

目录

INSTRUMENTS用户指南	I
INSTRUMENTS用户指南介绍	2
本文档组织结构.....	2
第一章 INSTRUMENTS快速入门	4
1.1 启动INSTRUMENTS.....	5
1.2 创建一个跟踪文档.....	5
1.3 浏览跟踪文档窗口.....	7
1.4 示例：快速使用一个跟踪.....	10
1.5 下一步是什么？.....	10
第二章 添加和配置INSTRUMENTS工具	11
2.1 使用INSTRUMENT库.....	11
2.1.1 修改库试图模式.....	12
2.1.2 查找库里面的某个instrument工具.....	13
2.1.3 新建一个自定义的instrument分组.....	15
2.2 添加和删除INSTRUMENTS工具.....	20
2.3 配置一个INSTRUMENT工具.....	21
第三章 记录跟踪数据	23
3.1 选择需要跟踪的进程.....	23
3.1.1 跟踪所有进程.....	23
3.1.2 跟踪一个已有的进程.....	24
3.1.3 跟踪一个新的进程.....	24
3.1.4 给每个Instrument工具指定不同的目标.....	26
3.2 收集数据.....	26
3.3 使用快速启动键启动INSTRUMENTS.....	27
3.4 以最小模式运行.....	28
3.5 从XCODE运行INSTRUMENTS应用.....	29
3.6 无线连接iOS设备.....	29
第四章 记录用户界面轨迹	32
4.1 记录用户界面轨迹.....	32
4.2 重复记录用户界面轨迹.....	33
4.3 回放用户界面轨迹.....	34

第五章	查看和分析跟踪数据	36
5.1	查看数据的工具	36
5.1.1	跟踪面板	36
5.1.2	详细面板	41
5.1.3	扩展详细面板	44
5.1.4	运行浏览器	46
5.2	分析技术	47
5.2.1	使用Sampler Instrument分析数据	47
5.2.2	使用Allocations Instrument工具分析数据	49
5.2.3	查找内存泄露	53
5.2.4	分析Core Data应用程序	54
第六章	保存和导入跟踪数据	55
6.1	保存跟踪文档	55
6.2	导出跟踪数据	55
6.3	从SAMPLE工具中导入数据	56
6.4	使用DTRACE数据	56
第七章	使用DTRACE创建自定义INSTRUMENTS工具	57
7.1	关于自定义INSTRUMENTS工具	57
7.2	创建自定义的INSTRUMENT工具	58
7.2.1	添加和删除探针	60
7.2.2	指定探针的提供者	60
7.2.3	给探针添加断言	61
7.2.4	给探针添加动作	63
7.2.5	编写自定义脚本的提示	65
7.2.6	编写BEGIN和END脚本	65
7.2.7	从自定义脚本里面访问内核数据	66
7.2.8	变量作用域	67
7.2.9	查找脚本错误	67
7.3	导出DTRACE脚本	68
第八章	内置INSTRUMENTS工具	69
8.1	CORE DATA INSTRUMENTS[CORE DATA相关]	69
8.1.1	Core Data Saves	69
8.1.2	Core Data Fetches	70
8.1.3	Core Data Faults	70
8.1.4	Core Data Cache Misses	71
8.2	DISPATCH INSTRUMENTS[并发相关]	72

8.2.1	Dispatch.....	73
8.3	ENERGY DIAGNOSTICS INSTRUMENTS[电池诊断相关]	75
8.3.1	电量使用（Energy Usage）	76
8.3.2	CPU 活动（CPU Activity）	76
8.3.3	显示亮度（Display Brightness）	77
8.3.4	休眠/唤醒（Sleep/Wake）	77
8.3.5	蓝牙（Bluetooth）	77
8.3.6	无线（WiFi）	77
8.3.7	定位（GPS）	78
8.4	FILE SYSTEM INSTRUMENTS[文件系统相关]	78
8.4.1	I/O 活动(I/O Activity).....	78
8.4.2	文件锁(File Locks)	81
8.4.3	文件属性(File Attributes)	81
8.4.4	文件活动（File Activity）	82
8.4.5	目录I/O（Directory I/O）	83
8.5	GARBAGE COLLECTION INSTRUMENTS[垃圾回收相关].....	84
8.5.1	GC Total	84
8.5.2	垃圾回收（Garbage Collection）	85
8.6	GRAPHICS INSTRUMENTS[绘图相关].....	86
8.6.1	核心动画（Core Animation）	86
8.6.2	OpenGL驱动器（OpenGL Driver）	87
8.6.3	OpenGL ES驱动器（OpenGL ES Driver）	87
8.6.4	OpenGL ES分析器（OpenGL ES Analyzer）	89
8.7	INPUT/OUTPUT INSTRUMENTS[输入输出相关]	90
8.7.1	读/写（Reads/Writes）	90
8.8	MASTER TRACKS INSTRUMENTS[界面操作跟踪相关].....	91
8.8.1	用户界面（User Interface）	91
8.9	MEMORY INSTRUMENTS[内存相关]	91
8.9.1	共享内存（Shared Memory）	91
8.9.2	分配内存（Allocations）	92
8.9.3	内存泄露（Leaks）	94
8.10	SYSTEM INSTRUMENTS[系统相关]	95
8.10.1	时间分析器（Time Profiler）	95
8.10.2	旋转监控器（Spin Monitor）	96
8.10.3	取样（Sampler）	97
8.10.4	进程（Process）	98
8.10.5	网络活动监控器（Network Activity Monitor）	99
8.10.6	内存监控器（Memory Monitor）	99

8.10.7	硬盘监控器（Disk Monitor）	99
8.10.8	CPU监控器（CPU Monitor）	100
8.10.9	活动监控器（Activity Monitor）	100
8.11	THREADS/LOCKS INSTRUMENTS[线程相关]	100
8.11.1	Java线程（Java Thread）	100
8.12	UI AUTOMATION[界面自动化相关]	101
8.12.1	使用Automation Instrument工具	101
8.12.2	访问和操作用户界面元素	104
8.12.3	添加灵活的超时间	113
8.12.4	验证测试结果	114
8.12.5	输出测试结果和数据的日子	114
8.12.6	处理警告	115
8.12.7	检测和指定设备的方向	116
8.12.8	测试多任务	118
8.13	USER INTERFACE INSTRUMENTS[用户界面相关]	118
8.13.1	Cocoa事件（Cocoa Events）	118
8.13.2	Carbon事件（Carbon Events）	119
结束语		120
推荐资源		121

Instruments用户指南介绍

Instruments 是应用程序用来动态跟踪和分析 Mac OS X 和 iOS 代码的实用工具。这是一个灵活而强大的工具，它让你可以跟踪一个或多个进程，并检查收集的数据。这样，Instruments 可以帮你更好的理解应用程序和操作系统的行为。

使用 Instruments 应用，你可以使用特殊的工具(即 instruments 工具)来跟踪同一进程不同方面的行为。你也可以使用该应用来记录一系列用户界面的动作并响应它们，同时也可以使用一个或多个 instruments 工具来收集数据。

Instruments 应用包含以下功能：

- 分析一个或多个进程的行为
- 记录一系列用户的动作并响应它们，可靠的再现这些事件并收集多次运行的数据
- 创建你自己自定义的 DTrace instruments 来分析系统和应用程序的行为
- 保存用户界面记录和 instruments 的配置为模板，并从 Xcode 里面访问

使用 Instruments，你可以：

- 追查代码中难以重现的问题
- 对你的程序进行性能分析
- 自动化测试你的代码
- 对你程序进行压力测试
- 进行一般的系统级故障诊断
- 对你的代码如何工作有更深入的了解

Instruments 在 Xcode 3.0 和 Mac OS X 10.5 及其之后可用。

本文档描述了 Instruments 的用户界面，给出了一个如何使用 Instruments 来跟踪进程和查看数据的预留。目的是让开发人员和系统管理员使用 Instruments 能更好的了解他们程序或系统作为一个整体的行为。

本文档组织结构

以下章节描述了如何使用 Instruments 应用：

- “[Instruments 快速入门](#)” 给出了 Instruments 的概要预览，并介绍了主体窗

口。

- “[添加和配置Instruments](#)” 描述了如何添加和配置instruments工具, 以及在一个或多个进程里面运行它们收集数据。本章还介绍如何对程序进行选择跟踪。
- “[记录跟踪数据](#)” 描述了如何初始化跟踪并收集跟踪数据的方法。
- “[记录用户界面轨迹](#)” 描述如何记录和重放一系列有顺的用户操作。
- “[查看和分析跟踪数据](#)” 描述了用来查看Instruments返回数据的工具。
- “[保存和导入跟踪数据](#)” 描述了如何保存跟踪文档和数据, 以及如何从其他来源导入数据。
- “[使用DTrace创建自定义工具](#)” 显示了如何创建和配置基于DTrace的自定义工具。
- “[内置的instruments工具](#)” 详细介绍了Instruments内置的工具。

第一章 Instruments快速入门

Instruments 是一个很强大的工具，你可以用它来收集关于一个或多个系统进程的性能和行为的数据，并跟踪随着时间产生的数据。不像其他大部分性能和调试工具那样，Instruments 让你可以广泛收集不同类型的数据，并且可以一边查看它们。这样你可以发现变化趋势，这在其他工具里面是很难做到的。比如，你之前需要采样程序的样本，并分析它们在两个独立的执行文件里面运行的内存行为。在 Instruments 里面，你可以同时完成这些工作。你可以用这些结果数据来发现你代码中正在运行部分的变化趋势和它们的内存使用情况。

Instruments 应用使用 **instruments** 工具来收集关于进程随时间推移产生的数据。每个 **instruments** 收集和显示不同类型的数据，比如文件访问、内存使用等等。Instruments 包括一个标准 **instruments** 工具库，你可以使用它分析你代码的很多方面。你可以配置 **instruments** 来收集关于同一个或者不同系统进程的数据。你可以使用自定义的 **instruments** 工具新建接口来创建新的自定义 **instruments** 工具，它使用 DTrace 程序来收集你想要的数据库。

注意:不少应用程序 (*iTunes*、*DVD Player* 和 *Front Row*，还有使用 *QuickTime* 的应用) 为了保护敏感数据，不允许使用 DTrace 来收集数据 (无论暂时的还是永久的)。

所有 Instruments 的工作都在一个跟踪文档 (trace documents) 里面完成。一个跟踪文档收集那些被 **instruments** 聚集的与该文档有关的数据。每个跟踪文档通常包含一个会话的价值数据，这也是作为一个单一的跟踪。你可以保存跟踪文档到你已经收集的跟踪数据备份里面，然后可以在以后再次打开并查看它们。

尽管大部分的 **instruments** 工具旨在收集数据，但是其中最精密的 **instruments** 工具可以帮助自动化收集数据。使用 Instruments User Interface 工具，你可以在收集数据的过程中记录用户事件。你可以使用这些数据来可靠的重复重现这一系列有序的事件。每次你通过这序列运行，你的跟踪文档在其他 **instruments** 工具里面收集新的跟踪数据，并且和之前一样边收集边显示这些数据。这些特性让你比较跟踪的数据来提高你的代码，并验证你的改动获得预期的结果。

1.1 启动 Instruments

Instruments 作为 Xcode 工具安装的一部分。有几个启动 Instruments 的方法：

- 你可以在 Finder 里面双击 Instruments 应用的图标来启动它。Instruments 应用位于 `<Xcode>/Applications` 目录下面，`<Xcode>`代表你 Xcode 安装的根路径。（Xcode 默认的安装根路径时 `/Developer` 目录。）
- 你可以通过 Xcode 来启动 Instruments，并把你工程下的可执行文件作为它的目标。在 Xcode 的菜单里面选择 `Run > Start with Performance Tool` 来选择 Instruments 模板（注：在 Xcode 4.0 以上已经无此方法，你需要通过 Profile 来启动，更多请 Google 一下）。Instruments 以特定的模板启动，并且准备好运行 Xcode 配置的当前可执行文件。

注意：当你以这种方式来启动一个 Instruments 模板的时候，在完成跟踪之后，你必须显示的停止你当前的执行文件（无论是从程序里面还是从 Instruments 里面）。这不会关闭 Instruments 的跟踪文档。相反，你可以重启可执行文件，Instruments 显示旧的对话框旁边显示新的对话框，就如图 4-2 显示那样。这样可以让你在 Xcode 里面修改你的代码，重新编译，运行和比较新的改变产生的跟踪数据。

- 你可以通过双击一个 Instruments 的模板文件或者跟踪文档来启动它。查看”保存和导入跟踪数据”部分来掌握如何保存一个 Instruments 的配置文件或者使用用户界面记录作为一个模板文件。

1.2 创建一个跟踪文档

当你启动 Instruments 后，应用会自动为你创建一个文档。你同样可以通过选择 `File > New` 来创建一个新的文档。

你每创建一个新的文档，Instruments 都会提示你选择一个开始模板。这些模板定义了一些你将要在你的跟踪文档里面使用的 instruments 工具集。Instruments 提供了几种不同的模板，在表 1-1 里面列举出来，每个模板都有不同的使用目的。如果你手工的给你的跟踪文档添加一个指定的 instruments 工具的话，你可以使用空白模板。

Table 1-1 Instruments starting templates

Template	Type	Description
Blank	Mac OS X, iOS, Simulator, User	Creates an empty trace document to which you can add your own combination of instruments. To learn how to select and add instruments, see “Adding and Configuring Instruments.” For descriptions of the individual built-in instruments, see “Built-in Instruments.”
Activity Monitor	Mac OS X, iOS, Simulator	Adds the Activity Monitor instrument to your document. Use this template if you want to correlate the system workload with the virtual memory size.
Allocations	Mac OS X, iOS, Simulator	Adds the Allocations and VM Tracker instruments to your document. Use this template to monitor memory and object-allocation patterns in your program. (To use this template, you must launch your process from Instruments.)
Automation	iOS, Simulator	Adds the Automation instrument to your document. Use this template to automate user interface tests of your iOS application.
Core Animation	iOS	Adds the Core Animation and Sampler instruments to your document. Use this template to measure the number of Core Animation frames per second in a process running on an iOS device, and to see visual hints that help you understand how content is rendered on the screen.
Core Data	Mac OS X	Adds the Core Data Fetches, Core Data Cache Misses, and Core Data Saves instruments to your document. Use this template to monitor data store interactions in a Core Data application.
CPU Sampler	Mac OS X, iOS, Simulator	Adds the Sampler and CPU Monitor instruments to your document. Use this template if you want to correlate the overall system workload with the work being done specifically by your application.
Dispatch	Mac OS X	Adds the Dispatch instrument to your document. Use this template if you want to capture information about GCD queues created by your application and about the block objects executing on these queues.
Energy Diagnostics	iOS	Adds the Energy Diagnostics, CPU Activity, Display Brightness, Sleep/Wake, Bluetooth, WiFi, and GPS instruments to your document. Use this template to get diagnostic information regarding energy usage in iOS devices.
File Activity	Mac OS X, iOS Simulator	Adds the File Activity, Reads/Writes, File Attributes, and Directory I/O instruments to your document. Use this template if you want to examine file usage patterns in the system. This combination of instruments monitors open, close, read, and write operations on files and also monitors changes in the file system itself, including permission and owner changes.
GC Monitor	Mac OS X	Adds the ObjectGraph, Allocations, and Garbage Collection instruments to your document. Use this template to measure the data reclaimed in the scavenge phase of the garbage collector.
Leaks	Mac OS X, iOS, Simulator	Adds the Allocations and Leaks instruments to your document. Use this template to monitor memory usage in your application and to detect memory leaks. (To use this template, you must launch your process from Instruments.)
Multicore	Mac OS X	Adds the Thread States and Dispatch instruments to your document. Use this template to analyze multicore performance, including thread state, dispatch queues, and block usage.
OpenGL ES Analysis	iOS	Adds the OpenGL ES Analyzer and OpenGL ES Driver instruments to your document. Use this template to measure OpenGL ES activity and get

		recommendations for addressing problems.
OpenGL ES Driver	iOS	Adds the OpenGL ES Driver and Sampler instruments to your document. Use this template to determine how efficiently you're using OpenGL and the GPU on iOS devices.
Sudden Termination	Mac OS X	Adds the Sudden Termination and Activity Monitor instruments to your document. Use this template to analyze sudden termination support. It reports unprotected file-system access the target process should be, but is not, guarding with calls to disable sudden termination. It also provides activity monitoring across all processes, including sudden termination status for each.
System Usage	iOS	Adds the I/O Activity instrument to your document. Use this template to record calls to functions that operate on files in a process running on an iOS device.
Threads	Mac OS X, iOS Simulator	Adds the Thread States instrument to your document. Use this template to analyze thread state transitions within a process, including running and terminated threads, thread state, and associated backtraces.
Time Profiler	Mac OS X, iOS, iOS Simulator	Adds the Time Profiler instrument to your document. Use this template to perform low-overhead time-based sampling of one or all processes.
UI Recorder	Mac OS X	Adds the User Interface instrument to your document. Use this template as a starting point for recording a series of user interactions with your application. You can use this feature to reproduce a series of events multiple times, gathering a new set of data during each successive run. You can then compare the sets of data knowing that the behavior that generated them was identical. Typically, you would start with this template and add additional instruments to your document to gather data.
Zombies	Mac OS X, iOS Simulator	Adds the Allocations instrument to your document. Use this template to measure general memory usage while focusing on the detection of over-released "zombie" objects.

如果你不想让 Instruments 在你新建一个文档的时候询问你使用那个模板，你可以在 Instruments 的偏好设置里面勾选 Suppress template chooser 选项来禁止模板选择。关于更多在 Instruments 文档里面添加和配置 instruments 工具的信息，参阅“添加和配置 Instruments”部分。

1.3 浏览跟踪文档窗口

跟踪文档自己包含一个收集和分析数据的空间。你使用这些文档来组织和配置你需要用来收集数据的 instruments 工具，并且可以使用这些文档来查看你已经收集的高级和低级数据。

图 1-1 显示了典型的跟踪文档。一个跟踪文档窗口显示很多信息，所以需要很好的组织。正如你使用跟踪文档来工作一样，信息流从左到右。在你的文档窗口里面，

越右边的窗口数据越详细。表 1-2 提供了该窗口关键区域的介绍。

Figure 1-1 The Instruments window while tracing

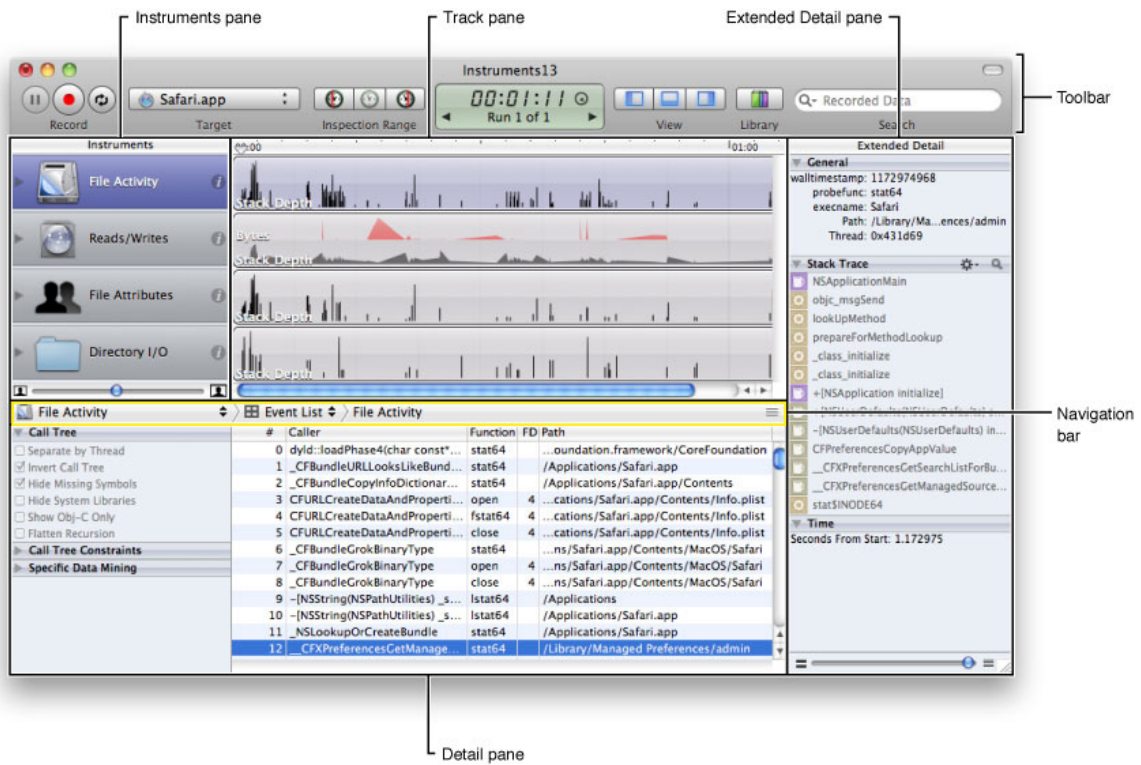


表 1-2 列出了图 1-1 中显示的关键特性，并提供了一个更深入使用这些特性的讨论。

Table 1-2 Trace document key features

Feature	Description
Instruments pane	This pane holds the instruments you want to run. You can drag instruments into this pane or delete them. You can click the inspector button in an instrument to configure its data display and gathering parameters. To learn more about instruments, see Adding and Configuring Instruments.
Track pane	The track pane displays a graphical summary of the data returned by the current instruments. Each instrument has its own “track,” which provides a chart of the data collected by that instrument. The information in this pane is read-only. You do use this pane to select specific data points you want to examine more closely, however. The track pane is described in more detail in “The Track Pane.”
Detail pane	The Detail pane shows the details of the data collected by each instrument. Typically, this pane displays the explicit set of “events” that were gathered and used to create the graphical view in the track pane. If the current instrument allows you to customize the way detailed data is displayed, those options are also listed in this pane. For more information about this pane, see “The Detail Pane.”
Extended Detail pane	The Extended Detail pane shows even more detailed information about the item currently selected in the Detail pane. Most commonly, this pane displays the complete stack trace, timestamp, and other instrument-specific data gathered for the given event. The Extended Detail pane is described in “The Extended Detail Pane.”
Navigation	The navigation bar shows you where you are and how you got there. It includes two

bar	menus—the active instrument menu and the detail view menu. You can click entries in the navigation bar to select the active instrument and the level and type of information in the detail view.
-----	--

跟踪文档的工具栏（toolbar）可以让你增加和控制 instruments 工具，打开视图，并配置跟踪面板。图 1-2 标示了这些工具栏中不同的控件，在表 1-3 里面提供了如何使用这些控件的详细说明。

Figure 1-2 The Instruments toolbar

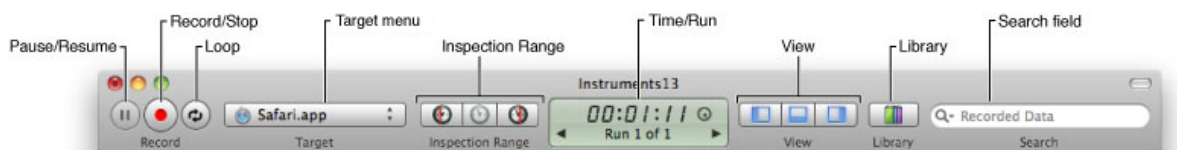


Table 1-3 Trace document toolbar controls

Control	Description
Pause/Resume button	Pauses the gathering of data during a recording. This button does not actually stop recording, it simply stops Instruments from gathering data while a recording is under way. In the track pane, pauses show up as a gap in the trace data.
Record/Stop button	Starts and stops the recording process. You use this button to begin gathering trace data. For more information, see “Collecting Data.”
Loop button	Sets whether the user interface recorder should loop during play back to repeat the recorded steps continuously. Use this to gather multiple runs of a given set of steps. For information about playing tracks, see “Replaying a User Interface Track.”
Target menu	Selects the trace target for the document. The trace target is the process (or processes) for which data is gathered. For more information on choosing a trace target, see “Choosing Which Process to Trace.”
Inspection Range control	Selects a time range in the track pane. When set, Instruments displays only data collected within the specified time period. Use the buttons of this control to set the start and end points of the inspection range and to clear the inspection range. For more information, see “Viewing Data for a Range of Time.”
Time/Run control	Shows the elapsed time of the current trace. If your trace document has multiple data runs associated with it, you can use the arrow controls to choose the run whose data you want to display in the track pane. For information about trace runs, see “Viewing Trace Runs.”
View control	Hides or shows the Instruments pane, Detail pane, and Extended Detail pane. This control makes it easier to focus on the area of interest.
Library button	Hides or shows the instrument library. For information on using the Library window, see “Using the Instrument Library.”
Search field	Filters information in the Detail pane based on a search term that you provide. Use the search field’s menu to select search options. For more information, see “Searching in the Detail Pane.”

1.4 示例：快速使用一个跟踪

为了记录跟踪的数据，你需要指定需要收集数据的目标，然后点击 Record 按钮。大部分 instruments 工具允许你指定一个系统进程作为目标，而另外一些 instruments 工具允许你收集多个进程的信息。

以下步骤描述了如何新建一个跟踪文档，配置它，并记录一些数据。Instruments 应用不能在运行之前执行这些步骤。

1. 启动 Instruments。应用会自动新建一个跟踪文档，并提示你选择一个模板。
2. 选择 Activity Monitor 模板，然后单击选择按钮。Instruments 会把 Activity Monitor instruments 工具添加到跟踪文档里面。
3. 在跟踪文档的默认目标菜单里面，选择 All Process。
4. 点击 Record 按钮。
5. 等待几秒以便 Instruments 来收集一些数据。
6. 点击 Stop 按钮。

恭喜你，你已经使用 Instruments 来收集了一些跟踪数据。Instruments 在跟踪面板（track pane）里面显示了几个和系统加载、虚拟内存大小相关的图形。在详细面板（detail pane）显示了在数据收集期间运行的进程列表。你可以在详细面板里面选择不同的视图模型来查看数据以不同方式组织的形式。

1.5 下一步是什么？

现在已经介绍了和 Instruments 相关的基本概念，你可以开始更详细的探索 Instruments 应用了。剩下的章节提供了一个 Instruments 特性的深入覆盖，包括如何添加和配置 instruments 工具，如何记录用户界面轨迹，如何分析你收集的数据，和如何把你收集的数据保存为以后使用。

记住程序分析是一门艺术也是门科学。在科学方面，也有一些可以遵循的指南来发现问题。比如，如果你的应用程序占用内存过大，那么有可能是应用程序在某些地方分页，导致性能表现不佳。艺术方面，因为每个应用程序如何减少内存占用的方案是不同的。是否应用程序收集的数据过多？是否它加载太多而又没有使用的库？是否内存泄露？这些都是你需要问你自己的问题，而 Instruments 只是一个你可以用来早点答案的工具。

第二章 添加和配置Instruments工具

Instruments 应用使用 instruments 工具来收集数据，并显示数据给用户。尽管目前理论上没有限制你在一个文档里面包含的 instruments 工具的个数，但是大部分文档出于性能原因一般包含少于 10 个工具。你甚至可以多次包含同一个 instrument 工具，配置每个 instrument 工具来收集来自不同系统进程的数据。

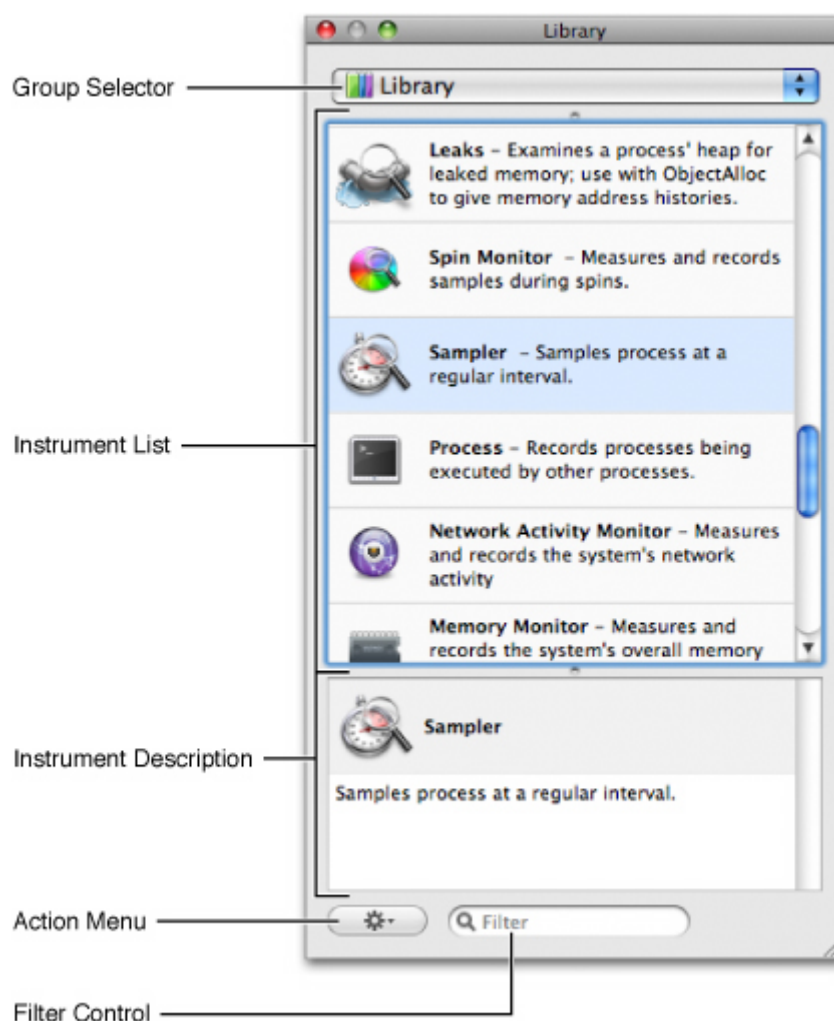
Instruments 应用内置了广泛的 instruments 工具，可以使用它们来收集一个或多个进程的特定数据。大部分这些 instruments 工具都需要少量甚至不需要任何的配置就可以使用。你只要简单的把它们添加到你的跟踪文档，即可开始收集跟踪数据。然而你也可以创建自定义的 instruments 工具，它们给你提供了广泛选择来收集数据。

本章重点是介绍如何添加已有的 instruments 工具到你的跟踪文档里面，并配置使用它们。关于如何创建自定义 instruments 工具的更多信息，参阅“使用 DTrace 创建自定义的 Instruments 工具”部分。

2.1 使用Instrument库

Instrument 库(如图 2-1 显示)显示了所有 instruments 工具，你可以把它们添加到你的跟踪文档里面。该库包含了所有内置的 instruments 工具和你已经定义好的自定义 instruments 工具。为了打开库的窗口，点击你的跟踪文档下面的 Library 按钮，或者选择 Window > Library。

Figure 2-1 The instrument library



因为库窗口里面的 instruments 工具太多，尤其是在你加入了你的自定义 instruments 工具之后，所以 Instruments 库窗口提供了几个组织和查找 instruments 工具的办法。以下部分就是讨论这些方法，并显示如何使用它们来组织可用的 instruments 工具。

2.1.1 修改库试图模式

库提供了不同的视图模式来帮你组织可用的 instruments 工具。视图模式允许你选择你要在每个 instrument 工具显示的信息数量，和你想要 instrument 工具占据的空间。Instruments 应用支持以下视图模式：

- **View Icons（查看图标）**. 只显示每个 Instrument 工具代表的图标。
- **View Icons 和 Labels（查看图标和标签）**. 显示每个 instrument 工具的图标和名称。

- **View Icons 和 Descriptions (查看图标和描述)**. 显示每个 instrument 工具的图标、名称和详细描述。
- **View Small Icons 和 Labels (查看小图标和标签)**. 显示每个 instrument 工具的名称和它小版本的图标。

注意: 不管你选择了那种视图模式, 库窗口总是在详细面板部分显示选择了的 *instrument* 工具的详细信息。

为了切换库的视图模式, 你需要选择库窗口底部的 Action Menu 的所需模式。

除了切换库的视图模式, 你也可以通过选择 Action Menu 里面的 Show Group Banners 来显示一组 instruments 工具的父组。库窗口为了方便以分组的方式组织 instruments 工具, 帮助你缩小查找想要 instrument 工具的范围。默认情况下, 组信息没有显示在库窗口的主页上面。要以分组显示则需要添加 Show Group Banners 的选项, 让它更容易识别 instruments 工具在某些视图里面的一般行为。

