JVM原理

# Class文件结构

大概如下图1-1所示

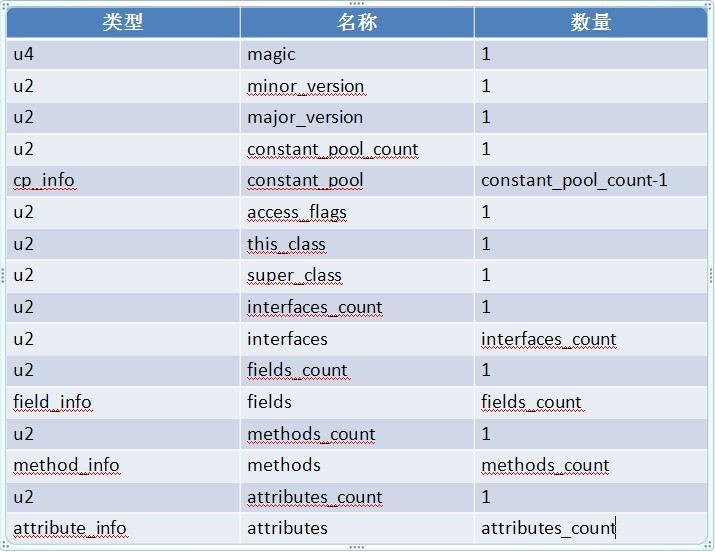


图1-1 class文件的信息结构

举例代码如下图1-2所示：

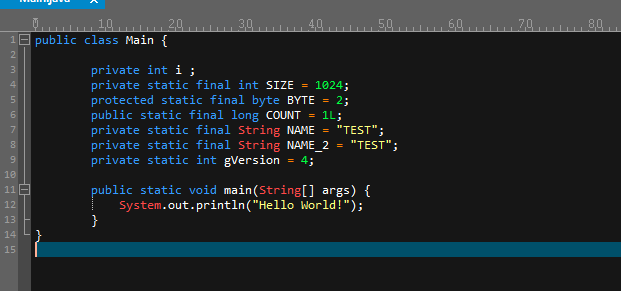


图1-2 举例代码

接下来我们一步一步来分析与查看对应的class文件信息。

## Magic信息

如图1-3 magic信息，如矩形的4个字节信息，由表1-1中，我们可以知晓magic的大小为u4，也就是4个字节。这也是class文件校验的非常重要的关键信息。内容为0xCAFEBABE，其实通俗翻译是café baby，咖啡，宝贝。而咖啡也是java的一个标志

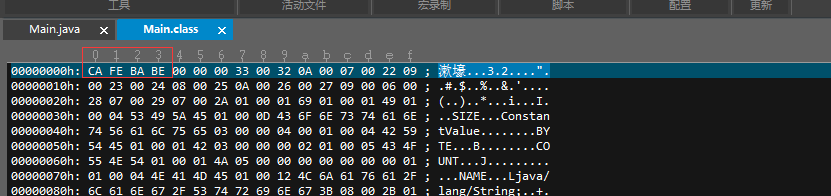


图1-3 magic信息

## 版本信息(minor、Major)

如图1-4 版本信息所示，矩形就是minor\_version的值ox0000，两个字节，类型为u2;圆形就是major\_version的值0x0033，两个字节，也就是51。也就是jdk1.7，看1-5的jdk版本对照图

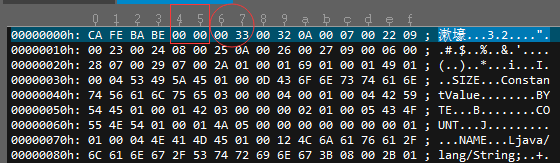


图1-4 版本信息

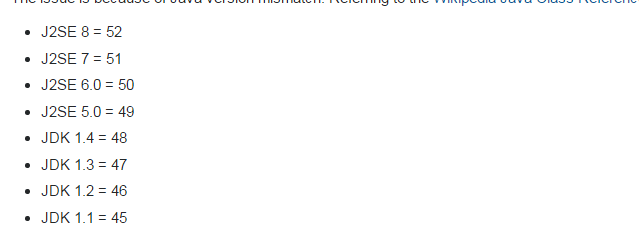


图1-5 jdk对照图

## 常量池(constant\_tool、constant\_tool\_count)

如图1-6 constant\_tool\_count常量池的个数大小，0x0032,2个字节，值为50，但是常量池是从1开始使用，将0号常量空出来，是了后续如果不使用常量值的时候使用。所以这里代表有常量49个。

