AppCDS / AOT Cache で実現する Java アプリケーション起動高速化

2026/06/27(土) Open Source Conference 2026 Hokkaido
NTTドコモソリューションズ株式会社

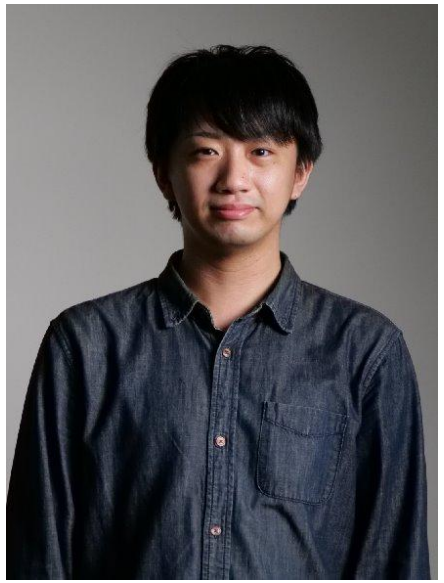
自己紹介

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）
  - 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（OSC2025福岡）



坂本 翔平
（テックリード）

陸 昱宏

本日のお話

つなごう。驚きを。幸せを。
NTT docomo Solutions

OpenJDK 起動高速化機能である AppCDS と AOT Cache について紹介する

- Java アプリケーションの課題とそれに対するアプローチについて
- AppCDS、AOT Cache の利用方法とサンプルアプリケーションを用いた性能改善例

