Системы динамической компиляции в JVM

Николай Иготти

Требования к компилятору

- Не требуется стандартом Java
- Основное поведенческое требование - **прозрачность**
- Функционально – только для повышения производительности
- Что делает компилятор?
- Степень управляемости
- Нежелательный источник недетерминизма

Что компилировать?

- Всё
- Классы
- Методы
- Базовые блоки
- Ещё?

Когда компилировать?

- Предварительно (AOT)
  - *Больше ресурсов на оптимизацию*
  - *Предсказуемый результат, проще поиск ошибок в компиляторе*
  - *Новый стандарт на промежуточное представление (или компиляция в существующие стандарты)*
  - *Сложности деоптимизации*
- На лету (JIT)
  - *Больше информации для оптимизации*
    - Аппаратное обеспечение
    - Контекст компиляции
    - Типичное поведение
  - *Деоптимизация по мере необходимости*
  - *Что компилировать (déjà vu)?*
- Комбинированные подходы
  - *Подсказки JITу*
  - *Сохранение исходного байткода*

Профилировка и динамическая оптимизация

- Цикл разработки: написание – компиляция-отладка – профилировка – оптимизация
- Объекты оптимизации: hotspots, common paths, дорогие операции
- Чем может помочь динамический профилятор+компилятор:
  - *Обнаружение часто исполняемого кода*
  - *Оптимизация под common path*
  - *Исключение дорогих операций*
  - *Реоптимизация при изменении нагрузки*

Простой компилятор

- Параметризированные шаблоны – кусочки машинного кода
- Трансляция сводится к последовательной замене байткодов на шаблоны
- Выигрыш:
  - *Не нужен диспетчер переходящий к следующему байткоду*
  - *Можно хранить часто используемые значения, например TOS в машинных регистрах*
  - *Ещё?*

Оптимизирующий комилятор

- *Анализ реальной семантики байткода (в IR)*
- *Оптимизация:*
  - *Стандартные компиляторные оптимизации (какие?)*
  - *Инлайнинг*
  - *Доступы к памяти*
  - *Выделение машинных регистров*
  - *Обработка NPE*
  - *Блокировки*
  - *Барьеры коллектора*
  - *Построение кода оптимизированного под типичное исполнение и типичные данные*

Деоптимизация и реоптимизация

- Динамический компилятор делает оптимистичекие предположения
- ***Tempora mutantur…***
- *Может быть необходимо остановить исполнение скомпилированного кода и получить его состояние*
- ***… et nos mutamur in illis***
- Заново породить код с учетом новых условий
- *Продолжить исполнение кода из заданного состояния*

Компилятору необходимо взаимодействие с…

- Коллектором
- Интерпретатором (если есть)
- Менеджером блокировок
- Менеджером исключений Ява
- Загрузчиком классов (зачем?)
- Отладчиком Ява (зачем?)
- Менеджером исключительных ситуаций ОС
- Коллектор релоцирует ссылки в порождённом коде

Вопросы

- Чем ещё хорош AOT?
- Какой язык более “динамический” – Java или C++? Как это влияет на компиляторы с этих языков?
- Какие оптимизации осмысленны на уровне Java -> байткод, а какие на уровне байткод -> машинный код?
- Как reflection может влиять на дизайн компилятора (AOT?, JIT?)