Programmazione Orientata agli Oggetti in Linguaggio Java

Strumenti di Sviluppo:
Ant
Parte c

versione 1.0

G. Mecca – Università della Basilicata – mecca@unibas.it

Sommario

- Funzionalità Avanzate
- Controllo del Processo di Build
  - ⇨Utilizzo di File di Proprietà
  - ⇨Build di Debug e di Release
  - ⇨Numero di Versione
- Riuso dei Buildfile
  - ⇨Il Task ant
  - ⇨Il Task import
- Altre Operazioni

Funzionalità Avanzate

- Ant è un linguaggio complesso
  - ⇨consente un controllo molto raffinato del processo di build
  - ⇨consente forme di riuso dei buildfile
  - ⇨consente di effettuare moltissime operazioni diverse attraverso i suoi task predefiniti
  - ⇨consente di definire nuovi task per estendere quelli predefiniti
  - ⇨nel seguito discutiamo questi aspetti

Controllo del Processo di Build

- Una caratteristica fondamentale di Ant
  - ⇨è possibile controllare il comportamento dei buildfile attraverso le proprietà
  - ⇨Ant fornisce funzionalità avanzate a questo proposito
- Metodi per impostare le proprietà
  - ⇨il task property
  - ⇨il flag –D
  - ⇨caricamento di file .properties

Controllo del Processo di Build

>> morraCinese\build-semplice.xml

- Un esempio
  - ⇨il progetto morraCinese
- Varie caratteristiche interessanti
  - ⇨utilizzo di file di properties per i riferimenti al di fuori del progetto
  - ⇨introduzione di build di debug e build di release
  - ⇨gestione delle versioni

Utilizzo di File di Proprietà

- All'interno del progetto
  - ⇨vengono usati esclusivamente percorsi relativi a ${basedir}
  - ⇨questo consente, una volta acquisita la cartella di progetto, di riprodurre le operazioni di build senza problemi su altre macchine
- I riferimenti alle librerie
  - ⇨sono esterni al progetto e devono necessariamente essere assoluti

Utilizzo di File di Proprietà

- Cartelle contenenti librerie
  - ⇨tipicamente due categorie
- Librerie di terze parti di utilizzo generale
  - ⇨es: junit.jar; ${lib.dir} c:\Programmi\lib
- Librerie dello sviluppatore
  - ⇨tipicamente altri progetti Java gestiti dallo stesso sviluppatore o nello stesso progetto; es: utilita.jar
  - ⇨${mylib.dir} e:\codice\lib

Utilizzo di File di Proprietà

- Per riprodurre il processo di build
  - ⇨è necessario che le stesse cartelle siano disponibili sulla nuova macchina
  - ⇨ma possono non essere nella stessa posiz.
- Idea
  - ⇨i valori delle due proprietà sono scritti in un file .properties che viene caricato dal buildfile
  - ⇨per cambiare i riferimenti basta cambiare il file .properties senza toccare il buildfile

Utilizzo di File di Proprietà

- Acquisire proprietà da un file .properties

```
 # File di properties dependent.properties
lib.dir=c:/Programmi/lib
mylib.dir=e:/codice/lib
```

```
<target name="init-dependent" depends="">
      <property file="./dependent.properties" />
      <echo message="lib.dir = ${lib.dir}" />
      <echo message="mylib.dir = ${mylib.dir}" />
</target>
```


Build di Debug e di Release

- Abbiamo già visto
  - ⇨che esistono due tipologie di build
- Build di debug
  - ⇨includono informazioni di debug; il codice non viene ottimizzato durante la compilaz.
- Build di release
  - ⇨escludono informazioni di debug ed ottimizzano il codice

Build di Debug e di Release

- Il task critico
  - ⇨il task di compilazione
  - ⇨nel task javac bisogna stabilire il valore per l'attributo debug e per l'attributo optimize
- La soluzione tipica
  - ⇨utilizzare una proprietà relase.build passata attraverso la linea di comando
  - ⇨assegnare valori diversi alle proprietà relative al livello di debug e di ottimizzazione

Build di Debug e di Release

- Per assegnare valori diversi
  - ⇨posso eseguire target diversi a seconda del valore della proprietà release.build
- Attributi if ed unless di un target
  - ⇨viene specificata una proprietà
  - ⇨if: il target viene eseguito solo se la proprietà è stata precedentemente impostata
  - ⇨unless: il target viene eseguito solo se la proprietà NON è stata impostata

```
<target name="init-release" if="release.build" depends="">
    <echo message="build di release" />
    <property name="build.debug" value="off" />
    <property name="build.debuglevel" value="lines" />
    <property name="build.optimize" value="on" />
</target>

<target name="init-debug" unless="release.build" depends="">
    <echo message="build di debug" />
    <property name="build.debug" value="on" />
    <property name="build.debuglevel" value="lines,vars,source" />
    <property name="build.optimize" value="off" />
</target>

<target name="init" depends="init-folders, init-classpath, init-release, init-debug">...

<target name="compile" depends="init, prepare">
    <javac srcdir="${src.dir}" destdir="${build.dir}"
        classpathref="compile.classpath"
        debug="${build.debug}"
        debuglevel="${build.debuglevel}"
        optimize="${build.optimize}">
    </javac>
</target>
```


