

JProfiler

JProfiler supports the following modes of operation:

✓ Live profiling of a local session

Once you define how your application is started, JProfiler can profile it and you immediately see live data from the profiled JVM. To eliminate the need for session configuration, you can use one of the [many IDE plugins](#) to profile the application from within your favorite IDE.

✓ Live profiling of a remote session

By modifying the VM parameters of the java start command you can get any Java application to listen for a connection from the JProfiler GUI. The profiled application can not only run on your local computer, JProfiler can attach to a profiled application over the network. In addition, JProfiler provides [numerous integration wizards](#) for all popular application servers that help you in setting up your application for profiling.

✓ Offline profiling

You do not have to connect with the JProfiler GUI to the profiled application in order to profile it: With [offline profiling](#) you can use the JProfiler API to control the profiling agent and save snapshots to disk. At a later time you can open these snapshots in the JProfiler GUI or programmatically export profiling views with the command line export tool or the export ant task.

✓ Snapshot comparisons

In JProfiler, you can save a snapshot of all current profiling data to disk. JProfiler offers a rich comparison facility to see what has changed between two or more snapshots. Alternatively you can create comparison reports programmatically with the command line comparison tool or the comparison ant task.

The following list gives a **high level overview** of the profiling views in JProfiler:

Memory profiling

JProfiler's memory view section offers dynamically updated views on memory usage and views that show information about allocations spots. All views have several aggregation levels and can show live and garbage collected objects.

✓ All objects

Shows classes or packages of all objects on the heap with instance counts and size

information. You can mark current values and show differences.



Recorded objects

Shows classes or packages of all recorded objects. You can mark current values and show differences.



Allocation call tree

Shows a call tree or methods, classes, packages or J2EE components with annotated allocations of selected classes.



Allocation hot spots

Shows a list of methods, classes, packages or J2EE components that allocate selected classes. You can mark current values and show differences. The tree of backtraces can be shown for each hot spot.



Heap walker

In JProfiler's heap walker you can take a snapshot of the heap and drill down to objects of interest by performing selection steps. The heap walker has five views:



Classes

Shows all classes and their instances.



Allocations

Shows allocation tree and allocation hot spots for recorded objects.



References

Shows a graph of references for individual objects and offers a "show path to garbage collector root" functionality. Also offers cumulated views for incoming and outgoing references.



Data

Shows instance and class data for individual objects.



Time

Shows a time-resolved histogram of recorded objects.



CPU profiling

JProfiler offers various ways to record the call tree to optimize for performance or detail. The thread or thread group as well as the thread status can be chosen for all views. All views can be aggregated

on a method, class, package or J2EE component level. The CPU view section contains:

- ✓ **Call tree**
Shows a cumulated top-down tree of all recorded call sequences in the JVM. JDBC, JMS and JNDI service calls are annotated into the call tree. The call tree can be split for different request URL to a servlet or JSP.
- ✓ **Hot spots**
Shows the list of the most time consuming methods. The tree of backtraces can be shown for each hot spot. Hotspots can be calculated for method calls, JDBC, JMS and JNDI service calls, as well as for URL invocations.
- ✓ **Call graph**
Shows a graph of call sequences starting from selected methods, classes, packages or J2EE components.



Thread profiling

For thread profiling, JProfiler offers the following views:

- ✓ **Thread history**
Shows a timeline with thread activity and thread status.
- ✓ **Thread monitor**
Shows a list of all live threads with their current activity.
- ✓ **Deadlock detection graph**
Shows a graph of all deadlocks in the JVM.
- ✓ **Current monitor usage**
Shows the currently used monitors and their associated threads.
- ✓ **Monitor usage history**
Shows the history of significant waiting and blocking events.
- ✓ **Monitor usage statistics**
Shows statistics for monitors grouped by monitors, threads and classes of monitors.



VM telemetry

To observe the internal state of your JVM, JProfiler offers various telemetry views:

- ✓ **Heap**

Shows a timeline with a graph of the used heap and the heap size.



Objects

Shows a timeline with a graph of live objects and arrays.



Garbage collector

Shows a timeline with a graph of garbage collector activity.



Classes

Shows a timeline with a graph of loaded classes.



Threads

Shows a timeline with a graph of active threads.