

Методические указания
по выполнению практической работы № 3
«Системы сборки»

по дисциплине «Технологии разработки программных приложений»

Жматов Д.В., Петрова А.А.

Цель работы

Основная цель работы с Gradle — автоматизация процесса сборки, управления зависимостями и развертывания программных проектов. Это позволяет разработчикам сократить время на рутинные задачи, повысить надежность и гибкость сборочного процесса.

2.1 Теоретическая часть

Gradle — система сборки проектов с открытым исходным кодом, в которой основной упор идет на гибкость и производительность. В настоящее время поддерживаются 2 языка программирования для написания скриптов сборки: groovy и kotlin.

Для работы с gradle нужна только установленная jdk, для всех проектов хорошим тоном считается иметь gradle-wrapper. Это обертка из небольшого jar-файла (порядка 58 Кб), 1 файла конфигурации и 2 скриптов для запуска — windows и *nix. При первом запуске-wrapperа загружается gradle необходимой версии и после чего начинает свою штатную работу. Данная схема необходима, так как в зависимости от множества причин версия системы сборки у членов команды и билд-сервера может отличаться.

Gradle далеко не первая система сборки для java и Android-приложений, поэтому данная система может использовать хранилища артефактов (библиотек), которые изначально предназначались для maven.

Вызов-wrapperа в корне проекта

- в Windows: gradlew.bat (в PowerShell ./gradlew.bat)
- в Unix-like: ./gradlew

Если все сделано правильно, то gradle вам сообщит свою версию и выведет небольшой help по своим командам:

```
Welcome to Gradle 6.8!

Here are the highlights of this release:
- Faster Kotlin DSL script compilation
- Vendor selection for Java toolchains
- Convenient execution of tasks in composite builds
- Consistent dependency resolution

For more details see https://docs.gradle.org/6.8/release-notes.html

> Task :help

Welcome to Gradle 6.8.

To run a build, run gradlew <task> ...

To see a list of available tasks, run gradlew tasks

To see a list of command-line options, run gradlew --help

To see more detail about a task, run gradlew help --task <task>

For troubleshooting, visit https://help.gradle.org

BUILD SUCCESSFUL in 1s
1 actionable task: 1 executed
```

Выполнение задач в gradle

Задачи в gradle выполняются следующим образом: вызов wrappers gradle задача1 задача2 задачаN. Выполнение задач будет именно в той последовательности, в которой вы их вызовете. Также некоторые задачи по умолчанию зависят друг от друга. Например, задачу compileJava вызывать нет никакой нужды, если вы собираете проект целиком командой build.

Самые популярные задачи:

- clean — очистка всех сборочных директорий
- build — собрать приложение целиком
- javadoc — сгенерировать JavaDoc документацию
- shadowJar — собрать UberJar (архив со всеми зависимостями, для использования необходимо импортировать плагин shadow)
- run — запуск приложения
- test — прогон тестов

Рассмотрим базовый скрипт для сборки при создании проекта в IntelliJ IDEA (для данного скрипта используется язык Groovy):

```
plugins {  
    id 'java'  
}  
  
group 'org.text'  
version '1.0'  
  
repositories { ma-  
venCentral()  
}  
  
dependencies {  
    testImplementation 'org.junit.jupiter:junit-jupiter-  
api:5.6.0' testRuntimeOnly 'org.junit.jupiter:junit-jupiter-en-  
gine'  
}
```

Рассмотрим основные разделы:

- 1) `plugins` — здесь перечисляются плагины, которые будут использоваться при сборке проекта. В сам `gradle` включены только “Core Plugins”. Если для нужд проекта чего-либо не хватает, то можно воспользоваться поиском на <https://plugins.gradle.org/>, либо реализовать свой плагин, который будет решать поставленную задачу.
- 2) `Пакет и версия` — нужны для идентификации вашего проекта. Если кто-то публикует приложение или библиотеку в `maven`-репозиториях, то по пакету и названию проекта можно его найти (указывается в рядом лежащем `settings.gradle`, например: `rootProject.name = 'sample'`).
- 3) `repositories` — здесь перечисляются репозитории, которые будут использоваться для поиска и загрузки артефактов. Самым популярным является `mavenCentral()`. Если у вас есть зависимости от других ваших проектов, которые вы пока нигде не публикуете, то не забудьте

указать еще `mavenLocal()` - в таком случае поиск артефактов будет производиться и по вашим локально собранным пакетам. Если вам необходимо указывать зависимости, которые есть, например, только в вашем рабочем хранилище артефактов, то такие зависимости указываются следующим образом внутри блока `repositories`:

```
maven {  
    url "https://plugins.gradle.org/m2/"  
}
```

