2017.12.12 ~ 2017.12.15

롯데정보통신

JAVA SE PROGRAMMING 고급

문성훈

목차

01. 객체지향 기본 개념

02. 객체지향 설계 원칙

03. ClassLoader

04. Java8 Lambda

05. Java Collection (Stream)

06. Java NIO

05. JAVA COLLECTION (STREAM)

❖ Stream 기본

- Java 7 이전까지는 List<String>와 같은 Collection에서 요소를 순차적으로 처리하기 위해 일반적으로 Iterator 반복자를 사용.

- Stream은 Java 8부터 추가된 Collection(배열 포함)의 저장 요소를 하나씩 참조해서 람다식으로 처리 할 수 있도록 해주는 반복자.

- Stream은 Iterator와 비슷한 역할을 하는 반복자이지만 다음과 같은 차이점이 존재
  - 람다식으로 요소 처리 코드를 제공하는 점
  - 내부 반복자를 사용하므로 병렬 처리가 쉽다는 점
  - 중간 처리와 최종 처리 작업을 수행할 수 있다는 점

❖ Stream 특징

- 람다식으로 요소 처리 코드를 제공.

  - Stream이 제공하는 대부분의 요소 처리 method는 함수적 인터페이스 매개 타입을 가지기 때문에 람다식 또는 Method Reference를 이용해서 요소 처리 내용을 파라미터로 전달할 수 있음.

  - 예제 작성)
    Collection에 저장된 Student를 하나씩 가져와 학생 이름과 수학 성적을 콘솔에 출력하도록 forEach() method를 이용해서 작성해 보세요.

❖ Stream 특징(cont.)

- 병렬 처리가 쉽다.

  - 외부 반복자(external iterator)란 개발자가 코드로 직접 Collection의 요소를 반복해서 가져오는 코드 패턴. (index를 이용하는 for문, Iterator를 이용하는 while 문은 모두 외부 반복자를 이용)

  - 내부 반복자(internal Iterator)는 Collection 내부에서 요소들을 반복시키고, 개발자는 요소당 처리해야 할 코드만 제공하는 코드 패턴. (Collection 내부에서 어떻게 요소를 반복시킬 것인가는 Collection에게 맡겨두고, 개발자는 요소 처리 코드에만 집중)

  - 내부 반복자는 멀티 코어 CPU를 최대한 활용하기 위해 요소들을 분배시켜 병렬 작업을 할 수 있음.

  - 예제작성)
    순차 처리 Stream과 병렬 처리 Stream을 이용할 경우 사용된 Thread의 이름 출력

❖ Stream 특징(cont.)

- 중간 처리와 최종 처리 기능 제공

  - Stream은 Collection 요소에 대해 중간 처리와 최종 처리를 수행할 수 있는데,
    중간 처리에서는 Mapping, Filtering, Sorting 등을 수행
    최종 처리에서는 Looping, Counting, Average, Sum 등의 집계 처리를 수행

  - 예제 작성)
    학생 리스트에서 남자 학생 중 수학성적이 70점 이상인 학생들에 대해 영어 성적의 평균을 구하세요.

❖ Stream 종류

- 모든 Stream은 자바 8부터 추가된 java.util.stream 패키지에 포함.

  - BaseStream이 상위 인터페이스로 일반적으로 사용되는 Stream은 이 인터페이스를 상속.

  - 하위 Stream 인 Stream, IntStream, LongStream, DoubleStream이 직접적으로 이용되는 Stream.
    (Stream은 객체 요소를 처리하는 Stream 이고, IntStream, LongStream, DoubleStream 은 각각 기본 타입인 int, long, double 요소를 처리하는 Stream)

  - 이 Stream 인터페이스의 구현 객체는 다양한 소스로부터 얻을 수 있는데 주로 Collection과 배열에서 얻지만 정수의 범위나 랜덤값, 파일과 같이 다양한 소스로부터 Stream 구현 객체를 얻을 수도 있음.

  - 예제작성)
    다양한 소스로부터 Stream을 생성해보자.

❖ Stream Pipeline

- Reduction
  대량의 데이터를 가공해서 축소하는 것으로 데이터의 합계, 평균값, 카운팅, 최대값, 최소값 등이 대표적인 Reduction 결과물.

- Collection의 요소를 reduction의 결과물로 바로 집계할 수 없을 경우에는 집계하기 좋도록 filtering, mapping, sorting, grouping 등의 중간 처리가 필요.

- Stream은 데이터의 filtering, mapping, sorting, grouping 등의 중간 처리와 합계, 평균, 카운팅, 최대값, 최소값 등의 최종 처리를 pipeline으로 해결.

❖ Stream Pipeline (cont.)

- Pipeline은 여러 개의 Stream이 연결되어 있는 구조
- Pipeline 에서 최종 처리를 제외하고는 모두 중간 처리 Stream입니다.
- 중간 Stream이 생성될 때 요소들이 바로 중간 처리(filtering, mapping, sorting)되는 것이 아니라 최종 처리가 시작되기 전까지 중간 처리는 lazy.
- 최종 처리가 시작되면 비로소 Collection의 요소가 하나씩 중간 Stream에서 처리되고 최종 처리까지 진행. Stream 인터페이스에는 filtering, mapping, sorting 등의 많은 중간 처리 method가 존재하고 이 method들은 다시 중간 처리된 Stream을 리턴.