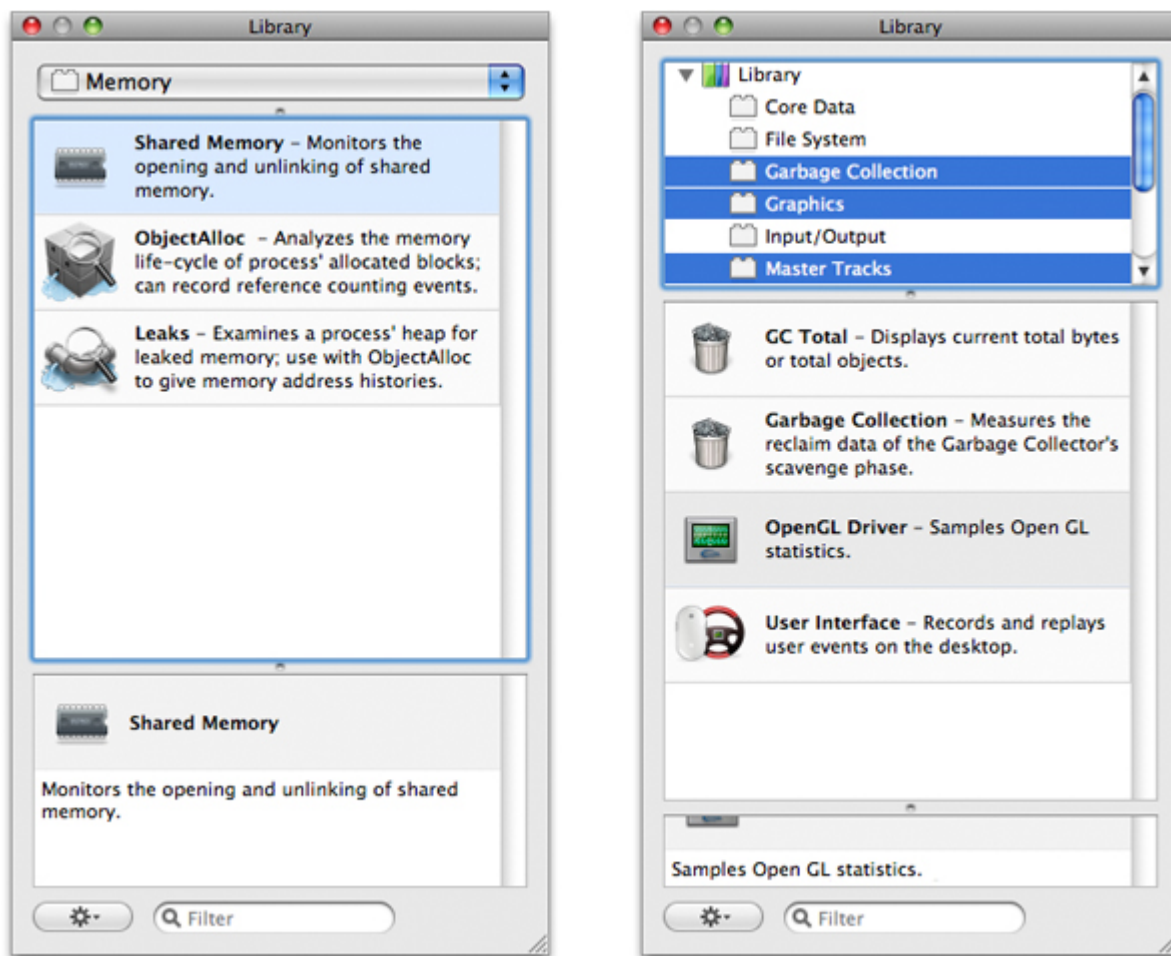
2.1.2 查找库里面的某个instrument工具

默认情况下, 库窗口显示全部可用的 instruments 工具。然而每个 instrument 工具属于一个大的分组, 它标示了一个 Instrument 工具的目的和它收集数据的类型。在库窗口的顶部, 你可以使用组选项控制器来选择一个或多个分组来限制库窗口显示 instruments 工具的数量。当库里面包含很多 instruments 工具时, 该方法可以让你更好的找到你想要的 instrument 工具。

组选项控制器包含两个不同的配置。其中一个配置是库窗口显示一个弹出菜单, 你可以从中选择单一分组。然而如果你拖动一个分割栏位于弹出菜单和 instrument 面板下面, 那么弹出菜单会变为一个大纲视图。在此配置下, 你可以通过按下 Command 键和 Shift 键来选择多个你想要的分组。

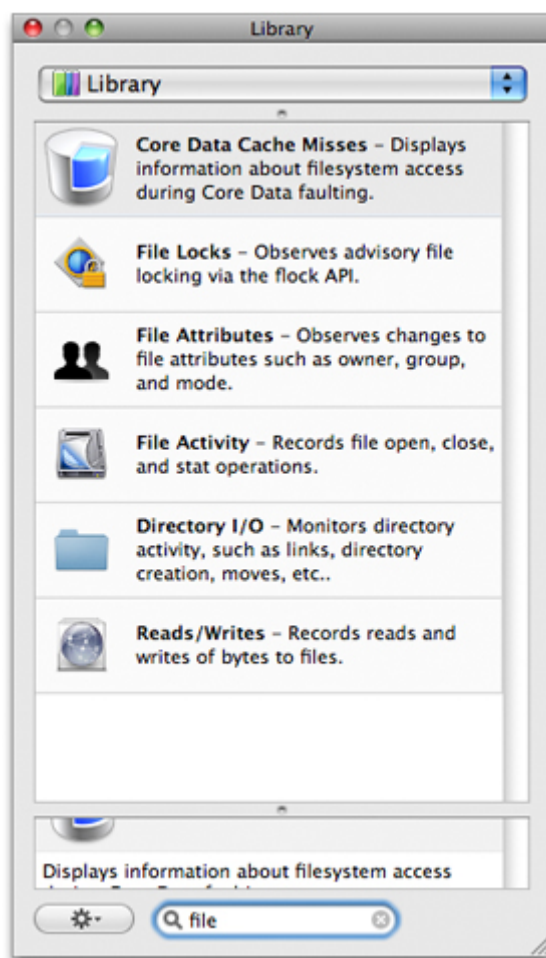
图 2-2 显示了库窗口的标准模式和大纲模式。左边的是标准模式, 你可以通过使用弹出菜单来选择一个分组。窗口右边显示的是大纲视图, 你可以选择多个分组和管理你自定义分组。

Figure 2-2 Viewing an instrument group



另外一个在库窗口过滤内容的办法是使用窗口底部的搜索区域。使用搜索，你可以快速的缩小需要查找的 instrument 工具的范围，你可以搜索它的名称、描述、分类，结果是一列匹配关键字的工具。比如图 2-3 显示了匹配 file 关键字的 instruments 工具。

Figure 2-3 Searching for an instrument



当库窗口是弹出框显示的时候，搜索过滤器的内容基于当前所选择的 instrument 组。如果是大纲视图显示时，不论那个分组被选中，搜索过滤器的内容是基于整个库的 instruments。

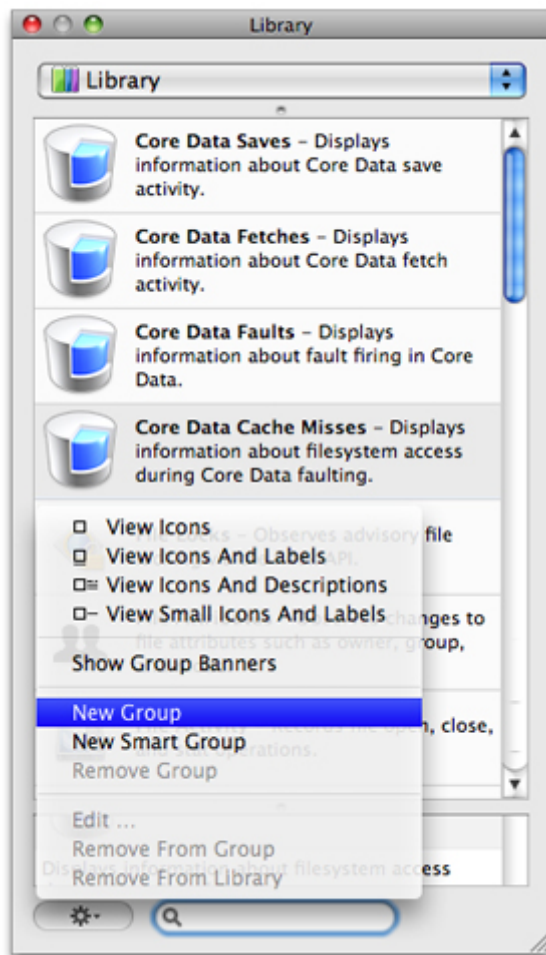
2.1.3 新建一个自定义的instrument分组

除了可以使用内置的 instrument 分组，你还可以创建自定义分组来按照自己的方式组织 instruments 工具。Instruments 支持两种自定义分组：静态分组(static groups)和智能分组(smart groups)。静态分组就是简单的静态。Instruments 不会改变它们的内容，让你自由的配置你想要的分组。另一方面，智能分组会根据指定的条件动态的改变分组内容。你可以使用智能分组来显示一个最近使用的列表，或者 instruments 名称、描述包含特定关键字的分组。

创建静态分组

为了创建一个静态分组，选择库窗口的 action menu 的 New Group，如图 2-4 所示。

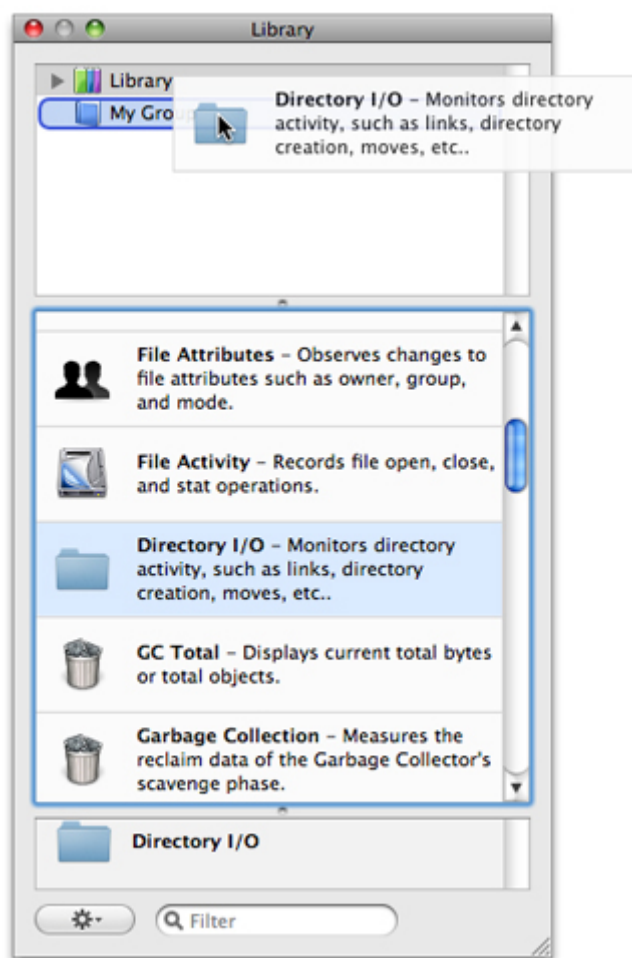
Figure 2-4 Creating a group



Instruments 创建了一个新的分组并把它添加到库窗口。为了查看新的分组和编辑它的名称和内容，拖动分割栏到弹出菜单的下面来打开一个顶部面板显示分组的结构，如图 2-5 所示。点击分组并填写一个新的名称。

Figure 2-5 Editing the name of a group

为了添加一个 instrument 工具到静态分组，点击 Library 来显示 instruments 工具列表，选择 instrument 并拖动它到上面的分组面板，如图 2-6 所示。

Figure 2-6 Adding an instrument to a custom group

你可以使用静态分组来创建层级的 instruments 组织。静态分组可以包含其他静态分组，而且它还可以包含智能分组。然而智能分组不能包含其他分组。为了创建分组的层级结构，你可以通过以下其中任何一种方式达到：

- 拖动一个分组到父静态分组
- 选择一个静态分组，然后选择 New Group 或 New Smart Group 来创建一个新的项目作为所选择的静态分组的子分组。

一个包含有其他分组的分组显示它自己的 instruments 工具和它所有子分组的 instruments 工具。如果同一个 instrument 出现在多个分组，并且 Show Group Banners 选择勾选了，那么同一个 instrument 工具会在每个包含它的分组下面列出。如果 Show Group Banners 选择没被勾选，那么库窗口合并相同的 instrument 副本到同一个主体里面。

为了把一个 instrument 从静态分组删除，你需要完成以下的事情：

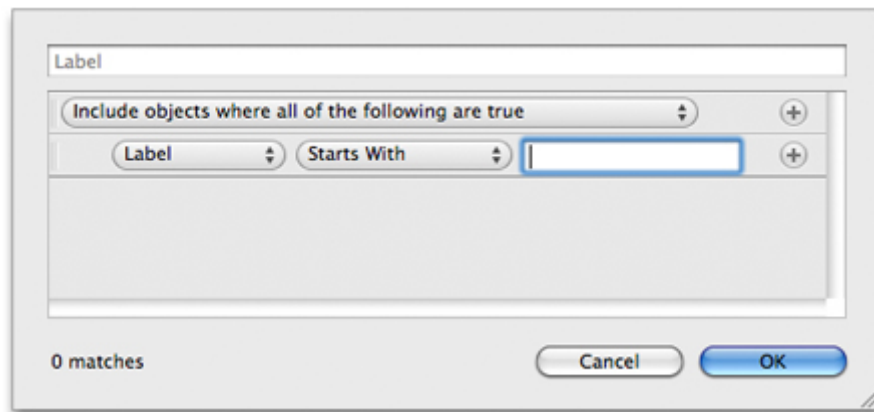
1. 选择一个分组。
2. 在分组里面选择一个你想要删除的 instrument。
3. 选择 action menu 的 Remove From Group，或简单的点击删除键。

为了从库里面删除一个静态分组，选择分组，然后选择 action menu 里面的 Remove Group。如果你当前正在使用大纲视图查看分组，你也可以选择对应的分组，然后按删除键。删除一个静态分组会删除该分组和它所包含的所有嵌套分组。然而该操作并没有从库里面删除它包含的 instruments 工具。你可以在库分组里面看到这些 instruments 工具。

创建一个智能分组

为了创建一个智能分组，选择在库窗口的 action menu 里面的 New Smart Group。图 2-7 显示了编辑规则。标签域标示了分组在库窗口里面出现的名称。其余的部分是专门的配置规则，确定那些 instruments 留在分组里面。

Figure 2-7 The smart group rule editor



每个智能分组都必须包含至少一个规则。你可以跟踪需要添加额外的规则，使用规则编辑的控制器和配置分组来应用所有或者部分规则。表 2-1 列举了你可以用来匹配 instruments 的标准。

Table 2-1 Smart group criteria

Criteria	Description
Label	Matches instruments based on their title. This criterion supports the Starts With, Ends With, and Contains comparison operators.
Used Within	Matches instruments based on when they were used. You can use this criterion to match only instruments that were used within the last few minutes, hours, days, or weeks.
Search Criteria Matches	Matches instruments whose title, description, category, or keywords include the specified string.

Category	Matches instruments whose library group name matches the specified string. This criterion does not match against custom groups.
----------	---

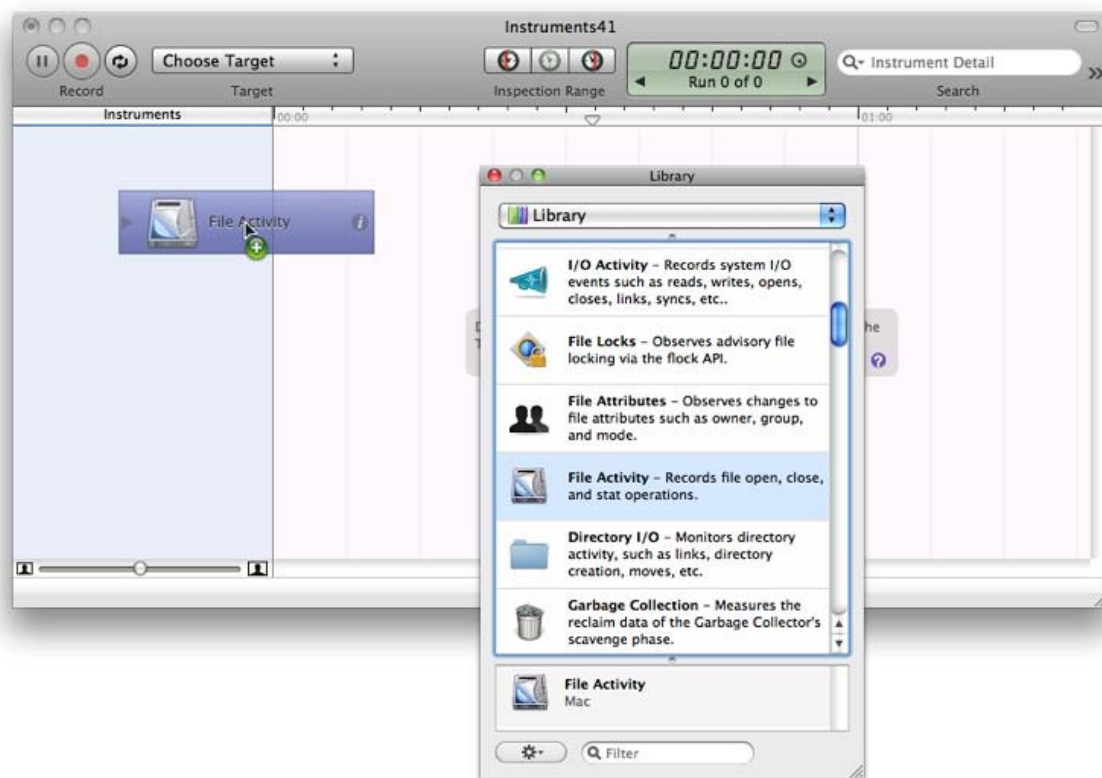
为了编辑一个已有的智能分组，选择库窗口的分组，然后选择编辑 action menu 的<Group Name>，其中<Group Name>是你对应智能分组的名称。Instruments 应用会再次显示规则编辑器，以便你可以修改已有的规则。

为了从库窗口里面删除一个智能分组，选择分组，然后选择 action menu 的 Remove Group 选项。如果你当前正使用大纲视图来查看分组，你也可以选择分组然后按删除键。

2.2 添加和删除 Instruments 工具

为了添加一个 instrument 到跟踪文档，从库窗口里面拖动到你跟踪文档的 Instruments 面板或者跟踪面板，如图 2-8 所示。

Figure 2-8 Adding an instrument



你可以添加任意多的 instruments 工具到你的跟踪文档里面。尽管大部分 instruments 可以跟踪系统进程，但是许多只能跟踪一个进程。对于这些 instruments 工具，你可以使用多个 instrument 的实例并把它们赋值给你想要的不同进程。通过

这样的方法，你可以同时收集多个程序的相同信息。比如，你可能需要在系统进程的样本和它相应的客户端进程上面这么做，分析整体的交互模式。

为了从一个跟踪文档删除一个 instrument 工具，选择 Instruments 面板中对应的 instrument 工具然后按删除键。

2.3 配置一个instrument工具

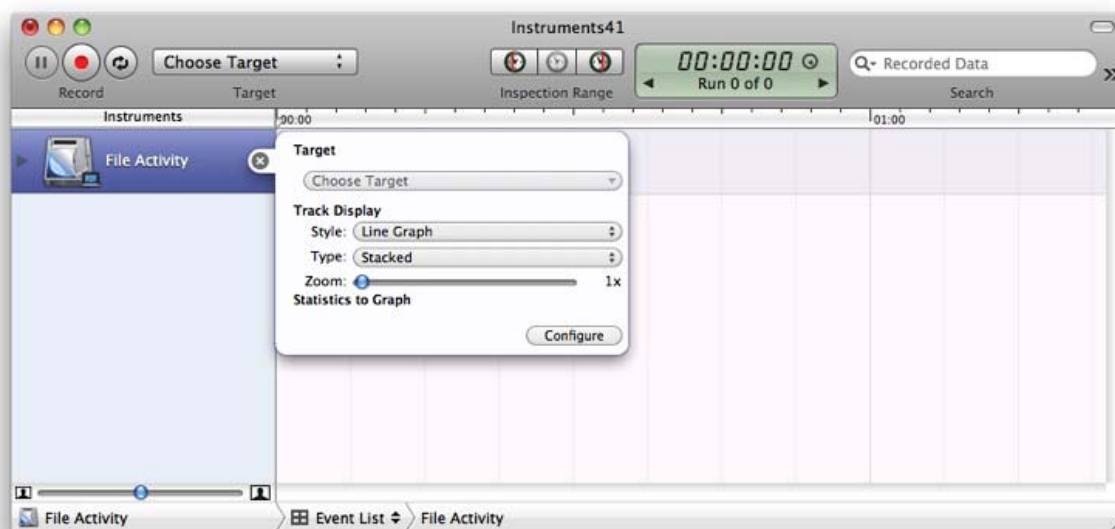
一旦你把 instrument 工具添加到 Instruments 面板，那么大部分 instruments 工具即可运行。部分 instruments 工具也可以使用 instrument 检查器来配置。检查器的内容因 instrument 工具不同而不同。大部分 instruments 工具包含了配置的可选选项来配置跟踪面板的内容，而少部分包含了额外的控制器来决定 instrument 工具自己收集什么类型的信息。

为了打开一个给定的 instrument 工具的检查器，选择 instrument 工具，然后做以下任何一种操作：

- 点击 instrument 工具名称右边的检查器图标
- 选项 File > Get Info
- 单击 Command-I

检查器会出现在 instrument 工具名称的旁边，如图 2-9 所示。为了隐藏检查器，你可以单击关闭按钮。（你用来打开检查器的命令同样可以用来关闭检查器）

Figure 2-9 Inspector for the File Activity instrument



在跟踪面板上面和显示信息相关的控制器可以在你给跟踪记录数据之前、期间、之后配置。Instruments 工具自动为显示选项收集它需要的数据，无论显示选项当前是否显示在跟踪面板上面。

在跟踪面板中的检查控制器里面的缩放控制器可以用来校验跟踪数据的放大率。改变缩放值会改变 instrument 工具在跟踪面板里面的高度。View > Decrease Deck Size 和 View > Increase Deck Size 命令做类似的工作来递减和递增跟踪面板里面的所选 instrument 工具的放大倍率。

关于 instruments 工具的列表，包含配置选项，参阅“内置 instruments 工具”部分。

第三章 记录跟踪数据

在决定使用什么 instruments 工具来收集数据之后，下一步就是选择需要跟踪的进程，并开始记录数据。如何选择进程依赖于你跟踪文档里面的 instruments 工具。部分 instruments 工具允许跟踪所有系统进程，其他要求你只能为一个进程记录数据。部分 instruments 工具甚至希望你从 Instruments 应用里面启动进程以便它们可以在进程执行之前收集数据。

Instruments 应用提供了几个初始化跟踪的可选项，它们可以让你更容易的整合 instruments 工具到你的开发周期里面。在本章里面，你将会学到如何为一个跟踪文档选择一个进程，并开始使用 Instruments 应用里面可选的选项来记录数据。

3.1 选择需要跟踪的进程

在你开始收集数据之前，你必须告知 Instruments 应用，它处理你想要的跟踪。你可以通过使用 Instruments 工具栏上面的 Target menu 来指定一个目标进程（或多个进程）来完成。该菜单提供了以下的选项：

- **All Processes (所有进程)**. 配置你的文档来跟踪所有系统进程。
- **Attach to Process (附加到进程)**. 配置你的文档来跟踪一个已经运行的进程。
- **Choose Target (选择目标)**. 配置你的文档来启动并跟踪一个进程。（如果进程已经运行，Instruments 应用会启动一个新的副本并跟踪它）
- **Instrument Specific (Instrument 指定)**. 选择该项目来给独立的 instruments 工具指定不同的跟踪目标。

以下各部分描述了每个选项更详细的信息。

3.1.1 跟踪所有进程

部分 instruments 工具可以给当前计算机所有运行进程收集数据。你可以使用这些功能来配置文档到整个系统中的事件或活动类型。比如，你可以使用 Disk Activity 的 instrument 工具来跟踪所有在特定周期内发生在你计算机上面的可读和可写的操作。为了跟踪所有运行的进程，选择 Target menu 里面的 All Process 选项。

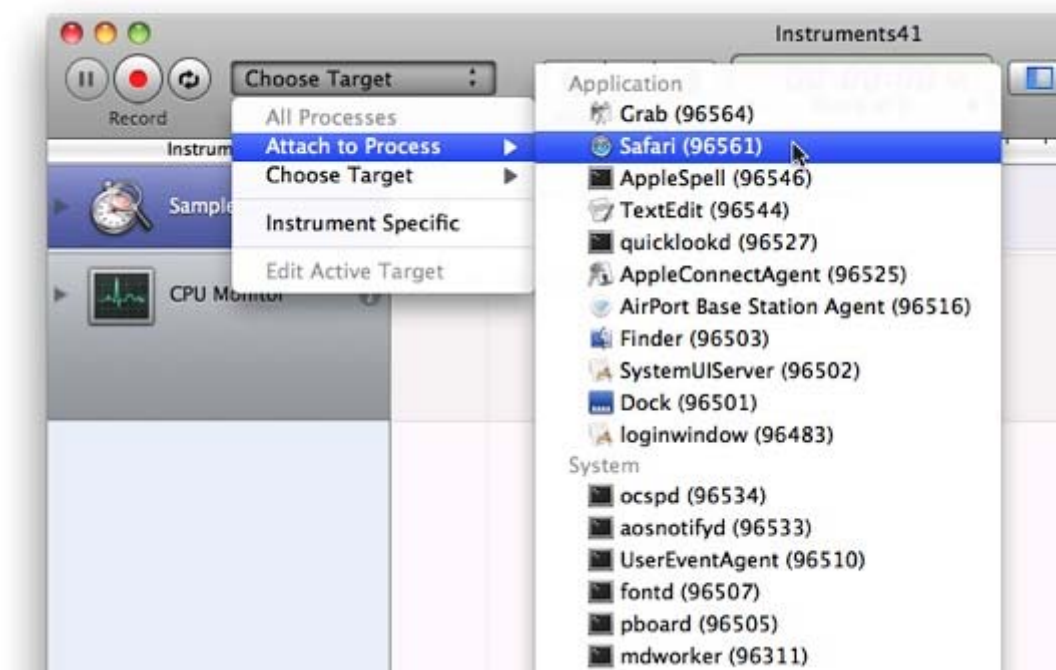
注意： *All Processes* 项目只有当你 Instruments 面板上面的所有 instruments 工具都支持

的时候才可用。

3.1.2 跟踪一个已有的进程

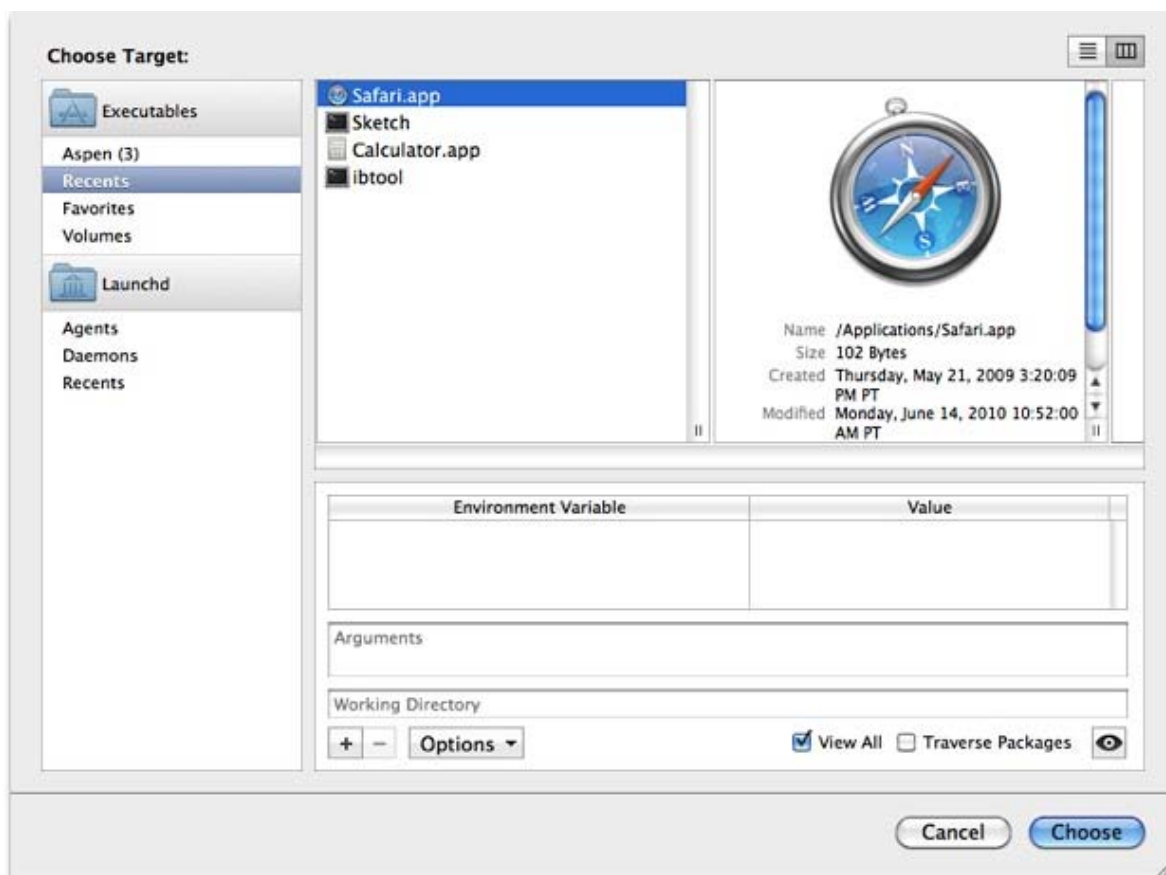
如果你想要跟踪的进程已经在计算机上面运行了，那么大部分 instruments 工具可以附加到该进程中并开始收集数据。为了跟踪一个已有的进程，选择 Target menu 里面的 Attach to Process 选项并需要跟踪的进程，如图 3-1 所示。

Figure 3-1 Tracing an existing process



3.1.3 跟踪一个新的进程

如果你想要跟踪的进程还没在计算机上面运行，或者你想要控制已经启动的进程的条件，选择 Target menu 上面的 Choose Target 选项。Instruments 应用会记录最近启动过的进程并把它们添加到 Choose Target 的子菜单下面提供快速访问。如果你想要启动的进程没有在菜单里面，选择 Choose Target 来显示对话框，如图 3-2 所示：

Figure 3-2 Choosing a target to launch

Choose Target 对话框让你选择要启动的程序，同时让你指定如何启动所选的程序。表 3-1 列出了对话框中额外的控制并解析如何使用它们。

Table 3-1 Options for launching an executable

Control	Description
Environment Variable	Identifies environment variables you want to set before running the process. You might use this option if your program has debugging options that are enabled using an environment variable. Use the plus (+) and minus (-) buttons to add or remove environment variables.
Arguments	Use this field to specify any launch arguments for the application. The arguments you specify are the same ones you would use from the command line when launching the application there.
Options	Use this menu to specify other runtime options. For example, you can direct the application's output to the Instruments console or the system console, or discard the output. You can also specify whether the application is launched in 32-bit or 64-bit mode.
View All	Sets the specified application as the target for all instruments in the trace document. This option is enabled by default. Disabling it lets you assign different targets to different instruments in your trace document. You might use this feature when you have two copies of the same instrument or when you want to trace the behavior of two processes running side by side.
Traverse Packages	Displays bundles (such as applications and plug-ins) as a navigable directory structure. Use this feature if the executable you want to run is inside a bundle.

