Họ tên: Lê Phúc Hưng

MSSV: 20215276

Mã lớp: 151902

Môn học: Phát triển ứng dụng cho thiết bị di động (IT4785)

Source Code: https://github.com/lephuchung/HustMobile

Bài: Lesson 13 – Coroutines

Contents

1. Giới thiệu

Báo cáo này tổng hợp 32 ví dụ minh họa cho việc sử dụng Kotlin Coroutines trong các tình huống khác nhau. Các ví dụ bao gồm từ cách tạo coroutine đầu tiên, quản lý context, xử lý lỗi, cho đến việc áp dụng các mô hình phức tạp như SupervisorJob và Flow. Tất cả đều nhằm mục đích giúp lập trình viên hiểu sâu hơn về lập trình bất đồng bộ và các khía cạnh quản lý concurrency trong Kotlin.

2. Phân tích các ví dụ

Bài 1: Coroutine đầu tiên

Ví dụ 1: Sử dụng GlobalScope.launch để tạo coroutine đầu tiên.

```
GlobalScope.launch {
    delay(1000)
    println("Hello")
}
println("World")
Thread.sleep(2000)
```

Đầu ra:

World

- Hello

Giải thích: Coroutine chạy trong GlobalScope không block thread chính, do đó "World" được in trước khi coroutine hoàn thành.

Ví dụ 2: Sử dụng runBlocking để block thread chính trong khi coroutine thực thi.

```
runBlocking {
    delay(1000)
    println("Hello")
}
println("After runBlocking")
```

Đầu ra:

Hello

- After runBlocking

Giải thích: runBlocking block thread chính cho đến khi toàn bộ coroutine bên trong hoàn thành.

Bài 2: Coroutine Context

Ví dụ 3: Dispatchers xác định thread mà coroutine chạy trên đó (Default, IO, Main, Unconfined).

Đầu ra:

Dispatchers Default run on DefaultDispatcher-worker-1

Dispatchers IO run on DefaultDispatcher-worker-2

Dispatchers Unconfined run on main

- Dispatchers Main run on main

Giải thích: Dispatcher quyết định thread thực thi, giúp tối ưu hóa hiệu năng.

Ví dụ 4: Dispatchers.Unconfined có thể chuyển sang thread mới nếu coroutine chạy quá lâu.

Đầu ra:

Before delay - Dispatchers Unconfined run on main

- Dispatchers Unconfined run on DefaultDispatcher-worker-1

Giải thích: Unconfined không gắn kết với thread cụ thể, phù hợp cho coroutine ngắn.

Ví dụ 5: Sử dụng withContext để thay đổi dispatcher trong một coroutine.

```
GlobalScope.launch(Dispatchers.IO) {
   withContext(Dispatchers.Main) {
      println("Update UI")
   }
```

- }

Đầu ra:

Update UI

Giải thích: withContext cho phép chuyển đổi dispatcher trong khi coroutine đang chạy.

- Ví dụ 6: Job để quản lý các coroutine. job1.join() đảm bảo coroutine được thực thi tuần tự.

Đầu ra:

Hello Kotlin

Giải thích: join chờ một job hoàn thành trước khi tiếp tục.

Ví dụ 7: Hủy coroutine với job.cancel().

Đầu ra:

I'm sleeping 0 ...

I'm sleeping 1 ...

- Cancelled coroutines

Giải thích: cancel dừng coroutine trước khi nó hoàn thành.

Ví dụ 8: cancelAndJoin đảm bảo coroutine bị hủy hoàn toàn trước khi tiếp tục.

Đầu ra:

job: I'm sleeping 0 ...

job: I'm sleeping 1 ...

main: I'm tired of waiting

- main: Now I can quit

Giải thích: cancelAndJoin dừng và chờ coroutine kết thúc hủy bỏ.

Ví dụ 9: Biến isActive được sử dụng để kiểm tra trạng thái coroutine.

Đầu ra:

job: I'm sleeping 0 ...

job: I'm sleeping 1 ...

main: I'm tired of waiting

- main: Now I can quit

Giải thích: isActive cung cấp cách kiểm tra trạng thái bên trong coroutine.

Bài 3: Async và Await

Ví dụ 16-17: Sử dụng async và await để chạy song song nhiều coroutine và tối ưu hóa thời gian.

val a = async { doSomething1() }

val b = async { doSomething2() }

- println(a.await() + b.await())

Đầu ra:

30

Giải thích: async cho phép chạy song song và await chờ kết quả.

