Kotlin

Coroutines Labs

Nội dung

- Bài 1. Xây dựng Coroutine đầu tiên
- Bài 2. Coroutine Context
  - 2.1. Dispachers
  - 2.2. withContext
  - 2.3. Job
  - 2.4. Time outs
- 3. Async và Await
- 4. CoroutineScope
- 5. Xử lý Exception và Supervision trong Coroutine
- 6. Sequence trong Kotlin
- 7. Giới thiệu về Flow trong Kotlin Coroutines

Bước 1

- Tạo Project Kotlin coroutine example
- Thêm các dependencies vào app/build.gradle.kt

```
implementation("org.jetbrains.kotlinx:kotlinx-coroutines-android:1.3.9")
```

Bước 2.

- Tạo package firstcoroutines, tạo file BuildFirstCoroutines.kt trong package này.
- Tạo hàm main.
- Ấn nút mũi tên xanh và chạy Run (hoặc ấn Ctrl+Shift+F10).

```
fun main() {

}
```

Chương trình coroutine đầu tiên

```
package vn.edu.hust.soict.gv.quangnh.coroutineexample.firstcoroutines

import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch

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

Kết quả

```
fun main() {
    GlobalScope.launch { this: CoroutineScope
        delay( timeMillis: 1000)
        println("Hello ")
    }
    println("World ")
    Thread.sleep( millis: 2000)
}
```

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
World
Hello

Process finished with exit code 0
```

Ví dụ 2

- Tạo coroutine dùng runBlocking: tạo ra coroutine và block thread hiện tại

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

a. Xác định kết quả của chương trình này
b. Hãy hiển thị tên tiến trình của từng đoạn code trong chương trình

```
println("Current Thread: ${Thread.currentThread().name}")
```

Kết quả

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help — Coroutine Example - BuildFirstCoroutine.kt [Coroutine_Example.app.main]

coroutineexample 〉 firstcoroutines 〉 BuildFirstCoroutine.kt 〉 main — BuildFirstCoroutineKt — Nexus S API 34

Project: .gradle, .idea, app, build, libs, src, androidTest, main, java, vn.edu.hust.soict.gv.quangnh, firstcoroutines, BuildFirstCoroutine.kt, MainActivity

activity_main.xml | MainActivity.kt | build.gradle.kts (:app) | BuildFirstCoroutine.kt

```kotlin
} */

fun main() {
    runBlocking { this: CoroutineScope
        delay( timeMillis: 1000)
        println("Hello ")
        delay( timeMillis: 1000)
        println("World ")
    }
    println("After runBlocking")
}
```

Run: app | BuildFirstCoroutineKt

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
Hello
World
After runBlocking

Process finished with exit code 0
```

Version Control | Run | TODO | Problems | Terminal | Services | App Quality Insights | App Inspection | Logcat | Build | Profiler | Layout Inspector

Gradle build finished in 3 s 643 ms (moments ago) — 21:14 CRLF UTF-8 4 spaces

Bài 2. Coroutine Context

2.1. Dispatchers

- Dispatchers: quyết định Thread mà Coroutine chạy
  - Dispatchers.Default
  - Dispatchers.IO: đọc Database, Networking
  - Dispatchers.Main: update UI
  - Dispatchers.Unconfined

Dispatcher. Ví dụ 3

- Tạo package coroutinecontext
- Tạo file TestDispatchers:

```
package
vn.edu.hust.soict.gv.quangnh.coroutineexample.coroutinecontext

import android.util.Log
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.launch
import
vn.edu.hust.soict.gv.quangnh.coroutineexample.MainActivity
```

```
object TestDispatchers {
    fun runMyFirstCoroutines() {
        GlobalScope.launch(Dispatchers.Default) {
            Log.d(MainActivity::class.java.simpleName, "Dispatchers Default run
on ${Thread.currentThread().name}")
        }
        GlobalScope.launch(Dispatchers.IO) {
            Log.d(MainActivity::class.java.simpleName, "Dispatchers IO run on
${Thread.currentThread().name}")
        }
        GlobalScope.launch(Dispatchers.Unconfined) {
            Log.d(MainActivity::class.java.simpleName, "Dispatchers Unconfined
run on ${Thread.currentThread().name}")
        }
        GlobalScope.launch(Dispatchers.Main) {
            Log.d(MainActivity::class.java.simpleName, "Dispatchers Main run on
${Thread.currentThread().name}")
        }
    }
}
```

File MainActivity.kt

```
package vn.edu.hust.soict.gv.quangnh.coroutineexample

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import
vn.edu.hust.soict.gv.quangnh.coroutineexample.coroutinecontext.TestDispatchers

