Javaの理論と実践: JMX を使ったアプリのインスツルメンテーション

Bean を追加するだけで簡単に表示を可能にする

Brian Goetz
Principal Consultant
Quiotix

2006年 9月 19日

デバッガーやプロファイラーはアプリケーションの動作の実態を教えてくれますが、私たちがこのようなツールを持ち出してくるのは通常、重大な問題が発生したときだけです。モニター・フックをアプリケーションに組み込めば、デバッガーを使用しなくてもプログラムが何を行っているか簡単に理解できるようになります。JMX (Java Management Extensions) が Java™ SE プラットフォームに統合され、jconsole ビューアーが汎用モニター GUI を提供するようになった今、これまで以上に簡単、効率的に JMX でアプリケーションにウィンドウを組み込めます。

このシリーズの他の記事を見る

実行中のアプリケーションを見て、「一体、何にこんなに時間がかかっているのか」と疑問に思うことがよくありませんか。そんなときは、モニター機能をもっとアプリケーションに組み込んでおけばよかったと後悔するものです。例えばサーバー・アプリケーションの場合、キューで処理を待っているタスクの数とタイプ、現在実行中のタスク、過去 1 分間または 1 時間のスループット統計、タスクの平均処理時間などが表示されると助かります。このような統計は簡単に収集できますが、必要なときにだけ簡単にデータを取り出す手段がなければ、あまり役には立ちません。

運用データをエクスポートする方法はいくつもあります。定期的に統計スナップショットをログ・ファイルに書き込む、Swing GUI を作成する、組み込み HTTP サーバーを使って Web ページに統計を表示する、あるいはアプリケーションのステータス照会に使用できる Web サービスを公開するなど、方法はさまざまです。ですが、データをモニターして公開するインフラストラクチャーがない限り、そこまでする開発者はほとんどいません。その結果、アプリケーションの動作状況の表示が必要なレベルにまで達していないという事態に至ります。

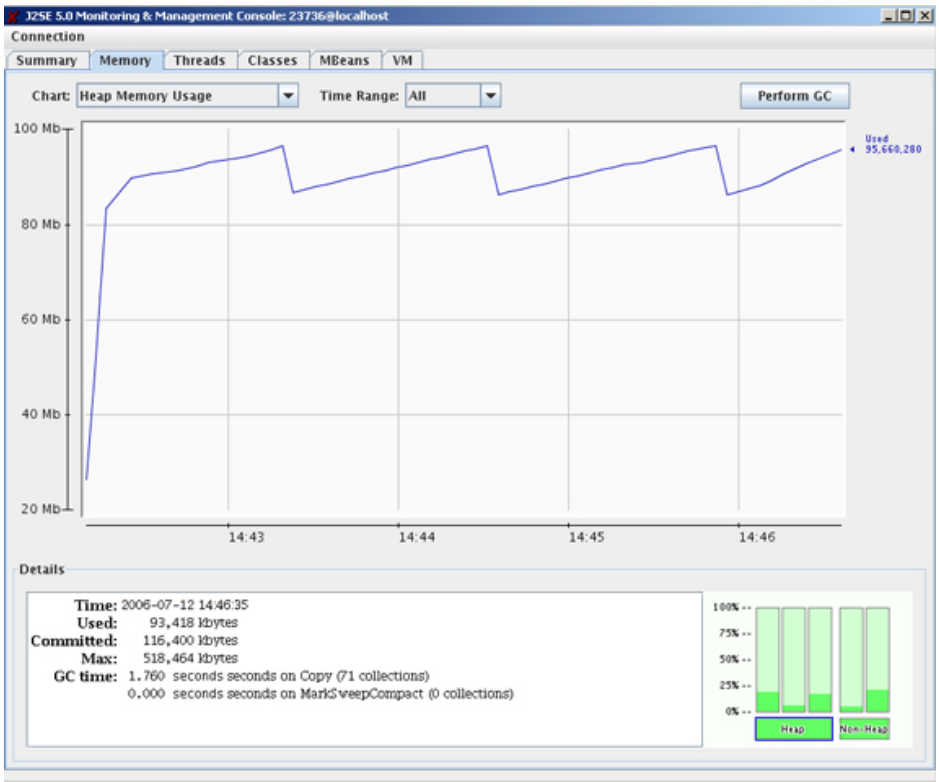
JMX

Java 5.0 のクラス・ライブラリーと JVM は管理およびモニターの包括的インフラストラクチャー、JMX を提供しています。JMX は、リモート・アクセスが可能な管理インターフェースを

