### 如何理解RunTime

实际上编程语境中的 runtime 至少有三个含义，而目前的回答都只侧重讲了其中的某一个，所以看起来很令人困惑。这几个含义分别可以这样概括：

1. 指「[程序运行的时候](https://link.zhihu.com/?target=https://en.wikipedia.org/wiki/Runtime_(program_lifecycle_phase)" \t "_blank)」，**即程序生命周期中的一个阶段**。例句：「*Rust 比 C 更容易将错误发现在编译时而非运行时。*」
2. 指「[运行时库](https://link.zhihu.com/?target=https://en.wikipedia.org/wiki/Runtime_library" \t "_blank)」，**即 glibc 这类原生语言的标准库**。例句：「*C 程序的 malloc 函数实现需要由运行时提供。*」
3. 指「[运行时系统](https://link.zhihu.com/?target=https://en.wikipedia.org/wiki/Runtime_system" \t "_blank)」，**即某门语言的宿主环境**。例句：「*Node.js 是一个 JavaScript 的运行时。*」

下面简单介绍一下个人的理解。

## 含义一：程序生命周期中的阶段

一个程序从写好代码字符串（起点）到跑完退出（终点），有一整套标准化的生命周期（流程），可以被拆分为多个阶段。这其中编译阶段是 compile time，链接阶段是 link time，那运行起来的阶段自然就是 run time 了。如果在前面的阶段预先做了通常在后面才方便做的事，我们就管这个叫 ahead of time。

注意所谓 ahead of time 其实只是英语口语中的常见词汇，并不是 AJAX 这种专有的技术概念。比如美军参谋长在通共电话里说的这句：  
If we're going to attack, I'm going to call you **ahead of time**. It's not going to be a surprise.[[1]](" \l "ref_1)

个人猜测 runtime 这个词衍生出的定义应该就源于 runtime，泛指那些「**供代码运行所需的最基础的软件**」。下面的两个定义其实也都没有超出这个范畴。

## 含义二：运行时库（runtime library）

怎样理解 runtime library 呢？要知道 C、C++ 和 Rust 这类「系统级语言」相比于 JavaScript 这类「应用级语言」最大的特点之一，就在于它们可以胜任嵌入式裸机、操作系统驱动等贴近硬件性质的开发——**而所谓 runtime library，大致就是这时候你没法用的东西**。

回想一下，我们在 C 语言里是怎么写 hello world 的呢？

#include <stdio.h> // 1int main(void)

{ // 2

printf("Hello World!\n");

// 3}

这里面除了最后一个括号，每行都和运行时库有很大关系：

1. stdio.h 里的符号是 C 标准库提供的 API，我们可以 include 进来按需使用（但注意运行时库并不只是标准库）。
2. main 函数是程序入口，但难道可执行文件的机器码一打开就是它吗？这需要有一个复杂的启动流程，是个从 \_start 开始的兔子洞。
3. printf 是运行时库提供的符号。可这里难道不是直接调操作系统的 API 吗？实际上不管是 OS 的系统调用还是汇编指令，它们都不方便让你直接把字符串画到终端上，这些过程也要靠标准库帮你封装一下。

