> 本文由 简悦 SimpRead 转码， 原文地址 www.imooc.com

前面章节围绕《手册》的一些重要知识点进行了学习和拓展。

**为什么有些人“一年的经验用十年”，而有些人的学习和排错能力极强呢?**

因为很多人呆在舒适区不愿改变，不敢尝试新的方法，面对新的方法，工具本能地进行排斥，最终错过了快速成长的机会。

本节将系统阐述在几年的学习和工作中总结出来的 Java 学习的好方法，希望能够对大家有帮助。

# 2.1 读书、看官方文档

学习的最常见手段之一就是读书。

就 Java 而言，正如前面章节研究问题时所提到的， Java 学习最核心的是《Java 语言手册》、《JVM 语言手册》，因为这是官方，最权威的资料。

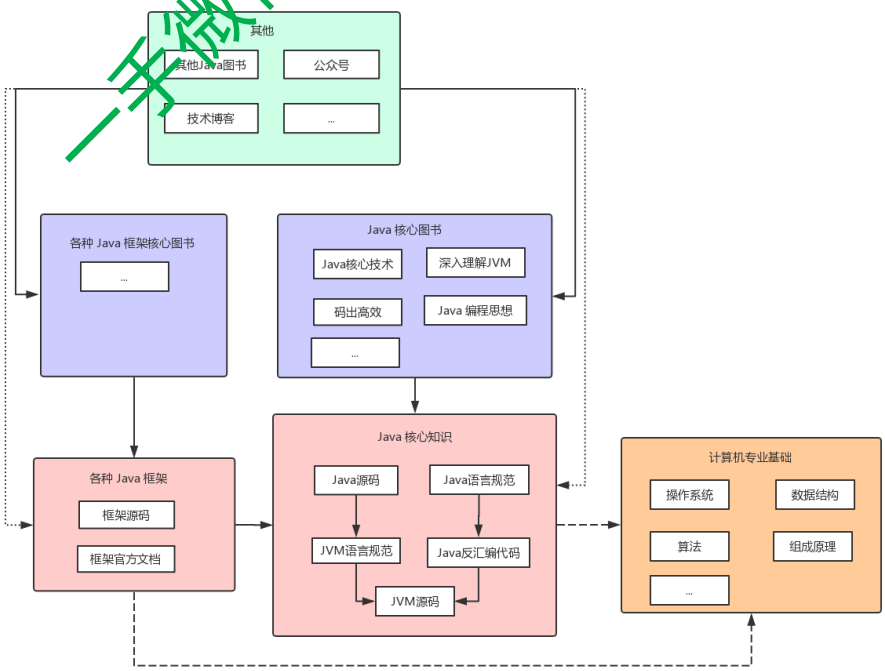
其次是围绕 java 语言典图书，如《Java 核心技术》、《Java 编程思想》、《深入理解 Java 虚拟机》、《码出高效》等。

