Agile Java

阿左 [1] Nobody [2]

December 12, 2012

[1]感谢读者
[2]感谢国家

Contents

List of Figures

List of Tables

Part I

基础入门

Chapter 1

开发环境

1.1 开发环境配置

基本环境变量的配置：

JAVA_HOME、ANT_HOME、PATH 这三个环境变量要配置正确。

1.2 编写第一个 ant 程序

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

<project name="myproject" default="firstExample">

  <target name="firstExample">
    <echo message="This is my first targe" />
    <echo message="          os: ${os.name}" />
    <echo message="     baseidr: ${basedir}" />
    <echo message="  ant fileis: ${ant.file}" />
    <echo message=" ant version: ${ant.version}" />
    <echo message="project name: ${ant.project.name}" />
    <echo message="java version: ${ant.java.version}" />
  </target>

</project>
```

Chapter 2

起步

2.1 project 元素

一个 ant 文件中一定要有一个 project 元素。name 属性定义工作名字；default 属性代表默认启动 target；description 属性定义说明文本。

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

<project name="myproject" default="firstExample">

  <description>
    Show this description with command:
    ant -projecthelp
  </description>

</project>
```

2.2 target 元素

target 元素一定要有 name 属性。

2.2.1 depends 属性

表示当前 target 执行依赖于其他的 target 成功执行。可以有多个 depends 之间用“,”分隔。

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

<project name="myproject" default="firstExample">

  <description>
    Show this description with command:
    ant -projecthelp
  </description>

</project>
```

2.2.2 if 属性

验证执行前必须设定的属性。例如设定 JDK：

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

<!--
   ant -f build.03.xml
   ant -f build.03.xml checkFinished
-->
<project name="myproject" default="firstExample">

  <target name="showJava" if="ant.java.version">
    <echo message="java version: ${ant.java.version}" />
  </target>

  <!--
  <property name="isFinished" value="finished" />
  -->

  <target name="checkFinished" if="isFinished">
    <echo message="is finished: ${isFinished}" />
  </target>

  <target name="firstExample" depends="showJava, checkFinished">
    <echo message="always show this" />
  </target>
</project>
```

2.2.3 unless 属性

没有设置才执行

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

<!--
  ant -f build.04.xml checkUnFinished
-->
<project name="myproject" default="firstExample">
  <!--
  <property name="isFinished" value="finished" />
  -->

  <target name="checkUnFinished" unless="isFinished">
    <echo message="show this if not set ifFinished" />
  </target>

  <target name="firstExample" depends="checkUnFinished">
    <echo message="always show this" />
  </target>

</project>
```

2.3 property 元素

property 可以看作为参数或是变量的定义。可以在构建文件中通过 property 元素建立，也可以在构建文件外建立一个 build.property 文件来存放。

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

<project name="myproject" default="firstExample">

  <!-- reference properties file -->
  <property file="build.properties" />

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

  <target name="init">
    <!-- get current time -->
    <tstamp/>
  </target>
```

```
14
15    <target name="firstExample" depends="init">
16      <echo message="author name:${author.name}" />
17      <echo message="author nick:${author.nickname}" />
18      <echo message="currentTime:${DSTAMP}" />
19    </target>
20
21  </project>
```

2.4 task 元素

task 元素是一系列完成指定功能的脚本和程序。比如自带的编译用的 javac、打包用的 jar、建立目录的 mkdir 等。用户也可以编写自己的 task。

2.5 命令行使用 ant

2.5.1 指定文件

-buildfile 或-file 或-f 都可以。

2.5.2 显示任务描述

-projecthelp。

2.5.3 指定变量的值

-D<property>=<value>

Chapter 3

核心类型

3.1 断言类型：Asseritions Type

断言类型指定哪些类需要让 ant 工具执行 java 断言。enableSystemAssertions 属性设定是否允许系统断言，默认为 unspecified。

断言类型内还可以定义 enable 类型与 disable 类型，通过这两个类型的 class 属性与 package 属性来设置指定的类或是包是否可以执行断言。

允许所有的用户代码执行断言：

```
<assertions>
  <enable />
</assertions>
```

只允许 Test 这个类执行断言：

```
<assertions>
  <enable class="Test"/>
</assertions>
```

允许一个包下所有的类都执行断言：

```
<assertions>
  <enable package="org.apache"/>
</assertions>
```

更复杂的定义：

```
<assertions>
  <enable  package="org.apache"/>
```

```
    <disable package="org.apache.tools.ant"/>
    <enable    class="org.apache.tools.ant.Main"/>
  </assertions>
```

引用一个已经存在的定义：

```
  <assertions id="project.assertions">
    <disable package="org.apache.test"/>
  </assertions>

  <assertions refid="project.assertions" />
```