■ 目次

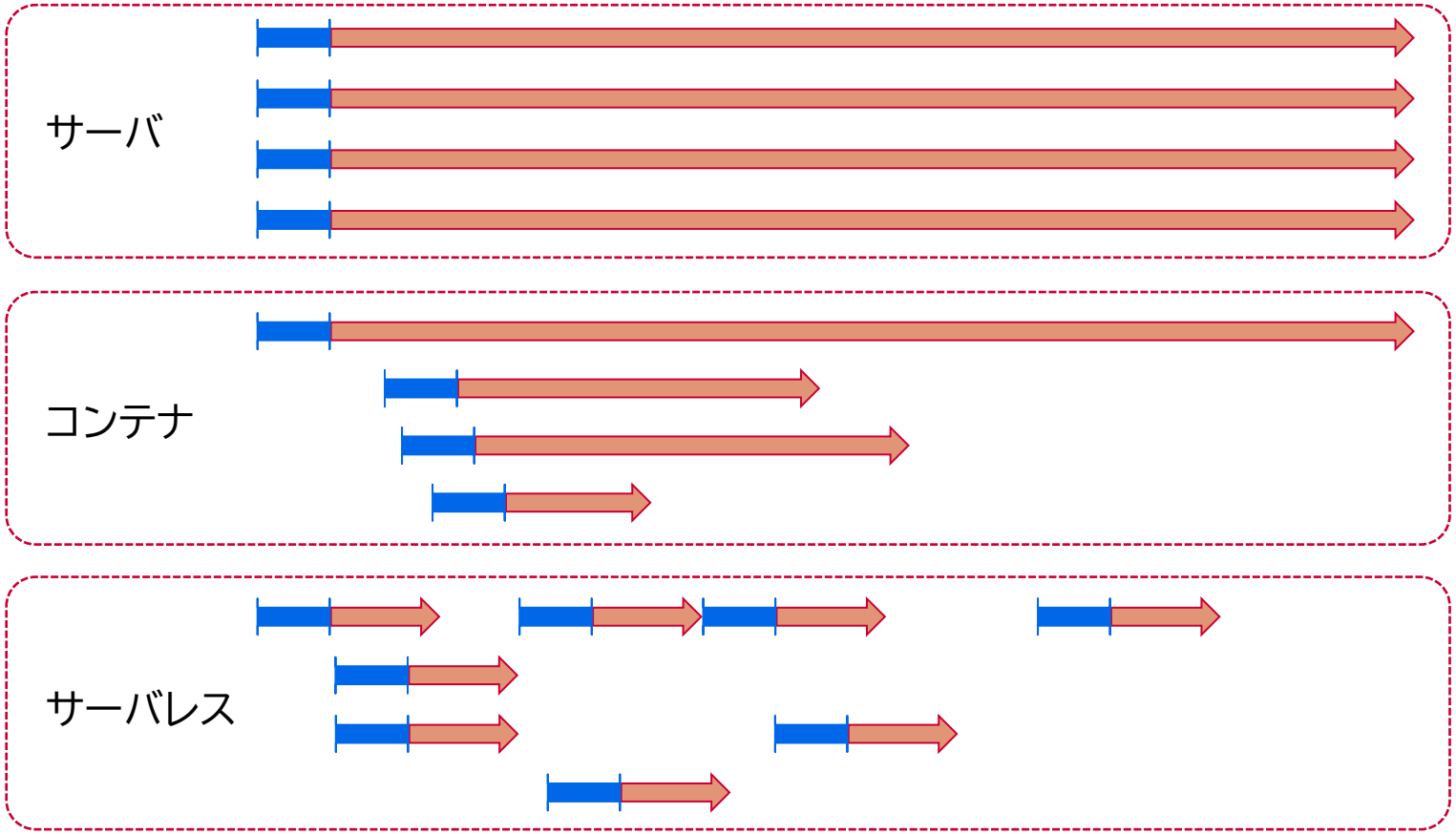
JVM 関連の起動高速化技術

Java を取り巻く環境の変化

つなごう。驚きを。幸せを。
NTT docomo Solutions

Java のスタートアップ時間の遅さがパフォーマンス上の問題として顕在化

Java App のライフサイクルは短くなる傾向であり、スタートアップ時間はパフォーマンスに影響する



起動後は動かしっぱなし
⇒スタートアップ時間はあまり課題にならない

必要な量に応じてコンテナを起動/停止
⇒コンテナ起動毎にスタートアップ時間が発生

イベントに応じて処理を起動/停止
⇒処理毎にスタートアップ時間が発生

［凡例］ アプリケーションの起動時間　アプリケーションの動作時間

JVM 関連の起動高速化技術

つなごう。驚きを。幸せを。
docomo Solutions

問題に対するアプローチは大きく分類して４種類あり

Java 実行環境の OpenJDK/GraalVM は JVM の起動時処理の高速化を進めている

技術	概要	起動時間	適用難易度
GraalVM Native Image	Java コードを静的解析＆変換してネイティブバイナリ化する	◎ 高速	× 難しい（ビルド時間長い、ライブラリ制約強い、デバッグ難易度高い）
CRaC	実行中の JVM 状態のスナップショットを取得し、それを復元することで高速化	◎ 高速	△ やや難しい（JDK やプラットフォームに制限あり、チェックポイントに対応したコード実装が必要）
CDS(AppCDS)	Java 起動時処理の一部を事前にアーカイブ化することで高速化 AOT Cache の前身機能	△ やや高速	◎ 簡単（JDK 21 以前で利用可） 制限緩い＆コード改修不要
AOT Cache	Java 起動時処理の一部を事前に学習しキャッシュ化することで高速化 CDS/AppCDS の発展機能 ※ CDS(AppCDS) とは併用不可	○ そこそこ高速	◎ 簡単（JDK 25 以降で利用可） 制限緩い＆コード改修不要

本日の対象

AppCDS

AppCDS とは

つなごう。驚きを。幸せを。
docomo Solutions

Java App のスタートアップ時間短縮を目的とした OpenJDK 機能

アプリケーションで利用するクラス情報を事前にアーカイブ化して再利用する仕組み
CDS（Class Data Sharing）をアプリケーションクラスにも拡張した機能
OpenJDK の機能なのでアプリケーションのソースコード変更なく利用できる点が魅力

機能	概要
CDS	JVM の初期化時に必要な Java クラスやメタデータを**事前にアーカイブ化**し、起動時にそれらを読み込むことで、クラスの読込・解析などの処理のオーバーヘッドを削減しスタートアップ時間短縮を実現する機能。
AppCDS	CDS を拡張し、**アプリケーションクラスもアーカイブ化**できるようにした機能。Java のコアライブラリのクラスのみでなく、アプリケーション固有のクラスも再利用できるため CDS より高いスタートアップ時間短縮効果が期待できる。

（参考）CDS 関連技術の歴史

つなごう。驚きを。幸せを。
NTT docomo Solutions

AppCDS は JDK 10 から利用可能

リリース	関連JEP等	内容	概要
JDK 5	-	CDS の導入	JVM の起動時に必要なクラスやメタデータを事前にアーカイブ化し、起動時間を短縮
JDK 10	JEP 310	AppCDS の導入	アプリケーションクラスもアーカイブに含めるように CDS 機能を拡張
JDK 12	JEP 341	デフォルト CDS アーカイブ	CDS アーカイブが**デフォルトで有効**となり、JDK ビルド時に作成された「デフォルト CDS アーカイブ」が付属（アーカイブにコアライブラリクラスが含まれる）
JDK 13	JEP 350	動的 CDS アーカイブ	アプリケーション終了時に使用されたクラスをもとに**アーカイブを動的に生成**可能
JDK 19	JDK-8261455	CDS アーカイブの自動生成	-XX:+AutoCreateSharedArchive オプションを使うことで、アーカイブが存在しない場合や異なる JDK で生成された場合に**アーカイブ自動作成**が可能

※参考：CDS and AppCDS in Hotspot - Dev.java

AppCDS 利用方法

つなごう。驚きを。幸せを。
NTT docomo Solutions

トレーニング＆アーカイブ作成 と 本番実行 の２ステップ（※）

（１）トレーニング＆アーカイブ作成

アーカイブ出力先を **-XX:ArchiveClassesAtExit** で指定し、起動完了後に App を停止させる
JVM 停止中にアーカイブファイル（app.jsa）が出力される

