Chapter 9

# Broadcast Receiver 應用元件

我們曾在前面章節介紹過，Activity 程式承襲了 Java 傳統視窗程式的「委任式事件處理模式」。本章將再介紹另一種「事件處理」，不同的是，這種事件處理屬於系統層級，而非應用程式層級。

## 9.1 Android 平台對應用程式的廣播

何謂系統層級的事件呢？舉例來說，系統完成啟動、使用者調整時間日期、電池電量過低等，皆屬於這一類型的事件。一旦發生此類型事件時，Android 平台便會藉由廣播（broadcast）的方式，通知所有註冊欲處理這些事件的接收程式（receiver）。

為實現此機制，Android 平台為每一種事件，定義對應的 Action 常數。接收程式僅需在自己的 AndroidManifest.xml 中，宣告欲接收事件的 Action 常數，在程式安裝的過程中，也會同時完成註冊的工作。一旦事件發生，Android 平台便廣播給所有註冊對應 Action 常數的接收程式。下表為 Android 平台標準的廣播 Action 常數：

| 常數名稱 | 常數值 | 功能說明 |
|---|---|---|
| ACTION_TIME_TICK | android.intent.action.TIME_TICK | 系統時間改變，每分鐘發送一次 |
| ACTION_TIME_CHANGED | android.intent.action.TIME_SET | 設定時間 |
| ACTION_TIMEZONE_CHANGED | android.intent.action.TIMEZONE_CHANGED | 時區變更 |
| ACTION_BOOT_COMPLETED | android.intent.action.BOOT_COMPLETED | 系統完成啟動 |
| ACTION_PACKAGE_ADDED | android.intent.action.PACKAGE_ADDED | 加入新的應用程式套件 |
| ACTION_PACKAGE_CHANGED | android.intent.action.PACKAGE_CHANGED | 現存套件變更，如元件被停止 |
| ACTION_PACKAGE_REMOVED | android.intent.action.PACKAGE_REMOVED | 應用程式套件被移除 |

| 常數名稱 | 常數值 | 功能說明 |
|---|---|---|
| ACTION_PACKAGE_RESTARTED | android.intent.action.PACKAGE_RESTARTED | 重啟應用程式 |
| ACTION_PACKAGE_DATA_CLEARED | android.intent.action.PACKAGE_DATA_CLEARED | 清除應用程式的資料 |
| ACTION_UID_REMOVED | android.intent.action.UID_REMOVED | 移除使用者 ID |
| ACTION_BATTERY_CHANGED | android.intent.action.BATTERY_CHANGED | 電池狀態改變 |
| ACTION_POWER_CONNECTED | android.intent.action.ACTION_POWER_CONNECTED | 外部電源連接 |
| ACTION_POWER_DISCONNECTED | android.intent.action.ACTION_POWER_DISCONNECTED | 外部電源移除 |
| ACTION_SHUTDOWN | android.intent.action.ACTION_SHUTDOWN | 準備關機之際 |

對 Action 常數更詳盡的介紹，有興趣的讀者可自行參考 Android 官網：

- http://developer.android.com/reference/android/content/Intent.html

接下來，本節將嘗試撰寫一個接收「外部電源連接/移除」事件的範例。如下為接收程式的原始碼：

```
01 package com.myreceiver;
02
03 import android.content.*;
04 import android.widget.Toast;
05
06 public class PowerReceiver extends BroadcastReceiver {
07
08  //接收到廣播的訊息
09  public void onReceive(Context context, Intent intent) {
10   Toast.makeText(context, "電源事件發生", Toast.LENGTH_LONG).show();
11  }
12}
```

簡單的很，該類別繼承了 BroadcastReceiver，同時覆寫 onReceive 成員函數，便大功告成。可想而知，onReceive 也是一個回呼函數（call back method），當所註冊的事件發生時，Android 平台便呼叫接收程式的 onReceive 函數。在本節範例中，onReceive 函數將會透過 Toast 元件，在螢幕上顯示一個長訊息給手機的使用者。

