周末福利 | 谈谈我对Java学习和面试的看法

2018-06-09 杨晓峰

周末福利 | 谈谈我对Java学习和面试的看法
朗读人：黄洲君　07′18″ | 3.35M

你好，我是杨晓峰。今天是周末，我们稍微放松一下来聊聊“Java 核心技术”之外的内容，正好也借这个机会，兑现一下送出学习奖励礼券的承诺。我在每一讲后面都留下了一道思考题，希望你通过学习，结合自身工作实际，能够认真思考一下这些问题，一方面起到检验学习效果的作用，另一方面可以查漏补缺，思考一下这些平时容易被忽略的面试考察点。我并没有给出这些思考题的答案，希望你通过专栏学习或者查阅其他资料进行独立思考，将自己思考的答案写在留言区与我和其他同学一起交流，这也是提升自己重要的方法之一。

截止到今天，专栏已经更新了 15 讲，走完了基础模块正式进入进阶模块。现在也正是一个很好的时机停下来回顾一下基础部分的知识，为后面进阶的并发内容打好基础。在这里，我也分享一下我对 Java 学习和面试的看法，希望对你有所帮助。

首先，有同学反馈说专栏有的内容看不懂。我在准备专栏文章的时候对一些同学的基础把握不太准确，后面的文章我进行了调整，将重点技术概念进行讲解，并为其他术语添加链接。

再来说说这种情况，有人总觉得 Java 基础知识都已经被讲烂了，还有什么可学的？

对于基础知识的掌握，有的同学经常是“知其然而不知其所以然”，看到几个名词听说过就以为自己掌握了，其实不然。至少，我认为应该能够做到将自己“掌握”的东西，准确地表达出来。

爱因斯坦曾经说过，“如果你不能把它简单地解释出来，那说明你还没有很好地理解它”。了解 - 掌握 - 精通，这是我们对事物掌握的一个循序渐进的过程。从自己觉得似乎懂了，到能够说明白，再到能够自然地运用它，甚至触类旁通，这是不断提高的过程。

在专栏学习中，如果有些术语很陌生，那么了解它就达到了学习目的，如果能够理解透彻达到掌握的程度当然更好。乐观点来看，反正都是有收获，也完全不必过分担心。

从学习技巧的角度，每个人都有自己的习惯，我个人喜欢动手实践以及与人进行交流。

- 动手实践是必要一步，如果连上手操作都不肯，你会发现自己的理解很难有深度。
- 在交流的过程中你会发现，很多似是而非的理解，竟然在试图组织语言的时候，突然就想明白了，而且别人的观点也验证了自己的判断。技术领域尤其如此，把自己的理解整理成文字，输出、交流是个非常好的提高方法，甚至我认为这是技术工作者成长的必经之路。

再来聊聊针对技术底层，我们是否有必要去阅读源代码？

阅读源代码当然是个好习惯，理解高质量的代码，对于提高我们自己的分析、设计等能力至关重要。

- 根据实践统计，工程师实际工作中，阅读代码的时间其实大大超过写代码的时间，这意味着阅读、总结能力，会直接影响我们的工作效率！这东西有没有捷径呢，也许吧，我的心得是：“无他，但手熟尔”。
- 参考别人的架构、实现，分析其历史上掉过的坑，这是天然的好材料，具体阅读时可以从其修正过的问题等角度入手。
- 现代软件工程，节奏越来越快，需求复杂而多变，越来越凸显出白盒方式的重要性。快速定位问题往往需要黑盒结合白盒能力，对内部一无所知，可能就没有思路。与此同时，通用平台、开源框架，不见得能够非常符合自己的业务需求，往往只有深入源代码层面进行定制或者自研，才能实现。我认为这也是软件工程师地位不断提高的原因之一。

那么，源代码需要理解到什么程度呢？对于底层技术，这个确实是比较有争议的问题，我个人并不觉得什么东西都要理解底层，懂当然好，但不能代表一切，毕竟知识和能力是有区别的，当然我们也要尊重面试官的要求。我个人认为，不是所有做 Java 开发的人，都需要读 JVM 源代码，虽然我在专栏中提供了一些底层源代码解读，但也只是希望真的有兴趣、有需要的工程师跟进学习。对于大多数开发人员，了解一些源代码，至少不会在面试问到的时候完全没有准备。