3.2 匹配模式：PatternSet

3.2.1 包含与排除

匹配格式通过以下四个属性指定包含与排除：

include 属性：可以指定多个模式用逗号或空格分隔，相当于一个列表。

includesfile 属性：指定具体包含的文件。可以引用在 properties 文件中指定的内容。

exclude 属性：和上面的相反。

excludesfile 属性：和上面的相反。

3.2.2 过滤的模式

以上四个属性都有三个属性来匹配指定的内容：

name 属性：文件名、路径或格式。

if 属性：指定的属性有定义则生效。

unless 属性：指定的属性没有定义则生效。

3.2.3 例子

类名中不包含 Test 的类：

```
  <patternset id="non.test.sources">
    <include name="**/*.java" />
    <exclude name="**/*Test.java" />
  </patternset>
```

引用已经定义的模式：

```
<patternset refid="non.test.sources">
```

可以包含多个定义：

```
<patternset id="sources">
  <include name="std/**/*.java" />
  <include name="prof/**/*.java" if="professional" />
  <exclude name="**/*Test.java" />
</patternset>
```

指定文件名中包含了"some-file" 的文件：

```
<patternset>
  <includesfile name="some-file" />
</patternset>
```

也可以简写成下面的格式：

```
<patternset includesfile="some-file" />
```

配合 if 条件的形式：

```
<patternset>
  <includesfile name="some-file" if="isInclue" />
</patternset>
```

3.3 匹配目录：DirSet

DirSet 类似于 PatternSet，用于匹配目录。必填属性 dir 用于指定一个目录。

DirSet 可以引用已经存在的 PatternSet，也可以直接使用 include、includesfile、exclude、excludesfile 属性。

另外，DirSet 还有 casesensitive 属性来设置是否大小写敏感；followsymlinks 属性来设置采用操作系统差异（如 linux 与 windows 的路径符号）等。

定义目录集合的例子：

```
<dirset dir="${build.dir}">
  <include name="**/*.java" />
  <exclude name="**/*Test.java" />
</dirset>

```

```
6   <dirset dir="${build.dir}">
7     <patternset id="non.test.classes">
8       <include name="**/*.java" />
9       <exclude name="**/*Test.java" />
10    </patternset>
11  </dirset>
12
13  <dirset dir="${build.dir}" >
14    <patternset refid="non.text.classes" />
15  </dirset>
```

3.4 匹配目录：FileSet

3.4.1 设置匹配

FileSet 匹配符合的文件，它不仅包含 DirSet 的属性外，还有一些其他的属性：

file 属性：指定单个文件，fileSet 定义中至少要 dir 属性或是 file 属性。

defaultexcludes：当值为 yes 时默认忽略指定的文件（如版本控制文件），常用的模式有：

**/*~, **/#*#, **/.#*, **/%*%, **/._*,

/CVS, **/CVS/, **/.cvsignore,

/SCCS, **/SCCS/, **/vssver.scc,

/.svn, **/.svn/, **/.DS_Store

3.4.2 过滤筛选：selector

selector 可以看作是 FileSet 中的一个元素。有以下几种常见的过滤工具：

3.4.3 内容过滤：contains

只选择包含 text 属性定义字符串的文件。

text 属性：文件包含的字符串，不能为空。

casesenitive 属性：大小写敏感。

ignorewhitespace 属性：忽略空白字符。

```
<fileset dir="${doc.path}" includes="**/*.html">
  <contains text="script" casesentive="no" />
</fileset>
```

3.4.4 时间过滤：date

datetime 属性：指定时间格式为：MM/DD/YYYY HH:MM AM 。例如：09/09/2006 10:10 am

milis 属性：指定毫秒时间。datetime 与 milis 两个必选一个。

when 属性：文件修改时间的比较方式，默认为 equals。其他值还有：before 与 after。

granularity 属性：允许的时间误差毫秒数。

pattern 属性：指定 datettime 是否兼容 Java 中的 SimpleDateFormat 格式。

checkdirs 属性：是否检查文件目录创建时间，默认为 false。

```
<fileset dir="${doc.path}" includes="**/*.jar">
  <date datetime="01/01/2001 12:00 AM" when="before" />
</fileset>
```

3.4.5 比较过滤：depend

比较两个目录下同名的文件，选择最后修改的那个。

targetdir 属性：指定进行比较的目录。

granularity 属性：允许的时间误差毫秒数。

比较 1.4 版与 1.5 版中有修改过的源文件。

```
<fileset dir="${ant.1.5}/src/main" includes="**/*.java">
  <depend targetdir="${ant.1.4}/src/main" />
</fileset>
```

3.4.6 深度过滤：depth

min 最小与 max 最大。

```
<fileset dir="${doc.path}" includes="**/*">
  <depend max="1" />
</fileset>
```

3.4.7 差异过滤：different

比较目录下的文件中否有不同（当前目录和一层子目录）。比较的内容有：

1）一个地方有另一个地方没有。

2）文件大小不同。

3）ignoreFileTimes 不为 off，且文件修改时间不同。

4）ignoreContents 为 true，内容不同。

属性有：targetdir、granularity（允许的时间误差范围）、ignoreFileTimes、ignoreContents。

```
1  <fileset dir="${ant.1.5}/src/main" includes="**/*.java">
2    <different targetdir="${ant.1.4}/src/main" />
3  </fileset>
```

3.4.8 文件名过滤：filename

主要属性：name（文件名匹配）、casesensitive、negate（反选，选中不匹配的）。

例如，选择所有的 css 样式文件：

```
1  <fileset dir="${doc.path}" includes="**/*">
2    <filename name="**/*.css" />
3  </fileset>
```

