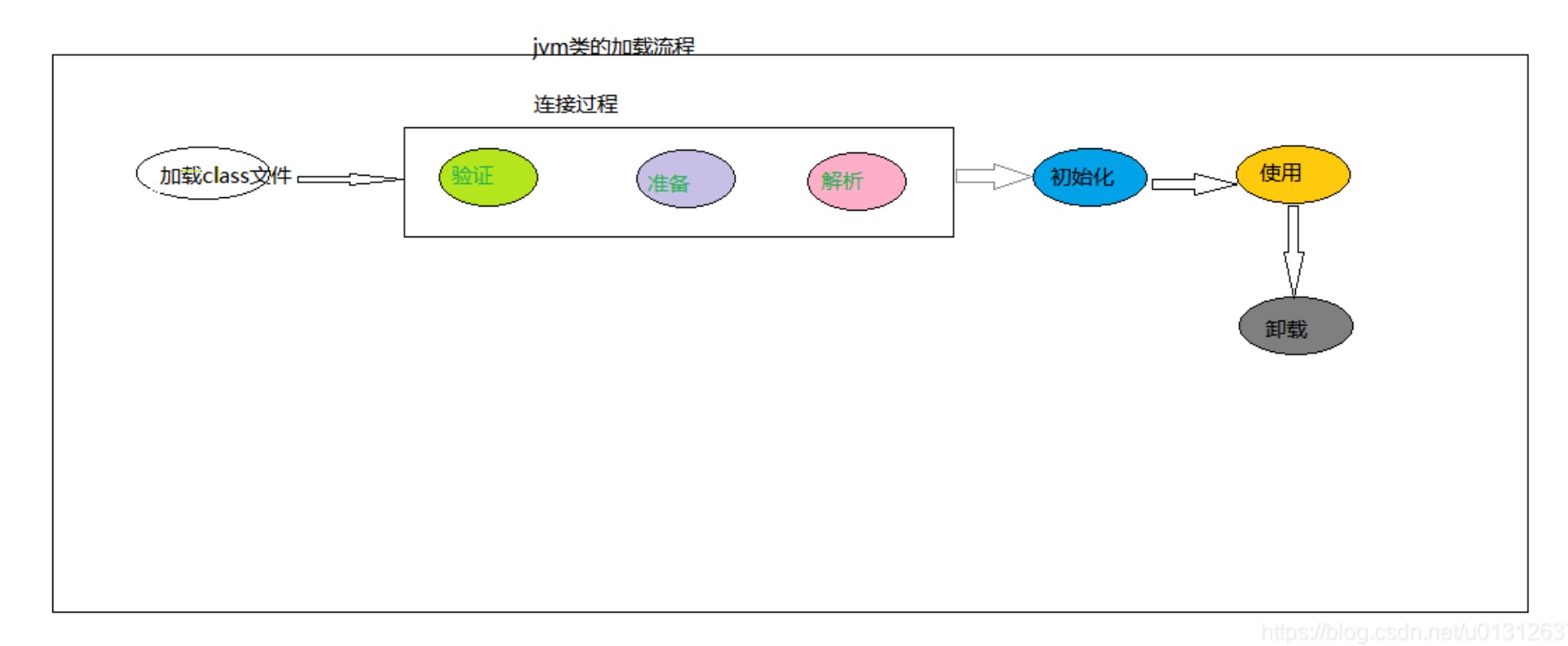
**1、类加载流程**



**加载：**指的就是将class文件加载到内存中，并将这些静态数据转换成方法区中的运行时数据结构，在堆中生成一个java.lang.Class对象。

**验证：**要确保加载的信息符合jvm的规范，没有安全问题。

**准备：**给类变量（静态变量）分配内存，并设置初始值。

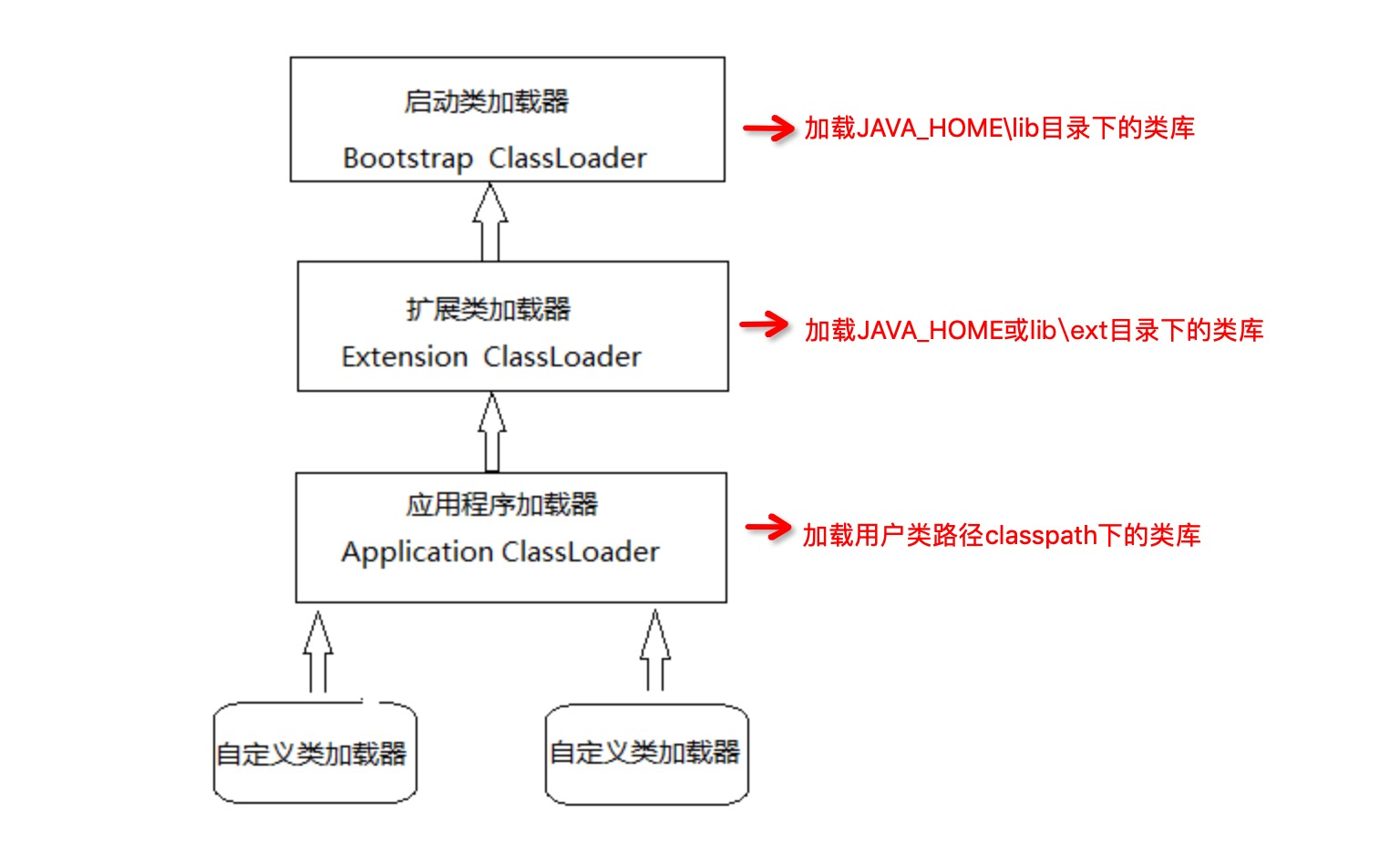
**解析：**虚拟机常量池的符号引用替换为直接引用的过程。

**初始化：**初始化是执行clinit方法（类构造方法）的过程，也就是执行静态变量赋值或者静态代码块

**卸载：**一个已经加载的类型被卸载的几率很小，一般来说用户自定义类加载器加载的类是可以被卸载的。

**2、类加载器**

类加载器采用的是双亲委派模型（组合关系非继承关系），同一个类被不同的类加载器加载后并不是同一个类



**双亲委派模型工作过程：**当子类加载器收到类加载请求时，它首先把这个请求委派给父类加载器去完成，以此递归到最终的启动类加载器中去加载，只有父类加载器反馈无法完成这个加载请求时，子类加载器才会尝试去自己加载

**双亲委派模型的好处：**能保证java程序的稳定运行，比如Object类，它存放在rt.jar中，无论哪一个类加载器进行加载，此Object的类都是同一个，相反，如果没有使用双亲委派模型，而是各个类加载器自行加载，那系统就会出现不同的Object类，非常混乱

**双亲委派模型的弊端：**当基础类（被启动类加载器加载）要调用非基础类中的代码该怎么办？因为启动类不认识这些代码。解决方案是，父类加载器通过线程上下文类加载器（Thread.getContextClassLoader，默认是应用程序类加载器，否则继承父线程中的值）来加载所需的代码，比如JNDI在rt.jar中，但它需要调用应用程序classpath下JNDI接口提供者，此时就需要获取当前线程上下文中的应用程序类加载器来加载了