14 - ASM之MethodWriter


舍是境界 关注
0.289 2022.01.24 07:08:03 字数 430 阅读 21

MethodWriter类的父类是MethodVisitor类。在ClassWriter类里，visitMethod()方法的实现就是通过MethodWriter类来实现的。

MethodWriter类

class info

MethodWriter类的父类是MethodVisitor类。需要注意的是，MethodWriter类并不带有public修饰，因此它的有效访问范围只局限于它所处的package当中，不能像其它的public类一样被外部所使用。

```
final class MethodWriter extends MethodVisitor {
}
```

fields

在MethodWriter类当中，定义了很多的字段。下面的几个字段，是与方法的访问标识、方法名和描述符等直接相关的字段：

```
final class MethodWriter extends MethodVisitor {
    private final int accessFlags;
    private final int nameIndex;
    private final String name;
    private final int descriptorIndex;
    private final String descriptor;
    private Attribute firstAttribute;
}
```

这些字段与ClassFile当中的method_info也是对应的：

```
method_info {
    u2             access_flags;
    u2             name_index;
    u2             descriptor_index;
    u2             attributes_count;
    attribute_info attributes[attributes_count];
}
```

下面的几个字段，是与“方法体”直接相关的几个字段：

```
final class MethodWriter extends MethodVisitor {
    private int maxStack;
    private int maxLocals;
    private final ByteVector code = new ByteVector();
    private Handler firstHandler;
    private Handler lastHandler;
    private final int numberOfExceptions;
    private final int[] exceptionIndexTable;
    private Attribute firstCodeAttribute;
}
```

这些字段对应于Code属性结构：

```
Code_attribute {
    u2 attribute_name_index;
    u4 attribute_length;
    u2 max_stack;
    u2 max_locals;
    u4 code_length;
    u1 code[code_length];
    u2 exception_table_length;
    {   u2 start_pc;
        u2 end_pc;
        u2 handler_pc;
        u2 catch_type;
    } exception_table[exception_table_length];
    u2 attributes_count;
    attribute_info attributes[attributes_count];
}
```

constructors

MethodWriter类定义的构造方法有哪些。

```
final class MethodWriter extends MethodVisitor {
    MethodWriter(SymbolTable symbolTable, int access, String name, String descriptor,
        super(Opcodes.ASM9);
        this.symbolTable = symbolTable;
        this.accessFlags = "<init>".equals(name) ? access | Constants.ACC_CONSTRUCTOR
        this.nameIndex = symbolTable.addConstantUtf8(name);
        this.name = name;
        this.descriptorIndex = symbolTable.addConstantUtf8(descriptor);
        this.descriptor = descriptor;
        this.signatureIndex = signature == null ? 0 : symbolTable.addConstantUtf8(sign
        if (exceptions != null && exceptions.length > 0) {
            numberOfExceptions = exceptions.length;
            this.exceptionIndexTable = new int[numberOfExceptions];
            for (int i = 0; i < numberOfExceptions; ++i) {
                this.exceptionIndexTable[i] = symbolTable.addConstantClass(exceptions[
            }
        } else {
            numberOfExceptions = 0;
            this.exceptionIndexTable = null;
        }
        this.compute = compute;
        if (compute != COMPUTE_NOTHING) {
            // Update maxLocals and currentLocals.
            int argumentsSize = Type.getArgumentsAndReturnSizes(descriptor) >> 2;
            if ((access & Opcodes.ACC_STATIC) != 0) {
                --argumentsSize;
            }
            maxLocals = argumentsSize;
            currentLocals = argumentsSize;
            // Create and visit the label for the first basic block.
            firstBasicBlock = new Label();
            visitLabel(firstBasicBlock);
        }
    }
}
```

