JDK 11からJDK 17に 移行する際の注意点について

2022年9月14日 第1.0版

日本電気株式会社

スマートエッジ事業部門

\Orchestrating a brighter world

NECは、安全・安心・公平・効率という社会価値を創造し、
誰もが人間性を十分に発揮できる持続可能な社会の実現を目指します。

目次

1. はじめに
2. JDK 11とJDK 17の非互換項目
3. 参考情報

はじめに

本書に関する注意事項
背景と目的
表記ルール

本書に関する注意事項

1. WebOTXは日本電気株式会社の商標登録です。
2. OracleとJavaは、Oracle Corporation及びその子会社、関連会社の米国及びその他の国における商標登録です。
3. 本書の一部または全部を無断に転載することを禁じます。
4. 本書に関しては将来予告なしに変更することがあります。
5. 弊社の許可なく複製、改変することを禁じます。
6. 対処した結果の影響については、弊社は一切責任を負いません。

背景と目的

◆ 背景

2022年9月に、Java SE Development Kit 17(JDK 17)に対応したWebOTX Application Server V11.1をリリースしました。Oracle社によれば、Java SE Development Kit 11 (Oracle JDK 11)の公式アップデートの提供は、2026年9月で終了予定です。それ以降は、Oracle JDK 11 で不具合が発覚したとしても修正されることは保証されません。

JDK 11 のサポート期間は JDK 17 より短いため、長期運用を目的とする場合は JDK 17 への切り替えが必要となります。

しかし、JDK 11 と JDK 17 間では、互換性を損ねる変更が行われているため、移行の際は注意が必要です。

◆ 目的と対象読者

本資料では、アプリケーションの移行や運用管理における注意点について記載しています。

WebOTX Application Server V10.4の利用者が、V11.1系以降の導入を契機にアプリケーションの動作環境をJDK 17 に変更する場合に有用です。

表記ルール

◆ パスの表記

本資料では、パスの表記を"/"としています。
Windows系の場合は"/"を"¥"に読み替えてください。

◆ インストールディレクトリの表記

プレースホルダ	説明
<code>\${JAVA_HOME}</code>	Javaのインストールディレクトリ

JDK 11とJDK 17の非互換項目

1. APIの削除
2. デフォルトでのJDK内部の強力なカプセル化
3. SecurityManager非推奨の影響
4. Hiddenクラスの実装
5. NumberFormatクラスの扱い
6. java.net.SocketとServerSocket APIの新実装
7. ルート証明書の削除
8. 非推奨になったトレース・フラグの廃止および同等の統合ログイン機能への置換え
9. Nashorn JavaScriptエンジンの削除
10. バイアス・ロックの無効化および非推奨
11. JSSE 1.0 アプリケーションとの互換性のためにのみ使用されていた内部 com.sun.net.sslパッケージの削除
12. VMオプション(-XX:+AggressiveOpts)の削除

APIの削除 (1/4)

◆ 現象

JDK 12から17で、下記のAPIが削除されています。この影響で、これらのAPIを使用するアプリケーションをコンパイル時はシンボルの解決に失敗するエラーが発生し、実行した場合は `java.lang.ClassNotFoundException`, `java.lang.NoSuchFieldException`, `java.lang.NoSuchMethodException`が発生します。

パッケージ

`java.rmi.activation`

`java.security.acl`

クラス

`java.util.jar.Pack200`

コンストラクタ

`javax.tools.ToolProvider.<init>()`

`java.lang.invoke.ConstantBootstraps.<init>()`

`java.lang.reflect.Modifier.<init>()`

フィールド

`java.management.remote.rmi.RMIConnectorServer.CREDENTIAL_TYPES`

メソッド

`java.lang.Runtime.traceInstructions(boolean)`

`java.lang.Runtime.traceMethodCalls(boolean)`

`java.io.InputStream.finalize()`

`java.io.OutputStream.finalize()`

`java.util.zip.Deflater.finalize()`

`java.util.zip.Inflater.finalize()`

`java.util.zip.ZipFile.finalize()`

APIの削除 (2/4)

◆ 対応

パッケージ	対応
java.rmi.activation	Javaプラットフォームでは、RMIアクティブ化メカニズムに代わるものはありません。 RMIアクティブ化のユーザーは、アプリケーションを他の実装方法に移行する必要があります。
java.security.acl	これは1.2以降java.security.Policyおよび関連するクラスに置き換えられました。 他の実装方法に移行する必要があります。

