JVM常用工具介绍

## Javap-查阅java字节码

Javap是一个能将class文件反汇编成为人类可读格式的工具。

$ javap -p -v Foo

$ javac Foo.java

$ javap -p -v Foo

Classfile ../Foo.class

Last modified ..; size 541 bytes

MD5 checksum 3828cdfbba56fea1da6c8d94fd13b20d

Compiled from "Foo.java"

public class Foo

minor version: 0

major version: 54

flags: (0x0021) ACC\_PUBLIC, ACC\_SUPER

this\_class: #7 // Foo

super\_class: #8 // java/lang/Object

interfaces: 0, fields: 4, methods: 2, attributes: 1

Constant pool:

#1 = Methodref #8.#23 // java/lang/Object."<init>":()V

#2 = Fieldref #7.#24 // Foo.tryBlock:I

#3 = Fieldref #7.#25 // Foo.finallyBlock:I

#4 = Class #26 // java/lang/Exception

#5 = Fieldref #7.#27 // Foo.catchBlock:I

#6 = Fieldref #7.#28 // Foo.methodExit:I

#7 = Class #29 // Foo

#8 = Class #30 // java/lang/Object

#9 = Utf8 tryBlock

#10 = Utf8 I

#11 = Utf8 catchBlock

#12 = Utf8 finallyBlock

#13 = Utf8 methodExit

#14 = Utf8 <init>

#15 = Utf8 ()V

#16 = Utf8 Code

#17 = Utf8 LineNumberTable

#18 = Utf8 test

#19 = Utf8 StackMapTable

#20 = Class #31 // java/lang/Throwable

#21 = Utf8 SourceFile

#22 = Utf8 Foo.java

#23 = NameAndType #14:#15 // "<init>":()V

#24 = NameAndType #9:#10 // tryBlock:I

#25 = NameAndType #12:#10 // finallyBlock:I

#26 = Utf8 java/lang/Exception

#27 = NameAndType #11:#10 // catchBlock:I

#28 = NameAndType #13:#10 // methodExit:I

#29 = Utf8 Foo

#30 = Utf8 java/lang/Object

#31 = Utf8 java/lang/Throwable

{

private int tryBlock;

descriptor: I

flags: (0x0002) ACC\_PRIVATE

private int catchBlock;

descriptor: I

flags: (0x0002) ACC\_PRIVATE

private int finallyBlock;

descriptor: I

flags: (0x0002) ACC\_PRIVATE

private int methodExit;

descriptor: I

flags: (0x0002) ACC\_PRIVATE

public Foo();

descriptor: ()V

flags: (0x0001) ACC\_PUBLIC

Code:

stack=1, locals=1, args\_size=1

0: aload\_0

1: invokespecial #1 // Method java/lang/Object."<init>":()V

4: return

LineNumberTable:

line 1: 0

public void test();

descriptor: ()V

flags: (0x0001) ACC\_PUBLIC

Code:

stack=2, locals=3, args\_size=1

0: aload\_0

1: iconst\_0

2: putfield #2 // Field tryBlock:I

5: aload\_0

6: iconst\_2

7: putfield #3 // Field finallyBlock:I

10: goto 35

13: astore\_1

14: aload\_0

15: iconst\_1

16: putfield #5 // Field catchBlock:I

19: aload\_0

20: iconst\_2

21: putfield #3 // Field finallyBlock:I

24: goto 35

27: astore\_2

28: aload\_0

29: iconst\_2

30: putfield #3 // Field finallyBlock:I

33: aload\_2

34: athrow

35: aload\_0

36: iconst\_3

37: putfield #6 // Field methodExit:I

40: return

Exception table:

from to target type

0 5 13 Class java/lang/Exception

0 5 27 any

13 19 27 any

LineNumberTable:

line 9: 0

line 13: 5

line 14: 10

line 10: 13

line 11: 14

line 13: 19

line 14: 24

line 13: 27

line 14: 33

line 15: 35

line 16: 40

StackMapTable: number\_of\_entries = 3

frame\_type = 77 /\* same\_locals\_1\_stack\_item \*/

stack = [ class java/lang/Exception ]

frame\_type = 77 /\* same\_locals\_1\_stack\_item \*/

stack = [ class java/lang/Throwable ]

frame\_type = 7 /\* same \*/

}

SourceFile: "Foo.java"

