佈署以網頁為基礎行動 APP 的框架

黃俊穎
國立交通大學
Email: chuang@cs.nctu.edu.tw
李昀陞
國立臺灣海洋大學
Email: 10457023@mail.ntou.edu.tw

摘　要

行動科技的進步使得我們有更多的開發方式來選擇開發手機 APP。以目前最普遍的來說，在手機運行的行動 APP 有三大類，分別為原生(native)、原生混合(hybrid)以及 mobile web 等開發方式。原生的開發方式就是以每個行動作業系統平台上的標準開發工具包(standard development kit, SDK)並使用特定的程式語言作開發。原生混合的方式則是在部分的程式碼會以 web 技術相關的方式做編寫，如 hypertext markup language (HTML)、Cascading Style Sheets (CSS)以及 JavaScript；最後的 mobile web 則是單純全部以 web 技術做開發，也就是包含前端技術與後端的 web 技術。

本論文研究是要去展示出一個有別於上述這三大類以外的一個新的框架，這個框架可以使開發者只要會 web 相關的技術就可以開發出一個 APP 外，透過我們的自訂的規則下將開發好的 mobile web 專案佈署到我們的框架上，讓一般的使用者透過一般的 web browser 就可以體驗到一個完整的手機行動 APP。

關鍵字：跨平台開發、行動網頁應用程式、軟體架構與設計

一、簡介

隨著智慧型手機行動裝置的快速發展，有越來越多的手機 APP 應用被開發出來，其中目前在市面上佔有率最高的三大行動作業系統 Android、iOS 以及 Windows Phone，前兩者在市占率最高。隨著許多的手機行動應用程式出現，我們把它們叫做 mobile APP。它們對於一般的使用者，讓人們越來越感到便利之外，也豐富了人們的生活。

行動 APP 開發對於應用程式開發者來說是一個新興且積極佈局市場。由這篇論文 [1]可以得知在 2015 年時，行動 APP 市場有高達約四百多億的收入且到了 2020 年會上看至一千多億。這項研究的結果也顯示使用者花在使用行動平台的時間超越了使用桌上型平台，而行動 APP 市場收入也會在未來超越桌上型 APP 市場。

在手機 APP 開發的方式日益進步下，目前有三種主流的開發方式，分別為原生、原生混合與 mobile web 等三種開發方式，其中原生開發方式最主要的好處為對於這個平台上有完整的支援系統特性之外，也是其指定使用的程式語言，因此在同一種平台上，效能與表現上會比其他的開發方式來的好。舉例來說，像是在 Android 平台上使用 JAVA 做為開發 APP 的程式語言；iOS 則是以 Swift 或 Objective-C 開發 APP 的程式語言。原生混合的開發方式則是利用 web 相關的技術來實現 APP 程式碼，通常這類的框架會需要依賴每個作業系統平台上的 SDK，框架上開發的核心函式庫使能夠與 SDK 做溝通，使其能夠讓使用者存取函式庫就可以在每個不同的行動作業系統上存取手機上的系統特性，例如：硬體感測器，像是藍芽(Bluetooth)、近場通訊(near field communication, NFC)等，其優點就是開發者不需要理解每個行動作業系統上不同的程式語言以及上面相關存取系統特性的 application Programming Interface (API)文件，只需要使用 web 技術並使用跨平台框架進行包裝即可開發出一個跨平台的行動 APP，而這個 APP 也是可以發佈到行動 APP 市場上的。這類開發方式的例子以 PhoneGap 或稱為 Cordova [2]最為代表。mobile web APP 則是單純以 web 技術做開發之外並透過網頁瀏覽器去執行 APP，是一種快速且方便的開發方式。

以原生開發方式來說，對於開發者除了需要熟悉指定的程式語言外，還需要了解 API 的文件以便能夠存取手機上的硬體感測器，這其實對於要開發多個不同的行動作業系統平台的 APP 開發者來說是非常不利的，因為幾乎需要為每一個作業系統開發一次，也需要了解不同作業系統的架構與存取作業系統特性的 API 文件，這樣不僅造成時間上的浪費，也在商業上額外增加了成本費用。若是以原生混合開發方式，則是比原生單純的多，只要熟悉 web 技術就可以開發出一個 APP 之外，這個 APP 還可以達到跨平台，除了可以節省成本的費用之外，還可以使 APP 開發者降低開發與學習的曲線。

除了上述三種開發方式之外，我們在這篇文中提出了另一種框架，我們把它稱為 Forklift，這個框架與開發 mobile web APP 一樣提供 APP 開發者可以使用 web 相關的技術做開發外，就可以依照我們定的規則把開發好的 APP 專案佈署到框架上面，接著使用者可以使用行動裝置並打開我們做好的客戶端(client) APP 進行操作 APP 了。我們框架這樣設計的好處是，與原生混合的開發方式比較，以往使用原生混合開發方式是需要透過跨平台框架上設計的核心函式庫做存取 SDK 並進而能夠存

取手機上相關的系統特性，我們框架雖然也會依賴各個行動作業系統上的 SDK，不過我們存取的方式有別於原生混合方式，其使用的是設計一個核心函式庫給開發者存取；而我們做法是使用 RESTful 的架構來設計出存取系統特性。具象狀態傳輸 (Representational State Transfer, REST) [3]是一種軟體架構的設計方式，而使用 REST 實現 web 相關應用程式的服務我們稱為 RESTful。

