【Treeman】Kotlin 相關

- 【Kotilin】延伸閱讀
- 未完成程式碼
  - 【反射】取得目前方法的名稱

- tip
  - null 判斷 ?:
  - 【Kotlin】集合Iterable/Collection/List/Set/Map

- 【Kotilin】基本型別
- 【Kotlin】coroutines
- 【MockK】Kotlin Mocking Library

【Kotilin】延伸閱讀

- Kotlin 官方文檔 簡體中文版

  https://book.kotlincn.net/

- Kotlin 官方教學範例(英文)

  https://play.kotlinlang.org/byExample/overview

- kotlin 風格指南

  https://book.kotlincn.net/text/coding-conventions.html

未完成程式碼

未完成程式碼

【反射】取得目前方法的名稱

```
/*
* 取得呼叫此方法的 methodName
*/
fun getMethodName(clazz: Any) {
        var fullName = ""


            val stackTrace = Thread.currentThread().stackTrace
            if (stackTrace.size >= 3) {
                val methodName = stackTrace[2].methodName
                if (methodName.isEmpty() || className.isEmpty()) {
                    return redisEnable
                }
                fullName = "${className}.${methodName}"
            }


    }
```

tip

tip

null 判斷 ?:

`(判斷式) ?: 0` 是 Kotlin 中的 Elvis 運算符。Elvis 運算符用於簡化空值檢查，它的語法形式是 `?:`。

具體地說，**`(判斷式) ?: 0`** 表示如果判定式的值不為 null，則使用判定式的值；如果判定式的值為 null，則使用後面的默認值 0。

下面是一個例子：

```
val result: Int? = someNullableValue // 假設 someNullableValue 是一個可能為 null 的變量

val finalResult = result ?: 0

println("最終結果是 $finalResult")
```

在這個例子中，如果 `someNullableValue` 不為 null，則 `finalResult` 將被賦值為 `someNullableValue` 的值；如果 `someNullableValue` 為 null，則 `finalResult` 將被賦值為默認值 0。

這種寫法可以簡化空值檢查，使代碼更加簡潔。Elvis 運算符的一般形式是 `expression ?: defaultValue`，其中 `expression` 是要檢查的表達式，而 `defaultValue` 是在表達式為 null 時使用的默認值。

tip

【Kotlin】集合 Iterable/Collection/List/Set/Map

出處：https://blog.csdn.net/vitaviva/article/details/107587134

下標操作類

- contains —— 判斷是否有指定元素
- elementAt —— 傳回對應的元素，越界會拋IndexOutOfBoundsException
- firstOrNull —— 傳回符合條件的第一個元素，沒有回傳null
- lastOrNull —— 傳回符合條件的最後一個元素，沒有回傳null
- indexOf —— 傳回指定元素的下標，沒有回傳-1
- singleOrNull —— 傳回符合條件的單一元素，如有沒有符合或超過一個，傳回null

判斷類

- any —— 判斷集合中是否有滿足條件的元素
- all —— 判斷集合中的元素是否都符合條件
- none —— 判斷集合中是否都不符合條件，是則回傳true
- count —— 查詢集合中符合條件的元素個數
- reduce —— 從第一項到最後一項累計

過濾類

- filter —— 過濾掉所有滿足條件的元素
- filterNot —— 過濾所有不符合條件的元素
- filterNotNull —— 過濾NULL
- take —— 傳回前n 個元素

轉換類別

- map —— 轉換成另一個集合（與上面我們實現的convert 方法作用一樣）；
- mapIndexed —— 除了轉換成另一個集合，還可以拿到Index(下標);
- mapNotNull —— 執行轉換前過濾掉為NULL 的元素
- flatMap —— 自訂邏輯合併兩個集合；
- groupBy —— 依照某個條件分組，返回Map；

