前言

當您從書局的書架上拿起了這本書，或是在朋友那聽了一些介紹決定買下這本書，不論您是因為工作上的需求，還是想要讓自己的想法、創作能夠快速地向全世界展示，我相信這都代表了您開始相信學會 iOS App 設計能夠幫助自己完成某些夢想。從 2008 年中 App 誕生至今，App 的開發無疑已經是程式設計領域的顯學之地，已經有無數的工程師或學生投入 App 程式設計領域。當大多數人手上都有一隻手機的時候，沒有什麼管道是比透過一個 App 能更快速的連結使用者各項事物了。

想要學習 iOS 程式設計，必須要有一台 Mac 電腦，因為 iOS 的開發環境 Xcode 必須在 Mac 上執行。除了 Mac 電腦外，您應該還要有一隻 iPhone 或 iPad，因為有很多功能是模擬器無法模擬的，例如拍照、各種感測元件、藍牙，這些都必須在實機上面才能使用。雖然這些設備添購起來花費的成本不小，但先摒除 iPhone 不談，Mac 電腦除了執行 macOS 外，還可安裝 Windows，此外，macOS 的核心為 UNIX，絕大多數的 UNIX 指令在 Mac 上都可以執行。一機三用，很值得。當然，iOS App 工程師的職缺與收入整體來說是很不錯的。

準備要開始大展身手了嗎？首先，到 https://developer.apple.com 開發者網站註冊一個開發者帳號，先不用付錢，使用免費方案即可。等到某天您想要讓您偉大的 App 上架到全世界都能夠下載，再付錢成為付費開發者。接下來到 App Store 下載 Xcode 開發者工具，執行後在選單「Xcode」找到「Preferences」，然後在「Accounts」頁面輸入您的開發者帳號。若想要在實機上執行，將 iPhone 或 iPad 接到電腦上即可。

現在，萬事具備，可以開始寫程式了。

開發者帳號有幾種等級，如下表：

	免費開發者	一般開發者	企業	大專院校
實體裝置測試	○	○	○	○
TestFlight 測試		○	○	
程式碼技術支援		○	○	
Ad Hoc 發佈		○	○	
In-House 發佈			○	
客制化 B2B 發佈		○		
App Store 發佈		○		
價格	免費	美金 99/year	美金 299/year	免費
備註	有限度在實體機器上測試	信用卡付費	需提供 D-U-N-S 編號	只提供給合格且具學位授予的高等教育機構

最常見的還是一般開發者。一般開發者在付費時可以選擇「個人」或是「公司」，不同處在於：個人開發者只有一個 Apple ID 帳號能進行開發，如果是公司開發者則不限。企業開發者與一般開發者的差異在於企業開發者有自己的 App Store，所發佈的 App 僅限企業內部使用。換句話說，除企業員工外，其他人在全球的 App Store 上是看不到這個 App 的。若申請公司或是企業開發者方案都需要向 Apple 出示 D-U-N-S 編號（鄧白氏環球編碼）證明申請者是個合法公司或是企業。如果申請一切順利，您很快就會在當初留下的 Email 中收到蘋果公司寄來的信件，跟著信件內容去啟動付費開發者帳號即可完成申請程序。

如何使用本書

本書分成三個部分：

第一篇談 SwiftUI 中的各種排版方式與元件使用。SwiftUI 是一種界面設計方式，透過簡潔的聲明式語法完成介面排版工作。這種界面設計需要靠程式碼來完成各種排版需求，雖然需要的程式碼不多，但以往透過拖放元件以及屬性設定來排版的方式，在 SwiftUI 中變的可有可無。過去我認為 Storyboard 專案比較適合新手，但現在需要重新調整一下這樣的看法，畢竟 SwiftUI 在未來會成為主要開發方式，而且 SwiftUI 所需要的程式碼，整體來說會比 Storyboard 少一點。

