

2

J820

Introdução ao Apache Ant

O que é Ant?

- Uma 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
- Onde encontrar: <http://ant.apache.org>

Para que serve?

- Para montar praticamente **qualquer** aplicação Java que consista de mais que meia dúzia de classes;

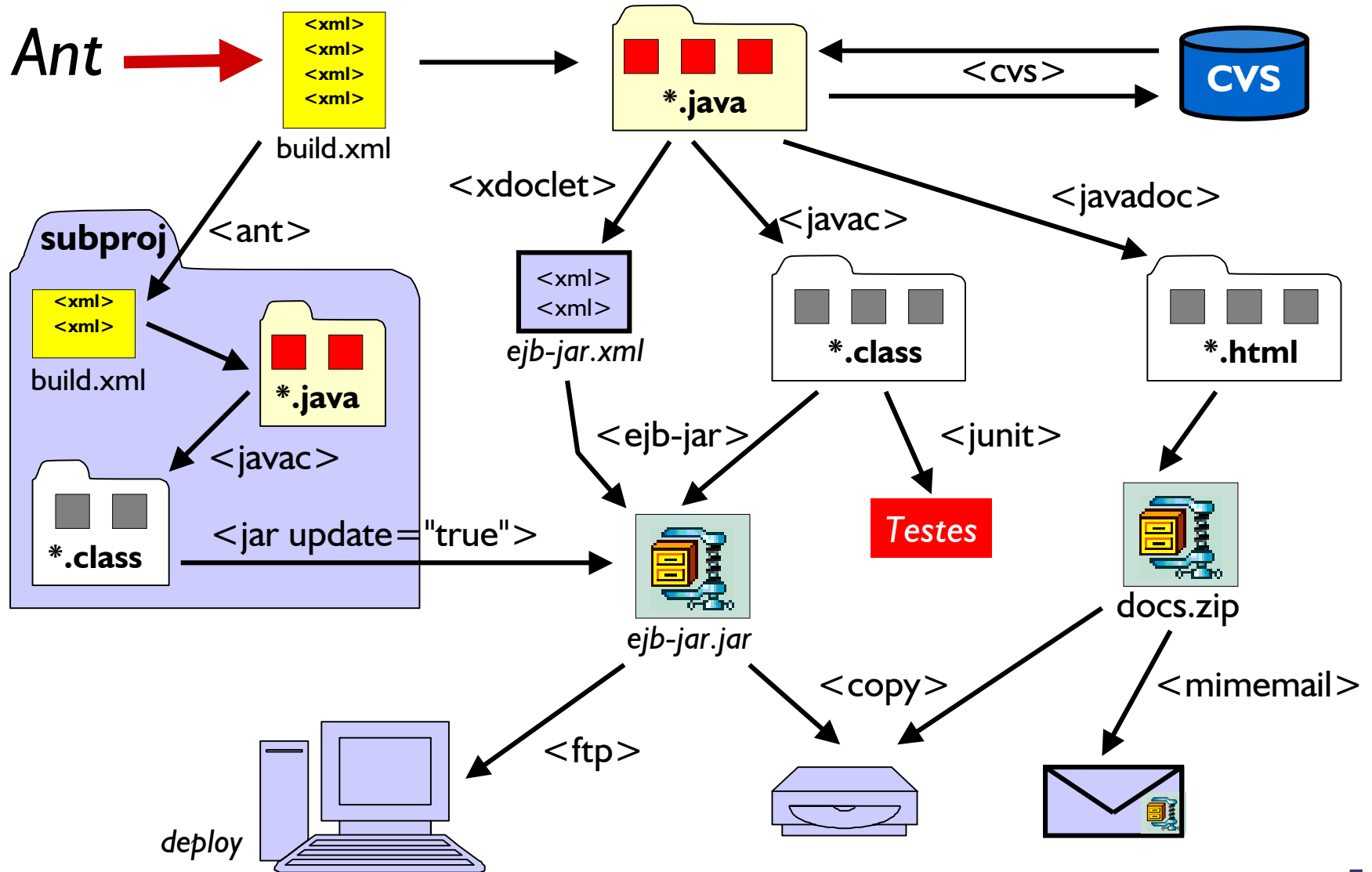
Aplicações

- Distribuídas em **pacotes**
- Que requerem a definição de **classpath** locais, e precisam vincular código a bibliotecas (JARs)
- Cujas 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)



