



下载APP



31 | 运行时（一）：从0到语言级的虚拟化

2020-08-24 宫文学

编译原理实战课

[进入课程 >](#)



讲述：宫文学

时长 20:27 大小 18.73M



你好，我是宫文学。今天，我会带你去考察现代语言设计中的运行时特性，并讨论一下与标准库有关的话题。

你可能要问了，咱们这门课是要讲编译原理啊，为什么要学运行时呢。其实，对于一门语言来说，除了需要提供编译器外，还必须提供运行时功能和标准库：一是，编译器生成的目标代码，需要运行时的帮助才能顺利运行；二是，我们写代码的时候，有一些标准的功能，像是读写文件的功能，自己实现起来太麻烦，或者根本不可能用这门语言本身来实现，这时就需要标准库的支持。



其实，我们也经常会接触到运行时和库，但可能只是停留在使用层面上，并不太会关注它们的原理等。如果真要细究起来、真要对编译原理有更透彻的理解的话，你可能就会有下面这些问题了：

到底什么是运行时？任何语言都有运行时吗？运行时和编译器是什么关系？

什么是标准库？标准库和运行时库又是什么关系？库一般都包含什么功能？

今天，我们就来探讨一下这些与运行时和标准库有关的话题。这样，你能更加充分地理解设计一门语言要完成哪些工作，以及这些工作跟编译技术又有什么关系，也就能对编译原理有更深一层的理解。

首先，我们来了解一下运行时，以及它和编译技术的关系。

什么是运行时 (Runtime) ？

我们在 [第 5 讲](#) 说过，每种语言都有一个特定的执行模型 (Execution Model) 。而这个执行模型就需要运行时系统 (Runtime System) 的支持。我们把这种可以支撑程序运行的运行时系统，简称为运行时。

那运行时都包含什么功能呢？通常，我们最关心的是三方面的功能：程序运行机制、内存管理机制和并发机制。接下来，我就分别以 Java、Python 以及 C、C++、Go 语言的运行时机制为例，做一下运行时的分析，因为它们的使用者比较多，并且体现了一些有代表性的运行时特征。

Java 的运行时

我们先看看 Java 语言的运行时系统，也就是 JVM。

其实，JVM 不仅为 Java 提供了运行时环境，还与其他所有基于 JVM 的语言提供了支撑，包括 Scala、Clojure、Groovy 等。我们可以通过 [JVM 的规范](#) 来学习一下它的主要特点。

第一，**JVM 规定了一套程序的运行机制**。JVM 支持基于字节码的解释执行机制，还包括了即时编译成机器码并执行的机制。

针对基于字节码的解释执行机制，JVM 规范定义下面这些内容：

定义了一套字节码来运行程序。这些字节码能支持一些基本的运算。超出这些基本运算逻辑的，就要自己去实现。比如，idiv 指令用于做整数的除法，当除数为零的时候，虚

拟缺省的操作是抛出异常。如果你自己的语言是专注于数学计算的，想让整数除以零的结果为无穷大，那么你需要自己去实现这个逻辑。

规定了一套类型系统，包括基础数据类型、数组、引用类型等。所以说，任何运行在 JVM 上的语言，不管它设计的类型系统是什么样子，编译以后都会变成字节码规定的基础类型。

定义了 class 文件的结构。class 文件规定了把某个类的符号表放在哪里、把字节码放在哪里，所以写编译器的时候要遵守这个规范才能生成正确的 class 文件。JVM 在运行时会加载 class 文件并执行。

提供了一个基于栈的解释器，来解释执行字节码。编译器要根据这个执行模型来生成正确的字节码。

除了解释执行字节码的机制，JVM 还支持即时编译成机器码并执行的机制。它可以调度多个编译器，生成不同优化级别的机器码，这就是分层编译机制。在需要的时候，还可以做逆优化，在不同版本的机器码以及解释执行模式之间做切换。

