前言

歡迎

「風箏是逆風而起，而非順風。」

—John Neal，摘自 *Enterprise and Perseverance*（*The Weekly Mirror*）。

歡迎來到《*Spring Boot*：**建置與執行**》。很高興你在這裡。

現今還有很多其他的 Spring Boot 書籍，那些好書都是用意良善的人所撰寫的，但是，每位作者都必須決定要在他們的書中包含哪些內容、排除哪些內容、如何展示那些內容，以及大大小小的更多決定，這些決策使他們的書與眾不同。對某位作者而言，感覺是可有可無的材料，對另一位作者來說，可能是絕對必要的。我們都是開發者，和所有的開發人員一樣，我們也都有自己的看法。

我的觀點是，有些部分被遺漏了，而我認為那些東西若不是必要的，就是對新加入 Spring Boot 的開發人員非常有幫助。隨著我與世界各地處於不同階段的開發者進行越來越多的交流，這份缺漏部分的清單也在不斷增加。我們都在不同的時間以不同的方式學習不同的東西。因此才有了這本書。

如果你是 Spring Boot 的新手，或覺得加強你的基礎知識會很有用（讓我們面對現實吧，這樣什麼時候會沒有用呢？），那本書就是為你而寫的。這是溫和友善的入門介紹，涵蓋了 Spring Boot 的關鍵功能，同時描述這些功能在**真實世界**中的應用。

感謝你和我一起踏上這段旅程。讓我們開始吧！

第一章

Spring Boot 概述

本章將探討 Spring Boot 的三大核心功能，以及它們如何成為身為開發人員的你之力量強化器。

Spring Boot 的三大基礎功能

Spring Boot 的三個核心功能是簡化的依存性管理（dependency management）、簡化的部署（deployment），以及自動組態（autoconfiguration），其他所有功能都建立在這三個基礎之上。

簡化的依存性管理之啟動器

Spring Boot 的一個超讚之處在於它讓依存性的管理……變得容易處理。

如果你開發過的軟體需要匯入（import）其他軟體，那麼無論時間長短，幾乎可以肯定是，你一定會對依存關係的管理感到頭疼。你在應用程式中提供的任何功能，通常都需要大量的「前線（frontline）」依存關係。例如，如果你想提供一個 RESTful Web API，你必須提供一種方法透過 HTTP 對外供應你的端點、收聽請求，並將這些端點與處理請求用的方法或函式綁定在一起，然後建構並回傳適當的回應。

幾乎無一例外的是，每個主要的依存關係都包含了許多其他的次要依存關係，以履行所承諾的功能。接續我們提供 RESTful API 的例子，我們可以預期會看到一個依存關係群集（放在某些合理但尚有討論空間的結構中），其中包含會以特定格式（例如 JSON、XML、HTML）提供回應的程式碼、將物件轉換為請求格式的程式碼；收聽和處理請求

並回傳回應的程式碼，還有程式碼來解碼建立多功能 API 用的複雜 URI；支援各種線路層協定（wire protocols）的程式碼等等。

即使是這個相當簡單的例子，我們的建置檔中就已經可能需要大量的依存關係。而且此時我們甚至尚未考慮我們應用程式中希望包含哪些功能，只考慮到它對外的互動。

現在，我們來談談版本（versions），那些依存關係中每一個的版本。

多個程式庫一起使用，需要一定的嚴謹性，因為一個個依存關係的某個版本可能只用另一個依存關係的特定版本進行過測試（或甚至是與之並用才能正確運作）。當這些問題不可避免地出現時，就會導致我所說的「依存關係打地鼠（Dependency Whack-a-Mole）」現象。

就像它同名的嘉年華遊戲，Dependency Whack-a-Mole 可能是一種令人沮喪的體驗。跟它的名字不同的是，追尋和打擊因依存關係之間的不匹配而產生的臭蟲（bug）時，沒有獎品可拿，只有難以捉摸的最終診斷和追查它們所浪費的**大量時間**。

進入 Spring Boot 和它的啟動器（starters）。Spring Boot 啟動器是所謂的物料清單（Bills of Materials，BOM），它建立在這樣一個已被證明的前提之下：絕大多數情況下，你提供一種特定功能時，你幾乎每次都是以同樣的方式來做的。

在前面的例子中，每次構建 API 時，我們都會對外供應端點、收聽請求、處理請求，轉換為物件或從物件轉為其他格式，以多種標準格式交換資訊，使用特定的協定透過線路收發資料等等。這種設計 / 開發 / 使用（design/development/usage）的模式並不會有太大變化，這是整個業界都採用的做法，頂多只有微小差異。而且就和其他類似的模式一樣，它很方便地都被 Spring Boot 啟動器囊括其中。

新增單一個啟動器，例如 `spring-boot-starter-web`，就能夠以**單一的應用程式依存關係**（*single application dependency*）提供所有的這些相關功能。這單一啟動器所包含的所有依存關係也都是版本同步（version-synchronized）的，這意味著它們已經一起被成功地測試過了，而所包含的程式庫 A 的版本已被證明能與所包含的程式庫 B 和 C 及 D 等等的版本一起正常運行。這極度簡化了你的依存關係清單，以及你的生活，因為你得提供的應用程式關鍵功能之依存關係之間難以識別的版本衝突問題，已經被它確實消除了。

在極少數情況下，當你必須加入由內建的依存關係的不同版本所提供的功能時，你可以單純覆寫那個測試過的版本就行了。

如果你非得覆寫一個依存關係的預設版本，就那樣做吧……但你也許應該提高你的測試水平，以減少你這樣做所帶來的風險。

如果某些依存關係對你的應用程式來說是不必要的，你也可以排除它們，但同樣的注意事項也適用。

總而言之，Spring Boot 的啟動器概念大大地簡化了你的依存關係，減少了為應用程式添加成套功能所需的工作量；此外，這也大幅減低了你在測試、維護和升級它們時所需負擔的額外開銷。

簡化版部署的可執行 JAR

很久以前，在應用伺服器（application servers）橫行的時代，Java 應用程式的部署（deployments）是一件很複雜的事情。

為了要賦予運作中的應用程式某項功能，譬如說資料庫的存取能力（就像今天的許多微服務和當時及現在幾乎所有的單體服務所需要的那樣），你得進行以下工作：

1. 安裝並設定 Application Server（應用伺服器）。
2. 安裝資料庫驅動程式（database drivers）。
3. 創建資料庫連線（database connection）。
4. 建立一個連線集區（connection pool）。
5. 建置並測試你的應用程式。
6. 將你的應用程式和它的（通常是為數眾多的）依存關係部署到 Application Server 上。

請注意，這個清單假設有管理員會設置機器或虛擬機器（virtual machine），並且在某些時候你已經獨立於這個過程建立好了資料庫。

Spring Boot 將這個繁瑣的部署過程徹底顛覆，將之前的那些步驟壓縮為一步，或者說兩步，如果你將複製的動作或 `cf push` 單個檔到目的地的動作算成實際的一個**步驟**（*step*）的話。