- 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
 - Pode-se estabelecer dependências entre alvos
 - Deve haver **pelos 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) - usadas 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="src" />
  <property name="build.dir" value="classes" />
  <target name="init">
    <mkdir dir="${build.dir}" />
  </target>
  <target name="clean">
    <delete dir="${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

Elementos embutidos nas tarefas

Exemplo

- Executando buildfile da página anterior

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

init:
  [mkdir] Created dir:
  C:\usr\palestra\antdemo\classes

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

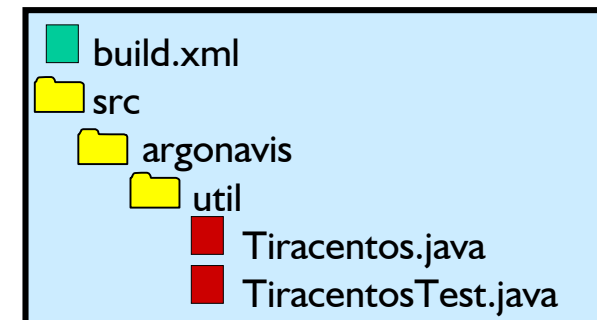
BUILD SUCCESSFUL
Total time: 4 seconds
C:\usr\palestra\antdemo> ant clean
Buildfile: build.xml

clean:
  [delete] Deleting dir:
  C:\usr\palestra\antdemo\classes

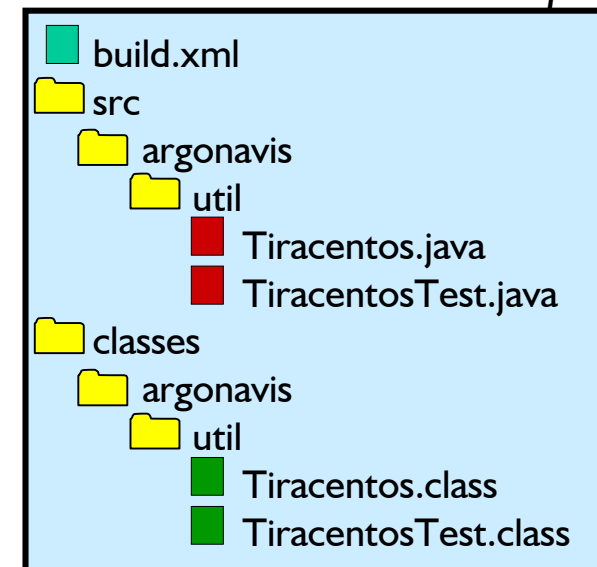
BUILD SUCCESSFUL
Total time: 2 seconds
C:\usr\palestra\antdemo>
```

ANTES de 'ant'

DEPOIS de 'ant clean'



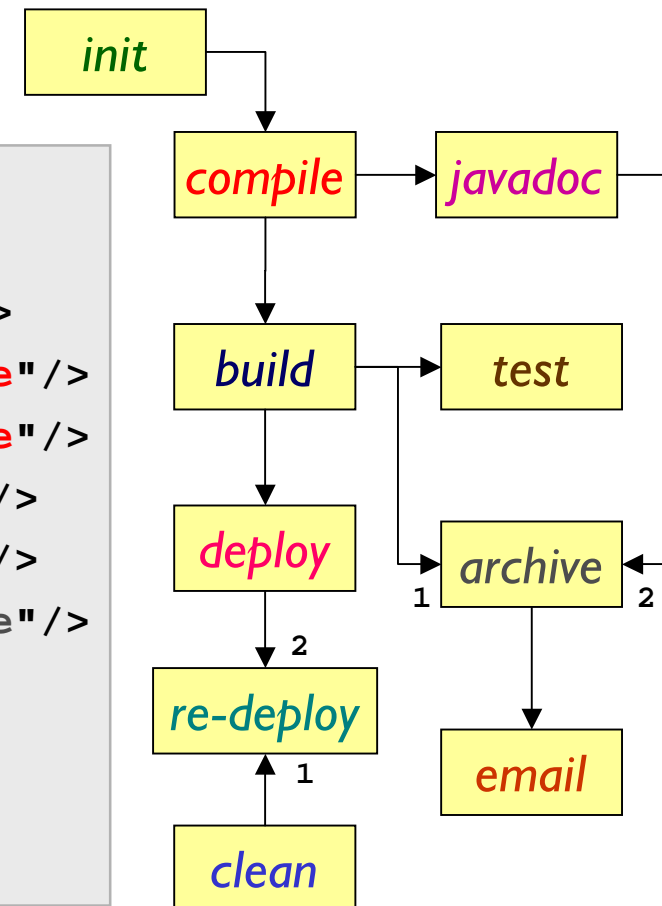
DEPOIS de 'ant' ou 'ant compile'



Dependências

- Fazem com que a chamada de um alvo cause a chamada de outros alvos, em determinada ordem
 - Promovem reuso de código

