02

Definição do classpath e refatoração do build

Caminhos e configurações duplicados

No capitulo anterior vimos como executar e compilar os testes. Criamos um `build.xml` parecido com o abaixo:

```
<project name="agenda" default="compilar">

    <target name="limpar">
        <delete dir="build" />
        <mkdir dir="build" />
        <mkdir dir="build/classes" />
        <mkdir dir="build/classes-teste" />
    </target>

    <target name="compilar" depends="limpar" >
        <javac destdir="build/classes" srcdir="src" >
            <classpath>
            <fileset dir="WebContent/WEB-INF/lib">
                <include name="*.jar" />
            </fileset>
            </classpath>
        </javac>
    </target>

    <target name="compilar-teste" depends="compilar">
        <javac destdir="build/classes-teste" srcdir="src-teste">
            <classpath>
            <pathelement location="build/classes"/>

            <fileset dir="lib-teste">
                <include name="*.jar" />
            </fileset>

            <fileset dir="WebContent/WEB-INF/lib">
                <include name="*.jar" />
            </fileset>
            </classpath>
        </javac>
    </target>

    <target name="executar-teste" depends="compilar-teste">
        <junit showoutput="true"  printsummary="on" >
            <batchtest>
                <formatter type="plain" />
                <fileset dir="src-teste">
                    <include name="**/*Test.java" />
                </fileset>
            </batchtest>
            <classpath>
                <pathelement location="build/classes-teste"/>
                <pathelement location="build/classes"/>
```

```
                <fileset dir="lib-teste">
                    <include name="*.jar" />
                </fileset>

                <fileset dir="WebContent/WEB-INF/lib">
                    <include name="*.jar" />
                </fileset>
                </classpath>
        </junit>
    </target>
</project>
```

Podemos reparar que existem várias definições repetidas dentro do classpath de cada target. Isso com certeza gerará problemas de manutenção já que será preciso alterar o XML em vários lugares para, por exemplo, mudar a pasta das bibliotecas da aplicação.

Refatoração do Classpath

Vamos refatorar o nosso XML, melhorando passo a passo para não repetir nenhuma configuração dentro do `build.xml` . A primeira configuração a melhorar será a pasta `WebContent/WEB-INF/lib` da aplicação. Para ter apenas uma definição central vamos utilizar a tag `path` . O `path` representa um caminho ou `fileset` s. Ele recebe uma identificação para ser referenciado por outras tags:

```
<path id="classpath-compilacao">
    <fileset dir="WebContent/WEB-INF/lib">
            <include name="*.jar" />
    </fileset>
</path>
```

Assim podemos reutilizar o caminho no nosso `build.xml` , basta usar a id. Repare o atributo `refid` da tag `classpath` dentro do target `compilar` :

```
<target name="compilar" depends="limpar" >
    <javac destdir="build/classes" srcdir="src" >
        <classpath refid="classpath-compilacao"/>
    </javac>
</target>
```

O próximo passo é criar o caminho para o `classpath` de teste. Ele é composto das bibliotecas da aplicação (já definido pelo `classpath-compilacao`), das bibliotecas de teste e das classes da aplicação. Claro que queremos reutilizar `classpath-compilacao` aqui também e por isso referenciaremos o caminho novamente. Veja como fica o `path` para compilar as classes de teste composto dos 3 elementos descritos:

```
<path id="classpath-compilacao-teste">
    <path refid="classpath-compilacao" />

    <fileset dir="lib-teste">
            <include name="*.jar" />
    </fileset>
```

```
    <pathelement path="build/classes" />
</path>
```

E o target `compilar-teste` também se refere ao este caminho:

```
<target name="compilar-teste" depends="compilar">
    <javac destdir="build/classes-teste" srcdir="src-teste">
        <classpath refid="classpath-compilar-teste" />
    </javac>
</target>
```

Por último vem a definição do caminho para executar os testes. Esse caminho é composto do classpath de teste (classpath-compilacao-teste) e as classes de testes:

```
<path id="classpath-execucao-teste">
    <path refid="classpath-compilacao-teste" />

    <pathelement path="build/classes-teste" />
</path>
```

