用户故事 易昊：程序员不止有Bug和加班，还有诗和远方

你好，我是编辑王惠。在处理这门课的留言时，我注意到易昊同学一直在跟随着宫老师的脚步，学习和实践编译原理的相关知识，留言的内容十分有见地、提出的问题也能看出是经过了他深入的思考。同时，咱们这门课也具有很强的互动性，所以我邀请他来和我们分享一下他的心得体会。

Hi，我是易昊，目前在武汉做Android开发，已经工作12年了。很高兴能在这里跟你分享，关于我学习编译原理的一些心得体会。

为什么我要再学编译原理?

首先，我想给你解释一下，我为什么会起“程序员不止有Bug和加班，还有诗和远方”这样一个标题呢?

这是因为，作为一名应用开发者，我经常会觉得，自己只是在和源源不断的Bug以及项目进度作斗争，日常工作好像已经无法给我带来技术上的成就感了。但我能肯定的是，我对于技术的情怀并没有消失。我也认为，我不应该只满足于完成日常的普通开发任务，**而是应该去做点更有挑战性的事情，来满足自己精神上的追求。**

那么，我为什么会选择学习编译技术呢?

首要的原因，是这门课的内容不像编程语言、数据结构那样清晰直观。加上在大学时期，学校安排的课时较短，只有半个学期，自己又没有对它足够重视起来，导致这门课只学了个一知半解，从而造成自己对计算机底层工作原理没有掌握透彻，留下了遗憾。

《程序员的自我修养–链接、装载与库》里，有句话让我印象深刻：“真正了不起的程序员对自己程序的每一个字节都了如指掌。”虽然说得有点夸张，但一个优秀的程序员确实应该理解自己的程序为什么能在计算机上运行起来。而不是在写完代码后，忐忑不安地看着IDE在进行编译，等终于能运行起来时，大喊一声：“哇，能编译运行了，好神奇”。所以工作了几年之后，我一直想找机会能够弥补一下大学时期的遗憾，把编译知识学扎实。

另外，**编译技术的巨大挑战性，也是我想重拾学习的重要原因之一。**

你可能听过一个段子：程序员的三大浪漫，是自己实现编译器、操作系统和渲染引擎。

其实一开始，我并不理解为什么编译器会在这其中，我猜大概是因为特别困难吧。

我刚接触编程的时候，觉得能学好C、Java这样的编程语言，已经很不容易了，更何况要自己去用编译器实现一门语言。

而且编译原理这门课中，还有相当多比较深奥、晦涩的理论，想要掌握起来非常困难，它完全不像学习普通技术那样，几行代码就能运行一个hello world。同时你光有理论又完全不够，编译器中包含了很多巧妙的工程化处理。比如，会用抽象语法树来表示整个代码的语法结构；在函数调用时，参数和返回值在栈帧中的内存地址顺序等。

所以，不得不说，编译器是一个非常复杂的工程，编译原理是一个非常不容易消化、掌握的基础技术，让人望而生畏。

但是一旦掌握了之后，可以说就打通了计算机技术的任督二脉，因为你已经掌握了计算机运行中相当底层部分的原理，那么再去看其他技术就不会有什么大的障碍了。

而且在工作中，即使没有什么机会需要自己去创造语言或者写编译器，编译原理对我也很有帮助。举个例子吧，Android有很多动态化的技术，像ReactNative、Weex都会使用JavaScript作为脚本语言来调用原生接口，那么为了实现原生语言（如Java、Objective-C）和JavaScript的通信，ReactNative和Weex都在框架中嵌入了JavaScriptCore这个组件，因为JavaSciptCore包含了一个JavaScript的解释器，可以在运行时执行JavaScript。

那么，如果你也想在自己的项目中使用类似的动态化技术，但又不愿意被ReactNative和Weex技术绑死，就需要自己去剖析JavaScriptCore的工作原理，甚至要去实现一个类似的脚本语言执行框架。

而真的要自己去下载JavaScriptCore的源码来看，你就会发现，如果没有一定的编译原理知识作为基础，是很难看懂的。但是如果你具备了编译原理知识基础，其实会发现，这些源码里面也不外乎就是词法分析、语法分析、IR等，这些都是编译原理中的常用概念和算法。

