liftIO 2021

Functional Programming at Work

연속된 우연으로 꾸려진 개발팀의 함수형 Scala 활용기

LINE 김대현

liftIO 2021

Functional Programming at Work

연속된 우연으로 꾸려진 개발팀의 함수형 Scala 활용기

LINE 김대현

발표 내용 agenda, actually

함수형 프로그래밍 좋은 점 advantages

함수형 프로그래밍에서 쓰는 매직템 magic items

~~함수형 Scala 개발팀이 꾸려진 경험담~~

우리의 명령형 프로그래밍 환경에서, 함수형 프로그래밍 도입하기

1 시작

함수형 프로그래밍, 호기심

함수형 프로그래밍이 좋다더라.

뭔지 알고 싶고, 내게도 좋은지 확인하고 싶다.

알아보니 뭐라는 건지 잘 모르겠다.

결국 직접 해봐야 알 것 같다.

프로그래밍 What is programming, anyway?

내가 풀기 원하는 문제를 컴퓨터에게 풀게 한다.

보통은 풀고자 하는 문제의 규모가 커서 한 번에 해결하기 힘들다.

인간의 두뇌로 한 번에 생각할 수 있는 범위는 제한적이다.

커다란 문제를 잘게 나눠 풀고,

그 해결책을 잘 조립해서, 원래의 (큰) 문제를 해결하자.

프로그래밍 방법별 조립법

절차형procedural 프로그래밍 - 코드 블럭으로 나눠서 점프, 복귀 프로시져, 서브루틴

OOP 객체지향 프로그래밍 - 내부 상태를 숨긴 객체를 상속, 위임

FP 함수형 프로그래밍 - 함수를 합성 function composition

함수형 프로그래밍 Functional Programming

거칠지만 단순히 얘기하자면,

내가 풀고자 하는 문제를 "함수를 주로 써서" 해결 하는 프로그래밍

함수로 충분한가? 함수가 그렇게 강력한가? 유연한가?

함수로 푼 해결책이 서로 잘 합쳐지는가?

함수형 프로그래밍 언어의 예

Java - Lambda, Function<T, R>

JavaScript - 고차함수, Python - lambda

imperative

Clojure - 동적 타입 함수형 언어. 부수 효과side effect를 관례로 분리!

Scala - OOP와 FP의 조화

Haskell - 순수pure 함수형 언어

functional

계승! 예제 factorial

$$n! = n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdot \cdots \cdot 3 \cdot 2 \cdot 1$$

$$n! = 1 \cdot 2 \cdot 3 \cdots (n-2) \cdot (n-1) \cdot n$$

$$n! = n \cdot (n-1)!.$$

$$n! = \prod_{i=1}^{n} i.$$

3! = 6

4! = 24

5! = 120

```scala
package purecoincidence

object HigherLevel extends App {
  def factA(n: Int): Int = {
    var r = 1
    for {
      i ← 1 to n
    } r *= i
    r
  }

  def factB(n: Int): Int =
    if (n ≤ 1) 1
    else n * factB(n - 1)

  def factC(n: Int): Int =
    (1 to n).product

  println((
    factA(5),
    factB(5),
    factC(5)
  ))  // ⇒ (120,120,120)
}
```

```java
package hopefullypure;

import static java.util.stream.IntStream.rangeClosed;
import static java.lang.System.out;
import static java.lang.String.format;

public class HigherLevel {
    static int factA(int n) {
        int r = 1;
        for (int i = 1; i ≤ n; i++) r *= i;
        return r;
    }

    static int factB(int n) {
        if (n ≤ 1) return 1;
        return n * factB(n - 1);
    }

    static int factC(int n) {
        return rangeClosed(1, n).reduce(1, (a, b) → a * b);
    }

    public static void main(String[] args) {
        out.println(
          format("%d, %d, %d",
            factA(5), factB(5), factC(5)));
    }
```

2 함수형 프로그래밍 매직템

FP 특징

기계 중심 관점과 사람 중심 관점의 차이

작은 문제를 해결하는 함수를 합성해 큰 문제를 해결하기 쉽다

(상태나 부수 효과 없이) 순수 함수의 입력과 출력만을 다룬다면,

그만큼 추론이나 테스트가 쉽다

시간과 상태를 고려하지 않아도 된다.

멀티스레드 처리가 훨씬 쉬워진다

이미 오랜 기간 탄탄히 갖춰진 수학적 모델을 빌려 쓰기 좋다. 증명된 탄탄함

고차함수 higher-order function

어떤 함수가,

다른 함수를 인수로 받거나,

호출 결과로 보통의 값이 아닌 함수를 반환하거나,

또는, 그 둘 다를 한다.