第二篇談 SwiftUI 與 Storyboard 的整合。雖然 SwiftUI 會成為開發主流，但現階段 SwiftUI 在某些功能上是不足的。除此之外，還是有很多專案是透過 Storyboard 在開發與維護。若兩者能夠整合，互相截長補短，這樣在 Storyboard 專案中也可以享受 SwiftUI 帶來的快速開發效益，而 SwiftUI 也可以使用 Storyboard 中才有的元件。

第三篇談模型，也就是常聽到的 MVC 或 MVVM 架構中的 Model 部分。所以這一章的內容主要是處理資料，跟介面設計較無太大關係，所以像是檔案存取、資料庫、藍牙...等。

本書以最新版本的 Swift 語法說明各個範例程式，若您對 Swift 語法還不熟悉，請到我架設的研蘋果官方網站點選「Swift 語言全面剖析」，我在這裡完整且有系統地介紹了 Swift 這個語言，網址如下。

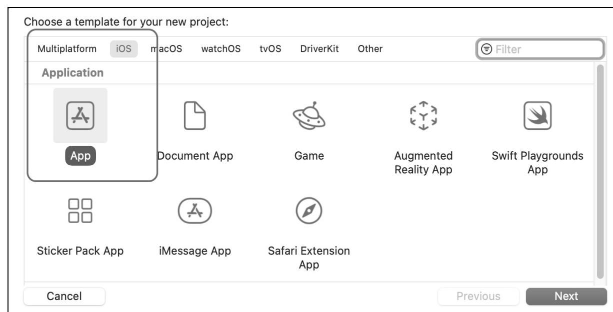
https://www.chainhao.com.tw/swift/

01 Hello SwiftUI

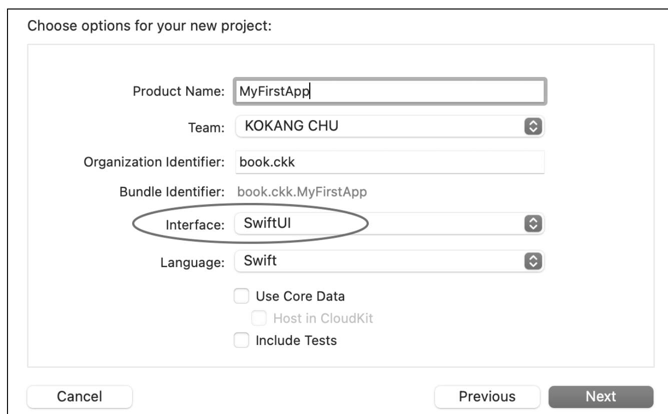
Part 1 SwiftUI

1-1 初體驗

SwiftUI 是一個專門用來設計使用者介面的框架名稱，包含了各式各樣的視覺化元件。SwiftUI 主要目的是在 Apple 的各作業系統間提供了統一的介面設計，也就是用同樣的介面設計與程式碼，就可以編譯出能在 iOS、iPadOS 與 macOS 系統上執行的應用程式。因此，有別於 UIKit 框架的 Storyboard 專案，SwiftUI 的介面設計與程式寫法是完全不同的，連開發介面也完全不同，雖然還是使用 Xcode 來開發，但體驗上完全不一樣。首先開啟 Xcode 後選擇 iOS App 類型的專案。

