1. 无关性
2. 概述

传统的代码编译的结果是将程序编译为二进制本地机器码，现在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。平台中立的理想只能在操作系统以上的应用层实现，Java虚拟机可以运行在不同硬件平台和操作系统之上，这些虚拟机可以载入和执行同一种平台无关的字节码。

1. 语言无关性

字节码是构成平台无关性的基石，同时语言无关性也很重要，越来越多的语言也能够在Java虚拟机上运行，实现语言无关性的基础仍然是虚拟机和字节码的存储格式，其他的语言同样能够将Java虚拟机作为它们语言的运行基础，以Class文件作为它们产品的交付媒介。

1. 字节码的描述能力

Java语言中的各种语法，关键字，常量变量和运算符号的语义都会由多条字节码指令组合来表达，这决定了字节码指令所能够提供的语言描述能力必须比Java语言更加强大。

1. Class类文件的结构
2. 结构

Class文件是一组以8个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文件之中，中间没有任何分隔符。Class文件的数据结构中只有无符号数和表。

1. 无符号数

无符号数属于基本的数据类型，包括u1,u2,u4,u8来分别表示1个字节，2个字节，4个字节，8个字节的无符号数，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。

1. 表

表是由多个无符号数或者其他表作为数据项构成的复合是数据类型，命名上都以“\_info”来结尾。

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，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，这种数据形式成为集合。

1. 魔数

每个Class文件的头4个字节成为魔数（magic），它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，即身份识别，Class文件的魔数为“0xCAFEBABE”。

1. 版本号
2. 次版本号

紧跟着魔数的两个字节存储的是次版本号（Minor Version），只被短暂使用过，从JDK1.2到JDK12都未使用过，全部固定为0。JDK12之后，为了公测一些复杂特性，将次版本号用于标识“技术预览版”，表示该Class文件使用了测试功能，要将次版本号标识为65535。

1. 主版本号

次版本号后就是主版本号的两个字节（Major Version），Java的版本号是从45开始IDE，高版本的JDK能够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，因为在Class文件校验部分明确要求了即使文件格式未发生变化，虚拟机也拒绝执行超过其版本号的Class文件。

1. 常量池

版本号后就是常量池入口，常量池是Class文件结构中与其他项目关联最多的数据，也是占据Class文件空间最大的数据项目之一。

1. 常量池容量计数器

在常量池的入口需要放置一项u2类型的数据，代表常量池中常量的数量，这个容量池是从1开始计数的。之所以不从0开始，是为了在其他的项目不引用任何一个常量池项目时，将索引值设置为0来表示这种极端情况。

1. 常量池

常量池中主要存放两大类常量：字面量和符号引用。字面量类似于Java的常量概念，如文本字符串，被声明为final的常量值等；而符号引用有类和接口的全限定名，字段的名称和描述符等。

符号引用用于Class文件的加载，Java文件在进行Javac编译时，没有C++的链接操作，而是在虚拟机加载Class文件时进行动态链接，Class文件不会保存各个方法，字段最终在内存中的布局信息，虚拟机需要从常量池中获取对应的符号引用，再在类创建时或运行时解析，翻译到具体的内存地址之中。

常量池中每项常量都是一个表，每个表结构的起始第一位是一个u1类型的标志位，代表着当前的常量属于哪种常量类型。如tag=1表示这是一个UTF-8编码的字符串。

我们通过javap命令在命令行中通过-verbose参数输出Class文件字节码的内容。

1. 访问标志

常量池之后，跟着的两个字节代表访问标志（access\_flags），这个标志用于识别一些类或者接口层次的访问信息，如这个Class是类还是接口，是否定义为public类型，是否定义为abstract类型等等。不同的标志值代表不同的含义，根据类或者接口的特性来计算最后的标志值，如ACC\_PUBLIC表示被public修饰，标志值为0x0001，ACC\_SUPER表示JDK1.0.2之后的版本，标志值为0x0020，则最后的标志值为0x0001 | 0x0020 = 0x0021。

1. 类索引，父类索引和接口索引集合

访问标志后就是索引，类索引和父类索引都是u2类型的数据，而接口索引集合是一组u2类型的数据的集合，Class文件中由这三项数据来确定该类型的继承关系。

1. 类索引

确定这个类的全限定名，指向一个类型CONSTANT\_Class\_info的类描述符常量，通过CONSTANT\_Class\_info类型的常量中的索引值可以找到定义在CONSTANT\_Utf8\_info类型的常量中的全限定名字符串。

