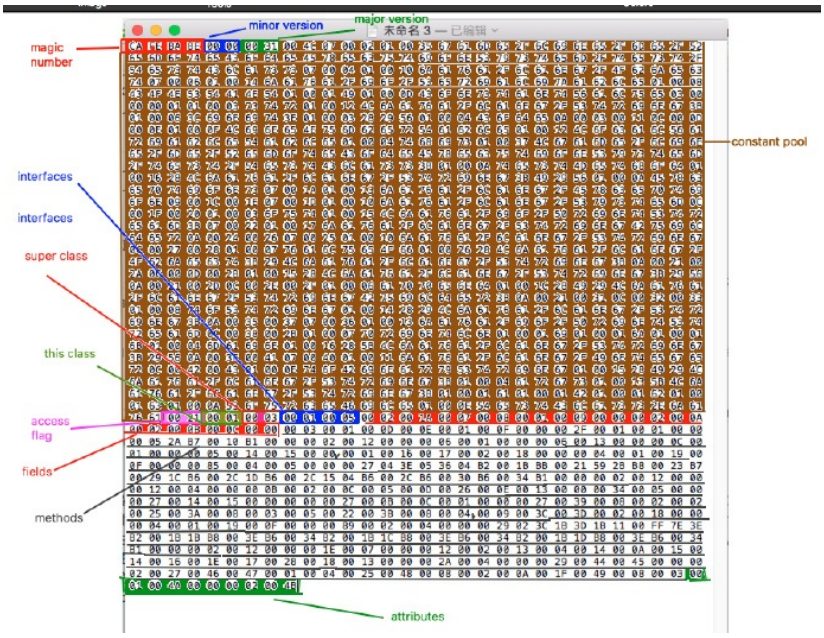
Java Class文件解析（167@365）

2017-06-16 15:42:28

最近，在学习java虚拟机相关的内容，研究了一下java的class文件的结构（学习教材：深入理解Java虚拟机），为了加深自己的理解，在这里记录一下。 Class文件是一组以8位字节为单位的二进制流，没有任何分隔符，对于8位字节以上空间的数据时，按照高位在前的方式分割成若干个8位字节存储。主要结构有两种：无符号数和表。无符号数（u1,u2,u4,u8（分别表示1，2，4，8个字节）），表由无符号数和其他表复合组成。

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)); } }
编译成Class文件之后，使用16进制文本编辑器打开的话，是一堆二进制数，经过分析大概如下：



使用javap -verbose TestClass.class 命令打开文件：

```
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
```

```
{
  public game.line.me.RemoteCodeExecutionSystem.test.TestClass();
    descriptor: ()V
    flags: ACC_PUBLIC
    Code:
      stack=1, locals=1, args_size=1
         0: aload_0
         1: invokespecial #16                 // Method java/lang/Object."<init>":()V
         4: return
      LineNumberTable:
        line 6: 0
      LocalVariableTable:
        Start  Length  Slot  Name   Signature
            0       5     0  this   Lgame/line/me/RemoteCodeExecutionSystem/test/TestClass;
```

```
public void testMethod(java.lang.String, int) throws java.lang.Exception;
  descriptor: (Ljava/lang/String;I)V
  flags: ACC_PUBLIC
  Exceptions:
    throws java.lang.Exception
  Code:
    stack=4, locals=5, args_size=3
       0: iconst_1
       1: istore_3
       2: iconst_2
       3: istore        4
       5: getstatic     #27                 // Field java/lang/System.out:Ljava/io/PrintStream;
       8: new           #33                 // class java/lang/StringBuilder
      11: dup
      12: aload_1
      13: invokestatic  #35                 // Method java/lang/String.valueOf:(Ljava/lang/Object;)Ljava/lang/String;
      16: invokespecial #41                 // Method java/lang/StringBuilder."<init>":(Ljava/lang/String;)V
      19: iload_2
      20: invokevirtual #44                 // Method java/lang/StringBuilder.append:(I)Ljava/lang/StringBuilder;
      23: iload_3
      24: invokevirtual #44                 // Method java/lang/StringBuilder.append:(I)Ljava/lang/StringBuilder;
      27: iload         4
      29: invokevirtual #44                 // Method java/lang/StringBuilder.append:(I)Ljava/lang/StringBuilder;
      32: invokevirtual #48                 // Method java/lang/StringBuilder.toString:()Ljava/lang/String;
      35: invokevirtual #52                 // Method java/io/PrintStream.println:(Ljava/lang/String;)V
      38: return
    LineNumberTable:
      line 11: 0
      line 12: 2
      line 13: 5
      line 14: 38
    LocalVariableTable:
      Start  Length  Slot  Name   Signature
          0      39     0  this   Lgame/line/me/RemoteCodeExecutionSystem/test/TestClass;
          0      39     1   str   Ljava/lang/String;
          0      39     2     i   I
          2      37     3     j   I
          5      34     4     k   I
```

```
  public static void main(java.lang.String[]) throws java.lang.Exception;
    descriptor: ([Ljava/lang/String;)V
    flags: ACC_PUBLIC, ACC_STATIC
    Exceptions:
      throws java.lang.Exception
    Code:
      stack=2, locals=4, args_size=1
         0: iconst_m1
         1: istore_1
         2: iload_1
         3: istore_2
         4: iload_1
         5: sipush        255
         8: iand
         9: istore_3
        10: getstatic     #27                 // Field java/lang/System.out:Ljava/io/PrintStream;
        13: iload_1
        14: invokestatic  #62                 // Method java/lang/Integer.toBinaryString:(I)Ljava/lang/String;
        17: invokevirtual #52                 // Method java/io/PrintStream.println:(Ljava/lang/String;)V
        20: getstatic     #27                 // Field java/lang/System.out:Ljava/io/PrintStream;
        23: iload_2
        24: invokestatic  #62                 // Method java/lang/Integer.toBinaryString:(I)Ljava/lang/String;
        27: invokevirtual #52                 // Method java/io/PrintStream.println:(Ljava/lang/String;)V
        30: getstatic     #27                 // Field java/lang/System.out:Ljava/io/PrintStream;
        33: iload_3
        34: invokestatic  #62                 // Method java/lang/Integer.toBinaryString:(I)Ljava/lang/String;
        37: invokevirtual #52                 // Method java/io/PrintStream.println:(Ljava/lang/String;)V
        40: return
      LineNumberTable:
        line 18: 0
        line 19: 2
        line 20: 4
        line 21: 10
        line 22: 20
        line 23: 30
        line 24: 40
      LocalVariableTable:
        Start  Length  Slot  Name   Signature
            0      41     0  args   [Ljava/lang/String;
            2      39     1     a   B
            4      37     2     b   I
           10      31     3     c   I
}
SourceFile: "TestClass.java"
```

