Guide to Working with Multiple Modules in Maven 4

(If you're working with Maven 3, please refer to the Maven 3 edition of this guide (./guide-multiple-modules.html))

As seen in the introduction to the POM, Maven supports project aggregation in addition to project inheritance. This section outlines how Maven processes projects with multiple modules, and how you can work with them more effectively.

The Reactor

The mechanism in Maven that handles multi-module projects is referred to as the *reactor*. This part of the Maven core does the following:

- Collects all the available modules to build
- Sorts the modules into the correct build order
- Selects which modules to build
- Builds the selected modules in order

Collecting Modules

Module collection starts from one aggregator project. That project defines the modules of which it consists using the `<modules>` element. This is a recursive process, so aggregators can have child modules which are aggregators themselves.

For this process to work, it does not matter which POM you start with. Maven attempts to find the root of a multi-module project, by traversing upwards in the directory structure until it finds a POM with a `.mvn` sibling directory. This allows Maven to resolve dependencies on modules from the same multi-module project, regardless of the location of the starting POM. When Maven fails to find the root, it assumes that the starting POM is the root. For consistent behaviour, create a `.mvn` directory in the root directory of the project.

There are two ways to point at a starting POM. By default, Maven reads the `pom.xml` file in the working directory. Using `-f`, you can point to another POM.

Sorting Modules

Because modules within a multi-module build can depend on each other, it is important that the reactor sorts all the projects in a way that guarantees any project is built before it is required.

The following relationships are honoured when sorting projects:

- a project dependency on another module in the build
- a plugin declaration where the plugin is another module in the build
- a plugin dependency on another module in the build
- a build extension declaration on another module in the build
- the order declared in the `<modules>` element (if no other rule applies)

Note that only “instantiated” references are used - `dependencyManagement` and `pluginManagement` elements do not cause a change to the reactor sort order.

Selecting Modules

By default, Maven builds all modules it has collected. However, you can select a subset of these modules to build using command line flags. These flags come in three categories:

- Inclusion and exclusion
- Relationships between modules
- Dealing with failures

This section ends with how these flags relate to each other.

Inclusion and exclusion

Using `--projects` you can specify which modules to build. You can do this by specifying a comma-delimited list of project selectors. A project selector is a string that is composed of the `groupId:artifactId`, only `:artifactId` or the relative path to a module.

A module can be selected (default), or excluded from the build. You exclude a module by prefixing the selector with a `!` or `-`. To explicitly select a module, prefix it with a `+`.

When a selector does not resolve to an existing module, Maven fails the build. You can prevent this by adding the `?` prefix. This prefix should always go after the other prefixes.

Relationships between modules

Modules inside a project can have two types of relationships: parent/child and dependency/dependent.

When selecting a parent (aggregator), Maven automatically selects the child modules as well. Similarly, Maven excludes child modules of an excluded parent (aggregator). To prevent this recursive behaviour, combine `--projects` with `--non-recursive`.

Maven knows about the dependencies between modules inside the multi-module project. Using `--also-make`, Maven includes all dependencies of the selected projects in the build. Similarly, `--also-make-dependents` lets Maven include all modules which are dependent on the selected projects.

Dealing with failures

There are several ways to customize how the reactor deals with failures. `--fail-at-end` fails the build after building as many modules as possible. In this case, modules that do not depend on a failed module, will still be built. `--fail-fast`, in contrast, fails the build as soon as one module has failed. This is the default behaviour. `--fail-never` ignores build failures.

When a build has failed, and you want to start it again, you can skip building the modules that were previously built successfully using `--resume`. To resume a build from a specific module, you can use `--resume-from <selector>`.

Bringing it together

As said, Maven includes all modules in the reactor, even when the starting POM is not the root of the project. The reactor then selects which modules to build using the following steps:

1. Reduce the full list of modules to the module of the starting POM and its children.
2. Further reduce this list to all modules included with `--projects` and, if `--resume` is given, the remaining modules of a previously failed build.
3. Further reduce this list by removing all modules that would have been built before the module selected by `--resume-from`.
4. Finally, further reduce this list by removing all modules that are excluded with `--projects` (using the `!` or `-` prefixes).

Each of the steps 1, 2 and 3 honor the `--also-make` and `--also-make-dependents` flags, if they are given.

