Manual introdutório do Apache ANT

Escrito por Alexandro Strack
(alexandrostrack@hotmail.com)

Conteúdo

1- Introdução....3
1.1- Visão Geral....3
1.2- História....3
2- Instalação....4
2.1- Obtendo o ANT....4
2.2- Instalando o ANT....4
2.3- Configurações adicionais....5
2.4- Testando a instalação....5
3- Usando o Ant....7
3.1- Estrutura do arquivo de build....7
3.1- Estrutura do arquivo de build....7
3.2- Principais elementos do arquivo build....8
3.2.1- Projects (<project ...>)....8
3.2.2- Targets (<target...>)....9
3.2.3- Tasks....11
3.2.4- Properties....11
3.2.5- Exemplo de um build completo....13
4- Rodando o ANT....14
4.1- A partir da linha de comandos....14
4.2- Integrado com IDEs....15
4.2.1- Eclipse....15
4.2.2- Oracle Jdeveloper....18
5- Ant Tasks....20
5.1- Compilação....21
5.2- Operações com arquivos....23
5.3- Empacotamento e compressão....26
5.4- Gerando documentação....29
5.5- Acesso remoto....30
6- Conceitos e Tipos....33
6.1- Conceitos....33
6.2- Tipos....34
6.2.1- Descrição (description)....34
6.2.2- Operações com estruturas de diretórios....34
6.2.3- PatternSet....36
6.2.4- FileSet....37
6.2.5- DirSet....38
6.2.6- FilterSet....39
6.2.7- Selectors....40
7- Como criar uma nova task....43
8- Próximos passos....45

1- Introdução

1.1- Visão Geral

O ANT é uma ferramenta destinada a construção (build) de programas JAVA. É semelhante a ferramentas como *make*, *nmake*, *jam* mas com o diferencial de ser multi-plataforma, pois é feita em JAVA. Atualmente é considerada como padrão de mercado para a criação de aplicações JAVA.

Um importante diferencial do ANT é a sua capacidade de ser extendido através da definição de novas tarefas, escritas também em JAVA. Desta forma, o desenvolvedor não está limitado a apenas o conjunto de tarefas pré-existentes no ANT ficando livre para incorporar as tarefas que julgar necessário.

Sendo assim, é possível que um mesmo arquivo de *build* seja utilizado em múltiplas plataformas sem necessitar de qualquer tipo de modificação.

1.2- História

O ANT nasceu a partir do projeto TOMCAT, onde era mais um componente. Após a doação do TOMCAT para o Apache Software Foundation (ocupando especificamente um lugar na área de projetos do Apache Jakarta) foi descoberta a grande utilidade do ANT. Neste momento o ANT se desvinculou do TOMCAT tornando-se um projeto independente e ocupando também um lugar na área de projetos do Apache Jakarta. Sua primeira versão independete nasceu em Julho de 2000.

O nome ANT é uma sigla para ***a****nother* ***n****eat* ***t****ool* (*mais uma ferramenta organizada*), segundo seu autor James Duncan Davidson.

2- Instalação

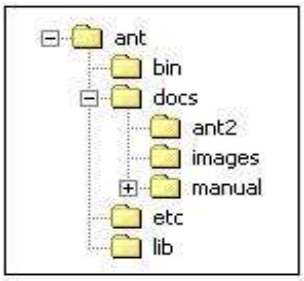
2.1- Obtendo o ANT

O ANT deve ser baixado através do seu web site em http://jakarta.apache.org/ant/ utilizando-se sempre a última versão estável (no momento de escrita deste manual a versão mais estável era 1.6.1). Esta versão é suportada pelo JDK 1.1 ou superiores (recomenda-se no mínimo 1.2).

Conforme dito anteriormente o ANT é uma ferramenta escrita em JAVA e portanto necessita de no mínimo um JRE (Java Runtime Environment) executável para que funcione. Vale lembrar que tendo apenas a JRE instalada muitos dos recursos do ANT não funcionarão. Sendo assim, recomendamos que você tenha instalado um JDK completo (de preferência a última versão estável disponível no site da SUN – http://java.sun.com/getjava).

2.2- Instalando o ANT

A versão de instalação do ANT é distribuída em um arquivo compactado no formato ZIP. Após a descompactação do arquivo de distribuição a estrutura de diretórios do ANT ficará assim:



Se a instalação for em ambiente WINDOWS recomenda-se que não sejam utilizados caminhos que possuam nomes longos, devido a uma limitação do sistema de batch destes sistemas operacionais. O ideal é utilizar caminhos curtos, tradicionais, na forma 8.3 (8 caracteres para o nome e 3 para a extensão) e sem espaços em braco.

Sugestão: c:\ant

Evite: c:\java tools\ant (devido ao espaço em branco entre java e tools).

2.3- Configurações adicionais

Antes que se possa utilizar o ANT é necessário configurar algumas variáveis de ambiente, a saber:

PATH: inclua o caminho ant/bin às demais entradas já existentes.

ANT_HOME: crie uma variável de ambiente com este nome contendo o diretório onde foi instalado o ant. Ex.: ANT_HOME=c:\ant

JAVA_HOME: é opcional. Entretanto para que se possa executar as tarefas baseadas na JDK como *javac*, *jar*, etc, é importante que a biblioteca *classes.jar* ou *tools.jar* (dependendo da sua versão da JDK) seja encontrada. Isto somente ocorrerá se a variável JAVA_HOME existir e estiver corretamente configurada, isto é, apontando para o diretório de instalação da sua JDK. Ex.: JAVA_HOME=C:\j2sdk1.4 .

2.4- Testando a instalação

Para que você possa se assegurar de que a instalação foi bem sucedida e que possui a ferramenta pronta para uso faça o seguinte teste:

- Utilizando a tela de console de comandos digite: ant – version. O resultado deverá ser como o da tela abaixo (dependendo da versão que você tem instalada).

- Outra forma de se fazer o teste é, também através da tela de console de comandos, digitando-se apenas: ant. O resultado deve ser como o visto na tela abaixo.

No caso de algum dos testes acima falhar então você deverá rever os passos de instalação descritos acima. Principalmente se o seu ambiente for Windows, onde os nomes extensos são causadores de uma grande parte das falhas de instalação do ANT. Outro ponto importante que deve ser revisto é a criação das variáveis de ambiente pois sem elas realmente não funcionará.

3- Usando o Ant

3.1- Estrutura do arquivo de *build*

O arquivo utilizado pelo ANT para o *build* da aplicação é um arquivo texto escrito em XML. Normalmente esse arquivo se chama build.xml (mas pode ter qualquer outro nome). Os elementos XML que compõem este arquivo devem fazer parte do conjunto de elementos que o ANT entende, mas novos elementos podem ser facilmente criados (veremos como fazer isso mais adiante neste manual).

O arquivo de *build* consiste basicamente de um elemento `project` que, por sua vez, contém um ou mais elementos `target`. Um target caracteriza uma das etapas que serão executadas na construção (*build*) da aplicação. Estas etapas são constituídas, por exemplo, pela compilação, empacotamento, distribuição, etc. Em cada uma destas etapas uma ou várias operações são executadas. Estas operações são definidas por outros elementos XML que serão detalhados mais adiante.

É possível que toda a construção (build) seja feita através de uma única etapa (target). Entretando isto reduz de forma dramática a possibilidade de reutilização de partes do build, que muitas vezes são suficientes para resolver problemas pontuais. Sendo assim, é comum dividir as etapas em subetapas menores (inclusive com níveis de dependência uma das outras) onde cada etapa possui apenas o número de operações suficientes para concluir o objetivo a que se destina.

O elemento de mais alto nível, `project`, deve informar qual será o `target` padrão a ser executado, caso nenhum seja informado no momento em que se rode o ANT.

Vejamos abaixo um exemplo básico de um arquivo *build*:

```
<?xml version="1.0"?>
<project default="init">
   <target name="init">
   </target>
</project>
```

Como todo arquivo XML, é possível utilizar um elemento que não possua subelementos através de sua declaração simplificada. Ou seja, no exemplo acima o elemento `target` poderia ser escrito da seguinte forma:

```
<target name="init"/>
```

3.2- Principais elementos do arquivo *build*

Conforme vimos no ítem anterior o arquivo de *build* segue uma estrutura bem definida de elementos XML. Para que se possa criar os *builds* de forma eficiente é necessário conhecer os detalhes de seus principais elementos. Então vamos a eles.

3.2.1- Projects (<project ...>)

Todo arquivo *build* deve conter um elemento `project`, afinal a construção de uma aplicação é um projeto com objetivo bem definido. Desta forma temos que identificar neste projeto qual é a tarefa (`target`) padrão, ou seja, aquela que não pode deixar de ser executada caso nenhuma outra seja especificada no momento de execução do ANT. Além disso é interessante que seja descrito o objetivo deste projeto e isso pode ser feito diretamente no próprio elemento `project`, atributo `name`, ou através de um elemento específico para descrições, `description`.

```
<project name="..." default="..." basedir="...">
</project>
```

Atributo	Descrição	Obrigatório
name	Nome do Projeto. Pequena descrição do objetivo deste projeto.	Não
default	Nome da tarefa padrão (target) que deverá ser executada caso nenhuma outra seja especificada no momento de execução do ANT.	Sim
basedir	Diretório base de onde todos os demais paths são derivados. Caso não seja informado então o diretório onde se encontra o arquivi build.xml é assumido como diretório base.	Não

Exemplos:

```
<project default=”init”>
...
</project>


<project name=”Projeto de Exemplo” default=”doIT”>
...
</project>

<project default=”init” basedir=”/src”>
...
</project>

<project name=”Projecto de exemplo.” default=”init”>
      <description>
            Projeto de exemplo com uma descrição mais detalhada usando o
elemento description.
      </description>
</project>
```

3.2.2- Targets (<target...>)

Para que seja feito um build é necessário definir as tarefas que serão executadas. Estas tarefas recebem o nome de *target* no ANT. Uma tarefa é composta de operações (são as chamadas *tasks*, como será visto logo a seguir). Estas tarefas podem depender umas das outras e esta organização de dependência torna cada uma das tarefas mais reutilizável. Através da dependência se garante também que uma tarefa somente será executada depois que suas dependentes tenham sido resolvidas. Para definir uma dependência se utiliza o atributo *depends.*

Através da definição de dependência se consegue garantir uma ordem de execução das tarefas, mas isso não significa que a execução será exatamente na ordem em que a dependência está definida. Vamos a um exemplo:

```
<target name="A"/>
<target name="B" depends="A"/>
<target name="C" depends="B"/>
<target name="D" depends="C,B,A"/>
```

A execução da tarefa B depende da A e logo será executada depois que A termine. Entretanto a tarefa D, que depende de C, B e A, será executada depois da tarefa C pois C já depende de B, que por sua vez depende de A. Sendo assim não se faz necessário executar as tarefas B e A de que D depende pois as mesmas já foram executadas indiretamente. Ou seja, uma tarefa é executada somente uma vez, ainda que várias tarefas dependam dela no mesmo ciclo de execução.

