BEA JRockit® Mission Control™

Monitoring Thread Activity with the JRockit Management Console

ProductNameShort 3.0.2
Document Revised: June, 2008

Contents

Introduction to Monitoring Threads

CHAPTER 1

Introduction to Monitoring Threads

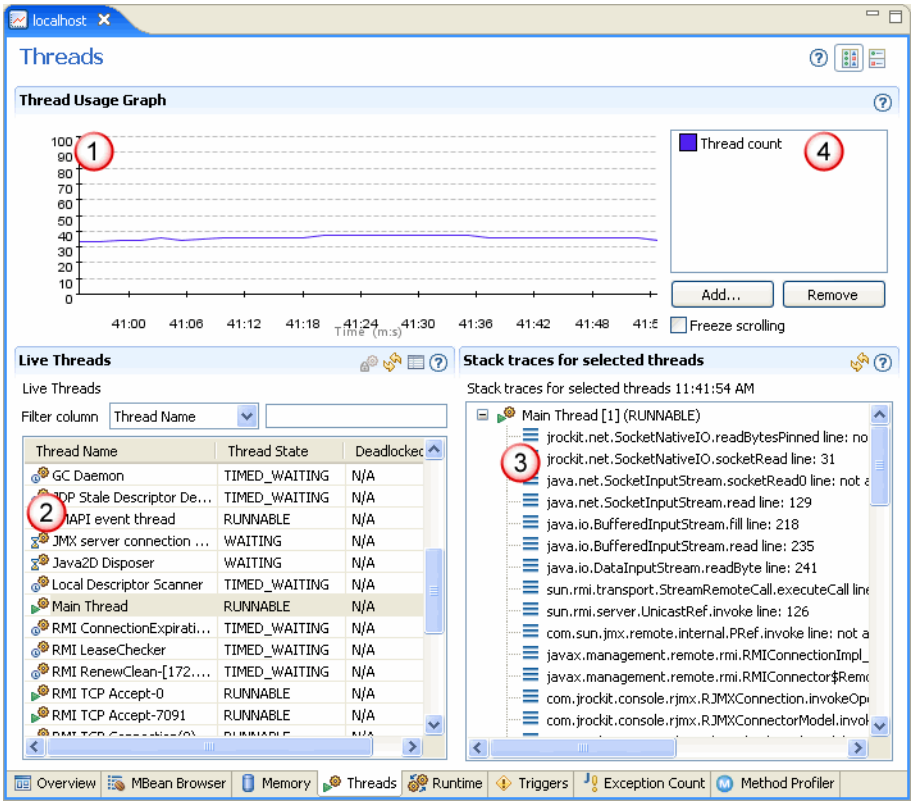
You can monitor thread activity for a running application by using the **Threads** tab on the Management Console. This tab contains both a graph that plots thread usage by an application over time and a sortable list of all live threads used by the application. It also displays thread stacktraces.

This topic includes information on the following subjects:

Getting Familiar with the Threads Tab

The **Threads** tab (Figure 1-1) displays thread usage, which threads are live, and stack traces of the threads.

Figure 1-1 The Threads Tab



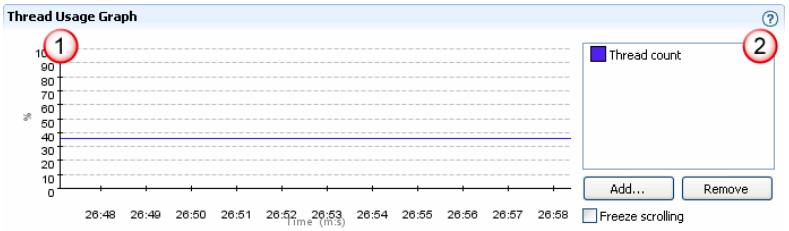
The **Threads** tab is divided into the following sections:

1. **Thread Usage Graph**—shows the number of available threads in use, by the application.
2. **Live Threads**—a listing of all live threads, along with such information as thread state. You can add columns to this table to show information about, for example, lock name, lock owner names and IDs, and wait count and time.
3. **Stacktraces for selected threads**— a table that shows the stacktrace for a thread when that thread is selected on the **Live Threads** table.
4. **Thread Usage Graph** legend—this legend lets you define attributes that you wish to monitor.

Viewing Thread Usage Information

The **Thread Usage Graph** (see Figure 1-2) plots the number of available threads in use, over time, by an application.

Figure 1-2 Thread Usage graph



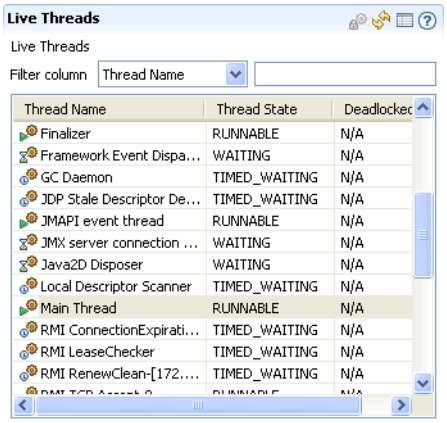
By default, upon Management Console startup, the **Thread Usage Graph** (marked 1 in Figure 1-2) shows the attribute *Thread count*. The attributes are identified in the legend to the right of the graph (marked 2 in Figure 1-2).

Please refer to Threads Tab Functionality for a description of the functions you can use with this graph.

