



热门故事

妻子去世半年，我再娶一个小十岁的女人有错吗？

代替公主和亲后，我成了敌国后宫“升职”最快的妃子

直播间打赏五十万，女主播主动私信我要见面

生完二胎，老公给我雇了一个“90后”小保姆

推荐阅读

fastjson SerializerFeature 序列化策略

阅读 110

Swift进阶-类与结构体

阅读 298

A站 的 Swift 实践

阅读 1,165

iOS开发 - 「Swift 学习」枚举类型

阅读 258

C++<第三十二篇>：转换构造函数和类型转换函数

阅读 63

07 - ASM之ClassWriter



舍是境界 200 关注

0.311 2022.01.16 07:47:12 字数 1,268 阅读 24

ClassWriter类

class info

ClassWriter的父类是ClassVisitor，因此ClassWriter类继承了visit()、visitField()、visitMethod()和visitEnd()等方法。

```
1 public class ClassWriter extends ClassVisitor {
2 }
```

fields

ClassWriter定义的字段有哪些。

```
1 public class ClassWriter extends ClassVisitor {
2     private int version;
3     private final SymbolTable symbolTable;
4
5     private int accessFlags;
6     private int thisClass;
7     private int superClass;
8     private int interfaceCount;
9     private int[] interfaces;
10
11     private FieldWriter firstField;
12     private FieldWriter lastField;
13
14     private MethodWriter firstMethod;
15     private MethodWriter lastMethod;
16
17     private Attribute firstAttribute;
18
19     //.....
20 }
```

这些字段与ClassFile结构密切相关：

```
1 ClassFile {
2     u4 magic;
3     u2 minor_version;
4     u2 major_version;
5     u2 constant_pool_count;
6     cp_info constant_pool[constant_pool_count-1];
7     u2 access_flags;
8     u2 this_class;
9     u2 super_class;
10    u2 interfaces_count;
11    u2 interfaces[interfaces_count];
12    u2 fields_count;
13    field_info fields[fields_count];
14    u2 methods_count;
15    method_info methods[methods_count];
16    u2 attributes_count;
17    attribute_info attributes[attributes_count];
18 }
```



constructors

ClassWriter定义的构造方法有两个，这里只关注其中一个，也就是只接收一个int类型参数的构造方法。在使用new关键字创建ClassWriter对象时，推荐使用COMPUTE_FRAMES参数。

```
1 public class ClassWriter extends ClassVisitor {
2     /* A flag to automatically compute the maximum stack size and the maximum number o
3     public static final int COMPUTE_MAXS = 1;
4     /* A flag to automatically compute the stack map frames of methods from scratch. *.
5     public static final int COMPUTE_FRAMES = 2;
6
7     // flags option can be used to modify the default behavior of this class.
8     // Must be zero or more of COMPUTE_MAXS and COMPUTE_FRAMES.
9     public ClassWriter(final int flags) {
10         this(null, flags);
11     }
12 }
```

methods

- visitXxx()方法

在ClassWriter这个类当中，我们仍然是只关注其中的visit()方法、visitField()方法、visitMethod()方法和visitEnd()方法。这些visitXxx()方法的调用，就是在为构建ClassFile提供“原材料”的过程。

```
1 public class ClassWriter extends ClassVisitor {
2     public void visit(
3         final int version,
4         final int access,
5         final String name,
6         final String signature,
7         final String superName,
8         final String[] interfaces);
9     public FieldVisitor visitField( // 访问字段
10         final int access,
11         final String name,
12         final String descriptor,
13         final String signature,
14         final Object value);
15     public MethodVisitor visitMethod( // 访问方法
16         final int access,
17         final String name,
18         final String descriptor,
19         final String signature,
20         final String[] exceptions);
21     public void visitEnd();
22     // .....
23 }
```

- toByteArray()方法

在ClassWriter类当中，提供了一个toByteArray()方法。这个方法的作用是将“所有的努力”（对visitXxx()的调用）转换成byte[]，而这些byte[]的内容就遵循ClassFile结构。

在toByteArray()方法的代码当中，通过三个步骤来得到byte[]：

- 第一步，计算size大小。这个size就是表示byte[]的最终的长度是多少。
- 第二步，将数据填充到byte[]当中。
- 第三步，将byte[]数据返回。

```
1 public class ClassWriter extends ClassVisitor {
2     public byte[] toByteArray() {
3
4         // First step: compute the size in bytes of the ClassFile structure.
5         // The magic field uses 4 bytes, 10 mandatory fields (minor_version, major_ver
6         // constant_pool_count, access_flags, this_class, super_class, interfaces_coun
7         // methods_count and attributes_count) use 2 bytes each, and each interface us
8         int size = 24 + 2 * interfaceCount;
```

