第一讲 Ant 入门

1 Ant 介绍

1.1 Ant 的起源

Ant 的作者 James Duncan Davidson 给**《Ant 权威指南》**一书做过评论，现在我们就引用这段评论来说明 Ant 的起源。

> 1998 年，有一位程序员改变了整个 Java 世界。James Duncan Davisdson 在试图使用当时的构建工具（GNU Make、批处理文件和 shell 脚本）来创建一个跨平台的 Tomcat 构建时，做了多种努力均不能成功。因此，他在从欧洲飞回美国的途中设计出了自己的 构建实用工具，并为命名为 Ant，因这是一个小东西，但却能做大事。James 为了解决自己的问题（即创建一个跨平台的构建）而提出的这种快速而简单的解决方案已经演变成 Java 环境中应用最为广泛的构建管理工具。
>
> 如果你在用 Java 进行开发，而没有使用 Ant，那么确定应该拥有这本不算厚的书。Ant 是一个可靠的、跨平台的构建工具，它可以充分利用 Java 环境的优势。Ant 本身也是用 Java 编写的，它可在多个平台（如 Unix、Linux 以及 Windows 等等）上工作，因此如果你要转换开发平台，那么学习 Ant 是值得的，Ant 很容易扩展。在你的工程中，目前是否存在某个需求，而所有 Ant 功能对此都不能予以满足呢？没有问题！你完全可以像 James 一 样，编写自己的 Ant 任务。没准儿你也能改变这个世界呢！
>
> 以上是使用 Ant 可以完成的一些工作：
>
> ·定义构造块、它们必须产生的结果以及它们之间的依赖关系；
>
> ·自动地由诸如 CVS 等源代码控制系统获取源代码；
>
> ·不仅可令 Ant 以适当的顺序编译必要的源文件，而且还可生成部署都所必需的 JAR 文件，由此完成应用的构造；
>
> ·仅需由一个构造文化（或一组构建文件）即可完成以上所有工作，而构建文件在 Ant 支持的所有平台上都会有同样的表现。
>
> 要进行 eXtreme（极限）编程吗？Ant 就是一个必备工具，利用它可以方便地完成频繁的构建，以此又将有利于使用 JUnit 和相关技术进行测试驱动的开发。
>
> Ant 构建文件是 XML 编写的，这是一种良构标准，因此可以确保使用 Ant 并不需要学习另一种脚本语言。Ant 是一个开源工程，这是 Apache 软件基金会所投资的 Jakarta 工程的一部分。
>
> “Jesse 和 Eric 可以教你如何使用当前的 Ant，他们做得相当棒。这本书提供了颇有深度的信息，对于一些以往从未考虑到的工作，书中还介绍了如何扩 展 Ant 来完成这些工作的详细内容。不论你是偶尔使用 Ant，还是需要管理各种大型工程，这都是一本需要持之在手的工具书。”
>
> —James Duncan Davidson，Ant 的创造者

1.2 什么是 Ant

Apache Ant 是一个将软件编译、测试、部署等步骤联系在一起加以优化的一个构建工具，常用于 Java 环境中的软件开发。Ant 的默认配置文件是 build.xml。

1.3 什么是构建

形象的说，构建就是把代码从某个地方拿来、编译、再拷贝到某个地方去等操作，当然不仅于此，但是主要用来干这个。

1.4 Ant 的优势

（1）跨平台：因为 ant 是使用 java 实现的，所以跨平台；

（2）使用简单：比其他构建工具简单，特别是和 ant 的兄弟 make 做比较；

（3）语法清晰：同样是和 make 做比较；

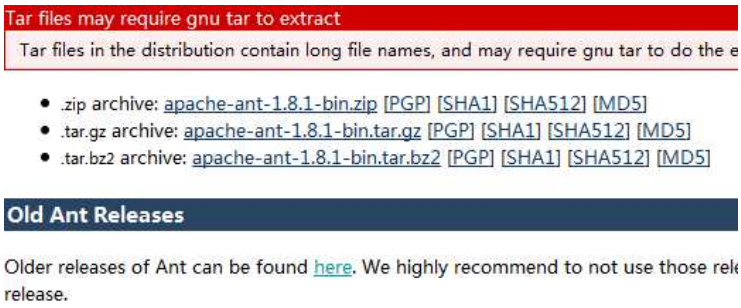
（4）功能强大：Ant 能做的事情很多，可能你用了很久，你仍然不知道它能有多少功能。当你需要自己开发一些 ant 插件的时候，你会发现它更多的功能。

Ant 于 Make 的区别：make 是一个类似于 Ant 的构建工具，不过 make 应用于 C/C++，Ant 则主要应用于 java。当然这不是一定的，当大部分人如此。

2 Ant 环境配置

2.1 下载

首先到 http://ant.apache.org/bindownload.cgi 下载 Ant，目前的最新版本是 1.8.1



我们可以下载最新的版本，也可以点**"here"**去下载历史版本。我们以最新的版本 apache-ant-1.8.1 来做演示。

2.2 配置环境变量

第一步：解压。我们以 C:\java\apache-ant-1.8.1 为例；

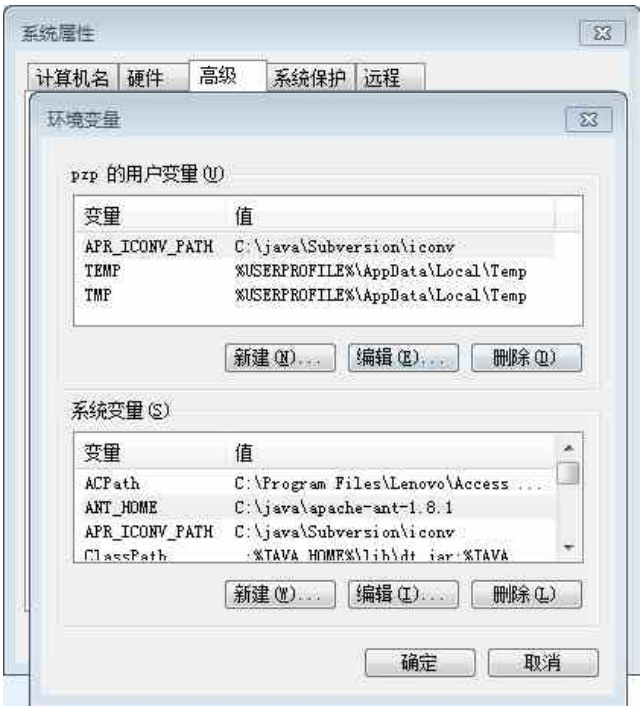
把上面下载的 apache-ant-1.8.1-bin.zip 解压到 C:\java\apache-ant-1.8.1；

新增系统变量：ANT_HOME，内容：C:\java\apache-ant-1.8.1；

第二步：增加环境变量。

在 PATH 环境变量中加入 Ant 的 bin 目录：%ANT_HOME%\bin；

如果要让 Ant 能支持 JUnit，需要直接将 JUnit 的 junit.jar 放置在 Ant 的 lib 目录，并记得修改变量 CLASSPATH 中原先有关于 JUnit 的设定，例如：%ANT_HOME\lib\junit.jar，虽然也有其它的方式可以设定，但这是最快最简单的方法。以下是 windows 中的设置，请看图：



