



下载APP



42 | 到这里，我们的收获和未尽的工作有哪些？

2021-11-19 宫文学

《手把手带你写一门编程语言》

课程介绍 >



讲述：宫文学

时长 19:18 大小 17.69M



你好，我是宫文学。

到今天这节课为止，我们已经把这门课程的主要内容都学完了。感谢你一路的坚持！

所以，在今天这节课，我想做一个简单的总结。我想先带你回顾一下我们一起闯过的那些技术关卡，以及取得的成果。接下来，我还想梳理一下我们尚未完成的工作，也对我们后续作为开源项目的 PlayScript 语言做一下规划。

在这个过程中，你可以暂时从技术细节中解脱出来，站在一个语言的架构师的角度，做一些高层面的思考，锻炼一下架构思维。



首先，我们简单总结一下当前已经完成的工作。

当前的收获

到目前为止，我们在 40 多节课的内容里，塞进了大量的知识点。我们按课程顺序来梳理一下。

基础篇：三大关卡

在第一部分基础篇中，我带你连续闯了三个关卡。

第一个关卡，是编译器前端技术，包括词法分析、语法分析和语义分析技术。

在词法和语法分析方面，我们这门课没有带你进入相关算法的迷魂阵，而是带你去掌握最佳实践。一方面，这些算法我在《编译原理之美》课程中已经讲过了。另一方面，如果你只是写个编译器，而不是写个像 Yacc、Antlr 这样的编译器生成工具，其实不需要深究那些算法，只要大概明白原理就行了。

即使是这样，对于递归下降中的左递归问题这样偏理论性的知识点，很多同学免不了还是有疑惑。比如，有同学会问我，我在课程里用到的有些文法，为什么仍然有一些是左递归的呢？这里其实涉及到 PEG 文法的一个知识点，我会在后面的加餐里讲一下 PEG。其实，并不是所有的左递归都没有办法处理。关于左递归，直到现在仍然是做算法研究的人感兴趣的一个领域。

在语义分析方面，我们体会了如何建立符号表、如何做引用消解、如何检查一些语义错误的过程，这样，你会对课本中讲到的一些抽象概念建立具象的理解。

为了编译 TypeScript 语言，编译器在语义分析阶段最重要的功能是进行类型的处理，其中的关键点又是类型的联合和窄化。在这门课里，我们主要采用了集合运算的手法来处理类型。在 PlayScript 开源项目中，我计划把这部分进一步优化，让类型计算更简洁、更精准。

对很多同学来说，闯过编译器前端的这个关卡，其实已经收获满满，可以在自己的项目里大展身手了。不过，如果你喜欢钻研底层实现，显然还不会满足止步于此，那你就可以继续闯第二关。

在第二关，我们实现了两个版本的字节码虚拟机。一个版本是用 TypeScript 实现的，另一个版本使用 C 语言实现的。

通过这个实现过程，你会了解到像 Java 这样的成熟语言的字节码是如何设计的，又是如何实际运行的。这样，在需要的时候，我希望你能够敢于自己生成 Java 或 .NET 的字节码，实现自己想要的软件编程功能。

并且，通过实现 C 语言版本的虚拟机，我们也能初步了解运行时的功能。特别是，你要知道，如果我们不能做好内存的管理，系统运行的性能就会大受影响。而且，通过我们多次的性能测试，你应该已经对一个解释器中影响性能的因素产生了直观的理解，这样你自己写程序的时候，也能够进行更明智的决策。

在初步实现了字节码解释器以后，我们又进入了第三关，挑战把源代码编译成二进制的可执行程序。

为了完成第三关的任务，我们必须对程序的运行机制有深入的了解，包括程序运行跟硬件是什么关系，跟操作系统和 ABI 又是什么关系。再进一步，我们还需要了解 CPU 架构和它支持的指令集，学会阅读甚至手写汇编代码。

生成汇编代码有两个关键点：第一，要熟悉 ABI，正确地维护栈帧和寄存器的状态，否则程序运行时就会报 segment 错误；第二，就是要实现寄存器分配算法。你要知道，不同的指令集会使用不同的寄存器，并且我们在函数调用前后要保护某些寄存器。做好这些以后，程序就可以充分利用寄存器，飞一般地运行了。

