71 | 如何阅读别人的代码?

2020-01-07 许式伟

许式伟的架构课 进入课程 >

讲述：姚迪迈
时长 09:01 大小 8.28M

你好，我是七牛云许式伟。今天聊聊如何阅读别人的代码。

为何要读别人的代码?

我们去阅读别人的代码，通常会带有一定的目的性。完整把一个系统的代码 “读懂” 需要极大的精力。所以明确阅读代码的目标很重要，因为它决定了你最终能够为这事付出多大的精力，或者说成本。

大体来说，我们可以把目标分为这样几种类型：

我要评估是否引入某个第三方模块；

我要给某个模块局部修改一个 Bug（可能是因为使用的第三方模块遇到了一个问题，或者可能是你的上级临时指定了一个模块的 Bug 给你）；

我要以某个开源模块为榜样去学习；

我要接手并长期维护某个模块。

为什么要把我们的目标搞清楚?

因为读懂源代码真的很难，它其实是架构的反向过程。它类似于反编译，但是并不是指令级的反编译，而是需要根据指令反推更高维的思想。

我们知道反编译软件能够将精确软件反编译为汇编，因为这个过程信息是无损的，只是一种等价变换。但是要让反编译软件能够精确还原出高级语言的代码，这就比较难。因为编译过程是有损的，大部分软件实体的名字已经在编译过程中被去除了。当然，大部分编译器在编译时会同时生成符号文件。它主要用于 debug 用途。否则我们在单步跟踪时，debug 软件就没法显示变量的名字。

即使我们能够拿到符号文件，精确还原出原始的高级语言的代码仍然非常难。它需要带一定的模型推理在里面，通过识别出这里面我们熟悉的 “套路”，然后按照套路进行还原。

我们可以想像一下，“一个精确还原的智能反编译器” 是怎么工作的。

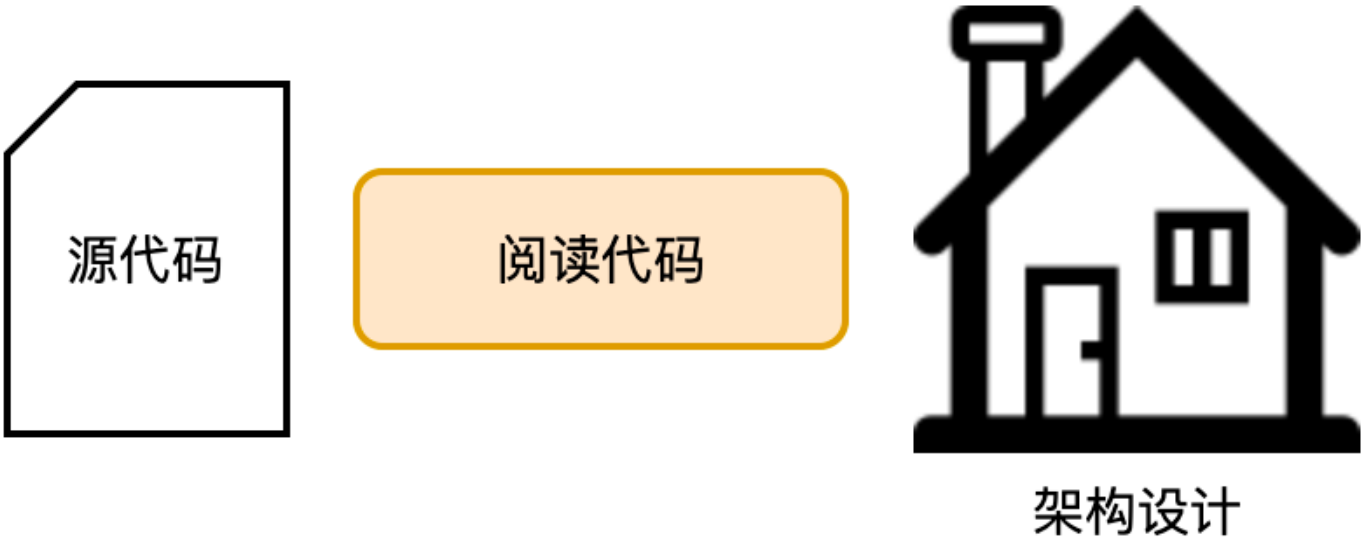
第一步，它需要识别出所采用的编程语言和编译器。这通常相对容易，一个非常粗陋的分类器就可以完成。尤其是很多编译器都有 “署名”，也就是在编程出的软件中带上自己签名的习惯。如果假设所有软件都有署名，那么这一步甚至不需要训练与学习。

