

讲堂 > 深入拆解 Java 虚拟机 > 文章详情

## 01 | Java代码是怎么运行的？

2018-07-20 郑雨迪



01 | Java代码是怎...

朗读人：郑雨迪 10'36" |  
6.09M

0:00 / 0:00

我们学院的一位教授之前去美国开会，入境的时候海关官员就问他：既然你会计算机，那你谈谈你用的都是什么语言吧？

教授随口就答了个 Java。海关一看是懂行的，也就放行了，边敲章还边说他们上学那会学的是 C+。我还特意去查了下，真有叫 C+ 的语言，但是这里海关官员应该指的是 C++。

事后教授告诉我们，他当时差点就问海关，是否知道 Java 和 C++ 在运行方式上的区别。但是又担心海关官员拿他的问题来考别人，也就没问出口。那么，下次你去美国，不幸地被海关官员问这个问题，你懂得如何回答吗？

作为一名 Java 程序员，你应该知道，Java 代码有很多种不同的运行方式。比如说可以在开发工具中运行，可以双击执行 jar 文件运行，也可以在命令行中运行，甚至可以在网页中运行。当然，这些执行方式都离不开 JRE，也就是 Java 运行时环境。

实际上，JRE 仅包含运行 Java 程序的必需组件，包括 Java 虚拟机以及 Java 核心类库等。我们 Java 程序员经常接触到的 JDK（Java 开发工具包）同样包含了 JRE，并且还附带了一系列开发、诊断工具。

然而，运行 C++ 代码则无需额外的运行时。我们往往把这些代码直接编译成 CPU 所能理解的代码格式，也就是机器码。

比如下图的中间列，就是用 C 语言写的 Helloworld 程序的编译结果。可以看到，C 程序编译而成的机器码就是一个一个的字节，它们是给机器读的。那么为了让开发人员也能够理解，我们可以用反汇编器将其转换成汇编代码（如下图的最右列所示）。

```
1 ; 最左列是偏移；中间列是给机器读的机器码；最右列是给人读的汇编代码
2 0x00: 55                                push    rbp
3 0x01: 48 89 e5                            mov     rbp, rsp
4 0x04: 48 83 ec 10                          sub     rsp, 0x10
```

[复制代码](#)

```
5 0x08: 48 8d 3d 3b 00 00 00 lea    rdi,[rip+0x3b]
6                                     ; 加载 "Hello, World!\n"
7 0x0f: c7 45 fc 00 00 00 00 mov    DWORD PTR [rbp-0x4],0x0
8 0x16: b0 00                                mov    al,0x0
9 0x18: e8 0d 00 00 00          call   0x12
10                                     ; 调用 printf 方法
11 0x1d: 31 c9                                xor    ecx,ecx
12 0x1f: 89 45 f8                          mov    DWORD PTR [rbp-0x8],eax
13 0x22: 89 c8                                mov    eax,ecx
14 0x24: 48 83 c4 10                        add    rsp,0x10
15 0x28: 5d                                pop     rbp
16 0x29: c3                                ret
```

既然 C++ 的运行方式如此成熟，那么你有没有想过，为什么 Java 要在虚拟机中运行呢，Java 虚拟机具体又是怎样运行 Java 代码的呢，它的运行效率又如何呢？

今天我便从这几个问题入手，和你探讨一下，Java 执行系统的主流实现以及设计决策。

## 为什么 Java 要在虚拟机里运行？

Java 作为一门高级程序语言，它的语法非常复杂，抽象程度也很高。因此，直接在硬件上运行这种复杂的程序并不现实。所以呢，在运行 Java 程序之前，我们需要对其进行一番转换。

这个转换具体是怎么操作的呢？当前的主流思路是这样子的，设计一个面向 Java 语言特性的虚拟机，并通过编译器将 Java 程序转换成该虚拟机所能识别的指令序列，也称 Java 字节码。这里顺便说一句，之所以这么取名，是因为 Java 字节码指令的操作码（opcode）被固定为一个字节。

举例来说，下图的中间列，正是用 Java 写的 Helloworld 程序编译而成的字节码。可以看到，它与 C 版本的编译结果一样，都是由一个个字节组成的。

并且，我们同样可以将其反汇编为人类可读的代码格式（如下图的最右列所示）。不同的是，Java 版本的编译结果相对精简一些。这是因为 Java 虚拟机相对于物理机而言，抽象程度更高。

```
1 # 最左列是偏移；中间列是给虚拟机读的机器码；最右列是给人读的代码
2 0x00:  b2 00 02          getstatic java.lang.System.out
3 0x03:  12 03            ldc "Hello, World!"
4 0x05:  b6 00 04          invokevirtual java.io.PrintStream.println
5 0x08:  b1                    return
```

[复制代码](#)

Java 虚拟机可以由硬件实现 [1]，但更为常见的是在各个现有平台（如 Windows\_x64、Linux\_aarch64）上提供软件实现。这么做的意义在于，一旦一个程序被转换成 Java 字节码，那么它便可以在不同平台上的虚拟机实现里运行。这也就是我们经常说的“一次编写，到处运行”。

虚拟机的另外一个好处是它带来了一个托管环境（Managed Runtime）。这个托管环境能够代替我们处理一些代码中冗长而且容易出错的部分。其中最广为人知的当属自动内存管理与垃圾回收，这部分内容甚至催生了一波垃圾回收调优的业务。

