ENTERPRISE ARCHITECT

用户指南系列

视觉执行分析

Author: Sparx Systems

Date: 20/06/2023

Version: 16.1

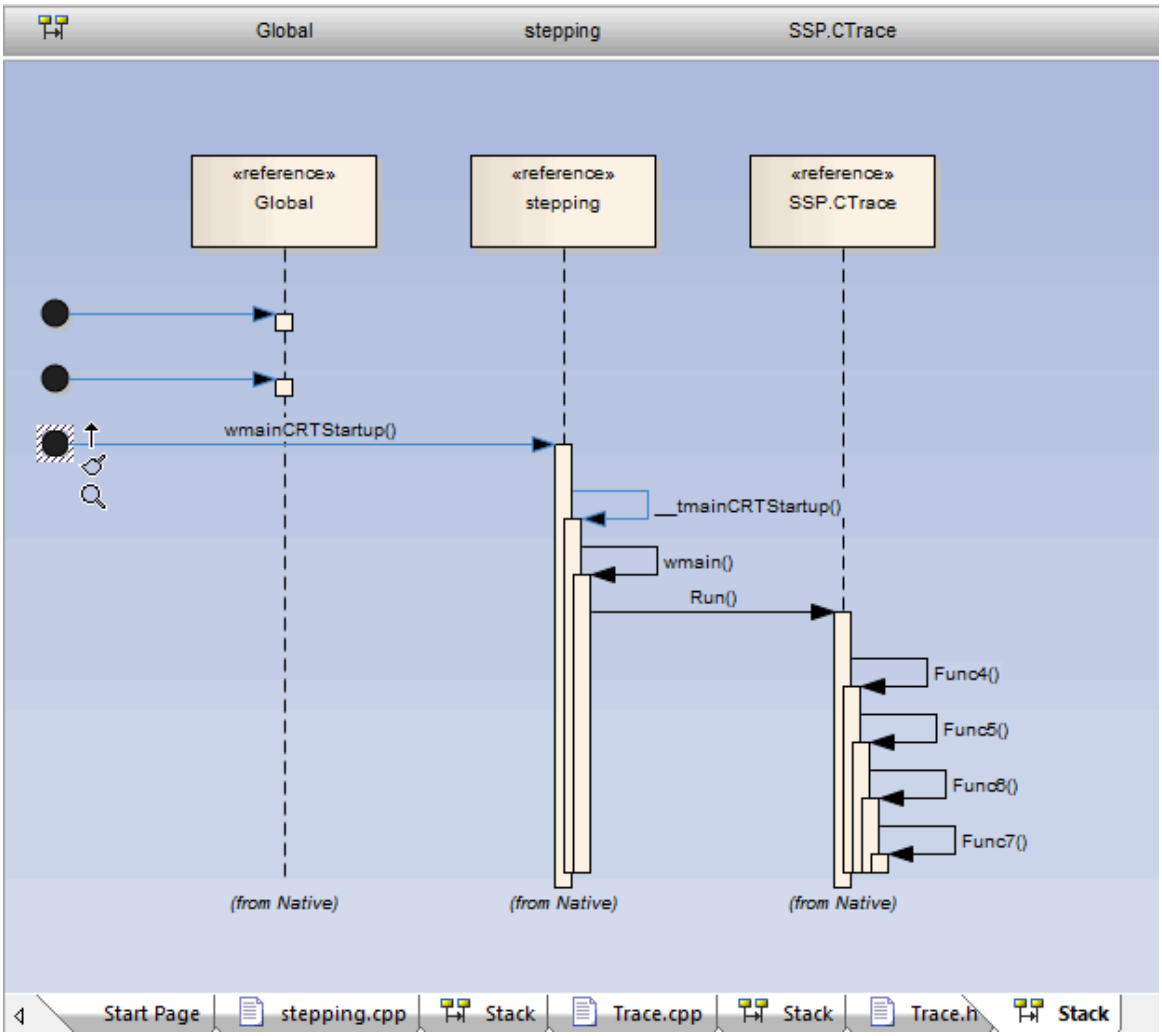
创建于 ENTERPRISE ARCHITECT

目录

Introduction

Enterprise Architect, in addition to its extensive features as a fully fledged Integrated Development Environment, provides tools for visualizing and analyzing software execution that have a profound effect on the way that code is managed, maintained and documented. Much of this capability is unique to Enterprise Architect, which uses its rich visualization and modeling facilities to bring programming code to life, effectively allowing the code to be put under the microscope.

Code hot spots and faults can be analyzed, errors and opportunities for speed improvements can be identified quickly, and resolutions to errors found. These features can be used while building and testing software products, but are also profoundly useful for documenting existing or legacy systems. Commonly there is no documentation for these systems and it is prohibitively expensive or even intractable to get the code documented by conventional means. Enterprise Architect can create sophisticated, elegant and valuable models and documents that completely describe the code base, including expressive Sequence diagrams and static models. This type of documentation is a number of generations more advanced than the simplistic and often opaque documentation generation tools such as JavaDoc.



执行分析

可视化执行分析器(VEA) 由一套先进而全面的工具组成，允许您构建、调试、记录、分析、模拟以及以其他方式构建和验证您的软件开发，同时保持代码与您的模型紧密集成。 Enterprise Architect为广泛的流行编译器和平台提供了丰富的支持，特别是Java 、.Net 和 Microsoft窗口++ 环境。软件开发成为一种高度简化的视觉体验，与在传统环境中工作完全不同。

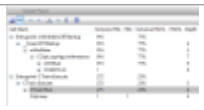
Enterprise Architect本身完全在Enterprise Architect内置的可视化执行分析器中建模、构建、编译、调试、测试、管理、分析和以其他方式构建。虽然 VEA 可用于补充其他工具套件，但当用作主要开发 IDE 时，它与Enterprise Architect提供的模型和项目管理功能紧密结合时也非常出色。

访问

功能区	开发>源代码>执行分析器>编辑分析器脚本 执行 > 工具 >分析器
键盘快捷键	Shift+F12

执行分析器特征

特征	描述
编译和调试 	使用链接到模型包的分析器脚本，可以将代码/构建/调试周期紧密集成到Enterprise Architect中。特别是对于Java 、.Net 和 Microsoft C++，链接到项目代码库并接管Enterprise Architect的模型驱动开发环境中的构建和调试非常简单。除了标准调试之外，与模型的强耦合以及高级调试特征（如行动点）的使用使Enterprise Architect成为设计和构建软件应用特征的理想平台。
仿真 	通过即时、实时的行为模型执行，让您的行为模型栩栩如生。结合用于管理触发器、事件、防护、效果、断点和模拟变量的工具，以及在运行时可视化跟踪执行的能力，模拟器是通过可视化行为模型的执行来 观察车轮转动”的有效手段.
剖析	揭开软件性能的面纱，看看实际发生了什么。快速清楚地了解某些任务为何表现不佳或比预期更差。无论是 Microsoft .NET 、本机 C++ 还是Java ，都可以使用配置文件有效地判断软件生命周期内的性能变化。

	
记录执行 	无需仪器即可记录代码的执行。通过过滤器和堆栈深度细节控件。生成说明类协作的漂亮序列图和图表。使用记录创建可与 VEA 测试点特征一起使用的测试域图。
测试 	创建和管理模型元素的测试脚本。探索测试接口、支撑单元、集成、场景、系统、验收测试。使用具有功能的合同方法进行编程。
物件工作台 	通过在物件工作台中实例化它们然后调用它们的操作，Workbench类行为在运行中。您甚至可以将工作台上的对象作为参数传递给其他工作台对象。
可视化执行分析器样本 	尝试使用我们的示例模式来设置和探索可视化执行分析器中丰富的分析特征集。

执行分析器的执行分析器

执行分析器为多个平台提供集成开发和测试环境，包括 Microsoft .NET 、 Java 、Native C++、Mono 和 Android。它包括功能丰富的调试器、执行记录和分析以及测试点管理。

它可以帮助您从单个记录中生成序列、测试域类和协作类图。这是理解和记录您的应用程序的好方法。

- 可视化程序执行
- 优化现有系统资源并了解资源分配
- 验证系统是否遵循设计的规则
- 生成更准确地反映系统行为的高质量文档
- 了解系统如何以及为何工作
- 培训新员工了解系统的结构和函数
- 全面了解现有代码的工作原理
- 识别代价高昂或不必要的函数调用
- 说明系统内的交互、数据结构和重要关系
- 跟踪特定代码行、系统交互或事件的问题
- 建立在系统故障之前立即发生的事件的序列
- 仿真行为模型的执行，包括状态机、活动和交互

操作

手术	描述
仿真行为	仿真UML行为模型以验证其逻辑和设计的正确性，用于： • 活动 • 相互作用和序列 • 状态机
记录执行	记录执行程序并将行为表示为UML序列图；支持录制： • Microsoft窗口原生 C、C++、Visual Basic • Microsoft .NET系列（C#、J#、VB） • Java • 单核细胞增多症 • 安卓 • PHP
配置文件行为	快速查看/报告正在运行的应用程序的行为。这些平台支持分析： • 微软本机 C、C++、Visual Basic • Microsoft .NET系列（C#、J#、VB）（包括任何非托管/托管代码组合） • Java • 单核细胞增多症
使用系统测试使用案例	功能Management 提供了将类模型的约束定义为合同的功能。合同提供了在其上创建测试域的资产。然后可以A单个测试点域来测试和报告多个应用程序的行为。您还可以使用执行分析器记录一个用例并生成一个测试域图。任何现有的测试点都会自动链接到生成的域，或者测试域图可以上下文新合约组合的时间。可以立即实时查看应用程序在给定测试域中的行为方式！每次合同通过或失败时，结果都会显示在测试点报告窗口中。测试测量与代码库的分离有很多好处，其中之一是帮助多个系统与一个公共测试域协调，而不是相互协调。 测试点系统支持以下合约： • 类不变量 • 方法前置条件 • 方法后置条件 • 线路条件
打开控制台窗口	控制台窗口是一个命令行解释器，您可以通过它快速创建一个终端窗口以启用脚本引擎并输入命令以作用于脚本（JScript、 JavaScript和 VBScript）。
物件工作台	使用动态物件工作台创建和处理在Enterprise Architect建模环境中创建的对象。 • 从类模型创建对象 • 调用方法并查看结果 • 工作台类协作 • 将对象作为参数传递给其他对象 • 完整的调试特征，包括记录
运行测试	为Java和 Microsoft .NET运行和 jUnit 测试 记录并记录结果。

从分析导入代码	Execution记录和 Profiling 都获取了相关代码文件的集合，您可以在一次操作中对当前模型进行逆向工程。

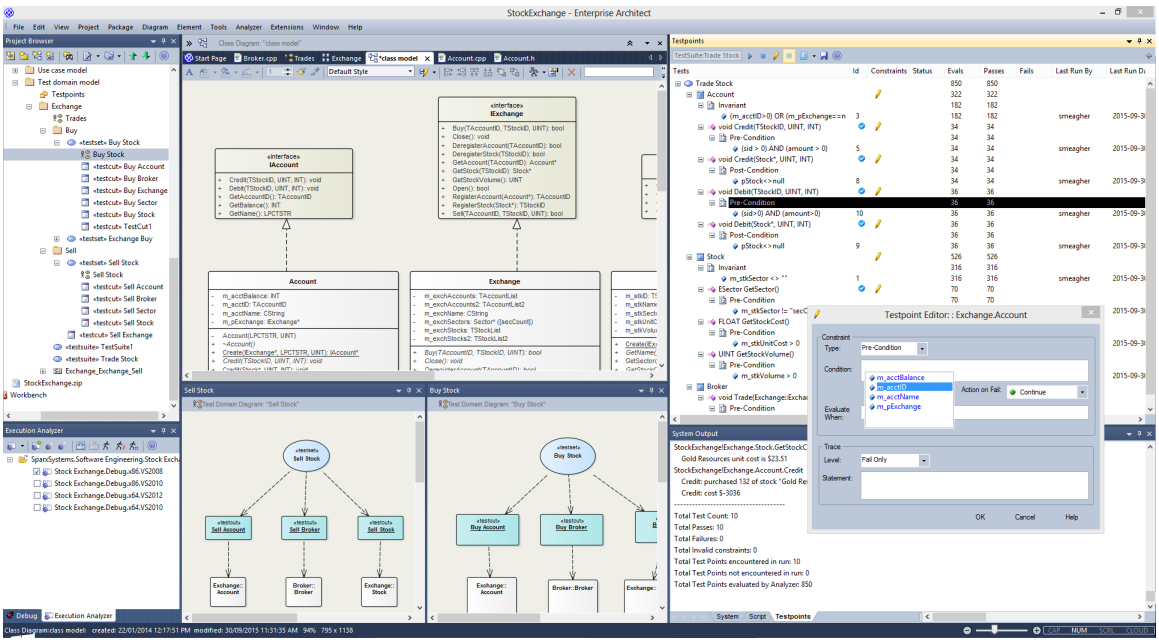
可视化运行状态

通过对object生命周期关键点的运行状态进行多次快照，可以记录单个object的状态转变。为此，只需将局部变量或成员变量拖到物件图上即可。

测试点

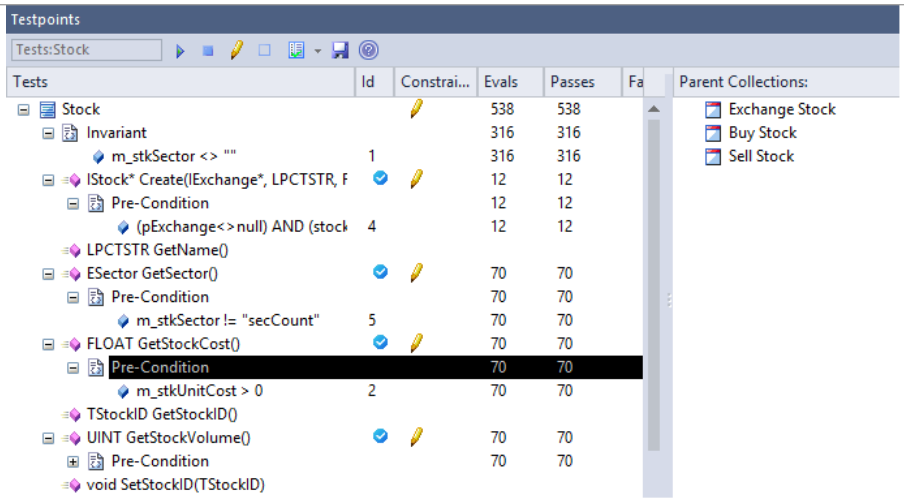
测试点提供了一种方案，通过该方案可以从模型中获取控制对象行为的约束和规则，并将其应用于一个或多个应用程序。像这样的方案提供的优点是对代码更改的容忍度——从函数中添加和减去行对管理它的约束没有影响。另一个优点是行为规则的改变不需要对源代码进行相应的改变；意味着什么都不需要重新编译！

此外，使用单个测试域验证多个应用程序的能力是一件简单的事情，而不是繁重的事情。测试域既是逻辑模型又是关系模型；类中的约束可以用测试截口模型口进行划分。这些可以使用连接器简单地聚合到测试集和测试套件中。由于测试域与代码库的解耦，可以简单地选择按钮来正常运行程序，或者运行特定的测试域运行它。该系统还提供了实际的好处，因为根本不需要任何仪器。在运行期间，测试结果会在程序运行时实时显示在报告窗口中。这些结果可以保留，并随时在 测试细节"对话框中或使用Enterprise Architect的文档特征进行审查。



特征

特征	细节
测试点组合	测试点组合是使用 Testpoints 窗口执行的。 Testpoints 窗口是上下文相关的，并在浏览器窗口或图表中显示所选元素的测试域。选择单个类将显示类结构。对具有现有约束的类和方法显示A 铅笔"图标。 当您选择测试截口、Set 或 Suite测试时，Testpoints 窗口会显示整个域结构，包括构成域的所有类。注记：您可以使用右侧的 导航"窗格导航域层次结构.测试点使用类成员的变量名组成表达式。智能感知快捷方式 Ctrl+Space 可在编辑器中找到这些内容。计算结果为True的表达式被视为通过。返回False意味着失败。



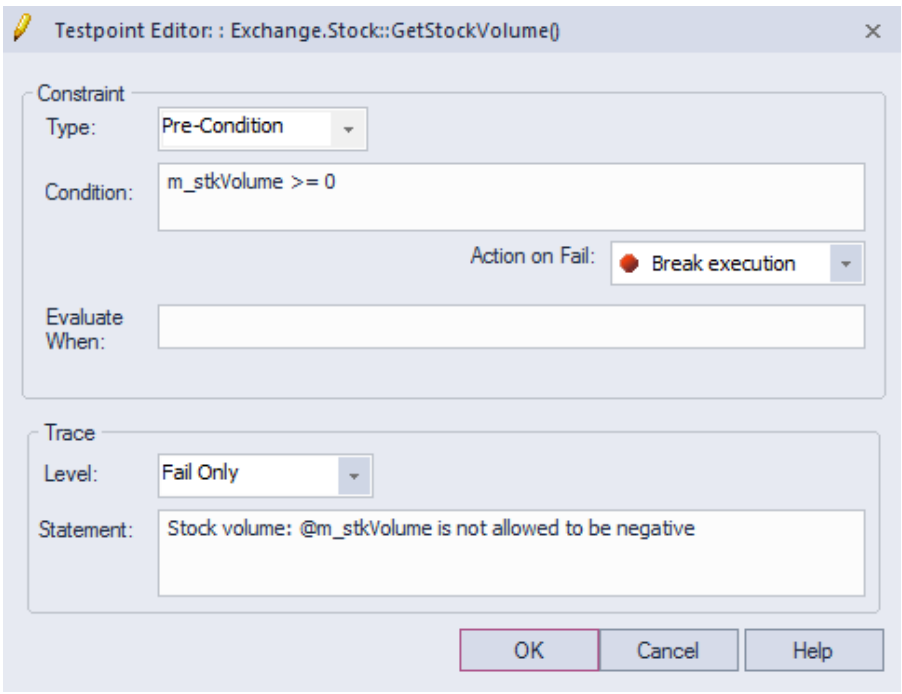
您可以通过双击类来添加或编辑现有的不变量。

您可以通过双击方法类似地添加或编辑现有的前置条件或后置条件。

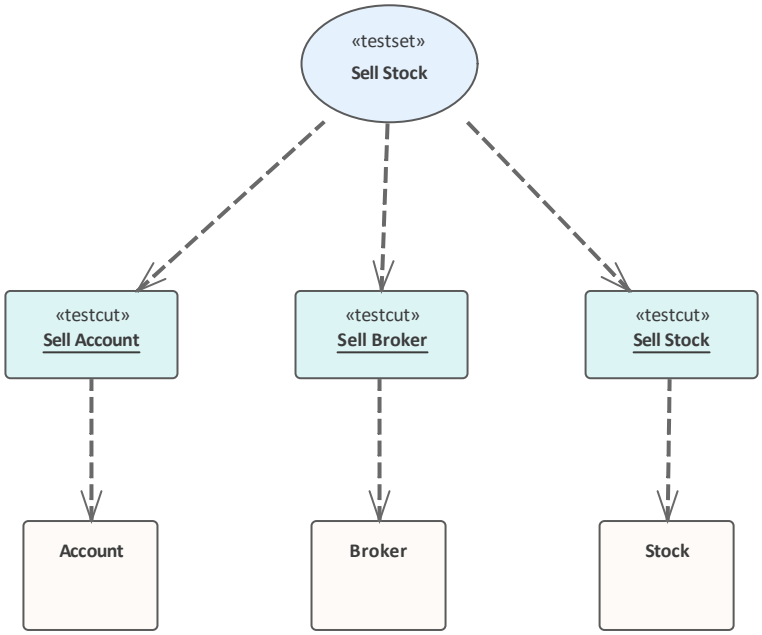
如果有源代码，双击测试点会自动显示源代码。

最好使用代码编辑器的快捷菜单从代码编辑器中添加行条件。

此图像是测试域中的先决条件。

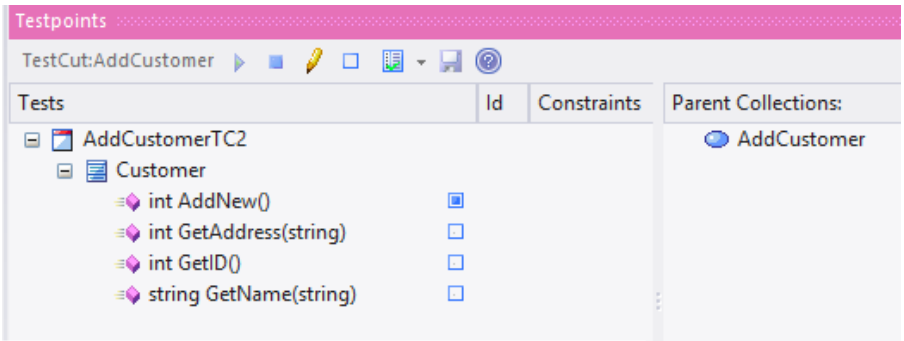


测试点跟踪陈述	每个测试点都可以有自己的跟踪语句。跟踪语句是一个动态消息，可以在其object或本地范围内引用变量。它们在测试评估期间输出。可以将它们配置为在每次评估约束时输出，或者更常见的是在测试失败时输出。可以将跟踪语句定向到系统输出窗口的 测试点”选项卡或外部文件。你可以在任何分析器脚本配置它。
测试域组合	测试域图是一个动态媒介，其中测试点被组装以测试使用案例。测试域图中的使用案例以三种不同的原型提供：测试截口、测试集和测试套件。域的管理就像在任何图表上建模一样简单。工具箱快捷菜单提供对任何测试域和工件的访问。简而言之，来自多个类的测试点被聚合成测试集。然后将测试集链接到形成测试套件。测试截口和测试集都是可重复使用的资产。将同一个测试集链接到一个或多个测试套件是绘制连接器的问题。



测试域和类模型

一个用例很少会涉及单个类的所有方法。它很可能是使用协作类中的各种方法来实现的。我们称此方法为 剪切口"的子工件，是我们用来进行测试截口的工具。测试点窗口将根据时间进行上下文，以适应测试域领域或类元素。此图显示了选择测试截口时的测试截口窗口。请注记复选框，这些复选框仅对测试截口可见。它们表示对测试集有贡献的方法（测试截口）。在这个例子中，测试域是由执行分析器生成的，它为我们完成了方法识别工作。



测试点评估

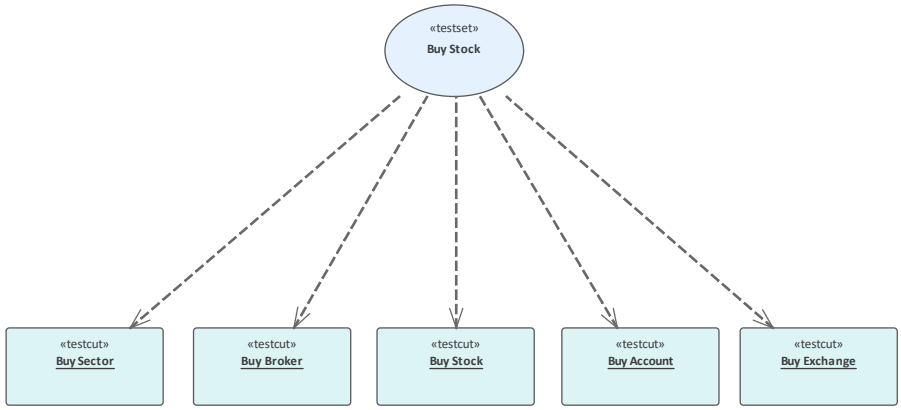
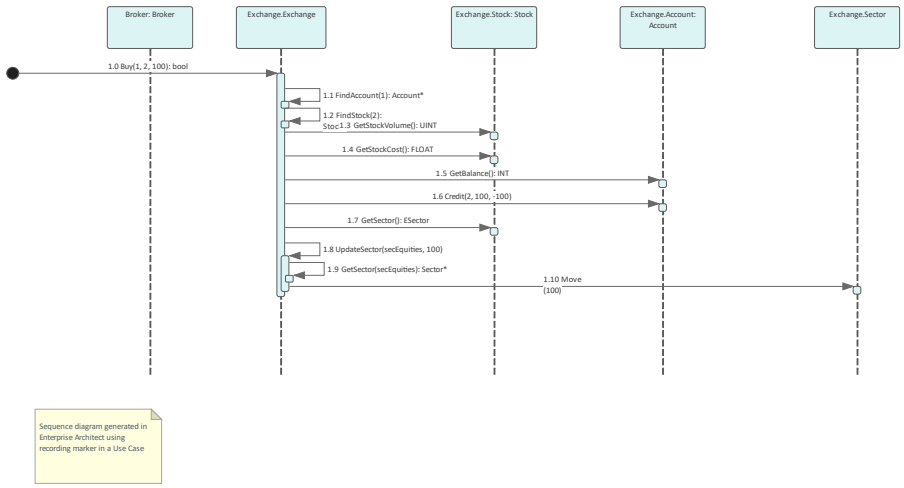
Testpoints 窗口用于评估测试域。该窗口有一个用于启动或附加到目标应用程序的工具栏。要测试的域总是由具有时间的元素上下文，因此如果您选择一个类，则窗口将仅显示该类的类结构和测试类。如果您选择一个测试套件，该窗口将显示整个域层次结构以及其中包含的所有测试点。单击运行按钮将加载执行分析器中的测试点域，然后在使用案例通过或失败时评估、收集和更新报告窗口。每个约束类型的确切细节以及该约束捕获的时间和方式是：

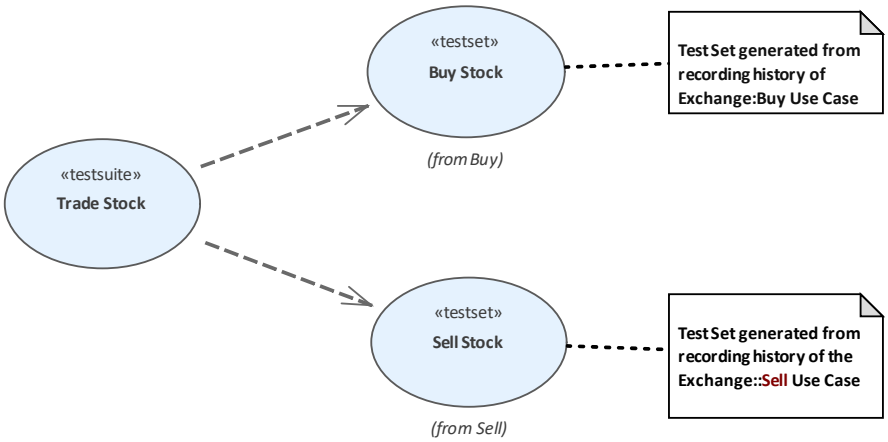
- 类对类类型的object调用的任何方法完成时，分析器都会评估A不变量；不变量用于测试符合object的状态是否已知且允许
- 在调用操作之前立即评估前置条件
- 方法完成时评估后置条件（同时作为类不变量）
- 如果以及何时在程序执行期间它们的特定代码行进入范围，则评估行条件

测试域图表

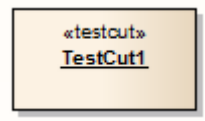
测试域图是您为特定领域组装和分组测试用例的媒介。测试域的一个示例可能是 顾客”。您组装的域的广度和深度取决于您。您可能为 添加顾客”和 删除顾客”设置了不同的域，这完全取决于您认为平衡域层次结构的最佳方式。工具图表快捷菜单提供了一个数字工具箱工件模型该领域。因为媒体是动态的，允许您重新访问和构建测试域之间的关系，所以该系统是一个很好的模型，用于向组织交付可重用资产，其开销低，并与UML世界视图和软件工程集成日常生活的基本要素。

功能

功能	细节
测试域生成	如果您认为编写测试域的过程很复杂，可以，但帮助就在眼前！执行分析器可以为您生成测试域图。它不能为你编写测试，但它可以做一些腿部工作。它可以识别类并仅挑选出参与用例的那些方法。这不是猜测。分析器测试域是从一个正在运行的程序中获得的。此图显示了执行分析器通过记录示例模型程序生成的测试域。  这就是生成测试域的记录本身（作为序列图）。 
测试域组合	测试域图的第一个任务是创建使用案例（测试集）。这些定义了这个特定域的职责。图表和快捷菜单提供工具箱工件您实现这一目标。这些元素中的第一个是测试截口，用于下一步；从类模型中识别出您认为参与使用示例的用例方法。测试截口工件测试，因为它允许划分一个类，选择那些相关的方法。测试截口可以运行或链接一个或多个。测试集反过来可以链接到一个或设置更多测试套件。在任何情况下，测试域树的任何元素都可以运行或作为

	一个整体运行。  «testset» Buy Stock Test Set generated from recording history of Exchange:Buy Use Case «testsuite» Trade Stock (from Buy) «testset» Sell Stock Test Set generated from recording history of the Exchange::Sell Use Case (from Sell)

测试截口



描述

测试截口元素是A原型的物件元素，用于Enterprise Architect内部，用于使用测试点代码测试功能定义测试集。

诸如 打印”之类A任务可能涉及对不同类的操作。为了创建 打印”测试，您可能只想包含这些类的 打印”操作并排除任何其他操作。

测试截口使您能够仅捕获表示为单个类定义的行为（在本例中为 打印”） A操作。然后，您可以将几个类中的每个类中的测试截口作为测试集放入一个任务中。

当您将测试截口元素拖到测试域图上时，您将创建一个与所需类元素的依赖关系。因此，当您在 Testpoints 窗口中选择测试截口元素时，该类的操作会在窗口中列出，每个类都带有一个复选框。然后选中每个类操作的复选框以包含在测试截口中。

工具箱icon

测试集

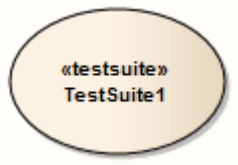
描述

测试集元素是A典型的用例元素，用于将可能跨越多个类的一组或多组方法（测试截口）聚合到单个任务中。测试集也可以聚合到测试套件中。

您使用依赖连接器将测试截口元素链接到测试集。

工具箱icon

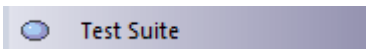
测试套件



描述

测试套件元素是A原型用例元素，用于聚合一组或多组任务（测试集）。
您使用依赖连接器将测试集元素链接到测试套件套件。

工具箱icon



测试点窗口

测试点窗口是组成测试域约束的中心。它也是让您验证程序上的特定测试域的控件。该程序可能已经在运行，或者可以使用控件的工具栏启动。在这里，您还可以实时查看测试结果。此控件是上下文相关的，响应浏览器窗口或图表中元素的选择。根据选择，可以对单个类、一个用例（测试集）或一组使用案例（ A测试套件）进行测试。

访问

功能区	执行 > 工具 > 测试器 > 显示测试点窗口

测试点窗口列

柱子	用途
测试	显示选定测试点object的名称及其下对象的层次结构。 选定的object可以是： • 类 • 手术 • 测试截口 • 测试设置或 • 测试套件
ID	对于一个操作，当分析器在目标应用程序中成功绑定该操作时，该列显示一个测试点标记图标（ ）。如果运行过程中该栏没有出现运行，则表示模型和代码库可能没有同步；也许函数的签名已经改变，或者操作是你正在处理的新方法，它存在于源代码中但尚未出现在你的模型中。 对于测试点，此列显示生成的 ID 号。此 ID 号用于跟踪输出中以指示正在引用哪个约束。
约束	此列中A铅笔图标（ ）表示为类或操作定义了一个或多个约束。
状态	在测试运行期间，指示这些可能的状态： • （ ）失败 -约束被评估为false一次或多次。 • （ ）Invalid Statement - 由于语法无效，约束解析失败。 • （ ）未找到变量 - A评估约束的位置未找到引用的变量名称。 如果约束已通过，则不显示任何图标。
评估	在测试运行期间，指示执行分析器评估此约束的次数。
通行证	在测试运行期间，表示测试通过的次数。
失败	在测试运行期间，指示测试失败的次数。

最后运行	显示最后一个运行此测试的人的用户名。 （值来自 “人物”对话框中的项目作者定义 - 设置 >参考>模型类型 > 人物 > 项目作者”。 ）
上次运行日期	显示上次评估此测试的日期和时间。
上次运行结果	显示上次测试运行的结果。
父集合窗格	列出包含选定object作为其设计的一部分的所有父集合。 双击此集合使其成为左窗格中的选定object 。 通过单击测试点窗口工具栏上的显示/隐藏父集合窗格按钮，可以隐藏父集合窗格。

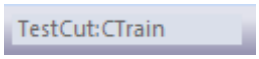
测试点工具栏

测试点窗口工具栏提供了对当前选择的测试点object执行配置的测试、停止当前正在进行的测试运行、过滤显示的项目以及保存已完成测试运行。

访问

功能区	执行 > 工具 > 测试器 > 显示测试点窗口

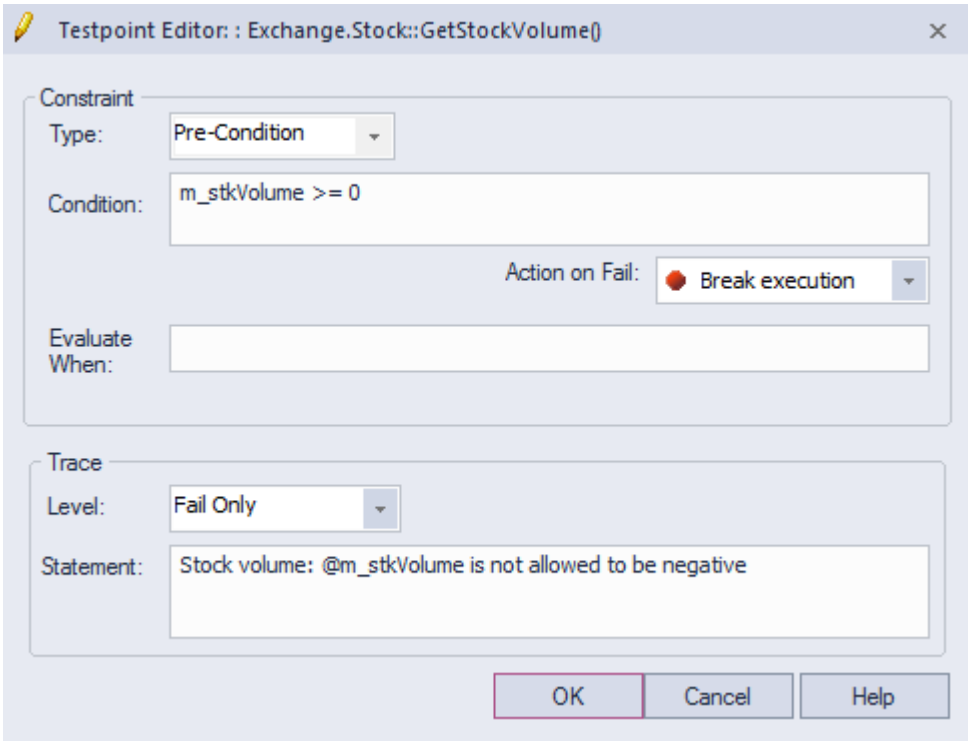
测试点工具栏选项

工具栏按钮	行动
	字段当前选中的测试点object的名称。
	执行测试运行。
	运行当前正在进行的测试。
	在显示所有项目和仅显示已定义约束的项目之间切换。
	在显示所有项目和仅显示已标记为包含在此测试截口中的操作之间切换；此按钮仅在选择测试截口object时启用。 当一个测试截口被选中时，其关联类的每一个操作都会显示一个复选框；您使用此复选框来标记适用于此测试截口的操作。
	单击此图标旁边的下拉箭头以显示 测试运行选项”菜单，提供以下选项： • '前缀跟踪输出With函数调用' -前缀所有带有执行函数名的输出行 • 'Enable Standard断点during testing' - 未勾选时，测试运行忽略当前断点集中的任何断点，并且在运行期间设置运行的任何尝试都将被忽略 • '视图跟踪输出' - 显示系统输出窗口的'测试点'选项卡
	测试运行完成后点击此图标可将结果保存到当前object的测试项中。可以使用测试工作空间查看保存的测试。 显示A提示以选择测试类- Unit、集成、系统、Inspection、Acceptance 或 Scenario。选择合适的测试类并点击确定按钮。
	显示测试点管理帮助主题。
	显示或隐藏父集合窗格。

测试点编辑

测试点编辑器用于构成类和操作的约束。允许的约束类型取决于所选object 。对于类，类型总是不变的。对于操作，类型可以是前置条件、后置条件或行条件。

当对选定类类型的object调用的任何方法完成时，分析器会评估不变量。每次调用指定操作开始时都会评估前置条件。在完成对指定操作的每次调用后评估后置条件。每次执行指定的代码行时都会评估行条件。



访问

功能区	1. 执行 > 工具 > 测试器 > 显示测试点窗口。 2. 在 Testpoints 窗口中，双击类或 Operation 以显示 测试点编辑器”对话框。

约束组字段

字段	用途
类型	所选类或操作的约束类型： • 不变 - 在指定类上调用的任何方法完成后评估 • 前置条件 - 在每次调用特定操作开始时进行评估 • 后置条件 - 在完成对特定操作的每次调用后评估 • 行条件 - 在操作中执行特定代码行时评估
抵消	仅适用于 Line-Conditions，指定操作中的行号，用于评估约束。

	如果测试点是使用代码编辑器上下文菜单创建的，则会自动设置偏移值。
条件	触发该测试点时要评估的约束。 A该约束条件的计算结果为true or false ，将记录通过或失败的状态。
行动失败	单击下拉箭头并从三个选项中进行选择： • 'Continue' - 忽略此约束的失败并继续执行 • 'Break execution' - 停止执行并显示堆栈跟踪 • 'Disable on fail' - 失败一次后不再执行约束
评估时间	（可选）在评估主要测试点条件之前必须满足的附加约束，从而更好地控制测试覆盖率。

跟踪组字段

选项	行动
等级	指定何时输出跟踪语句（如果已定义）。可用选项有： • 'Fail Only' - 仅在此测试点条件失败时输出跟踪语句 • 'Always' - 每次评估此测试点时输出跟踪语句
陈述	（可选）评估此测试点时要输出A消息。 当前范围内的变量可以包含在跟踪语句输出中，方法是在变量名称前加上 $ 标记（用于string变量）或 @ 标记（用于原始类型，例如 "int "或 "long"）。 监视状态的输出可以被定向到系统输出窗口的 测试点"选项卡，也可以定向到由跟踪分析器脚本包的外部文件。

测试点约束

约束通常由表达式中A局部变量和成员变量组成，由运算符分隔以定义必须满足的一个或多个特定标准。约束A评估为真才能被视为通过。如果约束评估为false ，则将其视为失败。

约束内引用的任何变量都必须在评估测试点或断点的位置的范围内。

一般/算法算子

操作员	描述
+	添加 示例：a + b > 0
-	减去 示例：a - b > 0
/	划分 示例：a / b == 2
*	乘 示例：a * b == c
%	模数 示例：a % 2 == 1
()	括号 - 用于定义复杂表达式中的优先级。 示例： ((a / b) * c) <= 100
[ ]	方括号 - 用于访问数组。 示例：Names[0].Surname == "Smith"
.	点运算符 - 用于访问类的成员变量。 示例：示例== "名称"
->	选择点运算符的表示法。 示例：站->名称== 弗林德斯”

比较运算符

操作员	描述
=	等于 示例：a = b

==	等于 示例：a == b
!=	不等于 示例：a != b
<>	不等于 示例：a <> b
&gt;	比...更棒 示例：a > b
>=	大于或等于 示例：a >= b
&lt;	少于 示例：a < b
<=	小于或等于 示例：a <= b

逻辑运算符

操作员	描述
和	逻辑与 示例： (a >= 1) AND (a <= 10)
或者	逻辑或 示例： (a == 1) 或 (b == 1)

位运算符

操作员	描述
&amp;	按位与 示例： (1 & 1) = 1 (1 & 0) = 0
|	按位或 示例： (1 | 1) = 1 (1 | 0) = 1

^	按位异或（异或） 示例： (1 ^ 1) = 0 (1 ^ 0) = 1

其他示例

示例	描述
((m_nValue & 0xFFFF0000) == 0)	使用十六进制值作为右操作数的位与运算符 (&) 来测试在变量的高位字节中没有设置任何位。
((m_nValue & 0x0000FFFF) == 0)	使用以十六进制值作为右操作数的位与运算符 (&) 来测试在变量的低位字节中没有设置任何位。
m_value[0][1] = 2	访问多维数组
a 与 (b 或 c)	结合 AND 和 OR 运算符，使用括号确保优先级。在此示例中，变量 “a”必须为真，“b”或 “c”必须为真。

注记

- 字符串比较区分大小写

单元测试

Enterprise Architect支持与单元测试工具的集成，以便更轻松地开发高质量的软件。

序列：

- 您下载并安装 NUnit 和 JUnit 应用程序（ NUnit - http://www.nunit.org/ JUnit - http://www.junit.org/ ）；Enterprise Architect不在安装程序中包含这些应用程序
- Enterprise Architect帮助您使用 NUnit 和 JUnit 转换创建测试类存根
- 您在类存根中定义您的测试代码
- 您针对任何包设置并运行测试脚本
- 所有测试结果都自动记录在Enterprise Architect中

设置单元测试

本主题说明在下载并安装 JUnit 和/或 NUnit 应用程序后设置 Unit测试时应采取的操作。

行动

行动	细节

创建单元测试存根	通过使用 JUnit 或 NUnit 转换和代码生成，您可以为每个类中的所有公共方法创建测试方法存根。
	
	（测试夹具）
	公共类CalculatorTest
	{
	(测试)
	公共void测试添加（ ）{
	}
	(测试)
	公共void testDivide(){
	}
	(测试)
	公共void testMultiply(){
	}
	(测试)
	公共void测试减法（ ）{
	}
	}

定义测试	在生成的代码存根中编写单元测试（在Enterprise Architect或您首选的 IDE 中）。
	这是 C# 中的 NUnit 示例，尽管它也可以是任何其他.NET语言或Java和JUnit。
	
	（测试夹具）
	公共类CalculatorTest
	{
	(测试)
	公共void测试添加（ ）{
	断言.AreEqual(1+1,2);
	}
	(测试)
	公共void testDivide(){
	断言.AreEqual(2/2, 1);
	}

	(测试)
	公共void testMultiply(){
	断言.AreEqual(1*1, 1);
	}
	(测试)
	公共void测试减法（ ）{
	断言.AreEqual(1-1, 1);
	}
	}
	
	或者，如果您没有执行 xUnit 转换，您可以将代码逆向工程到Enterprise Architect ，以便系统可以记录所有针对类的测试结果。

编译你的代码	选择被测试的源代码编译没有错误，以便测试脚本可以运行。
设置测试脚本	针对所需的包设置测试脚本，然后运行测试。

运行单元测试

在运行测试脚本时，您生成的测试结果存储为针对正在测试的类的测试用例。

访问

功能区	执行>运行>开始>测试
执行分析器窗口	ToolBar >运行测试脚本 上下文菜单>测试

任务

任务	细节
运行测试	在浏览器窗口中选择合适的包。 在执行分析器中选择 迩行测试脚本”选项来运行你之前为那个包设置的测试脚本。
视图结果	xUnit 测试的结果显示在系统输出窗口中，识别哪些测试已经运行，哪些测试失败。 结果还显示了失败的方法，以及失败发生的文件和行号。 双击错误消息； Enterprise Architect打开该代码行的编辑器，使您能够快速找到并修复错误。 Enterprise Architect还针对正在测试的类记录每个测试的运行状态；这些存储在元素测试中。 A在图表元素上显示测试脚本隔间，可以设置包含类的图表以显示这些测试用例。

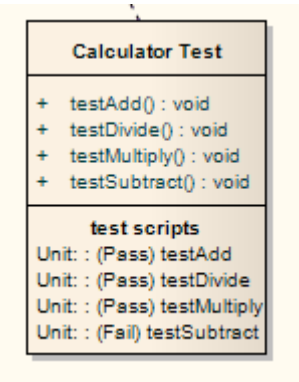
记录测试结果

Enterprise Architect能够通过Enterprise Architect中的测试脚本自动记录所有测试结果。

进程

为了使用这个特征，你必须将测试类逆向工程到包含你的测试脚本的包中。

一旦您的模型包含您的测试类，在下一次运行测试脚本时， Enterprise Architect会为类的每个测试方法添加测试案例；在此和所有后续测试运行中，所有测试用例都使用当前运行时间以及它们是否通过或失败进行更新，如下所示：



每个失败测试的错误描述与当前日期和时间一起添加到该测试用例的任何现有结果中。

随着时间的推移，这提供了每个测试用例失败的所有测试运行的log ，然后可以将其包含在生成的文档中，类似于：

失败于 2006 年 7 月 5 日1 1:02:08

预期：<0>

但是是：<1>

2006 年 6 月 28 日上午 8:45:36 失败

预期：<0>

但是是：<2>

物件工作台

物件工作台是一个Enterprise Architect调试工具，可帮助您从类模型中创建对象。 Workbench 允许任何类的多个实例在同一个会话中共存。每个物件都可以作为您要调用的方法的目标。它们还可以作为参数参与您调用的方法。 Java和 Microsoft .NET平台支持物件工作台。

工作台任务

任务
提供使用物件工作台的指南和要求。
解释什么是 Workbench 对象，以及如何创建它们。
解释如何在 Workbench物件上执行方法并提供有关传递参数的信息。
解释使用调试器逐步执行方法。
解释如何记录方法和生成序列图。
说明如何在完成后删除物件。
说明如何关闭调试器并在完成工作台后关闭它。

使用工作台

使用物件工作台很简单。从您的类模型中，选择要工作台的类并将它们单独拖到工作台窗口上。如果存在多个构造函数，您可能必须选择一个构造函数，然后简单地为变量命名。物件工作台准备所需的运行时，加载任何所需的模块并为您实例化对象。执行方法是从列表中选择的问题。可以在需要的地方输入参数。 Workbench 对象本身可以单独用作参数，也可以用作object数组。

访问

功能区	执行 > 工具 > 测试器 > 打开物件工作台

分析器需求脚本

需要一个已经配置调试的分析器脚本它应该指定以下信息：

- 与您的项目匹配的调试器
- 对于 Microsoft .NET ，将由物件工作台托管的程序集的位置
- 对于Java ，JDK 的位置和要使用的其他类路径

检查清单

- 选择所需的 Workbench类并按 F12；源代码应显示在代码编辑器中
- 按 Shift+F12 构建项目；构建的输出应该显示成功编译

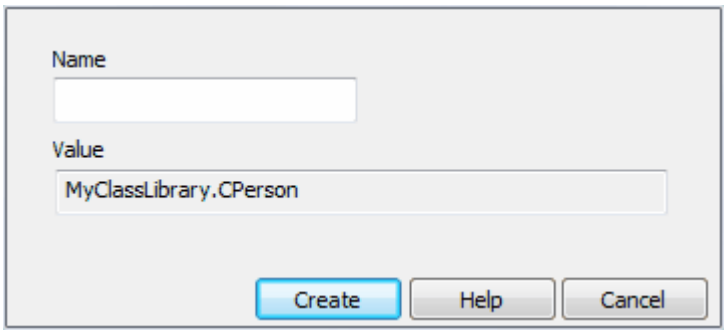
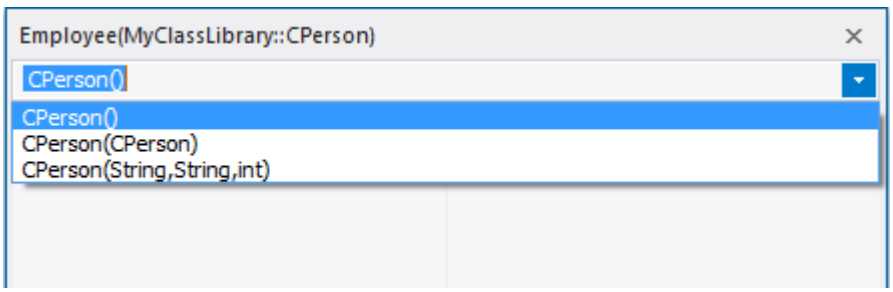
创建对象

本主题说明如何为您的模型中的类创建工作台实例。

访问

功能区	执行 > 工具 > 测试器 > 打开物件工作台
键盘快捷键	Ctrl+Shift+J
其它	将浏览器窗口中的类直接拖到工作台窗口中

任务

任务	细节
在工作台上创建一个物件	选择浏览器窗口中的类并将其拖到 Workbench 窗口中。 将显示 "工作台"对话框。  类型在新实例的名称中。 Workbench 的名称应该是唯一的。 单击创建按钮。
选择构造函数	'构造器' 对话框显示在构造器选项存在的地方。  从下拉列表中选择构造函数。
输入参数	为所选构造函数的参数提供值： • 字符串作为参数 - 在适当的地方用引号括起值，或者值与 Workbench object的名称冲突

	• 对象作为参数 - 输入 Workbench object的名称 • 字符串数组参数采用逗号分隔的文本值： 一，二，三， “一本书”， “一本更大的书” • 物件数组作为参数采用逗号分隔的object名称；提供以逗号分隔的命名 Workbench 对象，例如： 汤姆、迪克、哈利
调用构造函数	单击 Invoke 按钮以创建实例。该object可以通过其在 Workbench 窗口中的名称来识别。

