**MAVEN**

* Build is two step process --> 1st compilation, 2nd assembling
* compilation : raw source code -> object files i.e class files
* assembly : creating jar/war/ear files from object files
* converting .java files into .class files in java we call it as compilation

then we need to assemble all object files so that we have final product(output)

for java assembly file’s extension would be (jar,war,ear)

* jar --> (java archive)
* war --> creating for web application (web archive )
* ear --> to provide license for web application (enterprise archive)

build management is a process of compiling and assembling the code

**ADVANTAGES:**

* repetitive tasks can be eliminated
* we can track history
* old tool of java was ANT (open source)
* maven is java based tool
* maven takes care of entire project --> it divides project into several objects so that functionality would be easy

**maven follows set of standard:**

* it provides lifecycle
* maven takes care of dependencies --> java code sometimes need other dependencies which can be handled by maven
* maven called as project management tool
* maven is not just a build tool it is super set of various features like dependencies management tool
* every project have build confg files which contains info about what to do ,how to do and all complete details of that project

that confg file we called it as "pom.xml"

* maven needs some plugin to perform some actions like compilation, creating delivery or deleting files etc.
* in maven executable is mvn just like in git we have git
* instruction we passing in maven we call is goal
* so we are giving goal which tell maven what maven has to do

mvn -> goals ->plugins -> task (jar) jar files are nothing but maven plugins

if particular plugin is not present locally then it downloads plugin from apache maven remote repository

**MAVEN build follows a lifecycle:**

**default lifecycle of maven:**

- generate-source/generate-resource (list )

- compile

- test

- package (assembling(creating deliverable war files ))

- install (copying deliverable from other project) --> suppose you have two projects (A and B)--> for both project there are two different delivery files like for A it would be A.jar and for B it would be B.jar --> suppose for creating B.jar file the project might need A.jar file then what maven does it search for A.jar locally but its not available then maven goes to remote repo but again its not available there also --> so A.jar is organization based or project based file its not available locally as well as remote --> so to use A.jar for creating B.jar maven uses install

- deploy

# there also clean and site not part of lifecycle

clean delete runtime files

maven follows directory layout

workspace

===========

src/main/ --> under this path there should be all java files present

/java --> all java files present

/resources --> all dependencies or library present

/confg -->

src/test/ --> all test files present here which tests all java files present in src/main

(these files automatically gets generated)

ex.

src/main

/java/1.java

/resources

src/test

/java/1\_test.java -->which tests 1.java file in /src/main/java folder

/resources

in the same way entire project will be designed

for java project you will see /src/main

pom.xml

Day2

Maven uniquely identifies a project using GAV:

* group id: this will let us know organization for which we are doing this project
* artifact id : name of the project
* version : to identify releases/version of project we use version which consists of 3 strings 🡪

format{major, minor, maintenance}

GAV syntax= groupid:artifactid:version

* Install MAVEN

MAVEN Installation

======================

prerequisite is install JAVA first

1) http://maven.apache.org --> click Download Button --> click Get Maven --> download required zip file (apache-maven-3.8.1.zip)

2) extract this in C drive

3) set env variable --> this pc -->right click --> properties --> advance setting --> env variables --> add new M2\_MAVEN and give value of folder where maven extracted (C:\Users\apache-maven-3.8.1)

4) set another variable --> select path and edit and add new in existing --> C:\Users\apache-maven-3.8.1\bin

5) check in cmd mvn --version if you do not get output restart pc

SET FOLLOWING ENV VARIABLES :

* JAVA\_HOME 🡪 which points to path where java is installed

if you type java -version --> output 1.7.0\_79 -->1=major release,7=minor release,0=path ,\_79=build

# echo %JAVA\_HOME%

environment variable is nothing but the value we are storing at particular location to re-use it.

make sure you set environment variable for maven

* M2\_HOME 🡪(MAVEN extracted path)

for windows --> this pc --> properties --> adv system setting --> advance --> env variable --> add new --> value as M2\_HOME and path will be of maven dir

* M2 🡪 env variable which should be pointing to bin folder in maven folder
* PATH=%PATH%;%M2%
* Go to directory where you want to create template for your project. Open cmd to this path and run following command

mvn archetype:generate

MAVEN Repository

1) Dependencies are downloaded from repository

- via http

2) Downloaded dependencies are cached in local repository

- usually found on ${user.home}/.m2/repository

3) Repository follows a simple dir structure

- {Grpid}/{artifactid}/{version}/{artifactid}-{version}.jar

- groupif'.'is replaced with '/'

4) maven central is primary community repo

- <http://repo1.maven.org/maven2>

* What is POM ..?