关于阅读源代码和理解底层，我有些建议：

- 带着问题和明确目的去阅读，比如，以 debug 某个问题的角度，结合实践去验证，让自己能够感到收获，既加深理解，也有实际帮助，激励我们坚持下来。
- 一定要有输出，至少要写下来，整理心得，交流、验证、提高。这和我们日常工作是类似的，千万不要做了好长一段时间后和领导说，没什么结论。

大家大都是工程师，不是科学家，软件开发中需要分清表象、行为（behavior），还是约定（specification）。喜欢源代码、底层是好的，但是一定要区分其到底是实现细节，还是规范的承诺，因为如果我们的程序依赖于表现，很有可能带来未来维护的问题。

我前面提到了白盒方式的重要性，但是，需要慎重决定对内部的依赖，分清是 Hack 还是 Solution。出来混，总是要还的！如果以某种 hack 方式解决问题，临时性的当然可以，长久会积累并成为升级的障碍，甚至堆积起来愈演愈烈。比如说，我在实验 Cassandra 的时候，发现它在并发部分引用了 Unsafe.monitorEnter()/moniterExit()，这会导致它无法平滑运行在新版的 JDK 上，因为相应内部 API 被移除了，比较幸运的是这个东西有公共 API 可以替代。

最后谈谈我在面试时会看中候选人的哪些素质和能力。

结合我在实际工作中的切身体会，面试时有几个方面我会特别在乎：

- 技术素养好，能够进行深度思考，而不是跳脱地夸夸其谈，所以我喜欢问人家最擅长的东西，如果在最擅长的领域尚且不能仔细思考，怎么能保证在下一份工作中踏实研究呢。当然这种思考，并不是说非要死扣底层和细节，能够看出业务中平凡事情背后的工程意义，同样是不错的。毕竟，除了特别的岗位，大多数任务，如果有良好的技术素养和工作热情，再配合一定经验，基本也就能够保证胜任了。
- 职业精神，是否表现出认真对待每一个任务。我们是职场打拼的专业人士，不是幼儿园被呵护的小朋友，如果有人太挑活儿，团队往往就无法做到基本的公平。有经验的管理角色，大多是把自己的管理精力用在团队的正面建设，而不是把精力浪费在拖团队后腿的人身上，难以协作的人，没有人会喜欢。有人说你的职业高度取决于你“填坑”的能力，我觉得很有道理。现实工作中很少有理想化的完美任务，既目标清晰又有挑战，恰好还是我擅长，这种任务不多见。能够主动地从不清晰中找出清晰，切实地解决问题，是非常重要的能力。
- 是否 hands-on，是否主动。我一般不要求当前需要的方面一定是很 hands-on，但至少要表现出能够做到。

下面放出中奖名单和精选留言，送出 15 元学习奖励礼券，希望我的《Java 核心技术 36 讲》不仅能带你走进大厂 Java 面试场景，还能帮你温故知新基础知识，构建你的 Java 知识体系。也欢迎你在这里与我交流面试、学习方面的困惑或心得，一起畅所欲言、共同进步。


极客时间
重拾极客精神·提升技术认知
中奖名单
昵称
奖励
石头狮子
15元无门槛学习奖励券
Woj
15元无门槛学习奖励券
kursk.ye
15元无门槛学习奖励券
Miaozhe
15元无门槛学习奖励券
肖一林
15元无门槛学习奖励券
曹铮
15元无门槛学习奖励券
雷霹雳的爸爸
15元无门槛学习奖励券
vash_ace
15元无门槛学习奖励券
Walter
15元无门槛学习奖励券
I am a psycho
15元无门槛学习奖励券


magict4
15元无门槛学习奖励券
李林
15元无门槛学习奖励券
Woong
15元无门槛学习奖励券
L.B.Q.Y
15元无门槛学习奖励券
WINNING LIST
GOOD LUCK

