前言

看工程師寫著密密麻麻近似亂碼的程式碼，總會給外人一種迷幻的錯覺，好像在寫什麼密碼或是下什麼咒語般的神祕。但巷子裡的行家都知道，在開發者的每日工作裡，其實有很大一塊都是在做「資料處理」。從各式各樣的目標來源抓取、截取、爬取資料，接著整理、過濾、轉換其格式，最後輸出成有意義、有系統的資訊。

江湖一點訣，說穿了一點也不神祕。

筆者早期因為要處理動態網站而踏入後端及資料庫等領域，一開始選擇從動態型別的直譯語言（Interpreted Language）入手，那時語言的原始型別（Primitive Type）並不多、標準函式庫也以函式（Function）為主。而在處理資料時，大多習慣性使用 Array 操作，雖然沒有大問題，但總是被函式的參數順序、易讀性及可維護性所困擾。隨著開發經驗的累積，慢慢導入框架工具，開始學習如何使用集合（Collection）物件來操作資料，這才有了資料流串接（Pipeline）的觀念及技巧，開啟筆者使用集合來處理資料的視野。

後來因工作需要而接觸 Kotlin 程式語言，其強型別（Strong Typing）的設計搭配聰明的型別推斷（Type Inference），加上簡潔的語法及豐富的標準函式庫，許多過往經驗都能在 Kotlin 裡看到更優美的實作而深深著迷。隨著筆者陸續舉辦 Kotlin 讀書會、Kotlin 練功場等活動，在這段時間也有幾次分享 Kotlin Script、Kotlin DSL 的經驗，在準備講題的過程中體會到集合是各種進階知識的基石，若能好好運用，操作資料時會更有效率，也能減少很多重複、冗長的工作。

為了持續精進自己對 Kotlin 集合的瞭解，2020 年末以此為主題參加第 12 屆 iT 邦幫忙鐵人賽，有幸得到評審青睞並獲出版機會，這本書即以鐵人賽的內容為基礎重新設計，在結構、敘述脈絡及範例都有大幅度的改寫，希望能幫助更多 Kotlin 開發者成為其深入集合應用的敲門磚。

❑ 本書架構

本書架構共三大部份：技法、心法及實戰。

第一部份「技法篇」會以語法介紹為核心，從集合四大物件 Array、List、Set 及 Map 出發，說明其基本使用方式。接著以食譜書（Cookbook）的方式帶著讀者綜覽 Kotlin 集合方法（Collection Method），筆者會將這些方法依不同目的、特性做分類，從建立（Creation）、取值（Retrieving）、排序（Ordering）、檢查（Checking）、操作（Operation）、分群（Grouping）、轉換（Transformation）、聚合（Aggregation）、轉型（Conversion）等操作逐一以不同章節詳細說明，每一個方法也都有對應的範例程式碼，讀者可以實際看到各方法的使用範例。最後還會提供讀者一張集合方法速查地圖，透過這張心智地圖，方便讀者查詢與記憶這為數眾多的集合方法。

有了語法基礎後，第二部份的「心法篇」將著重於集合的底層實作。筆者會在一問一答的思辯之間，以閱讀標準函式庫原始碼的方式，帶著大家深入了解其實作奧祕，包括泛型（Generic）、Lambda、Extension Function、Inline Function、Infix Function 等，以及各種語法糖（Syntax Sugar）。除了介紹集合常用語法背後的原理外，也會跟讀者討論集合方法在命名時對於時態、詞性的使用邏輯與慣例。最後以一些常與集合併用的組合技，包括 Range、Progression、Sequence 及 Scope Function 等做結尾，期能讓讀者對集合能有更深入的認識。

只有理論是不夠的，唯有搭配實戰才能將知識落實在日常任務裡。第三部份的「實戰篇」會融合筆者的開發經驗匯集成一系列情境題，以類似刷題解題的方式，帶著讀者一起探索如何綜合運用集合功能來面對各種資料處理情境，活用從技法及心法篇學到的知識，並從過程中思考如何用集合來提升程式碼表達力並讓專案更好維護。

❑ 這本書適合誰？

本書適合所有對 Kotlin 集合有興趣的開發者，即便您對開發 Kotlin 程式、使用 IDE 及 SDK 不熟悉也不用擔心，本書都有對應的章節可以補齊這些知識。雖然本書範例是以運行在 JVM 平台而設計，但由於語法都沒有超出 Kotlin 標準函式庫的範圍，因此也不限定 Kotlin 開發者的類型，舉凡用 Kotlin 寫後端、行動應用甚至前端、原生開發皆適用。希望透過本書的內容能讓您更聰明地處理資料、寫出更好懂、更好維護的程式，輕鬆掌握 Kotlin 集合的賞玩門道。