```
<target name="init" />
<target name="clean" />
<target name="compile" depends="init"/>
<target name="javadoc" depends="compile"/>
<target name="build" depends="compile"/>
<target name="test" depends="build"/>
<target name="deploy" depends="build"/>
<target name="email" depends="archive"/>
<target name="archive"
  depends="build, javadoc"/>
<target name="re-deploy"
  depends="clean, deploy"/>
```



Tarefas condicionadas

- Algumas tarefas só são executadas dentro de determinadas condições
 - `<mkdir>` só cria o diretório se este não existir
 - `<delete>` só apaga o que existe (não faz nada se arquivo ou diretório não existir)
 - `<javac>` compila apenas os arquivos `*.java` que foram modificados desde a última compilação
- Comportamento condicional do `<javac>` depende da estrutura de pacotes
 - É preciso que a estrutura de diretórios dos fontes (diretório `src/`) reflita a estrutura de pacotes
 - Ex: se `Conta.java` declara pertencer a pacote `banco`, deve estar em diretório `banco` dentro de `src/`

O que se pode fazer com Ant?

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

Tarefas úteis (I)

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

A tarefa <javac>

- É a tarefa mais importante de um processo típico de montagem
 - Oferece uma fachada sobre compiladores Java
 - Suporta outros compiladores (por exemplo, o IBM Jikes)
- Opções de linha de comando do compilador tornam-se atributos ou subelementos de <javac>

```
javac -classpath abc.jar;c:\exemplos -d build *.java
```

é o mesmo que

```
<javac srcdir="." destdir="build">  
  <classpath path="abc.jar;c:\exemplos" />  
</javac>
```

<javac> define FileSets

- O FileSet é um tipo de dados especial que representa uma árvore de arquivos e diretórios
 - A tarefa <javac> é um tipo de FileSet
- Outra forma de escrever <javac> (usando path-like structures) para indicar o caminho de fontes (src) e classes (classpath)

```
<javac destdir="${classes.dir}">
  <src path="${gen.src.dir}">
    <patternset refid="default.excludes" />
  </src>
  <src path="${src.dir}" />
  <classpath refid="${app.path}" />
</javac>
```

Tarefas úteis (2)

- **<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" />
```

Tipos de dados (I)

- **<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>
```

Árvore a ser copiada para `${build.dir}/META-INF` consiste de

- O arquivo `ejb-jar.xml` localizado em `${xml.dir}`
- Todos os arquivos `.xml` de `${xml.dir}/jboss` com exceção dos arquivos terminados em `-orig.xml`

FileSets filtrados

- *Pode-se filtrar os elementos de um FileSet usando subelementos <include> e <exclude>*
 - *Pode-se também usar os atributos includes e excludes*
- *O bloco abaixo inclui apenas os arquivos *.txt diretamente abaixo de pasta*

```
<fileset dir="pasta">  
  <include name="*.txt" />  
</fileset>
```

- *Este outro inclui todos os *.txt em qualquer lugar da árvore exceto os *-old.txt logo abaixo de pasta*

