Мавен самый популярный, простой и решает необходимые задачи (зависимости, жизненный цикл сборки, поддержка плагина)

<https://habr.com/ru/post/77333/>

Maven сочетает в себе возможности ант (билдить и копировать файлы, проводить тесты), но кроме этого он помогает решать зависимости библиотек проекта, имеет вполне формальный lifecycle, имеет множество плагинов.

Жизненный цикл у мавена довольно ожидаемый:

* validate — проверяет корректность метаинформации о проекте
* compile — компилирует исходники
* test — прогоняет тесты классов из предыдущего шага
* package — упаковывает скомпилированые классы в удобноперемещаемый формат (jar или war, к примеру)
* integration-test — отправляет упаковынные классы в среду интеграционного тестирования и прогоняет тесты
* verify — проверяет корректность пакета и удовлетворение требованиям качества
* install — загоняет пакет в локальный репозиторий, откуда он (пакет) будет доступен для использования как зависимость в других проектах
* deploy — отправляет пакет на удаленный production сервер, откуда другие разработчики его могут получить и использовать  
    
  Особняком стоят фазы **clean** и **site**. Они не выполняются, если специально не указаны в строке запуска.
* clean

удаление всех созданных в процессе сборки артефактов: .class, .jar и др. файлов. В простейшем случае результат — просто удаление каталога target

* site

предназначена для создания документации (javadoc+сайт описания проекта)

Т. к . команда mvn понимает когда ему передают несколько фаз то для сборки проекта создания документации "с нуля" выполняют:

mvn clean package site

При этом все шаги последовательны. И если, к примеру, выполнить $ mvn package, то фактически будут выполнены шаги: validate, compile, test и package. Таким образом использовать мавен довольно просто. Написали код, выполнили mvn test и можно работать дальше, убедившись что код не содержит синтаксических и логических ошибок.

Отдельно стоит упомянуть о зависимостях. Maven конфигурируется файлом pom.xml, который может содежать блок <dependencies />. В нем описывается какие библиотеки нужны проекту для полноценного функционирования. На шаге validate мавен проверяет удовлетворены ли зависимости и если нет, то скачивает из удаленный репозиториев необходимые компоненты в репозиторий локальный. Так, если 10 проектов зависят от одной и той же библиотеки SomeSpecificOrm, то нам больше не надо подключать ее в 10 местах через svn:external, особенно, если библиотека занимает достаточно места, а достаточно указать ее в файле зависимостей и она будет браться из локального репозитория мавена. При указании зависимостей можно указывать не только имя библиотеки, но и ее версию — таким образом не возникнет проблем с обратной совместимостью.  
  
А теперь подойдем непосредственно к вопросу «чем мавен помогает в веб-разработке». Помимо компилирования, тестирования и пакетирования, мавен позволяет делать деплой проекта в tomcat посредством плагина tomcat-maven-plugin. Конфигурируется он в pom.xml

<https://m.habr.com/ru/post/77382/>

Вся структура проекта описывается в файле pom.xml (POM – Project Object Model), который должен находиться в корневой папке проекта. Ключевым понятием Maven является **артефакт** — это, по сути, любая библиотека, хранящаяся в репозитории. Это может быть какая-то зависимость или плагин.

**Зависимости** — это те библиотеки, которые непосредственно используются в вашем проекте для компиляции кода или его тестирования.  
  
**Плагины** же используются самим Maven'ом при сборке проекта или для каких-то других целей (деплоймент, создание файлов проекта для Eclipse и др.).maven-surefire-plugin у нас на проекте плагин который запускает тесты и генерирует отчёты по результатам их выполнения. По умолчанию отчёты сохраняются в ${basedir}/target/surefire-reports и находятся в двух форматах - txt и xml. maven-surefire-plugin содержит единственную цель surefire:test тесты можно писать используя как [JUnit](http://www.junit.org/) так и [TestNG](http://testng.org/).

*maven-compiler-plugin*. Он доступен по умолчанию, но практически в каждом проекте его приходится переобъявлять. В простейшем случае плагин позволяет определить версию java машины (JVM), для которой написан код приложения, и версию java для компиляции кода. Пример использования

<plugin>

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

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

<version>3.1</version>

<configuration>

<source>1.7</source> версия кода

<target>1.7</target> версия ява машины, на кот будет работать прога

<encoding>UTF-8</encoding>

</configuration></plugin>

В данном примере определена версия java-кода 1.7, на котором написана программа (source). Версия java машины, на которой будет работать программа, определена тегом <target>. В теге <encoding> указана кодировка исходного кода (UTF-8). По умолчанию используется версия java 1.3, а кодировка выбирается из операционной системы. Плагин позволяет указать путь к компилятору javac тегом <executable>.

Плагин *maven-compiler-plugin* имеет две цели :

* compiler:compile - компиляция исходников, по умолчанию связана с фазой compile;
* compiler:testCompile - компиляция тестов, по умолчанию связана с фазой test-compile.

Как правило, название артефакта состоит из названия группы, собственного названия и версии. К примеру Spring будет иметь вот такое название в среде Maven: org.springframework.spring:2.5.5. Последний домен означает всегда artifactId, все, что перед ним – groupId – хорошо это запомните!

В самом начале работы с Maven, пользователь непременно столкнется с таким понятием как архетип. **Архетип** — это некая стандартная компоновка файлов и каталогов в проектах различного рода (веб, swing-проекты и прочие). Другими словами, Maven знает, как обычно строятся проекты и в соответствии с архетипом создает структуру каталогов.

Поиск зависимостей в локальном репозитории (наш комп)

1. Если зависимости не обнаружены, происходит переход к шагу 2.
2. Поиск зависимостей в центральном репозитории (мавен рап). Если они не обнаружены и удалённый репозиторий определён (определяется самим разработчиком. Там могут храниться все необходимые зависимости.), то происходит переход к шагу 4.
3. Если удалённый репозиторий не определён, то процесс сборки прекращается и выводится сообщение об ошибке.
4. Поиск зависимостей на удалённом репозитории, если они найдены, то происходит их загрузка в локальный репозиторий, если нет – выводится сообщение об ошибке.

**Установка Maven**

Последнюю версию всегда можно скачать на [странице загрузки](http://maven.apache.org/download.html) на официальном сайте. Просто распаковываем архив в любую директорию. Далее необходимо создать переменную в Path, в которой необходимо указать путь к Maven. Заходим в Win + Pause – Дополнительно – Переменные среды – в верхнем окошке нажимаем Создать, вводим имя M2\_HOME и значение допустим “C:\apache-maven-2.2.1”. Далее там же создаем еще одну переменную M2 со значением %M2\_HOME%\bin. Так же убеждаемся, что есть переменная JAVA\_HOME с путем к JDK. Ее значение должно быть примерно таким «c:\Program Files\Java\jdk1.6.0\_10\». И наконец в том же окошке создаем/модифицируем переменную Path, в нее необходимо просто написать %M2%, чтобы наша папочка с исполняемым файлом Maven была видна из командной строки. Теперь необходимо проверить работоспособность нашей установки. Для этого заходим в командную строку и вводим команду

mvn –version

Должна появиться информация о версиях Maven, jre и операционной системе, что-то вроде:

Maven version: 2.2.1

Java version: 1.6.0\_10

OS name: "windows 2003" version: "5.2" arch: "x86" Family: "windows"

Maven создаст вам локальный репозиторий в вашей личной папке, например в каталоге C:\Documents and Settings\username\.m2\repository  
  
Все, Maven готов к работе, можно приступать к созданию приложения.

Полное руководство этого пиздеца  
<https://proselyte.net/tutorials/maven/>