Build di Debug e di Release

- Per ottenere una build di debug
  - ⇨comportamento standard
  - ⇨basta eseguire ant senza parametri aggiuntivi (build.release non viene impostata)
- Per ottenere una build di release
  - ⇨ant –Dbuild.release=true
- Nota: il valore di build.release è irrilevante
  - ⇨per if ed unless conta solo se è stato impostato o meno

ant –f build-semplice.xml

ant –f build-semplice.xml –Dbuild.release=true

Numero di Versione

- Un ultimo utilizzo delle proprietà
  - costruire un numero di versione aggiornato per ciascuna distribuzione
- Numero di versione
  - tipicamente fatto di tre parti
  - major.version
  - minor.version
  - build.numer
  - es: 2.1.3456

Numero di Versione

- Opzionalmente
  - può contenere anche un “timestamp”, ovvero un’indicazione della data o dell’ora in cui è stato prodotto
- Task di ant
  - il task buildnumber
  - il task tstamp

Numero di Versione

- Il task buildnumber
  - si applica ad un file di properties
  - cerca la proprietà build.number
  - se non esiste la aggiunge con valore 0
  - altrimenti ne incrementa il valore di 1
  - se viene eseguito prima di ogni distribuzione consente di avere un numero di build sempre aggiornato

Numero di Versione

- Il task tstamp
  - inizializza tre proprietà sulla base della data e dell'ora in cui viene eseguito
  - DSTAMP: data in formato aaaammgg
  - TSTAMP: ora in formato hhmm
  - TODAY: data in formato mese giorno anno

Numero di Versione

```
<target name="init-version" depends="init">
    <!-- Timestamp -->
    <tstamp />
    <echo message="${DSTAMP}" />
    <echo message="${TSTAMP}" />
    <echo message="${TODAY}" />
    <!-- Versione -->
    <buildnumber file="build.properties" />
    <property file="build.properties" />
</target>
```

```
#Build Number for ANT. Do not edit!
#Fri Dec 31 13:01:06 CET 2004
build.number=61
major.version=1
minor.version=0
```


```
<target name="dist" depends="rebuild, init-version, jar, javadoc">
    <zip zipfile="${dist.dir}/${ant.project.name}-java-
                ${major.version}.${minor.version}.${build.number}-${DSTAMP}.zip">
      <zipfileset dir="${src.dir}" prefix="${ant.project.name}/src" />
      <zipfileset dir="${test.src.dir}" prefix="${ant.project.name}/src/test"
                includes="**/*.java" />
      <zipfileset dir="${test.data.dir}" prefix="${ant.project.name}/src/test/data" />
      <zipfileset dir="${misc.dir}" prefix="${ant.project.name}/varie" />
      <zipfileset dir="${dist.dir}" prefix="${ant.project.name}" includes="*.jar" />
      <zipfileset dir="${mylib.dir}" prefix="${ant.project.name}/lib" includes="utilita.jar" />
      <zipfileset dir="${lib.dir}" prefix="${ant.project.name}/lib" includes="jdom.jar" />
      <zipfileset dir="${javadoc.dir}"
                prefix="${ant.project.name}/javadoc/${ant.project.name}" />
      <zipfileset dir="${misc.dir}" prefix="${ant.project.name}" includes="*.bat" />
    </zip>
</target>

<target name="create-manifest" depends="init, init-version">
    <manifest file="${build.dir}/MANIFEST.MF" >
      <attribute name="Built-by" value="${user.name}" />
      <attribute name="Implementation-Version"
                value="${major.version}.${minor.version}.${build.number}" />
      <attribute name="Main-Class" value="${main.class}" />
      <attribute name="Class-Path" value="lib/jdom.jar lib/utilita.jar" />
    </manifest>
</target>
```

Numero di Versione

ant –f build-semplice.xml dist

- Attenzione
  - ⇨con questi task è possibile gestire automaticamente solo il numero di build (progressivo) e il timestamp
  - ⇨non è possibile aggiornare invece major.version e minor.version
  - ⇨è lo sviluppatore che decide quando far progredire il numero di versione a seconda dell'evoluzione del progetto e aggiorna manualmente il file build.properties

