Lifecycle, Phases, goals, and Plugins?!

Maven Lifecycle

Maven is a build automation tool at heart. It has three built-in lifecycle that clearly define how the project will be build. There are three built-in build lifecycles:

1. default: handles the actual project compilation and deployment
2. clean: handles project cleaning, removing the target folder
3. site: handles the creation of project site documentation

You "can" add more lifecycle but then that go against the point of creating those three built-in build lifecycles. Those three built-in ones should be the standard of all the projects, you can create your own plugins and then hook it into the phases within the lifecycle.

Adding more lifecycle would take away the ease of reproducing the build and maintaining it.

Each of the lifecycle consists of phases. So you have life cycles like **default**, it consists of **phases**, and each **phases** consists of **goals**.

Phases in default

Not going to list them all but here are some of the phases in order:

- validate
- compile
- test
- package
- verify
- install
- deploy

When you execute a phase all the goals tied to the goal will be executed. In addition, when you execute a phase all the previous steps will be executed. For example, executing `mvn install` will execute, validate, compile, test, package, verify, and install in order.

> There is no way to execute the "lifecycle", to execute the lifecycle you would just specify the last phase to run through the entire phases. In this case running `mvn deploy` will in a sense

> run the entire lifecycle

Maven phases

Let's clear something up, all the available goals are provided by plugins. The compile phase's goals are provided by the "maven-compiler-plugin", the package phase's goal are provided by the "maven-jar-plugin", you get the idea.

> Official maven plugin like compile, package have standard name of "maven-<name>-plugin", whereas non-official maven plugin have "<name>-maven-plugin" naming convention.
>
> The implication is that when you want to invoke the goal directly without invoking the phase if you are using official maven plugin is just "<name>:<goal>" for example, `mvn compiler:compile`. You don't need to do `mvn maven-compiler-plugin:compile`. However, unofficial maven plugins then you would need to provide the entire plugin name to invoke the goal.

A build phase doesn't necessarily need to have any goal bound to it. If it has no goal bound to it, it will simply not execute any goal.

However, the same goal can be bounded to multiple different build phases, so that goal will just be executed multiple times when the phase is executed.

Furthermore, not all the goals from a plugin are bind a phase. For example, the `maven-compiler-plguin` have two goals, `compile / testCompile`. However, only the `compile` goal is bound to the compile phase. You can add `testCompile` to the phase by adding an `<executions>` section in your plugin tag, will go more over that in the next section.

Maven goals (Mojos)

Goals are the actual task that are executed, they help building and manage the project. Phases and lifecycle are essentially just abstraction of goals.

Goals may or may not not bound to a phase, those that are tied to a phase will be executed when you run the phase. Those are not bound to a phase, you can invoke them directly. By using the syntax discussed above.

To bound a plugin's goal to a phase if they are not bound by default, or you would like to bound to a different phase, i.e. adding jar plugin's jar goal to the compile phase. Whenever you run `mvn copmile` it will also package it as a jar, as oppose to whenever you run `mvn package` if you don't want to run test phase before it. What you would do is to add a `<executions>` section in your pom.xml file.

Configuring plugin execution

If you would like to add a goal that isn't by default mapped to one of the default phases for your plugin, or you would like to add it to different phase then you would need to append a `<executions>` section for your plugin.

For example, the `jar:jar` goal which build a JAR from the current project is by default bind to the `package` but if you want to run the goal whenever `compile` plugin is run then you can add the following section to your `pom.xml`

```
          <executions>
            <execution>
              <id>compile-jar</id>
              <phase>compile</phase>
              <goals>
                <goal>jar</goal>
              </goals>
            </execution>
          </executions>
```

What this section does is that it will bind the goal `jar:jar` remember goals can be bind to multiple phases, to the `compile` phase, with the id `compile-jar`. The id is there so that when you invoke the `mvn` command, the execution output will be in the form of

```
<plugin-name>:<plugin-version>:<phase> (<execution-id>)
```

So in this case, as part of the `compile` phase, it will be running `maven-jar-plugin:2.2.2:jar (compile-jar)`, so right after you compile your code, you will pack it into a jar.

Scenarios for executions

```
         <plugin>
              <groupId>org.apache.maven.plugins</groupId>
              <artifactId>maven-docck-plugin</artifactId>
              <version>1.0</version>
              <executions>
                  <execution>
                      <id>some-other-other-id</id> <!-- No goal for execution is defined -->
                      <phase>pre-site</phase>
                  </execution>
                  <execution>
                      <phase>pre-site</phase>   <!-- No id for execution is defined -->
                      <goals>
```

```
                        <goal>check</goal>
                    </goals>
                </execution>
                <execution>
                    <id>some-id</id>          <!-- No phase for execution is defined --
>
                    <goals>
                        <goal>check</goal>
                    </goals>
                </execution>
                <execution>
                    <id>some-other-id</id>     <!-- Both id and phase defined -->
                    <phase>pre-site</phase>
                    <goals>
                        <goal>check</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
```

Results in sample output as following

```
[INFO] --- maven-docck-plugin:1.0:check (default) @ MavenJavaApplication ---
[INFO] Skipping unsupported project: MavenJavaApplication
[INFO] No documentation errors were found.
[INFO]
[INFO] --- maven-docck-plugin:1.0:check (some-other-id) @ MavenJavaApplication ---
[INFO] Skipping unsupported project: MavenJavaApplication
[INFO] No documentation errors were found.
```

Execution 1: No goal for execution is defined

If no goal is specified then that execution is simply ignored. If a phase have no goal then it will just be ignored, because there is nothing to execute.

Execution 2: No id for execution is defined

If no id is defined for a specified execution, but goal and phase is specified then it will simply execute with the id of `default`.

Execution 3: No phase for execution is defined

If no phase is defined, then that execution of plugin goal is simply not run.

Execution 4: id, phase, and goal is defined

Then it will get ran during the specified phase with the specified id

`default-cli` vs `default-<goalName>`

The goals that are invoked directly from the command line for example `mvn jar:jar` (invoking the jar goal from the `maven-jar-plugin`) it will have the execution id of `default-cli` because you ran it in the CLI.

This ID is provided so that you can provide further configuration if needed for the plugin that's ran from CLI, by using the execution id of `default-cli` in your `pom.xml` file.

Plugin goals that are mapped to the default lifecycle like some of the official maven plugin such as `maven-jar-plugin` will have their execution id to be `default-<goalName>` for example:

```
          <artifactId>maven-clean-plugin</artifactId>
          <version>2.5</version>
          <executions>
            <execution>
              <id>default-clean</id>
              <phase>clean</phase>
              <goals>
                <goal>clean</goal>
              </goals>
            </execution>
          </executions>
```

The `clean` plugin's default goal mapping maps the `clean` goal to the `clean` phase with the execution id of `default-clean`.

Again this is there so that you can further configure these official built-in goals, it is a naming convention after all.

You can find out about this by looking at the effective pom via `mvn help:effective-pom.`

Revision #3
Created 11 May 2023 18:01:27 by Tamarine
Updated 13 May 2023 00:54:48 by Tamarine