1. 父类索引

确定这个类的父类的全限定名，由于Java语言不支持多重继承，因此父类索引只有一个。全限定名的获取与类索引相同，不再赘述。

1. 接口索引集合

描述这个类实现了哪些接口，按照函数签名中的接口顺序从左到右排列在接口索引集合中。入口的第一项u2类型的数据时接口计数器，表示索引表的容量，后面是索引表。

1. 字段表集合
2. 概述

索引之后，就是字段表，字段表用于描述接口或类中声明的变量。Java语言中的字段包括类级变量以及实例级变量，但布包扣在方法内部声明的局部变量。字段可以包括的修饰符有字段的作用域（public, private, protected），是否是实例变量(static)，可变性(final)，并发可见性(volatile)，可否被序列化(transient)，字段数据类型，字段名称。

1. 字段修饰符access\_flags

u2类型，与类中的access\_flags项目非常相似，都是描述字段的属性，如是否public, 是否static等等。

1. name\_index

表示字段的简单名称，是对常量池项的引用，简单名称是指没有类型和参数修饰的方法或者字段名称，如inc()方法和m字段的简单名称为inc和m。

而之前提到的类全名是指以“.”分隔的，如“org.springframework”，而全限定名，是以“/”分隔的，并且需要添加一个“；”如“org/springframework；’”

1. descriptor\_index

表示字段和方法的描述符，描述符用来描述字段的数据类型，方法的参数列表（数量，类型以及顺序）和返回值。基本数据类型和Void都用大写字符来表示，对象类型则用字符L加对象的全限定名来表示，如B表示byte，Ljava/lang/Object表示对象。

对于数组对象，每一个维度将使用一个前置的“[”描述，如一个”java.string.String[][]”类型的二维数组将被记录为“[[Ljava/lang/String;”

对于方法的参数列表，返回值等描述信息，参数列表按照参数的顺序放在一组小括号“()”之内，如void inc()的描述符为“()V”，方法int indexOf(char[] source, int soutce1, int soutce2)的描述符为“([CII])I”。

字段表所包含的固定数据项目到descriptor\_index就结束了了，不过在他之后，还有一个属性表集合，用于存储一些额外的集合，描述额外的信息，如额外的常量。同时，字段表不会列出从父类或者父接口中继承而来的字段，但有可能出现原本Java代码之中不存在的字段，如在内部类中为了保持对外部类的方文星，编译器会自动添加指向外部类实例的字段。

1. 方法表集合

Class文件存储格式中对于方法的描述与对字段的描述采用几乎一样的方式，方法表的结构与字段表一样，依次包括访问标志，名称索引，描述符索引，属性表集合。仅仅由于方法与字段的修饰关键字上的差异而导致了访问标志有些不同，同时属性表集合的可选项也有些不同，如方法的方法体会编译成字节码指令存放在方法属性表集合的Code属性中。

同时，方法表集合中可能会出现编译器自动添加的方法，常见的有类构造器“<clinit>()”方法和实例构造器“<init>()”方法。Java语言中，特征签名指的是一个方法中各个参数在常量池中的字段符号引用的集合，返回值没有包含在特征签名中，因此Java语言无法依靠返回值的不同来重载方法；在Class文件中，特征签名的范围更大一些，只要描述符不是完全一致的两个方法就可以共存，因此如果两个方法的返回值不同，也是可以共存于一个Class文件中的。

1. 属性表集合
2. 概述

属性表存在于Class文件，字段表，方法表之中，描述某些专有的信息。与Class文件中其他数据项目不同，它的格式要求比较宽松，不要求严格的顺序，只要不与已有的属性名重复，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，Java虚拟机在运行时会忽略掉不认识的属性。

1. Code属性

Java程序方法体中的代码经过Javac编译器处理后，变为字节码指令存储在Code属性内。以下介绍一些重要的参数：

1. max\_stack

操作数栈最大深度，在方法执行的任意时刻，操作数栈都不会超过这个深度，虚拟机运行时需要根据这个值来分配栈帧中操作栈的深度。

1. max\_locals

代表了局部变量表所需的存储空间，max\_locals的单位是变量槽，变量槽是虚拟机为局部变量分配内存所使用的最小单位，对于32位以下的数据类型，每个局部变量占用一个槽，对于64位的占用两个槽。操作数栈和局部变量表直接决定了一个该方法的栈帧所耗费的内存。

在任何实例方法中，都可以通过this关键字访问到此方法所属的对象，这种访问机制是通过在Javac编译器编译时将对this关键字的访问转变为对一个普通方法参数的访问，然后在虚拟机调用实例方法时自动传入此参数。因此，在实例方法的局部变量表中至少会存在一个指向当前对象实例的局部变量，局部变量表也会预留第一个变量槽来存放对象实例的引用。注意，这个处理只对实例对象有效，如果该对象是static，那么就是无效的。