除此之外，托管环境还提供了诸如数组越界、动态类型、安全权限等等的动态检测，使我们免于书写这些无关业务逻辑的代码。

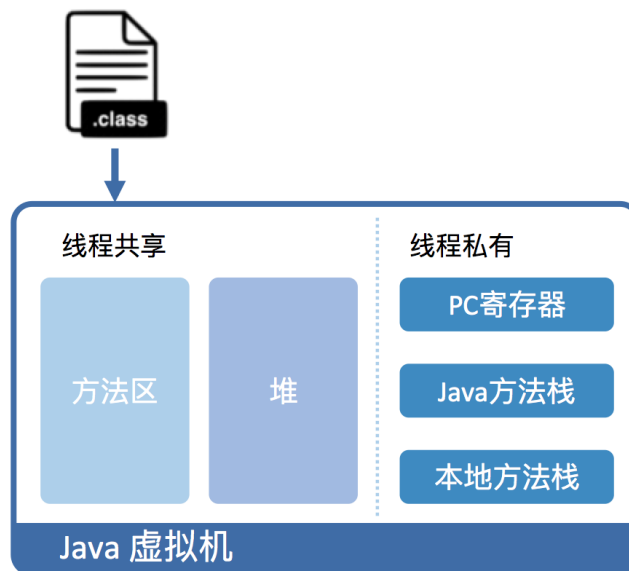
## Java 虚拟机具体是怎样运行 Java 字节码的？

下面我将以标准 JDK 中的 HotSpot 虚拟机为例，从虚拟机以及底层硬件两个角度，给你讲一讲 Java 虚拟机具体是怎么运行 Java 字节码的。

从虚拟机视角来看，执行 Java 代码首先需要将它编译而成的 class 文件加载到 Java 虚拟机中。加载后的 Java 类会被存放于方法区（Method Area）中。实际运行时，虚拟机会执行方法区内的代码。

如果你熟悉 X86 的话，你会发现这和段式内存管理中的代码段类似。而且，Java 虚拟机同样也在内存中划分出堆和栈来存储运行时数据。

不同的是，Java 虚拟机会将栈细分为面向 Java 方法的 Java 方法栈，面向本地方法（用 C++ 写的 native 方法）的本地方法栈，以及存放各个线程执行位置的 PC 寄存器。

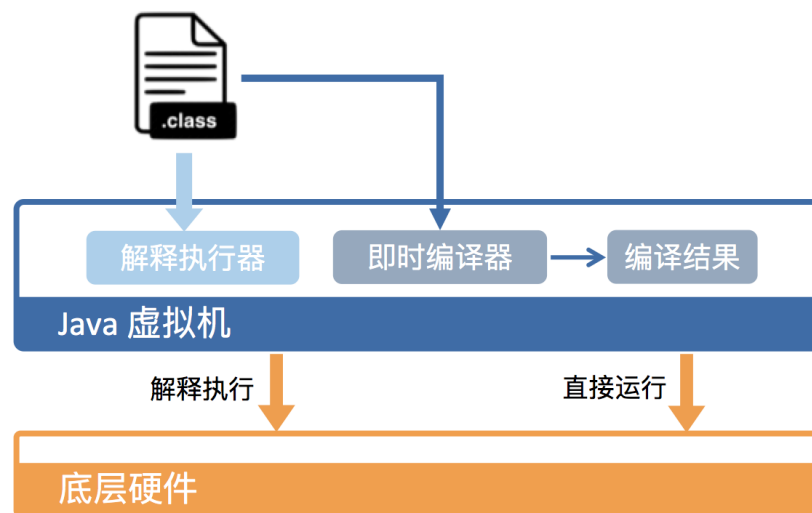


在运行过程中，每当调用进入一个 Java 方法，Java 虚拟机会在当前线程的 Java 方法栈中生成一个栈帧，用以存放局部变量以及字节码的操作数。这个栈帧的大小是提前计算好的，而且 Java 虚拟机不要求栈帧在内存空间里连续分布。

当退出当前执行的方法时，不管是正常返回还是异常返回，Java 虚拟机均会弹出当前线程的当前栈帧，并将之舍弃。

从硬件视角来看，Java 字节码无法直接执行。因此，Java 虚拟机需要将字节码翻译成机器码。

在 HotSpot 里面，上述翻译过程有两种形式：第一种是解释执行，即逐条将字节码翻译成机器码并执行；第二种是即时编译（Just-In-Time compilation, JIT），即将一个方法中包含的所有字节码编译成机器码后再执行。



前者的优势在于无需等待编译，而后者的优势在于实际运行速度更快。HotSpot 默认采用混合模式，综合了解释执行和即时编译两者的优点。它会先解释执行字节码，而后将其中反复执行的热点代码，以方法为单位进行即时编译。

Java 虚拟机的运行效率究竟是怎么样的？

HotSpot 采用了多种技术来提升启动性能以及峰值性能，刚刚提到的即时编译便是其中最重要的技术之一。

即时编译建立在程序符合二八定律的假设上，也就是百分之二十的代码占据了百分之八十的计算资源。

对于占据大部分的不常用的代码，我们无需耗费时间将其编译成机器码，而是采取解释执行的方式运行；另一方面，对于仅占据小部分的热点代码，我们则可以将其编译成机器码，以达到理想的运行速度。

理论上讲，即时编译后的 Java 程序的执行效率，是可能超过 C++ 程序的。这是因为与静态编译相比，即时编译拥有程序的运行时信息，并且能够根据这个信息做出相应的优化。

举个例子，我们知道虚方法是用来实现面向对象语言多态性的。对于一个虚方法调用，尽管它有很多个目标方法，但在实际运行过程中它可能只调用其中的一个。

这个信息便可以被子编译编译器所利用，来规避虚方法调用的开销，从而达到比静态编译的 C++ 程序更高的性能。

为了满足不同用户场景的需要，HotSpot 内置了多个即时编译器：C1、C2 和 Graal。Graal 是 Java 10 正式引入的实验性即时编译器，在专栏的第四部分我会详细介绍，这里暂不做讨论。

之所以引入多个即时编译器，是为了在编译时间和生成代码的执行效率之间进行取舍。C1 又叫做 Client 编译器，面向的是对启动性能有要求的客户端 GUI 程序，采用的优化手段相对简单，因此编译时间较短。

C2 又叫做 Server 编译器，面向的是对峰值性能有要求的服务器端程序，采用的优化手段相对复杂，因此编译时间较长，但同时生成代码的执行效率较高。



从 Java 7 开始，HotSpot 默认采用分层编译的方式：热点方法首先会被 C1 编译，而后热点方法中的热点会进一步被 C2 编译。

为了不干扰应用的正常运行，HotSpot 的即时编译是放在额外的编译线程中进行的。HotSpot 会根据 CPU 的数量设置编译线程的数目，并且按 1:2 的比例配置给 C1 及 C2 编译器。

