一文搞懂 java -jar 发生了什么

你好，我是 shengjk1，多年大厂经验，努力构建 通俗易懂的、好玩的编程语言教程。 欢迎！你会有如下收益：

1. 了解大厂经验
2. 拥有和大厂相匹配的技术等

希望看什么，评论或者私信告诉我！

一、前言

之前一直就很好奇 java -jar 到底发生了什么，为什么执行 java -jar 代码就自动运行了。今天我们来说明一下，尽量覆盖操作系统、编译原理、JVM 的一些东西。(本文将处于一个不断更新的状态，知道上面这些东西覆盖的差不多了为止，如果可以的话，也会加上硬件方面的东西)，主要的目的就是为了能以最简单的 java 代码来串一些相对来说比较底层的东西，让自己以及让每个读者对计算机能有一个相对全局的了解。

我们先约定如下：
1.操作系统仅仅指的是unix 或类unix

2. 64 位机器
3. 64位 jDK

我们把下面这个类，打成一个 jar 包然后执行。

```

```

```
/**
 * Created by shengjk1 on 2020/8/30.
 */
public class Test {
public static int b;
private int a;
```

```
public static void main(String[] args) {
Test test = new Test();
test.test();
System.out.println(”b = ” + b);
System.out.println(”执行完毕”);
}

public void test() {
byte i = 15;
int j = 8;
int k = i + j;
}
}
```

学过 java 的同学应该都知道这个 Test 类的每行代码都是干嘛的，就不一一解释了。

二、关于编译

首先会编译成 class 文件

在这里插入图片描述](https://p3-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/6afce86203ad411cb75a38df6f5b779d~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=538&h=636&s=304084&e=png&b=d1cefe)关于 java 的编译器
在这里插入图片描述](https://p3-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/9a1edefd0ff3445f8403787516bf9607~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=1131&h=469&s=111872&e=png&b=ffffff)编译的 class 内容
在这里插入图片描述](https://p3-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/b5659dd70263462bb55bdb617d754777~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=1052&h=701&s=399226&e=png&b=c5ecca)有 cafe babe 魔数，还有什么常量池呀之类的，稍后补充

下面开始执行

三、关于 shell

k3u1fbpfcp/d8011638eb0e4e8da586eafdf0c11417~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=896&h=303&s=31046&e=png&b=000000)
执行的时候我们启动了一个 命令行客户端 进程，可以理解为 shell 的一种。所谓的 shell 在操作系统中的位置
![在这里插入图片描述](https://p3-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/d5475e6b01c24f0cba475a2ea271e817~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=504&h=412&s=128795&e=png&b=fefdfe)
当然此 shell 非彼 shell，操作系统中的 shell 更加宽泛一下，像图形界面也是 shell 的一种。

四、关于进程

我们刚才仅仅用鼠标那么轻轻的一点就创建了一个 命令行客户端 进程，而对于操作系统而言进程是如何创建的呢?
会由用户态进入到内核态，然后由操作系统执行 fork 命令，此时进程开始创建，
会包括 虚拟地址空间、修改进程表、会占用寄存器、会有打开文件的清单等等信息，创建完成之后就可以执行了。我们的 命令行客户端 也就起来了
![在这里插入图片描述](https://p3-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/5e5cdf9158c6481a9616e9deb7d54400~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=905&h=539&s=56123&e=png&b=000000)等待用户输入，用户的每次输入，然后回车，其实对于操作系统而言都是创建一个新的进程。

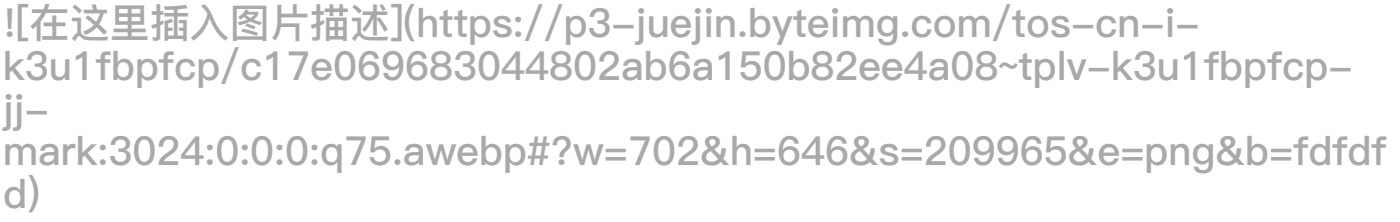
五、执行 java -jar

![在这里插入图片描述](https://p3-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/50cb06a7aa0e4d5fb554165e6f2ecb6b~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=900&h=297&s=40212&e=png&b=000000)同理会 fork 一个 JVM 进程出来，JVM 创建的过程中会启动 Bootstrap ClassLoader 加载 Java 的核心类库 (JAVA_HOME/jre/lib/rt.jar、resource.jar 或者是 sun.boot.class.path 路径下的内容)，供 JVM 自身需要。(关于 JDK、JRE、JVM 可以参考 [读 Differences between JDK, JRE and JVM](http://cxyroad.com/”https://blog.csdn.net/jsjsjs1789/article/details/106782221?ops_request_misc=%257B%2522request%255Fid%2522%253A%252216003218581972483582119 6%2522%252C%2522scm%2522%253A%252220140713.130102334.pc%255Fblog.%2522%257D&request_id=160032185819724835821196&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~blog~first_rank_v1~rank_blog_v1-1-

