昵称：晨曦语晴
园龄：5年5个月
粉丝：2
关注：0
+加关注

< 2019年6月 >

日	一	二	三	四	五	六
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	1	2	3	4	5	6

搜索

找找看

谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

Java(1)

随笔分类

ACM&ICPC(6)
C(2)
C#(1)
C++(5)
Clojure(1)
CSS
Emacs(3)
Java(8)
JavaScript(4)
JSP(1)
Linux(4)
Lisp(3)
OO技术(1)
Vim
WinOS底层&应用编程(8)
XHTML
编程思想(1)
操作系统(6)
函数式编程思想(3)
零碎知识&杂谈(3)
数据库(5)
算法(8)
网络编程(1)
网页前端(9)

随笔档案

2015年5月 (1)
2015年1月 (1)
2014年10月 (1)
2014年3月 (5)
2014年1月 (51)

阅读排行榜

1. JS创建table表格方法比较(4940)

java字节码(class)文件深度解析

1.Class文件基础

（1）文件格式

Class文件的结构不像XML等描述语言那样松散自由。由于它没有任何分隔符号，所以，以上数据项无论是顺序还是数量都是被严格限定的。哪个字节代表什么含义，长度是多少，先后顺序如何，都不允许改变。

（2）数据类型

仔细观察上面的Class文件格式，可以看出Class文件格式采用一种类似于C语言结构体的伪结构来存储，这种伪结构中只有两种数据类型：**无符号数和表。**
无符号数就是u1、u2、u4、u8来分别代表1个、2个、4个、8个字节。表是由多个无符号数或其他表构成的复合数据类型，以"_info"结尾。在表开始位置，通常会使用一个前置的容量计数器，因为表通常要描述数量不定的多个数据。

下图表示的就是Class文件格式中按顺序各个数据项的类型：

（3）兼容性

高版本的JDK能向下兼容以前版本的Class文件，但不能运行以后版本的Class文件，即使文件格式未发生任何变化。举例来说，JDK 1.7中的JRE能够执行JDK 1.5编译出的Class文件，但是JDK 1.7编译出来的Class文件不能被JDK 1.5使用。这就是target参数的用处，可以在使用JDK 1.7编译时指定-target 1.5。

2.一个简单的例子

[java] view plain copy print ?

```
1.  package com.cdai.jvm.bytecode;
2.
3.  public class ByteCodeSample {
4.
5.      private String msg = "hello world";
6.
7.      public void say() {
8.          System.out.println(msg);
9.      }
10.
11. }
```

```
package com.cdai.jvm.bytecode;

public class ByteCodeSample {

        private String msg = "hello world";

        public void say() {
                System.out.println(msg);
        }

}
```

编译成Class文件后的样子：

2. char* str = "abc" ;跟char str[] = "abc";的区别(2347)
3. HTML5 canvas保存与撤销接口(2272)
4. div/iframe自适应浏览器宽度高度问题(1734)
5. JavaScript世界的一等公民 - 函数(1483)

3.逐个字节分析

（1）魔数和版本号

在阅读这些十六进制码前需要声明，字节码是按大端存储的（不论在Intel还是在IBM上，这可能跟网络字节序用的也是为大端存储相关），具体的解析过程由JVM实现，前四个字节（u4）cafebabe就是Class文件的魔数，第5、6字节（u2）是Class文件的次版本号，第7、8字节（u2）是主版本号。十六进制0和32，也就是版本号为50.0，即JDK 1.6。之前介绍的target参数会影响这四个字节的值，从而使Class文件兼容不同的JDK版本。

（2）常量池

常量池是一个表结构，并且就像之前介绍过的，在表的内容前有一个u2类型的计数器，表示常量池的长度。十六进制23的十进制值为35，表示常量池里有下标为1~34的表项。下标从1开始而不是0，是因为第0个表项表示“不引用常量池中的任意一项”。每个表项的第一个字节是一个u1类型，表示12中数据类型。具体含义如下：

以第一项**07 00 02**为例，07表示该常量是个CONSTANT_Class_info类型，紧接着一个u2类型的索引执行第2项常量。再看第二项**01 00 24 63 6f 6d 2f ... 65**表示的就是字符串类型，长度为36（十六进制00 24），紧接着就是UTF-8编码的字符串"com/cdai/jvm/bytecode/ByteCodeSample"。很容易读懂吧？常量池主要是为后面的字段表和方法表服务的。