提供するために標準化された手段で、柔軟かつ強力な管理インターフェースをアプリケーションに簡単に追加する方法となります。MBean (管理対象 Bean) と呼ばれる JMX コンポーネントは、エンティティーの管理に関するアクセサーとビジネス・メソッドを提供する JavaBean です。管理対象エンティティー (アプリケーション全体、またはアプリケーション内のサービス) のそれぞれが MBean をインスタンス化し、これを人間が読める名前で登録します。JMX 対応アプリケーションは、MBean のコンテナーとして機能する MBeanServer に依存して、リモート・アクセス、名前空間の管理、そしてセキュリティー・サービスを行います。一方クライアント側では、jconsole ツールが汎用 JMX クライアントとして機能します。これらを元にした JMX のプラットフォーム・サポートは、外部管理インターフェースをサポートするために必要なアプリケーションの作業を、大幅に軽減してくれます。

Java SE 5.0 は MBeanServer 実装を提供するだけでなく、JVM のインスツルメンテーションを行ってメモリー管理、クラス・ロード、アクティブ・スレッド、ロギング、そしてプラットフォーム構成の状態を簡単に表示できるようにします。ほとんどのプラットフォーム・サービスに対するモニターと管理はデフォルトでオンにされるため (パフォーマンスへの影響は最小限です)、あとは JMX クライアントでアプリケーションに接続するだけのことです。図 1 に示す jconsole JMX クライアント (JDK の一部) は、メモリー管理ビューの一つ、時間の経過によるヒープ使用量を表示しています。Perform GC ボタンは、JMX には動作統計の表示機能に加え、操作を開始する機能もあることを示しています。

図 1. jconsole を使用したヒープ使用量の表示



トランスポートとセキュリティー

JMX が MBeanServer と JMX クライアント間の通信に指定しているプロトコルは、各種のトランスポートで実行できます。組み込みトランスポートにはローカル接続、RMI、SSL 用があ

り、JMX Connector API を使用して新しいトランスポートを作成することも可能です。トランスポートでは認証が行われます。ローカル・トランスポートでは、同じユーザー ID で稼動中のローカル・システム上の JVM に接続できる一方、リモート・トランスポートではパスワードと証明書による認証が可能です。ローカル・トランスポートは Java 6 ではデフォルトで有効に設定されますが、これを Java 5.0 で有効にするには、JVM の起動時にシステム・プロパティー com.sun.management.jmxremote を定義する必要があります。トランスポートを有効にして構成する手順は、資料「Monitoring and Management using JMX」(「参考文献」を参照) に記載されています。

Web サーバーのインスツルメンテーション

JMX を使用するようにアプリケーションのインスツルメンテーションを行うのは簡単です。その他多くのリモート呼び出しフレームワーク (RMI、EJB、および JAX-RPC) と同様、JMX もインターフェースをベースにしています。管理対象サービスを作成するには、管理メソッドを指定する MBean インターフェースを作成します。次に、このインターフェースを実装してインスタンス化し、MBeanServer に登録する MBean を作成します。

リスト 1 に示す MBean インターフェースは、Web サーバーなどのネットワーク・サービス用のインターフェースです。このインターフェースは、構成情報 (ポート番号など) と動作情報 (サービスが起動しているかどうかなど) を取得するゲッターを提供します。また、現行のロギング・レベルなどの構成可能パラメーターを表示して変更するためのゲッターとセッター、そして start() および stop() などの管理操作を呼び出すメソッドも含まれています。

リスト 1. Web サーバー用 MBean インターフェース

```
public interface WebServerMBean {
    public int getPort();

    public String getLogLevel();
    public void setLogLevel(String level);

    public boolean isStarted();
    public void stop();
    public void start();
}
```

MBean インターフェースは既存のエンティティーやサービスのプロパティーおよび管理操作を反映することになっているので、大抵の場合、MBean クラスの実装は簡単明瞭です。例えば、MBean の getLogLevel() と setLogLevel() メソッドは単純に、Web サーバーで使用している Logger の getLevel() と setLevel() メソッドに転送されます。JMX には命名上の制約がいくつかあり、例えば MBean インターフェース名は MBean で終わり、FooMBean の MBean クラスは Foo という名前でなければなりません (この制約は、拡張 JMX 機能である動的 MBean を使用して取り除くことができます)。MBean をデフォルトの MBeanServer に登録するのも同じく簡単です。リスト 2 を参照してください。

