后，接下来就构建 java 存档，例如： JAR 文件。创建 Ant 中的 JAR 文件十分简单，运用 jar 任务来生成 jar 包。在 jar 任务中常用的属性如下所示：

属性	描述
basedir	表示输出 JAR 文件的基目录。默认情况下，为项目的基目录。
compress	表示告知 Ant 对于创建的 JAR 文件进行压缩。
keepcompression	表示 project 基目录的绝对路径。
destfile	表示输出 JAR 文件的名字。
duplicate	表示发现重复文件时 Ant 执行的操作。可以是添加、保存、或者是使该重复文件失效。
excludes	表示移除的文件列表，列表中使用逗号分隔多个文件。
excludesfile	与上同，但是使用模式匹配的方式排除文件。
inlcudes	与 excludes 正好相反。
includesfile	表示在被归档的文件模式下，打包文件中已有的文件。与 excludesfile 相反。
update	表示告知 Ant 重写已经建立的 JAR 文件。

继续我们的 Hello World 传真应用项目，通过添加一个新的目标 target 来产生 jar 文件。 但是在此之前，让我们先来考虑下面给出的 jar 任务。

```
<jar destfile = "${web.dir}/lib/util.jar"
  basedir = "${build.dir}/classes"
  includes = "faxapp/util/**"
  excludes = "**/Test.class" />
```

这里，web.dir 属性指出了 web 源文件的路径。在我们的案例中， web 源文件路径也就是存放 util.jar 的地方。

在我们的案例中，build.dir 属性指出了配置文件夹的存储路径，也就是存放 util.jar 类文件的地方。

在上面的代码中，我们利用来自 faxapp.util 包中的类文件创建了一个名为 util.jar 的 jar 包。然而，我们排除名字为 Test 的类文件。输出的 jar 文件将会存放在 web 应用的配置文件 lib 中。

如果我们想 util.jar 成为可执行文件，只需在 Main-Class 元属性中加入manifest.

这样，上面给出的代码，在加入 Main-Class 元属性后，可以更新为如下形式：

```
<jar destfile = "${web.dir}/lib/util.jar"
  basedir = "${build.dir}/classes"
  includes = "faxapp/util/**"
  excludes = "**/Test.class">

  <manifest>
    <attribute name = "Main-Class" value = "com.tutorialspoint.util.FaxUtil"/>
  </manifest>
```

```
</jar>
```

为了执行 jar 任务，将它包装在目标 target 中，最常见的情况是，将 jar 任务包装在配置目标或者打包目标中（build 目标或 package 目标），并执行包装后的目标。

```
<target name="build-jar">
  <jar destfile="${web.dir}/lib/util.jar"
    basedir="${build.dir}/classes"
    includes="faxapp/util/**"
    excludes="**/Test.class">

    <manifest>
      <attribute name="Main-Class" value="com.tutorialspoint.util.FaxUtil"/>
    </manifest>

  </jar>
</target>
```

在上述文件上运行 Ant ，就能创建出 util.jar。

上述文件运行 Ant 后，得到以下的输出：

```
C:\>ant build-jar
Buildfile: C:\build.xml

BUILD SUCCESSFUL
Total time: 1.3 seconds
```

最后得到的输出 util.jar 将被存储在输出文件夹中。

10

生成 WAR 文件

使用 Ant 创建 WAR 文件是极其简单的。这与创建 JAR 文件任务非常类似。 毕竟，WAR 文件与 JAR 文件只是两种不同的 ZIP 文件。

WAR 任务是 JAR 任务的一个扩展，但是其对控制哪些文件进入 WEB-INF/classes 文件夹和生成 web.xml 文件进行了一些很好的补充。WAR 任务对指定 WAR 文件布局是非常有用的。

既然 WAR 任务是 JAR 任务的一个扩展，JAR 任务的所有的属性都适用于 WAR 任务。

属性	描述
webxml	web.xml 文件的路径
lib	指定什么文件可以进入 WEB-INF\lib 文件夹的一个组
classes	指定什么文件可以进入 WEB-INF\classes 文件夹的一个组
metainf	指定生成 MANIFEST.MF 文件的指令