Riuso dei Buildfile

- Nei progetti di grandi dimensioni
  - ⇨emergono due necessità
- Problema n. 1
  - ⇨frammentare il progetto in più sottoprogetti, ciascuno con il proprio file di build
- Problema n. 2
  - ⇨utilizzare operazioni di build per quanto possibile standard nei vari sottoprogetti

Il Task ant

- Tradizionalmente
  - ⇨con Ant era necessario mettere in piedi "trucchi" per risolvere questi problemi
- A partire da Ant 1.6
  - ⇨numerose funzionalità aggiuntive specificamente pensate per questo scopo
  - ⇨consentono di organizzare molto meglio il processo di build

Il Task ant

- Funzionalità per il riuso dei buildfile
  - ⇨il task ant
  - ⇨il task import (a partire da Ant 1.6)
  - ⇨il task subant (a partire da Ant 1.6
  - ⇨il task macrodef (a partire da Ant 1.6)
- Il task ant
  - ⇨consente di richiedere l'esecuzione di un buildfile da un altro

Il Task ant

- Esempio: gli esempi iniziali di POO
  - ⇨calcolatrice e varianti
  - ⇨circonferenza e varianti
  - ⇨segmenti e varianti
  - ⇨si tratta di sottoprogetti indipendenti di un unico progetto (esempiIniziali)
  - ⇨sul sito viene distribuito il progetto completo
  - ⇨è utile potere ricostruire tutti assieme i sottoprogetti prima di costruire lo zip

Il Task ant

- Soluzione
  - ⇨creo un file di build per ciascun sottoprogetto
  - ⇨creo un file di build per il progetto (“master build file”) in cui uso il task ant

```
<target name="prepare" depends="clean">
    <ant antfile="./calcolatrice.xml" target="build" />
    <ant antfile="./calcolatricestatica.xml" target="build" />
    <ant antfile="./circonferenzea.xml" target="build" />
    <ant antfile="./circonferenzeb.xml" target="build" />
    <ant antfile="./segmenti.xml" target="build" />
 </target>
```

Il Task ant

>> esempiIniziali\build.xml
>> esempiIniziali\calcolatrice.xml

- Nota
  - ⇨nell'utilizzo del task ant è fondamentale la caratteristica di "immutabilità" delle proprietà
  - ⇨infatti il master build file attribuisce i valori alle proprietà (init) prima di eseguire i file di build dei sottoprogetti
  - ⇨i file di build dei sottoprogetti "ereditano" i valori delle proprietà e il loro target init non ha nessun effetto

Il Task ant

- Questo meccanismo è controllabile
  - ⇨attributo inheritall del task ant
  - ⇨inheritall=true (default) produce l'effetto di "passare" ai file di build dei sottoprogetti tutti i valori delle proprietà
  - ⇨inheritall=false non passa al file di build nessun valore delle proprietà
  - ⇨analogamente per inheritrefs (riferimenti ai percorsi)

Il Task ant

- Un task collegato: il task antcall
  - ⇨consente di chiamare un target all'interno dello stesso file di build
  - ⇨serve nel caso in cui sia necessario ripetere più volte l'esecuzione di un task
  - ⇨tendenzialmente da evitare

Il Task ant

- A partire da ant 1.6
  - ⇨il task macrodef, che consente di definire "target" che sono procedure rieseguibili
  - ⇨il task subant, che consente di ripetere le stesse operazioni su varie cartelle

Il Task import

- In progetti diversi
  - ⇨tipicamente il file di build ha sostanzialmente la stessa struttura
- Soluzione n. 1
  - ⇨replicare lo stesso file di build in tutti i sottoprogetti
- Svantaggio
  - ⇨introduce ridondanza e complica la manutenzione nel caso ci siano modifiche

Il Task import

- Soluzione n. 2
  - ⇨importare i target di un file di build “modello” nel file di build di ciascun progetto
- Tradizionalmente
  - ⇨veniva fatto con trucchi (entità di XML)
  - ⇨con molti limiti
- A partire da Ant 1.6
  - ⇨il task import

Il Task import

- Semantica del task import
  - ⇨consente di importare i target di un file di build in un altro
  - ⇨il file che importa “eredita” i target importati e può utilizzarli liberamente
  - ⇨in aggiunta, può “sovrascriverli” specificando una definizione alternativa per il target
  - ⇨la versione sovrascritta prevale su quella ereditata

Il Task import

- Un file di build modello
  - ⇨console-template-build.xml
  - ⇨sostanzialmente identico al file di build visto per il progetto appuntamenti
  - ⇨ma reso leggermente più generale
- In particolare
  - ⇨vengono verificate alcune condizioni per renderlo adattabili a contesti diversi