另有一件事情必須做的就是建立接收程式與事件間的關聯，即為進行廣播通知的註冊工作，此工作不難，僅需在 AndroidManifest.xml 做以下之宣告即可：

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
03  package="com.myreceiver"
04  android:versionCode="1"
05  android:versionName="1.0">
06
07  <uses-sdk android:minSdkVersion="8" />
08
09  <application android:icon="@drawable/icon"
10               android:label="@string/app_name">
11   <receiver android:name="PowerReceiver">
12    <intent-filter>
13     <action android:name="android.intent.action.ACTION_POWER_CONNECTED"/>
14    </intent-filter>
15    <intent-filter>
16     <action android:name="android.intent.action.ACTION_POWER_DISCONNECTED"/>
17    </intent-filter>
18   </receiver>
19  </application>
20 </manifest>
```

在此我們藉由 <receiver> 標籤設定廣播接收程式為何，再透過其下的子標籤 <intent-filter>，進行註冊宣告。如此一來，一旦發生「外部電源連接」或「外部電源移除」的事件時，底層的 Android 平台便會通知 PowerReceiver 接收程式（註：回呼 PowerReceiver 的 onReceive 成員函數）。

倘若將上述範例程式，安裝至手機後，在抽拔與連接外部電源時，便可以看到預期的效果，即顯示 Toast 長訊息。然若只想透過 Android 模擬器來測試，該如何來觀察執行結果呢？

讀者可先開啟一個 DOS 視窗，再利用 telnet 連接到手機模擬器所提供的服務埠號。（注意：Android 模擬器預設的埠號為 5554，若讀者自行調整過，請設定符合環境的埠號。）

```
telnet localhost 5554
```

一旦正確連接至手機模擬器後，DOS 視窗應出現如下之訊息：

```
Android Console: type 'help' for a list of commands
OK
```

此時，讀者可再嘗試輸入下列指令，模擬將手機外部電源連接或是移除的動作：

```
power ac <on/off>
```

若執行成功，DOS 視窗將出現如下之訊息：

```
power ac off
OK
```

意謂著進行模擬外部電源連接與移除的動作，此時，手機模擬器的執行結果，應如右圖所示：

## 9.2 應用程式間的廣播

應用程式除了接收來自 Android 平台的廣播事件外，應用程式與應用程式間也可進行廣播通知的動作。以下我們先撰寫一個廣播發送程式，於其中加入一個發送廣播通知的按鈕元件：

```
01 <Button
02  android:id="@+id/btn"
03  android:layout_width="fill_parent"
04  android:layout_height="wrap_content"
05  android:text="發送廣播"/>
```

同時，在 Activity 程式中取得該按鈕元件的物件實體，同時委任其點選事件：

```
01 package com.mysender;
02
03 import android.app.Activity;
04 import android.content.Intent;
05 import android.os.Bundle;
06 import android.view.View;
07 import android.view.View.OnClickListener;
08 import android.widget.Button;
09
10 public class MySender extends Activity {
11  private static final String MyAction1 = "com.mybroadcast.action.Action1";
12
13  public void onCreate(Bundle savedInstanceState) {
14    super.onCreate(savedInstanceState);
15    setContentView(R.layout.main);
16
17    //取得發送廣播系統的按鈕
18    Button btn = (Button)this.findViewById(R.id.btn);
19
20    //委任點選按鈕的事件
21    btn.setOnClickListener(new OnClickListener(){
22     public void onClick(View view) {
23      //建立包裹廣播資訊的物件
24      Intent intent = new Intent();
25
```

```
26        //設定物件的識別碼
27        intent.setAction(MyAction1);
28
29        //設定要廣播的資訊
30        intent.putExtra("myMsg", "天雨路滑，放慢車速~");
31
32        //進行廣播
33        MySender.this.sendBroadcast(intent);
34       }
35     });
36   }
37 }
```

同樣地，此廣播發送程式也將所欲傳遞的訊息包裹在 Intent 物件內。不過，比較不一樣的是，使用了 Intent 物件的 setAction 成員函數，設定廣播事件的 Action 常數。如上所示，本節所設定的 Action 常數其內容為 “com.mybroadcast.action.Action1” ，稍後凡是宣告與註冊此常數值的接收程式，都將接收到來自本範例發送程式的廣播通知。最後，再利用繼承自 Context 類別的 sendBroadcast 成員函數，將訊息給廣播出去。右圖即為發送程式的執行結果：


接下來繼續撰寫廣播的接收程式。如同一開始時所介紹的，接收程式必須在 AndroidManifest.xml 中，宣告所欲接收的事件的 Action 常數：

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
03       package="com.myreceiver"
04       android:versionCode="1"
05       android:versionName="1.0">
06  <uses-sdk android:minSdkVersion="8" />
07
08  <application android:icon="@drawable/icon"
09               android:label="@string/app_name">
```

```
10   <receiver android:name="MyReceiver">
11    <intent-filter>
12     <action android:name="com.mybroadcast.action.Action1"/>
13    </intent-filter>
14   </receiver>
15  </application>
16 </manifest>
```