还有就是日常工作中会碰到的某些比较棘手的问题，**如果你不理解编译技术，可能就无法找到出现问题的根源。**

比如我早期的工作中，在开发C++代码的时候，经常会遇到链接时找不到符号的问题，那个时候我对“符号是怎么产生的”并不理解，所以遇到这类问题，我就会在IDE或者代码里盲目地尝试修改。

后来重新学习了编译技术之后，我就理解了编译器在编译过程中会产生符号表，每个符号包含了符号的名称、类型、地址等信息。之所以出现这类问题，可能是依赖的静态库没有包含这些

符号，或者是类型不正确，再或者是这些符号的地址无法被正确解析，所以我用相应的工具去检查一下静态库文件的符号表，一般就可以解决问题了。

你看，编译技术总能帮我解决一些，我之前挠破头都想不出解决方案的问题。到现在我工作了十几年以后，就会有一个越来越强烈的感悟：**最重要的还是底层知识。**以前我也曾经感慨过计算机技术发展之快，各种技术层出不穷，就拿Android技术来说，各种什么插件化、组件化、动态化技术让人眼花缭乱，要不就是今天谷歌在提倡用Kotlin，明天又开始推Flutter了。

所以有一段时间我比较迷茫，“我究竟该学什么呢”，但我后来发现，虽然那些新技术层出不穷，但万变不离其宗，计算机核心的部分还是编译原理、操作系统那些底层技术。如果你对底层技术真正吃透了，再来看这些时髦的新技术，就不会感到那么神秘了，甚至是可以马上就弄明白它背后的技术原理。**原理都搞懂了，那掌握起来也就非常快了。**

我是怎么学习专栏的?

我开始意识到自己需要重新学习编译技术是在2015年，当时为了能够在Android的原生代码里运行JavaScript脚本，我依次尝试了WebView、JavaScriptCore、Rhino等方案，觉得这种多语言混合开发的技术挺强大，也许能够改变主流的应用开发模式，于是就想继续研究这些框架是怎么工作的。但是我发现，自己对编译原理知识掌握得不牢，导致学习这些技术的时候有点无从下手。

那个时候还没有极客时间这样针对性较强的学习平台，我是自己买了些书，有权威的龙书和虎书，也有《两周自制脚本语言》这样的速成书籍，但总是不得要领。

像龙书、虎书这样的，主要花了大量的篇幅在讲理论。但面对工作和家庭的压力，又不允许我有那么大片的时间去学习理论，而且没有实践来练习的话，也很容易忘掉。

像速成书籍这样的，虽然实现了一个很简单的编译器，但它大量的篇幅是在讲代码的一些细节，原理又讲得太少，知其然而不知其所以然。后来我就没有继续深入地学下去了。

直到偶然地在极客时间上看到了《编译原理之美》这门课，我简单看了下目录之后，就立马买了下来，因为感觉非常实用，**既对理论部分的重点内容有深入浅出地讲解，也有与实际问题的紧密联系。**学完这门课后，我感觉有信心去尝试写一点东西了，正好临近春节，就计划着在春节期间**自己写些代码来验证学习的成果。**结果不成想遇到了疫情封城，因为没法复工，就索性在家自己照着龙书和课程中给出的思路，看看能否自己实现个编译器出来。

结果我花了快两个月的时间，真的写出来了一个简单的编译器，能够把类似C语言风格的代码编译成汇编执行。

回想那段时间，虽然疫情很让人焦虑，但当我全身心地投入到对编译技术的钻研时，可以暂时忘记疫情带来的困扰。通过写这个小项目，我算是对编译器的工作过程有了个切身的体会，还

