Java の性能トラブルを防止 ～Java のガーベジコレクション（GC）ログの見方を知る（CMS GC 編）～

シリアル GC/パラレル GC 編 | CMS GC 編 | G1 GC 編

2022 年 12 月 9 日 初版
倉繁 泰三

1. はじめに

Java アプリケーションは Java 仮想マシン（Java Virtual Machine：以降、「JVM」と表記します）により実行される際、不要になった Java オブジェクトの解放は、JVM が提供するガーベジコレクターによって自動的に行われます。ガーベジコレクターによるメモリー解放処理であるガーベジコレクション（以降、「GC」と表記します）により、ユーザーはメモリー解放を意識する必要はなくなります。しかし、一方で、一般的なガーベジコレクターでは、GC 処理中はアプリケーションの処理が停止するため、GC の長時間化や頻発は性能トラブルに直結します。

GC に関する性能トラブル調査には、GC ログが有用です。GC ログには、GC 処理に費やした時間や GC 前後のメモリー使用状況などが出力されるため、これらを分析することでチューニングやプログラムの修正に役立てることができます。
以前に掲載した『Java の性能トラブルを防止 ～Java のガーベジコレクション（GC）ログの見方を知る（シリアル GC / パラレル GC 編）～』では、シリアル GC / パラレル GC におけるログの読み方について説明しました。
今回は、Concurrent Mark-Sweep GC（以降、「CMS GC」と表記します）におけるログの読み方について説明します。

本記事は、FUJITSU Software Enterprise Application Platform の OpenJDK 11 を例に説明します。製品の詳細は下記より製品情報ページをご覧ください。

- FUJITSU Software Enterprise Application Platform

2. CMS GC の特徴

CMS GC は、GC 処理の一部をアプリケーションの実行と並列に処理することで、Full GC の発生をできる限り抑えて、アプリケーションの停止時間を短くするように動作します。

2.1. CMS GC の仕組み

CMS GC は、シリアル GC / パラレル GC と同様に、GC 対象となる領域が世代別に管理されています。New 世代領域を対象にした GC を「New GC」、New 世代領域・Old 世代領域などを対象にした GC を「Full GC」と呼びます。
CMS GC は、Old 領域における GC 処理の一部をアプリケーションの実行と並列に処理することで、アプリケーションの停止時間を短くすることが可能です。この、アプリケーションの実行と並列に行う GC 処理を、「コンカレント GC」と呼びます。コンカレント GC は、複数のフェーズから成ります。ただし、コンカレント GC には、アプリケーションと並列実行できないフェーズもあり、そのようなフェーズはアプリケーションを停止して行われます。
コンカレント GC におけるオブジェクトの回収が終了せず、Old 領域の容量が不足している場合は、Full GC が発生してアプリケーションが停止されます。

New 領域の GC はシリアル GC / パラレル GC と同様にアプリケーションを停止して行われます。

2.2. CMS GC の長所と短所

CMS GC は、適切なチューニングを行えば、Full GC の発生を抑え、アプリケーションの停止時間を短くすることが可能です。Java ヒープサイズ、Java ヒープ内の各領域のサイズを決めるオプション、CMS GC の動作を決めるオプション等を使用することで、チューニングを行うことができます。各オプションの詳細説明はオラクル社が提供する「Java Platform, Standard Edition ツール・リファレンス, リリース 11」を参照してください。

- Java ヒープサイズ、Java ヒープ内の各領域のサイズを決める「java の追加オプション」（オラクル社のページへ）
- CMS GC の動作を決める「java の拡張ガベージ・コレクション・オプション」（オラクル社のページへ）

一方、CMS GC の動作を決めるオプションは多く、どのようなチューニングを行うべきかはアプリケーションの実装に依存します。そのため、どのようなオプションを組み合わせて、どのような値を指定してチューニングを行うべきかを決めることは簡単ではありません。
また、アプリケーションと並列に GC 処理を行うため、CPU 使用率が増加しやすいです。

3. Java のガーベジコレクション（GC）ログの読み方（CMS GC）

3.1. 前置き

CMS GC では、「2.1. CMS GC の仕組み」で説明したように、New GC、Full GC、コンカレント GC の 3 種類の GC 処理が行われます。
-Xlog オプションで採取できる GC ログには、-Xlog:gc オプションで得られる簡易的なログ（以降、「簡易ログ」と表記します）と、-Xlog:gc*オプションで得られる詳細なログ（以降、「詳細ログ」と表記します）があります。