依赖库说明（简单看下）

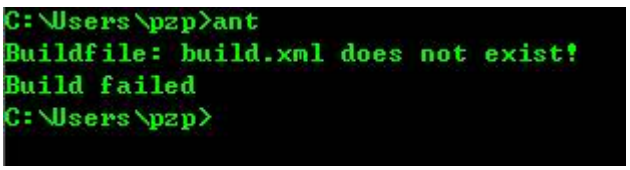
如果你需要执行特定的 task，你需要将对应的库放入 CLASSPATH 或放到 Ant 安装目录的 lib 目录下。

Jar Name	Needed For	Available At
jakarta-regexp-1.2.jar	regexp type with mappers	jakarta.apache.org/regexp/
jakarta-oro-2.0.1.jar	regexp type with mappers and the perforce tasks	jakarta.apache.org/oro/
junit.jar	junit tasks	www.junit.org
stylebook.jar	stylebook task	CVS repository of xml.apache.org
testlet.jar	test task	java.apache.org/framework
antlr.jar	antlr task	www.antlr.org
bsf.jar	script task	oss.software.ibm.com/developerworks/projects/bsf
netrexx.jar	netrexx task	www2.hursley.ibm.com/netrexx
rhino.jar	javascript with script task	www.mozilla.org
jpython.jar	python with script task	www.jpython.org
netcomponents.jar	ftp and telnet tasks	www.savarese.org/oro/downloads

2.3 运行 Ant

运行 Ant 非常简单，正确安装后，只要在命令行下输入 ant 就可以了。

开始-->运行-->cmd，进入命令行-->键入 ant 回车,如果看到



看到如下信息：

Buildfile: build.xml does not exist!

Build failed

说明配置成功。

没有指定任何参数时，Ant 会在当前目录下查询 build.xml 文件。如果找到了就用该文件作为 bui

如果你用 *-find* 选项。Ant 就会在上级目录中寻找 buildfile，直至到达文件系统的根。要想让 Ant

他的 buildfile，可以用参数 *-buildfile file*，这里 *file* 指定了你想使用的 buildfile。

我们也可以设定一些属性，以覆盖 buildfile 中指定的属性值。可以用 *-Dproperty=value* 选项，这里 *property* 是指属性的名称，而 *value* 则是指属性的值。我们也可以用这种办法来指定一些环境变量的值，再使用 property task 来存取环境变量。如-Daaa=aaa 传递给 Ant，我们就可以在我们的 buildfile 中用 ${aaa}来存取这些环境变量。

还有两个选项 *-quite*，告诉 Ant 运行时只输出少量的必要信息。而 *-verbose*，告诉 Ant 运行时要输出更多的信息。

可以指定执行一个或多个 target。当省略 target 时，Ant 使用标签<project>的 default 属性所指定的 target。如果有的话，-projecthelp 选项输出项目的描述信息和项目 target 的列表。先列出那些有描述的，然后是没有描述的 target。

命令行选项总结：

ant [options] [target [target2 [target3] ...]]

Options:

-help print this message

-projecthelp print project help information

-version print the version information and exit

-quiet be extra quiet

-verbose be extra verbose

-debug print debugging information

-emacs produce logging information without adornments

-logfile file use given file for log output

-logger classname the class that is to perform logging

-listener classname add an instance of class as a project listener

-buildfile file use specified buildfile

-find file search for buildfile towards the root of the filesystem and use the first one found

-Dproperty=value set property to value

例子:

ant

使用当前目录下的 build.xml 运行 Ant，执行缺省的 target。

ant -buildfile test.xml

使用当前目录下的 test.xml 运行 Ant，执行缺省的 target。

ant -buildfile test.xml dist

使用当前目录下的 test.xml 运行 Ant，执行一个叫做 dist 的 target。

ant -buildfile test.xml -Dbuild=build/classes dist

使用当前目录下的 test.xml 运行 Ant，执行一个叫做 dist 的 target，并设定 build 属性的值为 build/classes。

3 Ant 核心概念简介

XML：构建文件是以 XML 文件来描述的。

project：每个构建文件包含一个工程。

depends：每个工程包含若干个目标(target)。目标可以依赖于其他的目标。

task：目标包含一个或多个任务(task)。

易于扩展：易于使用 Java 语言增加新的任务：（自定义）。

语法简单：构建文件短小精悍，语法直观且易于理解。

4 Ant 初体验

就像每个语言都有 HelloWorld 一样，一个最简单的应用能让人感受一下 Ant。接下来我们将假想一个需求，再针对这个需求，写一个 Ant 脚本完成该需求。

4.1 假想一个需求

（1）编写一个程序

（2）编译它们

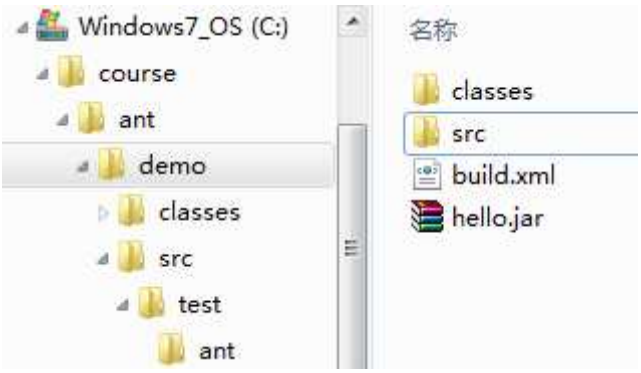
（3）把这个程序打成 jar 包

（4）把他们放在应该放置的地方

（5）运行它们

首先看下目录结构。

目录结构图示



目录结构文字说明

在建立 build.xml 文件之前，我们先说下目录结构：

工程主目录： C:\course\ant\demo

源程序目录： \src

编译后的 class 文件目录：\classes

打包好的 jar 程序目录：\hello.jar

工程配置文件：\build.xml

4.2 写第一个 Ant 配置文件

上面定义了需求，这一节主要是建立一个 build.xml 文件完成上面的步骤。

首先，我们使用文本编辑器写第一个程序：**HelloWorld.java**

```
package test.ant;
public class HelloWorld {
```

```
    public static void main(String[] args)
        System.out.println("hello world");
    }
}
```

再建立一个 build.xml 文件完成上面的步骤，内容如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<project name="HelloWorld" default="run" basedir=".">
    <property name="src" value="src" />
    <property name="dest" value="classes" />
    <property name="hello_jar" value="hello.jar" />

    <target name="init">
        <mkdir dir="${dest}" />
    </target>

    <target name="compile" depends="init">
        <javac srcdir="${src}" destdir="${dest}" />
    </target>

    <target name="jar" depends="compile">
        <jar jarfile="${hello_jar}" basedir="${dest}" />
    </target>

    <target name="run" depends="jar">
        <java classname="test.ant.HelloWorld" classpath="${hello_jar}" />
    </target>

    <target name="clean">
        <delete dir="${dest}" />
        <delete file="${hello_jar}" />
    </target>

    <target name="rerun" depends="clean, run">
        <ant target="clean" />
        <ant target="run" />
    </target>
</project>
```

4.3 解释以上 Ant 配置文件