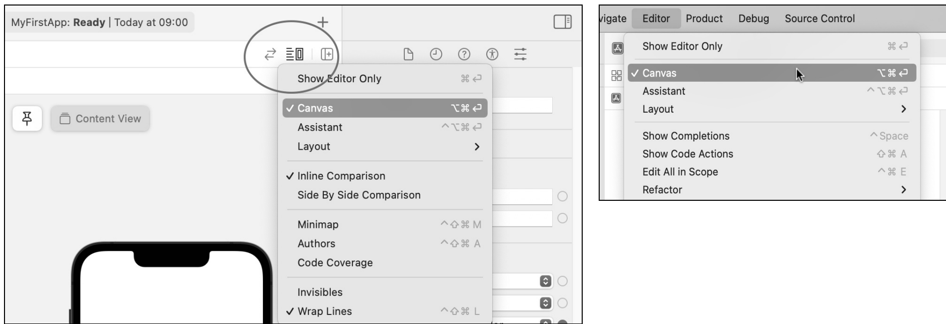


接著輸入專案名稱、Organization Identifier（個人開發的話隨意填，之後也可以改）以及專案類型，這裡要選 SwiftUI，選錯就要重來。



當 SwiftUI 專案建立後，Xcode 會自動產生 ContentView.swift 這個檔案，開啟後可以看到檔案中有兩個結構（struct），上方名稱為 ContentView 的結構是 App 的第一個畫面，而下方名稱為 ContentView_Previews 的結構則用來產生預覽畫面。我們可以使用 Command + Option + Return 這三個熱鍵開啟與關閉畫面預覽功能。下圖右方為預覽開啟後的樣子，可以看到預覽畫面會即時反應 ContentView 目前內容，也就是左方的程式碼只要一改變，右邊的畫面就會立即跟著改變。

想要開關預覽畫面除了使用熱鍵外，也可以在下圖左的圖圈起來圖示上點選 Canvas 選項，或者在下圖右的選單「Editor」中選「Canvas」選項，這三個方式都可以開啟預覽畫面。



現在我們已經知道如何開啟與關閉預覽畫面了。SwiftUI 的預覽功能非常完整，除了能即時反應左邊的程式碼外，也可以在預覽畫面上操作具有互動功能的元件，例如按鈕、開關、文字輸入等。事實上，這個預覽功能相當於把模擬器整合進 Xcode 裡面，所以大部分時候我們都不需要再啟動模擬器才能檢查畫面設計是否適當。

我們先把目標放在 ContentView 這個 struct 內容上。當 SwiftUI 專案建立後，Xcode 14 在 body 變數中預先產生的程式碼如下，不同 Xcode 版本預先產生的程式碼會不同，但都不影響 App 開發，因為 body 中的程式碼最後都會刪除。

```
struct ContentView: View {
    var body: some View {
        VStack {
            Image(systemName: "globe")
                .imageScale(.large)
                .foregroundColor(.accentColor)
            Text("Hello, world!")
        }
    }
}
```

首先，這個結構符合了 View 協定的規範。在 SwiftUI 架構中，所有要顯示在螢幕上的元件都必須符合 View 協定。也就是說我們可以自己設計一個奇形怪狀的按鈕，只要他能符合 View 協定，那這個奇形怪狀的按鈕就可以顯示在螢幕上。結構中有一個變數 body，資料型態為 some View，後面接了一個 Closure 區段。Closure 中的程式碼其實就是這個畫面的程式進入點，也就是第一行程式會從 Closure 中開始執行。保留字 some 稱為不透明型態（Opaque Type），所以 some View 代表某種類型的 View（例如 Button、Text、Image...等），所以後面的 Closure 會回傳某一種 View，至於是哪一種並不重要，只要符合 View 協定的元件就可以了，在 App 開發過程中，其實我們不需要特別瞭解不透明型態是什麼才能開發，所以如果搞不懂不透明型態，也不用太擔心。如果有興趣瞭解的讀者，可以來下列網站參考，這是本書作者在網路上發佈的不透明型態文章。

https://www.chainhao.com.tw/15-不透明型態/

到這裡稍微總結一下，在 SwiftUI 架構中，如果要自行設計一個能顯示在螢幕上的畫面，基本語法如下。

```
struct MyView: View {
    var body: some View {
        // 回傳某個 View 元件
    }
}
```

接下來把重點放在 body 的內容上，不同版本的 Xcode 產生的程式碼可能會不一樣，先前版本只會產生一個 Text 元件，並且顯示一個 Hello, World! 字串，而目前 Xcode 14 產生的程式碼如下，有圖片也有文字。

```
VStack {
    Image(systemName: "globe")
        .imageScale(.large)
        .foregroundColor(.accentColor)
    Text("Hello, world!")
}
```

