

# Table of Contents

|  |    |
|--|----|
| <b>Maven FAQ</b>   | 3  |
| <b>Что такое сборка проекта, автоматизация сборки?</b>   | 3  |
| <b>Что такое Maven? Как он работает?</b>   | 3  |
| <b>Какие преимущества Maven?</b>   | 3  |
| <b>Какие недостатки Maven?</b>   | 4  |
| <b>Какими аспектами управляет Maven?</b>   | 4  |
| <b>Как узнать, какую версию Maven вы используете?</b>  | 4  |
| <b>Для чего был создан Maven?</b>  | 5  |
| <b>Какая структура каталогов в Maven?</b>  | 5  |
| <b>Где хранятся файлы классов при компиляции проекта Maven?</b>  | 5  |
| <b>Что такое pom.xml?</b>  | 6  |
| <b>Какую информацию содержит pom.xml?</b>  | 6  |
| <b>Что такое супер POM?</b>  | 6  |
| <b>Какие элементы необходимы для минимального POM?</b>   | 6  |
| <b>Что такое зависимости в Maven?</b>  | 7  |
| <b>Что такое артефакт в Maven?</b>   | 7  |
| <b>Что такое плагин в Maven?</b>   | 7  |
| <b>Что такое задача в Maven?</b>   | 7  |
| <b>Что такое архетип в Maven?</b>  | 7  |
| <b>Что такое репозиторий в Maven?</b>  | 7  |
| <b>Какие типы репозитория существуют в Maven?</b>  | 8  |
| <b>Какая команда устанавливает JAR-файл в локальное хранилище (репозиторий)?</b>                                     | 8  |
| <b>Какой порядок поиска зависимостей Maven?</b>  | 8  |
| <b>Какие два файла настройки есть в Maven, как они называются и где расположены?</b>                                 | 8  |
| <b>Что такое жизненный цикл сборки в Maven?</b>  | 9  |
| <b>Назовите основные фазы жизненного цикла сборки Maven?</b>   | 9  |
| <b>Что делает команда mvn site?</b>  | 9  |
| <b>Что делает команда mvn clean?</b>   | 9  |
| <b>Из каких фаз состоит жизненный цикл сборки Clean?</b>   | 9  |
| <b>Из каких фаз состоит жизненный цикл сборки Default (Build)?</b>   | 10 |
| <b>Из каких фаз состоит жизненный цикл сборки Site?</b>  | 11 |
| <b>Что сделает команда "mvn clean dependency:copy-dependencies package"?</b>   | 11 |
| <b>Что такое профиль сборки (Build Profile)?</b>   | 11 |
| <b>Какие типы профилей сборки (Build Profiles) вы знаете?</b>  | 11 |
| <b>Как вы можете активировать профили сборки?</b>  | 12 |
| <b>Для чего используются Maven плагины?</b>  | 12 |
| <b>Какие типы плагинов существуют в Maven?</b>   | 12 |
| <b>Когда Maven использует внешние зависимости?</b>   | 13 |
| <b>Что нужно определить для внешней зависимости?</b>   | 13 |
| <b>Какая команда создает новый проект на основе архетипа?</b>  | 13 |
| <b>Что такое SNAPSHOT в Maven?</b>   | 13 |
| <b>В чем разница между snapshot и версией?</b>   | 13 |
| <b>Что такое транзитивная зависимость в Maven?</b>   | 13 |
| <b>Как Maven определяет, какую версию зависимостей использовать, когда встречается множественный вариант выбора?</b> | 14 |
| <b>Что такое область видимости зависимостей (dependency scope)? Назовите значения</b>                                |    |

|  |    |
|--|----|
| <b>dependency scope.</b> .....   | 14 |
| <b>Какой минимальный набор информации нужен для составления ссылки<br/>зависимостей в разделе dependencyManagement?</b> .....            | 14 |
| <b>Как сослаться на свойство(property) определенное в файле pom.xml?</b> .....   | 14 |
| <b>Для чего нужен элемент &lt;execution&gt; в POM файле?</b> .....   | 15 |
| <b>Каким образом можно исключить зависимость в Maven?</b> .....  | 15 |
| <b>Что является полным именем артефакта?</b> .....   | 15 |
| <b>Если вы не определяете никакой информации, откуда ваш POM унаследует её?</b><br>.....   | 15 |
| <b>При сборке проекта Maven постоянно проверяет наличие обновлений в интернете.<br/>Можете ли вы собрать проект без интернета?</b> ..... | 15 |
| <b>Если при сборке проекта в тестах произошла ошибка, как собрать проект без<br/>запуска тестов?</b> .....                               | 16 |
| <b>Как запустить только один тест?</b> .....   | 16 |
| <b>Как остановить распространение наследования плагинов для дочерних POM?</b> .....  | 16 |
| <b>Какие теги pom.xml вы знаете?</b> .....   | 16 |
| <b>См. также</b> .....   | 17 |

