What is Maven Build Lifecycle?

When we build a Maven project, it executes a set of clearly defined tasks based on the project pom.xml configuration and the command-line options. This standard set of tasks creates the maven build lifecycle. The benefit of a clearly defined lifestyle is that we have to remember only a few sets of commands to compile, build, install, and deploy our projects

Built-in Build Lifecycles

There are three built-in build lifecycles.

default: handles project build and deployment (important)

clean: handles project cleaning

site: handles the creation of project site documentation (report generation activities)

Total

23 phases

are there

in default

life cycle but listed

Maven Build Phases

Maven build lifecycle goes through a set of stages, they are called build phases. For example, the default lifecycle is made up of the following phases.

validate compile

test

Validates the project directory structure

----->compiles the source code

----> executes the junit test cases

package ---->packs the content into war file/jar file/...

verify ----> verfies post packing activities

install ----> keeps the packaged jar/war file to maven local repository

deploy ---> deploys the packaged war /jar file to the server

For all 23 phases of Default Lifecycle

refer bottom of this Document

The build phases are executed sequentially. When we run a maven build command, we specify the phase to be executed. Any maven build phases that come only important before the specified phase is also executed. For example, if we run mvn package then it will execute validate, compile, test, and package phases of the project. ones here (17th phase) (before 16 phases +17 th phase)

=>Every phase will have goals to complete i.e we can link our code/task to the goals of maven life cycle phases

For more info on goals refer this :: https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html

Phase

process-resources

compile

process-test-resources

plugin:goal

resources: resources

compiler:compile

resources:testResources

compiler:testCompile

=>maven latest version is 3.x

=>maven default life cycle is having 23 phases => maven clean life cycle is having 3 phases site life cycle is having 4 phases

=> maven

surefire:test

ejb:ejb or ejb3:ejb3 or jar: jar or par: par or rar:rar or war: war

test-compile

test

package

install

install: install

deploy

deploy: deploy

In Maven pom.xml file, the maven project, maven plugin and maven dependency (jar file) are identified with 3 details

a) group Id (company name of project/plugin/ jar file)

b) artfict id (project name /plugin name /jar file name)

c) version (project version/plugin version /jar file version)

=> Plugin is a patch software that performs

for example i want1add spring -context support library (jar file) to the maven project through pom.xml file

some task to complete.

<!-- https://mvnrepository.com/artifact/org.springframework/spring-context-support -->

<dependency>

<groupId>org.springframework</groupId> <artifactId>spring-context-support</artifactId>

<version>6.0.11</version>

</dependency>

place this <dependency> tag in pom.xml file

the maven project gets spring-context-support jar file

and its transitive dependencies from maven central repository

if they are not available in the Maven Local Repository. or Remote Repository

note:: When they brought from Central repository one copy will be kept maven Local Repository.

in the

=>Plugin is patch software that provides addtional functionalities to the existing software or software apps to
eg:: plugin added eclipse gives more facilities to eclipse

eg:: STS plugin, gui builder plugin, maven plugin, gradle plugin and etc..

to

note:: maven plugins are given to complete certain tasks of the build process by linking them various phases of maven life cycle.

examples of maven plugins

maven-compiler-plugin

maven -clean -plugin

=> A plugin is a patch/additional software that defines some functionality to perform

maven -exec-plugin

and etc..

phase

goal1

plugin1

Maven tool

Lifecycle

phase2

goal2

plugin2

phase3

=>In each Maven built-in Life cycle there will be multiple phases each phase contains 1 or more goals.. For these goal we can execute custom logics/actions through plugins

=> The goals of the phases belonging to Built-in life cycle are already linked with some built-in plugins having pre-defined actions to perform.. The same thing can be customized by linking other plugins to the goals

Example Scenario

==========

Default lifecycle is having compile phase --> this compile phase is having compile goal ---> this compile goal already linked with maven-compile-plugin for compilation of source code.. To make the same compile goal of compile phase also performing app execute operation we need to link "maven-exec-plugin" to compile goal of compile phase belonging to "default" life cycle

Example1 (Example on Dependency Managment --> jar file/Libraries)

(library management)

is

step1) launch eclipse sdk/Jee IDE (2020+ version -- becoz maven plugin built-in plugin in 2020+ eclipse IDE)

step 2) create maven project as the stanalone project using the "maven-archetype-quickstart" archetype

file menu ---> maven project ----> next ---> search for maven-archetype-quickstart -->

Filter:

maven-archetype-q

Group Id

com.github.ywchang

com.haoxuer.maven.archetype

org.apache.maven.archetypes

Artifact Id

maven-archetype-quickstart maven-archetype-quickstart maven-archetype-quickstart

--->next --->

=>"SNAPSHOT" in the

version indicates the

New Maven project

Specify Archetype parameters

Group Id: nit

Project is still in developed MavenProj01

=>RELEASE in the

version indicates the

Project is completely

0.0.1-SNAPSHOT

com.nt.test

(company name)

(project name)

(version)

(default package name)