石头狮子

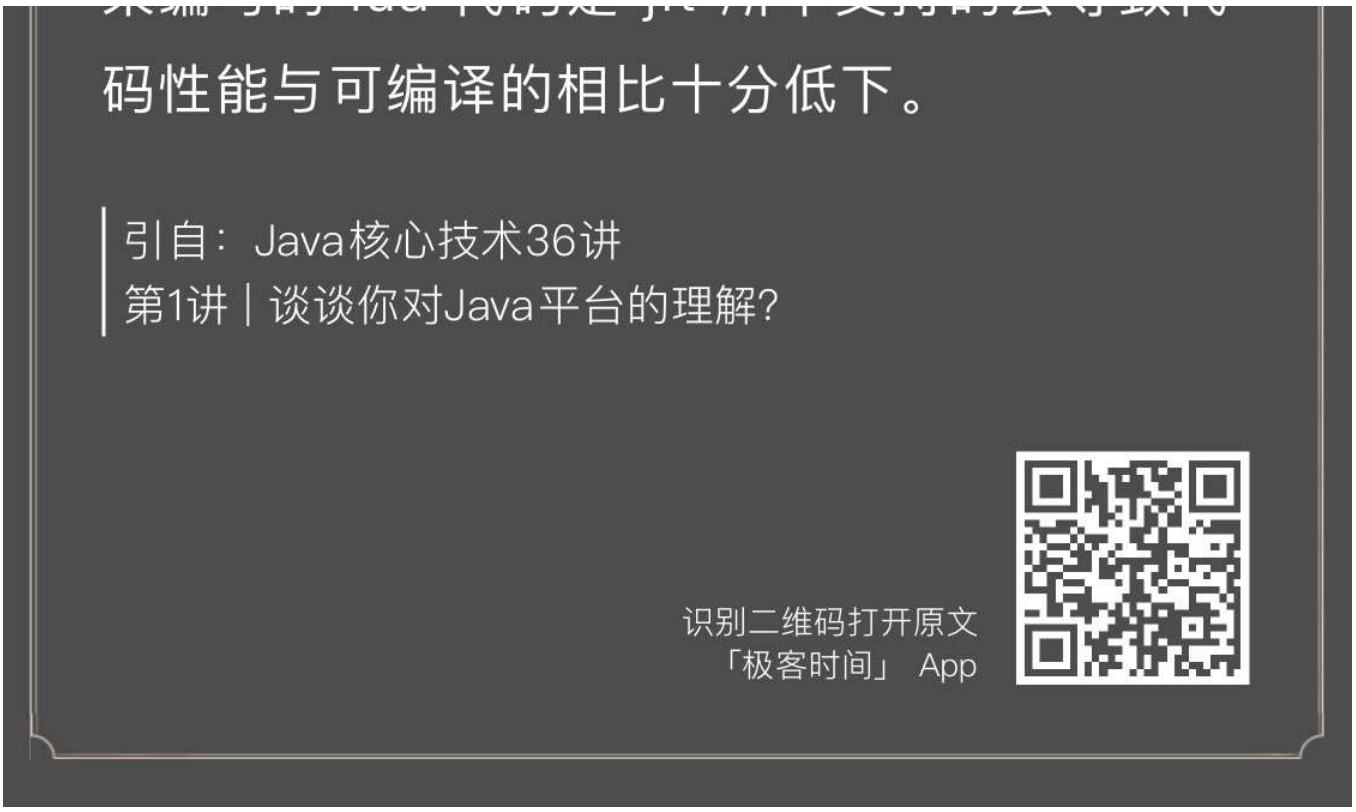
写于 2018/05/05

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。
引自：Java核心技术36讲
第1讲 | 谈谈你对Java平台的理解?
识别二维码打开原文
「极客时间」 App

Woj

写于 2018/05/05

“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。

> 引自：Java核心技术36讲
> 第1讲 | 谈谈你对Java平台的理解?

识别二维码打开原文
「极客时间」 App

kursk.ye
写于 2018/05/24

于是我 google 到了这篇文章,http://www.kdgregory.com/index.php?page=java.refobj ，花了几天（真的是几天，不是几小时）才基本读完，基本理解这几个 reference 的概念和作用，从这个角度来讲非常感谢作者，如果不是本文的介绍，我还以为 GC 还是按照 reference counter 的原理处理，原来思路早变了。话说回来，《Java Reference Objects》真值得大家好好琢磨，相信可以回答很多人的问题，比如 strong reference，soft reference，weak reference 怎么互转，如果一个 obj 已经 = null, 就 obj = reference.get() 呗，再有，文章中用 weak reference 实现 canonicalizing map 改善内存存储效率，减小存储空间的例子，真是非常经典啊。也希望作者以后照顾一下低层次读者，写好技术铺垫

