

№ 14 Maven

Задание

1. Установите maven. Настройте переменные среды и проверьте, что maven настроен корректно.
2. Переделайте проект л.р. 11-12 +13 под maven. Добавьте pom.xml
3. Изучите структуру pom.xml . Добавьте необходимые зависимости (на MySQL database driver, Log4j, junit и т.д.) и плагины. Произведите сборку проекта.
4. Изучите и продемонстрируйте основные фазы сборки проекта с консоли (compile, test, package, deploy и т.д.). Выполните сборку проекта с различными фазами жизненного цикла сборки
5. Создайте многомодульный проект. Один модуль - jar, другой модуль – war. Создайте parent модуль. Проверьте процесс сборки.
6. Подготовьте профили для своего проекта.

Вопросы

1. Что такое Maven? Для чего создан Maven?
2. Как узнать какую версию Maven вы используете?
3. Какая структура каталогов в Maven?
4. Что такое pom.xml?
5. Какую информацию содержит pom.xml?
6. Что такое супер POM?
7. Какие элементы необходимы для минимального POM?
8. Что такое артефакт? Что является полным именем артефакта?
9. Что такое зависимости в Maven?
10. Что такое плагин в Maven?
11. Что такое задача в Maven?
12. Что такое репозиторий в Maven? Какие типы репозитория существуют в Maven?
13. Какой порядок поиска зависимостей Maven?
14. Назовите основные фазы жизненного цикла сборки Maven?
15. Что делает команда mvn site? Что делает команда mvn clean?
16. Из каких фаз состоит жизненный цикл сборки Default (Build)?
17. Какие типы плагинов существуют в Maven?
18. Когда Maven использует внешние зависимости?
19. Какая команда создает новый проект на основе архетипа?
20. Что такое транзитивная зависимость в Maven?
21. Перечислите теги pom.xml.

Теоретическая часть

НАСТРОЙКА APACHE MAVEN ДЛЯ WINDOWS

ApacheMaven – это программный продукт, предназначенный для упрощения сборки проектов, пришедший на смену ApacheAnt.

Основные его достоинства следующие:

- Делает процессы сборки приложения простым
- Предоставляет унифицированную систему сборки
- Предоставляет информацию о качестве проекта
- Предоставляет описание лучших практик в разработке
- Позволяет производить прозрачную и понятную миграцию новых компонентов.

Maven – это программный продукт с интерфейсом командной строки и должен быть интегрирован с переменными среды Windows.

Официальный сайт Mavena <http://maven.apache.org>.

После завершения скачивания Maven, распакуйте архив в любую папку. Для Windows, как правило, путь к папке не должен содержать пробелов

Для начала использования Maven необходимо настроить переменные среды Windows. **M2_HOME** переменная должна быть установлена и **PATH** переменная должна быть модифицирована для включения папки, откуда запускается Maven.

Вы можете установить переменные среды вызовом «Настройки Системы» (System), используя «Панель Управления» (ControlPanel).

