富士通 Java ブログ

Java バージョンアップにともなう移行ポイント
～ OpenJDK における互換性と非互換の考え方 ～

2023 年 6 月 16 日 初版

数村 憲治

OpenJDK では 6 カ月ごとに新しいメジャーバージョンをリリースしており、Java 技術のイノベーションスピードが速くなっています。しかし、イノベーションと互換性は、トレードオフになることがあります。 Java 言語・ランタイムは、他のプログラミング言語に比べて、バージョン間の互換性は高くなっていますが、まったく非互換がない、というわけではありません。

本稿では、Java バージョンアップにともないアプリケーションを移行する際にポイントとなる、Java における互換性の定義、互換ポリシー、互換プロセスを解説します。

Java における「互換」の定義

Java の互換性は、OpenJDK Wiki で、定義されています。ここでは簡単にその内容を紹介します。 互換性の種類は以下の 3 つがあります。

- ソース互換
- バイナリ互換
- ふるまい互換

ソース互換

Java コンパイラ（javac コマンド）により、Java ソース（.java）がクラスファイル（.class）に変換されますが、ソース互換とは、この過程における互換性になります。単に、Java ソースがコンパイルできるかどうかだけではなく、生成されたクラスファイルの妥当性まで含まれています。Java コンパイラが行っていることの一つに 名前の変換 があり、これも互換性の対象となります。
ソース互換は、以下の 3 レベルに分類されます。

- レベル 1：既存のコードがコンパイルできるか
- レベル 2：レベル 1 が満たせる場合、クラスファイル内のすべてのバイナリ名が同じになるか
- レベル 3：レベル 2 が満たせない場合、クラスファイルのふるまいは同じになるか

レベル 1 を満たせない例

- 古い版のライブラリで提供されていたメソッドが、新しい版では提供されなくなった場合、そのメソッドを参照しているプログラムは、新しい版ではコンパイルできなくなります
- 言語仕様として新しいキーワード（例えば enum）が追加された場合、そのキーワードを識別子として使用しているプログラムはコンパイルできなくなります

レベル 2 を満たせない例

- クラスに、オーバーロードメソッドを追加した場合

例えば、あるクラスに、double を引数とするメソッドがあり

```
double foo(double d) {
  return d * 2.0;
}
```

新しい版で int を引数とするメソッドを追加すると

```
double foo(int i) {
  return i * 2.0;
}
```

foo（4）の呼出しは、古い版では foo（double d）の呼出しになりますが、新しい版では foo（int i）の呼出しになります。 この場合、クラスファイル内で使用される名前は変更され、レベル 2 を満たしません。 しかし、foo（4）の呼出し結果は同じになるので、レベル 3 を満たします。

レベル 3 を満たせない例

レベル 2 の例において、追加するオーバーロードメソッドが次のようにロジックが変更された場合

```
double foo(int i) {
  return i * 3.0;
}
```

foo（4）の呼出し結果は、古い版と新しい版で変わることになるため、レベル 3 を満たしません。

```
java -jar org.eclipse.transformer.cli-0.5.0.jar {入力ファイル} [出力ファイル]
```

(参考)名前の変換

名前の変換とは、Java ソース上に記述された名前から、クラスファイル中で表現される名前への変換のことを言います。例えば、Java ソースで、

```
System.out.println("hello");
```

と記載したとき、クラスファイルでは、以下のように変換されます。

javap コマンドによる出力例抜粋

```
#2 = Fieldref           #18.#19        // java/lang/System.out:Ljava/io/PrintStream;
#4 = Methodref          #21.#22        // java/io/PrintStream.println:(Ljava/lang/String;)V
#18 = Class              #26            // java/lang/System
#19 = NameAndType        #27:#28        // out:Ljava/io/PrintStream;
#21 = Class              #29            // java/io/PrintStream
#22 = NameAndType        #30:#31        // println:(Ljava/lang/String;)V

invokevirtual #4                  // Method java/io/PrintStream.println:(Ljava/lang/String;)V
```


バイナリ互換

Java のバイナリ互換については、Java 言語仕様 で定義されています。 バイナリ互換の定義は非常に簡単で、 「既存のバイナリが引き続きエラーなく リンク できること」になります。 逆に言うと、それ以外については、Java ではバイナリ互換の議論対象ではありません。 例えば、エラーをスローするメソッドがあったとして、 そのメソッドの実装が、エラーのスローではなく意味のある結果を返却するように変更した場合、 メソッドの動作は変わりますが、これはバイナリ互換の議論対象ではありません。

ふるまい互換

ふるまい互換とは、直感的には、「違うバージョンのライブラリやプラットフォームに対して、同じ入力をすれば、同じ結果になること」ですが、「同じ結果」の意味が重要になります。 例えば、タイミングに依存するようなふるまいは、Java のふるまい互換の対象外です。あるアプリケーションで、2 つのスレッド A と B があり、どちらのスレッドが先に終了するかを判定するプログラムがあったとき、ある JVM のバージョン下ではスレッド A が先に終わっていたけど、別のバージョンの JVM ではスレッド B が先に終わった場合、このようなふるまいの違いは、ふるまい互換の対象外となります。 ライブラリを修正するときに重要となるのは、仕様の中で定義されていることが「ふるまい」になるということです。例えば、HashSet の仕様では、そのイテレーションの順番は保証できないと記載されていて、実際、イテレーションのアルゴリズムは何度も変更されており、その順番に互換性はありませんが、それは、Java のふるまい互換の議論対象ではありません。

OpenJDK における互換審査プロセス

OpenJDK では、CSR（Compatibility & Specification Review） Group と呼ばれるグループがあります。CSR Group の役割は、JDK と JDK 使用ユーザ間に生じうる互換性についてレビューします。主に、仕様の変更が対象になりますが、それだけにとどまらず、実装の変更も対象となることがあります。 CSR Group によるレビューのワークフローは、Wiki で定義されています。このワークフローで承認されたものが、リリース可能となります。

CSR のポリシーは以下になり、必ずしも非互換を認めない、というものではありません。

- 十分な理由がない限り、バイナリ非互換を発生させない
- ソース非互換は避ける
- ふるまい非互換はマネジメントする

OpenJDK での実績

CSR の対象となったものは、JDK Bug System で管理されています。 例えば、JDK 11 で対象となった CSR は、以下の 92 個になります。

https://bugs.openjdk.java.net/issues/?filter=33839

JDK の非互換内容を調査する場合は、CSR の内容もみると理解が深まる場合があります。

まとめ

本稿では、Java における互換性の定義や、OpenJDK で実施されている互換審議プロセスを紹介しました。 アプリケーションを新しいバージョンの Java に移行する際の参考としてください。 Java は、プログラミング言語の中でもっとも互換性を重視する言語です。しかし、それは、まったく非互換を生じさせないということではありません。適切に互換性の意味を定義し、互換審議のプロセスを設定することで、Java は、互換性を維持しつつ、イノベーションを実現しています。