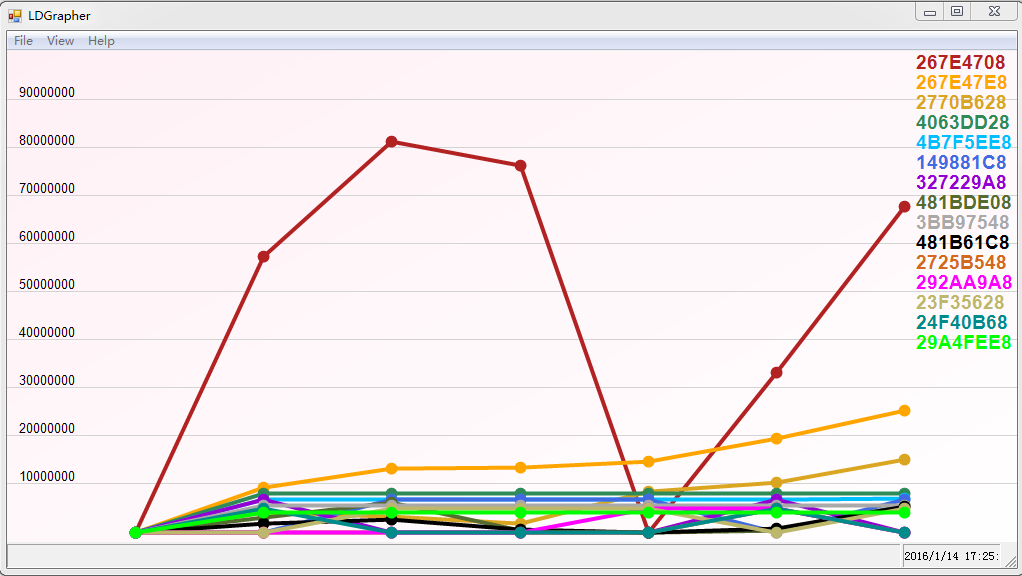
# LeakDiag使用说明

##### 介绍

LeakDiag是微软开发的内存泄露检测工具。需要配合LDGrapher工具使用。LDGrapher是LeakDiag的Log图形化分析工具。

通过它定位内存泄露的过程大致如下：

1. 在程序运行的过程中，记录某个时间点时，程序在Windows Heap中的内存申请情况，并将它打印到一个xml格式的log文件中，该log文件记录了所有函数堆栈在当前时间点时，在Heap中申请后未释放的内存大小和申请次数
2. 通过记录多个时间点的内存申请情况，再绘制成折线图，就可以知道函数堆栈在Heap中申请内存的变化情况。如果某个函数堆栈申请的内存一直在增长从未被释放，那么这个函数堆栈很有可能就发生了内存泄漏
3. 最后再根据函数堆栈就可以定位到发生内存泄漏的代码了

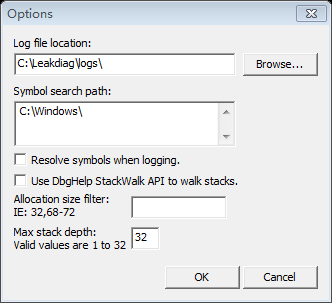


安装文件在这里：

[\\192.168.1.249\诺亚内\程序部\wc无尘\LeakDiag和配套查看工具LDGrapher](\\\\192.168.1.249\\诺亚内\\程序部\\wc无尘\\LeakDiag和配套查看工具LDGrapher)

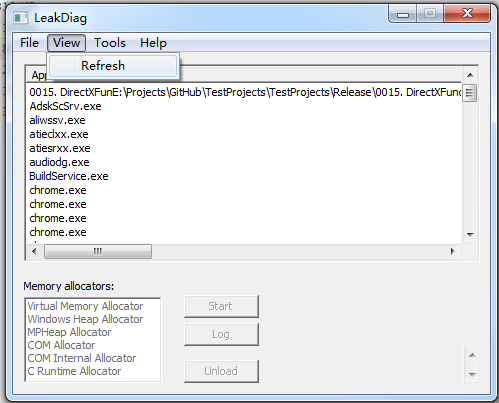
##### 安装与配置

* 1. 安装LeakDiag和LDGrapher，建议直接安装在默认路径（C盘根目录）
     + LeakDiag安装过程中可能会报错，忽略即可。
  2. 拷贝RunSdecode.bat到LeakDiag的安装目录
     + RunSdecode.bat是我写的用来调用sdecode.exe的脚本
     + sdecode.exe是对LeakDiag的log进行符号解析的工具
     + 符号解析（Symbols Resolve）：根据函数二进制地址和\*.pdb文件获取源码文件中的函数名
  3. LeakDiag主界面 -> Tools下拉菜单 -> Options，修改LeakDiag默认配置：
     + 将Resolve symbols when logging设置为不勾选（大型程序若勾选此项，打log时会无响应）
     + 将Max stack depth设为32（跟踪函数堆栈的最大深度）
     + 其他设置保持默认即可

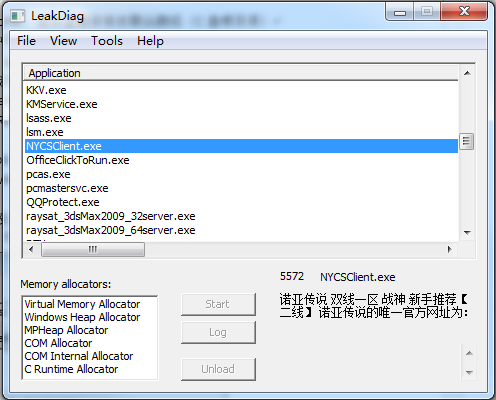


##### 使用方法

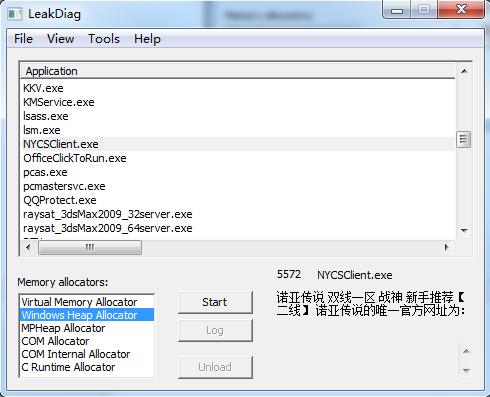
* 1. 准备工作
     + 打开下列程序
       1. 诺亚客户端（待测程序），建议使用Debug版本客户端，否则符号解析得到的结果很可能不准确
       2. LeakDiag
       3. Windows任务管理器
     + 点击LeakDiag主界面 -> View下拉菜单 -> Refresh，刷新进程列表



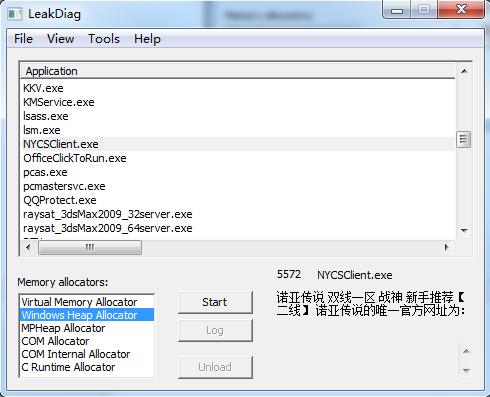
* + - 在进程列表中选中诺亚客户端的进程



* + - 在Memory allocators列表中选中Windows Heap Allocator

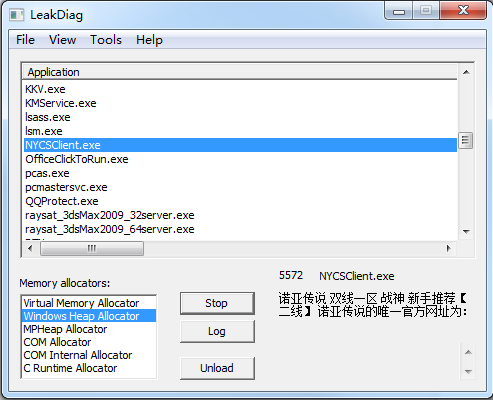


* + - 点击LeakDiag主界面下方的Start按钮，开始监控程序的内存分配



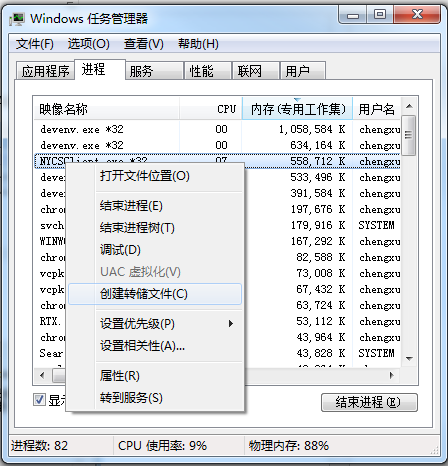
* 1. 获取内存分配信息
     + 点击LeakDiag主界面的Log按钮，获取log文件，可能会造成程序卡顿。

（保存在设置好的log文件保存目录中）

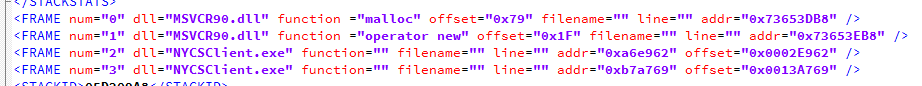


* + - 在Windows任务管理中右键诺亚进程，选择创建拓展名为DMP的内存转储文件，此文件将被用于对上一步获取的log文件进行符号解析，它与LeakDiag的log文件是一一对应的。这步操作可能会造成程序卡顿。

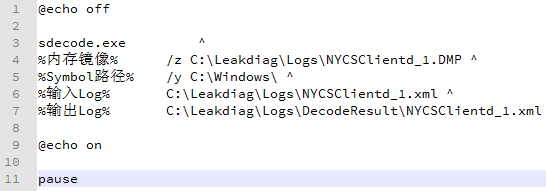
（保存在C:\Users\[Windows用户名]\AppData\Local\Temp文件夹中）



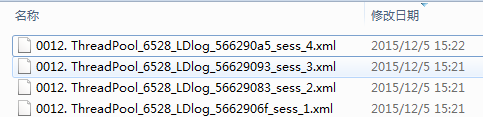
* + - 注意，右键LeakDiag主界面进程列表中的诺亚进程也可以创建内存转储文件，但该文件用于符号解析会出错。
  1. 解析log文件符号
     + 在执行符号解析前，LeakDiag得到的内存分配log是看不到函数名和文件名等信息的，如下图所示，function、filename、line等信息为空



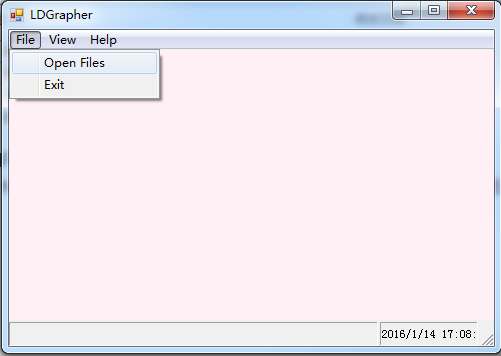
* + - 修改RunSdecode.bat，配置相应的参数。注意：
      1. 输出Log的文件夹一定要提前创建好
      2. DMP文件一定要与输入log文件对应
      3. Symbol路径无需修改

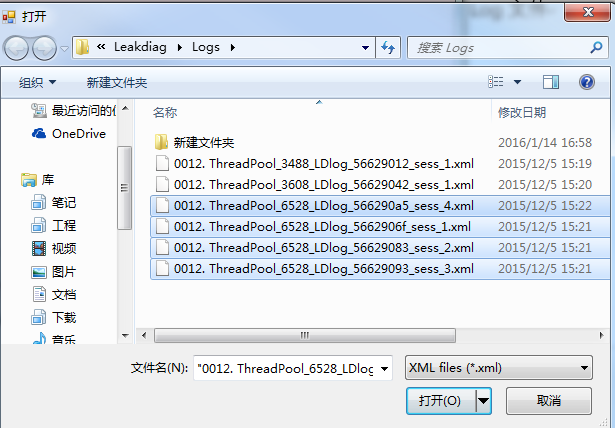


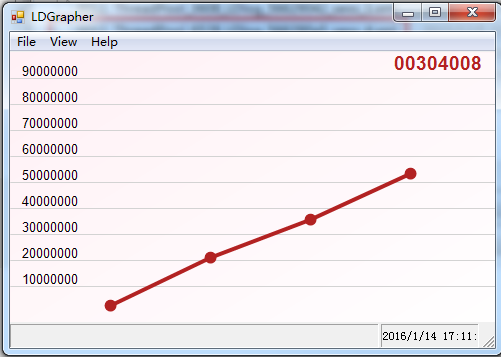
* + - 执行RunSdecode.bat，注意：
      1. 这一步需要的时间很长！分析一个200M的log需要6至8个小时
      2. 由于分析程序是单线程的，所以如果想充分利用CPU的话可以同时执行多个RunSdecode.bat脚本，充分利用CPU（建议同时跑3-4个）
      3. 如果运行了多个RunSdecode.bat脚本，分析完后需要保证输出log的文件修改时间顺序和打log的顺序一致，因为LDGrapher绘制内存分配曲线是按log的文件修改时间对每个内存分配节点进行排序的。为了达到这个目的可以用Notepad++把每个输出log按顺序打开，加一个空格，再删除空格，最后Ctrl + S保存。



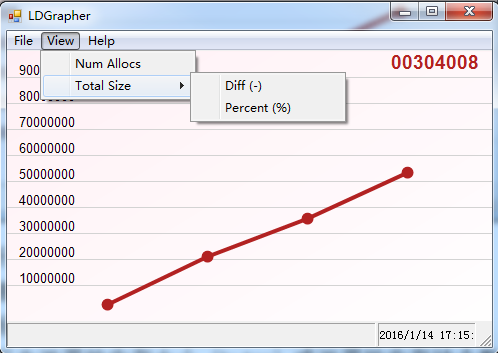
* 1. 使用LDGrapher获取图像化分析结果
     + LDGrapher主界面 -> File -> Open File，一次性选中所有的输出Log文件







* + - 得到上图所示的图像，其中
      1. 每条折线代表疑似发生内存泄漏的函数堆栈，右上角为堆栈的编号，一直在增长的折线发生内存泄露的可能性最大
      2. 横坐标为打Log的时间点
      3. 纵坐标为泄漏的内存大小（bytes）或泄漏的内存块个数，通过View下拉菜单可以进行相应的切换



* + - 双击折线或右上角编号可以查看折线对应的函数调用堆栈，根据堆栈就可以定位到发生内存泄露的代码了。

