Java SE 6 でのパフォーマンスの監視と診断

最新の Java リリースで強化されたパフォーマンスを生かし、増強された監視機能を活用する

Cathy Kegley
Software Engineer
IBM

Greg Roberts
Staff Software Engineer
IBM

2007年 9月 25日

パフォーマンスに重点を置いた Java SE (Java™ Platform, Standard Edition) バージョン 6 では、アプリケーションの管理、監視用ツール、そして一般的な問題を診断するためのツールが拡張されています。この記事では Java SE プラットフォームにおける監視および管理の基礎を概説するとともに、それに関連する Java SE 6 での機能強化について詳しく説明します。

徹底してパフォーマンスに重点を置いた Java SE 6 では、アプリケーションの管理、監視用ツール、そして一般的な問題を診断するためのツールが拡張されています。改善内容は以下のとおりです。

- 監視および管理 API の機能強化
- 改良されたグラフィカル監視ツール、JConsole の正式サポート
- Java 仮想マシン (JVM) の強化されたインスツルメンテーション

この記事では Java SE プラットフォームにおける監視および管理の基礎を概説するとともに、最新リリースでのパフォーマンスの監視および管理機能の強化について詳しく説明します。また、Java SE 6 プラットフォームで利用できる診断ツールとトラブルシューティング・ツールについて紹介します。

この記事を活用するには、以前の Java SE リリースで導入された監視および管理機能を十分に理解していることが必要です。背景となる詳細情報ついては、「参考文献」を参照してください。

監視および管理 API

Java SE 5 で導入された `java.lang.management` パッケージでは、プラットフォーム MBean あるいは MXBean とも呼ばれる MBean を 9 つ定義しています。それぞれの MXBean に、JVM の各機能領域がカプセル化されています。Java SE 5 以来、JVM にはプラットフォーム MBean サーバーという

組み込み MBean サーバーが含まれるようになっていますが、MBean はここに常駐し、このリポジトリーによって管理されます。表 1 に、Java プラットフォームでの 9 つの MXBean の概要を記載します。

表 1. プラットフォーム MBean

管理インターフェース	管理対象のリソース
ClassLoadingMXBean	クラス・ローダー
CompilationMXBean	コンパイラー
MemoryMXBean	メモリー
ThreadMXBean	スレッド
RuntimeMXBean	ランタイム
OperatingSystemMXBean	オペレーティング・システム
GarbageCollectorMXBean	ガーベッジ・コレクター
MemoryManagerMXBean	メモリー・マネージャー
MemoryPoolMXBean	メモリー・プール

JVM が提供するプラットフォーム MBean は、どのアプリケーションからでも取得して使用することができます。それには、必要な Bean のインスタンスを取得し、適切なメソッドを呼び出します。MXBean を使用すると、ローカル、リモートを問わず、JVM の振る舞いを監視でき、JVM に関する情報を取得することができます。

プラットフォーム MBean でアクセスできる情報は、ロードされたクラスの数、JVM の実行可能時間、メモリー使用量、実行中のスレッド数、そしてスレッド競合に関する統計などです。

JVM リソースの監視、管理を行うには、以下のいずれかの手段を使用することができます。

- MXBean インターフェースによる直接アクセス
- MBeanServer インターフェースによる間接アクセス

MXBean インターフェースによる直接アクセス

`MXBean` インスタンスは、ローカル側で実行している JVM の `MXBean` インターフェースに直接アクセスする静的ファクトリー・メソッドから取得することができます。`MXBean` を取得するための静的ファクトリー・メソッドを提供するのは `ManagementFactory` クラスです。リスト 1 に、このファクトリーを使って `RuntimeMXBean` を取得し、標準属性の 1 つ、`VmVendor` の値を取る例を示します。

リスト 1. MXBean への直接アクセス

```
RuntimeMXBean mxbean = ManagementFactory.getRuntimeMXBean();

// Get the standard attribute "VmVendor"
String vendor = mxbean.getVmVendor();
```

MBeanServer インターフェースによる間接アクセス

プラットフォーム `MBeanServer` インターフェースは `MXBeanServerConnection` を使ってリモート JVM に接続し、接続先のプラットフォームで実行中の MXBean にアクセスできるように

します。プラットフォーム MBean サーバーにアクセスするには、ManagementFactory クラスの getPlatformMBeanServer メソッドを使用します。リスト 2 は、リモート JVM で実行中の RuntimeMXBean を取得し、VmVendor 属性の値を取る例です。

リスト 2. MXBean への間接アクセス

```
MBeanServerConnection serverConn;

try {
   //connect to a remote VM using JMX RMI
   JMXServiceURL url = new JMXServiceURL( "service:jmx:rmi:///jndi/rmi://<addr>");

   JMXConnector jmxConnector = JMXConnectorFactory.connect(url);

   serverConn = jmxConnector.getMBeanServerConnection();

   ObjectName objName = new
   ObjectName(ManagementFactory.RUNTIME_MXBEAN_NAME);

   // Get standard attribute "VmVendor"
   String vendor =
   (String) serverConn.getAttribute(objName, "VmVendor");

} catch (...) { }
```

MXBean および java.lang.management API についての詳細は、「参考文献」を参照してください。

Java SE 6 での API 機能拡張

Java SE 5 で導入された java.util.concurrent.locks パッケージは、ロックおよび待機用のフレームワークを提供します。このフレームワークは Java の組み込み同期サポートとは明確に異なり、ロックを使用する際の柔軟性を大幅に向上させます。

Java SE 6 では、java.lang.management パッケージに java.util.concurrent.locks のサポートを追加しています。このサポートには、ロックに関する情報を提供する新規クラスの追加、そして ThreadInfo、ThreadMXBean および OperatingSystemMXBean インターフェースの機能拡張が含まれます。