```
<fileset dir="pasta">  
  <include name="**/*.txt" />  
  <exclude name="*-old.txt" />  
</fileset>
```

- *Tipo de FileSet*
- *Permite incluir prefixos em arquivos armazenados em ZIPs (ou JARs)*
- **Prefixos** *são transformados em diretórios na descompressão*
- *Exemplo de uso*

```
<jar destfile="arquivo.jar">  
  <zipfileset dir="dados"  
    prefix="extra/dados" />  
</jar>
```

Tipos de dados (2)

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

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

Padrões podem ser reusados e são identificados pelo ID

- **<path>**: coleção de caminhos (path-like structures)
 - 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>
```

Path-like structures

- São listas ordenadas de elementos de um caminho do sistema de arquivos
- Exemplo

```
<classpath>
  <pathelement location="lib/arquivo.jar" />
  <pathelement location="c:\classes" />
  <pathelement path="/usr/bin:/usr:/lib" />
</classpath>
```
- O atributo **location** permite especificar um único arquivo ou diretório. O atributo **path** permite especificar um caminho inteiro
 - As barras e separadores serão convertidos ao sistema operacional nativo

Paths e FileSets

- *Paths podem ser compostos de conjuntos de arquivos usando `<fileset>`*

```
<path id="app.path">
  <fileset dir="app.dir" />
  <fileset dir="mais.dir">
    <include name="**/*.jar" />
    <include name="**/*.zip" />
  </fileset>
  <pathelement location="c:\classes" />
  <pathelement path="/usr/bin:/usr:/lib" />
</path>
```

- São coleções de padrões `<include>` e `<exclude>`
- **Sintaxe**
 - *** combina com um ou mais caracteres
 - *?* combina com um caractere
 - ****, usado como nome de diretório, combina com todos os diretórios daquele ponto em diante na árvore
 - */* ou ** no final de um diretório, é equivalente a */***
- **Pattern sets podem ser definidos em**
 - Atributos `includes` e `excludes` de um fileset
 - Arquivos com um pattern por linha carregados através de atributos `includesfile` e `excludesfile`
 - Elementos `<patternset>` identificados por um id contendo listas de `<include>` e `<exclude>`

Exemplos de Pattern Sets

```
<patternset id="docs">  
  <include name="**/*.htm*" />  
  <exclude name="~*.htm*" />  
</patternset>
```

```
<patternset id="webdocs">  
  <include name="**/*.gif, **/*.jpg" />  
  <patternset refid="docs" />  
</patternset>
```

Tipos de dados (4): 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>
```

Exemplos com seletores

```
<copy todir="novos">
  <fileset dir="docs">
    <not>
      <present targetdir="public_html" />
    </not>
  </fileset>
</copy>
```

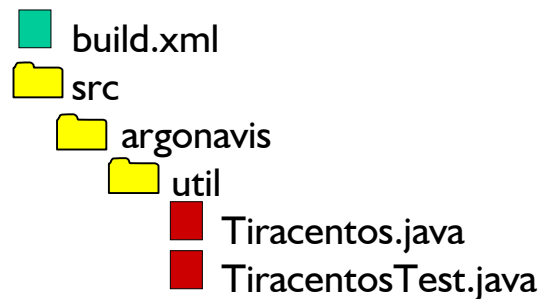
Seletores podem ser combinados dentro de **containers de seletores** para realizar lógica de seleção

<and>, *<or>*, *<none>*, *<majority>*

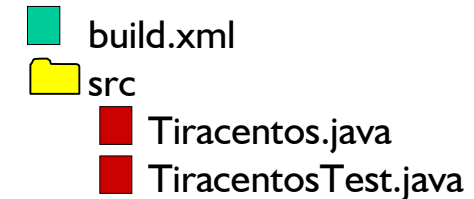
```
<copy todir="novos">
  <fileset dir="docs">
    <and>
      <contains text="Revision" />
      <not><present targetdir="public_html"/></not>
    </and>
  </fileset>
</copy>
```

Mapper

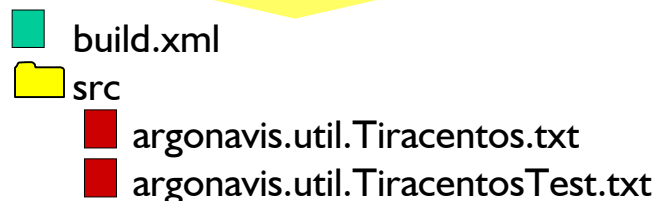
- **<mapper>**: altera nomes de arquivos durante cópias ou transformações (use dentro de <copy>, por exemplo)
 - Seis tipos: *identity*, *flatten*, *merge*, *regex*, *glob*, *package*



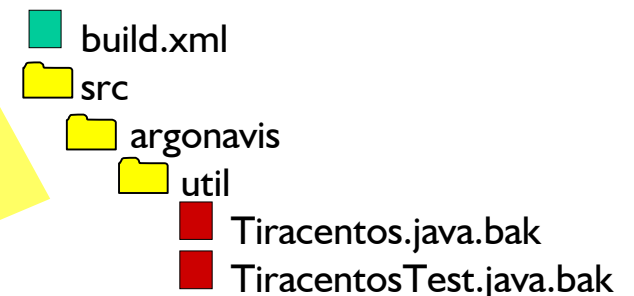
`<mapper type="flatten" />`



`<mapper type="package" from="*.java" to="*.txt" />`



`<mapper type="glob" from="*.java" to="*.java.bak" />`



Tipos de dados (5): filtros

- **<filter>** e **<filterset>**: Permite a substituição de padrões em arquivos durante a execução de uma tarefa
 - Caractere default: @
 - **Exemplo**: a cópia abaixo irá substituir todas as ocorrências de **@javahome@** por **c:\j2sdk1.4** nos arquivos copiados