Il Task import

>> console-template-build.xml

- Il task condition
  - ⇨imposta il valore di una proprietà se una certa condizione è vera
- Esempio: esiste la cartella test.data.dir ?

```
<target name="-init-folders" depends="">
    <property name="src.dir" location="./src" />
    <property name="test.data.dir" location="${test.src.dir}/dati" />
    ...
    <condition property="test.data.dir.present">
       <available file="${test.data.dir}" type="dir" />
    </condition>
</target>
```


Il Task import

- In questo modo
  - ⇨è possibile adattare il file a diversi progetti
- I file di build dei progetti
  - ⇨nella maggioranza dei casi si limitano ad importare il modello

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="build" name="appuntamenti">

   <import file="../console-template-build.xml" />

</project>
```

Il Task import

- Una nota sulle condizioni
  - ⇨per esplicita ammissione degli ideatori Ant non è un linguaggio procedurale
  - ⇨ma un linguaggio “dichiarativo”
  - ⇨non è opportuno abusare eccessivamente di funzionalità procedurali come if/condition/antcall

Il Task import

>> morraCinese\build.xml

- In altri casi
  - ⇨ci sono caratteristiche peculiari del progetto che rendono indispensabile ridefinire alcuni target
- Esempio
  - ⇨morraCinese: è necessario ridefinire il target init per ridefinire il nome della classe che contiene il metodo main()
  - ⇨e i target che creano la distribuzione

Il Task import

- Il grande vantaggio di questo approccio
  - ⇨è molto facile cambiare le operazioni di build
  - ⇨introducendo nuovi target oppure cambiando quelli esistenti
  - ⇨basta agire sul modello
  - ⇨i cambiamenti si ripercuotono automaticamente su tutti i progetti i cui file di build sono basati sul modello

Il Task import

- Attenzione all'uso dei percorsi
  - ⇨bisogna tenere in considerazione che i percorsi relativi sono espansi sulla base di basedir (ovvero .)
  - ⇨la proprietà basedir viene impostata nel file di build che importa il modello
  - ⇨quindi i percorsi vanno scritti relativamente alla cartella del progetto, non del modello
  - ⇨es: dependent.properties

Il Task import

- Una convenzione di stile di Ant
  - ⇨i target richiamabili dalla linea di comando hanno tutti l'attributo description specificato
  - ⇨i target considerati "interni" (ovvero di servizio e che non devono essere richiamati dalla linea di comando), hanno un nome che comincia per -

Altre Operazioni

- Finora
  - ⇨una panoramica molto ampia su Ant
  - ⇨che dimostra come il processo di costruzione sia completamente automatizzabile
- Ma, in realtà...
  - ⇨Ant consente di fare molto di più di quello che abbiamo visto

Altre Operazioni

- Esempi
  - operazioni di ricerca e sostituzione nei file (es: per aggiungere il numero di versione ai file .java)
  - operazioni di convalida di documeti XML e trasformazione con XSL
  - deployment sulla rete, attraverso il task ftp che trasferisce un file su un server
  - per dettagli è possibile consultare il manuale

Altre Operazioni

- Inoltre
  - esistono numerosissimi task sviluppati da terze parti, per molte altre operazioni
  - infine, è possibile sviluppare i propri task (improbabile che sia necessario, visto il numero impressionante di task esistenti)
- Come introdurre un nuovo task
  - attraverso il task taskdef

Altre Operazioni

- Un task di ant
  - ⇨è una operazione su una classe Java
  - ⇨Ant, oltre al resto, è un framework che stabilisce come scrivere le classi che corrispondono a nuovi task
- Utilizzo di taskdef

```
<taskdef name="myjavadoc"
        classname="com.mydomain.JavadocTask“
        classpath=“mydomain.jar” />
```


Altre Operazioni

- La struttura completa di un buildfile
  - ⇨un progetto (project)
  - ⇨uno o più target che contengono uno o più task (proprietà, percorsi, taskdef ecc.)
- In effetti, a partire da Ant 1.6
  - ⇨il task property, path, taskdef possono comparire anche al di fuori dei target

Altre Operazioni

- Semantica di un task fuori da un target
  - ⇨viene eseguito prima di qualunque target
- Ma...
  - ⇨è sconsigliabile utilizzare task fuori dai target perchè si perde il controllo sulle dipendenze
  - ⇨è buona norma utilizzare la struttura progetto-target-task nei target vista finora

Riassumendo

- Funzionalità Avanzate
- Controllo del Processo di Build
  - ⇨Utilizzo di File di Proprietà
  - ⇨Build di Debug e di Release
  - ⇨Numero di Versione
- Riuso dei Buildfile
  - ⇨Il Task ant
  - ⇨Il Task import
- Altre Operazioni

Termini della Licenza