❑ 本書編寫慣例

- 翻譯用語

  在解釋技術原理及範例內容時，會提到許多英文技術詞彙。針對不同詞彙本書做以下處理：

  - 本書核心 - 集合四大物件 Array、List、Set 及 Map 統一不翻保持原文。
  - 若該詞彙的中文翻譯已有普遍共識，則在文章裡第一次出現時於中文翻譯後以括號標註英文原文，之後以中文字詞為主。
  - 若該詞彙沒有統一的翻譯或翻譯後難辨識其原意，則在文章裡統一保留英文原文。
  - 本書另附有中英詞彙對照表於附錄，供讀者參考。

- 提示框

  若要補充段落裡提到的進階內容，或是提示故障排除的方法時，筆者會另以提示框的方式標記註解。若您想要了解更多進階技巧，或是跟著書中範例練習時發生問題，可以參考提示框裡的資訊。

- 範例程式碼

  書中範例程式碼會以 JetBrains Mono[1] 等寬字型及灰底區塊呈現，部份章節為節省篇幅，範例內的程式碼會依照敘述脈絡精簡呈現。若需取得本書完整範例程式碼，可至本書官網範例下載頁 https://collection.kotlin.tips/sample 取得。

❑ 勘誤

技術更新日新月異，即便筆者在撰寫過程已盡力校對，並成立技術審校小組協助找出書中可能的錯誤，但仍無法保證做到完美。若您在閱讀本書的過程中發現任何錯誤，可直接以 Email <shengyoufan@gmail.com> 與筆者聯絡，我會將勘誤公佈在本書官網讓讀者取得修正後的正確內容。

❑ 致謝

本書得以完成需要感謝許多人。首先要感謝 iT 邦幫忙鐵人賽主辦單位每年舉辦鐵人賽，除了每年都能獲得歷練自己的機會外，還能有幸出版書籍。同時要感謝博碩文化團隊的 Abby 不辭辛勞照顧我這種新手作者，並體諒過程中的寫作低潮。也感謝我的編輯小 P 在出版過程中的耐心，以及對各細節的把關和協助，讓本書的品質能符合彼此的期待。再來要感謝我的好朋友們 － Google DevRel 上官林傑、Kotlin GDE 黃健旻、Taiwan Kotlin User Group 主辦人趙家笙百忙之中為我撰文推薦。

我想特別感謝一路以來力挺我的技術審校小組 － Andy、Maggie 及 Tina，長達數月的審校工作就像跑馬拉松，除了要把本書讀好幾遍外，還要驗證所有範例（他 / 她們是最熟本書內容的讀者了 :p ），工作量可想而之。感謝三位不辭辛勞的付出，讓這本書能呈現出最好的樣子。

當然還要感謝我在 JetBrains 的同事，包括我的直屬老闆 Hadi Hariri、Kotlin 傳教士團隊 Svetlana Isakova、Sebastian Aigner、Anton Arhipov、Ekaterina Petrova、Pasha Finkelshteyn 以及 Package Search 團隊 Sebastiano Poggi、Jakub Senohrábek、Lamberto Basti，除了給我很大的空間嘗試外，也在範例上提供我靈感。

最後當然要感謝整個 Kotlin 社群的朋友們，大夥從舉辦讀書會、練功場、Meetup 及參與各式演講、研討會、鐵人賽建立起深厚的革命情感，推廣一個程式語言需要大家的支持與幫忙，謝謝您們！

CHAPTER

01

技法篇

寫程式就像廚藝一樣，在煮出一道美味的佳餚之前，必須先認識各種食材、對營養學等基礎知識有一定程度的了解，打好基礎才能更上層樓。

本書的第一部份「技法篇」會先以 Kotlin 集合的語法為核心，介紹其四大物件的基本使用方式，接著建立練習用的範例專案，並透過九個章節把集合可以使用的方法，地毯式掃描一遍。目標不是要大家把所有方法全背起來，而是讓讀者「知道」集合有哪些方法可以用，以便在實作遇到類似的情境時能「想起來」有這些招數可以使。

為了達成這個目的，這些章節會設計的像食譜書一樣，透過**目的**及**特性**做兩層分類，逐一展示每一個集合方法的核心用法。最後，筆者會將這些方法整理成一張速查地圖並標註方法對應的章節，在閱讀後續心法及實戰章節時，若覺得對該方法不夠熟悉，可再跳回對應的章節複習，多來回幾次可加深印象、強化記憶。

