**Сборка –** процесс в котором компилируются исходные файлы, разрешаются зависимости, проводятся тесты и в итоге получается исполняемый объект, готовый для использования.

**Maven –** фреймворк для автоматизации сборки проектов на основе описания их структуры в файлах на языке POM (англ. Project Object Model), являющемся подмножеством XML. Средство для управления и сборки проектов. Позволяет разработчикам полностью управлять жизненным циклом проекта; автоматизировать процессы, связанные со сборкой, тестированием, упаковкой проекта, управлением зависимостями и т.д.

Maven обеспечивает **декларативную** сборку проекта. Это значит что задается спецификация решения задачи, то есть описывается ожидаемый результат, а не последовательность действий.

**Особенности**

* Жесткая структура приложения
* Автоматическое разрешение зависимостей, версий и конфликтов.
* Выходом проекта является один файл (jar, war).
* Стандартный жизненный цикл.
* Возможна сборка из командной строки.

**Недостатки:**

* Требуется подключение к интернету для разрешения зависимостей (Либо придется разворачивать свой репозиторий).
* Если проект не типовой структуры, придется попотеть.

**Основные аспекты, которыми управляет maven**

* Создание (Build)
* Документирование (Documentation)
* Отчёты (Reporting)
* Зависимости (Dependencies)
* Релизы (Releases)

. SCM

* Список рассылки (Mailing list)
* Дистрибьюция (Distribution)

**Структура**

Универсальность Maven достигается и за счет жесткой структуры приложения. Maven создает структуру проекта, разработчики размещают соответствующие файлы в нужных каталогах.

|  |  |  |
| --- | --- | --- |
| src/main/java | Application/Library sources | исходный код приложения или библиотеки |
| src/main/resources | Application/Library resources | ресурсы приложения или библиотеки |
| src/main/filters | Resource filter files | файлы с параметрами фильтрации ресурсов |
| src/main/webapp | Web application sources | исходный код веб-приложения |
| src/test/java | Test sources | исходный код тестов (юнит-тестов) |
| src/test/resources | Test resources | ресурсы юнит тестов |
| src/test/filters | Test resource filter files | файлы с параметрами фильтрации ресурсов для тестов |
| src/it | Integration Tests (primarily for plugins) | интеграционные тесты (в основном для плагинов) |
| src/assembly | Assembly descriptors | дескрипторы сборки |
| src/site | Site | вебсайт приложения (документация) |
| LICENSE.txt | Project's license | лицензионное соглашение проекта |
| NOTICE.txt | Notices and attributions required by libraries that the project depends on | замечания и необходимые атрибуты библиотек, от зависит проект |
| README.txt | Project's readme | краткое описание проекта |
| /target |  | Собранный дистрибутив (jar, war) |
| /target/classes |  | Скомпилированный байт-код классов |

**POM**

POM (Project Object Model) – XML файл с настройками и конфигурацией проекта. Используется Maven для создания проекта.

Этот файл также содержит задачи и плагины.

Корневым элементом является **Project.**

В Project содержаться 3 обязательных элемента

* **groupId –** наименование организации Разработчика ПО. Обычно указывают доменное имя организации или сайта проекта.
* **artifactId –** идентификатор самого проекта, обычно название.
* **version –** версия проекта. Добавляется “-SNAPSHOT”, если версия в разработке и jar файл может меняться. Если нет -SNAPSHOT то maven будет брать версию из локального репозитория, иначе будет проверять на наличие обновлений.

Тег **<packaging>** определяет какого типа файл будет создаваться как результат сборки (jar, war, pom).

Также в POM содержатся

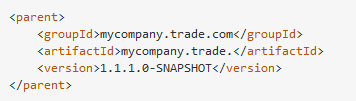
* Зависимости проекта (project dependencies)
* Плагины (plugins)
* Задачи/цели (goals)
* Профиль создания (build profiles)
* Список рассылки (mailing list)

POM файлы могут наследоваться друг от друга. Все файлы наследуются от **Super POM**, который содержит значения по умолчанию.

Один и тот же POM можно использовать для различных проектов, которые имеют одинаковую структуру.

Чтобы настроить **Родительский POM** нужно определить параметры GAV, и в теге <packaging> указать значение «pom». Дополнительно вводится секция <modules>, в которой перечисляются все дочерние проекты.

В pom.xml **дочерних** проектов необходимо ввести секцию <parent> и определить GAV-параметры родительского проекта.



**РЕПОЗИТОРИИ**

Репозиторий – место (директория), где хранятся все JAR, библиотеки, плагины и артефакты, которыми Maven может воспользоваться.

Типы репозиториев:

* **Локальный** – директория (по умолчанию %USER\_HOME%), которая хранится на вашем компьютере. Создается в момент первого выполнения любой команды Maven. Он хранит все зависимости проекта (библиотеки, плагины и т.д.) Все зависимости (их JAR-файлы) загружаются в локальный репозиторий при сборке проекта.
* **Центральный** – предоставляется сообществом Maven. Если Maven не может найти зависимость в локальном репозитории, то он ищет его в центральном.
* **Удаленный** – репозиторий, определяемый самим разработчиком. Если он определен, то загрузка происходит из него. Иначе процесс сборки прекращается, и выводится сообщение об ошибке.

