BEN EVANS

Ben Evans（@kittylyst）：jClarity の技術フェロー / 設立者であり、London Java Community（LJC）主催者。Java SE/EE Executive Committee のメンバーでもある。

パート1

JITWatchを使用したJava JITコンパイルの理解

Java HotSpot VMにおけるJITコンパイルの入門ガイド

オラクルの Java HotSpot VM には、非常に高度な Just-In-Time（JIT）コンパイラが搭載されています。これにより、クラス・ファイル（Java ソース・コードをコンパイルしたもの）が実行時にもう一度コンパイルされ、非常に高度に最適化されたマシン・コードへの変換が可能となります。最適化されたコードは極めて高速になり、通常はコンパイル済みの C/C++ コードと同等の速度で（場合によってはさらに高速に）実行されます。

そのため、JIT コンパイラは Java HotSpot VM の最重要要素の 1 つとなっていますが、多くの Java 開発者は JIT コンパイラについて十分には理解しておらず、開発中のアプリケーションで JIT コンパイラによる効果が発揮されるかどうかをチェックする方法も知りません。

折しも、JIT コンパイラによるコードの処理方法を開発者がよりよく理解できるようにするための、JITWatch という新しいオープンソース・ツールが開発されているところです。JITWatch ツールをもっとも効果的に使用するためには、開発者が JIT コンパイルの基本的メカニズムと用語をあらかじめ理解していることが必要です。

本記事は、Java HotSpot VM における JIT コンパイルの基本的な入門書となります。JIT コンパイラで単純なロギングを有効化する方法を紹介した後、最近の Java HotSpot VM バージョンで使用されている、もっとも一般的（かつ重要）ないくつかの JIT コンパイル・テクニックについて考察します。次に、（JITWatch が利用する）さらに詳細なロギング・オプションについて説明します。この説明を基盤として、本シリーズのパート 2 では包括的な JITWatch 概論を掲載する予定です。

はじめに、Java HotSpot VM で実行される JIT コンパイルについて、基本事項をいくつか学習します。

コード・インサイト

JITコンパイラによるコードの処理方法を開発者がよりよく理解できるようにするための、**JITWatchと呼ばれる新しいオープンソース・ツール**が開発されているところです。

基本的な JIT コンパイル

Java HotSpot VM は、どのメソッドが実行されているかを自動的に監視します。（呼出し頻度が高いなど、何らかの基準を満たすことで）コンパイル対象であると判断されたメソッドは、マシン・コードへのコンパイルがスケジュールされます。このようなメソッドをホット・メソッドと呼びます。マシン・コードへのコンパイルは個別の JVM スレッド上で実行され、プログラムの実行が中断されることはありません。実際、コンパイラ・スレッドがホット・メソッドをコンパイルしている間も、コンパイルが完了するまでは、Java 仮想マシン（JVM）ではコンパイル前のインタプリタされた状態のメソッドが引き続き使用されます。

JIT コンパイル・プロセスについて、詳しくは「Java HotSpot VM コード・キャッシュについて」と「Java HotSpot VM における JIT コンパイルの概要」を参照してください。

Java HotSpot VM での JIT コンパイルがコードに与える影響を理解するための第一歩は、コンパイル対象となるメソッドを確認することです。幸いにも、この確認は非常に簡単で、Java プロセスを開始するために使用するスクリプトに

写真：
JOHN BLYTHE

-XX:+PrintCompilation フラグを追加するだけです。

注：生成されるコンパイル・イベントのログは標準ログ（標準出力）に出力され、現時点では、別のファイルにエントリをリダイレクトする方法はありません。

PrintCompilation フラグを使用した場合のログ・エントリの実際の書式は、Java のバージョンによって異なります。ここで、各種 Java バージョンで出力されるログの書式の例をいくつか示します。**リスト 1** は JDK 6 で出力されるログの例です。

JDK 6 の場合、PrintCompilation フラグを使用したログ・エントリでは、最初の番号がコンパイル ID に相当します。この ID は、基本的に、個々のメソッドがコンパイルされ、最適化され、場合によっては脱最適化される間、各メソッドを追跡します。

