02

Publicar dependências no repositório próprio

Downloads

O projeto utilizado nesse vídeo pode ser baixado aqui (https://s3.amazonaws.com/caelum-online-public/PM-77/security-api.zip). O projeto deve ser importado no Eclipse.

Apresentação do projeto

Temos um projeto já pre-configurado no Eclipse. Ele se chama o `security-api` que possui uma pasta `src` para o código fonte com as classes para simular uma biblioteca de segurança. Dentro do projeto já se encontra a biblioteca do Ivy e os arquivos XMLs `build.xml` e `ivy.xml` .

No `ivy.xml` temos o elemento `info` configurado com o nome do módulo, a organização e uma dependência ao Spring Framework na versão 3.1.1 usando a configuração `default` .

Veja o conteúdo do arquivo `ivy.xml` :

```
<ivy-module version="2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://ant.apache.org/ivy/schemas/ivy.xsd">

    <info module="security-api" organisation="br.com.caelum" />

    <dependencies>
        <dependency org="org.springframework" name="spring-core" rev="3.1.1.RELEASE" conf="defau
    </dependencies>

</ivy-module>
```

O `build.xml` do ANT define a propriedade `build.dir` , o caminho (`path`) e a tarefa para usar o Ivy com o namespace configurado na declaração do projeto. Além disso, existem quatro targets definidos:

-
- clean - target para limpar e recriar a pasta `build`
-
- retrieve - target para carregar as dependências com IVY
-
- compile - target para compilar, depende de clean e retrieve
-
- jar - target para empacotar a aplicação

O conteúdo do `build.xml` fica:

```
<project name="security-api" xmlns:ivy="antlib:org.apache.ivy.ant" >

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

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

    <target name="retrieve" >
        <ivy:retrieve  />
    </target>

    <target name="compile" depends="clean, retrieve">
        <javac srcdir="src" destdir="${build.dir}">
            <classpath>
                <fileset dir="lib">
                    <include name="*.jar" />
                </fileset>
            </classpath>
        </javac>
    </target>

    <target name="jar" depends="compile">
        <jar destfile="${build.dir}/${ant.project.name}.jar" basedir="${build.dir}" />
    </target>

</project>
```

Testando o build

Vamos testar o `build.xml` na linha de comando com ANT, entrando no workspace e na pasta do projeto. Podemos mostrar todos os targets disponíveis com o comando:

```
ant -p
```

Mostra: (`clean` , `compile` , `jar` , `retrieve`)

Depois podemos executar a tarefa jar chamando:

```
ant jar
```

Apresenta:

```
Buildfile: /Users/caelum/Documents/IVY/IVY-workspace/security-api/build.xml

clean:
   [delete] Deleting directory /Users/caelum/Documents/IVY/IVY-workspace/security-api/build
    [mkdir] Created dir: /Users/caelum/Documents/IVY/IVY-workspace/security-api/build
```

```
retrieve:
[ivy:retrieve] :: Ivy 2.2.0 - 20100923230623 :: http://ant.apache.org/ivy/ ::
[ivy:retrieve] :: loading settings :: file = /Users/caelum/Documents/IVY/IVY-workspace/security-
[ivy:retrieve] :: resolving dependencies :: br.com.caelum#security-api;working@MacBook-Pro-de-Ca
[ivy:retrieve]     confs: [default]
[ivy:retrieve]     found org.springframework#spring-core;3.1.1.RELEASE
[ivy:retrieve]     found org.springframework#spring-asm;3.1.1.RELEASE
[ivy:retrieve]     found commons-logging#commons-logging;1.1.1
[ivy:retrieve] :: resolution report :: resolve 273ms :: artifacts dl 5ms
    ---------------------------------------------------------------------
    |                  |            modules            ||   artifacts   |
    |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
    ---------------------------------------------------------------------
    |      default     |   3   |   0   |   0   |   0   ||   3   |   0   |
    ---------------------------------------------------------------------
[ivy:retrieve]
....
compile:
    [javac] /Users/caelum/Documents/IVY/IVY-workspace/security-api/build.xml:27: warning: 'inclu
    [javac] Compiling 1 source file to /Users/caelum/Documents/IVY/IVY-workspace/security-api/bu

jar:
      [jar] Building jar: /Users/caelum/Documents/IVY/IVY-workspace/security-api/build/security-

BUILD SUCCESSFUL
Total time: 2 seconds
```

Na saida fica visível que foram carregadas as dependências do Spring Framework (`ivy:retrieve`). Também foi gerado o JAR da nossa biblioteca security-api que possui todas as classes compiladas.