1-1 集合四大物件

想像一下自己在整理房間的情境，通常在整理東西的時候，我們會拿有格子的容器，把同類型的東西一格一格的放好，方便我們儲存、排列或抽換。把這樣的概念對比到寫程式也是類似的，以一個活動報名系統為例，裡面儲存的就是參與者（姓名、電話、Email）的報名資料，這種資料類型就是我們定義的型別（Type），而把這些相同資料類型的物件儲存在同一個容器裡，就是程式語言裡所謂的集合（Collection）。當我們用程式來整理資料時，也會用整理房間同樣的策略，把相同類型的物件放在同個容器裡的不同儲存格裡，接著就可以依照業務需求來操作這些資料，包括排序、統計、過濾、分群、搜尋…等。

Kotlin 團隊在語言設計之初，就將 Kotlin 語言裡的一切設計成物件，並把集合相關的程式碼都放在 **`kotlin.collections`** 的套件（Package）中。集合也不只是拿來儲存資料，還可以透過其屬性及方法做很多額外的處理來滿足任務需求。針對不同的使用情境與目標，Kotlin 共有四種不同的集合類別供我們使用：Array、List、Set、Map。

在這個章節裡，我們將綜覽集合四大物件的基礎語法。

1-1-1 Array

我們從 Array 開始，它是 Kotlin 集合裡最簡單的結構，其概念及用法在各程式語言裡也幾乎是相通的。Array 是一個用來裝資料的有序結構，我們在使用前要先宣告 Array 的尺寸（或稱大小、長度）以及放入元素的型別，Array 會為這個容器裡的每一格標上編號，這個編號稱為索引（Index），每一格裡放入的物件我們稱為元素（Element 或 Item）。由於電腦底層設計的原因，索引一律從 0 開始計算，每增加一格，索引就往上加 1，以此類推。

Array 本身是一個固定尺寸（靜態）的容器，一經宣告後就無法再改變它的大小。想像一下電腦的記憶體裡有十個格子可以裝資料，當我們在程式裡宣告一個可以放五個整數（Int）的 Array 時，電腦就會從記憶體裡把其中五格劃分為這個 Array 的專屬空間。這時我們就沒有辦法從 Array 裡新增或刪除內容，因為電腦已經在記憶體裡把這五格固定綁在一起，但我們可以從這五格裡取出其中的內容或是更換放在裡面的元素。換句話說，Array 的尺寸是不可變（Immutable）但內容是可變（Mutable）的。

❑ 建立 Array

要宣告一個 Array 很簡單，Kotlin 標準函式庫提供 **`arrayOf()`** 函式，直接宣告放入的型別以及傳入想要儲存在 Array 裡的元素就可以建立對應型別、尺寸的 Array。

```
val numbers = arrayOf<Int>(1, 2, 3, 4, 5)
```

這段程式碼可拆解成幾個部份的資訊：

- 建立名為 **`numbers`** 的 Array。
- Array 裡元素的型別是 Int。
- 把 1、2、3、4、5 放進 Array 裡。
- Array 尺寸是五格。

也就是說，我們建立了一個裝著五個整數型別的 Array，裡面的內容分別是 1、2、3、4、5。

輸入完這段程式碼後，您會發現 IntelliJ IDEA 把型別宣告的部份標記成灰色，意思是我們可以省略型別宣告。這是為什麼呢？因為 Kotlin 編譯器會直接從我們放進 Array 的元素來做型別推斷，假如我們在 **`numbers`** 按下 ⌥+↩（macOS）或 Alt+Enter（Windows/Linux）並選擇「**Specifiy type explicitly**」，IntelliJ IDEA 就會依據 Kotlin 編譯器的資訊將其宣告為 **`Array<Int>`**。