3.4.9 目录过滤：paresent

选择一个与当前目录对应的目录（targetdir），找出指定目录中不存在或是两个目录中都存在的文件。主要属性有：

present 属性：值为 srconly 本目录有面 targetDir 没有的文件；值为 both 选中两个目录都有的文件。

```
1  <fileset dir="${ant.1.5}/src/main" includes="**/*.java">
2    <present present="only" targetdir="${ant.1.4}/src/main" />
3  </fileset>
```

3.4.10 正则过滤：containsregexp

```
<fileset dir="${doc.path}" includes="**/*">
  <containsregexp expression="[4-6]\.[0-9]" />
</fileset>
```

3.4.11 大小过滤：size

value 属性：值。

units 属性：单位。以 1000 为进制的 K，M，G。或是以 1024 为单位的 Ki，Mi，Gi。默认为 bytes。

when 属性：more，less，equal。

```
<fileset dir="${jar.path}">
  <patternset>
    <include name="**/*.jar" />
  </patternset>
  <size value="4" unites="Ki" when="more" />
</fileset>
```

3.4.12 类型过滤：type

指定是文件（file）还是目录（dir）。

```
<fileset dir="${src}">
  <type type="dir" />
</fileset>
```

3.5 文件列表：FileList

```
<filelist id="docfiles" dir="${doc.src}"
  files="foo.xml,bar.xml" />

<filelist id="docfiles2" refid="docfiles"/>

<filelist id="docfiles3" dir="${doc.src}" >
  <file name="foo.xml" />
  <file name="bar.xml" />
</filelist>
```

3.6 文件过滤器：FilterSet

过滤器可以在对文件进行复制或移动时对文件内容进行替换操作。

3.6.1 主要属性

1) begintoken 属性：用一个字符 (默认为 @) 指定过滤字符串的开始。如：@DATE@

2) endtoken 属性：定义结束。

3) recurse 属性：是否查找多个替换标记，默认为 true。

3.6.2 替换版权与日期的例子

```
<target name="ex01" >
  <tstamp>
    <format property="now" pattern="yyyy/MM/dd HH:mm:ss" />
  </tstamp>
  <copy todir="build" filtering="true">
    <fileset dir="src">
      <include name="**/*.java" />
    </fileset>
    <!-- find @COPYRIGHT@ and @BUILD_DATE@ -->
    <filterset>
      <filter token="BUILD_DATE" value="${now}" />
      <filter token="COPYRIGHT" value="Jade Shan" />
    </filterset>
  </copy>
</target>
```

可引用的版本：

```
<!-- get current time -->
<tstamp>
  <format property="now" pattern="yyyy/MM/dd HH:mm:ss" />
</tstamp>

<!-- find @COPYRIGHT@ and @BUILD_DATE@ -->
<filterset id="date.copyright.filterset">
  <filter token="BUILD_DATE" value="${now}" />
  <filter token="COPYRIGHT" value="Jade Shan" />
</filterset>
```

```
11
12   <target name="firstExample" >
13     <copy todir="build" filtering="true">
14       <fileset dir="src">
15         <include name="**/*.java" />
16       </fileset>
17       <filterset refid="date.copyright.filterset" />
18     </copy>
19   </target>
```

3.6.3 把替换的内容放在属性文件中

要替换的文本：

```
1 This is the test sample.
2 The message will be replaced: @MESSAGE@
```

属性文件：

```
1 MESSAGE=HELLO
```

构建文件：

```
1   <filterset id="prop.message">
2     <filtersfile file="abc.properties" />
3   </filterset>
4
5   <target name="filterProp" >
6     <copy todir="build" filtering="true">
7       <fileset dir="src">
8         <include name="src.txt" />
9       </fileset>
10      <filterset refid="prop.message" />
11    </copy>
12  </target>
```

3.7 属性集合：PropertySet

定义一套可以组其他标签引用的属性集合。

3.7.1 属性与功能

dynamic 属性：是否动态地加载属性。

negate 属性：取反默认为 false，如果为 true 代表返回除 PropertySet 外的属性。

3.7.2 引用已经存在的属性

Propertyref 类型可以引用一个已经定义的 property。主要属性有：

name 属性：引用的名字。

prefix 属性：引用指定开头的多个属性。

regex 属性：用正则匹配。

builtin 属性：引用 ant 内建的属性，值为 all 时表示所有内建属性。

3.7.3 使用属性集合的例子

组建一个属性集合：

```
<property name="lib1" value="lib1path" />
<property name="lib2" value="lib2path" />

<propertyset id="projectset1">
  <projectref name="lib1" />
  <projectref name="lib2" />
</properties>
```

projectset 间相互引用的例子：

```
<propertyset id="properties-starting-with-foo">
  <propertyref prefix="foo" />
</propertyset>

<propertyset id="properties-starting-with-bar">
  <propertyref prefix="bar" />
</propertyset>

<propertyset id="my-set">
  <propertyset refid="properties-starting-with-foo" />
  <propertyset refid="properties-starting-with-bar" />
</properties>
```

