计算缓存、优化算法和加速 Python 执行 第二部分

引入 Nim 语言

我们可以将计算密集型的函数移动到 C 扩展模块，来弥补 Python 这方面作为解释型脚本语言的不足，但是用 C 编写 Python 扩展模块并不容易，大多数人都选择使用Python，就是因为它不是 C。虽然为了性能提升引入一种新技术（语言）的代价其实有点大，但是如果这个语言和 Python 很接近，但是性能几乎和 C 语言一样，并且可以很方便的为 Python 提供扩展包，这样的话，这个引入还是值得一试的。

Nim 语言似乎还是比较小众，在它的官网网站，https://nim-lang.org/ 是这样介绍自己的：

Nim是一种系统和应用程序编程语言，支持 静态类型和编译，它通过优雅的方式提供无与伦比的性能。

Nim 具有以下特性：

- 高性能，自动垃圾回收
- 可以编译成 C、C++ 或者 JavaScript 语言
- 生成无依赖的二进制代码
- 可以运行在 Windows、macOS 和 Linux 等

是不是听上去很神奇，可以编译成二进制代码不算稀奇，可以编译成 JavaScript 还是很不错的，这样就可以全栈通吃了。（这一点很像 Kotlin 语言）我猜测 Nim 的工作原理可能也是将自己编译成 C 语言，然后再将 C 语言编译成二进制代码，同样，也可以将自己转换为 C++ 或者 JavaScript 的等价语言，当然通过 lex 和 yacc 的分析加上应该非常复杂的模板和优化技术。

Nim 语言的安装

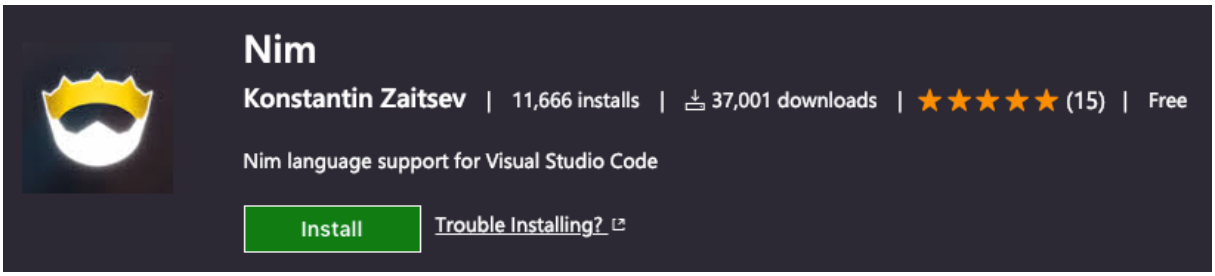
Nim 的安装看上去有点繁琐，可以参考 Nim 的官方文档：https://nim-lang.org/install_unix.html

在 macOS 上，最容易的安装方法是 `brew install nim` 。

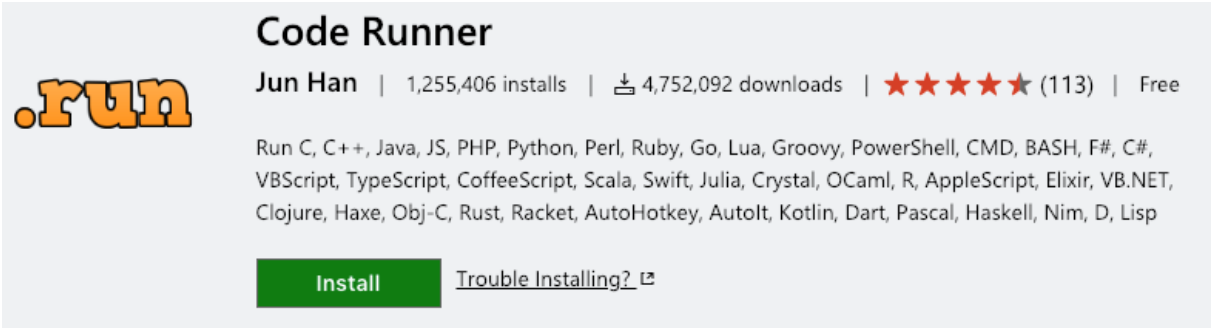
然后推荐几乎万能的 Visual Studio Code ，免费好用。可以到这里下载：https://code.visualstudio.com/。微软这几年还是做出了很多非常优秀的产品，且有开放的心态，才会有这样一个功能超强、支持很多种开发语言，并且还是开源免费的优秀开发工具。