在闯完这三关以后，你已经从前端到后端打通了技术路线。接下来在第二部分进阶篇中，我们把这条路线做了拓宽。

进阶篇：拓宽路径

怎么去做拓宽呢？我们的主线是支持更多的数据类型，包括浮点数、字符串和数组这几个基础类型，还包括自定义对象、高阶函数这种高级的类型。

为了支持这些类型，我们必须增强运行时的功能，需要设计对象的内存布局。在访问对象属性、数组成员的时候，我们也要能够正确计算出内存地址来。

进阶篇最难的部分，是**自动内存管理功能**，包括基于 Arena 的内存申请机制，以及垃圾回收机制。而实现垃圾回收的关键点，在于找到 GC 根，并顺着 GC 根去找到直接引用和间接引用的对象。因此，我们需要保存栈帧的布局信息、对象的元数据信息、闭包的元数据信息等各种静态信息。

如果你充分掌握了内存管理涉及的技术点，那么你在后面实现很多高级功能的时候都能用上。比如对程序做调试、支持运行时的类型判断和元编程功能，都需要用到我们提前保存的元数据信息。

学完第二部分以后，你对实现像面向对象、函数式编程等各种语言特性基本上心里有数了。在第三部分优化篇，我们就专注于解决一个问题，就是优化问题。

优化篇：基于图的 IR

优化是编译器最重要的功能之一。优化可以发生在全生命周期，包括前端、中端、后端和运行过程中。第三部分的核心，就是一个基于图的 IR。这个 IR 被 Java 和 JavaScript 的编译器采用，你可以想象一定有它的优势。

在使用这个 IR 的过程中，我们确实发现，我们很多优化的实现都变得很简单了。像公共子表达式删除，我们在生成 IR 的过程中顺带着就能完成。还有，像拷贝传播、死代码删除、值编号等这些优化，实现起来也很简单。最重要的是，这个 IR 更容易打通本地优化、全局优化和过程间优化三者的边界，让代码更容易在不同的基本块中移动，获取我们想要的优化效果。

并且，我们还了解了基于这个 IR 不断地做 lower，直到生成 LIR，然后基于 LIR 做指令选择、寄存器分配等后端优化的完整过程。通过这一部分的学习，你对于前端、中端、后端优化要做的工作就都比较清晰了。

好了，以上就是这门课中我们领略的各种技术风光，我希望你能够充分掌握，一方面这能开阔你的技术思路，另一方面这些技术也能用到你的实际项目中。

不过，受限于时间，我还没有把一门完整的语言完全实现完。所以，我后面会把这门课的示例代码，作为一个开源项目继续迭代下去，并形成完整的、实用的版本。至于当前已有的基础，我们就把它作为 0.1 版本吧！

那如果我们要实现一个实用的版本，还有哪些工作要做呢？

后续工作

第一，我们要对编译器前端做比较大的增强和重构。

首先，当前我们已有的词法分析、语法分析和语义分析功能，都要支持更多的特性，比如，除了我们已经支持的 for 循环和 if 分支语句外，还有 while 循环、switch 语句，等等。

而且，我们对编译错误的处理要更加友好。你应该也感受到了，目前我们的编译器，在遇到某些语法错误的时候，会持续不停地尝试，不断打印错误信息。这显然太不友好了，要做优化。

然后还有一个比较大的工作，就是对类型系统进行升级。我们要重构一下我们之前类型计算的算法，让它变得更加简洁和准确。我们目前使用的 Nominal 的类型系统，也要修改成支持 structural 的类型系统，并且我们还要让我们的类型计算支持泛型。

另外，我们在升级编译器前端的时候，对 AST 和符号表这两个重要的数据结构，也需要重构和优化一下。

第二，我们要把面向对象和函数式编程的特性实现完整。

比如，面向对象方面，我们需要实现严格的对象的初始化流程，需要支持访问权限，还要支持接口。在函数式编程方面，怎么着也要把 Lambda 表达式这些基础功能实现。这主要是工作量的问题，但需要前端、中端、后端和运行时各方面的配合。

第三，是升级编译器的中端优化功能。

基于目前的 IR，我们只实现了少量的优化，并且还没有支持面向对象等复杂的语言特性，这些都需要进行扩展和支持。在这个实现的过程中，我们 IR 的数据结构也会得到丰富和完善。

第四，是升级编译器的后端功能。

我们之前的编译器后端主要是基于 AST 来生成汇编代码。所以，在引入 IR 以后，我们编译器的后端也需要重构一下。从 AST 生成 IR 后，再基于 HIR 生成 LIR，然后在 LIR 的基础上重新实现指令选择和寄存器分配。

