一个Java文件从编码完成到最终执行，一般主要包括两个过程。

1.编译

2.运行

编译，即把我们写好的java文件，通过javac命令编译成字节码，也就是.class文件。

运行，则是把编译生成的.class文件交给java虚拟机（JVM）执行。

而我们所说的类加载过程即是指JVM虚拟机把class文件中类信息加载进内存，并进行解析生成对应的class对象的过程。

JVM在执行某段代码时，遇到了class A，然而此时内存中并没有class A的相关信息，于是JVM就会到相应的class文件中去寻找class A的类信息，并加载进内存中。由此可见，JVM不是一开始就把所有的类都加载进内存中，而是只有第一次遇到某个需要运行的类时才会加载，且只加载一次。

类加载过程主要分为三个部门：

1.加载、2.链接、3.初始化。

而链接也可以细分为三个小部分：

1.验证、2.准备、3.解析

加载：指的是把class字节码文件从各个来源通过类加载器装载到内存中。

下图部分的自定义类加载器需要进一步了解。

这里有两个重点：

- 字节码来源。一般的加载来源包括从本地路径下编译生成的.class文件，从jar包中的.class文件，从远程网络，以及动态代理实时编译
- 类加载器。一般包括启动类加载器，扩展类加载器，应用类加载器，以及用户的自定义类加载器。

注：为什么会有自定义类加载器？

- 一方面是由于java代码很容易被反编译，如果需要对自己的代码加密的话，可以对编译后的代码进行加密，然后再通过实现自己的自定义类加载器进行解密，最后再加载。
- 另一方面也有可能从非标准的来源加载代码，比如从网络来源，那就需要自己实现一个类加载器，从指定源进行加载。

验证：主要是为了保证加载进来的字节流符合虚拟机规范，不会造成安全错误。

包括对于文件格式的验证，比如常量中是否有不被支持的常量？文件中是否有不规范的或者附加的其他信息？

对于元数据的验证，比如该类是否继承了被final修饰的类？类中的字段，方法是否与父类冲突等。

对于字节码的验证，保证程序语义的合理性，比如要保证类型转换的合理性。

对于符号引用的验证，比如校验符号引用中通过全限定名是否能够找到对应的类？校验符号引用中的访问性是否可被当前类访问。

准备：

主要是为类变量（注意，不是实例变量）分配内存，并且赋予初值。

特别需要注意，初值，不是代码中具体写的初始化的值，而是java虚拟机根据不同变量类型的默认初始值。

比如8种基本数据类型的初值，默认为0；引用类型的初值则为null；常量的初值即为代码中设置的值，final static tmp = 456，那么该阶段tmp的初值就是456。

解析：

将常量池内的符号引用替换为直接引用的过程。

两个重点：

1.符号引用。即为一个字符串，但是这个字符串给出了一些能够唯一性识别一个方法，一个变量，一个类的相关信息。

2.直接引用。可以理解为一个内存地址，或者一个偏移量。比如类方法，类变量的直接引用是指向方法区的指针；而实例方法，实例变量的直接引用则是从实例的头指针开始算起到这个实例变量位置的偏移量

举个例子来说，现在调用方法hello()，这个方法的地址是1234567，那么hello就是符号引用，1234567就是直接引用。

在解析阶段，虚拟机会把所有的类名、方法名、字段名这些符号引用替换为具体的内存地址或偏移量，也就是直接引用。

初始化：

这个阶段主要是对类变量初始化，是执行类构造器的过程。

换句话说，只对static修饰的变量或语句进行初始化。

如果初始化一个类的时候，其父类尚未初始化，则优先初始化其父类。

如果同时包含多个静态变量和静态代码块，则按照自上而下的顺序依次执行。