第1讲 | 谈谈你对Java平台的理解？

2018-05-05 杨晓峰

第1讲 | 谈谈你对Java平台的理解？
朗读人：黄洲君　08′03″ | 3.69M

从你接触 Java 开发到现在，你对 Java 最直观的印象是什么呢？是它宣传的 “Write once, run anywhere” ，还是目前看已经有些过于形式主义的语法呢？你对于 Java 平台到底了解到什么程度？请你先停下来总结思考一下。

今天我要问你的问题是，谈谈你对 Java 平台的理解？“Java 是解释执行” ，这句话正确吗？

典型回答

Java 本身是一种面向对象的语言，最显著的特性有两个方面，一是所谓的“书写一次，到处运行”（Write once, run anywhere），能够非常容易地获得跨平台能力；另外就是垃圾收集（GC, Garbage Collection），Java 通过垃圾收集器（Garbage Collector）回收分配内存，大部分情况下，程序员不需要自己操心内存的分配和回收。

我们日常会接触到 JRE（Java Runtime Environment）或者 JDK（Java Development Kit）。JRE，也就是 Java 运行环境，包含了 JVM 和 Java 类库，以及一些模块等。而 JDK 可以看作是 JRE 的一个超集，提供了更多工具，比如编译器、各种诊断工具等。

对于“Java 是解释执行”这句话，这个说法不太准确。我们开发的 Java 的源代码，首先通过 Javac 编译成为字节码（bytecode），然后，在运行时，通过 Java 虚拟机（JVM）内嵌的解释器将字节码转换成为最终的机器码。但是常见的 JVM，比如我们大多数情况使用的 Oracle JDK 提供的 Hotspot JVM，都提供了 JIT（Just-In-Time）编译器，也就是通常所说的动态编译器，JIT 能够在运行时将热点代码编译成机器码，这种情况下部分热点代码就属于编译执行，而不是解释执行了。

考点分析

其实这个问题，问得有点笼统。题目本身是非常开放的，往往考察的是多个方面，比如，基础知识理解是否很清楚；是否掌握 Java 平台主要模块和运行原理等。很多面试者会在这种问题上吃亏，稍微紧张了一下，不知道从何说起，就给出个很简略的回答。

对于这类笼统的问题，你需要尽量表现出自己的思维深入并系统化，Java 知识理解得也比较全面，一定要避免让面试官觉得你是个“知其然不知其所以然”的人。毕竟明白基本组成和机制，是日常工作中进行问题诊断或者性能调优等很多事情的基础，相信没有招聘方会不喜欢“热爱学习和思考”的面试者。

即使感觉自己的回答不是非常完善，也不用担心。我个人觉得这种笼统的问题，有时候回答得稍微片面也很正常，大多数有经验的面试官，不会因为一道题就对面试者轻易地下结论。通常会尽量引导面试者，把他的真实水平展现出来，这种问题就是做个开场热身，面试官经常会根据你的回答扩展相关问题。

