Введение в Maven

Дмитрий Зинушин
@dzinushin
Instream http://instream.ru/

январь 2012

Что такое maven?

"Maven is a project development management and comprehension tool"

с сайта maven.apache.org

- инструмент для сборки и управления проектами [на Java] (build tool)
- инструмент для управления ЖЦ проекта
- инструмент для автоматизации

Откуда пошел и есть Maven

Пришел из мира Java
автор *Jason van Zyl*
@jvanzyl
Chief Technology Officer
компании Sonatype

Maven, a Yiddish word meaning accumulator of knowledge (из документации)
в переводе с американского английского maven значит *специалист, знаток*

Другие утилиты для сборки проектов

- shell/bat скрипты
- make
- cmake
- scons
- ant
- ...

Почему Maven?

- на текущий момент одна из самых широко распространенных утилит для сборки в мире Java (загляните в исходники почти любого проекта от apache.org - найдете там pom.xml)
- огромный актуальный репозиторий артефактов в репозиториях maven
- поддерживается большинством современных IDE (Eclipse, IntelliJ IDEA, NetBeans и т.д.)

Ключевые преимущества

- **декларативный** язык описания проекта (POM)
- автоматическое управление зависимостями
- огромный, поддерживаемый в актуальном состоянии репозиторий артефактов
- модульная расширяемая за счет плагинов архитектура, огромное количество плагинов

Главные недостатки

- сложность освоения
- неочевидность (контринтуитивность) в некоторых моментах
- не очень хорошая документация
- огромное количество плагинов (трудно сориентироваться)
- трудно разобраться если что то пошло не так (возникла ошибка)
- необходим доступ в Интернет или собственный репозиторий артефактов

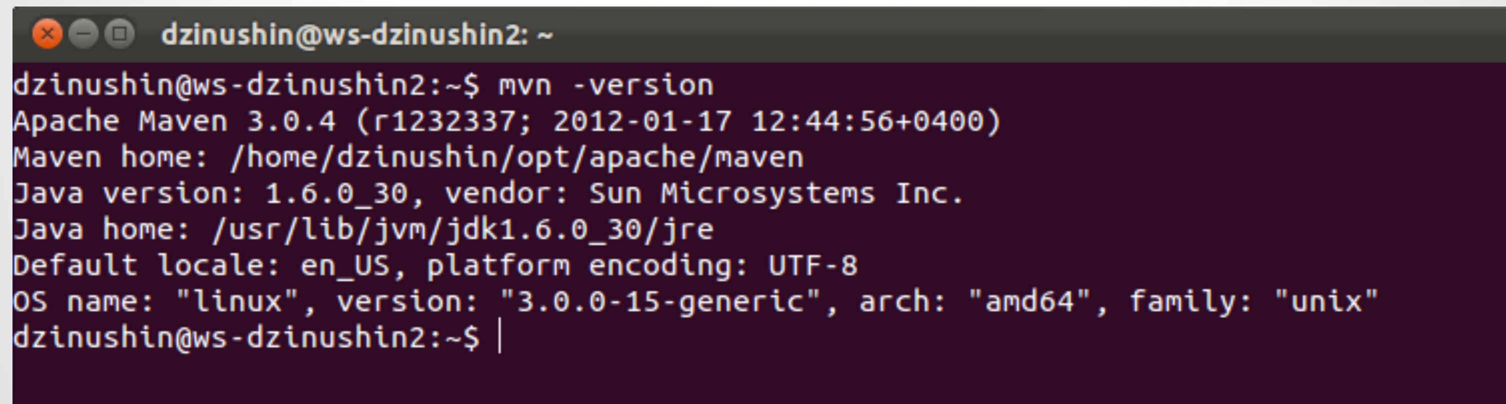
Установка Maven

- требует наличия на машине JDK версии >=1.5
- скачиваем с сайта проекта http://apache.maven.org
- разворачиваем архив
- прописываем переменную окружения M2_HOME
- прописываем путь $M2_HOME/bin в PATH
- запуск командой mvn