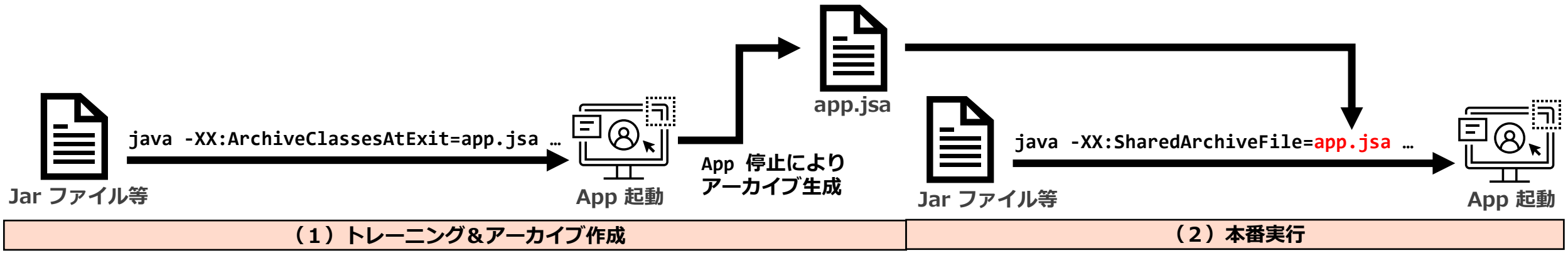
```
$ java -XX:ArchiveClassesAtExit=app.jsa -cp …
```

（２）本番実行

上記で作成したアーカイブファイルを **-XX:SharedArchiveFile** で指定して実行（スタートアップ高速化）

```
$ java -XX:SharedArchiveFile=app.jsa -cp …
```

※この利用方法（動的アーカイブ化）は JDK 13 から利用可能



AppCDS 利用時の注意点

つなごう。驚きを。幸せを。
NTT docomo Solutions

環境や設定に応じた適切なオプション選択と運用が重要

- **オプション選択について**
  - -Xshare:on はテスト環境向け。読み込みに失敗すると JVM が強制終了
  - 本番環境では -Xshare:auto の利用が推奨、アーカイブが使用できない場合は通常どおり起動
  - デフォルトでは -Xshare:auto が有効なので、特別な設定は不要
- **パフォーマンス向上の限界**
  - AppCDS はコア JDK クラスに対して効果的だが、クラス数が増えるほど効果が薄れる傾向
- **アーカイブ生成時のオーバーヘッド**
  - アーカイブ生成には時間がかかるため、開発環境での使用時には注意が必要
- **動的アーカイブとコアクラスの依存関係**
  - 動的アーカイブではコアクラスが含まれないため、CDS のデフォルトアーカイブを参照する。アーカイブが見つからない場合、パフォーマンスに影響を与える可能性がある

AppCDS 適用例（適用手順の確認）

つなごう。驚きを。幸せを。
NTT docomo Solutions

Spring PetClinic を Jar 展開して AppCDS を適用する手順例

```
# spring-petclinic ホームディレクトリで Maven Wrapper を使ってビルド
$ ./mvnw package

# Fat Jar の展開
$ java -Djarmode=tools -jar target/spring-petclinic-4.0.0-SNAPSHOT.jar ¥
    extract --destination target/application

# トレーニング＆アーカイブ作成
# -Dspring.context.exit=onRefresh を指定すると App 起動後に自動停止
$ java -Dspring.context.exit=onRefresh -XX:ArchiveClassesAtExit=archive.jsa ¥
 -jar target/application/spring-petclinic-4.0.0-SNAPSHOT.jar

# 本番実行
$ java -XX:SharedArchiveFile=archive.jsa ¥
 -jar target/application/spring-petclinic-4.0.0-SNAPSHOT.jar
…
o.s.s.petclinic.PetClinicApplication     : Started PetClinicApplication in 4.073 seconds (process
running for 4.324)
```

（参考）Spring Boot App の Jar 展開について

つなごう。驚きを。幸せを。 docomo Solutions

Fat Jar は展開してからトレーニングすること

Maven や Gradle で Spring Boot App をビルドすると成果物が **Fat Jar** ※1になる
クラウドやコンテナでも可搬性やクラスパスの煩雑さ回避の目的で Fat Jar のまま起動すると便利

ただし **Spring Boot は Fat Jar を展開しないと起動が遅くなる**ことは広く知られている※2
さらに AOT Cache や AppCDS は **Jar 展開してからトレーニングをしないと改善性能が低く**なる※3

以下のコマンドで Jar ファイルを展開してからトレーニング＆本番起動すること

```
# Spring Boot の機能で Jar を展開※4
$ java -Djarmode=tools -jar /path/to/app.jar extract --destination /path/to/extracted

# 展開した Jar を実行
$ java -jar /path/to/extracted/app.jar
```