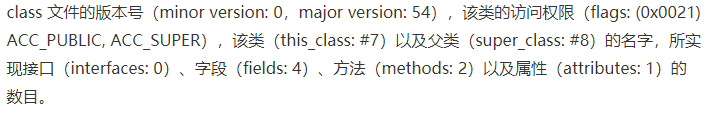
### -P

默认情况下，javap会打印所有非私有的字段和方法，当加了-p之后，他还将打印私有的字段和方法。

### -V

尽可能的打印所有信息

* 基本信息，涵盖源class文件的相关信息。



这里属性指的是class文件所携带的辅助信息。比如该class文件的源文件的名称。这类信息通常被用于java虚机的验证和运行，以及java程序的调试，一般无需了解。

Classfile ../Foo.class

Last modified ..; size 541 bytes

MD5 checksum 3828cdfbba56fea1da6c8d94fd13b20d

Compiled from "Foo.java"

public class Foo

minor version: 0

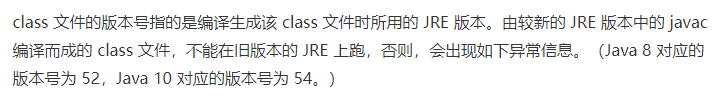
major version: 54

flags: (0x0021) ACC\_PUBLIC, ACC\_SUPER

this\_class: #7 // Foo

super\_class: #8 // java/lang/Object

interfaces: 0, fields: 4, methods: 2, attributes: 1



Exception in thread "main" java.lang.UnsupportedClassVersionError: Foo has been compiled by a more recent version of the Java Runtime (class file version 54.0), this version of the Java Runtime only recognizes class file versions up to 52.0

类的访问权限通常为ACC\_开头。具体每个常量的意义可以查询java虚机规范

* 常量池，用来存放各种常量以及符号引用

常量池的每一项都有一个对应的索引（如#1），并且可能应用其他的常量池项（#1 = Methodref #8.#23）

Constant pool:

#1 = Methodref #8.#23 // java/lang/Object."<init>":()V

...

#8 = Class #30 // java/lang/Object

...