這段程式碼用了三個符合 View 協定的元件（也可以稱為 View 元件或視圖元件），他們分別是 VStack、Image 與 Text。VStack 是排版專用元件，會將 Image 與 Text 這兩個元件以垂直方向排列，我們可以試試改成 HStack 元件，會發現預覽畫面中的排版方式立刻變成水平排列。另外有沒有發現 VStack 也是用 Closure 語法將需要排版的元件放在 Closure 區段裡面，事實上，SwiftUI 使用了極大量的 Closure 語法，到處都可以看到其蹤影。對 Closure 語法不熟悉的讀者，請參考本書作者公布的網路文件，裡面對 Closure 語法有詳細的說明。

https://www.chainhao.com.tw/7-closure/

第二個 Image 元件用來顯示一張圖片，圖片內容來自於內建的 SF Symbols 圖庫，這是 Apple 官方圖庫，建議安裝 SF Symbols App 方便搜尋所有圖片，網址為 https://developer.apple.com/sf-symbols/。Image 元件的初始化器後面帶了兩個函數，分別是 imageScale 與 foregroundColor，這兩個函數在 SwiftUI 中稱為修飾器（Modifier），作用是修改 Image 元件的預設值。修飾器 imageScale 用來將圖片放大一點，而 foregroundColor 看名字就知道是用來修改圖片顏色。最後一個元件 Text，純粹顯示一個字串到螢幕上，這個元件相當於 Storyboard 中的標籤元件 UILabel（定義在 UIKit 框架中），在 SwiftUI 中改名為 Text。

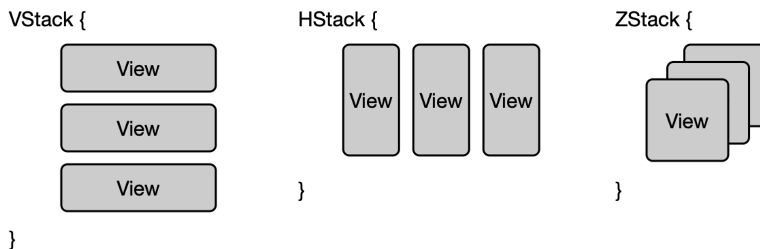
到這裡再總結一下兩個重點：（一）變數 body 後面的 Closure 區段只能回傳一個 View 元件，如果有兩個以上的 View 元件都要顯在畫面上，應該使用排版元件或是容器元件將他們合併成一個；（二）修改 View 元件的預設值，必須在該元件初始化後使用修飾器，而不同的修飾器有不同的功能。

02 排版元件與技巧

Part 1 SwiftUI

2-1 堆疊佈局

在 SwiftUI 中共有三種類型的堆疊元件可以讓我們排出想要的版型，這三種堆疊元件分別是 VStack（包含 LazyVStack）、HStack（包含 LazyHStack）與 ZStack。VStack 是將包含在其中的元件以垂直方向排列；HStack 則是水平方向；ZStack 是讓各元件在視線的深度上排列，也就是放在 ZStack 中第一個元件會被壓在最底層，最後一個元件會在最上層。三種堆疊元件造成的效果如下圖。



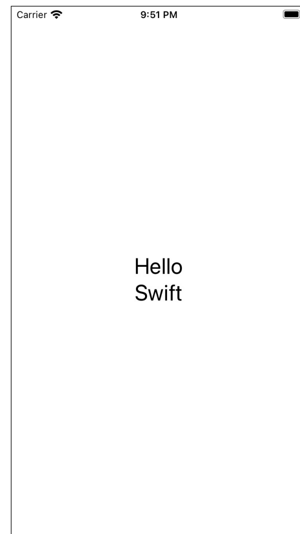
VStack 與 HStack 各有一個姊妹元件，分別是 LazyVStack 與 LazyHStack，功能與 VStack 與 HStack 一樣，但加上 Lazy 的這兩個元件在配合 ScrollView 元件一起使用時具備記憶體管理功能，稍後會解釋這兩個元件的細節內容。

