Ant 介绍

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。

下边开始一步步的带你走进 ant 的世界

ant 安装

1、到http://ant.apache.org/bindownload.cgi

下载 Ant，我使用的版本是 1.70

解压缩下载文件到你的工作目录，这里以 d:\ant_home 为例

2,如图新增系统变量

新增系统变量：ANT_HOME，内容：d:\ant_home\apache-ant-1.70

在 PATH 环境变量中加入 Ant 的 bin 目錄：%ANT_HOME%\bin

如果要让 Ant 能支持 JUnit，需要直接將 JUnit 的 junit.jar 放置在 Ant 的 lib 目錄，并记得改变 CLASSPATH 中原先有关于 JUnit 的设定，例如：%ANT_HOME\lib\junit.jar，虽然也有其它的方式可以設定，但這是最快最簡單的方法。

如果是 Windows 2000/XP，请在[系统内容/高级/环境变量]中设置[系统变量]，以完成以上的设定，例如：

3、测试安装

CMD 进入命令行界面，运行 Ant

出现如下提示，说明安装成功

E:\srcgen\webwork>ant

Buildfile: build.xml does not exist!

Build failed

二、第一个 ant 脚本

使用 ant 来达成目的，完成一件事情的实例

1、目的：

- 编写一个程序
- 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

我们用文本编辑器编写一个 HelloWorld 如下

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,然后用 ant 完成剩下的步骤。

建立一个 build.xml 文件,内容如下

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

解释上边的配置文件

<?xml version="1.0" encoding="UTF-8" ?>
build.xml 中的第一句话，没有实际的意义,指定一下 encoding，几乎所有的 xml 的第一行

<project name="HelloWorld" default="run" basedir=".">
</project>
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。

<property name="src" value="src"/>
类似程序中的变量，为什么这么做想一下变量的作用

<target name="compile" depends="init">
<javac srcdir="${src}" destdir="${dest}"/>
</target>
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

如我们的计划
编译：
<target name="compile" depends="init">
<javac srcdir="${src}" destdir="${dest}"/>
</target>

做 jar 包:
<target name="build" depends="compile">
<jar jarfile="${hello_jar}" basedir="${dest}"/>
</target>
运行：
<target name="run" depends="build">
<java classname="test.ant.HelloWorld" classpath="${hello_jar}"/>
</target>
为了不用拷贝，我们可以在最开始定义好目标文件夹，这样 ant 直接把结果就放在目标文件夹中了
新建文件夹:
<target name="init">
<mkdir dir="${dest}"/>
</target>
为了更多一点的功能体现，又加入了两个 target

删除生成的文件

```
<target name="clean">
    <delete dir="${dest}" />
    <delete file="${hello_jar}" />
</target>
```

再次运行，这里显示了如何在一个 target 里边调用其他的 target

```
<target name="rerun" depends="clean,run">
    <ant target="clean" />
    <ant target="run" />
</target>
```

好了，解释完成了，下边检验一下你的 ant 吧
新建一个 src 的文件夹，然后把 HelloWorld.java 按照包目录放进去
做好 build.xml 文件
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

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 build

ant 中的每一个任务都可以这样调用 ant + target name
好了，这样一个简单的 ant 任务完成了。

整合 ant

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

```
<?xml version="1.0" encoding="UTF-8" ?>
<project name="main" default="build" basedir=".">
    <property name="bin" value="${basedir}\bin" />
    <property name="src1" value="${basedir}\src1" />
    <property name="src2" value="${basedir}\src2" />
    <property name="src3" value="${basedir}\src3" />
    <target name="init">
        <mkdir dir="${bin}" />
    </target>
    <target name="run">
        <ant dir="${src1}" target="run" />
        <ant dir="${src2}" target="run" />
        <ant dir="${src3}" target="run" />
    </target>
    <target name="clean">
        <ant dir="${src1}" target="clean" />
        <ant dir="${src2}" target="clean" />
        <ant dir="${src3}" target="clean" />
    </target>
    <target name="build" depends="init,call">
        <copy todir="${bin}">
            <fileset dir="${src1}">
                <include name="*.jar" />
            </fileset>
            <fileset dir="${src2}">
                <include name="*.jar" />
            </fileset>
            <fileset dir="${src3}">
                <include name="*.jar" />
            </fileset>
```

```
            </copy>
        </target>
        <target name="rebuild" depends="build,clean">
            <ant target="clean" />
            <ant target="build" />
        </target>
</project>
```

