Функциональное программирование в реальности: а был ли мальчик-то?

Марат Ахин

15 ноября 2017 г.

Санкт-Петербургский политехнический университет

Функциональное программирование?

Что это такое?

- Функциональная программа является вычисляемым выражением, которое отвечает на интересующий нас вопрос
  - Сам процесс вычисления нас особо не интересует
  - Вычисления являются повторяемыми

Функциональное программирование?

- Функциональная программа отвечает на вопрос “что?”
- Императивная программа отвечает на вопрос “как?”

Черное и белое

Функциональное программирование в реальности

Посередине между черным и белым

- Кто-то попытался добавить ФП в свой императивный мир
  - Java
  - C++
  - C#
- Кто-то попытался добавить ИП в свой функциональный мир
  - OCaml
  - Clojure
- Кто-то попытался объединить две стороны
  - Scala
  - Kotlin
- Кто-то просто странный
  - JavaScript

Функции в Java [8]

Все — это функция?

```java
@FunctionalInterface
public interface Function<T, R> {
    R apply(T t);

    default <V> Function<V, R> compose(Function<? super V, ? extends T> before) {
        Objects.requireNonNull(before);
        return (V v) -> apply(before.apply(v));
    }

    default <V> Function<T, V> andThen(Function<? super R, ? extends V> after) {
        Objects.requireNonNull(after);
        return (T t) -> after.apply(apply(t));
    }

    static <T> Function<T, T> identity() {
        return t -> t;
    }
}
```

Функции в Java [8]

Подождите, а если мне нужно два аргумента?

```java
@FunctionalInterface
public interface BiFunction<T, U, R> {
    R apply(T t, U u);

    default <V> BiFunction<T, U, V> andThen(
            Function<? super R, ? extends V> after) {
        Objects.requireNonNull(after);
        return (T t, U u) -> after.apply(apply(t, u));
    }
}
```

Функции в Java [8]

А если три?..

Функции в Java [8]

...на самом деле, не совсем

```
@FunctionalInterface
public interface MySuperAwesomeTriFunction<A, B, C, R> {
    R apply(A a, B b, C c);

    default <V> MySuperAwesomeTriFunction<A, B, C, V> andThen(
            Function<? super R, ? extends V> after) {
        Objects.requireNonNull(after);
        return (A a, B b, C c) -> after.apply(apply(a, b, c));
    }
}
```

Лямбды в Java [8]

А как все это использовать?

```
final Function<Integer, String> int2string = new Function<Integer, String>() {
    public String apply(Integer value) {
        return value.toString();
    }
};
```

Лямбды в Java [8]

А можно проще?

```
final Function<Integer, String> int2string =
    value -> value.toString();
```

Лямбды в Java [8]

А можно еще проще?

```
final Function<Integer, String> int2string =
    Object::toString; // method reference
```

Связанные переменные в Java [8]

А если я хочу взять что-то из своего окружения?

```
int radix = 16;

final Function<Integer, String> int2hexString =
    value -> Integer.toString(value, radix);

radix = 8;

final Function<Integer, String> int2octString =
    value -> Integer.toString(value, radix); // NOPE!
```

Связанные переменные в Java [8]

А если я хочу взять что-то из своего окружения?

```
final int radix = 16; // [effectively] final variable

final Function<Integer, String> int2hexString =
    value -> Integer.toString(value, radix);
```

Каррирование в Java [8]

- Простой ответ: его нет

Каррирование в Java [8]

- Простой ответ: его нет

- Сложный ответ: его можно собрать самому

```
final Function<Integer, Function<Integer, String>> // oh-MI-god!
    int2hexString = value -> radix -> Integer.toString(value, radix);
```

Монады в Java [8]

А может быть, не надо?..