# Maven FAQ

[maven](#), [java](#)

Основные ссылки по Maven:

- [Сайт проекта](#)
- [Основной репозиторий Maven](#)

## Что такое сборка проекта, автоматизация сборки?

**Сборка (англ. assembly)** - двоичный файл, содержащий исполняемый код программы или другой, подготовленный для использования информационный продукт. Сборка проекта - это в том числе процесс создания необходимого двоичного файла.

**Автоматизация сборки** - этап написания скриптов или автоматизация широкого спектра задач применительно к ПО, применяемому разработчиками в их повседневной деятельности, включая такие действия, как:

- Компиляция исходного кода в бинарный код
- Сборка бинарного кода
- Выполнение тестов
- Разворачивание программы на производственной платформе
- Написание сопроводительной документации или описание изменений новой версии

## Что такое Maven? Как он работает?

[Apache Maven](#) - это Java фреймворк для автоматизации сборки проектов, компиляции, создания jar, создания дистрибутива программы, генерации документации.

Если собирать большие проекты с командной строки, то команда для сборки будет очень длинной, поэтому её иногда записывают в bat/sh скрипт. Но такие скрипты зависят от платформы. Для того чтобы избавиться от этой зависимости и упростить написание скрипта, используют инструменты для сборки проекта.

Maven обеспечивает декларативную, а не императивную сборку проекта. То есть, в файлах проекта `pom.xml` содержится его описание, а не отдельные команды. Все задачи по обработке файлов в Maven выполняется через плагины.

## Какие преимущества Maven?

Основные преимущества Maven:

- Независимость от ОС. Сборка проекта происходит в любой операционной системе. Файл проекта один и тот же.
- Управление зависимостями. Редко какие проекты пишутся без использования сторонних

библиотек(зависимостей). Эти сторонние библиотеки зачастую тоже в свою очередь используют библиотеки разных версий. Maven позволяет управлять такими сложными зависимостями. Это позволяет разрешать конфликты версий, и в случае необходимости, легко переходить на новые версии библиотек.

- Возможна сборка из командной строки. Такое часто необходимо для автоматической сборки проекта на сервере (🐙 [Continuous Integration](#)).
- Хорошая интеграция со средами разработки. Основные среды разработки на java легко открывают проекты, которые собираются с помощью maven. При этом зачастую проект настраивать не нужно: он сразу готов к дальнейшей разработке.
- Как следствие, если с проектом работают в разных средах разработки, то maven удобный способ хранения настроек. Настраиваемый файл среды разработки и для сборки один и тот же - меньше дублирования данных и, соответственно, ошибок.
- Декларативное описание проекта. Указано **что, где и когда** надо сделать, а не **как** надо сделать.

## Какие недостатки Maven?

Недостатки Maven:

- Неочевидность. Если в [Apache Ant](#) указывается команда на удаление, и удаляется файл, то в случае Maven надо всем сердцем [довериться плагину](#) и документации по нему.
- При таком объёме необходимых знаний документации не так много, особенно по каким-то специальным моментам. Да и просто читать придётся много. Порог вхождения, если потребуется собирать даже не самое сложное приложение куда выше, чем у Ant.
- Если нужно найти какой-то специальный плагин - это будет сделать непросто, плагинов много. И не факт, что найденный подойдёт на все 100% и будет работать без ошибок.
- Нужен доступ в интернет (или придётся разворачивать собственный репозиторий, что трудоёмко)
- Большие трудности, если проект не типовой.

## Какими аспектами управляет Maven?

Вот основные аспекты, которыми позволяет управлять Maven:

- Создание (Build)
- Документирование (Documentation)
- Отчёты (Reporting)
- Зависимости (Dependencies)
- Релизы (Releases)
- Системы контроля версий (SCM)
- Список рассылки (Mailing list)
- Дистрибьюция (Distribution)

## Как узнать, какую версию Maven вы используете?

Версию можно узнать с помощью следующей команды:

```
mvn --version
```

## Для чего был создан Maven?

Основной целью Maven является предоставление разработчику:

- Понятной модели для проектов, которая может быть использована повторно и была бы проста в поддержке.
- Плагинов, которые могут взаимодействовать с этой моделью.

