Diferenças entre Ant e Maven

Em Maven: O Guia Definitivo , escrevi sobre as diferenças entre Maven e Ant na introdução, na seção intitulada "As Diferenças entre Ant e Maven" . Aqui está uma resposta que combina as informações dessa introdução com algumas notas adicionais.

Uma comparação simples

Estou mostrando isso apenas para ilustrar a ideia de que, no nível mais básico, o Maven possui convenções integradas. Aqui está um arquivo de compilação Ant simples:

modelo-php

```
<project name="my-project" default="dist" basedir=".">
    <description>
        simple example build file
    </description>
    <!-- set global properties for this build -->
    <property name="src" location="src/main/java"/>
    <property name="build" location="target/classes"/>
    <property name="dist"  location="target"/>

    <target name="init">
      <!-- Create the time stamp -->
      <tstamp/>
      <!-- Create the build directory structure used by compile -->
      <mkdir dir="${build}"/>
    </target>

    <target name="compile" depends="init"
        description="compile the source " >
      <!-- Compile the java code from ${src} into ${build} -->
      <javac srcdir="${src}" destdir="${build}"/>
    </target>

    <target name="dist" depends="compile"
        description="generate the distribution" >
      <!-- Create the distribution directory -->
      <mkdir dir="${dist}/lib"/>

      <!-- Put everything in ${build} into the MyProject-${DSTAMP}.jar file
-->
      <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
    </target>

    <target name="clean"
        description="clean up" >
      <!-- Delete the ${build} and ${dist} directory trees -->
      <delete dir="${build}"/>
      <delete dir="${dist}"/>
```

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

Neste exemplo simples de Ant, você pode ver como precisa dizer ao Ant exatamente o que fazer. Existe um objetivo de compilação que inclui a tarefa `javac`, responsável por compilar o código-fonte no diretório `src/main/java` para o diretório `target/classes`. Você precisa informar ao Ant exatamente onde está o seu código-fonte, onde deseja que o bytecode resultante seja armazenado e como empacotar tudo isso em um arquivo JAR. Embora existam alguns desenvolvimentos recentes que ajudam a tornar o Ant menos procedural, a experiência de um desenvolvedor com Ant ainda se concentra na codificação de uma linguagem procedural escrita em XML.

Compare o exemplo anterior com Ant com um exemplo com Maven. No Maven, para criar um arquivo JAR a partir de um código-fonte Java, basta criar um arquivo pom.xml simples, colocar o código-fonte em ${basedir}/src/main/java e executar mvn install na linha de comando. O exemplo de pom.xml do Maven que alcança os mesmos resultados.