Spring Boot 並非所謂的 über JAR 的起源，但它為之帶來了革命性的變化。Spring Boot 的設計者並沒有從應用程式的 JAR 和所有依存的 JAR 中取出每一個檔案，然後把它們組合成單一的目標 JAR（這有時也被稱為 *shading*），而是從真正新穎的角度來處理問題：如果我們可以**把 *JAR* 內嵌成為巢狀**（*nest JAR*），同時保留它們要交付的預期格式，那會怎樣？

內嵌 JAR 而非遮蔽（shading）它們，可以減輕**許多**潛在問題的威脅，因為當依存的 JAR A 和依存的 JAR B 各自使用 C 的不同版本時，並不會遇到版本衝突；它還消除了由於重新包裝軟體，並將之與使用不同許可證（license）的其他軟體結合，而產生的潛在法律問題。讓所有依存的 JAR 保持原始格式，可以乾淨俐落地避免這些問題和其他議題。

如果你想提取 Spring Boot 可執行 JAR（executable JAR）的內容，也是很簡單的事情。在某些情況下，這樣做有一些很好的理由存在，我也會在本書中討論這些緣由。至於現在，你只需知道 Spring Boot 可執行 JAR 已經為你準備就緒了。

包含所有依存關係的這單一個 Spring Boot JAR 使部署變得輕而易舉。Spring Boot 外掛（plug-in）不需要收集和驗證所有的依存關係，而是確保它們都被壓縮到輸出的 JAR 中。一旦你有了這些，只要執行像 `java -jar <SpringBootAppName.jar>` 這樣的一道命令，應用程式就可以在具備 Java 虛擬機器（Java Virtual Machine，JVM）的地方運行。

還有更多。

藉由在你的建置檔（build file）中設置單一個特性（property），Spring Boot 的建置外掛（build plug-in）也可以讓這單一的 JAR 完全可（單獨）執行。還是假設有 JVM 存在，而不是得打入或編寫麻煩的整行命令 `java -jar <SpringBootAppName.jar>`，你可以單純鍵入 `<SpringBootAppName.jar>`（當然，要用你的檔名替換），然後就這樣，可以開始運行了。沒有比這更簡單的了。

自動組態

自動組態（autoconfiguration）有時被那些剛接觸 Spring Boot 的人稱為「魔法」，它也許是 Spring Boot 帶給開發者最大的「力量倍增器（force multiplier）」。我經常把它稱為開發人員的超能力：Spring Boot 將新穎的主張帶入廣泛並重複出現的使用案例中，藉此賦予了你**瘋狂的生產力**（*insane productivity*）。

軟體中的主張？這有什麼用!?

如果你已經做了很長時間的開發人員，你肯定會注意到一些模式經常重複。當然，並非一定如此，但比例很高，可能有 80-90% 的時間裡，事情都落在設計、開發或活動的某個範圍內。

我在前面提到了軟體內部的這種重複性，因為這就是使得 Spring Boot 的啟動器有驚人的一致性和實用性的原因。這種重複性也意味著，當涉及到一定得編寫以完成特定任務的程式碼時，就會是精簡這些活動的好時機。

借用 Spring Data（一個與 Spring Boot 相關並因為 Spring Boot 而變得可能的專案）的例子，我們知道，每次我們需要存取一個資料庫時，都得打開與該資料庫之間某種形式的連線。我們也明白，當我們的應用程式完成任務時，這個連線必須被關閉以避免潛在的問題。在這過程中，我們可能會使用查詢（簡單或複雜的、唯讀或可寫的），對資料庫發出很多請求，而這些查詢會需要一些努力才能正確建立。

現在想像一下，我們可以簡化這一切。在我們指定資料庫時，自動開啟一個連線；應用程式終止時，自動關閉連線。遵循一種簡單且可預期的慣例，只要你這位開發人員最少的努力就能**自動**創建出查詢。使得這最少量的程式碼也能輕鬆自訂，同樣是按照簡單的慣例，可靠地建立出一致且有效率的複雜訂製查詢。

這種編程方法有時被稱為**慣例重於組態**（*convention over configuration*），如果你是某項特殊慣例的新手，這乍看之下可能會顯得有些突兀，但若你以前實作過類似的功能，經常要寫幾百行重複、令人頭疼的設置 / 拆除 / 組態（setup/teardown/configuration）程式碼來完成最簡單的任務，那麼這就如同一股新鮮的空氣。Spring Boot（以及大多數 Spring 專案）都遵循**慣例重於組態**的真言，提供了這樣的保證：如果你遵循簡單、成熟且有豐富說明文件的慣例來做某件事，你必須編寫的組態程式碼就會是最少的，或者根本不需要。

自動組態賦予你超能力的另一種方式是透過 Spring 團隊對於「開發者優先（developer-first）」環境組態的絕對專注。身為開發人員，若我們能專注於手頭的任務，而非千篇一律的設置瑣事，我們的生產力就會是最高的。Spring Boot 如何實現這一點呢？

這裡讓我們借用另一個與 Spring Boot 相關的專案 Spring Cloud Stream 作為例子：連接到 RabbitMQ 或 Apache Kafka 等訊息傳遞平台時，開發人員通常必須為上述平台指定某些組態才能連接並使用它，例如主機名稱（hostname）、通訊埠（port）、證明資訊（credentials）等。注重開發體驗，意味著在沒有指定的情況下，所提供的預設值會**有利於開發者在本地端的工作**：localhost、預設通訊埠等。這作為一種**主張**（*opinion*）

是有道理的，因為這對開發環境來說，幾乎 100% 是一致的，雖然在生產環境中並不一定如此。在生產環境中，由於平台和託管環境（hosting environments）的差異很大，你就得提供具體的設定值。

使用這些預設值的共用開發專案，也消除了設置開發者環境所需的大量時間。你贏了，你的團隊也贏了。

有的時候，當你的具體用例並不完全符合 80-90% 的典型用例，這時你就屬於另外 10-20% 的有效用例。在那些情況中，自動組態可以選擇性地被覆寫，甚至完全停用自動組態，但當然，你那時會失去所有的超能力。覆寫特定主張通常是依照你的意願設定一或多個特性，或者提供一或多個「bean」來完成 Spring Boot 通常會替你進行自動組態的事情；換句話說，在非得那樣做的極少數情況下，這通常是一件非常簡單的事情。最後，自動組態是一項強大的工具，它默默地、不知疲倦地代替你處理工作，使你的生活更輕鬆，工作效率也大大提升。

總結

Spring Boot 的三大核心功能是簡化的依存性管理、簡化的部署和自動組態，其他的一切都建立在這三個基礎上。這三種功能都是可以自訂的，但你很少需要那樣做。而這三者都致力於讓你成為一名更好、更有效率的開發人員。Spring Boot 為你帶來了雙翼！

在下一章中，我們將介紹你在開始建立 Spring Boot 應用程式時的一些好選項。選擇是好東西！

第二章

挑選你的工具並開始動手

開始創建 Spring Boot 應用程式很容易，如你很快就會看到的那樣。最困難的部分可能是決定你想挑選的可用選項。

在本章中，我們將檢視你在建立 Spring Boot 應用程式時的一些絕佳選擇：建置系統（build systems）、語言、工具鏈（toolchains）、程式碼編輯器等。

