公告

昵称： 青山牧云人
园龄： 5年1个月
粉丝： 48
关注： 1
+加关注

< 2020年9月 >

日	一	二	三	四	五	六
30	31	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	1	2	3
4	5	6	7	8	9	10

搜索

找找看

谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

python(32)
Git(8)
Linux(5)
pip(3)
Bootstrap(3)
设计模式(3)
threading(2)
多线程(2)
死锁(2)
线程安全(2)
更多

随笔档案

2020年7月(2)
2020年3月(4)
2019年9月(1)
2019年8月(2)
2019年4月(2)
2018年12月(5)
2018年11月(5)
2018年10月(3)
2018年9月(9)
2018年7月(6)
2018年6月(6)

聊聊Python中的GIL

对于广大写Python的人来说，**GIL(Global Interpreter Lock**, 全局解释器锁)肯定不陌生，但未必清楚GIL的历史和全貌是怎样的，今天我们就来梳理一下GIL。

1. 什么是GIL

GIL的全称是 Global Interpreter Lock，全局解释器锁。之所以叫这个名字，是因为**Python的执行依赖于解释器。**Python最初的设计理念在于，**为了解决多线程之间数据完整性和状态同步的问题，设计为在任意时刻只有一个线程在解释器中运行。**而当执行多线程程序时，由GIL来控制同一时刻只有一个线程能够运行。即Python中的多线程是表面多线程，也可以理解为fake多线程，不是真正的多线程。

可能有的同学会问，**同一时刻只有一个线程能够运行，那么是怎么执行多线程程序的呢?其实原理很简单：解释器的分时复用。**即多个线程的代码，轮流被解释器执行，只不过切换的很频繁很快，给人一种多线程"同时"在执行的错觉。聊的学术化一点，其实就是"**并发**"。

再拓展一点"并发"和"并行"的概念：

普通解释：
并发：交替做不同事情的能力
并行：同时做不同事情的能力
专业术语：
并发：不同的代码块交替执行
并行：不同的代码块同时执行

那么问题来了，Python为什么要如此设计呢？即**为什么要保证同一时刻只有一个线程在解释器中运行呢**?

答案是为了**Python解释器中原子操作的线程安全。**

2. 什么是线程安全，什么又是原子操作?

2.1 线程安全

我们首先要搞清楚什么是进程，什么是线程。**进程是系统资源分配的最小单位，线程是程序执行的最小单位。**

举一个例子，如果我们把跑程序比作吃饭，那么进程就是摆满了饭菜的桌子，线程就是吃饭的那个人。

在多线程环境中，当各线程不共享数据的时候，那么一定是线程安全的。问题是这种情况并不多见，在多数情况下需要共享数据，这时就需要进行适当的同步控制了。

线程安全一般都涉及到synchronized，就是**多线程环境中，共享数据同一时间只能有一个线程来操作 不然中间过程可能会产生不可预制的结果。**

接着刚才的**例子**，桌子上有三碗米饭，一个人正在吃，吃了两碗米饭，但是还没有吃完，因此桌子上米饭的数量还没有更新；此时第二个人也想吃米饭，如果没有线程安全方面的考虑，第二个人要是想直接拿三碗米饭吃，就会出错。

2.2 原子操作

2.2.1 什么是原子操作

2018年5月(8)
2018年4月(6)
2018年3月(3)
2017年9月(1)
2017年5月(2)
2017年1月(1)
2016年11月(1)
2016年8月(1)
2016年7月(2)
2016年6月(4)
2016年4月(1)
2015年10月(1)
2015年9月(8)
2015年8月(2)

最新评论

1. Re:gcc 与 g++的区别（转）
是GNU不是GUN啦

--逍遥哥ShPd

2. Re:Git gnutls_handshake() failed解决办法
@不瘦的胖子 我遇到过这种情况 你可以找到当前project的git文件，找到submodule的链接，把https改成http 或者按照文章最后的方法 把git的https的proxy server...

--青山牧云人

3. Re:谈谈Python中元类Metaclass(二):ORM实践
老哥写的真好，学习了

--糖炒栗子AA

4. Re:Git gnutls_handshake() failed解决办法
请问，如果是git clone --recursive http://xxx.

时出现这个错误该怎么改，submodule使用的还是https

--不瘦的胖子

5. Re:Python包的相对导入时出现错误的解决方法
@蔚蓝的蓝 我推断你是直接执行了binary_tree_operations.py，此时binary_tree_operations.py为主函数，那么文件夹binarytree不会被解析为pack...

