**实验3对象运行时行为的调试与性能分析**

### 1.实验目的

（1）通过小段代码的调试的深入分析，了解面向对象程序运行时涉及的机制，从而对面向对象程序的行为有更深刻的理解。

（2）用TPTP、Jprofile等插件进行JAVA程序性能分析。

### 2. 知识要点

JVM中有几个比较重要的内存区域，在java类的生命周期中扮演着比较重要的角色：

（1）方法区：在java的虚拟机中有一块专门用来存放已经加载的类信息、常量、静态变量以及方法代码的内存区域，即方法区。

（2）常量池：常量池是方法区的一部分，主要用来存放常量和类中的符号引用等信息。

（3）堆区（heap）：用于存放类的对象实例。

（4）栈区（stack）：也叫java虚拟机栈，是由一个一个的栈帧组成的后进先出的栈式结构，栈桢中存放方法运行时产生的局部变量、方法出口等信息。当调用一个方法时，虚拟机栈中就会创建一个栈帧存放这些数据，当方法调用完成时，栈帧消失，如果方法中调用了其他方法，则继续在栈顶创建新的栈桢。

###### 2.2.1 理解JAVA中class的加载

(1) JVM在程序运行过程中根据需要动态加载相应的类，能够按照程序的要求创建相应的对象。

(2) 把相应类的.class文件(可执行bytecode)加载进内存

###### 2.2.2对象初始化

(1) JVM在首次加载Java类时，会对静态初始化块、静态成员变量、静态方法进行一次初始化

(2) 只有在调用new方法时才会创建类的实例

(3) 类实例创建过程：按照父子继承关系进行初始化。

###### 2.2.3 方法调用

（1）静态调用：类方法从JVM方法区调用，比如static, final, private等。

（2）动态调用：每调用一个方法，当前线程创建一个栈帧，然后放入当前stack中，每个栈帧都包括局部变量区，操作数栈和栈帧变量区。

###### 2.2.4 垃圾回收

（1）分配内存

（2）保证所有正在被引用的对象还存在于内存中

（3）回收执行代码已经不再引用的对象所占的内存

### 3. 工具讲解

3.1 学习使用JAVA Profiling工具观察和优化JAVA程序的性能

为了能准确地获得程序的性能信息，需要使用各种辅助工具。这些系统性能分析工具对性能瓶颈定位、系统故障排查都很有帮助。这类辅助工具也非常多，最为直接的有一些命令行工具，就在Bin目录下，例如jstack\jinfo等，还有一些比较方便的有图形界面的，例如JDK自带的获取Java程序堆信息、线程信息的相关工具jconsole、Visual VM等等，也有一些需要单独下载安装的插件如MAT、Jprofile、TPTP等。

###### 3.1.1 TPTP的安装和使用

TPTP是eclipse官方的profiling插件。常用的profiling简单来讲就对程序运行进行记录，然后从数据中分析哪些方法运行时间长，哪些对象占用内存比较多，哪些类的实例多等等。在本次实验里你将学习到如何使用eclipse官方插件对简单的Java应用程序进行概要分析，例如：内存分析、执行分析，并结合UML2序列图进行分析。

在<http://www.eclipse.org/tptp/>下载，目前机房所有机器安装的是All-in-one版本。下载后解压到任意目录下，里面带有Eclipse平台最新版本，TPTP all-in-one package已集成到Eclipse平台里面，无需作任何配置。免去了其它一些繁琐的安装步骤。

但大家注意，目前这种版本只适用于WIN32，如果你自己的系统是WIN64，请选择下载针对具体系统的All-Runtime版本，然后像其它插件一样解压到eclipse的目录，用eclipse -clean来刷新一下。检验你的TPTP是否安装成功的方法是看“window->open perspective”里面是否有“Profiling and logging”选项。在这个过程中，请注意以下三点：

* 原有的Eclipse版本要能够支持TPTP（版本需要兼容）
* 需要安装EMF和XSD
* 另外，以下所有讲解针对本地JAVA程序的性能观察，若今后的程序开发中用到服务器端JAVA程序的性能分析，请加装与TPTP平台版本匹配的Agent Controller。

###### 3.1.2 Profiling实验案例

[案例分析：JAVA Profiling ]

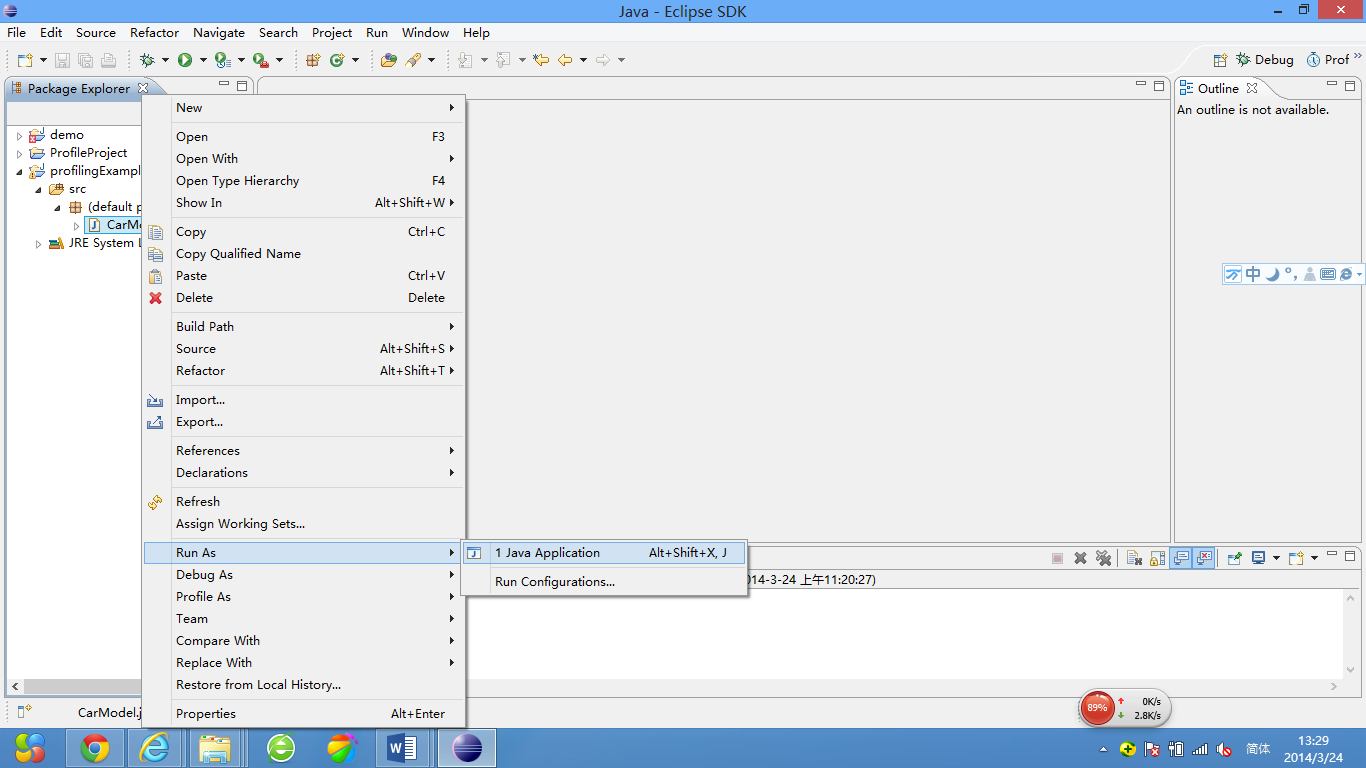
实验步骤：

一、新建JAVA工程

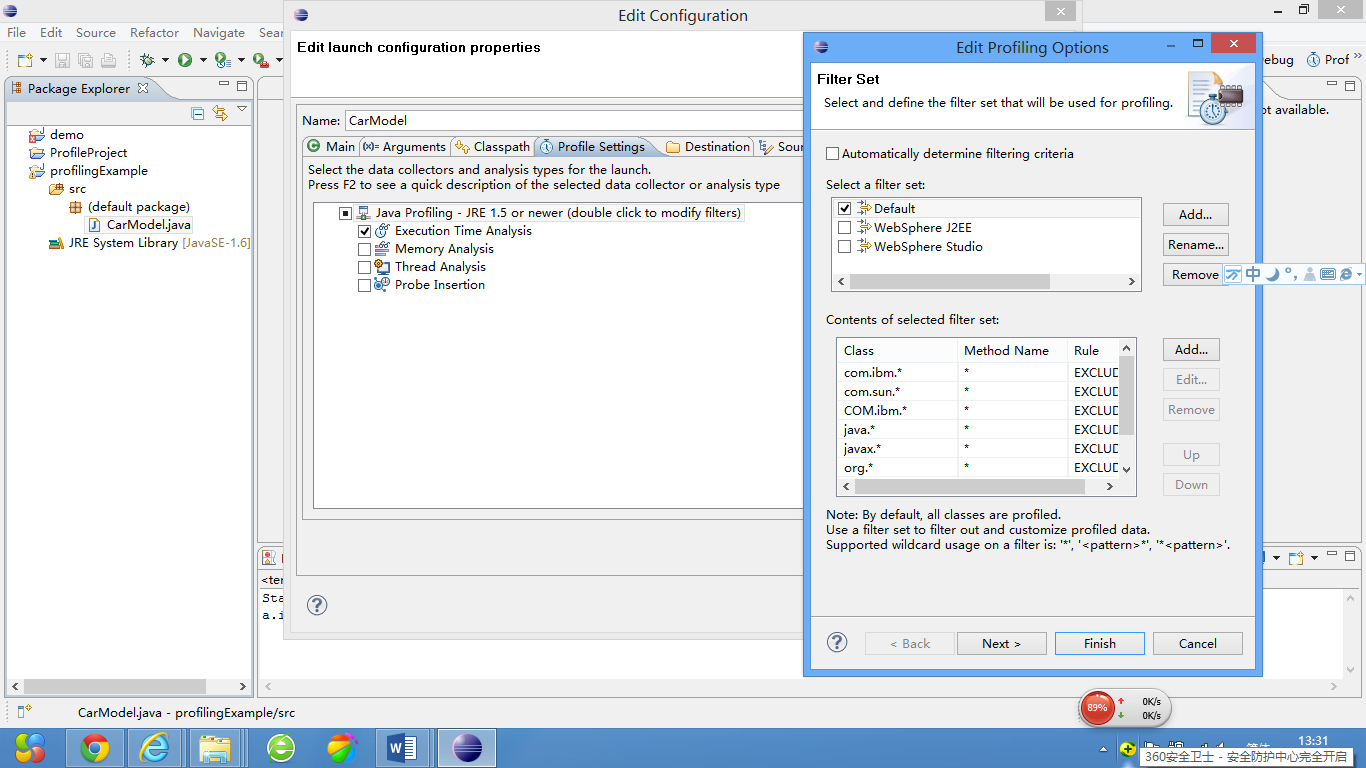
1. 新建一个JAVA工程，取名为ProfilingExample
2. 新建一个类，取名为CarModel.java
3. 保存并编译通过。