Maven 或 Gradle？

從歷史上看，Java 應用程式開發者有幾個專案建置工具（build tools）的選擇。隨著時間的推移，有些工具已經失寵了（出於對的原因），現在我們以兩個工具為中心凝聚成了一個社群：Maven 和 Gradle。這兩個工具 Spring Boot 都有同等的支援。

Apache Maven

Maven 是建置自動化系統（build automation system）流行且可靠的一個選擇，它已經存在相當長的一段時間，從 2002 年開始，到 2003 年成為 Apache 軟體基金會（Apache Software Foundation）的頂級專案。它的宣告式做法（declarative approach）在概念上比當時和現在的替代選擇更為簡單：只需創建一個名為 *pom.xml* 的 XML 格式檔，其中包含所需的依存關係和外掛。當你執行 `mvn` 命令，你可以指定一個要完成的「階段（phase）」，它可以完成所需的任務，如編譯、刪除先前的輸出、打包（packaging）、執行某個應用程式等等：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>2.4.0</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>com.example</groupId>
        <artifactId>demo</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>demo</name>
        <description>Demo project for Spring Boot</description>

        <properties>
                <java.version>11</java.version>
        </properties>

        <dependencies>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter</artifactId>
                </dependency>

                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-test</artifactId>
                        <scope>test</scope>
                </dependency>
        </dependencies>

        <build>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-plugin</artifactId>
                        </plugin>
                </plugins>
        </build>

</project>
```

Maven 也會創建並預期一個依照慣例結構化的特定專案。你通常不應該偏離這個結構太多（如果有的話），除非你準備好對抗你的建置工具，這會是一種適得其反的追求。對於絕大多數專案來說，慣例的 Maven 結構都能完美地運作，所以這不太可能是你需要改變的東西。圖 2-1 顯示了一個具有典型 Maven 專案結構的 Spring Boot 應用程式。

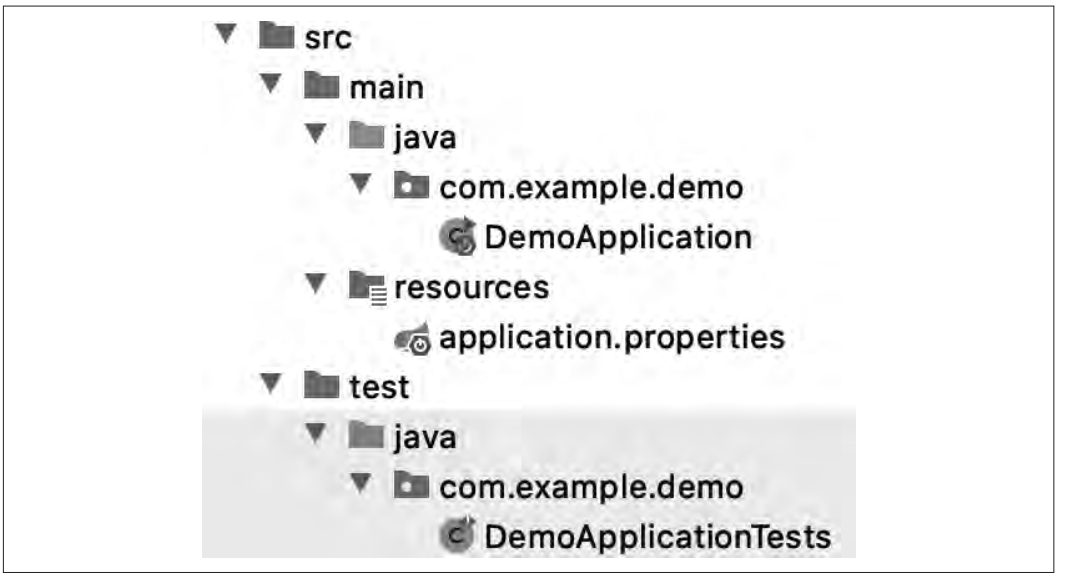

圖 2-1　一個 Spring Boot 應用程式中的 Maven 專案結構

關於 Maven 預期的專案結構，更多的細節請參閱 The Maven Project 的 Introduction to the Standard Directory Layout（*https://oreil.ly/mavenprojintro*）。

如果有一天，你覺得 Maven 的專案慣例或結構嚴密的建置做法顯得過於拘謹，那還有另一個很好的選擇。

Gradle

Gradle 是建置 Java 虛擬機器（Java Virtual Machine，JVM）專案的另一個熱門選擇。Gradle 最早發佈於 2008 年，它利用一個 DSL（Domain Specific Language，領域特定語言）來產生一個 *build.gradle* 建置檔案，它既簡潔又有彈性。Spring Boot 應用程式的一個 Gradle 建置檔範例如下：

```
plugins {
        id 'org.springframework.boot' version '2.4.0'
        id 'io.spring.dependency-management' version '1.0.10.RELEASE'
        id 'java'
}

group = 'com.example'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '11'

repositories {
        mavenCentral()
}

dependencies {
        implementation 'org.springframework.boot:spring-boot-starter'
        testImplementation 'org.springframework.boot:spring-boot-starter-test'
}