Definição do artefato a publicar

Para esta seção queremos publicar o arquivo `security-api.jar` dentro de um repositório próprio. O primeiro passo é adicionar um elemento no `ivy.xml` que se chama `publications` , aqui serão declarados todos os artefatos que queremos adicionar dentro do repositório. O elemento `publications` recebe a declaração do artefato a publicar. O artefato em si contem seguintes atributos:

-
- `name` - em nosso caso `security-api`
-
- `ext` - a extensão este artefato é um `ponto jar`
-
- `type` - este artefato é do tipo JAR

Segue o `ivy.xml` com o novo elemento `publications` :

```
<ivy-module version="2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://ant.apache.org/ivy/schemas/ivy.xsd">

    <info module="security-api" organisation="br.com.caelum" />
```

```
    <publications>
          <artifact name="security-api" type="jar" ext="jar"/>
    </publications>

    <dependencies>
        <dependency org="org.springframework" name="spring-core" rev="3.1.1.RELEASE" conf="defau
    </dependencies>

</ivy-module>
```

Tarefa para publicar

Dentro do `ivy.xml` declaramos os artefatos que queremos publicar mas quem fará o trabalho realmente é o ANT. Para isso vamos adicionar um novo target dentro do `build.xml` . Chamaremos esse target de `publish` , porque queremos publicar o nosso JAR. O Ivy oferece uma tarefa, `ivy:publish` , para este fim. A primeira versão do `build.xml` fica:

```
<project name="security-api" xmlns:ivy="antlib:org.apache.ivy.ant" >
    .....
    <target name="publish" depends="jar" >
        <ivy:publish>
            <!-- o que queremos publicar? -->
        </ivy:publish>
    </target>
    .....
</project>
```

Dentro da tarefa `ivy:publish` vamos adicionar os artifacts que gostaríamos de publicar. Definiremos um `pattern` , que indica onde se encontram nossos artifacts. Em nosso caso estão na pasta `build.dir` , seguido pelo o nome do artifacto (`[artifact]` ou seja `security-api`) com a extensão (`[ext]`) `jar` . Veja como fica a definição completa do target:

```
<project name="security-api" xmlns:ivy="antlib:org.apache.ivy.ant" >
    .....
    <target name="publish" depends="jar" >
    <ivy:publish>
        <artifacts pattern="${build.dir}/[artifact].[ext]"/>
    </ivy:publish>
    </target>
    ….
</project>
```

Além disso, do pattern a tarefa `ivy:publish` deveria definir a versão do artefato, em nosso caso 1.0 e o estatus do projeto (`beta, alfa, release`), mas em nosso caso usaremos `RELEASE` . Veja o target no `build.xml` :

```
    <target name="publish" depends="jar" >
        <ivy:publish status="release" pubrevision="1.0" overwrite="true" >
            <artifacts pattern="${build.dir}/[artifact].[ext]"/>
        </ivy:publish>
    </target>
```

Publicação de vários tipos de artefatos

Poderia ser interessante publicar tipo de artefatos diferentes. Além do JAR com os classes compiladas podemos, por exemplo, publicar um JAR com o javadoc, ou um outro com o código fonte. Nesse caso faria sentido distinguir o artefato pelo nome E o tipo. Por isso podemos definir na tarefa `ivy:publish` um `pattern` que define o tipo (`[type]`):

```
<target name="publish" depends="jar" >
    <ivy:publish status="release" pubrevision="1.0" overwrite="true" >
        <artifacts pattern="${build.dir}/[artifact]-[type].[ext]"/>
    </ivy:publish>
</target>
```

Segue um exemplo com dois artefatos no elemento `publications` do `ivy.xml` . Repare que também foi declarado um artefato com o javadoc:

```
<publications>
       <artifact name="security-api-jar" type="jar" ext="jar"/>
       <artifact name="security-api-javadoc" type="javadoc" ext="jar"/>
</publications>
```

Publicação e resolvers

Vamos testar o target `publish` executando o ANT na linha de comando:

```
ant publish
```

Com a saída:

```
.....
jar:
      [jar] Building jar: /Users/caelum/Documents/IVY/IVY-workspace/IVY-repository/security-api/

publish:
BUILD FAILED
/Users/caelum/Documents/IVY/IVY-workspace/IVY-repository/security-api/build.xml:36: no publish
```

Podemos ver que o ANT gerou um erro e acusou um parâmetro resolver. O problema é que Ivy não sabe para onde publicar o nosso JAR. Para isso é preciso criar um arquivo de configuração, o `ivysettings.xml` . Este arquivo define esse local do repositório.