**ЖИЗНЕННЫЙ ЦИКЛ**

**Жизненный цикл –** четко определенная последовательность фаз, которые определяют порядок, в котором должны выполняться цели.

**Фазы:**

1. **clean** — удаляются все скомпилированные файлы из каталога target (место, в котором сохраняются готовые артефакты);
2. **validate** — идет проверка, вся ли информация доступна для сборки проекта;
3. **compile** — компилируются файлы с исходным кодом;
4. **test** — запускаются тесты;
5. **package** — упаковываются скомпилированные файлы (в jar, war и т.д. архив);
6. **verify** — выполняются проверки для подтверждения готовности упакованного файла;
7. **install** — пакет помещается в локальный репозиторий. Теперь он может использоваться другими проектами как внешняя библиотека;
8. **site** — создается документация проекта;
9. **deploy** — собранный архив копируется в удаленный репозиторий.

**Задачи** – определенные действия, которые относятся к сборке проекта и его управлению. Может быть привязана как к нескольким фазам, так и не к одной.

**ПРОФИЛИ СБОРКИ**

**Профили сборки –** это множество настроек, которые могут быть использованы для установки и перезаписи стандартных значений сборки Maven в зависимости от окружения (Development, Production).

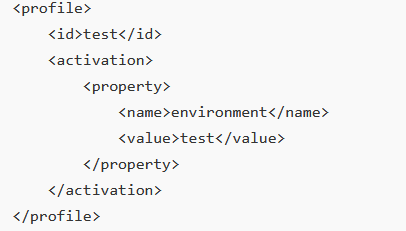
Профили настраиваются в файле pom.xml с помощью элементов activeProfiles / profiles и запускаются различными методами. Профили изменяют файл pom.xml во время сборки и используются для передачи параметров различным целевым окружениям

В Maven существует три основных **типа профилей сборки**:

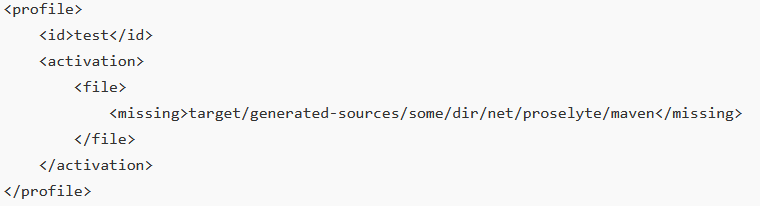
* **Per Project** - определяется в POM файле, pom.xml
* **Per User** - определяется в настройках Maven - xml файл (%USER\_HOME%/.m2/settings.xml).
* **Global** - определяется в глобальных настройках - xml файл (%M2\_HOME%/conf/settings.xml).

Профили сборки могут быть активированы различными способами:

* Явно с помощью команды и флага **-Pprofile**
* С помощью настроек Maven в файле settings.xml указать активные профили в секции <activeProfiles>  
  
* С помощью переменных окружения, добавив элемент <activation>



* С помощью ОС  
  
* С помощью присутствующих или отсутствующих файлов



**ПЛАГИНЫ**

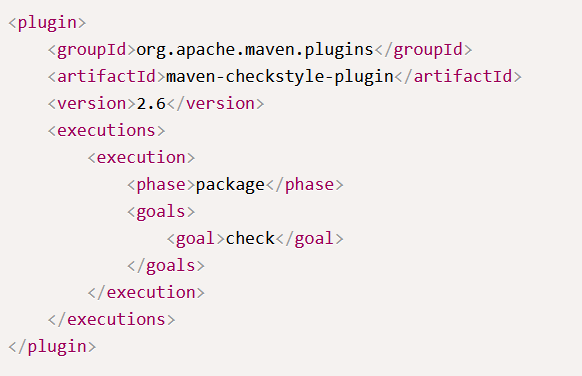
*Плагины –* представляют собой расширение функциональности maven.

Обычно подключаются к какой-то фазе. Представляет из себя набор целей (goals), то есть действий, которые плагин может выполнить.

Существует два типа плагинов в Maven:

* **Плагины сборки**Выполняются в процессе сборки и должны быть конфигурированны внутри блока <build></build> файла **pom.xml**
* **Плагины отчётов**Выполняются в процесса генерирования сайта (например отчеты о тестах) и должны быть конфигурированны внутри блока <reporting></reporting> файла **pom.xml**.

Подключаются в элементе <plagins><plagins/>.



К примерам можно отнести:

* maven-compiler-plugin - компилятор
* maven-archetype-plugin – чтобы создавать проект по существующему шаблону (архетипу)
* maven-javadoc-plugin – плагин для генерации документации
* maven-checkstyle-plugin – проверить стиль кода

**ЗАВИСИМОСТИ**

**Зависимости –** библиотеки, которые непосредственно используются в вашем проекте для компиляции кода или его тестирования.