如果我們要排出一個比較複雜的版面，只要適當地組合這幾個堆疊元件就可以產生非常多樣化的排版效果。接下來先舉 VStack 與 HStack 幾個例子，熟悉這兩個元件的使用方式就已經可以完成大多數的排版需求了，ZStack 留到稍後的單元與 overlay 修飾器一起介紹。

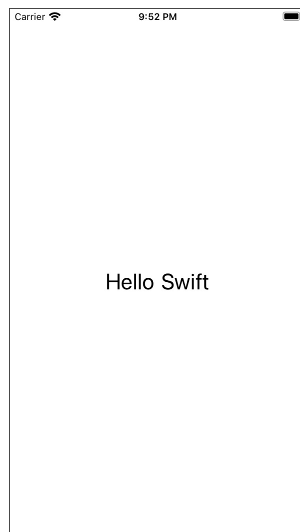
VStack 與 HStack 元件

如果要將多個元件以垂直方向或水平方向排列時需要使用 VStack 與 HStack 這兩個元件，其中 V 是 Vertical 的縮寫，而 H 表示 Horizontal，如下。

```
var body: some View {
    VStack {
        Text("Hello")
        Text("Swift")
    }
}
```



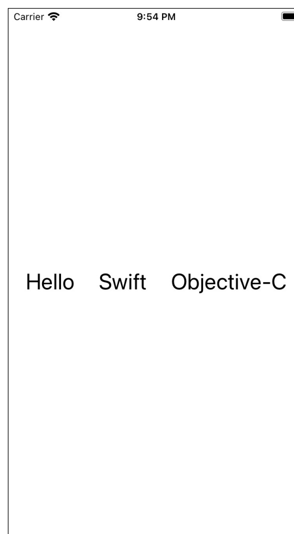

```
var body: some View {
    HStack {
        Text("Hello")
        Text("Swift")
    }
}
```



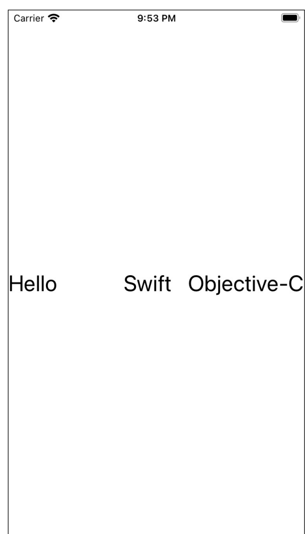
設定元件間距

若要設定堆疊中的元件間距，有兩種作法，一種是在堆疊元件中加上 spacing 參數，另外一種是使用 Spacer 元件，下面以 HStack 為例，讓元件與元件間的間距為 30pt，同樣的參數也適用在 VStack 上。

```
var body: some View {
    // 三個元件間的間距為 30pt
    HStack(spacing: 30) {
        Text("Hello")
        Text("Swift")
        Text("Objective-C")
    }
}
```

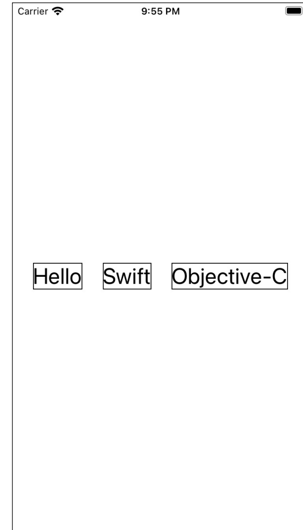



```
var body: some View {
    // 第二與第三個元件間距為 20pt
    // 剩下空間給第一與第二個元件
    HStack {
        Text("Hello")
        Spacer()
        Text("Swift")
        Spacer().frame(width: 20)
        Text("Objective-C")
    }
}
```



下面這個使用 Spacer 元件的例子，會水平均分各元件間的間距，包含最旁邊的元件與螢幕邊緣的距離。範例程式中特別替 Text 元件加上邊線，方便觀察 Text 在排版後的範圍。

