前言

當我在 2010 年春季首次拜訪 JetBrains 時，我相信世界上不需要另一個通用的程式語言。我認為現有的 JVM 語言已經夠好了，而且理性的人怎麼還會去建立一種新的語言呢？在大約一個小時的討論大規模程式碼庫中的產品之後，我被說服了，後來成為 Kotlin 部分的第一個想法，是在白板上勾勒出的。我很快就加入了 JetBrains，並負責領導語言的設計和編譯器的工作。

超過六年後的今天，我們已經發表了第二個版本。團隊超過了 30 人，有數千名活躍的使用者，我們仍然擁有比我更容易處理更多令人興奮的設計想法。但不要擔心，這些想法在進入語言之前，必須透過相當徹底的測試。我們希望未來的 Kotlin 仍然適合編成一本大小合理的書。

學習程式語言是一項令人興奮，且往往是非常有價值的工作。如果這是你的第一個語言，那麼你就透過它學習程式的全新世界。如果不是，它會讓你用新的術語思考熟悉的事物，從而更深入地理解它們，並在更高的抽象層次上理解它們。本書主要針對後一類讀者，那些已經熟悉 Java 的讀者。

從頭開始設計一門語言本身可能是一項具有挑戰性的任務，但使其與另一種語言良好地搭配是另一回事，其中包含許多憤怒的食人魔，以及一些陰沉的地下城（如果你不相信，可以問問 Bjarne Stroustrup，他是 C++ 的建立者）。Java 互用性（也就是 Java 和 Kotlin 如何混合互相呼叫）是 Kotlin 的基石之一，本書非常重視它。互用性對於逐漸將 Kotlin 引入現有 Java 程式碼非常重要。即使從 scratch 編寫一個新專案，也必須將該語言與該平台結合起來，並將其所有函式庫用 Java 編寫。

在寫這篇文章的時候，我們正在開發兩個新的目標平台：Kotlin 現在執行在 JavaScript 虛擬機，以實現全端 Web 開發，以及它將很快能夠直接編譯為原生程式碼，無需執行任何虛擬機。因此，雖然本書是以 JVM 為導向的，但從裡面學到的大部分內容，都可以應用於其他執行環境。

作者從早期就一直是 Kotlin 團隊的成員，所以他們非常熟悉此語言及其內部。他們在有關 Kotlin 的會議報告、研討會和課程方面的經驗，使他們能夠提供很好的解釋，來預測常見的問題和可能的陷阱。本書解釋了語言特性背後的高級概念，並提供了所有必要的細節。

我希望你會喜歡我們的語言和本書。正如我經常在我們的社群文章中所說：**祝你有個美好的 *Kotlin* 體驗！**

ANDREY BRESLAV
在 JETBRAINS 的 KOTLIN 首席設計師

序

Kotlin 的想法是在 2010 年 JetBrains 中構思的。那時，JetBrains 是許多語言（包括 Java、C#、JavaScript、Python、Ruby 和 PHP）的開發工具的建立供應商。作為我們旗艦產品的 Java IDE IntelliJ IDEA 還包含 Groovy 和 Scala 插件。

為這種多樣化的語言建置工具的經驗，使我們對語言設計空間整體有了獨特的理解和視角。而基於 IntelliJ 平台的 IDE，包括 IntelliJ IDEA，仍在使用 Java 開發。我們對在 .NET 團隊中使用 C# 開發的同事感到有些嫉妒，C# 是一種現代、功能強大且快速發展的語言。但是我們沒有看到任何可以用來代替 Java 的語言。

我們對這種語言的要求是什麼？第一個也是最明顯的是靜態型態。我們不知道有什麼其他的方法可以在多年的時間裡，開發出一種數百萬行的程式碼庫，而且不會發瘋。其次，我們需要完全相容現有的 Java 程式碼。該程式碼庫對於 JetBrains 來說是非常寶貴的資產，我們不能因為互用性的困難而失去它或貶低它。第三，我們不希望在工具品質方面接受任何妥協。JetBrains 作為一家公司來說，開發人員的生產力是最重要的價值，且出色的工具對於實現這一目標至關重要。最後，我們需要一種易於學習和推理的語言。

當看到我們公司尚未滿足的需求時，我們知道還有其他公司處於類似的情況，我們希望我們的解決方案，能在 JetBrains 公司之外找到很多使用者。考慮到這一點，於是決定開始建立一種新語言的專案：Kotlin。碰巧這個專案花費的時間比我們預期的要長，Kotlin 1.0 在第一次提交到版本函式庫後五年多了。但現在我們可以肯定，這個語言已經找到了他的群眾，並留在這裡。

Kotlin 以俄羅斯聖彼得堡附近的一座島嶼命名的，Kotlin 開發團隊大部分位於這座島。在使用島名時，我們遵循了由 Java 和 Ceylon 建立的先例，但我們決定尋找離家更近的東西。（英文名稱通常發音為「cot-lin」，而不是「coat-lin」或「caught-lin」。）

