

1. 자바 역사

모던 자바 인 액션 Chapter 1: 자바 8, 9, 10, 11에서 무슨 일이 일어나고 있을까?

자바의 역사상 가장 큰 변화, 자바 8이 가져온 혁신적인 변화들을 알아보자.

개요

1996년에 자바 개발 키트(JDK 1.0)가 발표된 이후로 수많은 프로그래머들이 크고 작은 프로젝트에서 자바를 적극적으로 활용해왔다. 자바 1.1부터 자바 7에 이르기까지 자바는 새로운 기능과 더불어 계속 발전해왔으며, 2018년 3월에는 자바 10, 9월에는 자바 11이 릴리스되었다.

1.1 역사의 흐름

자바 8: 역사상 가장 큰 변화

자바 역사를 통틀어 가장 큰 변화는 **자바 8**에서 일어났다. 자바 9에서도 중요한 변화가 있었으나 자바 8만큼 획기적이거나 생산성이 바뀌는 것은 아니었다. 자바 10에서는 타입 추론과 관련해 약간의 변화만 일어났다.

코드 비교: 자바 7 vs 자바 8

자바 7 이전의 방식:

```
Collections.sort(inventory, new Comparator<Apple>() {  
    public int compare(Apple a1, Apple a2) {  
        return a1.getWeight().compareTo(a2.getWeight());  
    }  
});
```

자바 8의 방식:

```
inventory.sort(comparing(Apple::getWeight));
```

자바 8을 이용하면 자연어에 더 가깝게 간단한 방식으로 코드를 구현할 수 있다.

하드웨어 변화의 영향

멀티코어 CPU 대중화와 같은 하드웨어적 변화도 자바 8에 영향을 미쳤다. 요즘 랩톱이나 데스크톱에는 듀얼 혹은 쿼드 코어 이상을 지원하는 CPU가 내장되어 있다. 하지만 지금까지 대부분의 자바 프로그램은 코

어 중 하나만을 사용했다.

자바 8 이전에는 나머지 코어를 활용하려면 스레드를 사용해야 했지만, 스레드는 관리하기 어렵고 많은 문제가 발생할 수 있다는 단점이 있었다.

자바의 진화 과정

- **자바 5** : 스레드 풀, 병렬 실행 컬렉션 등의 도구 도입
- **자바 7** : 병렬 실행에 도움을 줄 수 있는 포크/조인 프레임워크 제공 (하지만 개발자가 활용하기 어려웠음)
- **자바 8** : 병렬 실행을 새롭고 단순한 방식으로 접근할 수 있는 방법 제공
- **자바 9** : 리액티브 프로그래밍이라는 병렬 실행 기법 지원

자바 8의 핵심 기술

1. **Stream API**
2. 메소드에 코드를 전달하는 기법
3. 인터페이스의 디폴트 메소드

자바 8은 데이터베이스 질의 언어에서 표현식을 처리하는 것처럼 병렬 연산을 지원하는 **Stream**이라는 새로운 API를 제공한다. 스트림을 이용하면 에러를 자주 일으키며 멀티코어 CPU를 이용하는 것보다 비용이 훨씬 비싼 키워드 `synchronized` 를 사용하지 않아도 된다.

 자바 8 기법은 함수형 프로그래밍에서 위력을 발휘한다.

1.2 왜 아직도 자바는 변화하는가: 자바의 멀티코어 병렬성

프로그래밍 언어 생태계

지금까지 많은 언어가 쏟아져나왔고, 학계에서는 프로그래밍 언어가 마치 생태계와 닮았다고 결론을 내렸다. 새로운 언어가 등장하면서 진화하지 않은 기존 언어는 사장되었다.

예를 들어 C, C++은 프로그래밍 안전성은 부족하지만 작은 런타임 풋프린트 덕분에 운영체제와 다양한 임베디드 시스템에서 여전히 인기를 끌고 있다.