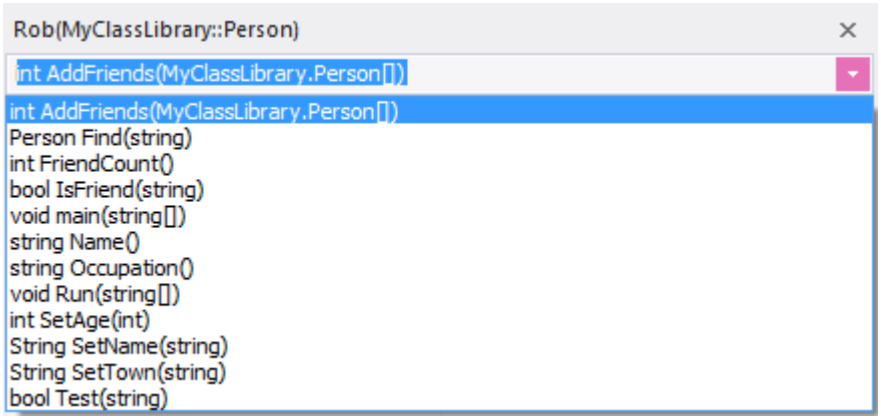
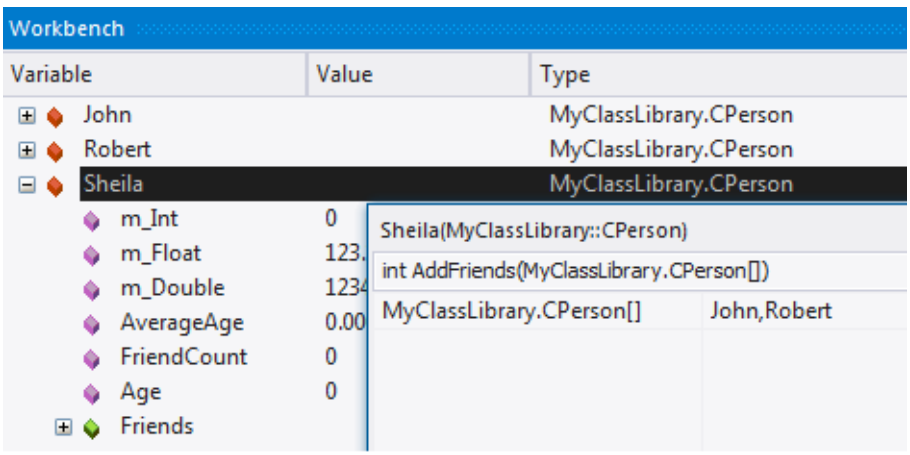
调用方法

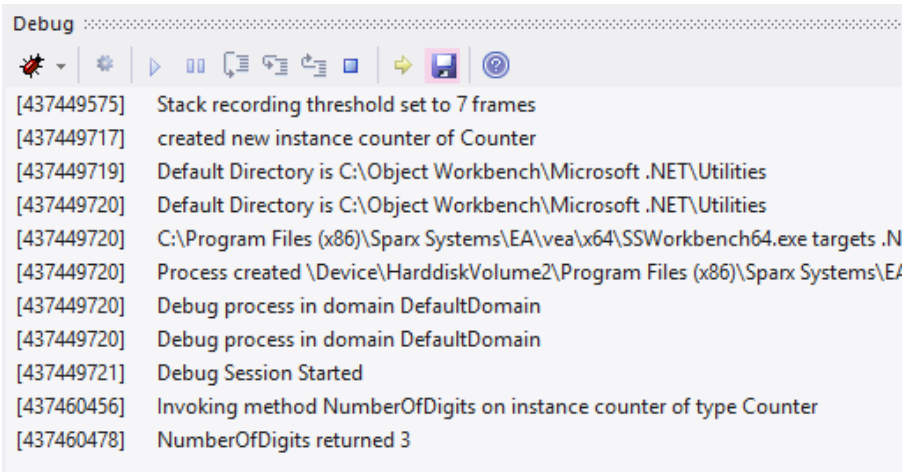
访问

在 Workbench 窗口中，右键单击要在其上执行方法的实例，然后选择 Ḯnvoke″。

功能区	执行 > 工具 > 测试器 > 打开物件工作台

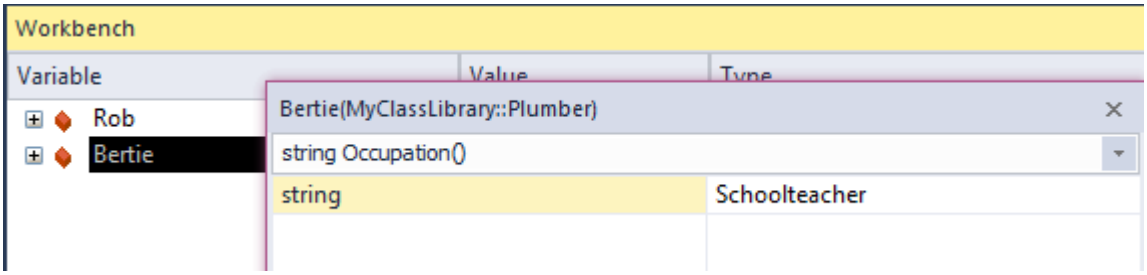
行动

行动	细节
选择方法	从列表中选择一个方法，然后单击 Invoke 按钮。 注记列出的所有方法都是公开的；私有方法不可用。 
提供参数	在此图像中，要调用的方法将对象数组作为其唯一参数。您可以通过命名 Workbench 上要传递给方法的其他实例来构造此参数。 
参数值类型	这些是 Workbench 支持的参数类型： • 字符串 • 数字

	• 对象 • 字符串数组 • 物件数组
参数值语法	• 字符串作为参数 - 必要时用引号将字符串括起来；例如，为了避免与 Workbench object名称冲突 • 字符串Arrays as arguments - 输入组成数组的元素，用逗号分隔；例如： A数学书》、A地理书》、A计算机书》 • Objects as arguments -类型将 Workbench object名称作为参数；调试器根据其 Workbench 实例列表检查在参数中输入的任何名称，并将在实际调用该方法时替换该实例 • 物件数组作为参数 - 输入工作台对象的名称以满足参数，用逗号分隔： 汤姆、约翰、彼得
调用	单击 调用"按钮以执行该方法。 调试窗口中显示确认此操作的输出。 

设置属性

对于支持属性的语言，我们可以像调用方法一样设置物件的属性值。在 Workbench 中选择实例，然后使用其上下文菜单选择 调用"选项。您会发现按字母顺序列出的类公开的属性及其方法。系统将提示您提供属性的新值。类型该值与您在方法调用中为参数输入的值相同。此图演示了更改名为*Bertie*的*Person*的职业属性；*Bertie*是*Person*的一种。



调试和工作台

当您在 Workbench 中工作时，您可能想要调试您正在开发或研究的一种或多种方法。这很容易实现。 Enterprise Architect的执行分析器的相同特征可供物件工作台的用户使用。调试可以在object构造和销毁期间以及方法执行期间执行。要访问调试器，只需在单步执行代码的点放置一个断点。您还可以将这些断点的条件设置为仅在某些条件下中断。

```
20          // The NumberOfDigits static method calculates the number of
21          public int NumberOfDigits(string theString)
22          {
23              int count = 0;
24              for ( int i = 0; i < theString.Length; i++ )
25              {
26                  if ( Char.IsDigit(theString[i]) )
27                  {
28                      count++;
29                  }
30              }
31              m_nResult = count;
32              return m_nResult;
33          }
34      }
35  }
```

调试时，使用调试器控件检查对象的状态。在这里，我们使用本地窗口窗口在执行停止时检查object的状态。

Locals

Variable	Value	Type
this		Utilities.Counter
m_nResult	3	int
theString	"123"	String
i	0	int
CS$1$0000	0	int
CS$4$0001	false	Boolean
count	0	int

当程序恢复时，物件上的物件将反映状态的任何变化。

Workbench

Variable	Value	Type
counter		Utilities.Counter
m_nResult	9	int

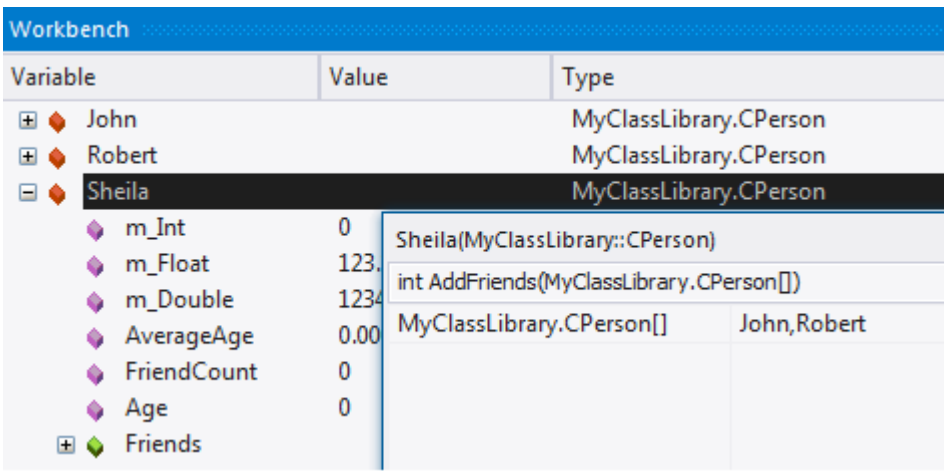
记录和工作台

当您在 Workbench 中工作时，您可能希望为您正在开发或研究的一种或多种方法生成序列图。这很容易实现。Enterprise Architect的执行分析器的相同特征在物件工作台中可用。您甚至可以通过首先记录序列图来开始 Workbench 会话，作为可视化您计划工作的一种方式。

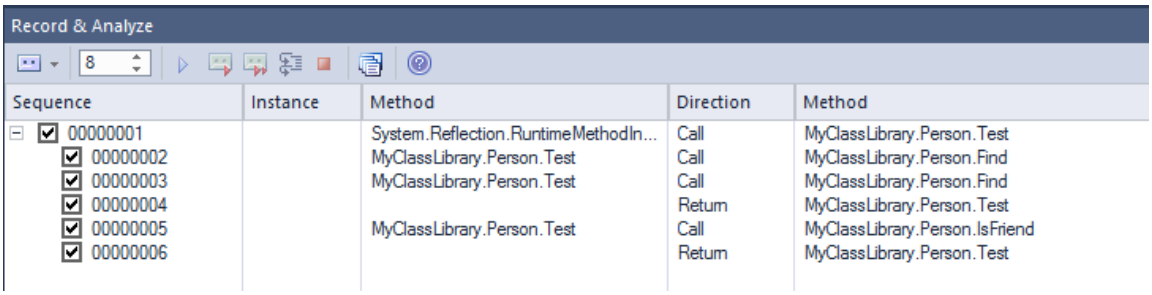
设置记录标记

```
134
135     public bool Test(string name)
136     {
137         Person px = Find(name);
138         if(px != null)
139         {
140             return true;
141         }
142         return IsFriend(name);
143     }
```

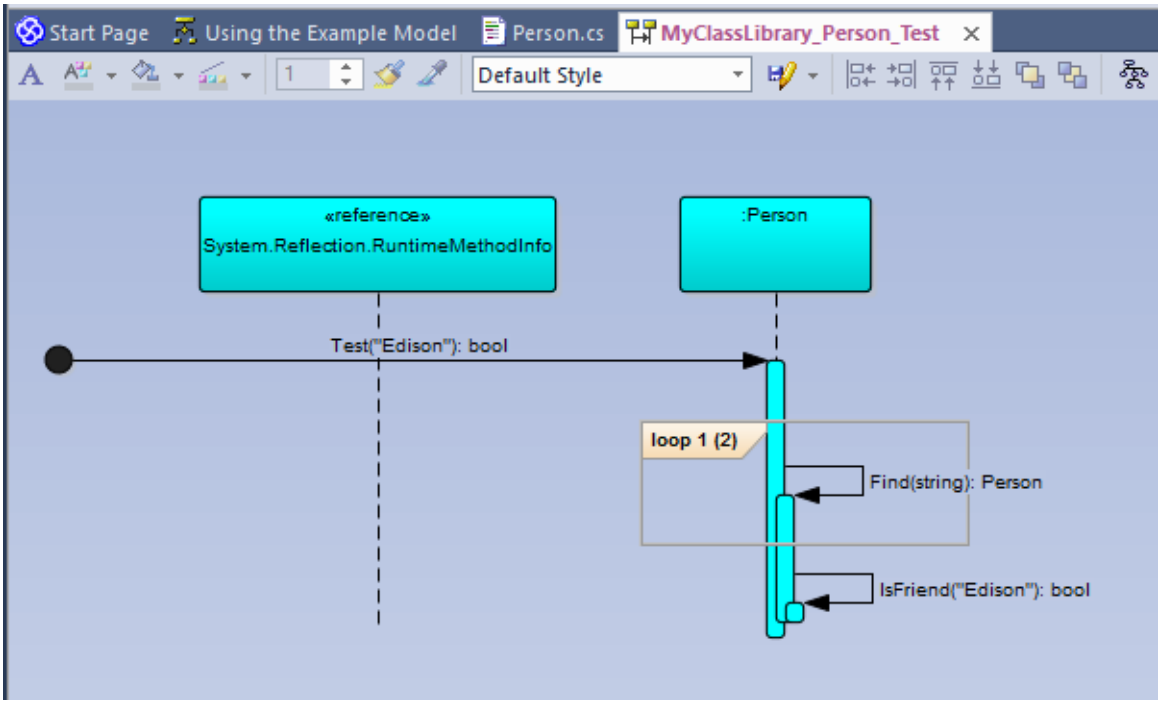
调用方法



视图记录历史



生成序列图表


Start Page
Using the Example Model
Person.cs
MyClassLibrary_Person_Test
Default Style
«reference»
System.Reflection.RuntimeMethodInfo
:Person
Test("Edison"): bool
loop 1 (2)
Find(string): Person
IsFriend("Edison"): bool

删除对象

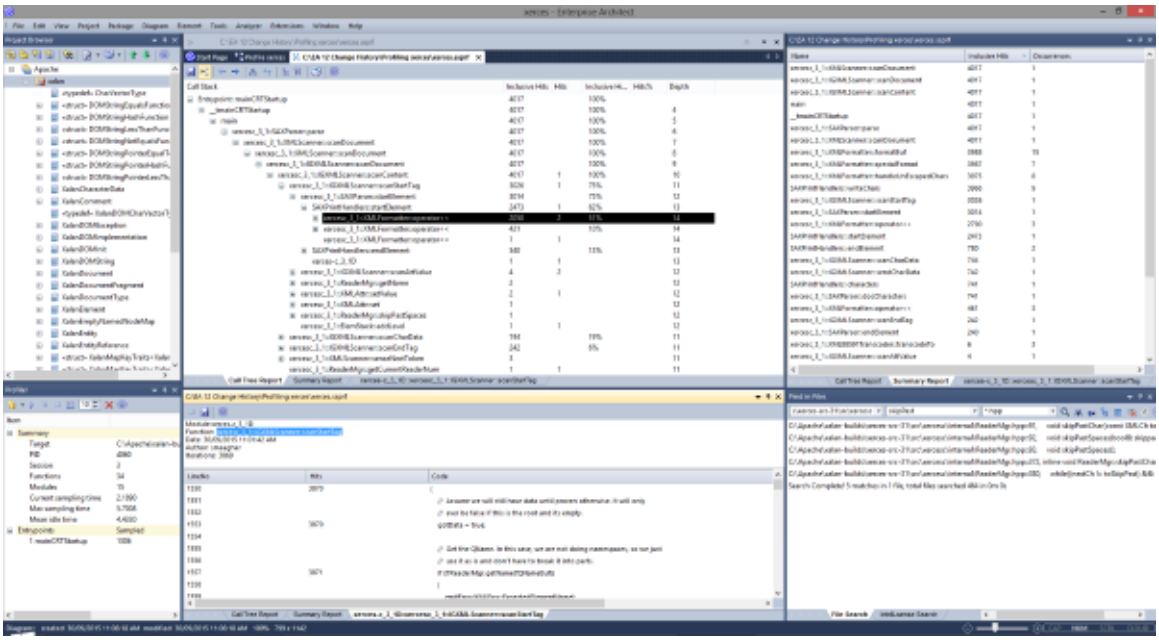
您可以通过在工作台中选择object 、右键单击它并选择 删除”选项来轻松删除它。

关闭工作台

要关闭 Workbench，请执行以下任何操作：

- 从物件工作台上下文菜单中选择 重置”
- 按任何调试器工具栏上的停止按钮
- 删除工作台上的所有对象

剖析

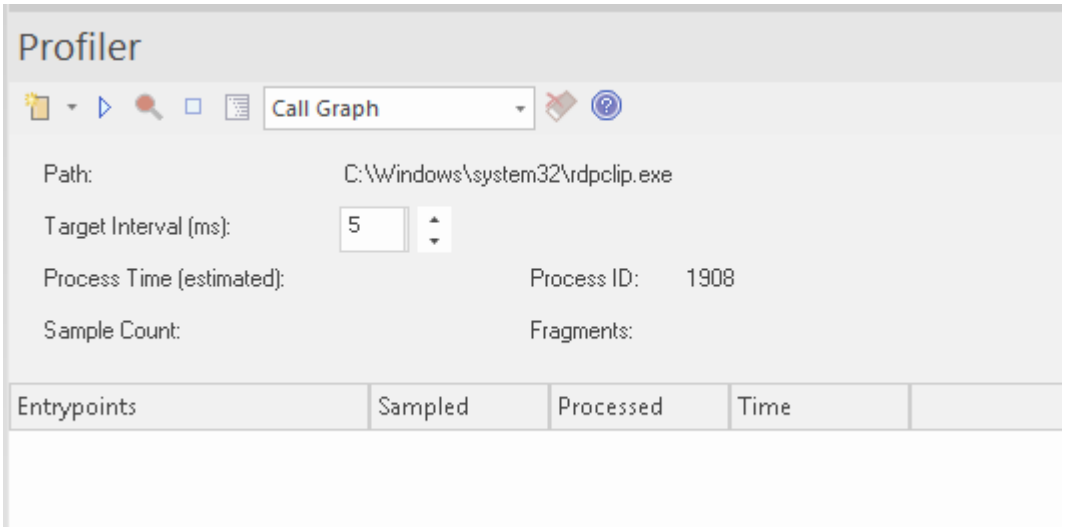


在软件应用程序的生命周期中，调查确定执行速度低于预期的应用程序任务的情况并不少见。您可能还只是想知道当您 按下此按钮”时发生了什么！您可以使用Enterprise Architect的 Profiler 快速解决此问题。结果通常可以在几秒钟内产生，您将很快能够看到正在使用应用程序的操作和所涉及的功能。在特征执行分析器中，特征采用了两个独立的策略；进程执行过程中的采样和进程。一种方法是定期采集样本以识别 CPU 密集型模式，而另一种方法是连接进程以记录对内存的需求。分析数据以产生加权调用图。行为通常可识别为图中的根（入口点），或这些点附近的分支。所有报告都可以按需审查。它们可以在模型中保存为文件，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

