第五章　訊息推播

推播是所有 APP 開發者，特別是較年輕，沒有學過 Web 或 PC 上的軟體開發經驗者來說，更是不易達成的任務。因為推播這一件事，對 APP 在市場的影響很大，為了安全，不管是 Android 或 iOS，都需要開發者以特定方式產生憑證或令牌來控制，而且這個憑證或令牌不應放在 APP 中，因為一旦 APP 被反組譯，或網路封包側錄等方式，被取得憑證，你的 APP 使用者就會可能會永無寧日。所以，必需由後台來進行發送，而由於 iOS 與 Android 的機制又不同，若加上其他系統的推播（Windows，Chrome，macOS 等），若再加上給不同系統使用發送用的 API，與分群，大量發送等功能，後台會非常複雜困難。

Firebase Cloud Messaging （以下簡稱 FCM）提供整合了 iOS，Android 與 Web 推播的系統，並且有分群、訂閱等功能，和 GCM 整合的部份甚至憑證都全自動幫你取得了，前台需要的發送 SDK 也都好了，後台開發者只需必要時，把訊息內容提供給 FCM，其他的就交由 FCM 就好了。

推播基本原理

首先讓我們了解一下推播訊息的基本原理。

在每一個平台上的推播方案，雖都有完整的解決方案，如 Apple 的 APNS，Google 的 GCM，或 Microsoft 的 WPNS 等，但只有 PNS （Push Notification Service）是不以完成安全與完整的推播的。最小的狀況下，需要有一個發送器的後台，來執行推播的發送。以 iOS 為例，整個推播發送的過程如下：

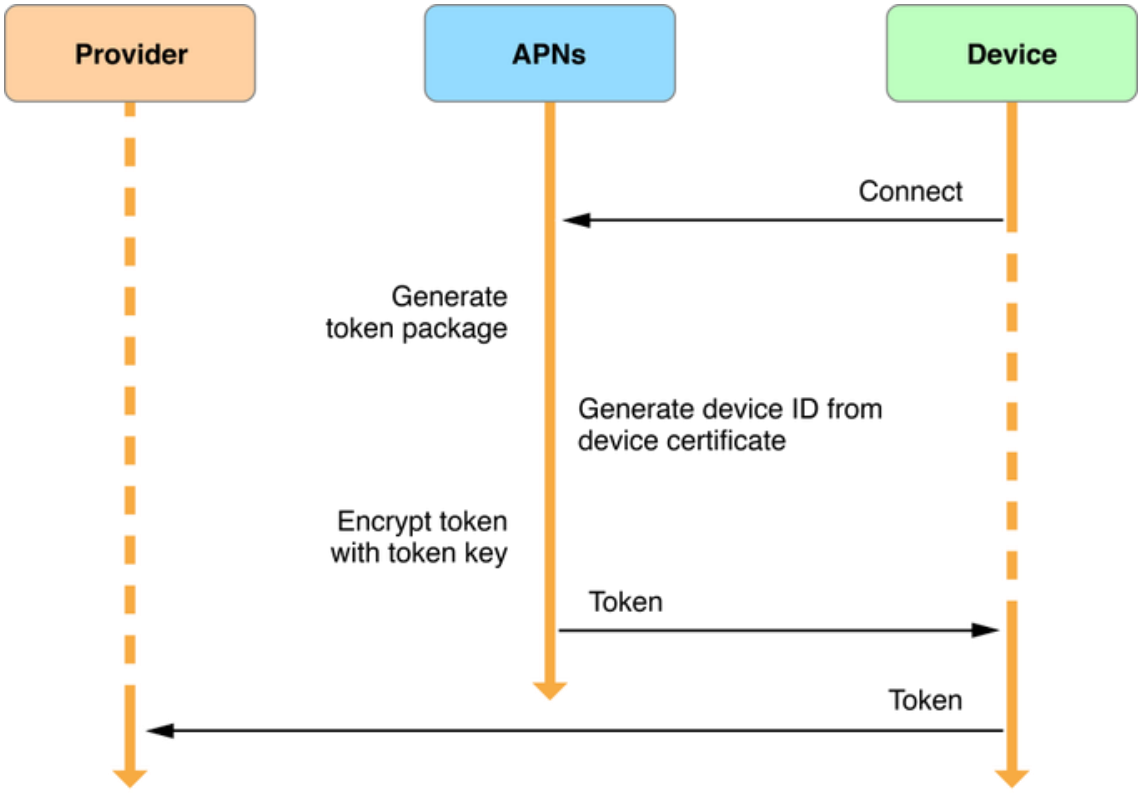
圖片來源：Apple 官方開發者網站

首先，你必需要有一個提供推播者（Provider）來通知 APNS，這個提供推播者至少要提供三個東西，APNS 才會執行通知

1. 安全憑證
2. 目標機器
3. 推播內容

安全憑證都是在建立應用時，各平台依據各自的 ID 所產生的，如 Android 的 Package Name 或 iOS 的 Bundle ID 經過管理平台加密後產生，解密的關鍵也只有平台有，所以平台就能知道該送給那一支 APP，而目標機器是在手機或平板在開啟時與系統加密產生再回送給 Provider 一個 Token，如下圖：

圖片來源：Apple 官方開發者網站 APNS



這只是 iOS 的部份，若你還有 Android 版本，雖然流程類似，但因為方法不完全一樣，所以你還要另外寫一套對應 Android 的後台。

了解基本原理後，我們就要討論 Provider 的角色了。

通常 Provider 有兩個設計的瓶頸，就是安全與效能。由於我們必需把憑證與 Token 存在其中，只要有這兩個東西就能讓手機不停的收到訊息，雖然系統在通

訊方法上，使用了很多加密技術，但若駭客直接進入 Provider 的機器中存取，就有機會讓使用者不停的受干擾，所以在 APP 中直接存放憑證是相當不安全的，任何一個駭客下載 APP ，反組譯之後，就很有機會得到需要的一切。所以放在伺服器中，避免駭客直接接觸到程式才是正確的做法。

另外，推播並沒有廣播的功能，若你的 APP 有數十萬或上百萬的使用者，使用單一個節點，即使網路再順暢，也要數十分鐘到數小時才能發送完畢，若你的訊息是有時效性的，如限時 30 分鐘特賣之類的，那自己發送就一點意義都沒有了。

FCM 除了整合了各平台的推播方式，不用自己架設外，也提供了很大的效能，雖然沒有提供保證，但根據官方的教學影片所述，在數百萬的目標設定後，有 95%的訊息可以在 0.25 秒內送達，這是自己架設後台發送幾乎不可能達成的目標。

