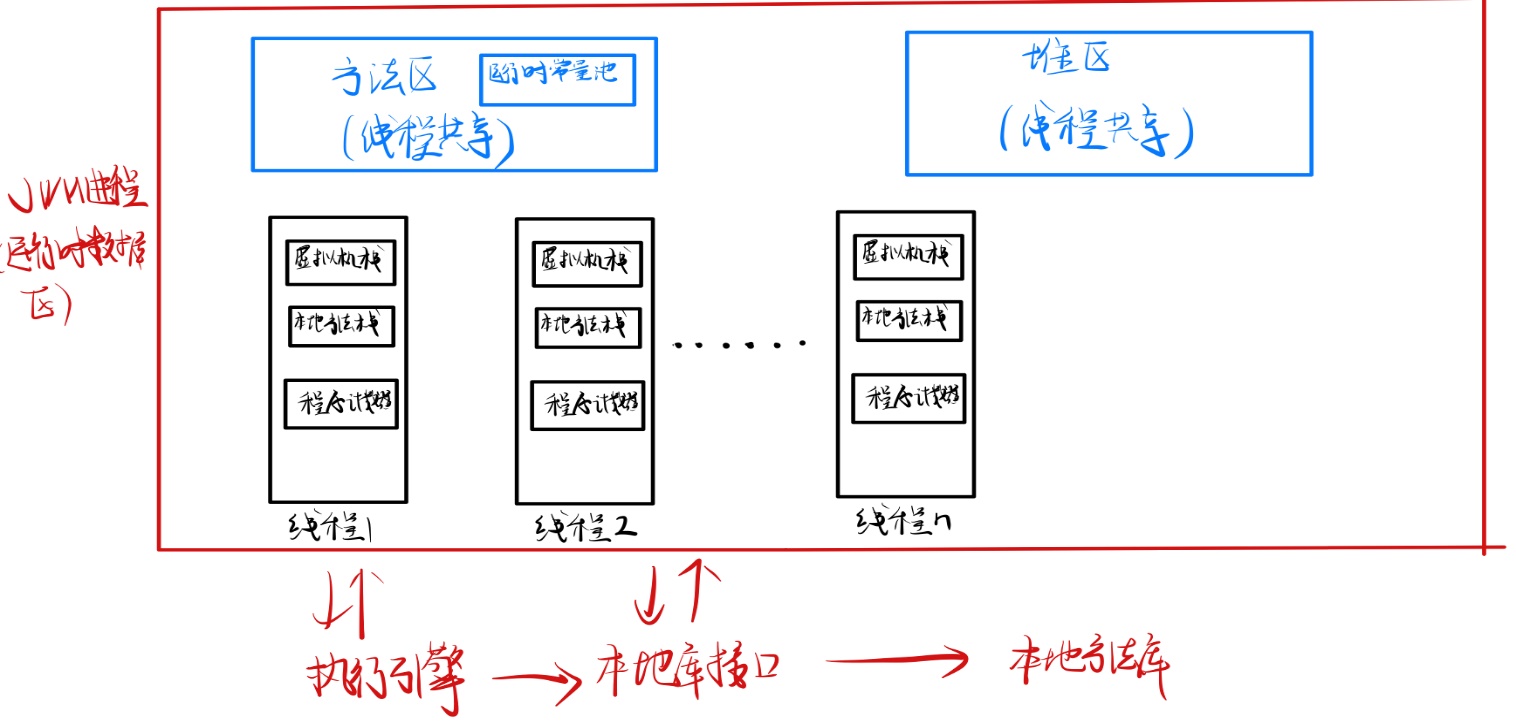
运行Java程序时

**一，首先进行编译，把 .java文件 通过编译器编译为 .class文件（二进制文件/字节码文件）**

**二，创建一个JVM线程（运行时数据区），如下图**



①堆区：存放实例对象

②方法区：存储已加载类的类型信息，常量(static final)，static属性

③程序计数器——JVM进程是多线程进程，每个线程都会执行各自的任务

一个线程如果执行Java方法——该线程的程序计数器记录正在执行的字节码地址

一个线程如果执行本地方法——该线程的程序技术器为空

④虚拟机栈——Java方法

虚拟机栈——存放栈帧（每个方法都有自己的栈帧）

线程A有自己的虚拟机栈，线程A调用方法B，那么B的栈帧入栈

方法B执行完毕，B的栈帧出栈

每个方法都有自己的栈帧

每个栈帧都存放着一个方法的 局部变量表，操作数栈，动态链接，方法出口等

局部变量表——存放编译期可知的基本数据类型和引用——这些数据再局部变量表中的存储空间以槽(Slot)表示，long和double占2个Slot，其他占1个Slot——局部变量表所需的空间在编译时完成分配