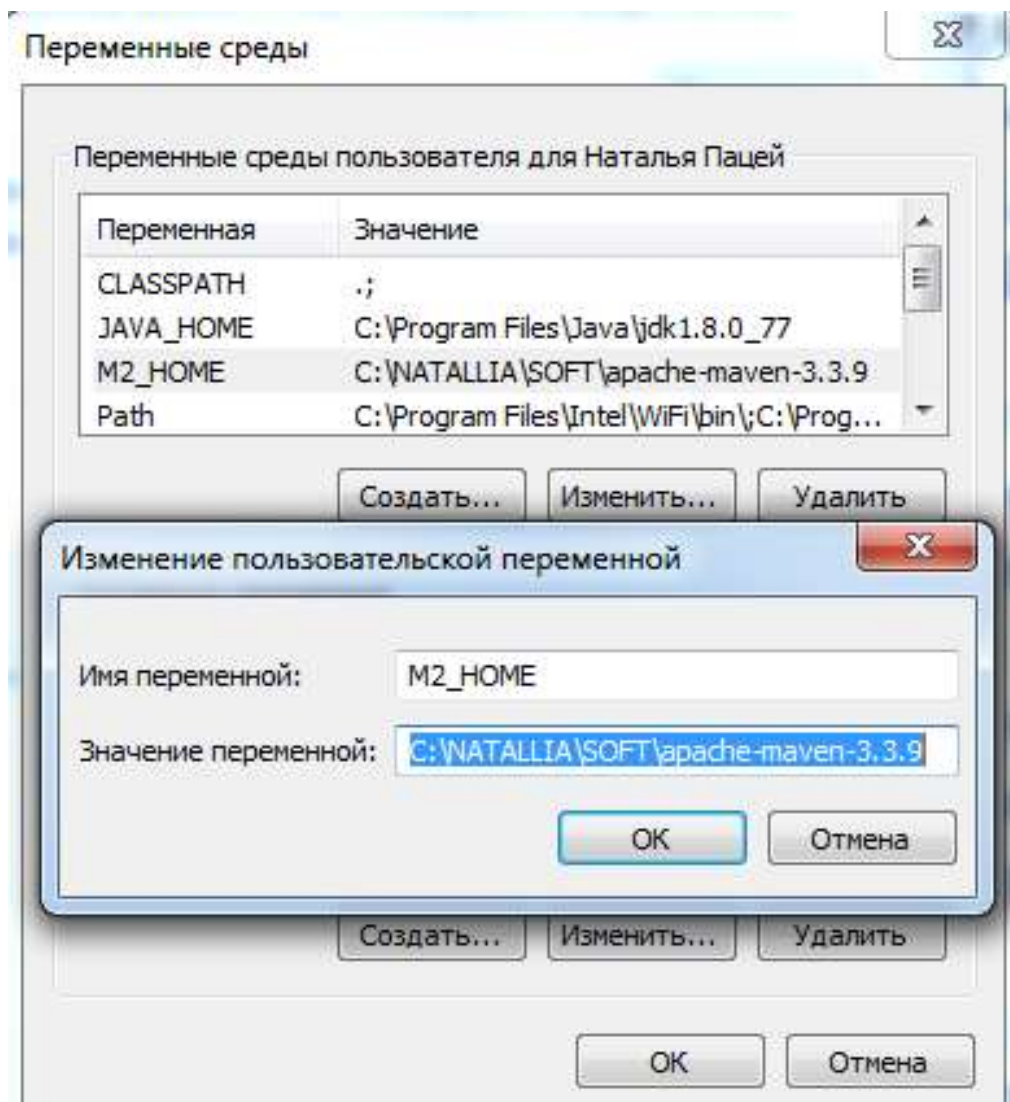


Рисунок Установка переменной M2_HOME

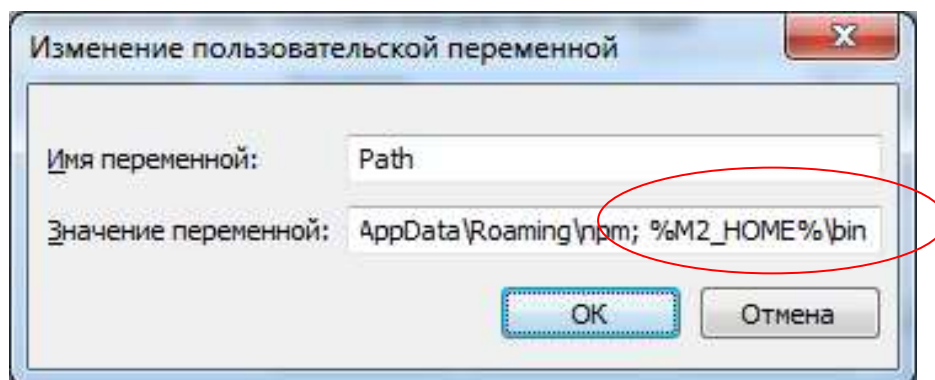


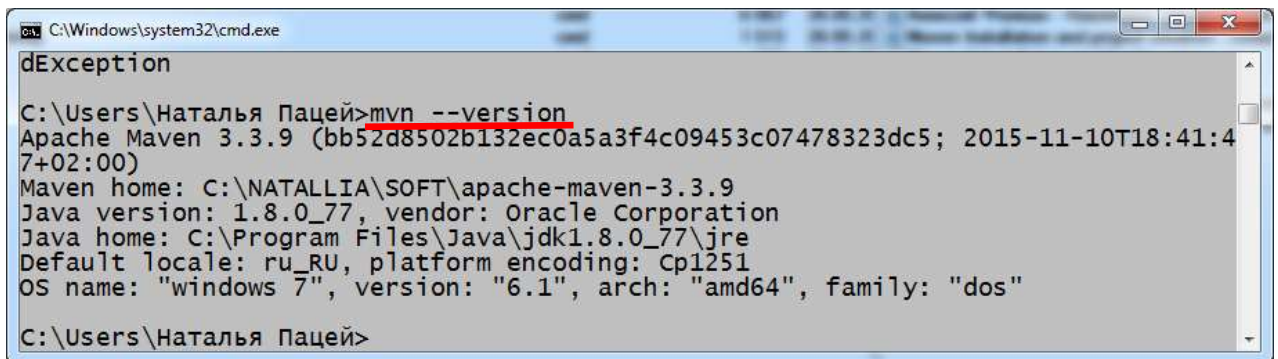
Рисунок Изменение переменной PATH.

Создайте новую переменную среды M2_HOME, указывающую путь к папке, куда был распакован Maven. Например для M2_HOME - **d:\opt\maven**. Переменная среды Path должна уже существовать. Она должна быть изменена, добавлением следующего текста в начало:

```
%M2_HOME%\bin;
```

ApacheMaven, теперь, готов к использованию. Он, также, доступен для интеграции с IDE и другими программными средствами, предназначенными для разработки.

Прежде чем использовать Apache Maven, вам необходимо проверить, как проверить установлен ли maven на вашем компьютере. Это можно сделать с помощью следующей команды:

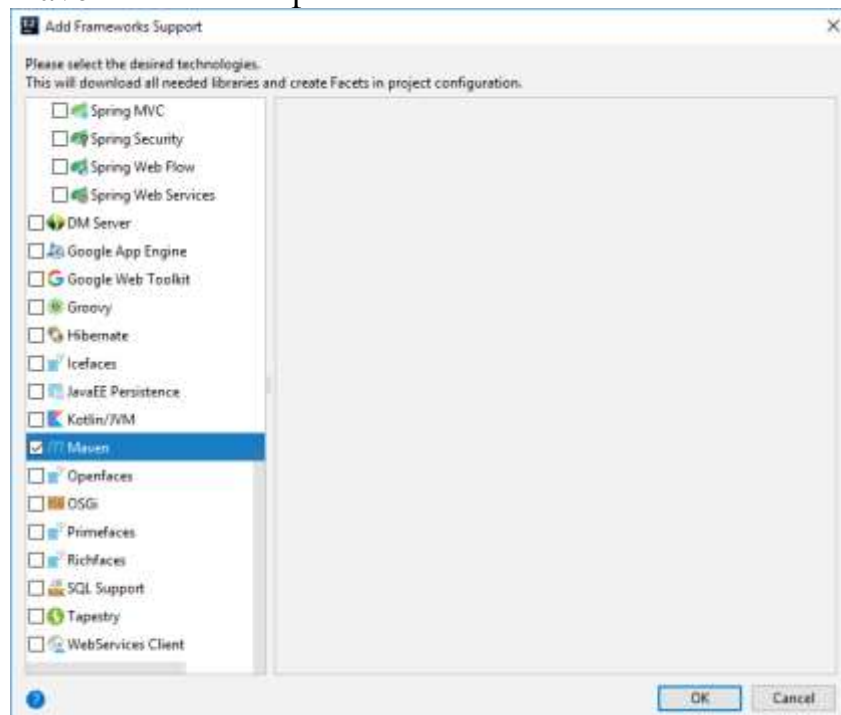


```
C:\Windows\system32\cmd.exe
dException
C:\Users\Наталья Пацей>mvn --version
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-10T18:41:47+02:00)
Maven home: C:\NATALLIA\SOFT\apache-maven-3.3.9
Java version: 1.8.0_77, vendor: Oracle Corporation
Java home: C:\Program Files\Java\jdk1.8.0_77\jre
Default locale: ru_RU, platform encoding: Cp1251
OS name: "windows 7", version: "6.1", arch: "amd64", family: "dos"
C:\Users\Наталья Пацей>
```

Рисунок Проверка версии maven

Если результат похожий напредставленный выше, тогда вы можете быть уверенными, что maven установлен и готов к использованию. Если же ваша операционная система не может найти команду **mvn**, тогда проверьте, что переменные среды **path** и **M2_HOME** установлены корректно.

7. Добавьте поддержку Maven к текущему проекту
Щелкните по проекту и выберите → Add frameworks support. Найдите maven поставьте флаг.



Поменяйте структуру папок проекта (это нужно для корректной сборки)

В корне должен появиться файл pom.xml следующего содержания:

```

<?xml version="1.0" encoding="UTF-8"?>
<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">
  <modelVersion>4.0.0</modelVersion>

  <groupId>by.patsei</groupId>
  <artifactId>Servlet</artifactId>
  <version>1.0-SNAPSHOT</version>

</project>

```

Здесь:

GroupId параметр, используемый для определения иерархического расположения проекта в Maven **репозитории**.

В этом случае, **репозиторий** – это локальный репозиторий Maven, расположенный на вашем компьютере. **GroupId** определяет типичный каталог верхнего уровня, и это может быть использовано в нескольких проектах в организации.

ArtifactId идентифицирует ваш проект и **version** указывает версию проекта. **ArtifactId** используется, когда артефакты распакованные в репозитории используются как зависимость в других проектах.

Создайте раздел с существующими зависимостями. Их можно копировать из репозитория:

→ ↻ <https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api/4.0.1>

Maven Repository

Search for groups, artifacts, categories

Indexed Artifacts (14.0M)

Popular Categories

- Act Oriented
- Frameworks
- Integration Metrics
- Tools
- Code Libraries
- Command Line Parsers
- API Implementations
- Cloud Computing
- Analysers
- Actions
- Configuration Libraries
- Utilities
- Build and Time Utilities
- Dependency Injection
- Embedded SQL Databases

Home » [javax.servlet](#) » [javax.servlet-api](#) » 4.0.1

Java Servlet API » 4.0.1

Java Servlet API

License	CDDL GPL 2.0
Categories	Java Specifications
Organization	GlassFish Community
HomePage	https://javaee.github.io/servlet-spec/
Date	(Apr 20, 2018)
Files	pom (15 KB) jar (93 KB) View All
Repositories	Central
Used By	10,499 artifacts

Maven [Gradle](#) [SBT](#) [Ivy](#) [Grape](#) [Leiningen](#) [Buildr](#)

```
<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>4.0.1</version>
  <scope>provided</scope>
</dependency>
```

