lambda expression

Date @2021/11/03

만약에 내 코드가 엄청 느리거나 가독성이 좋지 않아 복잡하면 다른 사람들이 쓰려고 할까?

더 좋은 방법으로 진행하는 것이 좋죠

1. 람다 표현식이 무엇인지 알 수 있다
2. 람다 표현식의 사용 방법과 응용할 수 있어야 한다.
3. 왜 람다 표현식을 사용해야 하는지 알 수 있다.

'구현해야 할 추상 메서드가 하나만 정의된 인터페이스' 일 때 가능하다.

각 언어별 람다 표현식

Java

'화살표' [→]

(매개변수목록)→{함수 몸체}

```
List<Map<String, Object>> result = new ArrayList<>();
boardRepository.findAll().forEach( boardList -> {
    Map<String, Object> obj = new HashMap<>();
    obj.put("uid", boardList.getUid());
    obj.put("title", boardList.getTitle());
    obj.put("contents", boardList.getContents());
    result.add(obj);
});
return result;
```

Python

lambda (augments) : (expression)

예 : lambda x : {리턴값}

```
# id로 seq 검색
algo_seq = list(map(lambda x: x['seq'], Type.objects.
```

Javascript

'화살표' [⇒]

(매개변수목록)⇒{함수 몸체}

```
axios.post(`${SERVER_URL}/apps/v1/login`,{
  'email': this.idInput,
  'password': this.passwordInput
})
  .then(res => {
    if(res.status !== 200){
      alert('로그인실패')
      return;
    }
    localStorage.setItem('email',res.data.userInfo[0].email)
```

람다 표현 식 문법 및 유의사항

Java

No arguments:	() -> System.out.println("SSAFY")
One argument:	s -> System.out.println(s)
Two arguments:	(x, y) -> x + y
explicit argument types:	(Interger x, Integer y) -> x + y

multiple statements:	(x, y) -> {
	System.out.println(x);
	System.out.println(y);
	System.out.println("싸피 사랑해!");
	}

1. 매개변수의 타입을 추론할 수 있는 경우에는 타입을 생략할 수 있습니다.
2. 매개변수의 하나인 경우에는 괄호()를 생략할 수 있습니다.
3. 함수의 몸체가 하나의 명령문만으로 이루어진 경우에는 중괄호 {}를 생략할 수 있습니다. (이때 세미콜론(;)은 붙이지 않음)

4. 함수의 몸체가 하나의 return 문으로만 이루어진 경우에는 중괄호{}를 생략할 수 없습니다.
5. return 문 대신 표현식을 사용할 수 있으며, 이때 반환 값은 표현식의 결과값이 됩니다. (이때 세미콜론(;)은 붙이지 않음)

Python

Extending Python lambda function

map(), filter(), reduce()

예)

```
result = list(map((lambda x : x + 2), [1,2,3,4,5]))
print(result)
>> [3,4,5,6,7]
```

Javascript

'화살표' [⇒]

(매개변수목록) ⇒ {함수 몸체}

예)

```
axtios.post('end-point', something)
.then( res ⇒ console.log(res.data))
```

HOW TO APPLY

- 함수형 인터페이스(Functional interface)
- 메소드 참조(Method reference)
- 생성자 참조(Constructor reference)
- 스트림 API(Stream API)

Functional interface

@FunctionalInterface

java.util.function

ㄴ IntFunction<R>

ㄴ BinaryOperator<T>

예)

```
IntFunction sum = (x) → x+1;
System.out.println(sum.apply(1));
```

java.util.function (Java Platform SE 8)

Functional interfaces provide target types for lambda expressions and method references. Each functional interface has a single abstract method, called the functional method for that functional interface, to which the lambda expression's parameter and return types are matched or adapted.

https://docs.oracle.com/javase/8/docs/api/java/util/function/package-summary.html

Method reference

람다 표현식이 단 하나의 메소드만을 호출하는 경우에 해당 람다 표현식에서 불필요한 매개변수를 제거하고 사용할 수 있음

메소드 참조를 사용하면 불필요한 매개변수를 제거하고 다음과 같이 ::(콜론2개) 기초를 사용하여 표현할 수 있음

예) 클래스 이름::메소드이름

참조변수이름::메소드이름

```
IntStream.range(0,10).forEach(System.out::println);
```

Constructor reference

생성자를 호출하는 람다 표현식도 앞서 살펴본 메소드 참조를 이용할 수 있음

즉, 단순히 객체를 생성하고 반환하는 람다 표현식은 생성자 참조로 변환할 수 있음

예)

```
(a) -> { return new Object(a); }
Object::new;
```

Stream API

1. 스트림은 외부 반복을 통해 작업하는 컬렉션과는 달리 내부 반복(internal iteration)을 통해 작업을 수행합니다.
2. 스트림은 재사용이 가능한 컬렉션과는 달리 단 한 번만 사용할 수 있습니다.
3. 스트림은 원본 데이터를 변경하지 않습니다.
4. 스트림의 연산은 필터-맵(filter-map) 기반의 API를 사용하여 지연(lazy) 연상을 통해 성능을 최적화합니다.
5. 스트림은 aprallelStream() 메소드를 통한 손쉬운 병렬 처리를 지원합니다.

```
example.stream().filter(x -> x < 2).count();
        스트림 생성      스트림 변환      스트림 사용
//Stream API를 이용한 간단한 짝수 판별
IntStream.range(1, 11).filter(i-> i%2==0)
    .forEach(System.out::println);
```

Stream API 예제

실제 코드

0부터 9가 들어있는 객체를 출력

- Java

```
List<Integer> array = Arrays.asList(0,1,2,3,4,5,6,7,8,9);
for (int i = 0; i < array.size() ; i++) {
    System.out.println(array.get(i));
}
```

일반식

```
IntStream.range(0, 10).forEach(value -> System.out.println(value));

IntStream.range(0, 10).forEach(System.out::println);
```

람다식

- Python

```
for i in range(10):
    print(i)
```

일반식

```
do = lambda x: print(x)
do([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

람다식

- Javascript

```
const array = [0,1,2,3,4,5,6,7,8,9];
for (let i = 0; i < array.length ; i++) {
    console.log(array[i]);
}
```

```
array.forEach( i => console.log(i));
```

왜 람다를 써야할까?

- 병렬처리 : 코드의 가독성을 높이면서 더 간결하게 쓸 수 있다.

  → 하지만 무분별하게 쓰면 의미를 해칠 수도 있다.

QUIZ

1. 람다표현식이 가능한 것은 객체에 이름이 없고 하나밖에 없는 추상 메소드를 가진 00 클래스를 호출하는 것을 람다식으로 표현할 수 있다.

   → 익명

2. 자바에서 람다식을 쓸 수 있는 버전은 0 버전 이상이다.

   → 8ver (1.8)

3. 다음 중 람다식에서 사용되는 화살표(Arrow token)은 어떤 언어의 람다식 화살표일까요?
   () ⇒ {}
   1) Java 2) Python 3) Javascript

   → 3

4. 람다식의 가장 큰 장점은 연산을 0000를 하고 가독성을 높이는 효과가 있다.

   → 병렬처리