JProfiler 使用说明

（版本号 1.0）

2008 年 1 月

文档更改历史记录

序号	主要更改内容	版本号	更改人	更改时间
1	建立文档	1.0		2008-01-11
2				
3				
4				
5				
6				
7				
8				
9				

目　录

1 JPROFILER'S START CENTER 5

1.1 OPEN SESSION 5

1.2 NEW SESSION 6

1.2.1 New Session 6

1.2.2 New server integration 7

1.3 CONVERT SESSION 17

1.4 OPEN SNAPSHOT 17

2 管理SESSION 18

2.1 APPLICATION SETTINGS DIALOG 18

2.1.1 session名 18

2.1.2 session类型 18

2.2 PROFILING SETTINGS 20

2.2.1 Adjusting call tree collection options 20

2.2.2 JAVA 子系统 23

3 监测视图 27

3.1 内存视图 27

3.1.1 所有对象（ALL objects） 28

3.1.2 记录的对象（Recorded objects） 29

3.1.3 分配调用树（Allocation call tree ） 29

3.1.4 分配热点视图（Allocation hot spots view） 29

3.1.5 类跟踪（Class tracker）.....30
3.2 CPU视图.....30
3.2.1 调用树视图（Call tree view）.....30
3.2.2 热点视图（Hot spot view）.....33
3.2.3 调用图（Call graph）.....34
3.3 线程视图.....40
3.3.1 线程历史视图.....40
3.3.2 线程监控视图.....41
3.3.3 死锁检测图形.....43
3.3.4 当前监控使用视图.....43
3.3.5 监控使用历史视图.....44
3.3.6 监控使用统计.....44
3.4 VM遥感监测视图.....45
3.4.1 Heap.....45
3.4.2 Objects.....45
3.4.3 Garbage collector.....45
3.4.4 Classes.....45
3.4.5 Threads.....45
4 IDE集成（ECLIPSE 3.X）..... 46

1 JProfiler's start center

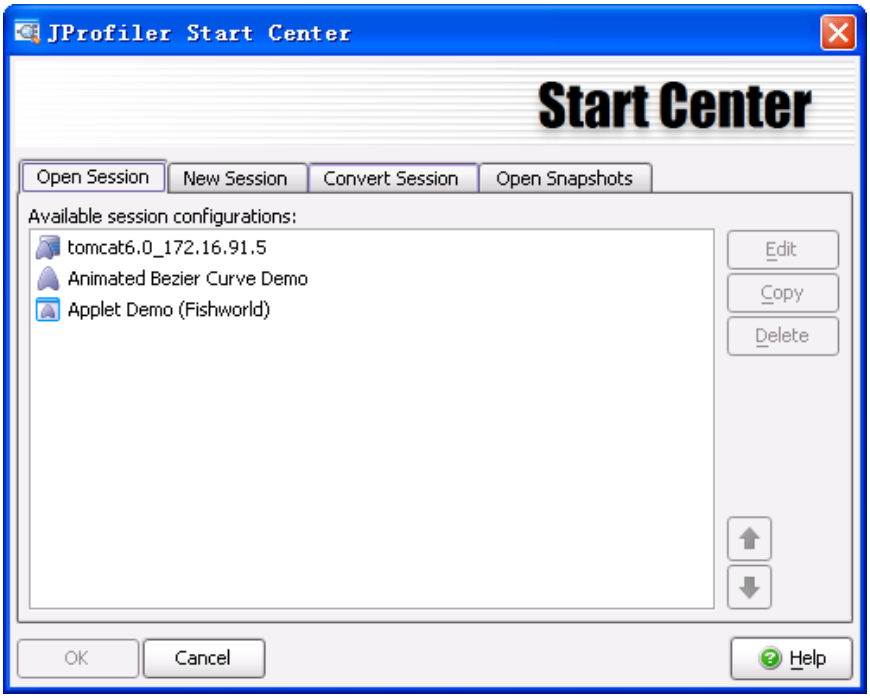
使用 JProfiler's start center，你可以创建新的会话，编辑已有会话或者打开已保存的会话。

在菜单中选择 session →start center ，有以下四个标签：Open session、New session、Convert session 和 Open snapshot。

也可以在工具栏中点击  图标打开 start center。

1.1 Open session

在 start center 中，打开 Open session 标签，在窗口中显示所有预先定义的会话，如图：



你可以选择一个会话，点击 OK 来打开预定义的会话。

会话也可以被修改、复制、删除和排序。

也可以菜单中选择 session → Open session，打开 Open session 窗口。

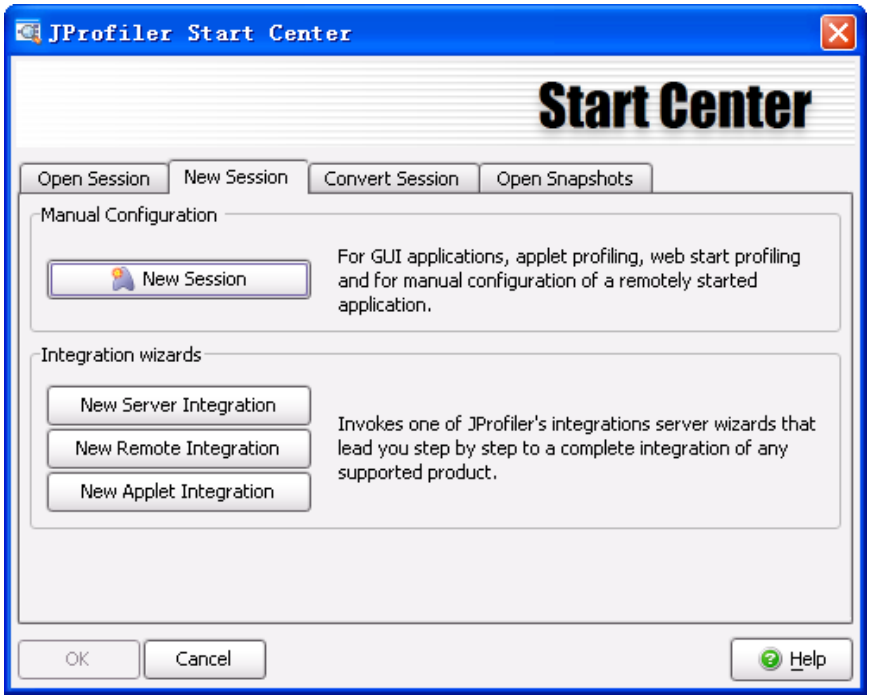
1.2 New session

会话能够通过两种途径创建：

✓ 人工配置：使用 [New session] 按钮手工配置一个新的 session，配置完成后，开始运行

✓ 通过集成向导：使用集成向导上的三个按钮：[New server integration] 、[New remote integration] 和 [New applet integration] 。

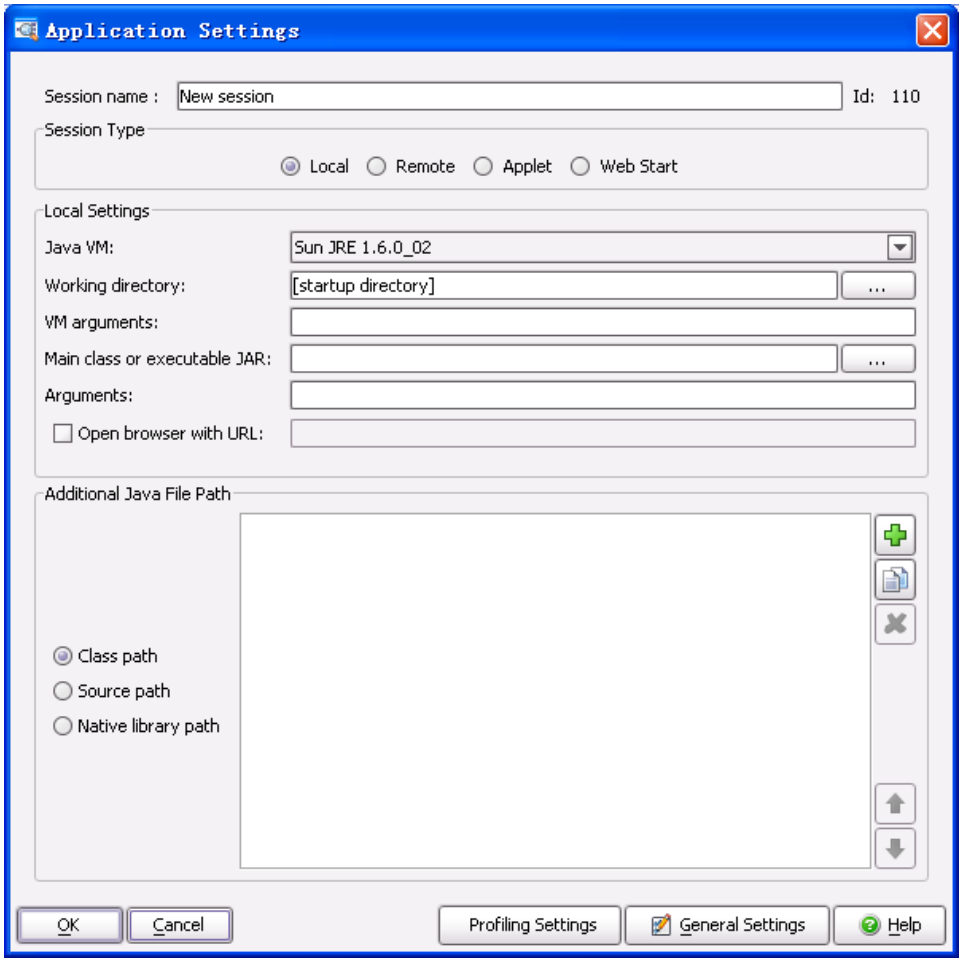
✓ 会话创建完成后可以立即运行。在 Open session 标签页中能够看到新建的会话。



1.2.1 New Session

点击New Session按钮，显示Application Settings窗体，关于Application Settings配

置，请参见2 管理session



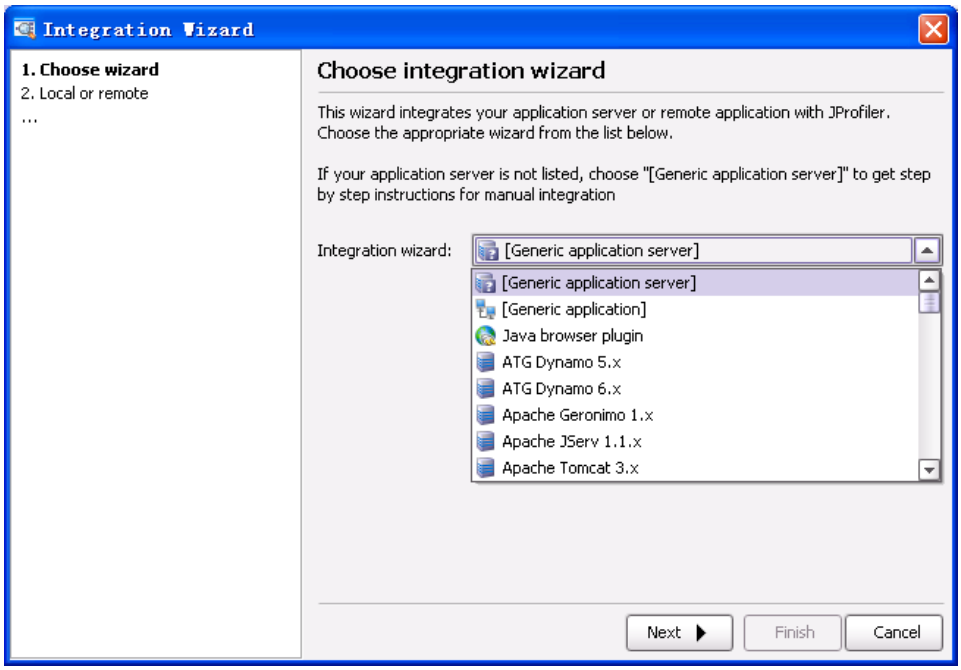
可以在菜单中选择 Session→New Session，打开 New Session 窗体

1.2.2 New server integration

点击 New server integration 按钮，打开集成向导，引导你将 JProfiler 与本地或远程的应用服务器进行集成，步骤：

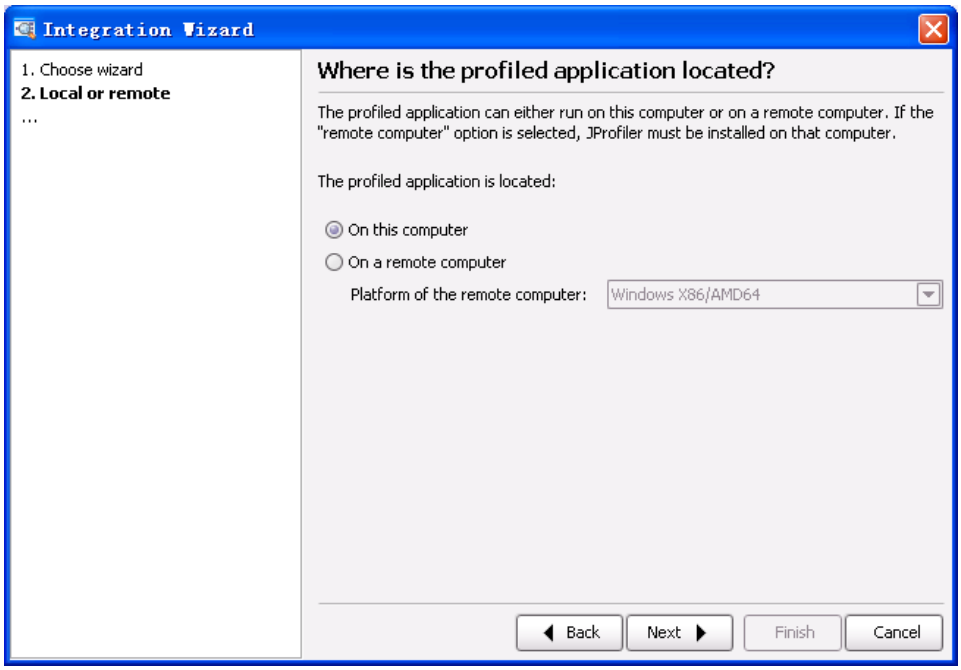
第一步、 选择需要集成的应用服务器

如果你所使用的应用服务器不在列表中，则选择“Generic application sever ”，并点击“下一步”；如图：



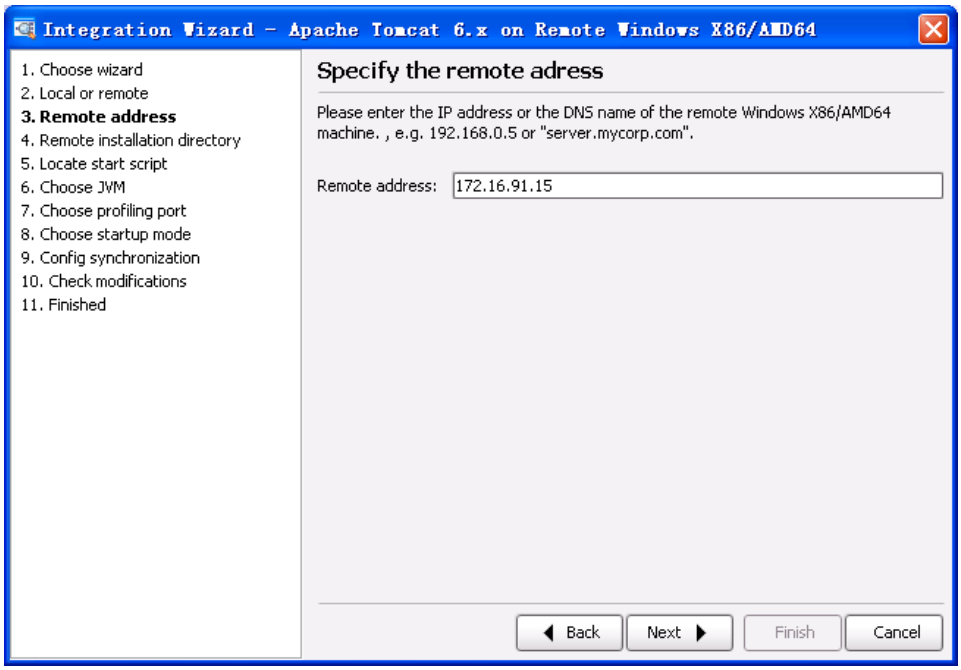
第二步、　选择要集成的应用服务器地址

选择要集成的应用服务器是本地的，还是远程的。如果你选择过程计算机，在选择的计算机上必须安装 JProfiler，并选择远程计算机的操作系统，点击“下一步”如图：



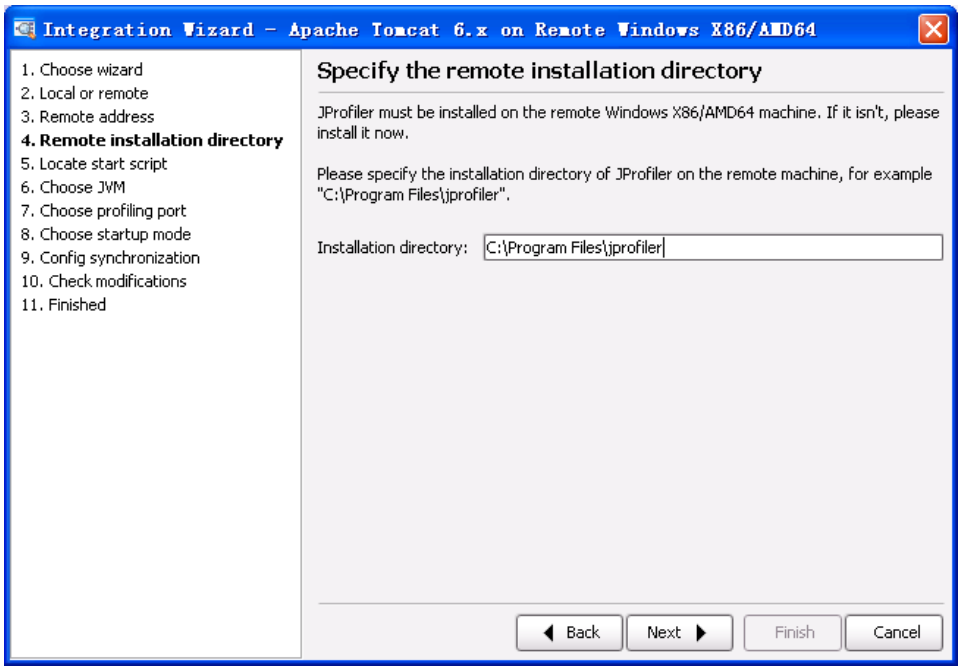
第三步、　指定远程地址

输入远程计算机的 IP 地址或 DNS 名称，点击下一步。如图：



第四步、　指定远程安装目录

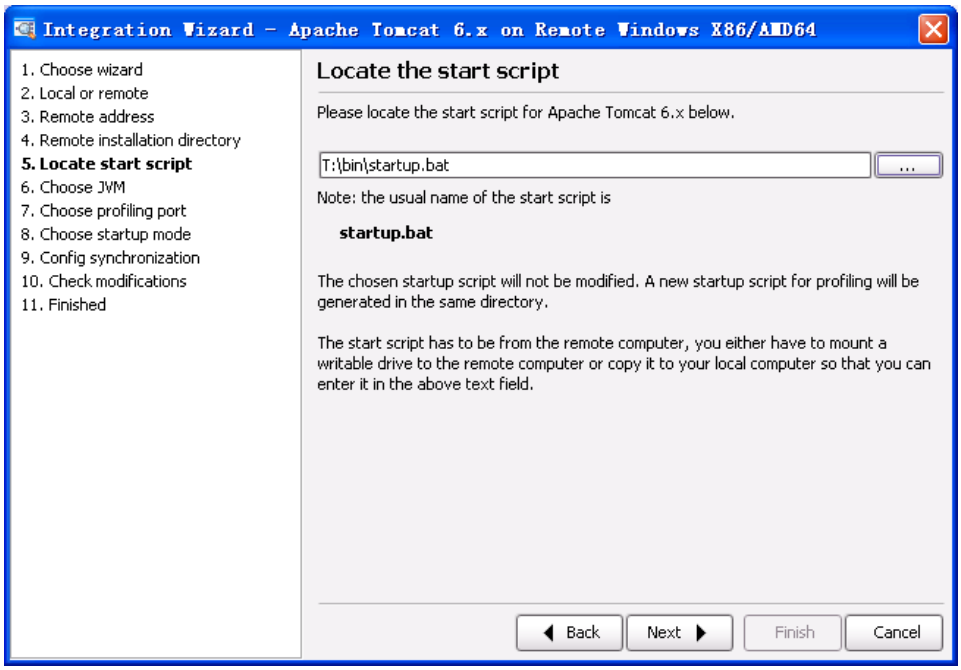
所选远程计算机上必须安装 JProfiler,如果未安装，请先安装。如果已经安装，请输入安装目录，点击下一步。如图：



第五步、　选择启动脚本

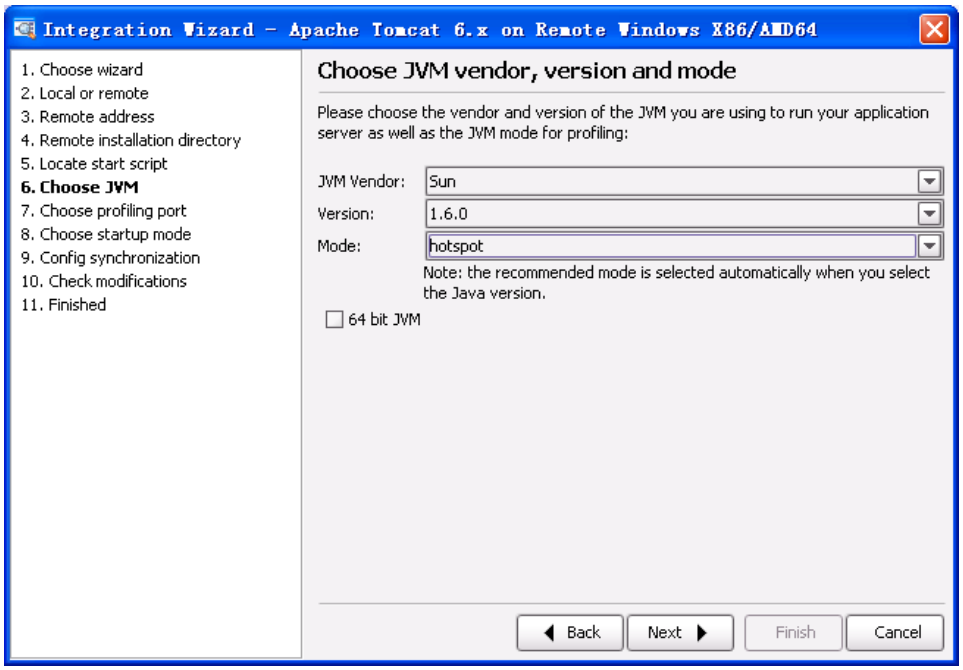
选择所集成的应用服务器的启动脚本，如，TOMCAT在WINDOW下常用的启动脚本为start.dat。如果脚本是远程的，你必须有远程计算机磁盘的可写权限，或者将启动脚本拷贝到本地。