```
<project name="HelloWorld" default="run" basedir=".">
</project>
```

Ant 的所有内容必须包含在这个里面，name 是我们给它取的名字，

basedir 看名字就知道是工作的根目录。其中的”.”代表当前目录。

default 代表默认要做的事情。

```
    <property name="src" value="src" />
    <property name="dest" value="classes" />
    <property name="hello_jar" value="hello.jar" />
```

类似程序中的变量，想一下变量的作用 就知道为什么要这么做了

```
    <target name="init">
        <mkdir dir="${dest}" />
    </target>
```

目的：创建一个文件夹。

把我们想做的每一件事写成一个 target，它有一个名字。使用 mkdir 来创建文件夹。

将第一个 target 命名为 init，是一个很好的做法，其他的 target 依赖这个初始化 target。

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

目的：编译程序，把源程序编译到目标地点。

depends 是它所依赖的 target，在执行这个 compile 的 target 之前，ant 会先检查 init 是否已经被执行过，如果执行过，着直接执行 compile，如果没有执行过，就会执行它依赖的 target，然后再执行本身的 target。

```
    <target name="jar" depends="compile">
        <jar jarfile="${hello_jar}" basedir="${dest}" />
    </target>
```

目的：打 jar 包

```
    <target name="run" depends="jar">
        <java classname="test.ant.HelloWorld" classpath="${hello_jar}" />
```

```
    </target>
```
目的：运行 jar 包内的程序

```
    <target name="clean">
        <delete dir="${dest}" />
        <delete file="${hello_jar}" />
    </target>
```
目的：删除生成的文件

```
    <target name="rerun" depends="clean, run">
        <ant target="clean" />
        <ant target="run" />
    </target>
```
目的：再次运行，这里显示了如何在一个 target 里边调用其他的 target

步骤再重申一遍：

（1） 建一个 src 文件夹，然后写一个 HelloWorld.java，按照包结构目录放进去；

（2） 写 build.xml 文件

（3） 在命令行输入 cmd, ant 做测试。

我们会发现上面的任务一个接一个的完成了。每次修改代码只要再次键入 ant

另外，我们有时候并不想运行默认的任务，或者想执行任意的步骤，
例如只想执行编译，则键入 ant compile

ant 中的每一个任务都可以这样执行，ant + target name
讲到这里，一个简单的任务已经完成，大家动手去体验一下 Ant 吧。

5 参考文献

1、http://www.apache.org

Ant 官方手册，文中大量引用了 Ant 官方手册中的例子

2、**《Ant 权威指南》**

作者: Jesse Tilly, Eric M. Burke 著, James Duncan Davidson 序，林琪 译
出版: 2003 年 7 月

第 2 讲 Ant 进阶

1 回顾上一讲

学习了上一讲，相信已经激起大家的兴趣，或许还有不少感触。当然上一讲我们只了解了 Ant 的九牛一毛，这一讲将会更加深入的介绍 Ant 的方方面面。

2 核心概念详解

2.1 project 属性

首先看上一讲中的例子：<project name="HelloWorld" default="run" basedir=".">

可以看出 project 有下面的属性：

Attribute	Description	Required
name	项目名称，可自由定义。	否
default	定义一个缺省（默认）的 target，当我们执行没有指定 target 的 build.xml 时，就会执行这个缺省的 target。	是
basedir	用于定义项目的基路径。	否
description	表示项目的描述	否

项目的描述以一个顶级的<description>元素的形式出现。

2.2 property 属性

一个项目可以有很多的 property。可以在 buildfile 中用 property task 来设定，或者在 ant 之外设定。一个 property 有一个名字和一个值。Property 可以用于 task 的属性值。这是通过将属性名放在“${}”之间并放在属性值的位置来实现的。例如有一个 property builddir 的值是“build”，这个 property 就可用于属性值：${builddir}/classes。这个值就可被解析为 build/class。

典型的如第一讲中的例子，

首先设定 property：

```
<property name="src" value="src" />
<property name="dest" value="classes" />
```

然后在其他地方引用 property：

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

2.3 内置属性

如果你使用了<property>定义了所有的系统属性，Ant 允许你使用这些属性，

例如${os.name}对应操作系统的名字。

要想得到系统属性的列表可以参考 the javadoc of System.getProperties。

使用如下测试代码可以查看一下：

```
public void testProperties() {
```

```
        Properties p = System.getProperties();
        for (Enumeration<?> e = p.propertyNames(); e.hasMoreElements();) {
            String key = (String)e.nextElement();
            System.out.println(key + ": " + p.getProperty(key) );
        }
    }
```

上面的代码就不做详细描述了，这些应该是看这个文档的人应该能看懂的。

除了 Java 的内置属性外，Ant 还定义了一些自己的内置属性：

Attribute	Description
basedir	可以理解为 project 项目的绝对路径，于<project>中的 basedir 属性一样
ant.file	buildfile 的绝对路径
ant.version	ant 的版本
ant.propertie.name	当前执行的 project 的名字，由<project>的 name 属性设定
ant.java.version	ant 检测到的 jvm 版本。
.....	.....

2.4 target 属性

还是看上一讲中的例子： <target name="compile" depends="init">，

target 可以有如下几个属性。

Attribute	Description	Required
name	target 的名字。	是
depends	用逗号分隔的 target 的名字列表，也就是依赖列表	否
if	执行 target 需要清除设定的属性名	否
unless	执行 target 需要清除设定的属性名	否
description	描述 target 的功能，这些描述可由-projexthelp 命令行输出。	否

2.4.1 depends

target 中的 depends 属性指定了 target 的执行顺序，也就是说一个 target 可以依赖
target。Ant 会依照 depends 属性中的 target 出现顺序依次执行每个 target。在执行前，
要执行它所依赖的 target。

例如，有一个用于编译的 target，一个用于生成执行文件的 target。在生成执行文件之

先编译通过，所以生成可执行文件的 target 依赖于编译 target。Ant 会处理这种依赖关系。

我们还要注意到，Ant 的 depends 属性只指定了 target 应该被执行的顺序。如果被依赖的 target 无法运行，这种 depends 对于指定了依赖关系的其他 target 没有影响。

Ant 会依照 depends 属性中 target 出现的顺序依次执行每个 target。然而，要记住的是只要某个 target 依赖于一个 target，后者就会被先执行。看下面的例子。

<target name="A"/> <target name="B" depends="A"/> <target name="C" depends="B"/> <target name="D" depends="C "/>	假设我们默认的 target 是 D，因为 D 依赖于 C，C 依赖于 B，B 依赖于 A。所以先执行 A，再执行 B，然后是 C，最后 D 被执行。

还可以写为这样的格式： <target name="D" depends="C,B,A"/>，执行顺序和上面的一样。

另外，一个 target 只能执行一次，即使有多个 target 依赖于它，也只执行一次。

2.4.2 if / unless

如果执行某个 target，由某些属性是否被设定来决定。这样，就能根据系统的状态（java version, OS, 命令行属性定义等等）来更好的控制 build 的过程。要想让一个 target 这样做，你就应该在 target 元素中，加入 if 或 unless 属性，带上 target 应该有所判断的属性。看下面的例子。

