ОГЛАВЛЕНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ПО — программное обеспечение.

ЯП — язык программирования.

Целевой язык —

JVM —

Байт-код —

Дерево разбора PSI —

ВВЕДЕНИЕ

В области разработки программного обеспечения довольно часто поднимается вопрос производительности разрабатываемых программ. Требование к производительности — одно из важнейших нефункциональных требований для большинства продуктов.

На пути от написания кода программы до исполнения соответствующего ей машинного кода на процессоре компьютера пользователя есть множество факторов, которые так или иначе могут влиять на производительность разрабатываемой программы.

Не углубляясь в конкретные факторы, выделим стадии с этими факторами от написания кода до исполнения машинного кода программы на процессоре:

- написание кода программистом,
- компиляция кода (некоторые стадии могут отсутствовать, либо быть совмещены):
    1. лексический анализ,
    2. синтаксический анализ,
    3. семантический анализ,
    4. оптимизации на абстрактном синтаксическом дереве,
    5. трансляция в промежуточное представление,
    6. машинно-независимые оптимизации,
    7. трансляция в конечное представление (машинный код),
    8. машинно-зависимые оптимизации.
- интерпретация или компиляция «на лету» промежуточного кода виртуальной машиной (при трансляции кода компилятором не в машинный код, а в код некоторой виртуальной машины),
- планирование исполнения машинного кода ядром операционной системы,

- исполнение кода программы на процессоре,

- исполнение кода функций библиотек поддержки времени исполнения, используемых в программе,

- исполнение кода системных вызовов, используемых в программе.

На каждой из перечисленных стадий существует множество факторов, способных в конечном счете повлиять на производительность программы. Под контролем программиста непосредственно целевой программы находится лишь одна группа факторов. За остальные группы факторов ответственны разработчики соответствующих инструментов и вспомогательных программ или их частей: виртуальных машин, библиотек поддержки времени исполнения, ядер операционных систем и непосредственно самих процессоров и других физических компонентов, способных влиять на производительность исполняемой программы.

В данной работе предлагается провести исследование, связанное с анализом влияния на производительность программ групп факторов в рамках стадий написания кода и его компиляции (в терминах теории компиляторов — в рамках стадии анализа, опуская стадию синтеза).

Хочется сразу отметить, что результаты такого анализа могут быть использованы как разработчиками целевых программ, так и разработчиками языка программирования, на котором эти программы составляются.

В качестве языка программирования, программы на котором и компилятор которого будут исследоваться, был выбрал Kotlin, как один из наиболее интересных и быстро развивающихся языков.

Таким образом, целью данной магистерской диссертации является разработка набора инструментов для анализа исходного кода программ на ЯП Kotlin и выявления потенциальных проблем производительности в них, а также проведение исследования по данной теме с использованием разработанного набора инструментов.

Практическая значимость работы заключается в получении по результатам исследования и разработки:

1. сгруппированного списка файлов с исходным кодом из достаточно объемного набора данных, являющихся с точки зрения тех или иных алгоритмов аномальным;
2. набора инструментов, позволяющего получать аналогичный список файлов на заданном проекте.

Первый результат должен стать важным и полезным в первую очередь разработчикам языка программирования Kotlin; второй же — пользователям языка — программистам, использующих для разработки своих проектов язык программирования Kotlin.

1 АНАЛИЗ ПРДЕМЕТНОЙ ОБЛАСТИ И ПОСТАНОВКА ЗАДАЧИ

1.1 Анализ предметной области

Предметной областью, как уже было сказано, является анализ исходного кода на языке программирования Kotlin (и других представлений этого исходного кода) с целью обнаружения потенциальных проблем производительности в программах, написанных на нём.

Компилятор ЯП Kotlin транслирует исходный код в промежуточный код (называемый также байт-кодом) — набор инструкций для виртуальной машины ЯП Java — JVM. Исследование предлагается проводить в рамках представлений исходного кода, ответственность за которые лежит непосредственно на стороне компилятора Kotlin (дальнейшая интепретация или компиляция «на лету» в машинный байт-код JVM и исполнение кода рассматриваться не будут, т. к. данные стадии лежат за рамками ответственности компилятора Kotlin).

В компиляторе Kotlin производятся следующие преобразования исходного кода:

1. Преобразование исходного кода в набор лексем (токенизация или лексический анализ);
2. Построение дерева разбора на основе набора лексем (в компиляторе Kotlin данное дерево именуется как PSI — Program Structure Interface — по общепринятой терминологии его можно назвать конкретным синтаксическим деревом);
3. Генерация байт-кода JVM на основе дерева разбора PSI.

Стоит отметить, что построение дерева разбора PSI возможно на любом исходном коде. Для синтаксически некорректных конструкций (тех, которые не вписываются в грамматику ЯП Kotlin) в дереве разбора будет создаваться узел специального типа; в то время, как генерация байт-кода JVM по исходному коду с синтаксически и/или семантически некорректными конструкциями невозможна.

К сожалению, на данный момент в компиляторе Kotlin нет представления исходного кода в виде абстрактного синтаксического дерева при компиляции в

целевой язык — JVM байт-код. Генерация JVM байт-кода производится напрямую по дереву разбора PSI.

Дерево разбора PSI, как уже было сказано, представляет из себя конкретное синтаксическое дерево. В отличие от абстрактного синтаксического дерева в нём в том числе содержаться узлы, несущие исключительно синтаксический характер (не несущие семантический характер с точки зрения целевого языка — например, отступы, пробелы, комментарии и т. д.). Листья данного дерева, как правило, соответствуют конкретным лексемам (таким образом, список листьев данного дерева является набором лексем, которые в свою очередь являются результатом работы лексического анализатора в компиляторе Kotlin). Корень дерева разбора PSI является узлом с типом FILE, который представляет весь исходный код анализируемого файла. Таким образом, по данному дереву разбора может быть однозначно восстановлен исходный код программы.

1.2 Постановка задачи

…

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. …