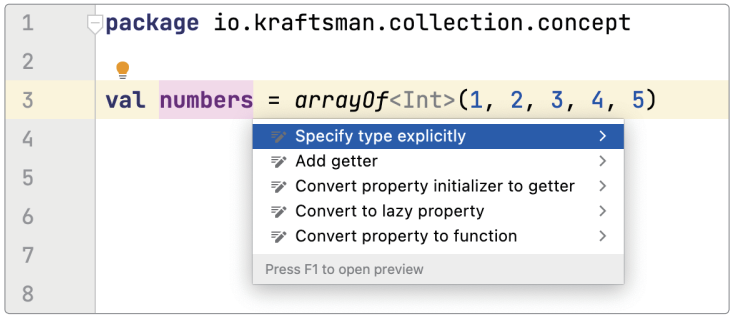

圖 1-1-1　使用 IntelliJ IDEA 自動產生型別宣告

換句話說，當我們用 **`arrayOf()`** 函式建立 Array 時，若有宣告放入的元素，就可以省略型別宣告，靠編譯器型別推斷的能力來省下一些程式碼。若我們在建立 Array 時，想直接指定其尺寸並填入預設內容的話，可以改用物件實例化的語法，比方說我們要宣告一個內含五個 **`a`** 字元的 Array，可以這樣寫：

```
val arrayOfFiveA = Array(5) { 'a' }
```

雖然在宣告時有傳入元素就可以依賴型別推斷，但假如 Array 是空的，就沒有可以推斷型別的依據，這時型別宣告就變成必要的。所以要宣告空的 Array 時，有以下幾種寫法：

```
val emptyArrayUsingArrayOf = arrayOf<Int>()
// 型別是 Array<Int>

val emptyArrayUsingEmptyArray = emptyArray<Int>()
// 型別是 Array<Int>

val emptyArrayOfNull = arrayOfNulls<Int?> (5)
// 型別是 Array<Int?>
```

一般來說，Array 只會放入相同型別的元素，若原本 Array 宣告是放 **`Int`**，但您試圖把其中的元素換成 **`String`** 的話，IntelliJ IDEA 馬上就會出現紅色波浪

1-1-5 四大物件綜合比較

一次看完 Kotlin 集合的四大物件後，是不是對它們之間相似又有點不同的特性覺得有點混淆呢？在這邊筆者快速重點回顧這四個物件的特色：

❑ Array

- 無法變更尺寸，一經宣告容量即固定的靜態容器。
- 只能存放相同型別的元素，可以變更內容。
- 以索引排序。
- 內容元素可以重複。

❑ List

- 有可變及不可變兩種。
  - **`List`** 內容不可變，一經宣告變無法更新內容。
  - **`MutableList`** 內容可變，可以使用如 **`add()`**、**`remove()`** 等方法修改 List 裡的內容。
- 以索引排序。
- 內容元素可以重複。

❑ Set

- 有可變及不可變兩種。
  - **`Set`** 內容不可變，一經宣告變無法更新內容。
  - **`MutableSet`** 內容可變，可以使用如 **`add()`**、**`remove()`** 等法修改 Set 裡的內容。
- 元素間沒有順序。
- 以雜湊值識別元素唯一性，內容不可重複。

❑ Map

- 有可變及不可變兩種。
  - **`Map`** 內容不可變，一經宣告變無法更新內容。
  - **`MutableMap`** 內容可變，可以使用如 **`put()`**、**`remove()`** 等法修改 Set 裡的內容。
- 適用於儲存 Key/Value 配對型資料。
- Key 就是索引，元素間沒有順序。
- Key 具唯一性，但值則允許重複。
- Key 必須是相同型別、Value 也得是相同型別，但 Key 與 Value 可分屬不同型別。

把三個重點特性整理成表格後，相信更有助於記憶和背誦：

物件	是否有序？	是否唯一？	儲存內容
Array	是	否	元素
List	是	否	元素
Set	否	是	元素
Map	否	Key 是、Value 否	Key/Value 配對

實務上在選擇物件時的心法，大多會先從不可變的 List 開始使用，即便需要修改排序、轉換、計算…等，也大多透過眾多的集合方法完成，真的有需要修改原始 List 內容時，還可以透過 **`toMutableList()`** 做轉型。除非一開始就確定會修改 List 內容，不然使用 List 即可滿足大部份的需求。

若需要限制元素不能重複，就使用 Set；若是對照表（Lookup Table）的結構，就使用 Map；若想要較輕量、原始型別的集合，就用 Array。抓緊每個物件最重要的特性，掌握好決策模型，就能在正確時機使用適合的物件喔！

1-4-13 Chunk 取區段方法

若需要把集合切成一塊一塊固定長度的小段，不必手動用迴圈動刀，標準函式庫提供的 **chunked()** 就可以用於這種情境。只要傳入一個指定區塊大小的正整數參數 **size**，**chunked()** 會依照參數值將集合切成數段 List 後，再以 **List<List<T>>** 的格式回傳，若集合尺寸無法被 **size** 參數整除，最後一個區塊的尺寸會不足 **size** 的大小。**chunked()** 的行為可用下圖解釋：