É possível também que uma tarefa venha a ser executada ou não dependendo do valor de uma propriedade (property). Isto é definido através dos atributos *if* e *unless*. Como ainda não tratamos do assunto property então vamos deixar para depois a exemplificação de como utilizar estes dois atributos de *target*.

Atributo	**Descrição**	**Obrigatório**
name	Define o nome da tarefa.	Sim
depends	Define as tarefas dependentes. Se existirem várias dependências então a lista deve ser separada por vírgulas (,).	Não
if	Define o nome da propriedade que deve estar com valor para que a tarefa seja executada.	Não
unless	Define o nome da propriedade que não deve estar com valor para que a tarefa seja executada.	Não
description	Descrição breve do objetivo da tarefa.	Não

ATENÇÃO: é comum que se utilize um hífem (-) antes do nome de uma tarefa para identificar aquelas tarefas que não devem ser chamadas diretamente. Ou seja, são tarefas utilizadas na lista de dependências de outras tarefas (estas últimas é que são chamadas diretamente). Isto é apenas uma regra
e não algo obrigatório.

Exemplos:

```
      <target name=”inicializacao” description=”Prepara o ambiente para a construcao
da aplicacao”>
      ...
      </target>

      <target name=”compilacao” depends=”inicializacao”>
      ...
      </target>

      <target name=”-limpeza” description=”Limpa os diretorios de deploy.”>
      ...
      </target>

      <target name=”deploy” depends=”-limpeza” description=”Realiza o deploy da
aplicacao”>
      ...
      </target>
```

3.2.3- Tasks

As *tasks* são as operações que são realizadas dentro de cada tarefa. Estas operações são na verdade programas que são executados para realizar cada uma das etapas que constituem uma tarefa. Existem muitas operações aceitas pelo ANT e cada uma delas tem a sua lista de atributos. O formato de um elemento *task* é:

```
<name attribute1="value1" attribute2="value2" ... />
```

Onde: name é o nome do programa que será executado, attribute*n* é o nome do parâmetro que é esperado e value*n* é o valor desse parâmetro.

É possível criar um identificador para uma operação. Desta forma a operação poderá ser referenciada através deste identificador. Veja os exemplos abaixo.

```
<taskname id=”taskID” ... />

<script>
      taskID.setDir=”/src/bin”;
</script>
```

Neste último caso a operação *taskID* terá o seu atributo *dir* modificado para conter o valor /*src/bin*.

Para conhecer a lista de operações (tasks) existentes (apenas as principais) veja mais adiante no manual o capítulo chamado ANT Tasks. Além disso na última parte do manual mostraremos como criar uma nova operação (*task*) para o ANT através de um programa em JAVA.

3.2.4- Properties

Um projeto pode possuir propriedades. Estas propriedades de um projeto funcionam como se fossem variáveis em uma linguagem de programação. A única diferença é que depois que a *property* recebe um valor este valor não pode ser modificado. O elemento *property* é uma das possíveis operações (tasks) do ANT e logo possui a mesma regra de formação descrita no ítem anterior. Ou seja:

```
<property name=”fontes” value=”/src” />
```

Depois de criada uma propriedade a mesma pode ser referenciada na forma `${`*nome da*

propriedade}. Vamos a um exemplo:

```
<property name=”fontes” value=”/src” />
<property name=”executaveis” value=”${fontes}/bin” />
```

Além da possibilidade de serem criadas as propriedades que forem necessárias o usuário poderá usufruir das propriedades pré-existentes do ANT. Também é possível utilizar todas as propriedades que são acessíveis através da classe System do JAVA (System.getProperties).

Exemplos:

Propriedades do ANT

basedir: diretório base do projeto. Configurado através do atributo *basedir* do elemento <project>.

ant.file: diretório onde se encontra o arquivo de *build*.

ant.version: versão do ANT.

ant.project.name: nome do projeto que está sendo executado.

ant.java.version: versão da JVM detectada pelo ANT.

Propriedades do System (veja todas as possibilidades no javadoc de System.getProperties)

os.name: nome do sistema operacional

user.name: nome do usuário

3.2.5- Exemplo de um build completo

```
      <?xml version="1.0"?>
      <project default="dist" name="Projeto Manual1">
            <description>Um projeto simples.</description>

            <property name="srcDir" location="src"/>
            <property name="buildDir" location="build"/>
            <property name="distDir" location="dist"/>

            <target name="init">
                  <tstamp/>
                  <mkdir dir="${buildDir}"/>
                  <mkdir dir="${distDir}"/>
            </target>

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

            <target name="dist" depends="compile">
                  <jar destfile="${distDir}/package-${DSTAMP}.jar"
basedir="${buildDir}">
                        <manifest>
                              <attribute name="Built-By" value="${user.name}"/>
                              <attribute name="Main-Class" value="package.Main"/>
                        </manifest>
                  </jar>
                  <jar destfile="${distDir}/package-src-${DSTAMP}.jar"
basedir="${srcDir}"/>
                  </target>

            <target name="clean">
                  <delete dir="${buildDir}"/>
                  <delete dir="${distDir}"/>
            </target>

      </project>
```

4- Rodando o ANT

O ANT é uma ferramenta que pode ser executada de diversas formas mas a que é mais comumente utilizada é a linha de comandos. Entretanto, atualmente, várias IDEs já suportam a criação de arquivos de *build* do ANT desde seus ambientes de desenvolvimento. Vale ressaltar que isso apenas facilita a vida para o desenvolvedor pois o arquivo que será usado para a construção da aplicação é compatível com qualquer forma de execução.

4.1- A partir da linha de comandos

Para executar o ANT a partir da linha de comandos basta digitar `ant`. O resultado será a criação da aplicação seguindo o descrito no arquivo de *build* padrão chamado *build.xml*. Ao digitar *ant* a ferramenta buscará no diretório atual a existência deste arquivo. Caso não encontre será mostrada uma mensagem de erro.

Existem vários parâmetros que podem ser passados na linha de comando que afetam a forma como a ferramenta é executada. Veja a seguir os principais parâmetros que podem ser utilizados.

```
ant [opções] [target [target2 [target3] ...]]

Opções:
  -help                  mostra na tela estas opções que estão aqui listadas.
  -projecthelp           mostra na tela as informações sobre o projeto.
  -version               mostra na tela a versão do ant.
  -diagnostics           mostra na tela informações que podem ajudar a
                         identificar problemas.
  -quiet, -q             não mostra informações na tela.
  -verbose, -v           mostra o máximo de informações possíveis na tela.
  -debug                 mostra na tela as informações para debug de
                         problemas.
  -emacs                 gera informações de log sem formatação.
  -logfile <file>        gera um arquivo de log com o nome especificado.
    -l     <file>                ''
  -logger <classname>    informa a classe responsável pelo log.
  -listener <classname>  informa a instância da classe que servirá de
                         listener para o projeto.
  -buildfile <file>      use como arquivo de build o nome do arquivo
                         especificado.
    -file    <file>              ''
    -f       <file>              ''
  -D<property>=<value>   atribui a uma propriedade do arquivo de build o
                         valor especificado.
  -propertyfile <name>   informa o nome de um arquivo de onde carregar os
                         valores para as propriedades do arquivo de build.
  -inputhandler <class>  nome da classe que será reponsável por tratar a
                         entrada de dados.
  -find <file>           procure por um arquivo de build em toda a árvore de
                         diretórios a partir do ponto onde foi digitado o
```

comando.