```
<target name="install-windows" if="osfamily-windows">
<target name="install-unix" if="osfamily-unix">
```

如果没有 if 或者 else，target 总会被执行。

2.5 task 属性

一个 task 是一段可执行的代码。task 可以有多个属性（也可以叫变量）。属性值可能包含对 property 的引用。这些应用会在 task 执行前被解析。

还是看上一讲中的例子：

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

其中的<javac srcdir="${src}" destdir="${dest}" /> 就是一个 task。

其中的 ${src} 就是对前面定义的 property 的引用。再执行 javac 之前这些${*} 会被解析。

下面是 task 的一般构造形式：

<name attribute1="value1" attribute2="value2" … />

这里 name 是 task 的名字，attribute 是属性名，value 是属性值。

另外，我们可以使用内置的 task 和一些可选的 task，我们也可以编写自己的 task。

3 Ant 常用 task

3.1设置 classpath

（1）当需要指定类似路径的值时，可以使用嵌套元素。一般的形式是

```
<classpath>
    <pathelement path="${classpath}"/>
    <pathelement location="lib/jar.jar"/>
</classpath>
```

location 属性指定了相对于 project 基目录的一个文件和目录，而 path 属性接受逗号或分号分隔的一个位置列表。path 属性一般用作预定义的路径——其他情况下，应该用多个 location 属性。

注意：你可以用":"和";"作为分隔符，指定类似 PATH 和 CLASSPATH 的引用。Ant 会把分隔符转换为当前系统所用的分隔符。

为简洁起见，classpath 标签支持自己的 path 和 location 属性。所以：

```
<classpath>
    <pathelement path="${classpath}"/>
</classpath>
```

可以被简写作：

```
<classpath path="${classpath}"/>
```

（2）也可通过<fileset>元素指定路径。构成 fileset 的多个文件加入 classpath 的顺序是未定的。

```
<classpath>
    <pathelement path="${classpath}"/>
    <fileset dir="lib">
        <include name="**/*.jar"/>
    </fileset>
    <pathelement location="${dest}"/>
</classpath>
```

上面的例子构造了一个路径值包括：${classpath}的路径，跟着 lib 目录下的所有 jar 文件，接着是${dest}目录。

（3）References

如果我们想在多个 task 中使用相同的 classpath，可以用<path>元素定义它们（与 target 同级），然后通过引用

buildfile 元素的 id 属性可用来引用这些元素。如果你需要一遍遍的复制相同的 XML 代码

一属性就很有用－－如多次使用<classpath>结构。

下面的例子： **rmic 代表任意一个 task**

```
<project…>
  <target…>
      <rmic…>
    <classpath>
    <pathelement location="lib/" />
    <pathelement path="${java.class.path}/"/>
      <pathelement location="${dest}"/>
    </classpath>
      </rmic>
  </target>

  <target…>
      <javac>
          ……
    <classpath>
    <pathelement location="lib/" />
    <pathelement path="${java.class.path}/"/>
      <pathelement location="${dest}"/>
    </classpath>
  </target>
</ project >
```

可以写成如下形式：

```
<project…>

  <path id="project.class.path">
      <pathelement location="lib/" />
      <pathelement path="${java.class.path}/"/>
      <pathelement location="${dest}"/>
  </path>
  <target…>
      <rmic…>
          <classpath refid="project.class.path"/>
      </rmic>
  </target>
```

```
  <target…>
      <javac>
        <classpath refid="project.class.path"/>
      </javac>
  </target>
</ project >
```

3.2使用 classpath

```
<target>
<javac>
    <classpath refid="project.class.path"/>
</javac>
</target>
```

3.3 输出信息

可以有多种输出信息的写法：

```
写法1：输出一段话
<echo message="XXXXXX">
```

```
写法2：输出一段话
<echo>XXXXXX</echo>
```

```
写法3：输出一段XML
<echoxml file="subbuild.xml">
    <project default="foo">
        <target name="foo">
            <echo>foo</echo>
        </target>
    </project>
</echoxml>
```

3.4 设置 property

（1）设置属性 name-value

```
<property name="src" value="src" />
```

（2）读取属性文件中的属性配置

```
<property file="foo.properties"/>
```

（3）读取网络中的 **property-set**

```
<property url="http://www.mysite.com/bla/props/foo.properties"/>
```

（4）读取文件中的属性配置

```
<property resource="foo.properties"/>
```

（5）读取环境变量

```
<property environment="env"/>
```

（6）读取属性文件中的属性，并作为全局引用

```
<property file="/Users/antoine/.ant-global.properties"/>
```

3.5 引入 properties 文件

读取 properties 文件中的属性配置

首先在项目路径下增加一个 build.properties 文件，用于提取设置公共的属性

比如针对上一讲中的

```
<property name="src" value="src" />
<property name="dest" value="classes" />
<property name="hello_jar" value="hello.jar" />
```

首先我们可以在 build.properties 中这么写

```
src = src
dest = dest
hello_jar = hello.jar
```

然后在 build.xml 中这样去修改：

```
<property file="build.properties" />
<property name="src" value="${ src } " />
<property name="dest" value="${ classes } " />
```

```
<property name="hello_jar" value="${ hello.jar } " />
```

3.6 引入 XML 文件

读取 XML 文件中的属性配置

首先在项目路径下增加一个 build_include.xml 文件，用于提取设置公共的属性，比如针对上一讲中的，内容如下

```
<property name="src" value="src" />
<property name="dest" value="classes" />
<property name="hello_jar" value="hello.jar" />
```

首先我们可以在 build_include.xml 中这么写

```
<?xml version="1.0" encoding="UTF-8" ?>
<property name="src" value="src"/>
<property name="dest" value="classes"/>
<target name="test" >
    <ant target="run" />
</target>
```

然后在 build.xml 中这样去修改：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--include a xml file ,it can be common propery ,can be also a target    -->
<!DOCTYPE project [
<!ENTITY share-variable SYSTEM "file:../include.xml">
]>
<project name="HelloWorld" default="run" basedir=".">
    .....
</project>
```

这个时候，我们只要在 include.xml 添加 propery，添加 target，
这个 build.xml 会自动添加这些 propery 和 target。

看下面的引用

```
build.xml 中的可以修改 peoperty 设置
<!--via include
<property name="src" value="src"/>
<property name="dest" value="classes"/>
```

```
-->
<target name="init">
    <mkdir dir="${dest}"/>
</target>
<target name="compile" depends="init">
    <javac srcdir="${src}" destdir="${dest}"/>
</target>
<target name="rerun"depends="clean,run">
    <ant target="clean"  />
    <ant target="run"      />
</target>
```

上面的${src}、${dest}、run 就是引用 xml 中的 target。

3.7 复制文件与目录

该标签用于删除一个文件或一组文件，其属性如下：
(1) file 表示要删除的文件。
(2) dir 表示要删除的目录。
(3) includeEmptyDirs 表示指定是否要删除空目录，默认值是删除。
(4) failonerror 表示指定当碰到错误是否停止，默认值是自动停止。

