Maven Commands

In this lesson, we'll look at Maven.

WE'LL COVER THE FOLLOWING

Maven is a build tool. The configuration for a project is stored in a `pom.xml` file. http://start.spring.io/ offers a simple possibility for generating new Spring Boot projects with suitable `pom.xml` files. To do so, the user has to enter some settings on the web page. Then the web page creates the project with a `pom.xml`.

Maven can combine multiple projects to a multi module project. In this case the definitions meant to apply to all modules are stored in a single `pom.xml`. All modules reference this `pom.xml`.

The `pom.xml` is stored in a directory, and the modules are saved in a subdirectory. They have their own `pom.xml` with the information specific to the respective module.

On the one hand, Maven can be started for the entire project in the directory containing the `pom.xml`. In this case Maven builds the entire project with all its modules. On the other hand, Maven can be started in the directory for a specific module. Then the Maven commands relate to this one module.

Directories

A Maven module has a fixed file structure.

- The directory `main` contains all files of the module.

- The directory `test` comprises files that are only needed for tests.

Beneath these directories there is a standardized directory structure.

- `java` contains the Java code.

- `resources` contains resources that are adopted into the application.

Maven wrapper

After the installation, Maven can be used by starting `mvn`. The rest of this appendix assumes such a Maven installation.

Instead of installing Maven, the Maven Wrapper can be used. In that case, a script is created that downloads and installs Maven. Then `./mvnw` (Linux, macOS) or `./mvnw.cmd` (Windows) must be used to execute Maven. All examples for the book include a Maven wrapper, so this approach can be used, too.

Commands

The most important commands for Maven are:

- `mvn package` downloads all dependencies from the Internet, compiles the code, executes the tests, and creates an executable JAR file. The result is provided in the sub directory `target` of the respective module. `mvn package -Dmaven.test.skip=true` does not execute the tests. `mvn package -DdownloadSources=true -DdownloadJavadocs=true` downloads the source code and the JavaDoc of the dependent libraries from the Internet. The JavaDoc contains a description of the API. Development environments can display JavaDoc and the library source code for the user.

- `mvn test` compiles and tests the code but does not create a JAR.

- `mvn install` adds a step to `mvn package` by copying the JAR files into the local repository in the `.m2` directory in the home directory of the user. This allows other projects and modules to declare the module as a dependency in `pom.xml`. However, this is not necessary for the examples so `mvn package` is enough.

- `mvn clean` deletes all results of preceding builds. Maven commands can be combined. `mvn clean package` compiles everything after the results of the old builds have been deleted.

The result of the Maven build is a JAR (Java Archive). The JAR contains all components of the application including the libraries. Java directly supports this file format. Therefore, it is possible to start a microservice with `java -jar target/microservice-order-0.0.1-SNAPSHOT.jar`.

Troubleshooting

When `mvn package` does not work:

- Try out the `mvn clean package` to delete all old build results.
- Use the `mvn clean package package -Dmaven.test.skip=true` in order to skip the tests.
- The tests might fail because there is still a server running on your machine on port 8080. Make sure this is not the case.

In the next lesson, we'll look at how to install Docker and some basic Docker commands!