**Ant** **sprint**

# User Case Sprint

## Sprint1 –helloworld2

File location: C:\working\ant\testant

Step1, HelloWorld2.java

package test.ant;

public class HelloWorld2{

public static void main(String[] args){

System.out.println("Hello World");

}

}

Step2:

Build2.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="hello1.jar"/>

<target name="init">

<mkdir dir="${dest}"/>

</target>

<target name="compile" depends="init">

<javac srcdir="${src}" destdir="${dest}" includeantruntime="on" />

</target>

<target name="build" depends="compile">

<jar jarfile="${hello\_jar}" basedir="${dest}"/>

</target>

<target name="run" depends="build">

<java classname="test.ant.HelloWorld2" 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>

Step3: 新建一个src的文件夹，然后把HelloWorld2.java按照包目录放进去

Step4: 在命令行下键入 ant –f build2.xml

你会发现一个个任务都完成了。每次更改完代码只需要再次键入ant

Step5: 有的时候我们可能并不想运行程序，只想执行这些步骤中的某一两个步骤，例如我只想重新部署而不想运行，键入  ant –f build2.xml build

Step6: ant中的每一个任务都可以这样调用ant + target name

Ex. ant –f build2.xml clean

# My Activities:

## 下载地址：<http://ant.apache.org/>，在本文中下载的是1.7.0版本。解压到某个目录（例如E:"apache-ant-1.7.0）

1. 一般情况下，大多数软件公司做开发的时候都不用myeclipse开发，这是利用ant部署就给我们带来极大的方便，它先将你的project打包成war包，然后部署到指定的服务器中。

# Diagram

## compile: [mkdir] Created dir: C:\working\ant\antstudy\build\classes [javac] C:\working\ant\antstudy\build.xml:8: warning: 'includeantruntime' was not set defaulting to build.sysclasspath=last; set to false for repeatable builds

# All excerpt

Ant的概念  
 当一个代码项目大了以后，每次重新编译，打包，测试等都会变得非常复杂而且重复，因此c语言中有make脚本来帮助这些工作的批量完成。在Java 中应用是平台无关性的，当然不会用平台相关的make脚本来完成这些批处理任务了，ANT本身就是这样一个流程脚本引擎，用于自动化调用程序完成项目的编译，打包，测试等。除了基于JAVA是平台无关的外，脚本的格式是基于[XML](https://www.baidu.com/s?wd=XML&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y4rymkujcsPH-bnvuhn1030ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnW0dnWcdn1D4nWb4PHTzrHRYPs)的，比make脚本来说还要好维护一些。  
 每个ant脚本（缺省叫build.[xml](https://www.baidu.com/s?wd=xml&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y4rymkujcsPH-bnvuhn1030ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnW0dnWcdn1D4nWb4PHTzrHRYPs)）中设置了一系列任务(target)：比如对于一个一般的项目可能需要有以下任务。  
 \* 任务1：usage 打印本脚本的帮助信息（缺省）  
 \* 任务2：clean <-- init 清空初始化环境  
 \* 任务3：javadoc <-- build <-- init 生成JAVADOC  
 \* 任务4：jar <-- build <-- init 生成JAR  
 \* 任务5：all <-- jar + javadoc <-- build <-- init 完成以上所有任务：jar javadoc   
 而多个任务之间往往又包含了一定了依赖关系：比如把整个应用打包任务(jar)的这个依赖于编译任务(build)，而编译任务又依赖于整个环境初始化任务(init)等。  
 注：我看到很多项目的ant脚本中的命名基本上都是一致的，比如：编译一般叫build或者compile；打包一般叫jar或war；生成文档一般命名为 javadoc或javadocs；执行全部任务all。在每个任务的中，ANT会根据配置调用一些外部应用并配以相应参数执行。虽然ANT可调用的外部应用种类非常丰富，但其实最常用的就2，3个：比如javac javadoc jar等。  
  
Ant的优点  
 Ant是[Apache软件基金会](https://www.baidu.com/s?wd=Apache%E8%BD%AF%E4%BB%B6%E5%9F%BA%E9%87%91%E4%BC%9A&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y4rymkujcsPH-bnvuhn1030ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnW0dnWcdn1D4nWb4PHTzrHRYPs)[JAKARTA](https://www.baidu.com/s?wd=JAKARTA&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y4rymkujcsPH-bnvuhn1030ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnW0dnWcdn1D4nWb4PHTzrHRYPs)目录中的一个子项目，它有以下的优点。  
 跨平台性。Ant是纯Java语言编写的，所示具有很好的跨平台性。  
 操作简单。Ant是由一个内置任务和可选任务组成的。Ant运行时需要一个[XML](https://www.baidu.com/s?wd=XML&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y4rymkujcsPH-bnvuhn1030ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnW0dnWcdn1D4nWb4PHTzrHRYPs)文件(构建文件)。  
 Ant通过调用target树，就可以执行各种task。每个task实现了特定接口对象。由于Ant构建文件时XML格式的文件，所以和容易维护和书写，而且结构很清晰。  
 Ant可以集成到开发环境中。由于Ant的跨平台性和操作简单的特点，它很容易集成到一些开发环境中去。

[**用Ant实现Java项目的自动构建和部署**](http://www.blogjava.net/amigoxie/archive/2007/11/09/159413.html)

        原文地址：<http://tech.it168.com/j/2007-11-09/200711091344781.shtml>  
        本文请勿转载！

       Ant是一个Apache基金会下的跨平台的构件工具，它可以实现项目的自动构建和部署等功能。在本文中，主要让读者熟悉怎样将Ant应用到Java项目中，让它简化构建和部署操作。

**一．**             **安装与配置**

下载地址：<http://ant.apache.org/>，在本文中下载的是1.7.0版本。解压到某个目录（例如E:"apache-ant-1.7.0），即可使用。

添加系统环境变量：ANT\_HOME，该变量指向Ant解压后的根目录，在此为E:"apache-ant-1.7.0。

安装与配置完毕后，读者可以测试一下Ant是否可用，首先进入Ant的bin目录，运行命令ant –version，若安装和配置成功，则会显示Ant版本信息，如下图所示：  
    

由上可以看出，读者运行Ant的命令时，需要进入到Ant的bin目录，如何才能让系统自动找到Ant呢？这时需要读者在系统环境变量path中添加Ant的bin目录。设置完成后，我们就可以在任何目录（例如C:"Documents and Settings"AmigoXie目录）输入Ant的命令，来获得命令的运行结果。

**二．**             **Ant的关键元素**

Ant的构件文件是基于XML编写的，默认名称为build.xml。为了更清楚的了解Ant，在这里编写一个简单的Ant程序，用来展现Ant的功能，让读者对Ant有一个初步的了解。首先在E盘下建立一个build.xml文件，内容如下：

http://www.blogjava.net/Images/OutliningIndicators/None.gif<?xml version="1.0"?>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif<project name="helloWorld">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif       <target name="sayHelloWorld">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif              <echo message="Hello,Amigo"/>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif       </target>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif</project>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif

读者可以进入E盘，然后运行ant sayHelloWorld，可以看到如下的运行结果：  
    

其中sayHelloWorld为需要执行的任务的名称。如果文件名不为build.xml，而为hello.xml时，读者运行同样的命令时，命令窗口会出现如下错误：

Buildfile: build.xml does not exist!

Build failed

由上面的命令的错误提示可以看出，ant命令默认寻找build.xml文件。若文件名为hello.xml时，读者还需要对命令做少许改变，改为：ant –f hello.xml sayHelloWorld、ant –buildfile hello.xml sayHelloWorld或ant –file hello.xml sayHelloWorld。

接下来开始向读者讲解本节的重点：Ant的关键元素project、target、property和task。

**1.**       **project元素**

project元素是Ant构件文件的根元素，Ant构件文件至少应该包含一个project元素，否则会发生错误。在每个project元素下，可包含多个target元素。接下来向读者展示一下project元素的各属性。

**1）name属性**

用于指定project元素的名称。

**2）default属性**

用于指定project默认执行时所执行的target的名称。

**3）basedir属性**

用于指定基路径的位置。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。

下面给读者一个简单的例子来展示project的各元素的使用。修改E:"build.xml文件，修改后的内容如下：

http://www.blogjava.net/Images/OutliningIndicators/None.gif<?xml version="1.0"?>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif<project name="projectStudy" default="sayBaseDir" basedir="E:"apache-ant-1.7.0">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif       <target name="sayBaseDir">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif              <echo message="The base dir is: ${basedir}"/>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif       </target>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif</project>

从上面的内容我们可以看出，在这里定义了default属性的值为sayBaseDir，即当运行ant命令时，若未指明执行的target时，默认执行的target的sayBaseDir，同时还定义了basedir属性的值为E:"apache-ant-1.7.0，进入E盘后运行ant命令，可看到运行的结果，如下图所示：  
     

因为设定了basedir的值，所以basedir属性的值变成了读者设置的值。读者可以自行将project元素的basedir属性去掉后运行ant看看运行结果，此时basedir的值变成了E:"，即为Ant构件文件的父目录。

有的时候，读者可能有这种需求，即想得到某个project下所有的target的名称，读者可以通过在ant命令里加上-proecthelp来达到该目的。例如针对上述的例子我们运行ant –projecthelp，输出结果如下：

Buildfile: build.xml

Main targets:

Other targets:

 sayBaseDir

Default target: sayBaseDir

**2.**       **target元素**

它为Ant的基本执行单元，它可以包含一个或多个具体的任务。多个target可以存在相互依赖关系。它有如下属性：

**1）name属性**

指定target元素的名称，这个属性在一个project元素中是唯一的。我们可以通过指定target元素的名称来指定某个target。

**2）depends属性**

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。

**3）if属性**

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。

**4）unless属性**

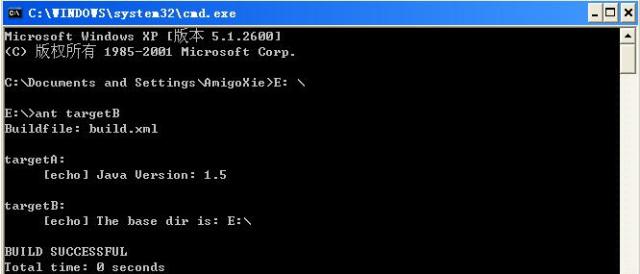
该属性的功能与if属性的功能正好相反，它也用于验证指定的属性是否存在，若不存在，所在target将会被执行。

**5）description属性**

该属性是关于target功能的简短描述和说明。

下面带领读者来看一个各属性综合使用的例子。修改E:"build.xml文件，修改后的内容如下：

http://www.blogjava.net/Images/OutliningIndicators/None.gif<?xml version="1.0"?>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif<project name="targetStudy">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif       <target name="targetA" if="ant.java.version">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif              <echo message="Java Version: ${ant.java.version}"/>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif       </target>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif       <target name="targetB" depends="targetA" unless="amigo">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif              <description>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif                            a depend example!  
http://www.blogjava.net/Images/OutliningIndicators/None.gif              </description>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif              <echo message="The base dir is: ${basedir}"/>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif       </target>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif</project>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif

进入E盘后运行ant targetB，可看到如下图所示的运行结果：  
    

读者分析结果后可以看到，我们运行的是名为targetB的target，因该target依赖于targetA，所以targetA将首先被执行，同时因为系统安装了java环境，所以ant.java.version属性存在，执行了targetA这个target，输出信息：[echo] Java Version: 1.5，targetA执行完毕后，接着执行targetB，因为amigo不存在，而unless属性是在不存在时进入所在的target的，由此可知targetB得以执行，输出信息：The base dir is: E:"。

**3.**       **property元素**

该元素可看作参量或者参数的定义，project的属性可以通过property元素来设定，也可在Ant之外设定。若要在外部引入某文件，例如build.properties文件，可以通过如下内容将其引入：<property file=” build.properties”/>

property元素可用作task的属性值。在task中是通过将属性名放在“${”和“}”之间，并放在task属性值的位置来实现的。

Ant提供了一些内置的属性，它能得到的系统属性的列表与Java文档中System.getPropertis()方法得到的属性一致，这些系统属性可参考sun网站的说明。