（1）复制一个**文件**到当前目录，并重命名；

```
<copy file="myfile.txt" tofile="mycopy.txt"/>
```

（2）复制一个**文件**到指定目录；

```
<copy file="myfile.txt" todir="build /newdir"/>
```

（3）复制一个**文件**到指定位置（使用 copyfile）；

```
<copyfile src="test.java" dest="subdir/test.java"/>
```

（4）复制一个**文件集合**到一个目录；

```
<copy todir="build/newdir">
    <fileset dir="src_dir"/>
</copy>
```

（5）复制一个**文件集合**到一个目录，但不包含某些文件；

```
第一种写法：
<copy todir="dest/dir">
```

```
        <fileset dir="src_dir">
          <exclude name="**/*.java"/>
        </fileset>
    </copy>
    第二种简便写法：
    <copy todir="dest/dir">
        <fileset dir="src_dir" excludes="**/*.java"/>
    </copy>
```

（6）复制一个**文件集合**到一个目录，同时建立备份文件；

```
    <copy todir="backup/dir">
        <fileset dir="src_dir"/>
        <globmapper from="*"   to="*.bak"/>
    </copy>
```

（7）复制一个**文件集合**到一个目录，并替换掉其中的字符，如替换掉@TITLE@；

```
    <copy todir="../backup/dir">
        <fileset dir="src_dir"/>
        <filterset>
            <filter token="TITLE" value="Foo Bar"/>
        </filterset>
    </copy>
    这里演示的是通过 Ant 批量转换文件的内容：设定 ant 的 filter token, 把该文件目录下
    的所有文件内的“@TITLE@”， 替换为"Foo Bar"
```

（8）复制一个**目录**到另一个目录

（使用 copydir，include 包含和 excludes 排除）；

```
    <copydir   src="${src}/resources"
               dest="${dist}"
               includes="**/*.java"
               excludes="**/Test.java"
    />
```

3.8 删除文件与目录

该标签用于删除一个文件或一组文件，其属性如下：
(1) file 表示要删除的文件。

(2) dir 表示要删除的目录。
(3) includeEmptyDirs 表示指定是否要删除空目录，默认值是删除。
(4) failonerror 表示指定当碰到错误是否停止，默认值是自动停止。

（1）删除一个**文件**

```
<delete file="/lib/aaa.jar"/>
```

（2）删除一个**目录**

```
<delete dir="lib"/>
```

（3）删除一个**目录下符合规则的文件**，包括子目录。这里是删除.bak 文件

```
<delete>
    <fileset dir="." includes="**/*.bak"/>
</delete>
```

（4）删除一个**目录下所有文件和目录**，包括当前目录

```
<delete includeEmptyDirs="true">
    <fileset dir="build"/>
</delete>

    <fileset dir="build"/>也可以写成<fileset dir="build" includes="**/*"/>
```

（5）删除**当前目录**下所有的 svn 相关文件

```
通常情况下，svn 文件默认是 excludes 的，所以这里要设置一下
<delete defaultexcludes="false">
    <fileset dir="src" includes="**/*.svn"/>
</delete>
```

（6）删除**文件目录树**

```
<deltree dir="dist"/>
```

3.9 剪切文件、文件集合、目录

```
<move todir="new/dir">
    <fileset dir="src/dir">
      <include name="**/*.jar"/>
```

```
        <exclude name="**/aaa.jar"/>
      </fileset>
  </move>
```

3.10 重命名文件、文件集合、目录

```
  <rename src="foo.jar" dest="foo-${version}.jar"/>
```

3.11 建立临时文件

建立一个文件名称为 temp.xml，后缀为.bak 的文件

```
  <tempfile  property="temp. xml"  destDir="build"  suffix=".bak"/>
```

3.12 touch 的使用

touch 主要是用于修改最后访问时间

（1）如果文件不存在,创建文件，如果存在,更改最后访问时间为当前系统时间

```
  <touch file="myfile"/>
```

（2）如果文件不存在,创建文件，更改最后访问时间为 11/21/2010 2:02 pm

```
  <touch file="myfile" datetime="11/21/2010 2:02 pm"/>
```

（3）如果文件不存在,创建文件，更改最后访问时间为 11/21/2010 2:02 pm

```
  <touch datetime="11/21/2010 2:02 pm">
      <fileset dir="src_dir"/>
  </touch>
```

3.13 condition 的使用

有<and> ,<or>,<not>等 tag

```
  <condition property="isMacOsButNotMacOsX">
      <and>
          <os family="mac"/>
              <not>
                  <os family="unix"/>
```

```
            </not>
        </and>
    </condition>
```

3.14 替换 replace

```
    <replace file="configure.sh"
            value="defaultvalue"
            propertyFile="source/name.properties">
      <replacefilter   token="@token1@"/>
      <replacefilter   token="@token2@"   value="value2"/>
      <replacefilter   token="@token3@"   property="property.key"/>
    </replace>
```

3.15 available 的使用

Available 主要用于判断 某个文件是否存在，再决定某个属性为 true 还是 false。

（1）如果类存在，则设置属性 Myclass.present 为 true，如果没有就 false

```
    <available                                    classname="org.whatever.Myclass"
    property="Myclass.present"/>
```

（2）如果文件存在则设置属性 jaxp.jar.presen 为 true，否则为 false

```
    <property name="jaxp.jar" value="./lib/jaxp11/jaxp.jar"/>
    <available file="${jaxp.jar}" property="jaxp.jar.present"/>
```

（3）如果目录存在，则设置属性为 **true**，否则为 **false**

```
    <available   file="/usr/local/lib"   type="dir"   property="local.lib.present"/>
```

（4）如果 **classpath** 下寻找 **class**，如果存在则设置属性为 **true**，否则为 **false**

```
    在工程 tag 下定义 path，在 target 中使用
    ...in project ...
    <property name="jaxp.jar" value="./lib/jaxp11/jaxp.jar"/>
    <path id="jaxp" location="${jaxp.jar}"/>

    ...in target ...
    <available classname="javax.xml.transform.Transformer"
```

```
classpathref="jaxp" property="jaxp11.present"/>
```

（5）如果在 **classpath** 下发现文件则设置属性为 **true** 否则为 **flase**

```
<available property="have.extras" resource="extratasks.properties">
  <classpath>
    <pathelement location="/usr/local/ant/extra.jar" />
  </classpath>
</available>
```

3.17 显示错误

（1）显示错误方式一

```
<fail>Something wrong here.</fail>
```

（2）显示错误方式二

```
<fail message=" ${属性}." />
```

（3）如果这个属性不存在显示错误

```
<fail unless="failCondition" message="unless Condition "/>
```

（4）如果这个属性存在显示错误

```
<fail if=" failCondition " message="if Condition "/>
```

（5）如果符合条件显示错误，这里的条件是（没有设置属性）

```
<fail message="tag condition">
<condition>
    <not>
    <isset property=" failCondition    "/>
    </not>
</condition>
</fail>
```

3.18 建立目录

建立一个目录

```
<mkdir dir="./build/lib"/>
<mkdir dir="${dist}/lib"/>
```

3.19 执行程序（cmd 命令）

```
<target name="help">
<exec executable="cmd">
    <arg value="/c"/>
    <arg value="ant.bat"/>
    <arg value="-p"/>
</exec>
</target>
```