还有 Java 开发岗位涉及到技术的官方手册，如 Spring、MyBatis、Hibernate 等官方手册等。

再如 Java 开发岗位涉及到的技术的经典图书，如《Redis 深度历险：核心原理与应用实践》、《深入理解 Apache Dubbo 与实战》等。

还有一些介绍编程技巧的书，如《重构：改善既有代码的设计》、《编写可读代码的艺术》、《代码整洁之道》、《修改软件的艺术》、《Effective Java》等。






如上图所示，越往底层信息密度越大，准确性越高，参考价值越大。

希望大家学习技术用法的同时，要注重源码、官方手册的学习，更加注重专业基础的学习。

**读书最关键的是选对书，其次是看书的方法。**

选取适合自己层次的图书很重要，初学者以介绍用法的图书为主，进阶的同学以读官方手册和相关的经典图书为主。

很多人读技术的书都会有这样的困惑：很多书读了一两遍还是会忘。

这是一个非常正常的现象。

为了克服遗忘，重点的图书需要反复阅读，这也就是所谓的 “书读百遍其义自见”，阅读时要加入一些思考，将所学知识和已经有的知识体系结合在一起。

## 2.2 看视频

和读书类似，看视频也是 Java 学习的重要手段。

视频更加生动形象，尤其对于初学者入门来说帮助很大。

对于有一定基础，尤其是工作后的同学一般会因为视频讲解过慢，进阶的免费视频较少等原因，看视频学习的时间会减少。

**俗话说 “外行人看热闹，内行人看门道”。**

很多人尤其是新手看视频，仅仅学习视频讲解的内容本身，导致学习新的知识没有视频就不知道如何下手。

其实看视频学习不仅是看作者讲解的内容本身，更应该关注作者的编码风格，作者学习新知识的方式，作者的编码思路，这些才是更有价值的通用的技能，也是看视频学习的精髓所在。

## 2.3 读源码

读源码是学习进阶的必由之路，但是同样是读源码，不同的方法效果差异很大。

对于 Java 程序员而言，最重要的就是读 JDK 的源码，其次是读一些经典框架的源码，如 Spring、Dubbo 等。

通过读源码的注释，可以深入了解函数的主要功能，甚至核心思路。

通过阅读源码，可以了解到一些优秀的编码风格，设计思想。还可以通过源码来学习和理解其中所运用的设计模式。

读源码时要特别注重思考为什么要这么设计，不这么设计会有什么问题。

然而，很多 Java 初学者，甚至工作一两年的 Java 程序员，读源码抓不住重点，以记忆为主，而不是思考为主。

读源码时迷失在细节之中，而不能先整体后局部，从设计者的角度来读源码。

如何更好地阅读源码，在后面的章节中将会重点介绍。

## 2.4 调试

IDE 的代码调试器也是 Java 学习重要手段。

通过代码调试可以清楚代码的运行轨迹，可以清楚地观察各个对象的状态。

但是很多 Java 初学者，甚至工作一两年的 Java 程序员， 也只是停留在打个断点，单步调试，并没有熟练掌握更高级的调试方法。

在后面的章节会系统地介绍调试的正确姿势。

## 2.5 看专栏

现在博客的质量参差不齐，尤其中文技术博客各种相互抄袭，很多不错的技术博客容易断更等。

随着近些年知识付费的普及，专栏成为学习知识的一个重要途径。专栏都是围绕着一个主题写作，购买和学习专栏可以对某一块知识有一个全面和深入地理解。

建议大家可以通过购买专栏的方式，系统地将自己不足的模块补齐，而不是低效地碎片化学习。

## 2.6 看公众号、博客等

对于 Java 程序员而言，技术公众号也是学习的重要途径。

比较不错的公众号有： 架构师之路、IT 技术精选文摘、JavaGuide、程序员小灰等等。

看一些分享学习和工作中的经验技巧的博客，也是学习的一个不错途径。

但由于公众号和博客的质量参差不齐，读公众号和博客要抱着怀疑的态度，不要轻信。

公众号和博客只能作为参考，不应该作为权威的依据。

## 2.7 各种图

作为 Java 程序员，思维导图和 UML 图都是学习和设计的强有力工具。

通过思维导图，可以整理需求，梳理所学知识并构建知识体系。

UML 图更是需求分析、系统设计、梳理系统逻辑必不可缺的强大工具。

思维导图和 UML 图在后续的章节中也会给出全面的讲解。

## 2.8 其他

还可以通过抓包（tcpdump -A | grep xxx）、反汇编、反编译，来学习 Java 相关知识。

也可以通过 Google 和 Stack OverFlow 来搜索问问题的答案。

不推荐使用百度搜索问题，是因为百度出的很多问题的答案千篇一律。

搜索到的解决方案也往往没有给出该问题最本质的原因。

而 Google 和 Stack OverFlow 搜索出来的资料质量相对较高，而且往往会给出问题的根本原因以及参考资料。



## 3.1 推演验证

对于大多数工作的朋友来说，入职的部门一般都会有已经设计好的项目模块，这是我们学习进阶的**绝佳素材**。

但是很多人并不会重视和分析之前的设计，也体会不到这种学习方式给自己能够带来的极大成长。

我们可以使用推演验证的方式来快速掌握系统的设计、熟悉功能、学习设计思路等。而不是进入侧重记忆，导致容易遗忘，无法灵活运用的怪圈。

> 所谓的**推演验证**，就是当我们熟悉一块功能，或者我们想通过某个模块来提高编码和设计能力时，我们可以找到需求文档和已经上线的产品去对比使用，然后模拟自己出一个技术方案，包括数据库表的设计，分布式中间件的使用还有一些其它细节等。