```
<copy todir="{dest.dir}">
  <fileset dir="{src.dir}" />
  <filterset>
    <filter token="javahome" value="c:\j2sdk1.4" />
  </filterset>
</copy>
```

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

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

➔ Substituição sem tokens pode ser feita com **<replacetoken>**

Exemplo com Filter Sets

- Exemplo com inserção de datas: teste.txt

- Data de hoje: @data@ / @hora@

- Bulildfile

```
<tstamp />
```

```
<copy todir="gerados">
```

```
  <fileset dir="testes" includes="teste.txt" />
```

```
  <filterset>
```

```
    <filter token="data" value="\${DSTAMP}" />
```

```
    <filter token="hora" value="\${TSTAMP}" />
```

```
  </filterset>
```

```
</copy>
```

- Resultado

- Data de hoje: 20030528 / 1458

FilterChains e FilterReaders

- *FilterReader* é um filtro pré-configurado
- *FilterChains* são grupos ordenados de *FilterReaders* executados em conjunto
- Usados dentro de `<copy>`, `<move>`, `<loadfile>` e `<loadproperties>`
- Alguns *FilterReaders*
 - `<classconstants>`, `<expandproperties>`,
 - `<linecontains>`, `<prefixlines>`, `<replacetokens>`,
 - `<stripjavacomments>`, `<striplinebreaks>`,
 - `<tabstospaces>`

Exemplos de FilterChain

```
<copy todir="novos">
  <filterchain>
    <stripcomments>
      <comment value="#" />
    </stripcomments>
  </filterchain>
</copy>
```

```
<loadproperties srcfile="Constantes.class">
  <filterchain>
    <classconstants />
    <prefixlines prefix="Constantes." />
  </filterchain>
</loadproperties >
```

Tarefas úteis (3)

- **<javadoc>**: Gera documentação do código-fonte.
 - Exemplo: alvo **generate-docs** abaixo gera documentação excluindo classes que terminam em '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>
```

Copiar de `${src.dir}`

Procurar em todos os subdiretórios

Onde achar as fontes

- *Todas podem ser lidas com `${nome}`*
- *Nativas do Ant*
 - ***ant.file** - caminho completo do build.xml*
 - ***ant.home** - diretório de instalação do ant*
 - ***ant.java.version** - versão do Java usado*
 - ***ant.project.name** - nome do projeto*
 - ***ant.version** - versão do Ant*
 - ***basedir** - diretório onde está o build.xml*
- *Nativas do Java ou do JVM*
 - *Qualquer propriedade nativa do Java pode ser lida da mesma forma*