和名词定义。顺便问一下大家是怎么留言的，在手机上打那么多字，还有排版是怎么处理的，我是先在电脑上打好字再 COPY 上来的，大家和我一样吗？

引自：Java核心技术36讲
第4讲 | 强引用、软引用、弱引用、幻象引用有什么区别？

识别二维码打开原文
「极客时间」 App

Miaozhe

写于 2018/05/18

接着上个问题：

老师，问个问题：我自己定义一个类，重写 finalize 方法后，创建一个对象，被幻想引用，同时该幻想对象使用 ReferenceQueue。

当我这个对象指向 null，被 GC 回收后，ReferenceQueue 中没有改对象，不知道是什么原因？如果我把类中的 finalize 方法移除，ReferenceQueue 就能获取被释放的对象。

2018-05-17 作者回复文章图里阐明了，幻象引用 enque 发生在 finalize 之后，你查查是不是卡在 FinalReference queue 里了，那是实现 finalization 的地方

杨老师，我去查看了，Final reference 和

Reference 发现是 Reference Handle 线程在监控，但是 Debug 进出去，还是没有搞清楚原理。

不过，我又发现类中自定义得 Finalize，如果是空的，正常。如果类中有任何代码，都不能进入 Reference Queue，怀疑是对象没有被 GC 回收。

引自：Java核心技术36讲
第4讲 | 强引用、软引用、弱引用、幻象引用有什么区别?

识别二维码打开原文
「极客时间」 App

肖一林

写于 2018/05/17

提一些建议：应该从两条线讲这个问题，一条从代理模式，一条从反射机制。不要老担心篇幅限制讲不清问题，废话砍掉一些，深层次的内在原理多讲些（比如 asm），容易自学的扩展知识可以用链接代替

代理模式（通过代理静默地解决一些业务无关的问题，比如远程、安全、事务、日志、资源关闭……让应用开发者可以只关心他的业务）

静态代理：事先写好代理类，可以手工编写，也可以用工具生成。缺点是每个业务类都要对应一个代理类，非常不灵活。

动态代理：运行时自动生成代理对象。缺点是生成代理代理对象和调用代理方法都要额外花费时间。

JDK 动态代理：基于 Java 反射机制

实现，必须要实现了接口的业务类才能用这种办法生成代理对象。新版本也开始结合 ASM 机制。

cglib 动态代理：基于 ASM 机制实现，通过生成业务类的子类作为代理类。

Java 发射机制的常见应用：动态代理（AOP、RPC）、提供第三方开发者扩展能力（Servlet 容器，JDBC 连接）、第三方组件创建对象（DI）……

我水平比较菜，希望多学点东西，希望比免费知识层次更深些，也不光是为了面试，所以提提建议。

> 引自：Java核心技术36讲
> 第6讲 | 动态代理是基于什么原理?

识别二维码打开原文
「极客时间」 App

曹铮

写于 2018/05/22

既然是 Java 的主题，那就用 PriorityBlockingQueue 吧。

如果是真实场景肯定会考虑高可用能持久化的方案。

其实我觉得应该参考银行窗口，同时三个窗口，就是三个队列，银台就是消费者线程，某一个窗口 vip 优先，没有 vip 时也为普通客户服务。要实现，要么有个 dispatcher，要么保持 vip 通道不许普通进入，vip 柜台闲时从其他队列偷

引自：Java核心技术36讲
第8讲 | 对比Vector、ArrayList、LinkedList 有何区别?

识别二维码打开原文
「极客时间」 App

雷霹雳的爸爸

写于 2018/05/22

在这个题目下，自然就会想到优先级队列了，但还需要额外考虑 vip 再分级，即同等级 vip 的平权的问题，所以应该考虑除了直接的和 vip 等级相关的优先级队列优先级规则问题，还得考虑同等级多个客户互相不被单一客户大量任务阻塞的问题，数据结构确实是基础，即便这个思考题考虑的这个场景，待调度数据估计会放在 redis 里面吧

引自：Java核心技术36讲
第8讲 | 对比Vector、ArrayList、LinkedList有何区别?