排序類

- reversed —— 反序
- sorted —— 升序
- sortedBy —— 自訂排序
- SortedDescending —— 降序

建立集合

不可變更集合 immutable

```
import java.util.*
import java.text.SimpleDateFormat
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    val list = listOf(1,2,3, null)
    // 不為 null List
    val notNullList = listOfNotNull(null,1,2,3,null)
    val emptyList = emptyList<Int>()
    val map = mapOf("foo" to "FOO", "bar" to "BAR", "bar" to "BB")
    val set = setOf(4,5,6,6)
    println(list)    //  [1, 2, 3, null]
    println(notNullList)  // [1, 2, 3]
    println(emptyList)  //  []
    println(map)   //  {foo=FOO, bar=BB}
    println(set)   //  [4, 5, 6]
}
```

可變更集合 mutable

```
import java.util.*
import java.text.SimpleDateFormat
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
  val list = mutableListOf(1,2,3,3)
  val map = mutableMapOf("foo" to "FOO", "bar" to "BAR", "bar" to "BB")
  val set = mutableSetOf(1,2,3,3)
  println(list) // [1, 2, 3, 3]
  println(map) // {foo=FOO, bar=BB}
  println(set) // [1, 2, 3]
}
```

addAll(), list 相加

```
import java.util.*
import java.text.SimpleDateFormat

fun main() {
 val list1 = mutableListOf(1,2,3,4)
    val list2 = mutableListOf(5,6,7,8)
   println(list1.addAll(list2)) // true (注意當下有返回值)
    println(list1)     // [1, 2, 3, 4, 5, 6, 7, 8

}
```

remove()，list 刪除

```
import java.util.*
import java.text.SimpleDateFormat

fun main() {

    val numberList = mutableListOf(1,2,3)
```

```
    numberList.remove(2)

    println(numberList)  // [1, 3]
}
```

索引

```
import java.util.*
import java.text.SimpleDateFormat

fun main() {
 val list = listOf(1,2,3)
    val indices: IntRange = list.indices
    println(indices)

    for(i in list.indices){
        println(list[i])
    }
    /*
     0..2
    1
    2
    3
    */
    println(list.first())  // 1
    println(list.last()) // 3
    println(list.lastIndex) // 2 = ( size - 1 )
    println(list.size)  // 3
}
```

Stream 與 kotlin 比較

Stream.allMatch()	可迭代.all()、Map.all()
Stream.anyMatch()	Iterable.any()、Map.any()
Stream.count()	Iterable.count()、Map.count()
Stream.distinct()	可迭代.distinct()
Stream.filter()	可迭代.filter(), Map.filter()
Stream.findFirst()	Iterable.first(), Iterable.firstOrNull()
Stream.flatMap()	Iterable.flatMap(), Map.flatMap()
Stream.forEach()	Iterable.forEach()、Map.forEach()
Stream.limit()	iterable.take()
Stream.map()	Iterable.map(), Map.map()
Stream.max()	Iterable.max()、Map.maxBy()
Stream.min()	Iterable.min()、Map.minBy()
Stream.noneMatch()	可迭代.none(), Map.none()
Stream.peek()	–
Stream.reduce()	可迭代.reduce()
Stream.sorted()	可迭代.sorted()
Stream.skip()	–
Stream.collect(toList())	Iterable.toList()、Map.toList()
Stream.collect(toMap())	可迭代.toMap()
Stream.collect(toSet())	可迭代.toSet()
Stream.collect(加入())	Iterable.joinToString()
Stream.collect(partitioningBy())	可迭代.partition()
Stream.collect(groupingBy())	可迭代.groupBy()
Stream.collect(減少())	可迭代.fold()
IntStream.sum()	可迭代.sum()
IntStream.average()	可迭代.average()

List 相關

list.map{}

```
import java.util.*
import java.text.SimpleDateFormat

fun main() {
 val nameList = listOf("Sam","John",null,"Mary")
    val resultList = nameList.map{ "name:" + it }
    println(resultList)  // [name:Sam, name:John, name:null, name:Mary]
```

```
    // 過濾null
    val resultList2 = nameList.mapNotNull{ it }
    println(resultList2)  // [Sam, John, Mary]
}
```

list 保留或排除(removeAll(), retainAll())

```
import java.util.*
import java.text.SimpleDateFormat

fun main() {
 val peopleList = mutableListOf(People("sam",18),People("John",23),People("Mary",30))
    // 移除 age > 25
    peopleList.removeAll{ it.age > 25 } // [People(name=sam, age=18), People(name=John, age=23)]
    println(peopleList)

    // 保留 name == sam
    val peopleList2 = mutableListOf(People("sam",18),People("John",23),People("Mary",30))
    peopleList2.retainAll{ it.name == "sam" } // [People(name=sam, age=18)]
    println(peopleList2)
}
```