☒ Include comment with link to declaration

Должно получиться как то так

```
<?xml version="1.0" encoding="UTF-8"?>
<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">
  <modelVersion>4.0.0</modelVersion>

  <groupId>by.patsei</groupId>
  <artifactId>Servlet</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>

  <dependencies>

    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>4.0.1</version>
      <scope>provided</scope>
    </dependency>

    <!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>jstl</artifactId>
      <version>1.2</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.projectlombok/lombok-maven
-->
    <dependency>
```

```

        <groupId>org.projectlombok</groupId>
        <artifactId>lombok-maven</artifactId>
        <version>1.18.6.0</version>
        <type>pom</type>
    </dependency>
    <!-- https://mvnrepository.com/artifact/log4j/log4j -->
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
    </dependency>

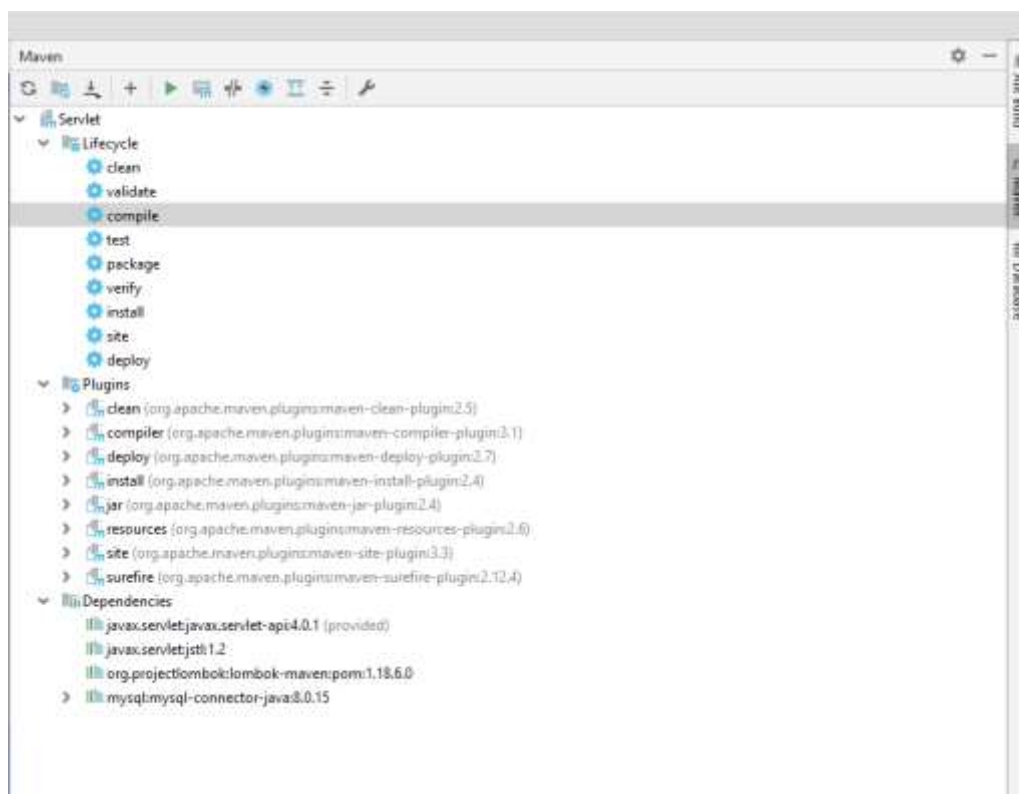
    <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.15</version>
    </dependency>

</dependencies>
<build>
    <finalName>FinalWebApplication</finalName>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.3</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
    </plugins>
</build>

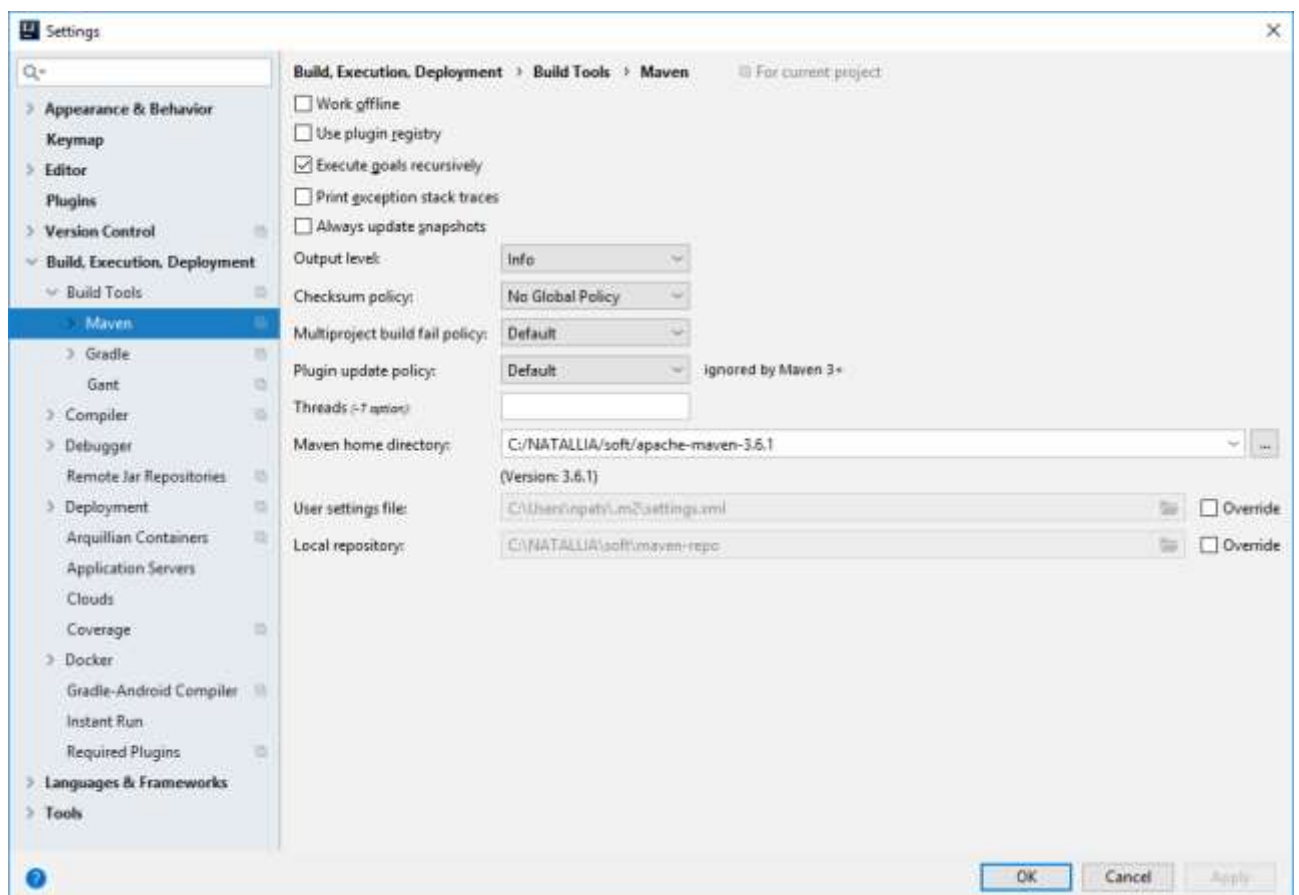
</project>

