Maven Compiler Plugin

Last modified: July 3, 2020

by Nguyen Nam Thai (https://www.baeldung.com/author/namthai-nguyen)

Maven (https://www.baeldung.com/category/maven)

Maven Basics (https://www.baeldung.com/tag/maven-basics)

This article is part of a series:

1. Overview

This quick tutorial introduces the *compiler* plugin, one of the core plugins of the Maven build tool.

For an overview of the other core plugins, refer to this article (/core-maven-plugins).

2. Plugin Goals

The *compiler* plugin is used to compile the source code of a Maven project. This plugin has two goals, which are already bound to specific phases of the default lifecycle:

- *compile* – compile main source files
- *testCompile* – compile test source files

Here's the *compiler* plugin in the POM:

```
<plugin>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.7.0</version>
    <configuration>
        ...
    </configuration>
</plugin>
```

We can find the latest version of this plugin here (https://search.maven.org/classic/#search%7Cga%7C1%7Cg%3A%22org.apache.maven.plugins%22%20AND%20a%3A%22maven-compiler-plugin%22).

3. Configuration

By default, the *compiler* plugin compiles source code compatible with Java 5, and the generated classes also work with Java 5 regardless of the JDK in use. We can modify these settings in the *configuration* element:

```
<configuration>
    <source>1.8</source>
    <target>1.8</target>
    <-- other customizations -->
</configuration>
```

For convenience, we can set the Java version as properties of the POM:

```
<properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

Sometimes we want to pass arguments to the *javac* (/javac) compiler. This is where the *compilerArgs* parameter comes in handy.

For instance, we can specify the following configuration for the compiler to warn about unchecked operations:

```
<configuration>
    <!-- other configuration -->
    <compilerArgs>
        <arg>-Xlint:unchecked</arg>
    </compilerArgs>
</configuration>
```

When compiling this class:

```
public class Data {
    List<String> textList = new ArrayList();

    public void addText(String text) {
        textList.add(text);
    }

    public List getTextList() {
        return this.textList;
    }
}
```

we'll see an unchecked warning on the console:

```
[WARNING] ... Data.java:[7,29] unchecked conversion
  required: java.util.List<java.lang.String>
  found:    java.util.ArrayList
```

As both goals of the *compiler* plugin are automatically bound to phases in the Maven default lifecycle, we can execute these goals with the commands *mvn compile* and *mvn test-compile*.

4. Java 9 Updates

4.1. Configuration

Until Java 8, we used the version number as 1.*x* where *x* represents Java's version, like 1.8 for Java 8.

For Java 9 and above, we can just use the version number directly:

```
<configuration>
    <source>9</source>
    <target>9</target>
</configuration>
```

Similarly, we can define the version using *properties* as:

```
<properties>
    <maven.compiler.source>9</maven.compiler.source>
    <maven.compiler.target>9</maven.compiler.target>
</properties>
```

Maven added its support for Java 9 in *3.5.0*, so we'll need at least that version. We'll also need at least *3.8.0* of the *maven-compiler-plugin* (https://search.maven.org/classic/#search%7Cga%7C1%7Cg%3A%22org.apache.maven.plugins%22%20AND%20a%3A%22maven-compiler-plugin%22):

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.0</version>
            <configuration>
                <source>9</source>
                <target>9</target>
            </configuration>
        </plugin>
    </plugins>
</build>
```

4.2. Build

Now it's time to test our configuration.

First, let's create a *MavenCompilerPlugin* class in which we're importing a package from another module (/java-9-modularity).

A simple one is *javax.xml.XMLConstants.XML_NS_PREFIX:*

```
public class MavenCompilerPlugin {
    public static void main(String[] args) {
        System.out.println("The XML namespace prefix is: "
          + XML_NS_PREFIX);
    }
}
```

Next, let's compile it:

```
mvn -q clean compile exec:java
  -Dexec.mainClass="com.baeldung.maven.java9.MavenCompilerPlugin"
```

When using Java 9 defaults, though, we'll get an error:

```
[ERROR] COMPILATION ERROR :
[ERROR] .../MavenCompilerPlugin.java:[3,20]
  package javax.xml is not visible
  (package javax.xml is declared in module java.xml,
  but module com.baeldung.maven.java9 does not read it)
[ERROR] .../MavenCompilerPlugin.java:[3,1]
  static import only from classes and interfaces
[ERROR] .../MavenCompilerPlugin.java:[7,62]
  cannot find symbol
symbol:   variable XML_NS_PREFIX
location: class com.baeldung.maven.java9.MavenCompilerPlugin
```

The error comes from the fact that this package is in a separate module that we haven't included yet in our build.

The simplest way to solve this is by creating *a module-info.java* class and indicating that we require the *java.xml* module:

```
module com.baeldung.maven.java9 {
    requires java.xml;
}
```

Now we can try again:

```
mvn -q clean compile exec:java
  -Dexec.mainClass="com.baeldung.maven.java9.MavenCompilerPlugin"
```

And our output will be:

```
The XML namespace prefix is: xml
```

5. Conclusion

In this article, we went over the *compiler* plugin and described how to use it. We also learned about Maven's support for Java 9.

The complete source code for this tutorial can be found over on GitHub (https://github.com/eugenp/tutorials/tree/master/maven-modules/compiler-plugin-java-9).

Next »

Quick Guide to the Maven Install Plugin (/maven-install-plugin)

« Previous
Maven Resources Plugin (/maven-resources-plugin)

Comments are closed on this article!

COURSES

ALL COURSES (/ALL-COURSES)
ALL BULK COURSES (/ALL-BULK-COURSES)
ALL BULK TEAM COURSES (/ALL-BULK-TEAM-COURSES)
THE COURSES PLATFORM (HTTPS://COURSES.BAELDUNG.COM)

SERIES

JAVA "BACK TO BASICS" TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
APACHE HTTPCLIENT TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)
SPRING REACTIVE TUTORIALS (/SPRING-REACTIVE-GUIDE)

ABOUT

ABOUT BAELDUNG (/ABOUT)
THE FULL ARCHIVE (/FULL_ARCHIVE)
EDITORS (/EDITORS)
JOBS (/TAG/ACTIVE-JOB/)
OUR PARTNERS (/PARTNERS)
PARTNER WITH BAELDUNG (/ADVERTISE)

TERMS OF SERVICE (/TERMS-OF-SERVICE)
PRIVACY POLICY (/PRIVACY-POLICY)
COMPANY INFO (/BAELDUNG-COMPANY-INFO)
CONTACT (/CONTACT)