第二章 安装

Maven 解压到安装目录后，需要设置两个环境变量——PATH

和M\_HOME。

M\_HOME=c:\Program Files\maven-2.0.9

PATH=%PATH%;%M\_HOME%\bin

测试maven安装是否成功

命令行:

mvn –v

第三章/第四章 例子

例子:为同目录下maven\_test文件夹中的simple

新建文件夹

修改名称为maventest

创建工程

命令行:(“-”之前需要有空格)

mvn archetype:create -DgroupId=org.sonatype.mavenbook.ch -DartifactId=simple -DpackageName=org.sonatype.mavenbook -Dversion=1.0

说明:

DgroupId 路径名

DartifactId 应用名

DpackageName 包路径名

Dversion 版本

打包该工程:

命令行:

mvn install(在包含pom.xml 的目录下运行)

mvn clean install -Dmaven.test.skip=true

mvn compile 编译工程

测试jar包可以使用

命令行:

java -cp target/simple-1.0-SNAPSHOT.jar org.sonatype.mavenbook.App

demo搞定！

pom.xml

pom.xml文件是Maven项目中最基础的POM，最开始的几个元素

——groupId ， artifactId, packaging, version—— 是Maven 的坐标，它们唯一标识了一个项目。

命令行:

mvn help:effective-pom(simple项目的根目录运行)

说明:

一旦你运行了此命令，你应该能看到一个大得多的POM，它暴露了Maven 的默

认设置

测试已经发布站点:

命令行:

mvn site

mvn依赖查找

<http://mvnrepository.com/>

查看项目所依赖的项目

命令行:

mvn dependency:resolve

mvn测试

命令行:

mvn test

mvn test -Dmaven.test.failure.ignore=true(表示忽略测试错误)

第五章 web应用

web工程:

例子:为同目录下maven\_test文件夹中的simple-webapp

命令行:

mvn archetype:create -DgroupId=org.sonatype.mavenbook.web -DartifactId=simple-webapp -DpackageName=org.sonatype.mavenbook -DarchetypeArtifactId=maven-archetype-webapp

说明:

定义为web工程

在pom文件中添加

<project>

[...]

<build>

<finalName>simple-webapp</finalName>

<plugins>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>tomcat-maven-plugin</artifactId>

<version>1.1</version>

</plugin>

</plugins>

</build>

[...]

</project>

说明:

此插件为tomcat插件,添加后即可使用tomcat:run命令

命令行:

mvn tomcat:run 使用maven的tomcat插件启动web工程

第七章 多模块企业级项目

例子:为同目录下maven\_test文件夹中的simple-parent

simple-parent项目有一个pom.xml，其中引用了五个子模块:simple-command，

simple-model，simple-weather，simple-persist，和simple-webapp

simple-parent的pom.xml文件中:

<project>

[...]

<groupId>org.sonatype.mavenbook.ch07</groupId>

<artifactId>simple-parent</artifactId>

<packaging>pom</packaging>

<version>1.0</version>

<name>Chapter 7 Simple Parent Project</name>

<modules>

<module>simple-command</module>

<module>simple-model</module>

<module>simple-weather</module>

<module>simple-persist</module>

<module>simple-webapp</module>

</modules>

[...]

</project>

说明:

Packaging中设置为pom表示该项目没有自己的代码,而是将其他的子项目整合起来成为一个父项目的一个文件夹而已

simple-model的pom.xml文件中:

<project>

[...]

<parent>

<groupId>org.sonatype.mavenbook.ch07</groupId>

<artifactId>simple-parent</artifactId>

<version>1.0</version>

</parent>

<artifactId>simple-model</artifactId>

<packaging>jar</packaging>

<name>Simple Object Model</name>

[...]

</project>

说明:

定义父项目位置 groupId与version跟随父项目

simple-weather的pom.xml文件中:

<project>

[...]

<parent>

<groupId>org.sonatype.mavenbook.ch07</groupId>

<artifactId>simple-parent</artifactId>

<version>1.0</version>

</parent>

<artifactId>simple-weather</artifactId>

<packaging>jar</packaging>

<name>Simple Weather API</name>

<dependencies>

<dependency>

<groupId>${project.groupId}</groupId>

<artifactId>simple-model</artifactId>

<version>${project.version}</version>

</dependency>

[...]

</dependencies>

[...]