Проверим что maven успешно установлен

Выполним на Linux команду

mvn -version



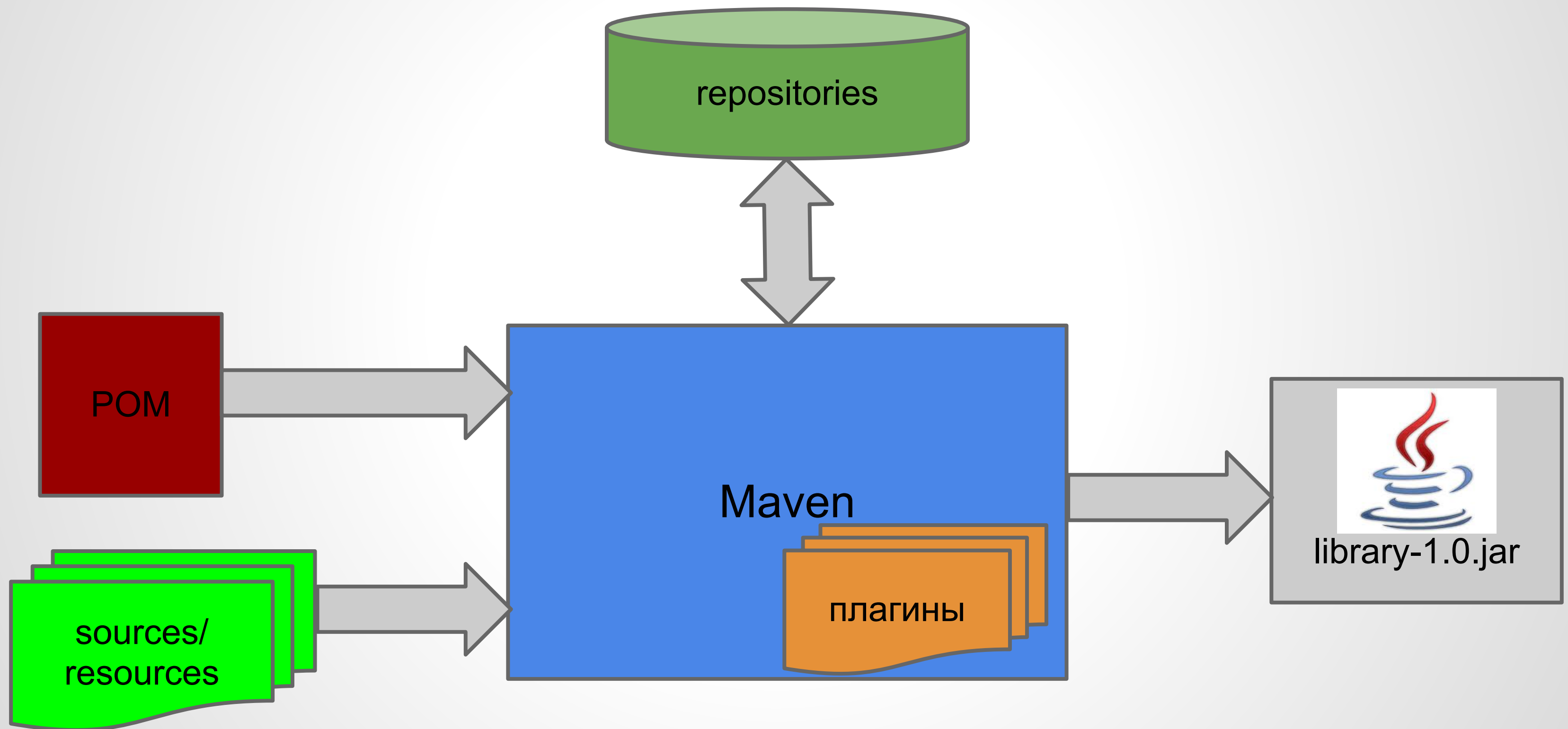
Как работает Maven?

With maven, you execute goals in plugins over the different phases of the build lifecycle, to generate artifacts.

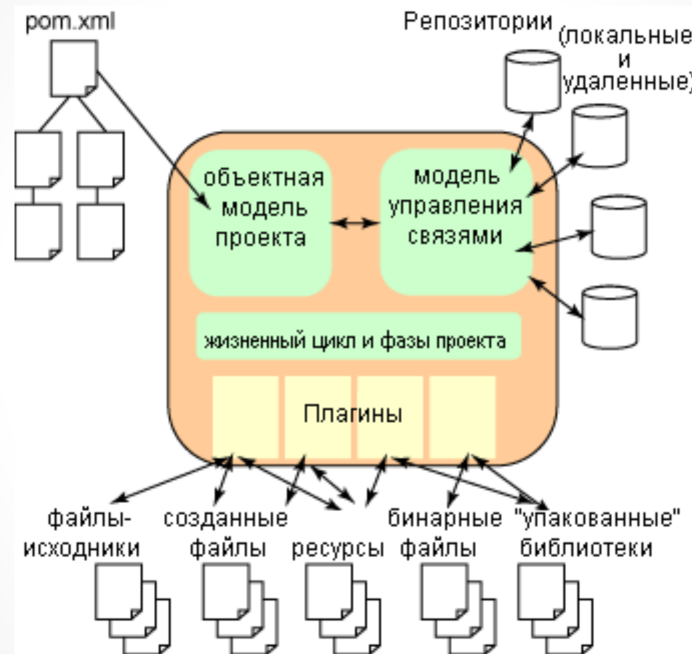
Examples of artifacts are jars, wars and ears. These artifacts have an artifactId, a groupId and a version. Together, these are called the artifact's "coordinates." The artifacts stored in repositories. Artifacts are deployed to remote repositories and installed into local repositories. A POM (Project Object Model) describes a project.

(c) Demystifying Maven by Mike Desjardins

Как все это работает?



Еще раз как все ЭТО работает?



(c) IBM developerworks

Артефакт

- Что есть Артефакт? Да все что угодно что производит наш проект (jar,war, ear и т.п.) или использует maven (плагин)
- Результатом работы Maven является создание (построение) артефакта, а так же ряд дополнительных действий над ним (тестирование, инсталляция в локальный репозиторий, deployment)
- Сам артефакт зависит от других артефактов (наших и внешних, плагинов maven)

Координаты артефакта

- groupId
- artifactId
- [packaging]

  default jar
- version

  в формате mmm.nnn.bbb-ssssss-dd , необязательными являются поля ssssss (спецификатор SNAPSHOT,RELEASE и т.п.) и dd (номер сборки)
- [classifier]

groupId:artifactId[:packaging]:version[:classifier]

Примеры maven координат

log4j

```
<groupId>log4j</groupId>
<artifactId>log4j</artifactId>
<version>1.2.16</version>
```

spring

```
<groupId>org.springframework</groupId>
<artifactId>spring-core</artifactId>
<version>3.1.0.RELEASE</version>
```

POM файл

POM - Project Object Model, xml файл, обычно называется pom.xml

POM файл содержит описание нашего проекта (декларативный стиль!) и все специфические его настройки.