访问

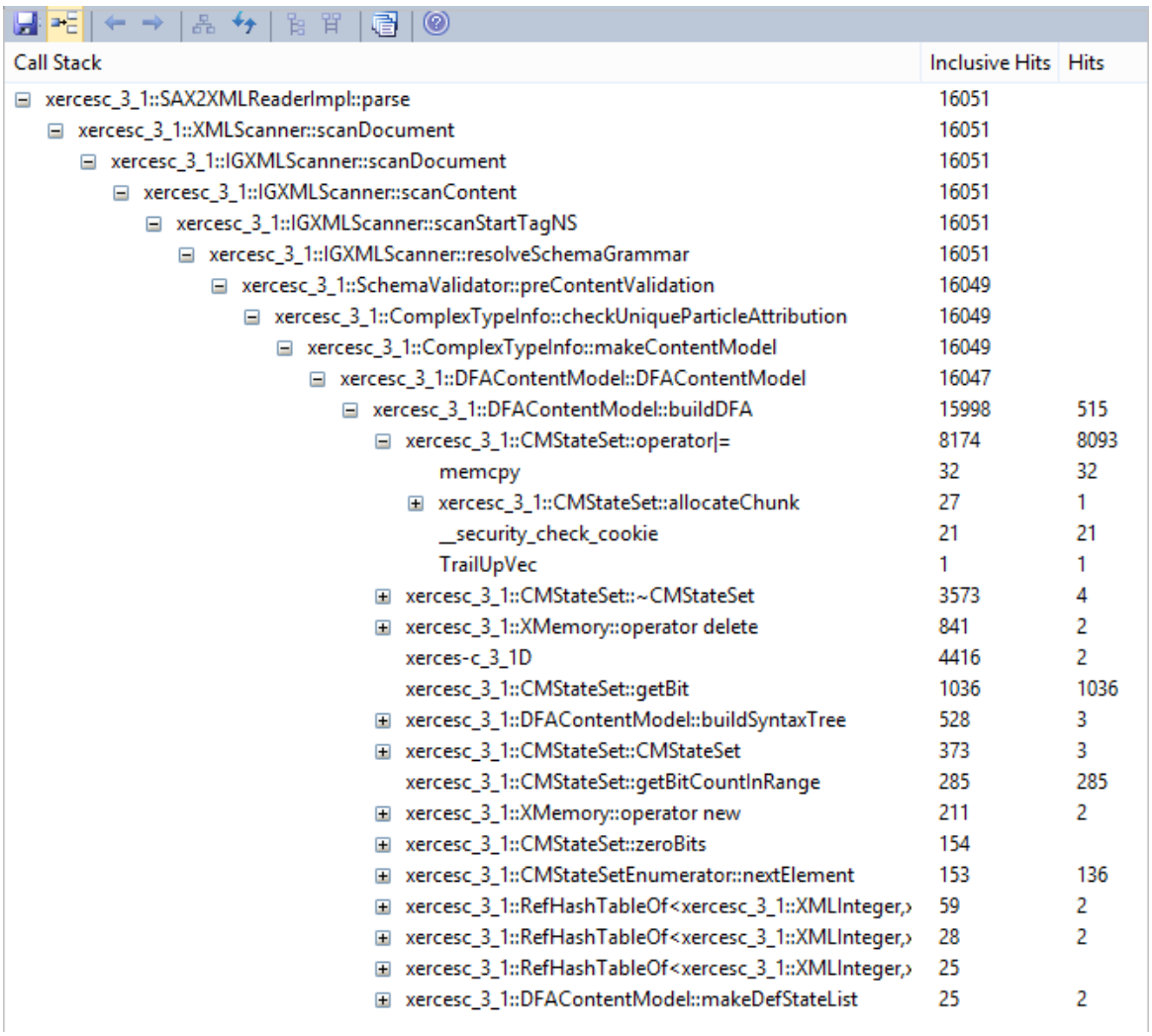
功能区	执行 > 工具 > 探查器
其它	执行分析器工具栏：分析器窗口|探查器

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。

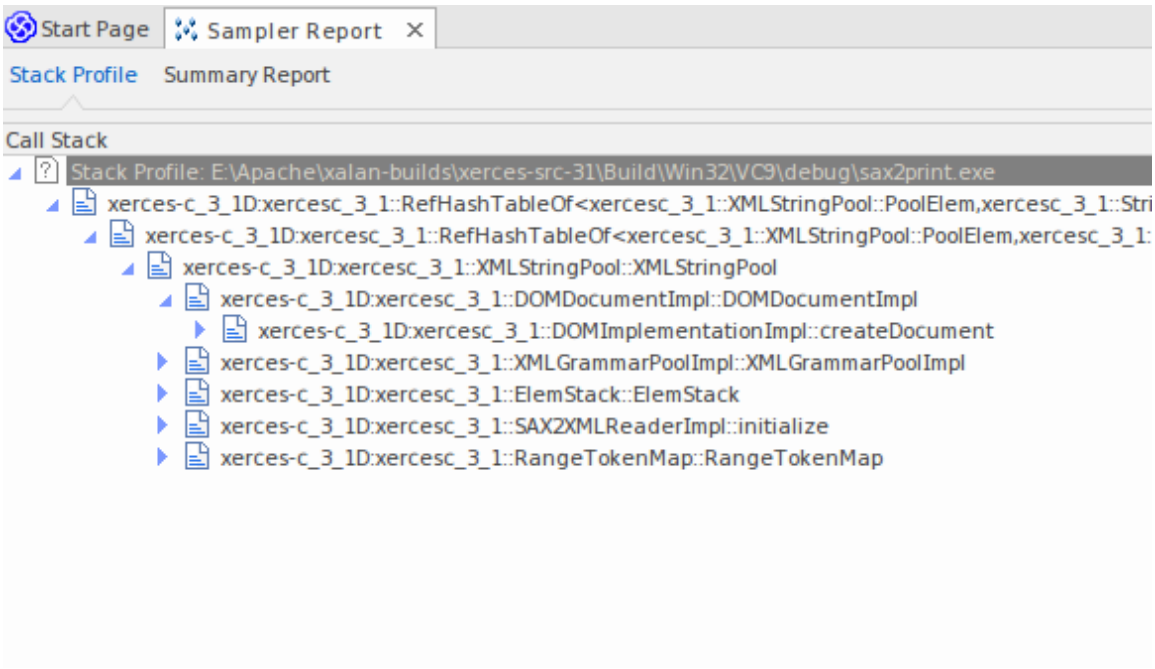
加权调用图



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。

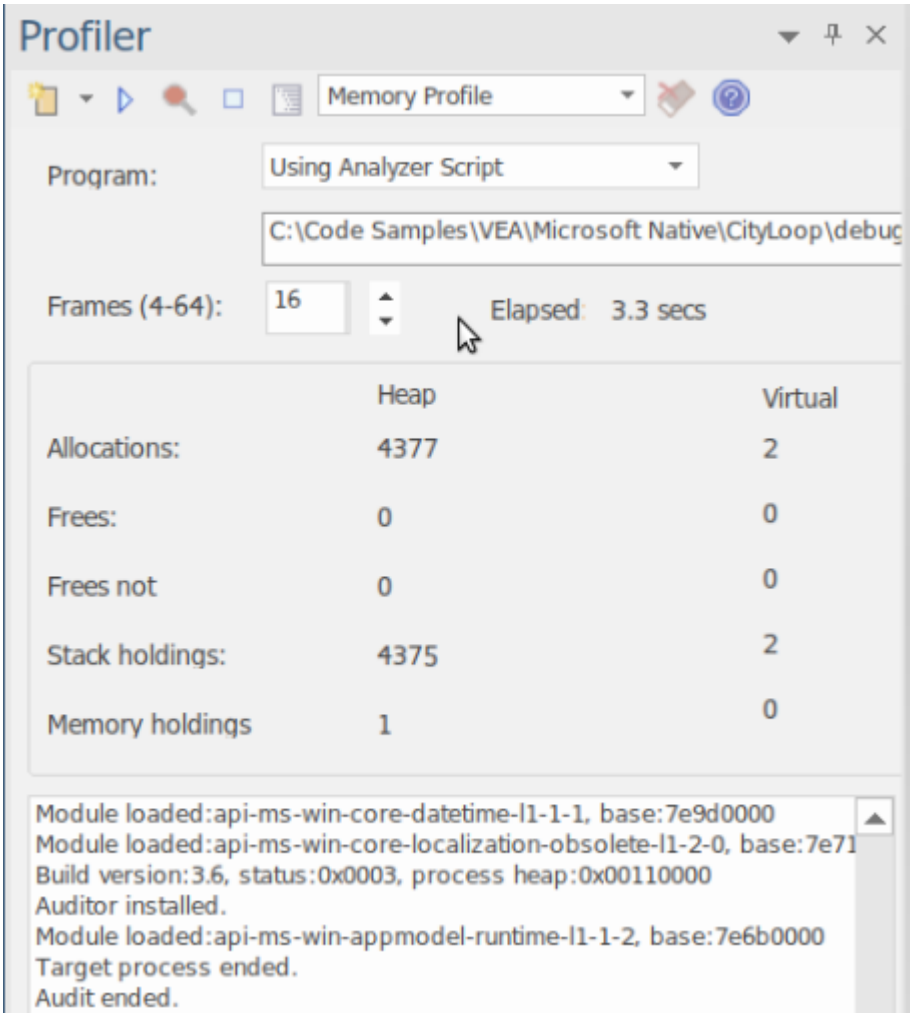
堆栈配置



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。

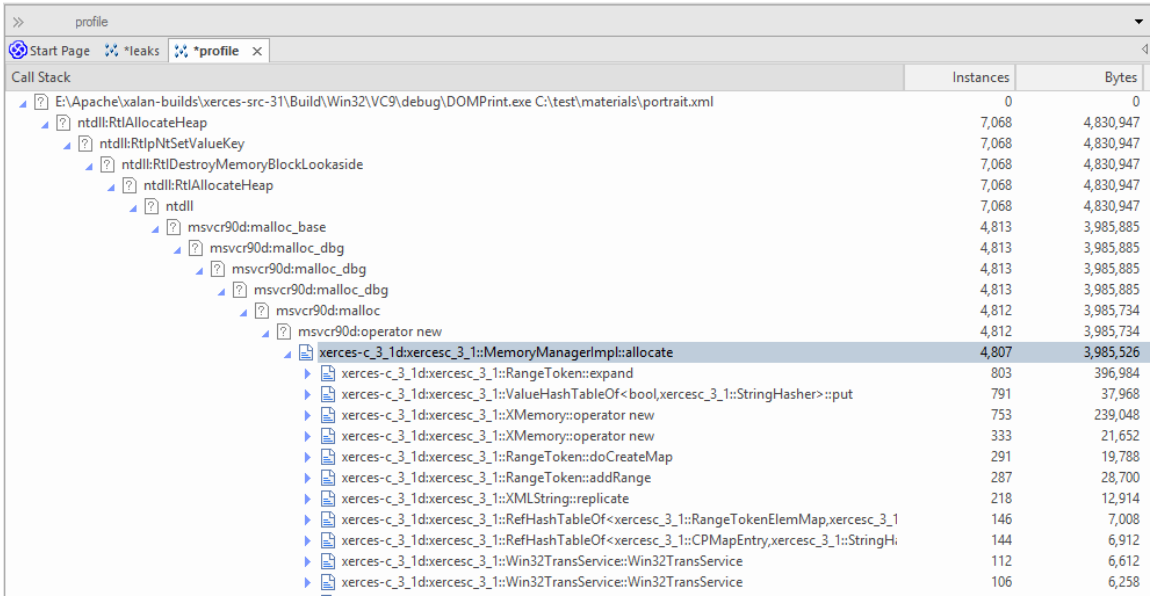
```
106
107 template <class TVal, class THasher>
108 void RefHashTableOf<TVal, THasher>::initialize(const XMLSize_t modulus)
109 {
110     if (modulus == 0)
111         ThrowXMLwithMemMgr(IllegalArgumentException, XMLExcepts::HshTbl_ZeroMo
112
113     // Allocate the bucket list and zero them
114     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

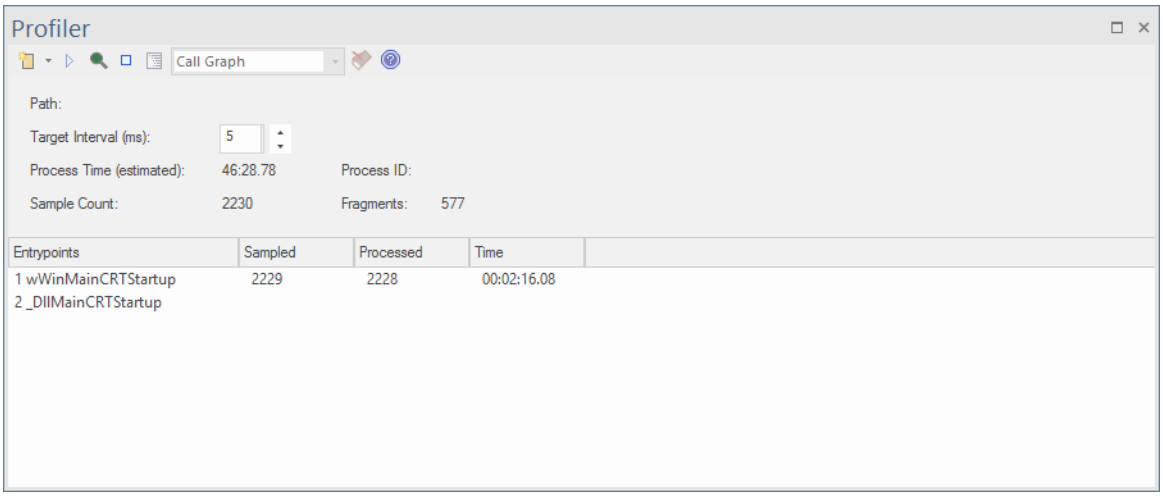
工具	描述
调用图	通过在程序中的活动期间取样来分析性能。每个样本代表一个堆栈。使用工具栏控制的间隔采集样本。在这种情况下，性能不佳是通过在采样时间段内重复最多的行为模式来评定的。这个数字是用来衡量生成的调用图的。
内存配置	通过挂钩程序进行的内存分配来分析性能。在这种情况下，性能不佳是由对内存请求最多的活动来评定的。这个数字是用来衡量生成的调用图的。
堆栈配置	Stack Profiler 使您能够在源代码中设置标记，以便每当执行命中该标记时，都会捕获完整的堆栈跟踪。随着应用程序继续执行并从正在运行的可执行文件中的多个位置访问标记的位置，将构建一个非常详细且有用的图片，显示代码中特定点的热点和使用场景。 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。

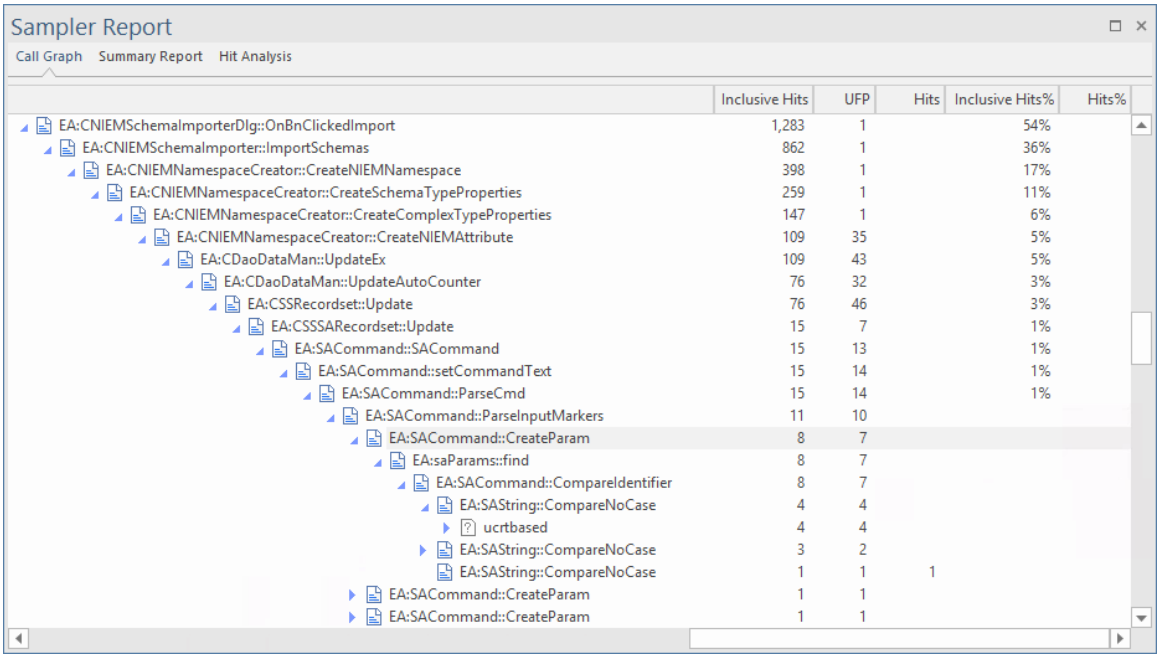
结果

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。

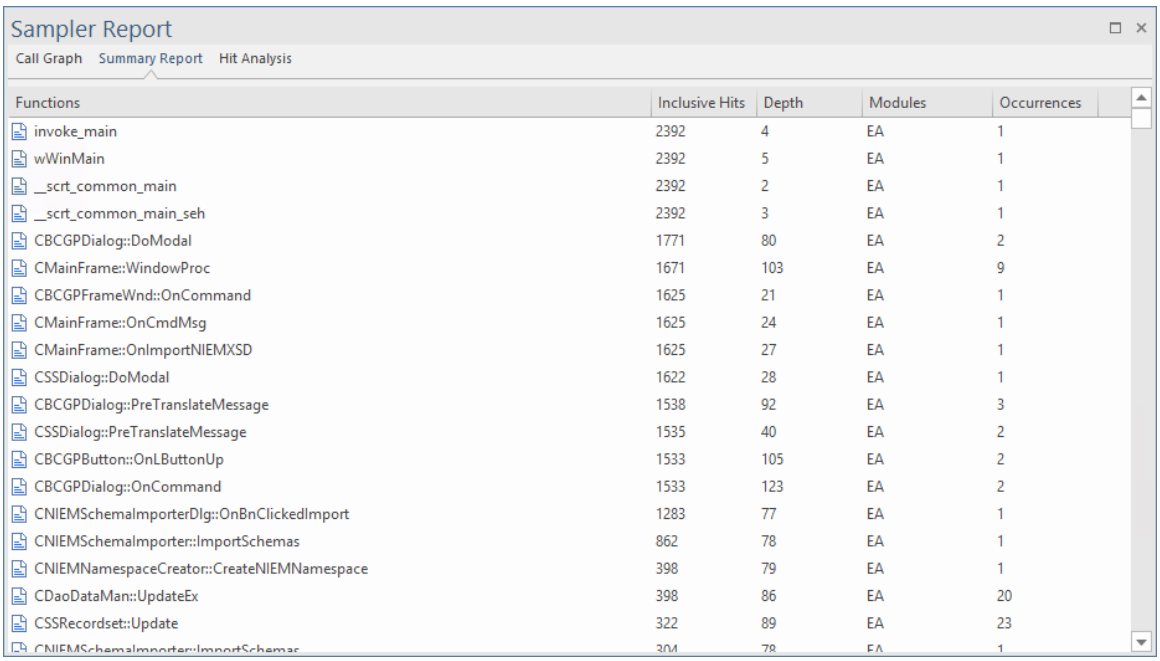
报告按钮将通过暂停捕获或完全停止 Profiler 来启用。

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



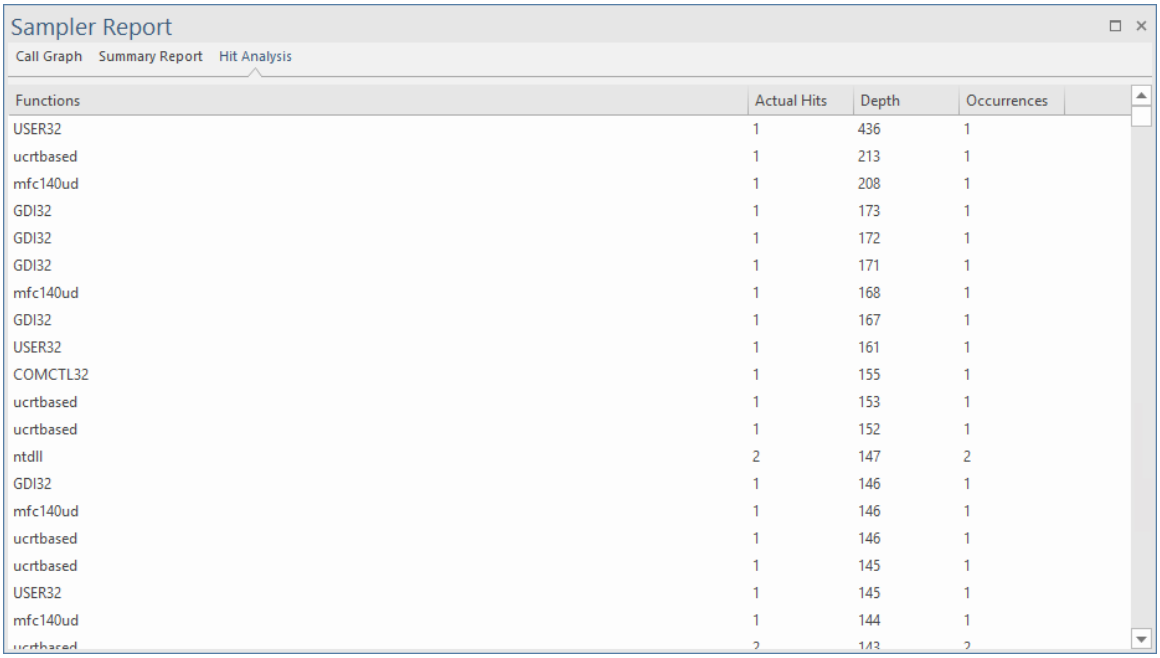
命中分析选项卡

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。



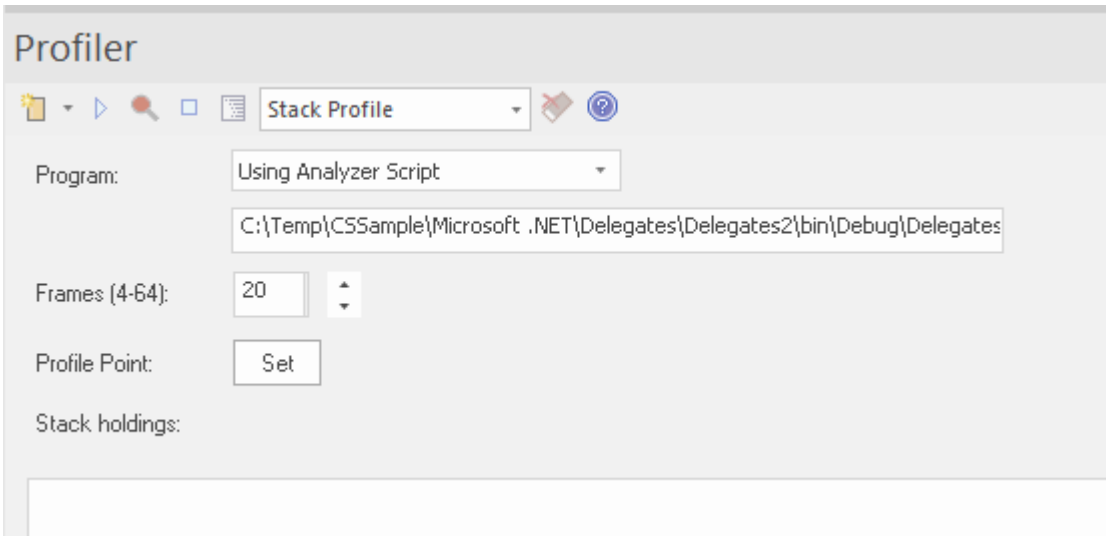
Sampler Report

Call Graph Summary Report Hit Analysis

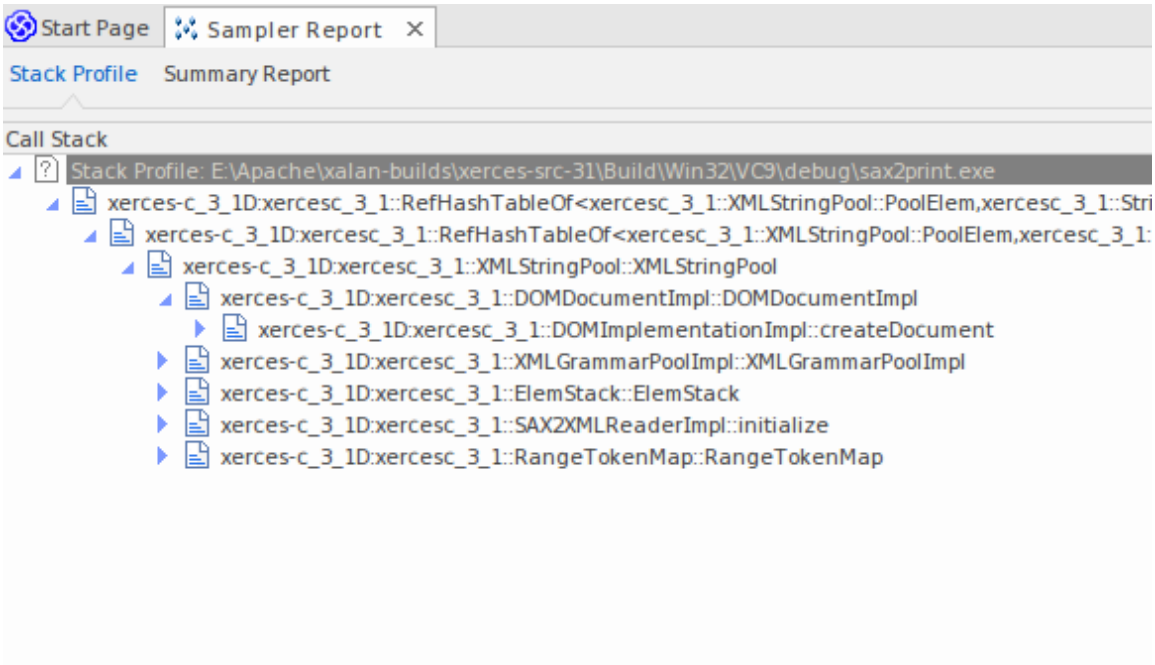
Functions	Actual Hits	Depth	Occurrences
USER32	1	436	1
ucrtbased	1	213	1
mfc140ud	1	208	1
GDI32	1	173	1
GDI32	1	172	1
GDI32	1	171	1
mfc140ud	1	168	1
GDI32	1	167	1
USER32	1	161	1
COMCTL32	1	155	1
ucrtbased	1	153	1
ucrtbased	1	152	1
ntdll	2	147	2
GDI32	1	146	1
mfc140ud	1	146	1
ucrtbased	1	146	1
ucrtbased	1	145	1
USER32	1	145	1
mfc140ud	1	144	1
ucrtbased	2	143	2

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。



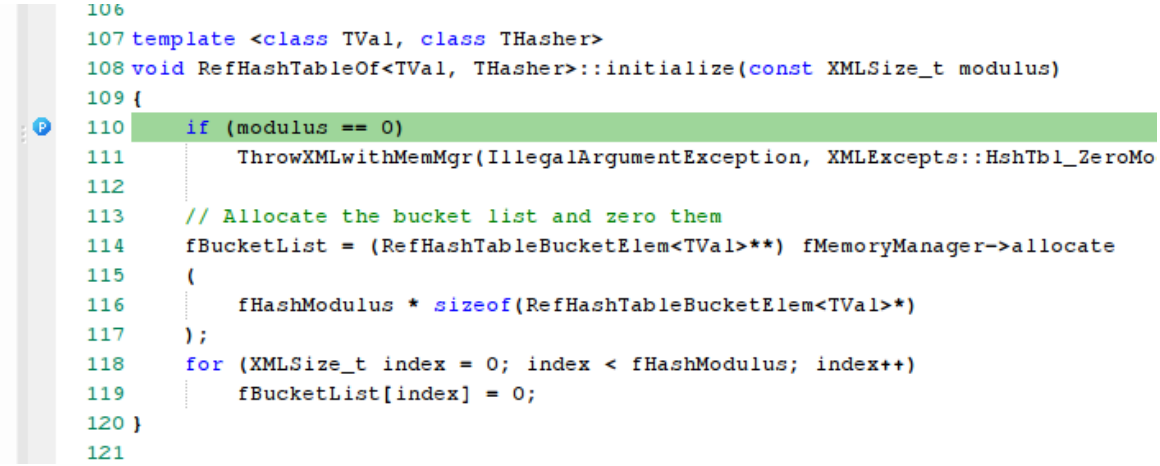
堆栈配置配置报告与内存配置报告一样，以 反向堆栈"的顺序显示。这意味着报告的根始终是单个节点（在本例中为标记），然后树呈扇形展开以显示已访问标记位置的所有不同位置。



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。

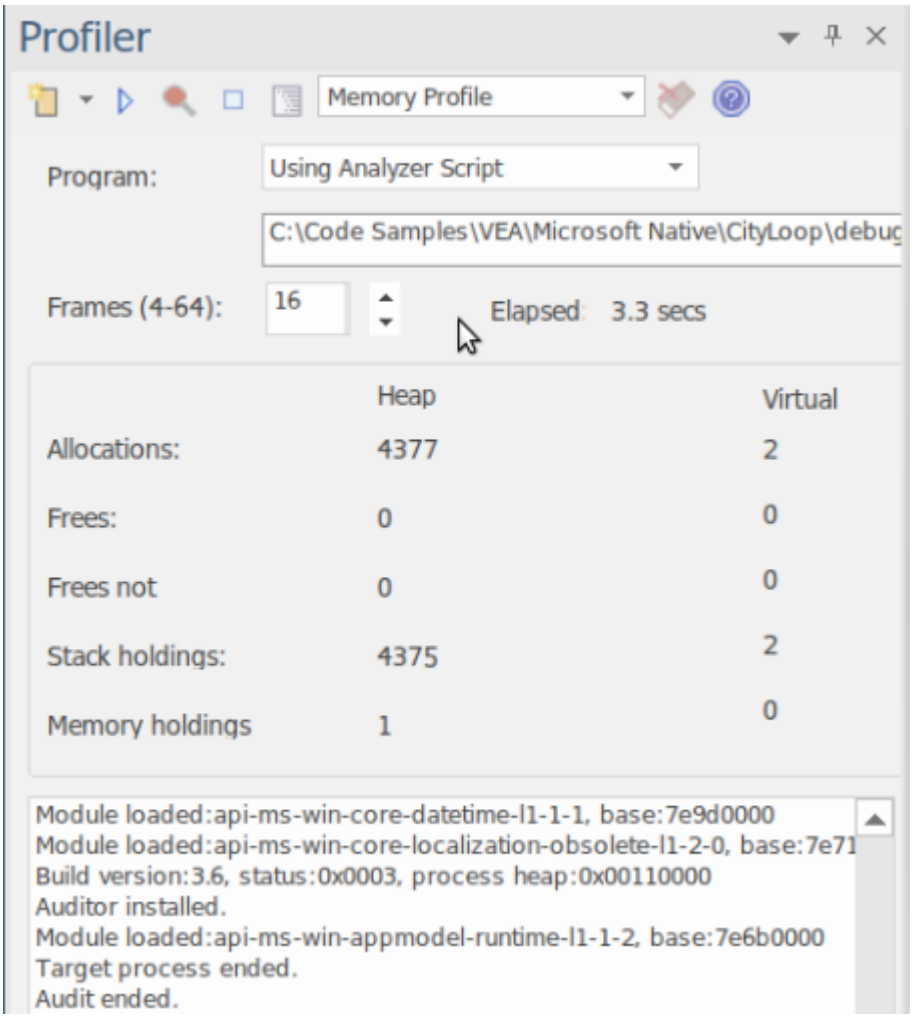
结果

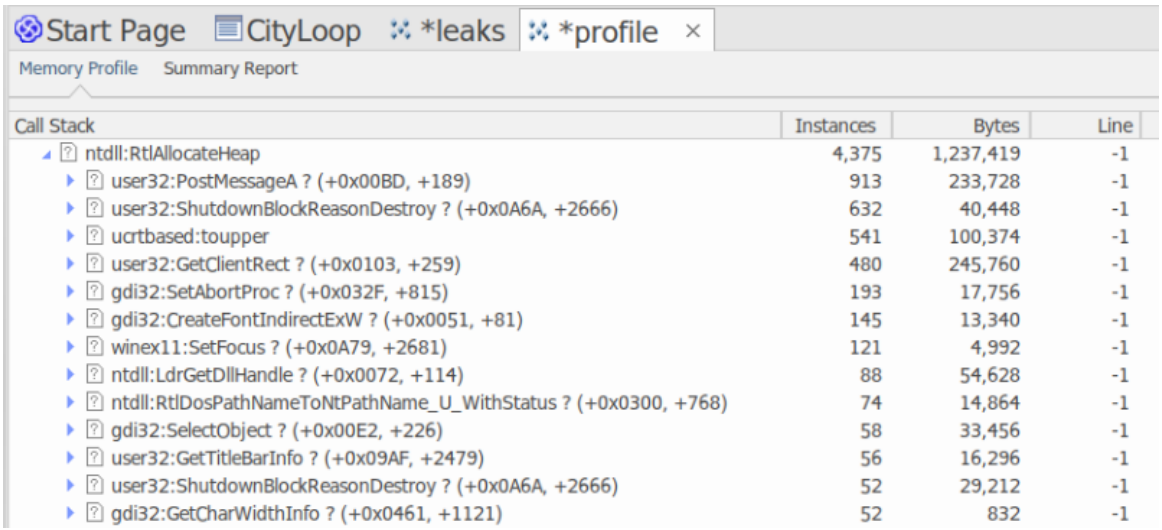
单击 Profiler 控件工具栏上A报告按钮可以生成结果。此按钮在以下任一情况下启用：

- 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

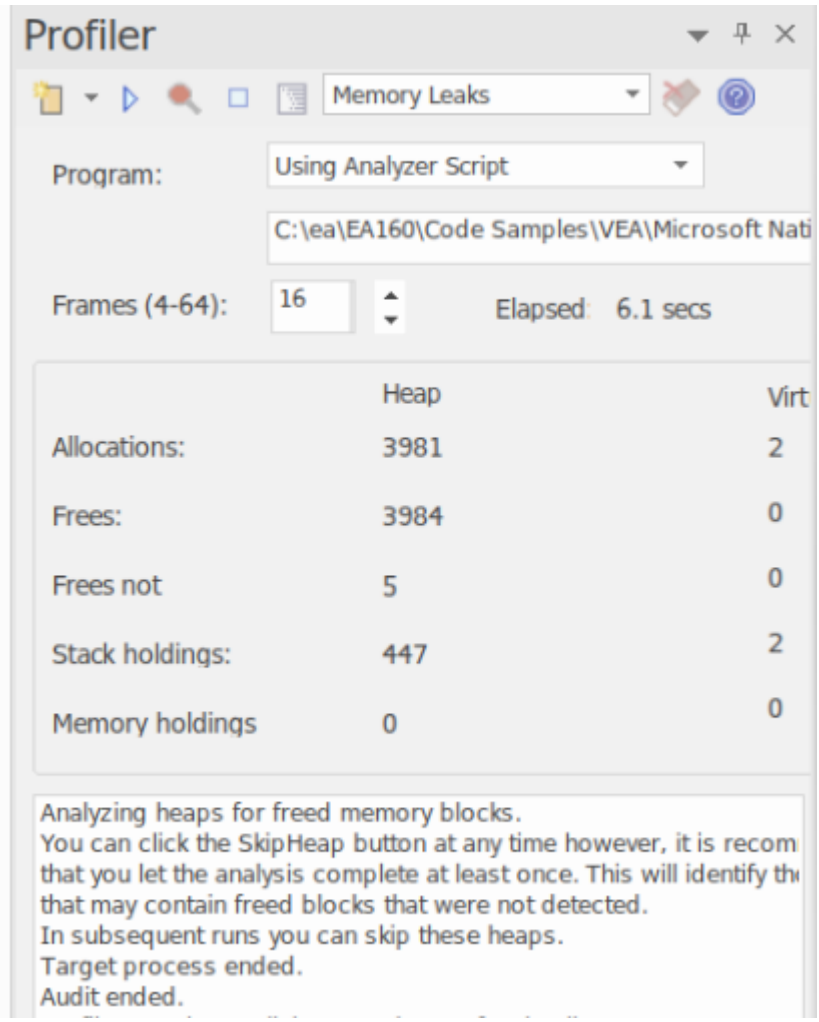
会议期间可随时产生结果；但是，必须禁用捕获才能使 报告”按钮变为活动状态。让运行多长时间由您决定。您可以通过暂停捕获或完全停止 Profiler 来启用 Report 按钮。

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。

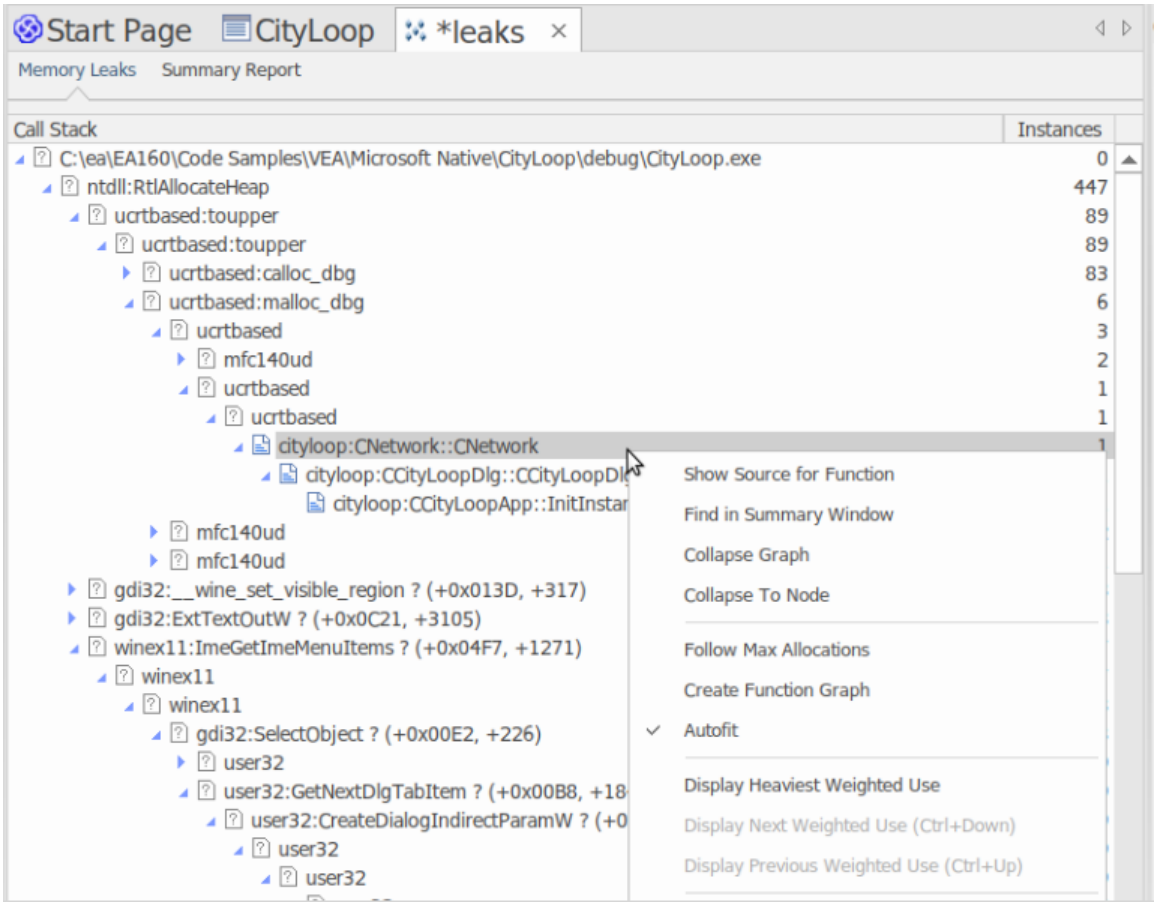
需求

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。

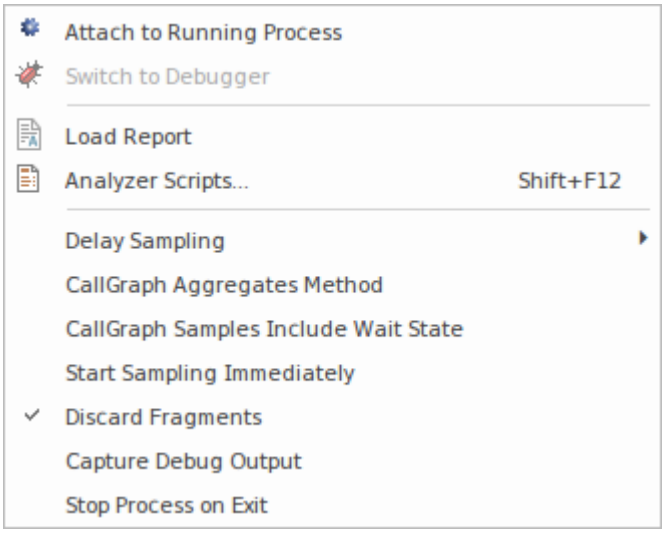
报告可能包含可变数量的 噪音"。要聚焦您特别关注的领域，请在摘要报告中找到您已知的函数，并使用它直接导航到图表中的特征线。

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。



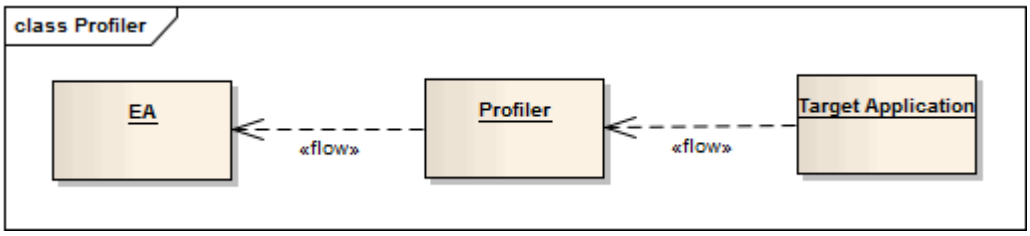
选项

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	
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。

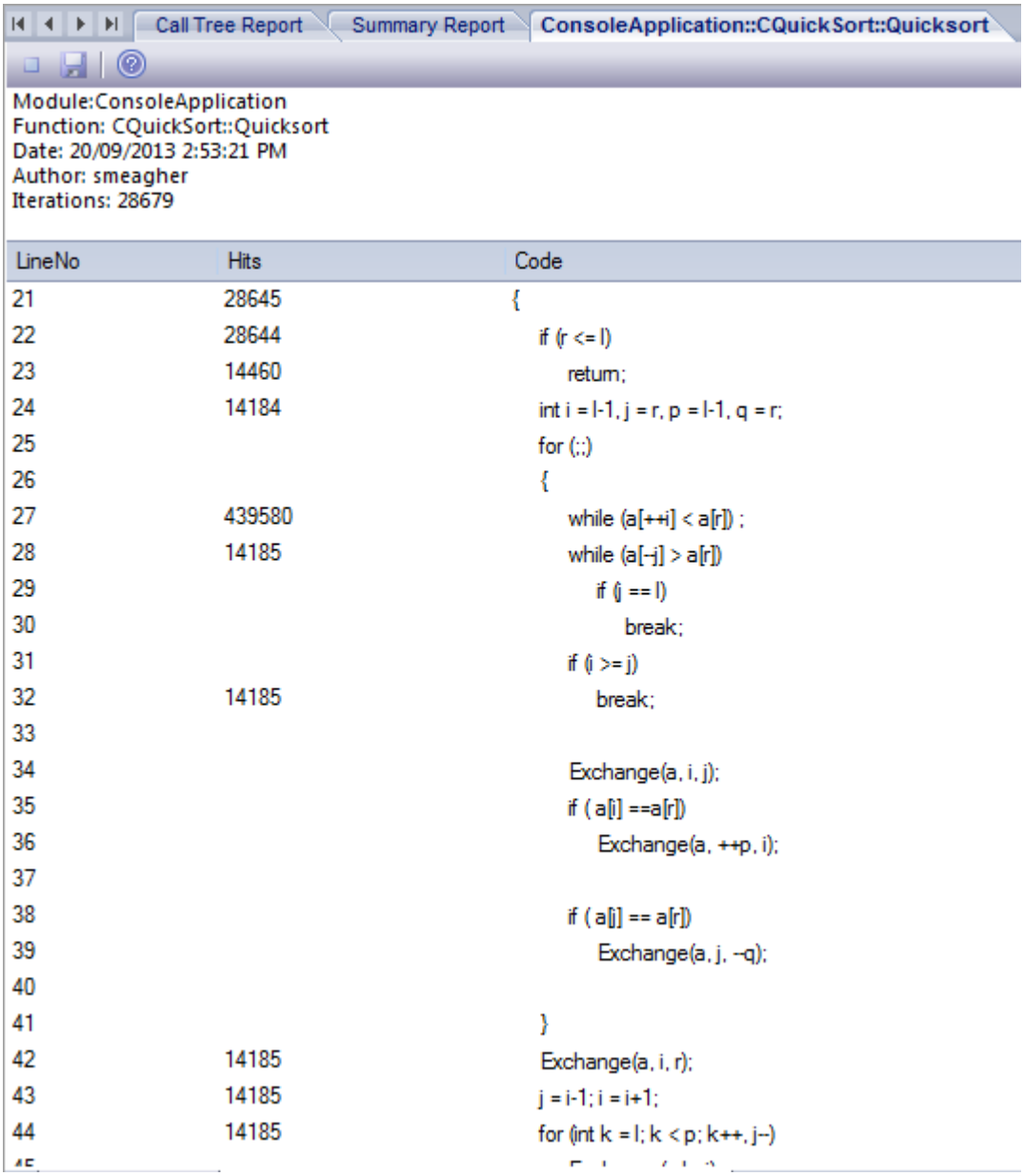
支持的平台

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。



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。

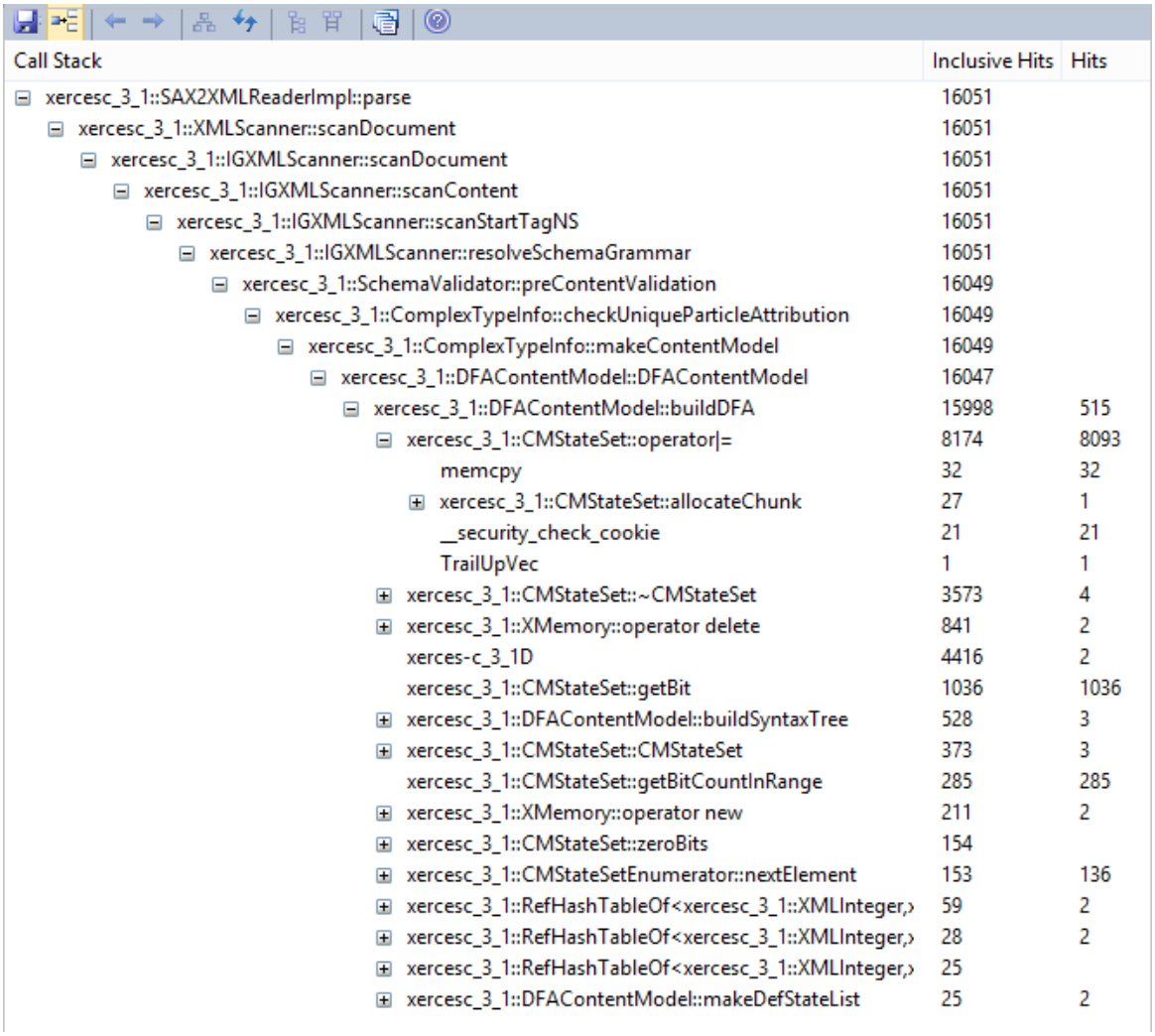
从文件加载报告

该选项可从 Profiler 窗口的下拉菜单中获得

生成报告

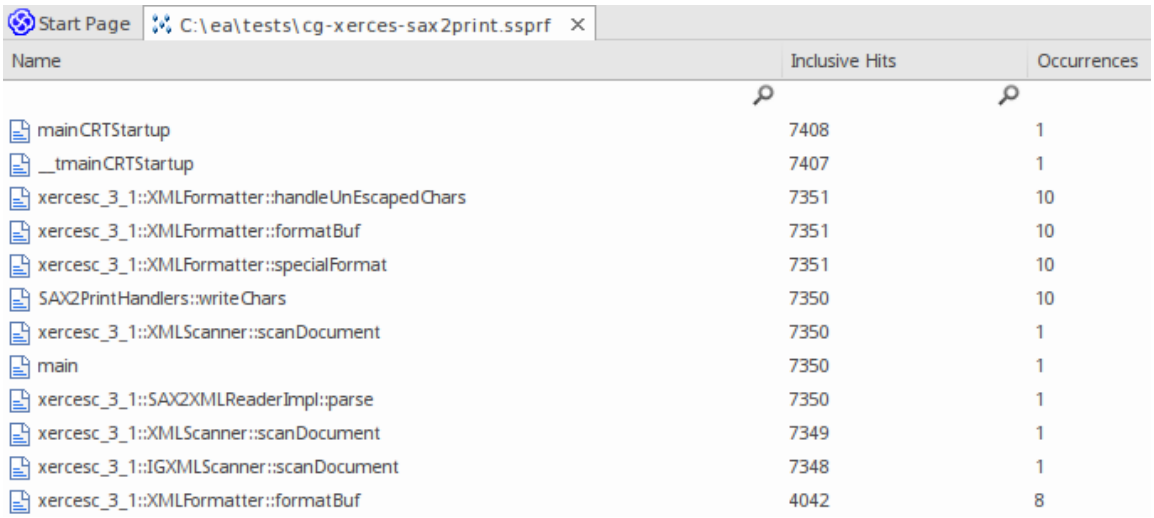
在 Profiler 窗口中，单击工具栏中的  图标。

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

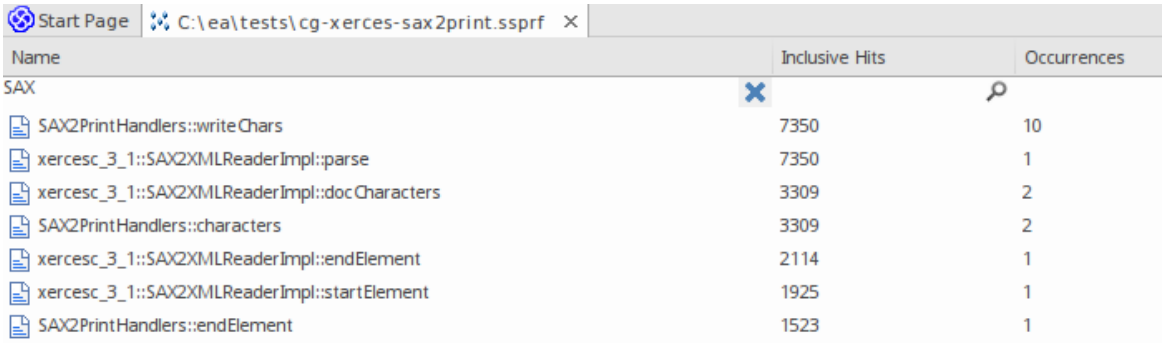
函数总结



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

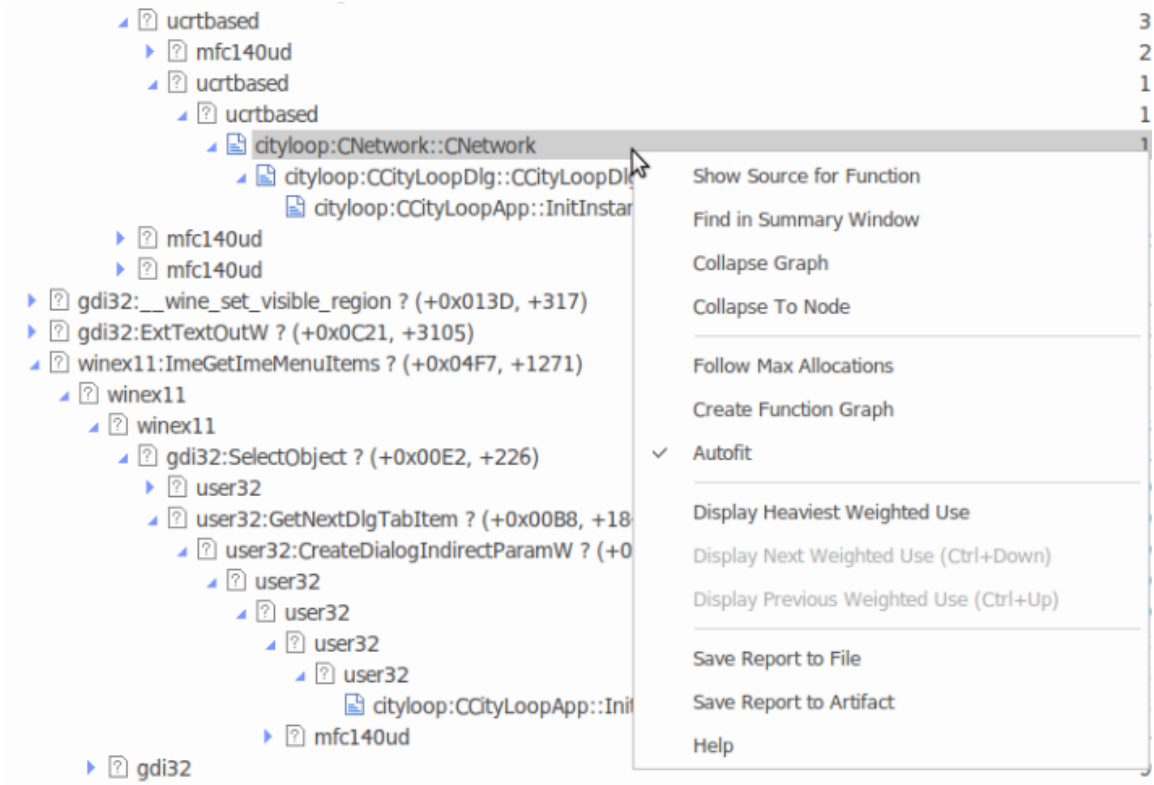
未过滤的摘要报告按包含命中的顺序列出所有参与函数。



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

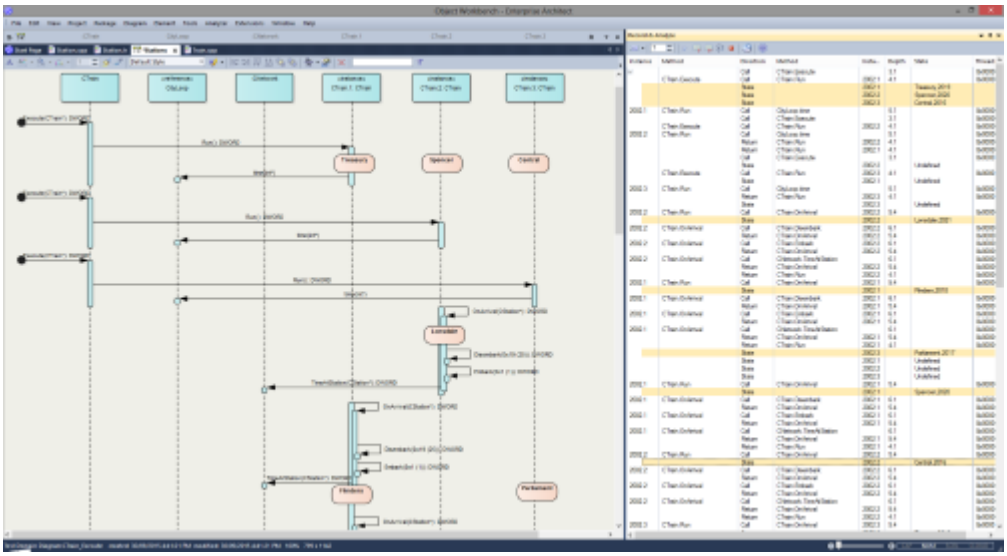
您可以将任何当前报告保存为团队图书馆中某个类别、主题或文档的资源。然后可以随时共享和查看报告，因为它与模型一起保存。这可以帮助您：

- 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

记录



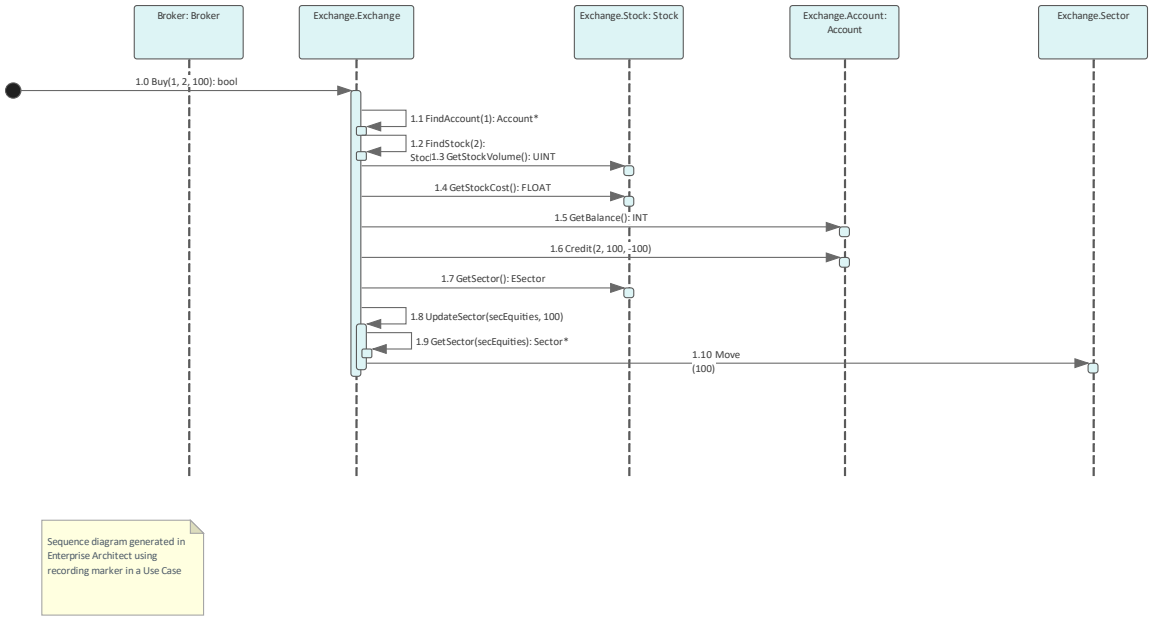
序列图是理解行为的极好帮助。类协作图也很有帮助。除了这些，有时调用图正是我们所需要的。再说一次，如果你有这些信息，你可以用它来记录一个用例，为什么不建立一个测试域呢？ Enterprise Architect分析器可以为您生成所有这些，并从单个记录中生成。它通过记录一个正在运行的程序来做到这一点，它适用于所有最流行的平台。

访问

功能区	执行 > 工具 > 记录器

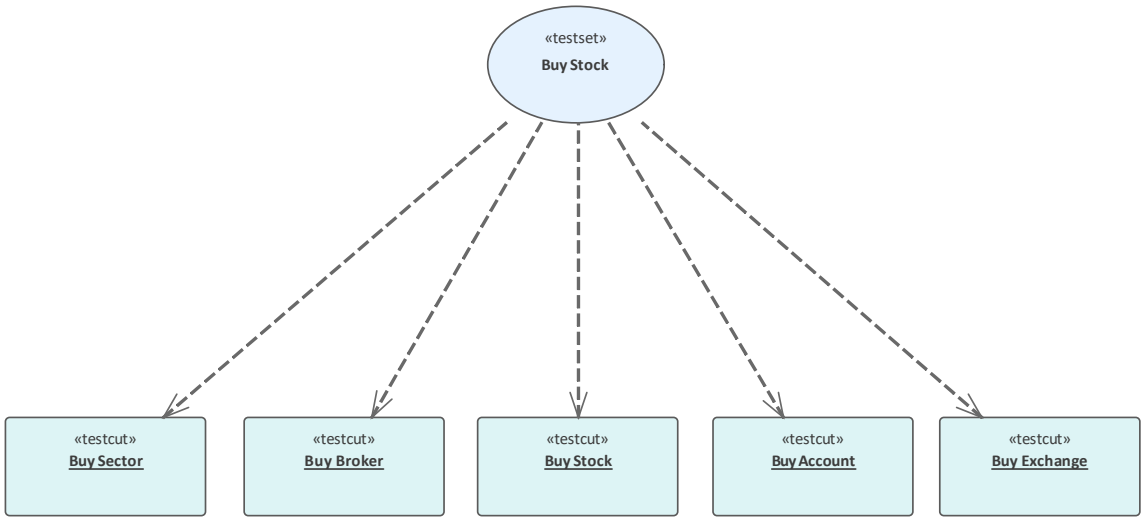
概述

在最简单的情况下，即使使用全新的模型，只需很少的步骤即可生成序列图。你甚至不必配置分析器脚本打开Enterprise Architect代码编辑器 (Ctrl+Shift+O)，在您选择的函数中放置一个记录标记，然后将Enterprise Architect调试器附加到运行该代码的程序中。每当调用该函数时，都会捕获其行为以形成记录历史。从这段历史可以很容易地创建这些图表。



示例模型记录中的序列图。

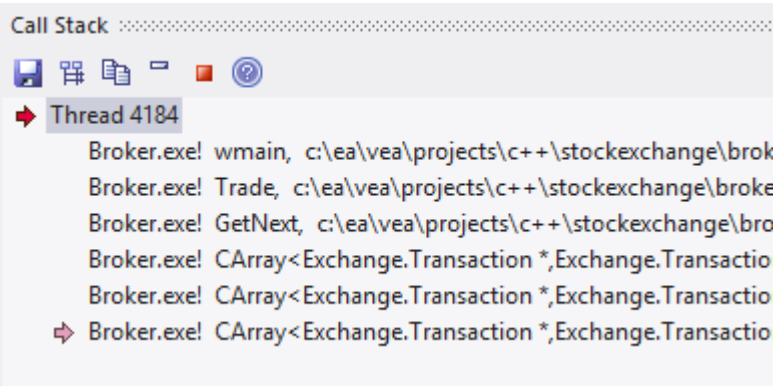
来自同一录音的类协作图。



来自同一记录的测试域图。

当然，分析器脚本是最好的想法，并且开辟了极其丰富的开发环境，但值得注意的是，没有分析器也可以获得显着的效果。 Enterprise Architect调试器和 Profiler 工具也是如此。

兴趣点：您可以A记录线程时查看线程的行为。在录制过程中显示调用堆栈堆栈的更新将实时显示，就像动画一样。这是一个很好的反馈工具，在某些情况下它可能就是所需要的。

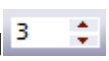


特征一目了然

图表生成

- 序列图
- 类协作图
- 测试域图
- 状态转移
  捕获
- 调用图

控件

- 支持多线程和单线程模型
- 支持堆栈深度控制 

- 支持过滤器限制捕获
- 过滤器通配符支持
- 实时堆栈更新

集成

- 类模型
- 测试域
- 状态机
- 可执行状态机
- 单元测试

平台

- 微软.NET
- 微软原生
- Java
- PHP
- 广发银行
- 安卓

需求

- 记录可供Enterprise Architect所有版本的用户使用

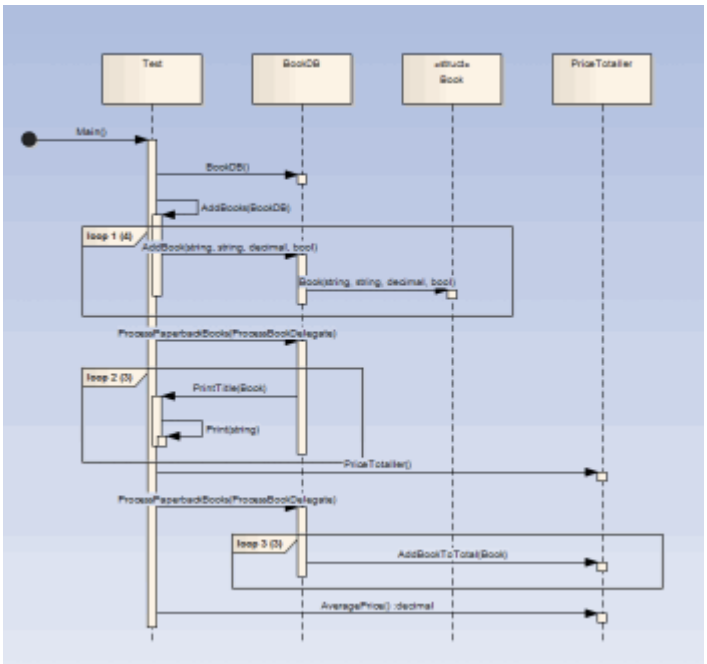
注记

- Oracle 的Java服务器平台 特征"不支持可视化执行分析器的调试和记录特性

这个怎么运作

本主题介绍了可视化执行分析器如何生成序列图。

解释

积分	细节
用途	可视化执行分析器使您能够从应用程序实时执行的记录中生成序列图。随着应用程序的运行，每个线程的历史都会被记录下来。此历史可用于生成序列图。 这是一个计算书籍价格的程序生成的序列图：  录音机如何知道要记录什么？ • 录音机通过录音笔工作；这些由您放置在感兴趣的功能中 调用Java中的堆栈可以延伸到肉眼所能看到的范围之外。我们如何将录制限制为仅十帧？ • 记录器由记录器工具栏上设置的深度或与存储在模型中的标记集相关联的深度控制
它是真实的	录制时，不修改目标应用程序；根本没有任何图像或模块的检测。使用程序的 发布”版本制作A录音是程序所做工作的可靠文档。
你从哪里开始	我们有一个非常大的服务器应用程序；那么我们从哪里开始呢？如果您对要录制的程序知之甚少或根本不了解，并且很少或根本没有模型可以说出来，那么您最好从 Profiler 开始。在以特定方式使用程序的同时运行 Profiler 可以从所呈现的入口点和调用图快速识别使用案例。拥有这些知识可以使您聚焦于未发现的区域并记录这些功能。 如果你源代码，你需要做的就是在你感兴趣的函数中放置一个记录标记。我们建议不要同时在多个函数中放置多个记录标记。在实践中，这已被证明不太有用。你在哪里放置录音标记？对于 Windows UI 程序，以及与某些业务用

	例相关的情况，您可能首先在事件处理程序中放置一个似乎最相关的消息。如果您正在调查实用程序函数，只需在开始处或附近设置一个方法记录标记。 对于服务、守护进程和批处理，您可能希望针对每个感兴趣的行为对程序进行一次概要分析，并使用报告来探索那些未发现的区域。
小费	最好在调试前快速浏览一下断点和标记窗口，并检查此处列出的标记是否符合您的预期。
场景	• Microsoft 本机 C 和 C++、VB（窗口程序、窗口服务、控制台程序、COM 服务器、IIS ISAPI 模块、旧版） • 微软.NET（ASP.NET、窗口Foundation (WPF)、窗口、工作流服务、设备、模拟器） • Java（应用程序、小程序、Servlet、Beans） • 安卓（对设备和模拟器使用 Android 调试桥） • PHP（网络网站脚本） • 广发银行（窗口/Linux 互操作性）

记录历史

当应用程序的执行分析遇到用户定义的记录标记时，记录的所有信息都保存在 Record & Analyze 窗口中。

访问

功能区	执行 > 工具 > 记录器 > 打开记录器

功能

功能	信息/选项
信息显示	记录和分析窗口中的列是： • - 唯一序列序列 • Threads - 操作系统线程 ID • Delta - 自序列开始以来经过的线程 CPU 时间 • 方法 - 有两个方法列：第一个显示调用者，如果返回，则显示当前帧；第二个显示调用的函数或返回的函数 • Direction - Stack Frame Movement，可以是调用、返回、状态、Breakpoint或Escape（Escape在内部制作序列图时使用，用来标记一次迭代的结束） • Depth - 调用时的堆栈深度；用于生成序列图 • -状态-状态之间的状态 • 源- 有两个源列：第一个显示调用者的源文件名和行号，如果是返回，则显示当前帧；第二个显示函数调用或返回函数的源文件名和行号 • Instance - 有两个Instance列，只有当产生的序列图包含状态Transitions时才有值；这些值由以逗号分隔的两项组成 - 第一项是捕获的类实例的唯一编号，第二项是类的实际实例 例如：假设一个类'类'的内部值为4567，程序创建了该类的两个实例；这些值可能是： - 4567, 1 - 4567,2 第一个条目显示类的第一个实例，第二个条目显示第二个实例
操作信息	Record & Analyze 窗口工具栏提供了功能用于控制记录分析器脚本执行的功能。 录制完成后，您可以使用 录制和分析”窗口上下文菜单对录制结果执行完成操作。

注记

- 每个操作的复选框用于控制此调用是否可用于从该历史记录中创建序列、测试域类或协作类图

- 除了使用复选框启用或禁用调用之外，您还可以使用上下文菜单选项来启用或禁用整个调用、对给定方法的所有调用或对给定类的所有调用

图表特征

当您生成序列图时，它包括以下特征：

特征

特征	细节
参考	当可视化执行分析器无法匹配模型中的操作的函数调用时，它仍然会创建序列，但也会创建任何它无法定位的类的引用。 它适用于所有语言。
片段	片段图中显示的序列代表一段代码的循环或迭代。 可视化执行分析器尝试将函数范围与方法调用相匹配，以尽可能准确地直观地表示执行。
状态	如果在录制过程中使用了状态机，则任何状态转换都会在导致转换发生的方法调用之后呈现。 状态是根据每个方法返回给它的调用者来计算的。

记录设置

本节说明如何准备记录应用程序的执行。

脚步

节
先决条件 - 要设置记录序列图的环境，您必须： • 已完成编译和调试的基本设置并为包创建了执行分析脚本 • 能够成功调试应用程序
通过应用过滤器缩小记录的聚焦。
通过调整堆栈深度来控制控件的细节。

控件深度

在应用程序中记录特别高级的点时，堆栈帧计数可能会导致收集大量信息；为了获得更快更清晰的图片，最好限制工具栏上的堆栈深度：

- 断点和标记窗口或
- 记录和分析窗口

访问

功能区	执行 > 工具 > 记录器 > 打开记录器

设置录音堆栈深度

您可以在断点和标记窗口或记录和分析窗口的工具栏上的数字字段中设置记录堆栈深度：

默认情况下，堆栈深度设置为三帧。可以输入的最大深度为 30 帧。

深度与遇到记录标记的堆栈帧相关；因此，在开始录制时，如果堆栈帧为 6，并且堆栈深度设置为 3，则调试器会记录第 6 到第 8 帧。

对于堆栈非常大的情况，建议您首先使用 2 或 3 的低堆栈深度。从那里您可以逐渐增加堆栈记录深度并插入额外的记录标记以扩展图片，直到所有必要的信息都显示。

放置录制标记

本节说明如何放置记录标记，使您能够在两点之间静默记录代码执行。该记录可用于生成序列图。
由于此过程记录了多个线程的执行，因此它在捕获事件驱动的序列（例如鼠标和计时器事件）时特别有用。

访问

功能区	执行>窗口>断点

行动

行动
可以使用不同的记录标记来记录执行流程；有关这些标记的属性和用法的信息，请参见相关链接。
在断点和标记窗口中管理断点。
激活和停用标记。
使用标记集 - 当您创建断点或标记时，它会自动添加到标记集，默认集或您为特定目的创建的集。

注记

- 断点和标记管理主题（软件工程）描述了不同的视角

设置记录标记

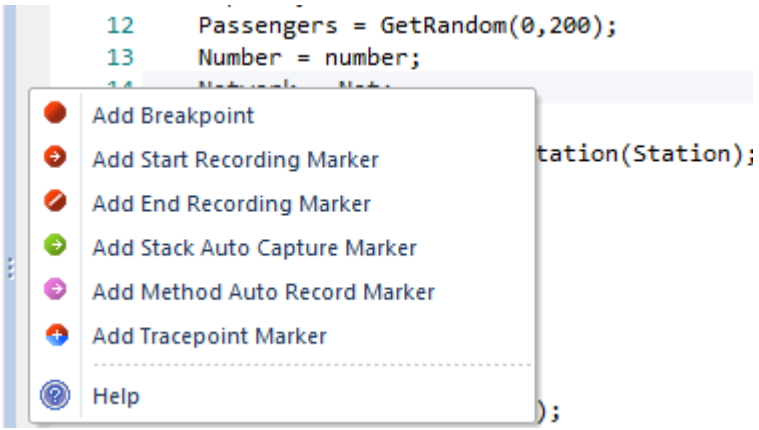
标记在源代码编辑器中设置。它们被放置在一行代码中；当该行代码执行时，执行分析器执行适合标记的记录动作。

访问

使用此处概述的方法之一，显示代码编辑器窗口并加载与所选类或类元素关联的源代码。

功能区	执行 >源> 编辑 > 编辑元素源 执行 >源> 编辑 > 打开源文件
键盘快捷键	在元素上按 Ctrl+E 或 F12 要打开 打开源文件”浏览器，请按 Ctrl+Alt+O

设置录音标记

节	行动
1	在集成源代码编辑器中打开源代码进行调试。
2	找到合适的行，在左边（Breakpoint）空白处右击，调出断点/标记上下文菜单；选择所需的标记类型： 
3	如果已设置开始记录标记，则还必须设置结束记录标记。

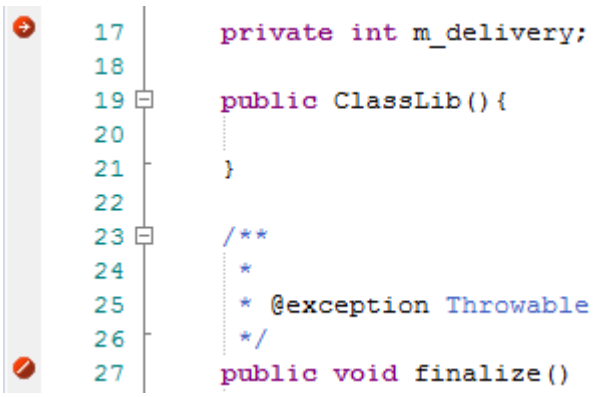
标记类型

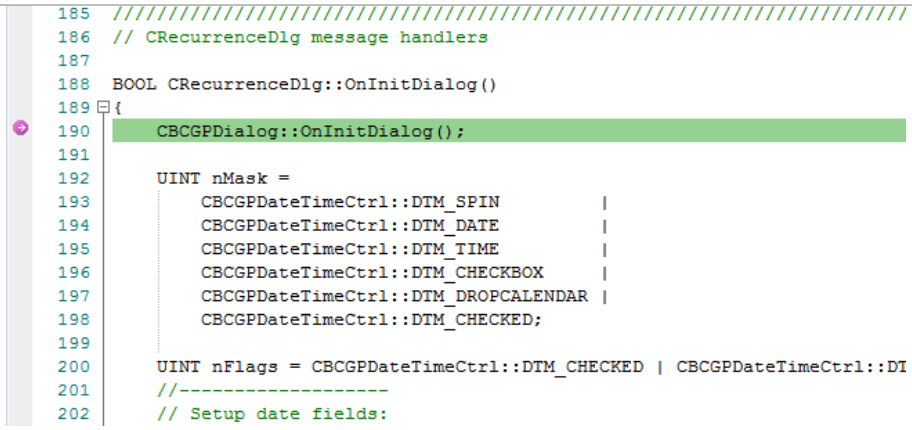
标记真的很棒。不同寻常的是，它们在小心使用时占用空间非常小，它们对正在录制的节目性能的影响可以忽略不计。标记有几种口味（实际上颜色很好），而且总是会添加更多。它们被放置并且在编辑器的左边距可见，所以你需要有一些源代码。

使用到

- 记录一个函数
- 记录函数的一部分
- 使用跨越多个功能的案例
- 记录调用堆栈
- 生成序列图
- 生成测试域图
- 生成类协作图

参考

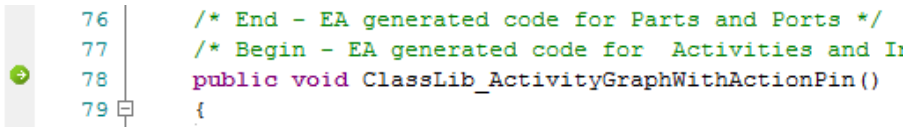
标记	细节
开始/结束记录标记	将标记放置在要记录的代码的开始行和结束行。这些不必在同一个函数中。  17 private int m_delivery; 18 19 public ClassLib(){ 20 21 } 22 23 /** 24 * 25 * @exception Throwable 26 */ 27 public void finalize() 当程序遇到开始录制标记时，将启动新的录制（相机开始滚动！）。当遇到结束标记时，当前录制结束（这是一个 *take* ）。如何使用这些标记取决于您和您对您所关心的系统的了解。 **高级东西（嵌套标记）：** 如果在录制过程中遇到开始录制标记，但在使用中的堆栈深度值禁止捕获的情况下，将启动单独的录制。每个记录都保存在一个堆栈上。当一个结束时，它被删除。这种技术可以在Enterprise Architect中用于在非常复杂的系统中记录和渲染场景。它类似于拼接视频中的短场景以创建预告片。如果你只想记录一个函数，你应该使用自动记录标记。
方法自动记录标记	A自动记录标记使您能够记录特定的函数。当函数完成时，调试器会自动结束录制。这很好，因为录制是一项密集的操作。 函数标记将开始记录标记和结束记录标记合二为一，因此记录在标记点之后执行，并且总是在该函数退出时停止。



```
185 ////////////////////////////////////////////////////////////////////////
186 // CRecurrenceDlg message handlers
187
188 BOOL CRecurrenceDlg::OnInitDialog()
189 {
190     CBCGPDialog::OnInitDialog();
191
192     UINT nMask =
193         CBCGPDateTimeCtrl::DTM_SPIN         |
194         CBCGPDateTimeCtrl::DTM_DATE         |
195         CBCGPDateTimeCtrl::DTM_TIME         |
196         CBCGPDateTimeCtrl::DTM_CHECKBOX     |
197         CBCGPDateTimeCtrl::DTM_DROPCALENDAR |
198         CBCGPDateTimeCtrl::DTM_CHECKED;
199
200     UINT nFlags = CBCGPDateTimeCtrl::DTM_CHECKED | CBCGPDateTimeCtrl::DT
201     //--------------------
202     // Setup date fields:
```

记录标记可以嵌套。当录制时点击新的自动标记时，要录制的堆栈深度将扩展为包括当前方法和该函数所需的深度。

堆栈自动捕获标记	