继续我们的 Hello World 传真应用项目，让我们添加一个新的目标来生成 jar 文件。 但是在此之前，我们需要考虑一下 war 任务。 请看下面的例子：

```
<war destfile = "fax.war" webxml = "${web.dir}/web.xml">

  <fileset dir = "${web.dir}/WebContent">
    <include name = "**/*.*"/>
  </fileset>

  <lib dir = "thirdpartyjars">
    <exclude name = "portlet.jar"/>
  </lib>

  <classes dir = "${build.dir}/web"/>

</war>
```

按照前面的例子中，web.dir 变量指向源 web 文件夹，即该文件包含 JSP，css 和 javascript 文件等等。

该 build.dir 变量指向输出文件夹，WAR 的包能在该文件夹下找到。 通常情况下， 类将被绑定到 WAR 文件下的 WEB-INF/classes 文件夹下。

在这个例子中，我们创建了一个名为 fax.war 的 war 文件。WEB.XML 文件可以从 web 源文件中获取。 所有 web 下来自 “WebContent” 的文件都被复制到 WAR 文件中。

WEB-INF/lib 文件夹中存储了来自于第三方 jar 文件夹中的 jar 文件。但是，我们排除了 portlet.jar，因为该 jar 文件已经存在于应用服务器的 lib 文件夹中了。最后，我们从一个构建目录下的 web 文件夹中复制所有的类，并将复制的类全部放入 WEB-INF/classes 文件夹下。

将一个 war 任务封装到一个 Ant 任务中并运行它。 这将在指定位置创建一个 WAR 文件。

类，库，metainf 和 webinf 目录完全可以进行嵌套以使得他们都能存在于项目结构下分散的文件夹中。 但是最佳的实践建议是，你的 web 项目的 web 内容架构应该与 WAR 文件类似。 传真应用项目的架构就是使用了这个基本原理。

要执行 war 任务，将其封装在一个目标里面，最常见的是，构建目标或者是包目标，然后运行它们。

```
<target name="build-war">

  <war destfile="fax.war" webxml="${web.dir}/web.xml">
    <fileset dir="${web.dir}/WebContent">
      <include name="**/*.*"/>
    </fileset>

    <lib dir="thirdpartyjars">
      <exclude name="portlet.jar"/>
    </lib>

    <classes dir="${build.dir}/web"/>
  </war>

</target>
```

在这个文件上运行 Ant 会替我们创建 fax.war 文件。

下述的输出就是运行 Ant 文件的结果：

```
>C:\>ant build-war
>Buildfile: C:\build.xml

>BUILD SUCCESSFUL
>Total time: 12.3 seconds
```

该 fax.war 文件当前被放置在输出文件夹中。war 文件的内容如下所示：

```
>fax.war:
  >+---jsp   ： 这个文件夹包含了 jsp 文件
  >+---css： 这个文件夹包含了 stylesheet 文件
  >+---js： 这个文件夹包含了 javascript 文件
  >+---images：这个文件夹包含了 image 文件
  >+---META-INF：这个文件夹包含了 Manifest.Mf
  >+---WEB-INF
        >+---classes： 这个文件夹包含了编译好的类
```

```
>+---lib： 第三方库和使用程序 jar 文件
>WEB.xml： 定义 WAR 包的配置文件
```