热门故事

妻子去世半年，我再娶一个小十岁的女人有错吗？

代替公主和亲后，我成了敌国后宫“升职”最快的妃子

直播间打赏五十万，女主播主动私信我要见面

生完二胎，老公给我雇了一个“90后”小保姆

推荐阅读

fastjson SerializerFeature 序列化策略

阅读 110

Swift进阶-类与结构体

阅读 298

A 站的 Swift 实践

阅读 1,165

iOS开发 - 「Swift 学习」枚举类型

阅读 258

C++<第三十二篇>：转换构造函数和类型转换函数

阅读 63

```
9      int fieldsCount = 0;
10     FieldWriter fieldWriter = firstField;
11     while (fieldWriter != null) {
12         ++fieldsCount;
13         size += fieldWriter.computeFieldInfoSize();
14         fieldWriter = (FieldWriter) fieldWriter.fv;
15     }
16     int methodsCount = 0;
17     MethodWriter methodWriter = firstMethod;
18     while (methodWriter != null) {
19         ++methodsCount;
20         size += methodWriter.computeMethodInfoSize();
21         methodWriter = (MethodWriter) methodWriter.mv;
22     }
23
24     // For ease of reference, we use here the same attribute order as in Section 4
25     int attributesCount = 0;
26
27     // .....
28
29     if (firstAttribute != null) {
30         attributesCount += firstAttribute.getAttributeCount();
31         size += firstAttribute.computeAttributesSize(symbolTable);
32     }
33     // IMPORTANT: this must be the last part of the ClassFile size computation, be
34     // statements can add attribute names to the constant pool, thereby changing i
35     size += symbolTable.getConstantPoolLength();
36
37
38     // Second step: allocate a ByteVector of the correct size (in order to avoid a
39     // dynamic resizes) and fill it with the ClassFile content.
40     ByteVector result = new ByteVector(size);
41     result.putInt(0xCAFEBAFE).putInt(version);
42     symbolTable.putConstantPool(result);
43     int mask = (version & 0xFFFF) < Opcodes.V1_5 ? Opcodes.ACC_SYNTHETIC : 0;
44     result.putShort(accessFlags & ~mask).putShort(thisClass).putShort(superClass);
45     result.putShort(interfaceCount);
46     for (int i = 0; i < interfaceCount; ++i) {
47         result.putShort(interfaces[i]);
48     }
49     result.putShort(fieldsCount);
50     fieldWriter = firstField;
51     while (fieldWriter != null) {
52         fieldWriter.putFieldInfo(result);
53         fieldWriter = (FieldWriter) fieldWriter.fv;
54     }
55     result.putShort(methodsCount);
56     boolean hasFrames = false;
57     boolean hasAsmInstructions = false;
58     methodWriter = firstMethod;
59     while (methodWriter != null) {
60         hasFrames |= methodWriter.hasFrames();
61         hasAsmInstructions |= methodWriter.hasAsmInstructions();
62         methodWriter.putMethodInfo(result);
63         methodWriter = (MethodWriter) methodWriter.mv;
64     }
65     // For ease of reference, we use here the same attribute order as in Section 4
66     result.putShort(attributesCount);
67
68     // .....
69
70     if (firstAttribute != null) {
71         firstAttribute.putAttributes(symbolTable, result);
72     }
73
74     // Third step: replace the ASM specific instructions, if any.
75     if (hasAsmInstructions) {
76         return replaceAsmInstructions(result.data, hasFrames);
77     } else {
78         return result.data;
79     }
80 }
81 }
```

热门故事

妻子去世半年，我再娶一个小十岁的女人有错吗？

代替公主和亲后，我成了敌国后宫“升职”最快的妃子

直播间打赏五十万，女主播主动私信我要见面

生完二胎，老公给我雇了一个“90后”小保姆

推荐阅读

fastjson SerializerFeature 序列化策略

阅读 110

Swift进阶-类与结构体

阅读 298

A站的 Swift 实践

阅读 1,165

iOS开发 - 「Swift 学习」枚举类型

阅读 258

C++<第三十二篇>：转换构造函数和类型转换函数

阅读 63

创建ClassWriter对象

推荐使用COMPUTE_FRAMES

在创建ClassWriter对象的时候，要指定一个flags参数，它可以选择的值有三个：

- 第一个，可以选取的值是0。ASM不会自动计算max stacks和max locals，也不会自动计算stack map frames。
- 第二个，可以选取的值是ClassWriter.COMPUTE_MAXS。ASM会自动计算max stacks和max locals，但不会自动计算stack map frames。
- 第三个，可以选取的值是ClassWriter.COMPUTE_FRAMES（推荐使用）。ASM会自动计算max stacks和max locals，也会自动计算stack map frames。

```
1 | ClassWriter cw = new ClassWriter(ClassWriter.COMPUTE_FRAMES);
```

