Методические указания

по выполнению практической работы № 2

«Системы сборки»

по дисциплине «Технологии разработки программных приложений»

Цель работы

Знакомство с системой сборки Gradle. Возможности gradle. Управление зависимостями.

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 враппера 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 враппера 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:

Выполнение задач в gradle

Задачи в gradle выполняются следующим образом: вызов враппера gradle задача1 задача2 … задачаN. Выполнение задач будет именно в той последовательности, в которой вы из 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 {
    mavenCentral()
}

dependencies {
    testImplementation 'org.junit.jupiter:junit-jupiter-api:5.6.0'
    testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine'
}
```

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 Задание для выполнения

Для выполнения необходимо клонировать (или форкнуть) git-репозиторий согласно варианту, и выполнить следующие задания:

1. Найти отсутствующую зависимость и указать ее в соответствующем блоке в build.gradle, чтобы проект снова начал собираться
2. В некоторых классах поправить имя пакета
3. Собрать документацию проекта, найти в ней запросы состояния и сущности по идентификатору
4. Собрать jar со всеми зависимостями (так называемый UberJar), после чего запустить приложение. По умолчанию, сервер стартует на порту 8080.
5. Запросить состояние запущенного сервера (GET запрос по адресу http://localhost:8080)
6. Запросить сущность по идентификатору (GET запрос по адресу: http://localhost:8080/сущность/идентификатор)
   *Идентификатором будут 3 последних цифры в серийном номере вашего студенческого билета***.**
7. В задаче shadowJar добавить к jar-файлу вашу фамилию
8. Выполнить задачу checkstyleMain. Посмотреть сгенерированный отчет. Устранить ошибки оформления кода.

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/dependency_management.html — статья по управлению зависимостями проекта

https://mvnrepository.com/ — инструмент для поиска артефактов

https://imperceptiblethoughts.com/shadow/introduction/ — 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, билдеров и т.д.

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. Изменения которые были проведены в исходном коде проекте для всех пунктов задания;
4. Скриншоты результатов, полученного в п.5 и п.6 задания;
5. Ответы на контрольные вопросы (6 любых);

Отчет о практической работы необходимо загрузить в СДО (в случае каких-либо технических проблем отчет необходимо выслать на почту преподавателя в домене @mirea.ru)