МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

Высшего профессионального образования «Саратовский государственный

Технический университет имени Гагарина Ю.А.»

Реферат

по дисциплине

«Современные проблемы информатики и вычислительной техники»

на тему

«Фреймворк для автоматизации сборки проектов Apache Maven»

Выполнила

студентка группы мИВЧТ-21

№ зачетной книжки 111093

Белоножкина Ирина Дмитриевна

Проверил

кандидат физико-математических наук, доцент кафедры ИСТ

Кинцель Дмитрий Александрович

Саратов 2013

**Содержание**

[1 Краткое описание инструмента Apache Maven и его назначение 3](#_Toc345606930)

[2 Основные специализированные термины 7](#_Toc345606931)

[2.1. Плагины 7](#_Toc345606932)

[2.2. Стандартный жизненный цикл проекта 7](#_Toc345606933)

[2.3. Зависимости 8](#_Toc345606934)

[2.4. Репозитории 9](#_Toc345606935)

[2.5. Профили 11](#_Toc345606936)

[3 Инструкция по установке Apache Maven 15](#_Toc345606937)

[4 Инструкция по использованию основных возможностей Apache Maven 17](#_Toc345606938)

[Список спользуемых источников 20](#_Toc345606939)

# Краткое описание инструмента Apache Maven и его назначение

Apache Maven — фреймворк для автоматизации сборки проектов, специфицированных на XML-языке POM (англ. Project Object Model).

Слово maven происходит из языка Идиш и означает примерно «собиратель знания».[3]

Официальный сайт проекта - <http://maven.apache.org>.

Maven - это инструмент для сборки Java проекта: компиляции, создания jar, создания дистрибутива программы, генерации документации. Простые проекты можно собрать в командной строке. 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 зависимости от платформы и упростить написание скрипта используют инструменты для сборки проекта.

Для платформы Java существуют два основных инструмента для сборки: Ant и Maven.

Основные преимущества Maven:

• Независимость от OS. 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. Что 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 с средами разработки. Основные среды разработки на java легко открывают проекты которые собираются c 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. Настроечный файл среды разработки и для сборки один и тот же - меньше дублирования данных и соответственно ошибок.

• Декларативное описание проекта.

Maven был создан канадцем Джейсоном ван Зилом (Jason van Zyl) и организованной им фирмой Sonatype. Он начался как подпроект Apache Turbine в 2002 г.. В 2003 году Maven был квалифицирован как Apache-проект верхнего уровня, тогда же появилась его первая версия, Maven 1.x. Она была опубликована 13-го июля 2004 как версия 1.0. Это происходило, однако, так быстро, что некоторые частности оказались непродуманными. Например, слишком много конфигурации, проблемы с производительностью.

Поэтому концепция была доработана и с 2005-го года началась параллельная разработка Maven 2.x, которая была сдана в версии 2.0 19-го октября 2005-го года.[13]

Maven 1.x не разрабатывается дальше и ограничивается поддержкой пользователей и устранением ошибок.

Разработка Maven 3.0 началась в 2008 г.. После восьми альфа-релизов, первая бета-версия Maven 3.0 была опубликована в октябре 2010 г. Особенное внимание было уделено её обратной совместимости с Maven 2. Для большинства проектов переход от 2-й к 3-й версии не требует никаких изменений.

Дальнейшая разработка Maven-а происходит в следующих подпроектах:

* Maven 1.x и Maven 2.x поддерживают старые версии Maven-а.
* Maven 3 развивает текущую линию продуктов Maven.
* Plugins разрабатывает большинство Maven-плагинов.
* Shared Components изготовляет компоненты програмного обеспечения, которые могут использоваться всеми другими подпроектами.
* Ant Tasks позволяет использовать возможности Maven-а из Ant-скриптов.
* Doxia это фреймворк для генерации контента из форматов Almost Plain Text (APT), Confluence, DocBook, FML (FAQ Markup Language), LaTeX, Rich Text Format (RTF), TWiki, XDoc и XHTML.
* SCM (Source Code Management) разрабатывает программное обеспечение для подключения Apache к различным системам версионирования как CVS или Subversion.
* Surefire разрабатывает тест-фреймворк для Maven-а.
* Wagon готовит абстракцию коммуникационных протоколов как «доступ к файлам», HTTP или FTP.