我們在這裡總結我們提出的 APP 開方方式主要有三點好處，第一點是便利性，相較於原生 APP 所產生的安裝檔，我們產生安裝檔的步驟與方式會較簡易。例如在 Android 作業系統上則是使用 Android application package (APK)做為手機 APP 安裝檔，而我們純粹是把 APP 專案佈署到框架上就成為一個可以在手機上透過客戶端 APP 操作的 APP 了。第二點是安全性的問題，若是我們採用 mobile web 做為開發的方式，還有另外一種是利用 WebView 這類的元件在 Android 系統上也能夠開發出另類的手機 APP，且大部分也是利用 web 技術開發，不過就我們所知道的，使用 WebView 在 APP 中使用其風險是偏高的，因為根據 WebView 相關安全議題的論文 [4]表示，此元件容易因為啟用了 JavaScript；而 JavaScript 可以透過 WebView 的橋梁(bridge)機制與手機 APP 上的 JAVA 進行互動，而橋梁上本身有注入的漏洞風險，進而導致發生安全性的問題發生，使得 APP 風險性提高。第三點則是我們的框架並沒有設計額外的 JavaScript 函式庫，而如果是利用 PhoneGap 這類原生混合開發方式的話則是有的，那些核心函式庫或是針對手機上相關的系統特性所撰寫的 JavaScript 函式庫做為 PhoneGap 的外掛，都是為了不直接接觸 SDK 核心而產生出來額外的 JavaScript 函式庫，而我們做法是將存取系統特性的這些函式庫全部用 RESTful 方式取代，達到方便且簡易存取手機硬體感測器的效果。

在此篇論文中，我們依照下面章節的分類並分別簡易的說明下列章節所需要做的事情，在相關研究工作中，我們會敘述我們尋找了哪些的方法，並分析我們找到的資料，以及歸納出建構這個框架相關的文獻資料。在設計目的中，我們會描述這個框架中的設計概念、設計的方法以及呈現系統框架的架構圖；在設計方法比較中，我們將章節中所提到的三種方法與我們的方法列出表格並比較；在框架中，我們會描述建置框架的方式外，另外會展示與描述使用者情境圖。在實做中，我們會分析框架中如何時做容器(container)的元件，在最後的結論中，我們對整篇論文做一個結論與回顧。

二、相關研究工作

隨著行動裝置越來越進步，有越來越多的跨平台與一般的 mobile APP 成為熱門的討論話題，但是在每個不同的行動平台上開發上跨平台的 mobile APP 是較為困難的 [5]。

有不少的研究者以建立不同模型為出發點去開發出一個跨平台的 mobile APP。由 Henning Heitkötter 等人所著作的論文即是如此，他們的做法是撰寫一個程式碼的產生器，其名字叫做 MD^2，而這個 MD^2 將會產生對應平台的程式碼之外，而且還是一個完整的行動 APP。Bouras 等人論文 [6]指出使用了 HTML 5 來做為開發他們的跨平台 mobile APP，作者做出的結論是他們所開發出來的跨平台 APP 所提供的功能可以與原生的 APP 一樣。不過，作者有在論文中列舉出清單並說明有一部分有包含使用系統特性的功能是無法在 mobile web APP 裡面支援的。像是 APP 離線的使用、存取行動裝置上的感測器以及和其他的行動 APP 做互動。

mobile web APP 開發目前仍是一種跨平台開發的一種方式 [7,8,9]。但是，web 開發者們仍是需要克服一些挑戰，像是使用者的偏好、有限的存取裝置上的感測器以及 APP 的穩定性等。有一些研究者研究過各式種類的跨平台開發工具並比較它們之間的差異性。Ohrt 與 Turau 等人撰寫的論文 [10]中，他們注意到有一關鍵點是開發工具需要滿足開發者的需要和滿足使用者對這個 APP 的期待。Xanthopoulos 等人的論文 [11]針對跨平台開發進行了比較，他將這些跨平台開發工具分成了四大類，分別是 web APP、hybrid APP、integrated APP 和 generated APP。在文章中的最後，作者總結了這四個種類的特性與優劣，Generated APP 有著最好效能，但是開發者必須使用新的且非標準的開發工具和開發環境進行開發。Heitk¨otter 等人寫的這篇文章 [12]，他們在最後的結論中，除非要存取到手機裝置上的硬體感測器等相關的系統特性之外，hybrid 的開發模型將會是一個對於跨平台開發者來說是首選的開發方式。

三、設計目的

我們將這個框架叫做 Forklift，Forklift 目的是在於提供一個開放平台給研究者與開發者做為研究跨平台 mobile APP 開發的可行性與使用標準的以及普遍的前端與後端技術等這些 Web 技術進行佈署 mobile APP。

Forklift 的設計目的包含了：

- 簡易性(Simplicity)：一個 APP 開發者可以簡單的將一個 mobile web APP 的專案進行解壓縮與佈署到 Forklift 的容器中。
- 延展性(Extensibility)：Forklift 支援了多個後端的網站技術，包含了 PHP、Python、Ruby 和 Node.js 而且我們可以簡單的去自行客製化去支援更多的後端相關的程式語言。
- 安全性(Security)：多個 APP 可以同時的安裝到 Forklift 的容器中，而每一個 APP 會在 Forklift 為這個 APP 所創建有限的環境下運行且避免惡意的 APP 能夠破壞其它的

Forklift APPs.