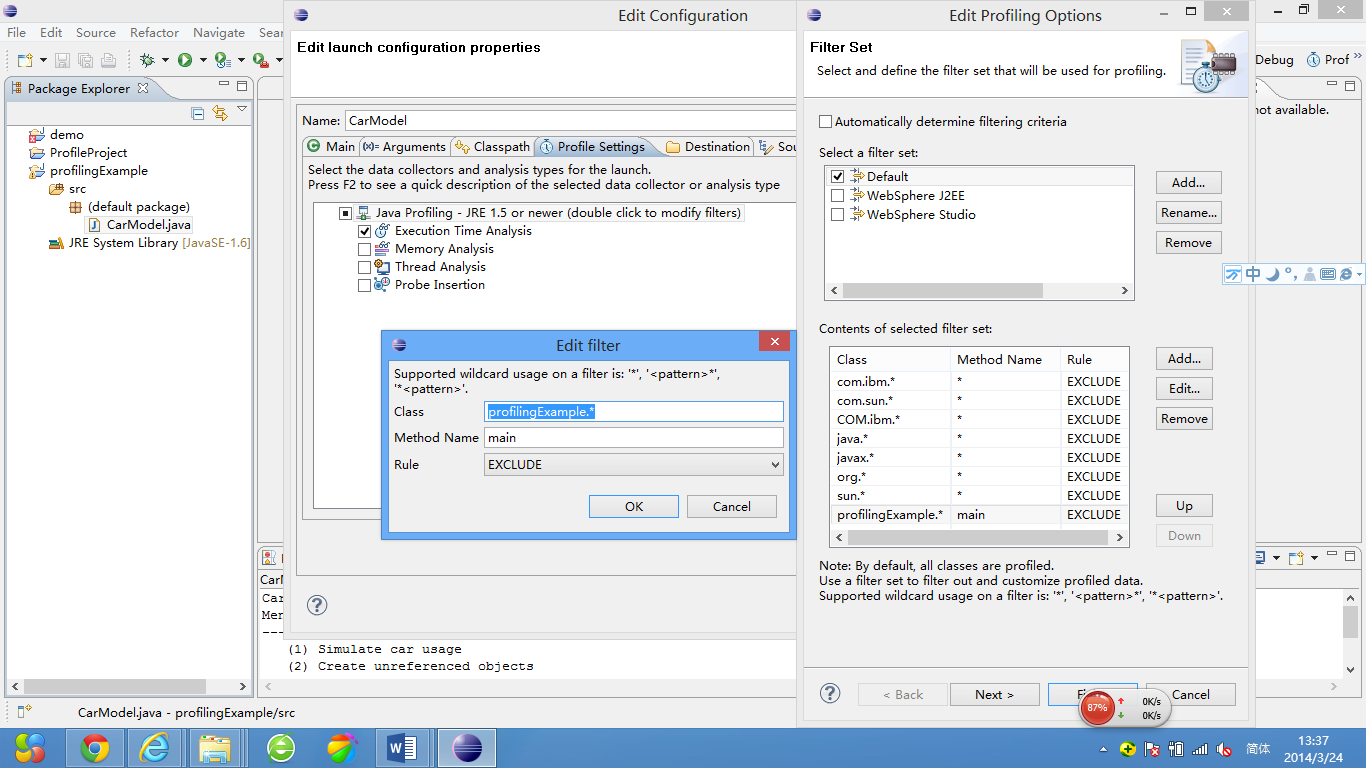
二、Profiling该例子

1. 打开Profiling视图：选定该类，右键选择“Profile as>Java Application”



1. 选择过滤器：在弹出的对话框中双击“double click to modify filters”，选择默认的过滤器（“Default”），在“Contents of selected filter set”中将类名“CarModel”(或使用通配符“\*”将ProfilingExample项目中所有的类)添加进去，方法名写“main”，过滤条件选“EXCLUDE”。