Java SE 6 が導入する新規クラスは以下の 2 つです。

- LockInfo。ロックに関する情報が含まれます。
- MonitorInfo。LockInfo を継承しており、オブジェクト・モニターのロックに関する情報が含まれます。

ThreadInfo クラスが上記の新規オブジェクトを使用するために導入しているメソッドには、以下の 3 つがあります。

- getLockInfo() は、ブロック状態にある特定のスレッドが待機している LockInfo オブジェクトを返します。
- getLockedMonitors() は、特定のスレッドによって現在ロックされている MonitorInfo オブジェクトを返します。
- getLockedSynchronizers() は、特定のスレッドによって現在ロックされている所有可能なシンクロナイザーを表す LockInfo オブジェクトを返します。

Java SE 5 の `ThreadMXBean.getThreadInfo` メソッドがレポートするのは、スレッドによる取得待ちとなっているオブジェクト・モニターか、スレッドによる取得開始がブロックされているオブジェクト・モニターだけですが、Java SE 6 ではこれらのメソッドが強化され、スレッドで取得待ちとなっている `AbstractOwnableSynchronizer` もレポートするようになっています。

`ThreadMXBean` インターフェースには次の 4 つのメソッドが新たに追加されています。

- `isObjectMonitorUsageSupported()` は、仮想マシンがオブジェクト・モニターの使用量の監視をサポートするかどうかをテストします。
- `isSynchronizerUsageSupported()` は、仮想マシンが所有可能なシンクロナイザーの使用量の監視をサポートするかどうかをテストします。
- `findDeadlockedThreads()` は、デッドロック状態にあるスレッド ID の配列を返します。デッドロック状態のスレッドはオブジェクト・モニターまたはシンクロナイザーが開始しないように互いにブロックし合います。
- `dumpAllThreads()` は、すべての有効なスレッドのスタック・トレース情報および同期情報を返します。

`OperatingSystemMXBean` インターフェースは更新され、最後の 1 分間のシステム負荷の平均を返す `getSystemLoadAverage()` メソッドが組み込まれています。

このようなプログラム面でのサポートに加え、Java SE 6 には複数の診断ツールやトラブルシューティング・ツールも組み込まれています。これらのツールは、問題を検出したり、JVM リソースの使用量を監視するために使用することができます。続く 2 つのセクションでは、こうした診断ツールのいくつかについて、実例を用いて説明します。

Java 監視および管理コンソール (JConsole)

Java SE 6 は、Java SE 5 で導入された監視および管理コンソール、JConsole を正式にサポートします。実行時に各種 JVM リソースの統計を監視できる JConsole は、デッドロック、ロック競合、メモリー・リーク、そしてスレッドの循環といった状況を検出するのに特に役立ちます。このコンソールはローカル JVM、リモート JVM のどちらかに接続して、以下を監視することができます。

- スレッドの状態 (関連するロックも含む)
- メモリー使用量
- ガーベッジ・コレクション
- ランタイム情報
- JVM 情報

以降のサブセクションでは、Java SE 6 で強化された JConsole の機能について説明します。JConsole の起動および使用方法についての詳細は、「参考文献」を参照してください。

Attach API のサポート

JConsole は Java SE 6で初めて Attach API を実装するようになりました。この API は `com.sun.tools.attach` と `com.sun.tools.attach.spi` という 2 つのパッケージで構成されます。これらのパッケージでは、アプリケーションの実装をターゲットの仮想マシンに動的に接続し、その JVM 内でアプリケーションのエージェントを実行することが可能です。

以前は、JConsole で監視するアプリケーションを開始するには `-Dcom.sun.management.jmxremote` オプションを使用しなければなりませんでしたが、その必要はもうありません。動的接続のサポートにより、JConsole は Attach API をサポートするアプリケーションであればどれでも監視できるようになっています。Attach API に準拠するアプリケーションは JConsole の起動時に自動的に検出されます。

