Java 类文件结构

一、概述

Java 语言中的各种变量,关键字和运算符号的语义最终都是由多条字节码命令组合而成的,因此字节码命令所能提供的语义描述能力肯定会比 Java 语言本身更加强大. 因此, 有一些 Java 语言本身无法有效支持的语言特性不代表字节码本身无法有效支持, 这也为其他语言实现一些有别于 Java 的语言特性提供了基础. 任何一个 class 文件都对应着唯一一个类或接口的定义信息,但反过来说,类或接口并一定都得定义在文件里(譬如类或接口也可以通过类加载器直接生成).所以下面描述的并不一定以磁盘文件的形式存在,而只是通俗地将任意一个有效的类或接口所应当满足的格式称为”Class 文件格式”

二、Class 类文件结构

Class 文件是一组以 8 位字节为基础单位的二进制流, 各个数据项目严格按照顺序紧凑地排列在 Class 文件之中, 中间没有添加任何分隔符, 这使得整个 Class 文件中存储的内容几乎全部是程序运行的必要数据, 没有空隙存在. 当遇到需要占用 8 位字节支上空间的数据项时,则会按照高位在前的方式分割成若干个 8 位字节进行存储. 根据 Java 虚拟机规范的规定, 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数据, 这种伪结构中只有两种数据类型: 无符号数和表, 后面的解析都要以这两种数据类型为基础, 所以先介绍这两个概念.

无符号数据属于基本的数据类型,以 u1, u2, u4, u8 来分别代表 1 个字节,2 个字节,4 个字节和 8 个字节的无符号数, 无符号数可以用来描述数字,索引引用,数量值或者按照 UTF-8 编码构成的字符串值.

表是由多个无符号数或者其他表作为数据项构成的复合数据类型,所有表都习惯性地以”_info”结尾. 表用于描述有层次关系的复合结构的数据, 整个 Class 文件本质上就是一张表, 它由下面所示的数据项

类型	名称	数量
U4	Magic	1
U2	Minor_version	1
U2	Major_version	1
C2	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

无论是无符号数还是表，当需要描述同一类型但数量不定的多个数据时,经常会使用一个前置的容量计数器加若干 个连续的数据项的形式，这时称这一系列连续的某一类型的数据为某一类型的集合.

Class 结构不像 XML 等描述语言,由于它没有任何分隔符号,所以在上表中的数据项,无论是顺序还是数量,甚至于数据存储的字节序(Byte Ordering，Class 文件中的字节序为 Big-endian)这样的细节,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. 下面我们来看看表中的各项的含义.

三、魔数与 Class 文件的版本：

每个 Class 文件的头 4 个字节称为魔数(Magic Number)，它唯一的作用是确定这个文件是否为一个能被虚拟机接受的 Class 文件. 很多文件存储标准中都使用魔数来进行身份识别，譬如图片格式 ,如 gif 或者 jpeg 等在文件头中都存有魔数. 使用魔数而不是扩展名来进行识别主要是基于安全方面的考虑，因为文件扩展名可以随意地必去. 文件格式的制定者可以自由地选择魔数值，只要这个魔数值还没有被广泛采用过同时又不会引起混淆即可. Class 文件的魔数的获得很有浪漫气息. 值为 0xCAFEBABE(咖啡宝贝?) 这个魔数值在 Java 还称做”Oak”语言的时候(大约是 1991 年前后)就已经确定下来了.

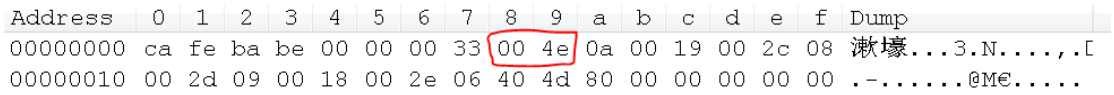
紧接着魔数的 4 个字节存储的是 Class 文件的版本号: 第 5 和第 6 个字节是次版本号(minor version)，第 7 和第 8 个字节是主版本号(Major Version). Java 的版本号是从 45 开始的，JDK1.1 之后的每个 JDK 大版本发布主版本号向上加 1(JDK1.0 ~_1.1 使用了 45.0 ~ 45.3 的版本号)，高版本的 JDK 能向下兼容以前版本的 Class 文件，但不能运行以后版本的 Class 文件,即使文件格式并未发生任何变化，虚拟机也必须拒绝执行超过其版本号的 Class 文件.

例如, JDK1.1 能支持的版本号为 45.0~45.65535 的 Class 文件. 无法执行版本号为 46.0 以上的 Class 文件,而 JDK1.2 则能支持 45.0-46.65535 的 Class 文件. 较新的 JDK 版本为 1.7，可生成的 Class 文件主版本号最大值为 51.0.

四、常量池：

紧接着主次版本号之后的是常量池入口，常量池可以理解为 Class 文件之中的资源仓库，它是 Class 文件结构中与其他项目关联最多的数据类型，也是占用 Class 文件空间最大的数据项目之一，同时它还是在 Class 文件中第一个出现的表类型数据项目.

由于常量池中常量的数据是不固定的，所以在常量池的入口需要放置一项 u2 类型的数据，代表常量池容量计数值. 与 Java 中语言习惯不一样的是，这个容量计数器是从 1 而不是从 0 开始的，如下图所示，常量池容量是 0x004E，值为 78，这就代表常量池中有 78 项常量，索引值范围为 1-78.



Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	ca	fe	ba	be	00	00	00	33	00	4e	0a	00	19	00	2c	08	漱壕...3.N....,.[
00000010	00	2d	09	00	18	00	2e	06	40	4d	80	00	00	00	00	00	.-.....@M€.....

常量池中主要存放两大类常量：字面量(Literal)和符号引用(Symbolic References)．字面量比较接近于 Java 语言层面的常概念，如文本字符串，声明为 final 的常量值等．而符号引用则属于编译原理方面的概念,包括了下面三类常量:

1) 类和接口的全限定名(Fully Qualified Name)
2) 字段的名称和描述符(Descriptor)
3) 方法的名称和描述符

Java 代码在进行 javac 编译的时候,并不像 C 和 C++那样有”连接”这一步骤，而是在虚拟机加载 Class 文件的时候进行动态连接．也就是说,在 Class 文件中不会保存各个方法,字段的最终内存布局信息，因此这些字段，方法的符号引用不经过运行期转换的话无法得到真正的内存入口地址，也就无法直接被虚拟机使用．当虚拟机运行时，需要从常量池获得对应的符号引,再在类创建时或运行时解析,翻译到具体的内存地址之中．

常量中的每一项数据都是一个表类型的结构,每个表类型结构的第 1 个字节是一个 u1 数据，用来标识常量的类型，而后面的部分则取决于它是什么类型,因为不同的类型的常量的结构是不同的．总共有十几种常量类型,所以对于它的解析是相对要耗时一些的．