Maven

Overview:

Maven is a project management and comprehension tool (Comprehension: ability to understand). Maven provides developers to manage the following:

1. Builds
2. Documentation
3. Reporting
4. Dependencies
5. SCMs (Source Code Management)
6. Releases
7. Management
8. Distributions
9. Mailing list

Maven can manage more number of projects which contain slightly different code I.e., maven shares similar jars across all the projects.

Maven’s main aim is to provide developers

1. A comprehensive model for projects which is reusable, maintainable and easier to comprehend
2. Plugins or tools that interact with this declarative model

Maven project’s structure and contents are declared in an xml file called *pom.xml* file (project object model) which is fundamental unit of Maven

Maven uses *Convention over Configuration* which means developers are not required to define each & every configuration in detail.

* It uses certain structure for projects
* Developers need to just place the files accordingly

In the following table, we can see how a project’s structure is created in maven. Here ${basedir} is the Project’s location.

|  |  |
| --- | --- |
| **Item** | **Default** |
| source code | ${basedir}/src/main/java |
| resources | ${basedir}/src/main/resources |
| Tests | ${basedir}/src/test |
| distributable JAR | ${basedir}/target |
| Complied byte code | ${basedir}/target/classes |

Maven plugins will manage much of the Project management and Build related tasks.

Features of Maven:

* Simple project setup that follows best practices
* Consistent usage across all projects
* Dependency management including automatic updating
* A large and growing repository of libraries
* *Extensible, with the ability to write plugins in java or other scripting language*
* Instant access to new features with little or no extra configuration
* **Model based builds :** Maven is able to build any number of projects into predefined output types like jar, war, metadata
* **Coherent site of Project Information:**using meta data, maven can generate a website and a pdf documentation including complete documentation
* **Release management and distribution publication:** Maven will integrates with source control system (VCS, DVCS like svn, git etc., ) and manages the release of projects with no additional configuration
* **Backward Compatibility:** we can easily port older version modules into newer version.
* **Automatic parent versioning:**  No need to specify the parent in the sub-module for maintenance.
* **Parallel Builds:**It analyzes the project dependency graph and enables you to build schedule modules in parallel. Using this feature, we can achieve the performance improvements of 20-50%
* **Better Error and Integrity reporting:** Maven improved error reporting and it provides a link to maven wiki page where we can get the full description of the error.

POM

POM stands for Project object model. It’s an xml file that resides in the base directory of the project.

* It contains basic information about the Project and various configuration detail used by maven to build the projects.
* POM.xml contains the goals and plugins.
* When maven executes the tasks or goals, maven always looks for POM.xml, collects the needed configuration from the POM.xml and then executes the goal.
* Some of the configuration specified in the POM is

Project dependencies

Plugins

Goals

Build profile

Project version

Developers

Mailing list

* GroupID(~Package) , ArtifactID(~Name of the class) and version must be present in POM.xml. The above three attributes helps to identify a project in a repository uniquely.

Project's notation in repository : groupid:artifactid:projectVersion

Ex: com.fiorano.tools.studio.ClassFirst:1.0

com.fiorano.tools.studio.ClassFirst:2.0

Project root, groupId, artifactId and version attributes are needed to be present in the POM.xml

Super POM

Super POM is Maven's 'default POM'. All POM's inherit configuration from 'Super POM'.

* Super pom helps the developers to write little configuration in their pom.xml files since their pom.xml inherits from Super POM.
* Although Configurations can be overridden easily.
* In a repository, we can find the effective POMs by using the following command.

mvn help:effective-pom

Maven Build Lifecycle

Note:

* 'Phase in maven' represents a 'Stage in Life cycle'.
* 'Goal in maven' is similar to 'Target in ant'
* Also 'Phase in maven' is similar to 'Task in ant'

A Build lifecycle is well defined sequence of phases, which define the order in which the goals are to be executed.

A 'typical (~Normal) Maven build Life cycle' consists of the following sequence of phases.

|  |  |  |
| --- | --- | --- |
| Phase | Handles | Description |
| Prepare-resources | Resource copying | Resource copying can be customized in this phase |
| Validate | Validating the Information | Validates if the project is correct and if all necessary info. Is available |
| Compile | Compilation | Compilation is done in this phase |
| Test | Testing | Test the compiled source code suitable for testing framework |
| Package | Packaging | This phase creates the JAR/WAR as mentioned in the POM.xml |
| Install | Installation | This phase installs the package in local/remote repository |
| Deploy | Deploying | Copies the final package to the remote repository |

There are always 'pre & post defined phases’ to register goals which must run after or prior to a particular phase

When maven starts building a project, it follows through a *defined sequence of phases and executes goals* which are registered with each phase.

Maven has the following three standard Life cycles

1. Clean

2. Default (build)

3. Site

A goal represents a specific task which contributes to the building and managing of a project. A goal may be bound to zero or more phases. A goal which doesn’t bound to any phase will be invoked directly from outside.

Execution order depends on the order in which the goals and build phases are invoked

Ex: mvn clean dependency:copy-dependencies install

In the above example, the phases clean and install are invoked along with the ‘dependency:copy-dependencies’ goal. The execution order will be like this…

1. Clean phase
2. dependency:copy-dependencies goal
3. Install phase

Clean Life Cycle:

When we execute post-clean command, maven will invoke the clean life cycle consisting of the following phases.

