02

Adaptando Ivy ao seu projeto

Downloads

Caso queira usar o projeto imediatamente a partir desse vídeo, pode baixar o projeto aqui (http://s3.amazonaws.com/caelum-online-public/PM-77/ivy-cap2-importar.zip). O projeto deve ser importado no Eclipse. Também deve ser instalado o Apache Tomcat.

Apresentação da aplicação Web

Vamos continuar resolvendo dependências com Apache Ivy, mas para esta aula focaremos nas dependências de uma aplicação Web. Ela terá dois tipos (escopos) de dependências. O primeiro são as dependências para rodar e compilar as classes de aplicação, o segundo são as dependências para compilar e rodar os testes. Veremos que será possível gerenciar esses dois tipos de dependências confortavelmente com Ivy.

Nossa aplicação web agenda de exemplo foi estruturada da seguinte maneira: uma pasta `src` com pacotes para controladores, modelo e outro para DAO's e uma pasta `src-teste` com as nossas classes de testes.

As dependências da aplicação também foram definidas: os JAR's da aplicação na pasta `WEB-INF/lib` e os JAR's na pasta `lib-teste` para rodar o junit. Por último, o JAR do Ivy na pasta `ivy-lib` .

Temos o arquivo `ivy.xml` com o elemento `info` e `depedencies` preparados, mas ainda sem dependências concretas.

```
<ivy-module version="2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceS

    <info module="agenda" organisation="br.com.caelum" />

    <dependencies>
    </dependencies>

</ivy-module>
```

Temos também o `build.xml` com o namespace definido na declaração do elemento project, o caminho definido para encontrar o JAR, a tarefa definida através do elemento `taskdef` e também o target do ANT para realmente carregar as bibliotecas.

```
<project name="agenda" xmlns:ivy="antlib:org.apache.ivy.ant" default="atualiza-dependencias">

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

    <taskdef resource="org/apache/ivy/ant/antlib.xml" uri="antlib:org.apache.ivy.ant" classpathr

    <target name="limpar" >
        <delete dir="lib" />
        <mkdir dir="lib" />
    </target>

    <target name="atualiza-dependencias" depends="limpar">
```

```
        <ivy:retrieve />
    </target>

</project>
```

Dependências da aplicação Web

A primeira coisa que precisamos fazer é definir as dependências do projeto web que são compostos de: Spring MVC, Driver MySQL e JSTL. Recorreremos aos repositórios do Maven para obtermos essas dependências.

Em mvnrepository.com (http://mvnrepository.com), após encontrar o Spring MVC mais recente, copiamos a declaração da dependência do Ivy para o arquivo `ivy.xml` . O próximo passo é resolver as dependências do driver do MySQL e do JSTL.

Segue `ivy.xml` com a definição das dependências:

```
<ivy-module version="2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"   xsi:noNamespa

    <info module="agenda" organisation="br.com.caelum" />

    <dependencies>
        <dependency org="org.springframework" name="spring-webmvc" rev="3.1.0.RELEASE" />
       <dependency org="mysql" name="mysql-connector-java" rev="5.1.18" />
       <dependency org="jstl" name="jstl" rev="1.2" />
    </dependencies>

</ivy-module>
```

Para cada dependência queremos que o IVY carregue apenas o JAR's (dependência e dependências transitivas) e nenhuma dependência opcional, nem o javadoc ou código fonte, por isso adicionaremos a configuração `default` para cada dependência.

```
<dependencies>
      <dependency org="org.springframework" name="spring-webmvc" rev="3.1.0.RELEASE" conf="defa
       <dependency org="mysql" name="mysql-connector-java" rev="5.1.18" conf="default"/>
       <dependency org="jstl" name="jstl" rev="1.2" conf="default"/>
</dependencies>
```