ok 你的任务完成了。

Ant 进阶

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

```
<?xml version="1.0" encoding="UTF-8" ?>
<project name="main" default="build" basedir=".">
    <property file="all.properties" />
    <property name="bin" value="${basedir}\bin" />
    <target name="init">
        <mkdir dir="${bin}" />
    </target>
    <target name="run">
        <ant dir="${src1}" target="run" />
        <ant dir="${src2}" target="run" />
        <ant dir="${src3}" target="run" />
    </target>
    <target name="clean">
        <ant dir="${src1}" target="clean" />
        <ant dir="${src2}" target="clean" />
        <ant dir="${src3}" target="clean" />
    </target>
    <target name="build" depends="init,call">
        <copy todir="${bin}">
            <fileset dir="${src1}">
                <include name="*.jar" />
            </fileset>
```

```
            <fileset dir="${src2}">
                <include name="*.jar" />
            </fileset>
            <fileset dir="${src3}">
                <include name="*.jar" />
            </fileset>
        </copy>
    </target>
    <target name="rebuild" depends="build,clean">
        <ant target="clean" />
        <ant target="build" />
    </target>
    <target name="test">
        <ant dir="${src1}" target="test" />
        <ant dir="${src2}" target="test" />
        <ant dir="${src3}" target="test" />
    </target>
</project>
```

如果你自己看的话你会看到这样一个 target

```
<target name="test">
    <ant dir="${src1}" target="test" />
    <ant dir="${src2}" target="test" />
    <ant dir="${src3}" target="test" />
</target>
```

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

但是有一种更好的方法。

你可以写一个 include.xml 文件，内容如下

```
<?xml version="1.0" encoding="UTF-8" ?>
<property name="src" value="src"/>
<property name="dest" value="classes"/>
<target name="test" >
    <ant target="run" />
</target>
```

然后更改你三个小组的 build.xml 文件,每个里边加入如下内容

```
<!--include a xml file ,it can be common propery ,can be also a target    -->
<!DOCTYPE project [
<!ENTITY share-variable SYSTEM "file:../include.xml">
]>
&share-variable;
```

变成如下的样子

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

而且不会让三个组的 build.xml 变得更复杂。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--include a xml file ,it can be common propery ,can be also a target    -->
<!DOCTYPE project [
<!ENTITY share-variable SYSTEM "file:../include.xml">
]>
<project name="HelloWorld" default="run" basedir=".">
    <!--use the include    -->
    &share-variable;
    <!--defined the property-->
    <!--via include
    <property name="src" value="src"/>
    <property name="dest" value="classes"/>
    -->
    <property name="hello_jar" value="hello1.jar"/>
    <!--define the op-->
    <target name="init">
        <mkdir dir="${dest}"/>
    </target>
    <target name="compile" depends="init">
        <javac srcdir="${src}" destdir="${dest}"/>
    </target>
    <target name="build" depends="compile">
        <jar jarfile="${hello_jar}" basedir="${dest}"/>
    </target>
    <target name="run" depends="build">
        <java classname="test.ant.HelloWorld" classpath="${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>
```

Ant 常用 task

好了，看完上边的内容，你应该知道怎么构建一个 ant 的骨架，但具体做起来才才发现很多东西不知道该如何写。下边介绍一下常用的 ant task。在 ant 手册中跟这些叫 core task

这里打乱一下顺序，不按照文档中的顺序来介绍，而是按照需求来介绍。

使用 classpath

```
<target>
    <javac>
      <classpath refid="project.class.path"/>
    </javac>
 </target>
```

设置 classpath