11

封装应用

我们通过 Hello World Fax Web 应用，已经琐碎地学习了 Ant 的不同方面的知识了。

现在是时候把我们所学的知识都运用起来创建一个全面和完整的 build.xml 文件了。考虑下面给出的 build.properties 和 build.xml 文件：

build.properties

```
eploy.path = c:\tomcat6\webapps
```

build.xml

```
<?xml version = "1.0"?>

<project name = "fax" basedir = "." default = "usage">

  <property file = "build.properties"/>
  <property name = "src.dir" value = "src"/>
  <property name = "web.dir" value = "war"/>
  <property name = "javadoc.dir" value = "doc"/>
  <property name = "build.dir" value = "${web.dir}/WEB-INF/classes"/>
  <property name = "name" value = "fax"/>

  <path id = "master-classpath">
    <fileset dir = "${web.dir}/WEB-INF/lib">
      <include name = "*.jar"/>
    </fileset>
    <pathelement path = "${build.dir}"/>
  </path>

  <target name = "javadoc">
    <javadoc packagenames = "faxapp.*" sourcepath = "${src.dir}"
      destdir = "doc" version = "true" windowtitle = "Fax Application">

      <doctitle><![CDATA[<h1> =  Fax Application  = </h1>]]>
      </doctitle>

      <bottom><![CDATA[Copyright ? 2011. All Rights Reserved.]]>
      </bottom>

      <group title = "util packages" packages = "faxapp.util.*"/>
      <group title = "web packages" packages = "faxapp.web.*"/>
      <group title = "data packages" packages = "faxapp.entity.*:faxapp.dao.*"/>
```

```
  </javadoc>
</target>

<target name = "usage">
  <echo message = ""/>
  <echo message = "${name} build file"/>
  <echo message = "------------------------------------"/>
  <echo message = ""/>
  <echo message = "Available targets are:"/>
  <echo message = ""/>
  <echo message = "deploy    --> Deploy application as directory"/>
  <echo message = "deploywar --> Deploy application as a WAR file"/>
  <echo message = ""/>
</target>

<target name = "build" description = "Compile main source tree java files">
  <mkdir dir = "${build.dir}"/>

  <javac destdir = "${build.dir}" source = "1.5" target = "1.5" debug = "true"
    deprecation = "false" optimize = "false" failonerror = "true">

    <src path = "${src.dir}"/>
    <classpath refid = "master-classpath"/>

  </javac>
</target>


<target name = "deploy" depends = "build" description = "Deploy application">
  <copy todir = "${deploy.path}/${name}" preservelastmodified = "true">

    <fileset dir = "${web.dir}">
      <include name = "**/*.*"/>
    </fileset>

  </copy>
</target>


<target name = "deploywar" depends = "build" description = "Deploy application as a WAR file">

  <war destfile = "${name}.war" webxml = "${web.dir}/WEB-INF/web.xml">
    <fileset dir = "${web.dir}">
      <include name = "**/*.*"/>
    </fileset>
```

```
    </war>

    <copy todir = "${deploy.path}" preservelastmodified = "true">
      <fileset dir = ".">
        <include name = "*.war"/>
      </fileset>
    </copy>

  </target>


  <target name = "clean" description = "Clean output directories">
    <delete>
      <fileset dir = "${build.dir}">
        <include name = "**/*.class"/>
      </fileset>
    </delete>
  </target>

</project>
```

在上面给出的例子中:

- 我们首先在 build.properties 文件中声明了存放 Tomcat 的 webapp 文件夹的路径，并用变量 deploy.path 来保存。
- 我们声明一个源文件夹来存放 java 文件，并用变量 src.dir 来保存。
- 接下来，我们声明另一个源文夹来存放 web 文件，并用变量 web.dir 来保存。变量 javadoc.dir 用来存储 java 文档，变量 build.dir 是用来存储配置输出文件的路径。
- 然后，我们给这个 web 应用命名，也就是 fax 传真。
- 我们还定义了包含 JAR 文件的基本类路径，在上面给出的项目中也就是： WEB-INF/lib 文件夹。
- 我们还将 build.dir 中的类文件存放在基本类路径下。
- 这个 Javadoc 目标产生项目所需的文档，以及说明目标使用的 javadoc 文档。

上述的例子向我们展示了两个部署目标： deploy 和 deploywar 。

这个 deploy 目标将文件从 web 目录复制到部署目录，并保存最后修改日期时间戳。这样很有用，特别是当我们将项目部署到服务器上,并且该服务器支持热部署。（ 释义：所谓热部署，就是在应用正在运行的时候升级软件，却不需要重新启动应用。）