modelo-php

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sonatype.mavenbook</groupId>
  <artifactId>my-project</artifactId>
  <version>1.0</version>
</project>
```

Isso é tudo que você precisa no seu arquivo pom.xml. Executar `mvn install` na linha de comando processará os recursos, compilará o código-fonte, executará os testes unitários, criará um arquivo JAR e instalará o JAR em um repositório local para reutilização em outros projetos. Sem modificações, você pode executar `mvn site` e encontrará um arquivo `index.html` em `target/site` que contém links para o JavaDoc e alguns relatórios sobre o seu código-fonte.

É verdade que este é o projeto de exemplo mais simples possível. Um projeto que contém apenas código-fonte e gera um arquivo JAR. Um projeto que segue as convenções do Maven e não requer dependências ou personalizações. Se quiséssemos começar a personalizar o comportamento, nosso arquivo pom.xml cresceria de tamanho, e nos maiores projetos é possível encontrar coleções de arquivos POM do Maven muito complexos, contendo uma grande quantidade de personalizações de plugins e declarações de dependências. Mas, mesmo quando os arquivos POM do seu projeto se tornam mais substanciais, eles contêm um tipo de informação completamente diferente do arquivo de build de um projeto de tamanho similar usando Ant. Os arquivos POM do Maven contêm declarações como: "Este é um projeto JAR" e "O código-fonte está em src/main/java". Os arquivos de compilação do Ant contêm instruções explícitas: "Este é o projeto", "O código-fonte está em `src/main/java`", "Execute `javac`neste diretório", "Coloque os resultados em `target/classses`", "Crie um JAR a partir de ....", etc. Enquanto o Ant precisava ser explícito sobre o processo, o Maven tinha algo "embutido" que simplesmente sabia onde o código-fonte estava e como ele deveria ser processado.

Comparação de alto nível

Quais as diferenças entre Ant e Maven neste exemplo? Ant...

- Não possui convenções formais como uma estrutura de diretórios de projeto comum; você precisa informar ao Ant exatamente onde encontrar o código-fonte e onde colocar a saída. Convenções informais surgiram ao longo do tempo, mas não foram codificadas no produto.
- É procedural; você precisa dizer ao Ant exatamente o que fazer e quando fazer. Você teve que dizer para ele compilar, depois copiar e, por fim, comprimir.
- Não possui um ciclo de vida; você precisa definir metas e dependências entre elas. Além disso, você precisa associar manualmente uma sequência de tarefas a cada meta.

Onde Maven...

- Por seguir as convenções, o programa já sabia onde estava seu código-fonte porque você as seguiu. Ele colocou o bytecode em target/classes e gerou um arquivo JAR em target.
- É declarativo. Tudo o que você precisa fazer é criar um arquivo pom.xml e colocar seu código-fonte no diretório padrão. O Maven cuida do resto.
- possui um ciclo de vida, que você invocou ao executar o comando `mvn install`. Esse comando instruiu o Maven a executar uma série de etapas sequenciais até atingir o ciclo de vida. Como efeito colateral dessa jornada pelo ciclo de vida, o Maven executou uma série de objetivos de plugin padrão que realizaram tarefas como compilar e criar um arquivo JAR.

E quanto à Ivy?

Certo, então alguém como Steve Loughran vai ler essa comparação e reclamar. Ele vai falar sobre como a resposta ignora completamente algo chamado Ivy e o fato de que o Ant pode reutilizar a lógica de compilação nas versões mais recentes. Isso é verdade. Se você tiver um grupo de pessoas inteligentes usando Ant + antlibs + Ivy, você terá uma compilação bem projetada e funcional. Mesmo estando bastante convencido de que o Maven faz sentido, eu usaria Ant + Ivy com prazer em uma equipe de projeto que tivesse um engenheiro de compilação muito competente. Dito isso, acho que você acabará perdendo vários plugins valiosos, como o plugin do Jetty, e que acabará fazendo um monte de trabalho desnecessário ao longo do tempo.

Mais importante do que Maven vs. Ant

1. Você utiliza um gerenciador de repositórios para controlar os artefatos de software? Sugiro que baixe o Nexus . Você pode usar o Nexus como proxy para repositórios remotos e para fornecer um local para sua equipe implantar artefatos internos.
2. Você possui uma modularização adequada dos componentes de software. Um único componente monolítico grande raramente escala ao longo do tempo. À medida que seu projeto se desenvolve, você precisará do conceito de módulos e submódulos. O Maven se adapta muito bem a essa abordagem.
3. Você adota algumas convenções para sua compilação. Mesmo que use o Ant, deve se esforçar para adotar alguma convenção que seja consistente com outros projetos. Quando um projeto usa o Maven, significa que qualquer pessoa familiarizada com o Maven pode pegar a compilação e começar a executá-la sem precisar mexer na configuração apenas para descobrir como fazer o código compilar.

Maven é um framework, Ant é um conjunto de ferramentas.

Maven é um carro de rua pré-montado, enquanto Ant é um conjunto de peças de carro. Com Ant, você precisa montar seu próprio carro, mas pelo menos, se precisar dirigir em terrenos acidentados, poderá construir o tipo certo de veículo.

Em outras palavras, o Maven é um framework, enquanto o Ant é um conjunto de ferramentas. Se você se contenta em trabalhar dentro dos limites do framework, o Maven funcionará perfeitamente. O problema para mim era que eu constantemente esbarrava nos limites do framework e ele não me permitia sair.

Verbosidade XML

O Tobrien entende muito de Maven e acho que ele fez uma comparação muito boa e honesta entre os dois produtos. Ele comparou um arquivo pom.xml simples do Maven com um arquivo de build simples do Ant e mencionou como os projetos Maven podem se tornar mais complexos. Acho que vale a pena dar uma olhada na comparação de alguns arquivos que você provavelmente encontrará em um projeto simples do mundo real. Os arquivos abaixo representam um único módulo em um build com múltiplos módulos.

Primeiro, o arquivo Maven:

modelo-php

