HITACHI
Inspire the Next

開発・運用時のガイド [Windows]

JDK8 への移行に伴う留意点

2015.10
October

はじめに

本書は、開発・運用フェーズで使用するドキュメントとして、Java™ Development Kit 8 への移行に伴う留意点について記述しています。

１．対象とする読者

本書は、Java™ Development Kit 8 を使用し、システムを設計・構築・運用する立場にある方を対象としています。

■英略語の表記

本書では、英略語を次のように表記しています。

表記	製品名
Java	Java™
Java EE	Java™ Platform, Enterprise Edition
Java SE	Java™ Platform, Standard Edition
JDK	Java™ Development Kit
	JDK™

■発行元

株式会社日立製作所 情報・通信システム社 IT プラットフォーム事業本部

目　次

1 JDK8 への移行に伴う留意事項

JDK8 への移行に伴う留意事項を説明します。

本章の構成

1.1 JDK8 への移行に伴う留意事項

1.1 JDK8 への移行に伴う留意事項

(1)ロケール依存サービスについて

Java SE 8 から，ロケールに依存するサービスを指定できるようになりました。

ロケール依存サービスの指定には java.locale.providers プロパティを使用します。

指定可能なロケール依存サービスと java.locale.providers プロパティの詳細は，下記の URL を参照してください。

http://docs.oracle.com/javase/jp/8/docs/technotes/guides/intl/enhancements.8.html

ロケール依存サービスの指定は，JavaVM オプションを用いて Java EE サーバの起動時に行います。この指定により，Java EE アプリケーションを含む Java EE サーバ全体のロケールが変わってしまうことがあるため，ロケール依存サービスを指定する場合は影響範囲に注意し，運用に問題のない範囲で行ってください。また，Java EE サーバ起動後に Java EE サーバや Java EE アプリケーションからロケール依存サービスの指定を変更することはできません。

なお，ロケール依存サービスの指定によっては，ロケールに依存する表記や処理に影響する場合があります。例えば，ロケール依存サービスの指定によっては，Java EE アプリケーションで明示的にロケールを指定していても，指定とは異なるロケール依存サービスが選択され，日付・時刻・通貨等の表記や処理が変更される場合があります。

以下に，ロケール依存サービスによる影響個所の確認観点の例を示します。

(a)

ロケール依存サービスは java.util.spi.LocaleServiceProvider クラスや java.util.spi.LocaleServiceProvider クラスのサブクラスにより実装されます。そのため，これらのクラスのカスタマイズ対象のロケールに影響する場合があります。カスタマイズ対象のロケールを観点として，Java プログラム内にロケール依存サービスによる影響個所がないか確認してください。

java.util.spi.LocaleServiceProvider の詳細は，下記の URL を参照してください。

https://docs.oracle.com/javase/jp/8/api/java/util/spi/LocaleServiceProvider.html

(b)

ロケール依存サービスの変更による影響がある個所では，ロケールを明示的に指定している可能性があります。ロケールの指定に使用する定数(ロケール定数)を観点として，Java プログラム内にロケール依存サービスによる影響個所がないか確認してください。

よく使われるロケール定数の詳細は，下記の URL を参照してください。

https://docs.oracle.com/javase/jp/8/api/java/util/Locale.html

例えば，java.util.Locale.JAPAN は国を指定するのに使用する定数です。

(2) クラスファイルのバージョンについて

Java SE 8 からクラスファイルのバージョンが 52 になりました。

これにより，ソースコードでラムダ式を利用した場合に，javac コマンドが出力する「invokedynamic

バイトコード」を出力するようになりました。

クラスファイル変換等でクラスファイルを出力する場合，出力するクラスファイルのバージョンとクラスファイルのフォーマットが一致している必要があります。一致していない場合は，クラスロード時に java.lang.VerifyError が発生するため，クラスファイル変換等の新仕様対応を行ってください。
なお，クラスファイルのバージョンはバイナリエディタや javap コマンドを使用してクラスファイル内の major_version から確認してください。

また, javac コマンドのクロスコンパイルオプション(target/source)を用いることで指定したバージョンのクラスファイルを作成することもできます。
ただし，その場合はクラスバージョンに対応する Java SE の機能までしか使用することはできません。

(3) 親プロセスと子プロセスとで標準(エラー)入出力を共有している場合の親プロセス終了について

Java SE 8 では，子プロセスが親プロセスと標準(エラー)入出力を共有していた場合，コマンドプロンプトを閉じる，または Ctrl+C キーを入力すると子プロセスが終了するようになりました。

