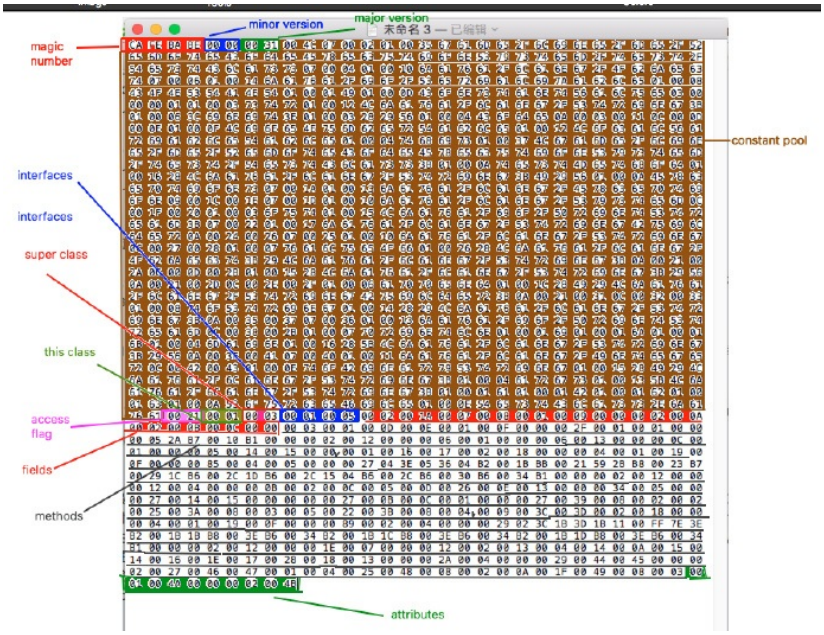
Java Class文件解析2（170@365）

2017-06-19 11:07:52

上回简单介绍了Class文件的结构，这回打算详细分析一下例子。 使用的测试文件TestClass： package game.line.me.RemoteCodeExecutionSystem.test;
import java.io.Serializable;
public class TestClass implements Serializable { private static final int CONSTANT = 1; private String str; public void testMethod(String str, int i) throws Exception{ int j = 1; int k = 2; System.out.println(str + i + j + k); } public static void main(String[] args) throws Exception { byte a = -1; int b = a; int c = a & 0xff; System.out.println(Integer.toBinaryString(a)); System.out.println(Integer.toBinaryString(b)); System.out.println(Integer.toBinaryString(c)); } }
Class文件结构：



Class文件的前八个字节分别是magic number和次版本号和主版本号，没什么好说的，紧接下来是常量池，常量池由于有多种结构，这里不打算一个一个的分析了，只简单举几个例子。 常量池：常量池的前两个字节是常量池的数目，这里是004c（0x004c，之后的数字都省略了0x，不过希望大家知道，这里面提到的Class文件里面的字节都是16进制的），十六进制数，4*16+12=76，由于位置0不存在常量，表示存在75个常量，从位置1开始。紧接下来的一个字节是07，代表tag=7，从常量池的11种结构表里面可以知道这个tag代表CONSTANT_Class_info，总共三个字节包含一个字节的tag和2个字节的index（指向全限定名常量项的索引），这里index = 0002 = 2，表示指向常量池里面的第二项的常量。那么，下面我们看看第二项常量是什么。tag=01，通过查询常量池的11种结构表可以知道，这代表CONSTANT_Utf8_info表，总共包含三项：tag，length（u2)和bytes（长度为length的UTF-8编码的字符串），length=0035=3*16+5=53， 接下来的53个字节转换成字符串就是该项的值“game/line/me/RemoteCodeExecutionSystem/test/TestClass”。就这样一项一项分析，就可以推出整个常量池的结构。

表 6-6　常量池中的 11 种数据类型的结构总表

常量	项目	类型	描述
CONSTANT_Utf8_info	tag	u1	值为 1
	length	u2	UTF-8 编码的字符串占用了字节数
	bytes	u1	长度为 length 的 UTF-8 编码的字符串
CONSTANT_Integer_info	tag	u1	值为 3
	bytes	u4	按照高位在前存储的 int 值
CONSTANT_Float_info	tag	u1	值为 4
	bytes	u4	按照高位在前存储的 float 值
CONSTANT_Long_info	tag	u1	值为 5
	bytes	u8	按照高位在前存储的 long 值
CONSTANT_Double_info	tag	u1	值为 6
	bytes	u8	按照高位在前存储的 double 值
CONSTANT_Class_info	tag	u1	值为 7
	index	u2	指向全限定名常量项的索引
CONSTANT_String_info	tag	u1	值为 8
	index	u2	指向字符串字面量的索引
CONSTANT_Fieldref_info	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 的索引项