No `ivy.xml` vamos adicionar um elemento `ivysettings` . Dentro dessa tag vem um outro elemento `resolvers` que define o novo repositorio, no nosso caso é baseado no `filesystem` local. Vamos dar um nome para esse repositório: `ivyrepolocal` . Dentro do `filesystem` vamos adicionar novamente um pattern para nossos artifacts, definindo o local onde ficarão os JAR. Veja o `ivysettings.xml` completo:

```
<ivysettings>
   <settings defaultResolver="ivyrepolocal" />
```

```
    <resolvers>
     <filesystem name="ivyrepolocal">
         <artifact pattern="${user.home}/ivyrepolocal/[module]/[artifact]-[revision].[ext]" />
         <ivy pattern="${user.home}/ivyrepolocal/[module]/ivy-[revision].xml" />
     </filesystem>
    </resolvers>
</ivysettings>
```

O `user.home` indica a pasta do usuário. Lá dentro será criado uma pasta `ivyrepolocal`, seguido por uma pasta com o nome do `[module]`, do `[artifact]`, colocando a `[revision]` e a extensão ([ext]). Cada artefato é sempre acompanhado pelo um `ivy.xml`, para isso também definiremos um `ivy pattern`.

Além dos `pattern` s foi definido um elemento para declarar o resolver padrão. No nosso caso é o `ivyrepolocal`. Finalmente usaremos o nome desse `resolver` no target `publish` do `build.xml`:

```
<target name="publish" depends="jar">
     <ivy:publish pubrevision="1.0" status="release" overwrite="true"  resolver="ivyrepolocal">
    <artifacts pattern="${build.dir}/[artifact].[ext]"/>
     </ivy:publish>
</target>
```

Vamos chamar o ANT novamente:

```
ant publish
```

Com a saída:

```
compile:
    [javac] /Users/caelum/Documents/IVY/IVY-workspace/IVY-repository/security-api/build.xml:22:
warning: 'includeantruntime' was not set, defaulting to
build.sysclasspath=last; set to false for repeatable builds
    [javac] Compiling 1 source file to
/Users/caelum/Documents/IVY/IVY-workspace/IVY-repository/security-api/build

jar:
      [jar] Building jar:
/Users/caelum/Documents/IVY/IVY-workspace/IVY-repository/security-api/build/security-api.jar

publish:
[ivy:publish] :: delivering ::
br.com.caelum#security-api;working@MacBook-Pro-de-Caelum.local :: 1.0
:: release :: Wed Aug 15 18:25:23 BRT 2012
[ivy:publish]   delivering ivy file to
/Users/caelum/Documents/IVY/IVY-workspace/IVY-repository/security-api/build/ivy.xml
[ivy:publish] :: publishing :: br.com.caelum#security-api
[ivy:publish]   published security-api to
/Users/caelum/ivyrepolocal/security-api/security-api-1.0.jar
[ivy:publish]   published ivy to
/Users/caelum/ivyrepolocal/security-api/ivy-1.0.xml
```

Foram publicados dentro da pasta do usuário, na pasta `ivyrepolocal` e nossa biblioteca `security-api-1.0.jar` e o XML com o conteúdo do nosso `ivy.xml` .

Baixar dependências do repositório local

O próximo passo é usar essa dependência dentro de um outro projeto. Para isso vamos criar um novo projeto e chamá-lo de `application` . Será usado um projeto java comum. Nesse projeto novo copiaremos o `ivy.xml` e `build.xml` e a bibliteco do IVY do projeto anterior.

