Something about Yourkit

Yourkit support all kind of application, any framework and technology and all popular operating systems. You can also do Local and remote profiling

**Local and remote profiling**

* Profile a remote application as easily as a local one:
* Remote EE server integration wizard enables profiling in a console mode
* Remote attach wizard
* Built-in SSH tunneling support simplifies profiling in restricted network conditions

**Controllable overhead**

* Full control of imposed profiling overhead by using appropriate startup options and utilizing profiling modes on demand
* Minor overhead by default: all profiling activities which may cause noticeable overhead are off unless you explicitly turn them on
* EE server integration wizard suggests even stricter default limitations for production use

**Profiling is easy**

* Easiest installation and configuration
* EE server integration wizards
* IDE support allows you to Profile applications from within Eclipse, IntelliJ IDEA, NetBeans, JDeveloper just like you Run or Debug
* Attach mode allows on-demand profiling of JVM instances started without the profiler agent
* If necessary, manually enable profiling of your application by adding a single JVM command line option; no need to tweak classpath etc.

**Developer friendly environment**

* Profile from within your integrated development environment (IDE):
* Plugins for Eclipse, IntelliJ IDEA, NetBeans, JDeveloper
* Profile just as easy as you Run or Debug
* Open exact source code line from the profiling results in your IDE

**CPU profiling - investigate performance issues**

* CPU usage graph showing total, kernel and garbage collector times is always available
* Choose from available profiling modes -- tracing, sampling, call counting -- to balance between result accuracy and fullness and profiling overhead
* Alternatively, estimate CPU usage retrospectively, basing on recorded thread telemetry data
* CPU tracing's adaptive mode automatically controls profiling overhead, user no longer needs to explicitly choose filters to exclude methods from profiling as other profilers require
* CPU views present results as call trees, hot spots, method lists, back traces, merged callees and callees lists
* High-level profiling available: SQL calls, web requests
* What-if feature: to concentrate on what's most important, you can exclude particular method calls from the profiling results, or focus on particular calls only
* Compare snapshots to see what's changed

**Memory profiling - memory leaks, usage, GC**

* Memory usage graphs show heap and non-heap memory pools, GC activity and, if recorded, object creation rate per-second
* Comprehensive heap inspection and analysis
* Object allocation recording to solve garbage collection and memory allocation issues. Available settings allow to balance between result fullness and profiling overhead. In particular, consider the unique object counting mode with almost zero overhead.
* The richest UI to analyze memory usage and find memory leaks
* Paths from roots to analyze memory leaks and object retention, with the ability to immediately see what would happen if particular references were excluded (i.e. to test a proposed memory leak fix effect without re-running the application)
* Find objects holding most memory with dominator tree and class list
* Dissect objects by category, on and on:
* Object explorer to inspect individual objects
* Garbage collection profiling to estimate garbage collector load, and, if garbage collection takes a significant amount of time, pin-point the problematic code
* Automatic inspections detect typical problems. Performing this type of analysis by hand would be a very complicated (if at all possible) task.
* Support of HPROF binary format snapshots and the JVM built-in heap dumper
* Ability to export the profiler format memory snapshots to HPROF binary format for processing with 3rd party tools
* Set description language to use in automated memory tests

**Threads, monitors, exceptions**

* Thread profiling: monitor thread states and stacks, estimate CPU usage in a time range
* Deadlock detector
* Exception profiling: where thrown, of what class and how many
* Monitor profiling: investigate synchronization issues

**High-level results**

* Event recording: in addition to low level profiling results such as method calls, the profiler can record higher level events with their essential properties such as database queries, web requests and I/O calls. Use built-in probes to recognize typical problems or write your own to inspect specifics of your particular application.
* Performance charts show basic and higher-level telemetry graphs of web, database, I/O activity.

**Powerful analysis capabilities**

* Immediately see profiling results as they come, in real time
* Capture a snapshot for a more sophisticated analysis, to save the results for history records, share them
* Compare performance or memory snapshots to see what's changed
* Lightweight basic telemetry graphs are always available. Very useful and unique feature is that the recent telemetry results are remembered inside the profiler agent. This allows you to connect to the profiled application on demand, being able to discover how the application behaved in the past.
* Automatic inspections detect typical problems. Performing this type of analysis by hand would be a very complicated (if at all possible) task.

**Automated profiling**

* In addition to the rich UI capabilities for interactive profiling, profiling can also be conducted semi- or fully automatically.
* Startup options to immediately activate desired profiling modes
* Triggers to perform actions on events such as particular method call or a monitored value reaching its threshold
* Probes to execute arbitrary code on invocation of particular methods
* API to control profiling of this or another application: switch profiling modes and store profiling results
* Command line tool to perform some profiling tasks without using the UI

**Export to external formats**

* In addition to the rich analysis capabilities of the profiler, the profiled results can be exported to external form to process with 3rd party tools or share with those not having the profiler.
* export to XML, HTML, CSV, plain text or image
* export from UI
* export automatically via a command line tool
* export memory snapshots in the profiler format to widely recognized HPROF binary format
* in tables and graphs, selected result text can be copied to clipboard

**Running applications with the profiler:**

The profiler agent

To profile a Java application, be it local or remote, the profiler agent should be loaded into the JVM.

Approach 1: Start JVM with the profiler agent

This approach is recommended as it provides the full set of profiling capabilities. To learn how to apply it, see appropriate subtopic: Local profiling, Remote profiling

Approach 2: Attach the profiler to a running JVM instance started without the agent

Attaching the profiler agent to a running JVM instance simplifies profiling, as it avoids a special step to enable profiling: any running Java application can be profiled on demand.

However, attaching to a running JVM is not always possible, and some profiling features are not available.