methods

在MethodWriter类当中，也有两个重要的方法：computeMethodInfoSize()和putMethodInfo()方法。这两个方法也是在ClassWriter类的toByteArray()方法内使用到。

```
final class MethodWriter extends MethodVisitor {
    int computeMethodInfoSize() {
        // ......
        // 2 bytes each for access_flags, name_index, descriptor_index and attributes_
        int size = 8;
        // For ease of reference, we use here the same attribute order as in Section 4
        if (code.length > 0) {
            if (code.length > 65535) {
                throw new MethodTooLargeException(symbolTable.getClassName(), name, de
            }
            symbolTable.addConstantUtf8(Constants.CODE);
```

```
            // The Code attribute has 6 header bytes, plus 2, 2, 4 and 2 bytes respect
            // max_locals, code_length and attributes_count, plus the ByteCode and the
            size += 16 + code.length + Handler.getExceptionTableSize(firstHandler);
            if (stackMapTableEntries != null) {
                boolean useStackMapTable = symbolTable.getMajorVersion() >= Opcodes.V1
                symbolTable.addConstantUtf8(useStackMapTable ? Constants.STACK_MAP_TAB
                // 6 header bytes and 2 bytes for number_of_entries.
                size += 8 + stackMapTableEntries.length;
            }
            // ......
        }
        if (numberOfExceptions > 0) {
            symbolTable.addConstantUtf8(Constants.EXCEPTIONS);
            size += 8 + 2 * numberOfExceptions;
        }
        //......
        return size;
    }

    void putMethodInfo(final ByteVector output) {
        boolean useSyntheticAttribute = symbolTable.getMajorVersion() < Opcodes.V1_5;
        int mask = useSyntheticAttribute ? Opcodes.ACC_SYNTHETIC : 0;
        output.putShort(accessFlags & ~mask).putShort(nameIndex).putShort(descriptorIn
        // ......
        // For ease of reference, we use here the same attribute order as in Section 4
        int attributeCount = 0;
        if (code.length > 0) {
            ++attributeCount;
        }
        if (numberOfExceptions > 0) {
            ++attributeCount;
        }
        // ......
        // For ease of reference, we use here the same attribute order as in Section 4
        output.putShort(attributeCount);
        if (code.length > 0) {
            // 2, 2, 4 and 2 bytes respectively for max_stack, max_locals, code_length
            // attributes_count, plus the ByteCode and the exception table.
            int size = 10 + code.length + Handler.getExceptionTableSize(firstHandler);
            int codeAttributeCount = 0;
            if (stackMapTableEntries != null) {
                // 6 header bytes and 2 bytes for number_of_entries.
                size += 8 + stackMapTableEntries.length;
                ++codeAttributeCount;
            }
            // ......
            output
                .putShort(symbolTable.addConstantUtf8(Constants.CODE))
                .putInt(size)
                .putShort(maxStack)
                .putShort(maxLocals)
                .putInt(code.length)
                .putByteArray(code.data, 0, code.length);
            Handler.putExceptionTable(firstHandler, output);
            output.putShort(codeAttributeCount);
            // ......
        }
        if (numberOfExceptions > 0) {
            output
                .putShort(symbolTable.addConstantUtf8(Constants.EXCEPTIONS))
                .putInt(2 + 2 * numberOfExceptions)
                .putShort(numberOfExceptions);
            for (int exceptionIndex : exceptionIndexTable) {
              output.putShort(exceptionIndex);
            }
        }
        // ......
    }
}
```

MethodWriter类的使用

关于MethodWriter类的使用，它主要出现在ClassWriter类当中的visitMethod()和toByteArray()方法内。

visitMethod方法

在ClassWriter类当中，visitMethod()方法代码如下：

```
public class ClassWriter extends ClassVisitor {
    public final MethodVisitor visitMethod(int access, String name, String descriptor,
        MethodWriter methodWriter = new MethodWriter(symbolTable, access, name, descri
        if (firstMethod == null) {
            firstMethod = methodWriter;
        } else {
            lastMethod.mv = methodWriter;
        }
        return lastMethod = methodWriter;
    }
}
```

toByteArray方法

```
public class ClassWriter extends ClassVisitor {
    public byte[] toByteArray() {

        // First step: compute the size in bytes of the ClassFile structure.
        // The magic field uses 4 bytes, 10 mandatory fields (minor_version, major_ver
        // constant_pool_count, access_flags, this_class, super_class, interfaces_coun
        // methods_count and attributes_count) use 2 bytes each, and each interface us
        int size = 24 + 2 * interfaceCount;
        // ......
        int methodsCount = 0;
        MethodWriter methodWriter = firstMethod;
        while (methodWriter != null) {
            ++methodsCount;
            size += methodWriter.computeMethodInfoSize();        // 这里是对MethodWriter
            methodWriter = (MethodWriter) methodWriter.mv;
        }
        // ......

        // Second step: allocate a ByteVector of the correct size (in order to avoid a
        // dynamic resizes) and fill it with the ClassFile content.
        ByteVector result = new ByteVector(size);
        result.putInt(0xCAFEBABE).putInt(version);
        symbolTable.putConstantPool(result);
        int mask = (version & 0xFFFF) < Opcodes.V1_5 ? Opcodes.ACC_SYNTHETIC : 0;
        result.putShort(accessFlags & ~mask).putShort(thisClass).putShort(superClass);
        result.putShort(interfaceCount);
        for (int i = 0; i < interfaceCount; ++i) {
            result.putShort(interfaces[i]);
        }
        // ......
        result.putShort(methodsCount);
        boolean hasFrames = false;
        boolean hasAsmInstructions = false;
        methodWriter = firstMethod;
        while (methodWriter != null) {
            hasFrames |= methodWriter.hasFrames();
            hasAsmInstructions |= methodWriter.hasAsmInstructions();
            methodWriter.putMethodInfo(result);                    // 这里是对MethodWrit
            methodWriter = (MethodWriter) methodWriter.mv;
        }
        // ......

        // Third step: replace the ASM specific instructions, if any.
        if (hasAsmInstructions) {
            return replaceAsmInstructions(result.data, hasFrames);
        } else {
            return result.data;
        }
    }
}
```

总结

本文对MethodWriter进行了介绍：

1. 对于MethodWriter类的各个不同部分进行介绍，以便从整体上来理解MethodWriter类。
2. 关于MethodWriter类的使用，它主要出现在ClassWriter类当中的visitMethod()和toByteArray()方法内。
3. 从应用ASM的角度来说，只需要知道MethodWriter类的存在就可以了，不需要深究；从理解ASM源码的角度来说，MethodWriter类也是值得研究的。