**Maven**

**Maven** is a powerful project management tool that is based on POM (project object model). It is used for projects build, dependency and documentation. It simplifies the build process like ANT. But it is too much advanced than ANT.

There are many problems that we face during the project development. They are discussed below:

**1) Adding set of Jars in each project:** In case of struts, spring, hibernate frameworks, we need to add set of jar files in each project. It must include all the dependencies of jars also.

**2) Creating the right project structure:** We must create the right project structure in servlet, struts etc, otherwise it will not be executed.

**3) Building and Deploying the project:** We must have to build and deploy the project so that it may work.

**What it does?**

Maven simplifies the above mentioned problems. It does mainly following tasks.

1. It makes a project easy to build
2. It provides uniform build process (maven project can be shared by all the maven projects)
3. It provides project information (log document, cross referenced sources, mailing list, dependency list, unit test reports etc.)
4. It is easy to migrate for new features of Maven

Apache Maven helps to manage

* Builds
* Documentation
* Reporing
* SCMs
* Releases
* Distribution

**What is Build Tool**

A build tool takes care of everything for building a process. It does following:

* Generates source code (if auto-generated code is used)
* Generates documentation from source code
* Compiles source code
* Packages compiled code into JAR of ZIP file
* Installs the packaged code in local repository, server repository, or central repository

**Ant** and **Maven** both are build tools provided by Apache. The main purpose of these technologies is to ease the build process of a project.

**ANT** stands for **Another Neat** Tool. It is a Java-based build tool from computer software development company Apache. Before going into the details of Apache Ant, let us first understand why we need a build tool.

A **maven repository** is a directory of packaged JAR file with pom.xml file. Maven searches for dependencies in the repositories. There are 3 types of maven repository:

**Local Repository**:- Maven **local repository** is located in your local system. It is created by the maven when you run any maven command.

By default, maven local repository is %USER\_HOME%/.m2 directory. For example: **C:\Users\SSS IT\.m2**.

**Central Repository**:- Maven **central repository** is located on the web. It has been created by the apache maven community itself.

**Remote Repository**:- Maven **central repository** is located on the web. It has been created by the apache maven community itself.

**POM** is an acronym for **Project Object Model**. The pom.xml file contains information of project and configuration information for the maven to build the project such as dependencies, build directory, source directory, test source directory, plugin, goals etc.

|  |  |
| --- | --- |
| **Element** | **Description** |
| **project** | This is the root element of pom.xml file. |
| **modelVersion** | This is the sub element of project which specifies the modelVersion. Model version should be 4.0.0. |
| **groupId** | This is the sub element of project which specifies the id for the project group. |
| **artifactId** | This is the sub element of project which specifies the id for the project. This is generally refers to the name of the project. The artifact ID is also used as part of the name of the JAR, WAR or EAR file produced when building the project. |
| **version** | This is the sub element of project which specifies the version of the project. |
| **packaging** | It is used to define the packaging type such as jar, war etc. |
| **name** | It is used to define the name of the maven project. |
| **url** | It is used to define the url of the project. |
| **dependencies** | It is used to define the dependencies for this project. |
| **dependency** | It is used to define a dependency. It is used inside dependencies element. |
| **scope** | It is used to define the scope for this maven project. It can be compile, provided, runtime, test and system. |

**Maven application**

To create a simple java project using maven, you need to open command prompt and run the **archetype:generate** command of mvn tool.

mvn archetype:generate -DgroupId=groupid -DartifactId=artifactid

-DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=booleanValue

**Maven web application**

We can create a simple maven web application example by executing the **archetype:generate** command of **mvn tool**.

**Syntax**

mvn archetype:generate -DgroupId=groupid -DartifactId=artifactid

-DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=booleanValue

1. mvn clean compile
2. java com.javatpoint.App  - Run

The **mvn package** command completes the build life cycle of the maven project such as:

* validate - validate the project is correct and all necessary information is available
* compile - compile the source code of the project
* test - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
* package - take the compiled code and package it in its distributable format, such as a JAR.
* verify - run any checks on results of integration tests to ensure quality criteria are met
* install - install the package into the local repository, for use as a dependency in other projects locally
* deploy - done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

The **maven plugins** are central part of maven framework, it is used to perform specific goal.

According to Apache Maven, there are 2 types of maven plugins.

1. Build Plugins:- These plugins are executed at the time of build. These plugins should be declared inside the **<build>** element.
2. Reporting Plugins:- These plugins are executed at the time of site generation. These plugins should be declared inside the **<reporting>** element.

|  |  |
| --- | --- |
| clean | clean up after build. |
| compiler | compiles java source code. |
| deploy | deploys the artifact to the remote repository. |
| failsafe | runs the JUnit integration tests in an isolated classloader. |
| install | installs the built artifact into the local repository. |
| resources | copies the resources to the output directory for including in the JAR. |
| site | generates a site for the current project. |
| surefire | runs the JUnit unit tests in an isolated classloader. |
| verifier | verifies the existence of certain conditions. It is useful for integration tests. |