Пример минимального POM файла (данный пример работает!!!):

```
<project xmlns=...>
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.codehaus.mojo</groupId>
  <artifactId>my-project</artifactId>
  <version>1.0</version>
</project>
```

Минимальные требования к POM

Минимально POM файл проекта должен содержать лишь версию модели и координаты артефакта проекта.

Декларативный стиль описания проекта в POM

Основная концепция maven в том что мы следуем станадартным практикам разработки (best practices) с зафиксированными правилами и настройками **по умолчанию.** Maven использует наследование, агрегирование и управление зависимостями при описании проекта в POM файле.

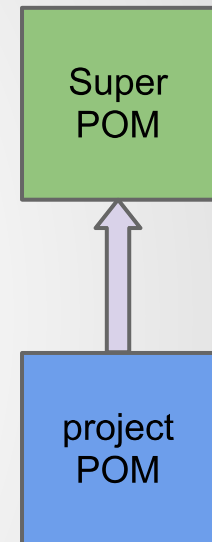
Пример стандартного размещения файлов java проекта:

Directory name	Purpose
project home	Contains the pom.xml and all subdirectories.
src/main/java	Contains the deliverable Java sourcecode for the project.
src/main/resources	Contains the deliverable resources for the project, such as property files.
src/test/java	Contains the testing classes (JUnit or TestNG test cases, for example) for the project
src/test/resources	Contains resources necessary for testing.

Build директория по умолчанию - target .

Super POM

Super POM это POM файл со всеми настройками по умолчанию. POM файл любого нашего проекта **наследует** от Super POM файла (наследуется стандартная структура директорий, набор плагинов, стандартные репозитории), и переопределяет его некоторые настройки и добавляя новое поведение за счет привязки новых goal-ов к фазам жизненного цикла.



Где находится этот Super POM? Внутри Maven-a (в его jar файлах) .

Т.е. Maven следует парадигме проектирования **Convention over configuration** (aka coding by convention) .

Convention over configuration

Convention over configuration (also known as coding by convention) is a software design paradigm which seeks to decrease the number of decisions that developers need to make, gaining simplicity, but not necessarily losing flexibility.

The phrase essentially means a developer only needs to specify unconventional aspects of the application. For example, if there's a class Sale in the model, the corresponding table in the database is called “sales” by default. It is only if one deviates from this convention, such as calling the table “products_sold”, that one needs to write code regarding these names.
When the convention implemented by the tool you are using matches your desired behavior, you enjoy the benefits without having to write configuration files. When your desired behavior deviates from the implemented convention, then you configure your desired behavior.

цитатата из Википедии

Эффективный POM

Эффективным POM файлом называется POM файл, образованный из Super POM файла и POM файла/ов проекта .

Увидеть его можно вот так:

mvn help:effective-pom

Анатомия POM файла

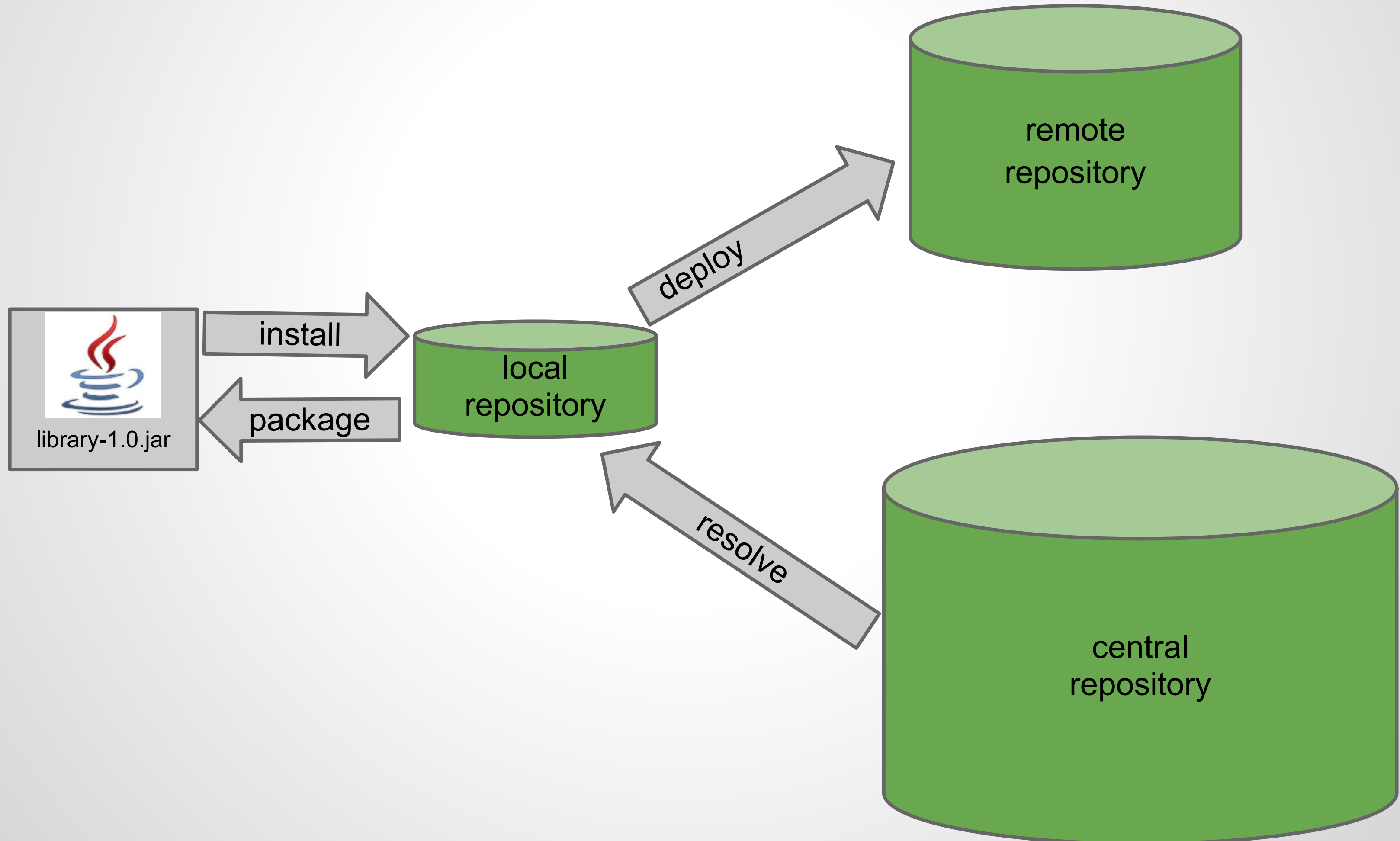
```
<project xmlns=http://maven.apache.org/POM/4.0.0 >
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.project</groupId>
  <artifactId>superproject</artifact>
  <packaging>jar</packaging>
  <version>1.0.0</version>
  <name>Super Duper Amazing Deluxe Project</name>
  <modules>
    <!-- sub-modules of this project -->
  </modules>
  <parent>
    <!-- parent POM stuff if applicable -->
  </parent>
  <properties>
    <!-- ad-hoc properties used in the build -->
  </properties>
  <dependencies>
    <!-- Dependency Stuff -->
  </dependencies>
  <build>
    <!-- Build Configuration  -->
    <plugins>
      <!-- plugin configuration -->
    </plugins>
  </build>
  <profiles>
    <!-- build profiles -->
  </profiles>
</project>
```

