本文以最新发布的 Ant 1.5.1 为例，介绍这款优秀的 Build 工具的安装配置、基本应用和一些高级话题。最新的 Ant 下载地址是 http://jakarta.apache.org/ant/。

Ant 是一种基于 Java 的 Build 工具。理论上来说，它有些类似于 C 中的 make，但比 make 优越。现在存在的大多数 Build 工具，如 make、gnumake、nmake、jam 等都存在这样或那样的不足，比如依赖于特定的平台、配置文件过于复杂或者对格式无法检查而容易出错等。与这些工具相比较，Ant 的两个特性决定了它是一款优秀的 Build 工具：

1. 基于 Java 的实现。具有良好的跨平台性，同时可以通过增加新的 Java 类来扩展 Ant 的功能，而无需去了解不同平台上不同的脚本语言。

2.基于 XML 的配置文件。Ant 以 XML 树来描述 Target/Task 的关系，文件结构清晰、易读易写，并且利用 XML 对格式的控制来避免由于配置文件的错误造成的 Build 操作失败。

安装与配置

Ant 的安装非常简单，把从网上下载的 jakarta-ant-1.5.1-bin.zip 解开到一个目录下即可（以下假定安装在目录 D:\jakarta-ant-1.5.1）。接下来需要进行环境变量配置：

```
SET ANT_HOME=D:\jakarta-ant-1.5.1 //注意是 Ant 的安装目录，不是 bin 子目录
SET PATH=%PATH%;%ANT_HOME%\bin;
```

在配置环境变量之前，请确认已经正确设置了 JAVA_HOME 系统变量。输入 ant 命令，看到如下输出说明已成功安装了 Ant 工具：

```
Buildfile: build.xml does not exist!
Build failed
```

提示信息表明在当前目录不存在 build.xml 配置文件，但这本身已经说明 Ant 成功运行了。

快速入门

下面用一个最简单也是最经典的例子-HelloWorld 来感受一下 Ant 吧。

```
//HelloWorld.java
package com.sharetop.antdemo;
public class HelloWorld  {
 public static void main( String args[] ) {
  System.out.println("Hello world. ");
 }
}
```

要让 Ant 编译这个文件，首先需要编写一个 Build 配置文件。在一般情况下，这个文件被命名为 build.xml。

```
<?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="${dest}"/>
</target>
<target name="build" depends="compile">
<jar jarfile="${hello_jar}" basedir="${dest}"/>
</target>
<target name="run" depends="build">
<java classname="com.sharetop.antdemo.HelloWorld"
classpath="${hello_jar}"/>
</target>
</project>
```

来看一下这个文件的内容，它描述了以下信息：工程的名字为 HelloWorld，工程有四个 target，分别是 init、compil、build 和 run，缺省是 run。compile 只有一个任务 javac，源文件位于 src 目录下，输出的类文件要放在 classes 目录下。build 的任务是 jar，生成的 jar 文件为 hello.jar，它打包时以 classes 为根目录。而 run 则是执行这个 HelloWorld 类，用 hello.jar 作为 classpath。这四个 target 之间有一个依赖关系，这种关系用 depends 来指定。即如果 Target A 依赖于 Target B，那么在执行 Target A 之前会首先执行 Target B。所以从下面运行缺省 Target（run）的输出看，这四个 Target 的执行顺序是：

init→compile→build→run。文件目录结构如图 1 所示。HelloWorld.java 文件在 src\com\sharetop\antdemo 子目录下。

图 1 ant_demo 应用的目录结构

在命令行输入命令：ant，然后运行，可以看到如下输出：

如果配置文件名不是 build.xml，比如是 build_front.xml，那么，可以使用-buildfile 命令参数指定：

```
G:\myDoc\ant_demo>ant -buildfile build_front.xml
```

也可以单独执行指定的某个 target，比如，只编译不打包执行，可以使用下面输入命令即可：

```
G:\myDoc\ant_demo>ant compile
```

在相应的目录下会找到编译出的 HelloWorld.class 文件。

再看看上面的 build.xml 配置文件，文件开头定义了 3 个属性，分别指定了源文件输出路径、类文件输出路径和生成的 Jar 文件名，后面对这些路径的引用都通过一个${property name}来引用。所以，要注意这样一个原则“ 目录的定义与目录的引用应该分开 ”。

基本应用

建立工程的目录

一般要根据工程的实际情况来建立工程的目录结构。但是，有一些比较通用的组织形式可供参考，比如所有的 jakarta 项目都使用类似的目录结构。下面让我们来看一下这种目录结构的特点。