3.8 文件映射：mapper

用来定义文件之间的对应关系的，主要属性有：

type 属性：定义一个实现的类型，可以用现成的也可以自己实现一个。

classname 属性：通过类名指定一个实现类型。type 和 classname 一定要选一个。

classpath 属性：查找类的路径。

classpathref 属性：引用已经有的 path 作为 classpath。

from 属性：源文件的位置。

to 属性：目标文件的位置。

对于文件映射，不同的实现类提供了不同的映射实现方法，以下各小节是现有的实现。每个类的调用都可以通过类名调用与 type 属性调用两种方法来写：

3.8.1 identity

源文件与目标文件同名。只取文件名，忽略路径：

```
<!--
   A.java                          -> A.java
   foo/bar/B.java                  -> B.java
   C.properties                    -> C.properties
   Classes/dir/dri2/A.properties -> A.properties
-->
<mapper type="identity" />
<identitymapper />
```

3.8.2 flatten

忽略目录把文件放到一个压缩文件中：

```
<!--
   A.java                          -> archive.tar
   foo/bar/B.java                  -> archive.tar
   C.properties                    -> archive.tar
   Classes/dir/dri2/A.properties -> archive.tar
-->
<mapper type="flatten" />
<flattenmapper />
```

3.8.3 glob

匹配路径与文件名：

```
<!--
   A.java                          -> A.java.bak
   foo/bar/B.java                  -> foo/bar/B.java.bak
   C.properties                    -> Ignored
   Classes/dir/dri2/A.properties -> Ignored
-->
<mapper type="glob" from="*.java" to="*.java.bak"/>
<globmapper         from="*.java" to="*.java.bak" />
```

3.8.4 merge

把源文件打包到压缩文件中：

```
<!--
   A.java                          -> archive.tar
   foo/bar/B.java                  -> archive.tar
   C.properties                    -> archive.tar
   Classes/dir/dri2/A.properties -> archive.tar
-->
<mapper type="merge" to="archive.tar" />
<mergemapper         to="archive.tar" />
```

3.8.5 regexp

通过正则来映射：

```
<!--
   A.java                          -> A.java.bak
   foo/bar/B.java                  -> foo/bar/B.java.bak
   C.properties                    -> Ignored
   Classes/dir/dri2/A.properties -> Ignored
-->
<mapper type="regexp" from="^(.*)\.java$$" to="\1.java.bak" />
<regexpmapper         from="^(.*)\.java$$" to="\1.java.bak" />
```

3.8.6 package

替换目录名称。

```
<!--
   org/apache/tools/ant/util/PackageMapperTest.java
    -> Test-org.apache.tools.ant.util.PackageMapperTest.xml
   org/apache/tools/ant/util/Helper.java
    -> ignored
-->
<mapper type="package" from="*Test.java" to="TEST-*Test.xml" />
<packagemapper         from="*Test.java" to="TEST-*Test.xml" />
```

3.8.7 composite

多个 mapper 都对源文件进行操作。Composite Mapper 不能通过 Mapper 类型的 type 属性定指定：

```
<!--
   foo/bar/A.java -> foo/bar/A.java
   foo/bar/A.java -> foo.bar.A
-->
<compositemapper>
  <identitymapper />
  <packagemapper from="*.java" to="" />
</compositemapper>
```

3.8.8 chained

包含多个 mapper，源文件依次经过每个 mapper 操作。

```
<!--
   foo/bar/A.java -> new/path/A.java1
   foo/bar/A.java -> new/path/A.java2
   foo/bar/B.java -> new/path/B.java1
   foo/bar/B.java -> new/path/B.java2
-->
<chainedmapper>
  <flattenmapper/>
  <globmapper from="*" to="new/path/*" />
  <mapper>
    <globmapper from="*" to="*1" />
```

```
12      <globmapper from="*" to="*2" />
13    </mapper>
14  </chainedmapper>
```

3.8.9 filtermapper

对文件名进行过滤：

```
1  <!--
2     foo\bar\A.java -> new/path/A.java1
3  -->
4  <filtermapper>
5    <replacestring from="\" to="/" />
6  </filtermapper>
```

3.9 压缩文件：zip

有两种压缩文件：

1) 当使用 src 属性时，目录下的文件会以.zip 文件格式进行组织。

2) 当使用 dir 属性时，目录下的文件会以文件系统形式进行组织。

3.9.1 属性与功能

prefix 属性：文件路径前缀，符合的会被选中。

fullpath 属性：包含全路径。

src 属性：用于替代当前的目录位置。

filemode 属性与 dirmode 属性：文件的权限，如 linux 下的权限 777。

3.9.2 例子

以 zip 格式压缩 htdocs/manual 目录下的所有文件。存放到 docs/user-guide 目录下。

同时添加 ChangeLog27.txt 文件到 zip 文件的 docs 目录下。

这个 zip 文件还包含 example.zip 文件。example.zip 文件包含 docs/examples 目录及其子目录下的所有 html 文件。

```
<zip desfile="${dist}/manual.zip" >
  <zipfileset dir="htdocs/manual"
    prefix="docs/user-guide" />
  <zipfileset dir="." includes="ChangeLog27.txt"
    fullpath="docs/ChangeLog.txt" />
  <zipfileset prefix="docs/examples" includes="**/*.html"
    src="example.zip" />
</zip>
```

