Enterprise Application Platform でのトラブルシューティング技法（第 2 回）：Java の性能劣化原因を突き止める ～コードキャッシュ領域不足の確認～

Part1 | Part2

2021 年 9 月 17 日 初版

笠井 輝美

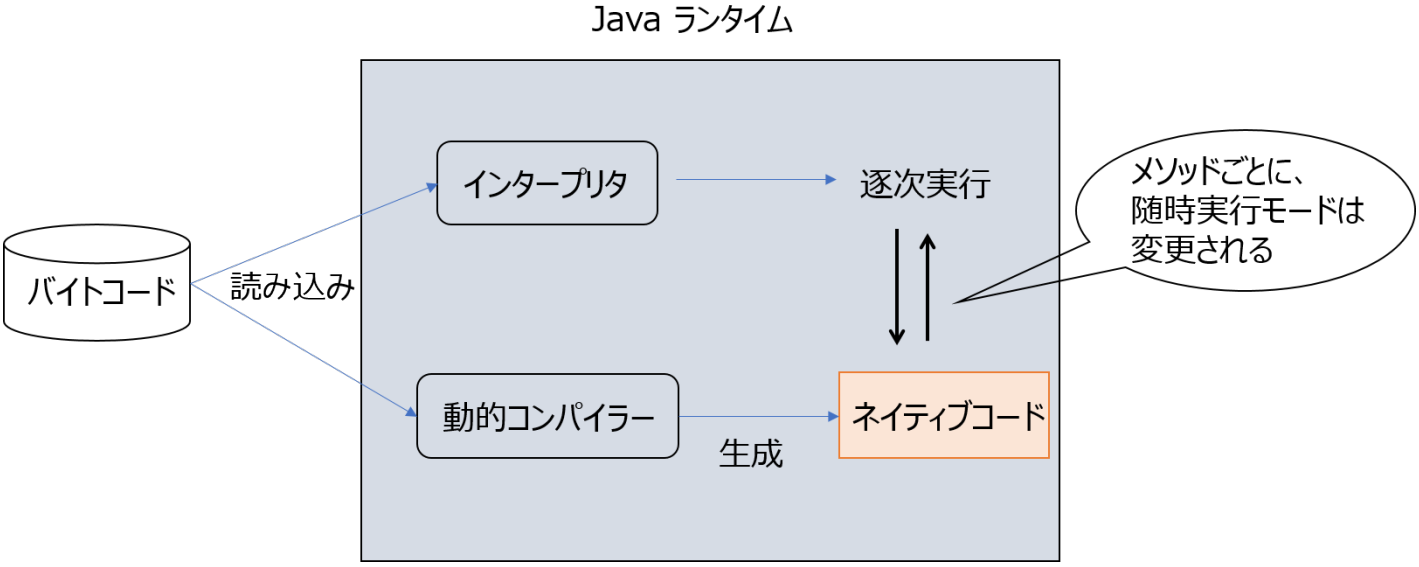
Java の標準的な実行環境である OpenJDK では、Java プログラムを実行しながら同時に動的コンパイラーで徐々にアプリケーション実行を高速化していきます。しかし、動的コンパイラーが使用する「コードキャッシュ」と呼ばれる領域のチューニングを適切に行っていない場合、期待したとおりの高速化が実現できなかったり、突如として性能劣化に陥ることがあります。
今回は、コードキャッシュ領域に起因するトラブルの紹介と、その原因・対処について説明します。

なお、本記事は、FUJITSU Software Enterprise Application Platform の OpenJDK 11 を対象に説明します。

（1）動的コンパイラーとコードキャッシュ領域

Java プログラムを OpenJDK で実行する場合、インタープリタによるバイトコードを 1 命令ずつ解釈して実行と、動的コンパイラーによりネイティブコードを生成してからの実行の二つのモードがあり、両方とも使用されます。