Viewing Live Threads Table

You can monitor live thread activity on the **Live Threads** table (Figure 1-3). This table shows all live threads in use by an application. Along with the thread name, other information about live threads is presented on the table, such as thread state and whether or not the thread is suspended. You can add columns to this table to show information such as the lock name, the lock owner names and IDs, the wait count and time, and so on. You can also use the **Live Threads** table to display stacktraces for each live thread.

Figure 1-3 Live Threads Table



By default, when the Management Console opens, the table shows the following information:

- **Thread Name**—the user-supplied name of the thread and name-related information.
- **Thread State**—identifies the thread's state; for example, **NEW**, **RUNNABLE**, **BLOCKED, WAITING**, **TIMED_WAITING**, and **TERMINATED**. For more information on thread states, please refer to the Javadoc for `java.lang.Thread.State`.
- **Deadlocked**—identifies whether or not the thread is deadlocked. This value will be either **Yes**, the tread is deadlocked, **No**, the thread is not deadlocked, or **N/A**, deadlock data is not available.

Please refer to Threads Tab Functionality for a description of the functions you can use with this table.

Data Columns in Live Threads Table Described

By default, the **Live Threads** table only displays three columns of data, but in fact, you can display up to 13 data elements for each thread by selecting them on the **Table Settings** dialog box.

The data elements you can add are listed in Table 1-1.

Table 1-1 Live Thread Data Elements

Data Element	Description
Thread Name	The name of the Thread.
Blocked Count	The total number of times that the thread blocked to enter or reenter a monitor.
Blocked Time	The approximate accumulated elapsed time (in milliseconds) that the thread has blocked to enter or reenter a monitor since thread contention monitoring was enabled.
Lock Name	The string representation of the monitor lock that the thread is blocked to enter or waiting to be notified through the Object.wait method.
Lock Owner ID	The ID of the thread which holds the monitor lock of an object on which the thread is blocking.
Lock Owner Name	The name of the thread which holds the monitor lock of an object on which the thread is blocking.
Thread ID	The ID of the thread.
Thread State	The state of the thread.
Waited Count	The total number of times that the thread waited for notification.
Waited Time	The approximate accumulated elapsed time (in milliseconds) the thread has waited for notification since thread contention monitoring was enabled.
Native	True if the thread is executing native code via the Java Native Interface (JNI).
Suspended	Identifies whether or not the thread is suspended. This value will be either **Yes**, the thread is suspended or **No**, the thread is not suspended.
Deadlocked	Identifies whether or not the thread is deadlocked. This value will be either **Yes**, the tread is deadlocked, **No**, the thread is not deadlocked, or **N/A**, deadlock data is not available.

Finding Dead Locked Threads

The **Threads** tab contains a function for finding dead locked threads in runtime. Combining the finding of dead locked threads and their stack traces provides a powerful way to find problems in your applications.

To find dead locked threads

- Click the **Dead Lock** button (Figure 1-4).

Figure 1-4 Dead Lock button

Note: When the **Dead Lock** button is active, it slows down your application. Turn off the button as soon as you have found the dead locks.

Viewing Thread Stacktraces

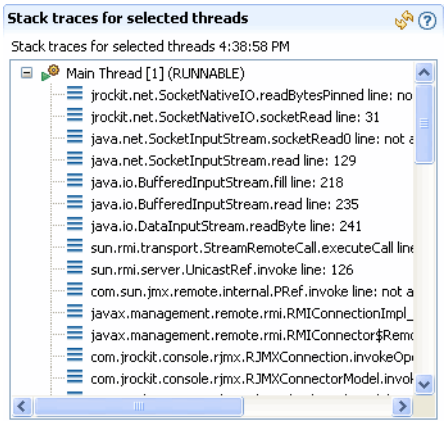
You can display stacktraces for individual threads (Figure 1-5). The stacktrace displays a snapshot of where in your application the computer is executing instructions, at which method it is currently executing instructions. The Management Console takes these snap shots continuously when the **Threads** tab is visible. If your application, for example, halts, you can use the stacktraces to find out where in your application the computer was executing code and which methods that was called before it stopped. The stacktrace can offer a hint of where to look in you application if you get a deadlock.

To display a stacktrace for a thread

- Highlight a thread on the **Live Threads** table.

  The **Stacktraces for selected threads** panel shows the stacktrace for the highlighted thread (Figure 1-5).

Figure 1-5 Stack traces for selected threads panel



Jumping to Application Source

If you are using the Management Console as an Eclipse plug-in, you can jump from the **Stacktraces for selected threads** panel directly to the source code. A feature called *Jump-to-Source* allows you not only to see the name of a "problem" class or method displayed in the stacktrace, but lets you jump from the displayed method or class name directly to that class or method's source, where you can evaluate the code to see what might be causing the problem. This feature extremely is useful in helping you locate and debug coding errors that are creating runtime problems for your application.

To jump to the source code from the threads stacktrace

1. In the stacktrace, right-click the problem method or class to open a context menu.
2. Select **Open Method** or **Open Type** (depending upon what you are jumping from).
3. The source code appears in a separate editor.

Threads Tab Functionality

The functions you can perform on this tab are:

- You can add and remove attributes.
- You can freeze the graph to better study it. Click the **Freeze scrolling** option to freeze the graph.

- You can filter columns in the **Live Thread** table.
- You can zoom-in to specific view on the graph, including a selected time period. You can also use the Zoom function to redefine the X-axis to show different time increments.
- You can redefine the Y-axis to better show a spread of data.
- You can rename the titles on the graphs to better suit the attributes you are monitoring.