

10 | 语言的实现：运行时，软件设计的地基

2020-06-15 郑晔

软件设计之美

[进入课程 >](#)



讲述：郑晔

时长 11:52 大小 10.88M



你好！我是郑晔。

通过前两讲的学习，相信你已经对程序设计语言有了全新的认识。我们知道了，在学习不同的程序设计语言时，可以相互借鉴优秀的设计。但是要借鉴，除了模型和接口，还应该有实现。所以，这一讲，我们就来谈谈程序设计语言的实现。

程序设计语言的实现就是支撑程序运行的部分：运行时，也有人称之为运行时系统，或运行时环境，它主要是为了实现程序设计语言的执行模型。

相比于语法和程序库，我们在学习语言的过程中，对运行时的关注较少。因为不理解语言的实现依然不影响我们写程序，那我们为什么还要学习运行时呢？



因为**运行时，是我们做软件设计的地基**。你可能会问，软件设计的地基不应该是程序设计语言本身吗？并不是，一些比较基础的设计，仅仅了解到语言这个层面是不够的。

我用个例子来进行说明，我曾经参与过一个开源项目：在 JVM 上运行 Ruby。这种行为肯定不是 Java 语言支持的，为了让 Ruby 能够运行在 JVM 上，我们将 Ruby 的代码编译成了 Java 的字节码，而字节码就属于运行时的一部分。

你看，**做设计真正的地基，并不是程序设计语言，而是运行时，有了对于运行时的理解，我们甚至可以做出语言本身不支持的设计**。而且理解了运行时，我们可以成为一个更好的程序员，真正做到对自己编写的代码了如指掌。

不过，运行时的知识很长一段时间内都不是显学，我初学编程时，这方面的资料并不多。不过，近些年来，这方面明显得到了改善，各种程序设计语言运行时的资料开始多了起来。尤其在 Java 社区，JVM 相关的知识已经成为很多程序员面试的重要组成部分。没错，JVM 就是一种运行时。

接下来，我们就以 JVM 为例，谈谈怎样了解运行时。

程序如何运行

首先，我们要澄清一点，对于大部分普通程序员来说，学习运行时并不是为了成为运行时的开发者，我们只是为了更好地理解自己写的程序是如何运行的。

运行时的相关知识很多，而**“程序如何运行”**本身就是一条主线，将各种知识贯穿起来。程序能够运行，前提条件是，它是一个可执行的文件，我们就从这里开始。

一般来说，可执行的程序都是有一个可执行文件的结构，对应到 JVM 上，就是类文件的结构。然后，可执行程序想要执行，需要有一个加载器将它加载到内存里，这就是 JVM 类加载器的工作。

加载是一个过程，那么加载的结果是什么呢？就是按照程序运行的需求，将加载过来的程序放到对应的位置上，这就需要了解 JVM 的内存布局，比如，程序动态申请的内存都在堆上，为了支持方法调用要有栈，还要有区域存放我们加载过来的程序，诸如方法区等等。

到这里，程序完成了加载，做好了运行的准备，但这只是静态的内容。接下来，你就需要了解程序在运行过程中的表现。一般来说，执行过程就是设置好程序计数器（Program Counter, PC），然后开始按照指令一条一条地开始执行。所以，重点是要知道这些指令到底做了什么。

在 Java 中，我们都知道，程序会编译成字节码，对于 Java 来说，字节码就相当于汇编，其中的指令就是 Java 程序执行的基础。所以，突破口就在于**了解指令是如何执行的**。

其实，大部分 JVM 指令理解起来都很简单，尤其在你了解内存布局之后。比如，加法指令就是在栈上取出两个数，相加之后，再放回栈里面。

我要提一个看上去很简单的指令，它是一根拴着牛的绳子，这就是 new，没错，就是创建对象的指令。那头牛就是内存管理机制，也就是很多人熟悉的 GC，这是一个很大的主题，如果展开来看的话，也是一个庞杂的知识体系。

有了对指令的理解，就对 Java 程序执行有了基本的理解。剩下的就可以根据自己的需要，打开一些被语法和程序库隐藏起来的细节。比如，synchronized 是怎样实现的，顺着这条线，我们可以走到内存模型（Java Memory Model, JMM）。