O `ivy.xml` será mais simples e não terá o elemento `publications` . E nome do modulo será `application` , `organization` continua da mesma forma. A dependência não será mais o springframework, ela será do módulo `security-api` na versão 1.0:

```
<ivy-module version="2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://ant.apache.org/ivy/schemas/ivy.xsd">

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

  <dependencies>
        <dependency org="br.com.caelum" name="security-api" rev="1.0" conf="default" />
  </dependencies>

</ivy-module>
```

No `build.xml` não publicaremos nada e também não geraremos um JAR. Só vamos compilar e baixar as dependências da nossa aplicação. O `build.xml` fica um pouco mais simples:

```
<project name="security-api" xmlns:ivy="antlib:org.apache.ivy.ant" >

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

    <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" classpa

    <target name="clean">
        <delete dir="${build.dir}" />
        <mkdir dir="${build.dir}" />
    </target>

    <target name="retrieve" >
        <ivy:retrieve />
    </target>

    <target name="compile" depends="clean, retrieve">
        <javac srcdir="src" destdir="${build.dir}">
            <classpath>
                <fileset dir="lib">
                    <include name="*.jar" />
                </fileset>
            </classpath>
```

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

Vamos testar o ANT na linha de comando, entrando no projeto `application</code, chamando:`

```
ant compile
```

```
Buildfile: /Users/admin/dev/IVY-repository/application/build.xml

clean:
   [delete] Deleting directory /Users/admin/dev/IVY-repository/application/build
    [mkdir] Created dir: /Users/admin/dev/IVY-repository/application/build

retrieve:
[ivy:retrieve] :: Ivy 2.2.0 - 20100923230623 :: http://ant.apache.org/ivy/ ::
[ivy:retrieve] :: loading settings :: file = /Users/admin/dev/IVY-repository/application/ivysettings.xml
[ivy:retrieve] :: resolving dependencies :: br.com.caelum#application;working@admins-MacBook-Air.local
[ivy:retrieve]     confs: [default]
[ivy:retrieve]     found br.com.caelum#security-api;1.0 in ivyrepolocal
[ivy:retrieve]     found org.springframework#spring-core;3.1.1.RELEASE in ivyrepolocal
[ivy:retrieve]     found org.springframework#spring-asm;3.1.1.RELEASE in ivyrepolocal
[ivy:retrieve]     found commons-logging#commons-logging;1.1.1 in ivyrepolocal
[ivy:retrieve] downloading /Users/admin/ivyrepolocal/security-api/security-api-1.0.jar ...
[ivy:retrieve] .. (0kB)
[ivy:retrieve] .. (0kB)
[ivy:retrieve]     [SUCCESSFUL ] br.com.caelum#security-api;1.0!security-api.jar (6ms)
[ivy:retrieve] :: resolution report :: resolve 327ms :: artifacts dl 21ms
    ---------------------------------------------------------------------
    |                  |            modules            ||   artifacts   |
    |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
    ---------------------------------------------------------------------
    |      default     |   4   |   1   |   1   |   0   ||   4   |   1   |
    ---------------------------------------------------------------------
[ivy:retrieve] :: retrieving :: br.com.caelum#application
[ivy:retrieve]     confs: [default]
[ivy:retrieve]     1 artifacts copied, 3 already retrieved (0kB/7ms)

compile:
    [javac] /Users/admin/dev/IVY-repository/application/build.xml:23: warning: 'includeantruntime'
BUILD SUCCESSFUL
```

Como o target `compile` usa
retrieve ele carrega as dependências por de baixo dos panos. Carregou quatro jars, o JAR da nossa biblioteca e as dependências transistivas.

Definição do repositório local

Voltando para o Eclipse, atualizando o projeto, podemos ver as 4 dependências, `security-api` e as dependências transitivas. Mas como isso funcionou? Porque dentro deste projeto, em nenhum lugar definimos o repositório local e nossa dependência só se encontra aparentemente no repositório local.

O detalhe que não sabemos ainda é que por padrão, o IVY nem olha no repositorio por padrão. Se o IVY acha dentro da pasta do usuário, uma pasta especial escondida que se chama `.ivy2`, ele procura as dependências ali. Esta pasta é um cache local e se o IVY encontra nesssa pasta os JARs, ele não procurará mais nos repositórios.

Olhando nessa pasta cache achamos nossa dependência de segurança e também as dependências transitivas do apache e do spring framework. Ou seja, apagando essa pasta `.ivy2`, o IVY é obrigatório de procurar as dependências nos repositórios e consequente deve dar erro pois não configuramos o repositório local no projeto `application`.