* Pre-clean
* Clean
* Post-clean

When we execute ‘mvn clean’ command, as the clean goal [Rep: clean:clean] is bound to the clean phase in the life cycle it deletes the output directory i.e., build directory of maven build.

We can customize this behavior by including different goals in the above phases.

Default (Build) Life Cycle:

Maven has 24 phases in default build lifecycle. All the 24 phases are sequential. If we execute 10th phase, the phases upto 10th and also 10th phase will also be executed.

Site Life Cycle:

Maven site plugin is generally used to create fresh documentation to create reports, deploy site etc., It has the following phases...

1. Pre-site
2. Site
3. Post-site
4. Site-deploy

Build Profiles

Build Profile is a set of configuration values, which can be used to set or override default values of Maven build. Using a build profile, we can customize build for different environments such as production, development environments.

Profiles are specified in POM.xml using its activeProfiles/ profiles elements and are triggered in variety of ways. Profiles modify the POM at build time, and are used to give parameters for different environments. [ for ex: path of db server in the development, testing and production environments ]

Types of build profiles:

1. Per project
2. Per user
3. Global

|  |  |
| --- | --- |
| Type | Where it is defined |
| Per Project | Defined in the project POM.xml, pom.xml |
| Per user | Defined in the maven settings.xml file... ($userhome/.m2/settings.xml file).  > If it doesn’t present at that location, create one and use |
| Global | Defined in the maven settings.xml file located @ $M2\_HOME/conf/settings.xml file.. |

Profile Activation:

1. Explicitly using console command input.
2. Through Maven Settings
3. Based on Environment Variables (User/System variables)
4. OS settings. (For ex: making certain profile to execute if the OS is of windows family)
5. Present/missing files.

Explicitly using console command input :

Defining the profile configuration in POM.xml

* We can specify more than one profile in POM.xml file.
* We can invoke one of the profiles using the following command line input.

Syntax: mvn <phase> -P<defined profile-name>

Profile activation 'Through Maven Settings' : (**IN SETTINGS.XML FILE**)

Open settings.xml in '.m2' directory. If it's not there, create one and open. Add TEST (say) profile as an active profile using 'activeprofiles' node (I/p and O/p tag Or Element ).

Now move into some directory that contains 'POM.xml' and run 'mvn' command there. No need to provide '-P' argument there. Just invoke like this :

Ex: mvn test

Based on Environment Variables: (**IN POM.XML FILE**)

Remove active profile section in 'settings.xml' file. Use activation element to profile element in POM.xml. Declare properties in the POM.xml file using property element.

The test profile will get trigger(to activate) when the property 'env' is set to 'test'.

Here also no need to provide the option '–P' with command 'mvn'

Ex: <profile>

<id>test</id>

<activation>   
 <property>  
 <name>env</name>  
 <value>test</value>  
 </property>  
 </activation>  
</profile>

Profile activation via Operating system:

Add 'os' element to activation element. The following 'test' profile will trigger when the OS is XP. Here too, we don’t need to provide the parameter '-P' with mvn command.

Ex:  
 <profile>  
 <id>test</id>  
 <activation>  
 <os>  
 <name>Windows XP</name>  
 <family>Windows</family>  
 <arch>x86</arch>  
 <version>5.1.2600</version>  
 </os>  
 </activation>

</profile>  
  
Profile activation via present/Missing file:

Here we'll provide the 'file' element inside the 'activation' element. In that 'file' element, we also provide either 'missing' or 'present' elements.

Here also, we don’t need to provide the parameter '-P'

Ex: <profile>  
 <id>test</id>  
 <activation>  
 <file>  
 <missing>target/generated-  
 sources/axistools/wsdl2java/com/companyname/group</missing>  
 </file>  
 </activation>  
 </profile>

Maven Repositories

Maven uses 3 types of repositories.

1. Local Repository
2. Central Repo
3. Remote Repo

In Maven terminology, a repository is a directory where all the "project jars, library jar or any other project specific artifacts are stored" and can be used by maven easily.

Local Repository:

Local Repo is a folder which resides on your local system. This will be created when you run any of the 'maven command (mvn)' for the first time in the local system.

* By the first time itself, maven downloads all the dependency jars, plugins to local system's repository such that it can avoid referring to central repository and remote repository when we run maven commands.
* Defaultly, it'll be created in the user home directory.

Ex: In linux: /root/.m2

In windows: c:/Users/administrator/.m2

* If we would like to change the default Repository path to be created, open settings.xml in 'maven\_home/conf/' and change the path to your desired one in "localrepository element".

Central Repository:

This is the one which resides in Internet and is maintained by Maven community. We can get almost all the plugins and jars what we didn’t get in local repository. If it doesn’t find here, it throws an error (I can't find like that...).

* If user configures remote repo, it looks there after the above failure

Remote Repository:

This is the one managed by user himself/herself in remote system. Maven will look into this when it doesn’t get jars from local and central repositories. This is the final hope to get needed things.

Note: we need internet to get the jars into the local system. Once local repo is created we can manage up to some extent without internet

Flow:

Ran some maven command --–--looks into-----> Local repo ---if not found in local repo, looks into-------> Central repo ----If not found here too-----> throws error.

If User configured Remote Repository:

Central repo ----- If not found here----> throws error & looks into remote repo [ This is the end ]