虚拟机的类加载机制

笔记本：	jvm		
创建时间：	2018/10/6/周六 10:04	**更新时间：**	2018/10/6/周六 18:28
作者：	1634896520@qq.com		
URL：	https://blog.csdn.net/u012900118/article/details/79525931		

虚拟机的类加载机制

一、什么是类加载机制？

虚拟机把描述类的数据从class文件加载到内存，并对数据进行校验、转换解析和初始化，最终形成可以被虚拟机使用的Java类型，这就是**Java虚拟机的类加载机制。**

二、类加载的时机

（一）类从被加载到虚拟机内存开始，到卸载出内存为止，它的生命周期包括：

加载--->验证-->准备-->解析-->初始化-->使用-->卸载

*注意：①其中“验证”、“准备”、“解析”统称为**“连接”**
②加载、验证、准备、初始化和卸载这5个阶段的顺序是确定的，而解析不一定：它在某些情况下可以在初始化阶段之后再开始，这是为了支持Java语言的运行时绑定。

（二）**什么情况下需要开始类加载过程的第一个阶段？**

Java虚拟机没有强制的规定，但是对于初始化阶段，虚拟机严格规定了有且只有5种情况必须立即对类进行初始化。

- 使用Java.lang.reflect包的方法对类进行反射调用的时候，如果类没有进行过初始化则需要先触发其初始化。
- 遇到new\getstatic\putstatic\invokestatic这四条字节码指令时，如果类没有进行过初始化，则需要先触发其初始化。
- 当初始化一个类的时候，若发现其父类都还没有初始化，那么久必须先初始化其父类。
- 当虚拟机启动时，用户需要指定一个要执行的主类，虚拟机会先初始化这个主类。
- 当使用jdk 1.7的动态语言支持时，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，并且这个方法句柄所对应的类没有进行过初始化，则需要先触发其初始化。

*：这5种场景中的行为成为对一个类进行主动引用。除此之外，所有引用类的方法都不会触发初始化，称为**被动引用。**

三、类加载的过程

（一）加载

加载是类加载过程的一个阶段，在加载阶段，虚拟机需要完成以下三件事情：

- 通过一个类的全限定类名获取定义此类的二进制字节流。
- 将这个二进制字节流所代表的静态存储结构转换为方法区的运行时数据结构。
- 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。

对于数组类而言，它是由Java虚拟机直接创建，一个数组类创建过程遵循以下几个原则：

- 如果数组的组件类型是引用类型，那就递归采用上面的加载过程去加载这个组件类型，数组C将在加载该组件类型的类加载器的类名称空间上被标识。
- 如果数组的组件类型不是引用类型，Java虚拟机将会把数组C标记为与引导类加载器关联。
- 数组类的可见性与它的组件类型的可见性一致，如果组件类型不是引用类型，那数组类的可见性将默认为public

（二）验证

验证是连接阶段的第一步，这一阶段的目的是为了确保class文件的字节流中包含的信息符合当前虚拟机的要求，并且不会危害虚拟机自身的安全。

验证阶段大致会完成下面4个阶段的动作：

- 文件格式验证：验证字节流是否符合class文件格式的规范，并且能够被当前版本的虚拟机处理。

  目的：保证输入的字节流能正确地解析并存储于方法区之内，格式上符合描述一个Java类型的信息的要求。
  该阶段的验证是基于二进制流字节流进行的，只有通过了这个阶段的验证后，字节流才会进入到内存的方法区中进行存储，所以后面的3个验证阶段全部是基于方法区的存储结构进行的，不会再直接操作字节流了。