Como criar propriedades

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

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

- Podem ser carregadas de um arquivo

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

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

arquivo.properties

- 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 são imutáveis

- Uma vez definida, uma propriedade não muda de valor
- O primeiro valor atribuído é sempre usado
 - Tentativas posteriores de alterar o valor da propriedade não funcionam
- Propriedades passadas em linha de comando são definidas **antes** das propriedades dentro do `build.xml` ou `build.properties`

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
 - Dependente de plataforma

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

Propriedades que apontam para caminhos

- Se uma propriedade representa um **caminho** no sistema de arquivos, ela pode ser definida com *name/value*, mas o valor será sempre um caminho relativo
- Usando *name/location* pode-se definir valores absolutos

```
<property name="build.dir"  
          location="classes" />
```

guardará o caminho completo ao diretório local build

Propriedades condicionais

- **<available>** irá definir uma propriedade (com valor *true*) se um determinado recurso existir

```
<available property="arquivo.existe"  
file="importante.txt"  
type="file" />
```

- A existência ou não de uma propriedade pode ser usada em blocos condicionais

```
<target name="nome" if="propriedade">  
...  
</target>
```

Propriedades condicionais

- **<uptodate>** seta uma propriedade se os arquivos de uma fonte estiverem em dia com os arquivos de um destino, identificado por um **<mapper>**
 - A propriedade pode ser usada para realizar controle condicional

- Exemplo de uso

```
<uptodate property="tests.unnecessary">  
  <srcfiles dir="src"  
    includes="**/*.java" />  
  <mapper type="glob"  
    from="*.java" to="*.class" />  
</uptodate>
```

Propriedades condicionais

- **<condition>** define propriedades usando diversas condições e operadores <and>, <or> e <not>
- Sub-elementos de <condition>
 - <available>, <uptodate>, <os>, <>equals>, <isset>, <checksum>, <http>, <socket>, <filesmatch>, <contains>, <istrue>, <isfalse>

Exemplos de <condition>

```
<condition property="tests.unnecessary">
  <and>
    <uptodate>
      <srcfiles dir="src" includes="**/*.java"/>
      <mapper type="glob" from="*.java"
        to="${build.dir}/classes/*.class" />
    </uptodate>

    <uptodate>
      <srcfiles dir="test" includes="**/*.java"/>
      <mapper type="glob" from="*.java"
        to="${test.dir}/*.class" />
    </uptodate>

    <uptodate>
      <srcfiles dir="test" excludes="**/*.java"/>
      <mapper type="glob" from="*" to="${test.dir}/*" />
    </uptodate>
  </and>
</condition>
```

Targets condicionais

- Às vezes o que mais interessa em uma propriedade é se ela está definida ou não
- Seu estado pode ser usado para executar alvos condicionalmente

```
<target name="copysource" if="copy.source">  
  <copy todir="build/classes">  
    <fileset dir="src"/>  
  </copy>  
</target>
```

```
<target name="jar" depends="compile,copysource">  
  <jar basedir="build/classes" jarfile="dist/our.jar">  
</target>
```

- O alvo acima executará se **qualquer** valor for atribuído
 - *ant -Dcopy.source=false jar*

Patternsets condicionais

- *Patternsets que incluem ou excluem padrões de localização de arquivos podem também ser condicionais*
- *No bloco abaixo alguns arquivos são excluídos se uma determinada propriedade não estiver definida*

```
<javac srcdir="src" destdir="${build.dir}/classes">  
  <exclude name="antbook/xdoclet/*.java"  
          unless="xdoclet.present" />  
</javac>
```

Falha condicional

- A tarefa `<fail>` pode ser usada para causar a falha de um build
 - Pode ser controlada com atributos `if` e `unless` para testar propriedades e falhar dependendo de condições

- *Todos os elementos do Ant podem ser identificados com um atributo id*
- *O elemento pode ser chamado por referência em diversas situações, referindo-se ao seu id através do atributo refid*

```
<path id="compile.classpath">  
  <pathelement location="{lucene.jar}"/>  
  <pathelement location="{tidy.jar}"/>  
</path>
```

```
<path id="test.classpath">  
  <path refid="compile.classpath"/>  
  <pathelement location="{junit.jar}"/>  
  <pathelement location="{build.dir}/classes"/>  
  <pathelement location="{build.dir}/test"/>  
</path>
```