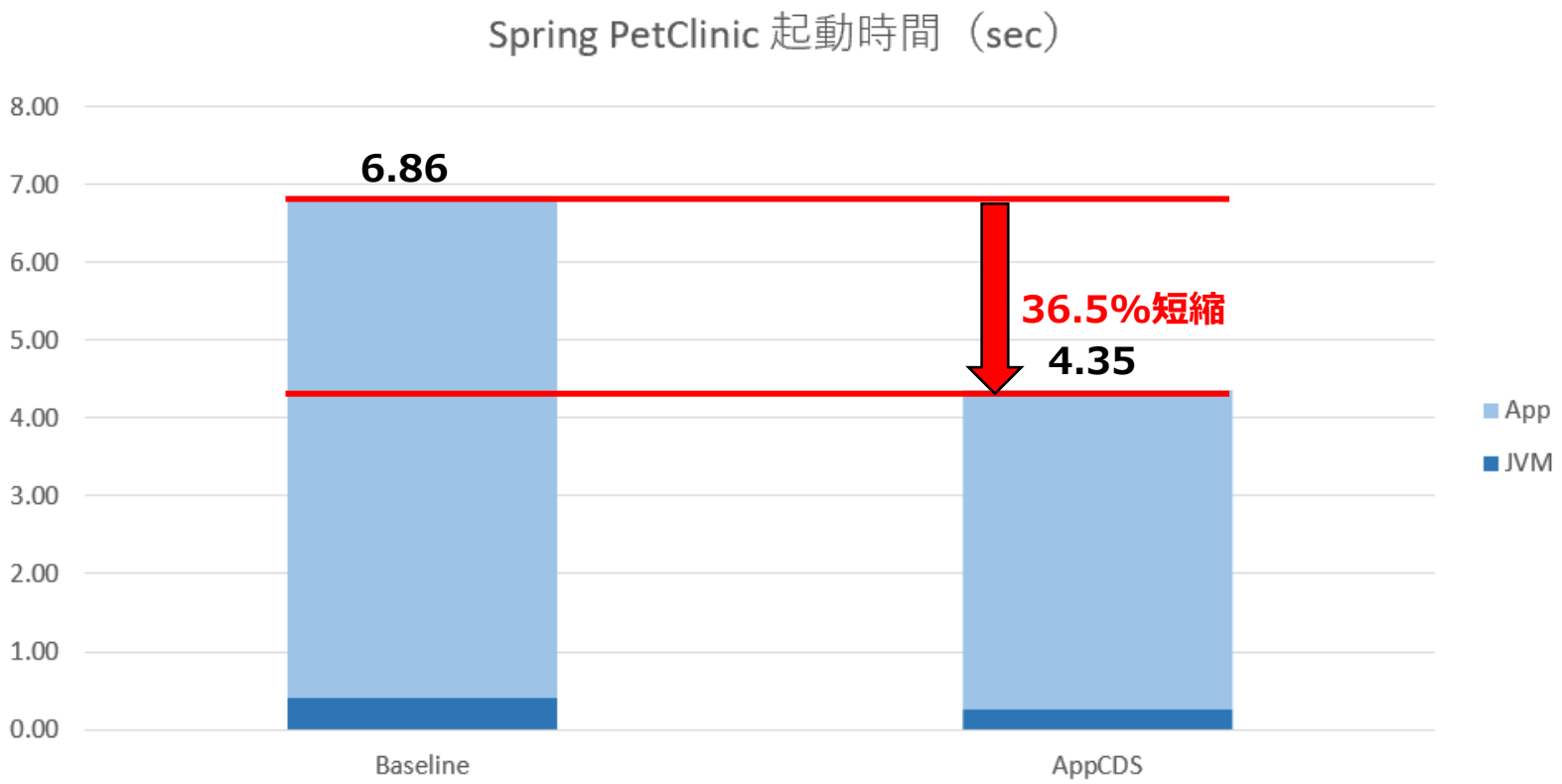
※1 アプリケーションコード、依存関係ライブラリ、ランチャーなどが含まれる自己完結型の Jar ファイルで、`java -jar app.jar` だけで実行可能
※2 Fat Jar は Spring Boot のカスタムクラスローダーを使ってクラスを読み込むためコストがかかる。Jar 展開するとカスタムクラスローダを使わなくなり、ディレクトリやワイルドカードを使用しないフラットなクラスパスを指定することと同じになり、JVM がクラスを直接読み込むため早くなる
※3 AOT Cache はカスタムクラスローダに対応していないため展開しないとすべてのクラスをキャッシュできないため改善性能が低くなる
※4 このコマンドは Spring Boot の機能によるもので、アプリケーションコードだけの app.jar と依存ライブラリ群を含む extracted/lib ディレクトリに展開する。app.jar には lib 内のライブラリを参照するマニフェストが含まれるため、これを -jar で指定するだけで起動できる

AppCDS の性能改善例

つなごう。驚きを。幸せを。
NTT docomo Solutions

Spring PetClinic へ適用するとスタートアップ時間を 37% 短縮できた

同一条件で 5 回測定した平均値を以下の通り比較した



■実行環境
- Ubuntu 24.04.4 LTS
- OpenJDK 25.0.3（Eclipse Temurin）
- Spring PetClinic (Commit 753d35c)
- CPU: AMD Ryzen 5 3550H 4コア8スレッド
- メモリ: DDR4 8GBx2

※Baseline、AppCDS の全てのパターンにおいて Jar 展開をしたうえで測定

AOT Cache

AOT Cache とは

つなごう。驚きを。幸せを。
docomo Solutions

スタートアップ＆ウォームアップ時間を早くする OpenJDK 機能

OpenJDK の Project Leyden により開発された機能で JDK 24 から利用可能。 Java 起動時の処理の一部を事前（Ahead-Of-Time:AOT）実行してキャッシュ（AOT Cache）化することで、スタートアップ＆ウォームアップ時間を早める。現在も開発が進行中。

リリース済の AOT Cache 関連 JEP 一覧

リリース	JEP	内容
JDK 24	JEP 483	AOT Cache 初回リリース（スタートアップ時間の高速化）
JDK 25	JEP 514	AOT Cache の利用方法の簡素化
JDK 25	JEP 515	メソッドプロファイリングのキャッシュ化（ウォームアップ時間の高速化）
JDK 26	JEP 516	AOT Cache の ZGC サポート対応