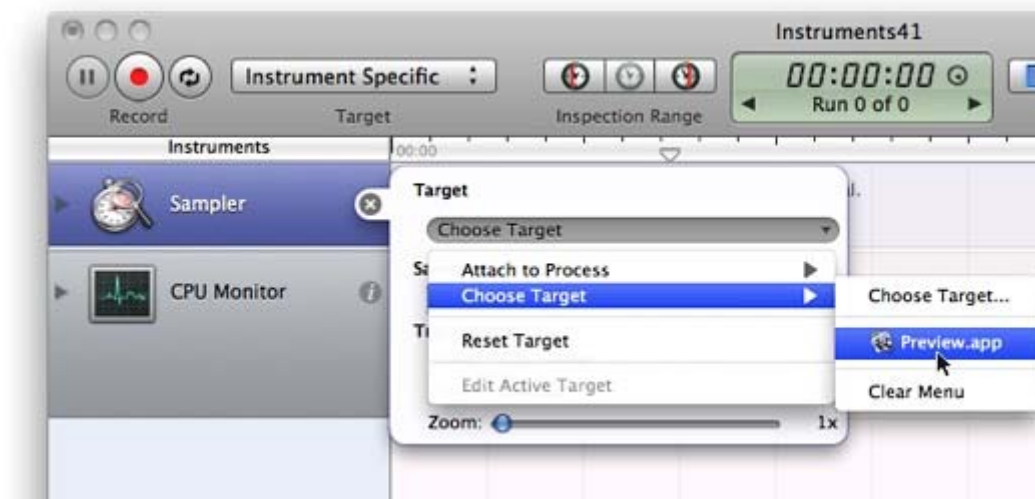
3.1.4 给每个Instrument工具指定不同的目标

当你在 Target menu 里面选择了 All Processes、Attach to Process 或 Choose Target 选项, Instruments 应用给你选择的程序的所有 instruments 工具设置默认的目标。然而无论何时你都可以给每个 instrument 工具设置不同的目标。你可能想要在同一时间采样两个不同的程序, 比如使用 instrument 示例。如果你已经有一些 instruments 工具可以跟踪所有进程, 你可能想让它们在单进程 instruments 样本的时候仅是一个进程。

为了给每个 instrument 工具设置基础的跟踪目标, 你可以做以下这些:

1. 在 Target menu 里面, 选择 Instrument Specific 的选项。
2. 选择你的 instruments 工具里面的其中一个。
3. 单击 i 图标来打开 instrument 工具的检查器。
4. 在检查器的窗口, 使用目标区域的弹出菜单来给 instrument 工具设置目标, 如图 3-3 所示。

Figure 3-3 Choosing per-instrument trace targets



5. 重复第 2 步到第 4 步来给每个 instrument 工具设置目标。

3.2 收集数据

在你已经选择了一个要跟踪的进程之后, 你就可以开始收集数据了。你可以使用两个不同的记录模式之一来收集数据:

- **Immediate display (立即显示)**。在测量的期间, Instruments 应用在跟踪面板和

详细面板立即显示收集的数据。Instruments 工具栏上面的时间控制器同样显示了你开始记录数据多长时间了。在这个模式下，Instruments 影响系统性能，因为 UI 要持续的更新。这个模式是 Instruments 应用的默认模式。为了选择立即显示，选择 File > Record Options > Immediate Display。

- **Deferred display(延迟显示)**。Instruments 应用延迟显示收集的数据直到记录停止。在测量期间，Instruments 对系统的影响非常小。当你停止记录的时候，Instruments 应用开始处理并显示收集的数据。延迟显示是一个很重要的特性，当应用程序对性能很敏感的时候。为了选择延迟显示，选择 File > Record Options > Deferred Display。
记录模式随文档持久性的。

点击 Record 按钮（或选择 File > Record Trace）来开始收集跟踪的数据。当你单击 Record 时，Instruments 启动指定的可执行文件或附加到指定进程，并开始收集数据。为了停止收集数据，单击 Stop 按钮或者选择 File > Stop Trace。

注意：当你单击 Record 按钮时，Instruments 应用有可能显示一个或多个认证对话框。一些 *instruments* 工具在开始记录任何数据之前需要验证你是否是管理员。Instruments 应用是一个很强大的工具，可以让你查看正在运行的应用，因此它应该只能被授权的用户使用。

在记录期间，如果你想要你的程序继续运行，但是又不想让 Instruments 应用收集数据，单击你的跟踪文档里面的 Pause 按钮。Instruments 应用暂时停止收集数据，但是并没有停止当前正在执行的记录。单击 Resume 按钮会让 Instruments 应用继续当前记录时间里面收集数据。所以暂停和继续会在你的跟踪面板上面产生一个数据空白区域。

3.3 使用快速启动键启动 Instruments

快速启动键是全局组合键，它可以让你启动 Instruments 应用程序并使用指定的文档模板立即开始收集跟踪数据。你可以使用这一特性，如果你当前正在运行一个应用程序或者查看一些你需要立即捕获的事情（比如调试或反应迟钝的行为）。你可以关联不同的组合键到不同的 *instruments* 模式上来捕获不同的行为。

为了赋值一个快速启动键，打开 Instruments 应用的偏好设置，并导航到快速启动面板。该面板显示了你可以赋值组合键的 *instruments* 模板的列表。所有的模板

没有快捷键启动。要赋值一个的话，找到包含设计模板的行，双击它的 Key 列来创建一个可编辑的单元。当该单元处于编辑模式时，选择你想要的组合键并单击它们。比如，为了赋值一个组合键 Command-Option-1，同时按下 Command、Option 和数字 1 的键。

为了从模板中删除一个组合键，选择一行并按删除键。

注意：快速启动组合键起码要使用至少两个不同的按键（Command、Option、Control、Shift）。你应该避免使用已经被其他应用程序使用的组合键。

为了使用快速启动键来给应用程序收集数据，你要做以下步骤：

1. 把光标放在属于你要跟踪的应用程序的窗口上面。
2. 按下合适的组合键来开始跟踪。
3. 运行应用程序。
4. 当你想停止跟踪的时候，你有两种选择：
 - 把光标放在其中之一的应用程序窗口上面，再次按下组合键。
 - 找到已经打开的跟踪文档并按下 Stop 按钮。

因为快速启动键要求你把光标移动到其中一个应用程序窗口之上，你可以使用同一个快速启动键来给不同的应用程序初始化多个跟踪对话而无需停止任何之前的跟踪。你也可以使用不同的组合键来给同一个应用程序开始不同类型的跟踪，并让它们在同一个时间收集所有的数据。

3.4 以最小模式运行

最小模式（Mini mode）为你提供了一种在收集数据的时候尽量减少 Instruments 应用程序可视化尺寸的方法。当你从特定类型的应用程序收集数据的时候，尤其是面向图形的应用程序，在你集中于你应用程序的时候，很多时候你需要收集你的数据。最小模式隐藏很多打开的跟踪文档，在它们的地方显示很多浮动的窗口，你可以使用它们来开始和停止跟踪。最小模式的一个优点是 Instruments 应用程序本身需要很少的绘画，所以对系统性能影响很少。

图 3-4 显示了打开几个跟踪文档的最小化 Instruments 应用窗口。Instruments 应用在最小模式的时候同一时间只显示三个跟踪文档，但是你可以使用向上和向下箭头来找你想要的跟踪文档。

Figure 3-4 The Mini Instruments window



单击跟踪文档附近的 Record 按钮开始为文档记录数据。文档在跟踪之前必须配置好它的 instrument 工具和目标进程。在跟踪期间，Instruments 应用显示跟踪开始后所用的时间，但是不显示其他 instruments 工具、控制器或数据。

为了启用最小模式，选择 View > Mini Instruments。为了禁用最小模式，单击最小模式的 Instruments 窗口的关闭框。你也可以在标准和最小模式之间切换，通过选择 View > Mini Instruments。

3.5 从Xcode运行Instruments应用

在开发期间，你可以从 Xcode 3 的用户界面直接加载你的应用程序到 Instruments 应用里面。这个整合功能可以让你同时快速的加载 Instruments 应用和收集跟踪数据，那会启动你的程序并使用 GDB 调试它。

Xcode 3 里面的 Run > Run with Performance Tool 子菜单提供了几种使用可用性能工具来加载你程序的方法，包括 Instruments。当加载你的应用程序到 Instruments 应用的时候，你可以通过选择合适的菜单项目告诉 Xcode 你想要使用的 Instruments 模板。Xcode 加载 Instruments, 使用特定的模板新建跟踪文档，为你的程序设置目标，并告诉 Instruments 来加载你的程序和开始记录数据。

除了在 Run with Performance Tool 子菜单已有的模板，Xcode 同样允许你添加自定义的跟踪模板到菜单栏。关于更多如何保存一个跟踪模板并把它添加到菜单的信息，参阅“保存 Instruments 跟踪模板”部分。

3.6 无线连接iOS设备

为了使用扩展访问给 iOS 应用程序收集数据，你可以在 Instruments 应用和你的设备之间建立一个无线连接。当一个无线设备不切实际或不方便的时候你也可以使无

线连接。

软件要求：该特性在 *iOS SDK 3.1* 及其之后可用。

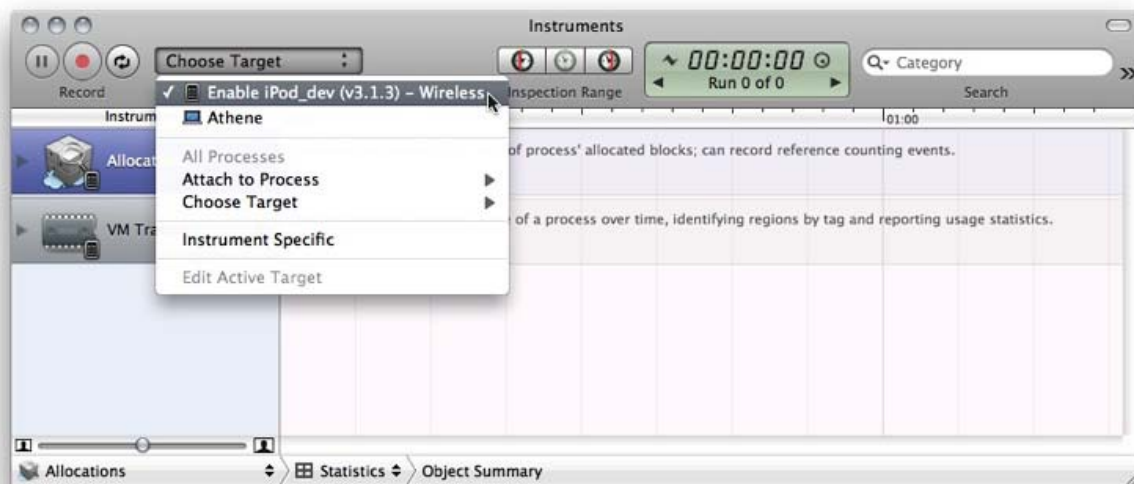
通常来说，如果你不需要附加一个扩展访问到你的设备，你需要使用一个有线连接。无线连接比有线连接慢，并且要耗更多设备的电来适应来自 Instruments 的繁忙信息流。结果，使用无线连接比有线连接耗费更多的电池。

重要：为了让 Instruments 应用可以无线连接你的设备，你的 Mac 设备和 iOS 设备都必须拥有开启的无线网络，并且必须连接到同一个无线接入点。该接入点必须被配置提供 Bonjour 服务。

为了建立一个 Instruments 应用和你设备自己的无线连接：

1. 把你的设备通过 USB 线连接到你的电脑。
2. 在 Xcode 里面，把你的应用程序载入你的设备。
3. 在 Instruments 里面，打开或新建一个跟踪文档。
4. 按下 Option 键，并选择 Target menu 里面的“Enable <device_name> - Wireless”选项，如图 3-5 所示。

Figure 3-5 Creating a wireless connection between Instruments and an iOS device



一会之后，一个无线的目标出现在 Target menu 上面，名为“<device_name> Wireless”。如果该无线目标不出现，你的 Wi-Fi 接入点可能没有被配置来提供 Bonjour 服务。

这个时候你有两个 Instruments 服务器运行在你的设备上面：一个使用有线连接，另一个使用无线连接。

5. 从你的设备上断开 USB 线，然后连接扩展访问。

现在你可以使用 Instruments 应用来收集关于你应用程序性能的数据，就如“选择要跟踪的进程”部分介绍的一样。

无线连接即使在 Instruments 退出或锁定你的设备之后，它依然起作用。在网络连接条件如之前提到的一样，你可以继续使用无线连接。

为了中断无线连接：

1. 在 Instruments 应用里面，打开一个跟踪文档。
2. 按下 Option 键，然后选择 Target menu 下面的“Disable *<device_name>* - Wireless”。过了一会之后，该“*<device_name>* Wireless”的目标变暗，意味着无线连接和 Instruments 服务器已经中断。

第四章 记录用户界面轨迹

用户界面轨迹记录运行程序的一系列事件和操作。在事件被记录之后，你可以多次回放跟踪来反复生成相同的一系列事件。每次你回放用户界面轨迹时，你可以在你的跟踪文档里面使用其他 instruments 工具来收集数据。这样做的好处是每次成功运行后你都可以比较你收集的数据，并且使用它来改变你程序的性能或行为。

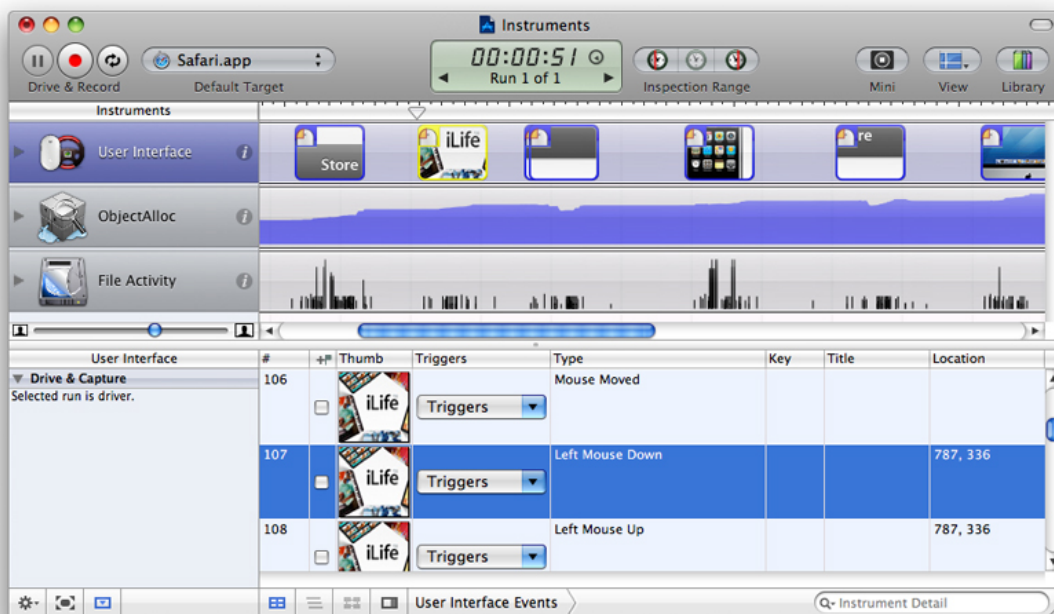
4.1 记录用户界面轨迹

你可以使用 User Interface 的 instrument 工具来记录用户界面轨迹。你把该 instrument 工具添加到跟踪文档，就像添加其他 instrument 工具那样。当你单击你文档里面的 Record 按钮时，User Interface instrument 工具开始收集和用户输入相关的事件，例如鼠标和键盘事件等。它收集这些事件并在跟踪面板上面显示它们给你查看和检查。

注意：如果你使用 UI Recorder 模板来新建一个新的跟踪文档，Instruments 应用会自动为你添加 User Interface 的 instrument 工具到文档里面。

在你为用户界面轨迹收集了一系列事件后，Record 按钮的标题会改为 Drive&Record。再次点击这个按钮会让 Instruments 应用来驱动用户界面，重放你之前记录的一系列有序的事件。当做这些的时候，在你跟踪文档里面的其他 instruments 工具通常会收集数据。在事件流完成后，你应该检查运行数据。图 4-1 显示了一个使用用户界面轨迹和其他 instruments 工具的跟踪文档。这里用户界面轨迹已经包含了一系列待重放的有序的事件。

Figure 4-1 Recording a user interface track



重要：录制用户界面通常使用屏幕阅读器和其他为残疾人使用的辅助设备的访问特性。为了保证你的计算机支持该特性，打开系统偏好设置的全局访问（Universal Access）面板，确保“允许访问辅助设备（Enable access for assistive devices）”的设置被启用。

User Interface instruments 工具的详细面板列出了被记录的事件，事件的定位，和被任何按下的键。该 instrument 工具同样可以捕获屏幕被用户操作的部分区域。缩略图版本的屏幕截屏被同时显示在跟踪面板和详细面板上面。每个事件都根据它的不同类型进行着色：

- 鼠标事件为蓝色
- 键盘事件为绿色
- 系统事件为黄色

4.2 重复记录用户界面轨迹

如何在捕获一系列事件后，你决定没有获得正确的事件序列，你可以回头并重新捕获需要的事件直到获得正确的。在你重新捕获事件之前，你必须告知 Instruments 应用不要使用老的事件序列来驱动用户界面。User Interface 检查器的 Action 区域包含了一个弹出菜单，它可以让你指定让 instrument 工具如何运行。通常在你捕获

事件序列后，Action 设置会被找到设置为 Drive。为了重新捕获，你必须修改设置为 Capture。完成这些之后，你可以开始记录新的事件序列了。因为每次捕获完成后 Action 设置会切换回 Drive，所以多次重新捕获一个事件序列的时候要求你每次记录之前把 Action 设置修改为 Capture。

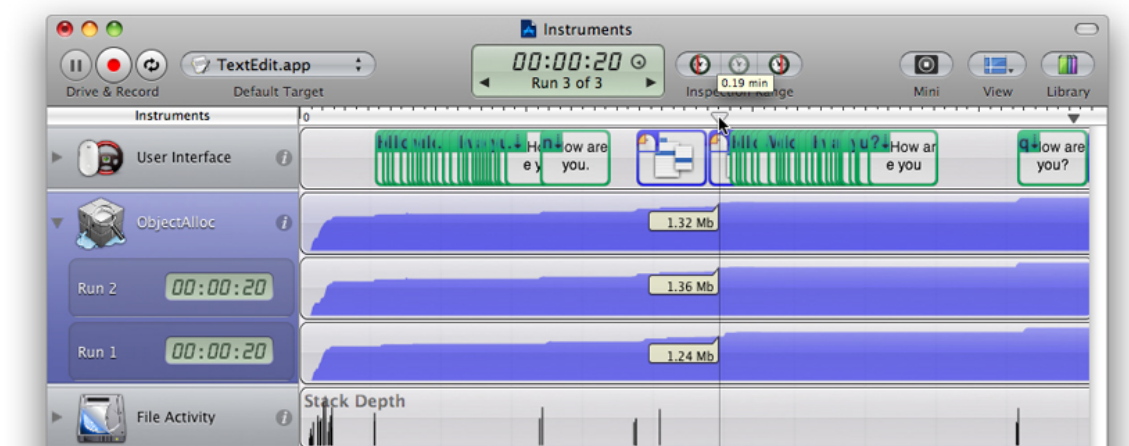
4.3 回放用户界面轨迹

在你记录了用户界面轨迹之后，你可以回放这些轨迹来重新生成一系列你记录的用户事件。在记录了一系列事件之后，Instruments 应用会自动把它的 Record 按钮的标题改为 Drive&Record。单击该按钮来启动选择的应用程序，并执行在用户界面轨迹里面每个记录的动作。与此同时，Instruments 应用开始在 Instruments 面板上面其他活动的 instruments 工具收集相关的数据。

每次成功运行用户界面轨迹后，Instruments 应用都会在跟踪面板显示运行的结果。为了修改跟踪面板中活动的结果，你可以使用事件控制器的箭头。单击这些箭头切换于改可用的数据集，会没每个可以的 instruments 工具更新显示结果。

你也可以同时查看同一个 instrument 工具所有运行的结果。每个 instrument 工具在它的左侧都会有一个下拉的三角形。点击该控件可以扩展 instrument 工具并对齐的显示之前运行的所有结果数据，如图 4-2 所示。

Figure 4-2 Viewing multiple runs



当收集跟踪数据时，你可以点击文档工具栏的 Loop 按钮来让 Instruments 应用多次重复用户界面轨迹的事件。该按钮是黏性的，所以单击它启动循环而再次单击则禁用循环。当循环启用是，Instruments 应用在连续循环里面运行一系列有序的事件，

在循环过程中每次收集新的跟踪数据。你可以使用该特性来重复一系列事件，这些事件是多次尝试后展示的不良行为。你也可以使用循环来让你的应用程序持续工作来对它进行压力测试。

第五章 查看和分析跟踪数据

因为 Instruments 应用可以让你同时收集多个 instruments 工具的数据，因此你可能获得的数据量在短时间是大量而且压倒性的。幸运的是，Instruments 提供了很好的接口来帮助你组织和显示你收集的数据以便你分析和导航这些数据。弄懂你跟踪文档窗口里面不同区代表的信息对于你查看趋势和查找潜在问题非常重要。

每个跟踪文档都包含了以下接口元素：

- 跟踪面板 (Track pane)
- 详细面板 (Detail pane)
- 展详细面板 (Extended Detail pane)
- 运行浏览器 (Run Browser)

这里的每个控件显示的都是你跟踪文档里面相同的数据。它们只是以不同的方式显示而已，关于每个数据点从高级的预览到详细的信息等。这样可以让你以不同的方式来查看你的数据。你可以通过查看高级的数据来分析变化趋势，然后查看详细的数据来确定在你的代码中将要发生的事情和制定关于如何解决潜在问题的思路。

除了用户界面的可选项，许多 instruments 工具可以通过配置来以不同的方式呈现数据。你可以使用这些配置选项在分析过程中获取对收集的数据新的透视。

本章中你可以学习如何使用这些 Instruments 应用给你提供的的数据，和如何按照你的工作流需求修改这些信息数据的呈现方式。

5.1 查看数据的工具

以下部分描述了 Instruments 用户界面里面关键的元素和如何使用这些元素来查看跟踪数据。

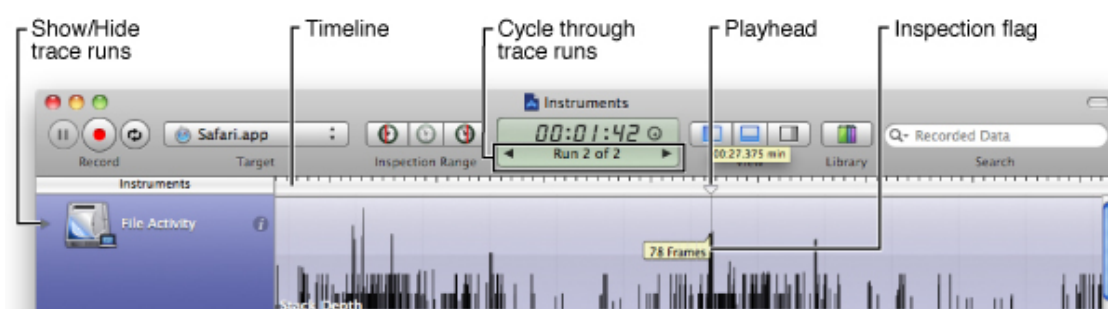
5.1.1 跟踪面板

跟踪面板是跟踪文档窗口里面最突出的部分。跟踪面板紧接于 instruments 面板的右边。该面板为每个 instrument 工具提供了收集数据的一个高级图形视图。你可以使用该面板来核查你从每个 instrument 工具收集到的数据，并可以选择你想进一步研究的区域。

跟踪面板中的自然图形可以让你更容易的发现你程序的趋势和潜在的问题。比如，图形中一个内存使用的尖峰信息意味着在该地方你的程序分配的内存比平时更多。该尖峰信息可能是正常的，也可能是意味你的代码比你预计在此地方创建了更多的对象或内存缓冲区。一个 instrument 工具（例如 Spin Monitor instrument）也同样可以指出你程序反应迟钝的地方。如果 Spin Monitor 的图形相对是空白的，你就知道你的程序是正常响应，但如果该图形不空白，那么你可能需要核查为什么会出现这样的情况。

图 5-1 显示了一个跟踪文档的示例，该示例显示了所有跟踪面板的基本特性。你可以使用该面板顶部的时间线来选择核查的地方。点击时间线并移动播放头来定位和显示检查器的旗帜 (inspection flags)，该旗帜总结了 Instrument 中该定位地方的信息。你也可以点击时间线来聚集数据点在详细面板上面的信息；参阅“详细面板”部分。

Figure 5-1 The track pane



虽然每个 instrument 工具都不同，但是它们绝大部分都会提供修改跟踪面板数据显示的选项。除此之外，部分 instruments 工具还可以通过配置在它们跟踪面板里面显示多个数据集。这两个特性可以让你选择按照你程序最方便的形式来显示数据。

以下部分提供了关于跟踪面板更多的信息，以及如何配置它们。

查看跟踪运行

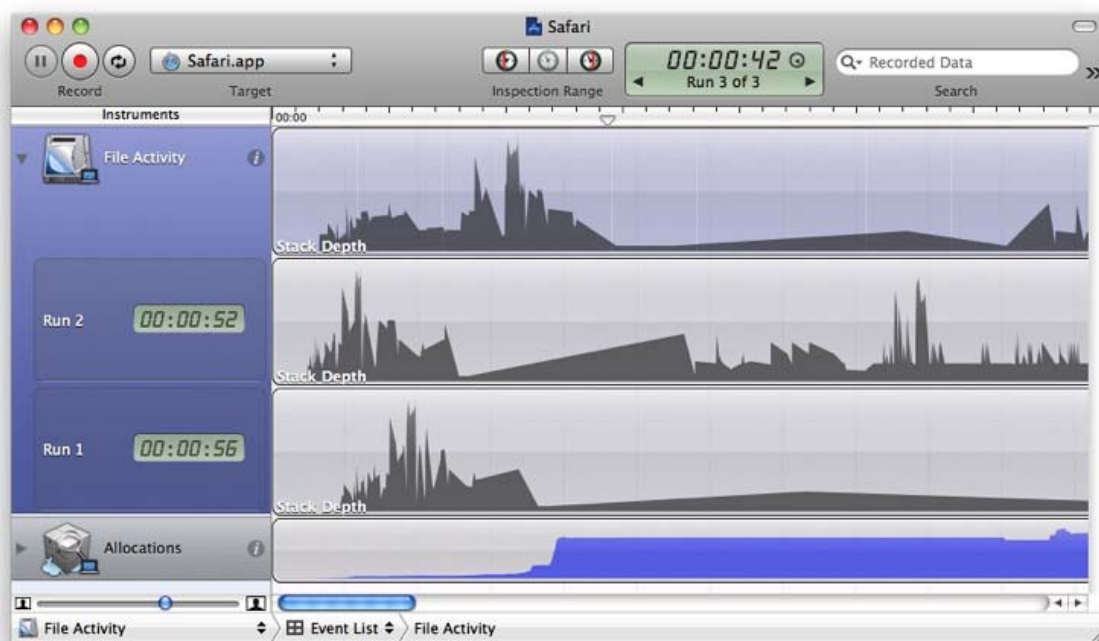
每次你点击跟踪文档里面的 Record 按钮，Instruments 应用就会开始收集目标进程的数据。Instruments 应用创建新的跟踪运行来保存数据，而不是通过把新的数据拼接到现有数据的后面。一个跟踪运行 (trace run) 构成由你点击了 Record 按钮来开始收集到点击 Stop 按钮来停止收集之间的数据。默认情况下，Instruments 应用只在跟踪面板里面显示最近的跟踪运行，但是你也可以通过以下任何一种方法来查看之

前的跟踪运行的数据：

- 使用工具栏上面的 Time/Run 控件来选择你想要查看的跟踪运行。
- 点击 instrument 工具旁边的下拉三角形来显示该 instrument 工具下面的所有跟踪运行的数据。

图 5-2 一个已经被扩展显示多个跟踪运行的 instrument 工具。在该例中，几个跟踪运行都已经收集了数据，这些数据都是由轻微不同的事件产生。当你正在尝试重现一个并不是每次运行都可以出现的问题时，你可以收集多个跟踪运行的数据，创建新的跟踪运行直到问题出现。然而如果你想要使用相同的事件集来比较多个跟踪运行，你需要创建新的用户界面轨迹和一个想要的进程，该进程在“使用用户界面轨迹”部分介绍。

Figure 5-2 Viewing multiple runs of an instrument



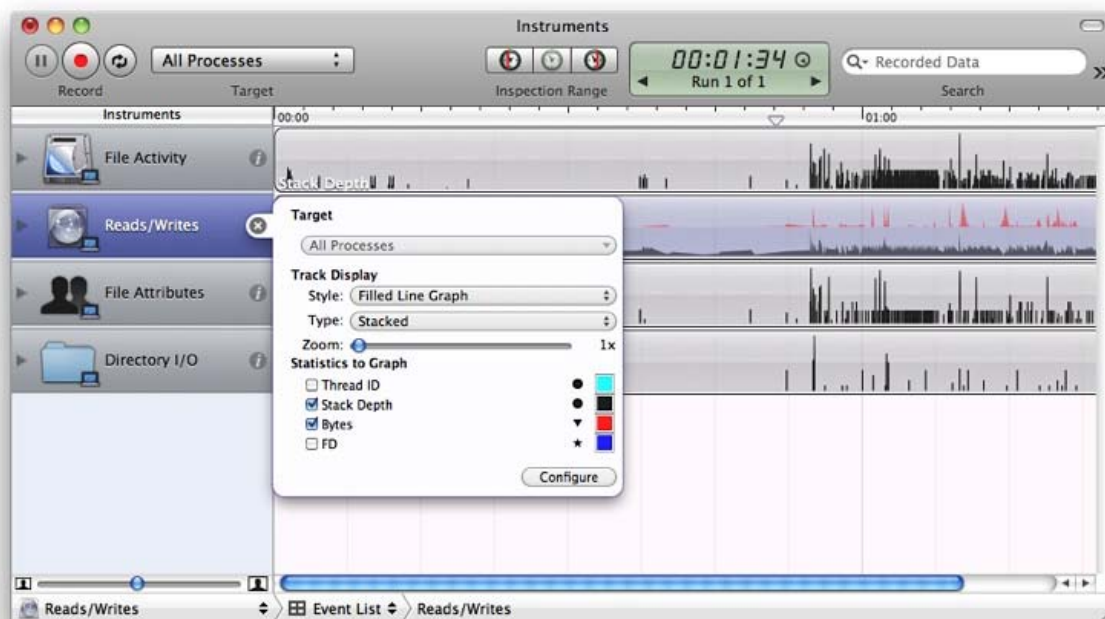
当你保存一个跟踪文档，Instruments 应用通常会摒弃所有跟踪运行的数据，除了你当前选择的那个。这样可以让你的跟踪文档体积不至于太大。如果你想要保存你跟踪文档里面所有跟踪运行的数据，进入 Instruments 应用的偏好设置的 General，并取消勾选 Save Current Run Only 选项。

显示同一个 Instrument 工具的多个数据集

一些 instruments 工具是可以在相应的跟踪面板显示多个数据流。你可以使用

instrument 检查器来显示特定的数据流，如图 5-3 所示。Statistics to Graph 列出了所有由 instrument 工具收集的整型的数据值。勾选相应的复选框来添加相应的统计到跟踪面板里面。为了编辑 Statistics to Graph 里面统计的列表，选择 Configure 按钮。

Figure 5-3 Configuring the statistics to graph

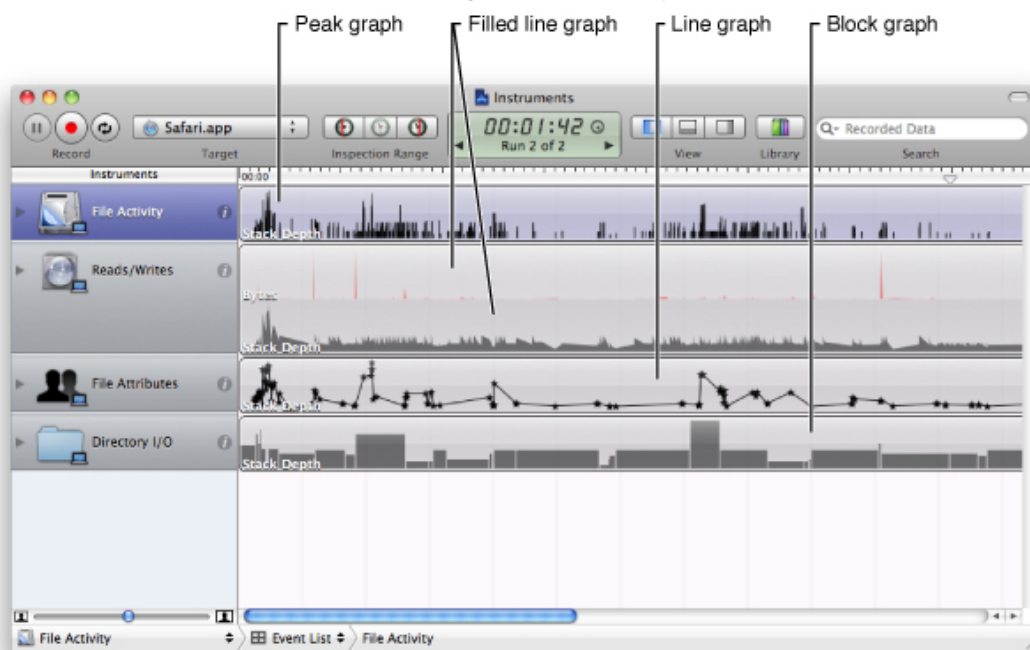


关于每个 instrument 工具收集的数据的信息，参阅“内置 instruments 工具”。

修改跟踪显示的样式

Instruments 应用提供了几种在跟踪面板显示收集的数据的样式。大部分 instruments 工具默认选择一个可以显示多个数据类型的样式。你可以使用 instrument 的检查器上面的样式弹出菜单来修改选择的样式。图 5-4 显示了几种样式。