第二步，通过软件的二进制，结合可选的符号文件（没有符号文件的结果是很多软件实体，比如类或函数的名字，会是一个随机分配的符号），加上它对该编译器的套路理解，就可以进行反编译了。

编译器的套路，就如同一个人的行为，持续进行观察学习，是可以形成总结的。这只需要反编译程序持续地学习足够多的该编译器所产生的样本。

我之所以拿反编译过程来类比，是希望我们能够理解，阅读源代码过程一方面是很难的，另一方面来说，也是需要有产出的。

有产出的学习过程，才是最好的学习方式。



那么阅读源代码的产出应该是什么？答案是，构建这个程序的思路，也就是架构设计。

理解架构的核心脉络

怎么做到？

首先，有文档，一定要先看文档。如果原本就已经有写过架构设计的文档，我们还要坚持自己通过代码一步步去反向进行理解，那就太傻了。

但是，一定要记住文档和代码很容易发生脱节。所以我们看到的很可能是上一版本的，甚至是最初版本的设计。

就算已经发生过变化，阅读过时的架构设计思想对我们理解源代码也会有极大的帮助作用。在这个基础上，我们再看源代码，就可以相互进行印证。当然如果发生了冲突，我们需要及时修改文档到与代码一致的版本。

看源代码，我们首先要做到的是理解系统的概要设计。概要设计的关注点是各个软件实体的业务范畴，以及它们之间的关系。有了这些，我们就能够理解这个系统的架构设计的核心脉络。

具体来说，看源码的步骤应该是怎样的呢？

首先，把公开的软件实体（模块、类、函数、常量、全局变量等）的规格整理出来。

这一步往往有一些现成的工具。例如，对 Go 语言来说，运行 go doc 就可以帮忙整理出一个自动生成的版本。一些开源工具例如 doxygen 也能够做到类似的事情，而且它支持几乎所有的主流语言。

当然这一步只能让我们找到有哪些软件实体，以及它们的规格是什么样的。但是这些软件实体各自的业务范畴是什么，它们之间有什么关系？需要进一步分析。

一般来说，下一步我会先看 example、unit test 等。这些属于我们研究对象的客户，也就是使用方。它们能够辅助我们理解各个软件实体的语义。

通过软件实体的规格、说明文档、example、unit test 等信息，我们根据这些已知信息，甚至包括软件实体的名字本身背后隐含的语义理解，我们可以初步推测出各个软件实体的业务范畴，以及它们之间的关系。

接下来，我们需要进一步证实或证伪我们的结论。如果证伪了，我们需要重新梳理各个软件实体之间的关系。怎么去证实或证伪？我们选重点的类或函数，通过看它们的源代码来理解其业务流程，以此印证我们的猜测。

当然，如果你能够找到之前做过这块业务的人，不要犹豫，尽可能找到他们并且争取一个小时左右的交流机会，并提前准备好自己遇到迷惑的问题列表。这会大幅缩短你理解整个系统的过程。

最后，确保我们正确理解了系统，就需要将结论写下来，形成文档。这样，下一次有其他同学接手这个系统的时候，就不至于需要重新再来一次 “反编译” 。

理解业务的实现机制

业务系统的概要设计、接口理清楚后，通常来说，我们对这个系统就初步有谱了。如果我们是评估第三方模块要不要采纳等相对轻的目标，那么到此基本就可以告一段落了。

只有在必要的情况下，我们才研究实现机制。刚才我们谈到系统架构梳理过程中，我们也部分涉及了源代码理解。但是，需要明确的是，前面我们研究部分核心代码的实现，其目的还是为了确认我们对业务划分猜测的正确性，而不是为了实现机制本身。