Монады в Java [8]

А может быть, не надо?..

```
class Monad m where
  (>>=)  :: m a -> (a -> m b) -> m b
  (>>)   :: m a ->       m b  -> m b
  return ::   a               -> m a
  fail   :: String -> m a
```

Монады в Java [8]

“Надо, Федя, надо...” (с)

```java
public final class Optional<T> {
    public<U> Optional<U> flatMap(Function<? super T, Optional<U>> mapper) {
        Objects.requireNonNull(mapper);
        if (!isPresent()) {
            return empty();
        } else {
            return Objects.requireNonNull(mapper.apply(value));
        }
    }

    public static <T> Optional<T> of(T value) {
        return new Optional<>(value);
    }

    // ...
}
```

Монады в Java [8]

А зачем надо-то?

```java
final Denizen denizen = db.readById(someUniqueDenizenId);
if (denizen != null) {
    final DenizenProfile profile = denizen.getProfile();
    if (profile != null) {
        final String email = profile.getEmail();
        if (email != null) {
            notifications.sendTo(email, someUsefulNotification);
        }
    }
}
```

Монады в Java [8]

А зачем надо-то?

```
final Optional<Denizen> denizen =
    Optional.of(db.readById(someUniqueDenizenId));
if (denizen.isPresent()) {
    final Optional<DenizenProfile> profile =
        denizen.map(d -> d.getProfile());
    if (profile.isPresent()) {
        final Optional<String> email =
            profile.map(p -> p.getEmail());
        if (email.isPresent()) {
            notifications.sendTo(email.get(), someUsefulNotification);
        }
    }
}
```

Монады в Java [8]

А зачем надо-то?

```
final Optional<Denizen> denizen =
    Optional.of(db.readById(someUniqueDenizenId));
if (denizen.isPresent()) {
    final Optional<DenizenProfile> profile =
        denizen.map(d -> d.getProfile());
    if (profile.isPresent()) {
        final Optional<String> email =
            profile.map(p -> p.getEmail());
        if (email.isPresent()) {
            notifications.sendTo(email.get(), someUsefulNotification);
        }
    }
}
```

Мы что-то забыли...

Монады в Java [8]

```
Optional.of(db.readById(someUniqueDenizenId))
    .map(d -> d.getProfile())
    .map(p -> p.getEmail())
    .ifPresent(e ->
        notifications.sendTo(e, someUsefulNotification)
    );
```

Потоки в Java [8]

Что, тоже монада?

```
listOfIds.stream()
    .map(db::readById)
    .map(Denizen::getEmail)
    .filter(Optional::isPresent)
    .map(Optional::get)
    .forEach(e ->
        notifications.sendTo(e, someUsefulNotification)
    );
```

А чем это принципиально отличается от работы со списками?

Потоки в Java [8]

Поток может быть бесконечным

```
final Stream<Integer> fibStream =
    Stream.iterate(Pair.of(0, 1), p -> Pair.of(p.second, p.first + p.second))
        .map(Pair::getSecond);

final List<Integer> firstTenFibs =
    fibStream.limit(10)
        .collect(Collectors.toList()); // терминирующая операция

final List<Integer> nextTenFibs =
    fibStream.skip(10)
        .limit(10)
        .collect(Collectors.toList()); // NOPE!
```

Потоки в Java [8]

Поток является ленивым...

(иначе он не мог бы быть бесконечным)

...и считается на терминирующей операции

```
public interface Collector<T, A, R> {
    Supplier<A> supplier();

    BiConsumer<A, T> accumulator(); // damn, son...

    BinaryOperator<A> combiner();

    Function<A, R> finisher();

    // ...
}
```

Потоки в Java [8]

Поток может обрабатываться параллельно

```
list.parallelStream()
    .map(...)
    .filter(...)
    .collect(...)
```

Потоки в Java [8]

Поток может обрабатываться параллельно

```
list.parallelStream()
    .map(...)
    .filter(...)
    .collect(...)
```

...на самом деле нет

И что получается?

Жалкое подобие нормального функционального языка

- Функциональный код проще композировать
- Функциональный код проще понимать (ну или как-то так...)

- Функциональный код плохо комбинируется с императивным
- Функциональный код не всегда работает так, как ожидается

А что в других языках?

- OCaml
  - Изменяемые поля
  - Ссылки
  - Циклы (sic!)
  - Объекты и классы
- Kotlin
  - Понемногу всего отовсюду =)
  - Хотите узнать больше — поступайте к нам на 1 курс =)

What I learned today