106782221.pc_v1_rank_blog_v1&utm_term=jre&spm=1018.2118.3001.4187”))

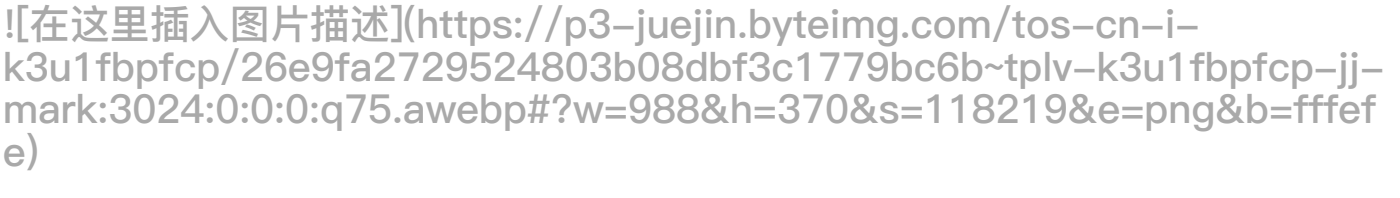
JVM 的准备工作完成之后，JVM会调用我们的 main() 方法，可是内存里面并没有 main 方法，这就是所说的 页面故障，操作系统会从磁盘上读取相应的指令。也就进入了 JVM 的类加载。

六、类加载

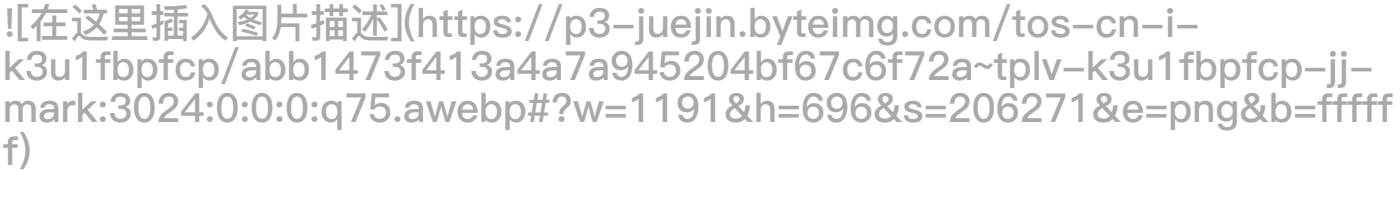
类加载得有加载器


七、加载

要加载 main() 方法所在的 Test 类，会首先判断有没有没有加载的父类，若有未加载的父类则会先加载其父类。在这里我们的 Test 类并没有明确的父类,JVM就把 Test 类加载到 JVM 的内存中形成一个 java.lang.Class 对象
而对象在JVM 中的内存布局如下：



所以说未压缩的情况下 class 对象至少占用 8 byte(32 位 JVM) 16byte (64 位 JVM)

这个过程中，会把类的版本、字段、方法、等描述信息以及代码缓存放入 Metaspace，把常量池表中的各种字面常量符号引用等放入方法区的运行时常量池。


八、验证

同时会对 class 文件进行验证，包括文件格式、元数据等，以保证 class 文件不危害虚拟机自身的安全。

九、准备

加载验证结束后，开始进入准备阶段，主要做两件事情

1. 类变量初始化，此处是初始化为 0 值，比如 int、long
2. 初始化虚方法表（java 多态 ，也就是在运行期间才能确定具体调用哪个方法都可以称为虚方法 ）

十、解析

准备阶段完成之后，开始解析，主要做一件事

1. 将常量池中的符号引用转化为直接引用
主要针对类或接口、字段、类方法、接口方法等

凡是在此阶段可以解析的方法引用都成为静态解析，调用的时候就叫静态调用
静态解析一般都是静态方法和私有方法，并且在运行期间是不变的

十一、初始化

我更喜欢类的初始化，因为我们调用了 main() 方法，实际上是 静态调用 invokestatic 。
类初始化的几种情况：

1. 遇到 new、getstatic、putstatic或 invokestatic 时，如果未初始化则先初始化(1. new 2.读取或设置一个类的静态字段 (被 final 修饰、已经在编译期把结果放入常量池的静态字段除外) 3. 调用一个类的静态方法)
2. 使用 java.lang.reflect 包的方法对类进行反射调用时，如果未初始化则先初始化
3. 当初始化类时，如果其父类未初始化则先触发其父类初始化
4. 当虚拟机启动时，用户需要指定一个要执行的主类，虚拟机会先初始化这个

