Генерация и выполнение JVM-байткода «на лету»

Алексей Владыкин

23.05.2012

План лекции

1. Модельная задача
2. Знакомство с байткодом JVM
3. Байткод своими руками

План лекции

- Есть набор интерфейсов с getter'ами.
- Некоторые могут возвращать специальное значение "!!!".

```
public interface MyInterface {
    String getFoo();
    String getBar();
    String getBaz();
    /* ... */
}
```

- Требуется написать классы-обертки, заменяющие "!!!" на null, и умеющие сообщать, где это было сделано.

```
MyInterface obj = getObject(...);
assert "!!!" == obj.getFoo();

MyInterface wrapper = new Wrapper(obj);
assert null == wrapper.getFoo();
assert ["Foo"] == getSpecialFields(wrapper);
```

Варианты реализации

- Руками
- Build-time кодогенерация
- **Run-time кодогенерация**
- java.lang.reflect.Proxy

План лекции

- The Java Virtual Machine Specification
- The Java Language Specification
- `http://docs.oracle.com/javase/specs/`

Структура .class файла

- Заголовок (`CAFEBABE`, версия формата)

Структура .class файла

- Заголовок (`CAFEBABE`, версия формата)
- Constant pool (числа, строки, имена классов, полей и методов)

Структура .class файла

- Заголовок (`CAFEBABE`, версия формата)
- Constant pool (числа, строки, имена классов, полей и методов)
- Объявление класса (модификаторы, имя класса, имя суперкласса, имена реализуемых интерфейсов)

Структура .class файла

- Заголовок (`CAFEBABE`, версия формата)
- Constant pool (числа, строки, имена классов, полей и методов)
- Объявление класса (модификаторы, имя класса, имя суперкласса, имена реализуемых интерфейсов)
- Поля класса

Структура .class файла

- Заголовок (`CAFEBABE`, версия формата)
- Constant pool (числа, строки, имена классов, полей и методов)
- Объявление класса (модификаторы, имя класса, имя суперкласса, имена реализуемых интерфейсов)
- Поля класса
- Методы класса

Структура .class файла

- Заголовок (`CAFEBABE`, версия формата)
- Constant pool (числа, строки, имена классов, полей и методов)
- Объявление класса (модификаторы, имя класса, имя суперкласса, имена реализуемых интерфейсов)
- Поля класса
- Методы класса
- Атрибуты класса (аннотации, debug info, . . .)

```
javap -v -p java.lang.String
```

Имена в байткоде

- Никаких импортов, все имена полные
- Имена классов: `java/lang/String`

Имена в байткоде

- Никаких импортов, все имена полные
- Имена классов: `java/lang/String`
- Имена типов:
  - `B`, `C`, `D`, `F`, `I`, `J`, `S`, `Z`
  - `Ljava/lang/Object;`
  - `[[I`

Имена в байткоде

- Никаких импортов, все имена полные
- Имена классов: `java/lang/String`
- Имена типов:
  - `B`, `C`, `D`, `F`, `I`, `J`, `S`, `Z`
  - `Ljava/lang/Object;`
  - `[[I`
- Имена методов:
  - `<init> ()V`
  - `<clinit> ()V`
  - `equals (Ljava/lang/Object;)Z`
  - `toString ()Ljava/lang/String;`
  - `sort ([III)V`

Имена в байткоде

- Никаких импортов, все имена полные
- Имена классов: `java/lang/String`
- Имена типов:
  - `B`, `C`, `D`, `F`, `I`, `J`, `S`, `Z`
  - `Ljava/lang/Object;`
  - `[[I`
- Имена методов:
  - `<init> ()V`
  - `<clinit> ()V`
  - `equals (Ljava/lang/Object;)Z`
  - `toString ()Ljava/lang/String;`
  - `sort ([III)V`
- Имена параметров и локальных переменных отсутствуют

Исполняемый код

- Состоит из простых инструкций (около 200)

Исполняемый код

- Состоит из простых инструкций (около 200)
- Работает в рамках одного фрейма стека вызовов

Исполняемый код

- Состоит из простых инструкций (около 200)
- Работает в рамках одного фрейма стека вызовов
- Имеет локальный стек заданного размера, где и выполняет все вычисления

Исполняемый код

- Состоит из простых инструкций (около 200)
- Работает в рамках одного фрейма стека вызовов
- Имеет локальный стек заданного размера, где и выполняет все вычисления
- Может обращаться к полям и методам объектов, а также к своим аргументам и локальным переменным

