Gevent 学习分享

什么是Gevent？ 来自官网的一句话

Gevent是一种基于协程的Python网络库，它用到Greenlet提供的，封装了libevent事件循环的高层同步API

Greenlet： 微线程，即 协程，纤程，更官方一些，用户态线程

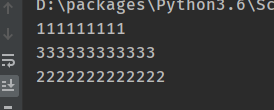
Libevent： C语言库，名字中带有一个 event 可以了解到，这是一个事件驱动库

所以 gevent 按我的理解，是一个由事件驱动的协程

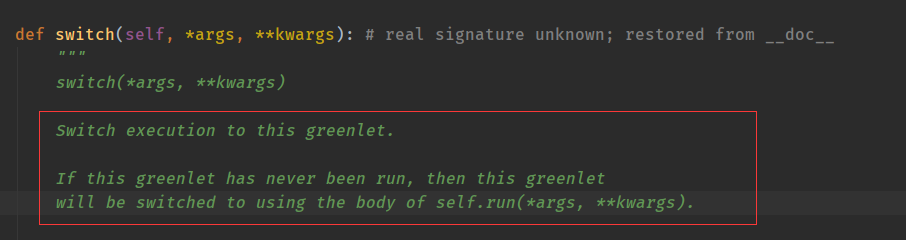
来看一段 greenlet 代码

大家觉得执行结果如何呢？





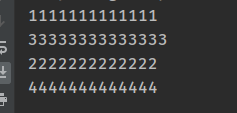
源码的解释



以 switch为驱动，运行当前协程

从 main greenlet 开始，走到 ①，遇到 switch，切换

那么gevent 呢 该如何运行



看到这，从代码和结果上判断，其实有两个问题：

1. gevent 似乎也要通过 gevent.sleep 去切换协程？
2. 为什么最后的 `444` 会打印？

第一个问题在下文中会展开讲，在这就先简单的解释下，用 gevent.sleep 是为了模拟io阻塞事件，即当程序执行过程中遇到阻塞事件，便自动切换协程。而且有没有发现，程序中并没有指定切换哪一个协程，而是由其底层自己切换的。其实说到这就能理解第二个问题为什么会打印了，当一个协程执行完之后，自动让出句柄，由其他未完成的协程获取。