簡易ログ、詳細ログでの New GC、Full GC、コンカレント GC のそれぞれのログの読み方について説明します。

3.2 簡易ログ（-Xlog:gc）

3.2.1 New GC

【出力形式】

太字の部分が可変情報で、それ以外の部分は固定情報です。

```
[タイムスタンプ][info][gc] GC(GC 通し番号) Pause Young (GC 発生理由) GC 直前のヒープ使用サイズ->GC 直後のヒープ使用サイズ(ヒープサイズ) GC 時間
```

【出力例】

```
[0.606s][info][gc] GC(2) Pause Young (Allocation Failure) 69M->7M(247M) 6.772ms
```

【意味】

出力例のログからは以下の内容が読み取れます。

- JVM 起動から 0.606 秒後に New 世代領域不足が原因で、2 回目の GC が発生しています。
- ヒープ使用量は 69MB から 7MB になり、GC 後のヒープサイズは 247MB です。
- この GC 処理に費やした時間は、6.772ms です。

3.2.2 Full GC

【出力形式】

太字の部分が可変情報で、それ以外の部分は固定情報です。

```
[タイムスタンプ][info][gc] GC(GC 通し番号) Pause Full (GC 発生理由) GC 直前のヒープ使用サイズ->GC 直後のヒープ使用サイズ(ヒープサイズ) GC 時間
```

【出力例】

```
[2.022s][info][gc] GC(9) Pause Full (Allocation Failure) 226M->176M(247M) 344.529ms
```

【意味】

出力例のログからは以下の内容が読み取れます。

- JVM 起動から 2.022 秒後に領域不足が原因で、9 回目の GC が発生しています。
- ヒープ使用量は 226MB から 176MB になり、GC 後のヒープサイズは 247MB です。
- この GC 処理に費やした時間は、344.529ms です。

3.2.3 コンカレント GC

コンカレント GC は複数のフェーズから成り、それぞれのフェーズのログが出力されます。
また、コンカレント GC のログは、New GC や Full GC のログの途中に混ざって出力される場合があります。GC 通し番号が一致するログを参照してください。

【出力形式】

太字の部分が可変情報で、それ以外の部分は固定情報です。また、Pause Initial Mark フェーズと Pause Remark フェーズは、フェーズ名だけの行が出力されません。フェーズによってサイズが出力されない場合もあります。

```
[タイムスタンプ][info][gc] GC(GC 通し番号) GC フェーズの処理名
[タイムスタンプ][info][gc] GC(GC 通し番号) GC フェーズの処理名 GC 直前の Old 世代領域使用サイズ->GC 直後の Old 世代領域使用サイズ(ヒープサイズ) GC 時間
```

【出力例】

```
[1.247s][info][gc] GC(6) Pause Initial Mark 122M->122M(247M) 1.714ms
[1.247s][info][gc] GC(6) Concurrent Mark
[1.326s][info][gc] GC(6) Concurrent Mark 79.083ms
[1.326s][info][gc] GC(6) Concurrent Preclean
[1.327s][info][gc] GC(6) Concurrent Preclean 0.541ms
[1.327s][info][gc] GC(6) Concurrent Abortable Preclean
[1.479s][info][gc] GC(6) Concurrent Abortable Preclean 152.527ms
[1.483s][info][gc] GC(6) Pause Remark 182M->182M(247M) 3.769ms
[1.483s][info][gc] GC(6) Concurrent Sweep
[1.531s][info][gc] GC(6) Concurrent Sweep 48.003ms
[1.531s][info][gc] GC(6) Concurrent Reset
[1.531s][info][gc] GC(6) Concurrent Reset 0.238ms
```

【意味】

出力例のログからは以下の内容が読み取れます。

- フェーズごとの処理時間は以下のとおりです。
    - Pause Initial Mark：1.714ms
    - Concurrent Mark：79.083ms
    - Concurrent Preclean：0.541ms
    - Concurrent Abortable Preclean：152.527ms
    - Pause Remark：3.769ms
    - Concurrent Sweep：48.003ms
    - Concurrent Reset：0.238ms
- Pause から始まるフェーズは、パラレル GC などと同様にアプリケーションを停止して処理を行います。Concurrent から始まるフェーズはアプリケーション処理のバックグラウンドで並列に実行されます。それぞれ合計すると、5.483ms はアプリケーションを停止し、280.392ms はアプリケーションの処理と並列で GC 処理が行われたことが分かります。