在计算资源充足的情况下，字节码的解释执行和即时编译可同时进行。编译完成后的机器码会在下次调用该方法时启用，以替换原本的解释执行。

## 总结与实践

今天我简单介绍了 Java 代码为何在虚拟机中运行，以及如何在虚拟机中运行。

之所以要在虚拟机中运行，是因为它提供了可移植性。一旦 Java 代码被编译为 Java 字节码，便可以在不同平台上的 Java 虚拟机实现上运行。此外，虚拟机还提供了一个代码托管的环境，代替我们处理部分冗长而且容易出错的事务，例如内存管理。

Java 虚拟机将运行时内存区域划分为五个部分，分别为方法区、堆、PC 寄存器、Java 方法栈和本地方法栈。Java 程序编译而成的 class 文件，需要先加载至方法区中，方能在 Java 虚拟机中运行。

为了提高运行效率，标准 JDK 中的 HotSpot 虚拟机采用的是一种混合执行的策略。

它会解释执行 Java 字节码，然后将其中反复执行的热点代码，以方法为单位进行即时编译，翻译成机器码后直接运行在底层硬件之上。


HotSpot 装载了多个不同的即时编译器，以便在编译时间和生成代码的执行效率之间做取舍。

下面我给你留一个小作业，通过观察两个条件判断语句的运行结果，来思考 Java 语言和 Java 虚拟机看待 boolean 类型的方式是否不同。



下载 `asmtools.jar` [2] , 并在命令行中运行下述指令 (不包含提示符 `$`) :

```
1 $ echo '  
2 public class Foo {  
3     public static void main(String[] args) {  
4         boolean flag = true;  
5         if (flag) System.out.println("Hello, Java!");  
6         if (flag == true) System.out.println("Hello, JVM!");  
7     }  
8 }' > Foo.java  
9 $ javac Foo.java  
10 $ java Foo  
11 $ java -cp /path/to/asmtools.jar org.openjdk.asmtools.jdis.Main Foo.class > Foo.jasm.1  
12 $ awk 'NR==1,/iconst_1/{sub(/iconst_1/, "iconst_2")} 1' Foo.jasm.1 > Foo.jasm  
13 $ java -cp /path/to/asmtools.jar org.openjdk.asmtools.jasm.Main Foo.jasm  
14 $ java Foo  
15
```

 复制代码

[1]: [https://en.wikipedia.org/wiki/Java\\_processor](https://en.wikipedia.org/wiki/Java_processor)

[2]: <https://wiki.openjdk.java.net/display/CodeTools/asmtools>

 极客时间

# 深入拆解Java虚拟机

Oracle 高级研究员 手把手带你入门 JVM



郑雨迪 Oracle Labs高级研究员，计算机博士

新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

版权归极客邦科技所有，未经许可不得转载

写留言

精选留言



小名叫大明

受益匪浅，多谢老师。

👍 11

请教老师一个问题，网上我没有搜到。

服务器线程数爆满，使用jstack打印线程堆栈信息，想知道是哪类线程数太多，但是堆栈里全是

一样的信息且没有任何关键信息，是哪个方法创建的，以及哪个线程池的都看不到。

如何更改打印线程堆栈信息的代码（动态）让其打印线程池信息呢？

2018-07-26



jiaobuchong

👍 2

对老师写的那段 awk 不懂得可参考：

<https://blog.csdn.net/jiaobuchong/article/details/83037467>

2018-10-14



东方

👍 164

jvm把boolean当做int来处理

```
flag = iconst_1 = true
```

awk把stackframe中的flag改为iconst\_2

if (flag) 比较时ifeq指令做是否为零判断，常数2仍为true，打印输出

if (true == flag) 比较时if\_cmpne做整数比较，iconst\_1是否等于flag，比较失败，不再打印输出

2018-07-20

作者回复

字节码高手！

2018-07-20



かっこすぎる郑一凡

👍 43

我想问下，JVM是怎么区别出热点代码和非热点代码的？

2018-07-20



novembersky

👍 41

文中提到虚拟机会把部分热点代码编译成机器码，我有个疑问，为什么不把java代码全部编译成机器码？很多服务端应用发布频率不会太频繁，但是对运行时的性能和吞吐量要求较高。如果发布或启动时多花点时间编译，能够带来运行时的持久性能收益，不是很合适么？

2018-07-20

| 作者回复

问得好！事实上JVM确实有考虑做AOT (ahead of time compilation) 这种事情。AOT能够在线下将Java字节码编译成机器码，主要是用来解决启动性能不好的问题。

对于这种发布频率不频繁(也就是长时间运行吧？)的程序，其实选择线下编译和即时编译都一样，因为至多一两个小时后该即时编译的都已经编译完成了。另外，即时编译器因为有程序的运行时信息，优化效果更好，也就是说峰值性能更好。

2018-07-20



醉人

👍 38

解释执行 执行时才翻译成机器指令，无需保存不占内存。但即时编译类似预编译，编译之后的指令需要保存在内存中，这种方式吃内存，按照二八原则这种混合模式最恰当的，热点代码编译之后放入内存避免重复编译，而其他运行次数较少代码则解释执行，避免占用过多内存

2018-07-20



陈晨

👍 27

作业终于做出来~\(\ge\le)/~喜大普奔

asmtools下载地址：

<https://adopt-openjdk.ci.cloudbees.com/view/OpenJDK/job/asmtools/lastSuccessfulBuild/artifact/asmtools-6.0.tar.gz>;

先是在window环境里，awk不能使用，看<https://zh.wikipedia.org/wiki/Awk>，AWK是一种优良的文本处理工具，Linux及Unix环境中现有的功能最强大的数据处理引擎之一，于是转战Linux，

```
[root@localhost cqg]# javac Foo.java
```

```
[root@localhost cqg]# java Foo
```

```
Hello,Java
Hello,JVM
[root@localhost cq]# java -cp /cq/asmtools.jar org.openjdk.asmtools.jdis.Main Foo.class > Foo.jasm.1
[root@localhost cq]# ls
asmtools.jar Foo.class Foo.jasm.1 Foo.java
[root@localhost cq]# vi Foo.jasm.1
[root@localhost cq]# awk 'NR==1,/iconst_1/{sub(/iconst_1/,"iconst_2")}' 1' Foo.jasm.1 > Foo.jasm
[root@localhost cq]# java -cp /cq/asmtools.jar org.openjdk.asmtools.jasm.Main Foo.jasm
asm
[root@localhost cq]# java Foo
Hello,Java
结果为啥是这个看点赞第一的高手;
另外asmtools使用方式还可以这样子:
java -jar asmtools.jar jdis Foo.class > Foo.jasm.1
java -jar asmtools.jar jasm Foo.jasm
2018-07-25
```