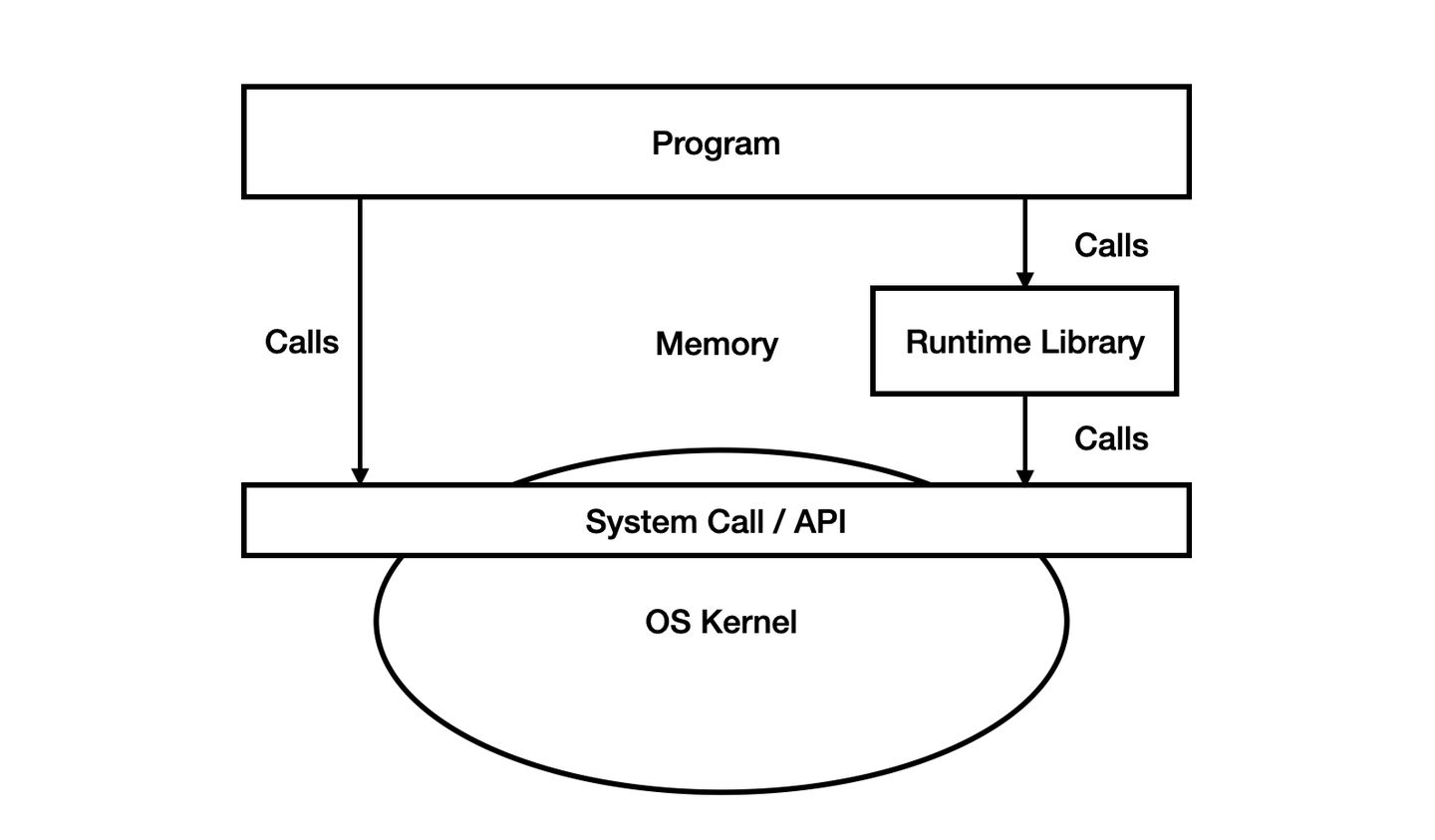
**在缺少操作系统和标准库的裸机环境下（例如 Rust 的 [no\_std](https://link.zhihu.com/?target=https://docs.rust-embedded.org/book/intro/no-std.html" \t "_blank)），上面的代码是跑不起来的**。而这里的 stdio 只是标准库的冰山一角，再举几个非常常见的例子：

* 负责数学运算的 math.h：很多精简指令集或嵌入式的低端 CPU 未必会提供做 sin 和 cos 这类[三角函数](https://www.zhihu.com/search?q=%E4%B8%89%E8%A7%92%E5%87%BD%E6%95%B0&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":2133648600}" \t "_blank)运算的指令，这时它们需要软件实现。
* 负责字符串的 string.h：你觉得硬件和操作系统会内置「比较字符串长度」这种功能吗？当然也是靠软件实现啦。
* 负责内存分配的 stdlib.h：直接通过 [mmap](https://link.zhihu.com/?target=https://en.wikipedia.org/wiki/Mmap" \t "_blank) 这类 OS 系统调用来分配内存是过于底层的，一般也需要有人帮你封装。分配内存的 malloc 虽然只是一个接受单个参数的函数，它的实现可远没有表面上的 API 那么简单。

