12 - ASM之FieldWriter

舍是境界 关注
0.295 2022.01.22 06:49:41 字数 432 阅读 17

FieldWriter类继承自FieldVisitor类。在ClassWriter类里，visitField()方法的实现就是通过FieldWriter类来实现的。

FieldWriter类

class info

FieldWriter类的父类是FieldVisitor类。需要注意的是，FieldWriter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 FieldWriter extends FieldVisitor {
}
```

fields

在FieldWriter类当中，一些字段如下：

```
final class FieldWriter extends FieldVisitor {
    private final int accessFlags;
    private final int nameIndex;
    private final int descriptorIndex;
    private Attribute firstAttribute;
}
```

这些字段与ClassFile当中的field_info是对应的：

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

constructors

在FieldWriter类当中，只定义了一个构造方法；同时，它也不带有public标识，只能在package内使用。

```
final class FieldWriter extends FieldVisitor {
    FieldWriter(SymbolTable symbolTable, int access, String name, String descriptor, S
        super(Opcodes.ASM9);
        this.symbolTable = symbolTable;
        this.accessFlags = access;
        this.nameIndex = symbolTable.addConstantUtf8(name);
        this.descriptorIndex = symbolTable.addConstantUtf8(descriptor);
        if (signature != null) {
            this.signatureIndex = symbolTable.addConstantUtf8(signature);
        }
        if (constantValue != null) {
```

```
            this.constantValueIndex = symbolTable.addConstant(constantValue).index;
        }
    }
}
```

methods

在FieldWriter类当中，有两个重要的方法：computeFieldInfoSize()和putFieldInfo()方法。这两个方法会在ClassWriter类的toByteArray()方法内使用到。

```
final class FieldWriter extends FieldVisitor {
    int computeFieldInfoSize() {
        // The access_flags, name_index, descriptor_index and attributes_count fields
        int size = 8;
        // For ease of reference, we use here the same attribute order as in Section 4
        if (constantValueIndex != 0) {
            // ConstantValue attributes always use 8 bytes.
            symbolTable.addConstantUtf8(Constants.CONSTANT_VALUE);
            size += 8;
        }
        // ......
        return size;
    }

    void putFieldInfo(final ByteVector output) {
        boolean useSyntheticAttribute = symbolTable.getMajorVersion() < Opcodes.V1_5;
        // Put the access_flags, name_index and descriptor_index fields.
        int mask = useSyntheticAttribute ? Opcodes.ACC_SYNTHETIC : 0;
        output.putShort(accessFlags & ~mask).putShort(nameIndex).putShort(descriptorIn
        // Compute and put the attributes_count field.
        // For ease of reference, we use here the same attribute order as in Section 4
        int attributesCount = 0;
        if (constantValueIndex != 0) {
            ++attributesCount;
        }
        // ......
        output.putShort(attributesCount);
        // Put the field_info attributes.
        // For ease of reference, we use here the same attribute order as in Section 4
        if (constantValueIndex != 0) {
            output
              .putShort(symbolTable.addConstantUtf8(Constants.CONSTANT_VALUE))
              .putInt(2)
              .putShort(constantValueIndex);
        }
        // ......
    }
}
```

FieldWriter类的使用

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

visitField方法

```
public class ClassWriter extends ClassVisitor {
    public final FieldVisitor visitField(int access, String name, String descriptor, S
        FieldWriter fieldWriter = new FieldWriter(symbolTable, access, name, descripto
        if (firstField == null) {
            firstField = fieldWriter;
        } else {
            lastField.fv = fieldWriter;
        }
        return lastField = fieldWriter;
    }
}
```

toByteArray方法

在ClassWriter类当中，toByteArray()方法代码如下：

```
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 fieldsCount = 0;
        FieldWriter fieldWriter = firstField;
        while (fieldWriter != null) {
            ++fieldsCount;
            size += fieldWriter.computeFieldInfoSize();    // 这里是对FieldWriter.comput
            fieldWriter = (FieldWriter) fieldWriter.fv;
        }
        // ......


        // 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(fieldsCount);
        fieldWriter = firstField;
        while (fieldWriter != null) {
            fieldWriter.putFieldInfo(result);             // 这里是对FieldWriter.putFiel
            fieldWriter = (FieldWriter) fieldWriter.fv;
        }
        // ......

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

小结

本文主要对FieldWriter类进行介绍，内容总结如下：

- 对于FieldWriter类的各个不同部分进行介绍，以便从整体上来理解FieldWriter类。
- 关于FieldWriter类的使用，它主要出现在ClassWriter类当中的visitField()和toByteArray()方法内。
- 从ASM应用的角度来说，只需要知道FieldWriter类的存在就可以了，不需要深究，我们平常写ASM代码的时候，由于它不带有public标识，所以不会直接用到它；从理解ASM源码的角度来说，FieldWriter类则值得研究，可以重点关注一下computeFieldInfoSize()和putFieldInfo()这两个方法。