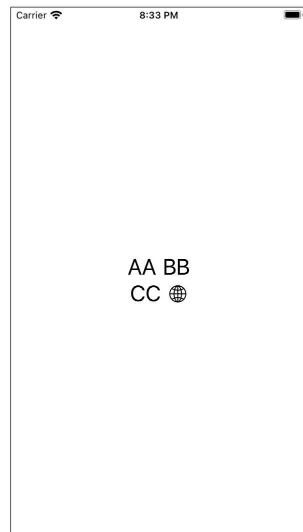
```
var body: some View {
    // 水平均分
    HStack {
        Spacer()
        Text("Hello").border(.black)
        Spacer()
        Text("Swift").border(.black)
        Spacer()
        Text("Objective-C").border(.black)
        Spacer()
    }
}
```



組合應用

VStack 與 HStack 可以互相疊套，透過這個方式就可以排出非常多樣化的版面。例如將四個元件排程田字形，作法如下。

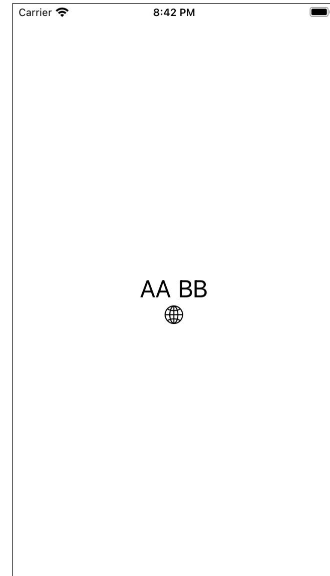
```
var body: some View {
    VStack {
        HStack {
            Text("AA")
            Text("BB")
        }

        HStack {
            Text("CC")
            Image(systemName: "globe")
        }
    }
}
```



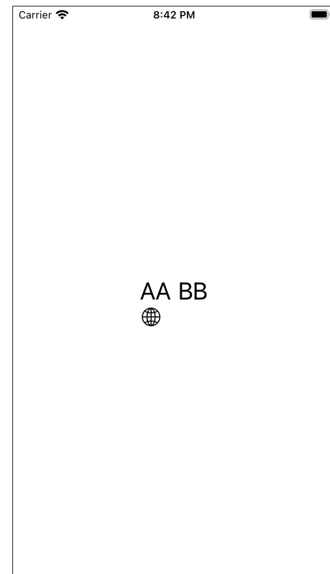
對齊

VStack 與 HStack 中的元件，預設對齊方式都是置中對齊，加上 alignment 參數就可以改變對齊方式。下面以 VStack 為例，說明有加 alignment 參數與沒有加的區別，注意地球圖案的位置不一樣。

```
var body: some View {
    VStack {
        HStack {
            Text("AA")
            Text("BB")
        }

        Image(systemName: "globe")
    }
}
```




```
var body: some View {
    VStack(alignment: .leading) {
        HStack {
            Text("AA")
            Text("BB")
        }

        Image(systemName: "globe")
    }
}
```



07 圖表

Part 1 SwiftUI

7-1 說明

圖表是一個非常重要的資料顯示方式，但過去 Xcode 一直沒有提供跟圖表有關的框架，所以在 App 中要繪製圖表就變成要使用第三方函數庫或是自己使用繪圖函數計算座標畫出需要的長條圖、折線圖這類的圖形，是很辛苦的一件工作。但從 Xcode 14 開始，Apple 終於重視圖表在 App 中的份量，提供了官方的圖表框架，名稱為 Charts。

有了 Charts 框架，現在要畫長條圖、折線圖、點圖...等一些常見的圖形就非常容易了。雖然這個框架可以呈現的圖表種類還不算太多，但未來一定會有更豐富的圖表類型出現，畢竟圖比原始資料更容易理解數據背後代表的意義。

Charts 框架目前提供了六種圖形：長條圖、折線圖、點形圖、基準線、面積圖與矩形圖。透過參數設定與組合應用，還可呈現更多種類的圖，例如熱區圖、箱型圖...等。透過 Charts 框架繪製圖表，程式架構很簡單，各種參數設定也很容易，整體來看，這是一個簡單易上手的框架，對於經常需要在 App 中繪製圖表的開發人員，一定會喜歡這個框架帶來的效益。