Репозитории

Репозиторий maven это файловое хранилище с метаинформацией и быстрым поиском и доступом

- Бывают двух типов
  - local (находятся в ~/.m2/repository)
  - remote (например, стандартный http://repo1.maven.org/maven2 или внутренний репозиторий компании, например, Nexus)
- используются для хранения и получения зависимостей (dependencies) проекта и плагинов maven

Из кода в репозиторий (и обратно)



Lifecycle (жизненный цикл) проекта

Существуют три стандартных lifecycles:

- clean - очистка проекта
- default - построение проекта из исходных кодов
- site - построение вторичных артефактов (документация, wiki, сайт и т.п.)

Жизненный цикл состоит из фаз. К каждой фазе может быть привязан ноль или более goal-ов различных плагинов. По умолчанию, набор фаз с привязанными плагинами стандартен и зависит от типа артефакта проекта (конкретно - от типа packaging).

Clean Lifecycle

Содержит три фазы

- pre-clean
- clean
- post-clean

By default **только** к фазе clean привязан goal плагина clean . Это можно изменить настройкой pom.xml .

Выполнить можно набрав:

```
mvn clean
```

Maven's Default Lifecycle

- validate
- initialize
- generate-sources
- process-sources
- generate-resources
- process-resources
- compile
- process-classes
- generate-test-sources
- process-test-sources
- generate-test-resources
- process-test-resources
- test-compile
- process-test-classes
- test
- prepare-package
- package
- pre-integration-test
- integration-test
- post-integration-test
- verify
- install
- deploy

содержит 23 (!) фазы

Фактический набор фаз (эффектививный набор)

Фактически задействованы только 8 следующих фаз:

- **validate** - validate the project is correct and all necessary information is available
- **compile** - compile the source code of the project
- **test** - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
- **package** - take the compiled code and package it in its distributable format, such as a JAR.
- **integration-test** - process and deploy the package if necessary into an environment where integration tests can be run
- **verify** - run any checks to verify the package is valid and meets quality criteria
- **install** - install the package into the local repository, for use as a dependency in other projects locally
- **deploy** - done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

Выполнение фаз default lifecycle

Осуществляется командой

mvn [имя фазы]

При выполнении определенной фазы автоматически последовательно выполняются все предыдущие фазы (те фазы от которых зависит выполняемая):

mvn package

validate -> compile -> test -> package

Пример сборки проекта с нуля

```
mvn clean install
```

Плагины

Плагины содержат goal's (один или более) которые могут вызываться в фазах жизненных циклов или напрямую. Плагины тоже являются артефактами и тоже хранятся в репозиториях.

