今まで知らなかった 5 つの事項: Java のパフォーマンス・モニタリング、第 1 回

JConsole と VisualVM を使って Java のパフォーマンスをプロファイリングする

Ted Neward
Principal
Neward & Associates

Alex Theedom
Senior Java developer
Consultant

2017年 8月 31日
(初版 2010年 6月 29日)

質の悪いコード (または質の悪いコード作成者) を非難しても、パフォーマンスのボトルネックを見つけたり Java アプリケーションの処理速度を改善したりするためには役立ちません。同様に、推測も役に立ちません。そこで、Ted Neward が皆さんの関心を Java パフォーマンス・モニタリング用ツールに向けるために、まずは JConsole という Java に付属のプロファイラーを使用する際の 5 つのヒントを紹介します。

このシリーズの他の記事を見る

アプリケーションのパフォーマンスに問題がある場合、ほとんどの開発者はパニックに陥りますが、それには十分な理由があります。Java アプリケーションのボトルネックの原因追跡は、昔から非常に困難でした。これは Java 仮想マシンがブラックボックスのようなものであるためであり、また従来は Java プラットフォーム用のプロファイリング・ツールにあまり良いものがなかったためでもあります。

けれどもそのような状況は、JConsole の導入によって一変しました。JConsole は組み込みの Java パフォーマンス・プロファイラーであり、コマンドラインや GUI シェルから実行することができます。JConsole は完璧ではありませんが、知ったかぶりをする上司がパフォーマンスの問題で皆さんのところに来た時に必要最低限の情報を提供する以上の機能を備えており、おなじみの Google に頼るよりも、はるかに優れています。

この連載、「今まで知らなかった 5 つの事項」の今回の記事では、JConsole (または、JConsole の仲間であり、JConsole よりも外観が洗練された VisualVM) を使ってJava アプリケーションのパ

フォーマンスをモニタリングし、Java コードのボトルネックを追跡するための 5 つの簡単な方法について説明します。

1. JDK に付属のプロファイラー

この連載について

皆さんは自分が Java プログラミングについて知っていると思うかもしれません。しかし実際には、ほとんどの開発者は Java プラットフォームの表面的な部分しか扱っておらず、当面の作業を完了するために十分なことしか学んでいません。この連載では、Ted Neward が Java プラットフォームのコア機能を深く掘り下げ、非常に厄介な Java プログラミングの難題の解決にも役立つ、ほとんど知られていない事実を紹介します。

JConsole (または、より最新の Java プラットフォームのリリースである VisualVM) は、組み込みのプロファイラーであり、Java コンパイラー同様、容易に起動することができます。JConsole を起動するには、`PATH` 上に JDK がある状態で、コマンド・プロンプトから `jconsole` と入力して実行するだけです。GUI シェルの場合には、JDK をインストールしたディレクトリーまでナビゲートし、bin フォルダーを開き、jconsole をダブルクリックします。

プロファイラー・ツールがポップアップ表示されると (その瞬間にどのバージョンの Java が実行中か、また他の Java プログラムがいくつ実行中かに応じて) 接続先のプロセスの URL を要求するダイアログ・ボックスが表示されるか、あるいは接続先となるローカル Java プロセスがいくつか一覧表示されます (そうした一覧の中に JConsole プロセス自体が含まれる場合もあります)。

JConsole を扱う

Java プロセスは、デフォルトで、プロファイリングされるようにセットアップされます。したがって、プロセスの起動時にコマンドライン引数 (`Dcom.sun.management.jmxremote`) を渡す必要はありません。アプリケーションを起動するだけで、自動的にモニタリングを行える状態になります。ある 1 つのプロセスが JConsole によって選択されると、そのプロセスをダブルクリックするだけで、そのプロセスのプロファイリングを開始することができます。

