[**深入理解JVM(七)——Class文件结构**](https://blog.csdn.net/u010425776/article/details/51245055)

<https://blog.csdn.net/u010425776/article/details/51245055>

## 什么是JVM的“无关性”？

Java具有平台无关性，也就是任何操作系统都能运行Java代码。之所以能实现这一点，是因为Java运行在虚拟机之上，不同的操作系统都拥有各自的Java虚拟机，因此Java能实现“一次编写，处处运行”。

而JVM不仅具有平台无关性，还具有语言无关性。   
平台无关性是指不同操作系统都有各自的JVM，而语言无关性是指Java虚拟机能运行除Java以外的代码！

这听起来非常惊人，但JVM对能运行的语言是有严格要求的。首先来了解下Java代码的运行过程。

Java源代码首先需要使用Javac编译器编译成class文件，然后启动JVM执行class文件，从而程序开始运行。   
也就是JVM只认识class文件，它并不管何种语言生成了class文件，只要class文件符合JVM的规范就能运行。   
因此目前已经有Scala、JRuby、Jython等语言能够在JVM上运行。它们有各自的语法规则，不过它们的编译器都能将各自的源码编译成符合JVM规范的class文件，从而能够借助JVM运行它们。

## 纵观Class文件结构

class文件是二进制文件，它的内容具有严格的规范，文件中没有任何空格，全是连续的0/1。class文件中的所有内容被分为两种类型：无符号数 和 表。   
- 无符号数   
它表示class文件中的值，这些值没有任何类型，但有不同的长度。根据这些值长度的不同分为：u1、u2、u4、u8，分别代表1字节的无符号数、2字节的无符号数、4字节的无符号数、8字节的无符号数。   
- 表   
class文件中所有数据(即无符号数)要么单独存在，要么由多个无符号数组成二维表。即class文件中的数据要么是单个值，要么是二维表。

### class文件的组织结构

1. 魔数
2. 本文件的版本信息
3. 常量池
4. 访问标志
5. 类索引
6. 父类索引
7. 接口索引集合
8. 字段表集合
9. 方法表集合

## Class文件的构成1：魔数

class文件的头4个字节称为魔数，用来表示这个class文件的类型。

魔数的作用就相当于文件后缀名，只不过后缀名容易被修改，不安全，因此在class文件中标示文件类型比较合适。

class文件的魔数是用16进制表示的“CAFEBABE”，非常具有浪漫主义色彩，谁说程序员的情商都很低！ 

## Class文件的构成2：版本信息

紧接着魔数的4个字节是版本号。它表示本class中使用的是哪个版本的JDK。

在高版本的JVM上能够运行低版本的class文件，但在低版本的JVM上无法运行高版本的class文件，即使该class文件中没有用到任何高版本JDK的特性也无法运行！ 

## Class文件的构成3：常量池

### 1. 什么是常量池？

紧接着版本号之后的就是常量池。常量池中存放两种类型的常量：

* 字面值常量   
  字面值常量即我们在程序中定义的字符串、被final修饰的值。
* 符号引用   
  符号引用就是我们定义的各种名字：   
  1. 类和接口的全限定名
  2. 字段的名字 和 描述符
  3. 方法的名字 和 描述符

### 2. 常量池的特点

* 常量池长度不固定   
  常量池的大小是不固定的，因此常量池开头放置一个u2类型的无符号数，用来存储当前常量池的容量。JVM根据这个值就知道常量池的头尾来。   
  **注：**这个值是从1开始的，若为5表示池中有4个常量。
* 常量池中的常量由而为表来表示   
  常量池开头有个常量池容量计数器，接下来就全是一个个常量了，只不过常量都是由一张张二维表构成，除了记录常量的值以外，还记录当前常量的相关信息。
* 常量池是class文件的资源仓库
* 常量池是与本class中其它部分关联最多的部分
* 常量池是class文件中空间占用最大的部分之一

### 3. 常量池中常量的类型

刚才介绍了，常量池中的常量大体上分为：字面值常量 和 符号引用。在此基础上，根据常量的数据类型不同，又可以被细分为14种常量类型。这14种常量类型都有各自的二维表示结构。每种常量类型的头1个字节都是tag，用于表示当前常量属于14种类型中的哪一个。

以CONSTANT\_Class\_info常量为例，它的二维表示结构如下：   
**CONSTANT\_Class\_info表：**

| **类型** | **名称** | **数量** |
| --- | --- | --- |
| u1 | tag | 1 |
| u2 | name\_index | 1 |

tag表示当前常量的类型(当前常量为CONSTANT\_Class\_info，因此tag的值应为7，表示一个类或接口的全限定名)；   
name\_index表示这个类或接口全限定名的位置。它的值表示指向常量池的第几个常量。它会指向一个CONSTANT\_Utf8\_info类型的常量，它的二维表结构如下：   
**CONSTANT\_Utf8\_info表：**

| **类型** | **名称** | **数量** |
| --- | --- | --- |
| u1 | tag | 1 |
| u2 | length | 1 |
| u1 | bytes | length |

