GraalVM Native Image を使った Java アプリケーションのデバッグ手法紹介

NTT docomo Solutions

Open Source Conference 2025 Fukuoka

2025年11月22日

© NTT DOCOMO SOLUTIONS, Inc. 2025

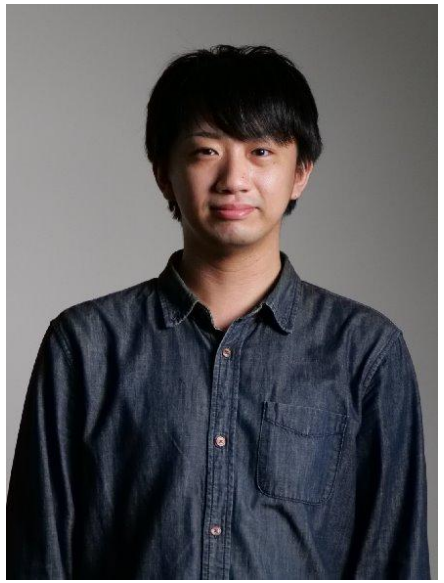
スピーカー紹介

つなごう。驚きを。幸せを。
NTT docomo Solutions

- NTTドコモソリューションズ株式会社（旧 NTTコムウェア）
  - Java／OpenJDK 専門チーム

- 担当業務
  - Java システムの技術サポートや新技術調査
    - JVM トラブルシューティング
    - OpenJDK 仕様調査（ソースコード解析）
    - OpenJDK クラッシュ解析
    - etc.

- 発表実績
  - JDK トラブルシューティング方法と事例紹介（ODC2022）
  - Javaプロセスのメモリ使用量測定を踏まえたコンテナ環境におけるJavaオプション設計指針の紹介（OSC2024広島）



坂本 翔平
（テックリード）

陸 昱宏

© NTT DOCOMO SOLUTIONS, Inc. 2025

本日のお話

つなごう。驚きを。幸せを。
NTT docomo Solutions

GraalVM Native Image のトラブルシューティング手法について紹介する。

- ODC2022 にて OpenJDK のトラブルシューティング方法について発表
- GraalVM の Native Image について、OpenJDK との取得方法の差異に着目しながら
  そのトラブルシューティング手法について紹介

■ **目次**

© NTT DOCOMO SOLUTIONS, Inc. 2025

GraalVM ／ Native Image とは

© NTT DOCOMO SOLUTIONS, Inc. 2025

GraalVM の概要

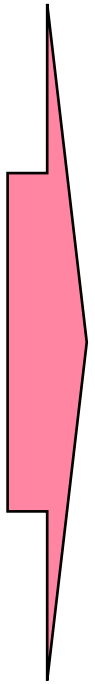
つなごう。驚きを。幸せを。
docomo Solutions

GraalVM は OpenJDK ベースで開発された次世代の Java 開発・実行環境。

GraalVM とは

- 次世代の Java 開発・実行環境
- 従来の Java VM と互換性があり、加えて右記の特徴を持つ高性能なランタイム
- OpenJDK コミュニティにソースコードが寄贈され OpenJDK の性能向上に期待
- 2025年9月に GraalVM 25 がリリース

- GraalVM のディストリビューション
  - GraalVM Community Edition
    - OpenJDK ベース
    - 無償利用可能
  - Oracle GraalVM
    - Oracle JDK ベース
    - GFTC の基、無償利用可能



Graal JIT コンパイラ

- 従来の C2 コンパイラに代わる高性能なもの
- C ではなく Java で実装
- 従来の Java アプリの性能向上も期待できる

多言語実行環境（Truffle）

- JVM 言語（Java、Scala、Kotlin、…）
- インタプリタ言語（Python、JavaScript、…）
- LLVM 対応言語（C、C++、…）

Native Image

- 単独実行可能なバイナリファイルを生成
- 詳細は次スライドにて説明

© NTT DOCOMO SOLUTIONS, Inc. 2025

GraalVM Native Image の概要

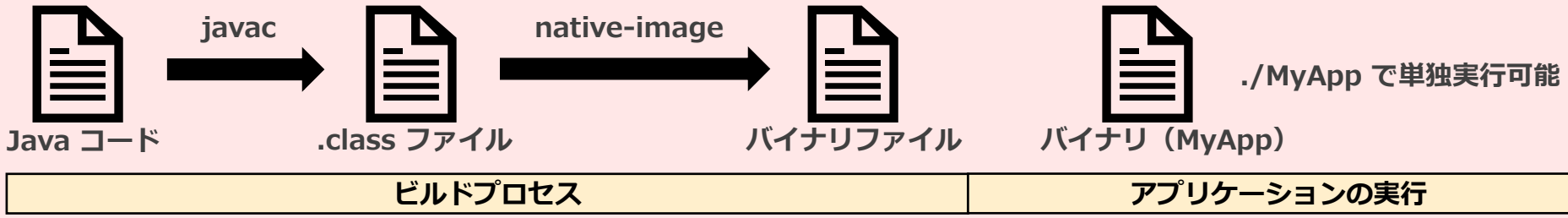
つなごう。驚きを。幸せを。
NTT docomo Solutions

Native Image は GraalVM の主要機能の１つであり、
AOT コンパイルにより自己完結型の実行ファイルを生成する。

GraalVM Native Image とは

- Java コードを静的解析して **AOT（Ahead-Of-Time: 事前）コンパイル**し、**単独実行可能なバイナリファイル**を生成する機能
- バイナリファイルの生成には GraalVM 同梱の **native-image** ビルドツール※ を用いる

```
$ javac MyApp.java
$ native-image MyApp
```



- Native Image で生成した実行ファイルを利用するメリット
  - JDK 不要で単独実行可能
  - AOT コンパイルによる高速起動・低メモリフットプリント
  - ファイルサイズの軽量化　など…

※ Maven や Gradle では Native Image ビルド用のプラグインが提供されており、専用のコマンド実行により Native Image によるビルドが可能。
（Maven）$ mvn –Pnative package
（Gradle）$ gradle nativeCompile