COMPUTE_FRAMES

在创建ClassWriter对象的时候，使用ClassWriter.COMPUTE_FRAMES，ASM会自动计算max stacks和max locals，也会自动计算stack map frames。

首先，来看一下max stacks和max locals。在ClassFile结构中，每一个方法都用method_info来表示，而方法里定义的代码则使用Code属性来表示，其结构如下：

```
1 | Code_attribute {
2 |     u2 attribute_name_index;
3 |     u4 attribute_length;
4 |     u2 max_stack;    // 这里是max stacks
5 |     u2 max_locals;   // 这里是max locals
6 |     u4 code_length;
7 |     u1 code[code_length];
8 |     u2 exception_table_length;
9 |     { u2 start_pc;
10 |      u2 end_pc;
11 |      u2 handler_pc;
12 |      u2 catch_type;
13 |     } exception_table[exception_table_length];
14 |     u2 attributes_count;
15 |     attribute_info attributes[attributes_count];
16 | }
```

如果我们在创建ClassWriter(flags)对象的时候，将flags参数设置为ClassWriter.COMPUTE_MAXS或ClassWriter.COMPUTE_FRAMES，那么ASM会自动帮助我们计算Code结构中max_stack和max_locals的值。

接着，来看一下stack map frames。在Code结构里，可能有多多个attributes，其中一个可能就是StackMapTable_attribute。StackMapTable_attribute结构，就是stack map frame具体存储格式，它的主要作用是对ByteCode进行类型检查。

```
1 | StackMapTable_attribute {
2 |     u2 attribute_name_index;
3 |     u4 attribute_length;
4 |     u2 number_of_entries;
5 |     stack_map_frame entries[number_of_entries];
6 | }
```

如果我们在创建ClassWriter(flags)对象的时候，将flags参数设置为ClassWriter.COMPUTE_FRAMES，那么ASM会自动帮助我们计算StackMapTable_attribute的内容。

热门故事

妻子去世半年，我再娶一个小十岁的女人有错吗？

代替公主和亲后，我成了敌国后宫“升职”最快的妃子

直播间打赏五十万，女主播主动私信我要见面

生完二胎，老公给我雇了一个“90后”小保姆

推荐阅读

fastjson SerializerFeature 序列化策略

阅读 110

Swift进阶-类与结构体

阅读 298

A站的 Swift 实践

阅读 1,165

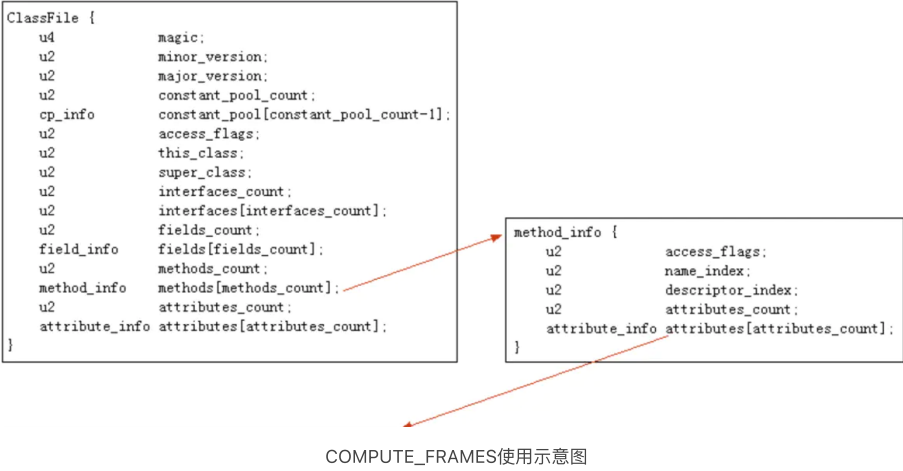
iOS开发 - 「Swift 学习」枚举类型

阅读 258

C++<第三十二篇>：转换构造函数和类型转换函数

阅读 63





我们推荐使用ClassWriter.COMPUTE_FRAMES。因为ClassWriter.COMPUTE_FRAMES这个选项，能够让ASM帮助我们自动计算max stacks、max locals和stack map frame的具体内容。

