Stream

sanghyuck.na@lge.com

Stream source operation

Stream simple terminal operation

1

Stream[8]

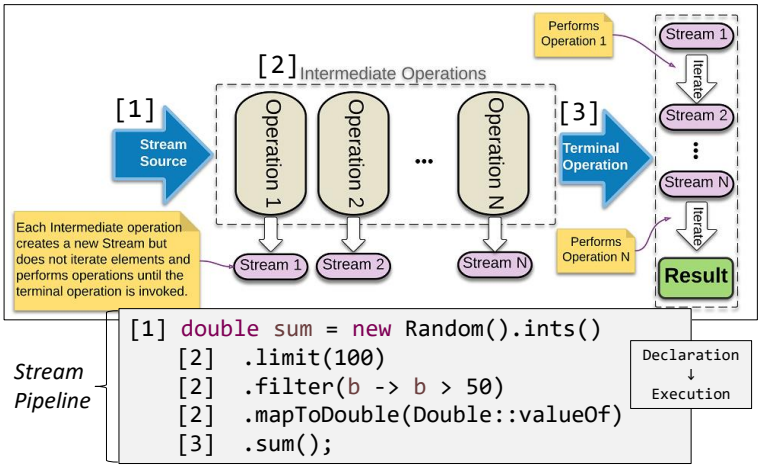
- 순차 혹은 병렬 집계연산 하는 연산의 순서, 파이프라인
  - 함수형 프로그래밍 연산을 지원하는 클래스의 모음 패키지
  - Collection에 대한 Map-reduce 변환 연산 지원
- 주요 특징
  - 저장 없는: 최대한 임시저장 하지 않고 바로 다음 연산자로 전달
  - 완전한 함수적으로: 원본을 수정하지 않음
  - 지연 탐색하는: stream연산은 터미널연산자에서 만 실행
  - 가능한 제한 없는 크기: 제한없는 크기, Short-circuiting 연산범위 특징

```
double sum = new Random().ints()
    .limit(100)
    .filter(b -> b > 50)
    .mapToDouble(Double::valueOf)
    .sum();
```

2

Workflow

- Stream은 3가지 모듈로 구성됩니다
  - Stream source, Intermediate Operation, Terminal operation



Stream Pipeline

```
[1] double sum = new Random().ints()
    [2]  .limit(100)
    [2]  .filter(b -> b > 50)
    [2]  .mapToDouble(Double::valueOf)
    [3]  .sum();
```

Declaration ↓ Execution

https://goo.gl/nRjDEq The stream machine

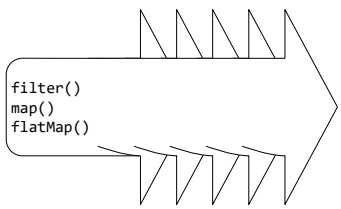
3

Stream source

15: 40 에 시작하겠습니다

Source	Method
Stream Static factory methods	Stream.of() IntStream.range() Stream.iterator()
Arrays	stream(Object[])
Collection	stream() parallelStream()
BufferedReader	BufferedReader.lines()
Files	Files.find() Files.list() Files.line()
Random	Random.ints()
Others	BitStet.stream() Pattern.splitAsStream() JarFile.

```
Stream Source
stream()
of()
empty()
iterate()
generate()
lines()
```




```
Terminal
reduce()
collect()
sum()
groupBy()
partitionBy()
reducing()
```

4

Stream source

- Stream static factory methods
  - Empty Stream, A sequence of item
  - From the array
  - Iterator
    - Stream<T> iterate(T seed, UnaryOperator<T> f)[8]
    - Stream<T> iterate(T seed, Predicate<? super T> hasNext, UnaryOperator<T> next)[9]
  - Stream<T> generate(Supplier<? extends T> s)

```
Stream<Integer> e = Stream.empty();
Stream<Integer> s = Stream.of(1, 2, 3);

String[] arr = Locale.getISOCountries();
Stream<String> sa = Stream.of(arr);

Stream<Integer> i = Stream.iterate(0, i -> i);
i = Stream.iterate(0, UnaryOperator.identity());

Stream<String> g = Stream.generate(() -> "Echo");
Stream<Double> g2 = Stream.generate(Math::random);
```

5

Stream source

- Arrays
  - static <T> Stream<T> stream(T[] array)[8]
  - static <T> Stream<T> stream(T[] array, int startInclusive, int endExclusive)[8]
- Collection
  - default Stream<E> stream()
  - default Stream<E> parallelStream()

```
String[] arr = Locale.getISOCountries();
Stream<String> sa2 = Arrays.stream(arr);
Stream<String> sa3 = Arrays.stream(arr, 0, 10);

Stream<Integer> fl = List.of(1, 2, 3).stream();
Stream<Integer> fs = Set.of(1, 2, 3).stream();
Stream<Entry<Integer, String>>
     = Map.of(1,"First", 2, "Second").entrySet().stream();
```

6

도전하세요!

