> 工欲善其事，必先利其器。介绍学习JVM需要用到的工具，javap查看字节码、asmtools、ASM字节码框架等内容。

JVM工具篇

版本：2018/9/15-1(18:00)

javap(20)

1、javap的作用

> 1. 能够将class文件，反汇编成人类可读格式的工具

```
javac xxx.java
javap -p -v xxx.class
// 或者
javap -p -v xxx
```

2、javap默认会打印所有非私有的字段和方法

3、javap -p 会额外打印私有的字段和方法

4、javap -v 会打印所有额信息

5、如果只需要方法对应的字节码，可以用-c代替-v

6、-v选项打印出的信息包括哪些部分?

> 1. 基本信息
> 2. 常量池
> 3. 字段区域
> 4. 方法区域

基本信息

7、-v选项包含的基本信息

> 1. class 文件的版本号（minor version: 0，major version: 54）
> 2. 该类的访问权限（flags:(0x0021) ACC_PUBLIC, ACC_SUPER）
> 3. 该类的名字（this_class: #7）
> 4. 父类的名字（super_class: #8）
>   的名字
> 5. 所实现接口的数量（interfaces: 0）
> 6. 字段的数量（fields: 4）
> 7. 方法的数量（methods: 2）
> 8. 属性的数量（attributes: 1）

```
Classfile ../Foo.class
  Last modified ..; size 541 bytes
  MD5 checksum 3828cdfbba56fea1da6c8d94fd13b20d
  Compiled from "Foo.java"
public class Foo
  minor version: 0 //class文件版本号
  major version: 54
  flags: (0x0021) ACC_PUBLIC, ACC_SUPER //访问权限
  this_class:   #7 // Foo              //该类名字
  super_class:  #8 // java/lang/Object //父类名字
  interfaces: 0,  //所实现接口的数量
  fields: 4,      //字段的数量
  methods: 2,     //方法的数量
  attributes: 1   //属性的数量
```

8、attribute属性是什么?

> 1. class 文件所携带的辅助信息
> 2. 比如该 class 文件的源文件的名称。
> 3. 这类信息通常被用于 Java 虚拟机的验证和运行，以及 Java 程序的调试，一般无须深入了解。

9、class文件的版本号是什么?

> 1. 指的是编译生成该 class 文件时所用的 JRE 版本。
> 2. 由较新的 JRE 版本中的javac 编译而成的class文件，不能在旧版本的JRE上跑。
> 3. 否则，会出现如下异常信息。（Java8 对应的版本号为 52，Java 10 对应的版本号为 54。）

```
Exception in thread "main" java.lang.UnsupportedClassVersionError: xxx
```

10、新版本JRE中的javac编译的class文件，不能在旧版本的JRE上运行?

> 是的

11、类的访问权限

Flag Name	Value	Interpretation
ACC_PUBLIC	0x0001	public; 可以在包外访问。
ACC_FINAL	0x0010	final;不允许有子类
ACC_SUPER	0x0020	invokespecial指令调用时，特殊处理父类的方法。
ACC_INTERFACE	0x0200	是一个接口，而不是类。
ACC_ABSTRACT	0x0400	abstract; 一定不能被实例化。

Flag Name	Value	Interpretation
ACC_SYNTHETIC	0x1000	synthetic(合成的); 在源代码中没有出现。
ACC_ANNOTATION	0x2000	annotation类型.
ACC_ENUM	0x4000	enum类型.
ACC_MODULE	0x8000	是一个Module，不是接口也不是类。