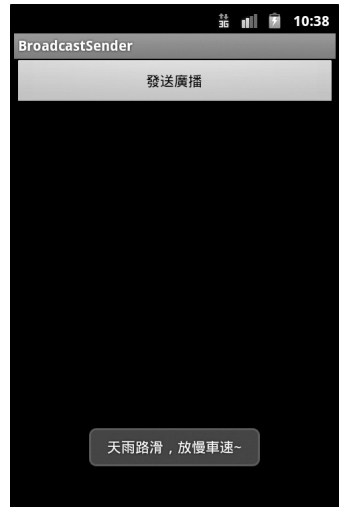
如上我們利用 receiver 標籤的 android:name 屬性，設定接收程式的名稱，再利用 intent-filter 標籤的 action 子標籤，設定事件的 Action 常數，如此一來，當完成安裝接收程式時，也會同時將相關的資訊註冊在 Android 平台之上。

接著便可開始撰寫接收程式。這一次，為能觀察接收程式的生命週期及其運作原理，因此，特地在接收程式的建構者函數中，加入計數物件實體數量的程式碼，同理，也在 MyReceiver 類別的 finalize 函數，加入扣除物件實體數量的程式碼。如此一來，透過 Log 的觀察，便可知道在記憶體中，目前有多少個 MyReceiver 接收程式的物件實體。

```
01 package com.myreceiver;
02
03 import android.content.*;
04 import android.util.Log;
05 import android.widget.Toast;
06
07 //沒有 GUI，單純的廣播接收器
08 public class MyReceiver extends BroadcastReceiver {
09  private static int cnt = 0;
10
11  public MyReceiver() {
12   super();
13   MyReceiver.cnt++;
14   Log.v("broadcast", "MyReceiver contruct, count:" + MyReceiver.cnt);
15  }
16
17  //接收到廣播的訊息
18  public void onReceive(Context context, Intent intent) {
19   String msg = intent.getStringExtra("myMsg");
20   Log.v("broadcast", "receive:" + msg);
```

```
21   Toast.makeText(context, msg, Toast.LENGTH_LONG).show();
22  }
23
24  protected void finalize() throws Throwable {
25   super.finalize();
26   MyReceiver.cnt--;
27   Log.v("broadcast", "MyReceiver finalize, count:" + MyReceiver.cnt);
28  }
29 }
```

請注意，本範例的廣播發送程式，其套件命名為 com.mysender，而接收程式的套件則為 com.myreceiver。如此一來，更可模擬兩個完全不同的 Android 應用程式間的廣播傳遞。這也意謂著，讀者在測試範例程式時，必須分兩次安裝才行。本範例執行結果如右圖：




倘若使用者多按幾次的發送按鈕後，接收程式在記憶體中的變化又會是如何呢？從 LogCat 所得到的觀察如下：

```
VERBOSE/broadcast(300): MyReceiver contruct, count:1
VERBOSE/broadcast(300): receive:天雨路滑，放慢車速~
VERBOSE/broadcast(300): MyReceiver contruct, count:2
VERBOSE/broadcast(300): receive:天雨路滑，放慢車速~
VERBOSE/broadcast(300): MyReceiver finalize, count:1
VERBOSE/broadcast(300): MyReceiver finalize, count:0
VERBOSE/broadcast(300): MyReceiver contruct, count:1
VERBOSE/broadcast(300): receive:天雨路滑，放慢車速~
VERBOSE/broadcast(300): MyReceiver contruct, count:2
VERBOSE/broadcast(300): receive:天雨路滑，放慢車速~
VERBOSE/broadcast(300): MyReceiver contruct, count:3
VERBOSE/broadcast(300): receive:天雨路滑，放慢車速~
……
```

Chapter 13

# 架構 Hadoop 雲端系統

經過前面十來章 Android 相關技術的洗禮之後，從本章開始，即將邁入下一段旅程，也就是另外一個重頭戲—雲端系統的架設。欲從無到有實作一套雲端平台，不是一件容易的事，這需要投入龐大的人力、物力才有“機會”（意謂還不見得會成功）。因此，本章將介紹目前市面上，最廣被採用，同時為開放源碼的雲端系統框架（framework）－Hadoop，做為各位讀者躍上雲端的叩門磚。

## 13.1 Hadoop 漫談

Hadoop 是 Apache 軟體基金會（Apache Software Foundation）所支援的開放原始碼專案（open source project）之一，最主要的開發人員 - Doug Cutting，本身是一位在搜尋引擎界相當知名的工程師，在 2004 年，Google 發表了 MapReduce 演算法後，Cutting 為了實作 Nutch 搜尋引擎，提出了 Hadoop 架構。

