类文件结构

笔记本：	jvm		
创建时间：	2018/10/5/周五 10:54	**更新时间：**	2018/10/6/周六 9:50
作者：	1634896520@qq.com		

类文件结构

一、class类文件的结构

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）。**

- class文件是一组以8位字节为基础单位的二进制流。当遇到需要占用8位字节以上空间的数据项时，则会按照高位在前的方式分割成若干个8位字节进行存储。

- class文件格式采用一种类似 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**。

1. **无符号数**：它属于基本类型，以u1\u2\u3\u4\u8来分别代表：一个字节、两个字节、三个字节、四个字节、八个字节的无符号数，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。
2. **表**：由多个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。

（一）魔数与class文件的版本

①每个class文件的头4个字节称为魔数，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。

②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改动。

③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，第7和8个字节是主版本号。

（二）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。

- 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，这个容量计数值是从1开始而不是从0开始的：

目的在于满足后面某些指向常量池的索引值的数据再特定情况下需要表达“不引用任何一个常量池项目”的含义，这种情况就可以把索引值置为0来表示。

- 常量池中主要存放两大类常量：字面量 和 符号引用

1. 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。
2. 符号引用：属于编译原理方面的概念，包括下面三类常量：

   类和接口的全限定类名、字段的名称和描述符、方法的名称和描述符

- 常量池中每一项常量都是一个表，表开始的第一位是一个u1类型的标志位，代表当前这个常量属于哪种常量类型。

- 有一个专门分析class文件字节码的工具：javap（位于jdk bin目录下）

（三）访问标志

在常量池结束之后，紧接着的两个字节代表访问标志（access_flag），这个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。

（四）类索引、父类索引与接口索引集合

- 类索引（this_class）和父类索引（super_class）都是一个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这个类的继承关系。

- 类索引用于确定这个类的全限定类名，父类索引用于确定这个类的父类的全限定类名。

- 除了java.lang.Object外，所有的Java类的父类索引都不为0.

- 类索引、父类索引和接口索引集合都按顺序排列在访问标志之后。

（五）字段表集合

字段表用于描述接口或者类中声明的变量。字段包括类级变量以及实例级变量，但不包括在方法内部声明的局部变量。

*：解释一下全限定名、简单名称、描述符？

全限定名：仅仅是把类全名中的‘.’替换成了'/' 而已，为了使连续的多个全限定名之间不产生混淆，在使用时最后一般会加入一个“；”表示全限定名结束。

简单名称：是指没有类型和参数修饰的方法或者字段名称，这个类中的inc()方法和m字段的简单名称分别是“inc”和“m”。

描述符：描述符的作用是描述字段的数据类型、方法参数列表（包括数量、类型以及顺序）和返回值。

（六）方法表集合

- 方法表的结构如同字段表一样，依次包括了访问标志、名称索引、描述符索引、属性表集合几项。

- 方法里的Java代码，经过编译器编译成字节码指令后，存放在方法属性表集合中一个名为“code”的属性里面。

- 与字段表集合相对应的，如果父类方法在子类中没有被重写，方法表集合中就不会出现来自父类的方法信息。

- 在Java语言中，要重载一个方法，除了要与原方法具有相同的简单名称之外，还要求必须拥有一个与原方法不同的特征签名，特征签名就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，

  也就是因为返回值不会包含在特征签名中，因此Java语言里面是无法仅仅依靠返回值的不同来对一个已有方法进行重载的。

- 如果两个Java方法有相同的名称和特征签名，但是返回值不同，那么也是可以合法共存于同一个class文件中的。

（七）属性表集合

在class文件、字段表、方法表都可以携带自己的属性表集合，以用于描述某些场景特有的信息。
与class文件中其他的数据项目要求严格的顺序、长度和内容不同，属性表集合的限制稍微宽松了一些，不再要求各个属性表具有严格顺序，并且只要不与已有属性名重复，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虚拟机运行时会忽略掉它不认识的属性。

1、code属性
Java程序方法体中的代码经过Javac 编译器处理后，最终变为字节码指令存储在code属性内。接口或者抽象类中的方法不存在code属性。
*：Slot是虚拟机为局部变量分配内存所使用的最小单位。
code_length 和code 用来存储Java源程序编译后生成的字节码指令。
虚拟机规范中明确限制了一个方法不允许超过65535条字节码指令。
Java虚拟机执行字节码是基于栈的体系结构，但是与一般基于堆栈的零字节指令又不太一样，某些指令后面还会带有参数。