1.2.1 프로그래밍 언어 생태계에서 자바의 위치

자바는 출발이 좋았으며, 처음부터 많은 유용한 라이브러리를 포함하는 잘 설계된 객체지향 언어로 시작했다. 자바는 처음부터 스레드와 락을 이용한 동시성도 지원했다.

코드를 JVM 바이트코드로 컴파일하는 특징 때문에 자바는 인터넷 애플릿 프로그램의 주요 언어가 되었다. 일부 애플리케이션에서는 JVM에서 실행되는 경쟁 언어인 스칼라, 그루비 등이 자바를 대체하기도 했다.

하지만 시간이 지나면서 프로그래머는 **빅데이터**(테라바이트 이상의 데이터 셋)라는 도전에 직면하면서 멀티코어 컴퓨터나 컴퓨팅 클러스터를 이용해서 빅데이터를 효과적으로 처리할 필요성이 커졌다.

1.2.2 스트림 처리

첫 번째 프로그래밍 개념은 ***“스트림 처리”***이다.

스트림의 정의

스트림이란 한 번에 한 개씩 만들어지는 연속적인 데이터 항목들의 모임이다. 이론적으로 프로그램은 입력 스트림에서 데이터를 한 개씩 읽어들이며, 마찬가지로 출력 스트림으로 데이터를 한 개씩 기록한다.

자바 8의 스트림 API

- 자바 8에는 `java.util.stream` 패키지에 스트림 API가 추가되었다
- 스트림 패키지에 정의된 `Stream<T>` 는 T 형식으로 구성된 일련의 항목을 의미한다
- 스트림 API는 파이프라인을 만드는 데 필요한 많은 메소드를 제공한다
- 우리가 하려는 작업을 고수준으로 추상화해서 일련의 스트림으로 만들어 처리한다

스트림의 장점

- 파이프라인을 이용해서 입력 부분을 여러 CPU 코어에 쉽게 할당할 수 있다
- 스레드라는 복잡한 작업을 사용하지 않으면서도 **공짜로 병렬성을 얻을 수 있다**

1.2.3 동작 파라미터화로 메소드에 코드 전달하기

두 번째 프로그래밍 개념은 **코드 일부를 API로 전달하는 기능**이다.

자바 8 이전에는 메소드를 다른 메소드로 전달하는 방법이 없었다. 하지만 자바 8에서는 메소드를 다른 메소드의 인수로 넘겨주는 기능을 제공한다. 이러한 기능을 이론적으로 **동작 파라미터화**라고 부른다.

1.2.4 병렬성과 공유 가변 데이터

세 번째 프로그래밍 개념은 **병렬성을 공짜로 얻을 수 있다**는 말에서 시작된다. 하지만 병렬성을 얻는 대신 스트림 메소드로 전달하는 코드의 동작 방식을 조금 바꿔야 한다.

안전한 코드 작성 조건

안전하게 실행할 수 있는 코드를 만들려면 **공유된 가변 데이터에 접근하지 않아야 한다**. 이러한 함수를 다음과 같이 부른다.

- 순수 함수
- 부작용 없는 함수
- 상태 없는 함수

프로그래밍 패러다임의 변화

- **함수형 프로그래밍** : 공유되지 않은 가변 데이터, 메소드, 함수 코드를 다른 메소드로 전달
- **명령형 프로그래밍** : 일련의 가변 상태로 프로그램을 정의

자바 8의 주요 변화

- 람다 (Lambda)
- 함수형 인터페이스 (Functional Interface)

- Stream API
- Optional 클래스
- 메소드 참조 / 생성자 참조
- 기본형 특화 함수형 인터페이스

1.2.5 자바가 진화해야 하는 이유

제네릭이 나타나고 틀에 박힌 Iterator 대신 for-each 루프를 사용할 수 있게 되었다. **고전적인 객체지향에서 벗어나 함수형 프로그래밍으로 다가섰다는 것이 자바 8의 가장 큰 변화이다.**

함수형 프로그래밍에서는 **우리가 하려는 작업이 최우선시되며**, 그 작업을 어떻게 수행하는지는 별개의 문제로 취급한다. 이를 도입함으로써 두 가지 프로그래밍 패러다임의 장점을 모두 활용할 수 있게 되었다.