Version: Package: Properties available from archetype: Name

Version

1.1

1.01

(select this)

1.4

Eclipse IDE

SDK:: Standard Development Kit

JEE :: Java Enterprise Edition

Eclipse SDK

==========

=> For standalone

Apps Development

Eclipse JEE

=========

=> For standalone,

web applications, distributed Apps development

1.5 (latest)

note: if archetype is not coming in the list then we need to add the archtype manually

add archetype ---->

Value

1.5

Archetype Group Id: org.apache.maven.archetypes Archetype Artifact Id: maven-archetype-quickstart Archetype Version:

Collect this info from mvnrepository.com

--->finish ---> say "y" in the console to confirm

Repository URL:

the Project creation process

& Tools

Dale

Files

ready for Release

step3) observe the directory sturfture of the Project

MavenProj01

#src/main/java

com.nt.test

› App.java

src/test/java

for placing java source packages

com.nt.test for placing junit code packages

> App Test.java

> JRE System Library [JavaSE-1.7]

✓

Maven Dependencies

>junit-4.11.jar - C:\Users\NATARAJ\.m2\repository\junit\junit\4.11

>Thamcrest-core-1.3.jar - C:\Users\NATARAJ\.m2\repository\org\hamcrest\hamcrest-core 1.3

src

main

✓ java

com

nt

✓ test

App.java

✓ test

✓ java

com

nt

test

AppTest.java

target

Mpom.xml

default jar files/libraries/

dependents in every maven project

To hold the generated outputs

Configuration file (xml file) to provide

inputs and instructions to maven tool

step4) change the project's java version to latest version/ u r choice version through pom.xml file

a) open pom.xml change java version from 1.7 to 17 max (in 2022-03 IDE)

<properties>

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

<maven.compiler.release> </maven.compiler.release>

</properties>

b) Perform maven update operation

21

right click on the project ---> maven ---> update project --> ...

step4) add main jar file <dependency> tag under <dependencies> tag of pom.xml to feel

the transitive dependency

Before::

In pom.xml

<dependencies>

<dependency>

<groupId>junit</groupId>

In Project

Maven Dependencies

נ

--> ok

default

dependency

<artifactId>junit</artifactId> <version>4.11</version>

<scope>test</scope>

in pom.xml file

</dependency>

</dependencies>

after adding our dependencies(jar files info)

>junit-4.11.jar - C:\Users\NATARAJ\.m

t\4.11

Ehamcrest-core-1.3.jar - C:\Users\NATARAJ\.m2\repository\org\hamcrest\hamcrest-core\1.3

default jar files

=>we can change to any compatible higher version of java to the Project maven project but we need not to install that version of java in our computer becoz maven s/w dynamically downloads that version of jdk s/w

In Project

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

default code

<version>4.11</version>

<scope>test</scope>

Maven Dependencies

mvnrepository.com/artifact/org.apache.maven.archetypes/maven-archetype-quickstart/1.5

Repositories

Ranking

Aug 20, 2024

pom (1 KB) maven-archelype (10 KB) View All

Central Apache Staging

#515137 in MvnRepository (See Top Artifacts)

#2187 in Maven Archetypes

Maven Gradle | Gradle (Short) Gradle (Kotlin) | SBT Ivy Grape Leiningen Buildr

```
<!--
https://mvnrepository.com/artifact/org.apache.maven.archetypes/maven-archetype-quickst
art --> <dependency>
```

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

```
<artifactId>maven-archetype-quickstart</artifactId>
<version>1.5</version>
</dependency>
```

> junit-4.11.jar - C:\Users\NATARAJ,m2\repository\junit\junit4.11 >hamcrest-core-1.3.jar -

C:\Users\NATARAJ_m2\repository\org\hamcrest\hamcrest-core 1.3

>spring-context-support-6.0.11.jar - C:\Users\NATARAJ\.m2\repository\org\springframework\sp
>spring-beans-6.0.11.jar - C:\Users\NATARAJ,m2\repository\org\springframework\spring-bean:
>spring-context-6.0.11.jar - C:\Users\NATARAJ\,m2\repository\org\springframework\spring-con >spring-aop-6.0.11.jar - C:\Users\NATARAJ\.m2\repository\org\springframework\spring-aop\6.0 >spring-expression-6.0.11.jar - C:\Users\NATARAJ\.m2\repository\org.springframework\spring-e >spring-core-6.0.11.jar - C:\Users\NATARAJ\.m2\repository\org\springframework\spring-core\6. >spring-jcl-6.0.11.jar - C:\Users\NATARAJ\.m2\repository\org\springframework\spring-jcl6.0.11

Here we are getting both

main and dependent jar files till the end of hierarchy satisfying transitive dependency

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-context-support -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context-support</artifactId>

<version>6.0.11</version>

</dependency>

Go to mvnrepository.com --> search for spring context support

select 6.0.11 version --> select maven tab --> copy and paste the xml code (latest)

note:: mvnrepository.com is not a maven central repository it is a third party website

supplying various details related to pom.xml configurations of maven tool

Lifecycle Reference

The following lists all build phases of the default, clean and site lifecycles, which are executed in the order given up to the point of the one specified.