```

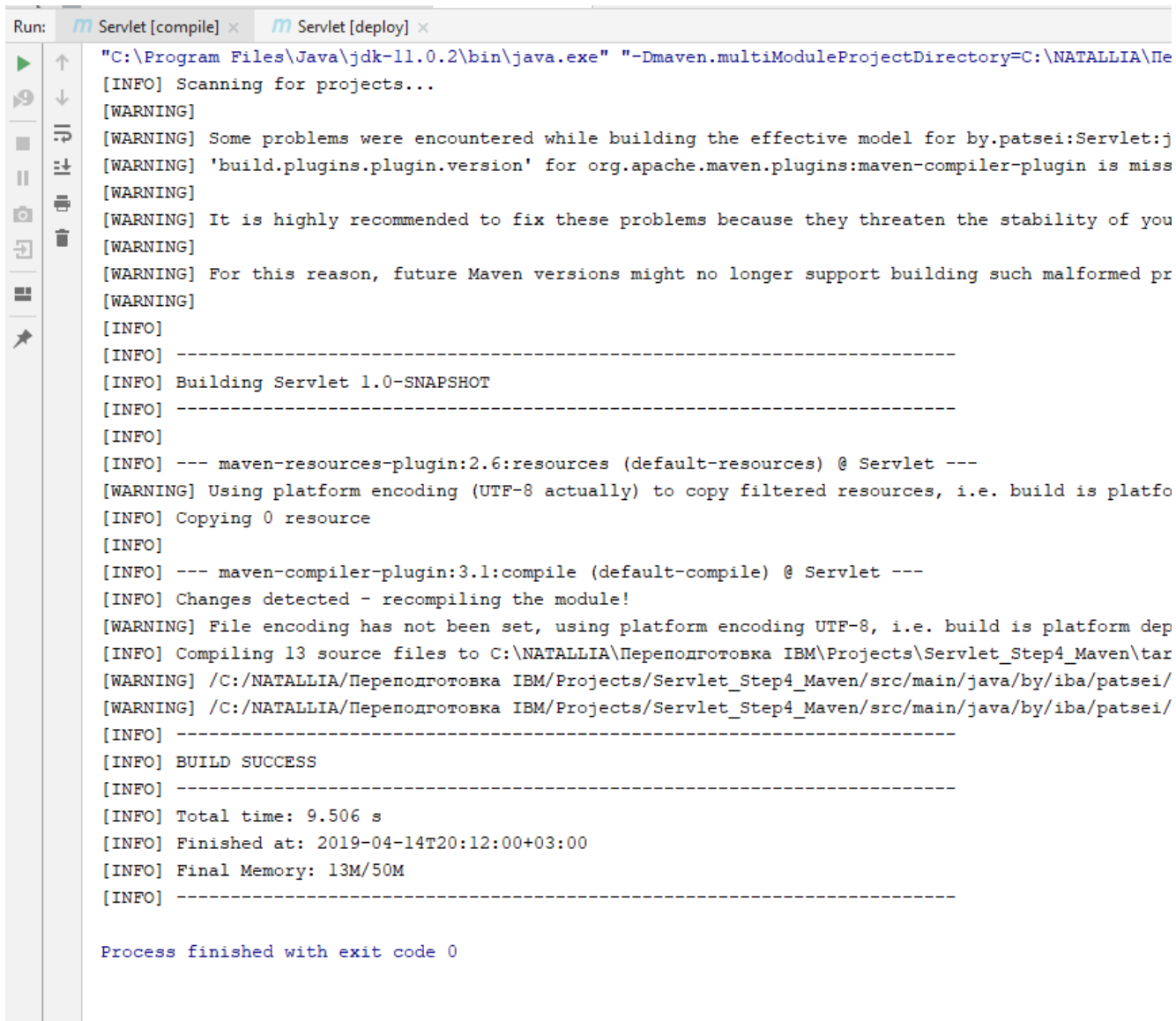
Удалите старые зависимости.
Изучите фазы жизненного цикла maven



IntelliJ IDEA может использовать собственный maven. Убедитесь, что установлена ссылка на ваш установленный



Попробуйте выполнить deploy



```
Run: m Servlet [compile] x m Servlet [deploy] x
"C:\Program Files\Java\jdk-11.0.2\bin\java.exe" "-Dmaven.multiModuleProjectDirectory=C:\NATALLIA\Пр
[INFO] Scanning for projects...
[WARNING]
[WARNING] Some problems were encountered while building the effective model for by.patsei:Servlet:j
[WARNING] 'build.plugins.plugin.version' for org.apache.maven.plugins:maven-compiler-plugin is miss
[WARNING]
[WARNING] It is highly recommended to fix these problems because they threaten the stability of you
[WARNING]
[WARNING] For this reason, future Maven versions might no longer support building such malformed pr
[WARNING]
[INFO]
[INFO] -----
[INFO] Building Servlet 1.0-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ Servlet ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platfo
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ Servlet ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform dep
[INFO] Compiling 13 source files to C:\NATALLIA\Переподготовка IBM\Projects\Servlet_Step4_Maven\tar
[WARNING] /C:/NATALLIA/Переподготовка IBM/Projects/Servlet_Step4_Maven/src/main/java/by/iba/patsei/
[WARNING] /C:/NATALLIA/Переподготовка IBM/Projects/Servlet_Step4_Maven/src/main/java/by/iba/patsei/
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 9.506 s
[INFO] Finished at: 2019-04-14T20:12:00+03:00
[INFO] Final Memory: 13M/50M
[INFO] -----

Process finished with exit code 0
```

По завершению вы должны увидеть следующее:

```
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
```

Новый проект **ApacheMaven** может быть создан «вручную» с `pom.xml` и папками, созданными или сгенеренными в соответствии с архитектурой Maven проекта.

Maven создал **папку для проекта**, содержащую главный **`pom.xml`** и папку для исходных кодов **`src`**, который содержит подпапки для приложения и папки для тестовых исходных кодов.

- Изучите и продемонстрируйте основные фазы сборки проекта с консоли (terminal) (compile, package, deploy и т.д.). Выполните сборку проекта с различными фазами жизненного цикла сборки

Сделайте install и найдите ваш артефакт в локальном репозитории.

- Работа с maven с консоли.

Выполните с консоли команду :

`mvn clean install`



```
Terminal: local +
[INFO] WEB-INF\web.xml already added, skipping
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ Servlet ---
[INFO] Installing C:\NATALIA\Персональные ИМ\Projects\Servlet_Step4_Maven_test\target\FinalWebApplication.war to C:\NATALIA\SOFT\maven-repo\by\patsei\Servlet\1.0-SNAPSHOT\Servlet-1.0-SNAPSHOT.war
[INFO] Installing C:\NATALIA\Персональные ИМ\Projects\Servlet_Step4_Maven_test\pom.xml to C:\NATALIA\SOFT\maven-repo\by\patsei\Servlet\1.0-SNAPSHOT\Servlet-1.0-SNAPSHOT.pom
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 19.115 s
[INFO] Finished at: 2019-04-15T14:58:23+03:00
[INFO]
```

Войдите в локальный репозиторий, найдите ваш war.

В случае ошибок добавьте плагины.

В помощь: НЕ забывайте выполнять очистку (у вас могут быть старые версии библиотек) и перезагрузку среды разработки.

ТЕОРИЯ

1) СОЗДАНИЕ НОВОГО ПРОЕКТА

После того, как вы установили arachemaven, вам необходимо создать новый проект. Для этого необходимо сделать следующее:

1. Запустите консоль (cmd в windows).
2. Запустите следующую команду:
`mvn archetype:generate`
3. Если это первый раз, когда вы запускаете эту команду, вы увидите статус скачивания в командной строке.
4. Затем вы увидите большой список архетипов, у каждого есть номер, имя, и короткое описание, описывающие, что каждый из них представляет. Выберем архетип по умолчанию. Мы предполагаем, что это архетип номер 513, с именем maven-archetype-quickstart. Список со временем может меняться.

```
[INFO] No archetype defined. Using maven-archetype-quickstart
```

```
(org.apache.maven.archetypes:maven-archetype-quickstart:1.0)
```

```
Choose archetype:
```

```
1: remote -> docbkx-quickstart-archetype (-)
```

```
...
```

```
100: remote -> maven-archetype-profiles (-)
```

```
101: remote -> maven-archetype-quickstart (An archetype which
contains a sample Maven project.)
```

```
...
```

```
1222: remote -> us.fatehi:schemacrawler-archetype-plugin-lint (-)
```

Choose a number: 513:

Вас попросят выбрать **версию архетипа**. Архетип по умолчанию (default) – это последняя стабильная версия архетипа.

Затем, у вас спросят ввести параметры проекта, такие как **groupid**, **artifactId**, **version** и **package**.

GroupId параметр, используемый для определения иерархического расположения проекта в Maven **репозитории**.

В этом случае, **репозиторий** – это локальный репозиторий Maven, расположенный на вашем компьютере. **GroupId** определяет типичный каталог верхнего уровня, и это может быть использовано в нескольких проектах в организации.

ArtifactId идентифицирует ваш проект и **version** указывает версию проекта. **ArtifactId** используется, когда артефакты распакованные в репозитории используются как зависимость в других проектах.

5. По завершению вы должны увидеть следующее:

```
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----
```

Мы используем **Maven**, в действительности **Mavenplugin**, для создания или генерации нового проекта.

Новый проект **ApacheMaven** может быть создан «вручную» с `pom.xml` и папками, созданными или сгенерированными в соответствии с архитектурой Maven проекта.

Maven создал **папку для проекта**, содержащую главный **`pom.xml`** и папку для исходных кодов **`src`**, который содержит подпапки для приложения и папки для тестовых исходных кодов.