Список стандартных плагинов можной найти здесь: http://maven.apache.org/plugins/

Как вызвать goal плагина напрямую из командной строки:

```
mvn имя_плагина:имя_goal
```

Например:

```
mvn clean:clean
mvn help:help
mvn dependency:copy-dependencies
mvn exec:exec -DmainClass=com.demo.App
```

Плагин help

Один из самых полезных плагинов ;-)

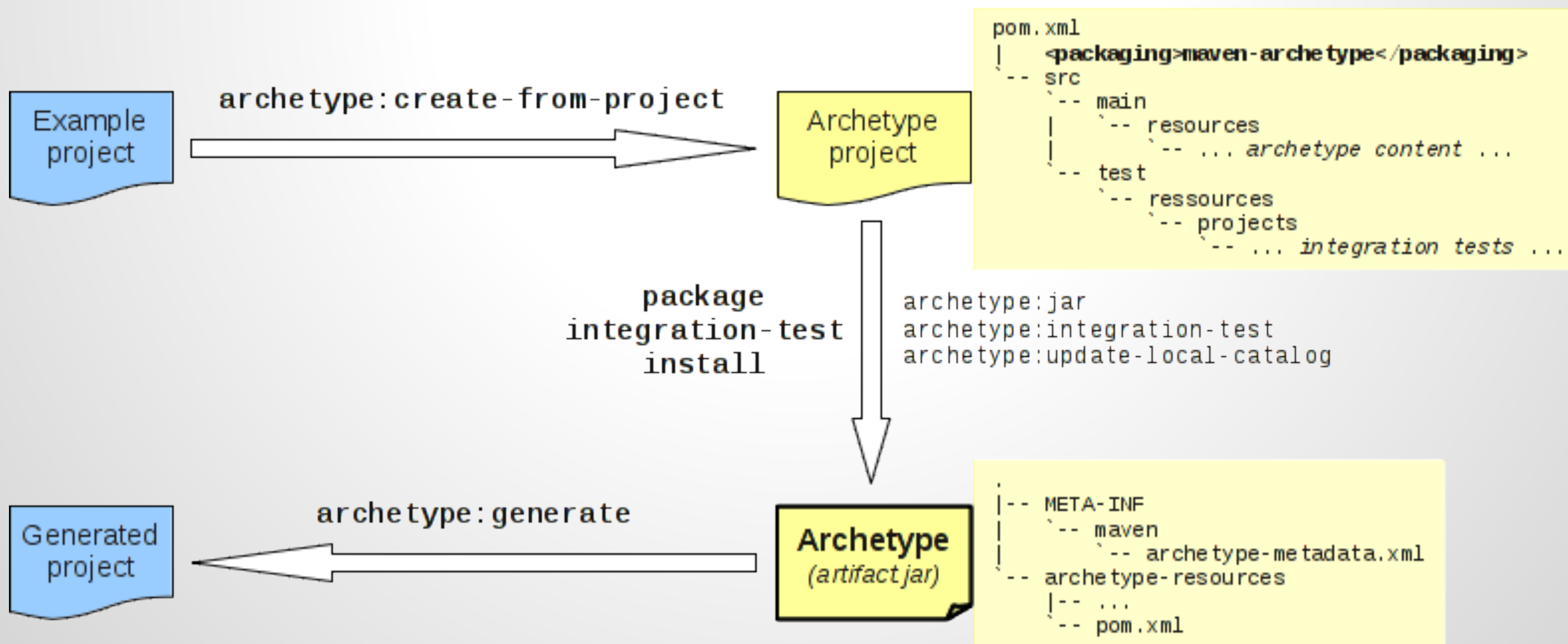
```
mvn help:help
mvn help:describe -Dcmd=install -Ddetail=true
mvn help:describe -Dcmd=deploy
mvn help:describe -Dplugin=org.apache.maven.plugins:maven-dependency-plugin
mvn help:describe -Dplugin=dependency
mvn help:describe -Dplugin=assembly -Ddetail=true
mvn help:describe -Dplugin=install
mvn help:effective-pom
mvn help:system
```

Плагин archetype

"The Archetype Plugin allows the user to create a Maven project from an existing template called an archetype."

из документации по плагину

В отличие от плагинов фаз создания, плагин archetype запускается вне жизненного цикла проекта Maven и используется для создания проектов Maven.



Создание простейшего проекта с помощью плагина archetype

В Maven встроен набор стандартных архетипов, а также из репозиториев доступно огромное количество сторонних архетипов.

Для создания простейшего проекта maven с помощью плагина archetype необходимо выполнить следующую команду:

```
mvn archetype:generate \
-DarchetypeArtifactId=maven-archetype-quickstart \
-DgroupId=test -DartifactId=test -Dversion=1.0-SNAPSHOT \
-DinteractiveMode=false
```

Другие стандартные плагины

- **clean** - имеет единственный goal clean. Используется для очистки директории проекта от вторичных файлов.
- **compile** - плагин для компиляции java исходников. По умолчанию использует javac и версию java 1.5 .
- **deploy** - плагин для загрузки артефактов в remote repositories
- **exec** - плагин для запуска приложений
- **antrun** - запуск ant tasks из maven
- ...

Пример использования нестандартного плагина

Рассмотрим в качестве примера cxf-codegen-plugin библиотеки CXF (cxf.apache.org) .

На примере данного плагина мы рассмотрим как задействовать нестандартный плагин, подключив его goal к фазе жизненного цикла проекта.

Задействуем goal wsdl2java для генерации из wsdl java кода, реализующего клиента для работы с WS.