常量池

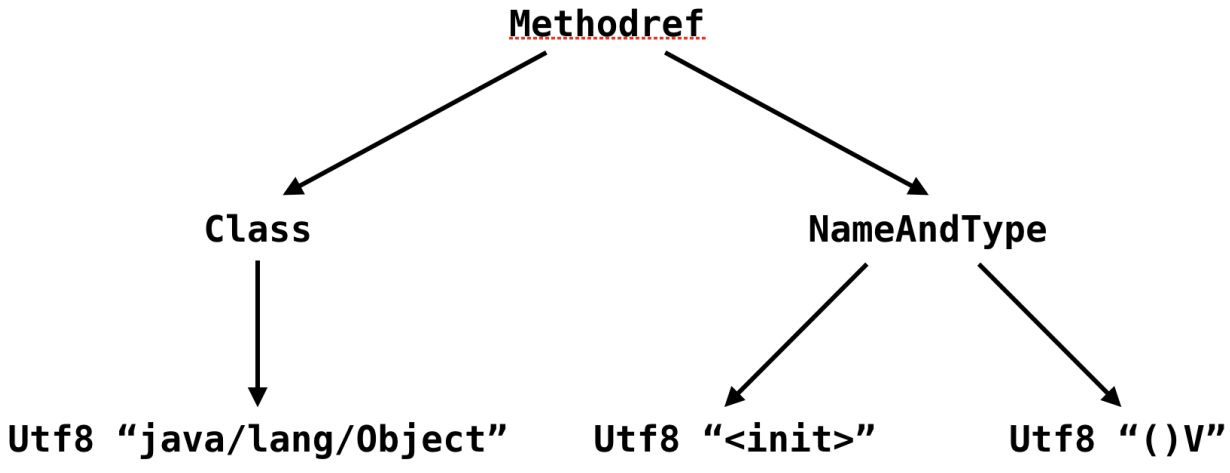
12、常量池用来干什么的?

> 1. 存放各种常量以及符号引用。
> 2. 每一项都有一个对应的索引 如`#1`
> 3. 每一项有可能引用其他常量池项。比如 `#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
```

13、 `#1 = Methodref #8.#23` 如何解释?

> 1. `#1` 常量池项，指向了一个Object类构造器的符号引用
> 2. `#8` 指向了一个Object类
> 3. `#23` 指向了构造器



字段区域

14、字段区域是做什么的?

> 用来列举该类中的各个字段

```
private int tryBlock;
private int catchBlock;
```

```
private int tryBlock;
  descriptor: I
  flags: (0x0002) ACC_PRIVATE
private int catchBlock;
  descriptor: I
  flags: (0x0002) ACC_PRIVATE
```

15、字段区域包含哪些部分?

> 1. 该字段的类型(descriptor: I)
> 2. 访问权限((0x0002) ACC_PRIVATE)
> 3. final声明的静态字段，如果是基本类型或者字符类型，还会包括该字段的常量值。

方法区域

16、方法区域的作用

> 1. 列举该类中的各个方法。

17、方法区域的组成部分(3)

> 1. 方法描述符
> 2. 访问权限
> 3. 最为重要的代码区域(`Code`)

```
public void test();
  descriptor: ()V
  flags: (0x0001) ACC_PUBLIC
  Code:
    xxx
```

18、代码区域的组成部分(8)

> 1. 声明操作数栈的最大值：stack=2
> 2. 声明局部变量数目的最大值: locals=3,这里是字节码中的局部变量，而不是Java程序中的局部变量。
> 3. 声明该方法接受的参数的个数: args_size=1

4. 方法的字节码：每个字节码还标注了偏移量(BCI, bytecode index),如 `10:goto 35` ， 偏移量为10.
5. 异常表。
6. 行数表: 就是Java源程序到字节码偏移量的映射。
7. 局部变量表：展示Java程序中每个局部变量的名字、类型、作用域
8. 字节码操作数栈的映射表：该表描述的是字节码跳转后操作数栈的分布情况。一般用于JVM去验证所加载的类和JIT相关的操作。

```
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
LocalVariableTable:
  Start Length Slot Name Signature
    14 5 1 e Ljava/lang/Exception;
    0 41 0 this LFoo;
StackMapTable: number_of_entries = 3
  frame_type = 77 /* same_locals_1_stack_item */
    stack = [ class java/lang/Exception ]
```

19、局部变量表，需要 `javac -g` 进行编译，才会出现。

20、行数表和局部变量表都属于调试信息

> JVM不要求class文件要必备这些信息

Code Tools(4)

1、OpenJDK的Code Tools项目包含了几个实用的小工具

1. ASMTools: 可以进行反汇编以及汇编操作
2. JOL: 查看JVM中对象的内存分布

asmtools

2、ASMTools的反汇编和汇编操作

> 反汇编

```
java -jar asmtools.jar jdis Test.class > Test.jasm
```

> 汇编

```
java -jar asmtools.jar jasm Test.jasm > Test.class
```

3、利用asmtools工具，可以来生成无法直接由Java编译器生成的类

1. 比如修改boolean在编译后生成的 `iconst_1` 为 `inconst_2`
2. 比如创建亮哥同名方法、参数相同、返回类型不同。

jol

4、jol的使用