© NTT DOCOMO SOLUTIONS, Inc. 2025

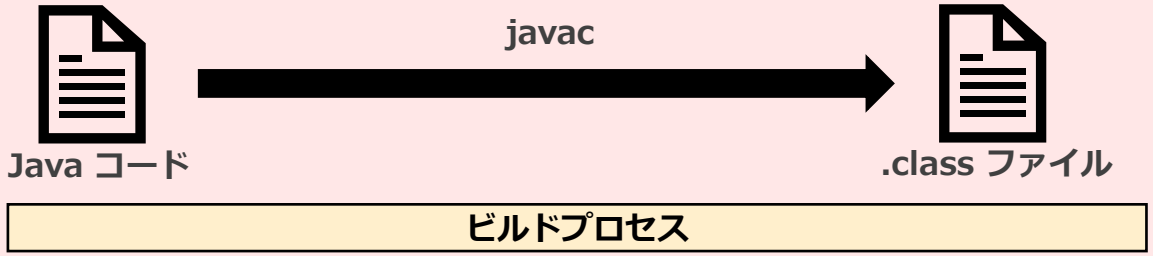
GraalVM Native Image の解析について

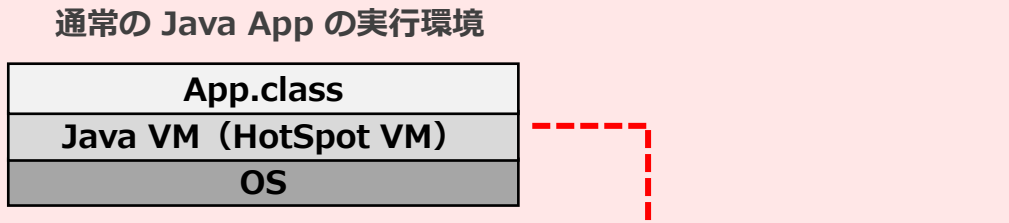
つなごう。驚きを。幸せを。
docomo Solutions

Native Image で生成したアプリケーションは Substrate VM 上で動作するため、
従来の Java VM（HotSpot VM） の解析手法がそのまま適用できない。

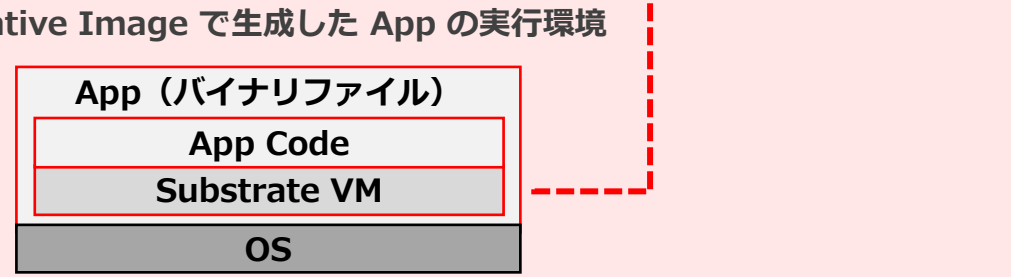
GraalVM Native Image の実行環境

- Native Image により生成されたバイナリファイルはその中に軽量の **Substrate VM** を持つ
- 通常の Java アプリケーションのビルドプロセスと実行環境





- Native Image アプリケーションのビルドプロセスと実行環境



VM が異なる

© NTT DOCOMO SOLUTIONS, Inc. 2025

Java VM の解析情報について

© NTT DOCOMO SOLUTIONS, Inc. 2025

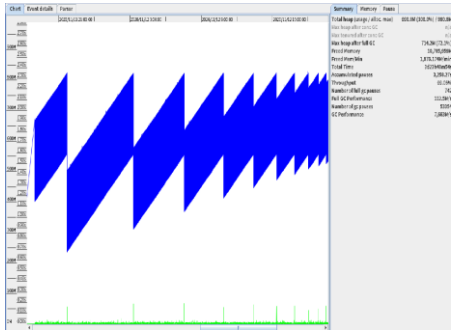
Java VM の解析に有用な情報

つなごう。驚きを。幸せを。
NTT docomo Solutions

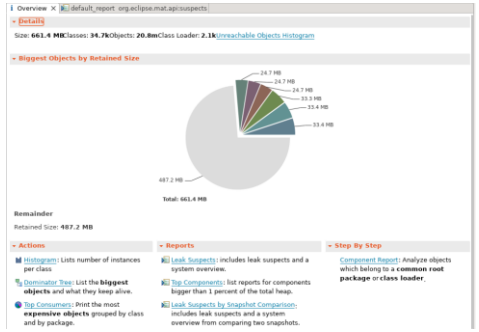
Java システムでは様々な問題が発生するケースが多々あるため、
JVM の出力情報がどういった問題の解析に役立つか把握しておくことが重要。

発生する問題の一例	解析に有用な情報	汎用的に有用な情報
GC の頻発による性能劣化	GCログ、ヒープダンプ	バージョン情報 オプション情報 フライトレコード　など
OutOfMemoryError の発生	GCログ、ヒープダンプ	
Java プロセスのハング	スレッドダンプ	
ネイティブメモリのリーク	Native Memory Tracking（NMT）	
Java プロセスのクラッシュ	エラーレポート、コアダンプ	

効率的に参照できる解析ツールも多数あり



GC ログ（GCViewer）



ヒープダンプ（Eclipse Memory Analyzer）

フライトレコード（JDK Mission Control）

各情報の詳細については ODC 2022 の発表 ※でも解説

※ JDK トラブルシューティング方法と事例紹介: https://event.ospn.jp/odc2022-online/session/637784

© NTT DOCOMO SOLUTIONS, Inc. 2025

Java VM における解析情報の取得方法

つなごう。驚きを。幸せを。
NTT docomo Solutions

従来の Java VM（HotSpot VM）での情報取得方法は以下の通り。

主に Java プロセス起動時に Java オプションを付与して取得する方法と、
実行中の Java プロセスに対してコマンドを実行して取得する方法がある。

情報種別	取得方法
バージョン情報	$ java -version
オプション情報	$ ps aux | grep java
GC ログ	-Xlog:gc=info など（Unified Logging）
スレッドダンプ	$ jcmd <pid> Thread.print
ヒープダンプ	-XX:+HeapDumpOnOutOfMemoryError $ jcmd <pid> GC.heap_dump <output>
フライトレコード	-XX:StartFlightRecording=… $ jcmd <pid> JFR.start
Native Memory Tracking（NMT）	$ jcmd <pid> VM.native_memory
エラーレポート	HotSpot VM による出力
コアダンプ	OS 機能による出力 $ sudo gcore <pid>

© NTT DOCOMO SOLUTIONS, Inc. 2025

Native Image のトラブルシューティング手法

© NTT DOCOMO SOLUTIONS, Inc. 2025

動作確認環境

つなごう。驚きを。幸せを。
NTT docomo Solutions

動作確認は Linux 環境で実施。詳細は以下の通り。

項目	種類やバージョンなど	備考
マシン環境	OS: Ubuntu 22.04 LTS CPU: 4コア8スレッド メモリ: 16GB	
Java アプリケーション	Spring PetClinic ※Commit **b26f235** ※を利用	・Spring Boot 3.5.6 ・Native Build Tools 0.10.6
Java 実行環境	GraalVM 25.0.1+8.1	