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        TestDispatchers.runMyFirstCoroutines()
    }
}
```

Kết quả khi chạy app

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help — Coroutine Example - MainActivity.kt [Coroutine_Example.app.main]

vn 〉 edu 〉 hust 〉 soict 〉 gv 〉 quangnh 〉 coroutineexample 〉 MainActivity — app — Nexus S API 34

Project

- src
  - androidTest
  - main
    - java
      - vn.edu.hust.soict.gv.quangnh
        - coroutinecontext
          - TestDispachers
        - firstcoroutines

TestDispachers.kt | MainActivity.kt

```
7  class MainActivity : AppCompatActivity() {
8      override fun onCreate(savedInstanceState: Bundle?) {
9          super.onCreate(savedInstanceState)
10         setContentView(R.layout.activity_main)
11         TestDispachers.runMyFirstCoroutines()
12     }
13 }
```

Logcat: Logcat

Nexus S API 34 (emulator-5554) Android 14, API 34 — MainActivity

```
I  Start proc 5608:vn.edu.hust.soict.gv.quangnh.coroutineexample/u0a199 for next-top-activity {vn.edu.hust.soict.gv.qu
V  Sent Transition #7 createdAt=12-02 21:37:36.516 via request=TransitionRequestInfo { type = OPEN, triggerTask = Task
D  Dispachers Default run on DefaultDispatcher-worker-1
D  Dispachers Unconfined run on main
D  Dispachers IO run on DefaultDispatcher-worker-2
D  Window{e9acfa3 u0 vn.edu.hust.soict.gv.quangnh.coroutineexample/vn.edu.hust.soict.gv.quangnh.coroutineexample.MainA
D  Dispachers Main run on main
I  Displayed vn.edu.hust.soict.gv.quangnh.coroutineexample/.MainActivity for user 0: +2s624ms
```

Version Control | Run | TODO | Problems | Terminal | Services | App Quality Insights | App Inspection | Logcat | Build | Profiler | Layout Inspector

Install successfully finished in 2 s 240 ms. (2 minutes ago) — 7:34 LF UTF-8 4 spaces

Nhận xét

- Các luồng chạy bất đồng bộ, không theo đúng thứ tự code
- Dispachers Unconfined và Main đều chạy trên Main thread. Tuy nhiên nếu Dispachers Unconfined chạy quá lâu thì sẽ được chuyển sang Thread mới.

Ví dụ 4

```
object TestDispachers {
    fun runMyFirstCoroutines() {
        GlobalScope.launch(Dispatchers.Unconfined) {
            Log.d(MainActivity::class.java.simpleName, "Before delay -
Dispachers Unconfined run on ${Thread.currentThread().name}")
            delay(1000)
            Log.d(MainActivity::class.java.simpleName, "Dispachers
Unconfined run on ${Thread.currentThread().name}")
        }
        GlobalScope.launch(Dispatchers.Main) {
            Log.d(MainActivity::class.java.simpleName, "Dispachers Main run
on ${Thread.currentThread().name}")
        }
    }
}
```

Kết quả

```
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help   Coroutine Example - TestDispachers.kt [Coroutine_Example.app.main]
routineexample 〉 coroutinecontext 〉 TestDispachers 〉 runMyFirstCoroutines   app   Nexus S API 34
TestDispachers.kt   MainActivity.kt
21        Log.d(MainActivity::class.java.simpleName, msg: "Dispachers Unconfined run on ${Thread.currentThr
22    }
23    GlobalScope.launch(Dispatchers.Main) { this: CoroutineScope
24        Log.d(MainActivity::class.java.simpleName, msg: "Dispachers Main run on ${Thread.currentThread()
25    }
26 }
Logcat:   Logcat
Nexus S API 34 (emulator-5554) Android 14, API 34   MainActivity
V Transition requested: android.os.BinderProxy@31ce525 TransitionRequestInfo { type = OPEN, triggerTask = TaskInfo{us
I START u0 {act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] flg=0x10000000 cmp=vn.edu.hust.soic
I Start proc 5754:vn.edu.hust.soict.gv.quangnh.coroutineexample/u0a199 for next-top-activity {vn.edu.hust.soict.gv.qu
V Sent Transition #9 createdAt=12-02 21:42:11.473 via request=TransitionRequestInfo { type = OPEN, triggerTask = Task
D Before delay - Dispachers Unconfined run on main
D Window{5854e72 u0 vn.edu.hust.soict.gv.quangnh.coroutineexample/vn.edu.hust.soict.gv.quangnh.coroutineexample.MainA
D Dispachers Main run on main
I Displayed vn.edu.hust.soict.gv.quangnh.coroutineexample/.MainActivity for user 0: +3s414ms
D Dispachers Unconfined run on kotlinx.coroutines.DefaultExecutor
Version Control  Run  TODO  Problems  Terminal  Services  App Quality Insights  App Inspection  Logcat  Build  Profiler  Layout Inspector
Install successfully finished in 1 s 76 ms. (a minute ago)   25:10  CRLF  UTF-8  4 spaces
```

2.2. withContext Ví dụ 5

