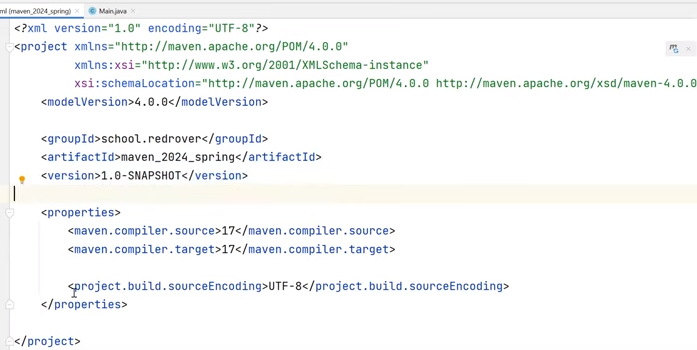
Это основной конфигурационный файл в проекте, который описывает его структуру, зависимости и настройки. Можно сказать, что это главная сила всего Maven.

В POM.xml мы используем xml формат.

Maven - знает что такое (теги) properties, version, artifactId, но если написать тег <Dima\_Rusak> </Dima\_Rusak> то с точки зрения xml формата мы всё сделали верно, у нас есть открывающий и закрывающий тег. Но Maven такой тег не понимает, у него есть свой набор тегов и атрибутов с которыми он работает и которые он знает и умеет обрабатывать. Поэтому хоть maven и работает с xml нельзя писать всё что захочется.



1. Корневой элемент ( <project> ).

Корневой элемент <project>, в котором прописана схема облегчающая редактирование и проверку, и версия POM.

Пример

<?xml version="1.0" encoding="UTF-8"?>

**<project xmlns="http://maven.apache.org/POM/4.0.0"**

**xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**

**xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">**

**...**

**</project>**

Внутри тега project содержится основная и обязательная информация о проекте:

Пример:

**<groupId>com.examclouds</groupId>**

**<artifactId>courses</artifactId>**

**<version>1.0-SNAPSHOT</version>**

В Maven каждый проект идентифицируется парой groupId, artifactId.

Во избежание конфликта имён, groupId - наименование организации или подразделения и обычно действуют такие же правила, как и при именовании пакетов в Java - записывают доменное имя организации или сайта проекта.

artifactId - название проекта.

Внутри тега version хранится версия проекта.

Тройкой groupId, artifactId, version (далее - GAV) можно однозначно идентифицировать jar файл приложения или библиотеки. Если состояние кода для проекта не зафиксировано, то в конце к имени версии добавляется "-SNAPSHOT" что обозначает, что версия в разработке и результирующий jar файл может меняться.

**<properties>**

<**maven.compiler.source**> - какой версией JAVA мы написали исходный код.

<**maven.compiler.target**> - в какую версию JAVA скоомпилировать наш код по завершению написания

<**project.build.sourceEncoding**> - в какой кодировке наш исходный код

**</properties>**

properties — также можно здесь указывать переменные проекта.

**packaging**

<packaging> определяет какого типа файл будет создаваться как результат сборки. Возможные варианты pom, jar, war, ear.

Тег является необязательным. Если его нет, используется значение по умолчанию - jar.

**Описание проекта**

Также может добавлятся информация, которая не используется самим Maven, но нужна для программиста, чтобы понять, о чём этот проект:

Пример:

<name>powermock-core</name> название проекта для человека

<description>PowerMock core functionality.</description> Описание проекта

<url>http://www.powermock.org</url> сайт проекта

<**dependencies**>

следующая очень важная часть pom.xml - тут хранится список всех библиотек (зависимостей) которые используются в проекте. Каждая библиотека идентифицируется так же как и сам проект - тройкой groupId, artifactId, version (GAV). Объявление зависимостей заключено в теге <dependencies>...</dependencies>.

Кроме GAV при описании зависимости может присутствовать тег <scope>. Он задаёт, для чего библиотека используется.