表 1

目录	文件
bin	公共的二进制文件，以及运行脚本
build	临时创建的文件，如类文件等
dist	目标输出文件，如生成 Jar 文件等。
doc/javadocs	文档。
lib	需要导出的 Java 包
src	源文件

对于一个简单的工程，一般包括表 1 的几个目录。其中 bin、lib、doc 和 src 目录需要在 CVS 的控制之下。当然在这样的目录结构上，也可以做一些调整，例如，可以建立一个 extra 目录来放置需要发布的 Jar 文件、Inf 文件及图像文件等。同样，如果开发 Web 应用可以建立一个 Web 目录放置 JSP、HTML 等文件。

如果我们开发的是一个比较复杂的项目，包括多个子项目，并且各个子项目是由不同的开发人员来完成的，那么要如何来设计它的目录结构？首先有一点是需要确定的，不同的子项目应该拥有不同的 Build 文件，并且整个项目也应该有一个总的 Build 文件。可以通过 Ant 任务或是 AntCall 任务调用子项目的 Build 文件，如下例：

```
<target name="core" depends="init">
<ant dir="components" target="core"/>
<ant dir="waf/src" target="core"/>
<ant dir="apps" target="core"/>
</target>
```

在各个子项目的耦合不是非常紧密的情况下，各个子项目应该有各自独立的目录结构，也就是说它们可以有自己的 src、doc、build、dist 等目录及自己的 build.xml 文件，但是可以共享 lib 和 bin 目录。而对于那些耦合紧密的子项目，则推荐使用同一个 src 目录，但是不同的子项目有不同的子目录，各个子项目的 build.xml 文件可以放在根目录下，也可以移到各个子项目的目录下。

编写 Build 文件

要用好 Ant 工具，关键是要编写一个 build.xml 文件。要编写出一个结构良好、灵活可扩展的 Build 文件，有两个问题要考虑，一是了解 Build 文件的基本结构，二是了解 Ant 定义的大量任务。

Ant 的 Build 文件是一个标准的 XML 文件，它包含一个根节点 Project，每个 Project 定义了至少一个或多个 Target，每个 Target 又是一系列 Task 的集合。它们之间的关系如图 2 所示。

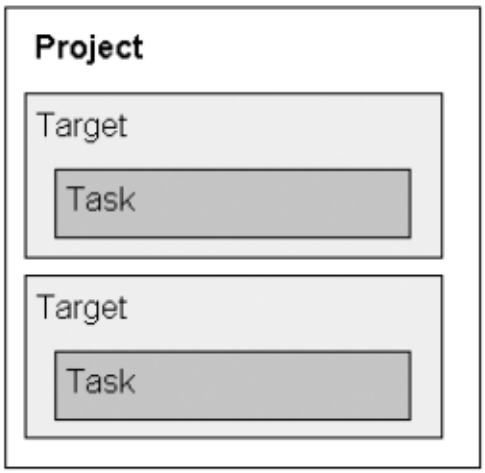

图 2 build.xml 文件的结构

每个 Task 是一段可被执行的代码，比如，前例中的 javac、jar 就是两个最常用的 Task。Ant 定义了大量的核心 Task，我们要考虑的第二个问题正是如何去掌握这大量的 Task。其实唯一的方法就是边学习边实践，这方面最好的参考就是官方的 Ant 使用手册。

外部文件的使用

使用外部的 Property 文件可以保存一些预设置的公共属性变量。这些属性可以在多个不同的 Build 文件中使用。

可以将一个外部的 XML 文件导入 Build 文件中，这样多个项目的开发者可以通过引用来共享

一些代码，同样，这也有助于 Build 文件的重用，示例代码如下所示：

```
<!DOCTYPE project [
<!ENTITY share-variable SYSTEM "file:../share-variable.xml">
<!ENTITY build-share SYSTEM "file:../build-share.xml">
]>
<project name="main" default="complie" basedir=".">
&share-variable;
&build-share;
... ...
```

