СПб АУ НОЦНТ РАН

Java 05

05.04.2016

Stream API. Мотивация

```
List<Integer> a = new ArrayList<>();
for (s in strings) {
  if (s.equals("stop")) break;
  if (s.contains("abc")) {
    a.add(s.length());
  }
}
```

Stream API. Мотивация

```
List<Integer> a = strings.stream()
   .takeWhile(s -> s != "stop")
   .filter(s -> s.contains("abc"))
   .map(String::length).collect(Collectors.toList());
```

Stream API. Мотивация

```
for (usage in usages) {
  element = usage.element;

  boolean allow = false;
  Node current = element;
  while (current != null) {
    if (element instanceof Class
        && isSubClass(element, target)) {
      allow = true;
    }
  }

  if (!allow) return error();

}
```

Stream API. Мотивация

```
if (usages.stream().anyMatch(
  usage ->
  Stream<Element> parents =
    Stream.iterate(
      usage.element, e -> e.parent
    ).takeWhile(Objects::nonNull);

  parents.noneMatch(e ->
      e instancef Class && isSubClass(e, target))
)) {
  return error();
}
```

Stream API. Мотивация

```
if (usages.stream().anyMatch(
  usage ->
  parents(usage.element).noneMatch(e ->
      e instanceof Class && isSubClass(e, target))
)) {
  return error();
}
```

java.util.stream.Stream<E>

- collection.stream()
- collection.parallelStream()
- Stream.of("e1 "e2")
- Stream.generate(-> random.nextInt())
- Stream.iterate(1, n -> n + 1)

java.util.stream.Stream<E>

- collection.stream()
- collection.parallelStream()
- Stream.of("e1 "e2")
- Stream.generate(-> random.nextInt())
- Stream.iterate(1, n -> n + 1)

- Ленивые не терминальные методы (возвращают Stream)
- Энергичные терминальные методы (возвращают результат)
- Одноразовые

java.util.stream.Stream<E>

- collection.stream()
- collection.parallelStream()
- Stream.of("e1 "e2")
- Stream.generate(-> random.nextInt())
- Stream.iterate(1, n -> n + 1)

- Ленивые не терминальные методы (возвращают Stream)
- Энергичные терминальные методы (возвращают результат)
- Одноразовые

```
Stream.of("abc", "cde").filter(x -> throw ..); // ?
```

java.util.stream.Stream<E>

- collection.stream()
- collection.parallelStream()
- Stream.of("e1 "e2")
- Stream.generate(-> random.nextInt())
- Stream.iterate(1, n -> n + 1)

- Ленивые не терминальные методы (возвращают Stream)
- Энергичные терминальные методы (возвращают результат)
- Одноразовые

```
Stream.of("abc", "cde").filter(x -> throw ..); // ?
Stream<String> x = Stream.of("abc", "cde");
x.collect(Collectors.toList());
print(x.collect(Collectors.toList()).size()); // ?
```

Коллекторы

```
public static<T, R> of(
  Supplier<R> supplier, // Function1<Void, R>
  BiConsumer<R, T> accumulator, // Function2<R, T, R>
  BinaryOperator<R> combiner // Function2<R, R, R>
)
```

Коллекторы

```
List<String> persons = ...;
StringBuilder result = persons.stream()
  .collect(
      StringBuilder::new, // constructor
      StringBuilder::append, // accumulator
      StringBuilder::append // combiner
  );
```

Коллекторы

```
List<String> persons = ...;
List<String> result = persons.stream()
   .map(s -> s + "#")
   .collect(
       Collectors.toList());
```

Коллекторы

```
List<String> persons = ...;
StringBuffer result = persons.stream()
  .collect(
    Collectors.joining("")
  );
```

Коллекторы

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

Коллекторы

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

Stream API

Следует обратить внимание:

- Comparator.comparing((String s) -> s.length()) — сравнение строк по их длине
- По возможности не пишем тип параметров лямбды и скобки (если очевидно из контекста)
- По возможности пользуемся method reference: s.map(String::length)
- IntStream.average()/min()
- Collectors.groupingBy
- Collectors.joining
- Collectors.counting

Ссылки

- `https://habrahabr.ru/company/luxoft/blog/270383/`
- Ctrl + N. Stream, Collectors. Ctrl + F12

Что делать сегодня

- Задание в ветке 04-streams
- Решаем первую часть, тесты проходят
- Решаем вторую часть, пишем простые тесты
- Решения первой части должны быть одним простым return-выражением

Дедлайн: 11.04 23:59