Java代码是如何在JVM中运行的

1.分享概要

本次分享儒猿专栏**《从 0 开始带你成为JVM实战高手》**中java代码在jvm中执行的内容。本次分享会先从一个java文件开始，一路跟踪它的编译、类加载器加载、以及在JVM中被字节码执行引擎执行，了解执行时是如何与JVM中的各种内存结构交互的。

在开始分享前，我们先思考下面的一些面试题：

1.在JVM层面java代码是如何运行的?

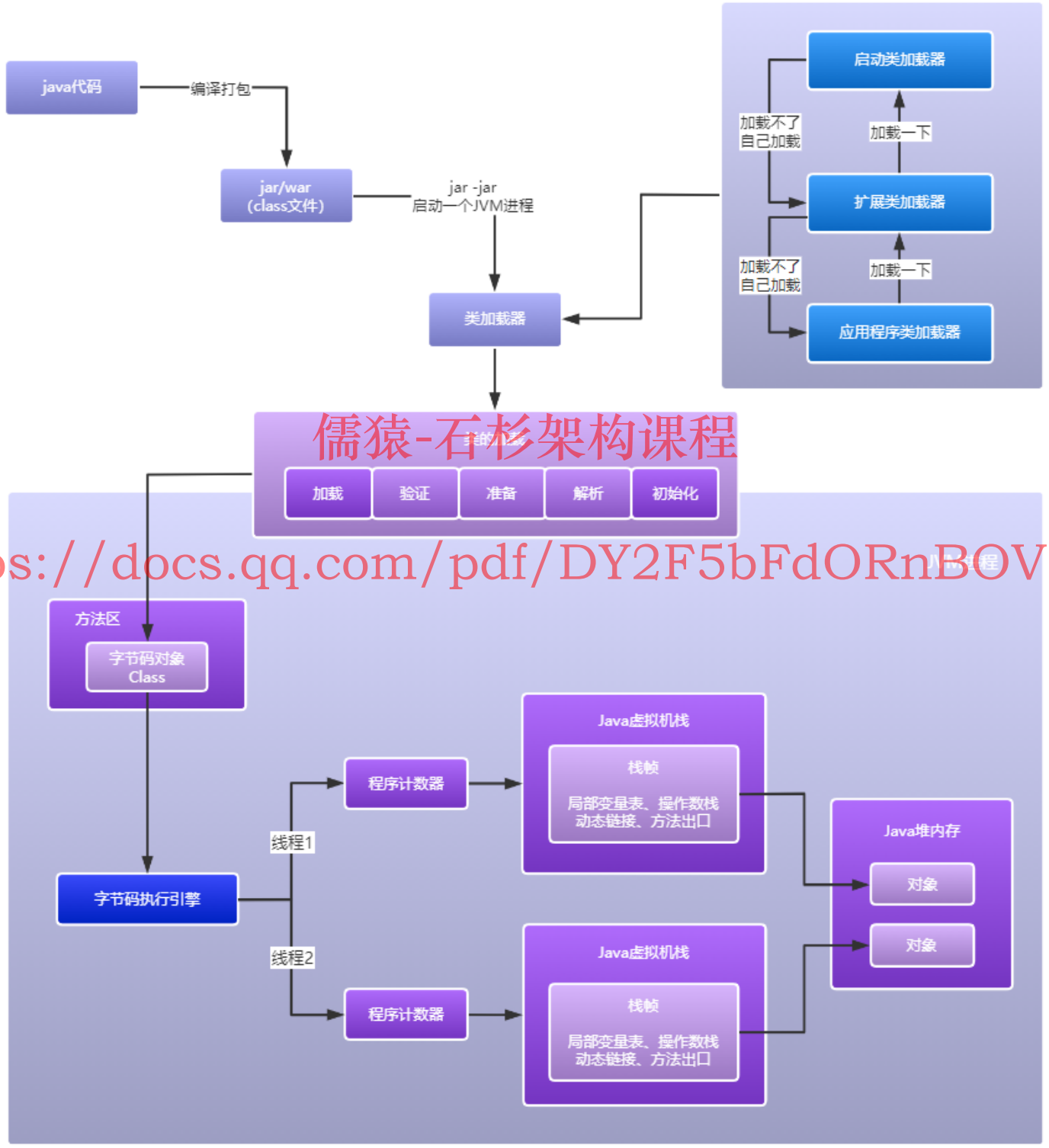
2.java代码可以编译成class文件，也可以反编译窃取信息，可以采用哪些安全措施呢?