Exemplos:

```
ant -version
ant -quiet
ant -Dsrcdir=”/src”
```

4.2- Integrado com IDEs

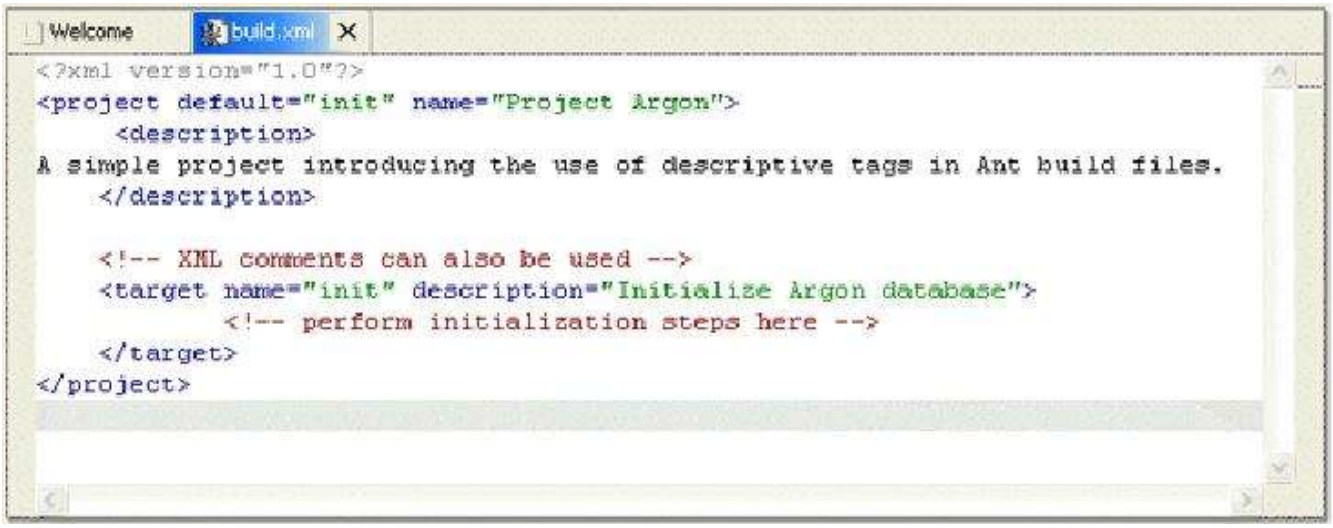
Apesar de ser amplamente utilizado através da linha de comandos, as principais IDEs já disponibilizam mecanismos de integração com o ANT. Isto vem a comprovar o enorme sucesso desta ferramenta que é efetivamente um padrão no desenvolvimento em JAVA.

A lista completa das ferramentas que já estão integradas com o ANT pode ser vista diretamente no seu site em http://ant.apache.org/external.html#IDE%20and%20Editor%20Integration .

Vamos exemplificar esta integração com 2 ferramentas: uma free que é o Eclipse e outra paga, o Oracle Jdeveloper.

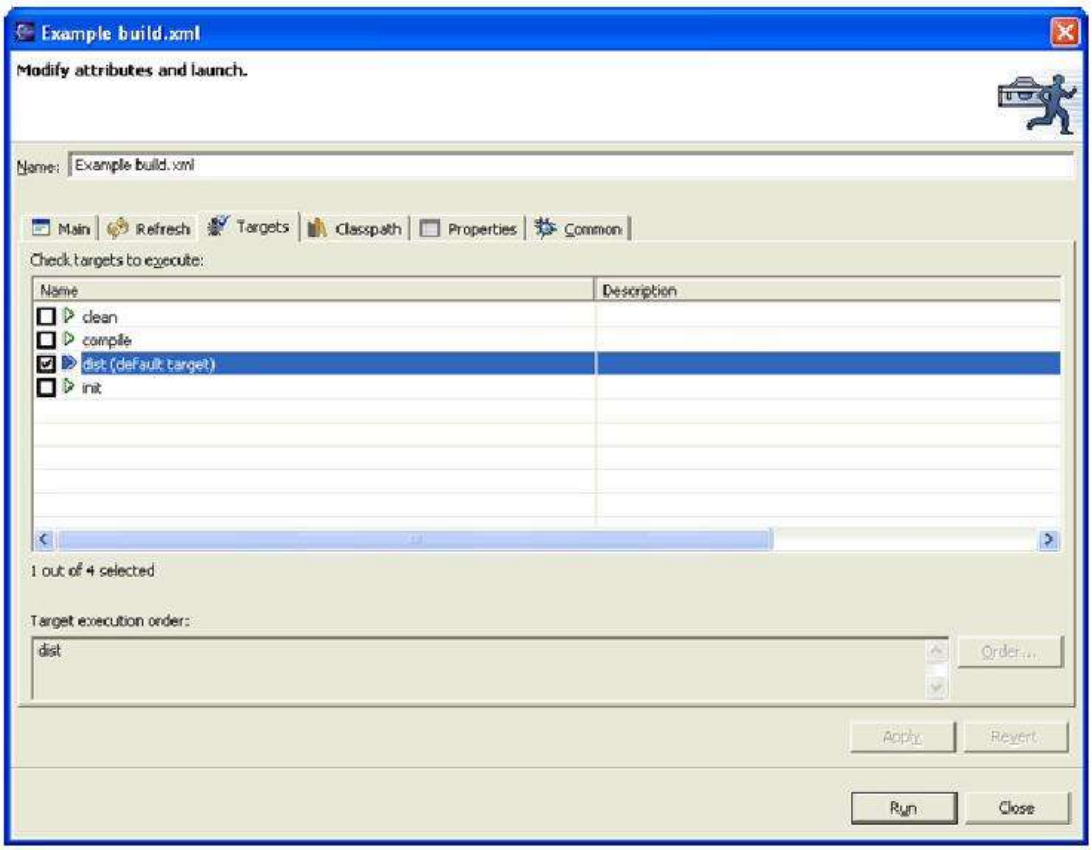
4.2.1- Eclipse

O Eclipse oferece um grande nível de integração com o ANT através do Eclipse ANT Editor. Este editor possui *syntax highlight* além de autocompletar os elementos que compõem a estrutura de um arquivo de *build*.



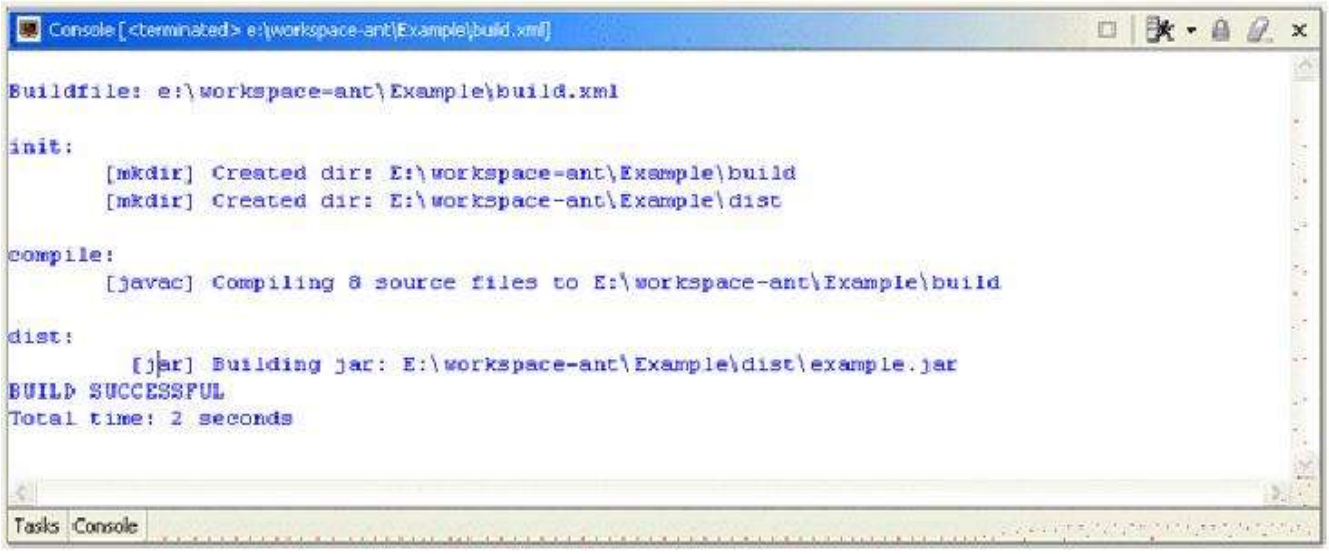
Ao incluir um arquivo build.xml ao seu projeto o Eclipse automáticamente o identificará como sendo um arquivo do ANT (caso você utilize outro nome então deverá rever a associação de arquivos do

Eclipse). Sendo assim ao clicar com o botão direito do mouse sobre o arquivo aparecerá no menu de contexto a opção ***Run Ant...***. Ao escolher esta opção surgirá uma tela como a que está abaixo.

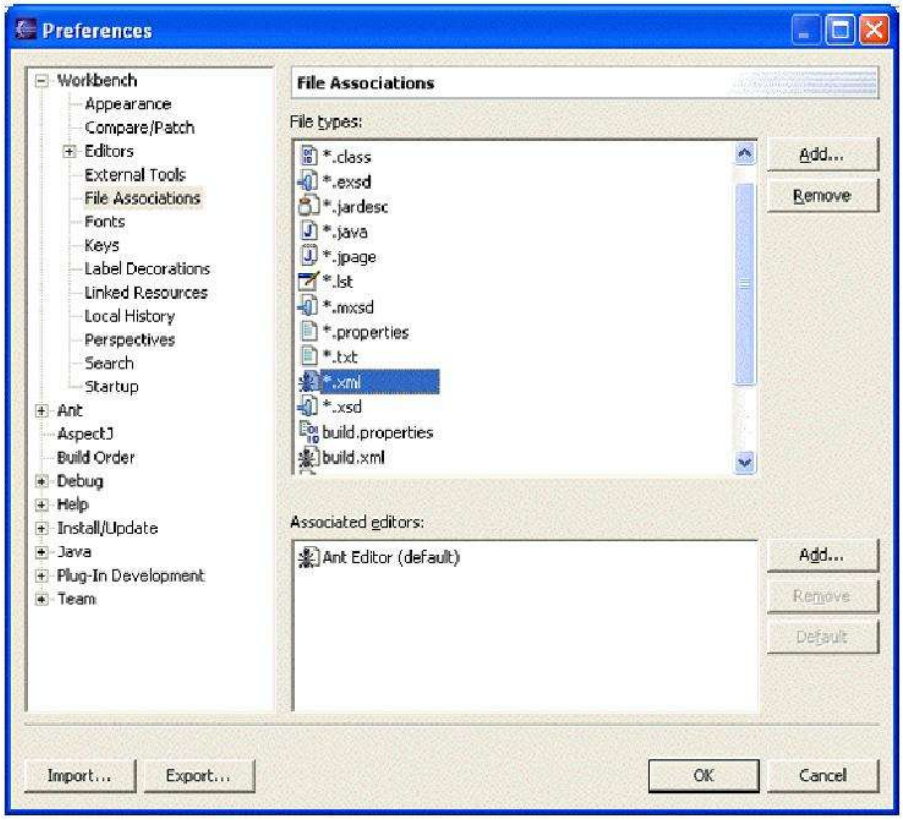


Nesta tela se pode escolher qual tarefa (target) será executada. A tarefa default vem sempre selecionada pois como já vimos anteriormente se nenhuma outra for identificada então esta será executada.