这个 clean 目标清楚所有之前的构建文件。

这个 deploywar 目标构建 war 文件,然后将 war 文件复制到应用程序服务器的部署目录。

12

部署应用

在之前的章节中，我们已经学会了如何去打包一个应用程序以及怎样将其部署到一个文件夹中。

在这个章节中，我们将要直接将 web 应用程序部署到一个应用服务器的部署文件夹中。随后，我们将添加一些 Ant 目标来启动和停止服务。让我们继续 `Hello World fax` web 应用程序。 这一章节是对之前的章节的一个延续，所有新的组件会用粗体突出显示。

build.properties

```
 # Ant properties for building the springapp

appserver.home=c:\\install\\apache-tomcat-7.0.19
 # for Tomcat 5 use $appserver.home}/server/lib
 # for Tomcat 6 use $appserver.home}/lib
appserver.lib=${appserver.home}/lib

deploy.path=${appserver.home}/webapps

tomcat.manager.url=http://www.tutorialspoint.com:8080/manager
tomcat.manager.username=tutorialspoint
tomcat.manager.password=secret
```

build.xml

```
<?xml version="1.0"?>

<project name="fax" basedir="." default="usage">
  <property file="build.properties"/>
  <property name="src.dir" value="src"/>
  <property name="web.dir" value="war"/>
  <property name="javadoc.dir" value="doc"/>
  <property name="build.dir" value="${web.dir}/WEB-INF/classes"/>
  <property name="name" value="fax"/>

  <path id="master-classpath">
    <fileset dir="${web.dir}/WEB-INF/lib">
      <include name="*.jar"/>
    </fileset>
  <pathelement path="${build.dir}"/>
  </path>

  <target name="javadoc">
    <javadoc packagenames="faxapp.*" sourcepath="${src.dir}"
```

```
    destdir="doc" version="true" windowtitle="Fax Application">

    <doctitle><![CDATA[<h1>= Fax Application = </h1>]]></doctitle>
    <bottom><![CDATA[Copyright © 2011. All Rights Reserved.]]></bottom>
    <group title="util packages" packages="faxapp.util.*"/>
    <group title="web packages" packages="faxapp.web.*"/>
    <group title="data packages" packages="faxapp.entity.*:faxapp.dao.*"/>

  </javadoc>
</target>

<target name="usage">
<echo message=""/>
<echo message="${name} build file"/>
<echo message="------------------------------------"/>
<echo message=""/>
<echo message="Available targets are:"/>
<echo message=""/>
<echo message="deploy    --> Deploy application as directory"/>
<echo message="deploywar --> Deploy application as a WAR file"/>
<echo message=""/>
</target>


<target name="build" description="Compile main source tree java files">
<mkdir dir="${build.dir}"/>
  <javac destdir="${build.dir}" source="1.5" target="1.5" debug="true"
    deprecation="false" optimize="false" failonerror="true">
    <src path="${src.dir}"/>
    <classpath refid="master-classpath"/>
  </javac>
</target>


<target name="deploy" depends="build" description="Deploy application">
  <copy todir="${deploy.path}/${name}"
    preservelastmodified="true">
    <fileset dir="${web.dir}">
      <include name="**/*.*"/>
    </fileset>
  </copy>
</target>


<target name="deploywar" depends="build" description="Deploy application as a WAR file">
```

```
    <war destfile="${name}.war" webxml="${web.dir}/WEB-INF/web.xml">
      <fileset dir="${web.dir}">
        <include name="**/*.*"/>
      </fileset>
    </war>

    <copy todir="${deploy.path}" preservelastmodified="true">
      <fileset dir=".">
        <include name="*.war"/>
      </fileset>
    </copy>
  </target>


  <target name="clean" description="Clean output directories">
    <delete>
      <fileset dir="${build.dir}">
        <include name="**/*.class"/>
      </fileset>
    </delete>
  </target>
```

```
<!-- ============================================================ -->
<!-- Tomcat tasks -->
<!-- ============================================================ -->

<path id="catalina-ant-classpath">
<!-- We need the Catalina jars for Tomcat -->
<!--  * for other app servers - check the docs -->
  <fileset dir="${appserver.lib}">
    <include name="catalina-ant.jar"/>
  </fileset>
</path>

<taskdef name="install" classname="org.apache.catalina.ant.InstallTask">
  <classpath refid="catalina-ant-classpath"/>
</taskdef>
<taskdef name="reload" classname="org.apache.catalina.ant.ReloadTask">
  <classpath refid="catalina-ant-classpath"/>
</taskdef>
<taskdef name="list" classname="org.apache.catalina.ant.ListTask">
  <classpath refid="catalina-ant-classpath"/>
</taskdef>
<taskdef name="start" classname="org.apache.catalina.ant.StartTask">
  <classpath refid="catalina-ant-classpath"/>
</taskdef>
```

```
<taskdef name="stop" classname="org.apache.catalina.ant.StopTask">
  <classpath refid="catalina-ant-classpath"/>
</taskdef>

<target name="reload" description="Reload application in Tomcat">
  <reload url="${tomcat.manager.url}"username="${tomcat.manager.username}"
    password="${tomcat.manager.password}" path="/${name}"/>
</target>
</project>
```