```
fun testMySecondWithContext() {
    GlobalScope.launch(Dispatchers.IO) {
        // Run long time task
        Log.d("myLog", "Run long time task - Thread:
${Thread.currentThread().name}")
        delay(2000)
        withContext(Dispatchers.Main) {
            // Update UI here
            Log.d("myLog", "Update UI - Thread:
${Thread.currentThread().name}")
        }
    }
}
```

Kết quả

```kotlin
fun testMySecondWithContext() {
    GlobalScope.launch(Dispatchers.IO) { this: CoroutineScope
        // Run long time task
        Log.d( tag: "myLog",  msg: "Run long time task - Thread: ${Thread.currentThread().name}")
        delay( timeMillis: 2000)
        withContext(Dispatchers.Main) { this: CoroutineScope
            // Update UI here
            Log.d( tag: "myLog",  msg: "Update UI - Thread: ${Thread.currentThread().name}")
        }
    }
}
```

```
Log    vn....t.gv.quangnh.coroutineexample  D  Run long time task - Thread: DefaultDispatcher-worker-1
Log    vn....t.gv.quangnh.coroutineexample  D  Update UI - Thread: main
```

2.3. Job. Ví dụ 6

```
package vn.edu.hust.soict.gv.quangnh.coroutineexample.coroutinecontext

import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.Job
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch
```

```kotlin
fun main() {
    val job1: Job = GlobalScope.launch {
        delay(2000)
        println("Hello Kotlin")
    }

    val job2: Job = GlobalScope.launch {
        // job2 chờ đợi công việc của job1 hoàn thành rồi mới thực hiện
        job1.join()
        delay(1000)
        println("I'm Coroutine")
    }
    Thread.sleep(4000)
}
```

Kết quả

app × TestJobKt ×

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
Hello Kotlin
I'm Coroutine

Process finished with exit code 0
```

Cancel coroutine. Ví dụ 7

```
fun main() {
    runBlocking {
        val job = launch(Dispatchers.Default) {
            repeat(1000) {
                delay(500)
                println("I'm sleeping $it ...")
            }
        }
        delay(1500)
        job.cancel()
        print("Cancelled coroutines")
    }
}
```

Kết quả

app × TestJobKt ×

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
I'm sleeping 0 ...
I'm sleeping 1 ...
Cancelled coroutines
Process finished with exit code 0
```

Hàm cancelAndJoin(). Ví dụ 8

```
fun main() {
    runBlocking {
        val startTime = System.currentTimeMillis()
        val job = launch(Dispatchers.Default) {
            var nextPrintTime = startTime
            var i = 0
            while (i < 5) { // computation loop, just waste CPU
                // print a message twice a second
                if (System.currentTimeMillis() >= nextPrintTime) {
                    println("job: I'm sleeping ${i++} ...")
                    nextPrintTime += 500
                }
            }
        }
        delay(1400) // delay a bit
        println("main: I'm tired of waiting")
        job.cancelAndJoin() // cancel the job and waits for its completion
        println("main: Now I can quit")
    }
```

Kết quả

app × TestJobKt ×

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
job: I'm sleeping 0 ...
job: I'm sleeping 1 ...
job: I'm sleeping 2 ...
job: I'm sleeping 3 ...
main: I'm tired of waiting
job: I'm sleeping 4 ...
main: Now I can quit

Process finished with exit code 0
```

Biến isActive. Ví dụ 9

```
fun main() {
    runBlocking {
        val startTime = System.currentTimeMillis()
        val job = launch(Dispatchers.Default) {
            var nextPrintTime = startTime
            var i = 0
            while (isActive) { // computation loop, just waste CPU
                // print a message twice a second
                if (System.currentTimeMillis() >= nextPrintTime) {
                    println("job: I'm sleeping ${i++} ...")
                    nextPrintTime += 500
                }
            }
        }
        delay(1400) // delay a bit
        println("main: I'm tired of waiting")
        job.cancelAndJoin() // cancel the job and waits for its completion
        println("main: Now I can quit")
    }
```