批注 [U1]: 未试过



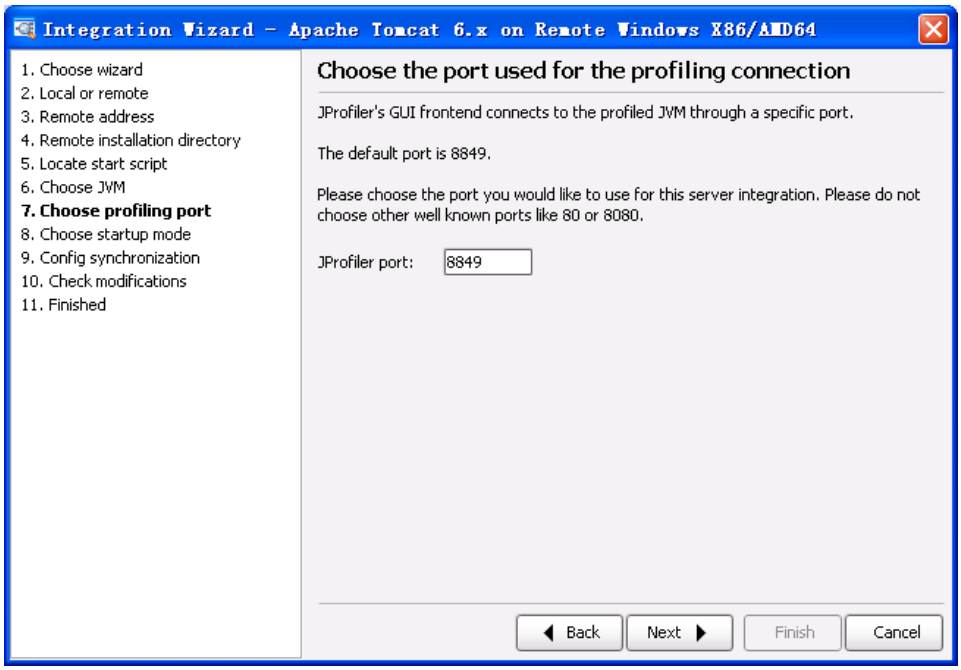
第六步、　选择 JVM 提供商，版本和模式

当你选择 JAVA 版本时，系统会自动选择推荐的模式



第七步、　选择监测的连接端口

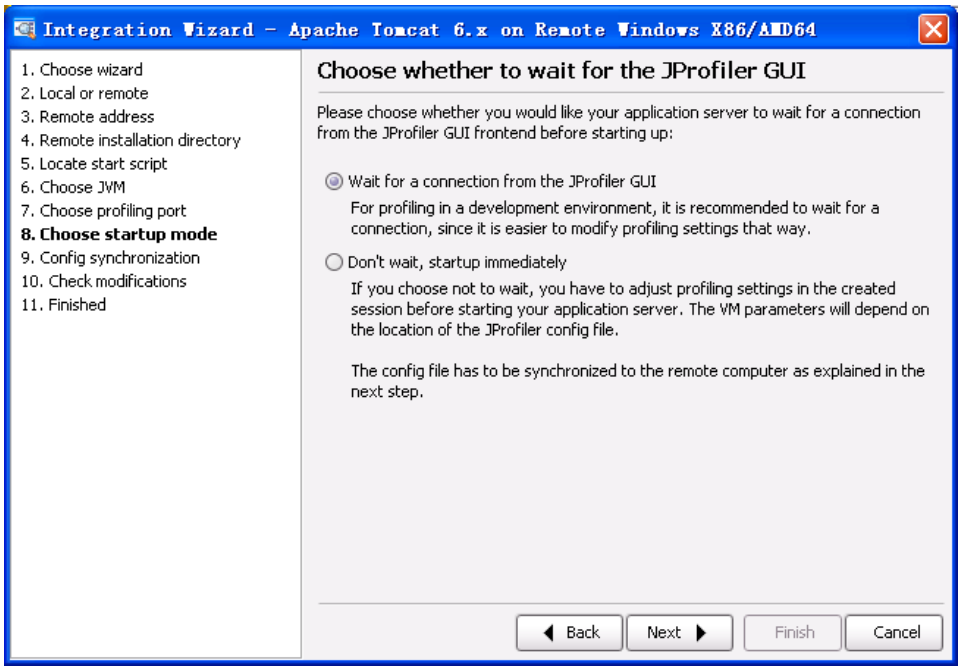
JProfiler GUI 前端通过指定的端口连接被监测的 JVM，默认端口为 8849。请不要选择常用的端口，如 80 或 8080。



第八步、　是否等待 JProfiler GUI

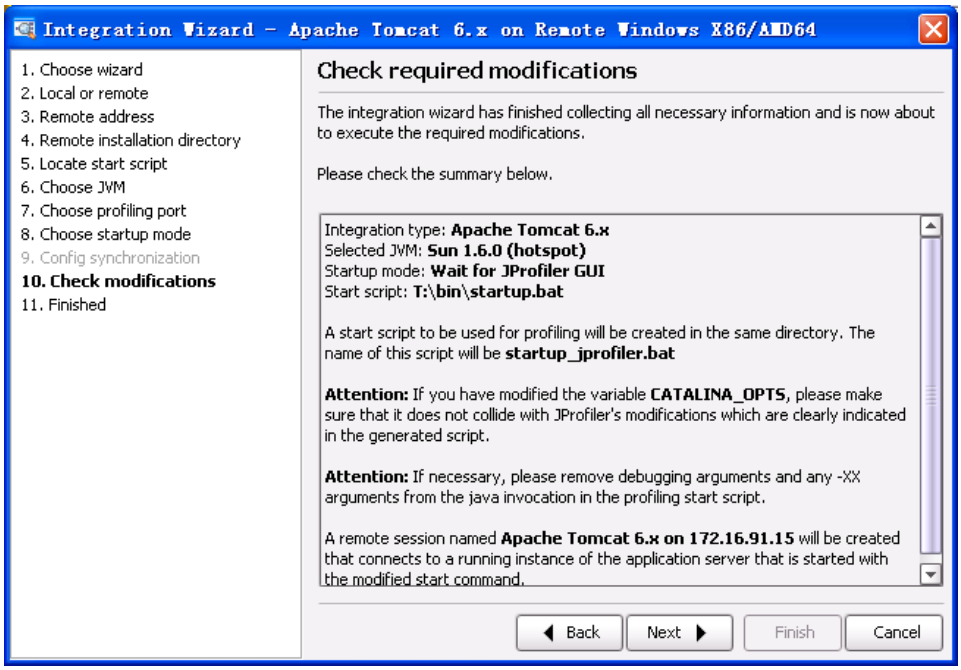
在开发环境，建立选择第一项，很容易修改监测设置；如果你选择不等待，在启动应用服务器之前要先配置监测设置。VM 参数依赖于 JProfiler 配置文件的位置；

配置文件要与远程计算机上的保持同步。



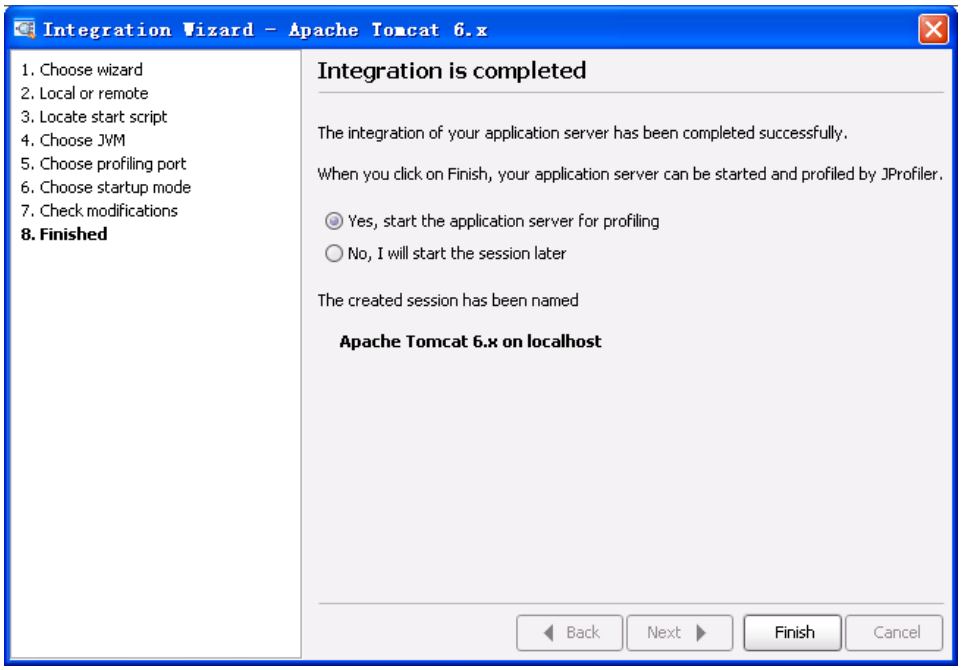
第九步、　检查需求通知

集成向导完成配置，所有必须的信息会进行通知，请检查，点击下一步。如图：



第十步、 完成

选择是否立即运行



1.3 Convert session

可能转换已经存在的本地的 session 成为远程session或者是offline profiling sessions

1.4 Open snapshot

可以通过打开*.jps 文件来打开原来保存过的 session

2 管理 session

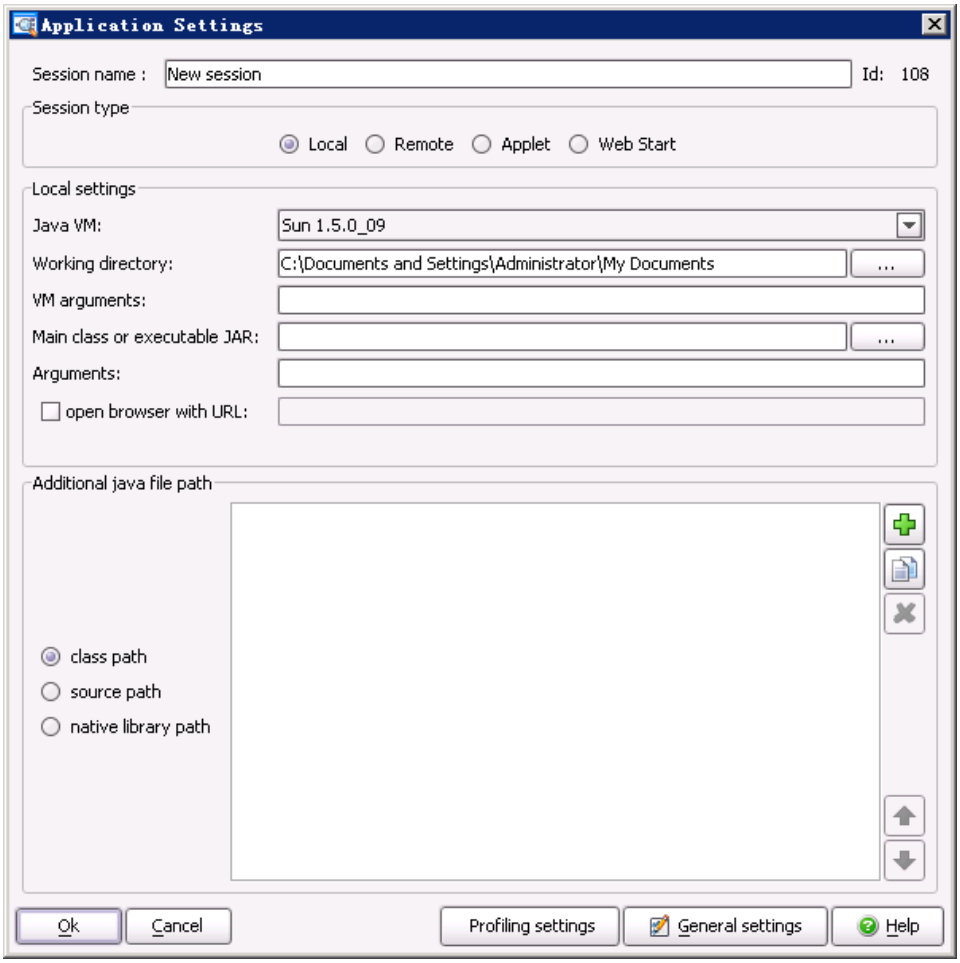
2.1 Application settings dialog

2.1.1 session 名

可以根据自己的需要给 session 命名

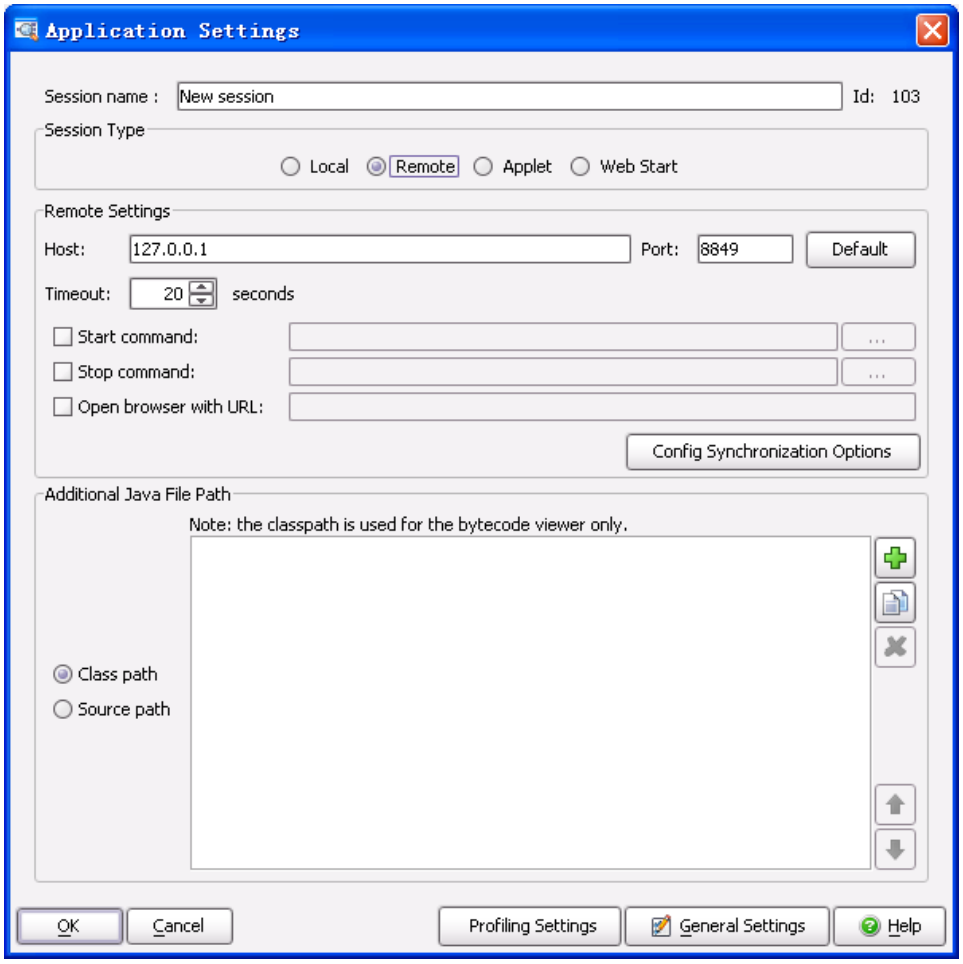
2.1.2 session 类型

2.1.2.1 Local sessions：



2.1.2.2 Remote sessions

可以连接由 JPofiler 剖析代理启动的应用。剖析代理监听的默认端口是 8849



2.1.2.3 **Applet sessions**

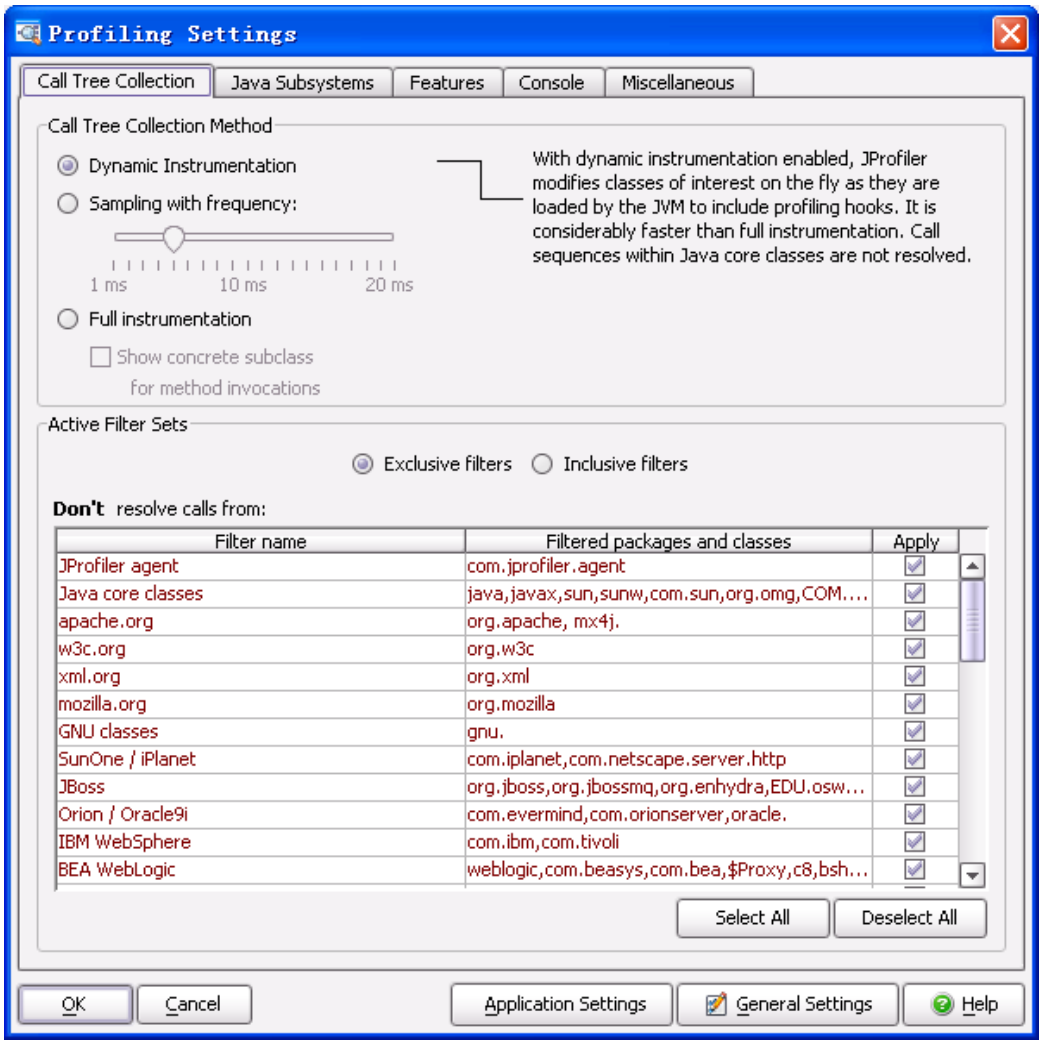
用来剖析 applets，你需要提供包含 applet 的 HTML 页面的 URL

2.1.2.4 **Web Start sessions**

也可以剖析 Java Web Start 应用，只需要给出 JNLP 文件的 URL 或者选择一个缓冲应用

2.2 Profiling settings

2.2.1 Adjusting call tree collection options



2.2.1.1 dynamic instrumentation（动态获取）

选择动态获取，在 JVM 加载类时，JProfiler 修改所有未过滤的类的字节码。JAVA 核心类(java.*)会自动被过滤掉，无法剖析。动态获取比取样式更加精确。

2.2.1.2 sampling

JProfiler 定期检查所有线程调用的堆栈数。即使不进行过滤，取样方式也会快很多，但精通性很低并且不能实时显示

取样周期可以在 1－20 ms 间调整。如果取样太频繁，会导致 CPU 占用过高。

2.2.1.3 full instrumentation（全部获取）

所有的方法调用都会被跟踪，这样可以监控到JAVA核心类(java. *),但比动态获取慢，尤其是 1.4 JVMs 。在 1.5 JVMs (JVMTI)中不能使用。

使用全部获取你可以随意地显示具体的类调用的方法而不是某个方法在哪个类中实现。

2.2.1.4 活动过滤设置(active filter sets)

- ✓ **Exclusive(独占的)**

在下表中，你为会话选择哪些过滤集是活动的。这些过滤器指定CPU监控的结束点。在包或类中匹配过滤集中的某一个，进一步的调用其它被过滤的类将在调用第一个被过滤的方法时失败。在被过滤的类中，方法是不透明的并且在call tree view中标为红色.