- （補足）GraalVM Native Image によるビルドについて
  - Spring PetClinic はデフォルトでネイティブビルドをサポート済
  - （Maven 利用の場合）以下コマンドの実行でネイティブビルド可能

```
$ ./mvnw -Pnative native:compile
...
$ ls ./target/
...
spring-petclinic
```

※ https://github.com/spring-projects/spring-petclinic/commit/b26f235250627a235a2974a22f2317dbef27338d

© NTT DOCOMO SOLUTIONS, Inc. 2025

バージョン情報

つなごう。驚きを。幸せを。
NTT docomo Solutions

バージョン情報は jcmd コマンドで取得できる。

Native Image のビルドオプションを指定

--enable-monitoring=jcmd を指定することで jcmd コマンドを有効化できる。

```
$ native-image --enable-monitoring=jcmd <java_app_name>
```

バージョン情報の取得方法

実行中のアプリケーションプロセスに対して **jcmd** コマンドを使用することで、
ビルドに使用した GraalVM のバージョンとベースの Java バージョンを取得できる。

```
$ jcmd <pid> VM.version
XXXXXX:
GraalVM CE 25.0.1+8.1 (serial gc)
JDK 25.0.1+8
```

© NTT DOCOMO SOLUTIONS, Inc. 2025

オプション情報

つなごう。驚きを。幸せを。
docomo Solutions

実行中のアプリケーションのオプション情報は ps コマンドで取得できる。

オプション情報の取得方法

ps コマンドで実行中のアプリケーションプロセスを確認することで、
指定したオプション情報を確認できる。

```
$ ps aux | grep <native_image_name>
ubuntu   1761532 13.4  0.7 35313416 231788 pts/6 Sl+  09:59   0:00 ./spring-petclinic -
XX:+PrintGC -XX:+PrintGCSummary
ubuntu   1761557  0.0  0.0   4848  2292 pts/12   S+   09:59   0:00 grep --color=auto
spring-petclinic
```

なお Native Image で生成したアプリケーションの実行時に指定可能なオプションの一覧取得には
オプション **-XX:PrintFlags=** を用いる。これはオプション名と概要の記述がセットで出力される。

```
$ <native_image> -XX:PrintFlags=
  -XX:ActiveProcessorCount=-1                 Overwrites the available number of
processors provided by the OS. Any value <= 0 means using the processor count from
                                              the OS.
  -XX:±AutomaticReferenceHandling              Determines if the reference handling is
executed automatically or manually. Default: + (enabled).
 (以下略)
```

© NTT DOCOMO SOLUTIONS, Inc. 2025

GCログ

つなごう。驚きを。幸せを。
NTT docomo Solutions

Native Image で生成したアプリケーションの実行時に、
GC ログオプションを指定することで GC ログが標準エラー出力される。

■ GC ログオプション（Serial GC の例）

GC ログオプション	説明
-XX:±PrintGC	各 GC 実行後の GC 情報を表示
-XX:±VerboseGC	各 GC 実行前後の詳細な情報を表示
-XX:±PrintGCSummary	アプリケーション終了後の GC 情報サマリを表示
-XX:±PrintGCTimes	各 GC のフェーズごとの実行時間を表示
-XX:±TraceHeapChunks	GC 実行中のヒープチャンクを追跡

Native Image を使ったアプリケーションにおける GC ログの注意点

指定できる GC ログオプションは GC 方式によって異なる点に注意。
また Native Image を使ったアプリケーションで出力される GC ログは HotSpot VM のそれとは異なり独自形式であるため、**GCViewer のような既存のグラフ化ツールに対応していない**点に注意。

© NTT DOCOMO SOLUTIONS, Inc. 2025

（参考）GC ログ オプション出力例（1／2）

つなごう。驚きを。幸せを。
docomo Solutions

■ -XX:±VerboseGC の出力例

```
$ <native_image> -Xmx100m -Xms100m -Xmn30m -
XX:+PrintGC -XX:+VerboseGC
[0.068s] GC(0) Using Serial GC
[0.068s] GC(0)   Memory: 31972M
[0.068s] GC(0)   GC policy: adaptive
[0.068s] GC(0)   Maximum young generation size:
30M
[0.068s] GC(0)   Maximum heap size: 100M
[0.068s] GC(0)   Minimum heap size: 100M
[0.068s] GC(0)   Aligned chunk size: 512K
[0.068s] GC(0)   Large array threshold: 128K
[0.068s] GC(0) Collect on allocation
[0.086s] GC(0)   Eden: 22.00M->0.00M
[0.086s] GC(0)   Survivor: 0.00M->0.00M
[0.086s] GC(0)   Old: 0.00M->4.50M
[0.086s] GC(0)   Free: 0.00M->22.00M
[0.086s] GC(0) Pause Full GC (Collect on
allocation) 22.00M->4.50M 17.566ms
```

-XX:±PrintGC **-XX:±VerboseGC**

■ -XX:+PrintGCSummary の出力例

```
$ <native_image> -Xmx100m -Xms100m -Xmn30m -
XX:+PrintGC -XX:+PrintGCSummary
[0.088s] GC(0) Pause Full GC (Collect on
allocation) 22.00M->4.50M 15.159ms
[0.145s] GC(1) Pause Incremental GC (Collect on
allocation) 26.50M->8.50M 10.249ms
[0.216s] GC(2) Pause Full GC (Collect on
allocation) 30.50M->10.50M 24.042ms
（省略）
GC summary
  Collected chunk bytes: 154.50M
  Collected object bytes: 154.01M
  Allocated chunk bytes: 186.00M
  Allocated object bytes: 184.12M
  Incremental GC count: 5
  Incremental GC time: 0.044s
  Complete GC count: 3
  Complete GC time: 0.070s
  GC time: 0.114s
  Run time: 0.477s
  GC load: 24%
```

-XX:±PrintGC **-XX:+PrintGCSummary**

© NTT DOCOMO SOLUTIONS, Inc. 2025

（参考）GC ログ オプション出力例（2／2）

つなごう。驚きを。幸せを。
docomo Solutions

■ -XX:+PrintGCTimes の出力例

```
$ <native_image> -Xmx100m -Xms100m -Xmn30m -
XX:+VerboseGC -XX:+PrintGCTimes
[0.074s] GC(0) Using Serial GC
[0.074s] GC(0)   Memory: 31972M
[0.074s] GC(0)   GC policy: adaptive
(省略)
 [GC nanoseconds:
    collection: 16796008
      rootScan: 16740019
        scanFromRoots: 5198619
        scanFromDirtyRoots: 11540783
          promotePinnedObjects: 8844
          blackenStackRoots: 99420
          walkThreadLocals: 2832
          blackenImageHeapRoots: 1987499
          blackenDirtyCardRoots: 142
          scanGreyObjects: 14634625
      referenceObjects: 3020
      releaseSpaces: 20195
    GCLoad: 18%][0.092s]
```