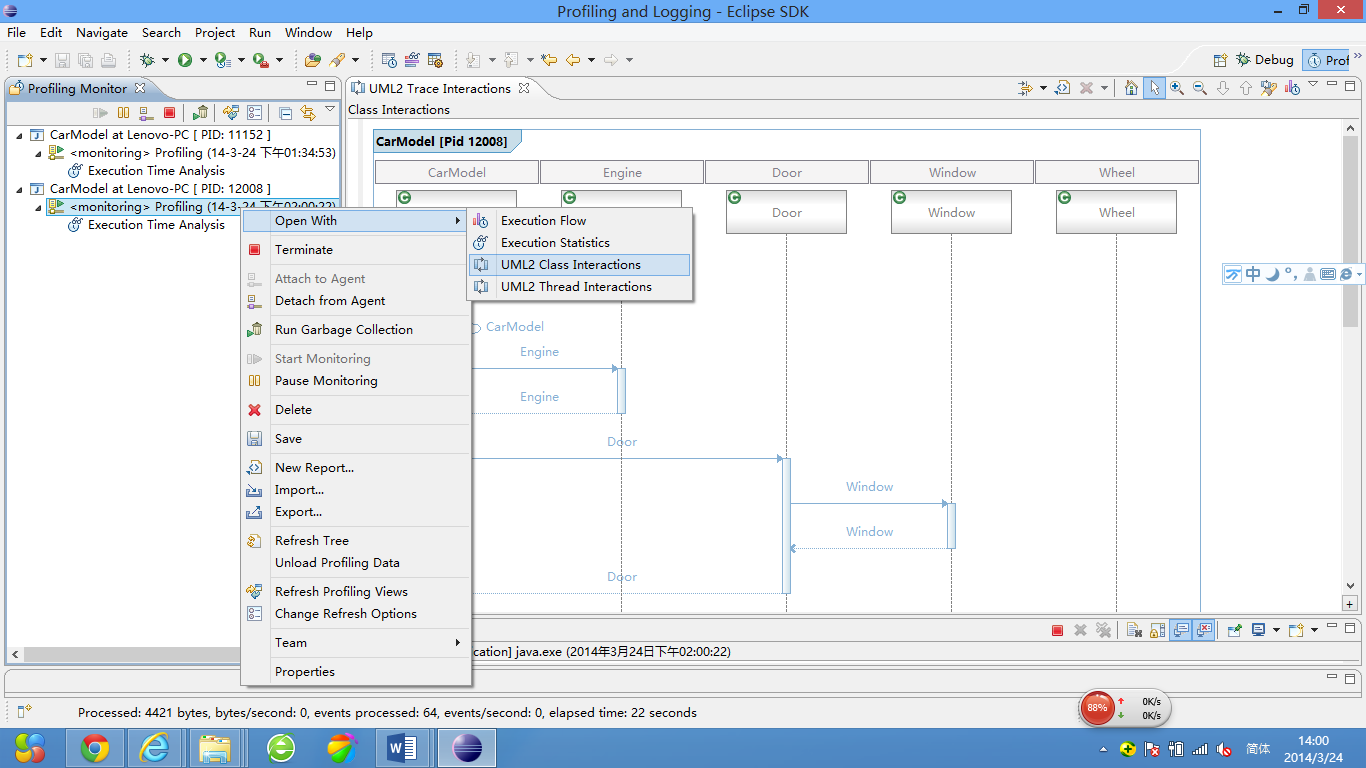




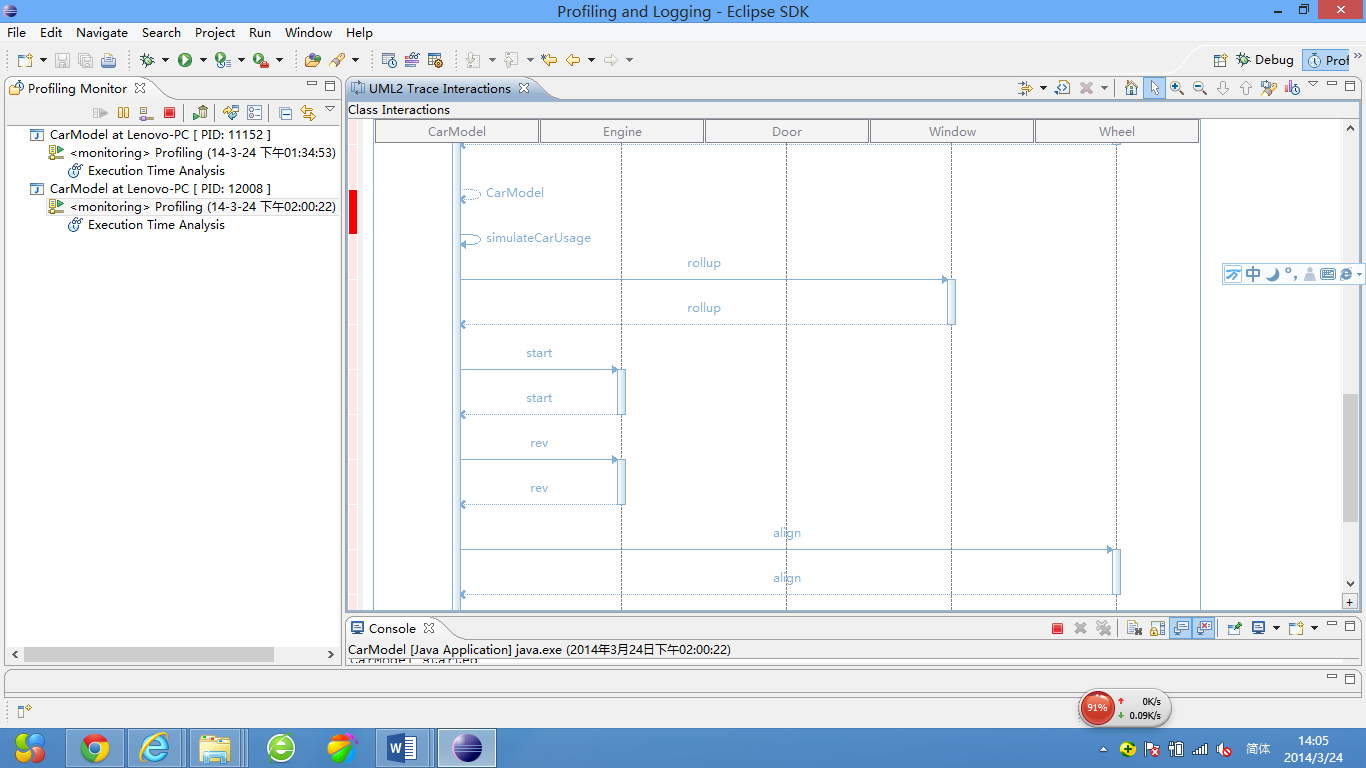
1. 性能分析

回到对话框的主界面，能够进行各种不同类型的Java程序性能分析

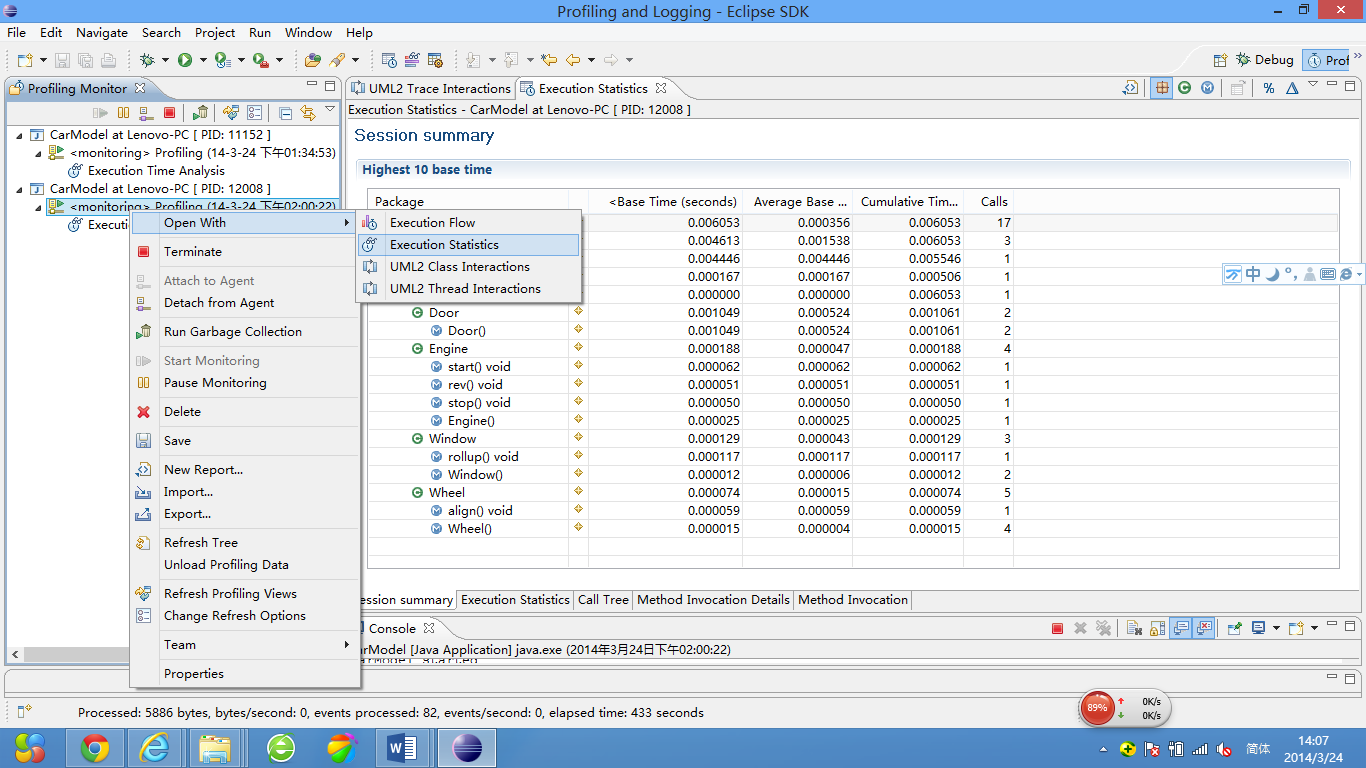
1. Execution Time Analysis: 提供了执行统计信息、图形化的程序执行详细信息、UML2图表生成以及实例级信息收集的功能。



* 选择“open with>UML2 Class Interactions”，获得各个类运行的时序图。打开“Console”工作台，按照提示选择输入，例如，输入“1”。观察程序运行后，原时序图的变化。

