JavaScript 不是我的菜，但我愛（下）

如果你也是 Java 開發者，不小心也喜歡上 JavaScript，卻有種看得到卻吃不到的感覺，那就讓我們一起來玩「Java + Script」吧！

用 JavaScript 實作 Java 程式

儘管 JavaScript 與 Java 是兩種八竿子打不著的語言，但你真的可以把 JavaScript 當做 Java 的 Scripting 來撰寫程式碼，實際上這樣的應用早已存在。

Java 虛擬機器（JVM，Java Virtual Machine）的設計支援多語言（Polyglot programming），只要能將程式原始碼編譯成支援的 Byte Code，就能在 JVM 執行。但是多年來 JDK 就只提供 Java 語言，讓開發者並沒有第二種語言可選；並不像.NET 可以有 C#、VisualBasic 與 JScript 等選項。

熟悉 Java 新趨勢的開發者，可能早已聽過或實作過 Groovy、Scala 或 Clojure 等新程式語言，或是 JRuby、Jython 這些被移植到 JVM 的語言。這些程式語言可被稱為 JVM Scripting Language。這類動態腳本語言的優勢，剛好可以彌補 Java 的諸多不足，帶給程式設計師更多的選擇。在 Java 世界使用更敏捷的 Script 語言寫程式，已經逐漸廣泛被接受，由 Java 社群所提出的規範即 JSR-223。

- JSR-223: Scripting for the Java Platform（http://goo.gl/jxpq8）
- List of JVM languages, Wikipedia（http://goo.gl/Xxlxx）
- Java Scripting Programmer's Guide（http://goo.gl/KJcx8）
- JSR 223: Scripting for the Java Platform（http://goo.gl/pzBVr）

在 JVM 執行其他程式語言，以往需要引入額外的函式庫，例如 Groovy 需要 groovy-all.jar 這個函式庫檔案。相當有趣的是，目前只有一種 Scripting Language 不需要添加額外的函式庫，那就是 JavaScript 語言。

Sun（已被 Oracle 併購）從 JDK 6 開始加入 Script Engine 的支援，讓 Java 程式在執行 Scripting 語言時，有標準的 API 實作可循，解決過去不同語言有不同 API 規格造成的困擾。JDK 內建的 JavaScript Engine 是由 Mozilla 開發的 Rhino 開源專案，這是一個開發已久、穩定適合實際應用的 JS 引擎。

- ScriptEngine, Java SE 6 API Doc（http://goo.gl/1BE9K）
- Mozilla Rhino https://github.com/mozilla/rhino

因此在 Java SE 6+ 以上版本，我們只要利用 ScriptEngine API 就能直接執行一段 JavaScript 程式碼。

```
ScriptEngineManager factory = new ScriptEngineManager();
ScriptEngine engine = factory.getEngineByName("JavaScript");
engine.eval("print('Hello, World')");
```

或者執行一個放置於外部的.js 程式檔案。

```
engine.eval(new java.io.FileReader("script1.js"));
```

如此一來，Java 開發者可藉由 JavaScript 的支援，得到更有彈性的架構設計方式，開發效率也有機會因此提高。

舉例來說，過去我們幫某廠商實作 SMS 簡訊通知系統，我們使用 java.net.* 的 Network API 實作相當穩定的 TCP/UDP Server，來接收第三方系統的發送簡訊通知；但是在發送簡訊的程式實作上，由於不同家電信商會有不同規格，且實作過程需要經常修改、測試，未來也很有機會再擴充支援新的電信商，這個部分就很適合採用 JavaScript 或其他 JVM Scripting 語言來處理。

因為 Java 專案的建置、佈署過程較麻煩，重新編譯過的程式，當然就要重新執行才能生效。但開發過程如果只因為一小段程式的修改，就要整個建置過程重新跑一次，未免也太過繁複。加入使用 ScriptEngine 方式的實作後，我們只要修改一段程式就能立即測試其結果，就能大幅增加開發效率，發揮 Scripting 語言的優勢。

