一、内存泄露的原因

1、 所用到的用 C 语言开发的底层模块中出现了内存泄露。

2、 代码中用到了全局的 list、 dict 或其它容器，不停的往这些容器中插入对象，而忘记了在使用完之后进行删除回收

3、 代码中有“引用循环”， python 垃圾处理机制无法进行回收

二、 内存泄露的诊断思路

无论是哪种方式的内存泄露，最终表现的形式都是某些 python 对象在不停的增长；因此，首先是要找到这些异常的对象。

三、 内存泄露的诊断步骤

用到的工具： gc 模块和 [objgraph](http://mg.pov.lt/objgraph.py) 模块

objgraph 是一个用于诊断内存问题的有用的工具

1、 在服务程序的循环逻辑中，选择出一个诊断点

2、 在诊断点，插入如下诊断语句

Python代码

|  |
| --- |
| import gc  import objgraph  ### 强制进行垃圾回收  gc.collect()  ### 打印出对象数目最多的 50 个类型信息  objgraph.show\_most\_common\_types(limit=50) |

3、 检查统计信息，找到异常对象。

运行加入诊断语句的服务程序，并将打印到屏幕上的统计信息重定向到日志中。

运行一段时间后，就可以来分析日志，看看哪些对象在不停的增长。

以我的程序为例，我将日志记录到 log.txt 中，运行一段时间后，发现 tuple 和 list 类型的对象不停增长：

Shell代码

|  |
| --- |
| # grep "^list " log.txt  # grep "^tuple " log.txt |

如果不停增长的对象，是一些非通用的类型（例如你自己实现的一个 class），那么问题就比较好定位，例如 [Tracing Python memory leaks](http://www.lshift.net/blog/2008/11/14/tracing-python-memory-leaks)中提到的案例。

而对 tuple 和 list 这类通用类型，要想知道对象到底是什么，泄露发生在哪里，还得想点办法。我采用了排查的方式。由于程序的模块化还不错，可以每次禁用一个模块，然后重新跑程序，重新检查日志，看看 tuple 和 list 是否仍然不停增长。这样，很快就能将故障定位到具体的模块中。