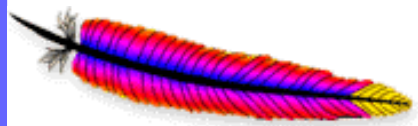


The Jakarta Project
http://jakarta.apache.org

Gerenciamento de projetos com o Apache Ant

Helder da Rocha (helder@acm.org)

Sobre este módulo

- Este módulo apresenta o Jakarta Ant - ferramenta importante para gerenciar projetos
  - Qualquer aplicação Java com mais que meia dúzia de classes ou organizada em pacotes deve ser organizada como um projeto
  - É uma boa prática manter scripts para automatizar procedimentos de desenvolvimento (compilar, testar, criar documentação, gerar JARs, etc.)
- O material disponível é muito extenso para tratamento detalhado neste curso
  - Abordagem será superficial, mas use-o como referência durante o curso

Ant: o que é?

- Ferramenta para construção de aplicações
  - Implementada em Java
  - Baseada em roteiros XML
  - Extensível (via scripts ou classes)
  - 'padrão' do mercado
  - Open Source (Grupo Apache, Projeto Jakarta)
- Semelhante a make, porém
  - Mais simples e estruturada (XML)
  - Mais adequada a tarefas comuns em projetos Java
  - Independente de plataforma

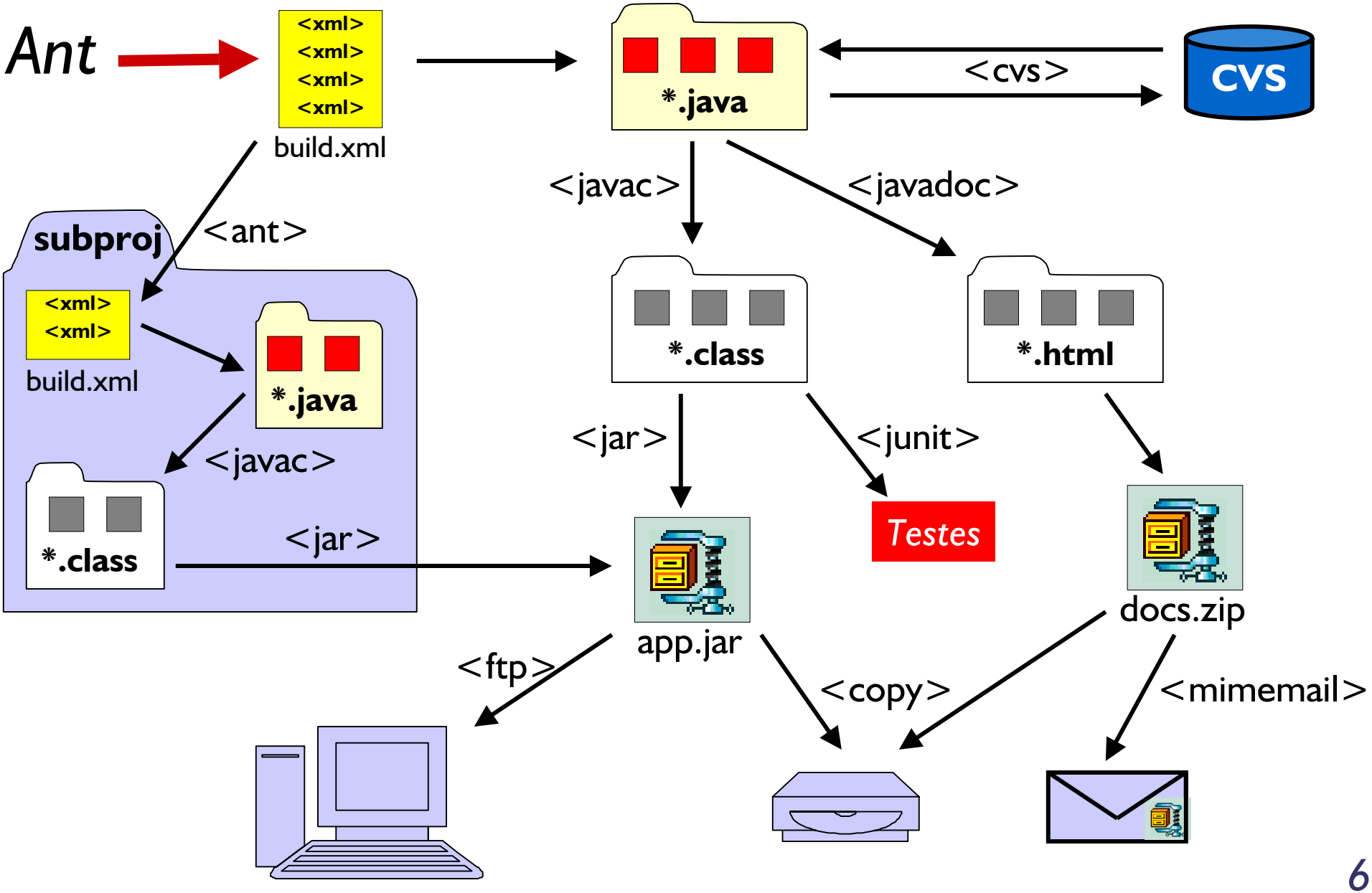
Para que serve?