test {
        useJUnitPlatform()
}
```

Gradle 允許身為開發者的你，挑選要用 Groovy 或 Kotlin 程式語言作為 DSL。它還提供了數個功能，旨在減少你等待專案建置的時間，例如下面這幾個：

- Java 類別的增量編譯（incremental compilation）
- 避免 Java 的編譯（在沒有發生變化的情況下）
- 一個專門用於專案編譯的常駐程式（daemon）

在 Maven 和 Gradle 之間擇一使用

你對建置工具的選擇在此時聽起來似乎並沒有什麼選擇可言，為什麼不乾脆選擇 Gradle 就好呢？

Maven 較為嚴格的宣告式（有些人可能會說是主張式的）做法使專案與專案之間、環境與環境之間保持了驚人的一致性。如果你遵循 Maven 的方式，通常很少會有問題出現，讓你能專注於你的程式碼，建置過程不會有什麼無謂的麻煩。

作為以程式設計 / 指令稿建構（programming/scripting）為中心的建置系統，Gradle 在消化新語言版本的初始發行版時，偶爾會出現問題。Gradle 團隊反應迅速，通常會急速發派這些問題以進行處理，但如果你喜歡（或必須）立即潛入早期試用的語言版本，這就值得考慮。

Gradle 可能建置地更快，有時會明顯**快很多**，尤其是在大型專案中。即使是這樣，對於你基於微服務（microservices-based）的典型專案而言，在類似的 Maven 和 Gradle 專案之間，建置時間不太可能相差那麼多。

Gradle 的靈活性對於簡單的專案和有非常複雜建置需求的專案而言，可以說是一股清流；但特別是在那些複雜的專案中，如果事情沒有按照你預期的方式運作，Gradle 額外的彈性可能會導致你花費更多的時間進行調整和故障排除。TANSTAAFL（There Ain't No Such Thing as a Free Lunch，天下沒有白吃的午餐）。

Spring Boot 同時支援 Maven 和 Gradle，而如果你使用 Initializr（將在後面的章節中介紹），就會為你創建專案和所需的建置檔，讓你快速啟動和運行。簡而言之，兩種選擇都可以嘗試看看，然後挑選最適合你的方式。無論用的是哪一種，Spring Boot 都很樂意支援你。

Java 或 Kotlin？

雖然可以在 JVM 上使用的語言有很多，但有兩個語言有最廣泛被使用：一個是最初的 JVM 語言 Java，另一個是相對較新的語言 Kotlin。這兩個語言在 Spring Boot 中都是一等公民。

Java

依據你是將公開的 1.0 版本還是該專案的起源視為其正式的誕生日期來看，Java 分別已經存在 25 年或 30 年了。不過，它並非停滯不前。自 2017 年 9 月以來，Java 一直以 6 個月的發行週期演進，使得功能的改進比之前更加頻繁。維護者清理了源碼庫，修剪掉了因為新功能而變得沒必要的功能，並引進了由 Java 社群驅動的重要功能。Java 比以往任何時候都更有活力。

這種活躍的創新步伐，加上 Java 的長壽和一貫的回溯相容性（backward compatibility），意味著全世界每天都有無數的 Java 商店在維護和建立關鍵的 Java 應用程式。這些應用中有許多都使用了 Spring。

Java 幾乎可以說是構成了整個 Spring 源碼庫的堅實基礎，因此，它是建置 Spring Boot 應用程式的最佳選擇。想要檢視 Spring、Spring Boot 和所有相關專案的程式碼，只需訪問承載它們的 GitHub 並在那裡查看程式碼，或者複製專案進行離線檢閱即可。而且，由於有大量的範例程式碼、範例專案和使用 Java 編寫的「入門（Getting Started）」指南，使用 Java 編寫 Spring Boot 應用程式，比起市面上任何其他工具鏈的組合，更可能得到更好的支援。

Kotlin

相對而言，Kotlin 就算是後起之秀了。Kotlin 是由 JetBrains 在 2010 年所創建，並在 2011 年公開，它的建立是為了解決 Java 可用性方面的缺陷。Kotlin 從一開始就被設計成：

簡潔的

Kotlin 只需要最少的程式碼來向編譯器（以及自己和其他開發人員）清晰地傳達意圖。

安全的

Kotlin 藉由在**預設情況下**避免空值（null values）來消除與空值相關的錯誤，除非開發者特地覆寫了行為以允許空值。

可互通的

Kotlin 的目標是與所有既存的 JVM、Android 和瀏覽器程式庫順利互通。

對工具友好

你能在眾多 IDE（Integrated Development Environments，整合式開發環境）中或從命令列建置 Kotlin 應用程式，就像 Java 一樣。

Kotlin 的維護者不僅非常謹慎，也以極快的速度擴展了該語言的功能。因為他們的核心設計焦點並非 25 年多的語言相容性，所以能夠迅速添加非常實用的功能，而這些功能很可能會出現在 Java 之後的一些版本中。

除了簡潔性，Kotlin 也是一門非常流暢的語言。先不談太多細節，有幾個語言特色促成了這種語言學上的優雅性，其中包含 `extension functions` 和 `infix notation`。我將在後面更深入討論這些概念，但 Kotlin 使這樣的語法選項變成可能：

