1. javac

1.1. 说明

1.1.1. 命令概述

1、javac [options] [sourcefiles] [@files]

- options：命令行选项
- sourcefiles：一个或多个要编译的源文件
- @files：一个或多个对源文件进行列表的文件
- **源文件的路径不必反映包名(但是-sourcepath 的参数，作为其他源文件的依赖项，则指定的路径需要反映包名)**

2、有两种方法可将源代码文件名传递给 javac

- 如果源文件数量少，在命令行上列出文件名即可
- 如果源文件数量多，则将源文件名列在一个文件中，名称间用空格或回车行来进行分隔。然后在 javac 命令行中使用该列表文件名，文件名前冠以"@"字符

1.1.2. 文件扩展名

1、源代码文件名称必须含有".java"后缀

2、类文件名称必须含有".class"后缀

3、源文件和类文件都必须有识别该类的根名

- 例如，名为 MyClass 的类将写在名为 MyClass.java 的源文件中，并被编译为字节码类文件 MyClass.class
- 内部类定义产生附加的类文件。这些类文件的名称将内部类和外部类的名称结合在一起，例如 MyClass$MyInnerClass.class

1.1.3. 目录树结构

1、**应当将源文件安排在反映其包树结构的目录树中(非常重要!!!)**

- 例如，若将所有的源文件放在 /workspace 中，那么 com.mysoft.mypack.MyClass 的代码应该在/workspace/com/mysoft/mypack/MyClass.java 中

1.1.4. 分隔符

1、在 windows 平台

- 文件路径的分隔符为反斜杠"\\"
- **java 文件列表的分隔符为分号";"**

2、在 Linux 下平台

- 文件路径的分隔符位斜杠"/"
- **java 文件列表的分隔符为冒号":"**

1.2. 查找类型

1.2.1. 去哪查找

1、当编译源文件时，编译器常常需要它还没有识别出的类型的有关信息。对于源文件中使用、扩展或实现的每个类或接口，编译器都需要其类型信息。这包括在源文件中没有明确提及、但通过继承提供信息的类和接口

2、当编译器需要类型信息时，它将查找定义类型的源文件或类文件。编译器先在自举类及扩展类中查找，然后在用户类路径中查找

- 用户类路径通过两种途径来定义：**通过设置 CLASSPATH 环境变量**或者**使用"-classpath"命令行选项**
- 如果使用"-sourcepath"选项，则编译器在 sourcepath 指定的路径中查找源文件；否则，编译器将在用**户类路径**中查找类文件和源文件
- 可用"-bootclasspath"和"-extdirs"选项来指定不同的自举类或扩展类

1.2.2. 查到后的处理

1、成功的类型搜索可能生成类文件、源文件或两者兼有。以下是 javac 对各种情形所进行的处理

- 搜索结果只生成类文件而没有源文件：javac 使用类文件
- 索结果只生成源文件而没有类文件：javac 编译源文件并使用由此生成的类文件
- 搜索结果既生成源文件又生成类文件：确定类文件是否过时。若类文件已过时，则 javac 重新编译源文件并使用更新后的类文件。否则，javac 直接使用类文件
- 缺省情况下，只要类文件比源文件旧，javac 就认为它已过时

1.2.3. 文件列表

1、为缩短或简化 javac 命令，可以指定一个或多个每行含有一个文件名的文件。在命令行中，采用"@"字符加上文件名的方法将它指定为文件列表。当 javac 遇到以"@"字符开头的参数时，**它对那个文件中所含文件名的操作跟对命令行中文件名的操作是一样的**

1.3. 选项

1、编译器有一批标准选项，目前的开发环境支持这些标准选项，将来的版本也将支持它。还有一批附加的非标准选项是目前的虚拟机实现所特有的，将来可能要有变化。非标准选项以-X 打头