打开 Visual Studio Code 后，安装 Nim 扩展，以及 Code Runner 扩展。

Nim 扩展使得 VS Code 具备支持 Nim 语法高亮。



而 Code Runner 扩展则功能强大，支持在 VS Code 中运行几乎所有主流语言。



按照 Nim 官方介绍，你可以运行一段最简单的 Hello World 程序来测试一下。输入 `echo "Hello World!"`，保存为 `helloworld.nim`，然后点击运行按钮，就会看到 `Hello World!`，如果没有出来正确结果的话，可以参考 Nim 官方文档。

我们可以运行一个复杂的，其实也就是刚才那个斐波那契数列的 Nim 程序版本：

```
import math, strformat, times

proc fib(n: int): int =
    if n <= 2:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```

```
when isMainModule:
    let x = 47
    let start = epochTime()
    let res = fib(x)
    let elapsed = (epochtime() - start).round(2)
    stderr.writeLine(&"Nim Computed fib({x})={res} in {elapsed} seconds")
```

用 Nim 制作 Python 扩展包

好，重点来了，我们通过 Nim 的扩展程序安装一个 nimpy 的包，就可以将 nim 编译后的程序作为 C 的二进制程序给 Python 程序使用了。

安装 nimpy 包：`nimble install nimpy`

下面 Nim 程序实现了算法：

```
import nimpy

proc fib(n: int): int {.exportpy.} =
    if n <= 2:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```

假设这个程序存盘文件名为 `fib_nimpy.nim`,

然后我们执行 `nim c -d:release --app:lib --gc:regions --out:fib_nimpy.so fib_nimpy.nim` ,

稍等片刻，看到一堆提示信息和 `operation successful (29124 lines compiled; 1.862 sec total; 47.984MiB peakmem; Release Build) [SuccessX]` ，编译已经成功，我们会看到一个 `fib_nimpy.so` 文件。so 文件是 unix 的动态连接库，是二进制文件，作用相当于windows下的 .dll 文件。我没有测试，估计在 Windows 环境下应该可以生成 dll 文件。

然后将 so 文件复制到和 Python 程序同样的路径，import 刚才生成的 so 文件。

```
import time
from fib_nimpy import fib
```

```
if __name__ == "__main__":
    x = 47
    start = time.time()
    res = fib(x)
    elapsed = time.time() - start
    print("Py3+Nim Computed fib(%s)=%s in %0.8f seconds" % (x, res,
elapsed))
```

我们可以看到 `from fib_nimpy` 就是刚才的 so 文件，其中的 `fib` 是我们在前面 Nim 语言中具体定义的函数。

执行上述代码，也就是没有任何优化的直接用递归硬算，性能是 Python 的 1000 倍，因为最慢的计算代码已经是和 C 语言同样速度的代码了。

沿袭这个思路，我们可以将所有消耗 CPU 资源的函数，特别是涉及到复杂计算的函数，都可以用 Nim 实现函数功能，然后被 import 到主 Python 程序来获得性能的大大提升。从前面的例子可以看到 Nim 语言可比 C 要简单好学得多。不过 Nim 语言还比较小众，甚至注定一直比较小众，没问题，聪明勤奋的你，为了性能优化是值得去学习的，再说 Nim 语言的优势还远远不止这点。