本内容包含了Ant的历史简要介绍，Ant的功能以及Ant框架的介绍，并对下载安装使用Ant进行了示例介绍，同时通过一个Java程序讲解了Ant的基本使用方法。

**1.       Ant简介：这里引用Ant帮助文档中对Ant的介绍：**

Apache Ant是一个基于Java的构建工具。从理论上讲，也是一种类似于Make的工具，只是去除了Make工具的缺点。

既然已经有了make, gnumake, nmake, jam以及其他的构件工具，为什么还要Ant呢？因为Ant的早期开发者发现所有以上这些工具都或多或少的有一些局限性，使得在跨平台开发软件成为困难。 类似于Make的工具都是传统的基于Shell的--首先进行依赖性检查，然后执行命令。这意味着你可以轻易的通过使用或者编写程序来扩展这些工具，以满 足不同的平台。当然，这也意味着你将局限于特定的平台，至少可以说局限于特定类型的平台，例如：Unix平台。

同时，Make文件也有一些先天的缺陷。好多人都会遇到恐怖的tab问题。Ant的最初开发者多次说“我的命令不能执行因为我在tab前面加了一个空格！”。一些工具如Jam一定程序上解决了这个问题，但仍有其它的格式问题。Ant与从基于命令的那些扩展开来的那些工具不同，Ant是由java类扩展的。不用编写shell命令，而是配置基于XML的文件，形成多个任务的目标配置树。每一个任务都是通过一个实现了一个规定接口的java类来运行的。

ant缺少了一些直接执行shell命令的能力，如find . -name foo -exec rm {}，但它给用户提供了跨平台的能力，可以在任何地方工作。实际上，Ant也提供了命令execute用来执行shell命令,这就是它的任务，它允许执 行基于操作系统的命令。

简单的说，Ant是一个基于Java，并且主要用于Java工程的构建工具。Ant本意是Another Neat Tool,也就是另一种整洁的工具，取首字符就是Ant。

构建工具就是为了减少重复工作而产生的。

**2.       Ant的一些核心概念：**

XML:构建文件是以XML文件来描述的，采用XML格式有很多好处。这里就不一一列举。

陈述式语法：构建文件短小精悍，且易于理解。

每个构建文件包含一个工程(project)。

每个工程包含若干个目标(target)。

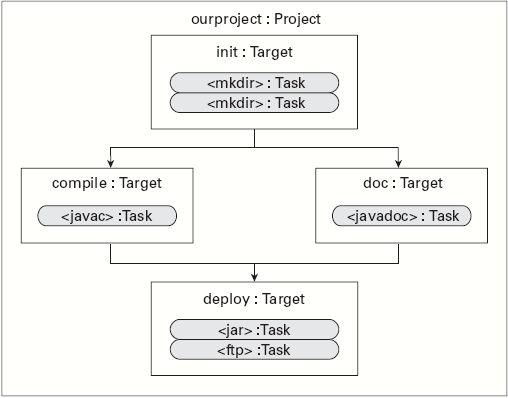
目标可以依赖于其他的目标(depends)。

目标包含任务(task)。

易于使用Java语言增加新的任务---易于扩展（自定义）。

**3.       Ant结构：**

Ant的结构如下图所示：



构建文件的概念视图：工程包含一个目标的集合。在每个目标里是任务的声明，它们是对Ant用于构建该目标的行为说明。目标生成一个依赖关系图表来声明该目标的依赖关系。当执行一个目标时，必须先执行它们依赖的目标。

**例子：一个典型的构建文件：**

<?xml version="1.0" ?>

<project name="OurProject" default="deploy">

<target name="init">

<mkdir dir="build/classes" />

<mkdir dir="dist" />

</target>

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

<javac srcdir="src" destdir="build/classes"/>

</target>

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

<javadoc destdir="build/classes" sourcepath="src" packagenames="org.\*" />

</target>

<target name="deploy" depends="compile,doc" >

    <jar destfile="dist/project.jar" basedir="build/classes"/>   
         <ftp server="" userid="" password="">

     <fileset dir="dist"/>

</ftp>

</target>

</project>

**该构建过程如下：**

系统初始化1、编译  2、生成JAVADOC   4、打包  5、上传到FTP，其中后两步结合到一起叫部署。

执行时输出如下：

> ant -propertyfile ftp.properties

Buildfile: build.xml

init:

[mkdir] Created dir: /home/ant/Projects/OurProject/build/classes

[mkdir] Created dir: /home/ant/Projects/OurProject/dist

compile:

[javac] Compiling 1 source file to /home/ant/Projects/OurProject/build/

classes

doc:

[javadoc] Generating Javadoc

[javadoc] Javadoc execution

[javadoc] Loading source files for package org.example.antbook.lesson1...