Class文件的总体结构如下：（出自书籍：深入理解java虚拟机）（u2：两个字节，u4：四个字节）

表 6-1　Class 文件格式

类　型	名　称	数　量
u4	magic	1
u2	minor_version	1
u2	major_version	1
u2	constant_pool_count	1
cp_info	constant_pool	constant_pool_count−1
u2	access_flags	1
u2	this_class	1
u2	super_class	1
u2	interfaces_count	1
u2	interfaces	interfaces_count
u2	fields_count	1
field_info	fields	fields_count
u2	methods_count	1
method_info	methods	methods_count
u2	attributes_count	1
attribute_info	attributes	attributes_count

magic number：每个Class文件的头四个字节，用于确定这个文件是否为一个能被虚拟机接受的Class文件。 minor version：第5、6字节，次版本号 major version：第7、8字节，主版本号 constant pool：之后是常量池入口，包括一个两个字节的常量池数目（pool_count）和pool_count-1个具体的常量。每个常量都是一个表，共有十一种不同的结构，这是一个表的共同点是第一位都是一个u1的标志位（tag）：

表 6-3　常量池的项目类型

类　型	标志	描　述
CONSTANT_Utf8_info	1	UTF-8 编码的字符串
CONSTANT_Integer_info	3	整型字面量
CONSTANT_Float_info	4	浮点型字面量
CONSTANT_Long_info	5	长整型字面量
CONSTANT_Double_info	6	双精度浮点型字面量
CONSTANT_Class_info	7	类或接口的符号引用
CONSTANT_String_info	8	字符串类型字面量
CONSTANT_Fieldref_info	9	字段的符号引用
CONSTANT_Methodref_info	10	类中方法的符号引用
CONSTANT_InterfaceMethodref_info	11	接口中方法的符号引用
CONSTANT_NameAndType_info	12	字段或方法的部分符号引用

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

access flag:常量池之后的两个字节表示访问标志，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

this class：一个u2的索引，用于确定这个类的全限定名 super class：一个u2的索引，用于确定这个类的父类的全限定名 interfaces：一个u2的集合，包括一个u2的interface_count和interface_count-1个interface的索引 fields：包括类级变量和实例级变量，包括一个u2的fields_count和fields_count-1个field_info表

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

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

methods：与fields类似，包括一个u2的methods_count和methods_count-1个method_info表。

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

attributes:在Class文件、字段表和方法表中都可以包含自己的属性表信息。主要要以下几种属性：

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

每个属性的结构都不怎么一样，这里不具体介绍了，等详细分析实例的时候在具体介绍，由于篇幅过长，实例分析部分放到下一次进行。