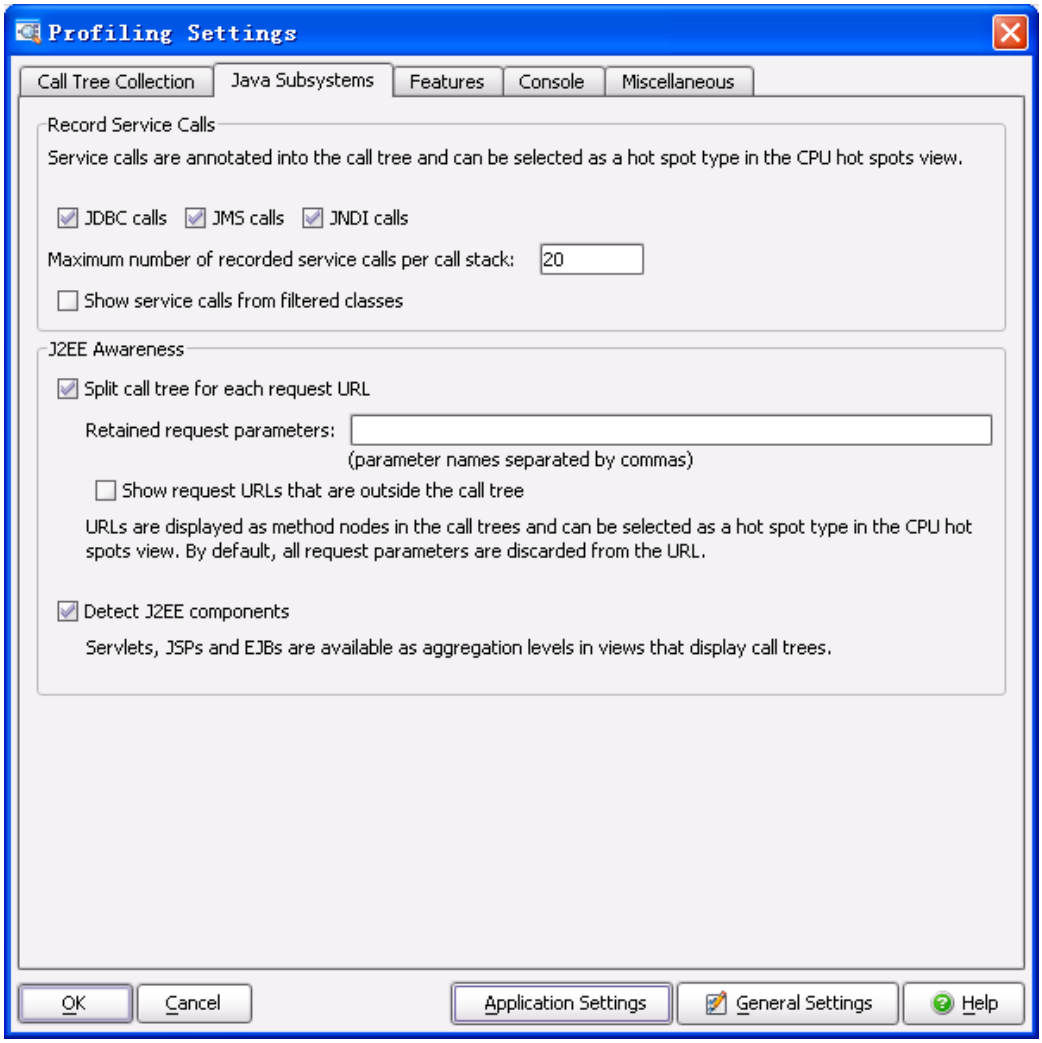
过滤集可以在general settings的"Filter sets" 标签中设定。

- ✓ **Inclusive(包含的)**

输入逗号分隔类和包的过滤器。在 CPU 监控中，只有符合这些包或类的调用才能够被显示。

例如，如果你指定com.mycorp.,com.othercorp. 作为一个包含的过滤器，一个调用 com.mycorp.MyClass.myMethod() 和它做的所有调用才会被度量和显示。而所有从 com.mycorp.MyClass.myMethod() 发起的调用都不属于包 com.mycorp 和 com.othercorp ，它们是不透明的并且在call tree view.标识为红色。

2.3 JAVA 子系统



在此标签下，你可以选择 JProfiler 如何记录 J2EE 相关的 JAVA 子系统。

✓ 记录服务调用

JProfiler 使用仪器监控几个J2EE服务层并记录象服务调用的执行时间等语义数据。记录仪器不需要依赖，可以为所有驱动或服务提供商工作。服务调用在call tree view中有解释并且可以在hot spot view选为热点。

下面的服务类型可以被单独激活：

- JDBC calls
- JMS calls
- JNDI calls

✓ J2EE awareness

如果选择了Split call tree for each request URL， JProfiler将分析调用servlets 和JSPs的URL。针对每个URL，在call tree中都创建一个新的节点。URL也能够在hot spot view选择成了一个热点类型。这样做的话，你可以在单独的页面或请求中区分性能问题。

缺省，只有没有查询参数的URL会使用上面的分离进程。为了在call tree中保留选择的参数，你可以在retained request parameters文本框中输入要保留的参数。

如果要显示所有的请求，请选择 Show request URLs that are outside the call tree.

2.4 features

✓ Enabled profiling features

在此标签中，你可以调整监控的属性，以提高软件执行速度，减少内存使用。

- **Disabled profiling features**
  - **Disable call tree collection**

When you record CPU data or allocations, JProfiler collects information about the call tree. You might want to record allocations without the overhead of recording the allocation call stacks: If you don't need the allocation view in the heap walker, the allocation call tree and the stack trace information in the monitor usage views, you can switch off call tree collection. This will speed up profiling considerably and reduce memory usage.

  - **Disable monitor contention views**

if you are not interested in monitor contention events, you may switch data collection off by selecting this checkbox to lower the memory consumption of the profiled application. If monitor contention views are **enabled**, the following settings govern the level of detail for the monitor contention views:

    - **Monitor contention threshold**

Select the minimum time threshold in microseconds (µs) for which a monitor contention (i.e. when a thread is blocking) is displayed in the monitor usage history view.

- **Monitor waiting threshold**

  Select the minimum time threshold in microseconds (µs) for which a monitor wait state (i.e. when a thread is waiting) is displayed in the monitor usage history view.

- **Allocation call tree**

The information depth of the allocation call tree and the allocation hot spots view is governed by this setting.

- **Live objects**

By default, only live objects can be displayed by the allocation views. Class-resolution is enabled.

- **Live and GCed objects without class resolution**

Live and garbage collected objects can be displayed by the allocation views, depending on the selection in the allocation options dialog. Class-resolution is disabled, i.e. class selection in the allocation options dialog will not work in this setting, only the cumulated allocations of all classes and array types can be displayed. This setting consumes more memory than the first setting and adds a considerable performance overhead.

- **Live and GCed objects**

Live and garbage collected objects can be displayed by the allocation views, depending on the selection in the allocation options dialog. Class-resolution is enabled. This setting consumes more memory than the other settings and adds adds a considerable performance overhead.

Select **record object allocation time** if you would like to be able to

- use the time view in the heap walker
- sort objects by allocation time in the reference graph and the data view of the heap walker.
- See allocation times for the current objects in the reference graph and the data view of the heap walker.

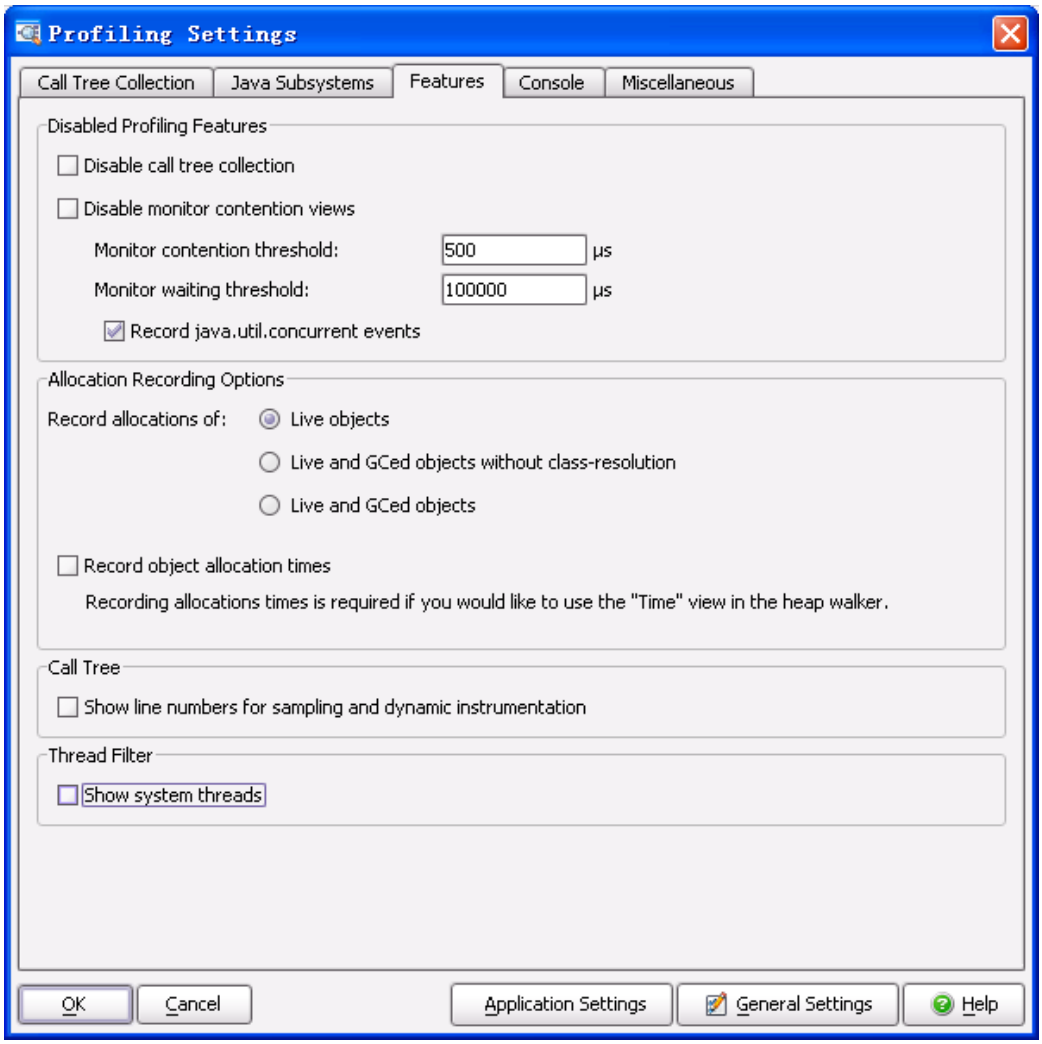
This setting consumes more memory for recorded objects.

- **Call tree**

By default, JProfiler does not resolve line numbers in call trees. If you enable show line numbers for sampling and dynamic instrumentation, line number resolution will be enabled for the call tree collections modes of "Sampling" and "Dynamic instrumentation". For "Full instrumentation", line number resolution is not available.

If the aggregation level is set to "methods" and a method calls another method multiple times in different lines of code, line number resolution will show these invocations as separate method nodes in the call tree and the allocation call tree. Backtraces in the hotspot views will also show line numbers.

Note that a line number can only be shown if the call to a method originates in an unfiltered class.



3 监测视图

3.1 内存视图

JProfiler 的内存视图部分可以提供动态的内存使用状况更新视图和显示关于内存分配状况信息的视图。所有的视图都有几个聚集层并且能够显示现有存在的对象和作为垃圾回收的对象。

3.1.1 所有对象（ALL objects）

所有对象视图显示所有加载的类的列表和在堆上分配的实例数。只有 Java 1.5 (JVMTI)才会显示此视图。要查看特定时间段对象的分配，并记录分配的调用堆栈，请使用“记录的对象视图”

有一个集合体等级选择器，你可以在以下几种类型中切换：

- ✓ 类（classes）：每一行显示一个单独的类，这是默认的集合体类型
- ✓ 包（packages）：每一行显示一个单独的包，子包不包含在内。在这个集合体等级内，表是树形的。你可以点击树节点，查看其包含的类
- ✓ J2EE 组件（ J2EE components）：每一行是一个 J2EE 组件。此模式类似类模式中的过滤器，能够让你快速检查应用中加载的 J2EE 组件。

在表中显示三行，可以排序

- ✓ 名字：根据集合体等级的不同，分别显示类、包或 J2EE
- ✓ 实例数：
- ✓ 大小：显示所以分配实际的总大小。只包括相应指针的大小，不包含引用数组和实例的大小。以字节显示。只包括对象数据，不包括类使用的 JVM 大小，也不包含类数据和本地变量大小。

更新频率可以在profiling settings dialog的miscellaneous tab中设置。所有对象视图的更新频率是根据堆上的对象数来自动调整的，如果堆上有太多对象，所有对象视图的计算变得昂贵，所以更新频率会降低。你可以随时刷新以获取最新数据。

你可以在class tracker增加包或类。如果类跟踪器没有记录，则开始记录class tracker中配置的所有类；如果正在记录不同的对象类型，所以的记录数据被清空

你可以冻结所有的视图让所有的视图对象保持静态

你可以标记当前值并显示差异值。

3.1.2 记录的对象（Recorded objects）

记录的对象视图显示所有已记录对象和数组的列表，包括堆上分配的实例数。只有已记录的对象在此视图中显示。配置记录的细节请参见 memory section overview

如果选择了某个包或类，你可以从记录对象视图跳转到分配调用树和分配热点。方法是右键点击，选择要跳转的视图。

你可以在class tracker增加包或类。如果类跟踪器没有记录，则开始记录class tracker中配置的所有类；如果正在记录不同的对象类型，所以的记录数据被清空

记录的对象视图可以根据对象的活动状态进行过滤：

- ✓ 活动对象（Live objects）
  只显示当前在内存中的对象
- ✓ 垃圾回收对象（Garbage collected objects）
  被显示被回收的对象
- ✓ 活动的和垃圾回收对象（Live and garbage collected objects）
  显示所有被创建的对象

右键选择 Change view mode 或者使用 View->Change view mode 切换三种模式

你可以标记当前值并显示差异值。

3.1.3 分配调用树（Allocation call tree ）

分配调用树视图 显示一棵请求树或者方法、类、包或对已选择类有带注释的分配信息的 J2EE 组件

3.1.4 分配热点视图（Allocation hot spots view）

分配热点视图显示所选类的对象被分配在哪儿的方法列表。分配到至少占总数 1% 的方法才会被显示。方法可以根据active filter sets设置进行过滤。此视图和CPU section里的hot spots view视图有些类似，只是显示的是分配的类的实例数和数组而不是时间度量

对于每个热点都可以显示它的跟踪记录树。

3.1.5 类跟踪（Class tracker）

3.2 堆遍历(Heap Walker)

3.2.1 类(Classes)

显示所有类和它们的实例

3.2.2 分配(Allocations)

为所有记录对象显示分配树和分配热点

3.2.3 索引(References)

为单个对象和“显示到垃圾回收根目录的路径”提供索引图的显示功能。还能提供合并输入视图和输出视图的功能

3.2.4 数据(Data)

为单个对象显示实例和类数据

3.2.5 时间(Time)

显示一个对已记录对象的解决时间的柱状图

3.3 CPU 视图

3.3.1 调用树视图（Call tree view）

调用树显示一个线程从上向下调用树。可以根据过滤设置向上或向下过滤显示。

JProfiler 自动检测 J2EE 组件并在调用树中显示相关的节点。使用不同的图标显示不同的 J2EE 组件类型

- ✓ servlets：黄色倒心形
- ✓ JSPs：蓝色倒心形

- ✓ EJBs：红色倒心形

对于 JSPs 和 EJBs, JProfiler 显示名为：

- ✓ JSPs：JSP 源文件路径
- ✓ EJBs：EJB 界面名称

如果 URL 可以被细分，每个 URL 请求使用一个特殊的符号创建一个新的结点以 URL：做前缀，后面跟上细分后的 URL 请求

调用树视图集合体等级选择有四种：

- ✓ 方法，也是默认等级。每个节点都是一个方法调用。特殊的 J2EE 组件方法有他们自己的图标和显示名，上面讲到过。真实的类名以方括号括起
- ✓ 类：每个节点是一个单独的类。特殊的 J2EE 组件方法有他们自己的图标和显示名，上面讲到过。真实的类名以方括号括起
- ✓ 包：每个节点是一个单独的包。不包括子包
- ✓ J2EE 组件：每个节点是一个 J2EE 组件，如果组件有独立的显示名，真实类名省略。

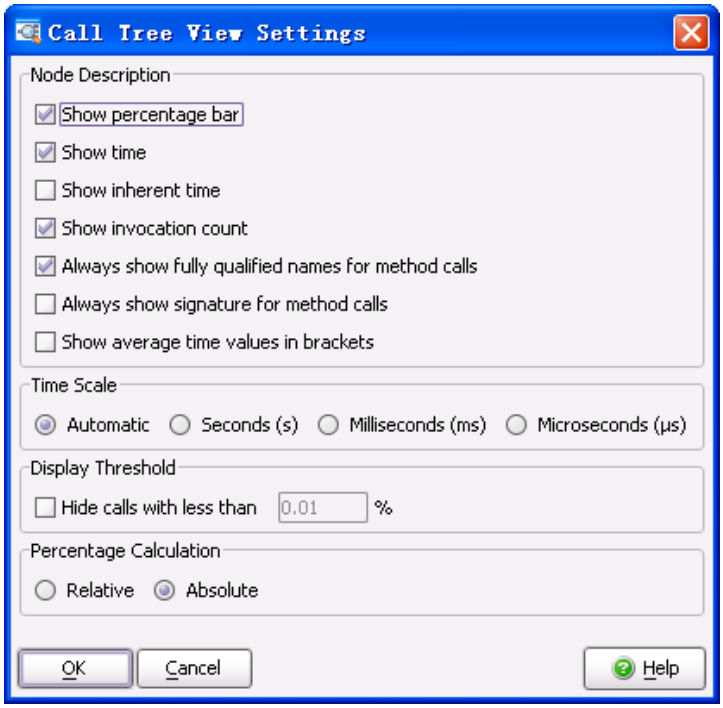
调用树不显示 JVM 中的所有方法，只显示：

- ✓ 未过滤的类：根据你配置的过滤器设置没有过滤的类
- ✓ 未过滤的类的第一级调用： 未过滤的类对过滤的类的第一级调用，对过滤类的深一级的调用不显示。过滤的节点在左上角以红色标记。
- ✓ 线程实体方法： 方法 Runnable.run()和主方法总是被显示的，无论是否过滤

一个特殊的结点是桥结点，本来在视图中不显示，但其子孙节点在视图中显示。

如果在 view settings中设置了percentage bar模式，调用树中的每个节点都会显示一个百分比条，显示当前节点的内部时间占总时间的百分比，当前节点包括其所有子孙节点和红色高亮的部分

在view settings可以设置以下内容：



1） 节点描述

- 百分比数：考虑树的根或调用节点
- 总时间度量：ms 或 µs 。包括调用其它节点的总时间。
- 内部时间度量：ms 或 µs。内部时间，不包括调用非过滤类
- invocation count：显示在此路径中节点被调用的频繁程度。
- 是否显示方法调用全名
- 是否显示方法调用签名（显示参数）
- 是否在括号内显示平均时间

2） 时间范围

- 自动
- s
- ms
- us

3） 显示极限

- 隐藏小于一定百分比的调用

4） 百分比计算

- 绝对
- 相对

名称显示依赖集合体等级：方法、类、包和 J2EE 组件

行数在以下情况下会显示：行数显示的是调用（invocation）的行数，还不是方法本身的行数

- ✓ 集合体等级　　为方法
- ✓ 在profiling settings设置中选择了显示行数
- ✓ 调用类为未过滤类

你可以选择任何一个节点，然后选择 View->Set as root ，将选择的节点改为根节点。如果在 view settings 里，百分比基数设置为“total thread time”，百分比会按照新的根节点重新计算。选择 View->Show all 返回所有视图

你可以停止或重启 CPU 数据获取来清空调用数，也可以冻结所有视图让调用数保持静态

3.3.2 热点视图（Hot spot view）

热点视图显示选择类型的调用列表。截去了占总时间小于 0.1%的点。

热点类型"hot spot type"下拉中可选择，包括两种：

- ✓ 方法调用（method calls）
  - method calls (show filtered classes separately)

  从方法调用中计算显示的热点，被过滤的类计算自己的热点，默认为此模式。
  - method calls (add filtered classes to calling class)

从方法调用中计算显示的热点，被调用的类被加到调用类上，除非是线程实体方法 (run 和 main 方法)

- 根据你选择的集合体等级不同，方法热点也会变换。

✓ J2EE 相关的调用

- JDBC calls

显示的热点为JDBC 调用。需要在profiling settings中设置为可用。

- JMS calls

显示的热点为JMS 调用。需要在profiling settings中设置为可用。

- JNDI calls

显示的热点为JNDI 调用。需要在profiling settings中设置为可用。

- URL invocations

显示的热点 URL调用。需要在profiling settings中设置为可用。在profiling settings中，你需要指定是所有的URL都被显示，还是调用一个未过滤类的URL才会被显示，默认为后者。

3.3.3 调用图（Call graph）

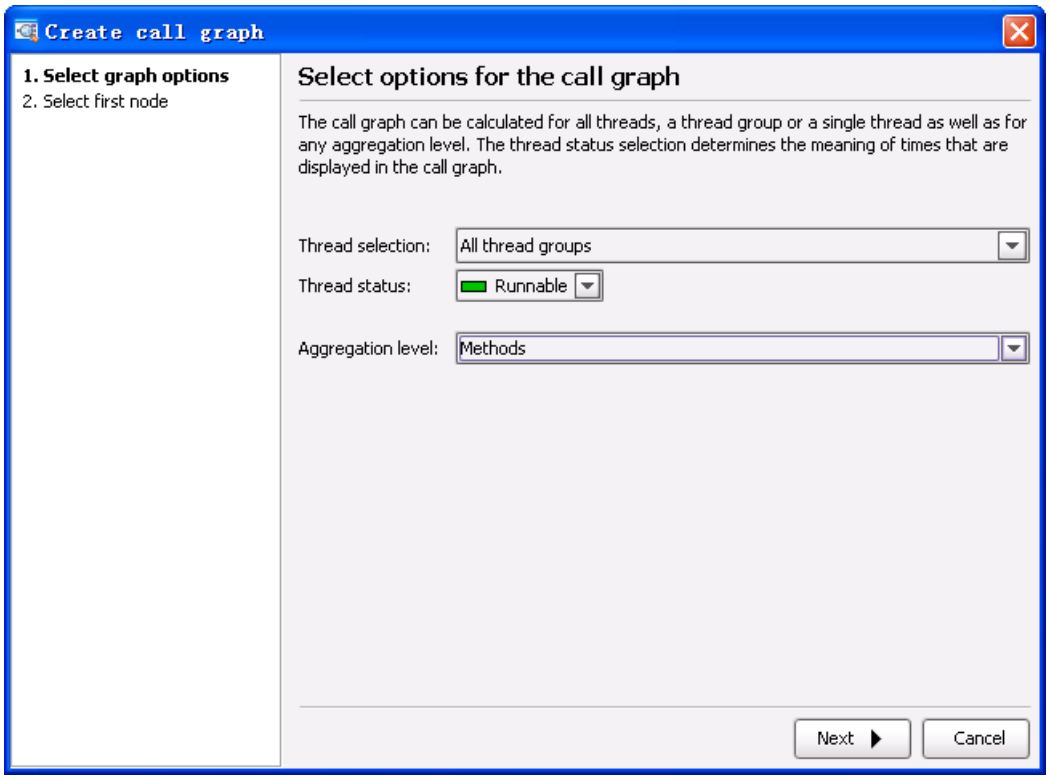
调用图静态地显示所选择节点计算出线程调用图。节点可以为方法、类、包或 J2EE 组件。

