Phases of a Maven project

5.1 List of project phases

Finally, we got to the assembly of the project. And then you will be a little surprised. Well, or strongly, as it turns out. Maven has revised its approach to building a project. And now you will be convinced of it.

The entire assembly of the project was divided into phases, the description of which I will give in the table below:

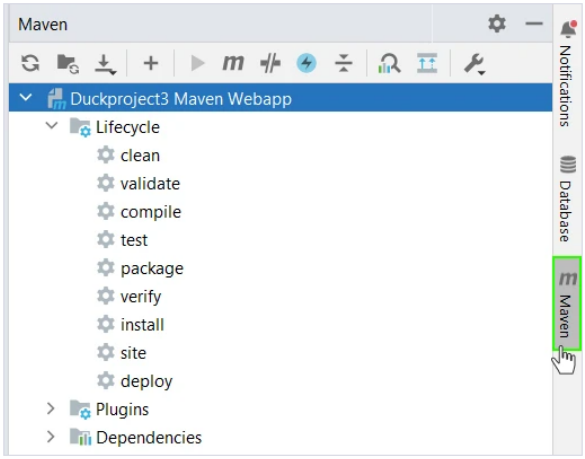
order	Phase	
1	**validate**	checks the correctness of meta-information about the project
2	**compile**	compiles sources
3	**test**	runs the class tests from the previous step
4	**package**	packs the compiled classes into a new artifact: jar, war, zip, ...
5	**verify**	checks the correctness of the artifact and the satisfaction of quality requirements
6	**install**	puts the artifact in the local repository
7	**deploy**	uploads an artifact to a production server or remote repository

At the same time, the steps are clearly sequential . If you tell Maven to run the package command, it will first run the validate, compile, test phases and only then package

In principle, there is nothing new here, except that there are separate phases for quality control: validate, test, verify. And as many as two phases for deploying the assembly - install and deploy.

To start a specific phase, it is enough to write the **maven phase** command . For example, to build, you need to run the **maven package** command . Etc.

Intellij IDEA is great at working with these phases and it has a special menu **on the right** for these purposes:



Here, in addition to the well-known phases, IDEA displays 2 more commands: **clean** and **site** . clean is used to completely clear the target folder, and site can create project documentation.

5.2 Building a project

If you want to compile the project, then you need to run the compile phase. This can be done using the command line: **mvn compile** or through the IDEA interface by clicking on the **compile** item .

After that, Maven will start building the project and you will see a log of the build process similar to this:

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 0.742 s
[INFO] Finished at: 2016-09-19T22:41:26+04:00
[INFO] Final Memory: 7M/18M
[INFO] ------------------------------------------------------------------------
```

If something went wrong, the log will look like this:

```
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-compiler-plugin:3.8.0:compile (default-compile
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible sliutions, please read the flilowing articles:
[ERROR] [Help 1]
http://cwiki.apache.org/confluence/display/MAVEN/MojoExecutionException
```

There will be a lot of useful information in the log, over time you will learn to understand and appreciate it.

5.3 Work cycles

All maven commands are divided into three groups - lifecycles. They are called **lifecycles** because they specify the order of the phases that run during a build or a particular lifecycle because not all Maven activities are builds.

There are three life cycles:

- **clean;**
- **default;**
- **site.**

And each of them has its own phase order. Clean has the shortest:

1. **pre clean;**
2. **clean;**
3. **post clean.**

Hidden additional pre-clean and post-clean phases have been added so that the cleanup scenario can be made more flexible.

Then comes the site life cycle , which, as you already know, is designed to automatically generate project documentation. It consists of four phases:

1. **pre-site**
2. **site;**
3. **post-site;**
4. **site-deploy.**

Standard life cycles can be enhanced with functionality using **Maven plugins** . We will talk about this later, because this is a very interesting topic that deserves a separate lecture.

And the default script has the longest list of phases:

1. **validate;**
2. **generate-sources;**
3. **process-sources;**
4. **generate-resources;**
5. **process-resources;**
6. **compile;**

7. **process-test-sources;**
8. **process-test-resources;**
9. **test compile;**
10. **test;**
11. **package;**
12. **install;**
13. **deploy.**

There are all the same phases that you already know about, but a few more optional ones have been added.

First, a popular stage in large projects is **generate-sources** : generating Java code based on XML, for example. And a pair of **process-sources** , which does something with this code.

Secondly, the generation of resources is **generate-resources** and its paired **process resources** method . You will often see some activities tied to these phases in large projects.

And finally, testing. It has three additional optional phases that help make running the test phase as flexible as possible: process-test-sources, process-test-resources, test-compile.