The steps to use maven in the new or existing project

1. mvn archetype:create –DgroupId=xxxxxx.xx.x.xx –DartifaceId=xxx

2. Modify the generated pom file. Add the common information like project name, developers and license etc.

3. If you are working on the existing project, just copy the pom.xml file into the root folder of the project. The pom.xml can be the good start point.

4. Add dependencies. We don’t need to manually input the dependent groupId,artifactId. Website [www.mvnrepository.com](http://www.mvnrepository.com/) can help us. We just need copy paste the dependency xml file into our pom file.

5. If your project involves a new plugin which is not the core plugin of maven, you can use mvn help:describe –Dplugin=pluginName –Dfull to find the exact goals the plugin provided.

6. mvn dependency:resolve shows all dependent jar files. Mvn dependency:tree shows more clear picture of the rely on relation.

7. mvn phase/plugin:goal –X → open debug mode to run mvn

8. once you have testcase, sometimes you like to ignore the failure of testcase by mvn test –Dmaven.test.failure.ignore=true. Or sometimes you like skip the testcase running, you can mvn install –Dmaven.test.skip=true

9. assemble everything into one. mvn assembly:assembly

create a simple web-application template from maven

. mvn archetype:create –DgroupId=xxx.xxx.xxx –DartifactId=xxx –Darchetypes=maven-archetype-webapp

. another simple way to create maven project is install maven eclipse plugin (m2eclipse). So you don’t need to remember the parameters in command line.

Create multiple module application

First of all, create a parent module. The packaging type is pom and define the sub-modules in there. Then maven chooses the parent pom. It will search for the pom in sub-modules and push into pom reaction stack. The module depended on other modules always pushed down. Especially in multiple module application, we’d better run ‘mvn clean’ to clean the artifacts built before. It means we’d better clean the environment before you start the real build.

## Optimize the pom configuration of multiple module application

In most of complicated maven projects, we usually manage the versions of dependent libraries or sub-modules in top POM. Therefore, the parent pom becomes the major place to control the version numbers.

### 1. optimize dependencies

1.1 find the libraries which shared by multiple modules and move them from each sub-module to parent module. These shared libraries should be defined in dependencyManagement section. Bear in mind, the shared libraries must be only defined in parent module. Each sub-module should not include them. Otherwise the libraries in sub-module will override the libraries defined in parent module.



1.2 Some of libraries always work as group. They should have the same version number. So we can declare a global version number in parent module and shared by the libraries in the group.



1.3 Find the best way to define the version relations among the sub-modules. We can define all the version numbers of sub-modules in the parent module. Or define the version numbers of the sub-modules in the module which refers the sub-modules.

### 2. Optimize plugin

2.1 for shared plugins, we can define them in the section of pluginManagement in parent pom.

Jan 20th , 2014

Optimize maven project

dependency analyze

mvn dependency:analyze

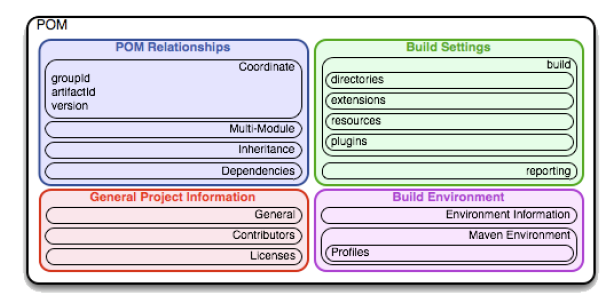
the best practice of maven dependency declaration is to explicitly declare the libraries when the libraries are applied in your code. mvn dependency:analyze able to help you find the directly references in your code. if you have the detail of direct or transit dependency, you can use mvn dependency:tree.

mvn dependency:analyze and dependency:tree

they can help you point out the problems in pom file. they can figure out what libraries should be explicitly declared and what libraries declared but never used. be careful when you see the second problem, you might not be able to simply remove them from the POM file because they are only used by a special phase.

Jan 21th 2014

understand the structure of pom is very important. it can give us the overview.

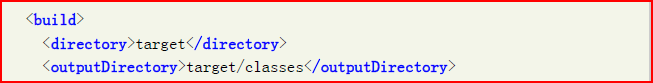




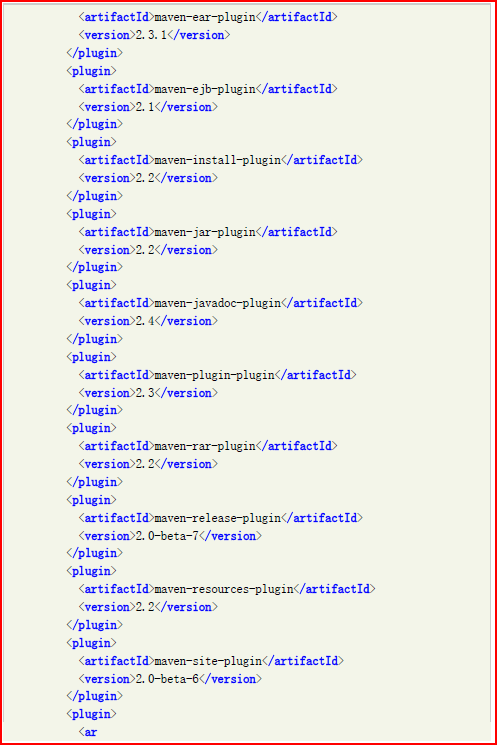
super pom which can be found from maven library













Jan 25th, 2014

## .properties in maven

there are 5 types of implicit properties maven can directly use.