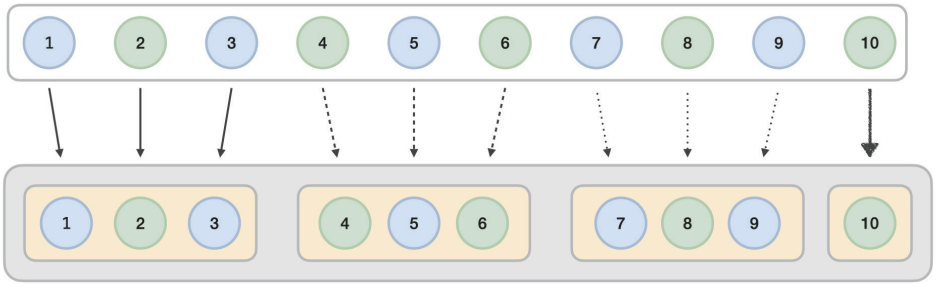

圖 1-4-4 圖解 checked() 方法行為

以一個有十個元素的集合為例，若每三個元素切一段，則可以切成三段完整、一段只有一個元素共四段子集合：

▶ 檔名：.../collection/technique/retrieving/parts/chunked.ws.kts

```
val numbers = listOf(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

numbers.chunked(3)
// [[1, 2, 3], [4, 5, 6], [7, 8, 9], [10]]
```

chunked() 支援以傳入的 λ 參數做二次處理，比方說把回傳的每一個子集合裡的數字加總，或將子集合裡的字串串接成一個大字串…等。一般做過二次處理後，最終回傳的型別就會被 λ 裡的的行為轉型。

▶ 檔名：.../collection/technique/retrieving/parts/chunked.ws.kts

```
val numbers = listOf(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
val chars = listOf('a', 'b', 'c', 'd', 'e', 'f', 'g')

numbers.chunked(3) { it.sum() }
// [6, 15, 24, 10]
// 型別為 List<Int>

chars.chunked(2) {
    it.joinToString(
        separator = "+",
        prefix = "(",
        postfix = ")"
    )
}
// [(a+b), (c+d), (e+f), (g)]
// 型別為 List<String>
```

`chunked()` 方法並不會更動原始集合裡的內容，因此方法使用過去分詞命名。

1-4-14 Window 取區段方法

另一種將集合分段的方式是先設定一段範圍，然後以這個範圍逐「格」移動來取出元素。想像一下前方有五個物件，而手上拿著一次只能看到三個物件的「窗戶」，從第一個物件的位置逐格移動，移動時把每一格可以看到的範圍收成一個子集合，這種取值方法就稱做 **`windowed()`**。**`windowed()`** 的行為可用下圖解釋：

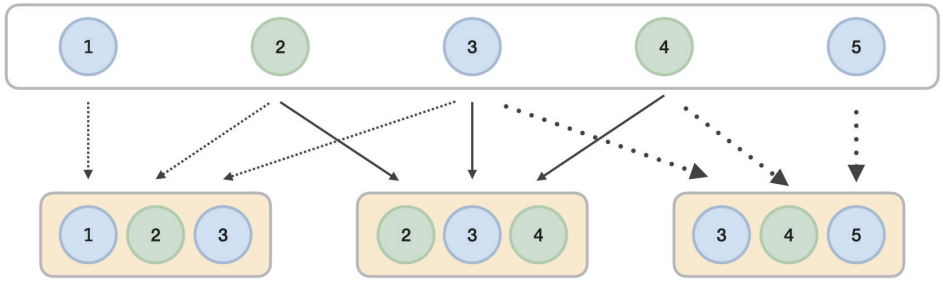

圖 1-4-5 圖解 windowed() 方法行為

以一個有五個元素的集合為例，若用一個三格窗戶移動取值，則從頭到尾移動完可以拿到三個子集合。要注意的是，**`windowed()`** 預設每次移動一格，所以回傳的子集合的第一個元素會是父集合裡的每一個元素，父集合裡的元素也會重複地出現在子集合裡：

▶ 檔名：.../collection/technique/retrieving/parts/windowed.ws.kts

```
val numbers = listOf(1, 2, 3, 4, 5)

numbers.windowed(3)
/*
  [
    [1, 2, 3],
    [2, 3, 4],
    [3, 4, 5]
  ]
*/
```

若這種行為模式不符需求，**`windowed()`** 也提供四個參數可做彈性調整：

1. 第一個參數 **`size`** 是窗戶的尺寸。
2. 第二個參數 **`step`** 每次移動的格數。
3. 第三個參數 **`partialWindows`** 則是布林值。在取值時，若想保留不足一段的子集合則傳 True，反之則傳 False