3.3 詳細ログ（-Xlog:gc*）

3.3.1 New GC

【出力形式】

太字の部分が可変情報で、それ以外の部分は固定情報です。また、UUU、SSS、RRR は、GC 処理で使用した時間を示す数値（秒）で、それぞれ、ユーザーCPU 時間、システム CPU 時間、経過時間を示します。

```
[タイムスタンプ][info][gc,start       ] GC(GC 通し番号) Pause Young (GC 発生理由)
[タイムスタンプ][info][gc,task        ] GC(GC 通し番号) Using (アクティブな退避スレッド数) workers of (全退避スレッド数) for evacuation
[タイムスタンプ][info][gc,heap        ] GC(GC 通し番号) ParNew: GC 直前の New 世代領域使用サイズ->GC 直後の New 世代領域使用サイズ(New 世代領域サイズ)
```



```
[タイムスタンプ][info][gc,heap        ] GC(GC 通し番号) CMS: GC 直前の Old 世代領域使用サイズ->GC 直後の Old 世代領域使用サイズ(Old 世代領域使用サイズ)
[タイムスタンプ][info][gc,metaspace   ] GC(GC 通し番号) Metaspace: GC 直前の metaspace 使用サイズ->GC 直後の metaspace 使用サイズ(metaspace サイズ)
[タイムスタンプ][info][gc             ] GC(GC 通し番号) Pause Young (GC 発生理由) GC 直前の Java ヒープ使用サイズ->GC 直後の Java ヒープ使用サイズ(ヒープサイズ) GC 時間
[タイムスタンプ][info][gc,cpu         ] GC(GC 通し番号) User=UUUs Sys=SSSs Real=RRRs
```

【出力例】

```
[0.738s][info][gc,start       ] GC(3) Pause Young (Allocation Failure)
[0.738s][info][gc,task        ] GC(3) Using 4 workers of 4 for evacuation
[0.859s][info][gc,heap        ] GC(3) ParNew: 76606K->8704K(78656K)
[0.859s][info][gc,heap        ] GC(3) CMS: 1270K->40865K(174784K)
[0.859s][info][gc,metaspace   ] GC(3) Metaspace: 12894K->12894K(1060864K)
[0.859s][info][gc             ] GC(3) Pause Young (Allocation Failure) 76M->48M(247M) 121.508ms
[0.859s][info][gc,cpu         ] GC(3) User=0.46s Sys=0.00s Real=0.12s
```

【意味】

出力例のログからは以下の内容が読み取れます。

- New 世代領域不足が原因で、3 回目の GC が発生しました。
    - New 世代領域（ParNew）の使用量は、76,606KB から 8704KB になり、GC 後の領域サイズは 78,656KB です。
    - Old 世代領域（CMS）の使用量は、1,270KB から 40,865KB になり、GC 後の領域サイズは 174,784KB です。
    - Metaspace 領域の使用量は 12,894K のままで、GC 後の領域サイズは 1,060,864KB です。
    - ヒープ全体では、使用量が 76MB から 48MB になり、処理時間が 121.508ms でした。

3.3.2 Full GC

【出力形式】

太字の部分が可変情報で、それ以外の部分は固定情報です。

```
[タイムスタンプ][info][gc,start       ] GC(GC 通し番号) Pause Full (Allocation Failure)
[タイムスタンプ][info][gc,phases,start] GC(GC 通し番号) Phase GC フェーズの通し番号: GC フェーズの処理名
[タイムスタンプ][info][gc,phases      ] GC(GC 通し番号) Phase GC フェーズの通し番号: GC フェーズの処理名 GC フェーズの処理時間
[タイムスタンプ][info][gc             ] GC(GC 通し番号) Pause Full (GC 発生理由) GC 直前の Java ヒープ使用サイズ->GC 直後の Java ヒープ使用サイズ(ヒープサイズ) GC 時間
```

【出力例】

```
[1.637s][info][gc,start       ] GC(9) Pause Full (Allocation Failure)
[1.637s][info][gc,phases,start] GC(9) Phase 1: Mark live objects
[1.754s][info][gc,phases      ] GC(9) Phase 1: Mark live objects 117.029ms
[1.754s][info][gc,phases,start] GC(9) Phase 2: Compute new object addresses
[1.838s][info][gc,phases      ] GC(9) Phase 2: Compute new object addresses 84.129ms
[1.838s][info][gc,phases,start] GC(9) Phase 3: Adjust pointers
[1.928s][info][gc,phases      ] GC(9) Phase 3: Adjust pointers 89.722ms
[1.928s][info][gc,phases,start] GC(9) Phase 4: Move objects
[1.980s][info][gc,phases      ] GC(9) Phase 4: Move objects 52.299ms
[1.980s][info][gc             ] GC(9) Pause Full (Allocation Failure) 226M->176M(247M) 343.834ms
```

【意味】

出力例のログからは以下の内容が読み取れます。

- 領域不足が原因で、9 回目の GC が発生しました。
    - フェーズごとの処理時間は以下のとおりです。
        - Mark live objects：117.029ms
        - Compute new object addresses：84.129ms