#14 = Utf8 <init>

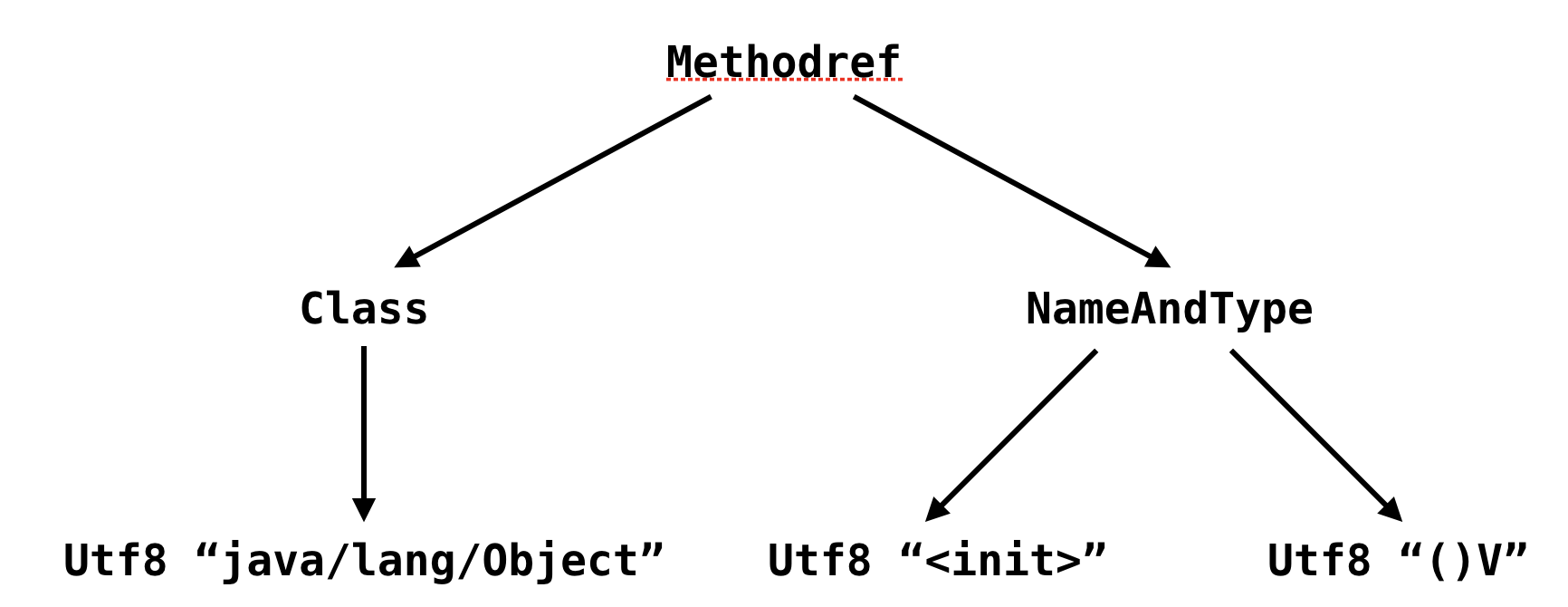
#15 = Utf8 ()V

...

#23 = NameAndType #14:#15 // "<init>":()V

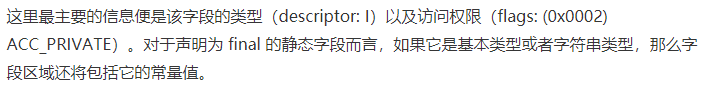
...

#30 = Utf8 java/lang/Object



一号常量池是一个指向Object类构造器的符号引用。它是由另外两个常量池项所构成。如果将它看成一个树结构，那么它的叶节点会是字符串常量。

* 字段区域，用来列举该类中的各个字段



private int tryBlock;

descriptor: I

flags: (0x0002) ACC\_PRIVATE

另外java虚机同样是还使用了“描述符”来描述字段的类型。

* 方法区域，用来列举该类中的各个方法

除了方法描述符以及访问权限外，每个方法还包括最为重要的区域

public void test();

descriptor: ()V

flags: (0x0001) ACC\_PUBLIC

Code:

stack=2, locals=3, args\_size=1

0: aload\_0

...

10: goto 35

...

34: athrow

35: aload\_0

...

40: return

Exception table:

from to target type

0 5 13 Class java/lang/Exception

0 5 27 any

13 19 27 any

LineNumberTable:

line 9: 0

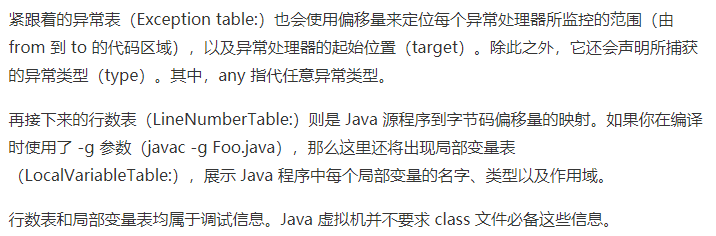
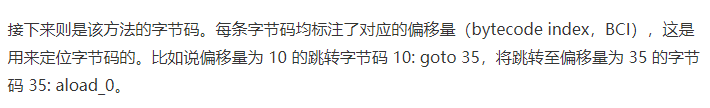
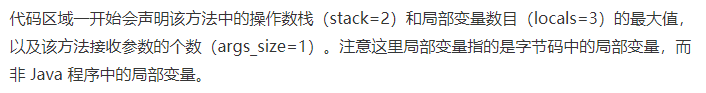
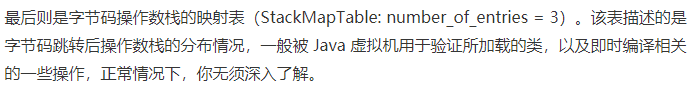
...

