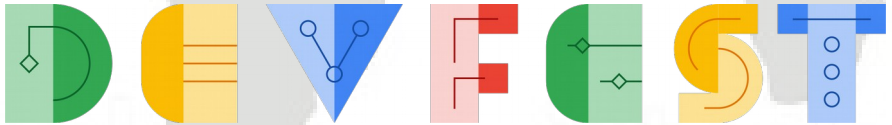
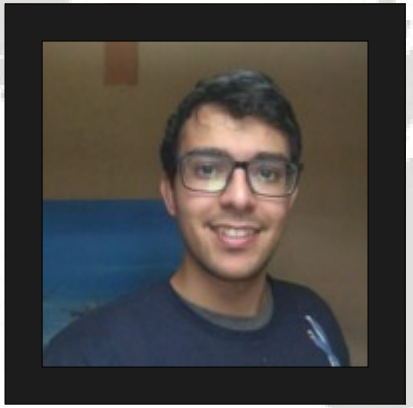
Programação Assíncrona com Kotlin Coroutines



2019



Lucas Borsatto

ame

Async: Concorrência & Paralelismo

Sequencial

1. CPU ociosa

2. Main Thread bloqueada

3. Congelamento do app

Async

Concorrente

Task inicia, roda e completa entre intervalos

Paralelismo

Task roda assíncronamente

Exemplo:
Atualização de PDVs

Abordagens

1. Síncrona
2. Com Threads
3. Com Coroutines

Síncrono

Síncrono - Código

```
fun getAmePrices() : List<Price> { ... }
fun getAmeProducts(): List<Product> { ... }
fun savePriceProductToStore(prices: List<Price>, products: List<Product>) { ... }
fun postNotificationToPOS(posId: Int) : Response { ... }


fun updatePosAssortment(posId): Response{
    val prices = getAmePrices()
    val products = getAmeProducts()
    val savePriceAndProductToPOS(prices, products)
    return postNotificationToPOS(posId)
}
```

Threads

Threads - Interface

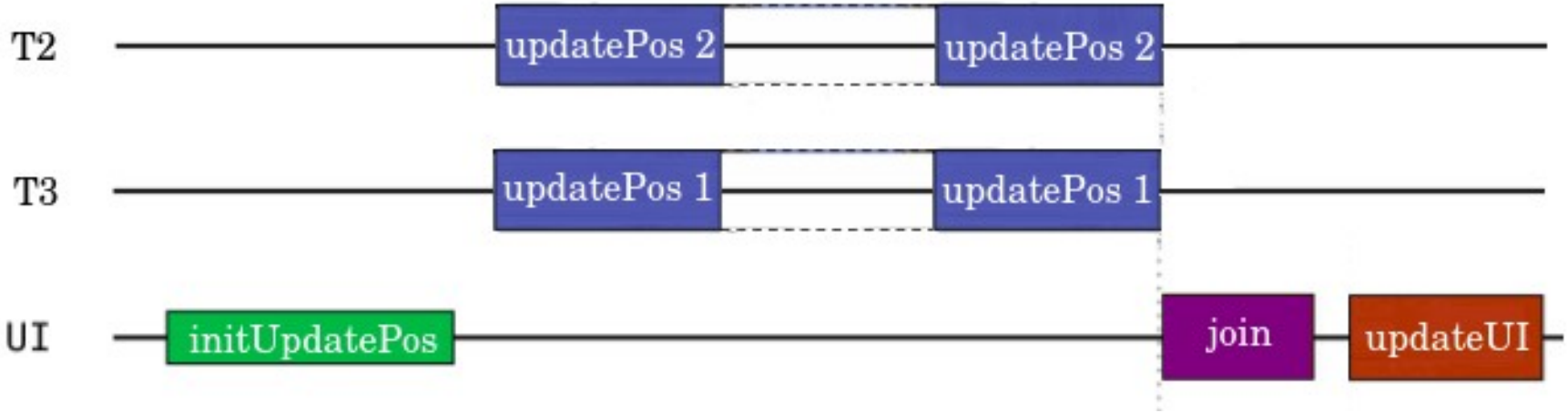
```
fun thread(
        start: Boolean = true,
        isDaemon: Boolean = false,
        contextClassLoader: ClassLoader? = null,
        name: String? = null,
        priority: Int = -1,
        block: () -> Unit
): Thread
```