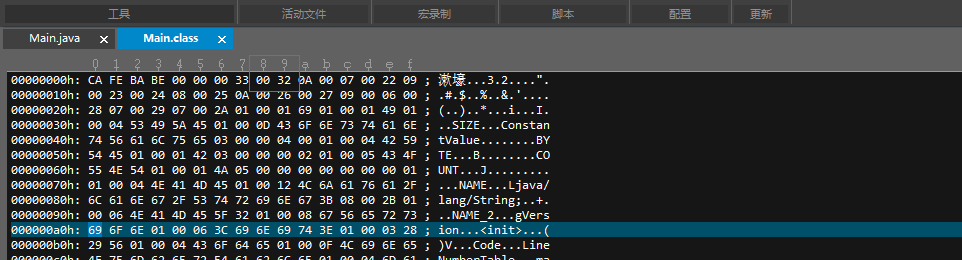


图1-6 constant\_tool\_count大小

### Constant\_tool

常量池：主要存放一些字面量(文本、定义为final的常量等)、符号引用(类和接口的全限定名、字段的名称和描述符、方法的名称和描述符)；

Java在javac之后，并没有像c、c++那样连接过程(生成函数的外部地址，也就是内存模型已经生成，然后供访问？java只有在运行时，被虚拟机加载类，才能确认函数的内存地址)保存了对应的实际调用地址。而是在运行时，根据符号引用，再解析具体的调用地址，从而实现多态。如表2-0 常量池的数据结构

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类型** | **简介** | **项目** | **类型** | **描述** |
| CONSTANT\_Utf8\_info | utf-8缩略编码字符串 | tag | u1 | 值为1 |
| length | u2 | utf-8缩略编码字符串占用字节数 |
| bytes | u1 | 长度为length的utf-8缩略编码字符串 |
| CONSTANT\_Integer\_info | 整形字面量 | tag | u1 | 值为3 |
| bytes | u4 | 按照高位在前储存的int值 |
| CONSTANT\_Float\_info | 浮点型字面量 | tag | u1 | 值为4 |
| bytes | u4 | 按照高位在前储存的float值 |
| CONSTANT\_Long\_info | 长整型字面量 | tag | u1 | 值为5 |
| bytes | u8 | 按照高位在前储存的long值 |
| CONSTANT\_Double\_info | 双精度浮点型字面量 | tag | u1 | 值为6 |
| bytes | u8 | 按照高位在前储存的double值 |
| CONSTANT\_Class\_info | 类或接口的符号引用 | tag | u1 | 值为7 |
| index | u2 | 指向全限定名常量项的索引 |
| CONSTANT\_String\_info | 字符串类型字面量 | tag | u1 | 值为8 |
| index | u2 | 指向字符串字面量的索引 |
| CONSTANT\_Fieldref\_info | 字段的符号引用 | tag | u1 | 值为9 |
| index | u2 | 指向声明字段的类或接口描述符CONSTANT\_Class\_info的索引项 |
| index | u2 | 指向字段描述符CONSTANT\_NameAndType\_info的索引项 |
| CONSTANT\_Methodref\_info | 类中方法的符号引用 | tag | u1 | 值为10 |
| index | u2 | 指向声明方法的类描述符CONSTANT\_Class\_info的索引项 |
| index | u2 | 指向名称及类型描述符CONSTANT\_NameAndType\_info的索引项 |
| CONSTANT\_InterfaceMethodref\_info | 接口中方法的符号引用 | tag | u1 | 值为11 |
| index | u2 | 指向声明方法的接口描述符CONSTANT\_Class\_info的索引项 |
| index | u2 | 指向名称及类型描述符CONSTANT\_NameAndType\_info的索引项 |
| CONSTANT\_NameAndType\_info | 字段或方法的部分符号引用 | tag | u1 | 值为12 |
| index | u2 | 指向该字段或方法名称常量项的索引 |
| index | u2 | 指向该字段或方法描述符常量项的索引 |