[javadoc] Constructing Javadoc information...

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

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

[javadoc] Building index for all classes...

deploy:

[jar] Building jar: /home/ant/Projects/OurProject/dist/project.jar

[ftp] sending files

[ftp] 1 files sent

BUILD SUCCESSFUL

Total time: 5 seconds.

在执行时使用命令行参数以传入一个属性文件，属性文件中包含连接FTP服务器使用的服务器名，用户名，用户密码来给特性使用。

这个例子很好的展示了Ant的一些基本要素：目标依赖、特性的使用、编译、文档生成、JAR打包（tar,Zip,WAR,EAR等），最后是部署。

Ant的简单任务(<mkdir>)都是由Java类库来实现相应的功能。而一些复杂的任务<ftp>、<junit>还需要第三方库的支持。

    Ant的一个强大之处：它总能工作。只要正确的指定构建文件，Ant就能计算出目标 的依赖性，并且按照正确的顺序调用目标。目标通过任务按序执行，而任务自身处理其文件依赖性以及实际的操作来完成工作。因为每个任务通常都是在高层陈述， 所以一两行XML语句经常就已经足够描述任务的内容。

**4.   下载并安装Ant**

     使用Ant前提条件，系统中已经安装JDK以及Ant。在文档编写之时，Ant的最新版本是Ant 1.7,但是为了稳定性，本文档使用版本为Ant 1.6.5.

首先下载Ant，到apache软件网站<http://www.apache.org/>。

其次，解压缩文件，放到指定的系统目录中，例如C:\Ant。

再次，将其添加到path，以便从命令行使用。(一些IDE，例如Eclipse可以不需要设置path,而通过IDE相关设置将Ant添加到path中。)

再次，设置一些环境变量指向JDK以及ANT。

最后，添加需要的可选库。

在Windows安装过程（以笔者的安装过程为例）

    下载apache-ant-1.6.5-bin.zip到本地硬盘，解压缩之后将文件夹命名为Ant,放在C:\Ant中。这个目录就是Ant主目录。

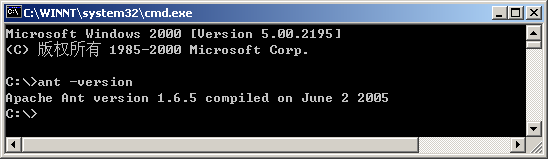
应该将主目录中的bin目录添加到path属性中，这样就可以在命令行中调用ant命令，ANT\_HOME是批处理文件所在目录的上级目录。最好明确设定。

现在许多工具已经集成了特定版本的Ant，一些操作系统甚至默认的已经安装了Ant。所以，你的系统中可能已经安装了Ant。

首先可以通过运行以下命令：  
**ant -version**  
和  
ant -diagnostics   
来确定。我们推荐您不设置CLASSPATH来运行Ant命令。如果任何版本的Ant可以从CLASSPATH加载 ，这时就会由于加载了不兼容的类而产生许多错误。

一些其他问题请参阅Ant的FAQ设置。

正常情况下，执行ant ?Cversion即可显示Ant版本，则说明安装配制成功：



**5.       运行第一个构建文件：**

首先创建一个Java工程，名为AntProject，工程中源文件和目标文件是分开的，分别为文件夹src和bin,然后创建一个Java类文件，类名为

com.neusoft.test.AntTest，只是为了测试，所以类的内容很简单：

package com.neusoft.test;

/\*\*

 \*This is just a test class.

 \*/

public class AntTest{

      public static void main(String[] args){

           for(int i=0;i<args.length;i++){

                 System.out.println(args[i]);

           }

      }

}

**然后我们在工程的路径下面建立一个构建文件build.xml，内容如下：**

<?xml version="1.0" ?>

<project name="structured" default="archive" >

<target name="init">

<mkdir dir="build/classes" />

<mkdir dir="dist" />

</target>

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

  <javac srcdir="src" destdir="build/classes"/>

</target>

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

<jar destfile="dist/project.jar"