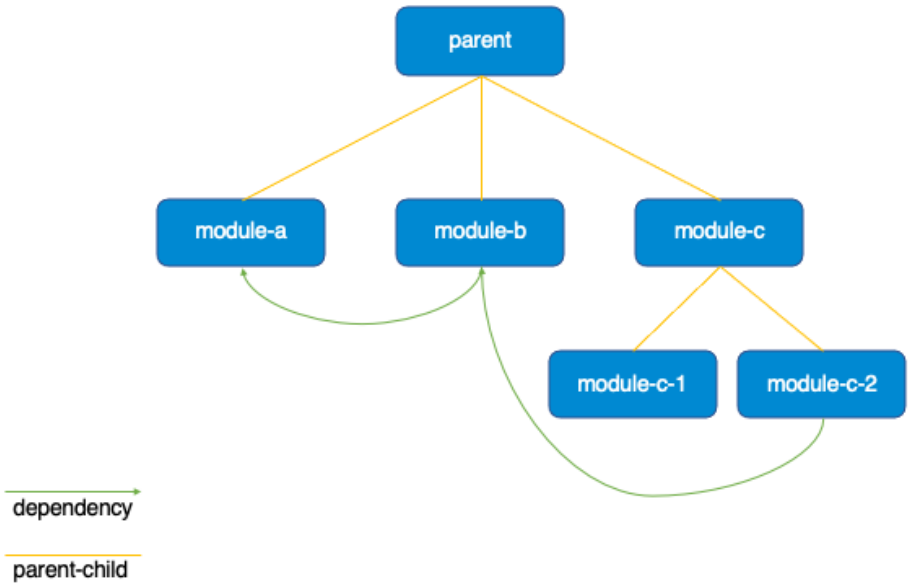
Command Line Options

No special configuration is required to take advantage of the reactor, however it is possible to customize its behavior.

The following command line switches are available:

Long	Short	Summary	Details
`--file`	`-f`	Selects an alternative POM file or directory containing a POM file.	Collecting Modules
`--non-recursive`	`-N`	Ignores any child modules that may be present in the starting POM. When combined with `-pl`, ignores the children of selected modules.	Collecting Modules and Relationships between modules (relationships-between-modules)
`--projects`	`-pl`	Specifies the modules to include or exclude from a build.	Inclusion and exclusion
`--also-make`	`-am`	Builds the specified modules, and any of their dependencies in the reactor.	Relationships between modules
`--also-make-dependents`	`-amd`	Builds the specified modules, and any that depend on them.	Relationships between modules
`--resume-from`	`-rf`	Resumes a reactor from the specified project (e.g. when it fails in the middle).	Dealing with failures
`--resume`	`-r`	Resumes a reactor from the module where the previous build failed.	Dealing with failures
`--fail-fast`	`-ff`	Stops the overall build immediately whenever a module build fails. This is the default behavior.	Dealing with failures
`--fail-at-end`	`-fae`	Continues the rest of the reactor if a particular module build fails and report all failed modules at the end instead.	Dealing with failures
`--fail-never`	`-fn`	Never fails the build, regardless of the project result.	Dealing with failures

Differences between Maven 3 and 4



The table below illustrates multiple scenarios which have changed between Maven 3 and 4. They assume a project structure as depicted above.

Scenario	Outcome (in order)	Maven 3	Maven 4
Build an aggregator and its children	module-c, module-c-1, module-c-2	`mvn compile -pl module-c, module-c-1, module-c-2`	`mvn compile -pl module-c`
Build an aggregator and ignore its children	module-c, module-c-1, module-c-2	`mvn compile -pl module-c`	`mvn compile -pl module-c -N`
Also make dependencies outside of current scope	parent, module-a, module-b, module-c, module-c-2	N/A	`cd module-c/module-c-2 && mvn compile -am`
Also make dependents outside of current scope	module-a, module-b, module-c-2	N/A	`cd module-a && mvn compile -amd`

Scenario	Outcome (in order)	Maven 3	Maven 4
Resuming from a module and build all dependencies	parent, module-a, module-b, module-c, module-c-2	N/A	`mvn compile -rf :module-c-2 -am` or `mvn compile -r -am`
Run specific goal on one submodule with dependencies from project	module-c-2	`mvn install && mvn jetty:run -f module-c/module-c-2`	`mvn compile && mvn jetty:run -f module-c/module-c-2`

More information

- Chapter 6. A Multi-module Project (Maven by Example) (http://books.sonatype.com/mvnex-book/reference/multimodule.html) - does not include Maven 4 changes!