“언어는 하드웨어나 프로그래머 기대의 변화에 부응하는 방향으로 변화해야 한다”

1.3 자바 함수

함수라는 용어는 메소드, 특히 정적 메소드와 같은 의미로 사용된다. 자바의 함수는 수학적 함수처럼 사용되며 부작용을 일으키지 않는 함수를 의미한다.

자바 8에서는 함수를 **새로운 값의 형식**으로 추가했다. 병렬 프로그래밍을 활용할 수 있는 스트림과 연계될 수 있도록 함수를 만들었기 때문이다.

일급 값 (First-Class Citizen)

프로그래밍 언어의 핵심은 값을 바꾸는 것이다. 역사적으로 그리고 전통적으로 프로그래밍 언어에서는 이 값을 “**일급 값**” 혹은 **“일급 시민”**이라고 부른다.

일급 값의 정의

프로그래밍 언어의 핵심 개념 중 하나로, 어떤 대상(값)이 얼마나 자유롭게 다뤄질 수 있는지를 나타낸다.

일급 값 조건

1. **변수에 할당 가능** – 값을 변수에 저장할 수 있어야 함
2. **함수의 인자로 전달 가능** – 함수의 매개변수로 넘길 수 있어야 함
3. **함수의 반환 값 사용 가능** – 함수가 그 값을 반환할 수 있어야 함

이급 시민 (Second-Class Citizen)

하지만 프로그램을 실행하는 동안 모든 구조체를 자유롭게 전달할 수는 없다. 이렇게 전달할 수 없는 구조체는 **“이급 시민”**이다.

이급 시민의 특징

- 변수에 담거나 함수의 인자로 넘길 수 없음
- 반환 값으로 쓸 수 없음

- 메소드, 클래스 등이 일급 자바 시민에 해당 (자바 8 이전)

1.3.1 메소드와 람다를 일급 시민으로

메소드를 사용하면 프로그래머가 활용할 수 있는 도구가 다양해지면서 프로그래밍이 수월해진다. 자바 8에 서는 메소드를 값으로 취급할 수 있는 기능이 스트림 같은 다른 자바 8 기능의 토대를 제공했다.

메소드 참조 (Method Reference)

자바 8 이전:

```
File[] hiddenFiles = new File(".").listFiles(new FileFilter() {  
    public boolean accept(File file) {  
        return file.isHidden();  
    }  
});
```

자바 8:


```
File[] hiddenFiles = new File(".").listFiles(File::isHidden);
```

자바 8의 `::` 연산자를 이용해 메소드 자체를 값처럼 전달할 수 있다.

람다 (Lambda): 익명 함수

메소드를 일급 값으로 취급할 뿐 아니라 람다를 포함하여 함수도 값으로 취급할 수 있다.

예를 들어 `(int x) -> x + 1` 처럼 인수 `x` 를 받아 `x+1` 을 반환하는 동작을 한 줄로 표현할 수 있다.

 람다 문법 형식으로 구현된 프로그램을 함수형 프로그래밍, 함수를 일급 값으로 넘겨주는 프로그램을 구현한다고 한다.

1.3.2 메소드 전달에서 람다로

메소드를 값으로 전달하는 것은 분명 유용한 기능이다. 한 번만 사용할 메소드는 따로 정의를 구현할 필요가 없다.

하지만 람다가 몇 줄 이상으로 길어진다면, 익명 람다보다는 코드를 수행하는 일을 잘 설명하는 이름을 가진 메소드를 정의하고 메소드 참조를 활용하는 것이 바람직하다.

1.4 스트림

거의 모든 자바 애플리케이션은 컬렉션을 만들고 활용한다. 하지만 컬렉션으로 모든 문제가 해결되는 것은 아니다. 예를 들어 필터링하고 결과를 그룹화해야 한다고 가정하면, 많은 기본 코드를 구현해야 한다. 그러나 스트림 API를 이용하면 이런 문제를 해결할 수 있다.