Figure 5-4 Track styles



注意: 你所选择的样式会影响所有 *instrument* 工具显示的数据流。

缩放跟踪面板

Instruments 应用提供了对跟踪面板数据在水平方向轴和垂直方向轴的缩小和放大。

- 为了修改水平方向(时间相关)的放大倍数, 使用 instruments 面板下面的滑块控制器。
- 为了修改垂直方向(幅度和体积相关)的放大倍数, 选择一个 instrument 工具, 并做下面任一事情:
 - 打开 instrument 工具的检查器, 使用缩放滑块。
 - 选择 View > Increase Deck Size 的菜单选项来递增缩放因子。
 - 选择 View > Decrease Deck Size 的菜单选择来递减缩放因子。

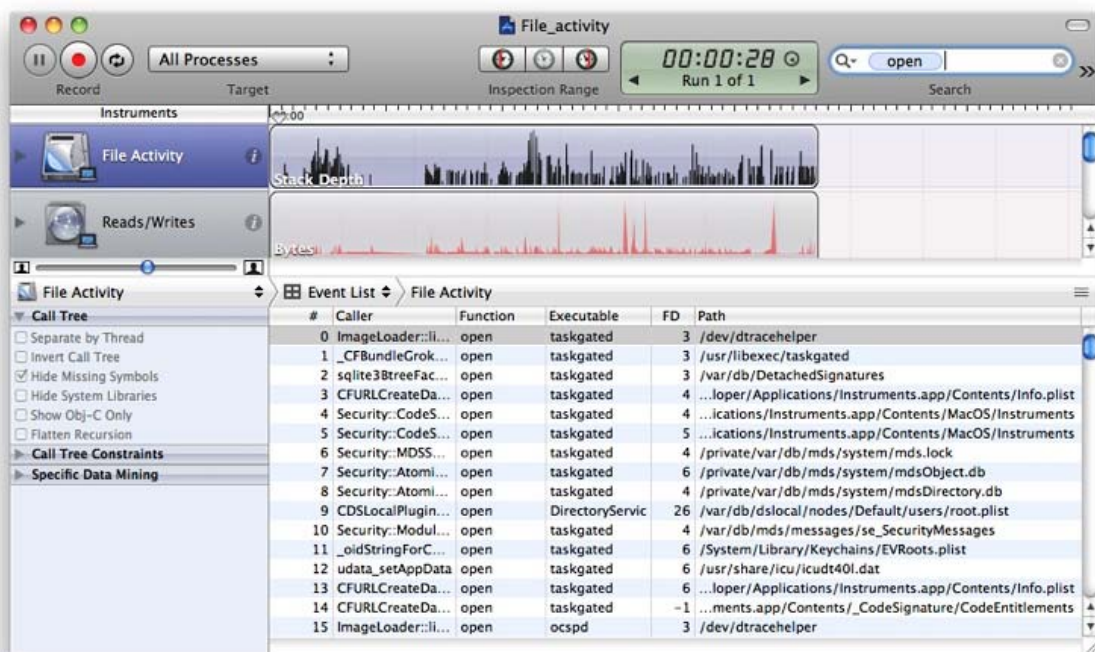
对于水平方向的缩放, Instruments 应用扩大和合并跟踪面板周围的播放头(playhead)的位置。如果你在缩放之前把播放头设置在特定的位置点, 你可以在该位置点下缩小和放大数据。

5.1.2 详细面板

当你在跟踪面板里面确定潜在问题区域后，你可以使用详细面板来核查该区域的数据。详细面板显示了和当前所选择的 instrument 工具的跟踪运行相关的数据。Instruments 应用在详细面板只显示一个 instrument 工具相关信息，所有你必须通过选择不同的 instruments 工具来查看不同的详细信息。

在详细面板里面不同的 instruments 工具显示不同类型的数据。如 5-5 显示了和 File Activity instrument 相关的详细面板，它记录了和指定文件系统例程相关的信息。这里面的详细面板显示了文件系统例程调用的方法和函数，使用的文件的描述，和访问的文件的路径。关于每个 instrument 工具在详细面板显示的信息，参阅“内置 instruments 工具”。

Figure 5-5 The Detail pane



你可以做以下任一事情来打开和关闭详细面板：

- 选择 View > Detail
- 单击工具栏上面的 Detail View 按钮

修改详细面板的显示样式

对于部分 instruments 工具，你可以在详细面板里面使用多于一个格式显示数据。

详细面板支持以下的格式模式：

- **列表模式 (Table mode)**，显示一系列扁平的汇总样本数据列表。
- **大纲模式 (Outline mode)**，显示样本的层级结构，通常使用堆栈跟踪或进程信息来组织。
- **图形模式 (Diagram mode)**，显示一系列独立的样本数据。

为了使用任意的格式来查看 instrument 工具的数据，选择导航栏最右边的合适的模式。instrument 工具是否支持该模式取决于通过该 instrument 工具收集的数据类型。比如，Sampler instrument 工具就不支持**图形模式**，但是 Activity Monitor instrument 工具就支持所有的模式。

但是使用大纲模式显示数据的时候，你可以使用相应行的左边的扩展三角形来向下扩展相应的层级结构。单击一个扩展三角形会扩展阅读或者关闭给定的行。为了扩展行和它所有的子行，按下 Option 键并同时单击扩展三角形。

在详细面板里面排序

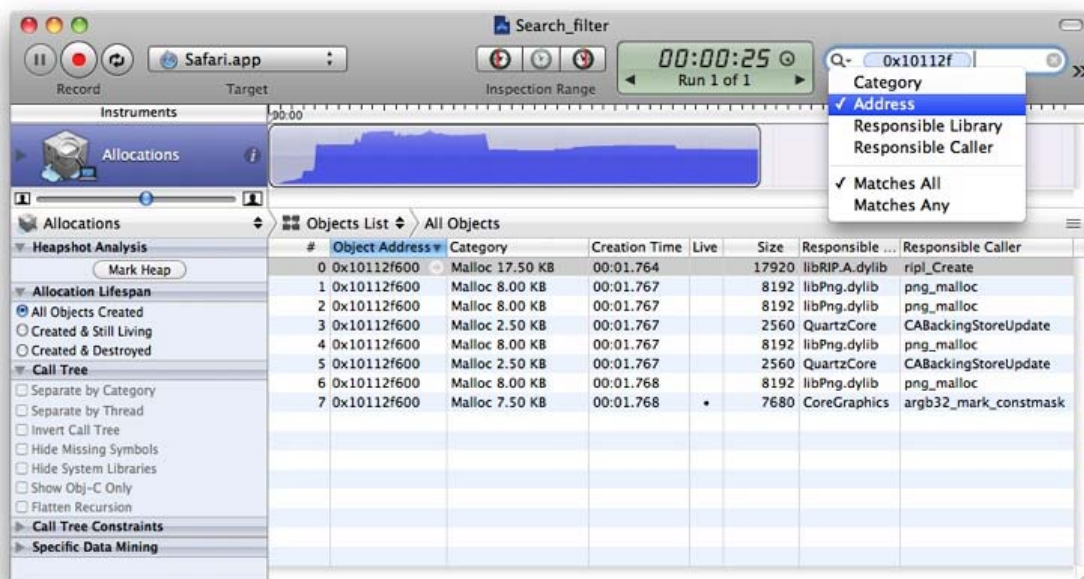
你可以根据特定列的数据来给详细面板显示的信息进行排序。为了达到这个目的，单击相应列的头。详细面板里面的列根据不同的 instrument 工具而不同。

在详细面板里面搜索

你可以在工具栏上面的搜索域里面输入要搜索的字符串来缩小在详细面板显示的信息的范围。默认情况下，Instruments 应用在 instrument 工具记录的所有数据里面搜索给定的字符串。然而使用部分 instrument 工具，你甚至可以提炼搜索的范围。比如，使用 Allocations instrument 可以在 instrument 数据子集里面搜索给定的字符串，比如创建内存块的库或例程。你甚至可以在特定的内存地址里面搜索对象。

范围过滤器根据不同的 Instrument 工具而不同。为了指定特定的目标区域，单击搜索域的放大镜按钮，并从可选项中选择范围。图 5-6 显示了 Allocations instrument 的搜索可选项。

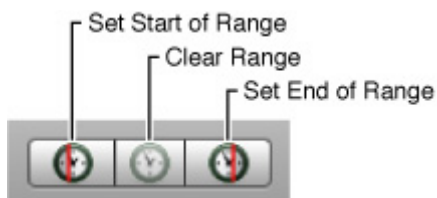
Figure 5-6 Filtering the Detail pane



查看时间范围的数据

虽然在跟踪面板里面放大一个特定的事件可以让你查看在具体时间里面发生了上面事情，但是你也可能对在某个事件范围没收集的数据感兴趣。你可以使用 Inspection Range 控件（如图 5-7 所示）来聚焦在某个特定的时间范围收集的数据。

Figure 5-7 Inspection Range control



可以执行以下的操作来标示一个时间范围：

1. 设置范围的开始
 - A. 在跟踪面板里面拖动播放头到预定开始的时间点
 - B. 单击 Inspection Range 控件里面最左边的按钮
2. 设置范围的结束
 - C. 拖动跟踪面板里面的播放头到预定的结束时间点
 - D. 单击 Inspection Range 控件里面最右边的按钮

Instruments 应用会在跟踪面板里面突出你指定范围的内容。当你设置了一个开始时间，Instruments 应用会自动的选择从开始的时间点到当前跟踪运行的点之间。如果你开始就设置了结束时间点，Instruments 应用会选择从跟踪运行的开始点到你

指定点之间。

你也可以在跟踪面板里面预设的 instrument 工具上通过按住 Option 键并单击和拖动核查范围来设置它的区域。单击并拖动鼠标活跃的 instrument 工具（如果它尚未活跃），并根据 instrument 工具来使用鼠标按下和鼠标放开来设置范围。

当你设定了一个时间范围，Instruments 应用会过滤详细面板的内容，显示只有符合指定范围的收集数据。你可以快速的缩小由 Instruments 应用收集信息的范围并只查看这些在特定周期发生的事件。

为了清空核查范围，单击 Inspection Range 控件里面中间的按钮。

5.1.3 扩展详细面板

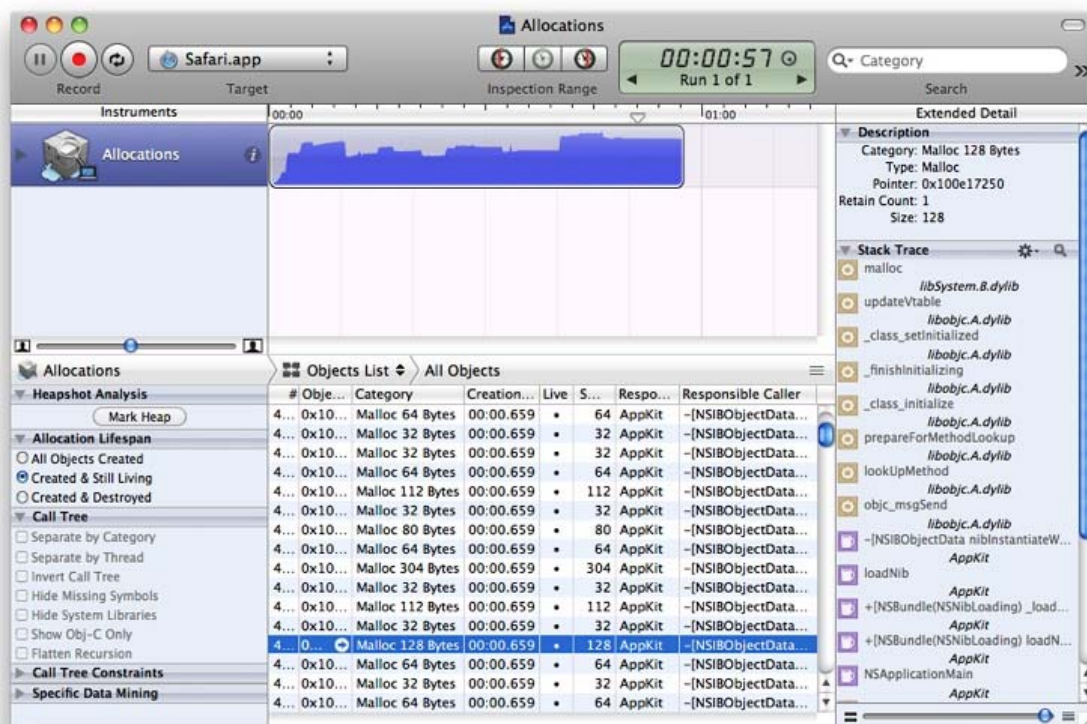
对于部分 instruments 工具，扩展详细面板显示了当前在详细面板里面所选择的项目的额外信息。你可以执行以下任一操作来打开和关闭扩展详细面板：

- 选择 View > Extended Detail。
- 单击工具栏下面的 Extended Detail View 按钮。

扩展详细面板通常包括一个被记录的探头（probe）或者事件的描述，一个堆栈跟踪，和信息被记录的时间。然而，并不是所有的 instruments 工具都会显示这些信息。部分 instruments 工具可能并未提供任何扩展详细信息，而其他可能在该面板上面提供其他信息。关于具体 instrument 工具在该面板显示什么信息，查看“内置 instruments 工具”里面的 instrument 工具描述。

图 5-8 显示了 Allocations instrument 的扩展详细面板。在该例中，Instrument 应用显示关于分配内存的类型的信息，包括它的类型，指针信息，和大小。

Figure 5-8 Extended Detail pane



你可以使用该区域顶部的 Action menu 来配置在堆栈跟踪里面显示的信息。点击并按住 Action menu 的图标将会显示一个菜单，在该菜单里面你可以启用和禁用相应表 5-1 中选项。

Table 5-1 Action menu options

Action	Description
Invert Stack	Toggles the order in which calls are listed in the stack trace.
Source Location	Displays the source file that defines each symbol whose source you own.
Library Name	Displays the name of the library containing each symbol.
Frame #	Displays the number associated with each frame in the stack trace.
File Icon	Displays an icon representing the file in which each symbol is defined.
Trace Call Duration	Creates a new Instruments instrument that traces the selected symbol and places that instrument in the Instruments pane.
Look up API Documentation	Opens the Xcode Documentation window and brings up documentation, if available, for the selected symbol.
Copy Selected Frames	Copies the stack trace information for the selected frames to the pasteboard so that you can paste it into other applications.

如你有一个 Xcode 项目，它的源码字符被列举在堆栈跟踪上面，你可以单击一个字符的名称来在扩展详细面板里面显示相关的源码。该源码被标注为和性能有关。

堆栈跟踪可以通过使用扩展详细面板下面的滑块来折叠显示。这就是所谓的过滤的回溯压缩。该特性的目的是减少堆栈跟踪的详细信息，并显示重要部分。

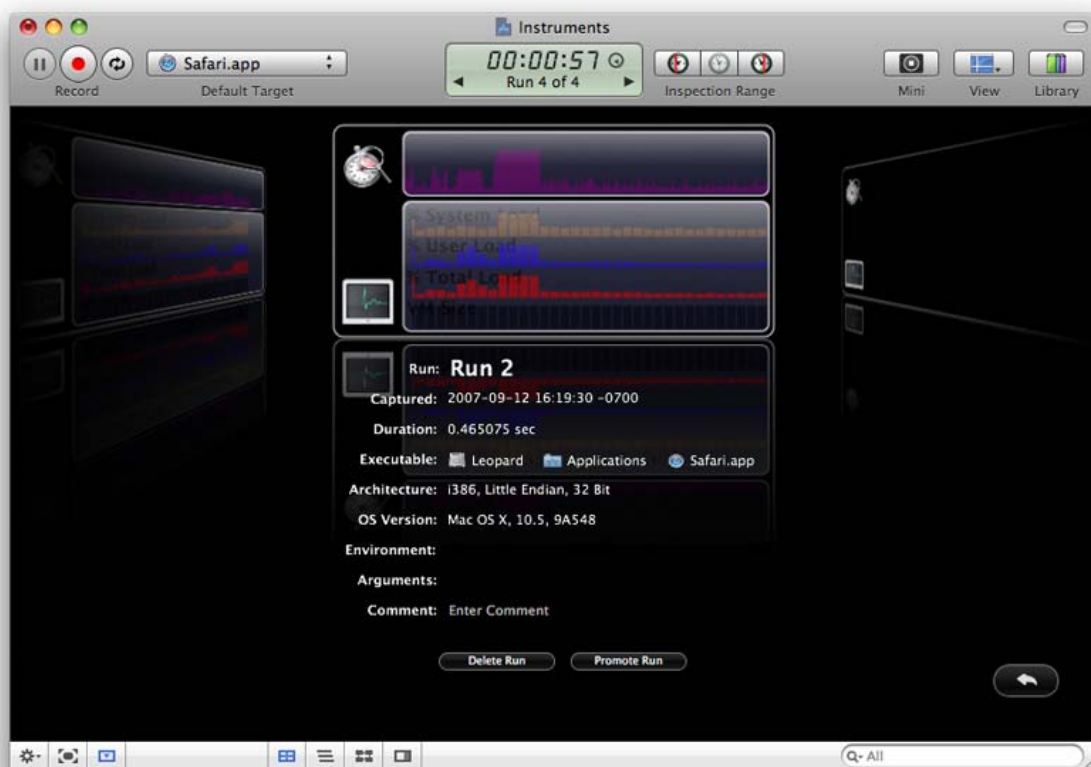
5.1.4 运行浏览器

运行浏览器是一个快速查看和管理之前运行的方法。如果你的跟踪文档包含数个运行，你可以使用该模式在列表的顶部扫描你想要并促进它的运行。你也可以删除这些运行，并给当前视图的运行添加注释。

选择 View > Run Browser 来打开运行浏览器。

运行浏览器显示了 Instruments 应用里面的 Run Browser 视图。单击所选视图的任何一边会让新的视图滚动进入聚焦。为了给视图添加一个注释，双击文本框里面的 Comment 域来让他进入编辑状态。为了退出一个运行浏览器视图，点击窗口底部右边的返回箭头按钮。

Figure 5-9 The Run Browser



5.2 分析技术

收集和查看跟踪数据很简单。但是分析数据并定位潜在的问题就比较困难。后者的任务把性能调优和调试的艺术变为现实。意识到在性能工具收集大量数据可能是艰巨的，即使是经验丰富的开发人员。这有合适的工具（如何知道如何使用它们）让它与众不同。Instruments 应用的 instruments 工具提供了很多组织和过滤跟踪数据的不同选项。以下部分描述了其中特定的 instruments 工具部分的行为和选项，和如何使用这些 instruments 工具来标示你代码中的问题。

5.2.1 使用Sampler Instrument分析数据

Sampler instrument 工具是一个在运行的程序上面执行统计分析的工具。执行样本统计包括在定期停止应用程序并记录在那个一刻执行的信息。对于每个线程，Sampler instrument 工具记录当前在堆栈上面执行的函数和方法，包括拥有函数或方法的名称和拥有它的模块。收集数据之后，它从独立的样本里面合并堆栈调用信息形成主应用程序的调用树。该树显示了所有采样期间执行路径和每个被采样了多少次。

样本统计的优点是它是一个轻量级的并很方便的方式来发现你应用程序在一个周期里面所做的工作。该技术可以在任何没有安装特别的插件代码的应用程序里面使用，而且它通常提供了一个合理准确了解你应用程序的运行行为的图片。

然而样本统计的缺点是它没办法给你的应用程序所做的事情提供一个 100%的图片。因为它只是定期的对调用堆栈做一个快照而已，Sampler instrument 工具并没有精确的记录函数和方法执行的历史。典型的采样间隔 1 到 10 毫秒，在这样本里面有可能多个函数和方法已经被调用。尽管这些图片看似不准确，在采集的样本足够多的时候它同样对大部分程序有效果。随着时间的推移，越多样本被采集，采样间隔越容易造成的扭曲。因此，统计样本依然是一个很好的方式来快速收集你应用程序的信息而无需太多的精力。

注意：*Sampler instrument 替代了 Sampler 程序，而样本程序在 Mac OS X v10.5 及其之后不可用。*

分析堆栈调用的数据

Sampler instrument 工具的主要目的是显示程序在该地方的地消耗的时间。它通过给你显示什么函数被调用并且被调用的次数来达到该目的。

Sampler 工具开始的地方是跟踪面板。默认情况下，跟踪面板中的图形显示应用程序所有线程的最大调用堆栈深度（你也可以改变该图形来替代显示 CPU 负载）。因为它提供了可视化的方法，跟踪面板可以帮你发现代码中的趋势。重复模式可以显示正在执行路径的相似代码。不同部分的图形也可以显示它们当前正在做的事情。通过单击感兴趣的区域，你可以开始使用详细面板分析数据。

你可以在详细面板里面对你的代码执行更深入的分析。详细面板支持列表和大概的视图模式。列表模式显示的是时间顺序的样本。你可以使用该列表来查看你应用程序中代码在给定的地方做了什么。在大纲模式下，显示给你看的是有调用堆栈组织的样本数据。在该视图下，你可以查看每个线程的执行分支确认那个比其他拥有更多数量的样本。扩展每个线程可以让你看到的独立的方法和函数，以及它们收集的样本数据。为了一次性扩展整个层级结构，按下 Option 键并同时点击扩展三角形。

除了在线程的主入口点启动和查找重要的分支，你也可以反转调用树来在叶子节点启动，并查看那个方法和函数被调用次数最多。反转调用树可以帮助你快速的识别最可能频繁使用的方法和函数。你可以在这些叶子节点里面扩展调用树来查找谁调用了该方法或函数，并且调用的次数。当你操作这些的时候，这会帮助显示给定方法或函数运行时所花费的毫秒数，以便你可以关联样本的数量和函数使用的实际时间。

注意：尽管 Sampler instrument 报告给定分支所花费的时间，但是这些时间都是近似值。你可以使用这些值来确定程序所在地方花费的时间，但是不能拿这些值来作为性能指标来衡量你代码执行的快慢。

扩展详细面板给你提供了样本数据的额外视图。在列表模式下，该面板显示了所选择的样本收集的数据，包括样本被采集的时候所有运行线程的堆栈跟踪。在大纲模式下，它显示了包含所选择的方法和函数的堆栈跟踪的深度。

刷选详细面板的内容

表 5-2 列出了你可以应用到由 Sampler instrument 收集的样本上的高级配置选项。你使用这些选项来集中分析在特定时间发生的事件或涉及你代码特定部分的事

件。

Table 5-2 Configuration options for the Sampler instrument

Configuration section	Description
Sample Perspective	Choose between displaying all samples that were captured or only those that were captured while the specified thread (or threads) were running. Viewing all samples can give you a complete picture of the behavior of a thread over a period of time, including how much time was spent blocked or waiting for data. Viewing only the running samples provides an approximate picture of how much time was spent actually executing your code. (The actual running time may differ somewhat from the time reported by Instruments so you should use the reported values only as a rough guide.)
Call Tree	Choose these options to flatten or hide uninteresting parts of the call tree. You can separate out symbols that were gathered from different threads of execution, hide missing symbols or libraries, flatten branches of the call tree that contain recursive calls, and more. These options help you trim irrelevant portions of the call tree and organize the remaining data in ways that make it easier to spot trends.
Call Tree Constraints	Choose the constraints for the data you want to view. You can use these configuration fields to prune the current data set. The Sampler instrument supports constraining data based on the number of samples gathered in a branch or the number of milliseconds spent executing a branch.
Active Thread	Choose the thread you want to analyze. Focusing on a specific thread displays only the samples for that thread, making it easier to see what the thread was doing. Viewing all threads lets you see all of the work being performed by your application.

Call Tree 的配置选项提供了几个方法来在不删除任何样本数据的前提下删减调用树。当你隐藏或夷平一组符号的时候，Sampler 会在样本里面同样隐藏这些调用函数或方法的符号。这样可以让你移除任何你不可控的代码，并集中于你自己的代码并查看他执行所消耗的时间。

另外，Call Tree Constraints 实际上从视图里面移除了一组样本来让你集中于你的满足特定条件的代码路径。例如，你可能使用一个基于时间的约束来集中于运行时间至少消耗 100 毫秒的代码路径。

当你在一个 instrument 工具上面应用这些配置选项的时候，不要忘记你也可以约束样本被收集的时候相应的样本数据。每个跟踪文档里面的 Inspection Range 控制器可以让你查看特定样本点的数据。该特性和其他 instrument 工具的其他配置选项组合使用。关于更多介绍如何使用 Inspection Range 控制器的信息，参阅“查看一个时间范围的数据”部分。

5.2.2 使用Allocations Instrument工具分析数据

Allocations instrument 工具是一个跟踪所有由应用程序分配的内存的工具。所以你可以使用那些信息来识别在你应用程序里面的内存分配模式，并识别你的应用程

序内存效率低下的地方。Allocations instrument 提供了和之前应用程序的 ObjectAlloc 一样甚至更好的数据修整和修剪设施。因为这整合在 Instruments 应用环境里面，你也可以使用该 instrument 工具来关联你应用程序的内存行为到其他类型的行为。

因为它跟踪整个应用程序生命周期的内存分配，你必须从 Instruments 应用里面加载你的程序以便 Allocations instrument 工具可以收集它所需要的数据。在加载的时候，Allocations instrument 使用系统中已有的挂钩(hooks)来记录与你应用程序中分配和释放事件相关的信息，无论这些事件起源于系统标准分配入口还是你自己自定义的分配库。随着数据流的到来，instrument 更新并实时的向你显示内存是如何被分配的。

Allocations instrument 可以工作在使用标准分配函数(如 malloc, calloc 或 free)的应用程序，而且也也可以工作在使用垃圾回收(garbage collected)的应用程序上面。后一种情况，收集器依然调用 free 来释放 GC-aware (GC 感知)内存。Allocations instrument 同样可以在构建在分配内存之上的例程里面工作，包括 Core Foundation 和 Cocoa 的内存分配例程。

注意: Allocations instrument 替代了 ObjectAlloc 应用程序，而 ObjectAlloc 应用程序在 Mac OS X v10.5 及其之后不可用。

分析对象分配的数据

Allocation instrument 工具的目的是为你显示你的应用程序如何使用内存。内存是系统重要的资源，你应该明智的使用它。每个内存分配都包含了直接成本和潜在的长期成本。直接成本就是它分配内存所消耗的时间，包括创建新的虚拟内存页面和把它们映射到物理内存上面。它也有可能包括把陈旧的内存页面写入硬盘。而从长远来看，保持块状的物理内存可能触发系统额外的页面，这和其他页面操作一样可能对系统性能损耗很大。

和所有工具一样，Allocation 工具开始的地方也是跟踪面板。默认配置下，跟踪面板图形化你当前应用程序使用内存数量的净额。使用 instrument 的检查器，你可以修改视图让它显示分配的密度，即内存分配发生的地方，或你也可以让它显示堆栈的深度。分配密度图可以让你查看在你程序里面内存分配发生的频率。分配密度的尖

峰意味着潜在的瓶颈，而你可以通过预先分配块或减少对其他块的依赖来减缓该情况。

无论你使用检查器的任何显示选项，跟踪面板默认情况下都会显示所有类型对象的分配情况。为了集中于特定内存分配的子集，你可以使用详细面板来配置你想要在跟踪面板图形显示的对象。为了集中于特定的对象类型或块大小，打开详细面板并把它设置为列表模式（table mode）。在该模式下，详细面板通过对象类型和大小对内存分配进行排序。图形的列包含了复选框可以让你想要选择图形化的对象。取消所有 Allocations 的复选框（默认情况下就是这样），然后选择其他对象类型的复选框来相应更新跟踪面板。如果你勾选了多个复选框，Allocations instrument 会为图形生成不同颜色的层。

详细面板（列表模式下）显示其他有用的信息来帮你发现潜在的分配问题。列表中整体分配的净分配的列显示了当前活动对象和它有史以来创建数量的直方图。随着净分配占整体分配比例缩小，直方图的条形颜色会跟着改变。蓝色的直方图条形代表了合理的比例，而当改变为红色时意味着比例降低，可能需要核查一下。

尽管列表模式对于你获得内存分配的全局图非常有帮助，但是详细面板集合了三个视图模式的优点。表 5-3 描述了每个模式的显示信息和你如何使用这些模式来发现问题。

Table 5-3 Analyzing data in the Detail pane

Mode	Description
Table	Use this mode to see the summary of net versus overall allocations and to choose which objects you want to graph in the Track pane. Allocations in this mode are grouped by size or object type initially. Clicking the follow link button next to an object type takes you a level deeper by showing you the individual allocation events for that object. Clicking the follow link button again shows you the history of events that occurred at the same memory address.
Outline	Use this mode to see the call trees associated with allocated objects. Clicking the follow link button next to an object type focuses on the call trees associated solely with that object type.
Diagram	Use this mode to see all objects in the order in which they were allocated. Clicking the follow link button next to the object address shows the allocation events associated with that memory address.

Allocations instrument 的扩展详细面板主要显示了所选择的分配事件的堆栈信息。对于某些分配事件，该面板也显示了关于事件的描述，包括事件的类型和大小，和对对象的引用数（retain count）。该信息可以帮你在你的代码中定位事件。

跟踪引用数的事件

当你把 Allocations instrument 添加到你的文档里面的时候，它的初始化配置是只记录内存的分配和释放的事件。默认情况下，它不会记录引用数的事件，比如 CFRetain 和 CFRelease 的调用。原因是记录引用数的事件会给从进程里面收集数据增加了额外的开销，而且在大部情况是不需要的。然而如果你的代码发生内存泄露，你可能想要配置 Allocations instrument 来记录这些事件作为你努力追查泄露的一部分。特别是，你可以查看任何不匹配的 retain 和 release 的事件来查看一个对象是否最后引用被删除了而它仍然保留着。

注意：随着“Record referenc counts”选项被设置，Allocations instrument 在模板文档里面查找内存泄露地方，这会帮助你追踪到内存的泄露。

过滤详细面板的内容

表 5-4 列出了高级的配置选项，你可以应用它们到 Allocations instrument 记录的事件上面。你使用这些选项来集中于分析特定时间发生的事件，或涉及你代码的特定部分。所有的这些选项仅当使用大纲或图形模式查看数据的时候才可用。在列表模式下，Allocations instrument 显示所有分配的历史记录。

Table 5-4 Configuration options for the Allocations instrument

Configuration section	Description
Allocation Lifespan	Choose between displaying all allocation events and those associated with objects that still exist.
Call Tree	Choose these options to flatten or hide uninteresting parts of the call tree. You can separate out memory blocks based on which thread allocated them, hide allocations made by system libraries, show allocations made from Objective-C calls only , and more. These options help you trim irrelevant portions of the call tree and organize the remaining data in ways that make it easier to spot trends.
Call Tree Constraints	Choose the constraints for the data you want to view. You can use these configuration fields to prune the current data set. The Allocations instrument supports constraining data based on the number of allocations made for a given type or the size (in bytes) of the allocations.