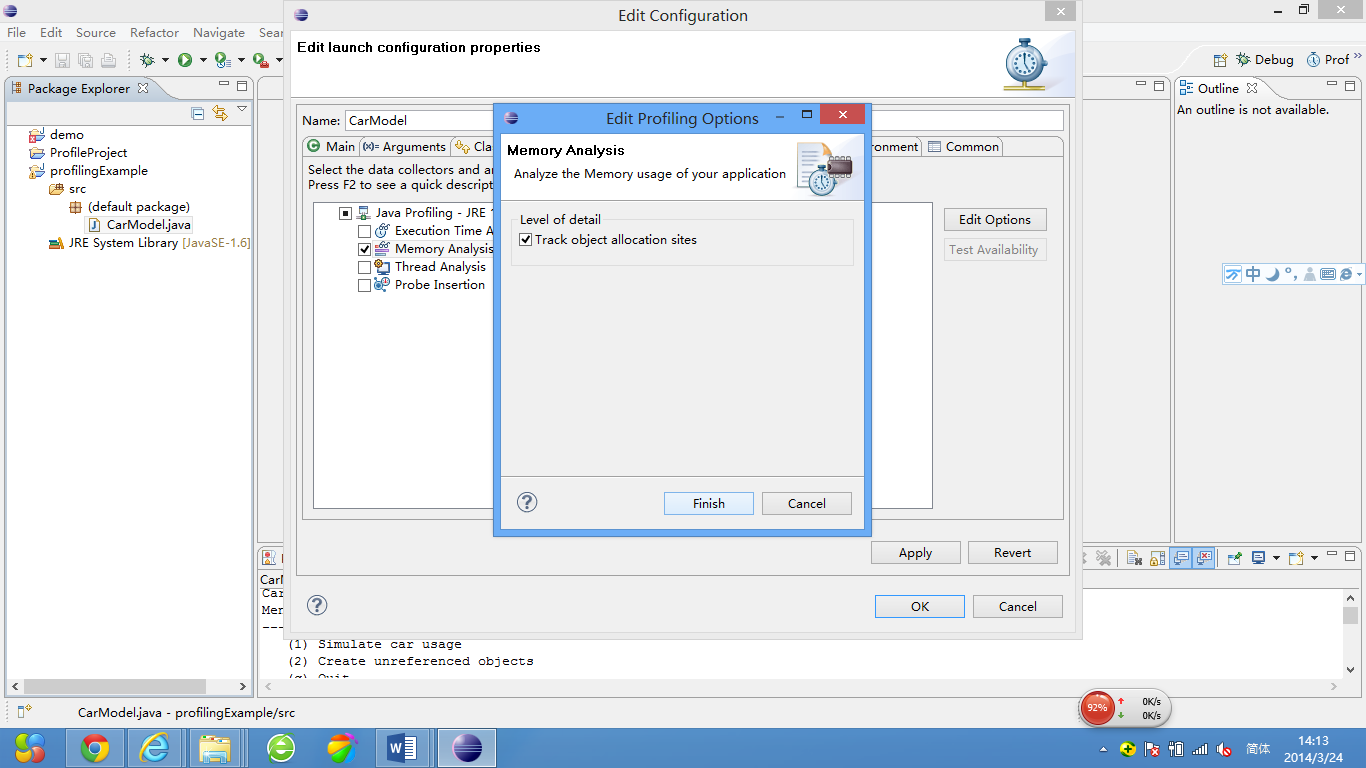


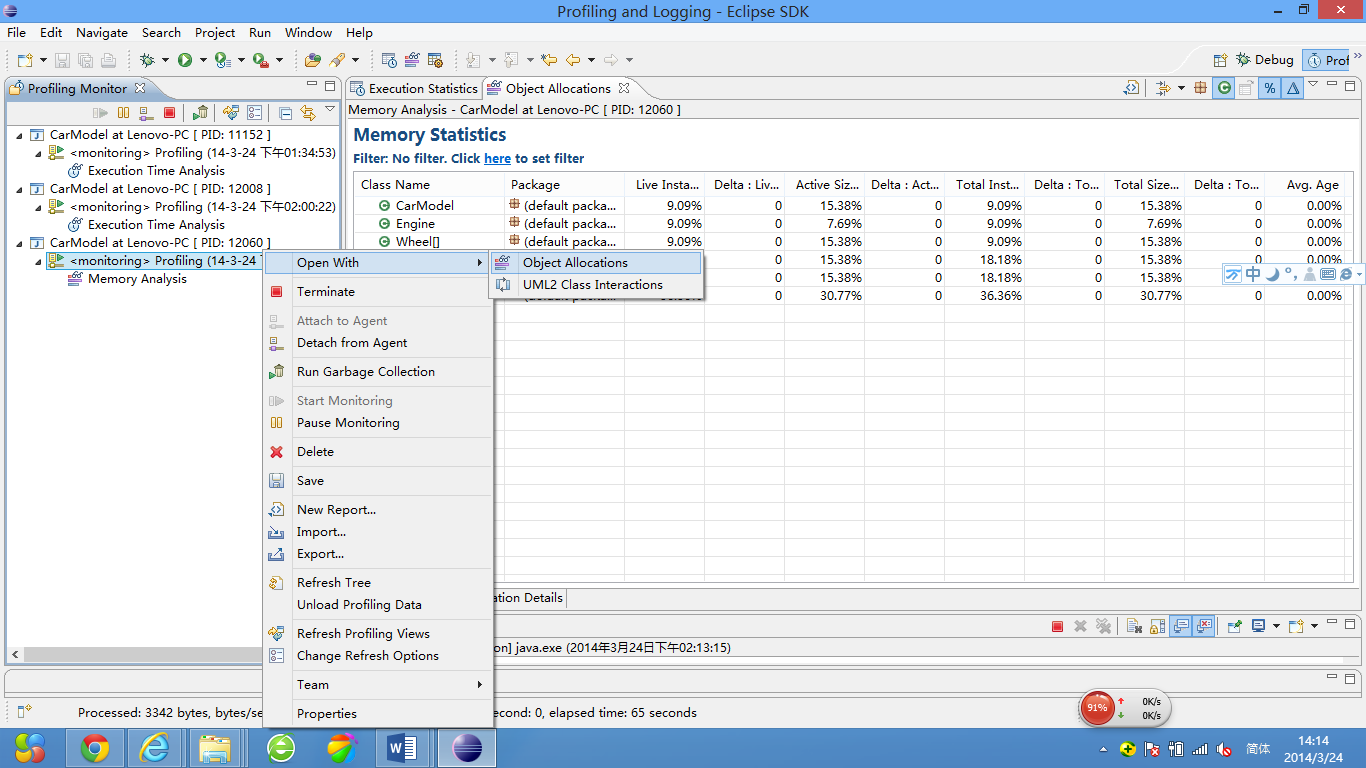
* 选择“open with>Executive Statistics”，观察各个类及方法的运行时间统计分析。打开“Console”工作台，按照提示选择输入，例如，输入“2”。观察程序运行后，原统计数据的变化。



1. Memory Analysis: 提供内在统计信息和对象引用信息

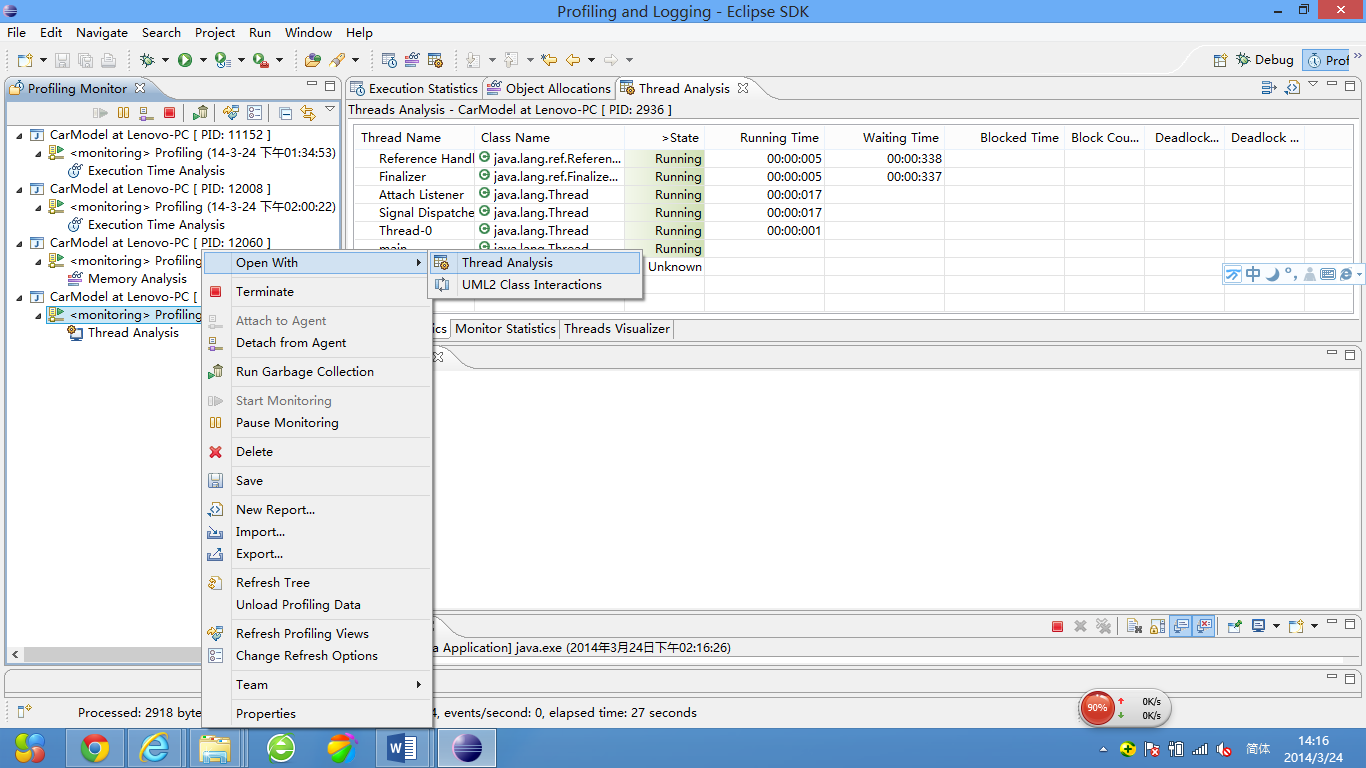
选择“open with>Object Allocations”,观察各个类实例的内存分配情况。





1. Thread Analysis: 提供线程的运行、挂起、阻塞等统计信息和可视化的展示。

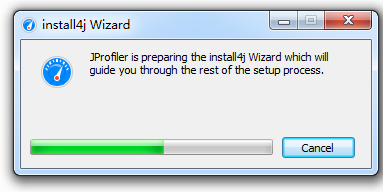
选择“open with>Thread Analysis”,观察各个线程的运行情况。