Ryan-Hou

👍 18

在为什么Java要在虚拟机里执行这一节您提到，java语法复杂，抽象度高，直接通过硬件来执行不现实，但是同样作为高级语言为什么C++就可以呢？这个理由作为引入虚拟机这个中间层的原因不是很充分吧

2018-07-20

作者回复

多谢指出！这里的直接运行指的是不经过任何转换(编译)，直接在硬件上跑。即便是C++，也不可以直接运行。

C++的策略是直接编译成目标架构的机器码，Java的策略是编译成一个虚拟架构的机器码。这个

虚拟架构可以有物理实现(可以搜Java processor), 也可以是软件实现, 也就是我们经常接触到的JRE。

2018-07-20



掌心童话

👍 14

我只想问, 你就没有教授的担忧? 万一我拿今天的知识点去问面试者, 答不上来咋办?

2018-07-21



godtrue

👍 13

1:为什么使用JVM?

1-1:可以轻松实现Java代码的跨平台执行

1-2:JVM提供了一个托管平台, 提供内存管理、垃圾回收、编译时动态校验等功能

1-3:使用JVM能够让我们的编程工作更轻松、高效节省公司成本, 提示社会化的整体快发效率, 我们只关注和业务相关的程序逻辑的编写, 其他业务无关但对于编程同样重要的事情交给JVM来处理

2:听完此节的课程的疑惑 (之前就没太明白, 原期待听完后不再疑惑的)

2-1:Java源代码怎么就经过编译变成了Java字节码?

2-2:JVM怎么就把Java字节码加载进JVM内了? 先加载那个类的字节码? 它怎么定位的? 拿到后怎么解析的? 不会整个文件放到一个地方吧? 使用的时候又是怎么找到的呢? 这些感觉还是黑盒

2-3:JVM将内存区域分成堆和栈, 然后将栈分成pc寄存器、本地方法栈、Java方法栈, 有些内存空间是线程可共享的, 有些是线程私有的。现在也了解不同的内存区块有不同的用处, 不过他们是怎么被划分的哪? 为什么是他们, 不能再多几种或少几种了吗? 共享的内存区和私有的又是怎么控制的哪?

2018-07-24

作者回复

总结得非常细致!

2-1 其实是这样的, JVM接收字节码, 要运行在JVM上只能选择转化为字节码。要是不想在JVM上跑, 可以选择直接转化为机器码。

2-2 类加载会在第三篇详细介绍。

2-3 具体的划分都是实现细节。你也可以选择全部冗杂在一起。但是这样子做性能较高，因为线程私有的可以不用同步。

2018-07-24



踏雪无痕

👍 8

您好，我现在所在的项目经常堆外内存占用非常多，超过总内存的70%，请问一下有没有什么方法能观察一下堆外内存有什么内容？

2018-07-20

作者回复

堆外内存的话，就把JVM当成普通进程来查找内存泄漏。可以看下Google Performance Tools 相关资料

2018-07-20



陈树义

👍 7

asmtools.jar 是在哪里下载的，怎么在给的链接页面没找到。

2018-07-21



欲风

👍 7

方法区和元空间是一个概念吧，能不能统一说法到jdk8之后的版本～

2018-07-20



周仕林

👍 6

看到有人说热点代码的区别，在git里面涉及到的热点代码有两种算法，基于采样的热点探测和基于计数器的热点探测。一般采用的都是基于计数器的热点探测，两者的优缺点百度一下就知道了。基于计数器的热点探测又有两个计数器，方法调用计数器，回边计数器，他们在C1和C2又有不同的阈值。😄😄

2018-07-23

作者回复

谢谢！



2018-07-24

不出腹肌  
不换头像

志文

👍 6

Java 作为一门高级程序语言，它的语法非常复杂，抽象程度非常高，所以不能直接在硬件上执行。所以要引入JAVA虚拟机。

我觉得理由不充分，JAVA为什么不能像c++一样直接转成机器码呢？从理论上是可以编译来实现这个的功能的。问题在于直接像c++那样编译成机器码，就实现不了跨平台了。那么是不是跨平台才是引入JAVA虚拟机的重要原因呢。请老师解答

2018-07-22



曾泽浩

👍 6

解释执行和即时执行这里听得有点蒙

2018-07-20



ace

👍 5

最后awk那段分析了下。希望分析正确 也对大家有帮助

- 1、NR==1,/iconst\_1/ 是用于匹配行 匹配第一行到第一个出现iconst\_1的行
  - 2、{}进行脚本执行。针对第一步中匹配的行执行内置的字符串函数sub 做替换
  - 3、1 都会被执行。在awk 1被计算为true，表现为默认操作 print \$0 也就是打印整行
- 整体效果是打印所以文本行，但第一个出现iconst\_1的做替换。

2018-08-05



那我懂你意思了

👍 5

老师，那个pc寄存器，本地方法栈，以及方法栈，java方法栈这三个组成的就是我们常统称的栈吧，然后也叫栈帧？

2018-07-20

| 作者回复

JVM里的栈指的应该是Java方法栈和本地方法栈。每个方法调用会在栈上划出一块作为栈帧(stack frame)。栈是由多个栈帧构成的，就好比电影是由一个个帧构成的。

2018-07-20



Kouichi

👍 4

为啥是"理论"上比cpp快...这样看起来 如果都编译成机器码了 应该就是挺快的呀... 那干啥不像Go一样 直接编译成目标平台的机器码... 咋感觉绕了一圈..