3.什么时候会加载一个类呢？什么时候会初始化一个类呢?

4.如何自定义一个类加载器?

5.JVM中class字节码对象什么时候才能被卸载?

Java代码编译、加载和在JVM中运行的流程图如下所示:

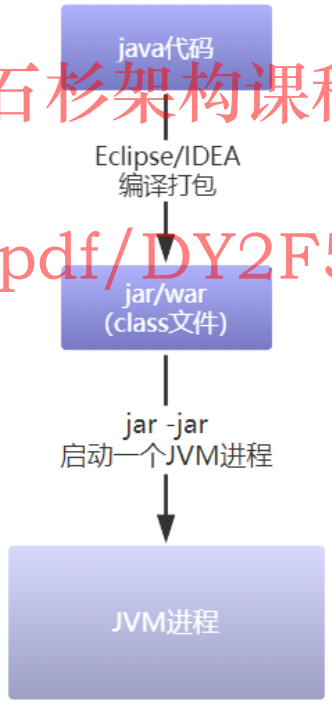


2.Java代码是如何运行起来的

我们在日常项目开发过程中写的java代码，都是以.java后缀的文件，然后编译器如 Eclipse或IDEA就会自动帮我们将java代码编译成以.class为后缀的字节码文件，字节码文件才是JVM可以读取的文件。

当整个项目都开发完成了之后，接下来就要将项目中的java代码编译打包成jar或war，然后通过jar –jar命令或tomcat等web容器部署启动项目了。

当然不管是jar –jar命令启动还是web容器启动，其实效果都是一样的，都会启动一个JVM进程，然后代码的各种逻辑的执行都会在这个JVM进程中开展，如下图所示：



3.类加载器工作原理

java代码通过编译打包后生成jar/war包，之所以能被JVM处理的，就是jar/war包中的java代码已经被编译为.class后缀的字节码文件，但是问题来了，.class后缀的文件也只是个文件啊，它是如何进入到JVM进程中的呢，我们继续看。

3.1 类加载器是如何加载一个类的

为了将class文件加载到JVM中，就要用到类加载器了，类加载器说白了就是实现了将.class后缀文件加载到JVM功能的的代码组件而已，类加载器加载一个类分为以下五个步骤：

加载：通过类的全限定名、获取字节码文件的二进制字节流并加载到JVM内存中;

验证：处理前需要检查下字节流中的信息是否符合JVM规范、字节码文件内容是否被篡改等检查操作;

准备：该阶段主要为类分配内存，并为一些静态的变量设置默认值，如类的静态变量如果是int类型，暂时先

分配0值，如果是引用类型则暂时分配null值；

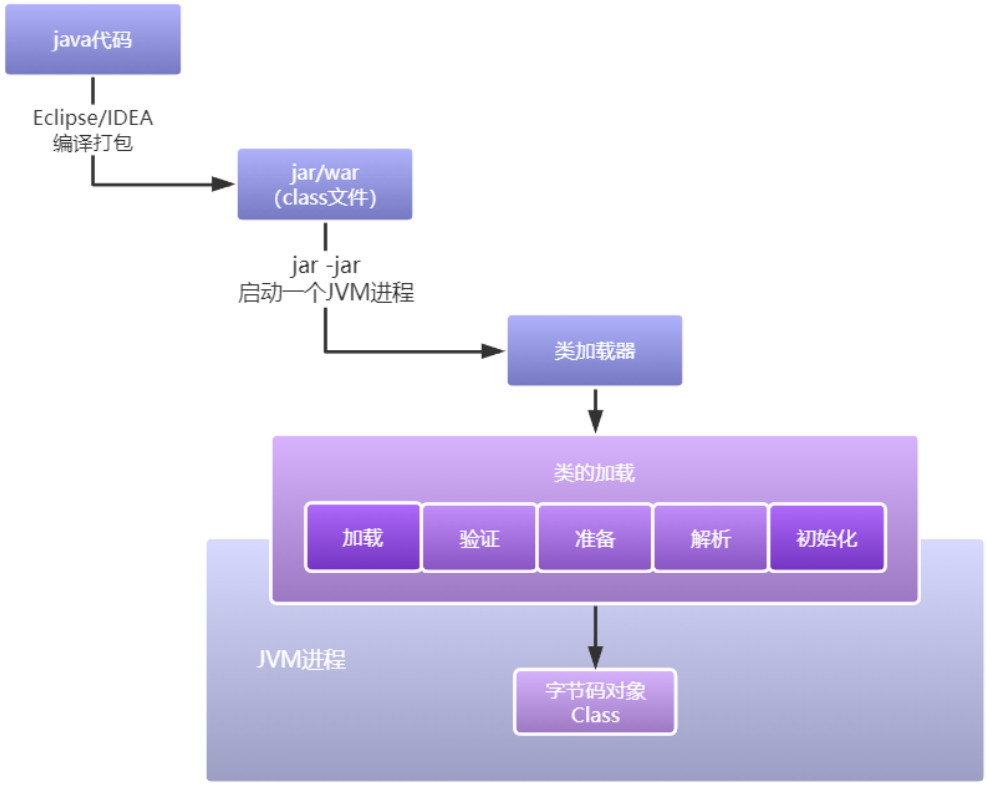
解析：这里主要是将符号引用转换为直接引用，直接引用就是直接指向引用对象的内存地址了;

