

Windows程序性能优化

性能问题是什么

• 性能问题实例

读出一段数据列表(20条)用了14秒左右,感觉比较慢

上网打开页面很慢, 打开一个网页要3-4秒钟

ESET NOD32绝不拖慢计算机, 侦测速度比其竞争对手快3到34倍

网络硬盘上传下载文件慢

登录卡的时间长短取决于电脑配置、网速、好友数、群数等QQ登录后需提取的相关数据

找好友往下拉的时候一晃一晃的卡死人

UCWEB, 资源消耗比较小, 耗流量小

内存占用, QQ2005正式版约为14M, 而MSN 8.0 Beta达到了30M

QQ2009装了卸了几次了, 最大的原因就是狂读硬盘, CPU占用高

玩LOL后者CF的时候, 卡一下我的角色就被打死

• 典型的软件性能问题

- 某个操作非常慢
- 卡住无响应
- CPU, GDI等资源占用多

软件性能应该包括哪些方面？

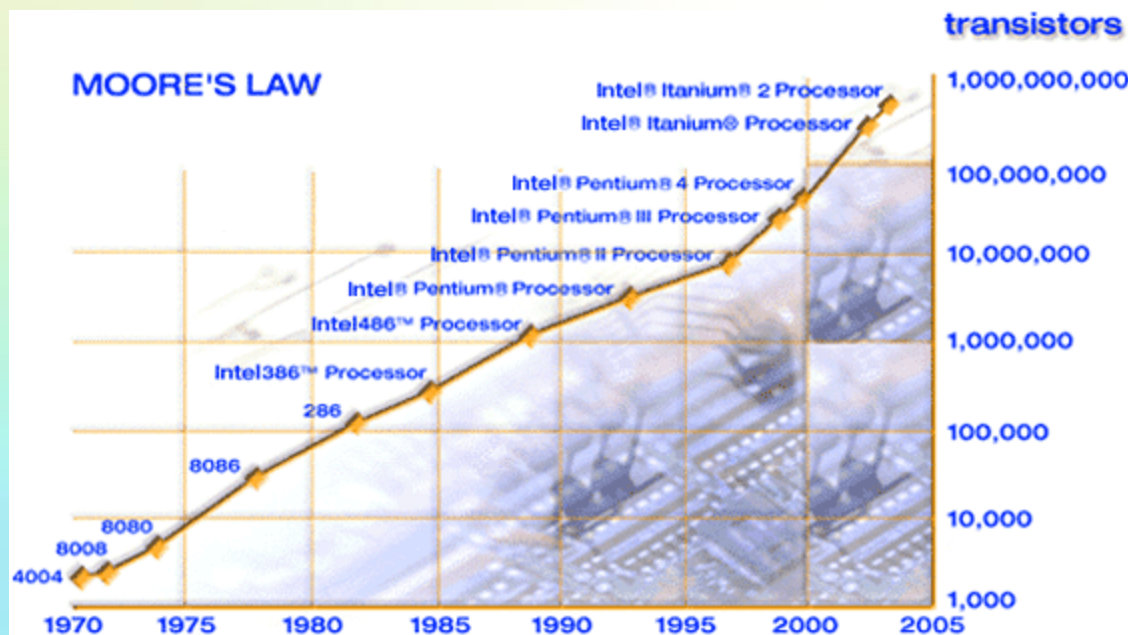
- **从表现角度来看**
 - 速度类
 - 资源消耗类
- **从软件行为的角度来看**
 - IO类
 - CPU运算类

内容大纲

- **性能工作之必要性**
- 如何提高性能
- 性能监控
- 建立性能备忘体系
- 性能与用户
- 性能工具

误区-硬件可以解决性能问题

- “硬件可以解决我的性能问题”
- CPU速度每两年翻一番, 就像戈登·摩尔在1975年预测的那样。



摩尔定律是指IC上可容纳的晶体管数目, 约每隔18个月便会增加一倍, 性能也将提升一倍。摩尔定律是由英特尔(Intel)名誉董事长戈登·摩尔 (Gordon Moore) 经过长期观察发现得之。

误区-性能工作的必要性

- CPU速度是每两年翻一番，可是其它部件呢？
 - 硬盘的寻址时间受机械限制
 - 网络速度无法超越光速
- 随着硬件水平的不断提高，人们希望更强大、更智能、更漂亮的软件
- 硬件可以帮忙 – 但不能解决所有性能问题
- 操作系统并不是只运行一个软件（比如QQ），还要运行其他网游，其他Office软件。。。不要一个软件把性能耗光了
- 死循环，死锁，Sleep，卡死系统等是机器再好也会出问题

性能关系产品成败

- C
- V
- S



me OS

内容大纲

- 性能工作之必要性
- **性能与用户**
- 如何提高性能
- 性能监控
- 建立性能备忘体系
- 性能工具

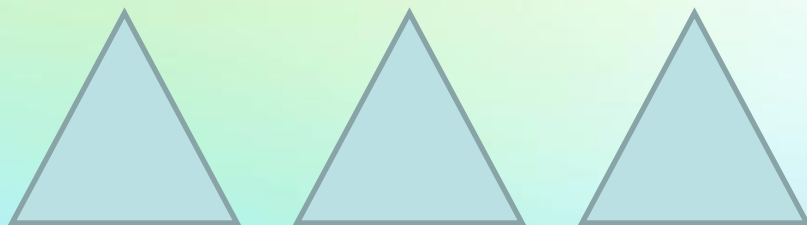
谁来评判性能？

- **用户**是最终的裁判
- 成功的关键是要达到用户的**期望**
- 用户的期望可能会改变
- 不同用户期望可能不同
- 性能应该是产品**必备**的特性



用户感知的三个重要的响应时间

- 0.1秒 反应迅速. 用户感觉控制不错
- 1秒 系统还工作. 但反应不够迅速
- 10秒 用户会觉得系统死掉了



性能工作的用户要领

- 性能问题的优先级和目标？

优先级取决于用户对该功能的使用频繁度，重要性。
目标取决于用户对于该功能的期望。

- 性能工作什么时候终止？

性能工作应该以达到用户期望为目标。而当目标已经达成，性能上的表现对用户来说将变得比较不关注。这个时候应该终止性能优化。否则，优化是无止境的，且从0.02ms到0.01ms的优化，用户根本就不Care。

内容大纲

- 性能工作之必要性
- 性能与用户
- **如何提高性能**
- 性能监控
- 建立性能备忘体系
- 性能工具

如何提高这些性能？

- 找出问题瓶颈，预估修改效果
- 通过性能优化或者性能策略修正问题
 - 1.从程序优化方面去优化性能
 - 2.从性能策略方面去优化性能

如何找出瓶颈？

性能工具法

API和关键路径监控法

性能日志法

1. 分解过程
2. 在各大步骤，分别加上Log
3. 通过Log数据，找出消耗分布情况
4. 找出消耗大头和消耗瓶颈，进行优化

这一步是解决性能问题的关键所在。如果找不到关键问题所在，绝对不要盲目优化！！！！

追查问题-注意事项

不要被表象数据所迷惑

设置控件大小非常耗时？

设置控件状态非常耗时？

分类	第一次打开耗时 (ms)	第二次打开耗时(ms)
CreateUIControlListBase	266	93
InitProfileFrame	204	78
InitPlugInFrame	391	172
InitPlugInFrame(put_state)	157	31
InitTopToolbarFrame	110	16
ShowAIO	219	63
合计	1190	422