Структура и содержание проекта Maven указывается в специальном xml-файле, который называется **Project Object Model (POM)**, который является базовым модулем всей системы.

## Какая структура каталогов в Maven?

В Maven [стандартная структура каталогов](#), благодаря ей отпадает необходимость прописывать пути к файлам проекта. В корневом каталоге проекта находится `pom.xml` и несколько текстовых файлов. Всё остальное хозяйство аккуратно разложено в подкаталогах. Главные из них - **src** и **target**. Однако, порядок сохраняется и вглубь:

|                    |  |   |
|--------------------|--|---|
| 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 которых зависит проект |
| README.txt         | Project's readme   | краткое описание проекта  |

## Где хранятся файлы классов при компиляции проекта Maven?

Файлы классов хранятся в: `${basedir}/target/classes/`.

## Что такое pom.xml?

pom.xml - это XML-файл, который содержит информацию о конфигурации и деталях проекта, используемых при создании проекта на Maven. Он всегда находится в базовом каталоге проекта. Этот файл также содержит описание задач, список и параметры плагинов.

Во время выполнения задач, Maven ищет файл pom.xml в базовой директории проекта. Он читает его и получает необходимую информацию, после чего выполняет задачи.

Корневой элемент <project> содержит ссылку на схему XML, которая облегчает редактирование и проверку pom.xml:

```
<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">
```

Внутри тега project содержится основная и обязательная информация о проекте.

## Какую информацию содержит pom.xml?

Среди информации которую содержит pom.xml, мы можем выделить следующие блоки:

- Зависимости проекта (project dependencies)
- Плагины (plugins)
- Задачи/цели (goals)
- Профиль создания (build profiles)
- Версия проекта (project version)
- Разработчики (developers)
- Список рассылки (mailing list)

## Что такое супер POM?

Все POM-файлы являются наследниками родительского pom.xml. Этот POM-файл называется **Super POM** и содержит значения, унаследованные по умолчанию.

## Какие элементы необходимы для минимального POM?

Обязательные элементы для минимального POM это корневой элемент, modelVersion, GroupID, artifactID и версия. Минимальный POM файл:

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>my-project</artifactId>
```

```
<version>1.0</version>  
</project>
```

## Что такое зависимости в Maven?

**Зависимость (dependency)** - это те библиотеки, которые непосредственно используются в вашем проекте для компиляции кода или его тестирования.

## Что такое артефакт в Maven?

**Артефакт (artefact)** - это, по сути, любая библиотека, хранящаяся в репозитории (месте хранения). Это может быть какая-то зависимость или плагин. Обычно артефактом является JAR-файл, который хранится в репозитории Maven. Каждый артефакт содержит group ID, artifact ID и версию.

## Что такое плагин в Maven?

**Плагин (plugin)** - это зависимости Maven'a, расширяющие его функционал.

## Что такое задача в Maven?

**Задача (goal)** - это специальная задача, которая относится к сборке проекта и его управлению. Она может привязываться как к нескольким фазам, так и ни к одной. Задача, которая не привязана ни к одной фазе, может быть запущена вне фаз сборки с помощью прямого вызова.

## Что такое архетип в Maven?

**Архетип (archetype)** - 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.

## Что такое репозиторий в Maven?

**репозиторий (repository)** - глобальное хранилище всех библиотек, доступных для Maven, это место где хранятся артефакты: jar файлы, pom-файлы, javadoc, исходники, плагины.

## Какие типы репозитория существуют в Maven?

В Maven существуют три типа репозитариев:

- **Локальный (local) репозиторий** - это директория, которая хранится на нашем компьютере. Она создаётся в момент первого выполнения любой команды Maven. По умолчанию она расположена в `<home директория>/.m2/repository` - персональная для каждого пользователя. Фактически она является кэшем для центрального и удалённого репозитариев.
- **Центральный (central) репозиторий** - это репозиторий, который обеспечивается сообществом Maven. Он содержит огромное количество часто используемых библиотек. Расположен по адресу <http://repo1.maven.org/maven2/> и доступен на чтение для всех пользователей в интернете. Если Maven не может найти зависимости в локальном репозитории, то автоматически начинается поиск необходимых файлов в центральном репозитории. Для поиска по центральному репозиторию можно использовать сайт <https://mvnrepository.com/>
- **Удалённый (remote) репозиторий**. Иногда Maven не может найти необходимые зависимости в центральном репозитории, например при отсутствии интернета. В этом случае процесс сборки прерывается, и в консоль выводится сообщение об ошибке. Для того, чтобы предотвратить подобную ситуацию, в Maven предусмотрен механизм **удалённого репозитория**, который является репозитарием, определённым самим разработчиком. Там могут храниться все необходимые зависимости.

## Какая команда устанавливает JAR-файл в локальное хранилище (репозиторий)?