list 轉 map

```
import java.util.*
import java.text.SimpleDateFormat

fun main() {
 val peopleList = listOf(People("Sam",12),People("Mary",18), People("Sam",13))
    // 指定 key : value
    val map = peopleList.associate { it.name to it }
    // 指定 key , value => 固定為物件
    val map2 = peopleList.associateBy { "name:" + it.name }
    println(map)  // {Sam=People(name=Sam, age=13), Mary=People(name=Mary, age=18)}
    println(map2) // {name:Sam=People(name=Sam, age=13), name:Mary=People(name=Mary, age=18)}

}

data class People(
 val name:String,
    val age: Int
)
```

isEmpty()

```
import java.util.*
import java.text.SimpleDateFormat

fun main() {
 val list1 = listOf(1,2,3,4)
    val list2 = emptyList<Int>()
    var list3:List<Int>? = null

    println(list1.isEmpty())  // flase
    println(list2.isEmpty())  // true
    println(list3?.isEmpty())  // null

    println(list1.orEmpty())  // [1, 2, 3, 4]
    println(list2.orEmpty())  // []
```

```
    println(list3?.orEmpty())  // null
}
```

take(), takeLast() 取出list 數量

```
import java.util.*
import java.text.SimpleDateFormat

fun main() {

   val numberList = mutableListOf(1,2,3,4,5,6)
   // 從前面取兩個
   println(numberList.take(2))  // [1, 2]
   // 從後面取兩個
   println(numberList.takeLast(2)) // [5, 6]

   val noneList :List<Int>? = emptyList()
   println(noneList?.take(1))  // []

    val nullList :List<Int>? = null
   println(nullList?.take(1))  // null
}
```

getOrElse() ,getOrNull()

```
import java.util.*
import java.text.SimpleDateFormat

fun main() {
 val list = listOf(1,2,3,4)

   println(list.getOrElse(1,{9})) // 1
    println(list.getOrElse(5,{9})) // 9

    println(list.getOrNull(1)) // 2
    println(list.getOrNull(5)) // null
}
```

Map 相關

containsKey()，包含 key

```
import java.util.*
import java.text.SimpleDateFormat

fun main() {

    val numberMap = mapOf("one" to 1,"two" to 2)

    println(numberMap.containsKey("one")) // true

    val worldMap = mapOf("one" to "ONE","two" to "TWO")
    println(worldMap.containsValue("one")) // false
    println(worldMap.containsValue("ONE")) // true
}
```

filter{}, filterNot{}, 過濾

filterKeys{}, filterValues{}

```
import java.util.*
import java.text.SimpleDateFormat

fun main() {

    val worldMap = mapOf("one" to "ONE","two" to "TWO", "three" to "THREE")

    println(worldMap.filter {it.key.contains("t")})   // {two=TWO, three=THREE}
    println(worldMap.filterNot {it.key.contains("t")})  // {one=ONE}

    println(worldMap.filterKeys {key -> key.contains("o")})   // {one=ONE, two=TWO}
    println(worldMap.filterValues {value -> value.contains("T")}) // {two=TWO, three=THREE}

}
```

forEach{}

```
import java.util.*
import java.text.SimpleDateFormat

fun main() {

    val map = mapOf("one" to "ONE","two" to "TWO", "three" to "THREE")

    for( (key, value) in map ){
        println("key:${key}, value:${value}")
    }
    // key=one, value=ONE
    // key=two, value=TWO
    // packagekey=three, value=THREE

    map.forEach{
        println("key:${it.key},value:${it.value}")
    }
    // key=one, value=ONE
    // key=two, value=TWO
    // packagekey=three, value=THREE
}
```

getOrElse()