Vamos então apagar esta pasta e depois chamar novamento o comando `ant compile`:

```
[ivy:retrieve]     confs: [default]
[ivy:retrieve] :: resolution report :: resolve 1662ms :: artifacts dl 0ms
    ---------------------------------------------------------------------
    |                  |            modules            ||   artifacts   |
    |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
    ---------------------------------------------------------------------
    |      default     |   1   |   0   |   0   |   0   ||   0   |   0   |
---------------------------------------------------------------------
[ivy:retrieve]
[ivy:retrieve] :: problems summary ::
[ivy:retrieve] :::: WARNINGS
[ivy:retrieve]         module not found: br.com.caelum#security-api;1.0
[ivy:retrieve]     ==== local: tried
[ivy:retrieve]       /Users/admin/.ivy2/local/br.com.caelum/security-api/1.0/ivys/ivy.xml
[ivy:retrieve]       -- artifact br.com.caelum#security-api;1.0!security-api.jar:
[ivy:retrieve]       /Users/admin/.ivy2/local/br.com.caelum/security-api/1.0/jars/security-api.j
[ivy:retrieve]     ==== shared: tried
[ivy:retrieve]       /Users/admin/.ivy2/shared/br.com.caelum/security-api/1.0/ivys/ivy.xml
[ivy:retrieve]       -- artifact br.com.caelum#security-api;1.0!security-api.jar:
[ivy:retrieve]       /Users/admin/.ivy2/shared/br.com.caelum/security-api/1.0/jars/security-api
[ivy:retrieve]     ==== public: tried
[ivy:retrieve]       http://repo1.maven.org/maven2/br/com/caelum/security-api/1.0/security-api-1
[ivy:retrieve]       -- artifact br.com.caelum#security-api;1.0!security-api.jar:
[ivy:retrieve]       http://repo1.maven.org/maven2/br/com/caelum/security-api/1.0/security-api-1
[ivy:retrieve]         ::::::::::::::::::::::::::::::::::::::::::::::
[ivy:retrieve]         ::          UNRESOLVED DEPENDENCIES         ::
[ivy:retrieve]         ::::::::::::::::::::::::::::::::::::::::::::::
[ivy:retrieve]         :: br.com.caelum#security-api;1.0: not found
[ivy:retrieve]         ::::::::::::::::::::::::::::::::::::::::::::::
[ivy:retrieve]
[ivy:retrieve] :: USE VERBOSE OR DEBUG MESSAGE LEVEL FOR MORE DETAILS

BUILD FAILED
/Users/admin/dev/IVY-repository/application/build.xml:19: impossible to resolve dependencies:
    resolve failed - see output for details
```

IVY tenta carregar a nossa biblioteca `security-api` mas não consegue resolver pois procurou no repositorio padrão, o Maven Repositorio. É claro que nossa biblioteca recentemente criada não está publicada lá.

Para resolver isso vamos configurar o repositório local dentro do projeto `application`. Isto é feito através da mesma configuração já usada dentro do `ivysettings.xml`. Vamos adicionar este arquivo no projeto dentro da pasta `src`:

```
<ivysettings>
    <settings defaultResolver="ivyrepolocal" />
    <resolvers>
          <filesystem name="ivyrepolocal">
        <artifact pattern="${user.home}/ivyrepolocal/[module]/[artifact]-[revision].[ext]" />
                <ivy pattern="${user.home}/ivyrepolocal/[module]/ivy-[revision].xml" />
    </filesystem>
    </resolvers>
</ivysettings>
```