從 Charts 框架的設計方式來看，這個框架主要是使用在 SwiftUI 介面上，若想要在 Storyboard 的 View Controller 中使用，建議的作法是透過 UIKit 的 UIHostingController 元件來讓 View Controller 中嵌入 SwiftUI 所繪出來的圖表，這樣做程式碼少又簡單應該是比較適當的作法，請參考「Storyboard 載入 SwiftUI View」章節，有詳盡解說如何操作。

7-2 長條圖

這個單元先介紹長條圖，這是一個非常常見的圖表類型，透過這個單元就可以理解圖表框架的基本操作邏輯與使用方式。

首先在 SwiftUI 專案中新增一個 SwiftUI View 類型的檔案，檔名可以任意，這裡取名 BarChart.swift。不新增這個檔案也可以，那就是把長條圖相關的程式碼全部放在預設的 ContentView.swift 中。

在 BarChart.swift 匯入 Charts 框架，並且定義一個存放商品資訊的結構。這個結構建議符合 Identifiable 協定，所以要實作 id（變數或常數都可以），這樣之後在畫圖時可以不需要透過 ForEach 迴圈來跑每一筆資料。其他三個屬性分別為 name 表示商品名稱，count 表示商品數量，color 作為長條圖上設定各長條的顏色，不自己指定顏色也沒關係，Charts 會提供預設的顏色。

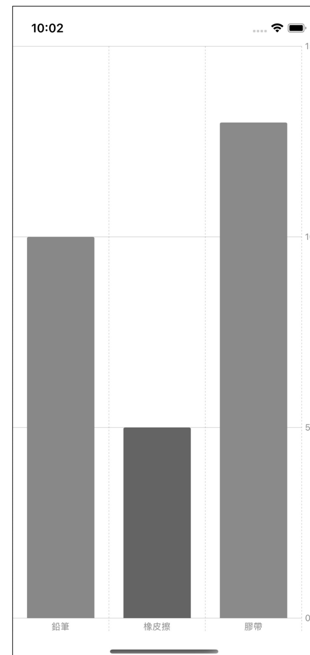
```
import Charts

struct Product: Identifiable {
    var id = UUID()
    var name: String
    var count: Int
    var color: Color
}
```

有了上面這個結構後，我們就可以產生一些商品資料放在這個結構中。實務上，這些資料應該從資料庫中取得，但這裡模擬一下。

```
var products: [Product] = [
    .init(name: "鉛筆", count: 10, color: .brown),
    .init(name: "橡皮擦", count: 5, color: .indigo),
    .init(name: "膠帶", count: 13, color: .mint)
]
```

在 BarChart 結構的 body 屬性中（如果沒有新增 BarChart.swift 的話就在 ContentView 的 body 屬性），透過 Chart 畫出圖表。畫法是在 Chart 的 Closure 中使用 BarMark 畫出長條圖，最後透過修飾器 foregroundStyle 就可以指定顏色，不加這個修飾器顏色會是預設的藍色。

```
var body: some View {
    Chart(products) { item in
        BarMark(
            x: .value("名稱", item.name),
            y: .value("銷售量", item.count)
        )
        .foregroundStyle(item.color)
    }
}
```



如果將 x 軸與 y 軸的內容對調，就會畫出水平方向的長條圖，也稱為橫條圖。

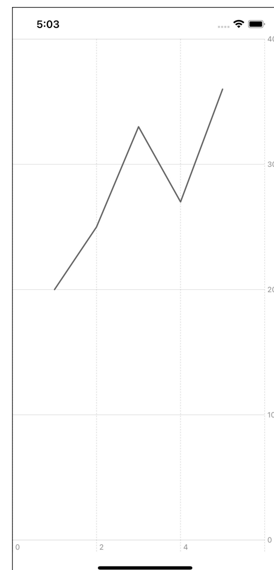
7-5 折線圖、點形圖與基準線

折線圖通常用來表示一個跟時間有關的序列資料變化，先來看一個簡單的折線圖。資料結構與資料定義如下，其中屬性 x 為折線圖的 x 軸資料，屬性 y 為折線圖的 y 軸資料。

```
struct Record: Identifiable {
    var id = UUID()
    var x: Int
    var y: Int
}

var records: [Record] = [
    .init(x: 1, y: 20),
    .init(x: 2, y: 25),
    .init(x: 3, y: 33),
    .init(x: 4, y: 27),
    .init(x: 5, y: 36)
]
```