初始化：初始化阶段就是准备阶段的续期了，准备阶段主要完成了前两步：为类和静态变量分配内存、设置

静态变量的默认值，但是实际值的赋值还得初始化阶段完成，比如int类型值只有现在才能被复制，

而静态变量的引用类型变量在该阶段后才不会为null；

经过了以上五个阶段后，才在JVM中创建了一个可用的class字节码对象，如下图所示：



3.2 类加载器的双亲委派机制

我们这里紧接着看下类加载器的双亲委派机制，java中有三种类加载器，分别是：

启动类加载器

扩展类加载器

应用类加载器

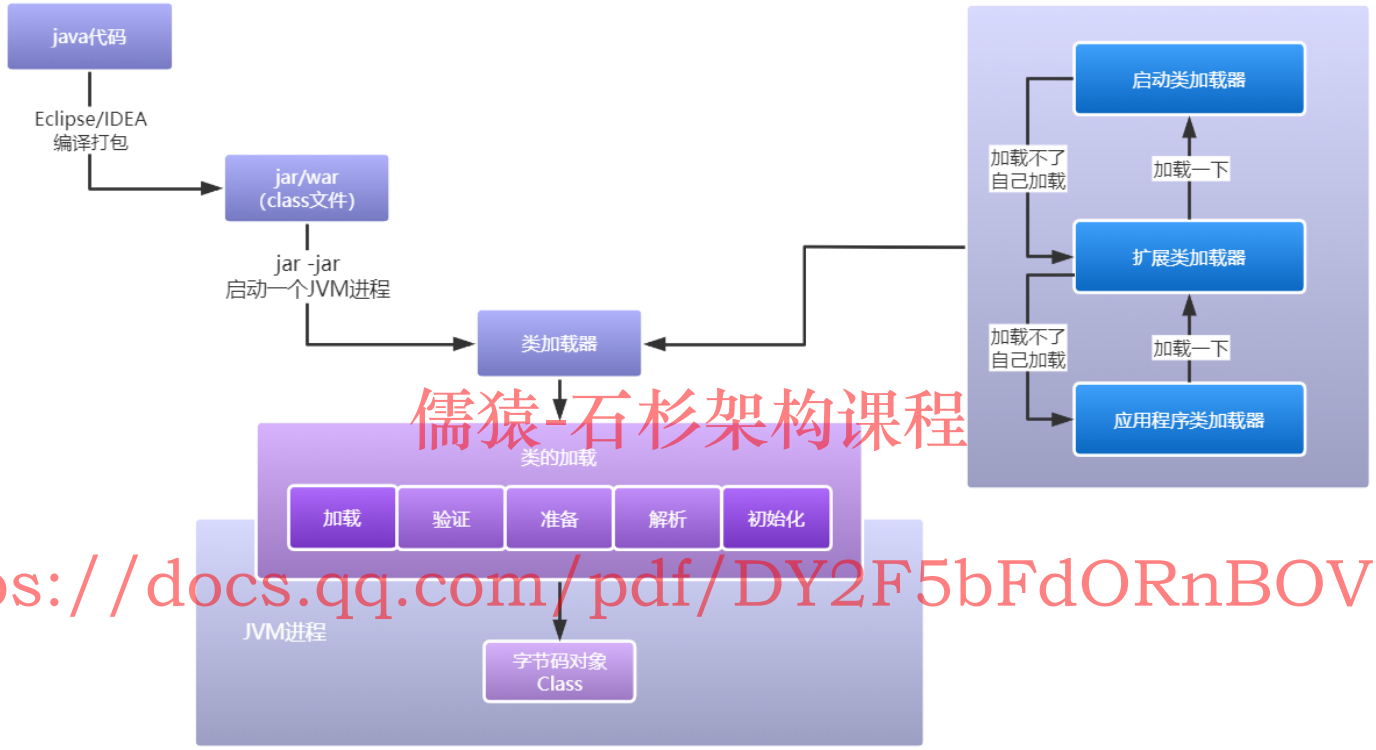
其中启动类加载器和扩展类加载器都是负责加载JDK自带的一些基础组件，而应用类加载器就负责加载我们自己的写的那些代码。