4. 第四個參數 **`transform`** 則是要做二次操作時，可傳入 λ 參數做轉換。

▶ 檔名：.../collection/technique/retrieving/parts/windowed.ws.kts

```
numbers.windowed(
    size = 3,
    step = 2,
    partialWindows = true
)
/*
   [
    [1, 2, 3],
    [3, 4, 5],
    [5]
   ]
*/

numbers.windowed(
    size = 3,
    step = 2,
    partialWindows = false
)
/*
   [
    [1, 2, 3],
    [3, 4, 5]
   ]
*/

numbers.windowed(size = 3,
    step = 2,
    partialWindows = false
) { it.sum() }
// [6, 12]
```

1-7-1 總覽圖

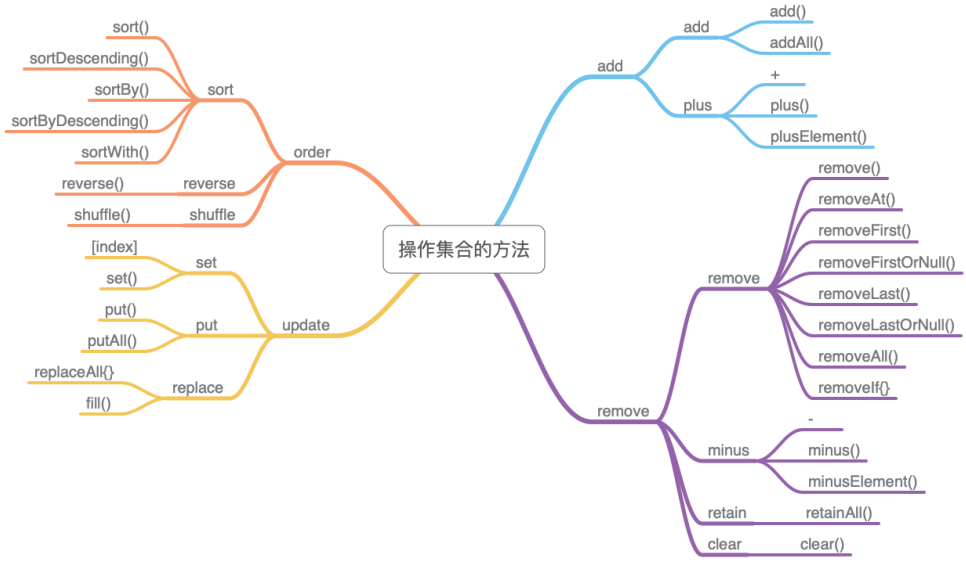

圖 1-7-1　操作集合的方法總覽圖

在「操作」這個用途底下，我們以四個關鍵字做分類，分別是：**`add`**、**`remove`**、**`update`** 及 **`order`**，下面將分別針對各方法做介紹。

1-7-2 Add 系列方法

在現有集合裡新增元素，可用 **`add()`** 方法把要新增進集合的元素當成參數傳入即可，方法會回傳 True/False 來表示新增成功與否。在撰寫程式碼時，IntelliJ IDEA 也會依型別推斷來檢查傳入的元素是否能放入集合裡，若傳入元素的型別不一致，則 IDE 會以紅色波浪線提示。

▶ 檔名：.../collection/technique/operation/add.ws.kts

```
val numbers = mutableListOf(1, 3, 5, 7, 9)

numbers.add(8)
// numbers 的內容為 [1, 3, 5, 7, 9, 8]
```

預設 **add()** 會將傳入的元素放到集合的最後一個位置，若是想要指定元素放置的位置，則可在呼叫 **add()** 時傳入兩個參數，第一個參數是索引位置、第二個參數是要放入集合的元素。

▶ 檔名：.../collection/technique/operation/add.ws.kts

```
numbers.add(2, 4)
// numbers 的內容為 [1, 3, 4, 5, 7, 9, 8]
```

在設定索引位置時，要注意不能超出集合的索引範圍，不然執行時會拋出 **IndexOutOfBoundsException** 例外。若需一次在集合裡增加多個元素，可以用 **addAll()**。使用時把要傳入的元素放在一個集合裡再整包傳進去，預設也是依序將這些元素接在原本集合元素的最後面。若需指定元素的插入位置，可以在第一個參數指定。