1.3.1. 标准选项

1、**-classpath [类路径参数]**

- 设置用户类路径，以分隔符进行分隔，**它将覆盖 CLASSPATH 环境变量中的用户类路径**
- 若既未指定 CLASSPATH 又未指定-classpath，则用户类路径由当前目录构成
- **类路径参数**可以**指定包含反映包名的含有".class 文件"的目录**，也可以是**JAR 归档文件**或**ZIP 归档文件**
- **如果指定的是目录：那么不会去解析.jar 文件**

```
javac -classpath .:/User/HCF/classes *.java
javac -classpath ".:/User/HCF/classes" *.java
```

2、**-d [目录]**

- 设置类文件的目标目录。如果某个类是一个包的组成部分，则 javac 将把

该类文件放入反映包名的子目录中，必要时创建目录

- 例如，如果指定"-d c:/myclasses"并且该类名叫com.mypackage.MyClass，那么类文件路径为c:/myclasses/com/mypackage/MyClass.class

➢ 若未指定"-d"选项，则 javac 将把类文件放到与源文件相同的目录中。

➢ 注意：-d 选项指定的目录不会被自动添加到用户类路径中

3、-deprecation

➢ 显示每种不鼓励使用的成员或类的使用或覆盖的说明

➢ 没有给出"-deprecation"选项的话，javac 将显示这类源文件的名称：这些源文件使用或覆盖不鼓励使用的成员或类

4、-encoding

➢ 设置源文件编码名称，例如 EUCJIS/SJIS

➢ 若未指定"-encoding"选项，则使用平台缺省的转换器

5、-g

➢ 生成所有的调试信息，包括局部变量。缺省情况下，只生成行号和源文件信息

6、-g:none

➢ 不生成任何调试信息

7、-g:{关键字列表}

➢ 只生成某些类型的调试信息，这些类型由逗号分隔的关键字列表所指定

➢ 有效的关键字有

- source：源文件调试信息
- lines：行号调试信息
- vars：局部变量调试信息

8、-nowarn

➢ 禁用警告信息

9、-O

➢ 优化代码以缩短执行时间。使用"-O"选项可能使编译速度下降、生成更大的类文件并使程序难以调试

➢ 在 JDK 1.2 以前的版本中，javac 的"-g"选项和"-O"选项不能一起使用。在 JDK 1.2 中，可以将"-g"和"-O"选项结合起来，但可能会得到意想不到的结果，如丢失变量或重新定位代码或丢失代码。"-O"选项不再自动打开"-depend"或关闭"-g"选项。同样，"-O"选项也不再允许进行跨类内嵌

10、**-sourcepath [源路径参数]**

➢ **指定用以查找类或接口定义的源代码路径(并不是指定你要编译的.java 文件的路径，而是编译你指定的.java 文件可能需要的其他依赖项，可以提供.class 文件也可以提供源文件)**

➢ 与用户类路径一样，源路径项用分隔符进行分隔，它们可以是目录、JAR 归档文件或 ZIP 归档文件

➢ **如果使用包，那么目录或归档文件中的本地路径名必须反映包名(与待编译的.java 文件不同，仅仅被编译而没有被其他.java 引用的.java 文件的路径不需要反映包名)**

➢ **注意：通过类路径查找的类，如果找到了其源文件，则可能会自动被重**

新编译(如果指定了-sourcepath 找到了某类的.java，又在 classpath 路径下找到了该类的.class，那么会重新编译该源文件成为.class 文件)

11、-verbose

- 冗长输出。它包括了每个所加载的类和每个所编译的源文件的有关信息

1.3.2. 联编选项

1、缺省情况下，类是根据与 javac 一起发行的 JDK 自举类和扩展类来编译。但 javac 也支持联编，在联编中，类是根据其它 Java 平台实现的自举类和扩展类来进行编译的。**联编时，"-bootclasspath"和"-extdirs"的使用很重要**

2、-target [版本]