3.10 过滤链与过滤读取器：FilterChains and FilterReader

一组有序的 FilterReader 组成 FilterChains，用户可以实现自己的 FilterReader。

FilterReader 通过 classname 属性指定实现类。

在 ant 任务 concat、copy、loadFile、loadProperties、move 中都可以直接使用 FilterChain 进行过滤操作。

```
<copy file="${src.file}" tofile="${dest.file}" >
  <!-- define the chains -->
  <filterchain>
    <filterreader classname="aa.FilterReader">
      <param name="foo" value="bar" />
    </filterreader>
    <!-- define your own reader -->
    <filterreader classname="bb.FilterReader">
      <!-- class path -->
      <classpath>
        <pathelement path="${classpath}" />
      </classpath>
      <param name="bar" value="blee" />
      <param name="abc" value="cadd" />
    </filterreader>
  </filterchain>
</copy>
```

3.11 定制与扩展

用户可以定制的有：conditions、selecters 和 filters 类型。

3.11.1 定制条件判断：Condition

实现一个判断字符串是大写的条件判断：

```
1  package exp.ant;
2
3  import org.apache.tools.ant.BuildException;
4  import org.apache.tools.ant.taskdefs.condition.Condition;
5
6  public class AllUpperCaseCondition implements Condition{
7    private String value;
8
9    public boolean eval() {
10     if(value == null){
11       throw new BuildException("value attribute is not set");
12     }
13     return value.toUpperCase().equals(value);
14   }
15
16   public void setValue(String value) {
17     this.value = value;
18   }
19 }
```

构建文件中通过 typedef 导入，然后使用定义的类。

```
1    <typedef name="alluppercase"
2      classname="exp.ant.AllUpperCaseCondition"
3      classpath="${src.class}" />
4
5    <condition property="allupper">
6      <alluppercase value="THIS IS ALL UPPER CASE" />
7    </condition>
```

3.11.2 定制选择器：Selector

```
1  package exp.ant;
2
```

```
import java.io.File;
import org.apache.tools.ant.types.selectors.FileSelector;

public class JavaSelector implements FileSelector {

  public boolean isSelected(File b, String filename, File f) {
    return filename.toLowerCase().endWith(".java");
  }
}
```

```
  <typedef name="javaselector"
    classname="exp.ant.JavaSelector"
    classpath="${src.class}" />

  <copy todir="to">
    <fileset dir="src">
      <javaselector/>
    </fileset>
  </copy>
```

ant 已经提供了一个 BaseSelector 做了一些预处理功能：setError(String errMsg) 可提供出错信息；validate() 方法会在 isSelected() 前进行验证。

3.11.3 定制过滤器：Filter

```
package exp.ant;

// ... other packages
import org.apache.tools.ant.types.filters.ChainableReader;

public class RevomeOddCharacters implements ChainableReader{

  public Reader chain(Reader reader) {
    return new BaseFilterReader(reader) {
      int count = 0;
      public int read() throws IOException{
        // other
      }
    }
  }

}
```

直接通过类名调用：

```
<copy file="${src.file}" todir="to">
  <filterchain>
    <filterreader classname="exp.ant.RemoveOddCharacters" />
  </filterchain>
</copy>
```

Chapter 4

核心任务

4.1 任务调用（Ant Task）

有一种类型的 Ant 任务就叫“Ant 任务”（Ant Task）。这种类型的 Task 可以去调用另一个 Ant 项目。

4.1.1 主要属性

指定要执行的文件（antfile 属性）：

```
<?xml version="1.0"?>
<!-- call another ant project projectB -->
<project name="projectA" default="callProjectB">
  <target name="callProjectB">
    <echo message="In projectA calling projectB" />
    <ant antfile="subfile/projectB.xml" />
  </target>
</project>
```

```
<?xml version="1.0"?>
<!-- call another ant project projectB -->
<project name="projectB" default="init">
  <target name="init">
    <echo message="In project B" />
  </target>
</project>
```

指定文件所在目录（dir 属性）：

```
<?xml version="1.0"?>
<!-- call another ant project projectB -->
<project name="projectA" default="callProjectB">
  <target name="callProjectB">
    <echo message="In projectA calling projectB" />
    <ant antfile="projectB.xml" dir="subfile"/>
  </target>
</project>
```

调用指定的任务 (target 属性):

```
<?xml version="1.0"?>
<!-- call another ant project projectB -->
<project name="projectA" default="callProjectB">
  <target name="callProjectB">
    <echo message="In projectA calling projectB" />
    <ant antfile="subfile/projectB2.xml" target="target2"/>
  </target>
</project>
```

```
<?xml version="1.0"?>
<!-- call another ant project projectB -->
<project name="projectB" default="init">
  <target name="init">
    <echo message="In project B" />
  </target>
  <target name="target2">
    <echo message="In project B, target2" />
  </target>
</project>
```

