Javaプログラミング実習

35. Stream API

株式会社ジードライブ

© ZDRIVE, K.K. All rights reserved.

今回学ぶこと

- Stream API の概要
- Stream API の使い方

© ZDRIVE, K.K. All rights reserved.

Stream API とは

- コレクション(List, Set, Map)に対する反復処理のためのAPI
  - forなどの反復処理と異なり、並び替えなどの中間操作を組み込むことができる
  - 並列／分散処理にも対応しており、マルチコアプロセッサの場合、実行速度の向上が期待できる

先に学習したストリーム入出力とは別のものです

© ZDRIVE, K.K. All rights reserved.

Stream API 利用の流れ

1. Streamの生成
   - コレクションからStreamを生成する
2. 中間操作の実行
   - 要素の並び替えや絞り込みなど
3. 終端操作の実行
   - 各要素に対する操作

© ZDRIVE, K.K. All rights reserved.

Streamの生成

- Stream API を使用するためには、まず**Streamオブジェクト**を生成する必要がある
- Stream操作には**順次実行モード**と**並列実行モード**の2種類の実行モードがある
  - 並列実行モードでは、マルチコアプロセッサの場合に実行速度の向上が期待できる
  - ただし、扱うデータ数が多い場合や処理内容によっては順次実行モードよりも処理時間がかかる場合がある

© ZDRIVE, K.K. All rights reserved.

Streamの生成 (1)

- ListからのStream生成

```
List<String> fruits = …;
Stream<String> stream = fruits.stream();
```

- SetからのStream生成

```
Set<String> fruits = …;
Stream<String> stream = fruits.stream();
```

- MapからのStream生成

```
Map<String, Integer> fruits = …;
Stream<Entry<String, Integer>> stream = fruits.entrySet().stream();
```

並列実行モードの場合は **.parallelStream()** にする

© ZDRIVE, K.K. All rights reserved.

Streamの生成 (2)

- 配列からのStream生成

```
String[] fruits = …;
Stream<String> stream1 = Arrays.stream(fruits); // 方法1
Stream<String> stream2 = Stream.of(fruits);     // 方法2
```

- 個別要素からのStream生成

```
Stream<String> stream =
                  Stream.of("りんご", "バナナ", "ぶどう");
```

© ZDRIVE, K.K. All rights reserved.

終端操作

- 要素の集合に対して最終的に行う操作
- 終端操作の例
  - ストリーム内の各要素を画面に表示する
  - ストリームをリストへ変換して後の処理へ回す
  - ストリーム内の要素数を数えるなどの集計を行う

© ZDRIVE, K.K. All rights reserved.

終端操作

主な終端操作

メソッド	役割	説明
forEach()	要素の処理	**T -> { }** となるラムダ式を渡し、各要素に対する処理を行う。
count()	要素の集計	要素の数を取得する。戻り値はlong型
collect()	要素の収集	Collectorオブジェクトを渡し、要素集合をリストとして取得するなどの操作を行う。

© ZDRIVE, K.K. All rights reserved.

終端操作の例

例：forEach()

```
List<String> fruits = Arrays.asList("りんご", "バナナ", "ぶどう");
fruits.stream().forEach(e -> System.out.println(e));
```

コンソール表示

```
りんご
バナナ
ぶどう
```

例：count()

```
long count = Stream.of(50, 20, 80).count();
System.out.println(count);
```

コンソール表示

```
3
```

© ZDRIVE, K.K. All rights reserved.

終端操作

例：collect()

```
Stream<String> fruitsStream =
        Stream.of("りんご", "バナナ", "ぶどう");

// StreamからListへの変換
List<String> fruits =
        fruitsStream.collect(Collectors.toList());
```

引数はCollectorsのメソッド

Collectors.toList()
Collectors.toSet()
Collectors.counting() など

© ZDRIVE, K.K. All rights reserved.

中間操作

主な中間操作

メソッド	役割	説明
filter()	要素の絞り込み	**T -> boolean** となるラムダ式を渡し、trueとなるものだけが残ったStreamを取得する。
map()	要素の変換	**T -> U** となるラムダ式を渡し、変換後のStreamを取得する。
sorted()	要素の並べ替え	**(T, T) -> int** となるラムダ式を渡し、並べ変えられたStreamを取得する。
limit()	要素の絞り込み	集合の先頭から引数で指定した数の要素のみを含んだStreamを取得する。

© ZDRIVE, K.K. All rights reserved.

中間操作の例

- filter()の例：値が60以上の要素のみ取得する

```
List<Integer> scores = Arrays.asList(60, 20, 80);
scores.stream()
    .filter(e -> e >= 60)
    .forEach(e -> System.out.println(e));
```

コンソール表示

```
60
80
```

- map()の例：Integer型からString型への変換

```
List<Integer> scores = Arrays.asList(60, 20, 80);
scores.stream()
    .map(e -> e + "点")
    .forEach(e -> System.out.println(e));
```

コンソール表示

```
60点
20点
80点
```

© ZDRIVE, K.K. All rights reserved.

中間操作

- sorted()の例： 昇順は e1 - e2　　降順は e2 - e1

```
List<Integer> scores = Arrays.asList(60, 20, 80);
scores.stream()
    .sorted((e1, e2) -> e2 - e1)
    .forEach(e -> System.out.println(e));
```

コンソール表示

```
80
60
20
```

- limit()の例：ストリームの先頭から2要素のみを取り出す

```
List<Integer> scores = Arrays.asList(60, 20, 80);
scores.stream()
    .limit(2)
    .forEach(e -> System.out.println(e));
```

コンソール表示

```
60
20
```

© ZDRIVE, K.K. All rights reserved.

練習

- 練習35-1
- 練習35-2

© ZDRIVE, K.K. All rights reserved.