▶ 檔名：.../collection/technique/operation/addAll.ws.kts

```
val numbers = mutableListOf(1, 3, 5, 7, 9)

numbers.addAll(listOf(4, 6, 8))
// numbers 的內容為 [1, 3, 5, 7, 9, 4, 6, 8]

numbers.addAll(2, listOf(4, 6, 8))
// numbers 的內容為 [1, 3, 4, 6, 8, 5, 7, 9, 4, 6, 8]
```

有別於 **add()** 和 **addAll()** 會將元素寫入集合，標準函式庫另外提供以 **plus** 開頭的方法讓開發者可以將一個或多個元素加進集合內，與 **add()** 不同的

2-2-1 時態慣例

觀察各集合方法名稱裡所使用的動詞，會發現有些方法雖使用相同動詞但時態不同。比方說在 **1-7 操作集合的方法**裡，有 **`sort()`**、**`sortDescending()`**、**`sortBy()`**、**`sortByDescending()`**、**`sortWith()`**、**`reverse()`** 及 **`shuffle()`** 這些跟排序有關的方法，而在 **1-5 排序集合的方法**裡，也有 **`sorted()`**、**`sortedDescending()`**、**`sortedBy()`**、**`sortedByDescending()`**、**`sortedWith()`**、**`reversed()`** 及 **`shuffled()`** 這些跟排序相關的方法。筆者將這兩類方法並列整理至下方表格：

操作集合的方法	排序集合的方法	功能說明
sort()	sorted()	依自然排序正向排序
sortDescending()	sortedDescending()	依自然排序反向排序
sortBy()	sortedBy()	以傳入的 λ 正向排序
sortByDescending()	sortedByDescending()	以傳入的 λ 反向排序
sortWith()	sortedWith()	以傳入的 Comparator 排序
reverse()	reversed()	反轉集合內元素的排列順序
shuffle()	shuffled()	隨機排列集合裡元素的順序

這兩類方法雖然在名稱裡使用相同的動詞，但操作集合的方法名稱使用原形動詞，而排序集合的方法名稱則是用過去分詞（加了 ed）。兩者間的根本差別，在於方法名稱含**原形動詞**的方法會**直接更動原始集合**裡的元素，回傳 Unit（意義上等同不回傳）；而方法名稱含**過去分詞**的方法不會更動原始集合，而是**把更動的結果放在全新的集合後回傳**。也因為作用的集合型別不同，所以**只有可變集合能呼叫以原形動詞命名的集合操作方法**，而**排序集合的方法則不論可變集合或不可變集合皆可使用**。

CHAPTER

03

實戰篇

在看完前面的技法與心法後，相信大家已經對 Kotlin 集合的四大物件、各類型的方法及實作原理有更深入的了解。前面的章節為了讓讀者容易吸收，會儘量簡化範例的複雜度，單純就方法的功能做示範。但當我們面對真實的開發需求時，不免會有些落差。換言之，實戰還是有其必要性，本書的第三部份「實戰篇」就是在補齊這塊拼圖。

接下來，筆者會從過往的開發經驗整理出一系列情境題，以類似刷題解題的方式，帶領讀者綜合運用集合的功能來面對各種資料處理情境，活用從心法與技法學到的知識。透過這些練習逐步掌握實戰技巧，真正將集合應用在工作上。同時也會在案例間穿插介紹延伸應用，包括使用 Kotlin Playground 分享程式碼、在網頁上內嵌程式碼區塊、結合 Faker 產生假資料、使用 Mordant 設定終端機輸出樣式，以及用 Ktor 實作 Mock Server，為讀者示範 Kotlin 廣泛的應用面向。

3-1 樂透選號

3-1-1 做一個發財夢

這天早上，筆者的新加坡同事傳了訊息過來：

早安！今天新加坡樂透的頭獎金額高達 $8,600,000（約新台幣 1.8 億）耶！快給我 1 組 6 個 1 到 49 之間的不重複數字，我要去買個機會，假如明天我沒來上班的話，你就知道發生什麼事了 :p

咦？有沒有覺得這樣的對話很熟悉呢？相信大家一定有過類似的經驗，尤其是年節期間，總是會冒出寫抽獎機、樂透機的需求。身為開發者雖然沒辦法預測或報明牌，不過要產生六個隨機數字到很容易。而且有前面心法與技法的熏陶，相信讀者心中已經冒出數個集合方法準備串接了對吧？