同时，Ant还提供了一些它自己的内置属性，如下：

basedir：project基目录的绝对路径，该属性在讲解project元素时有详细说明，不再赘述；

ant.file：buildfile的绝对路径，如上面的各例子中，ant.file的值为E:"build.xml；

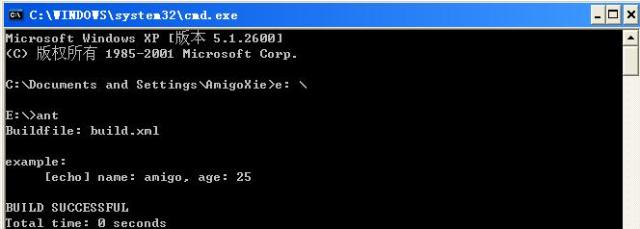
ant.version：Ant的版本，在本文中，值为1.7.0；

ant.project.name：当前指定的project的名字，即前文说到的project的name属性的值；

ant.java.version：Ant检测到的JDK的版本，在上例运行结果中可看到为1.5。

下面让读者来看一个property元素使用的简单例子。修改E:"build.xml文件，内容如下：

http://www.blogjava.net/Images/OutliningIndicators/None.gif<?xml version="1.0"?>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif<project name="propertyStudy" default="example">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif<property name="name" value="amigo"/>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif<property name="age" value="25"/>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif<target name="example">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif        <echo message="name: ${name}, age: ${age}"/>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif</target>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif</project>

该例的运行结果如下图所示：  
    

由此读者可以看出，通过如下两个语句：

<property name="name" value="amigo"/>

<property name="age" value="25"/>

我们设置了名为name和age的两个属性，这两个属性设置后，读者在下文中可以通过${name}和${age}分别取得这两个属性的值。

**三．**             **Ant的常用任务**

在Ant工具中每一个任务封装了具体要执行的功能，是Ant工具的基本执行单位。在本小节中，主要引导读者来看下Ant的常用任务及其使用举例。

**1.**       **copy任务**

该任务主要用来对文件和目录的复制功能。举例如下：

Eg1.复制单个文件：<copy file="file.txt" tofile="copy.txt"/>

Eg2.对文件目录进行复制：

   <copy todir="../newdir/dest\_dir">

            <fileset dir="src\_dir"/>

 </copy>

Eg3. 将文件复制到另外的目录：

 <copy file="file.txt" todir="../other/dir"/>

**2.**       **delete任务**

对文件或目录进行删除，举例如下：

Eg1. 删除某个文件：<delete file="photo/amigo.jpg"/>

Eg2. 删除某个目录：<delete dir="photo"/>

Eg3. 删除所有的备份目录或空目录：

        <delete includeEmptyDirs="true">

               <fileset dir="." includes="\*\*/\*.bak"/>

        </delete>

**3.**       **mkdir任务**

创建目录。eg：<mkdir dir="build"/>

**4.**       **move任务**

移动文件或目录，举例如下：

Eg1. 移动单个文件：<move file="fromfile" tofile=”tofile”/>

Eg2. 移动单个文件到另一个目录：<move file="fromfile" todir=”movedir”/>

Eg3. 移动某个目录到另一个目录：

        <move todir="newdir">

               <fileset dir="olddir"/>

        </move>

**5.**       **echo任务**

该任务的作用是根据日志或监控器的级别输出信息。它包括message、file、append和level四个属性，举例如下：

<echo message="Hello,Amigo" file="logs/system.log" append="true">

**四．**             **利用Ant构建和部署Java工程**

Ant可以代替使用javac、java和jar等命令来执行java操作，从而达到轻松的构建和部署Java工程的目的。下面来看几个知识点。

**1.**       **利用Ant的javac任务来编译java程序**

Ant的javac任务用于实现编译Java程序的功能。下面来看一个简单的例子：

首先我们建立名为antstudy的Java工程，建立src目录为源代码目录，在src目录下建立HelloWorld.java这个类文件。该类文件的内容如下：

http://www.blogjava.net/Images/OutliningIndicators/ExpandedBlockStart.gifpublic class HelloWorld {  
http://www.blogjava.net/Images/OutliningIndicators/ExpandedSubBlockStart.gif    public static void main(String[] args) {  
http://www.blogjava.net/Images/OutliningIndicators/InBlock.gif       System.out.println("Hello,Amigo");  
http://www.blogjava.net/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    }  
http://www.blogjava.net/Images/OutliningIndicators/ExpandedBlockEnd.gif}

   同时在antstudy工程的根目录下建立build.xml文件，在该文件中编译src目录下的java文件，并将编译后的class文件放入build/classes目录中，在编译前，需清除classes目录，该文件的内容如下：

http://www.blogjava.net/Images/OutliningIndicators/None.gif<?xml version="1.0"?>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif<project name="javacTest" default="compile" basedir=".">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    <target name="clean">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif       <delete dir="build"/>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    </target>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    <target name="compile" depends="clean">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif       <mkdir dir="build/classes"/>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    <javac srcdir="src" destdir="build/classes"/>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    </target>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif</project>

    运行该build.xml文件，可在工程中看到新增了build/classes目录，并在该目录中生成了编译后的HelloWorld.class文件。

**2.**       **使用Ant的java任务运行Java程序**

Ant中可以使用java任务实现运行Java程序的功能。下面在1的例子中进行如下的修改，修改后的build.xml文件的内容如下：

http://www.blogjava.net/Images/OutliningIndicators/None.gif<?xml version="1.0"?>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif<project name="javaTest" default="jar" basedir=".">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    <target name="clean">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif       <delete dir="build"/>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    </target>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    <target name="compile" depends="clean">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif       <mkdir dir="build/classes"/>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif      <javac srcdir="src" destdir="build/classes"/>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    </target>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    <target name="run" depends="compile">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif       <java classname="HelloWorld">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif           <classpath>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif              <pathelement path="build/classes"/>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif           </classpath>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif       </java>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    </target>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif</project>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif

    运行该build.xml文件，可在控制台看到HelloWorld的main方法的输出。

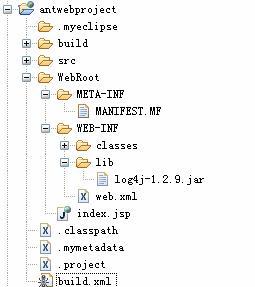
**3.**       **使用Ant的jar任务生成jar文件**

读者可以在上例的基础上更进一步，来生成jar包，可在run这个target下再加上如下target：

http://www.blogjava.net/Images/OutliningIndicators/None.gif<target name="jar" depends="run">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif<jar destfile="helloworld.jar" basedir="build/classes">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif           <manifest>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif              <attribute name="Main-class" value="HelloWorld"/>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif           </manifest>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif       </jar>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif</target>

此时将ant的project的default属性设置为jar，同时运行该build.xml文件，运行完毕后，可看到在工程目录下生成了一个jar包HelloWorld.jar。

**4.**       **使用Ant的war任务打包J2EE Web项目**

建立一个J2EE Web工程，其目录结构如下图所示：  


其中src为源代码目录，WebRoot为各jsp存放目录，lib为工程的包目录。在antwebproject工程目录下建立了build.xml文件，该文件为该工程的Ant构件文件。读者可以src目录下放入在前续例子中开发的HelloWorld.java文件，并在WebRoot下建立index.jsp文件，其内容很简单，就是输出Hello信息，代码如下所示：

http://www.blogjava.net/Images/OutliningIndicators/ExpandedBlockStart.gif<%@ page language="java" contentType="text/html; charset="UTF-8" pageEncoding="UTF-8"%>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif<html>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    <head>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif       <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif       <title>ant打包测试</title>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    </head>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    <body>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif       Hello,Ant  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    </body>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif</html>

   接下来编写build.xml文件，其内容如下：

http://www.blogjava.net/Images/OutliningIndicators/None.gif<?xml version="1.0"?>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif<project name="antwebproject"  default="war"basedir=".">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif <property name="classes" value="build/classes"/>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    <property name="build"value="build"/>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    <property name="lib"value="WebRoot/WEB-INF/lib"/>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    <!-- 删除build路径-->  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    <target name="clean">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif       <delete dir="build"/>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    </target>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    <!-- 建立build/classes路径，并编译class文件到build/classes路径下-->  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    <target name="compile" depends="clean">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif       <mkdir dir="${classes}"/>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif       <javac srcdir="src" destdir="${classes}"/>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    </target>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    <!-- 打war包-->  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    <target name="war" depends="compile">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif<war destfile="${build}/antwebproject.war" webxml="WebRoot/WEB-INF/web.xml">  
http://www.blogjava.net/Images/OutliningIndicators/None.gif           <!-- 拷贝WebRoot下除了WEB-INF和META-INF的两个文件夹-->  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    <fileset dir="WebRoot" includes="\*\*/\*.jsp"/>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif           <!-- 拷贝lib目录下的jar包-->  
http://www.blogjava.net/Images/OutliningIndicators/None.gif           <lib dir="${lib}"/>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif           <!-- 拷贝build/classes下的class文件-->  
http://www.blogjava.net/Images/OutliningIndicators/None.gif           <classesdir="${classes}"/>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif       </war>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    </target>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif</project>  
http://www.blogjava.net/Images/OutliningIndicators/None.gif

    各target的作用在内容中已经进行说明，在此不再赘述。运行该build文件，更新目录后，可看到在build目录下生成了antwebproject.war文件，解开后可看到其目录结构如下：

--META-INF

     --MANIFEST.MF

--index.jsp

--WEB-INF

     --lib

            --log4j-1.2.9.jar

     --classes

            --HelloWorld.class

     --web.xml

     读者可以将该war包拷贝到Tomcat的目录下看一下运行结果。

**五．**             **总结**

在本文中，笔者由浅至深详细描述了Ant的安装与配置、关键元素和常用任务。并通过实例讲述了Ant在我们Java项目中的应用，讲述了编译、运行java程序，以及打jar包、war包等知识，引领读者进入Ant的奇妙世界。在本文中可以看到，Ant在自动构建和部署Java程序方面方便易用，而且非常灵活，不失为我们Java开发者的绝佳帮手。

-------------------------------------------------------------

Java \_Ant详解

1,什么是ant  
ant是构建工具  
2,什么是构建  
概念到处可查到，形象来说，你要把代码从某个地方拿来，编译，再拷贝到某个地方去等等操作，当然不仅与此，但是主要用来干这个  
3,ant的好处  
跨平台   --因为ant是使用java实现的，所以它跨平台  
使用简单--与ant的兄弟make比起来  
语法清晰--同样是和make相比  
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。当你自己开发一些ant插件的时候，你会发现它更多的功能。  
4,ant的兄弟make  
ant做的很多事情，大部分是曾经有一个叫make的所做的，不过对象不同，make更多应用于c/c++ ,ant更多应用于Java。当然这不是一定的，但大部分人如此。  
一,构建ant环境  
要使用ant首先要构建一个ant环境，步骤很简单：  
1),安装jdk，设置JAVA\_HOME ,PATH ,CLASS\_PATH(这些应该是看这篇文章的人应该知道的)  
2),下载ant 地址<http://www.apache.org/>找一个你喜欢的版本，或者干脆最新的版本  
3),解压ant 你得到的是一个压缩包，解压缩它，并把它放在一个尽量简单的目录，例如D:/ant-1.6虽然你不一 定要这么做，但这么做是有好处的。  
4),设置ANT\_HOME PATH中添加ANT\_HOME目录下的bin目录  
5),测试一下你的设置,开始-->运行-->cmd进入命令行-->键入 ant 回车,如果看到  
Buildfile: build.xml does not exist!  
Build failed  
那么恭喜你你已经完成ant的设置  
二,体验ant  
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  
1,首先你要知道你要干什么，我现在想做的事情是：  
编写一些程序  
编译它们  
把它打包成jar包  
把他们放在应该放置的地方  
运行它们  
这里为了简单起见只写一个程序，就是HelloWorld.java程序代码如下：  
package test.ant;  
public class HelloWorld{  
public static void main(String[] args){  
   System.out.println("Hello world1");  
}  
};  
2，为了达到上边的目的，你可以手动的用javac 、copy 、jar、java来完成，但是考虑一下如果你有成百上千个类，在多次调试，部署的时候，一次次的javac 、copy、jar、  
java那将是一份辛苦的工作。现在看看ant怎么优雅的完成它们。  
要运行ant需要有一个build.xml虽然不一定要叫这个名字，但是建议你这么做  
下边就是一个完整的build.xml，然后我们来详细的解释每一句