Bài 4: CoroutineScope

Ví dụ 18: Coroutine con bị hủy khi coroutine cha bị hủy.

Đầu ra:

coroutine 1: Hello

- coroutine 2: Hello

Giải thích: Coroutine cha kiểm soát vòng đời của các coroutine con.

Ví dụ 19: Sử dụng GlobalScope để đảm bảo coroutine chạy độc lập với cha.

Đầu ra:

coroutine 3: Hello

- coroutine 3: Goodbye

Giải thích: GlobalScope không bị ràng buộc bởi phạm vi cha.

Bài 5: Exception Handling và Supervision

Ví dụ 22: Xử lý lỗi với try-catch.

Đầu ra:

Throw Exception from Launch

- java.lang.NullPointerException

Giải thích: try-catch cung cấp cách bắt lỗi trong coroutine.

- Ví dụ 25: Sử dụng CoroutineExceptionHandler để bắt lỗi trong coroutine tạo bằng launch.

Đầu ra:

Error here: java.lang.NullPointerException

Giải thích: Handler giúp quản lý lỗi ở cấp độ global.

Ví dụ 29-30: SupervisorJob và supervisorScope cho phép các coroutine con không bị ảnh hưởng bởi lỗi của nhau.

Đầu ra:

Print from First Child

print from second Child. First Child is Active: false

- Second Child Cancelled

Giải thích: Supervisor tách biệt quản lý lỗi giữa các coroutine.

Bài 6: Sequence trong Kotlin

Ví dụ 31: Sử dụng sequence để tạo một luồng dữ liệu lười biếng (lazy).

```
fun foo(n: Int): Sequence<Int> = sequence {
    for (i in 0..n) {
        if (i % 2 == 0) yield(i)
    }
}
foo(10).forEach { println(it) }
```

Đầu ra:

0

2

4

6

8

- 10

Giải thích: sequence tạo ra luồng dữ liệu chỉ khi cần thiết.

Bài 7: Flow trong Kotlin Coroutines

Ví dụ 32: Flow cho phép xử lý dữ liệu bất đồng bộ và chỉ phát dữ liệu khi cần.

```
fun foo(n: Int): Flow<Int> = flow {
   for (i in 0..n) {
      delay(1000)
      emit(i)
   }
}
```

Đầu ra:

i = 0

i = 1

i = 2

- i = 3

Giải thích: Flow hoạt động như một luồng dữ liệu bất đồng bộ, không chiếm dụng thread không cần thiết.Giải thích: Flow hoạt động như một luồng dữ liệu bất đồng bộ, không chiếm dụng thread không cần thiết.Giải thích: Flow hoạt động như một luồng dữ liệu bất đồng bộ, không chiếm dụng thread không cần thiết.Giải thích: Flow hoạt động như một luồng dữ liệu bất đồng bộ, không chiếm dụng thread không cần thiết.Giải thích: Flow hoạt động như một luồng dữ liệu bất đồng bộ, không chiếm dụng thread không cần thiết.Giải thích: Flow hoạt động như một luồng dữ liệu bất đồng bộ, không chiếm dụng thread không cần thiết.Giải thích: Flow hoạt động như một luồng dữ liệu bất đồng bộ, không chiếm dụng thread không cần thiết.Giải thích: Flow hoạt động như một luồng dữ liệu bất đồng bộ, không chiếm dụng thread không cần thiết.Giải thích: Flow hoạt động như một luồng dữ liệu bất đồng bộ, không chiếm dụng thread không cần thiết.Giải thích: Flow hoạt động như một luồng dữ liệu bất đồng bộ, không chiếm dụng thread không cần thiết.

3. Nhận xét và tổng kết

Kotlin Coroutines mang lại cách tiếp cận mạnh mẽ và linh hoạt để xử lý concurrency và bất đồng bộ. Từ các ví dụ, ta thấy rõ sự ưu việt trong việc quản lý luồng, xử lý lỗi và tối ưu hóa hiệu năng. Những kỹ thuật như Dispatchers, async-await, và Flow giúp lập trình viên xây dựng ứng dụng mạnh mẽ và dễ bảo trì hơn.

Tổng kết: Báo cáo này không chỉ mang tính chất tham khảo mà còn cung cấp các cách áp dụng thực tiễn cho từng tính năng của Kotlin Coroutines.