なお，親プロセスと子プロセスで標準(エラー)入出力を共有する方法には，下記の 2 つがあります。java プログラムが下記のいずれかに該当する処理を行っている場合，上記の挙動にご注意ください。

(a)java.lang.ProcessBuilder クラスの API を利用する
以下の API で親プロセスと子プロセスで，標準(エラー)入出力のどれかひとつでも共有する
- inheritIO()
- redirectInput()
- redirectError()
- redirectOutput()
- redirectErrorStream()

(b)java.lang.Runtime.exec()で子プロセスを開始する

(4) split()の仕様変更について

Java SE 8 では以下の API の仕様が変更されました。
-java.lang.String.split()
-java.util.regex.Pattern.split()
なお，シグネチャが異なる同名のメソッドも全て該当します。

split メソッドは指定された正規表現に一致する位置で文字列を分割します。
Java SE 7 では，分割する文字列の先頭でゼロ幅一致した場合，分割結果の先頭に空の文字列を生成していましたが, Java SE8 からは先頭に空の文字列を生成しません。Java SE 7 までの split()を使用して，かつゼロ幅一致した場合の処理を行っている Java プログラムにおいては，Java SE 8 からはゼロ幅一致した場合の分割結果が変わることに注意してください。

[例]

<ソース>

```
String[] result = "Test".split("");
```

<実行結果：Java SE 7 以前>

result = [, T, e, s, t]

<実行結果：Java SE 8>

result = [T, e, s, t]

例では空文字列をパターンとしていますが，これ以外にも「分割する文字列の先頭でゼロ幅一致する」を満たす正規表現を指定した場合にも該当します。

なお，String と Pattern では split()メソッドの this と第 1 引数の意味が逆になりますので，引数の指定の仕方に注意してください。

API の詳細は Java SE API リファレンスを参照してください。

-java.lang.String.split()

http://docs.oracle.com/javase/jp/8/docs/api/java/lang/String.html

-java.util.regex.Pattern.split()

http://docs.oracle.com/javase/jp/8/api/java/util/regex/Pattern.html

(5) java.lang.management.ManagementFactory.getMemoryPoolMXBeans()の挙動変更について

Java SE 8 から，Permanent 領域が廃止され Metaspace 領域が導入されました。

それに伴い，java.lang.management.ManagementFactory.getMemoryPoolMXBeans()で返却される MemoryPoolMXBean の配列の順番が変更されました。

Java SE 7 までは配列の順番は以下の通り固定となっていました。

(a)CodeCache 領域

(b)Eden 領域

(c)Survivor 領域

(d)Tenured 領域

(e)Permanent 領域

Java SE8 では，上記のうち Permanent 領域が Metaspace 領域に変更となります。さらに，配列の順番がオプション構成により変動します。

よって，上記メソッドを使用している場合，配列の順番に注意し，意図した領域の情報が取得できているか確認してください。なお, 取得した領域は, java.lang.management.MemoryPoolMXBean.getName()で領域名を確認できます。

(6) Permanent 領域の廃止と Metaspace 領域の導入について

Java SE 8 では, ロードされたクラスなどの情報が格納される領域を Metaspace 領域と呼びます。なお，Java SE 7 までは，ロードされたクラスなどの情報が格納される領域は Permanent 領域でしたが，Java

SE 8 では廃止されました。
Metaspace 領域の導入に伴い，JavaVM オプションなど下記の影響があります。

(a)PermSize，MaxPermSize の JavaVM オプションが廃止されます
代わりに, MetaspaceSize, MaxMetaspaceSize というオプションが導入されます。これらについては，Java SE 7 までの Permanent 領域の設定の考え方と同様に，アプリケーションで必要なクラス情報のサイズを見積もり，その値を設定することを推奨します。

(b)MaxMetaspaceSize のデフォルト値は MaxPermSize と異なります
MaxMetaspaceSize オプションのデフォルト値は，int 値で表現可能な最大値として定義されており，ユーザが特に指定しない場合は実質的に無限大となり，Metaspace のサイズに起因する OutOfMemoryError が発生しません。ただし，Metaspace 領域が肥大化すると，メモリスワップの多発によりシステム全体がスローダウンするなど，他のプロセスへの影響が起こり得ます。

Metaspace に関する詳細は，ユーザーズガイドを参照してください。

－以上－