스트림 API 예제

```
Map<Currency, List<Transaction>> transactionsByCurrencies = transactions.stream()
    .filter((Transaction t) -> t.getPrice() > 1000)
    .collect(groupingBy(Transaction::getCurrency));
```

1.4.1 멀티스레딩은 어렵다

이전 자바 버전에서 제공하는 스레드 API로 멀티스레딩 코드를 구현해서 병렬성을 이용하는 것은 쉽지 않다. 스레드를 잘 제어하지 못하면 원치 않는 방식으로 데이터가 바뀔 수 있다.

자바 8의 해결책

자바 8은 스트림 API로 컬렉션을 처리하면서 발생하는 두 가지 문제를 해결했다.

1. 모호함과 반복적인 문제
2. 멀티코어 활용의 어려움

반복 패턴의 해결

- 주어진 조건에 따라 데이터를 필터링
- 데이터를 추출
- 데이터를 그룹화

포킹 단계 (Forking Stage)

두 CPU를 가진 환경에서 리스트를 필터링할 때:

1. 한 CPU는 리스트의 앞부분을 처리
2. 다른 CPU는 리스트의 뒷부분을 처리
3. 각각의 CPU는 자신이 맡은 절반의 리스트를 처리
4. 마지막으로 하나의 CPU가 두 결과를 정리

 이 과정은 분할 정복 알고리즘을 기반으로 한다.

컬렉션 vs 스트림

- 컬렉션 : 어떻게 데이터를 저장하고 접근할지에 중점
- 스트림 : 데이터에 어떤 계산을 할 것인지 묘사하는 것에 중점

1.5 디폴트 메소드와 자바 모듈

요즘은 외부에서 만들어진 컴포넌트를 이용해 시스템을 구축하는 경향이 있다. 그동안 자바에서는 평범한 패키지 집합(JAR)만 제공했기에 인터페이스 변경 시 구현 클래스를 모두 손봐야 하는 문제가 있었다.

자바 8, 9의 해결책

- 자바 9 : 모듈 시스템을 도입해 패키지 묶음을 명확히 구분

- **자바 8** : 인터페이스를 쉽게 확장할 수 있도록 디폴트 메소드 지원

기존 인터페이스의 문제점

인터페이스에 새로운 메소드를 추가하면, 이를 구현한 모든 클래스가 그 메소드를 구현해야 했다. 현실적으로는 불가능하거나 매우 고통스러운 작업이었다.

디폴트 메소드의 등장

자바 8은 인터페이스 내에 몸체가 있는 메소드를 선언할 수 있게 해주는 `default` 키워드를 도입했다. 덕분에 기존 구현을 건드리지 않고도 인터페이스를 확장할 수 있다.

1.6 함수형 프로그래밍에서 가져온 다른 유용한 아이디어

자바에 포함된 함수형 프로그래밍의 핵심적인 두 아이디어를 살펴봤다.

1. 메소드와 람다를 일급 값으로 사용
2. 가변 공유 상태가 없는 병렬 실행으로 안전하고 효율적인 코드 작성

`Optional<T>` 클래스

널 포인터 예외(NPE)를 피하도록 도와주는 컨테이너 객체.

- 값을 갖거나 갖지 않을 수 있음
- 값이 없는 상황을 어떻게 처리할지 명시적인 API 제공
- NPE를 방지하는 패턴을 독려

1.7 마치며

핵심 정리

- 자바 8은 프로그램을 더 효과적이고 간결하게 구현할 수 있는 새로운 개념과 기능을 제공한다.
- 기존 기법으로는 멀티코어 자원을 온전히 활용하기 어렵다.
- 함수는 일급 값이며, 스트림과 결합해 병렬성 확보에 용이하다.
- 디폴트 메소드를 통해 기존 인터페이스도 유연하게 확장할 수 있다.

 다음 장에서는 **동작 파라미터화**에 대해 더 자세히 알아보자!

태그

#Java8 #모던자바 #함수형프로그래밍 #스트림 #람다 #디폴트메소드