代码编译的结果从本地机器码转变为字节码，是存储格式发展的一小步，却是编程语言发展的一大步。

# 1 概述

记得在第一节计算机程序课上我的老师就讲过：“计算机只认识0和1，所以我们写的程序需要经编译器翻译成0和1构成的二进制格式才能由计算机执行”。10多年时间过去了，今天的计算机仍然只能识别0和1，但由于最近10年内虚拟机以及大量建立在虚拟机之上的程序语言如雨后春笋般出现并蓬勃发展，阿景我们编写的程序编译成二进制本地机器码（Native Code）已不再是唯一的选择，越来越多的程序语言选择了与操作系统和机器指令集无关的、平台中立的格式作为程序编译后的存储格式。

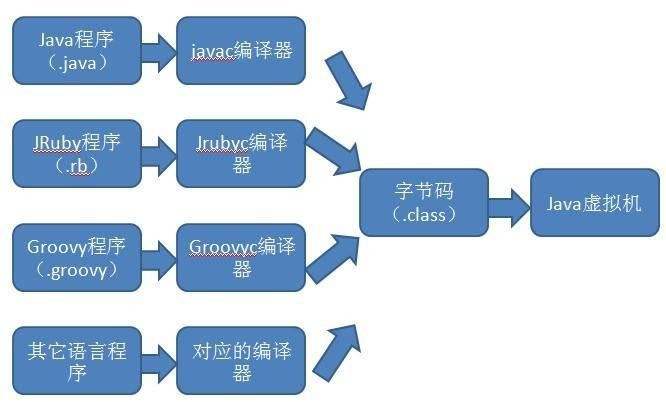
# 2 无关性基石

如果计算机的CPU指令集只有x86一种，操作系统也只有Windows一种，那也许Java语言就不会出现。Java在刚刚诞生之时曾经提出过一个非常著名的宣传口号：“一次编写，到处运行（Write Once，Run Anywhere）”，这句话充分表达了软件开发人员对冲破平台界限的渴望。在无时无刻不充满竞争的IT领域。不可能只有Wintel（微软公司的Windows与Intel公司的芯片相结合，曾经是业界最强大的联盟）存在，我们也不希望只有Wintel存在（寡头垄断市场对于消费者是极大地伤害），各种不同的硬件体系结构和不同的操作系统肯定会长期并存发展。“与平台无关”的理想最终实现在操作系统的应用层上（这里跨的平台很明显是OS，Java编译后的中间码可以跨平台，但是运行中间码的JVM却不是跨平台的，跨平台也是通过不同平台下的JVM将其解释为特定平台下的机器码实现的）：Sun公司以及其他虚拟机提供商发布了许多可以运行在各种不同平台上的虚拟机，这些虚拟机都可以载入和执行同一种平台无关的字节码，从而实现了程序的“一次编写，到处运行”。

各种不同平台的虚拟机与所有平台都统一使用的程序存储格式----字节码（ByteCode）是构成平台无关性的基石，但本节标题中刻意省略了“平台”二字，那是因为笔者注意到虚拟机的另外一种中立特性-----语言无关性正越来越被开发者所重视。到目前为止，或许大部分程序员都还认为Java虚拟机执行Java程序是一件理所当然和天经地义的事情。但在Java发展之初，设计者就曾经考虑过并实现了让其他语言运行在Java虚拟机之上的可能性，它们在发布规范文档的时候，也刻意把Java的规范拆分成了Java语言规范《The Java Language Specification》及Java虚拟机规范《The Java Virtual Machine Specification》，并且在1997年发布的第一版Java虚拟机规范中就曾经承诺过：“In the future，we will consider bounded extensions to the Java virtual machine to provide better support for other languages”(在未来，我们会对Java虚拟机进行适当的扩展，以便更好的支持其他语言运行于JVM之上)，当Java虚拟机发展到JDK1.7~1.8的时候，JVM设计者通过JSR-292基本兑现了这个承诺。

时至今日，商业机构和开源机构已经在Java语言之外发展出一大批在Java虚拟机之上运行的语言，如Clojure、Groovy、JRuby、Jython、Scala等。使用过这些语言的开发者可能还不是非常多，但是听说过的人肯定已经不少，随着时间的转移，谁能保证日后Java虚拟机在语言无关性上的优势不会赶上甚至超越它在平台无关性上的优势呢？

实现语言无关性的基础仍然是虚拟机和字节码存储格式（所有JVM上运行的都是遵循JVM规范的Class字节码文件）。Java虚拟机不和包括Java在内的任何语言绑定，它只与“Class文件”这种特定的二进制文件格式所关联，CLass文件中包含了Java虚拟机指令集和符号表以及若干其他辅助信息。基于安全方面的考虑，Java虚拟机规范要求在Class文件中使用许多强制性的语法和结构化约束，但任一门功能性语言都可以表示为一个能被Java虚拟机所接受的有效的Class文件。作为一个通用的、机器无关的执行平台，任何其他语言的实现者都可以将Java虚拟机作为语言的产品交付媒介。例如，使用Java编译器可以把Java代码编译为存储字节码的Class文件，使用JRuby等其他语言的编译器一样可以把程序代码编译成CLass文件，虚拟机并不关心Class的来源是何种语言，如下图所示：