1. <?xml version="1.0" encoding="UTF-8" ?>
2. <project name="HelloWorld" default="run" basedir=".">
3. <property name="src" value="src"/>
4. <property name="dest" value="classes"/>
5. <property name="hello\_jar" value="hello1.jar"/>
6. <target name="init">
7. <mkdir dir="${dest}"/>
8. </target>
9. <target name="compile" depends="init">
10. <javac srcdir="${src}" destdir="${dest}"/>
11. </target>
12. <target name="build" depends="compile">
13. <jar jarfile="${hello\_jar}" basedir="${dest}"/>
14. </target>
15. <target name="run" depends="build">
16. <java classname="test.ant.HelloWorld" classpath="${hello\_jar}"/>
17. </target>
18. <target name="clean">
19. <delete dir="${dest}" />
20. <delete file="${hello\_jar}" />
21. </target>
22. <target name="rerun" depends="clean,run">
23. <ant target="clean" />
24. <ant target="run" />
25. </target>
26. </project>
27. 解释：
28. <?xml version="1.0" encoding="UTF-8" ?>
29. build.xml中的第一句话，没有实际的意义
30. <project name="HelloWorld" default="run" basedir=".">
31. </project>
32. ant的所有内容必须包含在这个里边，name是你给它取的名字，basedir故名思意就是工作的根目录 .代表当前目录。default代表默认要做的事情。
33. <property name="src" value="src"/>
34. 类似程序中的变量，为什么这么做想一下变量的作用
35. <target name="compile" depends="init">
36. <javac srcdir="${src}" destdir="${dest}"/>
37. </target>
38. 把你想做的每一件事情写成一个target ，它有一个名字，depends是它所依赖的target，在执行这个target 例如这里的compile之前ant会先检查init是否曾经被执行过，如果执行
39. 过则直接直接执行compile，如果没有则会先执行它依赖的target例如这里的init，然后在执行这个target
40. 如我们的计划
41. 编译：
42. <target name="compile" depends="init">
43. <javac srcdir="${src}" destdir="${dest}"/>
44. </target>
45. 做jar包:
46. <target name="build" depends="compile">
47. <jar jarfile="${hello\_jar}" basedir="${dest}"/>
48. </target>
49. 运行：
50. <target name="run" depends="build">
51. <java classname="test.ant.HelloWorld" classpath="${hello\_jar}"/>
52. </target>
53. 为了不用拷贝，我们可以在最开始定义好目标文件夹，这样ant直接把结果就放在目标文件夹中了
54. 新建文件夹:
55. <target name="init">
56. <mkdir dir="${dest}"/>
57. </target>
58. 为了更多一点的功能体现，又加入了两个target
59. 删除生成的文件
60. <target name="clean">
61. <delete dir="${dest}" />
62. <delete file="${hello\_jar}" />
63. </target>
64. 再次运行，这里显示了如何在一个target里边调用其他的target
65. <target name="rerun" depends="clean,run">
66. <ant target="clean" />
67. <ant target="run" />
68. </target>
69. 好了，解释完成了，下边检验一下你的ant吧
70. 新建一个src的文件夹，然后把HelloWorld.java按照包目录放进去
71. 做好build.xml文件
72. 在命令行下键入ant ,你会发现一个个任务都完成了。每次更改完代码只需要再次键入ant
73. 有的时候我们可能并不想运行程序，只想执行这些步骤中的某一两个步骤，例如我只想重新部署而不想运行，键入
74. ant build
75. ant中的每一个任务都可以这样调用ant + target name
76. 好了，这样一个简单的ant任务完成了。

一,什么时候使用ant  
也许你听到别人说起ant，一时冲动准备学习一下ant，当你看完了上边的第一个实例，也许你感觉ant真好，也许你感觉ant不过如此，得出这些结论都不能说错，虽然ant很好用，  
但并不是在任何情况下都是最好的选择，例如windows上有更多更简单，更容易使用的工具，比如eclipse+myeclipse eclipse+wtp等等，无论是编译，部署，运行使用起来比ant更  
容易，方便但有些情况则是ant发挥的好地方：  
1,服务器上部署的时候  
当你的程序开发完成，部署人员要部署在服务器上的时候，总不能因为因为安装一个程序就配置一个eclipse+myeclipse吧，ant在这个时候是个很好的选择，因为它小巧，容易配  
置，你带着你写好的build.xml到任何一台服务器上，只需要做简单的修改（一些设定，例如目录），然后一两个命令完成，这难道不是一件美好的事情吗。  
2,linux上，很多时候是这样的，程序开发是在windows下，但是程序要在linux或者unix上运行，在linux或者  
在unix(特别是unix上)部署是个麻烦的事情，这个时候ant的特点又出来了，因为ant是跨平台的，你在build.xml可以在大多数操作系统上使用，基本不需要修改。  
3,当服务器维护者不懂编程的时候  
很多人都有过这样的经历，使用你们程序的人，并不懂得写程序。你得程序因为版本更新，因为修正bug需要一次又一次得重新部署。这个时候你会发现教一个人是如此得困难。但  
是有ant后，你只需要告诉他，输入ant xxx等一两个命令，一切ok.  
以上是我遇到得一些情况。  
看完以上得情况，好好考虑一下，你是否需要使用ant，如果是继续。  
  
进一步学习一个稍微复杂一点点的ant  
在实际的工作过程中可能会出现以下一些情况，一个项目分成很多个模块，每个小组或者部门负责一个模块，为了测试，他们自己写了一个build.xml,而你负责把这些模块组合到  
一起使用，写一个build.xml  
这个时候你有两种选择：  
1,自己重新写一个build.xml ，这将是一个麻烦的事情  
2,尽量利用他们已经写好的build.xml，减少自己的工作  
举个例子：  
假设你下边有三个小组，每个小组负责一个部分，他们分别有一个src 和一个写好的build.xml  
这个时候你拿到他们的src，你需要做的是建立三个文件夹src1 ,src2, src3分别把他们的src和build.xml放进去，然后写一个build.xml

1. <?xml version="1.0" encoding="UTF-8" ?>
2. <project name="main" default="build" basedir=".">
3. <property name="bin" value="${basedir}/bin" />
4. <property name="src1" value="${basedir}/src1" />
5. <property name="src2" value="${basedir}/src2" />
6. <property name="src3" value="${basedir}/src3" />
7. <target name="init">
8. <mkdir dir="${bin}" />
9. </target>
10. <target name="run">
11. <ant dir="${src1}" target="run" />
12. <ant dir="${src2}" target="run" />
13. <ant dir="${src3}" target="run" />
14. </target>
15. <target name="clean">
16. <ant dir="${src1}" target="clean" />
17. <ant dir="${src2}" target="clean" />
18. <ant dir="${src3}" target="clean" />
19. </target>
20. <target name="build" depends="init,call">
21. <copy todir="${bin}">
22. <fileset dir="${src1}">
23. <include name="\*.jar" />
24. </fileset>
25. <fileset dir="${src2}">
26. <include name="\*.jar" />
27. </fileset>
28. <fileset dir="${src3}">
29. <include name="\*.jar" />
30. </fileset>
31. </copy>
32. </target>
33. <target name="rebuild" depends="build,clean">
34. <ant target="clean" />
35. <ant target="build" />
36. </target>
37. </project>

ok你的任务完成了。  
  
ok,上边你完成了任务，但是你是否有些感触呢，在那些build.xml中，大多数是重复的，而且更改一次目录需要更改不少东西。是否能让工作做的更好一点呢，答案是肯定的。  
引入两个东西：  
1,propery  
2,xml include  
这两个东西都有一个功能，就是能把build.xml中<propery />中的内容分离出来，共同使用  
除此之外它们各有特点：  
propery的特点是维护简单，只需要简单的键值对，因为并不是所有人都喜欢xml的格式  
xml include的特点是不单可以提取出属性来，连target也可以。  
还是以前的例子：  
例如我们想把src1 src2 src3这三个属性从xml中提出来，可以新建一个文件叫all.properties  
里边的内容  
src1=D://study//ant//src1  
src2=D://study//ant//src2  
src3=D://study//ant//src3  
然后你的build.xml文件可以这样写，别人只需要更改配置文件，而不许要更改你的build.xml文件了

1. <?xml version="1.0" encoding="UTF-8" ?>
2. <project name="main" default="build" basedir=".">
3. <property file="all.properties" />
4. <property name="bin" value="${basedir}/bin" />
5. <target name="init">
6. <mkdir dir="${bin}" />
7. </target>
8. <target name="run">
9. <ant dir="${src1}" target="run" />
10. <ant dir="${src2}" target="run" />
11. <ant dir="${src3}" target="run" />
12. </target>
13. <target name="clean">
14. <ant dir="${src1}" target="clean" />
15. <ant dir="${src2}" target="clean" />
16. <ant dir="${src3}" target="clean" />
17. </target>
18. <target name="build" depends="init,call">
19. <copy todir="${bin}">
20. <fileset dir="${src1}">
21. <include name="\*.jar" />
22. </fileset>
23. <fileset dir="${src2}">
24. <include name="\*.jar" />
25. </fileset>
26. <fileset dir="${src3}">
27. <include name="\*.jar" />
28. </fileset>
29. </copy>
30. </target>
31. <target name="rebuild" depends="build,clean">
32. <ant target="clean" />
33. <ant target="build" />
34. </target>
35. <target name="test">
36. <ant dir="${src1}" target="test" />
37. <ant dir="${src2}" target="test" />
38. <ant dir="${src3}" target="test" />
39. </target>
40. </project>
41. 如果你自己看的话你会看到这样一个target
42. <target name="test">
43. <ant dir="${src1}" target="test" />
44. <ant dir="${src2}" target="test" />
45. <ant dir="${src3}" target="test" />
46. </target>
47. 有的时候你想给每个小组的build.xml加入几个target，一种做法是每个里边写，然后在这里调用
48. 但是有一种更好的方法。
49. 你可以写一个include.xml文件，内容如下
50. <?xml version="1.0" encoding="UTF-8" ?>
51. <property name="src" value="src"/>
52. <property name="dest" value="classes"/>
53. <target name="test" >
54. <ant target="run" />
55. </target>
56. 然后更改你三个小组的build.xml文件,每个里边加入如下内容
57. <!--include a xml file ,it can be common propery ,can be also a target   -->
58. <!DOCTYPE project [
59. <!ENTITY share-variable SYSTEM "file:../include.xml">
60. ]>
61. &share-variable;
62. 变成如下的样子
63. 这个时候，你只要在include.xml添加propery , 添加target，三个build.xml会同时添加这些propery和target
64. 而且不会让三个组的build.xml变得更复杂。
65. <?xml version="1.0" encoding="UTF-8" ?>
66. <!--include a xml file ,it can be common propery ,can be also a target   -->
67. <!DOCTYPE project [
68. <!ENTITY share-variable SYSTEM "file:../include.xml">
69. ]>
70. <project name="HelloWorld" default="run" basedir=".">
71. <!--use the include   -->
72. &share-variable;
73. <!--defined the property-->
74. <!--via include
75. <property name="src" value="src"/>
76. <property name="dest" value="classes"/>
77. -->
78. <property name="hello\_jar" value="hello1.jar"/>
79. <!--define the op-->
80. <target name="init">
81. <mkdir dir="${dest}"/>
82. </target>
83. <target name="compile" depends="init">
84. <javac srcdir="${src}" destdir="${dest}"/>
85. </target>
86. <target name="build" depends="compile">
87. <jar jarfile="${hello\_jar}" basedir="${dest}"/>
88. </target>
89. <target name="run" depends="build">
90. <java classname="test.ant.HelloWorld" classpath="${hello\_jar}"/>
91. </target>
92. <target name="clean">
93. <delete dir="${dest}" />
94. <delete file="${hello\_jar}" />
95. </target>
96. <target name="rerun" depends="clean,run">
97. <ant target="clean" />
98. <ant target="run" />
99. </target>
100. </project>