開発中の AOT Cache 関連 JEP

JEP	内容
JEP draft	JIT コンパイルにより最適化されたコードのキャッシュ化

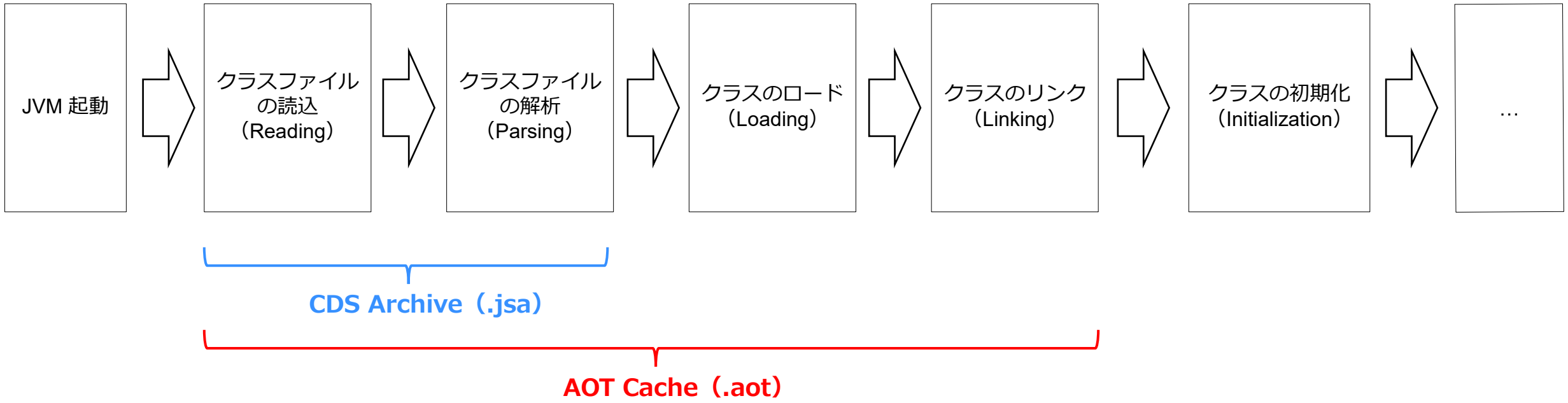
AOT Cache の仕組み

つなごう。驚きを。幸せを。
NTT docomo Solutions

JVM 起動時のクラスのロード・リンクまでの実行結果の状態をキャッシュ化する

JVM 起動時にロード＆リンクされたクラスの状態をキャッシュとして保存し、
次回実行時にキャッシュを用いることで即座に同じ状態に至るため、スタートアップ時間を短縮できる。
AOT Cache は CDS と比較して保存する状態の範囲が拡張されており、CDS を正当進化させた機能といえる。

JVM 起動時におけるクラス読込の関連処理順序（イメージ）



AOT Cache 利用方法

つなごう。驚きを。幸せを。
NTT docomo Solutions

トレーニング＆アセンブリ と 本番実行 の２ステップ

（１）トレーニング＆アセンブリ

キャッシュ出力先を **–XX:AOTCacheOutput** ※で指定し、起動完了後に App を停止させる
JVM 停止処理でキャッシュがアセンブリされ、キャッシュファイル（app.aot）が出力される

```
$ java -XX:AOTCacheOutput=app.aot –cp …
```

（２）本番実行

上記で作成したキャッシュファイルを **–XX:AOTCache** で指定して実行（スタートアップ高速化）

```
$ java -XX:AOTCache=app.aot –cp …
```

【再掲】AppCDS 利用方法

（１）トレーニング＆アーカイブ作成

アーカイブ出力先を **-XX:ArchiveClassesAtExit** で指定し、起動完了後に App を停止させる
JVM 停止中にアーカイブファイル（app.jsa）が出力される

```
$ java -XX:ArchiveClassesAtExit=app.jsa –cp …
```

（２）本番実行

上記で作成したアーカイブファイルを **-XX:SharedArchiveFile** で指定して実行（スタートアップ高速化）

```
$ java -XX:SharedArchiveFile=app.jsa –cp …
```


※このオプションは JDK 25 から利用可能

AOT Cache 利用時の注意点

つなごう。驚きを。幸せを。
NTT docomo Solutions

トレーニングと本番実行の環境は一致させること

AOT Cache は（アプリケーションに変更がなければ）起動時の処理は毎回同じことをするという前提
トレーニングと本番で環境や実行コードに違いがあると改善効果が小さくなるか通常起動にフォールバックする
以下の制約に従って、トレーニングと本番実行の環境は一致させること

- **JRE/JDK や OS などの環境を変更しない**
  - JRE/JDK はバイナリレベルで変更不可
- **クラスパスやモジュールオプションは変更禁止**
  - メインクラスは変更OK
- **JVMTI エージェント（クラス書換やクラスロード操作 API）は利用不可**
  - 監視系エージェントで使っていないか注意
- **GC 方式は例外的に変更可能**
  - ZGC 対応は JDK 26 から（JEP 516）

（参考）Spring Boot の AOT Cache 利用

つなごう。驚きを。幸せを。
docomo Solutions

Fat Jar は展開してからトレーニングすること

Maven や Gradle で Spring Boot App をビルドすると成果物が **Fat Jar** ※1になる
クラウドやコンテナでも可搬性やクラスパスの煩雑さ回避の目的で Fat Jar のまま起動すると便利

