# 第四章 JMF

在這一節裡我們將會對 Java Media Framework (JMF)做詳細的介紹，包括什麼是 JMF，JMF 的架構以及 JMF 物件和物件之間的關係等，最後提出一個簡單的範例，說明如何使用 JMF 所提供的 API 幫助我們建立可即時傳送語音資料的傳送端和接收端程式。

## 4.1 JMF 概述

Java Media Framework (JMF)是一套大型且多功能的 Java API，可以用來處理 time-based 的媒體，使得我們可以在 Java Applet 或 Application 程式裡加入語音、影像或其他 time-based 媒體資料，任何會隨著時間作有意義變化的資料，我們都可以稱他為 time-based 媒體，而 JMF 提供了各式各樣的功能以幫助我們處理媒體資料，其中包括有：

1. 在 Java 的 applet 或應用程式裡播放各式各樣的媒體檔案，其中支援的格式包括有 AU、AVI、MIDI、MPEG、QuickTime 和 WAV 等。
2. 播放網路上的媒體位元流
3. 使用麥克風和攝影機擷取語音和影像，並且用可支援的格式儲存
4. 處理 time-based 媒體以及改變媒體格式
5. 即時傳送語音和影像資料到網路上

6. 播放即時的廣播或電視節目

## 4.2 JMF 高階架構

為了讓我們更容易了解 JMF 架構，我們可以用日常生活的影音播放儲存系統做例子，如圖 4.1 所示，一般而言，我們會使用 video camera 擷取我們所需要的影像和語音資料，並將擷取到的資料儲存在 CD-ROM/DVD-ROM 或磁碟機裡，等到我們需要對媒體資料作存取或處理的動作時，我們便需要 player，在這裡 player 可能是位於個人電腦的軟體播放裝置，或是硬體放影機，經過 player 處理過後的資料便可藉由螢幕和喇叭呈現。

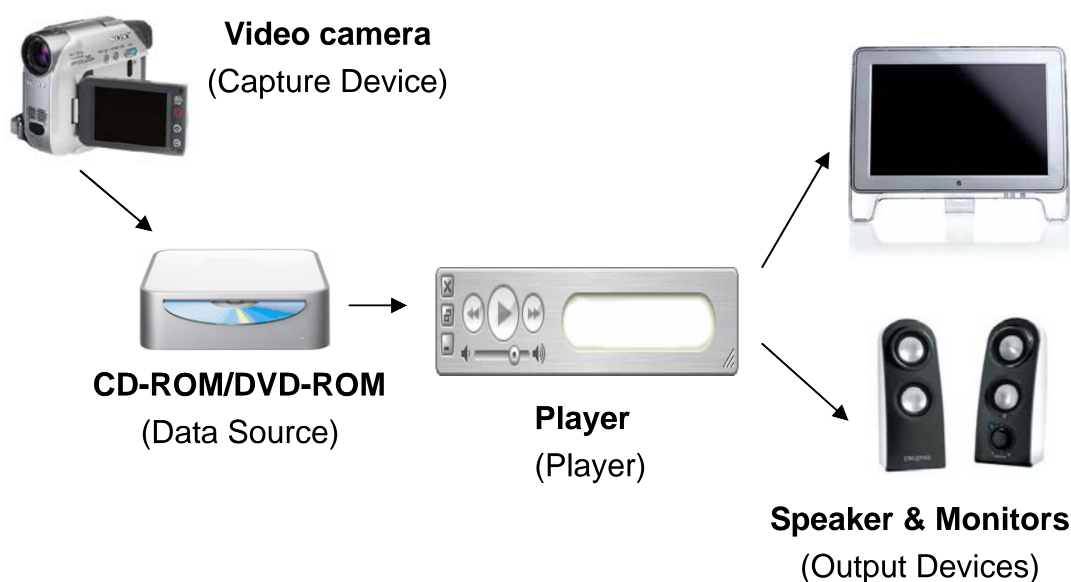


圖 4.1 JMF 的高階架構

JMF 的架構便依循了這樣的模式，JMF 定義 Capture Device 物件擷取影像和語音資料，並將媒體資料儲存在 Data Source 物件裡，當我們需要對 Data Source

做進一步的處理或存取時，便可透過 Player 物件將媒體資料藉由 Output Device 呈現，我們將在下一節裡詳細介紹這些物件在 JMF 的定義和功能。

## 4.3 JMF 主要物件

上一節我們有稍微提到幾個 JMF 的重要物件，在這一節裡我們將詳細介紹這些物件的定義以及他們所扮演的角色，其中包括有：

1. DataSource

在 JMF 裡我們使用 DataSource 物件封裝媒體位元流，如同我們常使用的 CD 片，DataSource 物件可用來代表語音、影像或者是這兩種資料的結合。此物件可以是已存在的檔案，或是位於網路上的媒體位元流。