计算调用图，点击工具栏中的 Generate graph 或者选择 View->Generate graph 。

在图形计算之前，会启动调用图向导。结果图形是静态的，并能够重新计算。调用图向导会记录你最近一次的选项。

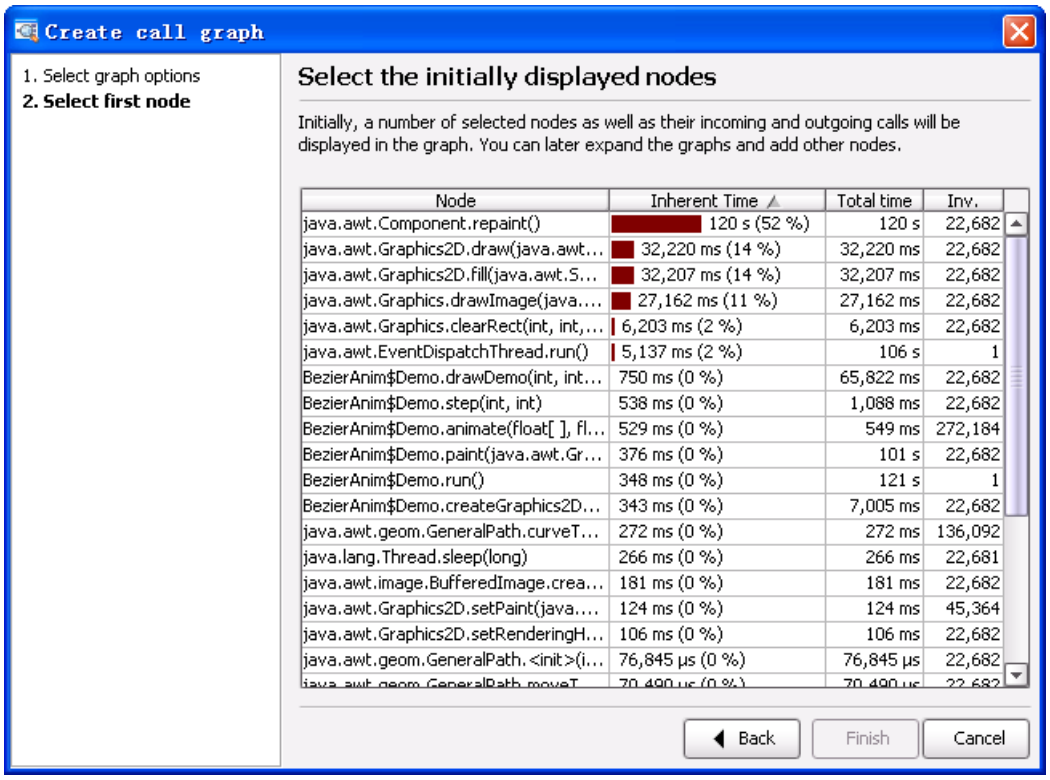
调用图向导如下：

第一步：选择图选项

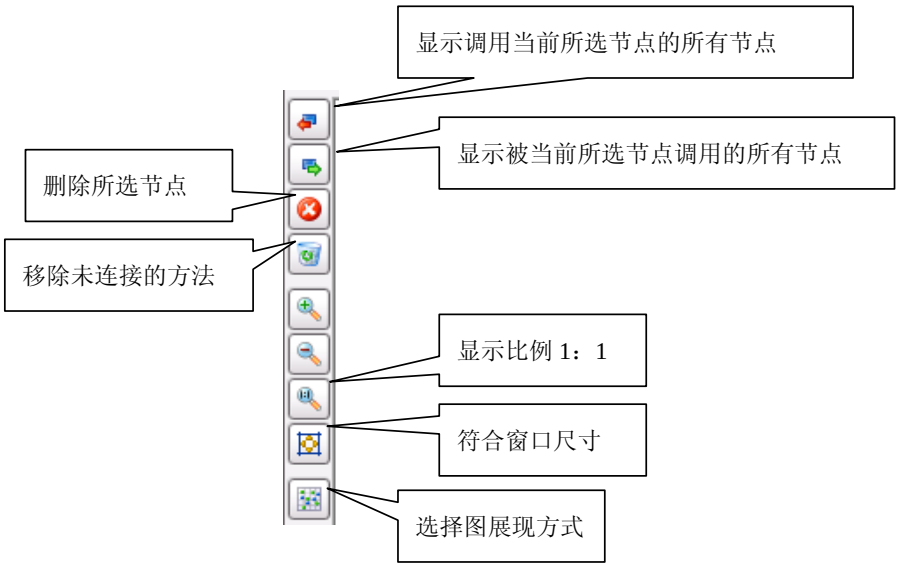


调用图能够为所有线程、一个线程组和单个线程的所有集成等级生成调用图。线程状态选择可以控制在调用图中显示的时间含义。如在线程状态中，选择了 Runnable，在调用图中，Total Time 表示的就是线程处于 Runnable 的时间。

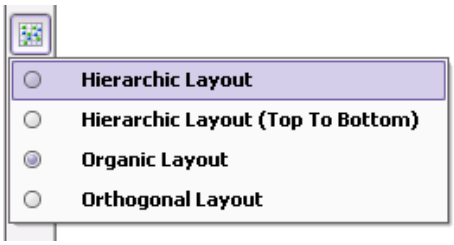
第二步：选择第一个节点：选择生成调用树的首节点，点击完成，生成调用树。



系统提供工具栏，可以对图形进行不同的显示操作。



选择图展现方式，包括四种方式：



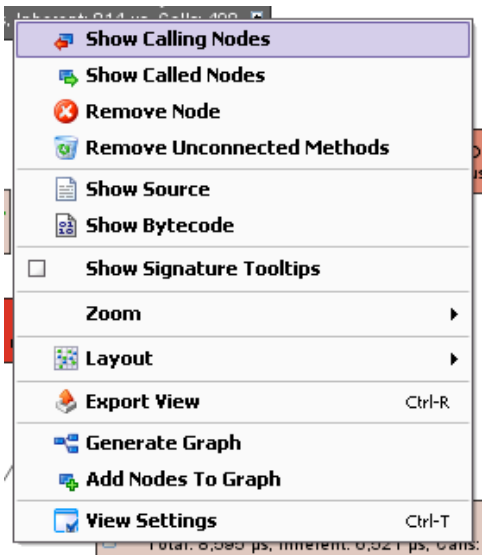
1） 按等级展现

2） 按等级展现（由上到下）

3） 组织式展现

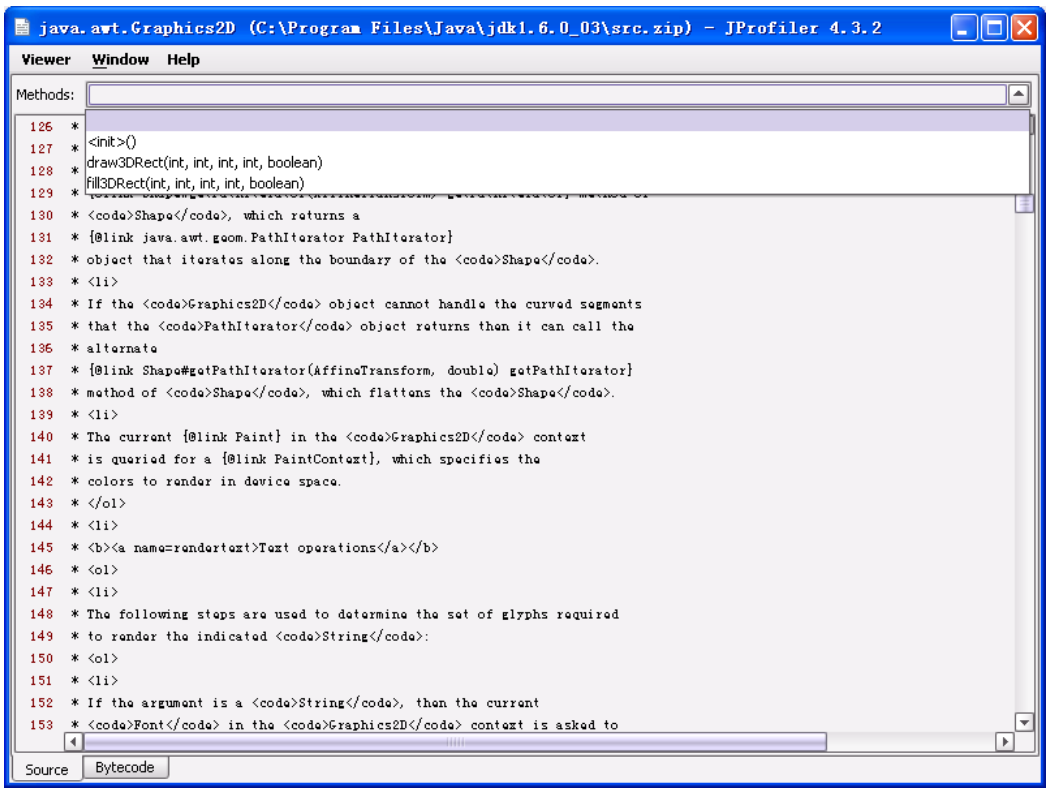
4） 正交展现

还可以选择一个节点，右键，显示的图形如下：

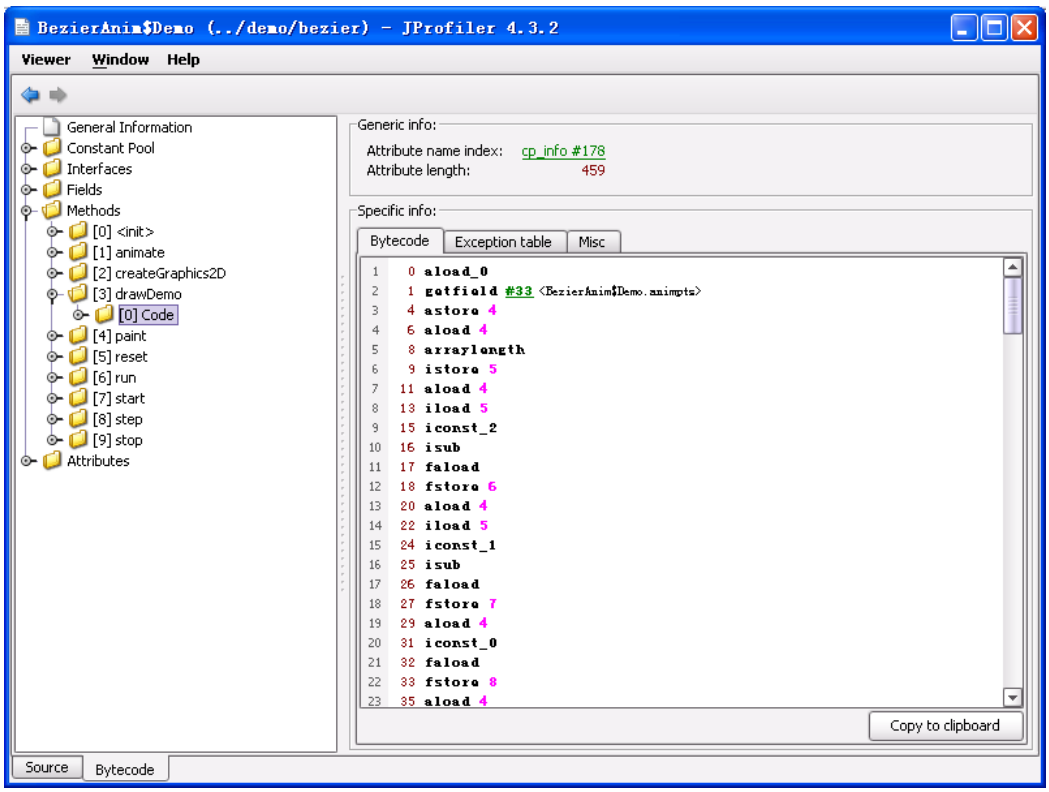


与屏幕右方显示的工作栏中基本操作一样，一些不同的，下面说明。

Show Source:显示此方法或类的源代码，如下图。



Show Bytecode:显示此方法或类的字节码，如下图：



3.4 线程视图

3.4.1 线程历史视图

按照线程开始的顺序显示 JVM 中所有线程状态的详细历史信息

在视图左手点，线程的名字固定显示，其它部分是滚动度量工具，在水平轴上显示时间。时间轴的开始时间与 JVM 的第一个线程的时间保持一致。每个活动的线程用带颜色的线标明，从线程开始到线程结束。颜色标识线程的状态：

✓ 绿色

绿色表明线程正在运行并能接收 CPU 时间。不表明线程正在消耗 CPU 时间，只表明线程准备运行并且没有阻塞或睡眠。线程被分配了多少 CPU 时间，依赖于不同的其它因素，如总的系统负载，线程优先级和调度的运算法则

✓ 橙色

橙色表示线程在等待。线程正在睡眠并等待计时器或其它线程唤醒

✓ 红色

红色表示线程阻塞。线程尝试进入同步代码区或由其它线程控制的同步方法

✓ 蓝色

亮蓝色表示线程在 Net I/O 操作，线程在等待 JAVA 库的网络操作完成。在线程监听 socket 连接或者等待读写数据到 socket 中时，会产生这种状态。

在视图的顶部，有一个线程过滤器，你可以按以下方法进行过滤：

✓ 活动状态 liveness status

活动的和死线程 Both alive and dead threads

- 只显示活动的线程
- 只显示死线程

✓ 名称

在文本框中，你可以输入线程的全名或部分名称进行过滤。也可以使用通配符("*" and "?")选择线程组。可以用逗号隔开多个过滤项进行过滤，如 AWT-, MyThreadGroup-*-Daemon.

线程历史视图有两种显示模式：

✓ 固定时间范围 fixed time scale

此种模式下，时间轴的刻度一直保持不变，如果显示超出屏幕范围，可以使用滚动条查看，如果视图是在自动跟踪模式下，刚总是显示当时时间下的图形。

也可以使用 zooming in 或 zooming out 来调整显示的范围。

✓ 固定窗口范围 scale to fit window

此种模式下，在当前视图中会显示整个时间段的图形。Zooming 在此模式下不能使用

3.4.2 线程监控视图

显示当前运行的线程列表以及相关的时间和状态信息。

显示的六列：

- ✓ 名称 Name

显示线程名称，如果线程没有被特别命名，则使用 JVM 提供的名称。想让此视图更加有用，最好将你自己创建的线程使用 setName() 进行命名。

- ✓ 组 Group

显示与此线程相关的线程组的名称

- ✓ Start time

显示线程开始的时间，时间是根据 JVM 中第一个线程创建的时间还计算的。

- ✓ End time

只有在 view settings 设置了显示死线程时才能看到这一列，显示线程死去的时间，或者线程还活着但成为空线程的时间，时间是根据 JVM 中第一个线程创建的时间还计算的。

- ✓ CPU time

显示线程消耗的 CPU 时间

Note: CPU时间一列，只有在profiling settings中的Miscellaneous栏中设置CPU time type 为Estimated CPU times ，并且当你记录CPU数据时，CPU时间才会被度量，否则CPU时间一栏总是空的。

如果你的系统和 JVM 不支持线程特定的 CPU 时间报告，这栏也是空的

- ✓ Creating thread

  显示线程的名字和创建此线程的线程组。

- ✓ Status

  显示线程的状态，相当于线程历史视图中的状态报告

如果你监控的是 JAVA1.5 或以上版本（JVMTI），在屏幕的上半部分就显示上面的表，屏幕的下半部分显示所选线程的线程创建堆栈跟踪。堆栈跟踪只有线程创建时记录 CPU 数据才会显示。

3.4.3 死锁检测图形

JVM 中所有死锁的图形化显示.

正常的状态如果没有死锁，就显示"No deadlocks detected"

死锁根据以下情况进行分析：

- ✓ 在 JAVA 平台上建立的最初的同步机制，如：使用同步的关键词
- ✓ 在 java.util.concurrent 包中的锁机制，不使用对象的监控而是不同的实现机制

死锁检测图形有以下特征：

- ✓ 在死锁中的线程以紫色的矩形表示。矩形包括以下信息：
  - ■ 线程名
  - ■ 线程组（括号括起来）
- ✓ 在死锁中的监视器以灰色的矩形表示，包括以下信息：
  - ■ 监控的类
  - ■ 可以用来继续跟踪的监控 ID
- ✓ 监控器的所有权用实线箭头表示。箭头指向从线程到监控器。想要了解详细信息，把鼠标停留到箭头上，可以看到提示窗口
- ✓ 导致线程死锁的阻塞原因使用虚线箭头来表示。前头指向从阻塞线程到线程想进行的监控器

3.4.4 当前监控使用视图

显示当前等待和阻塞的操作

显示以下 6 列：可排序

- ✓ 时间 Time

  事件起始时间
- ✓ Duration

  事件持续时间，事件必须还在进行中

- ✓ Type

  事件类型， "waiting" 或"blocked"中的一种

- ✓ Monitor ID

  在特定的监控实例上识别多个事件的 ID

- ✓ Monitor class

  监控器的类。如果没有 JAVA 对象与此监控器相关联，刚显示 [raw monitor]

- ✓ Waiting thread

  事件中正在或过去在等待的线程

3.4.5 监控使用历史视图

显示监控中等待和阻塞的操作

3.4.6 监控使用统计

显示监控使用的统计信息

点击工具栏上的 Calculate statistics，或者选择 View->Calculate statistics。

在统计开始前，会打开一个monitor usage statistics options对话框，统计结果表是静态的，并能重新计算。

包级别的统计表包含下面 5 列：

- ✓ Monitors/Threads/Classes

  在统计对象框中按分组标准显示名称

- ✓ Block count

  在此监控组中，阻塞操作的频繁程度

- ✓ Block duration

  在此监控组中，所有阻塞操作的累积的待续阻塞时间

- ✓ Wait count

  在此监控组中，等待操作的频繁程度

- ✓ Wait duration

  在此监控组中，所有等待操作的累积的待续等待时间

3.5 VM 遥感监测视图

3.5.1 堆(Heap)

显示最大的堆大小以及堆中已使用的和未使用的空间大小。可以显示线性图和区域图

3.5.2 对象(Objects)

显示堆上对象的总数，分为数组和非数组。可以显示线性图和区域图

3.5.3 垃圾回收(Garbage collector)

显示垃圾回收活动，包括对象释放的一条线和对象移动的一条线。只显示已记录的对象，如果没有记录对象，此视图不可用

3.5.4 类(Classes)

显示 JVM 调用的类的总，分为过滤类和非过滤类

3.5.5 线程(Threads)