双亲委派模式，特点在于委派方式，当有一个类需要加载时，先委派给应用类加载器；

应用类加载器毫不犹豫就委派给父类扩展类加载器，而扩展类加载器也毫不犹疑委派给了它的父类启动类加载器加载；

现在启动类加载器已经没有父类加载器了，只能自己加载。

启动类加载器如果自己能够加载就成功加载了，比如java.lang.Object这种JDK底层对象，如果是我们自己写的代码，不管是启动类加载器还是扩展类加载器当然都是加载不了的，此时就会一层一层就向下返回给我们的应用类加载器加载，如下图所示：



以上的这种类加载器层面委派的模式就是双亲委派机制，它可以很好的保证不同类加载器加载的内容不会混乱、各自加载各自的。

如归属启动类加载器加载的类，应用程序类加载器就不会被重复加载，假如有人在自己项目中写了一个java.lang.Object对象，想要修改JDK底层的Object对象是无法被加载的：

当应用类加载器加载java.lang.Object时，首先就会一路向上委派到启动类加载器中，启动类加载器检查发现该类是自己负责加载的，但是因为启动类加载器一开始就已经加载了一个java.lang.Object的类，同一个类加载器、对于同一个类的全限定名的类只会加载一次，所以启动类加载器就不会重复加载了；

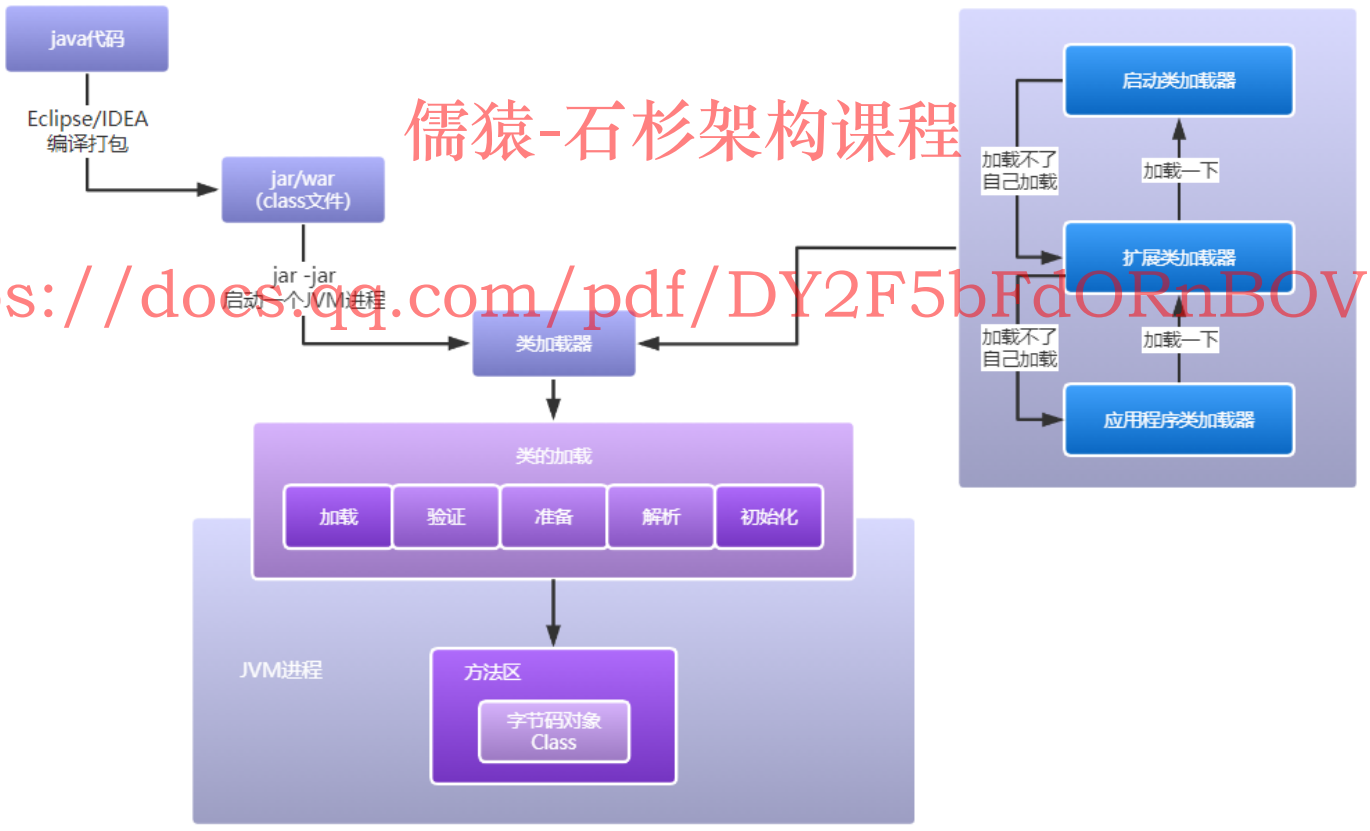
所以通过这种双亲委派模式，可以让各个类加载器各司其职、各自负责加载的范围不会交叉跨越。

