Maven – build tool & project management tool

* Learnings – To execute from Command line , customize maven default configurations , run & skip Unit Tests , external dependencies, setting parent & child in a multimodule project

Advantages of maven over others:

* Common interface for developers
* Convention over Configuration
* Dependency Management
* Easier to understand compared to other tools
* No need of creating Project Structure, like we have to do with other tools. Using maven Structure will be automatically created as we have archetypes present in maven for different project like Standalone , EAR.
* Inbuilt repository from where dependencies can be easily downloaded whether its available in opensource environment or local firm environment.

Reuse – Maven grabs whatever it needs as plugins.

Plugin Model – Compiler , SureFire(running unit tests) & WsImport(generate stubs from a wsdl file).

Creating a project from command line … use command –

mvn archetype:generate -DgroupId=com.kamal -DartifactId=hellomaven -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

archetype is a project goal here and we are passing parameters (with -D) in the goal . groupId is basically the package name, artifactId is the name of our project.

DarchetypeArtifactId is the type of project that we are creating which is actually a standalone project. Interactive mode is set to false as we don’t want to enter these values at the time of project creation. Here we are passing those values on Cmd itself.

Once the project is created we can build it using mvn install command & in the directory u will find that a new target Folder will be added and it will be having the jars created for the project.

You can run the project using jars as well like we do in java.

java -cp target\hellomaven-1.0-SNAPSHOT.jar com.kamal.App

highlighted field is the name of your source file.

What are maven plugins- collections of one or more goals. Example (archetype:generate & install:install)

Here Archetype is a plugin and generate is a goal.

A goal can be a specific task or it can be a part of a bigger build. And these goals can take several parameters as we have used with generate goal during project creation using cmd. These parameters can have default value .ex - DarchetypeArtifactId=maven-archetype-quickstart, if this value is not passed by default value will be picked.

Maven don’t know how to compile and do the packaging of project unless a plugin is used.

Maven LifeCycle

Whenever a mvn command is run , it goes through a lifecycle. Ex – Process-resources, compile , test, package. There are many other lifeclycle events which come in between, but these are the basic steps performed whenever mvn install command is run.

Every phase is attached with a goal itself.

Phase Goals

Process -resources resources:resources

Compile compiler:compiler

Test surefire:test

Package jar:jar

Depending on the type of project the association might change , if project is standalone , the package phase will be associated with jar:jar & if it’s a web application then it can be associated with war:war goal.

Coordinates – These four fields are called coordinates in Pom.

<groupId>com.kamal</groupId>

<artifactId>hellomaven</artifactId>

<packaging>jar</packaging>

<version>1.0-SNAPSHOT</version>

Maven plugins(such as jar & war) check the coordinates for the information related to project. Coordinates are internally representated as groupId:artifactId:version:packaging which will decide where the project will be stored in repository and what is the name of final output.

groupId:artifactId:version:packaging

helloMaven-1.0-SNAPSHOT.jar – you can compare the name of jar created at the end and the same will be pushed into the repository.

Maven Repositories

Whenever we run a command as mvn install, maven automatically starts downloading the dependencies as specified in pom.xml from the central repository. Default maven repository location – <http://repo.maven.apache.org/maven2>

In the dependencies section itself we can see the group Id & artifact Id which is used to download the jars available on repositories. Same Coordinates are used in the central repository as well.

--------------------------------------------------------------------------------------------------------------------------------------

Creating a project in eclipse

* Even if we have specified Jdk 1.8, after build You can see jRE system library(J2SE 1.5). reason -> if you move to effective pom section you can see the maven-compiler-plugin that is used. This plugin still uses j2se 1.5 that’s why J2SE is generated.

<build>

<plugins>

<plugin>

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

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

<version>3.7.0</version>

<configuration>

<!-- sepecify your own Configuration -->

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

Once changes are made for compiler plugin, right click on project ->maven->update project. You will see that it will change to JavaSe 1.8.

Skipping TestCases from commandLine

If we don’t want our testcases to be executed because of changes in the testcases by someone else.

Then we can easily do it using command line.

mvn install -DskipTests (we can also pass the value in -DskipTests to true or false also).

Skipping TestCases from Eclipse

Right click the project -> run as -> Run Configurations-> maven -> set the name of configuration -> check the SkipTest checkbox -> give the path of the base directory and click on run /

Test cases will be skipped.

If you run command – mvn clean . Maven will delete everything from target folder.

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

We can specify scopes in maven for dependencies. If we want a dependency to be utilised only during test we can mark the scope as test.

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

So here junit dependency will be used only during test but not at the time of compilation.

Similarly we do have other scopes like “provided”. If we are marking the scope as “provided” then this jar will not be packaged into the war file. For ex- if we don’t want servlets file to be packaged into war we can mark them as provided as those jars will be available on application servers like tomcat /glassfish/websphere.

Creating a Web application using maven

The only change here is that we will have Packaging set as War instead of jar in standalone project.

We will be getting an error in index.jsp file, saying The superclass "javax.servlet.http.HttpServlet" was not found on the Java Build Path.

It can be resolved by two ways : first is to set the dependencies for servlet in pom.xml .

Second is to right click on the project and set the targeted runtimes in properties. Select the apache tomcat we are using . Since tomcat itself contains support for servlet. The error will be removed.

Once mvn install command is run, A war file is created under target folder.

# Multi module project in Maven

To create a multi module project we need to create a parent pom i.e., placed in the present directory of all the projects.

We need to specify the group IDs & change the versioning & packaging accordingly. Place it in

Parent Node.

Make the same changes in Project & child project.

Now go to the project in cmd and run command mvn clean install. You will see that the project is getting build.

On Cmd u can see a section with name Reactor build order- In this section maven actually checks which project is dependent on which one and builds it in the same order.

If u go down the logs in cmd u can see that the maven builds the project (parent one) and pushes it into local repository.

All the code changes can be done accordingly and dependencies can be set in pom.xml of parent & related projects.

Now once u build your web project u can see there is a war file generated. If you open up that war using any Zipping software, u can see all the jars within it including the parent project dependency.

Jars will be present under web-inf -> Lib

## Scopes- Compile, Test, provided , runtime, system, import

Using scopes in maven to define the scopes of dependency in maven lifecycle informing when the dependency will available.

**Compile** – This dependency will be available during build, Test & run. By default scope is compile in maven. If I don’t declare any thing Default is picked up.

# Provided - Any dependency that you don’t want to be exported is marked as Provided. It will be available during build, test & run. For ex- if you are creating a web application, then you will be using servlet-api dependency. This dependency will be required when you are executing your code locally but is not required to be deployed to webcontainers as they have it already.

**Runtime –** Available during run & Test.

## Test- Available during compile & running the test.

**System -**  not used much. Somewhat similar to provided scope. It is not fetched from maven repository. If you want to point to a library that is not a part of maven dependency and not available in web container then use the system scope & provide the path to external dependency.

Ex - <SystemPath>

${baseDir}\war\ path

</SystemPath>

**Import-**  It is used on POM based project, not jar based or war based.