识别二维码打开原文
「极客时间」 App

vash_ace

写于 2018/06/01

其实在初始化 DirectByteBuffer 对象时，如果当前堆外内存的条件很苛刻时，会主动调用 System.gc() 强制执行 FGC。所以一般建议在使用 netty 时开启 XX:+DisableExplicitGC

引自：Java核心技术36讲
第12讲 | Java有几种文件拷贝方式？哪一种最高效？

识别二维码打开原文
「极客时间」 App

Walter

写于 2018/06/07

外观模式（Facade Pattern）隐藏系统的复杂性，并向客户端提供了一个客户端可以访问系统的接口。它向现有的系统添加一个接口，来隐藏系统的复杂性。

这种模式涉及到一个单一的类，该类提供了客户端请求的简化方法和对现有系统类方法的委托调用。

意图：为子系统中的一组接口提供一个一致的界面，外观模式定义了一个高层接口，这个接口使得这一子系统更加容易使用。

主要解决：降低访问复杂系统的内部子系统时的复杂度，简化客户端与之的接口。

何时使用： 1、客户端不需要知道系统内部的复杂联系，整个系统只需提供一个 " 接待员 " 即可。 2、定义系统的入口。

如何解决：客户端不与系统耦合，外观类与系统耦合。

关键代码：在客户端和复杂系统之间再加一层，这一层将调用顺序、依赖关系等处理好。

应用实例： 1、去医院看病，可能要去挂号、门诊、划价、取药，让患者或患者家属觉得很复杂，如果有提供接待人员，只让接待人员来处理，就很方便。 2、JAVA 的三层开发模式。

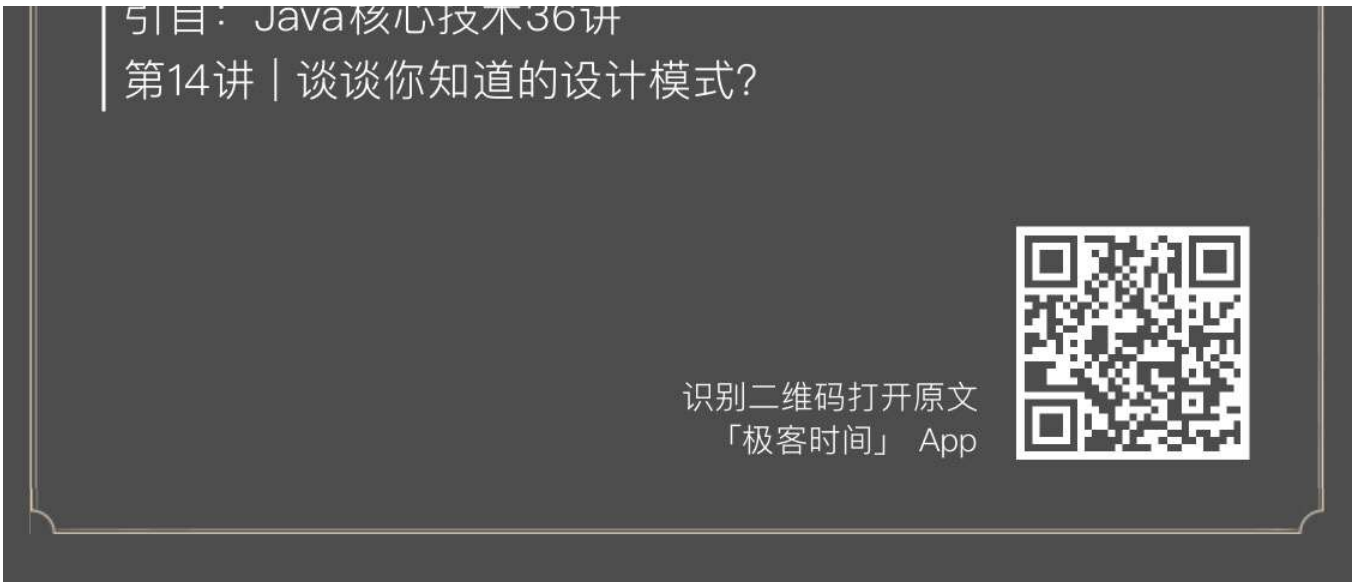
优点： 1、减少系统相互依赖。 2、提高灵活性。 3、提高了安全性。

缺点：不符合开闭原则，如果要改东西很麻烦，继承重写都不合适。

使用场景： 1、为复杂的模块或子系统提供外界访问的模块。 2、子系统相对独立。 3、预防低水平人员带来的风险。

注意事项：在层次化结构中，可以使用外观模式定义系统中每一层的入口。

引自：Java核心技术36讲



祝贺石头狮子、Woj、kursk.ye、Miaozhe、肖一林、曹铮、雷霹雳的爸爸、vash_ace、Walter，也要感谢I am a psycho、magict4、李林、Woong、L.B.Q.Y指出我文稿中的疏漏，一并送出学习奖励礼券。