隨著語言接近發表，我們意識到有一本關於 Kotlin 的書的價值，這本書由參與制定語言設計決策的人編寫，並且可以自信地解釋為什麼 Kotlin 中的事物就是這樣。本書是這項努力的結果，我們希望它能幫助你學習和理解 Kotlin 語言。祝你好運，願你永遠快樂地開發程式！

關於本書

《*Kotlin* **實戰手冊**》向你介紹 Kotlin 程式語言，以及如何使用它來建構在 Java 虛擬機和 Android 上執行的應用程式。它從語言的基本特徵開始，繼續涵蓋 Kotlin 更獨特的方面，如支援建置高級抽象和特定域語言。本書非常重視將 Kotlin 與現有的 Java 專案結合在一起，並幫助你將 Kotlin 引入當前的工作環境。

這本書涵蓋了 Kotlin 1.0。Kotlin 1.1 與本書的撰寫同步進行，並且在可能的情況下，我們已經提到了 1.1 中所做的更改。但是由於在撰寫本文時新版本仍在進行中，因此我們無法提供完整的敘述。有關新功能和修正的持續更新，請參閱 https://kotlinlang.org 上的線上文件。

誰應該讀這本書

本書主要是聚焦在具有一定 Java 經驗的開發人員。Kotlin 以 Java 的許多概念和技術為基礎，本書致力於透過你現有的知識幫助你快速掌握。如果你只是在學習 Java，或者如果你對其他程式語言（如 C# 或 JavaScript）有經驗，則可能需要引用其他資訊來源，以了解 Kotlin 與 JVM 交互的更複雜的方面，但是你仍然可以透過本書學習 Kotlin。我們關注的是整個 Kotlin 語言，而不是針對特定領域的問題，因此本書對於伺服器端開發人員、Android 開發人員，以及其他建構以 JVM 專案為目標的其他人同樣有用。

本書架構

本書分為兩部分。第 1 部分解釋如何開始將 Kotlin 與現有的函式庫和 API 結合使用：

- 第 1 章講述 Kotlin 的關鍵目標、價值和應用領域，並展示執行 Kotlin 程式碼的可能方法。

- 第 2 章解釋任何 Kotlin 程式的基本元素，包括控制結構、變數和函式宣告。
- 第 3 章詳細介紹如何在 Kotlin 中宣告函式，並介紹繼承函式和屬性的概念。
- 第 4 章關注類別宣告，並介紹資料類別和伴生物件的概念。
- 第 5 章介紹在 Kotlin 中使用 lambda，並使用 lambda 展示許多 Kotlin 標準函式庫函式。
- 第 6 章介紹 Kotlin 型態系統，特別聚焦在可空性和聚集的主題上。

第 2 部分教你如何在 Kotlin 中建置自己的 API 和抽象，並且涵蓋該語言的一些更深層的特性：

- 第 7 章討論了慣例原則，它給具有特定名稱的方法和屬性賦予了特殊的含意，並介紹委託屬性的概念。
- 第 8 章展示如何宣告更高階的函式—函式，它們接受其他函式和參數或回傳它們。它還介紹行內函式的概念。
- 第 9 章深入探討 Kotlin 中的泛型主題，從基本語法開始，進入更高級的領域，如實體型態參數和可變性。
- 第 10 章介紹註解和反映的使用，並以 JKid 為中心，JKid 是一個小型、真實的 JSON 序列化函式庫，它大量使用到這些概念。
- 第 11 章介紹特定域語言的概念，介紹 Kotlin 的建置工具，並提供許多 DSL 範例。

還有三個附錄。附錄 A 解釋了如何使用 Gradle、Maven 和 Ant 建置 Kotlin 程式碼。附錄 B 著重於編寫文件註解，並為 Kotlin 模組產生 API 文件。附錄 C 是探索 Kotlin 生態系統，並尋找最新線上資訊的指南。

完整看完本書的效果最好，但也可以閱覽你感興趣主題的特定章節，並在遇到陌生概念時遵循交叉引用。

1 Kotlin：它是什麼以及為什麼要學它

本章涵蓋：

- Kotlin 的基本介紹
- Kotlin 的主要特點
- 對 Android 和伺服器端開發的可能性
- Kotlin 與其他語言有什麼區別
- 編寫並執行 Kotlin 程式碼

什麼是 Kotlin ？這是一種針對 Java 平台的新程式語言。Kotlin 具有簡潔、安全、務實的特性，且專注於與 Java 程式碼的互用性（interoperability）。現今幾乎到處都可以使用 Java：對伺服器端的開發、Android 的應用程式（apps） 等等。Kotlin 可與所有現有的 Java 函式庫和框架一起使用，並且執行的效能與 Java 相同。在本章中，我們將詳細探討 Kotlin 的主要特徵。