Kết quả

app × TestJobKt ×

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
job: I'm sleeping 0 ...
job: I'm sleeping 1 ...
job: I'm sleeping 2 ...
job: I'm sleeping 3 ...
main: I'm tired of waiting
main: Now I can quit

Process finished with exit code 0
```

Coroutine với try … catch … finally

Ví dụ 10

```
fun main() {
    runBlocking {
        val job = launch {
            try {
                repeat(1000) {
                    delay(100)
                    println("Hello Coroutine")
                }
            } finally {
                println("Print from finally")
            }

        }
        delay(300)
        println("I want stop coroutine")
        job.cancel()
    }
}
```

Kết quả

app × TestJobKt ×

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
Hello Coroutine
Hello Coroutine
I want stop coroutine
Print from finally

Process finished with exit code 0
```

Hàm delay trong khối finally Ví dụ 11

```
fun main() {
    runBlocking {
        val job = launch {
            try {
                repeat(1000) {
                    delay(100)
                    println("Hello Coroutine")
                }
            } finally {
                println("Print from finally")
                delay(100)
                println("Please print me last times")
            }

        }
        delay(300)
        println("I want stop coroutine")
        job.cancel()
    }
}
```

Kết quả

app × TestJobKt ×

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
Hello Coroutine
Hello Coroutine
I want stop coroutine
Print from finally

Process finished with exit code 0
```

- Lý do: Hàm delay sẽ check coroutine còn alive hay không, do đó hàm delay và các câu lệnh sau đó không còn chạy

Hàm withContext(NonCancellable) Ví dụ 12

```
fun main() {
    runBlocking {
        val job = launch {
            try {
                repeat(1000) {
                    delay(100)
                    println("Hello Coroutine")
                }
            } finally {
                println("Print from finally")
                withContext(NonCancellable) {
                    repeat(2) {
                        delay(100)
                        println("Print from NonCancellable")
                    }
                }
            }

        }
        delay(300)
        println("I want stop coroutine")
        job.cancel()
    }
}
```

Kết quả

app × TestJobKt ×

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
Hello Coroutine
Hello Coroutine
I want stop coroutine
Print from finally
Print from NonCancellable
Print from NonCancellable

Process finished with exit code 0
```

- Nhận xét: khối lệnh bên trong withContext(NonCancellable) sẽ luôn được thực hiện.

2.4. Timeouts. Ví dụ 13

```
fun main() {
    runBlocking {
        withTimeout(1800) {
            repeat(1000) {
                println("I'm sleeping $it")
                delay(500)


            }
        }
    }
}
```

Nghĩa là coroutine này chỉ chạy tối đa 1800 ms.

Kết quả

app × TestTimeOutKt ×

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
I'm sleeping 0
I'm sleeping 1
I'm sleeping 2
I'm sleeping 3
Exception in thread "main" kotlinx.coroutines.TimeoutCancellationException: Timed out waiting for 1800 ms
    at kotlinx.coroutines.TimeoutKt.TimeoutCancellationException(Timeout.kt:184)
    at kotlinx.coroutines.TimeoutCoroutine.run(Timeout.kt:154)
    at kotlinx.coroutines.EventLoopImplBase$DelayedRunnableTask.run(EventLoop.common.kt:508)
    at kotlinx.coroutines.EventLoopImplBase.processNextEvent(EventLoop.common.kt:284)
    at kotlinx.coroutines.DefaultExecutor.run(DefaultExecutor.kt:108)
    at java.base/java.lang.Thread.run(Thread.java:833)

Process finished with exit code 1
```

Xử lý Exception bằng withTimeoutOrNull Ví dụ 14

```
fun main() {
    runBlocking {
        val result = withTimeoutOrNull(1800) {
            repeat(1000) {
                println("I'm sleeping $it")
                delay(500)
            }
            "Done"
        }
        println("Result = $result")
    }
}
```

Kết quả

app TestTimeOutKt

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
I'm sleeping 0
I'm sleeping 1
I'm sleeping 2
I'm sleeping 3
Result = null

Process finished with exit code 0
```

Nếu thời gian chạy coroutine ít hơn thời gian Timeout. Ví dụ 15

```
fun main() {
    runBlocking {
        val result = withTimeoutOrNull(1800) {
            repeat(2) {
                println("I'm sleeping $it")
                delay(500)
            }
            "Done"
        }
        println("Result = $result")
    }
}
```

Kết quả

app × TestTimeOutKt ×

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
I'm sleeping 0
I'm sleeping 1
Result = Done