掌握了上边的那些内容之后，你就知道如何去写一个好的ant，但是你会发现当你真的想去做的时候，你不能马上作出好的build.xml，因为你知道太少的ant的默认提供的命令.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。   
  
ant的例子很好找，各种开源框架都会带有一个build.xml仔细看看，会有很大收获  
另外一个经常会用到的，但是在开源框架的build.xml一般没有的是cvs  
如果使用的是远程的cvs，可以这样使用

1. <xml version="1.0" encoding="utf-8"?>
2. <project>
3. <property name="cvsroot" value=":pserver:wang:@192.168.1.2:/cvsroot"/>
4. <property name="basedir" value="/tmp/testant/"/>
5. <property name="cvs.password" value="wang"/>
6. <property name="cvs.passfile" value="${basedir}/ant.cvspass"/>
7. <target name="initpass">
8. <cvspass cvsroot="${cvsroot}" password="${cvs.password}" passfile="${cvs.passfile}"/>
9. </target>
10. <target name="checkout" depends="initpass">
11. <cvs cvsroot="${cvsroot}" command="checkout" cvsrsh="ssh" package="myproject" dest="${basedir}"
12. passfile="${cvs.passfile}"/>
13. </target>
14. </project>

在eclipse里边先天支持ant，所以你可以在eclipse里边直接写build.xml  
因为eclipse提供了提示功能，自动补充功能，它能让你事半功倍。  
使用方法，只需要建立一个工程，然后建立一个叫build.xml的文件。然后就可以在里边写你的ant build了  
但是时刻记住<http://www.apache.org/>永远能找到你需要的东西

# [Ant之build.xml详解](http://blog.csdn.net/bao19901210/article/details/23843749)

标签： [ant](http://www.csdn.net/tag/ant)

2014-04-16 14:21 12041人阅读 [评论](http://blog.csdn.net/bao19901210/article/details/23843749#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/bao19901210/article/details/23843749#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

java（48） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

**ANT build.xml文件详解（一）**

Ant的概念   
可能有些读者并不连接什么是Ant以及入可使用它，但只要使用通过Linux系统得读者，应该知道

make这个命令。当编译Linux内核及一些软件的源程序时，经常要用这个命令。Make命令其实就

是一个项目管理工具，而Ant所实现功能与此类似。像make，gnumake和nmake这些编译工具都有

一定的缺陷，但是Ant却克服了这些工具的缺陷。最初Ant开发者在开发跨平台的应用时，用样也

是基于这些缺陷对Ant做了更好的设计。

Ant 与 makefile   
Makefile有一些不足之处，比如很多人都会碰到的烦人的Tab问题。最初的Ant开发者多次强调”

只是我在Tab前面加了一个空格，所以我的命令就不能执行”。有一些工具在一定程度上解决了

这个问题，但还是有很多其他的问题。Ant则与一般基于命令的工具有所不同，它是Java类的扩

展。Ant运行需要的XML格式的文件不是Shell命令文件。它是由一个Project组成的，而一个

Project又可分成可多target，target再细分又分成很多task，每一个task都是通过一个实现特

定接口的java类来完成的。

Ant的优点

Ant是Apache软件基金会JAKARTA目录中的一个子项目，它有以下的优点。   
跨平台性。Ant是存Java语言编写的，所示具有很好的跨平台性。   
操作简单。Ant是由一个内置任务和可选任务组成的。Ant运行时需要一个XML文件(构建文件)。

Ant通过调用target树，就可以执行各种task。每个task实现了特定接口对象。由于Ant构建文件

时XML格式的文件，所以和容易维护和书写，而且结构很清晰。   
Ant可以集成到开发环境中。由于Ant的跨平台性和操作简单的特点，它很容易集成到一些开发环

境中去。

Ant 开发

Ant的构建文件   
当开始一个新的项目时，首先应该编写Ant构建文件。构建文件定义了构建过程，并被团队开发

中每个人使用。Ant构建文件默认命名为build.xml，也可以取其他的名字。只不过在运行的时候

把这个命名当作参数传给Ant。构建文件可以放在任何的位置。一般做法是放在项目顶层目录中

，这样可以保持项目的简洁和清晰。下面是一个典型的项目层次结构。   
(1) src存放文件。   
(2) class存放编译后的文件。   
(3) lib存放第三方JAR包。   
(4) dist存放打包，发布以后的代码。   
Ant构建文件是XML文件。每个构建文件定义一个唯一的项目(Project元素)。每个项目下可以定

义很多目标(target元素)，这些目标之间可以有依赖关系。当执行这类目标时，需要执行他们所

依赖的目标。   
每个目标中可以定义多个任务，目标中还定义了所要执行的任务序列。Ant在构建目标时必须调

用所定义的任务。任务定义了Ant实际执行的命令。Ant中的任务可以为3类。   
（1） 核心任务。核心任务是Ant自带的任务。   
（2） 可选任务。可选任务实来自第三方的任务，因此需要一个附加的JAR文件。   
（3） 用户自定义的任务。用户自定义的任务实用户自己开发的任务。   
1.<project>标签   
   每个构建文件对应一个项目。<project>标签时构建文件的根标签。它可以有多个内在属性，

就如代码中所示，其各个属性的含义分别如下。   
(1) default表示默认的运行目标，这个属性是必须的。   
(2) basedir表示项目的基准目录。   
(3) name表示项目名。   
(4) description表示项目的描述。   
每个构建文件都对应于一个项目，但是大型项目经常包含大量的子项目，每一个子项目都可以有

自己的构建文件。

2.<target>标签   
一个项目标签下可以有一个或多个target标签。一个target标签可以依赖其他的target标签。例

如，有一个target用于编译程序，另一个target用于声称可执行文件。在生成可执行文件之前必

须先编译该文件，因策可执行文件的target依赖于编译程序的target。Target的所有属性如下。   
(1).name表示标明，这个属性是必须的。   
(2).depends表示依赖的目标。   
(3)if表示仅当属性设置时才执行。   
(4)unless表示当属性没有设置时才执行。   
(5)description表示项目的描述。   
Ant的depends属性指定了target的执行顺序。Ant会依照depends属性中target出现顺序依次执行

每个target。在执行之前，首先需要执行它所依赖的target。程序中的名为run的target的

depends属性compile，而名为compile的target的depends属性是prepare，所以这几个target执

行的顺序是prepare->compile->run。   
一个target只能被执行一次，即使有多个target依赖于它。如果没有if或unless属性，target总

会被执行。

3.<mkdir>标签   
该标签用于创建一个目录，它有一个属性dir用来指定所创建的目录名，其代码如下：   
<mkdir dir=”＄{class.root}”/>   
通过以上代码就创建了一个目录，这个目录已经被前面的property标签所指定。

4<jar>标签   
该标签用来生成一个JAR文件，其属性如下。   
(1) destfile表示JAR文件名。   
(2) basedir表示被归档的文件名。   
(3) includes表示别归档的文件模式。   
(4) exchudes表示被排除的文件模式。

5．<javac标签>   
该标签用于编译一个或一组java文件，其属性如下。   
(1).srcdir表示源程序的目录。   
(2).destdir表示class文件的输出目录。   
(3).include表示被编译的文件的模式。   
(4).excludes表示被排除的文件的模式。   
(5).classpath表示所使用的类路径。   
(6).debug表示包含的调试信息。   
(7).optimize表示是否使用优化。   
(8).verbose 表示提供详细的输出信息。   
(9).fileonerror表示当碰到错误就自动停止。

6．<java>标签   
该标签用来执行编译生成的.class文件，其属性如下。   
(1).classname 表示将执行的类名。   
(2).jar表示包含该类的JAR文件名。   
(3).classpath所表示用到的类路径。   
(4).fork表示在一个新的虚拟机中运行该类。   
(5).failonerror表示当出现错误时自动停止。   
(6).output 表示输出文件。   
(7).append表示追加或者覆盖默认文件。

7.<delete>标签   
该标签用于删除一个文件或一组文件，去属性如下。   
(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。   
(5).verbose表示指定是否列出所删除的文件，默认值为不列出。

8.<copy>标签   
该标签用于文件或文件集的拷贝，其属性如下。   
(1).file 表示源文件。   
(2).tofile 表示目标文件。   
(3).todir 表示目标目录。   
(4).overwrite 表示指定是否覆盖目标文件，默认值是不覆盖。   
(5).includeEmptyDirs 表示制定是否拷贝空目录，默认值为拷贝。   
(6).failonerror 表示指定如目标没有发现是否自动停止，默认值是停止。   
(7).verbose 表示制定是否显示详细信息，默认值不显示。

Ant的数据类型   
在构建文件中为了标识文件或文件组，经常需要使用数据类型。数据类型包含在

org.apache.tool.ant.types包中。下面镜简单介绍构建文件中一些常用的数据类型。

1. argument 类型   
由Ant构建文件调用的程序，可以通过<arg>元素向其传递命令行参数，如apply,exec和java任

务均可接受嵌套<arg>元素，可以为各自的过程调用指定参数。以下是<arg>的所有属性。   
(1).values 是一个命令参数。如果参数种有空格，但又想将它作为单独一个值，则使用此属性

。   
(2).file表示一个参数的文件名。在构建文件中，此文件名相对于当前的工作目录。   
(3).line表示用空格分隔的多个参数列表。   
(4).path表示路径。

2.ervironment 类型   
   由Ant构建文件调用的外部命令或程序，<env>元素制定了哪些环境变量要传递给正在执行的系

统命令，<env>元素可以接受以下属性。   
(1).file表示环境变量值得文件名。此文件名要被转换位一个绝对路径。   
(2).path表示环境变量的路径。Ant会将它转换为一个本地约定。   
(3).value 表示环境变量的一个直接变量。   
(4).key 表示环境变量名。   
注意   file path 或 value只能取一个。

3.filelist类型   
Filelist 是一个支持命名的文件列表的数据类型，包含在一个filelist类型中的文件不一定是

存在的文件。以下是其所有的属性。   
(1).dir是用于计算绝对文件名的目录。   
(2).files 是用逗号分隔的文件名列表。   
(3).refid 是对某处定义的一个<filelist>的引用。   
注意   dir 和 files 都是必要的，除非指定了refid(这种情况下，dir和files都不允许使用)。

4.fileset类型   
Fileset 数据类型定义了一组文件，并通常表示为<fileset>元素。不过，许多ant任务构建成了

隐式的fileset,这说明他们支持所有的fileset属性和嵌套元素。以下为fileset 的属性列表。   
(1).dir表示fileset 的基目录。   
(2).casesensitive的值如果为false，那么匹配文件名时，fileset不是区分大小写的，其默认

值为true.   
(3).defaultexcludes 用来确定是否使用默认的排除模式，默认为true。   
(4).excludes 是用逗号分隔的需要派出的文件模式列表。   
(5).excludesfile 表示每行包含一个排除模式的文件的文件名。   
(6).includes 是用逗号分隔的，需要包含的文件模式列表。   
(7).includesfile 表示每行包括一个包含模式的文件名。

5.patternset 类型   
Fileset 是对文件的分组，而patternset是对模式的分组，他们是紧密相关的概念。

<patternset>支持4个属性：includes excludex includexfile 和 excludesfile,与fileset相

同。Patternset 还允许以下嵌套元素：include,exclude,includefile 和 excludesfile.