当你应用一个 instrument 的配置选项的时候，不要忘记你也可以限制样本数据基于这些样本何时被收集。每个跟踪文档里面的 Inspection Range 控件可以让你查看特定样本点的数据。该特性和其他 instrument 的配置选项组合使用。关于更多如何使用 Inspection Range 控件的信息，参阅“查看一个时间范围的数据”部分。

5.2.3 查找内存泄露

Leaks instrument 提供了和使用 leaks 命令行工具一样的检测泄露的能力。这个 instrument 分析你程序代码里面不在被引用而又正在使用的内存块。不被引用的内存块也被视为“leaks”，因为它们不能再被你的应用程序释放，而且一直占用内存空间直到程序退出。

为了你应用程序消除内存泄露是提高你程序可靠性的重要一步。这对于设计为长时间运行的程序尤为正确。泄露会提高你程序总的内存占用空间，这会引发分页。程序如果持续的发生内存泄露有可能无法完成它们的任何操作，因为它们无法分配必须的内存。在极端情况下，程序有可能受损以至于崩溃。

Leaks instrument 记录你程序中所有发生分配内存的事件，而且周期性的搜索程序可写内存，寄存器，和任何活跃内存块的栈引用。如果在这些地方找到一个没有对于引用的内存块，它会告知缓冲区发生了一个泄露，并在详细面板里面显示相关的信息。

在详细面板里面，你可以使用列表和大纲模式来查看泄露的内存块。在列表模式下，Instruments 应用显示了泄露块的完全列表，它按照大小排序。点击内存地址旁边的以下的 Link 按钮，显示在该地址的内存块分配的历史，最终分配事件中没有匹配的自由事件到显示最多。选择其中任何一个分配事件，将会在扩展详细面板上面显示堆栈跟踪和关于该内存块的通用信息。在大纲模式下，Leaks instrument 显示了由调用树组织的泄露。你可以使用该模式来找出在你代码中特定的分支有多少的内存泄露。选择一个分支会在扩展详细面板里面显示该分支的代码路径深度。

表 5-5 列出了 Leaks instrument 的配置选项。这些选项的大部分会影响查看泄露的相关信息，而部分会影响泄露缓冲区是如何被报告的。

Table 5-5 Configuration options for the Leaks instrument

Configuration option	Description
Leaks Configuration	Use the available option to enable automatic leak detection and to gather the contents of leaked memory blocks when a leak occurs.
Sampling Options	Use the specified field to set the frequency of automatic leak-detection checks.
Leaks Status	Displays the time until the next automatic leak-detection pass.
Check Manually	Use the provided button to initiate a check for memory leaks.

Call Tree	Choose these options to flatten or hide uninteresting parts of the call tree. You can hide allocations made by system libraries, show allocations made from Objective-C calls only, and more. These options help you manage the size of the call tree and organize it in ways that make it easier to spot trends.
-----------	---

关于 leaks 命令行工具的信息，参阅 leaks 主页。

5.2.4 分析Core Data应用程序

对于使用了 Core Data 来管理它们底层数据模型的应用程序，Instruments 应用提供了几个和 Core Data 相关的 instruments 工具来分析潜在的性能问题。这些 instruments 工具可以让你洞察 Core Data 背后发生的事情，帮你识别你程序中没有提取或有效保存数据的地方。表 5-6 列出了提供的 instruments 工具和如何使用没一个。

Table 5-6 Core Data instrument usage

Instrument	Description
Core Data Saves	Use this instrument to find a balance between saving data too often and not saving it enough. Saving too often can lead to I/O overhead as your program writes data frequently to the disk. Conversely, saving infrequently can increase the application's memory overhead and lead to paging.
Core Data Fetches	Use this instrument to optimize the data your application reads from disk. Fetch operations that take a long time might be improved by adding more specific predicates to retrieve only the data needed at that moment. Alternatively, if you notice gaps of inactivity followed by a large number of fetch requests, you might want to use those gaps to prefetch data that you know will be needed later.
Core Data Faulting	Use this instrument to track the lazy initialization of an <code>NSManagedObject</code> or its to-many relationship. Object faults can be mitigated by prefetching the object itself or the objects to which it is related.
Core Data Cache Misses	Use this instrument to locate potential performance issues caused by cache misses. Data not found in the caches must be fetched from the disk. Prefetching objects during relatively quiet periods can help mitigate cache misses by ensuring the required objects are already in memory.

使用 Core Data 模板新建的跟踪文档包含 Core Data Fetches，Core Data Cache Misses，和 Core Data Save instruments 工具。当你分析你的 Core Data 应用程序的时候推荐使用该模板。

关于更多调节 Core Data 应用程序的信息，参阅 Core Data Programming Guide。

第六章 保存和导入跟踪数据

Instruments 提供了几个保存 instrument 工具和跟踪数据的方法。对于一个给定的 Instruments 应用的文档窗口，你可以保存你已经记录的文档的跟踪数据，或者你可以保存文档的 instrument 工具的配置信息。保存跟踪数据可以让你维护一个你应用程序性能随着时间变化的记录。保存文档配置信息避免你每次运行 Instruments 应用都需要重新创建一个通用的配置。

以下部分解析了如何保存一个跟踪文档，并导出其他格式的跟踪数据给其他应用使用。

6.1 保存跟踪文档

在开发周期里面，你可能需要通过在你的程序上面运行一组固定的 instruments 工具来收集多个点的数据。以其每次运行 Instruments 应用的时候重复配置一组相同的 instruments 工具，你可以一次性的配置跟踪文档，然后保存它的跟踪模板。选择 File > Save As Template 来保存你文档当前使用的 instruments 工具和配置（包括任何用户界面轨迹）为一个模板。

跟踪模板的文档和你新建一个文档时出现的 Instruments 模板不一样。你打开跟踪模板和打开其他 instruments 文档的方式相同，都通过选择 File > Open。当你打开一个跟踪模板时，Instruments 应用会使用该模板配置来创建一个没有任何数据的跟踪文档。

Xcode 支持使用自定义的跟踪模板来启动你的应用程序。为了把你的跟踪模板添加到 Xcode 的 Run 菜单，找到本地系统下/Users/<username>/Library/Application Support/Instruments/Templates 目录下面的模板。选择 Run > Start with Performance Tool 菜单来打开它。

6.2 导出跟踪数据

Instruments 应用可以让你把跟踪数据导出为 CSV 的文件格式。该文件格式被大部分应用程序支持。比如，你可能保存你的跟踪数据为该文件格式以便你可以把它导入到电子表格的应用程序。

为了保存你的跟踪数据为 CSV 文件，选项 Instruments 应用的 Instrument > Export Data for: <Instrument Name>。Instruments 应用将会导出该文档近期运行的数据。

注意：并非所有的 *instruments* 工具都支持导出为 CSV 的文件。

6.3 从Sample工具中导入数据

如果你使用 sample 的命令行工具来对你的程序执行进行统计分析。你可以导入你的样本数据并使用 Instruments 应用来查看它们。从 sample 工具中导入数据会新建一个新的使用 Sampler instrument 的跟踪文档，并把数据加载到详细面板。因为样本并没有包含时间戳信息，所有你只能在详细面板里面使用大纲模式来查看数据。虽然你可以应用 Call Tree 配置选项到你的 Sampler instrument 来修整 (trim) 样本数据，但是你不可以使用 Call Tree Constraints 或 Inspection Range 控件来裁剪 (prune) 你的样本数据。

为了导入数据，选择 File > Import Data。Instruments 应用会提示你选择一个包含样本数据的文本文件。然后它会创建一个基于你所选择的文件的跟踪文档。

6.4 使用DTrace数据

你如果你跟踪一个包含自定义 instruments 工具的文档，你可以导出这些 instruments 工具底层的脚本，并使用 dtrace 命令行工具来运行它们。运行脚本后，你可以重新导入结果数据进 Instruments 应用里面。关于更多如何使用这些的信息，参阅“导出 DTrace 脚本”部分。

第七章 使用DTrace创建自定义instruments工具

Instruments 应用里面内置的 instruments 工具提供了大量关于程序内部工作的信息。然而有时候你可能想要定制收集和你代码更加相关的信息。比如，以其函数每次调用的时候都收集数据，你可能想要设置条件来决定何时收集数据。另外，你可能想要在你的代码里面进行比内置 instruments 工具更深入的研究。在这些情况下，Instruments 应用允许你创建自定义的 instruments 工具。

自定义 instruments 工具使用 DTrace 来实现。DTrace 是最初由 Sun 创建和移植到 Mac OS X v10.5 的一个动态追踪工具。由于 DTrace 深入操作系统内核，所以你可以访问内核本身或你计算机上运行的进程的底层的操作。许多内置的 instruments 工具都是基于 DTrace 的。而且虽然 DTrace 本身就是一个非常强大和复杂的工具，Instruments 应用给你访问这个强大的 DTrace 工具提供了一个套简单的接口，而无需太复杂的操作。

DTrace 并没有移植到了 iOS 上面，所以不能在运行在 iOS 设备上面的应用创建自定义的 instrument 工具。

重要：虽然自定义的 *Instrument Builder* 简化了创建 DTrace 探测器的过程，你仍然要比较熟悉 DTrace，并在创建新的 instrument 时了解它是如何工作的。许多强大的调试和数据收集动作需要你自已写 DTrace 脚本。为了学习 DTrace 和 D 脚本语言(*D script language*)，参阅 *OpenSolaris website* 上面的 *Solaris Dynamic Tracing Guide*。关于更多 *dtrace* 命令行工具的信息，参阅 *dtrace* 主页。

注意：部分 Apple 的应用程序 (*iTunes*, *DVD Player* 和 *Front Row*) 和使用 *QuickTime* 的应用程序为了包含敏感数据不允许使用 DTrace 来收集数据 (无论是临时的还是持久性的)。所以，你不应该在执行整个系统的数据收集时运行这些应用程序。

以下部分介绍了如何创建一个自定义的 instrument 工具，和如何在 Instruments 应用和 *dtrace* 命令行工具里面配合使用这些 instrument 工具。

7.1 关于自定义instruments工具

自定义 instruments 工具使用 DTrace 探针 (**probe**)。探针 (**probe**) 是一个就像是在你的代码里面放置了一个传感器。它响应 DTrace 可以绑定的位置和事件，比如

函数的主入口。当函数执行或事件产生的时候，相关的探针会被触发，且 DTrace 运行任何和探针相关的动作。大部分 DTrace 的动作只是简单的收集关于操作系统的数据和用户程序此刻的行为。然而可以运行一个自定义脚本作为动作的一部分。脚本可以让你使用 DTrace 的特性来微调你收集的数据。

每次遇到探针的时候它都会触发，但是和探针相关的动作不需要在探针每次触发的时候运行。断言 (**predicate**) 是一个可以让你限制探针动作何时运行的条件状态。比如，你可以限制一个探针到指定的进程或用户，或你可以在你 instrument 里面指定的条件为真的时候运行动作。默认情况下，探针不包含任何断言，意味着探针每次触发的时候相关的动作将会运行。然而你可以添加任何数量的断言到探针里面，使用 AND 和 OR 操作符来连接它们构成一个复杂的决策树。

一个 instruments 工具由以下几部分构成：

- 一个**描述部分**，包含了 instrument 的名称、类别和描述
- 一个**或多个探针**，每个包含了和它相关的动作和断言
- 一个**DATA 声明区域**，你可以使用它来声明所有探针共享的全局变量
- 一个**BEGIN 脚本**，它初始化任何全局变量，并执行任何 instrument 所需的启动任务
- 一个**END 脚本**，它执行任何最后清理的动作

所有的 instruments 工具都应该包含起码一个探针和它相关的动作。类似的，所有的 instruments 工具都应该包含一个合适的名称和描述来标识它们。Instruments 应用会在库窗口里面显示你的 instruments 工具的描述。好的描述可以更容易记住该 instruments 工具是做什么的和应该如何使用它。

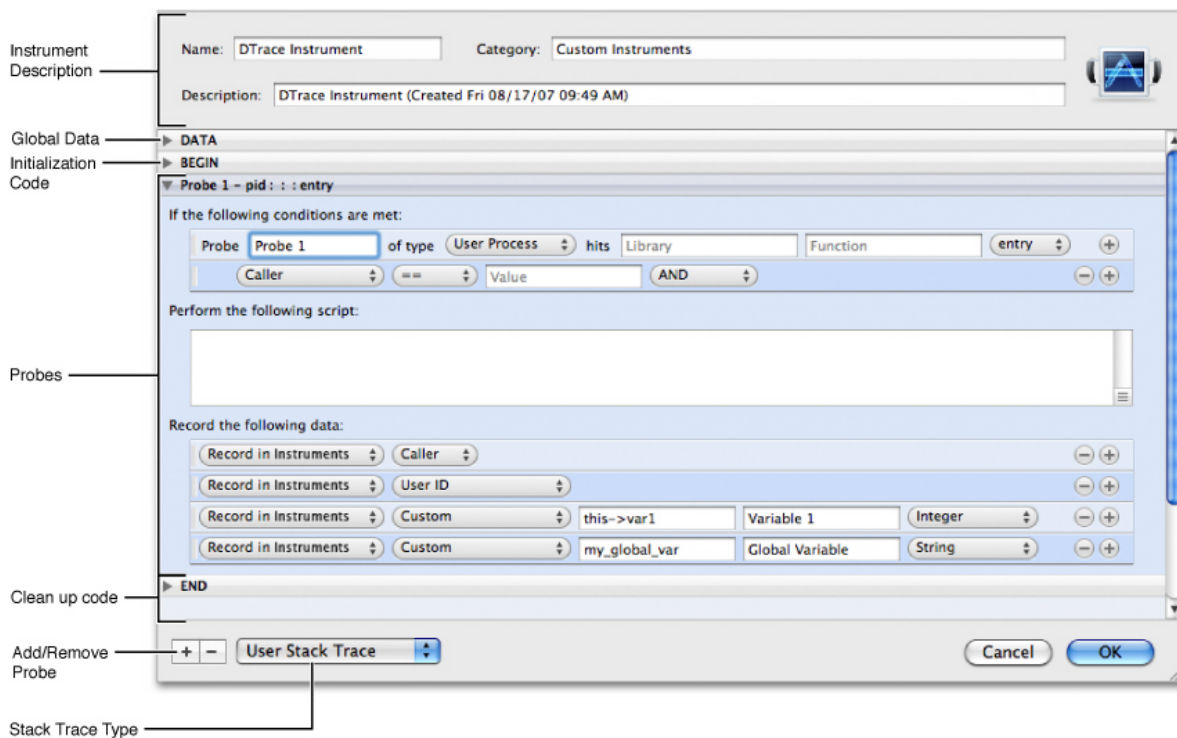
不需要在全局数据或开始和结束脚本里面包含探针。这些元素都是用于设计先进的 instrument 工具，当你相应地在多个探针之间共享数据或给你的 instrument 提供一些有序的初始化配置。创建 DATA, BEGIN 和 END 块会在“写自定义脚本的建议”部分介绍。

7.2 创建自定义的Instrument工具

为了创建一个自定义的 DTrace 的 instrument 工具，选择 Instrument > Build New Instrument（你也可以选择你跟踪文档下面的 Action menu 里面 Add Instrument >

DTrace Instrument 来执行相同的动作)。该命令显示了 instrument 配置表, 如图 7-1 所示。你可以使用该表来指定你的 instrument 工具的信息, 包括任何探针和自定义脚本。

Figure 7-1 The instrument configuration sheet



你最少应该为你创建的每个 instrument 提供以下信息:

- 名称(**Name**), 在库里面你的自定义 instrument 工具的相关名称。
- 类别(**Category**), 是你的 Instrument 工具出现在库里面的类别。你可以指定一个已有的类别, 比如内存(Memory), 或者创建一个你自己的类别。
- 描述(**Description**), 该 Instrument 工具的描述, 用于库窗口里面和 instrument 的帮助标签里面。
- 探针提供者(**Probe provider**), 包括探针的类别和它何时触发的详细信息。通常包含了探针应用的指定方法或函数。关于更多信息, 参阅“指定探针提供者”。
- 探针动作(**Probe action**), 当你探针触发时需要记录的数据或者需要执行的脚本。参阅“添加动作到探针”部分。

一个 instrument 工具应该包含起码一个探针, 且可以包含多个。探针的定义包含了提供者信息, 断言信息, 和动作。所有的探针都必须指定起码一个提供者的信息, 并且几乎所有的探针都定义某种有序动作。探针定义的断言是可选的, 但是它对于让

你的 instrument 工具集中于正确的数据非常有帮助。

7.2.1 添加和删除探针

每个新的 instrument 工具都伴随一个你配置的探针。你可以在 instrument 配置表的底部点击(+)按钮来添加多个探针。

为了从你的 instrument 工具里面删除一个探针，你可以单击选择探针并按下(-)按钮。

当添加探针的时候，最好给探针提供一个描述的名称。默认情况下，Instruments 应用通过使用像“Probe 1”和“Probe 2”这样的名称来枚举探针。

7.2.2 指定探针的提供者

为了指定一个触发探针的位置或事件，你必须关联一个探针到它的合理的提供者。提供者(Providers)是内核模块，它是 DTrace 的代理，提供了创建探针所需要的 instrument 信息。你不需要知道提供者是如何创建 instrument 的，你只需要知道每个提供者的基本能力。表 7-1 列出了在 Instruments 应用创建自定义 instrument 工具所支持的提供者。提供者的列列出在 instrument 配置表所示的名称，而 DTrace 提供者列列出在相应的 DTrace 脚本里面使用的提供者的名称。

Table 7-1 DTrace providers

Provider	DTrace provider	Description
User Process	pid	The probe fires on entry (or return) of the specified function in your code. You must provide the function name and the name of the library that contains it.
Objective-C	objc	The probe fires on entry (or return) of the specified Objective-C method. You must provide the method name and the class to which it belongs.
System Call	syscall	The probe fires on entry (or return) of the specified system library function.
DTrace	DTrace	The probe fires when DTrace itself enters a BEGIN, END, or ERROR block.
Kernel Function Boundaries	fbt	The probe fires on entry (or return) of the specified kernel function in your code. You must provide the kernel function name and the name of the library that contains it.
Mach	mach_trap	The probe fires on entry (or return) of the specified Mach library function.
Profile	profile	The probe fires regularly at the specified time interval on each core of the machine. Profile probes can fire with a granularity that ranges from

		microseconds to days.
Tick	<code>tick</code>	The probe fires at periodic intervals on one core of the machine. Tick probes can fire with a granularity that ranges from microseconds to days. You might use this provider to perform periodic tasks that are not required to be on a particular core.
I/O	<code>io</code>	The probe fires at the start of the specified kernel routine. For a list of functions monitored by this probe, use the <code>dtrace -l</code> command from Terminal to get a list of probe points. You can then search this list for probes monitored by the <code>io</code> module.
Kernel Process	<code>proc</code>	The probe fires on the initiation of one of several kernel-level routines. For a list of functions monitored by this probe, use the <code>dtrace -l</code> command from Terminal to get a list of probe points. You can then search this list for functions monitored by the <code>proc</code> module.
User-Level Synchronization	<code>plockstat</code>	The probe fires at one of several synchronization points. You can use this provider to monitor mutex and read-write lock events.
Core Data	<code>CoreData</code>	The probe fires at one of several Core Data–specific events. For a list of methods monitored by this probe, use the <code>dtrace -l</code> command from Terminal to get a list of probe points. You can then search this list for methods monitored by the <code>CoreData</code> module.
Ruby	<code>ruby</code>	The probe fires at one of several Ruby-specific events.

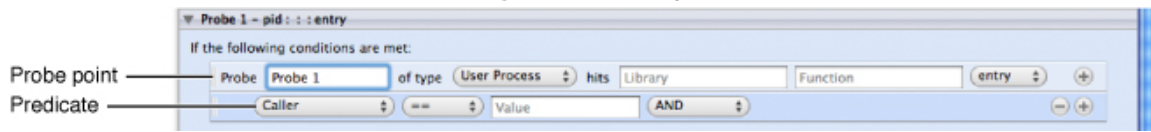
在你为你的探针选定了提供者后，你需要指定探针旁边的相关信息。对于函数级别的探针，它可能需要输入函数或方法的名称和代码模块或包含它的类。部分提供者可能只要简单的从弹出菜单里面选择包含相应的事件即可。

你配置完一个探针后，你可以继续增加额外的断言（来确定它何时触发）或你可以继续为探针定义动作。

7.2.3 给探针添加断言

断言可以让你控制一个探针的动作何时被 Instruments 应用执行。你可以使用它们来防止 Instruments 应用收集那些你不关心或有误的数据。比如，如果你的代码只当堆栈达到一定的深度才表现出不寻常的行为，你可以使用一个断言来指定目标堆栈的最低深度。每次探针触发的时候，Instruments 应用都会判定相关的断言。仅当它们判定为真的时候，DTrace 才会执行相关的动作。

你可以通过点击探针行最后面的(+)的按钮来添加一个断言到探针。Instruments 应用会添加一个断言到探针里面，如图 7-2 所示。你随后可以通过使用探针或断言后面(+)按钮来添加一个新的断言。你可以通过单击断言旁边的(-)按钮来删除一个断言。为了重新排列断言，可以点击断言的右侧侧的空白地方并把它拖拉到新的位置。

Figure 7-2 Adding a predicate

Instruments 应用按照断言从上到下的顺序来判断。你可以使用 AND 和 OR 操作符来连接断言，但是不能把它们分组来创建嵌套的条件块。相反，你必须谨慎的给你的断言排序并保证所有的断言条件都被检查。

断言行的第一个弹出框让你选择作为检测条件一部分的数据。表 7-2 列出了由 DTrace 定义的标准变量，你可以在你的断言或脚本里面使用它们。该变量列列出了它在 instrument 配置面板出现的名称，而 DTrace 变量列列出了变量在相应 DTrace 脚本使用的事件名称。除了标准变量，你可以测试自定义变量和你脚本里面在断言处指定自定义变量类型的常量。

Table 7-2 DTrace variables

Variable	DTrace variable	Description
Caller	caller	The value of the current thread's program counter just before entering the probe. This variable contains an integer value.
Chip	chip	The identifier for the physical chip executing the probe. This is a 0-based integer indicating the index of the current core. For example, a four core machine has cores 0 through 3.
CPU	cpu	The identifier for the CPU executing the probe. This is a 0-based integer indicating the index of the current core. For example, a four core machine has cores 0 through 3.
Current Working Directory	cwd	The current working directory of the current process. This variable contains a string value.
Last Error #	errno	The error value returned by the last system call made on the current thread. This variable contains an integer value.
Executable	execname	The name that was passed to <code>exec</code> to execute the current process. This variable contains a string value.
User ID	uid	The real user ID of the current process. This variable contains an integer value.
Group ID	gid	The real group ID of the current process. This variable contains an integer value.
Process ID	pid	The process ID of the current process. This variable contains an integer value.
Parent ID	ppid	The process ID of the parent process. This variable contains an integer value.
Thread ID	tid	The thread ID of the current thread. This is the same value returned by the <code>pthread_self</code> function.

Interrupt Priority Level	<code>ipl</code>	The interrupt priority level on the current CPU at the time the probe fired. This variable contains an unsigned integer value.
Function	<code>probefunc</code>	The function name part of the probe's description. This variable contains a string value.
Module	<code>probemod</code>	The module name part of the probe's description. This variable contains a string value.
Name	<code>probename</code>	The name portion of the probe's description. This variable contains a string value.
Provider	<code>probeprov</code>	The provider name part of the probe's description. This variable contains a string value.
Root Directory	<code>root</code>	The root directory of the process. This variable contains a string value.
Stack Depth	<code>stackdepth</code>	The stack frame depth of the current thread at the time the thread fired. This variable contains an unsigned integer value.
Relative Timestamp	<code>timestamp</code>	The current value of the system's timestamp counter, measured in nanoseconds. Because this counter increments from an arbitrary point in the past, you should use it to calculate only relative time differences. This variable contains an unsigned 64-bit integer value.
Virtual Timestamp	<code>vtimestamp</code>	The amount of time the current thread has been running, measured in nanoseconds. This value does not include time spent in DTrace predicates and actions. This variable contains an unsigned 64-bit integer value.
Timestamp	<code>walltimestamp/1000</code>	The current number of nanoseconds that have elapsed since 00:00 Universal coordinated Time, January 1, 1970. This variable contains an unsigned 64-bit integer value.
<code>arg0througharg9</code>	<code>arg0 through arg9</code>	The first 10 arguments to the probe represented as raw 64-bit integers. If fewer than ten arguments were passed to the probe, the remaining variables contain the value 0.
Custom	The name of your variable	Use this option to specify a variable or constant from one of your scripts.

除了条件变量，你还必须指定比较运算符和目标的值。

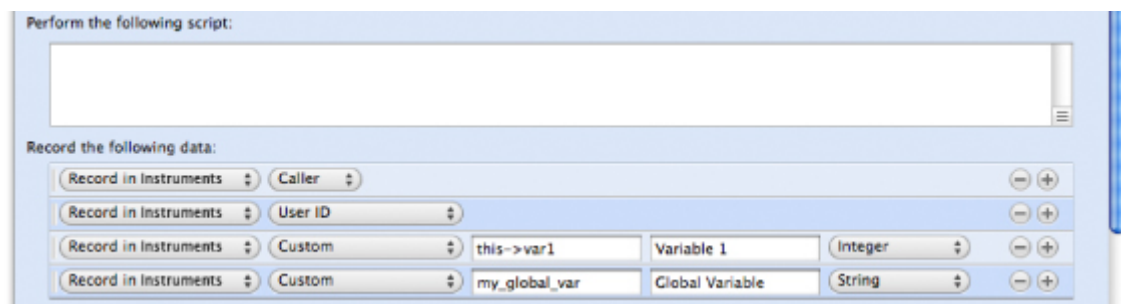
7.2.4 给探针添加动作

当 instrument 工具定义的一个探针点被激活时，并且探针的断言条件判断为真，DTrace 将会运行探针相关的动作。你使用探针的动作来收集数据或执行额外的处理。比如，如果你的探针监视一个特定的函数或方法，你可以让它方法函数的 caller 和任何 Instrument 堆栈跟踪信息。如果你想要一个稍微更高级的动作，你可以使用脚本变量来跟踪函数被调用的次数，并报告该信息。而且如果你想要更加高级的动作，你可以使用内核级别的 DTrace 函数来编写脚本来确定你函数使用的一个锁的状态。

在后面那种情况，你的脚本代码同样可能返回锁当前的拥有者（如果有的话）来帮助你确定代码中不同线程间的交互。

图 7-3 显示了 instrument 配置表的部分内容，在这里你可以指定你探针的动作。该脚本只是简单的包含了一个文本区域来为你脚本代码确定类型（Instruments 应用在把它传递给 DTrace 之前不会验证你的代码，所以你要仔细的检查你的代码）。底下部分包含了你想要在 instruments 工具里面 DTrace 返回的数据的控件。你可以使用弹出菜单来配置一个你想要返回的内置的 DTrace 变量。你也可以在弹出菜单里面选择自定义类型并返回一个你自己的脚本变量。

Figure 7-3 Configuring a probe's action



当你配置你的 instrument 工具返回一个自定义变量时，Instruments 应用会要你提供以下的信息：

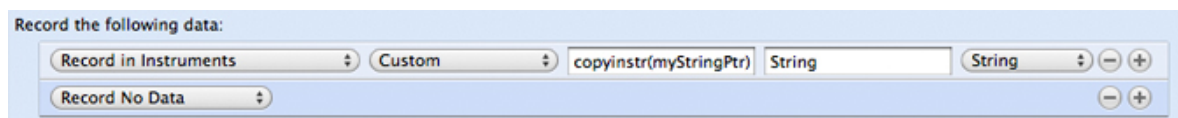
- 包含数据的脚本变量
- 在你的 instrument 工具接口上面给变量赋值的名称
- 变量的类型

你的探针返回给 Instruments 应用的任何数据都会被收集并显示在 instruments 工具的详细面板上面。详细面板显示所有的数据变量，无论它的类型是什么。一旦堆栈信息对一个特定的探针可用，Instruments 应用会在你的 instruments 工具的扩展详细面板上面显示该信息。此外，Instruments 应用会自动的查找由你 Instrument 工具返回的整型数据并把这些类型添加到显示在跟踪面板上面的分析列表里面。

因为 DTrace 脚本运行在内核空间，而 Instruments 应用运行在用户空间，如果你想要返回一个自定义指针类型的脚本变量的值给 Instruments 应用，你必须创建一个缓冲区来存储这些变量的数据。创建缓冲区最简单的方式是使用 DTrace 里面的 copyin 或 copyinstr 子程序。copyinstr 子程序需要一个 C 字符串指针的参数，并返回一个字符串内容形式，你可以把该字符串返回给 Instruments 应用。类似的，copyin

子程序需要一个指针和指针大小的参数，并返回一个缓冲区的数据，你可以在以后使用 `stringof` 关键字格式来把它转换为一个字符串。这两个子程序都是 DTrace 环境的一部分，并且可以被使用在你探针动作定义代码里面的任何部分。比如，为了从 C 风格的指针返回一个字符串，你需要简单的使用 `copyinstr` 子程序来封装该变量，如图 7-4 所示。

Figure 7-4 Returning a string pointer



重要： *Instruments* 应用会自动使用 `copyinstr` 来封装内置的变量（比如 `arg0` 到 `arg9` 的函数参数），前提是该变量类型被设置为 `String`。然后，它不会对你脚本的自定义变量做同样的操作。你负责确保你自定义变量里面的数据匹配该变量指定的类型。

关于 *Instruments* 应用支持的内置变量，参阅表 7-2。关于脚本和脚本变量的更多信息，参阅“编写自定义脚本的提示”部分。关于 DTrace 子程序的更多信息，包括 `copyin` 和 `copyinstr` 子程序，参阅 OpenSolaris 上的 Solaris Dynamic Tracing Guide。

7.2.5 编写自定义脚本的提示

你可以使用 D 脚本语言来编写 DTrace 脚本，它们的语法是 C 编程语言的一个大子集。D 语言结合了 C 语言编程结构的特殊函数和变量来帮助你在程序里面跟踪信息。

以下部分描述了几个在你自定义 *instruments* 工具里面使用脚本的通用形式。然而它们不提供对 D 语言的全面预留，也不提供编写 DTrace 脚本的过程。关于脚本和 D 语言的信息，参阅 OpenSolaris website 上面的 Solaris Dynamic Tracing Guide。

7.2.6 编写 BEGIN 和 END 脚本

如果你想在你的动作触发的时候不只是返回 DTrace 内置变量信息给 *Instruments* 应用，那么你需要编写自定义脚本。脚本可以在内核级别直接和 DTrace 交互，提供访问关于内核和活跃进程的底层信息。大部分 *instruments* 工具使用脚本来收集 DTrace 不容易提供的信息。你也可以使用脚本在数据返回给 *Instruments* 应用时操纵原始数据。比如，如果你想要更容易的和 *instruments* 跟踪面板的其他值比较，你可以使用脚本来格式化一个特定范围的数据值。

在 Instruments 应用里面，自定义 instrument 工具的配置表提供了几个你可以编写 DTrace 脚本的区域：

- DATA 部分包含了任何你想要在 instrument 工具里面使用的全局变量的定义。
- BEGIN 部分包含了你的 instrument 工具里面的代码的初始化。
- 每个探针包含了它的动作的脚本代码。
- END 部分包含了清理你 instrument 工具的任何代码。

所有脚本区域都是可选的。你不需要一定包含脚本初始化代码或清理 instrument 工具的代码（如果你的 instrument 工具不需要的话）。然而如果你的 instrument 工具在它的 DATA 部分定义了全局变量，推荐你提供初始化代码来设置这些全局变量为已知的值。D 语言不允许你在全局变量声明的时候内联的指定值，所有你必须在 BEGIN 部分给它们赋值。比如，DATA 部分可能简单的由一个变量声明构成，如下：

```
int myVariable;
```

相应的 BEGIN 部分可能包含以下对变量初始化的代码：

```
myVariable = 0;
```

如果你相应的探针动作改变了 myVariable 的值，你可能需要在你探针的 END 部分格式化并打印出变量的最终值。

你的大部分脚本都可能和单独的探针相关联。每个探针包含一个和它动作相关的脚本。当是时候执行探针动作的时候，DTrace 首先会运行你的代码，然后返回任何要求的数据给 Instruments 应用。因为传递数据回 Instruments 应用包括从内核空间拷贝数据到 Instruments 应用的程序空间，因此你应该总是通过在“Record the following data:”部分的 instruments 配置表里面配置合适的实体来传递数据返回给 Instruments 应用。在你的脚本代码里面人工返回的变量有可能无法正确的返回给 Instruments 应用。

7.2.7 从自定义脚本里面访问内核数据

因为 DTrace 脚本在系统的内核执行，它们可以访问内核的字符。如果你想要在自定义 instruments 工具里面查看内核全局变量和数据结构，你可以在你的 DTrace 脚本里面执行这些操作。为了访问内核变量，你必须在变量名称前面加一个反引号（```）。反引号字符告知 DTrace 查找当前脚本之外的指定变量。

列表 7-1 显示了一个动作脚本样本，它从内核变量 `avenrun` 检索当前加载信息，使用这些变量来计算系统平均加载的平均事件。如果你打算使用 `Profile provider` 来创建一个探针，你可以使用这个脚本来周期性的收集加载数据，然后在 `Instruments` 应用里面图示化该信息。

Listing 7-1 Accessing kernel variables from a DTrace script

```
this->load1a = `avenrun[0]/1000;

this->load1b = ((`avenrun[0] % 1000) * 100) / 1000;

this->load1 = (100 * this->load1a) + this->load1b;
```

7.2.8 变量作用域

DTrace 脚本有一个基本扁平的结构，因为缺乏流程控制语句，而且其设计为了保证探针的执行时间最小化。然而 DTrace 脚本里面的变量，根据你的需求划分到不同的作用域。表 7-3 列出了这些变量的作用域级别，和在每个级别使用这些变量的语法。

Table 7-3 Variable scope in DTrace scripts

Scope	Syntax example	Description
Global	<code>myGlobal = 1;</code>	Global variables are identified simply using the variable name. All probe actions on all system threads have access to variables in this space.
Thread	<code>self->myThreadVar = 1;</code>	Thread-local variables are dereferenced from the <code>self</code> keyword. All probe actions running on the same thread have access to variables in this space. You might use this scope to collect data over the course of several runs of a probe's action on the current thread.
Probe	<code>this->myLocalVar = 1;</code>	Probe-local variables are dereferenced using the <code>this</code> keyword. Only the current running probe has access to variables in this space. Typically, you use this scope to define temporary variables that you want the kernel to clean up when the current action ends.