ただし **Spring Boot は Fat Jar を展開しないと起動が遅くなる**ことは広く知られている※2
さらに AOT Cache や AppCDS は **Jar 展開してからトレーニングをしないと改善性能が低く**なる※3

以下のコマンドで Jar ファイルを展開してからトレーニング＆本番起動すること

```
# Spring Boot の機能で Jar を展開※4
$ java -Djarmode=tools -jar /path/to/app.jar extract --destination /path/to/extracted

# 展開した Jar を実行
$ java -jar /path/to/extracted/app.jar
```

※1 アプリケーションコード、依存関係ライブラリ、ランチャーなどが含まれる自己完結型の Jar ファイルで、`java -jar app.jar` だけで実行可能
※2 Fat Jar は Spring Boot のカスタムクラスローダーを使ってクラスを読み込むためコストがかかる。Jar 展開するとカスタムクラスローダを使わなくなり、ディレクトリやワイルドカードを使用しないフラットなクラスパスを指定することと同じになり、JVM がクラスを直接読み込むため早くなる
※3 AOT Cache はカスタムクラスローダに対応していないため展開しないとすべてのクラスをキャッシュできないため改善性能が低くなる
※4 このコマンドは Spring Boot の機能によるもので、アプリケーションコードだけの app.jar と依存ライブラリ群を含む extracted/lib ディレクトリに展開する。app.jar には lib 内のライブラリを参照するマニフェストが含まれるため、これを -jar で指定するだけで起動できる

AOT Cache 適用例（適用手順の確認）

つなごう。驚きを。幸せを。
NTT docomo Solutions

Spring PetClinic を Jar 展開して AOT Cache を適用する手順例

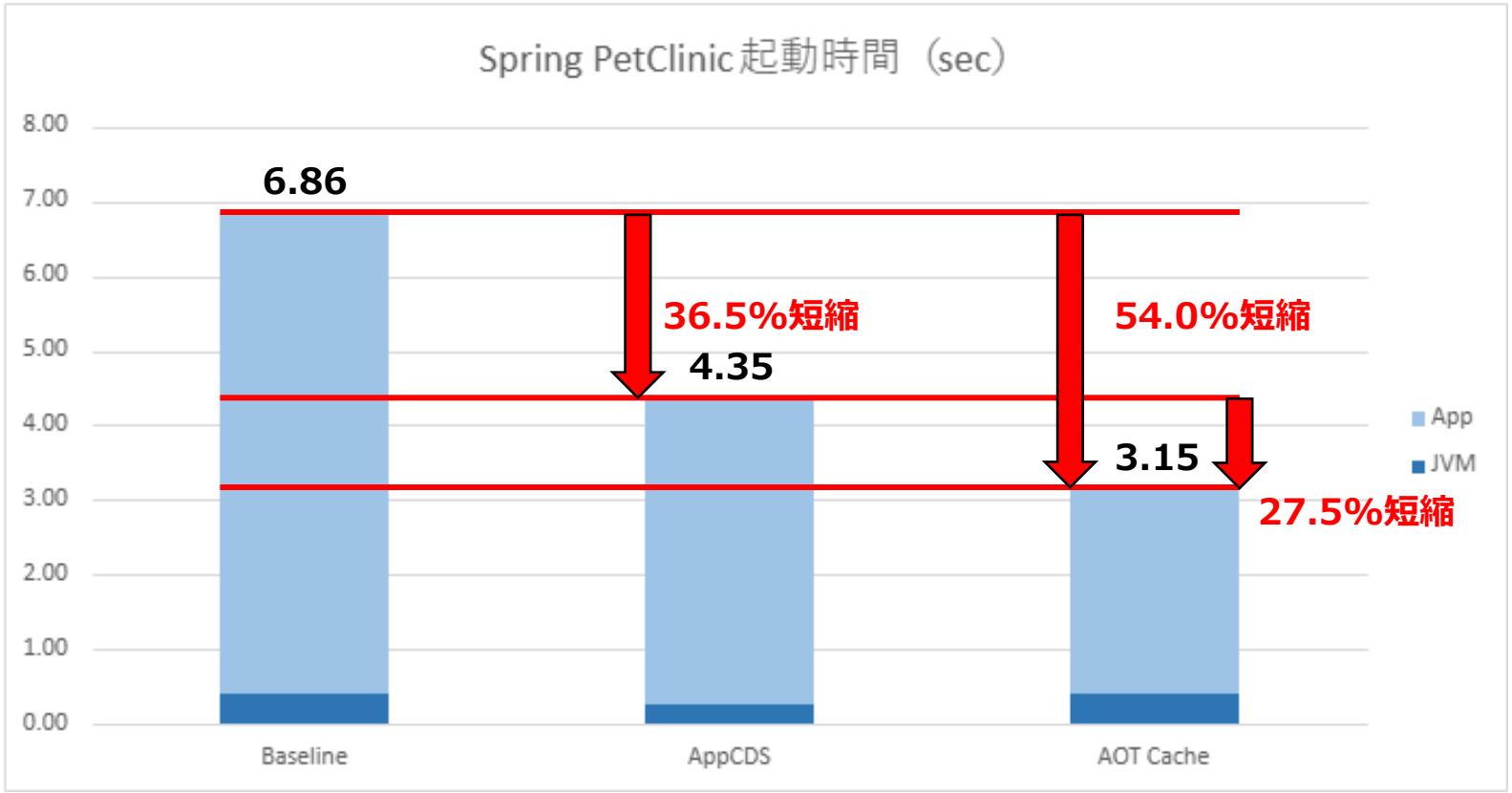
```
# spring-petclinic ホームディレクトリで Maven Wrapper を使ってビルド
$ ./mvnw package

# Fat Jar の展開
$ java -Djarmode=tools -jar target/spring-petclinic-4.0.0-SNAPSHOT.jar ¥
    extract --destination target/application

# トレーニング＆アセンブリ
# -Dspring.context.exit=onRefresh を指定すると App 起動後に自動停止
$ java -Dspring.context.exit=onRefresh -XX:AOTCacheOutput=app.aot ¥
    -jar target/application/spring-petclinic-4.0.0-SNAPSHOT.jar

# 本番実行
$ java -XX:AOTCache=app.aot -jar target/application/spring-petclinic-4.0.0-SNAPSHOT.jar
…
2026-06-03T13:20:05.748+09:00  INFO 3021735 --- [           main]
o.s.s.petclinic.PetClinicApplication     : Started PetClinicApplication in 2.655 seconds (process
running for 3.054)
```