显示 JVM 中活着的线程总数，分为活动的线程和不活动的线程。

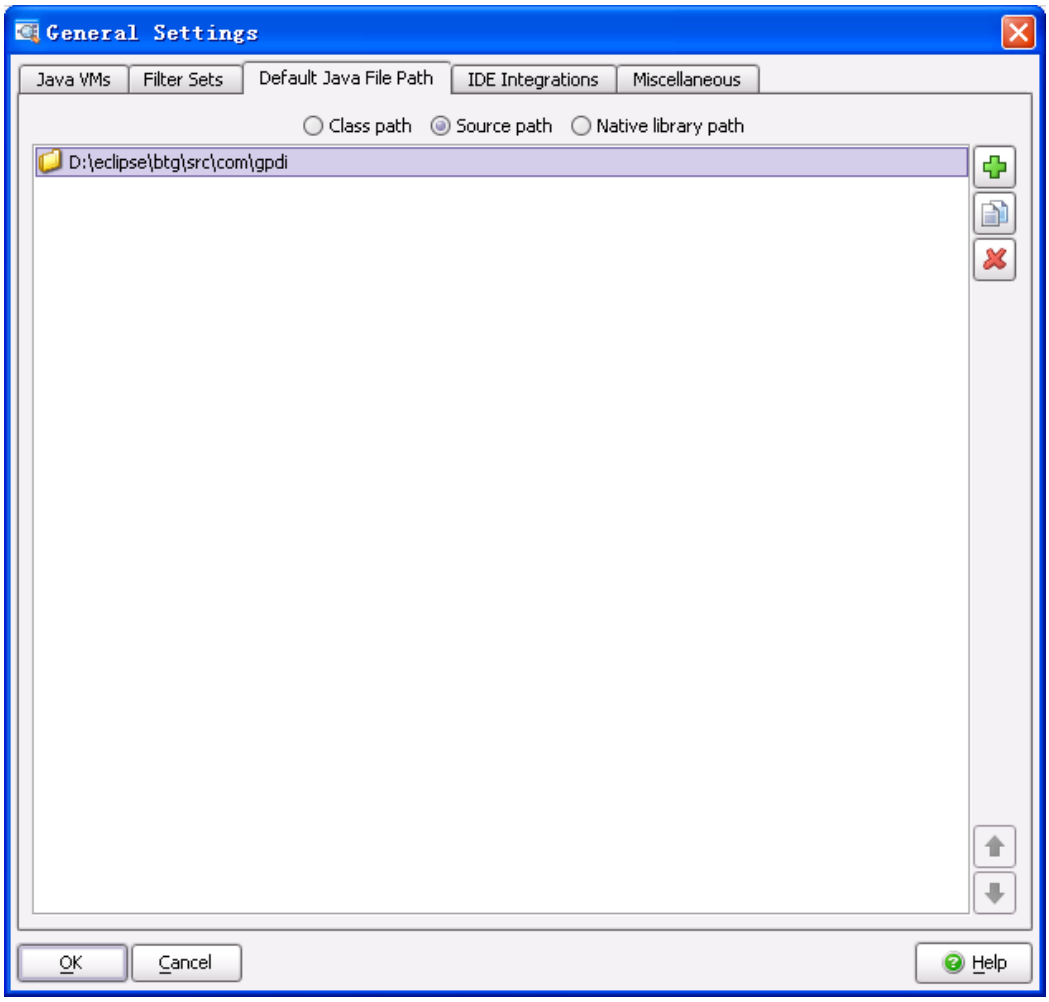
4 操作方法：

4.1 视图导出

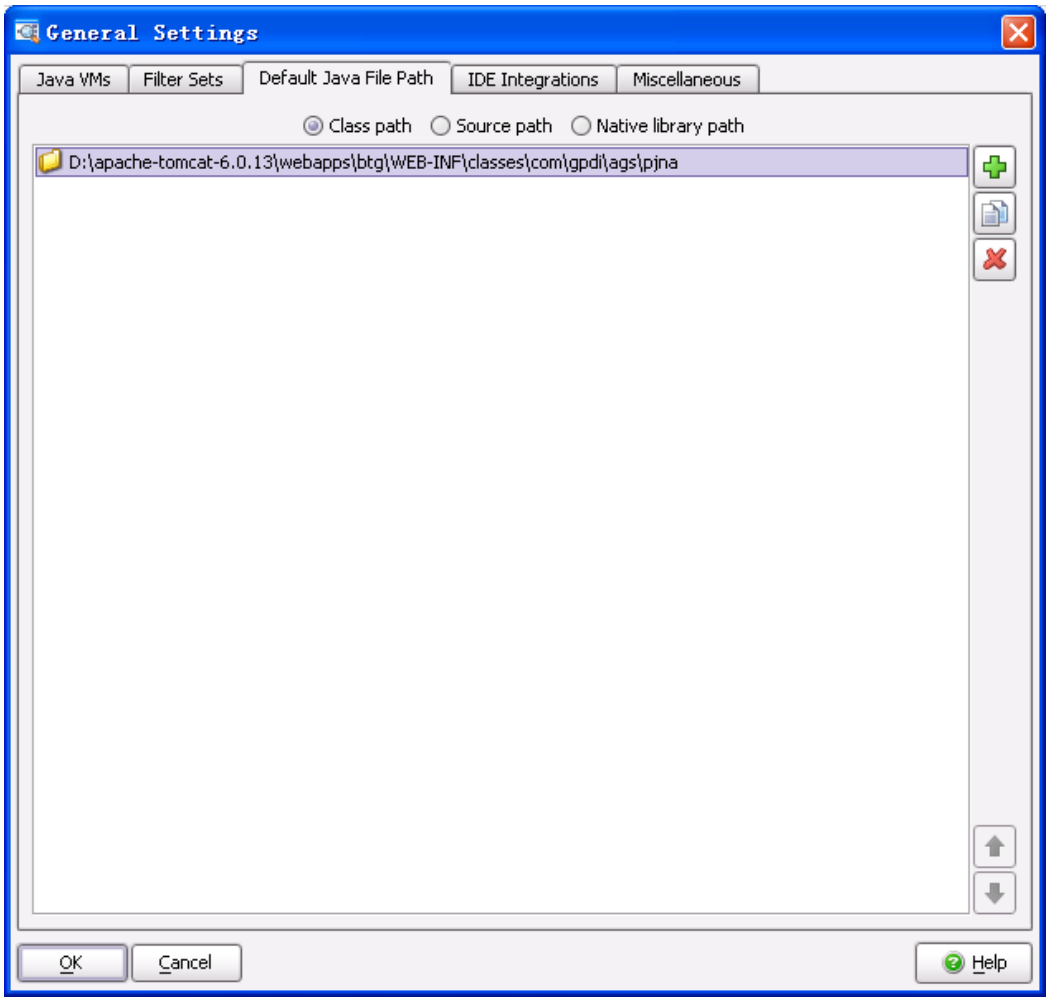
视图导出可以导出生成 HTML 文件或 CSV 文件，并可设定导出后是否立即打开文件。

4.2 查看源代码和字节码

查看源代码：需要在 Session→General Setting →Default Java File Path 下的 Source Path 中设定源代码路径，设定后要重启 JProfiler。如下图：



查看字节码：需要在 Session→General Setting →Default Java File Path 下的 Class Path 中设定发布的类的路径，设定后要重启 JProfiler。如下图：



4.3 监控设置模板

在会话开始前，会打开监控设置模板，显示一系列为各种情况预先配置好的多种监控设置模板，可以下拉选择。

可以在下拉框中选择 [Customize profiling settings and filters]打开监控设置对话框。见 2.2 Profiling settings

在 Startup 一栏里，你能够选择启动时是否立即记录 CPU 数据或分配数据。在很多监控里对应用的开始阶段不感兴趣。对于大的应用服务器，在开始时不记录会节省大量

内存，加快启动速度

- ✓ record CPU data on startup

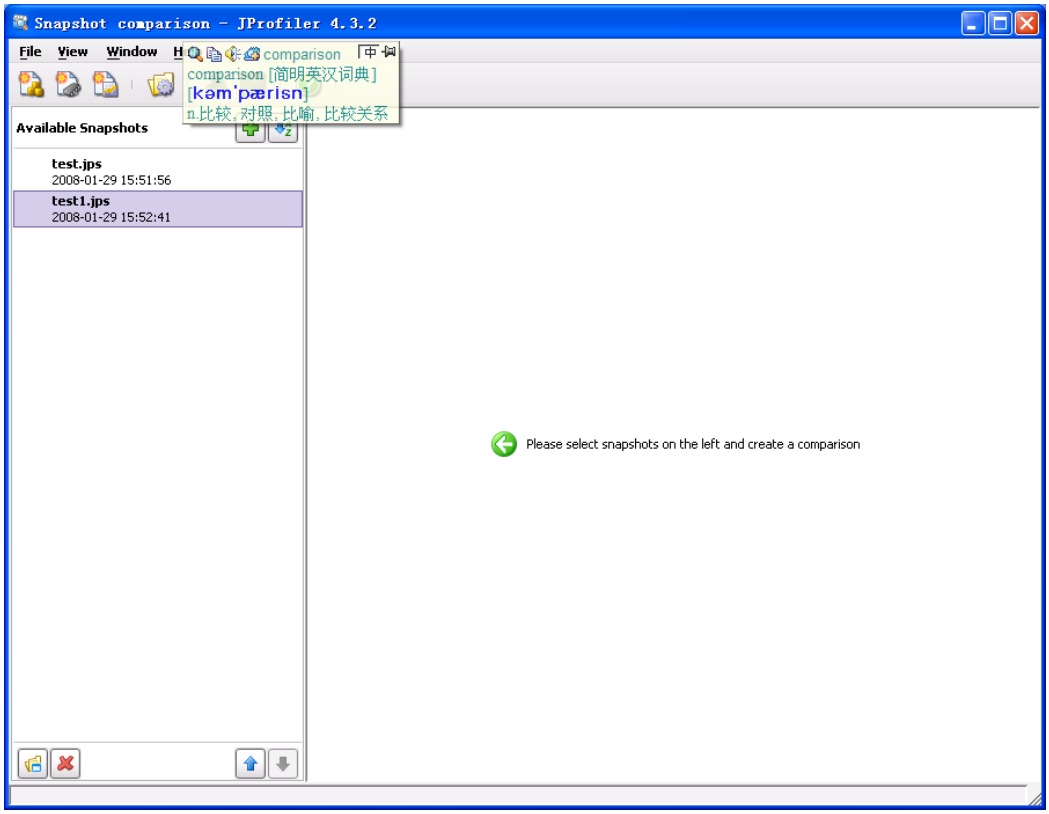
invocations view 和 **hot spots view** 将立即显示数据

- ✓ record allocations on startup

recorded objects view 将立即显示数据

5 在新的窗口中比较快照

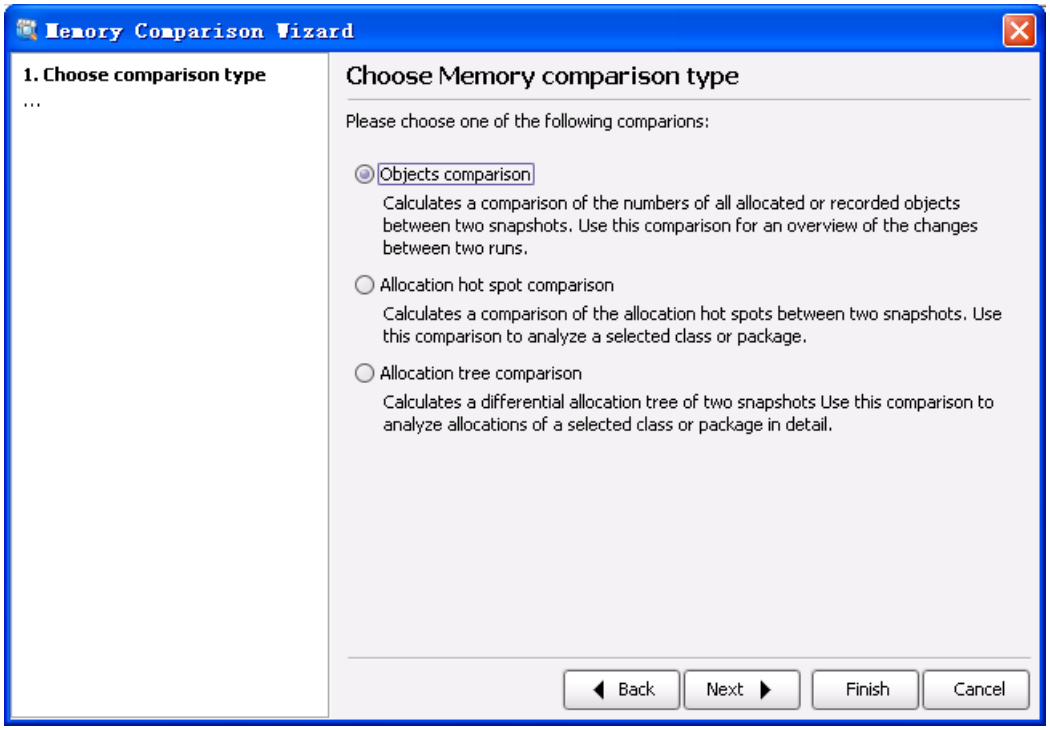
当前的视图可以保存下来，选择 session-->save snapshot 或者使用快捷方式 ctrl+s。多个快照可以进行比较，选择 session-->compare snapshots in new window，打开一个新的窗口，如图：



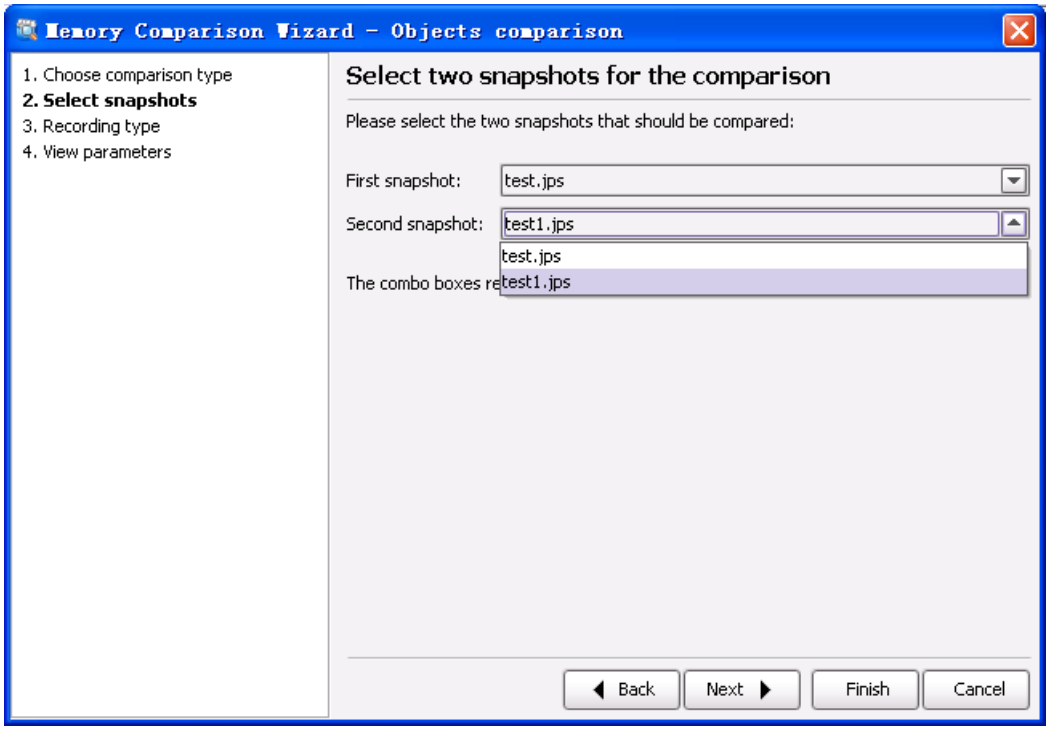
5.1 创建内存比较

选择 File→Create Memory Comparison，或使用快捷键 F5。

第一步：选择比较类型


Memory Comparison Wizard
1. Choose comparison type
...
Choose Memory comparison type
Please choose one of the following comparions:
Objects comparison
Calculates a comparison of the numbers of all allocated or recorded objects between two snapshots. Use this comparison for an overview of the changes between two runs.
Allocation hot spot comparison
Calculates a comparison of the allocation hot spots between two snapshots. Use this comparison to analyze a selected class or package.
Allocation tree comparison
Calculates a differential allocation tree of two snapshots Use this comparison to analyze allocations of a selected class or package in detail.
Back
Next
Finish
Cancel

第二步：选择快照



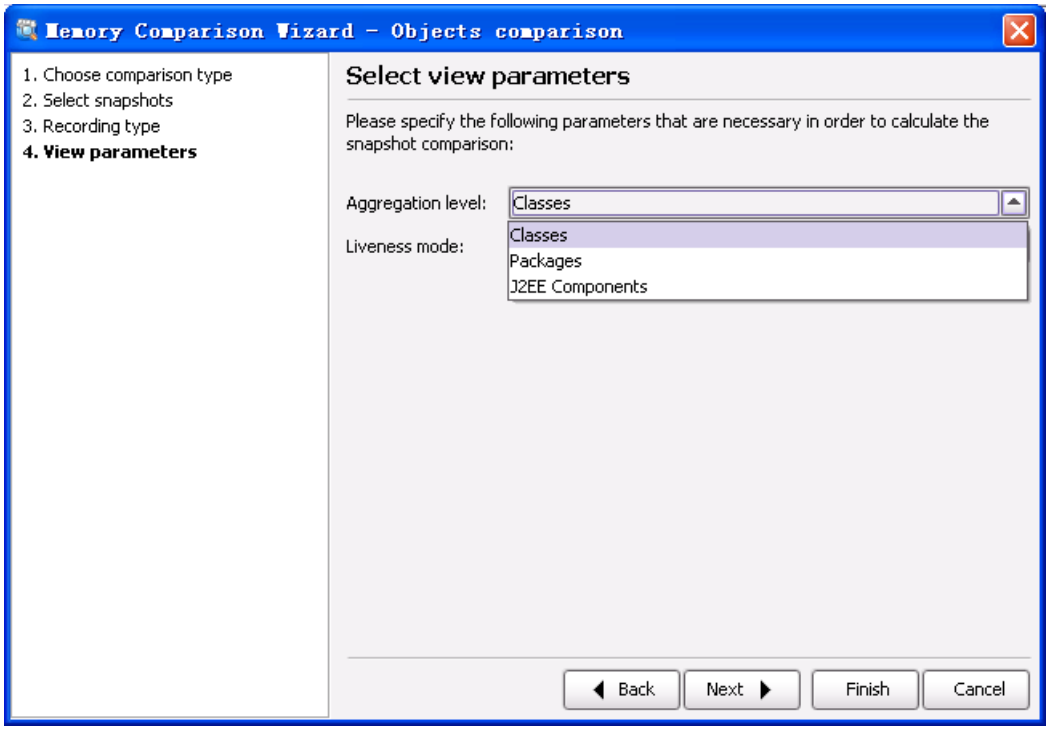
5.1.1 对象比较

第三步：选择记录类型

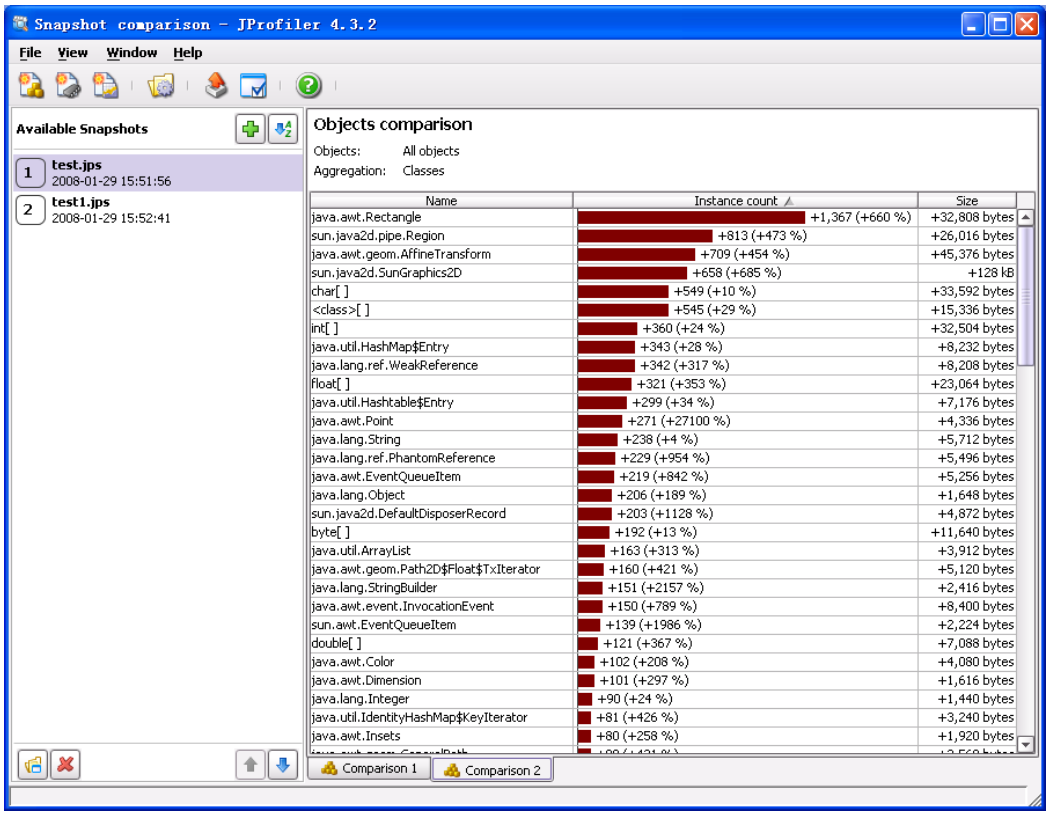
✓ 全部对象

✓ 记录对象

第四步：选择集成等级－－类、包、组件

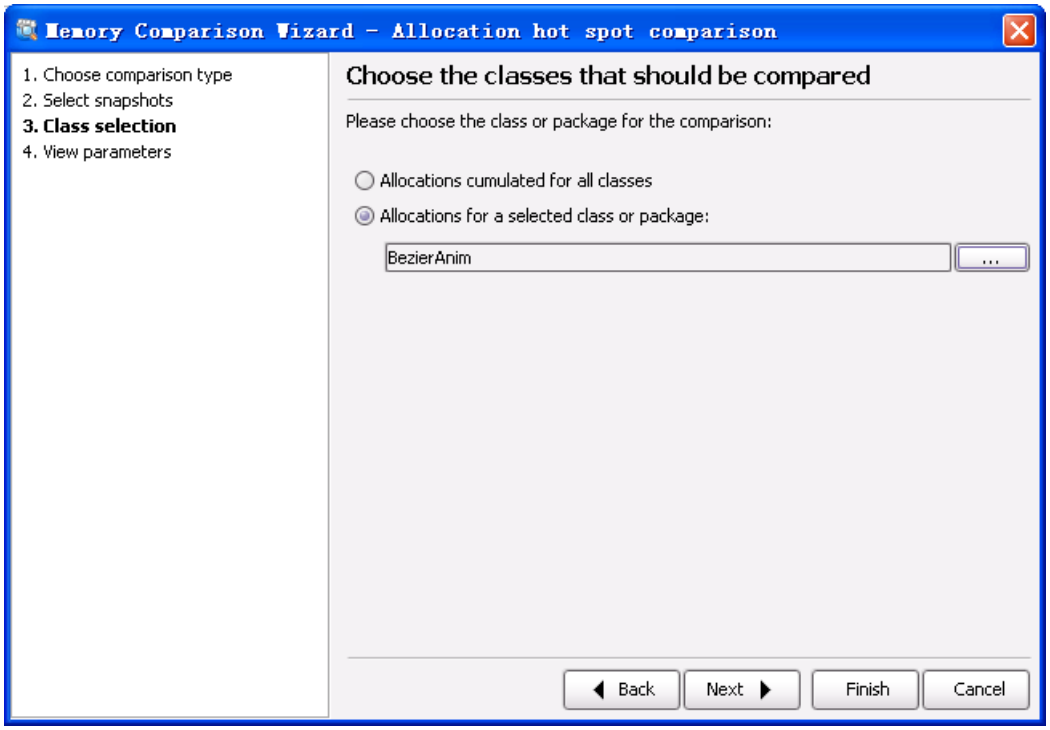

Memory Comparison Wizard - Objects comparison
1. Choose comparison type
2. Select snapshots
3. Recording type
4. View parameters
Select view parameters
Please specify the following parameters that are necessary in order to calculate the snapshot comparison:
Aggregation level:
Classes
Liveness mode:
Classes
Packages
J2EE Components
Back
Next
Finish
Cancel