- 如果将flags参数的取值为0，那么我们就必须要提供正确的max stacks、max locals和stack map frame的值；
- 如果将flags参数的取值为ClassWriter.COMPUTE_MAXS，那么ASM会自动帮助我们计算max stacks和max locals，而我们则需要提供正确的stack map frame的值。

那么，ASM为什么会提供0和ClassWriter.COMPUTE_MAXS这两个选项呢？因为ASM在计算这些值的时候，要考虑各种各样不同的情况，所以它的算法相对来说就比较复杂，因而执行速度也会相对较慢。同时，ASM也鼓励开发者去研究更好的算法；如果开发者有更好的算法，就可以不去使用ClassWriter.COMPUTE_FRAMES，这样就能让程序的执行效率更高效。

但是，不得不说，要想计算max stacks、max locals和stack map frames，也不是一件容易的事情。出于方便的目的，就推荐大家使用ClassWriter.COMPUTE_FRAMES。在大多数情况下，ClassWriter.COMPUTE_FRAMES都能帮我们计算出正确的值。在少数情况下，ClassWriter.COMPUTE_FRAMES也可能会出错，比如说，有些代码经过混淆（obfuscate）处理，它里面的stack map frame会变更非常复杂，使用ClassWriter.COMPUTE_FRAMES就会出现错误的情况。针对这种少数的情况，我们可以在不改变原有stack map frame的情况下，使用ClassWriter.COMPUTE_MAXS，让ASM只帮助我们计算max stacks和max locals。

使用ClassWriter类

使用ClassWriter生成一个Class文件，可以大致分成三个步骤：

- 第一步，创建ClassWriter对象。
- 第二步，调用ClassWriter对象的visitXxx()方法。
- 第三步，调用ClassWriter对象的toByteArray()方法。

```
1 import org.objectweb.asm.ClassWriter;
2
3 import static org.objectweb.asm.Opcodes.*;
4
5 public class HelloWorldGenerateCore {
6     public static byte[] dump () throws Exception {
7         // (1) 创建ClassWriter对象
8         ClassWriter cw = new ClassWriter(ClassWriter.COMPUTE_FRAMES);
9
10        // (2) 调用visitXxx()方法
11        cw.visit();
12        cw.visitField();
13        cw.visitMethod();
14        cw.visitEnd(); // 注意，最后要调用visitEnd()方法
15    }
16 }
```

热门故事

妻子去世半年，我再娶一个小十岁的女人有错吗？

代替公主和亲后，我成了敌国后宫“升职”最快的妃子

直播间打赏五十万，女主播主动私信我要见面

生完二胎，老公给我雇了一个“90后”小保姆

推荐阅读

fastjson SerializerFeature 序列化策略

阅读 110

Swift进阶-类与结构体

阅读 298

A站的 Swift 实践

阅读 1,165

iOS开发 - 「Swift 学习」枚举类型

阅读 258

C++<第三十二篇>：转换构造函数和类型转换函数

阅读 63


```
16 | // (3) 调用toByteArray()方法
17 | byte[] bytes = cw.toByteArray();
18 | return bytes;
19 | }
20 | }
```


小结

本文对ClassWriter类进行了介绍，说明了该类的作用以及使用方式，希望对你能有所帮助

 1人点赞 >



 Java系列




更多精彩内容，就在简书APP



"小礼物走一走，来简书关注我"

赞赏支持

还没有人赞赏，支持一下



舍是境界

200

专注高性能，高可用，高扩展架构领域

总资产147 共写了77.2W字 获得408个赞 共16个粉丝

关注

签名文件一键自动生成




推荐阅读

更多精彩内容 >

JVM_ASM技术原理分析

在前面的文章中，我们分析了Class 这个字节码文件的格式，知道了字节码的作用，那么我们就可以直接生成字节码文件，...

 wo883721

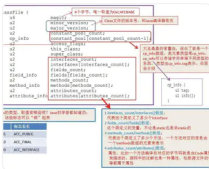
阅读 299 评论 0 赞 1

事件处理工具组件

event-spring-boot-starter是一个基于springboot starter机制，结合SPI ...

 javacoo

阅读 691 评论 2 赞 12



热门故事

妻子去世半年，我再娶一个小十岁的女人有错吗？

代替公主和亲后，我成了敌国后宫“升职”最快的妃子

直播间打赏五十万，女主播主动私信我要见面

生完二胎，老公给我雇了一个“90后”小保姆

推荐阅读

fastjson SerializerFeature 序列化策略

阅读 110

Swift进阶-类与结构体

阅读 298

A站的 Swift 实践

阅读 1,165

iOS开发 - 「Swift 学习」枚举类型

阅读 258

C++<第三十二篇>：转换构造函数和类型转换函数

阅读 63