isEmpty(), isNotEmpty()

```
import java.util.*
import java.text.SimpleDateFormat

fun main() {

    val map = mapOf("one" to "ONE","two" to "TWO", "three" to "THREE")

   println(map.getOrElse("one",{"Default"}))  // ONE
    println(map.getOrElse("four",{"Default"}))  // Default
```

```
    val map2 = mapOf<String,String>()
    val map3 :MutableMap<String,String>? = null

    println(map.isEmpty())   // false
    println(map.isNotEmpty())  // true
    println(map.orEmpty())   // {one=ONE, two=TWO, three=THREE}

    println(map2.isEmpty())   // true
    println(map2.isNotEmpty())  // false
    println(map2.orEmpty())   // {}

    println(map3?.isEmpty())  // null
    println(map3?.isNotEmpty())  // null
    println(map3?.orEmpty())  // null

}
```

count(), 數量, 有條件的數量

```
import java.util.*
import java.text.SimpleDateFormat

fun main() {

    val worldMap = mapOf("one" to "ONE","two" to "TWO", "three" to "THREE")
    println(worldMap.count()) // 3
    println(worldMap.count{ it.key.contains("t")}) // 2

}
```

【Kotilin】基本型別

基本型別

- 數字
- Boolean
- 字元
- 字串
- 陣列 Array

【Kotlin】coroutines

相關資源

官方文件:coroutines-guide

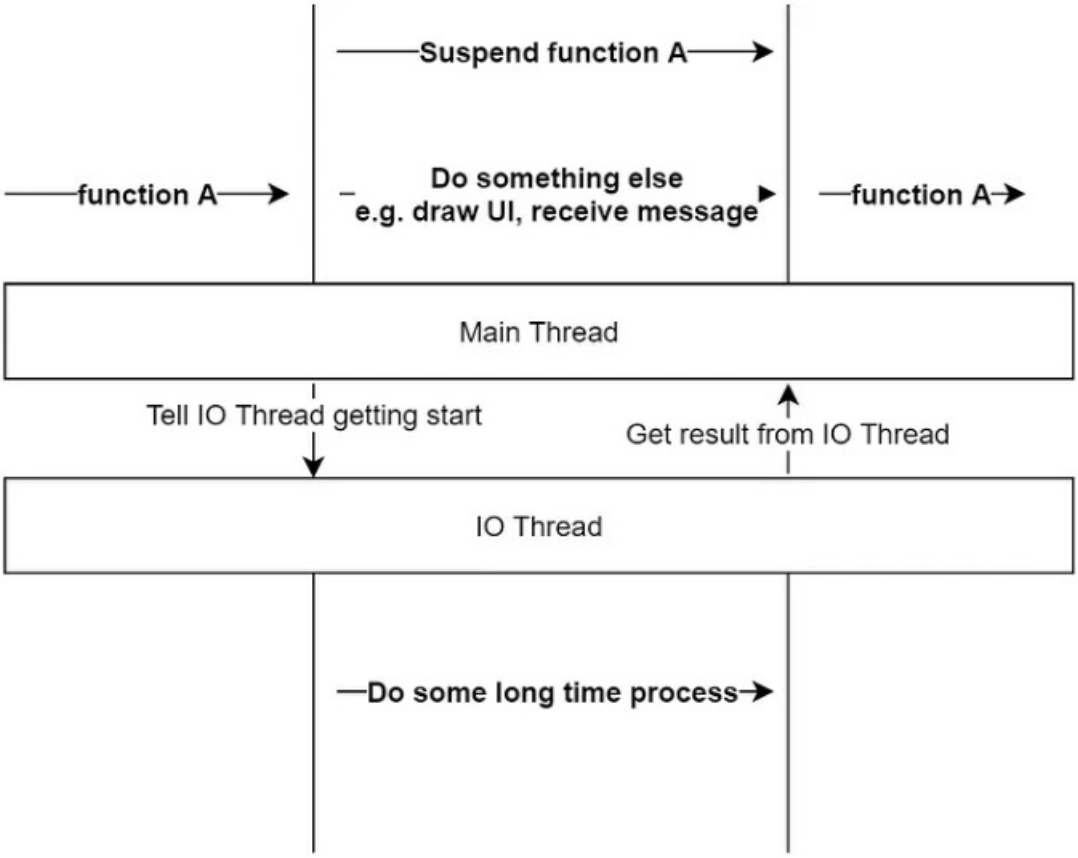
一. 什麼是coroutines?（協同程序）

（一）意義：

- coroutines的中文翻譯為「**協程**」，「深入淺出kotlin」一書中形容協程就像「輕量的執行緒」(light-weight threads)。使用協同程序的意義就在於，啟動一個背景工作時，同時讓其他程式不需要等待工作完成就可以做別的事情，用戶體驗會比較好
- Coroutines ，分別是 cooperation + routine， cooperation 意指合作，routine 意指例行作業、慣例，照這裡直接翻譯就會是合作式例行作業。
- 協程是一種輕量級的線程，可以被掛起和恢復。它允許你以同步的方式編寫非同步程式碼，從而避免回調地獄。

（二）優點：
1. 解決傳統執行緒的困擾：
影片及書本中都以「在main thread連接internet」為例，由於android明令禁止應用程式在主執行緒存取網路，因此，此時主執行緒會被阻塞。而傳統的執行序在主執行緒阻塞未被移除前，是不會再執行其他作業的，coroutines也是執行緒，只不過在這種情形會被暫停，但是不會阻塞父執行緒。

2. 很適合執行背景工作：
如（一）所述，使用協同程序的好處就在於，啟動一個背景工作時，同時讓其他程式不需要等待工作完成就可以做別的事情，用戶體驗會比較好。