如图2-1 常量类型所示的值为0A，代表是Constant\_Methodref的类型(如表2-2 Constant\_MethodRef的数据结构)，也就是一个方法引用。

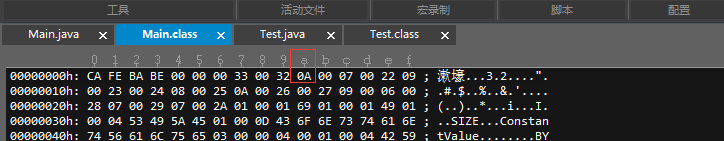
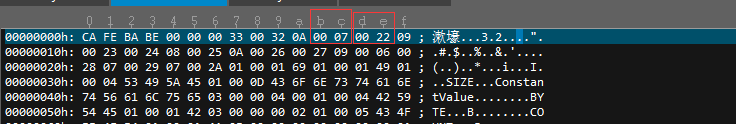


图2-1 常量类型

表 2-2 Constant\_Methodref数据结构

|  |  |  |
| --- | --- | --- |
| 类型 | 名称 | 说明 |
| u1 | tag | 10也就是上面所说的数据类型 |
| index | u2 | 指向声明方法的类描述Consant\_Class\_info的索引项 |
| index | u2 | 指向名称及类型描述符Constant\_NameAndType索引项 |

如图2-2 Constant\_Methodref值所示：第一个矩形0x0007表示第7个常量值的索引；第二个矩形0x0022表示第34个常量值的索引。如表2-3 常量池的内容解析，可以依次推算出常量表的内容



如图2-2 Constant\_Methodref值

表 2-3 常量表(16进制解析出来)

|  |  |  |  |
| --- | --- | --- | --- |
| 常量系数 | tag(类型，16进制) | 内容1(16进制) | 内容2(16进制) |
| 1. | 0A | 0007 | 0022 |
| 2. | 09 | 0023 | 0024 |
| 3. | 08 | 0025 |  |
| 4. | 0A | 0026 | 0027 |
| 5. | 09 | 0006 | 0028 |
| 6. | 07 | 0029 | |
| 7. | 07 | 002A | |
| 8. | 01 | 0001 | 69 |
| 9. | 01 | 0001 | 49 |
| 10. | 01 | 0004 | 53495A45 |
| 11. | 01 | 000D | 436F6E7474616E7456616C7565 |
| 12. | 03 | 00000400 | |
| 13. | 01 | 0004 | 42595445 |
| 14. | 01 | 0001 | 42 |
| 15. | 03 | 00000002 | |
| 16. | 01 | 0005 | 434F554E54 |
| 17. | 01 | 0001 | 4A |
| 18. | 05 | 000000000000000001 | |
| 20. | 01 | 0004 | 4E414D45 |
| 21. | 01 | 0012 | 4C6A6176612F6C616E672F537472696E673B |
| 22. | 08 | 002B | |
| 23. | 01 | 0006 | 4E414D455F32 |
| 24. | 01 | 0008 | 6756657273696F6E |
| 25. | 01 | 0006 | 3C696E69743E |
| 26. | 01 | 0003 | 282956 |
| 27. | 01 | 0004 | 436F6465 |
| 28. | 01 | 000F | 4C696E654E756D6265725461626C65 |
| 29. | 01 | 0004 | 6D61696E |
| 30. | 01 | 0016 | 285B4C6A6176612F6C616E672F537472696E673B2956 |
| 31. | 01 | 0008 | 3C636C696E69743E |
| 32. | 01 | 000A | 536F7572636546696C65 |
| 33. | 01 | 0009 | 4D61696E2E6A617661 |
| 34. | 0C | 0019 | 001A |
| 35. | 07 | 002C | |
| 36. | 0C | 002D | 002E |
| 37. | 01 | 000C | 48656C6C6F20576F726C6421 |
| 38. | 07 | 002F | |
| 39. | 0C | 0030 | 0031 |
| 40. | 0C | 0018 | 0009 |
| 41. | 01 | 0004 | 4D61696E |
| 42. | 01 | 0010 | 6A6176612F6C616E672F4F626A656374 |
| 43. | 01 | 0004 | 54455354 |
| 44. | 01 | 0010 | 6A6176612F6C616E672F53797374656D |
| 45. | 01 | 0003 | 6F7574 |
| 46. | 01 | 0015 | 4C6A6176612F696F2F5072696E7453747265616D3B |
| 47. | 01 | 0013 | 6A6176612F696F2F5072696E7453747265616D |
| 48. | 01 | 0007 | 7072696E746C6E |
| 49. | 01 | 0015 | 284C6A6176612F6C616E672F537472696E673B2956 |