リスト 2. 組み込み JMX 実装への MBean の登録

```
public class WebServer implements WebServerMBean { ... }

  ...

  WebServer ws = new WebServer(...);
  MBeanServer server = ManagementFactory.getPlatformMBeanServer();
  server.registerMBean(ws, new ObjectName("myapp:type=webserver,name=Port 8080"));
```

管理対象エンティティーを識別するのは、registerMBean() に渡される ObjectName です。指定されたアプリケーションには多くの管理対象エンティティーがあることが予想されるため、名前にはドメイン内の管理対象リソースを識別するドメイン (リスト 2 の「myapp」) とキーと値のペア数が含まれます。一般的に使用されるのは「name」と「type」のキーで、この 2 つが存在する名前は、ドメイン内に含まれる該当タイプのすべての MBean のなかから管理対象エンティティーを一意に識別します。別のキーと値のペアを指定することも可能で、JMX API にはオブジェクト名をワイルドカードでマッチングするための機能も組み込まれています。

MBean を作成して登録した直後から、アプリケーションで jconsole を指定すると (コマンド・ラインでjconsole を入力)、「MBeans」ビューに管理属性と操作が表示されるようになります。図 2 に新規 MBean の jconsole に表示される Attributes タブを、図 3 に Operations タブを示します。JMX はリフレクションを使用して読み取り専用のプロパティー (Started、Port) と読み取り書き込みプロパティー (LogLevel) を認識します。読み取り書き込みプロパティーについては、jconsole で変更できます。読み取り書き込みプロパティーのセッターが例外 (IllegalArgumentException など) をスローした場合は、JMX がクライアントに例外を報告します。

図 2. MBean の jconsole に含まれる Attributes タブ

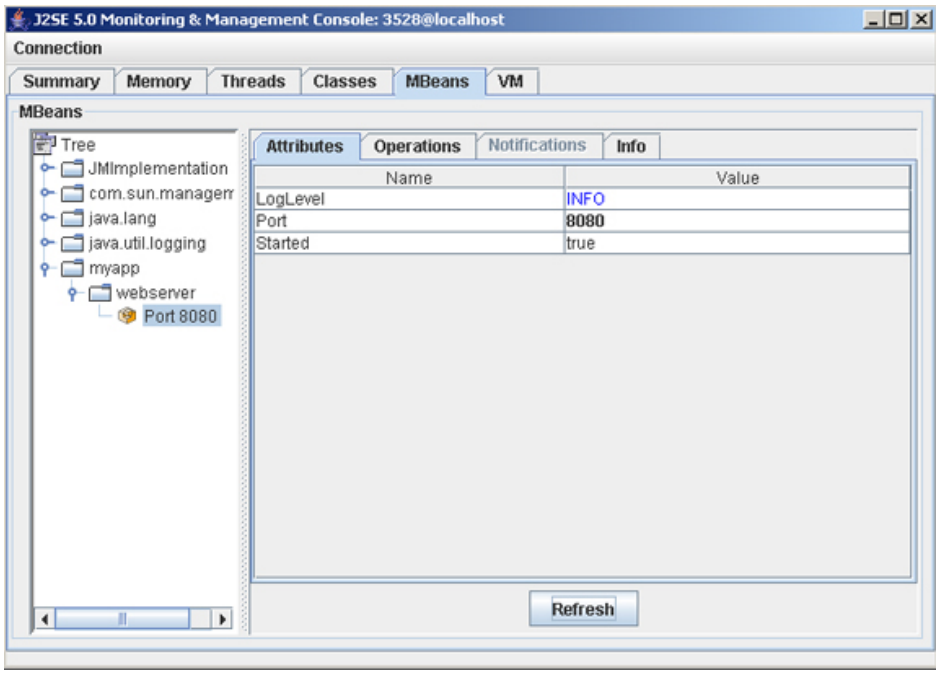
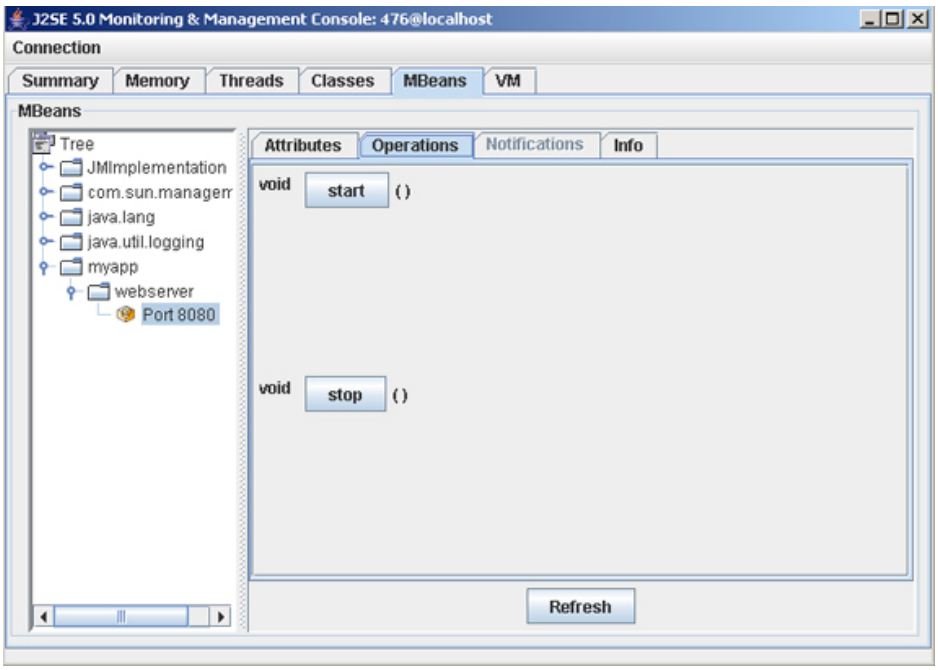


