Java Agile

2014年07月28日 18時00分 更新

Java 8はラムダ式でここまで変わる（終）：

Stream APIの特殊なメソッドとメソッド参照／コンストラクター参照 (2/4)

[長谷川智之，株式会社ビーブレイクシステムズ]

ListやMapなどに要素を格納した結果を作成するcollectメソッド

次は、collectメソッドを見てみましょう。このメソッドはStramが持つ要素を抜き出してListなどのCollectionやMapなど値を格納するものの状態を変更して、それを結果として受け取ることができるメソッドです。

このcollectメソッドは引数にjava.util.stream.Collectorインターフェースを持つものと3つの引数を持つものの2つのメソッドがあります。Collectorインターフェースを引数に持つメソッドは、java.util.stream.Collectorsクラスのメソッドを使ってListやMapなどを作成するのに使いやすいです。また3つの引数を持つものはCollectorsクラスでまかなえない場合や自作の値を格納するクラスを使いたい場合に使いやすいです。

戻り値	メソッド	概要
R	collect(Collector collector)	引数のCollectorインターフェースによってStreamの要素を使って生成される結果を返すメソッド。java.util.stream,Collectorsクラスのメソッドを使うのに適している
R	collect(Supplier<R> supplier, BiConsumer<R, ? super T> accumulator, BiConsumer<R, R> combiner)	Supplierで生成したRにStreamの要素Tから取得した値をaccumulatorでRに格納し、全ての要素の処理が終わった際のRを返す

また、Collectorsクラスにあるメソッドは Streamの要素を単純にListやMapに変換するだけではなく、Streamの要素を絞り込んだり、文字列連結など何らかの処理を行ったりするものもあります。興味がある人はCollectorsクラスのAPIを見てみるのもいいでしょう。

それではcollectメソッドを使ったサンプルを試してみましょう。ここでは次のPersonクラスがあるとします。

```
1.  public class Person {
2.
3.      private String lastName;
4.
5.      private String firstName;
6.
7.      public Person(String lastName, String firstName) {
8.          super();
9.          this.lastName = lastName;
10.         this.firstName = firstName;
11.     }
12.
13.     public String getLastName() {
14.         return lastName;
15.     }
16.
17.     public String getFirstName() {
18.         return firstName;
19.     }
20.
21.     @Override
22.     public String toString() {
23.         return "Person [lastName=" + lastName + ", firstName=" + firstName + "]";
24.     }
25. }
```

引数が1つのcollectメソッド

このPersonクラスを格納したListからStreamを生成し、collectメソッドを使ったサンプルを次に示します。

```
public class CollectSample1 {

    public static void main(String[] args) {
        List<Person> list = new ArrayList<>();
        list.add(new Person("テスト", "太郎"));
        list.add(new Person("テスト", "次郎"));
        list.add(new Person("テスト", "花子"));
        list.add(new Person("サンプル", "小太郎"));
        list.add(new Person("サンプル", "小次郎"));
        list.add(new Person("サンプル", "華子"));

        // --- 引数にCollectorsを使ったサンプル ---
        // PersonのfirstNameのListを作るサンプル
        List<String> firstNameList = list.stream()
                .map(element -> element.getFirstName()) // StringのStreamに変換
                .collect(Collectors.toList());
        System.out.println("[1] firstNameList = " + firstNameList);

        // lastNameをキーにPersonのMapを作成するサンプル
        Map<Object, List<Person>> personMap = list.stream()
                .collect(Collectors.groupingBy(element -> element.getLastName()));
        System.out.println("[2] personMap = " + personMap);

        // --- 引数を3つ受け取るサンプル ---
        // lastNameとfirstNameを結合した文字列を返すサンプル
        List<String> nameList = list.stream().collect(
                () -> new ArrayList<String>(),
                (container, element) ->
                    container.add(element.getLastName() + element.getFirstName()),
                (container1, container2) -> container1.addAll(container2));
        System.out.println("[3] nameList = " + nameList);
    }
}
```

```
[1] firstNameList = [太郎, 次郎, 花子, 小太郎, 小次郎, 華子]
[2] personMap = {サンプル=[Person [lastName=サンプル, firstName=小太郎], Person [lastName=サンプル, firstName=小次郎], Person [lastName=サンプル, firstName=華子]], テスト=[Person [lastName=テスト, firstName=太郎], Person [lastName=テスト, firstName=次郎], Person [lastName=テスト, firstName=花子]]}
[3] nameList = [テスト太郎, テスト次郎, テスト花子, サンプル小太郎, サンプル小次郎, サンプル華子]
```

