98 | 高效学习：如何学习和阅读代码

2018-9-6 陈皓

读文档还是读代码

杰夫·阿特伍德（Jeff Atwood）说过这么一句话：“Code Tells You How, Comments Tell You Why”。我把其扩展一下：

> **代 码 => What, How & Details**
>
> **文档/书 => What, How & Why**

可见，**代码并不会告诉你 Why**，看代码只能靠猜测或推导来估计Why，是揣测，不准确，所以会有很多误解。**而且，我们每个人都知道，Why是能让人一通百通的东西，也是能让人醍醐灌顶的东西**。

但是，**代码会告诉你细节**，这是书和文档不能给你的。**细节是魔鬼，细节决定成败**。这样的话我们不但听过很多，我们做技术的也应该体会过很多。当然，我们也要承认，这些代码细节给人带来的快感毕竟不如知道Why后的快感大（至少对我是这样的）。

书和文档是人对人说的话，代码是人对机器说的话（注：代码中有一部份逻辑是控制流程的逻辑，不是业务逻辑）。所以：

1. **如果你想知道人为什么要这么搞，那么应该去看书**（像Effective C++、Code Complete、Design Pattern、Thinking in Java等），**看文档**。
2. **如果你要知道让机器干了什么？那你应该看代码！**（就像Linus去看zlib的代码来找性能问题。）

因此，我认为都比较重要，关键看你的目的是什么了。

> **如果你想了解一种思想，一种方法，一种原理，一种思路，一种经验，恐怕，读书和读文档会更有效率一些**，因为其中会有作者的思路描述。像Effective C++之类的书，里面有很多对不同用法和设计的推敲，TCP/IP详解里面也会有对TCP算法好坏的比较……这些思维方式能让你对技术的把握力更强，而光看代码很难达到这种级别。（现在你知道什么样的书是好书了吧 ;-)）

如果你想了解的就是具体细节，比如某协程的实现，某个模块的性能，某个算法的实现，那么你还是要去读代码的，因为代码中会有更具体的处理细节（尤其是对于一些edge case或是代码技巧方面的内容）。

另外，看看下面的几个现象，你可以自己比较一下。

很多时候，我们去读代码，那是因为没有文档，或是文档写得太差。

很多时候，**在Google、Stack Overflow、GitHub过后，你会发现，你掌握的知识就是一块一块的碎片，既不系统，也不结构化，更别说融会贯通了。你会觉得自己需要好好地读一本书，系统地掌握知识。你的这种感觉一定很强烈吧**。

很多时候，在读别人代码的时候，你会因为基础知识或是原理不懂，或是你在不知道为什么的情况下，要么完全读不懂代码，要么会误解代码。比如，如果你没有C语言和TCP原理方面的基础知识，就根本读不懂Linux下TCP的相关代码。我们因为误解代码用意而去修改代码造成的故障还少吗？

很多时候，看到一个算法或是一个设计时，比如Paxos，你是不是会想去看一下这个算法的实现代码是什么样的？思考一下如何才能实现得好？（但是如果你没看过Paxos的算法思想，我不认为你光看代码实现，就能收获Paxos的思想。）

很多时候，**当你写代码的时候，你能感觉得到自己写的代码有点别扭，怎么写都别扭，这个时候，你也会有想去看别人的代码是怎么实现的冲动**。

类似的情况还有很多，但从代码中收获大，还是从书中收获大，在不同的场景、不同的目的下，会有不同的答案。这里，谈一谈人的学习过程吧。从学习的过程中，我们来分析一下看代码和看书这两个活动。人对新事物的学习过程基本都是从“感性认识”到“理性认识”的。

如果你是个新手，那应该多读代码，多动手写代码，因为你需要的是“感性认识”，这个时候“理性认识”你体会不到。一是因为，你没有切身的感受，即便告诉你Why你也体会不到。另一方面，这个阶段，你要的不是做漂亮，而是做出来。所以，在**新手阶段，你会喜欢GitHub这样的东西**。

如果你是个老手，你有多年的“感性认识”了，那么你的成长需要更多的“理性认识”。因为这个阶段，一方面，你会不满足于做出来，你会想去做更牛更漂亮的东西；另一方面，你知道的越多，你的问题也越多，你迫切地需要知道Why！这时，你需要大量地找牛人交流（读牛人的书，是一种特殊的人与人的交流），所以，**这个阶段，你会喜欢读好的书和文章**。