強化された UI および MBean の表示

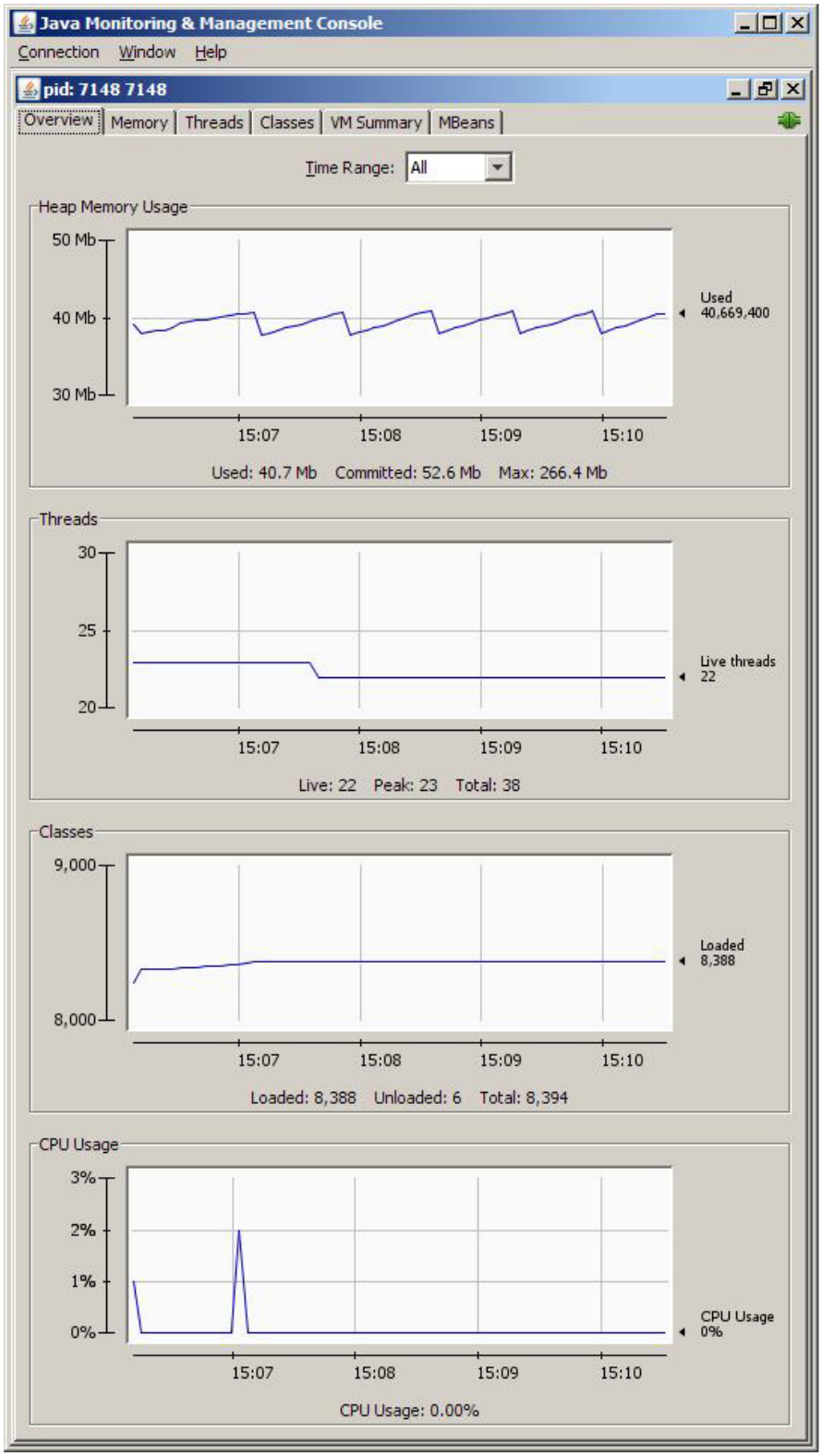
Java SE 6 では JConsole が更新され、どのプラットフォームで実行されているかによって Windows® オペレーティング・システムまたは GNOME デスクトップと同様のルック・アンド・フィールを持つようになっています。今後この記事に記載するスクリーン・ショットはすべて Windows XP からのもので、前のリリースから変更された UI 機能が示されています。

JConsole が起動されてアプリケーションと関連付けられると、そのビューには以下の 6 つのタブが表示されます。各タブでは、それぞれ異なる JVM リソースを 1 つ表示しているか、またはリソースをセットで表示しています。

- Overview
- Memory
- Threads
- Classes
- VM Summary
- MBeans

Overview タブは、メモリー使用量、スレッド、クラス、CPU 使用率の相関情報をグラフで表示します。Overview タブには一連の関連情報が 1 ページにまとめて表示されますが、以前は複数のタブを切り替えないと、これらの情報を表示できませんでした。図 1 は、サンプル・アプリケーションの Overview タブです。

図 1. JConsole の Overview タブ



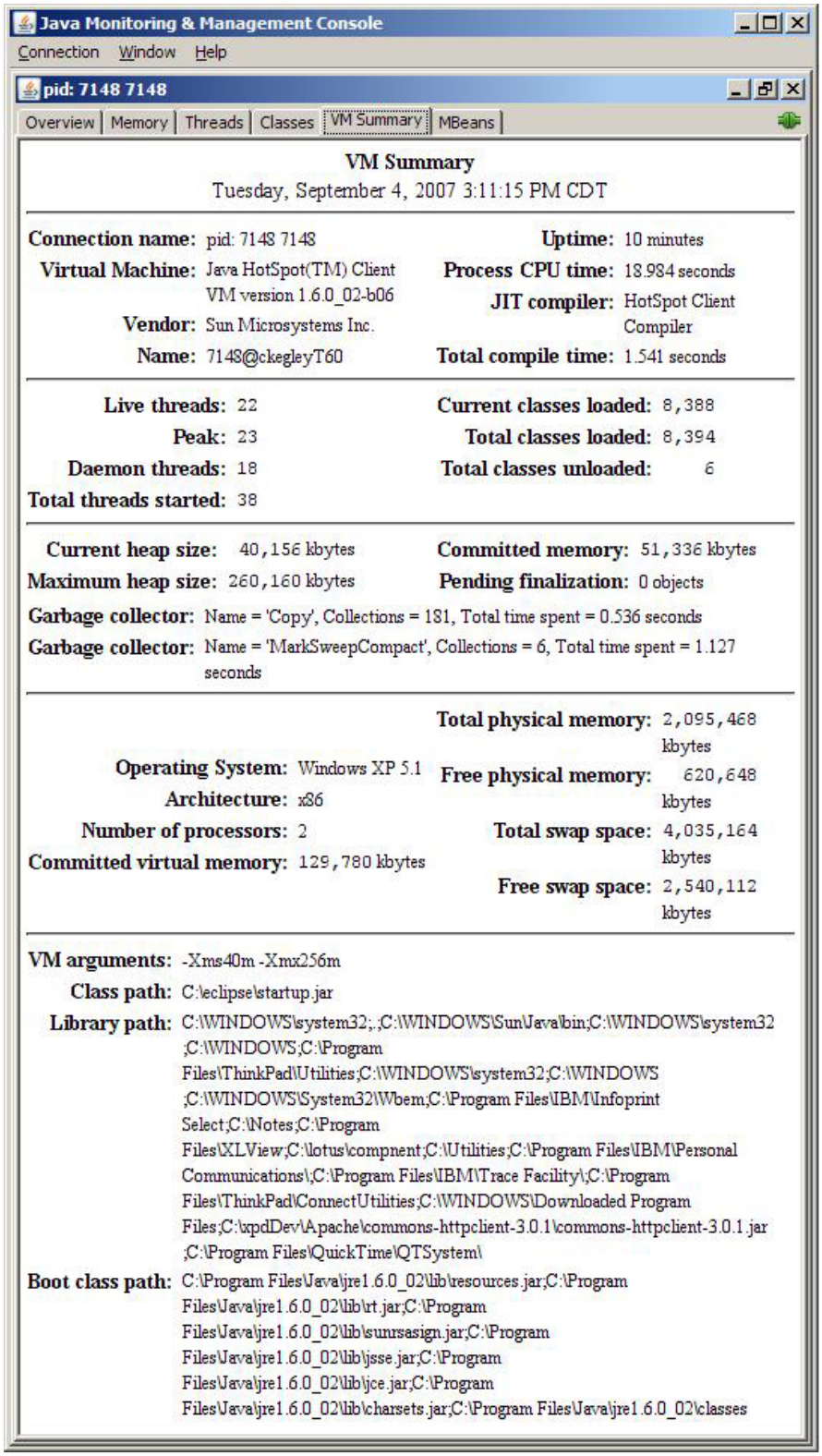
Overview タブには、VM リソース使用量の情報が 4 つのグラフで表示されます。このタブには選択リストもあり、このリストを使って、結果を表示させたい対象時間範囲を変更することもでき