什麼是 Hadoop 呢？Hadoop 是一個分散式運算的框架，它最重要的核心處理邏輯，乃是架構在 Map/Reduce 的理論基礎上，而 Map/Reduce 則是一種分而自治（divid and conquer）的思維。簡單地說，透過 Hadoop 提供的函式庫，程式設計師可以輕易地，將大量的資料分配到不同的伺服器進行運算，經由整合而得到最終的結果。伺服器的數量可以隨著需求而增加，從一台到上千台機器的串接皆無問題。而最重要的，也是最符合雲端運算精神的就是這些機器可以只是一般且平價的電腦設備即可，換句話說，雲端運算順應了當前流行的平價奢華概念，系統建置者無需購置昂貴的伺服器設備，即可享受高速運算的好處。

最知名的案例，就是在 2008 年 2 月 19 日，Yahoo 成功利用 Hadoop，串接 10,000 個微處理器核心，並實際應用在專案系統上。除此之外，Hadoop 也可以自動偵測與處理例外事件，並視伺服器失效為常態，因此便可以大幅提高服務的可用率與可靠率。總而言之，雲端運算該有的功能與條件，Hadoop 皆已具足。

Yahoo 是整個 Hadoop 專案中，注入最多資金與研究的支持者，甚至於直接聘請 Doug Cutting，在 Yahoo 內部大量導入 Hadoop，進行各種網路使用者行為的分析。目前在 Yahoo 內部，大約有 4 萬台電腦正在執行 Hadoop（超過 100,000 顆 CPU），其中最大的雲端叢集乃由 4 千 5 百個節點所串接而成，運用在廣告行為分析與網頁搜尋上。

基本上，Hadoop 包含下列三大部份：

1. **Hadoop MapReduce**：是一個可以對大量資料進行分散式處理的軟體框架，與 Google 的雲端架構一樣，二者皆是植基於 Map/Reduce 技術。
2. **Hadoop Distributed File System**：簡稱 HDFS，乃是參考 Google 的檔案系統（Google File System, GFS）所實作而成的。它是一個分散式的檔案系統，可以藉由高效率的方式，存取應用程式的資料檔案。
3. **Hadoop Common**：包含一些工具程式，以及其他 Hadoop 子專案。

本書撰寫之際，Hadoop 最穩定的版本為 0.20.203.0，最近一次的改版是在 2011 年 5 月。雖然，官網目前也提供 0.21.X 可以下載，但由於尚未通過嚴謹的測試，因此，本書將以 0.20.203.0 版做為主要的示範對象。順便一提的是，從 0.21.X 開始，HDFS 與 MapReduce 已經從 Hadoop 核心分開，而成為獨立的子專案，其餘的功能則歸類在 Hadoop Common 之中。

Hadoop 是以 Java 語言所開發的，自然承襲跨平台的優點。然在 Hadoop 官方文件中，仍建議採用 GNU/Linux 做為開發或成品的運作平台。因為 Hadoop 在 GNU/Linux 環境中，至少進行過 2000 個節點以上的串接測試；反之，在 Windows 平台上，卻尚未見較嚴謹的測試報告，所以 Windows 平台目前僅被建議做為開發測試環境之用。

此外，Hadoop 若想在 Windows 平台運行，尚須安裝如 Cygwin 等軟體，所涉及的要素或許更加複雜，基於此，本書將採用 2011/05/24 所發行的 Fedora 15，做為各項展示與測試平台之用。

### 13.2.3 執行虛擬分散模式

前一節的單機模式範例，其為單一行程，當然尚未聞雲端運算之味，本小節將透過 Hadoop 所提供的「虛擬分散模式」，介紹如何架構分散式的作業環境。同樣地，Hadoop 官網也提供測試的步驟範例，計有下列幾個步驟：

STEP 1 修改參數檔

STEP 2 設定無密碼之 ssh 連線

STEP 3 格式化分散式檔案系統

STEP 4 啟動 Hadoop

STEP 5 執行分散式運算

STEP 6 停止 Hadoop

讀者目前僅需依照步驟執行，相關的說明待稍後再解釋。

STEP 1 修改參數檔

請先調整下列各個參數檔的內容：

**conf/core-site.xml**

```
01 <?xml version="1.0"?>
02 <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
03 <configuration>
04  <property>
05   <name>fs.default.name</name>
06   <value>hdfs://localhost:9000</value>
07  </property>
08 </configuration>
```

**conf/hdfs-site.xml**

```
01 <?xml version="1.0"?>
02 <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
03 <configuration>
04  <property>
05   <name>dfs.replication</name>
```

```
06   <value>1</value>
07  </property>
08 </configuration>
```

■ conf/mapred-site.xml

```
01 <?xml version="1.0"?>
02 <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
03 <configuration>
04  <property>
05   <name>mapred.job.tracker</name>
06   <value>localhost:9001</value>
07  </property>
08 </configuration>
```