然而，对于计算机行业这个技术创新能力超强、技术种类繁多的行业来说，我们每个人都既是新手，也是老手。

如何阅读源代码

很多人问过我，如何读代码。因为我在外企里工作的时间较长，所以，我经常接手一些国外团队写的代码。我发现，虽然老外写的代码比国人好一点儿（有Code Review），但依然有文档缺失、代码注释不清、代码风格混乱等一些问题，这些都是阅读代码的障碍。这里，我把我的一些阅读源代码的经验分享给你，希望对你有用。

首先，在阅读代码之前，我建议你需要有下面的这些前提再去阅读代码，这样你读起代码来会很顺畅。

1. **基础知识**。相关的语言和基础技术的知识。
2. **软件功能**。你先要知道这个软件完成的是什么样的功能，有哪些特性，哪些配置项。你先要读一遍用户手册，然后让软件跑起来，自己先用一下感受一下。
3. **相关文档**。读一下相关的内部文档，Readme也好，Release Notes也好，Design也好，Wiki也好，这些文档可以让你明白整个软件的方方面面。如果你的软件没有文档，那么，你只能指望这个软件的原作者还在，而且他还乐于交流。
4. **代码的组织结构**。也就是代码目录中每个目录是什么样的功能，每个文档是干什么的。如果你要读的程序是在某种标准的框架下组织的，比如：Java的Spring框架，那么恭喜你，这些代码不难读了。

接下来，你要了解这个软件的代码是由哪些部分构成的，我在这里给你一个列表，供你参考。

1. **接口抽象定义**。任何代码都会有很多接口或抽象定义，其描述了代码需要处理的数据结构或者业务实体，以及它们之间的关系，理清楚这些关系是非常重要的。
2. **模块粘合层**。我们的代码有很多都是用来粘合代码的，比如中间件（middleware）、Promises模式、回调（Callback）、代理委托、依赖注入等。这些代码模块间的粘合技术是非常重要的，因为它们会把本来平铺直述的代码给分裂开来，让你不容易看明白它们的关系。
3. **业务流程**。这是代码运行的过程。一开始，我们不要进入细节，但需要在高层搞清楚整个业务的流程是什么样的，在这个流程中，数据是怎么被传递和处理的。一般来说，我们需要画程序流程图或者时序处理图。

4. **具体实现**。了解上述的三个方面的内容，相信你对整个代码的框架和逻辑已经有了总体认识。这个时候，你就可以深入细节，开始阅读具体实现的代码了。对于代码的具体实现，一般来说，你需要知道下面一些事实，这样有助于你在阅读代码时找到重点。

   代码逻辑。代码有两种逻辑，一种是业务逻辑，这种逻辑是真正的业务处理逻辑；另一种是控制逻辑，这种逻辑只是用控制程序流转的，不是业务逻辑。比如：flag之类的控制变量，多线程处理的代码，异步控制的代码，远程通讯的代码，对象序列化反序列化的代码等。这两种逻辑你要分开，很多代码之所以混乱就是把这两种逻辑混在一起了（详情参看《编程范式游记》）。

   出错处理。根据二八原则，20%的代码是正常的逻辑，80%的代码是在处理各种错误，所以，你在读代码的时候，完全可以把处理错误的代码全部删除掉，这样就会留下比较干净和简单的正常逻辑的代码。排除干扰因素，可以更高效地读代码。

   数据处理。只要你认真观察，就会发现，我们好多代码就是在那里倒腾数据。比如DAO、DTO，比如JSON、XML，这些代码冗长无聊，不是主要逻辑，可以不理。

   重要的算法。一般来说，我们的代码里会有很多重要的算法，我说的并不一定是什么排序或是搜索算法，可能会是一些其它的核心算法，比如一些索引表的算法，全局唯一ID的算法、信息推荐的算法、统计算法、通读算法（如Gossip）等。这些比较核心的算法可能会非常难读，但它们往往是最有技术含量的部分。

   底层交互。有一些代码是和底层系统的交互，一般来说是和操作系统或是JVM的交互。因此，读这些代码通常需要一定的底层技术知识，不然，很难读懂。

5. **运行时调试**。很多时候，代码只有运行起来了，才能知道具体发生了什么事，所以，我们让代码运行进来，然后用日志也好，debug设置断点跟踪也好。实际看一下代码的运行过程，是了解代码的一种很好的方式。

总结一下，阅读代码的方法如下：

一般采用自顶向下，从总体到细节的“剥洋葱皮”的读法。

画图是必要的，程序流程图，调用时序图，模块组织图……

代码逻辑归一下类，排除杂音，主要逻辑才会更清楚。

debug跟踪一下代码是了解代码在执行中发生了什么的最好方式。

对了，阅读代码你需要一个很好的IDE。我记得以前读C和C++代码时，有一个叫source insight的工具就大大提高了我的代码阅读效率。说白了就是可以查看代码间相互的调用

reference的工具，这方面Visual Studio做得是非常好的。

小结

总结一下今天的内容。我先跟你探讨了“是读文档，还是读代码”，分析对比了从文档和代码中各自能收获到哪些东西，然后给出建议，如果想了解思想、方法和原理，读书和读文档会更有效率；如果想知道具体细节，还是应该读代码。随后分享了一些我阅读代码和源代码时候的方法和技巧。希望对你有启发。