함수를 평범한 값으로 취급할 수 있어야 가능.

인수나 반환 값으로 지정 가능해야 하고, 실행 중에 즉시 함수를 만들 수 있어야.

언어마다 람다함수 등의 간결한 문법을 제공하는 경우가 흔함.

```scala
package purecoincidence

object HigherOrderFunction {
  def multiple(n: Int): Int => Int =
    x => x * n

  def pow(r: Int): Int => Int =
    x => List.fill(r)(x).product

  val triple: Int => Int = multiple(3)
  val square: Int => Int = pow(2)

  def main(args: Array[String]): Unit =
    println((
      List(1, 2, 3, 4).map(triple compose square),
      List(1, 2, 3, 4).map(square).map(triple)
    ))     // => List(3, 12, 27, 48), ...
}
```

```java
package hopefullypure;

import java.util.function.Function;
import java.util.stream.Stream;

import static java.util.stream.Collectors.toList;

public class HigherOrderFunction {
    static Function<Integer, Integer> multiple(int n) {
        return x -> x * n;
    }

    static Function<Integer, Integer> pow(int r) {
        return x -> (int)Math.pow(x, r);
    }

    static Function<Integer, Integer> triple = multiple(3);
    static Function<Integer, Integer> square = pow(2);

    public static void main(String[] args) {
        System.out.println(
          Stream.of(1, 2, 3, 4)
                .map(triple.compose(square))
                .collect(toList())
        );  // => [3, 12, 27, 48]
        // == Stream.of(1, 2, 3, 4).map(square).map(triple)...
    }
}
```

순수 함수 Pure Function

1. 같은 인수를 넘겨 호출하면, (언제나) 결과값이 같게 나온다.

2. 함수를 호출해도, 별도의 부수 효과가 발생하지 않는다.

사실상, 수학적 의미의 (그냥) 함수.

하지만, 명령형 프로그래밍 환경의 함수는 그렇지 않을 때가 많다.

```scala
package purecoincidence

object Impurity extends App {
  var current: Int = 0
  def impure(n: Int): Int = {
    current += n
    current
  }

  val i = impure(2)

  println((
    i + i,
    impure(2) + i,
    impure(2) + impure(2)
  ))  // (4,6,14)
}
```

```java
package hopefullypure;

import static java.lang.System.out;
import static java.lang.String.format;

public class Impurity {
    static int current = 0;
    static int impure(int n) {
        return current += n;
    }

    public static void main(String[] args) {
        int i = impure(2);
        out.println(
          format("%d, %d, %d",
                 i + i,
                 impure(2) + i,
                 impure(2) + impure(2)));
    }
}
```

```scala
package purecoincidence

object Purity extends App {
  def pure(a: Int, b: Int): Int =
    2*a*a + 3*b     // 2a^2 + 3b

  val p = pure(2, 3)    // ⇒ 17

  println((
    p + p,
    pure(2, 3) + p,
    pure(2, 3) + pure(2, 3),
  ))     // ⇒ (34,34,34)
}
```

```java
package hopefullypure;

import static java.lang.System.out;
import static java.lang.String.format;

public class Purity {
    static int pure(int a, int b) {
        return 2*a*a + 3*b;
    }

    public static void main(String[] args) {
        int p = pure(2, 3);
        out.println(
          format("%d, %d, %d",
                 p + p,
                 pure(2, 3) + p,
                 pure(2, 3) + pure(2, 3))); /
    }
}
```

```scala
package purecoincidence

object PurelyLong extends App {
  def pure(a: Int, b: Int): Int =
    2*a*a + 3*b     // 2a^2 + 3b

  def expressionA(a: Int, b: Int): Int =
    4*a*a + pure(a, b) + 6*b

  def expressionB(a: Int, b: Int): Int =
    3 * pure(a, b)

  println((
    expressionA(3, 4),
    expressionB(3, 4)
  ))     // ⇒ (90,90)
}
```

```java
package hopefullypure;

import static java.lang.System.out;
import static java.lang.String.format;

public class PurelyLong {
    static int pure(int a, int b) {
        return 2 * a * a + 3 * b;
    }

    static int expressionA(int a, int b) {
        return 4 * a * a + pure(a, b) + 6 * b;
    }

    static int expressionB(int a, int b) {
        return 3 * pure(a, b);
    }

    public static void main(String[] args) {
        out.println(
          format("%d, %d",
            expressionA(3, 4),
            expressionB(3, 4)));
    }
}
```

모노이드 monoid

empty, combine

- 결합법칙 (a + b) + c = a + (b + c)
- 항등원 a + 0 = a, 0 + a = a

Int + Int, String ++ String

결합 법칙이 성립하는 연산이 뭐가 좋은가?

