Kotlinの特徴として語られる機能の効果を実践での結果から紹介する from サーバーサイド

2019年7月2日 Kotlin愛好会 vol.12

竹端 尚人

自己紹介

概要

氏名:竹端 尚人
Twitter:@n_takehata
株式会社アプリボット所属

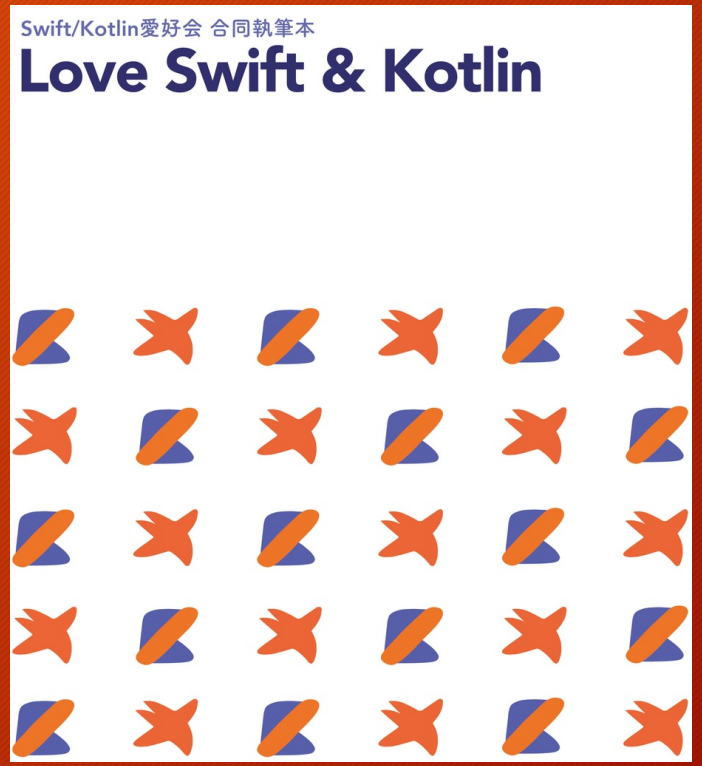
・サーバーサイドKotlin
・スマートフォンゲーム開発
・エンジニアは11年ほど

登壇、執筆など

・CEDEC 2018登壇

・Software Design 2019年2～4月号で短期連載

・Swift/Kotlin愛好会で技術書典執筆
https://booth.pm/ja/items/1315478



宣伝

- CEDEC 2019登壇

- **Unity C# × gRPC × サーバーサイドKotlinによる次世代のサーバー/クライアント通信 ～ハイパフォーマンスな通信基盤の開発とMagicOnionによるリアルタイム通信の実現～**

https://cedec.cesa.or.jp/2019/session/detail/s5c9dede391631

今日話すこと

Kotlinの特徴としてよく言われるものたち

- 安全
  - Null安全
  - val、var
- コードがシンプル
  - 型推論
  - データクラス
  - スコープ関数
- Javaからの移行コストが低い
  - Javaとの相互互換
  - Spring FrameworkのKotlin対応
- etc...

実践で使うとどう良いのか？

1年半開発した結果にならって話します
(サーバーサイド視点)

アジェンダ

1. Null安全
2. 変数は基本的にImmutableにできる
3. 不要なキーワードの削除
4. Lombokが不要になった
5. まとめ

安全性

コンパイルで防げるエラーが多く
安全性が高まった

1.Null安全Null可、Null不可の不整合がなくなる

Javaの場合

```
public void execute(Integer id) {
    createUser(id);
}

private void createUser(@NotNull Integer id) {
    // ...
}
```

②

①

①引数をNull不可にしている(Lombokを使用)
②Null可の変数を引数に渡して実行している

実行時にNullPointerExceptionが発生する

Kotlinの場合

```
fun execute(id: Int?) {
    createUser(id)
}

private fun createUser(id: Int) {
    // ...
}
```

②

①

①引数は?なしなのでNull不可
②Null可の変数を引数に渡して実行している

コンパイルエラーで気付ける

実際に使っていて

- 大体の変数、引数はNull不可で問題ない
- Java(or その他の言語)では意識せずNull可にしていた変数も、Null不可で良い場合が多い
- デフォルトがNull不可になっていることで意識が変わる

2. 変数定義は基本的にImmutableにできる

val、varの存在

- KotlinはVal、varで必ず変数のImmutable、Mutableを定義しないといけない
- JavaではデフォルトがMutableで、Immutableにするにはfinalを付ける必要があった

