Лекция #32. Корутины

В последнее время поддержка асинхронности и параллельных вычислений стала неотъемлимой чертой многих языков программирования. И Kotlin не является исключением. Зачем нужны асинхронность и параллельные вычисления? Параллельные вычисления позволяют выполнять несколько задач одновременно, а асинхронность позволяет не блокировать основной ход приложения во время выполнения задачи, которая занимает продолжительное время. Например, мы создаем графическое приложение для десктопа или мобильного устройства. И нам надо по нажатию на кнопку отправлять запрос к интернет-ресурсу. Однако подобный запрос может занять довольно много время. И чтобы приложение не зависало на период отправки запроса, подобные запросы к интернет-ресурсам следует отправлять асинхронно. При асинхронных запросах пользователь не ждет пока придет ответ от интернет-ресурса, а продолжает работу с приложением, а при получении ответа получит соответствующее уведомление.

В языке Kotlin поддержка асинхронности и параллельных вычислений воплощена в виде **корутин (coroutine)**. По сути корутина представляет блок кода, который может выполняться параллельно с остальным кодом. А базовая функциональность, связанная с корутинами, сосредоточена в библиотеке **kotlinx.coroutines**.

Рассмотрим определение и применение корутины на простейшем примере.

Добавим библиотеки для работы корутин в наше приложение. Для этого добавим

```
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.5.0'
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.5.0'
```

в зависимости нашего приложения в **build.gradle**

Затем опишем интерфейс нашего приложения в **activity_main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/mainConstraint"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textViewResult"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:layout_marginLeft="16dp"
        android:layout_marginRight="16dp"
        android:text="Result: "
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/editTextNumber" />

    <Button
        android:id="@+id/buttonCalculate"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginEnd="16dp"
        android:layout_marginBottom="16dp"
        android:text="Calculate"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

    <ProgressBar
        android:id="@+id/progressBar"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginEnd="16dp"
        android:layout_marginBottom="8dp"
        android:indeterminate="true"
        android:visibility="invisible"
        app:layout_constraintBottom_toTopOf="@+id/buttonCalculate"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        tools:visibility="visible" />

    <EditText
        android:id="@+id/editTextNumber"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="16dp"
        android:layout_marginEnd="16dp"
        android:ems="10"
        android:inputType="number"
        android:maxLength="2"
        android:minHeight="48dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@+id/textView"
```

```
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:text="Введите число:"
        app:layout_constraintBottom_toBottomOf="@+id/editTextNumber"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="@+id/editTextNumber" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Теперь у нас есть интерфейс приложения и мы можем перейти к его коду. Для этого откроем **MainActivity.kt.**

```
package com.awkitsune.coroutinesdemonstration

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.*
import kotlinx.coroutines.*
import kotlin.coroutines.CoroutineContext

class MainActivity : AppCompatActivity(), CoroutineScope {
    private var job: Job = Job()

    override val coroutineContext: CoroutineContext
        get() = Dispatchers.Default + job

    override fun onDestroy() {
        super.onDestroy()
        job.cancel()
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val buttonCalculate = findViewById<Button>(R.id.buttonCalculate)

        buttonCalculate.setOnClickListener {
            launch {
                val loadingProgressBar =
findViewById<ProgressBar>(R.id.progressBar)

                loadingProgressBar.post{
                    loadingProgressBar.visibility = View.VISIBLE
                }

                calculateFactorial()

                loadingProgressBar.post{
                    loadingProgressBar.visibility = View.INVISIBLE
                }
            }
        }
    }

    suspend fun calculateFactorial() {
```

```
        var result = GlobalScope.async {
            val numberEditText = findViewById<EditText>(R.id.editTextNumber)
            factorial(numberEditText.text.toString().toInt())
        }

        GlobalScope.launch {
            val resultTextView = findViewById<TextView>(R.id.textViewResult)
            resultTextView.text = "Result: ${result.await()}"
        }

        result.join()
    }

    suspend fun factorial(num: Int): Long{
        var result: Long = 1

        for (i in 2..num) {
            result *= i
            delay(500)
        }

        return result
    }
}
```

В нашем приложении будет вычисляться факториал числа, а чтобы увеличить время вычислений нам нужно создать **останавливаемую функцию**, которая помечается ключевым словом **suspend**:

```
suspend fun factorial(num: Int): Long{
    var result: Long = 1

    for (i in 2..num) {
        result *= i
        delay(500)
    }

    return result
}
```

Вызовы таких функций могут приостановить выполнение **сопрограммы** (библиотека может принять решение продолжать работу без приостановки, если результат вызова уже доступен). Функции остановки могут иметь параметры и возвращать значения точно так же, как и все обычные функции, но они могут быть вызваны только из сопрограмм или других функций остановки. В конечном итоге, при старте сопрограммы она должна содержать как минимум одну функцию остановки, и функция эта обычно анонимная (лямбда-функция остановки).