另外，我们目前只支持 x86-64 架构，并且也没有在多个操作系统上做测试。在后面，我们要支持至少两种 CPU 架构，我计划先支持的是 x86-64 和 Aarch64。前者被广泛用于 PC 和服务端中，后者被广泛用于智能手机和苹果新一代 Mac 电脑中。而且，我们还要兼容多种操作系统。

第五，是内存管理方面的升级。

在垃圾收集方面，我们的 GC 还是很基础的，达不到实用级别。那么，接下来我们首先要完善基本的标记 - 清除算法。之后，我计划实现一个自动引用计数的（ARC）的机制。

ARC 的原理是记录每个对象的引用数，当引用数为零的时候，就自动作为垃圾清理掉。ARC 的好处就是垃圾回收不会引起大的停顿，能让系统的响应比较平缓。苹果的 Objective-C 和 Swift 都采用了 ARC，这也是苹果的系统很少卡顿的原因之一。

第六，实现并发机制。

你应该也注意到了，目前我们这门课中并没有涉及并发机制。但如果不实现并发机制，显然会是一个遗憾。所以后面，我会给出协程功能的一个参考实现。至于另外的并发功能的设想，我接下来还会介绍到。

好了，上面就是要实现一个完整的、实用的、静态编译的语言会涉及到的工作量，细细看一下，还真是挺大的。不过，你也不用怵，这里并没有太多技术点是我们这门课没有涵盖到的，更多的是工作量和工程化的问题。

花费这么多的工作量，我并不完全是为了兴趣爱好，或者是做技术验证，还是想未来有一天，能把它用于一些实际的应用场景中。那接下来，我就谈谈对开源的 PlayScript 语言项目的一些规划。

对 PlayScript 的规划

为什么我会产生自己动手实现一门语言的想法呢？这其实是出于一些实际的需求。现有的语言，或者已有语言的现有实现，有时会让我很不满意。所以我就想，与其等着别人来满足我的需求，不如自己动手试试看。

我就分享一下我的这几个需求，看看你是否也遇到过类似的问题。

首先，我对后端编程语言不满意。

你知道吗？要实现像微信这样的应用的后端，你只能使用 C++ 这样的语言。并且，微信团队还开发了自己的协程库，才能应对海量并发的需求。我们每天用微信，觉得它总是会实时响应，其实后端的挑战是巨大的。你想想看就知道，成亿的用户，加上成亿的并发，绝对是顶级的技术挑战。

Java、Go 等典型的后端语言，都不能满足这种场景的需求。Java 的内存占用太大，自带的并发机制只有线程。虽然 Go 语言好一点，但它和 Java 都有一个致命的弱点，就是垃圾回收导致的停顿是不可控的。这对于微信这种大型的、高并发的平台，会带来灾难性的后果。但是，让普通的技术团队用 C++ 开发应用，门槛有点高。

而且，我对现有后端语言提供的可靠性也不满意。在这方面，我比较喜欢 Erlang，它的并发机制和其他特性结合起来，能提供 9 个 9 的可靠性。我觉得，如果每个应用都能实现这么高的可靠性就好了。但是，我对 Erlang 的性能又不满意，而且它的语法对于大多数程序员来说，也不是太友好。

实现一门高可靠性的语言，其实有隐含一个需求，就是语言中的功能是能够在运行时被动态替换的，因为你没有办法停下整个系统。所以，我们不能仅仅实现 AOT 的功能，还要有 JIT、动态优化、动态部署、动态 Dispatch 的功能。

所以，我理想的后端语言，是能够用比较低的成本，开发出高并发、高可靠性、资源消耗低的应用。不知道你是不是也有类似的需求呢？

第二，我对前端编程环境也不太满意。

我感觉，现在的前端编程环境太碎片化了，包括浏览器、Android、IOS、Windows、macOS、Linux 等不同的平台，而且国内还有好几个不同的小程序平台。

所以，我想要是有一个语言或者工具，能够开发一次，部署到很多客户端，那就好了。

第三，我对企业应用的编程语言也不满意。

我曾经参与过很多企业应用的开发工作。在企业应用的开发中，很多时候我们要更关注业务逻辑。但是，现在很多应用的业务逻辑和技术逻辑都是混杂的，企业应用的开发成本太高了。