Executando o ANT na linha de comando, `ant atualiza-dependencias` , o output gerado pelo IVY mostra que 13 artefatos (no nosso casos JARs) foram baixados:

```
atualiza-dependencias:
[ivy:retrieve] :: Ivy 2.2.0 - 20100923230623 :: http://ant.apache.org/ivy/ ::
[ivy:retrieve] :: loading settings :: url = jar:file:/Users/caelum/Documents/IVY/IVY-workspace/
[ivy:retrieve] :: resolving dependencies :: br.com.caelum#agenda;working@MacBook-Pro-de-Caelum.
[ivy:retrieve]     confs: [default]
[ivy:retrieve]     found org.springframework#spring-webmvc;3.1.0.RELEASE in public
[ivy:retrieve]     found org.springframework#spring-asm;3.1.0.RELEASE in public
[ivy:retrieve]     found org.springframework#spring-beans;3.1.0.RELEASE in public
[ivy:retrieve]     found org.springframework#spring-core;3.1.0.RELEASE in public
```

```
[ivy:retrieve]     found commons-logging#commons-logging;1.1.1 in public
[ivy:retrieve]     found org.springframework#spring-context;3.1.0.RELEASE in public
[ivy:retrieve]     found org.springframework#spring-aop;3.1.0.RELEASE in public
[ivy:retrieve]     found aopalliance#aopalliance;1.0 in public
[ivy:retrieve]     found org.springframework#spring-expression;3.1.0.RELEASE in public
[ivy:retrieve]     found org.springframework#spring-context-support;3.1.0.RELEASE in public
[ivy:retrieve]     found org.springframework#spring-web;3.1.0.RELEASE in public
[ivy:retrieve]     found mysql#mysql-connector-java;5.1.18 in public
[ivy:retrieve]     found jstl#jstl;1.2 in public
[ivy:retrieve] :: resolution report :: resolve 730ms :: artifacts dl 12ms
    ---------------------------------------------------------------------
    |                  |            modules            ||   artifacts   |
    |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
    ---------------------------------------------------------------------
    |      default     |   13  |   0   |   0   |   0   ||   13  |   0   |
    ---------------------------------------------------------------------
[ivy:retrieve] :: retrieving :: br.com.caelum#agenda
[ivy:retrieve]     confs: [default]
[ivy:retrieve]     13 artifacts copied, 0 already retrieved (4796kB/108ms)
```

No projeto, vimos que o IVY criou a pasta `lib` com os 13 JARs baixados, mas os mesmos JARs já estavam na pasta `WEB-INF/lib` para que fosse possível compilar e rodar a aplicação.

Definição do local das dependências

Claro que não queremos repetir os JARs, por isso vamos configurar um `pattern` para ensinar o Ivy que as bibliotecas baixadas devem ser gravadas na pasta `WEB-INF/lib` .

A tarefa `ivy:retrieve` recebe um parâmetro `pattern` . Podemos passar o parâmetro `artifact` , que indica o nome do arquivo e `ext` a extensão. Vamos apagar a pasta `lib` baixada anteriormente e os jars da pasta WEB-INF/lib, o target fica como:

```
<target name="atualiza-dependencias" >
    <ivy:retrieve pattern="WebContent/WEB-INF/lib/[artifact].[ext]" />
</target>
```

Ao executar `ant atualiza-dependencias` e depois atualizando o eclipse é percebido que na pasta apareceram todos os JAR's dentro da pasta definida pela pattern. Podemos perceber que não apareceram as versões dos JAR's.

No pattern podemos também adicionar o `revision` para aparecer a versão do JAR:

```
<target name="atualiza-dependencias" >
    <ivy:retrieve pattern="WebContent/WEB-INF/lib/[artifact]-[revision].[ext]" />
</target>
```

Portando temos mais um problema: os JAR's do junit. Quem deve resolver é o IVY, então vamos procurar pelo junit no repositório do maven (http://mvnrepository.com/) para pegar a versão mais nova:

```
<dependency org="junit" name="junit" rev="4.10" conf="default"/>
```

Vamos copiar a definição da dependência e colar no `ivy.xml` dentro do projeto pedindo para carregar apenas o padrão e nenhuma configuação (`conf="default"`).

Com a nova dependência vamos testar na linha de comando com ANT executando o comando: `ant atualiza-dependencias` :

```
.....
[ivy:retrieve]     found junit#junit;4.10 in public
[ivy:retrieve]     found org.hamcrest#hamcrest-core#jstl;1.1 in public
....
[ivy:retrieve] :: resolution report :: resolve 731ms :: artifacts dl 17ms
    ---------------------------------------------------------------------
    |                  |            modules            ||   artifacts   |
    |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
    ---------------------------------------------------------------------
    |      default     |   15  |   0   |   0   |   0   ||   15  |   0   |
    ---------------------------------------------------------------------
[ivy:retrieve] :: retrieving :: br.com.caelum#agenda
[ivy:retrieve]     confs: [default]
[ivy:retrieve]     15 artifacts copied, 13 already retrieved (322kB/46ms)
```

Carregou o junit, mas a biblioteca foi colocado pelo o IVY dentro da pasta `WebContent` . Os JAR's do junit devem fazer parte da pasta `WEB-INF/lib` . Podemos então apagar esses dois jars.