- Para montar praticamente qualquer aplicação Java que consista de mais que meia dúzia de classes; Aplicações
  - distribuídas em pacotes
  - que requerem a definição de classpaths locais, e precisam vincular código a bibliotecas (JARs)
  - cuja criação/instalação depende de mais que uma simples chamada ao javac. Ex: RMI, CORBA, EJB, servlets, JSP,...
- Para automatizar processos frequentes
  - Javadoc, XSLT, implantação de serviços Web e J2EE (deployment), CVS, criação de JARs, testes, FTP, email

Como funciona?

- Ant executa roteiros escritos em XML: 'buildfiles'
- Cada projeto do Ant possui um buildfile
  - subprojetos podem ter, opcionalmente, buildfiles adicionais chamados durante a execução do primeiro
- Cada projeto possui uma coleção de alvos
- Cada alvo consiste de uma seqüência de tarefas
- Exemplos de execução
  - `ant`
    - procura build.xml no diretório atual e roda alvo default
  - `ant -buildfile outro.xml`
    - executa alvo default de arquivo outro.xml
  - `ant compilar`
    - roda alvo 'compilar' e possíveis dependências em build.xml

Como funciona (2)



Buildfile

- *O buildfile é um arquivo XML:* ***build.xml*** *(default)*
- *Principais elementos*
  - `<project default="alvo_default">`
    - *Elemento raiz (obrigatório): define o projeto.*
  - `<target name="nome_do_alvo">`
    - *Coleção de tarefas a serem executadas em seqüência*
    - *Deve haver pelo menos um <target>*
  - `<property name="nome" value="valor">`
    - *pares nome/valor usados em atributos dos elementos do* build.xml *da forma* `${nome}`
    - *propriedades também podem ser definidas em linha de comando (*`-Dnome=valor`*) ou lidas de arquivos externos (atributo* `file`*)*
  - *tarefas (mais de 130) - dentro dos alvos.*
    - `<javac>`, `<jar>`, `<java>`, `<copy>`, `<mkdir>`, ...

Buildfile (2)