最后，Java 程序之间的互相调用，需要遵循一定的调用约定或二进制标准，包括如何传参数等等。这也是运行机制的一部分。

总体来说，JVM 代表了一种比较复杂的运行机制，既可以解释执行，又可以编译成机器码执行。V8 的运行时机制也跟 JVM 也很类似。

第二，**JVM 对内存做了统一的管理**。它把内存划分为程序计数器、虚拟机栈、堆、方法区、运行时常量池和本地方法栈等不同的区域。

对于栈来说，它的栈帧既可以服务于解释执行，又可以用于执行机器码，并且还可以在两种模式之间转换。在解释执行的时候，栈帧里会有一个操作数栈，服务于解释器。我们提到过 OSR，也就是在运行一个方法的时候，把这个方法做即时编译，并且把它的栈帧从解释执行的状态切换成运行机器码的状态。而如果遇到逆优化的场景，栈帧又会从运行机器码的状态，切换成解释执行的状态。

对于堆来说，Java 提供了垃圾收集器帮助进行内存的自动管理。减少整体的停顿时间，是垃圾收集器设计的重要目标。

第三，**JVM 封装了操作系统的线程模型，为应用程序提供了并发处理的机制。**我会在讲并发机制的时候再展开。

以上就是 JVM 为运行在其上的任何程序提供的支撑了。在提供这些支撑的同时，运行时系统也给程序运行带来了一些限制。

第一，JVM 实际上提供了一个基础的对象模型，JVM 上的各种语言必须遵守。所以，虽然 Clojure 是一个函数式编程语言，但它在底层却不得不使用 JVM 规定的对象模型。

第二，基于 JVM 的语言程序要去调用 C 语言等生成的机器码的库，会比较难。不过，对于同样基于 JVM 的语言，则很容易实现相互之间的调用，因为它们底层都是类和字节码。

第三，在内存管理上，程序不能直接访问内存地址，也不能手动释放内存。

第四，在并发方面，JVM 只提供了线程机制。如果你要使用其他并发模型，比如我们会在 34 讲中讲到的协程模型和 35 讲中的 Actor 模型，需要语言的实现者绕着弯去做，增加一些自己的运行时机制（我会在第 34 讲来具体介绍）。

好了，以上就是我要通过 JVM 的例子带你学习的 Java 的运行时，以及其编译器的影响了。我们再来看看 Python 的运行时。

Python 的运行时

在解析 Python 语言的时候，已经讲了 Python 的字节码和解释器，以及 Python 对象模型和程序调用的机制。这里，我再从程序运行机制、内存管理机制、并发机制这三个方面，给你梳理下。

第一，Python 也提供了一套字节码，以及运行该字节码的解释器。这套字节码，也是跟 Python 的类型体系互相配合的。字节码中操作的那些标识符，都是 Python 的对象引用。

第二，在内存管理方面，Python 也提供了自己的机制，包括对栈和堆的管理。

首先，我们看看栈。Python 运行程序的时候，有些时候是运行机器码，比如内置函数，而有些时候是解释执行字节码。

运行机器码的时候，栈帧跟 C 语言程序的栈帧是没啥区别的。而在解释执行字节码的时候，栈帧里会包含一个操作数栈，这点跟 JVM 的栈机是一样的。如果你再进一步，去看看操作数栈的实现，会发现解释器本身主要就是一个 C 语言实现的函数，而操作数栈就是这个函数里用到的本地变量。因此操作数栈也会像其他本地变量一样，被优化成尽量使用物理寄存器，从而提高运行效率。这个知识点你要掌握，也就是说，**栈帧中的操作数栈，其实是有可能基于物理寄存器的。**

然后，Python 还提供了对堆的管理机制。程序从堆里申请内存的时候，不是直接从操作系统申请，而是通过 Python 提供的一个 Arena 机制，使得内存的申请和释放更加高效、灵活。Python 还提供了基于引用的垃圾收集机制（我会在下一讲为你总结垃圾收集机制）。

第三，是并发机制。Python 把操作系统的线程进行了封装，让 Python 程序能支持基于线程的并发。同时，它也实现了协程机制（我会在 34 讲详细展开）。

好了，我们再看看第三类语言，也就是 C、C++、Go 这样的直接编译成二进制文件执行的语言的运行时。

C、C++、Go 的运行时

一个有意思的问题是，C 语言有没有运行时呢？我们对 C 语言的印象，是一旦编译完成以后，就是一段完全可以自主运行的二进制代码了，你也可以看到输出的完整的汇编代码。除此之外没有其他，C 语言似乎不需要运行时的支持。

所以，**C 语言最主要的运行时，实际上就是操作系统。**C 语言和现代的各种操作系统可以说是伴生关系，就像 Java 和 JVM 是伴生关系一样。所以，如果我们要深入使用 C 语言，某种意义上就是要深入了解操作系统的运行机制。