AOT Cache の性能改善例

つなごう。驚きを。幸せを。
NTT docomo Solutions

Spring PetClinic へ適用するとスタートアップ時間を 54% 短縮できた

AppCDS と比較しても 27.5% ほどスタートアップ時間を短縮できている



■実行環境
- Ubuntu 24.04.4 LTS
- OpenJDK 25.0.3（Eclipse Temurin）
- Spring PetClinic (Commit 753d35c)
- CPU: AMD Ryzen 5 3550H 4コア8スレッド
- メモリ: DDR4 8GBx2

※Baseline、AppCDS、AOT Cache の全てのパターンにおいて Jar 展開をしたうえで測定

（参考）Spring AOT の利用

つなごう。驚きを。幸せを。
NTT docomo Solutions

スタートアップ高速化＆メモリフットプリント削減できるフレームワークの機能

Spring AOT は **アプリケーションコードを静的解析して最適な形へ事前変換する機能**
簡単に言えばリフレクションなどの動的要素を削除したコードへ変換することでスタートアップを高速化する

AOT Cache と Spring AOT は全く別の機能であるため併用可能
AOT Cache と Spring AOT を組み合わせることでさらにスタートアップを高速化できる

ただし、もともと GraalVM Native Image をサポートするために実装された機能であり、
App で利用するフレームワークやライブラリが Spring AOT に対応していないと利用不可なため、
適用難易度は高い

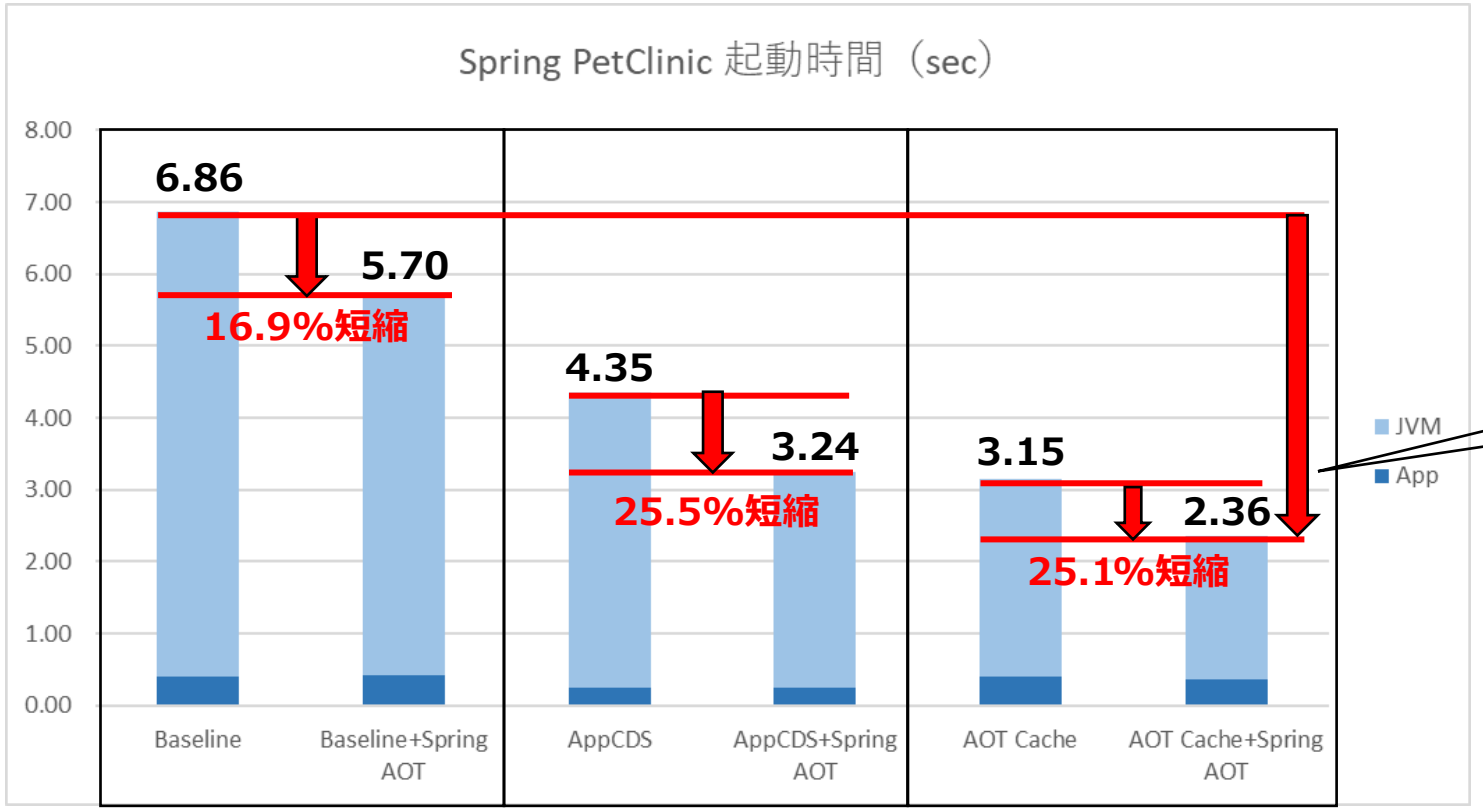
Spring AOT を利用するには以下の２つを設定する
1. Spring Boot Maven/Gradle Plugin で AOT 処理のプラグインを有効化してビルド
    1. （Maven）公式ドキュメント を参照
    2. （Gradle）公式ドキュメント を参照
2. システムプロパティ-Dspring.aot.enabled=true を指定して App を実行