追查问题-注意事项

规避测试本身对测试数据的影响

比如用Log来测试性能的话，要减少Log对数据造成的影响。

代码优化

通过提高代码本身的执行效率，来提高程序的执行效率，是最直接的优化方法。

代码优化指导1-硬盘IO

- 硬盘IO往往是瓶颈

**顺序读写速度远远大于
随机读写**

硬盘的速度只决定于其转速、缓存大小和平均寻道时间。主轴转速，目前市场上流行的是5400rpm(每分钟转数)和7200rpm的硬盘

数据读取时间 = 寻道时间 + 旋转时间 + 读写时间

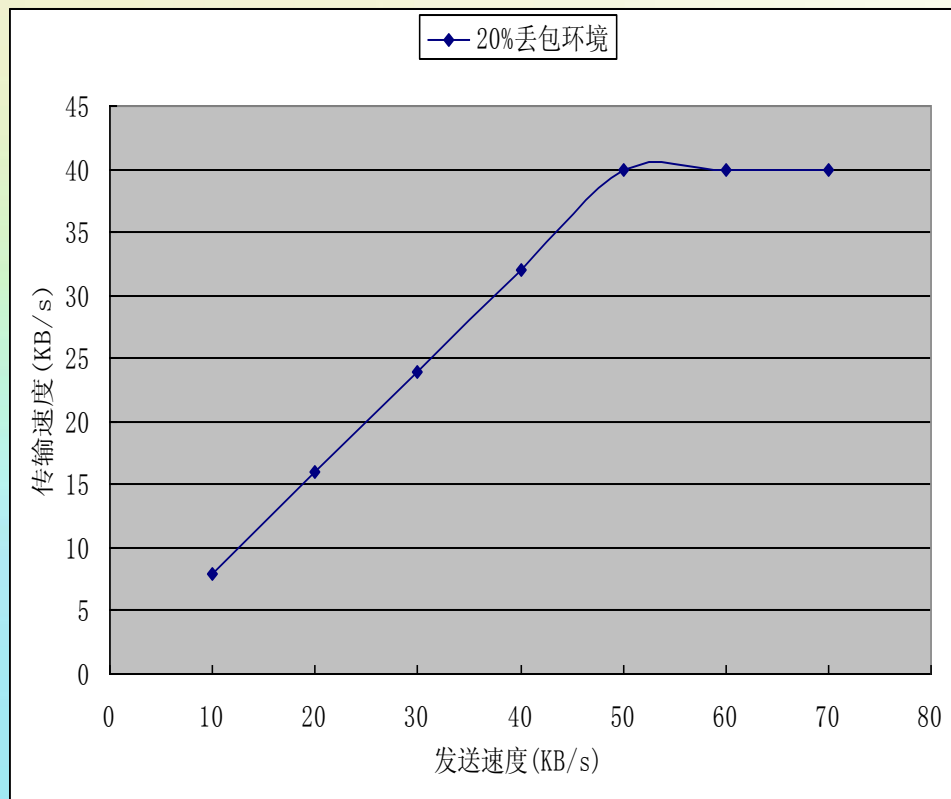
寻道时间一般占 60% - 70%

旋转时间一般占 20% - 30%

读取时间一般占 10% - 20%

代码优化指导2-网络IO

处理好网络IO将提高响应的速度



UDP传文件优化效果图

优化网络协议，减少网络往返次数，在每个包中可附带多方数据（当然要保证包的大小不能太大）

尽可能的利用当前网络环境以及让带宽饱和

代码优化指导3-去冗余

去冗余

名称	使用频率	大小
2007 Microsoft Office Suite Service ...	未知	未知
2007 Microsoft Office Suite Service ...	未知	未知
2007 Microsoft Office Suite Service ...	未知	未知
2007 Microsoft Office Suite Service ...	未知	未知
2007 Office system 兼容包	很少	63.7 MB
360安全卫士	很少	未知
7-Zip 4.43 alpha 2	有时	2.9 MB
ActivePerl 5.6.1 Build 631	很少	26.2 MB
Adobe ExtendScript Toolkit 2	很少	未知
Adobe Flash Player 10 ActiveX	未知	未知
Adobe Flash Player Plugin	未知	未知

冗余的逻辑是指,做了可以不用做的逻辑.
做多余的事意味着**浪费！！！！**

为什么2010正式版QQ挂机时，CPU在5%-7%（调用不必要的InvalidateRect，会引起绘制的冗余）

多余绘制区域的裁剪

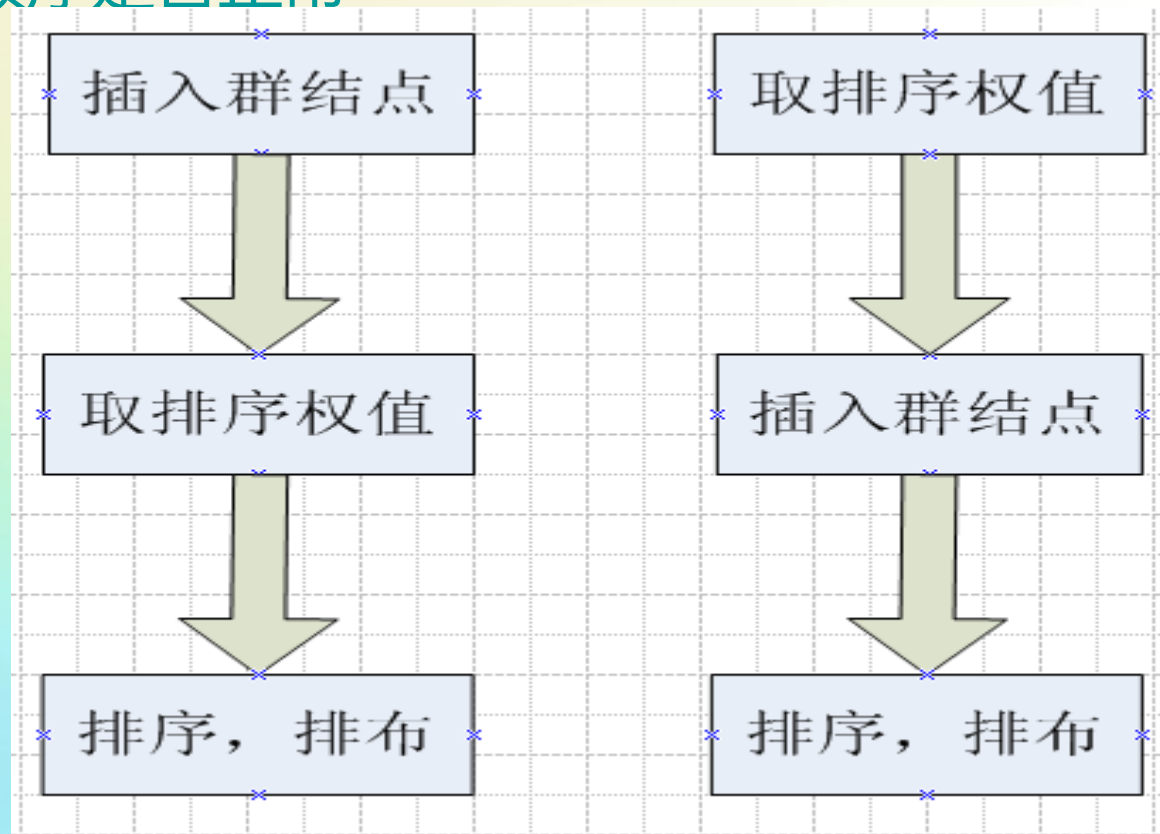
ListCtrl在做批量插入的时候，如不做SetRedaw(false),则每插入一个项都会重绘一遍

逻辑的精细化控制

调用频率，数据更新频率等控制

代码优化指导4-正确的调用顺序

调用顺序是否正常



排序算法做了优化，之前每次比较都会取一次权重值，现在是一性取得权重值并缓存起来，下次比较时直接用该值。↵

在 480 个群成员节点的情况下，**排序耗时从之前的 547ms 耗时下降为 62ms** 🍌 ↵

代码优化指导5-确保你的数据结构高效

当数据操作比较频繁时，选正确的数据结构将可能决定你代码的执行效率。

map和set的查找速度大于vector的速度。

stl容器都是基于分配拷贝原则的。

功能太强大的容器往往性能比较差一些。比如泛型数据结构的性能一般都远差于struct等数据结构。

TXData的泛滥使用，尽量避免数据结构之间的转换处理

代码优化指导6-CPU指令优化

- CPU指令集优化

MMX,SSE,SSE2,SSE3指令优化

MMX：8个64位MMX寄存器（mm0 - mm7），也可为各SSE扩展所使用；数据为整数，最多支持两个32位,运算中没有寄存器能够进行溢出指示;

SSE：8个128位xmm寄存器，MXSCR寄存器，EFLAGS寄存器支持单精度浮点,MXSCR含有rounding, overflow标志,支持64位SIMD整数

SSE2：执行环境同SSE ,双精度浮点,128位整数,双—单精度转换

代码优化指导6-CPU指令优化

我们的绘制函数中最核心的几个运算，主要包括缩放，alpha混合，BitBlt，以及关键色处理都使用了CPU指令集优化。理论上绘制的效率可以提高4倍，实测值也可以提高2-3倍。

```
pxor mm7, mm7 //mm7置0
movd mm6, 256 //mm6赋值为256
movd mm0, [esi] //读取源像素
movd mm1, [edi] //读取目标像素
punpcklbw mm1, mm7 //目标像素 00 00 00 00 AA RR GG BB 展开成 00 AA 00 RR 00 GG 00 BB
movd mm2, mm0 //将源像素的值赋给mm2
psrld mm2, 24 //按DWORD右移24位，mm2为00 00 00 00 00 00 00 AA
movq mm3, mm6 //mm3赋值为256
psubw mm3, mm2 //alpha = 256 - src.alpha
punpcklwd mm3, mm3 //展开为00 00 00 00 00 AA 00 AA
punpckldq mm3, mm3 //展开为00 AA 00 AA 00 AA 00 AA
pmullw mm1, mm3 //dest * alpha
psrlw mm1, 8
paddusb mm0, mm1 // (dest * (256 - src.alpha)) / 256 + src
packuswb mm0, mm7 //打包为00 00 00 00 AA RR GG BB
movd [edi] mm0
emms
```