図 3. MBean の jconsole に含まれる Operations タブ



データ型

MBeans のアクセサーと操作はシグニチャーの中であらゆるプリミティブ型ならびに String、Date、およびその他の標準ライブラリー・クラスを使用できます。また、許容される型の配列や集合を使用することもできます。MBean メソッドがこれ以外のシリアライズ可能なデータ型を使用することもありますが、この場合、相互運用性の問題が持ち上がることがあります。クラス・ファイルは JMX クライアントでも使用できるようにしなければならないためです (RMI トランスポートを使用している場合は、RMI の自動クラス・ダウンロード機能を利用して、必要なクラスをクライアントに配送することができます)。管理インターフェースで構造化データ型を使用する場合、クラスの可用性に関する相互運用性の問題を回避するには、JMX の Open MBeans 機能を使用して複合データまたは表データを表してください。

サーバー・アプリケーションのインスツルメンテーション

管理インターフェースを作成する際には、特定のパラメーターと操作が明らかにインターフェースに組み込む候補となります。例えば、構成パラメーター、操作統計、デバッグ操作 (ロギング・レベルの変更、アプリケーション状態のファイルへのダンプなど)、そしてライフ・サイクル操作 (起動、停止) などです。 これらの属性と操作へのアクセスをサポートするようにアプリケーションを改良するのは極めて簡単ですが、JMX を最大限に利用するには、設計段階で実行時にどのような種類のデータがユーザーとオペレーターに役立つかを検討することをお勧めします。

JMX を使用してサーバー・アプリケーションが実行中の内容を把握するには、作業単位を識別して追跡する手段が必要です。タスクの記述方法として、標準インターフェース Runnable および Callable を使用して (toString() メソッドの実装などにより) タスクのクラスを自己記述的にすると、これらのタスクをそれぞれのライフ・サイクルに従って追跡し、MBean メソッドで待機中、実行中、そして完了したタスクのリストを戻すことができます。

リスト 3 の TrackingThreadPool は、実行中のタスクならびに完了したタスクの時間統計を追跡する ThreadPoolExecutor のサブクラスを示しています。これらのタスクは、beforeExecute() および afterExecute() フックをオーバーライドし、収集したデータを取り出すゲッターを指定することによって達成されます。

リスト 3. 実行中のタスクとタスクの平均所要時間の統計を収集するスレッド・プール・クラス

```
public class TrackingThreadPool extends ThreadPoolExecutor {
    private final Map<Runnable, Boolean> inProgress
        = new ConcurrentHashMap<Runnable,Boolean>();
    private final ThreadLocal<Long> startTime = new ThreadLocal<Long>();
    private long totalTime;
    private int totalTasks;

    public TrackingThreadPool(int corePoolSize, int maximumPoolSize, long keepAliveTime,
       TimeUnit unit, BlockingQueue<Runnable> workQueue) {
        super(corePoolSize, maximumPoolSize, keepAliveTime, unit, workQueue);
    }

    protected void beforeExecute(Thread t, Runnable r) {
        super.beforeExecute(t, r);
        inProgress.put(r, Boolean.TRUE);
        startTime.set(new Long(System.currentTimeMillis()));
    }

    protected void afterExecute(Runnable r, Throwable t) {
        long time = System.currentTimeMillis() - startTime.get().longValue();
        synchronized (this) {
            totalTime += time;
            ++totalTasks;
        }
        inProgress.remove(r);
        super.afterExecute(r, t);
    }

    public Set<Runnable> getInProgressTasks() {
        return Collections.unmodifiableSet(inProgress.keySet());
    }

    public synchronized int getTotalTasks() {
        return totalTasks;
    }

    public synchronized double getAverageTaskTime() {
        return (totalTasks == 0) ? 0 : totalTime / totalTasks;
    }
}
```