```
<classpath id=" project.class.path" >
      <pathelement path="${classpath}"/>
      <fileset dir="lib">
        <include name="**/*.jar"/>
      </fileset>
      <pathelement location="classes"/>
      <dirset dir="build">
        <include name="apps/**/classes"/>
        <exclude name="apps/**/*Test*"/>
      </dirset>
      <filelist refid="third-party_jars"/>
 </classpath>
```

输出信息

<echo message="xxx" />

引入一个 xml 文件

```
<import file="../common-targets.xml"/>
```

拷贝一个文件

```
<copy file="myfile.txt" tofile="mycopy.txt"/>
```

拷贝一个文件到指定目录

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

拷贝一个目录到另一个目录

```
<copy todir="../new/dir">
    <fileset dir="src_dir"/>
</copy>
```

拷贝一个文件集合到一个目录

```
<copy todir="../dest/dir">
    <fileset dir="src_dir">
      <exclude name="**/*.java"/>
    </fileset>
  </copy>

  <copy todir="../dest/dir">
    <fileset dir="src_dir" excludes="**/*.java"/>
  </copy>
```

拷贝一个文件集合到一个目录，同时建立备份文件

```
<copy todir="../backup/dir">
    <fileset dir="src_dir"/>
    <globmapper from="*" to="*.bak"/>
  </copy>
```

拷贝一个集合的文件到一个目录，并替换掉@ TITLE @

```
<copy todir="../backup/dir">
    <fileset dir="src_dir"/>
    <filterset>
      <filter token="TITLE" value="Foo Bar"/>
    </filterset>
  </copy>
```

拷贝一个目录下的东西到另一个目录下，includes 加入，excludes 排除

```
<copydir src="${src}/resources"
           dest="${dist}"
           includes="**/*.java"
           excludes="**/Test.java"
  />
```

执行程序

```
<target name="help">
  <exec executable="cmd">
    <arg value="/c"/>
    <arg value="ant.bat"/>
    <arg value="-p"/>
  </exec>
</target>
```

出现一个错误

```
<fail>Something wrong here.</fail>或者
<fail message=" Something wrong here." />
```

打 jar 包

```
<jar destfile="${dist}/lib/app.jar" basedir="${build}/classes"/>
```

或者

```
<jar destfile="${dist}/lib/app.jar"
       basedir="${build}/classes"
       includes="mypackage/test/**"
       excludes="**/Test.class"
  />
```

运行 jar 包：

```
<java classname="test.Main">
     <arg value="-h"/>
     <classpath>
       <pathelement location="dist/test.jar"/>
       <pathelement path="/Users/antoine/dev/asf/ant-core/bootstrap/lib/ant-launcher.jar />
     </classpath>
</java>
```

或者设置一下运行的 jvm 的最大内存，来运行

```
<java classname="test.Main"
           dir="${exec.dir}"
           jar="${exec.dir}/dist/test.jar"
           fork="true"
           failonerror="true"
           maxmemory="128m"
           >
     <arg value="-h"/>
     <classpath>
       <pathelement location="dist/test.jar"/>
       <pathelement path="/Users/antoine/dev/asf/ant-core/bootstrap/lib/ant-launcher.jar />
     </classpath>
</java>
```

编译程序

```
<javac srcdir="${src}"
         destdir="${build}"
         classpath="xyz.jar"
         debug="on"
         source="1.4"
  />
```

制作 Javadoc

```
<javadoc ... >
     <doclet name="theDoclet"
             path="path/to/theDoclet">
        <param name="-foo" value="foovalue"/>
        <param name="-bar" value="barvalue"/>
     </doclet>
  </javadoc>
```

定义一个新的 task 类库

```
<taskdef name="myjavadoc" classname="com.mydomain.JavadocTask"/>
```

运行 sql

```
<sql
    driver="org.database.jdbcDriver"
    url="jdbc:database-url"
    userid="sa"
    password="pass"
    >
insert
into table some_table
values(1,2,3,4);

truncate table some_other_table;
</sql>
```

解压缩

```
<unzip src="apache-ant-bin.zip" dest="${tools.home}">
    <patternset>
        <include name="apache-ant/lib/ant.jar"/>
    </patternset>
    <mapper type="flatten"/>
</unzip>
```

压缩