4.JVM中的内存数据结构

通过类加载器的加载，此时class字节码文件，已经以class字节码对象的形式给加载到了JVM内存中了，那字节码对象存放在JVM内存的什么位置呢，我们继续看下。

4.1 方法区

存放类相关的信息的区域在JVM称为方法区，方法区同时也是常量池的所在地，如下图所示：



4.2 程序计数器

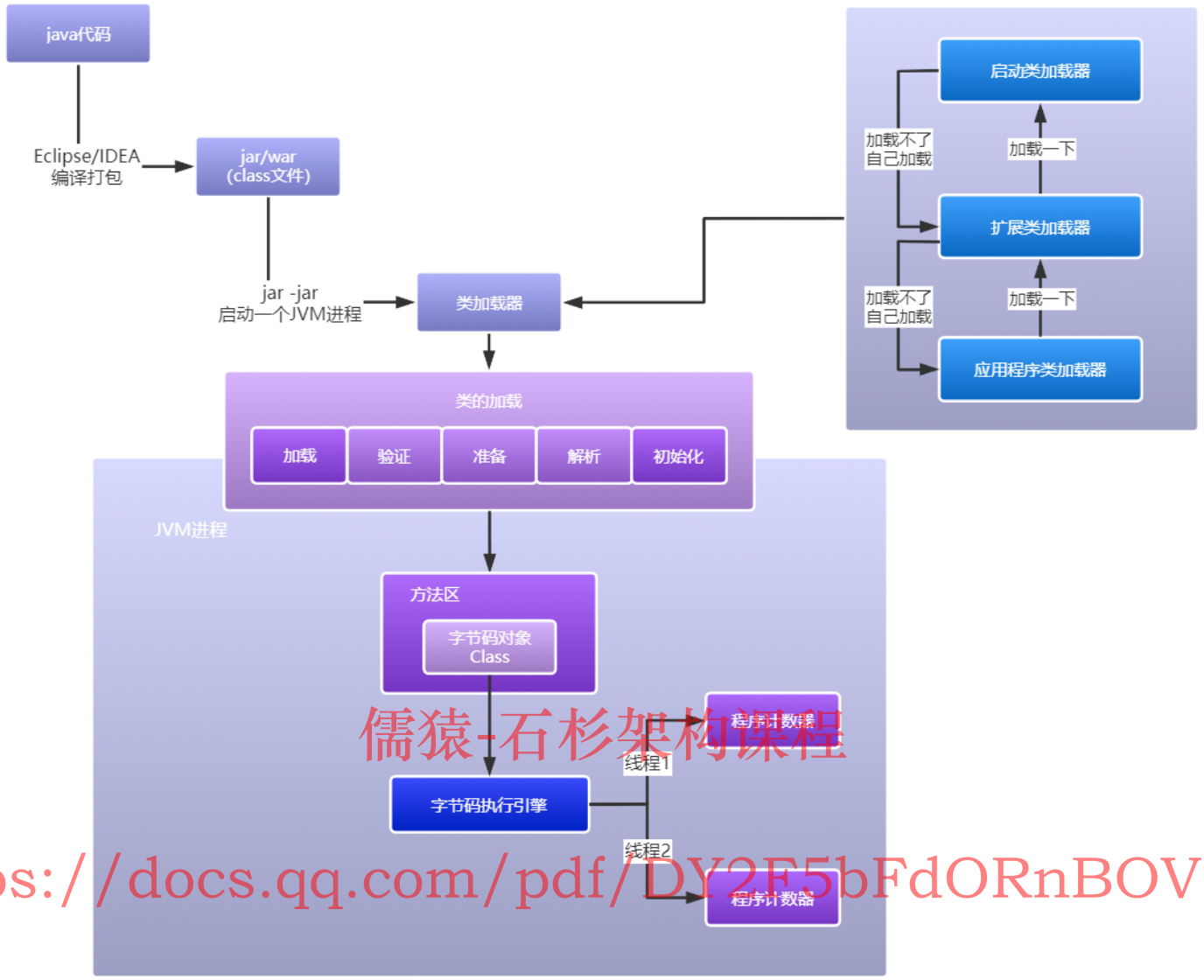
当我们将一个字节码对象给加载到方法区后，下一步就是使用它了，因为java文件中对应的都是一行行的java代码，JVM根本就识别不了，所以才编译成了class字节码文件，字节码文件加载到JVM内存后称为字节码对象，而字节码对象中的、就是JVM可以识别的一行一行字节码指令，而这些字节码的指令则是由JVM中的字节码执行引擎来执行的。