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!-- Compila diversos arquivos .java -->
<project default="compile" basedir=".">
  <property name="src.dir" value="${basedir}/src" />
  <property name="build.dir" value="build" />
  <target name="init">
    <echo> Criando diretório </echo>
    <mkdir dir="${build.dir}" />
  </target>
  <target name="compile" depends="init"
          description="Compila os arquivos-fonte">
    <javac srcdir="${src.dir}" destdir="${build.dir}">
      <classpath>
        <pathelement location="${build.dir}" />
      </classpath>
    </javac>
  </target>
</project>
```

Propriedades

Alvos

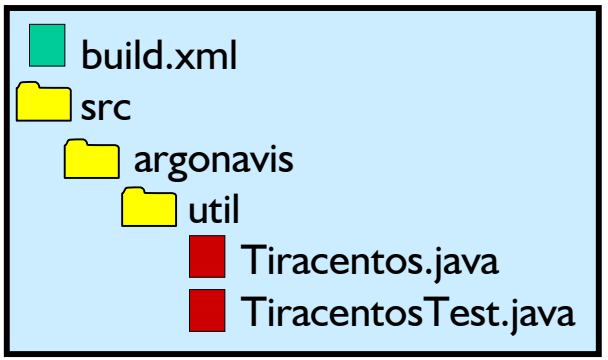
Tarefas

Exemplo

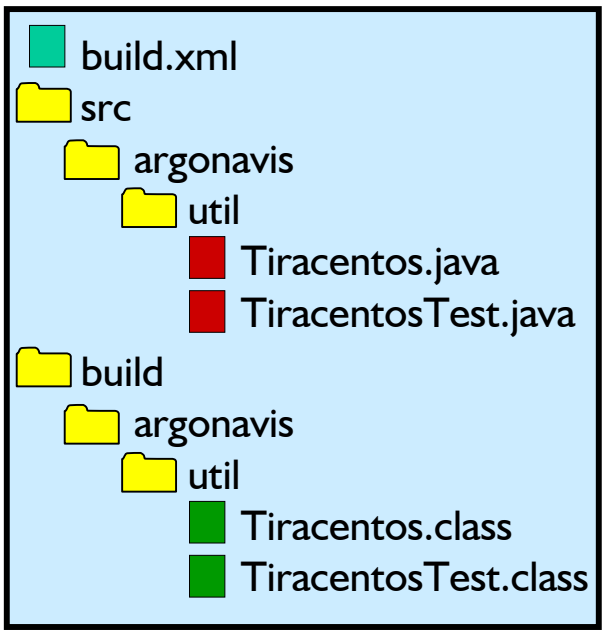
- *Executando buildfile da página anterior*

```
C:\usr\palestra\antdemo> ant
Buildfile: build.xml

init:
     [echo]  Criando diretório
     [mkdir] Created dir:
     C:\usr\palestra\antdemo\build

compile:
     [javac] Compiling 2 source files to
     C:\usr\palestra\antdemo\build

BUILD SUCCESSFUL
Total time: 4 seconds
C:\usr\palestra\antdemo>
```

ANTES



DEPOIS



Propriedades

- Podem ser definidas com **<property>**

```
<property name="app.nome" value="jmovie" />
```

- Podem ser carregadas de um arquivo

```
<property file="c:/conf/arquivo.conf" />
```

```
app.ver=1.0
docs.dir=c:\docs\
codigo=15323
```

arquivo.conf

- Podem ser passadas na linha de comando

```
c:\> ant -Dautor=Wilde
```

- Para recuperar o valor, usa-se ${nome}

```
<jar destfile="${app.nome}-${app.ver}.jar"/>
<echo message="O autor é ${autor}" />
<mkdir dir="build${codigo}" />
```

Propriedades especiais

- **<tstamp>**: Grava um instante
  - A hora e data podem ser recuperados como propriedades
    - `${TSTAMP}` *hhmm* *1345*
    - `${DSTAMP}` *aaaammdd* *20020525*
    - `${TODAY}` *dia mes ano* *25 May 2002*
  - Novas propriedades podem ser definidas, locale, etc.
  - Uso típico: `<tstamp/>`
- **<property** environment="env"**>**: Propriedade de onde se pode ler variáveis de ambiente do sistema
  - Dependende de plataforma