在程序执行机制方面，C 语言编译完毕的程序是完全按照操作系统的运行机制来执行的。

在内存管理方面，C 语言使用了操作系统提供的线程栈，操作系统能够自动帮助程序管理内存。程序也可以从堆里申请内存，但必须自己负责释放，没有自动内存管理机制。

在并发机制方面，当然也是直接用操作系统提供的线程机制。因为操作系统没有提供协程和 Actor 机制，所以 C 语言也没有提供这种并发机制。

不过有一个程序 crt0.o，有时被称作是 C 语言的运行时。它是一段汇编代码（crt0.s），由链接器自动插入到程序里面，主要功能是在调用 main 函数之前做一些初始化工作，比如设置 main 函数的参数（argc 和 argv）、环境变量的地址、调用 main 函数、设置一些中断向量用于处理程序异常等。所以，这个所谓的运行时所做的工作也特别简单。

不同系统的 crt0.s 会不太一样，因为 CPU 架构和 ABI 是不同的。下面是一个 crt0.s 的示例代码：

[复制代码](#)

```
1  .text
2  .globl _start
3  _start: # _start是链接器需要用到的入口
4      xor %ebp, %ebp          # 让ebp置为0，标记栈帧的底部
5      mov (%rsp), %edi        # 从栈里获得argc的值
6      lea 8(%rsp), %rsi       # 从栈里获得argv的地址
7      lea 16(%rsp,%rdi,8), %rdx # 从栈里获得envp的地址
8      xor %eax, %eax          # 按照ABI的要求把eax置为0，并与icc兼容
9      call main               # 调用main函数，%edi，%rsi，%rdx是传给main函数的三个
10
11     mov %eax, %edi          # 把main函数的返回值提供给_exit作为第一个参数
12     xor %eax, %eax          # 按照ABI的要求把eax置为0，并与icc兼容
13     call _exit              # 终止程序
```

可以说，C 语言的运行时是一个极端，提供了最少的功能。反过来呢，这也就是给了程序员最大的自由度。C++ 语言的跟 C 是类似的，我就不再展开了。总的来说，它们都没有 Java 和 Python 那种意义上的运行时。

不过，**Go 语言虽然也是编译成二进制的可执行文件，但它的运行时要复杂得多。**比如，它有垃圾收集器；再比如，Go 语言最显著的特点是提供了自己的并发机制，也就是 goroutine。对 goroutine 的运行管理，也是 go 的运行时的一部分。

无独有偶，在 Android 平台上，你可以把 Java 程序以 AOT 的方式编译成可执行文件。但这个可执行文件其实仍然包含了一个运行时，比如垃圾收集功能，所以与 C 语言编译形成的可执行文件，也是不一样的。

总结起来，运行时系统提供了程序的运行机制、内存管理机制、并行机制等功能。运行时和编译器的关系就是，编译器要跟这些运行时做配合，生成符合运行时要求的目标代码。

接下来，我们再看看语言的另一个重要组成部分，也就是标准库，并看看它跟编译器的关系。

库和标准库

我们知道，任何一门编程语言，要想很好地投入实际应用，必须有良好的库来支撑。这些库的作用就是封装了常用的、标准的功能，让开发者可以直接使用。

根据库的使用场景和与编译器的关系，这些库可以分为**标准库**、**运行时库**和**内置函数**三类。

第一，标准库，供用户的程序调用。我们在写一段 C 语言程序的时候，总要在源代码一开头的部分 include 几个库进来，比如 `stdio.h`、`stdlib.h` 等等。C++ 的 STL 库和标准库让程序员拥有比 C 语言里面更多的工具，比如各种标准的容器类。Java 刚面世的时候，就在 JDK 里打包了很多标准库。正是因为这些丰富又好用的库，使得 Java 能够被迅速接受。当然了，这些库也成了 JDK 标准的组成部分。而 Python 语言声称是“自带电池”的，也就是说有很多库的支持，可以迅速上手做很多事情。

第二类，运行时库，它们不是由用户直接调用的，而是运行时的组成部分。比如，Python 实现整数运算的功能很强大，支持任意长度整数的加减乘除。这些功能是由一些库函数实现的，并由 Python 的解释器来调用，实现 Python 程序中的加减乘除操作。