發送內容與參數

發送一個推播訊息給 APP，需要一些內容與參數，比較重要的觀念有以下幾個：

1. 訊息內容
   收到時時標題內容，只能顯示十幾個字，太長的內容是無法顯示的，而訊息並沒有絕對可以顯示幾個字，會因機種，作業系統版本。
2. 發送目標
   預計要發送的目標，可以是應用程式 ID，語言，主題或特定的使用者
3. 預計發送時間
   系統可以即時發送，也可以預約在特定時間發送該訊息
4. 音效
   是否發出音效提醒使用者
5. 有效時間
   推播是不保證也無法保證即時送達，因為使用者的手機不一定有開機，開了機也不一定有網路，也有可能使用者已刪除了 APP。這種情形時，各 PNS 就會保留一定的訊息時間，等到使用者上線時再發送，超過時間就會把訊息刪除，這個時間就是訊息送達保留的有效時間，各系統保留的最長時間不一樣，但不會超過四星期的時間，但某些訊息是有時效性的，如會議提醒等，此時就應該自訂有效時間了。

6. 徽章數字(badge number)
   通常用來表示未讀數量，也就是 Apple 的小紅點上的數字，Android 並沒有官方的處理方法，但許多機種有處理的方式，我們通常用統一的方式處理。
7. 訊息標籤
   自訂這則訊息在 Firebase 中的識別名稱，使用者並不會看到
8. 自定欄位
   除了官方要求的欄位外，也可以自訂資料欄位，讓 APP 開啟後，做後續的進一步處理。
9. Android 通知管道
10. Analytics 轉換事件

其中前五項是必要欄位，發時就應提供，其他為選項或有預設值的項目。

必要條件

若要發送訊息，iOS 與 Android 的必要條件有些不同。

iOS 上，你必需要正式的開發者帳號才能產生憑證或 Token，而且目標對象必需是實體機，因為模擬器無法產生接收所需的 Token，所以執行起來也不會有任何反應，但並不會產生錯誤，如果模擬器沒有反應，並不是你的程式有問題。

Android 條件就比較簡單，只要有 Google 帳號與模擬器就可以了，但模擬器必需是 4.2.2 及以上的版本，並必需要有 Google Play Service 的版本。另外，使用者的網路方面，可能必需開通 5228~5230 port 才能完整收發訊息。

實作推播

由於 iOS 與 Android 做法不太一樣，以下會

iOS 基礎實作推播

要在 iOS 上實作推播有兩個必要條件

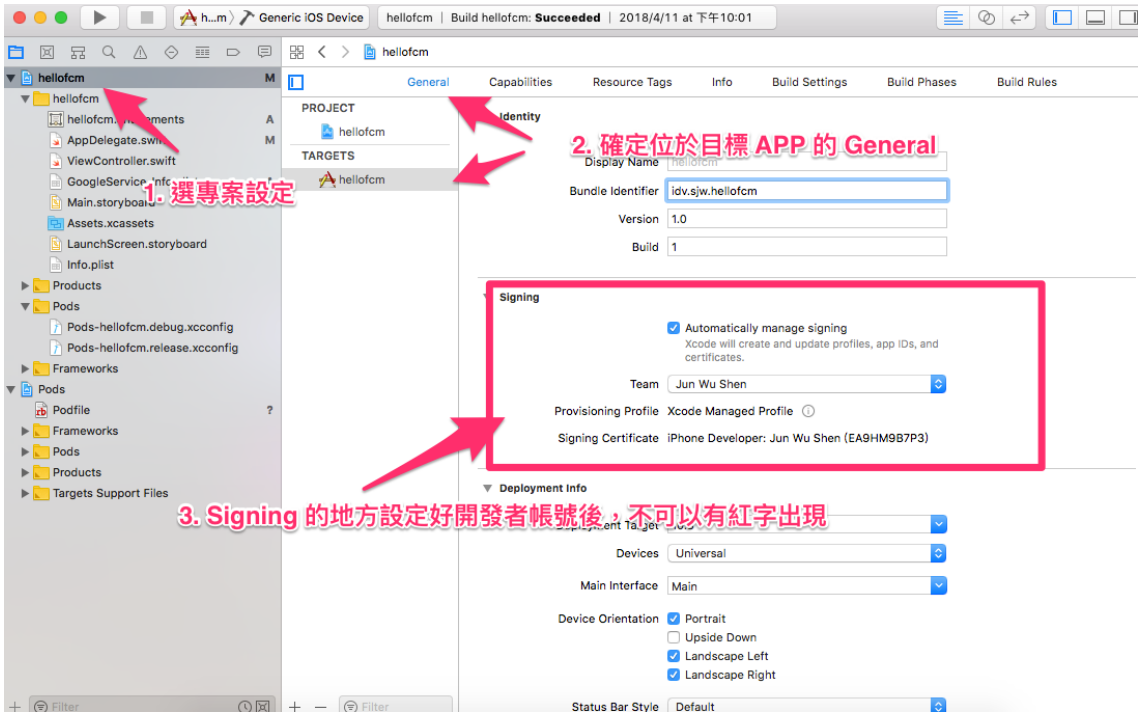
1. 必需要有付費的開發者帳號
2. 必需要有實體可連上網路的 iOS 設備，不能用虛擬機執行實作

帳號的部份，只支援能上架用的帳號，教育帳號是不行的。公司帳號也可以，但是因為要製作憑證，所以你必需要有製作憑證的權限。很不幸的，並沒有任何其他方式可以提供暫時測試，只能花錢解決問題。實體機的部份，只要是能執行 iOS 的實體都可以 iPad / iPhone / iPod Touch 都可以。如果以上兩個條件不存在，你還是可以寫程式，只是寫出來的東西不會有任何反應，也無法驗證是對或錯，所以只能先有概念，未來條件符合後再行測試。

無論如何，我們還是來開始實作吧

建立專案引入 SDK

首先建立一個 iOS 專案，在建立專案時，還有一件可能要注意的項目，就是 Bundle ID 的設定，必需是要通過 Signing 的，因為我們要作推播，就會最後必需實際登錄到 APP 進行實機安裝，並取得推播憑證，所以必需通過 Apple 開發者網站的檢查，但過程有點煩雜，還是一次就認真設定比較好，而要知道是否通過 Singning 只要在專案設定頁確認開發者帳號後，就可知道了。



如上圖，在 Signing 的地方，必需確認你的 Team 帳號，然後 Xcode 就會自動檢查了，如果不通過就會有紅字出現，出現紅字仍可以執行程式，不過卻不能在實體機執行，我們寫的推播也就完全無用武之地了，所以務必在此檢查過，以免做白工，或者需要事後做一堆事修補。