-XX:±VerboseGC

-XX:+PrintGCTimes

■ -XX:+TraceHeapChunks の出力例

```
$ <native_image> -Xmx100m -Xms100m -Xmn30m -
XX:+VerboseGC -XX:+TraceHeapChunks
[0.074s] GC(0) Using Serial GC
[0.074s] GC(0)   Memory: 31972M
[0.074s] GC(0)   GC policy: adaptive
 (省略)
|0x0000768f4c800000|0x0000768f4c800838,
0x0000768f4c880000, 0x0000768f4c880000|100%|
I|A| |0
|0x0000768f4c880000|0x0000768f4c880838,
0x0000768f4c8ffff8, 0x0000768f4c900000| 99%|
I|A| |0
|0x0000768f4c900000|0x0000768f4c900838,
0x0000768f4c980000, 0x0000768f4c980000|100%|
I|A| |0
|0x0000768f4c980000|0x0000768f4c980838,
0x0000768f4ca00000, 0x0000768f4ca00000|100%|
I|A| |0
```

-XX:±VerboseGC

-XX:+TraceHeapChunks

© NTT DOCOMO SOLUTIONS, Inc. 2025

スレッドダンプ（1／3）

つなごう。驚きを。幸せを。
NTT docomo Solutions

スレッドダンプはビルドオプションを指定し QUIT シグナルを送信するか、
もしくは jcmd コマンド（詳細は次スライド）により取得できる。

Native Image のビルドオプションを指定

--enable-monitoring=threaddump を指定することでスレッドダンプ取得機能を有効化できる。

```
$ native-image --enable-monitoring=threaddump <java_app_name>
```

QUIT シグナル送信によるスレッドダンプ取得

実行中のアプリケーションプロセスに対して **QUIT シグナルを送信**すればスレッドダンプを出力できる。

```
$ kill -QUIT <pid>
```

出力はアプリケーションプロセス側に標準エラー出力される

※ GraalVM for JDK 21 までは -H:±DumpThreadStacksOnSignal により機能を有効化できたが、
--enable-monitoring のサポート追加により非推奨となった。
このオプションを引き続き使用しているとビルド時に以下の警告が出力される。

```
Warning: Option 'DumpThreadStacksOnSignal' is deprecated and might be removed in a future
release: Please use '--enable-monitoring=threaddump'. Please refer to the GraalVM release
notes.
```

© NTT DOCOMO SOLUTIONS, Inc. 2025

スレッドダンプ（2／3）

つなごう。驚きを。幸せを。
NTT docomo Solutions

スレッドダンプは Native Image の jcmd サポート追加に伴い、
jcmd コマンドで取得することもできる。

jcmd コマンドによるスレッドダンプの取得

実行中のアプリケーションプロセスに対して、**jcmd <pid> Thread.print** を実行することで
スレッドダンプを取得できる。

```
$ jcmd <pid> Thread.print
XXXXXX:
Threads dumped.
```

jcmd コマンドでスレッドダンプを操作する場合、**ビルド時に jcmd を有効化する必要がある**ため注意。
なお前述の --enable-monitoring=threaddump は QUIT シグナルによるスレッドダンプ出力のための
機能であり、診断コマンド Thread.print についてはこのビルドオプションの指定がなくても使用可能。

また jcmd によるスレッドダンプも**アプリケーションプロセス側で標準エラー出力**される点は注意。

© NTT DOCOMO SOLUTIONS, Inc. 2025

スレッドダンプ（3／3）

つなごう。驚きを。幸せを。
NTT docomo Solutions

スレッドダンプの出力には Native Image 特有のスタック（Substrate VM 実装）が含まれる。

```
"http-nio-8080-exec-1" #53 daemon thread=0x00007ef92c000b80 state=WAITING
  i  SP 0x00007ef94cff8aa0 IP 0x00005a4889b6cf04 size=128
com.oracle.svm.core.posix.headers.Pthread.pthread_cond_wait(Pthread.java)
  A  SP 0x00007ef94cff8aa0 IP 0x00005a4889b6cf04 size=128
com.oracle.svm.core.posix.thread.PosixParker.park0(PosixPlatformThreads.java:372)
  A  SP 0x00007ef94cff8b20 IP 0x00005a4889b6cd0b size=48
com.oracle.svm.core.posix.thread.PosixParker.park(PosixPlatformThreads.java:351)
（省略）
  A  SP 0x00007ef94cff8d90 IP 0x00005a488a642207 size=32    java.lang.Thread.run(Thread.java:1474)
  A  SP 0x00007ef94cff8db0 IP 0x00005a4889bf1207 size=48
com.oracle.svm.core.thread.PlatformThreads.threadStartRoutine(PlatformThreads.java:832)
  A  SP 0x00007ef94cff8de0 IP 0x00005a4889bf1085 size=32
com.oracle.svm.core.thread.PlatformThreads.threadStartRoutine(PlatformThreads.java:808)
  A  SP 0x00007ef94cff8e00 IP 0x00005a4889a79b53 size=96
com.oracle.svm.core.code.IsolateEnterStub.PlatformThreads_threadStartRoutine_Z5jZ9wXZGDAvr0CL8KrTOA(I
solateEnterStub.java:0)
```

Substrate VM 実装の Java コード

© NTT DOCOMO SOLUTIONS, Inc. 2025

ヒープダンプ（1／4）

つなごう。驚きを。幸せを。
NTT docomo Solutions

ヒープダンプを取得するには、ビルド時にヒープダンプ出力を有効化した上で USR1 シグナルを送信する。

Native Image のビルドオプションを指定

--enable-monitoring=heapdump を指定することでヒープダンプ取得機能を有効化できる。

```
$ native-image --enable-monitoring=heapdump <java_app_name>
```

USR1 シグナル送信によるヒープダンプ取得

実行中のアプリケーションプロセスに対して **USR1 シグナルを送信**することでヒープダンプがカレントディレクトリに出力される。なおヒープダンプの取得時には Full GC が発生する。

```
$ kill -USR1 <pid>
```