我们通过需求文档、交互稿或者体验真实的功能后去反推实现方式，如果当初有技术文档，要和当初的实际的技术方案进行对比。通过对比找出自己设计的缺陷，思考对方当初为何这么设计，并努力找出对方的设计可以改进的地方。通过不断的推演和验证，自己的业务设计能力会提高的很快。

这种方法有点像学生时代做 “模拟题”，当初某个项目的设计方案就是我们 “模拟题的答案”，通过这种方式的训练，我们的 “工作经验” 会提高很快。而现实生活中，往往是我们没有和 “真题” 同等难度，甚至更难的 “模拟题” 的经验，而只是从自己趟过的坑，做过的项目，通过别人的指点来学习，这样就会收效甚微。

下面结合一个场景来为大家解释具体的做法：

> 比如很多人会发现市面上很少有很通俗易懂地教你如何根据实际的场景去设计数据库表结构的资料，肿么办？
>
> 对于还没工作的人来说，可以找一两个知名的开源项目；对于已经工作的人来说，可以通过自己开发的项目来学习表的设计。
>
> 如何学习呢？
>
> 我们先找某一个自己感兴趣的功能点，根据需求文档、页面表现等，熟悉功能。
>
> 然后根据功能自行**推演**，如应该会有几张表，每张表应该包含哪些字段等等。
>
> 然后去和实际的表结构去对比**验证**，如果不同，自己设计的表是否满足需求？对方的设计更好吗？好在哪里？自己是否有遗漏？等等。
>
> 通过多次训练，你会发现自己对表的设计掌握的会越来越好，你将更清楚为什么要这么设计以及如何设计。

后续的源码学习章节的其中一种高效的读源码方法也将采用 “推演和验证” 的方法。

只有真正尝试过这种方法的人才能体会到它的巨大价值，希望大家在平时开发和学习中多去尝试。

## 3.2 教是最好的学

我们学了好多年，但是很多人从来不会主动探索高效的学习方法，在学习过程中见到的最好的学习方法之一就是 “教学相长”。

俗话说 “教是最好的学”，在中国的古代文献中有类似的说法。

《礼记・学记》：“学然后知不足，教然后知困。知不足，然后能自反也；知困，然后能自强也。故曰：教学相长也。

《兑命》曰：“‘学学半。’其此之谓乎。” 郑玄注：“学则睹己行之所短，教则见己道之所未达”。

另外著名的费曼学习法也是强调 “教学相长”：

> **费曼学习法**的灵感源于诺贝尔物理奖获得者理查德・费曼（Richard Feynman），运用费曼技巧可以深入理解知识点，并且记忆深刻不易遗忘。知识有两种类型，我们绝大多数人关注的是错误的那种。第一类知识关注了解某个事物的名称。第二类知识关注了解某个事物，这是两码事，通过费曼学习法可以让我们对事物的理解更透彻。

费曼学习法的四个步骤：

- 第一步：将学的知识教给别人；
- 第二步：回顾。遇到卡壳的地方，回顾原材料，重新学习；
- 第三步：将语言条理化，简化。只有能够用简单易懂的语言描述某个知识，才代表真正理解了某个知识；
- 第四步：传授。确保自己理解无误的情况下，将知识传授给他人。

因此我们学习一个知识点时可以尝试教给别人，如果没有合适的倾听者可以讲给 “小黄鸭”（虚拟的听众）听。

另外也可以将所需的知识写到博客中，也可以在技术群里和别人交流，这也是另外一种 “教”。

## 3.3 PDCA 循环

PDCA 循环的来源和定义最早是由美国质量管理专家戴明提出来的，也称 “戴明环”。


图：PDCA 循环示意图（图片来自百度百科）

核心内容如下：

1. **P（Plan）** 即计划。“凡事预则立，不立则废”，做事之前一定要有计划；
2. **D（Do）** 即所谓的执行。很多人有了计划没有执行力，很难有效果；
3. **C（Check）** 即检查。分析执行的结果，明确执行中存在哪些问题；
4. **A（Action）** 即处理。对成功的经验要保持，将其标准化；对失败的经验要总结，引起重视。