クラス	対応
java.util.jar.Pack200	代替となるAPIは存在しません。 他の実装方法に移行する必要があります。

APIの削除 (3/4)

◆ 対処

コンストラクタ	対応
javax.tools.ToolProvider.<init>() java.lang.invoke.ConstantBootstraps.<init>()	staticメソッドのみを定義しているため、コンストラクタは不要です。
java.lang.reflect.Modifier.<init>()	isXXXメソッド群を定義しているクラスのため、コンストラクタは不要です。
フィールド	対応
java.management.remote.rmi.RMIConnectorServer.CREDENTIAL_TYPES	代わりにRMIConnectorServer.CREDENTIALS_FILTER_PATTERNを「フィルタ・パターン」文字列で使用してください。
メソッド	対応
java.lang.Runtime.traceInstructions(boolean) java.lang.Runtime.traceMethodCalls(boolean)	このメソッドは、命令のトレースを制御することを目的としていました。 JVM固有のトレース・メカニズムに置き換えられました。 置き換えられたメカニズムに対応する実装方法に移行する必要があります。

APIの削除 (4/4)

◆ 対処

メソッド	対応
java.io.FileInputStream.finalize() java.io.FileOutputStream.finalize() java.util.zip.Deflater.finalize() java.util.zip.Inflater.finalize() java.util.zip.ZipFile.finalize()	従来のJDK実装では、リソースのクローズ漏れを防ぐためにfinalizeメソッドでcloseが呼び出されていましたが、GCの延長でfinalizeメソッドが呼び出されるかはJVMの実装依存であり、全ての環境で呼び出される保証は無いため、JDK 12でfinalizeメソッドは削除されました。このため、リソースのクローズ処理をfinalizeメソッドに任せていた場合は、try-catch-resources文やcloseメソッドで適切にリソースをクローズする必要があります。また、これらのクラスを継承したサブクラスでfinalizeメソッドをオーバーライドしている場合は、finalizeメソッドを削除するか、java.lang.Throwable をスローするように宣言する必要があります。これを行わない場合は、コンパイルエラーが発生します。

◆ jdeprscan

JDK 17で削除対象としてマークされているAPIのリストを取得することができます。

実行例

```
${JAVA_HOME}/bin/jdeprscan --release 17 -l --for-removal <ディレクトリ or jarファイル or classファイル>
```

デフォルトでのJDK内部の強力なカプセル化

◆ 現象

JDK 17で強力なカプセル化を弱める--illegal-accessオプションがサポートされなくなりました。この影響で--illegal-accessオプションを使用時、以下の警告が出力されます。

```
OpenJDK 64-Bit Server VM warning: Ignoring option --illegal-access={permit/warn/debug/none}; support was removed in 17.0
```

また、内部APIにアクセスしようとする時、以下の例外が発生します。

```
java.lang.reflect.InaccessibleObjectException: Unable to make field private final {type} accessible: module java.base does not "opens {module}" to unnamed module {module}
```

◆ 対処

--illegal-accessオプションを使用せず、内部APIの呼び出しをサポートしているAPIの呼び出しに変更してください。

SecurityManager非推奨の影響(1/2)

◆ 現象①

JDK 17でSecurityManagerが非推奨になりました。この影響でアプリケーションがSecurityManagerに依存し有効化している場合、起動時に以下の警告が出力されます。

```
WARNING: A command line option has enabled the Security Manager  
WARNING: The Security Manager is deprecated and will be removed in a future release
```

コマンド実行時に-Djava.security.managerオプションを使用し、SecurityManagerが有効化することができます。

実行例

```
${JAVA_HOME}/bin/java -Djava.security.manager <クラス名>  
${JAVA_HOME}/bin/java -Djava.security.manager="" <クラス名>  
${JAVA_HOME}/bin/java -Djava.security.manager=default <クラス名>  
${JAVA_HOME}/bin/java -Djava.security.manager=com.foo.bar.Server <クラス名>
```

◆ 対処①

将来的に廃止される可能性があるため、SecurityManagerを使用しない実装への変更を検討してください。SecurityManagerを使い続ける場合は-Djava.security.manager=allowオプションを付けることで警告を非表示にすることができます。

SecurityManager非推奨の影響(2/2)

◆ 現象②

前述と同様の影響で、アプリケーション実行中にSystem::setSecurityManagerが呼び出された場合は以下のような警告が出力されます。

```
WARNING: A terminally deprecated method in java.lang.System has been called
WARNING: System::setSecurityManager has been called by com.foo.bar.Server (file:/tmp/foobarserver/thing.jar)
WARNING: Please consider reporting this to the maintainers of com.foo.bar.Server
WARNING: System::setSecurityManager will be removed in a future release
```