```
<zip destfile="${dist}/manual.zip"
       basedir="htdocs/manual"
       includes="api/**/*.html"
       excludes="**/todo.html"
  />
```

还有很多，可以参考 ant 手册的 ant core task

打 war 包：

```
<war destfile="myapp.war" webxml="src/metadata/myapp.xml">
  <fileset dir="src/html/myapp"/>
  <fileset dir="src/jsp/myapp"/>
  <lib dir="thirdparty/libs">
    <exclude name="jdbc1.jar"/>
  </lib>
  <classes dir="build/main"/>
  <zipfileset dir="src/graphics/images/gifs"
              prefix="images"/>
</war>
```

在 ant 中控制流程(if else)

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." name="learn" default="run">
    <property name="db.type" value="oracle" />
    <import file="properties.xml" />
    <taskdef resource="net/sf/antcontrib/antcontrib.properties"
classpath="${ant-contrib.jar}" />
    <target name="run">
        <if>
            <equals arg1="${db.type}" arg2="mysql" />
            <then>
                <echo message="The db is mysql" />
            </then>
            <else>
                <echo message="the db is oralce" />
            </else>
        </if>
    </target>
</project>
```

实例分析:
从开源框架中拿出一部分来分析一下：
我接触的一些开源程序的 ant 中,appfuse 是比较复杂的，Jpetstore 是最简单的。
这里拿一个最简单的出来，然后一路注释过去。（想挑战一下的去看 appfuse 的。）

```
<project name="JPetStore" default="all" basedir=".">
<!—定义属性→
  <property name="web" value="../web"/>
  <property name="src" value="../src"/>
  <property name="test" value="../test"/>
  <property name="lib" value="../lib"/>
  <property name="devlib" value="../devlib"/>

  <property name="wars" value="wars"/>
  <property name="warfile" value="${wars}/jpetstore.war"/>

  <property name="webapp" value="webapp"/>
  <property name="weblib" value="${webapp}/WEB-INF/lib/"/>
  <property name="webclasses" value="${webapp}/WEB-INF/classes/"/>

  <property name="reports" value="reports"/>
  <property name="junitreports" value="${reports}/junit/"/>
  <property name="coveragereports" value="${reports}/coverage/"/>

  <property name="work" value="work"/>
  <property name="classes" value="${work}/classes/"/>
  <property name="testclasses" value="${work}/testclasses/"/>
  <property name="instrumentedclasses" value="${work}/instrumentedclasses/"/>


<!—定义 classpath →
  <path id="classpath">
    <pathelement location="${instrumentedclasses}"/>
    <pathelement location="${classes}"/>
    <pathelement location="${testclasses}"/>
    <pathelement location="${src}"/>
    <pathelement location="${test}"/>
    <fileset dir="${lib}" includes="**/*.jar"/>
    <fileset dir="${devlib}" includes="**/*.jar"/>
  </path>
<!—引入新的 ant task 包,可以使用一些 ant 默认没有的 tag 和 task→
  <taskdef name="junit" classname="org.apache.tools.ant.taskdefs.optional.junit.JUnitTask"/>
  <taskdef resource="emma_ant.properties" classpathref="classpath"/>
```

```
<!—定义删除文件的 target，清除→
  <target name="clean">
    <delete dir="${reports}"/>
    <delete dir="${work}"/>
    <delete dir="${wars}"/>
    <delete dir="${webapp}"/>
    <delete file="coverage.ec"/>
    <delete>
      <fileset dir="${src}">
        <include name="**/*.class"/>
      </fileset>
    </delete>
    <delete>
      <fileset dir="${test}">
        <include name="**/*.class"/>
      </fileset>
    </delete>
  </target>

<!—创建所需的文件夹 →
  <target name="prepare" depends="clean">
    <mkdir dir="${classes}"/>
    <mkdir dir="${testclasses}"/>
    <mkdir dir="${instrumentedclasses}"/>

    <mkdir dir="${junitreports}"/>
    <mkdir dir="${coveragereports}"/>

    <mkdir dir="${webclasses}"/>
    <mkdir dir="${weblib}"/>

    <mkdir dir="${wars}"/>
  </target>
<!—编译→
  <target name="compile" depends="prepare">