在这个例子中，我们已经使用 Tomcat 作为我们应用的服务器。 首先，在构建属性文件中，我们已经定义了一些附加属性。

- appserver.home 指向 Tomcat 服务器的安装路径。
- appserver.lib 指向 Tomcat 服务器的安装文件下的库文件。
- deploy.path 变量当前指向 Tomcat 的 web 应用程序文件夹。

在 Tomcat 中的应用程序能通过使用 Tomcat 管理应用程序进行启动和停止。管理应用程序的统一资源定位器（URL），用户名和密码也在 build.properties 文件夹中进行指定。 接下来，我们声明一个新的 CLASSPATH 来包含 catalina-ant.jar。 若要通过 Apache Ant 来运行 Tomcat， 这个 jar 文件是必须的。

catalina-ant.jar 提供了下述的任务：

属性	描述
InstallTask	安装一个 web 应用程序。 类名字为： org.apache.catalina.ant.InstallTask
ReloadTask	重新安装一个 web 应用程序。类名字为： org.apache.catalina.ant.ReloadTask
ListTask	列出所有的 web 应用程序。类名字为： Class Name: org.apache.catalina.ant.ListTask
StartTask	启动一个 web 应用程序。类名字为： org.apache.catalina.ant.StartTask
StopTask	停止一个 web 应用程序。类名字为： org.apache.catalina.ant.StopTask
ReloadTask	重新加载一个无需停止的 web 应用程序。类名字为：org.apache.catalina.ant.ReloadTask

重载任务需要下列附加参数：

- 管理应用程序的 URL
- 重启 web 应用程序的用户名
- 重启 web 应用程序的密码
- 重启的 web 应用程序的名字

让我们发出部署 war (deploy-war)的命令来复制 web 应用程序到 Tomcat 的 webapps 文件夹中。同时，我们重新加载传真 web 应用程序。下述的输出是运行 Ant 文件的结果。

```
>C:\>ant deploy-war
>Buildfile: C:\build.xml

>BUILD SUCCESSFUL
>Total time: 6.3 seconds

>C:\>ant reload
>Buildfile: C:\build.xml

>BUILD SUCCESSFUL
>Total time: 3.1 seconds
```