请注意，这里并不是为了介绍某个高大上的概念，而是希望大家能够真正去思考去理解。

为了更好地理解 PDCA 的价值，我将生活和开发中的一些场景进行描述以帮助大家理解。

> 在学生时代很多人考试之前会做模拟题或者历年真题，你会发现，很多人做完题后只是把答案抄上去再也不看了。导致的一个严重的问题是，做过的题大概率还错。

这其实就是走了弯路而不自知，浪费了时间却没有成效。

在我们学习 Java 和参与工作中，很多人一样会存在这种问题：

> 开发一个模块之前没有充分分析（甚至轻视）需求，着急编写代码，编写代码过程中不能够有意识的和 master 分支进行比对，代码部署到测试服务器时随便点点就算，如果上线后遇到 BUG 就进行修复。

很多人学习时更喜欢 “做更多试卷” 给自己带来的虚假成就感，而不是珍惜错题给自己带来的价值。

同样地，做项目时，很多人喜欢做更多项目给自己带来的虚假 “成长”，每个项目并没有输出对下一个项目有用的经验。

这也是很多人 “一年经验用了十年”，成长很慢的原因之一。

**那么该怎么做呢?**

编写代码之前一定要充分了解需求，做好技术方案，然后再去编码设计。

> 很多有经验的人都会认为，其实软件开发最关键的是需求的分析，技术方案的设计，而编码只不过是一个时间问题。

编写完代码后将自己的分支和 master 比对，检查是否有冲突，进行自我代码审查。

> 养成和 master 对比代码的习惯，你可以在测试之前就发现自己编码的问题。

每个项目上线后，总结这些项目学到的经验，存在哪些问题，如果有 BUG，分析 BUG 的原因。

> 比如
>
> [1] 这次测试过程中发现某个错误日志在测试环境就已经看到过，但是认为和本功能无关就没注意，最终发现是本功能的修改触发了其他功能的 BUG。
>
> 此时我们可以将 “测试时一定要重视警告和错误日志” 积累到我们的验证流程中；比如测试时可以养成好的习惯，使用 tail -f error.log 看看有没有错误日志。
>
> [2] 这次开发过程中，我们本地代码都没编译通过就提交到了 git 仓库并在测试服务器打包部署，导致部署失败。
>
> 此时，我们可以将 “提交之前一定要本地编译通过” 积累到我们的开发经验中。比如 maven 项目我们可以每次提交前执行 mvn clean compile。
>
> [3] 比如我们代码上线后某个地方忘了打日志，不方便我们分析排查问题。那么我们可以梳理出哪些地方该打日志，下次一定开发时要重视起来。
>
> 等等。

然后将经验梳理到云笔记中或者思维导图中，并让它们成为我们的习惯，不断地帮助下一次开发。

这样才能造成良性循环，这样经验才能有质的的快速累积。

并不只是学习新的技术才叫学习，决定一个人是否牛的标准之一是他能否在某一个领域超越绝大多数人。

**很多人学习进阶的速度慢，不仅仅在于有些东西没有学，而是不懂学习的方法，不懂总结和反思。**

孔子说 **“温故而知新，可以为师矣”** 诚如是。

本节主要讲述了 Java 学习的主要途径，包括读书、看视频、读源码、调试等；还推荐了几种高效的 Java 学习方法，如费曼学习法、PDCA 循环等。本小节还分析了很多人学习和进阶速率慢的主要原因，即不重视方法，不重视总结和反思。人最可怕的是一直呆在舒适区，不愿意改变，不愿意接受新鲜事物，不愿意学习新的方法。希望通过本节的介绍，对大家能够有所启发。

下一节我们将学习代码调试的基本方式和高级姿势。