7.2.9 查找脚本错误

如果你的自定义 `instruments` 工具的脚本代码包含一个错误，当你使用 `DTrace` 边缘该脚本的时候，`Instruments` 应用会在跟踪面板显示该错误的信息。`Instruments` 应用在你你的跟踪文档里面按下 `Record` 按钮但跟踪又没有真正开始的时候报告你该错误。在错误消息的气泡里面包含了一个编辑按钮。单击该按钮来打开 `instrument` 工具的配置表，它此刻会标识探针的一个错误。

7.3 导出DTrace脚本

尽管 Instruments 应用提供了很方便的接口来收集跟踪数据，但是很多时候直接使用 DTrace 来收集数据会更加方便。例如，如果你是一个系统管理员，或者正在编写自动化测试脚本，你可能更喜欢使用 DTrace 命令行接口来加载一个进程并收集数据。然而，使用命令行工具要求你编写自己的 DTrace 脚本，该脚本可能很费时甚至出现很多错误。如果你已经拥有了一个包含一个或多个基于 DTrace 的 instrument 工具的跟踪文档，你可以使用 Instruments 应用程序来生成一个 DTrace 脚本，该脚本和你在跟踪文档里面的 instruments 工具提供了相同的行为。

Instruments 应用只支持那些所有 instruments 工具都是基于 DTrace 的文档导出 DTrace 脚本。这意味着你的文档可以包含自定义 instruments 工具和少数内置 instruments 工具(比如库窗口中 File System 和 CoreData 组的 instruments 工具)。关于一个 instrument 工具是否是基于 DTrace，请参考“内置 instruments 工具”的介绍。

为了导出一个 DTrace 脚本，选中包含 instruments 工具的跟踪文档，选择 File > DTrace Script Export。该命令为你的 instruments 工具把脚本放入了一个文本文件，它和你使用带有-s 选项的 dtrace 的命令工具创建的文本文件一样。比如，如果你导出一个名为 MyInstrumentsScript.d 的脚本，你可能要在终端运行以下命令：

```
sudo dtrace -s MyInstrumentsScript.d
```

注意：大部分情况下你必须使用超级用户的权限来运行 dtrace，这是为什么 sudo 命令会在上面的例子中被放在 dtrace 的前面。

为你的 Instruments 应用导出你脚本文件（相对应手动编写）的另一个优势是在你运行脚本后，你可以导入结果数据到 Instruments 应用里面，并评审它。从 Instruments 应用导出的脚本会打印一个开始标志(<dtrace_output_begin>)在 dtrace 输出文件的开头。为为了收集数据，简单的拷贝所有 DTrace 在终端输出的信息，并把它粘贴到文本文件里面，或者简单的在 dtrace 命令后面重定向输出到一个文件里面。为了导入 Instruments 应用的数据，选中生成原始脚本的跟踪文档，并选择 File > DTrace Data Import。

第八章 内置instruments工具

Instruments 应用里面内置了许多 instruments 工具。每个 instrument 工具都包含它自己的配置选项和显示信息的方式，及收集合适的数据类型。内置的 instruments 工具按照它收集的数据类型分组为少数几个类别。以下个部分更详细的描述了内置 instruments 工具。

8.1 Core Data Instruments[Core Data相关]

以下的 instruments 工具收集的数据和 Core Data 应用的事件相关。你可以使用这些 instruments 工具返回的信息来评估各种事件对应用性能的影响和来定位潜在问题并修复它。

8.1.1 Core Data Saves

Core Data Saves instrument 工具记录了 Core Data 应用中保存的操作。该 instrument 工具可以在单一进程或所有当前系统运行的进程中执行操作。它只为这些使用 Core Data 的进程记录数据。该 instrument 工具在它的实现上使用了 DTrace，并可以导入一个 DTrace 脚本。

详细面板的样本数据

该 instrument 工具捕获以下信息：

- Caller(调用者) 启动保存操作的方法的名称（包括栈跟踪信息）。
- Save duration(保存耗时) 保存的耗时时间，以微秒为单位。

跟踪面板的显示项

跟踪面板可以被设置来显示以下任一信息数据：

- Stack depth(栈深度) 栈调用的深度。
- Thread ID(线程 ID) 线程标示符。
- Save duration(保存耗时) 保存操作的消耗时间。

扩展详细面板的补充数据

对于详细面板的每个事件，你可以打开对应的扩展详细面板来查看它的栈调用的跟踪和事件发生的对应时间。

8.1.2 Core Data Fetches

Core Data Fetches instrument 工具记录 Core Data 应用中提取保存数据的操作。该 instrument 工具可以运行在单一进程或所有系统当前运行的进程之上。它只会记录使用 Core Data 的进程的数据。该 instrument 工具在实现上使用了 DTrace，并可以导入 DTrace 脚本。

详细面板的样本数据

该 instruments 工具捕获以下信息：

- Caller(调用者) 启动提取操作的方法名（包括栈跟踪信息）。
- Fetch entity(提取的条目) 被提取的条目的名称。
- Fetch count(提取总数) 提取条目的总数。
- Fetch duration(提取耗时) 提取操作消耗的时长，以微秒为单位。

跟踪面板的显式项

可以设置跟踪面板来显式以下任何的数据：

- Stack depth(栈深度) 栈调用的深度。
- Thread ID(线程 ID) 线程标示符。
- Fetch count(提取总数) 提取条目的总数。
- Fetch duration(保存耗时) 提取操作的消耗时间。

扩展详细面板的补充数据

对于详细面板的条目，你可以打开它对于的扩展详细面板来查看调用的栈信息和对应事件发生的时间。

8.1.3 Core Data Faults

Core Data Faults instrument 工具记录 NSManagedObject 或它的一对多关系的延迟初始化过程中发生的故障事件。该 instrument 工具可以运行在单一进程或所有

系统当前运行的进程上面。它只收集使用了 Core Data 的进程的数据。该 instrument 工具的实现使用了 DTrace，并可以导入 DTrace 脚本。

详细面板的样本数据

该 instruments 工具捕获以下信息：

- Caller(调用者) 触发故障的操作的方法名（包括栈跟踪信息）。
- Fault object(故障对象) 引发故障的对象的名称。
- Fault duration(故障耗时) 故障处理例程的执行时长（以微秒为单位）。
- Relationship fault source(相关故障源)
- Relationship(相关关系) 关系名称。
- Relationship fault duration(相关故障的耗时) 相关故障的耗时（以微秒为单位）。

跟踪面板的显式项

可以设置跟踪面板来显式以下任何的数据：

- Stack depth(栈深度) 栈调用的深度。
- Thread ID(线程 ID) 线程标示符。
- Fault duration(故障耗时)
- Relationship fault duration(相关故障的耗时)

扩展详细面板的补充数据

对于详细面板的条目，你可以打开它对于的扩展详细面板来查看调用的栈信息和对应事件发生的时间。

8.1.4 Core Data Cache Misses

Core Data Cache Misses instrument 工具记录高速缓存未命中导致的故障事件。该 instrument 工具可以运行在单一进程或所有系统当前运行的进程上面。它只记录使用了 Core Data 的进程。该 instrument 工具的实现使用了 DTrace，并可以导入 DTrace 脚本。

注意：该 instrument 工具提供了 Core Data Fault instrument 工具提供的行为的一个子集，但是它对你分析整个应用程序的性能更有帮助。

详细面板的样本数据

该 instruments 工具捕获以下信息：

- **Caller(调用者)** 触发高速缓存未命中的方法名（包括栈跟踪信息）。
- **Cache Miss(高速缓存未命中)** 引发高速缓存未命中的对象的名称。
- **CM duration(高速缓存未命中的耗时)** 故障处理例程执行所用时长(以微为单位)。
- **RCM source(相关高速缓存未命中源)** 高速缓存未命中的相关源。
- **RCM Relationship(相关高速缓存未命中关系)** 关系名称。
- **RCM duration (相关高速缓存未命中耗时)** 相关高速缓存未命中的耗时（以微秒为单位）。

跟踪面板的显式项

可以设置跟踪面板来显式以下任何的数据：

- **Stack depth(栈深度)** 栈调用的深度。
- **Thread ID(线程 ID)** 线程标示符。
- **CM duration(高速缓存未命中耗时)** 如上。
- **RCM duration(相关高速缓存未命中耗时)** 如上。

扩展详细面板的补充数据

对于详细面板的条目，你可以打开它对于的扩展详细面板来查看调用的栈信息和对应事件发生的时间。

8.2 Dispatch Instruments[并发相关]

该 instrument 工具收集和 Grand Central Dispatch(GCD) 相关的数据。GCD 是实现并发执行异步任务的技术。GCD 在 Mac OS X v10.6 及其之后可用，在 iOS 上面不可用。

为了高效使用 Dispatch instrument 工具，你需要对 GCD 的队列和 block 对象比较熟悉。关于更多信息，参阅 Concurrency Programming Guide(并发编程指南)。

8.2.1 Dispatch

Dispatch instrument 工具捕获由你应用程序创建的 GCD 队列和在这些队列上执行的 block 对象的信息。它向你展示了应用程序队列和 block 对象执行的行为。它记录队列的生命周期，和跟踪 block 的调用和执行的时间。

Dispatch 帮助你微调 blocks 的执行。它向你展示了那个 block 执行的次数最多和它们占用了多久 CPU 时间来执行。你可以找到这些你已经放入队列执行的 blocks 的热点并优化这些 blocks 的代码。你也可以找到队列中同步执行的 blocks 的情况。（GCD 队列在工作异步执行的时候可以更高效的执行）。

Dispatch 运行在当个进程上面。该 instrument 工具的实现使用了 DTrace，但是 Dispatch 的跟踪数据不能被导入到一个 DTrace 脚本里面。

对于详细面板的条目，你可以打开对于的扩展详细面板来查看栈跟踪的调用信息，和这些事件发生的对应时间。

Dispatch 提供了几种跟踪数据查看的方式。主要的查看模式有**队列视图 (queues view)**、**调用树视图 (call tree view)**、**块视图 (blocks view)**。默认视图是调用树视图。你可以使用位于详细面板下面的按钮来显示三个主要视图的任何一种。

队列视图(Queues View)

队列视图显示应用程序创建的所有队列、在它们上面执行的 blocks 和相关的分析。

队列视图显示以下信息：

- **Graph (图形)** 如果该选项被选中，Dispatch instrument 工具会在跟踪面板上面显示该队列的直方图统计信息。
- **Queue Name (队列名称)** 队列创建的时候用户给它赋值的名称。队列全局名称是有系统赋值的。
- **Conc (并行)** Concurrent 的缩写，表明该队列是并行的（而不是串行）。
- **Live (活跃)** 表明该队列是活跃的(还没被释放)。
- **#Blocks (块)** 当前被添加进入队列还没被调用的 blocks 的数量。
- **#Sync (同步)** 已经被同步调度的 blocks 的数量。
- **Total Processed (总处理)** 已经被执行了的 blocks 的数量。

- Latency (**潜伏期**) blocks 待在队列的平均时间(以微秒为单位)。换言之，调用时间和进入队列时间之间的差异。
- Total CPU Time (**总 CPU 时间**) 指定队列的 blocks 在 CPU 上面执行的总时间(以微秒为单位)。

你可以在跟踪面板上面选择显示一个或多个队列的数据。跟踪面板可以被设置来显示以下任何的数据：

- Blocks Processed (**已处理的块**) 特定时间周期被处理的 blocks 的数量。默认时间周期为 10 微秒。
- Block Count (**块数**) 特定的时间周期内当前仍在队列里面 blocks 的数量。
- CPU Usage (**CPU 占有率**) 特定周期内 CPU 的活跃程度。
- Work Time (**工作时间**) 特定时间周期内队列的 blocks 消耗的 CPU 总时间。
- Latency (**潜伏期**) 特定周期内 block 的平均潜伏期(以微秒为单位)。

对于队列视图上面显示的每个队列，你可以单击它的焦点按钮来查看已经添加进入队列的 blocks 和使用队列调用了的 blocks 的列表。如果你比较在乎你的 blocks 执行的顺序或你想要获得栈跟踪的详细视图的话，该队列视图是很有帮助的。你可以选中一个 block，并在扩展详细面板上面查看该 block 入队列和调用的栈跟踪信息。

调用树视图(Call Tree View)

调用树视图集合了所有栈跟踪和显示它们的调用树。如果你通过队列划分调用树，你可以查看那个队列是最活跃和有最多 blocks 被调用。

调用树视图显示以下的信息：

- %Calls (**调用百分比**) 当前栈跟踪已经出现的调用百分比。
- #Calls (**调用次数**) 当前栈跟踪已经出现的次数。
- Library (**库**) 栈跟踪出现所在的 framework 或 bundle 的名称。
- Symbol Name (**符合名**) 栈跟踪的帧标示符。

跟踪面板可以被设置来显示以下的数据：

- Blocks Invoked (**Blocks 调用**) 特定周期内指定调用类型的 blocks 的数量。
- Total Work Time (**总工作时间**) 特定周期内 blocks 的消耗的总 CPU 时间。

对于调用树视图的每个符号，你可以单击它的焦点按钮来修整的其余的树来集中

于你关注的节点和它的子节点。你可以选中一个 block 并在扩展详细面板上面查看该 block 最重要的栈跟踪信息。

Blocks视图 (Blocks View)

Blocks 视图显示 block 和队列的信息。该视图显示所有在队列上下文被执行的 blocks，包括没有被显式入队列的。比如，如果 block A 被添加入队列并且此时执行 block B，那么这两个 block 都会被显示在 Blocks 视图上面，但是只有 block A 被显示在队列视图上面 (Queues View)。

Blocks 视图显示以下信息：

- Graph (图形) 如果该选项被选中，该 instrument 工具会在跟踪面板上面显示该 block 的直方图统计信息。
- Blocks Name (Blocks 名称) 编译器给 block 赋值的名称。
- Block Library (Blocks 库) Block 声明所在的 framework 或 bundle。
- Total Work Time (总工作时间) 该 block 的所有调用的中的执行时间（以微秒为单位）。
- Average Work Time (平均工作时间) 该 block 的平均执行时间（以微秒为单位）。
- Count (调用次数) 该 block 被调用的次数。

该 block 的扩展详细面板显示了最多调用栈跟踪的信息。该 Block 在该栈跟踪里面经常被调用。

你可以在跟踪面板上面选择显示一个或多个 blocks 的数据信息。跟踪面板可以被设置来显示以下的数据：

- Blocks Invoked (Blocks 调用) 特定周期内指定调用类型的 blocks 的数量。
- Total Work Time (总工作时间) 特定周期内 blocks 消耗的总的 CPU 时间。

对于 Blocks 视图里面每个 block, 你可以单击它的焦点按钮来查看一系列使用来执行 block 的队列。对于每个队列，你可以单击它的焦点按钮来查看和该队列相关的一系列 blocks。如果你关注那个 blocks 被队列的上下文调用和它们相应的执行顺序的话，那个 Blocks 视图将非常有帮助。

8.3 Energy Diagnostics Instruments[电池诊断相关]

此部分的相关 instruments 工具提供了 iOS 设备上面关于能量使用的诊断。它们

同时测量设备主原价的开 - 关的状态。

iOS 设备在使用电池电源和外部电源的具有不同的表现行为。这会影响在这些 instruments 工具上面收集到的数据。特别是 Energy Usage instrument 工具被影响到。但设备连接到外部电源的时候，它不会影响测试能量使用（Energy Usage）情况。

以下是这些 instruments 工具的典型工作流程：

1. 连接设备到你的开发环境。
2. 启动 Xcode 或 Instruments 应用。
3. 在设备上面，选择 Settings > Developers，并打开电源日志（power logging）。
4. 断开设备，并执行所需的测试。
5. 重新连接设备。
6. 在 Instruments 应用里面打开 Energy Diagnostics 模板。
7. 选择 File > Import Energy Diagnostics from Device。

当你执行下面操作的时候，电量诊断的数据会被清空：

- 关闭设备的电源日志。
- 断开设备并重启。
- 电池电量消耗完成。

8.3.1 电量使用（Energy Usage）

Energy Usage instrument 工具测量设备启动后的电量使用。该 instrument 工具提供了大量工作流程的宏观测量。时间刻度对于比较运行的不同很有帮助。电源事件（标志）以编程方式添加。

详细视图将会显示以下的信息：

- Energy Usage Level（电量使用级别）在刻度 0-20 之间的相关电量使用。
- Power Source Events（电源事件）电池或外部电源的转换。

注意：*Energy Usage instrument 工具当前支持 iPhone 3GS 和第三代 iPod touch 及以上的设备。*

8.3.2 CPU 活动（CPU Activity）

CPU Activity instrument 工具给出了一个设备在做什么的指示。该 instrument

工具提供了一个 Activity Monitor instrumentation 的简明版本。

详细视图显示了以下信息：

- Time (**时间**) 测量的时间间隔。
- Total Activity (**总活动**) CPU 活动的百分比。
- Foreground App Activity (**前台应用的活动**) 前台应用程序的活动的百分比。
- Audio Processing (**音频处理**) 音频活动的百分比。
- Graphics (**图形**) 图形活动的百分比。
- App Activity (**应用活动**) 应用状态切换。

8.3.3 显示亮度 (Display Brightness)

Display Brightness instrument 工具记录影响电量使用的亮度的改变情况。该 instrument 工具不会记录因为环境光传感器造成的亮度改变事件。

你可以设置屏幕的默认亮度，通过选择 Settings > Brightness。当屏幕开启，记录跳转到预设的水平。当屏幕关闭时，记录下降到零。

8.3.4 休眠/唤醒 (Sleep/Wake)

如果设备正在运行，这 Sleep/Wake instrument 工具会显示一个红色带，但如果设备正处于休眠状态，或试图进入休眠状态，或重休眠状态中唤醒时，它则显示一个较深颜色的频段。在休眠期间，电量测量会显示为零。该 instrument 工具对相关的 instruments 工具非常有帮助。

8.3.5 蓝牙 (Bluetooth)

如果蓝牙开启可用时，Bluetooth instrument 工具会显示一个红色频段，否则如果蓝牙关闭时，这显示一个黑色频段。

8.3.6 无线 (WiFi)

如果 WiFi 启用的时候，WiFi instrument 工具显示一个红色的频段，否则如果 WiFi 关闭时，显示一个黑色频段。

8.3.7 定位（GPS）

如果 GPS 启用时，GPS instrument 工具显示一个红色的频段，否则如果 GPS 关闭，则显示一个黑色的频段。

8.4 File System Instruments[文件系统相关]

该部分的 instruments 工具分析文件系统的信息和活动，比如读和写操作，权限等等。

8.4.1 I/O 活动(I/O Activity)

I/O Activity instrument 工具记录 I/O 事件：函数调用，比如在文件系统上面的 read、write、open、close 等操作。你可以使用该 instrument 工具来启动和样本分析单个运行在 iOS 设备上面的进程。尽管 I/O Activity instrument 工具提供了一个调用树的回溯跟踪视图，在 Mac OS X 上有类似 I/O Activity instrument 工具的 fs_usage 实用工具。

在详细面板，你可以选择以下的一个或多个类别。每个类别包含了一组探针（probes）（BSD 函数）。

- File Attributes（文件属性）
 - getattrlist
 - setattrlist
 - listxattr
- File Permissions（文件权限）
 - chmod
 - fchmod
 - chown
 - fchown
 - access
- Open and Close（打开和关闭文件）
 - open
 - fdopen

- fopen
- freopen
- close
- fclose
- Other (其他)
 - lseek
 - fsync
 - dup
 - dup2
 - link
 - unlink
- Read and Write (读写操作)
 - read
 - pread
 - readv
 - write
 - pwrite
 - writev
- Shared Memory (共享内存)
 - shm_open
 - shm_unlink
- Sockets (套接字)
 - recv
 - recvfrom
 - recvmsg
 - send
 - sendmsg
 - sendto
- Stats (统计)

- lstat
- lstat64
- stat
- stat64
- fstat
- fstat64

I/O Activity instrument 工具捕获以下信息：

- Function (**函数**) 被调用的函数的名称。
- Duration (**时长**) 函数调用的时长。
- In File (**输入文件**) 输入文件描述符。
- In Bytes (**输入字节**) 要求读或写的字节的数量。
- Out File (**输出文件**) 输出文件描述符。
- Out Bytes (**输出字节**) 实际读或写的字节数量。
- Thread ID (**线程 ID**) 线程的标示符。
- Stack Depth (**栈深度**) 函数调用期间使用的栈帧的数量。
- Error (**错误**) 函数调用期间最近出现的错误。
- Path (**路径**) 可执行文件执行操作的文件路径。
- Parameters (**参数**) 函数调用的参数。对于特定函数的参数的描述，参阅相应 API 文档。

跟踪面板可以被设置来显示以下的数据信息：

- Sample number (**样本数量**)
- Call duration (**调用时长**)
- Input file descriptor (**输入文件描述符**)
- Input bytes (**输入字节**)
- Output file descriptor (**输出文件描述符**)
- Output bytes (**输出字节**)
- Thread ID (**线程 ID**)
- Stack depth (**栈深度**)

对于任何函数的调用，你可以打开该调用的对应扩展详细面板来查看整个调用的

回溯跟踪信息。该 instrument 工具还在详细面板提供了一个调用树视图。

I/O Activity instrument 工具某些时候被用来配合其他 iOS instruments 工具使用。比如，你可以配合使用 I/O Activity instrument 工具和 OpenGL ES Driver instrument 工具来检查纹理加载进程。

8.4.2 文件锁(File Locks)

File Locks instrument 工具记录调用 flock 函数时咨询文件锁的操作。该 instrument 工具可以运行在单个进程或所有当前系统运行的进程上面。该 instrument 工具的实现使用了 DTrace，并可以导入 DTrace 脚本。

该 instrument 工具捕获以下信息：

- 函数名称
- 函数调用者（包括可执行文件名称和栈跟踪信息）
- 锁住的文件路径
- 操作类型，它是一个整型变量，有以下相应的值（包括这些值的组合）：
 - 1 - 共享锁
 - 2 - 独占锁
 - 4 - 不阻塞锁
 - 8 - 解锁

跟踪面板可以被设置来显示以下的数据：

- 栈深度（**Stack depth**）
- 线程 ID（**Thread ID**）
- 时间戳（**Timestamp**）

对于详细面板的条目，你可以在扩展详细面板打开该调用的栈跟踪信息，和任何探针的信息、事件发生的时间。

8.4.3 文件属性(File Attributes)

File Attributes instrument 工具记录文件系统中文件的所有者和访问权限的改变事件。该 instrument 工具可以运行在单个进程或所有当前系统运行的进程上面。该 instrument 工具的实现使用了 DTrace，并可以导入 DTrace 脚本。该 instrument

工具关于每个函数调用的以下信息：

- 通过 `chmod` 和 `fchmod` 函数修改的文件权限
 - 通过 `chown` 和 `fchown` 函数修改文件所有者和所在的组
- 对于每个函数调用，`instrument` 工具捕获以下信息：
- 函数的名称
 - 函数的调用者（包括可执行文件的名称和栈跟踪信息）
 - 可执行文件操作的文件路径
 - 被修改的文件的文件描述符
 - 模式标志，它指示了可以在文件上面应用的权限（该值只在调用 `fchmod` 的 `chmod` 时才被捕获）
 - 文件的新的所有者的用户 ID（该值只在调用 `fchown` 的 `chown` 时才被捕获）
 - 文件的新组的组 ID（该值只在调用 `fchown` 的 `chown` 时才被捕获）

注意：关于模式标志的解析更多信息，参见 *chmod* 的主页。

跟踪面板可以被设置来显示以下的信息：

- 栈深度（**Stack depth**）
- 线程 ID（**Thread ID**）
- 文件描述符（**File descriptor**）
- 模式（**Mode**）
- 用户 ID（**User ID**）
- 组 ID（**Group ID**）

对于详细面板的条目，你可以在扩展详细面板打开该调用的栈跟踪信息，和任何探针的信息、事件发生的时间。

8.4.4 文件活动（File Activity）

`File Activity instrument` 工具可以让你监听文件的访问。该 `instrument` 工具可以运行在单个进程或系统所有当前运行的进程之上。该 `instrument` 的实现使用了 `DTrace`，并可以导入 `DTrace` 脚本。该 `instrument` 工具捕获以下函数的调用信息：

- 打开和新建一个可以读取或写入的文件（`open`）
- 删除从每个进程的引用表的描述符（`close`）

- 获取关于一个文件的信息 (fstat)

对于每个函数的调用, 该 instrument 捕获以下的信息:

- 函数的名称
- 函数的调用者 (包括可执行文件的名称和栈跟踪信息)
- 可执行文件操作的文件路径
- 文件的文件描述符

跟踪面板可以被设置来显示以下的数据:

- 栈深度 (Stack depth)
- 线程 ID (Thread ID)
- 文件描述符 (File descriptor)

对于详细面板的条目, 你可以在扩展详细面板打开该调用的栈跟踪信息, 和任何可用探针的信息、事件发生的时间。

8.4.5 目录I/O (Directory I/O)

Directory I/O instrument 工具记录目录的相关操作, 比如移动目录, 创建符合连接等等。该 instrument 工具可以运行在单个进程或所有当前系统运行的进程之上。该 instrument 工具的实现使用了 DTrace, 并可以导入 DTrace 脚本。该 instrument 工具捕获以下函数的调用信息:

- **delete** - 删除一个文件和目录
- **link** - 创建一个硬连接或符合连接
- **mkdir** - 创建一个目录
- **mount** - 挂载一个文件系统
- **rename** - 修改一个文件或目录的名称
- **rmdir** - 移除一个目录
- **symlink** - 新建一个符合连接
- **unlink** - 移除一个连接
- **unmount** - 卸载一个文件系统

对于每个函数的调用, 该 instrument 捕获以下的信息:

- 函数的名称

- 函数的调用者（包括可执行文件的名称和栈跟踪信息）
- 可执行文件操作的文件或目录的路径
- 新文件或目录的名称（在适当情况下）

跟踪面板可以被设置来显示以下的数据：

- 栈深度（**Stack depth**）
- 线程 ID（**Thread ID**）

对于详细面板的条目，你可以在扩展详细面板打开该调用的栈跟踪信息，和任何可用探针的信息、事件发生的时间。

8.5 Garbage Collection Instruments[垃圾回收相关]

该部分的 `instruments` 收集由垃圾回收器回收内存的信息。为了使用这些 `instruments` 工具，程序必须是建立在垃圾回收器之上并启用垃圾自动回收功能。关于编写一个垃圾回收器的程序，参阅 `Garbage Collection Programming Guide`。

8.5.1 GC Total

`GC Total instrument` 工具追踪所有由垃圾回收器分配和释放的对象或字节的总数量。该 `instrument` 工具可以运行在单个进程或所有当前系统运行的进程之上。该 `instrument` 工具的实现使用了 `DTrace`，并可以导入 `DTrace` 脚本。它记录只启用了垃圾回收的进程的数据。

该 `instrument` 工具捕获以下信息：

- 对象初始化和回收函数（包括栈跟踪信息）
- 有垃圾回收器回收的对象的数量
- 由垃圾回收器回收的全部字节的数量
- 全部已分配且仍然使用中的自己数量
- 全部已经回收和正在使用中的字节的数量

跟踪面板可以被设置来显示以下数据：

- 栈深度（**Stack depth**）
- 线程 ID（**Thread ID**）
- 回收的对象（**Objects reclaimed**）

- 回收的字节 (Bytes reclaimed)
- 正在使用的字节 (Bytes in use)
- 全部字节 (Total bytes)

对于详细面板的条目，你可以在扩展详细面板打开该调用的栈跟踪信息，和任何可用探针的信息、事件发生的时间。

8.5.2 垃圾回收 (Garbage Collection)

Garbage Collection instrument 工具测量垃圾回收器清除阶段的回收数据。该 instrument 工具可以运行在当个进程或所有当前系统运行的进程之上。该 instrument 工具的实现使用了 DTrace, 并可以导入一个 DTrace 脚本。它只记录启动了垃圾回收的进程的数据。

该 instrument 工具捕获以下信息：

- 对象的初始化和回收函数（包括栈跟踪信息）
- 分配内存的区(zone)
- 事件是否是分代的（1 为 YES, 0 为 NO）
- 由垃圾回收器回收的对象的数量
- 由垃圾回收器回收的字节的数量
- 清除时间的持续时间（以微秒为单位）

跟踪面板可以被设置来显示以下信息：

- 栈深度 (Stack depth)
- 线程 ID (Thread ID)
- 区 (Zone)
- 是否分代 (Is generational) (分代回收器查找由非分代回收器错过的不在使用的比较旧的对象，但它需要运行更长的时间，所以不经常运行)
- 回收的对象 (Objects reclaimed)
- 回收的字节 (Bytes reclaimed)
- 持续时间 (Duration)

对于详细面板的条目，你可以在扩展详细面板打开该调用的栈跟踪信息，和任何可用探针的信息、事件发生的时间。

8.6 Graphics Instruments[绘图相关]

该部分的 instruments 工具收集和绘图相关的数据。

8.6.1 核心动画 (Core Animation)

Core Animation instrument 工具测量一个运行在 iOS 设备上面的进程每秒的核心动画帧的数量，以及屏幕外的帧计数。

此外，Core Animation instrument 工具可以提供一個可视化的提示(hints)来帮助你理解内容是如何被渲染到屏幕的。该 instrument 工具包含了许多可选的选项来允许你选择特定类型的渲染提示(rendering hints)。提示工作在设备上的任何程序。没有必要记录样本数据来激活提示。但你关闭 Instruments 文档或删除该 instrument 工具时，提示会被伴随关闭。如果有一个 OpenGL 的表层被显示，渲染提示不会影响该表层。

该 instrument 工具包含以下的渲染提示：

- 着色混合层 (Color Blended Layers)，放置一个红色(red)的覆盖层在使用混合绘画的图层上面。放置一个绿色(green)的覆盖层在不使用混合绘画的图层上面。
- 着色复印的图像 (Color Copied Images)，放置一个蓝绿色(cyan)的覆盖层在核心动画复印的图像上面。
- 立即着色 (Color Immediately)，执行着色刷新操作之后，不需要等待 10 毫秒。
- 着色未对齐的图像 (Color Misaligned Images)，放置一个洋红色(magenta)的覆盖层在源像素不对齐目标像素的图像上面。
- 着色屏幕外渲染为黄色 (Color Offscreen-Rendered Yellow)，放置一个黄色的覆盖层在屏幕外渲染的内容上面。
- 着色 OpenGL 快速路径为蓝色 (Color OpenGL Fast Path Blue)，放置一个蓝色的覆盖层在从合成器分离的内容之上。
- 闪光更新的区域 (Flash Updated Regions) 屏幕闪光更新的区域为黄色。

你可以使用这些渲染提示来在不需要做必要绘图的时候找出没有改变的重绘内容。

跟踪面板可以显示每秒的帧数。扩展详细面板显示每个样本点的分析。

8.6.2 OpenGL 驱动器 (OpenGL Driver)

OpenGL Driver instrument 工具样本分析 OpenGL 的统计信息。该 instrument 工具可以运行在单个进程或所有当前系统运行的进程上面。

该 instrument 工具捕获以下信息：

- 交换缓冲区数 (**Buffer swap count**)
- 客户端 OpenGL 等待时间 (**Client GLWait time**)
- 每个样本的 2D 指令字节 (**Command 2D Bytes per sample**)
- 上下文 2D 的计数 (**Context 2D Count**)
- 上下文 OpenGL 计数 (**Context GLCount**)
- 上下文空闲缓冲区 2D 等待时间 (**Free Context Buffer 2D Wait time**)
- 图形地址映射表的字节大小 (**Gart size bytes**) - Gart 为 [Graphics Address Remapping Table](#)
- 表层计数 (**Surface count**)
- 纹理计数 (**Texture count**)
- 显存空闲字节 (**Vram free bytes**) - Vram 为 [Video Random Access Memory](#)
- 更多 (**and much more**)