```
76      /* End - EA generated code for Parts and Ports */
77      /* Begin - EA generated code for  Activities and I
78      public void ClassLib_ActivityGraphWithActionPin()
79      {
```

堆栈标记使您能够捕获应用程序中某个点出现的任何唯一堆栈跟踪；它们提供了从何处调用应用程序中某个点的快速且有用的图片。

要在代码中的所需点插入标记，请右键单击该行并选择 添加堆栈自动捕获标记”选项。

每次调试器遇到标记时，它都会执行堆栈跟踪；如果堆栈跟踪不在记录历史记录中，则将其复制，然后应用程序继续运行。

限制记录深度	您可以使用记录器和断点工具栏上的堆栈深度控件来限制任何记录中的帧深度。

断点和标记窗口

使用断点和标记窗口，您可以在记录执行以生成序列图时对可视执行分析进行控制；例如，您可以：

- 启用、禁用和删除标记
- 将标记作为集进行管理
- 组织标记的显示方式，无论是在列表视图中还是按文件或类分组

访问

功能区	执行>窗口>断点

使用标记集

标记集使您能够将标记创建为命名组，您可以将其重新应用于代码文件以用于特定目的。

您可以单独从断点和标记窗口执行某些操作，但要理解和使用标记和标记集，您还应该在 源代码查看器”中显示适当的代码文件（单击类元素并按 F12）。

访问

功能区	执行>窗口>断点：  工具栏图标

使用标记集

行动	细节
使用示例	您可以创建一组自动标记来记录代码中各种函数的操作，并创建一组 Stack Capture 标记来记录导致调用这些函数的调用序列。 然后，您可以根据每个集合下的记录创建序列图。
创建标记集	要从断点和标记窗口创建标记集，请单击  图标上的下拉箭头并选择 新建集”选项。 显示 新断点标记集”对话框；在 输入新集名称”字段中，输入集的名称，然后单击 保存”按钮。 设置名称显示在 设置选项”图标左侧的文本字段中。 或者，您可以从 设置选项”下拉列表中选择 另存为设置”选项，以制作当前所选设置的精确副本，然后您可以对其进行编辑。
访问集	要访问标记集，请单击 设置选项”图标左侧文本字段上的下拉箭头，然后从列表中选择所需的集。 集合中的标记列在断点和标记窗口中。 您通常会在要捕获动作的点之前加载标记集。 例如，要记录涉及特定对话框的序列，当您开始调试时，您将在调用对话框之前加载该集合；一旦您在应用程序中打开对话框，您标记的操作就会被记录下来。
添加标记以设置	要将标记添加到标记集，请将每个必需的标记添加到 源代码查看器”中相应的代码行。 标记会立即添加到断点和标记窗口中当前列出的任何集合中。 对话框中列出的每个标记在 已启用”列中都有一个复选框；新添加的标记会自动启用，但您可以在检查代码时快速禁用和重新启用标记。
储存	当您创建标记集时，它会立即保存在模型中；任何使用该模型的用户都可以访问该集合。 但是，模型始终存在的默认集是个人工作区，不共享，而是存储在模型外部。

从集合中删除标记	右键单击标记并选择 删除断点”选项。
删除一组	如果您不再需要标记集，请在断点和标记窗口中访问它，然后从 设置选项 ”下拉列表中选择 删除选定集”选项。 您还可以通过选择 删除所有集”选项来清除所有用户定义的标记集；将显示一个提示以确认删除。

注记

- 标记集非常简单和灵活，但是由于模型的任何用户都可以使用它们，因此它们很容易损坏；考虑以下准则：
  - 命名一套时，请在名称中使用您的首字母并尽量表明其用途，以便其他模型
  用户可以识别它的所有者和目的
  - 使用默认以外的集合时，避免过度实验，以免添加
  集合中的许多临时标记
  - 确保您知道断点和标记窗口中暴露了哪些标记集
  因为您可以很容易地在不经意间向集合中添加与代码文件无关的标记
  集是为
  - 在任何集合中，如果您添加了不必保留的标记，请删除它们以保持
  集合的目的；对于默认集尤其如此，它可以快速累积
  冗余的临时标记

控制录制过程

记录和分析窗口使您能够控制记录会话。该控件有一个工具栏和一个历史记录窗口，该窗口在捕获时显示记录历史记录。此窗口中的每个条目代表一个由一个或多个函数调用组成的调用序列。

访问

使用此处概述的方法之一打开 记录和分析"窗口。

您还必须打开执行分析器窗口（ 执行>分析器|分析器脚本"），其中列出了模型中的所有脚本；您必须为录制选择并激活相应的脚本。

功能区	执行 > 工具 > 记录器 > 打开记录器

记录器工具栏

您可以通过 Record &功能工具栏访问用于启动、停止和调节执行分析记录会话的功能。

访问

功能区	执行 > 工具 > 记录器 > 打开记录器 探索 >门户> 显示工具栏 > 记录

纽扣

按钮	描述
	显示用于定义录制会话操作的选项菜单。 • Attach to进程- 启用即使没有分析器脚本，该选项会显示一个对话框，您可以通过该对话框选择要记录的进程和要使用的调试平台；您还可以选择在录制期间使用记录标记集和/或状态机 • 生成序列图表记录- 从执行分析器跟踪生成序列/图形状态图 • 从历史生成测试点图表- 从执行分析器跟踪生成测试域图，可以与测试点功能一起使用 • 从历史生成类类图表- 从执行分析器协作生成一个类图，仅描述记录的动作中涉及的那些类和操作（用例） • 生成调用图from History - 从记录历史生成动态调用图，如您在VEA配置文件工作区执行结构分析布局；这在识别所涉及的唯一调用堆栈方面可能比序列图更有用 • 全部生成——从执行分析器trace中生成序列、测试点和协作类图 • 工件- 在当前选择的包中创建一个工件，在当前元素的浏览器窗口中如果您随后将这个元素工件并双击它，则记录在分析类图上的历史记录工件被复制回 • 从文件加载序列历史 - 选择一个 XML 文件，从中恢复以前保存的记录历史 • 将序列历史保存到文件 - 将记录历史保存到 XML 文件
	选择标记集的记录堆栈深度；也就是说，从录制开始的点开始的帧数。
	启动并记录脚本中描述的应用程序；您可以选择在记录期间使用记录标记集和/或状态机。 该图标在活动分析器脚本为调试时启用。
	在调试会话期间执行当前线程的临时手动记录。 使用调试器的 步骤”按钮使用此函数；由于 step 命令而调用的每个函数都会记录到历史记录窗口。 如果没有进行记录并且您当前处于断点（即调试），则启用该图标。

	在调试会话期间执行临时自动记录。 当您单击此图标时，分析器开始记录并且在程序结束、停止调试器或单击停止图标之前不会停止。 如果没有进行记录并且您当前处于断点（即调试），则启用此图标。
	节一个函数，在 History 窗口中记录函数调用，然后退出。 仅对手动录制启用。
	停止记录执行跟踪。
	显示 同步模型”对话框，您可以通过该对话框将模型与记录配置文件操作期间生成的代码文件同步。

使用记录历史

您可以使用 记录和分析”窗口上下文菜单对或从记录会话的结果执行许多操作。

选项

选项	行动
来电显示源	在源源代码查看器中显示调用序列的方法的源代码。
显示源for Callee	在源源代码查看器中显示被序列调用的方法的源代码。
生成图表序列的图形	为记录历史中选择的单个序列生成序列图。
图表生成序列	生成包含记录历史中所有序列的序列图。
清除	清除当前显示在 记录和分析”窗口中的记录历史。
将记录历史保存到文件	将记录历史保存到 XML 文件。 将显示A浏览器窗口，您可以在其中指定 XML 文件的文件路径和名称。
从文件加载记录历史	从 XML 文件加载以前保存的记录历史。 将显示A浏览器窗口，您可以在其中指定要加载的 XML 文件的文件路径和名称。
禁用所有呼叫	禁用 记录和分析”窗口中列出的每个呼叫。
禁用此调用	禁用选定的呼叫。
禁用此方法	禁用所选方法。
禁用这个类	禁用选定的类。
禁用此调用之外的所有调用	禁用 记录和分析”窗口中列出的每个呼叫，所选呼叫除外。
启用所有呼叫	启用 记录和分析”窗口中列出的每个呼叫。
启用这个调用	启用选定的呼叫。
启用此方法	启用所选方法。
启用类	启用选定的类。
帮助	显示记录和分析窗口的帮助主题。

开始记录

当您将执行流程记录为序列图时，您可以通过选择 记录和分析”窗口工具栏上的 记录”图标来开始记录。 录制”对话框显示，录制选项设置为默认值；即当前的Breakpoint and Markers Set，当前分析器中定义的脚本，recording mode as basic。

访问

功能区	执行 > 工具 > 记录器 > 打开记录器： 

记录对话框选项

字段/按钮	细节
记录集	记录标记确定记录的内容。 如果您有要使用的记录集，请单击下拉箭头并选择它。
附加过滤器	调试器使用过滤器从记录历史中排除匹配的函数调用。记录过滤器在分析器脚本定义。 在 附加过滤器”字段中，您可以为此特定运行添加其他过滤器。如果您指定多个过滤器，请用分号分隔它们。
基本记录模式	在基本模式下，调试器在遇到适当的记录标记时记录程序进行的函数调用的历史记录。
跟踪命名类的实例	在跟踪实例模式下，调试器还捕获您指定的类实例的创建。然后它将该信息包含在历史记录中。然后，生成的序列图可以显示该类的每个实例的类，在适当的情况下，函数调用链接到生命线。
跟踪状态Transitions	记录还可以使用指定的状态机图捕获状态变化。状态机图必须作为类的子级存在。 执行分析器捕获类的实例，并在当前记录序列中的函数返回时计算每个实例的状态。
确定	单击此按钮以启动调试器。

节通过函数调用

可以通过单击 记录和分析”窗口工具栏上的 节通过”按钮来执行 节通过”命令。
或者，按 Shift+F6 或选择 执行 >运行>节输入”功能区选项。
'节Through' 命令导致执行'节Into' 命令；如果检测到任何函数，则该函数调用会记录在 历史记录”窗口中。
然后调试器退出，并且可以重复该过程。
此按钮使您无需实际进入函数即可录制通话；该按钮仅在断点和手动记录模式下启用。

嵌套记录标记

当第一次遇到记录标记时，记录从当前堆栈帧开始并继续直到帧弹出，记录额外的帧，直到记录工具栏上定义的深度。考虑这个调用序列：

A -> B -> C -> D -> E -> F -> G -> H -> I -> J -> K -> L -> M -> N -> O -> P -> Q - > R -> S -> T -> U -> V -> W -> X -> Y -> Z

如果您在K处设置一个录音标记并将录音深度设置为 3，这将记录通话序列：

K -> L -> M

如果您还想将调用 X、Y 和 Z 记录为序列图的一部分，您可以在 X 处放置另一个记录标记，分析器将记录：

K -> L -> M -> X -> Y -> Z

但是，当 XYZ 分量的记录结束时（弹出 X 帧），当重新进入 KLM序列的第M帧时，记录将恢复。使用这种技术可以帮助记录图表中的信息由于堆栈深度而被排除在外，并且它可以让您聚焦于要捕获的特定区域。

生成图表序列

本主题描述了您可以对执行分析会话的记录执行哪些操作。

访问

功能区	执行 > 工具 > 记录器 > 打开记录器

参考

行动	细节
生成图	在浏览器窗口中选择适当的包，用于存储序列图。 要从所有记录的序列创建图表，请执行以下任一操作： • 单击 记录和分析”窗口工具栏中的 记录菜单”图标 ()，然后选择 图表从记录生成序列”选项，或 • 右键单击窗口主体并选择 生成序列图表选项 要从单个序列创建图表，请执行以下任一操作： • 单击 记录和分析”窗口工具栏中的 记录菜单”图标 ()，然后选择 图表从记录生成序列”选项，或 • 右键单击序列并选择 从选定序列生成图表”选项
将记录的序列保存到 XML 文件	单击序列，单击 记录和分析”窗口工具栏中的 记录菜单”图标 ()，然后选择 将序列历史保存到文件”选项。
访问现有的序列XML 文件	任何一个： • 单击 Record & Analyze 窗口工具栏中的  ，然后选择 Load序列 from文件”选项，或 • 右键单击屏幕的空白区域，然后单击 加载序列从文件”选项 将显示 窗口”对话框，从中选择要打开的文件。

使用到

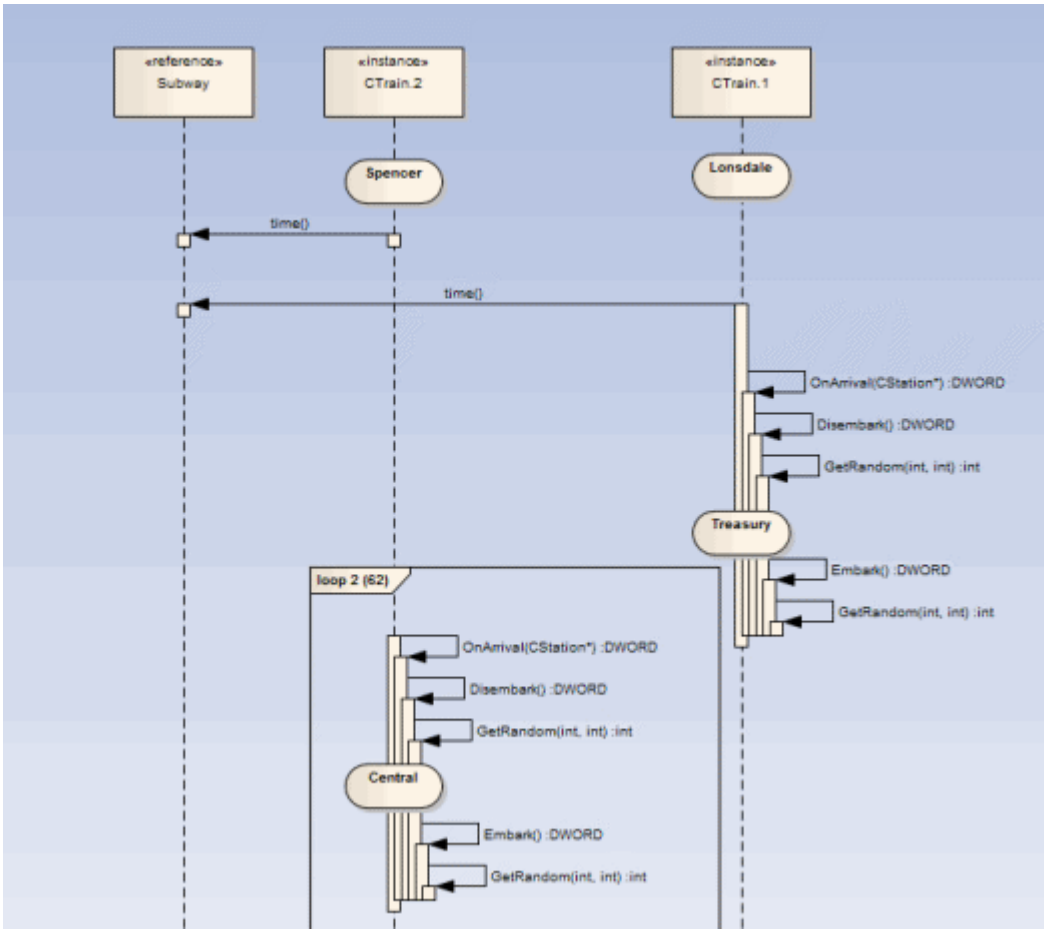
- 从记录的执行分析会话中生成序列图，用于：
- 所有记录的序列或
- 会话中的单个序列
- 将记录的序列保存到文件
- 检索保存的记录并将其加载到 记录和分析”窗口

报告状态Transitions

本节介绍如何生成显示程序执行时状态转换的序列图。

使用到

在程序执行时报告用户定义的状态转换的生成序列图（如生成的示例图所示）



话题
在要上报的类下创建一个状态机。
针对每个状态设置约束以定义要报告的状态变化。

上报状态机

执行分析器可以记录一个序列图，我们知道的。您可能不知道的是，它可以同时使用状态机来检测沿途可能发生的状态转换。这些状态在object生命线上的时间点表示。生命线上的转变也很明显。任何无效或非法的过渡都将用红色边框突出显示。看一看。

进程

首先你为适当的类元素模型一个状态机。

然后，您可以使用每个状态的 约束”选项卡来编写定义每个状态的状态。

这些简单的表达式是使用来自类模型和实际代码库的属性名称形成的。它们不是 OCL 语句。每个表达式应出现在单独的行上。

m_strColor == "蓝色"

然后使用 Recorder 窗口启动调试器。

记录器窗口运行按钮与其他调试器工具栏上的按钮不同。

如果您不知道状态机名称，记录器窗口将允许您浏览状态机机。 '状态转移

' 对话框显示整个模型的状态机列表，您可以在其中找到并选择适当的图表（参见示例）。

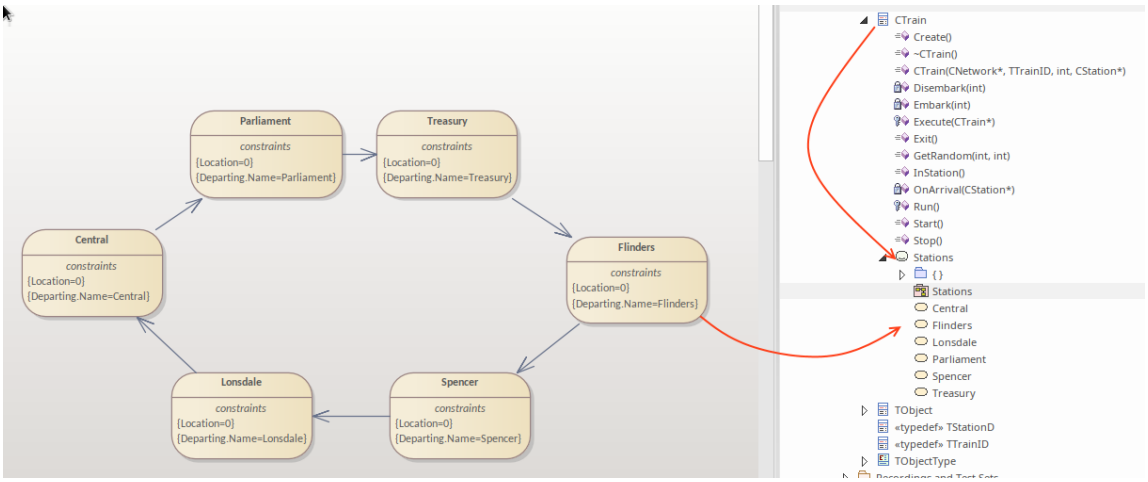
当你生成序列图时，它不仅描绘了序列，而且描绘了序列中各个点的状态变化；参与检测过程的每个类实例都有自己的生命线显示。

示例

Stations状态机显示了墨尔本地铁环路系统内的不同状态。

运行在地铁网络上A列车可以停靠在状态机上表示的任何一个车站。

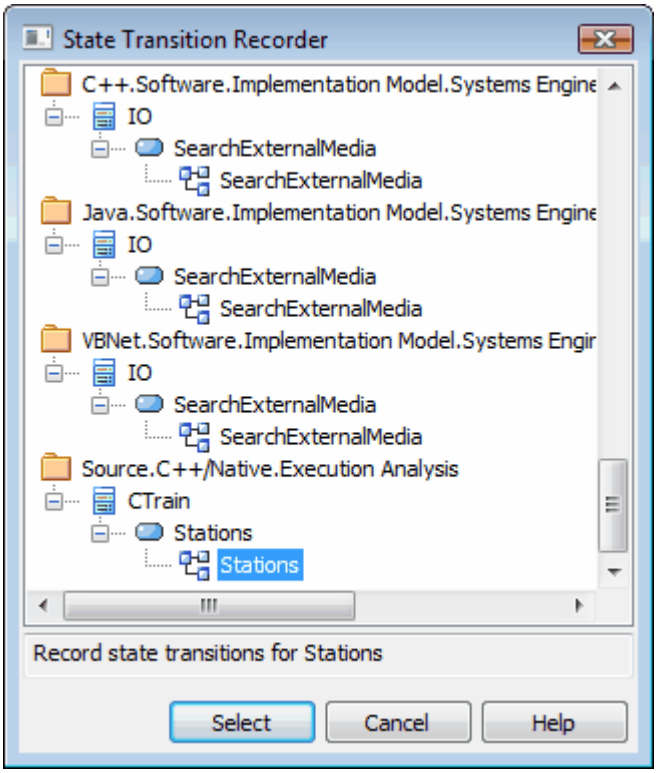
Stations状态机是CTrain类的子对象。



当您浏览 状态转移

”中的图表时转移

Recorder'对话框，层次结构只显示根包、父类和子SubMachine和图表；没有列出其他模型组件。


State Transition Recorder
C++.Software.Implementation Model.Systems Engine
IO
SearchExternalMedia
SearchExternalMedia
Java.Software.Implementation Model.Systems Engine
IO
SearchExternalMedia
SearchExternalMedia
VBNet.Software.Implementation Model.Systems Engir
IO
SearchExternalMedia
SearchExternalMedia
Source.C++/Native.Execution Analysis
CTrain
Stations
Stations
Record state transitions for Stations
Select
Cancel
Help

记录和映射状态修改

本主题讨论如何在一个类下的状态机中针对每个状态设置约束，来定义要记录的状态变化。

示例

这个 状态"对话框的示例是针对称为 属性"的状态； 约束"选项卡打开，显示状态如何与类状态关联。

状态可以A单个约束或多个约束来定义；在示例中，议会状态有两个约束：

约束的值只能针对元素、枚举和string类型进行比较

CXTrain类有一个名为 Location 的int类型的成员，以及一个名为名称类型 CString 的成员；此约束的含义是，在以下情况下，此状态被评估为True ：

- CXTrain类的一个实例存在并且
- 它的成员变量 Location 的值为 0 并且
- 成员变量名称的值为议会

约束的运营商

您可以在约束上使用两种类型的运算符来定义状态：

- 逻辑运算符 AND 和 OR 可用于组合约束
- 等价运算符 {= 和 !=} 可用于定义约束条件

除非另有说明，否则状态的所有约束均受 AND 运算；您可以改为对它们使用 OR 操作，因此您可以将示例中的约束重写为：

位置=0 或

位置=1 并且

Departing.Name!=中央

以下是使用等价运算符的一些示例：

Departing.Name!=Central AND

位置！=1

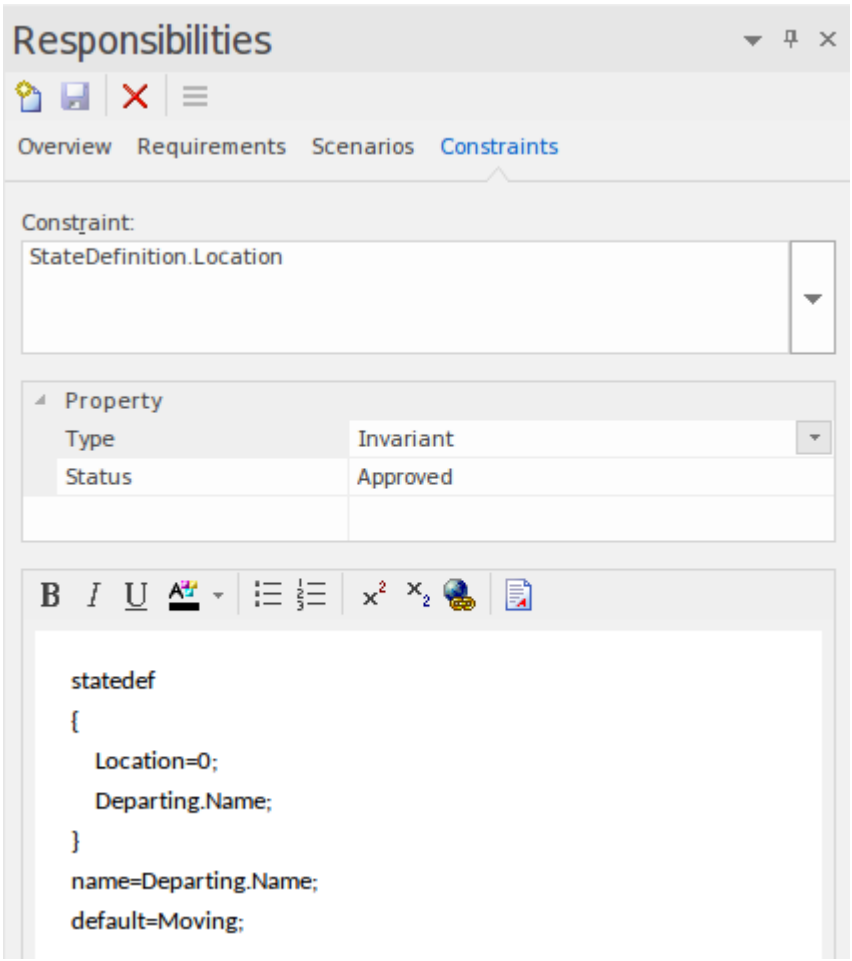
注记

- 字符串周围的引号是可选的；在确定约束的真实性时，字符串的比较始终区分大小写

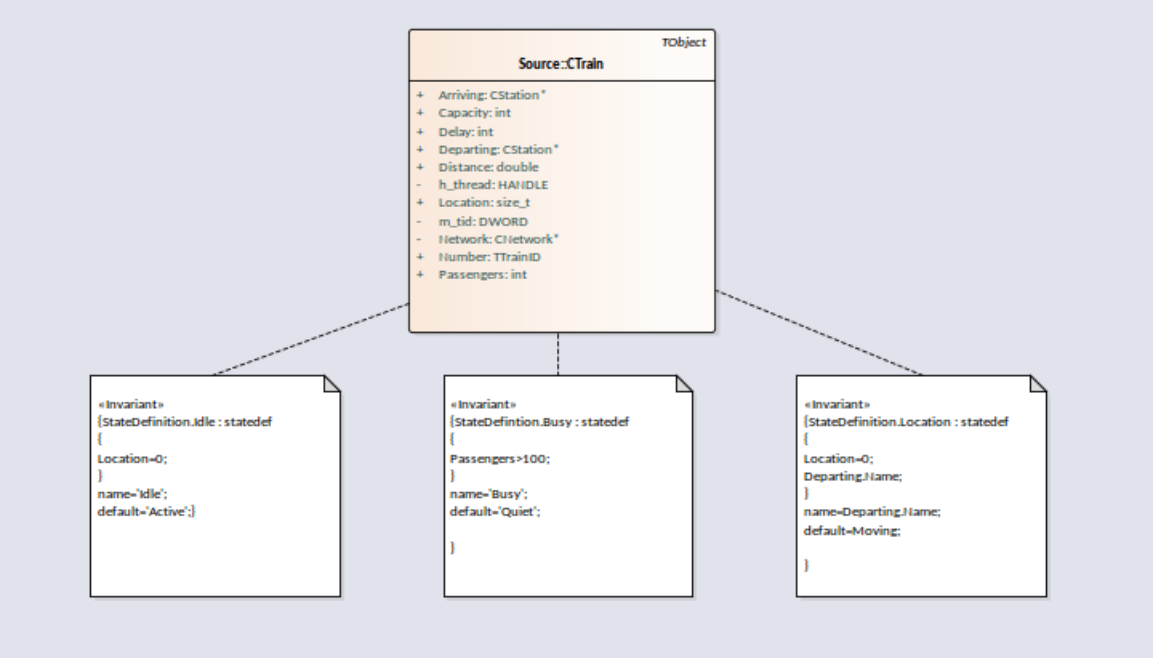
状态分析器

状态分析器是一个可以分析、检测和记录类实例状态的特征。特征通过组合状态定义（在类上定义为约束）和称为状态点的标记来工作。它适用于执行分析器支持的任何语言，包括 Microsoft.NET、Mono、 Java和原生 C++。

我们首先选择一个类并编写我们的状态定义。



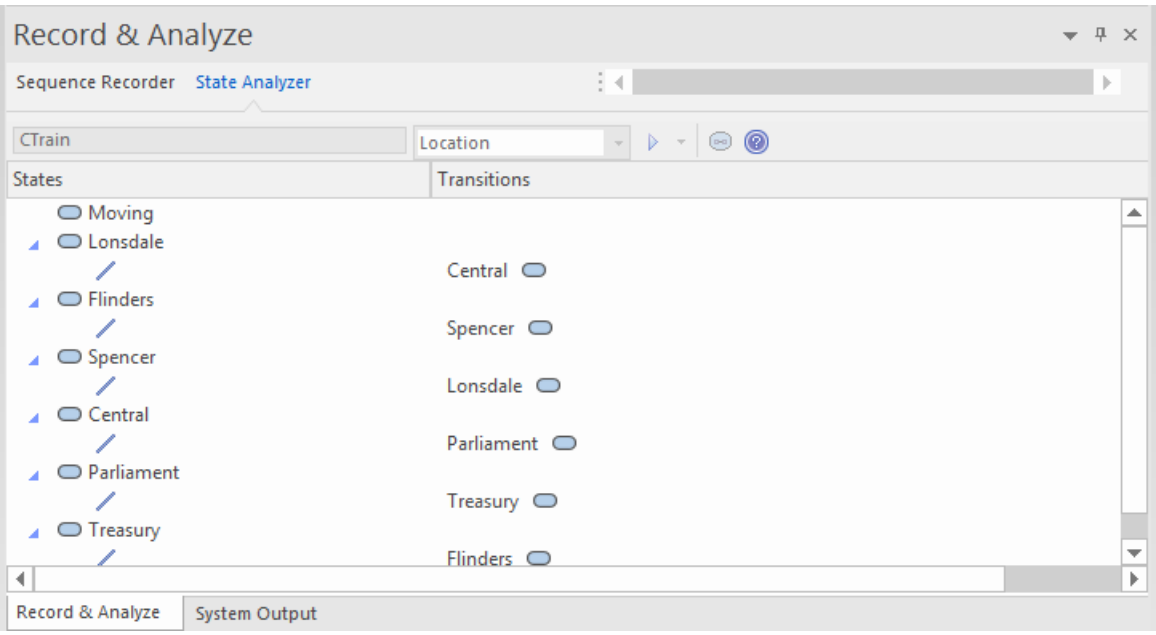
我们可以通过将类放在图表上并链接到本身链接到特定状态定义约束的类注记来获得我们定义的所有状态定义的图片。我们将在后面的部分中解释如何做到这一点。



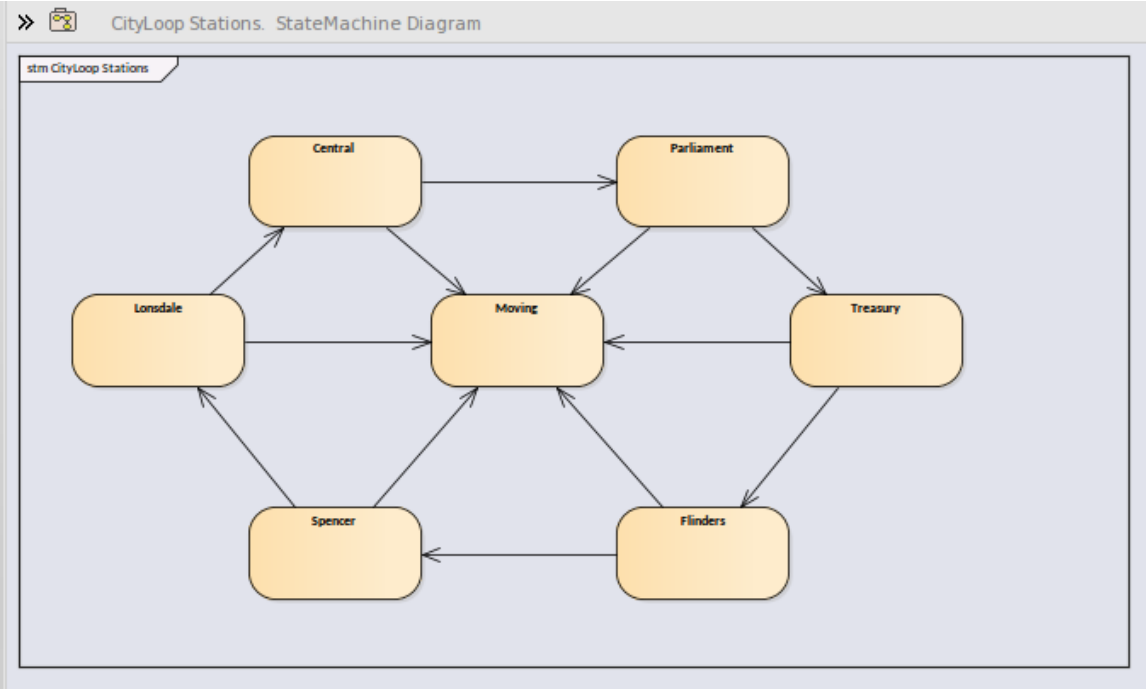
通过在相关源代码中放置一个或多个标记来设置状态点。

