Effective Java

Item 47. 반환 타입으로는 스트림보다 컬렉션이 낫다

Presentation by. RAZEL

목차

- 원소 시퀀스
- 스트림 사용
- 컬렉션 사용
- 요약 정리

원소 시퀀스

원소 시퀀스

- 원소 시퀀스란?

순서가 있는 데이터 요소의 나열

EX) 1, 2, 3, 4, 5…

자바 7까지의 원소 시퀀스 반환 타입

1) Collection 인터페이스
   1. 기본적으로 사용

2) Iterable 인터페이스
   1. for-each 문에서만 쓰이는 경우
   2. 원소 시퀀스가 일부 컬렉션 메서드를 구현할 수 없는 경우

3) 배열
   1. 반환 원소가 기본 타입인 경우
   2. 성능에 민감한 경우

스트림 사용

스트림 사용

- 스트림은 반복(iteration)을 지원하지 않는다.

```java
public interface Iterable<T> {
    Iterator<T> iterator();

    default void forEach(Consumer<? super T> action) {
        Objects.requireNonNull(action);
        for (T t : this) {
            action.accept(t);
        }
    }

    default Spliterator<T> spliterator() {
        return Spliterators.spliteratorUnknownSize(iterator(), 0);
    }
}
```

```java
public interface Stream<T> extends BaseStream<T, Stream<T>> {
    {...}
    void forEach(Consumer<? super T> action);
    {...}
}
```

```java
public interface BaseStream<T, S extends BaseStream<T, S>> extends AutoCloseable {
    Iterable<T> iterator();
    Spliterator<T> spliterator();
     {...}
}
```

스트림 사용

스트림을 반환하는 메서드의 반환 값을 받아서
반복을 사용하고 싶다면 어떻게 되지?

스트림 사용 - 스트림 반환 시 반복을 사용

```
1  for (ProcessHandler ph : (Iterable<ProcessHandler>) ProcessHandler.allProcess()::iterator) {
2
3  }
```

Stream과 iterable 중개 메서드

```
1  public static <E> Iterable<E> iterableOf(Stream<E> stream) {
2      return stream::iterator;
3  }
```

```
1  for (ProcessHandler ph : iterableOf(ProcessHandler.allProcess())) {
2  }
```

스트림 사용 - 반복을 반환 시 스트림 사용

Stream과 iterable 중개 메서드

```java
public static <E> Stream<E> streamOf(Iterable<E> iterable) {
    return StreamSupport.stream(iterable.spliterator(), false);
}
```

```java
Set<String> fruits = Set.of("apple", "banana");

for (Set<String> subset : PowerSet.of(fruits)) {
    System.out.println(subset); // ✅ Iterable로 반복
}

PowerSet.streamOf(fruits)
        .filter(s -> s.size() == 1)
        .forEach(System.out::println); // ✅ Stream으로도 사용 가능
```

스트림 사용

- 스트림은 일회성이다.

```
1  Stream<String> stream = fruits.stream();
2  stream.forEach(System.out::println);
3  stream.count(); // Stream은 한 번만 사용할 수 있습니다.
```

🤔스트림을 어디서 사용할 수 있지?

스트림 사용

- 데이터를 즉시 처리하고 한번만 순회하는 경우

  map, filter, reduce 등의 **연산을 바로 쓸 수 있다.**

- 지연 처리(lazy evaluation)가 중요한 경우

  계산을 미뤄 **비용을 절감해** 성능을 향상시킬 수 있다.

- 데이터 양이 많은 경우

  원소를 필요할 때 하나씩 처리하므로 **메모리 사용을 줄일 수 있다.**

컬렉션 사용

컬렉션 사용

```java
public interface Collection<E> extends Iterable<E> {
    default Stream<E> stream() {
        return StreamSupport.stream(spliterator(), false);
    }

    default Stream<E> parallelStream() {
        return StreamSupport.stream(spliterator(), true);
    }
}
```

iterable의 하위 타입!

Stream 메서드 지원 중!

컬렉션 사용

```java
List<String> names = List.of("Alice", "Bob", "Charlie");

names.stream()
     .map(String::toUpperCase)
     .forEach(System.out::println);
```

```java
List<String> names = List.of("Alice", "Bob", "Charlie");

for (String name : names) {
    System.out.println(name.toUpperCase());
}
```

반환 타입으로 컬렉션을 사용할 때

스트림이 반환 타입인 경우보다 훨씬 깔끔하게

사용할 수 있고, 재사용성 가능하다.

요약 정리

요약 정리

- 스트림(Stream)은 반복(iteration)에 적합하지 않다.
- 컬렉션(Collection)은 데이터를 보관하고, 반복하고, 재사용할 수 있다.
- 따라서 API의 반환 타입으로는 Stream보다 Collection이나 List, Set을 사용하는 것이 일반적으로 낫다.

- The End -

간단 실습) Stream과 List의 성능 차이