第三类，是一些叫做 Built-in 或者 Intrinsics 的内置函数，它们是用来辅助生成机器码的。它们往往由汇编代码实现，也有的是用编译器的 LIR 实现的，在编译的时候直接内联进去。这些函数有时开发者也可以调用，比如在 C 语言中，可以像调用普通函数一样，调用 CPU 厂家提供的与 SIMD 指令有关的 Intrinsics。但这些函数会直接生成汇编码，不像 C 语言编写的程序那样需要经过优化和代码生成的过程。

好了，我们了解了库的三种分类，也就是标准库、运行时库和内置函数。不过我要提醒你的是，这些分类有时候是模糊的，比如有的语言（比如微软的 C 和 C++ 语言）谈到运行时库的时候，实际上就包括了标准库。

接下来，我们主要看看与标准库相关的几个问题。

标准库的特殊性

与普通程序相比，标准库主要有以下三个方面的不同。

第一，有的库可以用本语言来实现，而有的库必须要用其他语言来实现，因为用本语言实现有困难。这就要求库的编写者要具备更高的技能，能够掌握更加底层的语言。

比如，Java 有少量库（比如网络通讯模块）就需要用 C 语言来编写，而 Python、PHP、Node.js 等语言的大量库都是用 C 语言编写的。甚至，标准库中的某些底层功能会采用汇编语言来写。

第二，标准库的接口不可以经常变化，甚至是要保持一直不变。因此，标准库的设计一定要慎重，这就要求设计者有更高的规划和设计能力。因为几乎每个程序都会用到标准库的功能，库的接口如果变化的话，就会影响到所有已经写好的程序。

第三，标准库往往集中体现了一门语言的核心特点。同样的功能，面向对象编程语言、函数式编程语言、基于 Actor 的语言，会采用各自的方式来实现。库的编写者要写出教科书级的代码，充分发挥这门语言的优势。这样的话，编程人员使用这些标准库的过程，实际上就是潜移默化地学习这门语言的编程思想的过程。

好了，看来编写一个好的标准库确实是有挑战的事情。但是标准库一般需要包含哪些内容呢？

标准库需要包含什么功能？

第一，包含 IO 功能，包括文件 IO、网络 IO 等。

还记得吧，我们学习每一门新语言的时候，都会在终端上打印出一个“Hello World!”，这似乎已经成了一种具有仪式感的行为。可是你注意到没有，你在打印输出到终端的时候，通常就是调用了一个标准的 IO 库。因为终端本身就相当于一个文件，这实际上是用了文件 IO 功能。

除了文件 IO，网络 IO 也必不可少，这样的话手机上的 App 程序才能够跟服务端的程序通讯。

第二，支持内置的数据类型。

首先是针对整型、浮点型等基础数据类型做运算的功能。比如有的数学库的数学计算功能支持任意长度的整数的运算，并支持准确的小数运算（计算机内置的浮点数计算功能是不精确的）。此外数据类型转换、对字符串操作等，也是必不可少的。

像 Java、Python 这样的语言，提供了一些标准的内置类型，比如 String 等。像 Scala 这种纯面向对象语言，连整型、浮点型等基础数据类型，也是通过标准库来提供的。