Agora podemos testar na linha de comando:

```
ant compile
```

Com a saída:

```
retrieve:
[ivy:retrieve] :: Ivy 2.2.0 - 20100923230623 :: http://ant.apache.org/ivy/ ::
[ivy:retrieve] :: loading settings :: file = /Users/admin/dev/IVY-repository/application/ivyset
[ivy:retrieve] :: resolving dependencies :: br.com.caelum#application;working@admins-MacBook-Ai
[ivy:retrieve]     confs: [default]
[ivy:retrieve]     found br.com.caelum#security-api;1.0 in ivyrepolocal
[ivy:retrieve] downloading /Users/admin/ivyrepolocal/security-api/security-api-1.0.jar ...
[ivy:retrieve] .. (0kB)
[ivy:retrieve] .. (0kB)
[ivy:retrieve]     [SUCCESSFUL ] br.com.caelum#security-api;1.0!security-api.jar (5ms)
[ivy:retrieve] :: resolution report :: resolve 175ms :: artifacts dl 7ms
    ---------------------------------------------------------------------
    |                  |            modules            ||   artifacts   |
    |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
    ---------------------------------------------------------------------
    |      default     |   2   |   1   |   1   |   0   ||   1   |   1   |
    ---------------------------------------------------------------------
[ivy:retrieve]
[ivy:retrieve] :: problems summary ::
[ivy:retrieve] :::: WARNINGS
[ivy:retrieve]         module not found: org.springframework#spring-core;3.1.1.RELEASE
[ivy:retrieve]     ==== ivyrepolocal: tried
[ivy:retrieve]       /Users/admin/ivyrepolocal/spring-core/ivy-3.1.1.RELEASE.xml
[ivy:retrieve]       -- artifact org.springframework#spring-core;3.1.1.RELEASE!spring-core.jar:
[ivy:retrieve]       /Users/admin/ivyrepolocal/spring-core/spring-core-3.1.1.RELEASE.jar
[ivy:retrieve]         ::::::::::::::::::::::::::::::::::::::::::::::
[ivy:retrieve]         ::          UNRESOLVED DEPENDENCIES         ::
[ivy:retrieve]         ::::::::::::::::::::::::::::::::::::::::::::::
[ivy:retrieve]         :: org.springframework#spring-core;3.1.1.RELEASE: not found
[ivy:retrieve]         ::::::::::::::::::::::::::::::::::::::::::::::
[ivy:retrieve]
[ivy:retrieve] :: USE VERBOSE OR DEBUG MESSAGE LEVEL FOR MORE DETAILS

BUILD FAILED
/Users/admin/dev/IVY-repository/application/build.xml:19: impossible to resolve dependencies: re
```

O IVY achou a dependência `security-api` mas não encontrou as dependências transitivas, os JARs do spring framework.

Configuração da cadeia de repositórios

Não foram encontradas as dependências transitivas. Isso ocorreu porque as dependências transitivas vem do repositório do Maven. Nos definimos apenas um repositório local, agora é preciso definir tambem o do Maven.

Declararemos um novo resolver que representa o repositório do Maven. Este resolver se chama `ibiblio` por padrão. Para usar os dois resolvers, `ibiblio` e `ivyrepolocal`, em conjunto vamos definir uma cadeia de resolvers (`chain`). Veja como que fica o arquivo `ivysettings.xml` :

```
<ivysettings>

<settings defaultResolver="repochain" />

<resolvers>
   <chain name="repochain">
       <ibiblio m2compatible="true" name="mavenrepo"/>
       <filesystem name="ivyrepolocal">
              <artifact pattern="${user.home}/ivyrepolocal/[module]/[artifact]-[revision].[ext]"
              <ivy pattern="${user.home}/ivyrepolocal/[module]/ivy-[revision].xml" />
       </filesystem>
   </chain>
  </resolvers>
</ivysettings>
```

Como `chain` é composto pelo repositório do Maven e nosso repositório local o IVY vai procurar em ambos lugares as dependências. Repare também que o `defaultResolver` se chama `repochain` .