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

3、LineNumberTable属性
描述Java源码行号与字节码行号之间的对应关系，它并不是运行时必须的属性，但默认会生成到class文件之中，可以在javac中分别使用-g:none 或 -g:lines 选项来取消或者要求生成这项信息。
如果选择不生成这个属性，对程序运行产生的主要影响就是当抛出异常时，堆栈中将不会显示出错的行号，并且在调试程序的时候，也无法按照源码行来设置断点。

4、LocalVariableTable属性
这个属性用于描述栈帧中局部变量表中的变量与Java源码中定义的变量之间的关系，它也不是运行时必须的属性，但默认会生成到class文件之中，可以在Javac中分别使用-g:none 或 -g:vars选项来取消或要求生成这项信息。如果没有生成这项属性，最大的影响就是当其他人引用这个方法的时候，所有参数名称都将会丢失，IDE将会使用诸如arg0、arg1之类的占位符代替原有的参数名，这对程序运行没有影响，但是会对代码编写带来较大不便，而且在调试期间无法根据参数名称从上下文中获得参数值。

5、SourceFile属性
用于记录生成这个class文件的源码文件名称。可以分别使用Javac的-g:none 或 -g:source选项来关闭或要求生成这项信息。如果不生成这项属性，当抛出异常时，堆栈中将不会显示出错代码所属文件名。

6、ConstantValue属性
通知虚拟机自动为静态变量赋值，只有被static修饰的变量（类变量）才可以使用这项属性。

7、InnerClasses属性
用于记录内部类与宿主类之间的关联。如果一个类中定义了内部类，那编译器将会为它以及它所包含的内部类生成InnerClasses属性。

8、Deprecated以及Synthetic属性
都属于标志类型的布尔属性

- Deprecated属性用于表示某个类、字段或者方法，已经被程序作者定为不再推荐使用，通过@Deprecated注解进行设置。
- Synthetic属性代表此字段或者方法并不是由Java源码直接产生的，而是由编译器自行添加的。

所有由非用户代码产生的类、方法以及字段都勇当至少设置Synthetic属性和ACC_SYNTHETIC标志位中的一项，唯一的例外是实例构造器“<init>”方法和类构造器“<clinit>”方法。

9、StackMapTable属性

这个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，目的在于代替以前比较消耗性能的基于数据流分析的类型推导验证器。

StackMapTable属性汇总包含零至多个栈映射帧，每个栈映射帧都显示或者隐式地代表了一个字节码偏移量，用于表示该执行到该字节码时局部变量表和操作数栈的验证类型。

10、Signature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属性会为它记录泛型签名信息。

二、字节码指令

JAVA字节码在JAVA虚拟机中的地位相当于实体机的机器码，一切在Java虚拟机上运行的程序都要被解释或编译成字节码，一切在实体机上运行的程序最后也都要编译成机器码。Java字节码指令可以对字节码进行操作，在实体机中对机器码进行操作的是汇编语言。所以Java字节码指令对应汇编语言，Java字节码指令集对应汇编指令集。

Java虚拟机的指令是由一个字节长度的、代表某种特殊操作含义的数字（称为操作码）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）而构成。

字节码指令集是一种具有鲜明特点、优劣势都很突出的指令集架构，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；又由于class文件格式放弃了编译后代码的操作数长度对齐，这就意味着虚拟机处理那些超过一个字节数据的时候，不得不在运行时从字节中重建出具体的数据结构。

（一）字节码与数据类型

- 在Java虚拟机的指令集中，大多数的指令都包含了其操作所对应的数据类型信息。
- 对于大部分与数据类型相关的字节码指令，它们的操作码助记符都有特殊的字符来表明专门为哪种数据类型服务。
- 对于arraylength指令，它没有代表数据类型的特殊字符，但操作数永远都只能是一个数组类型的对象。
- 无跳转指令goto时与数据类型无关的。
- Java的指令集对于特定的操作只提供了有限的类型相关指令去支持它。

（二）关于具体指令可参考：https://www.jianshu.com/p/d95cfde7fc49