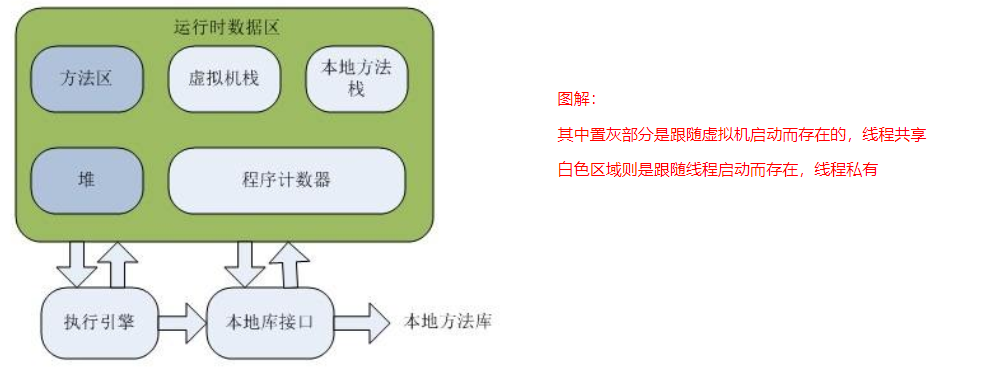
**JVM**

一、JVM结构

1. JVM虚拟机的构成

* 类加载器
* 执行引擎
* 运行时数据区
* 本地库接口

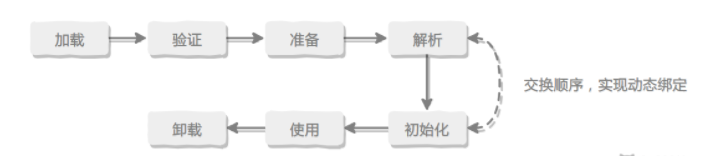
2. JVM运行时数据区的构成



二、垃圾回收

三、类加载机制

1. 类加载的生命周期



在上图中，加载、验证、准备、初始化、卸载的顺序是固定的，而解析阶段则不一定。

2. 加载

在加载阶段，虚拟机需要完成以下三件事情：

* 通过类的全限定名获取定义此类的二进制字节流
* 将这个类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Java堆中生成一个代表这个类的java.lang.Class对象，作为方法区中这些数据的访问入口

3. 验证

在类的验证阶段，主要是确保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。

* 文件格式验证
* 元数据验证
* 字节码验证
* 符号引用验证

4. 准备

准备阶段是正式为类变量分配内存并设置初始值的阶段，这些内存都在方法区中进行分配。但是进行分配的仅仅只是类变量（即static修饰的变量），而不包括实例变量。实例变量将会在对象实例化时随着对象一起分配在Java堆中。

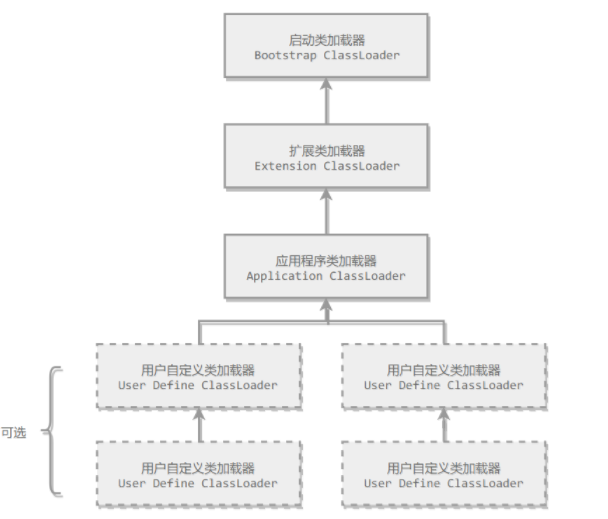
5. 解析

解析阶段是虚拟机将常量池内的符号引用替换为直接饮用的过程。

6. 初始化

初始化阶段才真正开始执行类中定义的 Java 程序代码。初始化阶段是虚拟机执行类构造器 <clinit>() 方法的过程。

7. 双亲委派模型



如果一个类加载器收到了类加载的请求，它首先不会自己去尝试加载这个类，而是把请求委派给父类的加载器去完成，每一个层次的类加载器都是如此。因此最终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，只有当父加载器反馈自己无法完成这个请求时，子加载器才会尝试自己去加载。

优点：使得 Java 类随着它的类加载器一起具有一种带有优先级的层次关系，从而使得基础类得到统一。例如 java.lang.Object 存放在 rt.jar 中，如果编写另外一个 java.lang.Object 并放到 ClassPath 中，程序可以编译通过。由于双亲委派模型的存在，所以在 rt.jar 中的 Object 比在 ClassPath 中的 Object 优先级更高，这是因为 rt.jar 中的 Object 使用的是启动类加载器，而 ClassPath 中的 Object 使用的是应用程序类加载器。rt.jar 中的 Object 优先级更高，那么程序中所有的 Object 都是这个 Object。