その後に続くのは、メソッドの特性を示すフラグです。たとえば、s はメソッドが同期されていることを示し、! はメソッドに例外ハンドラが含まれていることを示します。

次に、完全修飾形式のメソッド名と、コンパイルされるメソッドに含まれるバイトコードのバイト数が続きます。ここで若干面倒なのは、メソッド・シグネチャが出力に含まれていない点です。

リスト 2 は JDK 7 以降で出力されるログの例です。JDK 7 のログ書式における大きな変更点は、最初の列がコンパイル実行時間（JVM が起動されてからのミリ秒数）になったことです。その他の出力内容は基本的に JDK 6 と同じです。

PrintCompilation を使用したログ出力の読み方に関しては、優れた記事が多数 Web で公開されています。たとえば Stephen Colebourne 氏と Chris Vest 氏の記事はいずれも非常にお勧めです。

Scala や Groovy など、Java 以外の JVM 言語を使用している場合は、各言語のコンパイラによるクラス・ファイルの生成過程で、メソッド名の（大きな）変更やメソッドの追加または削除が行われる可能性がある点に注意が必要です。

理解が必要

JIT コンパイラは Java HotSpot VM の最重要要素の 1 つとなっていますが、多くの Java 開発者は JIT コンパイラについて十分には理解しておらず、開発中のアプリケーションで JIT コンパイラによる効果が発揮されるかどうかをチェックする方法も知りません。

JIT コンパイル・テクニック

Java HotSpot VM が使用するもっとも一般的な JIT コンパイル・テクニックの 1 つは、インライン化です。インライン化は、メソッドの呼出し元にそのメソッドの本体を展開する手法です。インライン化することでメソッドの呼出しコストを軽減でき、新しいスタック・フレームを作成する必要もありません。デフォルトでは、Java HotSpot VM は、35 バイト未満の JVM バイトコードを含むメソッドのインライン化を試みます。

Java HotSpot VM が実行する、もう 1 つの一般的な最適化は単一ディスパッチです。単一ディスパッチは、あるメソッドのオブジェクト参照の型がコード・パスによって変わることは（通常は）ないという観測された事実に基づいて行われます。

Java は静的型付けのため、コード・パスごとに型が異なるようなケースは生じないだろうと思われがちですが、サブタイプのインスタンスは常にスーパータイプの有効なインスタンスになることを思い出す必要があります（この原則は Barbara Liskov にちなんで、リスコフの置換原則と呼ばれています）。この原則は、1 つのメソッドに対して 2 つのパス（たとえば、スーパータイプのインスタンスを渡すパスと、サブタイプのインスタンスを渡すパス）が存在し得ることを意味しており、この状況は Java の静的型付けのルールにのっとっています（実際にそのような状況が発生します）。

ただし、通常のケース（単一ケース）では、パスによって型が異なるような状況は発生しません。したがって、どのオーバーライドが実際に使用されるかをチェックする必要はないため、渡されたオブジェクトに対してメソッドが呼び出されたときに、どのメソッド定義が呼び出されるのかは正確に分かっています。すなわち、仮想メソッド検索を実行するオーバーヘッドを解消できるため、JIT コンパイラは C++ の同等の呼出しよりもしばしば高速な最適化マシン・コードを生成できます（C++ では、仮想検索の解消が容易ではないためです）。

Java HotSpot VM はその他多数のテクニックを使用して、JIT コンパイルが生成するコードを最適化します。ループの最適化や型の明確化、デッド・コードの削除、固有設定は、Java HotSpot VM が最大限の最適化を実施するために使用する、その他の手段のほんの一部です。複数のテクニックを重ねて使用することが多いため、特定の最適化が適用された時点で、実行可能な別の最適化をコンパイラが確認できるようになる場

リスト1　リスト2

```
JDK 6:

22      java.util.HashMap::getEntry (79 bytes)
23 s!   sun.misc.URLClassPath::getLoader (136 bytes)
```