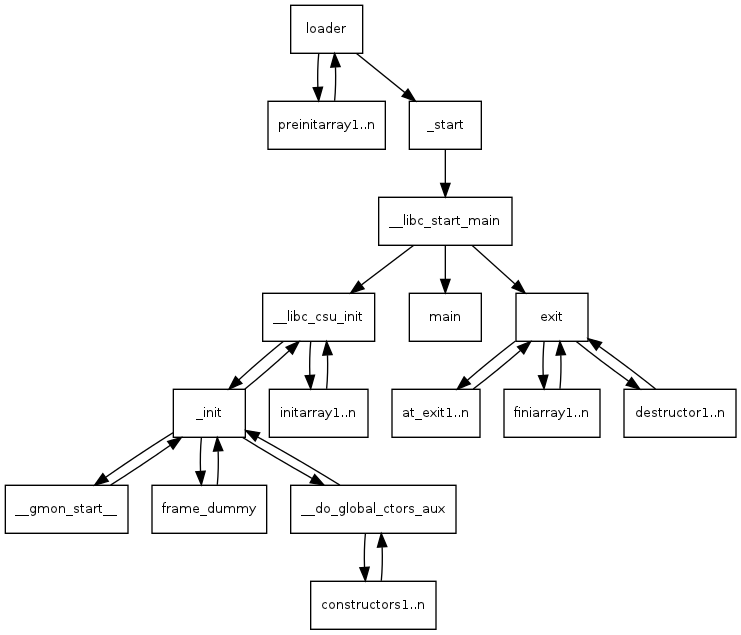
换句话说，虽然 C 的 if、for 和函数等语言特性都可以很朴素且优雅地映射（lowering）到汇编，但必然会有些没法直接映射到系统调用和汇编指令的常用功能，比如上面介绍的那几项。对于这些脏活累活，它们就需要由运行时库（例如 Linux 上的 glibc 和 Windows 上的 CRT）来实现。

如果你熟悉 JavaScript 但还不熟悉 C，我还有篇讲「[C 手动内存管理基础入门](https://zhuanlan.zhihu.com/p/356214452)」的教程应该适合你。

我们可以把「应用程序、运行时库和 OS」三者间的关系大致按这样来理解：



注意运行时库并不只是标准库，你就算不显式 include 任何标准库，也有一些额外的代码会被编译器插入到最后的可执行文件里。比如上面提到的 main 函数，它在真正执行前就需要大量来自运行时库的辅助，一图胜千言（具体细节推荐参考 [Linux x86 Program Start Up](https://link.zhihu.com/?target=http://dbp-consulting.com/tutorials/debugging/linuxProgramStartup.html" \t "_blank)）：



除了加载和退出这些程序必备的地方以外，运行时库还可以起到类似前端社区 polyfill 的作用，在程序执行过程中被隐式而「按需」地调用。例如 gcc 的 [libgcc](https://link.zhihu.com/?target=https://gcc.gnu.org/onlinedocs/gccint/Libgcc.html" \t "_blank) 和 clang 的 [compiler-rt](https://link.zhihu.com/?target=https://compiler-rt.llvm.org/" \t "_blank)（后者还被移植成了 Rust 的 [compiler-builtins](https://link.zhihu.com/?target=https://github.com/rust-lang/compiler-builtins" \t "_blank) ），这些库都是特定于编译器的，我们一般比较少听到，但其实也很好理解。

举个例子，我在移植 QuickJS 引擎到索尼 PSP 的时候，发现虽然把 libc 的静态库链接进来了，但链接时始终找不到 \_\_truncdfsf2 这个符号。这非常让人困惑，因为那个报错位置的源码简单到了这种程度：

// 这是 QuickJS 相应位置的源码static double js\_math\_fround(double a){

return (float)a;}

我把这个函数在 .o 目标文件里反汇编以后的结果读来读去，也完全没有看到 \_\_truncdfsf2 这个东西。但其实是这样的：double 到 float 的转换并不能由 PSP 的 CPU 指令直接完成（PSP 刻意阉割了对双精度浮点数的硬件支持），因此编译 PSP 应用时需要通过软件实现来兼容，这个软浮点算法就叫 \_\_truncdfsf2，它本来应该由编译器在链接出可执行文件时自动插入，但我用的 Rust 工具链恰好没有实现它（[Issue #327 · compiler-builtins](https://link.zhihu.com/?target=https://github.com/rust-lang/compiler-builtins/issues/327" \t "_blank)），于是就有了这个报错。最后我把找来的一个软浮点函数的代码贴进来，就可以正确完成链接了。这其实也是个人第一次意识到原来所谓「运行时库」并不仅仅是 stdio.h 里提供的那些符号——哪有什么 include 进来一把梭的岁月静好，还要有编译器和运行时替你默默负重前行。