3.20 编译程序（执行 javac）

```
<javac encoding="utf-8"
        srcdir="${src.dir}"
        destdir="${build.classes.dir}"
        debug="on"
        deprecation="false"
        optimize="true"
        failonerror="true"
        source="1.6"
        target="1.6">
    <classpath refid="classpath" />
</javac>
```

注：Ant 的 Target <javac/> (Target <batchtest/>亦如此)默认是调用运行 Ant 本身的 JVM 的编译器，然而如果你想要单独地调用编译器——比如你要使用一个高级别的 编译器——你可以使用 <javac fork="true"/>

3.21 制作文档（执行 javadoc）

```
<javadoc ... >
    <doclet name="theDoclet" path="path/to/theDoclet">
        <param name="-foo" value="foovalue"/>
        <param name="-bar" value="barvalue"/>
```

```
        </doclet>
    </javadoc>
```

3.22 打 jar 包

```
<target name="jar" depends="compile">
    <jar jarfile="${hello_jar}" basedir="${dest}" />
</target>
或者
<jar destfile="${dist}/lib/app.jar"
        basedir="${build}/classes"
        includes="mypackage/test/**"
        excludes="**/*Test*.class"
/>
```

3.23 运行 jar 包

（1）带参数运行

```
最简单的运行 jar 包
<target name="run" depends="jar">
    <java classname="test.ant.HelloWorld" classpath="${hello_jar}" />
</target>

传递参数运行
<java classname="test.Main">
    <arg value="-h"/>
    <classpath>
        <pathelement location="dist/test.jar"/>
        <pathelement
path="/Users/antoine/dev/asf/ant-core/bootstrap/lib/ant-launcher.jar />
    </classpath>
</java>
```

（2）设置运行的 JVM 再运行

```
<java classname="test.Main"
```

```
                dir="${exec.dir}"
                jar="${exec.dir}/dist/test.jar"
                fork="true"
                failonerror="true"
                maxmemory="128m" >
              <arg value="-h"/>
        <classpath>
            <pathelement location="dist/test.jar"/>
            <pathelement
path="/Users/antoine/dev/asf/ant-core/bootstrap/lib/ant-launcher.jar />
        </classpath>
</java>
```

3.24 运行 SQL

```
第一种：
    <target name="initDB1">
        <sql driver="com.mysql.jdbc.Driver"
            url="jdbc:mysql://localhost/login/"
            userid="root"
            password="" >
            <classpath refid="classpath" />
            insert into user values(10000, 'pzp', '123', 'pzpyp@163.com');
        </sql>
    </target>
第二种
    <target name="initDB2">
        <sql driver="com.mysql.jdbc.Driver"
            url="jdbc:mysql://localhost/"
            userid="root"
            password=""
            src="./mysql.sql">
            <classpath refid="classpath" />
        </sql>
    </target>

其中的 classpath 可以用如下方式替代
<classpath>
```

```
    <pathelement
location="c:\java\jdbc\mysql-connector-java-3.1.0-alpha-bin.jar"/>
</classpath>
```

3.25 发送邮件

```
    <mail mailhost="smtp.myisp.com" mailport="1025" subject="Test build">
    <from address="config@myisp.com"/>
    <replyto address="me@myisp.com"/>
    <to address="all@xyz.com"/>
    <message>The ${buildname} nightly build has completed</message>
    <attachments>
        <fileset dir="dist">
            <include name="**/*.zip"/>
        </fileset>
    </attachments>
</mail>
```

3.26 发布程序

如把程序部署到 web 服务器

```
    <property name="project.name" value="antdemo"/>
    <property name="webserver" value="C:/java/tomcat-6.0.18/webapps" />
    <!-- 发布项目（其实就是 WebContent 目录）到 web 容器下 -->
    <target name="deploy" depends="init,compile,test" description="将工程打
成 war 包">
        <echo message="deploy 项目到 Web 服务器" />
        <mkdir dir="${webserver}/${project.name}" />
        <copy todir="${webserver}/${project.name}" overwrite="true">
            <fileset dir="${WebContent.dir}" />
        </copy>
    </target>
```

3.27 Filter 的使用

（1）利用 Ant 提供的 filter 任务替换属性值。

现在根据不同环境的需要，对某些配置文件的值做一些替换。在 Ant 中，提供了 filter 1

使得替换值很方便。filter 主要用来在同一行内容中的替换，而正则表达式一下子可以替换多行内容。filter 的使用例子：

把所有的@aaaa@替换成 bbbb

```
<filter token="aaaa" value="bbbb"/>
    <copy todir="${dest.dir}" filtering="true">
        <fileset dir="${src.dir}"/>
    </copy>
</filter>
```

这段脚本的意思就是在 src.dir 目录下的所有文件中，如果有预先定义好的"@aaaa@"占位符的话，在拷贝到 dest.dir 目录后，所有的占位符都被替换成了"bbbb"。

替换属性文件中的内容

我们也可以将所有被替换的值放到某个属性文件中，filter 任务将属性文件中的每一个条目读出来并且设置成一个 Filter。如下所示：

```
<filter filtersfile="deploy_env.properties"/>
<copy todir="${dest.dir}" filtering="true">
    <fileset dir="${src.dir}"/>
</copy>
```

上面的脚本表示所有在 deploy_env.properties 中出现的条目将被作为一个 filter，在拷贝到 dest.dir 目录后，所有 src.dir 目录中存在的占位符将被替换成 deploy_env 中的值。

3.28 Length 的使用

把字符串"foo"的长度保存到属性"length.foo"中

```
<length string="foo" property="length.foo" />
```

把文件" bar"的长度保存到属性" length.bar "中

```
<length string="foo" property="length.foo" />
```

3.29 输入 input

提示消息"Please enter db-username,然后把用户的输入赋值给 db.user

如果没有输入则有默认值 Scott-Tiger

```
<input message="Please enter db-username:"
    addproperty="db.user"
```

```
        defaultvalue="Scott-Tiger"
    />
```

3.30 target 之间的调用

在一个 target 中调用另外一个 targetcall

```
（1）使用<ant target="***">调用另外一个 target
<target name="rerun" >
        <ant target="clean"  />
        <ant target="run"      />
</target>
可以写成 <target name="rerun"depends="clean,run" >

（2）使用 antcall 调用另外一个 target
<target name="default">
   <antcall target="isInvoked">
        <param name="param1" value="value"/>
        </antcall>
</target>
<target name=" isInvoked ">
        <echo message="param1=${param1}"/>
</target>
```

3.31 调用其他目录下的 ant.xml

调用目录 aaa/bbb/下的 build.xml

```
第一种写法：<ant antfile="aaa/bbb/build.xml" />
第二种写法：<ant antfile="bbb/build.xml" dir="aaa" />
第三种写法：<ant antfile="build.xml" dir="aaa/bbb" />
```

调用指定文件中的制定 target

```
<ant antfile="subproject/subbuild.xml" target="compile"/>
```

3.33 压缩、解压缩 zip 包

压缩 zip 文件、

```
<zip destfile="${dist}/manual.zip"
        basedir="htdocs/manual"
        includes="api/**/*.html"
        excludes="**/todo.html"
/>
```