    <javac srcdir="${src}" destdir="${classes}" deprecation="on" debug="on">
      <classpath refid="classpath"/>
    </javac>
    <javac srcdir="${test}" destdir="${testclasses}" deprecation="on" debug="on">
      <classpath refid="classpath"/>
    </javac>
  </target>
```

```
<!--########## →
  <target name="coverage.instrument" depends="compile">
    <emma enabled="yes">
      <instr instrpath="${classes}"
        destdir="${instrumentedclasses}"
        metadatafile="./coverage.ec"
        merge="true"
        >
      </instr>
    </emma>
  </target>


<!—测试 →
  <target name="test" depends="coverage.instrument">
    <junit printsummary="yes" haltonfailure="no">
      <classpath refid="classpath"/>

      <formatter type="xml"/>

      <batchtest fork="yes" todir="${junitreports}">
        <fileset dir="${test}">
          <include name="**/*Test*.java"/>
          <exclude name="**/AllTests.java"/>
        </fileset>
      </batchtest>
    </junit>
  </target>

<!—测试报告 →
  <target name="test.report" depends="test" >
    <junitreport todir="${junitreports}">
      <fileset dir="${junitreports}">
        <include name="TEST-*.xml"/>
      </fileset>
      <report format="frames" todir="${junitreports}"/>
    </junitreport>
  </target>
```

```
<target name="coverage.report" depends="test" >
    <emma enabled="yes">
        <report sourcepath="${src}"
            sort="+block,+name,+method,+class"
            metrics="method:70,block:80,line:80,class:100"
            >
            <fileset dir="./">
                <include name="*.ec"/>
            </fileset>
            <html outfile="${coveragereports}/coverage.html"
                depth="method"
                columns="name,class,method,block,line"
                />
        </report>
    </emma>
</target>
```

```
<!—拷贝到一起→
<target name="assemble" depends="test">
    <copy todir="${webclasses}">
        <fileset dir="${src}">
            <exclude name="**/*.java"/>
            <exclude name="**/*.class"/>
        </fileset>
    </copy>
    <copy todir="${webclasses}">
        <fileset dir="${classes}"/>
    </copy>
    <copy todir="${weblib}">
        <fileset dir="${lib}"/>
    </copy>
    <copy todir="${webapp}">
        <fileset dir="${web}"/>
    </copy>
</target>

<target name="assembleWebapp">
    <copy todir="${webapp}">
        <fileset dir="${web}"/>
    </copy>
</target>
```

```
<!—打 war 包 →
  <target name="war" depends="assemble">
    <jar jarfile="${warfile}">
      <fileset dir="${webapp}">
        <include name="**/*"/>
      </fileset>
    </jar>
  </target>

  <target name="all" depends="test.report, coverage.report, war"/>
</project>
```

如何继续学习

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 使用 cvs 的实例

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，但是在开远框架的 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

```
<?xml version="1.0" encoding="utf-8"?>
<project>
 <property name="cvsroot" value=":pserver:wang:@192.168.1.2:/cvsroot"/>
 <property name="basedir" value="/tmp/testant/"/>
 <property name="cvs.password" value="wang"/>
 <property name="cvs.passfile" value="${basedir}/ant.cvspass"/>
 <target name="initpass">
    <cvspass cvsroot="${cvsroot}" password="${cvs.password}" passfile="${cvs.passfile}"/>
 </target>
 <target name="checkout" depends="initpass">
    <cvs cvsroot="${cvsroot}" command="checkout" cvsrsh="ssh" package="myproject" dest="${basedir}" passfile="${cvs.passfile}"/>
 </target>
</project>
```

QA：

1、 如果执行 ant 过程中出现 Outofmemory 的错误怎么办？

答：加大内存，设置环境变量 ANT_OPTS=-Xms128m -Xmx256m