- MapReduce

옵션 모나드 Option Monad

Monad - Sequence of computation

pure, flatMap

Option, State, IO, ...

Option 모나드 - 값 유무. 에러 유무를 다루기 편한 무언가.

scala.lang.Option[T] -> Some(x) | None

java.util.Optional<T>

```scala
package purecoincidence

object NoValueThing {
  def divideA(x: Int, y: Int): Int = x / y

  divideA(8, 2)
  divideA(6, 3)
  //  divideA(3, 0) // ⇒ ArithmeticException

  def divideB(x: Int, y: Int): Int = {
    assert(y ≠ 0)
    x / y
  }

  def divideC(x: Int, y: Int): Int = {
    if (y == 0) -1
    else x / y
  }

  def divideD(x: Int, y: Int): (Int, Boolean) =
    if (y == 0) (0, false)
    else (x / y, true)

  def divide(x: Int, y: Int): Option[Int] =
    if (y == 0) None
    else Some(x / y)
}
```

```java
package hopefullypure;

import java.util.Optional;

public class NoValueThing {
    static int divideA(int x, int y) {
        return x / y;
    }

    static int divideB(int x, int y) {
        assert y ≠ 0;
        return x / y;
    }

    static int divideC(int x, int y) {
        if (y == 0) return -1;
        return x / y;
    }

    static Optional<Integer> divide(int x, int y) {
        if (y == 0) return Optional.empty();
        else return Optional.of(x / y);
    }
}
```

```scala
package purecoincidence

object PureExistence extends App {
  def divide(x: Int, y: Int): Option[Int] =
    if (y == 0) None
    else Some(x / y)

  def parseInt(s: String): Option[Int] = {
    try {
      Some(s.toInt)
    } catch {
      case _: NumberFormatException => None
    }   // == scala.util.Try(s.toInt).toOption
  }

  def stringDivide0(sx: String, sy: String): Option[Int] =
    parseInt(sx).flatMap { x =>
      parseInt(sy).flatMap { y =>
        divide(x, y)
      }
    }

  def stringDivide(sx: String, sy: String): Option[Int] =
    for {
      x <- parseInt(sx)
      y <- parseInt(sy)
      d <- divide(x, y)
    } yield d
}
```

```java
package hopefullypure;

import java.util.Optional;

public class PureExistence {
    static Optional<Integer> divide(int x, int y) {
        if (y == 0) return Optional.empty();
        else return Optional.of(x / y);
    }

    static Optional<Integer> parseInt(String s) {
        try {
            return Optional.of(Integer.parseInt(s));
        } catch (NumberFormatException e) {
            return Optional.empty();
        }
    }

    static Optional<Integer> stringDivide(String sx, String sy) {
        return parseInt(sx).flatMap(x ->
                parseInt(sy).flatMap(y ->
                  divide(x, y)));
    }
}
```

3 우연의 연속

FP 실무에 쓰려 한다면...어렵죠

상사, 동료를 설득해야 할 터.

이해관계자 다수가 FP에 (동시에) 관심 있기는 어렵지 않겠나?

이미 잘 갖춰진 기존 프로젝트에 큰 변화를 가져오기는 어렵고,

큰 규모의 회사에서 신규 프로젝트를 맡는 것은 쉽지 않은 일.

예외적으로 신규 프로젝트를 맡는다고 해도,

새로운 언어를 배우고, 새 환경의 프레임워크를 익히느라 일정 압박 👆

팀원 구성, 채용의 문제

우연의 합성

이직한 회사의 본부 저장소를 보니, 이미 Scala를 꽤 쓰고 있었다.

신규 프로젝트를 "혼자" 맡게 되었다.

충원 1. 오래전 동료가 Scala로 개발하고 있었고, 우리팀으로 이직했다.

충원 2. 계열사 전직 지원자가 함수형 프로그래밍에 관심

충원 3. 또 다른 오랜 동료가, Clojure를 좋아하고, Scala도 해보겠다고.

4 Scala FP 라이브러리

웹서비스가 처리할 일들

HTTP API 서비스

JSON 인코딩/디코딩

RDB 등의 데이터베이스에 쓰고 읽기

Kafka 등의 브로커에 메시지 전달 (at-least-once)

이런 일을 함수형 프로그래밍으로 잘할 수 있으면 되는 거 아닌가?