```
<project
      xmlns="http://maven.apache.org/POM/4.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-4_0_0.xsd">

      <parent>
            <groupId>com.mycompany</groupId>
            <artifactId>app-parent</artifactId>
            <version>1.0</version>
      </parent>

      <modelVersion>4.0.0</modelVersion>
      <artifactId>persist</artifactId>
      <name>Persistence Layer</name>

      <dependencies>

            <dependency>
                  <groupId>com.mycompany</groupId>
                  <artifactId>common</artifactId>
                  <scope>compile</scope>
                  <version>${project.version}</version>
            </dependency>

            <dependency>
                  <groupId>com.mycompany</groupId>
                  <artifactId>domain</artifactId>
```

```
        <scope>provided</scope>
        <version>${project.version}</version>
    </dependency>

    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate</artifactId>
        <version>${hibernate.version}</version>
        <scope>provided</scope>
    </dependency>

    <dependency>
        <groupId>commons-lang</groupId>
        <artifactId>commons-lang</artifactId>
        <version>${commons-lang.version}</version>
        <scope>provided</scope>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring</artifactId>
        <version>${spring.version}</version>
        <scope>provided</scope>
    </dependency>

    <dependency>
        <groupId>org.dbunit</groupId>
        <artifactId>dbunit</artifactId>
        <version>2.2.3</version>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>org.testng</groupId>
        <artifactId>testng</artifactId>
        <version>${testng.version}</version>
        <scope>test</scope>
        <classifier>jdk15</classifier>
    </dependency>

    <dependency>
        <groupId>commons-dbcp</groupId>
        <artifactId>commons-dbcp</artifactId>
        <version>${commons-dbcp.version}</version>
        <scope>test</scope>
    </dependency>
```

```
        <dependency>
            <groupId>com.oracle</groupId>
            <artifactId>ojdbc</artifactId>
            <version>${oracle-jdbc.version}</version>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>org.easymock</groupId>
            <artifactId>easymock</artifactId>
            <version>${easymock.version}</version>
            <scope>test</scope>
        </dependency>

    </dependencies>

</project>
```

E o arquivo Ant equivalente:

modelo-php

```
<project name="persist" >

    <import file="../build/common-build.xml" />


    <path id="compile.classpath.main">
        <pathelement location="${common.jar}" />
        <pathelement location="${domain.jar}" />
        <pathelement location="${hibernate.jar}" />
        <pathelement location="${commons-lang.jar}" />
        <pathelement location="${spring.jar}" />
    </path>


    <path id="compile.classpath.test">
        <pathelement location="${classes.dir.main}" />
        <pathelement location="${testng.jar}" />
        <pathelement location="${dbunit.jar}" />
        <pathelement location="${easymock.jar}" />
        <pathelement location="${commons-dbcp.jar}" />
        <pathelement location="${oracle-jdbc.jar}" />
        <path refid="compile.classpath.main" />
    </path>


    <path id="runtime.classpath.test">
        <pathelement location="${classes.dir.test}" />
        <path refid="compile.classpath.test" />
```