E aplicando no target:

```
<target name="executar-teste" depends="compilar-teste">
    <junit showoutput="true"  printsummary="on" >
        <batchtest>
            <formatter type="plain" />
            <fileset dir="src-teste">
                <include name="**/*Test.java" />
            </fileset>
        </batchtest>
        <classpath refid="classpath-execucao-teste" />
    </junit>
</target>
```

O `build.xml` completo fica muito melhor, com as definição dos `path` no inicio:

```
<project name="agenda" default="compilar">

    <path id="classpath-compilacao">
        <fileset dir="WebContent/WEB-INF/lib">
            <include name="*.jar" />
        </fileset>
    </path>

    <path id="classpath-compilacao-teste">
        <path refid="classpath-compilacao" />
        <fileset dir="lib-teste">
            <include name="*.jar" />
        </fileset>
        <pathelement path="build/classes" />
    </path>
```

```
    <path id="classpath-execucao-teste">
        <path refid="classpath-compilacao-teste" />
        <pathelement path="build/classes-teste" />
    </path>

    <target name="limpar">
        <delete dir="build" />
        <mkdir dir="build" />
        <mkdir dir="build/classes" />
        <mkdir dir="build/classes-teste" />
    </target>

    <target name="compilar" depends="limpar" >
        <javac destdir="build/classes" srcdir="src" >
            <classpath refid="classpath-compilacao"/>
        </javac>
    </target>

    <target name="compilar-teste" depends="compilar">
        <javac destdir="build/classes-teste" srcdir="src-teste">
            <classpath refid="classpath-compilacao-teste"/>
        </javac>
    </target>

    <target name="executar-teste" depends="compilar-teste">
        <junit showoutput="true"  printsummary="on" >
            <batchtest>
                <formatter type="plain" />
                <fileset dir="src-teste">
                    <include name="**/*Test.java" />
                </fileset>
            </batchtest>
            <classpath refid="classpath-execucao-teste" />
        </junit>
    </target>
</project>
```

Um caminho aproveita o caminho anterior, criamos uma dependência entre os caminhos:

classpath-execução-teste depende de classpath-compilacao-teste depende de classpath-compilacao.

Propriedades duplicadas

O `build.xml` melhorou mas ainda podemos perceber que há várias propriedades duplicadas e espalhadas. Por exemplo, o nome da pasta para as classes compiladas `build` aparece na definição do caminho `classpath-compilacao-teste` , `classpath-execucao-teste` e nos targets `limpar` , `compilar` e `compilar-teste` . Ele está espalhado por todo arquivo!

Para resolver isso podemos adicionar propriedades no `build.xml` e ao invés de repetir o nome da pasta repetir apenas o nome da propriedade, por exemplo:

```
<property name="build.dir" value="BUILD" />
```

Para utilizar a propriedade existe uma expressão: `${build.dir}` . Para, por exemplo, aplicar a expressão na tarefa `delete` :

```
<delete dir="${build.dir}" />
```

Dessa maneira podemos definir propriedades para todas as pastas usadas nesse `build.xml` :

```
<property name="build.dir" value="BUILD" />
<property name="classes.dir" value="CLASSES" />
<property name="classes.teste.dir" value="CLASSES-TESTE" />
```

e aplicando, por exemplo, no target `limpar` :

```
<target name="limpar">
    <delete dir="${build.dir}" />
    <mkdir dir="${build.dir}" />
    <mkdir dir="${build.dir}/${classes.dir}" />
    <mkdir dir="${build.dir}/${classes.teste.dir}" />
</target>
```

Externalizar as propriedades

As propriedades têm costume de mudar com frequência entre ambientes diferentes. Pode ser útil tirá-las do XML e colocá-las em um arquivo de propriedades separado. Isso simplifica as adaptação e alteração do build já que não mais preciso mexer no XML que possui naturalmente uma sintaxe um pouco mais complexa.

Ou seja, podemos criar um arquivo de propriedades na mesma pasta do `build.xml` com o mesmo nome. Segue o conteúdo do arquivo `build.properties` :

```
build.dir=BUILD
classes.dir=CLASSES
classes.teste.dir=CLASSES-TESTE
```

Com essa definição podemos apagar as propriedades do `build.xml` , mas é preciso importar o arquivo `build.properties` . Para isso vamos também utilizar a tag `property` novamente:

```
<property file="build.properties" />
```