ヒープダンプ出力先の指定方法

アプリケーション実行時にオプション **-XX:HeapDumpPath=** を指定することで、ヒープダンプの出力先を設定できる。

© NTT DOCOMO SOLUTIONS, Inc. 2025

ヒープダンプ（2／4）

つなごう。驚きを。幸せを。
NTT docomo Solutions

アプリケーション実行時にオプションを指定することで
OutOfMemoryError 発生時にもヒープダンプを出力できる。

OOME 発生時におけるヒープダンプの取得方法

アプリケーション実行時にオプション **-XX:+HeapDumpOnOutOfMemoryError** を指定することで、
OutOfMemoryError 発生時にヒープダンプが出力される。

```
$  ./outofmemoryerror -Xmx100m -Xms100m -Xmn30m -XX:+HeapDumpOnOutOfMemoryError
Dumping heap to svm-heapdump-1765483-OOME.hprof ...
Heap dump file created [85026528 bytes in 0.104 secs]
Exception in thread "main" java.lang.OutOfMemoryError: Array allocation too large.
（以下略）
```

© NTT DOCOMO SOLUTIONS, Inc. 2025

ヒープダンプ（3／4）

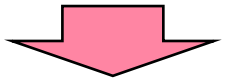
つなごう。驚きを。幸せを。
NTT docomo Solutions

jcmd サポート追加に伴い、jcmd コマンドでヒープダンプを取得することもできる。

Native Image のビルドオプションを指定

--enable-monitoring=jcmd,heapdump を指定することでヒープダンプ取得機能を有効化できる。

```
$ native-image --enable-monitoring=jcmd,heapdump <java_app_name>
```



jcmd コマンドによるヒープダンプの取得

実行中のアプリケーションプロセスに対して、**jcmd <pid> GC.heap_dump** を実行することで
ヒープダンプを取得できる。

```
$ jcmd <pid> GC.heap_dump heapdump.hprof
XXXXXX:
Dumped to: heapdump.hprof
```

© NTT DOCOMO SOLUTIONS, Inc. 2025

ヒープダンプ（4／4）

つなごう。驚きを。幸せを。
docomo Solutions

Native Image アプリケーションから生成されたヒープダンプは
Substrate VM 実装の Java コードや固有の byte 情報を含む。

専用ツールによるヒープダンプの解析

svm-heapdump-1765483-OOME.hprof

Overview | dominator_tree

Class Name	Shallow Heap	Retained Heap	Percentage
<Regex>	<Numeric>	<Numeric>	<Numeric>
java.lang.Thread @ 0x75556b8f5688 main JNI Global, Thread	184	75,498,896	91.30%
<Java Local> byte[75497470] @ 0x7555704b7048 OOME will occur.OOME will occur.OOME will occur.OOME will occur.OOME will occur.C	75,497,496	75,497,496	91.30%
<Java Local> java.lang.OutOfMemoryError @ 0x75556e000838	64	728	0.00%
<JNI Local> com.oracle.svm.core.handles.ThreadLocalHandles @ 0x75556bb80960	40	456	0.00%
objects java.lang.Object[34] @ 0x75556bb80e68	296	376	0.00%
frameStack int[4] @ 0x75556bb80f88	40	40	0.00%
Σ **Total: 2 entries**			
<Java Local> java.lang.StringBuilder @ 0x75556bb80880 OOME will occur.OOME will occur.OOME will occur.OOME will occur.OOME wil	32	32	0.00%
Σ **Total: 4 entries**			
byte[801357] @ 0x75556ba80660 .X.....%a.....f....."....."....Q"....k....@.O.....O.....:.O.....:.O.....W....o.....t....%.x.....)1}....W.....0}.....0}.....&.	801,384	801,384	0.97%
byte[444242] @ 0x75556b400838 ...4...4..4.....t,.n,.....4.....|,...D....."j4..|&..d...H...4$.4...t...*r42z4 .....0..48./D...H4..L@..tF....P.<Nx4X.<fV,p.	444,272	444,272	0.54%
byte[175216] @ 0x75556b480838 x.....e.].v.....oo.....ef...q.fifAR.`\v..H.8.r...L...4.WQ..=.!.....,\..@..4..(.....T...$ $.....[.....P.T9s....{..q?.....N.s?.q.....	175,240	175,240	0.21%
java.lang.String[11014] @ 0x75556b4ab4b8 JNI Global	88,136	88,136	0.11%
byte[81276] @ 0x75556b4c0cf8 .....}.....>.....'...>...M.....*...t.....3...Y.....C...o...../.	81,304	81,304	0.10%
byte[48301] @ 0x75556b761d38 .e..B.B.B..B".BV..,.....B.....,. .B,.B.B.B...B.B... .....6...B...B.,... .....B...B.B...C.....C.,. .C.C.C...C.C.,.C.C...!.!.C...C.!.C....	48,328	48,328	0.06%
byte[46148] @ 0x75556b46cfa0 .....q...q...q.....$.....i.....x.....W.x.....h.5.....k.5.k.V.....x...x.O.....4...q...Y.....T.....	46,176	46,176	0.06%

OOME 原因

Substrate VM 実装の Java コード

Native Image 固有の byte 情報

© NTT DOCOMO SOLUTIONS, Inc. 2025

フライトレコード（1／2）

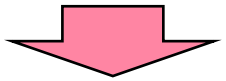
つなごう。驚きを。幸せを。
NTT docomo Solutions

フライトレコードは jcmd コマンドで操作・取得できる。
なお説明は省略するが従来通り起動オプションによる取得も可能。

Native Image のビルドオプションを指定

--enable-monitoring=jfr,jcmd を指定することで、jcmd による JFR 操作を有効化する。
jfr のみの指定だと Java 起動オプションによる JFR 制御はできるが、jcmd による操作はできない。

```
$ native-image --enable-monitoring=jfr,jcmd <java_app_name>
```



jcmd コマンドによるフライトレコードの操作

実行中のアプリケーションプロセスに対して jcmd コマンドで JFR を操作できる。

```
$ jcmd <pid> JFR.start
XXXXXX:
Started recording 1. No limit specified, using maxsize=250MB as default.
```

診断コマンドは **JFR.start / JFR.stop / JFR.dump / JFR.check** の４種が使用できる。
なお HotSpot VM で使用できた JFR.view および JFR.configure については未実装である。