在 J2EE 项目中的应用

只要掌握了 Ant 的使用方法,在 J2EE 项目中的应用与在其它项目中的应用并没有太大的不同，但是仍有几点是需要注意的。

一是要清楚 War 和 Jar 文件的目录结构，主要是 War 的配置文件 web.xml 文件的位置和 EJB 的配置文件（ejb-jar.xml 和 weblogic-ejb-jar.xml 等）的位置，在调用 Jar 任务打包文件时一定要记得把它们也包含进来。一般在编译之前就要注意把这些需打包的文件拷入相应目录下。二是在 J2EE 项目中可能会涉及到一些特殊的任务，比如在 Weblogic 中会调用 ejbc 预编译 EJB 的代码存根，或者需要在 Ant 中同时发布 Jar 到相应的服务器中等。可以用两种途径实现这些任务,一是扩展 Ant 任务实现这些任务,二是直接用 Java 任务来执行这些命令。下面是打包、发布一个 EJB 的 build.xml 配置文件片断，代码如下：

```
<target name="deploy_HelloEJB" depends="compile">
<delete dir="${temp}/ejb_make"/>  <!-- 首先删除临时目录 -->
<delete file="${temp}/helloEJB.jar"/>
<!-- 删除 WebLogic 域中老版本的 EJB -->
<delete file="${weblogic.deploy.dest}/helloEJB.jar"/>
<!-- 创建 META-INF 目录，放置 ejb-jar.xml 和 weblogic-ejb-jar.xml -->
<mkdir dir="${temp}/ejb_make/META-INF"/>
<!-- 拷贝 ejb-jar.xml 和 weblogic-ejb-jar.xml 到临时目录-->
<copy todir="${temp}/ejb_make/META-INF">
<fileset dir="etc/baseinfo">
<include name="*.xml"/>
</fileset>
</copy>
<!-- 拷贝所有的 helloEJB 类到临时目录 -->
<copy todir="${temp}/ejb_make/">
```

```
<fileset dir="${dest.classes}/">  <!-- dest.classes 是输出的类文件目
录 -->
<include name="${dest.classes}/helloEJB/**"/>
</fileset>
</copy>
<!-- 将所有这些文件打包成 helloEJB.jar -->
<jar jarfile="${temp}/helloEJB.jar" basedir="${temp}/ejb_make"/>
<!-- 进行 weblogic.ejbc 编译 -->
<java classpath="${wl_cp}" classname="weblogic.ejbc" fork="yes" >
<classpath>
<fileset dir="lib">
<include name="*.jar" />
</fileset>
</classpath>
<arg value="${temp}/helloEJB.jar" />
<arg value="${temp}/helloEJB_deploy.jar" />
</java>
<!-- 拷贝/发布到 WebLogic 的{DOMAIN}\applications 目录 -->
<copy file="${temp}/helloEJB_deploy.jar"
todir="${weblogic.deploy.dest}"/>
</target>
```

用 Ant 配合 JUnit 实现单元测试

Ant 提供了 JUnit 任务,可以执行单元测试代码。如何使用 JUnit，以及如何编写测试用例(TestCase)，感兴趣的读者可以参阅 JUnit 的相关文档。在 Ant 中使用 JUnit 的方法非常简单，首先需要把 junit.jar 拷入 ANT_HOME\lib 下，确认在这个目录下有 optional.jar，因为 JUnit 是 Ant 的扩展任务，需要引用这个扩展包。然后就是在 Build 文件中加入 JUnit 的任务，代码如下：

```
<target name="run" depends="client">
<junit printsummary="yes" fork="yes" haltonfailure="yes">
<classpath>
<pathelement location="client.jar" />
</classpath>
<formatter type="plain" />
<test name="com.sharetop.antdemo.HelloWorldTest" />
</junit>
</target>
```

高级话题

为 Ant 开发扩展任务

为 Ant 实现扩展任务其实是非常容易的，只需按照以下几个步骤即可：

1. 创建一个 Java 类继承 org.apache.tools.ant.Task 类；

2. 对每个属性实现 set 方法。Ant 会根据需要自动完成类型转换；

3. 如果扩展的任务需要嵌套其它的 Task，那么这个 Java 类必需实现接口 org.apache.tools.ant.TaskContainer；

4. 如果扩展的任务要支持 Text，需要增加一个方法 void addText(String)；

5. 对每个嵌套的元素，实现 create、add 或 addConfigured 方法；

6. 实现 public void execute 方法；

7. 在 build.xml 文件中使用<taskdef>来引用自定义的 Task。

下面以一个简单的例子来说明如何为 Ant 增加一个 hello 任务，它可以连续打印多条信息，打印的次数由属性 count 指定，而打印的内容则由它内嵌的一个 helloinfo 任务的 message 属性指定，看上去这非常类似 JSP 中自定义标签的一些概念，实现代码如下：

```
//HelloInfoTask.java
package com.sharetop.antdemo;
import org.apache.tools.ant.*;
public class HelloInfoTask {
 private String msg;
 public void execute() throws BuildException {
  System.out.println(msg);
 }
 public void setMessage(String msg) {
  this.msg = msg;
 }
}
```