実行結果

引数が3つのcollectメソッド

また、引数を3つ受け取るcollectメソッドの第3引数は、reduceメソッドと同様に並列処理の場合に使われるものになります。先ほどのサンプルで引数を3つ受け取るメソッドの第2引数と第3引数が受け取る値を標準出力するように変更したものを見てみましょう。

```
public class CollectSample2 {

    public static void main(String[] args) {
        List<Person> list = new ArrayList<>();
        list.add(new Person("テスト", "太郎"));
        list.add(new Person("テスト", "次郎"));
        list.add(new Person("テスト", "花子"));
        list.add(new Person("サンプル", "小太郎"));
        list.add(new Person("サンプル", "小次郎"));
        list.add(new Person("サンプル", "華子"));

        // --- 引数を3つ受け取るサンプル ---
        System.out.println("--- 直列処理 ---");
        List<String> nameList = list.stream().collect(
                () -> new ArrayList<String>(),
                (container, element) -> {
                    System.out.println("container = " + container);
                    System.out.println("element = " + element);
                    container.add(element.getLastName() + element.getFirstName());
                },
                (container1, container2) -> {
                    System.out.println("container1 = " + container1);
                    System.out.println("container2 = " + container2);
                    container1.addAll(container2);
                });
        System.out.println("[1] nameList = " + nameList);

```

```
28.         System.out.println("--- 並列処理 ---");
29.         List<String> nameParallelList = list.parallelStream().collect(
30.                 () -> new ArrayList<String>(),
31.                 (container, element) -> {
32.                     System.out.println("container = " + container);
33.                     System.out.println("element = " + element);
34.                     container.add(element.getLastName() + element.getFirstName());
35.                 },
36.                 (container1, container2) -> {
37.                     System.out.println("container1 = " + container1);
38.                     System.out.println("container2 = " + container2);
39.                     container1.addAll(container2);
40.                 });
41.         System.out.println("[2] nameParallelList = " + nameParallelList);
42.     }
43. }
```

```
--- 直列処理 ---
container = []
element = Person [lastName=テスト, firstName=太郎]
container = [テスト太郎]
element = Person [lastName=テスト, firstName=次郎]
container = [テスト太郎, テスト次郎]
element = Person [lastName=テスト, firstName=花子]
container = [テスト太郎, テスト次郎, テスト花子]
element = Person [lastName=サンプル, firstName=小太郎]
container = [テスト太郎, テスト次郎, テスト花子, サンプル小太郎]
element = Person [lastName=サンプル, firstName=小次郎]
container = [テスト太郎, テスト次郎, テスト花子, サンプル小太郎, サンプル小次郎]
element = Person [lastName=サンプル, firstName=華子]
[1] nameList = [テスト太郎, テスト次郎, テスト花子, サンプル小太郎, サンプル小次郎, サンプル華子]
--- 並列処理 ---
container = []
element = Person [lastName=サンプル, firstName=小太郎]
container = []
container = []
element = Person [lastName=サンプル, firstName=小次郎]
container = []
element = Person [lastName=テスト, firstName=太郎]
container = []
element = Person [lastName=テスト, firstName=花子]
container = []
element = Person [lastName=サンプル, firstName=華子]
container1 = [サンプル小次郎]
container2 = [サンプル華子]
container1 = [サンプル小太郎]
container2 = [サンプル小次郎, サンプル華子]
element = Person [lastName=テスト, firstName=次郎]
container1 = [テスト次郎]
container2 = [テスト花子]
container1 = [テスト太郎]
container2 = [テスト次郎, テスト花子]
container1 = [テスト太郎, テスト次郎, テスト花子]
container2 = [サンプル小太郎, サンプル小次郎, サンプル華子]
[2] nameParallelList = [テスト太郎, テスト次郎, テスト花子, サンプル小太郎, サンプル小次郎, サンプル華子]
```

実行結果

次ページからは、メソッド参照とコンストラクター参照について見ていきましょう。

メソッド参照

前のページへ 1 2 3 4 次のページへ

チェックしておきたい人気記事