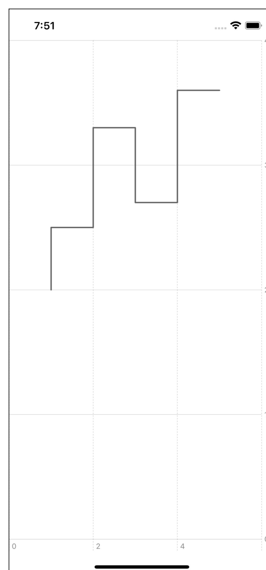
接下來畫出折線圖，程式碼如下。

```
var body: some View {
    Chart(records) { record in
        LineMark(
            x: .value("X", record.x),
            y: .value("Y", record.y)
        )
    }
}
```



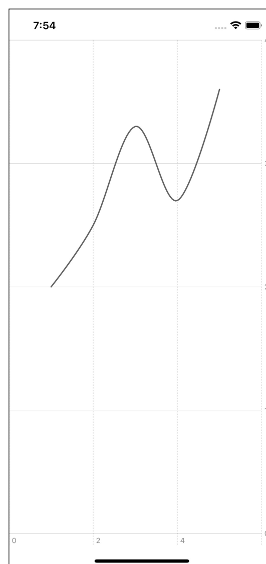
如果加上 interpolationMethod 修飾器，就可以改變折線圖的呈現樣式，例如畫成階梯折線圖。

```
Chart(records) { record in
    LineMark(
        ...
    )
    .interpolationMethod(.stepStart)
}
```



也可以畫成曲線圖，如下。

```
Chart(records) { record in
    LineMark(
        ...
    )
    .interpolationMethod(.cardinal)
}
```



目前支援的折線圖形狀有 cardinal、catmullRom、linear、monotone、stepCenter、stepEnd 與 stepStart 這七種，有興趣的讀者就自行試試看了。

23-4 姿勢偵測

姿勢偵測與人臉偵測幾乎是一樣的程式碼。姿勢偵測用來偵測人體關節處，使用的請求函數為 VNDetectHumanBodyPoseRequest()，運算結果傳回的資料型態為 VNHumanBodyPoseObservation。改寫人臉偵測的 faceDetection()函數為姿勢偵測，如下。

```
private func poseDetection() async {
    let request = VNDetectHumanBodyPoseRequest { request, error in
        guard error == nil else {
            print(error!)
            return
        }

        if let results = request.results as? [VNHumanBodyPoseObservation] {
            self.bodyPoints = results
        }
    }

    do {
        let handler = VNImageRequestHandler(cgImage: uiImage.cgImage!)
        try handler.perform([request])
    } catch {
        print(error)
    }
}
```

辨識結果放入如下的變數中。

```
@State private var bodyPoints: [VNHumanBodyPoseObservation] = []
```

最後在 Canvas 元件中將辨識結果繪出來即可。在 recognizedPoints()中除了 all 代表所有可偵測的點之外，想要得到特定的區域，例如上半身軀幹，可以換成 torso 或是 leftLeg（左腿）。

```
Canvas { context, size in
...
   bodyPoints.forEach { item in
      if let recognizedPoints = try? item.recognizedPoints(.all) {
         recognizedPoints.forEach { body in
            let point = VNImagePointForNormalizedPoint(
               body.value.location, Int(size.width), Int(size.height)
            )
            var path = Path()
            ...
         }
      }
   }
}
```

執行結果如下，左圖為原圖，右圖為姿勢偵測結果。

目前在姿勢上可以偵測到點，共 19 個，如下圖。