6.filterset 类型   
Filterset定义了一组过滤器，这些过滤器将在文件移动或复制时完成文件的文本替换。   
主要属性如下：   
(1).begintoken 表示嵌套过滤器所搜索的记号，这是标识其开始的字符串。   
(2).endtoken表示嵌套过滤器所搜索的记号这是标识其结束的字符串。   
(3).id是过滤器的唯一标志符。   
(4).refid是对构建文件中某处定义一个过滤器的引用。

7.Path类型   
Path元素用来表示一个类路径，不过它还可以用于表示其他的路径。在用作揖个属性时，路经中

的各项用分号或冒号隔开。在构建的时候，此分隔符将代替当前平台中所有的路径分隔符，其拥

有的属性如下。   
(1).location 表示一个文件或目录。Ant在内部将此扩展为一个绝对路径。   
(2).refid 是对当前构建文件中某处定义的一个path的引用。   
(3).path表示一个文件或路径名列表。

8.mapper类型   
Mapper类型定义了一组输入文件和一组输出文件间的关系，其属性如下。   
(1).classname 表示实现mapper类的类名。当内置mapper不满足要求时，用于创建定制mapper.   
(2).classpath表示查找一个定制mapper时所用的类型路径。   
(3).classpathref是对某处定义的一个类路径的引用。   
(4).from属性的含义取决于所用的mapper.   
(5).to属性的含义取决于所用的mapper.   
(6).type属性的取值为identity，flatten glob merge   regexp   其中之一，它定义了要是用的

内置mapper的类型。

Ant 的运行   
安装好Ant并且配置好路径之后，在命令行中切换到构建文件的目录，输入Ant命令就可以运行

Ant.若没有指定任何参数，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

构建文件。如果使用了 –find 选项，Ant 就会在上级目录中找构建文件，直至到达文件系统得

跟目录。如果构建文件的名字不是build.xml ，则Ant运行的时候就可以使用 –buildfile file

,这里file 指定了要使用的构建文件的名称，示例如下：   
Ant   
如下说明了表示当前目录的构建文件为build.xml 运行 ant 执行默认的目标。

Ant –buildfile   test.xml   
使用当前目录下的test.xml 文件运行Ant ,执行默认的目标

**ant的build.xml文件详解（二）**

以build\_for\_ejb\_templet.xml为示例，讲解Ant中常用的元素和任务。   
  
约定： "…"表示这里有很多代码，未列出   
  
build\_for\_ejb\_templet.xml任务的分为以下几大部分   
i. 开始   
ii. 初始化   
iii. 定义classpath   
iv. 为编译作准备   
v. 编译EJB部分   
vi. 编译WEB部分   
vii. 编译J2EE Application   
viii. 部署Application   
ix. 创建组件的API   
x. 确定build的目标   
  
build\_for\_ejb\_templet.xml的讲解   
  
**开始**  
<?xml version="1.0" encoding="UTF-8"?>   
讲解：encoding="UTF-8"表示XML文件采用UTF-8编码格式，如果要使用GBK编码，需定义为encodeing="GBK"。  
  
<!--    
   Build file for 'componentName'    
   Creation date : $Date: yyyy-m-d $    
   Updated date : $Date: yyyy-m-d $    
   Author:   developerName  
   Copyright 2002 CompanyName, Inc. All rights reserved.    
-->    
讲解：此部分为文件内容的简介，包括文件名称、创建日期、最后修改日期、创建文件的作者、版权。  
componentName 表示 文件名称  
yyyy-m-dd 表示 创建日期、最后修改日期的格式，如2002-5-1  
developerName 表示 创建文件的作者  
CompanyName 表示 公司名称或URL  
  
<project name="componentName" default="core" basedir=".">  
…  
</project>  
讲解：此部分定义了一个名称为componentName的项目元素，缺省的任务为"core"任务，根目录为当前目录。componentName表示组件的名称，这里指EJB的名称。  
  
  
  
**初始化**  
<target name="init">  
…  
</target>  
讲解：此部分用于初始化所有的变量  
  
<property file="../../build.properties" />  
讲解：导入项目根目录下build.properties中的全局变量，开发人员也可以在此文件中重新定义全局变量。  
  
<property name="jboss.lib" value="${jboss.home}/lib" />    
<property name="jboss.client" value="${jboss.home}/client" />    
<property name="jboss.deploy" value="${jboss.home}/server/${jboss.configuration}/deploy" />    
…  
<property name="jboss.dir"       value="${jboss.home}" />    
…  
<property name="deploy.ormi"     value=""/>    
<property name="deploy.username" value=""/>    
<property name="deploy.password" value=""/>  
讲解：定义和Jboss Application Server有关的变量，包括lib目录、client目录、  
deploy目录和J2EE Application部署要用到的一些变量。  
  
<property name="name"   value="componentName"/>    
讲解：定义组件的名称  
  
<property name="src.dir" value="${basedir}/src"/>    
讲解：定义源代码目录路径  
  
<property name="etc.dir" value="${basedir}/etc"/>    
讲解：定义资源目录路径  
  
<property name="lib.dir" value="${basedir}/lib"/>    
讲解：定义库目录路径  
  
<property name="build.dir" value="${basedir}/build"/>    
讲解：定义build目录路径  
  
<property name="src.main.dir" value="${src.dir}/main"/>  
讲解：定义源代码的主目录路径  
  
<property name="src.ejb.dir" value="${src.main.dir}/ejb"/>    
讲解：定义存放EJB的源代码目录路径  
  
<property name="src.javabean.dir" value="${src.main.dir}/javabean"/>  
讲解：定义存放JavaBean的源代码目录路径  
  
<property name="src.servlet.dir" value="${src.main.dir}/servlet"/>  
讲解：定义存放Servlet的源代码目录路径  
  
<property name="src.web.dir" value="${src.main.dir}/web"/>    
讲解：定义存放WEB部分文件（包括JSP程序、HTML文件、图片、CSS文件、JS脚本等）的目录路径  
  
<property name="javadoc.dir" value="${lib.dir}/docs/api"/>   
讲解：定义存放组件API的开发文件目录路径  
  
<property name="ejb-classes.dest" value="${lib.dir}/ejb"/>    
讲解：定义存放EJB的编译代码目录路径  
  
<property name="javabean-classes.dest" value="${lib.dir}/javabean"/>    
讲解：定义存放JavaBean的编译代码目录路径  
  
<property name="web-classes.dest" value="${lib.dir}/web/WEB-INF/classes" />  
讲解：定义WEB目录的类目录路径  
  
<property name="web-lib.dest" value="${lib.dir}/web/WEB-INF/lib" />    
讲解：定义WEB目录的库目录名称  
  
<property name="pkg-dist.name" value="${name}-pkg"/>    
讲解：定义压缩文档的名称  
  
<property name="ProjectName.components.home" value="../../components" />    
讲解：定义项目的组件目录路径  
  
<!-- Define componentName Component -->  
讲解：这段为注释，说明以下是定义当前组件的变量  
  
<property name="ProjectName.componentName.name" value="componentName"/>  
讲解：定义当前组件的名称  
  
<property name="ProjectName.componentName.home" value="${ProjectName.components.home}/componentName"/>  
讲解：定义当前组件的目录路径  
  
<property name="ProjectName.componentName.classbindir"   
value="${ProjectName.componentName.home}/lib/ejb"/>  
讲解：定义当前组件的EJB编译目录路径  
<property name="ProjectName.componentName.ejbjar"   
value="${ProjectName.componentName.home}/build/componentName-ejb.jar"/>  
讲解：定义当前组件内的EJB包的路径  
  
<property name="ProjectName.componentName.ejbjar.client"   
value="${ProjectName.componentName.home}/build/componentName-ejb-client.jar"/>  
讲解：定义当前组件内的EJB客户端包的路径  
  
<!-- Define referencesComponentName Component -->  
讲解：这段为注释，说明以下是定义引入其它组件的变量  
  
<property name="ProjectName.referencesComponentName.name"   
value="referencesComponentName"/>  
讲解：定义指定组件的名称  
  
<property name="ProjectName.referencesComponentName.home"   
value="${ProjectName.components.home}/referencesComponentName"/>  
讲解：定义指定组件的目录路径  
  
<property name="ProjectName.referencesComponentName.classbindir"   
value="${ProjectName.referencesComponentName.home}/lib/ejb"/>  
讲解：定义指定组件的EJB编译目录路径  
  
<property name="ProjectName.referencesComponentName.ejbjar"   
value="${ProjectName.referencesComponentName.home}  
/build/referencesComponentName-ejb.jar"/>  
讲解：定义指定组件内的EJB包的路径  
  
<property name="ProjectName.referencesComponentName.ejbjar.client"   
value="${ProjectName.referencesComponentName.home}  
/build/referencesComponentName-ejb-client.jar"/>  
讲解：定义指定组件内的EJB客户端包的路径  
  
<property name="build.classpath"   
value="${jboss.client}/jboss-j2ee.jar:${jboss.client}  
/jnp-client.jar:${jboss.client}/jnp-client.jar:${jboss.client}  
/jbossmq-client.jar:${jboss.client}/jbosssx-client.jar:${jboss.client}  
/concurrent.jar:${jboss.client}/jaas.jar:${jboss.lib}  
/jboss-jmx.jar:${jboss.home}/server/${jboss.configuration}  
/lib/jbosssx.jar:${jboss.home}/server/${jboss.configuration}  
/lib/mail.jar:${servlet-lib.path}:${ejb-classes.dest}:  
${web-classes.dest}:${ProjectName.componentName.classbindir}:  
${ProjectName.componentName.ejbjar.client}:  
${ProjectName.referencesComponentName.classbindir}:  
${ProjectName.referencesComponentName.ejbjar.client}" />  
讲解：定义classpath，编译bean时要用到。这是定义classpath的一种方法，下面还有另一种方法。  
  
**定义classpath**  
<!-- ================================================================== -->    
   <!-- Define the classpath for compile the component                     -->  
   <!-- ================================================================== -->    
   <path id="base.path">    
       <pathelement location="${jboss.client}/jboss-j2ee.jar" />    
       <pathelement location="${jboss.client}/jnp-client.jar" />    
       <pathelement location="${jboss.client}/jbossmq-client.jar" />    
       <pathelement location="${jboss.client}/jbosssx-client.jar" />    
       <pathelement location="${jboss.client}/concurrent.jar" />   
       <pathelement location="${jboss.client}/jaas.jar" />    
       <pathelement location="${jboss.lib}/jboss-jmx.jar" />    
       <pathelement   
location="${jboss.home}/server/${jboss.configuration}/lib/jbosssx.jar" />    
       <pathelement   
location="${jboss.home}/server/${jboss.configuration}/lib/mail.jar" />    
</path>    
讲解：此段定义应用服务器中包文件，如支持ejb的jboss-j2ee.jar、支持客户端程序的jnp-client.jar、jbossmq-client.jar、jbosssx-client.jar、支持JavaMail的mail.jar等。  
  
   <path id="project.path">    
       <path refid="base.path"/>  
       <pathelement location="${ProjectName.componentName.classbindir}"/>  
       <pathelement location="${ProjectName.componentName.ejbjar.client}"/>  
       <pathelement  
location="${ProjectName.referencesComponentName.classbindir}"/>  
       <pathelement location="${ProjectName.referencesComponentName.ejbjar.client}"/>  
   </path>    
讲解：此段定义项目中要用到的包文件。  
  
   <path id="web.path">  
         <path refid="project.path"/>  
         <pathelement location="${servlet-lib.path}"/>  
         <pathelement location="${ejb-classes.dest}"/>  
   </path>  
讲解：此段定义在编译servlet时的classpath，${ servlet-lib.path }是系统指定的Servlet引擎包。  
 **为编译作准备工作**  
<!--=============================================================== -->    
   <!-- Removes all created files and directories                       -->    
   <!-- ============================================================== -->    
   <target name="clean" depends="init">    
       <delete dir="${lib.dir}" />    
       <delete dir="${build.dir}" />    
   </target>  
讲解：清除build、lib目录下的所有文件和目录。  
  