1. set in env. ${env.PATH} means the operating system path variable.

2. we can directly use the the tagValue by following the orders of the tag in pom file. eg.

<project>

<groupId>com.aeroinfo.mdm</groupId>

….

</project>

then at other places, if you like to refer to the groupId, you can apply it as ${project.groupId}

3. the properties defined in settings.xml file eg.

${settings.offline} settings.xml file is located at .m2/setting.xml

4. all the properties if you can access from java.lang.System.getProperties(), you can directly use them in pom. eg. ${os.name}, ${java.home}

5. directly define properties in pom file, ${foo}

<project>

<properties>

<foo>bar</foo>

</properties>

</project>

## Dependency management in maven

1. dependent scope

. compile scope

it is default scope. it will be set in the classpath and packaged into the war or jar file.

. provided scope

the class is only used in compiling. it will be set to the classpath during compile. it will not be involved in runtime or packaging. it is not transit dependency.

.runtime scope

it will not be involved in compiling. but it will be used in runtime. so it will be packed into the final artifact.

. test scope

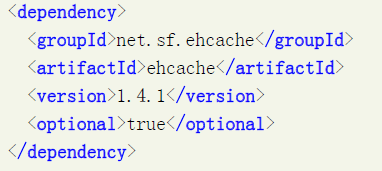
it is only involved in the test phase. it will not be packed into artifact.

. system scope

it is similar to provided scope. but we have to manually define the class into the environment classpath to make the compile work. provided scope dependency will be automatically added into classpath by maven. maven will not look for system scope dependency from central depository.

2. optional dependency

if we don’t like the artifact be transit, we can set the artifact as optional.



bear in mind, in the real env, we have rare case like optional dependency.

3. version management of dependency

the range declaration sign can be used in maven version control. if we like the version 1.0.0<version<=4.0.1, we can write as <version>(1.0.0,4.0.1]. for some special cases,

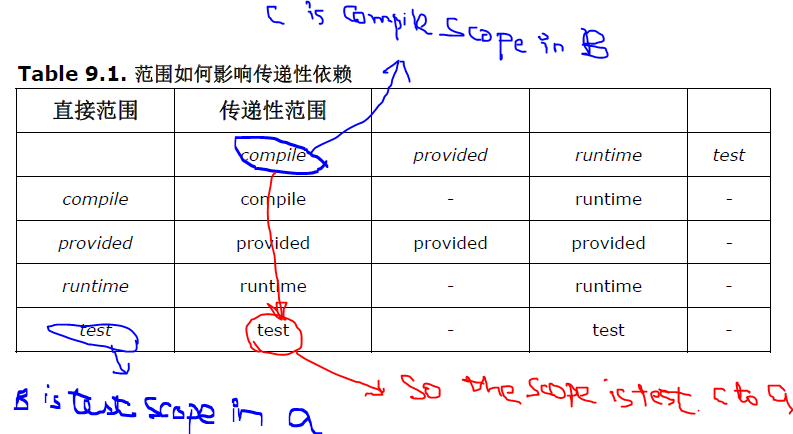
like (,4.0.1] etc.



if we set version [1.2.0], it means we can only use 1.2.0 version. in the most of case, we never set the version like this. be aware of the difference of <version>1.2.0</version> and <version>[1.2.0]</version>. the first scenario, maven will automatically adjust the version number if found conflict among the artifacts. but the second case, it forces maven has to use 1.2.0 version and nothing else. so the conflict has to be resolved by our own. the best practise of version control is to define the version <dependencyManagement> in parent pom.

4. dependency transit

dependency transit is one of the most amazing features of maven. how does maven transit the dependent artifacts? how maven handles the dependency scope for the dependent artifacts? let’s say the project-a depends on project-b by test scope. project-b depends on project-c by compile scope, what is the dependency scope of project-c in project-a when maven automatically transits project-c into project-a? we can find the dependency scope after auto transit. so project-c is test scope in project-a.



5. exclusive the dependency

in many third party artifacts, their dependencies are not free or not existed in maven central repository. so we have to exclude them or replace them by including other artifacts. hibernate is the example, hibernate depends on javax.transaction.JTA which is in sun JTA Api which is not free download. so we have replace it.