POM stands for project object model

As a fundamental unit of work in maven , POM is an XML file that contains information about project and confg details used by maven to build a project

#Describes a project

- Name and Version

- Artifact Type

- Source code locations

- Dependencies

- plugins

- profiles (Alternate build configurations )

#users XML by default

- not the way bant users xml

Day3

======

<project>

<build>

<plugins>

# what is the plugin GAV

<groupid> xxx </groupid>

<artifactid> yyy </artifactid>

<version> 123 </version>

<executions>

<execution>

# when the plugin has to be invoked

<phase> compile </phase>

<goals>

<goal> test </goal>

</goals>

<configuration>

# what the plugin has to do

<configuration>

</execution>

</executions>

</plugins>

</build>

</project>

MAVEN COMMANDS

===================

1. mvn clean

This command cleans the maven project by deleting the target directory. The command output relevant messages are shown below.

2. mvn compiler:compile

This command compiles the java source classes of the maven project.

3. mvn compiler:testCompile

This command compiles the test classes of the maven project.

4. mvn package

This command builds the maven project and packages them into a JAR, WAR, etc.

5. mvn install

This command builds the maven project and installs the project files (JAR, WAR, pom.xml, etc) to the local repository.when we want other project's jar file for other project then we we use this command

6. mvn deploy

This command is used to deploy the artifact to the remote repository. The remote repository should be configured properly in the project pom.xml file distributionManagement tag. The server entries in the maven settings.xml file is used to provide authentication details.

7. mvn validate

This command validates the maven project that everything is correct and all the necessary information is available.

8. mvn dependency:tree

This command generates the dependency tree of the maven project.

9. mvn dependency:analyze

This command analyzes the maven project to identify the unused declared and used undeclared dependencies. It’s useful in reducing the build size by identifying the unused dependencies and then remove it from the pom.xml file.

10. mvn archetype:generate

Maven archetypes is a maven project templating toolkit. We can use this command to generate a skeleton maven project of different types, such as JAR, web application, maven site, etc.

11. mvn site:site

This command generates a site for the project. You will notice a “site” directory in the target after executing this command. There will be multiple HTML files inside the site directory that provides information related to the project.

12. mvn test

This command is used to run the test cases of the project using the maven-surefire-plugin.

13. mvn compile

It’s used to compile the source Java classes of the project.

14. mvn verify

This command build the project, runs all the test cases and run any checks on the results of the integration tests to ensure quality criteria are met.

Maven Options

Maven provides a lot of command-line options to alter the maven build process. Let’s look at some of the important maven options.

15. mvn -help

This command prints the maven usage and all the available options for us to use.

16. mvn -f maven-example-jar/pom.xml package

This command is used to build a project from a different location. We are providing the pom.xml file location to build the project. It’s useful when you have to run a maven build from a script.

17. mvn -o package

This command is used to run the maven build in the offline mode. It’s useful when we have all the required JARs download in the local repository and we don’t want Maven to look for any JARs in the remote repository.

18. mvn -q package

Runs the maven build in the quiet mode, only the test cases results and errors are displayed.

19. mvn -X package

Prints the maven version and runs the build in the debug mode. It’s opposite of the quiet mode and you will see a lot of debug messages in the console.

20. mvn -v

Used to display the maven version information.

21. mvn -V package

This command prints the maven version and then continue with the build. It’s equivalent to the commands mvn -v;mvn package.

22. mvn -DskipTests package

The skipTests system property is used to skip the unit test cases from the build cycle. We can also use -Dmaven.test.skip=true to skip the test cases execution.

23. mvn -T 4 package

This command tells maven to run parallel builds using the specified thread count. It’s useful in multiple module projects where modules can be built in parallel. It can reduce the build time of the project.

https://www.journaldev.com/33645/maven-commands-options-cheat-sheet

<?xml version="1.0" encoding="UTF-8"?>

<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>scmlearning</groupId>

<artifactId>demo</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging> jar </packaging>

<name>demo</name>

<url>http://maven.apache.org</url>

<build>

<plugins>

<plugin>

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

<artifactId>maven-antrun-plugin</artifactId>

<version>1.1</version>

<executions>

<execution>

<id>id.clean</id>

<phase>compile</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo> ========= </echo>

<echo> ANT Clean Tasks </echo>

<echo> ========= </echo>

</tasks>

</cofiguration>

<execution>

</executions>

</plugin>

<plugin>

<groupId>org.codehaus.nojo</groupId>

<artifactId>exec-maven-plugin<artifactId>

<version>1.2.1</version>

<configuration>

<executable>git</executable>

<arguments>

<argument>--version</argument>

</arguments>

</configuration>

</plugin>

</plugins>

</build>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.0.1</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>