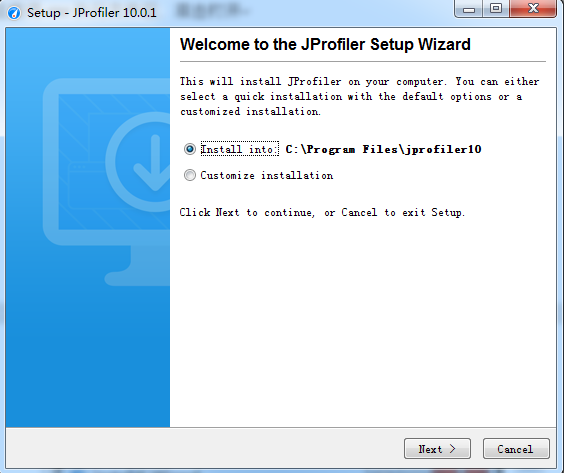
3.2 商业软件Jprofiler

3.2.1 Jprofiler安装

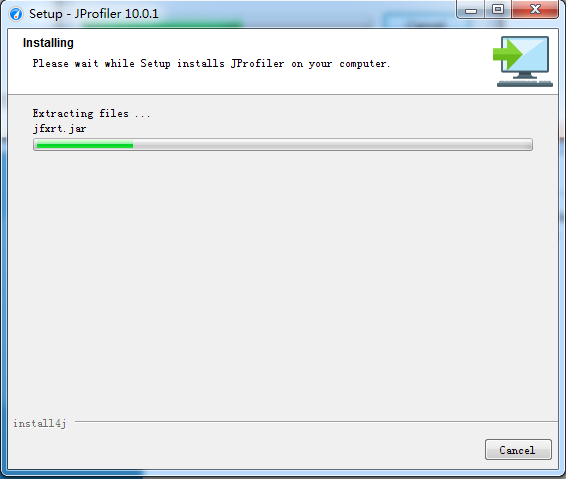
在资源区获得.exe执行文件后，双击打开



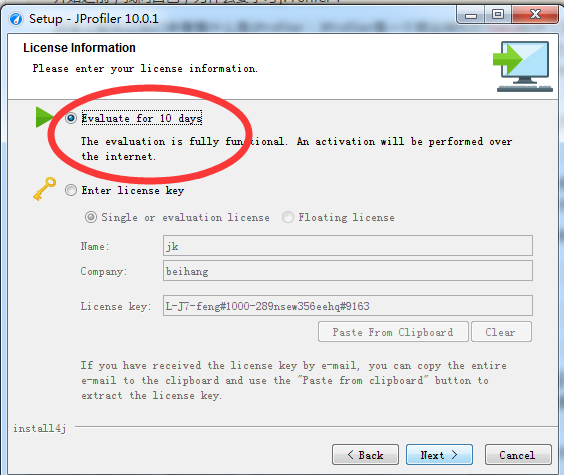
完成后选择默认安装方式



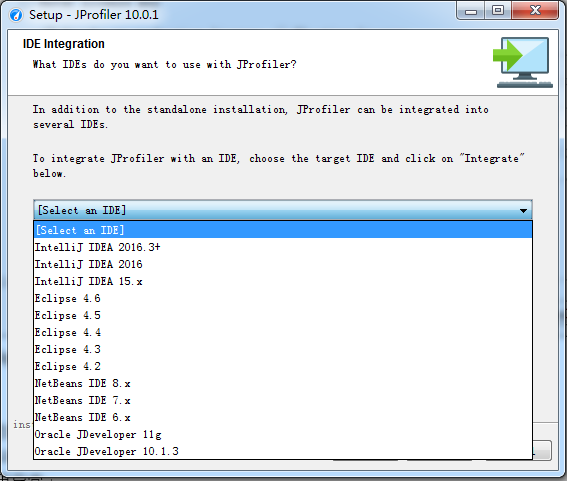
选择accept后等待安装完成



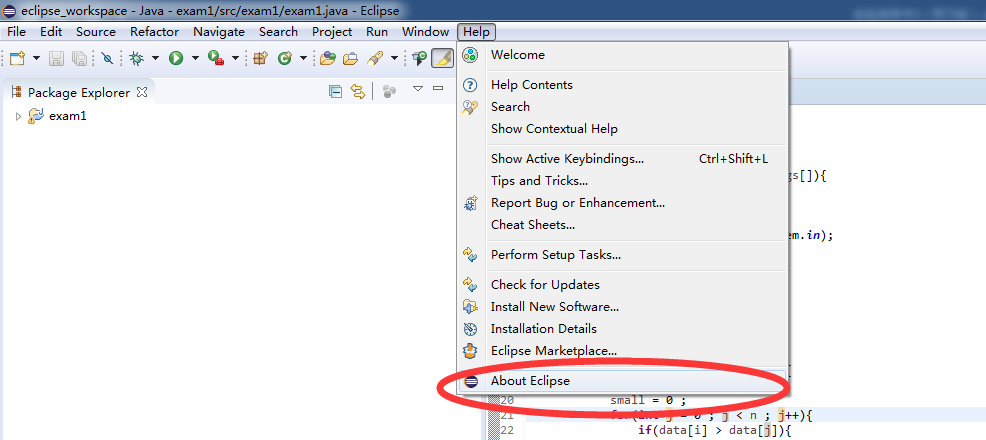
选择10天免费试用版

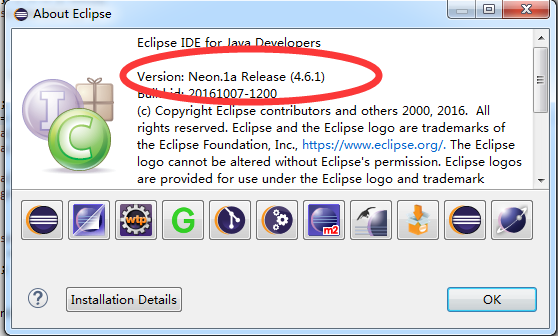


选择eclipse版本

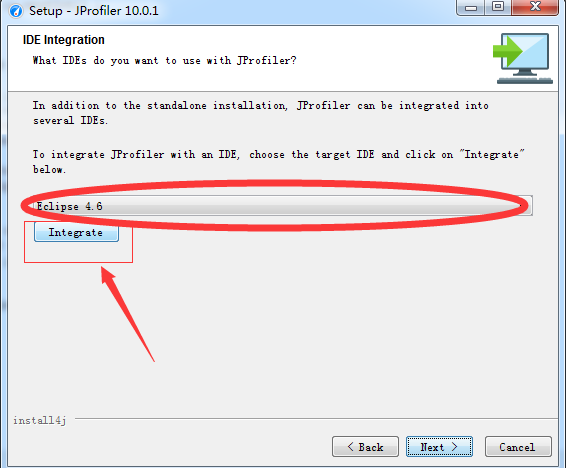