然後就可以如同其他使用 Firebase 的專案一樣，使用 cocoa pods 引入 SDK，或修改 Podfile 再重新執行 pod install。 為了使用 FCM，Podfile 至少需要以下兩個項目。

```
pod 'Firebase/Core'
pod 'Firebase/Messaging'
```

其中 'Firebase/Messaging' 就是 FCM 用的 SDK，安裝完成後開啟 workspace 檔，並且在 Appdelegate 中設好初始化就可以了。

其他要做的項目，我會在後面陸續說明。

建立 APNS 推播憑證

Apple 推播憑證早期都是用證書簽名請求（CSR）到開發者網站上針對指定 APP

申請，然後在開發者本機上產出憑證 .p12 檔，有效期一年，過程有點麻煩，而且每年都要更新，相當的不便。但在 WWDC 2016 之後，Apple 推出了一個新的方法，是以 token 為基礎產生金鑰 .p8 檔的方式，不只沒有憑證過期的問題，而且在同一帳號下的所有 APP 都適用，非常方便。

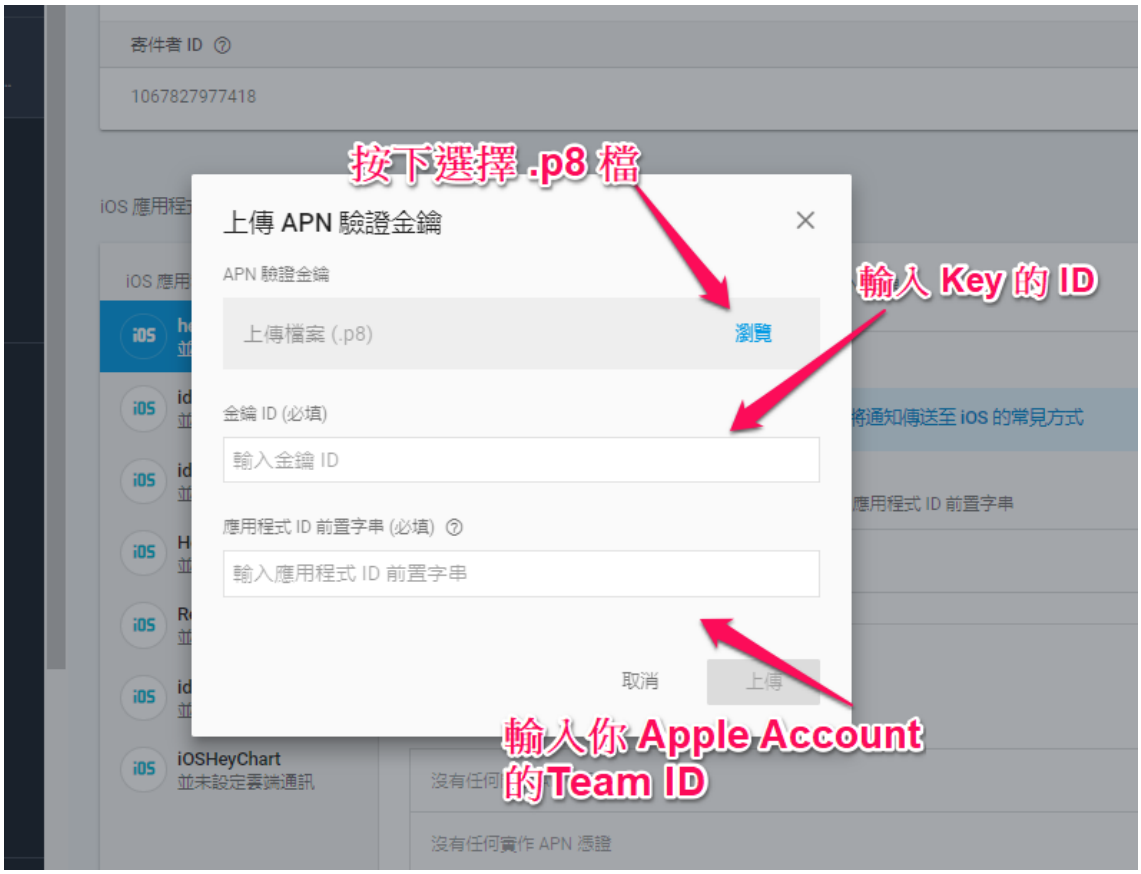
本章只介紹 .p8 的產生方式，但若你之前已經產生過 .p12 檔，那也是可以使用在 FCM 的。

我們先到 Firebase 控制台中，看看要上傳的資料有那些。到設定頁面後，選定 Cloud Messaging 就會進入設定頁面。

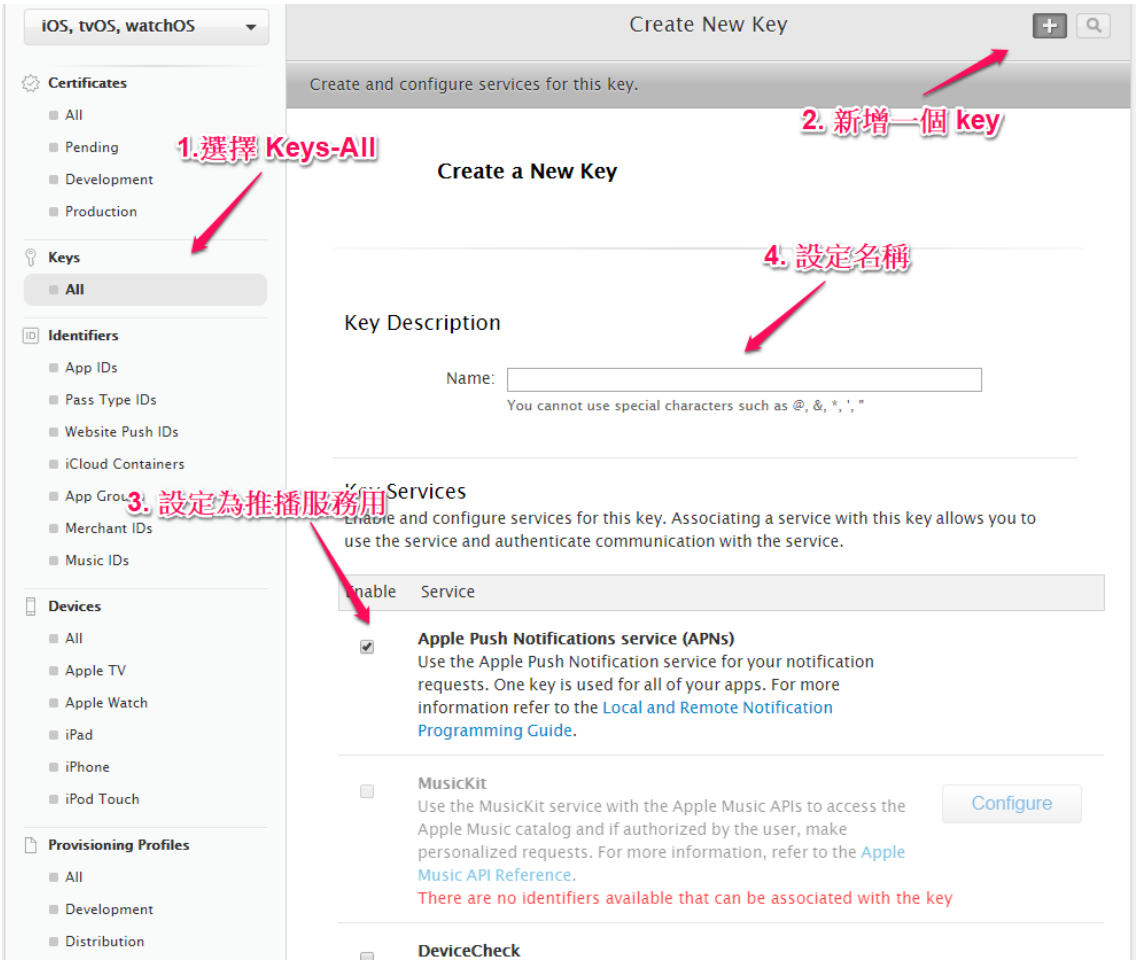
在這個畫面的下方，有 iOS 應用程式設定的清單，若你有正確連接 AP，指定你的 iOS APP，就有可以上傳憑證檔的畫面：

