Java 8

Collectors

Редукция

Пример – сумма (sum)

- Результат – число. Начальное значение равно 0
- При получении нового элемента его надо прибавить к результату
- Чтобы совместить два частичных результата их надо сложить

Редукция

Итого. У нас есть 3 операции:

- Определение контейнера, который будет содержать итоговый результат
- Добавление элемента к контейнеру
- Совмещение двух частично наполненных контейнеров

Редукция

- Редукция основана на трех операциях:
- Constructor: Supplier
- Accumulator: Function
- Combiner: Function

Редукция

- Редукция основана на трех операциях:
- Constructor: Supplier

```
() -> new StringBuffer() ;
```

- Accumulator: Function

```
(StringBuffer sb, String s)->sb.append(s);
```

- Combiner: Function

```
(StringBuffer sb1, StringBuffer sb2) ->
sb1.append(sb2) ;
```

Редукция

- Редукция основана на трех операциях:
- Constructor: Supplier

```
StringBuffer::new ;
```

- Accumulator: Function

```
StringBuffer::append ;
```

- Combiner: Function

```
StringBuffer::append ;
```

```
class Person {
    String name; int age;
    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public String toString() {
        return name;
    }
}

List<Person> persons = Arrays.asList(
        new Person("Max", 18),
        new Person("Peter", 23),
        new Person("Pamela", 23),
        new Person("David", 12));
```

Пример

```
List<Person> persons = ...;
StringBuffer result = persons.stream()
      .filter(person -> person.getAge() > 20)
      .map(Person::getLastName)
      .collect(
            StringBuffer::new, // constructor
            StringBuffer::append, // accumulator
            StringBuffer::append // combiner
      );
```

Пример

```
List<Person> persons = ...;
String result = persons.stream()
      .filter(person -> person.getAge() > 20)
      .map(Person::getLastName)
      .collect(
            Collectors.joining()
      );
```

Пример

```
List<Person> persons = ...;
String result = persons.stream()
      .filter(person -> person.getAge() > 20)
      .map(Person::getLastName)
      .collect(
            Collectors.joining(", ")
      );
```

Пример

```
List<Person> persons = ...;
String result = persons.stream()
      .filter(person -> person.getAge() > 20)
      .map(Person::getLastName)
      .collect(
            Collectors.joining(", ", "{", "}")
      );
```

Немного теории

Collector<T, A, R> - интерфейс

Работает со следующей логикой:

- Supplier<A> supplier
- BiConsumer<A,T> accumulator
- BinaryOperator<A> combiner
- Function<A,R> finisher

- Часто вместо A везде стоит ? – так как A – «невидимый» тип

Пример-2

- Constructor: Supplier

```
() -> new ArrayList() ;
```

- Accumulator: Function

```
(list, element) -> list.add(element) ;
```

- Combiner: Function

```
(list1, list2) -> list1.addAll(list2) ;
```

Пример-2

- Constructor: Supplier

```
ArrayList::new ;
```

- Accumulator: Function

```
Collection::add ;
```

- Combiner: Function

```
Collection::addAll ;
```

Пример-2

```
ArrayList<String> result = persons.stream()
.filter(person -> person.getAge() > 20)
.map(Person::getLastName)
.collect(
    ArrayList::new,     //constructor
    Collection::add,    //accumulator
    Collection::addAll //combiner
);
```

Пример-2

```
List<String> result = persons.stream()
.filter(person -> person.getAge() > 20)
.map(Person::getLastName)
.collect(
    Collectors.toList()
);
```

Пример-2

```
HashSet<String> result = persons.stream()
.filter(person -> person.getAge() > 20)
.map(Person::getLastName)
.collect(
     HashSet::new,     //constructor
     Collection::add,   //accumulator
     Collection::addAll //combiner
);
```

Пример-2

```
Set<String> result = persons.stream()
.filter(person -> person.getAge() > 20)
.map(Person::getLastName)
.collect(
    Collectors.toSet()
);
```

Пример-2

```
TreeSet<String> result = persons.stream()
.filter(person -> person.getAge() > 20)
.map(Person::getLastName)
.collect(
    Collectors.toCollection(TreeSet::new)
);
```

Collectors

- Всего в классе Collectors 33 статических метода!
- Есть все на все случаи жизни)

HashMap: age / list of the people

```
Map<Integer, List<Person>> result =
persons.stream()
.collect(
     Collectors.groupingBy(Person::getAge)
);
```

Немного теории - 2

```
public static <T, K>
Collector<T, ?, Map<K, List<T>>> groupingBy(
      Function<? super T, ? extends K> classifier
)
```

Returns a Collector implementing a "group by" operation on input elements of type T, grouping elements according to a classification function, and returning the results in a Map.