```
<target name="init">
  <property environment="env"/>
  <property name="j2ee.home"
            value="env.J2EE_HOME" />
</target>
```

O que se pode fazer com Ant?

- Compilar.
  <javac>, <csc>
- Gerar documentação
  <javadoc>, <junitreport>, <style>, <stylebook>
- Executar programas
  <java>, <apply>, <exec> <ant>, <sql>
- Testar unidades de código
  <junit>
- Empacotar e comprimir
  <jar>, <zip>, <tar>, <war>, <ear>, <cab>
- Expandir, copiar, instalar
  <copy>, <delete>, <mkdir>, <unjar>, <unwar>, <untar>, <unzip>
- Acesso remoto
  <ftp>, <telnet>, <cvs>, <mail>, <mimemail>
- Montar componentes
  <ejbc>, <ejb-jar>, <rmic>
- Criar novas tarefas
  <taskdef>
- Executar roteiros e sons
  <script>, <sound>

Compilação e JAR

- **<javac>**: Chama o compilador Java

```
<javac srcdir="dirfontes" destdir="dirbuild" >
    <classpath>
        <pathelement path="arquivo.jar" />
        <pathelement path="/arquivos" />
    </classpath>
    <classpath idref="extra" />
</javac>
```

- **<jar>**: Monta um JAR

```
<jar destfile="bin/programa.jar">
   <manifest>
      <attribute name="Main-class"
                 value="exemplo.main.Exec"/>
   </manifest>
   <fileset dir="${build.dir}"/>
</jar>
```

Tarefas do sistema de arquivos

- **<mkdir>**: *cria diretórios*

```
<mkdir dir="diretorio" />
```

- **<copy>**: *copia arquivos*

```
<copy todir="dir" file="arquivo" />
<copy todir="dir">
    <fileset dir="fonte"
             includes="*.txt" />
</copy>
```

- **<delete>**: *apaga arquivos*

```
<delete file="arquivo />
<delete dir="diretorio"/>
```

Geração de documentação

- **<javadoc>**: Gera documentação do código-fonte.
  - *Alvo abaixo gera documentação e exclui classes que contém 'Test.java'*

```
<target name="generate-docs">
  <mkdir dir="docs/api"/>
  <copy todir="tmp">
    <fileset dir="${src.dir}">
      <include name="**/*.java" />
      <exclude name="**/**Test.java" />
    </fileset>
  </copy>
  <javadoc destdir="docs/api"
           packagenames="argonavis.*"
           sourcepath="tmp" />
  <delete dir="tmp" />
</target>
```

Tipos de dados: arquivos e diretórios

- **<fileset>**: árvore de arquivos e diretórios
  - Conteúdo do conjunto pode ser reduzido utilizando elementos <include> e <exclude>
  - Usando dentro de tarefas que manipulam com arquivos e diretórios como <copy>, <zip>, etc.