這個畫面有兩種上傳選擇，上面那個是【APN 驗證金鑰】，也就是較新的 .p8 格式的檔案。下面那個是【APN 憑證】，也就是傳統的 .p12 格式，APN 憑證還分為開發測試與實作上架兩種憑證。在這兒我們只討論【APN 驗證金鑰】的製作上傳，但若你已有 APN 憑證與了解它的操作方式，也是可以在這個畫面上傳。

按下【APN 驗證金鑰】的上傳鈕，就會出現上傳檔案的畫面。

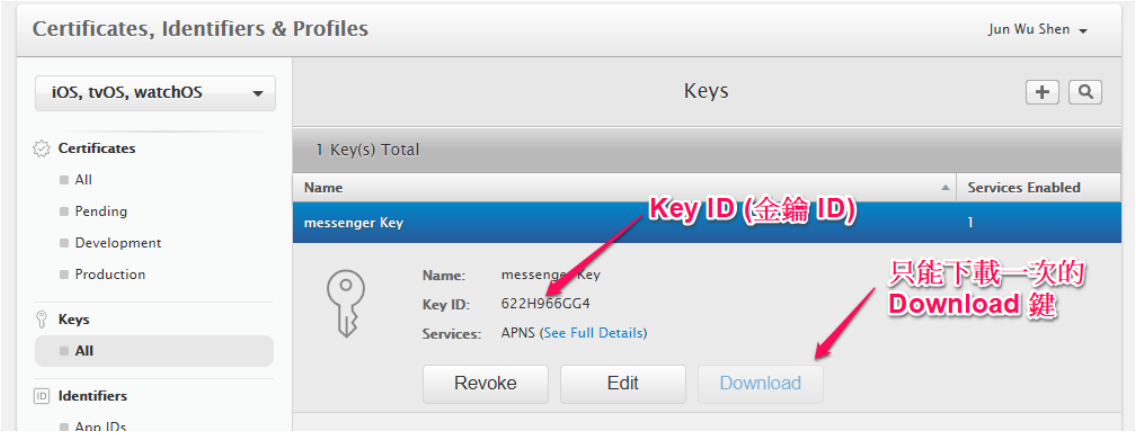


在這個畫面上，除了金鑰檔外，還有【金鑰 ID】 與 【應用程式 ID 前置字串】要輸入，這個翻譯有點怪，不過當我們來產生 .P8 金鑰檔時，這些資料就會出現了。所以現在要另開一個瀏覽器畫面，開啟 Apple Developer 的憑證網頁 https://developer.apple.com/account/ios/certificate/ 來產生我們要的檔案。

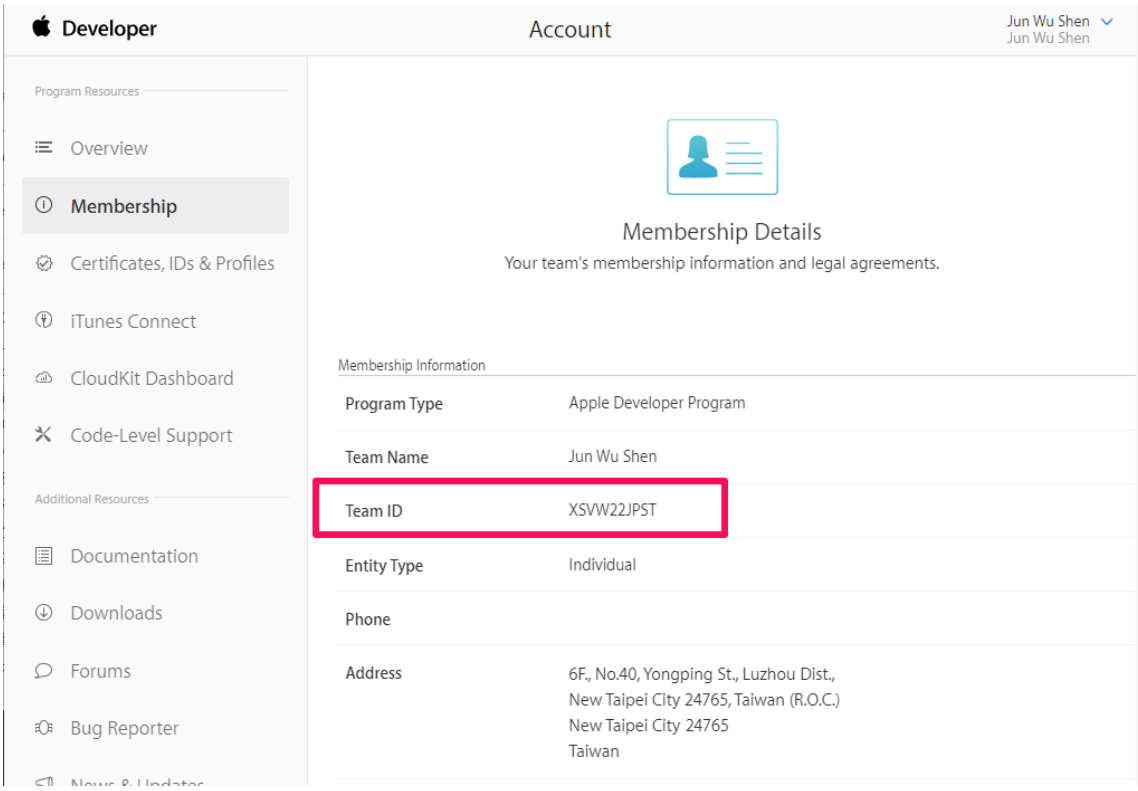


如果你使用付費帳號登入這個頁面，你就會在右則選單中，找到 Keys > All 的選單，在這個頁面你就可以按下右上角的【+】來新增，至少要選定 APNs 服務，並設好名稱之後就可以新增，下載了。