Maven, в отличие от другого сборщика проектов Apache Ant, обеспечивает декларативную, а не императивную сборку проекта. То есть, в файлах проекта pom.xml содержится его декларативное описание, а не отдельные команды. Все задачи по обработке файлов Maven выполняет через плагины.

Информация для программного проекта, поддерживаемого Mavenом, содержится в XML-файле с именем pom.xml (от Project Object Model). При исполнении Maven проверяет прежде всего, содержит ли этот файл все необходимые данные и все ли данные синтаксически правильно записаны.

Конфигурация включает имя проекта, его собственника и его зависимости от других проектов. Возможно, также, конфигурировать индивидуальные фазы процесса построения проекта (build process), реализованные плагинами. Например, можно конфигурировать плагин компилятора так, что он будет использовать определённую версию Java, или специфицировать упаковку проекта даже в случае негативного результата прохождения некоторых тестов.

Maven базируется на Plugin-архитектуре, которая позволяет применять плагины для различных задач (compile, test, build, deploy, checkstyle, pmd, scp-transfer) для данного проекта, без необходимости их в явном виде инсталлировать. Это возможно потому, что информация поступает плагину через стандартный вход, а результаты пишутся в его стандартный выход. Теоретически, это позволяет кому угодно писать плагины для взаимодействия со средствами построения проекта (компиляторы, средства тестирования, и т. п.) для любого другого языка. В реальности, поддержка других языков кроме Java сейчас минимальна. Существует плагин для .NET –фреймворка, а также плагины для C/C++ maven-native и maven-nar

Количество плагинов стало сейчас очень впечатляющим: от плагинов, позволяющих прямо из Maven-а стартовать web-приложение, чтобы тестировать его в браузере, через те, которые позволяют тестировать или создавать банки данных, и до таких, которые генерируют Web Services. Задача разработчика ограничивается нередко только тем, чтобы выявить и применить необходимый плагин.

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

Также добавляется информация, которая не используется самим Mavenом, но нужна для программиста, чтобы понять, о чём этот проект:

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

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

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

Тэг <build> не является обязательным т.к. существуют значения по умолчанию. Этот раздел содержит информацию по самой сборке: где находятся исходные файлы, где ресурсы, какие плагины используются.

Крупные проекты должны быть поделены на несколько модулей, или подпроектов, каждый со своим собственным POM. Можно написать затем корневой POM, через который все модули компилируются единой командой. POM-ы могут наследовать конфигурацию от других POM-ов. Все POM-ы наследуют от Супер POM-а[4] по умолчанию. Супер POM обеспечивает конфигурацию по умолчанию, такую, как структуру каталогов по умолчанию, используемые по умолчанию плагины, и т. п.

Maven поддерживает принцип «соглашения прежде конфигурации» (Convention over Configuration). Поскольку проект придерживается избранной системы соглашений, постольку отпадает необходимость специфицировать их, что сильно упрощает pom.xml. Однако, почти все стандарты, на которые опирается Maven, могут быть изменены индивидуальной конфигурацией.

Одна из реализаций принципа «соглашения прежде конфигурации» - стандартная структура каталогов. Поскольку проект её придерживается — отпадает необходимость специфицировать пути к файлам, что сильно упрощает pom.xml.

Корневой каталог проекта: файл pom.xml и все дальнейшие подкаталоги:

* src: все исходные файлы
* src/main: исходные файлы собственно для продукта
* src/main/java: Java-исходный текст
* src/main/resources: другие файлы, которые используются при компиляции или исполнении, например Properties-файлы
* src/test: исходные файлы, необходимые для организации автоматического тестирования
* src/test/java: JUnit-тест-задания для автоматического тестирования
* target: все создаваемые в процессе работы Mavenа файлы
* target/classes: компилированные Java-классы

# Основные специализированные термины

## Плагины

Большая часть функциональности Maven-а осуществляется плагинами. Плагин обеспечивает достижение ряда целей с помощью следующего синтаксиса:

mvn [имя плагина]:[имя цели].

Например, Java-проект может быть скомпилирован плагином-компиляторомпутем выполнения команды ***mvn compiler:compile***.

Существуют Maven-плагины для построения, тестирования, контроля исходного текста, запуска web-сервера, генерации Eclipse-проектных файлов и множество других. Плагины перечисляются и конфигурируются в <plugins>-секции файла pom.xml. Некоторая базовая группа плагинов включается в каждый проект по умолчанию. Они имеют гибкую конфигурацию по умолчанию.