```
<copy todir="${build.dir}/META-INF">
   <fileset dir="${xml.dir}" includes="ejb-jar.xml"/>
   <fileset dir="${xml.dir}/jboss">
      <include name="*.xml" />
      <exclude name="*-orig.xml" />
   </fileset>
</copy>
```

- **<dirset>**: árvore de diretórios
  - Não inclui arquivos individuais

Tipos de dados: coleções

- **<patternset>**: representa coleção de padrões

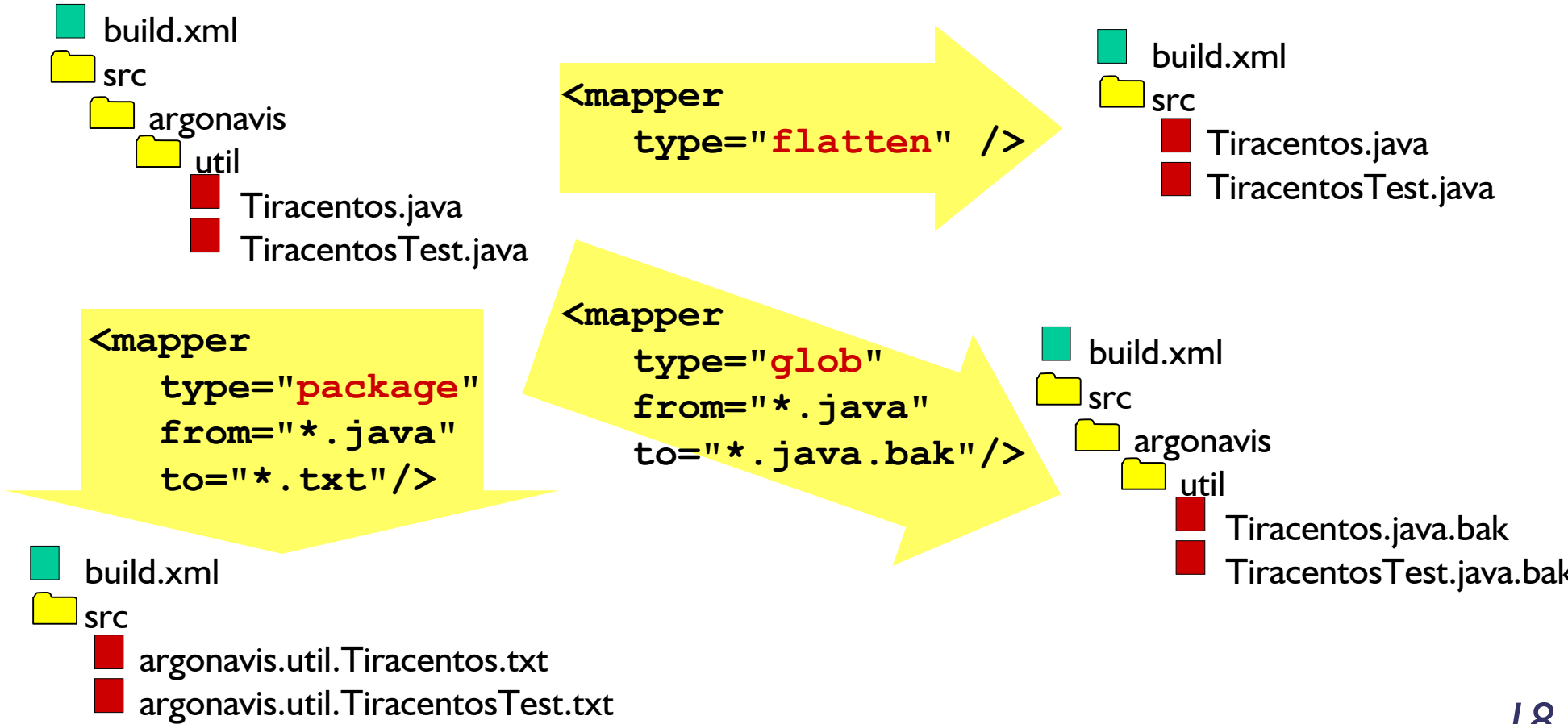
```
<patternset id="project.jars" >
  <include name="**/*.jar"/>
  <exclude name="**/*-test.jar"/>
</patternset>
```

- **<path>**: representa uma coleção de caminhos
  - Associa um ID a grupo de arquivos ou caminhos

```
<path id="server.path">
  <pathelement path="${j2ee.home}/lib/locale" />
  <fileset dir="${j2ee.home}/lib">
    <patternset refid="project.jars" />
  </fileset>
</path>

<target name="compile" depends="init">
   <javac destdir="${build.dir}" srcdir="${src.dir}">
      <classpath refid="server.path" />
   </javac>
</target>
```

Tipos de dados: File Mapper

- **<mapper>**: *altera nomes de arquivos durante cópias ou transformações*
  - *Seis tipos: identity, flatten, merge, regexp, glob, package*



Tipos de dados: seletores

- Permitem a seleção dos elementos de um fileset usando critérios além dos definidos por <include> e <exclude>
- Sete seletores básicos (pode-se criar novos)
  - **<contains>** - Seleciona arquivos que contém determinado texto
  - **<date>** - Arquivos modificados antes ou depois de certa data
  - **<depend>** - Seleciona arquivos cuja data de modificação seja posterior a arquivos localizados em outro lugar
  - **<depth>** - Seleciona arquivos encontrados até certa profundidade de uma árvore de diretórios
  - **<filename>** - Equivalente ao include e exclude
  - **<present>** - Seleciona arquivo com base na sua (in)existência
  - **<size>** - Seleciona com base no tamanho em bytes

***Exemplo**: Seleciona arquivos do diretório "fonte" que também estão presentes em "destino"*