一旦上述的任务被运行，web 应用程序就是已经被部署好且重新加载了的。

13

执行 Java 代码

你可以用 Ant 来执行 Java 代码。在下面的例子中，给出的 java 类文件需要一个参数（管理员的邮箱地址），执行后将发送一封邮件。

```
public class NotifyAdministrator
{
  public static void main(String[] args)
  {
    String email = args[0];
    notifyAdministratorviaEmail(email);
    System.out.println("Administrator "+email+" has been notified");
  }
  public static void notifyAdministratorviaEmail(String email
  {
     //.....
  }
}
```

这里给出上面 java 类文件需要的 build.xml 构建文件。

```
<?xml version="1.0"?>
<project name="sample" basedir="." default="notify">
  <target name="notify">
    <java fork="true" failonerror="yes" classname="NotifyAdministrator">
      <arg line="admin@test.com"/>
    </java>
  </target>
</project>
```

当 build.xml 被执行后，将会产生下面的输出：

```
C:\>ant
Buildfile: C:\build.xml

notify: [java] Administrator admin@test.com has been notified

BUILD SUCCESSFUL
Total time: 1 second
```

在这里例子中， java 代码完成了一件简单的事情 —— 发送一封邮件。我们也可以运用 Ant 任务来完成这项操作。然而，既然你已经有了这个想法，你可以扩展你的 build.xml 文件来调用 java 代码，完成同样的事情，比如：你还可以加密你的代码。

14

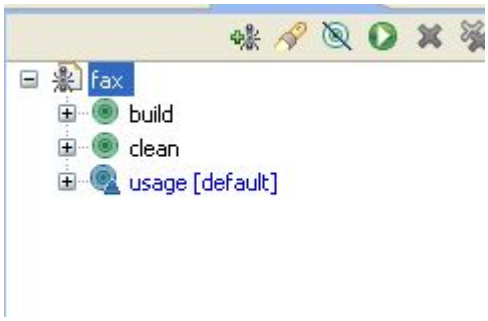
Eclipse 集成

如果你已经下载并安装了 Eclipse， 你只要再做一点点事情就可以开始了。Eclipse 附带预先绑定的 Ant 插件，随时可以使用。

按照以下简单的步骤，将 Ant 集成到 Eclipse 中。

- 确保 build.xml 文件是你的 Java 项目的一部分，并且该文件的位置在项目内。
- 通过以下步骤，启用 Ant 视图， Window > Show View > Other > Ant > Ant 。
- 打开项目资源管理器中，拖动 build.xml 到 Ant 视图。