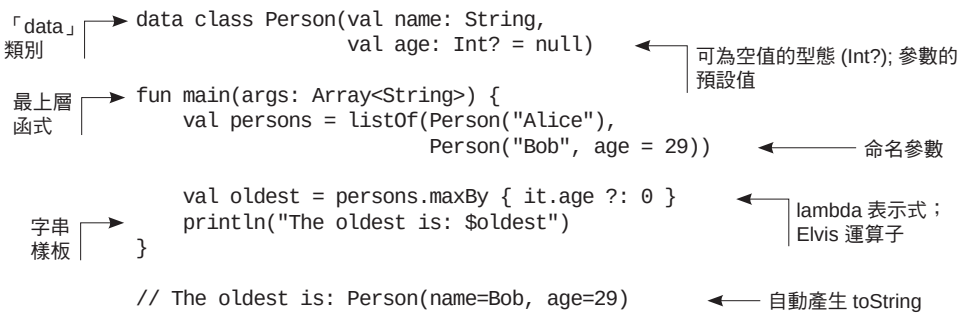
1.1 品嚐 Kotlin

我們先從一個小範例來展示 Kotlin 的面貌。此範例定義一個 `Person` 類別、建立一個人的集合（collection）、找到最年長的那一個人，並印出結果。即使在這一小段程式碼中，你也可以看到 Kotlin 的許多有趣的特性；我們已經明確指出了其中的一

些特性，所以稍後可以在書中輕鬆找到它們。程式碼會簡要地加以說明，若有什麼不清楚的話，請不要擔心，稍後還會詳細討論它。

如果想嘗試執行這個範例，最簡單的方式是使用 http://try.kotl.in 的線上 playground。輸入範例，接著點擊 Run 按鈕，便會執行程式碼。

範例程式 1.1　淺嚐 Kotlin




```
data class Person(val name: String,
                  val age: Int? = null)

fun main(args: Array<String>) {
    val persons = listOf(Person("Alice"),
                         Person("Bob", age = 29))

    val oldest = persons.maxBy { it.age ?: 0 }
    println("The oldest is: $oldest")
}

// The oldest is: Person(name=Bob, age=29)
```

宣告一個具有兩個屬性（name 和 age）的簡單資料類別。age 屬性預設值為 null（如果沒有指定）。在建立 people 列表時，省略了 Alice 的年齡，因此，將會使用預設值 null。接著使用 maxBy 函式，尋找列表中最年長的人。傳給函式的 lambda 表示式接受一個參數，並將 it 作為該參數的預設名稱。如果 age 為 null，*Elvis 運算子*（?:）將回傳 0。由於 Alice 的年齡沒有被規定，所以 Elvis 運算子將其替換為 0，最後 Bob 得到最年長的獎項。

你喜歡你所看到的嗎？請繼續閱讀以了解更多，並成為一位 Kotlin 的專家。我們希望不久的將來，你會在自己的專案中用到 Kotlin 的程式碼，不僅只有在本書中。

1.2　Kotlin 的主要特徵

你可能對 Kotlin 是什麼樣的語言，已經有了初步的想法。現在來更詳細地來檢視它的關鍵屬性。首先，看看可以用 Kotlin 建立什麼樣的應用程式。

1.2.1　目標平台：伺服器端、Android、Java 能執行的任何地方

Kotlin 的主要目標是提供一個更簡潔、更高效率、更安全的 Java 替代品，適用於當今使用 Java 的任何環境。Java 是一種非常流行的語言，從智慧卡（Java Card 技術）到由 Google、Twitter、LinkedIn，以及其他網路公司營運的資料中心都使用

Java。在這大多數的地方，使用 Kotlin 可以幫助開發人員用更少的程式碼和更少的煩惱來實現目標。

使用 Kotlin 最常見的領域是：

- 建構伺服器端的程式碼（通常是網頁的後台）
- 建構在 Android 上執行的 APP

但是 Kotlin 也可以在其他情況下使用。例如，可以使用 Intel Multi-OS Engine（https://software.intel.com/zh-cn/multi-os-engine）在 iOS 設備上執行 Kotlin 程式碼。要建構電腦的應用程式，可以將 Kotlin、TornadoFX（https://github.com/edvin/tornadofx）和 JavaFX 一起使用。[1]

除 Java 之外，Kotlin 可以被編譯為 JavaScript，允許在瀏覽器中執行 Kotlin 程式碼。但截至撰寫本文時，JavaScript 仍持續支援在 JetBrains 上進行探索和原型化，所以它不在本書的討論範圍之內，其他平台也正在考慮未來的 Kotlin 版本。

如你所見，Kotlin 的應用非常廣泛。Kotlin 不關注單個領域，也不涉及當今軟體開發人員面臨的單一型態的挑戰。相反地，它為整個開發過程，提供了全面的生產力改善。它也提供了支援特定領域或程式規範的函式庫完美整合。接下來看看 Kotlin 作為一種程式語言的關鍵特性。

1.2.2 靜態型態

就像 Java 一樣，Kotlin 是一個**靜態型態**（*statically typed*）的程式語言。這意味著程式中每個表示式的型態，在編譯時都是已知的，編譯器可以驗證你正在擷取的物件上存在的方法和欄位。

這與 Groovy 和 JRuby 在 JVM 上表示的**動態型態**（*dynamically typed*）程式語言形成了明顯的對比。這些語言允許你定義可以儲存或回傳任何型態資料的變數和函式，並在執行時解析引用的方法和欄位。此允許更短的程式碼和建立資料結構時有更大的靈活性。但缺點是編譯期間無法檢測拼寫錯誤等問題，並可能導致執行時的錯誤。

1　"JavaFX: Getting Started with JavaFX," Oracle, http://mng.bz/500y.