© NTT DOCOMO SOLUTIONS, Inc. 2025

フライトレコード（2／2）

つなごう。驚きを。幸せを。
NTT docomo Solutions

Native Image アプリケーションでは JFR イベントが未実装なものもあるが、
直近でもサポート対応が進められている。

JFR イベント	追加されたバージョン	説明	備考
jdk.AllocationRequiringGC	GraalVM for JDK 22	GC を必要とする割当情報	
jdk.SystemGC		System.gc() の情報	
jdk.ThreadAllocationStatistics		スレッド割当統計	
jdk.ObjectAllocationSample	GraalVM for JDK 23	オブジェクトの割当のサンプリング情報	
jdk.OldObjectSample		Old 領域の生存オブジェクトのサンプリング情報	
jdk.NativeMemoryUsage		カテゴリごとのメモリ使用量	
jdk.NativeMemoryUsageTotal		Total のメモリ使用量	
jdk.NativeMemoryUsagePeak		カテゴリごとのメモリ使用量のピーク	Native Image 固有イベント
jdk.NativeMemoryUsageTotalPeak		Total のメモリ使用量のピーク	Native Image 固有イベント

未実装の JFR イベントについては Graal VM の Issue #5410 ※で実装検討中。

※ https://github.com/oracle/graal/issues/5410

© NTT DOCOMO SOLUTIONS, Inc. 2025

（参考）未実装の JFR イベント例

つなごう。驚きを。幸せを。
NTT docomo Solutions

VisualVM で可視化可能な項目のうち未実装なものは以下の通り。

JFR イベント	VisualVM メニュー	情報の説明	備考
jdk.ThreadDump	Overview – Thread dump	スレッドダンプ	
jdk.CPULoad	Monitor - CPU usage/Environment - CPU utilization	CPU 使用量	
jdk.MetaspaceSummary	Monitor – Metaspace usage	Metaspace 使用量	
jdk.FileRead jdk.FileWrite	File IO	ファイル I/O	
jdk.SocketRead jdk.SocketWrite	Socket IO	ソケット I/O	
jdk.JavaErrorThrow jdk.JavaExceptionThrow	Exceptions	エラーおよび例外一覧	
jdk.GCConfiguration jdk.GCHeapConfiguration jdk.YoungGenerationConfiguration jdk.GCSurvivorConfiguration jdk.GCTLABConfiguration	GC – GC configuration GC – Heap configuration GC – Young Generation configuration GC – Survivor configuration GC – TLAB configuration	各種 GC 関連の設定値	
jdk.ObjectCount	Sampler - Heap histogram	各オブジェクトの数	#7402 ※で検討中
jdk.CPUInformation	Environment - CPU details	CPU 情報	
jdk.NetworkUtilization	Environment - Network utilization	ネットワーク使用量	
jdk.SystemProcess	Environment - System processes	ホストで実行中のプロセス一覧	

※ https://github.com/oracle/graal/issues/7402

© NTT DOCOMO SOLUTIONS, Inc. 2025

Native Memory Tracking（1／2）

NMT は起動オプションや jcmd コマンドから取得できる。

Native Image のビルドオプションを指定

--enable-monitoring=nmt,jcmd を指定することで、NMT 機能を有効化できる。

```
$ native-image --enable-monitoring=nmt,jcmd <java_app_name>
```

NMT 情報の取得方法

NMT 機能を有効化することで -XX:+PrintNMTStatistics が利用できる。
これはアプリケーションの終了時に NMT 情報を出力する。

```
$ ./myapp -XX:+PrintNMTStatistics
```

またビルド時に jcmd サポートを有効化することで jcmd を用いた NMT 情報の取得が可能となる。
jcmd で取得するには診断コマンド VM.native_memory を使用する。

```
$ jcmd <pid> VM.native_memory
XXXXXX:
Native memory tracking
…
```

© NTT DOCOMO SOLUTIONS, Inc. 2025

Native Memory Tracking（2／2）

つなごう。驚きを。幸せを。
NTT docomo Solutions

jcmd コマンドで summary 情報を能動的に取得できる。出力例は以下の通り。

```
$ jcmd XXXXXX VM.native_memory
XXXXXX:
Native memory tracking

Total
    (reserved=33554456KB, committed=138139KB)
    (malloc=24KB, count=28)
    (peak malloc=2448KB, count at peak=33)
    (mmap: reserved=33554432KB, committed=138114KB)
    (mmap: peak reserved=33554432KB, peak committed=149378KB)
Compiler
    (reserved=0KB, committed=0KB)
    (malloc=0KB, count=0)
    (peak malloc=0KB, count at peak=0)
    (mmap: reserved=0KB, committed=0KB)
    (mmap: peak reserved=0KB, peak committed=0KB)
Code
    (reserved=0KB, committed=0KB)
…
```

Native Image における NMT 情報では summary 情報相当のみ出力可能。
detail 相当の情報や、baseline との diff 情報の取得はできない点に注意。

各項目について、
reserved / committed、malloc および mmap の
メモリ量を出力するのは HotSpot VM と同様

© NTT DOCOMO SOLUTIONS, Inc. 2025

エラーレポート（1／2）

つなごう。驚きを。幸せを。
NTT docomo Solutions

Substrate VM は Segmentation Fault 発生時に独自形式のエラーレポートを標準エラー出力する。

エラーレポートの取得方法

Substrate VM の Segfault ハンドラがクラッシュ発生時にエラーレポートを標準エラー出力（≠ファイル出力）する。

エラーレポートの出力内容

■**Substrate VM の Segmentation Fault エラーレポート出力例（先頭部分抜粋）**

```
[ [ SegfaultHandler caught a segfault in thread 0x00006314d1b4f680 ] ]
siginfo: si_signo: 11, si_code: 0, si_addr: 0x000003e800187b4e (heapBase -
130522809337010)

General purpose register values:
  RAX 0xfffffffffffffffc is an unknown value
 (省略)

Printing instructions (ip=0x00007aa5b5e98d71):
  0x00007aa5b5e98c71: 0xff 0x66 0x0f 0x1f 0x44 0x00 Ip 0xb8 0x16 0x00 Ip 0x00 0xc3 0x66
0x90 0x48
```

© NTT DOCOMO SOLUTIONS, Inc. 2025

エラーレポート（2／2）

つなごう。驚きを。幸せを。
NTT docomo Solutions

Segfault ハンドラ機能を無効するとクラッシュ時にコアダンプが出力される。

Segfault ハンドラ機能を無効化することによるコアダンプの取得

Native Image のビルドオプション **–R:-InstallSegfaultHandler** か、実行オプション **-XX:-InstallSegfaultHandler** を指定すると Segfault ハンドラを無効化できる。
Segfault ハンドラが無効されるとクラッシュ時にエラーレポートが出力されず、代わりにコアダンプが出力されるようになる。

■コアダンプ出力時のログ例（Segfault ハンドラ無効時）

```
$ <native_image> –XX:-InstallSegfaultHandler
Segmentation fault (core dumped)
```

※クラッシュのコアダンプの出力は OS の機能に依存するため、出力されない場合はリソース制限設定（ulimit -c）と出力場所（core pattern）を確認すること。