4) `dependencies` — зависимости проекта. Скорее всего, когда приложение становится чуть больше `HelloWorld`, возникает необходимость в импортировании кода от третьих лиц (возможно, это будет библиотека от ваших же коллег, которые вынесли все, что вы используете вместе с ними, в отдельный `common`-модуль, которые все подключают как зависимость). Наиболее популярные типы:

- `annotationProcessor` — обработка аннотаций в процессе компиляции (чаще всего используется для кодогенерации, см. `lombok` в разделе 2.4)
- `implementation` — зависимости, необходимые для компиляции проекта, которые не являются частью API, предоставляемого проектом. Например, проект использует `Hibernate` для работы с БД, но API для работы с ним нет.
- `api` — зависимости, необходимые для компиляции проекта, которые являются частью API, предоставляемого проектом. Например, проект использует `Guava` и предоставляет публичные интерфейсы с классами `Guava` в их сигнатурах методов.

Также чаще всего в проектах указывают следующие разделы

1) Для того, чтобы явно указать версию `java` для работы

```
java {  
    sourceCompatibility = JavaVersion.toVersion("15")
```

```
        targetCompatibility = JavaVersion.toVersion("15")
    }
}
```

- 2) Указание кодировки (особенно актуально, у кого комментарии, javadoc или строки в хранилище текста могут быть на русском языке

```
compileJava {
    options.encoding = 'UTF-8'
}

compileTestJava { options.encoding = 'UTF-8'
}

javadoc {
    options.encoding = 'UTF-8'
}
```

2.2 Практическая часть

Для выполнения необходимо клонировать (или форкнуть) git-репозиторий согласно варианту.

Перед сборкой проекта нужно проверить настройки: установленный Java Development Kit (jdk) и версию Gradle 6.8. Рекомендовано использование jdk версии 11, поскольку она совместима с версией Gradle 6.8. Стоит учитывать, что при повышении версии jdk, также стоит повысить версию Gradle. Ниже представлена таблица совместимости версий.

Gradle Version	Supported Java Versions
Gradle 6.x	8, 9, 10, 11
Gradle 7.0	8, 11, 15
Gradle 7.1–7.3	8, 11, 15
Gradle 7.4–7.6	8, 11, 15, 16
Gradle 7.6.1	8, 11, 15, 16, 17
Gradle 8.0	17–20
Gradle 8.1–8.3	17–20

В файле для автоматической загрузки Gradle **gradle-wrapper.properties** в `distributionUrl` указана версия 6.8. Gradle Wrapper автоматически скачивает Gradle, если он не установлен локально.

```

1 distributionBase=GRADLE_USER_HOME
2 distributionPath=wrapper/dists
3 distributionUrl=https\://services.gradle.org/distributions/gradle-6.8-bin.zip
4 zipStoreBase=GRADLE_USER_HOME
5 zipStorePath=wrapper/dists

```

В случае, если в системе несколько версий Java; а также если версия Java не определяется автоматически, рекомендуется настроить системную переменную `JAVA_HOME` со значением корректного пути к нужной папке JDK. Например, `C:\Program Files\Java\jdk-11.0.17`

Задание на практическую работу

1. Первая попытка сборки проекта через задачу **build** приведет к набору ошибок. Исправив все ошибки, необходимо продемонстрировать успешную сборку. Также в отчете необходимо показать исправление всех ошибок.

Некоторые из ошибок при сборке:

1. Найти отсутствующую зависимость и указать ее в

соответствующем блоке в build.gradle.

2. В некоторых классах поправить имя пакета.

3. Добавить импорт отсутствующего класса (в зависимости от варианта задания).

Примечание: в некоторых вариантах ошибки могут отличаться. Необходимо продемонстрировать решение всех из них.

2. Собрать документацию проекта, найти в ней запросы состояния и сущности по идентификатору.

3. Собрать jar со всеми зависимостями (так называемый UberJar), после чего запустить приложение. По умолчанию, сервер стартует на порту 8080.

4. Запросить состояние запущенного сервера (GET запрос по адресу <http://localhost:8080>)

5. Запросить сущность по идентификатору (GET запрос по адресу: <http://localhost:8080/сущность/идентификатор>). Идентификатором будут 3 последние цифры в серийном номере Вашего студенческого билета.

Примечание: для варианта №9 запрос происходит по сущности employee (ошибка в проекте).