若欲從虛擬分散模式回到單機模式，只需刪除加到參數檔中的資料即可。

STEP 2 設定無密碼之 ssh 連線

緊接著，設定以 ssh 連線至本地端電腦時，無需使用密碼驗證（passphraseless ssh）的工作模式。由於 sshd 提供多種身份驗證的方式，包括使用公鑰以及密碼等，因此在預設的情況下，若輸入 ssh localhost 欲進行連線時，系統在判斷尚無公鑰存在時，會請求使用者輸入密碼，參見如下：

```
The authenticity of host 'localhost (::1)' can't be established.
RSA key fingerprint is
04:3e:85:0e:de:45:62:02:75:74:84:cf:c9:37:a2:2b.
Are you sure you want to continue connecting (yes/no)?
```

此時請先回應“yes”；接著系統便會要求使用者輸入密碼：

```
Warning: Permanently added 'localhost' (RSA) to the list of known hosts.
allan@localhost's password:
```

請先不要輸入密碼，改以按下「Ctrl ＋ C」，離開登入的動作，接著再執行下列指令，用以建立金鑰對，並準備關閉密碼功能：

```
$ ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
```

此時，便會建立金鑰對如下：

```
Generating public/private dsa key pair.
Your identification has been saved in /home/allan/.ssh/id_dsa.
Your public key has been saved in /home/allan/.ssh/id_dsa.pub.
The key fingerprint is:
65:e7:61:c1:c3:dd:47:3d:28:55:2e:e2:14:0c:1b:33
allan@linux1.freejavaman
The key's randomart image is:
+--[ DSA 1024]----+
|         Eo+ooo+o|
|          =o=oo.+|
|         .oo=o .o|
|         oo+...  |
|        S  ..    |
|                 |
|                 |
|                 |
|                 |
+-----------------+
```

所產生的金鑰對，會分別被儲存在 id_dsa 與 id_dsa.pub 兩個檔案中。接著再執行下列複製指令：

```
$ cp ~/.ssh/id_dsa.pub ~/.ssh/authorized_keys
```

此時公鑰將被複製到 authorized_keys 檔案內，往後若再執行 ssh localhost 指令，便不會再要求使用者輸入密碼了。

若此時還是無法進行無密碼的本端連線時，使用者可以執行 ssh -vvv localhost 指令，將整個連線交握（hand shaking）過程的 log 顯示在螢幕上，來判斷問題的環節出現在什麼地方。

一般來說，可能的問題之一是 sshd 並未開啟公鑰驗證的功能。此時，請先轉換成 root 的身份，並修改/etc/ssh/sshd_config 檔，將其中的公鑰驗證功能開啟：

```
PubkeyAuthentication yes
```

sshd 的組態設定完畢之後，毋須將整個作業系統重開，取而代之的，僅需執行下列指令，即可重新啟動 sshd 服務，執行新設定的功能。

```
$ service sshd restart
```

STEP 3 格式化分散式檔案系統

為因應分散式檔案存取之需，吾人必須依照 Hadoop 的規範，對資料儲存目錄進行格式化的工作，此項格式化工作不難，僅需執行下列指令即可：

```
$ bin/hadoop namenode -format
```

執行結果之輸出如下所示：

```
INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = linux1.freejavaman/127.0.0.1
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 0.20.203.0
STARTUP_MSG:   build =
http://svn.apache.org/repos/asf/hadoop/common/branches/branch-0.20-
security-203 -r 1099333; compiled by 'oom' on Wed May  4 07:57:50
PDT 2011
************************************************************/
Re-format filesystem in /tmp/hadoop-allan/dfs/name ? (Y or N) Y
INFO util.GSet: VM type       = 32-bit
INFO util.GSet: 2% max memory = 19.33375 MB
INFO util.GSet: capacity      = 2^22 = 4194304 entries
INFO util.GSet: recommended=4194304, actual=4194304
INFO namenode.FSNamesystem: fsOwner=allan
INFO namenode.FSNamesystem: supergroup=supergroup
INFO namenode.FSNamesystem: isPermissionEnabled=true
INFO namenode.FSNamesystem: dfs.block.invalidate.limit=100
INFO namenode.FSNamesystem: isAccessTokenEnabled=false
accessKeyUpdateInterval=0 min(s), accessTokenLifetime=0 min(s)
INFO namenode.NameNode: Caching file names occuring more than 10 times
INFO common.Storage: Image file of size 111 saved in 0 seconds.
.........
```

若是在格式化過程中，沒有發生任何錯誤，則最後將在/tmp/hadoop-**帳號**/dfs/目錄中，建立資料儲存空間，此處目錄名稱中的“帳號”，即為執行格式化指令的系統帳號。需注意的是，Hadoop 在運作時，請勿執行格式化的動作，否則運算所需的暫存資料也將一併被刪除。

