链滴

Kotlin 协程入门

作者：RustFisher

原文链接：https://ld246.com/article/1632492657279

来源网站：链滴

开发环境

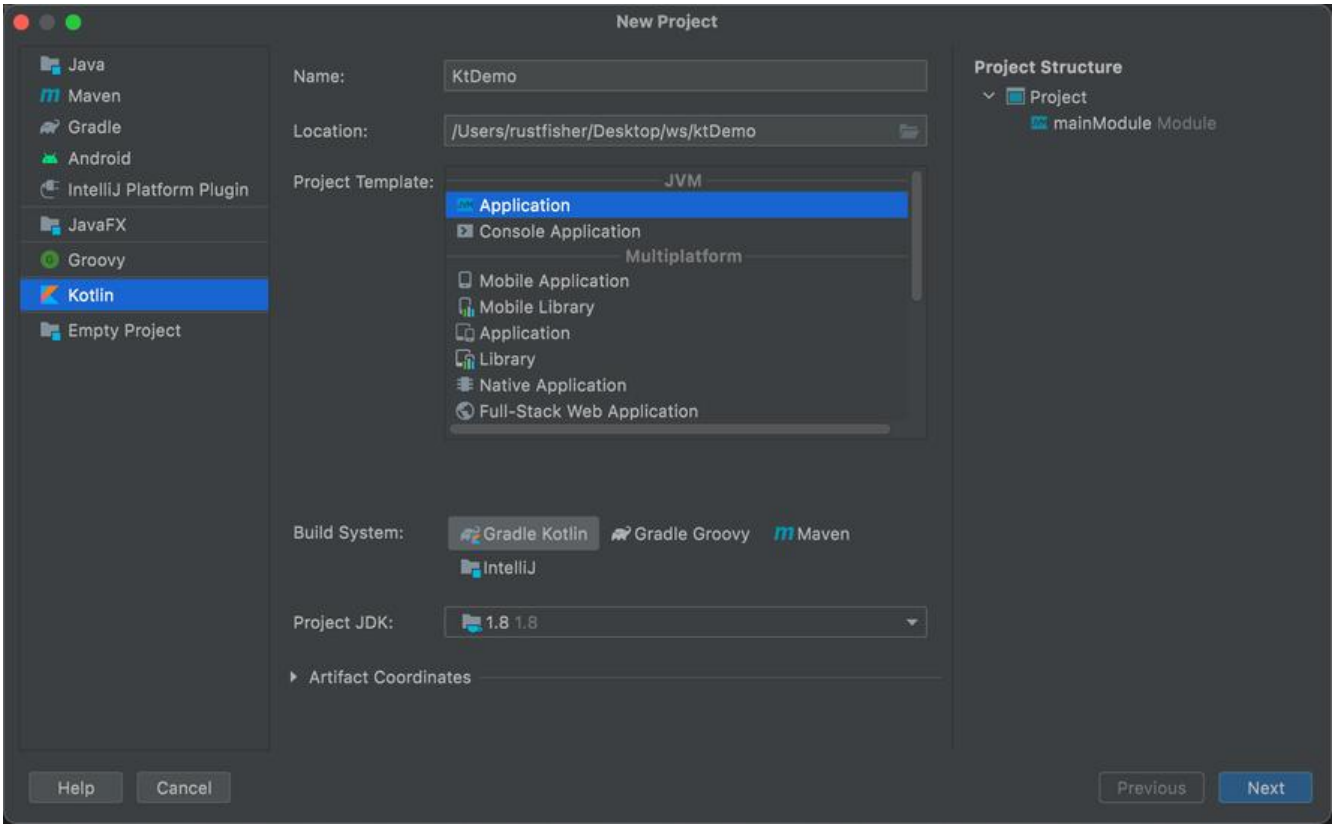
- IntelliJ IDEA 2021.2.2 (Community Edition)
- Kotlin: 212-1.5.10-release-IJ5284.40

介绍Kotlin中的协程。用一个例子来展示协程的基本用法。

第一个例子

新建工程

我们使用的是社区版IntelliJ IDEA 2021.2.2。新建一个Kotlin工程用来测试。



引入协程

项目用gradle进行管理，我们在Github上找到协程的依赖。

```
dependencies {
    implementation("org.jetbrains.kotlinx:kotlinx-coroutines-core:1.5.2")
}
```

修改项目的gradle配置。把依赖添加进去。

代码示例

创建一个类然后在main方法中写协程的相关代码。

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

fun main() {
    GlobalScope.launch { // 在后台启动一个新的协程并继续
        delay(300)      // 等待300毫秒
        "rustfisher.com".forEach {
            print(it)
            delay(200)   // 每次打印都等待一下
        }
    }
    println("RustFisher")
    Thread.sleep(3000) // 阻塞主线程防止过快退出
}
```

代码运行结果

本质上，协程是轻量级的线程。

我们用**GlobalScope**启动了一个新的协程，这意味着新协程的生命周期只受整个应用程序的生命周期制。

可以将 GlobalScope.launch { ..... } 替换为 thread { ..... }，并将 delay(.....) 替换为 Thread.sleep(...) 达到同样目的。注意导入包kotlin.concurrent.thread

协程换成线程

```
import java.lang.Thread.sleep
    import kotlin.concurrent.thread

    fun main() {
        thread {
            sleep(300)
            "rustfisher.com".forEach {
                print(it)
                sleep(200)
            }
        }
        println("RustFisher")
        sleep(3000) // 阻塞主线程防止过快退出
    }
```

如果thread{}中含有delay，编译器会报错

Suspend function 'delay' should be called only from a coroutine or another suspend function

因为delay是一个特殊的挂起函数，它不会造成线程阻塞，但是会挂起协程，并且只能在协程中使用。

至此，小结一下第一个协程示例

- gradle引入协程　kotlinx-coroutines-core
- GlobalScope.launch启动协程
- 协程中的　挂起函数delay(long)可以达到延时的效果，并且它只能在协程中使用

协程所在线程

本质上协程是轻量级的线程。我们观察一下协程所处的线程信息。

```
import kotlinx.coroutines.GlobalScope
    import kotlinx.coroutines.launch
    import java.lang.Thread.sleep

    fun main() {
        println("main线程信息 ${Thread.currentThread().id}")
        for (i in 1..3) { // 多启动几次协程
            GlobalScope.launch {
                println("协程启动#$i 所在线程id: ${Thread.currentThread().id}")
            }
        }
        sleep(2000) // 阻塞主线程防止过快退出
        println("RustFisher 示例结束")
    }
```

运行log如下

```
main线程信息 1
  协程启动#1 所在线程id: 11
  协程启动#3 所在线程id: 14
  协程启动#2 所在线程id: 13
  RustFisher 示例结束
```

可以看到多次启动协程，这些协程它们不一定在同一个线程中。也就是说有在**同一个线程**的可能性。

于是试试疯狂地启动协程，把循环次数加大for (i in 1..3000)。观察log可以看出，有重复的线程id。