Ao concluir a escolha de tarefas a executar bastará pressionar o botão ***Run*** que a tela de console será apresentada mostrando o resultado da execução do ANT.



Se for necessário rever a associação de arquivos ao Eclipse, devido a utilização de outro nome para o build, que não seja o build.xml, então escolha a opção **Preferences** do menu **Window** e a seguir expanda o grupo **Workbench**. Selecione **File Associations** e coloque os parâmetros conforme a sua necessidade.

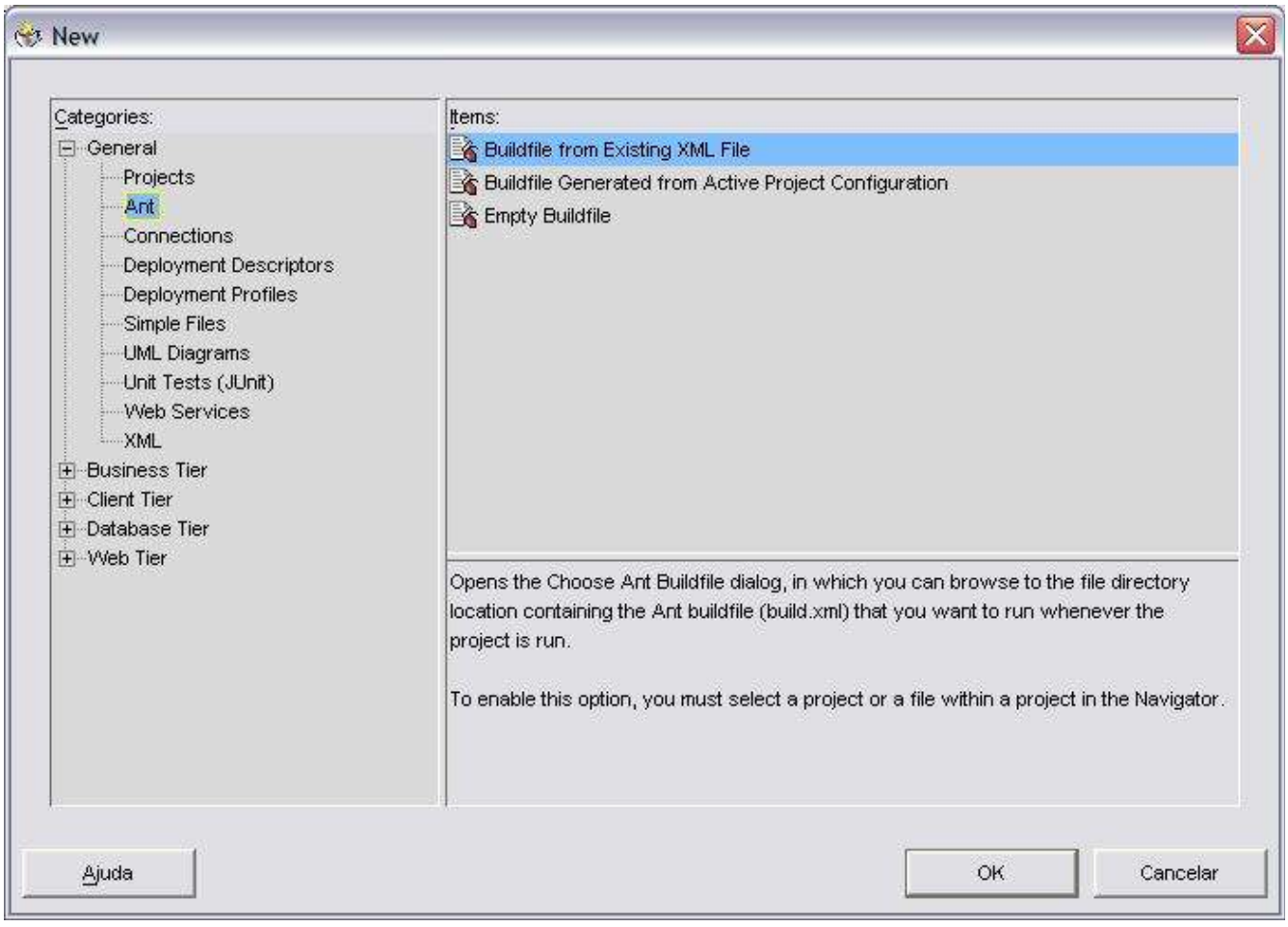


4.2.2- Oracle Jdeveloper

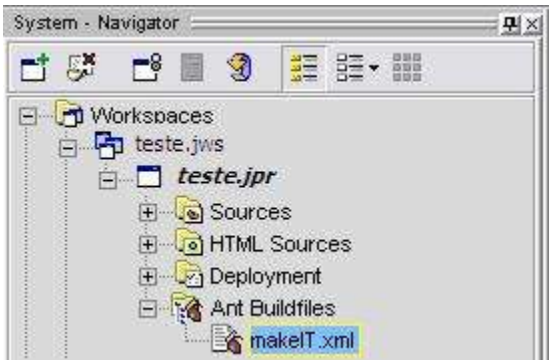
O JDeveloper seguindo a tendência das demais IDEs também possibilita a integração com o ANT, desde que seja uma versão igual ou superior a 1.5. A integração acontece de forma transparente sempre que um arquivo de *build* é adicionado ao projeto.

É possível também criar o arquivo de *build* dentro do projeto mas não existe um editor que dê suporte aos elementos do ANT. Logo a única verificação é quanto a estrutura XML do arquivo e não quanto a sintaxe dos elementos. Mesmo assim se poderá utilizar o editor de XML já existente na IDE para facilitar a edição do arquivo de *build*.

Para incluir um arquivo de build já existente ao seu projeto utilize a opção **File**, ítem **New**. A seguir selecione em **Categories / General** a opção **ANT**. Na lista de opções a esquerda escolha **Buildfile from Existing XML File**. Surgirá então a tela de escolha de arquivos para que seja identificado o arquivo XML que se deseja incluir ao projeto.



Após a escolha do arquivo de *build* o seu projeto deverá ficar da seguinte forma:



Observe que o arquivo de build pode ter qualquer nome.

Uma das possibilidades com a integração é que o arquivo de build poderá ser criado diretamente na IDE. Isto é feito seguindo-se os passos anteriores só que escolhendo a opção **Empty Buildfile**. Neste caso deverá ser informado o diretório e o nome do novo arquivo de *build*.

Para executar um arquivo de build clique com o botão direito sobre o arquivo desejado e escolha a opção **Build Default Target** no menu de contexto. Se não quiser executar a tarefa default então escolha a tarefa desejada através da opção **Build Target** no mesmo menu de contexto (todas as tarefas estarão ali listadas).

5- Ant Tasks

O ANT proporciona um grande número de operações, são mais de 130 opções já disponíveis. Para facilitar o entendimento e o objetivo de cada operação iremos agrupá-las.
Estes grupos sintetizam tudo aquilo que pode ser feito através da ferramenta. Enfim, o que podemos fazer com o ANT?

- Compilar
    - <javac>
    - <csc>
- Gerar Documentação
    - <javadoc>
    - <style>
    - <junitreport>
    - <stylebook>
- Executar programas
    - <java>
    - <ant>
    - <sql>
    - <exec>
- Empacotar e comprimir
    - <jar>
    - <zip>
    - <tar>
    - <ear>
- Expandir, copiar e instalar
    - <copy>
    - <delete>
    - <mkdir>
    - <unjar>
    - <unzip>
- Acesso remoto
    - <ftp>

        - <telnet>
        - <mail>
- Montar componentes
        - <ejb-c>
        - <ejb-jar>
- Testar unidades de código
        - <junit>
- Executar roteiros e sons
        - <script>
        - <sound>
- Criar novas operações (*tasks*) – característica que proporciona ao ANT uma capacidade ilimitada de se expandir.
        - <taskdef>

obs: a lista de operações em cada ítem acima é apenas uma pequena amostra, pois existem muitas outras opções. Para ver a lista completa utilize a documentação oficial do ANT em http://apache.ant.org/docs/manual/anttaskslist.html .

O detalhamento de como funciona cada uma destas operações pode ser encontrado na documentação oficial do ANT. Destacarei algumas operações e técnicas de como utilizá-las de forma produtiva.

5.1- Compilação

<javac> - executa o compilador java. Vale ressaltar que somente serão compilados os fontes que não possuirem seus equivalentes .class ou aqueles que tenham data mais recente que os .class. Ao terminar a compilação é mostrado o número de fontes que foram efetivamente compilados.

Principais parâmetros:

Atributo	Descrição	Obrigatório
srcdir	Localização dos fontes. Diretório de onde serão compilados os arquivos fonte.	Sim

Atributo	Descrição	Obrigatório
destdir	Destino dos arquivos compilados. Diretório onde serão armazenados os arquivos .class.	Não
classpath	Classpath que deverá ser utilizado. Semelhante ao parâmetro -classpath do javac.	Não
debug	Informa se o arquivo compilado deve conter instruções de debug.	Não

Com os parâmetros acima se resolve grande parte das tarefas de compilação mais comuns. Seguem alguns exemplos:

`<javac srcdir="src" destdir="bin" />` - compila os fontes do diretório src gerando os arquivos .class no diretório bin.

`<javac srcdir="src" destdir="bin" debug="true" />` - ídem ao enterior só que incluindo nos arquivos compilados instruções de debug.