媒體資料可以從各種不同的來源獲得，獲得的方式可能是本地的檔案資料或是位於網路上的檔案，依據獲得方式的不同，JMF 將 DataSource 分成下列兩種類型：

a. Pull Data Source：指 Client 端從網路或本地檔案獲得的資料稱之

b. Push Data Source：指 Server 端儲存在檔案或傳送到網路上的資料稱之

2. CaptureDevice

Capture Device 代表我們用來擷取資料的硬體裝置，例如麥克風、照相機或攝

影機等，擷取的資料可以提供給 Player 物件播放、轉變為其他格式或是儲存成檔案以供日後使用。

3. Player

Player 物件可用來處理多媒體資料，以供螢幕或喇叭呈現，此物件根據 DataSource 物件產生的，產生後只能處理自己的 DataSource 而無法處理其他 DataSource，圖 4.2 是一個簡單的 Player 物件模型，根據 DataSource 所產生的 Player 將處理過的媒體資料藉由螢幕或喇叭呈現。

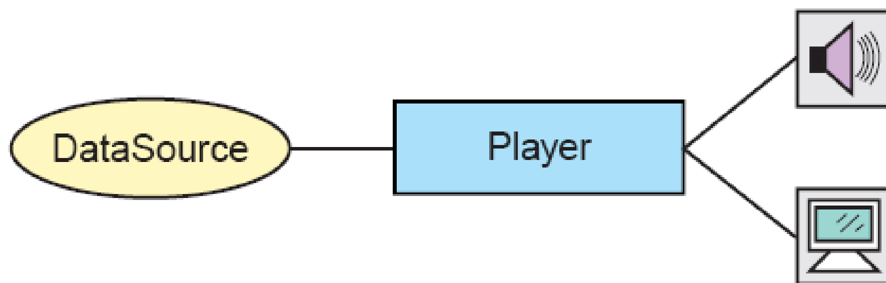


圖 4.2 Player Model

在 JMF 定義的 Player 物件擁有六個狀態，如圖 4.3 所示，以下介紹這六種狀態分別代表的涵義：

a. Unrealized：Player 物件被產生，但尚未知道媒體內容稱之
b. Realizing：當我們呼叫 Player 物件的 realize() method 時，Player 物件將從 Unrealized 狀態轉變成 Realizing 狀態，這時候 Player 開始判斷播放此媒體資料需要何種資源。

c. Realized：在此階段 Player 已經知道播放此媒體資料需要何種資源，並且也知道媒體資料的格式內容。

d. Prefetching：當我們呼叫 Player 物件的 prefetch() method 時，Player 的狀態便從 Realized 轉變成 Prefetching，在這個階段 Player 載入要播放的媒體，以及播放所需的資源，資源的類型除了一般的軟體資源外，也包含了螢幕和喇叭等獨占性硬體資源。

