**Recording Objects**

1. **Introduction**

By default, JProfiler does not track the creation of all objects. This reduces the runtime overhead of the profiling agent regarding execution speed as well as memory consumption.

However, allocation recording is not only a way to increase runtime performance, it also helps you to focus on important parts of your application and to reduce clutter(零碎的) in the memory views.

Imagine you have a web application that's started in the framework of an application server. The server allocates a huge number of objects in a great number of classes. If you want to focus on the objects created by your web application, the objects from the server startup will be in the way（服务器启动后分配的对象）. In JProfiler, you can start allocation recording before you perform a certain action 可以在进行一定动作后进行分配记录这样能简化显示那个动作后导致分配的对象。and so reduce the displayed objects to those that are allocated as a direct consequence of that action.

**2. Starting and stopping allocation recording**

The profiler menu as well as the toolbar allow you to start and stop allocation recording.

If no allocations have ever been recorded, the dynamic memory views show placeholders with the corresponding "record" button.

If you wish to enable allocation recording for the entire application run, you can do so in the profiling settings dialog

When you stop allocation recording, the garbage collection of the recorded objects will still be tracked by the dynamic memory views. In this way you can observe if the objects created during a certain period of time are actually garbage collected at some point. Please note that the manual garbage collection button in JProfiler just invokes the System.gc() method. This leads to a full GC in 1.3 JREs where the garbage collector makes the best effort to remove all unreferenced objects. However, 1.4 and 1.5 JREs perform incremental garbage collection, so full garbage collection is not available when working with such a recent JRE. To check if the remaining objects are really referenced, or if the garbage collector just doesn't feel like collecting them yet, you can take a heap snapshot. The heap walker offers the option "Remove unreferenced and weakly referenced objects" which is the equivalent of a full GC.

JProfiler also keeps statistics on garbage collected objects. All dynamic memory views have a mode selector where you can choose whether to display only live objects on the heap, only garbage collected objects, or both of them.

When you have stopped allocation recording and you restart it, the previous contents of the dynamic memory views will be deleted. In this way, allocation recording gives you the ability to do differencing of the heap between two points in time.

If you have very specific requirements as to where allocation recording should start and stop, you can use the [offline profiling API](../../../doc/offline/api.html) to control allocation recording programmatically.

**3. Implications of unrecorded objects**

For "unrecorded" objects there are the following implications:

* JProfiler does not know the allocation spot for an unrecorded object. This becomes apparent in the heap walker. The heap walker takes a heap snapshot and is able to show all objects on the heap, however, the allocation information is not available from the JVMPI/JVMTI and the "Allocations" view will contain top-level method nodes that are labeled as "Unrecorded objects".
* JProfiler does not know the class name for an unrecorded object. This influences the monitor views and locking graphs where JProfiler is only able to display the name of a monitor object if the object has been recorded.

The object graph in the VM telemetry views is not affected by allocation recording.

**4. Allocation recording and the heap walker**

In the heap walker options dialog that is displayed before a heap snapshot is taken, the first option is labeled "Select recorded objects".

This allows you to work with a set of objects that has been created during a certain period of time. This is just an initial selection step and does not mean that the heap walker will discard all unrecorded objects. In the reference view you can still reach all referenced and referencing objects and create a new object set with unrecorded objects.

If you use the "take heap snapshot with selection" action in the dynamic memory views, the number of selected objects will only match approximately, if "Select recorded objects" is checked and "Remove unreferenced and weakly referenced objects" is not checked in the heap walker options dialog. The numbers might still not match exactly since the dynamic memory views can change in time while a heap snapshot is fixed.