Entretanto existem compilações que são mais complicadas, isto é, exigem cuidados especiais. Para estes casos outros parâmetros auxiliam na tarefa de compilação. São eles:

Atributo	Descrição	Obrigatório
includes	Lista separada por vírgulas com os fontes que devem ser incluídos na compilação. Quando não é informada esta lista então todos os fontes do diretório serão compilados.	Não
includesfile	Arquivo contendo a lista dos fontes que devem ser considerados para a compilação.	Não
excludes	Lista separada por vírgulas com os fontes que devem ser excluídos do processo de compilação. Se não for informado não se exclui nenhum arquivo do processo de compilação.	Não
excludesfile	Arquivo contendo a lista dos fontes que devem ser excluídos da compilação.	Não
fork	Indica se a compilação deve ser executada por um JDK a parte.	Não
executable	Contém o caminho completo para o executável responsável pela compilação no caso em que fork for verdadeiro (true). Se não for informado um valor então será utilizado o compilador da VM que está executando o ANT.	Não

Atributo	Descrição	Obrigatório
optimize	Indica quando a compilação deve ser feita de forma otimizada. O padrão é não compilar de forma otimizada.	Não
compiler	Indica o compilador a ser utilizado. Isto significa que podem ser usados outros compiladores java que não sejam o da JDK que executa o ANT. Caso não seja informado um compiler então será utilizado o da VM que está executando o ANT (propriedade build.compiler do arquivo de configurações do ANT).	Não

Exemplos mais complexos:

```
<javac srcdir="${src}"
  destdir="${build}"
  includes="mypackage/p1/**,mypackage/p2/**"
  excludes="mypackage/p1/testpackage/**"
  classpath="xyz.jar"
  debug="on"
/>
```

Neste caso os fontes são obtidos em ${src} e o resultado é colocado em ${build}. No classpath é incluído a biblioteca xyz.jar. Serão compilados todos os fontes que estiverem nos pacotes p1 e p2 com exceção para o sub-pacote testpackage dentro de p1.

Também é possível utilizar alguns parâmetros na forma de elementos desta operação. Isto ocorre porque na verdade com a operação javac se está definindo um grupo de arquivos (*fileset*). Logo todo os elementos usados em *fileset* poderão ser usados junto com o javac. Será visto mais adiante como são criados os conjuntos de arquivos (*fileset*).

Ex.:

```
<javac srcdir="${src}"
  destdir="${build}"
  includes="mypackage/p1/**,mypackage/p2/**"
  excludes="mypackage/p1/testpackage/**">
  <classpath>
     <pathelement path=”xyz.jar” />
  </classpath>
</javac>
```

5.2- Operações com arquivos

É muito comum que no processo de construção (build) de uma aplicação seja definida a estrutura de armazenamento desta aplicação. Ou seja, se define durante o processo de construção a estrutura que irá receber os objetos da aplicação. Isto é feito através da manipulação de arquivos e diretórios. Uma das

vantagens de se utilizar o ANT para esta tarefa é o fato das operações funcionarem, geralmente, em diversas plataformas. Sendo assim o que for definido no arquivo build com relação a manipulação de arquivos funcionará em ambientes UNIX bem como Windows ou qualquer outro que possua um JVM.

Obs: os parâmetros que serão vistos não refletem todas as possibilidades mas sim as que julguei mas importantes e úteis neste momento.

<mkdir> - usado para criar diretórios (ou subdiretórios). Possui um único parâmetro, `dir`, que identifica o nome do diretório a ser criado. Caso o diretório já exista nenhum erro ocorrerá mas obviamente a operação não será realizada.

Atributo	Descrição	Obrigatório
dir	Identifica o nome do diretório que será criado.	Sim

Ex.:

`<mkdir dir="${src}/bin"/>` - cria um diretório bin abaixo do diretório ${src}.
`<mkdir dir="${src}/bin/lib"/>` - cria um diretório lib abaixo de bin.

<copy> - usado para copiar arquivos ou conjunto de arquivos (*fileset*). Os arquivos somente serão copiado se no diretório destino eles não existirem ou forem versões mais recentes. É possível gravar por cima de arquivos já existentes através do parâmetro *overwrite*.

Atributo	Descrição	Obrigatório
file	Nome do arquivo que será copiado.	Sim
tofile	Novo nome do arquivo no diretório destino. Para renomear um arquivo é possível utilizar a operação *move*, como veremos mais adiante.	Não
todir	Diretório de destino do arquivo.	Não
overwrite	Informa se o arquivo poderá ser gravado no diretório destino caso já exista outro mais recente.	Não

Ex.:

`<copy file="${src}/criabanco.java" todir="${src}/oldsources"/>` - copia o arquivo criabanco.java para o diretório oldsource abaixo de ${src}.

```
<copy
      file="${src}/criabanco.java"
      tofile="${src}/oldsources/criabanco_old.java"
      todir="${src}/oldsources"
      overwrite="true"
/>
```
- faz o mesmo que o exemplo anterior mas renomeia o arquivo para criabanco_old.java.

<delete> - usado para apagar um arquivo ou conjunto de arquivos (*fileset*) e até mesmo um diretório com toda a sua estrutura (subdiretórios e arquivos).

Atributo	Descrição	Obrigatório
file	Identifica o arquio a ser apagado.	Sim (um dos dois parâmetros deve existir)
dir	Identifica o diretório a ser apagado.	Sim (um dos dois parâmetros deve existir)

Ex.:

`<delete file="${lib}/mylib.jar"/>` - apaga o arquivo mylib.jar que está no diretório lib.

`<delete dir="${temp}"/>` - apaga o diretório temp e todos os seus subdiretórios e arquivos.

<move> - usado para mover um arquivo para um diretório. Também é possível utilizar esta operação para renomear um arquivo. Ao contrário da operação de cópia (*copy*) o move grava por cima (overwrite) sempre. Para que isso não ocorra é necessário desligar o atributo overwrite.

Atributo	Descrição	Obrigatório
file	Nome do arquivo que será movido.	Sim
tofile	Nome do arquivo no diretório destino.	Não
todir	Diretório destino.	Não
overwrite	Indica se o arquivo movido pode ser gravado por cima de outro com o mesmo nome. O valor padrão é *true*.	Não

Ex.:

`<move file="${bin}/mylib.jar" todir="${lib}"/>` - move o arquivo mylib.jar do diretório ${bin} para o diretório ${lib}.

`<move file="${bin}/mylib.jar" tofile="${bin}/applib.jar"/>` - renomeia o arquivo mylib.jar para applib.jar dentro do mesmo diretório ${bin}.

<replace> - usado para substituir uma string dentro de um arquivo texto por outra informada como atributo. É possível que esta mudança ocorra não apenas em um arquivo mas em vários (neste caso é necessário usar um conjunto de arquivos – *fileset*).

Atributo	Descrição	Obrigatório
file	Nome do arquivo onde ocorrerá a substituição da string.	Sim
dir	Diretório base utilizado no caso de a substituição ocorrer em vários arquivos.	Não
token	String que deverá ser substituída.	Sim
value	String que substituirá a informada em token. Caso não seja informada a nova string então o valor default "" será utilizado.	Não
summary	Informa um resumo da operação após a sua conclusão. Informações como: número de substituições, número de arquivos processados, etc.	Não

Ex.:

```
<replace file="readme.txt" token="[ERR_NUM]" value="1085"/>
```
- substitui no arquivo readme.txt a string [ERR_NUM] por 1085.

```
<replace file="readme.txt" token="[ERR_NUM]" value="1085" summary="true"/>
```
- faz o mesmo que o ítem anterior mas mostra o resultado da operação. Neste caso mostrará o número de substituições efetuadas.

5.3- Empacotamento e compressão

Durante o processo de construção (*build*) de uma aplicação é possível que sejam criados alguns pacotes que facilitam a distribuição da aplicação. Como o ANT pode ser utilizado para outras tarefas além da construção de aplicações também é possível gerar arquivos compactados que venham a faciliatar a distribuição e/ou documentação de arquivos relacionados. Veremos a seguir duas operações destinadas a estes fins.

<jar> - usado para gerar um arquivo JAR. O destino do arquivo (*destfile*) deve ser informado e é possível refinar os arquivos que farão parte do JAR. A estrutura deste elemento define um conjunto de arquivos (*fileset*), logo é possível utilizar atributos como <include>, <exclude>, etc.

O arquivo JAR, além de "juntar" vários arquivos gerando um novo arquivo com extensão JAR, guarda outras informações que são identificadas através do *metainf* e do *manifest*. Sendo assim é posssível utilizar os elementos <metainf> e <manifest> para incluir no JAR as informações necessárias.

Atributo	Descrição	Obrigatório
destfile	Indica o caminho de destino do arquivo JAR.	Sim
basedir	Indica o caminho onde estão os arquivos que serão incluídos no JAR.	Não
compress	Indica se os arquivos deverão ser compactados. O valor default é True.	Não
includes	É uma lista separada por vírgula dos arquivos que devem ser incluídos no JAR.	Não
includesfile	É o nome de um arquivo contendo a lista de arquivos que devem ser incluídos no JAR.	Não
excludes	É uma lista separada por vírgula dos arquivos que não devem ser incluídos no JAR.	Não
excludesfile	É o nome de um arquivo contendo a lista de arquivos que não devem ser incluídos no JAR.	Não
manifest	Indica o nome do arquivo *manifest* que deverá ser incluído no JAR. Pode ser um arquivo ou até mesmo o nome de um outro JAR, neste caso o *manifest* deverá estar localizado em META-INF/MANIFEST.MF	Não

Ex.:

```
<jar destfile="${build}/jars/Exmp.jar" basedir="${fontes}/classes"/>
```
- cria o arquivo Exmp.jar no diretório ${build}/jars contendo todos os arquivos que estão em ${fontes}/classes.

```
<jar destfile="${build}/jars/Exmp.jar"
     basedir="${fontes}/classes"
     excludes="${fontes}/classes/**.old"
/>
```
- faz o mesmo que o descrito no exemplo anterior mas sem incluir os arquivos *.old que estão no diretório ${fontes}/classes.