リスト 4 の ThreadPoolStatusMBean は、TrackingThreadPool の MBean インターフェースを示しています。このインターフェースはアクティブ・タスク、アクティブ・スレッド、完了したタスク、待機中のタスクの総数を表示し、現在待機中のタスクと現在実行中のタスクをリストします。管理インターフェースに待機中および実行中のタスクのリストを含めると、アプリケーションの動作状況だけでなく、現在の操作対象も確認できます。この機能により、アプリケーションの動作に限らず、アプリケーションが操作中のデータ・セットの特性まで把握することができます。

リスト 4. TrackingThreadPool の MBean インターフェース

```
public interface ThreadPoolStatusMBean {
    public int getActiveThreads();
    public int getActiveTasks();
    public int getTotalTasks();
    public int getQueuedTasks();
    public double getAverageTaskTime();
    public String[] getActiveTaskNames();
    public String[] getQueuedTaskNames();
}
```

タスクのサイズがかなり大きい場合は、さらに拡張して、それぞれのタスクがサブミットされるごとに MBean を登録 (そしてタスクの終了後に登録解除) するという方法も考えられます。この場合、管理インターフェースを使って、各タスクに実行中の内容と実行に費やしている時間を照会したり、タスクのキャンセルを要求することができます。

リスト 5 の ThreadPoolStatus は ThreadPoolStatusMBean インターフェースを実装して、その中で各アクセサーを明確に実装しています。MBean 実装クラスでの常として、各操作の実装は至って簡単で、基礎となる管理対象オブジェクトに委譲しています。この例では、JMX コードが管理対象エンティティーのコードから完全に分かれています。TrackingThreadPool は JMX をまったく意識しないため、関連する属性に管理メソッドとアクセサーを指定して独自のプログラマチックな管理インターフェースを提供します。管理機能は、実装クラスに直接実装すること (TrackingThreadPool に TrackingThreadPoolMBean インターフェースを実装させること) を、別途実装することもできます (リスト 4 とリスト 5 の例)。

リスト 5. TrackingThreadPool の MBean 実装

```
public class ThreadPoolStatus implements ThreadPoolStatusMBean {
    private final TrackingThreadPool pool;

    public ThreadPoolStatus(TrackingThreadPool pool) {
        this.pool = pool;
    }

    public int getActiveThreads() {
        return pool.getPoolSize();
    }

    public int getActiveTasks() {
        return pool.getActiveCount();
    }

    public int getTotalTasks() {
        return pool.getTotalTasks();
    }

    public int getQueuedTasks() {
        return pool.getQueue().size();
    }

    public double getAverageTaskTime() {
        return pool.getAverageTaskTime();
    }

    public String[] getActiveTaskNames() {
        return toStringArray(pool.getInProgressTasks());
    }

    public String[] getQueuedTaskNames() {
```

```
        return toStringArray(pool.getQueue());
    }

    private String[] toStringArray(Collection<Runnable> collection) {
        ArrayList<String> list = new ArrayList<String>();
        for (Runnable r : collection)
            list.add(r.toString());
        return list.toArray(new String[0]);
    }
}
```

これらのクラスがアプリケーションの操作対象を明らかにする方法を理解するには、その操作をリモート・ページのフェッチとページの索引付けという 2 種類のタスクに分けている Web クローラー・アプリケーションについて考えてみてください。各タスクは、FetchTask または IndexTask のいずれかによって記述されます (リスト 6 を参照)。これらのタスクを処理して JMX に登録するために使用するスレッド・プールには、ThreadPoolStatus MBean を作成して管理インターフェースを提供することができます。