（续）

常量	项目	类型	描述
CONSTANT_Methodref_info	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 的索引项
CONSTANT_InterfaceMethodref_info	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 的索引项
CONSTANT_NameAndType_info	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

常量池之后紧接着的是：访问标志（两个字节），用于识别一些类或者接口层次的访问信息。

表 6-7　访问标志

标志名称	标志值	含　义
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，JDK 1.2 之后编译出来的类的这个标志为真
ACC_INTERFACE	0x0200	标识这是一个接口
ACC_ABSTRACT	0x0400	是否为 abstract 类型，对于接口或抽象类来说，此标志值为真，其他类值为假
ACC_SYNTHETIC	0x1000	标识这个类并非由用户代码产生的
ACC_ANNOTATION	0x2000	标识这是一个注解
ACC_ENUM	0x4000	标识这是一个枚举

访问标志：这里的值是0021，其中0001表示ACC_PUBLIC，0020表示ACC_SUPER（见上表访问标志）（0021 = 0020 | 0001） 之后的是类索引（u2），父类索引（u2)和接口索引集合。 类索引：类索引是两个字节的索引，指向常量池里面的该类的全限定名，这里是0001，表示指向常量池的第一项，上面我们已经分析过了，常量池的第一项指向第二项，第二项里面的常数值就是类的全限定名。 父类索引：父类索引指向了该类的父类的全限定名，这里是0003，指向常量池第三项，由于我们没有分析到第三项，可以使用javap -verbose TestClass.class命令把整个Class文件解析出来看一下，对照一下，这里只贴一下常量池部分。从下面的截图可以看到#3指向了#4，#4的值是java/lang/Object，可知父类就是Object类了。

```
public class game.line.me.RemoteCodeExecutionSystem.test.TestClass implements java.io.Serializable
  minor version: 0
  major version: 49
  flags: ACC_PUBLIC, ACC_SUPER
Constant pool:
   #1 = Class              #2             // game/line/me/RemoteCodeExecutionSystem/test/TestClass
   #2 = Utf8               game/line/me/RemoteCodeExecutionSystem/test/TestClass
   #3 = Class              #4             // java/lang/Object
   #4 = Utf8               java/lang/Object
   #5 = Class              #6             // java/io/Serializable
   #6 = Utf8               java/io/Serializable
   #7 = Utf8               CONSTANT
   #8 = Utf8               I
   #9 = Utf8               ConstantValue
  #10 = Integer            1
  #11 = Utf8               str
  #12 = Utf8               Ljava/lang/String;
  #13 = Utf8               <init>
  #14 = Utf8               ()V
  #15 = Utf8               Code
  #16 = Methodref          #3.#17         // java/lang/Object."<init>":()V
  #17 = NameAndType        #13:#14        // "<init>":()V
  #18 = Utf8               LineNumberTable
  #19 = Utf8               LocalVariableTable
  #20 = Utf8               this
  #21 = Utf8               Lgame/line/me/RemoteCodeExecutionSystem/test/TestClass;
  #22 = Utf8               testMethod
  #23 = Utf8               (Ljava/lang/String;I)V
  #24 = Utf8               Exceptions
  #25 = Class              #26            // java/lang/Exception
  #26 = Utf8               java/lang/Exception
  #27 = Fieldref           #28.#30        // java/lang/System.out:Ljava/io/PrintStream;
  #28 = Class              #29            // java/lang/System
  #29 = Utf8               java/lang/System
  #30 = NameAndType        #31:#32        // out:Ljava/io/PrintStream;
  #31 = Utf8               out
  #32 = Utf8               Ljava/io/PrintStream;
  #33 = Class              #34            // java/lang/StringBuilder
  #34 = Utf8               java/lang/StringBuilder
  #35 = Methodref          #36.#38        // java/lang/String.valueOf:(Ljava/lang/Object;)Ljava/lang/String;
  #36 = Class              #37            // java/lang/String
  #37 = Utf8               java/lang/String
```

```
#38 = NameAndType        #39:#40        // valueOf:(Ljava/lang/Object;)Ljava/lang/String;
#39 = Utf8               valueOf
#40 = Utf8               (Ljava/lang/Object;)Ljava/lang/String;
#41 = Methodref          #33.#42        // java/lang/StringBuilder."<init>":(Ljava/lang/String;)V
#42 = NameAndType        #13:#43        // "<init>":(Ljava/lang/String;)V
#43 = Utf8               (Ljava/lang/String;)V
#44 = Methodref          #33.#45        // java/lang/StringBuilder.append:(I)Ljava/lang/StringBuilder;
#45 = NameAndType        #46:#47        // append:(I)Ljava/lang/StringBuilder;
#46 = Utf8               append
#47 = Utf8               (I)Ljava/lang/StringBuilder;
#48 = Methodref          #33.#49        // java/lang/StringBuilder.toString:()Ljava/lang/String;
#49 = NameAndType        #50:#51        // toString:()Ljava/lang/String;
#50 = Utf8               toString
#51 = Utf8               ()Ljava/lang/String;
#52 = Methodref          #53.#55        // java/io/PrintStream.println:(Ljava/lang/String;)V
#53 = Class              #54            // java/io/PrintStream
#54 = Utf8               java/io/PrintStream
#55 = NameAndType        #56:#43        // println:(Ljava/lang/String;)V
#56 = Utf8               println
#57 = Utf8               i
#58 = Utf8               j
#59 = Utf8               k
#60 = Utf8               main
#61 = Utf8               ([Ljava/lang/String;)V
#62 = Methodref          #63.#65        // java/lang/Integer.toBinaryString:(I)Ljava/lang/String;
#63 = Class              #64            // java/lang/Integer
#64 = Utf8               java/lang/Integer
#65 = NameAndType        #66:#67        // toBinaryString:(I)Ljava/lang/String;
#66 = Utf8               toBinaryString
#67 = Utf8               (I)Ljava/lang/String;
#68 = Utf8               args
#69 = Utf8               [Ljava/lang/String;
#70 = Utf8               a
#71 = Utf8               B
#72 = Utf8               b
#73 = Utf8               c
#74 = Utf8               SourceFile
#75 = Utf8               TestClass.java
```