在执行字节码指令时，体现到具体的代码上的操作、可能就是在多个地方new对象，然后执行类中的各种各样的方法，方法执行过程中完全可能存在多线程的操作，毕竟同一个类，我在多个线程中同时new对象是再正常不过的一个场景了。

由于java多线程的执行，每个线程底层是根据CPU分配给它的时间片的方式、依次轮流来执行的，可能A线程执行一段时间后就切换为B线程来执行了，B线程执行时间结束后，再切换回A线程执行了，此时线程A肯定要知道自己上一次执行到字节码指令的哪个位置了，才能在上次的位置继续执行下去。

此时程序计数器就扮演了这样一个、记录每个线程执行字节码指令位置的角色：

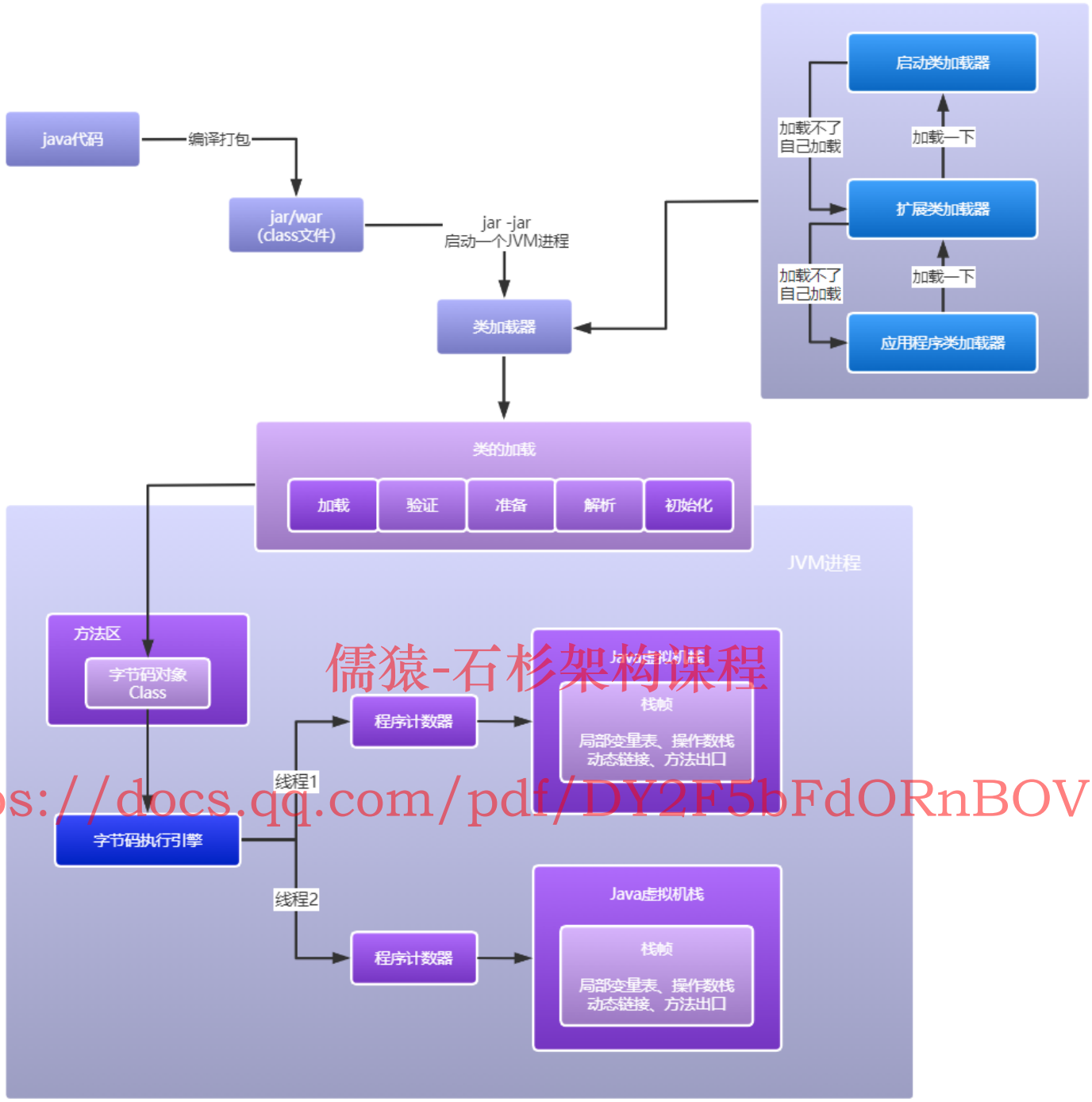
程序计数器每个线程都是私有的，专门为各自线程记录线程、每次执行字节码指令的位置，方便下次线程切换回来时还能找的到上次执行的位置继续执行，如下图所示：



