

# *Programação Assíncrona com Kotlin Coroutines*

*Wellington Costa Pereira*

# Wellington Costa Pereira



- +4 anos de experiência
- Mestrando em Ciências da Computação
- Android Leader na Resource IT Solutions
- Open source <3
- Compartilhar conhecimento <3

GitHub: <https://github.com/WellingtonCosta>

E-mail: [wellington.costa128@gmail.com](mailto:wellington.costa128@gmail.com)

**FIGHT !!**

# *Exemplo*

```
fun updateUserData(username: String) {  
    val user = fetchUserFromApi(username)  
    val repos = fetchReposFromApi(username)  
    saveUserDataInDb(user, repos)  
}
```

# *Estilo Tradicional*

```
fun updateUserData(username: String) {  
    val user = fetchUserFromApi(username)  
    val repos = fetchReposFromApi(username)  
    saveUserDataInDb(user, repos)  
}
```

# *Estilo Tradicional*

```
fun updateUserData(username: String) {  
    val user = fetchUserFromApi(username)  
    val repos = fetchReposFromApi(username)  
    saveUserDataInDb(user, repos)  
}
```

# *Estilo Tradicional*

```
fun updateUserData(username: String) {  
    val user = fetchUserFromApi(username)  
    val repos = fetchReposFromApi(username)  
    saveUserDataInDb(user, repos)  
}
```

*Callbacks to the rescue*



# Callback

```
fun updateUserData(username: String) {  
    fetchUserFromApi(username) { user ->  
        fetchReposFromApi(username) { repos ->  
            saveUserDataInDb(user, repos)  
        }  
    }  
}
```

# Callback

```
fun updateUserData(username: String) {  
    fetchUserFromApi(username) { user ->  
        fetchReposFromApi(username) { repos ->  
            saveUserDataInDb(user, repos)  
        }  
    }  
}
```



*Coroutines to the rescue*

# Coroutines

```
suspend fun updateUserData(username: String) {  
    val user = fetchUserFromApi(username)  
    val repos = fetchReposFromApi(username)  
    saveUserDataInDb(user, repos)  
}
```

# Coroutines

```
suspend fun updateUserData(username: String) {  
    val user = fetchUserFromApi(username)  
    val repos = fetchReposFromApi(username)  
    saveUserDataInDb(user, repos)  
}
```

**ASYN**



**ALL THE THINGS!**

# *Principais Conceitos*

# *Principais Conceitos*

- Modificador **suspend**
- *launch* vs *async*
- Dispatchers
- *withContext*
- *coroutineScope*
- *runBlocking*



# *Modificador* **suspend**

A suspending function is simply a function that **can be paused and resumed at a later time**. They can execute a long running operation and wait for it to complete **without blocking**.

# Modificador **suspend**

```
suspend fun updateUserData(username: String) {  
    val user = fetchUserFromApi(username)  
    val repos = fetchReposFromApi(username)  
    saveUserDataInDb(user, repos)  
}
```

# *launch vs async*

**launch:** A background job. Conceptually, a job is a cancellable thing with a life-cycle that culminates in its completion. It returns a **Job** instance.

**async:** Deferred value is a non-blocking cancellable future. It's a Job that **has a result**. It returns a **Deferred** instance.

# *launch vs async*

```
fun doSomething() {  
    launch {  
        // do some operation  
    }  
}
```

# *launch vs async*

```
private val jobs = ArrayList<Job>()
```

```
fun doSomething() {  
    jobs.add(  
        launch {  
            // do some operation  
        }  
    )  
}
```

```
override fun onCleared() {  
    jobs.forEach { if(!it.isCancelled) it.cancel() }  
}
```

# *launch vs async*

```
fun doSomething() {  
    async {  
        // do some operation  
    }  
}
```

# *launch vs async*

```
fun doSomething() {  
    val deferred = async {  
        // do some operation  
    }  
    val result = deferred.await()  
}
```

# *launch vs async*

```
fun doSomething() {  
    val result = async {  
        // do some operation  
    }.await()  
}
```



# *Dispatchers*

- Dispatchers.**Main**: A coroutine dispatcher that is confined to the Main thread operating with UI objects.
- Dispatchers.**Default**: A coroutine dispatcher to execute CPU-bound tasks.
- Dispatchers.**IO**: A coroutine dispatcher that is designed for offloading blocking IO tasks to a shared pool of threads.

# *withContext*

Calls the suspending block with a given **coroutine context**, suspends until it completes and then returns the result.

# *withContext*

```
suspend fun updateUserData(username: String) {  
    withContext(Dispatchers.IO) {  
        val user = fetchUserFromApi(username)  
        val repos = fetchReposFromApi(username)  
        saveUserDataInDb(user, repos)  
    }  
}
```

# *withContext*

```
suspend fun updateUserData(username: String) {  
    withContext(Dispatchers.IO) {  
        val user = fetchUserFromApi(username)  
        val repos = fetchReposFromApi(username)  
        saveUserDataInDb(user, repos)  
    }  
}
```

# *withContext*

```
suspend fun updateUserData(username: String) {  
    withContext(Dispatchers.IO) {  
        val user = fetchUserFromApi(username)  
        val repos = fetchReposFromApi(username)  
        saveUserDataInDb(user, repos)  
    }  
}
```

# *coroutineScope*

This function is designed for a ***parallel decomposition*** of work. When **any child coroutine in this scope fails**, this scope fails and all the rest of the **children are cancelled**.

# *coroutineScope*

```
suspend fun processPayments(): List<Payment> {  
    coroutineScope {  
        val payments = withContext(Dispatchers.IO) {  
            async { loadPayments() }  
        }  
  
        val processedPayments = withContext(Dispatchers.Default) {  
            async { calculateInterestRate(payments.await()) }  
        }  
  
        return processedPayments.await()  
    }  
}
```

# *runBlocking*

Runs new coroutine and **blocks** current thread until its completion.



# *runBlocking*

```
suspend fun processPayments(): List<Payment> {  
    // load payments and calculate its interest rate  
}
```

```
@Test fun myAwesomeTest() {  
    val payments = processPayments()  
    // assert payments interest rate  
}
```

# *runBlocking*

```
suspend fun processPayments(): List<Payment> {  
    // load payments and calculate its interest rate  
}
```

```
@Test fun myAwesomeTest() = runBlocking {  
    val payments = processPayments()  
    // assert payments interest rate  
}
```

*Perguntas?*



*That's all Folks!*

*Obrigado!*