类
5. 当使用 动态语言支持时，如果 java.lang.invoke.MethodHandle 的解析结构为 REF_*static、REF_new*句柄，并且这个句柄对应类没有进行初始化，需要先初始化
6. 当有 默认方法 接口的实现类发生了初始化，则该接口要在其初始化之前初始化
接口并不要求父接口全都完成初始化，只有在真正使用到 父接口 的时候才会初始化
类初始化其实就是调用类构造器() 方法的过程，而() 是由编译器 Javac 自动收集类中的所有类变量的赋值动作和静态语句块中的语句合并而成的(顺序不变)，并且 JVM 会保证在子类 () 执行前，父类的()已经被执行完毕 (第一个被执行的 一定是 java.lang.Object)，并且 JVM 会保证一个类的 () 线程安全，被正确的加锁同步，并且有且仅会有一个线程去执行 () (同一个类加载器下，一个类型只会被加载一次)，其他线程会阻塞直到 () 执行完毕

当然了类初始化完了之后如果需要会进行对象的初始化，调用对象的构造器 ()，调用之前会先调用父类的。

十二、main 方法调用

执行 main 方法也就需要方法调用，对于方法调用 JVM 是通过几条指令来实现的

)方法对应的符号引用主要有两种

1. 一部分在 类加载解析阶段或者第一次使用转为直接引用 (静态解析 方法在真正运行前就有一个可确定的调用版本，并且在运行期是不变的 。主要有静态方法和私有方法 (不可能通过继承或别的方式重写))
2. 一部分在 每一次运行期间都转化为直接引用 (动态链接 invokevirtual)

只要能被 invokestatic 和 invokespecial 指令调用的方法，都可以在类加载的解析阶段转化为直接引用 (静态方法、私有方法、实例构造器、父类方法(super.)、被final 修饰的方法)，对应的方法称为非虚方法，其他的都是虚方法 (在运行期间根据实际类型确定方法执行版本)。
虚方法主要揭示了 java 多态的一些特征，像多态、方法重写。

十三、main 方法执行

我们都知道方法是在栈中执行的，方法的执行过程其实就是不断的出栈入栈的过程

我们以 test() 方法为例来具体分析一下

0: bipush 将 15 放入栈中
2: istore_1 将栈顶元素方入局部变量表第 1 个位置
3: bipush 将 8 放入栈中
5: istore_2 将栈顶元素方入局部变量表第 2 个位置
6: iload_1 将局部变量表的第 1 个位置元素放入栈
7: iload_2 将局部变量表的第 2 个位置元素放入栈
8: iadd 相加
9: istore_3 将栈顶元素(也就是相加的结果)方入局部变量表第 3 个位置

==========

==========

==========

==========

6，7 一起

==========

mark:3024:0:0:0:q75.awebp#?w=1248&h=542&s=41338&e=png&b=fefefe)==========

![在这里插入图片描述](https://p3-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/df50ea44a2164933bbbc7457672bc8b9~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=1268&h=556&s=41352&e=png&b=fefefe)

然后 return ，主方法(调用该方法的方法)的 PC寄存器的值可以作为返回地址，然后继续执行。

打印输出

打印输出会从用户态进入内核态，操作系统会调用 IO 操作输出相应的结果。

退出

发生系统调用，JVM 退出

十四、补充
---

1. 在电脑中，系统调用（英语：system call），指运行在用户空间的程序向操作系统内核请求需要更高权限运行的服务。系统调用提供用户程序与操作系统之间的接口。大多数系统交互式操作需求在内核态运行。如设备IO操作或者进程间通信。

![在这里插入图片描述](https://p3-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/6a0e0aa0c0ce4a2eb4e91e27caf72cfa~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=1442&h=1060&s=683443&e=png&b=00a305)2. 操作系统的进程空间可分为用户空间和内核空间，它们需要不同的执行权限。其中系统调用运行在内核空间。

3.常见系统调用
![在这里插入图片描述](https://p3-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/d953bef8db6f4d229d53287002e82a79~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=652&h=736&s=696444&e=png&b=efefef)

十五、总结

```
本文详细介绍了Java程序执行的过程，从编译到最终执行，包括操作系统、进程、类加载、方法调用等方面的内容。通过对每个阶段的详细解释，读者可以更好地理解Java程序的执行过程。同时，本文也介绍了一些相关的概念，如系统调用、进程空间等。本文对于Java程序员来说是一篇非常有价值的文章。
```

原文链接: https://juejin.cn/post/7367292352275218471