第三，支持各种容器型的数据结构。

有的语言（比如 Go），会在语法层面提供 map 等容器型的数据结构，并通过运行时库做支持；还有些语言（比如 Java、C++），是在标准库里提供这些数据结构。

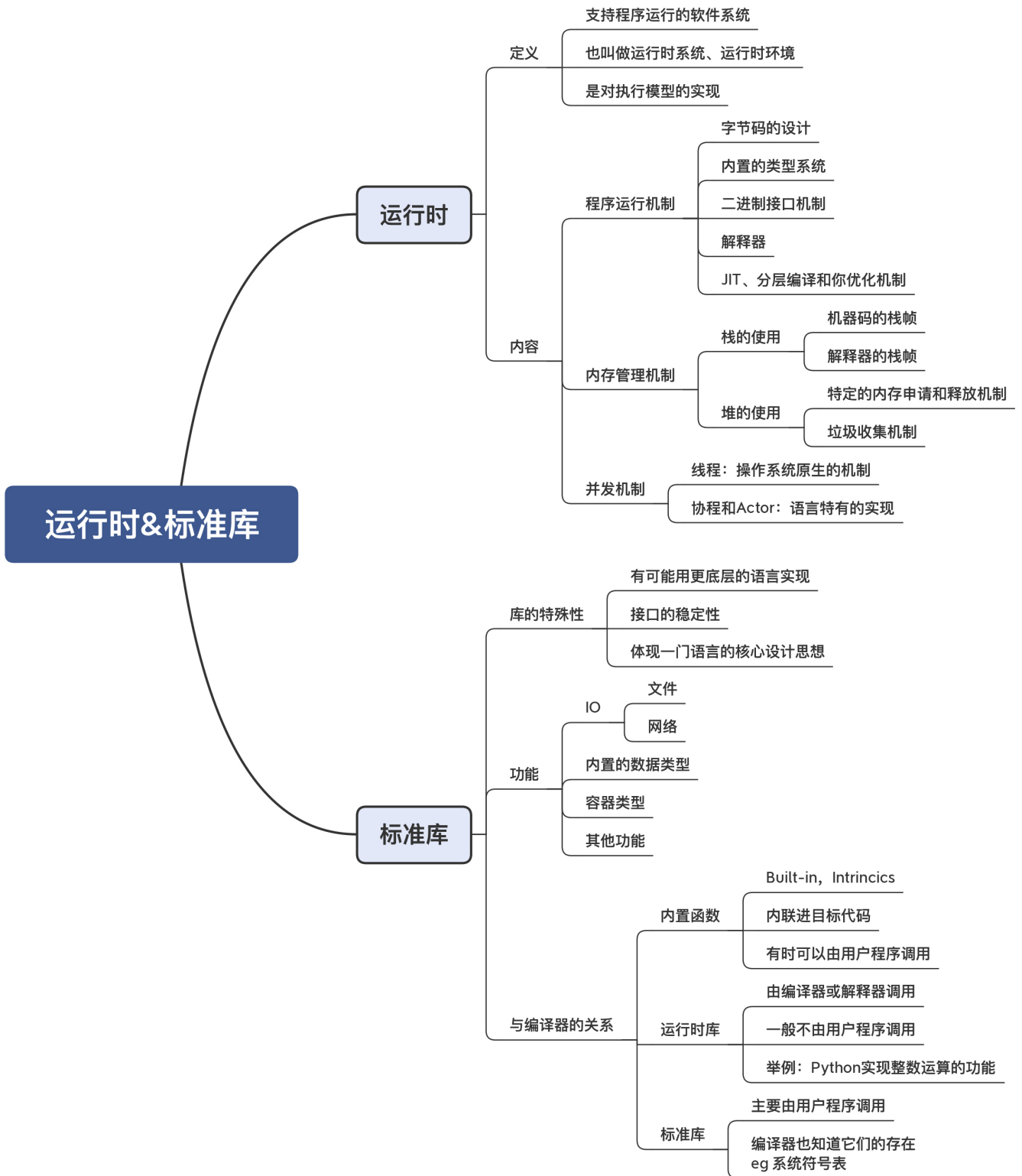
此外，标准库还要包含一些其他功能，比如对日期、图形界面等各种不同的功能支持。

课程小结

今天，我们一起学习了一门语言除编译器之外的一些重要组成部分，包括运行时和各种库。编译器拥有运行时和库的知识，并根据这些知识作出正确的编译。当你设计一门语言的时候，应该首先要把它运行机制设计清楚，然后才能设计出正确的语法、语义，并实现出相应的编译器。

所以，我们这一讲的目标，就是帮你从一个更高的维度来理解编译技术的使用环境，从而更加全面地理解和使用编译技术。

我把今天的知识点也整理成了思维导图，供你参考：



一课一思

挑你熟悉的一门语言，分享一下它的运行时和标准库的设计特征，以及对编译器的影响。

欢迎你在留言区表达自己的见解，也非常欢迎你把今天的内容分享给更多的朋友。感谢阅读，我们下一讲再见。

[提建议](#)

更多课程推荐

MySQL 实战 45 讲

从原理到实战，丁奇带你搞懂 MySQL

林晓斌

网名丁奇
前阿里资深技术专家



立省 ¥30



今日秒杀 **¥99**，5.8W 人在学

© 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 30 | 后端总结：充分发挥硬件的能力

精选留言

写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。