```
73 DWORD   CTrain::OnArrival( CStation* S)
74 {
75     Departing = S;
76     Location = 0;
77     Delay = (Disembark(GetRandom()) + Embark(GetRandom()));
78     DWORD ScheduleTime = Network->TimeAtStation(Departing);
79     if(Delay > (int)ScheduleTime)
80         return Delay;
81     return ScheduleTime;
82 }
```

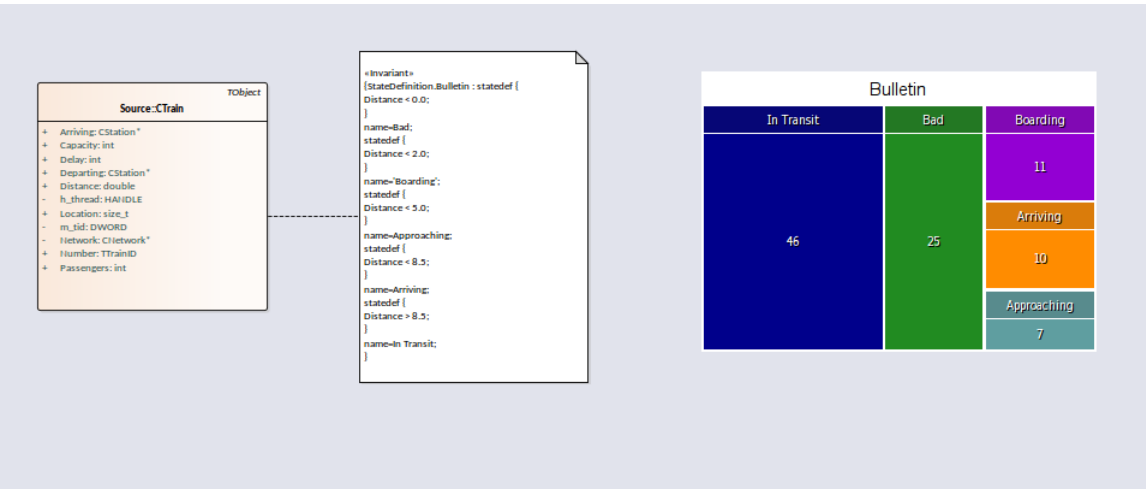
要分析的程序运行使用状态分析器控件运行。当执行分析器遇到任何状态点时，分析类的当前实例。在实例的值域与状态定义匹配的情况下，记录一个状态。每次实例变化时，都会检测到新的状态。控件列出发现的每个状态。在每个状态下，控件列出了由类的实例进行的到其他状态的离散转换集。



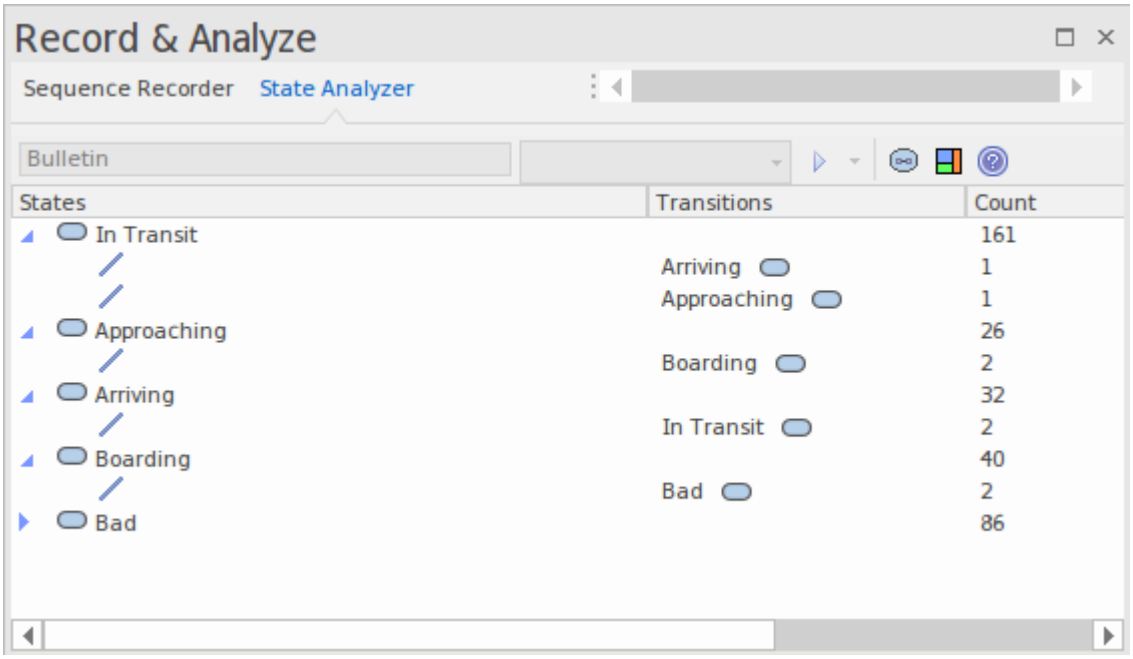
该信息可用于创建状态机。



使用相同的信息，我们可以轻松生成热图。这个例子展示了一个'Train'类，它的'Bulletin'状态定义（作为一个链接的注记），以及它产生的热图。地图中的数字是百分比。从地图中我们可以观察到火车在 46% 的时间处于 "在途"状态。



这是生成我们的热图的 "公告"状态定义的分析。



访问

功能区	执行 > 工具 > 记录器 > 打开记录器 >状态分析器 设计>元素>编辑>约束

状态定义

状态定义由类元素的约束属性组成。约束类型应命名为*StateDefinition.name* ，其中 'name' 是您选择的定义标题。 These titles are listed in the combo box of the状态分析器whenever a类is selected.在运行程序之前，您可以从此组合框中选择一个定义。我们示例中的状态定义名为 状态"。它根据 CTrain类实例的位置定义状态。

状态定义由一个或多个规范组成。每个状态规范都以关键字 "statedef"开头，然后是一个或多个语句。语句定义描述状态的约束，以及可选的变量，其值可用于命名状态。语句用大括号括起来，并以分号结尾，如下所示：

状态定义 {

位置=0；

出发。名称；

}

使用变量命名状态

在此示例中，"Location"是一个常量，"Departing.name"是一个变量。附加语句遵循约束并指示要从变量值分配的状态名称。这是命名指令的定义。

状态定义 {

位置=0；

出发。名称；

}

名称=出发。名称；

使用文字命名状态

在此示例中，状态定义仅包含常量，并且状态使用文字命名。

状态定义 {
位置=100；
}
名称='中央';

定义多个状态规范A单个状态定义。

状态定义 {
乘客> 100；
}
名称=忙；
状态定义 {
乘客>= 50；
}
名称=安静；
状态定义 {
乘客<50；
}
名称=非常安静；
状态定义 {
乘客= 0;
}
名称=空闲；

默认状态

状态定义可以指定A默认的 ć̋atch all"状态，当没有其他状态为真时，该状态将描述状态的状态。您使用类似于以下的语句为定义定义默认状态：

状态定义 {
位置=0；
出发。名称；
}
名称=出发。名称；
默认=移动；

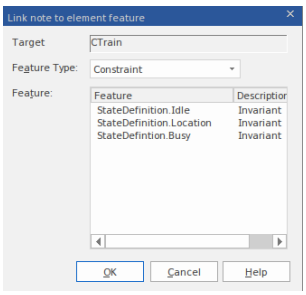
在此示例中，在执行过程中，检测到具有非零 位置"属性的任何实例都将被记录为处于 移动"状态。

您可以通过禁用状态分析器工具栏下拉菜单上的 包括默认状态”选项来选择排除默认状态的记录。这将排除到正在记录的任何 默认”状态的转换。

在显示状态的类元素上创建注记定义定义

本节介绍如何创建显示为类定义的所有状态定义的类图。

行动

显示类图	打开现有的类图或创建一个新的。
创建指向类元素的链接	将感兴趣的类作为链接拖到图表上。
创建一个注记元素	在图表上创建一个注记元素并将其链接到类。
将注记链接到状态定义	选择注记和类之间的链接，并使用其上下文菜单，选择 将注记链接到元素特征”选项。
选择要在注记上显示的定义 	在元素对话框中，从下拉组合中选择 约束”。将列出任何已定义的状态定义供您选择。
重复	对类上的任何其他状态定义重复过程。

同步

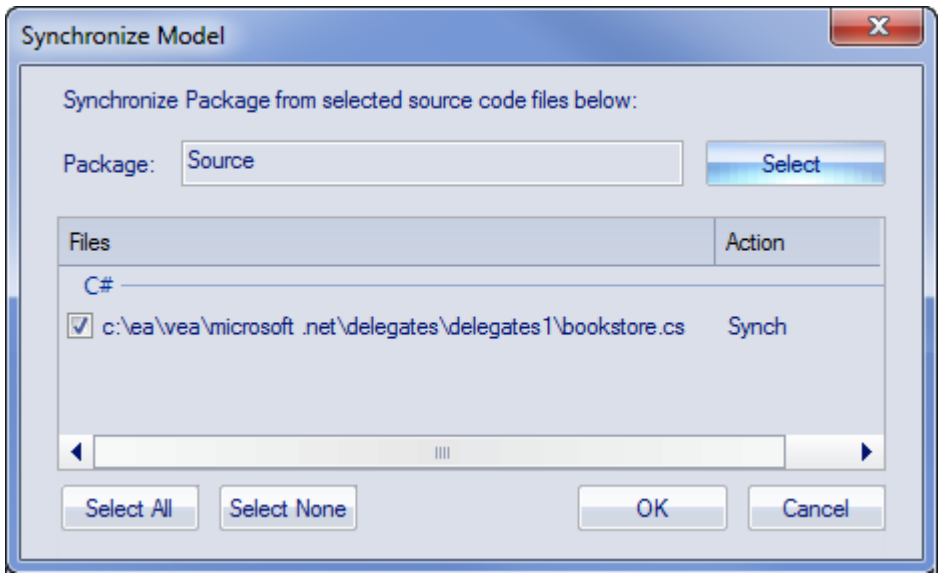
记录产生了许多资产，记录历史是主要的。记录还标识了一组源代码文件。该套件可用于制作类图和测试域图，也可用于同步您的模型。

同步模型A图表元素和类模型之间提供快速准确的导航。

访问

功能区	执行 > 工具 > 记录器 > 打开记录器 > 工具栏  按钮
上下文菜单	右键单击 记录和分析”窗口 |将模型与源代码同步

同步模型



字段/按钮	行动
包	单击 选择”按钮并选择要对代码文件进行逆向工程的目标包。
文件/行动	列出在一个或多个记录期间识别的文件。每个文件旁边都列出了相应的操作。
全选	单击此按钮以选中 文件”列表中每个文件的复选框。
选择无	单击此按钮可清除 文件”列表中每个文件的复选框。
确定	单击此按钮开始操作。将显示同步进度。
取消	单击此按钮可中止同步并关闭对话框。

样本

Enterprise Architect使您能够轻松导入完成的示例模型（包），包括所有必要的模型信息、代码和构建脚本。这些示例模式使探索和试用可视化执行分析器简单。您可以为以下内容生成示例模型：

- Java
- 微软.NET
- 微软 C++
- PHP 阿帕奇

访问

功能区	开发 >源代码 > 从模式创建 > VEA 示例

显示样本

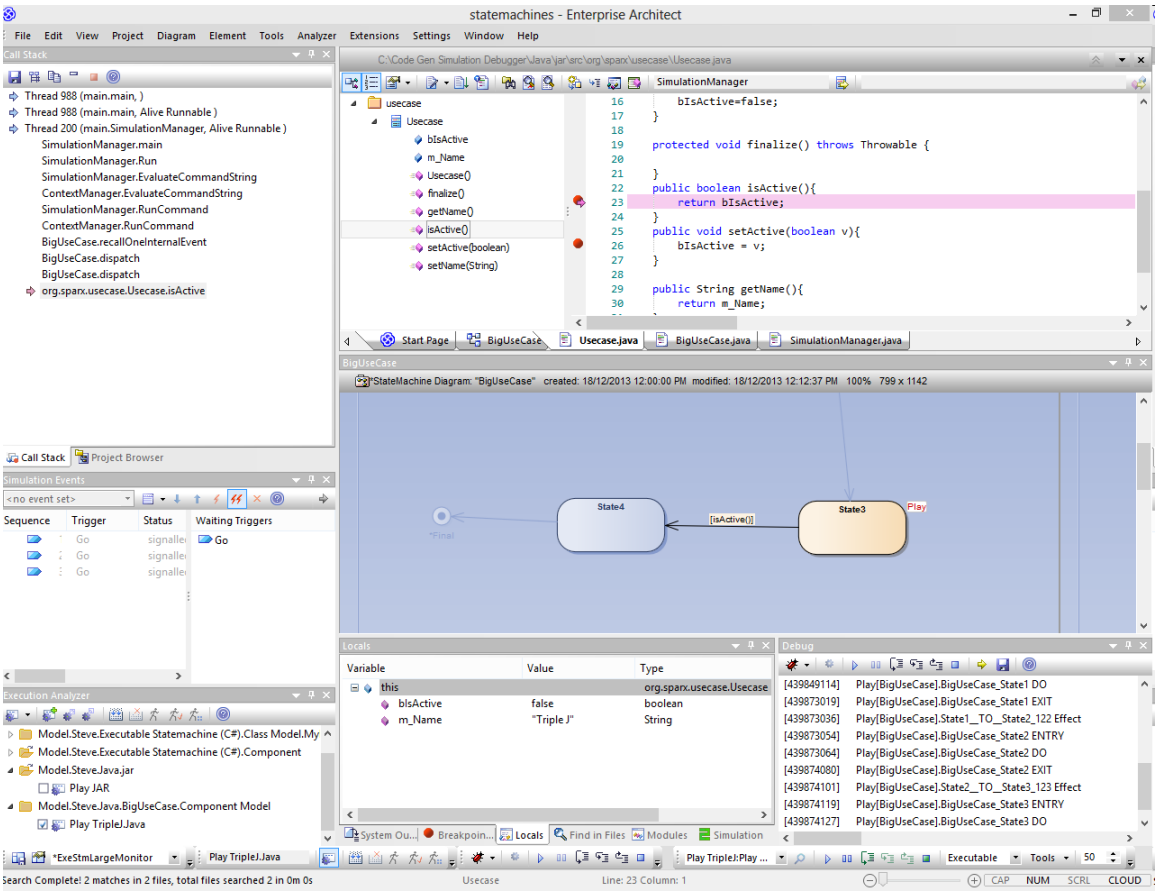
字段	行动
技术	选择合适的技术。
名称	显示可用于所选技术的示例；选择需要导入的样本。
描述字段	显示所选样本的描述。
目标文件夹	浏览并选择加载示例源代码的目录。
使用本地路径	启用选择现有的本地路径来放置源代码；将 "目标文件夹"字段更改为下拉选择。
编译器命令	显示所选技术的默认编译器命令路径；您必须： • 确认可以在此路径中找到编译器，或者 • 编辑编译器位置的路径
编辑本地路径	许多 VEA 示例使用本地路径指定其编译器。 第一次使用任何示例时，您必须单击此按钮以显示 "本地路径"对话框，您可以在该对话框上检查并 - 如有必要 - 更正指向正确编译器位置的本地路径。

注记

- 如果需要，您可以通过将文件添加到安装Enterprise Architect的 AppSamples 目录来定义自定义示例；顶级目录被列为技术并且可以包含一个图标文件来自定义为技术显示的图标
  下面的目录被定义为模式列表中的组；该模式由四个具有匹配名称的文件定义：压缩文件 (.zip)、XMI 文件 (.xml)、配置文件 (.cfg) 和可选图标 (.ico)

- 配置文件支持以下字段：
  - [provider], [language], [platform], [url], [description], [version] - 全部显示在'description'中场地
  - [xmirootpaths] - 导出的XMI中源代码的根路径；这被替换为用户应用应用程序模式时选择的目标文件夹

编译和调试



Enterprise Architect建立在其已经卓越的代码生成、图表和设计能力之上，具有完成的工具来构建、调试、可视化、记录、测试、配置文件以及以其他方式构建和验证软件应用程序。该工具集与建模和设计功能密切相关，并提供了一种独特而实用的方法来从模型构建软件并保持模型和代码同步。

Enterprise Architect帮助您定义链接到模型包的 分析器脚本”，描述如何编译应用程序、使用哪个调试器以及其他相关信息，例如模拟命令。分析器脚本将代码链接到Enterprise Architect中的构建、调试、测试、分析和部署功能的核心配置项。

作为工具集能力的衡量标准，应该注意的是， Enterprise Architect实际上是在Enterprise Architect开发环境中完全构建、调试、分析、测试和构建的。许多高级调试工具（例如行动点）已被开发用于解决构建大型复杂软件应用程序（例如Enterprise Architect ）中固有的问题，并且Sparx Systems开发团队每天都会使用这些工具。

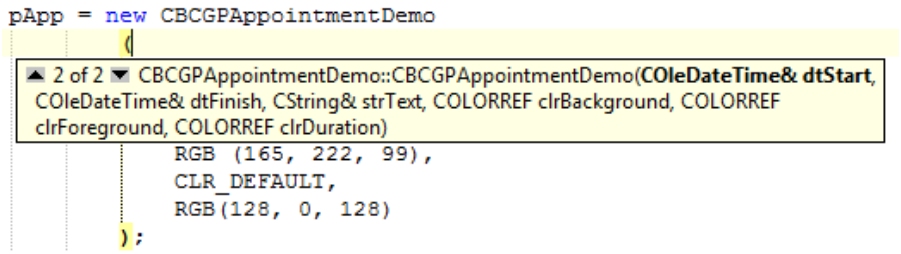
建议新用户花时间充分了解分析器脚本的使用以及他们如何将模型与代码、编译器和构建软件所需的其他工具联系起来。

集成模型和代码

模型驱动工程是一种现代的软件开发方法，承诺更高的生产力和更高质量的代码，从而使系统更快地进入市场并减少故障。使这种方法引人注目的是架构和系统设计能够在模型中进行描述和维护，然后生成编程代码和模式，这些代码和模式可以与模型同步并在模型中可视化。

Enterprise Architect的模型驱动开发环境(MDDE) 支持这种方法并提供一组灵活的工具来提高生产力和减少错误。其中包括在模型中定义架构和设计、从这些模型生成代码、将代码与模型同步以及在复杂的代码编辑器中维护代码的能力。也可以导入源代码或二进制文件，用户可以记录和记录预- 现有或最近开发的代码。分析器脚本帮助您描述如何构建、调试、测试和部署应用程序。

功能	描述

模型驱动开发	与传统的编码驱动周期相比，模型驱动开发提供了更强大、更易于访问和更快的开发周期。 A良好，与源代码构建、运行、调试、测试和部署密切相关的能力，提供丰富、易于导航和易于理解的目标架构可追溯性模型，与使用案例、组件和其他模型的链接，以及能力轻松记录和记录预先存在或最近开发的代码，使Enterprise Architect的开发环境具有独特的有效性。 Enterprise Architect结合了行业标准的智能编辑、调试器和建模语言。
模型驱动开发环境（MDDE）	MDDE 提供了设计、可视化、构建和调试应用程序的工具： • UML技术和工具到模型软件 • 用于生成/逆向工程源代码的代码生成工具 • 导入源代码和二进制文件的工具 • 支持不同编程语言的代码编辑器 • 智能感知帮助编码 • 分析器使用户能够描述如何构建、调试、测试和部署应用程序的脚本 • 与Java 、.Net、Microsoft C++ 等编译器集成 • Java 、 .NET 、Microsoft C++ 等的调试功能 • 高级可视化、记录、检查、测试和分析能力 

服务

Enterprise Architect提供了两种服务来促进远程脚本执行和远程调试。这些服务主要支持在 Linux 上运行的 Enterprise Architect ，以允许用户运行原生 Linux shell 脚本和调试 Linux 程序。卫星服务支持分析器脚本，代理人服务支持调试。

访问

功能区	执行 > 工具 > 服务

卫星服务

卫星服务负责在其运行的机器上执行分析器脚本。该特征可以帮助 Linux 用户绕过Wine直接执行本机 Linux 程序和 shell 命令。该服务可以从功能区进行管理，也可以运行于终端运行。

Linux 外壳

Enterprise Architect使用的默认 shell 是 "bash"。要覆盖Enterprise Architect使用的 Linux Shell，请打开 Linux 终端，运行 "wine regedit"并将string值添加到此注册表项：

HKEY_CURRENT_USER\Software\ Sparx Systems \EA400\EA\Options

在哪里：

- 键名："LINUX"
- 关键值：路径

*path*是 shell 程序 "/bin/bash"的 Linux 路径，例如。

权限

在 Linux 下，您必须检查服务程序是否具有适当的权限。这些程序位于Enterprise Architect安装文件夹下的子目录 "EA/x86/linux"中。选择此目录中的每个程 都具有所有者的执行权限。

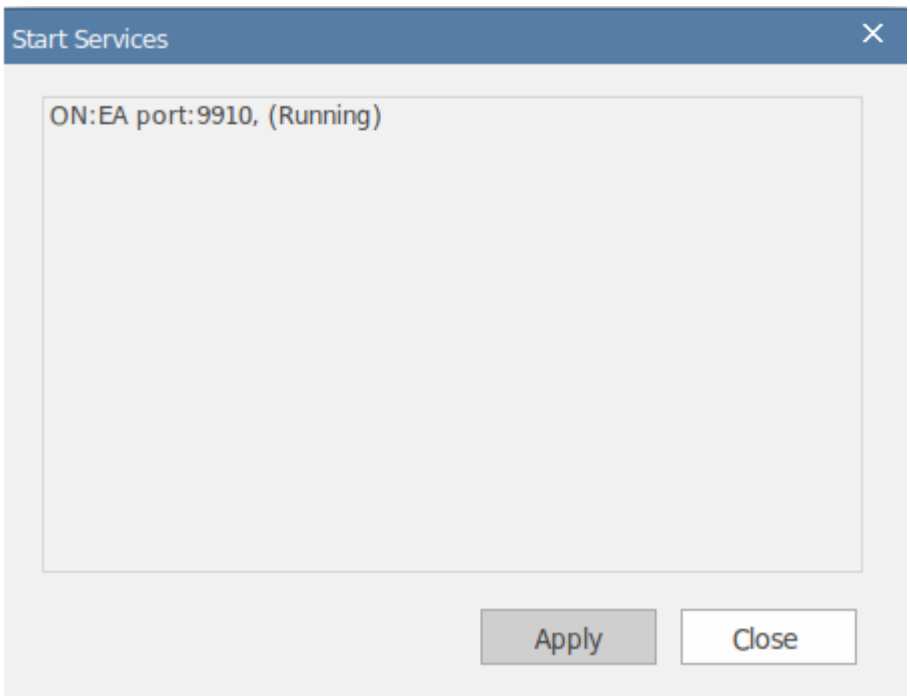
注记

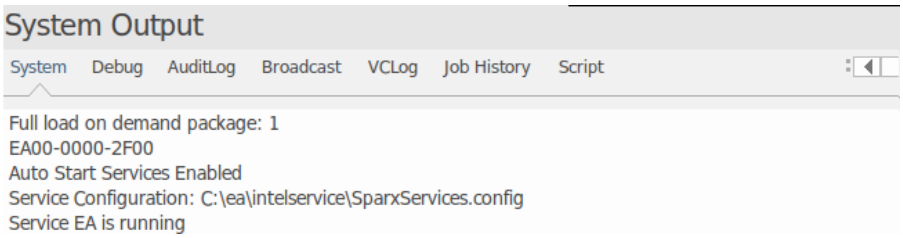
- Enterprise Architect统一版和终极版中启用了卫星服务

代理人服务

代理人服务负责管理Enterprise Architect的 GDB 调试器的调试会话。该服务允许Enterprise Architect用户调试 Linux 程序。可以从功能区管理服务。它也可以运行于终端运行。

服务菜单

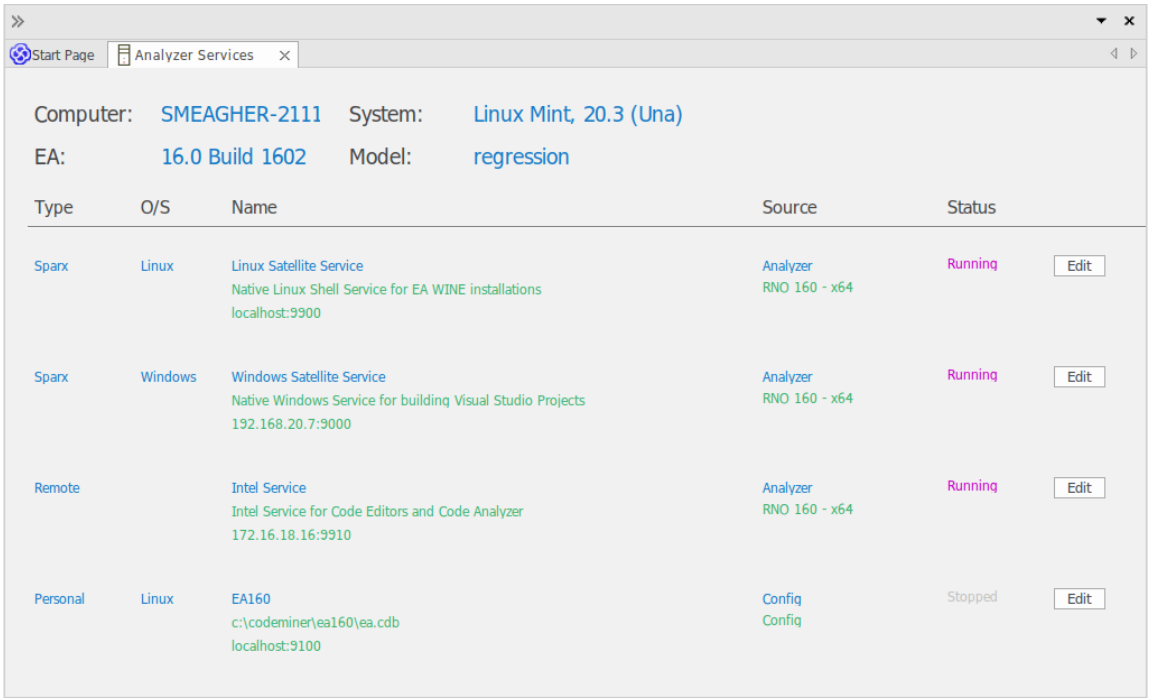
选项	描述
所有服务的视图状态	显示一个视图，列出配置文件中状态的每个Enterprise Architect服务的状态及其状态。
卫星服务	
开始	启动服务。该服务侦听任何分析器配置的卫星端口脚本Page。
停止	停止服务。
测试	测试卫星服务的状态，是否运行。
代理人服务	
开始	启动服务。该服务代理人端口配置的分析器脚本页面。
停止	停止服务。
测试	测试状态代理人的状态，是否正在运行。
代码矿工服务	
开始	该选项读取当前的Service配置文件，启动配置为运行的服务，停止运行未配置为运行的运行。 A以下情况下配置服务： 1. 它在配置文件中命名。 2. 它具有属性状态：ON。 
停止所有	此选项停止当前正在运行的所有服务。

编辑配置文件	此选项提示要使用的服务配置文件，然后在Enterprise Architect文本编辑器中打开该文件。系统会记住文件所在的位置。 31 # <number> - digits 32 # ---------- 33 # 34 { 35 name=project1, 36 status=ON, 37 lazyload=true, 38 port=9910, 39 allow=localhost, 40 network=local, 41 autoupdate=true, 42 show=true, 43 logoutput=c:\My Documents\project1.txt, 44 loglevel=information|warning|error, 45 database=c:\My Documents\project1\project1.cdb 46 } 47
自动开始使用 EA	此选项会在模型打开时自动启动具有 状态：ON"属性的服务。  当模型打开时，此处记录到系统输出窗口的消息表明该服务已经在运行。
关闭时自动停止	此选项在Enterprise Architect关闭时自动停止运行服务。

分析器服务窗口

分析器服务窗口显示以下各项的状态：

- 由主动分析器脚本的服务（通过'执行>工具>分析器'功能区选项维护）
- 使用服务配置文件单独管理的任何本地服务（可从 执行 > 工具 > 服务 >代码矿工服务 > 编辑配置文件”功能区选项中获得）



例如，该窗口让您可以一目了然地看到您正在使用的英特尔服务，或者您用于在Enterprise Architect中工作的窗口服务正在运行。

窗口中的所有数据都是只读的，但您可以通过单击 编辑”按钮来编辑服务。这将显示分析器脚本编辑器窗口的 私人选项 - 服务”页面，您可以根据需要对其进行更新。

访问

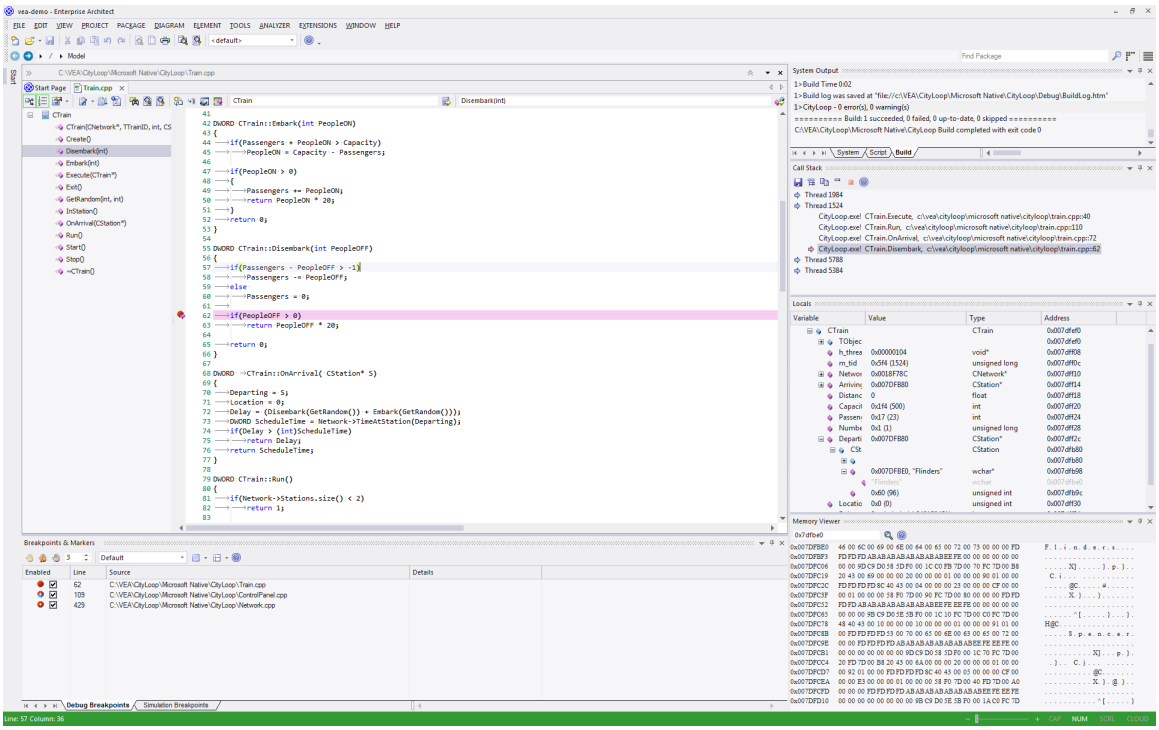
功能区	执行 > 工具 > 服务 >视图所有服务的状态 >开始> 所有窗口> 执行 > 分析 |分析器服务
键盘快捷键	Alt+4 |分析|分析器服务

分析器服务窗口字段

字段	描述
电脑	运行服务的工作站。
系统	工作站运行的操作系统。

EA	您正在使用的Enterprise Architect版本的发行号和内部版本号。
模型	您当前正在使用的模型的名称。
类型	服务类型。列为 "Sparx"类型的服务是Enterprise Architect Services。
O / S	运行服务的操作系统。
名称	服务的名称和描述。 • 当应用程序在WINE下的 Linux 上运行时，Linux卫星服务由Enterprise Architect自动管理 • 窗口卫星服务是一项可选服务，通常用于WINE安装中，以帮助在远程窗口机器（如 VM（虚拟机））上构建 Visual Studio 项目；该服务仅在窗口窗口本地进行管理
源	定义服务的脚本组或配置文件。
状态	服务的状态。 当您第一次打开窗口时，每个服务最初都有 状态测试"作为系统评估服务。状态然后更改为适当的值，例如 正在运行"或 已停止"。
编辑	单击此按钮可查看或编辑服务定义。

调试



Enterprise Architect不仅仅是一个绘图工具——它还提供了您在 IDE 中可能期望的每个特征。提供了适用于许多主要平台的综合调试环境和工具。在建模工具中集成调试功能允许其作者开发、构建和管理代码。在集成模型中工作和协作使行动变得重要，并且每一个行动都以使用其他工具链无法实现的方式负责。

特征

速度

Enterprise Architect中的调试器很快！逐步完成程序不会花费一整天的时间。

记录程序无需手动执行即可执行。

支持

- C++、C 和 Visual Basic
- 微软.NET , ASP.NET WCF
- Java ，使用套接字传输（JDWP）或内存模型（JVMIT）
- 模拟器或设备上的 Android
- JavaScript , VBScript 和 JScript
- Apache Web 服务器上的 PHP 脚本
- 在窗口中使用Enterprise Architect的远程 Linux GDB 进程
- 仿真- UML和 BPMN 中的调试仿真
- 可执行状态机——调试执行状态机

隔离

调试器在Enterprise Architect的进程外运行，将其与副作用隔离开来。

效率

启动和停止调试器既快速又轻松。它不会阻止你。设计为响应式 UI，主 UI 线程与不属于其职责的职责隔离。

生产率

从建模切换到需求，从提出变更请求到跟踪整个组织共享的模型中的代码变更，再到分析最近的代码变更。一站式工具。

注记

- 可视化执行分析器特征调试和记录特性不适用于 Oracle 的Java服务器平台 Ẅeblogic”

运行调试器

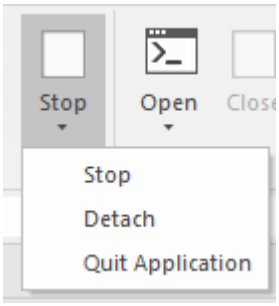
Enterprise Architect提供了许多方法来启动和控制调试会话。有主调试窗口，以及调试工具栏和 执行”功能区中的 运行”面板。每次运行调试会话时最好显示调试窗口，因为这是捕获所有调试输出的地方。

访问

功能区	执行 >运行>开始 执行 > 工具 >调试器>开始调试
键盘快捷键	Alt+8（显示调试窗口） F6（开始执行被调试的应用程序）
工具栏	探索 >门户> 显示工具栏 >调试

使用调试窗口

行动	细节
开始调试器	当分析器脚本已配置为支持调试时，您可以通过以下方式启动调试器： • 从功能区中，选择 执行 >运行>开始>运行” • 从功能区中，选择 执行 > 工具 >调试器>开始调试” • 在 调试”工具栏上，单击  按钮，或 • 按 F6 您也可以通过 分析器脚本窗口”中的上下文菜单启动任何脚本的调试器，或按 Shift+F12 如果你没有分析器脚本仍然可以通过直接附加到该进程来调试正在运行的应用程序： • 从功能区中，选择 执行 > 工具 >调试器> 附加到进程”，或者 • 在 调试”工具栏上，点击  （附加）按钮，手动选择调试平台
暂停/恢复调试	您可以通过以下方式暂停调试会话，或在暂停后恢复会话： • 从功能区中，选择 执行 >运行> 暂停” • 在 调试”工具栏上，单击  按钮
停止调试器	调试器通常在当前调试进程终止时结束；但是，某些应用程序和服务（例如Java虚拟机）可能需要手动停止调试器。要停止调试，请执行以下任一操作： • 在 调试”工具栏上，单击  （停止）按钮 • 按 Ctrl+Alt+F6 • 选择 执行 >运行> 停止”功能区选项上的下拉箭头

	功能区选项显示一个简短菜单，提供三种终止调试应用程序的方法。  • 停止 - 停止调试器并停止正在调试的进程（单击功能区图标时的默认设置） • 分离 - 停止调试器但让进程继续运行 • 退出应用程序 - 停止调试器并将 WM_QUIT 消息发布到进程的主窗口（如果有的话）
节过代码行	跳过下一行代码： • 从功能区中，选择 执行 >运行>节结束”，或 • 在 调试”工具栏上，单击（节过）按钮，或 • 按 Alt+F6
节函数调用	进入函数调用： • 从功能区中，选择 执行>运行>节In”，或 • 在 调试”工具栏上，单击（节入）按钮，或 • 按 Shift+F6 如果目标函数没有可用的源，则调试器立即返回给调用者。
节输出函数	退出函数： • 从功能区中，选择 执行>运行>节输出” • 在 调试”工具栏上，单击（节输出）按钮，或 • 按 Ctrl+F6 如果调试器跳出一个没有源代码的函数，它将继续跳出，直到找到一个源代码的点。
显示执行点	在调试器暂停的同时，返回调试器即将执行的源文件和代码行： • 从功能区中，选择 执行>运行>开始>显示执行点” • 在 调试”工具栏上，点击（显示执行点）按钮。 相应的行将突出显示，屏幕左边缘有一个粉红色箭头。
输出	在调试会话期间，调试窗口中显示的消息详细说明： • 启动的启动 • 会话终止 • 例外 • 错误 • 跟踪消息，例如使用Java系统.out 或.NET系统的输出。诊断。调试 如果双击调试消息，则：

	• 弹出窗口A更多完成消息文本，或 • 如果存在内存泄漏，则文件将显示在错误发生的位置
保存输出（并清除输出）	您可以将调试输出的全部内容保存到外部 .txt 文件，也可以将输出中的选定行保存到Enterprise Architect剪贴板。 要将所有输出保存到文件，请单击  （将输出保存到文件）按钮。 要将所选行保存到剪贴板，请右键单击所选内容并选择 将所选行复制到剪贴板”选项。 当您保存输出或不想再显示它时，右键单击当前输出并选择 清除结果”选项。
切换到探查器	如果您正在代码上运行调试会话，您可以停止调试会话并立即切换到分析会话。 从调试器切换到 Profiler： • 从功能区中，选择 执行 > 工具 >调试器> 切换到 Profiler” • 在调试窗口中，单击  |切换到 Profiler 选项，或 • 在调试工具栏上，单击  |切换到 Profiler 选项 Profiler 附加到当前运行的进程。 此功能不适用于Java调试器。

断点和标记管理

Enterprise Architect中的断点工作与任何其他调试器中的工作方式相同。标记类似于断点，但在Enterprise Architect中它们具有特殊的功能。简而言之，标记执行断点不执行的操作——例如记录执行和分析。断点的作用始终是停止程序。

您可以在源代码编辑器中设置任何标记或断点，它们在左边距中可见。单击此边距将在该行添加一个断点。断点和标记是可互换的 - 您可以使用其 属性”对话框将断点更改为标记，反之亦然。您可以使用 Ctrl 并单击编辑器边缘或断点和标记窗口中的图标，快速查看和编辑断点或标记的属性。

断点保持成套。每个模型都有一个默认设置，每个断点通常都驻留在那里，但您可以将当前断点配置保存为命名集，创建新集并在它们之间切换。断点集是共享的；也就是说，它们可供模型社区使用。例外是默认集，它是分配给任何模型的每个用户的私有和个人集。

访问

功能区	执行>窗口>断点 仿真>动态仿真仿真>断点

断点和标记选项

选项	细节
删除断点或标记	要删除特定断点： • 如果断点开启，点击源代码编辑器左边空白处的红色断点圆圈，或者 • 右键单击源代码编辑器、断点文件夹或断点和标记窗口中的断点或标记，然后选择 删除”选项，或 • 在 调试断点”选项卡中选择断点，然后按删除键
删除所有断点	单击删除所有断点按钮 ()。
断点属性	在断点窗口或代码编辑器中，使用标记的上下文菜单调出属性。您可以在此处更改标记类型、添加或修改约束以及输入跟踪语句。 (有用的快捷键：按住 Ctrl 键的同时单击标记，快速显示其属性。)
禁用断点	取消选中断点或标记对应的复选框。
启用断点或标记	选中断点或标记对应的复选框。
禁用所有断点	单击  按钮
启用所有断点	单击启用所有断点按钮 ()。
修改内存地址时中断	单击数据断点按钮 ()。

识别或更改标记集	选择断点和事件窗口工具栏中的断点 Default 字段。 如有必要，单击下拉箭头并选择不同的标记集。 默认设置通常用于调试，并且是您的用户 ID 个人的；其他标记集在模型内的所有用户之间共享。
更改断点和标记如何在断点和事件窗口上分组	断点和标记可以按类或代码文件分组。要对项目进行分组，请单击工具栏中  图标上的向下箭头，然后单击相应的选项。如果您不想对项目进行分组，请单击已选择的选项以取消选择它；然后按行号列出断点和标记。

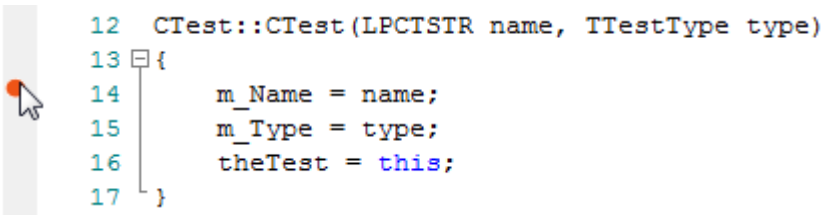
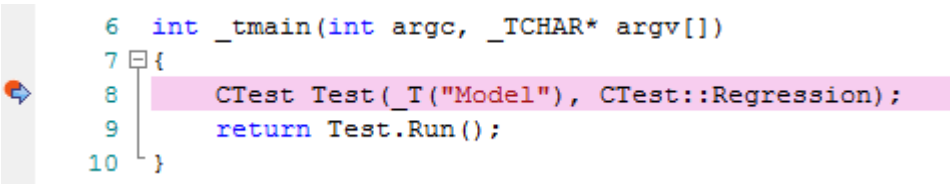
断点状态

状态	评论
	调试*Running:* Bound 调试未运行：已启用
	调试运行：禁用 调试未运行：禁用
	调试运行：未绑定 - 这通常意味着模块尚未加载。此外，dll 有时会被卸载。 调试未运行： N /a
	调试运行：失败 - 这意味着调试器无法将这行代码与任何已加载模块中的指令相匹配。可能源来自另一个项目或项目配置已过期。注记，如果模块日期早于断点的源代码日期，您将在调试器窗口中看到通知。文字是红色的，所以它们会脱颖而出。这清楚地表明该项目需要建设。 调试未运行： N /a

设置代码断点

正常断点通常设置在一行源代码上。当调试器在正常执行过程中到达指定行时，调试器停止执行并显示局部变量、调用堆栈、线程和其他运行时信息。

在一行代码上设置断点

节	行动
1	在集成源代码编辑器中打开源代码进行调试。
2	找到相应的代码行并单击左边距列 - 边距中的实心红色圆圈表示已在该位置设置断点。  如果代码当前在断点处暂停，则该点由标记旁边的蓝色箭头指示。  或者，您可以通过右键单击所需行的左边距来设置断点标记（或其他标记），以显示断点/标记上下文菜单；选择适当的标记类型。

跟踪声明

跟踪语句是在调试会话执行期间输出A消息。可以在Enterprise Architect中定义跟踪语句，而无需对应用程序源代码进行任何更改。

跟踪点标记在代码编辑器中设置。像断点一样，它们被放置在一行代码中。当该行代码执行时，调试器评估该语句，其结果记录到调试窗口（或者如果被分析分析器器脚本覆盖，则记录到文件）。

访问

任何现有的跟踪声明都可以在断点和标记窗口中查看和管理。可以使用此处概述的任何一种方法显示断点和标记窗口。

功能区	执行>窗口>断点

添加跟踪点标记

节	行动
1	在源代码编辑器中打开源代码进行调试。
2	找到适当的代码行，右键单击左边距并选择 添加跟踪点标记”选项。 如果标记已经存在，请按 Ctrl 并单击以显示断点属性窗口。
3	确保选中 跟踪状态”复选框。
4	在 跟踪语句”复选框下的文本字段中，键入所需的跟踪语句。
5	点击确定按钮。 A Marker 显示在代码编辑器的左边距中。

```
55 DWORD CTrain::Disembark(int PeopleOFF)
56 {
57     if(Passengers - PeopleOFF > -1)
58         Passengers -= PeopleOFF;
59     else
60         Passengers = 0;
61
62     if(PeopleOFF > 0)
63         return PeopleOFF * 20;
64
65     return 0;
66 }
```

指定跟踪

跟踪语句可以是A自由格式的文本。当前范围内的任何变量的值也可以通过在变量名称前加上特殊标记来包含

在跟踪语句中。

可用的令牌是：

- $ - 当变量被解释为string时
- @ - 当变量是原始类型时（ int 、double、char ）

使用图像中的示例，我们可以使用以下语句输出下火车的人数：

在@PeopleOFF 在$Arriving 下车之前有@Passengers。名称Station

除了跟踪代码中的变量值之外，您还可以在跟踪语句中使用 $stack 和 $frame 关键字来打印当前堆栈跟踪；利用：

- $stack - 打印所有帧，或
- $frame[start](count) - 从给定帧开始从堆栈中打印特定数量的帧；例如，$frame[0](5) 将打印当前帧和 4 个祖先

注记

- 跟踪语句可以包含在任何类型的断点或标记上。

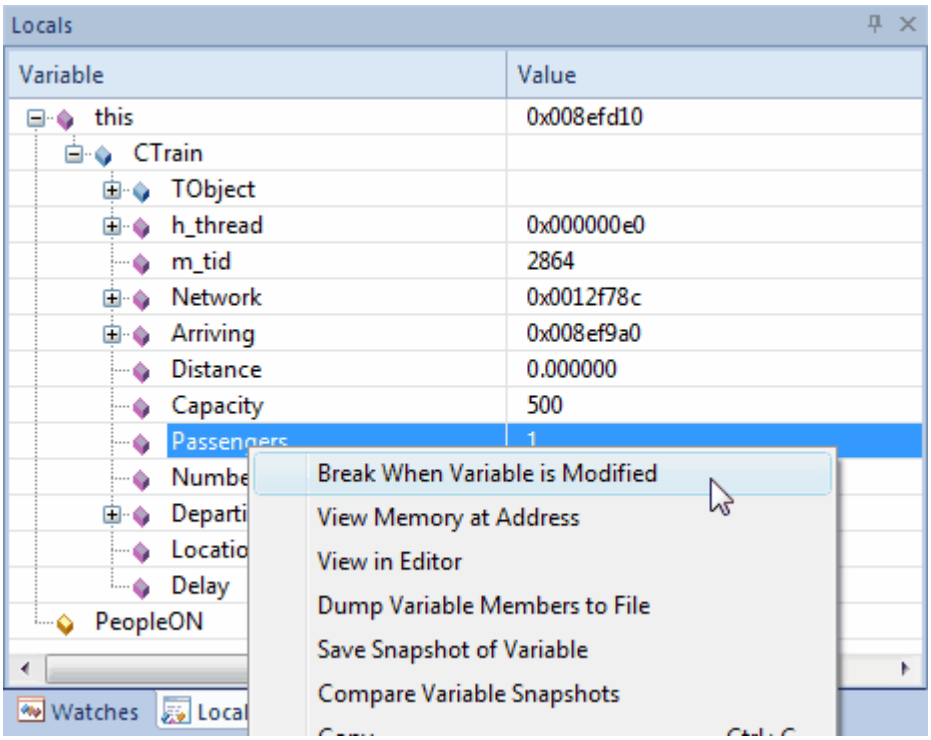
变量修改值时中断

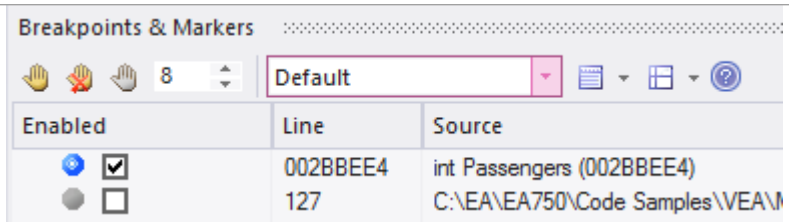
可以在预先确定的内存变量上设置数据断点，以使调试器在刚刚导致变量值更改的代码行处停止执行。这在尝试跟踪程序执行期间变量被修改的点时很有用，尤其是在不清楚程序执行如何影响特定object状态的情况下。

访问

功能区	执行 >窗口> 局部变量：右键单击变量 > 修改变量时中断或 执行 >窗口> 监视：右键单击变量 > 修改变量时中断
其它	在代码编辑器窗口中：右键单击感兴趣的变量 |修改项目时中断

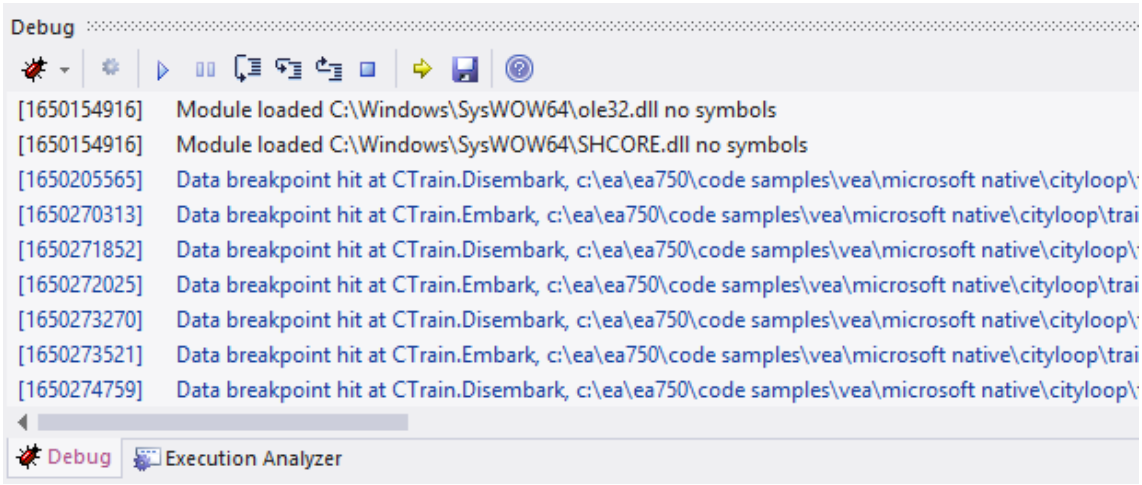
使用数据断点捕获对变量的更改

脚步	细节
1	在代码中设置一个正常的断点，以便您可以选择一个变量。然后运行调试器 (F6)。
2	当程序有断点时，选择感兴趣的变量，然后从其上下文菜单中，选择 "Break When Variable is Modified"选项。 
3	代码中没有断点指示符，但是断点和事件窗口中的数据断点很容易识别，是一个带有白色菱形的蓝色图标。 Enterprise Architect显示变量的名称及其地址而不是行号。



4 设置数据断点后，您可以禁用您可能拥有的任何其他断点。程序将在任何更改此变量值的代码行处停止。现在运行你的程序。

5 修改此变量时，调试器会暂停并在编辑器中显示当前代码行。这不是导致中断的行，而是事件之后的代码行。该事件被记录到调试窗口。

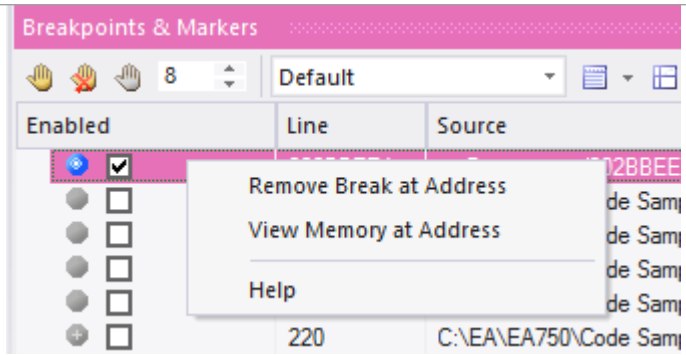


现在我们知道这个值（它的状态）如何以及在何处发生了变化。例如，第 58 行的语句刚刚更新了乘客的数量。

```
55 DWORD CTrain::Disembark(int PeopleOFF)
56 {
57     if(Passengers - PeopleOFF > -1)
58         Passengers -= PeopleOFF;
59     else
60         Passengers = 0;
61
62     if(PeopleOFF > 0)
63         return PeopleOFF * 20;
64
65     return 0;
66 }
```

6 发现此值和其他更改此值的位置后，请务必在继续之前删除通知。您可以通过在断点窗口中选择数据断点并按删除键来快速删除数据断点。

您也可以使用右键单击上下文菜单来执行此操作。



注记

- Microsoft .NET平台目前不支持此特征

跟踪变量修改值时

当您的代码执行时，它可能会更改变量的值。可以在调试窗口中捕获此类更改和变量的新值。然后，您可以双击更改记录以在代码编辑器中显示导致更改的代码行。

访问

功能区	执行>窗口>局部变量：右键单击变量>跟踪变量被修改或 执行 >窗口>跟踪: 右键单击变量 > 当变量被修改时
其它	在代码编辑器中 |右击变量跟踪当变量被修改时

设置跟踪

您要跟踪的变量必须在范围内，因此要识别和选择它，请在您知道该变量将存在的代码行上设置一个普通断点。当调试器到达此断点时，找到变量并使用其上下文菜单启用跟踪。

定位变量：

- 如果您在源代码中看到变量，请将鼠标悬停在其上，右键单击并选择 显示变量”选项； Enterprise Architect将找到它
- 如果变量在范围内（本地，或 this”或 this”的成员），请在本地窗口窗口中查找它（ 执行>窗口>局部变量”）
- 如果变量是全局变量（C、C++），则显示 Watches窗口（'Execute > Window> Watches'）并按名称搜索
- 如果变量是类静态成员，则显示 Watches 窗口（ 执行 >窗口> Watches”）并输入其完全限定名

启用跟踪后，您可以禁用所有其他断点并让程序运行。每次变量更改值时，都会记录到调试器的 输出”选项卡中。选择值的更改并双击该行以在代码编辑器中显示代码。

注记

- 发生更改事件时调试器不会停止，它只记录更改
- 此功能在 Microsoft Native 和Java平台上可用
- Microsoft .NET不支持值断点

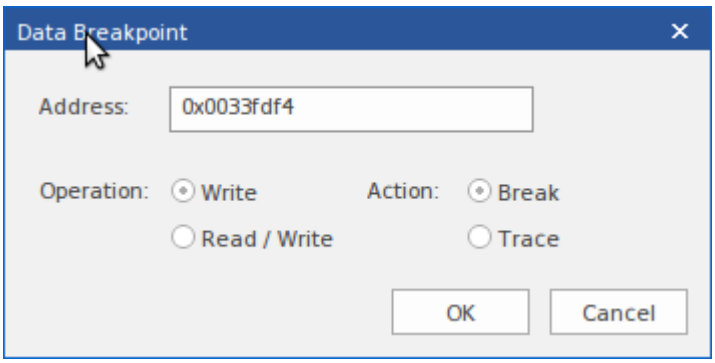
检测内存地址操作

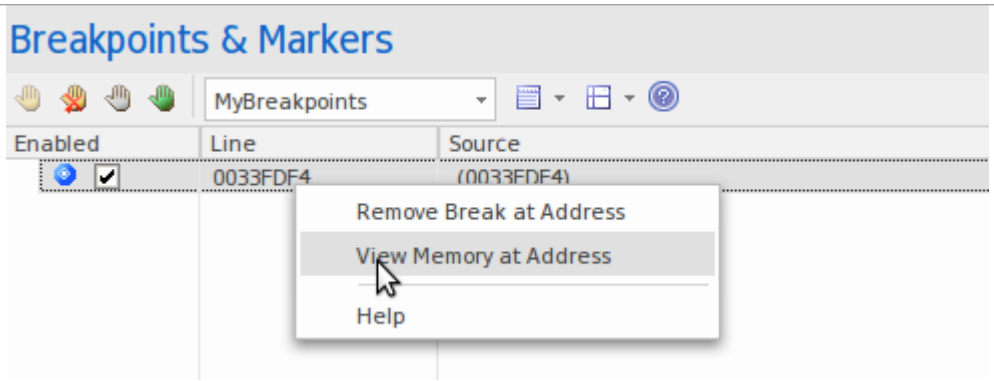
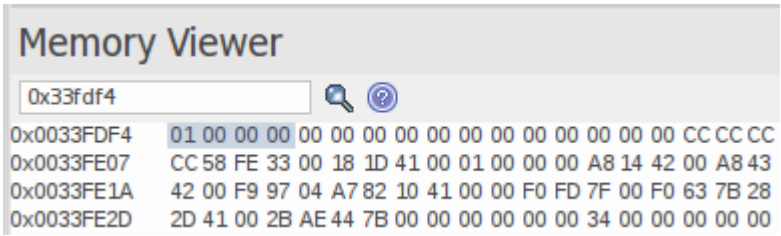
能够检测到内存区域被读取或写入的位置和时间对于调查人员来说是一个很大的帮助，即使代码库已经被很好地理解了。如果没有这个工具，C++ 开发人员可能会面临一项艰巨的任务，即跟踪访问全局变量的位置和时间，并调试这些函数。数据断点允许 C++ 程序员跟踪何时读取或写入变量/内存位置。当检测到该操作时，调试器将停止执行，并且该操作之后的代码行将显示在代码编辑器中。

访问

功能区	执行>窗口>断点

检测对内存地址的操作

节	行动
1	单击按钮。
2	输入要观看的内存地址。您可以从本地窗口（局部变量）窗口复制地址。 
3	选择要检测的操作。如果选择 写入”，则在写入地址时调试器将中断。如果您选择 读/写”，调试器将在读取或写入地址时通知您。
4	选择要执行的操作。如果您选择 Break”，调试器将停止程序并且代码行将显示在编辑器中。如果您选择 跟踪”，调试器将不会停止执行，但会在地址上log任何操作。此输出显示在调试器窗口中。
5	数据断点被添加到断点和标记窗口。

	
6	您可以使用数据上下文上的时间点菜单来检查内存地址处的值。 
7	要删除数据断点，请在断点和标记窗口中选择它，然后按删除键。或者，取消选中它旁边的复选框。数据断点在禁用时被删除；它们不像其他断点那样持续存在。

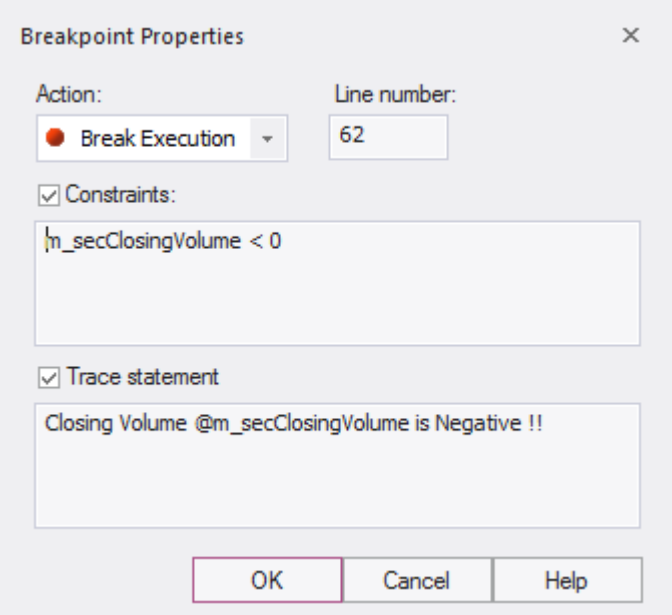
系统需求

C/C++ 本机调试器支持内存地址断点。

断点属性

断点有许多附加属性，这些属性确定在执行断点适用的代码行时会发生什么。
这些属性定义：

- 要执行的动作
- 断点适用的代码行
- 确定断点时是否执行动作的约束
- 断点时输出的跟踪信息



访问

有几种方法可以显示 断点属性”对话框：

代码编辑器	• 右键单击断点标记 |属性或 • Ctrl+单击断点标记或 • 右键单击具有断点标记的代码 |断点 |属性
断点和标记窗口	• 断点右键|属性

选项

字段	细节
行动	命中断点时的行为。
线	此断点适用的源代码行。
	对于堆栈捕获标记，要记录的调用者帧数。要记录整个堆栈，请将值设置为

堆叠高度	0。
约束	定义将执行断点操作的条件。对于正常断点，这将是停止执行的条件。在此示例中，对于正常断点，当条件评估为True时，执行将在此行停止。每次执行代码行时都会评估约束。 (this.m_FirstName="Joe") 和 (this.m_LastName="Smith")
跟踪声明	当断点被命中时，向调试窗口输出A消息。当前在作用域内的变量可以包含在跟踪语句输出中，方法是在变量名称前加上 $ 标记（用于string变量）或 @ 标记（用于原始类型（例如int或 long））。例如： 账户$pAccount->m_sName 余额为@pAccount->m_fBalance

绑定断点失败

A绑定断点出现问题，则会发生断点失败。断点失败最常见的原因是源文件被更改而没有重新构建应用程序。断点有时可以绑定到不同的行，导致它们被移动。如果断点不能在这一行或后面的三行绑定到二进制文件，则会显示一个问号。

断点和事件窗口的 详细信息”列中会显示A警告消息，指出问题的类型：

- 断点的源文件与用于构建应用程序映像的源文件不匹配
- 文件上的时间戳大于图像的时间戳

警告信息也会输出到调试窗口A

调试一个正在运行的应用程序

您可能希望调试已经在系统上运行的应用程序（进程），而不是从Enterprise Architect中显式启动进程。

在这种情况下，您可以使用调试功能附加到已经运行的进程。如果您将适当的调试信息写入正在运行的进程和/或关联的调试文件（例如 .PDB 文件），调试器就会绑定到该进程并启动调试会话。

您也可以在完成检查后从流程中 分离"并让运行正常运行。

访问

功能区	执行>运行>开始>附加到进程
调试器窗口	调试器窗口工具栏有一个附加按钮

阶段

阶段	描述
显示流程	当您选择调试另一个进程时，将显示 附加到进程"对话框。 您可以使用对话框顶部的单选按钮限制显示的进程；要查找 Apache Tomcat 或 ASP.NET 等服务，请选择系统单选按钮。
选择调试器	当您选择一个进程时，您可能必须从调试器下拉列表中选择调试器；但是，如果选择的包已经在分析器脚本中进行了配置，那么脚本中列出的调试器会预设在对话框中。
进程选择	双击包含调试信息的进程后， Enterprise Architect将附加到该进程： • 调试器检测到遇到的任何断点 • 遇到断点时停止进程，并且 • 该信息在调试窗口中可用
脱离进程	要从进程中分离，请单击  （调试停止）按钮。

视图局部变量

本地窗口窗口显示执行系统的变量。无论你是录制C#，调试Java 、C++或VBScript，调试一个可执行状态机，还是运行一个模拟，这个窗口都是系统变量所在的地方。当前值仅在程序停止时显示。当在调试期间遇到断点时，当您跨过一行代码或当您在模拟中的状态之间切换时，就会发生这种情况。

访问

功能区	执行 >窗口> 局部变量 仿真>动态仿真> 局部变量
上下文菜单	在代码编辑器中 |右键单击任何变量标识符 > 显示变量

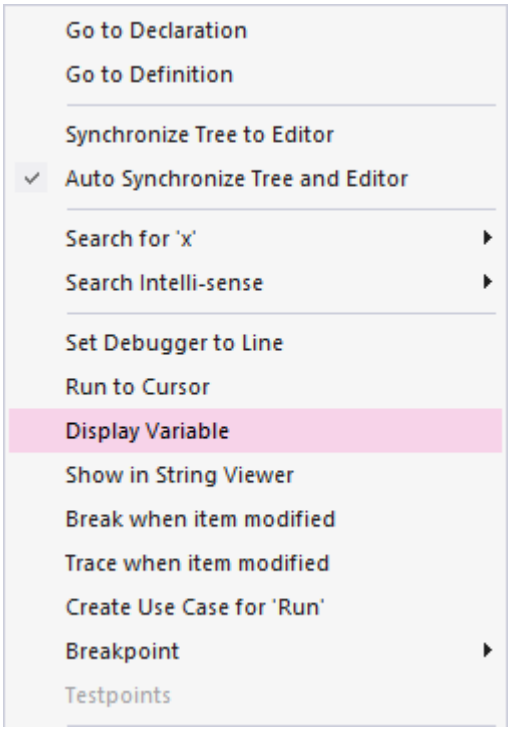
图标

任何范围内变量的值和类型都显示在树中；每个变量都有一个标识变量类型的彩色框图标：

- 蓝色-物件with members
- 绿色 - 数组
- 粉红色 - 元素类型
- 黄色 - 参数
- 红色 - 工作台实例

查找变量

查找变量的最简单方法是首先在代码编辑器中找到它，然后使用变量上的右键单击上下文菜单，选择 显示变量 ″。 Enterprise Architect将发现并揭示范围内的任何变量，包括深度嵌套的成员。如果在不同的范围（全局、文件、模块、静态）中找到变量，它将显示在 Watches 窗口中（请参阅其它范围视图的视图变量）。

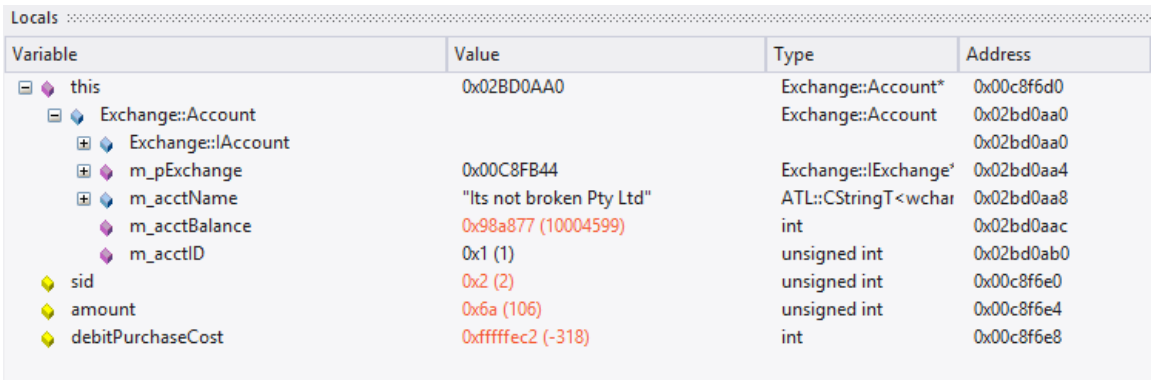


持久视图

变量的检查通常涉及在树中挖掘以显示感兴趣的值。在经历了这些麻烦之后，可能会很烦人，然后上下文下一行代码，只是由于时间的变化，这些变量又被隐藏起来了。本地窗口窗口有一个持久视图，在运行或 step 命令后会停留一段时间。当您在Enterprise Architect中单步执行函数时，变量结构会逐行保留。这使得逐步完成一个函数变得又快又容易。

发生了什么变化

作为持久视图的一部分，本地窗口窗口跟踪值的变化并突出显示它们。



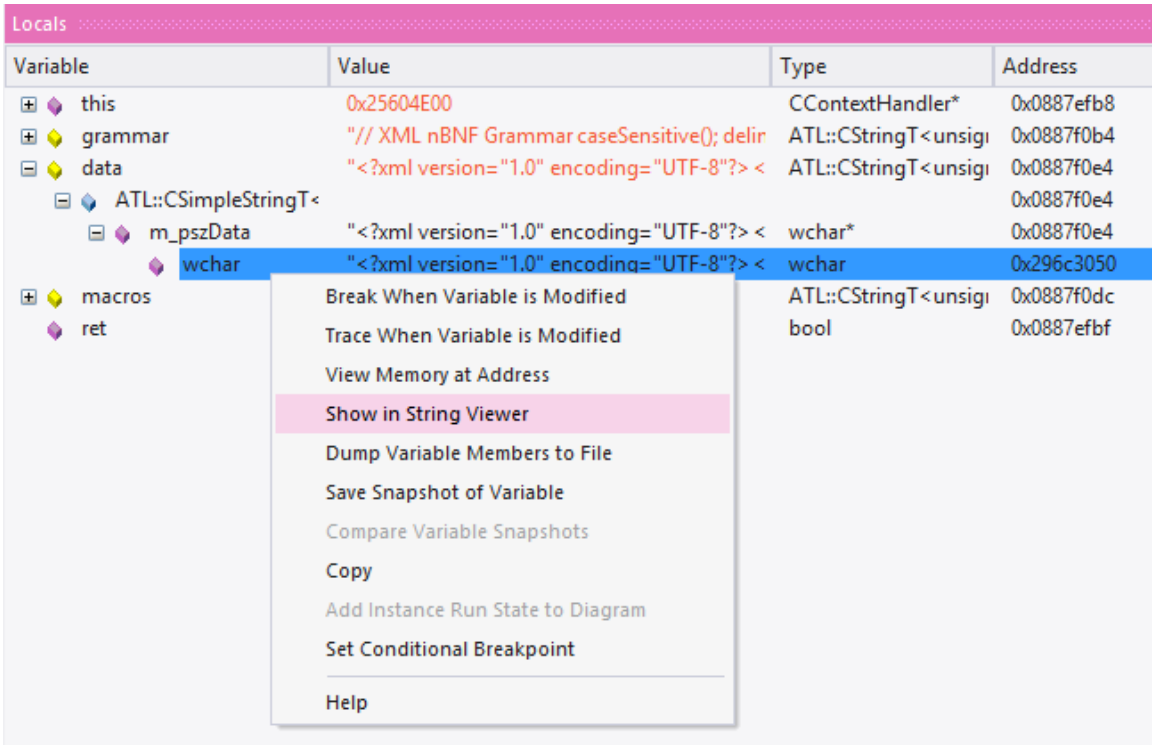
上下文菜单

功能	细节

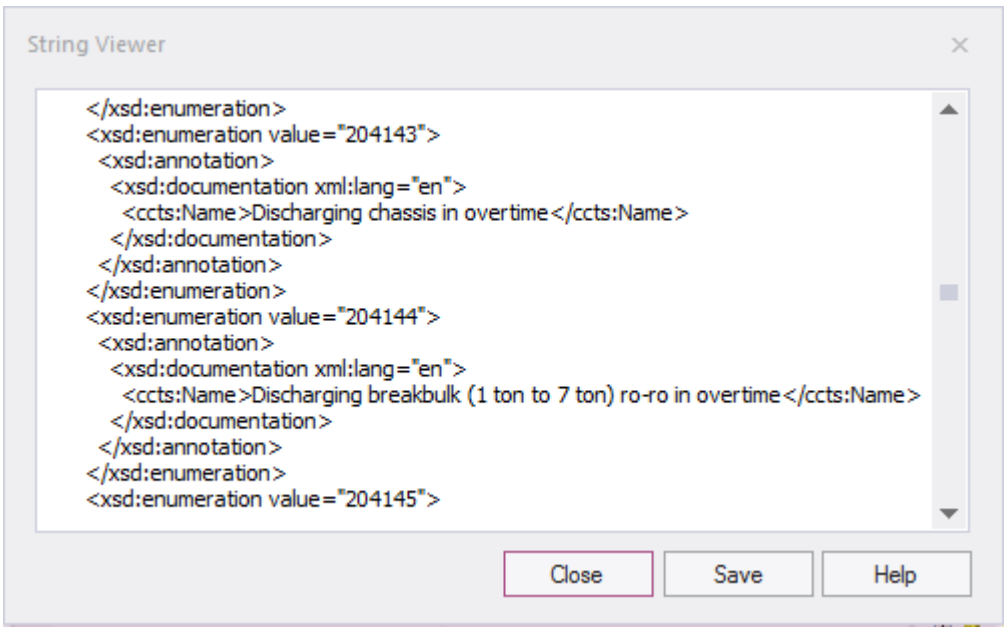
修改变量时中断	在选定的内存变量上设置数据断点，以在刚刚导致变量值更改的代码行处停止调试器的执行。
视图地址的内存	以十六进制和 ASCII 格式显示内存中所选地址的原始值。
在字符串查看器中显示	在 "字符串查看器"对话框中显示变量string 。
将变量成员转储到文件	捕获所选变量并将其存储到单独的位置；将显示一个浏览器以选择适当的 .txt 文件名和文件路径。
保存变量快照	在变量生命周期的特定时间点捕获变量的值。
比较变量快照	比较该变量生命周期中不同时间点的变量值。
复制	将所选变量复制到Enterprise Architect剪贴板。
将实例运行状态添加到图表	如果您打开了包含正在调试源代码的类的物件的模型图，此选项将使用变量值表示的运行状态更新该物件。
设置条件断点	在当前执行位置添加一个断点，并为此变量匹配其当前值。

视图长字符串的内容

为了提高效率，本地窗口窗口只显示部分字符串。但是，您可以使用 "字符串查看器"显示string变量的全部内容。



此示例显示了保存 XML 模式文件内容的变量的值。

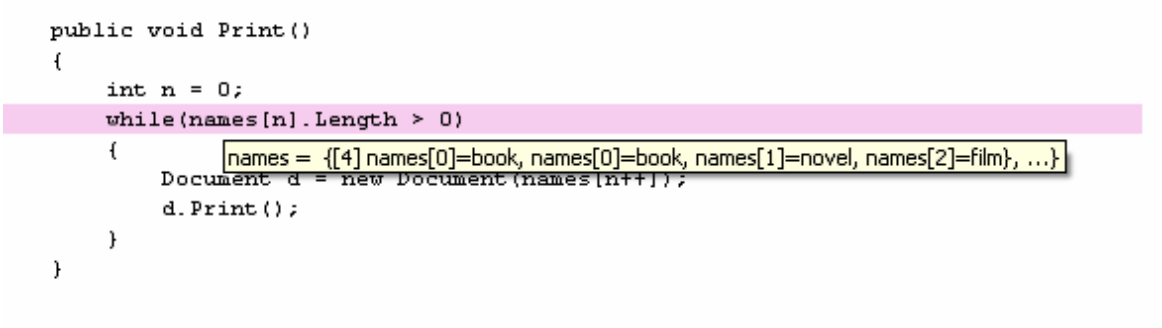


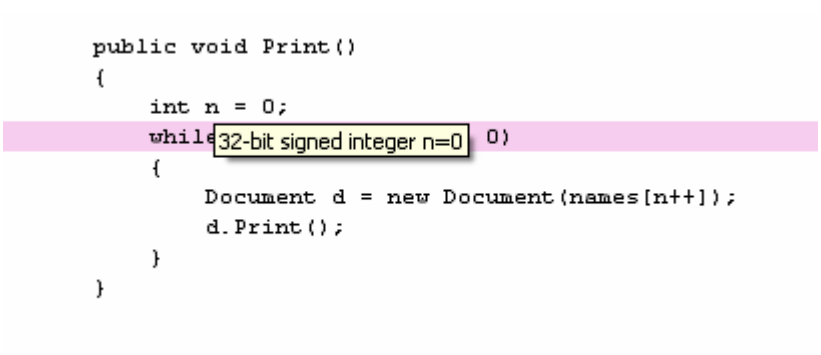
访问

从代码编辑器或本地窗口窗口	右键单击string变量 |在字符串查看器中显示

视图代码调试代码编辑器中的变量

发生断点时，您将在该窗口中看到所有局部变量。您还可以通过将鼠标悬停在引用上来检查源代码编辑器中的变量。这里有些例子。



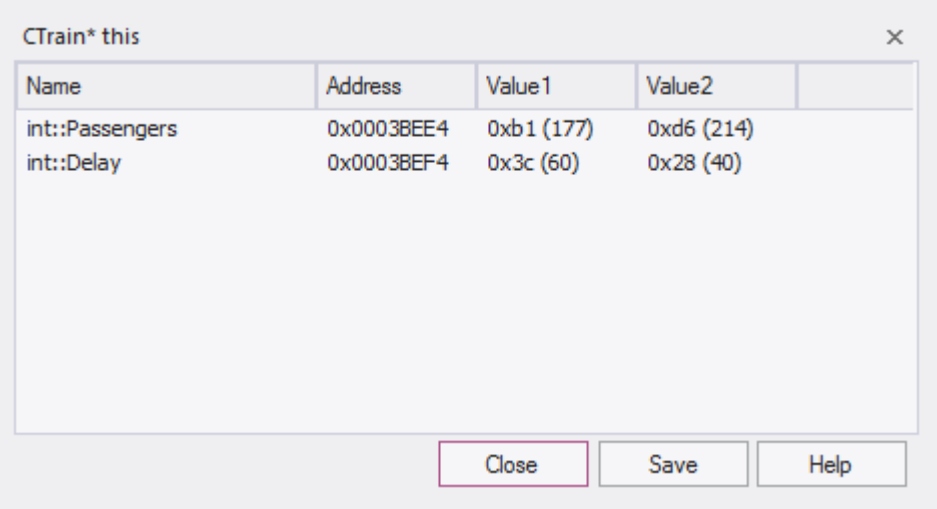


注记：变量不必是局部变量之一。它可以具有文件或模块范围。

可变快照

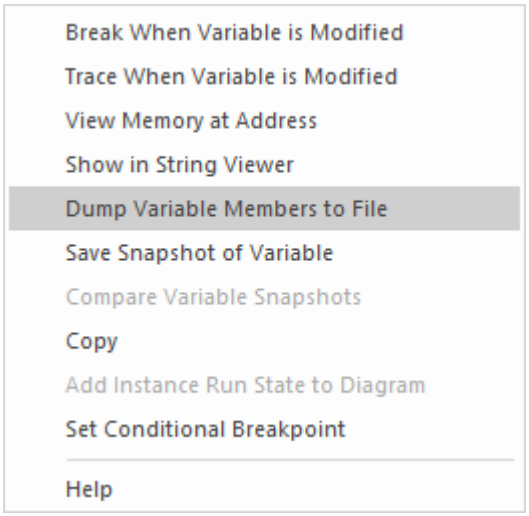
当您的程序遇到断点时，可以对变量进行 快照”，并使用此快照来查看变量的值在其生命周期的不同点是如何变化的。调试器不会只复制所选变量的值；对于复杂变量，它会复制所选变量的值及其每个成员层次结构的值，直到它不再为某个成员找到更多调试信息或找不到更多成员。

捕获变量快照

节	行动
1	在代码编辑器中，设置两个断点：一个在函数的开头，另一个在函数的结尾。
2	在开始断点处，右键单击本地窗口窗口中的变量并选择 保存变量快照”菜单选项。
3	运行应用程序。
4	当到达结束断点时，右键单击本地窗口窗口中的变量并选择 比较变量快照”选项。 将显示A对话框，其中显示了第一个快照的原始值和第二个快照的当前值，如从 EA 获取的此图中所示。示例模型。 

将变量快照保存到文件

您可以使用其右键单击上下文菜单将变量的状态保存到文件中。



这是文件内容的摘录。