basedir="build/classes" />

</target>

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

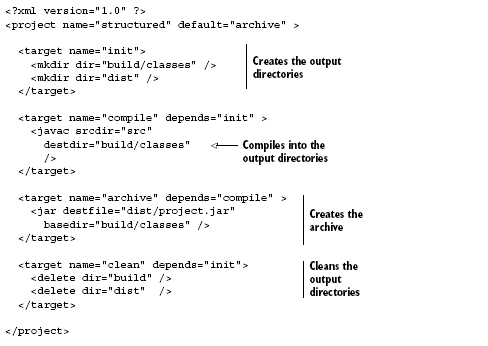
<delete dir="build" />

<delete dir="dist" />

</target>

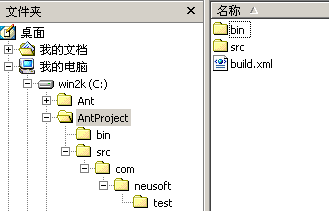
</project>

构建文件说明如下图：



关于XML的知识，请参考其他书籍，这里不做介绍。

以上创建完成后，目录结构如下图：



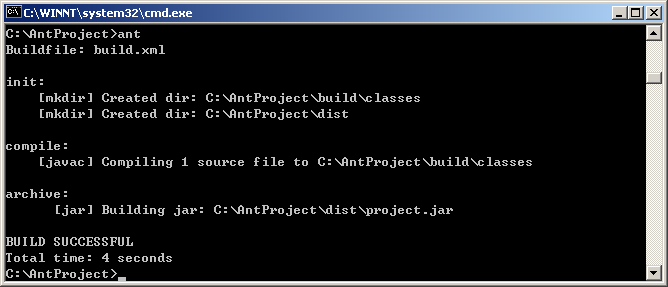
     Ant构建文件总是有一个<project>元素做为根元素，它有两 个属性，name和default，<target>元素是<project>元素的子元素，可以有多个，它有两个属 性，name和depends，<target>元素包含的元素就是一些任务元素。

<target>可以由命令行进行显示的调用，也可以在内部使用如可以直接调用ant init、ant compile等。如果不写参数，则默认的build文件是build.xml，默认的目标是<project>的default属性定义的 目标。目标的名称是唯一的，可以是任意字符串。

下面我们先运行一下这个Ant构建，再讲解其他的内容，进入工程目录，执行

ant

这里就相当于执行默认的目标，也就是<project name="structured" default="archive" >中的archive目标。



这里说明了首先初始化创建两个目录，然后编译了一个JAVA文件，然后进行了打包的操作。

**这里讲解一下如果构建失败了怎么办？**

首先有可能是XML语法书写不正确(将<target>写 成<targe>)，或者在任务执行过程中出现了错误(.java文件中包含编译错误)，或者任务名称书写错误 (将<javac>写成<javacc>)等等，这些都不是Ant的错误，不需要填写Bug Report。写XML时一定要细心，一些IDE已经有验证功能，可以很好的防止书写的错误。

**出现错误时，可以使用**

ant ?Cverbose

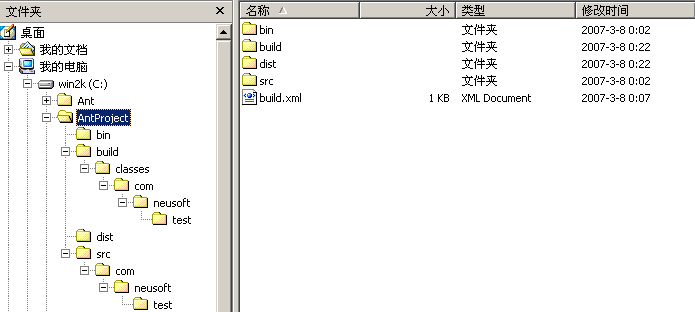
或者

ant ?Cdebug来获取更加详细的构建信息，以解决问题。

下图是使用ant ?Cverbose时的输出，使用ant ?Cdebug将获取比这更详细的信息，这里就不举例了。



本例中直接使用了软件工程中的构建结构，使用src作为源文件目录，build/class作为中间生成文件，以dist作为可发布文件。在最后把一些可执行文件可以放在bin目录中。此时目录结构如下图所示：



我们需要一种办法来确定某些任务先执行，而有些任务后执行，比如必须先编译，才能执行程序或者打包。我们在声明目标的时候，就在其依赖属性中列出其依赖关系：

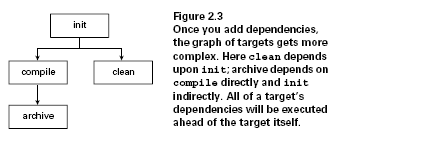
<target name="compile" depends="init" >

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

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

如果一个目标依赖与多个其他目标，需要将它们都写到依赖属性中，例如：

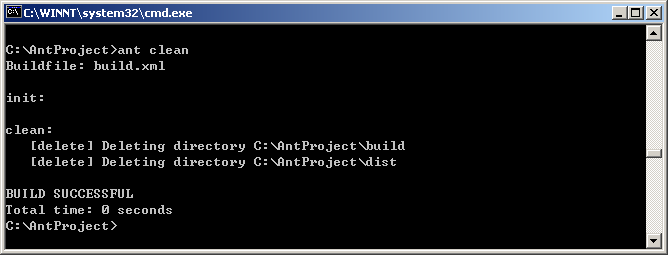
depents=”compile,test”。在我们的构建中，archive依赖于init和compile，但是我们不需要去写，因为compile已经依赖于init了。即：Ant的依赖关系是传递的，但不是自反的。