第五步：查看比较结果



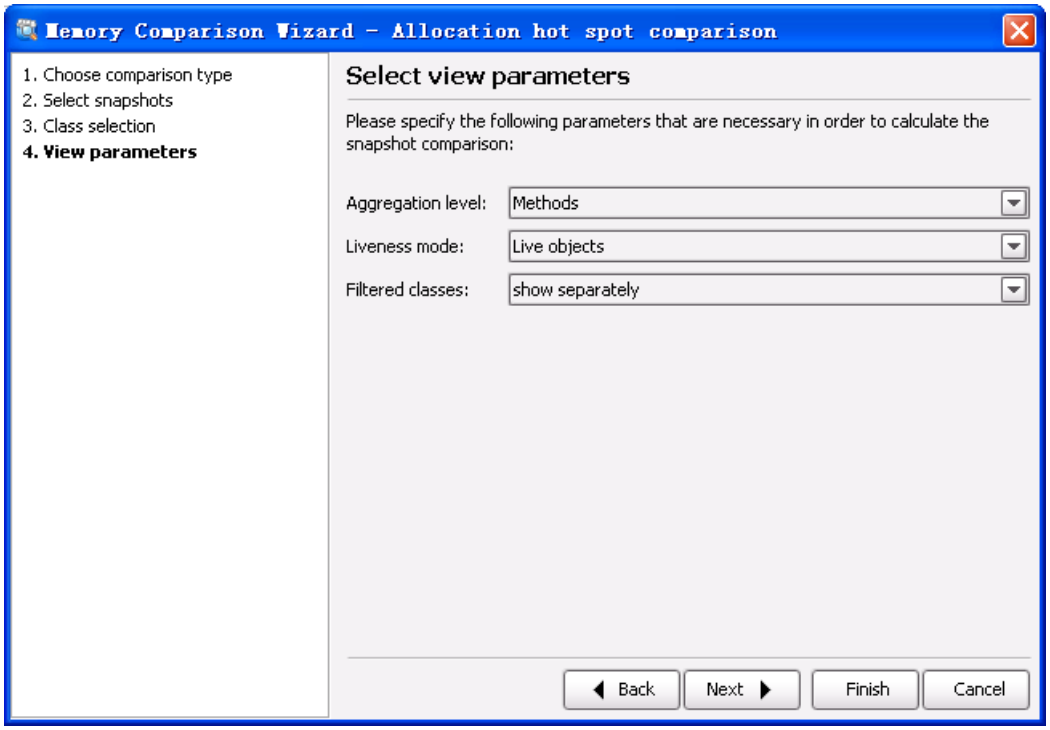
5.1.2 分配热点比较

第三步：选择比较的类：所有类、选择的类

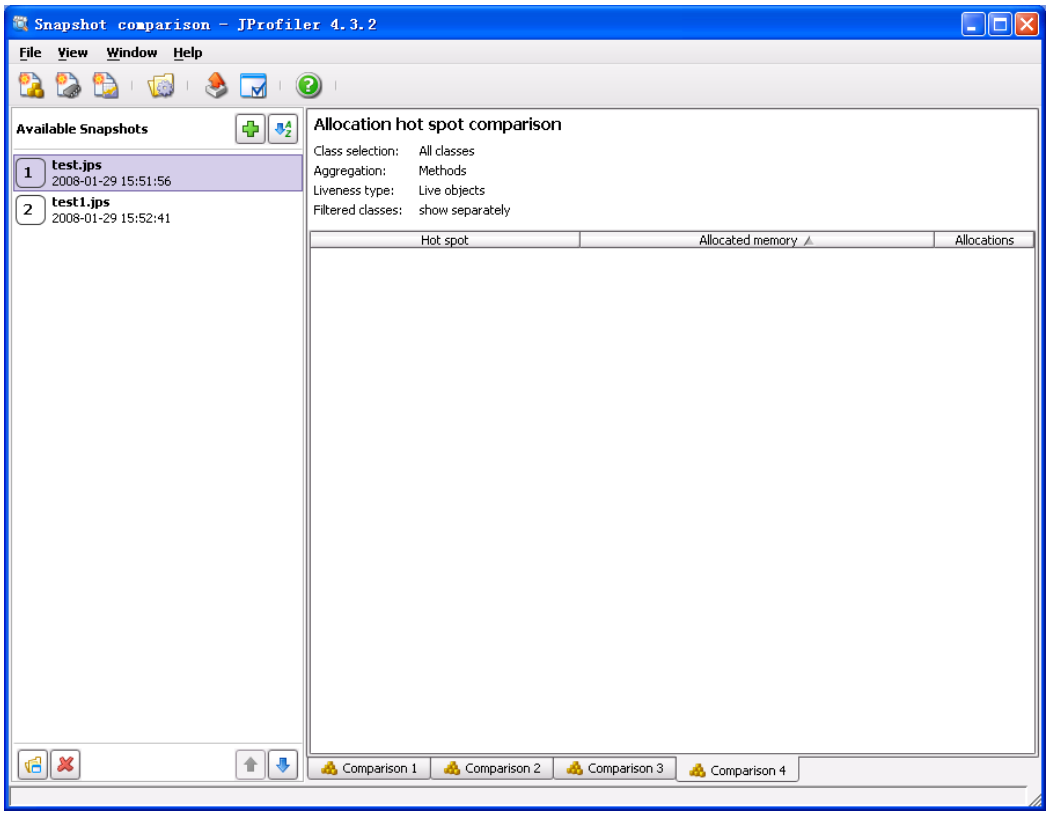


第四步：选择视图参数

- ✓ 集成等级：类、包、组件
- ✓ 活动模式：存活对象、垃圾回收对象、存活的和垃圾回收的对象
- ✓ 类过滤：独立显示、在调用类上增加分配

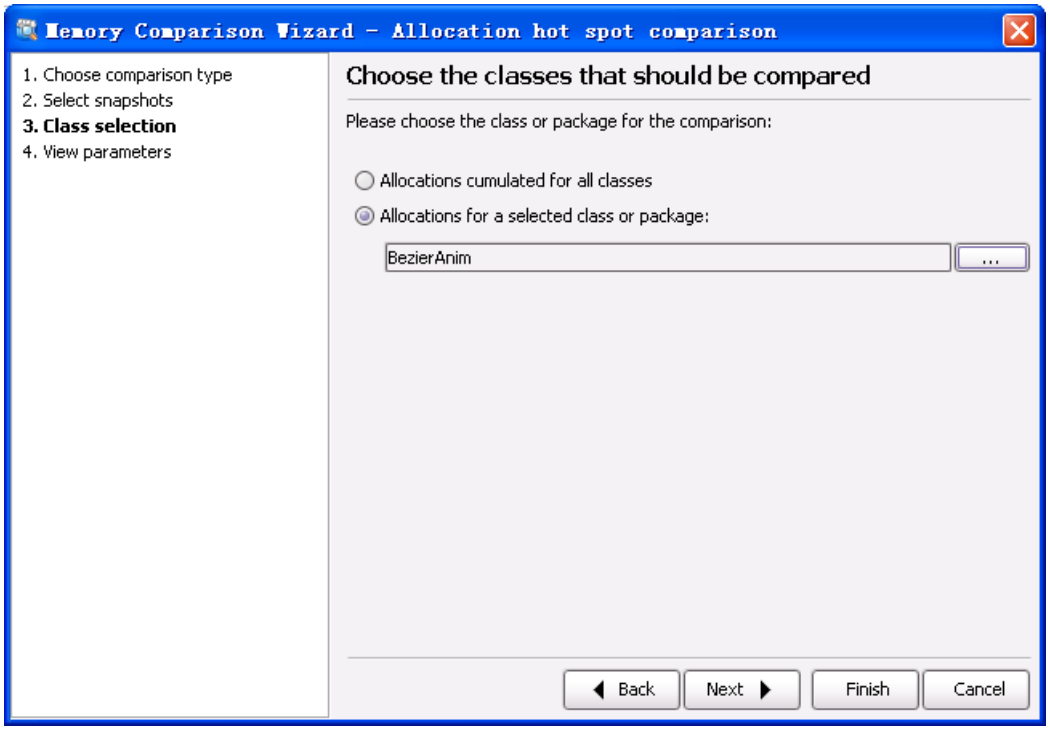


第五步：查看比较结果（例中无）



5.1.3 分配树比较

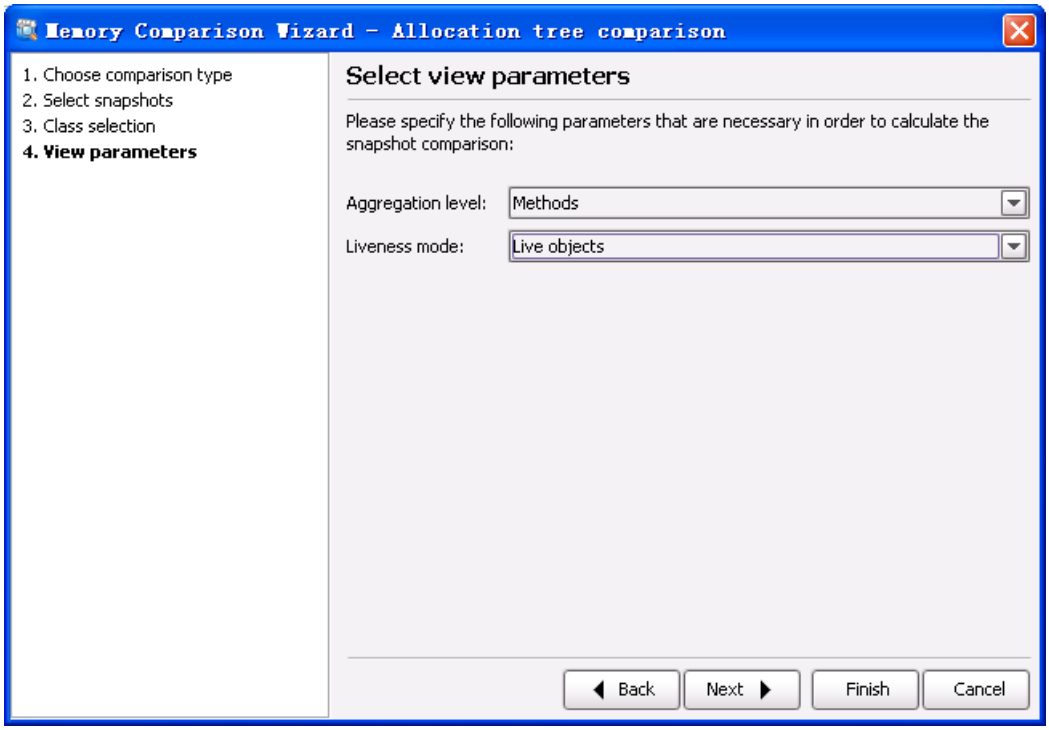
第三步：选择比较的类：所有类、选择的类



第四步：选择视图参数

- ✓ 集成等级：方法、类、包、组件
- ✓ 活动模式：存活对象、垃圾回收对象、存活的和垃圾回收的对象

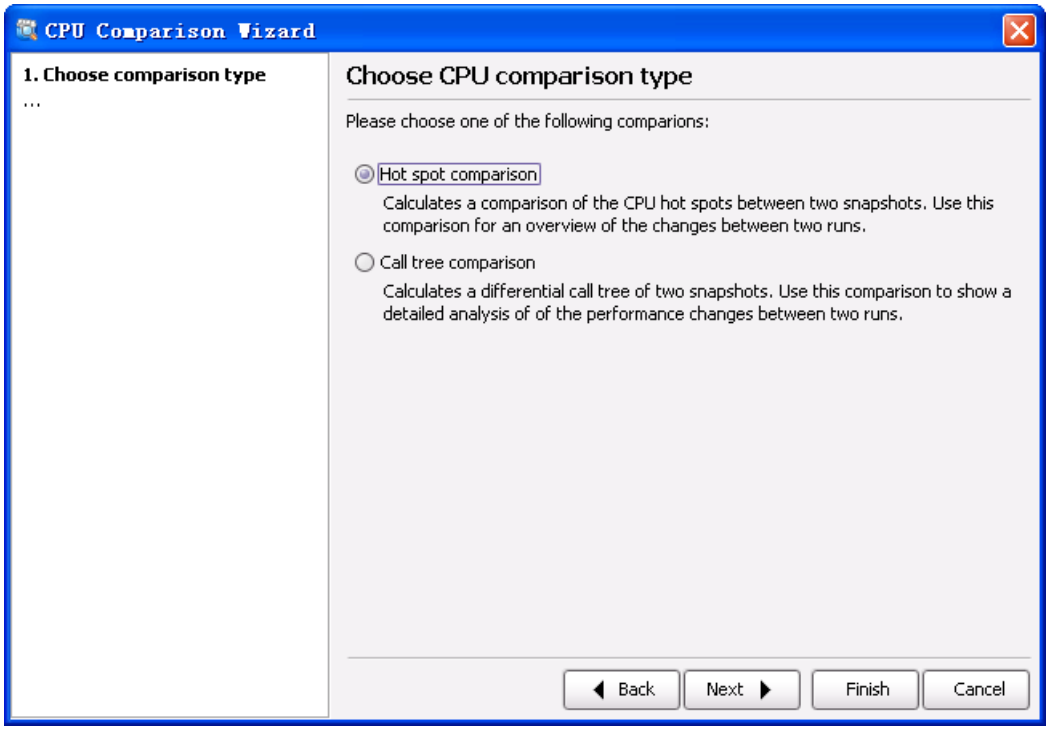
批注 [U2]: 此视图的最小集成等级为方法



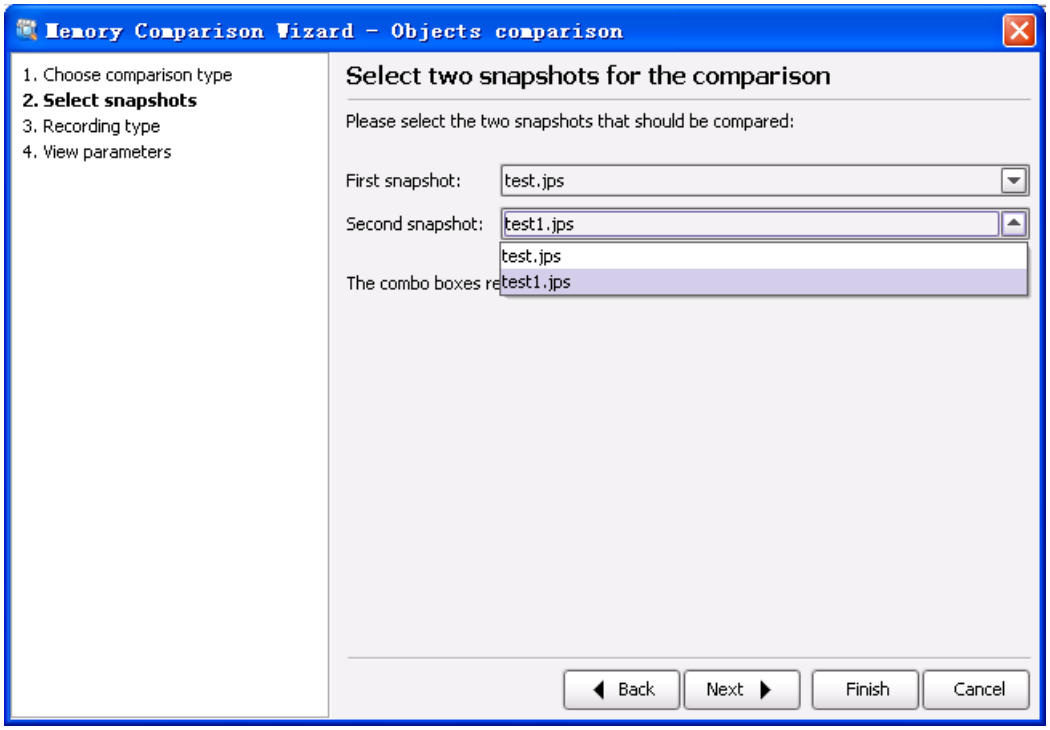
5.2 创建 CPU 比较

选择 File→Create CPU Comparison，或使用快捷键 F6

第一步，选择比较类型


CPU Comparison Wizard
1. Choose comparison type
...
Choose CPU comparison type
Please choose one of the following comparions:
Hot spot comparison
Calculates a comparison of the CPU hot spots between two snapshots. Use this comparison for an overview of the changes between two runs.
Call tree comparison
Calculates a differential call tree of two snapshots. Use this comparison to show a detailed analysis of of the performance changes between two runs.
Back
Next
Finish
Cancel

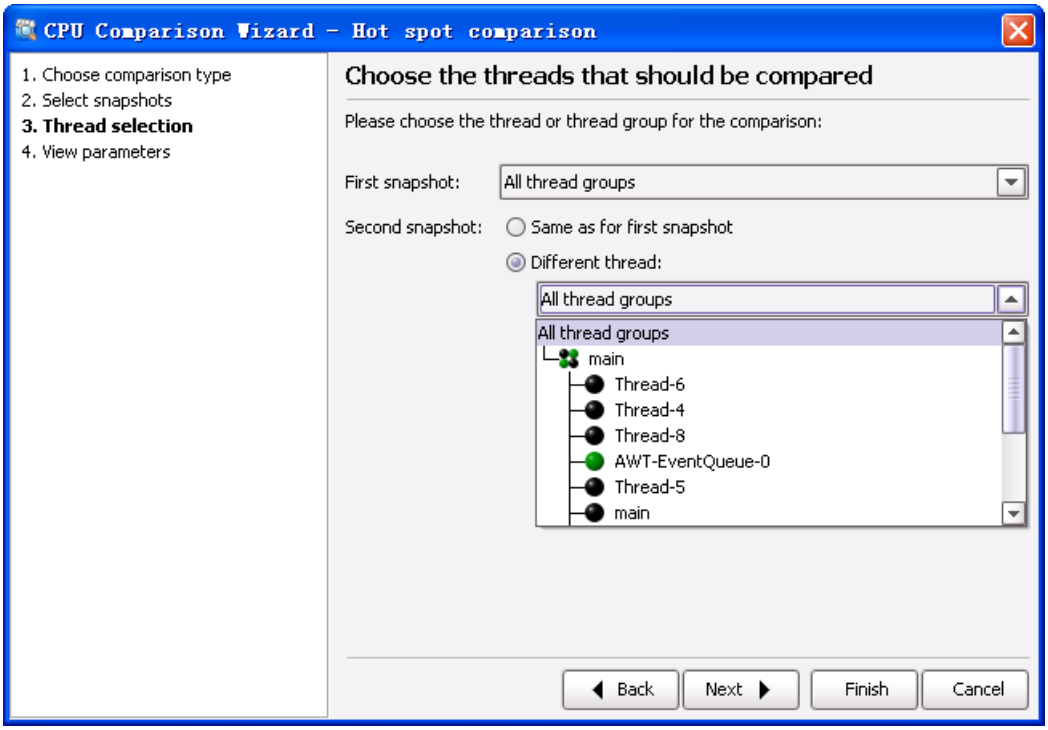
第二步：选择快照



第三步：选择线程

选择第一个快照需要比较的线程

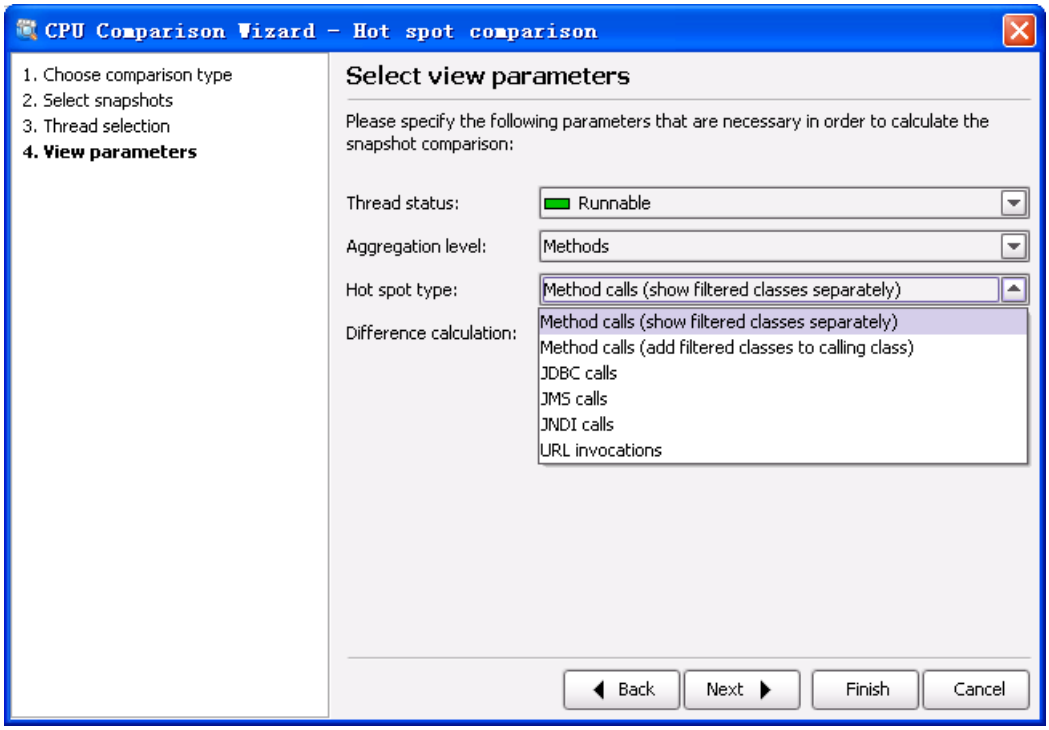
针对第二个快照，有两种选择：与第一个快照相同的线程；不同的线程，可任何选择



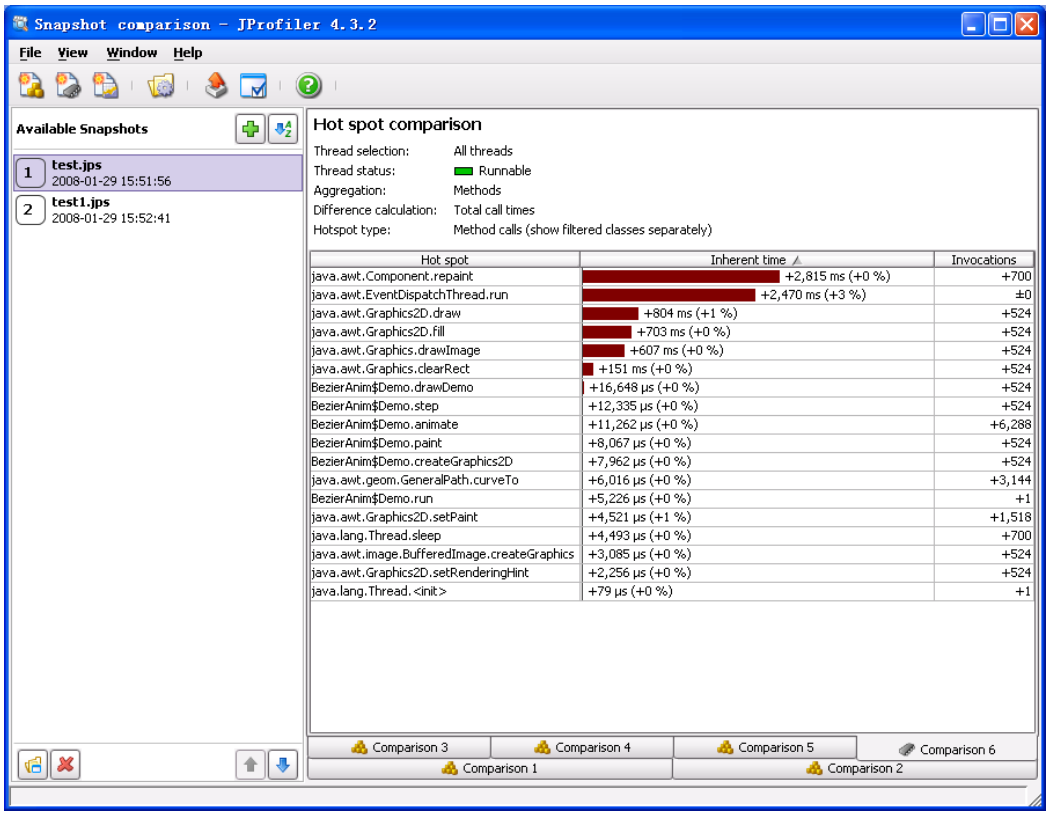
5.2.1 热点比较

第四步：选择视图参数

- ✓ 线程状态：所有状态、可运行的、等待状态、阻塞状态、NET I/O
- ✓ 集成等级：方法、类、包、组件
- ✓ 热点类型：方法调用、JDBC 调用、JMS 调用、JNDI 调用
- ✓ 不同的计算方法：总调用时间、平均调用时间