Path como string

- Um *path* pode ser transformado em *String*, se for necessário, simplesmente passando sua referência na definição de uma propriedade

- **Exemplo**

```
<path id="the.path">  
  <pathelement path="some.jar;another.jar"/>  
</path>
```

```
<property name="path.string" refid="the.path"/>  
<echo message="path = ${path.string}"/>
```

- **Resultado**

```
[echo] path = /home/ant/some.jar: /home/ant/another.jar
```

Tarefas úteis (4): J2EE

```
<ear destfile="app.ear" appxml="application.xml">
  <fileset dir="${build}" includes="*.jar,*.war"/>
</ear>
```

```
<ejbjar srcdir="${build}" descriptor="${xml.dir}" ... >
  <jboss destdir="${deployjars.dir}" />
</ejbjar>
```

Há suporte aos principais servidores de aplicação

```
<war destfile="bookstore.war" webxml="meta/metainf.xml">
  <fileset dir="${build}/${bookstore2}" >
    <include name="*.jsp" />
    <exclude name="*.txt" />
  </fileset>
  <classes dir="${build}" >
    <include name="database/*.class" />
  </classes>
  <lib dir="bibliotecas" />
  <webinf dir="etc" />
</war>
```

WEB-INF/web.xml

Fileset para raiz do WAR

Fileset para
WEB-INF/classes

Fileset para
WEB-INF/lib

Fileset para WEB-INF/

Tarefas úteis (5): extensão



<ejbdoclet> e **<webdoclet>***: Geram código

- Requer JAR de `xdoclet.sourceforge.net`
- Ideal para **geração automática de arquivos de configuração** (`web.xml`, `ejb-jar.xml`, `application.xml`, `taglibs`, `struts-config`, etc.) e **código-fonte** (`beans`, `value-objects`)

```
<ejbdoclet sourcepath="src" destdir="${build.dir}"
           classpathref="xdoclet.path" ejbspec="2.0">
  <fileset dir="src">
    <include name="**/*Bean.java" />
  </fileset>
  <remoteinterface/>
  <homeinterface/>
  <utilobject/>
  <entitypk/>
  <entitycmp/>
  <deploymentdescriptor destdir="${dd.dir}"/>
  <jboss datasource="java:/OracleDS" />
</ejbdoclet>
```

Detalhes da configuração do componente estão nos comentários de JavaDocs do código-fonte dos arquivos envolvidos e arquivos de template

* Nomes convencionais criados a partir de tarefa externa

Tarefas úteis (6): execução

- **<java>**: executa 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 úteis (7): 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>
```

Tarefas úteis (8): XSLT

- **<style>**: Transforma documentos XML em outros formatos usando folha de estilos XSLT (nativa)
 - Usa TrAX (default), Xalan ou outro transformador XSL

```
<style basedir="xmldocs"
       destdir="htmldocs"
       style="xmltohtml.xsl" />
```

- Elemento **<param>** passa valores para elementos **<xsl:param>** da folha de estilos

```
<style in="cartao.xml"
       out="cartao.html"
       style="cartao2html.xsl">
  <param name="docsdir"
        expression="/cartoes"/>
</style>
```

build.xml

cartao2html.xsl

```
(...)  
<xsl:param name="docsdir"/>  
(...)  
<xsl:valueof select="$docsdir"/>  
(...)
```

Tarefas úteis (9): JDBC

- **<sql>**: Comunica-se com banco de dados através de um driver JDBC

```
<property name="jdbc.url"
  value="jdbc:cloudscape:rmi://server:1099/Cloud" />
<target name="populate.table">
  <sql driver="COM.cloudscape.core.RmiJdbcDriver"
    url="${jdbc.url}"
    userid="helder"
    password="helder"
    onerror="continue">

    <transaction src="droptable.sql" />
    <transaction src="create.sql" />
    <transaction src="populate.sql" />
    <classpath refid="jdbc.driver.path" />