ます。一番上の Heap Memory Usage グラフは、ヒープ・メモリー使用量の推移をメガバイト単位で示したものです。このグラフは、メモリー・リークを検出するのに役立ちます。アプリケーションでメモリー・リークが発生している場合、ヒープ・メモリー使用量は時間の経過とともに増え続けるからです。

Threads グラフは有効なスレッド数の推移を表し、Classes グラフはロードされたクラスの数を示しています。また、CPU Usage グラフは、アプリケーションがそのライフサイクルのさまざまな時点で使用した CPU のパーセンテージを示しています。

図 2 に示す VM Summary タブも、Java SE 6 リリースで新たに追加されたものです。このタブには、合計実行可能時間、スレッドの情報、ロードされたクラス、メモリー統計、ガーベッジ・コレクション、オペレーティング・システムの情報など、JVM に関する詳細情報が表示されます。

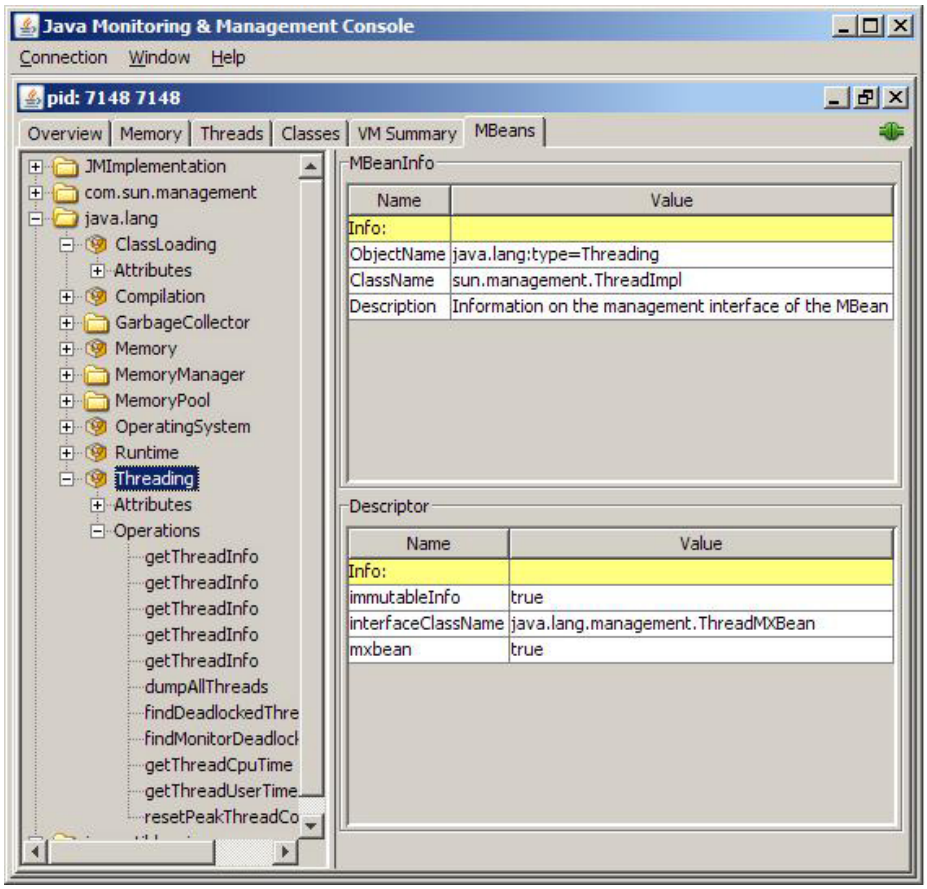
図 2. JConsole の VM Summary タブ



MBeans タブは改良され、MBean の操作と属性にアクセスしやすくなっています。このタブにはプラットフォームに登録されているすべての MBean に関する情報が表示されるので、すべて