另一方面，與 Java 相比，Kotlin 不要求在程式碼中明確指定每個變數的型態。在許多情況下，變數的型態可以從上下文自動判定，從而可以省略型態宣告。下面是最簡單的例子：

```
val x = 1
```

宣告了一個變數，因為它是用一個整數值初始化，所以 Kotlin 自動判定它的型態是 `Int`。編譯器根據上下文確定型態的能力稱為**型態推論**（*type inference*）。

以下是靜態型態的一些好處：

- **效能**——呼叫方法更快，因為不需要在執行時找出需要呼叫的方法。
- **可靠性**——編譯器驗證程式的正確性，所以執行時崩潰的可能性較小。
- **可維護性**——使用不熟悉的程式碼更容易，因為可以看到程式碼與哪種物件一起執行。
- **工具支援**——靜態語賦予可靠的重構、精確的程式碼實作和其他 IDE 功能。

由於 Kotlin 支援型態推論，靜態語言大多數的不便消失了，因為不需要明確地宣告型態。

如果看看 Kotlin 型態系統的細節，會發現很多熟悉的概念。類別、介面和泛型的工作方式與 Java 非常相似。因此雖然有些知識是新的，但大部分的 Java 知識都能夠輕鬆轉移到 Kotlin。

其中最重要的是 Kotlin 對**可空型態**的支援，它允許你在編譯時透過檢測可能的 `null` 指標例外來編寫更可靠的程式。本章後面我們將回到可空的型態，並在第 6 章詳細討論它們。

Kotlin 型態系統中的另一個新功能是支援**函式型態**。為了看看這是什麼，讓我們來看看函式程式設計的主要思想，看看它在 Kotlin 中的支援。

1.2.3 函式程式設計與物件導向

作為一名 Java 開發人員，你毫無疑問熟悉物件導向的核心概念，但函式程式設計對你來說可能是新的。函式程式設計的關鍵概念如下：

- **一流的功能**——使用函式（行為）作為值。可以將它們儲存在變數中，將它們作為參數傳遞，或者從其他函式中回傳。

- **不變性**——使用不可變的物件，這保證了他們的狀態建立後不能改變。
- **沒有副作用**——可以使用純函式，在給定相同的輸入的情況下回傳相同的結果，不修改其他物件的狀態或與外界互動。

可以從函式程式設計中獲得什麼好處？首先是**簡潔**（*conciseness*）。函式程式設計與其命令式程式碼相比，可以更加優雅和簡潔，因為使用函式作為值，可以為你提供更多的萃取（abstraction）能力，從而避免程式碼的重複。

假設有兩個相似的程式碼片段來實現類似的工作（例如在一個集合中尋找配對的元素），但細節不同（如何檢測配對元素）。我們可以輕鬆地將邏輯的公用部分提取到函式中，並將不同的部分作為參數傳遞。這些參數本身就是函式，但是可以使用稱為 *lambda* **表示式**（*lambda expressions*）的匿名函式之簡潔語法來表示它們：

```
fun findAlice() = findPerson { it.name == "Alice" }
fun findBob() = findPerson { it.name == "Bob" }
```

findPerson() 包含尋找一個人的一般邏輯

大括號中的程式碼標示需要尋找的特定人物

函式程式設計的第二個好處是**安全的多執行緒**（*safe multithreading*）。多執行緒程式中最大的錯誤來源之一，就是修改多執行緒中的相同資料而沒有正確的同步。如果使用不可變資料結構和純函式，則可以確保不會發生這種不安全的修改，並且不需要提出複雜的同步方案。

最後，函式程式設計意味著**更容易測試**（*easier testing*）。沒有副作用的函式可以獨立地進行測試，而不需要大量的設置程式碼來建構他們所依賴的整個環境。

一般來說，函式程式設計可以用在任何程式語言，包括 Java，其中很多部分被提倡為良好的程式風格。但並不是所有的語言都提供輕鬆使用它所需的語法和函式庫；例如，這種支援在 Java 8 之前的版本中大部分都是缺少的。Kotlin 具有豐富的功能，可以從一開始就支援函式程式設計。這些包括以下內容：

- **函式型態**（*function type*），允許函式接收其他函式作為參數或回傳其他函式
- *lambda* **表示式**（*lambda expression*），讓你用最少的樣板傳遞程式碼區塊
- **資料類別**（*data class*），為建立不可變值的物件提供簡潔的語法
- 標準函式庫中豐富的 *API*，用於處理函式程式設計中的物件和集合