Cats

Search Enter keywords here... GitHub API Documentation

Lightweight, modular, and extensible library for functional programming

View on GitHub

Home Type Classes Data Types Motivations Resources for Learners FAQ Contributing Colophon Glossary Guidelines Typelevel Ecosystem

Community Announcements

- Oct 31 2020 Cats 2.3.0-M1 is released with Dotty supported
- Sept 3 2020 Cats 2.2.0 is released
- Dec 18 2019 Cats 2.1.0 is released
- Sep 9 2019 Cats 2.0.0 is released

http4s

Typeful, functional, streaming HTTP for Scala

Getting Started

Latest Stable Release

0.23.6

Latest Milestone Release

1.0.0-M29

License

Apache 2.0

Project Info

- Versions
- Changelog
- Getting Help
- Contributing
- Adopters
- Code of Conduct
- Further Reading

circe

Search Enter keywords here...

GitHub

API Documentation

A JSON library for Scala powered by Cats

View on GitHub

Parsing JSON Traversing and modifying JSON Encoding and decoding Optics Performance

build unknown | coverage 86% | gitter join chat | maven-central v0.15.0-M1

doobie

build unknown | gitter join chat | maven-central v1.0.0-RC1 | javadoc 1.0.0-RC1

doobie is a pure functional JDBC layer for Scala and **Cats**. It is not an ORM, nor is it a relational algebra; it simply provides a functional way to construct programs (and higher-level libraries) that use JDBC. For common use cases **doobie** provides a minimal but expressive high-level API:

```scala
import doobie._
import doobie.implicits._
import cats.effect.IO
import scala.concurrent.ExecutionContext

import cats.effect.unsafe.implicits.global

val xa = Transactor.fromDriverManager[IO](
  "org.postgresql.Driver", "jdbc:postgresql:world", "postgres", ""
)

case class Country(code: String, name: String, population: Long)

def find(n: String): ConnectionIO[Option[Country]] =
  sql"select code, name, population from country where name = $n".query[Country].option
```

And then ...

5 당신의 업무에 FP 도입

명령형 언어에서 함수형 프로그래밍 가능

함수도 값으로 다루기. 고차함수 활용. 함수 조합

vavr.io - A functional library for Java

효과effect를 일으키는 프로시저와 순수함수pure function를 분리

함수형 언어를 직업적으로 쓰기 위한 조건

지속적인 관심과 바램

내게 (그닥) 중요치 않은 것들을 포기할 용기

충분한 자신감

- 개인 프로젝트로 셀프 훈련 👟
- 실무 프로젝트의 단위 테스트 코드에 점진적 반영해서 경험치 👆

협상의 옳바른 출발점

저희 프로젝트에 함수형 프로그래밍을 도입하면 어떨까요? ❌

새로운 시도의 가치를 증명해야 하고, 힘든 변화를 리드해야 한다.

(1) 그냥 (묵묵히) 저지른다. (2) 또는, 적용하고 나서 (친절히) 통보 ⭕

지르고 나서, (거꾸로) 왜 안되는 지를 (예의 바르게) 물어보자.

범죄 행위나 부도덕한 일이 아니지 않은가?

당당하지 못 할 이유가 없다.

6 채용 홍보

ADS Platform Server Engineer

Bundang | LINE Plus | Engineering | Server-side | Full-time

일본, 태국, 대만 등의 글로벌 국가에 LINE 광고를 제공하는 광고플랫폼 Server-side를 개발합니다.

담당업무

- 광고 시스템의 Software Engineer
- Campaign Manager RESTful API, JPA ORM, Big Data Report System 개발
- 고성능 AD Server, CTR Prediction, Ranking Logic 등의 개발
- A/B 테스트를 통한 로직 검증 및 반영

자격요건

- Java, Kotlin, Scala 언어 혹은 C/C++, Golang 중 하나 이상의 언어에 숙련되신 분
- RESTful API 디자인에 능숙하신 분
- Pair 프로그래밍, 코드리뷰 등에 거부감이 없고 테스트 코드 작성의 필요성을 알고 있으신 분
- 해외 출장 및 근무에 결격 사유 없으신 분

(뒤늦은) 발표자 소개

케케묵은 백엔드 개발자

(現) LINE+ > ADS Engineering > MONAD LEAD

NHN > Dooray 개발실 수석

Daum > 클라우드컴퓨팅 팀장

함수형 프로그래밍에 관심 갖고 Clojure, Scala, Haskell 학습 중

https://medium.com/@hatemogi

LINE 회사 / 본부 / 팀 모두 채용 중 (신입가능)

Questions