Scalaの文字列処理

Day 5　ミュータビリティとフォーマット

ミュータビリティ

ミュータビリティとは、可変性。

コレクションクラスにおいて、

・ミュータブル　＝＞　可変長

・イミュータブル　＝＞　固定長

スレッドセーフティ

マルチスレッドの問題：複数のスレッドが同一クラスを操作するとスレッドAが操作している変数を操作が終わっていないのにスレッドBが操作してしまう問題が発生する。

・スレッドセーフ　＝　スレッドAが操作している時にスレッドBが操作しないように排他制御されている状態

・スレッドアンセーフ　＝　排他制御されていない状態

文字列に関するクラスの
ミュータビリティとスレッドセーフ

	ミュータビリティ	スレッドセーフ
String	イミュータブル（固定長）	スレッドセーフ
StringBuilder	ミュータブル（可変長）	スレッドアンセーフ
StringBuffer	ミュータブル（可変長）	スレッドセーフ

Stringはインスタンス生成後に操作可能な変数がないためスレッドセーフ

・イミュータブルをミュータブルにしたり、

・スレッドアンセーフをスレッドセーフにしたりすると？

クラスにより多くのメソッドが乗ったり、処理にオーバーヘッドが乗ったりします。

適材適所で使い分けましょう！

String

イミュータブル（固定長）でスレッドセーフな文字列クラス

＋演算子、unionメソッド、concatメソッドで文字列を結合できるが、新たにインスタンスを生成する。

StringBuffer/StringBuilder

java.lang.StringBufferとjava.lang.StringBuilderは
java.lang.AbstractStringBuilderを継承し同様のAPI
が使用可能

Scalaで一般的に使用する
scala.collection.mutable.StringBuilderの中身は
java.lang.StringBuilder

StringBuffer/StringBuilderのappendとtoString/result

- append / appendCodePoint
  - 文字や文字列やDoubleや数値型などをCharとして追加(Char追加が最速)
  - メソッドチェーンで記述可能
  - 前方に追加するprependメソッドはない（insertメソッドかreverseメソッドで対応可能）
- toString / result
  - 収容されたCharの配列をStringに変換して出力

StringBuffer/StringBuilderのcapacityとlength

- capacity
  - 容量（Char数）。不足したら自動で増やすが増やす処理がオーバヘッドとなる
    - インスタンス生成時に特に指定しないとデフォルト値として16が指定される
- length
  - 収容されているCharの数
- trimToSize
  - capacityをlengthに揃えることでメモリの使用を減らせる

StringBuffer/StringBuilderのdeleteとsetLength(0)/clear

- delete(0, length)とsetLength(0) / clear
- lengthを0にして収容物を破棄
- capacityは変えないため高速
- setLength(0) / clearの方がdelete(0, length)より高速

java.lang.StringBuffer

ミュータブル（可変長）でスレッドセーフな文字列クラス

すべてのメソッドがsynchronized修飾子付き

java.io.StringWriterの中身

StringBuilder

ミュータブル（可変長）でスレッドアンセーフな文字列クラス

scala.collection.mutable.StringBuilderの中身はjava.lang.StringBuilder

java.lang.StringBuilderはjava.lang.StringBufferとAPIの互換性を保つように設計

StringJoiner

StringJoinerはデリミタ（と接頭辞・接尾辞）をインスタンス化時に与えて、文字列を結合するクラス

addメソッドで文字列を追加し、toStringでStringに変換

String.joinメソッド

String.joinメソッドは第一引数をデリミタとし、文字列を結合

String.joinメソッドの中身はStringJoinerクラス

接頭辞・接尾辞は与えられない点がStringJoinerクラスと違う(mkStringメソッドと組合せて実現)

Javaの可変長引数やjava.lang.Iterableが引数となるため、Javaとの互換性に注意

PrintWriter/PrintStream

- PrintWriter (StringWriter)
- PrintStream (ByteArrayOutputStream)

を使いprint, println, printfメソッドで文字列を生成

PrintWriterの方が高速

StringWriterの中身はStringBuffer

java.nio.Buffer

主にBufferを継承したByteBuffer/CharBufferがエンコーダ/デコーダの内部で使用される

capacityを超えるとオーバフローする点でStringBuffer/StringBuilderとは異なる

次の１）と２）を交互に行う

１）positionからlimitまで書き込む（読み込む）

２）positionとlimitを変化させる

フォーマット（書式）

書式に値を埋め込みStringを生成

printf, str.format, f補間子, String.format, Formatter, DateTimeFormatter, SimpleDateFormatなどで使用可能

printfスタイルの書式の定義はjava.util.Formatterクラスので説明されている

StringのformatメソッドJava由来のstaticなString.formatメソッドとScalaで使用可能なstr.formatメソッドがあるが、互換性の問題があるためScalaでは後者を使うべき

Java由来：String.format(“%d%%", 100.asInstanceOf[java.lang.Integer])

Scala固有：”%d%%".format(100)

DateTimeFormatter

日付・時刻用のフォーマットクラス

val tdf = DateTimeFormatter.ofPattern(“yyyy/MM/dd")

- ZonedDateTimeクラスをformatしてStringに変換
- StringをparseしてZonedDateTimeクラスに変換

日付・時刻のクラスが持つformat/parseメソッドの中身

SimpleDateFormatクラスの改良版

DateFormatと SimpleDateFormat

DateFormatやSimpleDateFormatより
DateTimeFormatterを使いましょう！

DateFormatは、getDateInstance（日付），
getTimeInstance（時刻），getDateTimeInstance
（日付・時刻）でインスタンスを生成

SimpleDateFormatは、DateTimeFormatterと同様
の書式が利用可能

NumberFormatと DecimalFormat

NumberFormatは、getIntegerInstance（整数）、
getCurrencyInstance（通貨）、
getPercentInstance（%）でインスタンスを生成

ロケールに合わせて通貨コードも取得可能

DecimalFormatは、NumberFormatの
getIntegerInstance

ChoiceFormat

値の範囲にラベルをつけられる（例：0未満を「負の数」、0を「0」、0より大きい数を「正の数」）

MessageFormat

引数番号を指定して値を埋めることができる

“今日は{0,date,yyyy年MM月dd日}、時刻は{0,time}。
天気は{1}です。"

DateFormat、SimpleDateFormat、
NumberFormat、DecimalFormat、ChoiceFormat
の上位互換

テンプレートエンジン

複数行を%nや\nで表す１行の雛形は読みづらい

生文字リテラル（＋フォーマットや文字列補間）と
Scala XMLで複数行の雛形に値を埋めることが可能

テンプレートエンジンは、複数行を許す雛形（テンプレート）に変数を埋め込むためのライブラリ

例：Apache VelocityやHandlebars.javaなど