Теория и технология программирования

Программирование на языке Java

Лекция 14. Java Virtual Machine

Глухих Михаил Игоревич, к.т.н., доц.

mailto: glukhikh@mail.ru

Главный источник

- The Java Virtual Machine Specification
- Action “Show Bytecode” in IDEA

Основные понятия

- JVM = “виртуальный компьютер”, имеющий, как минимум, процессор и память
- JVM имеет собственный машинный язык (это НЕ Java, хотя язык сделан так, чтобы было удобно компилировать Java в этот язык)
- JVM имеет стековую архитектуру (в отличие от регистровой архитектуры большинства настоящих процессоров)
  - Стековая = операнды хранятся на стеке, количество регистров минимизировано, многие операции вроде ADD не имеют аргументов
  - Регистровая = операнды хранятся в регистрах, операции имеют аргументы, например ADD AX, BX

Программа для JVM

- Программа для JVM состоит из class-файлов, их формат строго специфицирован, каждый class-файл описывает ровно 1 класс
  - Отметим, что машинный язык «знает», что такое классы
- Поддерживаемые типы данных
  - Примитивные: **byte**, **short**, **int**, **long**, **char**, **float**, **double**, **boolean**, *returnAddress*
  - Ссылочные: class, interface, array, all are nullable
  - *returnAddress* ~ адрес инструкции в байт-коде, используется для вызова так называемых процедур (subroutine)

Устройство памяти JVM

- Регистр PC
- JVM stack = привязан к каждой JVM-нити, служит для хранения локальных переменных, параметров методов и операций, состоит из **фреймов**
- JVM heap = общий для всех нитей
- JVM method area = хранит машинные коды
- JVM constant pool = хранилище констант, своё для каждого класса
- JVM frame = создаётся для каждого вызванного метода на стеке, состоит из массива локальных переменных, **стека операндов** (Operand stack, не путать с JVM stack!), ссылки на пул констант для класса, к которому относится метод

Машинные инструкции

- Любая инструкция содержит код выполняемой операции размером один байт (поэтому – **байт-код**)
- Также **может** содержать дополнительные операнды
- Для вычислений используются типы: int, long, float, double, reference, return address
- Группы инструкций
  - LOAD (local variable → operand stack)
  - STORE (local variable ← operand stack
  - Константы: CONST / LDC
  - ADD / SUB / MUL / DIV / REM / NEG / SHR / SHL / USHR / AND / OR / XOR / INC / CMP
  - Преобразования типов вроде I2L
  - Стековые: POP / DUP / SWAP

Машинные инструкции

- Группы инструкций
  - Объекты: NEW / GETFIELD / PUTFIELD / GETSTATIC / PUTSTATIC / INSTANCEOF / CHECKCAST
  - Массивы: NEWARRAY / ALOAD / ASTORE / ARRAYLENGTH
  - Методы: INVOKEVIRTUAL / INVOKESTATIC / INVOKESPECIAL / INVOKEINTERFACE / RETURN
  - Передача управления: GOTO / IFEQ / IFNE / IF<other> / TABLESWITCH / LOOKUPSWITCH
  - Процедуры (не путать с методами): JSR / RET

Вызовы методов

- Нужный метод всегда указывается через пул констант, там содержится его имя в виде BaseClass.doSomething
- Ссылка на константу – аргумент invoke
- INVOKEVIRTUAL #index
  - Берёт с вершины стека операндов будущий this, так называемый receiver (получатель)
  - Оттуда же берёт аргументы
  - Ищет нужный метод вначале внутри класса получателя (в соответствии с его **реальным** типом времени выполнения), потом его базовых классов, вызывает первый найденный

Вызовы методов

- INVOKEINTERFACE #index
  - Работает примерно как INVOKEVIRTUAL
  - Основная разница в том, что базовый метод определён в интерфейсе, а не в классе
- INVOKESTATIC #index
  - Не имеет получателя
  - Берёт аргументы с вершины стека
  - Вызывает именно тот метод, название которого было взято из пула констант

Вызовы методов

- INVOKESPECIAL #index
  - К специальным методам относятся конструктор, вызов метода базового класса из метода производного класса, вызов закрытого метода (из метода того же класса)
  - Конструктор в байт-коде имеет название вида MyClass.<init>
  - В большинстве случаев INVOKESPECIAL вызывает тот метод, который указан непосредственно в ней (исключение составляют защищённые методы базового класса)

Примеры формирования байт-кода

- См. в IDE

Языки для JVM

- Java (1995) #1 in Tiobe Index, ~18%
  ~ 65000 projects with 5+ stars at GitHub
- Groovy (2003) #19 in Tiobe Index, 1.8%
  ~ 2000 projects with 5+ stars at GitHub
- Scala (2004) #32 in Tiobe Index, 0.7%
  ~ 7000 projects with 5+ stars at GitHub
- Clojure (2007) #50 in Tiobe Index, 0.2%
  ~ 5000 projects with 5+ stars at GitHub
- Kotlin (2016) ~ #90 in Tiobe Index
  ~ 700 projects with 5+ stars at GitHub
- Gosu (2014) ~ 40 projects with 5+ stars at GitHub
- Ceylon (2011) ~ 30 projects with 5+ stars at GitHub
- Fantom (2005) no information