```
00000006|0x00731F00|name|TObjectType::Type  |value|TypeIsStation|
00000005|0x00731F08|name|wchar::Name    |value|"Treasury"|
00000005|0x00731F0C|name|unsigned::Location |value|0x40 (64)|
00000003|0x0003BED8|name|float::Distance    |value|0|
00000003|0x0003BEE0|name|int::Capacity  |value|0x1f4 (500)|
00000003|0x0003BEE4|name|int::Passengers    |value|0xd6 (214)|
00000003|0x0003BEE8|name|unsigned::Number   |value|0x3 (3)|
00000003|0x0003BEF0|name|unsigned::Location |value|0x0 (0)|
00000003|0x0003BEF4|name|int::Delay |value|0x28 (40)|
```

行动点

行动点是可以执行动作的断点。遇到断点时，调试器会调用运行脚本，然后进程继续运行。行动点是复杂的调试工具，可为专业开发人员提供额外的命令套件。有了它们，开发人员可以改变函数的行为，捕捉行为改变的点，并修改/检测object的状态。为了支持这些特征，行动点可以改变原始局部变量和成员变量的值，可以定义自己的 "用户定义变量"并改变程序执行。

行动点和断点中的用户定义变量

用户定义的变量 (UDV)：

- 提供在 Actionpoint 语句中设置 UDV 原语或string的方法
- 可用于多个标记/断点的条件语句
- 可以在同一个局部变量窗口中轻松查看
- 调试结束时会记录所有 UDV 的最终值。

在 UDV 语法中，UDV 名称：

- 必须以 # (哈希) 字符开头
- 不区分大小写

行动点声明

Actionpoint 语句可以包含 set 命令、goto 命令和 jmp 命令。

设置命令

设置变量值。一个 Actionpoint 语句可以包含多个 "set"命令，所有这些命令都应该在任何 "goto"命令之前。

'set' 命令语法是：

设置 *LHS* = *RHS*

在哪里：

- **LHS** = 变量的名称为：
  - 用户定义变量 (UDV)，例如 #myval
  - 局部变量或成员变量，例如 strName 或 this.m_strName
- **RHS** = 要分配的值：
  - 作为文字或局部变量
  - 如果是文字，则为以下之一：整数、布尔值、浮点数、字符或string

set 命令 - 变量示例

UDV 示例	局部变量示例
设置 #mychar = 'a'	设置 this.m_nCount=0
设置#mystr = "一个string "	设置 bSuccess=false

设置#myint = 10	
设置#myfloat = 0.5	
设置#mytrue = true	

转到命令

此命令将执行切换到函数中的不同行号。 Actionpoint 语句只能包含一个 goto 命令，作为语句中的最后一个命令。

goto 命令语法为：

转到*L*

其中**L**是当前函数中的行号。

goto命令使用断点来实现其目标，这会导致代码执行稍有延迟。这在非常频繁执行的代码区域中可能很明显，因此您可能更喜欢在此类代码中使用 jmp 命令，以实现相同的执行转移但延迟更少。

jmp 命令

jmp命令实际上与goto命令相同。

跳转125

转到125

这两个命令都会导致执行更改为第 125 行。

然而， jmp语句在内部使用检测来指示程序移动执行，而goto语句使用断点来执行此操作，这会导致处理延迟。因此，区别在于jmp语句的卓越性能，尤其是在代码区域执行非常频繁的情况下。

整数符

如果存在用户定义变量 (UDV) 并且它的类型为int ，则可以使用 ++ 和 -- 运算符对其进行递增和递减。例如：

1. 创建一个 UDV 并将其值和类型设置为本地整数变量。
   AP1：设置#myint = nTotalSoFar
2. 增加 UDV。
   AP2：#myint++
3. 减少 UDV。
   AP3：#myint--

定时器操作

行动点可以报告两点之间经过的时间。只有一个定时器可用，可以通过 startTimer 命令重置或启动。然后可以使用 printTimer 命令打印当前经过的时间。最后，打印总经过时间并使用 endTimer 命令结束计时器。

示例动作点条件

使用文字和常量：

- (#mychar='a')
- (#mystr <> "")
- (#myint > 10)
- (#myfloat > 0.0)

使用局部变量：

- （#myval == this.m_strValue）
- (#myint <> this->m_nCount)
- (#myint != this->m_nCount)

指令记录

指令记录可用于检测已知行为的变化；执行点 (B) 与先前的执行 (s) (A) 不同。命令是：

- recStart - 开始录制或开始比较之前的录制是否存在
- recStop - 结束录制
- recPause - 暂停录音
- recResume - 恢复录制

recStart命令开始记录指令。然后存储执行的指令。当遇到**recStop**命令时，记录会被保存。两个行动点之间的任一时间只能保存一个录音。当遇到**recStart**并且存在先前的记录时，调试器将开始将每个后续指令与其记录进行比较。它可以执行许多比较。如果并且当检测到差异时，调试器将中断并且行为改变的代码行将显示在代码编辑器中。比较的迭代也被打印出来。

录音默认存储在内存中，但也可以使用命令语法将其存储到文件中：

recStart 文件规范

例如：

recStart c:\mylogs\onclickbutton.dat

当遇到指定文件的**recStart**命令并且该文件存在时，它将被加载到内存中并且调试器将立即进入比较模式。

表达式

Breakpoint、Actionpoint 和测试点条件表达式中没有隐含的优先级。在复杂表达式中，必须使用括号。请参阅以下示例：

类型	示例
Actionpoint UDV 示例	(#myint=1) 与 (#mystr="德国")
局部变量示例	(this.m_nCount > 10) 或 (nCount%1) (this.m_nCount > 10) 或 (bForce)
条件表达式中的等式运算符	<> - 不等于 != - 不等于 == - 相等 = - 相等
Actionpoint 中的赋值运算	= - 将 RHS 分配给 LHS

符	
条件表达式中的算法运算符	/ - 分配 + - 加号 - - 减 * - 乘法 % - 模量
条件表达式中的逻辑运算符	AND - 两者都必须为真 或 - 一个必须为真 && - 两者都必须为真 || - 一个必须是真的 ^ - 异或（只有一个必须为真）

视图的其它变量

访问

功能区	执行 >窗口> 手表
其它	执行分析器window工具栏：  |手表

视图

视图	描述
手表	Watches 窗口对于本机代码（C、C++、VB）最有用，它可用于评估不能作为局部变量使用的数据项 - 具有模块或文件范围的数据项和静态类成员项。 您还可以使用该窗口评估Java和.NET中的静态类成员项 要添加监视，请在工具栏中键入要监视的变量的名称，然后按 Enter 键。 要检查 C++、 Java或 Microsoft .NET中的静态类成员变量，请输入其完全限定名称： CMyClass::MyStaticVar 要检查具有模块或文件范围的 C++ 数据符号，只需输入其名称。 通过查看当前范围来评估变量；即当前栈帧所在的模块（可以在断点处双击帧中的帧调用堆栈范围）。 如果全局变量存在于不同的模块中，您可以通过在变量前面加上模块名称来检查变量 模块名！变量名 错误输入数据项名称很容易，因此如果您出错或发现错误，请突出显示string，按 F2 并重新输入文本。这还通过在发现匹配项时中断搜索来加快调试器中命名项的解析。
历史	输入项目的历史记录被保留。可以使用工具栏文本框中的向上箭头键和向下箭头键再次选择以前输入的名称或表达式。历史也将在同一台机器上的Enterprise Architect或模型的任何实例中为用户保留。

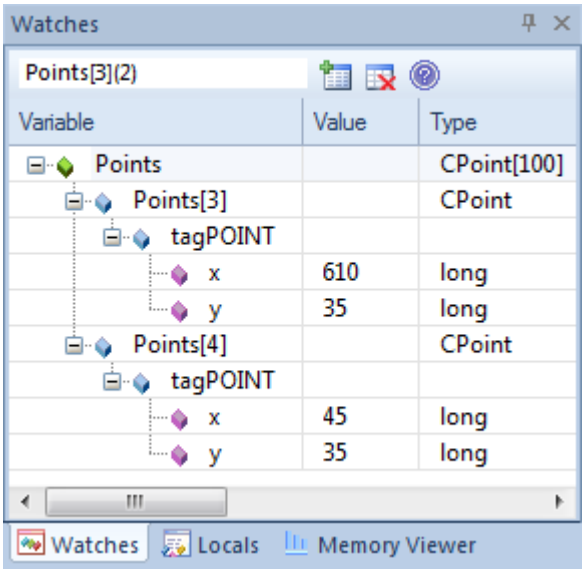
视图元素of Array

您可以使用 Watches 窗口检查数组的一个或多个特定元素。

在 Watches 窗口工具栏左侧的字段中，键入数组的变量名称，后跟开始元素和要显示的元素数。起始元素用方括号括起来，元素个数用括号括起来；那是：

变量[start_element](count_of_elements)

例如，Points[3](2) 显示 Points 数组的第四个和第五个元素，如图所示。



如果您输入 Points[3]，Watches 窗口将仅显示第三个数组元素。

访问

功能区	执行 >窗口> 手表
其它	执行分析器window工具栏：  |手表

查看调用堆栈

调用堆栈窗口用于显示一个进程中当前正在运行的所有线程。它可用于在程序故障发生之前立即识别哪个线程正在运行。

当仿真处于活动状态时，调用堆栈会显示当前运行上下文的执行时间。这将包括每个并发模拟 线程”的单独上下文堆栈。

每当线程被挂起、通过其中A步骤操作或遇到断点时，都会显示堆栈跟踪。调用堆栈窗口可以记录堆栈变化的历史，并且可以根据这个历史生成序列图。

访问

功能区	执行 >窗口>调用堆栈
其它	执行分析器调用堆栈工具栏：  |执行分析器

使用到

- 视图堆栈历史以了解进程的执行
- 视图线程
- 保存调用堆栈以备后用
- 记录序列图生成的调用堆栈更改
- 从调用堆栈生成序列图
- 视图源代码编辑器中的相关代码行

功能

功能	描述
指标	• A红色箭头突出显示当前堆栈帧 • 蓝色箭头表示正在运行A线程 • 红色箭头表示正在记录堆栈跟踪历史记录A线程
将调用堆栈保存为 .TXT文件	目前不可用。
在调试会话中记录线程	要记录线程的执行并将记录指向Record &调用堆栈窗口，右键单击资源中的线程并选择适当的上下文选项： • 'Record' - 在调试会话期间手动记录当前线程 与调试器的 步骤”按钮一起使用；由于 step 命令而调用的每个函数都会记录到 Record & Analyze 窗口 • 'Auto-Record' - 在调试会话期间执行自动记录 当您选择此图标时，分析器开始记录并且在程序结束、停止调试器或单

	击 停止”图标之前不会停止
停止记录	如果您已经开始手动或自动记录线程，您可以在完成之前将其停止；选择线程（由红色箭头指示），然后： • 单击工具栏中的  （停止记录）按钮或 • 右键单击并选择 停止”选项
生成调用堆栈序列图表	要从调用堆栈轨迹生成序列图，可以： • 单击  （生成堆栈序列图表）按钮，或 • 右键单击并选择 生成序列图表”选项
将堆栈复制到记录历史	要立即将堆栈详细信息添加到 记录和分析”窗口（用于以后生成序列图），请执行以下任一操作： • 单击  按钮，或 • 右键单击并选择 将堆栈复制到记录历史记录”选项
切换堆栈深度	要在显示完整堆栈和仅显示带有源的帧之间切换，请单击  （切换堆栈深度）按钮。
在源代码编辑器中显示相关代码	双击一个线程/帧，在源代码编辑器中显示相关的代码行；局部变量也会为选定的帧刷新。

创造图表的序列调用堆栈

调用堆栈窗口记录了堆栈变化的历史，您可以从中生成序列图。

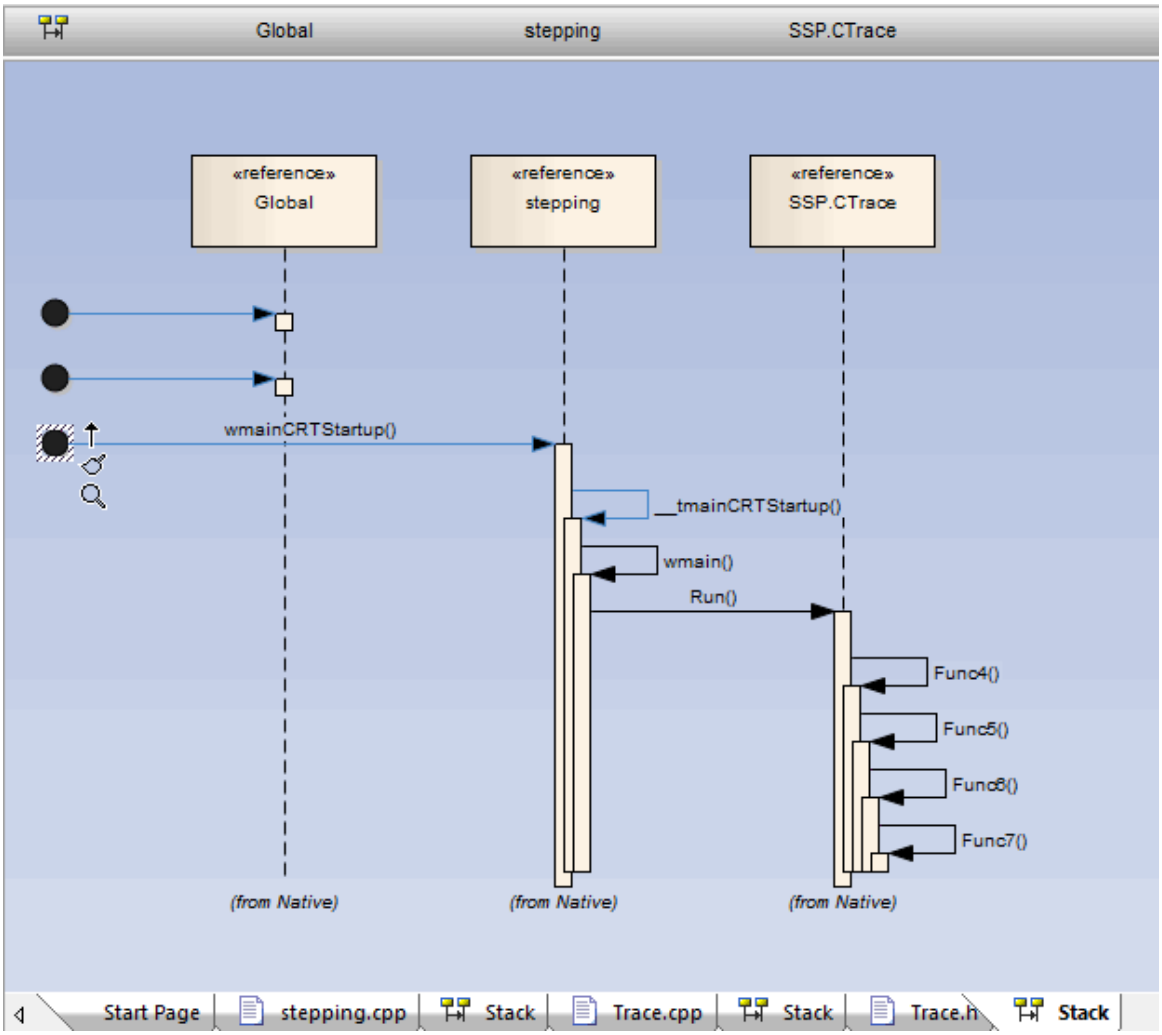
访问

功能区	执行 >窗口>调用堆栈
其它	执行分析器调用堆栈工具栏：  |执行分析器

使用到

- 记录调用堆栈图生成的序列
- 生成来自二调用堆栈的序列图

要生成序列图堆栈，请单击当前工具栏上的调用堆栈堆栈的  (生成序列图表) 按钮。
这会立即在图形图表视图中生成一个序列图。


Global
stepping
SSP.CTrace
«reference»
Global
«reference»
stepping
«reference»
SSP.CTrace
wmainCRTStartup()
__tmainCRTStartup()
wmain()
Run()
Func4()
Func5()
Func6()
Func7()
(from Native)
(from Native)
(from Native)
Start Page
stepping.cpp
Stack
Trace.cpp
Stack
Trace.h
Stack

检查进程内存

使用内存查看器，您可以以十六进制和 ASCII 格式显示内存的原始值。您可以在 地址”字段（右上角）中手动定义内存地址，或者右键单击本地窗口窗口或监视窗口中的变量并选择 视图地址处的内存”选项。

访问

功能区	执行 >窗口> 内存查看器
其它	执行分析器window工具栏：  |内存查看器 从本地窗口窗口或监视窗口：右键单击变量|视图地址处的内存

注记

- 内存查看器可用于调试在WINE窗口运行的 Microsoft 本机代码应用程序（C、C++、VB）

显示加载的模块

对于.NET和本机窗口应用程序，您可以使用 模块”窗口列出被调试进程加载的 DLL。此列表还可以包括调试器使用的关联符号文件（PDB 文件）。

访问

功能区	执行 >窗口> 模块

模块窗口显示

柱子	描述
小路	显示加载模块的文件路径。
加载地址	显示加载模块的基内存地址。
修改日期	显示本地文件日期和修改模块的时间。
调试符号	显示： ● 调试符号类型 ● 模块中是否存在调试信息，以及 ● 模块是否存在线路信息（调试时需要）
符号文件匹配	表示符号文件的有效性；如果值为false ，则符号文件已过期。
符号路径	显示符号文件的文件路径，它必须存在才能进行调试。
修改日期	显示创建符号文件的本地文件日期和时间。

处理首次异常

访问

功能区	执行 > 工具 >调试器>处理首次异常
其它	调试窗口工具栏：  |处理首次异常

加工元素

元素	描述
调试进程	当应用程序正在被调试并且调试器被通知异常时，应用程序被暂停并且调试器以它配置的方式响应；它要么： • 恢复应用程序并将异常留给应用程序管理，或 • 保持应用程序暂停并将异常传递给适当的例程以进行自动解决或手动干预
第二次机会例外	Enterprise Architect调试器默认为第一个列出的行为。 如果应用程序可以处理异常，则继续处理；如果它不能处理异常，调试器会再次收到通知，这一次它必须暂停应用程序并解决异常情况。 在这种行为中，因为调试器已经遇到过两次异常，所以称为第二次机会异常；在这种情况下，如果异常没有停止执行，它会被忽略，您可以避免将时间花在不影响处理整体结果的条件上。 您可能会在大型或复杂系统上以这种方式工作，这些系统总是在处理路径中的某处涉及异常条件。
第一次机会例外	但是，如果您想检查发生的每个异常，您可以将调试器设置为采用第二种行为。 因为调试器在第一次接触时响应异常，所以它被称为第一次机会异常。 您可能会以这种方式处理必须干净或根本不工作的单个函数或例程。
选择	选择 处理首次异常”选项以在第一次联系时调试异常。 取消选择仅当应用程序发生异常时才处理异常的选项。

即时调试器

您可以将Enterprise Architect调试器注册为操作系统即时调试器，以便在系统上的Enterprise Architect外部运行的应用程序遇到异常或崩溃时调用。当您这样做时，应用程序崩溃将导致Enterprise Architect被打开，并显示崩溃的源和原因。

访问

功能区	执行>工具>调试器>设置为JIT调试器

编译申请

本主题说明如何在Enterprise Architect中对您的应用程序执行编译脚本。

访问

功能区	执行>源>编译>编译
键盘快捷键	Ctrl+Shift+F12
其它	'编译'工具栏 >  执行分析器窗口 | 

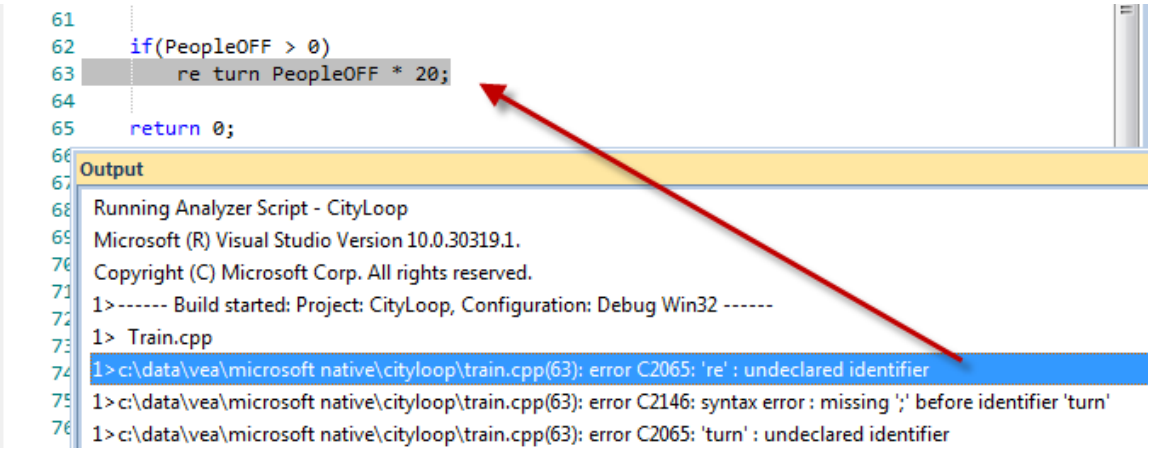
行动

当您选择 编译”选项时，它会在执行分析器窗口中选择的脚本中执行 编译”命令。构建操作的进度和结果显示在系统输出窗口的 编译”选项卡中。

您可以通过双击错误快速访问出现的任何编译错误的代码行。

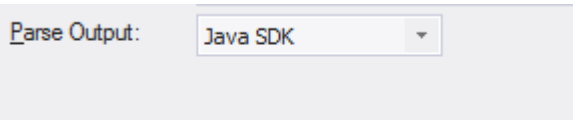
在代码中定位编译器错误

当您使用分析器脚本应用程序时，编译器输出会记录在系统输出窗口中。您可以双击此处出现的任何错误消息并转到源代码。当您这样做时，光标将位于包含错误的行上。



小费

如果缺少输出，请检查分析器分析器中是否提到了语言脚本（Shift+F12）。

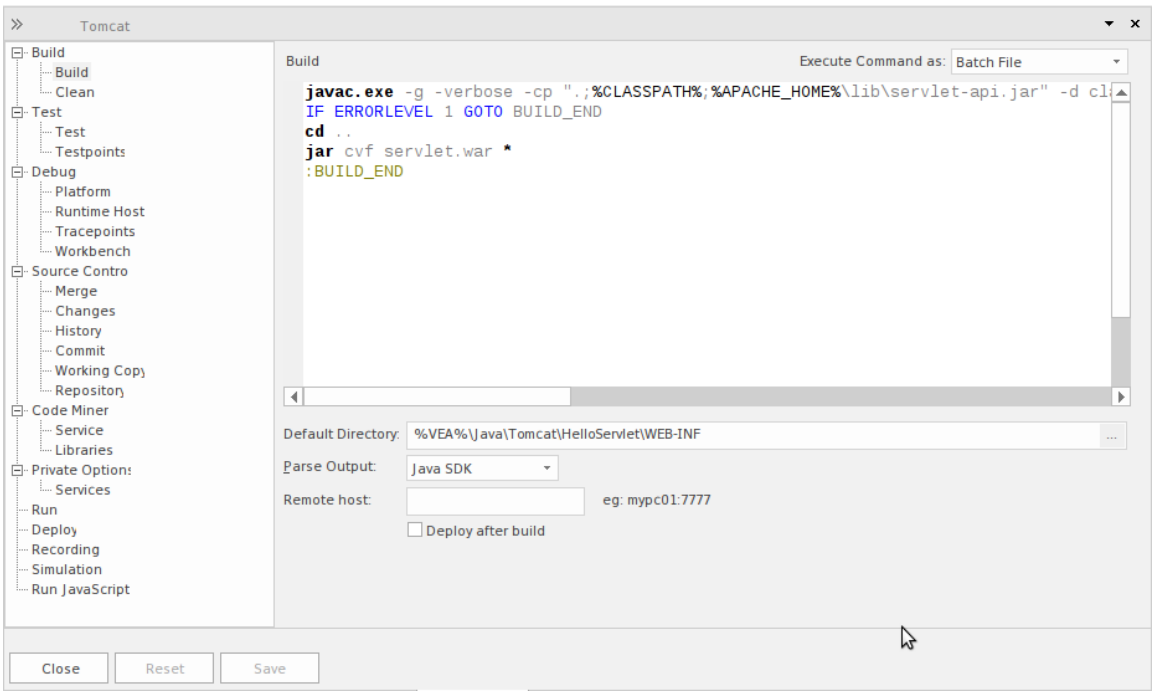


访问

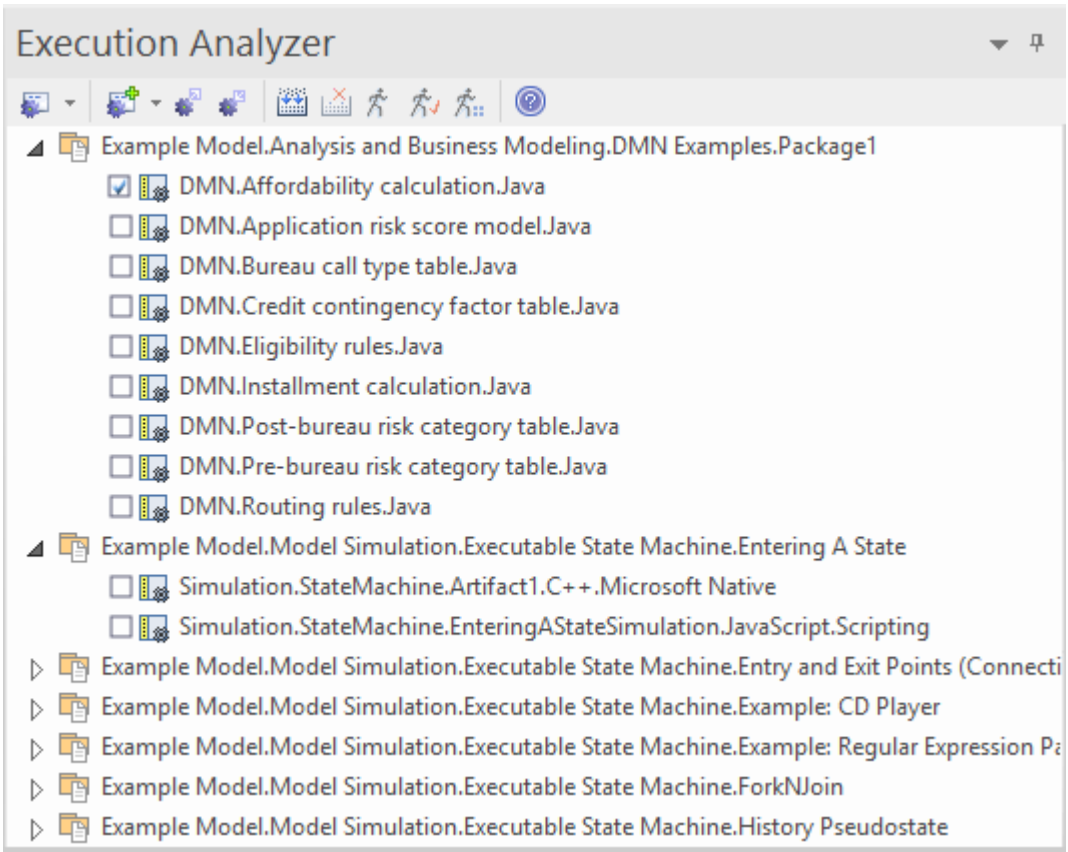
功能区	>开始> 所有窗口>设计> 探索 >系统
键盘快捷键	Ctrl+Shift+8

分析器脚本

分析器脚本由执行分析器使用。您无需担心创建这些。它们与JavaScript或 PHP 不是同一类型的脚本，而是使用熟悉的用户界面（树形视图）进行管理，并且您可以快速定位要更改的特征。分析器脚本可以由社区模型的用户共享并且是轻松导入和导出为 XML 文件。



A项目可以有多个配置，并且可以在分析器窗口中找到这些配置。



每个分析器脚本为一个包定义的，所以项目可以很愉快地共存。在许多组织中，管理系统的过程是分布式的，并且因人而异，因人而异。分析器Enterprise Architect模型中的脚本可以通过信任单个、共享和负责的过程为这些组织提供一些安心用于构建和部署任何种类的配置。脚本的所有方面都是可选的。例如，您可以在没有人的情况下进行调试；但是，只需几行代码，它们就可以启用这些有用的特征：

- 建造
- 测试
- 调试
- 记录
- 执行
- 部署
- 仿真

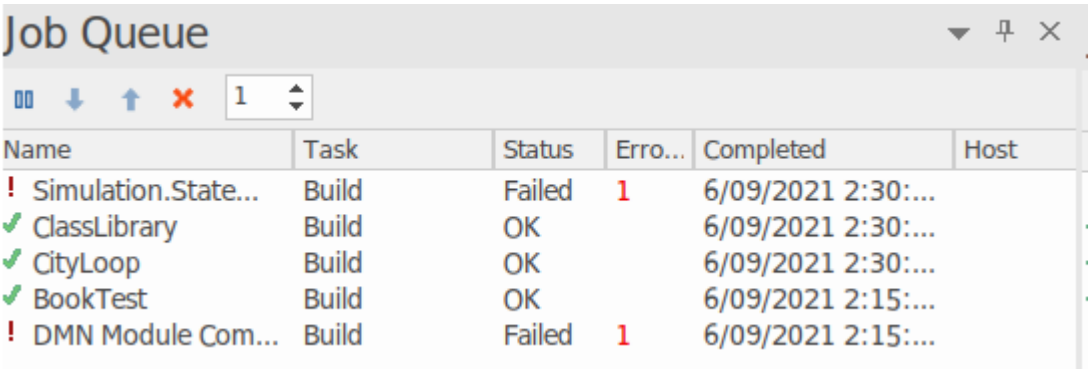
远程脚本执行

编译、运行等各种分析器脚本，提供了一个'Remote Host'字段。此字段用于描述脚本应在其上运行的运行。为了使用这个特征，Sparx卫星服务必须在机器上运行。该字段的格式是*hostname:port*，其中*hostname*是Linux机器的IP地址或网络名称， *port*是卫星服务正在监听的端口窗口号。此特征的主要目标是允许在 Linux 上运行的Enterprise Architect用户执行 Linux 原生的命令。

作业队列窗口

Job Queue 窗口简化了使用分析器脚本的过程，这些脚本最初是单独处理的，如果没有其他脚本正在执行。在Enterprise Architect脚本以后的版本中，当一个分析器执行上下文菜单选项被执行时（例如，'编译'），它被放置在一个作业队列中；可以将多个作业排入队列，并在处理作业时执行其他工作。

以分析器的脚本作为作业名称。一个分析器脚本有多个部分，例如编译、测试、运行和部署，每个部分都被分配为作业的一个任务。



从作业队列窗口执行的每个作业的输出被捕获到系统输出窗口中的 作业历史”选项卡。

访问

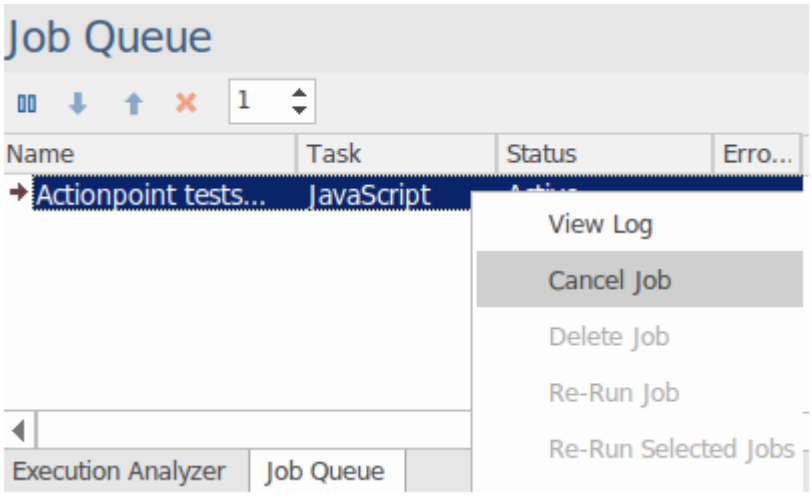
功能区	执行 > 工具 >分析器>视图作业队列 >开始> 所有窗口>设计> 探索系统工作经历

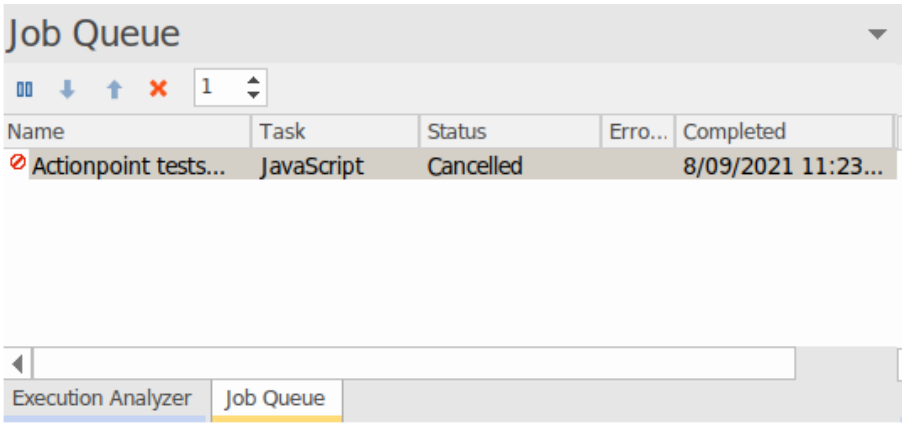
作业队列列

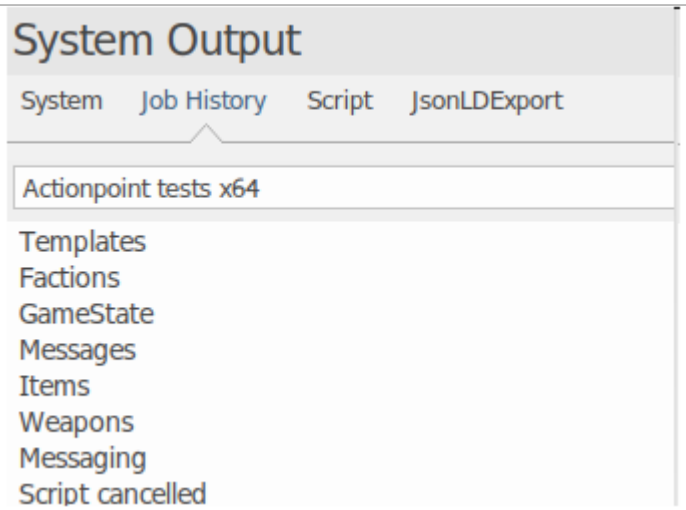
柱子	描述
名称	已将作业添加到作业脚本中的分析器名称。如果作业已执行，则名称前面将有一个勾号（表示成功完成）或感叹号（表示失败）。
任务	作业正在执行的分析器脚本- 例如，编译或部署。
状态	作业的完成状态- 作业是成功完成（'确定'）还是失败。
错误	如果作业已执行并失败，则出现的错误数。
完全的	作业完成的日期和时间。
主持人	如果作业正在远程运行，运行远程机器的 IP 地址或主机名。
接收	如果作业正在运行，则来自主机的任何返回消息。

上下文菜单选项

右键单击作业名称或窗口背景，以显示 作业队列”上下文菜单。

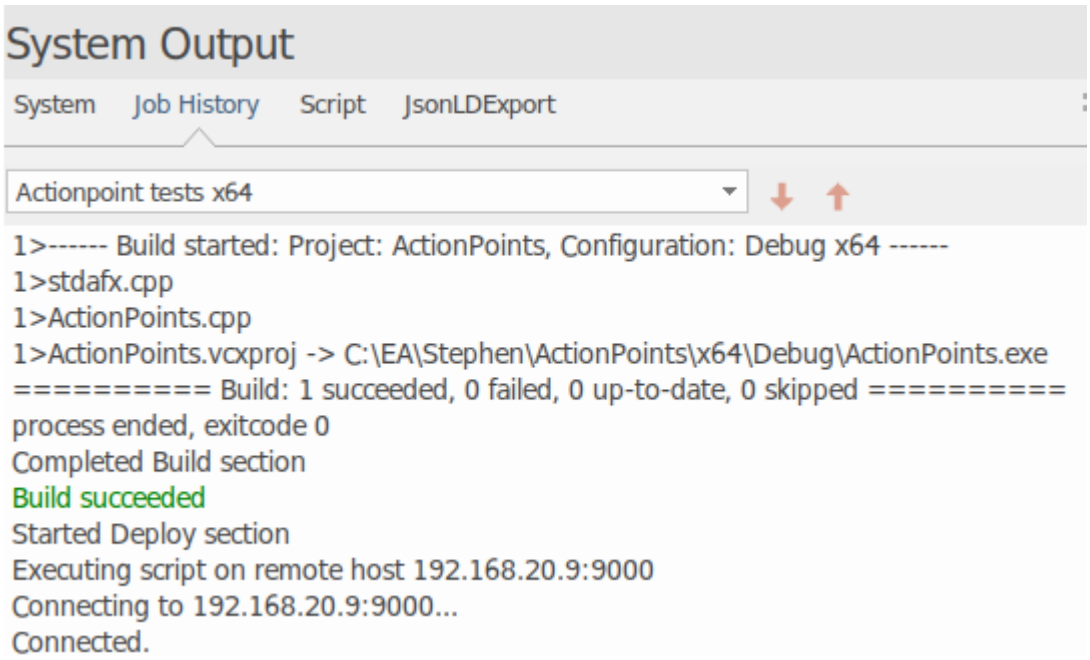


选项	描述
视图日志	选择此选项以显示系统输出窗口的 作业历史”选项卡，以显示已执行的作业。
取消作业	右键单击作业名称，然后根据需要随时单击此选项以取消选定的JavaScript任务。 取消在作业队列窗口中由作业名称旁边的 无条目”图标指示。  在系统输出窗口的 作业历史”选项卡中，输出以 脚本已取消”消息终止。

	
删除工作	单击尚未开始的作业并选择此选项以将其从作业队列中删除。
重新运行作业	选择此选项可再次执行选定的已完成作业。
重新运行选定的作业	（按住 Ctrl 并单击多个所需的作业。）选择此选项可再次执行所有选定的已完成作业。
重新运行已完成的作业	选择此选项可再次执行当前列表中的所有已完成作业。

作业历史选项卡

从作业队列窗口执行的每个作业的输出被捕获到系统输出窗口中的 作业历史”选项卡。从那里，您可以自行决定查看任何作业log ，方法是从工具栏的下拉列表中选择它。如果作业失败并且有错误消息，您可以使用红色箭头图标从一条消息跳到另一条消息。如果没有活动的错误消息，这些图标将被禁用。



作业队列工具栏选项

选项	描述
	单击作业名称并单击此图标可暂停或恢复所选作业。
	单击作业名称和这两个箭头之一以在作业队列中向上或向下移动作业，使其按处理顺序更早或更晚。
	单击尚未开始的作业的作业名称，然后单击此图标可从 作业队列"窗口中删除该作业。
	单击向上或向下箭头可设置可运行运行的作业数，最多为 8 个。 计数默认为1 ，以便 Job Queue 一次处理一个作业，First In First输出。

代码矿工脚本

代码矿工系统使用一组数据库来提供对从现有源代码派生的信息的快速和全面的访问。 Enterprise Architect的代码编辑器的智能感知特征及其搜索工具可以利用从这些数据库中挖掘的信息。

通过代码矿工脚本页面，您可以指定用于特定项目的代码矿工数据库，您可以创建、更新和添加新数据库到代码矿工库。 服务"页面允许您指定本地代码矿工库，或者您希望通过 Sparx 英特尔服务访问可用的库。

每个分析器可以指定不同的代码矿工脚本，所以使用的代码矿工库由激活的分析器脚本。

访问

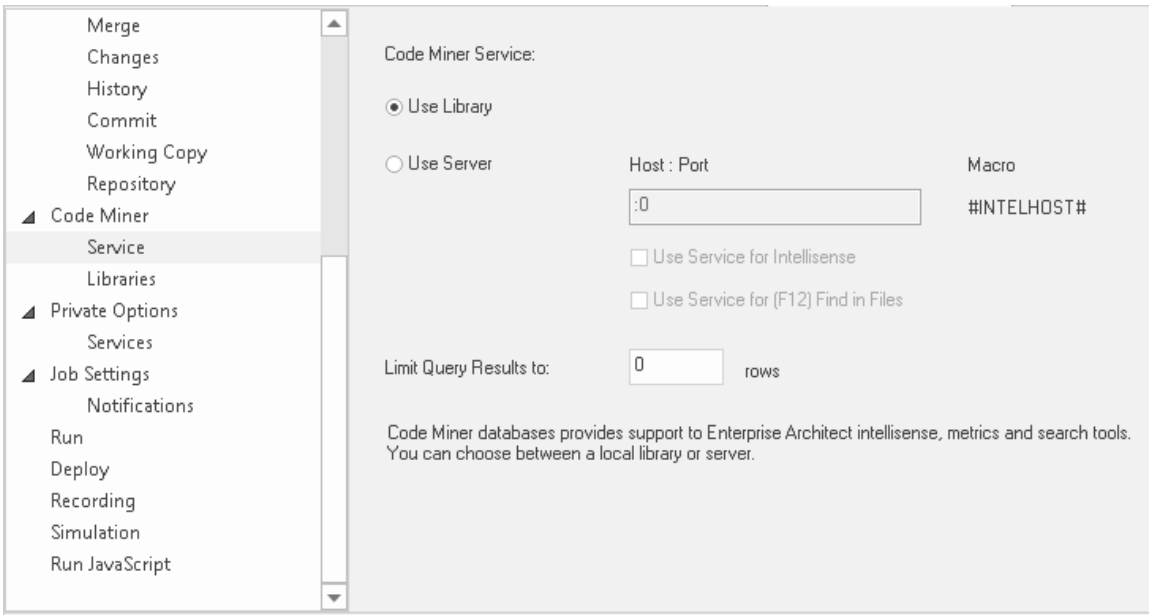
在执行分析器窗口中：

- 找到并双击所需的脚本，然后选择 代码矿工"页面或
- 点击窗口工具栏中的 ，选择要新建脚本的包，选择 代码矿工"页面

功能区	执行>工具>分析器>视图分析器脚本>双击脚本名称>代码矿工>服务 开发>源代码>执行分析器>编辑分析器脚本>双击脚本名称>代码矿工>服务
键盘快捷键	Shift+F12

Sparx 英特尔服务

代码矿工A可以在本地使用，也可以部署到可以为多个客户端提供服务的服务器位置。您在分析器脚本的 代码矿工脚本"页面上选择要使用的分析器。



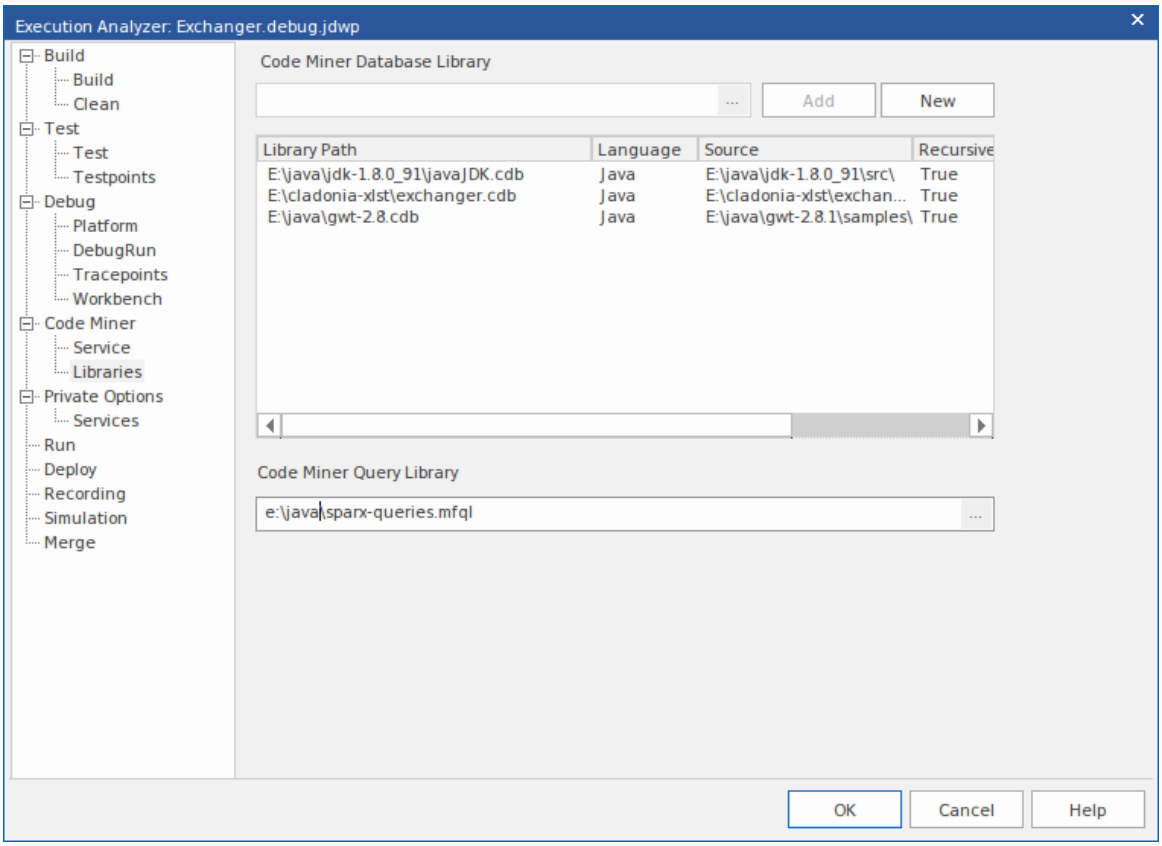
代码矿工库

代码矿工数据库库是包含从源代码派生的信息的数据库集合。一起，这些代码矿工库由Enterprise Architect特征

智能感知数据库使用。通常，将为每个框架或项目创建一个库。代码矿工库页面允许创建新数据库，以及从库中添加、更新或删除现有数据库。

代码矿工查询库是一组函数，用代码矿工的 mFQL 语言编写，捆绑到一个源文件中。

给定分析器的数据库代码矿工库和查询库在 代码矿工脚本”中指定。脚本编辑的图书馆页面。



服务脚本

分析器的 服务”页面描述了各种脚本可视化执行分析器（导入项目、生成可执行状态机）创建脚本时使用的默认端口。您可以在此页面上更新任何端口规范。

访问

在执行分析器窗口中：

- 找到并双击所需的脚本，然后选择 私人选项 |服务页面或
- 单击窗口工具栏中的  ，选择要在其中创建新脚本的包，然后选择 Private Options |服务页面

功能区	开发>源代码>执行分析器>编辑分析器脚本 执行 > 工具 >分析器
键盘快捷键	Shift+F12

合并脚本

分析器脚本中A分析器脚本为用户提供了执行某些操作的附加命令。合并操作取决于您的要求。

访问

在执行分析器窗口中：

- 找到并双击所需的脚本，然后选择 合并”页面或
- 点击窗口工具栏中的  ，选择要创建新脚本的包，然后选择'Merge'页面

功能区	开发>源代码>执行分析器>编辑分析器脚本 执行>源>合并
键盘快捷键	Ctrl+Alt+M

记录脚本

录音的美妙之处并不在于我们总能看到更大的画面，而是有机会看到一个有一些真相的小画面。我们都见过不太有用的序列图。（同一条消息在图表上连续出现 *100* 次确实可以告诉我们一些信息，但不多。）幸运的是，Enterprise Architect通过使用片段来处理第一点。重复行为被标识为模式，并在序列图上以片段的形式表示一次。该片段根据迭代次数进行标记。当然，记录历史总是显示整个历史。我们还需要工具来帮助我们聚焦特定感兴趣区域的记录并减少其他人的噪音。我们可以使用过滤器来做到这一点。使用过滤器，您可以从任何记录中排除任何类、函数甚至模块。您可以创建多组过滤器并将它们与标记集一起使用以针对不同的使用案例。

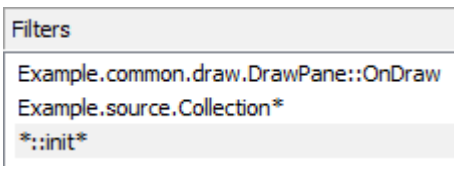
访问

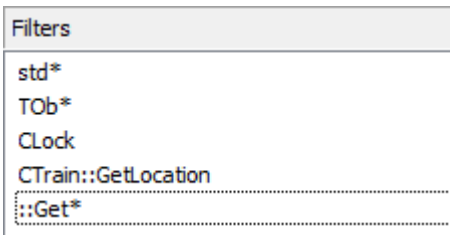
在执行分析器窗口中：

- 找到并双击所需的脚本，然后选择 记录”页面或
- 点击窗口工具栏中的  ，选择要新建脚本的包，然后选择 记录”页面

功能区	开发>源代码>执行分析器>编辑分析器脚本 执行 > 工具 >分析器
键盘快捷键	Shift+F12

过滤器Strings

元素	讨论
过滤	如果在执行分析器脚本器的 记录”页面上选中了 启用过滤器”复选框，则调试器将从记录中排除对匹配方法的调用。比较区分大小写。 要添加值，请单击 排除过滤器”框右上角的 新建”（ 插入”）图标，然后输入比较string ；每个过滤器string采用以下形式： class_name_token::method_name_token class_name_token 排除了对名称与令牌匹配的类或类的所有方法的调用；string可以包含通配符 *（星号）。 method_name_token 不包括对名称与令牌匹配的方法的调用；同样， string可以包含通配符 *。 两个令牌都是可选的；如果没有类标记，则过滤器仅应用于全局或公共函数（即不属于任何类的方法）。
示例	在此Java示例中，调试器将排除： • 调用类示例的 OnDraw 方法 • 调用任何名称以示例.源.Collection 开头的类的任何方法 • 调用任何类的任何构造函数（例如 <clint> 和 <init>） 

	在本机代码示例中，调试器将排除： • 对标准模板库命名空间的调用 • 对任何以类开头的类的调用 • 调用类CLock的任何方法 • 调用类CTrain的GetLocation方法 • 调用名称以 Get 开头的任何全局或公共函数 

过滤器

使用过滤器Entry	到过滤器
：：得到*	录制会话中名称以“Get”开头的所有公共函数（例如，窗口中的 GetClientRect）。
：：得到	任何类中以“Get”开头的所有方法。
C类::获取*	CClass类的所有以 Get 开头的方法。
C类::*	CClass类的所有方法。
ATL* 标准*	属于标准模板和活动模板库的类的所有方法。
CClass::GetName	CClass类的特定方法 GetName。

部署脚本

这些部分解释了如何创建用于部署当前包的命令脚本。可以通过选择 执行>源>编译>部署”功能区选项或按 Ctrl+Shift+Alt+F12 来执行脚本。

访问

在执行分析器窗口中：

- 找到并双击所需的脚本，然后选择 部署”页面或
- 点击窗口工具栏中的  ，选择要在其中创建新脚本的包，然后选择 部署”页面

功能区	开发>源代码>执行分析器>编辑分析器脚本 执行 > 工具 >分析器
键盘快捷键	Ctrl+Shift+Alt+F12

行动

行动	细节
执行命令为：	**Process** If the deployment is handled externally, enter the path to the program or batch file to run, followed by any parameters; the program is launched in a separate process. Example: C:\apache-ant-1.7.1\bin\ant.cmd myproject deploy **Batch File** When using this option, you can enter multiple commands that are then executed as a single script in a command console; you have access to any environment variables available in a standard command console. Example: @echo on IF NOT EXIST "%1%" GOTO DEPLOY_NOWAR IF "%APACHE_HOME%" == "" GOTO DEPLOY_NOAPACHE xcopy /L "%1%" "%APACHE_HOME%\webapps" GOTO DEPLOY_END rem rem NO WAR FILE rem :DEPLOY_NOWAR echo "%1% WAR file not found" GOTO DEPLOY_END rem

	rem NO APACHE ENVIRONMENT VARIABLE rem :DEPLOY_NOAPACHE echo "APACHE_HOME environment variable not found" :DEPLOY_END pause
解析输出	从列表中选择解析器会导致部署脚本的输出被捕获；根据从列表中选择的语法解析输出。 要显示系统输出窗口，请选择 开始>所有窗口>设计>探索>系统”功能区选项。

运行脚本

本节介绍如何创建用于运行可执行代码的命令。

访问

在执行分析器窗口中：

- 找到并双击所需的脚本，然后选择 运行”页面或
- 点击窗口工具栏中的  ，选择要新建脚本的包，选择 运行”页面

功能区	开发>源代码>执行分析器>编辑分析器脚本 执行 > 工具 >分析器
键盘快捷键	Ctrl+Alt+N

脚本元素

元素	描述
命令	这是当您选择 执行>运行>开始”功能区选项时执行的命令；在最简单的情况下，脚本将包含要运行的文件的位置和名称。
例子	这两个示例显示了配置为在Enterprise Architect中运行和Java应用程序的脚本。 。网： C:\benchmark\cpp\example_net_1\release\example.exe Java ： 顾客 此字段中列出的命令就像来自命令提示一样执行；因此，如果可执行路径或任何参数包含空格，它们必须用引号括起来。

注记

- Enterprise Architect提供了正常启动应用程序或从同一脚本进行调试的能力； '分析器' 菜单有单独的启动正常运行和调试运行的选项

调试脚本

配置分析器的调试部分的过程脚本是一次性的，很少需要重新访问。因此，一旦您的脚本正常工作，您可能就不必再考虑它了。您提供的细节并不复杂，但定义脚本提供了许多好处，例如：

- 调试
- 序列图记录
- 可执行状态机执行与模拟
- 测试域创作和记录
- 各种运行时进程的行为分析

您需要做的就是选择合适的平台并输入一些基本细节。您可以使用的调试器平台包括：

- Java
- Java调试Wire Protocol (JDWP)
- 微软.NET调试器
- Microsoft Native Code调试器(C++, C, VB)
- 单核细胞增多症
- PHP调试器
- GNU调试器(GDB)

访问

功能区	开发>源代码>执行分析器>编辑分析器脚本 执行 > 工具 >分析器
键盘快捷键	F6

注记

- 调试Enterprise Architect模型脚本（例如JavaScript或 VBScript）不需要分析器脚本

运营系统具体需求

Enterprise Architect调试器能够在许多不同的平台上运行。此表描述了在每个平台上进行调试的个别要求。

平台

平台	细节
微软.NET	• Microsoft™ .NET框架 4.0、3.5 和 2.0 • 语言支持：C、C#、C++、J#、VB.NET
Java	• Oracle™ 的Java SE 开发工具包（最低版本 5.0）（32 位或 64 位 JDK） Java平台调试器架构(JPDA) 是在Java SE 5.0 版中引入的。 JPDA 提供了两种调试协议； Java虚拟机工具 (接口) 和Java接口调试连线协议 (JDWP)。 Enterprise Architect的调试器支持这两种协议。
GNU调试器(GDB)	Enterprise Architect支持使用 GNU调试器进行调试，它使您能够在 Linux 下本地或远程调试您的应用程序。 需要 GDB 7.0 或更高版本。 源代码文件路径不能包含空格。
原生应用程序窗口	Enterprise Architect支持调试使用 Microsoft™ 编译器编译的本地代码（C、C++ 和 Visual Basic），其中关联的 PDB 文件可用。
PHP	Enterprise Architect使您能够在 Web 服务器中执行 PHP 脚本的本地和远程调试。 需要将 Web 服务器配置为支持 PHP。 需要将 PHP 配置为支持 XDebug PHP（第 3 方 PHP 扩展）。

注记

- Enterprise Architect的所有版本都提供调试功能

支持 UAC 的操作系统

微软系统窗口7提供用户账户控件（UAC）来管理应用程序的安全性。

Enterprise Architect可视化执行分析器是 UAC 兼容的，启用 UAC 的系统的用户可以在只有用户组成员的帐户下使用可视化执行分析器和相关功能执行操作。

但是，当附加到在启用 UAC 的操作系统上作为服务运行的进程时，可能需要以管理员身份log 。

以管理员身份登录

节	行动
1	在运行Enterprise Architect之前，右键单击桌面上的Enterprise Architect图标并选择 以运行身份运行”选项。

或者

编辑或创建指向Enterprise Architect的链接，并配置指向以运行身份运行的链接。

节	行动
1	右键单击Enterprise Architect图标并选择 属性”选项。 显示Enterprise Architect的 属性”对话框。
2	单击高级按钮。 显示 高级属性”对话框。
3	选中 以运行身份运行”复选框。
4	单击确定按钮，然后再次单击 Enterprise Architect属性”对话框。

WINE 调试

配置在WINE下调试Enterprise Architect

节	行动
1	在命令行中，运行$运行。
2	选择 应用程序”选项卡。从Enterprise Architect安装文件夹中添加Enterprise Architect可执行文件“EA.exe”。然 从 VEA 子目录添加这些程序： • SSampler32.exe • SSampler64.exe • SSProfiler32.exe • SSProfiler64.exe
3	依次选择每个程序，然后切换到 库”选项卡。确保以（本机、内置）优先级列出这些值： • 数据库帮助 • msxml4 • msxml6
4	将应用程序源代码和可执行文件复制到您的瓶子中。 路径必须与编译后的版本一致；那是： 如果窗口源= C:\源\SampleApp，Crossover 下必须是C:\源\SampleApp。
5	复制应用程序使用的任何 Side-By-Side 程序集。

权限

Enterprise Architect的安装包含一些本地 Linux 程序，这些程序在Wine下为Enterprise Architect提供构建和调试服务。这些程序需要使用 Linux 文件系统或 shell 进行检查，以确保它们具有适当的 执行”权限设置。这些程序位于Enterprise Architect安装的 VEA/x86/linux”子目录中。

访问违规异常

由于WINE处理直接绘制和访问 DIB 数据的方式，在调试窗口工具栏的下拉菜单中提供了一个附加选项，用于忽略或处理程序直接访问 DIB 数据时引发的访问冲突异常。

选择此选项可捕获真正的（意外）访问违规；取消选择它以忽略预期的违规行为。

由于调试器无法区分预期和意外违规，您可能必须使用试错法来捕获和检查真正的程序崩溃。

注记

- 如果WINE崩溃，后面的痕迹可能不正确

- 如果您使用 MFC，请记住将调试并行程序集复制到 C:\window\winsxs 目录
- 要将 windows 路径添加到WINE ，请修改注册表项：
  HKEY_LOCAL_MACHINE\系统\CurrentControlSet\控件\会话管理器\环境

Java

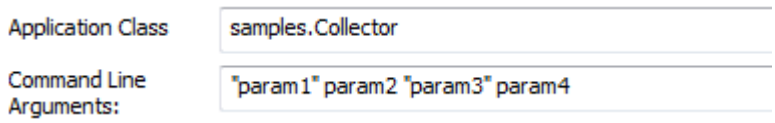
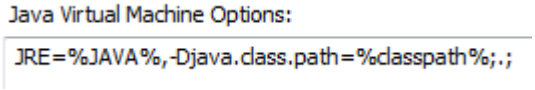
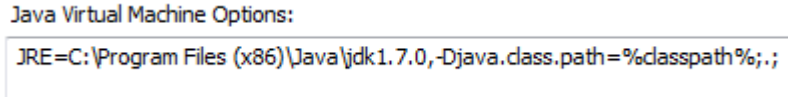
本节介绍如何设置Enterprise Architect以调试Java应用程序和网络服务器。

Java的常规设置

调试Java应用程序的一般设置支持两个选项：

- 调试应用
- 附加到正在运行的应用程序

选项1 -调试应用程序

字段	行动
调试器	选择Java 。
x64	如果您正在调试 64 位应用程序，请选中此复选框。 如果您正在调试 32 位应用程序，请取消选中该复选框。
模式	选择运行。
默认目录	创建Java虚拟机时，此路径会添加到类路径属性中。
应用类	确定要调试的全限定类名；类必须有一个用这个签名声明的方法： public static void main(字符串()); 
命令行参数	指定要传递给 Application类的 main 方法的任何参数。 包含空格的参数应该用双引号括起来。
Java虚拟机选项	指定用于创建虚拟机的命令行选项。 您还必须为Java运行时环境(JRE) 提供一个参数作为搜索 jvm.dll 的路径；这是 Sun Microsystems ™作为运行时环境或 JDK 的一部分提供的 DLL。 JRE 参数可以是： • 企业架构师定义的本地路径 • 用于调试的Java JDK 安装文件夹的绝对文件路径（不带双引号） JRE 参数必须指向Java JDK 的安装文件夹。要成功调试，必须A JDK。 JRE 不应指向公共Java运行时环境（如果已安装）的安装。在指定 VM 启动选项时可以使用环境变量，例如类路径。 例如，使用： • Enterprise Architect Local Path JAVA 和环境变量类路径：  • 或者 JDK 安装目录的绝对路径和环境变量类路径： 

	在这两个示例中，调试器将使用位于 JRE 参数值的 JDK 创建一个虚拟机。 如果未指定类路径，则调试器始终创建虚拟机，其类路径属性等于环境变量中包含的任何路径加上在此脚本的默认工作目录中输入的路径。 如果源文件和 .class 文件位于不同的目录树下，classpath属性必须包括源文件的根路径和二进制类文件的根路径。

选项 2 - 附加到虚拟机

附加到 VM 时几乎不需要指定；但是，VM 必须加载Sparx Systems调试代理。

字段	行动
调试器	选择Java
模式	选择附加到虚拟机

高级技术

除了标准的Java调试技术，您还可以：

- Attach to Virtual Machine
- Internet Browser Java Applets

附加到虚拟机

您可以通过附加到托管Java虚拟机的进程来调试Java应用程序；您可能希望这样做以附加到 Tomcat 或 JBOSS 等网络服务器。

Sun Microsystems 的Java Virtual Machine Tools接口是Enterprise Architect使用的 API；它允许在创建 JVM 时指定调试代理。

要从Enterprise Architect调试正在运行的 JVM，必须在启动时将Sparx Systems的调试代理指定为 JVM 的启动选项； Tomcat 和 JBOSS 等产品如何实现这一点应由该产品自己的文档提供。

对于 java.exe，加载Enterprise Architect调试代理的命令行选项可能是（取决于您的环境）：

- -agentpath:"c:\程序文件\sparx 系统\ea\VEA\x86\SSJavaProfiler32"
- -agentpath:"c:\程序文件 (x86)\sparx 系统\ea\VEA\x86\SSJavaProfiler32"
- -agentpath:"c:\程序文件 (x86)\sparx 系统\ea\VEA\x64\SSJavaProfiler64"

适当的选项取决于您的操作系统以及您是在使用 32 位应用程序还是 64 位应用程序。

或者，如果将适当的 VEA 目录添加到 PATH 环境变量中，则可以选择使用：

- -agentlib:SSJavaProfiler32
- -agentlib:SSJavaProfiler64

附加到虚拟机时无需配置分析器脚本您可以只使用分析器工具栏之一上的附加按钮。

如果你配置一个分析器脚本那么只有两件事是必须选择的：

- 选择 "Java "作为调试平台
- 选择 附加到虚拟机"选项

互联网浏览器Java Applets

本主题描述了从Enterprise Architect调试在浏览器中运行的Java Applet 的配置要求和过程。

从Enterprise Architect附加到托管Java虚拟机 (JVM) 的浏览器进程

节	行动
1	确保要调试的小程序代码的二进制文件已使用调试信息构建。
2	使用Java控件面板配置JVM。
3	在 "Java Applet 运行时设置"面板中，单击视图按钮。
4	在要使用的已安装版本上，在 "运行时参数"字段中包含以下选项之一，具体取决于您的环境以及您使用的是 32 位应用程序还是 64 位应用程序： -agentpath:"c:\程序文件\sparx 系统\ea\VEA\x86\SSJavaProfiler32" -agentpath:"c:\程序文件 (x86)\sparx 系统\ea\VEA\x86\SSJavaProfiler32" -agentpath:"c:\程序文件 (x86)\sparx 系统\ea\VEA\x64\SSJavaProfiler64"
5	在此字段中添加所需的类路径。 这些路径中至少有一个应包含用于调试的源文件的根路径。
6	设置断点。
7	启动浏览器。
8	从Enterprise Architect附加到浏览器进程。

使用Java网络服务器

如果您在Enterprise Architect中调试 JBOSS 和 Apache Tomcat 等Java Web 服务器（包括服务器配置和窗口服务配置），请应用这些配置要求和过程。

注记：Oracle 的Java服务器平台 "Weblogic"不支持可视化执行分析器的调试和记录特征。

从Enterprise Architect附加到托管Java虚拟机的进程

节	行动
1	用于调试 Web 服务器代码的编译二进制文件，带有调试信息。
2	使用 "虚拟机启动"选项启动服务器，如服务器配置中所述。
3	导入源代码导入Enterprise Architect模型，或同步现有代码。
4	设置断点。
5	启动客户端。
6	从Enterprise Architect附加到流程。

服务器配置

Web 服务器与Enterprise Architect交互所需的配置必须解决这两个基本点：

- 任何要由服务器调试、创建或托管的 VM 都必须指定Sparx Systems代理人命令行选项或在 VM 启动选项中指定（即：
  -agentlib:SSJavaProfiler32 或 -agentlib:SSJavaProfiler64)
- CLASSPATH，不管是传给VM的，都必须指定包源文件的根路径

Enterprise Architect调试器使用被调试VM 中的类属性，定位执行过程中发生的断点对应的源文件；例如，要调试的类称为：

abC

这位于物理目录中：

C:\源\a\b

因此，要成功调试，CLASSPATH 必须包含根路径：

c:\源

分析器脚本配置

使用 "编译"脚本的 "调试"选项卡，为您导入的代码创建一个脚本，然后：

- 选择 "附加到进程"单选按钮，然后在其下方的字段中输入 "附加"
- 在 "使用调试器"字段中，单击下拉箭头并选择 "Java"

所有其他字段都不重要； "目录"字段通常在没有任何类路径属性的情况下使用。

运行调试器

断点可能会显示一个问号。在这种情况下，类可能尚未被 VM 加载。如果即使在您确定包含断点的类已加载后问号仍然存在，那么：

- 服务器正在执行的二进制文件不是基于源代码的
- 调试器无法将断点协调到源文件（检查类路径），或者
- JVM 尚未加载Sparx Systems代理

节	行动
1	运行服务器并检查服务器进程是否已加载Sparx Systems代理人： DLL SSJavaProfiler32.DLL 或 SSJavaProfiler64 使用'使用'或类似的工具来证明服务端进程已经加载了代理。
2	在Enterprise Architect中，打开源代码并设置一些断点。
3	点击Enterprise Architect中的运行调试按钮。 将显示 附加到进程”对话框。
4	选择托管应用程序的服务器进程。
5	点击确定按钮。 调试窗口中会显示A确认消息，说明该进程已附加。

JBOSS服务器

在这个 JBoss 示例中，对于 32 位应用程序，简单 servlet 的源代码位于目录位置：

C:\Benchmark\ Java \JBOSS\Inventory

JBOSS 执行的二进制文件位于以下位置的 JAW.EAR 文件中：

C:\JBOSS\03b-dao\build\distribution

Enterprise Architect调试器必须能够在调试期间定位源文件；为此，它还使用 CLASSPATH，在任何列出的路径中搜索匹配的 JAVA源文件，因此 CLASSPATH 必须包含包根的路径，以便Enterprise Architect在调试期间找到源。

这是执行 JBOSS 服务器的命令文件的摘录；要调试的类在：

com/库存/dto/carDTO

因此，该路径的根被包含在JBOSS_CLASSPATH 中。

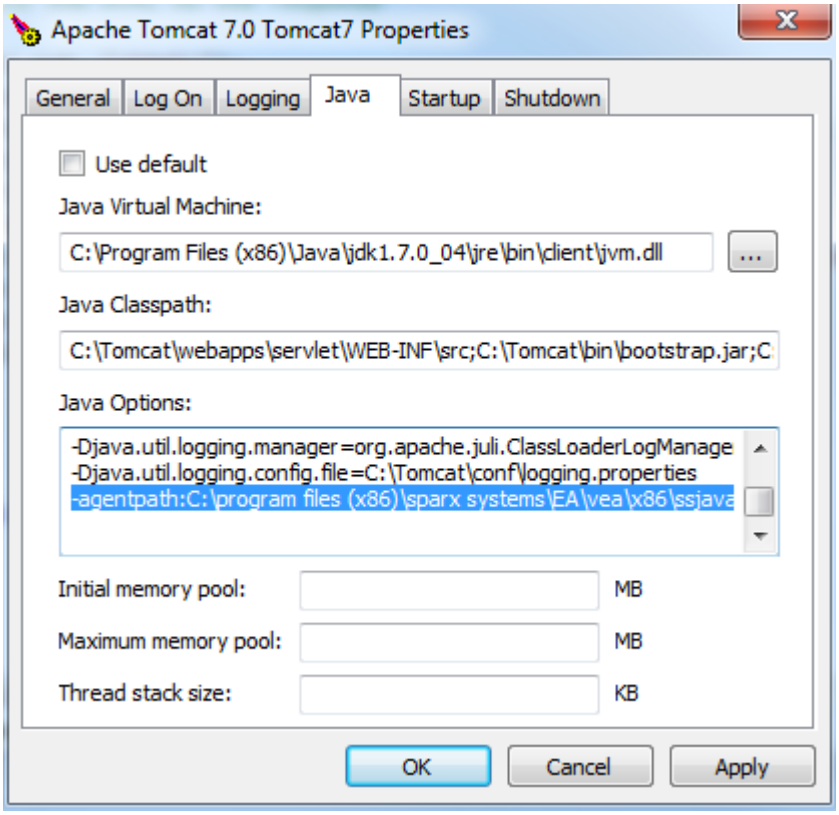
示例代码

RUN.BAT