❖ Stream 제공 method

- Filtering (distinct(), filter())
  - Filtering은 중간 처리 기능으로 요소를 걸러내는 역할.
  - 필터링 method인 distinct()와 filter() method는 모든 Stream이 가지고 있는 공통 method.
  - distinct() : 중복을 제거.
    Stream의 경우 Object.equals(Object)가 true 이면 동일한 객체로 판단하고 중복 제거.
    IntStream, LongStream, DoubleStream은 동일값일 경우 중복 제거.
  - filter() : 인자로 주어진 Predicate가 true를 리턴하는 요소만 필터링.
  - 예제 작성)

❖ Stream 제공 method (cont.)

- Mapping

  - 중간 처리 기능으로 Stream의 요소를 다른 요소로 대체.
  - 제공하는 mapping method는 flatMapXXX()와 mapXXX(), 그리고 asDoubleStream(), asLongStream(), boxed().
  - flatMapXXX() method는 요소를 대체하는 복수 개의 요소들로 구성된 새로운 Stream을 리턴.
  - mapXXX() method는 요소를 대체하는 요소로 구성된 새로운 Stream을 리턴.
  - asDoubleStream() method는 IntStream의 int 요소 또는 LongStream의 long 요소를 double 요소로 타입 변환해서 DoubleStream 을 리턴.
  - 마찬가지로 asLongStream() 메소드는 IntStream의 int 요소를 long 요소로 타입 변환해서 LongStream을 리턴.
  - boxed() 메소드는 int, long, double 요소를 Integer, Long, Double 요소로 Boxing해서 Stream을 리턴.

❖ Stream 제공 method (cont.)

- Sort
  - Stream은 요소가 최종 처리되기 전에 중간 단계에서 요소를 정렬해서 최종 처리 순서를 변경할 수 있음.
  - 객체 요소일 경우에는 클래스가 Comparable을 구현하지 않으면 sorted() 메소드를 호출했을 때 ClassCastException이 발생하기 때문에 Comparable을 구현한 요소에서만 sorted() method를 호출.
  - 예제 작성)

❖ Stream 제공 method (cont.)

- Looping

  - peek(), forEach()
  - 이 method는 기능에서는 동일하지만, peek() 는 중간 처리 method이고, forEach()는 최종 처리 method
  - peek() method는 중간 처리 단계에서 전체 요소를 looping하면서 추가적인 작업을 하기 위해 사용.
  - 최종 처리 method가 실행되지 않으면 지연되기 때문에 반드시 최종 처리 method가 호출되어야 동작.
  - 반면, forEach() 는 최종 처리 method이기 때문에 파이프라인 마지막에 looping하면서 요소를 하나씩 처리. 따라서 이후에 sum() 과 같은 다른 최종 method를 호출하면 안됨.

  - 예제 작성)

❖ Stream 제공 method (cont.)

- Match

  - 최종 처리 단계에서 요소들이 특정 조건에 만족하는지 조사할 수 있도록 세 가지 matching method 제공

  - allMatch() : 모든 요소들이 파라미터로 주어진 Predicate의 조건을 만족하는지 조사
  - anyMatch() : 최소한 한 개의 요소가 파라미터로 주어진 Predicate의 조건을 만족하는지 조사.
  - noneMatch() : 모든 요소들이 파라미터로 주어진 Predicate의 조건을 만족하지 않는지 조사.

  - 예제 작성)

❖ Stream 제공 method (cont.)

- Aggregation

  - 최종 처리 기능으로 요소들을 처리해서 카운팅, 합계, 평균, 최대값, 최소값 등과 같이 하나의 값으로 산출.
  - Aggregation은 대량의 데이터를 가공해서 축소하는 Reduction.
  - 이런 aggregation method는 java.util 패키지의 Optional, OptionalDouble, OptionalInt, OptionalLong 클래스 타입의 객체를 리턴.
  - Optional 클래스는 단순히 집계 값만 저장하는 것이 아니라,
    집계 값이 존재하지 않을 경우 디폴트 값을 설정할 수도 있고, 집계 값을 처리하는 Consumer도 등록할 수 있다.
  - 예제 작성)

❖ Stream 제공 method (cont.)

- Custom Aggregation
  - Stream은 기본 집계 method인 sum(), average(), count(), max(), min()을 제공
  - 추가적으로 개발자가 프로그램화해서 다양한 집계 결과물을 만들 수 있도록 reduce() method를 제공.
  - Stream에 요소가 전혀 없을 경우 디폴트 값인 identity 파라미터가 리턴.
  - 예제 작성)

❖ Stream 제공 method (cont.)

- Collect
  - Stream은 요소들을 filtering 또는 mapping 후 요소들을 수집하는 최종 처리 method인 collect()를 이용하여 필요한 요소만 새로운 collection에 담을 수 있음.
  - 예제 작성)

❖ Stream 병렬처리

● 주의할 점

- ➢ 병렬 처리가 순차 처리보다 항상 실행 성능이 좋은가?
- ➢ 무엇을 살펴보아야 하나?
- ➢ 요소의 수와 요소당 처리 시간
- ➢ collection에 요소의 수가 적고 요소당 처리 시간이 짧으면 순차 처리가 빠름 (Thread 추가 비용발생)
- ➢ 스트림 소스의 종류
- ➢ ArrayList, 배열은 인덱스로 요소를 관리하기 때문에 병렬 처리 시간이 절약되지만 Set, Tree계열은 요소
- ➢ 분리에 시간이 걸려서 상황을 보고 판단해야 함.
- ➢ 싱글 코어 CPU일 경우에는 당연히 순차 처리가 빠름.

Thanks for Listening