- 元数据验证：对字节码描述的信息进行语义分析，以保证其描述的信息符合Java语言规范的要求。

  目的：对类的元数据信息进行语义校验，保证不存在不符合Java语言规范的元数据信息。

- 字节码验证：这个阶段将对类的方法体进行解析，保证被校验的类的方法不会在运行时危害虚拟机。

  目的：通过数据流和控制流分析，确定语义是合法、符合逻辑的。

- 符号引用验证：这个验证发生正在虚拟机将符号引用转化为直接引用的时候，这个转化动作将在连接的第三个阶段----解析阶段发生。

  符号引用验证可以看作是对类自身以外的信息进行匹配性检验。
  目的：确保解析动作能够正常执行，如果无法通过符号引用验证，那么将会抛出一些异常。

（三）准备

准备阶段是正式为类变量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将在方法区中进行分配。

*注意：1、这时候进行内存分配的仅包括类变量（被static修饰的变量），而不包括实例变量，实例变量在对象实例化时随着对象一起分配在Java堆中
2、这里所说的初始值“通常情况下”是数据类型的零值，就比如：
public static int value =123; //那value在准备阶段后的值是0 而不是123
但如果加上 final，比如：public static final int value =123; //那value在准备阶段后的值就是123

（四）解析

解析阶段是虚拟机将常量池内的符号引用替换为直接引用的过程。
解析动作主要针对类或接口、字段、类方法、接口方法、方法类型、方法句柄和调用点限定符7类符号引用进行。

（五）初始化

根据程序员通过程序制定的主观计划去初始化类变量和其他资源，或者可以从另外一个角度来表达：初始化阶段是执行类构造器<clinit>（）方法的过程。

- <clinit>()方法是由编译器自动收集类中的所有类变量的赋值动作和静态语句块中的语句合并产生的，编译器收集的顺序是由语句在源文件中出现的顺序所决定的，静态语句块中只能访问到定义在静态语句块之前的变量，定义在它之后的变量，在前面的静态语句块可以赋值，但是不能访问。

- <clinit>() 方法与类的构造函数不同，它不需要显式的调用父类构造器，虚拟机会保证在子类的<clinit>()方法执行之前，父类的<clinit>()方法已经执行完毕。
- 由于父类的<clinit>()方法先执行，也就意味着父类中定义的静态语句块要优先于子类的变量赋值操作。
- <clinit>()方法对于类或者接口来说并不是必须的。
- 接口中不能使用静态语句块。
- 虚拟机会保证一个类的<clinit>()方法在多线程环境中被正确地加锁、同步。

四、类加载器

虚拟机设计团队把类加载阶段中的“通过一个类的全限定类名来获取描述此类的二进制字节流”这个动作放到java虚拟机外部去实现，以便让应用程序自己决定如何去获取所需要的类。实现这个动作的代码模块成为“类加载器”。

（一）类与类加载器

对于任意一个类，都需要由加载它的类加载器和这个类本身一同确立其在Java虚拟机中的唯一性，每一个类加载器，都拥有一个独立的类名称空间。

通俗的讲就是：

比较两个类是否“相等”，只有在这两个类是由同一个类加载器加载的前提下才有意义，否则，即使这两个类来源于同一个class文件，被同一个虚拟机加载，只要加载他们的类加载不同，那这两个类就必定不相等。

*（二）双亲委派模型

从Java虚拟机来讲，只存在两种不同的类加载器：一种是启动类加载器，这个类加载器使用c++语言实现，是虚拟机自身的一部分；另一种就是所有其他的类加载器，这些类加载器都由Java语言实现，独立于虚拟机外部，并且全都继承自抽象类java.lang.ClassLoader

绝大部分Java程序都会使用到以下三种系统提供的类加载器：

- 启动类加载器： 加载<JAVA_HOME>\lib目录下核心库
- 扩展类加载器： 加载<JAVA_HOME>\lib\ext目录下扩展包
- 应用程序加载器：加载用户路劲上指定的类库

我们应用程序都是由这三种类加载器互相配合进行加载的，如果有必要，还可以加入自己定义的类加载器。

那么什么是双亲委派模型？

双亲委派模型要求除了顶层的启动类加载器除外，其余的类加载器都应当有自己的父类加载器。类加载器的这种层次关系，就叫作双亲委派模型。

双亲委派模型的工作过程：

如果一个类加载器收到类加载的请求，它首先不会尝试加载这个类，而是把这个请求委派给父类加载器去完成，每一个层次的类加载器都是如此，因此所有的加载请求最终都应该传送到顶层的启动类加载器中，只有当父类加载器反馈自己无法完成这个加载请求时，子加载器才会尝试自己去加载。