JVMPI を利用した Java™VM 内部状態監視及びメソッド実行監視方式の実装*

森本 義章　　土井 宏治　　川崎 進一郎　　里山 元章†

(株)日立製作所 システム開発研究所‡

1. 背景：Java システム監視の必要性

近年、Java が様々な用途で利用されている。特に、サーバの分野では Java のエンタープライズ向けの仕様である J2EE™ を使用したシステムが様々な分野で利用されつつある。これに伴い、Java をベースにしたシステムが、ミッションクリティカルな用途にも利用されるようになり、システムの安定動作を実現する Java システムの監視機能が重要視されている。

２．システム安定稼働のための監視の課題

Java システムを安定稼働するためにはシステムの挙動を監視し、管理者に適切に情報を伝え、状態に応じた対処を可能にすることが必要である。システムの監視には以下の二種類がある。

2.1 システム通常運用時の監視

システムを安定動作させるために、システムが通常動作している状態でチューニングに必要な情報を採取出来ることが必要である（例えば起動時パラメタを決定するための情報）。また、メモリーの消費量やシステムの応答時間が異常値になっていないかなどの監視を運用中に行い、必要であれば管理者がシステム構成を変更するなどの処置を行うことが必要である。このためには、オーバヘッドが少なく、かつリモートから容易にシステムを監視できる監視機能が必要となる。

2.2 問題究明時の監視

一旦システム障害が発生した場合、問題究明のために情報採取を行う事がある。開発段階のシステムではこのような場合、プログラムのソースにログを生成するコードを挿入し、再ビルドして実行する方法が利用される。しかし、システムが稼働している環境ではビルドが不可能なことが多く、ソース修正や再コンパイルが必要な手法は利用できないことが多い。従って、2.1 節に示した条件に加えて、最小限のシステム変更で実行の詳細情報が得られる監視手段が必要となる。

2.3 従来ツールの問題点

Java には従来からデバッガやプロファイラと呼ばれるツールが存在する。プロファイラではプログラムのどの部分がどの様に実行されたかの詳細な情報を得ることが出来る。これを利用することで効果的な性能向上が行えるが、適用時のオーバヘッドが大きいために、実際に運用されているシステム上で利用することはできず、デバッガと同様、開発時にのみでしか利用できなかった。

３　内部監視機能実装の概要

3.1　JVMPI の利用によるイベント監視

前述の問題を解決するために、筆者らは、JavaVM の内部状態の監視、スレッド情報、及びメソッドの実行監視を行う JVMPI(JavaVM Profiler Interface)準拠のエージェントを開発した。JVMPI はプログラムのプロファイルを取得するために Java に備えられた API であり、ヒープ使用量やスレッドの状態がイベントとして取得できるので、実行時の内部状態をリアルタイムに知ることが可能である。今回の実装では図1に示す構成とし、エージェントの得た情報をネットワーク経由で外部の監視モニタに出力した。JVMPI エージェントは共有ライブラリとして実装し、JavaVM の起動オプションにエージェントの共有ライブラリ名を指定して起動する。

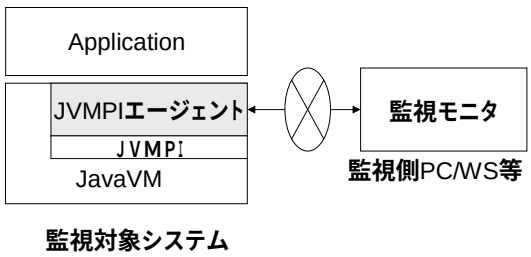

図 1　JVMPI を利用した監視システム概要図

本方式適用時のオーバヘッドを VolanoMark 2.1.2[1]ベンチマークにより測定し、システムに与える影響を測定した。その結果を表１に示す。

*Implementation with JVMPI for monitoring the status of Java virtual machine and method invocation

†Yoshiaki MORIMOTO, Kouji DOI, Shinichiro KAWASAKI, Mototaki SATOYAMA

‡Hitachi, Ltd., Systems Development Laboratry

表 1. JVMPI ｴｰｼﾞｪﾝﾄ利用時のオーバヘッド

	監視するイベント	性能低下率(%)
①	ヒープ利用率(250ms 毎)	1
②	①＋GC 監視＋スレッド生成・終了監視	2～3
③	②＋スレッド状態監視	10～15