下面是通过javap解析后常量池的全貌（执行**javap -c -l -s -v ByteCodeSample，使用-p可以获取私有成员信息，具体查看-help**）

```
 Constant pool:
const #1 = class        #2;     //com/cdai/jvm/bytecode/ByteCodeSample
const #2 = Asciz        com/cdai/jvm/bytecode/ByteCodeSample;
const #3 = class        #4;     //java/lang/Object
const #4 = Asciz        java/lang/Object;
const #5 = Asciz        msg;
const #6 = Asciz        Ljava/lang/String;;
const #7 = Asciz        <init>;
const #8 = Asciz        ()V;
const #9 = Asciz        Code;
const #10 = Method      #3.#11; //java/lang/Object."<init>":()V
const #11 = NameAndType #7:#8;//"<init>":()V
const #12 = String      #13;    //  hello world
const #13 = Asciz       hello world;
const #14 = Field       #1.#15; // com/cdai/jvm/bytecode/ByteCodeSample.msg:Ljava/lang/String;
const #15 = NameAndType #5:#6;//  msg:Ljava/lang/String;
const #16 = Asciz       LineNumberTable;
const #17 = Asciz       LocalVariableTable;
const #18 = Asciz       this;
const #19 = Asciz       Lcom/cdai/jvm/bytecode/ByteCodeSample;;
const #20 = Asciz       say;
const #21 = Field       #22.#24;        //  java/lang/System.out:Ljava/io/PrintStream;
const #22 = class       #23;    //  java/lang/System
const #23 = Asciz       java/lang/System;
const #24 = NameAndType #25:#26;//  out:Ljava/io/PrintStream;
const #25 = Asciz       out;
const #26 = Asciz       Ljava/io/PrintStream;;
const #27 = Method      #28.#30;        //  java/io/PrintStream.println:(Ljava/lang/String;)V
const #28 = class       #29;    //  java/io/PrintStream
const #29 = Asciz       java/io/PrintStream;
const #30 = NameAndType #31:#32;//  println:(Ljava/lang/String;)V
const #31 = Asciz       println;
const #32 = Asciz       (Ljava/lang/String;)V;
const #33 = Asciz       SourceFile;
const #34 = Asciz       ByteCodeSample.java;
```

（3）访问标志

显然，00 21表示的就是公有的类。

（4）类、父类、接口

这三个u2类型的值分别表示类索引1、父类索引3、接口索引集合0。查看之前的常量池，第1项为"com/cdai/jvm/bytecode/ByteCodeSample"，第3项为"java/lang/Object"。第0项表示此类没有实现任何接口，这也就是常量池第0项的作用！

（5）字段表

00 01表示有1个字段。00 02是字段的访问标志，表示private权限的。00 05是字段的名称索引，指向常量池里第5项"msg"。00 06是字段的**描述符**索引，指向常量池里的第6项"Ljava/lang/String"。最后的00 00表示该字段没有其他**属性表**了。

描述符的作用就是用来描述字段的数据类型、方法的参数列表和返回值。而属性表就是为字段表和方法表提供额外信息的表结构。对于字段来说，此处如果将字段声明为一个static final msg = "aaa"的常量，则字段后就会跟着一个属性表，其中存在一项名为ConstantValue，指向常量池中的一个常量，值为的"aaa"。

属性表不像Class文件中的其他数据项那样具有严格的顺序、长度和内容，任何人实现的编译器都可以向属性表中写入自己定义的属性信息，JVM会忽略掉它不认识的属性。后面的方法表中还要用到属性表的Code属性，来保存方法的字节码。

（6）方法表

00 02表示有两个方法。00 01是方法的访问标志，表示公有方法。00 07和00 08与字段表中的名称和描述符索引相同，在这里分别表示"<init>"和"()V"。00 01表示该方法有属性表，属性名称为00 09即我们前面提到的Code属性。

要注意的是：Code属性表也可以有自己的属性，如后面的LocalVariableTable和LineNumberTable。它们分别为JVM提供方法的栈信息和调试信息。这是因为源码的行数在编译成字节码后，行数必然会增加为了在运行和调试过程（在JVM中进行）中能正确地显示出字节码所处在源码中的行数，因此需要各行进行映射，信息保留在这两个Table中

以下是javap解析后的结果：

```
public com.cdai.jvm.bytecode.ByteCodeSample();

Signature: ()V
  LineNumberTable:
   line 3: 0
   line 5: 4
   line 3: 10

  LocalVariableTable:
   Start  Length  Slot  Name
Signature
   0      11      0    this
Lcom/cdai/jvm/bytecode/ByteCodeSample;


  Code:
   Stack=2,
Locals=1, Args_size=1
   0:   aload_0
   1:   invokespecial   #10;
//Method java/lang/Object."<init>":()V
   4:   aload_0
   5:
ldc     #12; //String hello world
   7:   putfield        #14; //Field
msg:Ljava/lang/String;
   10:  return
  LineNumberTable:
   line 3:
0
   line 5: 4
   line 3: 10

  LocalVariableTable:
   Start
```

```
Length  Slot  Name   Signature
   0      11      0    this
Lcom/cdai/jvm/bytecode/ByteCodeSample;


public void say();

Signature: ()V
  LineNumberTable:
   line 8: 0
   line 9: 10


LocalVariableTable:
   Start  Length  Slot  Name   Signature
   0
11      0    this       Lcom/cdai/jvm/bytecode/ByteCodeSample;


Code:
   Stack=2, Locals=1, Args_size=1
   0:   getstatic       #21;
//Field java/lang/System.out:Ljava/io/PrintStream;
   3:   aload_0
   4:
getfield        #14; //Field msg:Ljava/lang/String;
   7:   invokevirtual
#27; //Method java/io/PrintStream.println:(Ljava/lang/String;)V
   10:
return
  LineNumberTable:
   line 8: 0
   line 9: 10


LocalVariableTable:
   Start  Length  Slot  Name   Signature
   0
11      0    this       Lcom/cdai/jvm/bytecode/ByteCodeSample;
```

4.小结

更深入的学习，参考：
http://www.blogjava.net/DLevin/archive/2011/09/05/358033.html 系列文集

分类: Java

标签: Java

晨曦语晴
关注 - 0
粉丝 - 2
+加关注

0 0

» 下一篇: CSS HACK

posted @ 2014-01-20 14:19 晨曦语晴 阅读(1257) 评论(0) 编辑 收藏