這種混搭架構讓 Java 既有的優點保留，也能同時享有 JavaScript 開發的好處。以下是實作方式的參考程式碼範例：

在 ExecuteProgram.java 中，依照功能名稱執行 JavaScript 程式碼，並傳遞一個 addr1 變數：

```
ScriptEngineManager manager = new ScriptEngineManager();
ScriptEngine engine = manager.getEngineByName("JavaScript");
engine.put("addr1", "127.0.0.1");
engine.eval(new     java.io.FileReader("js- lib/"+funcName
+".js"));
```

如此一來，在 js-lib/dump.js 的程式碼中，即可使用 addr1 這個變數。

```
var f = function(x) {
    print('x is ' + x);
};
f(addr1);
```

用 Java Web Start 技術啟動 JavaScript 程式

瀏覽器中的 JavaScript Engine 就算再快再好，基於安全性等考量，當然不可能直接深入到客戶端的系統中，執行一些例如檔案操作的危險任務!

但是對於某些特定用途，例如企業內的資訊系統，我們可能希望網站的功能不要被瀏覽器侷限。過去使用的方法通常是 ActiveX 或 Java Applet 技術。

其實除了上述兩種選擇，已經存在相當長一段時間的 Java Web Start 技術，應該也是一種可行的解決方案。這項技術讓一個由 Java 撰寫的程式，可以方便透過網路佈署到客戶端執行，並享有與 Desktop Application 幾乎相同的權限與功能，包括使用 Java Swing 產生視窗圖形介面等，想要在客戶端運行一個嵌入式的網路應用服務也不是問題。

- 何為 Java Web Start，以及啟動方式？（http://goo.gl/nBznX）

但是過去在開發 Java Web Start（簡稱 JWS）程式時，從編譯、建置、測試到佈署，一直都是過程瑣碎麻煩的事情，因為修改好的 Java 程式原始碼，需要先經過編譯成為 Byte Code（*.class 檔案），然後再打包成標準的 JAR（*.jar）檔案，經過 jarsigner 工具簽章完成，最後佈署到網頁伺服器，才能利用網頁執行 JWS 程式的測試。

儘管加上一些 Unit Test 與其他方法，可以簡化傳統 Java 程式開發過程的測試麻煩；但開發效率仍然遠不及 Scripting 語言。

在這種情況下，魚與熊掌如何兼得？也許 JavaScript 是一個答案。

使用 JavaScript 來開發 Java Web Start 應用程式，方法其實相當簡單，就是將一個可以從特定路徑（通常是遠端 URL 位址）執行 JavaScript 原始碼的 Java 程式，先包裝成一個標準 JWS 程式，如此只需要佈署一次成功後，功能的實作就能交給更容易修改並立即測試的 JavaScript 程式負責。

我們在開發 CodeCanaan 教學系統時，已驗證利用 JWS + Script 混搭實作的可行性，成功將一個內嵌 Web Server（Jetty）的程式利用 JWS 方式發佈，整個打包完的 JAR 檔案體積僅約 10MB 左右，第一次下載到客戶端執行後，日後便可利用這個 JWS 程式執行遠端的 Script 檔案。

發佈內嵌 Web Server 的 JWS 程式到客戶端，是一種有趣的作法，利用客戶端的 JWS 程式來執行指定的 JavaScript 程式，就可以透過本地端（localhost）的 HTTP Port 與遠端網頁的 JavaScript 進行通訊。如此一來，在網頁上執行的 JavaScript 可以利用 AJAX 技術與遠端網頁伺服器通訊，同時也透過同樣的方法，達成操作客戶端執行的 JavaScript 程式，無論系統存取權限或應用範圍，都與本地端執行 Java 程式相同。

也許 JavaScript 也不是你的菜，但不管愛或不愛，遇到適合讓它派上用場的時機，可就千萬別錯過啦！

完…

@作者 lyhcode 目前從事程式設計教學與顧問工作。（http://lyhcode.info/）