2018-07-20

作者回复

因为实际上会插入一些虚拟机相关的代码，稍微拉低了运行效率。

至于为什么不采用直接编译的方法，在峰值性能差不多的这个前提下，线下编译和即时编译就是两种选项，各有优缺点。JVM这样做，主要也是看重字节码的可移植性，而牺牲了启动性能。

另外呢，现代工程语言实现都是抄来抄去的。JVM也引入了AOT编译，在线下将Java代码编译成可链接库。

2018-07-20



临风

4

我跟楼上的novembersky同学一样疑惑，对于性能要求高的web应用，为什么不直接使用即时编译器在启动时全部编译成机器码呢？虽然启动耗时，但是也是可以接受的

2018-07-20

作者回复

通常，对于长时间运行的程序来说，大部分即时编译就发生在前几个小时。

再之后的即时编译主要是一些非热点代码，以及即时编译器中的bug造成的反复去优化重新编译。

2018-07-20



Fyypumpkin

4

老师，问一下这个asmtools是做什么用的

2018-07-20

作者回复

就是Java字节码的反汇编器和汇编器。

2018-07-20



xianhai

👍 3

即时编译生成的代码是只保存在内存中吗？会不会写到磁盘上？如果我怀疑优化后的代码有bug，有办法debug吗？

2018-07-21



祁颜·恩和

👍 3

java10的javafx能ppapi吗？我网上没查到该资料。告诉我呗

2018-07-20



jikelinj

👍 2

老师我看的疑问更多了，我看留言中也有一部分和我一样的疑惑。我使用jdk1.7和1.8都试过了，两个都可以打印，真心求一下困惑解答

2018-08-01



mingrui

👍 2

安卓ART虚拟机和其他Java虚拟机有什么区别

2018-07-21



Lilee

👍 2

你好，可以问一下虚拟机是怎么判断某一段代码是否为热点代码的吗？

2018-07-20



雾里听风

👍 2

理论上讲，即时编译后的 Java 程序的执行效率，是可能超过 C++ 程序的。

我们导师当时是这么解释的，c是所有CPU指令集的交集，而jit可以根据当前的CPU进行优化，调用交集之外的CPU指令集，往往这部分指令集效率很高。

作者如何看待这句话？

2018-07-20

作者回复

这句话不准确。现代编译器一般都分为平台无关的前端和平台相关的后端。如果你要生成某个平台的代码，编译器会选择相应的后端。因此，无论是C编译器还是JIT编译器，都是基于目标CPU的指令集来做优化的。

2018-07-20



abs

👍 2

我懂了，最后一个命令，又重新编译为class文件了

2018-07-20



abs

👍 2

请问下练习中生成了jasm文件，没有把更改写入Foo.class是怎么对结果造成影响的呢？

2018-07-20



mj4ever

👍 1

Java语言规范中，boolean类型的只有两种 true和false（false是默认值）；  
Java虚拟机规范中，boolean类型要转换为int类型，true => 1，false => 0；  
在编译而成的class文件中，除了字段和传入参数外，基本看不出boolean类型：  
比如，第一次时为：

```
public class Foo {  
    public Foo() {  
    }  
    public static void main(String[] var0) {  
        byte flag = 1;  
        if (flag != 0) {  
            System.out.println("Hello, Java!");  
        }  
        if (flag == 1) {  
            System.out.println("Hello, JVM!");  
        }  
    }  
}
```

第2次执行时，正则表达式将flag替换为2：

```
public class Foo {  
    public Foo() {  
    }  
    public static void main(String[] var0) {
```

```
byte flag = 2;  
if (flag != 0) {  
    System.out.println("Hello, Java!");  
}  
if (flag == 1) {  
    System.out.println("Hello, JVM!");  
}  
}  
}
```

2018-10-05



leohuachao

👍 1

java -cp /path/to/asmtools.jar org.openjdk.asmtools.jdis.Main Foo.class > Foo.jasm.1

可以简写为

java -jar /path/to/asmtools.jar jdis Foo.class > Foo.jasm.1

2018-09-19



G1

👍 1

元空间和永久代只是不同版本下的方法区实现，所以原文中用方法区称呼是正确的

2018-08-20



蒙奇•D•273°

👍 1

静态方法和实例方法都是如何分配的

2018-07-30



王刚

👍 1

老师我想问下，Java虚拟机指的就是jre吗？

如果不是Java虚拟机和jre有啥区别嘛？

2018-07-27



Hizkijah

👍 1

1楼说的第二个无法输出，我验证了可以输出啊。

2018-07-23



小奶狗

👍 1

老师，关于题目，为什么执行awk后，反编译Foo.class，反编译后的代码和Foo.java都不同了，表现在“初始化变量为2”了（所以改动后只会打印Hello,Java!），可我们没直接对Foo.class做改动啊。还有用javap -c Foo.class，得到的结果有一段为“Compiled from Foo.jasm”，我觉得应该是“Compiled from Foo.java”才对，到底awk这一步做了什么，好疑惑。

2018-07-22



字节码不懂 不会题目

👍 1

2018-07-22



韶华易逝~

👍 1

老师，hotspot分层编译，并使用额外编译线程。那么c1和c2是在什么时候进行编译的啊，应用程序启动时还是启动后的运行时啊？

2018-07-20



mover

👍 1

你好，在回答Ryan-hou时你的回答：这里的直接运行指的是不经过任何转换(编译)，直接在硬件上跑。即便是C++，也不可以直接运行。

这个地方可以展开说一下吗？c++不可以直接执行，why？c++不是编译成机器码了吗？JAVA像c++那样直接编译成目标机器的机器码为什么不现实？

2018-07-20



涛哥

👍 1

我用jdk8运行了下，两个都输出了呀，我javap看了下是两条ifeq，没有得出上面那个人说的下面没有输出这是为啥，不过我看到了虚拟机对boolean是看成整形

2018-07-20



Bert.zhu

👍 1

老师，对于jvm的即时编译，当方法里有很多if,elseif这样的判断，jvm也是整个方法进行编译，还是只部分编译？

2018-07-20

### 作者回复

JVM有两种编译方式，整个方法进行编译，或者对热循环进行编译。后面那种涉及到一个叫on stack replacement的技术。不论是那种，都要比if else的粒度大。