e. Prefetched：在這個階段 Player 已經完成擷取媒體資料以及載入資源的動作，隨時可以開始播放。

f. Started：Player 正在播放媒體資料。

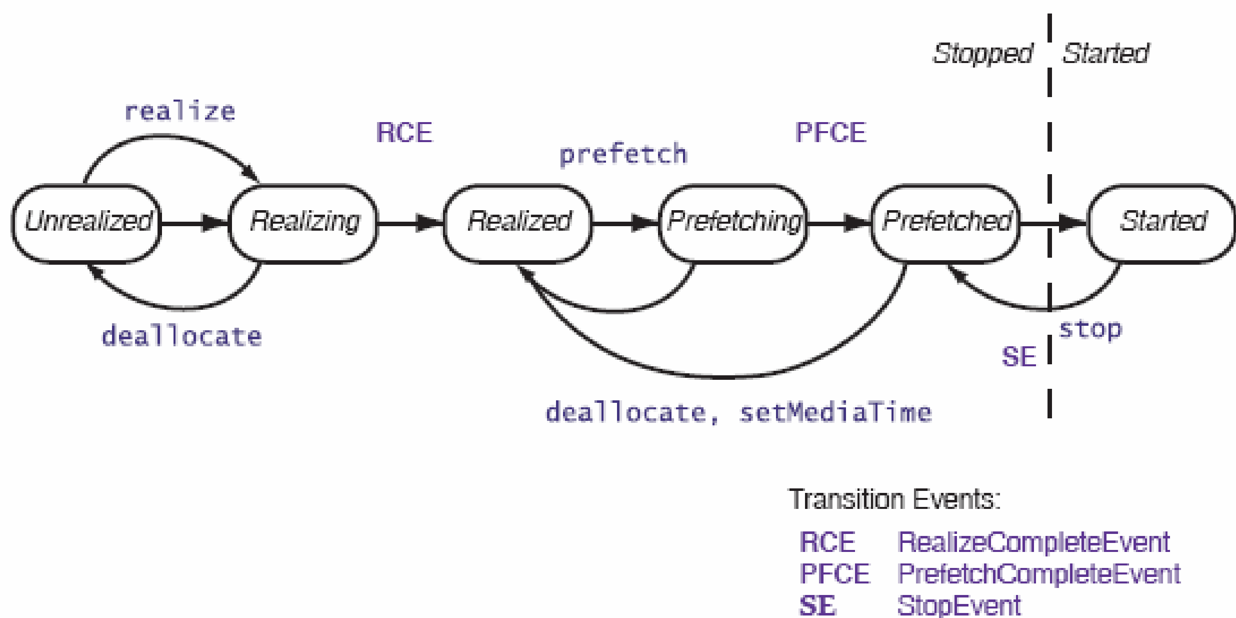


圖 4.3 Player States

4. Processor

Processor 是 Player 的一種，在 JMF API 裡，Processor 介面繼承 Player 而來，

所以 Processor 擁有和 Player 相同的功能，不同的是，除了可以將媒體資料加以處理播放外，Processor 還可以將處理過後的 DataSource 儲存成另一種格式，以供其他的 Processor 或 Player 做進一步處理。

Processor 除了擁有和 Player 相同的六種狀態外，還另外在 Unrealized 和 Realizing 之間增加了 Configuring 和 Configured 兩種狀態，如圖 4.4 所示，主要目的是更進一步的調整運算設定，以確保和 DataSource 之間的關係以及資料的格式，以下介紹這兩種新增的狀態。

a. Configuring：當 Processor 物件呼叫 configure() method 時，則會從 Unrealized 狀態進入 Configuring 狀態，在這個狀態裡 Processor 試圖與 DataSource 取得聯繫，並分離資料的語音和影像部份，了解媒體資料的格式。

b. Configured：當 Processor 進入此狀態代表已得知 DataSource 的格式，且已將資料的語音和影像部分分離。

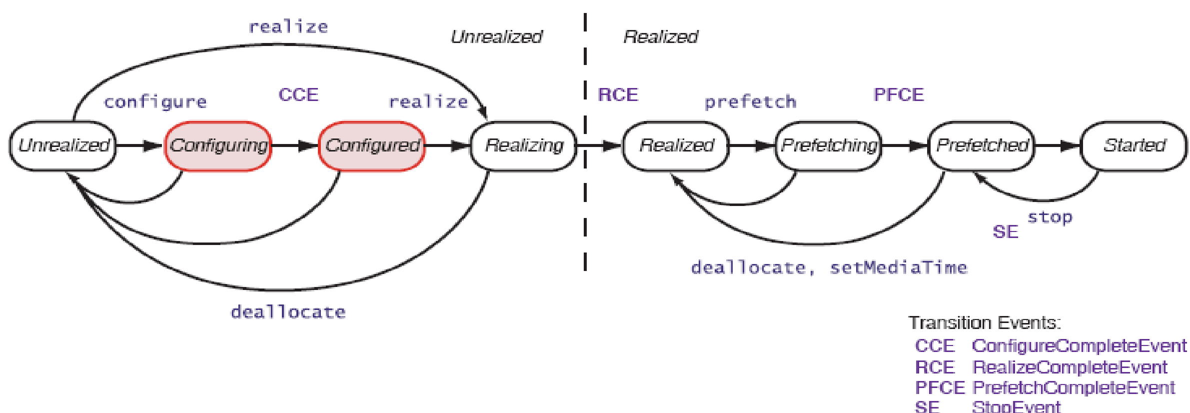


圖 4.4 Processor States

5. DataSink

類似 File-Writer，用來讀取 DataSource 的內容，並將它存放在某個位置

6. Format

Format 物件用來代表媒體資料的確切格式，描述的內容包括編碼的格式名稱，以及格式所需要的資料型態，在 Format 之下包括了 AudioFormat 和 VideoFormat 兩種子類別，而 VideoFormat 又根據編碼格式的不同分成六大類，其中包括有：H261Format、H263Format、IndexedColorFormat、JPEGFormat、RGBFormat、YUVFormat 六個子類別。

7. Manager

Manager 物件扮演中介者的角色，藉由他我們可以整合實作上述的重要 interface，例如利用已存在的 DataSource 物件建立相對應的 Player 物件。