Eclipse版本可以在如下页面进行查看



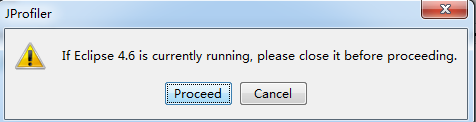


选择4.6.1（根据实际版本情况进行选择）

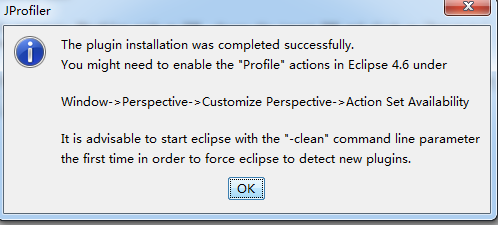


选择当前设备eclipse安装文件夹

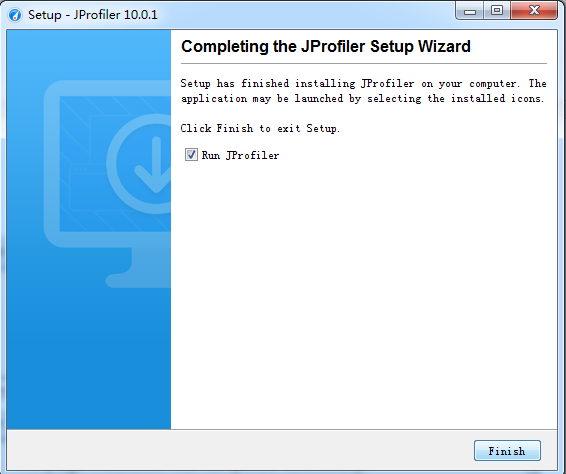
注意：（此时应关闭正在运行的eclipse）



出现如下tip表示成功：

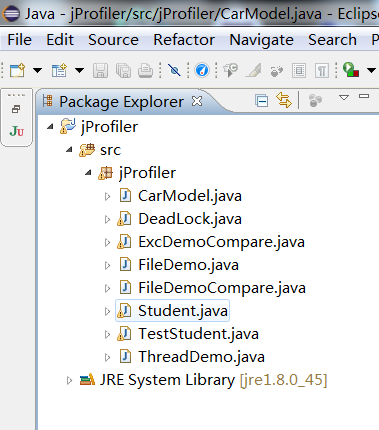


点选OK->next



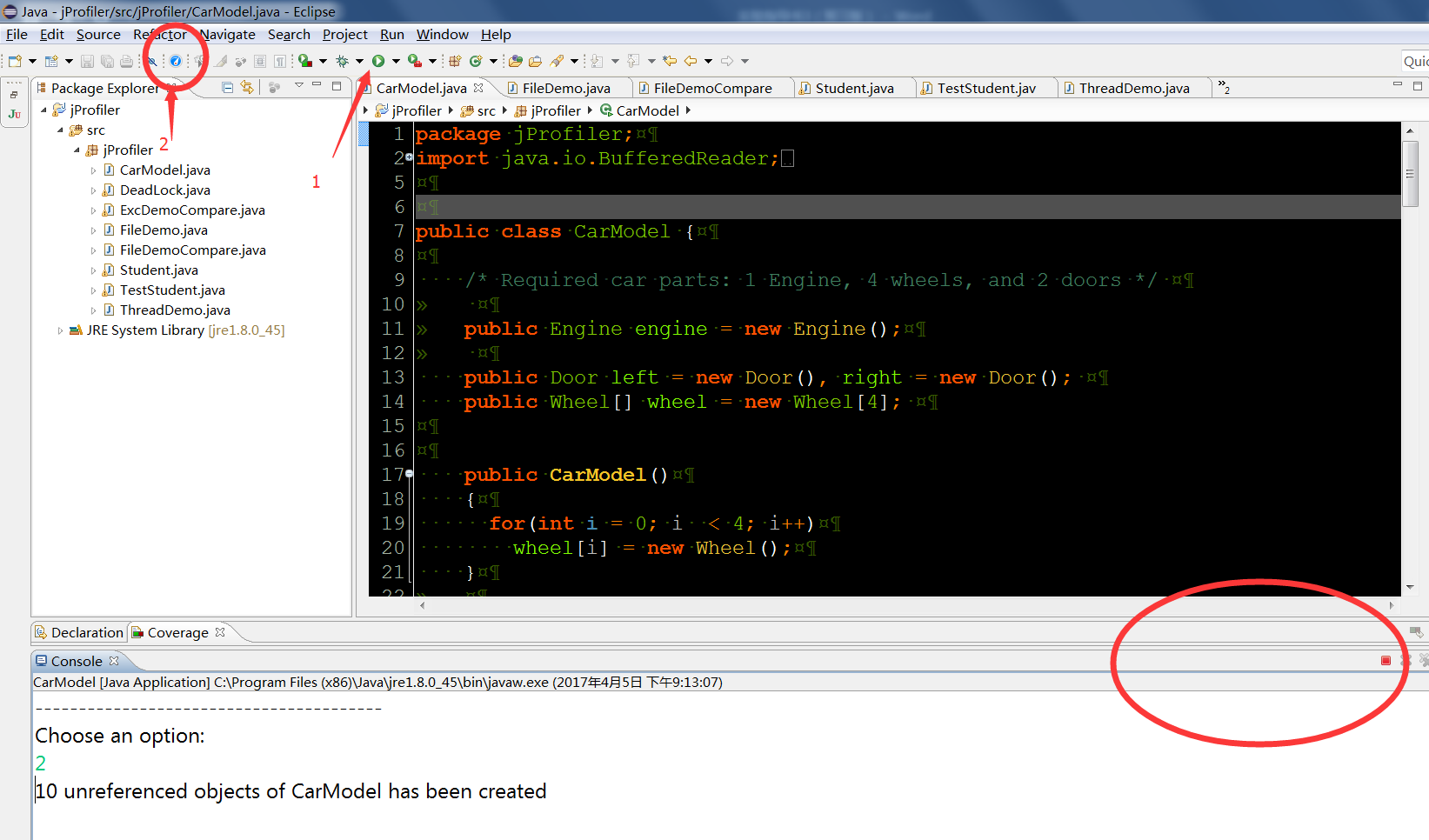
安装完成。

* + 1. Jprofiler使用简明教程
* 新建工程

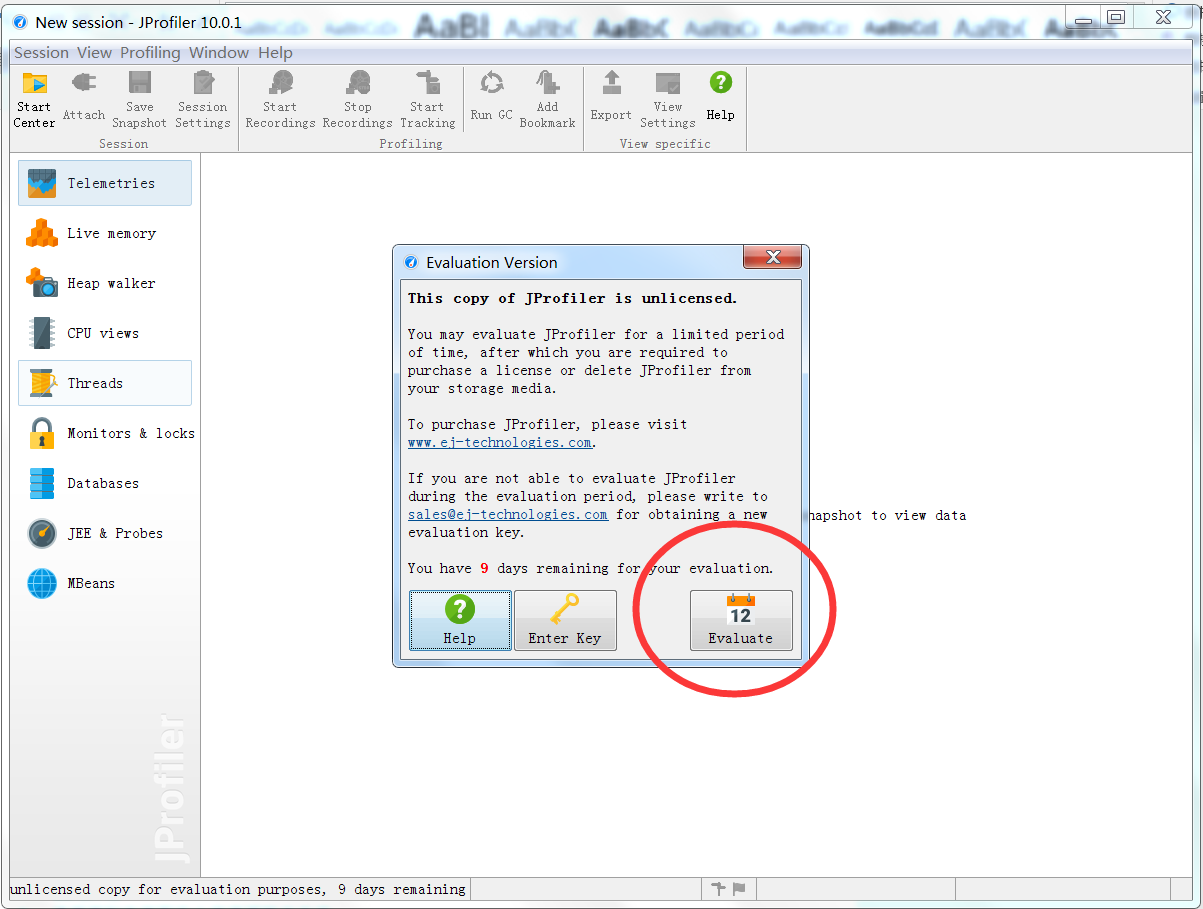