- 生成将在指定版本的虚拟机上运行的类文件。缺省情况下生成与 1.1 和 1.2 版本的虚拟机都兼容的类文件。JDK 1.2 中的 javac 所支持的版本有
- 版本若为 1.1：保证所产生的类文件与 1.1 和 1.2 版的虚拟机兼容。这是缺省状态
- 版本若为 1.2：生成的类文件可在 1.2 版的虚拟机上运行，但不能在 1.1 版的虚拟机上运行

3、-bootclasspath [自举类路径]

- 根据指定的自举类集进行联编。和用户类路径一样，自举类路径项用分隔符进行分隔
- 它们可以是目录、JAR 归档文件或 ZIP 归档文件

4、**-extdirs [目录]**

- 根据指定的扩展目录进行联编。目录是以分隔符分隔的目录列表
- **在指定目录的每个 JAR 归档文件中查找类文件(只会查找 jar 文档，无视该目录下的.class 文件)**
- 注意，"-extdirs"后跟的是**包含 jar 的目录路径**，而非.jar 路径
- **不会**递归查找子文件夹中的 jar 文件

5、-Djava.ext.dirs=[目录]

- 根据指定的扩展目录进行联编。目录是以分隔符分隔的目录列表
- javac -Djava.ext.dirs=./lib Test.java
- **与联编选项中的-extdir [目录]类似**

1.3.3. 非标准选项

1、-X

- 显示非标准选项的有关信息并退出

2、-Xdepend

- 递归地搜索所有可获得的类，以寻找要重编译的最新源文件
- 该选项将更可靠地查找需要编译的类，但会使编译进程的速度大为减慢

3、-Xstdout

- 将编译器信息送到 System.out 中
- 缺省情况下，编译器信息送到 System.err 中

4、-Xverbosepath

- 说明如何搜索路径和标准扩展以查找源文件和类文件

5、-J [选项]

- 将选项传给 javac 调用的 java 启动器
- 例如，-J-Xms48m 将启动内存设为 48 兆字节
- 虽然它不以-X 开头，但它并不是 javac 的"标准选项"
- 用"-J"将选项传给执行用"Java"编写的应用程序的虚拟机是一种公共约定

1. 4. 小技巧

1、javac 并不支持递归编译指定目录中的所有.java 源文件，若要一个个指定源文件路径会非常得麻烦

- 利用 linux 的 find 命令制作 sourcefile
- find <基准路径> -name "*.java" > sourcefile.txt
- javac <其他参数> @sourcefile.txt

2. java

2.1. 用法

1、java [-options] class [args...]

2、java [-options] -jar jarfile [args...]

2.2. 选项

1、-classpath [参数]

- 设定要搜索的类的路径，可以是反映包名的包含.class 文件的目录，JAR 归档文件，ZIP 归档文件(里面都是 class 文件)，以分隔符进行分隔
- 该参数会覆盖掉所有的 CLASSPATH 的设定
- **由于所要执行的类也是要搜索的类的一部分，所以一定要把这个类的路径也放到-classpath 的设置里面**
- 若既未指定 CLASSPATH 又未指定-classpath，则用户类路径由当前目录构成

2、-Djava.ext.dirs=[包含 jar 的目录路径]

- 指定包含 jar 的目录，以分隔符进行分隔
- 不会递归查找子文件夹下的 jar

3、注意

-

2.3. 示例

1、java -Djava.ext.dirs=../lib/ -classpath classes lee.BeanTest

- 指定了 jar 包的路径
- 指定了 classpath，后面的 lee.BeanTest 以 classpath 为基准

3. jar

3.1. 说明

1、jar 是随 JDK 安装的，在 JDK 安装目录下的 bin 目录中，Windows 下文件名为 jar.exe，Linux 下文件名为 jar。它的运行需要用到 JDK 安装目录下 lib 目录中的 tools.jar 文件。不过我们除了安装 JDK 什么也不需要做，因为 SUN 已经帮我们做好了。我们甚至不需要将 tools.jar 放到 CLASSPATH 中

3.2. 用法

1、jar [-ctxu][-vfm0M] [jar 文件] [manifest 文件] [-C 目录] [文件名]...