Process finished with exit code 0
```

3. Async và Await. Ví dụ 16

```
package
vn.edu.hust.soict.gv.quangnh.coroutineexample.async_await

import kotlinx.coroutines.delay
import kotlinx.coroutines.runBlocking
import kotlin.system.measureTimeMillis
```

```
fun main() {
    runBlocking {
        val time = measureTimeMillis {
            val a = doSomethingFunny1()
            val b = doSomethingFunny2()
            println("a + b = ${a + b}")
        }
        println("Time = $time")
    }
}
suspend fun doSomethingFunny1(): Int {
    delay(1000)
    return 10
}

suspend fun doSomethingFunny2(): Int {
    delay(1000)
    return 20
}
```

Kết quả

app × TestAsyncAwaitKt ×

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
a + b = 30
Time = 2101

Process finished with exit code 0
```

- Như vậy thời gian chạy cần đến 2101 ms (vì là chạy tuần tự).
- Có cách khác để chạy nhanh hơn, đó là sử dụng async – await.

Ví dụ 17: async - await

```
import kotlinx.coroutines.Deferred
import kotlinx.coroutines.async
import kotlinx.coroutines.delay
import kotlinx.coroutines.runBlocking
import kotlin.system.measureTimeMillis
```

```
fun main() {
    runBlocking {
        val time = measureTimeMillis {
            val a: Deferred<Int> = async { doSomethingFunny1() }
            val b: Deferred<Int> = async { doSomethingFunny2() }
            println(a.await() + b.await())
        }
        println("Time = $time")
    }
}
suspend fun doSomethingFunny1(): Int {
    delay(1000)
    return 10
}

suspend fun doSomethingFunny2(): Int {
    delay(1000)
    return 20
}
```

Kết quả

app × TestAsyncAwaitKt ×

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
30
Time = 1146

Process finished with exit code 0
```

4. CoroutineScope

```
fun main() {
    runBlocking { this: CoroutineScope
        launch { this: CoroutineScope


        }
        async { this: CoroutineScope

        } ^runBlocking
    }
}
```

- Nhận xét: cả runBlocking, launch và async đều chạy trong một CoroutineScope

Ví dụ 18

```
fun main() {
    runBlocking {

        val job1 = launch {
            launch {
                delay(100)
                println("coroutine 1: Hello")
                delay(1000)
                println("coroutine 1: Goodbye")
            }
            launch {
                delay(100)
                println("coroutine 2: Hello")
                delay(1000)
                println("coroutine 2: Goodbye")
            }
        }
        delay(500)
        job1.cancel()

    }
}
```

Kết quả

app × TestCoroutineScopeKt ×

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
coroutine 1: Hello
coroutine 2: Hello

Process finished with exit code 0
```

- Nhận xét: coroutine cha bị cancel thì coroutine con cũng bị hủy theo.
- Nếu tác vụ nhất thiết phải hoàn thành kể cả khi coroutine cha bị hủy thì dùng GlobalScope.

Ví dụ 19

```
fun main() {
    runBlocking {

        val job1 = launch {
            launch {
                delay(100)
                println("coroutine 1: Hello")
                delay(1000)
                println("coroutine 1: Goodbye")
            }
            launch {
                delay(100)
                println("coroutine 2: Hello")
                delay(1000)
                println("coroutine 2: Goodbye")
            }
            GlobalScope.launch {
                delay(100)
                println("coroutine 3: Hello")
                delay(1000)
                println("coroutine 3: Goodbye")
            }
        }
```

```
        delay(500)
        job1.cancel()
        delay(2500)

    }
}
```

Kết quả

app × TestCoroutineScopeKt ×

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
coroutine 1: Hello
coroutine 2: Hello
coroutine 3: Hello
coroutine 3: Goodbye

Process finished with exit code 0
```

Ví dụ 20

```
fun main() {
    runBlocking {
        val job = launch {
            repeat(3) {
                delay(100)
                println("coroutine: $it")
            }
            println("Print from parent")
        }
        job.join()
        delay(1000)
    }
}
```

Kết quả

app × TestCoroutineScopeKt ×

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
coroutine: 0
coroutine: 1
coroutine: 2
Print from parent

Process finished with exit code 0
```

Ví dụ 21

```
fun main() {
    runBlocking {
        val job = launch {
            repeat(3) {
                launch {
                    delay(100)
                    println("coroutine: $it")
                }
            }
            println("Print from parent")
        }
        job.join()
        delay(1000)
    }
}
```

Kết quả

app × TestCoroutineScopeKt ×

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
Print from parent
coroutine: 0
coroutine: 1
coroutine: 2