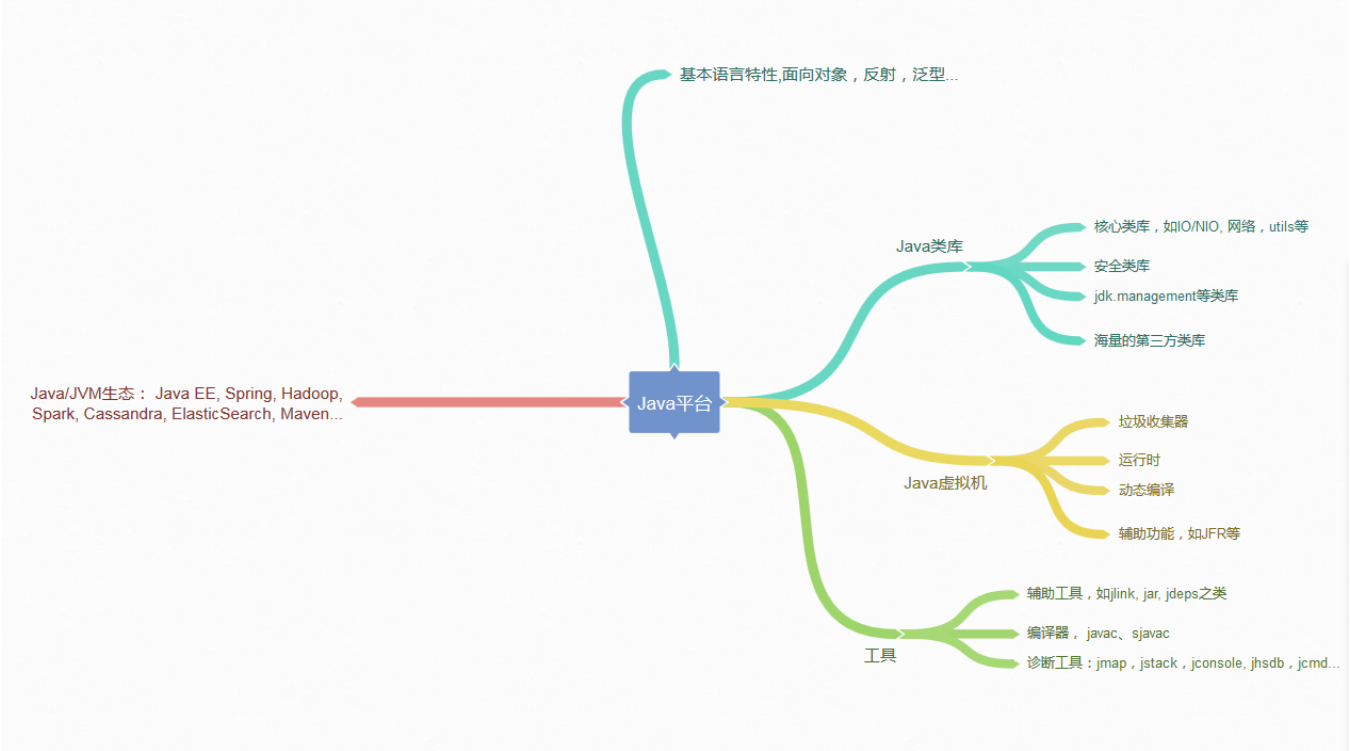
知识扩展

回归正题，对于 Java 平台的理解，可以从很多方面简明扼要地谈一下，例如：Java 语言特性，包括泛型、Lambda 等语言特性；基础类库，包括集合、IO/NIO、网络、并发、安全等基础类库。对于我们日常工作应用较多的类库，面试前可以系统化总结一下，有助于临场发挥。

或者谈谈 JVM 的一些基础概念和机制，比如 Java 的类加载机制，常用版本 JDK（如 JDK 8）内嵌的 Class-Loader，例如 Bootstrap、 Application 和 Extension Class-loader；类加载大致过程：加载、验证、链接、初始化（这里参考了周志明的《深入理解 Java 虚拟机》，非常棒的 JVM 上手书籍）；自定义 Class-Loader 等。还有垃圾收集的基本原理，最常见的垃圾收集器，如 SerialGC、Parallel GC、 CMS、 G1 等，对于适用于什么样的工作负载最好也心里有数。这些都是可以扩展开的领域，我会在后面的专栏对此进行更系统的介绍。

当然还有 JDK 包含哪些工具或者 Java 领域内其他工具等，如编译器、运行时环境、安全工具、诊断和监控工具等。这些基本工具是日常工作效率的保证，对于我们工作在其他语言平台上，同样有所帮助，很多都是触类旁通的。

下图是我总结的一个相对宽泛的蓝图供你参考。



不再扩展了，回到前面问到的解释执行和编译执行的问题。有些面试官喜欢在特定问题上“刨根问底儿”，因为这是进一步了解面试者对知识掌握程度的有效方法，我稍微深入探讨一下。

众所周知，我们通常把 Java 分为编译期和运行时。这里说的 Java 的编译和 C/C++ 是有着不同的意义的，Javac 的编译，编译 Java 源码生成“.class”文件里面实际是字节码，而不是可以直接执行的机器码。Java 通过字节码和 Java 虚拟机（JVM）这种跨平台的抽象，屏蔽了操作系统和硬件的细节，这也是实现“一次编译，到处执行”的基础。