Clean Lifecycle

Phase

`pre-clean`

`clean`

`post-clean`

Default Lifecycle

Description

execute processes needed prior to the actual project cleaning

remove all files generated by the previous build

execute processes needed to finalize the project cleaning

Phase

Description

`validate`

validate the project is correct and all necessary information is available.

`initialize`

generate-resources

process-resources

compile

process-classes

initialize build state, e.g. set properties or create directories.

generate resources for inclusion in the package.

copy and process the resources into the destination directory, ready for packaging. compile the source code of the project.

post-process the generated files from compilation, for example to do bytecode enhancement on Java classes. generate any test source code for inclusion in compilation.

generate-test-sources

process-test-sources

process the test source code, for example to filter any values.

generate-test-resources create resources for testing.

process-test-resources copy and process the resources into the test destination directory.

test-compile

process-test-classes

test

prepare-package

package

integration-test

post-integration-test

verify

install

deploy

Site Lifecycle

compile the test source code into the test destination directory

post-process the generated files from test compilation, for example to do bytecode enhancement on Java classes.

run tests using a suitable unit testing framework. These tests should not require the code be packaged or deployed. perform any operations necessary to prepare a package before the actual packaging. This often results in an unpacked, processed version of the package.

take the compiled code and package it in its distributable format, such as a JAR.

process and deploy the package if necessary into an environment where integration tests can be run.

perform actions required after integration tests have been executed. This may including cleaning up the environment.

run any checks to verify the package is valid and meets quality criteria.

install the package into the local repository, for use as a dependency in other projects locally.

done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

Phase

Description

`pre-site`

`site`

execute processes needed prior to the actual project site generation

generate the project's site documentation

`post-site`

execute processes needed to finalize the site generation, and to prepare for site deployment

`site-deploy`

deploy the generated site documentation to the specified web server

What is the difference b/w Archetype and artifact In maven Project?

Ans) Archetype in maven represents Project Template using which we can create certain category Project

=>Artifact in maven represents an item, this item can be project or jar file or plugin

=> Artifact name means project name or jar file name or plugin name

How to see dependencies and dependencies hierarchy related to pom.xml file?

Ans) use dependencies and dependencies hierarchy tabs of pom.xml file as shown below

Dependencies

junit-jupiter-api [test] (managed:5.11.0)

junit-jupiter-params [test] (managed:5.11.0)

contains the info about the

dependencies added to the pom.xml file

spring-context-support: 6.2.5

Dependency Hierarchy

junit-jupiter-api : 5.11.0 [test]

opentest4j: 1.3.0 [test]

✓ junit-platform-commons: 1.11.0 [test]

apiguardian-api: 1.1.2 [test]

Contains the info about dependencies

and their hierarchy

apiguardian-api: 1.1.2 [test]

junit-jupiter-params: 5.11.0 [test]

junit-jupiter-api : 5.11.0 [test]

apiguardian-api: 1.1.2 [test]

spring-context-support: 6.2.5 [compile]

spring-beans: 6.2.5 [compile]

spring-core: 6.2.5 [compile] >spring-context: 6.2.5 [compile]

✓ spring-aop: 6.2.5 [compile]

spring-beans: 6.2.5 [compile] spring-core: 6.2.5 [compile]

spring-beans : 6.2.5 [compile]

spring-core: 6.2.5 [compile]

spring-expression: 6.2.5 [compile]

spring-core: 6.2.5 [compile]

✓ micrometer-observation : 1.14.5 [compile]

micrometer-commons: 1.14.5 [compile]

spring-core: 6.2.5 [compile]

spring-jcl: 6.2.5 [compile]

What is the location of maven local repository?

<user_home>/.m2/repository

C

> This PC > Local Disk (C:)

↑ Sort

> Users > Nataraz > .m2 > repository >

View✓

A

Name

antlr

aopalliance

Date modified

lype

Size

asm

9/20/2024 7:10 AM 9/20/2024 8:26 AM 9/25/2024 10:40 AM

File folder File folder

Filc folder

avalon-framework backport-util-concurrent

ch

classworlds

com

10/23/2024 10:11 AM 9/20/2024 8:25 AM 9/20/2024 7:09 AM 9/20/2024 8:25 AM 10/23/2024 10:11 AM

I ile folder

File folder

File folder

File folder File falder

What is the url of maven Central Repository?

←

→ C

0-

../

HTTPClient/

abbot/

academy/ acegisecurity/

activation/

activecluster/ activeio/

repo.maven.apache.org/maven2/

if archetype is not list out while creating maven project then we need to pass add archetype as shown below

Add Archetype

Add Archetype

Specify Archetype and Maven repository URL

Archetype Group Id:

org.apache.maven.archetypes

Archetype Artifact Id: maven-archetype-quickstart

Archetype Version:

1.5

Repository URL:

==> next ---> next --> ....

X