ОС Фантом

Сборка мусора в персистентной среде и интеграция со средой Java

Кратко о Фантом

- ОС Фантом обеспечивает своим приложениям персистентность - прозрачное продолжение работы после рестарта системы.
- Приложения выполняются в виртуальной машине Фантом - близким аналогом является JVM или рантайм .Net.

Задачи

- Полная сборка мусора на терабайтных объёмах виртуальной (отображённой на диск) памяти без остановки системы
- Достаточно прозрачное взаимодействие естественной объектной среды Фантом и компилятора Java

Java - проблемы

- Видимость классов Фантом для исходников на Java
- Совместимая схема вызова конструкторов (байткод, статические методы, компилятор, выявление и генерация конструкторов)

Как пишется компилятор

- Регресс-тесты: сформированы референсные листинги AST-дерева и байткода, ведётся анализ изменений
- Виртуальная машина содержит инструкцию отладки и умеет трассировать исполнение байт-кода

Минорные тяжести

- Длительное время в компиляторе не было численных типов кроме int, модификация казалась тривиальной, но повлекла тектонические изменения
- Статические вызовы методов потребовали новых проверок типов в ядре: виртуальные методы точно принадлежат классу в силу вызова через VMT, для статических класс задаётся в байткоде и должен быть проверен.

Сборка мусора

- Два сборщика - быстрый и неполный в runtime (был давно) и медленный, по снапшоту и полный (наконец-то появился)
- Полный сборщик может приносить дополнительную пользу

Полное сканирование

- Полный сборщик мусора сканирует абсолютно все достижимые объекты и может проанализировать недостижимые.
- Можно верифицировать неполный сборщик.
- Можно собирать статистику по размерам объекта и степени связности
- Хотелось бы генерировать рекомендации по группировке объектов, но нет.

Escape-анализ

- Мы контролируем компилятор и можем провести анализ достижимости объекта извне функции и аллоцировать его на стеке, избежав полностью задачи сборки мусора для него. Таких объектов много, польза велика.
- Виртуальная машина не может доверять байткоду и принимать на веру не-escape-ность объекта
- Хорошо бы анализировать на приличную глубину вызовов функций

JIT?

- Escape-анализ на глубину в несколько вызовов делать сложно. Проще поступить как в Java - агрессивно инлайнить функции автоматически. Это приносит пользу и по другим причинам, но, в частности, позволяет делать более широкий escape-анализ.
- Такой онлайн можно делать в компиляторе, в загрузчике и в рамках… JIT-кодогенератора.

Контакты

- Дмитрий Завалишин
- dz@dz.ru
- Digital Zone: http://dz.ru
- DZ Systems: http://dzsystems.ru