   <!-- ================================================================ -->    
   <!-- Makes sure the needed directory structure is in place               -->    
   <!-- ================================================================ -->    
   <target name="prepare" depends="init,clean">    
       <mkdir dir="${lib.dir}" />    
       <mkdir dir="${lib.dir}/ejb" />    
       <mkdir dir="${lib.dir}/ejb/META-INF" />    
       <mkdir dir="${lib.dir}/web" />    
       <mkdir dir="${lib.dir}/web/WEB-INF" />    
       <mkdir dir="${lib.dir}/web/WEB-INF/classes" />    
       <mkdir dir="${lib.dir}/web/WEB-INF/lib" />    
       <mkdir dir="${lib.dir}/j2ee" />    
       <mkdir dir="${lib.dir}/META-INF" />    
       <mkdir dir="${lib.dir}/docs/api" />  
       <mkdir dir="${build.dir}" />    
   </target>    
讲解：创建build中要用到的所有目录，根据需要可以加入自定义的目录如：  
       <mkdir dir="${lib.dir}/javabean/ " />  
  
**编译EJB部分**  
<!-- ================================================================ -->    
   <!-- Compilation of the EJB part of the application                     -->    
   <!-- ================================================================ -->    
   <target name="ejb-classes" depends="prepare">    
       <javac srcdir="${src.ejb.dir}"    
             destdir="${ejb-classes.dest}"    
             includes="com/\*\*"    
             classpathref="base.path" />  
   </target>    
讲解：此小段用来完成编译ejb和其它help classes。根据需要可以加入一个非常有用的元素：  
             encoding="${javac.encoding}"  
  
   <target name="ejb-meta-inf" depends="prepare">    
       <copy file="${etc.dir}/ejb-jar.xml"    
             tofile="${lib.dir}/ejb/META-INF/ejb-jar.xml" />    
       <copy file="${etc.dir}/jaws.xml"    
             tofile="${lib.dir}/ejb/META-INF/jaws.xml" />    
       <copy file="${etc.dir}/jboss.xml"    
             tofile="${lib.dir}/ejb/META-INF/jboss.xml" />    
       <copy file="${etc.dir}/jbosscmp-jdbc.xml"    
             tofile="${lib.dir}/ejb/META-INF/jbosscmp-jdbc.xml" />    
   </target>  
讲解：此小段用来拷贝EJB部署文件，在JAS中EJB部署文件有jaws.xml、jboss.xml、jbosscmp-jdbc.xml。  
  
   <target name="ejb-jar" depends="ejb-classes,ejb-meta-inf">    
       <jar jarfile="${build.dir}/${name}-ejb.jar"    
           basedir="${lib.dir}/ejb" />    
   </target>    
讲解：此小段用来把class和部署文件压缩成包文件，这个包文件就是一个EJB组件。  
  
   <target name="ejbclientjar" depends="ejb-jar,web-classes">  
       <copy   
file="${ProjectName.referencesComponentName.home}/build/  
${ProjectName.referencesComponentName.name}-ejb-client.jar"    
           tofile="${build.dir}/${ProjectName.referencesComponentName.name}  
           -ejb-client.jar" />    
       <jar jarfile="${build.dir}/${name}-ejb-client.jar"    
           basedir="${lib.dir}/ejb"  
           excludes="com/ProjectName/componentName/ejb/ComponentNameEJB.class" />    
   </target>  
讲解：此小段用来把class和部署文件压缩成包文件，以支持客户端运行。  
  
**编译WEB部分**  
<!-- ================================================================== -->    
   <!-- Compilation of the web part of the application                     -->    
   <!-- ================================================================== -->    
   <target name="web-classes" depends="prepare,ejb-jar">    
       <javac srcdir="${src.servlet.dir}"    
             destdir="${lib.dir}/web/WEB-INF/classes"    
             includes="com/\*\*"    
             classpath="${build.classpath}" />    
   </target>    
讲解：此小段用来完成编译servlet。  
  
   <target name="web-web-inf" depends="prepare">  
       <copy file="${etc.dir}/jboss-web.xml"    
             tofile="${lib.dir}/web/WEB-INF/jboss-web.xml" />    
       <copy file="${etc.dir}/web.xml"    
             tofile="${lib.dir}/web/WEB-INF/web.xml" />    
   </target>  
讲解：此小段用来拷贝WEB部署文件，在JAS中WEB部署文件有jboss-web.xml。  
  
   <target name="war" depends="web-classes,web-web-inf">    
       <copy todir="${lib.dir}/web" >  
       <fileset dir="${src.web.dir}"/>  
       </copy>  
       <copy  
file="${build.dir}/${ProjectName.referencesComponentName.name}-ejb-client.jar"    
tofile="${lib.dir}/web/WEB-INF/lib/${ProjectName.referencesComponentName.name}  
-ejb-client.jar" />      
     <jar jarfile="${build.dir}/${name}-web.war"    
           basedir="${lib.dir}/web" />    
   </target>  
讲解：此小段用来把所有的JSP程序、Html、Css、图片和部署文件等压缩成WAR文件。  
  
**编译J2EE Application**  
<!-- ================================================================== -->    
   <!-- Compilation of the complete J2EE application (both web and EJB)     -->    
   <!-- ================================================================== -->    
   <target name="j2ee-meta-inf" depends="prepare">    
       <copy file="${etc.dir}/application.xml"    
             tofile="${lib.dir}/j2ee/META-INF/application.xml" />    
   </target>    
讲解：此小段用来拷贝Application的部署文件。  
  
   <target name="ear" depends="ejb-jar,war,j2ee-meta-inf">    
       <copy file="${build.dir}/${name}-ejb.jar"    
             tofile="${lib.dir}/j2ee/${name}-ejb.jar" />    
       <copy file="${build.dir}/${name}-web.war"    
             tofile="${lib.dir}/j2ee/${name}-web.war" />    
       <jar jarfile="${build.dir}/${name}.ear"    
           basedir="${lib.dir}/j2ee" />    
   </target>    
讲解：此小段用来把EJB组件、支持客户端运行的包和部署文件压缩成EAR文件，它就是一个J2EE Application。这里要说明，在进行build时，根据需要可以不生成EAR文件。  
  
**部署Application**  
<!-- ================================================================ -->  
   <!-- Deploy EAR file                                                   -->  
   <!-- ================================================================ -->  
   <target name="deploy-server" depends="ear,war">    
       <copy todir="${jboss.deploy}">    
         <fileset dir="${build.dir}" includes="\*.ear">    
         </fileset>    
       </copy>    
   </target>    
讲解：此小段用来部署Application，在JAS3.0中${jboss.deploy}是JAS的热部署目录。  
  
**创建组件的API**  
<!-- =================================================================== -->  
   <!-- Create class and package usage pages                                 -->  
   <!-- =================================================================== -->  
   <target name="docs" depends="init">  
     <javadoc locale="${javadoc.locale}" packagenames="${package.names}.${name}.\*"    
             destdir="${javadoc.dir}"    
             classpath="${build.classpath}"  
             encoding="${javadoc.encoding}"  
             author="${javadoc.author}"  
version="${javadoc.version}"   
use="${javadoc.usage}"    
             windowtitle="${project.name} ${name} Component API"  
             doctitle="${project.name} ${name} Component"    
             bottom="Copyright ${sign.copyright} ${project.date}   
${company.signature}. All Rights Reserved.">    
<classpath >  
         <pathelement path="${lib.dir}/ejb"/>    
       </classpath>  
       <sourcepath>  
         <pathelement path="${src.main.dir}/ejb"/>  
       </sourcepath>  
     </javadoc>  
   </target>  
讲解：此小段用来创建组件的API。这里强烈要求类设计人员和编码人员按照Javadoc定义的标签对源代码进行注释。  
  
  
  
**确定build的目标**  
<target name="startbanner">  
       <echo>+---------------------------------------+</echo>  
       <echo>+     Building ${name} Component         +</echo>  
       <echo>+---------------------------------------+</echo>  
   </target>  
   <target name="endbanner" depends="main" >  
       <echo>+---------------------------------------+</echo>  
       <echo>+     Finished ${name} Component         +</echo>  
       <echo>+---------------------------------------+</echo>  
   </target>  
   <target name="main" depends="startbanner, ejb-jar, ejbclientjar" />  
   <target name="main2" depends="startbanner, ejb-jar,  
ejbclientjar,war" />  
   <target name="main3" depends="startbanner, ejb-jar,   
ejbclientjar,war,ear,deploy-server" />  
   <target name="core" depends="ejb-war" />  
   <target name="ejb-war" depends="main2,endbanner" />  
   <target name="deploy" depends="main3,endbanner" />  
   <target name="all" depends="core, docs" />  
讲解：此小段用来确定build的目标。缺省目录为core，所以在build不加参数时，系统将只生成jar文件和war文件。如果 build时加上参数，系统将根据需要来生成文件，例如：在命令行输入ant deploy，系统将生成jar文件、war文件、ear文件，同时将ear文件进行部署。

**ant build.xml 详解(三)**

**在Eclipse中使用Ant Ant是Java平台下非常棒的批处理命令执行程序，能非常方便地自动完成编译，测试，打包，部署等等一系列任务，大大提高开发效率。如果你现在还没有开始使用Ant，那就要赶快开始学习使用，使自己的开发水平上一个新台阶。**

**Eclipse中已经集成了Ant，我们可以直接在Eclipse中运行Ant。**

**以前面建立的Hello工程为例，创建以下目录结构：**

**新建一个build.xml，放在工程根目录下。build.xml定义了Ant要执行的批处理命令。虽然Ant也可以使用其它文件名，但是遵循标准能更使开发更规范，同时易于与别人交流。**

**通常，src存放Java源文件，classes存放编译后的class文件，lib存放编译和运行用到的所有jar文件，web存放JSP等web文件，dist存放打包后的jar文件，doc存放API文档。**

**然后在根目录下创建build.xml文件，输入以下内容：**

**Xml代码**

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

**<project name="Hello world" default="doc">**

**<!-- properies -->**

**<property name="src.dir" value="src" />**

**<property name="report.dir" value="report" />**

**<property name="classes.dir" value="classes" />**

**<property name="lib.dir" value="lib" />**

**<property name="dist.dir" value="dist" />**

**<property name="doc.dir" value="doc"/>**

**<!-- 定义classpath -->**

**<path id="master-classpath">**

**<fileset file="${lib.dir}/\*.jar" />**

**<pathelement path="${classes.dir}"/>**

**</path>**

**<!-- 初始化任务 -->**

**<target name="init">**

**</target>**

**<!-- 编译 -->**

**<target name="compile" depends="init" description="compile the source files">**

**<mkdir dir="${classes.dir}"/>**

**<javac srcdir="${src.dir}" destdir="${classes.dir}" target="1.4">**

**<classpath refid="master-classpath"/>**

**</javac>**

**</target>**

**<!-- 测试 -->**

**<target name="test" depends="compile" description="run junit test">**

**<mkdir dir="${report.dir}"/>**

**<junit printsummary="on"**

**haltonfailure="false"**

**failureproperty="tests.failed"**

**showoutput="true">**

**<classpath refid="master-classpath" />**

**<formatter type="plain"/>**

**<batchtest todir="${report.dir}">**

**<fileset dir="${classes.dir}">**

**<include name="\*\*/\*Test.\*"/>**

**</fileset>**

**</batchtest>**

**</junit>**

**<fail if="tests.failed">**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*   One or more tests failed!   Check the output ...   \*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**</fail>**