◆ 対処②

将来的にSecurityManagerが無効化される可能性があるため、SecurityManagerを使用しない実装への変更を検討してください。すぐに移行ができない場合は、JDK 17時点で振る舞いに変更はないため、警告表示を無視してください。

Hiddenクラスの実装

◆ 現象

JDK 17で、`sun.misc.Unsafe.defineAnonymousClass()`メソッドが削除されました。

この影響で、隠しクラスの実装に本メソッドを使用している場合は、コンパイル・実行時にエラーを引き起します。

◆ 対処

本メソッドを以下のメソッドに置き換えてください。

- `java.lang.invoke.MethodHandles.Lookup::defineHiddenClass`
- `java.lang.invoke.MethodHandles.Lookup::defineHiddenClassWithClassData`

NumberFormatクラスの扱い (1/2)

◆ 現象

JDK 14でUnicodeロケール拡張に対応した、数値書式と会計書式がサポートされました。

この影響でNumberFormat.getCurrencyInstance(Locale);を呼び出す際、Localインスタンスのロケール拡張情報 (Locale.UNICODE_LOCALE_EXTENSIONフィールド) に"cf-account"が設定されている場合は出力される書式に変更があります。

実装例

```
Locale locale = new Builder().setLanguage("ja")
    .setRegion("JP")
    .setScript("Latn")
    .setExtension(Locale.UNICODE_LOCALE_EXTENSION, "cf-account")
    .build();
NumberFormat fmt = NumberFormat.getCurrencyInstance(locale);
System.out.println(fmt.format(-1234.56));
```

出力 (JDK 11)

- ¥ 1,235

出力 (JDK 17)

(¥ 1,235)

NumberFormatクラスの扱い (2/2)

◆ 対処

JDK 11と同等の結果を出力させる場合は、Localインスタンスのロケール拡張情報 (Locale.UNICODE_LOCALE_EXTENSIONフィールド) に設定する値を”**cf-account**”から”**cf-standard**”に変更してください。

実装例

```
Locale locale = new Builder().setLanguage("ja")
    .setRegion("JP")
    .setScript("Latn")
    .setExtension(Locale.UNICODE_LOCALE_EXTENSION, "cf-standard")
    .build();
```

出力フォーマットスタイルは、[「UNICODE LOCALE DATA MARKUP LANGUAGE \(LDML\) PART 3: NUMBERS – Currency Format](#)」を参照してください。

java.net.SocketとServerSocket APIの新実装

◆ 現象

JDK 13で、java.net.Socket, java.net.ServerSocketの標準実装が新しい実装に置き換わりました。この影響で旧実装と比較しパフォーマンスや振る舞いが異なる可能性があります。現段階で確認されている振る舞いの違いは、「[JEP 353](#)」を参照してください。

◆ 対処

従来通りのパフォーマンスや振る舞いを期待する場合は、コマンド実行時に旧実装の使用を指定する-Djdk.net.usePlainSocketImpl オプションを付与してください。しかし、本オプションを使用しても振る舞いが異なるコーナーケースも存在します。

実行例

```
${JAVA_HOME}/bin/java -Djdk.net.usePlainSocketImpl <クラス名>
```

ルート証明書の削除

◆ 現象

JDK 13, 16で以下の認証局証明書（表記：エイリアス名）がcacertsキーストアから削除されました。この影響で削除された証明書を使用する場合は、予め証明書を用意する必要があります。

JDK 16で削除されたルート証明書	JDK 13で削除されたルート証明書
thawtepremiumserverca	deutschetelekomrootca2
verisignclass2g2ca	certplusclass2primaryca
verisignclass3ca	certplusclass3pprimaryca
verisignclass3g2ca	utnuserfirstclientauthemailca
verisigntsaca	utnuserfirsthardwareca

◆ 対処

使用する証明書を信頼する認証局から証明書を取得し、トラストストアに格納してください。

非推奨になったトレース・フラグの廃止および同等の統合ロギング機能への置換え (1/2)

◆ 現象 (1/2)