那么协程切换时延呢？

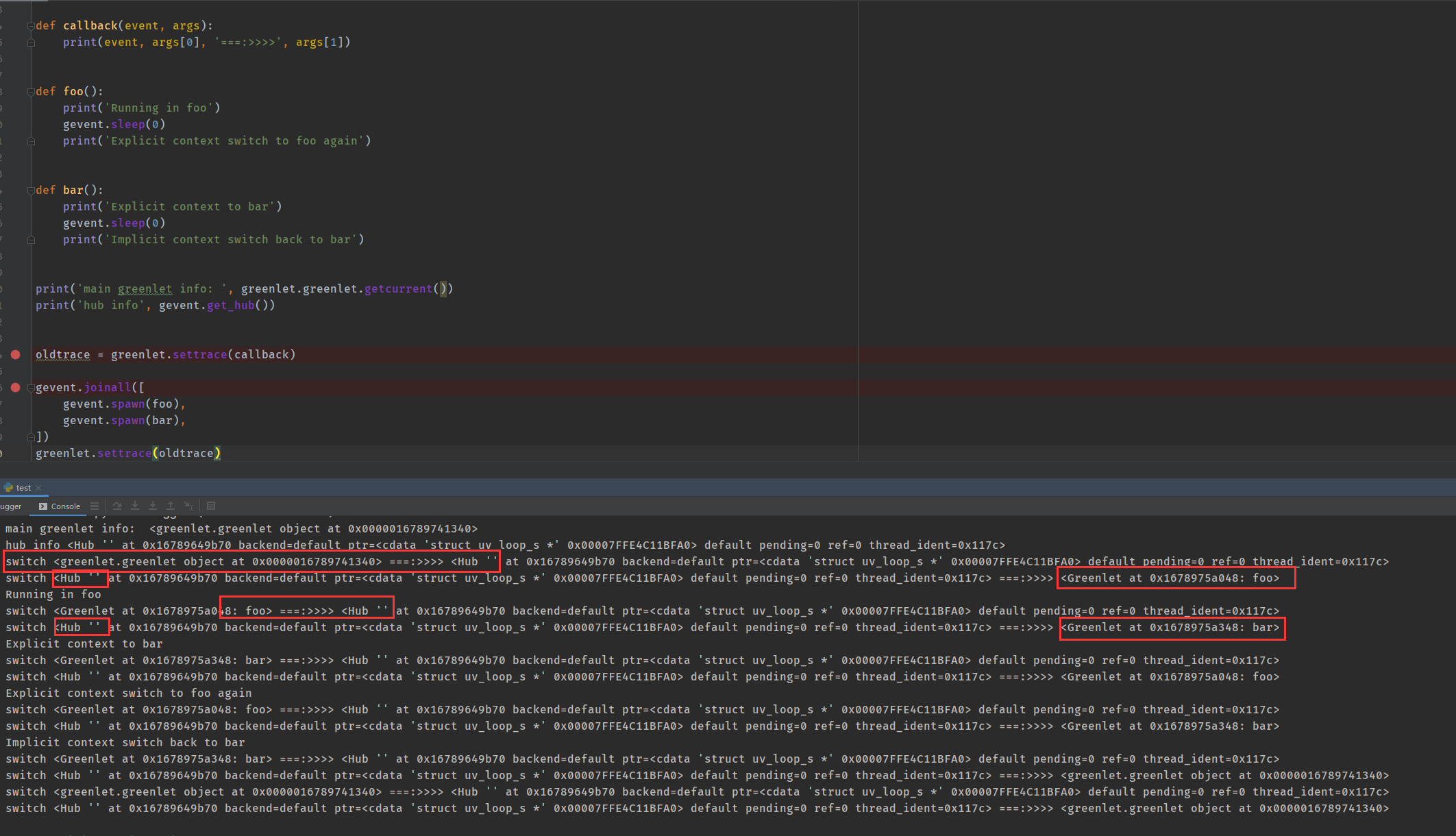
 

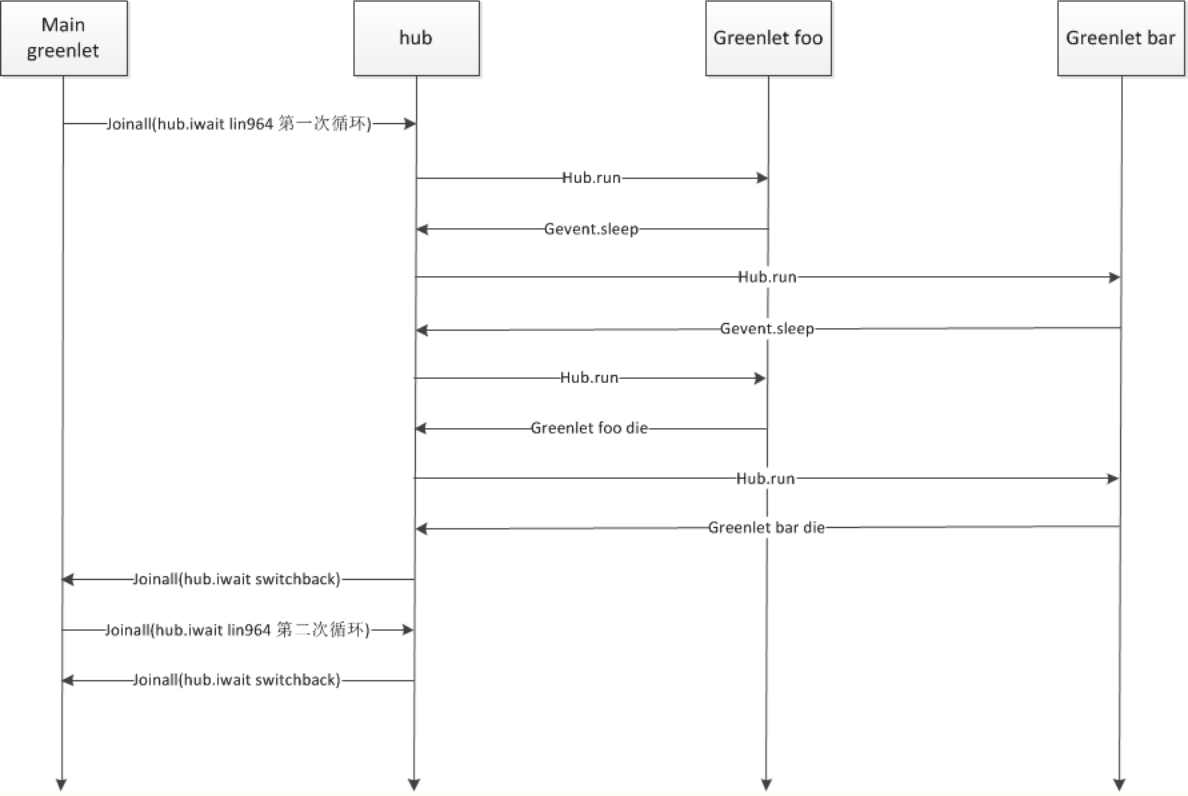
1000次切换 共2000次的 获取/释放 句柄 平均时延大概是 0.004/2000