当然，这里的内容并不是为了与你详细讨论 JVM 的实现，无论是哪个知识点真正展开后，实际上都还会有很多的细节。

这里只是以 JVM 为例进行讲解，学习其他语言的运行时也是类似的，带着“程序如何运行”这个问题去理解就好了。只不过，每种语言的执行模型是不同的，需要了解的内容也是有所差异的。比如，理解 C 的运行时，你需要知道更多计算机硬件本身的特性，而理解一些动态语言的运行时，则需要我们对语法树的结构有一定认识。

有了对运行时的理解，我们就可以把一些好的语言模型借鉴到自己的语言中，比如，使用 C 语言编程时，我们可以实现多态，做法就是自己实现一个虚拟表，这就是面向对象语言实现多态的一种方案。

运行时的编程接口

我们前面说过，做软件设计的地基是运行时，那怎样把我们的设计构建在运行时之上呢？这就要依赖于运行时给我们提供的接口了。所以，我们学习运行时，除了要理解运行时本身的机制之外，还要掌握运行时提供的编程接口。

在 Java 中，最简单的运行时接口就是运行时类型识别的能力，也就是很多人熟悉的 `getClass`。通过这个接口，我们可以获取到类的信息，一些程序库的实现就会利用类自身声明的信息。比如，之前说过，有些程序库会利用 Annotation 进行声明式编程，这样的程序库往往会在运行的过程中，以 `getClass` 为入口进行一系列操作将 Annotation 取出来，然后做相应的处理。

当然，这样的接口还有很多，一部分是以标准库的方式提供的，比如，动态代理。通过阅读 JDK 的文档，我们很容易学会怎么去运用这些能力。还有一部分接口是以规范的方式提供的，需要你对 JVM 有着更好的理解才能运用自如，比如，字节码。


前面我们说了，通过了解指令的执行方式，可以帮助我们更好地理解运行时的机制。有了这些理解，再来看字节码，理解的门槛就大幅度地降低了。

如果站在字节码的角度思考问题，我们甚至可以创造出一些 Java 语言层面没有提供的能力，比如，有的程序库给 Java 语言扩展 AOP (Aspect-oriented programming, 面向切面编程) 的能力。这样一来，你写程序的极限就不再是语言本身了，而是变成了字节码的能力极限。

给你举个例子，比如，Java 7 发布的时候，字节码定义了 `InvokeDynamic` 这个新指令，当时语言层面上并没有提供任何语法。如果你需要，就可以自己编写字节码使用这个新指令，像 JRuby、Jython、Groovy 等一些基于 JVM 的语言，开发者就可以利用这个指令改善自己的运行时实现。当然 `InvokeDynamic` 的诞生，本身就是为了在 JVM 上更好地支持动态语言。

好消息是，操控字节码这件事的门槛也在逐渐降低。最开始，操作字节码是一件非常神秘的事情，在许多程序员看来，那是只有 SUN 工程师才能做的事情（那时候，Java 还属于 SUN）。

后来，出现了一个叫 [ASM](#) 的程序库，把字节码拉入了凡间，越来越多的程序员开始拥有操作字节码的能力。不过，使用 ASM，依然要对类文件的结构有所理解，用起来还是比较

麻烦。后来又出现了各种基于 ASM 的改进，现在我个人用得比较多的是  ByteBuddy。

有了对于字节码的了解，在 Java 这种静态的语言上，就可以做出动态语言的一些效果。比如，Java 语言的一些 Mock 框架，为什么可以只声明接口就能够执行，因为背后常常是动态生成了一个类。

一些动态语言为了支持自己的动态特性，也提供了一些运行时的接口给开发者。比如，Ruby 里面很著名的 `method_missing`，很多框架用它实现了一些效果，即便你未定义方法也能够执行的。你也许想到了，我们提到过的 Ruby on Rails 中各种 `find_by` 方法就可以用它来实现。

`method_missing` 其实就是一个回调方法，当运行时在进行方法查找时，如果找不到对应方法时就调用语言层面的这个方法。所以，你看出来了，这就是运行时和语言互相配合的产物。如果你能够对于方法查找的机制有更具体的了解，使用起来就可以更加地得心应手，就能实现出一些非常好的设计。