就是我在 main thread 執行到 function A 需要等 IO thread 耗時處理的結果，那我先暫停 function A， 協調讓出 main thread 讓 main thread 去執行其他的事情，等到 IO thread 的耗時處理結束後得到結果後再回復 function A 繼續執行

二.基本概念

掛起函數（Suspending Function）

掛起函數是可以在不阻塞執行緒的情況下掛起的函數。它使用suspend關鍵字定義。例如：

```
suspend fun doSomething() {
    // Your code here
}
```

協程作用域（Coroutine Scope）

協程作用域管理協程的生命週期。協程只能在某個作用域內啟動。

```
runBlocking {
    launch {
        // Coroutine code
    }
}
```

啟動協程

Kotlin 提供了幾種方式來啟動協程，常用的方法有`launch`和`async`。

- launch

```
runBlocking {
    launch {
        delay(1000L)
        println("World!")
    }
    println("Hello,")
}

// Hello
// World!
```

- async
- 與`launch`類似，但它會傳回一個`Deferred`對象，可以在將來取得結果。

```
runBlocking {
    val deferred = async {
        delay(1000L)
        "World!"
    }
    println("Hello,")
    println(deferred.await())
}

// Hello
// World!
```

協程上下文與調度器

協程上下文包含協程的各種元素，例如調度器（Dispatcher）、作業（Job）等。

- **Dispatchers.Main**：在主執行緒上運行，用於更新UI。
- **Dispatchers.IO**：用於執行IO 操作，如網路請求、檔案讀寫。
- **Dispatchers.Default**：用於執行CPU 密集型任務。
- **Dispatchers.Unconfined**：啟動協程時繼承目前線程，只有在協程掛起後才會切換到其他線程。

```
runBlocking {
    launch(Dispatchers.Main) {
        // 在主□程运行
    }
    launch(Dispatchers.IO) {
        // 在IO□程运行
    }
    launch(Dispatchers.Default) {
        // 在默□□程池运行
    }
}
```

```
fun main() = runBlocking { // this: CoroutineScope
    launch { // launch a new coroutine and continue
        delay(1000L) // non-blocking delay for 1 second (default time unit is ms)
        println("World!") // print after delay
    }
```

```
    println("Hello") // main coroutine continues while a previous one is delayed
}

// Hello
// World!
```

【MockK】Kotlin Mocking Library

相關連結

- **官方網站**
  https://mockk.io/#chinese-guides-and-articles
- MockK：一款強大的 Kotlin Mocking Library (Part 1 / 4)
  https://medium.com/joe-tsai/mockk-%E4%B8%80%E6%AC%BE%E5%BC%B7%E5%A4%A7%E7%9A%84-kotlin-mocking-library-part-1-4-39a85e42b8