--青山牧云人

阅读排行榜

1. Python包的相对导入时出现错误的解决方法(76233)
2. 如何用git命令生成Patch和打Patch(75242)
3. 在Python中建立N维数组并赋初值(37371)
4. 利用Python从文件中读取字符串（解决乱码问题）(32874)

原子操作就是不会因为进程并发或者线程并发而导致被中断的操作。原子操作的特点就是**要么一次全部执行，要么全不执行。**不存在执行了一半而被中断的情况。

当对全局资源存在写操作时，如果不能保证写入过程的原子性，会出现脏读脏写的情况。

非原子操作示例：

```
import threading

count = 0

def run_thread():
    global count
    for i in range(10000):
        count += 1

t1 = threading.Thread(target=run_thread,args=())
t2 = threading.Thread(target=run_thread,args=())
t1.start()
t2.start()
t1.join()
t2.join()

print(count)
```

如果运行上面的代码，打印出的count的结果是不确定的，它会小于20000.但count实际是被增加了20000次。

为什么这样呢？其实就是这里的写入操作`count += 1`并不是原子的。它实际经过了三步：

1. 读入count变量指向的值；
2. +1
3. 让count变量指向新的结果值。

原子操作示例：

```
lst = [4, 1, 3, 2]

def foo():
    lst.sort()
```

2.2.2 如何分辨原子操作与非原子操作

比如对下面这个函数：

```
n = 0

def foo():
    global n
    n += 1
```

我们可以看到这个函数用 Python 的标准 dis 模块编译的字节码：

```
>>> import dis
>>> dis.dis(foo)
LOAD_GLOBAL              0 (n)
LOAD_CONST               1 (1)
INPLACE_ADD
STORE_GLOBAL             0 (n)
```

代码的一行中， n += 1，被编译成 4 个字节码，进行 4 个基本操作：

5. Intel CPU命名规则的简略解析(14684)

评论排行榜

1. Python包的相对导入时出现错误的解决方法(13)
2. 最清楚的01背包问题讲解(5)
3. Git gnutls_handshake() failed解决办法(3)
4. SSH小结(3)
5. 聊聊Python中的GIL(2)

推荐排行榜

1. SSH小结(6)
2. Python包的相对导入时出现错误的解决方法(6)
3. Python函数参数中的冒号与箭头(5)
4. 聊聊Python中的GIL(3)
5. 如何用git命令生成Patch和打Patch(3)

1. 将 n 值加载到堆栈上
2. 将常数 1 加载到堆栈上
3. 将堆栈顶部的两个值相加
4. 将总和存储回 n

记住，一个线程每运行 1000 字节码，就会被解释器打断夺走 GIL 。如果运气不好，这（打断）可能发生在线程加载 n 值到堆栈期间，以及把它存储回 n 期间。很容易可以看到这个过程会如何导致更新丢失：

```
threads = []
for i in range(100):
    t = threading.Thread(target=foo)
    threads.append(t)

for t in threads:
    t.start()

for t in threads:
    t.join()

print(n)
```

通常这个代码输出 100，因为 100 个线程每个都递增 n 。但有时你会看到 99 或 98 ，如果一个线程的更新被另一个覆盖。

所以，**尽管有 GIL，你仍然需要加锁来保护共享的可变状态**：

```
n = 0
lock = threading.Lock()

def foo():
    global n
    with lock:
        n += 1
```

3. GIL的优点与缺点

GIL的优点是显而易见的，GIL可以保证我们在多线程编程时，无需考虑多线程之间数据完整性和状态同步的问题。

GIL缺点是：我们的多线程程序执行起来是"并发"，而不是"并行"。因此执行效率会很低，会不如单线程的执行效率。

网上很多人都提到过这样的疑问："**为什么我多线程Python程序运行得比其只有一个线程的时候还要慢**?"显然，大家觉得一个具有两个线程的程序要比其只有一个线程时要快。事实上,这个问题是确实存在的，原因在于GIL的存在使得Python多线程程序的执行效率甚至比不上单线程的执行效率。很简单，**由于GIL使得同一时刻只有一个线程在运行程序，再加上切换线程和竞争GIL带来的开销，显然Python多线程的执行效率就比不上单线程的执行效率了。**

4. 为什么会有GIL，GIL的历史

大家显然会继续思考，**为什么GIL需要保证只有一个线程在某一时刻处于运行中?难道不可以添加细粒度的锁来阻止多个独立对象的同时访问?**并且为什么之前没有人去尝试过类似的事情?

