记一次调试python内存泄露的问题

2018-05-04阅读 1.8K0

这两天由于公司需要， 自己编写了一个用于接收dicom文件(医学图像文件)的server. 经过各种coding-debuging-coding-debuging之后, 终于上线了, 上线后心里美滋滋的, 一切正常.

第二天一上班, 负责人和我说接收太慢了, 卡的要死. 我想难道是python本身的问题?(程序员本征思维)我好奇的打开了终端输入

ps –aux | grep python

找到进程id：

即 21610

我这里还没传几张图片就到78m了, 看来是内存问题. 其实生产环境占用更多, 因为生产环境保密所以只能在测试环境测试比较少的数据, 生产环境曾一度上升到3.7g的内存占用.

这样果断不行啊. 我发现有新的文件上传之后内存占用就会增大, 初步断定是dicom文件相关对象占用的内存. 现在的首要工作就是找到一个能进行内存泄露的调试工具了.

说道这里可能大家会有疑问, python作为动态类型语言同时拥有垃圾回收机怎么会有内存泄露? 其实也有可能出现内存泄露的情况, 有如下几种:

1. 对象一直被全局变量所引用, 全局变量生命周期长.
2. 垃圾回收机被禁用或者设置成debug状态, 垃圾回收的内存不会被释放.
3. 也是非常罕见的内存泄露的方式就是今天遇到的问题, 我周旋这个问题两天才debug出来, 现在分享给大家.客官请您继续往下看

说到查看python内存泄露的工具, 其实有挺多, 现在简短介绍一下

* gc: python 内置模块, 函数少功能基本, 使用简单, 作为python开发者里边的内容必须过一遍
* objgraph: 可以绘制对象引用图, 对于对象种类较少, 结构比较简单的程序适用, 我这个一个库套一个库, 内存还用的这么多,
* guppy: 可以对堆里边的对象进行统计, 算是比较实用
* pympler: 可以统计内存里边各种类型的使用, 获取对象的大小

上边这些虽然有用但是总是搞不到点子上, 上边这些都需要改我的源程序, 比较费劲, 线上的代码不是说改就能改的, 而且他们功能也都比较弱, 后来发现两个强大的工具:

* tracemalloc: 究极强, 可以直接看到哪个(哪些)对象占用了最大的空间, 这些对象是谁, 调用栈是啥样的, python3直接内置, python2如果安装的话需要编译
* pyrasite: 牛逼的第三方库, 可以渗透进入正在运行的python进程动态修改里边的数据和代码(其实修改代码就是通过修改数据实现)

我开始的时候非常想用tracemalloc, 可是对python2特别不友好, 需要重新编译python, 而且只能用python2.7.8编译, 编译好了也不容易嵌入到虚拟环境中, 头大, 果断换第二个.

注: pyrasite使用之前需要在root用户下运行命令 echo 0 > /proc/sys/kernel/yama/ptrace\_scope后才能正常使用

pyrasite里边有一个工具叫pyrasite-memory-viewer, 功能和guppy差不多, 不过可以对内存使用统计和对象之间的引用关系进行快照保存, 很易用也很强大.运行

pyrasite-memory-viewer <pid>

可以看到占用内存最多的是DicomFileLike这种类型的对象.已经达到上万个, 这是不能忍受的.

就目前来看可能会有上边说的两种内存泄露原因导致不能回收这个对象.打开

pyrasite-shellpyrasite-shell <pid>

我先通过

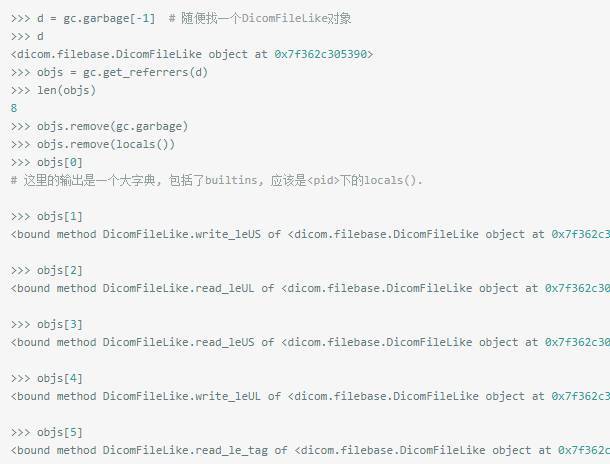
gc.isenabled()

判断gc是否在工作, 结果发现是True, 也就是正常工作的, 而且使用gc.setdebug(gc.STATUS)设置gc为debug模式, 然后gc.collect()进行垃圾回收发现并没有更多内存释放,则否认了第二种泄露的可能.

现在来看gc.garbage中不能被释放的对象, 让我来检查一下是否有全局变量指向它们(这里极有可能是一个列表或者是一个字典)

gc.garbage 可以看到被塞满了各种DicomFileLike对象

所以我们的目的就是先找到一个对象然后一级一级的向上寻找相互的引用.



到这里发现其实没有更多的全局变量指向这个d了, 而且发现所以有的方法的对象地址和d是相同的, 说明了这个对象其实是自循环引用的.

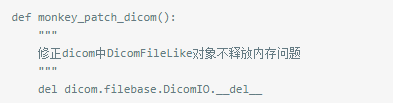
那么python不可能不支持循环引用对象的回收吧? 跟着这个问题我查了一下stackoverflow

Does Python GC deal with reference-cycles like this?

这个问题的第一个回答介绍的很清楚了, 如果用户不自定类的\_\_del\_\_方法, gc可以回收带有自引用的对象, 但是你自己实现了\_\_del\_\_方法就不行了.

这就是python内存泄露的第三个可能.

回头看DicomFileLike的源码, 果然在\_\_init\_\_函数上方定义了一个\_\_del\_\_函数, 我这里使用了一个猴子补丁删除了这个方法, 内存泄露的问题就得以解决了.



**总结**

到这里整个调试过程就结束了, 然而实际上过程中做了很多曲折的工作, 在pyrasite中会找到几个引用DicomFileLike对象的object, 比较不容易辨别, 最开始我以为是某个全局的对象引用的DicomFileLike, 比如是列表什么的, 后来发现其实是locals()和globals()字典, 如果使用pyrasite-memory-viewer保存下来的数据会发现有一个大列表指向所有没有回收的DicomFileLike对象, 捯饬半天发现其实是gc.garbage, 好囧, 曾让我一度怀疑是第一种泄露方式, 但是怎么找这个对象都没有找到. 其中还有几次看到线程达到140+, 后来发现其实和线程一点关系没有, 线程维持在这个数目上边很稳定.

在这个过程中用到的其他几个hack的技巧有:

**查看进程的线程数量**

ps –o nlwp <pid>

**根据对象的id/address动态获取对象**

import ctypes

obj = ctypes.cast(<addr\_or\_id>,ctypes.py\_object).value

**查看垃圾回收的日志**

gc.set\_debug(…)

作者：weidwonder 来源：http://www.jianshu.com/p/2d06a1a01cc3

参考2：

https://blog.csdn.net/muzico425/article/details/104404293?utm\_medium=distribute.pc\_relevant.none-task-blog-baidujs\_title-0&spm=1001.2101.3001.4242

# 记一次Pytorch内存泄露的排查与处理

https://blog.csdn.net/No\_\_names/article/details/106087249

# python 内存监控模块之memory\_profiler

https://www.cnblogs.com/kaituorensheng/p/5669861.html