研究实现是非常费时的，毕竟系统的 UserStory 数量上就有很多。把一个个 UserStory 的具体业务流程都研究清楚写下来，是非常耗时的。如果这个业务系统不是我们接下来重点投入的方向，就没必要在这方面去过度投入。

这时候目标就很重要。

如果我们只是顺带解决一下遇到的 Bug，无论是用第三方代码遇到的，还是上级随手安排的临时任务，我们自然把关注点放在要解决的 Bug 本身相关的业务流程上。

如果我们是接手一个新的业务系统，我们也没有精力立刻把所有细节都搞清楚。这时候我们需要梳理的是关键业务流程。

怎么搞清楚业务流程?

> 程序 = 数据结构 + 算法

还是这个基础的公式。要搞清楚业务流程，接下来要做的事情是，把这些业务流程相关的数据结构先理清楚。

数据结构是容易梳理的，类的成员变量、数据库的表结构，通常都有快速提取的方式。除了 MongoDB 可能会难一些，因为弱 schema 的原因，我们需要通过阅读代码的方式去理解 schema。更麻烦的是，我们不确定历史上经历过多少轮的 schema 变更，这通过最新版本的源代码很可能看不出来。一个不小心，我们就可能会处理到非预期 schema 的数据。

理清楚数据结构，事情就解决了大半。

剩下来就是理各个 UserStory 的业务流程，并给这些业务流程画出它的 UML 时序图。这个过程随时可以补充。所以我们挑选对我们当前工作最为相关的来做就好了。

最后，还是同样地，我们要及时把我们整理的结论写下来，变成架构文档的一部分。这样随着越来越多人去补充完整架构设计文档，才有可能把我们的项目从混沌状态解脱出来。

结语

对于任何一个项目团队来说，阅读代码的能力都极其重要。哪怕你觉得你的团队共识管理很好，团队很默契，大家的工程习惯也很好，也都很乐意写文档，但这些都替代不了阅读代码这个基础活动。

阅读代码是不可或缺的能力。

为什么这么说？因为：代码即文档，代码是理解一致性更强的文档。

另外，作为一个小补充，我们需要指出的一点是：阅读代码的结果，有时不一定仅仅是架构设计文档的补充与完善。我们有时也会顺手修改几行代码。

这是正常现象，而且应该被鼓励。为什么鼓励改代码？是因为我们鼓励随时随地消除臭味。改几行明显风格不太好的代码，是非常好的一件事情。

但是我们也要有原则。

其一，不做大的改动，比如限定单个函数内的改动不能超过 10 行。

其二，确保改动前后的语义完全一致。这种一致需要包括所有 corner case 上的语义一致，例如错误码，条件语句的边界等。

其三，不管多自信，有改动就需要补全相关的单元测试，确保修改代码的条件边界都被覆盖。

如果你对今天的内容有什么思考与解读，欢迎给我留言，我们一起讨论。下一讲我们谈谈“发布单元与版本管理”。

如果你觉得有所收获，也欢迎把文章分享给你的朋友。感谢你的收听，我们下期再见。

上一篇　70 | 怎么写设计文档?

下一篇　72 | 发布单元与版本管理

精选留言 (5)

写留言

Vackine
2020-01-07

终于等到😄

展开

3

霜花香似海
2020-01-08

实际工作中，很少有阅读代码的时间，或者精力。一般都是赶时间完成需求，只能自己私下里去看代码，修改代码都是需要权限和测试的，即使修改之后更完美，为了不影响使用大多数都选择谨慎。但阅读代码确实是必须的，是成长的一部分

展开

作者回复: 接手代码还没有搞懂就改，会出大问题。所以阅读代码是工作的重要组成部分。

1

细雨平湖
2020-01-08

就指导如何阅读源代码而言，这篇是目前我读到的最好的文章！感谢老师！

Aaron Cheung
2020-01-07

不做大改动 语义一致 能补单元测试补单元测试

展开

思维
2020-01-07

1、阅读代码的目的 2、要有产出

展开