プロファイラーには、プロファイラー自体のオーバーヘッドがあります。そのため、少し時間を取って、そうしたオーバーヘッドが何によって生じているのかを明らかにしておくのは賢明なことです。JConsole のオーバーヘッドを明らかにする最も簡単な方法は、最初にアプリケーションをそれ単体で実行し、次にそのアプリケーションをプロファイラーの下で実行し、その違いを測定する方法です。(このとき、そのアプリケーションの規模は大きすぎても小さすぎてもいけません。私がよく使用しているのは、JDK に付属している SwingSet2 デモ・アプリケーションです。) 例えば、私は SwingSet2 を `-verbose:gc` オプションを付けて実行してガーベッジ・コレクションのスイープを調べ、次に JConsole プロファイラーを接続して同じ SwingSet2 を実行しました。JConsole が接続されると、ガーベッジ・コレクションのスイープが連続的に実行されました。このスイープは JConsole が接続されていない場合には実行されなかったものです。このスイープがプロファイラーのパフォーマンスのオーバーヘッドです。

2. リモートでプロセスに接続する

Web アプリケーションのプロファイラーでは、ソケットを介した接続でプロファイリングを行うことを想定しています。そのため、少し構成を行うだけで、リモートで実行されるアプリケー

ションのモニタリングやプロファイリングを行えるように、JConsole (または JVMTI ベースの任意のプロファイラー) をセットアップすることができます。

例えば、「webserver」という名前のマシンで Tomcat が実行されており、その JVM では JMX が有効でポート 9004 をリッスンしているとすると、そのTomcat に JConsole (または他の任意の JMX クライアント) から接続するためには、JMX の URL は「service:jmx:rmi:///jndi/rmi://webserver:9004/jmxrmi」でなければなりません。

要するに、リモートのデータ・センターで実行中のアプリケーション・サーバーをプロファイリングするために必要なものは、JMX の URL のみです。(JMX と JConsole を使ってリモートのモニタリングや管理を行うための詳細については「関連トピック」を参照してください。)

3. 統計を追跡する

典型的な手段に頼らない

アプリケーション・コードにおけるパフォーマンスの問題を見つけた場合の開発者の対応はさまざまですが、開発者のタイプによってその反応を予測することもできます。Java の初期の頃からプログラミングを行ってきた開発者であれば、古めかしい IDE を起動してコード・ベースの主要部分のコード・レビューを開始し、ソースの中にある、おなじみの「要注意部分」(同期ブロック、オブジェクトの割り当て、等々) を探すでしょう。そうした人達よりもプログラミング経験が少ない開発者であれば、JVM でサポートされている -X フラグを詳しく調べ、ガーベッジ・コレクターを最適化する方法を探すでしょう。そしてもちろん、初心者であれば、まず Google にアクセスし、コードを作成し直さなくてすむような、JVM を高速化する魔法のスイッチを誰か他の人が見つけているのでは、と望むものです。

そうした方法はどれも、決して本質的に問題があるわけではありませんが、場当たり的です。パフォーマンスの問題に対して最も効果的に対応するためには、プロファイラーを使うことです。プロファイラーが Java プラットフォームに組み込まれた今、プロファイラーを使わずに済ませられるような言い訳はまったくありません。

JConsole には統計の収集に便利なタブがいくつかあります。例えば以下のようなタブがあります。

- Memory (メモリー): JVM のガーベッジ・コレクターのさまざまなヒープに対するアクティビティーを追跡することができます。
- Threads (スレッド): ターゲットとなる JVM の現在のスレッドのアクティビティーを検証することができます。
- Classes (クラス): VM にロードされたクラスの総数を監視することができます。

これらのタグ (および関連するグラフ) はいずれも、JMX オブジェクトによってサポートされます。すべての Java VM はこれらの JMX オブジェクトを JMX サーバーに登録して、その JMX サーバーを JVM に組み込むという仕組みになっています。指定された JVM の中で利用可能な Bean の完全なリストは、一部のメタデータと共に「MBeans」タブに表示されます。「MBeans」タブには、そうしたデータを見たり操作を実行したりするための限定的なユーザー・インターフェースも用意されています。(ただし JConsole のユーザー・インターフェースに登録して通知を行うことはできません。)