指定输出流 (output 属性):

```
<?xml version="1.0"?>
<project name="projectA" default="callProjectB">
  <target name="callProjectB">
    <echo message="In projectA calling projectB" />
    <ant antfile="subfile/projectB.xml" output="out.log" />
  </target>
</project>
```

被调用文件可以使用调用它文件中的属性 (inheritAll 属性): 类似于 Java 中的继承属性，默认值为“true”。

被调用文件可以使用调用它文件中的 reference 任务 (inheritRefs 属性): reference 任务的作用是把当前属性复制到被调用的任务中使用。它有两个可配置的属性:

1) refid 属性: 当前 project 中的属性 id。 2) torefid 属性: 指定被调用的 project 中的引用 id。

例: 把当前 project 中的 path1 属性传递给被调用的 project 的属性 path2 使用:

```
1  <reference refid="path1" torefid="path2" />
```

4.1.2 实例: 一个任务整合了多个子任务

实际工作中, 一个大的项目会被独立为几个小的项目:

```
1  <?xml version="1.0"?>
2  <project name="projectA" default="buildAll">
3
4    <property file="default.properties" />
5
6    <target name="buildAll" depends="buildSub1, buildSub2">
7      <echo message="buildAll" />
8    </target>
9
10   <target name="buildSub1">
11     <ant antfile="subfile/sub1.xml" target="readme" />
12   </target>
13
14   <target name="buildSub2">
15     <ant antfile="subfile/sub2.xml" target="readme" >
16       <property name="testParam" value="hello" />
17     </ant>
18   </target>
19
20 </project>
```

```
1  <?xml version="1.0"?>
2  <project name="sub1" default="init">
3    <target name="init">
4      <echo message="In sub1" />
5    </target>
6    <target name="readme">
7      <echo message="this is sub1: readme" />
```

```
    <echo message="properties from parent file: ${path1}" />
  </target>
</project>
```

```
<?xml version="1.0"?>
<project name="sub2" default="init">
  <target name="init">
    <echo message="In sub2" />
  </target>
  <target name="readme">
    <echo message="this is sub2: readme" />
    <echo message="param from parent file: ${testParam}" />
  </target>
</project>
```

4.2 执行过程中调用其他 target (AntCall Task)

4.2.1 主要属性

三个主要属性：target、inheritAll、inheritRefs 参照前一部分 Ant Task。

4.2.2 例子

```
<?xml version="1.0"?>
<project name="projectA" default="t1">
  <target name="init">
    <echo message="init" />
  </target>
  <target name="t1">
    <echo message="target A start" />
    <antcall target="t2" />
    <echo message="target A end" />
  </target>
  <target name="t2" depends="init">
    <echo message="target B" />
  </target>
</project>
```

4.3 调用系统命令（Apply/ExecOn Task）

4.3.1 主要属性

executable：指定要执行的命令，不带命令行参数。必填。

dest：执行命令的目标文件位置。

spawn：不输出日志，默认为 false 表示输出日志。

dir：在哪个目录下执行这个命令。

relative：是否支持相对路径。默认 false 不支持。

forwardslash：文件路径是否支持斜线分隔符。

os：支持这个命令的操作系统。

output：输出重定向。

error：错误输出重定向。

logError：错误输出重定向到 ant 的日志中去。

append：追加内容面不是覆盖已经有的文件。默认为 false。

outputproperty：指定输出定向到的属性名字（定义一个文件则输出到文件中）。

errorproperty：错误重定向到属性的名字。

input：从指定文件中读取属性，可以以命令执行过程中引用。

inputstring：把指定的字符串传递给执行的命令。

resultproperty：执行后存放结果。

timeout：设定执行的超时时间。

failonerror：出错是中否中断。

failifexecutionfails：不能执行程序时中断。默认 true。

skipemptyfilesets：如果目录中没有文件，则跳过执行。

parallel：如果为 ture，则构建命令只执行一次，并把附加的文件作为命令参数。如果为 false 则每一个附加文件都会执行一次这个命令。默认为 false。

type：说明参数类型：文件（file）、目录（dir）、路径（path）。默认为 file。

newenvironment：如果当前环境变量被声明，则不传递旧的环境变量。默认为 false。

vmlauncher：默认为 true。通过 java 虚拟机的特性来执行构建文件；如果为 false 则通过操作系统本身的脚本来执行。

resolveExecutable：默认为 false。如为 true，命令会在 project 的根目录下执行。在 UNIX 或 Linux 下只允许用户在自己的路径下执行这个命令，要把这个属性设为 false。