```
set SOURCE=C:\Benchmark\Java\JBOSS\Inventory
set JAVAC_JAR=%JAVA_HOME%\lib\tools.jar
if "%JBOSS_CLASSPATH%" == ""
(
set JBOSS_CLASSPATH=%SOURCE%;%JAVAC_JAR%;%RUNJAR%;
)
else
(
set JBOSS_CLASSPATH=%SOURCE%;%JBOSS_CLASSPATH%;%JAVAC_JAR%;%RUNJAR%;
)
set JAVA_OPTS=%JAVA_OPTS% -agentpath:"c:\program files\sparx systems\vea\x86\ssjavaprofiler32"
```

Apache Tomcat服务器

Apache Tomcat服务器可以配置为使用Enterprise Architect中的Java调试器进行调试。此示例显示了运行窗口的 PC 上 Apache Tomcat 7.0 的配置对话框。



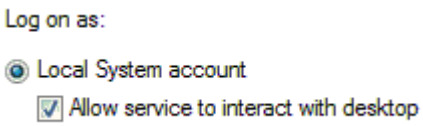
这三点很重要：

- “Java虚拟机”指定安装Java JDK 的运行时
- 将要调试的任何 servlet 的源路径添加到Java Classpath；在这种情况下，我们将路径添加到 Tomcat servlet：c:\tomcat\webapps\servlet\WEB-INF\src
- “Java选项”包括Sparx Systems调试代理的路径：
  -agentpath:c:\程序文件 (x86)\sparx 系统\vea\x86\ssjavaprofiler32

Apache Tomcat窗口服务

配置

对于将 Apache Tomcat作为窗口服务运行的用户，配置该服务以启用与桌面的交互非常重要；不这样做会导致在Enterprise Architect中调试失败。



选中 允许服务与桌面交互”复选框。

.NET

本节介绍如何配置Enterprise Architect以调试.NET应用程序。这包括：

- General Setup for .NET
- Debugging an Unmanaged Application
- Debug COM Interop
- Debug ASP .NET

.NET的常规设置

这是调试 Microsoft .NET应用程序的一般设置。调试时有两种选择：

- 调试一个应用程序
- 附加到正在运行的应用程序

选项1 -调试应用程序

字段	行动
调试器	选择 Microsoft .NET作为调试平台。
x64	如果您正在调试 64 位应用程序，请选中此复选框。 如果您正在调试 32 位应用程序，请取消选中该复选框。
模式	选择运行单选按钮。
默认目录	这被设置为正在调试的进程的默认目录。
申请途径	选择并输入应用程序可执行文件的完整或相对路径。 • 如果路径包含空格，则指定完整路径；不要使用相对路径 • 如果路径包含空格，则路径必须用引号引起来
命令行参数	在启动时传递给应用程序的参数。
显示控制台	为调试器创建一个控制台窗口；不适用于附加到进程。
符号搜索路径	指定任何其他路径来定位调试器的调试符号；用分号分隔路径。

选项 2 - 附加到正在运行的应用程序

字段	行动
调试器	选择 Microsoft .NET作为调试平台。
x64	如果您正在调试 64 位应用程序，请选中此复选框。 如果您正在调试 32 位应用程序，请取消选中该复选框。
模式	选择附加到进程单选按钮。

调试非托管应用程序

如果使用非托管应用程序调试托管代码，调试器可能无法检测到要加载的公共语言运行时 (CLR) 的正确版本。
如果您还没有在脚本的调试命令中指定的调试应用程序的配置文件，您应该指定一个配置文件。
配置文件应与您的应用程序位于同一目录中，并采用以下格式：
名称.exe.config
其中 “名称”是您的应用程序的名称。
您指定的 CLR 版本应与由调试对象调用的托管代码加载的版本相匹配。

在下面的示例代码中，“clr_version”表示您的插件或 COM 代码所针对的 CLR 的版本。

示例配置文件