```
infix fun Int.multiplyBy(x: Int): Int { ... }

// 使用中綴記法（infix notation）呼叫函式
1 multiplyBy 2

// 等同於
1.multiplyBy(2)
```

正如你所想像的那樣，能夠定義你自己的更流暢的「語言中的語言」，對 API 設計來說可是一大福音。結合 Kotlin 的簡潔性，這可使 Kotlin 編寫的 Spring Boot 應用程式比它們相應的 Java 程式更短、更易讀，而且在意圖的溝通上沒有損失。

自 2017 年秋季發佈 5.0 版本以來，Kotlin 一直都是 Spring Framework 中的一等公民，在那之後，完整的支援也延伸到 Spring Boot（2018 年春季）和其他的元件專案。此外，所有的 Spring 說明文件都正在擴展，將會包含 Java **以及** Kotlin 的範例。這意味著，在效果上，你可以像使用 Java 一樣，輕鬆地以 Kotlin 編寫整個 Spring Boot 應用程式。

在 Java 和 Kotlin 之間擇一使用

令人訝異的是，你實際上不必選擇。Kotlin 編譯出來的位元組碼（bytecode）輸出與 Java 相同，而且由於創建出來的 Spring 專案可以同時包含 Java 原始碼檔案（source files）和 Kotlin，並且能夠輕易調用這兩種編譯器，所以即使是**在同一個專案中**，你也可以選用對你更有意義的任何一個。魚與熊掌兼得的感覺如何啊？

當然，如果你更喜歡其中一個，或有其他個人或專業的約束，你顯然可以只用其中一個或另一個來開發整個應用程式。有選擇是好事，不是嗎？

選擇一個 Spring Boot 版本

對於生產用的應用程式（production applications），你應該始終使用當前版本的 Spring Boot，但有以下暫時性且適用範圍窄的例外：

- 你們目前執行的是舊版本，但正依據某種順序升級、重新測試並部署你們的應用程式，因此只是進度尚未到達這個特定的 app 而已。
- 你們目前執行的是舊版本，因為有一個已確定的衝突或錯誤，已經向 Spring 團隊回報了，並被指示等待 Boot 或問題所在的依存關係之更新。
- 你們需要利用 snapshot（快照）、milestone（里程碑）或 release candidate（發佈候選）這類 GA（General Availability，可供一般使用）前版本中的功能，並願意接受尚未宣佈為 GA（即「可供生產使用」）的程式碼固有的風險。

Snapshot、milestone 和 Release Candidate（RC）版本在發佈前都經過了廣泛的測試，所以在確保其穩定性方面已經做了大量的嚴格工作。不過在完整的 GA 版本被批准並發佈之前，總是有可能出現 API 變更、錯誤修正等情況。對你應用程式的風險是很低的，但是考慮使用任何早期取用（early-access）軟體時，你都得自行決定（並測試和確認）是否可以控制這些風險。

Spring 的 Initializr

創建 Spring Boot 應用程式的方法有很多，但大多數都會回到單一的起點：Spring Initializr，如圖 2-2 所示。

有時單純以 `start.spring.io` 這個 URL 來參考，Spring Initializr 可透過著名 IDE 的專案建立精靈（project creation wizards）、命令列（command line）取用，最常見的是透過 Web 瀏覽器。使用 Web 瀏覽器提供了一些額外的實用功能，而這些功能目前無法透過其他途徑取用。

要以「最佳的可能方式（Best Possible Way）」開始建立 Spring Boot 專案，請指引瀏覽器前往 *https://start.spring.io*。從那裡，我們會選擇幾個選項，然後開始進行。

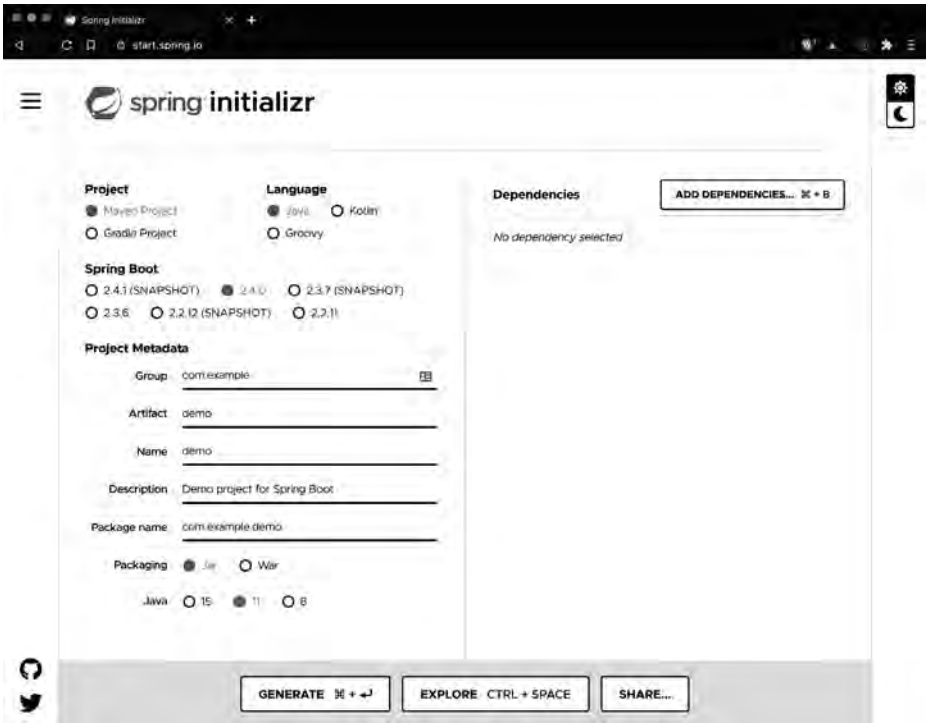

圖 2-2　Spring Initializr

安裝 Java

我假設，到了這一步，你已經事先在你的機器上安裝了目前版本的 Java Development Kit（JDK），這有時也被稱為 *Java Platform, Standard Edition*。如果尚未安裝 Java，你需要在繼續之前先行安裝。

詳細的指示並不在本書的範圍之內，但提供一些建議也無妨，對吧？ :)

我發現在機器上安裝和管理一或多個 JDK 最簡單的方法是使用 SDKMAN!（*https://sdkman.io*）。這個套件管理器（package manager）還便於你安裝以後會用到的 Spring Boot 命令列介面（Command Line Interface，CLI）以及許多其他工具，所以它是一個非常實用的工具 app。依循 *https://sdkman.io/install* 的指示進行，就能助你準備就緒。

SDKMAN! 是用 bash（Unix/Linux shell）指令稿編寫的，因此，它可以原生地（natively）在 MacOS 和 Linux 以及其他具有 Unix 或 Linux 基礎的作業系統上安裝和運行。SDKMAN! 也可以在 Windows 上運行，但不是原生的；為了要在 Windows 環境中安裝並執行 SDKMAN!，你必須先安裝 Windows Subsystem for Linux（WSL，*https://oreil.ly/WindowsSubL*）、Git Bash for Windows（*https://oreil.ly/GitBashWin*）和 MinGW（*http://www.mingw.org*）。細節請參閱前面連結的 SDKMAN! 安裝頁面。

在 SDKMAN! 中，使用 `sdk list java` 來查看選項，然後執行 `sdk install java <insert_desired_java_here>` 就能安裝所需的 Java 版本。有許多很好的選擇存在，但一開始，我建議你選擇目前以 AdoptOpenJDK 和 Hotspot JVM 所打包的 Long Term Support（LTS，長期支援）版本，例如 `11.0.7.hs-adpt`。

如果你出於某種原因不想使用 SDKMAN!，你也可以選擇直接從 *https://adoptopenjdk.net* 下載並安裝 JDK。這樣做可以讓你設定好並開始運行，但會增加更新的難度，而且對你將來的更新動作或多個 JDK 的管理沒有幫助。

要開始使用 Initializr，首先要選擇我們計畫在專案中使用的建置系統。如前所述，我們有兩個很好的選擇：Maven 和 Gradle。在此例中，我們選擇 Maven。

接著，我們將選擇 Java 作為這個專案的（語言）基礎。

你可能已經注意到了，Spring Initializr 為所呈現的選項挑選了夠多的預設值，足以創建一個專案，無需你做任何輸入。抵達這個網頁時，Maven 和 Java 都已經預先被選定了。當前版本的 Spring Boot 也是如此，而對於這個專案（以及大多數的專案）來說，那都是你會想要選擇的。

我們可以讓 Project Metadata 底下的選項保持原樣，不會有問題，雖然我們會在未來的專案中修改它們。

至於現在，我們也不引入任何依存關係。如此一來，我們就可以專注於專案創建的機制，而非任何特定的結果。

不過，在生成該專案之前，我還想指出 Spring Initializr 有幾個非常不錯的功能，連同一個旁註。

如果想在專案根據你當前的選擇產生專案之前，檢視你專案的詮釋資料（metadata）和依存關係之細節，你可以點擊 Explore 按鈕，或使用鍵盤快速鍵 Ctrl+Space，開啟 Spring Initializr 的 Project Explorer（如圖 2-3 所示）。然後 Initializr 將向你展示專案的結構和建置檔，這些將包含在你即將下載的壓縮（*.zip*）過的專案中。你可以查看目錄 / 套件（directory/package）的結構、應用程式特性檔（application properties file，後面會有更多介紹），以及你建置檔中指定的專案特性和依存關係：因為我們在這個專案中使用 Maven，所以我們的會是 *pom.xml*。

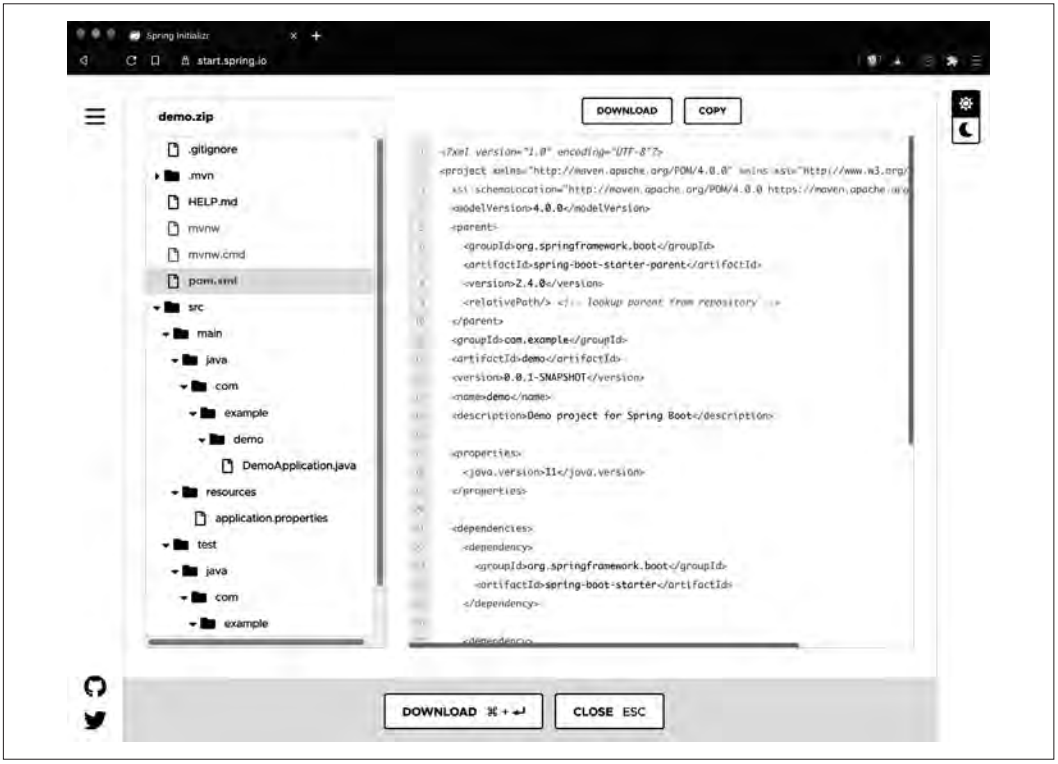

圖 2-3　Spring Initializr 的 Project Explorer

這是在下載、解壓並將全新的空專案載入到 IDE 之前，驗證專案組態和依存關係的一種快速且便利的方法。

Spring Initializr 另一個較小但受到眾多開發者歡迎的功能，就是黑暗模式（dark mode）。藉由點擊頁面頂部的 `Dark UI` 進行切換，如圖 2-4 所示，你就可以切換到 Initializr 的黑暗模式，並使其成為每次你訪問該頁面時的預設模式。這是一個小功能，但如果你的機器在其他地方都保持黑暗模式，它肯定會使 Initializr 的載入不那麼刺眼，更令人愉快。你會想要一直回來的！

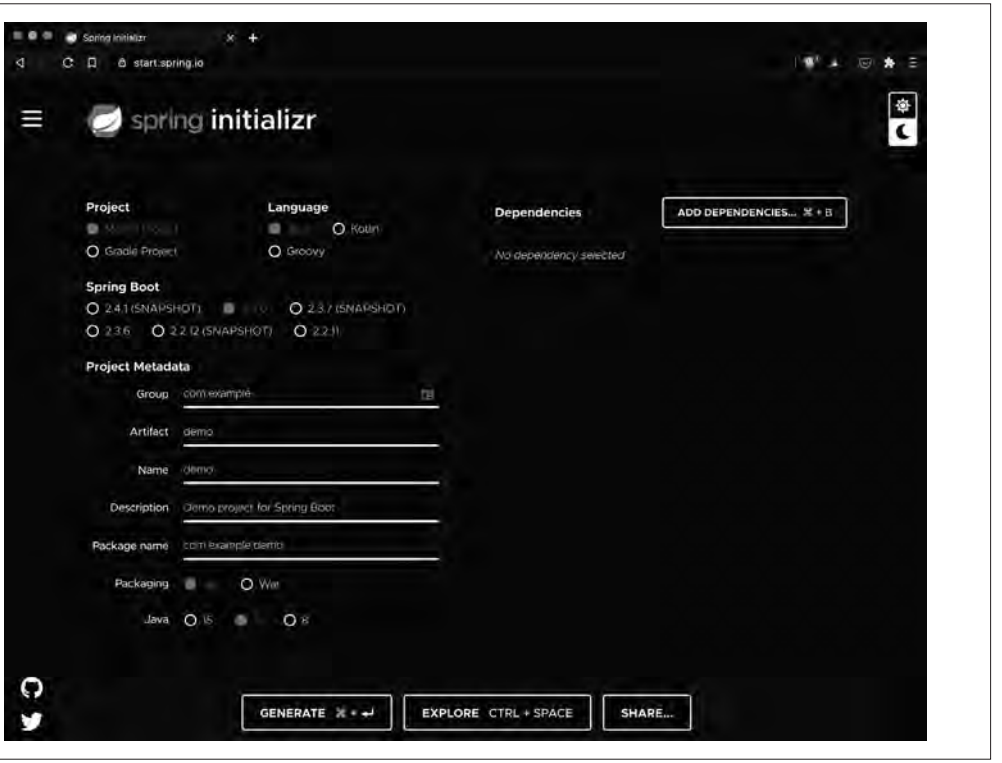

圖 2-4　Spring Initializr，黑暗模式！

除了主應用程式類別（main application class）和它的主方法（main method），再加上一個空的測試（empty test），Spring Initializr 不會為你產生程式碼，它是依據你的指引，為你生成專案（*project*）。這是一個很小但卻非常重要的區別：程式碼生成的結果差異非常大，而且往往在你開始修改的那一刻起就束縛了你。藉由產生專案結構，包括帶有指定依存關係（dependencies）的建置檔（build file），Initializr 為你提供了一個有利的起跑點，讓你編寫得以運用 Spring Boot 自動組態（autoconfiguration）功能所需的程式碼。自動組態賦予了你沒有束縛的超能力。

接下來，點擊 Generate 按鈕來產生、打包並下載你的專案，將其保存到你在本地機器上所選的位置，然後前往那個下載的 *.zip* 檔案並將之解壓縮，準備開發你的應用程式。

Straight Outta 命令列

如果你樂於花費盡可能多的時間在命令列上，或者希望最終用指令稿（script）創建專案，Spring Boot Command Line Interface（CLI，命令列介面）就是專為你設計的。Spring Boot CLI 有許多強大的功能，但現在，我們會把重點放在一個新的 Boot 專案的創建上。

安裝 Spring Boot CLI

也許安裝 Spring Boot CLI 最簡單的方法，就是透過 SDKMAN!，就像安裝 JDK、Kotlin 工具程式等一樣。在終端機視窗中，你可以執行