## 访问标志

如图2-3 例子中的访问标志值，0x0021，两个字节，刚好是ACC\_PUBLIC|ACC\_SUPER的值。也就是刚好是public和可作

表2-4 访问标志

|  |  |  |
| --- | --- | --- |
| 标志名称 | 标志值 | 描述 |
| ACC\_PUBLIC | 0x0001 | 是否为public类型 |
| ACC\_FINAL | 0x0010 | 是否被声明为final，只有类可设置 |
| ACC\_SUPER | 0x0020 | 是否允许使用invokespecial字节码指令，JDK1.2以后编译出来的类这个标志为真 |
| ACC\_INTERFACE | 0x0200 | 标识这是一个接口 |
| ACC\_ABSTRACT | 0x0400 | 是否为abstract类型，对于接口和抽象类，此标志为真，其它类为假 |
| ACC\_SYNTHETIC | 0x1000 | 标识别这个类并非由用户代码产生 |
| ACC\_ANNOTATION | 0x2000 | 标识这是一个注解 |
| ACC\_ENUM | 0x4000 | 标识这是一个枚举 |

ACC\_SUPER就是保证能调用到正确的父类方法；举例：

A {…void a()}

B extends A {}

c extends B extends B {

void c() {

super.a();

}

}

如果是没有ACC\_SUPER的话，则会直接根据method\_ref里面的父类直接调用到A的。如果这时候修改了B，如下

B extends A {void a();}

如果不重新编译C的话，会导致C还是调用了A的a方法。明显这不是我们所要的结果。所以只有通过invokespecial方式，找到最近一个的父类方法调用



图2-3 类的访问标志

## 类索引、父类索引与接口索引

如图2-4 索引所示的值分别为：类索引(this的指向，0x0006，u2)、父类索引(super，0x0007，u2)、接口索引集合(个数，0x0000，u2表示没有实现任何接口，所以内容到此结束)