Kotlin 讓你的程式是函式程式設計，不需要強制它。當你需要它的時候，可以使用可變資料，編寫有副作用的函式而不用跳過任何額外的限制。當然，使用基於介面和類別層次結構的框架與 Java 一樣簡單。在 Kotlin 中編寫程式碼時，可以將物件導向和函式程式設計方式結合起來，並使用最適合解決問題的工具。

1.2.4 免費及開源

包括編譯器、函式庫和所有相關工具的 Kotlin 語言完全是開源的（open source），而且免費地用於任何目的。它在 Apache 2 許可下使用；在 GitHub（http://github.com/jetbrains/kotlin）上公開發表，歡迎社群貢獻。還可以選擇三種開源 IDE 來開發 Kotlin 應用程式： IntelliJ IDEA Community Edition、Android Studio，以及 Eclipse 皆完全支援。（當然，IntelliJ IDEA Ultimate 也可以。）

現在你已經了解 Kotlin 是什麼樣的語言，讓我們來看看 Kotlin 在特定實際應用中的優勢。

1.3 Kotlin 應用領域

正如我們前面提到的，Kotlin 可以使用的兩個主要領域是伺服器端和 Android 程式的開發。讓我們依次看看這些領域，了解為什麼 Kotlin 適合他們。

1.3.1 伺服器端的 Kotlin

伺服器端編程是一個相當廣泛的概念。它包含以下所有型態的應用程式：

- 將 HTML 頁面回傳給瀏覽器的 Web 應用程式
- 透過 HTTP 公開 JSON API 的行動應用程式的後台
- 透過 RPC 協議與其他微服務（microservices）進行通信的微服務

開發人員已經使用 Java 建構了這些應用程式多年，並累積了大量的框架和技術用來幫助建構它們。這樣的應用通常不是單獨的開發或從頭開始的，幾乎總是有一個現有的系統正在被擴展、改進或取代，新的程式碼必須與系統的現有部分結合在一起，而這些部分可能是在多年前寫的。

Kotlin 在這個環境中的巨大優勢，是它與現有 Java 程式碼的無縫互用性。無論是在編寫新的組件，還是將現有服務的程式碼移植到 Kotlin 中，Kotlin 都是合適的。當需要在 Kotlin 中繼承 Java 類別或以某種方式註釋方法和欄位時，不會遇到問題。

而且好處是系統的程式碼會更緊湊、更可靠、更容易維護。

與此同時，Kotlin 為開發這樣的系統提供了許多新技術。例如，對 Builder 樣式的支援，使你可以使用簡潔的語法建立任何物件圖，同時保留語言中的全套萃取和程式碼重用工具。

該功能最簡單的範例之一，就是 HTML 產生函式庫，它可以用一個簡潔和完全型態安全的解決方案來替換外部樣板語言。下面是一個例子：

```
fun renderPersonList(persons: Collection<Person>) =
    createHTML().table {
        for (person in persons) {
            tr {
                td { +person.name }
                td { +person.age }
            }
        }
    }
}
```

一般的 Kotlin 迴圈

對映到 HTML 標籤的函式

可以輕鬆地將對映到 HTML 標記的函式和標準 Kotlin 語言結構組合起來。不再需要使用單獨的樣板語言、使用單獨的語法學習，只是在產生 HTML 頁面時使用迴圈。

另一個案例是持久性的框架，它使用 Kotlin 簡潔俐落的 DSL 。例如，Exposed 框架（https://github.com/jetbrains/exposed）提供了一個易於閱讀的 DSL，用於描述 SQL 資料庫的結構，完全可從 Kotlin 程式碼執行查詢，並進行完整型態檢查。下面是一個小範例，展示了可以做什麼：

```
object CountryTable : IdTable() {
    val name = varchar("name", 250).uniqueIndex()
    val iso = varchar("iso", 2).uniqueIndex()
}

class Country(id: EntityID) : Entity(id) {
    var name: String by CountryTable.name
    var iso: String by CountryTable.iso
}

val russia = Country.find {
    CountryTable.iso.eq("ru")
}.first()

println(russia.name)
```

描述資料庫中的表格

建立一個對應於資料庫實體的類別

可以使用純 Kotlin 程式碼查詢此資料庫

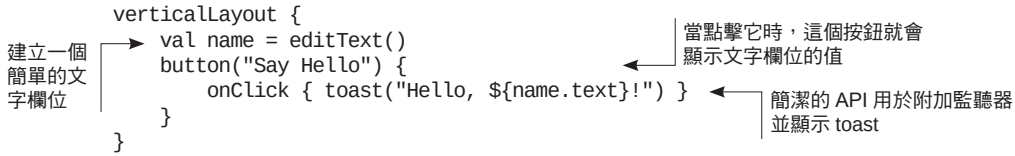
我們將在本書後面的第 7.5 節和第 11 章中更詳細地介紹這些技術。

1.3.2 Android 上的 Kotlin

典型的行動應用程式（mobile application）與典型的企業應用程式有很大不同。它要小得多、少依賴與現有程式碼函式庫的整合，而且通常需要快速發布，同時確保在各種設備上的執行是可靠的。Kotlin 正好適用於這類專案。