1. code

存储字节码指令的一系列字节流，每个指令就是一个u1类型的单字节，当虚拟机读取到code中的一个字节码时，就可以对应找出这个字节码代表的是什么指令，一个u1类型的数据可以表达256条指令。

1. exception\_table

异常表，检查从start\_pc行到end\_pc行中是否有类型为catch\_type或者其子类的异常，如果有就转到handler\_pc行中处理，如果catch\_type的值为0，说明任意异常都需要跳转。

1. Exceptions属性

Exceptions属性的作用是列举出方法中可能抛出的受查异常，也就是方法描述时在throws关键字后面列举出的异常。

1. LinuNumberTable属性

用于描述Java源码行号和字节码行号之间的对应关系，如果不生成该属性，当抛出异常时，堆栈中将不会显示出错的行号，在调试程序时，也无法按照源码行来设置断点。

1. LocalVariableTable属性与LocalVariableTypeTable属性

用于描述栈帧中局部变量表的变量与Java源码中定义的变量之间的关系，如果不生成该属性，则当其他人引用这个方法时，所有的参数名称都会丢失，对于代码的编写极为不便。

JDK5引入泛型后，对于非泛型类型来说，描述符和特征签名能描述的信息是一直的；对于泛型来说，描述符中泛型的参数化类型被擦除了，描述符无法准确地描述泛型类型了，因此需要新增一个LocalVariableTypeTable属性，使用字段的特征签名完成泛型的描述。

1. SourceFile属性与SourceDebugExtension属性

用于记录生成这个Class文件的源码文件名称，如果不生成这个属性，当抛出异常时，堆栈中将不会显示出错代码所属的文件名。为了方便在编译器和动态生成的Class中加入供程序员使用的自定义内容，JDK5新增了SouceDebugExtension属性用于存储额外的代码调试信息，对于非Java语言编写的程序的调试信息可以存储在该属性中。

1. ConstantValue属性

通知虚拟机自动为静态变量赋值，只有被static变量修饰的变量才可以使用这项属性。对于非static类型的变量（实例变量），赋值是在实例构造器<init>()方法中进行的；对于static变量（类变量），赋值可以选择在类构造器<clinit>()方法中或者使用ConstantValue属性。Javac编译器的选择是如果变量被final和static修饰，同时数据类型是基本类型或者java.lang.String，就生成ConstantVlaue属性来进行初始化。

1. InnerClasses属性

用于记录内部类和宿主类之间的关联，如果一个类定义了内部类，编译器就会为它以及他所包含的内部类生成InnerClasses属性。每个内部类都由一个表来米哦按花素，其中包含了内部类和宿主类的符号引用，内部类的名称，内部类的访问标志。

1. Deprecated与Synthetic属性

Deprecated属性表示某个类，字段或者方法已经不再推荐使用，可通过注解@deprecated设置；

Synthetic属性表示此字段或者方法不是由Java源码直接产生的，而是由编译器自行添加的。

1. StackMapTable属性

JDK6添加的相当复杂的变长属性，会在虚拟机类加载的字节码验证阶段被新类型检查验证器使用，目的在于代替之前比较消耗性能的基于数据流分析的类型推导验证器，验证字节码的行为逻辑合法性。

1. Signature属性

JDK5添加的定长属性，任何类，接口，初始化方法或者成员的泛型签名如果包含了类型变量或者参数化类型，则Signature属性都会为它记录泛型签名信息。因为Java语言的泛型采用的是擦除法实现的伪泛型，编译之后都会被擦除，虽然这种实现方式简单，但是在运行期间做反射时无法获取泛型信息。而Signature属性能够弥补这个缺陷，使Java的反射API获取到泛型的类型。

1. MethodParameters属性

JDK8加入的变长属性，用于方法表中，记录方法的各个形参名称和信息。最初Class文件为了存储空间，不存储方法参数名称，这一点对于程序执行没有任何问题，但是对于程序的二次传播就有很大的限制，由于Class文件中没有参数的名称，必须附加上JavaDoc。

之后，使用-g:var参数，将方法采纳数的名称生成到LocalVariableTable属性中，不过局部变量表属性必须有方法体时才会生成，对于接口和抽象方法还是无法存储参数名称。JDK8新增了MethodParameters属性，编译器可以将参数的名称写到Class文件中，运行时通过反射API获取。