理解问题原因后再去看上面的 C 代码，可以感受到这里运行时库所起到的作用，跟 JavaScript 中用于支持新语法的 babel [转译](https://www.zhihu.com/search?q=%E8%BD%AC%E8%AF%91&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":2133648600}" \t "_blank)产物颇有些相似之处。这还是挺有趣的。

总之，由于系统级语言被设计成既可以用来写操作系统上的原生应用，也可以用来写 bare metal 的裸机程序，因此这类语言需要的运行时（runtime）被设计成了可以按需使用的库（library），于是我们就自然地得到了 runtime library 这个概念。

## 含义三：运行时系统（runtime system）

上面介绍的运行时库，主要针对的是 C、C++ 和 Rust 这些「系统级语言」。只要将这个概念继续推广到其他高级语言，这时候的「运行时」指的就是 runtime system 了——**如果讨论某门高级语言的运行时，我们通常是在讨论一个更重、更大而全的运行时库**。

比如 Java 的运行时是 JRE，C# 的运行时是 CLR。这两者都相当于一个需要在 OS 上单独安装的软件，借助它们来解释执行相应语言的程序（编译出的字节码）。而对 JavaScript 来说，一般「JS 引擎」是个不带 IO 支持的[虚拟机](https://www.zhihu.com/search?q=%E8%99%9A%E6%8B%9F%E6%9C%BA&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":2133648600}" \t "_blank)，需要浏览器和 Node 这样的「JS 运行时」才能让它控制文件、网络、图形等硬件资源而真正实用。这些都是很经典的模型了。

典型的高级语言「运行时系统」里大概需要这些基础组件：

* 一个解释执行字节码的虚拟机，多半得带个垃圾回收器。
* 如果语言是源码解释执行，那么需要一个编译器前端做词法分析和语法分析。
* 如果运行时支持 JIT 优化，那么还得藏着个编译器后端（动态生成[机器码](https://www.zhihu.com/search?q=%E6%9C%BA%E5%99%A8%E7%A0%81&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":2133648600}" \t "_blank)）。
* IO 相关能力，比如 Node.js 的 fs.readFile 之类。

可以看到相比上面 C 语言的「运行时」，这已经是个复杂的基础软件系统了。

稍微再展开一点，**注意上面的「运行时」里是不包含应用程序业务逻辑的**。那么拿 JavaScript 举例来说，如果我们把业务逻辑先编译成字节码，再把它和运行时一起编译成一个可执行文件，那不就相当于「**直接把 JavaScript 编译成机器码**」了吗？QuickJS 就可以这么做，但其实这时候业务逻辑解释执行的天性不会变——难道真有黑科技能把弱类型的脚本直接靠静态分析编译达到系统级语言的水平？这更多地只是概念定义上的话术而已。

因此，理论上任意的弱类型动态语言都可以基于这种形式来 AOT 编译成「原生机器码」，你看 Dart、Swift 和 Java 都可以直接编译成可执行文件，区别只是这个运行时的轻重量级不同——当然实际情况肯定没有这么理想化，譬如哪怕编译成了 ARM 机器码，Flutter 里的 Dart 运行时也必然需要比 C 做更多的类型检查和 stop the world 的 GC，这都是有成本的。但对于[应用层](https://www.zhihu.com/search?q=%E5%BA%94%E7%94%A8%E5%B1%82&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":2133648600}" \t "_blank)开发来说，能做到这样已经够好了。

所以我们甚至可以激进地认为对于 OS 上的应用程序，各种编程语言都是或多或少地需要运行时的，大家只有运行时轻重的区别———「**其 实 都 一 样**」。

综上所述，runtime 在技术讨论中有多个含义，我们经常用它作为 runtime library 和 runtime system 的简称，因此可能造成一些误解。