STEP 4 啟動 Hadoop

完成格式化的工作後，即可試著啟動 Hadoop 了。請切換到 Hadoop 工作目錄，並執行下列指令：

```
$ bin/start-all.sh
```

在啟動的過程中，可能會遇到下列錯誤訊息：

```
localhost: Unrecognized option: -jvm
localhost: Could not create the Java virtual machine
```

欲探究此問題發生的原因，必須檢查 hadoop 啟動檔的內容：

```
if [[ $EUID -eq 0 ]]; then
  HADOOP_OPTS="$HADOOP_OPTS -jvm server $HADOOP_DATANODE_OPTS"
else
  HADOOP_OPTS="$HADOOP_OPTS -server $HADOOP_DATANODE_OPTS"
Fi
```

如上，原來錯誤發生的主要原因在於，若是使用 root 帳號啟動 Hadoop 時，啟動檔中會帶入 jvm 參數，然而，並不是每一種 JDK 都支援 jvm 參數。解決的方法只要另外建立一個系統帳號，再由該系統帳號重新啟動 Hadoop 即可。當然，直接修改啟動檔內容，也是一種可行的解決方案。建立一組新的系統帳號，可以參考下列指令：

```
$ useradd hadoop
$ passwd hadoop
```

需留意的是，建立新的帳號後，亦須對該系統帳號，設定無須密碼的 ssh 連線；同時，透過新建立的帳號，來格式化分散式檔案系統。

除了上述原因之外，若是配置給 JVM 的記憶體過大時，在啟動過程中，也會引發例外事件。此時，就必須調整下列兩個設定檔的內容，用以縮小記憶體的配置空間：

■ conf/hadoop-env.sh

```
# The maximum amount of heap to use, in MB. Default is 1000.
export HADOOP_HEAPSIZE=1000
……
```

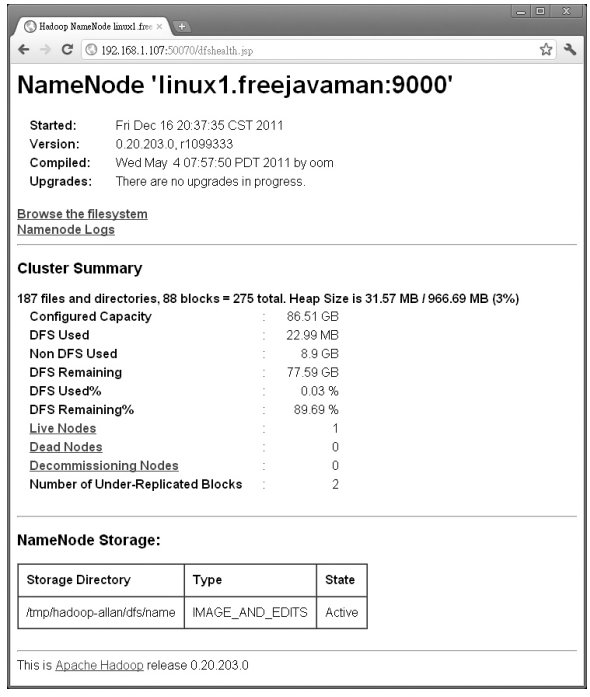
■ bin/hadoop

```
JAVA_HEAP_MAX=-Xmx1000m
……
```

在啟動 Hadoop 的過程中，若沒有顯示任何錯誤訊息，那就表示啟動成功了。Hadoop 預設會將 Log 資料，儲存在 Hadoop 工作目錄的 logs 子目錄之中，所有與除錯有關的資訊，皆可以在這裡找到。

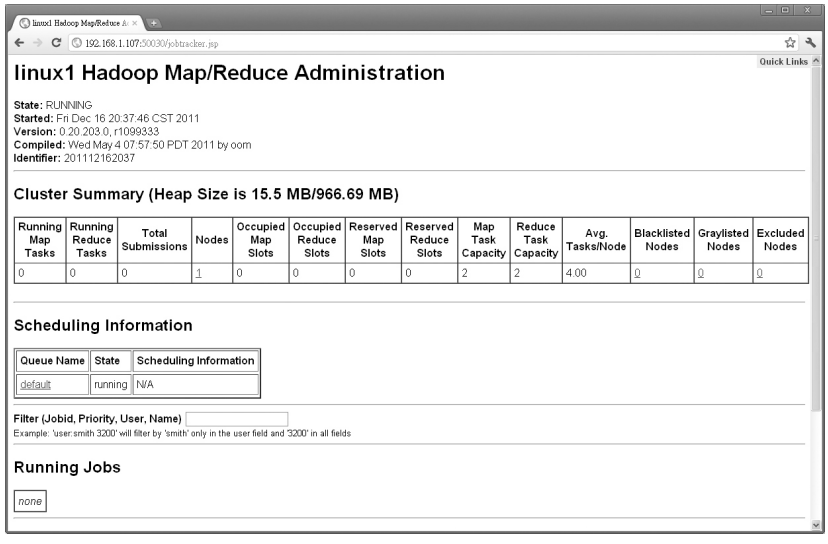
Hadoop 啟動成功之後，預設會同時開啟兩個 HTTP 服務，分別是 NameNode 與 JobTracker，這是為了用來管理 Hadoop 系統的 Web 界面。請使用瀏覽器分別連接至下列網址（註：筆者測試主機的 IP 為 192.168.1.107，讀者可根據所需自行調整 IP）。