Kotlin 的語言特性與支援 Android 框架的特定編譯器插件相結合，將 Android 開發變成了更高效和更愉快的體驗。常見的開發任務，比如將監聽器加到控制元件，或將布局元素綁定到欄位，可以用更少的程式碼來完成，或者根本沒程式碼（編譯器會為你產生）。由 Kotlin 團隊建構的 Anko 函式庫（https://github.com/kotlin/anko），透過在許多標準的 Android API 上加入適用於 Kotlin 的轉接器，進一步提高了體驗。

下面是一個簡單的 Anko 範例，只是為了讓你了解使用 Kotlin 開發 Android 的感覺而已。可以把這段程式碼放在 `Activity` 中，簡單的 Android 應用程式就準備好了！



使用 Kotlin 的另一大優勢是更好的應用可靠性。如果有開發 Android 應用程式的經驗，那麼無疑會對「Unfortunately, Process Has Stopped」這個對話框感到非常熟悉。當應用程式拋出一個未處理的例外（通常是 `NullPointerException`）時，就會顯示此對話框。Kotlin 的型態系統，對 `null` 的精確跟蹤，使 `NullPointerException` 的問題更加緊迫。在 Java 中導致 `NullPointerException` 的大部分程式碼都無法在 Kotlin 中編譯，以保在應用程式到達使用者之前修復錯誤。

同時，因為 Kotlin 完全相容 Java 6，所以它的使用不會引入任何新的相容性問題。你將受益於 Kotlin 的所有新語言功能，即使使用者不執行最新版本的 Android，使用者仍然可以在自己的設備上執行應用程式。

就效能而言，使用 Kotlin 也不會帶來任何不利之處。由 Kotlin 編譯器產生的程式碼與一般的 Java 程式碼一樣有效率地執行。Kotlin 使用的執行時間相當短，因此編譯後的應用程式套件的大小不會增加很多。而當你使用 lambdas 時，許多 Kotlin 標準函式庫函式將內嵌（inline）它們。內嵌 lambdas 確保不會建立新的物件，並且應用程式不會受到額外的 GC 暫停的影響。

考慮到 Kotlin 與 Java 相比的優勢，現在讓我們看看 Kotlin 的哲學——辨別 Kotlin 與針對 JVM 的其他現代語言的主要特徵。

1.4 Kotlin 的哲學

當我們談論 Kotlin 時，我們喜歡說這是一個注重互用性的務實、簡潔、安全的語言。這些詞的意思是什麼？現在來看看它們。

1.4.1 務實

務實（*pragmatic*）對我們來說意味著是簡單的事情：Kotlin 是一種旨在解決現實世界問題的實用語言。其設計基於多年建立大型系統的實務經驗，其功能被選擇來解決許多軟體開發人員遇到的使用情況。而且，JetBrains 和社群內的開發者已經使用 Kotlin 的早期版本好幾年了，他們的回饋已經形成了這個語言的發布版本。這使我們有信心說，Kotlin 可以幫助解決實際專案中的問題。

Kotlin 也不是一門研究性的語言。我們不是試圖提高程式設計語言的藝術層次，而是在電腦科學領域探索創新思維。相反地，只要有可能，我們就依靠已經出現在其他程式語言中的功能和解決方案，並且已經證明是成功的。這樣可以減少語言的複雜性，讓你可以依靠熟悉的概念輕鬆學習。

另外，Kotlin 不強制使用任何特定的編程風格或範例。當你開始學習這語言時，可以使用 Java 經驗中熟悉的風格和技巧。稍後，將逐漸發現 Kotlin 更強大的功能，並學習將它們應用於自己的程式碼中，使其更加簡潔和上手。

Kotlin 實用主義的另一個方面是對工具的關注。一個聰明的開發環境對於開發人員的生產力來說，就像一個好語言一樣的重要和基本；因此，將 IDE 支援作為事後考慮是不可取的。就 Kotlin 而言，IntelliJ IDEA 插件是與編譯器同步開發的，而且語言特性總是以工具為基礎而設計的。

IDE 支援也在幫助你發現 Kotlin 的功能方面發揮重要作用。在很多情況下，這些工具會自動檢測可以被更簡潔的結構所取代的通用程式碼樣式，並為你修復程式碼。透過研究自動修復所使用的語言功能，可以學習在自己的程式碼中應用這些功能。

1.4.2 簡潔

開發人員閱讀現有程式碼比編寫新程式碼花更多時間是眾所皆知的。想像一下，你是團隊的一份子負責開發一個大專案，當需要添加一個新功能或修復一個 bug 時，

你的第一步是什麼？先尋找需要更改的程式碼的確切部分，接著才能執行修復。你會需要讀很多程式碼來找出要做的事情。這段程式碼可能是最近由你的同事、或不再從事這個專案的人、或是你很早以前寫的，只有了解了周圍的程式碼之後，才能進行必要的修改。