JDK 11では、本トレース・フラグ使用時に警告が表示されていました。

JDK 17では、以下の廃止されたトレース・フラグを使用時、不正なオプションとして扱われます。

トレースフラグ	ログレベル	タグ
PrintSharedSpaces	Info	cds
TraceBiasedLocking	Info	biasedlocking
TraceClassLoading	Info	class, load
TraceClassLoadingPreorder	Debug	class, preorder
TraceClassPaths	Info	class, path
TraceClassResolution	Debug	class, resolve
TraceClassUnloading	Info	class, unload
TraceExceptions	Info	exceptions
TraceInvokeDynamic	Debug	methodhandles, indy
TraceLoaderConstraints	Info	class, loader, constraints

非推奨になったトレース・フラグの廃止および同等の統合ロギング機能への置換え (2/2)

◆ 現象 (2/2)

トレースフラグ	ログレベル	タグ
TraceMonitorInflation	Trace	monitorinflation
TraceSafepointCleanupTime	Info	safepoint, cleanup
TraceJVMTIObjectTagging	Debug	jvmti, objecttagging
TraceRedefineClasses	Info	redefine, class
PrintJNIResolving	Debug	jni, resolve

◆ 対処

コマンド実行時に-Xlogオプションにてトレース・フラグの使用を同等の統合ロギングに明示的に置き換えてください。

実行例(JDK 11)

```
${JAVA_HOME}/bin/java -XX:+PrintSharedSpaces <クラス名>
```

実行例(JDK 17)

```
${JAVA_HOME}/bin/java -Xlog:cds=info <クラス名>
```

Nashorn JavaScriptエンジンの削除

◆ 現象

JDK 15で、Nashorn JavaScript スクリプトエンジンとAPI, jjsツールが削除されました。

この影響で、Nashorn JavaScriptを使用しているアプリケーションは改修する必要があります。

削除されたJDKモジュールとの関係性は、以下の通りです。

- `jdk.scripting.nashorn` : Nashorn JavaScriptスクリプトエンジンとAPI
- `jdk.scripting.nashorn.shell` : jjsツール

◆ 対処

Nashorn JavaScriptエンジンの代替として有力な候補の一つにGraalJSが挙げられます。

JavaScriptエンジンをNashorn JavaScriptエンジンから別のエンジン(GraalJS等)への移行を検討してください。

バイアス・ロックの無効化および非推奨

◆ 現象

JDK 15でバイアス・ロックがデフォルトで無効化されました。

この影響でデフォルトで有効化されていることを想定しているアプリケーションは、実行時にエラーを引き起こします。

一部のアプリケーションでは、バイアス・ロックを無効にすることでパフォーマンスが低下する可能性があります。

◆ 対処

バイアス・ロックの有効状態を期待する場合は、コマンド実行時にバイアス・ロックを有効化する `-XX:+UseBiasedLocking` オプションを設定してください。

実行例

```
${JAVA_HOME}/bin/java -XX:+UseBiasedLocking <クラス名>
```

JSSE 1.0 アプリケーションとの互換性のためにのみ使用されていた内部 `com.sun.net.ssl` パッケージの削除

◆ 現象

JDK 13で、セキュアな通信を可能とするJSSE(Java Secure Socket Extension)との互換性のために、使用されていた内部 `com.sun.net.ssl` パッケージが削除されました。

この影響で本パッケージを使用しているアプリケーションは、実行時にエラーを引き起こします。

◆ 対処

`javax.net.ssl`パッケージで提供されている`HostNameVerifier`、`KeyManager`、`TrustManager`などの標準的な代替 API に置き換えてください。

VMオプション(-XX:+AggressiveOpts)の削除

◆ 現象

JDK 13で、将来のリリースでデフォルトになりそうな最適化フラグを有効にするVMオプション(-XX:+AggressiveOpts)がサポートされなくなりました。

この影響で、コマンド実行時に本オプションを使用している場合は、VM初期化時にエラーを引き起こします。

◆ 対処

コマンド実行時に、必要となるフラグ毎にオプションで指定（有効化）してください。

JDK 11で本VMオプション利用時に有効化されたフラグ一覧を、VMオプション(-XX:+PrintFlagsFinal)を付与することで取得可能です。

実行例(JDK 11)

```
${JAVA_HOME}/bin/java -XX:+PrintFlagsFinal -XX:+AggressiveOpts
```

参考情報

参考情報
改版履歴

参考情報

- ◆ 「Java Platform Standard Edition ~Oracle JDK 移行ガイド リリース17~」
<https://docs.oracle.com/javase/jp/17/migrate/index.html>

改版履歴

版数	発行日	改版内容
1.0	2022/9/14	新規作成

\Orchestrating a brighter world

NEC