- 公開性(Openness)：我們計畫將這個平台公佈出來，Forklift 的使用對於開發者來說是免費的但研究者與開發者應該要遵守套件與 Forklift 框架的許可證的聲明。

四、設計方法比較

我們將先前所提到，分別是：以 WebView 做為基礎開發的行動 APP、原生 APP 以及原生混合 APP 等三種方法與我們的做法進行比較，如下表一可以得知，以 WebView 基礎與原生這兩個開發方式的優勢是一樣的，而原生又會比以 WebView 基礎好，其原因就是以 WebView 基礎若開發不注意，容易發生漏洞；原生與跨平台相比較，若不考慮學習曲線因素，原生各方面表現會比跨平台來的優勢許多，若考慮學習曲線與單純是呈現資料且不會存取裝置感測器的 APP 類型的話，跨平台的方式是可以考慮選擇的。最後用 Forklift 框架來與原生和跨平台相比較，Forklift 具有跨平台的學習曲線較低與原生的對硬體感測支援度高，至於劣勢消耗記憶體的問題，我們相信隨著科技的進步，手機硬體配備會越來越好，這方面在硬體的配合下，劣勢會逐漸的消失。

表一、三種開發方法比較

	以 WebView 基礎	原生 Native	跨平台 Cordova	Forklift 框架
優勢	對於平台相容性佳，對硬體感測器支援度高。	對於平台的相容性佳，對硬體感測器支援度高。	容易跨平台，減少開發的成本。	硬體感測器支援度高，網頁開發者學習曲線較低。
劣勢	開發者若不小心則容易發生安全性問題。	網頁開發者學習曲線較高與不容易跨平台。	存取裝置感測器反應較慢。	存在著執行 APP 使用記憶體較大的問題。

五、框架

我們在討論 Forklift 框架設計之前，我們先來用情境圖來說明 Forklift 框架對於開發者與使用者來說是如何運作的；在圖一中我們可以看到對於 Forklift APP 開發者與使用者的情境圖。開發者可以選擇使用他們所偏好的前端與後端框架、工具以及函式庫進行開發之後並確認所有的 APP 檔案都放到同一個的 APP 檔案根目錄中。這個 APP 開發模型有點類似 Firefox OS。在開發者的 APP 專案完成且穩定之後，開發者就可以將這整個專案的檔案壓縮成單一個檔案的方式進行包裝，而這個檔案包含了一個給 Forklift 平台的設定檔。這個設定檔裡面包含了一些 APP 相關的設定，其包含了 APP 的識別 id、APP 的版本號、所需要的後端服務以及所需要存取的系統特性，如藍芽、NFC 等。使用 Forklift 框架，可以在最小變動下將一個已經存在的 mobile web APP 可以輕易的移植到在行動平台上。接著開發者上傳內有 APP 的壓縮檔到 APP 的儲存庫(repositories)中，也就是應用程式相關專案的地方。藉由分享網址方式給 Forklift 使用者來達到發佈 APP 的目的。

為了要安裝這個 APP，Forklift 使用者需要得到這個 APP 的壓縮網址，並下載這個壓縮檔並使用我們開發的 Forklift 客戶端程式進行安裝。Forklift 客戶端程式，我們簡稱為 Forklift 客戶端，它是我們這個 Forklift 框架的核心，Forklift 客戶端程式是在所支援的行動平台上原生方式所開發而成的行動 APP，我們為了簡潔度，所以我們將下面所談到的應用 Forklift 客戶端程式改稱做 Forklift 客戶端。相對的，我們實做的 APP 是使用 Forklift 框架並使用 Forklift 客戶端將 APP 啟動成 Forklift APP。Forklift 客戶端運作方式很像一個正常的行動 APP 且會在有限制的環境下負責安裝、管理和啟動 Forklift APP。一個簡易的 Forklift 客戶端介面示意圖，如圖二所示。所有已經安裝的 APP 會在 Forklift 客戶端介面中以清單方式調列出來。使用者可以從清單中選擇他們所要使用的 APP 並啟動它們接著與進行操作 APP。在本節剩餘的後段部分，我們將會介紹 Forklift 整個框架的架構以及 Forklift 客戶端的執行期環境的設計。