将给出的样例中的java文件导入工程（注意，给每个java文件添加包名）

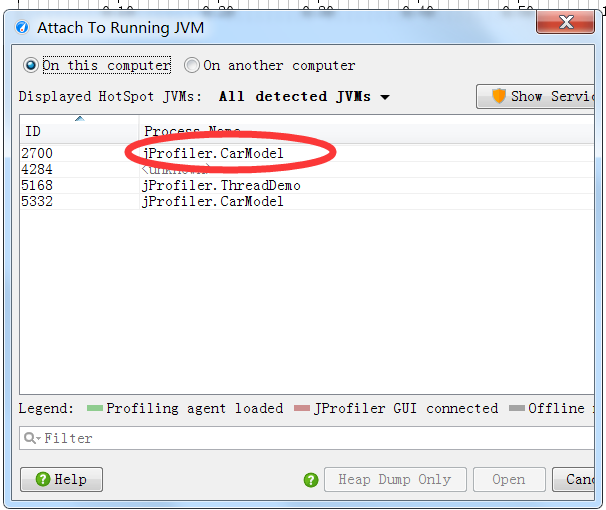
* 运行工程并打开jprofiler



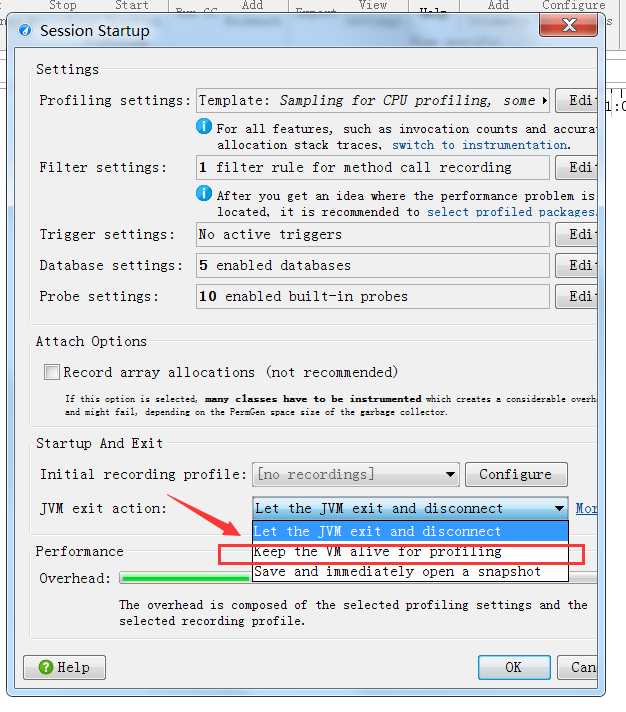
按照以下操作进行



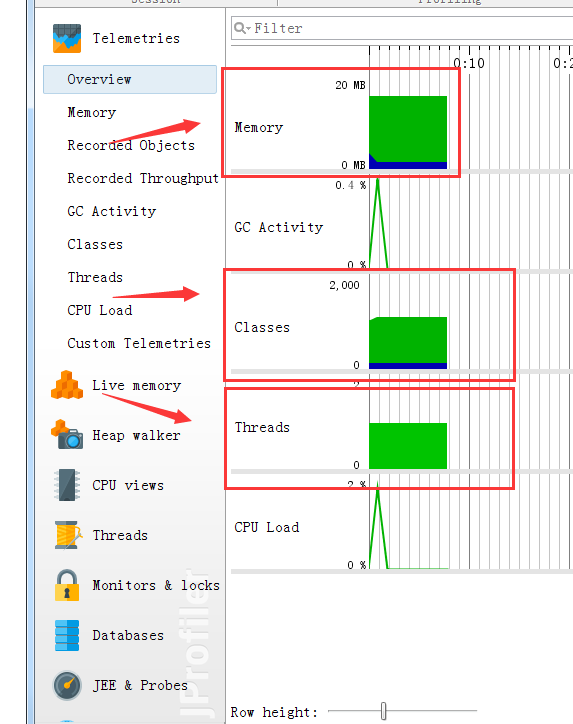
选择当前jvm中正在运行的目标工程



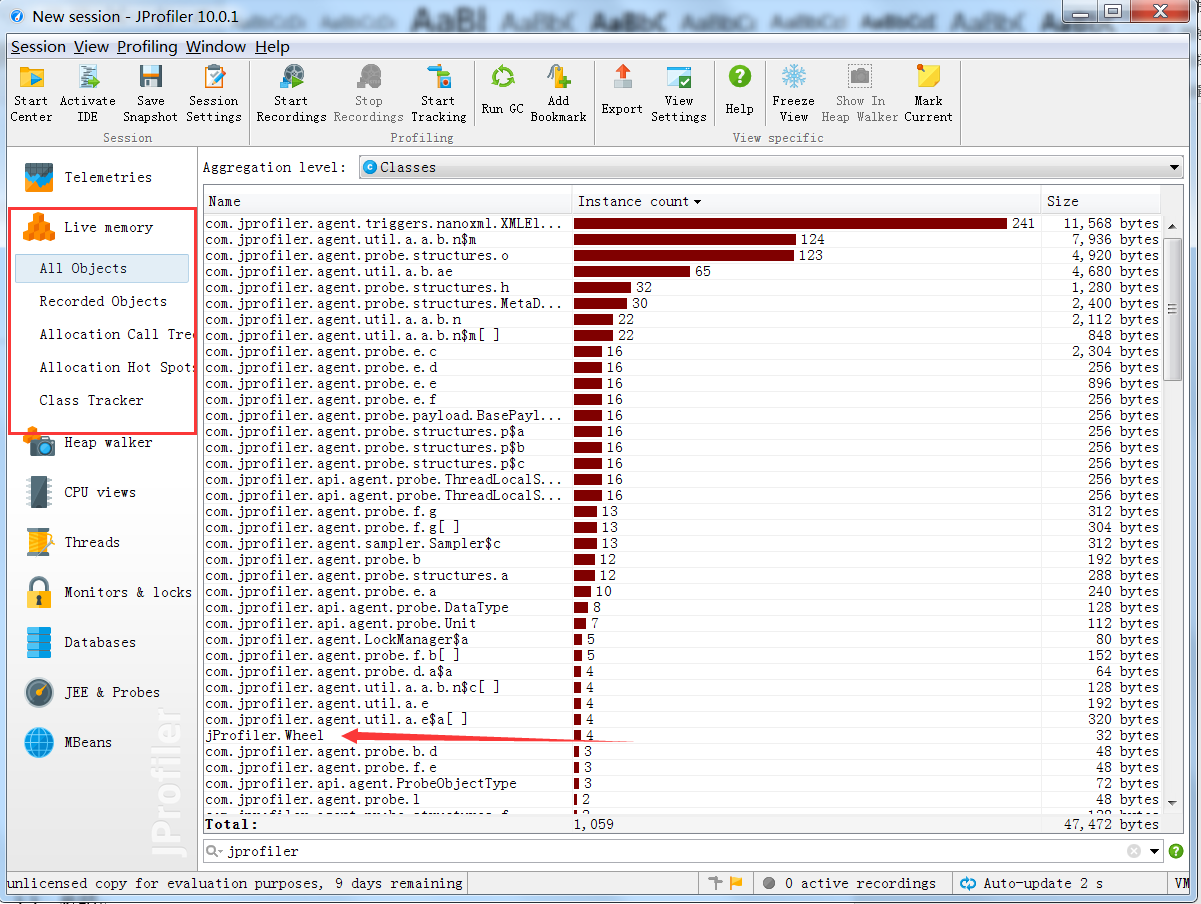
open后，在jvm exit action中



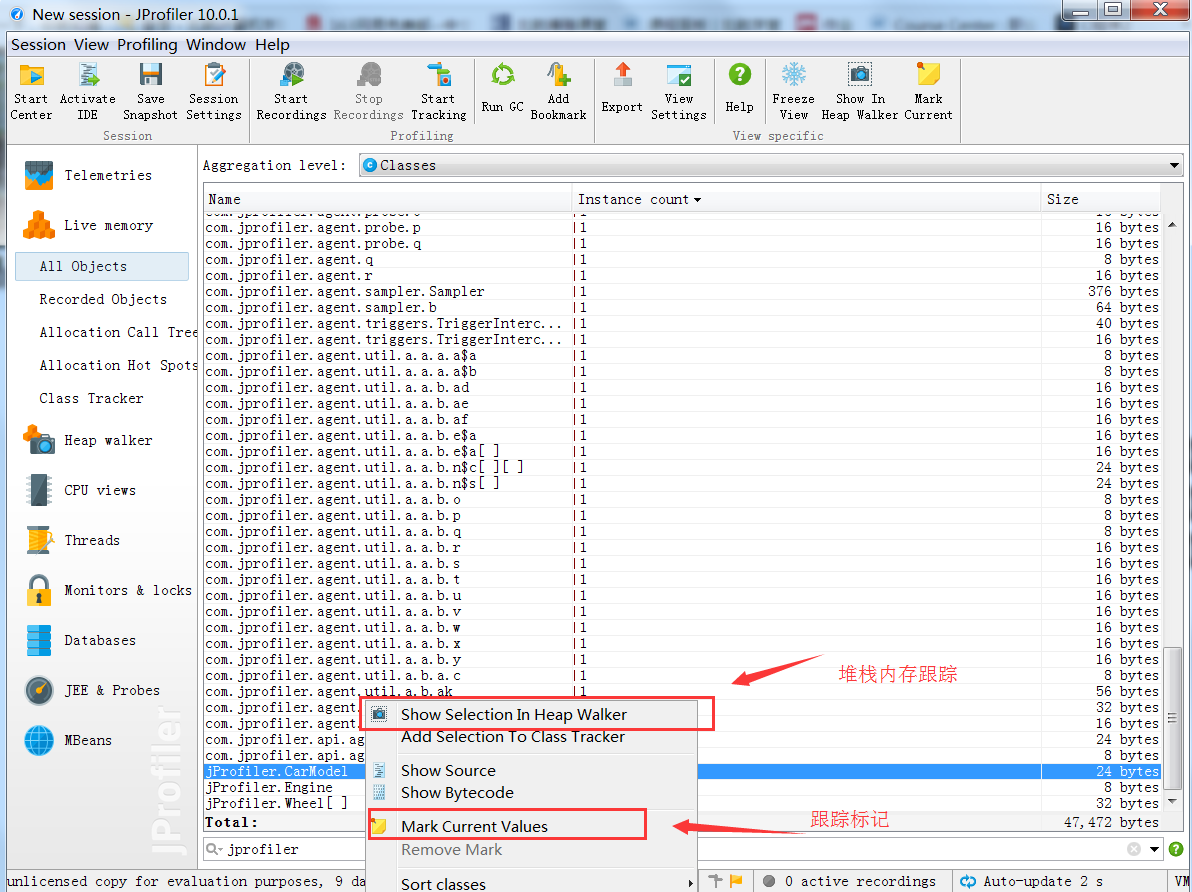
选择ok，进入Jprofiler界面如下，整体概览



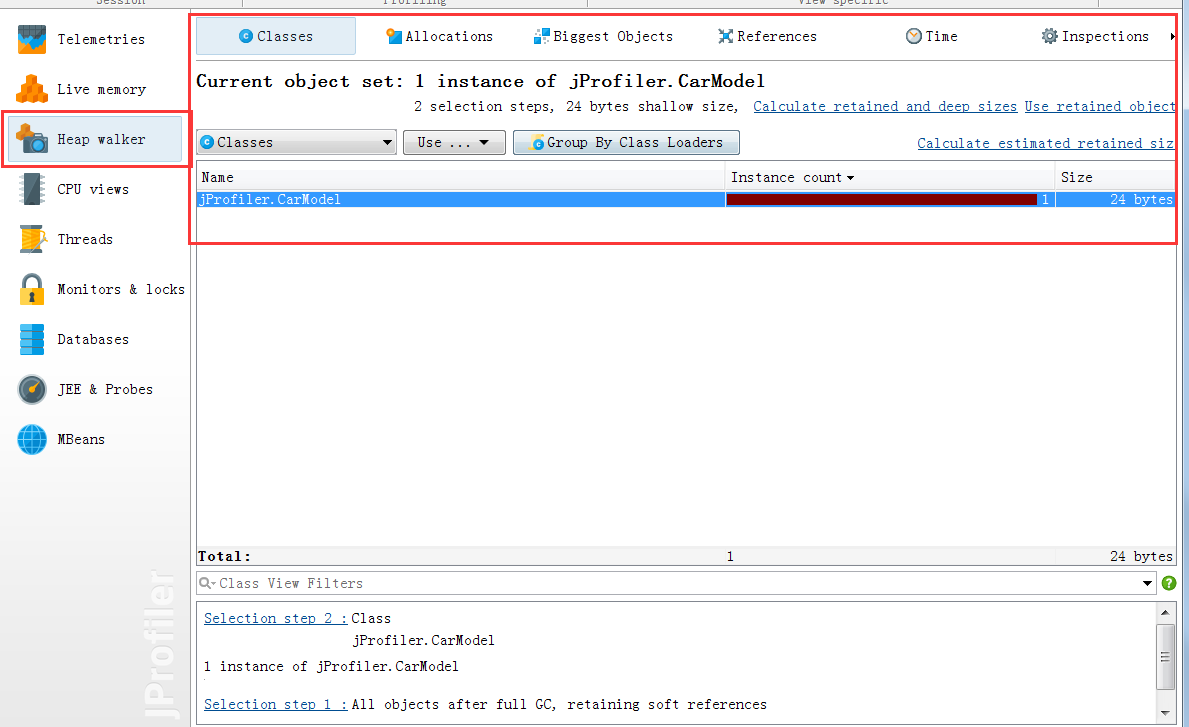
查看memory



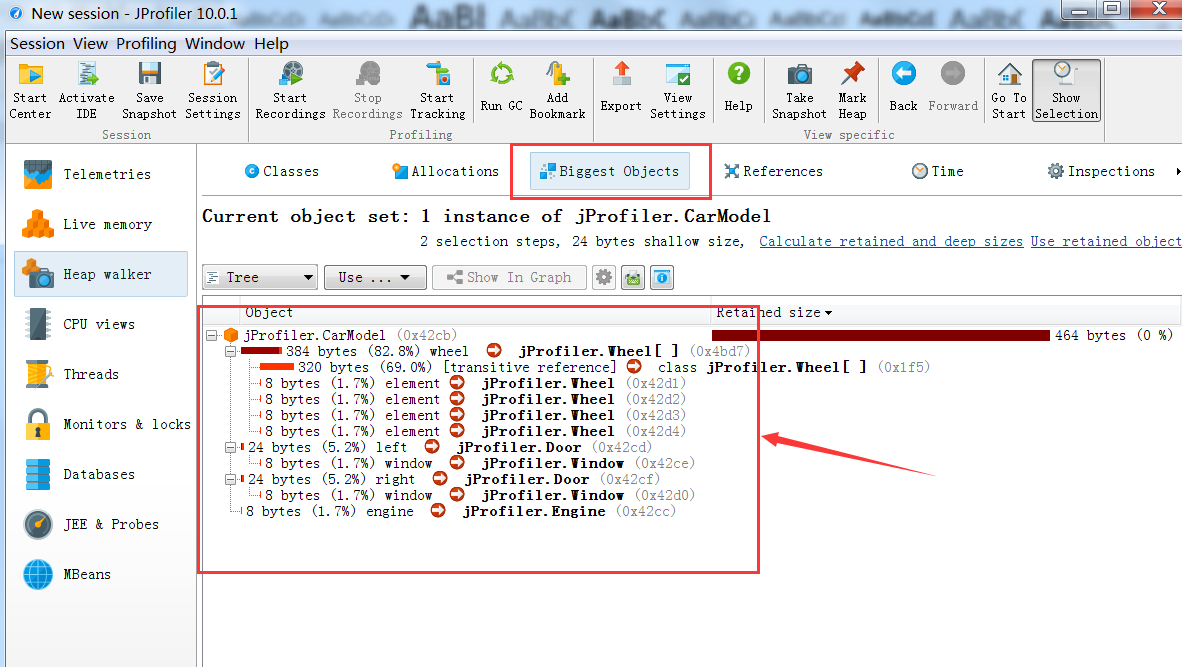