在运行时，JVM 会通过类加载器（Class-Loader）加载字节码，解释或者编译执行。就像我前面提到的，主流 Java 版本中，如 JDK 8 实际是解释和编译混合的一种模式，即所谓的混合模式（-Xmixed）。通常运行在 server 模式的 JVM，会进行上万次调用以收集足够的信息进行高效的编译，client 模式这个门限是 1500 次。Oracle Hotspot JVM 内置了两个不同的 JIT compiler，C1 对应前面说的 client 模式，适用于对于启动速度敏感的应用，比如普通 Java 桌面应用；C2 对应 server 模式，它的优化是为长时间运行的服务器端应用设计的。默认是采用所谓的分层编译（TieredCompilation）。这里不再展开更多 JIT 的细节，没必要一下子就钻进去，我会在后面介绍分层编译的内容。

Java 虚拟机启动时，可以指定不同的参数对运行模式进行选择。 比如，指定“-Xint”，就是告诉 JVM 只进行解释执行，不对代码进行编译，这种模式抛弃了 JIT 可能带来的性能优势。毕竟解释器（interpreter）是逐条读入，逐条解释运行的。与其相对应的，还有一个“-Xcomp”参数，这是告诉 JVM 关闭解释器，不要进行解释执行，或者叫作最大优化级别。那你可能会问这种模式是不是最高效啊？简单说，还真未必。 “-Xcomp”会导致 JVM 启动变慢非常多，同时有些 JIT 编译器优化方式，比如分支预测，如果不进行 profiling，往往并不能进行有效优化。

除了我们日常最常见的 Java 使用模式，其实还有一种新的编译方式，即所谓的 AOT（Ahead-of-Time Compilation），直接将字节码编译成机器代码，这样就避免了 JIT 预热等各方面的开销，比如 Oracle JDK 9 就引入了实验性的 AOT 特性，并且增加了新的 jaotc 工具。利用下面的命令把某个类或者某个模块编译成为 AOT 库。

```
jaotc --output libHelloWorld.so HelloWorld.class
jaotc --output libjava.base.so --module java.base
```

然后，在启动时直接指定就可以了。

```
java -XX:AOTLibrary=./libHelloWorld.so,./libjava.base.so HelloWorld
```

而且，Oracle JDK 支持分层编译和 AOT 协作使用，这两者并不是二选一的关系。如果你有兴趣，可以参考相关文档：http://openjdk.java.net/jeps/295。AOT 也不仅仅是只有这一种方式，业界早就有第三方工具（如 GCJ、Excelsior JET）提供相关功能。

另外，JVM 作为一个强大的平台，不仅仅只有 Java 语言可以运行在 JVM 上，本质上合规的字节码都可以运行，Java 语言自身也为此提供了便利，我们可以看到类似 Clojure、Scala、Groovy、JRuby、Jython 等大量 JVM 语言，活跃在不同的场景。

今天，我简单介绍了一下 Java 平台相关的一些内容，目的是提纲挈领地构建一个整体的印象，包括 Java 语言特性、 核心类库与常用第三方类库、Java 虚拟机基本原理和相关工具，希望对你有所帮助。

一课一练

关于今天我们讨论的题目你做到心中有数了吗？知道不如做到，请你也在留言区写写自己对 Java 平台的理解。我会选出经过认真思考的留言，送给你一份学习鼓励金，欢迎你与我一起讨论。

你的朋友是不是也在准备面试呢？你可以“请朋友读”，把今天的题目分享给好友，或许你能帮到他。

精选留言

Woj 181

“一次编译、到处运行”说的是Java语言跨平台的特性，Java的跨平台特性与Java虚拟机的存在密不可分，可在不同的环境中运行。比如说Windows平台和Linux平台都有相应的JDK，安装好JDK后也就有了Java语言的运行环境。其实Java语言本身与其他的编程语言没有特别大的差异，并不是说Java语言可以跨平台，而是在不同的平台都有可以让Java语言运行的环境而已，所以才有了Java一次编译，到处运行这样的效果。

严格的讲，跨平台的语言不止Java一种，但Java是较为成熟的一种。“一次编译，到处运行”这种效果跟编译器有关。编程语言的处理需要编译器和解释器。Java虚拟机和DOS类似，相当于一个供程序运行的平台。

程序从源代码到运行的三个阶段：编码——编译——运行——调试。Java在编译阶段则体现了跨平台的特点。编译过程大概是这样的：首先是将Java源代码转化成.CLASS文件字节码，这是第一次编译。.class文件就是可以到处运行的文件。然后Java字节码会被转化为目标机器代码，这是是由JVM来执行的，即Java的第二次编译。