Process finished with exit code 0
```

5. Xử lý Exception và Supervision trong Coroutine

Ví dụ 22

```
fun main() {
    runBlocking {
        val job = GlobalScope.launch {
            println("Throw Exception from Launch")
            throw NullPointerException()
        }
        // chờ đợi coroutine hoàn thành
        job.join()
        val deferred = GlobalScope.async {
            println("Throw Exception from Async")
            throw IndexOutOfBoundsException()
        }
    }
}
```

Kết quả

```
TestExceptionHandlingKt
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
Throw Exception from Launch
Exception in thread "DefaultDispatcher-worker-1" java.lang.NullPointerException Create breakpoint
    at vn.edu.hust.soict.gv.quangnh.coroutineexample.exception_handling.TestExceptionHandlingKt$main$1$job$1
     .invokeSuspend(TestExceptionHandling.kt:12)
    at kotlin.coroutines.jvm.internal.BaseContinuationImpl.resumeWith(ContinuationImpl.kt:33) <5 internal lines>
    Suppressed: kotlinx.coroutines.DiagnosticCoroutineContextException:
     [StandaloneCoroutine{Cancelling}@46b9e67a, Dispatchers.Default]
Throw Exception from Async

Process finished with exit code 0
```

- Lý do async không tạo ra các thông báo lỗi vì các thông báo lỗi này đã được bắt bởi biến deferred.

Khi thêm câu lệnh await() Ví dụ 23

```
fun main() {
    runBlocking {
        val job = GlobalScope.launch {
            println("Throw Exception from Launch")
            throw NullPointerException()
        }
        // chờ đợi coroutine hoàn thành
        job.join()
        val deferred = GlobalScope.async {
            println("Throw Exception from Async")
            throw IndexOutOfBoundsException()
        }
        deferred.await()
    }
}
```

Kết quả

TestExceptionHandlingKt

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
Throw Exception from Launch
Exception in thread "DefaultDispatcher-worker-1" java.lang.NullPointerException Create breakpoint
    at vn.edu.hust.soict.gv.quangnh.coroutineexample.exception_handling.TestExceptionHandlingKt$main$1$job$1
     .invokeSuspend(TestExceptionHandling.kt:12)
    at kotlin.coroutines.jvm.internal.BaseContinuationImpl.resumeWith(ContinuationImpl.kt:33) <5 internal lines>
    Suppressed: kotlinx.coroutines.DiagnosticCoroutineContextException:
     [StandaloneCoroutine{Cancelling}@34039da4, Dispatchers.Default]
Throw Exception from Async
Exception in thread "main" java.lang.IndexOutOfBoundsException Create breakpoint
    at vn.edu.hust.soict.gv.quangnh.coroutineexample.exception_handling
     .TestExceptionHandlingKt$main$1$deferred$1.invokeSuspend(TestExceptionHandling.kt:18)
    at kotlin.coroutines.jvm.internal.BaseContinuationImpl.resumeWith(ContinuationImpl.kt:33) <5 internal lines>

Process finished with exit code 1
```

```
fun main() {
    runBlocking {
        val job = GlobalScope.launch {
            try {
                println("Throw Exception from Launch")
                throw NullPointerException()
            } catch (e: NullPointerException) {
                println(e.toString())
            }
        }
        // chờ đợi coroutine hoàn thành
        job.join()
        val deferred = GlobalScope.async {
            println("Throw Exception from Async")
            throw IndexOutOfBoundsException()
        }
        try {
            deferred.await()
        } catch (e: IndexOutOfBoundsException) {
            println(e.toString())
        }
```

Xử lý lỗi trong coroutine dùng try ... catch

Ví dụ 24

Kết quả

TestExceptionHandlingKt

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
Throw Exception from Launch
java.lang.NullPointerException
Throw Exception from Async
java.lang.IndexOutOfBoundsException

Process finished with exit code 0
```

```
fun main() {
    runBlocking {
        val handler = CoroutineExceptionHandler { _, exception ->
            println("Error here: ${exception.toString()}")
        }
        val job = GlobalScope.launch(handler) {
            println("Throw Exception from Launch")
            throw NullPointerException()
        }
        // chờ đợi coroutine hoàn thành
        job.join()
        val deferred = GlobalScope.async {
            println("Throw Exception from Async")
            throw IndexOutOfBoundsException()
        }
        try {
            deferred.await()
        }catch (e: IndexOutOfBoundsException) {
            println(e.toString())
        }
    }
}
```

Bắt lỗi với CoroutineExceptionHandler Ví dụ 25

Kết quả

TestExceptionHandlingKt

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
Throw Exception from Launch
Error here: java.lang.NullPointerException
Throw Exception from Async
java.lang.IndexOutOfBoundsException

Process finished with exit code 0
```