我一直认为，如果你是做企业软件的厂商，那应该有相应的开发语言才好，比如，德国的 SAP 就有自己的 ABAP 语言。不过，这个语言在现代的应用架构下已经过时了，用 ABAP 开发不出很容易横向扩展的应用。再有一点，这个语言是企业私有的，不是公共的。

当然，现在我们已经迎来了低代码开发的一波浪潮。但是，如果低代码工具的开发不是像微软这么有实力的厂商，很难维护一个完全私有的生态。这样的情况下，客户在你私有的平台上开发应用，就是比较有风险，所以还是应该有一个公共开放的语言。

而且，就算是低代码开发，我也希望是基于某个语言的，而不是仅仅提供一些图形化的定制工具。最好呢，是语言可以转化为图形，图形也可以转化为代码。这种代码和图形化表达双向转化的能力，我在华为的 HarmonyOS 开发工具上看到了，感觉很喜欢。其实我觉得，低代码开发工具也应该实现类似的功能才算合格，这种能把应用表达为代码的能力，是保证应用的可移植性、保护企业投资的关键。

第四，如有可能，我希望能让物联网应用的开发变得更简单一点。

我在之前的一篇加餐里，介绍过工控领域软件开发的情况。工控领域的技术原来叫做 OT，它们的技术跟 IT 是不一样的。但在我看来，现在很多 IT 技术可以进入 OT 领域。比如，现在儿童编程都可以用一个图形化开发工具控制机器人，这跟控制发电机、控制高铁，其实没有本质的区别。所以，我们应该也可以把这两个领域的开发工具打通才对。就算是 OT 强调高可靠性，但其实，IT 里现在就有更高可靠性的技术，比如我前面提到过的 Erlang 的 9 个 9 的可靠性。

另外，OT 的技术生态，原来都把控在少数外国的企业手里。我觉得，如有可能，我们最好可以搞搞破坏，把它搞成一个开放的生态。

最后，现在工控应用和消费级的物联网应用，开发成本还是太高了。我觉得，我们应该可以用 TypeScript 这样简单的语言，就能生成 footprint 很小、性能很高、能够方便地跟硬件接口打交道的应用才对。

上面就是我对新的语言和开发工具的一些需求，也是我对现有的语言和工具不满意的地方。但我目前只把 PlayScript 作为一个开源的兴趣项目，不一定能完全实现这些目标。我现在只是把观点和期待在这里分享一下。

PlayScript 的项目地址在这里：<https://gitee.com/richard-gong/PlayScript>。我会把代码整理一下放进去，并保持更新。如果你对哪方面的特性感兴趣，也欢迎你一起参与。

最后，还有一个重要的观点需要说明一下：**实现一门语言和设计一门语言是两码事**。你也看到了，我们这门课讨论的都是如何实现语言，并没有讨论怎么去设计一门语言。一方面，我自己对设计一门语言并没有把握。另一方面，我觉得我的这些需求，似乎也并不需要设计一门全新的语言，而是对现有的某个语言重新做一个实现就行了。使用大家都已经熟悉的语法，对我来说似乎是更好的选择。

课程小结

今天这节课，我把我们这门课的内容做了一下总结，也梳理了我们后续要继续实现的功能。所以，这节课并没有什么需要你记住的硬知识点。

不过，通过对这门课程的学习，以及我们今天的梳理，我希望你能产生一个信心，就是**实现一门语言所需要的技术，其实并没有那么高不可攀**。其实到现在，我们真的已经涉猎了几乎所有的技术点，现在已经没有哪个技术点是真正有难度的、能够挡住你的。

在这个信心的基础上，我们再讨论 PlayScript 未来的演化，讨论需要一些新的语言或者开发工具的场景，才有意义。因为你要知道，你其实是有能力完成这些工作的。我们在中国的软件领域，可以做很多类似的、基础性的，甚至开创性的工作，不用什么事情都仰人鼻息。

思考题

今天，我分享了我的职业生涯中，对现有的语言和工具各种不满意，并且，我也分享了希望有人能够站出来改进的一些方面。你有没有类似的需求和感受呢？请你也跟我们分享一

下吧！

欢迎你把这节课分享给更多感兴趣的朋友。我是宫文学，我们下节课见。

分享给需要的人，Ta订阅后你可得 **20** 元现金奖励

 生成海报并分享

 赞 0  提建议

© 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。

上一篇 41 | 后端优化：生成LIR和指令选择

精选留言 (1)

 写留言



奋斗的蜗牛

2021-11-19

课程这么快就要结束了，期待老师后面的分享

展开 ∨