下面是外部 Task 类的代码：

```
//HelloTask.java
package com.sharetop.antdemo;
import org.apache.tools.ant.*;
public class HelloTask extends Task implements
org.apache.tools.ant.TaskContainer
{
 private Task info;
 private int count;
 public void execute() throws BuildException {
  for(int i=0;i<count;i++)
  info.execute();
 }
 public void setCount(int c){
  this.count=c;
 }
 public void addTask(Task t){
  this.info=t;
 }
}
```

实现了这两个 Task，在 build.xml 文件中定义它的 task name，就可以在 Target 中执行它了。如果你不想使用<taskdef>标签来定义 Task，也可以通过修改 default.properties 文件来实现引入新 Task，这个文件位于 org.apache.tools.ant.taskdefs 包里。下例是一个使用标签来引入新 Task 的 Build 文件部分：

```
<target name="hello" depends="client">
<taskdef name="hello"
classname="com.sharetop.antdemo.HelloTask" classpath="client.jar"/>
<taskdef name="helloinfo"
classname="com.sharetop.antdemo.HelloInfoTask"
classpath="client.jar"/>
<hello count="3" >
<helloinfo message="hello world" />
</hello>
</target>
```

在自己的程序中调用 Ant

Ant 的任务其实就是一段功能代码。Ant 内置的大量任务对于我们开发 Java 应用具有非常大

的意义，为什么我们不能直接使用呢？

因为尽管在程序中调用 Ant 的任务并不复杂，而且我们知道 Ant 的任务其实都是一些 Java 类，调用的方法无非就是执行这些类而已，不过在执行之前需要对它做一些初始化的工作，所以我们需要引用一个 Task 类的子类来实现这个功能，比如如下代码：

```
package com.sharetop.antdemo;
import org.apache.tools.ant.*;
import org.apache.tools.ant.taskdefs.*;
import java.io.File;
public class RunAntTask {
 public RunAntTask() {
 }
 public static void main(String args[]){
  AntJAR j = new AntJAR();
  j.setBasedir(new File("./classes"));
  j.setJarfile(new File("aaa.jar"));
  j.execute();
 }
}
final class AntJAR extends Jar {
 public AntJAR() {
  project = new Project();
  project.init();
  taskType = "jar";
  taskName = "jar";
 }
}
```

注意 AntJAR 类的构造方法，先创建了 Project 并初始化它，这是直接调用 Task 的必需条件。

如果要在自己的程序中执行 Ant，需要了解的是 Ant 定义的几个 BuildEvent，它包括：

◆ Build started

◆ Build finished

◆ Target started

◆ Target finished

◆ Task started

◆ Task finished

◆ Message logged

我们需要做的是实现 BuildListener 接口来处理各种事件,而执行 Ant 的方法与上面给的例子非常类似，以实际开发的 AntBuilder 软件的部分代码为例：

```
public void buildTarget(String targetName,String buildFileName) {
 try {
  Project p = new Project();
  p.init();
  File f = new File(buildFileName);
  p.setUserProperty("ant.file",f.getAbsolutePath());
  ProjectHelper.configureProject(p,f);
  p.addBuildListener(this);
  if( targetName==null )
  p.executeTarget(p.getDefaultTarget());
  else
  p.executeTarget(targetName);
 }
 catch (Exception ex) {
  jTextArea1.append(ex.getMessage());
 }
}
```

创建 Project 并初始化，设置它的配置文件（build.xml），执行它缺省的或指定的 Target，然后在实现了 BuildListenser 接口的监听器类中对你感兴趣的事件作处理，代码如下：

```
public void buildStarted(BuildEvent event){ /* nothing*/ }
  public void buildFinished(BuildEvent event) { /* nothing*/ }
  public void targetStarted(BuildEvent event) {
    this.jTextArea1.append(event.getTarget().getName()+": \n\r");
  }
  public void targetFinished(BuildEvent event) {/* nothing*/ }
  public void taskStarted(BuildEvent event) {/* nothing*/ }
  public void taskFinished(BuildEvent event) { /* nothing*/ }
  public void messageLogged(BuildEvent event) {
   int prior = event.getPriority();
```

```
   switch(prior){
    case Project.MSG_ERR :
    this.jTextArea1.append("["+event.getTask().getTaskName()+"]Err:"
   +event.getMessage());
     break;
    case Project.MSG_INFO:

this.jTextArea1.append("["+event.getTask().getTaskName()+"]"+event.getMessage
 ());
     break;
    case Project.MSG_WARN:
    this.jTextArea1.append("["+event.getTask().getTaskName()+"]"
   +event.getMessage());
     break;
    case Project.MSG_VERBOSE:
     this.jTextArea1.append(event.getMessage());
     break;
   }
  }
```

Build.xml 文件的写法每个公司都有不同，这里没有太大的参考价值，所以略去。