CONSTANT\_Utf8\_info表示字符串常量；   
tag表示当前常量的类型，这里应该是1；   
length表示这个字符串的长度；   
bytes为这个字符串的内容(采用缩略的UTF8编码)

**问：为什么Java中定义的类、变量名字必须小于64K？**   
类、接口、变量等名字都属于符号引用，它们都存储在常量池中。而不管哪种符号引用，它们的名字都由CONSTANT\_Utf8\_info类型的常量表示，这种类型的常量使用u2存储字符串的长度。由于2字节最多能表示65535个数，因此这些名字的最大长度最多只能是64K。

**问：什么是UTF-8编码？什么是缩略UTF-8编码？**   
前者每个字符使用3个字节表示，而后者把128个ASKII码用1字节表示，某些字符用2字节表示，某些字符用3字节表示。 

## Class文件的构成4：访问标志

在常量池之后是2字节的访问标志。访问标志是用来表示这个class文件是类还是接口、是否被public修饰、是否被abstract修饰、是否被final修饰等。   
由于这些标志都由是/否表示，因此可以用0/1表示。   
访问标志为2字节，可以表示16位标志，但JVM目前只定义了8种，未定义的直接写0. 

## Class文件的构成5：类索引、父类索引、接口索引集合

类索引、父类索引、接口索引集合是用来表示当前class文件所表示类的名字、父类名字、接口们的名字。   
它们按照顺序依次排列，类索引和父类索引各自使用一个u2类型的无符号常量，这个常量指向CONSTANT\_Class\_info类型的常量，该常量的bytes字段记录了本类、父类的全限定名。   
由于一个类的接口可能有好多个，因此需要用一个集合来表示接口索引，它在类索引和父类索引之后。这个集合头两个字节表示接口索引集合的长度，接下来就是接口的名字索引。 

## Class文件的构成6：字段表的集合

### 1. 什么是字段表集合？

接下来是字段表的集合。字段表集合用于存储本类所涉及到的成员变量，包括实例变量和类变量，但不包括方法中的局部变量。   
每一个字段表只表示一个成员变量，本类中所有的成员变量构成了字段表集合。

### 2. 字段表结构的定义

| **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   
  字段的访问标志。在Java中，每个成员变量都有一系列的修饰符，和上述class文件的访问标志的作用一样，只不过成员变量的访问标志与类的访问标志稍有区别。
* name\_index   
  本字段名字的索引。指向一个CONSTANT\_Class\_info类型的常量，这里面存储了本字段的名字等信息。
* descriptor\_index   
  描述符。用于描述本字段在Java中的数据类型等信息(下面详细介绍)
* attributes\_count   
  属性表集合的长度。
* attributes   
  属性表集合。到descriptor\_index为止是字段表的固定信息，光有上述信息可能无法完整地描述一个字段，因此用属性表集合来存放额外的信息，比如一个字段的值。(下面会详细介绍)

### 3. 什么是描述符？

成员变量(包括静态成员变量和实例变量) 和 方法都有各自的描述符。   
对于字段而言，描述符用于描述字段的数据类型；   
对于方法而言，描述符用于描述字段的数据类型、参数列表、返回值。

在描述符中，基本数据类型用大写字母表示，对象类型用“L对象类型的全限定名”表示，数组用“[数组类型的全限定名”表示。   
描述方法时，将参数根据上述规则放在()中，()右侧按照上述方法放置返回值。而且，参数之间无需任何符号。

### 4. 字段表集合的注意点

1. 一个class文件的字段表集合中不能出现从父类/接口继承而来字段；
2. 一个class文件的字段表集合中可能会出现程序猿没有定义的字段   
   如编译器会自动地在内部类的class文件的字段表集合中添加外部类对象的成员变量，供内部类访问外部类。
3. Java中只要两个字段名字相同就无法通过编译。但在JVM规范中，允许两个字段的名字相同但描述符不同的情况，并且认为它们是两个不同的字段。

## Class文件的构成7：方法表的集合

在class文件中，所有的方法以二维表的形式存储，每张表来表示一个函数，一个类中的所有方法构成方法表的集合。   
方法表的结构和字段表的结构一致，只不过访问标志和属性表集合的可选项有所不同。

| **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 |

方法表的属性表集合中有一张Code属性表，用于存储当前方法经编译器编译过后的字节码指令。

### 方法表集合的注意点

1. 如果本class没有重写父类的方法，那么本class文件的方法表集合中是不会出现父类/父接口的方法表；
2. 本class的方法表集合可能出现程序猿没有定义的方法   
   编译器在编译时会在class文件的方法表集合中加入类构造器和实例构造器。
3. 重载一个方法需要有相同的简单名称和不同的特征签名。JVM的特征签名和Java的特征签名有所不同：   
   * Java特征签名：方法参数在常量池中的字段符号引用的集合
   * JVM特征签名：方法参数＋返回值

## Class文件的构成8：属性表的集合