## 4.4 JMF 主要物件間關係

在這一節裡藉由介紹物件之間的關係，幫助更詳細的了解每一個物件所代表的意義，以及物件之間的關連性。

1. JMF Manager 物件與其他物件間的關係

在 JMF API 裡使用 Manager 物件作為中介者的角色，如圖 4.5 所示，藉由他幫助我們建立 DataSource、Player、Processor、DataSink 等物件，並建立起物件

和物件之間的關連性，例如我們可以藉由 DataSource 建立相對應的 Player 物件。

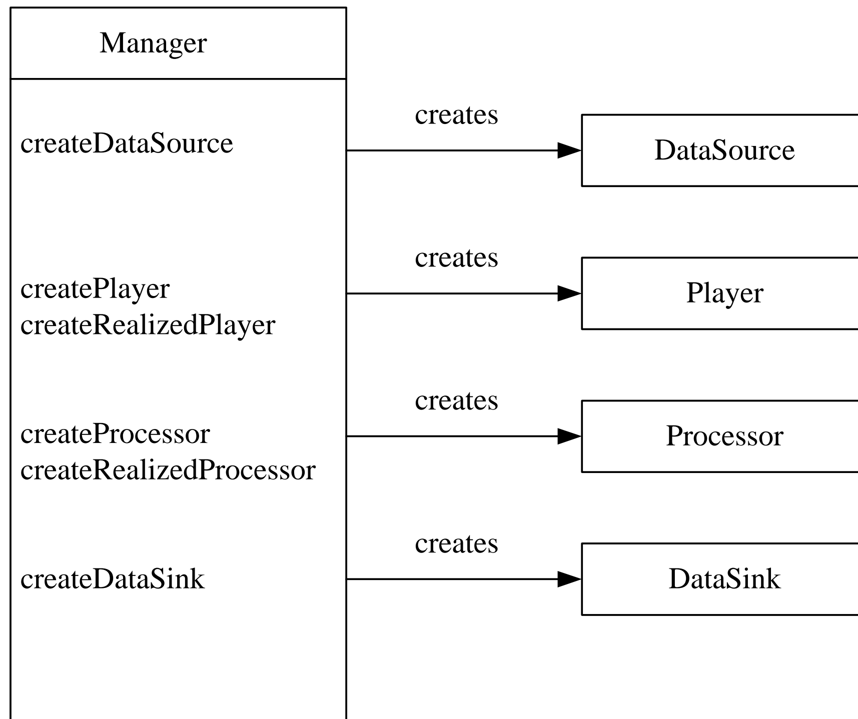


圖 4.5 JMF Manager 物件與其他物件間的關係

2. JMF Players

a. TimeBase

可用來代表現在時間，TimeBase 物件的預設值通常為系統時間

b. Clock

用來提供基礎時間來源，以及確保媒體資料同步時所需的資訊，每一個 Clock 物件產生時都需要一個 TimeBase 物件，當作時間的基準點，Clock 物件擁有

Started 和 Stopped 兩種狀態。

c. Controller

Controller 繼承了 Clock 物件的 Started 和 Stopped 兩種狀態，並將 Stopped 狀態分割為五種狀態，以用來細部處理複雜的媒體資料，在 Controller 物件裡的每一個特定的狀態下只能呼叫特定的 method。

d. MediaHandler

MediaHandler 為讀取和管理 DataSource 物件的基本介面。

e. Player

Player 繼承 Controller 和 MediaHandler 兩個介面而來，所以 Player 擁有和 MediaHandler 一樣讀取管理 DataSource 的功能，並且包含了 Controller 物件的五種狀態，和 Controller 不同的地方是，Player 在每一種狀態下可以呼叫的 method 較多，限制也較寬鬆。

f. Processor

Processor 繼承 Player 物件而來，和 Player 物件一樣可以對 DataSource 物件作適當的處理，不同的地方則是多了兩個狀態，在這兩種狀態之下，Processor 可以針對 DataSource 作細部的處理，並可將處理完的媒體資料儲存成另一種格式的 DataSource 物件。