- Adjust pointers：89.722ms
- Move objects：52.299ms

○ ヒープ全体では、使用量が 226MB から 176MB になり、処理時間が 343.834ms でした。

3.3.3 コンカレント GC

【出力形式】

太字の部分が可変情報で、それ以外の部分は固定情報です。また、UUU、SSS、RRR は、GC 処理で使用した時間を示す数値（秒）で、それぞれ、ユーザーCPU 時間、システム CPU 時間、経過時間を示します。なお、フェーズによってサイズが出力されない場合もあります。さらに簡易ログの場合は、gc,start、gc,cpu、gc,heap の行が出力されません。

```
[タイムスタンプ][info][gc,start       ] GC(GC 通し番号) GC フェーズの処理名
[タイムスタンプ][info][gc               ] GC(GC 通し番号) GC フェーズの処理名 GC 直前の Old 世代領域使用サイズ->GC 直後の Old 世代領域使用サイズ(ヒープサイズ) GC 時間
[タイムスタンプ][info][gc,cpu          ] GC(GC 通し番号) User=UUUs Sys=SSSs Real=RRRs
[タイムスタンプ][info][gc,heap          ] GC(GC 通し番号) Old: GC 直前の Old 世代領域使用サイズ->GC 直後の Old 世代領域使用サイズ(Old 世代領域使用サイズ)
```

【出力例】

```
[1.221s][info][gc,start       ] GC(6) Pause Initial Mark
[1.222s][info][gc             ] GC(6) Pause Initial Mark 122M->122M(247M) 1.420ms
[1.222s][info][gc,cpu         ] GC(6) User=0.00s Sys=0.00s Real=0.00s
[1.222s][info][gc             ] GC(6) Concurrent Mark
[1.304s][info][gc             ] GC(6) Concurrent Mark 81.922ms
[1.304s][info][gc,cpu         ] GC(6) User=0.17s Sys=0.00s Real=0.09s
[1.304s][info][gc             ] GC(6) Concurrent Preclean
[1.305s][info][gc             ] GC(6) Concurrent Preclean 0.567ms
[1.305s][info][gc,cpu         ] GC(6) User=0.00s Sys=0.00s Real=0.00s
[1.305s][info][gc             ] GC(6) Concurrent Abortable Preclean
[1.448s][info][gc             ] GC(6) Concurrent Abortable Preclean 143.151ms
[1.448s][info][gc,cpu         ] GC(6) User=0.37s Sys=0.00s Real=0.14s
[1.448s][info][gc,start       ] GC(6) Pause Remark
[1.452s][info][gc             ] GC(6) Pause Remark 183M->183M(247M) 3.849ms
[1.452s][info][gc,cpu         ] GC(6) User=0.01s Sys=0.00s Real=0.00s
[1.452s][info][gc             ] GC(6) Concurrent Sweep
[1.501s][info][gc             ] GC(6) Concurrent Sweep 49.232ms
[1.501s][info][gc,cpu         ] GC(6) User=0.12s Sys=0.00s Real=0.05s
[1.501s][info][gc             ] GC(6) Concurrent Reset
[1.501s][info][gc             ] GC(6) Concurrent Reset 0.248ms
[1.501s][info][gc,cpu         ] GC(6) User=0.00s Sys=0.00s Real=0.00s
[1.502s][info][gc,heap        ] GC(6) Old: 115317K->152780K(174784K)
```

【意味】

出力例のログからは以下の内容が読み取れます。

- JVM 起動から 1.221 秒後に、CMS GC の処理が開始されました。
- フェーズごとの処理時間は以下のとおりです。
  - ○ Pause Initial Mark：1.420ms
  - ○ Concurrent Mark：81.922ms
  - ○ Concurrent Preclean：0.567ms
  - ○ Concurrent Abortable Preclean：143.151ms
  - ○ Pause Remark：3.849ms
  - ○ Concurrent Sweep：49.232ms
  - ○ Concurrent Reset：0.248ms
- 5.269ms はアプリケーションを停止し、275.12ms はアプリケーションの処理と並列で GC 処理が行われました。
- Old 世代領域の使用量は、115,317KB から 152,780KB になり、GC 後の領域サイズは 174,784KB です。

4. おわりに

今回は、CMS GC ログの読み方を紹介しました。CMS GC ログを読むことで、Old 領域の GC 処理の一部がアプリケーションの実行と並列に処理されていることを確認してください。また、本記事を、CMS GC を用いたアプリケーションの安定稼働に向けて参考にしてください。

次回は、G1 GC のログの読み方を紹介します。