```
sdk list
```

以查看可供安裝的各種套件。圖 2-5 顯示了 Spring Boot CLI 的進入點。接下來，執行

```
sdk list springboot
```

來查看可用的 Spring Boot CLI 版本，然後用以下方法安裝（目前）最新的版本

```
sdk install springboot
```

如果你在執行 SDKMAN! 命令 `install <tool> <version_identifier>` 的時候沒有提供特定的版本識別字（version identifier），SDKMAN! 通常會安裝該語言 / 工具（language/tool）推薦的最新正式版本。這對不同的支援套件有不同的含義，舉例來說，會安裝的是 Java 最新的 Long Term Support（LTS）版本，而非可用的較新的（非 LTS）版本。這是因為 Java 新編號的版本每六個月發佈一次，並且定期將一個版本指定為 LTS 版本，這意味著受到官方支援的通常會有一或多個較新的版本，每個版本只支援六個月（用於功能評估、測試，甚或生產部署），同時還有一個特定的 LTS 版本會有更新和錯誤修復的完整支援。

這個備註有點泛泛而談，因為不同的 JDK 供應商之間可能會有一定的差異，儘管大多數都不會偏離（如果有的話）習慣性的指定方式。整個討論都專注在細節上，但對於我們在此的目的而言，這沒有任何影響。

```
Spring Boot (2.4.0)                    http://projects.spring.io/spring-boot/

Spring Boot takes an opinionated view of building production-ready Spring
applications. It favors convention over configuration and is designed to get you
up and running as quickly as possible.