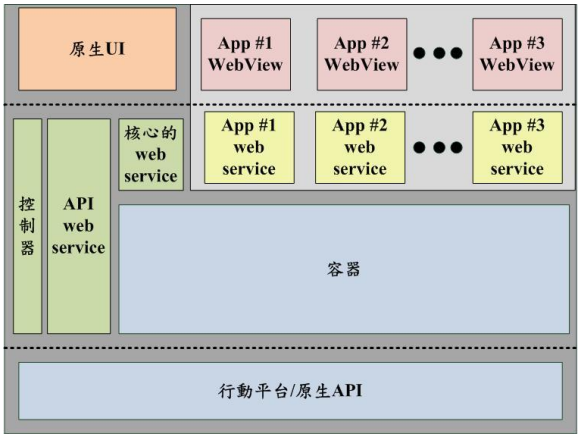
圖一、簡易情境描述 Forklift APPs 開發者與使用者

圖二、簡易的 Forklift 客戶端介面

5.1、架構概述

Forklift 客戶端是實做成一個普通的 APP，其建立在既有的行動作業系統平台上。舉例來說，Android 與 iOS 系統。Forklift 客戶端如圖三所示，在圖中，我們可以看到最底層是行動平台，它提供了原生(Native) API 讓上層的 APP 可以存取，在 Forklift 的架構中有許多的元件，而我們將這些元件分成兩大類。一類是 Forklift 核心元件，而另一類是 Forklift 的 APPs 的元件，當 Forklift 客戶端啟動或是可能的話會以系統服務的方式啟動，在這樣的形式上會將所有的元件都帶起來。即使裡面並沒有正在執行的 Forklift APPs，這些元件仍是需要

的。相反的，如果至少有一個正在執行的 Forklift APP 的話元件而也只需要這些 APPs 元件；我們將在下面詳細的介紹每個元件的功能。



圖三、正在執行 APP 時的 Forklift 客戶端架構

- 原生(Native)使用者介面：原生使用者介面(UI)提供了原生的介面給使用者下載、安裝、管理、選擇和執行 Forklift APPs。而這個元件也可以讓使用者自行切換正在運行的 APPs。
- 控制器(Controller)：控制器是負責提供安裝各式的 Forklift 客戶端所需要的工具與技術，控制器會將 API web service 與核心(core) web service 帶起來，它同時也提供了介面供其他的元件去查詢 API 與 web service 服務的窗口。
- API web service：API web service 提供了一個 Web 介面並透過這個 RESTful Web 介面來讓 Forklift 元件可以存取原生系統上的特性。這些 API web service 在是使用原生的程式碼所實做出來的。
- 核心的(Core) web service：核心的 web service 提供了給 Forklift APPs 後端的相關的支援。後端相關的支援包括了關連式資料庫、鍵(key)-值(value)的儲存方式以及支援額外的後端腳本式的程式語言，例如：Python、PHP、Ruby 等。
- 容器(Container)：容器的目的是確保在執行時環境的每一個 Forklift APP 都擁有自己的權限。一個 Forklift APP 只能存取自己的檔案以及公開權限提供所有人都可以存取的檔案，Forklift APPs 是不可能存取到 Forklift 核心(core)元件相關的檔案以及那些其他的 Forklift APPs 的檔案。
- APP web service：每個正在執行的 Forklift APP 都有一個專屬的 APP web service 可以使用，而一個 APP web service 會隨著 Forklift APP 的執行與中斷。
- APP WebView：WebView 是一個 UI 元件，它負責展示 Forklift APP 的畫面給使用者。圖中數個 WebView 取決於在執行的 APPs 數量。然而，只會有一個活躍的 WebView 能夠與使用者做互動，要注意的是當 Native UI 是活動的時候，其它的 APPs WebView 是不活動的。

我們在這個章節中，討論了當 Forklift 客戶端初始化 APPs、安裝 APPs、啟動 APPs 以及存取系統上的特性的時候是如何與架構之間的元件做互動的。在最後章節中，我們也會討論在 Forklift 框架中的安全性問題。

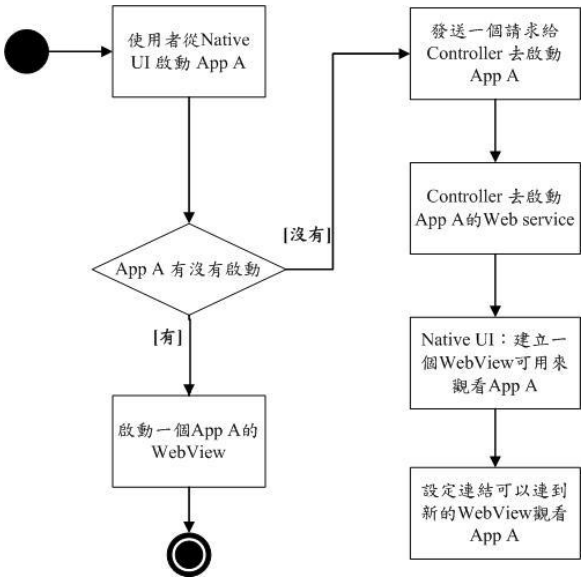
5.2、初始化(Initialization)

當 Forklift 客戶端開始執行時，控制器會將 API web service 與核心的 web service 帶起來。在預設的情況下，有兩個 web services 綁定到在本地位址(local address)上的隨機埠號(port number)，通常隨機的埠號則以 1024 以上為非應用層上通訊協定所指定的特殊埠號，舉例來說，像是 127.0.0.1 或 localhost，而埠號可能是 8080 或是 8081。為了確保其它核心的元件能夠時常的發現這兩個 web services，因此 Forklift 使用了多點傳送域名系統協定(multicast Domain Name System, mDNS) [13]，並使用這個協定註冊這兩個 web service 的位址與埠號，接著核心的 web service 會帶動其他的後端服務，像是關聯式資料庫、鍵-值儲存以及其他的服務，這些所有的後端服務皆會在本地位址綁定隨機的埠號；核心的 web service 會處理與決定這些後端服務的位址，而且也提供了一個 RESTful 介面提供其他的元件做查詢可以使用的後端相關的特性。