4.3 JVM栈

JVM栈和程序计数器一样，也是每个线程私有的；

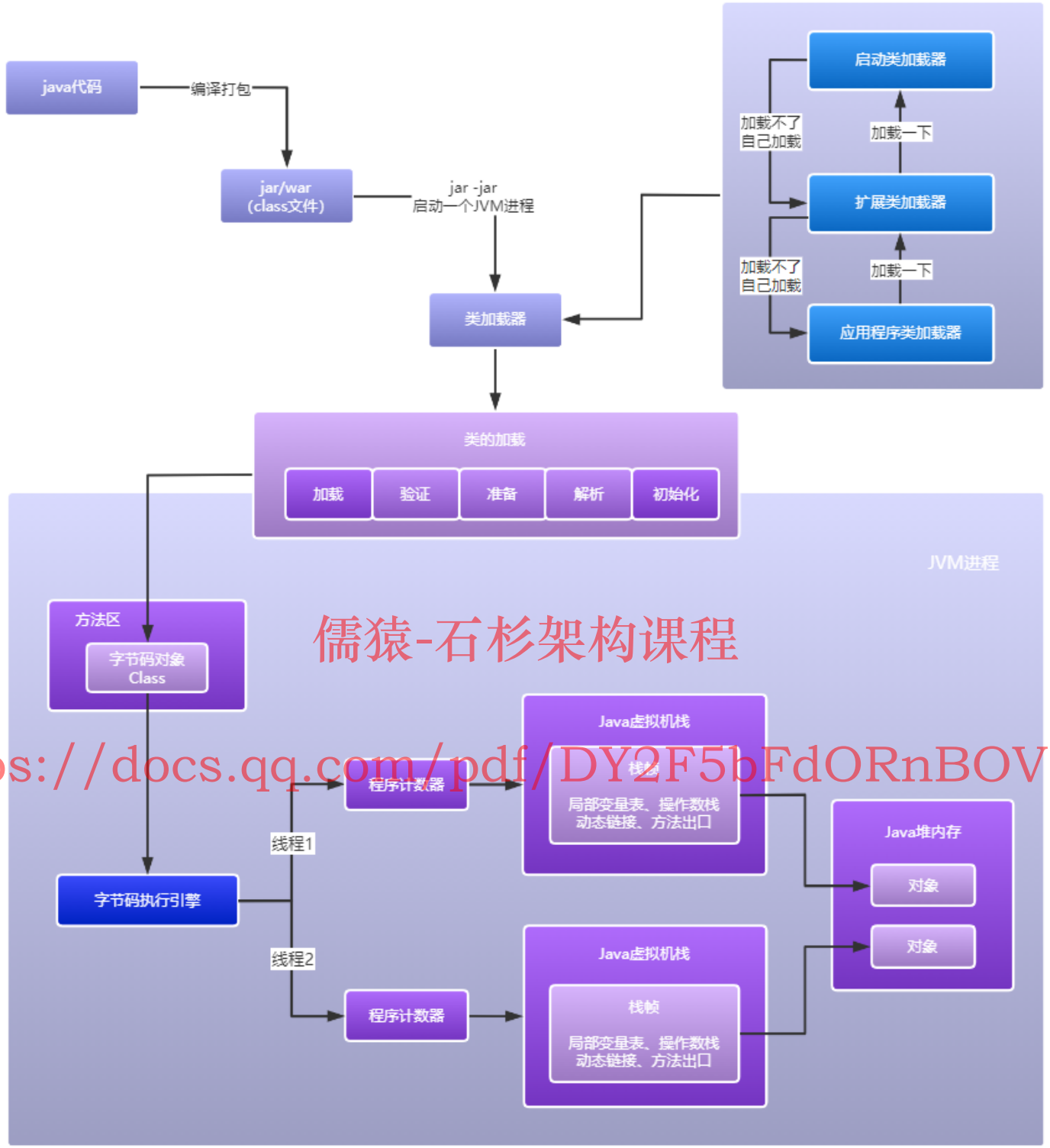
当字节码执行引擎开始执行字节码指令时，对应就是执行java类中的一个个方法的代码逻辑，每执行一个方法就会生成一个栈帧、给压到JVM栈中，每个栈帧中包括局部变量表、操作数栈、动态链接以及方法出口等信息，如下图所示：