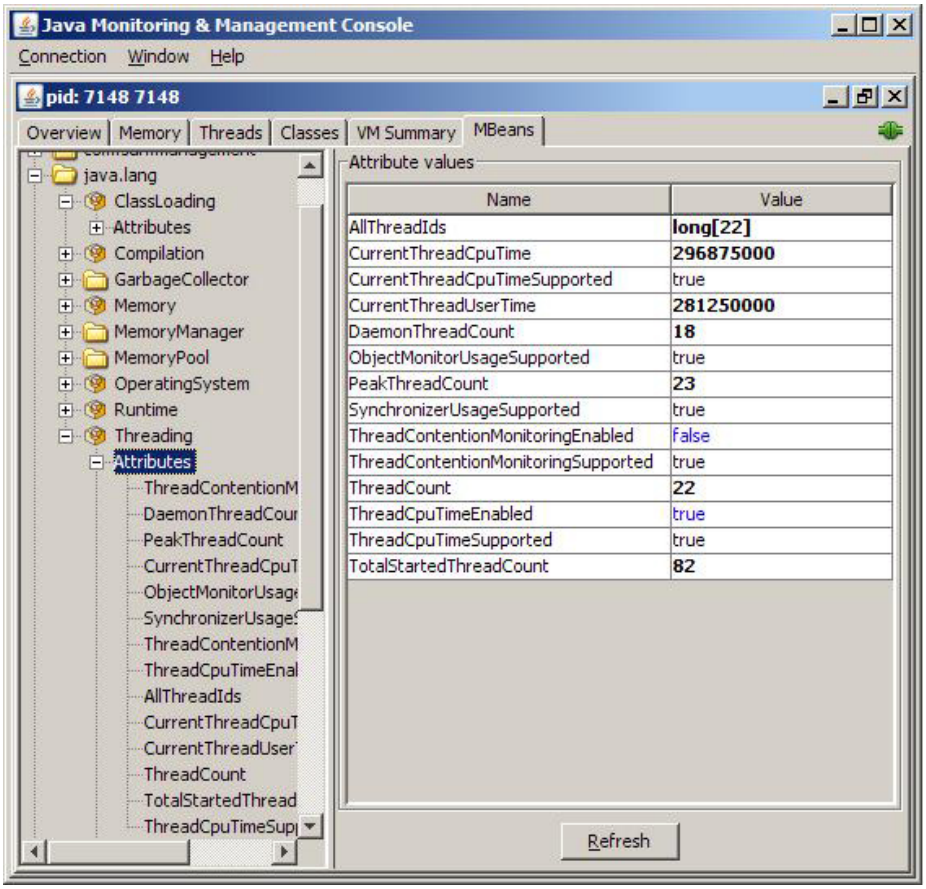
のプラットフォーム MBeanにはここからアクセスすることができます。左側にあるツリー構造には現在実行中のすべての MBean が表示されます。このツリーで MBean を選択すると、その MBeanInfo と記述子が右側の表にリストされます (図 3 を参照)。

図 3. JConsole の MBean タブ



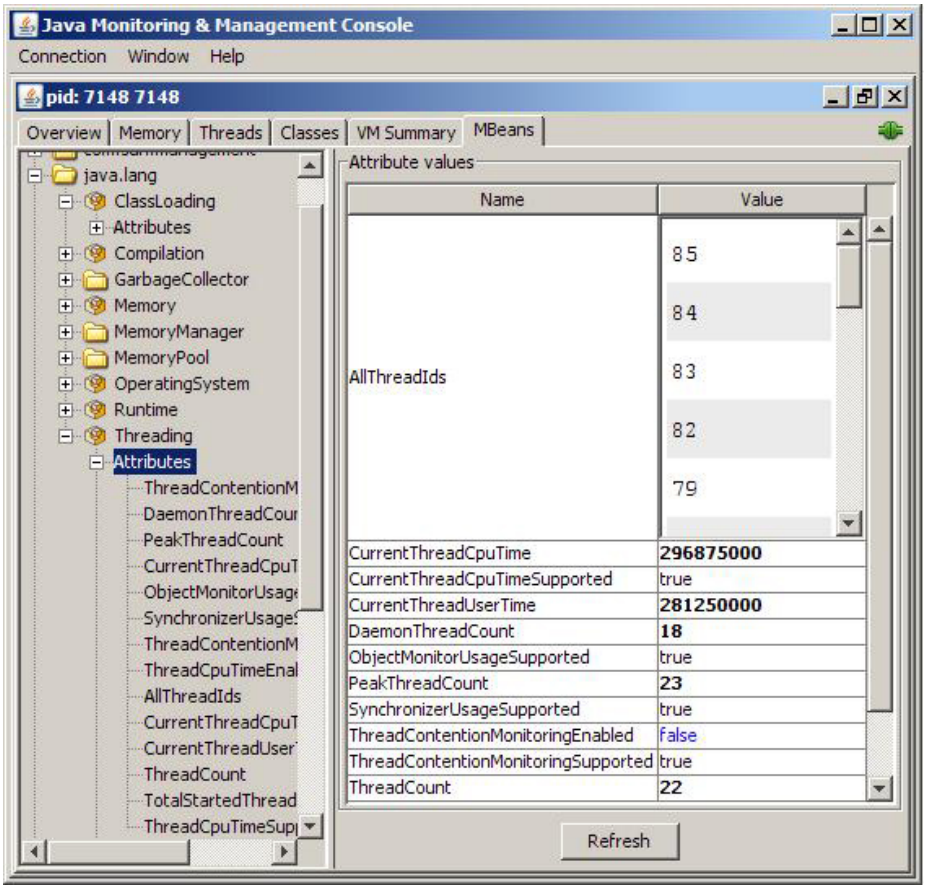
Attributes ノードを選択すると、Threading MBean のすべての属性が表示されます (図 4 を参照)。

