SRA の技術情報誌
GS LetterNeo
Vol.164　2022 年 3 月

LTS の Java 17 がリリース！

佐々木　和繁

関西事業部

3 年ぶりの LTS バージョン

Java は Java9 以降、毎年 3 月と 9 月にメジャーバージョンアップされるようになりました。そして 2018 年の 9 月にリリースされた Java 11 は LTS と呼ばれる長期サポートバージョンになり、以降は 3 年ごとに LTS 版がリリースされる予定で実際 2021 年の 9 月に LTS の Java 17 がリリースされました。

実際に LTS とするかどうかは JDK のディストリビュータによりますが、基本的に多くのディストリビュータは Oracle が決めたこの LTS サイクルに合わせています。

もし今後 3 年ごとに LTS がリリースされるとしたら LTS のバージョンは

Java 23（2024 年）

Java 29（2027 年）

Java 35（2030 年）

Java 41（2033 年）

Java 47（2036 年）

Java 53（2039 年）

Java 59（2042 年）

となります。

これらのバージョンを見て勘の良い方はお気づきだと思いますが、Java 35 を除いてバージョン番号が全て素数です。もちろん、これらの間には 31、37、 43 という素数だけど LTS でないバージョンも存在しますが、あるバージョンが LTS かどうか判断する目安に素数かどうかが使えるかもしれません。

冗談はさておき、現在 Oracle 社は LTS のリリースサイクルを 3 年ではなく 2 年に変更する案を検討しています。その場合次の LTS は Java 23（2024 年）ではなく Java 21（2023 年）になる予定です。そうなると先ほどの素数の話は全く関係なくなってしまいます。

話を Java 17 に戻しまして、本記事では Java 17 で追加された API の中から「16 進数表記用クラス」と「乱数生成用クラス」について紹介します。

Java 17 で追加された API を全て知りたいという方は、以下のサイトで Java 11 から Java 17 までに追加された API を参照することができます。

“New API List (Java SE 17 & JDK 17)”

https://docs.oracle.com/en/java/javase/17/docs/api/new-list.html

16 進数表記用のクラス

現在私が開発に関わっているプロジェクトは、IoT 寄りのシステムでバイナリデータを扱う機会が多く、バイト配列と 16 進数文字列の間の変換をプログラミングすることがよくあります。

そのような場合、今までは外部のライブラリを使うことが多かったのですが、Java 17 からは新たに追加された java.util.HexFormat クラスを使えばよさそうです。ちなみに HexFormat クラスは immutable でスレッドセーフです。

HexFormat クラスを使うには static メソッドの **of()** でオブジェクトを取得します。

```
HexFormat hex = HexFormat.of();
```

バイト配列から 16 進数文字列に変換するには **formatHex()** メソッドを使います。

16 進数文字列をバイト配列に変換したい場合は **parseHex()** メソッドを使います。

```
byte[] bytes = hex.parseHex("0aff");
// bytes ==> byte[2] {10, -1}
```

引数に渡す 16 進数文字列は大文字でも小文字でも、それらの混合でも OK です。

```
byte[] bytes = hex.parseHex("0AfF");
// bytes ==> byte[2] {10, -1}
```

バイト配列を大文字の 16 進数文字列に変換したいときは **withUpperCase()** メソッドを使って HexFormat オブジェクトを取得します。

```
HexFormat hex = HexFormat.of().withUpperCase();

byte[] bytes = {0, 1, -1, 10};
String hexString = hex.formatHex(bytes);
//hexString ==> "0001FF0A"
```

他にも色々メソッドがありますが、よく使いそうなのはこれらになると思います。

乱数生成用のクラス

Java 17 で乱数生成用に RandomGenerator というインターフェイスと RandomGeneratorFactory というクラスが追加されました。

Java には元々java.util.Random という乱数生成用のクラスがありましたが、実はこの Random クラス、あまり評判が良くありません。

というのも、乱数といいつつ生成する値は一様に分布されず「偏り」が大きいためです。特に Random オブジェクトを生成して最初に取得する値にはかなり偏りがあることで有名です。