MMX指令完成alpha混合代码

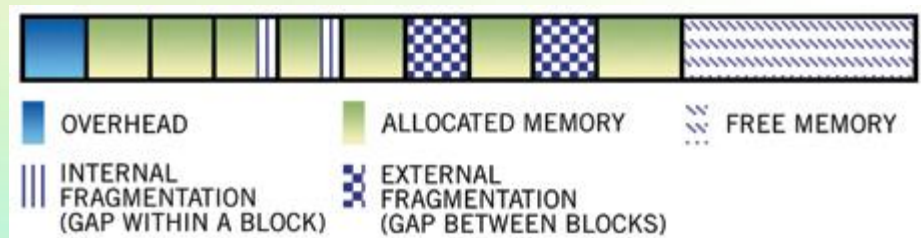
代码优化指导7-避免频繁分配内存

- 全局且频繁访问的数据用内存池
- 避免频繁分配内存

频繁分配内存可能引起内存碎片

频繁分配内存可能引起页面切换以及页面错误的产生

频繁分配内存本身就意味着你在不断做数据操作



- 如何避免

vector自身用*2分配原则避免频繁分配

避免过多CString的拷贝或者转换

避免过多的数据的拷贝

STL顺序容器在可预估大小情况下的预分配

为什么正式版2010在半透明下页面错误非常多？

代码优化指导8-更多的细节优化方法

结合实际情况，还有更多更细节的优化方法。

避免在临时文件夹目录操作文件

找出问题的程序段之后，查看该程序段使用的算法，结合实际情况优化具体算法。

减少dll数目，比如插件使用脚本等来完成。

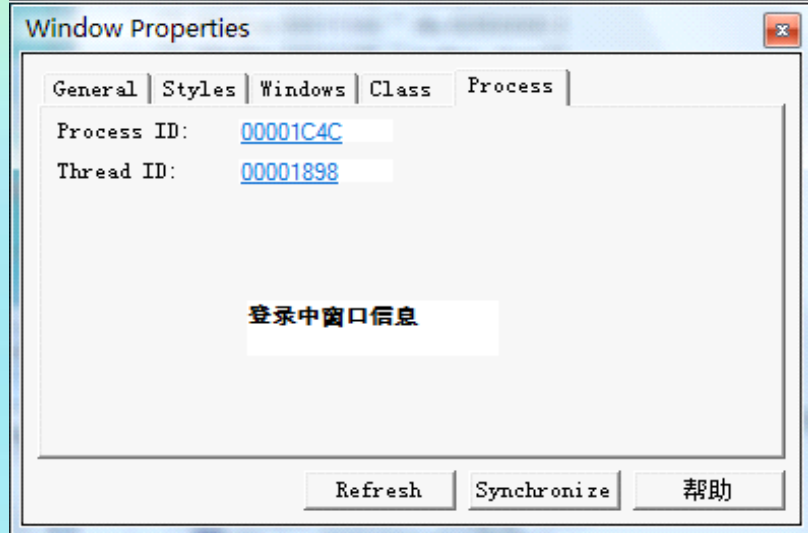
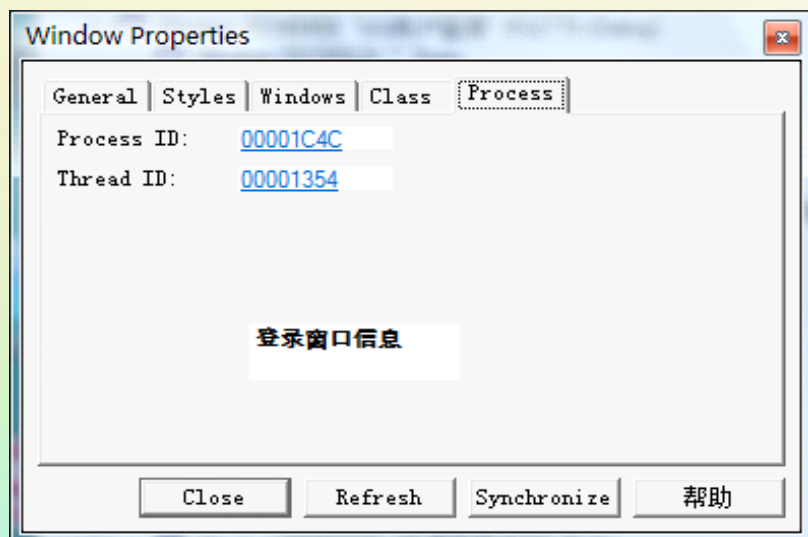
坚决杜绝死循环在代码中出现

优化策略

优化策略，不一定是从提高程序的实际执行速度，但是却能从其他方面去提高用户的性能体验。
同时有些优化方法是一定要结合优化策略来执行才能够进行的。

优化策略指导1-多线程多进程策略

登录中界面的多线程



Chrome的多线程模型和
多进程模型

QQ的跨进程IE/Flash

QQ计划尝试的多线程多
进程模型

优化策略指导2- 同步问题变成异步

- 同步动作变成异步

菜单关闭刷新

同步函数的使用导致的卡

如GetHostByName、
GetEnumWindow、
GetTrayPositon、
GetWindowText、主线程
Sleep、主线程
SendMessageTimeout等等



优化策略指导3-缓存策略

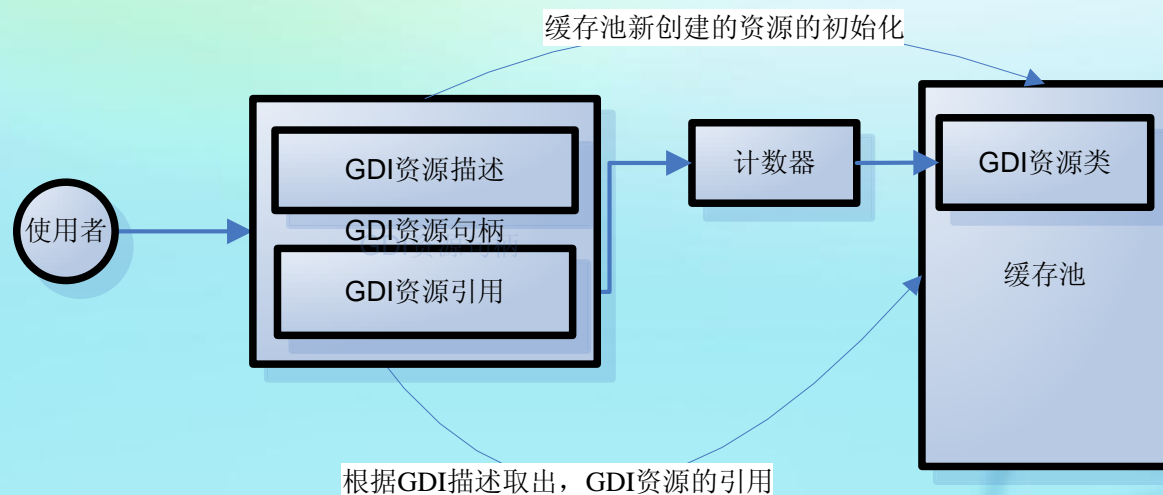
IO缓存策略：资源以及其他相关文件的缓存

绘制缓存：把一些绘制过程缓存成bitmap

淘汰策略：一般有先进先出，后进先出，末位淘汰，手动淘汰等。
这里主要做到提高命中率。

清理缓存：一定要找时机清理缓存,否则会有资源问题

Draw资源描述->Key

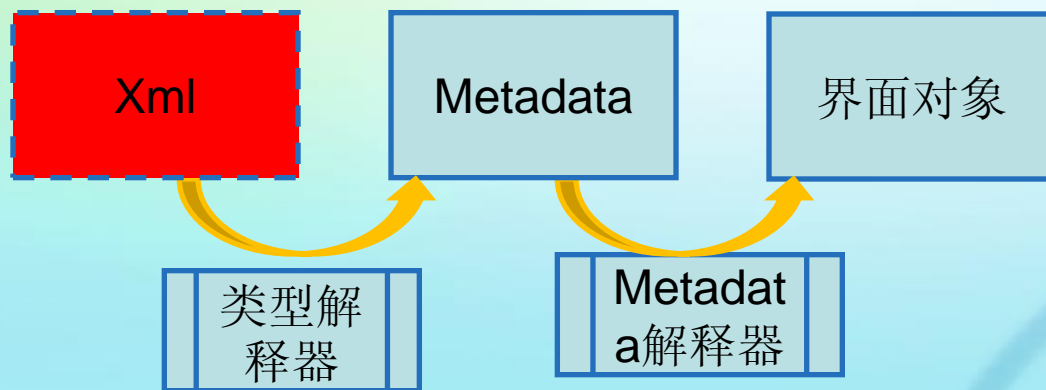


优化策略指导4-部分工作可预处理时做

把部分工作放到发布之前去做，这样就可以省掉这些工作的消耗

QQ会把配置的XML文件转换为MetaData 二进制文件保存到发布版本中去。

WPF也会把对应的XAML文件转换为BAML 文件（是XAML的二进制形式，被当作程序集的资源文件嵌入）保存到发布程序中去。



优化策略指导5-越频繁调用越要做原子优化

- 了解代码在运行栈中的位置是很重要的
 - 谁调用我？多么经常？在什么情况下？
 - 我会调用谁？多少次？是预料之中的吗？存在性能问题吗？

对于非常频繁的被人调用的块，要进行非常细致的原子优化

优化策略指导6-延迟或者提前加载处理

- 了解功能的常用度以及使用时机
 - 对于经常用 或者 肯定要用到的功能，可以做提前加载。
 - 对于很少使用的 或者 部分用户不会使用的功能，可以做延迟加载。

比如dll的懒加载，插件的懒启动，配置文件的提前加载。

公司同事如果到达5-6万人以后，RTX如何支撑？

- 1.分清主次和关键路径
- 2.OWnerData模式
- 3.懒加载 懒处理 懒创建

优化策略指导7-简化系统

简化系统

越复杂的系统，越可能出现的问题

不要写过于复杂的处理函数,过于复杂的处理过程可以分步骤走，过于复杂的函数可以分解成几个部分。

时间片优化方案的讨论。

红绿灯思想

优化策略指导8-视觉类性能问题

- 不要给用户在视觉上造成性能错觉
 - 以前登录完毕，插件一个一个的显示出来。
 - 以前打开一个聊天窗口，个签，QQShow,广告都是一个一个显示出来的。

一个一个显示出来，会给用户造成显示过程非常慢，而且也非常影响到视觉上的体验，不如全部准备完毕一起显示出来。

400ms-6s 体验原则 不断给用户反馈

举例讨论：老大给你安排一个复杂的任务，需要1年才能完成，你是一年后才去告诉你的老大说我完成了吗？

优化策略指导9-产品策略化解

产品策略

- 当确认性能无法提升，需要从产品角度来解决问题



互动环节

以前在工作或者学习过程中，有解决过性能问题吗？怎么发现和解决的？

内容大纲

- 性能工作之必要性
- 性能与用户
- 如何提高性能
- **性能监控**
- 建立性能备忘体系
- 性能工具

性能监控的缘由

问题：

一个QQ,四个中心都在增加代码，公司所有业务都在上面增加功能和需求，性能恶化速度怎么控制？怎么办？

第一时间知道性能变化！！

- 1.不会把问题拖到最后
- 2.刚引入的问题最容易查
- 3.问题拖到后期解决，修改成本非常高。

性能监控做法

- 重点路径的监控
 - 所有的消耗源头
 - 重要业务功能点
 - 以前出现过性能问题的点
- 定期自动运行，发布前对比关键监控点的性能
- 思考？
QQ应该要监控哪些地方

性能运营监控

- 重点路径Imoss系统数据
- 投诉运营

内容大纲

- 性能工作之必要性
- 性能与用户
- 如何提高性能
- 性能监控
- **建立性能备忘体系**
- 性能工具

全面的性能保障体系

建立完善的用户
反馈体系

确保测试人员将性能
测试作为常项工作

每个开发员在写代码
的时候都要想到性能

每一个功能组都将性
能看作自己的责任

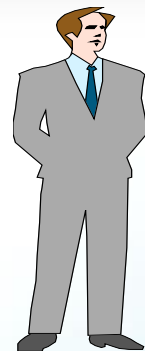
确保架构将性能的各
个方面都加以考虑

确保在提出需求的时候
就开始考虑性能特征

思考？
有前面5步，是完善的保障体系吗？

产品经理检查清单

- 1 PM必须收集可能影响性能的各个方面
- 2 考虑到使用上的不同特性
- 3 要考虑到数据量和并发用户相关的特性



开发人员检查清单

- 选择正确的设计能够处理不同情况下的使用特性，数据量，并发用户.只有有好的性能的架构才是好的架构！



测试人员检查清单

- 测试是确保最终产品是否符合用户期望的最后手段
- 当代码做改动后，很难预测对整体性能的影响
- 因此需要：
 - 创建子系统和场景测试
 - 用接近现实的数据来测试
 - 用合理的并行人数来测试
 - 用接近现实的环境来测试

请问：什么是接近真实的环境和数据？



完善的用户反馈体系

- 1.用性能工具采集用户投诉现场分析处理
- 2.数据上报
- 3.性能调查体系

举例：

插件的一些原则：比如微博插件
漫游皮肤，多态登录，多帐号登录等新需求
个人资料改版等问题
QQ秀引起CPU高，热词搜索引起死循环等
性能架构的一些优化

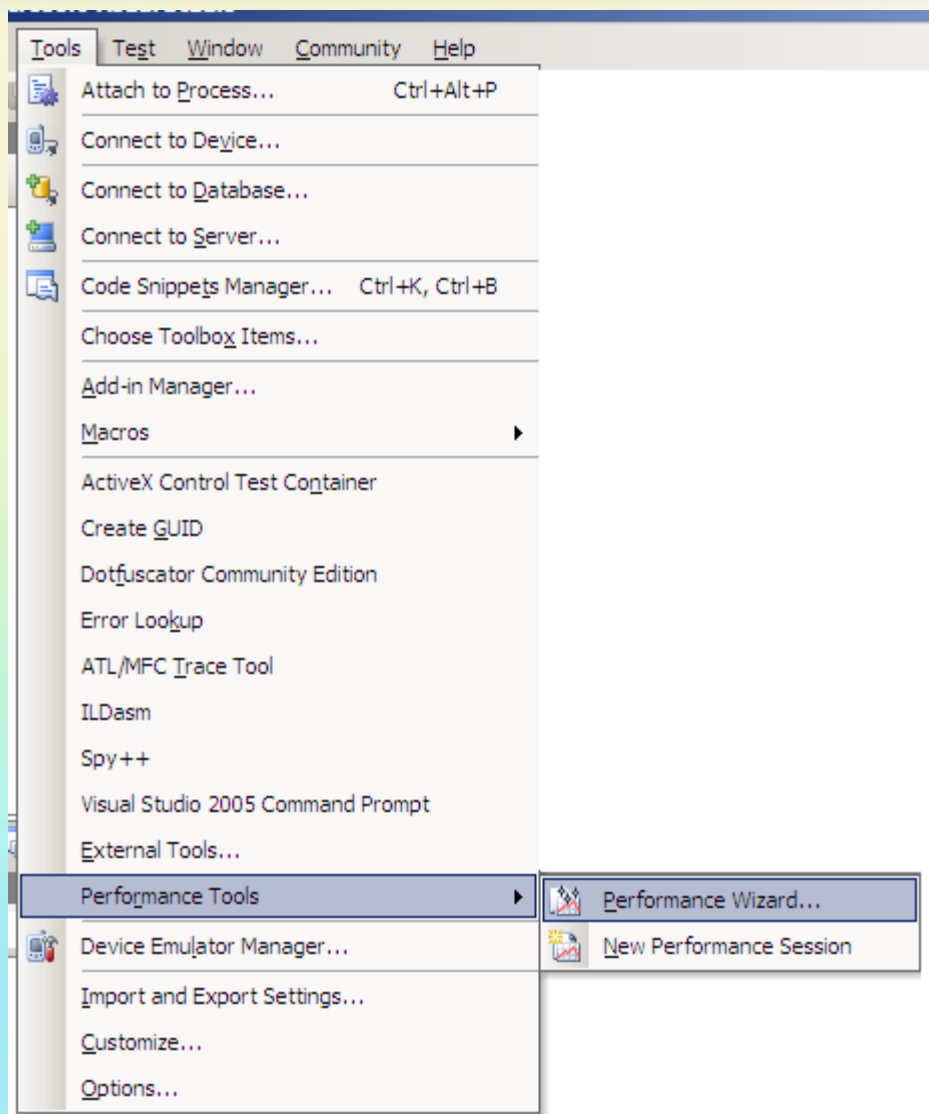
内容大纲

- 性能工作之必要性
- 如何提高性能
- 性能监控
- 建立性能备忘体系
- 性能与用户
- **性能工具**

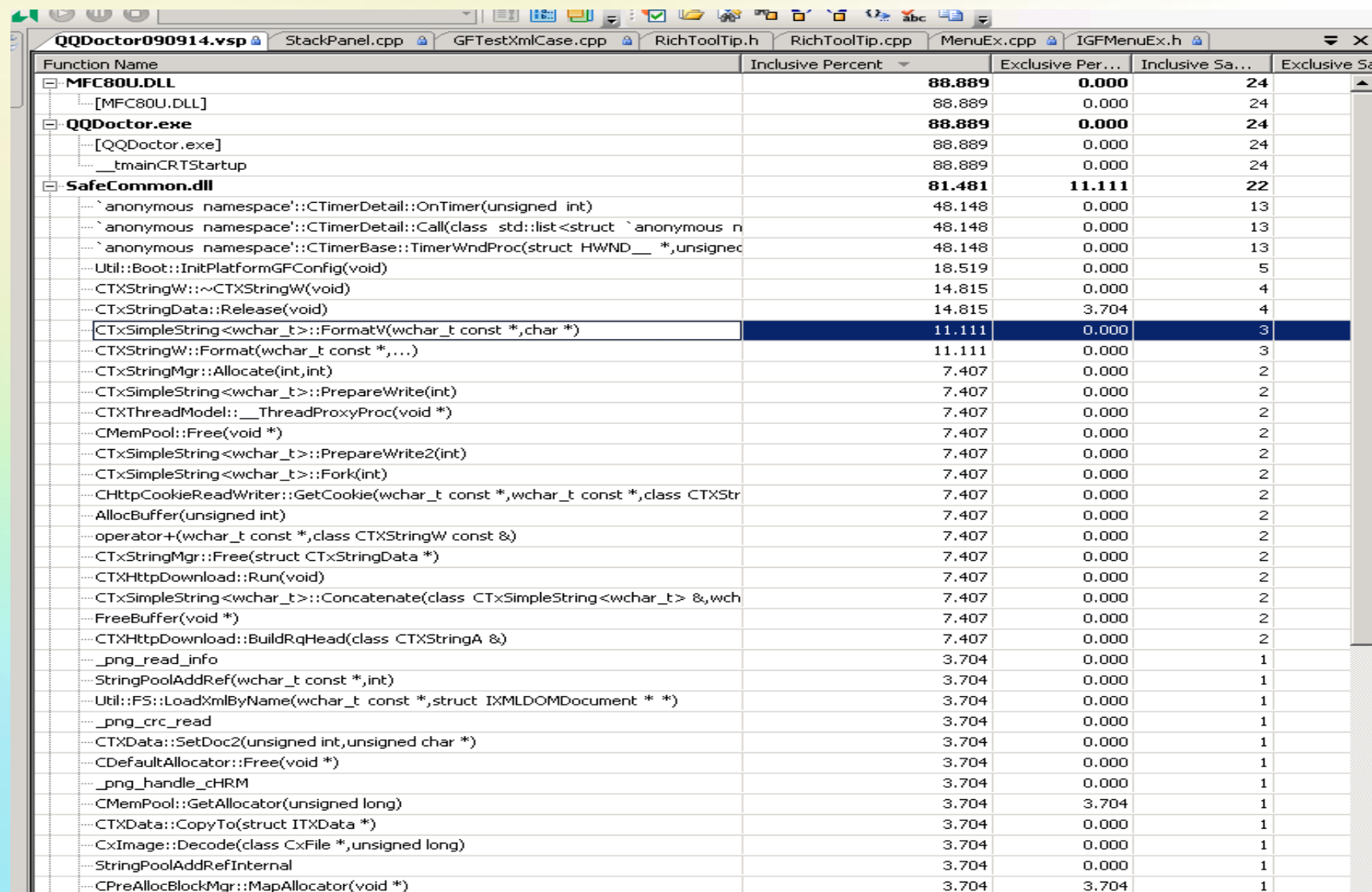
工具

- Visual Studio 2005
- Compuware
Devpartner Studio
 - BoundsChecker
- IBM PurifyPlus
- AutomatedQA
- Telelogic Logiscope
- AppVerifier
- DriverVerifier
- Perfmon
- NetMon
- UMDH
- 性能定位工具（自研）
- 内存，GDI泄露检查工具

VS2005-Performance Tools



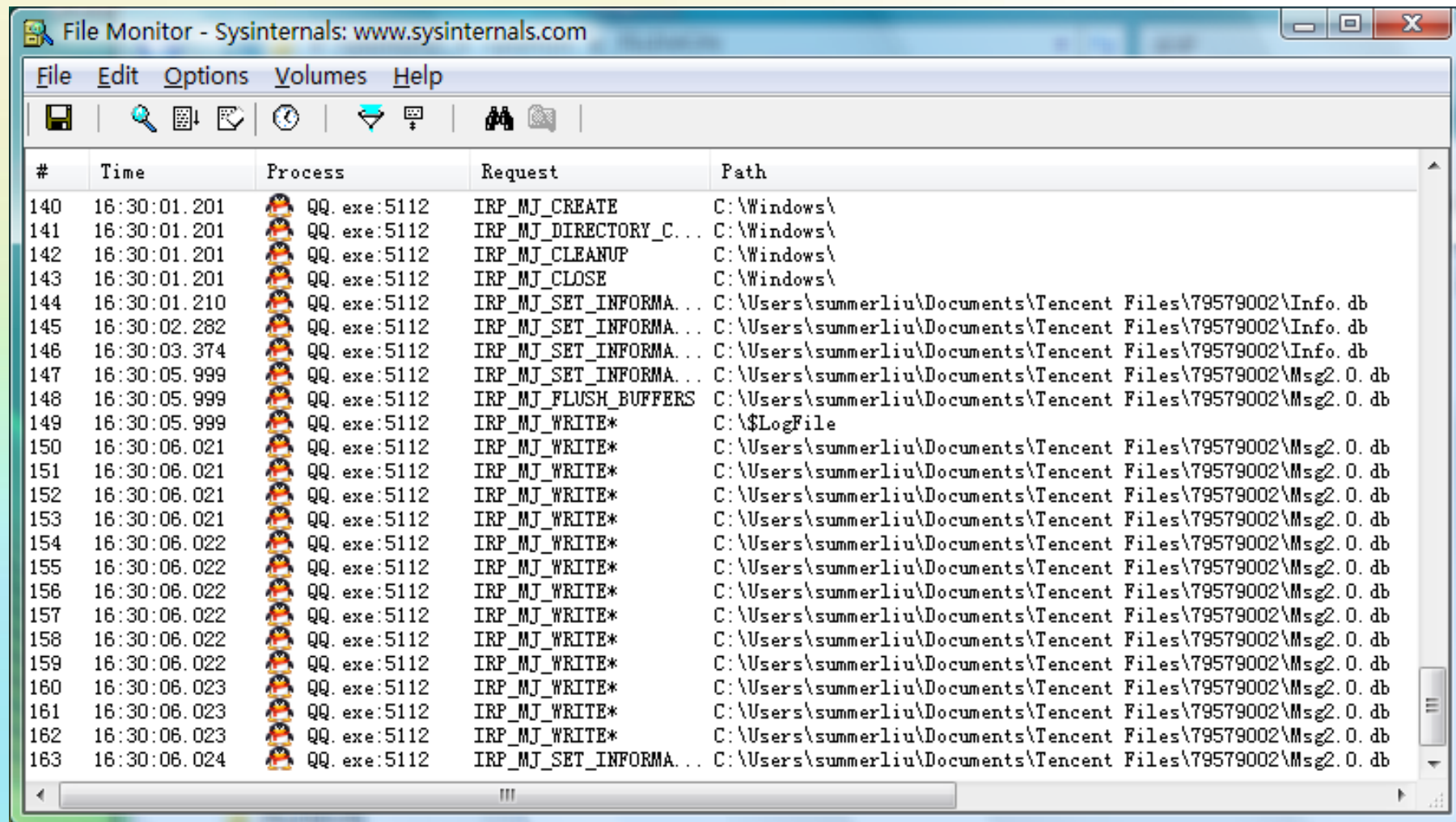
VS2005-PT查性能问题



Function Name	Inclusive Percent	Exclusive Per...	Inclusive Sa...	Exclusive Sa...
MFC80U.DLL	88.889	0.000	24	
[MFC80U.DLL]	88.889	0.000	24	
QQDoctor.exe	88.889	0.000	24	
[QQDoctor.exe]	88.889	0.000	24	
__tmainCRTStartup	88.889	0.000	24	
SafeCommon.dll	81.481	11.111	22	
anonymous namespace::CTimerDetail::OnTimer(unsigned int)	48.148	0.000	13	
anonymous namespace::CTimerDetail::Call(class std::list<struct anonymous namespace::CTimerDetail::TimerWndProc> &)	48.148	0.000	13	
anonymous namespace::CTimerBase::TimerWndProc(struct HWND__ *, unsigned int, unsigned int)	48.148	0.000	13	
Util::Boot::InitPlatformGFCConfig(void)	18.519	0.000	5	
CTXStringW::~~CTXStringW(void)	14.815	0.000	4	
CTxStringData::Release(void)	14.815	3.704	4	
CTxSimpleString<wchar_t>::FormatV(wchar_t const *, char *)	11.111	0.000	3	
CTXStringW::Format(wchar_t const *, ...)	11.111	0.000	3	
CTxStringMgr::Allocate(int, int)	7.407	0.000	2	
CTxSimpleString<wchar_t>::PrepareWrite(int)	7.407	0.000	2	
CTXThreadModel::__ThreadProxyProc(void *)	7.407	0.000	2	
CMemPool::Free(void *)	7.407	0.000	2	
CTxSimpleString<wchar_t>::PrepareWrite2(int)	7.407	0.000	2	
CTxSimpleString<wchar_t>::Fork(int)	7.407	0.000	2	
CHttpCookieReadWriter::GetCookie(wchar_t const *, wchar_t const *, class CTXStringW &)	7.407	0.000	2	
AllocBuffer(unsigned int)	7.407	0.000	2	
operator+(wchar_t const *, class CTXStringW const &)	7.407	0.000	2	
CTxStringMgr::Free(struct CTxStringData *)	7.407	0.000	2	
CTXHttpDownload::Run(void)	7.407	0.000	2	
CTxSimpleString<wchar_t>::Concatenate(class CTxSimpleString<wchar_t> &, wchar_t const *)	7.407	0.000	2	
FreeBuffer(void *)	7.407	0.000	2	
CTXHttpDownload::BuildRqHead(class CTXStringA &)	7.407	0.000	2	
_png_read_info	3.704	0.000	1	
StringPoolAddRef(wchar_t const *, int)	3.704	0.000	1	
Util::FS::LoadXmlByName(wchar_t const *, struct IXMLDOMDocument *)	3.704	0.000	1	
_png_crc_read	3.704	0.000	1	
CTXData::SetDoc2(unsigned int, unsigned char *)	3.704	0.000	1	
CDefaultAllocator::Free(void *)	3.704	0.000	1	
_png_handle_cHRM	3.704	0.000	1	
CMemPool::GetAllocator(unsigned long)	3.704	3.704	1	
CTXData::CopyTo(struct ITXData *)	3.704	0.000	1	
CxImage::Decode(class CxFile *, unsigned long)	3.704	0.000	1	
StringPoolAddRefInternal	3.704	0.000	1	
CPreAllocBlockMgr::MapAllocator(void *)	3.704	3.704	1	

Perf Tool查QQ医生启动过程

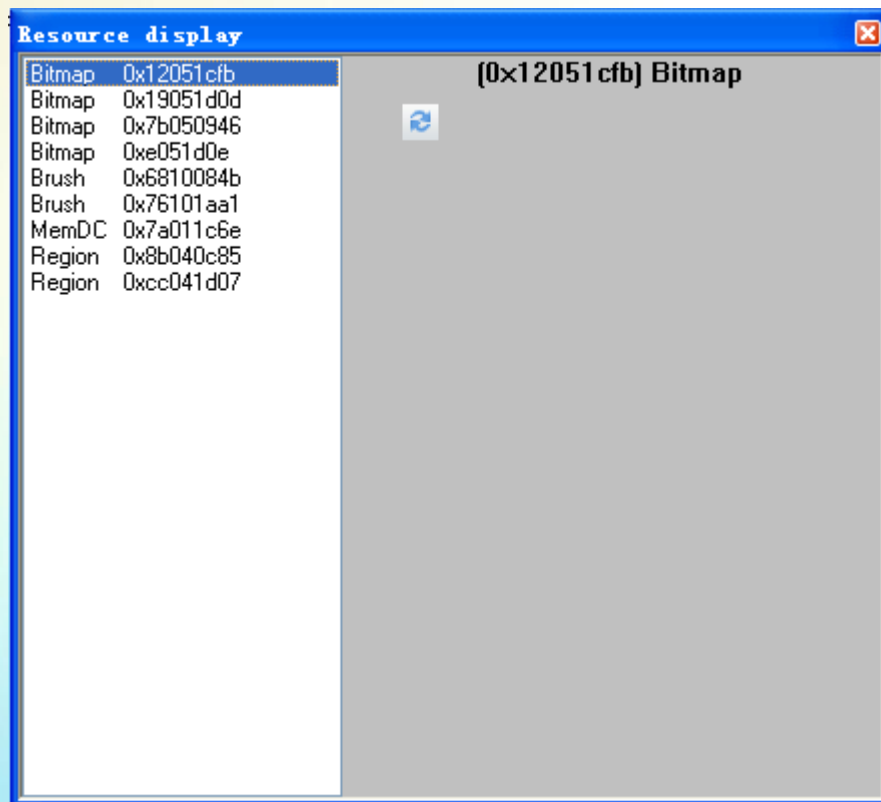
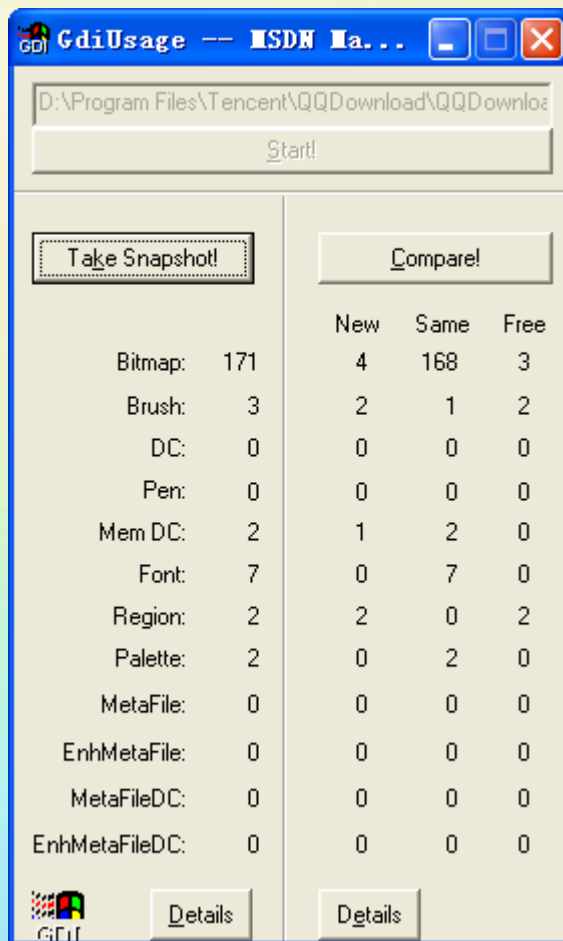
monitor-监控文件IO情况



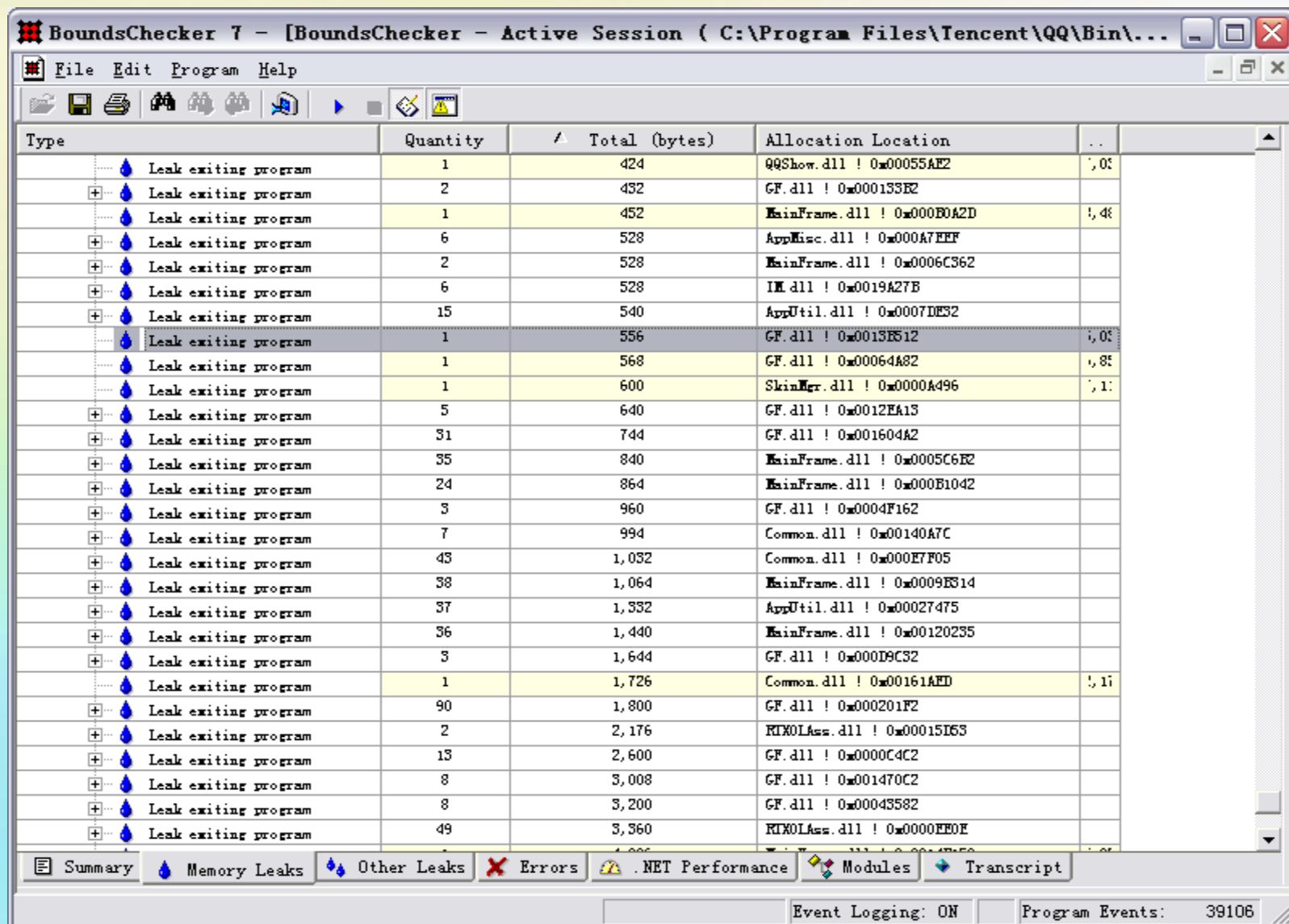
The screenshot shows the File Monitor application window. The title bar reads "File Monitor - Sysinternals: www.sysinternals.com". The menu bar includes "File", "Edit", "Options", "Volumes", and "Help". The toolbar contains icons for saving, searching, printing, pausing, and other functions. The main display area is a table with the following columns: "#", "Time", "Process", "Request", and "Path". The table lists 14 file I/O operations performed by the QQ.exe process (PID 5112) between 16:30:01.201 and 16:30:06.024. The requests include IRP_MJ_CREATE, IRP_MJ_DIRECTORY_C..., IRP_MJ_CLEANUP, IRP_MJ_CLOSE, IRP_MJ_SET_INFORMA..., IRP_MJ_FLUSH_BUFFERS, and IRP_MJ_WRITE*.

#	Time	Process	Request	Path
140	16:30:01.201	QQ.exe:5112	IRP_MJ_CREATE	C:\Windows\
141	16:30:01.201	QQ.exe:5112	IRP_MJ_DIRECTORY_C...	C:\Windows\
142	16:30:01.201	QQ.exe:5112	IRP_MJ_CLEANUP	C:\Windows\
143	16:30:01.201	QQ.exe:5112	IRP_MJ_CLOSE	C:\Windows\
144	16:30:01.210	QQ.exe:5112	IRP_MJ_SET_INFORMA...	C:\Users\summerliu\Documents\Tencent Files\79579002\Info.db
145	16:30:02.282	QQ.exe:5112	IRP_MJ_SET_INFORMA...	C:\Users\summerliu\Documents\Tencent Files\79579002\Info.db
146	16:30:03.374	QQ.exe:5112	IRP_MJ_SET_INFORMA...	C:\Users\summerliu\Documents\Tencent Files\79579002\Info.db
147	16:30:05.999	QQ.exe:5112	IRP_MJ_SET_INFORMA...	C:\Users\summerliu\Documents\Tencent Files\79579002\Msg2.0.db
148	16:30:05.999	QQ.exe:5112	IRP_MJ_FLUSH_BUFFERS	C:\Users\summerliu\Documents\Tencent Files\79579002\Msg2.0.db
149	16:30:05.999	QQ.exe:5112	IRP_MJ_WRITE*	C:\\$LogFile
150	16:30:06.021	QQ.exe:5112	IRP_MJ_WRITE*	C:\Users\summerliu\Documents\Tencent Files\79579002\Msg2.0.db
151	16:30:06.021	QQ.exe:5112	IRP_MJ_WRITE*	C:\Users\summerliu\Documents\Tencent Files\79579002\Msg2.0.db
152	16:30:06.021	QQ.exe:5112	IRP_MJ_WRITE*	C:\Users\summerliu\Documents\Tencent Files\79579002\Msg2.0.db
153	16:30:06.021	QQ.exe:5112	IRP_MJ_WRITE*	C:\Users\summerliu\Documents\Tencent Files\79579002\Msg2.0.db
154	16:30:06.022	QQ.exe:5112	IRP_MJ_WRITE*	C:\Users\summerliu\Documents\Tencent Files\79579002\Msg2.0.db
155	16:30:06.022	QQ.exe:5112	IRP_MJ_WRITE*	C:\Users\summerliu\Documents\Tencent Files\79579002\Msg2.0.db
156	16:30:06.022	QQ.exe:5112	IRP_MJ_WRITE*	C:\Users\summerliu\Documents\Tencent Files\79579002\Msg2.0.db
157	16:30:06.022	QQ.exe:5112	IRP_MJ_WRITE*	C:\Users\summerliu\Documents\Tencent Files\79579002\Msg2.0.db
158	16:30:06.022	QQ.exe:5112	IRP_MJ_WRITE*	C:\Users\summerliu\Documents\Tencent Files\79579002\Msg2.0.db
159	16:30:06.022	QQ.exe:5112	IRP_MJ_WRITE*	C:\Users\summerliu\Documents\Tencent Files\79579002\Msg2.0.db
160	16:30:06.023	QQ.exe:5112	IRP_MJ_WRITE*	C:\Users\summerliu\Documents\Tencent Files\79579002\Msg2.0.db
161	16:30:06.023	QQ.exe:5112	IRP_MJ_WRITE*	C:\Users\summerliu\Documents\Tencent Files\79579002\Msg2.0.db
162	16:30:06.023	QQ.exe:5112	IRP_MJ_WRITE*	C:\Users\summerliu\Documents\Tencent Files\79579002\Msg2.0.db
163	16:30:06.024	QQ.exe:5112	IRP_MJ_SET_INFORMA...	C:\Users\summerliu\Documents\Tencent Files\79579002\Msg2.0.db

GdiUsage-GDI泄露检查



BoundsChecker-内存泄露检查



The screenshot shows the BoundsChecker 7 application window. The title bar reads "BoundsChecker 7 - [BoundsChecker - Active Session (C:\Program Files\Tencent\QQ\Bin\...)". The menu bar includes "File", "Edit", "Program", and "Help". The toolbar contains icons for file operations and execution. The main window displays a table of memory leaks.

Type	Quantity	Total (bytes)	Allocation Location
Leak exiting program	1	424	QQShow.dll ! 0x00055AE2
Leak exiting program	2	432	GF.dll ! 0x000133E2
Leak exiting program	1	452	MainFrame.dll ! 0x000B0A2D
Leak exiting program	6	528	AppMisc.dll ! 0x000A7FEF
Leak exiting program	2	528	MainFrame.dll ! 0x0006C362
Leak exiting program	6	528	IM.dll ! 0x0019A27B
Leak exiting program	15	540	AppUtil.dll ! 0x0007DE32
Leak exiting program	1	556	GF.dll ! 0x0013E512
Leak exiting program	1	568	GF.dll ! 0x00064A82
Leak exiting program	1	600	SkinMgr.dll ! 0x0000A496
Leak exiting program	5	640	GF.dll ! 0x0012FA13
Leak exiting program	31	744	GF.dll ! 0x001604A2
Leak exiting program	35	840	MainFrame.dll ! 0x0005C6E2
Leak exiting program	24	864	MainFrame.dll ! 0x000B1042
Leak exiting program	3	960	GF.dll ! 0x0004F162
Leak exiting program	7	994	Common.dll ! 0x00140A7C
Leak exiting program	43	1,032	Common.dll ! 0x000E7F05
Leak exiting program	38	1,064	MainFrame.dll ! 0x0009E514
Leak exiting program	37	1,332	AppUtil.dll ! 0x00027475
Leak exiting program	36	1,440	MainFrame.dll ! 0x00120235
Leak exiting program	3	1,644	GF.dll ! 0x000D9C32
Leak exiting program	1	1,726	Common.dll ! 0x00161AED
Leak exiting program	90	1,800	GF.dll ! 0x000201F2
Leak exiting program	2	2,176	RIMOLAss.dll ! 0x00015D53
Leak exiting program	13	2,600	GF.dll ! 0x0000C4C2
Leak exiting program	8	3,008	GF.dll ! 0x001470C2
Leak exiting program	8	3,200	GF.dll ! 0x00043582
Leak exiting program	49	3,360	RIMOLAss.dll ! 0x0000EE0E

The bottom of the window features a tabbed interface with "Summary", "Memory Leaks", "Other Leaks", "Errors", ".NET Performance", "Modules", and "Transcript". The "Memory Leaks" tab is currently selected. At the bottom right, there are checkboxes for "Event Logging: ON" and "Program Events: 39106".

自研工具

• 既然有如此强大的性能工具阵营，为什么还需要自己做工具？

用业界工具去追查我们的问题的时候发现：

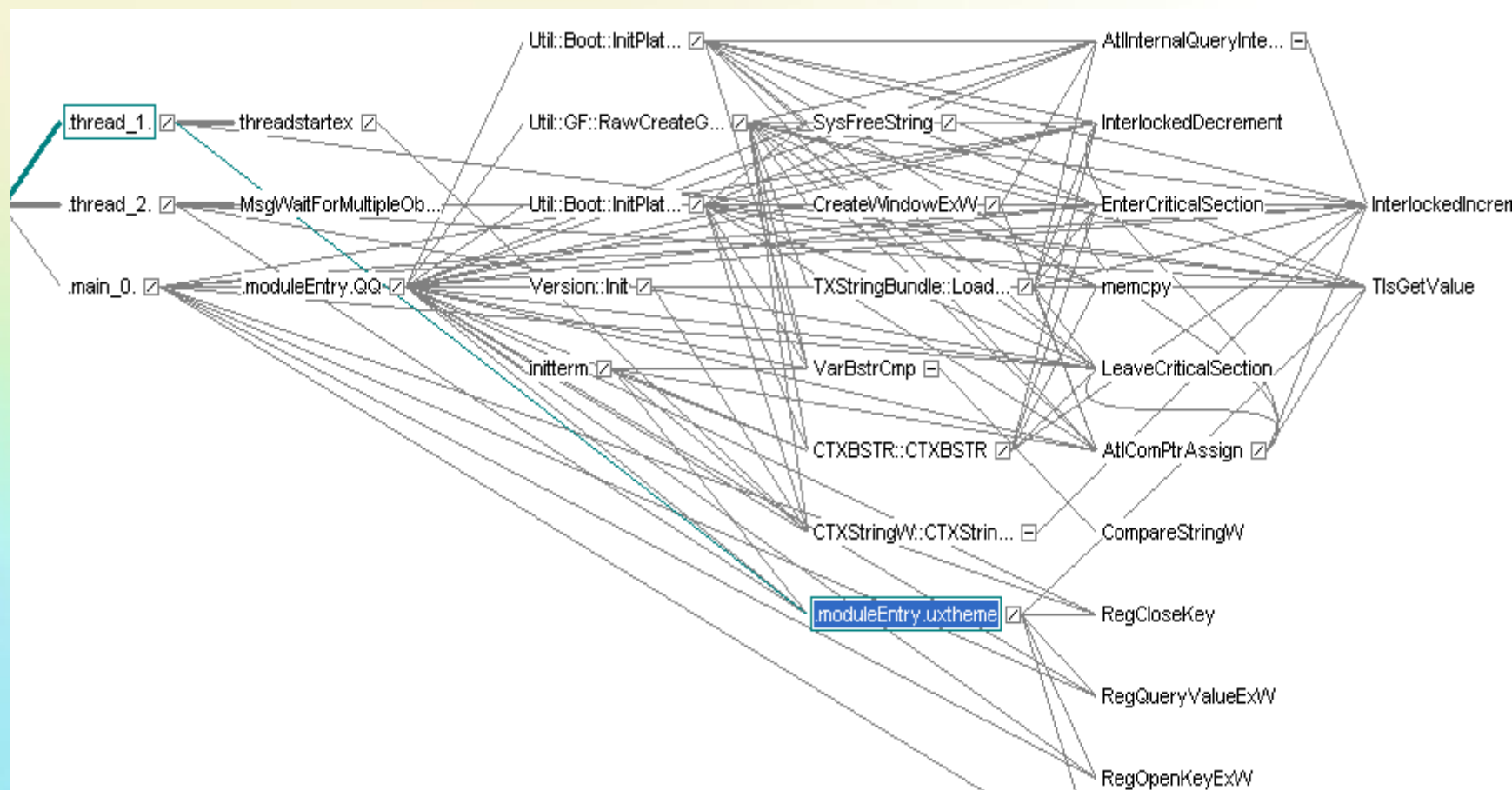
1.1 整个追查问题的过程QQ会变得非常慢，登录都要十几分钟。

1.2 分析出来的结果关系网都非常复杂。

1.3 他们均会把最后的很多消耗直接归结到底层。

1.4 他们都只能发现很面上的东西，而不能发现点上的东西，故不能快速的定位到问题所在。

复杂的关系网



Rational Quantify查QQ出登陆框过程

消耗简单归结到底层

QQ医生现在启动的时候，初始化GF/common 模块和创建第一个GF窗口耗费了70%的时间。

+ [WS2_32.dll]	2.381	0.000
+ __endthreadex	4.762	0.000
- __tmainCRTStartup	92.857	0.000
- AfxWinMain(struct HINSTANCE__ *, struct HINSTANCE__ *, wchar_t *, int)	92.857	0.000
- CLightDogApp::InitInstance(void)	92.857	0.000
+ _AfxSocketInit(struct WSADATA *)	2.381	0.000
- CLightDogApp::ShowMainDlg(void)	90.476	0.000
- CLgMainDlg::ShowMainDlg(int)	26.190	0.000
+ [SafeGF.dll]	14.286	0.000
+ Util::GF::CreateGFObjByHtmlEx<struct IGFStandardWin>(wchar_t *, struct IGFStandardWin	11.905	0.000
+ CLightDogApp::MessageLoop(void)	11.905	0.000
+ CLightDogApp::StartSelfUpdate(void)	28.571	0.000
- CTXIMSBASE::Init(void)	23.810	0.000
+ [SafeCommon.dll]	23.810	0.000

VS的Perf Tool查QQ医生启动问题

并不能把问题定位到比较细的可修改的点

SafeCommon.dll	81.481	11.111	22
... \anonymous namespace\::CTimerDetail::OnTimer(unsigned int)	48.148	0.000	13
... \anonymous namespace\::CTimerDetail::Call(class std::list<struct `anonymous n	48.148	0.000	13
... \anonymous namespace\::CTimerBase::TimerWndProc(struct HWND__ *,unsigned	48.148	0.000	13
... Util::Boot::InitPlatformGFCConfig(void)	18.519	0.000	5
... CTXStringW::~~CTXStringW(void)	14.815	0.000	4
... CTXStringData::Release(void)	14.815	3.704	4
... CTXSimpleString<wchar_t>::FormatV(wchar_t const *,char *)	11.111	0.000	3
... CTXStringW::Format(wchar_t const *,...)	11.111	0.000	3
... CTXStringMgr::Allocate(int,int)	7.407	0.000	2
... CTXSimpleString<wchar_t>::PrepareWrite(int)	7.407	0.000	2
... CTXThreadModel::__ThreadProxyProc(void *)	7.407	0.000	2
... CMemPool::Free(void *)	7.407	0.000	2
... CTXSimpleString<wchar_t>::PrepareWrite2(int)	7.407	0.000	2

Ontimer占了60%,请问我该如何去修改？

- 1.知道是哪个Timmer ID吗？
- 2.知道Timmer响应的是哪个类的哪个函数吗？

特色1-精确定位，直接指导修改

项名	时...	百...	次数
[-] PerfStat	3734	100%	1
[-] Thread (2768)	3734	100%	1
[-] GFMsg-HandleOnWinMsg:15	3624	97%	19
[-] GFMsg-DispatchFrameMsg	3560	98%	4728
[-] GFM_PaintBkg	3522	98%	1182
[-] GFDraw-PaintInternal	3506	99%	257
[-] platformres:crm\mainpanel\tree_background.bmp	3270	93%	57
[-] platformres:crm\mainpanel\mainboard_bkg.bmp	117	3%	19
[-] platformres:crm\mainpanel\fastreply\background.bmp	63	1%	19
[+] GFM_Paint	23	0%	1182
[+] __Unknown	63	1%	1
[+] GFMsg-DispatchFrameMsg	90	2%	1503
[+] IsNetConnectionOK	14	0%	7
[+] CTKCSProcessor::OnRecvData	0	0%	1