line 16: 40

StackMapTable: number\_of\_entries = 3

frame\_type = 77 /\* same\_locals\_1\_stack\_item \*/

stack = [ class java/lang/Exception ]

...  

### Javac –g Foo.java

如果在编译的时候采用javac –g Foo.java，那么javap –v Foo还将出现局部变量表，展示java程序中每个局部变量的名字、类型以及作用域。

行数和局部变量表均属于调试信息。Java虚机并不要求class文件必备这些信息。

LocalVariableTable:

Start Length Slot Name Signature

14 5 1 e Ljava/lang/Exception;

0 41 0 this LFoo;

### -C

如果只需要查阅方法对应的字节码，可以用-C代替-V

## OpenJDK项目Code Tools：实用小工具集

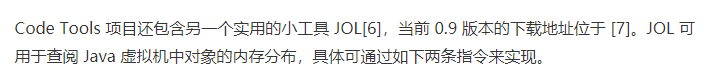
ASMTools的反汇编以及汇编操作所对应的命令分别为：

$ java -cp /path/to/asmtools.jar org.openjdk.asmtools.jdis.Main Foo.class > Foo.jasm

和

$ java -cp /path/to/asmtools.jar org.openjdk.asmtools.jasm.Main Foo.jasm

返汇编输出的格式和javap输出的不尽相同。一般只用它来进行一些简单的字节码修改。以此生成无法直接用java编译器生成的类。它在java的HotSpot虚机自身的测试中比较常见。



$ java -jar /path/to/jol-cli-0.9-full.jar internals java.util.HashMap

$ java -jar /path/to/jol-cli-0.9-full.jar estimates java.util.HashMap

## ASM：java字节码框架

是一个字节码分析以及修改的框架。广泛用于许多项目，例：Groovy、Kotlin的编译器，代码覆盖测试工具Cobertura、JaCoCo，以及各种通过字节码注入实现的程序行为监控工具。甚至java8的Lambda表达式的适配器类，也是接住ASM来动态生成的。

可以生成新的class文件，也可以修改已有的class文件

$ echo '

public class Foo {

public static void main(String[] args) {

boolean flag = true;

if (flag) System.out.println("Hello, Java!");

if (flag == true) System.out.println("Hello, JVM!");

}

}' > Foo.java

# 这里的 javac 我使用的是 Java 8 版本的。ASM 6.0 可能暂不支持新版本的 javac 编译出来的 class 文件

$ javac Foo.java

$ java -cp /PATH/TO/asm-all-6.0\_BETA.jar org.objectweb.asm.util.ASMifier Foo.class | tee FooDump.java

...

public class FooDump implements Opcodes {

public static byte[] dump () throws Exception {

ClassWriter cw = new ClassWriter(0);

FieldVisitor fv;

MethodVisitor mv;

AnnotationVisitor av0;

cw.visit(V1\_8, ACC\_PUBLIC + ACC\_SUPER, "Foo", null, "java/lang/Object", null);

...

{

mv = cw.visitMethod(ACC\_PUBLIC + ACC\_STATIC, "main", "([Ljava/lang/String;)V", null, null);

mv.visitCode();

mv.visitInsn(ICONST\_1);

mv.visitVarInsn(ISTORE, 1);

mv.visitVarInsn(ILOAD, 1);

...

mv.visitInsn(RETURN);

mv.visitMaxs(2, 2);

mv.visitEnd();

}

... 