Elementos opcionais:

<metainf> - neste elemento é possível incluir um conjunto de arquivos (*fileset*) que serão incluídos ao META-INF do arquivo JAR. Se existir um *manifest* neste conjunto de arquivos então o mesmo será ignorado e uma mensagem de *warning* será mostrada.

<manifest> - com este elemento é possível definir o arquivo *manifest* no momento de criação do JAR em lugar de utilizar um arquivo externo. Caso seja definido um arquivo externo através do atributo *manifest* então o *manifest* resultante será o *merge* dos dois arquivos (o informado mais o definido através deste elemento).

Ex.:

```
<jar destfile="exmp.jar" basedir=".">
        <include name="build"/>
        <manifest>
                <attribute name="Built-By" value="${user.name}"/>
                <section name="common/class1.class">
                        <attribute name="Sealed" value="false"/>
                </section>
        </manifest>
</jar>
```
- cria o exmp.jar no diretório raíz e define o conteúdo do arquivo manifest.

<zip> - usado para criar um arquivo ZIP. A estrutura deste elemento define um conjunto de arquivos (*fileset*) e com isso é possível utilizar qualquer um dos sub-elementos de um *fileset*.

Atributo	Descrição	Obrigatório
destfile	Indica o caminho de destino do arquivo ZIP.	Sim
basedir	Indica o caminho onde estão os arquivos que serão incluídos no ZIP.	Não
compress	Indica se os arquivos deverão ser compactados. O valor default é True.	Não
includes	É uma lista separada por vírgula dos arquivos que devem ser incluídos no ZIP.	Não
includesfile	É o nome de um arquivo contendo a lista de arquivos que devem ser incluídos no ZIP.	Não
excludes	É uma lista separada por vírgula dos arquivos que não devem ser incluídos no ZIP.	Não
excludesfile	É o nome de um arquivo contendo a lista de arquivos que não devem ser incluídos no ZIP.	Não
update	Informa se o arquivo ZI P que será gerado deverá substituir outro já existente com o mesmo nome (overwrite) ou se deverá modificá-lo (update - true). Se não for indicado este atributo o arquivo ZIP sempre substitui um já existente.	Não
duplicate	Indica o que fazer no caso de existir um arquivo duplicado. Os valores possíveis são: *add*, *preserve* e *fail*. O valor default é *add*.	Não

Ex.:

```
<zip destfile="${dist}/manual.zip"
     basedir="${build}/manual"
     update="true"
/>
```
- gera um arquivo manual.zip em ${dist} com todos os arquivos de ${build}/manual. Se já existir um arquivo manual.zip então o arquivo será acrescentado dos arquivos que estão em ${build}/manual.

```
<zip destfile="${dist}/manual.zip"
     basedir="${build}/manual"
     excludes="index.html"
/>
```
- faz o mesmo que o exemplo anterior mas exclui o arquivo index.html do arquivo ZIP que será gerado.

5.4- Gerando documentação

Os desenvolvedores JAVA já possuem a possibilidade de documentar enquanto escrevem os programas. A documentação é feita através de *tags* que identificam para o programa JAVADOC o que deve conter o arquivo de documentação. A estrutura da documentação segue o padrão definido pela SUN o que facilita muito a sua utilização por parte dos desenvolvedores. Sendo assim, é necessário que os arquivos sejam processados através do programa JAVADOC para que sejam então produzidos os arquivos HTML contendo a documentação de cada arquivo fonte. Como esta é uma tarefa repetitiva podemos utilizar o ANT para automatizá-la (como veremos a seguir). Existem outras operações para gerar documentação mas seguramente a mais utilizada é a *javadoc*.

<javadoc> - usada para gerar os arquivos de documentação a partir de fontes JAVA. Em outras palavras, executa o JAVADOC sobre um conjunto de arquivos fonte, gerando os arquivos HTML com a documentação.

Atributo	Descrição	Obrigatório
sourcepath	Indica onde devem ser encontrados os arquivos fontes.	Sim(*)
sourcefiles	É uma lista separada por vírgulas com os arquivos fontes que devem ser considerados.	Sim(*)
destdir	Diretório de destino dos arquivos de documentação.	Não
Public	Mostra apenas classes e membros públicos.	Não
Protected	Mostra classes e métodos públicos/protected. Este é o valor default.	Não
Private	Mostra classes e membros públicos/protected e privados.	Não
verbose	Mostra na tela mensagens informando o que está sendo feito durante o processamento de Javadoc.	Não

(*) – pelo menos um destes atributos deve ser informado.

Ex.:

```
<javadoc sourcepath="src"
         destdir="docs/api"
         verbose="true"
```

/> - gera os arquivos de documentação em docs/api de todos os fontes que estão em src. Durante a execução do JAVADOC serão mostradas mensagens informando o que está sendo feito no processamento.

5.5- Acesso remoto

No processo de construção(build) de uma aplicação podemos acessar remotamente alguns arquivos ou até mesmo copiar arquivos para localizações remotas. Também é possível avisar (e até mesmo anexar arquivos), através de email, a realização de uma tarefa. O acesso remoto é útil no processo de construção de aplicações que atualmente se encontram em ambientes cada vez mais distribuídos.

<ftp> - usado para realizar ações de FTP. Esta operação implementa um cliente de FTP simples que permite enviar, receber e apagar arquivos que estejam armazenados remotamente. Esta operação utiliza comandos no formato UNIX.

Atributo	Descrição	Obrigatório
server	Endereço do servidor de FTP.	Sim
port	Número da porta do servidor FTP. O valor default é 21.	Não
userid	Nome do usuário para o servidor de FTP.	Sim
password	Tsenha do usuário para o servidor de FTP.	Sim
remotedir	Nome do diretório para onde serão copiados os arquivos. É o diretório no servidor de FTP.	Não
action	Ação FTP. São suportadas as seguintes ações: "send", "put", "get", "del", "list", "chmod" and "mkdir". O valor default é “send”.	Não
binary	Identifica se o modo é binário (“yes”) ou texto (“no”). O valor default é “yes”.	Não
separator	Identifica o separador de nomes. O valor default é “/”.	Não

Ex.:

```
<ftp server="ftp.apache.org"
     userid="anonymous"
     password="me@myorg.com">
  <fileset dir="htdocs/manual"/>
</ftp>
```

- copia os arquivos do diretório htdocs/manual para o diretório default do usuário anonymous.

```
<ftp action="get"
     server="ftp.apache.org"
     userid="anonymous"
     password="me@myorg.com">
  <fileset dir="htdocs/manual">
    <include name="**/*.html"/>
  </fileset>
```

`</ftp>` - faz download de todos os arquivos .html para o diretório htdocs/manual.

```
<ftp action="del"
     server="ftp.apache.org"
     userid="anonymous"
     password="me@myorg.com">
  <fileset>
    <include name="**/*.tmp"/>
  </fileset>
</ftp>
```

- exclui os arquivos .tmp do diretório default para o usuário anonymous.

<mail> - usado para enviar mensagens (email) SMTP. É possível enviar mensagens em formato texto ou usando arquivos com seu formato MIME identificado.

Atributo	Descrição	Obrigatório
from	Endereço de email de quem está enviando.	Sim
tolist	Lista separada por vírgulas com os emails dos destinatários.	Sim(*)
cclist	Lista separada por vírgulas com os emails das pessoas que serão copiadas.	Sim(*)
bcclist	Lista separada por vírgulas com os emails das pessoas que serão copiadas.	Sim(*)
message	Mensagem a ser enviada. Escrito em formato texto.	Sim(**)
messagefile	Arquivo contendo a mensagem que será enviada.	Sim(**)
messagemimetype	Tipo mime da mensagem. O valor default é text/plain.	Não
files	Lista separada por vírgulas com o nome dos arquivos que serão anexados.	Não
mailhost	Hostname do servidor SMTP. O valor default é localhost.	
mailport	Porta TCP do servidor SMTP. Valor default é 25.	Não
subject	Assunto da mensagem.	Não

(*) – pelo menos um destes atributos deve ser informado.
(**) – pelo menos um destes atributos deve ser informado.

É possível que alguns destes atributos sejam definidos através de sub-elementos. São eles:

<from> / **<to>** / **<cc>** / **<bcc>** - estes elementos tem o mesmo sentido que os atributos *from*, *tolist*, *cclist* e *bcclist* respectivamente.

Atributo	Descrição
name	Nome com que será mostrado (display).
address	Endereço de email.

<message>

Atributo	Descrição
src	Nome do arquivo que contém a mensagem.
mimetype	Tipo mime do arquivo contendo a mensagem.

Se não for informado o atributo *src* então é possível escrever a mensagem diretamente no corpo do elemento *message*.

Ex.:

```
<mail mailhost="smtp.red.com" mailport="1025" subject="Aplicação
      comercial" subject="Nova versão do sistema comercial.">
  <from address="ast@home.com"/>
  <to address="all@xyz.com"/>
  <message>
           A nova versão da aplicação comercial já está
           disponível.
  </message>
</mail>
```

- envia uma mensagem para ast@home.com sobre a nova versão do sistema comercial.

6- Conceitos e Tipos

O ANT trabalha de forma transparente para o usuário novato. Entretanto é importante conhecermos alguns detalhes que possibilitam um maior controle sobre as operações individuais e sobre como o ANT interpreta certos conceitos.

Nesta parte vamos entender os conceitos que permeiam o funcionamento da ferramenta e como podemos criar tipos que facilitarão a definição do que deve ser feito nas operações que suportem este tipos.

6.1- Conceitos

Os conceitos são as premissas pelas quais a ferramenta orienta o seu funcionamento genérico.