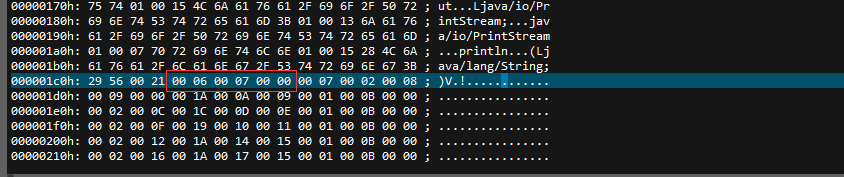


图2-4 索引

## 字段集合

如图2-5 字段内图所示：0x0007（u2，表示属性个数），根据表2-5 字段内容定义、表2-6字段访问类型定义。我们来看第一个属性的描述

access\_flags: 0x0002 私有

name\_index: 0x0008 第八个常量，由表2-3可以找到值为69(16进制)，对应的就是i

description\_index: 0x0009 第8个常量，由表2-3，值为49(16进制)，对应就是I，也就是类型为int。对照表2-7

attributes\_count: 0x0000表示没有。于是则没有对应attrbutes

表2-5 字段内容定义

|  |  |  |
| --- | --- | --- |
| 类型 | 字段名称 | 数量 |
| u2 | access\_flags(访问标志) | 1 |
| u2 | name\_index(名称索引，从常量表查找) | 1 |
| u2 | descriptor\_index(描述类型索引，常量表查找) | 1 |
| u2 | attributes\_count(额外描述数量，比如可能是static final这种类型，会被加上Constant Value描述) | 1 |
| attribute\_info | attributes(属性描述的值) | attributes\_coun |

表2-6 字段访问标识定义

|  |  |  |
| --- | --- | --- |
| 名称 | 值 | 描述 |
| ACC\_PUBLIC | 0x0001 | 字段是否为public |
| ACC\_PRIVATE | 0x0002 | 字段是否为private |
| ACC\_PROTECTED | 0x0004 | 字段是否为protected |
| ACC\_STATIC | 0x0008 | 字段是否为static |
| ACC\_FINAL | 0x0010 | 字段是否为final |
| ACC\_VOLATILE | 0x0040 | 字段是否为volatile |
| ACC\_TRANSIENT | 0x0080 | 字段是否为transient |
| ACC\_SYNTHETIC | 0x1000 | 字段是否为编译器自动产生 |
| ACC\_ENUM | 0x4000 | 字段是否为enum |

表2-7 描述符标志字符含义

|  |  |  |  |
| --- | --- | --- | --- |
| 标识字符 | 含义 | 标识字符 | 含义 |
| B | byte | J | long |
| C | char | S | short |
| D | double | Z | Boolean |
| F | float | V | void |
| I | int | L | 对象类型;如  Ljava/lang/Object |

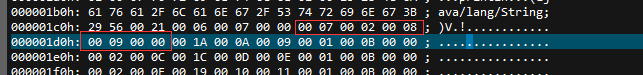


图2-5 字段个数与第一个字段内容(private int i)

第二个字段，如图2-6 第二个字段内容

access\_flags: 0x001A(0x0002|0x0008|0x0010)对照表2-6，知道是private static final

name\_index: 0x000A，对应SIZE变量

description\_index: 0x0009，对应I，也就是int

attributes\_count: 0x0001，有一个额外属性描述

attributes:

0x000B: 根据表2-3 可以查出为ConstantValue（祥看[ConstantValue](#_ConstantValue)）

0x00000002: 是固定值，表示长度，4个字节

0x000C：对应的值索引(根据表2-3)，查看可以得知是1024

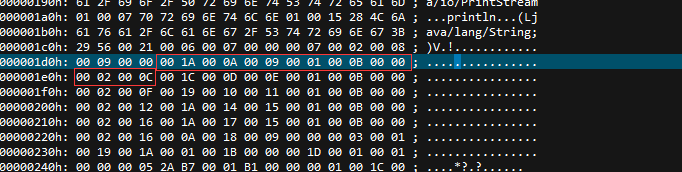


图2-6 第二个字段内容(private static final int SIZE = 1024)

## 方法集合

有点类似于字段集合的展示方式；首先是函数个数，然后每个函数的定义。如表2-8 方法访问标志，表2-9 方法表结构

表2-8 方法访问标志

|  |  |  |
| --- | --- | --- |
| 名称 | 值 | 描述 |
| ACC\_PUBLIC | 0x0001 | public |
| ACC\_PRIVATE | 0x0002 | private |
| ACC\_PROTECTED | 0x0004 | protected |
| ACC\_STATIC | 0x0008 | static |
| ACC\_FINAL | 0x0010 | final |
| ACC\_SYNCHRONIZED | 0x0020 | 同步方法 |
| ACC\_BRIDGE | 0x0040 | 编译器产生的桥接方法 |
| ACC\_VARARGS | 0x0080 | 方法是否接受不定参数 |
| ACC\_NATIVE | 0x1000 | native方法 |
| ACC\_ABSTRACT | 0x0400 | 抽象方法 |
| ACC\_STRICTFP | 0x0800 | strictfp方法(浮点数将按IEEE-754规范精度计算) |
| ACC\_SYNTHETIC | 0x1000 | 编译器自动产生 |

