MTC True Tech

Лекция №2

CI with Maven & Gradle

Бобряков Дмитрий

Senior разработчик

Меня хорошо видно и слышно?

Проверить, идет ли запись

Ставим «+», если все хорошо
«-», если есть проблемы

Дмитрий Бобряков

Senior разработчик

- 7 лет в IT
- Senior-разработчик в BigData Streaming
- Experience: NetCracker, AlphaBank, MWS
- Java-ментор в MWS
- Преподаватель в МФТИ

Содержание лекции

- «Уровни» языков программирования
- Начинаем работать с языком Java
- Пакетные менеджеры
- Пример CI на Java & Maven

Что такое язык программирования?

ЯП – формальный язык, предназначенный для записи компьютерных программ. Язык программирования определяет набор лексических, синтаксических и семантических правил, определяющих внешний вид программы и действия, которые выполнит исполнитель (обычно — ЭВМ) под её управлением.

Пример 1- Assembler

Assembler — транслятор в машинный код.

```
        global  _main
        extern  _printf

        section .text
_main:
        push    message
        call    _printf
        add     esp, 4
        ret
message:
        db      'Hello, World', 10, 0
```

Пример 1- Assembler

Проблемы:

- Диалект Assembler отличается на разных операционных системах. Это значит, что программу, которую вы написали на Windows, не получится запустить на Linux.
- Уровень взаимодействия с компьютером слишком низкоуровневый. Даже для такой простой задачи как вывод в консоль надписи Hello, World потребовалось работать с регистрами и стеком процессора напрямую.
- Поддержка таких программ обходится дорого.

Пример 2- C

C — более высокоуровневый ЯП

+ компилятор, который преобразует понятный человеку код в формат для выполнения компьютером
- довольно близок к «железу», как и Assembler
- выделение памяти и ее последующее очищение возлагается на плечи программиста
- перекомпиляция под разные платформы

```c
#include <stdio.h>

int main() {
    printf("Hello, World");
    return 0;
}
```

Языки высокого уровня

Требования:

- Безопасность. Программа не должна «крашиться», например, из-за неверного обращения к адресу памяти.
- Скорость разработки. Цикл создания продукта должен быть достаточно быстрым, чтобы подстраиваться под меняющиеся требования рынка.
- Поддержка. Бизнес-приложения создаются с расчетом на долгосрочную эксплуатацию. Например, если вы создали сервис по заказу такси через смартфон, со временем его характеристики могут меняться: тарифы оплаты, система отзывов, алгоритмы подбора водителей. Чтобы обеспечить такие требования, во-первых, код должен быть расширяем. Во-вторых, программный код должен быть достаточно понятен, чтобы в нем мог разобраться новый программист.

Языки высокого уровня

Язык программирования должен предоставлять:

1. Сборщик мусора (Garbage Collector) – это часть среды выполнения языка программирования, которая автоматически освобождает память для неиспользующихся объектов.

2. Отсутствие указателей на конкретные участки памяти. Операции с памятью напрямую, как мы уже выяснили, могут быть очень опасны. Представьте себе, что из-за ошибки передачи указателя страховая компания согласовала человеку сумму в 10 раз больше одобренной изначально. Поэтому высокоуровневый язык программирования должен быть освобожден от подобных вещей.

3. Кроссплатформенность. Это значит, что результирующий «билд» должен запускаться не только на той системе, на которой он был собран, а на любой (или почти любой). Предположим, что вы поставляете клиентам некое бизнес-приложения для запуска на системе Windows, в то время как ваши программисты работают на Linux. Если «билд» не является переносимым, то вам придется дать исходный код продукта вашим клиентам, чтобы они могли скомпилировать его самостоятельно. Во-первых, клиенты могут не располагать нужными компетенциями. Во-вторых, код может быть защищен авторским правом.

4. Более низкий порог входа. Программисты приходят и уходят. А код, который они написали, остается. Это значит, что язык не должен быть чрезмерно усложнен, чтобы новые разработчики, уровень которых также может быть разным, могли быстро «влиться» в процесс разработки.

