02

Deployar e rodar a aplicação no servidor

Criação do WAR para o deploy automático

Disponibilizar a aplicação dentro de um servidor de aplicação (deploy) faz parte do ciclo da vida da aplicação e é uma tarefa recorrente no dia-a-dia. Até para a mesma versão da aplicação é preciso repetir o deploy várias vezes pois há ambientes diferentes como o de desenvolvimento, teste, homologação ou produção. O deploy torna-se uma tarefa repetitiva e é um candidato ideal para ser automatizado com ANT.

Para este capítulo usaremos uma aplicação web que será disponibilizada dentro de um servlet container. Para isso é preciso empacotar a aplicação como web archive (WAR) para depois subir e rodar no servidor. E o ANT já possui tarefas prontas para ajudar.

Nesse exemplo, ilustraremos a configuração do empacotamento baseado em projetos criados pela IDE Eclipse, portanto, alguns detalhes podem mudar se estivermos trabalhando com outra IDE.

Por padrão, o Eclipe utiliza a pasta `WebContent` como raiz da aplicação web, ou seja, tudo que é essencial para o funcionamento da aplicação está nela, inclusive as classes do projeto, que encontram-se na pasta WEB-INF/classes.

A tarefa para gerar o web archive se chama `war` e recebe a pasta raiz do nosso projeto web e as classes e arquivos de configurações que dependemos. Pela tag `fileset` informamos à tarefa `war` do nome da pasta raiz, e pela tag `classes` definimos a pasta com as classes e arquivos de configurações.

```
<war ... >
     <fileset dir="WebContent" />
     <classes dir="build/classes" />
</war>
```

Além das configurações básicas da aplicação web é preciso informar para a tarefa `war` onde se encontra o arquivo `web.xml` e nome e o local do arquivo `WAR` gerado. Veja a tarefa completa:

```
<war destfile="build/agenda.war" webxml="WebContent/WEB-INF/web.xml">
     <fileset dir="WebContent" />
     <classes dir="build/classes" />
</war>
```

Falta apenas englobarmos nossa tag `war` em uma tarefa que dependa do target `compilar` :

```
<target name="empacotar" depends="compilar" >
 <war destfile="build/agenda.war" webxml="WebContent/WEB-INF/web.xml">
    <fileset dir="WebContent" />
    <classes dir="build/classes" />
 </war>
</target>
```

Executando o target `empacotar` uma saída possível é:

```
...
empacotar:
      [war] Building war: /workspace/agenda/build/agenda.war
BUILD SUCCESSFUL
Total time: 5 seconds
```

Mais sobre a tarefa na documentação:
(http://ant.apache.org/manual/Tasks/war.html)http://ant.apache.org/manual/Tasks/war.html
(http://ant.apache.org/manual/Tasks/war.html)

Deployar e executar o projeto no servidor

Com o nosso WAR devidamente gerado, podemos subir o mesmo para o nosso servidor e reiniciá-lo afim das mudanças ficarem disponíveis. Vamos utilizar o servlet container Apache Tomcat.

O Tomcat é encontrado no site: (http://tomcat.apache.org/)http://tomcat.apache.org/ (http://tomcat.apache.org/)

Para fazer o deploy teremos dois targets, um para copiar o WAR para a pasta do Tomcat e outro para subir o Tomcat.

O primeiro target será simples, apenas copiará o arquivo WAR para a pasta de deploy do Tomcat. Usamos a tag `copy` que recebe o arquivo sendo copiado `file` e o destino do mesmo todir.

```
<target name="deploy-local" depends="empacotar">
    <copy file="build/agenda.war" todir="/pastaDoTomcat/webapps" />
</target>
```

E executando com ANT: `ant deploy-local`

Imprime na console:

```
...
deploy-local:
     [copy] Copying 1 file to /appservers/apache-tomcat-6.0.32/webapps
...
```

Quando o Tomcat sobe ele vai encontrar este WAR e extrair os arquivos para disponibilizar a aplicação. Isso funciona bem para outros servidores como Jetty e JBoss, o que muda é apenas a pasta de deploy (`todir`).

No caso do Tomcat existe mais uma possibilidade. O Tomcat possui um gerenciador de aplicações (Tomcat Application Manager). Com o gerenciador instalado podemos fazer um deploy pela interface gráfica e também existe uma tarefa própria do ANT para fazer o deploy através dele.

Mais informações sobre esse gerenciador são encontradas na documentação do Tomcat:

(http://tomcat.apache.org/tomcat-6.0-doc/manager-howto.html#Executing_Manager_Commands_With_Ant)http://tomcat.apache.org/tomcat-6.0-doc/manager-howto.html
(http://tomcat.apache.org/tomcat-6.0-doc/manager-howto.html)

O target acima copia o WAR na pasta de `deploy`, mas falta ainda inicializar o Tomcat. Para isso vamos aproveitar a tag `exec` que sabe chamar um executável passado no atributo `executable`. Com ela podemos subir o servidor através do ANT chamando o executável que a instalação do Tomcat oferece:

```
<target name="rodar-tomcat-local">
        <exec executable="/pastaDoTomcat/bin/startup.sh" />
</target>
```

No Linux utiliza-se o arquivo `startup.sh`, no Windows `startup.bat`.