这些实用的问题有着十分有趣的回答。首先要明确一点，Python解释器的实现是有多个版本的：CPython，Jpython等。CPython就是用C语言实现Python解释器，JPython是用Java实现Python解释器。那么 GIL的问题实际上是存在于CPython中的。**GIL的问题得不到解决，一方面是因为CPython中一开始就使用GIL的设计理念，并且很多Package依赖于CPython甚至依赖于GIL。因此造成尾大不掉，实际上是个历史问题。**

为了利用多核，Python开始支持多线程。而解决多线程之间数据完整性和状态同步的最简单方法自然就是加锁。 于是有了GIL这把超级大锁，而当越来越多的代码库开发者接受了这种设定后，他们开始大量依赖这种特性（即默认python内部对象是thread-safe的，无需在实现时考虑额外的内存锁和同步操作）。

慢慢的这种实现方式被发现是蛋疼且低效的。但**当大家试图去拆分和去除GIL的时候，发现大量库代码开发者已经重度依赖GIL而非常难以去除了。**有多难？做个类比，像MySQL这样的"小项目"为了把Buffer Pool Mutex这把大锁拆分成各个小锁也花了从5.5到5.6再到5.7多个大版为期近5年的时间，本且仍在继续。MySQL这个背后有公司支持且有固定开发团队的产品走的如此艰难，那又更何况Python这样核心开发和代码贡献者高度社区化的团队呢?

GIL对诸如当前线程状态和为垃圾回收而用的堆分配对象这样的东西的访问提供着保护。这是该实现的一种典型产物。现在也有其它的Python解释器(和编译器)并不使用GIL。虽然,对于CPython来说,自其出现以来已经有很多不使用GIL的解释器。

那么为什么不抛弃GIL呢?许多人也许不知道,在1999年,针对Python 1.5,一个经常被提到但却不怎么理解的"free threading"补丁已经尝试实现了这个想法,该补丁来自Greg Stein。在这个补丁中,GIL被完全的移除,且用细粒度的锁来代替。然而,**GIL的移除给单线程程序的执行速度带来了一定的代价。当用单线程执行时,速度大约降低了40%。使用两个线程展示出了在速度上的提高,但除了这个提高,这个收益并没有随着核数的增加而线性增长。由于执行速度的降低,这一补丁被拒绝了**,并且几乎被人遗忘。

不过,"free threading"这个补丁是有启发性意义的,其证明了一个关于Python解释器的基本要点:移除GIL是非常困难的。由于该补丁发布时所处的年代,解释器变得依赖更多的全局状态,这使得想要移除当今的GIL变得更加困难。值得一提的是,也正是因为这个原因,许多人对于尝试移除GIL变得更加有兴趣。困难的问题往往很有趣。

但是这可能有点被误导了。让我们考虑一下:如果我们有了一个神奇的补丁,其移除了GIL,并且没有对单线程的Python代码产生性能上的下降,那么我们将会获得我们一直想要的:一个线程API可能会同时利用所有的处理器。但这确实是一个好事吗?

基于线程的编程毫无疑问是困难的。在编码过程中，总是会悄无声息的出现一些新的问题。因此有一些非常知名的语言设计者和研究者已经总结得出了一些线程模型。就像某个写过多线程应用的人可以告诉你的一样,不管是多线程应用的开发还是调试都会比单线程的应用难上数倍。程序员通常所具有的顺序执行的思维模恰恰就是与并行执行模式不相匹配。**GIL的出现无意中帮助了开发者免于陷入困境。在使用多线程时仍然需要同步的情况下,GIL事实上帮助我们保持不同线程之间的数据一致性问题。**

所以**简单的说GIL的存在更多的是历史原因**。如果推到重来，多线程的问题依然还是要面对，但是至少会比目前GIL这种方式会更优雅。

5. 如何规避GIL带来的影响

用multiprocess（多进程）替代Thread（推荐）

multiprocess库的出现很大程度上是为了弥补thread库因为GIL而低效的缺陷。它完整的复制了一套thread所提供的接口方便迁移。唯一的不同就是它使用了多进程而不是多线程。**每个进程有自己的独立的GIL，因此也不会出现进程之间的GIL争抢。**