一般的にはプログラム起動時にはインタープリタによる実行割合が多く、プログラムの実行が進むにつれてネイティブコードによる実行割合が多くなっていきます。ネイティブコードは動的コンパイラーによって生成されますが、その名の通り動的に作成され、作成したネイティブコードは Java プロセス内のメモリー上に格納されます。（C 言語などの静的コンパイラーは、ネイティブコードを実行可能ファイルとしてディスクに生成します。）プロセス内のメモリー上でネイティブコードを格納する領域を、「コードキャッシュ領域」と呼びます。



OpenJDK の動的コンパイラーは階層型コンパイラーと呼ばれ、複数の最適化レベルに対応したいくつかのコンパイラーから構成されています。コンパイル処理は一度実施したら終わりではなく、同じ Java メソッドに対して最適化レベルを徐々に上げながら何度もコンパイルします。また逆に、ネイティブコード実行中からインタープリタによる実行に切り替わる場合もあります。（動的コンパイル時に特定の条件が必ず成功することを仮定した最適化を行うことがありますが、その仮定が成立しないことを実行時に検出した場合にインタープリタへの切り替えが発生します）。

なお、ネイティブコードの格納領域が「キャッシュ」と名付けられているのは、ネイティブコードは永続的なものではなくプロセスが終了すれば当然消滅しますが、プログラム実行中においても再コンパイルにより途中で使われなくなり捨てられることもあるためです。

（2）コードキャッシュ領域不足による性能劣化

コードキャッシュが無制限にメモリーを使用しないようにするため、コードキャッシュ領域の大きさはあらかじめ決めておきます。デフォルトサイズや種別については、「（3）コードキャッシュ領域の構成とデフォルトサイズ」で説明します。
このようにコードキャッシュ領域の上限が決まっているため、大規模なシステムを長時間運用している環境ではコードキャッシュ領域不足することがあります。そうなってしまうと動的コンパイラーが動作できず、性能劣化が発生することになります。
コードキャッシュ領域が満杯になった時に、古いコードを削除するフラッシング（Code Cache Flushing）という機能があります。フラッシングによりコードキャッシュ領域に空きができる場合は動的コンパイル処理が再開しますが、動的コンパイル処理が停止している間は、動的コンパイルできなかった Java メソッドはインタープリタ実行されます。
また、フラッシングにより削除されたコードは、再びネイティブコードに変換されるまでインタープリタ実行されます。

コードキャッシュ領域不足により問題となるよくあるケースとしては、性能上重要となる Java メソッドがネイティブコード実行されていたところ、「（1）動的コンパイラーとコードキャッシュ領域」で述べたように、一旦インタープリタ実行に戻り、再び動的コンパイラーがネイティブコードに変換しようとした際に領域不足でコンパイルできず、Java メソッドがインタープリタ実行される場合で、プログラム実行時に突然性能劣化するような現象に見えます。このような事象を回避するためには、コードキャッシュ不足を検出したら速やかにチューニングすることが重要です。
これらについては「（4）コードキャッシュ領域不足の検出」および「（5）コードキャッシュ領域不足時の対処方法」で詳細を説明しますが、その前に必要となるコードキャッシュ領域の構成とデフォルトサイズについて、次に説明します。

（3）コードキャッシュ領域の構成とデフォルトサイズ

コードキャッシュ領域のデフォルトサイズは、OpenJDK 11 の階層型コンパイラーでは 240MB です。コードキャッシュ領域の使用方法は 2 種類あり、セグメント化（Segmented Code Cache）が有効か無効によって切り替わります。どちらを使うかは、JavaVM が自動的に決定します。
セグメント化が有効な場合は、コードキャッシュ領域を 3 つのセグメントに分け、ネイティブコードコードの種類によって格納するセグメントを決定します。
セグメント化が無効な場合は、コードキャッシュ領域をセグメント分けせず、すべての種類のネイティブコードを 1 つの領域に格納します。