Пример

<dependencies>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.39</version>

</dependency>

<dependency>

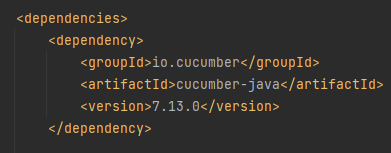
<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<version>1.4.196</version>

</dependency>

</dependencies>



**<build>**

<build> необязательный, так как существуют значения по умолчанию. Этот раздел содержит информацию по самой сборке:

где находятся исходные файлы,

где ресурсы,

какие плагины используются.

Пример

<build>

<**sourceDirectory**>src</sourceDirectory>

<resources>

<resource>

<directory>resources</directory>

</resource>

</resources>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>2.0.2</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

<encoding>UTF-8</encoding>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-checkstyle-plugin</artifactId>

<version>2.17</version>

<configuration>

<suppressionsLocation>suppressions.xml</suppressionsLocation>

</configuration>

</plugin>

</plugins>

</build>

<**sourceDirectory**> - определяет, откуда Maven будет брать файлы исходного кода. По умолчанию это src/main/java, но вы можете определить, где это вам удобно. Директория может быть только одна (без использования специальных плагинов).

<**resources**> и вложенные в неё теги <**recource**> определяют, одну или несколько директорий, где хранятся файлы ресурсов. Ресурсы в отличие от файлов исходного кода при сборке просто копируются. Директория по умолчанию src/main/resources.

<**outputDirectory**> - определяет, в какую директорию компилятор будет сохранять результаты компиляции - \*.class файлы. Значение по умолчанию - target/classes.

<**finalName**> - имя результирующего jar (war, ear ...) файла с соответствующим типу расширением, который создаётся на фазе package. Значение по умолчанию — artifactId-version.

**Plagins**

**Plagins -** are for to compile and run the project.

There are 2 types of plugins we have:

1)Maven compiler plugin

2)Maven surefire plugin

Располагать плагины перед депенденси в блоке <build> (это в моём примере, но может где то и по другому)

<build>

<pluginManagement>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.13.0</version>

<configuration>

<!-- put your configurations here -->

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.2.5</version>

</plugin>

</plugins>

</pluginManagement>

</build>

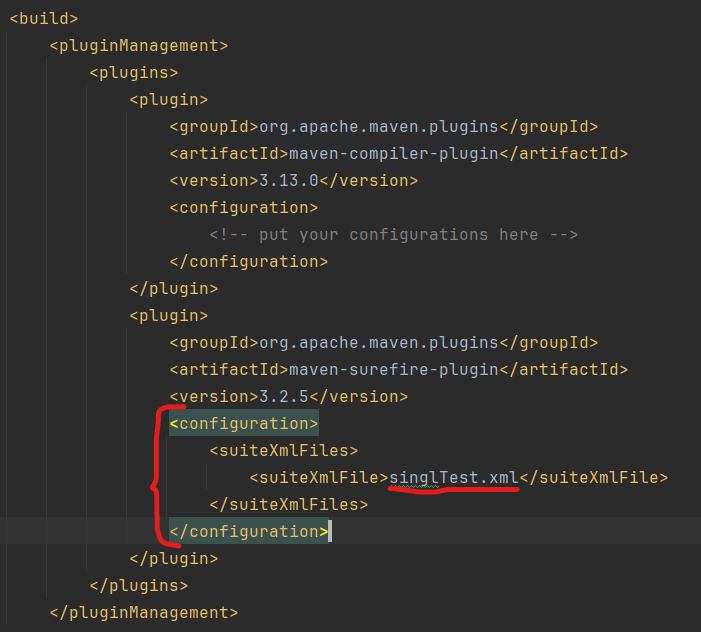
Плагины добовляют возможность выполнения проекта, т.е. мы можем добавить конфигурационный xml файл в плагин, он прописывается в теге

<configuration>

<!-- put your configurations here -->

</configuration>

Например добавим xml файл в surefire plugin



Теперь можно запустить пом ник. Для этого открыть терминал>**mvn clean test**

Можно добавить несколько xml конфигурационных файлов и все они по очереди выполнятся.

Maven плагины позволяют задать дополнительные действия, которые будут выполняться при сборке. Например, в приведённом примере добавлен плагин, который автоматически делает проверку кода на наличие "плохого" кода и потенциальных ошибок.