2018-07-20



追梦

方法区是不是属于堆的一部分？

2018-07-20

👍 1

### 作者回复

不属于。JVM中的堆是用来存放Java对象的。

2018-07-20



隔离样

初中还是高中时真的学的是c+

2018-07-20

👍 1

### 作者回复

哈哈，一般初高中应该使用工程性没那么强的语言吧？

2018-07-20

毕亮亮

有风来仪

👍 0

老师问个问题，java代码最终运行在jvm还是操作系统呢？如果是jvm，为啥还要转为机器码呢

2018-10-18



dream

👍 0

最后小作业中：awk 'NR==1,/iconst\_1/{sub(/iconst\_1/, "iconst\_2")} 1' Foo.jasm.1 > Foo.jasm 这个命令，我只知道awk 在linux中是类似于vim的与记事本相关的命令，但是中间到底做了什么一直不明白，经过查资料归纳如下，不对的希望多加指正：awk是linux下一个强大的文本分析工具 NR(Number of Record)：行号，当前处理的文本行的行号。sub(regex,sub,string)sub 函数执行一次子串替换。它将第一次出现的子串用 regex 替换。第三个参数是可选的，默认为 0。

0 完整的输入记录 n 当前记录的第n个字段，字段间由FS分隔



iconst\_1 为 pattern

/iconst\_1/ 两个/表示正则表达式的开始结束符号

最后那个1为定值, 非0

参考资料如下:

<https://blog.csdn.net/xp5xp6/article/details/50531396>

<http://www.runoob.com/linux/linux-comm-awk.html>

<http://www.runoob.com/w3cnote/awk-built-in-functions.html>

但是把第一个iconst\_1替换为iconst\_2有什么用呢? 有点不理解。

2018-10-11

作者回复

第二篇有讲解。boolean在字节码中被映射为int类型。这里我们在尝试整数2是true还是false, 还是it depends。

2018-10-12



dream

0

请问一下这段话是什么意思: \$ awk 'NR==1,/iconst\_1/{sub(/iconst\_1/, "iconst\_2")} 1' Foo.jasm.1 > Foo.jasm, 我知道awk是类似于vim的记事本工具, 但是这段代码到底做了什么不理解

2018-10-11

作者回复

你可以理解为把文件中第一次出现的iconst\_1改为iconst\_2

2018-10-12



chenfei

0

你好, 既然类都被加载到方法区了, 反射调用是否有可能直接访问方法区, 还是要必须访问类型类对象?

2018-10-01



maytwo

0

老师, 运行在x86的java程序, 怎样才能运行在arm架构

2018-10-01

## 作者回复

下AArch64版本的OpenJDK

2018-10-01



兰传科 安卓青铜

👍 0

第一种是解释执行，即逐条将字节码翻译成机器码并执行；第二种是即时编译（Just-In-Time compilation...）

JIT和ART的区别我没太理解，边翻译边执行不是JIT吗？

2018-09-27



大明

👍 0

对不起，听了29篇文章了，至今不太清楚hotspot和openjdk两者之间的关系。

2018-09-26

## 作者回复

HotSpot是JVM里的引擎，可以理解为JDK中用C++写的部分。Oracle JDK/OpenJDK包括Hot Spot。

2018-09-28



D→\_→M

👍 0

做作业的时候发现自己了解的真的是太少了，又去看了几篇java字节码和awk的文章才勉强把作业搞懂。看了一些awk的用法文章，了解了一些语法，但还不是很懂老师的那天awk命令，老师能给讲解一下吗？

2018-09-24

## 作者回复

awk就是一个文本处理修改工具，要用时查查man文档即可。

这一条awk指令相当于用记事本打开，搜索iconst1，替换为iconst2。你可以就在记事本这样的GUI程序里操作，也挺方便的。

2018-09-28



L

👍 0



既然解释执行的时候都已经将字节码翻译成机器码了，为什么还要再对热点代码进行即时编译呢？为什么不能在解释执行的同时将已经翻译成机器码的代码保存起来，下次执行的时候如果是已经翻译过的代码就直接执行机器码，如果是未翻译过的那就进行解释执行？

2018-09-18



suzuiyue

0

JIT程序重启之后还需要再来一遍吗？

2018-09-18

作者回复

程序关闭后，即时编译的结果就没了，因此需要再来一遍

2018-09-18



Phoenix

0

解释执行是将字节码翻译为机器码，JIT也是将字节码翻译为机器码，为什么JIT就比解释执行要快这么多？

如果说JIT检测到是热点代码并且进行优化，那么为什么解释执行不直接用这种优化去解释字节码？

一些比较浅的问题，希望老师能指点一二

2018-09-16

作者回复

1. 就单条加法字节码而言，解释执行器需要识别字节码，然后将两个操作数从Java方法栈上读取出来并相加，最后将结果存入Java方法栈中。而JIT生成的机器码就只是一个CPU加法指令。

2. 因为JIT比较费时。如果字节码需要JIT后才跑，那么启动性能会很糟糕

2018-09-17

rachel

0

java程序需要运行在虚拟机的原因？

我觉得主要是为了避免系统语言（例如C语言、C++等）与平台的强耦合性，不易移植的缺点，从而实现“一次编写，到处运行”的平台无关性。

2018-09-13



%E6%9D%8E%E6%9D%83%E6%...

👍 0

即时编译器收集信息会不会产生额外的开销？这个开销会不会随着单位时间内访问次数的增大而增大？Java的动态编译和C++的静态编译都是转换成机器码，Java的优势在哪，运行期间的优化方案有哪些？

阅读完后我还有这些疑问，麻烦老师解答一下😊

2018-09-05

作者回复

可以直接看第16 17篇，你的问题基本上都覆盖到了

2018-09-07



dingwood

👍 0

“栈帧存储局部变量以及字节码的操作数，栈帧的大小是提前计算好的”，郑老师，这段话中字节码的操作数是指什么？不明白这个概念。另外 栈帧的大小 在哪可以看到，在linux 用ulimit -a，显示的stack大小即为栈帧大小？两个问题，盼复，谢谢！

2018-08-22



kernel

👍 0

评论比文章精彩，学习到更多

2018-08-13



D.Onlyone

👍 0

看来我还是太菜，能看懂，就是作业完成不了，😭，还是看的评论才弄好，为啥我就看不懂这些字节码？求老师给推荐个书。

2018-08-13



蓝白之间

👍 0

C++如何运行在不同操作系统的呢？比如Linux 和windows

2018-08-11

午夜的汽笛

👍 0

老师好，Java虚拟机在执行时怎么定义热点方法和热点代码的？

2018-08-06



三分热狗

思路清晰 通俗易懂 受益匪浅

2018-08-05

👍 0

hacker time

解释器的程序能讲解一下吗?