精选留言

whhbbq 5

老师的招人标准，学习了，很实用，看了也很有感触。你能填上别人填不上的坑，就成功了。工作中大多数时候的任务目标并不清晰，特别当你是一个团队的小leader时，没人告诉你后面的方向，要做成什么样，很考验能力。

2018-06-09

vash_ace 3

感谢杨老师的鼓励，在受宠若惊之余，我觉得这篇“课外阅读”的参考价值不输于任何一篇技术分享。因为文中提到的这些努力或坚持的方向，确实是对个人职场生涯有着巨大的影响和帮助(亲测有效)。
其实道理大家都懂，但很多时候想当架构相做技术大牛的我们就是会以各种理由(项目忙，赶进度，不想加班...)不去对一个bug或一次线上故障做刨根问底的努力，又或者是放弃原本是对的坚持(比如，技术笔记，技术阅读与分享...)；
那这个时候的你所需要的鸡汤或兴奋剂不再是XXX的成功，而是想象一下自己的个人价值。往大了说，你对技术圈做了什么贡献？影响了多少人？影响越大成就感越满足。成就感是个好东西，你越享受就越会上瘾；往俗了说，就是看看自己的收入在行业内处于什么样的水平？工资多少不一定能完全体现一个人的真实水平，但至少绝大部分公司和猎头都能根据你上一家公司的收入来定位你属于哪一个level。

2018-06-11

公号-Java大后端 3

阅读源码的时候，

首先，可通过各种公开的渠道（google、公开文档等）了解代码的总体框架、模块组成及其模块间的关系；

然后，结合源码的注释进行解读，对不是很明白的部分打断点，调试，甚至可按照自己的想法进行修改后再调试；

最后，对于重点核心模块进行详细调试，可以把核心类的功能、调用流程等写下来，好记性总是敌不过烂笔头的。

除此之外，个人觉得最最重要的是:看源码的时候要有"静气"。

2018-06-09

夏洛克的救赎 3

看评论都能涨知识，希望评论提供交互功能

2018-06-09

雷霹雳的爸爸 2

意外，感谢，更重要的是也复盘下这段学习过程中发现的自己的各种不足，再接再厉！

2018-06-09

iLeGeND 1

我们应该面向接口编程，面向规范编程，在单纯的开发中，使jdk或者框架，应该以其api文档为参考，如果有问题就看源码，那岂不是面子实现编程了，不同的版本，其实现不见得一样，我们的代码用不能一直改吧

2018-06-09

Hidden 0

我在阅读源码的时候，只能勉强理解一半，剩下那一半 再怎么也理解不了，很是奇怪，

2018-06-13

Zoe.Li 0

谢谢杨老师的分享

2018-06-12

Miaozhe 0

感谢杨老师分享，这次学习收获很大，特别是认真阅读了HashMap的源码，桶的设计和Hash的位运算正的设计很妙。以前没有看懂，这次参考老师的"死磕"，终于看懂了。

2018-06-11

LenX 0

正文中(非留言区)，倒数第二个推荐的读者留言中说:

“所以一般建议在使用 Netty 时开启 XX:+DisableExplicitGC”。

注意，参数前使用的是 + 号，我觉得不对吧！
这就表明 Ststem.gc 变成空调用了，这对于 Netty，如果这么做会导致堆外内存不及时回收，反而更容易 OOM。

是这样吗？

2018-06-10

作者回复

我认为看场景和侧重角度，如果发现cleaner自动回收不符合需求，用system.gc至少可以避免oom；如果应用没这问题，调用它也可能导致应用反应不稳定等问题。
所以没有一劳永逸的办法或者最佳实践，只能是个思路参考，看实际需求

2018-06-10

肖一林 0

谢谢老师的奖励，每一篇都在看，最近也在组织以前的笔记，放在自己的技术公众号。希望清理技术债务，达到系统学习的目的。结合以前所学，加上老师文章提到的一些底层原理，用自己的方式表达一遍

2018-06-09

作者回复

加油，互相提高

2018-06-10

zt 0

话说今天不更新了吗，大神能不能加快下更新的速度，学习完去面试，战线时间太长有点熬人

2018-06-09

作者回复

有规定的节奏

2018-06-10

iLeGeND 0

我们应该面向接口编程，面向规范编程，在单纯的开发中，使jdk或者框架，应该以其api文档为参考，如果有问题就看源码，那岂不是面子实现编程了，不同的版本，其实现不见得一样，我们的代码用不能一直改吧

2018-06-09