Для самых распространённых интегрированных сред разработки (IDE) имеются плагины, позволяющие удобно управлять Maven-ом. Их список включает:

* Eclipse
* IntelliJ IDEA
* NetBeans
* JBuilder
* JDeveloper (version 11.1.2)
* MyEclipse

Эти плагины обеспечивают также возможность удобно редактировать POM или использовать POM для полного описания зависимостей проекта для нужд используемого IDE.

## Стандартный жизненный цикл проекта

Жизненный цикл проекта — это список поименованных фаз, определяющий порядок действий при его построении. Maven использует по умолчанию следующий жизненный цикл:

1. Создание темплейта и обработка ресурсов (archetype): На этой фазе разрешаются и, при необходимости, скачиваются из интернета зависимости.

2. Компиляция (compile)

3. Обработка тестовых ресурсов. (Например - скачивается из интернета JUnit-пакет).

4. Компиляция тестов. (Тестирующие классы не передаются конечным пользователям.)

5. Тестирование (test)

6. Упаковка (package). Обычно речь идёт о создании JAR- или WAR-файла.

7. Инсталляция проекта в локальном Maven-репозитории (install). Теперь он доступен как модуль для других локальных проектов.

8. Инсталляция в удаленном Maven-репозитории (deploy). Теперь стабильная версия проекта доступна широкому кругу разработчиков.

Maven имеет также стандартный жизненный цикл для чистки (cleaning) и для генерации его страницы (site). Если бы 'clean' было частью обычного жизненного цикла, проект подвергался бы чистке при каждом построении, что нежелательно.

Стандартные жизненные циклы могут быть существенно дополнены Maven-плагинами и Maven-архетипами (англ. Archetypes). Maven-плагины позволяют вставлять в стандартный цикл новые шаги (например, распределение на сервер приложений) или расширять существующие шаги. Maven-архетипы представляют собой заготовки для различнейших программных пакетов (если они отвечают стандартам Maven-структуры).

Если структура проекта соответствует стандартам Maven-а, то команда ***mvn package*** откомпилирует все Java-файлы, запустит предусмотренные тесты, и упакует поставляемый программный код и ресурсы в target/my-app-1.0.jar (в предположении, что artifactId было определено как 'my-app' и версия — как 1.0.)

Используя собственно Maven, пользователь обеспечивает только конфигурацию своего проекта, так как реальную работу по компиляции проекта, чистке целевых каталогов, выполнению элементных тестов, генерации API-документов и т. д. выполняют конфигурируемые плагины. В общем случае, пользователь не должен сам писать плагины. Сравните это с Ant и make, где для выполнения указанных задач пишутся императивные процедуры.

## Зависимости

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

В файле pom.xml задаются зависимости, которые имеет управляемый Maven-ом пакет от других программных пакетов.

Эти зависимости Maven разрешает, то есть, сначала он проверяет, находятся ли необходимые файлы в локальных каталогах или в локальном Maven-репозитории. Если зависимость не может быть локально разрешена, Maven пытается связаться с конфигурированным для него Maven-репозиторием в интранете или в интернете и копировать необходимые данные оттуда в локальный репозиторий. По умолчанию Maven использует 'Maven Central Repository', но разработчик может конфигурировать и другие публичные Maven-репозитории, такие, как Apache, Ibiblio, Codehaus или Java.Net.

Поиск зависимостей (open Source библиотек и модулей) ведётся по их координатам (groupeId, artefactId и version). Эти координаты могут быть определены с помощью специальных поисковых машин. Введя в такой машине, например, поисковый признак «pop3», вы получите, среди прочих, и строку с groupeId="com.sun.mail" и artefactId="pop3", что выглядит очень прилично. Чаще же результаты будут менее вразумительны и только опытным путем вы сможете определить, что заказали именно тот jar-файл, который хотели.

Принадлежащие фирме и расположенные в её интранете репозитории обычно реализуются с помощью менеджеров репозиториев Maven (Maven Repository Manager), таких как Apache Archiva, Nexus (ранее Proximity), Artifactory, Codehaus Maven Proxy или Dead Simple Maven Proxy.

## Репозитории

Репозитории - это место где хранятся артефакты: jar файлы, pom -файлы, javadoc, исходники.