Testando denovo na linha de comando:

```
ant compile
```

```
retrieve:
[ivy:retrieve] :: Ivy 2.2.0 - 20100923230623 :: http://ant.apache.org/ivy/ ::
[ivy:retrieve] :: loading settings :: file = /Users/admin/dev/IVY-repository/application/ivyset
[ivy:retrieve] :: resolving dependencies :: br.com.caelum#application;working@admins-MacBook-Ai
[ivy:retrieve]     confs: [default]
[ivy:retrieve]     found br.com.caelum#security-api;1.0 in ivyrepolocal
[ivy:retrieve]     found org.springframework#spring-core;3.1.1.RELEASE in mavenrepo
[ivy:retrieve]     found org.springframework#spring-asm;3.1.1.RELEASE in mavenrepo
[ivy:retrieve]     found commons-logging#commons-logging;1.1.1 inmavenrepo
[ivy:retrieve] downloading http://repo1.maven.org/maven2/org/springframework/spring-core/3.1.1.
[ivy:retrieve] ..................................................................
[ivy:retrieve] ................
[ivy:retrieve] ........................................
[ivy:retrieve] ............................
[ivy:retrieve] ..........................................
[ivy:retrieve] ............................ (438kB)
[ivy:retrieve] .. (0kB)
```

```
[ivy:retrieve]     [SUCCESSFUL ] org.springframework#spring-core;3.1.1.RELEASE!spring-core.jar
[ivy:retrieve] downloading http://repo1.maven.org/maven2/org/springframework/spring-asm/3.1.1.RI
[ivy:retrieve] ..................................... (51kB)
[ivy:retrieve] .. (0kB)
[ivy:retrieve]     [SUCCESSFUL ] org.springframework#spring-asm;3.1.1.RELEASE!spring-asm.jar (2:
[ivy:retrieve] downloading http://repo1.maven.org/maven2/commons-logging/commons-logging/1.1.1/c
[ivy:retrieve] ........................................ (59kB)
[ivy:retrieve] .. (0kB)
[ivy:retrieve]     [SUCCESSFUL ] commons-logging#commons-logging;1.1.1!commons-logging.jar (204!
[ivy:retrieve] :: resolution report :: resolve 17028ms :: artifacts dl 14712ms
    ---------------------------------------------------------------------
    |                  |            modules            ||   artifacts   |
    |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
    ---------------------------------------------------------------------
    |      default     |   4   |   3   |   3   |   0   ||   4   |   3   |
    ---------------------------------------------------------------------
[ivy:retrieve] :: retrieving :: br.com.caelum#application
[ivy:retrieve]     confs: [default]
[ivy:retrieve]     0 artifacts copied, 4 already retrieved (0kB/5ms)

compile:
    [javac] /Users/admin/dev/IVY-repository/application/build.xml:23: warning: 'includeantruntir

BUILD SUCCESSFUL
```

Foram encontradas todas as bibliotecas, incluindo as dependências transitivas.

Instalando todas as dependências no repositório local

Para nossa aplicação não desejamos usar um repositório publico. Todas as dependências devem se encontrar dentro do repositório local. Então vamos apagar a cadeia de resolver dentro de `ivysettings.xml` e continuar utilizando apenas o `repolocal` como padrão:

```
<ivysettings>
    <settings defaultResolver="ivyrepolocal" />
    <resolvers>
        <filesystem name="ivyrepolocal">
        <artifact pattern="${user.home}/ivyrepolocal/[module]/[artifact]-[revision].[ext]" />
                <ivy pattern="${user.home}/ivyrepolocal/[module]/ivy-[revision].xml" />
    </filesystem>
    </resolvers>
</ivysettings>
```

Todas as dependências devem estar dentro do repositório local. Para nossa aplicação isso significa que além do `security-api` também o spring framework deve ficar no repositório local.

Há uma tarefa do IVY que podemos usar dentro do `build.xml` para copiar qualquer dependência entre repositiórios. No nosso caso queremos copiar o spring framework do repositório do Maven para o nosso repositório local. A tarefa se chama `ivy:install` :

```
<target name="install">
  <ivy:install from="mavenrepo" to="ivyrepolocal"
```

```
        organisation="org.springframework" module="spring-core"
        revision="3.1.1.RELEASE" transitive="true" overwrite="true" />
</target>
```

Aqui declaramos que queremos instalar do (`from`) mavenrepo para (`to`) ivyrepolocal, `overwrite="true"` para sobreescrever e definindo a organização, modulo e revisão.