</project>

说明:

定义父项目位置 在使用到父项目下同级子项目时 groupId与version使用当前项目的数值 而当前项目的groupId与version默认使用的父类的，从而保证兄弟项目之间的依赖一致

所需的配置文件写在src/main/resources目录下 文件打包时即可放classpath下 在打成jar时放入根目录下 而打成war包时放入classes目录下

simple-persist

simple-webapp

命令行:

mvn hibernate3:hbm2ddl 让hibernate自己自建所需数据库

去父目录夹下运行:

mvn clean install

可以将下面的5个子项目都install

第八章 优化

子项目中重复定义和使用spring或者hibernate，则可在simple-parent的POM中使用dependencyManagement来统一定义

simple-parent的POM文件中:

<project>

[...]

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring</artifactId>

<version>2.0.7</version>

</dependency>

<dependency>

<groupId>org.apache.velocity</groupId>

<artifactId>velocity</artifactId>

<version>1.5</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-annotations</artifactId>

<version>3.3.0.ga</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-commons-annotations</artifactId>

<version>3.3.0.ga</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate</artifactId>

<version>3.2.5.ga</version>

<exclusions>

<exclusion>

<groupId>javax.transaction</groupId>

<artifactId>jta</artifactId>

</exclusion>

</exclusions>

</dependency>

</dependencies>

</dependencyManagement>

[...]

</project>

说明:

dependencyManagement即在父项目中定义了公共依赖，子项目中如果再次定义依赖会覆盖父项目中的定义

而hibernate-annotations与hibernate-commons-annotations版本的不一致，可以创建一个称为hibernate-annotations-version 的属性来统一。

<project>

[...]

<properties>

<hibernate.annotations.version>3.3.0.ga</hibernate.annotations.version>

</properties>

<dependencyManagement>

[...]

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-annotations</artifactId>

<version>${hibernate.annotations.version}</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-commons-annotations</artifactId>

<version>${hibernate.annotations.version}</version>

</dependency>

[...]

</dependencyManagement>

[...]

</project>

说明:

依靠自定义的version，就可以使得hibernate的几个依赖统一版本了

兄弟项目依赖问题:

第一种方案是像其他依赖一样将他们移到父项目的dependencyManagement中，在最顶层的父项目中定义所有兄弟项目的版本

第二种方案是如simple-weather的pom.xml文件中:

<project>

[...]

<dependencies>

<dependency>

<groupId>${project.groupId}</groupId>

<artifactId>simple-model</artifactId>

<version>${project.version}</version>

</dependency>

[...]

</dependencies>

[...]

</project>

说明:

兄弟间的依赖可以使用maven内置的${project.groupId}与${project.version}属性

插件优化也和依赖优化类似

将插件定义在pluginManagement中

如下:

<project>

[...]

<build>

<pluginManagement>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>1.5</source>

<target>1.5</target>

</configuration>

</plugin>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>hibernate3-maven-plugin</artifactId>

<version>2.1</version>

<configuration>

<components>

<component>

<name>hbm2ddl</name>

<implementation>annotationconfiguration</implementation>

</component>

</components>

</configuration>

<dependencies>

<dependency>

<groupId>hsqldb</groupId>

<artifactId>hsqldb</artifactId>

<version>${hsqldb.version}</version>

</dependency>

</dependencies>

</plugin>

</plugins>

</pluginManagement>

</build>

[...]

</project>

使用Maven Dependency优化

命令行:

mvn dependency:analyze

结果:

[INFO] [dependency:analyze]

[WARNING] Used undeclared dependencies found:

[WARNING] javax.persistence:persistence-api:jar:1.0:compile

说明:

运行dependency:analyze分析这个项目，查看是否有直接依赖，或者一些引用了但不是直接声明的依赖。结果中dependency 插件指出有一个对于javax.persistence:persistence-api的”使用了但为未声明的依赖”。为了进一步调查，到simple-model目录下运行dependency:tree命令

命令行:

mvn dependency:tree

结果:

[INFO] [dependency:tree]

[INFO] org.sonatype.mavenbook.ch08:simple-model:jar:1.0

[INFO] +- org.hibernate:hibernate-annotations:jar:3.3.0.ga:compile

[INFO] | \- javax.persistence:persistence-api:jar:1.0:compile

说明:

该命令会列出项目中所有的直接依赖以及传递性依赖。

由此可以看出javax.persistence:persistence-api依赖是hibernate传递过来的。简单的修复手段是添加对这个依赖的直接引用，将依赖版本放到simple-parent 的dependencyManagement片段中