解压缩 zip 文件

```
<unzip src="apache-ant-bin.zip" dest="${tools.home}">
    <patternset>
        <include name="apache-ant/lib/ant.jar"/>
    </patternset>
    <mapper type="flatten"/>
</unzip>
```

3.34 打 tar 包、解 tar 包

打 tar 包

```
<tar destfile="/Users/antoine/dev/asf/ant-core/docs.tar">
    <tarfileset dir="${dir.src}/docs"
                fullpath="/usr/doc/ant/README"
                preserveLeadingSlashes="true">
        <include name="readme.txt"/>
    </tarfileset>
    <tarfileset dir="${dir.src}/docs"
                prefix="/usr/doc/ant"
                preserveLeadingSlashes="true">
        <include name="*.html"/>
    </tarfileset>
</tar>
```

解 tar 包

```
<untar src="tools.tar" dest="${tools.home}"/>
```

3.35 打 war 包、解 war 包

打 war 包

```
<target name="war-test" depends="test">
    <delete dir="${buildWar}" />
    <mkdir dir="${buildWar}" />
    <war warfile="${buildWar}/${assembly.war}"
        webxml="${buildWebRoot}/WEB-INF/web.xml">
            <fileset dir="${buildWebRoot}" />
        </war>
</target>
```

解 war 包：我们首先得建立一个目录（这里是 unpack）用来存放解压后的文件。Ant 中提供了 unzip 命令用来解压 ear/war/jar 包。除了这个目录外，根据不同的目标环境，在运行目录下建立一个与目标环境相对应的目录，重新打好的 war 包就放在这个目录下，比如针对场景中的情况，如果需要创建一个产品环境下的部署包，我们可以建立一个 TestWebProduct 目录，目录名写在配置文件中（${pack.base.dir}）。

```
<target name="init">
    <echo>init in deploy</echo>
    <property file="deploy.properties" />
    <delete dir="${unpack.base.dir}" failonerror="false" />
    <delete dir="${pack.base.dir}" failonerror="false" />
    <mkdir dir="${unpack.base.dir}" />
    <mkdir dir="${pack.base.dir}" />
</target>

<target name="unpack">
    <echo>unpack the ${war.name}</echo>
    <copy file="${dest.dir}/${war.name}" todir="${unpack.base.dir}" />
    <unzip src="${unpack.base.dir}/${war.name}"
        dest="${unpack.base.dir}/${projectName}" />
    <delete file="${unpack.base.dir}/${war.name}" />
</target>
```

在 init 段中首先删除掉上次解压的目录和目标打包目录，然后重新建立目录；在 unpack 中，首先将编译好的默认 war 包拷贝至 unpack 定义的目录，解压之后把 unpack 下的 war 包删除。下面是这时候对应的属性文件。

3.36 打 ear 包

打 ear 包

```
<ear destfile="build/myapp.ear" appxml="src/metadata/application.xml">
    <fileset dir="build" includes="*.jar,*.war"/>
</ear>
```

3.37 Ant 控制流程（if…else…）

在 ant 中控制流程(if else)

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." name="learn" default="run">
    <property name="db.type" value="oracle" />
    <import file="properties.xml" />
    <taskdef resource="net/sf/antcontrib/antcontrib.properties"
classpath="${ant-contrib.jar}" />
    <target name="run">
        <if>
            <equals "${db.type}" arg2="mysql" />
            <then>
                <echo message="The db is mysql" />
            </then>
            <else>
                <echo message="the db is oralce" />
            </else>
        </if>
    </target>
</project>
```

4 实例分析

为了演示大部分 Ant target，笔者特地写了一个 web 工程，

放在本文档的同级目录下：**antdemo-web**

下面就是一个比较完整的 build.xml，大家可以做借签。

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="test" basedir="." default="deploywar">

    <!-- 引入外部的 properties 文件 -->
    <property file="build.properties" />

    <!-- 定义项目名称和 web 容器的部署路径 -->
    <property name="project.name" value="antdemo"/>
    <property name="webserver" value="C:/java/tomcat-6.0.18/webapps" />

    <!-- 源代码路径 -->
    <property name="src.dir" value="src" />

    <!-- 编译的目标路径 -->
    <property name="build.dir" value="./build" />
    <property name="build.classes.dir" value="./build/classes" />
    <!-- Junit report 路径 -->
    <property name="build.report.dir" value="${build.dir}/testReport" />

    <!-- WebContent 结构下的几个路径 -->
    <property name="WebContent.dir" value="./WebContent" />
    <property name="WebContent.webinf.dir"
value="${WebContent.dir}/WEB-INF" />
    <property name="WebContent.lib.dir"
value="${WebContent.dir}/WEB-INF/lib" />
    <property name="WebContent.classes.dir"
value="${WebContent.dir}/WEB-INF/classes" />

    <!-- 定义 classpath 路径，抽象定义，可以多处引用 -->
    <path id="classpath">
        <fileset dir="${WebContent.lib.dir}">
            <include name="**/*.jar" />
        </fileset>
```

```
        <pathelement location="${build.classes.dir}" />
        <pathelement location="${WebContent.classes.dir}" />
    </path>

    <!-- 初始化删除、创建 classes 路径 -->
    <target name="init">
        <echo message=" ====== 初始化创建编译目录 ======" />
        <delete dir="${build.dir}" />
        <mkdir dir="${build.dir}" />
        <mkdir dir="${build.classes.dir}" />
        <mkdir dir="${build.report.dir}" />
        <!-- WebContent 下的 classes -->
        <delete dir="${WebContent.classes.dir}" />
        <mkdir dir="${WebContent.classes.dir}" />
    </target>

    <!-- 编译 java 文件，编译到 build/classes 下，再把 build/classes 目录复制到 WebContent/WEB-INF/classes 下-->
    <target name="compile" depends="init" >
        <echo message=" ====== 编译源代码 ======" />
        <javac encoding="utf-8" srcdir="${src.dir}" destdir="${build.classes.dir}"
         debug="on" deprecation="false" optimize="true" failonerror="true" source="1.6" target="1.6">
            <classpath refid="classpath" />
        </javac>
        <!-- <javac srcdir="${src.dir}" destdir="${build.classes.dir}" classpathref="classpath" /> -->

        <copy todir="${WebContent.classes.dir}">
            <fileset dir="${build.classes.dir}">
                <exclude name="**/*Test*.*" />
            </fileset>
        </copy>
    </target>

    <!-- Junit 测试 -->
    <target name="test" depends="compile" description="run junit test">
        <echo message=" ====== Junit 测试，等待... ======" />
        <delete dir="${build.report.dir}" />
```

```
        <mkdir dir="${build.report.dir}" />
        <junit printsummary="on"
               haltonfailure="false"
               failureproperty="tests.failed"
               showoutput="true">

            <classpath refid="classpath" />
            <formatter type="plain" />
            <batchtest   fork="yes" todir="${build.report.dir}">
                <fileset dir="${build.classes.dir}">
                    <include name="**/*Test*.*" />
                    <include name="**/*test*.*" />
                </fileset>
            </batchtest>
        </junit>
        <fail if="tests.failed" />
    </target>

    <!-- 打 war 包 -->
    <target name="war" depends="test" description="将工程打成 war 包">
        <echo message=" ======  打 war 包  ======" />
        <delete file="${build.dir}/${project.name}.war"/>
        <war                                warfile="${build.dir}/${project.name}.war"
webxml="${WebContent.webinf.dir}/web.xml">
            <fileset dir="${WebContent.dir}" />
        </war>
    </target>

    <!-- 发布项目（其实就是 WebContent 目录）到 web 容器下 -->
    <target  name="deploy"  depends="init,compile,test"  description="将工程打
成 war 包">
        <echo message="deploy 项目到 Web 服务器" />
        <delete dir="${webserver}/${project.name}" />
        <mkdir dir="${webserver}/${project.name}" />
        <copy todir="${webserver}/${project.name}" overwrite="true">
            <fileset dir="${WebContent.dir}" />
        </copy>
    </target>
```

```
    <!-- 发布打成 war 包的项目到 web 容器下 -->
    <target name="deploywar" depends="init,compile,test,war" description="
将工程打成 war 包">
        <echo message="deploy 打成 war 包的项目到 Web 服务器" />
        <delete file="${webserver}/${project.name}.war" />
        <copy todir="${webserver}" overwrite="true">
            <fileset file="${build.dir}/${project.name}.war" />
        </copy>
    </target>

    <!-- 清除 -->
    <target name="delete" description="删除服务器上的 war">
        <delete dir="${build.classes.dir}"/>
        <delete dir="${build.report.dir}" />
        <delete file="${build.dir}/${project.name}.war"/>
        <delete dir="${webserver}/${project.name}"/>
        <delete file="${webserver}/${project.name}.war"/>
    </target>

    <!-- 直接使用 SQL 语句书此话数据库 -->
    <target name="initDB1">
        <sql driver="com.mysql.jdbc.Driver"
            url="jdbc:mysql://localhost/"
            userid="root"
            password="" >
            <classpath refid="classpath" />

            DROP database IF exists login1;
            CREATE database login1;

            use login1;
            create table user(
              user_id   int(11) auto_increment,
              user_name varchar(50) not null,
              user_pass varchar(50) not null,
              email varchar(100),
              primary key(user_id),
              unique(user_name)
            );
```

```
                insert into user values(10000, 'pzp', '123', 'pzpyp@163.com');
            </sql>
        </target>

        <!-- 使用.sql 脚本文件 初始化数据库 -->
        <target name="initDB2">
            <sql driver="com.mysql.jdbc.Driver"
                url="jdbc:mysql://localhost/"
                userid="root"
                password=""
                src="./mysql.sql">
                <classpath refid="classpath" />
            </sql>
        </target>
    </project>
```