- 다음 조건으로 Stream을 만들어 보세요!
  1. Stream<String> ss: 초기값 "It's me"에서 뒤에 문자"+"를 계속 붙이는
  2. Stream<BigInteger> bs: 초기값 2 에서 이후 그 값에 x2 하는
  3. Stream<Double> rd: Math.random()값을 연속해서 발생시키는
  4. Stream<Integer> seq: 나열된 데이터 79, 68, 55, 59, 77로
  5. Stream<Double> sa: 배열 myds = {0.0466, 0.5751, 0.6599}에서 index 1, 2 값으로만
  6. Stream<Integer> pis: 리스트 ints에서 병렬스트림

```
Stream<String> ss = ?;
Stream<BigInteger> bs = ?;
Stream<Double> rd = ?;
Stream<Integer> seq = ?;
double myds[] = new double[] {0.0466, 0.5751, 0.6599};
DoubleStream sa = ?;

List<Integer> ints = new ArrayList<>(){{ add(-1387513903);
add(164529915);}};
Stream<Integer> pis = ?;
```

7

From Files

16:30 시작하겠습니다

- Files
  - File, Directory를 조작하는 static method의 집합 Helper method
  - File 연산을 수행하는데 관련 File system provider에 맞춰 동작
- Stream<String> Files.lines(Path path) [8]
- DirectoryStream<Path> Files.newDirectoryStream(Path dir)[8]
- Stream<Path> Files.walk(Path start, FileVisitOption… options)[8]
- Stream<Path> list(Path dir)[8]
- Stream<Path> find(Path start, int maxDepth, BiPredicate<Path,BasicFileAttributes> matcher, FileVisitOption... options)[8]

```
String dir = "C:\\Windows\\System32\\drivers\\etc";
String filename = "\\hosts";
String fpath = dir + filename;
try (Stream<String> lines = Files.lines(Paths.get(fpath))) {
    lines.forEach(System.out::println);
} catch (IOException e) {
    e.printStackTrace();
}
```

8

From Pattern

- Pattern
  - 지정된 문자열을 정규표현식으로 Compile한 정규표현식 임의의 문자열과 일치하는 지 검사하는데 주로 사용
- Pattern.splitAsStream(CharSequence input)[8]

```
String ip = "10.221.51.2";
Pattern.compile("\\.").splitAsStream(ip).
     forEach(System.out::println);
```

9

Simple Terminal Collect

- Mutable reduction 연산자
  - 주어진 Collector에 따라 데이터 Reduction
  - Collect.to?()는 (Supplier, BiConsumer, BiConsumer)를 캡슐화 하여 제공
- <R,A> R Collector.collect(Collector<? super T,A,R> collector)
  - T – Element type
  - A - the mutable accumulation type
  - R – type of the result

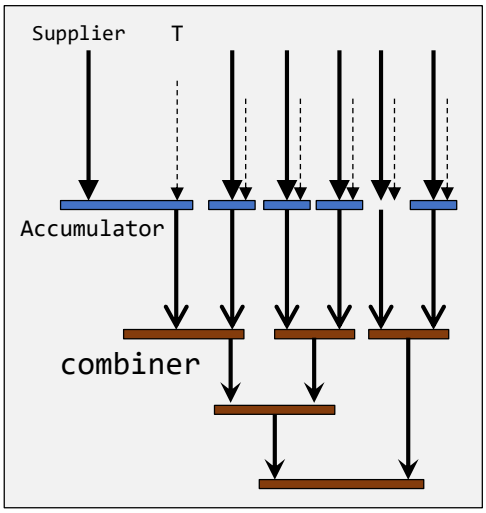
```
List<Integer> lst = Stream.of(1, 2, 3)
   .collect(Collectors.toList());

Set<Integer> set = Stream.of(1, 2, 3)
   .collect(Collectors.toSet());
```

11

Simple Terminal Collect

- Collect의 기본연산자
  - <R,A> R collect(Collector<? super T,A,R> collector)
  - <R> R collect(Supplier<R> supplier, BiConsumer<R,? super T> accumulator, BiConsumer<R,R> combiner)
- Supplier
  - Result container 생성
- Accumulator
  - Element와 임시Container 결합
- Combiner
  - 두 임시 Container 결합



12

Simple Terminal Collect

- Method Reference Expression로 간결한 표현가능
  - Supplier: Constructor reference
  - Accumulator: Method Reference Expression type 3
  - Combiner: Method Reference Expression type 3

```
List<Integer> lst = Stream.of(1, 2, 3)
    .collect(
        () -> new LinkedList<>(),
        (l, e) -> l.add(e),
        (l, b) -> l.addAll(b)
);
```