- **其中[-ctxu]是 jar 命令的子命令，每次 jar 命令只能包含 ctxu 中的一个**
  - -c：创建新的 JAR 文件包
  - -t：列出 JAR 文件包的内容列表
  - -x：展开 JAR 文件包的指定文件或者所有文件
  - -u：更新已存在的 JAR 文件包(添加文件到 JAR 文件包中)
- [-vfm0M]中的选项可以任选，也可以不选，它们是 jar 命令的选项参数
  - -v：生成详细报 告并打印到标准输出
  - **-f：指定 JAR 文件名，通常这个参数是必须的**
  - -m：指定需要包含的 MANIFEST 清单文件
  - -O：只存储，不压缩，这样产生的 JAR 文件包会比不用该参数产生的体积大，但速度更快
  - -M：不产生所有项的清单(MANIFEST)文件，此参数会忽略-m 参数
- [jar 文件]：即需要生成、查看、更新或者解开的 JAR 文件包，它是-f 参数的附属参数
- [manifest 文件]：即 MANIFEST 清单文件(.MF)，它是-m 参数的附属参数
  - 如果使用-m 参数并指定 manifest.mf 文件，那么 manifest.mf 是作为清单文件 MANIFEST 来使用的，它的内容会被添加到.jar 中；但是，如果作为一般文件添加到 JAR 文件包中，它跟一般文件无异
- [-C 目录]：表示转到指定目录下去执行这个 jar 命令的操作。它相当于先使用 cd 命令转该目录下再执行不带-C 参数的 jar 命令，**它只能在创建和更新 JAR 文件包的时候可用**
- [文件名]...：指定一个文件或目录列表
  - 这些文件或目录就是要添加到 JAR 文件包中的文件或目录
  - **如果指定了目录，那么 jar 命令打包的时候会自动把该目录中的所有文件和子目录打入包中**

4. Ant

4.1. Ant 简介

1、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。

2、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

4.1.1. Ant 与 makefile

1、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。

2、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

4.1.2. Ant 的优点

1、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

4.1.3. Ant 开发

1、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

2、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

3、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

4、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

5、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

6、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

4.2. 元素介绍

4.2.1. <project.../>元素

1、Ant 生成文件的根元素是<project.../>，每个项目下面可以定义多个生成目标，每个生成目标以一个<target.../>元素来定义，它是<project.../>元素的子元素

2、project 元素可以有多个属性，project 元素的常见属性的含义如下

- **default**：表示默认的运行目标，这个属性是必须的，如果运行 ant.bat 命令时没有显式指定想执行的 target，Ant 将执行该 target
- **basedir**：表示项目的基准路径，生成文件中的其他相对路径都是基于该路径的
- name：指定项目名，没有太大实际作用
- description：表示项目的描述，没有太大实际作用

3、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

4.2.2. <target.../>元素

1、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 用于生成可执行文件。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

2、Target 的所有属性如下

- **name**：指定该 target 的名称，该属性是必须的，非常重要，当希望 Ant 运行指定的生成目标时，就是根据该 name 来确定生成目标的，因此同一个生成文件里面不能有两个同名的 target 元素
- depends：该属性可指定一个或多个 target 名，表示运行该 target 之前应该先运行该 depends 属性所指定的一个或多个 target
- if：该属性指定一个属性名，表示仅当设置了该属性时才执行此 target
- unless：该属性指定了一个属性名，表示仅当没有设置该属性时才执行此 target
- description：指定该 target 的描述信息

3、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。例如程序中的名为 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

4、每个生成目标又可能由一个或者多个任务序列组成，当执行某个生成目标时，实际上就是依次完成该目标所包含的全部任务

4.2.3. <property.../>元素

1、**<property.../>是<project.../>的子元素，用于定义一个或多个属性，Ant 生成文件中的属性类似于编程语言中的宏变量**，它们都具有名称和值，与编程语言不同的是，Ant 生成文件中的属性值不可改变

2、形式如下

```
<property name="builddir" value="dd"/>
```

- **如果需要获取属性值，则使用${propName}的形式**
- **注意$是特殊符号，如果要将$作为普通字符，应该使用$$**

3、<property.../>元素可以接受如下几个属性

- name：指定需要设置的属性名
- value：指定需要设置的属性值
- resource：指定属性文件的资源名称，Ant 将负责从属性文件中读取属性名和属性值
- file：指定属性文件的文件名，Ant 将负责从属性文件中读取属性名和属性值
  - **<property file="test.properties"/>**
  - 该文件的内容由一系列的"name=value"组成
  - 读取完毕之后，就可以以${name}的方式来取用值
- url：指定属性文件的 URL 地址，Ant 将负责从属性文件中读取属性名和属性值
  - **<property url ="http://www.crazyit.org/props/foo.properties"/>**
  - 与 file 类似，读取完毕之后，就可以以${name}的方式来取用值
- environment：用于指定系统环境变量的前缀，通过这种方式允许 Ant 访问系统环境变量
  - **<property environment="env"/>**
  - 那么就可以用${<前缀名>.<环境变量名>}来引用变量值
  - 例如${env.JAVA_HOME}以及${env.Path}等
- classpath：指定搜索属性文件的 classpath
- classpathref：指定搜索属性文件的 classpath 引用，该属性并不是直接给出 classpath 值，而是引用<path.../>元素定义的文件或路径集

4.2.4. <path.../>元素和<classpath.../>元素

1、**使用 Ant 编译、运行 Java 文件时常常需要引用第三方 JAR 包，这就需要使用<classpath.../>元素**

2、<path.../>和<classpath.../>元素都用于定义文件和路径集

- **classpath 通常作为其他任务的子元素**，既可引用已有的文件和路径集，也可临时定义一个文件和路径集
- **<path.../>元素作为<project.../>元素的子元素，用于定义一个独立的，有名称的文件和路径集，用于被引用**

3、<path.../>和<classpath.../>这两个元素都用于收集系列文件和路径，这两个元素都可接受如下**子元素**

1) <dirset.../>：采用模式字符串的方式指定系列目录
2) <fileset.../>：采用模式字符串的方式制定系列文件，该元素允许指定以下两个**属性**
   - dir：指定文件集里多个文件所在的基准路径，必须的属性
   - casesensitive：指定是否区分大小写，默认区分
   - 还允许使用<include.../>和<exclude.../>两个子元素来指定包含和不包含哪些文件，支持通配符
3) <filelist.../>：采用直接列出系列文件名的方式指定系列文件，该元素允许指定以下两个**属性**
   - dir：指定文件集里多个文件所在的基准路径，必须的属性
   - files：多个文件名列表，多个文件名之间以英文逗号或空白隔开
   - 还允许使用多个<file.../>子元素来指定文件列表
4) <pathelement.../>：用于指定一个或多个目录，该元素可以指定如下两个**属性**的其中一个
   - path：指定一个或多个目录(或者 JAR 文件)，多个目录或 JAR 文件之间以英文冒号或英文分号分开
   - location：指定一个目录和 JAR 文件

4、**几乎所有的 Ant 元素都可以指定两个属性：id 和 refid，其中 id 用于为该元素指定一个唯一标识符，而 refid 用于指定引用另一个元素**

4.2.5. <mkdir.../>元素

1、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

4.2.6. <jar.../>元素

1、该标签用来生成一个 JAR 文件，其属性如下

- destfile：表示 JAR 文件名
- basedir：表示被归档的文件名
- includes：表示被归档的文件模式
- exchudes：表示被排除的文件模式

4.2.7. <javac.../>元素

1、该标签用于编译一个或一组 java 文件，其属性如下：

- **srcdir**：表示源程序的目录，必须的，除非嵌套了<src.../>元素
  - 会递归编译该该目录以及其子目录下所有.java 文件，并生成反映包名