Узнать о данном плагине можно на странице проекта http://cxf.apache.org/docs/maven-cxf-codegen-plugin-wsdl-to-java.html

или уже знакомым нам образом:

```
mvn help:describe -Dplugin=org.apache.cxf:cxf-codegen-plugin -Ddetail
```

Как подключить goal плагина в POM к фазе lifecycle

```
<project>
  ...
  <build>
    ...
    <plugins>
       <plugin>
          <executions>
            <execution>
              <id>id</id>                          <!-- уникальный id (строка) execution -->
              <phase>phase-name</phase> <!-- имя фазы lifecycle -->
              <goals>
                <goal>goal-name</goal>       <!-- имя goal плагина -->
              </goals>
               ...
            </execution>
          </executions>
       </plugin>
    </plugins>
    ...
  </build>
</project>
```

Пример настройки ч.1

```xml
<!-- Generate Java classes from WSDL during build -->
<!-- phase generate-sources -->
<!--sources generated in ${basedir}/target/generated-sources/cxf -->
<plugin>
	<groupId>org.apache.cxf</groupId>
	<artifactId>cxf-codegen-plugin</artifactId>
	<version>${cxf.version}</version>
	<executions>
		<execution>
			<id>generate-sources</id>
			<phase>generate-sources</phase>
			<goals>
				<goal>wsdl2java</goal>
			</goals>
			<configuration> ... </configuration>
		</execution>
	</executions>
</plugin>
```

Пример настройки ч.2

```
<configuration>
    <wsdlOptions>
        <wsdlOption>
            <wsdl>${wsdlFile}</wsdl>
            <bindingFiles>
                <bindingFile>${bindingsFile}</bindingFile>
            </bindingFiles>
            <extraargs>
                <extraarg>-verbose</extraarg>
                <extraarg>-client</extraarg>
            </extraargs>
        </wsdlOption>
    </wsdlOptions>
</configuration>
```

Управление зависимостями в Maven (dependency managment)

- зависимости (dependecies)
  - включают в себя все внешние библиотеки (jar, war, ear и т.п.) и файлы необходимые для успешной сборки нашего проекта
  - тоже являются артефактами (проект зависит от конкретной версии библиотеки)
  - зависимости могут быть транзитивными (например Spring, log4j и т.п.)
- механизм разрешения зависимостей
  - позволяет выбрать ту версию библиотеки которая удовлетворит все зависимости

Как найти нужную библиотеку

http://search.maven.org

SEARCH | ADVANCED SEARCH | BROWSE | QUICK STATS

log4j

SEARCH

Advanced Search | API Guide | Help

Search Results

< 1 2 3 > displaying 1 to 20 of 55

GroupId	ArtifactId	Latest Version	Updated	Download
de.huxhorn.lilith	log4j	0.9.41 all (2)	02-May-2011	pom jar javadoc.jar sources.jar
log4j	log4j	1.2.16 all (13)	31-Mar-2010	pom jar zip tar.gz javadoc.jar sources.jar
org.mod4j.org.eclipse.xtext	log4j	1.2.15	14-Aug-2009	pom jar
ant	ant-apache-log4j	1.6.5 all (4)	09-Nov-2005	pom jar
ant	ant-jakarta-log4j	1.6.1 all (2)	09-Nov-2005	pom jar
plexus	plexus-log4j-logging	1.0	09-Nov-2005	pom jar
log4j	apache-log4j-extras	1.1 all (2)	02-Dec-2010	pom jar javadoc.jar sources.jar
org.apache.ant	ant-apache-log4j	1.8.2 all (5)	27-Dec-2010	pom jar
net.sf.buildbox	strictlogging-log4j	1.0.1 all (2)	14-Nov-2010	pom jar
de.huxhorn.lilith	de.huxhorn.lilith.log4j.master	0.9.39 all (5)	12-May-2010	pom
net.sourceforge.openutils	openutils-log4j	2.0.5 all (8)	06-Sep-2009	pom jar sources.jar
org.objectweb.monolog	monolog-wrapper-log4j	2.1.12 all (5)	09-Mar-2009	pom jar
net.sf.buildbox.strictlogging	strictlogging-log4j	1.0.0	14-Jan-2008	pom jar
org.apache.geronimo.gshell	gshell-diet-log4j	1.0-alpha-1	22-Dec-2007	pom jar
com.sdicons.jsontools	jsontools-log4j	1.3 all (2)	17-Sep-2006	pom jar
avalon-logging	avalon-logging-log4j	1.0.dev-0	09-Nov-2005	pom jar
org.slf4j13	slf4j-log4j13	1.0-beta9	09-Nov-2005	pom
org.mortbay.jetty.testwars	test-war-log4j_1.2.15	8.1.0.RC5 all (34)	21-Jan-2012	pom war config.jar javadoc.jar sources.jar
org.mortbay.jetty.testwars	test-war-log4j_1.1.3	8.1.0.RC5 all (31)	21-Jan-2012	pom war config.jar javadoc.jar sources.jar
com.yammer.metrics	metrics-log4j	2.0.0-RC0 all (3)	19-Jan-2012	pom jar javadoc.jar sources.jar

< 1 2 3 > displaying 1 to 20 of 55

Настройка зависимостей проекта в POM файле

```
<dependencies>
  <dependency>
    <groupId>com.library</groupId>
    <artifactId>library-A</artifactId>
    <version>1.3.5</version>
    <scope>compile</scope>
    <optional>false</optional>
  </dependency>
</dependencies>
```

Управление версиями зависимостей (на примере)

- 1.3.5 - предпочтительной является версия 1.3.5, но допустимы и более новые версии при разрешении конфликтов
- (1.3.4,1.3.9) - любая версия внутри диапазона с 1.3.2 по 1.3.9, исключая границы диапазона
- [1.3.4,1.3.9] - любая версия из диапазона с 1.3.2 по 1.3.9, включая границы диапазона
- [,1.3.9] - любая версия до 1.3.9 (включительно)
- [1.3.5] - допустима **только** версия 1.3.5

scope зависимостей

Зависимости являются артефактами со стандартными координатами, а так же для них указывается дополнительный параметр - scope

scope может принимать одно из следующих значений:

- compile (default)
  - обычные библиотеки
- test
  - junit и т.п.
- provided
  - servlet-api
- runtime
  - log4j
- system (следует избегать)

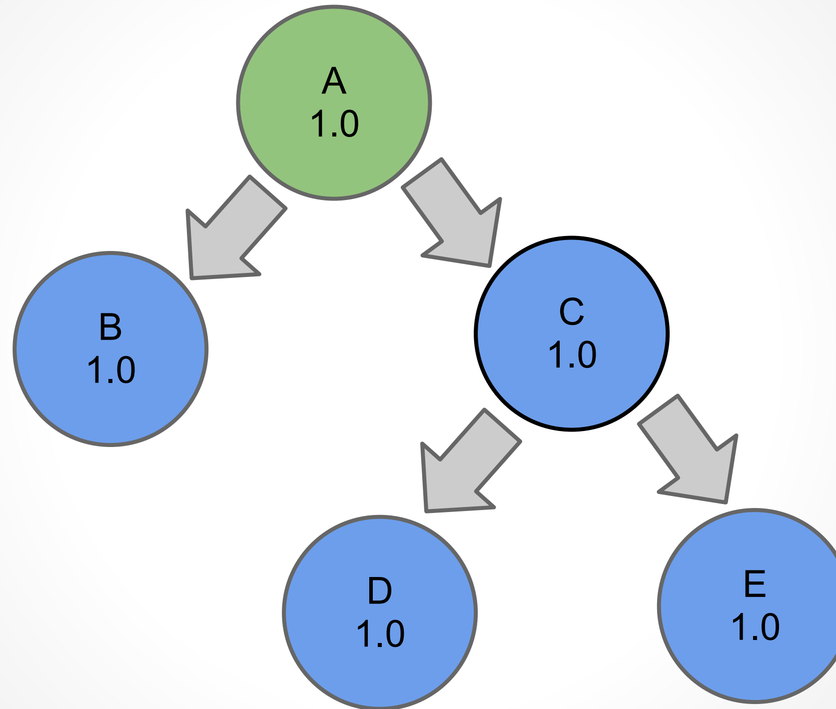
optional

Необязательный атрибут (по умолчанию false). Если данный аттрибут true и наш артефакт используется сам как зависимость в другом проекте, то данная зависимость не будет учитываться при транзитивном разрешении зависимостей этого проекта:

project A -> lib-A (optional = false)

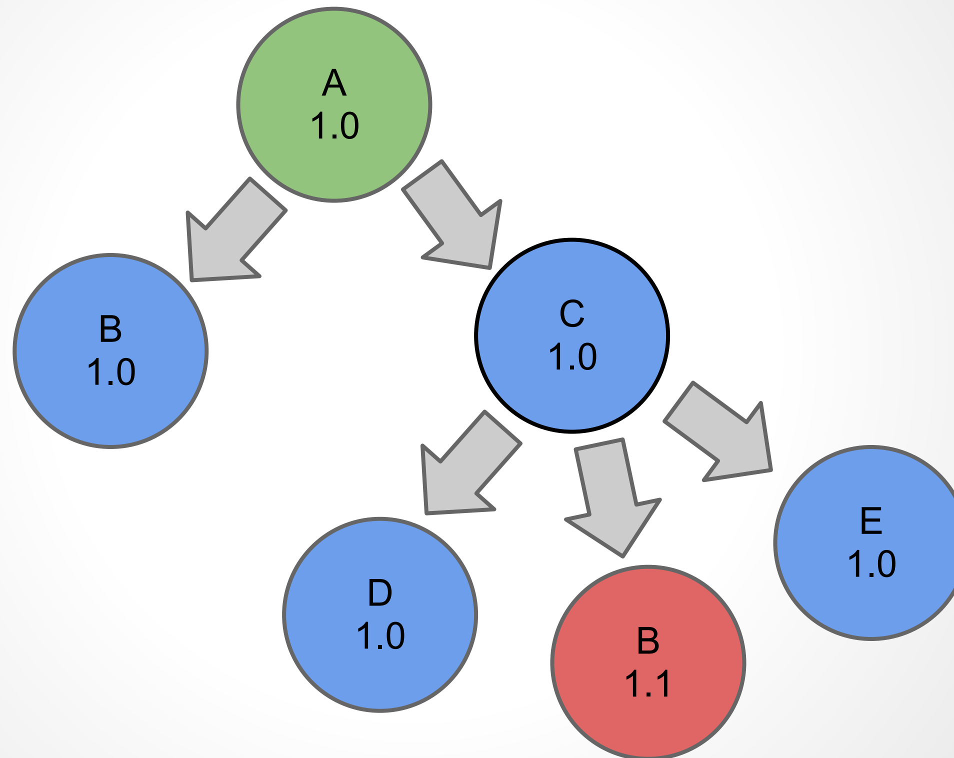
project B -> project A (lib-A не будет использована)