Name	Size	Type
TestCreateApp	2 items	folder
src	2 items	folder
main	1 item	folder
java	1 item	folder
net	1 item	folder
srirangan	1 item	folder
packt	1 item	folder
maven	1 item	folder
App.java	188 bytes	Java source code
test	1 item	folder
java	1 item	folder
net	1 item	folder
srirangan	1 item	folder
packt	1 item	folder
maven	1 item	folder
AppTest.java	653 bytes	Java source code
pom.xml	773 bytes	XML document

Рисунок Структура файлов maven проекта

2) КОМПИЛИРОВАНИЕ И ТЕСТИРОВАНИЕ ПРОЕКТА

Для того, что бы откомпилировать и протестировать проект с помощью maven, необходимо сделать следующее:

1. Откройте консоль (*cmd*)
2. Перейдите в проектную папку, содержащую *pom.xml*
3. Запустите следующую команду:

```
mvn compile
```

Archetype Maven начинает скачивать зависимости (артефакты), если они не доступны в локальном репозитории, и затем компилирует проект.

```
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----
```

Если результат **Build Success**, это значит **maven завершил компиляцию** и сборку приложения.

Maven сделал тестирование частью стандартного жизненного цикла сборки.

Когда используется стандартное соглашение maven, то проект включает src/test папку, которая содержит все тесты для кода. Для запуска тестов, вам необходимо запустить следующую команду:

```
mvn test
```

Эта команда запускает на выполнение тесты maven, находящиеся в *../src/test* папки. Когда выполнение команды закончится, вы увидите отчет на консоли:

```
-----  
T E S T S  
-----  
  
Running net.srirangan.packt.maven.AppTest  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed:  
0.037  
sec  
Results :  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0  
  
Запускаем следующую команду:  
  
mvn compile
```

Команда запустит Java компилятор, ассоциированный с проектом. Откомпилированный код кладется в target директорию. Target директория обычно содержит откомпилированные артефакты (это может быть jar файл по умолчанию) вместе с папками для откомпилированных классов и тестов. Эта папка, также, содержит pom.properties файл, а также тестовые отчеты и временные файлы.

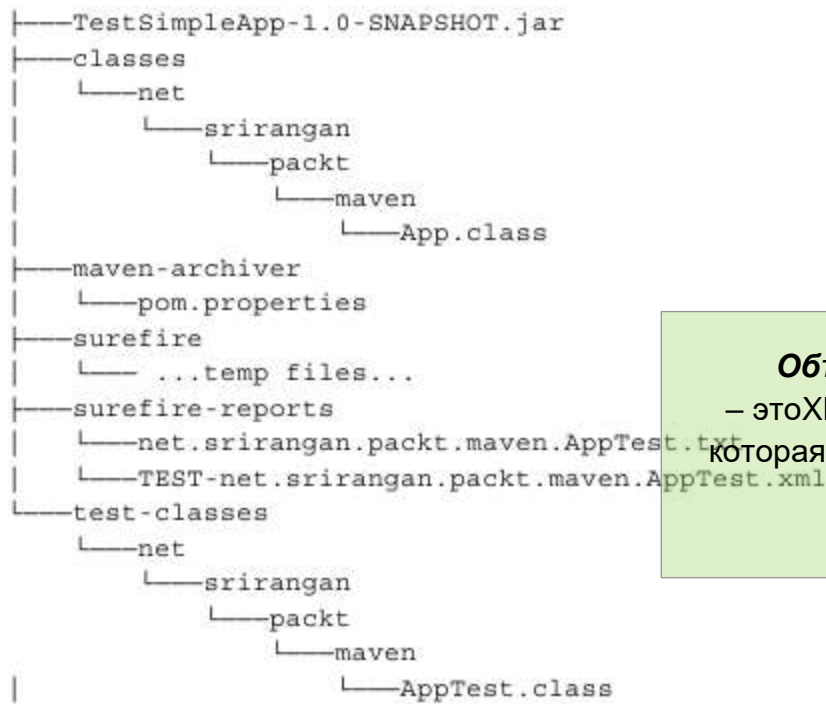


Рисунок Target директория

4) ОПИСАНИЕ ОБЪЕКТНОЙ МОДЕЛИ ПРОЕКТА (POM)

Каждый ApacheMaven проект содержит файл ***pom.xml***.

Объектная модель проекта содержит проектную конфигурацию, детали системы учета дефектов, проектную организацию и лицензии, проектные пути, зависимости и т.д. Структура типового стандартного файла ***pom.xml*** выглядит следующим образом:

```

<project ... >

<modelVersion>4.0.0</modelVersion>

<!--Базис -->

<groupId>...</groupId>

<artifactId>...</artifactId>

<version>...</version>

<packaging>...</packaging>

<dependencies>...</dependencies>

<parent>...</parent>

<dependencyManagement>...</dependencyManagement>

<modules>...</modules>

<properties>...</properties>

<!-- Настройки сборки -->

<build>...</build>

<reporting>...</reporting>

```

```

<!--Метаданные проекта -->

<name>...</name>

<description>...</description>

<url>...</url>

<inceptionYear>...</inceptionYear>

<licenses>...</licenses>

<organization>...</organization>

<developers>...</developers>

<contributors>...</contributors>

<!--Среда разработки -->

<issueManagement>...</issueManagement>

<ciManagement>...</ciManagement>

<mailingLists>...</mailingLists>

<scm>...</scm>

<prerequisites>...</prerequisites>

<repositories>...</repositories>

<pluginRepositories>...</pluginRepositories>

<distributionManagement>...</distributionManagement>

<profiles>...</profiles>

</project>

```

Минимальное число обязательных полей pom-файл включает в себя - groupId, artifactId и version. Эти три поля определяют расположение артефакта в репозитории.

Представленный выше пример **pom файла** включает 4 основные секции:

1. **Базис.** Эта секция содержит поля, определяющие уникальность артефакта (groupId, artifactId и version), управление зависимостями, а также детали наследования.
2. **Настройки сборки.** Детали сборки располагаются в этой секции.
3. **Метаданные проекта.** Эта секция включает специфичные для проекта параметры, такие как имя, организация, разработчики, адрес сайта организации, которая разрабатывает продукт, год поставки и т.д.
4. **Среда разработки.** Эта секция содержит всю информацию, касающуюся среды разработки, включая детали о системе контроля версий, системе управления дефектами, системы непрерывной сборки, списки рассылок, репозиториях и т.д.

5) ОПИСАНИЕ ЖИЗНЕННОГО ЦИКЛА СБОРКИ.

Жизненный цикл сборки точно определяет процесс сборки, тестирования, выпуска артефактов, и это является сердцем каждого maven проекта.

Существует три встроенных жизненных циклов сборки: *стандартный, очистить и сайт*.

Стандартный жизненный цикл обрабатывает компиляцию проекта, тестирование и установку. Он содержит более чем 20 фаз, наиболее важные из которых перечислены ниже:

1. **Проверка (validate)**: проверяет, что вся проектная информация доступна и корректна.
2. **Компилирование (compile)**: компилирует исходный код.
3. **Тестирование (test)**: запускает модульные тесты.
4. **Упаковка (package)**: упаковывает откомпилированный код в соответствующий выходной формат.
5. **Интеграционное тестирование (integration-test)**: запускает интеграционные тесты на тестовой среде.
6. **Проверка упаковки (verify)**: Запускает проверки на правильность упаковки.
7. **Установка (Install)**: устанавливает упакованный артефакт на локальный репозиторий.
8. **Установка на удаленный репозиторий (deploy)**: установка финального упакованного артефакта на удаленный репозиторий.

Как бы вы не запускали на выполнение фазу сборки, все предыдущие фазы выполняются последовательно. Например, выполняя команду

```
mvn integration-test,
```

вы запускаете проверку, компиляцию, тестирование и упаковку, до того момента пока выполнится фаза интеграционного тестирования.