В самом начале нашего Activity мы объявляем

```
private var job: Job = Job()
```

и

```
override val coroutineContext: CoroutineContext
    get() = Dispatchers.Default + job
```

Это необходимо для того, чтобы изменить контекст корутины и иметь возможность завершить её при закрытии приложения и недопустить утечку памяти:

```
override fun onDestroy() {
    super.onDestroy()
    job.cancel()
}
```

Так же мы назначаем обработчик нажатия на кнопку, где мы показываем полосу загрузки, вычисляем значение и скрываем полосу загрузки. Здесь используется метод **launch** для запуска асинхронного кода:

```
launch {
    val loadingProgressBar = findViewById<ProgressBar>(R.id.progressBar)

    loadingProgressBar.post{
        loadingProgressBar.visibility = View.VISIBLE
    }

    calculateFactorial()

    loadingProgressBar.post{
        loadingProgressBar.visibility = View.INVISIBLE
    }
}
```

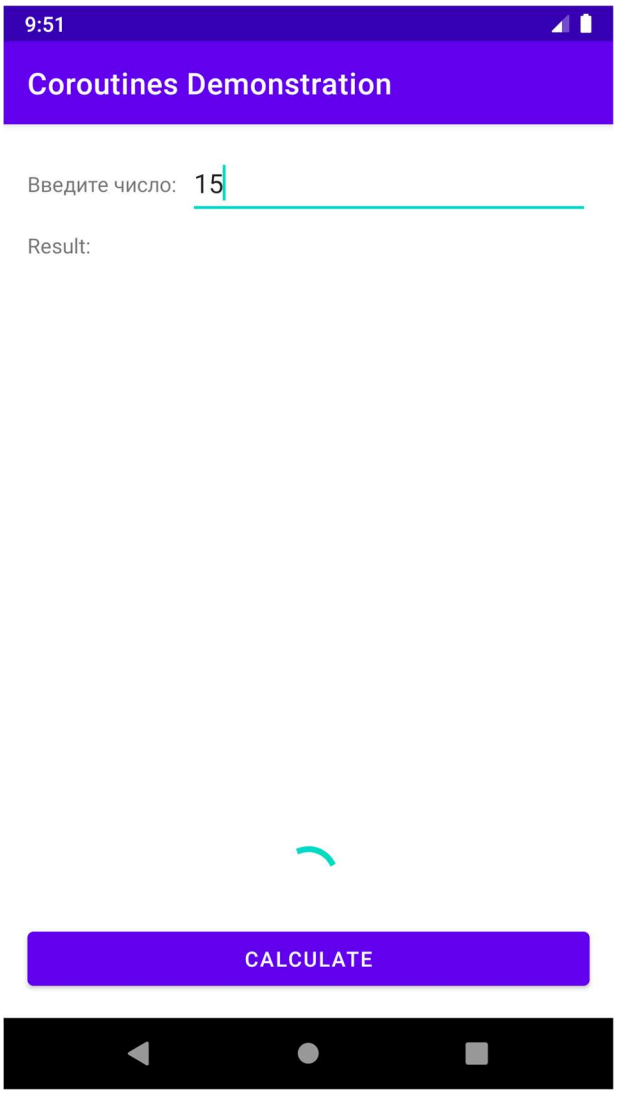
Функция **calculateFactorial()** тоже приостанавливаемая и запускает в себе асинхронно вычисление факториала.

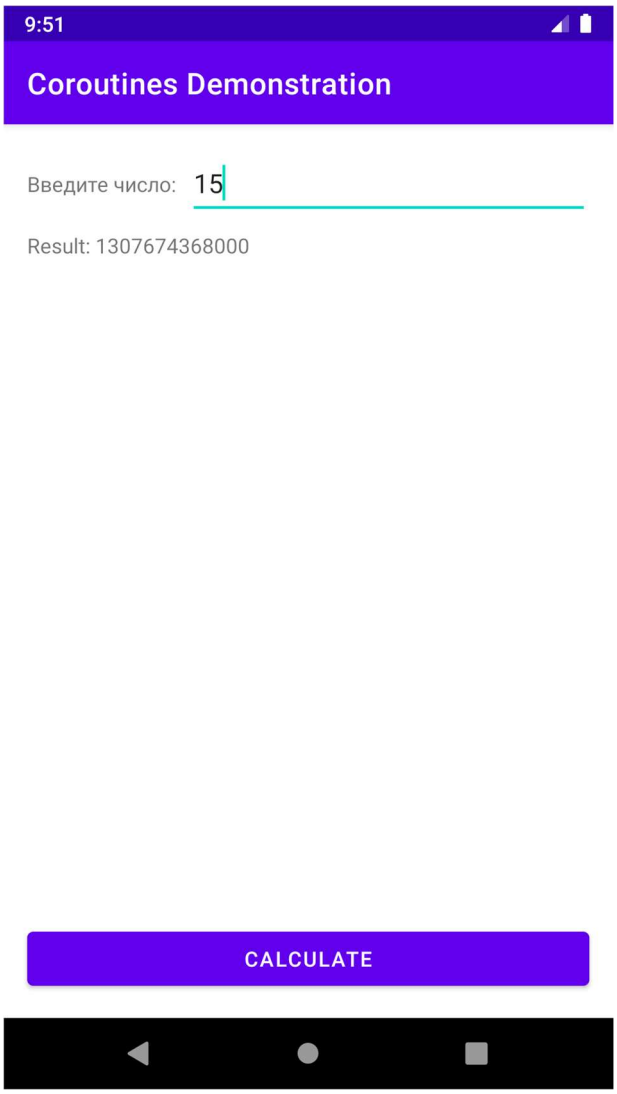
Переменная **result** здесь возвращает т.н. обещание того, что вернёт объект типа **Long**:

```
var result = GlobalScope.async {
    val numberEditText = findViewById<EditText>(R.id.editTextNumber)
    factorial(numberEditText.text.toString().toInt())
}
```

А метод **launch** показывает результат в нужный **TextView**

В конце концов вызывается метод **result.join()**, который запускает вычисления и приостонавливат корутину до получения результата. Но так как корутина выполняется отдельно от главного потока нашего приложения, она не блокирует его и приложение продолжает свою работу, пока в фоне выполняются вычисления


9:51
Coroutines Demonstration
Введите число: 15
Result:
CALCULATE


9:51
Coroutines Demonstration
Введите число: 15
Result: 1307674368000
CALCULATE