```
<fileset dir="fonte">
  <present targetdir="destino"/>
</fileset>
```

Tipos de dados: filtros

- **<filter>** e **<filterset>**: Permite a substituição de padrões em arquivos durante a execução de uma tarefa
  - Caractere default: @
  - Exemplo: a tarefa abaixo irá substituir todas as ocorrências de `@javahome@` por `c:\j2sdk1.4.0` nos arquivos copiados

```
<copy todir="${dest.dir}">
   <fileset dir="${src.dir}"/>
   <filterset>
      <filter token="javahome" value="c:\j2sdk1.4.0"/>
   </filterset>
</copy>
```

- Pares *token=valor* podem ser carregados de arquivo:

```
<filterset>
   <filtersfile file="build.properties" />
</filterset>
```

Substituição pode ser feita também sem tokens com <replace>

Execução de aplicações

- **<java>**: *roda o interpretador Java*

```
<target name="runrmiclient">
  <java classname="hello.rmi.HelloClient" fork="true">
    <jvmarg value="-Djava.security.policy=rmi.policy"/>
    <arg name="host" value="${remote.host}" />
    <classpath refid="app.path" />
  </java>
</target>
```

- **<exec>**: *executa um comando do sistema*

```
<target name="orbd">
  <exec executable="${java.home}\bin\orbd">
    <arg line="-ORBInitialHost ${nameserver.host}"/>
  </exec>
</target>
```

- **<apply>**: *semelhante a <exec> mas usado em executáveis que operam sobre outros arquivos*

Tarefas de rede

- ***<ftp>***: *Realiza a comunicação com um servidor FTP remoto para upload ou download de arquivos*
  - *Tarefa opcional que requer NetComponents.jar (http://www.savarese.org)*

```
<target name="remote.jboss.deploy" depends="dist">
    <ftp server="${ftp.host}" port="${ftp.port}"
         remotedir="/jboss/server/default/deploy"
         userid="admin" password="jboss"
         depends="yes" binary="yes">
        <fileset dir="${basedir}">
            <include name="*.war"/>
            <include name="*.ear"/>
            <include name="*.jar"/>
        </fileset>
    </ftp>
</target>
```

Efeitos sonoros

- **<sound>**: define um par de arquivos de som para soar no sucesso ou falha de um projeto
  - Tarefa opcional que requer Java Media Framework
- Exemplo:
  - No exemplo abaixo, o som frog.wav será tocado quando o build terminar sem erros fatais. Bark.wav tocará se houver algum erro que interrompa o processo:

```
<target name="init">
  <sound>
    <success source="C:/Media/frog.wav"/>
    <fail source="C:/Media/Bark.wav"/>
  </sound>
</target>
```

Ant programável

- Há duas formas de estender o Ant com novas funções
  - Implementar roteiros usando JavaScript
  - Criar novas tarefas reutilizáveis
- A tarefa <script> permite embutir JavaScript em um buildfile. Pode-se
  - realizar operações aritméticas e booleanas
  - utilizar estruturas como if/else, for, foreach e while
  - manipular com os elementos do buildfile usando DOM
- A tarefa <taskdef> permite definir novas tarefas
  - tarefa deve ser implementada em Java e estender Task
  - método execute() contém código de ação da tarefa
  - cada atributo corresponde a um método setXXX()

Integração com outras aplicações

- Ant provoca vários eventos que podem ser capturados por outras aplicações
  - Útil para implementar integração, enviar notificações por email, gravar logs, etc.
- Eventos
  - Build iniciou/terminou
  - Alvo iniciou/terminou
  - Tarefa iniciou/terminou
  - Mensagens logadas
- Vários listeners e loggers pré-definidos
  - Pode-se usar ou estender classe existente.
  - Para gravar processo (build) em XML:

```
ant -listener org.apache.tools.ant.XmlLogger
```

Integração com editores e IDEs

- Produtos que integram com Ant e oferecem interface gráfica e eventos para buildfiles:
  - Antidote: GUI para Ant (do projeto Jakarta)
    - http://cvs.apache.org/viewcvs/jakarta-ant-antidote/
  - JBuilder (AntRunner plug-in)
    - http://www.dieter-bogdoll.de/java/AntRunner/
  - NetBeans e Forté for Java
    - http://ant.netbeans.org/
  - Visual Age for Java (integração direta)
  - JEdit (AntFarm plug-in)
    - http://www.jedit.org
  - Jext (AntWork plug-in)
    - ftp://jext.sourceforge.net/pub/jext/plugins/AntWork.zip

Como gerenciar projetos com o Ant

- Crie um diretório para armazenar seu projeto. Nele guarde o seu build.xml
  - Use um arquivo build.properties para definir propriedades exclusivas do seu projeto (assim você consegue reutilizar o mesmo build.xml em outros projetos)
- Dentro desse diretório, crie alguns subdiretórios
  - src/ Para armazenar o código-fonte
  - lib/ Opcional. Para guardar os JARs de APIs usadas
  - doc/ Opcional. Para guardar a documentação gerada
- O seu Ant script deve ainda criar
  - build/ Ou classes/. Onde estará o código compilado
  - dist/ Ou jars/ ou release/. Onde estarão os JARs criados

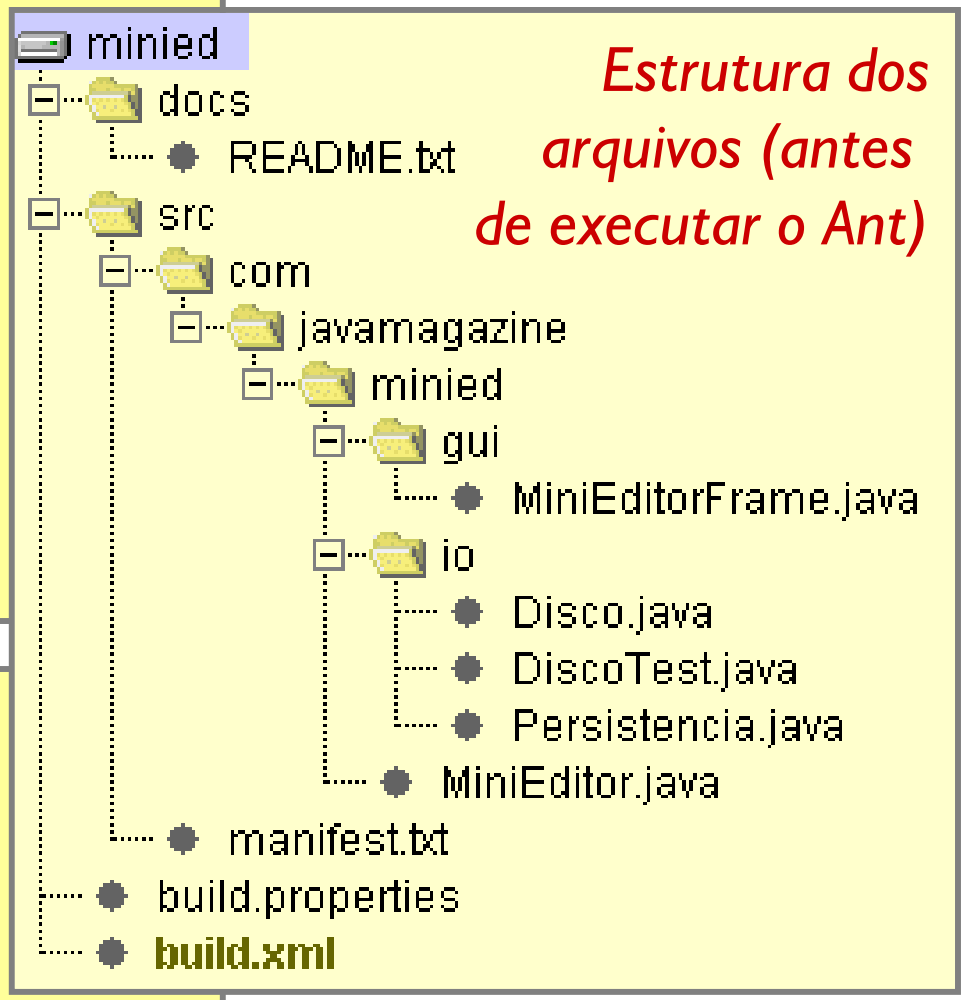
Alvos básicos do build.xml

- Você também deve padronizar os nomes dos alvos dos seus build.xml. Alguns alvos típicos são
  - *init* Para criar dietórios, inicializar o ambiente, etc.
  - *clean* Para fazer a faxina, remover diretórios gerados, etc.
  - *compile* Para compilar
  - *build* Para construir a aplicação, integrar, criar JARs
  - *run* Para executar um cliente da aplicação
  - *test* Para executar os testes da aplicação
- Você pode usar outros nomes, mas mantenha um padrão
- Também pode criar uma nomenclatura que destaque alvos principais, usando maiúsculas. Ex:
  - *CLEAN, que chama clean-this, clean-that, undeploy, etc.*
  - *BUILD, que chama build-depend, build-client, build-server*

Exemplo de projeto

build.xml