統計を使う

例えば、Tomcat プロセスが `OutOfMemoryError` によって頻繁に動作が停止するとします。何が起きているのかを知るためには、JConsole を開いて Classes タブをクリックし、時間と共にクラスの

カウント数がどう変化するかを観察します。カウントが上昇を続けているようであれば、アプリケーション・サーバーまたは皆さんのコードのどこかに `ClassLoader` のリークがあり、しばらくすると `PermGen` スペースが足りなくなる、と考えることができます。この問題をもっと詳細に確認する必要がある場合には、「Memory (メモリー)」タブを調べます。

4. ヒープ・ダンプを作成してオフラインで分析する

本番環境では、さまざまなものが瞬く間に変化するため、アプリケーション・プロファイラーを使って調べるだけの有効な時間が取れないかもしれません。プロファイラーを使う代わりに、Java 環境すべてのスナップショットを取って保存し、後で調べる方法があります。この方法は、JConsole で実行することができ、さらには VisualVM を使えばもっとうまく実行することができます。

まず、「MBeans」タブにナビゲートして `com.sun.management` ノードを開き、次に `HotSpotDiagnostic` ノードを開きます。そして「`Operations` (操作)」を選択し、右側のペインに表示される「dumpHeap」ボタンに注目します。ダンプ先のファイル名を最初の (文字列を入れる) 入力ボックスに入れて dumpHeap に渡すと、dumpHeap は JVM ヒープ全体のスナップショットを取り、そのファイルにダンプ出力します。

後から、商用の多種多様なプロファイラーを使ってそのファイルを分析することもでき、あるいは VisualVM を使ってスナップショットを分析することもできます。(VisualVM はスタンドアロンのダウンロード・ファイルとして入手できます)。

5. JConsole は難しくはありません

JConsole は便利なプロファイラー・ユーティリティーですが、他にももっと便利なプロファイラー・ツールがあります。一部のプロファイラーには分析用のアドオンや優れたユーザー・インターフェースが用意されており、また一部のプロファイラーはデフォルトで JConsole よりも多くのデータを追跡することができます。

JConsole の本当の素晴らしさは、このプログラム全体が「昔ながらの単純な Java」で作成されている点、つまりどんな Java 開発者でも JConsole のようなユーティリティーを作成できる点にあります。実際 JDK には、JConsole をカスタマイズする方法の例として、JConsole 用の新しいプラグインを作成する方法まで含まれています (「**関連トピック**」を参照)。NetBeans の上に構築されている VisualVM は、そうしたプラグインの概念をさらに推し進めています。

JConsole (あるいは VisualVM、その他のツール) では望みどおりのことが行えない場合や、追跡したいものを追跡できない場合、あるいは望みどおりの方法で追跡できない場合には、独自のツールを作成することができます。また Java コードでは複雑すぎると思える場合には、Groovy や JRuby、あるいは 10 種類ぐらいある他の JVM 言語を使用することで、より迅速にツールを作成することができます。

皆さんにとって本当に必要なものは、JMX を介して接続され、関心対象のデータのみを思いどおりの方法で追跡できる、素早く手軽に作成できるコマンドライン・ツールなのです。

まとめ

Java のパフォーマンス・モニタリングは JConsole や VisualVM のみで終わるわけではありません。JDK には、ほとんどの開発者が知らない多種多様なツールが隠されています。この連載の次回の記事では、皆さんに必要なパフォーマンス・データを得るために役立つ、実験的なコマンドライン・ツールについて説明します。これらのツールは通常は特定のデータに焦点を絞っているため、完全なプロファイラーよりも小さくて軽量であり、そのため完全なプロファイラーと同じパフォーマンスのオーバーヘッドを生ずることがありません。

著者について

Ted Neward

Ted Neward has written over 250 articles and a dozen books across many different technologies, including .NET, iOS, Java, Android, and JavaScript. He resides in Seattle with his wife, two kids, nine laptops, fourteen mobile devices, and two cats. Email him if you're interested in having him or his company work with you.

Alex Theedom

May 2017