IMG_256

总结来说，编译就是将源文件编译为符合JVM规范的指令集，解释就是将二进制文件解释为符合平台规范的机器码指令集。

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。

# 3 Class类文件结构

本章的主要内容是解析Class文件的数据结构，我将力求在保证逻辑准确的前提下，用尽量通俗的语言和案例去讲述虚拟机中与开发关系最为密切的内容。但是，对数据结构方面的讲解不可避免会比较枯燥，而这部分内容又是了解虚拟机的重要基础之一。如果想比较深入的了解虚拟机，那么这部分是不能不接触的。

在本章关于Class文件结构的讲解中，我们将以《Java虚拟机规范（第2版）》（1999年发布，对应于JDK1.4时代的Java虚拟机）中的定义为主线，这部分内容虽然古老，但它所包含的指令、属性是Class文件中最重要和最基础的。同时，我们也会以后续JDK1.5~JDK1.7中添加的内容为支线进行较为简洁的、介绍性的讲解，如果读者对这部分内容特别感兴趣，建议参考笔者所翻译的《Java虚拟机规范（Java SE7）》中文版。

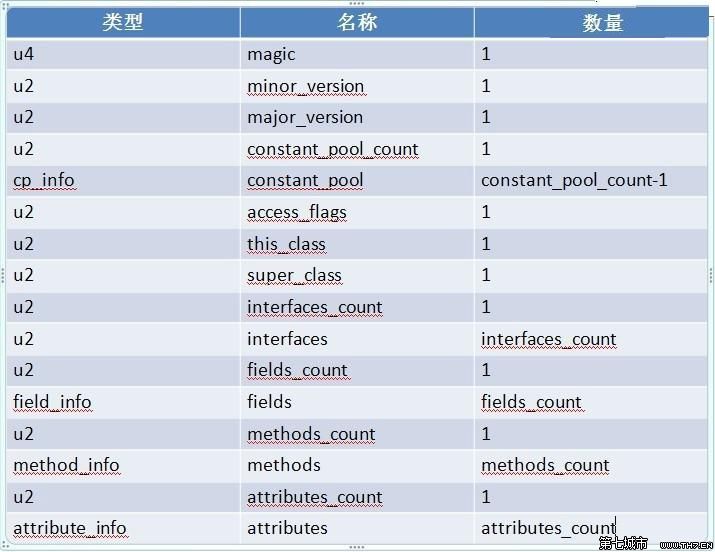
注意：任何一个Class文件都对应着唯一一个类或接口的定义信息，但反过来说，类或接口并不一定都得定义在文件里（譬如类或接口也可以通过类加载器直接生成）。本章中，笔者只是通俗地将任意一个有效的类或接口所应当满足的格式称为“CLass文件格式”，实际上它并不一定以磁盘文件的格式存在。

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位字节以上空间的数据项时，则会按照高位在前（这种顺序称为“Big-Endian”,，具体是指最高位字节在地址最低位、最低位字节在地址最高位的顺序来存储数据，它是SPARC、PowerPC等处理器的默认多字节存储顺序，而x86等处理器则是使用了相反的“Little-Endian”顺序来存储字节）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，所以这里要先介绍这两个概念。

无符号数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，它有下表所示的数据项构成。



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。（集合的Class表现形式）

在结束本小节之前，需要再重复讲一下，Class的结构不像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，长度是什么，先后顺序如何，都不允许改变。接下来我们将一起看看这个表中各个数据项的具体含义。

## 3.1 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），它的唯一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，因为二年间扩展名可以随意地改动。文件格式的制定者可以自由的选择魔数值，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还称作“Oak”语言的时候（大约1991年前后）就已经确定下来了。它还有一段很有趣的历史，据Java开发小组最初的关键成员Patrick Naughton所说：“我们一直在寻找一些好玩的、容易记忆的东西，选择0xCAFEBABE是因为它象征着著名咖啡品牌Peet’s Coffee中深受欢迎的Baristas咖啡”，这个魔数似乎也预示着日后“Java”这个商标名称的出现。

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什么变化，虚拟机也必须拒绝执行超过其版本号的Class文件。（JVM可以运行小于等于其版本号的Class，但不能运行大于其版本的Class，即向下兼容不向上兼容）

例如，JDK1.1能支持版本号为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。现在，我使用的JDK版本是1.7，可生成的Class文件的主版本号最大值为51.0。

为了讲解方便，下面准备了一段代码，本章后面的内容都将以这段小程序使用JDK1.6编译输出的Class文件为基础来进行讲解。

## 指令集与平台