```
<project default="compile" name="MiniEd">
  <property file="build.properties"/>
  <target name="init">
    <mkdir dir="${build.dir}"/>
    <mkdir dir="${dist.dir}"/>
  </target>
  <target name="clean">    ...   </target>
  <target name="compile"
          depends="init"> ...   </target>
  <target name="build"
          depends="compile">...</target>
  <target name="javadoc"
          depends="build"> ... </target>
  <target name="run"
          depends="build"> ... </target>
</project>
```



Estrutura dos arquivos (antes de executar o Ant)

```
# Nome da aplicação
app.name=minied
# Nomes dos diretórios
src.dir=src
docs.dir=docs
build.dir=classes
dist.dir=jars
# Nome da classe executável
app.main.class=com.javamagazine.minied.MiniEditor
root.package=com
```

build.properties

Conclusões

- O Ant é uma ferramenta indispensável em qualquer projeto de desenvolvimento Java
  - Permite automatizar todo o processo de desenvolvimento
  - Facilita a montagem da aplicação por outras pessoas
  - Ajuda em diversas tarefas essenciais do desenvolvimento como compilar, rodar, testar, gerar JavaDocs, etc.
  - Independe de um IDE comercial (mas pode ser facilmente integrado a um)
- Use o Ant em todos os seus projetos
  - Crie sempre um projeto e um buildfile, por mais simples que seja a sua aplicação
  - Escreva buildfiles que possam ser reutilizados

Exercícios

- 1. Crie um buildfile para cada projeto que você montar nos próximos módulos
  - Use o template básico de build.xml mostrado neste capítulo
  - Configure-o e personalize-o para incluir os alvos e tarefas específicas do seu projeto
  - Inclua alvos com tarefas de javadoc, jar e execução
- 2.Pratique com os exemplos apresentados
  - Execute os buildfiles e use o código como exemplo

Dica: consulte a documentação do Ant

  - Muito bem estruturada
  - Contém exemplos de todos os tags

Curso J100: Java 2 Standard Edition

Revisão 17.0

© 1996-2003, Helder da Rocha
(helder@acm.org)

argonavis.com.br