図 4. MBean の属性



注目する点として、右側の表にリストされた属性とそれぞれの値は、前に説明した `java.lang.management` パッケージの `ThreadMXBean API` によって到達可能な属性値にマッピングされています。リストされた属性に関するさらに詳しい情報を表示するには、属性値をダブルクリックします。展開できる属性は、太字で記載されている属性のみです。例えば、`AllThreadIds` の値をダブルクリックすると、全部で 22 あるスレッドのスレッド ID が表示されます (図 5 を参照)。

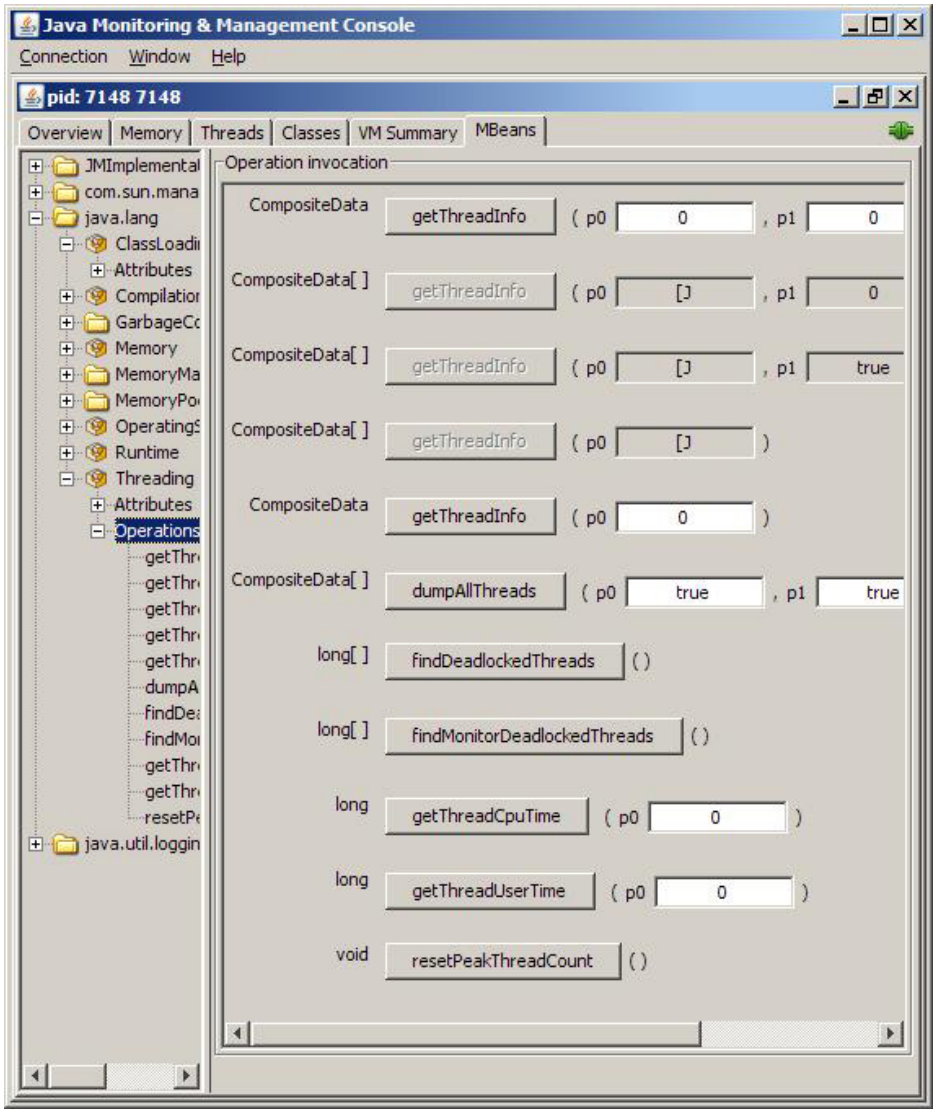
図 5. 展開した属性値



書き込み可能な属性は青字で示されます。これらの属性は、クリックして新しい値を入力することで編集することができます。図 5 の場合、このようにして JConsole ビューから編集できる属性は `ThreadContentionMonitoringAvailable` です。

左側のツリー構造で Operations ノードを選択すると、該当する MBean に関連付けられた操作が表示されます。右側にボタンとして表示された MBean の操作をクリックすると、その特定のメソッドが呼び出されます。図 6 には、ThreadMXBean に有効な操作が示されています。