Транзитивные зависимости



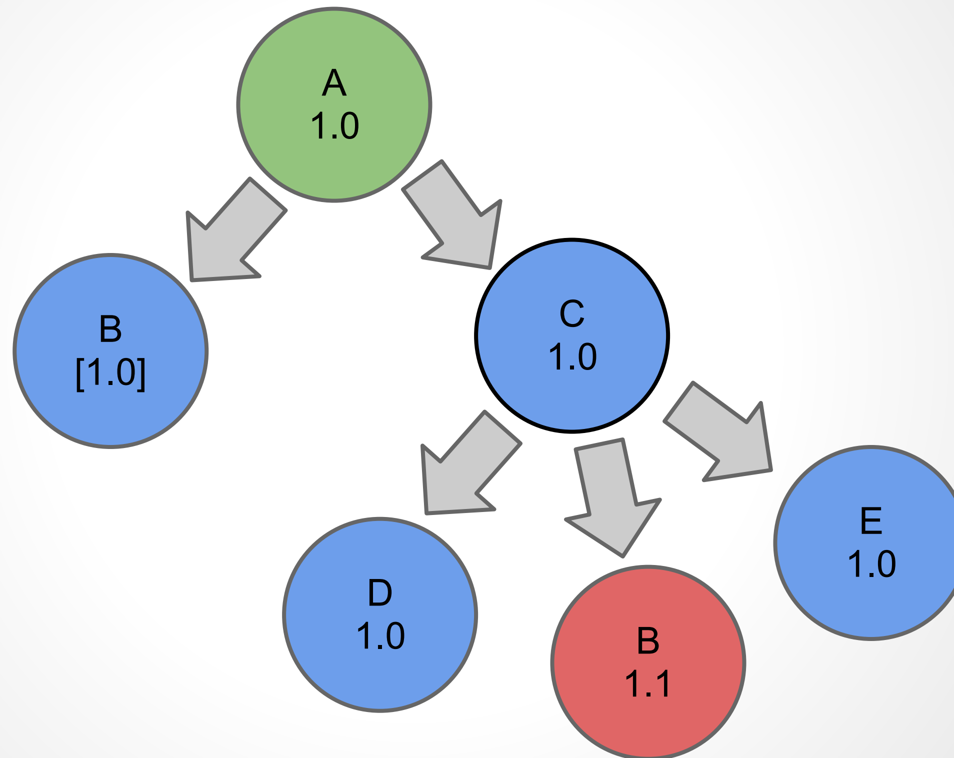
Если наш проект A зависит от двух библиотек B и C, а библиотека C, в свою очередь, зависит от библиотек D и E, то Maven автоматически загрузит и будет использовать библиотеки B, C, D и E.

Транзитивные зависимости 2



В данном случае все как в предыдущем примере, только теперь библиотека C зависит от библиотеки B версии 1.1 . Если в POM проекта A не указано жесткое требование версии библиотеки B 1.0 или младше то при разрешении зависимостей для всего проекта будет использоваться библиотека B версии 1.1 .

Транзитивные зависимости 3



Ой! Теперь наш проект A требует библиотеку B версии 1.0 и ТОЛЬКО 1.0 . А библиотека C требует библиотеку B версии 1.1 или выше... Сборка у нас сломается. Получили конфликт версий при разрешении транзитивных зависимостей.

Dependency exclusions

один из способов побороть проблему конфликта версий использовать dependency exclusions (необходимо понимать что делаешь!)

```xml
<dependencies>
  <dependency>
    <groupId>com.moduleB</groupId>
    <artifactId>project-b</artifactId>
    <version>[1.0]</version>
  </dependency>
  <dependency>
    <groupId>com.company</groupId>
    <artifactId>project-c</artifactId>
    <exclusions>
      <exclusion>
        <groupId>com.company</groupId>
        <artifactId>project-b</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
</dependencies>
```

Ну и напоследок - да, нет ничего идеального в мире ;-)

maven sucks

Поиск

Результатов: примерно 524 000 (0,16 сек.)

Все результаты
Картинки
Карты
Видео
Новости
Ещё

Москва
Изменить место

Весь Интернет
Только на русском
Перевод результатов

Все результаты
Похожие запросы

Показать настройки

top ten reasons why **maven sucks**
betarelease.github.com/.../top-ten-reasons-w... - Перевести эту страницу
top ten reasons why **maven sucks**. β on: 01 Jun 2009. 10. maven corrupts – software, people. 9. maven uses an archetype(appfuse) and expects that all your ...

Software Maven: Top 10 reasons why **Maven sucks**...not!
software**mavens**.blogspot.com/.../top-10-rea... - Перевести эту страницу
25 Jan 2010 – Top 10 reasons why **Maven sucks**...not! I'm having a hard time understanding all the Maven bashing that takes place these days. Maven has ...

Maven Hate - 68% People Agree (641 opinions)
amplicate.com/hate/**maven** - Перевести эту страницу
Maven is symbolic of all the things that **suck** about Java. Bloated, verbose, slow, arcane, crumbling under the weight of its legacy and not keeping up with the ...

Let's Push Things Forward: Why **Maven Sucks** (as compared to ...
lptf.blogspot.com/.../why-**maven-sucks**-as-c... - Перевести эту страницу
20 Feb 2008 – [Carapetyan, 2008] (**Maven** evangelist) Do they have a chapter called "Why do the repositories **suck** so much?" :) [TSS Comment on free **Maven** ...

[PDF] **Maven sucks** – No(w) really
www.iternum.com/.../**maven**.../document.pd... - Перевести эту страницу
Формат файлов: PDF/Adobe Acrobat - Быстрый просмотр
MAVEN SUCKS –. NO(W) REALLY. 26.01.2009. Building Projects with Maven vs. Ant by Karl Banke. In the past few years Maven has surpassed Ant as the build ...

Спасибо за ~~терпение~~ внимание !