要特別注意的一件事是，這個 Key 只有一次下載機會，而且同一時間，只能有兩個 Key 可以運作，所以下載完成後，一定要好好保管這個 .p8 檔，沒有下次了，如果沒有保管好，那只能砍掉重練，並更新到你所有過去用過的 APP，所以【一定要好好保管】。



在下載頁面中，除了下載之外，還有一個種要的資訊是 Key ID，我們在 Firebase 控制台上傳金鑰的時候，就要輸入這個 ID。在 Firebase 控制台上上傳金鑰所需的三個項目，其中兩個在這個頁面可以取得。而另一個【應用程式 ID 前置字串】實際上在 Apple 開發者網站上叫做 Team ID，這個資訊在網站的 Account 頁面上，網址是 https://developer.apple.com/account/#/membership/



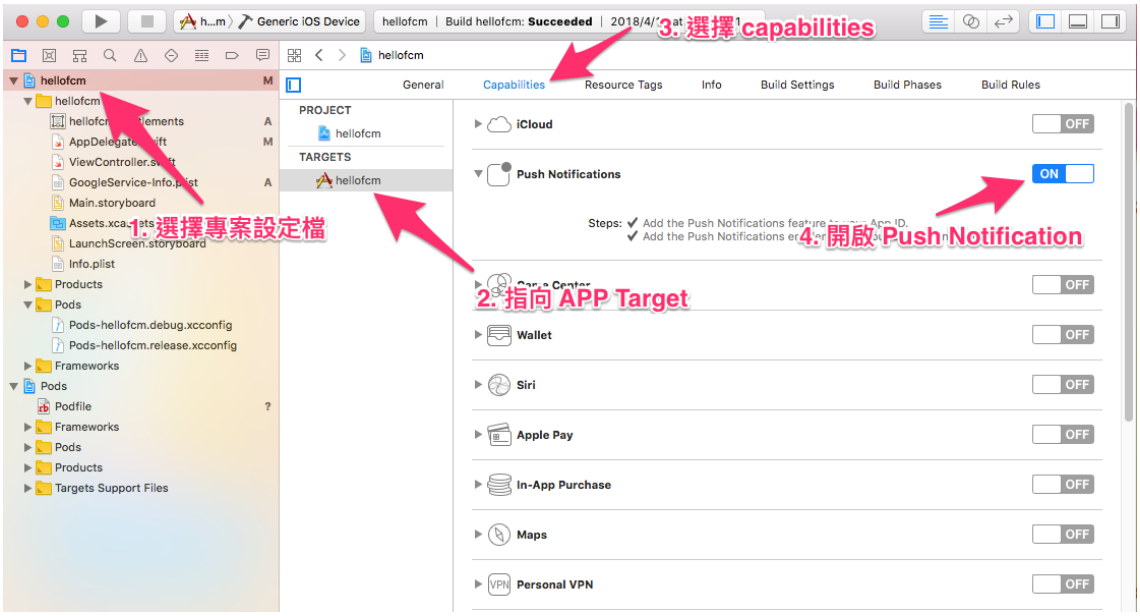
圖中的 Team ID 即為 Firebase 控制台所稱【應用程式 ID 前置字串】，如此一來，我們所需的東西齊備了，我們就可以上傳金鑰了。

設定 iOS APP 專案

後台的事設定好了，就該來設定開發專案了，有以下幾件事要做

1. 開啟 App 的 PushNotifictaions 功能
2. 在 AppDelegate 中的要求使用推播的登錄
3. 回傳得到的 FCM 推播 Token
4. 設定收到推播時，要處理的方法

我們來陸續完成以上工作，先打開 Xcode 專案的設定頁面，在要指向的 Target 選擇 Capabilites 頁，會有一堆功能的設定，如下圖：



其中，【Push Notifications】是我們要開啟的項目，開啟這個項目後 Xcode 就會自動幫我們在 Apple 建立 APP ID 並且作設定，Porvisioning File 也有可能一併更新，不能再使用 Wildcard，因為要在 Apple 的開發者網站登錄推播，所以會檢查帳號是否合於規定，不過這一切 Xcode 都會全自動幫我們設好，我們只要確認帳號與 ID 的唯一性就好。

接下來我們要在 AppDelegate 要求使用推播的登錄，在 AppDelegate.swift 中，

加入 import Firebase，並在 application(_ application:, didFinishLaunchingWithOptions launchOption:) 方法中，加入這幾行 code

```
        //指定推播的代理人
        Messaging.messaging().delegate = self

        //要求三種型式的推播
        UNUserNotificationCenter.current().delegate = self
        let authOptions:UNAuthorizationOptions  = [.alert, .sound,  .badge]
        UNUserNotificationCenter.current().requestAuthorization(options:
authOptions) { (_, _) in}
        application.registerForRemoteNotifications()
```

這段程分為兩個部份， 首先是取得 FCM 的實體，然後把 Deletate 指向自己，就如同 UNUserNotificationCenter 也就是 iOS 內建的訊息物件，也要做相同的事情，所以 AppDelegate 也要擴充 MessagingDelegate 與 UNUserNotificationCenterDelegate 兩個協定。這兩個協定都沒有 Required 方法，但我們通常也會實作一些方法。

