从C++到Rust之路

原创 Ajonbin AJonbin的杂货铺 2023年12月24日 21:47 上海

今天开始开篇谈谈Rust。

作为C++ Programmer，一道绕不过的坎不是不断升级的C++版本，而是Rust。

有人说C++的尽头是Rust，作为C++的忠实用户，一路从C到Classic C++ 再到Morden C++，蹚过JAVA的河，翻过Python的山，爬出Javascript的坑，现在面对Rust，除了学习只有学习了。

先来看看大家是怎么吹嘘Rust的。

安全

安全可以分为两个部分：行为安全和数据安全。

先来谈谈行为安全

首先先来谈谈为什么C++被认为是不安全的。这就是著名的"undefined behavior"。
在C++17标准中，是这么定义Undefined behavior:

behavior for which this International Standard imposes no requirements.
本国际标准对于此行为没有强制性要求

也就是说，随便你怎么搞，我都行。
这一点也就常会被利用成为病毒和攻击软件。

一般来说，C++中“undefined behavior” 包括但不限于：

- 解析空指针
- 写操作时，数组越界
- 访问未初始化的指针
- 修改常量
- 使用已经被释放的对象

Rust通过语法和编译器来避免上述问题的发生。
只要通过Rust编译器，你的代码就不会产生未定义行为。

安全还有一层意思，是数据安全。

在多线程代码中，你需要精心设计数据访问锁和访问顺序，来确保的数据在不同线程之间读写操作时保持一致。读写数据时，C++多数情况下都不是线程安全的。

但Rust设计的原则就保证了只要通过编译，函数就是线程安全的。Rust会在编译器发现可能存在的数据竞争问题，而不是在运行起来之后给你个惊喜。

快速

Rust和C++一样，都是以zero-overhead令开销未设计原则。

也就是说你不必为你不用的东西付出代价。

这就保证了Rust和C++可以在你的控制下高效执行。当然你的代码必须是高效的，低效的代码同样会导致Rust和C++运行缓慢。 这里强调的是你行，Rust就行，Rust不会夹带私货。

garbage collection就是一种额外的开销。它在让你不必关心释放对象内存的同时，也增加了系统的额外负担。 对于Rust而言，这些都需要你自己处理，得到的好处就是可控，高效。

包管理

Rust可以说是自带了包管理工具cargo，它会管理你的依赖，你只需要制定你代码直接依赖的库，库的依赖cargo会自动帮你下载编译。这就很方便了。你添加你需要的依赖，指定你需要的版本，其他cargo都会帮你完成。

而对于C++来说，你可能就需要写Makefile，这很麻烦，基本是完全手动的，所有依赖关系你必须写明白。所以就出现了许多build C++工程的工具， CMake或是bazel之类。
这些工具本身也比较复杂，需要写不同的规则。

良好的并行Parallelism支持

在设计C++的年代，基本没有并行计算。早期很多是嵌入式程序是单进程加一个大loop，随着程序越来越复杂，再加入多线程进行并发计算。
现在多核基本就是标配了，对于C++要写出数据安全的代码就有了更高的要求。这就要求程序员必须设计好你的并发程序，确保数据在多线程之前保持同步。
当然，现在C++也有了并发库，来帮助我们更好的写出并发程序，但是从本质上说C++对并发不是那么友好，有许多工作要做。
作为一门新的语言Rust在设计之初就考虑了对并发的支持。

再看看反C++程序员的语法

刚开始写Rust的时候，确实很不习惯这个语法，甚至觉得是反人类的。
你必须很明确的知道你用的borrow还是move，写起来没有C++这样自由顺手。
好在Rust编译器会尽量告诉你错误是什么，解决编译错误还是相对容易的。
Rust编译器的错误提示要比C++好很多，C++的编译错误往往让人摸不到头脑，让人无从改起。

最后，不做比较，只说Rust是一门非常靠谱的语言。

毕竟C++是一门历史悠久，十分优秀的语言，但也正式悠久的历史，导致它有很多包袱，要考虑很多兼容性。

结论

拥抱Rust吧