AUG 6TH, 2016

Daemon is not daemon, but what is it?

"daemon thread" 是一个困扰了我很久的概念。官方文档是这么说的:

> A thread can be flagged as a “daemon thread”. The significance of this flag is that the entire Python program exits when only daemon threads are left.

然而文档并没有解释这个概念是怎么来的。比如，为什么我在操作系统课上没听过呢？下面是一番搜索之后的结果。

历史

Daemon(守护进程)

daemon，或者 daemon process，在计算机领域里一般指一种在后台执行的程序。这里有一个简单明了的解释：

> "Daemon (Daemon Process)" is a notion in UNIX denoting a process detached from any controlling terminal, typically waiting for some event to occur and to respond to in some way. Windows services are similar but I assume Bil et alia chose deliberately a different word for them.

常见的守护进程就是 Linux 里的那一堆 “xxxd” 程序。下文中为避免歧义，称这里的 daemon 为守护进程。

顺便说一句，Python 有专门的 PEP 和库(python-daemon) 来实现守护进程。但这并非本文重点，而且我也不懂ヽ(´_ゝ)ノ

daemon thread 和守护进程没什么卵关系

是的，他们主要的共同点就是都包含单词“daemon”。但非要说一点联系没有也是不对的。一般而言，被 flag 为 daemon 的线程需要长期在后台执行（比如发送心跳包、检查未读消息等），并且不需和用户直接交互，和守护进程类似。

Python 中的 daemon thread 来自 Java

Google “daemon thread”，第一页全是 Java。我觉得很奇怪，于是找到了 `threading.py` 的第一次 commit，前两行赫然写着：

```
# threading.py:
# Proposed new threading module, emulating a subset of Java's threading model
```

当初看 `concurrent.futures` 源码的时候我还在想“直接照搬 Java 的 API 真的好么”，没想到 Guido 居然在 98 年就叛变革命了……

既然 `threading.py` 是抄来的，`daemon` 的概念自然也是，之前 Python 的线程 API `thread` 里可是没有 daemon 的。顺便说一句现在 `thread` module 变成了 `_thread`，功能还是一样，对操作系统的线程作了最基本的封装。

Java 文档如是说：

> Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority. Each thread may or may not also be marked as a **daemon**. When code running in some thread creates a new Thread object, the new thread has its priority initially set equal to the priority of the creating thread, and is a **daemon** thread if and only if the creating thread is a **daemon**.
>
> When a Java Virtual Machine starts up, there is usually a single non-daemon thread (which typically calls the method named main of some designated class). The Java Virtual Machine continues to execute threads until either of the following occurs:
> * The exit method of class Runtime has been called and the security manager has permitted the exit operation to take place. * All threads that are not **daemon** threads have died, either by returning from the call to the run method or by throwing an exception that propagates beyond the run method.

嗯，基本一个意思。

应用

Daemon thread 的特性很容易验证，不细说。

```
import threading
import os
```

```python
import time

def func():
    time.sleep(5)
    print("finish")

threading.Thread(target=func).start()
threading.Thread(target=func, daemon=True).start()
print("aaa")
```

更有意义的问题是，为什么 Java/Python 要引入 daemon thread，有什么用处？好在已经有人解释过了：

> Some threads do background tasks, like sending keepalive packets, or performing periodic garbage collection, or whatever. These are only useful when the main program is running, and it's okay to kill them off once the other, non-daemon, threads have exited.
>
> Without daemon threads, you'd have to keep track of them, and tell them to exit, before your program can completely quit. By setting them as daemon threads, you can let them run and forget about them, and when your program quits, any daemon threads are killed automatically.

简单来说就是，本来并没有 daemon thread，为了简化程序员的工作，让他们不用去记录和管理那些后台线程，创造了一个 daemon thread 的概念。这个概念唯一的作用就是，当你把一个线程设置为 daemon，它会随主线程的退出而退出。关键点有三个：

1. background task
2. only useful when the main program is running
3. ok to kill

被设置为 daemon 的线程应当满足这三条。第一点需要说一下，比如一个线程需要用 `join` 执行，那么 daemon 就没有意义了，因为程序总是需要等待它完成才能继续执行。

Daemon process

前面说到 Python 里的 daemon 概念抄自 Java，但 Guido 并未止步于此，他把 daemon 的概念推广到了多进程。当用 `multiprocessing` 创建新的进程时，也可以设置 daemon 属性。

Python 中有很多创建新进程的方法，并非都可以设置为 daemon process。下面列举一些：

`multiprocessing.Process`：可以设置 daemon
`os.system`：不可设置 daemon，因为该命令会阻塞当前程序的执行，不满足“background task”
`subprocess.Popen`：不可设置 daemon，因为 `Popen` 打开的是外部程序，不满足“only useful when the main program is running”
`concurrent.futures.ProcessPoolExecutor`：worker process 默认是 daemon

Daemon thread 的实现

最后来看一下 daemon thread 的实现，其实很简单。使用最初始（第一次commit）的 `threading.py` 来分析。

首先看 `_MainThread` 这个类，它表示主线程，在 `import threading` 的时候会初始化一个实例。`__init__` 函数里有这么一句：

```
_sys.exitfunc = self.__exitfunc
```

`_sys` 就是 `sys`，`sys.exitfunc` 已经被 `atexit` 替代，作用都是在程序退出的时候执行清理操作。那么 `__exitfunc` 是什么呢？

```
def __exitfunc(self):
    self._Thread__stop()
    t = _pickSomeNonDaemonThread()
    if t:
        if __debug__:
            self._note("%s: waiting for other threads", self)
    while t:
        t.join()
        t = _pickSomeNonDaemonThread()
    if self.__oldexitfunc:
        if __debug__:
            self._note("%s: calling exit handler", self)
        self.__oldexitfunc()
    if __debug__:
        self._note("%s: exiting", self)
    self._Thread__delete()
```

首先调用基类 `Thread` 中的 `__stop` 函数。下面是关键：

1. 首先，`t = _pickSomeNonDaemonThread()` 顾名思义返回一个 non-daemon 线程。`_pickSomeNonDaemonThread` 其实就是遍历两个保存了已创建和创建中线程的字

典，并检查其 daemon 属性，如果是 non-daemon 则返回。

2. 这三句是个人都知道在干嘛吧

```
while t:
  t.join()
  t = _pickSomeNonDaemonThread()
```

3. 于是所有 non-daemon threads 都执行完了，主线程退出。