至於後面的區段，就是設定我這個 APP 需要推播，而且要支持提醒，音效，數字標示等選項，完成後要求後要執行的程式等，最後 application.registerForRemoteNotifications() 執行時就會實際去 APNS 做推播登錄要求了，這個項目會和使用 APP 的人要求准許後，再到 APNS 登錄推播。如果不想了解也沒有關係，只要照打就可以了。這個範例只支援 iOS 10 以上的相容性，若是要製作 iOS 9 甚至更早的版本，這段要改寫成：

```
        //指定推播的代理人
        Messaging.messaging().delegate = self

        if #available(iOS 10.0, *) {
            UNUserNotificationCenter.current().delegate = self
            let authOptions: UNAuthorizationOptions  = [.alert, .badge, .sound]
            UNUserNotificationCenter.current().requestAuthorization(options:
authOptions,completionHandler: {_, _ in })
        } else {
```

```
            let settings: UIUserNotificationSettings =
            UIUserNotificationSettings(types: [.alert, .badge, .sound], categories:
nil)
            application.registerUserNotificationSettings(settings)
        }
        application.registerForRemoteNotifications()
```

這樣寫可以讓 iOS 9 之前的版本正確收到推播，但一些行為模式會和 iOS 10 之後版本不一樣，因為要實機測試，我也無法實測，所以後面的內容，限於 iOS 10 之後的版本為主，不再另行說明 iOS 9 之前的相容性。

其實只要設定好以上的內容，然後第一次執行，成功登錄推播系統後，就可以收到推播了，但是通常我們會在 AppDelegate 實作以下兩個方法，第一個是 messaging(messaging: didReceiveRegistrationToke fcmToken:)，也就是接收到發給特定的機器時，用的方法，範列如下：

```
    func messaging(_ messaging: Messaging, didReceiveRegistrationToken fcmToken:
String) {
        print("Get Token:\(fcmToken)")
    }
```

如果是未來要針對這一台機器發送推播的話，我們就需要這一個 Token。大部份的狀況是傳送到另一個地方，例如 Realtime Database 或自己建的後台。但如果你只是想群發的話，只要設好這些方法確認有完成推播登錄就好了。在這兒的範例，我們只把它印出來，作為測試使用。

另一個是收到推播時會執行的方法，範例如下：

```
    func application(_ application: UIApplication, didReceiveRemoteNotification
userInfo: [AnyHashable : Any], fetchCompletionHandler completionHandler:
@escaping (UIBackgroundFetchResult) -> Void) {
        print("User Info:\(userInfo)")
    }
```

當 APP 開啟狀態下，收到推播時，這個方法就會被呼叫。我們在這是把收到的訊息印出來，做為測試用。

Android 基礎實作推播

Android 一開始有推播時，是用 C2DM 系統來執行，後來由 GCM 來取代 C2DM，而現在 Google 主推 FCM，但 FCM 並非是要取代 GCM，而是簡化整個程序，在實際執行過程中，幾乎看不到 GCM 的相關設定，但實際上，FCM 是在 GCM 的外層再包了一層服務，而這個服務，同時也可以服務 iOS。 那我們就來看看整個服務如何執行。首先，我們建立一個空白專案，然後依下例動作執行。

建立專案引入 SDK

第一件要做的事情就是要引入 SDK，使用 Android Studio 自帶的工具，這件事很容易達成。建議先引入 Analytics 測試連線無誤後，再來引入 FCM 的 SDK，這些動作 Android Studio 都有自動工具協助完成，只是要注意不要選錯專案就可以。另外 Package Name 必需是唯一的，由於推播在真實世界執行，所以若 GCM 已經有人用和你一樣的 Package Name 那你就必需換一個名字重來。

要檢查自動工具有沒有幫我們正確的設定，就要查看一下 app 層級的 build.gradle 是否有正確設這兩行，並正確完成 Sync

```
implementation 'com.google.firebase:firebase-core:11.8.0'
implementation 'com.google.firebase:firebase-messaging:11.8.0'
```

這兩行應該在該檔案的 dependencies 區段，也許你看到的版本和我不一樣，那是正常的，但一般來說，這兩個 SDK 的版本應該是相同的。

建立網路推播憑證

要在 Android 上推播，除了引入 SDK 外，還要建立憑證，也就是機器的證明。由於我們在連繫 Firebase 時，已經做過機器的確認，所以 FCM 會幫我們打理一

切，除非你換電腦與指紋檔時，才要重新建立一次。建憑證的方法非常簡單，只要進入 Firebase 控制台，選定你的專案，在【專案設定】中，按下 Cloud Messaging，就會出現設定畫面。

這個畫面有許多設定，因為憑證可以每一個 APP 都不同，iOS 與 Android 當然也不同，但若你的 Android APP 從未設定過推播憑證，在畫面的最下方，會有一個產生憑證的按鈕，按下去就可以產生憑證，我們就完成了憑證的設定。

若你之前有實作過 GCM 的推播，而原來的推播系統也要繼續使用的話，你可以不用產生新的金鑰，而直接匯入現有的金鑰也是可以在這個畫面操作的。

建立服務

為了隨時接收訊息，我們要建立兩個 Service，所以在 AndroidMainfest.xml 中的 application 段，加上下面兩個 Service 的記錄。

```
        <!-- [START firebase_service] -->
        <service
            android:name=".MyFirebaseMessagingService">
            <intent-filter>
                <action
android:name="com.google.firebase.MESSAGING_EVENT"/>
            </intent-filter>
        </service>
        <!-- [END firebase_service] -->

        <!-- [START firebase_iid_service] -->
        <service
            android:name=".MyFirebaseInstanceIDService">
            <intent-filter>
                <action
android:name="com.google.firebase.INSTANCE_ID_EVENT"/>
            </intent-filter>
        </service>
        <!-- [END firebase_iid_service] -->
```

設定了兩個服務，當然就要設定相對應的 Class。我們需要新增兩個 Class，第一個 叫 MyFirebaseInstanceIDService 需繼承 FirebaseInstanceIdService，另一個 MyFirebaseMessagingService 需繼承 FirebaseMessagingService。

我們先來看 MyFirebaseInstanceIDService 的範例。

```
public class MyFirebaseInstanceIDService extends FirebaseInstanceIdService {
    @Override
    public void onTokenRefresh() {
        String refreshedToken = FirebaseInstanceId.getInstance().getToken();
        Log.d("MyFirebaseIIDService", "Refreshed token: " + refreshedToken);
    }
```

}

FirebaseInstanceIdService 的功能是當 APP 完成和 FCM 登記後，FCM 會回傳一個 Token 如果是未來要針對這一台機器發送推播的話，我們就需要 Token。當然，自己是不需要發送給自己的，所以大部份的狀況是傳送到另一個地方，例如 Realtime Database 或自己建的後台。但如果你只是想群發的話，只要設好這些方法確認有完成推播登錄就好了。