```
mvn install
```

## Какой порядок поиска зависимостей Maven?

Когда мы выполняем сборку проекта в Maven, автоматически начинается поиск необходимых зависимостей в следующем порядке:

1. Поиск зависимостей в локальном репозитории. Если зависимости не обнаружены, происходит переход к шагу 2.
2. Поиск зависимостей в центральном репозитории. Если они не обнаружены и удалённый репозиторий определён, 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.

## Какие два файла настройки есть в Maven, как они



## называются и где расположены?

В Maven, файлы настройки называются `settings.xml`, и они расположены в двух местах:

- каталог где установлен Maven: `$M2_Home/conf/settings.xml`
- домашняя директория пользователя: `${user.home}/.m2/settings.xml`

## Что такое жизненный цикл сборки в Maven?

**Жизненный цикл сборки (Lifecycle)** - это чётко определённая последовательность фаз, во время выполнения которых должны быть достигнуты определённые цели. Здесь фаза представляет собой стадию жизненного цикла.

## Назовите основные фазы жизненного цикла сборки Maven?

Когда Maven начинает сборку проекта, он проходит через определённую последовательность фаз сборки, и выполняет определенные задачи, которые указаны в каждой из фаз.

В Maven есть следующие три стандартных жизненных цикла:

- **Очистка (clean)** - очищает артефакты, созданные до сборки.
- **Сборка (default or build)** - используется для создания приложения.
- **Создание сайта проекта (site)** - генерирует документацию сайта для проекта.

## Что делает команда `mvn site`?

```
mvn site
```

создает веб-сайт проекта.

## Что делает команда `mvn clean`?

```
mvn clean
```

эта команда очищает целевую директорию от созданных в процессе сборки файлов.

## Из каких фаз состоит жизненный цикл сборки Clean?

Жизненный цикл сборки **Clean** состоит из следующих этапов:

- **pre-clean**
- **clean**
- **post-clean**

## Из каких фаз состоит жизненный цикл сборки Default (Build)?

**Default (Build)** - это основной жизненный цикл Maven, который используется для сборки проектов. Он включает в себя следующие фазы:

- **validate** - проверяет корректность метаданных о проекте, подтверждает, является ли проект корректным и вся ли необходимая информация доступна для завершения процесса сборки.
- **initialize** - инициализирует состояние сборки, например, различные настройки.
- **generate-sources** - включает любой исходный код в фазу компиляции.
- **process-sources** - обрабатывает исходный код (подготавливает). Например, фильтрует определённые значения.
- **generate-resources** - генерирует ресурсы, которые должны быть включены в пакет.
- **process-resources** - копирует и отправляет ресурсы в указанную директорию. Это фаза перед упаковкой.
- **compile** - компилирует исходный код проекта.
- **process-classes** - обработка файлов, полученных в результате компиляции. Например, оптимизация байт-кода Java классов.
- **generate-test-sources** - генерирует любые тестовые ресурсы, которые должны быть включены в фазу компиляции.
- **process-test-sources** - обрабатывает исходный код тестов. Например, фильтрует значения.
- **test-compile** - компилирует исходный код тестов в указанную директорию тестов.
- **process-test-classes** - обрабатывает файлы, полученные в результате компиляции исходного кода тестов.
- **test** - запускает тесты классов, используя приемлемый фреймворк юнит-тестирования (например, Junit).
- **prepare-package** - выполняет все необходимые операции для подготовки пакета, непосредственно перед упаковкой.
- **package** - преобразует скомпилированный код и пакет в дистрибутивный формат. Такие как JAR, WAR или EAR.
- **pre-integration-test** - выполняет необходимые действия перед выполнением интеграционных тестов.
- **integration-test** - обрабатывает и распаковывает пакет, если необходимо, в среду, где будут выполняться интеграционные тесты.
- **post-integration-test** - выполняет действия, необходимые после выполнения интеграционных тестов. Например, освобождение ресурсов.
- **verify** - выполняет любые проверки для подтверждения того, что пакет пригоден и отвечает критериям качества.
- **install** - переносит пакет в локальный репозиторий, откуда он будет доступен для использования как зависимость в других проектах.
- **deploy** - копирует финальный пакет (архив) в удалённый репозиторий для того, чтобы сделать его доступным другим разработчикам и проектам.

Здесь также необходимо уточнить два момента:

- Когда мы выполняем команду Maven, например `install`, то будут выполнены фазы до `install` и фаза `install`.
- Различные задачи Maven будут привязаны к различным фазам жизненного цикла Maven в зависимости от типа архива (JAR/WAR/EAR).

## Из каких фаз состоит жизненный цикл сборки Site?

Жизненный цикл сборки **Site** состоит из следующих этапов:

- **pre-site**
- **site**
- **post-site**
- **site-deploy**

## Что делает команда "mvn clean dependency:copy-dependencies package"?

Порядок выполнения зависит от порядка вызова целей и фаз. Рассмотрим данную команду:

```
mvn clean dependency:copy-dependencies package
```

Аргументы `clean` и `package` являются фазами сборки, в то время как `"dependency:copy-dependencies"` является задачей.

В этом случае, сначала будет выполнена фаза `clean`, после этого будет выполнена задача `"dependency:copy-dependencies"`. После чего будет выполнена фаза `package`.

## Что такое профиль сборки (Build Profile)?

**Профиль сборки** - это множество настроек, которые могут быть использованы для установки или перезаписи стандартных значений сборки Maven.

Используя профиль сборки Maven, мы можем настраивать сборку для различных окружений, таких как `Development` или `Production`.

Профили настраиваются в файле `pom.xml` с помощью элементов `activeProfiles` / `profiles` и запускаются различными методами.

## Какие типы профилей сборки (Build Profiles) вы знаете?

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`).

## Как вы можете активировать профили сборки?

Профиль сборки Maven может быть активирован различными способами:

- использованием команды в консоли
- с помощью настроек Maven
- с помощью переменных окружения
- в настройках ОС
- существующими или отсутствующими файлами

## Для чего используются Maven плагины?

Maven плагины используются для:

- создания jar-файла
- создания war-файла
- компиляции кода файлов
- юнит-тестирования кода
- создания отчётов проекта
- создания документации проекта

## Какие типы плагинов существуют в Maven?

В Maven существует два типа плагинов:

- **Плагины сборки (Build plugins)** - выполняются в процессе сборки и должны быть сконфигурированы внутри блока

```
<build></build>
```

файла `pom.xml`.

- **Плагины отчётов (Reporting plugins)** - выполняются в процессе генерирования сайта и должны быть сконфигурированы внутри блока

```
<reporting></reporting>
```

файла `pom.xml`.

## Когда Maven использует внешние зависимости?

Если необходимые файлы не найдены ни в центральном, ни на удалённом репозитории, тогда для решения этой проблемы используются внешние зависимости.

## Что нужно определить для внешней зависимости?

Внешние зависимости могут быть сконфигурированы в файле `pom.xml` таким же образом, как и другие зависимости, для этого нужно:

- 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

## Какая команда создает новый проект на основе архетипа?

Переходим в нужную нам директорию и выполняем в терминале следующую команду:

```
mvn archetype:generate
```

## Что такое SNAPSHOT в Maven?

**SNAPSHOT** - это специальная версия, которая показывает текущую рабочую копию. При каждой сборке Maven проверяет наличие новой **snapshot** версии на удалённом репозитории.

## В чем разница между snapshot и версией?

В случае с **обычной версией**, если Maven однажды загрузил версию `data-service:1.0`, то он больше не будет пытаться загрузить новую версию 1.0 из репозитория. Для того, чтобы скачать обновлённый продукт `data-service` должен быть обновлён до версии 1.1.

В случае со **snapshot**, Maven автоматически будет подтягивать крайний snapshot (`data-service:1.0-SNAPSHOT`) каждый раз, когда будет выполняться сборка проекта.

## Что такое транзитивная зависимость в Maven?

**Транзитивная зависимость** - позволяет избегать необходимости изучать и указывать библиотеки, которые требуются для самой зависимости, и включает их автоматически.

Необходимые библиотеки подгружаются в проект автоматически. При разрешении конфликта версий используется принцип «ближайшей» зависимости, то есть выбирается зависимость, путь к которой через список зависимых проектов является наиболее коротким.

## Как Maven определяет, какую версию зависимостей использовать, когда встречается множественный вариант выбора?

**Dependency mediation** - определяет, какая версия зависимости будет использоваться, когда встречается несколько версий артефактов. Если две версии зависимости на той же глубине в дереве зависимостей, то будет использоваться та которая объявлена первой. Здесь важен порядок объявления: первое объявление выигрывает.

## Что такое область видимости зависимостей (dependency scope)? Назовите значения dependency scope.

Существуют следующие **области видимости зависимостей**:

- **compile** - это область по умолчанию, используются, если ничего больше не определено. Compile зависимости доступны во всех classpath проекта.
- **provided** - это очень похоже на compile, но указывает на то, что вы ожидаете от JDK или контейнера предоставить зависимость в ходе выполнения. Эта область доступна только на compilation и test classpath и не является транзитивной.
- **runtime** - эта область указывает на то, что зависимость не обязательна для compilation, но для фаз выполнения.
- **test** - эта область указывает, что зависимость не обязательна для нормального использования приложения.
- **system** - эта область похожа на provided за исключением того, что вы предоставляете JAR. Артефакт всегда доступен и не смотрит в репозиторий.
- **import** - эта область используется в зависимости типа pom в <dependencyManagement> разделе. Это указывает на то, что определенный POM будет заменен зависимостями в этом POM <dependencyManagement> разделе.

## Какой минимальный набор информации нужен для составления ссылки зависимостей в разделе dependencyManagement?

Минимальный набор информации такой: {groupId, artifactId, type, classifier}.

## Как сослаться на свойство(property) определенное в

## файле pom.xml?

На все свойства в pom.xml, можно сослаться с помощью префиксов “**project.**” или “**pom.**”  
Ниже приведён пример некоторых часто используемых элементов:

- `${project.build.directory}` - “target” директория, или тоже самое `${pom.project.build.directory}`
- `${project.build.outputDirectory}` - путь к директории куда компилятор складывает файлы, по умолчанию “target/classes”
- `${project.name}` или `${pom.name}` - имя проекта
- `${project.version}` или `${pom.version}` - версия проекта

## Для чего нужен элемент <execution> в POM файле?

Элемент **<execution>** содержит информацию, необходимую для выполнения плагина.

## Каким образом можно исключить зависимость в Maven?

Файл описания проекта предусматривает возможность исключить зависимость в случае обнаружения цикличности или отсутствия необходимости в определённой библиотеке. Зависимость может быть исключена используя элемент `exclusion`.

## Что является полным именем артефакта?

```
<groupId>:<artifactId>:<version>
```

## Если вы не определяете никакой информации, откуда ваш POM унаследует её?

Все POM-ы наследуются от родителя, несмотря на то, определен ли он явно или нет. Это базовый POM известный как “супер POM”, он содержит значения, которые наследуются по умолчанию.

## При сборке проекта Maven постоянно проверяет наличие обновлений в интернете. Можете ли вы собрать проект без интернета?

Да, можете, если в вашем локальном репозитории есть все необходимые для сборки артефакты.

## Если при сборке проекта в тестах произошла ошибка, как собрать проект без запуска тестов?

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

```
mvn install -Dmaven.test.skip=true
```

## Как запустить только один тест?

Для запуска только одного теста добавьте `-Dtest=[Имя класса]` к команде в строке запуска maven. Например:

```
mvn install -Dtest=org.apache.maven.utils.ConverterTest
```

## Как остановить распространение наследования плагинов для дочерних POM?

Установить `<inherited>` в **false**.

## Какие теги `pom.xml` вы знаете?

Вот некоторые из них:

- **project** - описывает проект, это элемент верхнего уровня во всех файлах `pom.xml`
- **groupId** - по-сути, это имя пакета. Полностью отражается в структуре каталогов
- **artifactId** - название проекта. В структуре каталогов не отображается
- **version** - версия проекта
- **packaging** - определяет, какой тип файла будет собран. Варианты: `pom`, `jar`, `war`, `ear`
- **dependencies** - указываются зависимости
- **build** - информация о сборке проекта
- **name** - это уже необязательное описание проекта. В данном случае его название
- **description** - элемент представляет собой общее описание проекта. Это часто используется в генерации документации Maven
- **url** - интернет-страница проекта
- **repositories** - репозитории для артефактов
- **pluginRepositories** - репозитории для плагинов Maven

Эта страница содержит переработанные и откорректированные материалы с <https://jsehelper.blogspot.com/2016/05/maven-1.html>



## См. также

- [Заметки по Java](#)

From:

<https://kibi.ru/> - **КибИ.ру**

Permanent link:

<https://kibi.ru/notes/java/maven>

Last update: **2018/12/19 18:35**