maxparallel：最大的平行值，指定一次执行源文件的最大数目。如果小于 0 表示没有限制 (默认)。

addsourcefile：自动添加源文件名到执行命令中，默认为 true。

verbose：输出命令执行时的概要信息，默认为 false 不输出。

ignoremissing：忽略不存在的文件，默认为 true。

force：是否通过 timestame 来对 target 文件进行对比。默认为 false。

4.3.2 主要参数

FileSet、FileList、DirSet、Arg(<arg> 指定参数)。

Mapper (可能指定 dest 属性的文件的映射关系)。

SrcFile (在<arg> 参数后使用，指定源文件)。

TargetFile (与 srcFile 作用相似，用于指定目录文件的参数)。

Env (环境变量)。

4.3.3 例子

调用“ls”命令，参数“-l”。分别排除和包含“properties”文件。

```
1  <?xml version="1.0"?>
2  <project name="projectA" default="t1">
3
4    <fileset id='prop' dir=".">
5      <patternset>
6        <include name="**/*.properties" />
7      </patternset>
8    </fileset>
9
10   <target name="t1" depends="t2">
11     <apply executable="ls">
12       <!-- arg -->
13       <arg value="-l" />
14       <!-- file set -->
15       <fileset dir=".">
16         <patternset>
```

```
17            <exclude name="**/*.properties" />
18          </patternset>
19        </fileset>
20      </apply>
21    </target>
22
23    <target name="t2" >
24      <apply executable="ls">
25        <!-- arg -->
26        <arg value="-l" />
27        <!-- file set -->
28        <fileset refid="prop" />
29      </apply>
30    </target>
31
32  </project>
```

4.3.4 使用 Mapper、SrcFile 的例子

以一个编译 C 源文件的例子：对于每一个比.o 文件更加新的.c 文件，执行：

```
cc -c -o targetfile sourcefile
```

在这个文件中用.o 文件的名称替换 targetfile，用.c 文件的名称替换 sourcefile。

```
1  <apply executable="cc" dest="src/c" parallel="false">
2    <arg value="-c" />
3    <arg value="-c" />
4    <targetfile />
5    <srcfile />
6    <fileset dir="src/c" includes="*.c" />
7    <mapper type="glob" from="*.c" to="*.o" />
8  </apply>
```

4.4 改变文件的权限 (Chmod Task)

4.4.1 主要属性

file、dir、include、excludes、perm (新的权限)。

parallel：是否为每个文件单独执行 chmod。默认为 true。

type：只改文件的权限 (file)；只改目录的权限 (dir)；都改权限 (both)

maxparallel：最大的平行值，指定一次执行源文件的最大数目。如果小于 0 表示没有限制 (默认)。

verbose：输出命令执行时的概要信息，默认为 false 不输出。

defaultexcludes：当值为 yes 时默认忽略指定的文件 (如版本控制文件)，常用的模式有：

```
**/*~, **/#*#, **/.#*, **/%*%, **/._*,
**/CVS, **/CVS/**, **/.cvsignore,
**/SCCS, **/SCCS/**, **/vssver.scc,
**/.svn, **/.svn/**, **/.DS_Store
```

4.4.2 例子

所有 cgi 或 old 结尾的文件，private_ 开头的目录以及内部的文件。

```
<?xml version="1.0"?>
<project name="projectA" default="t1">

  <target name="t2">
    <chmod file="others/aa.sh" perm="ugo+rx" />
  </target>


  <target name="t3">
    <chmod file="others/aa.sh" perm="700" />
  </target>

  <target name="t1">
    <chmod perm="700" type="file" >

      <fileset dir="others" >
        <include name="**/*.cgi" />
        <include name="**/*.old" />
      </fileset>

      <dirset dir="others" >
        <include name="**/private_*" />
      </dirset>

    </chmod>
```

```
26    </target>
27
28  </project>
```

4.5 删除文件 (Delete Task)

4.5.1 主要属性

file、dir、verbose、quiet (当文件不存在时，不显示提示信息)、failonerror、includes、includesfile、excludes、excludesfile (不推荐使用)、defaultexcludes (不推荐使用)。

deleteonexit：当文件存在时才删除。默认为 false。

includeemptydirs：当使用 FileSet 类型时是否删除空的目录。

4.5.2 例子

```
1  <?xml version="1.0"?>
2  <project name="projectA" default="t1">
3
4
5    <target name="t2">
6      <delete file="others/tu.jar" />
7    </target>
8
9    <target name="t3">
10     <delete dir="others/libs" />
11   </target>
12
13   <!-- delete all *.bak file, and the empty folder -->
14   <target name="t1">
15     <delete includeEmptyDirs="true" >
16       <fileset dir="others/build" includes="**/*.bak" />
17     </delete>
18   </target>
19
20 </project>
```

4.6 输出信息 (Echo Task)

4.6.1 主要属性

message、file、append (追加到原有文件后面)、level (error、warning、info、verbose、debug)。

4.6.2 例子

```
<echo message="hello" />
<echo message="hello" file="logs/01.log" />
```

4.7 创建目录 (Mkdir Task)

```
<property name="dist" value="dist" />
<property name="tmp" value="tmp" />
<mkdir dir="${dist}" />
<mkdir dir="${tmp}" />
```

4.8 移动文件与目录 (Move Task)