- NameNode 服務：
  http://192.168.1.107:50070/dfshealth.jsp




■ JobTracker 服務：

http://192.168.1.107:50030/jobtracker.jsp



STEP 5 執行分散式運算

延續前一節的範例，現在試著將它在分散式環境來運行。首先，請執行下列指令，用來在分散式檔案系統中建立指定的目錄，並且將資料檔案複製到該目錄上（註：conf 為資料檔案的來源目錄，input 為目的目錄）。

```
$ bin/hadoop dfs -mkdir input
$ bin/hadoop dfs -put conf input
```

接著，再執行範例程式：

```
$ bin/hadoop jar hadoop-examples-*.jar grep input output 'dfs[a-
z.]+'
```

讀者可以發現，執行範例程式的指令，與單機模式完全一模一樣。這是一個吸引人的情況，表示程式可以不用做任何修改，也不用重新編譯，只需要透過設定的方式，就可以決定程式是要在單機模式或分散式運算模式之下執行。

雖然目前範例程式已在分散式環境之下執行，但是所花費的執行時間，卻比單機模式還要久，其緣由主要是因為在目前所建置的分散式環境中，只有一個計算節點，而運算工作又會先經過 map/reduce，將資料進行拆解與加總，因此，自然會比單機模式，增加許多額外的工作項目（overhead），進而拖累整體執行時間。

倘若所進行運算的是巨量的資料集合，同時節點數也配合增加，那麼就可以真正看出平行運算的執行效能了。運算完成之後，請再執行下列指令，將執行結果從分散式的檔案系統，複製到所指定的目錄中：

```
$ bin/hadoop dfs -get output output
```

以 cat 指令觀看執行結果：

```
$ cat output/*
```

得到的是與單機模式一樣的執行結果：

```
1     dfsadmin
```

STEP 6 停止 Hadoop

若欲停止 Hadoop，可以如下之指令為之：

```
$ bin/stop-all.sh
```

其停機過程所示之訊息如下：

```
stopping jobtracker
localhost: stopping tasktracker
stopping namenode
localhost: stopping datanode
localhost: stopping secondarynamenode
```

## 13.3 Map/Reduce 運作原理

讀者經由前一節的介紹已習得如何讓 Hadoop 執行在「單機模式」或「虛擬分散模式」之術，當然想進一步瞭解真正執行在多台伺服器上的「分散模式」；在進入“分散式的世界”之前，我們得先來探討 Hadoop 中，實作分散式運算的核心，也就是 Map/Reduce 技術。

補充說明

1：有些書籍翻譯成“映射/歸納”，本書擬採用原文，較為貼切。

2：本書中，“Map/Reduce”指的是一種演算邏輯。而“MapReduce”則是指 Hadoop 根據此演算邏輯的實作品。Map 是指映射運算，而 reduce 則是歸納運算。

Map/Reduce 並不是新的技術或是思維，它的出現已經有 50 多年的歷史，最早是由提出“人工智慧（Artificial Intelligence）”一詞，也就是 1956 年 Turing 獎得主－John McCarthy，在他所發表的人工智慧語言－LISP 中，