你的 Ant 视图类似于：

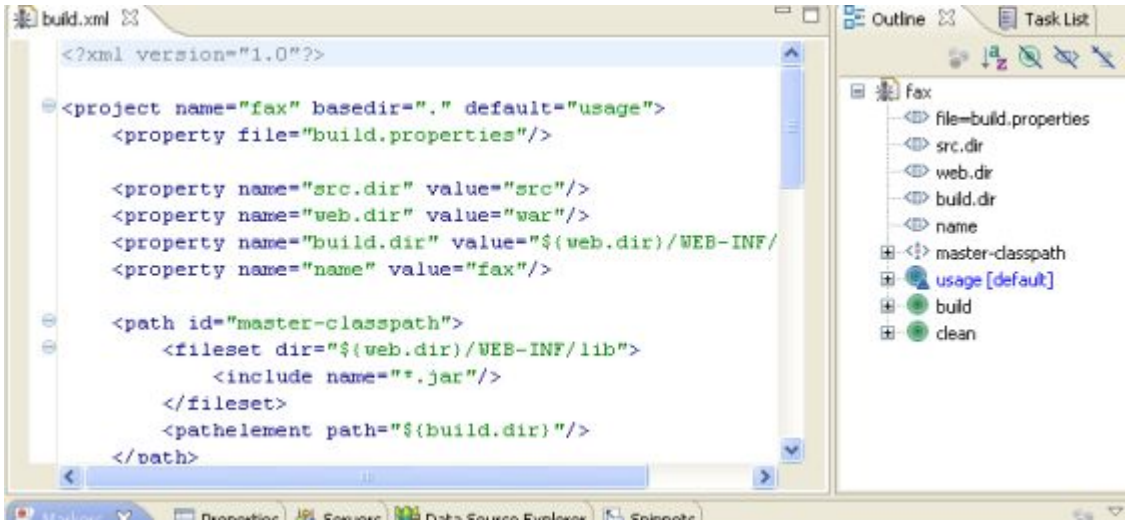


点击目标，build/clean/usage 将运行 Ant 以及目标。

点击 “fax” 将执行默认的目标－usage。

Ant 的 Eclipse 插件还附带了一个很好的编辑器，其可用来编辑 build.xml 文件。该编辑器可以识别 build.xml 模式，并可以协助你完成代码。

为了使用 Ant 编辑器，右键单击你的 build.xml（从工程资源管理器中），然后选择用 Ant 编辑器打开。 Ant编辑器看起来应该类似于下图：



15

JUnit 集成

JUnit 是一个 Java 语言的单元测试框架。它易于使用且易于扩展。有许多可用的 JUnit 扩展。如果你对 JUnit 不是很熟悉，你可以在 www.junit.org 网址上下载相关手册。

这一章节将向你展示如何用 Ant 来扩展 JUnit 。Ant 直接使用 JUnit 任务。

下面给出 JUnit 任务的相关属性：

属性	描述
dir	表示从哪里调用 VM。当 **fork** 被禁用的时候，这个属性将会被忽略。
jvm	表示调用 JVM 的命令。当 **fork** 被禁用的时候，这个属性将会被忽略。
fork	表示在独立的 java 虚拟机中运行测试文件。
errorproperty	表示当有一个 JUnit 失效的时候，设置属性的名字。
failureproperty	表示当有一个 JUnit 失效的时候，设置属性的名字。
haltonerror	表示当一个测试有错误的时候，停止执行过程。
haltonfailure	表示当有故障发生的时候，停止执行过程。
printsummary	表示告知 Ant 展示每个测试例子的简单统计。
showoutput	表示告知 Ant 展示输出结果到 log 日志文件或者格式器上。
tempdir	表示存放 Ant 将会运用到的临时文件。
timeout	表示测试过程耗时太长，超过了设置时间（毫秒级）。

让我们继续 Hello World 传真应用这个主题，并加入 JUnit 任务。

下面给出的例子展示了一个简单的 JUnit 测试例子的执行过程：

```
<target name="unittest">
  <junit haltonfailure="true" printsummary="true">
    <test name="com.tutorialspoint.UtilsTest"/>
  </junit>
</target>
```

上面给出的例子展示了 com.tutorialspoint.UtilsTest junit 类的执行过程。运行上面的代码，将会看到以下输出：

```
test:
[echo] Testing the application
[junit] Running com.tutorialspoint.UtilsTest
[junit] Tests run: 12, Failures: 0, Errors: 0, Time elapsed: 16.2 sec
BUILD PASSED
```

16

扩展 Ant

Ant 带有一组预定义的任务，但是你可以创建自己的任务，如下面的例子所示。

定制 Ant 任务应扩展 org.apache.tools.ant.Task 类，同时也应该拓展 execute() 方法。下面是一个简单的例子：

```
package com.tutorialspoint.ant;

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

public class MyTask extends Task {
  String message;
  public void execute() throws BuildException {
    log("Message: " + message, Project.MSG_INFO);
  }

  public void setMessage(String message) {
    this.message= message;
  }
}
```

为了运行定制的任务，你需要添加下列内容到 Hello World 传真 web 应用程序中：

```
<target name="custom">
  <taskdef name="custom" classname="com.tutorialspoint.ant.MyTask" />
  <custom message="Hello World!"/>
</target>
```

执行上述的定制任务后将打印出一条信息为： 'Hello World!'

```
>c:\>ant custom
>test:
>[custom] Message : Hello World!
>elapsed: 0.2 sec
>BUILD PASSED
```

这只是一个简单的例子，你可以使用 Ant 的能力去做任何你想提高你的构建和部署过程效率的事情。