Separando dependências através de configurações

O que precisamos fazer agora e a separação das dependências, para isso existe o elemento configurations. É possível definir a minha própria configuração, por exemplo, para as bibliotecas da minha aplicação web e também para as bibliotecas que precisamos para rodar os testes. Isso nos possibilita associar uma dependência com a minha própria configuração:

```
<configurations>
   <conf name="webapp" />
   <conf name="teste" />
</configurations>
```

E agora associar com as dependências, por exemplo:

```
<dependencies>
   <dependency org="junit" name="junit" rev="1.4" conf="teste->default" />
   <dependency org="org.springframework" name="spring-mvc" rev="3.1.0.RELEASE" conf="webapp->de
  .....
</dependencies>
```

Assim a configuração `teste` foi mapeado (->) para default `default` para aquele dependência, `wepapp` carregara tudo que é `default` .

Aplicando configurações no build

Voltando ao nosso `build.xml` , na tag `ivy:retrieve` vamos aplicar as configurações `webapp` e `teste` no para associar a tarefa com a configuração, ou seja no final com um grupo de dependências.

Rodando mais uma vez na linha de comando o ANT: `ant atualiza-dependencias` vemos que DOIS relatórios foram gerados: uma para a configuração `webapp` e outro para `teste` :

```
    ---------------------------------------------------------------------
    |                  |            modules            ||   artifacts   |
    |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
    ---------------------------------------------------------------------
    |      webap     |   13  |   0   |   0   |   0   ||   13  |   0   |
    ---------------------------------------------------------------------
.....
    ---------------------------------------------------------------------
    |                  |            modules            ||   artifacts   |
    |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
    ---------------------------------------------------------------------
    |      teste     |   2   |   0   |   0   |   0   ||   2   |   0   |
    ---------------------------------------------------------------------
```

No nosso projeto vemos que a pasta `lib-teste` contem os JARs do junit.

Mais mapeamentos e resumo

Em nosso `ivy.xml` vamos declarar as configurações (por exemplo `webapp`) e as dependencias (por exemplo Spring MVC). A configuração é apenas um nome para indicar a finalidade das dependências. A nossa configuração foi mapeada (->) para alguma configuração dessa dependência, por exemplo, `default` :

```
<dependency org="org.springframework" name="spring-webmvc" rev="3.1.1.RELEASE" conf="webapp->de
```

Existem outras opções para `default` . Poderíamos indicar apenas as dependencias para compilação e mapear `webapp->compile` , ou para rodar a aplicação (`webapp->runtime`), ou `master` indicando apenas o jar principal sem as dependências transitivas.

Outras opções seriam:

- `optional` - o jar principal, todas dependencias transitivas e também os opcionais.
  - `provided` - são dependências que o conteiner fornece.

Além dessas opções que definem a abrangência podemos querer carregar o código fonte também (`sources`), ou a documentação `javadoc` . É possível realizar combinações, por exemplo:

```
webapp->master, sources
webapp->compile, javadoc
teste->*
```

No exemplo acima, `webapp` foi mapeado para master e sources. Isso significa que queremos carregar o jar principal apenas e o código fonte dessa dependência.

No `build.xml` definimos a tarefa para carregar, o `ivy:retrive` . Ela realmente carrega essas dependências definidas no `ivy.xml` . Na tarefa associamos qual configuração queremos carregar e qual é o pattern (caminho de pasta e nomenclature dos JARs).

```
<ivy:retrieve pattern="/[config]/[type]/[artifiact]-[revision].[ext]" conf="webapp" />
```

Nesse caso, o pattern é composto pelo `config` (ou seja `webapp`), o `type` (por exemplo `jar` ou `javadoc`). `artifact` é o nome da dependência (`spring-webmvc`) e `revision` e a versão, além da e a extensão do artifato (`jar` normalmente).