- build.sysclasspath: o valor desta propriedade indica como o classpath será interpretado durante a execução do ANT. Os possíveis valores para esta propriedade são:
    - **only**: com este valor será utilizado o classpath definido no sistema. Nenhum outro classpath definido através de atributos será considerado.
    - **ignore**: com este valor o resultado é exatamente oposto ao obtido com *only*. Ou seja, se usa somente o classpath definido nos atributos e se ignora o classpath do sistema.
    - **last**: com este valor o classpath é concatenado a qualquer outro que tenha sido definido por atributos. A concatenação é feita no final do último classpath.
    - **first**: com este atributo qualquer classpath definido através de atributos será concatenado ao classpath do sistema.
- Atributos comuns a todas as operações: todas as operações têm os atributos abaixo em comum. Isto significa que em qualquer operação se pode utilizar os atributos da tabela abaixo.

Atributo	Descrição	Obrigatório
id	Identificador único da operação. Este ID pode ser usado para referenciar a operação dentro dos scripts.	Não
taskname	Nome diferente para a operação dentro desta instância da operação. Este nome aparecerá no log.	Não
description	Espaço para colocar comentários.	Não

6.2- Tipos

Os tipos são utilizados para definir conjuntos de informações que serão manipuladas de forma unitária. Sendo assim são mais comuns os tipos para manipulação de diretórios e arquivos pois, depois de definidos estes tipos, é possível manipular um conjunto de arquivos através de um único nome. Entretanto existem outros tipos que auxiliam na identificação do conjunto de informações sobre o qual se deseja aplicar determinadas operações. Os tipos detalhados a seguir não são todos os existentes mas são aqueles mais utilizados. Para conhecer a lista completa utilize o manual do ANT em http://ant.apache.org/manual .

6.2.1- Descrição (*description*)

Este tipo permite definir uma descrição para o projeto. Esta descrição aparecerá quando for utilizada a opção -projecthelp na linha de comando do ant (ant -projecthelp). Este tipo não possui atributos.

```
<description>
      Descrição do projeto.
</description>
```

6.2.2- Operações com estruturas de diretórios

As operações com estruturas de diretórios são amplamente utilizadas nas demais operações que vimos anteriormente. Existem várias operações que possuem estruturas de diretório internamente como por exemplo <javac>, <zip>, etc. Estas operações utilizam de forma ampla os padrões (patterns) e por isso é fundamental que se tenha esse conhecimento bem fundamentado.

Padrões (patterns)

Padrões são usados para inclusão e exclusão de condições de mapeamento de nomes de diretórios. Os padrões básicos se assemelham muito com os utilizados em comandos DOS ou UNIX, são eles:

* - para mapeamento de um ou mais caracteres. Ex.: *.java – significa que qualquer nome de arquivo permitido que tenha extensão .java será incluído ou excluído da condição ao qual está associada.

? - para mapeamento de apenas um caracter. Ex.: myap?.java – significa que qualquer nome de arquivo que comece com myap e tenha apenas mais uma letra no nome além de extensão .java será

incluído ou excluído da condição ao qual está associada.

Vale lembrar que se pode associar * e ? sem nenhum tipo de limitação. Ex.: / lib/*/app/*.java , /lib/*/app_?/*.html .

Como é possível utilizar * para mapear diretórios e como isso se tornou muito comum para mapear uma estrutura totalmente variável , foi criado o **. Este último identifica que qualquer nível de estrutura de diretórios poderá ser mapeada. Ex.: /lib/**/*.java – isto significa que qualquer estrutura de diretórios abaixo de /lib será considerada válida para encontrar arquivos com extensão .java. Isto facilita muito a escrita pois não é necessário especificar todas as condições de subníveis que se deseja considerar (além de tornar a instrução totalmente reutilizável pois independerá da estrutura que for criada).

Se um padrão for definido com terminador / ou \ o ANT automatocamente considerará a presença de um ** após o terminador. Ex.: /test/ será interpretado como /test/**.

A utilização de padrões (patterns) nas operações com diretórios possibilita a inclusão e exclusão de subconjuntos de arquivos/diretórios. Sendo assim é possível definir um subconjunto com o qual se deseja trabalhar. Existem duas formas de se definir um subconjunto:

1- Através da inclusão de arquivos/diretórios que correspondão a um padrão.

2- Através da exclusão de arquivos/diretórios que correspondão a um padrão.

Ao utilizar ambas condições temos um importante ferramental de definição exata do subconjunto de arquivos/diretórios que se quer utilizar.

Ex.:

```
<copy todir="${dist}">
  <fileset dir="${src}"
           includes="**/images/*"
           excludes="**/*.gif"
  />
</copy>
```

- serão copiados todos os arquivos abaixo de images, respeitando qualquer nível de diretórios desde ${src}, com exceção dos arquivos com extensão .gif.

Existem algumas definições que são excluídas de qualquer operação de diretórios, são elas:

```
**/*~
**/#*#
**/.#*
**/%*%
**/._*
```

```
**/CVS
**/CVS/**
**/.cvsignore
**/SCCS
**/SCCS/**
**/vssver.scc
**/.svn
**/.svn/**
```

Para que não sejam utilizadas estas exclusões é necessário utilizar o atributo `defaultexcludes="no"`.

6.2.3- PatternSet

Os padrões (*patterns*) podem ser organizados em conjuntos (*sets*) e identificados (através do atributo `id`) para posterior utilização. A vantagem de ser definido um *patternset* é que sua reutilização pode ser feita em todo o projeto. Entretanto é possível definir um *patternset* válido somente para uma operação.

Se o objetivo for a reutilização então o *patternset* deve ser definido no mesmo nível que uma tarefa (*target*).

Atributo	Descrição
includes	Lista separada por vírgulas ou espaços dos padrões (*patterns*) de arquivos e diretórios que devem ser inluídos. Se não for especificado um padrão para nome de arquivos então será utilizado * como default.
includesfile	Nome de um arquivo contendo os padrões para inclusão. Cada linha do arquivo deverá conter um padrão.
excludes	Lista separada por vírgulas ou espaços dos padrões (*patterns*) de arquivos e diretórios que serão excluídos. Se não for especificado então nenhum arquivo será excluído(exceção feita apenas para os padrões de exclusões default).
excludesfile	Nome de um arquivo contendo os padrões para exclusão. Cada linha do arquivo deverá conter um padrão.

Ex.:

```
<patternset id="sources"
   includes="src/**/*.java"
   excludes="**/*Test*"
/>
```

- cria um patternset incluindo todos os arquivos com extensão .java que estiverem abaixo de src e exclui tudo que estiver abaixo de diretórios que possuam Test em alguma parte do nome.

É possível criar patternset através de subelementos no lugar de atributos. Os subelementos permitidos são:

<include> / <exclude>

Atributo	Descrição	Obrigatório
name	Nome do padrão a ser incluído/excluído.	Sim
if	Usa o padrão somente se a propriedade estiver com valor.	Não
unless	Usa o padrão somente se a propriedade não estiver com valor.	Não

<includesfile> / <excludesfile>

Os parâmetros são os mesmos de <include> / <exclude>.

Ex.:

```
<patternset id="sources">
  <include name="src/**/*.java"/>
  <include name="**/*.html" if="documentar"/>
  <exclude name="**/*Test*"/>
</patternset>
```

- Faz o mesmo que o exemplo anterior mas inclui os arquivos com extensão .html se a propriedade documentar tiver com valor.

6.2.4- FileSet

Os conjuntos de arquivos (*filesets*), como o próprio nome diz, se trata de um agrupador de arquivos. Este conjunto é definido através de padrões (*patterns*) e pode ser usado internamente a operações que se comportem como agrupadoras de arquivos ou no mesmo nível de tarefas (*target*, onde poderá ser mais reutilizável).

Atributo	Descrição	Obrigatório
dir	Indica o diretório raíz que contén os arquivos deste fileset.	Sim
defaultexcludes	Indica quando as exclusões default (visto anteriormente na parte de padrões) devem ser utilizadas. Se for omitido este parâmetro então será utilizado o conjunto padrão de exclusões.	Não
includes	Lista separada por vírgulas ou espaços dos padrões (*patterns*) de arquivos e diretórios que devem ser inluídos. Se não for especificado um padrão para nome de arquivos então será utilizado * como default.	Não
includesfile	Nome de um arquivo contendo os padrões para inclusão. Cada linha do arquivo deverá conter um padrão.	Não
excludes	Lista separada por vírgulas ou espaços dos padrões (*patterns*) de arquivos e diretórios que serão excluídos. Se não for especificado então nenhum arquivo será excluído(exceção feita apenas para os padrões de exclusões default).	Não

Atributo	Descrição	Obrigatório
excludesfile	Nome de um arquivo contendo os padrões para exclusão. Cada linha do arquivo deverá conter um padrão.	Não

Como o fileset é definido através de patternsets então é permitido que se usem os subelementos de patternsets. Sendo assim são válidos os subelementos: <include> / <exclude> / <includesfile> / <excludesfile>.

Ex.:

```
<fileset dir="${server.src}">
  <include name="**/*.java"/>
  <exclude name="**/*Test*"/>
</fileset>
```
- define um fileset a partir do diretório ${server.src} incluindo todos os arquivos com extensão .java e excluíndo todos os arquivos em diretórios contedo no nome Test.

6.2.5- DirSet

Os conjuntos de diretórios (*dirsets*), como o próprio nome diz, se trata de um agrupador de diretórios. Este conjunto é definido através de padrões (*patterns*) e pode ser usado internamente a operações que se comportem como agrupadoras de arquivos/diretórios ou no mesmo nível de tarefas (*target*, onde poderá ser mais reutilizável).