Threads - Código

```
fun updatePosAssortment(posId: Int): Response{
    val prices = getAmePrices()
    val products = getAmeProducts()
    val savePriceAndProductToPOS(prices, products)
    return postNotificationToPOS(posId)
}

for(pos in posList){
    thread(start=true){
        updatePosAssortment(pos.posId)
    }
}
```

Threads - Fluxo



Threads - Desvantagens

Quanto mais threads:

1. Maior trabalho pro SO
2. Maior consumo de memória
3. Maior tempo de CPU ociosa

Coroutines

Coroutines - Interface

```
fun CoroutineScope.launch(
        context: CoroutineContext = EmptyCoroutineContext,
        start: CoroutineStart = CoroutineStart.DEFAULT,
        block: suspend CoroutineScope.() -> Unit
): Job (source)
```

Coroutines - Código

```
suspend fun getAmePrices() : List<Price> { ... }
suspend fun getAmeProducts(): List<Product> { ... }
suspend fun savePriceProductToStore(prices: List<Price>, products: List<Product>)
{ ... }
suspend fun postNotificationToPOS(posId: Int) : Response { ... }

suspend fun updatePosAssortment(posId): Response{
    val prices = getAmePrices()
    val products = getAmeProducts()
    val savePriceProductToPOS(prices, products)
    return postNotificationToPOS(posId)
}
```

Coroutines - Código

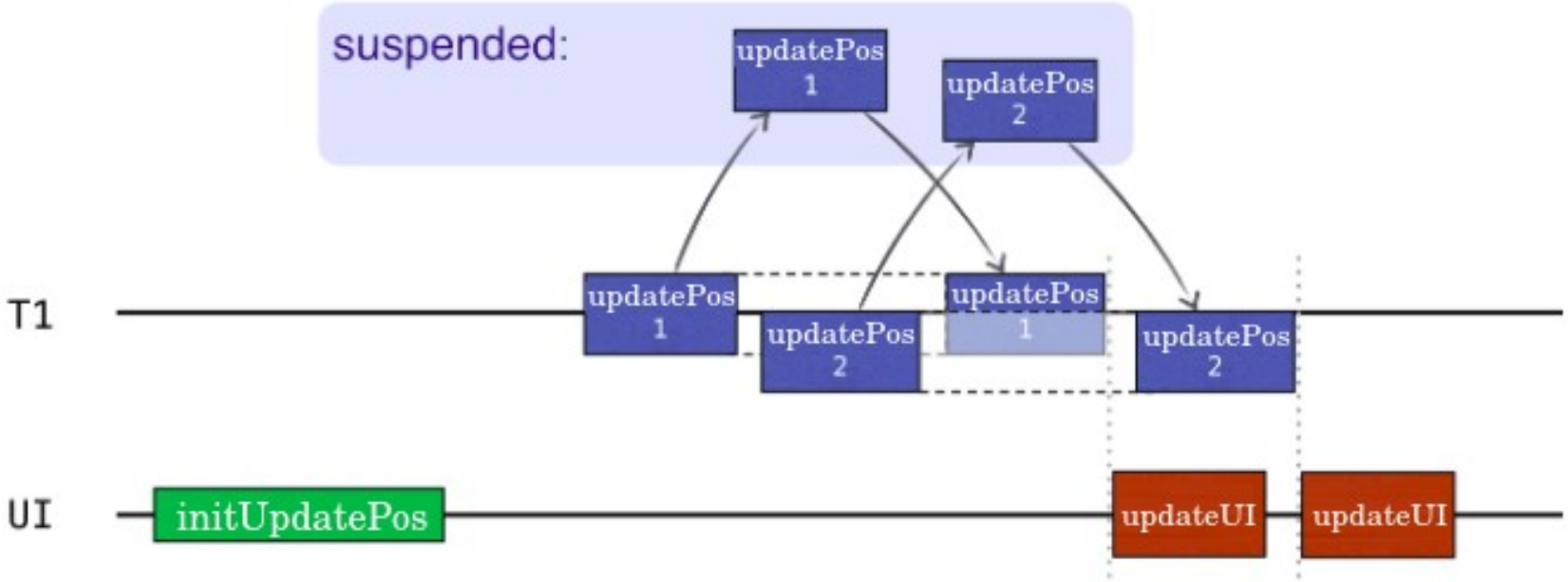
```
runBlocking {
    posList.forEach {
        launch(Dispatchers.Default) {
            updatePosAssortment(it.posId)
        }
    }
}
```