分析同事的需求，拆解出來的虛擬碼（Pseudocode）大概是這樣的：

1. 產生從 **1** 到 **49** 的數字後，放入箱子裡。
2. 把箱子搖一搖（將元素隨機排序）。
3. 從箱子中取出不重複的六個數字。
4. 把數字排序（為了方便同事填單）。
5. 用「,」連接成字串（為了方便閱讀）。
6. 把最終結果輸出。

在下手寫程式碼前，先把虛擬碼寫下來，對解題會很有幫助。它能夠幫助我們先專注在拆解邏輯與操作步驟，等覺得解題策略和流程是正確的後，再把虛擬碼以 Kotlin 程式碼實作即可。接下來，筆者就照著前面的虛擬碼以學習到的技巧實作一次。

3-2 資料統計運算

3-2-1 微型資料科學

Kotlin User Group（或簡寫為 KUG）是分佈在全世界，專門討論、分享 Kotlin 知識的技術社群。任何對 Kotlin 技術有興趣的愛好者（不需要是開發者，任何人都可以！），都可以為自己所在的城市向 Kotlin 官方申請成立，並登記在 Kotlin 官網的 User Group 清單頁[1]。目前在台灣除了有 Taiwan Kotlin User Group 外，在 Facebook 上還有 Kotlin Taipei 社團，從 2021 年開始，兩個社群聯手舉辦 Kotlin Meetup，視疫情狀況舉辦實體或線上的技術分享聚會，讓 Kotlin 開發者能持續接收新知、彼此交流心得，是很重要的「取暖」活動。

筆者有幸從活動舉辦初期就擔任志工至今，為了讓主辦群能了解聽眾的參與心得與建議，每次活動舉辦時，都會準備線上報名及意見回饋表單，透過活動後的數據統計與分析，讓工作小組檢視活動舉辦過程需要改善之處，讓 Kotlin Meetup 能愈辦愈好。Kotlin Meetup 目前使用 Google Form 製作表單，在表單關閉後可從後台下載 CSV 格式的資料，接著就可以用 Kotlin 分析這些資料，邁出「微型資料科學（其實就是統計啦）」的第一步。

本章以活動報名資料為例，串連多個集合方法做統計運算。請先在練習專案的 **`practice`** 資料夾底下新增 **`statistics`** 資料夾，做為本章練習的根目錄。在資料夾內建立名為 **`processing.ws.kts`** 的 Kotlin WorkSheet 檔案，在此檔案內撰寫統計運算程式碼。要讀取的資料集 CSV 檔案放在 **`dataset`** 資料夾內，不過由於活動報名資料含敏感個人資訊，活動結束後工作小組也會依管理辦法處置個資，因此本章裡的運算皆為假想範例，使用的來源檔案是仿照活動報名表單欄位，以假資料產生出的模擬資料。讀者可先打開 **`dataset`** 資料夾裡，以

除了以上集合方法外，本章還介紹如何在專案中增加套件，並以 Mordant 套件設定終端機輸出樣式，讓畫面輸出更好看；以及示範如何依需求撰寫產生器以產生練習用的資料集，讓開發時可以自行模擬情境。透過以上這些集合方法的綜合運用，一些基本的統計運算可以用幾行程式碼就完成，不需依賴辦公室軟體、不需記憶軟體介面操作，也完全不用背誦公式，全部以高語意的 Kotlin 程式碼達成！

3-3 萬用 Mock Server

3-3-1 當個龍的傳人

筆者近年在推廣 Kotlin 時，曾與朋友 Nevin 合作一場《Kotlin 一條龍 - 打造全平台應用》[1] 的線上技術分享，由 Nevin 以 Kotlin Multiplatform Mobile 技術開發手機應用程式，而筆者以 Ktor 框架配合前端手機應用程式實作後端 API 服務。兩人合力在講座裡，以 Kotlin 程式語言，打造橫跨前後端的多平台應用。

當初在設計這場講座的內容時，就是想展示 Kotlin 多平台的能力，除了可以開發 Android 外，也可以用於 iOS、後端、前端，甚至應用在資料科學，真正做到圈內暗黑笑話所謂「一條龍」生產的境界，紮紮實實的當個龍的傳人。

如同所有前後端分離的專案，在準備這場講座時，兩人是分別準備各自負責的部份。所以在開發初期，優先針對 API 規格進行討論。為了快速驗證兩人的想法，前端的部份只做了簡單的 UI 及換頁，就開始串接後端 API 服務，實作處理 HTTP Request 與 Response 的功能。而後端為配合前端的開發進度，先用 Ktor 框架快速實作出一個 Mock Server，讓後端在沒有資料庫的情況下，就可以模擬回傳符合 API 規格的內容。