g. DataSource

用來儲存媒體資料，可視為一般的媒體檔案，在下一節裡將會做詳細的介紹。

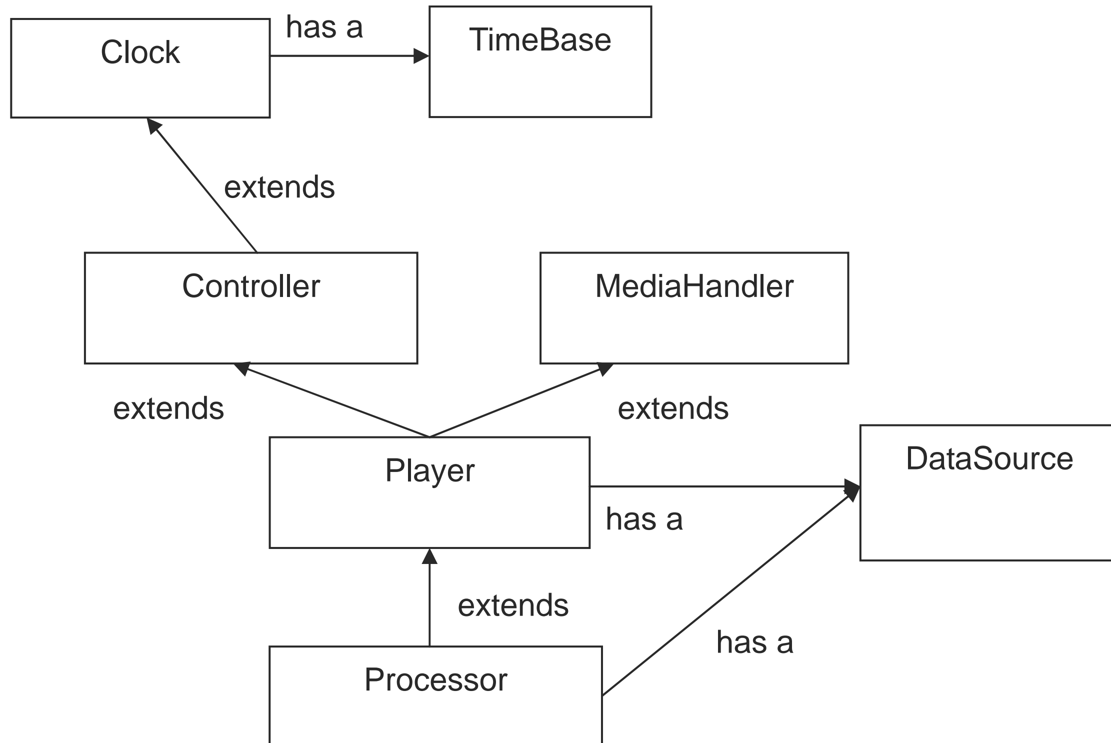


圖 4.6 JMF Players

3. Data Model

a. Control

JMF 的 Control 介面提供了設定和查詢物件屬性的機制，它通常也會根據不同的物件提供相關的使用者介面，以方便使用。

b. Duration

Durations 物件提供影片從開始到結束的時間，而每一個媒體物件都必須繼承此介面。

c. DataSource

DataSource 介面繼承 Control 和 Duration 兩個介面而來，所以我們可以對

DataSource 進行設定和存取的動作，也可以知道此物件的播放時間長度，DataSource 包含了媒體的儲存位置、使用的協定以及該使用何種軟體播放等資訊，當 DataSource 被建立後，則無法重複使用傳送其他的媒體資料。

根據傳送的方式不同 DataSource 又可分成 PullDataSource 和 PushDataSource 兩種，另外以 Buffer 為傳送單位的則稱為 PullBufferDataSource 和 PushBufferDataSource。