HashMap: age / number of the people

```
Map<Integer, Long> result =
persons.stream()
.collect(
      Collectors.groupingBy(
         Person::getAge,
         Collectors.counting() // downstream collector
      )
);
```

Немного теории - 3

```
public static <T, K, D, A>
Collector<T, ?, Map<K, D>> groupingBy(
      Function<? super T, ? extends K> classifier,
      Collector<? super T, A, D> downstream
)

public static <T, K, D, A, M extends Map<K, D>>
Collector<T, ?, M> groupingBy(
      Function<? super T, ? extends K> classifier,
      Supplier<M> mapFactory,
      Collector<? super T, A, D> downstream
)
```

HashMap: Person / number of the people

```
Map<Person, Long> result =
persons.stream()
.collect(
      Collectors.groupingBy(
         person -> person,
         Collectors.counting() // downstream collector
      )
);
```

HashMap: Person / number of the people

```
Map<Person, Long> result =
persons.stream()
.collect(
      Collectors.groupingBy(
         Function.identity(),
         Collectors.counting() // downstream collector
      )
);
```

HashMap: Person / **int** number of the people

```
Map<Person, Integer> result =
persons.stream()
.collect(
      Collectors.groupingBy(
         Function.identity(),
         Collectors.summingInt(e -> 1) // collector
      )
);
```

HashMap: age / names joined in a single String

```
Map<Integer, String> result =
persons.stream()
.collect(
     Collectors.groupingBy(
        Person::getAge,
        Collectors.mapping( //1st downstream collector
            Person::getLastName,
            Collectors.joining(", ") // 2nd collector
        )
     )
);
```

HashMap: age / names sorted alphabetically

```
Map<Integer, TreeSet<String>> result =
persons.stream()
.collect(
      Collectors.groupingBy(
         Person::getAge,
         Collectors.mapping( // 1st downstream
collector
               Person::getLastName,
               Collectors.toCollection(TreeSet::new)
         )
      )
);
```

HashMap: the same, sorted by age

```
TreeMap<Integer, TreeSet<String>> result =
persons.stream()
.collect(
      Collectors.groupingBy(
         Person::getAge,
         TreeMap::new,
         Collectors.mapping( // 1st downstream
collector
               Person::getLastName,
               Collectors.toCollection(TreeSet::new)
         )
      )
);
```

```
List<Person> filtered =
    persons
        .stream()
        .filter(p -> p.name.startsWith("P"))
        .collect(Collectors.toList());

System.out.println(filtered);
```

```
Map<Integer, List<Person>> personsByAge = persons
    .stream()
    .collect(Collectors.groupingBy(p -> p.age));

personsByAge.forEach(
  (age, p) -> System.out.format("age %s: %s\n",
                age, p));

// age 18: [Max]
// age 23: [Peter, Pamela]
// age 12: [David]
```

```
Double averageAge = persons
    .stream()
    .collect(Collectors.averagingInt(
                p -> p.age));

System.out.println(averageAge);     // 19.0
```

```
IntSummaryStatistics ageSummary =
    persons
        .stream()
        .collect(Collectors.summarizingInt(
                        p -> p.age));

System.out.println(ageSummary);
// IntSummaryStatistics{count=4, sum=76,
min=12, average=19.000000, max=23}
```

```
String phrase = persons
    .stream()
    .filter(p -> p.age >= 18)
    .map(p -> p.name)
    .collect(
     Collectors.joining(
" and ", "In Germany ", " are of legal age."));

System.out.println(phrase);
//In Germany Max and Peter and Pamela are of legal age
```

```
Map<Integer, String> map = persons
    .stream()
    .collect(Collectors.toMap(
        p -> p.age,
        p -> p.name,
        (name1, name2) -> name1 + ";" +
name2));

System.out.println(map);
// {18=Max, 23=Peter;Pamela, 12=David}
```

```
Collector<Person, StringJoiner, String> personNameCollector =
    Collector.of(
        () -> new StringJoiner(" | "),          // supplier
        (j, p) -> j.add(p.name.toUpperCase()),  // accumulator
        (j1, j2) -> j1.merge(j2),               // combiner
        StringJoiner::toString);                // finisher

String names = persons
    .stream()
    .collect(personNameCollector);

System.out.println(names);  // MAX | PETER | PAMELA | DAVID
```