6. В задаче shadowJar добавить к jar-файлу Вашу фамилию. Также продемонстрировать на рисунке созданный jar-файл с фамилией.

7. Выполнить задачу checkstyleMain. Посмотреть сгенерированный отчет. Устранить ошибки оформления кода. Отобразить сформированную таблицу с количеством ошибок и таблицу с отсутствием ошибок.

Примечание: некоторые из ошибок могут остаться даже при успешной сборке проекта на 1 этапе. Сгенерированный отчет должен показывать, что все ошибки исправлены.

2.3 Варианты для выполнения работ

- 1) репозиторий: <https://github.com/rtu-mirea/trpp-second-1>, сущность ru.mirea.entity.Employee

- 2) репозиторий: <https://github.com/rtu-mirea/trpp-second-2>, сущность ru.mirea.entity.Student
- 3) репозиторий: <https://github.com/rtu-mirea/trpp-second-3>, сущность ru.mirea.entity.Client
- 4) репозиторий: <https://github.com/rtu-mirea/trpp-second-4>, сущность ru.mirea.entity.Organization
- 5) репозиторий: <https://github.com/rtu-mirea/trpp-second-5>, сущность ru.mirea.entity.Medicine
- 6) репозиторий: <https://github.com/rtu-mirea/trpp-second-6>, сущность ru.mirea.entity.Message
- 7) репозиторий: <https://github.com/rtu-mirea/trpp-second-7>, сущность ru.mirea.entity.Movie
- 8) репозиторий: <https://github.com/rtu-mirea/trpp-second-8>, сущность ru.mirea.entity.Book
- 9) репозиторий: <https://github.com/rtu-mirea/trpp-second-9>, сущность ru.mirea.entity.Nomenclature
- 10) репозиторий: <https://github.com/rtu-mirea/trpp-second-10>, сущность ru.mirea.entity.Game
- 11) репозиторий: <https://github.com/rtu-mirea/trpp-second-11>, сущность ru.mirea.entity.Devices
- 12) репозиторий: <https://github.com/rtu-mirea/trpp-second-12>, сущность ru.mirea.entity.Animal
- 13) репозиторий: <https://github.com/rtu-mirea/trpp-second-13>, сущность ru.mirea.entity.Car
- 14) репозиторий: <https://github.com/rtu-mirea/trpp-second-14>, сущность ru.mirea.entity.History
- 15) репозиторий: <https://github.com/rtu-mirea/trpp-second-15>, сущность ru.mirea.entity.Receipt

2.4. Полезные ссылки

<https://docs.gradle.org/current/userguide/userguide.html> — документация gradle

https://docs.gradle.org/current/userguide/declaring_dependencies.html — статья по управлению зависимостями проекта

<https://mvnrepository.com/> — инструмент для поиска артефактов (если ссылка не сработает по прямому переходу, то можно сделать вставку в адресную строку)

https://gradleup.com/shadow/getting-started/#__tabbed_1_1 — документация по gradle-плагину shadow

<https://micronaut.io/documentation.html> — документация по фреймворку, который используется в данной практической работе

<https://www.postman.com/> — удобный REST-клиент для тестирования сервера

<https://projectlombok.org/> — библиотека, которая позволяет избавиться от написания рутинных аксессоров к полям классов, конструкторов, сборщиков.

Вывод

Работа с Gradle направлена на оптимизацию и ускорение процесса разработки программного обеспечения. Благодаря автоматизации рутинных задач, удобному управлению зависимостями и высокой производительности, Gradle является мощным инструментом для разработчиков.

Контрольные вопросы:

1. Чем компиляция отличается от сборки?
2. Что такое система сборки?
3. Что такое репозиторий?
4. Как указать зависимости проекта?
5. Что такое gradle?
6. Что такое maven?
7. Что такое mavencentral?
8. Что делает задача clean?
9. Что делает задача build?
10. Что делает задача compileJava?
11. Что делает задача run?
12. Что делает задача test?
13. Что такое javadoc?
14. Что такое checkstyle?
15. Что такое UberJar? При помощи какой задачи его собрать?
16. Что такое micronaut?
17. Что такое lombok?
18. Что такое postman?
19. Что такое аннотация в Java?

Требования к отчету:

По итогу выполнения практической работы необходимо оформить отчёт, включающий:

1. Титульный лист;
2. Оглавление;
3. Выполнение всех пунктов задания;
4. Письменные развернутые ответы на все контрольные вопросы;
5. Вывод.

Отчет практической работы необходимо загрузить в СДО в установленные сроки