5.3、APP 的安裝

安裝一個 Forklift APP 是非常的簡單。當使用 Forklift 客戶端下載了一個 APP 之後，控制器則負責把這個 APP 壓縮檔進行解壓縮並將解壓的檔案解壓到適當的位址以及安裝 APP。假設我們要安裝一個 APP 的 id 為 A，而使用者 id 為 U_A，在容器中會建立一個 APP 的目錄 d_A，而使用者 U_A 是擁有 d_A 這個目錄，這個目錄只有使用者與建立容器的人才有權限可以存取 d_A 這個目錄。容器會去負責將使用者 APP id 對應到相關的使用者 id 與目錄，舉例來說，則是會：$\{A \rightarrow (U_A, d_A)\}$；這些相關的訊息存在控制器中內部的儲存空間。除了建立使用者 id、建立 APP 目錄以及複製 APP 的檔案外，如果 APP 還需要存取系統上的特性或是後端相關服務技術的話，控制器還需要為了這個 APP 執行環境上的設定。我們在章節 4.6 提到。

5.4、APP 的執行



圖四、執行一個 APP 的流程示意圖

我們假設使用者試著要執行 APP A，執行 APP 的相關步驟如圖四所示。首先，使用者會先點擊在原生 UI 中列出的 APP 清單中 APP A 的圖示，原生 UI 首先會先檢查 APP 是否正在執行，如果 APP 已經是正在執行了，則會切換到 APP A 的 WebView，若 APP A 沒有在執行，則原生 UI 會發送一個請求到控制器並啟動 APP A，在 APP A 的 web service 啟動之前，控制器會確切的運行 web service 的執行期環境。我們將會在章節 4.6 討論有關於這方面的細節，APP A 的 web service 試運行在容器裡面，在權限上我們將 web service 設定給 U_A，而 web service 中的根目錄設定給 d_A，接著控制器會叫原生 UI 去建立一個 APP A 的 WebView，設定可以連到 APP A 的 WebView 網址以及 APP A 的 web service 的位址，最後我們就可以啟動 WebView 並顯示其 APP 的介面。

5.5、系統功能(System Feature)

系統功能是一個在 Mobile APPs 中非常重要的功能。在 Forklift 框架中，我們提供了三種模式讓前端網站腳本程式語言，如 JavaScript 等可以存取系統功能，其分別是：同步模式、非同步模式以及持久模式(persistent)，這三種模式的示意圖，同步模式通常會使用在回應較為「快速」的 API 呼叫，如果 API 功能可以在較短的時間內快速的完成工作並回應，我們則會選擇使用同步模式來處理。使用同步模式，前端 web 腳本程式語言通常會發送一個 Asynchronous JavaScript and XML (AJAX)請求到 API web service。這個是一種前端網頁的技術，使用此技術並不需要將整個頁面送至後端程式進行處理，只需要傳送必要的資料至伺服器並讓後端程式接收，處理完之後再傳回來前端所需要的資料，這樣的做法可以節省網路頻寬之外，也可以讓後端的伺服器處理效率更快以及使得伺服器負荷量也減少了。接著 API web service 會呼叫原生 API 並回傳結果給請求發送端讓它發送回應結果傳回到前端介面顯示，在系統功能上像是開關手機上的手電筒和發送一個通知到系統欄上都適合使用同步模式來處理；同步模式運作原理圖可以參考圖 5.1。

非同步模式通常是處理較「慢」的 API 呼叫，如果一個 API 呼叫的動作需要經過長時間才會有回應的話，我們使用非同步模式來處理，如果 API 呼叫結果不是直接回應一個結果而是使用回呼(callback)函式或是一個事件監聽器(event listener)做回傳的話，我們也可以使用非同步模式來做處理；這樣非同步模式運作原理圖可以參考圖 5.2。在非同步模式下，前端的網站腳本程式語言會使用 hypertext transfer protocol (HTTP) 長輪詢 (long polling)的方式進行輪詢結果直到這個較慢的 API 呼叫已經回傳結果或者是已經回傳一個回呼函式或者是一個事件監聽器為止。這類的系統功能像是使用照相機照像或是錄影都是適合使用非同步模式的；這樣的非同步模式且搭配非同步呼叫的運作原理圖可以參考圖 5.3。持久(persistent)模式是通常使用在「持續」的 API 的呼叫，如果前端的 web 腳本程式語言希望發出的請求在 API 呼叫完之後仍需要持續的接收資料的時候，我們就會需要使用持久模式。使用持久模式，前端的 web 腳本程式語言會透過 WebSocket 這個通訊協定連接 API web service，請求和回應會在 WebSocket 內部的連接中做交換。

系統的特性像是藍芽，NFC 這樣的感測器需要讀取裝置名稱，標籤等相關資料以及需要掃描網路找到可以使用的無線網路熱點等，這些動作都適合使用持久模式；持久模式的示意圖如圖 5.4 所示。