（参考）Spring AOT の性能改善例

つなごう。驚きを。幸せを。
docomo Solutions

Spring AOT 適用によりいずれのケースでも性能改善が見られる

Spring AOT 適用によりスタートアップ時間を平均 22.5% 短縮できている



■実行環境
- Ubuntu 24.04.4 LTS
- OpenJDK 25.0.3（Eclipse Temurin）
- Spring PetClinic (Commit 753d35c)
- CPU: AMD Ryzen 5 3550H 4コア8スレッド
- メモリ: DDR4 8GBx2

Baseline（Spring AOTなし） から比較すると**約 65.6% の短縮**

※Baseline、AppCDS、AOT Cache の全てのパターンにおいて Jar 展開をしたうえで測定

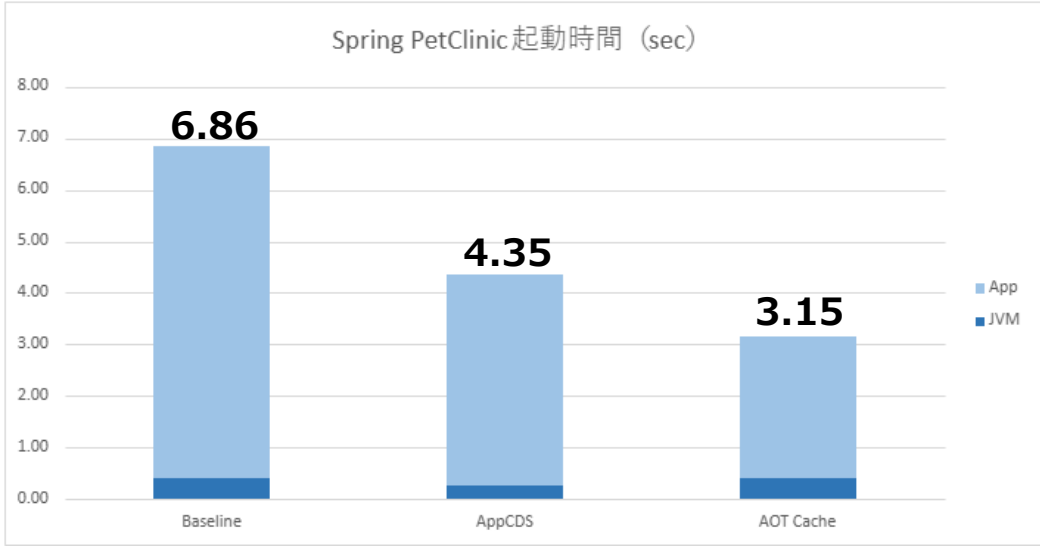
まとめ

つなごう。驚きを。幸せを。
docomo Solutions

AppCDS や AOT Cache を利用することで容易にスタートアップ時間の短縮を実現

- AppCDS や AOT Cache は OpenJDK 標準機能
  - アプリケーションのソースコードを変更しなくても利用できる点が魅力
- AppCDS、AOT Cache どちらも使い方は２ステップ
  - （１）トレーニングしてアーカイブやキャッシュファイルを生成
  - （２）生成したファイルを用いて本番実行
- AppCDS で 36.5%、AOT Cache で 54.0% のスタートアップ時間短縮を確認

- すぐにスタートアップ時間短縮を実現したいなら…
  - JDK 21 以前なら AppCDS を利用
  - JDK 25 以降では AOT Cache を利用
  - Spring Boot アプリケーションならば以下も有効
    - Fat Jar の展開
    - Spring AOT の適用



Q&A