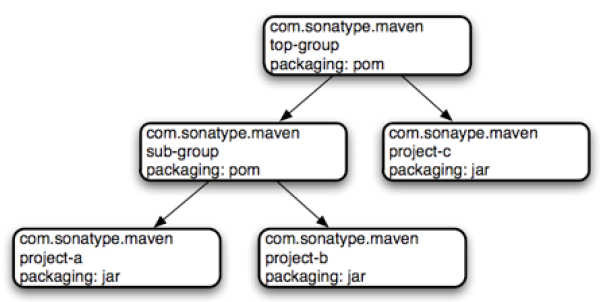
6. dependency version management

define the version number in parent POM under <dependencyManagemane>. in child pom, we can define the dependency but without the version number. therefore, maven will automatically search upwards till to find the dependencyManagement declaration. if we define the version number in child, it will override the version which defined in <dependencyManagement>.

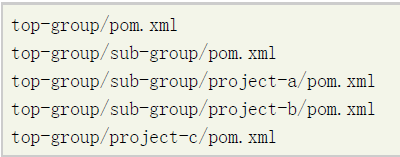


## multiple module project management

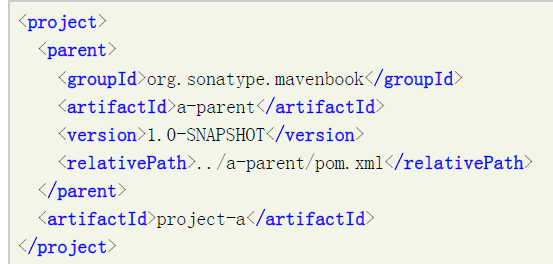
we can define our own way to manage multiple modules. it completely depends on the relations of those modules. for example,



the folder structure looks like



usually the parent pom is at the parent folder of the child pom ie. ../subprj. maven by default searching for parent pom from parent folder or local repository. but maven also allows us to define the parent pom by <relativePath>



## Maven best practice

1. define pom for the special and often used dependencies

we can define a pom project which contains the dependencies. eg. the libraries of hibernate or spring framework can be organized into one pom file. in the future, we can just depend on the the pom file instead of depending on the hibernate or spring libraries. it makes the often used libraries easier to use.





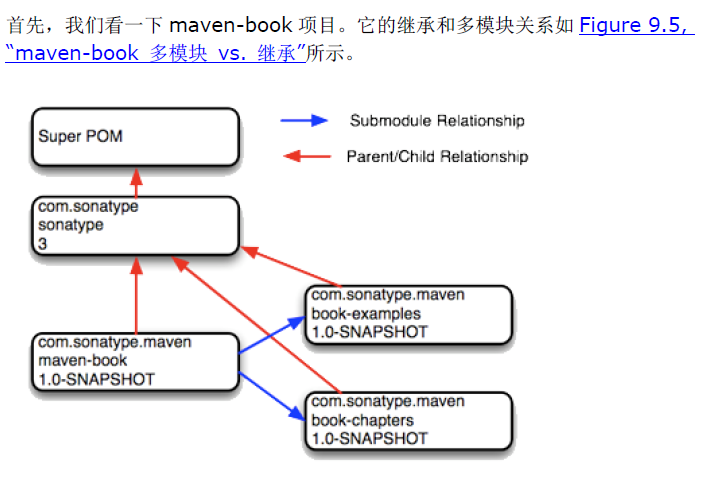
the above persistent-dep project can be install to the local repository. it can be used by other project as the dependency.



this persistence-deps can be shared by all the projects in the company. so all the projects can share the same experience. it is also the best practice to shorten the length of the pom file.

2. inheritance and sub-module

it is difference between the following two cases. the first case is that the pom of a project A contains the <parent> tag. but the modules of the parent don’t include the project A. the other scenario is that the parent contains the sub-module. the first case means the module will inherit all the pom definitions from the parent. the second case means the parent manages the module. the second case is called multiple module project. we usually define it from top to bottom. the first case just build the association. the purpose of the first case is to inherit the pom parameter from the parent. the second is to manage the sub-modules from the parent.



3. profiles

the purpose of profile is to make the artifact creation more portable. the best transportable system is able to build on any environment without any change. usually the open source projects have the best portability. profile setting is optional.

we have 4 approaches to configure profiles.

a. define directly in pom file.

b. define an independent profiles.xml file which is located in the same folder of the pom file.

c. define in the ~/.m2/settings.xml (only impact the current local user)

d. define in the %M2\_HOME%/conf/settings.xml file (impact all the users who use the maven tool).

command mvn help:active-profiles is able to help you where the profile defined.

multiple approaches to activate the profiles.

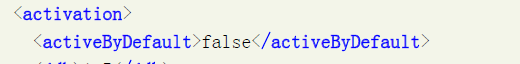
a. mvn install -PprofileId to activate the specific profile.

b. by env. the following example shows the profile will be picked up when you run under jdk6





c. through the configuration



the secured information should be defined in the user local level of settings.xml. if we have the company level remote repository server, you like to all the maven points to there. we can change the settings.xml file in maven package and ask all the developers to use this version of maven package. then everyone no need to change the pom or settings.xml file.

best practices:

<http://stackoverflow.com/questions/921599/declare-dependency-in-dependencymanagement-section-even-if-dependency-not-used>

<http://motlin.com/2011/maven-best-practices/>