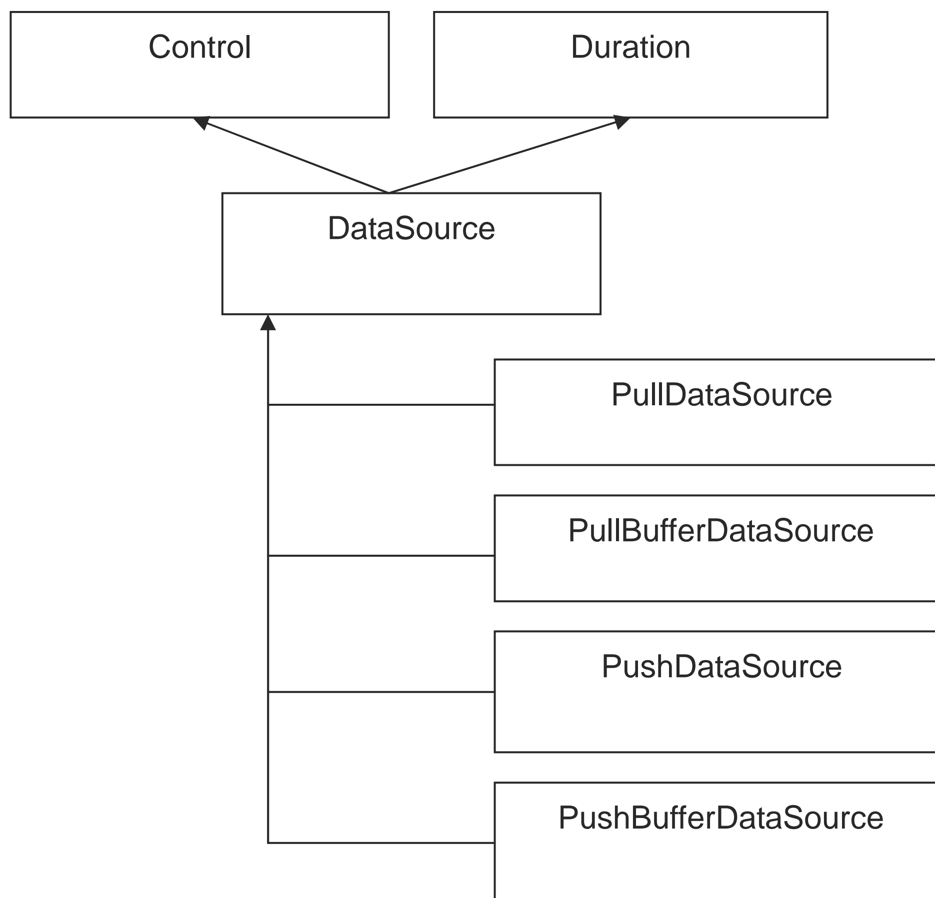


圖 4.7 以 DataSource 物件為中心和其他物件關係

## 4.5 JMF RTP API

我們實作的範例程式是採用 RTP 協定傳送，所以在我們正式進入實際討論範例程式之前，先簡單介紹在 JMF 裡所提供的 RTP API，這些 API 包含在 javax.media.rtp、javax.media.rtp.event 以及 javax.media.rtp.rtcp 封包裡，而我們可以使用這些 API 播放從網路上接收到的 RTP 媒體位元流、將它們儲存在檔案裡或者同時實作以上這兩項功能，如圖 4.8 所示，我們可以使用 DataSink 物件將媒體位元流儲存成檔案，或者藉由 Player 物件將處理過後的媒體位元流經由播放器呈現，甚至我們也可以使用 Processor 物件將媒體位元流儲存成另一種格式的 DataSource 物件，以供其他 Player 或 DataSink 使用。

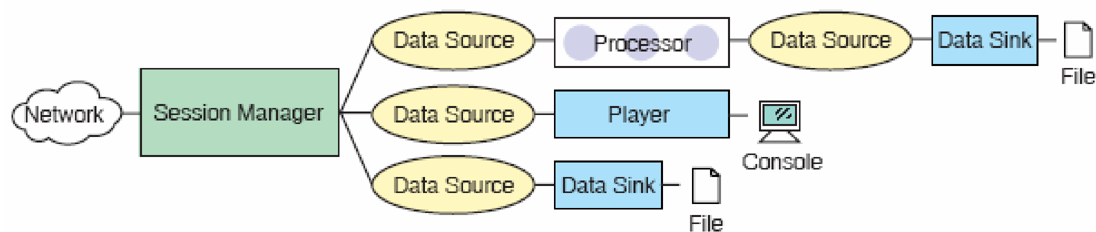


圖 4.8 RTP Reception

相反的，我們也可以使用 RTP API 將擷取到或儲存的媒體位元流使用 RTP 協定傳輸到網路上，如圖 4.9 所示，我們將內部的檔案或從擷取裝置獲得的資料存放在 DataSource 物件裡，經由 Processor 處理過後，儲存成另一種可供 RTP 傳送的媒體格式，經由 Session Manager 傳送到網路上，Session Manager 可幫助我們開啟、管理、關閉 RTP 會議，以下介紹在 RTP API 裡的重要介面。

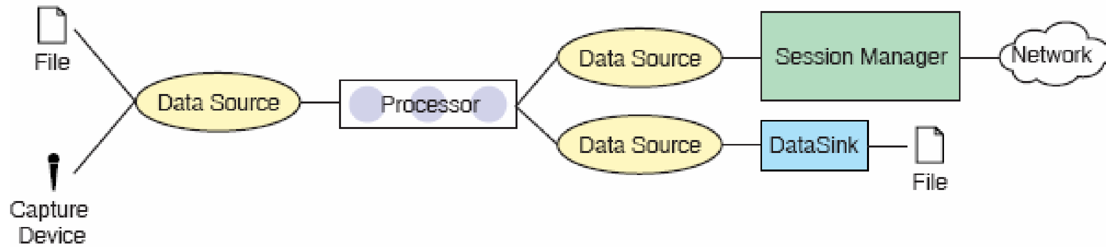


圖 4.9 RTP Transmission

1. RTP Manager

在 JMF 裡將 Session Manager 定義為 RTPManager 介面，RTPManager 可用來開啟、管理、結束 RTP 會議，並且持續紀錄在會議裡的參與者，藉由 RTP Manager 我們可以將媒體位元流採用 RTP 協定傳送到網路上，或是播放、儲存網路上的媒體位元流，以下介紹 RTP Manager 管理的兩大物件：Participant 和 Stream。