4.8.1 主要属性

file、tofile、todir、overwrite、failonerror、verbose、

preservelastmodified：移动后文件的时间与源文件相同。

filtering：允许使用过滤符号。

flatten：没有目录结构，都在一级目录下。

includeEmptyDirs：忽略空目录。

encoding：过滤器的编码方式。

outputencoding：输出文件的编码。

granularity：允许文件修改时间的误差。默认 0，DOS 系统为 2。

4.8.2 例子

```
<?xml version="1.0"?>
<project name="projectA" default="t1">

  <!-- move file -->
  <target name="t2">
    <move file="others/a.old" tofile="toDir/a.new" />
  </target>

  <!-- move all file under foler "others/build"
     to "toDir" -->
  <target name="t3">
    <move todir="toDir" >
      <fileset dir="others/build" />
    </move>
  </target>

  <!-- move all file under foler "others/build"
     to "toDir" -->
  <!-- after ant 1.6.3 -->
  <target name="t4">
    <move file="others/build" tofile="toDir" />
  </target>

  <!-- move the folder "others/build" to "toDir" -->
  <target name="t5">
    <move file="others/build" todir="toDir" />
  </target>

  <!-- move file and rename to *.bak-->
  <target name="t1">
    <move todir="toDir" includeemptydirs="false" >
      <fileset dir="others/build">
        <exclude name="**/*.bak" />
      </fileset>
      <mapper type="glob" from="*" to="*.bak" />
    </move>
  </target>

</project>
```

4.9 压缩 zip 文件 (Zip Task)

4.9.1 主要属性

distfile、basedir、compress (是否压缩默认 true)、encoding、fileonly、includes、includesfile、excludes、excludesfile、defaultexcludes、

update：覆盖目标文件。

whenempty：当没有可压缩的文件时结果为：报错 (fail)、忽略 (skip)、创建空 zip 文件 (create)。

duplicate：文件重复时：默认为覆盖 (add)、跳过 (preserve)、报错 (fail)。

roundup：文件修改时间是否采用一下个连续的秒数。

keepcompression：已经压缩的文件保持原先的压缩数据。

comment：zip 文件的备注。

4.9.2 例子

任务 t1 直接按目录生成了压缩文件。任务 t2 指定了不同文件在生成的压缩文件中的位置。任务 t3 把一些其他的压缩文件放到了产生的压缩文件中。

```
<?xml version="1.0"?>
<project name="projectA" default="t1">

  <!-- -->
  <target name="t2">
    <zip destfile="zipDir/manual1.zip"
      basedir="manual" includes="**/*.html"
      excludes="**/todo.html" />
  </target>

  <!-- -->
  <target name="t3">
    <zip destfile="zipDir/manual2.zip" >
      <!-- define dist folder in zip file -->
      <zipfileset dir="manual/readme" prefix="docs/user-guide" />
      <!-- define dist file in zip file -->
      <zipfileset dir="manual" includes="todo.html"
        fullpath="docs/todo.html" />
      <!-- load file from another zip file -->
      <zipfileset src="manual/others.zip" includes="**/*.html"
```

```
21          prefix="docs/examples" />
22      </zip>
23    </target>
24
25    <!-- -->
26    <target name="t1">
27      <zip destfile="zipDir/manual3.zip" >
28        <zipfileset dir="manual" prefix="docs/user-guide" />
29        <!-- put other zip file in dist zip file -->
30        <zipgroupfileset dir="." includes="examples*.zip" />
31      </zip>
32    </target>
33
34  </project>
```

4.10 加载属性文件 (LoadProperties Task)

4.10.1 主要属性

srcFile、resource (同 srcFile)、encoding、classpath、classpathref。

4.10.2 例子

把复制的目标和来源都定义在属性文件中。

```
1  copy.src=others
2  copy.dist=toDir
```

加载过程中只加载“copy”开头的属性。

```
1  <?xml version="1.0"?>
2  <project name="projectA" default="t1">
3
4    <loadproperties srcFile="copy.properties">
5      <filterchain>
6        <linecontains>
7          <contains value="copy." />
8        </linecontains>
9      </filterchain>
10   </loadproperties>
11
```

```
12   <target name="t1">
13     <copy todir="${copy.dist}" >
14       <fileset dir="${copy.src}" />
15     </copy>
16   </target>
17
18 </project>
```

4.11 定义日期格式 (Tstamp Task)

4.11.1 主要属性

property：定义名称，可以在以后引用。

pattern：格式。同 java 中的 SimpleDateFormat。

timezone：时区。同 java 中的 Timezone。

unit：设定与当时时间相差的单元，可为：millisecond、second、minute、hour、day、week、month、year。

offset：设定与当时时间差，单元由 unit 设定。

locale：地区设置。

4.11.2 例子

```
1  <?xml version="1.0"?>
2  <project name="projectA" default="t1">
3
4    <tstamp>
5      <format property="today_UK" pattern="d-MMMM-yyyy" locale="en" />
6    </tstamp>
7
8    <tstamp>
9      <format property="today_CN" pattern="d-MMMM-yyyy" locale="zh" />
10   </tstamp>
11
12   <tstamp>
13     <format property="touch.time" pattern="MM/dd/yyyy hh:mm aa"
14       offset="5" unit="hour" />
15   </tstamp>
16
```

```
17
18   <target name="t1">
19     <echo message="${DSTAMP}" />
20     <echo message="${today_UK}" />
21     <echo message="${today_CN}" />
22     <echo message="${touch.time}" />
23   </target>
24
25 </project>
```

Part II

工具整合

Part III

实践