Maven не разрешает циклические зависимости.

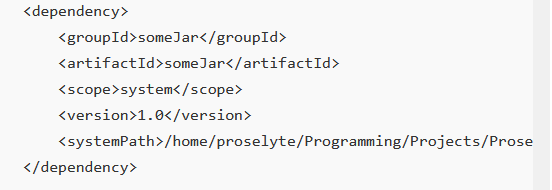
Подмодули наследуют зависимости своих родителей.

Зависимости описываются в секции <dependencies>.



**Внешние зависимости –** файлы библиотек, которые не содержатся не в центральном, не в удаленном репозитории. Подключаются таким же образом как и другие зависимости.

* определить groupId таким же именем, как и имя файла
* определить artifactId таким же именем, как и имя файла
* определить область видимости зависимости как system
* указать абсолютный путь к файлу



Зависимости являются **транзитивными –** то есть, если подключенные нами зависимости зависят от других библиотек, они будут подгружены автоматически (Информация о зависимостях есть в pom в удаленных репозиториях).

Чтобы граф зависимостей не разрастался, есть возможность ограничить включение зависимостей.

**Dependency mediation –** определяет, какая версия зависимости будет использоваться, когда встречается несколько версий артефакта. Maven выбирает ту версию, которая ближе к вашему проекту в дереве зависимости. Если 2 версии находятся на одинаковой глубине в дереве, побеждает первая декларация. Версию можно гарантировать, указав ее явно в POM вашего проекта.

**Scope –** область видимости, определяет этап жизненного цикла проекта, в котором эта зависимость будет использоваться и будет доступна.

* **Compile –** доступна на всех этапах ( при компиляции приложения, тестов и на стадии их запуска).
* **Provided –** артефакт используется на этапе компиляции и тестирования, но в сборку не включается. Предполагается, что среда исполнения (JDK или WEB-контейнер) предоставят данную зависимость во время выполнения.
* **Runtime –** указывает, что артефакт нужен только на стадии выполнения.
* **Test –** зависимость будет доступна только при компиляции, запуске тестов и построении отчета с результатами.
* **System –** аналогична provided, но указывает, что зависимость находится на локальной машине. Также нужно указать абсолютный путь к файлу. Обычно к таким артефактам относятся собственные наработки, либо те, которых нет в центральном либо удаленном репозитории.
* **Import –** используется только в зависимостях типа pom. Это значит что мы подключим все зависимости, объявленные в указанном pom.

**Секция** **<dependencyManagement>** содержит внутри себя <dependencies>. Эта секция является просто объявлением, и ничего не подключает. Подключение будет идти в последующих <dependencies>. Причем в них необязательно нужно будет указывать версию.

Это бывает полезно в многомодульных проектах, чтобы все дочерние модули использовали одинаковые версии библиотек. Объявление будет описано в родительском pom.

Можно отключить зависимость в случае обнаружения цикличности или отсутствия необходимости в определенной библиотеке используя элемент exclusion.

По сути, то что указано в <dependencies> всегда будет включено в pom наследник, а все что в <dependencyManagement> только если мы явно укажем.

Секция **<build>** является необязательной, т. к. существует значение по умолчанию. Данная секция содержит информацию по самой сборке, т.е. где находятся исходные файлы, файлы ресурсов, какие плагины используются.

* <finalName> - наименование результирующего файла сборки (jar, war, ear..), который создаётся в фазе package. Значение по умолчанию — «artifactId-version»;
* <sourceDirectory> - определение месторасположения файлов с исходным кодом. По умолчанию файлы располагаются в директории «${basedir}/src/main/java», но можно определить и в другом месте;
* <outputDirectory> - определение месторасположения директории, куда компилятор будет сохранять результаты компиляции - \*.class файлы. По умолчанию определено значение «target/classes»;
* <resources> и вложенные в неё тэги <resource> определяют местоположение файлов ресурсов. Ресурсы, в отличие от файлов исходного кода, при сборке просто копируются в директорию, значение по умолчанию которой равно «src/main/resources».

**Settings.xml**

**<localRepository>** - путь к локальному хранилищу системы.

**<interactiveMode>true</interactiveMode>** - разрешить ли мавен взаимодействовать с пользователем, запрашивая его ввод. По умолчанию – true.

**<offline>false</offline> -** может ли система работать в автономном режиме. По умолчанию false.

**<pluginGroups> -** groupId плагинов, которые можно использовать в консоли без указания groupId.

**<proxies><proxy>**- прокси

**Профили.** Значение активного профиля будет переопределять любые эквивалентные значения профиля в файле pom.xml.

* **<properties>**
* **<activation>**
* **Repositories**
* **Plugin Repositories**
* **Active Profiles**

**ДРУГИЕ ВОПРОСЫ**

Для запуска сборки без выполнения тестов добавьте -Dmaven.test.skip=true к команде в строке запуска maven: