ENTERPRISE ARCHITECT

用户指南系列

剖析

Author: Sparx Systems

Date: 20/06/2023

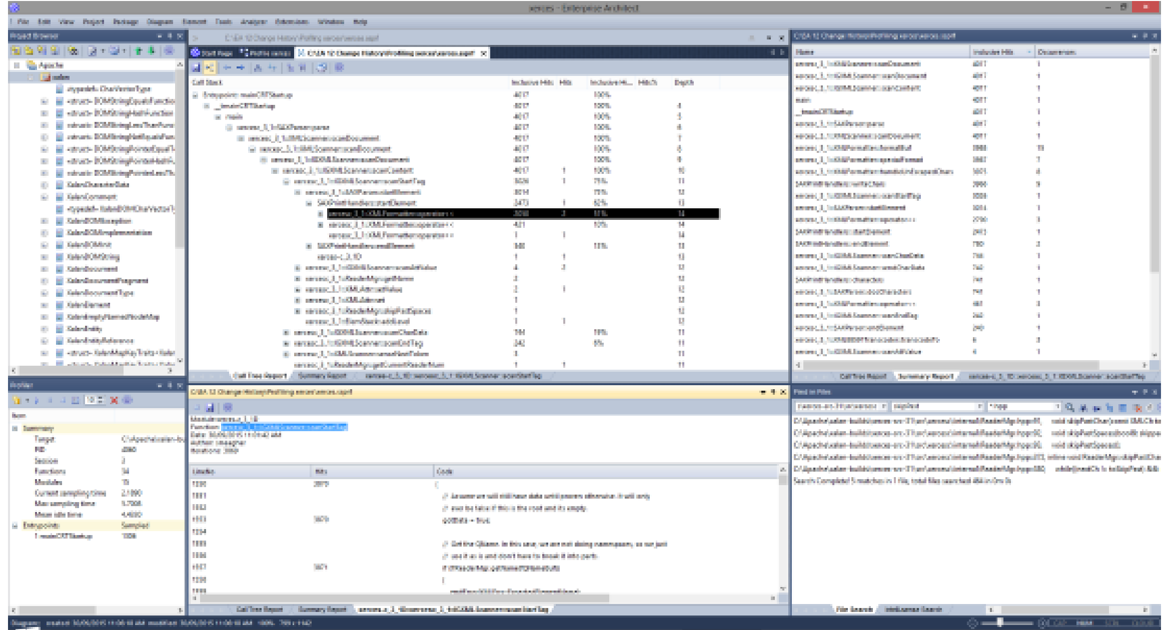
Version: 16.1

创建于

目录

剖析 3
- 系统需求 9
- 开始 10
- 调用图 12
- 堆栈配置 15
- 内存配置 17
- 内存泄漏 19
- 设置选项 22
- 启动和停止分析器 24
- 函数行报告 26
- 生成，保存和加载概要文件报告 29
- 在团队图书馆中保存报告 34

剖析

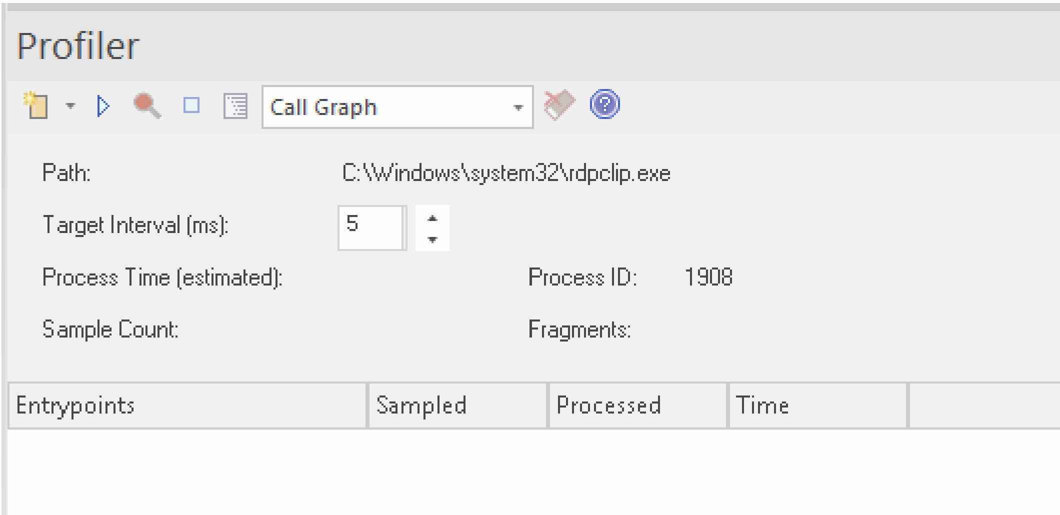


在软件应用程序的生命周期中，调查确定执行速度低于预期的应用程序任务的情况并不少见。您可能还只是想知道当您 按下此按钮"时发生了什么！您可以使用Enterprise Architect的 Profiler 快速解决此问题。结果通常可以在几秒钟内产生，您将很快能够看到正在使用应用程序的操作和所涉及的功能。在特征执行分析器中，特征采用了两个独立的策略；进程执行过程中的采样和进程。一种方法是定期采集样本以识别 CPU 密集型模式，而另一种方法是连接进程以记录对内存的需求。分析数据以产生加权调用图。行为通常可识别为图中的根（入口点），或这些点附近的分支。所有报告都可以按需审查。它们可以在模型中保存为文件，作为工件和团队图书馆发布。

访问

功能区	执行 > 工具 > 探查器
其它	执行分析器工具栏：分析器窗口|探查器

调用抽样



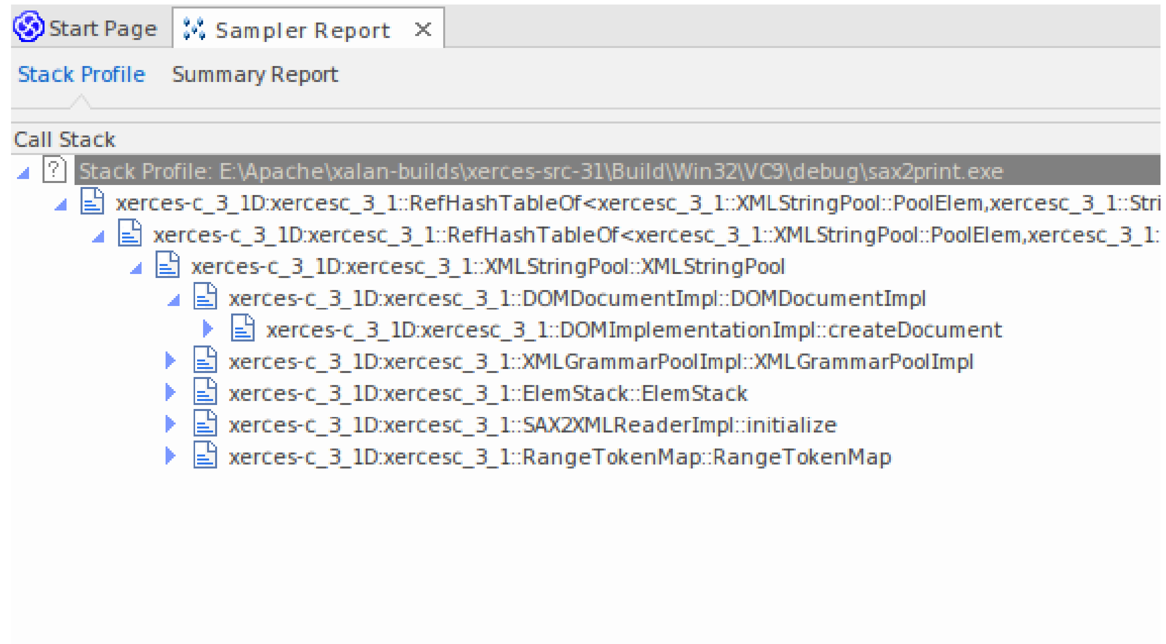
Profiler 使用其工具栏按钮进行控制。在这里，您可以将 Profiler 附加到现有进程（或 JVM），或为活动的分析器脚本应用程序。 Profiler 窗口显示目标进程的详细信息，因为它被分析。这些详细信息提供反馈，让您查看采集的样本数量。您还可以选择暂停和恢复捕获、清除捕获的数据和生成报告。您可以通过暂停捕获来访问报告特征- 在数据捕获过程中报告特征被禁用。

加权调用图

Call Stack	Inclusive Hits	Hits
⊟ xercesc_3_1::SAX2XMLReaderImpl::parse	16051	
⊟ xercesc_3_1::XMLScanner::scanDocument	16051	
⊟ xercesc_3_1::IGXMLScanner::scanDocument	16051	
⊟ xercesc_3_1::IGXMLScanner::scanContent	16051	
⊟ xercesc_3_1::IGXMLScanner::scanStartTagNS	16051	
⊟ xercesc_3_1::IGXMLScanner::resolveSchemaGrammar	16051	
⊟ xercesc_3_1::SchemaValidator::preContentValidation	16049	
⊟ xercesc_3_1::ComplexTypeInfo::checkUniqueParticleAttribution	16049	
⊟ xercesc_3_1::ComplexTypeInfo::makeContentModel	16049	
⊟ xercesc_3_1::DFAContentModel::DFAContentModel	16047	
⊟ xercesc_3_1::DFAContentModel::buildDFA	15998	515
⊟ xercesc_3_1::CMStateSet::operator|=	8174	8093
memcpy	32	32
⊞ xercesc_3_1::CMStateSet::allocateChunk	27	1
__security_check_cookie	21	21
TrailUpVec	1	1
⊞ xercesc_3_1::CMStateSet::~CMStateSet	3573	4
⊞ xercesc_3_1::XMemory::operator delete	841	2
xerces-c_3_1D	4416	2
xercesc_3_1::CMStateSet::getBit	1036	1036
⊞ xercesc_3_1::DFAContentModel::buildSyntaxTree	528	3
⊞ xercesc_3_1::CMStateSet::CMStateSet	373	3
xercesc_3_1::CMStateSet::getBitCountInRange	285	285
⊞ xercesc_3_1::XMemory::operator new	211	2
⊞ xercesc_3_1::CMStateSet::zeroBits	154	
⊞ xercesc_3_1::CMStateSetEnumerator::nextElement	153	136
⊞ xercesc_3_1::RefHashTableOf<xercesc_3_1::XMLInteger,›	59	2
⊞ xercesc_3_1::RefHashTableOf<xercesc_3_1::XMLInteger,›	28	2
⊞ xercesc_3_1::RefHashTableOf<xercesc_3_1::XMLInteger,›	25	
⊞ xercesc_3_1::DFAContentModel::makeDefStateList	25	2

该详细报告将调用堆栈/行为的唯一集合显示为加权调用图。每个分支的权重由命中计数来描述，该计数是该分支的总命中加上从该点开始的所有分支。通过跟踪命中轨迹，您可以快速识别在捕获期间占用程序最多的代码区域。

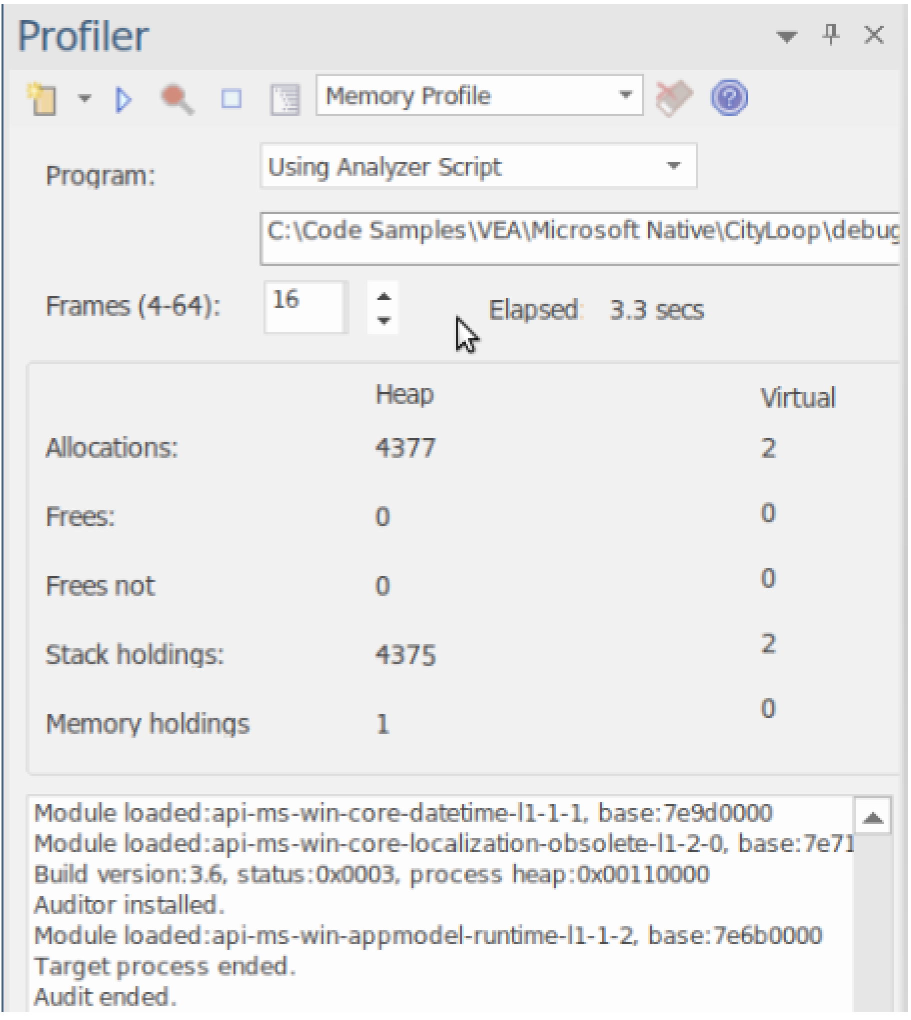
堆栈配置



堆栈Profiles用于发现在程序运行期间调用特定函数的不同方式（堆栈）和方式计数。与其他分析器模式不同，此配置文件是通过使用配置文件点激活的，这是一种特殊的断点标记。标记像任何其他断点一样设置在源代码中。当程序遇到断点时，堆栈被捕获。当您稍后生成报告时，会分析堆栈并生成加权调用图。该图显示了在分析器运行期间该函数所涉及的唯一堆栈， 命中计数"列表示相同堆栈发生的次数。

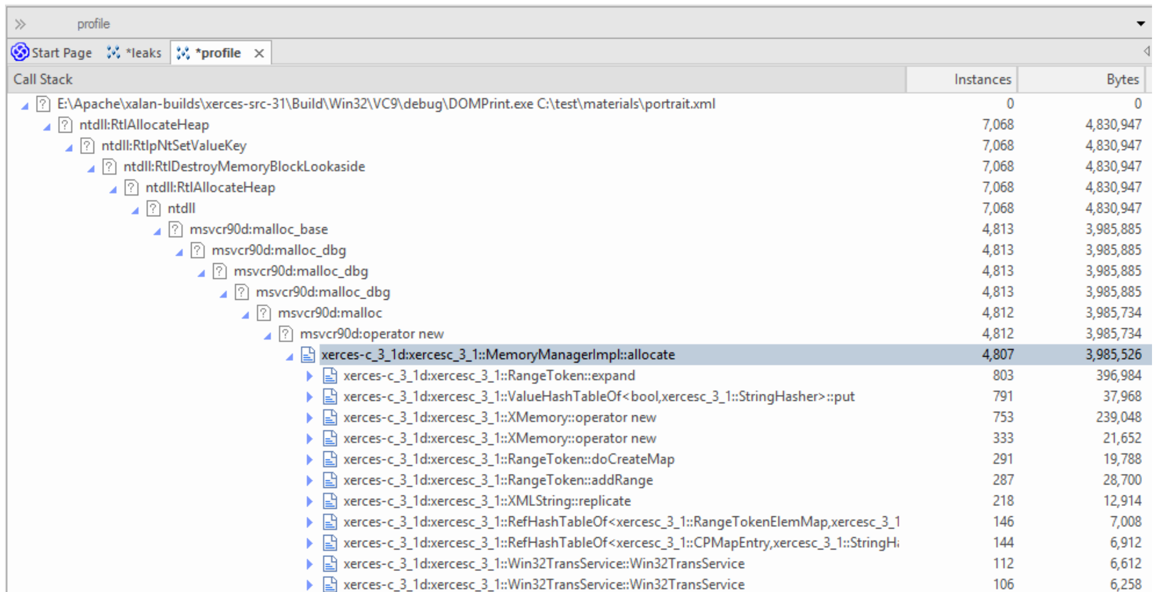
```
106
107 template <class TVal, class THasher>
108 void RefHashTableOf<TVal, THasher>::initialize(const XMLSize_t modulus)
109 {
110     if (modulus == 0)
111         ThrowXMLwithMemMgr(IllegalArgumentException, XMLExcepts::HshTbl_ZeroMo
112
113     // Allocate the bucket list and zero them
114     fBucketList = (RefHashTableBucketElem<TVal>**) fMemoryManager->allocate
115     (
116         fHashModulus * sizeof(RefHashTableBucketElem<TVal>*)
117     );
118     for (XMLSize_t index = 0; index < fHashModulus; index++)
119         fBucketList[index] = 0;
120 }
121
```

内存Profiles



内存配置跟踪分配，忽略内存何时释放。它使用此信息来评估执行代码对内存的需求，而不是内存量，而是需求频率。 *Allocations*数字是请求的内存分配总数。 *Stack Holdings*是在那些时间进行的堆栈跟踪的数量，而*Heap Holding*数字是这些调用获得的内存总量。注记可以按需打开和关闭分析。由于不涉及任何链接，因此也无需重建程序以使其正常工作。

内存图



此示例是根据 Apache 的 Xerces 项目中的演示程序分析生成的报告。该程序对提供的 XML 文件的文档物件模型 (DOM) 进行迭代。

函数总结报告

Name	Inclusive Hits
profiler/Example.Run	156
profiler/Example.main	156
java/io/FileOutputStream.write	154
java/io/PrintStream.println	154
profiler/Example.Print	154
profiler/Example.MakeItalianCars	2
profiler/Example.NewCar	2

此摘要报告列出了函数，并且仅列出了在样本期间执行的函数。函数按总调用次数列出，其中在单独的调用堆栈中出现两次的函数出现在仅出现一次的函数之前。

函数线报告

LineNo	Hits	Code
54	1	for(int n = 0; n < 10000; n++)
55		{
56	1408	m_Cars = new Collection<Car>();
57	1408	if((n % 3)>0)
58		{
59	938	for(int i = 0; i < 1000; i++)
60		{
61	938000	MakeItalianCars();
62		}

此详细报告逐行显示函数的源代码，并在其旁边显示每个函数的总执行次数。我们使用这份报告发现了代码，该代码暴露了代码中似乎从未执行过的 case 语句。

支持

Profiler 支持以 C、C++、Visual Basic、 Java和 Microsoft .NET语言编写的程序。内存分析目前可用于本机 C 和 C++ 程序。

注记

- Profiler 在Enterprise Architect专业版及以上版本中可用
- Profiler 也可以在WINE （ Linux 和 Mac ）下用于对WINE环境中部署的标准窗口应用程序进行分析

系统需求

使用 Profiler，您可以分析为这些平台构建的应用程序：

- Microsoft ™ Native（C++、C、Visual Basic）
- Microsoft .NET （支持托管和非托管代码的混合）
- Java

微软本机应用程序

对于 C、C++ 或 Visual Basic 应用程序，Profiler 要求应用程序使用 Microsoft ™ Native 编译器进行编译，并且对于每个感兴趣的应用程序或模块，都有可用的 PDB 文件。 Profiler 可以对应用程序的调试和发布配置进行采样，前提是每个可执行文件的 PDB 文件存在并且是最新的。

Microsoft .NET应用程序

对于 Microsoft .NET应用程序，Profiler 要求安装适当的 Microsoft .NET框架，并且对于要分析的每个应用程序或模块，都有可用的 PDB 文件。

Java

对于Java ，Profiler 要求安装来自 Oracle 的适当 JDK。

感兴趣的类也应该使用调试信息进行编译。例如："java -g *.java"

- 从Enterprise Architect启动应用程序 VM 的新实例 - 无需其他操作
- 现有应用程序虚拟机从Enterprise Architect中附加 - 目标Java虚拟机必须已使用Enterprise Architect分析代理启动

以下是使用特定 JVMTI 代理创建Java VM 的命令行示例：

1. java.exe -cp "%classpath%;.\" -agentpath:"C:\Program Files (x86)\ Sparx Systems \EA\vea\x86\ssamplerlib32" myapp
2. java.exe -cp "%classpath%;.\" -agentpath:"C:\Program Files (x86)\ Sparx Systems \EA\vea\x64\ssamplerlib64" myapp

（有关 -agentpath VM 启动选项的详细信息，请参阅 JDK 文档。）

开始

Profiler 可用于调查性能问题，提供四种不同的工具供您选择，即：

- 调用图
- 堆栈配置
- 内存配置
- 内存泄漏

您可以从 Profiler 工具栏中选择这些工具。

访问

功能区	执行 > 工具 > 探查器

工具

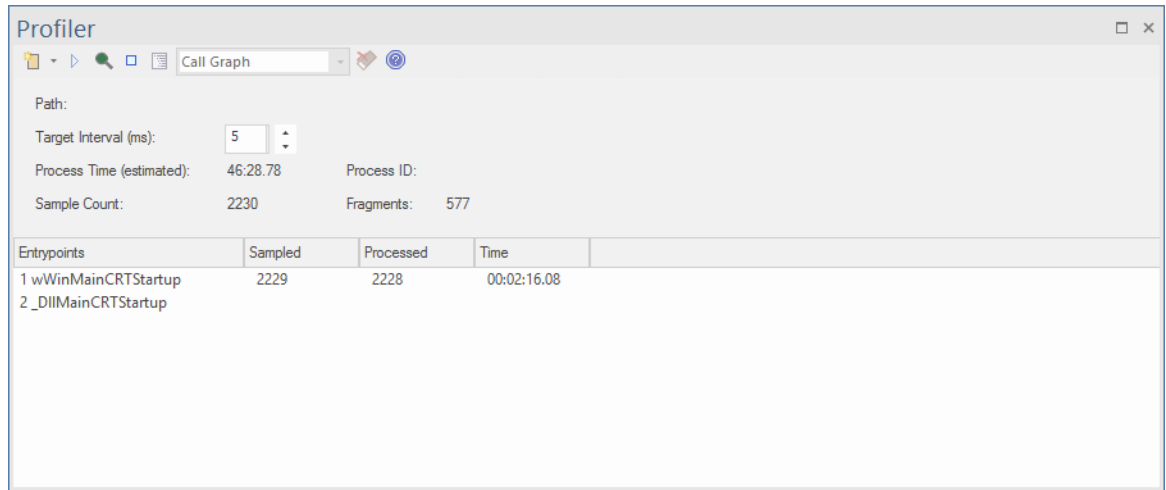
工具	描述
调用图	通过在程序中的活动期间取样来分析性能。每个样本代表一个堆栈。使用工具栏控制的间隔采集样本。在这种情况下，性能不佳是通过在采样时间段内重复最多的行为模式来评定的。这个数字是用来衡量生成的调用图的。
内存配置	通过挂钩程序进行的内存分配来分析性能。在这种情况下，性能不佳是由对内存请求最多的活动来评定的。这个数字是用来衡量生成的调用图的。
堆栈配置	Stack Profiler 使您能够在源代码中设置标记，以便每当执行命中该标记时，都会捕获完整的堆栈跟踪。随着应用程序继续执行并从正在运行的可执行文件中的多个位置访问标记的位置，将构建一个非常详细且有用的图片，显示代码中特定点的热点和使用场景。 堆栈配置配置报告与内存配置报告一样，以 反向堆栈"的顺序显示。这意味着报告的根始终是单个节点（在本例中为标记），然后树呈扇形展开以显示已访问标记位置的所有不同位置。
内存泄漏	通过挂钩程序执行的内存操作来分析内存泄漏。生成的是调用图，表示调用堆栈分配了未检测到空闲操作的内存。

工具栏按钮

按钮	行动
	显示用于管理分析会话的选项菜单。
	启动要分析的已配置应用程序。默认情况下，这是在主动分析器脚本配置的

	应用程序。
	指示采样器的状态。绿色时，启用采样；当红色时，采样被禁用。
	停止 Profiler 进程；如果已收集任何样本，则报告按钮和丢弃数据按钮处于活动状态。
	从当前数据集合生成报告。
	显示正在使用的 Profiling 工具，该工具确定 Profiler 窗口中显示的字段。单击下拉箭头并选择一个不同的工具，该工具会更改窗口字段。
	丢弃收集的数据。系统会提示您确认丢弃。
	显示此窗口的帮助主题。

调用图



- 快速发现程序在任何时间点正在做什么
- 轻松识别性能问题
- 惊讶于您能以多快的速度实现改进
- 查看您在工作中的改进并拥有证据
- 支持 C/C++ .NET和Java平台

用途

调用图"选项通常用　活动执行速度比预期慢的情况，但它也可以简单地用于更好地理解活动期间的行为模式。

手术

Profiler 通过在一段时间内定期采样 - 或调用堆栈 - 进行操作；使用 Profiler 工具栏设置间隔。您可以使用运行特定程序，也可以附加到现有进程。 Profiler 捕获是受控的，您可以随时暂停和恢复捕获。您还可以选择在 Profiler 启动时立即启动捕获。如有必要，您可以丢弃任何捕获的样本并在同一会话期间重新开始。如果您无法继续进行相同的会话，重新启动 Profiler 既快捷又简单。

注记 进程时间(estimated)"字段显示被分析的进程已经运行了多长时间的估计，考虑到分析器在收集样本时对进程的中断。

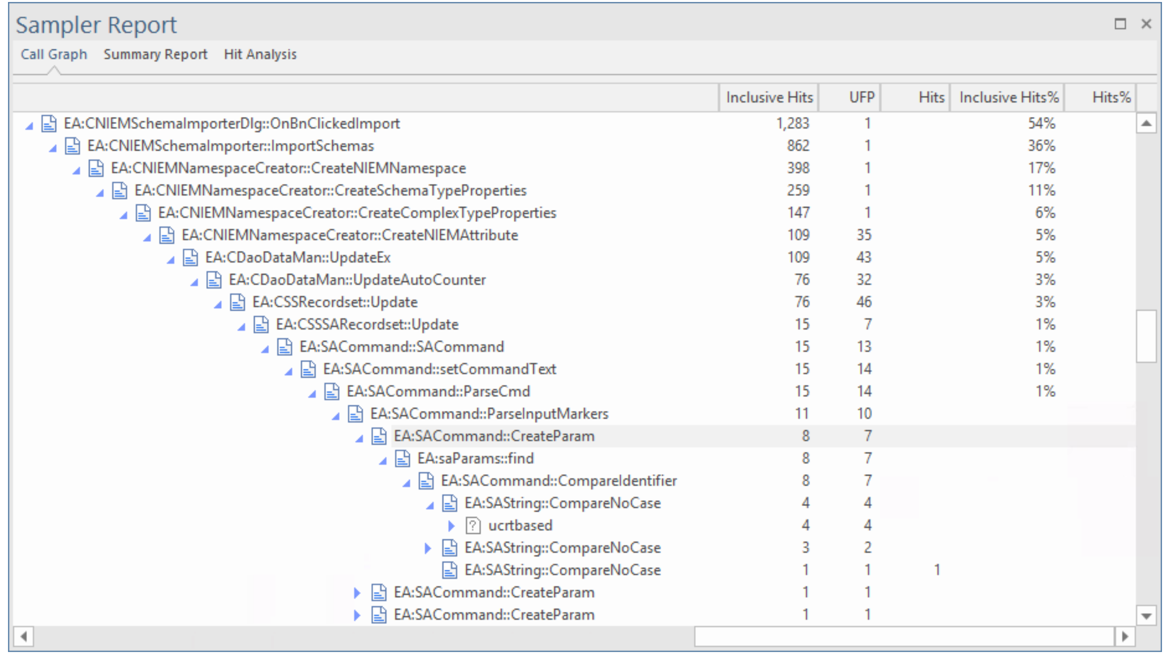
结果

会议期间可随时产生结果；但是，必须禁用捕获才能使 报告"按钮变为活动状态。让运行多长时间由您决定。您可能知道活动何时完成，或者由于其他原因可能很明显。您在这里的原因可能是一项活动根本没有完成。

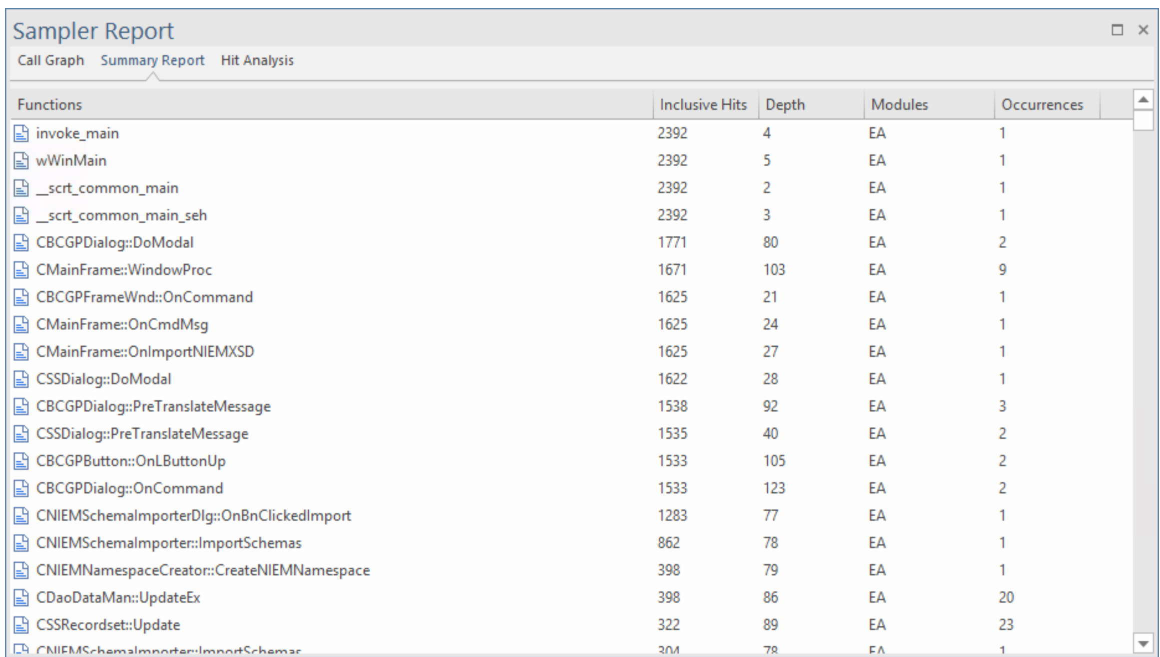
报告按钮将通过暂停捕获或完全停止 Profiler 来启用。

结果显示在报告视图中。报告打开时最初可见三个选项卡：调用图、摘要报告 (函数摘要) 和命中分析选项卡。报告可以保存为文件，存储在模型工件中或张贴在团队图书馆中。

调用图选项卡



摘要报告选项卡



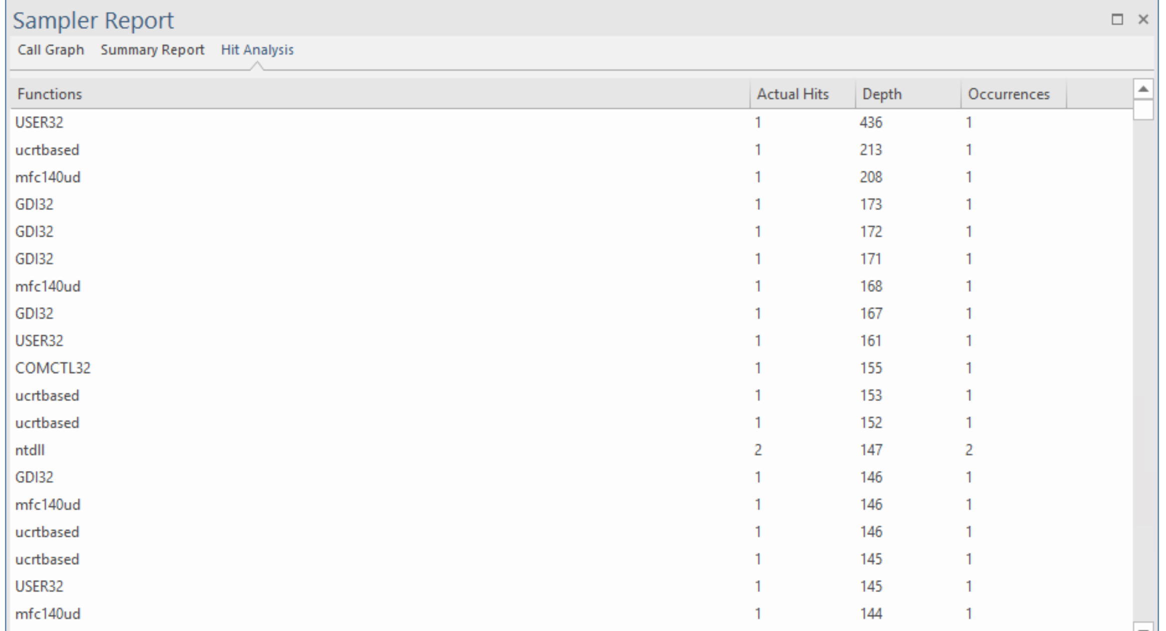
命中分析选项卡

命中分析"选项卡显示了许多列：

- 函数：函数的名称（如果模块没有符号，则为模块）
- Hits：执行函数的采样数。
- 深度：发生命中的帧号或堆栈深度。
- Occurrences：函数在此特定堆栈深度处被命中的次数

特定函数的命中数根据采样时的堆栈帧深度进行聚合。

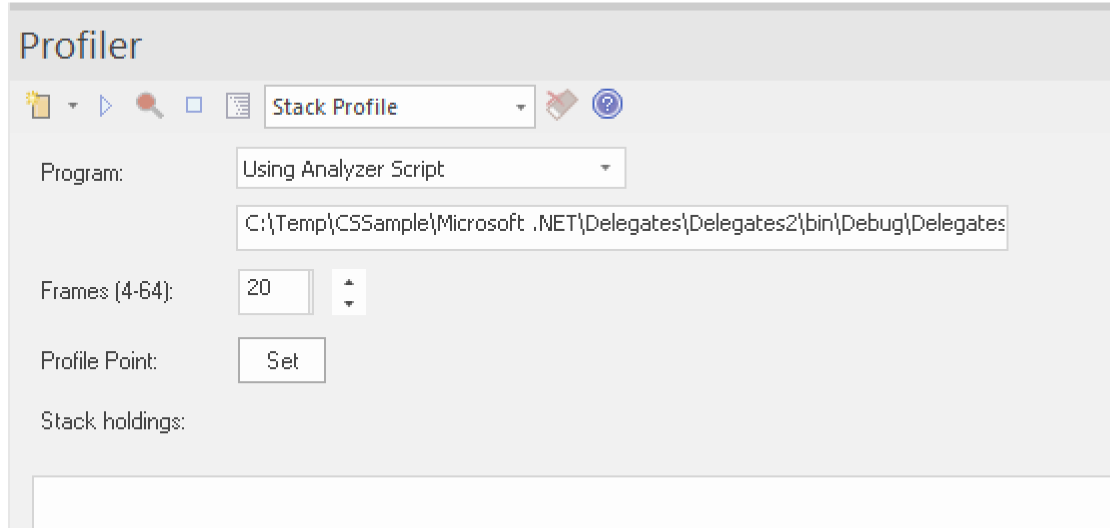
如果函数名不可用，例如窗口系统DLL，如 User32 或没有调试信息的 DLL，则显示模块名。



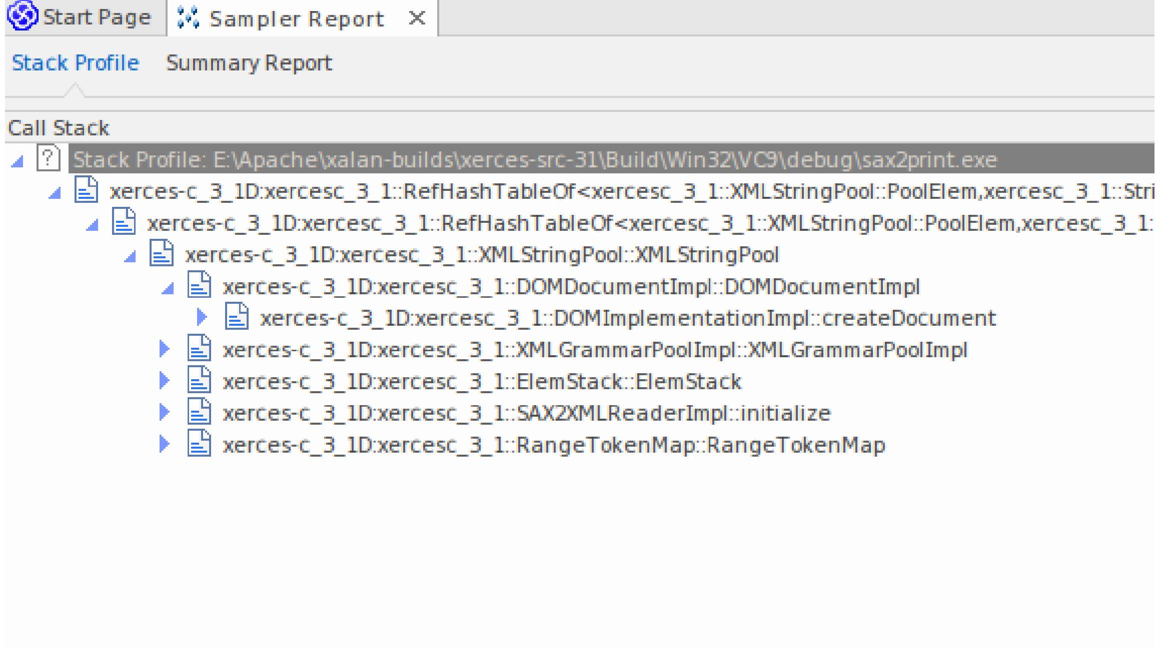
Functions	Actual Hits	Depth	Occurrences
USER32	1	436	1
ucrtbased	1	213	1
mfc140ud	1	208	1
GDI32	1	173	1
GDI32	1	172	1
GDI32	1	171	1
mfc140ud	1	168	1
GDI32	1	167	1
USER32	1	161	1
COMCTL32	1	155	1
ucrtbased	1	153	1
ucrtbased	1	152	1
ntdll	2	147	2
GDI32	1	146	1
mfc140ud	1	146	1
ucrtbased	1	146	1
ucrtbased	1	145	1
USER32	1	145	1
mfc140ud	1	144	1

堆栈配置

Stack Profiler 使您能够在源代码中设置标记，以便每当执行命中该标记时，都会捕获完整的堆栈跟踪。随着应用程序继续执行并从正在运行的可执行文件中的多个位置访问标记的位置，将构建一个非常详细且有用的图片，显示代码中特定点的热点和使用场景。



堆栈配置配置报告与内存配置报告一样，以 反向堆栈”的顺序显示。这意味着报告的根始终是单个节点（在本例中为标记），然后树呈扇形展开以显示已访问标记位置的所有不同位置。



用途

使用堆栈配置模式生成报告，显示在程序运行期间可以调用函数的独特方式。确定依赖此函数的模型部分及其频率。

手术

```
106
107 template <class TVal, class THasher>
108 void RefHashTableOf<TVal, THasher>::initialize(const XMLSize_t modulus)
109 {
110     if (modulus == 0)
111         ThrowXMLwithMemMgr(IllegalArgumentException, XMLExcepts::HshTbl_ZeroMo
112
113     // Allocate the bucket list and zero them
114     fBucketList = (RefHashTableBucketElem<TVal>**) fMemoryManager->allocate
115     (
116         fHashModulus * sizeof(RefHashTableBucketElem<TVal>*)
117     );
118     for (XMLSize_t index = 0; index < fHashModulus; index++)
119         fBucketList[index] = 0;
120 }
121
```

使用 Profiler 控件工具栏选择 Profiler 模式。如果已创建 Profiler Point，则会显示它。 Profiler Point 是捕获堆栈跟踪的点。选择模式后，您可以使用控件本身的 Set 按钮设置 Profiler Point。确定配置文件点后，构建项目以确保一切都是最新的，然后启动 Profiler。在运行期间可以看到检测到的唯一运行持有量。

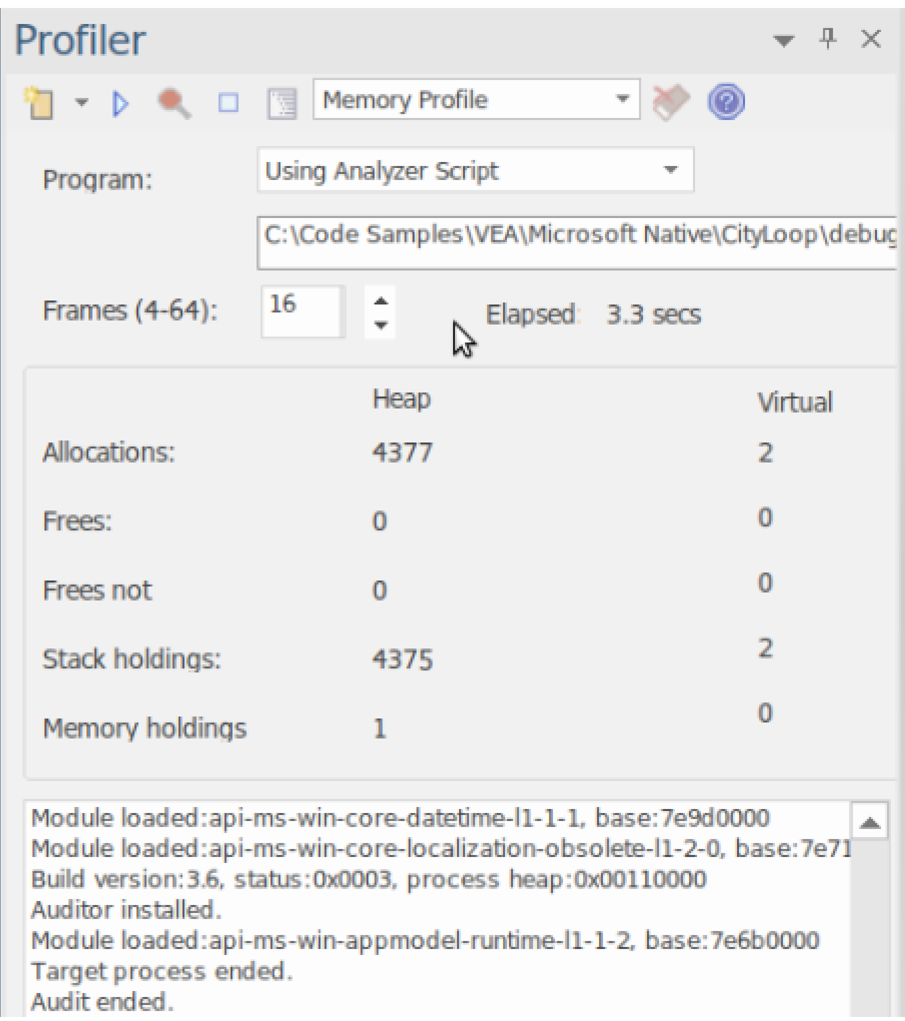
结果

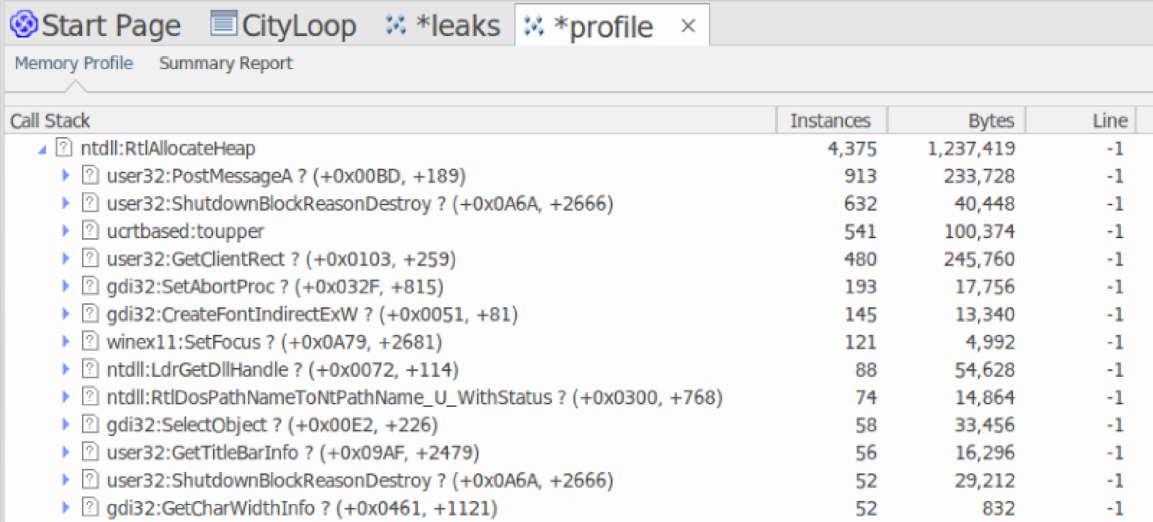
单击 Profiler 控件工具栏上A报告按钮可以生成结果。此按钮在以下任一情况下启用：

- 捕获已关闭（使用暂停按钮）或
- Profiler 停止（使用停止按钮）

生成的结果显示为加权调用图，其中图表上的线条代表唯一的堆栈，并加权以首先显示较高频率的堆栈。然后可以使用报告本身的上下文菜单将报告保存到文件或模型中。

内存配置





- 快速评估您感兴趣的活动的表现
- 没有什么比证据更能影响讨论了
- 通过在那些会有所作为的领域工作来奖励您的努力

- 通过提供您可能不知道存在的优化来给自己惊喜

用途

内存配置可用于揭示活动在内存消耗方面的执行情况。使用这种模式，用户会对在任务期间对记忆的需求频率感兴趣。他们对实际消费量不太感兴趣。 A管理良好的活动可能会进行相对较少的调用来分配资源，但会分配足够的内存来有效地完成其工作。其他活动可能会发出其它请求，这通常会降低它们的效率。此模式对于检测这些场景很有用。

手术

内存配置通过挂钩有问题的进程来工作，因此必须使用Enterprise Architect中的工具启动该程序。与调用图选项不同，您不能附加到现有进程。当程序启动时，挂钩机制跟踪内存分配；此信息在Enterprise Architect中收集和整理。您可以轻松监控正在分配的数量。此外，过程是受控的；也就是说，内存挂钩可以按需打开和关闭。如果您可能错过了某些操作，您可以轻松地暂停捕获、丢弃数据并再次恢复捕获。

结果

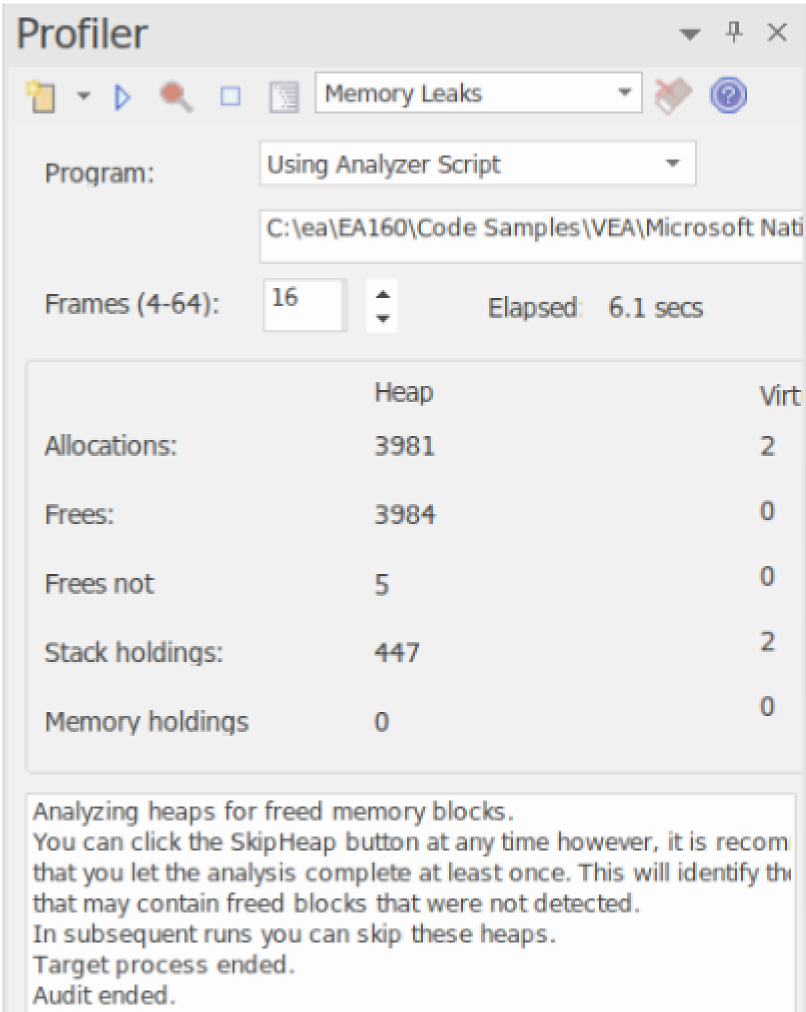
会议期间可随时产生结果；但是，必须禁用捕获才能使 报告"按钮变为活动状态。让运行多长时间由您决定。您可以通过暂停捕获或完全停止 Profiler 来启用 Report 按钮。

结果显示在报告视图中。报告最初打开时显示两个选项卡；一个单独的加权调用图和一个函数摘要。调用图描述了导致内存分配的所有调用堆栈，这些调用堆栈根据模式的频率进行聚合和加权。

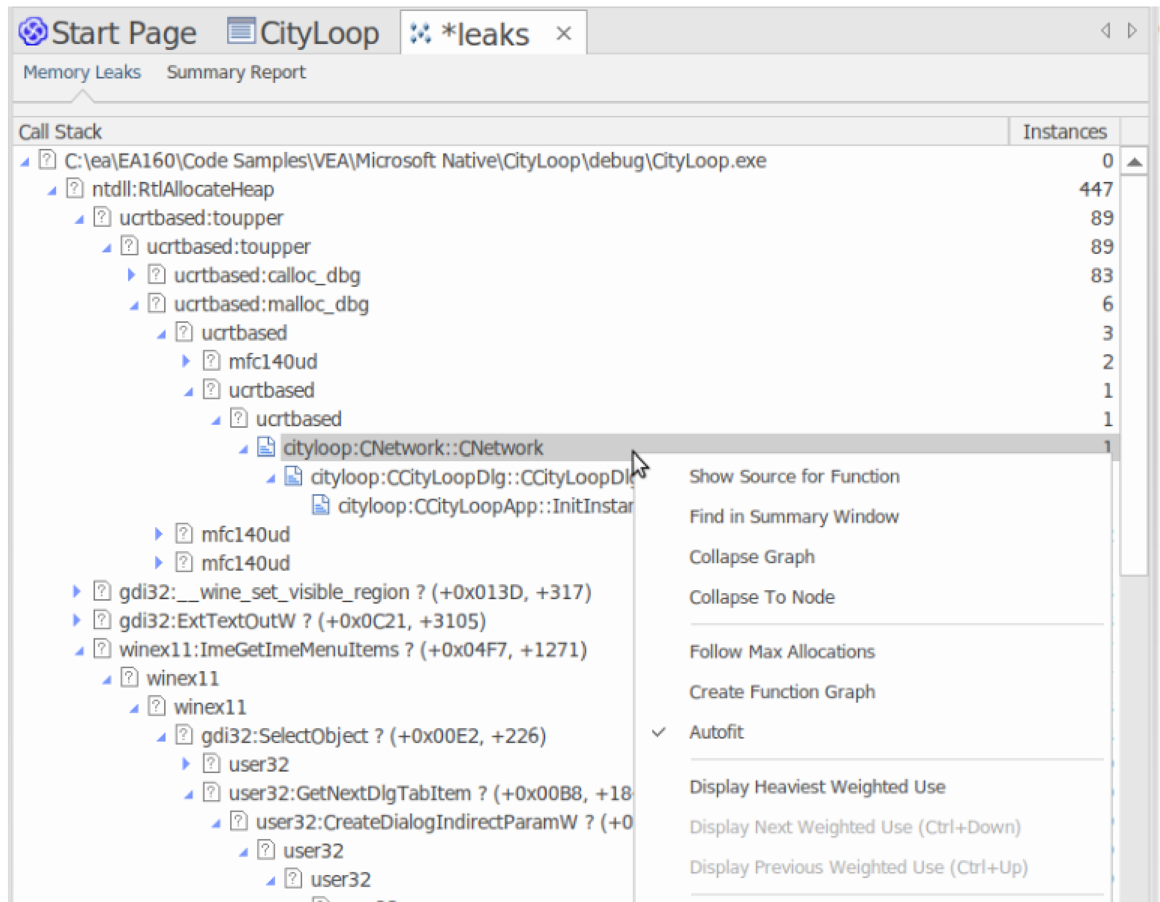
需求

为获得最佳结果，应在构建映像及其模块时包含调试信息，并且不进行优化。任何具有帧指针省略 (FPO) 优化的模块都可能产生误导性结果。

内存泄漏



Profiler 控件，显示内存分配计数和可用内存操作的计数。



A表现良好的程序。

内存泄漏检测是一条行之有效的道路。尽管还有许多其他不错的选择，但我们相信我们的方法有很大的好处，例如：

- 对现有项目构建完全没有更改
- 项目代码不需要头文件
- 无需担心运行时依赖项
- 无需考虑系统配置

用途

人们A使用此模式来跟踪应用程序或应用程序内的活动中的内存泄漏。 A Profiler 的角度来看，内存泄漏是对内存分配函数的成功调用，该函数返回一个内存地址，没有针对该地址进行匹配调用来释放该地址。

手术

内存泄漏检测通过挂钩工作。进程的内存例程被挂钩以跟踪何时分配和释放内存。调用堆栈在分配点被捕获，并在Enterprise Architect中整理此信息以生成调用图形式的报告。捕获受控；也就是说，可以根据需要启用或禁用挂钩机制。

根据程序的类型及其内存消耗，您可以采用适当的策略。对于小型程序，您可能会从头到尾跟踪程序。对于较大的窗口程序，您可能会通过在特定任务之前和之后切换捕获来避免跟踪太多数据来做得更好。

结果

会议期间可随时产生结果；但是，必须禁用捕获才能使 报告”按钮变为活动状态。让运行多长时间由您决定。您可以通过暂停捕获或完全停止 Profiler 来启用 Report 按钮。

结果显示在报告视图中。报告最初打开时显示两个选项卡；一个单独的加权调用图和一个函数摘要。调用图描述了导致内存分配的所有调用堆栈，并根据模式的频率进行聚合和加权。

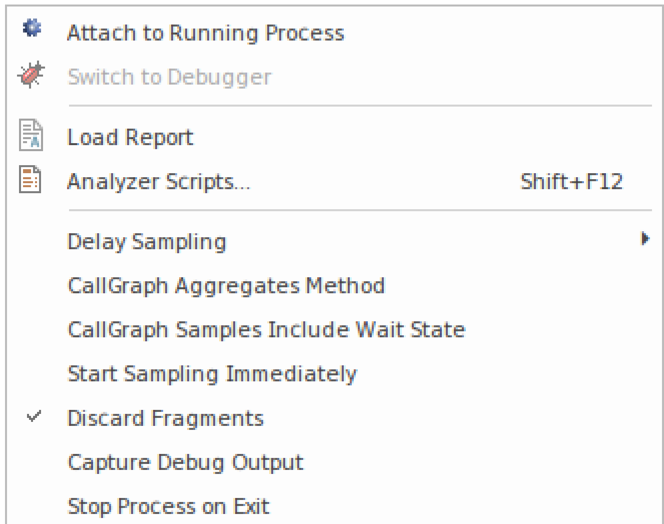
报告可能包含可变数量的 噪音”。要聚焦您特别关注的领域，请在摘要报告中找到您已知的函数，并使用它直接导航到图表中的特征线。

需求

为获得最佳结果，应在构建映像及其模块时包含调试信息，并且不进行优化。任何具有帧指针省略 (FPO) 优化的模块都可能产生误导性结果。

设置选项

Profiler 窗口工具栏上的第一个图标显示了一个选项列表，您可以设置这些选项来定制您的 Profiling 会话。



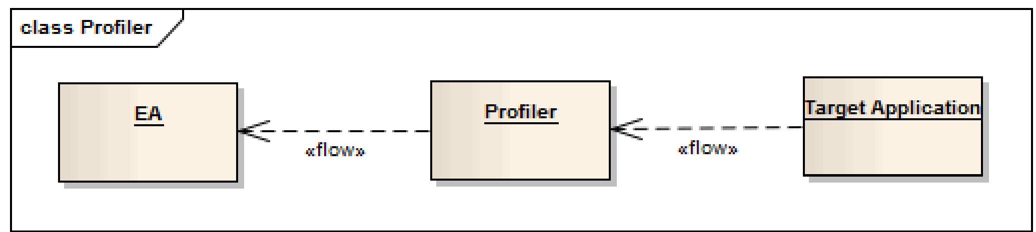
选项

选项	描述
附加到正在运行的进程	选择此选项以显示 附加到进程文件"对话框，您可以从中选择一个活动进程到配置文件。
切换到调试器	选择此选项可将操作从分析更改为调试。调试器有一个等效的下拉菜单选项，您可以使用它从调试切换到分析。
负载报告	选择此选项可从文件系统加载以前保存的报告。
分析器脚本	选择此选项打开分析器脚本，这是用于配置构建、调试和所有其他可视化执行分析器选项的模型库。
延迟采样	选择此选项可设置单击 开始分析"选项和分析实际开始之间的延迟。延迟可以是 3、5 或 10 秒。选择 无"以取消任何延迟设置。
CallGraph 聚合方法	选择此选项时，相同堆栈序列的实例将按方法聚合。也就是说，方法中的行号/指令将被忽略，因此两个堆栈将被视为一个堆栈，它们仅在最后一帧中的行号不同。
状态样本包括等待状态	选择此选项后，Profiler 将对所有线程进行采样，包括处于等待状态的线程。未选中时，Profiler 仅对自上次间隔到期以来累积 CPU 时间的线程进行采样。
立即开始采样	选择此选项可在启动时立即触发数据收集。您通常会使用此选项在启动期间对进程进行概要分析。

丢弃片段	当堆栈无法与线程的入口点协调时，它们被称为片段。采样期间遇到的片段数显示在采样器摘要窗口中。您可以设置此选项来收集或丢弃碎片；当丢弃片段选项是： • 已选中，片段不会出现在报告中，尽管遇到的数量仍在更新 • 取消选择，在调用图中创建一个名为 片段”的特殊集合来容纳它们，并确保它们的数据不会与完成样本混合
捕获调试输出	（适用于进程采样）。选中后，调试期间通常可见的输出将被捕获并显示在调试窗口中。注记只有调试版本通常会发出调试输出。
退出进程停止	此选项确定 Profiler 的终止行为。选择该选项后，目标进程将在 Profiler 停止时终止。

启动和停止分析器

剖析是数据收集和报告的两个阶段过程。在Enterprise Architect中，数据收集具有作为后台任务的优势——因此您可以在它运行时自由地做其他事情。发送回Enterprise Architect的信息将被存储，直到您生成报告。要查看报告，必须关闭捕获。生成报告后，您可以单击按钮恢复捕获。如果出于某种原因，您决定废弃数据并重新开始，您可以轻松完成此操作，而无需停止并重新启动程序。



访问

功能区	执行 > 工具 > 探查器 > 打开探查器
其它	执行分析器工具栏：分析器窗口|探查器

行动

行动	细节
工具栏	Profiler Call Graph
策略选择	从工具栏上的可用选项中选择分析策略。
启动开始器	单击运行窗口上的运行按钮
停止探查器	如果出现以下情况，则进程退出： • 你点击停止按钮 • 目标应用程序终止，或 • 你关闭当前模型 如果您停止 Profiler 并且该进程仍在运行，您可以再次快速附加到它。
暂停和恢复捕获	您可以在会话期间随时暂停和恢复捕获。 打开捕获后，将从目标中收集样本。暂停时，Profiler 进入并保持等待状态，直到启用捕获、Profiler 停止或应用程序完成。
生成报告	报告按钮在捕获期间被禁用，但在关闭捕获时可用。
模式下拉菜单	点击下拉菜单，选择Profiling -调用图、堆栈配置、内存配置或内存配置的模

	式。
清除数据收集	您可以随时清除收集的任何数据样本并恢复。首先通过单击暂停按钮暂停捕获。与报告按钮一样，只要关闭捕获，就会启用丢弃按钮。在单击放弃按钮时，您将被要求确认操作。此操作无法撤消。

函数行报告

在正在执行的应用程序上运行 Profiler 并生成运行报告后，您可以通过从该项目生成函数行报告来进一步分析报告中列出的特定函数的活动。函数行报告显示函数A每一行被执行的次数。您一次生成一个函数行报告，使用 Sampler 报告中具有有效源文件的任何方法。函数行报告对于执行包含条件分支的循环的函数特别有用；覆盖率可以提供单个方法中最频繁和最不频繁执行的代码部分的图片。

您生成的线路报告在您保存 Sampler 报告时被保存。函数正文也与函数行报表一起保存，以保存当时的函数状态。

此功能不适用于内存配置报告。

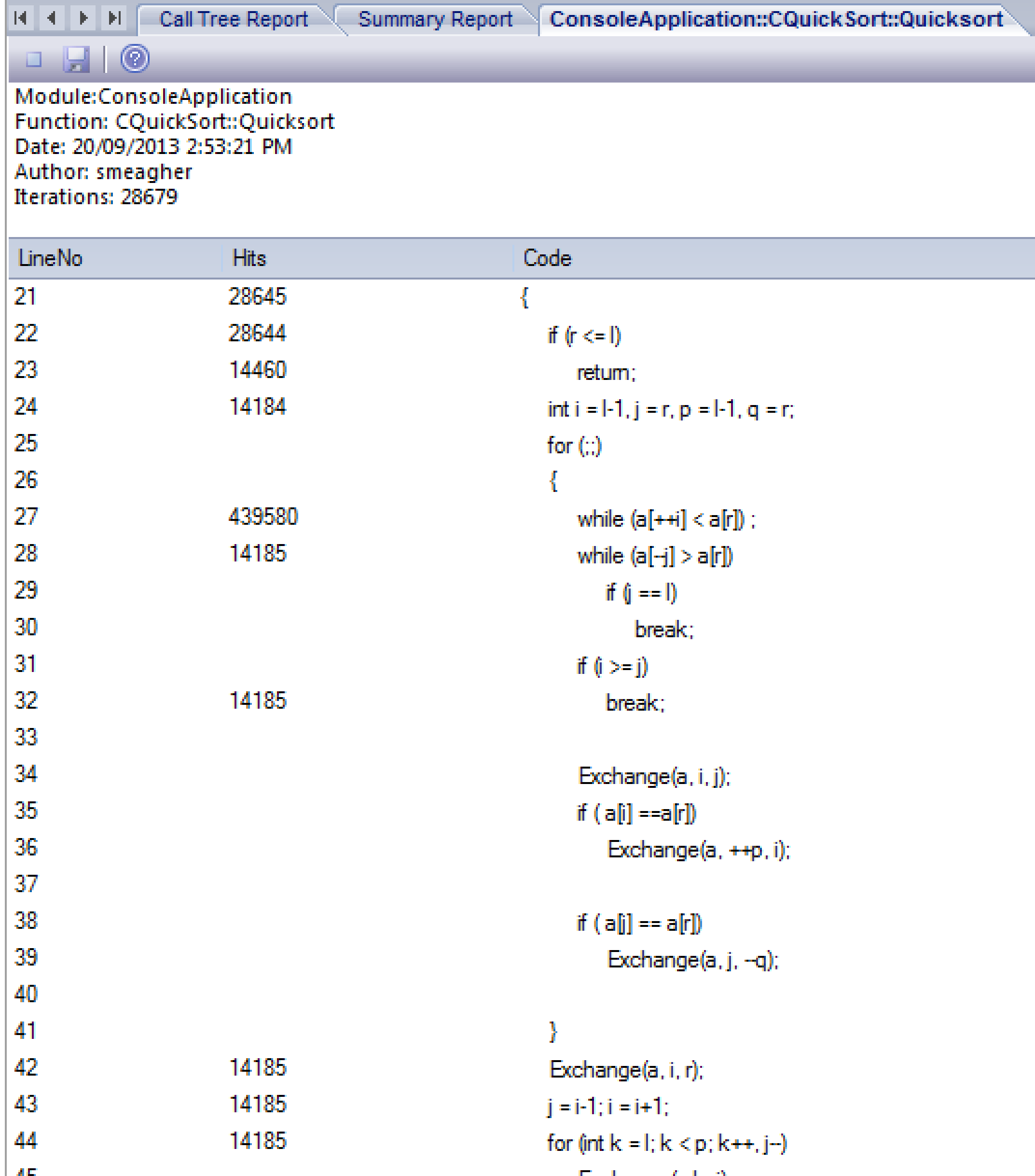
支持的平台

Java , Microsoft .NET和 Microsoft 本机代码

创建行报告

在 Sampler 报告中，右键单击要分析的函数的名称，然后选择 "Create Line Report for函数"选项。

一旦 Profiler 绑定了方法，函数行报告就会在 Sampler Report 窗口中打开。报告显示函数的主体，包括行号和文本。随着每一行的执行，命中值将针对该行累积。计时器A大约每秒更新一次报告。



Module:ConsoleApplication
Function: CQuickSort::Quicksort
Date: 20/09/2013 2:53:21 PM
Author: smeagher
Iterations: 28679

LineNo	Hits	Code
21	28645	{
22	28644	if (r <= l)
23	14460	return;
24	14184	int i = l-1, j = r, p = l-1, q = r;
25		for (;;)
26		{
27	439580	while (a[++i] < a[r]) ;
28	14185	while (a[--j] > a[r])
29		if (j == l)
30		break;
31		if (i >= j)
32	14185	break;
33		
34		Exchange(a, i, j);
35		if (a[i] ==a[r])
36		Exchange(a, ++p, i);
37		
38		if (a[j] == a[r])
39		Exchange(a, j, --q);
40		
41		}
42	14185	Exchange(a, i, r);
43	14185	j = i-1; i = i+1;
44	14185	for (int k = l; k < p; k++, j--)

结束行报告捕获

一旦捕获到足够的信息，或者函数已经结束，单击 Profiler 工具栏上的 Stop 按钮停止记录捕获。

保存报告

使用函数调用堆栈工具栏上的 Save 按钮将 Sampler 报告和任意 Line 报告保存到文件中。

删除线路报告

关闭 行报告”选项卡将关闭该报告，但只有在保存报告时才会删除报告数据。

生成，保存和加载概要文件报告

报告可以在会话期间的任何时间生成，或者在程序结束时自然生成。但是，要在程序运行时启用 Report 按钮，您需要通过切换 Pause/Resume 按钮或使用 Stop 按钮终止 Profiler 来暂停 Profiling。您有一些查看和共享结果的选项：

- 视图报告
- 将报告保存到文件
- 将报告作为团队图书馆资源分发
- 将文档附加到工件元素
- 通过对参与配置文件的源代码进行逆向工程来同步模型

访问

功能区	执行 > 工具 > 探查器 > 从当前数据创建报告
探查器	在 Profiler 窗口中，单击工具栏中的图标。

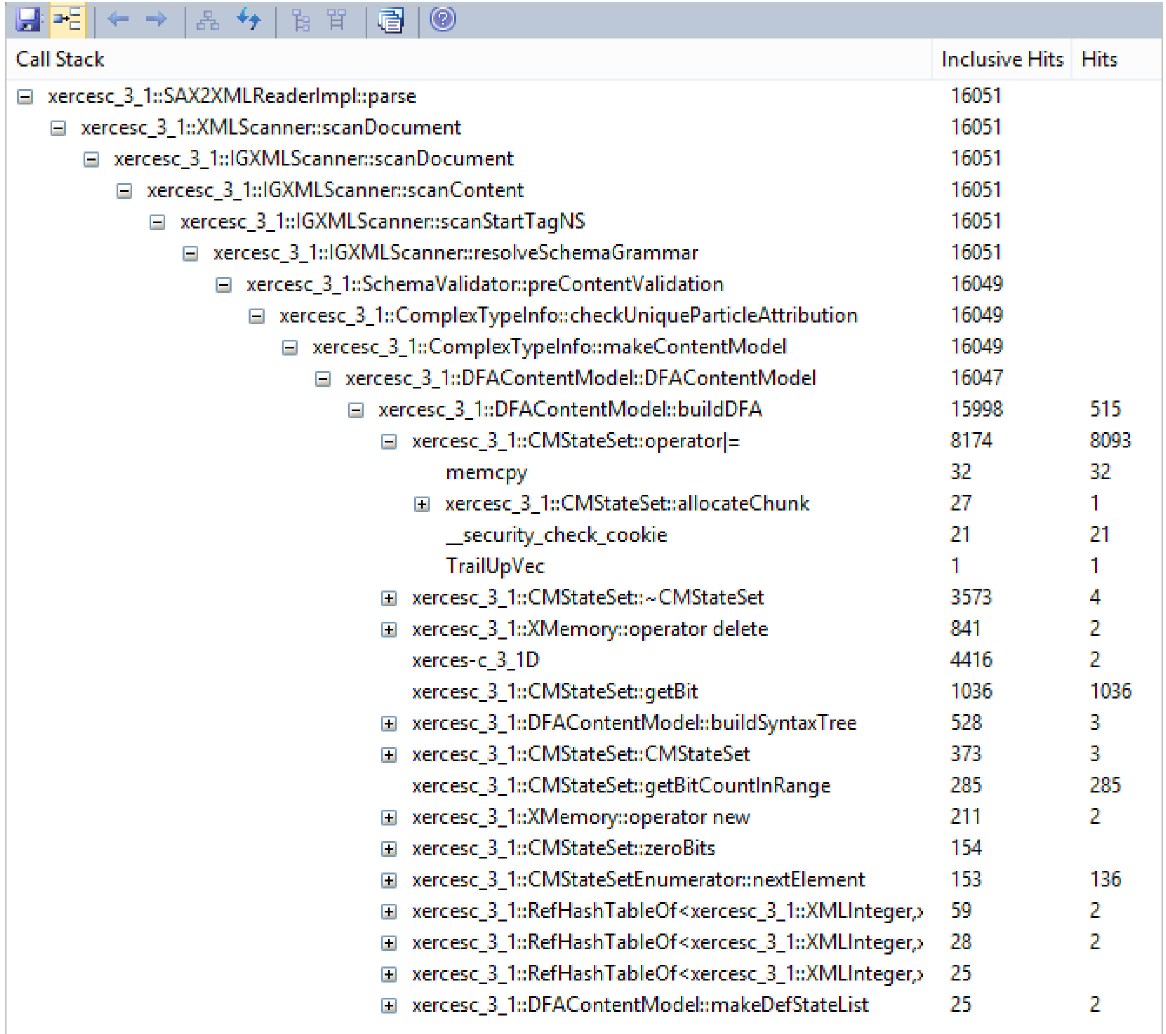
从文件加载报告

该选项可从 Profiler 窗口的下拉菜单中获得

生成报告

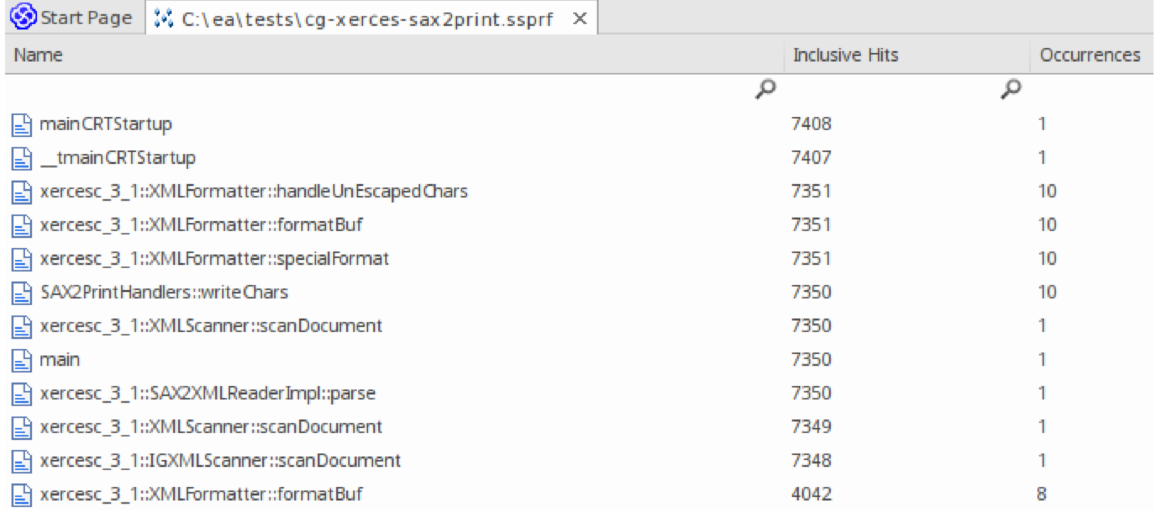
在 Profiler 窗口中，单击工具栏中的图标。

呼叫频率报告



Call Stack	Inclusive Hits	Hits
xercesc_3_1::SAX2XMLReaderImpl::parse	16051	
xercesc_3_1::XMLScanner::scanDocument	16051	
xercesc_3_1::IGXMLScanner::scanDocument	16051	
xercesc_3_1::IGXMLScanner::scanContent	16051	
xercesc_3_1::IGXMLScanner::scanStartTagNS	16051	
xercesc_3_1::IGXMLScanner::resolveSchemaGrammar	16051	
xercesc_3_1::SchemaValidator::preContentValidation	16049	
xercesc_3_1::ComplexTypeInfo::checkUniqueParticleAttribution	16049	
xercesc_3_1::ComplexTypeInfo::makeContentModel	16049	
xercesc_3_1::DFAContentModel::DFAContentModel	16047	
xercesc_3_1::DFAContentModel::buildDFA	15998	515
xercesc_3_1::CMStateSet::operator|=	8174	8093
memcpy	32	32
xercesc_3_1::CMStateSet::allocateChunk	27	1
__security_check_cookie	21	21
TrailUpVec	1	1
xercesc_3_1::CMStateSet::~CMStateSet	3573	4
xercesc_3_1::XMemory::operator delete	841	2
xerces-c_3_1D	4416	2
xercesc_3_1::CMStateSet::getBit	1036	1036
xercesc_3_1::DFAContentModel::buildSyntaxTree	528	3
xercesc_3_1::CMStateSet::CMStateSet	373	3
xercesc_3_1::CMStateSet::getBitCountInRange	285	285
xercesc_3_1::XMemory::operator new	211	2
xercesc_3_1::CMStateSet::zeroBits	154	
xercesc_3_1::CMStateSetEnumerator::nextElement	153	136
xercesc_3_1::RefHashTableOf<xercesc_3_1::XMLInteger,>	59	2
xercesc_3_1::RefHashTableOf<xercesc_3_1::XMLInteger,>	28	2
xercesc_3_1::RefHashTableOf<xercesc_3_1::XMLInteger,>	25	
xercesc_3_1::DFAContentModel::makeDefStateList	25	2

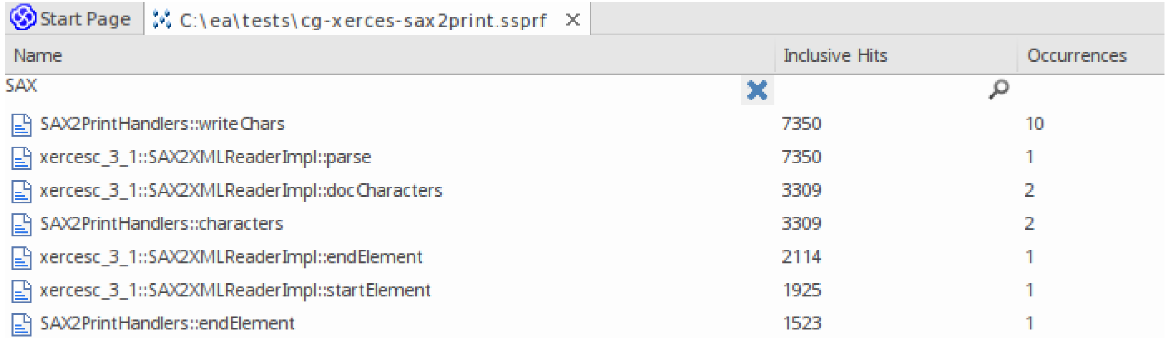
函数总结



Start Page | C:\ea\tests\cg-xerces-sax2print.ssprf

Name	Inclusive Hits	Occurrences
mainCRTStartup	7408	1
__tmainCRTStartup	7407	1
xercesc_3_1::XMLFormatter::handleUnEscapedChars	7351	10
xercesc_3_1::XMLFormatter::formatBuf	7351	10
xercesc_3_1::XMLFormatter::specialFormat	7351	10
SAX2PrintHandlers::writeChars	7350	10
xercesc_3_1::XMLScanner::scanDocument	7350	1
main	7350	1
xercesc_3_1::SAX2XMLReaderImpl::parse	7350	1
xercesc_3_1::XMLScanner::scanDocument	7349	1
xercesc_3_1::IGXMLScanner::scanDocument	7348	1
xercesc_3_1::XMLFormatter::formatBuf	4042	8

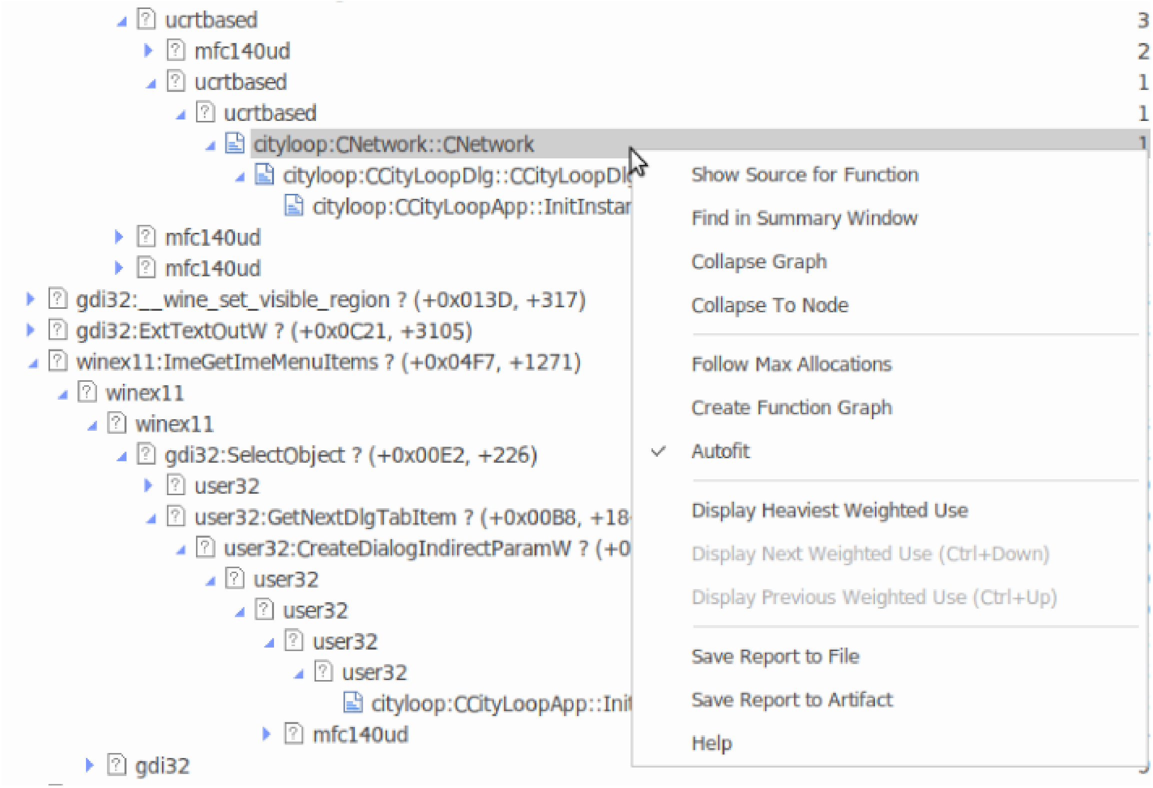
未过滤的摘要报告按包含命中的顺序列出所有参与函数。



您可以过滤和重新组织报告中的信息，方法与对模型搜索结果的操作相同。

报告选项

右键单击报告以显示上下文菜单。



注记列出的选项取决于显示的报告类型；此处所示的报告是一份内存配置报告。

行动	细节
函数显示源	对于选定的框架，选择此选项可在代码编辑器中显示相应的代码行。具有可用源的框架可通过其图标来识别。
在摘要窗口中查找	选择此选项可在 摘要"窗口中找到函数。
折叠图	选择此选项可折叠整个图形，包括可见或不可见的子节点。
折叠到节点	选择此选项可折叠整个图形，然后展开并将聚焦设置为选定节点。

关注最大分配	选择此选项可在图表中展开整条线。
为函数创建行报告	选择此选项以启动 Profiler（如果尚未运行），立即绑定所选函数并准备好进行记录。绑定后，会在当前报告视图中打开一个额外的选项卡。此报告将立即更新，显示每行执行的次数。当然，报告会继续记录函数中的活动，即使是不可见的。 注记： • 在窗口程序中，通常需要在应用程序中执行一些操作来调用函数 • 此选项不适用于内存配置报告
创建函数图	选择此选项可创建一个附加选项卡，该选项卡单独显示所选函数。对于调用频率配置文件，这会生成一个图表，显示导致该函数被调用的所有行（即调用者）。对于内存配置，这会生成一个图表，显示从这个函数（即被调用者）发出的所有行。
将初始帧标记为调用堆栈的图表	使用之前创建一个调用堆栈序列图来限制堆栈长度。选择此选项时，会标记框架并突出显示其文本。高于该帧的帧将从生成的任何序列图中排除。 此选项不适用于内存配置报告。
删除标记	从先前标记为 初始"的帧中删除标记。 此选项不适用于内存配置报告。
创建调用堆栈图表	为图中的单个堆栈生成序列图。所选帧被描述为堆栈中的终端帧。如果没有标记 初始"帧，则堆栈的初始帧默认为根。 此选项不适用于内存配置报告。
创建加权调用图图表	生成一个序列图，为从选定帧分支的每个可见堆栈分支呈现一个序列。通过展开和折叠感兴趣的节点，您可以根据自己的喜好定制序列图内容。 此选项不适用于内存配置报告。
显示最重的加权使用	选择此选项可在图表中显示该函数出现的权重最高的线。
显示下一个加权使用	选择此选项可导航到图表中出现函数的下一行。 您可以使用快捷键组合 Ctrl+向下箭头。
显示上一个加权使用	选择此选项可导航到图表中出现此函数的上一行。 您也可以使用快捷键组合 Ctrl+向上箭头。
导入源代码	选择此选项可将选定的源代码导入报告。 此选项不适用于内存配置报告。
自动调整	启用后，自动将列调整到可用的显示区域。
将报告保存到文件	选择此选项可显示 另存为"对话框，允许您选择存储报告的位置。
保存报告工件	注记浏览器在此选项之前，转到窗口并选择要在其下创建工件元素的包或元素。 系统会提示您提供报告（和元素）的名称；输入这个然后点击确定按钮。 工件是在浏览器中创建的，位于选定的包元素或元素。 如果您将工件添加到图表中作为一个简单的链接，当您双击该元素时，报表将重新打开

注记

- 如果您将 Profiler 报告添加到工件元素并附加链接，则 Profiler 优先并在您双击时元素；您可以使用 '编辑链接文档"上下文菜单选项显示链接文档

在团队图书馆中保存报告

您可以将任何当前报告保存为团队图书馆中某个类别、主题或文档的资源。然后可以随时共享和查看报告，因为它与模型一起保存。这可以帮助您：

- 保留 Profiler 报告以与未来的运行进行比较
- 允许其他人调查个人资料

访问

上下文菜单	在库窗口中单击鼠标右键 |分享资源|活动Profiler Report