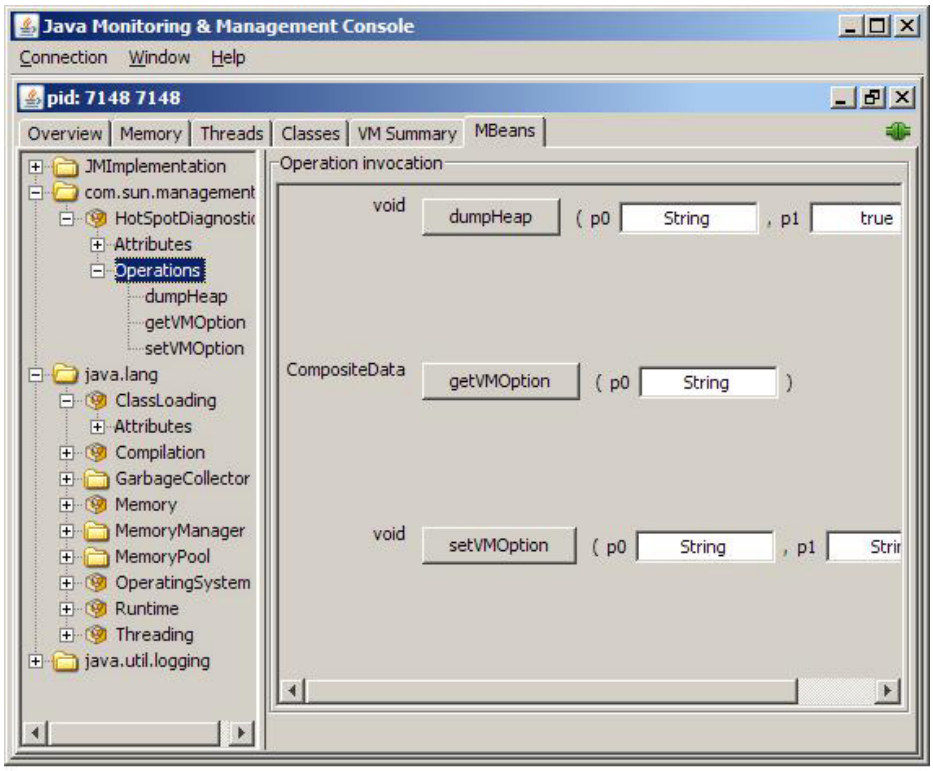
図 6. MBean の操作



HotSpot Diagnostic MBean

Java SE 6 の JConsole には、HotSpot Diagnostic MBean のサポートが備わっています。この MBean は、即時診断操作を実行できるようにするために、このリリースで導入されたものです。ユーザーはこの MBean の API を使用して、実行中にヒープ・ダンプを実行したり、その他の VM オプションを設定することができます。HotSpot Diagnostic MBean にアクセスするには、MBean タブで `com.sun.management node` ノードを展開して HotSpotDiagnostic を選択してください。`HotSpot Diagnostic MBean` から選択できるメソッドは、図 7 に示されています。

図 7. HotSpot Diagnostic MBean



JConsole のプラグイン・サポート

Java SE 6 から、JConsole にはプラグイン・サポートが組み込まれるようになっています。これにより、ユーザーが独自に作成したプラグインを JConsole で実行することが可能になりました。例えば、JConsole のメイン・ビューにアプリケーション固有の MBean にアクセスしたり、独自の監視アクティビティーを行うためのカスタム・タブを追加することができます。

カスタム JConsole プラグインを作成するには、`com.sun.tools.jconsole.JConsolePlugin` 抽象クラスを継承する必要があります。プラグインが JConsole ビューに正しく表示されるようにするためには、以下の 2 つのメソッドを実装します。

- `newSwingWorker()`。プラグインの GUI 更新を行う SwingWorker オブジェクトを返します。
- `getTabs()`。JConsole ウィンドウに追加するタブのマップを返します。

JConsole はそれ自体のサービス・プロバイダー・メカニズムを使用してすべてのプラグイン・クラスを検出し、ロードします。そのため、作成したプラグイン・クラスは、META-INF/services/com.sun.tools.jconsole.JConsolePlugin というファイルが含まれる JAR ファイルに含めなければなりません。このファイルには、行ごとにプラグイン・クラスの完全修飾名がリストされているはずです。新しいプラグインを JConsole ビューにロードするには、コマンドラインから以下のコマンドで JConsole を実行します。

```
jconsole -pluginpath plugin_path
```

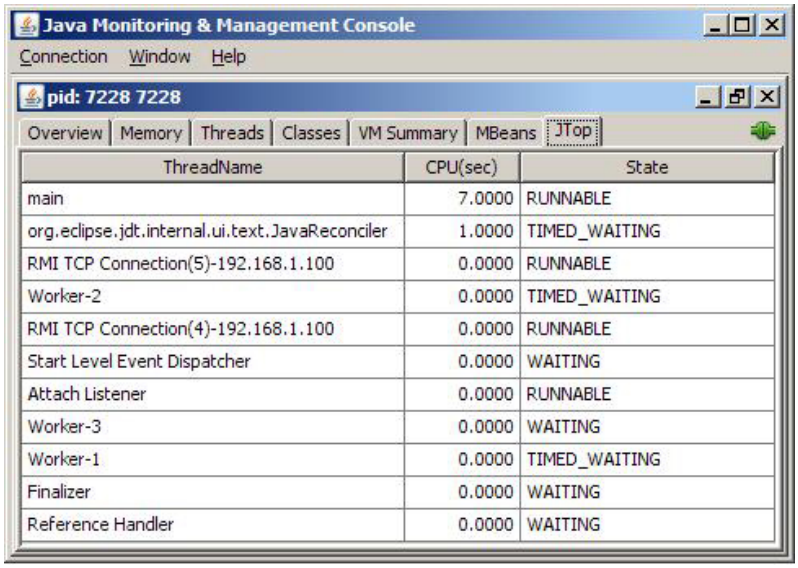
このコマンドの `plugin_path` には、JConsole プラグインのディレクトリーまたはアーカイブのパスを指定します。ここには複数のパスを指定することができます。

Java SE 6 には、JTop という名前のサンプル JConsole プラグインが付属しています。JTop は、現行アプリケーション内で実行中のスレッドの CPU 使用率を表示するプラグインです。JConsole と JTop を連動させるには、以下のコマンドを実行します。

```
jconsole -pluginpath JAVA_HOME/demo/management/JTop/JTop.jar
```

図 8 に、JTop タブが選択された場合の JConsole の例を示します。左側の列には実行中のすべてのスレッドの名前がリストされ、それぞれの CPU 使用率、そしてスレッドの状態が表示されます。このビューは、統計値が変わると自動的に更新されます。この JTop プラグインは、CPU 使用率が高いスレッドを識別するのに役立ちます。