接口索引集合：接口索引集合是一组u2类型数据集合，最前面两个字节表示长度length，之后的length个u2字节表示接口的索引，可以多个，这里length = 0001，表示就一个接口，接下来的就是接口索引0005，表示常量池第五项：#5指向#6，#6的值为java/io/Serializable
接下来是字段表集合： 字段表集合：这一部分主要存储了类里面的字段。开始的两个字节表示字段数目，这里是0002，表示2个字段，接下来分别是两个字段的字段表。每个字段表分别包括访问标志、名称索引、描述索引和属性表信息。

表 6-8　字段表结构

类　型	名　称	数　量
u2	access_flags	1
u2	name_index	1
u2	descriptor_index	1
u2	attributes_count	1
attribute_info	attributes	attributes_count

access_flags=001A，从下面的字段访问标志表里面可以看到字段的访问标志的设定，0010表示final，000A=0002+0008表示private static，name_index=0007，常量池的第7项，所以常量名为CONSTANT，descriptor_index=0008，常量池第8项I，表示类型为int，attributes_count=0001，表示属性表存在一个，开始的两个字节0009表示属性名称的索引，常量池第9项，ConstantValue属性，查看该属性的结构，可知接下来四个字节是属性长度（00000002），之后的两个字节是constantvalue_index=000A，常量池第10项1，综合起来：public static final int CONSTANT = 1；第二个字段以此类推。

表 6-9　字段访问标志

标志名称	标志值	含　义
ACC_PUBLIC	0x0001	字段是否 public
ACC_PRIVATE	0x0002	字段是否 pravate
ACC_PROTECTED	0x0004	字段是否 protected
ACC_STATIC	0x0008	字段是否 static
ACC_FINAL	0x0010	字段是否 final
ACC_VOLATILE	0x0040	字段是否 volatile
ACC_TRANSIENT	0x0080	字段是否 transient
ACC_SYNTHETIC	0x1000	字段是否由编译器自动产生的
ACC_ENUM	0x4000	字段是否 enum

