BELLSOFT

Особенности фаззинга JIT в Liberica JDK

WWW.BELL-SW.COM

2020

BellSoft - благородные корни

20 лет истории инжиниринга и профессионализма

1997

MCST – Moscow Center of SPARC Technologies
The company worked as a contractor for Sun Microsystems

2004

Sun Microsystems founded its own Development Center in Saint-Petersburg.

2010

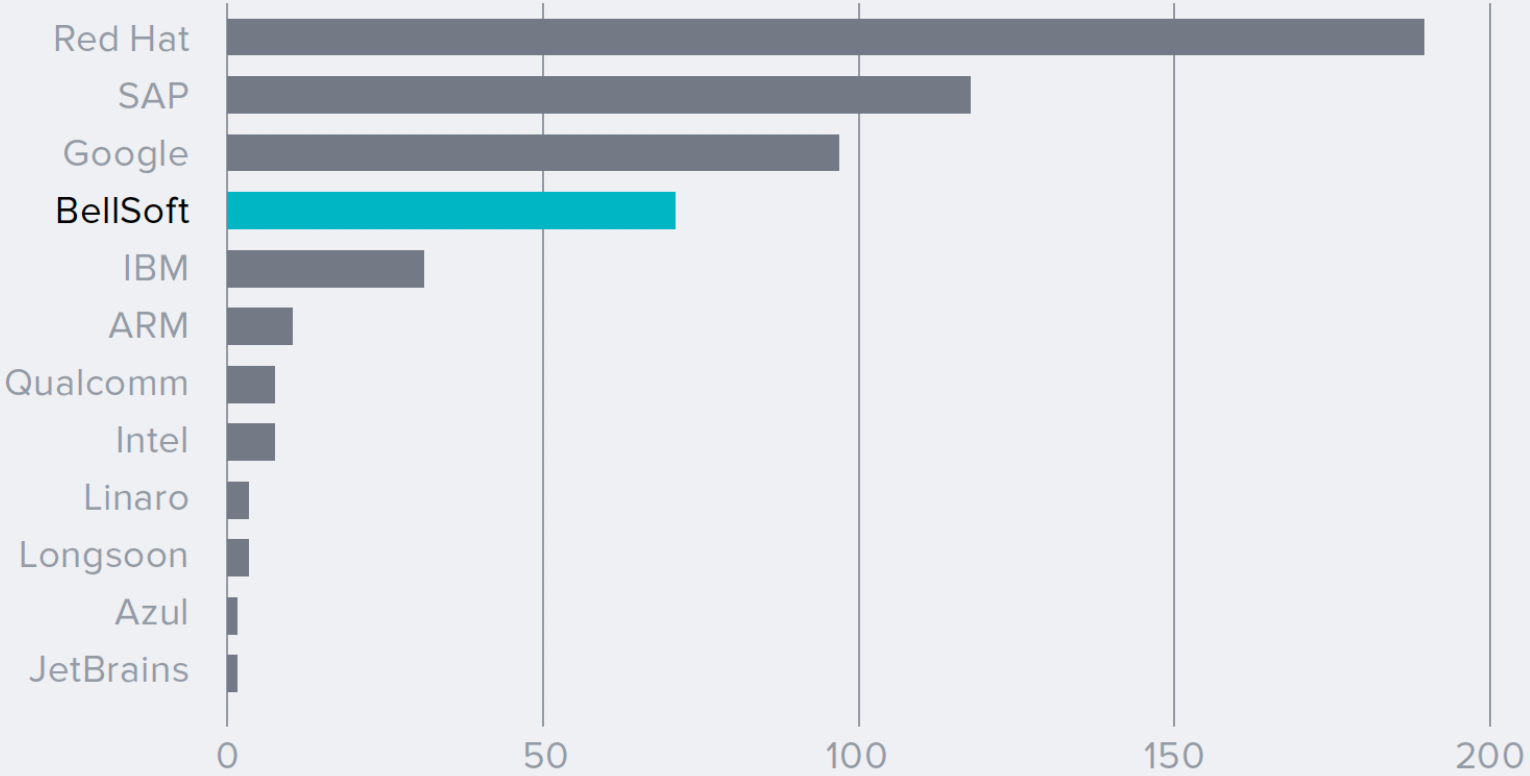
Oracle Corp. acquired Sun Microsystems

2017

Проекты с открытым кодом

We contribute to

Number of external contributions to OpenJDK upstream (8.2017 - 8.2018)



Альтернативная Java от BellSoft - Liberica JDK

- Liberica Java 8, Java 11 поддерживаемые платформы:
  - Linux x86 32 и 64 bit
  - Windows 32 и 64 bit
  - MacOS 64 bit
  - Linux ARM 32 и 64
  - SPARC & x86 Solaris 10, 11
  - ELBRUS, Baikal

- Обновления безопасности Liberica JDK выходят параллельно с Java обновлениями
- Liberica верифицирована TCK-тестами (BellSoft имеет лицензию TCK http://openjdk.java.net/groups/conformance/JckAccess/jck-access.html)

Формы поставки

- Пакеты и инсталляторы
  - Windows binaries (with installer)
  - MacOS binaries (with installer)
  - Linux Repos: APT, YUM
  - Docker containers at hub.docker.com
    - Full JDK with Debian or CentOS
    - Liberica JDK Lite with Alpine Linux – only 96 MB
      - smallest Java 11 Docker container in the world
- Российские OS
  - Astra Linux
  - ALT Linux
  - РЕД ОС
  - РОСА

Своевременные обновления безопасности

Коллективный подход

https://openjdk.java.net/groups/vulnerability/

BellSoft является участником закрытой группы по безопасности OpenJDK, что позволяет своевременно реагировать и выпускать обновления и патчи безопасности

Liberica JDK 8u262 выпущен в течение 2 часов с момента выхода Java SE

Java – это качественный продукт !