図 8. JConsole の JTop プラグイン



監視およびトラブルシューティング用のツール

Java SE 6 は JConsole の他にも多数のコマンドライン・ツールをサポートします。これらの診断ツールは、あらゆるアプリケーションに接続できますが、そのアプリケーションを特殊なモードで起動する必要はありません。診断ツールでアプリケーションに関する詳細情報を入手することによって、そのアプリケーションが期待通りに振る舞っているかどうかを判断することができます。ただし、これらのツールは試用版としてリストされているので、今後の Java SE リリースでは完全にサポートされない可能性があることに注意してください。

監視ツール

Java SE 6 には表 2 にリストした 3 つのコマンドライン・ユーティリティーが組み込まれています。これらのユーティリティーは、JVM パフォーマンス統計の監視に役立ちます。

表 2. 監視ツール

ツール	説明
`jps`	JVM プロセス·ステータス・ツール
`jstat`	JVM 統計監視ツール

jstatd	JVM jstat デーモン

jps ユーティリティーは、ターゲット・システムの現行ユーザーの仮想マシンをリストします。このユーティリティーが活躍するのは、VM が標準 Java ランチャーではなく JNI Invocation API で起動される環境です。このような環境ではプロセス・リストで Java プロセスを把握するのが難しいことがありますが、jps ツールによってこの問題は軽減します。

以下に示してあるのは jps ユーティリティーの使用例です。このようにコマンドラインに jps と入力するだけで、ユーザーがアクセス権を持つ仮想マシンおよびプロセス ID がリストされます (リスト 3 を参照)。

リスト 3. jps ユーティリティーの使用例

```
$ jps
16217 MyApplication
16342 jps
```

jstat ユーティリティーは JVM の組み込みインスツルメンテーションを使用して、実行中アプリケーションのパフォーマンスとリソース使用量に関する情報を提供します。このツールはパフォーマンス上の問題、特にヒープのサイズ変更とガーベッジ・コレクションに関連した問題を診断する際に役立ちます。

jstatd デーモンは、JVM の作成と終了を監視する RMI (Remote Method Invocation) サーバー・アプリケーションで、ローカル・ホストで実行中の JVM にリモート監視ツールを接続するためのインターフェースとなります。例えば jps ユーティリティーはこのデーモンを使用して、リモート・システム上のプロセスをリストすることができます。

それぞれの監視ツールに関する資料と使用例については、「参考文献」を参照してください。

トラブルシューティング・ツール

Java SE 6 には表 3 にリストしたトラブルシューティング・ツールも組み込まれています。これらのツールによって、アプリケーションのなかで期待通りに動作していない部分を特定することができます。

表 3. トラブルシューティング・ツール

ツール	説明
jinfo	構成情報
jhat	ヒープ・ダンプ・ブラウザー
jmap	メモリー・マップ
jsadebugd	サービスアビリティー・エージェント・デバッグ・デーモン
jstack	スタック・トレース

jinfo は、実行中の Java プロセスやクラッシュ・ダンプから構成情報を取得し、仮想マシンの起動に使用されたシステム・プロパティーまたはコマンドライン・フラグを出力するコマンドライン・ユーティリティーです。

`jhat` ツールはヒープ・スナップショットでオブジェクト・トポロジーを閲覧するのに重宝な手段となります。Java SE 6 リリースで HAT (Heap Analysis Tool) に代わって導入されたこのツールは、メモリー・リークの検出に役立ちます。

`jmap` は、実行中の VM またはコア・ファイルにメモリー関連の統計を出力するコマンドライン・ユーティリティーです。このユーティリティーでも `jsadebugd` デーモンを使ってリモート・マシンのプロセスやコア・ファイルを照会することができます。`jmap` ツールは、`OutOfMemoryError` の原因となり得るファイナライザーの過剰な使用を診断する上で役立ちます。

サービスアビリティー・エージェント・デバッグ・デーモン (`jsadebugd`) は Java プロセスまたはコア・ファイルに接続し、デバッグ・サーバーとして機能します。このユーティリティーは現在、Solaris OS と Linux® でしか利用できません。`jstack`、`jmap`、`jinfo` などのリモート・クライアントがこのサーバーに接続するには、Java RMI を使用する必要があります。

`jstack` コマンドライン・ユーティリティーは指定されたプロセスやコア・ファイルに接続し、Java スレッドや VM 内部スレッドをはじめとする、仮想マシンすべてのスレッドのスタック・トレースを出力し、またオプションでネイティブ・スタック・フレームを出力します。このユーティリティーではデッドロックを検出できるだけでなく、`jsadebugd` デーモンを使ってリモート・マシンのプロセスやコア・ファイルを照会することも可能です。`jstack` はデッドロックの診断に役立つツールです。

各ツールに関する詳細の資料と使用例については、「参考文献」を参照してください。

まとめ

VM インスツルメンテーション、管理 API、そして JDK ツールに機能強化をもたらす Java 6 プラットフォームは、Java アプリケーションでのパフォーマンスとメモリーの問題を効率的に特定し、診断できるようにします。この記事では Java SE の監視および管理フレームワークに加えられた改善内容を説明するとともに、開発者に用意された診断コマンドライン・ユーティリティーを紹介しました。

Java アプリケーションの平均実行速度は着実に改善されていますが、Java SE 6 のリリースによって Java のパフォーマンスは C や C++ のパフォーマンスに匹敵するまでに至っています。多くの場合に Java コードが大幅に高速化されているだけでなく、この記事で説明したツールを使用してパフォーマンスをさらに最適化することができます。まずは試しにツールを使ってみてください。今までまったく気付かなかったアプリケーション最適化の可能性が見つかるはずです。

著者について

Cathy Kegley

Cathy Kegley は、IBM Lotus Expeditor クライアント・チームのソフトウェア・エンジニアです。

Greg Roberts

Greg Roberts は、IBM Lotus Expeditor クライアント開発チームのスタッフ・ソフトウェア・エンジニアです。