表2-9 方法表结构

|  |  |  |
| --- | --- | --- |
| 类型 | 字段名称 | 数量 |
| u2 | access\_flags(访问标志) | 1 |
| u2 | name\_index(名称索引，从常量表查找) | 1 |
| u2 | descriptor\_index(描述类型索引，常量表查找) | 1 |
| u2 | attributes\_count(额外描述数量，比如可能是static final这种类型，会被加上Constant Value描述) | 1 |
| attribute\_info | attributes(属性描述的值) | attributes\_coun |

如图2-7 方法个数(0x0003，表示3个方法)

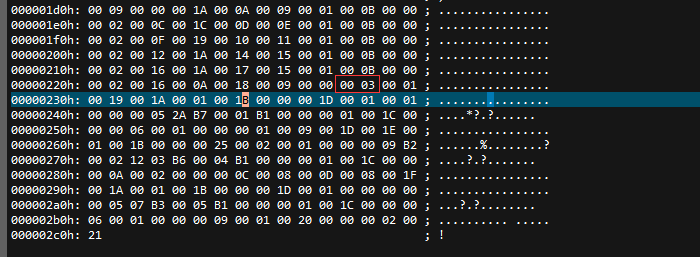


图2-7 方法个数

如图2-8 第一个方法(void <init>) 所示：

access\_flags: 0x0001，表示public

name\_index: 0x0019，找到表中对应的变量为<init>

description\_index: 0x001A，找到表中对应的变量为()V

attributes\_count: 0x0001，为1个

attribute\_info： 0x001B，找到表中对应为Code(祥看[Code](#_Code))，如图2-9 Code内容

attribute\_name\_index: 0x001B

attribute\_length: 0x0000001D

max\_stack: 0x0001

max\_locals:0x0001

code\_lenngth: 0x00000005，表示有5条指令

code: 2A、B7、00、01、B1可去对应查看“虚拟机字节码指令表”对照

exception\_table\_length: 0x0000，表示没有异常信息

exception\_table:空

attributes\_count: 0x0001，表示一个属性

attributes

attribute\_name\_index: 0x001C（对应找到第28变量LineNumberTable，详见[LineNumberTable](#_LineNumberTable)）

attribute\_length: 0x00000006，表长度为6

line\_number\_table\_length: 0x0001，1个

line\_number\_table

start\_pc: 0x0000 0行

line\_number: 0x0001 第1行

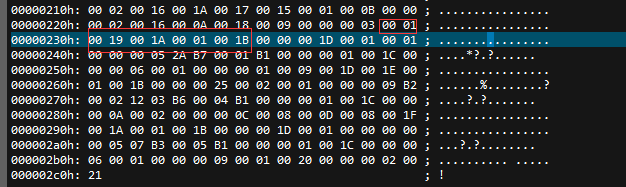


图2-8第一个方法(void <init>)

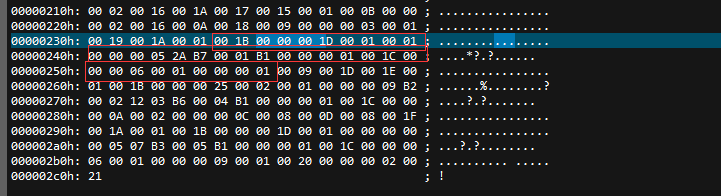


图2-9 第一个方法Code内容

如图3-0 main内容所示：

access\_flags: 0x0009(0x0001|0x0008)，public static

name\_index: 0x0001D，main

description\_index: 0x001E，([Ljava/lang/String;)V

attribute\_count: 0x0001 1个

attribute\_info:

attribute\_name\_index: 0x001B Code

attribute\_length: 0x00000025

max\_stacks: 0x0002，2

max\_locals: 0x0001，1

code\_length: 0x00000009,9个指令码

code: B2、00、02、12、03、B6、00、04、B1

exception\_table\_length: 0x0000，表示没有异常

exception\_info: 空

attributes\_count: 0x0001，1个

attributes:

attribute\_name\_index: 001C LineNumberTable

attribute\_length: 0x0000000A，10

line\_number\_table\_length: 0x0002，2个

line\_number\_tables:

|  |  |
| --- | --- |
| start\_pc | line\_number |
| 0x0000 | 0x000C |
| 0x0008 | 0x000D |

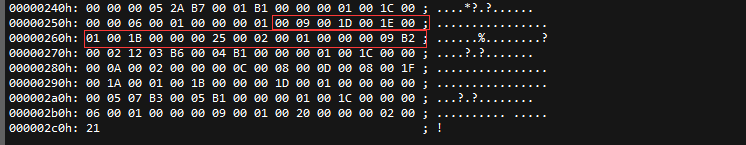


图3-0 main内容

## 类源文件描述

但是很多资料都没有讲出来。如图3-1 类文件描述所示：

sourcefile\_length: 0x0001 1个

sourcefile\_info(详见[SourceFile](#_SourceFile)):

attribute\_name\_index: 0x0020（32个常量，SourceFile）

attribute\_length: 0x00000002,2个字节

sourcefile\_index: 0x0021(33个常量，Main.java)

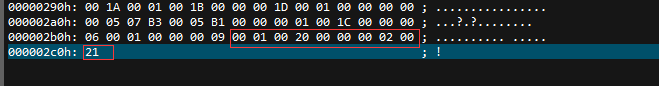


图3-1 类文件描述

## 属性描述

### Code

如表3-1 Code属性表的结构

表3-1 Code属性表的结构

|  |  |  |
| --- | --- | --- |
| 类型 | 名称 | 数量 |
| u2 | attribute\_name\_index | 1（固定指向Code） |
| u4 | attribute\_length | 1 |
| u2 | max\_stack | 1 |
| u2 | max\_locals | 1 |
| u4 | code\_length | 1 |
| u1 | code | code\_length |
| u2 | exception\_table\_length | 1 |
| exception\_info | exception\_table | exception\_table\_length |
| u2 | attributes\_count | 1 |
| attribute\_info | attributes | attributes\_count |

max\_stack: 操作数栈深度的最大值

max\_locals: 局部变量，以slot(4个字节一个)，但不是所有局部变量就开一个slot，可能多个变量共用一个。

code\_length与code来存储字节码

### LineNumberTable

用于描述java源码行与字节码行号。可指定-g:none和-g:lines来取消或者生成。如果没有生成，会导致无法调试，异常的时候不会抛出行数

表3-2 LineNumberTable描述

|  |  |  |
| --- | --- | --- |
| 类型 | 名称 | 数量 |
| u2 | attribute\_name\_index | 1（固定指向Code） |
| u4 | attribute\_length | 1 |
| u2 | line\_number\_table\_length | 1 |
| line\_number\_info | line\_number\_table | line\_number\_table\_length |

表3-3 line\_number\_info

|  |  |  |
| --- | --- | --- |
| 类型 | 名称 | 数量 |
| u2 | start\_pc(字节码行号) | 1 |
| u2 | line\_number(代码行号) | 1 |

### ConstantValue

表3-4 ConstantValue结构定义

|  |  |  |
| --- | --- | --- |
| 类型 | 名称 | 数量 |
| u2 | attribute\_name\_index | 1（固定指向Constant\_value） |
| u4 | attribute\_length | 1 |
| u2 | constant\_value\_index | 1 |

### SourceFile

表3-5 SourceFile结构定义

|  |  |  |
| --- | --- | --- |
| 类型 | 名称 | 数量 |
| u2 | attribute\_name\_index | 1（固定指向SourceFile） |
| u4 | attribute\_length | 1 |
| u2 | sourcefile\_index | 1 |