ネイティブコードの代表的な種類は以下になります。

- Java メソッドを動的コンパイルした Java ネイティブコード
- インタープリタから Java ネイティブコードの呼び出すとき、および、Java ネイティブコードからインタープリタの呼び出す時に使用するアダプターコード
- Java ネイティブコードから、JavaVM のランタイムルーチンを呼び出す時に使用するスタブコード
- Java ネイティブコードから、（C 言語などで作成した）ネイティブメソッドを呼び出す時に使用するスタブコード

セグメント化が有効な場合の、3 つのセグメントと格納するマシンコードの種類は以下のとおりです。

領域名（セグメント名）	ネイティブコードの種類
non-profiled nmethods	メソッドを最適化レベル 1 または 4 で動的コンパイルしたコード
profiled nmethods	メソッドを最適化レベル 2 または 3 で動的コンパイルしたコード
non-nmethods	上記以外のマシンコード（アダプターやスタブコードなど）

（注 1） 最適化レベルは、階層型コンパイラーの最適化レベルです。

セグメント化が有効な場合のそれぞれのセグメントサイズは JavaVM が決定しますが、おおよそのデフォルトサイズは以下のとおりです。

領域名（セグメント名）	おおよそのデフォルトサイズ
non-profiled nmethods	120,000KB
profiled nmethods	120,000KB
non-nmethods	5,690KB

コードキャッシュ領域およびセグメントのサイズは、後述する jcmd コマンドで調べることができます。

（4）コードキャッシュ領域不足の検出

コードキャッシュ領域不足が発生した場合、OpenJDK は以下に示す警告メッセージを出力します。セグメント化の有無やセグメント種別によってメッセージは 3 種類あります。いずれも、コードキャッシュ領域あるいはセグメントが満杯になったため、新たにネイティブコードを格納できない状態になったときに出力されます。

（4.1）コードキャッシュ領域不足のメッセージ

3 種類のメッセージは以下のとおりです。

- **セグメント化が有効な場合の profiled セグメント不足時のメッセージ**

```
[0.456s][warning][codecache] CodeHeap 'profiled nmethods' is full. Compiler has been disabled.
[0.456s][warning][codecache] Try increasing the code heap size using -XX:ProfiledCodeHeapSize=
OpenJDK 64-Bit Server VM warning: CodeHeap 'profiled nmethods' is full. Compiler has been disabled.
OpenJDK 64-Bit Server VM warning: Try increasing the code heap size using -XX:ProfiledCodeHeapSize=
```

- **セセグメント化が有効な場合の non-profiled セグメント不足時のメッセージ**

```
[0.470s][warning][codecache] CodeHeap 'non-profiled nmethods' is full. Compiler has been disabled.
[0.470s][warning][codecache] Try increasing the code heap size using -XX:NonProfiledCodeHeapSize=
OpenJDK 64-Bit Server VM warning: CodeHeap 'non-profiled nmethods' is full. Compiler has been disabled.
OpenJDK 64-Bit Server VM warning: Try increasing the code heap size using -XX:NonProfiledCodeHeapSize=
```

- **セグメント化が無効な場合のメッセージ**

```
[8.041s][warning][codecache] CodeCache is full. Compiler has been disabled.
[8.041s][warning][codecache] Try increasing the code cache size using -XX:ReservedCodeCacheSize=
OpenJDK 64-Bit Server VM warning: CodeCache is full. Compiler has been disabled.
OpenJDK 64-Bit Server VM warning: Try increasing the code cache size using -XX:ReservedCodeCacheSize=
```