的 class 文件

- destdir：表示 class 文件的输出目录
- include：表示被编译的文件的模式
- excludes：表示被排除的文件的模式
- classpath：表示所使用的类路径
- debug：表示包含的调试信息
- optimize：表示是否使用优化
- verbose：表示提供详细的输出信息
- fileonerror：表示当碰到错误就自动停止

2、还可以含有如下子元素

- <src path="?"/>：指定需要编译的 Java 文件所在位置
  - 会递归编译该路径下所有.java 文件
  - 无论指定的 path 与包目录中间隔了几层目录，只要包含的.java 都能编译，即未必要指定 path 位于顶层包的位置
- <classpath refid="?"/>：指定编译 Java 文件所需的第三方类库所在位置

4.2.8. <java.../>元素

1、该标签用来执行编译生成的.class 文件，其属性如下：

- classname：表示将执行的类名
- jar：表示包含该类的 JAR 文件名
- classpath：所表示用到的类路径
- fork：表示在一个新的虚拟机中运行该类
- failonerror：表示当出现错误时自动停止
- output：表示输出文件
- append：表示追加或者覆盖默认文件

4.2.9. <delete.../>元素

1、该标签用于删除一个文件或一组文件，去属性如下：

- file：表示要删除的文件
- dir：表示要删除的目录
- includeEmptyDirs：表示指定是否要删除空目录，默认值是删除
- failonerror：表示指定当碰到错误是否停止，默认值是自动停止
- verbose：表示指定是否列出所删除的文件，默认值为不列出

4.2.10. <copy.../>元素

1、该标签用于文件或文件集的拷贝，其属性如下：

- file：表示源文件
- tofile：表示目标文件
- todir：表示目标目录
- overwrite：表示指定是否覆盖目标文件，默认值是不覆盖
- includeEmptyDirs：表示制定是否拷贝空目录，默认值为拷贝
- failonerror：表示指定如目标没有发现是否自动停止，默认值是停止
- verbose：表示制定是否显示详细信息，默认值不显示

4.3. Ant 的数据类型

1、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

4.3.1. argument 类型

1、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

2、以下是<arg>的所有属性

- 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
- 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
- line：表示用空格分隔的多个参数列表
- path：表示路径

4.3.2. ervironment 类型

1、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

2、<env>元素可以接受以下属性：

- 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
- path：表示环境变量的路径。Ant 会将它转换为一个本地约定
- value：表示环境变量的一个直接变量
- key：表示环境变量名
- 注意 file、path 或 value 只能取一个

4.3.3. filelist 类型

1、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

2、以下是其所有的属性：

- dir：用于计算绝对文件名的目录
- files：用逗号分隔的文件名列表
- refid：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)**

4.3.4. fileset 类型

1、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

2、以下为 fileset 的属性列表

- dir：表示 fileset 的基目录
- casesensitive：其值如果为 false，那么匹配文件名时，fileset 不是区分大

小写的，其默认值为 true

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

4.3.5. patternset 类型

1、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

2、<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

4.3.6. filterset 类型

1、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

2、主要属性如下

- 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
- 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
- id：是过滤器的唯一标志符
- refid：是对构建文件中某处定义一个过滤器的引用

4.3.7. Path 类型

1、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

- 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
- refid：是对当前构建文件中某处定义的一个 path 的引用
- path：表示一个文件或路径名列表

4.3.8. mapper 类型

1、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

- 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
- classpath：表示查找一个定制 mapper 时所用的类型路径
- classpathref：是对某处定义的一个类路径的引用
- from：属性的含义取决于所用的 mapper
- to：属性的含义取决于所用的 mapper
- 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

5. native2ascii

5.1.用法

1、native2ascii -[options] [inputfile [outputfile]]

- -reverse：将 Unicode 编码转换为本地或指定编码，不指定编码情况下，将转为本地编码
- -encoding encoding_name：转换为指定编码，encoding_name 为编码名称