Bắt lỗi + chỉ định context

```
val job = GlobalScope.launch(handler + Dispatchers.Default) {
    println("Throw Exception from Launch")
    throw NullPointerException()
}
```

CoroutineExceptionHandler không bắt được lỗi với async Ví dụ 26

```
fun main() {
    runBlocking {
        val handler = CoroutineExceptionHandler { _, exception ->
            println("Error here: ${exception.toString()}")
        }
        val job = GlobalScope.launch(handler + Dispatchers.Default) {
            println("Throw Exception from Launch")
            throw NullPointerException()
        }
        // chờ đợi coroutine hoàn thành
        job.join()
        val deferred = GlobalScope.async(handler) {
            println("Throw Exception from Async")
            throw IndexOutOfBoundsException()
        }
        deferred.await()
    }
}
```

Kết quả

```
TestExceptionHandlingKt
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
Throw Exception from Launch
Error here: java.lang.NullPointerException
Throw Exception from Async
Exception in thread "main" java.lang.IndexOutOfBoundsException Create breakpoint
    at vn.edu.hust.soict.gv.quangnh.coroutineexample.exception_handling
     .TestExceptionHandlingKt$main$1$deferred$1.invokeSuspend(TestExceptionHandling.kt:23)
    at kotlin.coroutines.jvm.internal.BaseContinuationImpl.resumeWith(ContinuationImpl.kt:33) <5 internal lines>

Process finished with exit code 1
```

- Tổng kết:
  - Sử dụng CoroutineExceptionHandler để bắt lỗi với coroutine tạo ra bằng launch.
  - Sử dụng try … catch để bắt lỗi với coroutine tạo ra bằng async.

Nếu trong Coroutine cha có nhiều coroutine con, và các coroutine con có khả năng tạo ra các lỗi. Ví dụ 27

- Khi Coroutine thứ 2 throw Exception thì các coroutine khác sẽ dừng.

```
fun main() {
    runBlocking {
        val handle = CoroutineExceptionHandler {_, exception ->
            println("Exception: $exception")
        }
        val job = GlobalScope.launch(handle) {
            launch {
                println("Coroutine 1")
                delay(300)
                println("Coroutine 1 continue")
                throw IndexOutOfBoundsException("Coroutine 1")
            }
```

```
launch {
        println("Coroutine 2")
        delay(200)
        throw NullPointerException("Coroutine 2")
    }
    launch {
        println("Coroutine 3")
        delay(400)
        println("Coroutine 3 continue")
        throw ArithmeticException("Coroutine 3")
    }
  }
  job.join()
  delay(1000)
 } // end of runBlocking
}
```

Kết quả

TestExceptionHandlingKt

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
Coroutine 1
Coroutine 2
Coroutine 3
Exception: java.lang.NullPointerException: Coroutine 2

Process finished with exit code 0
```

Bắt lỗi với suppressed. Ví dụ 28

```
fun main() {
    runBlocking {
        val handle = CoroutineExceptionHandler {_, exception ->
            println("Exception: $exception with suppressed
${exception.suppressed.contentToString()}")
        }
        val job = GlobalScope.launch(handle) {
            launch {
                println("Coroutine 1")
                delay(300)
                println("Coroutine 1 continue")
                throw IndexOutOfBoundsException("Coroutine 1")
            }
```

```
launch {
            try {
                delay(Long.MAX_VALUE)
            } finally {
                throw ArithmeticException("Coroutine 2")
            }
        }

        launch {
            println("Coroutine 3")
            delay(400)
            println("Coroutine 3 continue")
            throw ArithmeticException("Coroutine 3")
        }
    }
    job.join()
    delay(1000)
  } // end of runBlocking
}
```

Kết quả

```
TestExceptionHandlingKt
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
Coroutine 1
Coroutine 3
Coroutine 1 continue
Exception: java.lang.IndexOutOfBoundsException: Coroutine 1 with suppressed [java.lang.ArithmeticException:
 Coroutine 2]

Process finished with exit code 0
```

- Exception aggregation: When multiple children of a coroutine fail with an exception, the general rule is "the first exception wins", so the first exception gets handled.
- All additional exceptions that happen after the first one are attached to the first exception as suppressed ones.

SupervisorJob và SupervisorScope Ví dụ 29

```
fun main() {
    runBlocking {
        val supervisorJob = SupervisorJob()
        with(CoroutineScope(coroutineContext +
supervisorJob)) {
            val firstChild = launch {
                println("Print from First Child")
                throw NullPointerException()
            }
```