当一个方法执行完毕后就会将栈帧给弹出JVM栈，对应的栈帧中的数据也就销毁了。

4.3 堆内存

在方法执行时，可能我们就会执行new操作创建对象，创建出来对象的内存空间的分配，就要分配到另外一个JVM区域即堆内存，如下图所示：



Java堆内存空间和方法区一样，都是多线程共享的区域；

当方法执行完毕后，栈帧就会从JVM栈中弹出，JVM栈中的局部变量表的变量，之前可能指向了堆内存的对象，此时堆内存的对象就没有被局部变量引用了、就成为垃圾对象，下一次gc时就会被垃圾回

收器给回收掉。

讲解到这里，一个java类如何通过编译、类加载器加载到JVM中，然后通过JVM中的字节码执行引擎，配合着JVM中的各种内存区域，完成了整个类的代码逻辑执行，整个流程都已经梳理通了。

5.面试题剖析

1.在JVM层面java代码是如何运行的?

首先java代码通过开发工具编译成class文件，然后由类加载器通过加载、验证、准备、解析过程给加载到JVM中的方法区中。

当字节码执行引擎执行到class字节码对象的指令时，如new操作等，此时就会触发class字节码对象的初始化。

然后线程一边通过字节码执行引擎执行字节码指令、一边程序计数器记录着线程执行指令的位置，每开始执行一个方法就会创建一个栈帧压入到JVM栈中，每次在方法中创建一个对象就会相应在JVM堆内存中开辟一块内存创建对象，由栈帧中局部变量表中的局部变量指向堆内存对象的地址。

方法执行完毕，方法对应的栈帧出栈，同时JVM对堆内存的对象此时可能已经没有局部变量引用了，下一次gc时就可以回收堆内存对象了，整个过程就是java代码运行的轨迹了。

2.java代码可以编译成class文件，也可以反编译窃取信息，可以采用哪些安全措施呢?

可以在java代码编译时，对编译过程进行加密和混淆处理，然后加载类的时候通过我们自定义的类加载器进行解密操、然后对类进行针对性的加载。

3.什么时候会加载一个类呢？什么时候会初始化一个类呢?

加载一个类的时机比较简单，只要用到一个类时就会加载；

而类的初始化时机一般有以下四种时机：

（1）执行了一些特殊的字节码指令，如

getstatic、putstatic、invokestatic、new

这些字节码指令分别对应着：获取一个类的静态变量值、设置类的静态变量值、执行类的静态方法和创建一个类的对象操作，这些操作发生时就会触发一个类的初始化。

（2）通过反射API操作一个类时，会触发类的初始化；

（3）初始化一个类时，该类的父类一定会首先被初始化；

（4）当启动一个JVM进程时，JVM优先会找一个含main方法的类进行初始化；

4.如何自定义一个类加载器?

可以通过创建一个类继承抽象类ClassLoader，然后覆写它的loadClass方法、在方法中自定义类的加载逻辑。

5.JVM中class字节码对象什么时候才能被卸载?

（1）class字节码对象所有的对象实例都要被卸载回收;

（2）加载该字节码对象的类加载器同时也要被卸载;

（3）class字节码对象同时不能被其他对象引用;

此时才能卸载class字节码对象。