总结时刻

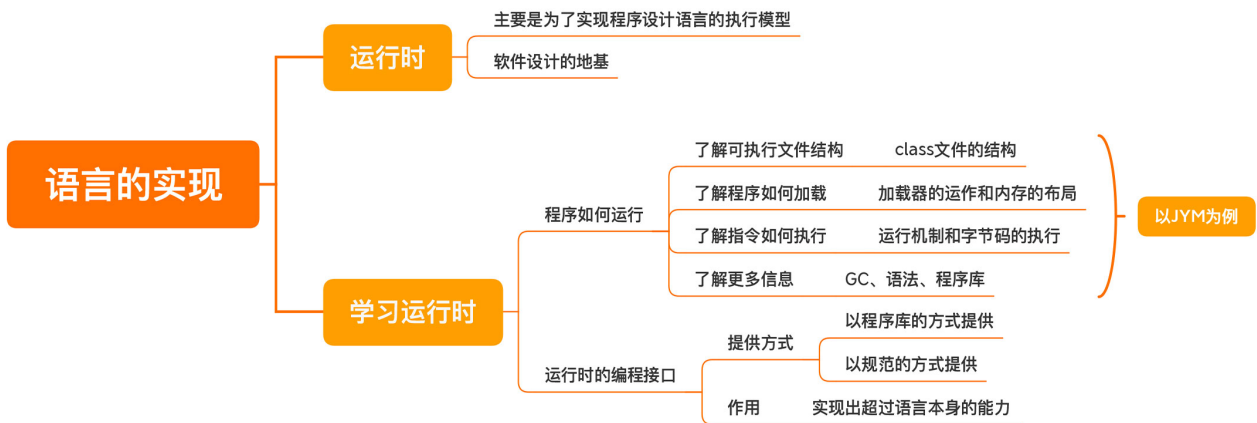
今天，我们讨论了程序设计语言的实现：运行时。**对于运行时的理解，我们甚至可以做出语言本身不支持的设计。所以，做设计真正的地基，并不是程序设计语言，而是运行时。**

理解运行时，可以将“**程序如何运行**”作为主线，将相关的知识贯穿起来。我们从了解可执行文件的结构开始，然后了解程序加载机制，知道加载器的运作和内存布局。然后，程序开始运行，我们要知道程序的运行机制，了解字节码，形成一个整体认识。最后，还可以根据需要展开各种细节，做深入的了解。

有一些语言的运行时还提供了一些语言层面的编程接口，程序员们可以与运行时进行交互，甚至拥有超过语言本身的能力。这些接口有的是以程序库的方式提供，有的则是以规范的方式提供。如果你是一个程序库的开发者，这些接口可以帮助你写出更优雅的程序。

关于程序设计语言的介绍，我用了三讲分别从模型、接口和实现等不同的角度给你做了介绍。目的无非就是一个，想做好设计，不仅仅要有设计的理念，更要有实际落地的方式。下一讲，我们来讲一个你可以在项目中自己设计的语言：DSL。

如果今天的内容你只能记住一件事，那请记住：**把运行时当作自己设计的地基，不受限于语言。**



思考题

最后，我想请你分享一下，你知道哪些程序库的哪些特性是利用运行时交互的接口实现的？欢迎在留言区分享你的想法。

感谢阅读，如果你觉得这一讲的内容对你有帮助的话，也欢迎把它分享给你的朋友。

更多福利推荐

充值限时膨胀
充 ¥500 得 ¥580
下单即赠精选爆款商品

戳此充值

充 ¥500 得 ¥580

充 ¥1000 得 ¥1200

充 ¥2000 得 ¥2500



精选留言 (3)

写留言



Kăfkă²⁰²⁰

2020-06-15

我们自己的分布式日志收集系统就使用了运行时字节码生成技术，这样既对业务应用没有任何影响（性能上微乎其微），又能根据抓取的日志分析服务调用链、异常等有用的信息

作者回复：你做的事情与APM有相似的地方。



6



Jxin

2020-06-15

1.在web flux出来之前，java的web开发大多都是基于servlet的。可以认为servlet容器是java web项目的运行时环境。servlet 3.0的"ServletContext"（通过编程方式配置servlet,filter, listener而不依赖web.xml）和"运行时插拔" 两大特性（两个特性前后关联），相当于以编程库的方式提供了运行时容器的服务增减的能力。这也是spring 3.X后spring web自动装配的实现基础。...

展开

作者回复：你这个理解的角度很有趣，非常好的补充！



1



阳仔

2020-06-15

运行时是程序语言设计中的地基

理解运行时可以深入理解程序语言是如何加载，执行的，这能让我们对语言添加一些原本不支持的功能

对于JAVA语言就是要了解jvm执行的机制，它包括加载字节码，解析，内存管理，线程调度等方面...

展开

作者回复：很好的总结！

1

