# Class类文件结构

## Java跨平台的基础

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，也是语言无关性的基础。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。

## Class类的本质

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，Class文件实际上它并不一定以磁盘文件的形式存在。

Class文件是一组以8位字节为基础单位的二进制流。

## Class文件格式

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。

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。

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。

## Class文件格式详解

Class的结构不像XML等描述语言，由于它没有任何分隔符号，所以在其中的数据项，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。

按顺序包括：

### 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。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改动。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。

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（MinorVersion），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，JDK 1.1之后的每个JDK大版本发布主版本号向上加1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。

### 常量池

常量池中常量的数量是不固定的，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（constant\_pool\_count）。与Java中语言习惯不一样的是，这个容量计数是从1而不是0开始的

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语言层面的常量概念，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：

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

### 访问标志

用于识别一些类或者接口层次的访问信息，包括：这个Class是类还是接口；是否定义为public类型；是否定义为abstract类型；如果是类的话，是否被声明为final等

### 类索引、父类索引与接口索引集合

这三项数据来确定这个类的继承关系。类索引用于确定这个类的全限定名，父类索引用于确定这个类的父类的全限定名。由于Java语言不允许多重继承，所以父类索引只有一个，除了java.lang.Object之外，所有的Java类都有父类，因此除了java.lang.Object外，所有Java类的父类索引都不为0。接口索引集合就用来描述这个类实现了哪些接口，这些被实现的接口将按implements语句（如果这个类本身是一个接口，则应当是extends语句）后的接口顺序从左到右排列在接口索引集合中

### 字段表集合

描述接口或者类中声明的变量。字段（field）包括类级变量以及实例级变量。

而字段叫什么名字、字段被定义为什么数据类型，这些都是无法固定的，只能引用常量池中的常量来描述。

字段表集合中不会列出从超类或者父接口中继承而来的字段，但有可能列出原本Java代码之中不存在的字段，譬如在内部类中为了保持对外部类的访问性，会自动添加指向外部类实例的字段。

### 方法表集合

描述了方法的定义，但是方法里的Java代码，经过编译器编译成字节码指令后，存放在属性表集合中的方法属性表集合中一个名为“Code”的属性里面。

与字段表集合相类似的，如果父类方法在子类中没有被重写（Override），方法表集合中就不会出现来自父类的方法信息。但同样的，有可能会出现由编译器自动添加的方法，最典型的便是类构造器“＜clinit＞”方法和实例构造器“＜init＞”

### 属性表集合

存储Class文件、字段表、方法表都自己的属性表集合，以用于描述某些场景专有的信息。如方法的代码就存储在Code属性表中。

# 字节码指令

Java虚拟机的指令由一个字节长度的、代表着某种特定操作含义的数字（称为操作码，Opcode）以及跟随其后的零至多个代表此操作所需参数（称为操作数，Operands）而构成。

由于限制了Java虚拟机操作码的长度为一个字节（即0～255），这意味着指令集的操作码总数不可能超过256条。

大多数的指令都包含了其操作所对应的数据类型信息。例如：

iload指令用于从局部变量表中加载int型的数据到操作数栈中，而fload指令加载的则是float类型的数据。

大部分的指令都没有支持整数类型byte、char和short，甚至没有任何指令支持boolean类型。大多数对于boolean、byte、short和char类型数据的操作，实际上都是使用相应的int类型作为运算类型

**阅读字节码作为了解Java虚拟机的基础技能，请熟练掌握。请熟悉并掌握常见指令即可。**

## 加载和存储指令

用于将数据在栈帧中的局部变量表和操作数栈之间来回传输，这类指令包括如下内容。

将一个局部变量加载到操作栈：iload、iload\_＜n＞、lload、lload\_＜n＞、fload、fload\_＜n＞、dload、dload\_＜n＞、aload、aload\_＜n＞。

将一个数值从操作数栈存储到局部变量表：istore、istore\_＜n＞、lstore、lstore\_＜n＞、fstore、fstore\_＜n＞、dstore、dstore\_＜n＞、astore、astore\_＜n＞。

将一个常量加载到操作数栈：bipush、sipush、ldc、ldc\_w、ldc2\_w、aconst\_null、iconst\_m1、iconst\_＜i＞、lconst\_＜l＞、fconst\_＜f＞、dconst\_＜d＞。