跟以上脚本配套的实际项目放在，大家可以直接打开来查看

5 Ant 调用第三方软件的实例

5.1 Ant 调用 Junit 的实例

Ant集成junit，必须把junit.jar引入到classpath中。在命令行中就必须把junit.jar引入到ant/lib/下

```
<target name="test" depends="compile" description="run junit test">
        <echo message=" ====== Junit 测试，等待... ======" />
        <delete dir="${report.dir}" />
        <mkdir dir="${report.dir}" />

        <junit printsummary="on"
            haltonfailure="false"
            failureproperty="tests.failed"
            showoutput="true">

            <classpath refid="classpath" />
            <formatter type="plain" />
            <batchtest  fork="yes" todir="${report.dir}">
                <fileset dir="${buildWebRoot.classes}">
                    <include name="**/*Test*.*" />
                    <include name="**/*test*.*" />
                </fileset>
            </batchtest>
        </junit>
        <delete dir="${buildWebRoot}/repository" />
        <fail if="tests.failed">
        </fail>
    </target>
```

5.2 Ant 调用 SVN 的实例

（1）Ant 本身不支持 SVN，所以需要 svnant 任务插件支持，才能在 ant 脚本中获取 svn 中的内容，

Tigris.org 提供了 Subclipse，SvnClientAdapter 和 SvnAnt。网址 http://subclipse.tigris.org/index.html，

Subclipse 是一个 Eclipse 的插件，实现了 IDE 与 Subversion 得集成；

SvnClientAdapter 是 Subversion 的一套 Java API，封装了客户端对 Subversion 的一

操作；

SvnAnt 是用于访问 Subversion 的 Ant 任务，其依赖于 SvnClientAdapter。

（2）它的 URL 是：http://subclipse.tigris.org/svnant.html，这里使用的是 svnant-1.2.1 版本，使用前需要将它依赖的类库放在 ant 安装目录下面的 lib 目录；

（3）要想使用 svnant 中的任务访问 svn 的话，首先要求 svnant.jar 在系统 classpath 中

```
<typedef resource="org/tigris/subversion/svnant/svnantlib.xml"
classpath="svnant.jar">
    <classpath>
        <pathelement location="lib/svnant.jar"/>
    </classpath>
</typedef>
```

（4）由于是每日构建，需要每天将最新版本的程序更新下来，因此这里使用的是 **export** 任务，而不是 **checkout**。

```
<target name="export" depends="clean">
    <svn javahl="true" username="${svn.name}"
password="${svn.password}" failonerror="false">
        <export srcUrl="${svn.remote.url}" destPath="${svn.destpath}"
revision="HEAD" />
    </svn>
</target>
```

这里将**javahl**设为true是为了使用纯java实现的svn访问中间层，而不是使用svn的二进制命令行客户端。

需要注意的是，**export**命令也好，**checkout**命令也好，都不允许将本地目录和SVN的一个以上的url关联，也就是说无法将svn上的两个以上目录 的内容导出到本地的一个目录。

其实，ant 还可以集成 CVS，FTP，Axis 等等，就不一一举例了，请需要的时候去查找相关资料。

6 实践出真知

掌握了上面的那些内容后，应该知道如何去写一个好的 Ant build 了，但当你真的想去做的时候，你可能写不出好的 build.xml，因为你对 Ant 提供的默认命令知道得太少了。这个时候如果你想完成任务，并提高自己，有很多办法：

（1）很多开源的程序都带有 build.xml，看看它们如何写的

（2）Ant 的 document，里边详细列写了 Ant 的各种默认命令，及其丰富

（3）google，永远不要忘记它

OK，在这之后随着你写的 Ant build 越来越多，你知道的命令就越多，ant 在你的手里也就越来越强大了。这个是一个慢慢积累的过程。

7 QA

问题一：如果执行 Ant 过程中出现 Outofmemory 的错误怎么办？

答：加大内存，设置环境变量 ANT_OPTS=-Xms128m -Xmx256m

（1）在 ant.bat 文件中增加缓存，追加-Xms128m -Xmx256m 参数：

java -Xms128m -Xmx256m -classpath "%ANTCLASSPATH%" org.apache.tools.ant.Main %ANT_ARGS%

（2）使用 eclipse 运行 ant 时，应如下设定参数：

build.xml 文件右键-〉运行-〉外部工具-〉选择 JRE 页-〉VM 参数处设定 -Xms128m -Xmx256m 即可。

8 参考文献

1、http://www.apache.org

ant 官方手册，文中大量引用了 ant 官方手册中的例子

2、《Ant 权威指南》

作者: Jesse Tilly, Eric M. Burke 著, James Duncan Davidson 序，林琪 译
出版: 2003 年 7 月