```
$ java -jar jol-cli-0.9-full.jar internals java.util.HashMap
$ java -jar jol-cli-0.9-full.jar estimates java.util.HashMap
```

ASM(7)

1、ASM是什么?

1. 字节码分析及修改框架
2. 广泛用于 `Groovy`、`Kotlin` 编译器、代码覆盖测试工具Cobertura、JaCoCo，甚至Java8中Lambda表达式的适配器类，都是通过ASM动态生成。
3. 可以生成新的class文件
4. 也可以修改已有的class文件

2、ASMifier

1. ASM提供了一个辅助类:ASMifier
2. 能接收一个class文件并且输出一段生成该class文件原始字节数组的代码。

修改已有Class

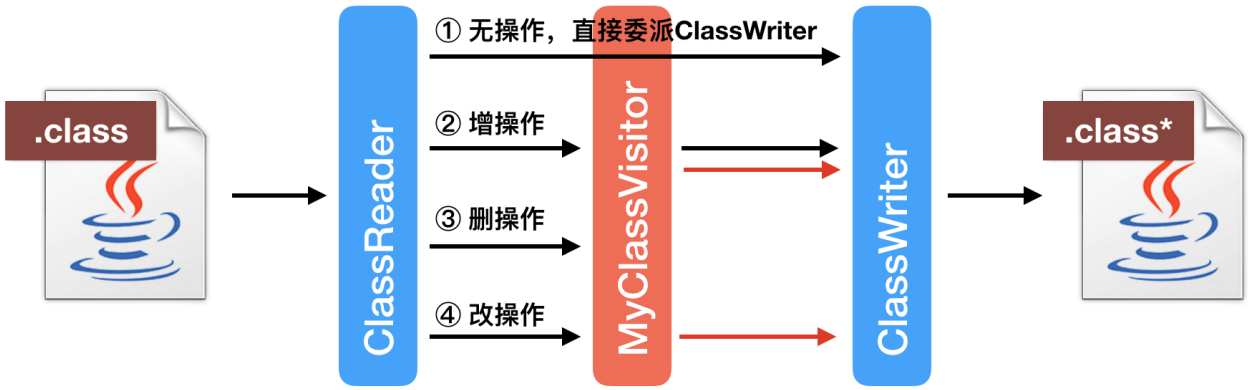
3、ASM修改已有class文件的简单实例

1. ClassReader读取Test类的原始字节，并且翻译成对应的访问请求。
2. ClassReader.accept: 将ClassReader的访问请求，交给一个访问者，并将该访问者，委派给ClassWriter。

```
public class Main {
    public static void main(String[] args) throws IOException {
      // 1、读取Test.java文件，并转换为ASM的数据结构
        ClassReader cr = new ClassReader("Test"); //已经有一个Test.class文件
        ClassWriter cw = new ClassWriter(ClassWriter.COMPUTE_FRAMES);
      // 2、将请求委派给ClassWriter
        cr.accept(cw, ClassReader.SKIP_FRAMES);
      // 3、转换为字节数组，然后写入到Test.class文件中
        Files.write(Paths.get("Test.class"), cw.toByteArray());
    }
}
```

4、如何利用进行增加、删除、修改操作?

1. 新增操作：在某一需要转发的请求后面，附带新的请求来实现。
2. 删除操作：可以通过不转发请求来实现。
3. 修改操作：可以通过忽略原请求，新建并发出另外的请求来实现。
4. 无操作：直接委派给ClassWriter



5、ASM修改已有class文件的实例

1-Student类：需要被修改的类

```
public class Student {
    public static void main(String[] args){
        boolean flag = true;
        if (flag) System.out.println("Hello, Java!");
        if (flag == true) System.out.println("Hello, JVM!");
    }
}
```

2-Main类：测试类，去加载Student.class到内存，在进行拦截修改后，生成新的class文件。

```
public class Main {
    public static void main(String[] args) throws IOException {
        // 1、Class的读取者。去加载Student的原始字节，并且翻译成访问请求。
        ClassReader cr = new ClassReader("Student");
        // 2、Class的写入者。
        ClassWriter cw = new ClassWriter(ClassWriter.COMPUTE_FRAMES);
        // 3、Reader和Writer的中间层，对访问操作进行拦截和处理。如果找到目标方法，就替换成自定义的M
        ClassVisitor cv = new ASMHelper.MyClassVisitor(ASM5, cw);
        cr.accept(cv, ClassReader.SKIP_FRAMES);
        // 4、将Write中的数据转为字节数组，写入到class文件中。
        Files.write(Paths.get("Student.class"), cw.toByteArray());