**</target>**

**<!-- 打包成jar -->**

**<target name="pack" depends="test" description="make .jar file">**

**<mkdir dir="${dist.dir}" />**

**<jar destfile="${dist.dir}/hello.jar" basedir="${classes.dir}">**

**<exclude name="\*\*/\*Test.\*" />**

**<exclude name="\*\*/Test\*.\*" />**

**</jar>**

**</target>**

**<!-- 输出api文档 -->**

**<target name="doc" depends="pack" description="create api doc">**

**<mkdir dir="${doc.dir}" />**

**<javadoc destdir="${doc.dir}"**

**author="true"**

**version="true"**

**use="true"**

**windowtitle="Test API">**

**<packageset dir="${src.dir}" defaultexcludes="yes">**

**<include name="example/\*\*" />**

**</packageset>**

**<doctitle><![CDATA[<h1>Hello, test</h1>]]></doctitle>**

**<bottom><![CDATA[<i>All Rights Reserved.</i>]]></bottom>**

**<tag name="todo" scope="all" description="To do:" />**

**</javadoc>**

**</target>**

**</project>**

**以上xml依次定义了init（初始化），compile（编译），test（测试），doc（生成文档），pack（打包）任务，可以作为模板。**

**选中Hello工程，然后选择“Project”，“Properties”，“Builders”，“New…”，选择“Ant Build”：**

**填入Name：Ant\_Builder；Buildfile：build.xml；BaseDirectory：${workspace\_loc: /Hello}（按“BrowseWorkspace”选择工程根目录），由于用到了junit.jar包，搜索Eclipse目录，找到 junit.jar，把它复制到Hello/lib目录下，并添加到Ant的Classpath中：**

**然后在Builder面板中钩上Ant\_Build，去掉Java Builder：**

**再次编译，即可在控制台看到Ant的输出：**

**Buildfile: F:\eclipse-projects\Hello\build.xml**

**init:**

**compile:**

**[mkdir] Created dir: F:\eclipse-projects\Hello\classes**

**[javac] Compiling 2 source files to F:\eclipse-projects\Hello\classes**

**test:**

**[mkdir] Created dir: F:\eclipse-projects\Hello\report**

**[junit] Running example.HelloTest**

**[junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0.02 sec**

**pack:**

**[mkdir] Created dir: F:\eclipse-projects\Hello\dist**

**[jar] Building jar: F:\eclipse-projects\Hello\dist\hello.jar**

**doc:**

**[mkdir] Created dir: F:\eclipse-projects\Hello\doc**

**[javadoc] Generating Javadoc**

**[javadoc] Javadoc execution**

**[javadoc] Loading source files for package example...**

**[javadoc] Constructing Javadoc information...**

**[javadoc] Standard Doclet version 1.4.2\_04**

**[javadoc] Building tree for all the packages and classes...**

**[javadoc] Building index for all the packages and classes...**

**[javadoc] Building index for all classes...**

**[javadoc] Generating F:\eclipse-projects\Hello\doc\stylesheet.css...**

**[javadoc] Note: Custom tags that could override future standardtags:   @todo. To avoid potential overrides, use at least one periodcharacter (.) in custom tag names.**

**[javadoc] Note: Custom tags that were not seen:   @todo**

**BUILD SUCCESSFUL**

**Total time: 11 seconds**

**Ant依次执行初始化，编译，测试，打包，生成API文档一系列任务，极大地提高了开发效率。将来开发J2EE项目时，还可加入部署等任务。并且，即使脱离了Eclipse环境，只要正确安装了Ant，配置好环境变量ANT\_HOME=<Ant解压目录&gt;， Path=…;%ANT\_HOME%\bin，在命令行提示符下切换到Hello目录，简单地键入ant即可。**

**Ant 的最完整build.xml之标签：**

Ant 开发   
  
Ant的构建文件   
   当开始一个新的项目时，首先应该编写Ant构建文件。构建文件定义了构建过程，并被团队开发中每个人使用。Ant构建文件默认命名为 build.xml，也可以取其他的名字。只不过在运行的时候把这个命名当作参数传给Ant。构建文件可以放在任何的位置。一般做法是放在项目顶层目录中，这样可以保持项目的简洁和清晰。下面是一个典型的项目层次结构。   
(1) src存放文件。   
(2) class存放编译后的文件。   
(3) lib存放第三方JAR包。   
(4) dist存放打包，发布以后的代码。   
Ant构建文件是XML文件。每个构建文件定义一个唯一的项目(Project元素)。每个项目下可以定义很多目标(target元素)，这些目标之间可以有依赖关系。当执行这类目标时，需要执行他们所依赖的目标。每个目标中可以定义多个任务，目标中还定义了所要执行的任务序列。Ant在构建目标时必须调用所定义的任务。任务定义了Ant实际执行的命令。Ant中的任务可以为3类。   
（1） 核心任务。核心任务是Ant自带的任务。   
（2） 可选任务。可选任务实来自第三方的任务，因此需要一个附加的JAR文件。   
（3） 用户自定义的任务。用户自定义的任务实用户自己开发的任务。   
1.<project>标签   
  每个构建文件对应一个项目。<project>标签时构建文件的根标签。它可以有多个内在属性，就如代码中所示，其各个属性的含义分别如下。   
(1) default表示默认的运行目标，这个属性是必须的。   
(2) basedir表示项目的基准目录。   
(3) name表示项目名。   
(4) description表示项目的描述。   
每个构建文件都对应于一个项目，但是大型项目经常包含大量的子项目，每一个子项目都可以有自己的构建文件。   
  
2.<target>标签   
一个项目标签驴梢杂幸桓龌蚨喔?/span>target标签。一个target标签可以依赖其他的target标签。例如，有一个target用于编译程序，另一个target用于声称可执行文件。在生成可执行文件之前必须先编译该文件，因策可执行文件的target依赖于编译程序的 target。Target的所有属性如下。   
(1).name表示标明，这个属性是必须的。   
(2).depends表示依赖的目标。   
(3)if表示仅当属性设置时才执行。   
(4)unless表示当属性没有设置时才执行。   
(5)description表示项目的描述。   
Ant的depends属性指定了target的执行顺序。Ant会依照depends属性中target出现顺序依次执行每个target。在执行之前，首先需要执行它所依赖的target。程序中的名为run的target的depends属性compile，而名为compile的target的 depends属性是prepare，所以这几个target执行的顺序是prepare->compile->run。一个target只能被执行一次，即使有多个target依赖于它。如果没有if或unless属性，target总会被执行。   
  
3.<mkdir>标签   
该标签用于创建一个目录，它有一个属性dir用来指定所创建的目录名，其代码如下：   
<mkdir dir=”＄{class.root}”/>   
通过以上代码就创建了一个目录，这个目录已经被前面的property标签所指定。   
  
4<jar>标签   
该标签用来生成一个JAR文件，其属性如下。   
(1) destfile表示JAR文件名。   
(2) basedir表示被归档的文件名。   
(3) includes表示别归档的文件模式。   
(4) exchudes表示被排除的文件模式。   
  
5．<javac标签>   
该标签用于编译一个或一组java文件，其属性如下。   
(1).srcdir表示源程序的目录。   
(2).destdir表示class文件的输出目录。   
(3).include表示被编译的文件的模式。   
(4).excludes表示被排除的文件的模式。   
(5).classpath表示所使用的类路径。   
(6).debug表示包含的调试信息。   
(7).optimize表示是否使用优化。   
(8).verbose 表示提供详细的输出信息。   
(9).fileonerror表示当碰到错误就自动停止。   
  
6．<java>标签   
该标签用来执行编译生成的.class文件，其属性如下。   
(1).classname 表示将执行的类名。   
(2).jar表示包含该类的JAR文件名。   
(3).classpath所表示用到的类路径。   
(4).fork表示在一个新的虚拟机中运行该类。   
(5).failonerror表示当出现错误时自动停止。   
(6).output 表示输出文件。   
(7).append表示追加或者覆盖默认文件。   
  
7.<delete>标签   
该标签用于删除一个文件或一组文件，去属性如下。   
(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。   
(5).verbose表示指定是否列出所删除的文件，默认值为不列出。   
  
8.<copy>标签   
该标签用于文件或文件集的拷贝，其属性如下。   
(1).file 表示源文件。   
(2).tofile 表示目标文件。   
(3).todir 表示目标目录。   
(4).overwrite 表示指定是否覆盖目标文件，默认值是不覆盖。   
(5).includeEmptyDirs 表示制定是否拷贝空目录，默认值为拷贝。   
(6).failonerror 表示指定如目标没有发现是否自动停止，默认值是停止。   
(7).verbose 表示制定是否显示详细信息，默认值不显示。

