结束语-实战是唯一标准！

你好，我是宫文学。

转眼之间，“编译原理实战课”计划中的内容已经发布完毕了。在这季课程中，你的感受如何？有了哪些收获？遇到了哪些困难？

很多同学可能会觉得这一季的课程比上一季的“编译原理之美”要难一些。不过为什么一定要推出这么一门课，来研究实际编译器的实现呢？这是因为我相信，**实战是检验你是否掌握了编译原理的唯一标准，也是学习编译原理的真正目标。**

计算机领域的工程性很强。这决定了我们学习编译原理，不仅仅是掌握理论，而是要把它付诸实践。在我们学习编译原理的过程中，如果遇到内心有疑惑的地方，那不妨把实战作为决策的标准。

这些疑惑可能包括很多，比如：

- 词法分析和语法分析工具，应该手写，还是用工具生成？
- 应该用LL算法，还是LR算法？
- 后端应该用工具，还是自己手写？
- 我是否应该学习后端？
- IR应该用什么数据结构？
- 寄存器分配采用什么算法比较好？
- ……

上述问题，如果想在教科书里找到答案，哪怕是“读万卷书”，也是比较难的。而换一个思路，“行万里路”，那就很容易了。你会发现每种语言，因为其适用的领域和设计的目标不同，对于上述问题会采用不同的技术策略，而每种技术策略都有它的道理。从中，你不仅仅可以为上述问题找到答案，更重要的是学会权衡不同技术方案的考虑因素，从而对知识点活学活用起来。

我们说实战是标准。那你可能会反问，难道掌握基础理论和原理就不重要了吗？这是在很多领域都在争论的一个话题：理论重要，还是实践重要。

理论重要，还是实践重要？

理论和原理当然重要，在编译原理中也是如此。形式语言有关的理论，以及前端、中端和后端的各个经典算法，构筑了编译原理这门课坚实的理论基础。

但是，在出现编译原理这门课之前，在出现龙书虎书之前，工程师们已经在写编译器了。

你在工作中，有时候就会遇到理论派和实践派之争。举例来说，有时候从理论角度，某一个方案会“看上去很美”。那到底是否采用该方案呢？这个时候，就需要拿实践来说话了。

我拿Linux内核的发展举个例子。当年Linus推出Linux内核的时候，并没有采用学术界推崇的微内核架构，为此Linus还跟Minix的作者有一场著名的辩论。而实践证明，Linux内核发展得很成功，而GNU的另一个采用微架构的内核Hurd发展了20多年还没落地。

客观地说，Linux内核后来也吸收了很多微内核的设计理念。而声称采用微内核架构的Windows系统和macOS系统，其实在很多地方也已经违背了微内核的原则，而具备Linux那样的单内核的特征。之所以有上述的融合，其实都是一个原因，就是为了得到更好的实用效果。所以，实践会为很多历史上的争论划上句号。

在编译技术和计算机语言设计领域，也存在着很多的理论与实践之争。比如，理论上，似乎函数式编程更简洁、更强大，学术界也很偏爱它，但是纯函数的编程语言，至今没有成为主流，这是为什么呢？

再比如，是否一定要把龙书虎书都读明白，才算学会了编译原理呢？

再进一步，如果你使用编译技术的时候，遇到一个实际的问题，是跟着龙书、虎书还有各种课本走，还是拿出一个能解决问题的方案就行？

在课程里，我鼓励你抛弃一些传统上学习编译原理的困扰。如果龙书、虎书看不明白，那也不用过于纠结，这并不会挡住你学习的道路。多看实际的编译器，多自己动手实践，在这个过程中，你自然会明白课本里原来不知所云的知识点。

那么如何以实践为指导，从而具备更好的技术方案鉴别力呢？在本课程里，我们有三个重点。包括研究常用语言的编译器、从计算机语言设计的高度来理解编译原理，以及从运行时的实现来理解编译原理。

对于你所使用的语言，应该把它的编译器研究透

这门课程的主张是，你最好把自己所使用语言的编译器研究透。这个建议有几个理由。

第一，因为这门语言是你所熟悉的，所以你研究起来速度会更快。比如，可以更快地写出测试用的程序。并且，由于很多语言的编译器都已经实现了自举，比如说Go语言和Java语言的编译器，所以你可以更快地理解源代码，以及对编译器本身做调试。

第二，这门语言的编译器所采用的实现技术，一定是体现了该语言的特性的。比如V8会强调解析速度快，Java编译器要支持注解特性等，值得你去仔细体会。

第三，研究透编译器，会加深你对这门语言的理解。比如说，你了解清楚了Java的编译器是如何处理泛型的，那你就会彻底理解Java泛型机制的优缺点。而C++的模板机制，对于学习C++的同学是有一定挑战的。但一旦你把它在编译期的实现机制弄明白，就会彻底掌握模板机制。我也计划在后续用一篇加餐，把C++的模板机制给你拆解一下。