13

정리

- Stream
- Source
- Files
- Patterns
- Simple Terminal
- Simple Terminal Collect

14

Intermediate operations

sanghyuck.na@lge.com

filter()

map()

flatmap()

15

Intermediate Operation

- 데이터 변환 연산자
  - Pipeline에서 터미널연산을 실행하기 전 까지는 작업
  - JVM이 최적화작업을 통해 실제 정의한 연산은 빠질 수 있습니다

연산	반환 Stream 특징	분류
filter	주어진 조건(Predicate)에 일치하는 요소만 선택	Stateless
map	스트림의 요소에 주어진 Mapping함수를 적용	Stateless
flatMap	스트림의 각 요소를 주어진 Mapping함수를 적용한 Mapped stream변환	Stateless
limit	주어진 최대 데이터 개수 제한	Stateless
skip	주어진 첫 n개 요소는 버리고 그 이후 데이터 발생	Stateless
peek	주어진 action을 데이터에 적용하지만 수정은 없음	Stateless
concat	Stream 2개를 1개로 합침	Stateless
distinct	데이터의 중복을 제거	Stateful
sorted	주어진 Comparator에 따라 정렬	Stateful

16

Filter

- 주어진 조건에 맞는 데이터만 통과
  - T : 입력 데이터 타입
  - 주어진 조건(Predicate)에 일치하는 요소만 선택한 Stream반환
- 조건식 Predicate<T>
  - 다음 스트림으로 전달된다면 true, 그렇지 않으면 false 반환하여 데이터필터링 수행

```
Stream<T> filter(Predicate<? super T> predicate)

interface Predicate<T> {
    boolean test(T t);
}
```

```
Predicate<Integer> p = i -> i > 1;
Stream <Integer> s = Stream.of(1, 2, 3).filter(p)
```

17

Filter

- 논리 부정
  - Predicate<T> negate()
  - Predicate<T> not(Predicate<? super T> target)[11]
- 논리 연산
  - Predicate<T> and(Predicate<? super T> other)
  - Predicate<T> or(Predicate<? super T> other)

```
Predicate<Integer> p = i -> i > 1;
Stream.of(1, 2, 3).filter(p.negate());

Stream.of(1, 2, 3).filter(Predicate.not(p));

Stream.of(1, 2, 3).filter(p.and(e -> e > 2));

Stream.of(1, 2, 3).filter(p.or(e -> e == 1));
```

18

도전하세요!

- 리스트 in에서 "K" 또는 "C" 로 시작하는 나라이름 개수를 구하세요

```
List<String> in = List.of(Locale.getISOCountries());

long cnt = ?
```

- 다음 리스트 lines에서 "mkyong"가 아닌 아이템 개수를 구하세요

```
List<String> lines = Arrays.asList("spring", "node",
"mkyong");
long cnt = ?
```

19

도전하세요!

- 다음 리스트 customers에서 적립금 points가 >100 면서 이름이 "Charles"인 고객수를 구하세요

```
class Customer {
    String name; int points;

    Customer(String name, int points) {
        this.name = name; this.points = points;
    }
    int getPoints() { return this.points; }
    String getName() { return this.name; }
}

List<Customer> customers = List.of(new Customer("John P.", 15),
    new Customer("Sarah M.", 200),
    new Customer("Charles B.", 150), new Customer("Mary T.", 1));

long cnt = ?
```

20

Map

- 주어진 mapper로 어떠한 결과 변환
  - Mapper함수를 적용한 결과 스트림 반환
  - DoubleStream mapToDouble(ToDoubleFunction<? super T> mapper)
  - IntStream mapToInt(ToIntFunction<? super T> mapper)
  - LongStream mapToLong(ToLongFunction<? super T> mapper)
- 변환자 mapper
  - T는 함수의 입력 데이터타입, R은 함수의 리턴 데이터타입

```
<R> Stream<R>
    map(Function<? super T,? extends R> mapper)

interface Function<T,R> {
        R apply(T t)
}
```

```
Function<String, Integer> f = (s)->Integer.parseInt(s);
Stream <Integer> s = Stream.of("1", "2", "3").map(f);
s.forEach(System.out::println);
```

21

Map

- 동치
  - <T> Function<T,T> identity()
- 전치
  - <V> Function<V,R> compose(Function<? super V,? extends T> before)
- 후치
  - <V> Function<T,V> andThen(Function<? super R,? extends V> after)

```
Function<String, String> identity = (s)-> s;
Stream.of("1", "2", "3").map(identity)

Function<String, String> mapper = s -> "(" + s + ")";
Stream.of("1", "2", "3").map(mapper.compose(s -> "-" + s))

Stream.of("1", "2", "3").map(mapper.andThen(s -> "+" + s))
```

22

도전 하세요!