**ant教程详解--javac，java，jar，war，delete，copy，mkdir：**

|  |
| --- |
| Ant 是一个 Apache 基金会下的跨平台的构件工具，它可以实现项目的自动构建和部署等功能。  在本文中，主要让读者熟悉怎样将 Ant 应用到 Java 项目中，让它简化构建和部署操作。  一．              安装与配置  下载地址： http://ant.apache.org/ ，在本文中下载的是 1.7.0 版本。解压到某个目录（例如 E:"apache-ant-1.7.0 ），  即可使用。  添加系统环境变量： ANT\_HOME ，该变量指向 Ant 解压后的根目录，在此为 E:"apache-ant-1.7.0 。  安装与配置完毕后，读者可以测试一下 Ant 是否可用，首先进入 Ant 的 bin 目录，运行命令 ant –version ，  若安装和配置成功，则会显示 Ant 版本信息，如下图所示：  由上可以看出，读者运行 Ant 的命令时，需要进入到 Ant 的 bin 目录，如何才能让系统自动找到 Ant 呢？  这时需要读者在系统环境变量 path 中添加 Ant 的 bin 目录。设置完成后，我们就可以在任何目录  （例如 C:"Documents and Settings"AmigoXie 目录）输入 Ant 的命令，来获得命令的运行结果。  二．              Ant 的关键元素  Ant 的构件文件是基于 XML 编写的，默认名称为 build.xml 。为了更清楚的了解 Ant ，  在这里编写一个简单的 Ant 程序，用来展现 Ant 的功能，让读者对 Ant 有一个初步的了解。  首先在 E 盘下建立一个 build.xml 文件，内容如下：  <? xml version="1.0" ?>  < project  name ="helloWorld">  < target  name ="sayHelloWorld">  < echo  message ="Hello,Amigo"/>  </ target >  </ project >  读者可以进入 E 盘，然后运行 ant sayHelloWorld ，可以看到如下的运行结果：  其中 sayHelloWorld 为需要执行的任务的名称。如果文件名不为 build.xml ，而为 hello.xml 时，  读者运行同样的命令时，命令窗口会出现如下错误：  Buildfile: build.xml does not exist!  Build failed  由上面的命令的错误提示可以看出， ant 命令默认寻找 build.xml 文件。若文件名为 hello.xml 时，  读者还需要对命令做少许改变，改为： ant –f hello.xml sayHelloWorld 、 ant –buildfile hello.xml sayHelloWorld  或 ant –file hello.xml sayHelloWorld 。  接下来开始向读者讲解本节的重点： Ant 的关键元素 project 、 target 、 property 和 task 。  1.        project 元素  project 元素是 Ant 构件文件的根元素， Ant 构件文件至少应该包含一个 project 元素，否则会发生错误。  在每个 project 元素下，可包含多个 target 元素。接下来向读者展示一下 project 元素的各属性。  1 ） name 属性  用于指定 project 元素的名称。  2 ） default 属性  用于指定 project 默认执行时所执行的 target 的名称。  3 ） basedir 属性  用于指定基路径的位置。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。  下面给读者一个简单的例子来展示 project 的各元素的使用。修改 E:"build.xml 文件，修改后的内容如下：  <? xml version="1.0" ?>  < project  name ="projectStudy"  default ="sayBaseDir"  basedir ="E:" apache-ant-1.7.0" >  < target  name ="sayBaseDir">  < echo  message ="The base dir is: ${basedir}"/>  </ target >  </ project >  从上面的内容我们可以看出，在这里定义了 default 属性的值为 sayBaseDir ，即当运行 ant 命令时，  若未指明执行的 target 时，默认执行的 target 的 sayBaseDir ，同时还定义了 basedir 属性的值为  E:"apache-ant-1.7.0 ，进入 E 盘后运行 ant 命令，可看到运行的结果，如下图所示：  因为设定了 basedir 的值，所以 basedir 属性的值变成了读者设置的值。读者可以自行将 project 元素的  basedir 属性去掉后运行 ant 看看运行结果，此时 basedir 的值变成了 E:" ，即为 Ant 构件文件的父目录。  有的时候，读者可能有这种需求，即想得到某个 project 下所有的 target 的名称，读者可以通过在  ant 命令里加上 -proecthelp 来达到该目的。例如针对上述的例子我们运行 ant –projecthelp ，输出结果如下：  Buildfile: build.xml  Main targets:  Other targets:  sayBaseDir  Default target: sayBaseDir  2.        target 元素  它为 Ant 的基本执行单元，它可以包含一个或多个具体的任务。多个 target 可以存在相互依赖关系。  它有如下属性：  1 ） name 属性  指定 target 元素的名称，这个属性在一个 project 元素中是唯一的。我们可以通过指定 target 元素的名称  来指定某个 target 。  2 ） depends 属性  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。  3 ） if 属性  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。  4 ） unless 属性  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。  5 ） description 属性  该属性是关于 target 功能的简短描述和说明。  下面带领读者来看一个各属性综合使用的例子。修改 E:"build.xml 文件，修改后的内容如下：  <? xml version="1.0" ?>  < project  name ="targetStudy">  < target  name ="targetA"  if ="ant.java.version">  < echo  message ="Java Version: ${ant.java.version}"/>  </ target >  < target  name ="targetB"  depends ="targetA"  unless ="amigo">  < description >  a depend example! </ description >  < echo  message ="The base dir is: ${basedir}"/>  </ target >  </ project >  进入 E 盘后运行 ant targetB ，可看到如下图所示的运行结果：  读者分析结果后可以看到，我们运行的是名为 targetB 的 target ，因该 target 依赖于 targetA ，  所以 targetA 将首先被执行，同时因为系统安装了 java 环境，所以 ant.java.version 属性存在，  执行了 targetA 这个 target ，输出信息： [echo] Java Version: 1.5 ， targetA 执行完毕后，  接着执行 targetB ，因为 amigo 不存在，而 unless 属性是在不存在时进入所在的 target 的，  由此可知 targetB 得以执行，输出信息： The base dir is: E:" 。  3.        property 元素  该元素可看作参量或者参数的定义， project 的属性可以通过 property 元素来设定，也可在 Ant 之外设定。  若要在外部引入某文件，例如 build.properties 文件，可以通过如下内容将其引入：  <property file=” build.properties”/>  property 元素可用作 task 的属性值。在 task 中是通过将属性名放在“ ${ ”和“ } ”之间，  并放在 task 属性值的位置来实现的。  Ant 提供了一些内置的属性，它能得到的系统属性的列表与 Java 文档中  System.getPropertis() 方法得到的属性一致，这些系统属性可参考 sun 网站的说明。  同时， Ant 还提供了一些它自己的内置属性，如下：  basedir ： project 基目录的绝对路径，该属性在讲解 project 元素时有详细说明，不再赘述；  ant.file ： buildfile 的绝对路径，如上面的各例子中， ant.file 的值为 E:"build.xml ；  ant.version ： Ant 的版本，在本文中，值为 1.7.0 ；  ant.project.name ：当前指定的 project 的名字，即前文说到的 project 的 name 属性的值；  ant.java.version ： Ant 检测到的 JDK 的版本，在上例运行结果中可看到为 1.5 。  下面让读者来看一个 property 元素使用的简单例子。修改 E:"build.xml 文件，内容如下：  <? xml version="1.0" ?>  < project  name ="propertyStudy"  default ="example">  < property  name ="name"  value ="amigo"/>  < property  name ="age"  value ="25"/>  < target  name ="example">  < echo  message ="name: ${name}, age: ${age}"/>  </ target >  </ project >  该例的运行结果如下图所示：  由此读者可以看出，通过如下两个语句：  <property name="name" value="amigo"/>  <property name="age" value="25"/>  我们设置了名为 name 和 age 的两个属性，这两个属性设置后，读者在下文中可以通过  ${name} 和 ${age} 分别取得这两个属性的值。  三．              Ant 的常用任务  在 Ant 工具中每一个任务封装了具体要执行的功能，是 Ant 工具的基本执行单位。在本小节中，  主要引导读者来看下 Ant 的常用任务及其使用举例。  1.        copy 任务  该任务主要用来对文件和目录的复制功能。举例如下：  Eg1. 复制单个文件： <copy file="file.txt" tofile="copy.txt"/>  Eg2. 对文件目录进行复制：  <copy todir="../newdir/dest\_dir">  <fileset dir="src\_dir"/>  </copy>  Eg3. 将文件复制到另外的目录：  <copy file="file.txt" todir="../other/dir"/>  2.        delete 任务  对文件或目录进行删除，举例如下：  Eg1. 删除某个文件： <delete file="photo/amigo.jpg"/>  Eg2. 删除某个目录： <delete dir="photo"/>  Eg3. 删除所有的备份目录或空目录：  <delete includeEmptyDirs="true">  <fileset dir="." includes="\*\*/\*.bak"/>  </delete>  3.        mkdir 任务  创建目录。 eg ： <mkdir dir="build"/>  4.        move 任务  移动文件或目录，举例如下：  Eg1. 移动单个文件： <move file="fromfile" tofile=”tofile”/>  Eg2. 移动单个文件到另一个目录： <move file="fromfile" todir=”movedir”/>  Eg3. 移动某个目录到另一个目录：  <move todir="newdir">  <fileset dir="olddir"/>  </move>  5.        echo 任务  该任务的作用是根据日志或监控器的级别输出信息。它包括 message 、 file 、 append 和 level 四个属性，  举例如下：  <echo message="Hello,Amigo" file="logs/system.log" append="true">  四．              利用 Ant 构建和部署 Java 工程  Ant 可以代替使用 javac 、 java 和 jar 等命令来执行 java 操作，从而达到轻松的构建和部署 Java 工程的目的。  下面来看几个知识点。  1.        利用 Ant 的 javac 任务来编译 java 程序  Ant 的 javac 任务用于实现编译 Java 程序的功能。下面来看一个简单的例子：  首先我们建立名为 antstudy 的 Java 工程，建立 src 目录为源代码目录，  在 src 目录下建立 HelloWorld.java 这个类文件。该类文件的内容如下：  public   class  HelloWorld  { public   static   void  main(String[] args)  { System.out.println("Hello,Amigo"); } }  同时在 antstudy 工程的根目录下建立 build.xml 文件，在该文件中编译 src 目录下的 java 文件，  并将编译后的 class 文件放入 build/classes 目录中，在编译前，需清除 classes 目录，该文件的内容如下：  <? xml version="1.0" ?>  < project  name ="javacTest" default ="compile"  basedir =".">  < target  name ="clean">  < delete  dir ="build"/>  </ target >   < target  name ="compile"  depends ="clean">  < mkdir  dir ="build/classes"/>  < javac  srcdir ="src"  destdir ="build/classes"/>  </ target >  </ project >  运行该 build.xml 文件，可在工程中看到新增了 build/classes 目录，并在该目录中生成了编译后的  HelloWorld.class 文件。  2.        使用 Ant 的 java 任务运行 Java 程序  Ant 中可以使用 java 任务实现运行 Java 程序的功能。下面在 1 的例子中进行如下的修改，  修改后的 build.xml 文件的内容如下：  <? xml version="1.0" ?>  < project  name ="javaTest"  default ="jar"  basedir =".">  < target  name ="clean">  < delete  dir ="build"/>  </ target >   < target  name ="compile"  depends ="clean">  < mkdir  dir ="build/classes"/>  < javac  srcdir ="src"  destdir ="build/classes"/>  </ target >   < target  name ="run"  depends ="compile">  < java  classname ="HelloWorld">  < classpath >  < pathelement  path ="build/classes"/>  </ classpath >  </ java >  </ target >  </ project >  运行该 build.xml 文件，可在控制台看到 HelloWorld 的 main 方法的输出。  3.        使用 Ant 的 jar 任务生成 jar 文件  读者可以在上例的基础上更进一步，来生成 jar 包，可在 run 这个 target 下再加上如下 target ：  < target  name ="jar"  depends ="run">  < jar  destfile ="helloworld.jar"  basedir ="build/classes">  < manifest >  < attribute  name ="Main-class"  value ="HelloWorld"/>  </ manifest >  </ jar >  </ target >  此时将 ant 的 project 的 default 属性设置为 jar ，同时运行该 build.xml 文件，运行完毕后，  可看到在工程目录下生成了一个 jar 包 HelloWorld.jar 。  4.        使用 Ant 的 war 任务打包 J2EE Web 项目  建立一个 J2EE Web 工程，其目录结构如下图所示： 其中 src 为源代码目录， WebRoot 为各 jsp 存放目录， lib 为工程的包目录。  在 antwebproject 工程目录下建立了 build.xml 文件，该文件为该工程的 Ant 构件文件。  读者可以 src 目录下放入在前续例子中开发的 HelloWorld.java 文件，并在 WebRoot 下建立 index.jsp 文件，  其内容很简单，就是输出 Hello 信息，代码如下所示：  <% @ page language="java" contentType="text/html; charset="UTF-8" pageEncoding="UTF-8" %>  <! DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd" >  < html >  < head >  < meta  http-equiv ="Content-Type"  content ="text/html; charset=ISO-8859-1">  < title > ant 打包测试 </ title >  </ head >  < body >  Hello,Ant </ body >  </ html >  接下来编写 build.xml 文件，其内容如下：  <? xml version="1.0" ?>  < project  name ="antwebproject"   default ="war" basedir =".">  < property  name ="classes"  value ="build/classes"/>  < property  name ="build" value ="build"/>  < property  name ="lib" value ="WebRoot/WEB-INF/lib"/>  <!--  删除build 路径-->  < target  name ="clean">  < delete  dir ="build"/>  </ target >   <!--  建立build/classes 路径，并编译class 文件到build/classes 路径下-->  < target  name ="compile"  depends ="clean">  < mkdir  dir ="${classes}"/>   < javac  srcdir ="src"  destdir ="${classes}"/>  </ target >   <!--  打war 包-->  < target  name ="war"  depends ="compile">  < war  destfile ="${build}/antwebproject.war"  webxml ="WebRoot/WEB-INF/web.xml">  <!--  拷贝WebRoot 下除了WEB-INF 和META-INF 的两个文件夹-->  < fileset  dir ="WebRoot"  includes ="\*\*/\*.jsp"/>   <!--  拷贝lib 目录下的jar 包-->  < lib  dir ="${lib}"/>  <!--  拷贝build/classes 下的class 文件-->  < classesdir ="${classes}"/>  </ war >  </ target >  </ project >  各 target 的作用在内容中已经进行说明，在此不再赘述。运行该 build 文件，更新目录后，  可看到在 build 目录下生成了 antwebproject.war 文件，解开后可看到其目录结构如下：  --META-INF  --MANIFEST.MF  --index.jsp  --WEB-INF  --lib  --log4j-1.2.9.jar  --classes  --HelloWorld.class  --web.xml  读者可以将该 war 包拷贝到 Tomcat 的目录下看一下运行结果。  五．              总结  在本文中，笔者由浅至深详细描述了 Ant 的安装与配置、关键元素和常用任务。并通过实例讲述了 Ant  在我们 Java 项目中的应用，讲述了编译、运行 java 程序，以及打 jar 包、 war 包等知识，  引领读者进入 Ant 的奇妙世界。在本文中可以看到， Ant 在自动构建和部署 Java 程序方面方便易用，  而且非常灵活，不失为我们 Java 开发者的绝佳帮手 |