程式碼越簡單，越簡潔，就能越快理解到底發生了什麼事。當然，好的設計和有意義的變數名稱在這裡有著重要的作用。但語言的選擇及其**簡潔**程度也很重要。如果語言的語法清楚地表達了你讀的程式碼的意圖，並且不需要用規定如何完成意圖的樣板來模糊它，那麼這個語言是簡潔的。

在 Kotlin，我們努力確保你編寫的所有程式碼都具有意義，而不僅僅是滿足程式碼結構要求。很多標準的 Java 樣板，比如 getter、setter，以及為欄位分配構造函式參數的邏輯都隱含在 Kotlin 中，並且不會混淆你的程式碼。

不必要冗長程式碼的另一個原因，是必須編寫明確的程式碼來執行一般的任務，例如在集合中尋找元素。就像很多其他現代語言一樣，Kotlin 有一個豐富的標準函式庫，可以讓你用函式庫方法呼叫來替代這些冗長、重複的程式碼區段。Kotlin 對 lambda 表示式的支援，使得可能輕鬆地將小區塊程式碼傳遞給函式庫函式。這樣可以將所有通用部分封裝在函式庫中，並且只保留使用者程式碼中唯一、特定於任務的部分。

同時，Kotlin 不會試圖將程式碼萎縮到最少的字元。例如，即使 Kotlin 支援運算子重載，使用者也不能定義他們自己的運算子。因此，函式庫開發者不能用神秘的標點符號替換方法名稱。單字通常比標點符號更容易閱讀、更容易找到文件。

簡潔的程式碼可以省下許多輸入的時間，更重要的是縮短了閱讀的時間，這可以提高工作效率、更快地完成工作。

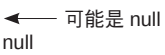
1.4.3 安全

一般來說，當我們說程式語言是**安全**的，其意思是它的設計可以防止程式中某些型態的錯誤。當然，沒有辦法保證沒有錯誤，沒有一種語言可以防止所有可能產生的錯誤。另外，防止錯誤通常需要付出代價，需要為編譯器提供關於程式預期操作的更多資訊，以便編譯器可以驗證該資訊與程式的功能是否相符。正因為如此，所獲得的安全等級與需要更詳細的解釋所造成生產效率的降低，這之間總是存在一個折衷點。

而使用 Kotlin，我們已經試著達到比 Java 更高的安全等級，且總體成本更低。在 JVM 上執行已經提供了很多安全保證：例如，記憶體安全、防止緩衝區溢出，以及由錯誤使用動態分配的記憶體引起的其他問題。作為 JVM 上的靜態語言，Kotlin 還可以確保應用程式的型態安全。這比使用 Java 的成本更低：不必指定所有的型態宣告，因為在多數的情況下，編譯器會自動推論其型態。

Kotlin 也不止於此，更多的錯誤可以透過在編譯時檢查出來，而不是在執行時產生錯誤。最重要的是，Kotlin 努力從程式中刪除 `NullPointerException`。Kotlin 的型態系統跟蹤值可以和不可以為 `null`，並禁止在執行時導致 `NullPointerException` 例外的操作。為此所需的額外成本是非常小的：標記一個型態是可以為空只需一個在最後加上一個問號的字元即可：

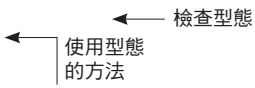
```
val s: String? = null
val s2: String = ""
```



另外，Kotlin 提供了許多簡便的方式來處理可空的資料，這有助於減少應用程式崩潰。

Kotlin 幫助避免的另一種例外型態是 `ClassCastException`。當你把一個物件轉換成一個型態，而沒有先檢查它是否具有正確的型態時，就會發生這種狀況。在 Java 中，開發人員經常會忽略檢查，因為型態名稱必須在檢查和之後的轉型（cast）間重複執行。另一方面，在 Kotlin 中，檢查和轉型組合成一個單一的操作：一旦檢查了型態，可以引用該型態的成員，而不需要任何額外的轉型。因此，沒有理由跳過檢查，也就沒有機會犯錯。下面是一個範例：

```
if (value is String)
    println(value.toUpperCase())
```



1.4.4 互用性

關於互用性，第一個擔心可能是「可以使用現有的函式庫嗎？」對於 Kotlin 來說，答案是「絕對可以」。無論函式庫要求你使用哪種型態的 API，都可以使用 Kotlin 進行操作。可以呼叫 Java 方法、繼承 Java 類別並實現介面、將 Java 註釋應用於你的 Kotlin 類別，等等。

與其他一些 JVM 語言不同，Kotlin 進一步提高了互用性，使得從 Java 呼叫 Kotlin 程式碼變得毫不費力。不需要任何技巧：Kotlin 類別和方法可以像正規的 Java 類別和方法一樣被呼叫。這給你在專案中的任何位置混合使用 Java 和 Kotlin 程式碼，提供了極大的靈活性。當在 Java 專案中開始採用 Kotlin 時，你可以在程式碼函式

