UNIT II COMPILE AND BUILD USING MAVEN & GRADLE 6

Introduction, Installation of Maven, POM files, Maven Build lifecycle, Build phases(compile build, test, package) Maven Profiles, Maven repositories(local, central, global),Maven plugins, Maven create and build Artifacts, Dependency management, Installation of Gradle, Understand build using Gradle

MAVEN PROFILES

A Maven profile is a customizable set of configuration parameters within a Maven Project Object Model (POM) file that allows for the modification of build behavior based on specific environments or conditions. It provides a mechanism to tailor the build process for different scenarios, such as development, testing, or production environments, without requiring changes to the core POM file or maintaining multiple project configurations.

Key aspects of Maven profiles:

Customization: Profiles enable overriding or extending default Maven build settings, including dependencies, plugins, plugin configurations, properties, and build directories.

Environment-specific builds: They facilitate creating distinct builds optimized for different environments, such as using different database configurations for development versus production, or employing specific testing frameworks for integration tests.

Activation: Profiles can be activated explicitly via command-line flags (e.g., mvn -P profile-id), or automatically based on various conditions like operating system, JDK version, presence of specific files, or system properties.

Portability: By centralizing environment-specific configurations within profiles, Maven aims to maintain project portability and reduce inconsistencies across different developer machines or build servers.

Location: Profiles are typically defined within the <profiles> section of the pom.xml file, but can also be defined in the settings.xml file for user-specific or global configurations.

Profiles are specified using a subset of the elements available in the POM itself (plus one extra section), and are triggered in any of a variety of ways. They modify the

POM at build time, and are meant to be used in complementary sets to give equivalent-but-different parameters for a set of target environments. Profiles can easily lead to differing build results from different members of the team.

Different types of profile:

- Per Project
  - Defined in the POM itself (pom.xml).
- Per User
  - Defined in the Maven-settings (%USER_HOME%/.m2/settings.xml).
- Global
  - Defined in the global Maven-settings (${maven.home}/conf/settings.xml).

Profile Inheritance

The profiles are not inherited as other POM elements by child POMs. They are resolved very early by the Maven Model Builder and only the effects of active profiles are inherited.

Elements of profile

A profile in Maven is defined within the <profiles> element in the pom.xml or settings.xml file, and it contains several key elements:

1. **<id>:**
   This is a mandatory element that provides a unique identifier for the profile. This ID is used to activate the profile, either explicitly via the command line (e.g., mvn -P<profile_id>) or through other activation mechanisms.
2. **<activation>:**
   This element defines the conditions under which a profile will be automatically activated. Activation can be based on various factors, including:
   - **<activeByDefault>**: If set to true, the profile will be active unless another profile is explicitly activated.
   - **<jdk>**: Activates the profile based on the Java Development Kit (JDK) version.
   - **<os>**: Activates the profile based on the operating system name, family, arch, or version.

- **<property>**: Activates the profile if a specific system property is present or has a particular value.
- **<file>**: Activates the profile based on the existence or non-existence of a file.

3. **<properties>**:

   This element allows the definition of profile-specific properties as key-value pairs. These properties can be used to override or define configuration values that are specific to the activated profile, such as database URLs, API keys, or environment-specific settings.

4. **<build>:**

   A profile can contain a <build> element to modify the build process itself. Within this, you can configure elements such as:

   - **<plugins>**: Define or modify the configuration of Maven plugins, including their goals and executions.
   - **<resources> and <testResources>**: Specify different resource directories for the main source and test source.
   - **<finalName>**: Override the final name of the project's artifact.

5. **<repositories> and <pluginRepositories>:**

   These elements allow the specification of remote repositories for dependencies and plugins, respectively, that are specific to the profile.

6. **<dependencies> and <dependencyManagement>:**

   Profiles can define or modify project dependencies, including their versions and scopes, and also influence dependency management settings.

7. **<modules>:**

   If the project is a multi-module project, a profile can define which modules should be included in the build for that specific profile.

8. **<reporting>:**

   This element allows customization of reporting configurations within a profile.

9. **<distributionManagement>:**
   Profiles can override distribution management settings, which define where the project's artifacts are deployed (e.g., to a staging server in a "staging" profile).

Profile Activation:

A profile can be activated in several ways:

- Explicitly
- Implicitly
  - Based on OS
  - Based on system properties
  - Based on presence of files

Explicit profile activation

Profiles can be explicitly specified using the -P command line flag. This flag is followed by a comma-delimited list of profile IDs to use. The profile(s) specified in the option are activated in addition to any profiles which are activated by their activation configuration or the <activeProfiles> section in settings.xml. To make the profile optional, the profile identifier is prefixed with an ?.

```
mvn groupId:artifactId:goal -P profile-1,profile-2,?profile-3
```

Profiles can be activated in the Maven settings, via the <activeProfiles> section. This section takes a list of <activeProfile> elements, each containing a profile-id inside.

```
<settings>
 ...
 <activeProfiles>
   <activeProfile>profile-1</activeProfile>
 </activeProfiles>
 ...
</settings>
```

Profiles listed in the <activeProfiles> tag would be activated by default every time a project use it. Profiles can also be active by default using a configuration like the following in a POM:

```
<profiles>
 <profile>
   <id>profile-1</id>
   <activation>
     <activeByDefault>true</activeByDefault>
   </activation>
   ...
 </profile>
</profiles>
```

This profile will automatically be active for all builds. All profiles that are active by default are automatically deactivated when a profile in the POM is activated on the command line or through its activation config.

Implicit profile activation

Profiles can be automatically triggered based on the detected state of the build environment. These triggers are specified via an <activation> section in the profile itself. Currently, this detection is limited to JDK version matching, operating system matching or the presence/the value of a system property. The implicit profile activation always only refers to the container profile

JDK:

The following configuration will trigger the profile when the JDK's version starts with "1.4" (eg. "1.4.0_08", "1.4.2_07", "1.4"), in particular it won't be active for newer versions like "1.8" or "11":

```
<profiles>
 <profile>
   <activation>
     <jdk>1.4</jdk>
   </activation>
   ...
 </profile>
</profiles>
```

Ranges can also be used. Range values must start with either [or (. Otherwise, the value is interpreted as a prefix. The following honours versions 1.3, 1.4 and 1.5.

```
<profiles>
 <profile>
   <activation>
     <jdk>[1.3,1.6)</jdk>
   </activation>
   ...
 </profile>
</profiles>
```

OS

This next one will activate based on the detected operating system.

```
<profiles>
 <profile>
   <activation>
     <os>
       <name>Windows XP</name>
       <family>Windows</family>
       <arch>x86</arch>
       <version>5.1.2600</version>
     </os>
   </activation>
   ...
 </profile>
</profiles>
```

The values are interpreted as Strings and are matched against the Java System properties **os.name, os.arch, os.version** and the family being derived from those.
Each value can be prefixed with ! to negate the matching.

Files

This example will trigger the profile when the generated file is missing.

```
<profiles>
 <profile>
   <activation>
     <file>

<missing>target/generated-sources/axistools/wsdl2java/org/apache/maven</missing>
     </file>
   </activation>
   ...
 </profile>
</profiles>
```

The tags <exists> and <missing> can be interpolated. Supported variables are system properties like ${user.home} and environment variables like ${env.HOME}. Please note that properties and values defined in the POM itself are not available for interpolation here, e.g. the above example activator cannot use ${project.build.directory} but needs to hard-code the path target.