“到处运行”的关键和前提就是JVM。因为在第二次编译中JVM起着关键作用。在可以运行Java虚拟机的地方都内含着一个JVM操作系统。从而使JAVA提供了各种不同平台上的虚拟机制，因此实现了“到处运行”的效果。需要强调的一点是，java并不是编译机制，而是解释机制。Java字节码的设计充分考虑了JIT这一即时编译方式，可以将字节码直接转化成高性能的本地机器码，这同样是虚拟机的一个构成部分。

2018-05-05

作者回复

高手

2018-05-05

magict4 75

我对『Compile once, run anywhere』这个宣传语提出的历史背景非常感兴趣。这个宣传语似乎在暗示 C 语言有一个缺点：对于每一个不同的平台，源代码都要被编译一次。我不解的地方是，为什么这会是一个问题？不同的平台，可执行的机器码必然是不一样的。源代码自然需要依据不同的平台分别被编译。 我觉得真正问题不在编译这一块，而是在 C 语言源文件这一块。我没有 C 语言的编程经验，但是似乎 C 语言程序经常需要调用操作系统层面的 API。不同的操作系统，API 一般不同。为了支持多平台，C 语言程序的源文件需要根据不同平台修改多次。这应该是一个非常大的痛点。我回头查了一下当时的宣传语，原文是『Write once, run anywhere』，焦点似乎并不在编译上，而是在对源文件的修改上。

以上是自己一点不成熟的想法，还请大家指正！

2018-05-05

作者回复

汗颜，是我记错了，非常感谢指正

2018-05-05

三军 61

Java特性:
面向对象（封装，继承，多态）
平台无关性（JVM运行.class文件）
语言（泛型，Lambda）
类库（集合，并发，网络，IO/NIO）
JRE（Java运行环境，JVM，类库）
JDK（Java开发工具，包括JRE，javac，诊断工具）

Java是解析运行吗？
不正确！
1，Java源代码经过Javac编译成.class文件
2，.class文件经JVM解析或编译运行。
（1）解析:.class文件经过JVM内嵌的解析器解析执行。
（2）编译:存在JIT编译器（Just In Time Compile 即时编译器）把经常运行的代码作为"热点代码"编译与本地平台相关的机器码，并进行各种层次的优化。
（3）AOT编译器: Java 9提供的直接将所有代码编译成机器码执行。

2018-05-05

作者回复

精辟

2018-05-05

thinkers 46

jre为java提供了必要的运行时环境，jdk为java提供了必要的开发环境！

2018-05-05

作者回复

剧透一下，未来jre将退出历史舞台！

2018-05-05

Jerry银银 36

关注了好久，终于期盼到了第一讲。

在看到这个题目时，我并没有立马点进来看原文，而是给了自己一些时间进行思考。

首先，个人觉得这个题目非常的抽象和笼统，这个问题没有标准答案，但是有『好』答案，而答案的好坏，完全取决于面试者自身的技术素养和对Java系统性的了解。我的理解如下：

宏观角度：
跟c/c++最大的不同点在于，c/c++编程是面向操作系统的，需要开发者极大地关心不同操作系统之间的差异性；而Java平台通过虚拟机屏蔽了操作系统的底层细节，使得开发者无需过多地关心不同操作系统之间的差异性。
通过增加一个间接的中间层来进行”解耦“是计算机领域非常常用的一种”艺术手法“，虚拟机是这样，操作系统是这样，HTTP也是这样。

Java平台已经形成了一个生态系统，在这个生态系统中，有着诸多的研究领域和应用领域：
1. 虚拟机、编译技术的研究(例如：GC优化、JIT、AOT等)：对效率的追求是人类的另一个天性之一
2. Java语言本身的优化
3. 大数据处理
4. Java并发编程
5. 客户端开发（例如：Android平台）
6. ……

微观角度：
Java平台中有两大核心：
1. Java语言本身、JDK中所提供的核心类库和相关工具
2. Java虚拟机以及其他包含的GC

1. Java语言本身、JDK中所提供的核心类库和相关工具
从事Java平台的开发，掌握Java语言、核心类库以及相关工具是必须的，我觉得这是基础中的基础。
>> 对语言本身的了解，需要开发者非常熟悉语言的语法结构；而Java又是一种面对对象的语言，这又需要开发者深入了解面对对象的设计理念；
>> Java核心类库包含集合类、线程相关类、IO、NIO、J.U.C并发包等；
>> JDK提供的工具包含：基本的编译工具、虚拟机性能检测相关工具等。