Жизненный цикл очистки отвечает за процесс очистки проекта от предыдущих файлов сборки состоит из следующих фаз:

1. **Подготовка к очистке (pre-clean)**: выполняет процессы необходимые до очистки предыдущих файлов сборки проекта.
2. **Очистка (clean)**: удаляет все файлы полученные при предыдущей сборке.
3. **Фаза после очистки (post-clean)**: выполняет процессы необходимые для завершения очистки файлов предыдущей сборки проекта.

Жизненный цикл сайт отвечает за генерацию и установку документации проекта, состоит из следующих фаз:

1. Подготовка к генерации документации (pre-clean): выполняет процессы необходимые до генерации документации проекта.
2. Генерация документации (site): генерация проектной документации.

6) ОПИСАНИЕ ПРОФАЙЛОВ СБОРКИ.

Проекты в maven портируемы. Портируемость достигается посредством **конфигурации POM**, обходя все особенности файловой системы, и сильно зависима от локального репозитория, в котором сохранены необходимые метаданные.

Однако, не всегда возможно достичь портируемости, благодаря внутренним взаимозависимостям конфигурации и файловой системы. В таких случаях maven представляет концепцию **профайлов сборки**.

Профайл сборки – это настройки, описанные в pom.xml, которые могут быть запущены когда необходимо.

Существуют следующие варианты запуска профайлов:

1. Запуск профайла через командную строку.
2. Запуск профайла в зависимости от настроек maven.
3. Запуск профайла в зависимости от среды разработки.

Профайл может быть запущен через **командную строку** при помощи параметра `-P`. Список профайлов, разделенных запятыми, которые должны быть активированы, должны быть перечислены после флага `-P`.

```
mvn install -P profile-1,profile-2
```

В этом случае, только **профайлы**, указанные в команде, все же другие сконфигурированные **профайлы**, будут **неактивны**.

Обратное действие также **возможно**. Вы можете указать, какие профайлы не должны быть активированы.

```
mvn install -P !profile-1,!profile-2
```

Настройки maven могут активировать профайлы, если они указаны в секции `<activeProfiles>` файла settings.xml.

```
<settings>
...
<activeProfiles>
<activeProfile>profile-1</activeProfile>
<activeProfile>profile-2</activeProfile>
</activeProfiles>
...
</settings>
```

Профайлы, указанные в настройках maven активируются по умолчанию каждый раз и их не нужно явно указывать в командной строке.

Профайлы могут быть запущены в зависимости от среды разработки. Среда, в которой профайл должен быть активирован, определяется в секции `<profiles>` файла pom.xml.

```
<profiles>
<profile>
<activation>
<property>
```

```
<name>environment</name>  
<value>dev</value>  
</property>  
</activation>  
</profile>  
</profiles>
```

В коде, указанном выше, профайл может быть активирован в dev среде разработке. Пример типичной команды maven, в которой среда разработки указана явно, выглядит так:

```
mvn groupId:artifactId:goal -Denvironment=dev
```

1) АВТОМАТИЗАЦИЯ СБОРКИ

Автоматизация сборки – это реализованные на языке программирования задачи, которые разработчики выполняют ежедневно для запуска и тестирования написанного кода.

Эти задачи включают:

1. Компиляцию исходного кода в исполняемый код.
2. Упаковка исполняемого кода.
3. Запуск тестов.
4. Установка упакованного исполняемого кода на удаленные системы.
5. Создание документации к релизу и продукту.

Автоматизация сборки представляет ряд преимуществ, включая ускорение сборки, ограничение неудачных сборок, стандартизация и унификация в командах и организациях, увеличение эффективности и улучшение в продуктивном качестве. Сегодня, эта практика необходима для ежедневного использования.

Если вам необходим **готовый maven проект**. Если у вас нет готового, запустите следующую команду в командной строке, для создания простого Java проекта:

```
mvn archetype:generate -DgroupId=com.epam.maven  
-DartifactId=MySampleApp
```

Команда `archetype:generate` создал Apache Maven проект для нас. Если мы выберем `maven-archetype-quickstart` тип из списка, наша структура проекта будет выглядеть похожей на рисунок



Рисунок Структура проекта

В каждом ApacheMaven проект, включая тот, который мы сгенерировали, сборка запускает автоматически стандартный жизненный цикл сборки.

Тоже самое вы можете выполнить вручную:

```
mvn validate
...
mvn compile
...
mvn package
...
mvntest
...
```

Вы можете включить некоторые фазы жизненного цикла сборки индивидуальной командой. Maven позволяет вам **автоматизировать запуск всех фаз сборки в правильном порядке**.

Запустите команду `mvn install` и вы запустите часть стандартного жизненного цикла сборки, включая компилирование, тестирование, упаковку и установку артефакта в локальный репозиторий.

Автоматизация сборки стандартизирует структуру проекта, что может упростить разработку и тестирование продуктов.

2) Модульность ПРОЕКТА

Предположим, что вы собираете большое приложение, требующее взаимодействие со старой базой данных, работающее с существующими сервисами, предоставляющие современные графические интерфейсы как для web- и прикладных-составляющих, и предоставляющих API другим приложениям заказчиков. Такая сборка подразумевает разделение этого достаточно большого проекта на **подпроекты или модули**.

Apache maven предоставляет верную и точную поддержку, для такой проектной организации, через **Apache Maven мультимодульные проекты**. Мультимодульные проекты состоят из **«родительского проекта»**, который включает **«дочерние проекты»** или модули.

Родительский `pom` файл содержит ссылки на все дочерние модули. Каждый может быть разного типа, с разным упаковочным расширением.

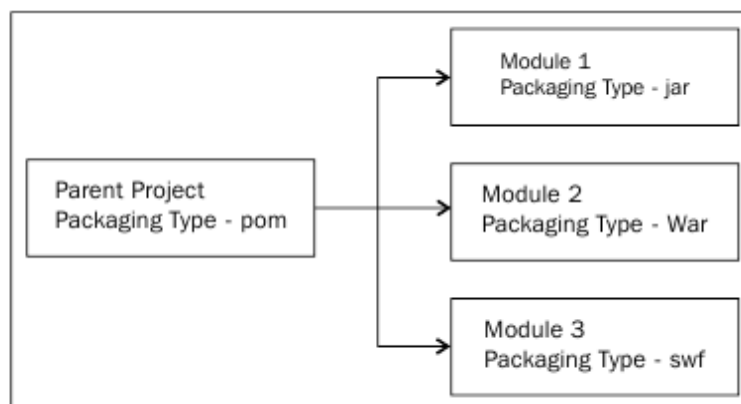


Рисунок Мультимодульный проект.

Мы начинаем с создания родительского модуля. Не забудьте поставить значение **packaging в pom**, как представлено в следующем **примере**:

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

```

<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">
<modelVersion>4.0.0</modelVersion>
<groupId>com.epam.maven</groupId>
<artifactId>TestModularApp</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>pom</packaging>
<name>ModularApp</name>
</project>

```

Это **базовый родительский pom-файл** для нашего проекта ModularApp. Он не содержит никаких дочерние модули на данный момент.

Для создания нашего **первого дочернего модуля**, запустите консоль командной строки (cmd), перейдите в директорию, где лежит родительский pom- файл и запустите следующую команду:

```
mvn archetype:generate
```

Данная команда покажет список всех архетипов на ваш выбор. Вы можете выбрать **архетип maven-archetype-quickstart**, который сгенерирует базовый Java модуль. Команда *archetype:generate* требует заполнения параметров maven проекта: groupId, artifactId, package, version и др.

После генерации модуля, просмотрите родительский pom файл, и вы увидите, что следующий блок был добавлен в секцию модулей:

```

<modules>
<module>moduleJar</module>
</modules>

```

Дочерний модуль, который мы создали, автоматически был добавлен в проект к родительскому модулю. Эта процедура работает достаточно просто – **никакого внешнего вмешательства не требуется**.

Сейчас мы создадим другой дочерний модуль. Пришло время создать web-модуль, путем вызова следующей команды:

```
mvn archetype:generate -DarchetypeArtifactId=maven-archetype-
webapp
```

Давайте, снова, посмотрим на родительский pom файл, и мы увидим, что оба дочерних модуля добавлены в секцию модулей:

```

<modules>
<module>moduleJar</module>
<module>moduleWar</module>
</modules>

```

Общая структура проекта (структура директорий) теперь имеет вид, как на рисунке.