Типизация языков программирования

1. Со статической/динамической типизацией
2. С сильной/слабой типизацией
3. С явной/неявной типизацией

Статическая/динамическая типизация

В статически типизированных языках корректность типов проверяется на этапе компиляции. То есть на момент выполнения программы среда исполнения уже точно знает, какой тип ожидается в той или иной строчке кода. В динамических же языках типы становятся известны лишь на моменте выполнения конкретной инструкции.

К статическим языкам относятся: Java, C++, C#, Scala. К динамических языкам относятся: Python, Javascript, PHP.

Статическая/динамическая типизация

```
class Main {
    public static void main(String[] args) {
        int a = 45;
        int b = 55;
        System.out.println(sum(a, b));
    }

    public static int sum(int a, int b) {
        return a + b;
    }
}
```

Статическая/динамическая типизация

```
def sum(a, b):
    return a + b


if __name__ == '__main__':
    a = 45
    b = 55
    print(sum(a, b))
```

Плюсы статической типизации

- Типы гарантированно проверяются на этапе компиляции: мы уверены в значениях, которые получаем
- Меньше вероятности допустить ошибку из-за жесткой системы типов
- Статически типизированные языки практически всегда быстрее динамических, потому что нет накладных расходов на дополнительное определение типа значения в runtime.
- Ускорение разработки при помощи IDE: подсказки кода

Плюсы динамической типизации

- Простота создания универсальных коллекций типов
- Легкость в освоении

Почему выбирают статическую типизацию?

- Несоответствие типов гарантирует, что программа не запустится. В динамических языках выполнение дойдет до точки, где есть ошибка.
- Безопасность- не придёт число/строка/объект в обработчик
- Сложность предметной области.
- Большое количество кода: autocomplete помогает.

Сильная/слабая типизация

Сильная/слабая типизация означает отсутствие или наличие автоматического преобразование типов.

- Слабая- вы можете в одном выражении применять совершенно не связанные между собой объекты, и компилятор (или интерпретатор) приведет их к одному типу.
- Сильная, то подобные смешивания недопустимы. Программист должен вручную привести все типы к единому значению.

Что лучше?

Может сложиться впечатление, что мы пытаемся представить ситуацию так, как будто один язык программирования лучше другого. Это не так. Язык – это инструмент для решения задачи.

Разные языки лучше себя показывают в различных сценариях. Нет смысла пытаться забить гвоздь пассатижами, если есть молоток, которым сделать это гораздо удобнее. Здесь также можно вспомнить шутку Бьерна Страуструпа, создателя языка C++.

Есть два вида языков программирования. На одни все жалуются, другими никто не пользуется.

Создание и запуск проекта в Idea

<DEMONSTRATION>

Как Java внедряет сторонние библиотеки

```
package com.example;

class Main {

  public static void main(String[] args) {
    String inputStr = args[0];
    String hash = calculateMD5Hash(inputStr);
    System.out.println(hash);
  }

  private static String calculateMD5Hash(String input) {
    // implementation...
  }
}
```

Как Java внедряет сторонние библиотеки

```
package com.example;

import com.apache.HashUtil;

class Main {

  public static void main(String[] args) {
    String inputStr = args[0];
    String hash = HashUtil.md5(inputStr);
    System.out.println(hash);
  }
}
```

Часть 2. Пакетные менеджеры

Если проводить аналогию, представьте, что вы собираете сложный конструктор (ваш проект).

• Maven/Gradle — это и инструкция (которая описывает, что собирать и в каком порядке), и робот-помощник (который сам находит нужные детали на складе и собирает их).

• Зависимости (dependencies) — это те самые недостающие детали конструктора (например, библиотеки для работы с базой данных или веб-фреймворк Spring), которые ваш робот-помощник автоматически скачивает из интернета.

Часть 2. Пакетные менеджеры

Суть пакетного менеджера проста:

1.Каждая зависимость характеризуется тремя параметрами: groupId, artifactId и version.

2.Все модули публикуются в едином реестре – репозитории. По умолчанию Maven скачивает зависимости из Maven Central. Но в компаниях часто используются приватные репозитории, которые недоступны извне.

3.Зависимости описываются декларативно в файле pom.xml.