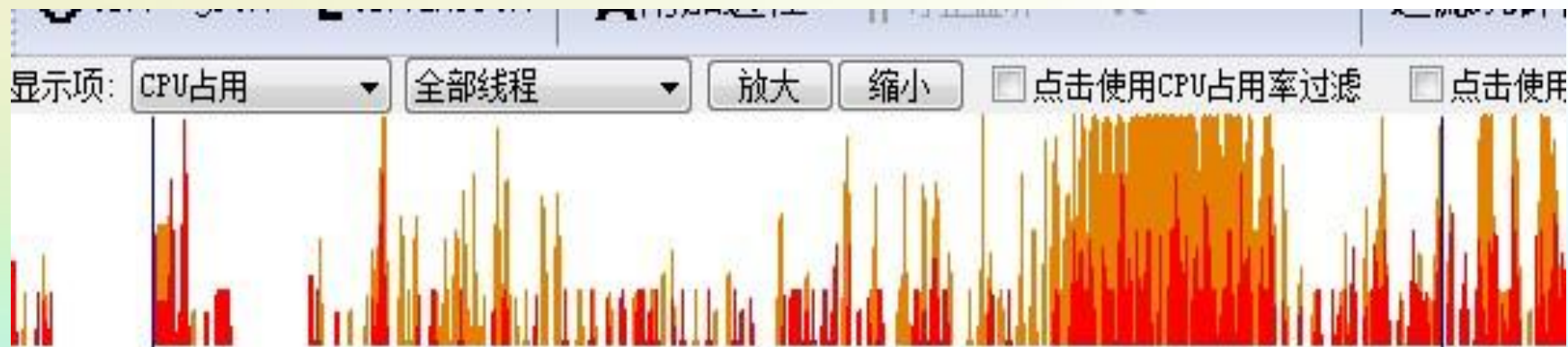
自研工具追查CRM CPU占用高问题

特色2-统计挖掘，直接得出有效数据

统计模型			
Data	27	59%	15
Xml	13	28%	6
LoadXml	12	91%	3
Xml-LoadXml	12	100%	2
C:\DOCUME~1\airywu\LOCALS~1\Temp\SBC`%__D28DIQV7YWZ918		71%	1
systeme:Config\Theme.xml	3	28%	1
LoadXmlByName	1	8%	4
Xml-LoadXmlByName	1	100%	3
platform:TypeDef\BasicTypeDef.xml	0	50%	1
platform:TypeDef\ExtraTypeDef.xml	0	26%	1
platform:TypeDef\IMSTypeDef.xml	0	22%	1
Api	6	14%	15
Dll	5	76%	6
File	1	23%	17
CreateFileW	1	87%	11
Api-File_CreateFileW	1	100%	6
C:\DOCUME~1\airywu\LOCALS~1\Temp\SBC`%__D28DIQV7YW0		55%	2
C:\WINDOWS\Registration\R0000000000012.clb	0	24%	1
F:\测试\QQ医生测试版本\QQ医生测试版本\Release\Type0		8%	1
F:\测试\QQ医生测试版本\QQ医生测试版本\Release\Type0		6%	1
F:\测试\QQ医生测试版本\QQ医生测试版本\Release\Type0		5%	1
WriteFile	0	6%	2
ReadFile	0	5%	4
FS	0	1%	11
GFCreat	0	0%	2

你想知道 整段时间的绘制消耗（排除IO）吗？

特色3-分析区段过滤

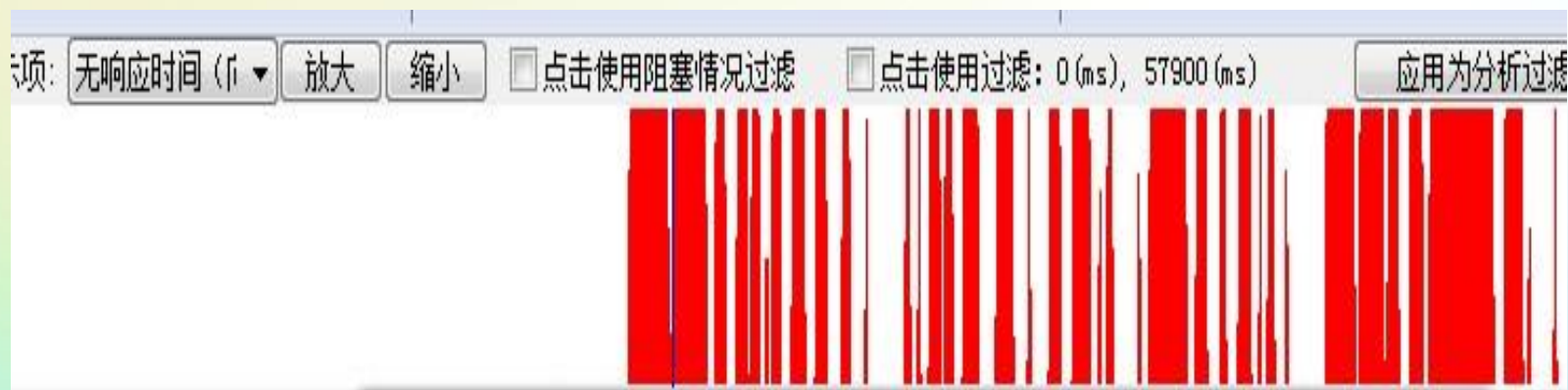


你想知道 CPU 占用超过50%的地方在做什么吗？

特色4-丰富的堆栈信息，方便查未知问题

任何时段任何线程的发生了诡异的事情，都能得到堆栈的使用情况

特色5-丰富的现场信息，直接解决关键问题



你想知道用户是什么时段卡住了么？

你想知道用户采集现场的时段 CPU 占用情况么？

你想知道用户采集现场的时段内存的占用情况么？

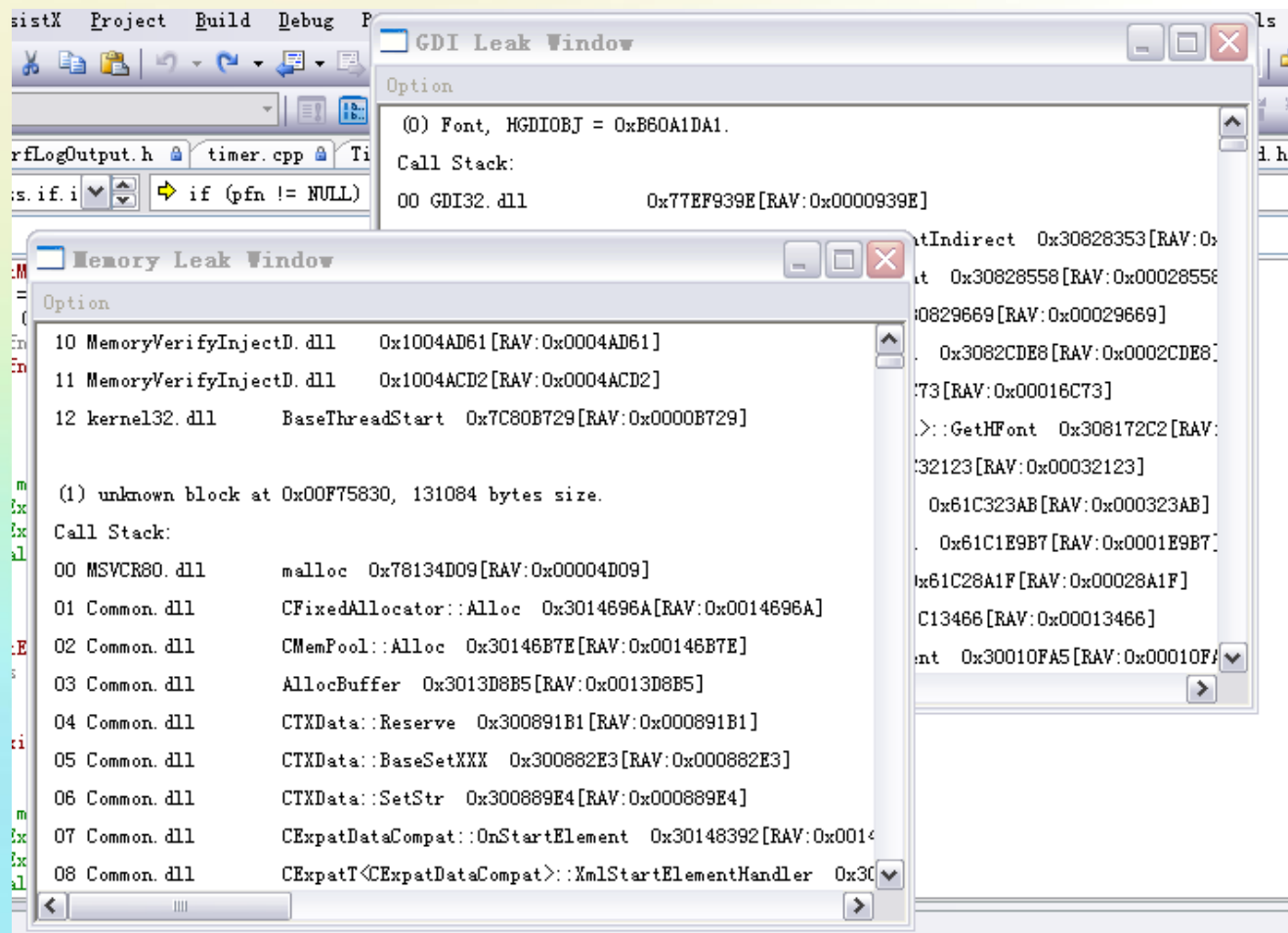
你想知道用户采集现场的时段的IO情况么？

自研-性能定位工具的基本原理

Windows客户端软件，都是基于**消息驱动**的。通过**覆盖所有的消耗源头**，在所有的消耗源加上比较详细的附带信息。包括耗时API。

然后把所有消耗点的数据，整理成一个数据库集的形式，通过对Log库分析统计挖掘，同时提取关键信息，得出整个过程的消耗占用情况以及相关关键信息。

自研-内存和GDI泄露检查工具



自研-泄露检查工具原理

通过API Hook,监管所有的GDI操作，以及内存分配释放操作，以及当时的CallBack信息，最后通过统计结合pdb信息，即可以打印出GDI以及内存有泄露的地方，同时还可以跳转到泄露对应的代码行上去。

建议与观点

如何让性能走得更好？您有更好的高招么？



谢谢！