10 000 次 共20 000次的 获取/释放 句柄 平均时延大概是 (0.037~0.04)/20 000

虽然有点不严谨，但可以看出在这种情况下，greenlet切换的时间开销大约是在 10^-6 s级别的

所以在gevent 中 协程是如何切换的呢？





也许大家会好奇，为什么采用这种模式，为什么每次都要切换到hub？我想理由有二：

1.hub是事件驱动的核心，每次切换到hub后将继续循环事件。如果在一个greenlet中不出来，那么其它greenlet将得不到调用。（joinall 中有timeout 来控制协程阻塞

2.维持两者关系肯定比维持多个关系简单。每次我们所关心的就是hub以及当前greenlet，不需要考虑各个greenlet之间关系。

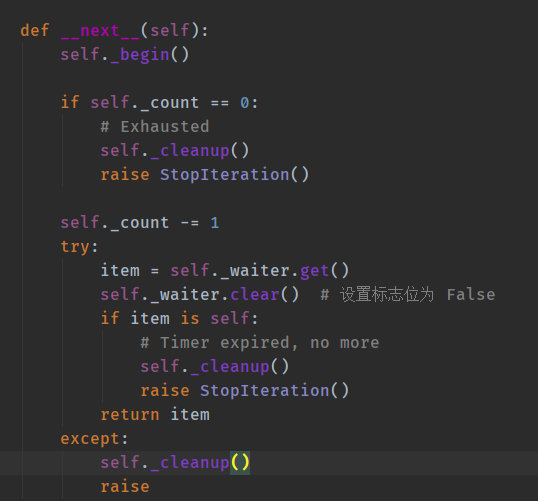
其实通过源码可以发现，hub的底层还是继承了 greenlet 类，并且并且执行switch 方法交出协程句柄，切换协程。由手动切换变成自动切换。