そういった理由により、偏りの少ない乱数を生成する必要がある場合はこの Random クラスではなく java.security.SecureRondom クラスを使うことが推奨されていました。ただし SecureRandom クラスは Random クラスに比べるとかなりパフォーマンスが悪く、暗号化キーなど機密性が求められる場面のみで使われることが多いと思います。（例えば Tomcat のセッション ID の生成にはこのクラスが使われています）

Java 17 では乱数生成用のインターフェイスとして RandomGenerator が追加され、java.util.Random や java.security.SecureRandom クラスなどはこの interface を実装するようになりました。

さらに RandomGenerator オブジェクトを生成するためのクラスとして RandomGeneratorFactory が追加されています。

RandomGeneratorFactory を使って RandomGenerator オブジェクトを生成するには、まず **of(String name)**メソッドを使って RandomGeneratorFactory オブジェクトを生成し、その RandomGeneratorFactory オブジェクトの **create()** メソッドで RandomGenerator オブジェクトを生成します。そうして取得した RandomGenerator オブジェクトは今までの java.util.Random オブジェクトと同じように使えます。

```
RandomGeneratorFactory<RandomGenerator> factory =
  RandomGeneratorFactory.of(“Random”);

RandomGenerator random = factory.create();
int i = random.nextInt();
```

一度ファクトリを生成することで手順が増えることになりましたが、このファクトリを生成するメソッド **of(String name)**が今回の**キモ**で引数に渡す文字列によって乱数生成のアルゴリズムを指定することができます。

指定することができるアルゴリズムは上記のサンプルにある **Random** のほかに **Xoroshiro128PlusPlus** や **Xoshiro256PlusPlus**、**L128X1024MixRandom** などがあります。このように乱数生成のアルゴリズムを指定できるようにすることで java.util.Random クラスが持っている問題にも対応できるようになりました。（”Random”を指定すると従来の java.util.Random クラスが使われます）

ちなみに指定できるアルゴリズムの一覧は以下のサイトから参照できます。

”java.util.random (Java SE 17 & JDK 17)”

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/random/package-summary.html#algorithms

このサイトを見るとアルゴリズムごとに **StateBits** という項目がありますが、これは**乱数生成の内部状態のビット数**です。今まで乱数という言葉を使ってきましたが API が生成する乱数は疑似乱数であり、計算によって求めた値になります。そして疑似乱数の計算の際には、API は内部に状態を保持して、値を生成するたびに内部状態を変更します。その変更された状態が過去に値を生成した状態と同じ時にはその時と同じ値を生成し、それ以降の値の列も同じものになります。つまり、**疑似乱数の値列は周期的である**と言えます。

この節の始めに java.util.Random クラスがあまり良い評判でないと述べましたが、このクラスのもう一つの問題は内部の状態が 48 ビットであることです。先ほどのサイトを見るとわかるように Random 以外のアルゴリズムでは最低でも 64 ビット、中には 1000 ビット以上の内部状態を持つものもあります。これは Random クラスの周期が比較的短いことを意味します。乱数の周期を長くしたい場合は StateBits のビット数が多いものを選択すればいいので、そのような場合にも今回の変更は有意義なのではないでしょうか。

おわりに

今回は 2021 年 9 月にリリースされた Java 17 について紹介しました。2022 年 3 月には Java 18 がリリースされますが、慌てて Java 18 を適用する必要はないと思います。Java のバージョンを検討するなら、まずは LTS である Java 17 の適用をお勧めします。

Java のメジャーバージョンアップが半年ごとになって、ついていくのが大変だと考えている開発者も多いですが「メジャーバージョンアップ」という言葉に惑わされず LTS バージョンをしっかりキャッチアップすることが大切です。そのための助けとして本稿が役立てば幸いです。

GSLetterNeo Vol.164
2022 年 3 月 20 日発行
発行者　株式会社 SRA 先端技術研究所

編集者　熊澤努　方 学芳
バックナンバー https://www.sra.co.jp/public/sra/gsletter/
お問い合わせ　gsneo@sra.co.jp

株式会社SRA
〒171-8513 東京都豊島区南池袋 2-32-8