  </sql>
</target>
```

Tarefas úteis (10): chamadas

- **<ant>**: chama alvo de subprojeto (buildfile externo)

```
<target name="run-sub">
  <ant dir="subproj" />
</target>
```

Chama *alvo default* de *build.xml*
localizado no subdiretório *subproj/*

```
<target name="run-sub">
  <ant dir="subproj" >
    <property name="versao"
      value="1.0" />
  </ant>
</target>
```

Define propriedade que
será lida no outro build.xml

- **<antcall>**: chama alvo local

```
<target name="fazer-isto">
  <antcall target="fazer">
    <param name="oque"
      value="isto" />
  </antcall>
</target>
```

```
<target name="fazer-aquilo">
  <antcall target="fazer">
    <param name="oque"
      value="aquilo" />
  </antcall>
</target>
```

```
<target name="fazer" if="oque">
  <tarefa atributo="{oque}" />
</target>
```

← Template!

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 `festa.wav` será tocado quando o build terminar sem erros fatais. `vaia.wav` tocará se houver algum erro que interrompa o processo:

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

Extensão usando XML

- Como o buildfile é um arquivo XML, pode-se incluir trechos de XML externos através do uso de *entidades externas*

sound.xml

```
<property file="sound.properties" />
<sound>
  <success source="{success.sound}"/>
  <fail source="{fail.sound}"/>
</sound>
```

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE project [
  <!ENTITY sound SYSTEM "sound.xml">
]>
<project default="dtd">
  <description>Gera um DTD para o Ant</description>
  <target name="init">
    &sound;
  </target>
  <target name="dtd" depends="init">
    <antstructure output="ant.dtd" />
  </target>
</project>
```

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/>
 - **Eclipse**
 - <http://eclipse.org>
 - **JEdit** (AntFarm plug-in)
 - <http://www.jedit.org>
 - **Jext** (AntWork plug-in)
 - <ftp://jext.sourceforge.net/pub/jext/plugins/AntWork.zip>

Integração com o JEdit

**Tela do AntFarm mostra alvos do Ant.
Pode-se clicar sobre o alvo para executá-lo**

**Resultados são mostrados
no Console do JEdit**


```
151 .
152 /**
153  * Description of the Method.
154  *
155
158
159
161 XMLScanner xs = u.scanner();
162 Validator v = u.valid:
163 xs.takeStart("message)
164 while (xs.atAttribute
165 String an = xs.ta
166 if (an.equals("ti
```

Ant

Targets:

```
RUN-TESTS message-cleanup deploy CLEAN generate-bean undeploy-all
undeploy-tests deploy-tests clean build-tests test-cleanup undeploy build
[default] init compile JAVADOC DEPLOY
```

- [1] *Richard Hightower e Nicholas Lesiecki. Java Tools for eXtreme Programming. Wiley, 2002. Explora Ant e outras ferramentas em ambiente XP.*
- [3] *Apache Ant User's Manual. Ótima documentação repleta de exemplos.*
- [3] *Steve Lougran. Ant In Anger - Using Ant in a Production Development System. (Ant docs) Ótimo artigo com boas dicas para organizar um projeto mantido com Ant.*
- [4] *Martin Fowler, Matthew Foemmel. Continuous Integration. <http://www.martinfowler.com/articles/continuousIntegration.html>. Ótimo artigo sobre integração contínua e o CruiseControl.*
- [5] *Erik Hatcher. Java Development with Ant. Manning Publications. August 2002. Explora os recursos básicos e avançados do Ant, sua integração com JUnit e uso com ferramentas de integração contínua como AntHill e CruiseControl.*
- [6] *Jesse Tilly e Erik Burke. Ant: The Definitive Guide. O'Reilly and Associates. May 2002. Contém referência completa e ótimo tutorial sobre recursos avançados como controle dos eventos do Ant e criação de novas tarefas.*
- [7] *Karl Fogel. Open Source Development with CVS. Coriolis Press. <http://cvsbook.red-bean.com/>.*



Curso J820
Produtividade e Qualidade em Java:
Ferramentas e Metodologias

Revisão 1.1

© 2002, 2003, Helder da Rocha
(helder@acm.org)

 argonavis.com.br