（4.2）コードキャッシュ領域不足メッセージの読み方

コードキャッシュ領域不足メッセージの読み方について説明します。

セグメント化が有効な場合

以下のログは、セグメント化が有効な場合のメッセージです。なお、行頭の番号は説明用に付与しているもので実際とは異なります。

- **セグメント化が有効な場合の profiled セグメント不足時のメッセージ**

```
1: [0.456s][warning][codecache] CodeHeap 'profiled nmethods' is full. Compiler has been disabled.
2: [0.456s][warning][codecache] Try increasing the code heap size using -XX:ProfiledCodeHeapSize=
3: OpenJDK 64-Bit Server VM warning: CodeHeap 'profiled nmethods' is full. Compiler has been disabled.
```



```
 4: OpenJDK 64-Bit Server VM warning: Try increasing the code heap size using -XX:ProfiledCodeHeapSize=
 5: CodeHeap 'non-profiled nmethods': size=156Kb used=151Kb max_used=155Kb free=4Kb
 6:  bounds [0x00007fa4a7077000, 0x00007fa4a709e000, 0x00007fa4a709e000]
 7: CodeHeap 'profiled nmethods': size=152Kb used=142Kb max_used=144Kb free=9Kb
 8:  bounds [0x00007fa4a7051000, 0x00007fa4a7077000, 0x00007fa4a7077000]
 9: CodeHeap 'non-nmethods': size=5692Kb used=1075Kb max_used=1090Kb free=4616Kb
10:  bounds [0x00007fa4a6ac2000, 0x00007fa4a6d32000, 0x00007fa4a7051000]
11: total_blobs=671 nmethods=258 adapters=326
12:  compilation: disabled (not enough contiguous free space left)
13:               stopped_count=1, restarted_count=0
14:  full_count=0
```

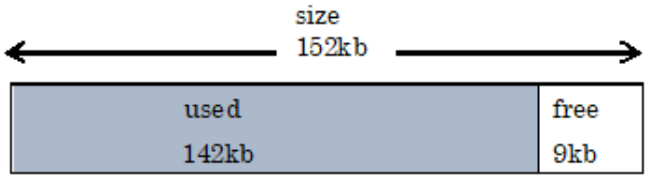
- **セグメント化が有効な場合の non-profiled セグメント不足時のメッセージ**

```
 1: [0.470s][warning][codecache] CodeHeap 'non-profiled nmethods' is full. Compiler has been disabled.
 2: [0.470s][warning][codecache] Try increasing the code heap size using -XX:NonProfiledCodeHeapSize=
 3: OpenJDK 64-Bit Server VM warning: CodeHeap 'non-profiled nmethods' is full. Compiler has been disabled.
 4: OpenJDK 64-Bit Server VM warning: Try increasing the code heap size using -XX:NonProfiledCodeHeapSize=
 5: CodeHeap 'non-profiled nmethods': size=156Kb used=147Kb max_used=155Kb free=8Kb
 6:  bounds [0x00007fa4a7077000, 0x00007fa4a709e000, 0x00007fa4a709e000]
 7: CodeHeap 'profiled nmethods': size=152Kb used=142Kb max_used=144Kb free=9Kb
 8:  bounds [0x00007fa4a7051000, 0x00007fa4a7077000, 0x00007fa4a7077000]
 9: CodeHeap 'non-nmethods': size=5692Kb used=1082Kb max_used=1090Kb free=4609Kb
10:  bounds [0x00007fa4a6ac2000, 0x00007fa4a6d32000, 0x00007fa4a7051000]
11:  total_blobs=672 nmethods=257 adapters=326
12:  compilation: disabled (not enough contiguous free space left)
13:               stopped_count=2, restarted_count=1
14:  full_count=1
```

profiled セグメント不足時のメッセージと non-profiled セグメント不足時のメッセージは、1 行目以外の形式が同じため、以下 profiled セグメントを例に説明します。