a. Participant

Participant 代表在 RTP 會議裡傳送和接收資料的參與者，參與者可能擁有一個或以上的媒體位元流 (Stream)我們可以藉由他得知參與此媒體會議的使用者名稱為何。

b. RTPStream

RTPStream 代表位於 RTP 會議裡的媒體位元流，並且由 RTPManager 負責管理，JMF 將 RTPStream 分為 SendStream 和 ReceiveStream 兩種：

ReceiveStream：代表從遠端參與者 (participant)傳送過來的媒體位元流

SendStream：代表從 Processor 或 DataSource 物件傳出的媒體位元流資料，

當我們需要建立一個新的 SendStream 可藉由呼叫 RTPManager 的 createSendStream() method。

2. Event

在 javax.media.rtp.event.這個封包裡包含了許多特別為 RTP 協定定義的 Event，我們使用這些事件得知 RTP 會議和位元流的狀態，以下是在我們範例程式裡所有會使用到的 Event，以及各個事件之間的關係

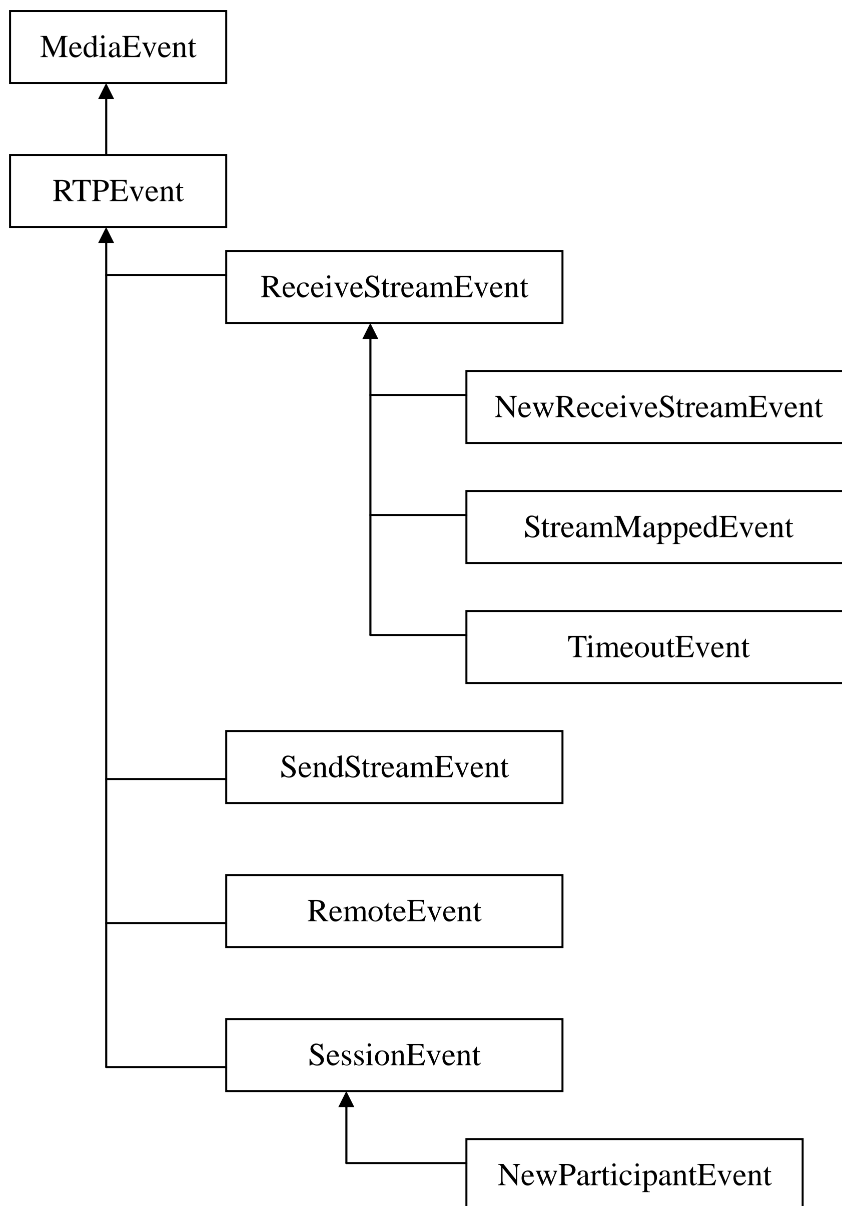


圖 4.10 RTP Events

由圖 4.10 可知，RTPEvent 繼承 MediaEvent 而來，而 RTPEvent 之下又分成四大類，其中包括有 ReceiveStreamEvent、SendStreamEvent、RemoteEvent 和 SessionEvent 這四種，針對每一種不同類型的 Event，JMF 都定義了不同的 Listener Interface 用來接收處理這些 Event，以下根據這些不同的 Listener，分別介紹位於這四大類 Event 裡的子事件：

a. ReceiveStreamListener: 藉由實作此介面接收任何改變狀態的 StreamEvent，其中這些狀態的改變可能包含新的媒體位元流已建立，或是媒體位元流已結束等，其中在我們的範例程式使用到 RTPEvent 有：

- NewReceiveStreamEvent：代表 RTP Manager 收到從新的 DataSource 傳送過來的媒體位元流
- StreamMappedEvent：當我們收到此事件時代表之前曾經收到過尚未辨別 Participant 的媒體位元流，而現在此媒體位元流已確認是由某 Participant 傳送過來
- TimeoutEvent：代表結束傳送媒體位元流

b. SendStreamListener: 藉由實作此介面接收任何從 RTP Manager 傳送出去的新事件，這些事件可能包含新 stream 的建立、開始或結束傳送媒體位元流以及媒體位元流的格式改變等事件，在本範例程式並沒有特別處理 SendStreamEvent。

c. RemoteListener: 藉由實作此 Listener 接收從遠端 Participant 所傳送過來的事件，本範例程式無實作此 Listener。

d. SessionListener: 藉由實作此程式接收任何和會議有關的事件，事件的內容通常為新 Participant 的加入，在此範例程式裡使用到的 Event 為 NewParticipantEvent，當接收到此事件時，代表新的 Participant 已加入此會議。