Да, но нужно помнить о 20 000 открытых багов в OpenJDK

Динамическая компиляция в HotSpot JVM

Тестирование JIT в Liberica JDK

- Тесты на соответствие стандарту Java SE (TCK)
- Статические тесты (jtreg)
- Реальные приложения и их тесты (HiBench, Elastic, итд)
- Статический анализ (SVACE)
- Тестирование производительности
- Фаззинг

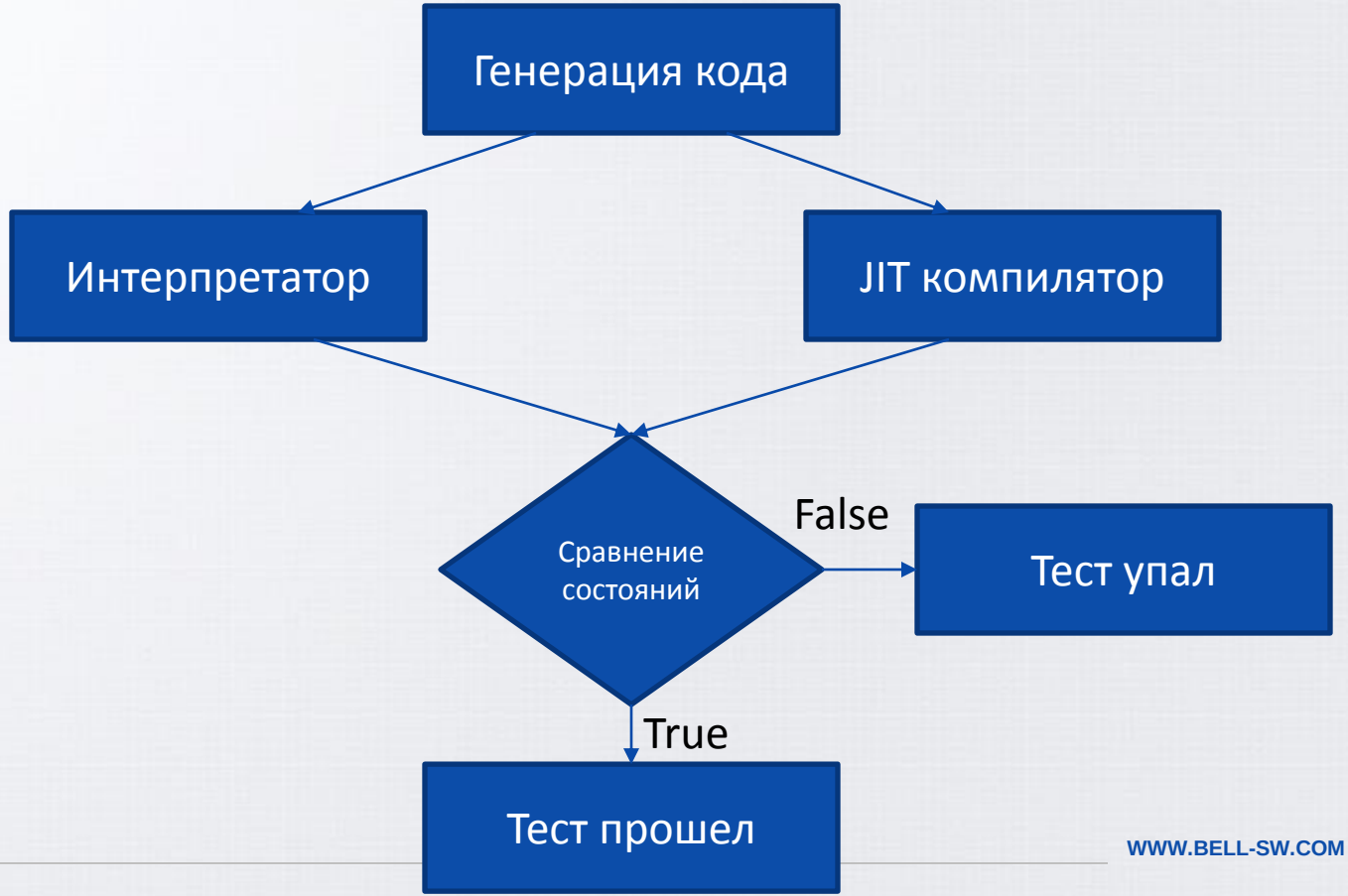
Классический фаззинг

- Шифрование
- Веб-приложения
- Базы данных

Особенности JIT компиляции

- Профиль постоянно меняется
- Порядок компиляции также меняется
- Перекомпиляция методов происходит множество раз