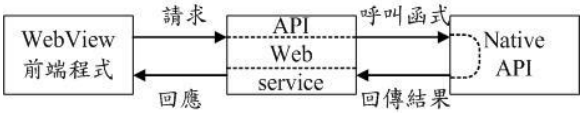


圖 5.1、同步模式(一般的 HTTP 請求)，快速呼叫

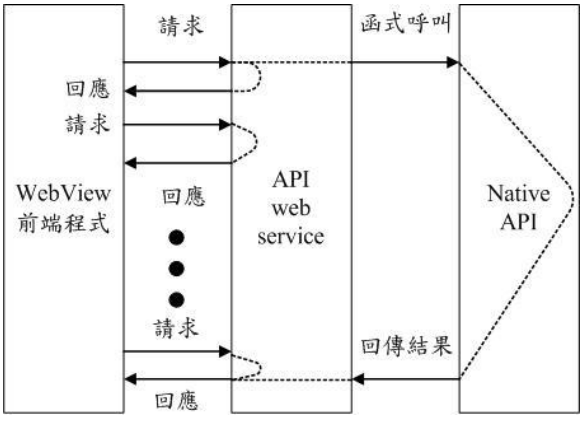


圖 5.2、非同步模式(HTTP 長輪詢)，慢的呼叫

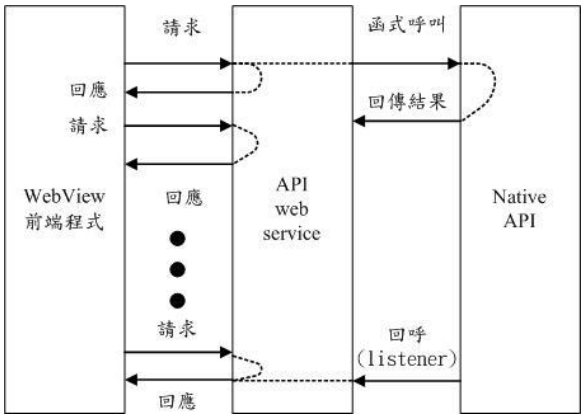


圖 5.3、非同步模式(HTTP 長輪詢)，非同步呼叫

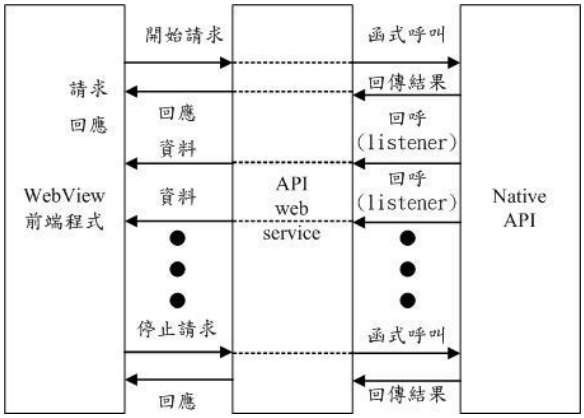


圖 5.4、持久模式(WebSocket)持續性的呼叫

5.6、安全性

我們在先前的章節中提到在限制的容器中使用控制器來安裝每個 APP，且這每個 APP 都擁有自己私人權限存取空間和執行 APP。隨著安裝與執行 APP 將只能存取自己的根目錄中相關的檔案以及大家都可以存取的全域的檔案。為了要在安全的行為下可以存取系統相關的特性，控制器需要執行一些需要為這個 APP 進行的一些相關的設定。一般來說，一個 APP 需要知道後端服務或系統功能的網址以及要有一個認證機制用來存取相關的特性或服務。

API web service 和核心 web service 分別提供了系統功能和後端服務。因為這兩個服務是由控制器把它們帶起來的，所以控制器知道這兩個服務的埠號以及位址。如果這個 APP 在設定檔中已經寫上需要請求去存取確切的系統功能或後端服務，控制器則需要儲存 API web service 的服務位址和有關的後端服務位址以及由控制器建立的每個 APP 服務位址存取設定檔，服務的設定檔也儲存在 APP 的根附錄下面。舉例來說，APP A 的服務設定檔是存在 d_A/.forklift/services。接著這個 Forklift APP 可以讀取這個服務設定檔並設定需要的系統功能和後端服務的埠號與位址。需要注意的是服務的埠號以及位址是公開的，如果這樣訊息被其他的 APPs 知道的話是沒有關係的。

與服務網址設定類似，當控制器安裝一個 APP，需要有一個認證機制(credential)去將存取相關的系統功能或服務進行初始化。這樣的認證機制我們可以分成兩種類型：前端與後端程式認證機制。後端認證機制是產生並儲存在這個目錄，且這個目錄只有在本地端的程序(process)中執行的後端程式才可以進行讀取。比如說，APP A 在 d_A/.forklift/backend/mysql.php 的這個檔案，而這個檔案是一個 PHP 的後端程式，其儲存需要後端認證機制，如此一來一個 PHP/MySQL 的應用程式會包含一些可以存取資料庫以及登入資料庫相關的資訊。要注意的是由於安全性的考量，服務的埠號，位址(address)以及一些認證機制都會在執行 APP 前更新，而在執行時期的設定時，Forklift 元件和認證機制檔案的更新都有更新認證機制檔案的權限。