```
<configuration>
    <startup>
        <requiredRuntime version="clr_version"/>
    </startup>
</configuration>
```

调试COM互操作

Enterprise Architect使您能够在本地或进程内服务器中调试使用 COM 执行的.NET托管代码。
此特征对于调试插件和 ActiveX 组件很有用。

调试使用 COM 执行的.NET托管代码

节	行动
1	在Enterprise Architect中创建一个包并导入代码进行调试。
2	确保 COM 组件是使用调试信息构建的。
3	为包创建脚本。
4	在'调试|平台”页面，您可以选择附加到非托管进程或指定非托管应用程序的路径以调用您的托管代码。
5	在源代码中添加断点进行调试。

附加到非托管进程

如果您正在使用：

- 进程内 COM 服务器，附加到客户端进程
- 本地 COM服务器A附加到服务器进程

单击调试窗口运行按钮（或按 F6）以显示可供选择的进程列表。

注记

- 从您一直在调试的 COM 互操作进程中分离会终止该进程；这是 Microsoft .NET Framework 的一个已知问题，有关它的信息可以在许多 MSDN .NET博客上找到

调试ASP .NET

对 ASP 等 Web 服务的调试要求Enterprise Architect调试器能够附加到正在运行的服务。

首先确保包含 ASP .NET服务项目的目录已导入Enterprise Architect ，如果需要，包含客户端网页的 web 文件夹。

如果您的 web 项目目录位于网站托管目录下，您可以从根目录导入，同时包含 ASP 代码和网页。

必须先启动客户端，因为 ASP .NET服务进程可能尚未运行；使用浏览器加载客户端 - 这可确保 Web 服务器正在运行。

然后在调试设置中选择 附加”单选按钮。选择此选项后，调试器每次都会提示您要调试的进程。

点击调试窗口运行按钮启动调试器；将显示 附加到进程”对话框。

如*ASP .NET SDK*中所述，进程的名称因 Microsoft 操作系统而异；例如，在窗口上，进程的名称类似于 aspnet_wp.exe，尽管该名称可以反映它所支持的.NET框架的版本。

XP下可以有多个ASP.NET进程运行；您必须确保附加到正确的版本，这将是托管您的应用程序运行的.NET框架版本的版本；检查 Web 服务的 web.config 文件以验证它所绑定的.NET框架的版本。

调试窗口停止按钮应该被启用并且任何断点应该是红色的，表明它们已经被绑定。

您可以随时在 Web 服务器代码中设置断点。如果您导入它们，您还可以在 ASP 网页中设置断点。

注记

一些断点可能没有成功绑定，但如果根本没有绑定（用带问号的深红色表示），则某些断点不同步；尝试重建并重新导入源代码

Mono调试器

Mono 是一个由.NET基金会赞助的软件平台，用于促进跨平台开发。它因其丰富的游戏、基于 API 和特征的特点而受到游戏开发者的欢迎。

Enterprise Architect通过为建模和开发软件提供现代环境来为 Mono 社区提供支持。现有项目可以在 Linux 和窗口上本地导入、构建和调试。

概述

Mono下的调试涉及到三个进程的配合。 Mono 运行时管理应用程序并使用套接字协议与Enterprise Architect调试器通信，后者又与作为前端的Enterprise Architect通信。当您需要指示它以支持调试时，您可以使用命令行指令来实现，在该指令中您命名主机并启动 Mono 应该监听的端口号。主机可以省略，在这种情况下，Mono 将接受来自任何 IP 地址的连接。主机可以具有值 ' localhost ' 来限制与同一台机器的连接。端口号码是您选择的号码。

主机和端口号是重要的信息，在配置分析器脚本使用。

用于需求窗口

- Enterprise Architect （最低版本 14 ）
- 用于窗口的 Mono（最低版本 5.4 ）

Linux需求

- Enterprise Architect （最低版本 14 ）
- 适用于 Linux 的 Mono（最低版本 5.4 ）
- 适用于 Linux 的Wine

运行时主机页面

此页面是可选的，仅在 Mono 和Enterprise Architect将在同一台机器上运行时才有用。它提供了在Enterprise Architect调试器启动之前首先使用所需的调试指令运行的能力。调试器连接后，它会恢复 Mono 运行时，该运行时已以挂起状态启动。如果应用程序在与您正在使用的Enterprise Architect不同的机器上运行，您应该清除此部分。

调试配置Linux

调试器配置

本节介绍Linux下调试脚本的分析器的调试部分。此处未列出的字段不是必需的。

调试器	选择 单声道”。
默认目录	这是应用程序所在的 Unix 格式的完全限定的本机 Linux 路径。
联系	• port：调试端口 • host：运行 Mono 的机器的名称或 IP 地址（如果机器相同，则为 ' localhost ' ） • localpath：窗口格式源代码的根路径；这是用于在Enterprise Architect的代码编辑器中设置断点的源文件的路径 • remotepath：Unix格式源代码的根路径，这是Linux下构建程序的源文件的路径 这些路径在调试事件期间返回，然后映射到本地路径，以便Enterprise Architect可以在断点或步骤期间显示源文件 - 两个参数可以指定相同的物理源文件根，但必须使用窗口或 Unix每个字段的格式 • 关机：（ true or false ）；当调试器停止时，VM 被终止 • timeout：套接字调用的超时时间（以毫秒为单位） • 输出：要写入的log文件的Wine /窗口路径 • 日志记录：（ true or false ）；如果为 true，则在调试窗口中记录额外的消息，并将套接字消息记录到指定的输出文件中

自动启动单声道

您可以将Enterprise Architect配置为在您启动调试器时为您启动 Mono。你可以通过配置你的分析器的'Runtime脚本'页面来做到这一点。命令的格式如下所述：

cd 程序路径

/usr/bin/mono --debug --debugger-agent=transport=dt_socket,address= *host:port* ,server=y,suspend=y程序

在哪里：

- *path-to-program*是程序所在的目录路径

- 主机是其中之一：

- localhost
- IP 地址
- 联网机器名称

- 端口是端口的端口

- *program*是应用程序的名称（例如 MonoProgram.exe）

使用命令行手动启动 Mono

您可以从控制台手动启动 Mono。在文件资源管理器中找到该程序，然后在该位置打开控制台。命令行的格式如下所述：

/usr/bin/mono --debug --debugger-agent=transport=dt_socket,address= *host:port* ,server=y,suspend=y程序

其中主机是其中之一：

- localhost
- IP 地址
- 联网机器名称

*port*是套接字的端口， *program*是应用程序的名称（例如端口）。

调试配置窗口

调试器配置

本节介绍一个分析器脚本关于在窗口下调试 Mono 的调试部分。此处未列出的字段不是必需的。

字段	描述
调试器	选择 单声道”。
x64	选择要调试的程序是否为 64 位可执行文件。
运行或附加	选择 迩行”来命名要启动的程序。如果您将始终附加到正在运行的进程，请选择 附加”。
默认目录	程序运行时将采用的默认目录。
申请途径	Mono 应用程序的完整路径。
命令行参数	要传递给程序的任何参数。如果参数包含空格，请用双引号 (") 将它们括起来

自动启动单声道

您可以将Enterprise Architect配置为在您启动调试器时为您启动 Mono。你可以通过配置你的分析器的'Runtime脚本'页面来做到这一点。命令的格式如下所述：

cd程序路径

mono --debug --debugger-agent=transport=dt_socket,address= *host:port* ,server=y,suspend=y程序

在哪里：

- *path-to-program*是程序所在的目录路径

- 主机是其中之一：

- localhost
- IP 地址
- 联网机器名称

- 端口是端口的端口

- *program*是应用程序的名称（例如 MonoProgram.exe）

使用命令行手动启动 Mono

您可以从控制台手动启动 Mono。在文件资源管理器中找到该程序，然后在该位置打开控制台。命令行的格式如下所述：

mono --debug --debugger-agent=transport=dt_socket,address= *host:port* ,server=y,suspend=y程序

其中主机是其中之一：

- localhost
- IP 地址

- 联网机器名称

*port*是套接字的端口， *program*是应用程序的名称（例如端口）。

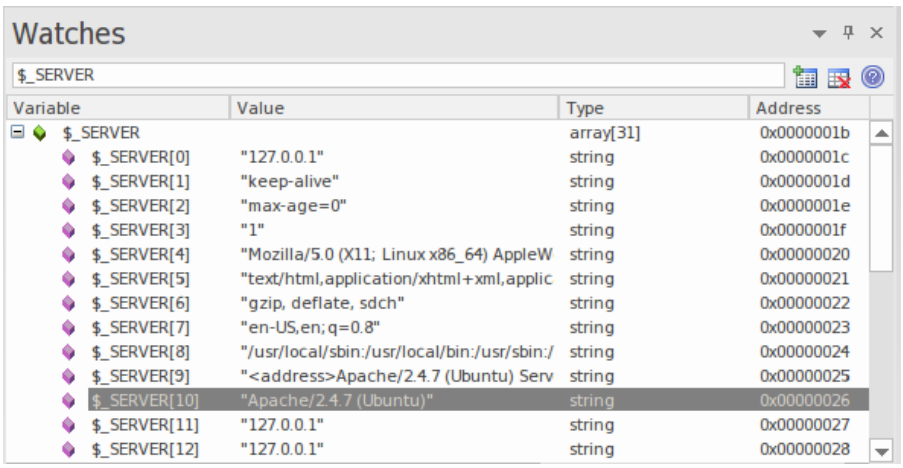
PHP调试器

Enterprise Architect PHP调试器器使您能够调试 PHP.exe 脚本。本节讨论基本设置和常见的各种调试场景；这些场景涉及文件路径的映射，这对于远程调试会话的成功至关重要。

- 脚本Setup
- 本地窗口机器（Apache服务器）
- 本地窗口机 (PHP.exe)
- 远程 Linux 机器（Apache服务器）
- 远程 Linux 机器 (PHP.exe)

设置和场景

设想	细节
脚本Setup	分析器脚本Enterprise Architect中调试的基本要求；您使用执行分析器的工具栏创建一个脚本。 选择PHP.XDebug作为调试平台；当您选择此平台时，属性页面会显示以下连接设置： • host - localhost - Enterprise Architect监听来自 PHP 的传入连接的适配器 • localpath - %LOCAL% - 指定要映射到远程文件路径的本地文件路径；这是一个远程调试设置 - 对于本地调试，清除该值，该值是一个占位符，您应该对其进行编辑以适合您的特定场景 • remotepath - %REMOTE% - 指定本地文件路径要映射到的远程文件路径；这是一个远程调试设置 - 对于本地调试，清除该值，该值是一个占位符，您应该对其进行编辑以适合您的特定场景 • logging - 输入true or false以启用来自 XDebug 服务器的通信记录 • 输出 - 命名远程机器上要与日志记录选项一起使用的文件路径；此文件将始终被覆盖
本地机器 Apache服务器	在这种情况下，请考虑以下配置： • O ： S • 网络计算机名称：MyPC • 网络共享 MyShare 映射到 c:\myshare • Enterprise Architect中的源文件已从 c:\myshare\apache\myapp\scripts 导入 • Apache 文档根设置为 //MyPC/MyShare/apache 在这种情况下，连接参数的分析器脚本可能配置为： • 主机： localhost • 端口：9000 • 本地路径：c:\myshare\apache\ • 远程路径：MyPC/MyShare/apache/
本地机器 PHP.EXE	在这种情况下，连接参数的分析器脚本可能会被配置为如图所示，因为文件路径总是映射到相同的物理路径： • 主机： localhost • 端口：9000 • 本地路径：

	• 远程路径：
远程 Linux 机器 Apache服务器	在这种情况下，请考虑以下配置： 本地机器： • O ： S • Enterprise Architect中的源文件已从 c:\myshare\apache\myapp\scripts 导入 远程机器： • O ： S • Apache文档根设置为家/apache/htdocs • Apache 中的源文件位于家/apache/htdocs/myapp/scripts 在这种情况下，连接参数的分析器脚本可能配置为： • 主机： localhost • 端口：9000 • 本地路径：c:\myshare\apache\ • 远程路径：家/apache/htdocs/
远程 Linux 机器 PHP.exe	在这种情况下，请考虑以下配置： • 本地机器 • O ： S • Enterprise Architect中的源文件已从 c:\myshare\apache\myapp\scripts 导入 • 远程机器 • O ： S • Apache 中的源文件位于家/myapp/scripts 在这种情况下，连接参数的分析器脚本可能配置为： • 主机： localhost • 端口：9000 • 本地路径：c:\myshare\apache\ • 远程路径：家/
PHP 全局变量	当您处于断点时，您可以使用分析器Watches 窗口检查 PHP 全局变量的值。要列出每个全局变量，请在字段中输入 "globals"或 "superglobals"。要显示单个项目，请输入其名称。此图像显示正在显示的 PHP 环境变量 $_SERVER 的值。 

PHP调试器-系统需求

本主题确定Enterprise Architect PHP 调试器的系统要求和操作系统。

系统需求：

- Enterprise Architect版本 9
- PHP 5.3 或以上版本
- PHP zend 扩展 XDebug 1或以上
- 对于 Apache 等 Web 服务器，支持 PHP 版本的服务器版本

支持的操作系统：

- 客户（ Enterprise Architect ）
- Microsoft窗口及以上版本
- Linux 运行 Crossover Office
- 服务器(PHP)
- Microsoft窗口及以上版本
- Linux

PHP调试器检查清单

本主题提供了在Enterprise Architect中调试 PHP 脚本的故障排除指南。

选择积分

选择点	细节
系统需求	• Apache HTTP网络服务器版本 2.2 • PHP 5.3 或以上版本 • XDebug 版本 2. 1 . 1
Enterprise Architect	• 该模型有一个分析器配置为使用PHP脚本平台 • PHP源代码已导入模型（用于记录和测试点） • 当从 "分析器"对话框中选择 PHP脚本平台时，默认运行时设置会在 "连接"字段中列出： 本地路径：%LOCAL% 远程路径：%REMOTE% 为这些默认变量定义本地路径或编辑脚本以提供实际路径。 例如：本地源、远程源 本地路径：c:\code samples\vea\php\sample 远程路径：网络服务器/示例 • 'webserver' 是网络或本地共享 • 'sample' 是共享下面的文件夹
PHP	为了在Enterprise Architect中调试 PHP 脚本，需要正确配置 PHP 以加载 XDebug 扩展。 应使用与这些类似的设置： • [xdebug] • xdebug.extended_info=1 • xdebug.idekey=ea • xdebug.remote_enable=1 • xdebug.remote_handler=dbgp • xdebug.remote_autostart=1 • xdebug.remote_host=XXXX • xdebug.remote_port=9000 • xdebug.show_local_vars=1 IP 地址 XXXX 指并应与模型分析器脚本指定的主机相匹配。 IP 地址是 XDebug 连接的地址，也是Enterprise Architect PHP 代理监听的地址。
阿帕奇	对于使用 Apache 进行调试，这些行应该存在于 Apache 配置文件 httpd.conf 中： LoadModule php5_module "php_home/php5apache2_2.dll" AddHandler 应用程序/x-httpd-php .php

	PHPIniDir "php_home" 值 "php_home"是PHP安装路径（php.ini和apache dll存在的路径）。
故障排除	为了防止调试会话期间 PHP 和 Apache 超时，这些设置可能需要修改。 在Enterprise Architect中开发 PHP调试代理时使用了这些设置。
PHP	文件：php.ini ; Enterprise Architect在调试 PHP 扩展时防止 PHP 超时 max_execution_time = 0 ; Enterprise Architect在调试 PHP 扩展时防止 Web 服务器超时 最大输入时间 = -1 ; Enterprise Architect记录错误 display_errors = 开 ; Enterprise Architect显示启动错误 display_startup_errors = 开
阿帕奇	文件：httpd.conf ; Enterprise Architect在调试 php 扩展时防止超时 超时 60000

GNU调试器(GDB)

在调试您的应用程序时，您可以使用 GNU调试器(GDB)，它是可移植的，可在 Linux 等类 Unix 系统以及窗口上运行。 GDB 适用于多种编程语言，包括 Ada、 Java 、C、C++ 和 Objective-C。使用 GDB，您可以在本地或远程调试您的应用程序。

访问

在执行分析器窗口中：

- 找到并双击所需的脚本，然后选择 调试>平台"页面或
- 点击窗口工具栏中的  ，选择要新建脚本的包，选择 调试>平台"页面

功能区	开发>源代码>执行分析器>编辑分析器脚本 执行 > 工具 >分析器
上下文菜单	浏览器窗口 |右键单击包|执行分析器
键盘快捷键	Shift+F12

设置 GNU调试器

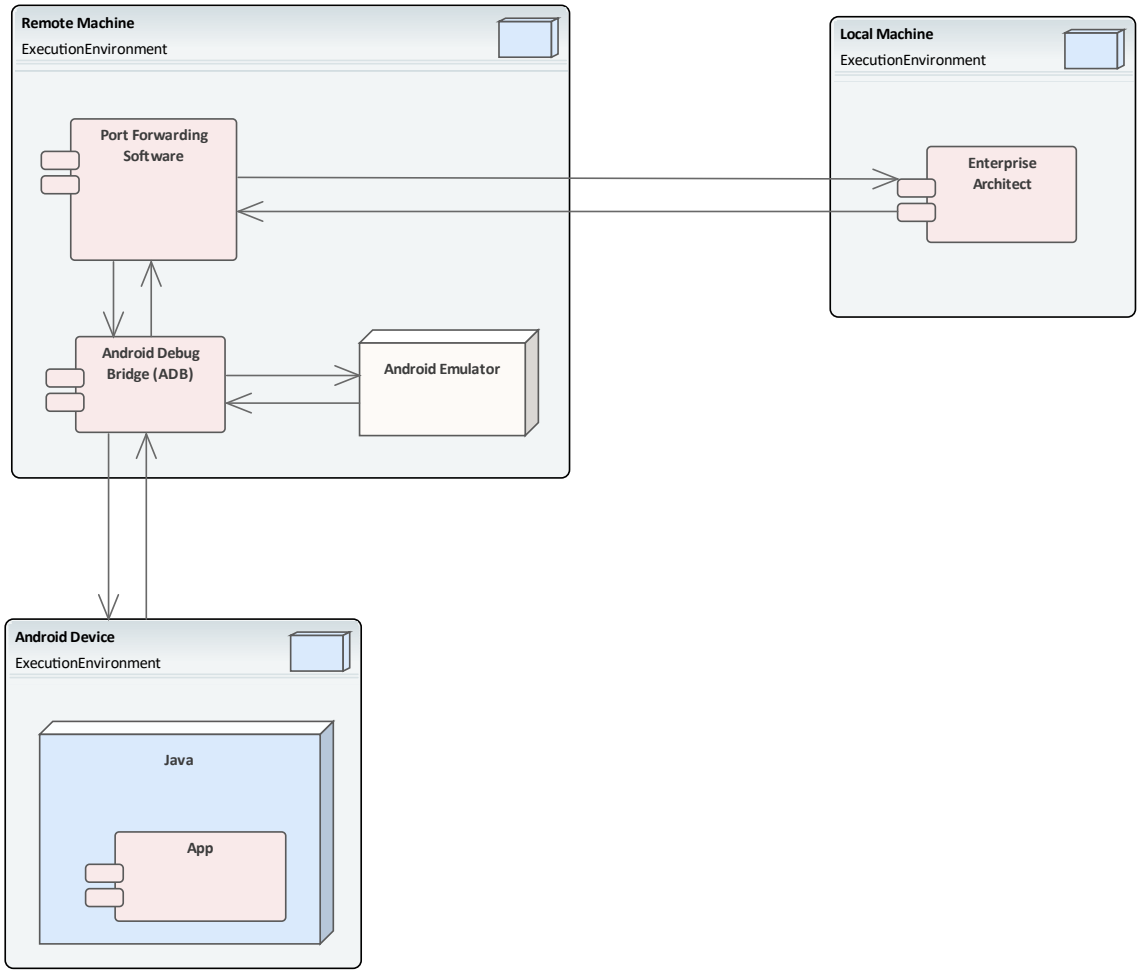
任务	细节
设置脚本	分析器脚本Enterprise Architect中调试的基本要求；您使用执行分析器工具栏创建一个脚本。 在执行分析器脚本器的 平台"页面，在 调试器"字段中点击下拉箭头并选择 "GDB"。
定义连接设置	属性面板显示您为其提供值的许多连接设置。 • path - <path> - GDB 可执行文件的完成文件路径；只有在系统路径中找不到 GDB 时才指定此项 • 源- <path>, <path> - 调试器搜索源文件的路径，如果它们不驻留在可执行目录中 • remote - F - 设置为远程调试；否则留空 • port - <端口> - 远程服务器上要连接的端口 • host - localhost - 要连接的主机名 • fetch - T - 设置从远程系统检索二进制文件 • dumpgdb - <path> - 将 GDB 输出写入到的文件名 • initpath - <path> -完成文件的完成文件路径

注记

- GDB 的A要求是你的源代码文件路径不能包含空格；调试器将无法运行运行文件路径中的空格

Android调试器

如果您正在开发在 Android 设备或模拟器上运行的Java应用程序，您还可以调试它们。本地和远程机器可以在 32 位平台或 64 位平台上。



系统需求

在远程机器上，需要此软件：

- Android SDK，包括android调试桥，ADB（你需要熟悉SDK及其工具）
- Java JDK（32 位和 64 位支持）
- 端口转发软件（第3方）

在本地计算机上，需要此软件：

- Enterprise Architect 10 或更高版本

分析器脚本

字段/按钮	行动
调试器	单击下拉箭头并选择Java (JDWP)。

运行	单击此单选按钮。
默认目录	不适用 - 留空。
申请途径	不适用 - 留空。
命令行参数	不适用 - 留空。
先编译	不适用 - 留空。
显示控制台	不适用 - 留空。
显示诊断消息	不适用 - 留空。
联系	不适用 - 留空。
端口	这是应用程序端口，使用 adb 或其他方式前向分配， Enterprise Architect和 Android 虚拟机 (VM) 可以通过该端口进行通信。
主持人	主机（默认为localhost ） 如果 Android 在连接到联网计算机的设备上的模拟器上运行，请在此处输入网络名称。 默认情况下，调试将尝试连接到您在本地机器上指定的端口。
源	这是Java中类路径设置的源等价物。 应该列出每个源树的根。如果指定了多个，则应以分号分隔；那是： c:\myapp\src;c:\myserver\src 您必须至少指定一个根路径。 当断点发生时，调试器会在此处列出的每个源树中搜索 java源。
日志记录	允许记录来自调试器的附加信息 可能的值：真、 false 、 1 、是、否
输出	指定要写入的本地log文件的全名。 该文件夹必须存在，否则不会创建log 。 log文件通常包含调试器和 VM 之间发送的字节转储。
平台	如果要调试在任何 android 场景下运行的Java ，请选择 Android。 对于所有其他方案，请选择Java 。

配置端口用于调试-端口转发（本地）

调试器一次只能调试一个虚拟机；它使用单个端口与 VM 通信。可以使用 Android SDK 提供的 ADB 分配要调试的应用程序的端口。

在调试之前，在设备中启动一次应用程序。当应用程序启动时，发现它的进程标识符（pid）：

亚行 jdwp

列出的最后一个数字是上次启动的应用程序的 pid；注记pid 并使用它来允许调试器连接到 VM：

- adb forward tcp:port jdwp:pid
  - 端口 = 分析器脚本中列出的端口号
  - pid = 设备上应用程序的进程 ID

配置端口用于调试-端口转发（远程）

要进行远程调试，应该遵循与本地计算机相同的过程，但通信需要额外的转发，因为使用 adb forward 命令创建的套接字将仅在本地适配器上侦听。套接字绑定到localhost ，并尝试连接到此端口。将遇到 连接被拒绝”消息

为了实现远程调试，需要在远程机器上运行一个代理，该代理侦听所有传入连接并将所有流量转发到 adb；端口有许多软件产品可以做到这一点。

除非您使用Enterprise Architect配置了代理端口转发器，否则远程调试将不起作用。

Java JDWP调试器

Java提供了两种主要的调试技术：一种基于进程内代理的系统，称为Java虚拟机工具接口(JVMTI)，另一种是基于套接字的范例，称为Java调试连线协议 (JDWP)。 Java A机可以命名其中之一，但不能同时命名两者，并且必须在启动 JVM 时配置特征。

系统需求

1. Enterprise Architect JDWP 调试器将只能与使用 "JDWP"选项启动的 JVM 通信。以下是命令行选项的示例：
   java -agentlib:jdwp=transport=dt_socket,address=localhost:9000,server=y,suspend=n -cp "c:\java\myapp;%classpath%" demo.myApp "param1" "param2"
2. 虚拟机当前不应附加到调试器。
3. Enterprise Architect和 Eclipse 不能同时调试 VM。

分析器脚本

字段/按钮	行动
调试器	单击下拉箭头并选择Java (JDWP)。
运行	单击此单选按钮可在执行脚本时运行调试器。
默认目录	不适用 - 留空。
申请途径	不适用 - 留空。
命令行参数	不适用 - 留空。
先编译	不适用 - 留空。
显示控制台	不适用 - 留空。
显示诊断消息	不适用 - 留空。
联系	不适用 - 留空。
端口	在Java命令行选项中设置启动期间分配给 VM 进程的应用程序端口转发。
主持人	设置主机（默认为localhost ） 如果 VM 在联网计算机上运行，请在此处输入网络名称或 url。 默认情况下，调试将尝试连接到您在本地机器上指定的端口。
源	这是Java中类路径设置的源等价物。 列出每个源树的根；指定至少一个根路径。如果您指定多个，请用分号分隔它们；例如： c:\myapp\src **;** c:\myserver\src 当断点发生时，调试器会在此处列出的每个源树中搜索Java源。

日志记录	启用或禁用来自调试器的附加信息的日志记录。 可能的值包括： • 真的 • false • 1 • 0 • 是的 • 不
输出	指定要写入的本地log文件的全名。如果文件夹不存在，则不会创建log 。 log文件通常包含调试器和 VM 之间发送的字节转储。
平台	选择Java 。

用于调试的配置端口

调试器一次只能调试一个虚拟机；它使用单个端口与 VM 通信。待调试应用程序的端口是在创建 VM 时分配的。

本地调试

如果Enterprise Architect和Java VM 在同一台机器上运行，您可以执行本地调试。必须在启用 JDWP 传输的情况下启动 VM - 有关命令行选项规范，请参阅 Oracle 的*Java*平台调试器架构*(JPDA)*文档。例如：

java -agentlib:jdwp=transport=dt_socket,address=localhost:9000,server=y,suspend=n -cp "c:\samples\java\myapp;%classpath%" samples.MyApp "param1" "param2"

在这个例子中，分析器脚本的值是'host: localhost '和'port:9000'。

远程调试

当Enterprise Architect在本地机器上运行而Java VM 在远程机器上运行时，您可以执行远程调试。有必要在启用 JDWP 传输的情况下启动 VM - 有关命令行选项规范，请参阅 Oracle 的 JPDA 文档。这是一个示例，其中远程计算机的网络名称为 testmachine1：

java -agentlib:jdwp=transport=dt_socket,address=9000,server=y,suspend=n -cp "c:\samples\java\myapp;%classpath%" samples.MyApp "param1" "param2"

注记地址中没有主机名。这意味着 VM 将侦听来自任何机器的连接。在此示例中，分析器脚本的值将是 ″主机：分析器″和 ″端口：9000″。

追踪点输出

分析器脚本的分析器页面使您能够指导任何跟踪语句的输出在调试会话期间的去向脚本

访问

在执行分析器窗口中：

- 找到并双击所需的脚本，然后选择 调试> Tracepoints"页面或
- 点击窗口工具栏中的  ，选择要新建脚本的包，选择 调试> Tracepoints"页面

功能区	开发>源代码>执行分析器>编辑分析器脚本 执行 > 工具 >分析器> 选择并运行脚本
上下文菜单	浏览器窗口 |右键单击包|执行分析器
键盘快捷键	Shift+F12

跟踪点属性

字段	细节
输出	您可以从两个选项中进行选择： • '屏幕' (默认) - 输出被定向到调试窗口 • '文件' - 输出被定向到文件
文件夹	输入用于跟踪语句log文件的文件夹。
文件名	输入用于跟踪语句log文件的名称。
覆盖	如果选中，则每次启动调试会话时都会覆盖指定的文件。
自动编号	如果选中，则跟踪log文件由您指定的文件名和一个数字组成。 每次启动调试会话时，数字都会增加。
前缀函数的跟踪输出	如果选中，则在调试会话运行期间执行的任何跟踪状态都以当前函数调用为前缀。

工作台设置

本主题描述在Java和 Microsoft .NET上设置物件工作台的要求。

访问

在执行分析器窗口中：

- 找到并双击所需的脚本，然后选择 调试> Workbench"页面或
- 点击窗口工具栏中的 ，选择要新建脚本的包，选择 调试> Workbench"页面

功能区	开发>源代码>执行分析器>编辑分析器脚本 执行 > 工具 >分析器
键盘快捷键	Shift+F12

平台

平台	细节
支持的平台	Workbench 支持以下平台： • Microsoft .NET （2.0 版或更高版本） • Java （JDK 1 .4 或更高版本）
微软.NET工作台	.NET工作台需要一个程序集，用于创建工作台项。 您在分析器的 工作台"页面上指定程序集的脚本。 使用.NET工作台有两个限制： • 不支持在托管代码中定义为结构的成员 • 不支持定义为内部的类
Java工作台	Java工作台使用分析器脚本调试'页面中配置的虚拟机设置来创建JVM。

Microsoft C++ 和本机（C、VB）

只有当可执行文件有相应的 PDB 文件时，您才能调试本机代码。作为构建应用程序的结果，会创建A PDB 文件。

构建应包括完整的调试信息，并且不应设置优化。

脚本必须指定两件事来支持调试：

- 可执行文件的路径
- Microsoft Native 作为调试平台

常规设置

这是调试 Microsoft 本机应用程序（C++、C、Visual Basic）的一般设置。调试时有两种选择：

- 调试一个应用程序
- 附加到正在运行的应用程序

选项1 -调试应用程序

字段	行动
调试器	选择 Microsoft Native 作为调试平台。
x64	如果您正在调试 64 位应用程序，请选中此复选框。 如果您正在调试 32 位应用程序，请取消选中该复选框。
模式	选择运行单选按钮。
默认目录	这被设置为正在调试的进程的默认目录。
申请途径	选择并输入应用程序可执行文件的完整或相对路径。 • 如果路径包含空格，则指定完整路径；不要使用相对路径 • 如果路径包含空格，则路径必须用引号引起来
命令行参数	在启动时传递给应用程序的参数。
显示控制台	为调试器创建一个控制台窗口；不适用于附加到进程。
符号搜索路径	指定任何其他路径来定位调试器的调试符号；用分号分隔路径。

选项 2 - 附加到正在运行的应用程序

字段	行动
调试器	选择 Microsoft Native 作为调试平台。
x64	如果您正在调试 64 位应用程序，请选中此复选框。 如果您正在调试 32 位应用程序，请取消选中该复选框。
模式	选择附加到进程单选按钮。
符号搜索路径	指定任何其他路径来定位调试器的调试符号。 如果您愿意，可以在此处指定符号服务器；用分号或逗号分隔路径。

调试符号

对于使用 Microsoft Platform SDK 构建的应用程序，在构建应用程序时将调试符号写入应用程序 PDB 文件。

可视化执行调试器使用的 API窗口调试工具使用这些符号向执行分析器控件提供有意义的信息。

这些符号很容易过时并导致异常行为 - 调试器可能会在断点处突出显示编辑器中的错误代码行；因此，最好确保在任何调试或记录会话之前构建应用程序。

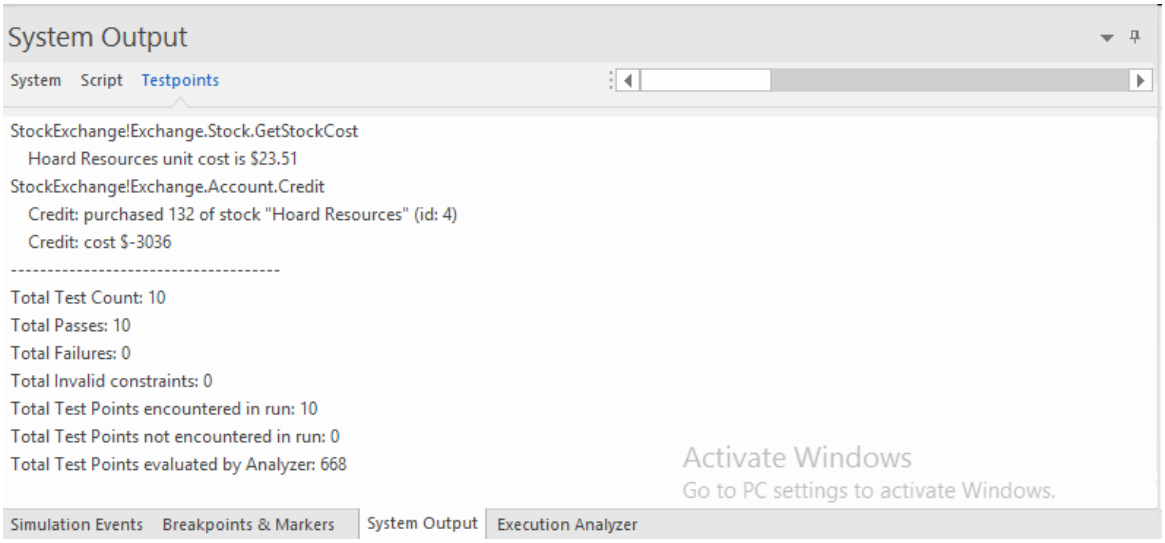
调试器必须通知 API 如何协调正在调试的映像中的地址；它通过指定一些 API 路径来告诉它在哪里寻找 PDB 文件来做到这一点。

对于没有找到调试符号的系统 DLL（kernel32、调用堆栈），二进制显示一些仅带有模块名称和地址的帧。

您可以通过向 API 传递额外的路径来补充翻译的符号；您在 调试"选项卡中以分号分隔的列表传递其他符号路径。

测试点输出

分析器脚本的分析器脚本帮助您配置测试点运行的输出。
默认情况下，输出记录到系统输出窗口，如本例所示。



访问

在执行分析器窗口中：

- 找到并双击所需的脚本，然后选择 测试>测试点"页面或
- 点击窗口工具栏中的 ，选择要新建脚本的包，选择 测试> Testpoints"页面

功能区	开发>源代码>执行分析器>编辑分析器脚本 执行 > 工具 >分析器>视图分析器脚本
键盘快捷键	Shift+F12

选项

选项	描述
输出	您可以从两个选项中进行选择： • '屏幕'（默认）- 输出被定向到系统输出窗口的'测试点'选项卡 • '文件' - 输出被定向到文件
文件夹	单击用于测试点log文件的文件夹。
文件名	输入用于测试点log文件的名称。
覆盖	选择此选项时，每次执行测试点运行时都会覆盖指定的文件。

自动编号	选择该选项时，测试点输出由您指定的文件名和测试运行次数组成；每次执行测试运行时，数字都会增加。
前缀函数的跟踪输出	选择此选项时，在测试点运行期间执行的任何跟踪语句都以当前函数调用为前缀。

测试脚本

这些部分解释了如何配置分析器的 测试”页面来对你的脚本进行单元测试。大多数用户会将其应用于 NUnit 和 JUnit 测试场景。 Enterprise Architect接受来自这些系统的输出，并且可以自动添加和管理每个单元测试用例历史。要查看案例历史记录，您可以选择测试案例类元素并按 Alt+2 >测试。

访问

在执行分析器窗口中：

- 找到并双击所需的脚本，然后选择 测试>测试”页面或
- 点击窗口工具栏中的  ，选择要新建脚本的包，选择 测试>测试”页面

功能区	开发>源代码>执行分析器>编辑分析器脚本 执行 > 工具 >分析器
分析器上下文菜单	测试
键盘快捷键	Shift+F12

行动

测试 类型该字段中的测试命令或脚本。例如：

- NUnit - "C:\Program Files\NUnit\bin\nunit-console.exe" "bin\debug\Calculator.exe"
- JUnit - java junit.textui.Testrunner %N

此字段中列出的命令就像来自命令提示一样执行；因此，如果可执行路径或任何参数包含空格，它们必须用引号引起来。

如果您在测试脚本中包含string %N，则在执行脚本时，它将被当前所选类的完全命名空间限定名称替换。

执行命令为 单击下拉箭头并选择适当的选项：

- 批处理文件- 使用此选项创建一个在系统命令窗口中执行的 shell 脚本；可以通过此脚本中的命令访问环境变量
- 进程-使用此选项运行单个程序 - 命令应指定程序的路径，以及任何命令行参数；如果路径或参数包含空格，请用引号将路径括起来 - 例如：“c:\program files (x86)\java\bin\javac.exe”

默认目录 默认为为编译脚本输入的值。如果尚未为编译脚本设置值，请浏览或键入清理脚本进程的默认目录运行。

解析输出 选择解析器后，可以从模型中解析、保存和管理 nUnit 和 jUnit 测试的输出；（ Alt+2 >测试 ）。请注意，仅在选择解析器时才会捕获输出。

远程主机 远程主机系统的 ID 及其端口中的类型；例如，mypc01:7777。

如果将此属性设置为#属性#，则脚本在窗口运行时发送到窗口卫星服务，在Wine下运行时发送到Linux卫星服务。服务 ID 和端口在分析器脚本编辑器的

私人选项 - 服务”部分中定义。

编译第一 选择以确保每次运行测试时都会编译包。

清理脚本

增量构建是只构建那些以某种方式发生变化的资产的做法。然而，有时有理由从头开始重新构建一切。此命令用于这些情况，以删除与特定构建或配置关联的二进制文件和中间文件。然后可以重建该项目。当您在脚本上执行 重建"菜单选项时，将执行您在此字段中指定的命令，然后立即执行来自同一分析器脚本的 编译"命令。一些编译器可以为您执行此操作。例如，Visual Studio 有 /t:clean"命令行开关。

这是一个示例脚本：清理/ 收调试MyProject.sln

访问

在执行分析器窗口中：

- 找到并双击所需的脚本，然后选择 编译>清理"页面或
- 点击窗口工具栏中的 ，选择要新建脚本的包，选择 编译>清理"页面

功能区	开发>源代码>执行分析器>编辑分析器脚本 执行>源>编译>清理
分析器上下文菜单	清理
键盘快捷键	Shift+F12

方面

方面	细节
清理	输入从脚本上下文菜单中选择 清理"时要执行的命令。
执行命令为	单击下拉箭头并选择适当的选项： • 批处理文件- 使用此选项创建一个在系统命令窗口中执行的 shell 脚本；可以通过此脚本中的命令访问环境变量 • 进程-使用此选项运行单个程序 - 命令应指定程序的路径，以及任何命令行参数； - 如果路径或参数包含空格，请用引号将它们括起来 - 例如：“c:\program files (x86)\java\bin\javac.exe”
默认目录	默认为为编译脚本输入的值。如果尚未为编译脚本设置值，请浏览或键入清理脚本进程的默认目录运行。
解析输出	单击下拉箭头并选择自动解析编译器输出的方法。 如果您选择此选项，脚本的输出将记录在系统输出窗口中； Enterprise Architect根据您指定的语法解析输出。
远程主机	远程主机系统的 ID 及其端口中的类型；例如，mypc01:7777。 如果将此属性设置为#属性#，则脚本在窗口运行时发送到窗口卫星服务，在Wine下运行时发送到Linux卫星服务。服务 ID 和端口在分析器脚本编辑器的 私人选项 - 服务"部分中定义。

编译脚本

编译”页面使您可以输入命令来构建您的项目。您可以在编写命令行时使用Enterprise Architect本地路径和环境变量。您可以选择创建自己的构建脚本，输入各种 shell 命令。您还可以选择简单地运行外部程序或批处理文件，例如 Ant 脚本。

访问

在执行分析器窗口中：

- 找到并双击所需的脚本并选择 编译>编译”页面或
- 点击窗口工具栏中的 ，选择要新建脚本的包，选择 编译>编译”页面

功能区	开发>源代码>执行分析器>编辑分析器脚本 执行 > 工具 >分析器
分析器上下文菜单	编译
键盘快捷键	Ctrl+Shift+F12

编译

使用 windows shell 命令在此文本框中编写脚本；本节的格式和内容取决于您用于构建项目的实际编译器。如果路径或参数包含空格，请用引号将它们括起来；例如：“c:\program files (x86)\java\bin\javac.exe”。

这里有些例子：

视觉工作室：

"C:\Program Files (x86)\Microsoft Visual Studio 9.0\Common7\IDE\devenv.com" /重建调试RentalSystem.sln

使用本地路径的 Visual Studio：

"%VsCompPath%\devenv.exe" /build调试Subway.sln

Java ：

C:\Program Files (x86)\ Java \jdk1.6.0_22\bin\javac.exe" -g -cp "%classpath%;." %r*.java

Java使用本地路径：

"%JAVA%\bin\javac.exe" -g -cp "%classpath%;." %r*.java

通配符Java构建 (%r) - 子文件夹中的源文件可以使用 %r 令牌构建。如示例所示，该令牌具有对所有子文件夹中的任何文件递归执行相同命令的效果。

执行命令为

单击下拉箭头并选择执行脚本的模式：

- 批处理文件-使用此选项在系统命令窗口中执行 shell 脚本；可以通过此脚本中的命令访问环境变量
- 进程-使用此选项将命令作为单个程序执行；该命令应指定程序的路径，以及任何命令行参数

默认目录

浏览或输入构建脚本进程将在其中运行的默认目录运行。

解析输出

单击下拉箭头并选择自动解析编译器输出的方法。

如果您选择此选项，脚本的输出将记录在系统输出窗口中； Enterprise Architect根据您指定的语法解析输出。

远程主机

远程主机系统的 ID 及其端口中的类型；例如，mypc01:7777。

如果将此属性设置为#属性#，则脚本在窗口运行时发送到窗口卫星服务，在Wine下运行时发送到Linux卫星服务。服务 ID 和端口在分析器脚本编辑器的 私人选项 - 服务"部分中定义。

编译后部署

选择此框可在此编译脚本完成后立即执行 Deploy 脚本。

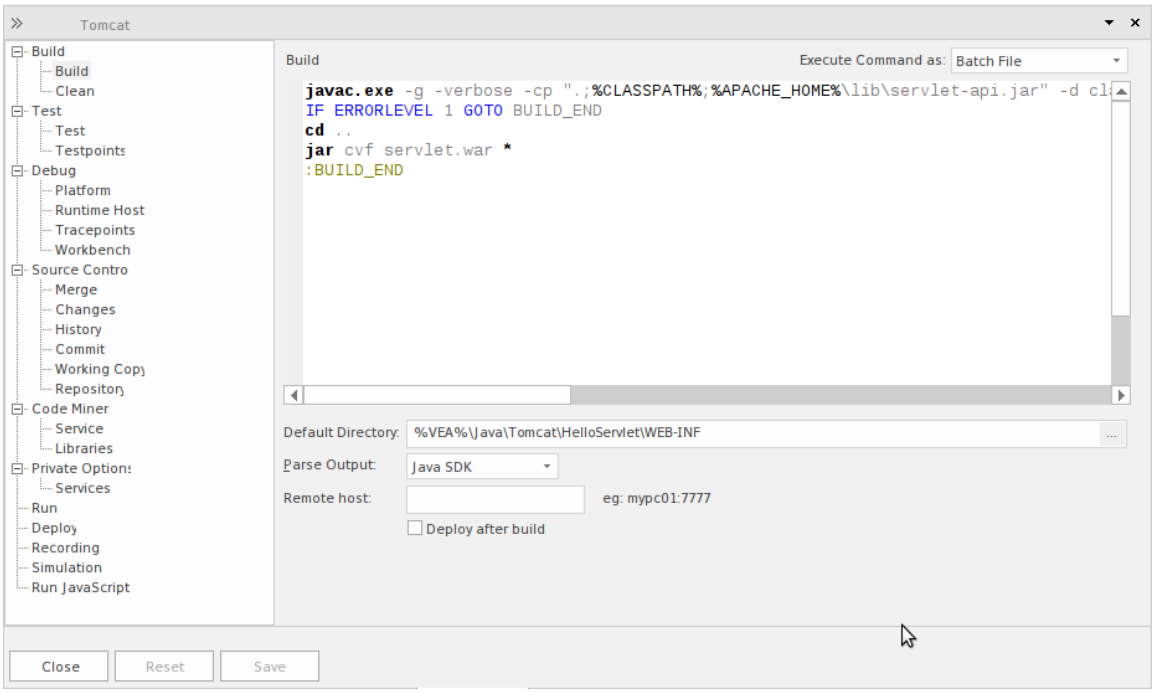
注记

要执行编译脚本，请单击浏览器窗口中的包，然后：

- 右键单击任何工具栏并选择 分析器工具栏|编译"，或
- 按 Ctrl+Shift+F12 或
- 选择 执行>源>编译>编译"功能区选项

分析器脚本

分析器脚本器有一个简单的用户界面，左侧是脚本的树形视图，可以轻松导航脚本组，右侧是内容视图，您可以在其中定义和配置脚本。



访问

在 执行分析器"窗口中，可以：

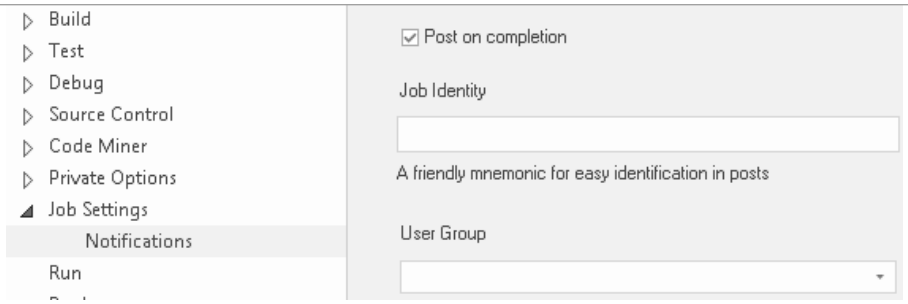
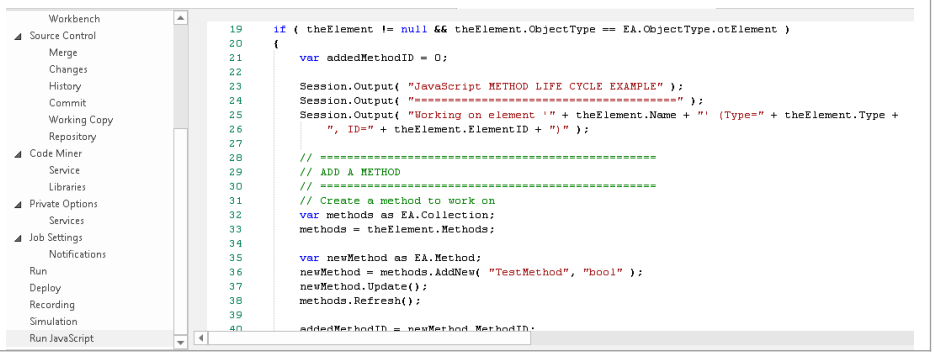
- 双击脚本进行编辑或
- 右键单击脚本并选择 编辑"选项

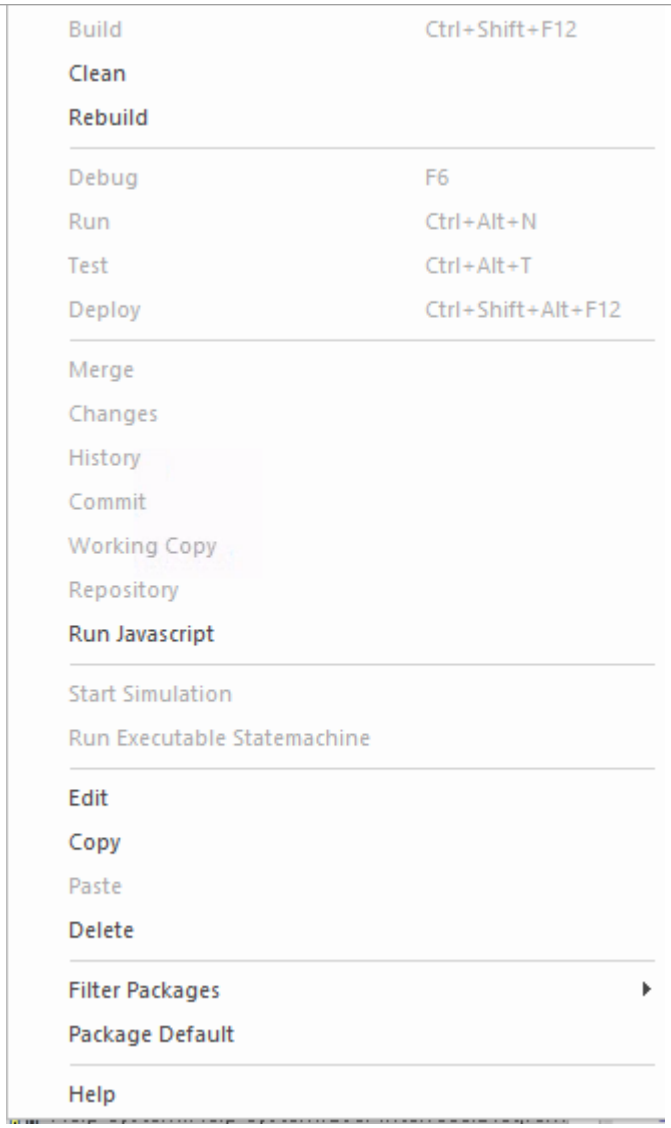
功能区	开发>源代码>执行分析器>编辑分析器脚本 执行 > 工具 >分析器
键盘快捷键	Shift+F12

执行分析器脚本

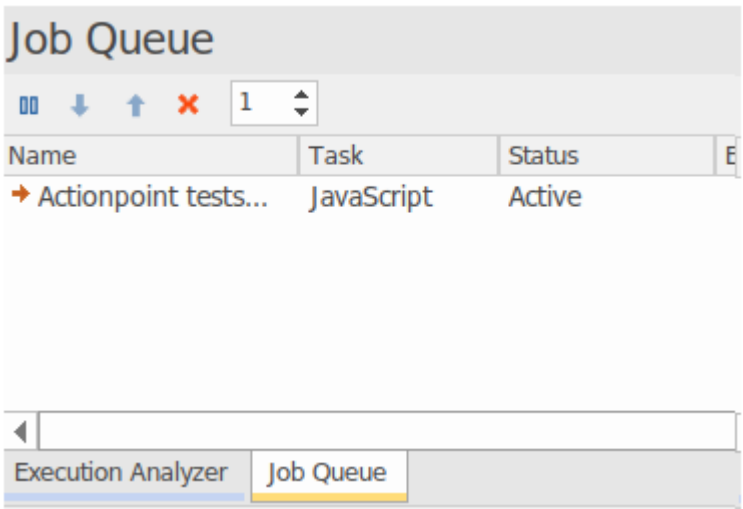
任务- Page	行动
编译-编译	输入脚本或命令来构建应用程序。这可以是 Apache Ant 或 Visual Studio 命令，但也可以根据您的开发环境进行定制。注记：请记住选择解析器以在出现任何错误时直接获取源代码。解析器字段位于同一页面上，并提供对多种语言的支持。
编译-清理	输入脚本或命令来清理以前的构建。这是您构建系统时通常发出的命令行。这可以是 Apache Ant 或 Visual Studio 命令，具体取决于您的开发环境。

测试-测试	输入脚本或命令来测试应用程序。这通常是可以配置 nUnit 或 jUnit 调用的地方，但它也可以很容易地成为任何过程或程序。
测试- 测试点	指定测试点运行的输出被发送到哪里。
调试-平台	指定调试平台、要调试的应用程序和调试模式（附加到进程或运行）。
调试- 跟踪点	指定在调试会话期间遇到的跟踪点的输出发送到哪里。
调试-工作台	对于.NET项目，要加载的程序集。 Java不需要。
调试-运行时主机	允许Enterprise Architect使用命令行启动要调试的程序。这通常用于使用套接字传输进行调试的 Mono 或Java程序。该命令在调试器运行之前运行。此命令中的端口应与传递给调试中指定的 端口"选项的值相同。当调试器启动时，它将尝试连接到此端口上的运行时。如果成功，它将绑定任何断点并恢复程序，它假定程序已挂起。 Java和 Mono 在调试传输上都有命令行选项，以在调试器连接之前暂停进程。
源控制- Merge	这是从上下文脚本时间菜单中选择 合并"选项时执行的分析器。它提供了一个运行程序或 shell 脚本来检查源文件之间差异的地方。
源控制-修改	这是从上下文脚本时间菜单中选择 修改"选项时执行的分析器。它为运行源控制程序（例如 "svn"）提供了一个地方，该程序可能会列出对源控制存储库的当前更改。
源控制-历史	这是从上下文脚本时间菜单中选择 历史"选项时执行的分析器。它为运行源控制程序（例如 "svn"）提供了一个地方，该程序可能会列出对源控制存储库的更改历史记录。
源控制- Commit	这是从上下文脚本时间菜单中选择 提交"选项时执行的分析器。它为运行源控制程序（例如 'svn'）提供了一个地方，该程序可能会提交对源控制工作副本的更改。
源控制-工作副本	这是从上下文脚本时间菜单中选择 工作副本"选项时执行的分析器。它为运行源控制程序（例如 运行"）提供了一个地方，以对源存储库的当前工作副本执行操作。
源控制-存储库	这是从上下文脚本时间菜单中选择 存储库"选项时执行的分析器。它提供了一个运行源控制程序（例如 "svn"）对源存储库执行操作的地方。
代码矿工- 服务	在本部分中，您可以选择代码矿工服务的运行方式。您可以选择远程服务器或在本地使用库。
代码矿工- 库	本节为代码矿工库的管理提供了场所。在这里，您可以基于项目代码库或存储库创建库。在这里创建的代码矿工库可以使用 mFQL 查询进行搜索。在 mFQL 中组成的查询可用于在单个操作中搜索一个或多个库。
私人期权 - 服务	这是为 Linux 和窗口配置Enterprise Architect卫星服务的 IP 地址和端口的地方。这些服务为系统管理功能和远程调试方案提供企业范围的支持。
作业设置	分析器中包含的大部分命令都是作为 Job脚本中的作业执行的。每个脚本都可以配置为在指定作业完成时向指定用户组发布通知。 作业设置"组提供 通知"选项，该选项显示您输入作业身份的字段，作为文本的一部分显示给模型邮件用户组的成员，以及用户组名称。

	 • 完成后发布 - 选中此复选框以启用其他两个字段 • Job Identity - 要显示的文本中的类型，它也应该标识工作；例如： 安装程序已完成” • 用户组 - 单击下拉箭头并选择适当的模型邮件用户组
运行JavaScript	在本节中，您可以通过从上下文的分析器菜单中选择 迈行JavaScript ”选项来创建和存储可以执行的JavaScript脚本 



选择此选项后，将在 作业队列"窗口中创建一个作业。



运行	输入命令以运行应用程序。
部署	输入脚本或命令来部署项目。编译您的 jar 文件。部署到您的设备、模拟器或

	Tomcat 服务器。发布一个网站。由你决定。
记录	你的序列图像国家电网吗？使用过滤器减少混乱。过滤器在您的代码库中定义了排除区域，可以显着减少正在记录的任何 噪音”。即使是准确的噪音也不总是有用的。
仿真	完全控件仿真配置。

管理分析器脚本

执行分析器窗口使您能够管理模型中的所有分析器脚本。您可以使用窗口工具栏按钮或脚本上下文菜单选项来控制脚本任务。脚本按包列出；该列表仅显示具有针对它们定义的分析器脚本的包。每个用户可以设置自己的活动脚本，独立于同一模型的其他用户；一个用户激活脚本不会影响其他用户当前活动的脚本或影响他们可用的脚本。活动脚本控制执行分析器的行为；例如，从菜单中选择构建命令或单击工具栏上的调试按钮时。

窗口中可能有很多包含脚本的包。为了帮助您定位和隔离特定的包，请使用 "过滤器包"上下文菜单选项，如本主题的上下文菜单选项表中所述。

注记当脚本在执行分析器窗口中高亮显示时，窗口上下文菜单选项和执行分析器窗口工具栏按钮将对高亮的脚本进行操作。但是，任何功能区或浮动工具栏或调试器（不在窗口上）中的执行分析器选项将始终使用默认分析器脚本 - 在脚本名称旁边具有选中复选框的脚本。

访问

功能区	开发>源代码>执行分析器>编辑分析器脚本 执行 > 工具 >分析器
键盘快捷键	Shift+F12

工具栏选项

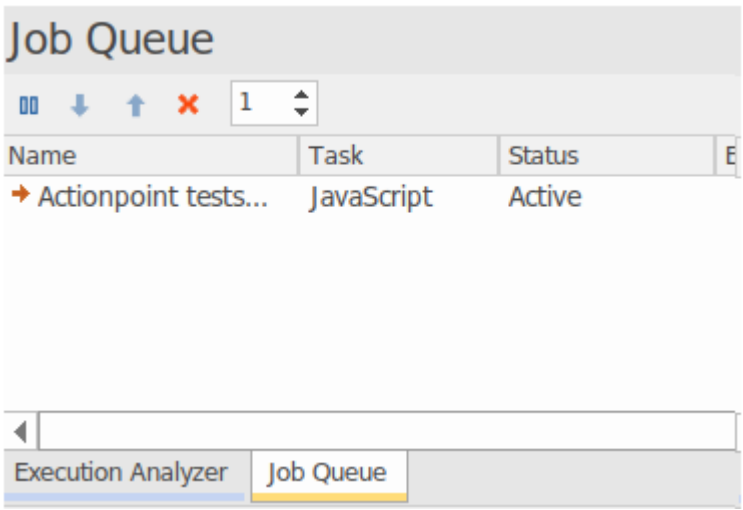
工具栏按钮	行动
	快速访问分析器核心窗口，如调用堆栈或局部变量，以及功率特征： • 剖析 • 记录 • 测试点 • 仿真
	在 Linux 或分析器下为所选包创建和编辑新的分析器脚本窗口
	导出脚本。 导出一个或多个分析器脚本导出为 XML 文件，可用于将脚本导入另一个模型。 将显示 "执行分析器：导出"对话框，您可以从中选择要导出的一个或多个脚本，然后显示目标文件名和位置的提示。
	导入脚本。 导入一个或多个分析器脚本从先前导出的 XML 文件导入当前模型。 将显示 "查找包"对话框，您可以在其中选择要导入脚本的包，然后显示源文件名和位置的提示。
	执行活动脚本的 "编译"命令。

	取消当前正在进行的 编译”命令。
	执行活动脚本的 运行”命令。
	执行活动脚本的 测试”命令。
	执行活动脚本的 部署”命令。

上下文菜单选项

右键单击所需的脚本或包以显示上下文菜单。

选项	行动
加新脚本	将新脚本添加到选定的包。 执行分析器窗口显示，显示 编译”页面。
粘贴脚本	粘贴将Enterprise Architect剪贴板中的脚本复制到选定的包中。 您可以多次粘贴复制的脚本；每个副本都有后缀 副本”。 要重命名复制的脚本，请按 F2 并改写脚本名称。
导出脚本	从所选包中导出脚本。 将显示 执行分析器：导出”对话框，从中选择要导出的一个或多个脚本，然后显示目标文件名和位置的提示。
导入脚本	将导入文件中的脚本导入到选定的包中。 A显示源文件名和位置的提示。
过滤器包	显示选项子菜单以： • 输入要过滤包列表的包路径 - 当您选择 过滤器包”选项时，将显示一个提示以接受当前选择的包路径，或者您可以删除路径的结尾元素以指定更大的套包；当您单击确定按钮时，列表中的第一个包被展开以列出它包含的脚本 • 在显示包的完整列表和隐藏包的完整列表之间切换以仅显示当前选择的包 • 删除当前活动的过滤器以显示包的完整列表
在项目中选择浏览器	在浏览器窗口中突出显示选定的包。 显示浏览器窗口，该窗口现在展开以显示突出显示的包。
编译	执行所选脚本的 编译”命令。
清理	执行选中脚本的 清理”命令
重建	执行所选脚本的 清理”和 编译”命令。
调试	执行所选脚本的 调试”命令。

运行	执行所选脚本的 运行”命令。
测试	执行所选脚本的 测试”命令。
部署	执行所选脚本的 部署”命令。
合并	执行所选脚本的'Merge'源控制命令。
修改	执行选中脚本的 修改”源控制命令。
历史	执行所选脚本的'History'源控制命令。
犯罪	执行所选脚本的'Commit'源控制命令。
工作副本	执行所选脚本的'Working Copy'源控制命令。
存储库	执行所选脚本的 存储库”源控制命令。
运行JavaScript	执行所选脚本的 运行JavaScript ”命令。选择此选项 ，将在 作业队列”窗口中创建一个作业。 
开始仿真	开始 分析器脚本”页面仿真引用的模拟。
运行可执行状态机	选定的可执行状态机开始工件。
编辑	在 分析器脚本编辑器”中打开选定的脚本。
复制	将选定的脚本复制到Enterprise Architect剪贴板。
粘贴	粘贴最近复制的脚本到与所选脚本相同的包中。 您可以多次粘贴复制的脚本；每个副本都有后缀 副本”。 要重命名复制的脚本，请按 F2 并改写脚本名称。
删除	删除选定的脚本；没有提示确认。 要从执行分析器窗口中删除一个包，请从该包中删除脚本。删除最后一个脚本时，不再列出该包。

包默认	将所选脚本设置为包的默认脚本。 脚本左侧的图标改变颜色；任何以前的包默认恢复正常。