Существуют:

* + - Локальный репозиторий по умолчанию он распопожен в <home директория>/.m2/repository - персональный для каждого пользователя.
    - центральный репозиторий который расположен в http://repo1.maven.org/maven2/ и доступен на чтение для всех пользоватей в интернете.
    - Внутренний "Корпоративный" репозиторий- дополнительный репозиторий, один на несколько пользователей.

***Локальный репозиторий***

Локальный репозиторий по умолчанию он распопожен в <home директория>/.m2/repository. Здесь лежат артефакты которые были скачаны из центрального репозитория либо добавлены другим способом. Например если вы наберёте команду ***mvn install*** в текущем проекте, то соберётся jar (или war, pom в зависимости от содержимого тэга packaging) который установится в локальный репозиторий. Найти его можно в <home директория>/.m2/repository/<groupIdPath>/<artifactId>/<version>/<artifactId>-<version>.jar где groupIdPath получается заменой всех точек на слеши. Например для проекта

<groupId>ru.apache-maven</groupId>

<artifactId>site</artifactId>

<version>1.0-SNAPSHOT</version>

jar файл храниться по пути: <home директория>/.m2/repository/ru/apache-maven/site/1.0-SNAPSHOT/site-1.0-SNAPSHOT.jar

***Центральный репозиторий***

Чтобы самому каждый раз не создавать репозиторий, сообщество для Вас поддерживает центральный репозиторий. Если для сборки вашего проекта не хватает зависимостей, то они по умолчанию автоматически скачиваются с http://repo1.maven.org/maven2. В этом репозитории лежат практически все опенсорсные фреймворки и библиотеки.

Самому в центральный репозиторий положить нельзя. Т.к. этот репозиторий используют все то перед тем как туда попадают артефакты они проверяются, тем более что если артефакт однажды попал в репозиторий, то по правилам изменить его нельзя.

Для поиска нужной библиотеки очень удобно пользоваться сайтами http://mavenrepository.com/ и http://findjar.com/

***Корпоративный репозиторий***

Если вы хотите создать свой репозиторий, содержимое которого вы можете полностью контролировать(как локальный), и сделать так, чтобы он был доступен для нескольких человек, вам будет полезен корпоративный репозиторий. Доступ к артефактам можно ограничивать настройками безопасности сервера так, что код ваших проектов не будет доступен из вне.

Чтобы добавить репозиторий в список откуда будет скачиваться зависимости, нужно добавить секцию repositories в pom.xml, например:

<project>

...

<repositories>

<repository>

<id>my-company-repo</id>

<url>http://my-company-site.ru/repo</url>

</repository>

</repositories>

...

</project>

Существуют несколько реализаций серверов - репозиториев maven. Наиболее известные это artifactory, continuum, nexus.

## Профили

Maven изначально создавался принимая во внимание портируемость в разных окружениях Но часто приложение приходится cобирать и запускать в разном окружении: например для разработки используется одна база данных, в рабочем сервере используется другая. При этом могут понадобиться разные настройки, разные зависимости и плагины. Для этих целей в maven используются профайлы.

Определим два профайла: один для разработки, другой для производственного сервера. Для разработки вполне подойдёт база hsqldb которая хранит все данные в памяти. На производственном сервере же используется база данных postgres которая сохраняет все данные на диск. В профайлах для каждой конфигурации определены свои проперти database.url и зависимости для разных jdbc драйверов.

Ниже приведён пример объявления таких профайлов.

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

<project>

<profiles>

(1) <profile>

<id>development</id>

<properties>

<database.url>jdbc:hsqldb:mem:testdb</database.url>

</properties>

<dependencies>

<dependency>

<groupId>org.hsqldb</groupId>

<artifactId>hsqldb</artifactId>

<version>2.0.0</version>

</dependency>

</dependencies>

</profile>

(2) <profile>

<id>productionServer</id>

<properties>

<database.url>jdbc:postgresql://databseserver/database</database.url>

</properties>

<dependencies>

<dependency>

<groupId>postgresql</groupId>

<artifactId>postgresql</artifactId>

<version>9.0-801.jdbc4</version>

</dependency>

</dependencies>

</profile>

</profiles>

.....

</project>

Цифрами 1 и 2 обозначены начала объявления профайлов. Каждый профайл имеет идентификатор в данном случае development и productionServer.