當系統完成推播登錄時，系統會呼叫 onTokenRefresh() 方法，所以我們必需 override 這個方法，而回傳的 Token 可以用 FirebaseInstanceId.getInstance().getToken() 取得。上面的範例只是將輸出寫到 Log，以確認有成功登錄。雖然我們隨時都可以使用以上方法取得目前的 Token 但只有這個方法能在更新 Token 的第一時間取得，避免傳送錯誤。推播 Token 雖然一般來說不太容易變更，但是仍然不是完全不會變更，比如說若手機執行重置時，就有可能變更，所變處理的最佳時間點就是這一個方法。有一種狀況是若你的通知對象是特定帳號的話，就可以在這兒先把 Token 存到 APP 資料如 SharedPreference 中，然後每次開啟 APP 完成登入時，一併回傳。

FirebaseMessagingService 則是 APP 在開啟狀態下，所需要處理的事。先看一下 MyFirebaseMessagingService 的範例：

```
public class MyFirebaseMessagingService extends FirebaseMessagingService {
    @Override
    public void onMessageReceived(RemoteMessage remoteMessage) {
        Log.d("Msg",remoteMessage.getNotification().getBody());
    }
}
```

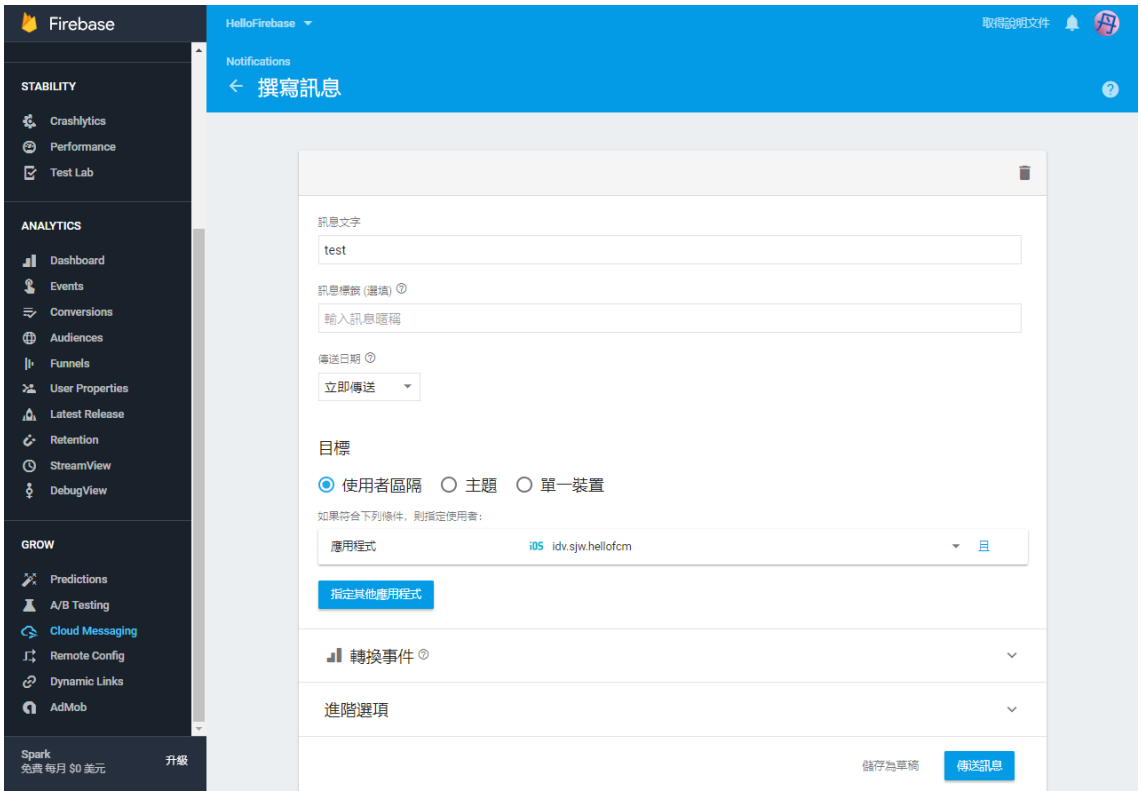
當 APP 是在開啟狀態，收到訊息時，系統就會呼叫 onMessageReceived(RemoteMessage remoteMessage) 方法，而訊息內容就是這裡回傳的 RemoteMessage 物件。這次，我們用最簡單的方法，確認傳送的內容，把它顯示在內容中。

如此一來，我們的第一步就完成了，目前的狀況是已經可以接收和傳送推播了，我們先測試完成實際收發沒有問題，其他的項目就可以慢慢發展了。

發送推播

第一則測試訊息

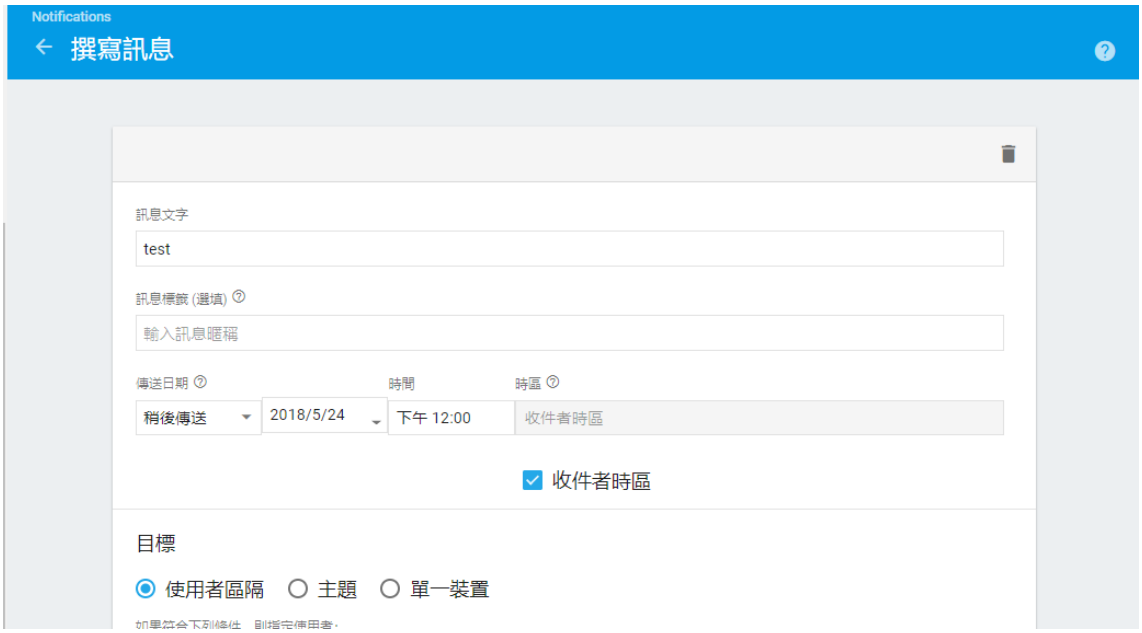
當然，我們 APP 寫好之後，就要實作發送了，我先使用 Firebase 控制台內建的發送平台，來進行第一個發送的動作



