02

Customizando o processo de build com tarefas próprias

Criação de Tarefas

Tarefas próprias são uma alternativa quando não há tasks prontas. A criação de tarefas pode ser necessário para adaptar o build para um uso especifico embora seja mais trabalhosa e exija conhecimento sobre classes específicas do ANT.

Nesse capítulo criaremos uma tarefa própria que o ANT executará pelo `build.xml`. Para isso a equipe de desenvolvedores da ANT criou uma classe mãe que todas as tarefas devem herdar. Essa classe se chama `Task` e vem do pacote `org.apache.tools.ant`.

Vamos começar com uma tarefa simples `HelloANTTask`:

```
package br.com.caelum.ant.task;

import org.apache.tools.ant.Task;

public class HelloANTTask extends Task {

    //deve ter alguma implementação aqui

}
```

A classe `Task` é do JAR `ant.jar` que vem com a distribuição do ANT. Na distribuição há uma pasta `lib`. Com esse JAR no classpath podemos usar a classe `Task` do ANT no nosso projeto.

A nossa tarefa deve executar algo, por exemplo imprimir uma mensagem no console. Esse código deve ficar no método `execute`. O método dever ser sobrescrito, ele vem da classe mãe usando o famoso Design Pattern *Command*.

```
public class HelloANTTask extends Task {

    public void execute() throws BuildException {
        System.out.println("Rodando uma task customizada para meus projetos");
    }
}
```

Declaração da tarefa no build

Agora vamos usar a tarefa nova em um `build.xml`. Para isso é preciso definir a tarefa nesse `build.xml` usando o elemento `taskdef` que recebe o nome classe que criamos no atributo `classname` e também um nome para utilizar dentro do `build.xml`:

```
<project name="ProjetoQueUsaATarefaPropria">

    <taskdef name="tarefaSimples" classname="br.com.caelum.ant.task.HelloANTTask" />
```

```
</project>
```

Depois disso podemos utilizá-la em um alvo (`target`) dentro do mesmo `build.xml` :

```
<project name="ProjetoQueUsaATarefaPropria" default="main">

    <taskdef name="tarefaSimples" classname="br.com.caelum.ant.task.HelloANTTask" />

    <target name="main">
       <tarefaSimples />
    </target>
</project>
```

Só falta chamar esse `build.xml` , mas para isso funcionar o ANT deve encontrar a classe `br.com.caelum.ant.task.TarefaSimples` dentro do classpath de execução. Um jeito simples é gerar um JAR e adicionar esse JAR no classpath do ANT. Com o JAR criado, basta copiar ele, por exemplo, para a pasta `lib` do projeto e usar o parâmetro `-lib` ao executar o comando `ant` .

Estrutura do projeto:

```
projeto/
   build.xml
   lib/tarefa-simples.jar
```

Chamando o ANT na raiz da pasta `projeto` : `ant -lib tarefa-simples.jar` deve aparecer as mensagens:

```
Buildfile: /workspace/projeto/build.xml

main: [tarefaSimples] Rodando uma task customizada para meus projetos

BUILD SUCCESSFUL
Total time: 0 seconds
```

Definição de atributos da Tarefa

Já vimos que as tarefas do ANT podem ser parametrizadas através de atributos ou declarações no corpo da tag. Por exemplo, a tarefa padrão `mkdir` tem o atributo `dir` que define a pasta a ser criada:

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

Vamos escrever uma nova tarefa `mensagem` que há um atributo `prioridade` e recebe uma mensagem no corpo da tarefa, algo como a tarefa abaixo:

```
    <mensagem prioridade="3">
        Hello ANT Task
    </mensagem>
```

Como vimos, qualquer tarefa é representada por uma classe. Nesse exemplo, a classe poderia se chamar `MensagemTask` e claro que ela também teria o método `execute`. Mas agora vamos receber na classe o valor do atributo `prioridade` e o texto no corpo da tag.

Para o atributo `prioridade` é preciso criar um setter na classe `MensagemTask`. O ANT também já converte o valor da prioridade para o tipo especifico do Java, no nosso caso para `int`. O texto da tag podemos receber pelo método `addText`. Veja como fica o código: Queremos

```
public class MensagemTask extends Task {

    private int prioridade;
    private String mensagem;

    @Override
    public void execute() throws BuildException {

        if(prioridade > 1) {
            this.mensagem = this.mensagem.toUpperCase();
        }

        System.out.println(this.mensagem);
    }

    public void addText(String mensagem) {
        this.mensagem = mensagem == null ? "" : mensagem.trim();
    }

    public void setPrioridade(int prioridade) {
        this.prioridade = prioridade;
    }
}
```

Aqui também é preciso criar um JAR que possui a classe da tarefa, adicionar o JAR no classpath do ANT e a `taskdef` no `build.xml`:

A definição das tarefas:

```
<project name="ProjetoQueUsaATarefaMensagem" default="main">

    <taskdef name="mensagem" classname="br.com.caelum.task.MensagemTask" />

</project>
```

e o uso da tarefa dentro de um target:

```
<project name="ProjetoQueUsaATarefaMensagem" default="main">
    <taskdef name="mensagem" classname="br.com.caelum.task.MensagemTask" />

    <target name="main">
        <mensagem prioridade="3">
                    Hello ANT Task
            </mensagem>
```

```
    </target>
</project>
```

Executando o ANT `ant -lib lib/tarefa-simples.jar -f build.xml` imprime na saida:

```
Buildfile: /workspace/projeto/build.xml

main:  [mensagem] HELLO ANT TASK

BUILD SUCCESSFUL
Total time: 0 seconds
```

Classpath da tarefa no build.xml

Vimos que foi necessário chamar o ANT na linha de comando com o parâmetro `-lib` . É facil de esquecer esse parâmetro. Melhor seria se o `build.xml` não depende de um parâmetro externo. Para isso podemos definir o classpath para a execução da tarefa própria no próprio `build.xm` :

Basta definir um caminho que importa o JAR e referenciar esse caminho pela definição do da tarefa própria:

```
<path id="lib.path">
    <fileset dir="lib" includes="*.jar"/>
</path>

<taskdef name="mensagem" classpathref="lib.path" classname="br.com.caelum.task.MensagemTask" />
```