    }
}
```

> 3-ASMHelper类：辅助工具，对方法的请求，进行拦截和替换。

```
import jdk.internal.org.objectweb.asm.ClassVisitor;
import jdk.internal.org.objectweb.asm.MethodVisitor;
import jdk.internal.org.objectweb.asm.Opcodes;

public class ASMHelper implements Opcodes{
    /**==================================
     * 1、自定义的类访问者：MyClassVisitor
     *  1. visitMethod()获取到方法的访问请求，根据判断可以替换成自定义的MethodVisitor。
     *==========================================*/
    static class MyClassVisitor extends ClassVisitor {
        public MyClassVisitor(int api, ClassVisitor cv) {
            super(api, cv);
        }
        @Override
        public MethodVisitor visitMethod(int access, String name, String descriptor, String sig
            MethodVisitor visitor = super.visitMethod(access, name, descriptor, signature, exce
            // 将main()方法替换为自定义的MethodVisitor
            if ("main".equals(name)) {
                return new MyMethodVisitor(ASM5, visitor);
            }
            return visitor;
        }
    }
    /**==================================
     * 2、自定义的方法访问者
     *================================*/
    static class MyMethodVisitor extends MethodVisitor {
        private MethodVisitor mv;
        public MyMethodVisitor(int api, MethodVisitor mv) {
            super(api, null);
            this.mv = mv;
        }
        @Override
        public void visitCode() {
            mv.visitCode();
            mv.visitFieldInsn(GETSTATIC, "java/lang/System", "out", "Ljava/io/PrintStream;");
            mv.visitLdcInsn("Hello, World!");
            mv.visitMethodInsn(INVOKEVIRTUAL, "java/io/PrintStream", "println", "(Ljava/lang/St
            mv.visitInsn(RETURN);
            mv.visitMaxs(2, 1);
            mv.visitEnd();
        }
    }
}
```

> 总结：Student调用的 `main()` 方法，会调用中间层ClassVisitor的visitMethod()方法，然后被中间层替换为了自定义的方法。最终会执行 `Hello, World!`

6、ClassWriter的介绍

> 1. 继承自ClassVisitor

> 2. 将class信息读取至内存

7、ASM修改已有class文件的逻辑

> 1. 将目标class文件读取至内存中
> 2. 中间层ClassVisitor对方法调用请求进行拦截，然后替换为新的MethodVistor(替换为新方法)
> 3. 将修改后的内容写入到目标class文件中。

问题汇总

> 考考你，这不知道这些问题的大难。

1. javap的作用?
2. javap默认会打印所有非私有的字段和方法
3. javap -p 会额外打印私有的字段和方法
4. javap -v 会打印所有额信息
5. 如果只需要方法对应的字节码，可以用-c代替-v
6. -v选项打印出的信息包括哪些部分?
7. -v选项包含的基本信息有哪些?
8. attribute属性是什么?
9. class文件的版本号是什么?
10. 新版本JRE中的javac编译的class文件，不能在旧版本的JRE上运行?
11. 类的访问权限有哪些?
12. 常量池用来干什么的?
13. `#1 = Methodref #8.#23` 如何解释?
14. 字段区域是做什么的?
15. 字段区域包含哪些部分?
16. 方法区域的作用
17. 方法区域的组成部分(3)
18. 代码区域的组成部分(8)
19. 局部变量表需要使用javac的哪个配置项?
20. 行数表和局部变量表都属于调试信息
21. OpenJDK的Code Tools项目包含了几个实用的小工具
22. ASMTools的反汇编和汇编操作
23. asmtools工具有什么用?
24. jol如何使用?
25. ASM是什么?
26. ASMifier是什么?
27. ASM如何修改已有class文件?
28. 如何利用进行增加、删除、修改操作?
29. ClassWriter是什么？有什么用?
30. ASM修改已有class文件的总体逻辑是什么?

参考资料

1. 极客时间-JVM工具篇
2. JVM规范4.1小结-类的访问权限
3. ASTMTools的下载链接
4. asmtools的使用
5. JOL工具下载
6. ASM下载
7. ASM教学