```
    </path>


</project>
```

Tobrien usou seu exemplo para mostrar que o Maven possui convenções integradas, mas isso não significa necessariamente que você escreverá menos XML. Descobri que o oposto é verdadeiro. O arquivo pom.xml é três vezes maior que o build.xml, e isso sem nos desviarmos das convenções. Na verdade, meu exemplo com Maven é mostrado sem as 54 linhas extras necessárias para configurar plugins. Esse pom.xml é para um projeto simples. O XML começa a crescer significativamente quando você adiciona requisitos extras, o que não é incomum em muitos projetos.

Mas você precisa dizer ao Ant o que fazer.

Meu exemplo com Ant acima não está completo, é claro. Ainda precisamos definir os alvos usados para limpar, compilar, testar etc. Estes são definidos em um arquivo de build comum que é importado por todos os módulos em um projeto com múltiplos módulos. O que me leva ao ponto de que tudo isso precisa ser escrito explicitamente no Ant, enquanto no Maven é declarativo.

É verdade, economizaria tempo se eu não precisasse escrever explicitamente esses alvos do Ant. Mas quanto tempo? O arquivo de build comum que uso atualmente foi escrito há 5 anos, com apenas pequenos ajustes desde então. Depois de dois anos de experiência com o Maven, tirei o antigo arquivo de build do Ant do armário, tirei a poeira e o coloquei de volta em funcionamento. Para mim, o custo de ter que dizer explicitamente ao Ant o que fazer se resume a menos de uma semana ao longo de 5 anos.

Complexidade

A próxima grande diferença que gostaria de mencionar é a da complexidade e o impacto que ela tem no mundo real. O Maven foi criado com a intenção de reduzir a carga de trabalho dos desenvolvedores responsáveis pela criação e gerenciamento de processos de compilação. Para isso, ele precisa ser complexo. Infelizmente, essa complexidade tende a anular o objetivo pretendido.

Em comparação com o Ant, o responsável pela compilação em um projeto Maven gastará mais tempo:

- Leitura da documentação: Existe muito mais documentação sobre o Maven, porque há muito mais para aprender.
- Capacitar os membros da equipe: Eles acham mais fácil perguntar a alguém que sabe do que tentar encontrar as respostas por conta própria.
- Solução de problemas na compilação: O Maven é menos confiável que o Ant, especialmente com plugins não essenciais. Além disso, as compilações do Maven não são repetíveis. Se você depende de uma versão SNAPSHOT de um plugin, o que é muito provável, sua compilação pode falhar sem que você tenha alterado nada.
- Escrevendo plugins do Maven: Os plugins geralmente são escritos com uma tarefa específica em mente, como criar um pacote webstart, o que torna mais difícil reutilizá-los para outras tarefas ou combiná-los para atingir um objetivo.

Portanto, você pode precisar escrever um plugin próprio para contornar as lacunas no conjunto de plugins existente.

Em contraste:

- A documentação do Ant é concisa, abrangente e está toda reunida em um só lugar.
- Ant é simples. Um novo desenvolvedor que esteja tentando aprender Ant só precisa entender alguns conceitos básicos (alvos, tarefas, dependências, propriedades) para conseguir deduzir o restante do que precisa saber.
- O Ant é confiável. Não houve muitos lançamentos do Ant nos últimos anos porque ele já funciona bem.
- As compilações do Ant são repetíveis porque geralmente são criadas sem quaisquer dependências externas, como repositórios online, plugins experimentais de terceiros, etc.
- O Ant é abrangente. Por ser um conjunto de ferramentas, você pode combinar as ferramentas para executar praticamente qualquer tarefa que desejar. Se precisar escrever uma tarefa personalizada, é muito simples fazê-lo.

Familiaridade

Outra diferença reside na familiaridade. Novos desenvolvedores sempre precisam de tempo para se adaptar. A familiaridade com produtos existentes ajuda nesse sentido, e os defensores do Maven afirmam, com razão, que essa é uma de suas vantagens. Claro, a flexibilidade do Ant permite que você crie as convenções que desejar. A convenção que eu utilizo é colocar meus arquivos de código-fonte em um diretório chamado src/main/java. Minhas classes compiladas vão para um diretório chamado target/classes. Parece familiar, não é?

Gosto da estrutura de diretórios usada pelo Maven. Acho que faz sentido. Assim como seu ciclo de vida de compilação. Por isso, uso as mesmas convenções em minhas compilações com Ant. Não apenas porque faz sentido, mas porque será familiar para qualquer pessoa que já tenha usado o Maven antes.