⑤本地方法栈——本地方法

同虚拟机栈，不过这里对应的是本地方法

**三，把main()所在的类加载到内存中，调用main( )——开始执行程序**

编译时，会把所有的 .java文件都编译成 .class文件

但是JVM不会把全部的类都加载进内存，而是只有在需要使用某个类时，才会把这个类加载进来，而且只**加载1次**

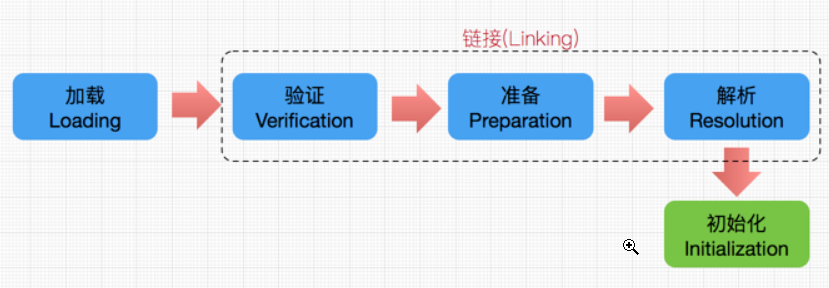
**类加载时机**

**①new 一个实例对象 ②访问类的静态属性，静态方法 ③反射**

**④JVM的启动类(含有main()的类)**

**⑤如果父类的子类进行 类加载，那么父类会在子类 类加载完成前 先完成类加载**

当**触发类加载时机**时，我们会对这个类进行 **类加载**，流程如下



**①加载Loading——由类加载器完成( (1),(2),(3)都是ClassLoader来做)**

**总结来说：将二进制字节流读入JVM内存，并为之创建一个java.lang.Class对象**

(1)通过类的**全限定名**来获取这个类的.class文件(可以从磁盘上获取，也可以从网络获取)

(2)将class文件中的的静态存储结构 转换为 方法区中的运行时数据结构

(3)然后在内存中生成一个代表这个类的**java.lang.Class对象**，作为方法区中运行时数据结构的访问入口，所有对类数据的访问和使用都必须通过这个**Class对象**

**具体看笔记ClassLoader**

**②验证Verification**

(1)**文件格式验证**——主要验证字节流是否符合Class文件格式规范，并且能被当前的虚拟机加载处理。例如：主，次版本号是否在当前虚拟机处理的范围之内。常量池中是否有不被支持的常量类型。指向常量的中的索引值是否存在不存在的常量或不符合类型的常量。

(2)**元数据验证**

这个类是否有父类(Object除外)

这个类是否继承了不允许继承的类(final类)

如果这个类不是抽象类，这个类是否实现了父类和接口中要求实现的所有方法

类中的字段，方法是否与父类产生矛盾

(3)**字节码验证**——最重要的验证环节，分析数据流和控制，确定语义是合法的，符合逻辑的。主要的针对元数据验证后对方法体的验证。保证类方法在运行时不会有危害出现。

(4**)符号引用验证**——主要是针对符号引用转换为直接引用的时候，是会延伸到第三解析阶段，主要去确定访问类型等涉及到引用的情况，主要是要保证引用一定会被访问到，不会出现类等无法访问的问题。

**③准备Preparation**

为static属性在方法区中分配空间并初始化为默认值**（static int i=6，这里会把i放入方法区中，并初始化为0，而不是6，static属性的真正复制在初始化完成）**

为常量在方法区中分配空间，并赋予其应有的值（**final static int a=5,会把a放入方法区，并赋予值5）**

**④解析Resolution**

将.class文件的常量池中的符号引用替换为直接引用

**⑤初始化Initialization——就是执行类构造器<clinit>()方法的过程**

**!!子类初始化前，必须完成父类的初始化**

**!!子类的<clinit>()方法执行前，父类的<clinit>()方法必须执行完**

对于类

(1)检查父类是否已经加载——若未加载，则先加载父类

(2)检查类的接口是否已经加载——若未加载，则先加载接口

(3)Javac编译器会自动收集 static属性赋值语句，static代码块 生成<clinit>()方法（收集顺序按照代码中的顺序） ——然后执行类构造器<clinit>()

**!! 注意static代码块只可以访问定义在它之前的变量，定义在它之后的变量，只能赋值，不能访问**

对于接口

没有(1)(2)——只有用到父接口时，才会加载父接口

并且执行接口的<clinit>()方法并不会先执行父类的<clinit>()方法，只有当使用到父接口时，父接口才会被初始化

JVM必须保证一个类的<clinit>()方法在多线程环境中可以被正确的加锁同步，如果多个线程同时去 初始化一个类，那么只有一个线程可以执行这个类的<clinit>()方法，其他线程都会被阻塞，直至 那个线程执行完<clinit>()方法

<clinit>()和<init>()的区别

<clinit>()

①<clinit>()只有在类加载时才会执行，因为一个类只会加载一次，因此<clinit>()只会执行一次

②<clinit>是javac自动收集 静态赋值语句和 静态代码块 形成的

③<clinit>()的执行发生在类加载的初始化阶段，并且子类初始化前，必须保证父类和接口初始化完毕——也就是说父类 和 接口的<clinit>()方法会先执行，然后再执行子类的<clinit>()

<init>()

①<init>()在创建类对象时执行，每创建一个实例对象，都会执行一次<init>()

②<init>是javac自动收集 构造代码块和构造方法 形成的

③创建子类实例对象前，会先创建父类的实例对象，也就是说会先执行父类的<init>()，然后才会执行子类的<init>()

所以这就是为什么**第一次创建子类对象**时，调用顺序为

父类静态代码块—子类静态代码块—父类构造代码块—父类构造方法—子类构造代码块—子类构造方法