至於前端的認證機制有別於後端的認證機制，前端認證機制是存在一個目錄且這個目錄可以從本地的程序和 WebView 存取。一個 APP 可能為前端認證機制目錄設定密碼，但這個動作對於 Forklift 來說是不強制的。假設 APP A 需要存取系統功能或是從前端程式來存取後端服務，像是 JavaScript 等這類的程式語言就屬於前端的程式，則圖三的架構中圖中的 API web service 和核心 web service 分別會在 d_A/.forklift/frontend/cred.api 和 d_A/.forklift/frontend/cred.core 產生與儲存前端認證機制檔案。當發送一個請求到 API web service 或核心 web service 時，一個前端程式需要發送一個 APP id 以及正確的認證機制以及服務的序列的號碼。除了檢查 APP id 和認證機制，web service 也會在請求的標頭中檢查引用，如此一來可以提高欺騙(spoof)攻擊的難度。與後端認證機制相同的是，基於安全性的考量，在執行 APP 之前，控制器有更新這個 APP 的前端認證機制文件權限。

六、實做

大部份的 Forklift 元件，可以使用標準的原生行動 APP 的原生程式實做出來。舉例來說，在 Android 系統上，使用 JAVA 做為開發原生行動 APP 的程式語言；在 iOS 系統上，我們使用 Swift 或是 Objective-C 做為開發原生 APP A 的程式語言。然而，我們需要專注在容器、APP web service 以及核心 web service 實做上。核心的 web service 與 APP web service 是建立在當今流行的 web server 之上。而一個 web server 通常包含了三種元件，分別是處理靜態網頁的 HTTP 伺服器(server)、處理動態網頁的後端程式語言以及儲存資料的資料庫系統。舉例來說，Apache、Lighttpd 或者是 thttpd 皆是 HTTP server。為了要支援數個後端的服務，Forklift 框架也提供了外掛的介面可以安裝其它客製化的服務

元件，像是 PHP、Python、Ruby 則是屬於後端程式語言，而 MySQL 則是儲存資料的關聯式資料庫系統。在每個行動平台上的 SDK 使用跨平台的編譯器來建置這些上述所提到的所有元件。

由於 Forklift 框架的安全性模型，Forklift 可能需要超級使用者權限，Forklift 安裝了每個 Forklift APP 並給予個別的權限與限制的環境，且確保屬於某個 APP 的檔案並不會讓其它的 Forklift APP 有機會可以存取到。因此，取得超級使用者權限的方式可以較容易達到將每個 Forklift APP 做權限分離與控制。在本章節中，我們會討論在實做 Forklift 框架上的兩個可能的方式。一種是使用超級使用者的權限，另一種則是沒有超級使用者權限，我們會說明不需要超級使用者權限也可以把 Forklift 框架佈署到行動裝置上。

5.1、有超級使用者權限

在行動平台上，得到超級使用者權限是有可能且對於使用者來說是普遍常見的；使用者準備好一支已經 root 過且搭載 Android 平台的行動裝置，而 iOS 平台上的行動裝置則是將其越獄(jail break)。Android 平台與 iOS 平台，這兩者都是以 UNIX 為基礎的作業系統，「root」與「越獄」的動作都是為了要取得 root 的權限。有了 root 權限，對於實做 Forklift 框架是一件很簡單的事情。要注意的是，如果我們有 root 權限的話，我們是不需要有容器的元件。這是因為 Forklift 容器提供了一些基本的動作，包含：1)建立個別的使用者，2)建立根目錄與為每個目錄的設定這個目錄擁有者的權限，3)指定使用者的權限去啟動 web service。上述的這些基本的三個動作都可以在有超級使用者權限下完成，因此容器在這個情況下是不需要的。

5.2、沒有超級使用者權限

如果 Forklift 只能運行在標準的原生行動 APP 上且沒有超級使用者權限時，容器是需要實做出來的。在這個案例中，我們需要去「欺騙」核心 web service 與 APP web service，讓它們以為在個別獨立的環境中執行。為了達到上述的目的，我們使用了一個在 Linux 系統中以 ptrace 為基礎的模擬超級使用者的技術。ptrace 是一個系統呼叫，並讓一個程序可以去觀察並控制另一個程序。追蹤器(tracer)也可以用來測試與改變「tracees」 暫存器(registers)與記憶體。這個 ptrace 系統呼叫通常被用來實做一個除錯器(debugger)或是系統呼叫追蹤器，而且藉由在使用者的空間中攔截所有系統呼叫相關的檔案系統的動作也可以用來實做一個檔案系統容器。除此之外，追蹤器也可以用來挾持使用者識別的 id 與權限控制的系統呼叫去模擬一個假的超級使用者環境；下面接著會敘述有兩種方式來與 ptrace 進行搭配使用。