Chamando a tarefa com Ant:

```
ant install
```

Mostra a saída:

```
install:
[ivy:install] :: Ivy 2.2.0 - 20100923230623 :: http://ant.apache.org/ivy/ ::
[ivy:install] :: loading settings :: file = /Users/admin/dev/IVY-repository/security-api/ivyset
[ivy:install] :: installing org.springframework#spring-core;3.1.1.RELEASE ::
[ivy:install] :: resolving dependencies ::
[ivy:install]     found org.springframework#spring-core;3.1.1.RELEASE in mavenrepo
[ivy:install]     found org.springframework#spring-asm;3.1.1.RELEASE in mavenrepo
[ivy:install]     found commons-logging#commons-logging;1.1.1 in mavenrepo
[ivy:install]     found commons-collections#commons-collections;3.2 in mavenrepo
[ivy:install]     found log4j#log4j;1.2.15 in mavenrepo
[ivy:install]     found net.sf.jopt-simple#jopt-simple;3.0 in mavenrepo
[ivy:install]     found org.apache.ant#ant;1.7.0 in mavenrepo
[ivy:install]     found org.apache.ant#ant-launcher;1.7.0 in mavenrepo
[ivy:install]     found org.aspectj#aspectjweaver;1.6.8 in mavenrepo
[ivy:install] :: downloading artifacts to cache ::
[ivy:install] downloading ….....
[ivy:install] .. (0kB)
[ivy:install]     [SUCCESSFUL ] org.apache.ant#ant;1.7.0!ant.jar (7431ms)
[ivy:install] downloading http://repo1.maven.org/maven2/org/apache/ant/ant-launcher/1.7.0/ant-la
[ivy:install] ..... (11kB)
[ivy:install] .. (0kB)
[ivy:install]     [SUCCESSFUL ] org.apache.ant#ant-launcher;1.7.0!ant-launcher.jar (608ms)
[ivy:install] :: installing in ivyrepolocal ::
[ivy:install]     published spring-core to /Users/admin/ivyrepolocal/spring-core/spring-core-3.1
[ivy:install]     published spring-core to /Users/admin/ivyrepolocal/spring-core/spring-core-3.1
[ivy:install]     published ivy to /Users/admin/ivyrepolocal/spring-core/ivy-3.1.1.RELEASE.xml
[ivy:install]     published spring-asm to /Users/admin/ivyrepolocal/spring-asm/spring-asm-3.1.1
[ivy:install]     published ivy to /Users/admin/ivyrepolocal/spring-asm/ivy-3.1.1.RELEASE.xml
[ivy:install]     published commons-logging to /Users/admin/ivyrepolocal/commons-logging/common
[ivy:install]     published ivy to /Users/admin/ivyrepolocal/commons-logging/ivy-1.1.1.xml
[ivy:install]     published commons-collections to /Users/admin/ivyrepolocal/commons-collection
[ivy:install]     published ivy to /Users/admin/ivyrepolocal/commons-collections/ivy-3.2.xml
[ivy:install]     published log4j to /Users/admin/ivyrepolocal/log4j/log4j-1.2.15.jar
[ivy:install]     published ivy to /Users/admin/ivyrepolocal/log4j/ivy-1.2.15.xml
[ivy:install]     published jopt-simple to /Users/admin/ivyrepolocal/jopt-simple/jopt-simple-3.0
[ivy:install]     published ivy to /Users/admin/ivyrepolocal/jopt-simple/ivy-3.0.xml
[ivy:install]     published aspectjweaver to /Users/admin/ivyrepolocal/aspectjweaver/aspectjweav
[ivy:install]     published ivy to /Users/admin/ivyrepolocal/aspectjweaver/ivy-1.6.8.xml
[ivy:install]     published ant to /Users/admin/ivyrepolocal/ant/ant-1.7.0.jar
[ivy:install]     published ivy to /Users/admin/ivyrepolocal/ant/ivy-1.7.0.xml
[ivy:install]     published ant-launcher to /Users/admin/ivyrepolocal/ant-launcher/ant-launcher
```

```
[ivy:install]     published ivy to /Users/admin/ivyrepolocal/ant-launcher/ivy-1.7.0.xml
[ivy:install] :: install resolution report ::
[ivy:install] :: resolution report :: resolve 0ms :: artifacts dl 48147ms
    ---------------------------------------------------------------------
    |                  |            modules            ||   artifacts   |
    |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
    ---------------------------------------------------------------------
    |      default     |   9   |   6   |   6   |   0   ||   10  |   7   |
        ---------------------------------------------------------------------

BUILD SUCCESSFUL
Total time: 1 minute 9 seconds
```

Foram instalados todas as dependências e dependências transitivas no repositório local.

Resumo

Nessa aula vimos como trabalhar com dois repositórios. O primeiro era o repositório publico do Maven, o segundo um repositório local. O repositório local era preciso porque criamos uma biblioteca própria e não queríamos disponibilizar ela dentro do repositório publico do Maven. Usamos a tarefa `ivy:publish` para publicar o artefato dentro do nosso repositório local. Dessa maneira outras maquinas e projetos podem declarar a dependência também referenciando o repositório local. Para definir os repositórios usamos o arquivo de configuração, `ivysettings.xml` . Caso quiséssemos ficar independente do repositório publico do Maven, é preciso instalar todas as dependências e dependências transitivas no repositório local. Fizemos isso através da tarefa do `ivy:install` .