行番号	メッセージ	メッセージの意味
1	Compiler has been disabled	メッセージ出力時点で動的コンパイル処理を行わないように変更したことを示します。
5 7 9	CodeHeap	各セグメント名とセグメントの使用状況を示します。数字の意味は以下のとおりです。 • size：セグメントのサイズ • used：メッセージ出力時点のセグメントの使用サイズ • max_used：メッセージ出力時までのセグメントの最大使用サイズ • free：メッセージ出力時点のセグメントの空きサイズ
6 8 10	bounds	3 つの値は順番に、各領域の下端のアドレス、使用済みの終端のアドレス、上端のアドレスを示しています。上端のアドレスから下端のアドレスを引いた値が、その領域で使用できる最大サイズになります。
11	total_blobs	生成されたマシンコード数を示します。1blob は 1 回の変換で作成されるマシンコード列の塊です。
	nmethods	total_blobs のうち、メソッドを動的コンパイルして作成したコード数を示します。
	adapters	total_blobs のうち、アダプター用のコード数を示します。
12	compilation	動的コンパイル処理の状態「有効（enabled）、無効（disabled）」を示します。
13	stopped_count	動的コンパイル処理を停止した回数を示します。
	restarted_count	動的コンパイル処理を再開した回数を示します。
14	full_count	コードキャッシュ領域のセグメントが満杯状態になった回数を示します。一回目の満杯状態を 0 としてカウント開始します。

このログから、使用サイズ（used）がセグメントサイズ（size）に近くなり、空きサイズ（free）が小さくなりコードキャッシュ領域が満杯状態であることが分かります。



セグメント化が無効な場合

以下のログは、セグメント化無効な場合のメッセージです。

- **セグメント化が無効な場合のメッセージ**

```
[8.041s][warning][codecache] CodeCache is full. Compiler has been disabled.
[8.041s][warning][codecache] Try increasing the code cache size using -XX:ReservedCodeCacheSize=
OpenJDK 64-Bit Server VM warning: CodeCache is full. Compiler has been disabled.
OpenJDK 64-Bit Server VM warning: Try increasing the code cache size using -XX:ReservedCodeCacheSize=
CodeCache: size=2496Kb used=2102Kb max_used=2183Kb free=393Kb
 bounds [0x00007f0c09dde000, 0x00007f0c0a04e000, 0x00007f0c0a04e000]
 total_blobs=857 nmethods=409 adapters=359
 compilation: disabled (not enough contiguous free space left)
              stopped_count=1, restarted_count=0
 full_count=0
```

読み方はセグメント化が有効な場合のメッセージと同じです。ネイティブコードの種類に関係なく、コードキャッシュ領域を 1 つの領域として管理しています。

（5）コードキャッシュ領域不足時の対処方法

影響

"CodeCache is full. Compiler has been disabled."のメッセージが出力された場合、コードキャッシュ領域全体が満杯になり動的コンパイル処理が停止されたことを示します。同様に、"CodeHeap 'profiled nmethods' is full. Compiler has been disabled."および"CodeHeap 'non-profiled nmethods' is full. Compiler has been disabled."のメッセージが出力された場合、コードキャッシュ領域のメッセージで示されたセグメントが満杯になり動的コンパイル処理が停止されたことを示します。
この場合、「(2) コードキャッシュ領域不足による性能劣化」で述べたような重大な性能劣化問題を引き起こす場合があるので、タイムリーに適切に対応する必要があります。

対処方法

メッセージに示された Java コマンドラインオプションを指定してコードキャッシュ領域全体、またはセグメントのサイズを大きくするようにします。メッセージが出力されたときの使用最大値（メッセージの"max_used"の値）以上の値を設定します。再び警告メッセージが出力される場合は、設定値を警告メッセージが出力されなくなるまで増加させます。

運用中の Java プロセスを JDK ツールの jcmd コマンドおよび jconsole コマンドで測定して、使用最大サイズがコードキャッシュ領域に余裕をもって収まるようにします。

例 1. ReservedCodeCacheSize を 300MB に設定する場合

```
-XX:ReservedCodeCacheSize=300M
```