2018-08-05

👍 0

三思

PC寄存器和栈什么关系啊，一直对这儿不太理解

2018-08-05

👍 0



网络已断开

iconst\_2;

istore\_1;

iload\_1;

ifeq L14;

getstatic Field java/lang/System.out:"Ljava/io/PrintStream;"

ldc String "Hello,Java!"

invokevirtual Method java/io/PrintStream.println:"(Ljava/lang/String;)V"

L14: stack\_frame\_type append;

locals\_map int;

iload\_1;

iconst\_1;

if\_icmpne L27;

getstatic Field java/lang/System.out:"Ljava/io/PrintStream;"

ldc String "Hello,JVM!"

invokevirtual Method java/io/PrintStream.println:"(Ljava/lang/String;)V"

L27: stack\_frame\_type same;

return;

👍 0

line1:入栈2

line2:栈顶int存入第二个局部变量(第一个存放的是this)

line3:局部变中量表第二个局部变量入栈

line4:判断栈顶是否为零(为2不为0)

line10:局部变中量表第二个局部变量入栈

line11:入栈1

line12:比较栈顶是否相等(1和2自然是不等的)

2018-08-02

张皮皮

👍 0

听之后又重新了解了一些，之前看过，不过忘了，作业暂时还没做

2018-08-01



jikelinj

👍 0

这个平台还不够完善，只能给作者留言。希望老师一定要看到留言给解答一下吧

2018-08-01



jikelinj

👍 0

老师，我的demo怎么两个都可以打印输出？

hello java 和hello jvm都可以，但是我看到留言中有位高手的详解中说第二个不打印，有点郁闷

2018-08-01

**作者回复**

在汇编后，也就是调用jasm.Main后，用javap查看一下Foo类的字节码，看有没有iconst\_2

2018-08-02



helloworld

👍 0

终于把JVM中的解释执行和JIT及时编译分清楚了！讲得浅显易懂！

2018-07-31



helloworld

👍 0

终于把JVM中的解释执行和JIT及时编译分清楚了！讲得浅显易懂！

2018-07-31

Bale

 0

老实.....原谅我做作业慢了

我这边windows是可以做作业的，首先把.gz里的.zip打开然后进入asmtool6.0/lib中，把asmtools.jar解压到指定位置，然后将作业代码，asmtools的路径对应修改即可。

原理请看评论第一高手

2018-07-30

迈克播

 0

想请问老师:静态常量池和方法区是怎样的关系呢？Java类加载到方法区，和类中的静态常量和静态块加载的顺序是怎样的呢？希望老师看到能够解答一下。

2018-07-29

李帆

 0

我按作业执行最后还是都打印了。我原来Foo.class的字节码是0:iconst\_1，Foo.jasm.1的字节码是iconst\_1，Foo.jasm的字节码变成了iconst\_2，运行java cp asmtools.jar org.openjdk.asmttools.jasm.Main Foo.jasm后Foo.class字节码变成了0:iconst\_2，我想知道wsm没有成功，我用的是jdk1.8，awk版本是20070501，asmtools.jar是通过hg clone拉下来有ant编译的

2018-07-28

jimforcode

 0

老师，Java 虚拟机会在当前线程的 Java 方法栈中生成一个栈帧，用以存放局部变量以及字节码的操作数。这个局部变量应该是一个引用吧，真正的值是存储在堆里面。这个帧栈是提前计算好的，那具体有多大，对变量有什么限制吗？

2018-07-28

jimforcode

 0

老师，Java 虚拟机会在当前线程的 Java 方法栈中生成一个栈帧，用以存放局部变量以及字节码的操作数。我想知道这个局部变量可以有多大，如果是个大对象，比如一个含有几百兆的字节数组的对象，比如文件数据，是不是对系统的性能有很大影响？

2018-07-28

Cc养鱼人

 0

非常好，简洁，打击准确。对于我意愿深入理解Android 虚拟机提供了学习指导。继续学习



2018-07-27

咖啡猫口里的咖啡猫🐱

👍 0

老师怎么好编译jdk8，或以后的，，两天了，能不能介绍的文章出处啊😁，我还是喜欢自己有代码调试的感觉

2018-07-24

天亮了

👍 0

老师请问下，这工具在那里下载？并且这工具怎么使用啊。小白表示这样不懂。。。

2018-07-24

笨笨蛋

👍 0

搞不懂，没有讲清楚堆栈到底如何共享？有些文章说栈数据共享，但又说每个线程都会有一个堆栈，那堆栈的数据还如何共享？还有堆有时候说数据不共享，但又说线程间数据共享？这老师能解答一下吗？

2018-07-24

| 作者回复

线程各自的栈空间是不共享的，但可以通过堆空间来共享数据。如果只有一个线程知道某个数据存放在堆的哪个位置，那也相当于不共享。注意不是等同于不共享，因为其它线程可以扫描整个堆，来找到这个位置。

2018-07-24

Struggling

👍 0

我一直以为方法区是从堆划出的一部分，用来存放类和静态信息等，方法区中又包含常量池😁

2018-07-24

| 作者回复

这种理解也没啥问题，都是概念上的。说不定哪天我司里的架构师就说方法区属于堆了

2018-07-24

笨笨蛋

👍 0

什么时候使用C1，什么时候使用C2，他是怎么区分热点方法的呢？

2018-07-24

| 作者回复

刚刚看到一个同学总结了。JVM会统计每个方法被调用了多少次，超过多少次，那就是热点方法。(还有个循环回边计数器，用来编译热循环的。)

默认的分层编译应该是达到两千调C1，达到一万五调C2。

2018-07-24

lubibo

👍 0

想问一下java方法栈和本地方法栈有什么区别

2018-07-23

王旭林

👍 0

先编译为Java bytecode

运行时 即时编译 分C1. C2, G. 多种可能

先C1 再 C2

java9有了aot解决启动时间问题

asmtools jasm/jdis

2018-07-23

bosiam

👍 0

对于占据大部分的不常用的代码，我们无需耗费时间将其编译成机器码，而是采取解释执行的方式运行；另一方面，对于仅占据小部分的热点代码，我们则可以将其编译成机器码，以达到理想的运行速度。