© NTT DOCOMO SOLUTIONS, Inc. 2025

（参考）エラーレポート表示項目一覧（１／２）

つなごう。驚きを。幸せを。
NTT docomo Solutions

項目名（原文）	概要
SegfaultHandler caught a segfault in thread…	シグナルハンドラがキャッチしたシグナル情報＆発生スレッド
General purpose register values	レジスタ値一覧
Printing instructions	インストラクション一覧
Top of stack	スタックのトップアドレス一覧
VM thread locals for the failing thread	問題発生スレッドの VM スレッドローカル一覧
Java frame anchors for the failing thread	問題発生スレッドの Java フレームアンカー（アドレス）
Stacktrace for the failing thread	問題発生スレッドのスタックトレース
Threads	スレッド一覧
The 30 most recent VM operation status changes	直近30個の VM オペレーションログ
VM mutexes	VM ミューテックス一覧
Build time information	ビルド時情報
Runtime information	実行時情報
OS information	OS 情報
Command line	アプリケーションのコマンドライン引数

© NTT DOCOMO SOLUTIONS, Inc. 2025

（参考）エラーレポート表示項目一覧（２／２）

つなごう。驚きを。幸せを。
NTT docomo Solutions

項目名（原文）	概要
Heap settings and statistics	ヒープ設定と統計
Heap usage	ヒープ使用量
GC policy	GC 設定情報
Image heap boundaries	Native Image のヒープ境界
Heap chunks	ヒープチャンク

© NTT DOCOMO SOLUTIONS, Inc. 2025

コアダンプ（1／3）

つなごう。驚きを。幸せを。
NTT docomo Solutions

ビルド時のオプションによりコアダンプ時にデバッグ情報を生成する。

Native Image のビルドオプションを指定

コンパイルやネイティブビルド時に **-g** を指定することでデバッグ情報を生成できる。
また **-O0** を指定することで最適化を無効化し完全なデバッグ情報を生成できるので指定を推奨。

```
$ javac -g MyApp.java
$ native-image -g -O0 MyApp
```

デバッグ情報はデバッグインフォ（.debug）とデバッグソース（sources）の２つからなり、
ビルド時に -g を指定することでこれらのデバッグ情報のファイル一式が生成される。

```
Build artifacts:
 /path/to/build/home/myapp/gdb-debughelpers.py (debug_info)
 /path/to/build/home/myapp (executable)
 /path/to/build/home/myapp.debug (debug_info)
 /path/to/build/home/sources (debug_info)
```

なお GraalVM for JDK 24 から GDB の Python 用 API（gdb-debughelpers.py）※も追加されている。

※ https://www.graalvm.org/latest/reference-manual/native-image/guides/debug-native-image-process-with-python-helper-script/

© NTT DOCOMO SOLUTIONS, Inc. 2025

コアダンプ（2／3）

つなごう。驚きを。幸せを。
NTT docomo Solutions

コアダンプは実行中のプロセスに対して gcore コマンドを使用して取得できる。

コアダンプの取得方法

コアダンプはプロセスのクラッシュ時に OS 機能により出力されるか、実行中のプロセスに対して gcore コマンドを使用することで取得できる。

```
$ sudo gcore <pid>
```

© NTT DOCOMO SOLUTIONS, Inc. 2025

コアダンプ（3／3）

つなごう。驚きを。幸せを。
docomo Solutions

コアダンプの情報参照には GDB を用いる。

コアダンプの参照方法

Native Image アプリケーションのコアダンプは GDB で参照できる。

```
$ gdb <native-image> <coredump>
```

デバッグ情報を生成し GDB が読み取ることで、コアダンプの内容を確認できる。
以下はデバッグ情報有無の差分である。

■デバッグ情報なしの場合

```
(gdb) bt
#0  __futex_abstimed_wait_common64 (private=0,
cancel=true, abstime=0x0, op=393, expected=0,
futex_word=0x7e828708c08c) at ./nptl/futex-
internal.c:57
…
#5  0x000061b5c8b24833 in ?? ()
#6  0x00007e8287d42738 in ?? ()
#7  0x000061b5c8b2b716 in ?? ()
…
```

■デバッグ情報ありの場合

```
(gdb) bt
#0  __futex_abstimed_wait_common64 (…
…
#5  0x000061b5c8b24833 in
com.oracle.svm.core.posix.headers.Pthread::pt
hread_cond_wait_no_transition(com.oracle.svm.
core.posix.headers.Pthread$pthread_cond_t,
com.oracle.svm.core.posix.headers.Pthread$pth
read_mutex_t) (
    __0=<optimized out>, __1=<optimized out>)
```

© NTT DOCOMO SOLUTIONS, Inc. 2025

まとめ

© NTT DOCOMO SOLUTIONS, Inc. 2025

まとめ

つなごう。驚きを。幸せを。
NTT docomo Solutions

GraalVM 25 における Native Image の解析情報の取得方法について以下にまとめる。

情報種別 アップデートあり	取得方法	
	必要なビルドオプション	起動オプションや取得コマンド
バージョン情報	--enable-monitoring=jcmd	$ jcmd <pid> VM.version
オプション情報	-	$ ps aux | grep <native_image>
GC ログ	-	-XX:+PrintGC, -XX:+VerboseGC など
スレッドダンプ	--enable-monitoring=jcmd	$ jcmd <pid> Thread.print
	--enable-monitoring=threaddump	$ kill -QUIT <pid>
ヒープダンプ	--enable-monitoring=heapdump	-XX:+HeapDumpOnOutOfMemoryError $ kill –USR1 <pid>
	--enable-monitoring=jcmd,heapdump	$ jcmd <pid> GC.heap_dump <output>
フライトレコード	--enable-monitoring=jfr	-XX:StartFlightRecording=…
	--enable-monitoring=jcmd,jfr	$ jcmd <pid> JFR.start/stop/dump/check
Native Memory Tracking（NMT）	--enable-monitoring=nmt	-XX:+PrintNMTStatistics
	--enable-monitoring=jcmd,nmt	$ jcmd <pid> VM.native_memory
エラーレポート	-	Substrate VM による出力
コアダンプ	-g（デバッグ情報の生成のため）	OS機能による出力 / sudo gcore <pid>

© NTT DOCOMO SOLUTIONS, Inc. 2025

Q&A

© NTT DOCOMO SOLUTIONS, Inc. 2025

つなごう。驚きを。幸せを。

NTT docomo Solutions

© NTT DOCOMO SOLUTIONS, Inc. 2025

（参考）GraalVM 22-25 までの解析機能アップデート

解析に関連する GraalVM 25 までの主なアップデートは以下の通り。
①jcmd サポートの追加（GraalVM for JDK 24）
②スレッドダンプの機能サポート追加と従来のオプションの非推奨化
③新規 JFR イベントのサポート追加（GraalVM for JDK 22-23）
④NMT の初期サポートの追加（GraalVM for JDK 23）