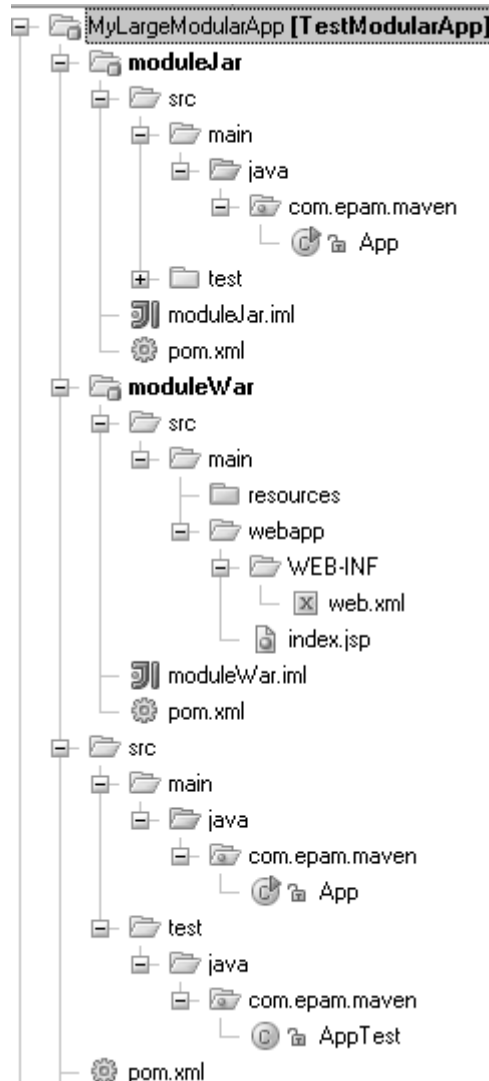


Рисунок Структура директорий нашего мульти модульного проекта.

Компиляция и установка (еще бывает и распаковка на сервере) **обоих дочерних модулей** (в правильном порядке, в случае, если дочерние модули взаимозависимы) – это **главное**. Данная операция может быть сделана, путем запуска командной строки, перехода в директорию, где располагается родительский pom файл, и запуска следующей команды:

```
mvn cleaninstall
```

Данная команда, запуская процесс сборки для родительского модуля, так же автоматически запускает сборку всех дочерних модулей в правильном порядке.

Вы должны увидеть результат в командной строке, подобный следующему:

```
-----  
[INFO] Reactor Summary:  
[INFO] MyLargeModularApp ..... SUCCESS [0.439s]  
[INFO] MyModuleJar .....SUCCESS [3.047s]  
[INFO] MyModuleWar Maven Webapp ..... SUCCESS [0.947s]  
-----  
[INFO] BUILD SUCCESS  
-----
```

3)УПРАВЛЕНИЕ ЗАВИСИМОСТЯМИ

Управление зависимостями заслуженно считается одной из лучших возможностей Apache Maven. В мультимодульных проектах, где количество зависимостей достигает десятков или даже сотен, Apache Maven отличается от других систем сборки, тем, что позволяет сохранять высокий уровень контроля и стабильности системы.

Управление зависимостями в Apache Maven— это прозрачный процесс. Это значит, что maven автоматически находит артефакты, от которых зависит ваш проект. Эта возможность стала доступной, начиная с версии Maven 2, и это стало особенно удобно для многих проектов с открытым исходным кодом (open source projects) и для их зависимостей, которые мы используем в сегодняшних проектах.

Зависимости в maven имеют 6 возможных областей действий:

1. Зависимость, необходимая для компилирования (**compile**): Эта область действия по умолчанию. Зависимость доступна в classpath.
2. Зависимость, представлена во времени выполнения (**provided**): Эта область действия предполагает, что JDK или среда разработки предоставляет зависимость во времени выполнения.
3. Зависимость, представлена во время выполнения (**runtime**): Зависимость необходима во время выполнения и определена в runtimeclasspath.
4. Зависимость, необходима для тестирования (**Test**): Зависимости необходимы для компиляции и выполнения тестов.
5. Системная зависимость (**System**): Зависимость всегда доступна, и в любом случае наличие JAR обеспечено.
6. Импортрируемая зависимость (**Import**):Импортрируемая зависимость представлена в POM, включая <dependencyManagement/> секцию.

Зависимости для Apache Maven проекта описаны в проектном pom-файле. Пока мы глубже начали рассматривать, как работать с зависимостями, мы рассмотрим Apache Maven плагин управления зависимостями (**ApacheMaven dependency plugin**).

В соответствии с <http://maven.apache.org/plugins/maven-dependency-plugin/>:

Это плагин, который предоставляет ряд удобных задач (goals), которые заключаются в следующем:

1. **mvn dependency:analyze**. Анализирует зависимости (используемые, неиспользуемые, декларируемые или не декларируемые зависимости).
2. **mvn dependency:analyze-duplicate**. Определяет дублирующие друг друга зависимости.
3. **mvn dependency:resolve**. Определить все зависимости.
4. **mvn dependency:resolve-plugin**. Определить все плагины.
5. **mvn dependency:tree**. Определить дерево зависимостей.

Большинство maven проектов зависимы от других артефактов (других проектов, библиотек и инструментов). Управление и интеграция зависимостями является одной из основных особенностей maven. Зависимости проекта указаны в pom-файле.

```
<dependencies>
<dependency>
<groupId>...</groupId>
<artifactId>...</artifactId>
```



```

<version>...</version>
<scope>...</scope>
</dependency>
</dependencies>

```

В мультимодульных проектах, зависимости могут быть определены в родительском pom-файле и могут впоследствии унаследованы в дочерних pom- файлах, когда это необходимо. Расположение всех зависимостей в одном источнике делает версиюность зависимостей более простой, что делает большие проектные зависимости более управляемыми и масштабируемыми во времени.

Следующий пример показывает мультимодульный проект, обладающий зависимостью от MySQL.

Родительский pom будет содержать полное описание зависимости:

```

<dependencyManagement>
<dependencies>
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>5.1.2</version>
</dependency>
</dependencies>
</dependencyManagement>

```

Все дочерние модули требуют MySQL будут включать опрощенное определение зависимости:

```

<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
</dependency>

```

При таком описании не будет конфликта версий между многочисленными дочерними модулями, имеющие те же зависимости.

Область действия и тип зависимости по умолчанию – это **compile** и **JAR**. Однако, они могут быть переопределены по требованию:

```

<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>4.8.2</version>
<scope>test</scope>
</dependency>
<dependency>

```

```
<groupId>...</groupId>
<artifactId>...</artifactId>
<version>...</version>
<type>war</type>
</dependency>
```

Системные зависимости не ищутся в репозитории. Мы должны указывать путь к таким зависимостям сами:

```
<dependencies>
<dependency>
<groupId>sun.jdk</groupId>
<artifactId>tools</artifactId>
<version>1.5.0</version>
<scope>system</scope>
<systemPath>${java.home}/../lib/tools.jar</systemPath>
</dependency>
</dependencies>
```

Однако очень рекомендовано избегать системных зависимостей, потому что это противоречит общей концепции управления зависимостями ApacheMaven.

В идеале, разработчик должен иметь возможность взять код из системы контроля версий и запустить ApacheMaven командой. После этого, ApacheMaven должен отвечать за управление зависимостями.

4)АВТОМАТИЗАЦИЯ УСТАНОВКИ ПРОЕКТА.

Maven Deploy плагин используется для добавления артефактов на удаленный репозиторий, в процессе фазы установки в жизненном цикле сборки.

Deploy плагин представляет две задачи:

1. **deploy:deploy**: Установка проекта и всех его артефактов.
2. **deploy:deploy-file**: Установка одного простого артефакта.