这句话有个疑问：

解释执行的方式运行也是需要编译成机器码的对吧，只不过是逐条编译？

2018-07-23

Desperado

 0

希望案列在手

2018-07-23

性能

 0

老师，即时编译是啥算法？编译哪些代码？何时编译完成？为啥我每次压测启动后，top命令查看，同样的代码编译线程工作时长不太一样？

2018-07-23

| 作者回复

即时编译就是一个编译器，里面有很多不同的优化，对应不同的算法。触发即时编译用的是JVM维护的统计方法调用次数的计数器。编译时间取决于编译器自己的效率。由于程序的不确定性，在多线程环境下即时编译器干的活可能多可能少。

2018-07-24

ybin

 0

想问下，方法中的局部变量是在方法结束时出栈回收，那方法中的循环或是if块中的变量是在循环或是if结束后被出栈被回收呢，还是在方法结束后再回收呢，毕竟if中定义的变量还是不能被if外的代码使用？

2018-07-23

李子

 0

听上去，栈溢出除了跟方法调用层级有关，也跟方法的局部变量多少大小有关。

有些性能调优会让把线程栈内存调低一点，这是为什么？只是为了能有更多的线程么？

2018-07-23

秋

 0

>> Java 作为一门高级程序语言，它的语法非常复杂，抽象程度也很高..直接在硬件上运行不现实

c++的语法和抽象程度不比Java低，但是在本地跑，感觉这句话不是很准确，逻辑可以调整下

2018-07-23

孙歌

 0

老师，您好！想问一下方法区存储Java类是采用什么结构形式来存储的呢，还有方法区的Java类会不会被垃圾回收呢？

2018-07-22

和风暖林

 0

写得很好呀，期待后面的内容

2018-07-22



追梦

 0

请问Jdk8里面的方法区也不属于堆的吗？

2018-07-22

xzchaoo

 0

对于小作业，为何编译器对这两种if产生了不一样的字节码呢？就这个例子而言，两者应该是等价的。

2018-07-22

Andy

 0

很不错，解决了我许多疑问！大概多久更新一集，希望每天更新一集

2018-07-22

laoyaozhang

 0

有一点看了好几个相关文章都没提到，即时编译生成的机器码一般是存到哪里的？是内存吗？如果是内存的话，是在给JvM分配的内存呢？还是再单独开辟一块内存？

2018-07-22

杨春鹏

 0

老师，解释执行是逐行将字节码编译成机器码。如何理解为这里的“逐行”。是所谓的一行代码吗，也就是按照分号来划分？

2018-07-22

小菜鸟

 0

老师好！问下！何时将字节码加载到jvm呢

2018-07-21

小菜鸟

👍 0

你好！有个疑问问下哈！

什么时候会将字节码加载到jvm的方法区呢？

另外您说的编译是指什么？当java文件变为class文件的时候不就是已经编译过了吗？这个时候在jvm中存放的不就是class文件了吗

2018-07-21

Lisa Li

👍 0

我可以理解为class文件先被解释执行，等收集到足够的信息之后，C1会对其中的热点代码进行优化（方法为单位），C2又会对C1中的热点代码(等收集到足够信息)进行优化（方法为单位).对吗？

2018-07-21

天敌

👍 0

老师，我的代码报错 java.lang.UnsupportedClassVersionError, 请问是因为我使用jdk1.7的原因吗？如果是，请问我需要采用什么版本的jdk？

2018-07-21

边风

👍 0

老师你好，看了你的讲解之后终于明白为什么服务刚启动的时候响应速度比较慢，是因为这个时候还没有即时编译完成，请求过一段时间之后即时编译差不多完成性能就有了很大提升。那么除了这个原因还有别的原因导致刚启动的服务比较慢么？

2018-07-21



xianhai

👍 0

运行asmtools那一步时出错。Foo.class: 1 Error invalid character in input

2018-07-21

Bin

👍 0

hotspot长时间运行后，字节码都会变成机器码吗？还是说有一个比例或者虚拟机只即时编译热点代码，那么虚拟机以什么样的标准去判断热点代码呢？

2018-07-21



立刀化白

👍



酷平勿迎~

0

老师，上述中提到即时编译有c1和c2，主要应对client端型程序和server端长时间运行型程序。7版本的hotspot的又使用额外线程进行分层编译。我迷惑的是c1和c2是什么时候编译，应用启动还是启动后的运行时？

2018-07-20

kmax

0

开门红，期待后面有一些具体案例实践分析

2018-07-20

程序设计的艺术

0

在方法内实例化了很多个对象，调用其方法，会不会因为对象过多引起堆内存泄漏或者溢出？谢谢

2018-07-20

程序设计的艺术

0

我现在在后台应用中new了很多对象，都是方法内实例化的，如果大并发的话，会不会有问题？

2018-07-20



mover

0

理论上讲，即时编译后的 Java 程序的执行效率，是可能超过 C++ 程序的。

你好，理论上讲，可能超过，那么实际上呢，是什么情况？从统计上来看，实现同样的功能，JAVA程序比c++慢多少？慢一倍？10%？

2018-07-20

Nic-愛

0

关于热点代码通过C2编译器编译，那怎么检测热点代码呢？其次即时编译在一些场景会逆优化退回解释执行，那一般在哪些场景会逆优化呢？

2018-07-20

Axis

0

我希望能深度介绍下c2编译器 尤其是sea of node 谢谢

2018-07-20

Axis

👍 0

我希望能介绍一下c2 特别是sea of node的相关论文

2018-07-20

流浪的猫

👍 0

java的即时编译会编译出文件来？还是在内存中的？

2018-07-20

kyrano

👍 0

“Java 虚拟机不要求栈帧在内存空间里连续分布” 能说一下底层是如何实现不连续的吗？

2018-07-20

包子

👍 0

不全部jit编译的原因，个人觉得是由于边运行边编译可以采集运行时信息用于优化编译。还有其

他原因么

2018-07-20