- 다음 리스트 cos의 각 문자열을 첫 글자만 대문자하고 나머지 글자는 모두 소문자로 변환하여 화면에 모두 출력하세요

```
List<String> cos = List.of(Locale.getISOCountries());
cos ??? .forEach(System.out::println);
```

- 다음 리스트 d에서 짝수 만 x 3을 한 후 화면에 출력하세요

```
List<Integer> d = Arrays.asList(3, 6, 9, 12, 15);
?
```

- 다음 리스트 s의 문자열을 정수로 변환하여, 0부터 그 정수까지 난수를 발생시키고, 그 총 합계를 화면에 출력하세요
  - 참고: int Random.nextInt(int bound)

```
List<String> s = List.of("3", "6", "9");
Random r = new Random();
int sum = ?
```

23

도전 하세요!

- 다음 리스트customers에서 고객의 적립금(points)에 대한 평균을 구하세요

```
class Customer {
    String name; int points;

    Customer(String name, int points) {
        this.name = name; this.points = points;
    }
    int getPoints() { return this.points; }
    String getName() { return this.name; }
}

List<Customer> customers = List.of(new Customer("John P.", 15),
    new Customer("Sarah M.", 200),
    new Customer("Charles B.", 150), new Customer("Mary T.", 1));


?
```

24

map In Primitive Stream

- Autoboxing의 성능지연을 보완

```
Stream<R> map(Function<? super T,? extends R> mapper)

DoubleStream
        mapToDouble(ToDoubleFunction<? super T> mapper)
IntStream
        mapToInt(ToIntFunction<? super T> mapper)
LongStream
        mapToLong(ToLongFunction<? super T> mapper)
```

25

FlatMap

- 주어진 mapper로 Stream만 변환
  - 원본 스트림의 각 요소에 1:n 변환을 적용시 유용
  - 결과값은 단일 스트림으로 차원축소(flattening)에 주로 사용

```
<R> Stream<R> flatMap(Function
    < ? super T, ? extends Stream<? extends R>> mapper)
```

```
List<Integer> alst = List.of(1, 2);
List<Integer> blst = List.of(3, 4);

Stream<List<Integer>> all = Stream.of(alst, blst);
Stream<Integer> lst = all.flatMap(
                      (List<Integer> e) -> e.stream());
```

```
Integer[][] arrays = new Integer[][] {
    { 1, 2, 3, 4, 5 }, { 3, 4, 5, 6 }};

Stream<Integer[]> s = Stream.of(arrays);
Stream<Integer> t = s.flatMap(
                  (Integer[] one) -> Arrays.stream(one));
```

26

도전하세요!

- 클래스 Customer는 이름 name과 희망제품 wlst로 구성됩니다.
- 모든 고객의 wlst합친 리스트 allWlst를 만들려고 합니다. flatmap()을 사용해서 다음 "?" 을 완성하세요.

```
class Customer {
    String name; List<String> wlst;
    Customer(String name, List<String> wlst) {
        this.name = name;
        this.wlst = wlst;
    }
}

List<Customer> clst = List.of(
    new Customer("Jack", List.of("Car", "Home")),
    new Customer("Ellin", List.of("Pen", "Desk")),
    new Customer("Nilson", List.of("Bag", "Phone")));
Stream<String> streamWs = ??;
List<String> allWlst = streamWs.collect(Collectors.toList());
```

27

도전하세요!

- 다음 리스트 strs는 문자열로 구성되어 있습니다. strs을 단어 단위로 분할하여 단어리스트 wordsLists를 만들려고 합니다. flatMap()을 사용해서 다음 "?"을 완성하세요

```
List<String> strs = List.of(
    "Imagine driving down the highway at 70 miles ",
    "per hour, when suddenly the wheel turns hard right. ",
    "You crash. And it was because someone hacked your
    car.");

Stream<String> words = ?
List<String> wordsList = words.collect(Collectors.toList());
```

28

flatmap In Primitive Stream

- Autoboxing의 성능지연을 보완

```
<R> Stream<R> flatMap(Function<? super T
            ,? extends Stream<? extends R>> mapper)


DoubleStream flatMapToDouble(Function<? super T
            ,? extends DoubleStream> mapper)

IntStream flatMapToInt(Function<? super T
             ,? extends IntStream> mapper)

LongStream flatMapToLong(Function<? super T
            ,? extends LongStream> mapper)
```

29

정리

- filter()
- map()
- flatMap()

30