2. Java虚拟机
Java语言具有跨平台的特性，也正是因为虚拟机的存在。Java源文件被编译成字节码，被虚拟机加载后执行。这里隐含的意思有两层：
1）大部分情况下，编程者只需要关心Java语言本身，而无需特意关心底层细节。包括对内存的分配和回收，也全权交给了GC。

2）对于虚拟机而言，只要是符合规范的字节码，它们都能被加载执行，当然，能正常运行的程序光满足这点是不行的，程序本身需要保证在运行时不出现异常。所以，Scala、Kotlin、Jython等语言也可以跑在虚拟机上。

围绕虚拟机的效率问题展开，将涉及到一些优化技术，例如：JIT、AOT。因为如果虚拟机加载字节码后，完全进行解释执行，这势必会影响执行效率。所以，对于这个运行环节，虚拟机会进行一些优化处理，例如JIT技术，会将某些运行特别频繁的代码编译成机器码。而AOT技术，是在运行前，通过工具直接将字节码转换为机器码。

2018-05-06

作者回复

2018-05-06

zaiweiwoaini 36

看评论也能学习知识。

2018-05-05

作者回复

搬个板凳，哈哈

2018-05-05

刻苦滴涛涛 32

我理解的java程序执行步骤:
首先javac编译器将源代码编译成字节码。
然后jvm类加载器加载字节码文件，然后通过解释器逐行解释执行，这种方式的执行速度相对会比较慢。有些方法和代码块是高频率调用的，也就是所谓的热点代码，所以引进jit技术，提前将这类字节码直接编译成本地机器码。这样类似于缓存技术，运行时再遇到这类代码直接可以执行，而不是先解释后执行。

2018-05-05

作者回复

不错，JIT是运行时编译

2018-05-05

欧阳田 28

1，JVM的内存模型，堆、栈、方法区；字节码的跨平台性；对象在JVM中的强引用，弱引用，软引用，虚引用，是否可用finalise方法救救它？；双亲委派进行类加载，什么是双亲呢？双亲就是多亲，一份文档由我加载，然后你也加载，这份文档在JVM中是一样的吗？；多态思想是Java需要最核心的概念，也是面向对象的行为的一个最好诠释；理解方法重载与重写在内存中的执行流程，怎么定位到这个具体方法的。2，发展流程，JDK5(重写bug)，JDK6(商用最稳定版)，JDK7(switch的字符串支持)，JDK8(函数式编程)，一直在发展进化。3，理解祖先类Object，它的行为是怎样与现实生活连接起来的。4，理解23种设计模式，因为它是道与术的结合体。

2018-05-05

作者回复

高手

2018-05-05

一叶追寻 17

对Java平台的理解，首先想到的是Java的一些特性，比如平台无关性、面向对象、GC机制等，然后会在这几个方面去回答。平台无关性依赖于JVM，将.class文件解释为适用于操作系统的机器码。面向对象则会从封装、继承、多态这些特性去解释，具体内容就不在评论里赘述了。另外Java的内存回收机制，则涉及到Java的内存结构，堆、栈、方法区等，然后围绕什么样的对象可以回收以及回收的执行。以上是我对本道题的理解，不足之处还请杨老师指出，希望通过这次学习能把Java系统的总结一下~

2018-05-05

作者回复

非常棒，不同语言对平台无关的支持是不同的，Java是最高等级，未来也许会在效率角度出发，进行某种折衷，比如AOT

2018-05-05

曹铮 14

这种基于运行分析，进行热点代码编译的设计，是因为绝大多数的程序都表现为“小部分的热点耗费了大多数的资源”吧。只有这样才能做到，在某些场景下，一个需要跑在运行时上的语言，可以比直接编译成机器码的语言更“快”

2018-05-05

作者回复

对，看到本质了

2018-05-05

姜亮 12

写个程序直接执行字节码就是解释执行。写个程序运行时把字节码动态翻译成机器码就是jit。写个程序把java源代码直接翻译为机器码就是aot。造个CPU直接执行字节码，字节码就是机器码。