如果在执行过程中两个目标共享同一个目标，则先导目标只被执行一次。

可以通过指定目标来运行构建：

例如执行完ant后，可以执行ant clean来清理构建：



ant等价于ant archive

ant init

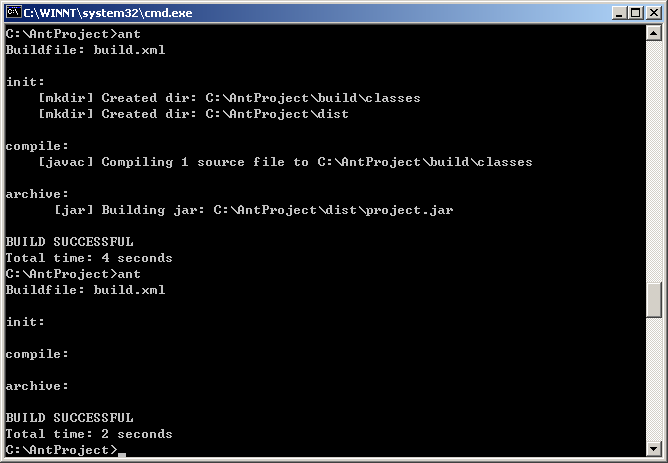
ant clean

ant compile

ant archive

ant clean archive

**当构建完成一次以后，再次执行构建会发生什么呢？**



第二次执行构建时只花了2s，相比第一次的4s。并且没有任何一个目标表示做了任何工作。

原因如下：所有的任务都检查了它们的依赖关系：

<mkdir>没有创建目录因为已经存在

<javac>比较了源文件和类文件的时间戳

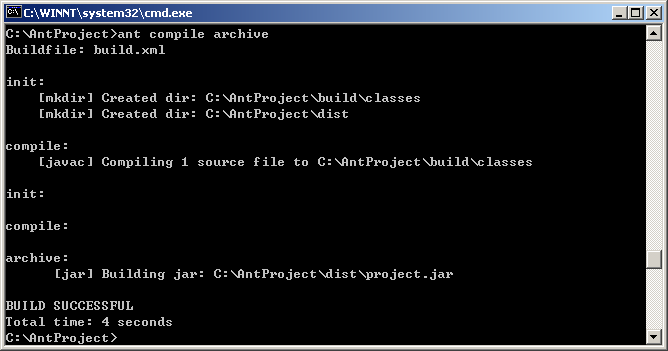
<jar>比较了要被加入文件与已经存在文件的时间

只有更新的时候才进行任务执行。

**Ant如何处理命令行上的多个目标？**

执行ant compile archive会怎么样？

先实验一下：



Ant依次执行每个目标和其依赖目标，即Ant的执行顺序是init compile init compile archive，虽然这样看起来增加了额外的工作，但是通过上面的执行过程就会发现，由于其依赖性检查的阻止，第二次的init和compile并未真正 的执行，执行时间与直接执行archive的时间是一样的。

**运行程序：**

普通执行该类的方法是：

java ?Ccp build/class com.neusoft.test.AntTest args1 args2

而我们使用Ant的任务来执行它仅仅需要增加一个任务，好处在于：

让用于执行的目标依赖与编译的目标，确保运行最新版本

易于传递复杂参数

设置classpath更方便

在Ant自身的JVM中运行，载入更快

增加一个新的目标：

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

<java classname="com.neusoft.test.AntTest"

