02

Sem Maven é fácil?

Transcrição

Neste curso, analisaremos um projeto que possui código fonte em Java com bibliotecas, arquivos de teste e todos os recursos necessários. A questão é: como transformar tudo isso em um único arquivo e apresentá-lo ao cliente, seja em um formato `.war` ou `.jar`? Como podemos utilizar o código fonte de um projeto e executar o *build* de um artefato final?

Em um primeiro exemplo, estudaremos como se daria esse processo caso ele fosse executado manualmente no terminal. Criaremos uma classe chamada `Calculadora`, supondo que o nosso programa tenha como finalidade realizar cálculos matemáticos.

`Calculadora` recebe os argumentos da linha de comando e imprime o resultado da conta `5+5`.

```
public class Calculadora {
    public static void main(String[] args) {
        System.out.println("5 mais 5 é " + (5+5));
    }
}
```

Salvaremos nosso arquivo, e pronto! Nosso programa está criado e contém apenas um arquivo, e caso necessário, teremos de criar mais, mas no momento criamos apenas um no pacote raiz, de maneira muito simples.

Analisaremos nosso terminal — caso esteja utilizando o sistema operacional Windows, use o comando `dir`, e não `ls`, para acessar os arquivos do diretório. E para compilarmos `Calculadora.java`, faremos uso do `javac`. Digitaremos os seguintes comandos:

```
ls
javac Calculadora.java
```

Devemos fazer esse procedimento todas as vezes que formos compilar um arquivo, e isso pode se tornar muito cansativo, ainda mais se o projeto tiver vários arquivos. Devemos colocá-los na mesma linha de comando, o que pode não ser um bom método para construirmos nossa aplicação.

```
javac Calculadora.java Soma.java Subtracao.java
```

Imagine que façamos alterações simples em nosso código, como calcular novos valores, `6 + 6` ao invés de `5 + 5`. Teríamos de acessar o terminal novamente, e compilar o programa. Além de tudo, o processo está confuso, afinal os conteúdos estão armazenados no mesmo diretório. Para organizarmos melhor o nosso projeto, criaremos um diretório chamado `calculadora`, e moveremos `Calculadora.java` para esse novo diretório.

```
javac Calculadora.java
mkdir calculadora
mv Calculadora.java calculadora/
cd calculadora/
```

Ao abrirmos o diretório `calculadora` clicando em "File > Open...", veremos que `Calculadora.java` está armazenado em seu interior. Há ainda um problema: se deixarmos todos os conteúdos em um mesmo diretório teremos dificuldades para realizar a compilação, pois nesse panorama temos todos os **arquivos fonte** e os **artefatos criados** em mesmo local.

Separaremos a fonte do *build*, e criaremos um arquivo fonte, que em inglês seria *source*. Em programação, é comum utilizarmos a abreviação "src", portanto criaremos um novo diretório com este nome. Moveremos `Calculadora.java` para `src`, e assim começamos a organizar melhor o nosso projeto.

A próxima etapa implica em criarmos um diretório em que depositaremos os arquivos gerados, isto é, criaremos o diretório de destino ou alvo. "Alvo" em inglês é traduzido como *target*, portanto esse será o nome padrão do diretório no qual colocamos nossas classes.

De volta ao terminal, consultaremos com o comando `ls` quais são os nossos arquivos até agora: `src` e `target`, que ainda encontra-se vazio. Queremos compilar `Calculadora.java`, mas este arquivo está no diretório `src`, e não na raiz do projeto, portanto devemos passar a instrução para que o Java compile o arquivo `Calculadora.java` de `sourcepath src` para o diretório `target`, por meio do seguinte comando:

```
javac -sourcepath src -d target src/Calculadora.java
```

Assim, o arquivo será compilado. Se usarmos o comando `ls target/`, teremos que no diretório `target` há `Calculadora.class` e, da mesma forma, se utilizarmos `ls src/`, notaremos que em `src` há `Calculadora.java`.

Os comandos estão bem mais complexos, e devemos fazer o mesmo procedimento para todos os arquivos `.java`. Para adicionar uma dificuldade extra, imagine que nosso projeto utilize uma biblioteca, como o **XStream**, e que os objetos sejam serializados em XML.

Iremos até o navegador e procuraremos por XStream, faremos o download da biblioteca da versão Core (*XStream Core only*) que possui apenas os arquivos `.jar`. Terminado o download temos o arquivo `xstream-1.4.8.jar`, e incorporaremos a biblioteca ao nosso projeto.

Devemos analisar em qual diretório ela será armazenada, `src` ou `target`. Em geral considera-se que bibliotecas sejam *source*, mas criaremos um diretório chamado `lib` (de *library*, ou "biblioteca") seguindo a convenção. Logo, temos três diretórios: `src`, `target` e `lib`.

Como iremos compilar o projeto com base nas fontes do diretório `src`, armazenando em `target` e utilizando a biblioteca de `lib`? Veremos o comando a ser utilizado:

```
javac -sourcepath src -d target -cp lib/xstream-1.47.jar src/Calculadora.java
```

Ao executarmos o programa, veremos que a compilação ocorre, mas notem que há um erro na versão da biblioteca, pois o correto seria `xstream.1.4.8.jar`. Não tivemos nenhuma notificação desse erro, embora não fosse grave! Ao corrigirmos a versão da biblioteca, o programa é novamente compilado, sem que tenhamos ideia se todas as informações estão corretas.

Percebam a dificuldade de fazermos todo o processo de forma manual. O panorama seria ainda mais complexo caso tivéssemos arquivos de teste. Teríamos em nosso projeto um diretório chamado `main`, que abrigaria `Calculadora.java`, e um diretório chamado `test`, que abrigaria os arquivos de teste.

No arquivo `test` criaríamos uma classe `CalculadoraTest`:

```
public class CalculadoraTest {

}
```

Como compilaríamos esse projeto agora? Faremos a compilação dos arquivos do diretório `target` e mandaremos isso para o cliente?

O cliente só deve receber a calculadora, portanto devemos separar o teste no diretório `target` das classes verdadeiras. Ou seja, não é mais no diretório `target` que devemos realizar a compilação, mas sim em `target/classes` . Vamos tentar compilar.

```
javac -sourcepath src -d target/classes -cp lib/xstream-1.4.8.jar src/Calculadora.java
```

Teremos um erro: o arquivo não foi encontrado ("javac: file not found: src/Calculadora.java"). Isso acontece porque mudamos o diretório. Devemos fazer a alteração de `src` para `src/main` .

```
javac -sourcepath src/main -d target/classes -cp lib/xstream-1.4.8.jar src/main/Calculadora.java
```

Ao compilarmos novamente, obteremos um novo erro: não foi encontrado o diretório `target/classes` . Portanto, precisaremos criá-lo.

Feitas as modificações, nosso programa é compilado. Aprendemos quão trabalhoso é lidar com a construção de um projeto manualmente. É possível criar scripts com as mais diversas bibliotecas para facilitar e automatizar o processo de *build* e criação de artefatos.

Com as ferramentas de criação de *build* como o próprio Maven — que suporta Java e outras tantas linguagens —, podemos lidar melhor com os arquivos `.class` do projeto, executar os testes, gerar um arquivo `.jar` de uma biblioteca, fazer com que um projeto dependa de outro, e mais.