安装该 instrument 工具并打开扩展详细面板，来查看捕获数据的全部列表。跟踪面板指示了那些数据被收集。扩展详细面板显示了每个样本点的统计。

8.6.3 OpenGL ES 驱动器 (OpenGL ES Driver)

OpenGL ES Driver instrument 工具在 iOS 设备上查询 GPU 驱动器来给单独进程进行 OpenGL 样本统计分析。该 instrument 工具帮你确定你已经使用了设备的 OpenGL 和 GPU 的效率。

注意： Apple 已经售出各种 GPUs，每个都有不同的统计数据集。Instruments 应用在查询设备之前是不知道有什么统计数据将要显示的。

GPU 的硬件由两部有效的组合：平铺器(Tiler)和渲染器(Renderer)。一个场景被平铺并渲染。平铺器和渲染器组件通常工作在不同的场景。每个组件的利用率可能达到 100%。

平铺器和渲染器利用率对于确定瓶颈很有帮助。渲染器利用率低下时意味着进程

正等待被平铺，此时降低场景的复杂度可能对缓解利用率问题。平铺器和渲染器利用率低下时暗示程序的其他地方出现了瓶颈。

该 instrument 工具捕获以下信息：

- 上下文计数 (**Context Count**) 全局 OpenGL 上下文的数量。需要注意的是可以有其他运行的进程（比如，SpringBoard）负责创建一个上下文。这一统计分析可以帮助你重点发现任何没有得到销毁的错误上下文。
- 命令缓冲区分配的字节数 (**Command Buffer Allocated Bytes**) 被分配用于存储和提交命令缓冲区数据的字节数。该空间可以用来提交所有 OpenGL 命令和用户指定的顶点数据。
- 命令缓冲区提交的字节数 (**Command Buffer Submitted Bytes**) 已经提交给驱动器的命令缓冲区字节数。该数量包括所有 OpenGL 命令和用户指定的顶点数据。每次提交时，已提交字节数 (**Submitted Bytes**) 会从总的内存量里面递增。你可能需要通过它除以提交次数 (**Submit Count**) 所得的值来获取每次提交的评价使用值（该平均值总是稍微少于分配的字节数，因为分配的大小就是实际使用的边界）。
- 命令缓冲区提交的次数 (**Command Buffer Submit Count**) 驱动器处理的命令缓冲区的次数。一个命令缓冲区也许包含多个渲染器和切换器。每个命令缓冲区被传给 GPU 时，该值会随着递增（每个命令缓冲区可能包含零个或多个场景）。
- 命令缓冲区渲染的次数 (**Command Buffer Render Count**) GPU 渲染的 3D 帧的数量。
- 命令缓冲区切换次数 (**Command Buffer Transfer Count**) 驱动程序处理的交换命令的显示次数。
- 渲染利用率% (**Renderer Utilization %**) GPU 花在执行分片处理的时间比例。
- 平铺器利用率% (**Tiler Utilization %**) GPU 花在顶点处理和平铺的时间比例。
- 设备利用率% (**Device Utilization %**) GPU 花在执行一些平铺和渲染工作的时间比例。
- 平铺场景的字节数 (**Tiled Scene Bytes**) 用于平铺场景的字节数。值越大意味着场景越复杂。如果你的场景复杂到超出填充平铺场景的字节时，你可以使用分割场景模式。但通常情况下你应该避免使用该模式。统计数据是由每个场景递加

的。同时你需要使用字节数除以场景数。

- 分割场景数 (**Split Scene Count**) 一个场景的部分进入分割场景模式的次数。但场景复杂度很高而且无法把整个场景填充进入一个平铺场景字节缓冲区的时候, 需要使用分割场景模式。分割的场景数和平铺场景字节数可以用来确定渲染的路径, 而且你需要降低该路径的复杂度。更多的时候你应该避免使用分割场景。
 - 资源字节数 (**Resource Bytes**) 用于纹理的字节数。
 - 资源计数 (**Resource Count**) 使用中的纹理的数量。
 - 每秒核心动画帧数 (**Core Animation Frames Per Second**) 核心动画每秒显示新的合成帧的数量。这些帧可能包含了 CAEAGLLayer 对象的 OpenGL ES 帧。
- 跟踪面板指示数据何时被收集的。

因为事件不被捕获, 所以在扩展详细面板上面没有相应的回溯跟踪。相反, 扩展详细面板显示了每个样本点的统计分析的全列表。

8.6.4 OpenGL ES 分析器 (OpenGL ES Analyzer)

OpenGL ES Analyzer 是一个在测量和分析应用程序中 OpenGL ES 活动的 iOS instrument 工具。该 instrument 工具包含了一个专业系统, 该系统查找问题并提供基于 Apple 硬件和软件平台最近实践和复杂的知识。该 instrument 工具同样提供了一个大量的性能统计分析。

应用程序每次调用 OpenGL ES 框架时, 该 instrument 工具跟踪调用并记录时间、时长、回溯跟踪和其他参数, 并把信息更新到主机上。该 instrument 工具分析 OpenGL 命令流来计算有用的性能分析数据并驱动专业系统, 然后反过来提供修正建议和性能建议。

比如, 该 instrument 工具可能会告诉你它检测到没有使用顶点缓冲区对象的顶点数据数组。顶点数组是保存在主内存的客户端数据。如果把这些数据以顶点缓冲区对象上传到 GPU 的话将会更加高效。尽管它的实现有点复杂, 但性能提高是显著的。

该 instrument 工具提供以下视图:

- 帧统计 (**Frame statistics**) 对应图形的时间刻度。它会以表格形式呈现图形中渲染的数据。
- 分析结果 (**Analysis findings**) 专业系统的建议, 和扩展详细视图的栈跟踪。

使用不同的颜色来标示问题的严重性。你可以扩展子层来查看每个建议的特定事件，然后更深层的扩展来查看由建议产生的 OpenGL ES 命令的序列。

- 跟踪功能 (**Function trace**) OpenGL 命令的全部列表，和扩展详细视图的参数和回溯跟踪。
- API 统计 (**API statistics**) 列出 OpenGL 调用唯一总时间和每次调用的平均时间。
- 调用树 (**Call tree**) 提供所有用户调用 OpenGL ES 或 EAGL 函数的导航，利用 instrument 工具的数据挖掘工具。

但你双击分析结果或跟踪功能的回溯跟踪符合时，将会显示相关的源码。

覆盖部分是用来绕过在图形管道的阶段。这可让你隔离问题并找到你的代码中的瓶颈。

注意： *OpenGL ES Analyzer instrument* 工具不支持 iPhone 3GS 和第三代 iPod touch 之前的设备。

8.7 Input/Output Instruments[输入输出相关]

以下的 instrument 工具收集和 I/O 操作相关的数据。

8.7.1 读/写 (Reads/Writes)

Reads/Writes instrument 工具记录文件的读取和写入操作。该 instrument 工具可以运行在单个进程或系统当前运行的所有进程之上。该 instrument 工具的实现使用了 DTrace，并可以导入 DTrace 脚本。它收集关于每个读取和写入函数的相关信息，包括 read、write、pread 和 pwrite。

该 instrument 工具捕获以下信息：

- 函数的名称
- 函数的调用者（包括可执行文件的名称和栈跟踪信息）
- 可执行文件操作的文件的路径
- 被修改文件的描述
- 读取或写入的字节数

跟踪面板可以被设置来显示以下的数据：

- 栈深度 (**Stack depth**)

- 线程 ID (Thread ID)
- 文件描述符 (File descriptor)
- 字节数 (Bytes)

关于任何这些调用，你可以打开扩展详细面板来查看该调用的栈跟踪信息，和任何可用探针的信息、事件发生的时间。

8.8 Master Tracks Instruments[界面操作跟踪相关]

主跟踪轨迹 (Master Tracks) 部分包含了用户界面的记录器，它可以让你记录并回放程序中用户的一系列动作。

8.8.1 用户界面 (User Interface)

User Interface instrument 工具可以加载一个程序或附加到一个进程之上，并记录你和界面之间的交互。你可以多次回放这些记录，并运行任何其他你选择的 instruments 工具进行这样的操作。你可以使用该 instrument 工具来创建作为你质量保证计划的一部分的用户界面的可重复测试，并捕获偶然发生的错误。User Interface instrument 工具的详细使用在“使用用户界面跟踪工作 (Working with a User Interface Track)”部分介绍。

8.9 Memory Instruments[内存相关]

该部分的 instruments 工具跟踪内存使用情况。

8.9.1 共享内存 (Shared Memory)

Shared Memory instrument 工具记录共享内存的打开和取消链接。该 instrument 工具可以运行在单个进程或系统当前正在运行的所有进程之上。该 instrument 工具的实现使用了 DTrace 并可以导入 DTrace 脚本。它收集关于每个共享内存的访问信息，包括 shm_open 和 shm_unlink。

该 instrument 工具捕获以下信息：

- 函数的名称
- 函数的调用者（包括可执行文件的名称和栈跟踪信息）

- 共享内存区域的名称
- 用来打开共享内存区域的标记（查看 `shm_open` 主页）
- 模式标记，指示该共享区域的访问权限（查看 `chmod` 主页）

跟踪面板可以被设置来显示以下的数据：

- 栈深度（**Stack depth**）
- 线程 ID（**Thread ID**）
- 标志（**Flags**）
- `mode_t`

关于任何这些调用，你可以打开扩展详细面板来查看该调用的栈跟踪信息，和任何可用探针的信息、事件发生的时间。

8.9.2 分配内存（Allocations）

Allocations instrument 工具跟踪应用的内存分配情况。该 instrument 工具要求你加载一个进程，以便它能收集进程开始之后的数据。

该 instrument 工具捕获以下信息：

- 类别（**Category**）通常是一个 Core Foundation 对象、Objective-C 类、或原始内存块(block)。
- 净分配字节数（**Net Bytes**）当前已经分配内存但是仍然没有被释放的字节的总数。
- 净分配数（**#Net**）当前已经分配内存但仍然没有被释放的对象或内存块的数量。
- 总分配字节数（**Overall Bytes**）所有已经分配内存，而且包括已经被释放了的字节的总数。
- 总分配数（**#Overall**）所有当前已经分配内存，包括已经被释放了的对象或内存块的总数。
- 净余或全部内存分配（**#Allocations**）当前和全部分配数的直方图。直方条通常为蓝色。当对象总数和最大值之间的比例或最大值和当前分配数的比例少于 1/3 时，直方条会被修改为黄色。当比例等于 1/10 或更少时，直方条变为红色。

尽管显示的比例不一定是坏事（通常它们在应用程序长期运行期间是正常），

Instruments 应用通常给它们标示不同的颜色来指出分配模式以便进行进一步的研究。如果你发现类别 (categories) 的颜色为红色或黄色, 你可能需要尽量消除应用程序给定类型的非必要的临时内存分配。类似的, 你可能只是简单的尽量消除高水位标记的对象的数量。

详细面板的数据表格包含了一个图形列, 其中包含了表中的每一行的复选框。当指定类别的复选框被勾选时, instrument 工具在跟踪面板里面显示特定类别的图形。Instruments 应用通常给每个图形类别赋一个颜色。

当你鼠标移动到详细面板上面的类别名称上时, 会在类别名称的旁边显示一个更多信息的按钮。单击该按钮会显示关于该类别上的对象的详细信息, 包含以下属性:

- 块 (block) 地址。
- 函数调用或造成该分配事件的类。比如, 你可以看到该类里面的那个方法对对象进行了引用。
- 对象的创建时间。
- 负责创建对象的库。

对于任一这些事件, 你可以打开扩展详细面板来查看每个对象内存分配的栈跟踪, 包括分配的类型和事件发生的时间。

对于特定对象 (或内存块) 的实例, 你可以单击该对象地址列的更多信息按钮来查看对象相关的内存分配事件。对于每个内存分配事件, 该 instrument 工具显示以下信息:

- 对象 (它的类型) 的类别
- 事件类型
- 每个事件的时间戳
- 块地址
- 块的大小
- 负责分配块的库
- 引发分配事件的函数

对于任何分配事件, 你可以打开扩展详细面板来查看栈跟踪信息, 同时也可以查看任何可用的事件信息和事件具体发生的时间。

为了进一步的过滤详细面板的信息, 你可以配置 Allocation Lifespan options

（内存分配存活期选项）。这些选项可以让你过滤基于以下标准的分配事件：

- 所有创建的对象（**All Objects Created**） - 显示所有对象，无论它们是否已经被释放掉了。
- 已创建的&仍然有效的（**Created & Still Living**） - 仅显示当你停止记录时仍然存在内存的对象。

Allocations instrument 工具的检查器可以让你配置 instrument 工具跟踪信息的方法。你可以从该检查器设置一下选项：

- 记录引用数。使用该选项来跟踪每个对象的引用计数。
- 停止时丢弃未被记录的数据。使用该选项来丢弃任何已经被收集了的当还没被 Allocations instrument 工具处理的数据。

关于 Allocations instrument 工具的更多额外信息，参阅“使用 Allocations Instrument 工具分析数据（Analyzing Data with the Allocations Instrument）”。

8.9.3 内存泄露（Leaks）

Leaks instrument 工具检查进程堆泄露的内存。你可以使用该 instrument 工具配合 Allocations instrument 工具来获取内存地址的历史记录。该 instrument 工具要求你加载一个进程以便它可以从进程启动时收集数据。

该 instrument 工具捕获以下信息：

- 内存泄露的数量（**The number of leaks**）
- 每个泄露内存块的大小（**The size of each leak**）
- 泄露内存块的地址（**Address of the leaked block**）
- 泄露对象的类型（**Type of the leadked object**）

详细面板的每个视图模式以轻微不同的函数显示泄露的数据。在列表模式下，该 instrument 工具显示每个泄露的百分比，而每个内存泄露有助于发现内存泄露的总数。在大纲模式下，数据被重组以便你可以查看给定的符号有多少内存泄露。对于任何模式的条目，在扩展详细面板显示内存泄露引发的深入栈跟踪信息。

关于 Leaks instrument 工具的更多额外信息，参阅“查找内存泄露（Looking for Memory）”。

8.10 System Instruments[系统相关]

该部分的 instruments 工具收集系统活动和资源的数据。

8.10.1 时间分析器 (Time Profiler)

Time Profiler instrument 工具在规定的间隔内停止一个 Mac OS X 的程序并记录该程序内部线程的栈跟踪信息。你可以使用这些信息来确定花费在你程序上面的执行时间并提升你的代码来减少运行时间。不像很多 instruments 工具, Time Profiler 工具不需要使用 DTrace 探针来实现功能。Time Profiler 工具运行在单个进程或所有进程之上。

在采样期间, 该 instrument 工具捕获以下信息:

- 采样开始的时间
- 采样的时长
- 栈跟踪信息 (包括库和调用者信息)
- 采样期间遇到的最大栈深度
- 采样期间遇到的最频繁的函数 (热点帧)

Time Profiler 工具让你可以以不同的方式来查看这些信息。在列表模式下, 你可以查看按照时间顺序采集的样本, 它显示你代码执行的顺序。在大纲模式下, Time Profiler 提供了你程序调用栈的树形视图并显示了调用栈里对应调用函数的样本数量。

为了显示一个函数的详细调用栈, 你可以在大纲模式下扩展对应的项目或选择一个函数并打开扩展详细面板。在大纲模式下, 你可以通过单击 Option 键和条目的扩展三角形来扩大给定条目的整个调用栈。

跟踪面板默认显示每个样本时间点的栈深度。该视图对于确定你代码执行的情况很有帮助。因为它不像两个不同的执行路径结果产生相同的栈深度, 当你查看图形的重复结构时, 它可能是相同的代码被重复的执行。

Time Profiler 的一个特性是在没有真正运行 Instruments 应用也可以记录一个分析信息 (profile)。当你需要记录一个瞬间事件或它可能需要很多时间来打开并配置 Instruments 应用时, 该特性将会排上用场。为了以这种方式来记录一个分析文件, 首先你需求确保 Instruments 应用没有正在运行。按下 Dock 下的 Instruments 应用

的图标。Dock 将会显示一个菜单项，其包含了 Time Profiler 的命令、设置和分析文件。你可以选择分析一个特定进程，所有进程，或自动分析任何阻塞的进程。在分析文件创建后，它会出现在 Instruments 应用的 Dock 菜单里面的 Recent Time Profiles (最近时间分析文件) 里面。Instruments 应用打开并显示分析数据。

Time Profiler instrument 工具和 Sampler instrument 工具有点类似，但是它们之间也有很多的不同：

- Time Profiler 以诈骗 (Shark) 的方式从内核空间收集回溯跟踪数据。而 Sampler 工具从用户空间收集数据。因此，Time Profiler 在收集数据方面比 Sampler 工具更高效。

注意：如果目标进程被优化来忽略帧指针，那么 Time Profiler (以 Shark 方式) 可能产生不准确的回溯跟踪数据。

- Time Profiler 可以收集一个或多个进程的数据。Sampler 工具只能采集单一进程。
- Time Profiler 可以采集所有线程状态或只在当前运行的线程。Sampler 工具通常采样所有线程状态。通常情况下，你可能只对正在运行的线程感兴趣。当你的应用被挂起时，你需要检查所有线程的状态。

8.10.2 旋转监控器 (Spin Monitor)

Spin Monitor instrument 工具自动采样系统上无响应的应用程序。当一个应用程序 3 秒或更长内没有从窗口服务器检索事件时，它将会变为无响应的应用程序。在这段时间内没有响应的应用程序实际上可能会做一些有益的工作，也可能是被挂起。你可以该 instrument 工具生成的样本信息来修改你的代码以便保证你的程序可以一直保持及时处理事件。该 instrument 工具可以运行在单个进程或系统当前运行的所有进程之上。

在采样期间，该 instrument 工具捕获以下信息：

- 采样开始时间
- 采样的时长
- 栈跟踪信息 (包括库和调用者信息)
- 采样期间遇到的最大栈深度

- 采样期间遇到的最频繁的函数（热帧）

详细面板的每个视图模式以细微不同的方式显示样本数据。列表模式和大纲模式开始时显示此期间那个应用程序被采样的会话。每个会话负责一个时间段，该时间段内应用程序被认定为无响应，而你可以扩展给定的会话来查看此时应用程序在做什么。在列表模式下，instrument 工具显示了采样期间最频繁调用的函数相关的数据。在大纲模式下，instrument 工具显示每个会话生成的样本的数量。你也可以使用 Sample Perspective 选项来显示样本运行的时间。

该 instrument 工具的检查器可以让你设置收集样本的速率。默认情况下，该 instrument 工具每 10 毫秒收集一次样本。

8.10.3 取样（Sampler）

Sampler instrument 工具在指定的时间间隔内停止一个应用程序并记录应用程序每个线程的栈跟踪信息。你可以使用这些信息来确定花费你应用程序执行时间的地方并提高你的代码来减少运行时间。不像许多 instruments 工具，Sampler instrument 工具不要求使用 DTrace 探针来实现功能。该 instrument 工具运行在单个进程之上。

Sampler instrument 工具记录每个样本的以下数据类型：

- 被执行的函数
- 应用程序每个线程的栈跟踪信息
- 样本被采集的时间

Sampler instrument 工具可以让你以不同的方式查看这些信息。在列表模式下，你可以按照样本被采集的顺序来查看它们，它显示了你代码执行的顺序。在大纲模式下，Sampler instrument 工具提供了你应用程序调用栈的树型视图，并显示了里面每个调用函数执行时的样本的数量。

在研究的一个正在运行程序的性能时，你应该比较函数的影响和函数的执行成本。如果你的程序花费很长的时间来执行一个低影响的函数，该 instrument 工具可以标记出该情况。然后你可以使用样本数据来找出为何你的程序会花费这些时间和谁调用这些函数，通过这样你可以修复你的代码让它减少被调用的频率。

为了显示一个函数的详细调用栈，你可以在大纲模式下扩展相应项目或选择一个函数，并打开扩展详细面板。在大纲模式下，你可以通过按下 Option 键和单击对于

条目扩展按钮来扩展一个给定条目的全部调用栈。

跟踪面板默认显示每个样本时间点的栈深度。该视图可以帮助识别你代码当前正在做什么。因为两个不同的执行路径不太可能产生相同的栈深度，所以当你看到图形有重复的结构时，有可能是同一代码被重复执行了。如果该段代码需要消耗很长时间来执行，那它就是很好的优化目标。

关于 Sampler instrument 工具的额外信息，参阅“使用 Sampler Instrument 工具分析数据部分 (Analyzing Data with the Sampler Instrument)”。

8.10.4 进程 (Process)

Process instrument 工具记录由另外进程派生的进程。该 Instrument 工具可以运行在单个进程或系统所有当前运行的进程之上。该 instrument 工具的实现使用了 DTrace，并可以导入 DTrace 脚本。

该 instrument 工具捕获以下信息：

- 执行一个进程 (**execve**)
- 进程退出 (**exit**)

Process instrument 工具返回这些函数每次调用的信息，包括：

- 函数名称 (**execve** 或 **exit**)
- 函数调用者 (包括可执行文件名称，路径，和栈跟踪信息)
- 进程 ID
- 进程退出状态

跟踪面板可以被设置来显示以下任何数据：

- 栈深度 (**Stack depth**)
- 线程 ID (**Thread ID**)
- 进程 ID (**Process ID**)
- 退出状态 (**Exit status**)

对于任何调用，你可以打开扩展详细面板来查看该调用的栈跟踪，和任何可用的探针信息和事件发生的具体时间。

8.10.5 网络活动监控器 (Network Activity Monitor)

Network Activity Monitor instrument 工具记录电脑网络传输信息。该 instrument 工具可以运行在单个进程或系统当前所有运行的进程之上。

跟踪面板可以被设置来默认显示以下网络相关的数据，但你也可以配置它来显示其他类型的数据。默认情况下，它显示以下信息：

- 每秒发送的字节数量
- 每秒收接收字节的数量
- 每秒发送包的数量
- 每秒接收包的数量

8.10.6 内存监控器 (Memory Monitor)

Memory Monitor instrument 工具记录进程使用的实际内存和虚拟内存的数量。该 instrument 工具可以运行在单个进程或系统所有当前运行的进程之上。

跟踪面板可以被设置来默认显示以下内存相关的数据，但你也可以配置它来显示其他类型的数据。默认情况下，它显示以下的信息：

- 虚拟内存页面交换进入的数量 (**virtual memory page ins**)
- 虚拟内存页面交换出去的数量 (**virtual memory page outs**)
- 正在使用的虚拟内存空间的总数量
- 空闲物理内存的总数量
- 已用物理内存的总数量

8.10.7 硬盘监控器 (Disk Monitor)

Disk Monitor instrument 工具记录硬盘的读取和写入操作。该 instrument 工具可以运行在单个进程或系统所有当前运行的进程之上。

跟踪面板可以被设置来默认显示以下和硬盘相关的数据，但你也可以配置它来显示其他类型的数据。默认情况下，它显示以下信息：

- 每秒写入硬盘的字节数量
- 每秒从硬盘读取的字节数量
- 每秒处理的写操作的数量

- 每秒处理的读操作的数量

8.10.8 CPU监控器 (CPU Monitor)

CPU Monitor instrument 工具记录系统的负载。该 instrument 工具可以运行在单个进程或系统所有当前运行的进程之上。

跟踪面板可以被设置来默认显示以下负载值，但你也可以配置它来显示其他类型的的数据。默认情况下，它显示以下信息：

- 系统产生的负载的数量
- 用户产生的负载的数量
- 系统总负载

8.10.9 活动监控器 (Activity Monitor)

Activity Monitor instrument 记录由虚拟内存大小测量的系统负载。该 instrument 工具可以运行在单个进程或系统所有当前运行的进程之上。

跟踪面板可以被设置来默认显示以下的负载值，你也可以配置它来显示其他类型的的数据。默认情况下，它显示以下信息：

- 正在使用的虚拟内存控件的总量
- 系统产生的负载数量
- 用户产生的负载数量
- 系统总负载

8.11 Threads/Locks Instruments[线程相关]

以下 instruments 收集线程相关的数据。

8.11.1 Java线程 (Java Thread)

Java Thread instrument 工具记录 Java 线程的初始化和销毁。它显示：

- 每个测量的时间
- 总线程数

你可以指定特定颜色标示线程运行时，等待，和阻塞的状态。

8.12 UI Automation[界面自动化相关]

8.12.1 使用Automation Instrument工具

Automation instrument 工具允许你让 iOS 应用的用户界面测试自动化。自动化界面测试可以让你：

- 省去关键人员和释放其他工作资源
- 执行更多综合测试
- 开发可重复的回归测试
- 减少程序错误
- 提高开发周期，产品更新

Automation instrument 工具由你的测试脚本指导，演示你应用的用户界面元素，允许你记录分析结果。自动化功能可以模拟许多用户设备支持的用户操作，比如 iOS4.0 或更高版本支持的多任务。你的测试脚本可以运行在 iOS 设备和 iOS 模拟器之上而不需要任何改动。

Automation instrument 工具的一个最大的好处是可以可以和其他 instruments 工具一起执行复杂的测试，比如跟踪内存泄露和隔离性能问题的原因。

注意：为了保护，该 instrument 工具不允许你处理任何和你证书不相关的进程。这包括拷贝任何在 iTunes App Store 下载的应用。

重要：模拟动作可能无法防止测试设备自动锁定屏幕。所以在设备上运行测试之前，你应该设置设备的 Auto-Lock 偏好设置为 Never（设置->通用->自动锁定->永不）。

测试自动化脚本

你以 JavaScript 脚本的方式编写自动化测试，使用界面自动化 API 来指定在你程序运行中应该执行的动作。

使用脚本来实现自动化测试可以减少开发和部署时间以及测试人员编程技巧的要求。此外，JavaScript 提供了复杂程序来支持复杂的操作。

参阅 UI Automation Reference Collection 的 API 详情。

你的测试脚本必须是一个本机 Instrument 应用可以访问的合法的可执行 JavaScript 脚本文件。它在你的程序之外执行，所以你程序的测试版本可以和你提

交到 iTunes App Store 的版本相同。

你可以创建任意多的脚本，但是你同一时间只能运行一个脚本。API 提供一个 `#import` 的指导，它运行你编写较小，可重复使用的离散的测试脚本。比如，如果你打算在一个文件里定义通用的函数 `TestUtilities.js`，你可以通过引入定义该函数的脚本文件来使用这些函数。

```
#import "<path-to-library-folder>TestUtilities.js"
```

加载Automation Instrument工具

加载 Automation instrument 工具和其他内置的 instrument 工具有细微的不同。以下是执行的步骤：

1. 启动 Instrument 应用。
2. 选择 Automation 模板来创建一个跟踪文档。（可选的，你可以在 Instrument 应用的工具库里的 UI Automation 组找到 Automation instrument 工具，并拖动它到跟踪文档里面）。
3. 确保详细视图被显示。（如果有必要选择 View > Detail）
4. 在 Target 菜单里面选择目标 iOS 设备，然后从 iOS 应用程序里面列表选择你的应用程序。

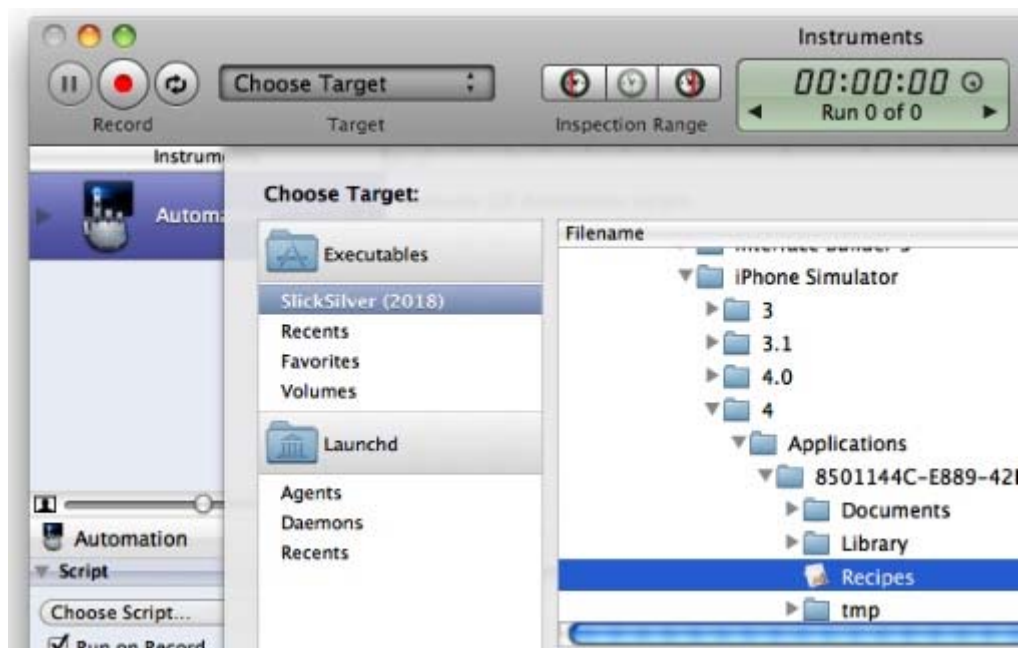
测试iOS模拟器里面的应用

当选择目标应用程序运行在 iOS 模拟器之上时，你可能需要通过导航栏里面的 Choose Target 文件浏览器选择目标应用程序，使用以下的本地路径

```
~/Library/Application Support/iOS Simulator/<iOS version>/Applications/
```

并替换 `<iOS version>` 为 iOS 实际版本号，比如图 8-1 中的 4。

Figure 8-1 Targeting an application running in iOS Simulator



一旦选定了 iOS 模拟器上面的应用程序，有两个加载项将会变得可用。

- **Process I/O:** 传输 I/O 消息到 Instruments 应用控制台，系统控制台，或 /dev/null。
- **Simulator Configuration:** 选择目标硬件设备和 iOS 版本的组合。

运行Automation Instrument

为了运行 Automation instrument 工具，执行以下步骤：

1. 如果有必要单击脚本的扩展三角形，显示面板主体内容。
2. 单击 Choose Script。
3. 在打开的面板里面，找到脚本文件并打开它。
4. 单击 Instruments 工具栏的 Record 按钮。脚本日志条目将会出现在详细面板里面。
5. 单击扩展面板里面的任何脚本日志条目来在扩展详细面板里面显示该条目的更多信息。
6. 使用停止（stop）和开始（start）的控制按钮来停止/暂停和开始/恢复你测试脚本序列的执行。

为了配置 Automation instrument 工具来自动化开始和停止你的脚本，可以通过控制 Instruments 应用工具栏的 Record 按钮，选择 Run on Record 的复选框。

如果你的应用程序崩溃了或进入后台，你的脚本将会被阻塞直到应用程序再次运行在前台，此时脚本继续执行。

要注意，你必须显式的停止记录。完成或中断你的脚本都不会关闭记录。

8.12.2 访问和操作用户界面元素

UI Automation 特性下的基础辅助机制（Accessibility-based）代表你应用程序的每个控制作为一个独特的可标示元素。为了在你应用程序的元素上面执行操作，你需要显式的标识应用程序元素的层级结构。

注意： 为了完全理解本部分内容，你应该熟悉 *iOS 人机交互指南*（*iOS Human Interface Guidelines*）。

为了阐明元素的层级结构，本部分引用图 8-2 所示的 Recipes iOS 应用（食谱应用），该应用可以在 iOS Dev Center 上面下载代码示例 iPhoneCoreDataRecipes。

Figure 8-2 The Recipes application (Recipes screen)

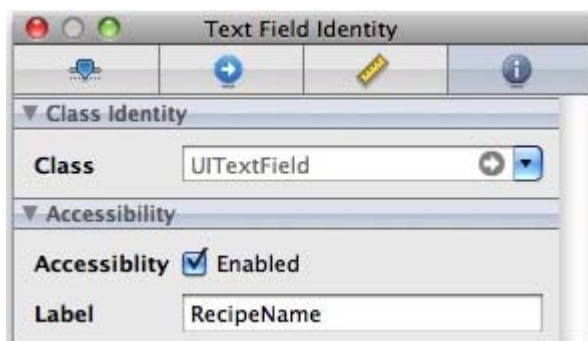


UI元素可访问性

每个可访问的元素都是继承自基础元素 `UIAElement`。每个元素都可以包含零个或多个的其他元素。

如下文详细介绍，你的脚本可以访问每个独立元素的在层级结构的位置。然而，你可以通过在设置 Interface Builder 的可访问标签来给每个元素所代表的控件赋值一个特定的名称，如图 8-3 所示。