すべてのリストのテキストをダウンロード

合もあります。

コンパイル・モード

実のところ、Java HotSpot VM の内部には C1 と C2 と呼ばれる 2 つの JIT コンパイラ・モードが別々に存在しています。C1 は、素早い起動と堅牢な最適化が求められるアプリケーションで使用されます。GUI アプリケーションは多くの場合、C1 に適しています。一方、C2 はもともと、長時間実行される、おもにサーバー・サイドのアプリケーションを対象としていました。最近のいくつかの Java SE 7 リリースより前は、C1 モードは -client スイッチ、C2 モードは -server スイッチを指定することで使用できていました。

2 つのコンパイラ・モードは別々の JIT コンパイル・テクニックを使用しており、同じ Java メソッドに対して大きく異なるマシン・コードを出力する可能性があります。しかし、最近の Java アプリケーションは通常、両方のコンパイル・モードを使用できます。この事実を活用すべく、最近の Java SE 7 リリースの一部からは、階層型コンパイルと呼ばれる新機能が利用可能になりました。階層型コンパイルでは、最初に C1 コンパイラ・モードを使用することで、より高い起動パフォーマンスを実現します。アプリケーションが正常に実行されるようになった後は、C2 コンパイラ・モードに引き継ぐことで、より積極的な最適化と（通常は）いっそう高いパフォーマンスを実現します。Java SE 8 では、階層型コンパイルがデフォルトの動作となりました。

モードの選択

2 つの Java HotSpot VM コンパイラ・モードは別々の JIT コンパイル・テクニックを使用しており、同じ Java メソッドに対して大きく異なるマシン・コードを出力する可能性があります。**しかし、最近の Java アプリケーションは通常、両方のコンパイル・モードを使用できます。**

Java HotSpot VM には、コンパイル・イベントのさらに詳細なログを生成する機能が搭載されています。ここからは、このようなログの生成を有効化する方法について確認していきます。

JIT コンパイルの完全ロギング

完全ロギングを有効化するスイッチは -XX:+LogCompilation ですが、その前に -XX:+UnlockDiagnosticVMOptions オプションを指定する必要があります。-XX:+LogCompilation スイッチを使用した場合、起動ディレクトリ内に hotspot_pid<PID>.log という個別ログ・ファイルが生成されます。ファイルの場所を変更するためには、-XX:LogFile=< ファイルのパス > を使用します。

出力されるログは大きい XML ファイル（数十～数百 MB に達することも多い）であり、Java HotSpot VM コンパイラが行った決定に関する詳細な情報が含まれています。このログには、前述の単純書式のログよりも非常に多くの情報が含まれます。

リスト 3 は詳細コンパイル・ログの出力例であり、メソッド（ここでは String::hashCode() メソッド）のコンパイル時に Java HotSpot VM がコンパイルに関して行った決定の詳細情報が大量に含まれています。ただし、書式が複雑であるため、このログの使用は難しく、多くの開発者にとって障壁となります。つまり、この詳細ログを使用してアプリケーションを理解することはできません。しかし、幸いにも、手助けとなるツールが提供されています。

本シリーズのパート 2 では、JITWatch と呼ばれる新しいオープンソース・ツールを紹介します。JITWatch は詳細コンパイル・ログを読み取り、JIT コンパイルの多数の側面をシンプルなグラフィカル表示で提供します。JITWatch は GitHub からダウンロードできます。GitHub では、JITWatch の開発が継続して実施されています。

リスト3

```
<nmethod compile_id='2' compiler='C1' level='3'
entry='0x00000001023fe240' size='1224'
address='0x00000001023fe0d0' relocation_offset='288'
insts_offset='368' stub_offset='880' scopes_data_offset='1032'
scopes_pcs_offset='1104' dependencies_offset='1200'
nul_chk_table_offset='1208'
method='java/lang/String hashCode ()I' bytes='55' count='512'
backedge_count='8218' iicount='512' stamp='0.350'/>
```

すべてのリストのテキストをダウンロード

まとめ

本記事では、Java HotSpot VM にデプロイされた JIT コンパイルの基本的概念のいくつかを紹介しました。また、簡易な書式での出力と、より広範囲にわたる XML 出力の両方のコンパイル・ログの出力に必要なフラグについて解説しました。以上に基づいて、本シリーズのパート 2 では新しい視覚化ツールについて説明する予定です。**</article>**

MORE ON TOPIC:

Java Virtual Machine

LEARN MORE

- JITWatchのGitHubリポジトリ