在 joinall 源码中有个可迭代对象，其中有这么一段代码 ，这其实和 epoll很像，

count = 就绪事件的个数，

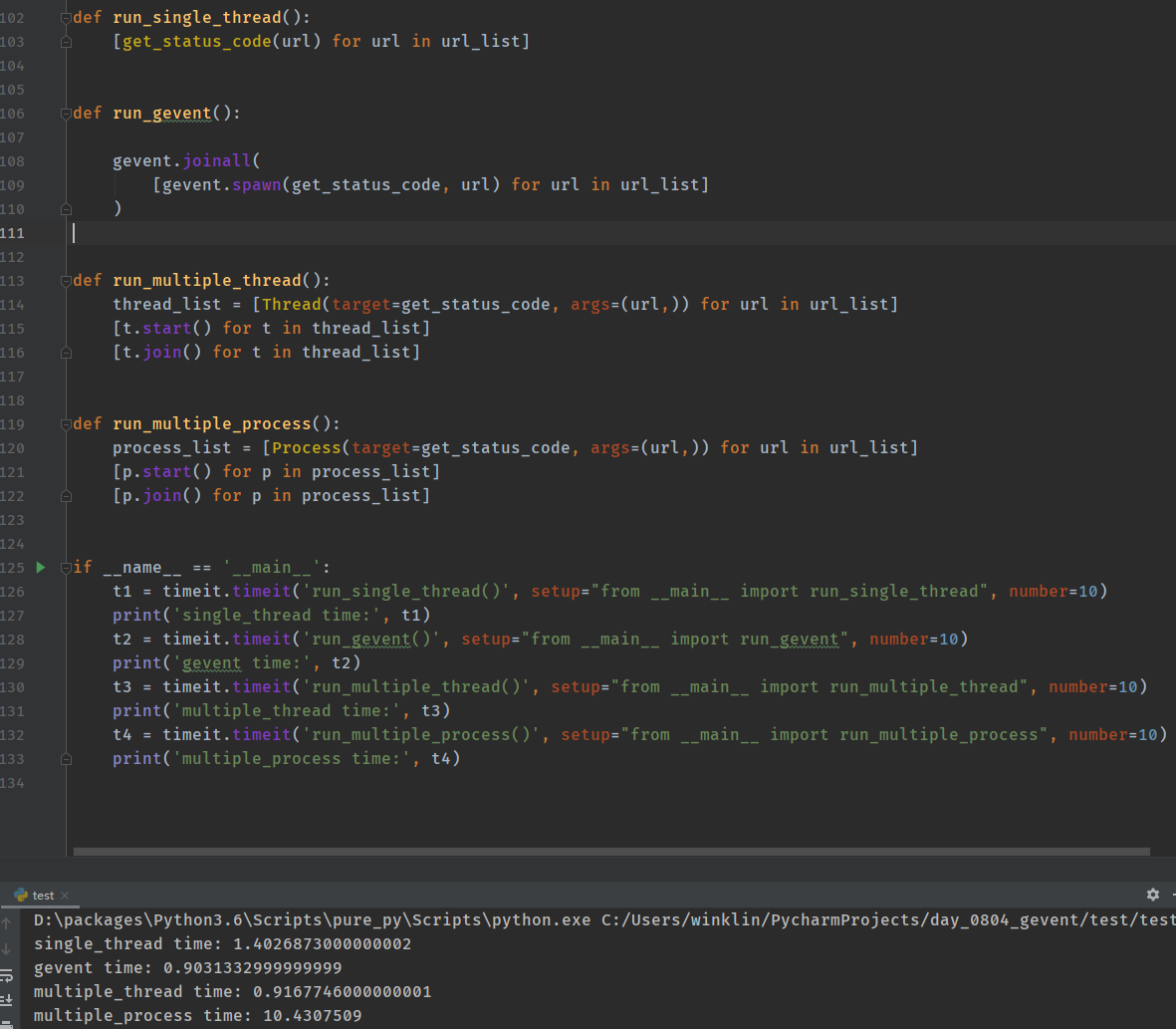
调用get 方法，（其实是switch 到 hub），让出句柄，并清除标志位（底层为 queue，从队列中移除）如果由 main greenlet 获取句柄则结束。

因为count是已知的，所以每次遍历只需 o（1）的时间复杂度



其实 gevent 就是基于 epoll实现的，这边就不展开了。

接下去是 gevent 和 单线程 多线程 多进程 的性能对比



gevent > 多线程 > 单线程 > 多进程

因为 gevent 是基于 epoll io多路复用，所不会像多线程一样有线程上下文的切换开销，也不会有单线程的阻塞，也没有多进程的进程调度的 用户态-内核态-用户态 的切换。

但 gevent 终归是线程中运行的，在逻辑上看起来是异步的，但当某个 greenlet 发生阻塞时，就无法switch 到 hub，别的协程就无法获取句柄，所以在内核层级上，它仍然是一个同步io。

在这种情况下，就无法充分利用多核资源，但更适用于io密集型的场景。