[Google performance Tools (gperftools) 使用心得](http://www.cnblogs.com/persistentsnail/p/3294843.html)

Google performance Tools (gperftools) 使用心得  
  
  
gperftools是google开发的一款非常实用的工具集，主要包括：性能优异的malloc free内存分配器tcmalloc；基于tcmalloc的堆内存检测和内存泄漏分析工具heap-profiler,heap-checker；基于tcmalloc实现的程序CPU性能监测工具cpu-profiler.  
  
上述所说的三种工具在我们服务器进程的性能分析监控，定位内存泄漏，寻找性能热点，提高malloc free内存分配性能的各个方面上都有非常成功的使用经验。

**1.tcmalloc**

之前我们一直使用glibc的malloc free，也是相安无事，后来就出现了内存泄漏的问题，服务器在出现一段高负载的情况后，内存使用率很高，当负载降下来很长一段时间后，进程所占用的内存仍然保持高占用率下不来。一开始就确定是内存泄漏，判断是libevent收到的网络包没处理过来，在接收缓冲区越积越多，使得缓冲区占用的内存也越来越大（因为之前就有这样的经验）。我们开始来监控libevent的内存请况，因为libevent可以替换malloc free(google libevent的event\_set\_mem\_functions),我们将默认的malloc free替换为有日志信息的自己定制的版本，然后再启动服务器，分析结果。日志显示：在比较大压力下，libevent使用的内存确实很多，基本符合top显示出的内存使用情况，但当压力过去了，日志明显显示libevent已经释放掉了内存，可是top显示的内存使用还是很高。free掉的内存没有还回给系统，而且过了一两天也还是现状，我们都知道malloc free是用了内存池的，但是把长时间没使用的大块内存还给系统，不是应该是内存池该做的事吗？ 既然这只是glibc的malloc free实现，也许他的内存分配算法就是这样，可能有些其他原因也不一定，不了解其实现和原理，不好妄作评论。因为其实以前就有同事加上了tcmalloc，所以马上想换换tcmalloc试试，这也是听说过tcmalloc是最快的malloc free实现，google内部许多工程都是使用它做底层云云。  
  
  
换上tcmalloc很容易，根据需要以动态库或静态库的形式链接进你的可执行程序中即可（gcc ... -o myprogram -ltcmalloc)。使用tcmalloc后按照上述的情形再跑了一次，发现在同样的压力下，cpu占用率下降了10%～20%左右，甚是惊喜。但是在压力过去很长一段时间后，同样的问题出现了，内存占用率还是居高不下。这个非常困惑我，于是去网上去看官方文档，看其实现原理和特性。然后就在在文档发现了--把没用的内存还回给系统原来是可以控制的行为。  
  
  
以下完全参考官方文档http://google-perftools.googlecode.com/svn/trunk/doc/tcmalloc.html：  
在tcmalloc中控制把未使用的内存还回给系统，主要是两个方法，一种是修改环境变量TCMALLOC\_RELEASE\_RATE，这个值代表把未使用的内存还回给系统的速度吧，取值在0～10，值越高返还的速率越高，值为0表示从不返还，默认是1.0，其实是一个比较低的速率，所以在之前的表现中，就是很长一段时间内存降不下来。这个值的改变可以在程序运行期间就起作用。这个环境变量也有对应的函数调用SetMemoryReleaseRate，可以在程序代码中调用。还有一种是一个函数调用：  
 MallocExtension::instance()->ReleaseFreeMemory();  
这个函数如其名，把未使用的内存全部返还给系统。我按照第二种方法，让运行着的服务器程序动态执行我的命令，这个命令就是运行上面的函数调用（我们服务器有套机制，能够在运行时执行一些我们自定义的命令，实现方式有很多，如信号），结果top上显示的内存占用率立马降下来很多，顿时让我觉得世界都清净了。  
后记：我估计glibc的malloc free也有类似控制，只不过现阶段我并没有时间和很大兴趣去关注了。

**2.heap-profiler, heap-checker**

这两个工具其实挺像的，heap-checker专门检测内存泄漏，heap-profiler则是内存监控器，可以随时知道当前内存使用情况（程序中内存使用热点），当然也能检测内存泄漏。我们工作中一直是使用heap-profiler，实时监控程序的内存使用情况。文档在此：http://google-perftools.googlecode.com/svn/trunk/doc/heapprofile.html。  
heap-profiler是基于tcmalloc的，正规开启它的方法是在代码中调用 HeapProfilerStart方法，参数是profile文件前缀名，相应的关闭则需调用 HeapProfilerStop。前面有介绍过我们的服务器有运行时执行自定义命令的机制，我们把两个命令MemProfilerStart,MemProfilerStop实现成调用上述相应的开启/关闭heap-profiler的API，这样我们可以在服务器运行时，随时开启和关闭heap-profiler，非常方便的查看当前程序内存使用情况。  
  
  
heap-profiler会在每当一定量的内存被新申请分配出来时或者当一段固定时间过去后，输出含有当前内存使用情况的profile文件，文件名类似这种：  
    <prefix>.0000.heap  
    <prefix>.0001.heap  
    <prefix>.0002.heap  
           ...  
prefix是前文所说的profile数据文件的前缀，如果并没有指定成绝对路径则会在你的程序的工作目录中生成，这些文件可以被一个脚本工具pprof解析输出成各种可视的数据格式文件，pprof使用了dot语言绘图，需要安装graphviz。pprof也是下文解析CPU profile文件的工具，pprof工具可以把profile信息输出成多种格式，如pdf，png，txt等，如果是以图的形式显示，则是根据调用堆栈的有向图，像下图这样：  
  
  
这个图里面每个节点代表一个函数调用，比如GFS\_MasterChunkTableUpdateState节点，176.2(17%) of 729.9(70%) 大致表示这个函数本身自己直接消耗了17%的内存，加上子调用共消耗了70%的内存，然后每条边则显示每个子调用花费了多少内存。因为我们的linux系统并未安装图形界面，通常都是直接生成了txt文件：  
% pprof --text gfs\_master /tmp/profile.0100.heap  
   255.6  24.7%  24.7%    255.6  24.7% GFS\_MasterChunk::AddServer  
   184.6  17.8%  42.5%    298.8  28.8% GFS\_MasterChunkTable::Create  
   176.2  17.0%  59.5%    729.9  70.5% GFS\_MasterChunkTable::UpdateState  
   169.8  16.4%  75.9%    169.8  16.4% PendingClone::PendingClone  
    76.3   7.4%  83.3%     76.3   7.4% \_\_default\_alloc\_template::\_S\_chunk\_alloc  
    49.5   4.8%  88.0%     49.5   4.8% hashtable::resize   
第一列代表这个函数调用本身直接使用了多少内存，第二列表示第一列的百分比，第三列是从第一行到当前行的所有第二列之和，第四列表示这个函数调用自己直接使用加上所有子调用使用的内存总和，第五列是第四列的百分比。基本上只要知道这些，就能很好的掌握每一时刻程序运行内存使用情况了，并且对比不同时段的不同profile数据，可以分析出内存走向，进而定位热点和泄漏。  
  
  
在我们的实践中，也经常发现一些环境变量非常有用：  
HEAP\_PROFILE\_ALLOCATION\_INTERVAL：上文说每当一定量的内存被新申请分配出来时，就会输出profile文件，这个变量值就是控制多少字节，默认是（1024\*1024\*1024）1GB，粒度相对比较大，常常会被我们调整为几百MB甚至更小。  
HEAP\_PROFILE\_MMAP:有时候程序申请内存的方式是通过mmap,sbrk系统调用而不是malloc free，如果想profile这些内存，可以开启这个变量，默认是false。我们工程代码中就有些调用了mmap申请大块内存的地方，开启这个环境变量，能更准确跟踪内存走向。  
HEAP\_PROFILE\_MMAP\_ONLY:如其名，只profile mmap，sbrk申请的内存。  
更改这些环境变量是可以在启动命令中完成的：  
% env HEAPPROFILE=/tmp/mybin.hprof /usr/local/bin/my\_binary\_compiled\_with\_tcmalloc

**3.CPU profiler**

CPU profiler的使用方式类似heap-profiler，区别就是要在构建你的程序时不仅要链接-ltcmalloc还要链接-lprofiler。它也是需要一个函数调用(ProfilerStart)来开启,和一个函数调用(ProfilerStop)来关闭，调用这些函数需要include <google/profiler.h>。当然我们还是通过内部的自定义指令机制来运行时控制profiler的开启和关闭。ProfilerStart接受输出文件名作参数，ProfilerStop关闭profiler时同时输出含有profile信息的文件，这些信息也是要pprof解析后可以生成各种可读格式：  
% pprof /bin/ls ls.prof  
                       Enters "interactive" mode  
% pprof --text /bin/ls ls.prof  
                       Outputs one line per procedure  
% pprof --gv /bin/ls ls.prof  
                       Displays annotated call-graph via 'gv'  
% pprof --gv --focus=Mutex /bin/ls ls.prof  
                       Restricts to code paths including a .\*Mutex.\* entry  
% pprof --gv --focus=Mutex --ignore=string /bin/ls ls.prof  
                       Code paths including Mutex but not string  
% pprof --list=getdir /bin/ls ls.prof  
                       (Per-line) annotated source listing for getdir()  
% pprof --disasm=getdir /bin/ls ls.prof  
                       (Per-PC) annotated disassembly for getdir()  
% pprof --text localhost:1234  
                       Outputs one line per procedure for localhost:1234  
% pprof --callgrind /bin/ls ls.prof  
                       Outputs the call information in callgrind format  
我们实践中用的最多的是导出成pdf格式，非常直观，描述文档在此http://google-perftools.googlecode.com/svn/trunk/doc/heapprofile.html，因为这个有向图代表的意义类似之前在heap-profiler中所描述的图，所以不再多着笔墨。实践证实CPU profiler效果很好，为我们一次次定位了性能热点，为此让人好奇其实现原理。CPU profiler的原理类似许多其他proflier，都是对运行着的程序定时采样，最后根据每次记录的堆栈频度导出采样信息。网上有人这样解释到：相当于用gdb attach一个正在运行的进程，然后每隔一段时间中断程序打印堆栈，当然耗时最多的调用最频繁的堆栈是最常被打印出来的。有稍微看过gperftools这方面的实现，大致就是注册一个定时触发的信号（SIGPROF）处理函数，在此函数中获取当前堆栈信息，通过hash算法以此做hash表的key，放入样本统计hash table中，如果hash table中已经有同样的堆栈key了，value就加1，这样当采样结束，就把hash table的统计信息导出到文件供pprof程序解析，从而得到真正直观的profile信息。  
  
  
  
最后，如果根本不想使用任何profiler功能，只想使用其快速的tcmalloc，可以直接链接-ltcmalloc\_minimal。