```
val secondChild = launch {
            firstChild.join()
            println("print from second Child. First Child is
Active: ${firstChild.isActive}")
            try {
                delay(1000)
            } finally {
                println("Second Child Cancelled")
            }
        }
        firstChild.join()
        println("Cancelling SupervisorJob")
        supervisorJob.cancel()
        secondChild.join()
    }
  }
}
```

Kết quả

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
Print from First Child
Exception in thread "main" java.lang.NullPointerException Create breakpoint
    at vn.edu.hust.soict.gv.quangnh.coroutineexample.exception_handling
     .TestExceptionHandlingKt$main$1$1$firstChild$1.invokeSuspend(TestExceptionHandling.kt:20)
    at kotlin.coroutines.jvm.internal.BaseContinuationImpl.resumeWith(ContinuationImpl.kt:33) <7 internal lines>
    at vn.edu.hust.soict.gv.quangnh.coroutineexample.exception_handling.TestExceptionHandlingKt.main
     (TestExceptionHandling.kt:15)
    at vn.edu.hust.soict.gv.quangnh.coroutineexample.exception_handling.TestExceptionHandlingKt.main
     (TestExceptionHandling.kt)
    Suppressed: kotlinx.coroutines.DiagnosticCoroutineContextException:
     [StandaloneCoroutine{Cancelling}@cb0ed20, BlockingEventLoop@8e24743]
print from second Child. First Child is Active: false
Cancelling SupervisorJob
Second Child Cancelled

Process finished with exit code 0
```

- Nhận xét: secondChild vẫn chạy ngay cả khi firstChild đã throw Exception

SupervisorScope
Ví dụ 30

```
fun main() {
    runBlocking {
        supervisorScope {
            val firstChild = launch {
                println("Print from First Child")
                throw NullPointerException()
            }
            val secondChild = launch {
                firstChild.join()
                println("print from second Child. First Child is Active:
${firstChild.isActive}")
                try {
                    delay(1000)
                } finally {
                    println("Second Child Cancelled")
                }
            }
            firstChild.join()
            secondChild.join()
        }
    }
}
```

Kết quả

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
Print from First Child
Exception in thread "main" java.lang.NullPointerException Create breakpoint
    at vn.edu.hust.soict.gv.quangnh.coroutineexample.exception_handling
     .TestExceptionHandlingKt$main$1$1$firstChild$1.invokeSuspend(TestExceptionHandling.kt:19)
    at kotlin.coroutines.jvm.internal.BaseContinuationImpl.resumeWith(ContinuationImpl.kt:33) <7 internal lines>
    at vn.edu.hust.soict.gv.quangnh.coroutineexample.exception_handling.TestExceptionHandlingKt.main
     (TestExceptionHandling.kt:15)
    at vn.edu.hust.soict.gv.quangnh.coroutineexample.exception_handling.TestExceptionHandlingKt.main
     (TestExceptionHandling.kt)
    Suppressed: kotlinx.coroutines.DiagnosticCoroutineContextException:
     [StandaloneCoroutine{Cancelling}@76a3e297, BlockingEventLoop@4d3167f4]
print from second Child. First Child is Active: false
Second Child Cancelled

Process finished with exit code 0
```

6. Sequence trong Kotlin
Ví dụ 31

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

fun main() {
    foo(10).forEach {
        println(it)
    }
}
```

Kết quả

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
0
2
4
6
8
10

Process finished with exit code 0
```

Kết hợp sequence với map

```
fun main() {
    foo(10).map{it * it}.forEach {
        println(it)
    }
}
```

- Kết quả

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
0
4
16
36
64
100

Process finished with exit code 0
```

Kết hợp sequence với filter

```
fun main() {
    foo(10).filter { it < 8 }.forEach {
        println(it)
    }
}
```

- Kết quả

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
0
2
4
6

Process finished with exit code 0
```

7. Giới thiệu về Flow trong Kotlin Coroutines
Ví dụ 32

```
fun main() {
    runBlocking {
        val foo = foo(200)
        foo(5).collect {
            println("i = $it")
        }
    }
}

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

Kết quả

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
i = 0
i = 1
i = 2
i = 3
i = 4
i = 5

Process
```

- Nhận xét:
  - Flow chỉ cấp dữ liệu khi cần, do đó kết quả được in ra mà không cần đợi 200 giây do lệnh val foo = foo(200).
  - Flow chạy bất đồng bộ nên không ảnh hưởng đến Thread hiện tại.