Часть 2. Пакетные менеджеры

```
<dependencies>
  <dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
    <version>3.12.0</version>
  </dependency>
</dependencies>
```

Часть 2. Пакетные менеджеры

Создать новый Java проект на Maven можно двумя способами:

1. С помощью командной строки. Например, приведенная ниже команда создаст Maven проект для Java со стандартными директориями.

2. С помощью IDEA. Выберите пункт: File -> Project... -> Maven

```
mvn archetype:generate \
    -DgroupId=com.mts.meta \
    -DartifactId=student-project \
    -DarchetypeArtifactId=maven-archetype-quickstart \
    -DinteractiveMode=false
```

Часть 2. Пакетные менеджеры

```
<build>

    <plugins>

      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-shade-plugin</artifactId>
        <version>3.5.0</version>
        <executions>
          <execution>
            <goals>
              <goal>shade</goal>
            </goals>
            <configuration>
              <shadedArtifactAttached>true</shadedArtifactAttached>
              <transformers>
                <transformer implementation=
                  "org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
                  <mainClass>org.example.Main</mainClass>
                </transformer>
              </transformers>
            </configuration>
          </execution>
        </executions>
      </plugin>

    </plugins>

</build>
```

Часть 2. Пакетные менеджеры

Maven и Gradle автоматизируют этот процесс, решая ключевые задачи:

1.Управление зависимостями (Dependency Management): Автоматическое скачивание библиотек, от которых зависит ваш проект, и всех их зависимостей (транзитивные зависимости) из центральных репозиториев (как Maven Central).

2.Сборка проекта (Build Tool): Стандартизация и автоматизация процессов:
    1.Компиляция исходного кода
    2.Запуск тестов
    3.Упаковка кода в артефакты (JAR, WAR, EAR файлы)
    4.Развертывание (деплой) артефактов

3.Стандартизация структуры проекта: Задают единую структуру папок (src/main/java, src/test/java), что позволяет любому разработчику быстро разобраться в любом проекте.

4.Управление жизненным циклом (Lifecycle): Определяют стандартный набор фаз (компиляция, тестирование, упаковка), которые выполняются по порядку.

Что выбрать?

•Maven: Отлично подходит для большинства стандартных проектов. Проще для изучения новичками благодаря своей предсказуемости.

•Gradle: Выбирают для больших и сложных проектов (например, Android-приложений, где он является стандартом de facto), где критична скорость сборки и нужна возможность тонкой настройки процесса.

Часть 2. Пакетные менеджеры

Основные команды:

- mvn compile - скомпилировать проект
- mvn test - запустить тесты
- mvn package - упаковать в JAR/WAR
- mvn clean - очистить целевую директорию
- mvn install - установить артефакт в локальный репозиторий (~/.m2)

Как они вписываются в CI/CD?

Оба инструмента являются фундаментом CI/CD-пайплайна. Именно их команды (mvn test, gradle build) выполняются на этапах сборки и тестирования внутри таких систем, как Jenkins, GitLab CI или GitHub Actions.

```yaml
- name: Build with Maven
  run: mvn compile

- name: Run tests with Maven
  run: mvn test
```

Пример CI

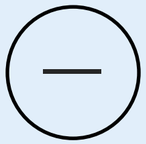
```yaml
name: Java CI with Maven

on:
  push:
    branches: [ "master" ]
  pull_request:
    branches: [ "master" ]

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3
      - name: Set up JDK 21
        uses: actions/setup-java@v3
        with:
          java-version: '21'
          distribution: 'temurin'
          cache: maven
      - name: Build with Maven
        run: mvn package
```

Выводы

1. Освойте Git на практике — начните использовать ветвление и Pull Requests в своих проектах
2. Автоматизируйте сборку и тестирование даже в учебных проектах с помощью GitHub Actions
3. Уделяйте внимание качеству коммитов — осмысленные сообщения и атомарные изменения
4. Изучайте не только инструменты, но и практики — код-ревью, тест-драйв разработку, принципы DevOps

Вопросы?

Ставим «+»,
если есть вопросы



Ставим «-»,
если нет вопросов

MTC True Tech

СПАСИБО ЗА ВНИМАНИЕ

Дмитрий Бобряков

Senior big data developer

dmitrybobryakov@gmail.com

@DmitryBobryakov