發送訊息，第一件事要選內容，第一次我們就用【test】或任何一個想打的文字即可。傳送時間選定立即傳送，目標則選用使用者區隔，並選定你要發送的 APP 的 PackageName 或 Bundle ID，按下傳送訊息，使用者就會收到訊息了。

預約推播

傳送日期一欄是指何時發送訊息，也就是發送的時間，這個時間要考慮到使用者收到的時間，如果是不適合的時間，很可能會引起使用者的抱怨。筆者的經驗中，就有因為發送時間太晚，而受到客戶申訴過，一般來說，早上九點到晚上九點是比較適合的時間，但還要考慮使用者可能各種原因會晚一點收到訊息，所以時間會設到 20:00 之前，若你是全球發佈的 APP 的話，還要考慮不同時區的問題，還好 Firebase 已幫我們處理了這一點，若不是立即傳送，而是預約的話，可以設定以收件者的時區來預約發送，但立即傳送時，就要仔細思考收件者的時區問題，而改用預約發送，但時間是接進最近的時間，就可以用使用者各自的時區來發送了。



目標對象

目標對象就是收件者是誰。可用三種方式選擇，第一種是【使用者區隔】，第二種是【主題】，第三種是【單一裝置】。

【單一裝置】裝置的概念最簡單清楚，就是只給單一使用者，而所需要輸入的內容，就是之前程式碼提到的 FCM Token，輸入某特定裝置回傳的 FCM Token，就可以傳送給該使用者。這種狀況除非為了測試，否則大多數的狀況下，是由程式來發送，比如說某個會員完在網站上完成特定的任務後，就可以推播給同帳號的使用者，開啟優惠訊息或遊戲功能等。

【使用者區隔】是根據 PackageName，Bundle ID 與 Analytics 的 Audiences 或 User Properties 來選定。先要選定 PackageName / Bundle ID 然後如果有必要的話，可以設定進一步的條件，比如說在預設的狀況下，可以選定只發送給消費過的使用者接收訊息，就可以設定 Audiences 為 Purchasers 的使用者，如下圖：

這個設定可以同時指定不同的條件，同時發送給 iOS 及 Android 或同一個專案下的多個 APP，一起收到同一則訊息。

【主題】則是一種分群的概念，你要輸入一個字串當 key 而訂閱過該主題的人才會收到。而訂閱該主題要在 APP 程式中，設定好訂閱的程式。例如說，Topic 叫做【news】時，訂閱該主題的人就會收到，而在 iOS/Android 訂閱該主題的程式如下：

```
iOS:        Messaging.messaging().subscribe("news")
Android:    FirebaseMessaging.getInstance().subscribeToTopic("news");
```

只要 APP 層經執行過它們，就會收到主題是【news】的訊息，當然，APP 也可以使用 unsubscribe 方法來取消訂閱。

進階選項

我們打開進階選項，會看到許多東西。【標題】是一個縮短的內容，用在 Apple Watch 或 Android ware 上顯示區較小時的替代內容【Android 通知管道】與【自訂資料】是推播額外資訊，推播發出時，除了系統要求的內容以外，其實可以附帶點額外的訊息，在收到訊息後，這些資料不會直接顯示給使用者，但程式可以在收到後解析其內容，並做出合適的處理，比方說，切換到特定的頁面等。推播

系統通常稱這些資料為 payload 的自訂訊息，由程式解析後處理，但實作的方法是決定於 GCM 或 APNS 進階處理項目，並不會由 Firebase SDK 來處理，有興趣的讀者可以查閱 GCM 或 APNS 有關 push notification payload 的相關訊息。

再來有三個選項，【優先等級】、【音效】和【iOS 標誌】。【優先等級】的性質和前面的 payload 差不多，目前因為 FCM 95% 的訊息都是 0.25 秒送達，所以優先等級只是告知使用者而已。【音效】是指收時會不會發出聲音，如果你想在不適合收件的時間發送訊息的話，至少你需要把聲音提醒停用。最後一個是【iOS 標誌】，開啟它的話，還要輸入一個整數，也就是 APP 右上角的小紅點，通常稱為 badge。這個東西 Android 上也常見，但 Google 並沒有官方的作法，但是大多數 Android 手機都有通用的處理方式，只資料要用自訂資料，也就是 payload 的方式來處理。

最後一個是【有效期限】。因為發送當下，收件者可能因各種原因無法連到網路無法收取，也有可能手機根本已經轉移或故障，再也無法收取了。當無法找到收件對象時，FCM 會保留訊息等到使用者上線後再次傳送，但保留並不是無限期，GCM 與 APNS 也不會無限期的等待，所以有個最終放棄的時間，也就是我們設定的有效期限。因 APNS/GCM 的限定，有效期最大與預設都是 28 天(四周)，但某些狀況下，我們必需縮短這個時間。比如說，一則通知七天後開會的訊息，其實過了七天之後是沒有意義的，若第八天使用者收到 ”昨天” 開會的訊息，一定會產生很多抱怨，在這種狀況下，我們就應該把有效期減少到開會時間為止就好了。

程式化發送說明

手機上的 SDK 是沒有提供發送功能的，要程式發化送，就透過 HTTPS 或 XMPP

來打 API，你何以用任何程式語言來執行自動化的目地，但發送推播的內容，其實其資料格式可能比想像中要複雜一些。簡單的講就是把上一節可以手動設定的所以資料，串成一個 JSON 來打 Firebase 的 API。

相對於我們直接打 APNS/GCM 的 API 來說，FCM 的主題功能，提供了很大的效能發揮，特別是在使用者多的時候。因為 APNS/GCM 的處理能量雖大，但他們只提供發送到單一機器的功能，所以若要發送到十萬個手機上，就要傳送十萬個機器序號，又因為有每筆傳送的限制，十萬個可能要分成一千次以上傳送，每傳送一次還要間隔一小段時間等待回應。你用單一機器傳送，十萬光是傳送要求時間，就數十分鐘過去了，可別談到達時間了。但透過 FCM，且有設定好 Topic 時，就會利用 FCM 為數眾多的服務器分散發送給 APNS/GCM 效能是非常好的。

本書主要的討論內容是行動 APP 前的技術，並不是要討論如何由後台發送，但是推播並不只是前台的事，所以要和實作後台的人，好好商量 FCM Token 回傳的方式，與 Topic 如何設計，就能輕鬆完成整體推播的執行了。