字段表集合之后是方法表集合。 方法表集合：这个部分存储了类里面定义的方法。前两个字节0003表示了方法的数目，之后的字节分别是三个方法表，代表三个方法的详细信息。access_flags=0001，表示public，name_index=000D，常量池第13项，，这个方法我们的类里面并没有定义，是编译器添加的实例构造器，descriptor_index=000E，常量池第14项()V，表示方法的返回值为void，没有参数，attributes_count=0001，表示有一个属性，000F表示属性名称的索引，常量池第15项Code，表示里面存的是方法的代码。code属性的结构也挺复杂的，接下来四个字节0000002F表示属性长度，之后max_stack=0001， 操作栈的最大深度，max_locals=0001，局部变量表所需的存储空间，code_length=00000005，代码长度，表示接下来的5个字节存储的代码code=2A B7 00 10 B1这是五个字节码指令，具体的意思需要查询相应的指令表，这里就不详细介绍了，接下来是异常表，可以发现exception_table_length=0000，不存在异常表，接下来是属性表，attributes_count=0002，有两个属性，属性名称索引0012=16+2=18，常量池第18项LineNumberTable，对应java的源码行号和字节码的行号，接下来四个字节是属性长度=00 00 00 06， 然后是line_number_table_length=0001， 表示有一个line_number_info表，接下来的四个字节就是line_number_info表，00000005，前两个字节是字节码行号，后两个是java源码行号，可知对应关系0：5。接下来是下一个属性的name_index=00 13，常量池第19项，LocalVariableTable，简单来说这个属性就是存局部变量的名称的，不是必需的。它的结构和LineNumberTable挺像的，接下来四个字节是属性长度=00 00 00 0C，local_variable_table_length=00 01，表示有一个local_variable_info表，总共有五个部分，10个字节，start_pc和length定义了变量作用域，name_index和descriptor_index是变量的名称和描述符，index是这个变量在栈帧局部变量表中Slot的位置。 值=00 00 00 05 00 14 00 15 00 00， start_pc=0，length=5，name_index=0014=20，常量池第20项this，descriptor_index=0015=21，常量池第21项Lgame/line/me/RemoteCodeExecutionSystem/test/TestClass,表示TestClass 这个类的类型，index=0。一个方法就分析完了，以此类推可以分析剩下的两个方法。

表 6-11　方法表结构

类　型	名　称	数　量
u2	access_flags	1
u2	name_index	1
u2	descriptor_index	1
u2	attributes_count	1
attribute_info	attributes	attributes_count

表 6-12　方法访问标志

标志名称	标志值	含　义
ACC_PUBLIC	0x0001	方法是否为 public
ACC_PRIVATE	0x0002	方法是否为 private
ACC_PROTECTED	0x0004	方法是否为 protected
ACC_STATIC	0x0008	方法是否为 static
ACC_FINAL	0x0010	方法是否为 final
ACC_ SYNCHRONIZED	0x0020	方法是否为 synchronized
ACC_BRIDGE	0x0040	方法是否是由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	0x0100	方法是否为 native
ACC_ABSTRACT	0x0400	方法是否为 abstract
ACC_STRICT	0x0800	方法是否为 strictfp
ACC_SYNTHETIC	0x1000	方法是否是由编译器自动产生的

表 6-15　Code 属性表的结构

类　型	名　称	数　量
u2	attribute_name_index	1
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

方法表后面的最后一部分就是属性表集合了。 属性表集合：属性表的位置可以在方法表里面、字段表里面，也可以在类里面，甚至可以在其他属性里面，下面的表列举了一些属性类型。看我们的例子，属性部分一共10个字节00 01 00 4A 00 00 00 02 00 4B，前两个字节表示属性数目00 01， 表示有一个属性，00 4A 是属性名称索引，常量池第74项，SourceFile，然后四个字节是属性长度=00 00 00 02，接下来两个字节是sourcefile_index=00 4B，常量池第75项TestClass.java，表示源文件是TestClass.java。至此，Class文件就结束了。

表 6-13　虚拟机规范预定义的属性

属性名称	使用位置	含　义
Code	方法表	Java 代码编译成的字节码指令
ConstantValue	字段表	final 关键字定义的常量值
Deprecated	类、方法表、字段表	被声明为 deprecated 的方法和字段
Exceptions	方法表	方法抛出的异常
InnerClasses	类文件	内部类列表
LineNumberTable	Code 属性	Java 源码的行号与字节码指令的对应关系
LocalVariableTable	Code 属性	方法的局部变量描述
SourceFile	类文件	源文件名称
Synthetic	类、方法表、字段表	标识方法或字段为编译器自动生成的