                                                    $ sdk install springboot
```

圖 2-5　SDKMAN! 之上的 Spring Boot CLI

一旦安裝了 Spring Boot CLI，你就能用下面的命令建立我們剛剛創建的同一個專案：

```
spring init
```

要將壓縮後的專案解壓縮到名為 *demo* 的目錄中，可以執行以下命令：

```
unzip demo.zip -d demo
```

等等，怎麼可能如此簡單？一句話：預設值。Spring CLI 使用與 Spring Initializr（Maven、Java 等）相同的預設組態，讓你只為你想改變的值提供引數。讓我們特地為其中的一些預設值提供指定值（並為專案的解壓縮添加一個有用步驟），以便更好地瞭解其中所涉及的東西：

```
spring init -a demo -l java --build maven demo
```

我們仍然使用 Spring CLI 初始化一個專案，但現在我們提供下列引數：

- `-a demo`（或 `--artifactId demo`）允許我們為專案提供一個人為的 ID，在本例中，我們稱它為「demo」。
- `-l java`（或 `--language java`）讓我們指定 Java、Kotlin 或 Groovy[1] 作為這個專案的主要語言。
- `--build` 是建置系統引數的旗標（flag），有效的值有 `maven` 與 `gradle`。
- `-x demo` 請求 CLI 解壓縮 Initializr 回傳的結果專案之 *.zip* 檔案。注意這個 `-x` 是選擇性的，指定一個沒有延伸檔名（extension）的文字標籤（就像我們在此所做的那樣）會被推斷為一個解壓縮目錄。

所有的這些選項都可以在命令列執行 `spring help init` 以進一步檢視。

在指定依存關係時，事情會變得更加複雜一點。正如你所想像的那樣，從 Spring Initializr 所提供的「選單」中進行挑選就好所帶來的簡便性，是很難抵抗的。但 Spring CLI 的彈性在快速啟動、指令稿編寫和管線的建置方面是非常有利的。

還有一點：預設情況下，CLI 運用 Initializr 來提供它的專案建置功能，這意味著，藉由這任一種機制（CLI 或 Initializr 網頁）所創建的專案都是完全相同的。這種一致性對於直接使用 Spring Initializr 功能的店家來說是絕對必要的。

不過，偶爾一個組織會施加嚴格的控管，限制他們開發人員能用來創建專案的依存關係有哪些。很老實的說，這種做法讓我很難過，感覺非常有時效性，阻礙了組織的敏捷性和使用者 / 市場（user/market）的反應能力。如果你在這樣的組織中，這會使你「完成工作（get the job done）」的能力複雜化，妨礙你達成想要完成的任何任務。

在這種情況下，你可以建立自己的專案產生器（甚或複製 Spring Initializr 的儲存庫），並透過結果網頁直接使用它……或只對外開放 REST API 的部分，並在 Spring CLI 中運用它。要做到這一點，只需將此參數添加到前面顯示的命令中（當然，用你的有效 URL 替換上去）：

```
--target https://insert.your.url.here.org
```

1 Spring Boot 中仍然提供對 Groovy 的支援，但遠不如 Java 或 Kotlin 被廣泛使用。

留在 IDE 中

無論你是如何創建 Spring Boot 專案的，你都會需要開啟它，並編寫一些程式碼以建立出一個有用的應用程式。

有三個主要的 IDE 和眾多的文字編輯器（text editors）可以為你的開發提供支援。這些 IDE 包括（但不限於）Apache NetBeans（*https://netbeans.apache.org*）、Eclipse（*https://www.eclipse.org*）和 IntelliJ IDEA（*https://www.jetbrains.com/idea*）。這三種都是開源軟體（open source software，OSS），而且在很多情況下都是免費的[2]。

在本書中，和我的日常生活一樣，主要使用 IntelliJ Ultimate Edition。選擇 IDE 的時候，其實並沒有什麼錯誤的選擇可言，而只是個人的偏好（或組織的命令或偏好），所以請使用最適合你和你品味的 IDE。大多數的概念在主要的選擇之間都能轉移得很好。

也有幾個編輯器在開發人員中獲得了大量的追隨者。有些像是 Sublime Text（*https://www.sublimetext.com*），是付費的應用程式，基於其品質和壽命，有很多的追隨者。其他較新進入該領域的應用程式，例如 Atom（*https://atom.io*，由 GitHub 創建，現歸 Microsoft 所有）和 Visual Studio Code（*https://code.visualstudio.com*，簡稱 VSCode，由 Microsoft 創建），正在迅速增加功能和忠實的追隨者。

在本書中，我偶爾會使用 VSCode 或從相同源碼庫建置出來，但禁用遙測 / 追蹤（telemetry/tracking）的對應軟體 VSCodium（*https://vscodium.com*）。為了支援大多數開發人員期望或需要從他們的開發環境中獲得的一些功能，我為 VSCode/VSCodium 新增了下列擴充功能：

Spring Boot Extension Pack（*Pivotal*）（*https://oreil.ly/SBExtPack*）

> 這包含其他一些擴充功能，例如 `Spring Initializr Java Support`、`Spring Boot Tools` 以及 `Spring Boot Dashboard`，它們分別便於在 VSCode 內創建、編輯和管理 Spring Boot 應用程式。

Debugger for Java（*Microsoft*）（*https://oreil.ly/DebuggerJava*）

> Spring Boot Dashboard 的依存關係。

2 它們有兩種選擇：Community Edition（CE，社群版）和 Ultimate Edition（UE，終極版）。Community Edition 支援 Java 和 Kotlin 的 app 開發，但要獲得所有可用的 Spring 支援，必須使用 Ultimate Edition。某些用例有資格獲得 UE 的免費授權，當然你也可以進行購買。此外，三者都為 Spring Boot 應用程式提供了出色的支援。

IntelliJ IDEA Keybindings（*Keisuke Kato*）（*https://oreil.ly/IntellijIDEAKeys*）

因為我主要使用 IntelliJ，這讓我更容易在兩者之間切換。

Java™ 的語言支援（*Red Hat*）（*https://oreil.ly/JavaLangSupport*）

Spring Boot Tools 的依存關係。

Maven for Java（*Microsoft*）（*https://oreil.ly/MavenJava*）

方便使用基於 Maven 的專案。

你可能會發現還有其他一些擴充功能對處理 XML、Docker 或其他輔助技術很有幫助，但就我們目前的用途而言，這些就是基本的要素了。

接續我們的 Spring Boot 專案，接下來你要在所選的 IDE 或文字編輯器中開啟它。對於本書中的大多數例子，我們將使用 IntelliJ IDEA，這是由 JetBrains 公司開發的非常強大的 IDE（以 Java 和 Kotlin 編寫）。如果你已經將 IDE 與專案建置檔關聯起來了，你可以雙擊專案目錄底下的 `pom.xml` 檔（在 Mac 上使用 Finder；在 Windows 上使用檔案總管，或在 Linux 上使用各種檔案管理員），然後自動將專案載入到 IDE 中。如果沒有，你可以在你的 IDE 或編輯器中，以其開發者推薦的方式打開專案。

許多 IDE 和編輯器都提供某種方式建立啟動和載入用的命令列捷徑，只要一道簡短的命令就能帶出你的專案。例如 IntelliJ 的 `idea`、VSCode/VSCodium 的 `code`，以及 Atom 的 `atom` 捷徑。

巡覽 main()

現在我們已經在 IDE（或編輯器）中載入了專案，讓我們看看是什麼讓 Spring Boot 專案（圖 2-6）有別於標準的 Java 應用程式。

一個標準的 Java 應用程式（預設情況下）包含一個空的 `public static void main` 方法。當我們執行一個 Java 應用程式，JVM 會搜索這個方法作為該程式的起始點，若沒有這個方法，應用程式會啟動失敗，出現類似這樣的錯誤：