2018-05-07

作者回复

好主意，当年确实有类似项目

2018-05-07

scott 8

解释执行和编译执行有何区别

2018-05-07

作者回复

类比一下，一个是同声传译，一个是放录音

2018-05-07

石头狮子 8

1. 一次编译，到处运行。jvm 层面封装了系统API，提供不同系统一致的调用行为。减少了为适配不同操作系统，不同架构的带来的工作量。
2. 垃圾回收，降低了开发过程中需要注意内存回收的难度。降低内存泄露出现的概率。虽然也带来了一些额外开销，但是足以弥补带来的好处。合理的分代策略，提高了内存使用率。
3. jit 与其他编译语言相比，降低了编译时间，因为大部分代码是运行时编译，避免了冷代码在编译时也参与编译的问题。
提高了代码的执行效率，之前项目中使用过 lua 进行相关开发。由于 lua 是解释性语言，并配合使用了 lua-jit。开发过程中遇到，如果编写的 lua 代码是 jit 所不支持的会导致代码性能与可编译的相比十分低下。

2018-05-05

作者回复

高手

2018-05-05

公众号:代码荣耀 5

今日文章心得:个人理解的Java平台技术体系包括了以下几个重要组成部分：
Java程序设计语言
各种硬件平台上的Java虚拟机
Class文件格式
Java API类库及相关工具
来自商业机构和开源社区第三方Java类库

可以把Java程序设计语言、Java虚拟机、Java API类库及相关工具，这三部分统称为JDK，JDK是用于支持Java程序开发的最小环境；可以把Java API类库中的Java SE API子集和Java虚拟机这两部分统称为JRE，JRE是支持Java程序运行的标准环境。

提起Java，必然会想起TA跨平台的特性，但是跨平台重要吗？重要！因为可以write once，run anywhere，这是程序员的终极梦想之一。但是跨平台重要吗？不重要！作为程序语言，会更加关注TA的生态、兼容性、安全性、稳定性，以及语言自身的与时俱进。要要理解Java平台，JVM是必须要迈过去的坎，将会看到另外的风景。

为什么我们就不能把JVM作为透明的存在呢？

勿在浮沙筑高台，以JVM的GC为例。既然Java等诸多高级程序语言都已经实现了自动化内存管理，那我们为什么还要去理解内存管理了？因为当我们需要排查各种内存溢出、泄漏等底层问题时，当垃圾收集成为我们开发的软件系统达到更高并发量、更高性能的瓶颈时，我们就需要对这些“自动化”技术实施必要的监控与调节优化。

2018-05-05

作者回复

对，深入有利于解决更多有难度的工作

2018-05-06

非常非常非常非常的普通中下 4

没有一个问题是加一个中间层解决不了的，如果解决不了就加两个

2018-05-08

吴有为 4

老师，您好，看了文章和大家的评论有点疑问，程序执行的时候，类加载器先把class文件加载到内存中，一般情况下是解释执行，解释器把class里的内容一行行解释为机器语言然后运行。疑问1.每次执行class文件都需要解释整个class文件吗？疑问2.当new了一个对象的时候是怎么解释这个类的，是解释整个这个类对应的class？疑问3.JIT编译的热点代码是指class文件还是class文件的部分内容？

2018-05-06

作者回复

我的理解不是以class为单位；JIT是方法级

2018-05-06

迎xiang李 4

Java新手表示学习了，java的运行机制算是看明白了，但是发现还是有很多词汇不太了解。只有看高手们的文章才能发现自己的短板和不熟悉的领悟。期待后面更精彩，全面深入的讲解。感谢作者和各位大神的精彩评论！

2018-05-05

作者回复

交流有利于提高

2018-05-06

佳人如玉巧弄心弦 4

关于JIT与AOT，我想KVM更有发言权，哈哈，好的东西终被学习与借鉴

2018-05-05

作者回复

行家

2018-05-05

凌 3

国富论中讲到，社会的分工细化起到了提高生产力的关键作用。我觉得一次编写到处运行也是社会分工的一种模式，他使大部分业务程序员注重领域模型的逻辑设计，不必关心底层的实现，使软件工程达到了专业的人做专业的事这一个高度。虽然现在掌握一门技术远远不够，但是对于大部分业务程序员来说，只有把精力花在最重要的地方比如领域模型的设计，才会让业务更加流畅完善。所以我觉得JVM机制蕴含了一定的经济学原理。

2018-05-08

墨川 3

老师讲的很精彩受教了，评论区好多高手。赶紧拿个小本本记下来

2018-05-06