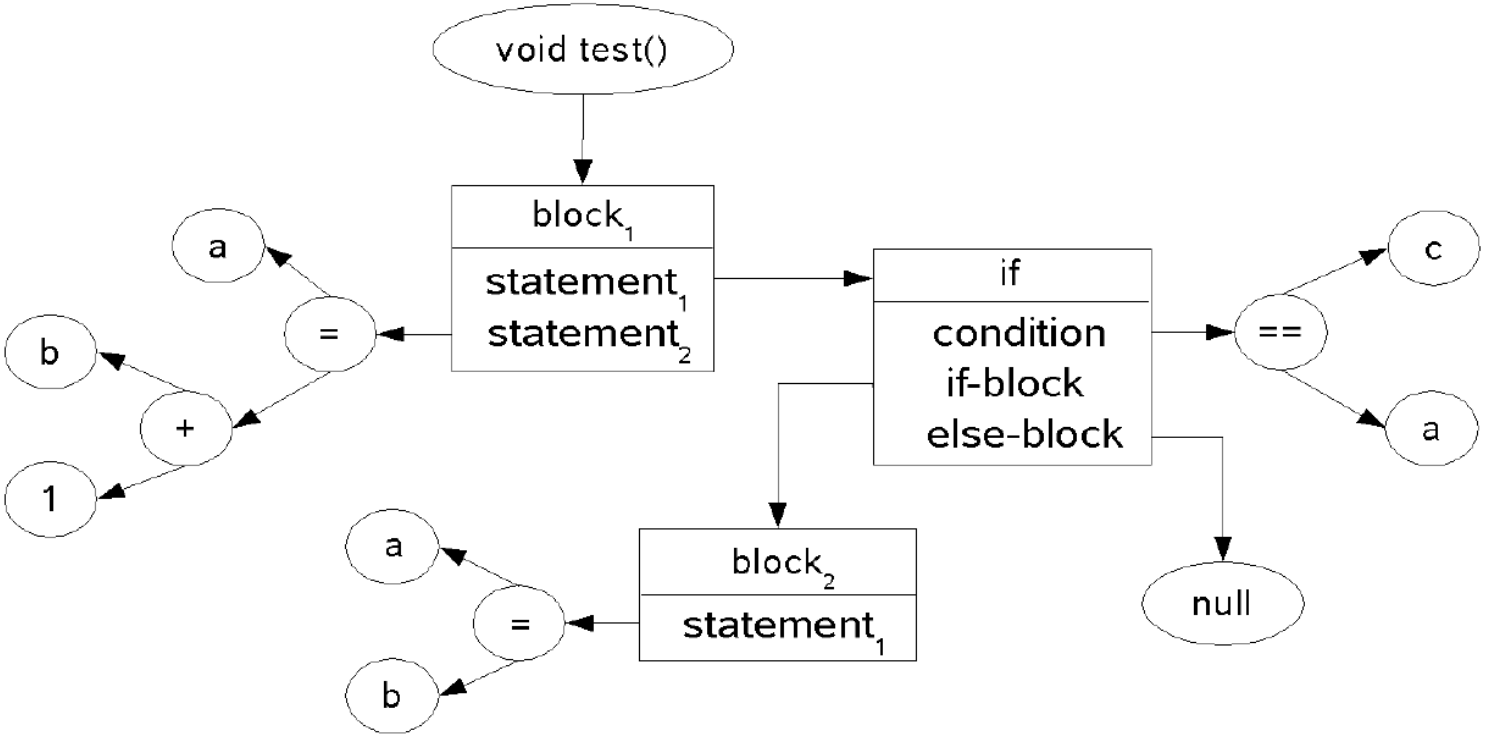
Фаззер JIT


Генерация кода
Интерпретатор
JIT компилятор
Сравнение состояний
False
Тест упал
True
Тест прошел

Автоматический генератор тестов

- Логика генерации кода может быть описана модифицированной Формой Бэкуса-Наура
- Реализация этой генерации для каждого теста представляет собой Марковскую цепь.

Шаг 1: Генерация IR



Шаг 2: генерация кода на базе IR


```
void test() {
if (a==c) {
    a=b;
}
a = b + 1;
}
```

Имплементация

- Генерация унарных, бинарных и тернарных выражений
- Up-cast и Down-cast
- Массивы
- Управление исполнением
- Функциональные вызовы
- Синтез иерархии классов
- Расширяемость
- Ограничение времени исполнения

Использование JIT фаззера

- Написан на Java
- Генерация тестов в формате jtreg
- Каждый тестовый цикл генерируются 10К тестов
- Результаты
  - В первые 6 месяцев использования было обнаружено более 20 дефектов
  - Сейчас в среднем 1 баг в 6 месяцев

Следующие шаги

- Потенциально эффективным будет расширить JIT фаззер генератором валидного байт-кода. Это позволит получать конструкции недостижимые для компилятора javac


@gigabel
@bellsoftware
info@libericajdk.ru