更新は基本的に
オブジェクトのプロパティにすることが多い

- IntやStringの変数を直接更新することは少ない
- O/RマッパーのEntityクラスなどのオブジェクトを取得して更新することが多い

オブジェクトの取得、更新

```
data class User(val id: Int, var name: String)
```

```
val user = selectUser(id)   ①
user.name = "hoge"          ②
updateUser(user)
```

①Userオブジェクトを取得
②nameプロパティを更新

valで変数定義しても、プロパティの値は書き換えられる
(プロパティがvarで定義されていれば)

IntやStringの変数も書き換えないことが多い

```
val product: Product = getProduct(id)
val userCount: Int = getUserCount(id)
product.userCount = userCount
```

①

②

①Int型の変数に値を取得
②オブジェクトのプロパティに設定

関数で取得した値を入れて、そのまま使う場合が多い

varを使うパターン

インクリメントしていきたい数値など

```
fun calcAmmount(itemList: List<Item>) {
    var sumAmmount = 0 ——— ①
    itemList.forEach {
        if (sumAmmount + it.price > MAX_BUY_AMMOUNT) {
            return
        }
        sumAmmount += it.price ——— ②
    }
}
```

①varで合計値の変数を定義
②ループの中で変数に加算していく

実際に使っていて

- 大体の変数はvalで定義できる(Immutableにできる)
- var を使うのは最低限の箇所に留め、基本は val とし変更不可にするのが安全
- val、varの存在やCollectionのデフォルトがImmutableなことでImmutableで事足りる気付く

Collectionも基本的にImmutable

実際に使っていて

- やっぱりListを取得して使うだけで、変更を加えない場合が多い
- Collectionライブラリが優秀なので、変更を加える場合はこちらを使った方が分かりやすい

  →MutableListを使う機会は少ない

コードがシンプル

コード量(ステップ数)は2割減った

3.不要なキーワードの削除

Javaの場合

```
public void execute() {
    User user = new User();
    user.setId(1);
    user.setName("Applibot");
    registerUser(user);
    System.out.println("id=" + user.getId() + "name=" + user.getName());
}
```

Kotlinの場合

```
fun execute() {
    val user = User(1, "Applibot")
    registerUser(user)
    println("id=${user.id} name=${user.name}")
}
```

- デフォルトのアクセス修飾子がpublic
- newキーワードが不要
- 型推論
- String Template
- セミコロンが不要
- etc...

実際に使っていて

- 記述量が減ることで、考えることも減り思考がスムーズになった
- いくらIDEが補完してくれても、記述が必要だと意識はしないといけない
- セミコロンやnewなど、細かいところもバカにならない

4.Lombokが不要になった

@Getter、@Setter

Javaの場合

```
public class User {
    @Getter
    private Integer id;

    @Getter
    @Setter
    private String name;
}
```

①

②

①Getterのみ生成
②Getter、Setter両方を生成

Kotlinの場合

```
class User {
    val id: Int = 0
    var name: String = ""
}
```

①

②

①Getterのみ生成
②Getter、Setter両方を生成

@Data

Javaの場合

```
@Data
public class User {
    private Integer id;

    private String name;
}
```

Kotlinの場合

```
data class User(val id: Int, val name: String, var age: Int)
```

データクラスを使う

@Builder

Javaの場合

```
@Builder
@Data
public class User {
    private Integer id;
    private String name;
    private String profile;
    private Integer age;
}
```

```
return User().builder()
        .id(1)
        .name(”LoveKotlin")
        .age(30)
        .build();
```

Kotlinの場合

```
data class User(val id: Int, val name: String, val profile: String = "", val age: Int)
```

```
return User(id = 1, name = "Applibot", age = 30)
```

デフォルト引数、名前付き引数を使う

Kotlinの場合(別解)

```
val user = User().also {
    it.id = 1
    it.name = "Applibot"
    it.age = 30
}
```

- コンストラクタがない場合(データクラスでない場合等)に使える
- Javaで作られたクラスに対して使う場合にも有効(ORM関連の自動生成クラス等)

実際に使っていて

- 記述量が減ることで、考えることも減った
- Lombokが不要になり、標準機能で大体のことが実現できるのが大きい

5.まとめ

- 変数は大体Null不可にできる
- 変数は大体Immutableにできる
- 記述量が少ないとプログラミングがスムーズに
- 標準機能でできることが増えた
- etc...

全文はこちらご御覧ください

1年半開発してきて実感したサーバーサイドKotlinのメリット

https://blog.applibot.co.jp/2019/06/20/benefits-of-server-side-kotlin/

Kotlinを使っていて良かった！

ご清聴ありがとうございました