更多当前页面操作



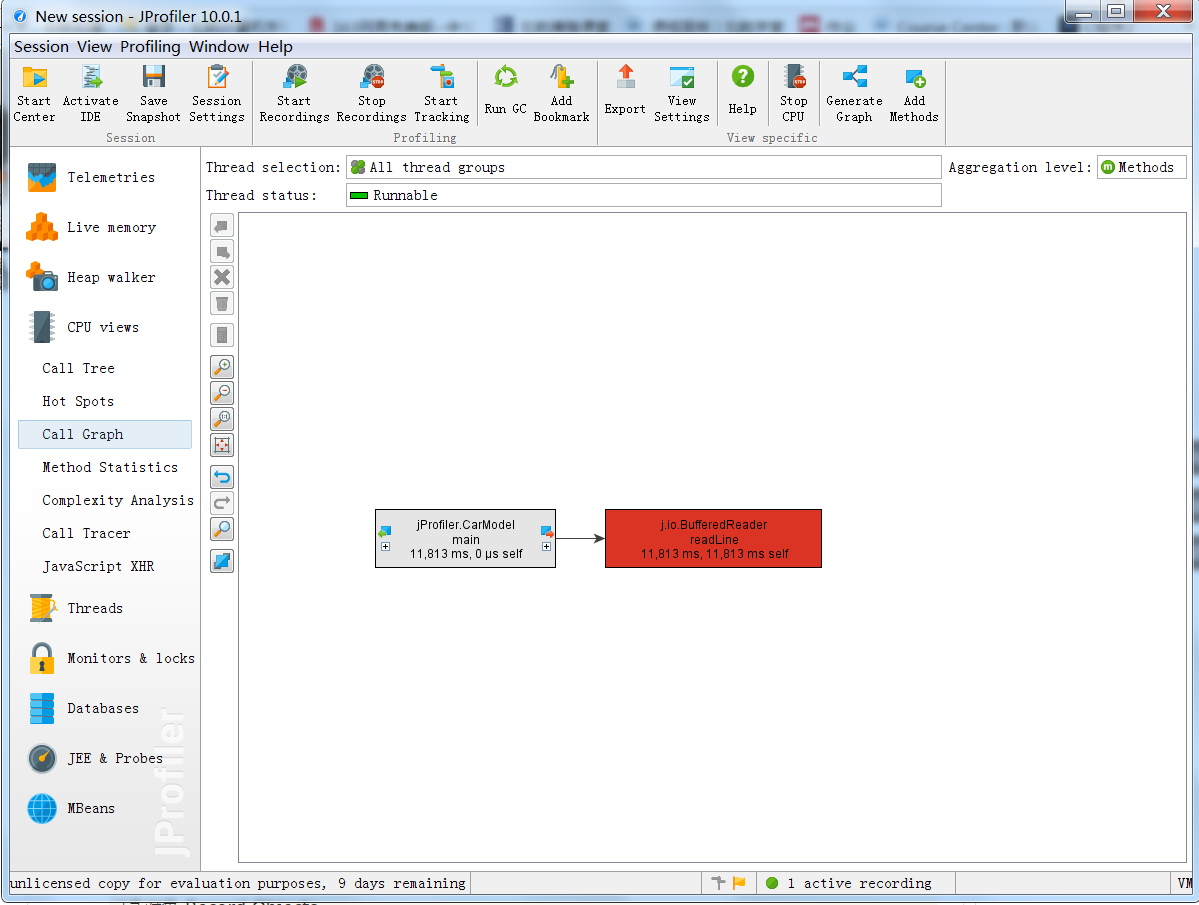
堆中信息跟踪



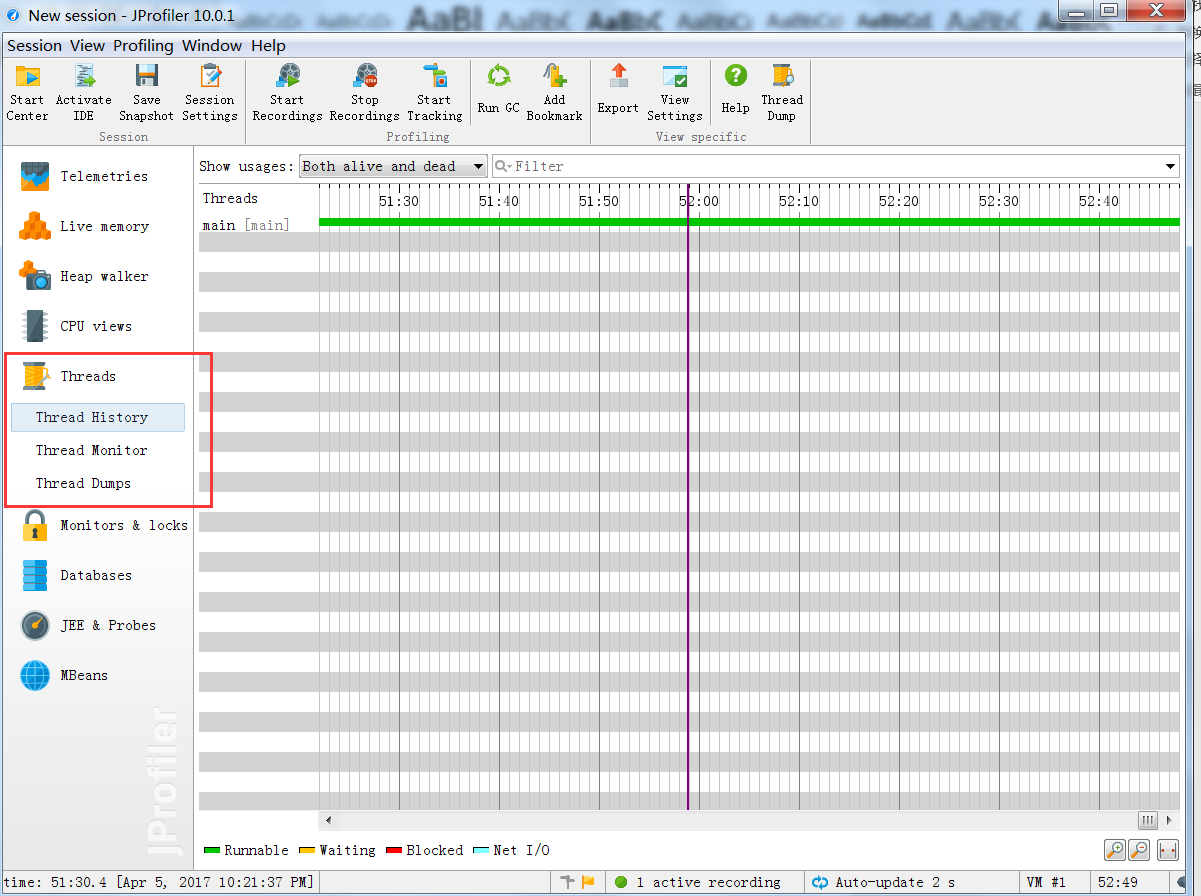
查看类和对象信息



查看调用图



查看线程信息



more information：

更多细节教程参考：

<http://blog.csdn.net/chendc201/article/details/22897999>

3.3 总结

改善JAVA程序的性能有一些小的TIPS，例如但不限于：

* 慎用异常
* 使用局部变量而非大量使用静态变量
* 位运算代替乘除法
* 替换switch
* 一维数组代替二维数组
* 提取表达式
* 展开循环
* 布尔运算代替位运算
* 使用Buffer进行I/O操作
* 使用clone( )代替new
* 静态方法替代实例方法
* ……