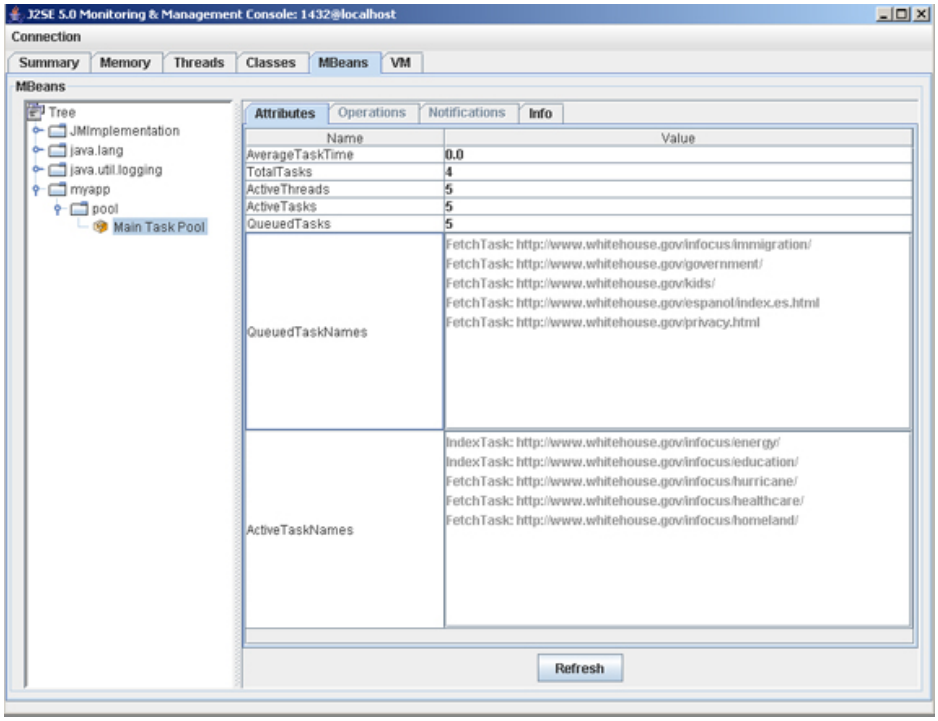
リスト 6. Web クローラー・アプリケーションで使用する FetchTask クラス

```
public class FetchTask implements Runnable {
    private final String name;

    public FetchTask(String name) {
        this.name = name;
    }

    public String toString() {
        return "FetchTask: " + name;
    }

    public void run() {  /* Fetch remote resource */  }
}
```

それぞれのページがクローラーで処理されるごとに、そのページからリンクされたページをフェッチする新しいタスクがキューに入れられるため、未完了のフェッチするタスクと索引付けを行うタスクは常時混在することになります。処理中のページや処理を待機中のページを正確に識別できれば、アプリケーションのパフォーマンス特性だけでなく、操作対象のデータの特性も理解できることになります。

図 4 は、whitehouse.gov サイトを処理中の Web クローラーのスナップショットです。この図を見ると、ホーム・ページのフェッチと索引付けはすでに完了していて、クローラーがこのページから直接リンクされたページのフェッチおよび索引付けを行っていることが分かります。Refresh ボタンを押すと、アプリケーションからタスクのフローをサンプリングできます。大々的なロギングを導入したり、アプリケーションをデバッガーで実行したりしなくても、このサンプルがアプリケーションの動作状況に関する豊富な情報を提供してくれます。

図 4. Web クローラー・アプリケーションでのアクティブ・タスクと待機中タスク



まとめ

プラットフォームでの JMX サポートと jconsole JMX クライアントを組み合わせると、管理機能とモニター機能を手っ取り早くアプリケーションに追加できます。アプリケーションに特定の管理要件がないとしても、これらの機能を組み込むことによって、プログラムの動作状況や処理対象のデータの特性を簡単に見抜くことが可能になります。アプリケーションがエクスポートする管理インターフェースによってその操作対象を確認できれば、アプリケーションの実行内容を詳しく知ることができるだけでなく、ロギング・コードを追加したり、デバッガーやプロファイラーなどの煩わしい手段を使わなくても、アプリケーションが期待通りに機能しているという確信を一層強めることができます。

著者について

Brian Goetz

Brian Goetz は18 年間以上に渡って、専門的ソフトウェア開発者として働いています。彼はカリフォルニア州ロスアルトスにあるソフトウェア開発コンサルティング会社、Quiotixの主席コンサルタントであり、またいくつかのJCP Expert Groupの一員でもあります。2005年の末にはAddison-Wesleyから、Brianによる著、Java Concurrency In Practiceが出版される予定です。Brian著による有力業界紙に掲載済みおよび掲載予定の記事のリストを参照してください。