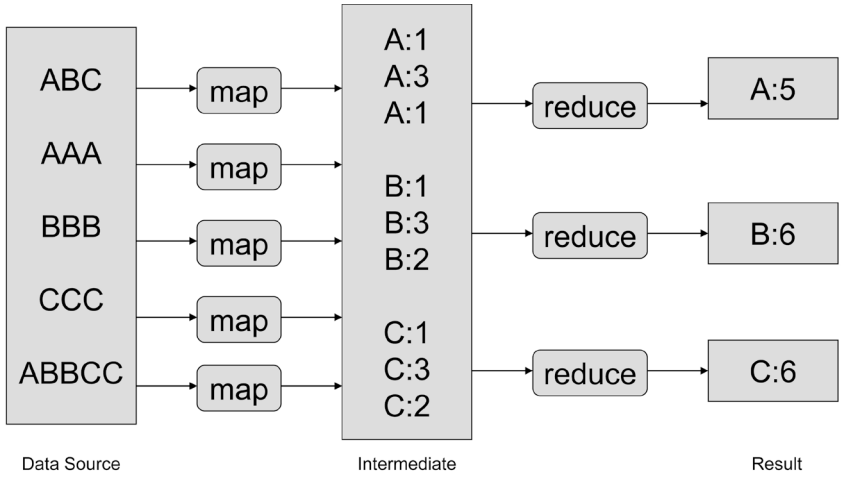
就已實作 Map/Reduce 功能。雖然，Map/Reduce 並不是 Google 的創舉，但 Google 的工程師卻很巧妙地將之應用在平行運算中，也算是將之發揚光大的功臣。

簡單地說，Hadoop 的 MapReduce 是一個分散式的軟體框架，它可以讓程式設計師透過簡單的方式，在一個巨大的伺服器叢集之中，平行地處理大量的資料。

Hadoop 的 MapReduce 包含兩個運算，即 map 與 reduce。一開始的時候，軟體框架會將輸入的資料分割成彼此之間不相關的資料片段（splits），並採用“主鍵-資料值（key-value）”的格式，分別傳給不同的 map 平行進行處理。

而在所有的 map 皆完成工作之後，軟體框架會再擔負起搜集所有 map 的輸出，以及將執行結果排序的工作，用以做為 reduce 的輸入，此時的資料結構也是採用“主鍵-資料值”的格式。最後，reduce 整合所有的結果，進而得到最終的輸出。

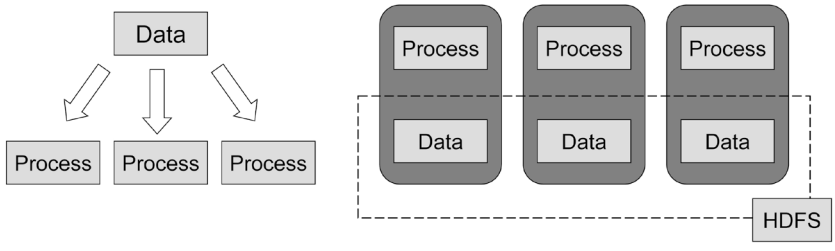
下圖為 Hadoop MapReduce 運作示意圖，其中，字母 A、B 與 C 為運算結果的主鍵，在其後的數字為運算結果的資料值。map 負責計算每一個字母在一筆資料片段中出現的次數，而 reduce 則會將主鍵值（字母）相同的資料，進行加總的工作。



在 Hadoop 的 MapReduce 軟體框架中，實作 map 與 reduce 的物件分別是 Mapper 與 Reducer，在稍後實作範例程式時，對此將有更詳細的說明。此外，在一般情況下，Hadoop 的 MapReduce 的輸出入資料，是以檔案的型式存放，但隨著 HBase 的問世（Hbase 是 Hadoop 上的一種資料庫，類似 Google 的 Bigtable），資料來源與輸出結果也可以存放在分散式資料庫之中。

MapReduce 軟體框架除了進行資料分割、傳遞與排序之外，同時，還負責排程監控，以及錯誤重新執行的工作。因此，程式設計師不再需要考慮底層的運作細節，例如資料應如何放置、如何進行切割…等，僅需要專注在處理邏輯的設計即可；如此一來，便可大幅加速系統開發的時程。

Hadoop 在進行 Map/Reduce 運作時，通常需要分散式檔案系統，也就是 HDFS（Hadoop Distributed File System）的協同運作〔參見下圖〕。圖左為傳統的平行運算架構，計算所需的資料會獨立存放在單一伺服器節點上，負責執行計算的每一個節點，皆會到這個集中的資料節點，取得所需的資料。因此，所有的負載將落在資料節點上，同時網路與 I/O 動作也較為頻繁。




圖右之 Hadoop 的 MapReduce 模式則配合 HDFS 的使用，先將資料分割成小塊，並分散儲存在每一個節點上。計算節點往往僅需負責處理該節點所儲存的部份資料即可，如此一來，便可以大幅減少網路以及 I/O 存取的動作，這也就是雲端運算為何可以提升計算速度的主要原因之一。而這種運作方式，一般稱之為“計算往儲存移動”的運作架構。

以上圖來比較，可以很明顯地區分，傳統分散式架構屬於計算密集型的應用，而藉由分散式檔案系統的配合，MapReduce 同時在資料密集型的應用上，也會有絕佳的表現。