Стековая арифметика

- $2 + 3 * 4$

Стековая арифметика

- $2 + 3 * 4$
- $2\ 3\ 4 * +$

Стековая арифметика

- $2 + 3 * 4$
- $2\ 3\ 4 * +$
- ```
  iconst_2
  iconst_3
  iconst_4
  imul
  iadd
  ```
  ```

Основные инструкции

- Значения в стеке и локальных переменных: `*const*`, `ldc*`, `*load*`, `*store*`

Основные инструкции

- Значения в стеке и локальных переменных:
  `*const*`, `ldc*`, `*load*`, `*store*`
- Арифметика:
  `*mul`, `*div`, `*add`, `*sub`

Основные инструкции

- Значения в стеке и локальных переменных:
  `*const*`, `ldc*`, `*load*`, `*store*`
- Арифметика:
  `*mul`, `*div`, `*add`, `*sub`
- Работа с объектами:
  `new`, `getfield`, `putfield`, `getstatic`, `putstatic`

Основные инструкции

- Значения в стеке и локальных переменных:
  `*const*`, `ldc*`, `*load*`, `*store*`
- Арифметика:
  `*mul`, `*div`, `*add`, `*sub`
- Работа с объектами:
  `new`, `getfield`, `putfield`, `getstatic`, `putstatic`
- Вызовы методов:
  `invokestatic`, `invokevirtual`, `invokespecial`, `*return`

Основные инструкции

- Значения в стеке и локальных переменных:
  `*const*`, `ldc*`, `*load*`, `*store*`
- Арифметика:
  `*mul`, `*div`, `*add`, `*sub`
- Работа с объектами:
  `new`, `getfield`, `putfield`, `getstatic`, `putstatic`
- Вызовы методов:
  `invokestatic`, `invokevirtual`, `invokespecial`, `*return`
- Проверки и переходы:
  `*cmp`, `if*`, `goto*`

План лекции

Библиотека ASM

- “all purpose Java bytecode manipulation and analysis framework [...] focused on simplicity of use and performance“
- Подходит для чтения, трансформации и создания новых классов
- Берет на себя constant pool и адресацию в коде
- `http://asm.ow2.org/`

Visitor pattern

```
interface ClassVisitor {
    void visit(...);
    AnnotationVisitor visitAnnotation(...);
    FieldVisitor visitField(...);
    MethodVisitor visitMethod(...);
    ...
    void visitEnd();
}

class ClassReader {...}
class ClassAdapter implements ClassVisitor {...}
class ClassWriter implements ClassVisitor {...}
```

Пример HelloWorld

```
ClassWriter cw = new ClassWriter(0);
cw.visit(V1_1, ACC_PUBLIC, "HelloWorld",
    null, "java/lang/Object", null);
MethodVisitor mw = cw.visitMethod(
    ACC_PUBLIC | ACC_STATIC, "main",
    "([Ljava/lang/String;)V", null, null);
mw.visitFieldInsn(GETSTATIC, "java/lang/System",
    "out", "Ljava/io/PrintStream;");
mw.visitLdcInsn("Hello world!");
mw.visitMethodInsn(INVOKEVIRTUAL,
    "java/io/PrintStream", "println",
    "(Ljava/lang/String;)V");
mw.visitInsn(RETURN);
mw.visitMaxs(2, 2);
mw.visitEnd();
cw.visitEnd();
```

Пример Wrapper

```
ClassWriter cw = new ClassWriter(0);
cv.visit(V1_5,
    ACC_PUBLIC | ACC_FINAL,
    "Wrapper", null,
    "java/lang/Object",
    new String[] {"MyInterface"});

cv.visitField(ACC_PRIVATE | ACC_FINAL,
     "delegate",
     "MyInterface", null, null);

// ...

cv.visitEnd();
```

Пример Wrapper (конструктор)

```
MethodVisitor mv = cv.visitMethod(ACC_PUBLIC,
    "<init>", "(LMyInterface;)V", null, null);
mv.visitMaxs(2, 2);
mv.visitCode();
mv.visitVarInsn(ALOAD, 0);
mv.visitMethodInsn(INVOKESPECIAL,
    "java/lang/Object", "<init>", "()V");
mv.visitVarInsn(ALOAD, 0);
mv.visitVarInsn(ALOAD, 1);
mv.visitFieldInsn(PUTFIELD,
    "Wrapper",
    "delegate", "LMyInterface;");
mv.visitInsn(RETURN);
mv.visitEnd();
```

Пример Wrapper (getFoo)

```
// generate call to delegate.getFoo();
mv.visitInsn(DUP);
mv.visitLdcInsn("!!!");
mv.visitInsn(SWAP);
mv.visitMethodInsn(INVOKEVIRTUAL,
    "java/lang/String", "equals",
    "(Ljava/lang/Object;)Z");

Label lbl = new Label();
mv.visitJumpInsn(IFEQ, lbl);
mv.visitInsn(POP);
mv.visitInsn(ACONST_NULL);
mv.visitLabel(lbl);
mv.visitInsn(ARETURN);
```

Пример Wrapper (getSpecialFields)

Оставляю в качестве упражнения :)

Подгрузка класса в runtime

- `ClassLoader.defineClass()`
- На загрузке работает Class Verifier, может бросить `VerifyError`
- `Class.newInstance()`

Другие применения генерации байткода

- Библиотеки аспектно-ориентированного программирования
- Реализации Java Persistence API
- Инструменты измерения покрытия кода
- Компиляторы

Спасибо за внимание!
Вопросы?

http://alexey.vladykin.name/
vladykin@gmail.com