classpath="build/classes">

<arg value="a"/>

<arg value="b"/>

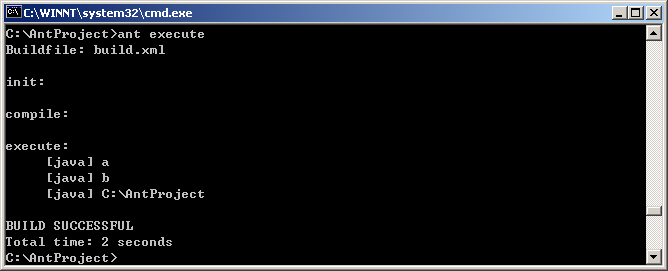
<arg file="."/>

</java>

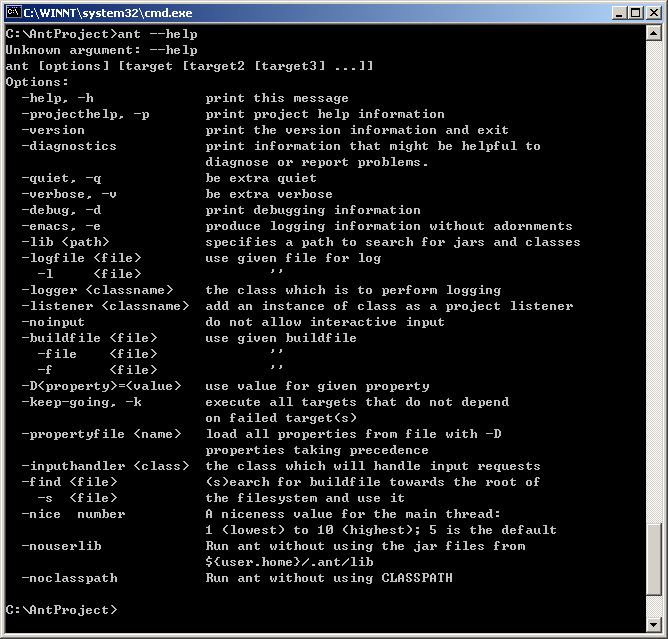
</target>

最后一个参数是file=”.”，表示传入的参数是一个目录，为文件绝对路径。

运行该目标，输出如下：



Ant命令行选项：



请参阅相关手册进行查询相关选项的功能。

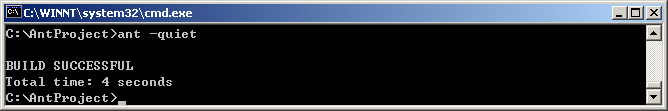
**当有多个构建文件时，可以指定构建文件：**

ant ?Cbuildfile build.xml compile

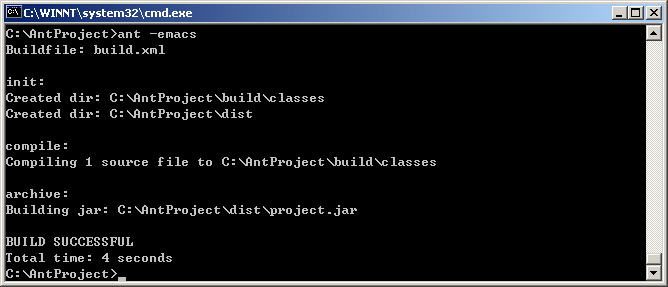
来表示执行build.xml这个构建文件中的compile目标。

控制提供的信息量：

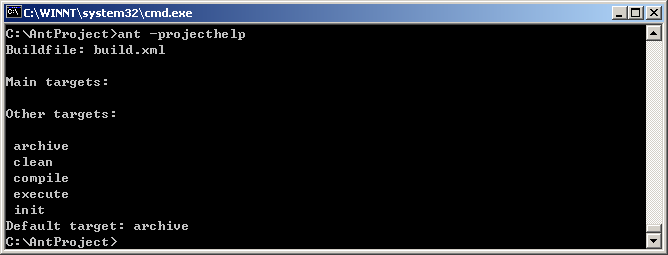
ant ?Cquiet:安静模式，不给出任何输出。



ant ?Cemacs:简单模式，不显示任务名称。



ant ?Cprojecthelp:获取项目信息。



**最终的构建文件，添加了description属性。**

<?xml version="1.0" ?>

<project name="secondbuild" **default="execute"** >

**<description>Compiles and runs a simple program</description>**

<target name="init">

<mkdir dir="build/classes" />

<mkdir dir="dist" />

</target>

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

**description="Compiles the source code"**>

<javac srcdir="src" destdir="build/classes"/>

</target>

<target name="archive" depends="compile"

**description="Creates the JAR file"**>

<jar destfile="dist/project.jar" basedir="build/classes"/>

</target>

<target name="clean" depends="init"

**description="Removes the temporary directories used"**>

<delete dir="build" />

<delete dir="dist" />

</target>

<target name="execute" depends="compile"

**description="Runs the program"**>

**<echo level="warning" message="running" />**

<java classname="org.example.antbook.lesson1.Main"

classpath="build/classes">

<arg value="a"/>

<arg value="b"/>

<arg file="."/>

</java>

</target>

</project>