扩充局部变量表的访问索引的指令：wide。

## 运算或算术指令

用于对两个操作数栈上的值进行某种特定运算，并把结果重新存入到操作栈顶。

加法指令：iadd、ladd、fadd、dadd。

减法指令：isub、lsub、fsub、dsub。

乘法指令：imul、lmul、fmul、dmul等等

## 类型转换指令

可以将两种不同的数值类型进行相互转换，

Java虚拟机直接支持以下数值类型的宽化类型转换（即小范围类型向大范围类型的安全转换）：

int类型到long、float或者double类型。

long类型到float、double类型。

float类型到double类型。

处理窄化类型转换（Narrowing Numeric Conversions）时，必须显式地使用转换指令来完成，这些转换指令包括：i2b、i2c、i2s、l2i、f2i、f2l、d2i、d2l和d2f。

## 创建类实例的指令：

new。

## 创建数组的指令：

newarray、anewarray、multianewarray。

## 访问字段指令：

getfield、putfield、getstatic、putstatic。

## 数组存取相关指令

把一个数组元素加载到操作数栈的指令：baload、caload、saload、iaload、laload、faload、daload、aaload。

将一个操作数栈的值存储到数组元素中的指令：bastore、castore、sastore、iastore、fastore、dastore、aastore。

取数组长度的指令：arraylength。

## 检查类实例类型的指令：

instanceof、checkcast。

## 操作数栈管理指令

如同操作一个普通数据结构中的堆栈那样，Java虚拟机提供了一些用于直接操作操作数栈的指令，包括：将操作数栈的栈顶一个或两个元素出栈：pop、pop2。

复制栈顶一个或两个数值并将复制值或双份的复制值重新压入栈顶：dup、dup2、dup\_x1、dup2\_x1、dup\_x2、dup2\_x2。

将栈最顶端的两个数值互换：swap。

## 控制转移指令

控制转移指令可以让Java虚拟机有条件或无条件地从指定的位置指令而不是控制转移指令的下一条指令继续执行程序，从概念模型上理解，可以认为控制转移指令就是在有条件或无条件地修改PC寄存器的值。控制转移指令如下。

条件分支：ifeq、iflt、ifle、ifne、ifgt、ifge、ifnull、ifnonnull、if\_icmpeq、if\_icmpne、if\_icmplt、if\_icmpgt、if\_icmple、if\_icmpge、if\_acmpeq和if\_acmpne。

复合条件分支：tableswitch、lookupswitch。

无条件分支：goto、goto\_w、jsr、jsr\_w、ret。

## 方法调用指令

invokevirtual指令用于调用对象的实例方法，根据对象的实际类型进行分派（虚方法分派），这也是Java语言中最常见的方法分派方式。

invokeinterface指令用于调用接口方法，它会在运行时搜索一个实现了这个接口方法的对象，找出适合的方法进行调用。

invokespecial指令用于调用一些需要特殊处理的实例方法，包括实例初始化方法、私有方法和父类方法。

invokestatic指令用于调用类方法（static方法）。

invokedynamic指令用于在运行时动态解析出调用点限定符所引用的方法，并执行该方法，前面4条调用指令的分派逻辑都固化在Java虚拟机内部，而invokedynamic指令的分派逻辑是由用户所设定的引导方法决定的。

方法调用指令与数据类型无关。

## 方法返回指令

是根据返回值的类型区分的，包括ireturn（当返回值是boolean、byte、char、short和int类型时使用）、lreturn、freturn、dreturn和areturn，另外还有一条return指令供声明为void的方法、实例初始化方法以及类和接口的类初始化方法使用。

## 异常处理指令

在Java程序中显式抛出异常的操作（throw语句）都由athrow指令来实现

## 同步指令

有monitorenter和monitorexit两条指令来支持synchronized关键字的语义

# 类加载机制

## 类加载器

## 系统的类加载器

## 双亲委派模型

## Tomcat类加载机制

# 栈桢

## 栈桢详解

# 方法调用详解

# 基于栈的字节码解释执行引擎

Class本质是二进制流，8个字节为基础单位的

{

int ddd;

char[] name;

{

float yyy;

…..

}

}

gif,jpeg

java的版本号从45

常量池：字面量，符合引用

字面量类似于我们常见的常量

符合引用：类和接口的全限定名，字段的名称等等，方法的名称等等

javap –verbose

ONO电信

指令 1个字节（0~255=256个） 特定的数字，指令数不能超过256

dup

ifeq iflt ifne ifnull if\_ while for goto

invoke开头的指令都是方法调用指令

byte short int char Boolean 🡺int

long double float a(代表引用类型)

return

fload