Внутри тэга <profile> содержатся все те же объявления что и внутри <project>: properties, dependencies, и др. Вот полный список тегов которые могут содержаться внутри профайлов:

<repositories>

<pluginRepositories>

<dependencies>

<plugins>

<properties>

<modules>

<reporting>

<dependencyManagement>

<distributionManagement>

<build> тэг, который может содержать

<defaultGoal>

<resources>

<testResources>

<finalName>

Чтобы содержимое тэга профайла "работало" нужно профайл активировать. Когда профайл активирован его содержимое объединяется с основной частью pom.xml Нужно заметить что активных профилей одновременно может быть несколько.

Активировать профайл можно несколькими способами:

* + - Во первых, это можно задать вручную в командной строке запуска maven, например: mvn package -P production
    - Во вторых, при объявлении самого профайла можно задать тэг <activation>, который определяет профайл будет активирован: в нашем примере профайл development активный по умолчанию: <activation><activeByDefault>true</activeByDefault></activation>. Кроме активации по умолчанию можно задать активацию на основе операционной системы, установленных переменных окружения, версии JDK.

В коммандной строке можно задать какие профили будут деактивированы: ***mvn goal -P !profile-1,!profile-2***.

Активные профайлы можно также задать в ~/.m2/settings.xml

<settings>

...

<profiles>

<profile>

<id>appserverConfig</id>

<properties>

<appserver.home>/path/to/appserver</appserver.home>

</properties>

</profile>

</profiles>

<activeProfiles>

<activeProfile>appserverConfig</activeProfile>

</activeProfiles>

...

</settings>

Чтобы проверить работу и возможно найти ошибки полезны следующие плагины:

Чтобы показать какие профайлы сейчас активны можно набрать:

***mvn help:active-profiles,*** причём опции командной строки принимаются во внимание. Команда ***mvn help:active-profiles -P productionServer,development***  выведет как и положено

The following profiles are active:

- development (source: pom)

- productionServer (source: pom)

Также можно посмотреть pom полученный после объединения основной части и активных профайлов:

mvn help:effective-pom -P productionServer

# Инструкция по установке Apache Maven

Для установки инструмента необходимо зайти на официальный сайт в раздел ***Загрузка*** и скачать последнюю стабильную версию.

Архив распаковывается в инсталляционную директорию. Например в C:\Program Files\maven\ в Windows или /opt/maven в Linux.

Установка переменной окружения M2\_HOME:

В Windows щелчок правой кнопкой мыши на "мой компьютер" ->свойства->дополнительные параметры->переменные среды->системные переменные -> добавить "M2\_HOME" и " C:\Program Files\maven\" .

В Linux можно добавить строку "export M2\_HOME=/opt/maven"в файл /etc/profile .

Для установки переменной окружения PATH В Windows в переменной PATH необходимо добавить к списку директорий строку %M2\_HOME%\bin". В Linux можно добавить строку "export PATH=$PATH:$M2\_HOME/bin"в файл /etc/profile .

Для проверки корректности установки используется в команда ***mvn –version*** в командной строке.

Если результат примерно такой

***dima@myhost ~ $ mvn -version***

***Apache Maven 3.0 (r1004208; 2010-10-04 15:50:56+0400)***

***Java version: 1.6.0\_22***

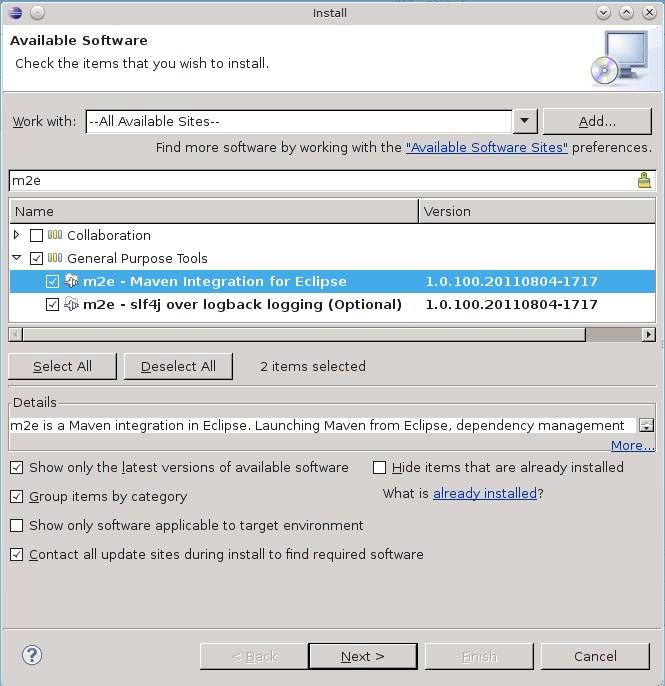
***Java home: /opt/sun-jdk-1.6.0.22/jre***

***Default locale: ru\_RU, platform encoding: UTF-8***

***OS name: "linux" version: "2.6.34-gentoo-r12" arch: "amd64" Family: "unix",***

то Maven успешно установлен.