例 2. ProfiledCodeHeapSize を 150MB に設定する場合

```
-XX:ProfiledCodeHeapSize=150M
```


コードキャッシュ領域の使用状況の確認（jcmd コマンド）

jcmd コマンドにより、コードキャッシュ領域の使用状況を確認する例を示します。出力される値の意味はそれぞれ下表のとおりです。なお、下表に記載されていない bounds、stopped_count、restarted_count などは「(4.2) コードキャッシュ領域不足メッセージの読み方」に記載したとおりです。

size	コードキャッシュ領域（セグメント）のサイズ
used	測定時点におけるコードキャッシュ領域（セグメント）の使用サイズ
max_used	測定時点までのコードキャッシュ領域（セグメント）の最大使用サイズ
free	測定時点におけるコードキャッシュ領域（セグメント）の空きサイズ
compilation	測定時点において動的コンパイル処理が有効（enabled）か無効（disabled）か

セグメント化が有効な場合の例

312006 は、対象の java プロセスの PID です。

```
jcmd 312006 Compiler.codecache
312006:
CodeHeap 'non-profiled nmethods': size=120036Kb used=1376Kb max_used=1439Kb free=118659Kb
 bounds [0x00007ff8c036a000, 0x00007ff8c05da000, 0x00007ff8c78a3000]
CodeHeap 'profiled nmethods': size=120032Kb used=1609Kb max_used=3628Kb free=118422Kb
 bounds [0x00007ff8b8e32000, 0x00007ff8b91c2000, 0x00007ff8c036a000]
CodeHeap 'non-nmethods': size=5692Kb used=1098Kb max_used=1143Kb free=4594Kb
 bounds [0x00007ff8b88a3000, 0x00007ff8b8b13000, 0x00007ff8b8e32000]
 total_blobs=1617 nmethods=1171 adapters=359
 compilation: enabled
              stopped_count=0, restarted_count=0
 full_count=0
```

セグメント化が無効な場合の例

332376 は、対象の java プロセスの PID です。

```
/opt/FJSVeapf/openjdk/jdk11/bin/jcmd 332376 Compiler.codecache
332376:
CodeCache: size=245760Kb used=6123Kb max_used=6129Kb free=239636Kb
 bounds [0x00007fda03633000, 0x00007fda03c43000, 0x00007fda12633000]
 total_blobs=2075 nmethods=1629 adapters=359
 compilation: enabled
              stopped_count=0, restarted_count=0
 full_count=0
```

ログに出力されている情報はサイズ情報がコードキャッシュ全体を示してることを除いて、セグメント化有効な場合と同じです。セグメント化が有効、無効、いずれの場合も以下のポイントを確認します。

- compilation が disabled になっていないか？
- stop_count および restarted_count の値から、動的コンパイル処理の起動と停止を繰り返している状態になっていないか？
- full_count の値から、コードキャッシュ領域が満杯の状態を何度も発生していないか？