Установка в репозиторий означает не только копирование артефакта в директорию, но для обновления метаданных, так же как и артефактов. Это требует:

1. **Targetrepository**: Целевой репозиторий – это где артефакты будут установлены. Это необходимое расположение, протокол доступа (FTP, SCP, SFTP) и специфическая пользовательская информация о профиле.
2. **Targetartifacts**: Целевые артефакты – это артефакты, которые будут установлены. GroupId, artifactId, version, packaging, и classifier информация артефакта необходимы.

3. *Deployermethod*: Методустановки, с помощью которого происходит установка. Метод может быть реализован либо в кросс- платформенном, либо в платформенном виде.

Проектный pom файл должен иметь <distributionManagement/> секцию с <repository/> элементами внутри, в которых описаны расположение и параметры удаленного репозитория.

```
<distributionManagement>
<repository>
<id>by.park</id>
<name>MyPrivateRepository</name>
<url>...</url>
</repository>
</distributionManagement>
```

Для этого, вам необходимо описать информацию о сервере в вашем settings.xml (<USER_HOME>/.m2/settings.xml или <M2_HOME>/conf/settings.xml) для предоставления идентификационной информации для репозитория.

```
<server>
<id>by.park</id>
<username>ivan</username>
<password>myTopSecretPassword</password>
</server>
```

Установка может быть выполнена из командной строки, перейдя в директорию проекта и вызвав следующую команду:

```
mvn deploy
```

ApacheMavenDeploy плагин вызывается в процессе выполнения deploy фазы жизненного цикла сборки. Как мы видели, <distributionManagement/> секция с <repository/> элементом, необходима для запуска работы плагина.

Snapshots и releases могут быть разделены определением <snapshotRepository/> элемента.

```
<distributionManagement>
<repository>
<uniqueVersion>false</uniqueVersion>
<id>corp1</id>
<name>Corporate Repository</name>
<url>scp://repo/maven2</url>
<layout>default</layout>
</repository>
<snapshotRepository>
<uniqueVersion>true</uniqueVersion>
```

```

<id>propSnap</id>
<name>Propellors Snapshots</name>
<url>sftp://propellers.net/maven</url>
<layout>legacy</layout>
</snapshotRepository>
...
</distributionManagement>

```

Секция `<distributionManagement/>` наследуется всеми дочерними модулями. Действительная установка происходит на основе протоколов, определенных в репозитории. Чаще всего используют FTP и SSH протоколы. Wagon-ftp и wagon-ssh-external – основные провайдеры для этих протоколов.

Если удаленный репозиторий доступен через FTP протокол, то `<build/>` секция должна включать определение wagon-ftp.

```

<distributionManagement>
  <repository>
    <id>sri-ftp-repository</id>
    <url>ftp://...</url>
  </repository>
</distributionManagement>

<build>
  <extensions>
    <extension>
      <groupId>org.apache.maven.wagon</groupId>
      <artifactId>wagon-ftp</artifactId>
      <version>1.0-beta-6</version>
    </extension>
  </extensions>
</build>

```

Если удаленный репозиторий доступен через SSH, определяемое по artifactId и URL, тогда `<build/>` секция включает определение wagon-ssh-external.

```

<distributionManagement>
  <repository>
    <id>sri-ssh-repository</id>
    <url>scpexe://....</url>
  </repository>
</distributionManagement>

<build>
  <extensions>

```

```

<extension>
<groupId>org.apache.maven.wagon</groupId>
<artifactId>wagon-ssh-external</artifactId>
<version>1.0-beta-6</version>
</extension>
</extensions>
</build>

```

Идентификационная информация будет храниться в settings.xml.

5)СБОРКА WEB-ПРИЛОЖЕНИЯ.

Мы начнем с концепции POWA (простое web приложение) – это web проект, состоящий из сервлетов и нескольких JSP. Шаблон, который может использоваться для генерации POWA называется org.sonatype.mavenbook.simpleweb.

Откройте командную строку и запустите следующую команду:

```

mvn archetype:generate -DarchetypeArtifactId=maven-archetype-webapp
-DartifactId=testWebApp -DgroupId=com.epam.maven -Dversion=1.0-SNAPSHOT
-Dpackage=com.epam.maven

```

Вы должны увидеть следующее в консоли:

```

[INFO] -----
--

[INFO] Using following parameters for creating project from Old (1.x)
Archetype: maven-archetype-webapp:1.0

[INFO] -----
--

[INFO] Parameter: groupId, Value: com.epam.maven
[INFO] Parameter: packageName, Value: com.epam.maven
[INFO] Parameter: package, Value: com.epam.maven
[INFO] Parameter: artifactId, Value: testWebApp
[INFO] Parameter: basedir, Value: G:\Work\maven
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir:
G:\Work\maven\testWebApp

[INFO] -----
--

[INFO] BUILD SUCCESS

[INFO] -----
--

```

Команда archetype:generate создала новую проектную директорию testWebApp, которая содержит следующий pom-файл:

```

<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/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.epam.maven</groupId>
    <artifactId>testWebApp</artifactId>
    <packaging>war</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>testWebApp Maven Webapp</name>
    <url>http://maven.apache.org</url>
    <dependencies>
    <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
    </dependency>
    </dependencies>
    <build>
    <finalName>testWebApp</finalName>
    </build>
    </project>

```

Сейчас, мы добавим build plugin, для компилирования проекта с помощью JDK 1.6. <build/> секция pom файла должна иметь похожий вид:

```

..
<build>
<finalName>testWebApp</finalName>
<plugins>
<plugin>
<artifactId>maven-compiler-plugin</artifactId>
<configuration>
<source>1.6</source>
<target>1.6</target>
</configuration>
</plugin>
</plugins>
</build>
..

```

ApacheMavenweb проект будет работать, как многие другие maven проекты. Структура проекта изображена на рисунке 2.5.1.

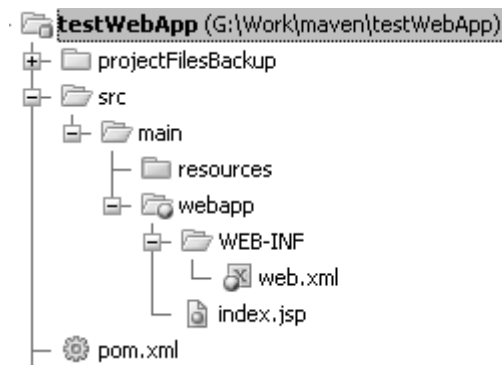


Рисунок 2.5.1 Структура maven web проекта.

Вы можете выполнить следующие команды для компилирования и тестирования проекта:

1. mvn compile
2. mvn test
3. mvn install

По окончании установки, выходной артефакт будет упакован и установлен в локальный репозиторий, как другие maven проекты. Он также может быть установлен на удаленный репозиторий.

Если вы посмотрите, вы можете найти проектную директорию `src/main/webapp`, в которой находится файл `index.jsp`. Вы можете изменить этот файл, как вам необходимо, например вот так:

```

<html>

  <body>

    <h2>Maven is very useful tool</h2>

  </body>

</html>

```

Итак, давайте запустим сборку проекта - `mvninstall`.

В консоли мы увидим следующий результат:

```

[INFO] --- maven-install-plugin:2.3.1:install (default-install) @
testWebApp

[INFO] Installing G:\Work\maven\testWebApp\target\testWebApp.war to
f:\opt\globoforce\repository\com\epam\maven\testWebApp\1.0-
SNAPSHOT\testWebApp-1.0-SNAPSHOT.war

[INFO] Installing G:\Work\maven\testWebApp\pom.xml to
f:\opt\globoforce\repository\com\epam\maven\testWebApp\1.0-
SNAPSHOT\testWebApp-1.0-SNAPSHOT.pom

[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----

```


Как мы видим, после успешной сборки, target директория была создана. В target директории вы можете найти WAR файл с именем testWebApp.war. Давайте посмотрим на pom файл, упакованный в данном архиве.

```
...  
<artifactId>testWebApp</artifactId>  
<packaging>war</packaging>  
<version>1.0-SNAPSHOT</version>  
<name>testWebApp Maven Webapp</name>  
...
```

Как мы можем видеть, что packaging тип установлен в WAR и выходной артефакт отражает это свойство.