下篇文章是《高效学习》系列的最后一篇，我将分享一下面对枯燥和量大的知识时，我们该怎样做。

下面是《高效学习》系列文章的目录。

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

精选留言 19

孙悟空
1536194114

耗子叔，可以推荐几个代码质量不错，适合精读的c/c++/java/go的开源项目么。

metalmac.kyle
1536194929

Code Tells You How,Comments Tell You Why。 Brilliant！这总结太精妙了

Aleck
1536245666

耗子叔，看了您前端、后端、机器学习等等讲了很多，就是没有讲移动的。
您啥时候能讲一下移动的发展路径和知识树吗？然后自己也感觉未来移动的需求会趋向于饱和，请问耗子叔对移动的未来发展怎么看

lovedebug
1536230267

读源码推荐opengrok，自己搭建一个server就行

Y024
1536590920

Source Insight 感觉有点上古神器的样子，当年在外企必备神器，怀念下。

刘鹏
1536201262

不知道Linus如果不幸离世，还有没有人看得懂Linux内核

且听风吟
1536195293

这个方法很赞呀。最近正要看Java concurrent相关的源码，希望能啃下来。。。

拯救地球好累
1565366741

文档和书籍有助于理解原理和思想，代码更有助于了解细节。
阅读源代码并非是上来就硬钢代码，首先要补充相应的基础知识，明确软件功能并简单实践，阅读相应的文档说明。在进入细节前还应先把握代码整体的组织结构，再从接口抽象和模块粘合中把握基本要点，再弄清楚业务流转的流程(画图)。在看具体实现时应当区分业务逻辑和控制逻辑，正常代码和出错处理代码，正常逻辑和数据处理逻辑，核心算法与其他算法，软件内的实现与底层实现。区分这几种代码可以让我们更有效地抓住重点，删繁就简。
最后，可以考虑运行时调试。

张健_Bamvor
1562373710

听完耗子哥这篇文章之后，发现自己模块的文档，有写的不够清楚的地方，主要没有讲清楚要做什么，当前做到什么程度，将来会怎么引进。

edisonhuang
1560472901

读代码和读文档，两者一个是抓住原理，掌握道，一个抓住细节，掌握术。在实践过程中的不同时刻，会需要不同的侧重。
耗子哥读代码的经验在于，首先采用自顶向下的方式，先归类，然后了解软件功能，深入模块，运行跟踪，摸透细节。有好的代码跟踪工具会让整个过程事半功倍。

godtrue
1545918673

最近再读JDK源码，浩哥的方法正好用上

caohuan
1540291780

耗子哥 说的太到位了，我属于死磕物种，比如16年的项目交接的时候，项目的老人走了，没留下 任何文档，到我们做维护，需要修改功能或者做优化时，一般是瞎猜 如果运气好 秒解，有时候 很悲催 一周也搞不定，往往需要重新捋代码的业务，我们开发的项目也是以没有规范和文档为荣，后来业务需要调整，可之前的设计逻辑忘得差不多了，来回来去的 做了很

多无用功，自己认识规范的重要性都有好大的代价，所以跟随 耗子哥 可以少走没必要的弯路。
还有 觉得 我应该去找从未有过 的why的体验。

薛璇
1540132717

这也意味着写应用程序时，规范的设计应该分几块去考虑，接口列表，粘合层，控制逻辑，错误处理，业务逻辑。

青蛙爱吃土豆
1536408518

有什么代码质量比较好的java项目可以推荐下吗？

锎杜
1536245883

耗子叔，到您这个水平是不是做什么业务都已经游刃有余？我个人非常好奇

小薛薛
1536202106

放下急于求成，慢慢能体会到理解深挖一个个小知识点的酸爽了。 why就是舒服。

袋鼠先生
1536197601

总结很到位

太阳雪
1536193545

这期的内容感同身受，计算机本身就是个复杂的东西，如果没有技巧和方法就会迷路，您总结的非常好，如果更早的知道这些方法和指导，能提升很大的效率。

太阳雪
1536193545

这期的内容感同身受，计算机本身就是个复杂的东西，如果没有技巧和方法就会迷路，您总结的非常好，如果更早的知道这些方法和指导，能提升很大的效率。