項目（赤字: 21から差分あり）	**GraalVM 22-25 のアップデート**	**備考**
バージョン情報	①jcmd サポートの追加	
オプション情報	①jcmd サポートの追加	動作確認では取得不可
GC ログ	差分なし	
スレッドダンプ	①jcmd サポートの追加 ②スレッドダンプの取得サポートの追加と -H:+DumpThreadStacksOnSignal の非推奨化	②は GraalVM for JDK 22 で --enable-monitoring=threadump がサポートされたことによりオプションが非推奨となった ①のサポート前は kill -QUIT による取得のみ可能
ヒープダンプ	①jcmd サポートの追加	
フライトレコード	①jcmd サポートの追加 ③新規 JFR イベントのサポート追加	
Native Memory Tracking（NMT）	①jcmd サポートの追加 ④NMT の初期サポートの追加	
エラーレポート	差分なし	
コアダンプ	差分なし	

© NTT DOCOMO SOLUTIONS, Inc. 2025

（参考）jcmd サポート（1／2）

つなごう。驚きを。幸せを。
NTT docomo Solutions

GraalVM for JDK 24 で追加され、各種情報取得が容易となった。

Native Image のビルドオプションを指定

--enable-monitoring=jcmd を指定することで、jcmd 機能を有効化できる。

```
$ native-image --enable-monitoring=jcmd <java_app_name>
```

jcmd によるコマンド実行

実行中のアプリケーションプロセスに対して jcmd コマンドを実行できる。実行例※ は以下の通り

・バージョン情報

※スレッドダンプ、フライトレコード、Native Memory Tracking も取得可能だが詳細は別ページで説明

```
$ jcmd <pid> VM.version
XXXXXX:
GraalVM CE 25+37.1 (serial gc)
JDK 25+37
```

jcmd は GraalVM 同梱のバイナリを使用

・ヒープダンプ

```
$ jcmd <pid> GC.heap_dump heapdump.hprof
XXXXXX:
Dumped to: heapdump.hprof
```

© NTT DOCOMO SOLUTIONS, Inc. 2025

（参考）jcmd サポート（2／2）

つなごう。驚きを。幸せを。
docomo Solutions

実行可能な診断コマンドは、指定したビルドオプションにより変わる点に注意。

実行可能な診断コマンドの変化

ビルドオプション --enable-monitoring= で指定した文字列で実行可能な診断コマンドが変化※ する。
実行可能な jcmd の診断コマンドは **jcmd <pid> help** で出力できる。
（例） --enable-monitoring=jcmd のみ

```
$ jcmd <pid> help
XXXXXX:
The following commands are available:
GC.run
Thread.dump_to_file
Thread.print
VM.command_line
VM.system_properties
VM.uptime
VM.version
help
```

なお --enable-monitoring=all を指定することで全ての診断コマンドが利用できる。

※Supported Diagnostic Commands: https://www.graalvm.org/latest/reference-manual/native-image/debugging-and-diagnostics/jcmd/#supported-diagnostic-commands

© NTT DOCOMO SOLUTIONS, Inc. 2025

（参考）オプション情報の jcmd による取得

つなごう。驚きを。幸せを。
docomo Solutions

jcmd サポートが追加されたが、
動作確認ではオプション情報は取得できなかった。

■ オプション情報

```
$ ps aux | grep myapp
User    955644  4.8  1.3 35319040 189748 pts/3 Sl+  11:33   0:00 ./myapp -Xmx100m -
Xms100m -XX:+PrintGC

$ jcmd 955644 VM.command_line
955644:
VM Arguments:
java_command:
```

オプション情報が出力されない

■ （参考）HotSpot VM の出力例

```
$ jcmd 125737 VM.command_line
125737:
VM Arguments:
jvm_args: -Xmx100m -Xms100m -Xlog:gc
java_command: myapp
…
```

© NTT DOCOMO SOLUTIONS, Inc. 2025

（参考）GraalVM バージョン間の差分

つなごう。驚きを。幸せを。
docomo Solutions

GraalVM for JDK 21 と GraalVM 25 の情報取得の差分は以下の通り。

情報種別 差分あり	取得方法	
	GraalVM for JDK 21	GraalVM 25
バージョン情報	$ strings <native_image> | grep “com.oracle.svm.core.VM”	$ jcmd <pid> VM.version
オプション情報	$ ps aux | grep <native_image>	
GC ログ	-XX:+PrintGC -XX:+VerboseGC など	

スレッドダンプ	（ビルド時）-H:+DumpThreadStacksOnSignal
	$ kill -QUIT <pid>
		$ jcmd <pid> Thread.print
		（ビルド時）--enable-monitoring=threaddump
		$ kill -QUIT <pid>

ヒープダンプ	-XX:+HeapDumpOnOutOfMemoryError （ビルド時）--enable-monitoring=heapdump $ kill -USR1 <pid>	左に加えて $ jcmd <pid> GC.heap_dump <output>
フライトレコード	（ビルド時）--enable-monitoring=jfr -XX:StartFlightRecording=…	左に加えて $ jcmd <pid> JFR.start/stop/dump/check
Native Memory Tracking（NMT）	取得不可	（ビルド時）--enable-monitoring=nmt -XX:+PrintNMTStatistics $ jcmd <pid> VM.native_memory
エラーレポート	Substrate VM による出力	
コアダンプ	（ビルド時） -g（デバッグ情報生成のため） / sudo gcore <pid> など	

© N

（参考）NMT 情報の出力項目の差異

つなごう。驚きを。幸せを。
NTT docomo Solutions

HotSpot VM と Native Image の Substrate VM では、
NMT 情報の出力項目に差異がある。

以下は動作確認の出力結果を基に項目の差異を確認した結果である。

- **HotSpot VM と Substrate VM の共通項目**
  - Total, Java Heap, Thread, Code, GC, Compiler, Internal, Native Memory Tracking, Serviceability
- **HosSpot VM のみ出力される項目**
  - Class, Symbol, Shared class space, Arena Chunk, Module, Safepoint, Synchronization, Metaspace, String Deduplication, Object Monitors
- **Substrate VM のみ出力される項目**
  - Heap Dump, Image Heap, JFR, JNI, jvmstat, JVMTI, PGO, Unsafe

© NTT DOCOMO SOLUTIONS, Inc. 2025