CPU Comparison Wizard - Hot spot comparison
1. Choose comparison type
2. Select snapshots
3. Thread selection
4. View parameters
Select view parameters
Please specify the following parameters that are necessary in order to calculate the snapshot comparison:
Thread status:
Runnable
Aggregation level:
Methods
Hot spot type:
Method calls (show filtered classes separately)
Difference calculation:
Method calls (show filtered classes separately)
Method calls (add filtered classes to calling class)
JDBC calls
JMS calls
JNDI calls
URL invocations
Back
Next
Finish
Cancel

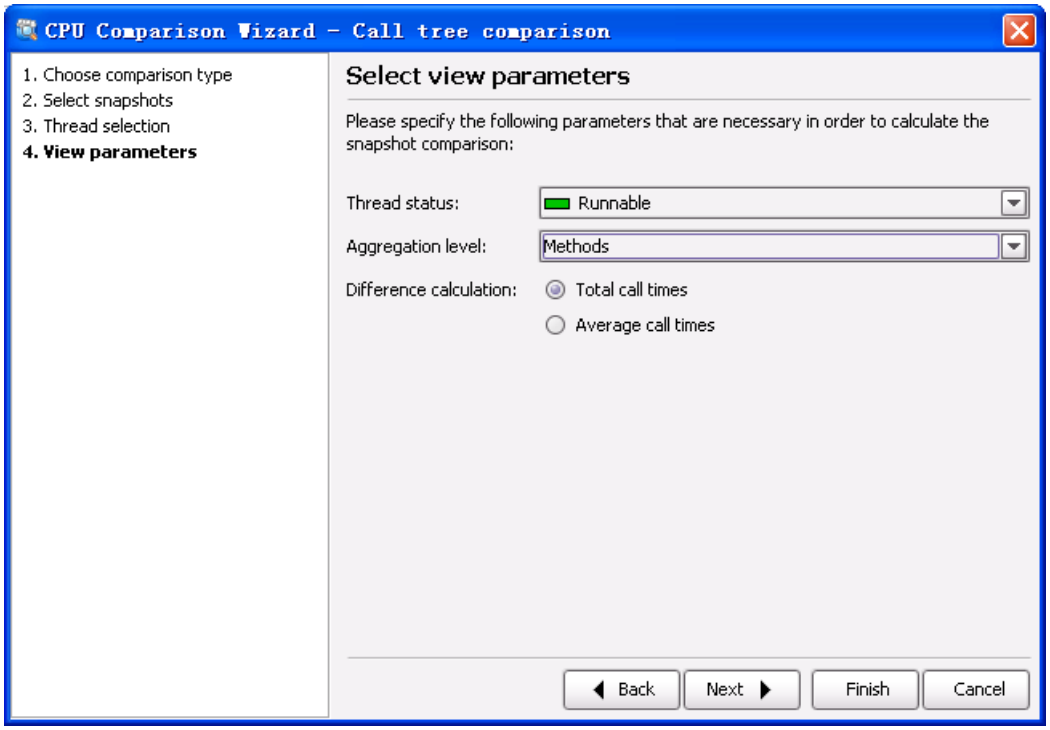
第五步：查看视图结果



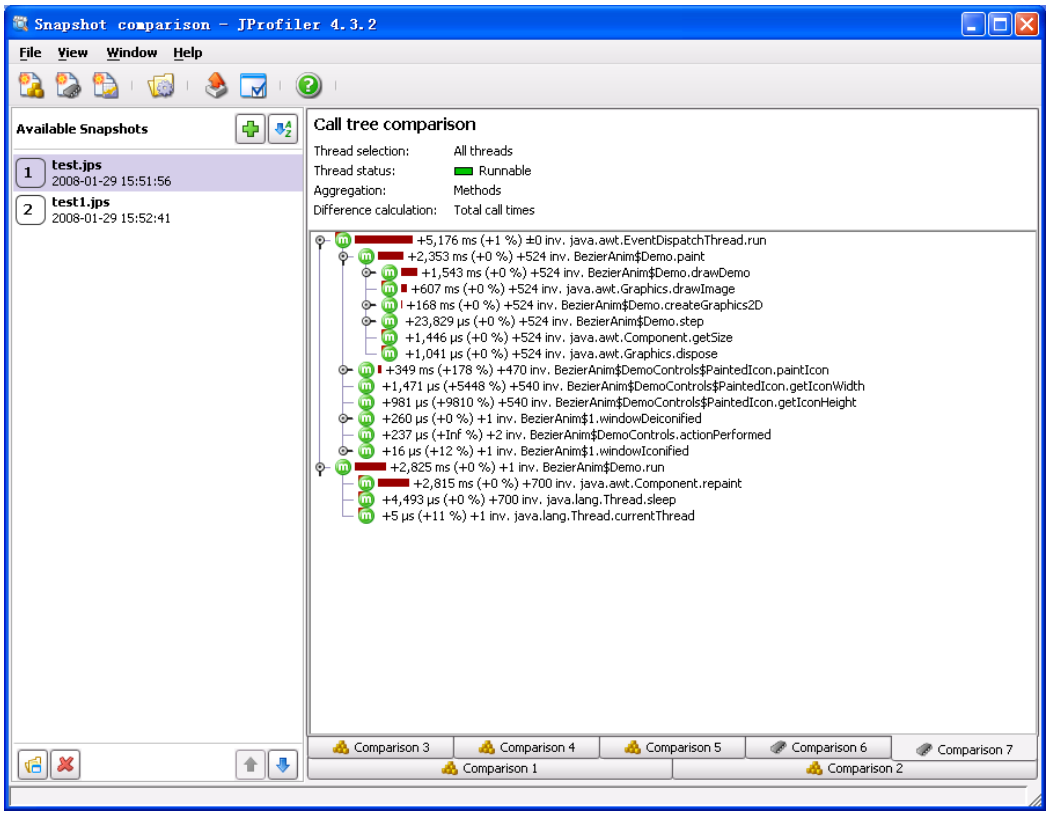
5.2.2 调用树比较

第四步：选择视图参数

- ✓ 线程状态：所有状态、可运行的、等待状态、阻塞状态、NET I/O
- ✓ 集成等级：方法、类、包、组件
- ✓ 不同的计算方法：总调用时间、平均调用时间


CPU Comparison Wizard - Call tree comparison
1. Choose comparison type
2. Select snapshots
3. Thread selection
4. View parameters
Select view parameters
Please specify the following parameters that are necessary in order to calculate the snapshot comparison:
Thread status:
Runnable
Aggregation level:
Methods
Difference calculation:
Total call times
Average call times
Back
Next
Finish
Cancel

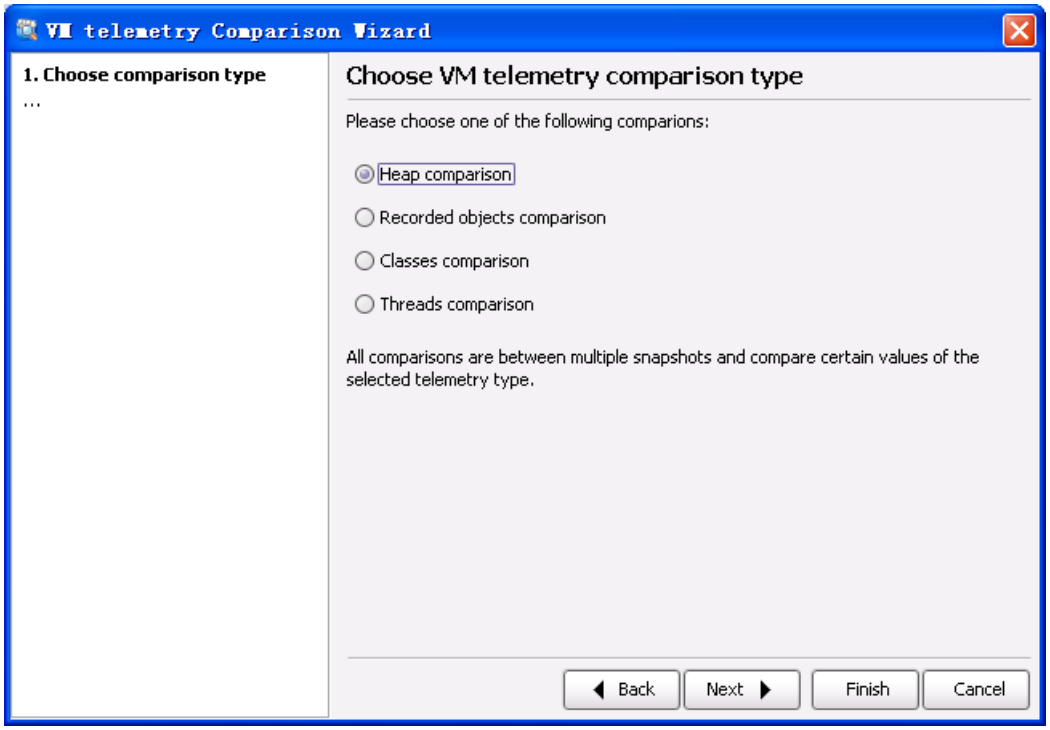
第五步：查看比较结果



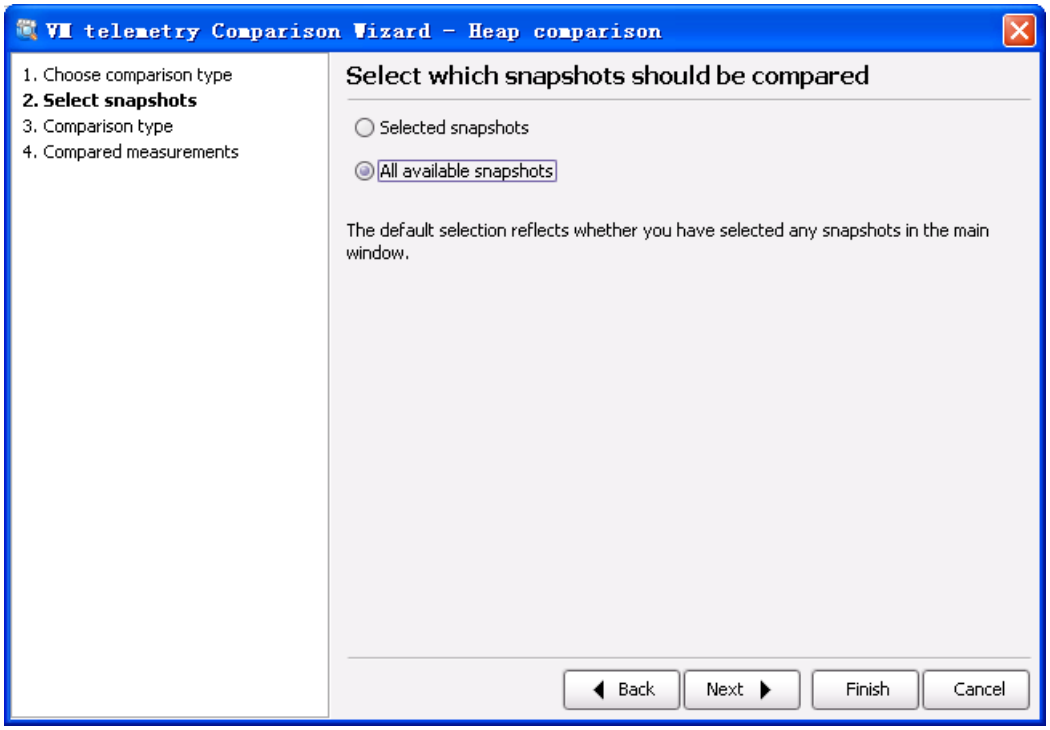
5.3 创建遥感比较

选择 File→Create Telemetry Comparison，或使用快捷键 F7

第一步：选择比较类型：



第二步：选择要比较的快照

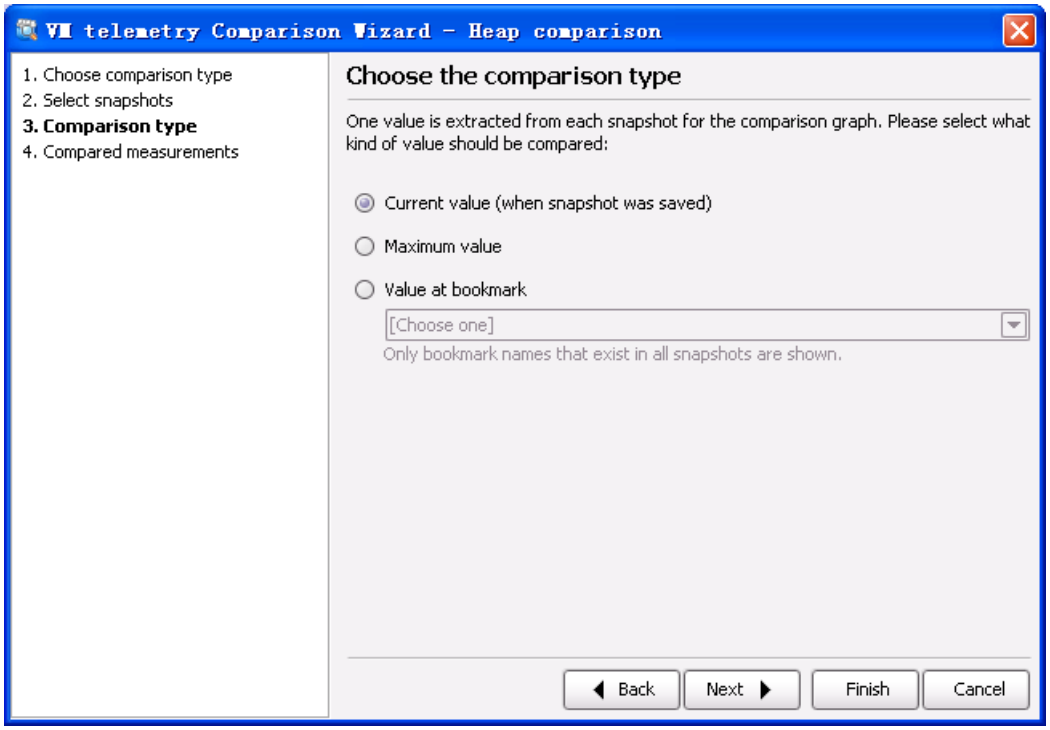


只比较选中的快照，在打开比较向导之前，要多选需要比较的快照。

对比所有可用快照。

第三步：选择比较类型

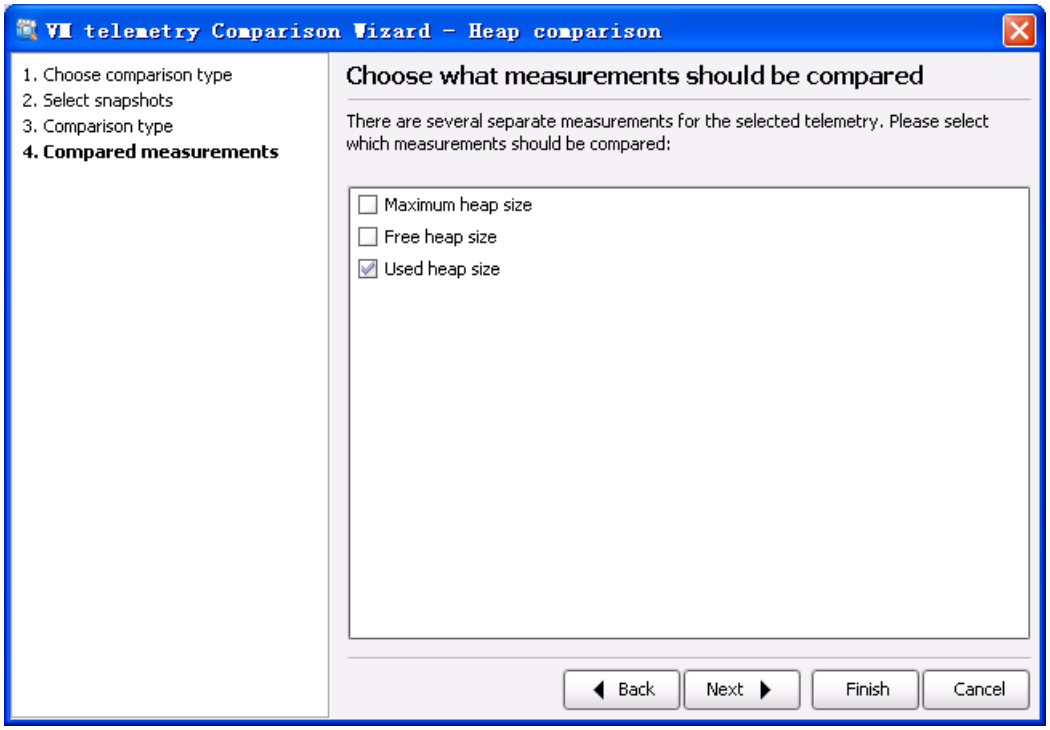
- ✓ 当前值（快照存储时的值）
- ✓ 最大值
- ✓ 某个书签标记值



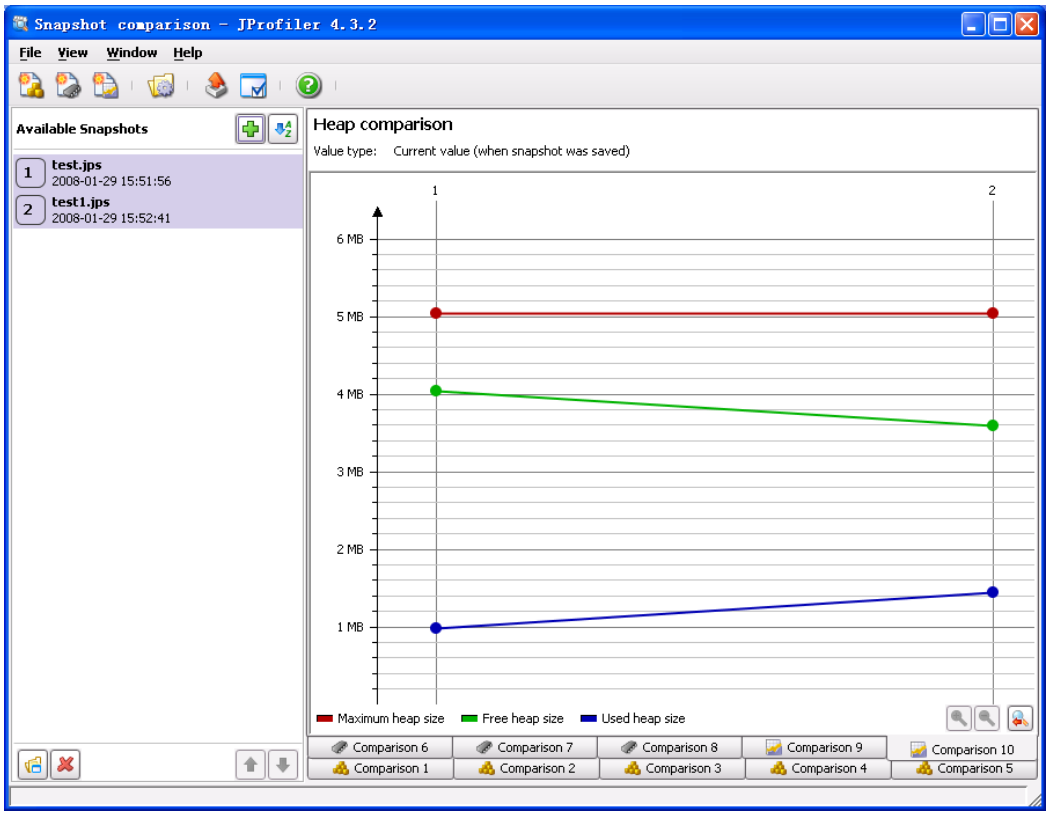
5.3.1 堆比较

第四步：选择要比较的度量，可以多选

- ✓ 最大的堆大小
- ✓ 未使用的堆大小
- ✓ 使用的堆大小


VM telemetry Comparison Wizard - Heap comparison
1. Choose comparison type
2. Select snapshots
3. Comparison type
4. Compared measurements
Choose what measurements should be compared
There are several separate measurements for the selected telemetry. Please select which measurements should be compared:
Maximum heap size
Free heap size
Used heap size
Back
Next
Finish
Cancel