Coroutines - Código

```
val responses = mutableListOf<Response>()

runBlocking {
    posList.map {
        async(Dispatchers.IO) {
            updatePosAssortment(it.posId).await()
        }
    }.forEach {
        responses.add(it.await())
    }
}
```

Coroutines - Processo



Mas como funciona?

Continuation on Direct Style

```
fun updatePosAssortment(posId: Int): Response{
    val prices = getAmePrices()
    val products = getAmeProducts()
    val savePriceProductToPOS(prices, products)
    return postNotificationToPOS(posId)
}
```

Continuation on Direct Style

```
fun updatePosAssortment(posId: Int): Response{
    val prices = getAmePrices()
    val products = getAmeProducts()
    val savePriceProductToPOS(prices, products)
    return postNotificationToPOS(posId)
}
```

Continuation

Continuation on Direct Style

```
fun updatePosAssortment(store): Response{
    val prices = getAmePrices()
    val products = getAmeProducts()
    val savePriceProductToPOS(prices, products)
    return postNotificationToPOS(posId)
}
```

Continuation

Continuation-Passing Style

```
fun updatePosAssortment(store): Response{
    getAmePrices { prices ->
        val products = getAmeProducts()
        val savePriceProductToPOS(prices, products)
        postNotificationToPOS(posId)
    }
}
```

Continuation

Continuation-Passing Style

```
fun updatePosAssortment(store): Response{
    getAmePrices { prices ->
        getAmeProducts { products ->
            val savePriceProductToPOS(prices, products)
            postNotificationToPOS(posId)
        }
    }
}
```

Continuation

Continuation-Passing Style

```
fun updatePosAssortment(store): Response{
    getAmePrices { prices ->
        getAmeProducts { products ->
            savePriceProductToPOS(prices, products) {
                postNotificationToPOS(posId)
            }
        }
    }
}
```

Continuation

Continuation-Passing Style

```
fun updatePosAssortment(store): Response{
    getAmePrices { prices ->
        getAmeProducts { products ->
            savePriceProductToPOS(prices, products) {
                postNotificationToPOS(posId)
            }
        }
    }
}
```

Continuation

CPS em Kotlin

```kotlin
suspend fun updatePosAssortment(posId : Int): Response { ... }
```

Compilação JVM

```
Object updatePosAssortment(Int posId, Continuation<Response> cont) { ... }
```

Máquina de estados

```
suspend fun updatePosAssortment(posId: Int): Response{
    //LABEL 0
    val prices = getAmePrices()
    //LABEL 1
    val products = getAmeProducts()
    //LABEL 2
    val savePriceProductToPOS(prices, products)
    //LABEL 3
    return postNotificationToPOS(posId)
}
```

Máquina de estados

```
suspend fun updatePosAssortment(posId: Int): Response{
    switch (label){
        case 0:
            val prices = getAmePrices()
        case 1:
            val products = getAmeProducts()
        case 2:
            val savePriceProductToPOS(prices, products)
        case 3:
        postNotificationToPOS(posId)
    }
}
```

Máquina de estados

```
suspend fun updatePosAssortment(posId: Int): Response{
    val sm = object : CoroutineImpl { ... }
    switch (label){
        case 0:
            sm.label = 1
            sm.posId = posId
            val prices = getAmePrices(sm)
        case 1:
            val products = getAmeProducts()
        case 2:
            val savePriceProductToPOS(prices, products)
        case 3:
        postNotificationToPOS(posId)
    }
}
```

CPU Schedulers

Preemptivo	Não-preempivo
Processo mantido na CPU até terminar ou ficar em espera	Processo mantido na CPU por uma quantidade determinada de tempo

Pontos relevantes

1. Coroutines com Java
2. Bibliotecas Java são usadas em Kotlin
3. “Coroutines are light-weight threads”

PERGUNTAS?

TEMOS VAGAS

ame