Atributo	Descrição	Obrigatório
dir	Indica o diretório raíz que contén a estrutura deste dirset.	Sim
includes	Lista separada por vírgulas ou espaços dos padrões (*patterns*) de arquivos e diretórios que devem ser inluídos. Se não for especificado um padrão para nome de arquivos então será utilizado * como default.	Não
includesfile	Nome de um arquivo contendo os padrões para inclusão. Cada linha do arquivo deverá conter um padrão.	Não
excludes	Lista separada por vírgulas ou espaços dos padrões (*patterns*) de arquivos e diretórios que serão excluídos. Se não for especificado então nenhum arquivo será excluído(exceção feita apenas para os padrões de exclusões default).	Não
excludesfile	Nome de um arquivo contendo os padrões para exclusão. Cada linha do arquivo deverá conter um padrão.	Não

Como o dirset é definido através de patternsets então é permitido que se usem os subelementos de

patternsets. Sendo assim são válidos os subelementos: <include> / <exclude> / <includesfile> / <excludesfile>.

Ex.:

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

- define um dirset agrupando todas as classes existentes no diretório classes (lembre-se que podem existir outros arquivos que não sejam classes) mas excluí aquelas que estiverem em deiretórios que contenham no nome a palavra Test.

6.2.6- FilterSet

Os conjuntos de filtros (*filtersets*), como o próprio nome diz, se trata de um agrupador de filtros. Os filtros são definidos como pares de elementos a serem buscados e substituídos. Estes elementos podem ser definidos através de atributos ou arquivo externo. O *filterset* pode ser usado internamente a operações que suportem filtros ou definidos no mesmo nível de tarefas (*target*, onde poderá ser mais reutilizável).

Os filtersets permitem o uso dos atributos `id` e `refid`. Com estes atributos é possível definir um identificador para um filtro e reutilizá-lo através do atributo `refid`.

Atributo	Dscrição	Default	Obrigatório
begintoken	String que identifica o início do elemento que será comparado (token). Ex.: @data@	@	Não
endtoken	String que identifica o fim do elemento que será comparado (token). Ex.: @data@	@	Não

Como definimos antes os *filtersets* trabalham agrupando filtros. Logo necessitamos ver como definir estes filtros.

<filter>

Atributo	Descrição	Obrigatório
token	O elemento a ser buscado e substituído.	Sim
value	O valor a substituir o elemento buscado.	Sim

<filtersfile>

Atributo	Descrição	Obrigatório
file	Arquivo de propriedades contendo os pares elemento a ser buscado/substituído e valor a substituir.	Sim

Ex.:

```
<filterset id="filtrodata" begintoken="%" endtoken="*">
  <filter token="DATE" value="${TODAY}"/>
</filterset>

<copy file="${build.dir}/version.txt" toFile="${dist.dir}/
version.txt">
  <filterset refid="filtrodata"/>
</copy>
```

Neste exemplo se define um filtro que substitui a string DATA pela data atual. Este filtro é reutilizado na operação de cópia do arquivo version.txt para substituir o valor de DATE pela data atual.

6.2.7- Selectors

Os *selectors* são mecanismos de seleção que permitem que um *fileset* seja identificado por outras opções que não apenas <include> / <exclude>. Os selectors são utilizados dentro das definições de *filesets*.

Existem vários tipos de selector mas vamos nos ater aos mais utilizados.

<contains> - usado para selecionar arquivos que possuam uma determinada string.

Atributo	Descrição	Obrigatório
text	Identifica o texto que será buscado.	Sim
casesensitive	Indica se devem ser observados os caracteres em maiúsculas e minúsculas. O valor default é “true”.	Não

Ex.:

```
<fileset dir="${doc.path}" includes="**/*.html">
    <contains text="script" casesensitive="no"/>
</fileset>
```
- define um fileset incluindo arquivos com extensão .html que possuam a palavra script no corpo do arquivo.

<date> - usado para selecionar arquivos que tenham sido alterados antes ou depois de uma data específica.

Atributo	Descrição	Obrigatório
datetime	Indica a data que será usada na validação. O formato segue a string MM/DD/YYYY HH:MM AM_or_PM.	Sim
when	Indica a situação de comparação com a data indicada no primeiro atributo. As possibilidades são: • before – seleciona os arquivos com data de modificação exatamente anterior ao atributo datetime. • after - seleciona os arquivos com data de modificação exatamente posterior ao atributo datetime. • equal – seleciona os arquivos com data igual ao definido em datetime O valor default é “equal”.	Não

Ex.:

```
<fileset dir="${jar.path}" includes="**/*.jar">
    <date datetime="01/01/2001 12:00 AM" when="before"/>
</fileset>
```
- define um fileset incluindo os arquivos com extensão .jar com data de modificação anterior a 01/01/2001 12:00 AM.

<depend> - seleciona arquivos que tenham sido modificados mais recentemente que outras versões do mesmo arquivo em outros diretórios.

Atributo	Descrição	Obrigatório
targetdir	Indica o diretório onde estão os arquivos que serão comparados.	Sim

Ex.:

```
<fileset dir="${ant.1.5}/src/main" includes="**/*.java">
    <depend targetdir="${ant.1.4.1}/src/main"/>
</fileset>
```
- define um fileset contendo apenas os arquivos com extensão .java mais recentes que as existentes no diretório ${ant.1.4.1}/src/main.

<size> - seleciona arquivos que sejam maiores ou menores que um determinado tamanho.

Atributo	Descrição	Obrigatório
value	Indica o tamanho que será usado para o teste.	Sim
units	Indica a unidade de medida em que foi informado o atributo value. Valores possíveis: "k","M", ou "G". O default é não usar este atributo o que faz com que o valor de value seja intepretado em bytes.	Não
when	Indica como deve ser interpretada a comparação com o tamanho informado no atributo value. Valore possíveis: • less – seleciona arquivos com tamanha menor. • more – seleciona arquivos com tamanho maior, • equal – selecion arquivos de igual tamanho. O valor default é “equal”.	Não

Ex.:

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

- define um fileset com todos os arquivos de extensão .jar maiores de 4kb que estão a estrutura abaixo de ${jar.path}.

7- Como criar uma nova task

Como vimos até agora o ANT possui uma quantidade de operações (*tasks*) que possibilitam a realização de muitas atividades. Entretanto, pode existir alguma operação que não exista, mas que você como desenvolvedor já tenha feito um programa JAVA para resolvê-la. Se este é o caso, então você pode extender as operações do ANT incluindo esta operação como mais uma possibilidade.

É possível que você execute programas externos ao ANT através da operação `exec` mas, se o seu programa é em JAVA, fica mais elegante você extender as operações do ANT (até porque fazê-lo não é algo complicado).

Para criar uma operação simples basta derivar a sua classe de `org.apache.tools.ant` e sobre-escrever o método `execute()`. Então o esqueleto de sua nova operação será:

```
import org.apache.tools.ant.BuildException;
import org.apache.tools.ant.Task;

public class myTask extends Task {

      public void execute() throws BuildException {}
}
```

Observe que o método `execute()` dispara uma exceção `BuildException`. Isto significa que se algo der errado então passaremos para o ANT a informação de erro que ele tratará de mostrá-la ao usuário.

As operações normalmente se utilizam de atributos para parametrizar o seu funcionamento. Para fazer isso é necessário criar métodos *set* com o nome do atributo desejado. Por exemplo:

```
public void setMeuAtributo(String str) {
}
```

Note que o parâmetro deste método é uma String pois como vimos os atributos são normalmente configurados através de strings. Vale lembrar que poderíamos utilizar outros tipos mas não vamos nos deter neste ponto aqui.

Como você pode concluir se implementarmos uma classe como descrito acima, incluindo o método `setMeuAtributo()`, o ANT passará a contar com uma nova operação chamada *myTask* que terá como atributo *MeuAtributo*. A utilização desta operação é igual as demais, ou seja:

```
<myTask MeuAtributo=”exemplo” />
```

Desta forma podemos extender facilmente as operações do ANT o que nos dá a possibilidade de criar qualquer operação que seja necessária para as nossas atividades. Lembre-se, entretanto, que muitas vezes é possível utilizar as operações já exsitentes para realizar as nossas tarefas (a lista de operações é grande).

Se o seu caso pede a extensào do ANT então vale a pena se aprofundar mais no assunto e o melhor lugar para encontrar as informações é no próprio manual do ANT em http://ant.apache.org/manual .

8- Próximos passos

Agora que você já conhece a ferramenta ANT e com certeza pode perceber o seu portencial, é importante que se aprofunde no assunto. O ANT pode ser utilizado para trazer produtividade a projetos e aplicações. É amplamente utilizado não só como ferramenta de build mas também como ferramenta de suporte a metodologias de desenvolvimento, como XP. Nesta linha se destaca o conceito de integração contínua além da automatização de testes, integrado ao framework JUnit.

Existe documentação disponível na internet e nos grupos de usuários Java (JUGs) sobre estes assuntos. Uma pesquisa no Google poderá ajudar a selecionar os assuntos que mais lhe agradem sobre o tema ANT.

Sobre o autor

Alexandro Strack é formado em Matemática com pós-graduação em análise e gerência de sistemas pela PUC-RJ. Atua na área de desenvolvimento desde 1989. Desde o início de sua carreira profissional trabalhou com Orientação a Objetos através de linguagens de 3a e 4a gerações. Possui a certificação SCJP 1.4 e trabalha com a tecnologia JAVA desde 2000. Atua neste momento em vários projetos JAVA voltados para web além de ser membro do projeto *open source* Hotwork (http://hotwork.sourceforge.net).

Bibliografia

Manual oficial do ANT – http://ant.apache.org/manual

Apache Ant 101: Make Java builds a snap – http://www.ibm.com/developerWorks

Java Magazine Ed. 2

Tutorial ANT (GUJ) - http://www.guj.com.br/user.article.get.chain?page=1&article.id=30

Construção de Aplicações JAVA com ANT – Helder da Rocha – http://www.argonavis.com.br