Figure 8-3 Setting the accessibility label in Interface Builder



UI Automation 使用可访问标签（如果它设置的话）来为每个元素派生一个名字属性。除了显而易见的好处，使用这样的名称可以大大简化你测试脚本的开发和维护。

名称属性是这些元素在你测试脚本里面非常有用的四个属性之一。

- 名称 (**name**): 派生自可访问标签
- 值 (**value**): 当前控件的值，比如，文本域的文字
- 子元素集 (**elements**): 当前元素所包含的任何子元素集，比如，列表视图的单元格
- 父元素 (**parent**): 包含当前元素的父元素

理解元素的层级结构

在元素层级结构的顶层是 `UIATarget` 类，它代表了被测系统 (System under Test-SUT) 高级用户界面元素，即设备（或模拟器）和运行在你设备之上的 iOS 和你的应用程序。为了测试，你的应用程序必须是前台活跃程序（或目标程序），标识如下：

```
UIATarget.localTarget().frontMostApp();
```







为了获得应用窗口，和你应用的主窗口，你应该指定

```
UIATarget.localTarget().frontMostApp().mainWindow();
```

开始时，食谱应用程序窗口如图 8-2 所示。

在窗口内部，食谱列表代表了一个独立的视图，此时是一个列表视图（table view）：

Figure 8-4 Recipes table view

	Chocolate Cake Chocolate cake with chocolate frosting	1 hour	>
	Crêpes Suzette Crêpes flambées with grand marnier.	20min	>
	Gaufres de Liège Belgian-style waffles	1 hour	>
	Ginger snaps Nana's secret recipe	45 minutes	>
	Macarons Macarons français: chocolat, pistache, fram...	1 hour	>
	Tarte aux Fraises Delicious tart	25 min	>
	Three Berry Cobbler Raspberry, blackberry, and blueberry cobbler	1.5 hours	>

它是你应用里面列表视图数组的第一个列表视图，所以你使用下标 0 来指定它（[0]），如下：

```
UIATarget.localTarget().frontMostApp().mainWindow().tableViews()[0];
```

在列表视图内部，每个食谱由一个不同的单元格表示。你可以以类似的方式指定独立的单元格。比如，使用下标 0（[0]），你可以指定第一个单元格如下：

```
UIATarget.localTarget().frontMostApp().mainWindow().tableViews()[0].cells()[0];
```

这些独立的单元格元素的每一个都被设计作为自定义子元素来包含一个食谱记录。在第一个单元格的记录是 chocolate cake，你可以使用下面的代码来访问它的名称：

```
UIATarget.localTarget().frontMostApp().mainWindow().tableViews()[0].cells()[0].elements("Chocolate Cake");
```

显示元素的层级结构

你可以使用 logElementTree 方法来显示每个元素的所有子元素集。以下的代码举例说明列出食谱应用的主界面的元素。

```
// List element hierarchy for the Recipes screen
```



```

UIALogger.logStart("Logging element tree ...");

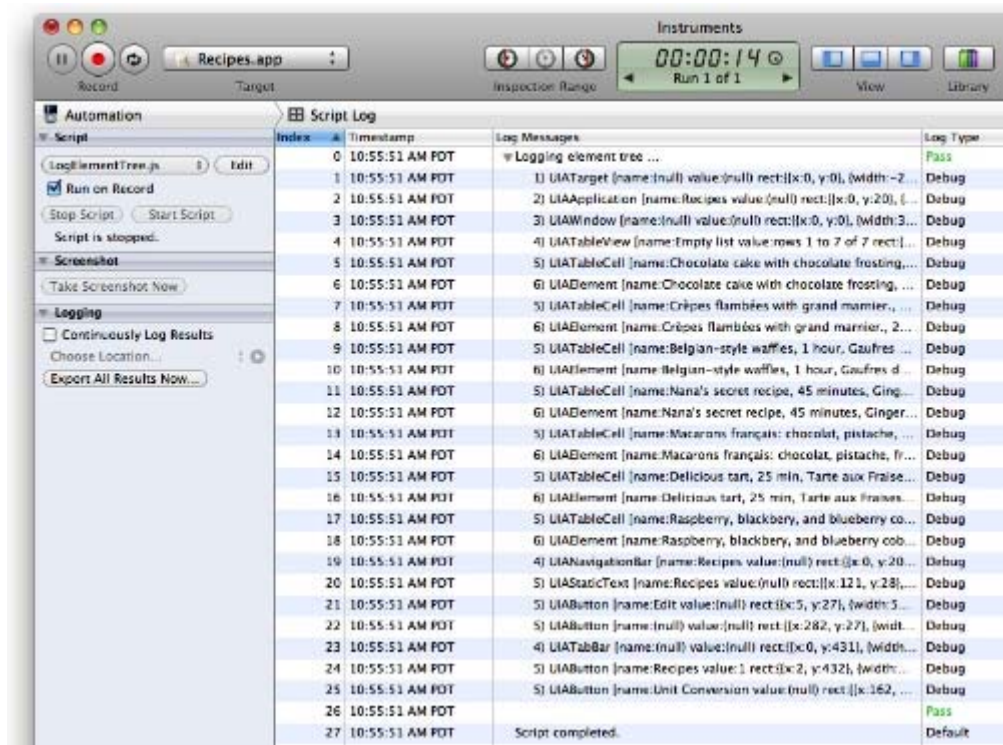
UIATarget.localTarget().logElementTree();

UIALogger.logPass();

```

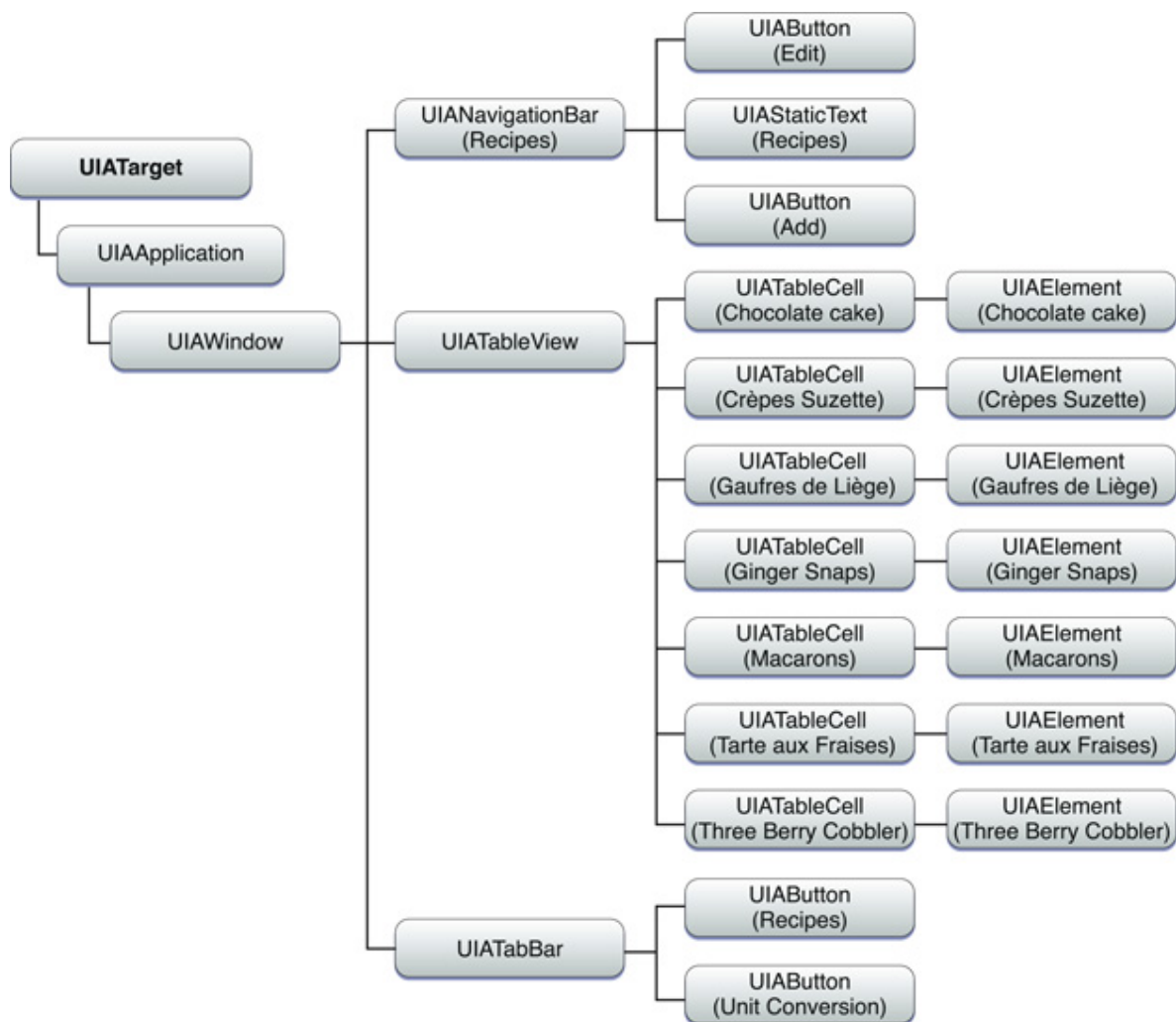
该命令的输出被 Automation instrument 工具捕获并日志输出，如图 8-5 所示。

Figure 8-5 Output from logElementTree method



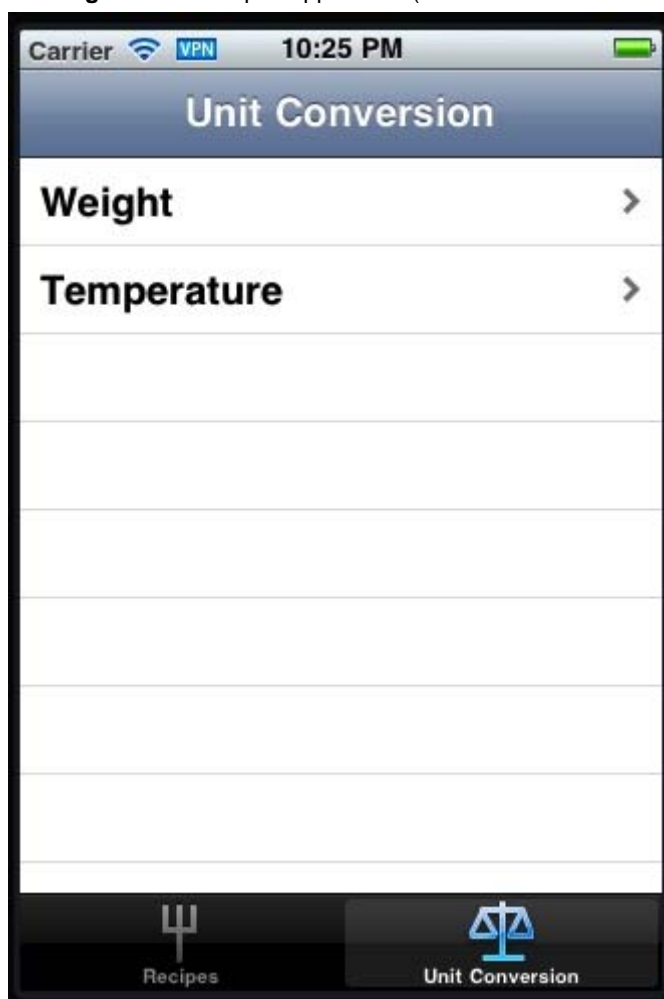
注意到每个元素开始的行项目的数量，意味它元素的在层级结构的级别。这些级别可以被视为概念图，如图 8-6 所示。

Figure 8-6 Element hierarchy (Recipes screen)



尽管屏幕技术上不算是 iOS 编程组件，而且没有显式的出现在层级结构，但它对于理解层级界面非常有帮助。轻击标签栏的 Unit Conversion 标签显示 Unit Conversion 的屏幕，如图 8-7 所示。

Figure 8-7 Recipes application (Unit Conversion screen)



以下代码轻击标签栏的 Unit Conversion 标签来显示相关的屏幕，并打印相关的每个元素的层级结构日志。

```
// Switch screen (mode) based on value of variable

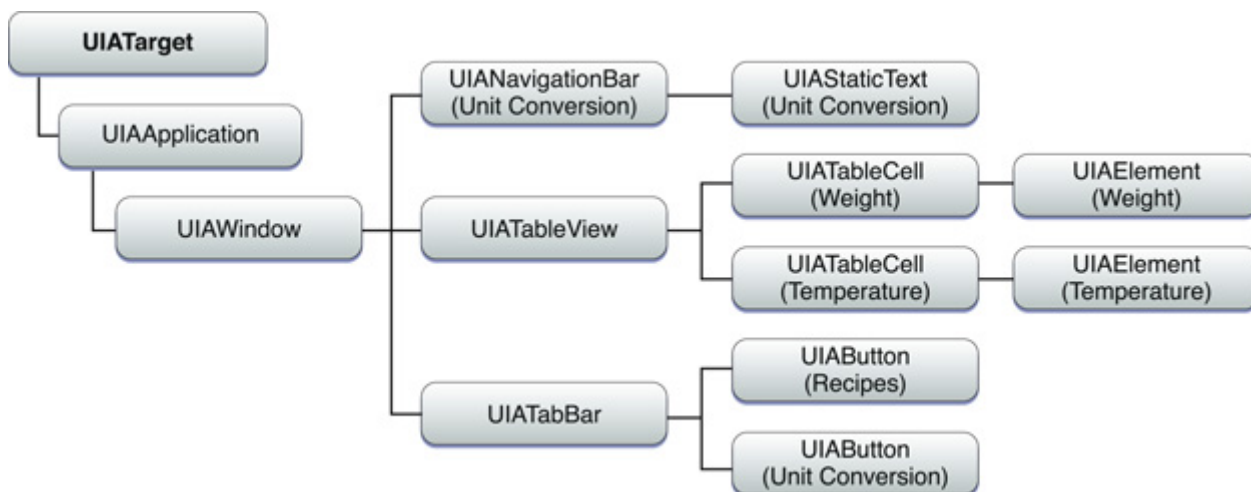
var target = UIATarget.localTarget();
var app = target.frontMostApp();
var tabBar = app.mainWindow().tabBar();

var destinationScreen = "Recipes";

if (tabBar.selectedButton().name() != destinationScreen) {
    tabBar.buttons()[destinationScreen].tap();
}
```

由此产生的日志输出层级结构，如图 8-8 所示。注意到和之前的例子相同，logElementTree 被目标调用，但是结果是当前屏幕，在该例中是 Unit Conversion 屏幕。

Figure 8-8 Element hierarchy (Unit Conversion screen)



指定元素层级结构导航

之前的例子代码介绍了使用变量来代码元素层级结构的部分。该技术可以在你的脚本中使用更短，更简单的命令。

使用变量方式同时允许在你代码中使用和重用的抽象和灵活性。以下示例使用变量(destinationScreen)来控制食谱应用程序里两个屏幕间的切换（Recipes 和 Unit Conversion）。

```

// Switch screen (mode) based on value of variable

var target = UIATarget.localTarget();
var app = target.frontMostApp();
var tabBar = app.mainWindow().tabBar();

var destinationScreen = "Recipes";

if (tabBar.selectedButton().name() != destinationScreen) {
    tabBar.buttons()[destinationScreen].tap();
}

```

使用轻微的改变，也可使该代码的代码工作，比如，对于标签工具栏包含多个标

签或不同名称的标签。

执行用户界面手势

一旦你懂得如何访问所需的元素，那么操纵相应的元素就会相对简单和直接。UI Automation API 接口提供了执行大部分 UIKit 用户动作的方法，包括多点触控手势。关于这些方法的全面详细信息，参阅 UI Automation Reference Collection。

轻击（Tapping）

或许最常用的触摸手势就是简单的轻击。在一个已知的 UI 元素上面实现单指轻击是非常简单的。比如，轻击食谱应用导航栏右边的按钮（图中显示+号），将会显示一个新的界面来添加一个新的食谱。



轻击该按钮所需的命令为：

```
UIATarget.localTarget().frontMostApp().navigationBar().buttons()["Add"].tap();
```

需要注意的是它使用了字符 Add 来标识该按钮，假设已经设置了适当的辅助标签，如上所述。

当然，大部分复杂的手势需要需要进行彻底的测试。你可以指定任何标准轻击手势。比如，为了轻击屏幕上面任意地方，你只需要提供屏幕的坐标：

```
UIATarget.localTarget().tap({x:100, y:200});
```

该命令轻击有 x 和 y 指定的坐标，而不管屏幕上该地方是什么。

同样可以执行更复杂的轻击动作。为了双击同一个地方，你可以使用以下代码：

```
UIATarget.localTarget().doubleTap({x:100, y:200});
```

比如，执行测试两个手指捏合放大和缩小动作，你可以使用如下代码：

```
UIATarget.localTarget().twoFingerTap({x:100, y:200});
```

捏（Pinching）

捏开动作通常用于放大或扩展屏幕的对象，而相应的捏合动作通常是缩小屏幕对象。你需要指定定义捏合开始坐标或捏开的结束坐标，然后跟着手势需要执行的时间长度。时长参数允许你灵活指定捏动作的速度。

```
UIATarget.localTarget().pinchOpenFromToForDuration(({x:20, y:200}, {x:300, y:200}, 2);
```

```
UIATarget.localTarget().pinchCloseFromToForDuration(({x:20, y:200}, {x:300, y:200}, 2);
```

拖拽和轻弹 (Dragging and Flicking)

如果你需要滚动一个列表或移动一个屏幕的元素，你可以使用 `dragFromToForDuration` 方法。你提供开始点的坐标和结束点的坐标，还有一个时长（以秒为单位）。以下例子指定了一个拖到手势从点 160, 200 到点 160, 400，时长为 1 秒。

```
UIATarget.localTarget().dragFromToForDuration(({x:160, y:200}, {x:160, y:400}, 1);
```

轻弹手势有点类似，但它通常是一个更快的动作，所以它一般不需要时长的元素。

```
UIATarget.localTarget().flickFromTo(({x:160, y:200}, {x:160, y:400});
```

输入文字

你的脚本有可能需要测试应用处理文本输入是否正确。为此，可以通过简单的指定一个目标文本域并使用 `setValue` 来设置它的文本值来输入文本到特定的文本域里面。下面的例子使用了一个本地变量来提供一个长字符串作为当前屏幕第一个文本域（下标为[0]）的测试案例。

```
var recipeName = "Unusually Long Name for a Recipe";
UIATarget.localTarget().frontMostApp().mainWindow().textFields()[0].setValue(recipeName);
```

导航应用中标签

为了测试你应用屏幕之间的导航，你有可能需要轻击一个标签工具栏的标签。轻击一个标签和轻击一个按钮类似；你访问合适的标签工具栏，指定所需的按钮，然后轻击该按钮，如下面例子那样。

```
var tabBar = UIATarget.localTarget().frontMostApp().mainWindow().tabBar();
var selectedTabName = tabBar.selectedButton().name();
if (selectedTabName != "Unit Conversion") {
    tabBar.buttons()["Unit Conversion"].tap();
}
```

首先，一个本地变量被声明来表示一个工具栏。脚本使用该变量访问工具栏来确定当前所选择的标签并获取标签的名称。最后，如果当前选中的标签的名称和想要的标签不匹配（该例中为” Unit Conversion”），脚本轻击所需的标签。

滚动元素

滚动是许多应用用户交互的一个大部分。UI Automation 提供了一系列滚动的方法。基础的方法允许滚动到下一个元素的左，右，上和下。大部分复杂的方法支持更

灵活的在滚动中指定动作。其中之一比如 `scrollToElementWithPredicate`, 它允许你滚动一个元素到你指定的区域。下面的例子通过元素层级结构访问合适的列表视图, 并滚动它到以配方名为” Turtle Pie” 开始的单元格。

```
UIATarget.localTarget().frontMostApp().mainWindow().tableViews()[0].scrollToElementWithPredicate("name beginswith 'Turtle Pie'");
```

使用 `scrollToElementWithPredicate` 方法允许滚动到一个可能不知道确切名称的元素。

使用谓词的功能可以显著的扩展你脚本的能力和适用性。关于使用谓词的更多信息, 参见 `Predicate programming Guide`(谓词编程指南)。

其他灵活滚动的有用方法包括 `scrollToElementWithName` 和 `scrollToElementWithValueForKey`。参见 `UIAScrollView Class Reference` 更多信息。

8.12.3 添加灵活的超时间

你的脚本可能需要等待某些动作完成。比如在食谱应用程序中, 用户通过点击 `Recipes` 标签从 `Unit Conversion` 屏幕返回到 `Recipes` 屏幕。然而, `UI Automation` 可能检测到存在 `Add` 按钮, 尝试使用测试脚本在按钮被真正绘画之前企图点击它, 而且应用实际上已经准备好接收该点击事件。需要执行一个精确的测试来保证 `Recipes` 屏幕完全绘画完而且应用程序在屏幕控制操作之前已经准备好接收用户的交互。

为了让这些情况更灵活而且更好的控制时间, `UI Automation` 提供了超时周期, 在此周期内它会在失效之前重复的尝试执行指定的动作。如果动作在超时周期内完成, 该行代码返回, 并且你的脚本可以处理它。如果动作未在超时之前完成, 将会抛出异常。默认的超时周期是 5 秒, 但是你的脚本可以可以改变它为任何时间值。

为了让该特性更好用, `UI Automation` 使用了栈模型。你推入一个自定义超时周期到栈顶, 如下面的代码, 它把超时时间缩短为 2 秒。

```
UIATarget.localTarget().pushTimeout(2);
```

你可以运行下面的代码来执行动作和把自定义超时时间推出栈。

```
UIATarget.localTarget().popTimeout();
```

使用该方法你可以创建一个强大的脚本, 它可以在一个合理的时间等于某一事件的发生。

注意: 经管通常不推荐使用显式的迟延, 但在某些时候可能必须使用。以下的代码显示了如

何指定一个 2 秒的延迟: `UIATarget.localTarget().delay(2);`

8.12.4 验证测试结果

测试的关键是能够验证每个测试已执行的, 而且知道测试通过还是失败。下面的示例代码运行测试脚本 `testName` 来测试目前食谱列表上面现有的合法的食谱元素的名称是否以“Tarte”开头。首先, 使用一个本地变量来指定单元格标准:

```
var cell = UIATarget.localTarget().frontMostApp().mainWindow().tableViews()[0].cells()
    .firstWithPredicate("name beginswith 'Tarte'");
```

其次, 脚本使用 `isValid` 的方法来测试一个现有合法的元素是否匹配这些标准。

```
if (cell.isValid()) {
    UIALogger.logPass(testName);
}
else {
    UIALogger.logFail(testName);
}
```

如果发现一个合法的单元格, 代码会输出一个通过的日子消息, 反之, 会输出一个失败消息。

需要注意的是, 该测试脚本指定 `firstWithPredicate` 和 “name beginswith 'Tarte' ”。这些准则已经适用于 Tarte aux Fraises 单元格上, 它在已经 Recipes 示例应用中出来默认数据。然而, 如果一个用户给 Tarte aux Framboises 添加一条食谱, 例子可能可以或不可能给出预想的结果。

8.12.5 输出测试结果和数据的日子

你的脚本报告日志信息给 Automation instrument 工具, 而 instrument 工具收集并报告给你分析结果。

当编写你测试脚本时, 如果为了帮你诊断出现的任何故障的地方, 你应该尽可能的输出更多的信息。最低限度是当每个脚本开始和结束, 确定测试执行, 并记录通过和失败状态时, 你应该输出日志。这种最小记录在 UI 自动化的自动完成的。你只需要简单的使用你的测试脚本的名称来调用 `logStart`, 运行你的测试脚本, 然后合适的时候调用 `logPass` 或者 `logFail`, 如下面的代码那样。

```
var testName = "Module 001 Test";

UIALogger.logStart(testName);

//some test code

UIALogger.logPass(testName);
```

但是当你的脚本和控制器交互的时候，输出所发生的事情是一个很好的做法。无论你是验证应用程序的部分是否正确执行，或在追查 Bug 信息，很难想象如果有太多日志信息需要分析。为了避免这样，你可以使用 `logMessage` 来输出任何发生的地方，而且你甚至可以补充文本数据和截屏。

以下的示例代码扩展之前的输出，包括一个自由的日志输出信息和一个截屏。

```
var testName = "Module 001 Test";
UIALogger.logStart(testName);
//some test code
UIALogger.logMessage("Starting Module 001 branch 2, validating input.");
//capture a screenshot with a specified name
UITarget.localTarget().captureScreenWithName("SS001-2_AddedIngredient");
//more test code
UIALogger.logPass(testName);
```

在示例代码里面的截屏将会被指定文件名为 `SS001-2_AddedIngredient` 保存到 Instruments 应用里面。

注意：当前在 iOS 模拟器上不支持截屏功能。然而，如果你试图使用截屏功能，简化在日志里面输出一个失败的信息。

8.12.6 处理警告

除了验证你的应用警告是否工作正常，你的测试脚本应该包含测试在应用之外出现非预期的警告。比如，在检查天气或玩游戏的时候通常不应该出现文本信息。更糟糕的是，一个电话销售自动拨号可以获取你的电话号码就像你启动你的脚本那样。

处理外部产生的警告

尽管看起来可能有些自相矛盾，你的应用程序和你的测试应该期待你的应用程序运行时会发生意想不到的警报。幸运的是，UI Automation 包含了一个默认的处理程序，它可以让你脚本很容易显示外部产生的警告。你的脚本提供一个名为 `onAlert` 的警告处理函数，它会在警告产生的时候被调用，此时它会采取相应的措施，然后简单的返回警告给默认处理程序隐藏它。

以下示例代码举例说了一个非常简单的警告例子。

```
UIATarget.onAlert = function onAlert(alert) {
var title = alert.name();
UIALogger.logWarning("Alert with title '" + title + "' encountered.");
// return false to use the default handler
return false;
}
```

该处理程序所做的事情是输出一个警告发生的类型的消息然后返回 False。返回错误可以指导 UI Automation 的默认警告处理程序隐藏它。例如，在警告显示一个收到的文本消息时，UI Automation 简单的单击关闭按钮。

注意：默认处理程序达到警告的上限数量时将会停止隐藏新进的警告。在不太可能发生的情况下，你的测试达到此上限，你应该检查你的测试环境和程序找出可能出现的问题。

处理内部产生的警告

作为你应用的一部分，你有可能需要处理一些警告。在这些情况下，你的警告处理程序可能需要执行相应的响应并返回 True 给默认处理程序，告知该警告已经被处理。

下面的例子代码简单的扩展了基本警告处理程序。在输出该警告类型后，它测试警告是否是预期的。如果是，它单击 Continue 按钮，并返回 True 来跳过默认隐藏动作。

```
UIATarget.onAlert = function onAlert(alert) {
var title = alert.name();
UIALogger.logWarning("Alert with title '" + title + "' encountered.");
if (title == "The Alert We Expected") {
alert.buttons()["Continue"].tap();
return true; //alert handled, so bypass the default handler
}
// return false to use the default handler
return false;
}
```

该基础警告处理程序尽可能简单，在允许你的脚本继续运行时，它通常可以响应任何收到的警告。

8.12.7 检测和指定设备的方向

一个好的 iOS 应用应该在设备方向改变的时候做相应的调整，所以你的脚本应该

可以预知并测试这些改变。

UI Automation 工具提供了 `setDeviceOrientation` 方法来模拟一个设备方向的改变。该方法使用了列表 8-1 中的常量。

注意：至于设备方向的处理，该功能完全由软件来模拟。硬件特性比如原始加速度计数据无法使用 *UI Automation* 特性而且不受它影响。

Table 8-1 Device orientation constants

Orientation constant	Description
UIA_DEVICE_ORIENTATION_UNKNOWN	The orientation of the device cannot be determined.
UIA_DEVICE_ORIENTATION_PORTRAIT	The device is in portrait mode, with the device upright and the home button at the bottom.
UIA_DEVICE_ORIENTATION_PORTRAIT_UPSIDEDOWN	The device is in portrait mode but upside down, with the device upright and the home button at the top.
UIA_DEVICE_ORIENTATION_LANDSCAPELEFT	The device is in landscape mode, with the device upright and the home button on the right side.
UIA_DEVICE_ORIENTATION_LANDSCAPERIGHT	The device is in landscape mode, with the device upright and the home button on the left side.
UIA_DEVICE_ORIENTATION_FACEUP	The device is parallel to the ground with the screen facing upward.
UIA_DEVICE_ORIENTATION_FACEDOWN	The device is parallel to the ground with the screen facing downward.

与设备方向相反的是界面方向，它代表了你的应用界面在设备方向改变时所需要的旋转。需要注意的是在横屏模式下，设备方向和界面方向是相反的，因为旋转设备需要以相反的方向旋转内容。

UI Automation 提供了 `interfaceOrientation` 方法来获取当前界面方向。该方法使用了列表 8-2 列举的常量。

Table 8-2 Interface orientation constants

Orientation constant	Description
UIA_INTERFACE_ORIENTATION_PORTRAIT	The interface is in portrait mode, with the bottom closest to the home button.
UIA_INTERFACE_ORIENTATION_PORTRAIT_UPSIDEDOWN	The interface is in portrait mode but upside down, with the top closest to the home button.
UIA_INTERFACE_ORIENTATION_LANDSCAPELEFT	The interface is in landscape mode, with the left side closest to the home button.

UIA_INTERFACE_ORIENTATION_LANDSCAPERIGHT	The interface is in landscape mode, with the right side closest to the home button.
--	---

下面的示例代码改变设备的方向（该例中，修改为横屏先左），然后把方向修改为原来方向（竖屏）。

```
var target = UIATarget.localTarget();
var app = target.frontMostApp();
//set orientation to landscape left
target.setDeviceOrientation(UIA_DEVICE_ORIENTATION_LANDSCAPELEFT);
UIALogger.logMessage("Current orientation now " + app.interfaceOrientation());
//reset orientation to portrait
target.setDeviceOrientation(UIA_DEVICE_ORIENTATION_PORTRAIT);
UIALogger.logMessage("Current orientation now " + app.interfaceOrientation());
```

当然，一旦你修改了方向，最后你需要把方向修改回来。

当执行了一个包含改变设备方向的测试时，最好的做法是在开始测试之前设置设备的方向，然后在测试完成的时候把方向设置回来。这样可以保证你的脚本返回到已知状态。

你有可能已经注意到了示例代码中输出的方向信息。这些日志输出给你的测试脚本和测试人员提供了额外的验证，确保它们不迷失方向。

8.12.8 测试多任务

当用户点击 Home 按钮退出你应用的时候，或促使其他应用进入前台时，你的应用将会被暂停。为了模拟这种情形，UI Automation 提供了 `deactivateAppForDuration` 方法。你只需要调用该方法，并指定一个时长（以秒为单位），这样你的应用就可以被暂停，如下面的代码那样。

```
UIATarget.localTarget().deactivateAppForDuration(10);
```

这样简单的一行代码可以促使你的应用被暂停 10 秒钟，和用户点击了退出按钮而且 10 秒后返回应用的效果一样。

8.13 User Interface Instruments[用户界面相关]

以下的 instruments 工具为应用层事件收集数据。

8.13.1 Cocoa事件（Cocoa Events）

Cocoa Events instrument 工具记录通过 `NSApplication` 类 `sendEvent:` 方法发送

事件。该方法是分配事件给 Cocoa 应用的主要方法。你可以使用该 instrument 工具来把应用程序事件和其他应用程序行为关联起来，比如内存和 CPU 占有率等。该 instrument 工具运行在当个进程之上。它的实现使用了 DTrace 技术，并且可以导入 DTrace 脚本。

该 instrument 工具捕获被发送事件的类型。

跟踪面板可以被设置来显示以下任何数据信息：

- 栈深度 (**Stack depth**)
- 线程 ID (**Thread ID**)
- 事件种类 (**The Event Kind**)

对于任何调用，你可以打开扩展详细面板来查看该调用的栈跟踪，和具体发生的时间。

8.13.2 Carbon 事件 (Carbon Events)

Carbon Events instrument 工具记录由 Carbon Event Manager 里面的函数 `WaitNextEvent` 返回的事件。你可以使用该 instrument 来把应用的事件和其他应用的行为关联起来，比如内存和 CPU 占用率等。该 instrument 工具运行在单个进程之上。它的实现使用了 DTrace 技术，并可以导入 DTrace 脚本。

该 instrument 工具捕获发送事件的类型。

跟踪面板可以被设置来显示以下数据信息：

- 栈深度 (**Stack depth**)
- 线程 ID (**Thread ID**)
- 事件种类 (**The Event Kind**)

对于任何调用，你可以打开扩展详细面板来查看该调用的栈跟踪，和具体发生的时间。