6.虚拟机类加载机制

6.1 类加载的时机

生命周期：加载、验证、准备、解析、初始化、使用、卸载。其中验证、准备、解析统称为连接。

> 虚拟机严格规定有且只有5中情况必须对类进行初始化
>
> 1. 遇到new、getstatic、putstatic换货invokestatis这4条字节码指令时。
> 2. 使用java.lang.reflect包的方法对类进行反射调用时
> 3. 当子类初始化时必先初始化父类（若父类还未初始化）
> 4. 虚拟机启动时主类（包含main()方法的类），虚拟机会先初始化这个主类
> 5. 当使用JDK1.7动态语言支持时，如果一个java.lang.invoke.MethodHandle实例最后的解析结果REF_ getStatic、REF_ putStatic、REF_ invokeStatic的方法句柄，并且句柄对应的类没有初始化时会先触发其初始化

6.2 类加载的过程

6.2.1 加载

加载阶段，虚拟机需完成三件事：

1. 通过一个类的全限定名来获取此类的二进制字节流
2. 将这个字节流所代表的静态存储结构转化为方法区的运行时数据结构
3. 在内存中生成一个代表这个类的java.lang.Class对象，作为方法区这个类的各种数据的访问入口

> 数组类本身不通过类加载器创建，由Java虚拟机直接创建

加载阶段完成，虚拟机外部的二进制字节流就按照虚拟机所需的格式存储在方法区之中。然后在内存中创建一个java.lang.Class类的对象

> 加载阶段与连接阶段时交叉进行的

6.2.2 验证

1. 文件格式验证，验证字节流是否符合Class文件格式的规范。如常量池中是否有不被支持的常量类型、是否以魔术0xCAFEBABE开头等
2. 元数据验证，对字节码的描述信息进行语意分析，主要进行语义校验。验证点如：这个类是否有父类、其父类是否继承了不被继承的类（被final修饰的类）
3. 字节码验证，最复杂的阶段，主要通过数据流和控制流分析，确定程序语义是合法的、符合逻辑的。
4. 符号引用验证，最后一个校验，发生在虚拟机将符号引用转化为直接引用的时候，在解析阶段中发生

验证阶段是一个非常重要但非必要的阶段，若所运行的代码已经被反复使用和验证过，那么可以通过使用-

Xverify:none来关闭大部分类验证措施，以缩短虚拟机类加载时间

6.2.3 准备

准备阶段是正式为类分配内存并设置类变量初始值的阶段，这些变量所使用的内存在方法区中进行分配。

> 通常情况下变量都是零值（int类型为0，long类型为0L等），但存在特殊情况：如果类字段属性存在ConstantValue属性，准备阶段value就会被初始化ConstantValue属性所指定的值。如：public static final int value = 123.编译时Javac将会为value生成ConstantValue属性，准备阶段将value赋值成123。

6.2.4 解析

解析阶段是虚拟机将常量池内的符号引用替换成直接引用的过程。

> 符号引用：用一组符号来描述所引用的目标，符号可以是任何形式的字面亮，只要使用时能无歧义地定位到目标即可
>
> 直接引用：是指向目标的指针、相对偏移量或是能间接定位到目标的句柄

6.2.4.1 类或接口的解析

...在解析完成前还要进行符号引用验证，确认D是否具备对C的访问权限，无权限则抛出java.lang.IllegalAccessError异常

6.2.4.2 字段解析

1. C本身存在该字段，返回直接引用
2. 否则，C中实现了接口，按照继承关系搜索各个接口和父接口，找到则返回直接引用
3. 否则，C不是java.lang.Object的话，搜索父类是否有改字段，找到则返回直接引用
4. 否则，查找失败抛出java.lang.NoSuchFieldError异常

6.2.4.3 类方法解析

1. 接口和类方法是分开的，若发现C是个接口，抛出java.lang.IncompatiableClassChangeError异常
2. 在类C及其父类中寻找，有则返回方法的直接引用
3. ...

6.3 类加载器

> 同一个class文件由不同的加载器加载出来的类也不同
>
> 双亲委派模型，所有的类加载都会交由自己的父加载器去加载，若加载失败才会尝试自己加载。启动类加载器是顶层类，是所有了加载器的最终父类加载器

7 虚拟机字节码与执行引擎

执行引擎是java虚拟机最核心的组成部分之一，执行java代码，解释执行和编译执行

> Java方法在执行时都会创建一个栈帧，程序在编译的时候就确定了栈帧需要多大局部变量表、多深的操作数栈，分配的内存大小确定，不会在运行时改变

7.1 运行时栈帧结构

1. 局部变量表，存放方法参数和方法内定义的局部变量。以变量槽（slot）为最小单位，单位大小随处理器、操作系统或虚拟机有关。
2. 操作数栈，方法执行过程存放变量
3. 动态链接，每个栈帧都包含一个运行时常量池中该栈帧所属方法的引用，为了支持方法调用过程中的动态链接
4. 方法返回地址，以正常返回语句或抛出异常结束方法
5. 附加信息，如调试相关的信息，具体取决于具体的虚拟机实现

7.2 方法调用

```
java虚拟机提供5条方法调用字节码指令
1. invokestatic:
2. invokespecial:调用实力构造器<init>方法、私有方法和父类方法
3. invokevirtual:
4. invokeinterface:
5. invokedynamic:先解析所调用的方法（多态），再执行方法
前两个指令都可以在解析时唯一确定调用版本
```

类型自动转换顺序，'a', 按照char>int>long>float>double的顺序转型

```
动态类型语言：类型检查的主体过程是在运行期而不是编译期

methodHandle，一种类似于反射的方法调用，但不同的是反射基于java代码层次的方法调用，methodHandle模拟字节码层次的方法调用，其存在3个方法--findStatic(),findVirtual(),findSpecial();methodHandle更加轻量，用的不多
```

7.3 基于栈的字节码解释执行引擎

7.3.1 解释执行

词法分析、语法分析、语义分析、抽象语法树

7.3.2 基于栈的指令集于基于寄存器的指令集比较

> 基于栈的指令集 优点：1.可移植性，相比于寄存器，寄存器由硬件直接提供，程序不可避免的会收到硬件的约束，如不同的处理器提供不同的寄存器；2.代码相对尽愁，每个字节一条指令，编译器实现更贱简单 缺点：执行速度相对慢。指令操作更多，频繁的内存访问，耗时多

计算'1+1'的结果，栈的指令集如下

```
iconst_1
iconst_1
iadd
istroe_0
```

参考

1. 深入理解java虚拟机 第二版
2. https://www.jianshu.com/p/c69f9f7c273b