WindowsXP(P4-2GHz, RAM512MB),J2SE v.1.4.0

表からわかるように、②の設定であれば数%のオーバヘッドに収まり、運用時に利用可能と考える。

3.2 バイトコード挿入によるメソッド実行監視

JVMPI にはメソッドの実行・実行終了時にイベントを発生する機能があるが、これを enable にすると、全てのメソッド実行毎にイベントが発生し極端に性能が悪化するため、利用することができなかった。この対策として今回、バイトコードを操作することで、低オーバヘッドのメソッド実行監視を実現した。

3.2.1 メソッド監視の従来方式

バイトコードを変更して、監視コードを既存のプログラムに挿入する方式[2]はよく知られた手法である。この方法は、既にコンパイルされたプログラムを直接変更するので、ソースの変更や再コンパイルが必要無い。しかし従来知られている殆どの方法は、クラスローダを変更して読み込むクラスを操作するものか、継承を利用したものである。この方法はプラットフォーム非依存であるという利点があるが、前者はシステムのクラスローダを入れ替える必要があり、後者は private 情報が取得できないため監視に必要な情報が得られないことがあった。

3.2.2 JVMPI を使用したバイトコード挿入方式

そこで筆者らは JVMPI の機能の一つである ClassLoad イベントに注目した。これはクラスが VM に読み込まれた場合に発生するイベントで、イベントの発生と同時に読み込んだクラスデータを変更し、変更したクラスを実行できる。

本実装ではこれを用いて、メソッド実行を監視するコードをロード時に挿入し、実行時は監視コードを経由して監視対象のメソッドを実行することでメソッド呼出の監視を実現した。

また、監視用メソッドを、監視対象メソッドと同じクラスに挿入することで、クラス継承を利用した方式では困難だった private や final 装飾子の付いたメソッドや変数の監視も可能とした。

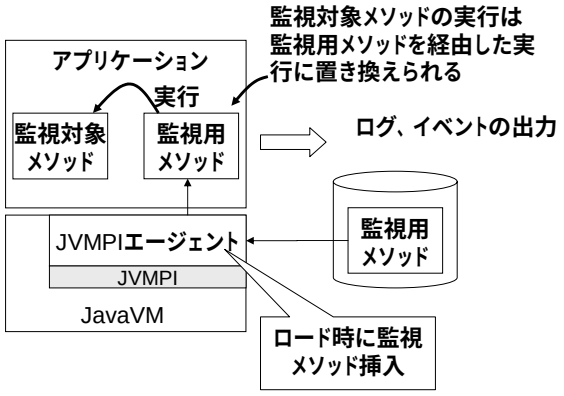

図 2. 監視コード挿入方法の概要

本方式を利用するには、今回開発した監視エージェントである共有ライブラリをシステムに追加するだけで良い。また共有ライブラリはネイティブコードで記述されるために高速処理が可能で、Java で記述されたクラスローダを利用する方式より性能向上に有利と考える。

4 考察・結言

本監視システムの実装を通じて、運用時に利用できる VM 内部監視方式を実現した。JVMPI は性能や安定性で問題があると言われているが、使用するイベントを正しく選択すれば、有効な監視方式として利用可能であることが判明した。ヒープの利用率に閾値を設定してこれを超えると警告するなどの運用時の監視が行える目処が付いた。

また、JVMPI を利用したバイトコードの操作方式を実装し、この方式の有効性を示した。本方式によるバイトコード操作は導入が容易で、制限が少ないため、監視以外の用途に利用できる可能性がある。

今後、Java の最新技術を適用したさらなる機能の追加と、使いやすさの追及を行っていく予定である。例えば、問題の生じたシステムに、システム管理者がマウス操作だけで簡単に監視機構を組み込み、監視できるようなツール群の開発を行っていく。

5 参考文献

[1] http://www.volano.com/benchmarks.html
[2] 田中他：ﾊﾞｲﾄｺｰﾄﾞ編集による Java 言語の表明検査の制御, ｿﾌﾄｳｪｱ科学会, 2000/9