Для установки плагина в eclipse необходимо зайти в меню Help->Install New Software, выбрать Work with --All Available Sites--, в фильтре набрать m2e и отметить плагины как показано на рисунке:



После установки плагина можно:

• Создавать проекты через File->New Progect->Maven->Maven Project;

• Открыть существующий проект File->Import->Maven->Existing Maven Projects.

# Инструкция по использованию основных возможностей Apache Maven

**Создание нового проекта**

Новый проект создается при помощи команды ***mvn archetype:generate***.

Выполнив эту команду, maven покажет список шаблонов(архетипов) для разных проектов. После выбора проекта и его версии по умолчанию и нажатия Enter, Enter, когда команда спросит groupId и artifactId, необходимо ввести данные:

***ru.apache\_maven***

***testproject1***

В результате сгенерируется проект со стандартной структурой директорий

testproject1

|-- pom.xml

`-- src

|-- main

| `-- java

| `--ru

| `-- ru.apache\_maven

| `-- App.java

`-- test

`-- java

`-- ru

`-- ru.apache\_maven

`-- AppTest.java

с примером приложения и с JUnit тестом для этого приложения. pom.xml - это файл для maven.

Чтобы скомпилировать, нужно набрать в консоли mvn compile. Если в консоль выведется:

***...***

***[INFO] BUILD SUCCESS***

***.....*** ,

то компиляция прошла успешно и в созданной директории target/classes будут class файлы c нашей программой.

Если набрать ***mvn package*** в директории target будет создан jar файл testproject1-1.0-SNAPSHOT.jar

Для запуска скомпилированной программы используется следующая команда:

[name@myhost testproject1]$ java -cp ./target/classes ru.apache\_maven.App

Полученный проект можно сразу открывать средой разработки:

* для среды IntelliJ Idea проект открывается сразу ("File/OpenProject")
* для среды NetBeans проект открывается сразу ("Файл/Создать проект/Проект maven с существующим POM")
* для eclipse нужно предварительно установить соответствующий плагин.

**Создание проекта с помощью archetype:generate**

Для генерации web приложение с помощью плагина maven-archetype:plugin используется команда ***mvn archetype:generate***.

В результате выводится список архитипов, и maven предлагает выбрать. Выбирать в таком списке довольно сложно, поэтому будет удобно его отфильтровать командой **maven-archetype-webapp** и maven предложит список из нескольких архитипов.

Choose archetype:

1: internal -> org.apache.maven.archetypes:maven-archetype-webapp (A simple Java web application)

Необходимо выбрать [1] org.apache.maven.archetypes:maven-archetype-webapp и написать

groupId : ru.apache-maven

artifactId : webapptest

Остальные значения можно оставить по умолчанию.

В результате получится проект с такой структурой директорий:

├── pom.xml

└── src

└── main

├── resources

└── webapp

├── index.jsp

└── WEB-INF

└── web.xml

**Сборка проекта с помошью mvn package**

В результате в директории target образуется testwebapp.war. Он запускается напрямую командой ***mvn tomcat:run.***

В этом случае запустится tomcat и приложение будет сразу доступно по адресу http://localhost:8080/webapptest/

# Список спользуемых источников

1. http://maven.apache.org/guides/introduction/introduction-to-the-pom.html/ POM
2. http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#Lifecycle\_Reference Maven Build Lifecycle Reference]
3. Алексей Литвинюк Автоматизация процесса разработки Java–приложений с помощью Maven 2 . «Компьютерная газета».
4. Выговский Леонид Установка maven-репозитория Artifactory (5 января 2010)
5. Нил Санче Погружение в EJB Web-приложения с Geronimo). «IBM developerWorks Россия» (19 июля 2007). — Совместное применение Maven, Struts и XDoclet.