Maven Fundamentals

Frida Cano Falcón

Java Backend Academy MTY
August 2024

Week 2

August 23, 2024

Introduction

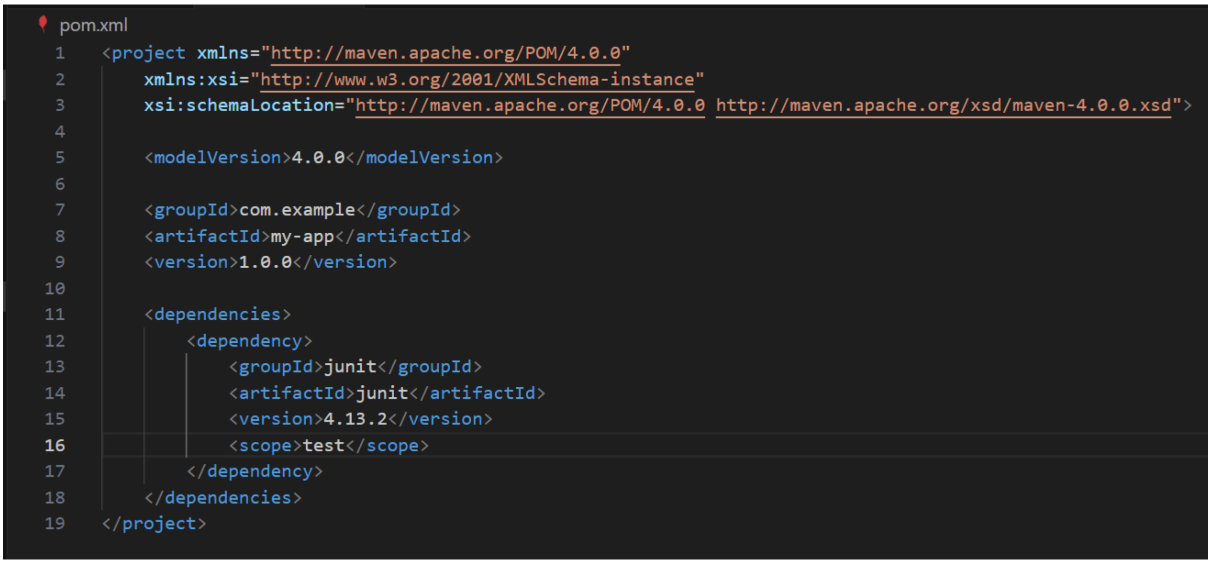
Maven is a build automation tool primarily used for Java projects. It simplifies the build process by managing dependencies, compiling source code, running tests, and packaging applications. One of Maven's most powerful features is its ability to manage project dependencies automatically, making it a popular choice for developers working on large-scale applications. By defining a project's configuration in a single XML file known as the Project Object Model (POM), Maven simplifies and standardizes the build process, allowing developers to focus more on coding and less on build management.

Project Object Model (POM)

The Project Object Model (POM) is the fundamental unit of Maven, representing the configuration file for a Maven project. The POM file, typically named *pom.xml*, defines the project’s structure, dependencies, and build configurations.

There are key elements in a POM file:

- *<groupId>*: Defines the unique base name of the project, typically following the reverse domain name convention (e.g., com.example.project).
- *<artifactId>*: Specifies the name of the project or module, which is used to identify the build output (e.g., my-app).
- *<version>*: Indicates the current version of the project (e.g., 1.0.0).
- *<dependencies>*: Lists all external libraries that the project depends on, with each dependency defined by its groupId, artifactId, and version.



```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>my-app</artifactId>
    <version>1.0.0</version>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

Example of a pom.xml

Dependency Management

Maven’s dependency management is one of its most powerful features. It allows developers to automatically download and include external libraries required for their projects. This is done by declaring dependencies in the POM file, and Maven handles the rest, including downloading the appropriate versions and managing any transitive dependencies.

Repositories

- *Local Repository*: A directory on your local machine where Maven stores downloaded dependencies.
- *Central Repository*: The default repository from which Maven downloads dependencies (hosted by Maven).
- *Remote Repository*: Additional repositories that can be specified in the POM file to download dependencies not available in the central repository.

Build Lifecycle

Maven's build lifecycle is a sequence of phases that define the steps required to build, test, and package a project. There are three primary lifecycles:

- **Default Lifecycle**: Handles project deployment. Key phases include:
  - **validate**: Validates the project is correct and all necessary information is available.
  - **compile**: Compiles the source code.
  - **test**: Runs tests using a suitable testing framework.
  - **package**: Packages the compiled code (e.g., into a JAR file).
  - **verify**: Runs checks to ensure the package is valid and meets quality standards.
  - **install**: Installs the package into the local repository.
  - **deploy**: Copies the package to a remote repository for sharing with other developers or projects.
- **Clean Lifecycle**: Handles project cleaning.
  - **clean**: Removes files generated by the previous build.
- **Site Lifecycle**: Handles the creation of project documentation.
  - **site**: Generates project documentation.

Customizing the Build Lifecycle

Maven allows developers to customize the build lifecycle by specifying additional goals or plugins in the POM file. This provides flexibility in handling unique build requirements for different projects.

Project Inheritance and Aggregation

Maven supports project inheritance and aggregation, which are essential for managing multi-module projects.

Project Inheritance

Inheritance allows a project to inherit configurations from a parent POM. This is useful for setting up consistent configurations across multiple projects.

- **Parent POM**: Defines common configurations shared by child projects.
- **Child POM**: Inherits configurations from the parent POM but can also override them.

Project Aggregation

Aggregation is used to manage multiple modules within a single project. The parent POM aggregates all modules, allowing them to be built and managed together.

Conclusion

Maven is an indispensable tool for Java developers, offering a robust, standardized approach to project management and build automation. By centralizing project configurations in the POM file, Maven simplifies dependency management, automates the build process, and promotes consistency across development environments. Its support for plugins, project inheritance, and aggregation further enhances its flexibility, making it an ideal choice for projects of all sizes. Adopting Maven can significantly streamline development workflows, allowing developers to focus on writing code rather than managing builds.

References

- Apache Cassandra. (n.d.). Wikipedia. Retrieved August 15, 2024, from https://es.wikipedia.org/wiki/Apache_Cassandra

- JRebel. (n.d.). What is Maven?. Retrieved August 15, 2024, from https://www.jrebel.com/blog/what-is-maven

- Automateinfra. (2022, December 1). How to setup Maven in Jenkins. Retrieved August 15, 2024, from https://automateinfra.com/2022/12/01/how-to-setup-maven-in-jenkins /

- Udemy. (n.d.). Apache Maven: Construcción de Proyectos de Cero a Avanzado. Retrieved August 15, 2024, from https://www.udemy.com/course/apache-maven-construccion-de-proyectos-de-cero-a-avanzado/?couponCode=SKILLS4SALEA