また、コードキャッシュ領域のサイズチューニングが必要な場合は、運用中の最大サイズがコードキャッシュ領域にある程度の余裕をもって収まるようにします。

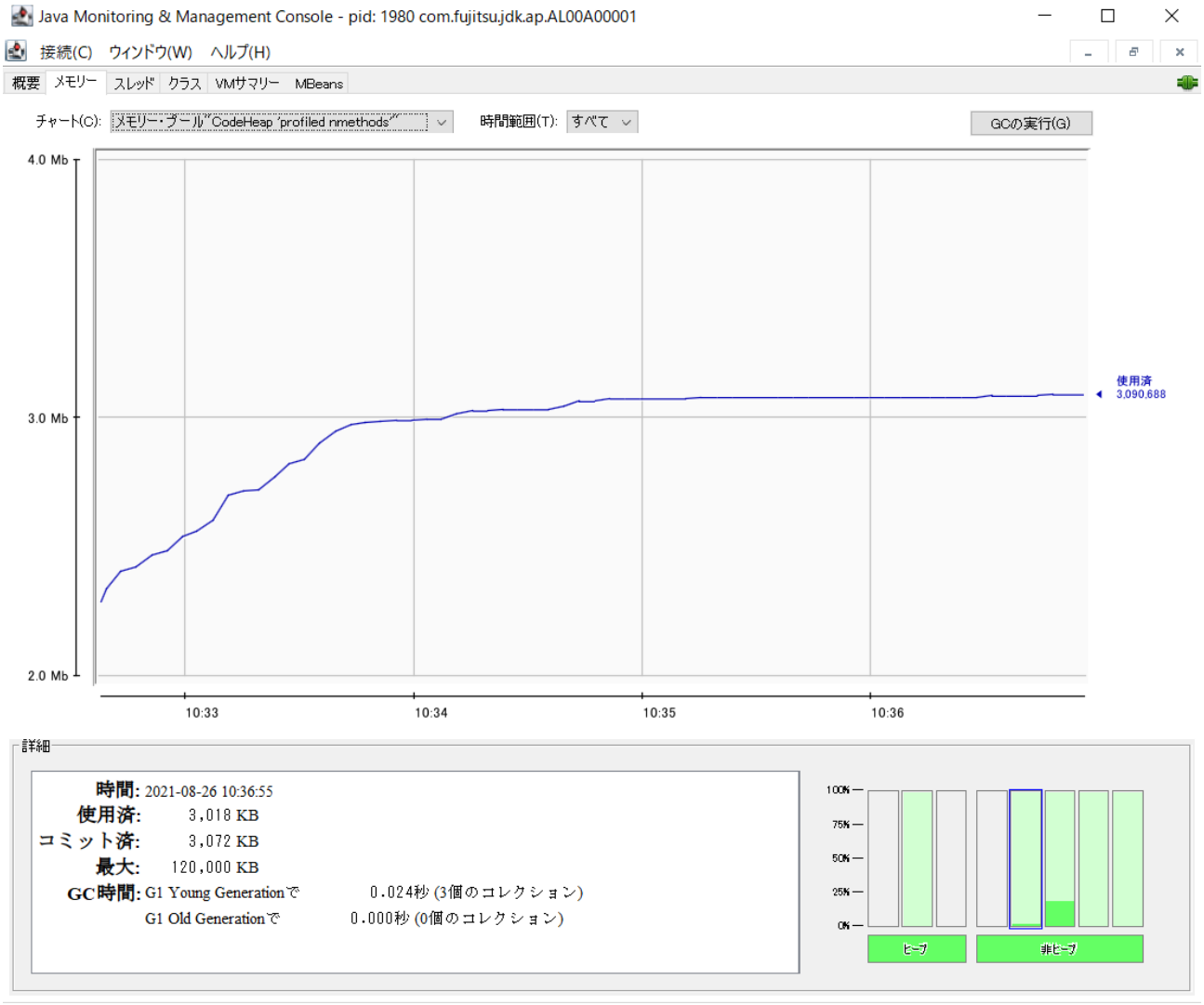
コードキャッシュ領域の使用状況の確認（JConsole）

jconsole コマンドで対象の Java プロセスを監視します。

セグメント化が有効な場合

JConsole 画面で以下のどちらかを選択します。

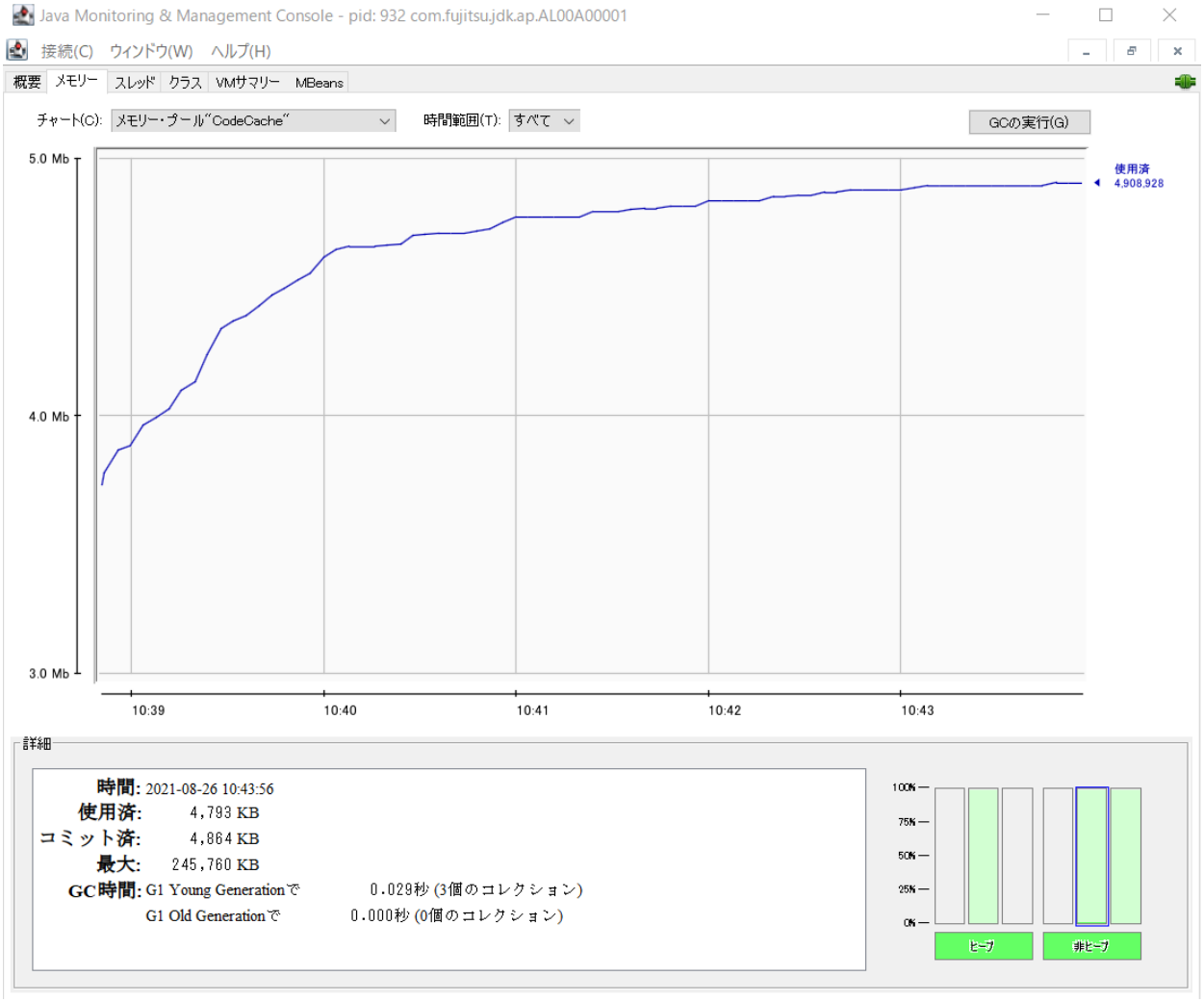
- 「メモリー」タブ – チャート – 「メモリー・プール"CodeHeap 'profiled nmethods'"」
- 「メモリー」タブ – チャート – 「メモリー・プール"CodeHeap 'none nmethods'"」



セグメント化が無効な場合

JConsole 画面で以下を選択します。

- **「メモリー」タブ – チャート – 「メモリー・プール"CodeCache"」**



セグメント化が有効、無効、いずれの場合も使用済みサイズが最大サイズを使い切っていないか確認します。また、コードキャッシュ領域のサイズチューニングが必要な場合は、運用中の最大サイズがコードキャッシュ領域にある程度の余裕をもって収まるようにします。

最後に

今回は、大規模システムで長時間運用続ける場合に突然性能劣化が発生する Java 特有の問題として、コードキャッシュ領域不足による性能劣化について紹介しました。適切なチューニングを実施し、本記事を安定稼働の参考にしてください。