庫中的任何一個類別上，執行 Java-to-Kotlin 轉換器，其餘程式碼將繼續編譯和工作，無需做任何修改。無論轉換的類別作用為何，這都會有效。

Kotlin 關注互用性的另一個領域是，盡可能使用現有的 Java 函式庫。例如，Kotlin 沒有自己的集合函式庫。它完全依賴於 Java 標準函式庫類別，並將它們繼承為更多的功能，以便在 Kotlin 中更方便地使用。（我們將在第 3.3 節更詳細地介紹這一機制。）這意味著從 Kotlin 呼叫 Java API 時，不需要包裝或轉換物件，反之亦然。Kotlin 提供的所有 API 豐富功能都是免費提供的。

Kotlin 工具還提供對跨語言專案的全面支援。它可以編譯 Java 和 Kotlin 檔案的任意組合，而不管它們如何相互依賴。IDE 功能也可以跨語言工作，使你可以：

- 在 Java 和 Kotlin 檔案之間自由導覽
- 除錯混合語言專案，並在使用不同語言編寫的程式碼之間進行切換
- 重構 Java 方法，並在正確更新的 Kotlin 程式碼中使用它們，反之亦然

希望現在我們已經說服你試著用用看 Kotlin。而要怎麼開始使用它？在下一節，將討論編譯和執行 Kotlin 程式碼的過程，無論是用命令行介面，還是使用不同的工具。

1.5 使用 Kotlin 工具

就像 Java 一樣，Kotlin 是一種編譯語言。意味著在執行 Kotlin 程式碼之前需要編譯它。讓我們來討論一下編譯的過程，接著看看為你做好的不同工具。如果需要更多關於設置環境的資訊，請參閱 Kotlin 網站（https://kotlinlang.org/docs/tutorials）上的「Tutorials」部分。

1.5.1 編譯 Kotlin 程式碼

Kotlin 程式碼通常儲存在副檔名為 .kt 的檔案中。Kotlin 編譯器像 Java 編譯器一樣，分析程式碼，並產生 .class 檔案。產生的 .class 檔案隨後使用運作應用程式型態的標準過程，進行打包和執行。在最簡單的情況下，可以使用 `kotlinc` 命令從命令行編譯程式碼，並使用 `java` 命令來執行程式碼：

```
kotlinc <source file or directory> -include-runtime -d <jar name>
java -jar <jar name>
```

圖 1.1 顯示了 Kotlin 建構過程的精簡描述。

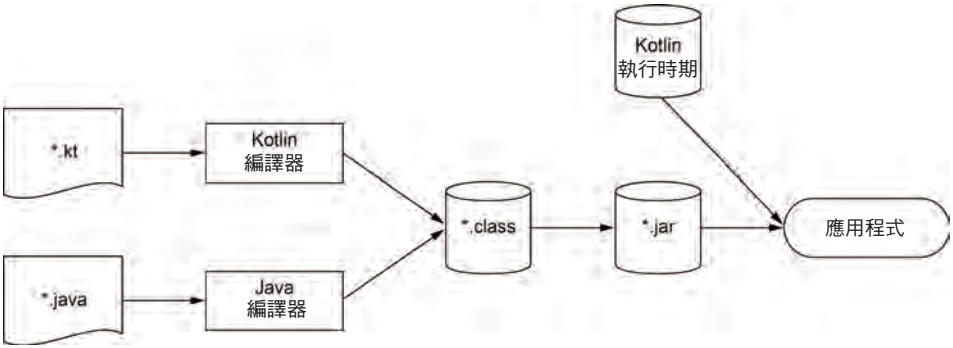


圖 1.1　Kotlin 建構過程

使用 Kotlin 編譯器編譯的程式碼取決於 *Kotlin* **執行時期函式庫**。它包含 Kotlin 自己的標準函式庫類別的定義，以及 Kotlin 為標準 Java API 加入的繼承。執行時期函式庫需要與應用程式一起配置。

在大多數的實際案例中，會使用 Maven、Gradle 或 Ant 等建構的系統來編譯程式碼。Kotlin 與上述這些建構系統相容，在附錄 A 將討論它們的詳細資訊。這些建構系統還支援將 Kotlin 和 Java 組合在同一程式碼函式庫中的混合語言專案。另外，Maven 和 Gradle 負責載入 Kotlin 執行時期函式庫隸屬，做為你的應用程式。

1.5.2　IntelliJ IDEA 和 Android Studio 的插件

Kotlin 的 IntelliJ IDEA 插件 （plug-in） 與語言並行開發，是 Kotlin 最全面的開發環境。它成熟穩定，為開發 Kotlin 提供了一套完整的工具。

IntelliJ IDEA 15 及後續版本包含了 Kotlin 插件，因此不需要其他設定。可以使用免費的開源 IntelliJ IDEA Community Edition 或 IntelliJ IDEA Ultimate。在 New Project 對話框中選擇 Kotlin，就可以開始了。

如果使用 Android Studio，則可以從插件管理器安裝 Kotlin 插件。在「Settings」中，選擇「Plugins」，接著點擊「Install JetBrains Plugin」按鈕，並從列表中選擇「Kotlin」。