Executando e testando o target com ANT: `ant rodar-tomcat-local`

Uma possível saída no console será:

```
...
rodar-tomcat-local:
     [exec] Using CATALINA_BASE:   /appservers/apache-tomcat-6.0.32
     [exec] Using CATALINA_HOME:   /appservers/apache-tomcat-6.0.32
     [exec] Using CATALINA_TMPDIR: /appservers/apache-tomcat-6.0.32/temp
     [exec] Using JRE_HOME:        /usr/lib/jvm/java-6-sun/jre
     [exec] Using CLASSPATH:       /appservers/apache-tomcat-6.0.32/bin/bootstrap.jar
...
```

Para parar o Tomcat a mesma tag `exec` pode ser usada, agora chamando o executável `shutdown.sh`:

```
<target name="parar-tomcat-local">
       <exec executable="/pastaDoTomcat/bin/shutdown.sh" />
</target>
```

No Windows utiliza-se `shutdown.bat`.

Acesso a qualquer servidor remoto via SCP

Em algum momento queremos disponibilizar nossa aplicação no servidor de integração, homologação ou produção. Queremos que isso funcione confortavelmente pelo ANT.

Como existem tarefas para copiar e executar localmente, o ANT também possui tarefas para acessar um servidor remotamente (`sshexec`) e copiar um arquivo para ele (`scp`). As tarefas do ANT só representam uma "casca" sobre os comandos `ssh` e `scp` comumente usados nos sistemas Unix.

Antes de definir e executar as tarefas é preciso configurar uma biblioteca. O ANT necessita do uso de um JAR especial para executar os comandos ssh ou scp.

A biblioteca se chama Java Secure Channel (jsch) e está disponível em:
(http://www.jcraft.com/jsch/)http://www.jcraft.com/jsch/ (http://www.jcraft.com/jsch/)

Para adicionar o JAR no classpath podemos usar o parâmetro na linha de comando:

```
ant -lib jsch-0.x.xx.jar
```

Ou colocar o JAR na pasta padrão do ANT: `/pastaDoUsuario/.ant/lib` Essa pasta automaticamente faz parte do classpath do ANT.

Com ANT sendo configurado corretamente podemos executar o comando `touch` remotamente que modifica a data do arquivo e assim força um redeploy da aplicação:

```
<target name="touch-war-no-servidor-remoto" >
      <sshexec host="192.168.0.100" username="caelum" password="senhaSecreta"
            command="touch /pastaDeployRemoto/agenda.war" />
</target>
```

Para copiar um arquivo com `scp` para um servidor remoto, usa-se:

```
<target name="deploy-remoto" >
      <scp file="build/agenda.war" todir="usuario@ip-do-servidor:/pastaDeployRemoto"
                        password="senhaSecreta"  trust="true" />
</target>
```

A execução da `target` mostra no console:

```
deploy-remoto:
      [scp] Connecting to ip-do-servidor:22
      [scp] done
```

Geração do projeto para ambientes de homologação e produção

Quando a aplicação cresce, surge a necessidade de deployar ela em ambientes diferentes. Por exemplo, podemos pensar em um servidor de integração que roda através do ANT os testes. Para esse ambientes queremos modificar um pouco a arquivo WAR para, por exemplo, usar um banco de dados em memória ou habilitar um log mais detalhado. O WAR para o ambiente de produção não deve ter essas configurações.

Portanto, o ANT deve gerar WARs diferentes para cada ambientes com apenas pequenas modifcações entre eles. Vamos adaptar o build.xml para ser apto de gerar isso.

Uma tarefa condicional

Uma forma simples de gerar WARs diferentes é criar `target` s condicionais, ou seja, `target` s que só serão executados caso alguma condição seja verdadeira. O elemento `target` possui um atributo `if` que serve justamente para isso:

```
<target name="copiar-recursos-testes" if="ambiente.teste">
    <copy  todir="build/classes">
        <fileset dir="src-teste">
            <include name="hibernate.cfg.xml"/>
            <include name="log4j.xml"/>
        </fileset>
    </copy>
</target>
```

Nesse caso, o target acima só é executado se a variável `ambiente.teste` for verdadeira. Para deixar isso flexível o ANT, assim como Java, tem uma maneira de definir uma propriedade como parâmetro do executável `ant` :

```
ant -Dambiente.teste=true
```

Então o ambiente que chama esse comando define o valor da propriedade. Podemos omitir o parâmetro, que é por padrão `false` .