那么，既然编译器是为具体语言服务的，所以，我们也在课程里介绍了计算机语言设计所考虑的那些关键因素，以及它们对编译技术的影响。

从计算机语言设计的高度，去理解编译技术

在课程里你已经体会到了，语言设计的不同，必然导致采用编译技术不同。

其实，从计算机语言设计的高度上看，编译器只是实现计算机语言的一块底层基石。计算机语言设计本身有很多的研究课题，比如类型系统、所采用的编程范式、泛型特性、元编程特性等等，我们在课程里有所涉猎，但并没有在理论层面深挖。有些学校会从这个方向上来培养博士生，他们会在理论层面做更深入的研究。

什么样的计算机语言是一个好的设计？这是一个充满争议的话题，我们这门课程尽量不参与这个话题的讨论。我们的任务，是**要知道当采用不同的语言设计时，应该如何运用编译技术来落地**。特别是，**要了解自己所使用的语言的实现机制**。

如果说计算机语言设计，是一种偏理论性的视角，那么程序具体的运行机制，则是更加注重落地的一种视角。

从程序运行机制的角度，去理解编译技术

学习编译原理的一个挑战，就在于你必须真正理解程序是如何运行的，以及程序都可以有哪几种运行方式。这样，你才能理解如何生成服务于这种运行机制的目标代码。

最最基础的，你需要了解像C语言这样的编译成机器码直接运行的语言，它的运行机制是怎样的。代码放在哪里，又是如何一步步被执行的。在执行过程中，栈是怎么变化的。函数调用的过程中，都发生了些什么事情。什么数据是放在栈里的，什么数据是放在堆里的，等等。

在此基础上，如果从C语言换成C++呢？C++多了个对象机制，那对象在内存里是一个什么结构？多重继承的时候是一个什么结构？在存在多态的时候，如何实现方法的正确绑定？这些C++比C语言多出来的语义，你也要能够在运行时机制中把它弄清楚。

再进一步，到了Go语言，仍然是编译成机器码运行的，但跟C和C++又有本质区别。因为Go语言的运行时里包含了垃圾收集机制和并发调度机制，这两个机制要跟你的程序编译成的代码互相配合，所以编译器生成的目标代码里要体现内存管理和并发这两大机制。像Go语言这种特殊的运行机制，还导致了跨语言调用的难度。用Go语言调用C语言的库，要有一定的转换和开销。

然后呢，语言运行时的抽象度进一步增加。到了Java语言，就用到一个虚拟机。字节码也正式登台亮相。你需要知道栈机和寄存器机这两种不同的运行字节码的解释器，也要知道它们对应的字节码的差别。而为了提升运行速度，JIT、分层编译和逆优化机制又登场，栈上替换（OSR）技术也产生。这个时候，你需要理解解释执行和运行JIT生成的本地代码，是如何无缝衔接的。这个时候的栈桢，又有何不同。

然后是JavaScript的运行时机制，就更加复杂了。V8不仅具备JVM的那些能力，在编译时还要去推断变量的类型，并且通过隐藏类的方式安排对象的内存布局，以及通过内联缓存的技术去加快对象属性的访问速度。

这样从最简单的运行时，到最复杂的虚拟机，你都能理解其运行机制的话，你其实不仅知道在不同场景下如何使用编译技术，甚至可以参与虚拟机等底层软件的研发了。

不再是谈论，来参与实战吧！

今天，我们学习编译原理，目标不能放在考试考多少分上。中国的技术生态，使得我们已经能够孕育自己的编译器、自己的语言、自己的虚拟机。方舟编译器已经带了个头。我想，中国不会只有方舟编译器孤军奋战的！

就算是开发普通的应用软件，我们也要运用编译技术，让它们平台化，让中国制造的软件，具有更高的技术含量，颠覆世界对于“中国软件”的品牌认知。这样的颠覆，在手机、家电等制造业已经发生了，也应该轮到软件业了。

而经验告诉我们，一旦中国的厂商和工程师开始动起来，那么速度会是非常快的。编译技术并没有多么难。

我相信，只要短短几年时间，中国软件界就会在这个领域崭露头角！

这就是我们这门课程的目的。不是为了学习而学习，而是为了实战而学习。

当然，课程虽然看似结束了，但也代表着你学习的重新开始。后面我计划再写几篇加餐，会针对C++、Rust等编译器再做一些解析，拓展你的学习地图。并且，针对方舟编译器，我还会进一步跟你分享我的一些研究成果，希望我们可以形成一个持续不断地对编译器进行研究的社群，让学习和研究不断深入下去，不断走向实用。

另外，我还给你准备了一份毕业问卷，题目不多，希望你能在问卷里聊一聊你对这门课的看法。欢迎你点击下面的图片，用1～2分钟时间填写一下，期待你畅所欲言。当然了，如果你对课程内容还有什么问题，也欢迎你在留言区继续提问，我会持续回复你的留言。

我们江湖再见！