类加载机制

虚拟机类加载机制：

类从加载到卸载出内存的过程包括：加载、验证、准备、解析、初始化、使用和卸载七个阶段（其中验证、准备和解析统称为连接）
初始化阶段（触发类初始化）：

- 遇到new、getstatic、putstatic或invokestatic这4条字节码指令时，如果类没有进行初始化，则需要触发其初始化（使用new实例化对象、读取或设置类的静态字段（被final修饰，把结果放在常量池的静态字段除外）以及调用一个类的静态方法时）
- 使用java.lang.reflect包的方法对类进行反射调用时
- 当初始化一个类，如果父类还没有进行过初始化，先触发其父类初始化
- 虚拟机启动时，初始化用户指定的主类
- 当使用JDK1.7的动态语言支持时，如果java.lang.invoke.MethodHandle实例解析结果方法句柄所对应的类没有进行初始化。

备注：

- 通过子类引用父类定义的静态字段，只会触发父类的初始化而不会触发子类的初始化
- 通过数组定义来引用类，不会触发类的初始化
- 输出引用类的常量时，不会触发类的初始化（常量在编译阶段会存入调用类的常量池中，本质上没有直接引用到定义常量的类，不会触发定义常量的类的初始化）
- 接口与类加载稍有不同：当一个接口在初始化时，不需要其父接口全部完成初始化，只有在真正使用父接口时才会初始化

类加载的过程：

加载：

- 通过一个类的全限定名来获取定义此类的二进制字节流
- 通过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
- 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
- 开发人员可以通过定义自己的类加载器去控制字节流的获取方式（即重写loadClass（）方法）

验证：确保Class文件的字节流包含的信息符合当前虚拟机的要求，包含文件格式验证、元数据验证、字节码验证以及符合引用验证
文件格式验证、元数据验证、字节码验证、符号引用验证
准备：正式为类变量（静态变量）分配内存并设置类变量初始值的阶段，内存都将在方法区内进行分配
解析：将常量池内的符号引用替换为直接引用的过程（类或接口的解析、字段解析、类方法解析、接口方法解析）
初始化：执行类构造器<clinit>()方法的过程

类加载器：

对于任意一个类，都需要由加载它的类和这个类本身一同确立其在虚拟机中的唯一性，比较两个类是否相等，必须满足两个类是同一个加载器加载
两种不同的类加载器：启动类加载器（虚拟机自身）、其他的类加载器（虚拟机外部，继承自java.lang.ClassLoader）

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外，其余的类加载器都应当有自己的父类加载器，顺序依次是：

- BootsStrap ClassLoder(启动类加载器）：加载<JAVA_HOME>\lib中的类
- 扩展类加载器：加载<JAVA_HOME>\lib\ext目录中的类
- 应用程序类加载器：加载用户路径上指定的类库（ClassPath）
- 自定义类加载器

双亲模型的工作过程是：

如果一个类加载器收到了类加载的请求，他首先不会自己去尝试加载这个类，而是把这个请求委派给父类加载器去完成，每一层的类加载器都是如此，因此所有的加载请求最终都应传达到顶层的启动类加载器中，只有当父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。

双亲委派模型的好处：

JAVA类随着它的类加载器一起具备了一种带有优先级的层次关系，由于每个类加载都会经过最顶层的启动类加载器，比如java.long.object这样的类在各个类加载器下都是同一个类(只有当两个类是由同一个类加载器加载的才有意义，这两个类才相等)，如果没有双亲委派模型，由各个类加载器自行加载的话。当用户自己编写了一个java.long.objec这样的类，那样系统中就会出现多个Object类，这样 Java 程序中最基本的行为都无法保证，程序会变的非常混乱。