第一種方式是我們使用了 Android 平台下的 Native Development Kit (NDK)並利用編譯工具在 Linux 系統下進行跨編譯的動作，我們目的是要移植常見的 web server 套件，因此我們選擇了 PHP、MySQL 以及 Lighttpd 等進行移植，因為 Android 系統上環境的問題，使用動態鏈結模式(dynamic mode)的路徑會有問題，因此我們一律將套件編譯成靜態模式(static mode)；接著，再將這些編譯好的套件可執行檔放到手機裝置上後與模擬超級使用者的工具進行搭配，便完成容器了。第二種方式是我們建立了一個小型的(minified) Linux 根檔案系統(root file system)並在檔案系統裡面安裝了所有我們需要的相關服務元件，比如說所需要使用到的 web server 相關的套件，如 Apache、PHP、MySQL 等。使用以 ptrace 為基礎的容器，我們夠執行 chroot 這個指令可以去切換到這個最小化的檔案系統中的根目錄下，並形成一個密閉的空間；在使用 chroot 指令之後的根檔案系統中進行運作。

第二種的方式有幾個好處。首先，我們的最小型的檔案系統提供了標準的 C 函式庫並可以把動作上做簡化為一個服務。其次則是這樣的做法也可以在跟檔案系統中可以較輕易的使用共享(shared)函式庫並讓程序以最小化的方式佔用記憶體；與第一種方法相較之下，執行容器的時候，每佈署一次 Forklift APPs 的時候，就需要執行一次相對應的 web server 可執行檔，再加上這些執行檔是以靜態鏈結模式編譯而成的，每次載入就需要載入一次相關函式庫而且這些函式庫無法與其他執行檔共享，進而造成記憶體上的浪費。最後是使用假的 root 環境，即使我們只是一個普通的使用者，我們仍可以提供一個較好的控制方式去處理特權(privileges)的問題。我們透過實做的方式將上述的概念做一個驗證，並得知在以 ARM 為基礎與 x86 為基礎的 Android 平台上都可以順利的執行。因此由上述的兩種實作容器的方法，我們分析與比較的結果，使用第二種方法較為合適的。

六、結論

我們展示了一個 Forklift 框架，並嘗試消除網站開發者在開發需要使用到系統相關特性與離線的行動 APP 之障礙，開發者能夠使用標準的網站前端以及後端技術和工具去建立一個 APP 並將這個 APP 安裝到 Forklift 框架中。即使這個行動裝置是在離線狀態下，還是能夠執行這個 APP。我們為了要證明我們所敘述的概念，我們實做並展示了 Forklift 框架，並讓這個框架能夠實做在已經取得超級使用者權限的 Android 與 iOS 平台上。我們也展示甚至在沒有超級使用者權限的 Android 平台上也能夠將 Forklift 框架實做出來。建立一個給 Forklift APP 的的 APP 生態系統是有可能的。然而，建立一個類似 Google Play 這樣市場相關的商店不在我們這篇論文討論範圍內，而這個議題是 Forklift 框架未來的研究工作。

參考文獻

[1] T.-M. Gr˜A¨ynli, J. Hansen, G. Ghinea, and M. Younas. Mobile APPlication platform

heterogeneity: Android vs Windows Phone vs iOS vs Firefox OS. In Proceedings of IEEE 28th International Conference on Advanced Information Networking and APPlications, pages 635–641, 2014.

[2] Apache Cordova. https://cordova.apache.org.

[3] REST. Roy Thomas Fielding. Architectural Styles and the Design of Network-based Software Architectures. The DOCTOR OF PHILOSOPHY, pages 76-105, 2000.

[4] E. Chin and D. Wagner. Bifocals: Analyzing WebView vulnerabilities in Android APPlications. In Proceedings of the 14th International Workshop on Information Security APPlications, pages 138–159, 2013.

[5] M. E. Joorabchi, A. Mesbah, and P. Kruchten. Real challenges in mobile APP development. In Proceedings of ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pages 15–24, 2013.

[6] C. Bouras, A. Papazois, and N. Stasinos. A framework for cross-platform mobile web APPlications using HTML5. In Proceedings of the 2014 International Conference on Future Internet of Things and Cloud, FICLOUD '14, pages 420–424, 2014.

[7] D. Sin, E. Lawson, and K. Kannoorpatti. Mobile Web APPs- the non-programmer's alternative to native APPlications. In Proceedings of the 5th International Conference on Human System Interactions, pages 8–15, 2012.

[8] P. Hazarika, R. R. CP, and S. Tolety. Recommendations for Webview based mobile APPlications on Android. In Proceedings of 2014 International Conference on Advanced Communication Control and Computing Technologies, pages 1589–1592, 2014.

[9] H. Heitk¨otter, S. Hanschke, and T. A. Majchrzak. Evaluating cross-platform development APProaches for mobile APPlications. In Proceedings of the 8th International Conference on Web Information Systems and Technologies, pages 120–138, 2012.

[10] J. Ohrt and V. Turau. Cross-platform development tools for smartphone APPlications. IEEE Computer, 45(9):72–79, September 2012.

[11] S. Xanthopoulos and S. Xinogalos. A comparative analysis of cross-platform development APProaches for mobile APPlications. In Proceedings of the 6th Balkan Conference in Informatics, pages 213–220, 2013.

[12] H. Heitk¨otter, S. Hanschke, and T. A. Majchrzak. Evaluating cross-platform development APProaches for mobile APPlications. In Proceedings of the 8th International Conference on Web Information Systems and Technologies, pages 120–138, 2012.

[13] S. Cheshire and M. Krochmal. Multicast DNS. RFC 6762, February 2013.