当然multiprocess也不是万能良药。它的引入会增加程序实现时线程间数据通讯和同步的困难。就拿计数器来举例子，如果我们要多个线程累加同一个变量，对于thread来说，申明一个global变量，用thread.Lock的context包裹住三行就搞定了。而multiprocess由于进程之间无法看到对方的数据，只能通过在主线程申明一个Queue，put再get或者用

share memory的方法。这个额外的实现成本使得本来就非常痛苦的多线程程序编码，变得更加痛苦了。

用其他解析器（不推荐）

之前也提到了既然GIL只是CPython的产物，那么其他解析器是不是更好呢？没错，像JPython和IronPython这样的解析器由于实现语言的特性，他们不需要GIL的帮助。然而由于用了Java/C#用于解析器实现，他们也失去了利用社区众多C语言模块有用特性的机会。所以这些解析器也因此一直都比较小众。毕竟功能和性能大家在初期都会选择前者，Done is better than perfect。

GIL与互斥锁

值得注意的是GIL 并不会保护开发者自己编写的代码。这是因为同一时刻固然只能有一个 Python 线程得到执行，但是，当这个线程正在操作某个数据结构的时候，其他线程可能会打断它，一旦发生这种现象，就会破坏程序的状态，从而使相关的数据结构无法保持其一致性。为了保证所有线程能够得到公平地执行，Python 解释器会给每个线程分配大致相等的处理器时间。为了达到这样的分配策略，Python 系统可能当某个线程正在执行的时候将其暂停，然后使另一个线程继续往下执行。由于我们无法提前获知 Python 系统会在何时暂停这些线程，所以我们无法控制程序中某些操作是原子操作。

为了防止线程中出现数据竞争的行为，使开发者可以保护自己的数据结构不受破坏，Python 在 threading 模块中提供了最简单、最有用的工具：Lock 类，该类相当于互斥锁。

在开发中我们可以使用互斥锁来保护某个对象，使得在多线程同时访问某个对象的时候，不会将该对象破坏。因为同一时刻，只有一个线程能够获得这把锁。也就是说对将要访问的对象进行隔离，那么使用线程隔离的意义在于：是当前线程能够正确的引用到它自己创造的对象，而不是引用到其它线程锁创建的对象。

总结

Python GIL其实是功能和性能之间权衡后的产物，它尤其存在的合理性，也有较难改变的客观因素。我们可以做以下一些简单的总结：

- **因为GIL的存在，只有IO Bound场景下得多线程会得到较好的性能**
- **如果对并行计算性能较高的程序可以考虑把核心部分也成C模块，或者索性用其他语言实现**
- **在Python编程中，如果想利用计算机的多核提高程序执行效率，用多进程代替多线程**
- **即使有GIL存在，由于GIL只保护Python解释器的状态，所以对于非原子操作，在Python进行多线程编程时也需要使用互斥锁（如thread中的lock）保证线程安全。**
- **GIL在较长一段时间内将会继续存在，但是会不断对其进行改进**

参考链接：

1. 什么是线程安全和线程不安全 https://blog.csdn.net/zjy_android_blog/article/details/69817476

2. Python GIL https://www.aliyun.com/jiaocheng/446166.html

3. python之理解GIL https://www.jianshu.com/p/573aaa001b35

4. python面试不得不知道的点——GIL https://blog.csdn.net/weixin_41594007/article/details/79485847

5. 详解Python GIL https://blog.csdn.net/liangkaiping0525/article/details/79490323

6. 深入理解python多线程与GIL https://blog.csdn.net/ybdesire/article/details/77842438

7. python中的GIL详解 https://www.cnblogs.com/SuKiWX/p/8804974.html

8. 深入理解 GIL：如何写出高性能及线程安全的 Python 代码 http://python.jobbole.com/87743/

9. 谈谈有关 Python 的GIL 和 互斥锁 https://blog.csdn.net/Amberdreams/article/details/81274217

10. Python中的原子操作 https://www.jianshu.com/p/42060299c581

标签: python, GIL

青山牧云人
关注 - 1
粉丝 - 48

3 0

+加关注

« 上一篇: Python中的单元测试模块Unittest快速入门
» 下一篇: Python中的staticmethod和classmethod

posted @ 2018-11-19 18:50 青山牧云人 阅读(4079) 评论(2) 编辑 收藏

评论列表

#1楼 2018-11-23 14:32 你知道所有的未来 回复 引用

楼主转发一下，留原文地址

支持(0) 反对(0)

#2楼 [楼主] 2018-11-23 18:10 青山牧云人 回复 引用

@ 你知道所有的未来
好的

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

发表评论

编辑 预览

支持 Markdown

提交评论 退出 订阅评论

[Ctrl+Enter快捷键提交]