```
Error:
Main method not found in class PlainJavaApp, please define the main method as:
        public static void main(String[] args)
or a JavaFX application class must extend javafx.application.Application
```

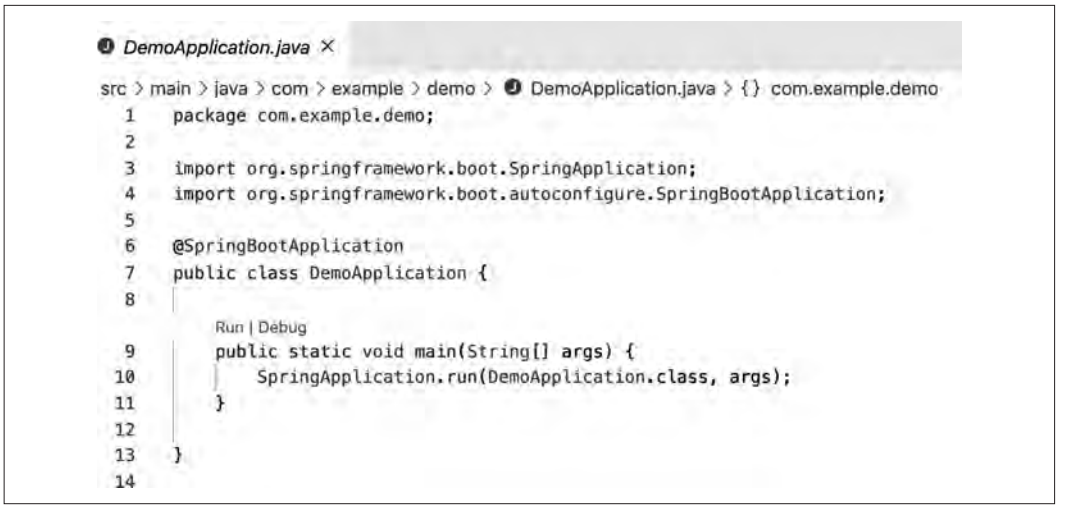

圖 2-6　我們 Spring Boot 的 demo 應用程式的主應用程式類別

當然，你可以把應用程式啟動時要執行的程式碼放在一個 Java 類別的主方法（main method）中，Spring Boot 應用程式正是這樣做的。啟動時，一個 Spring Boot app 會檢查環境、設定應用程式的組態、創建初始情境（context），然後執行 Spring Boot 應用程式。它透過單一個頂層注釋（top-level annotation）和單行程式碼來完成這些工作，如圖 2-7 所示。

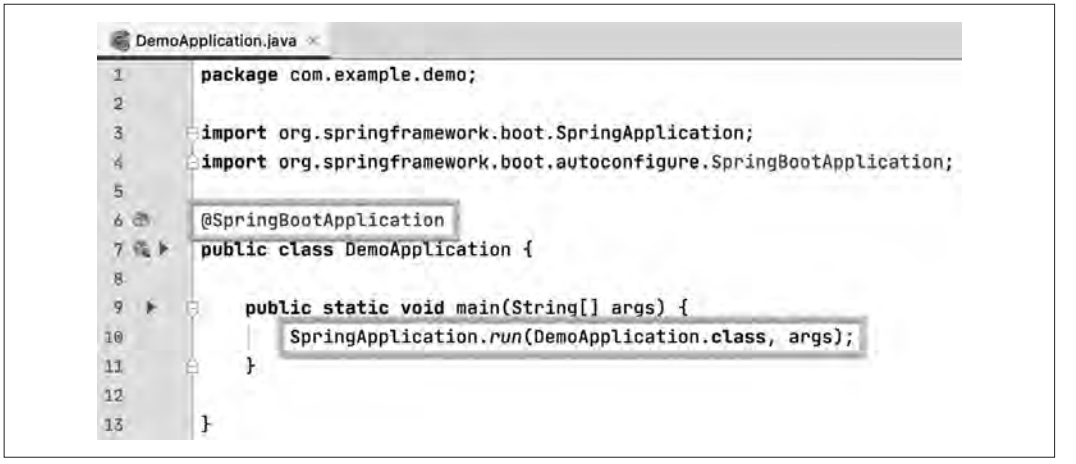

圖 2-7　Spring Boot 應用程式的要素

隨著本書的開展，我們將深入瞭解這些機制。現在只需說，Boot **在設計上**（*by design*）而且**在預設情況下**（*by default*）就能讓我們在應用程式啟動過程中，免除很多繁瑣的應用程式設定工作，這樣你就可以迅速著手撰寫有意義的程式碼。

總結

本章檢視了你在創建 Spring Boot 應用程式時的一些絕佳選擇。無論你是喜歡使用 Maven 或 Gradle 建置專案，還是使用 Java 或 Kotlin 編寫程式碼，或者透過 Spring Initializr 或其命令列夥伴 Spring Boot CLI 提供的 Web 介面建立專案，你都能夠運用 Spring Boot 的完整功能，並享受其易用性，絲毫無須任何妥協。你還可以使用具有頂級 Spring Boot 支援的各種 IDE 和文字編輯器來處理 Boot 專案。

正如這裡和第 1 章所介紹的那樣，Spring Initializr 為你努力工作，讓你的專案得以快速、輕鬆地創建。Spring Boot 透過以下功能在整個開發生命週期中做出了有意義的貢獻：

- 簡化的依存性管理，從專案的創建到開發及維護，都有它的身影。
- 自動組態（autoconfiguration）大大減少或消除你在處理問題領域本身之前可能需要寫的樣板程式碼（boilerplate）。
- 簡化部署，使打包和部署變得輕而易舉。

而且無論你在途中選擇了何種建置系統、語言或工具鏈，所有這些功能都是完全受支援的。這是彈性驚人且強大的組合。

在下一章中，我們將創建第一個真正有意義的 Spring Boot 應用程式：提供 REST API 的一個 app。