把多年未碰的汇编又重新拾起来投入使用，可以说是收获颇丰，疫情带给我的回忆也没有那么痛苦了，从另一个角度看，甚至还有一定的成就感。

这个简单的编译器项目完成了之后，就激发了我更大的兴趣。因为我毕竟只是实现了一个玩具型的编译器，那么工业界的编译器是如何工作的呢？

这个编译器，我是用LL算法来实现的语法分析，但龙书上还大篇幅地讲了LR、SLR、LALR，那么实际中到底是使用LL算法还是LR算法呢？

还有，我的编译器没有什么优化的功能，真实的编译器都有哪些优化措施呢？

这些问题吸引着我要去寻找答案。结果正巧，宫老师又推出了《编译原理实战课》，**深入浅出地讲解各大编译器的工作原理**，这可正好对我的胃口，于是又毫不犹豫地买下了。

上了几节课之后，觉得收获很大，特别是讲解javac和Graal编译器的部分。这两部分都给出了如何基于源码去剖析编译器的原理，实际操作性很强，**我跟着宫老师给出的步骤，下载和一步步地调试、跟踪源码**，印象十分深刻。特别是宫老师介绍的javac在处理运算符优先级时，引入了LR算法，从而避免了教科书上介绍的：当使用LL方式时，为了消除左递归，要写多级Tail函数的问题，这些都让我对编译技术的实际应用理解得更加深刻了。在学习Graal时，我也是花了不少的时间去配置Windows环境，包括下载安装Kali Linux、OpenJDK等，才终于把Graal跑起来。通过这样的实际操作，体验到了“折腾”的乐趣，对动手能力也是一种锻炼。

除此之外，**课程中还有丰富的流程图、类图和思维导图，因此我可以按图索骥，去研究自己感兴趣的知识点，不用再苦苦地从海量的源码中去大海捞针，学习效率得到了很大提升。**

如何更好地学习编译原理？

通过这段时间的学习后，我发现，编译原理其实并没有想象中的那么困难。我觉得计算机技术的一个特点就是像在搭积木，有时候只是知识点多、层次关系复杂而已。

编译技术尤其如此，从前端到后端，从词法分析到语法分析到语义分析、IR，最后到优化。这些地方包含着各种知识点，但这些知识也不是凭空变出来的，而是环环相扣、层层叠加出来的。**因此你在学习编译技术时一定要静下心来，一点一点地去吃透里面的技术细节，并且还需要不断地总结和提炼，否则对知识的理解可能就不够透彻，浮于表面。**

另外，你需要多去动手实践，特别是和算法相关的部分。比如，在编译器的语法分析阶段，有个内容是计算First集合和Follow集合，其实理解起来并不困难，但它包含的细节不少，容易算错，所以需要在课后多去练习。

我就是在课下，把宫老师布置的习题和龙书上相关的题目都算了一遍，还写了计算First集合和Follow集合的程序。不过就算是做到了这些，我感觉对这部分的理解还不够透彻，但是我对编

译技术的恐惧感已经消除了，对后续进一步的深入挖掘也打下了基础。所以说，静下心来学，勤动手去练，对学习编译原理这门课程来说非常重要。

我还有一个体会就是，**学习编译原理没有捷径可走**。因为编译技术是一个复杂而精密的工程，它的知识是环环相扣的。比如说，如果你对词法分析和语法分析的知识掌握得不够牢固，不熟悉一些常用语法规则的推导过程，那后面的IR、三地址代码、CFG等，你就会学得一头雾水。

所以，我们只能够一步一个脚印地去学习。

当然了，如果你和我一样是一个有家有口的上班族，只利用碎片时间可能不好做到连续学习，那我建议你在学习课程的时候，可以稍微花点时间去**参考一下宫老师介绍的开源编译器代码**，比如JavaCompiler的源码、Graal的源码。这样就可以和课程中的内容结合起来。因为看代码和调试代码会更加直观，也更容易理解。

同时，**你也可以看看别人的代码设计**，有哪些地方是可以做成组件的，哪些函数是可以自己去实现来练手的。然后，你可以先给自己定一个小目标，就是利用业余时间去完成这些组件或者函数的开发，因为单个组件的工作量并不是太大，因此还是可以尝试完成的。这样在巩固理论知识的同时，还能锻炼自己的动手能力，我想热爱编程的你，应该可以从中得到不少快乐。

我相信，只要最终坚持下来，你和我，都可以掌握好编译这门“屠龙”技术。

上一页 下一页