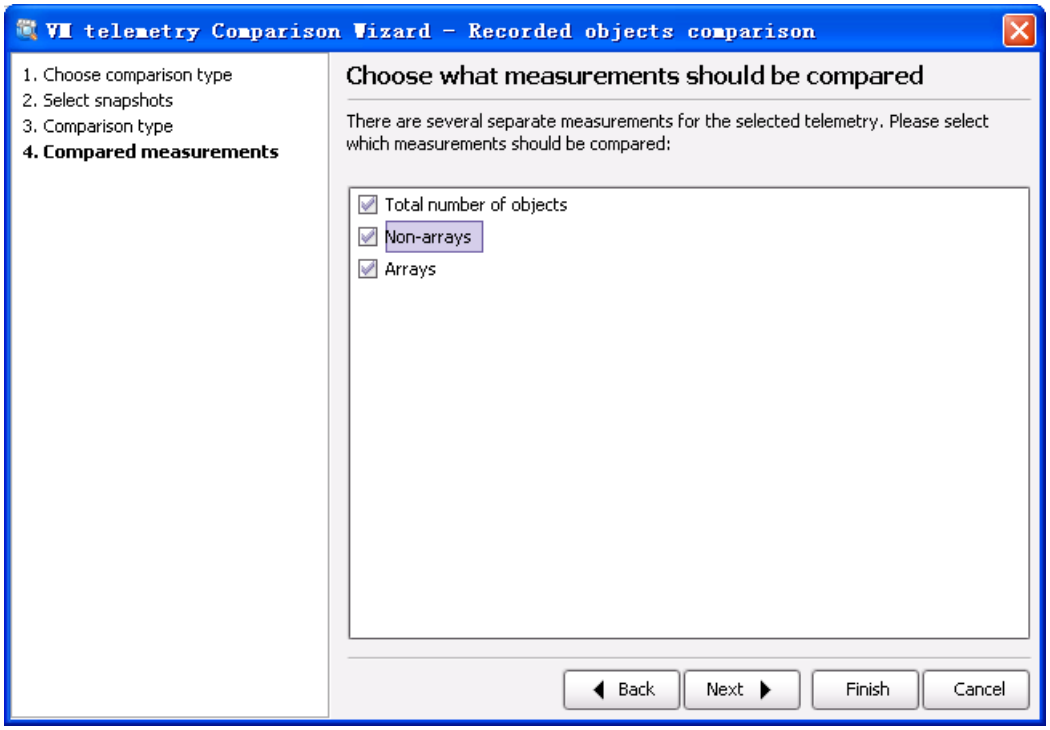
第五步：查看比较结果



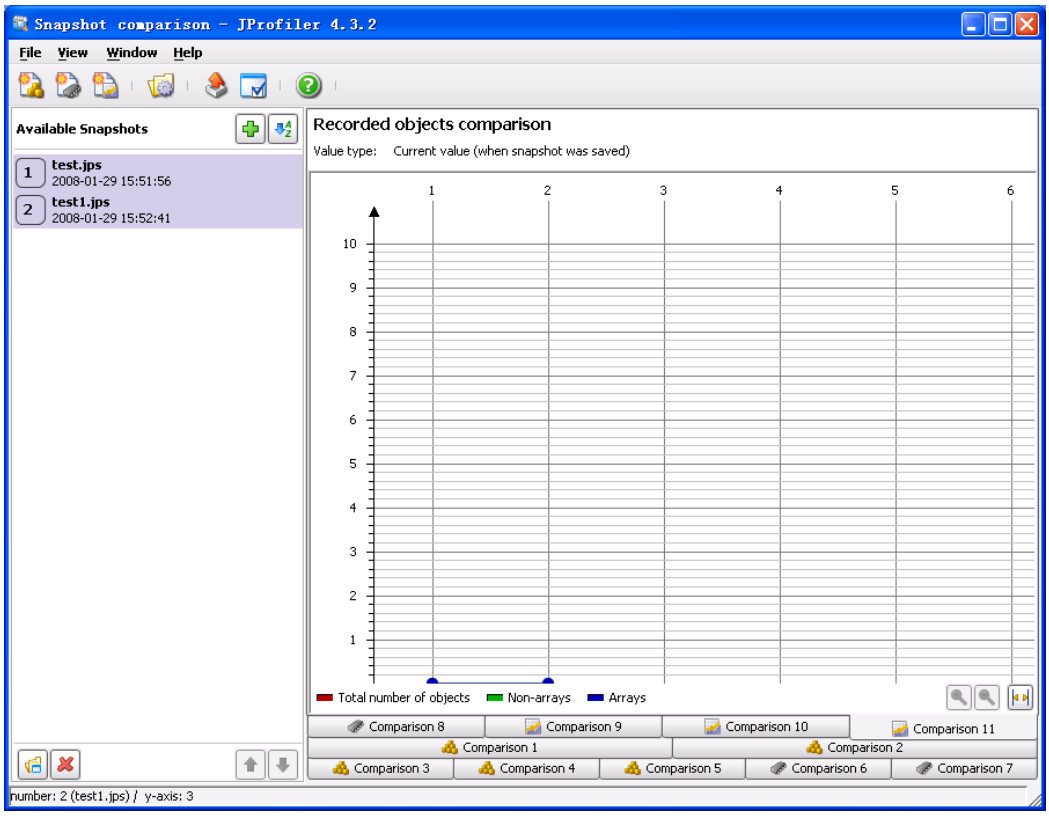
5.3.2 记录对象比较

第四步：选择要比较的度量，可以多选

- ✓ 全部对象
- ✓ 非数组
- ✓ 数组


VM telemetry Comparison Wizard - Recorded objects comparison
1. Choose comparison type
2. Select snapshots
3. Comparison type
4. Compared measurements
Choose what measurements should be compared
There are several separate measurements for the selected telemetry. Please select which measurements should be compared:
Total number of objects
Non-arrays
Arrays
Back
Next
Finish
Cancel

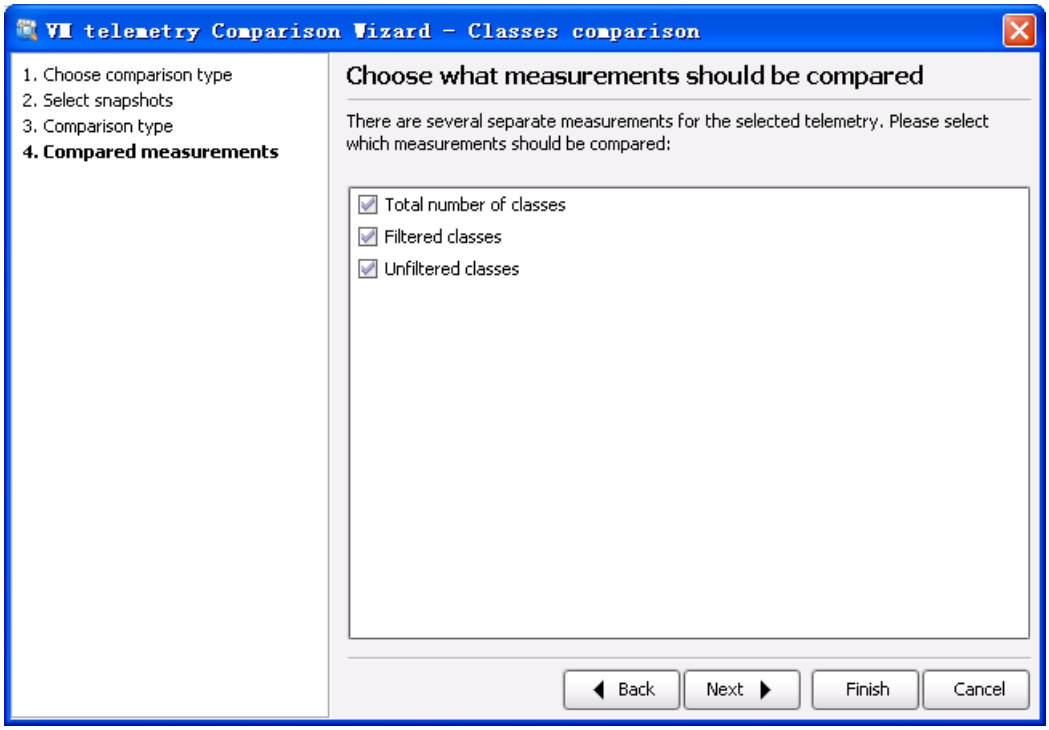
第五步：查看比较结果



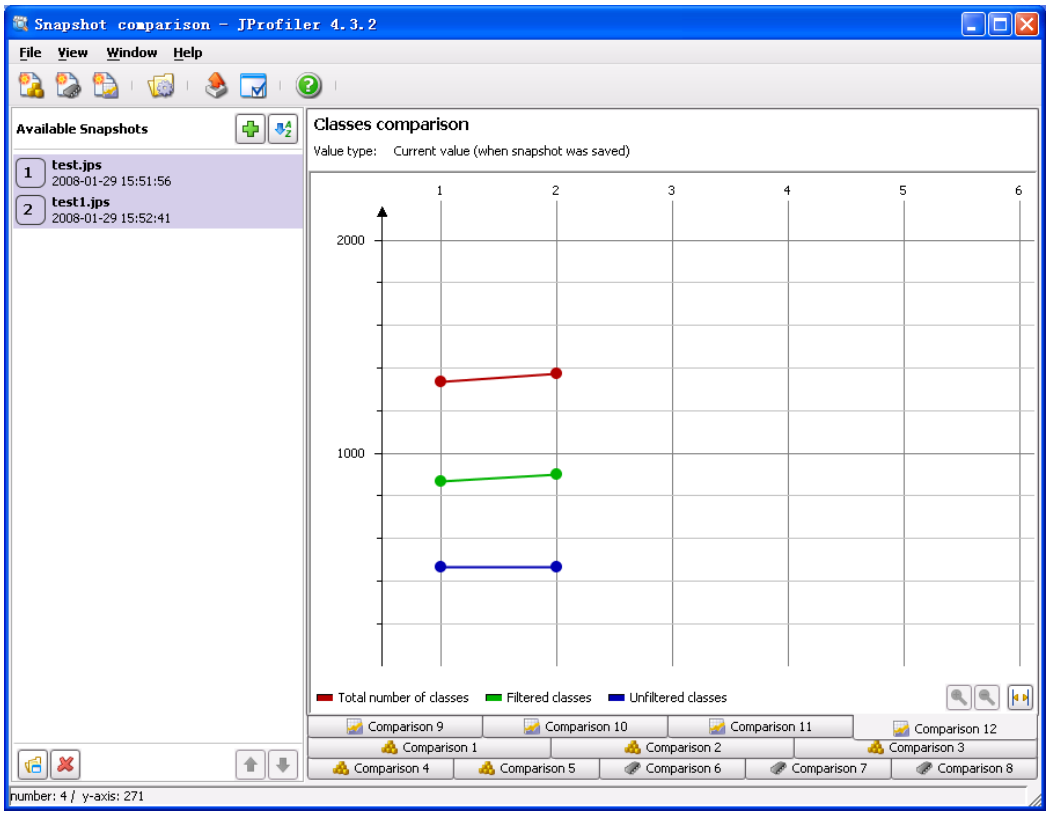
5.3.3 类比较

第四步：选择要比较的度量，可以多选

- ✓ 全部类
- ✓ 过滤类
- ✓ 未过滤类


VM telemetry Comparison Wizard - Classes comparison
1. Choose comparison type
2. Select snapshots
3. Comparison type
4. Compared measurements
Choose what measurements should be compared
There are several separate measurements for the selected telemetry. Please select which measurements should be compared:
Total number of classes
Filtered classes
Unfiltered classes
Back
Next
Finish
Cancel

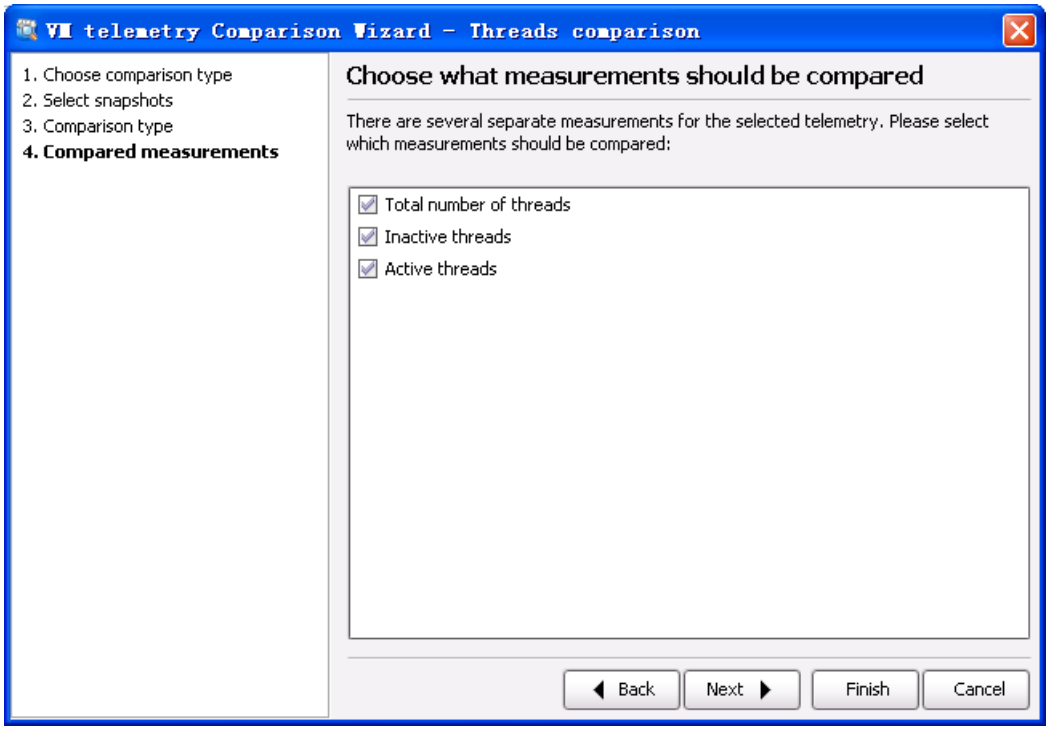
第五步：查看比较结果



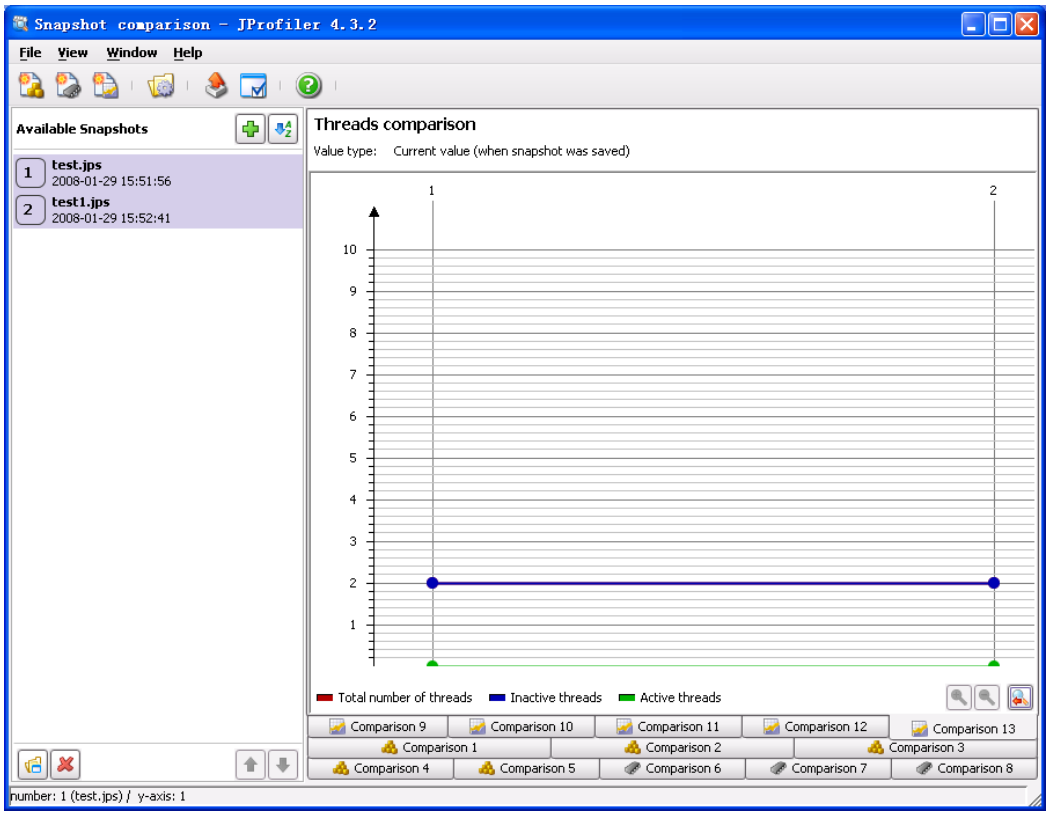
5.3.4 线程比较

第四步：选择要比较的度量，可以多选

- ✓ 全部线程
- ✓ 活动线程
- ✓ 不活动线程


VM telemetry Comparison Wizard - Threads comparison
1. Choose comparison type
2. Select snapshots
3. Comparison type
4. Compared measurements
Choose what measurements should be compared
There are several separate measurements for the selected telemetry. Please select which measurements should be compared:
Total number of threads
Inactive threads
Active threads
Back
Next
Finish
Cancel

第五步：查看比较结果


Snapshot comparison - JProfiler 4.3.2
File View Window Help
Available Snapshots
1 test.jps 2008-01-29 15:51:56
2 test1.jps 2008-01-29 15:52:41
Threads comparison
Value type: Current value (when snapshot was saved)
1
2
10
9
8
7
6
5
4
3
2
1
Total number of threads
Inactive threads
Active threads
Comparison 9
Comparison 10
Comparison 11
Comparison 12
Comparison 13
Comparison 1
Comparison 2
Comparison 3
Comparison 4
Comparison 5
Comparison 6
Comparison 7
Comparison 8
number: 1 (test.jps) / y-axis: 1

6 IDE 集成（Eclipse 3.x）

当 Jprofiler 与 eclipse 3.x IDE 集成后, JProfiler 不需要配置 session ，可以直接在 IDE 中调用。

需求：需要 eclipse 3.0 或 eclipse 3.1 全部 SDK 的插件。JProfiler 集成不能在 eclipse 框架部分安装的情况下使用。

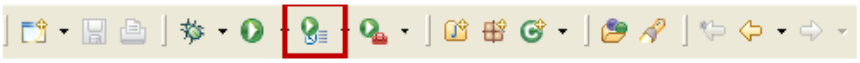
具体安装请参见 JProfiler's setup wizard

提醒：请在插件安装之前关闭 eclipse，要按照 JProfiler's setup wizard 来安装 JProfiler's，请在完成全部安装之前不要启动 eclipse。

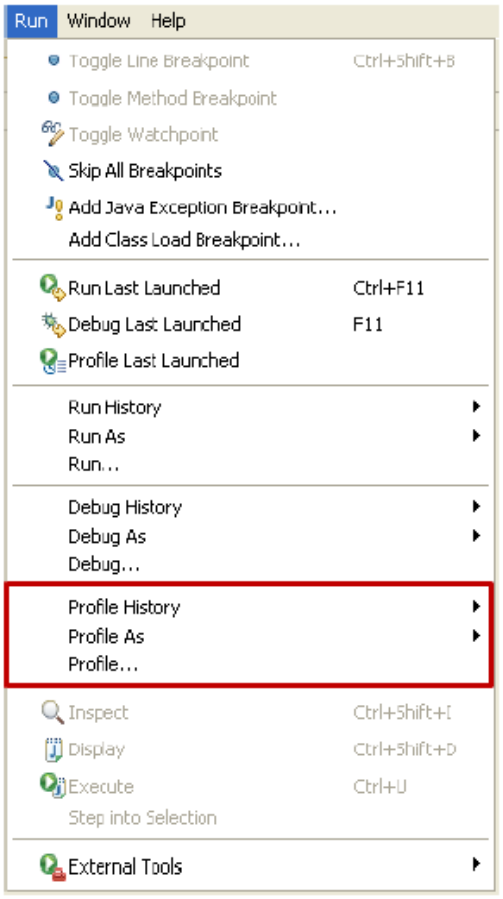
文件选择器会提醒你安装在 eclipse 的安装目录下。

当收到安装完成提示后，你可以打开 eclipse 并检查安装是否成功。如果在 Java perspective 中， 菜单 Run->Profile ...不存在。请在 Window->Customize perspective 中，在 Command 标签中的"Profile”选项前打勾

从 eclipse 中剖析应用，在 Run 菜单下选择 profiling 命令，或者点相关的按钮。



Main eclipse toolbar with "Profile" button



eclipse "Run" menu with "Profile" actions

FAQ：如果远程连接不成功，请检查

1、 本地和监控机是否都安装了 JProfiler，并用版本相同

2、 远程 JProfiler 的安装路径是否正确

3、 本地和监控机的 JProfiler 的端口是否一致，最好都使用默认的 8849

4、 所选 JVM 的提供商和版本是否正确，可能机子上有多个 JAVA 版本，一定要选择应用程序启动时所用的

5、 应用目录必须可写共享，监控机需要映射共享目录，配置完后在本地服务器上会生成 JProfiler 监控的启动文件。