3. RTP Controls

在 JMF 的 RTP API 裡定義了 RTPControl 物件用來控制存取 DataSource 的資訊,所以每一個 DataSource 物件都會有一個 RTPControl 介面,以獲取 DataSource 資料格式等資訊。

4. Session Address

JMF 的 RTP API 將網路位址物件(InetAddress)和 Port Number 封裝成 SessionAddress 物件，InetAddress 物件定義在 java.net.InetAddress 封包裡，主要的目的是用來封裝 IP 位址,而 SessionAddress 則包含了 InetAddress 和 Port Number 兩個部份，可用來定義會議位址。

## 4.6 實作範例程式介紹

從以上的說明我們可以大略了解 JMF 的架構以及各物件和物件之間的關係,並包含了 JMF 所定義的 RTP API,在這一節裡我們將會介紹如何使用 JMF API

分別實作可傳送和接收語音的範例程式，以下先介紹傳送端範例程式。

### 4.6.1 實作傳送端範例程式

在我們的傳送端範例程式裡將會分成四個主要部份：

1. Import Package
2. 建立用來處理 DataSource 物件的 Processor
3. 將 Processor 處理過後的 DataSource 使用 RTP 協定傳送到網路上
4. 開始傳送媒體位元流

首先介紹第一個部份：

1. Import Package

本範例程式裡所用到的 Package 包括有以下三種：

a. java.io.* package：包含了所有系統用來處理 IO 的 API
b. java.net.InetAddress package：將網路的 IP Address 封裝成 InetAddress 物件，以供我們對網路位址存取使用
c. javax.media.* package：包含所有 JMF 提供的 API

在我們實作的範例程式裡必須加入這些 package，以下原始程式碼為加入的方法：

---------------------------------------------------------------------------------------------------

```
import java.io.*;
```

```
import java.net.InetAddress;
import javax.media.*;
```

----------------------------------------------------------------------------------------------------

有了以上這些 package，我們便可實作傳送和接收語音的範例程式。

2. 建立用來處理 DataSource 物件的 Processor

在這個範例程式裡 DataSource 物件是由外部的擷取裝置，也就是透過麥克風獲得的，其主要物件間的關係流程如圖 4.11 所示，首先我們必須先獲得 MediaLocator 物件，MediaLocator 建立的方式為 new MediaLocator("javasound:/8000")，其中 ”javasound:/8000”參數代表我們的麥克風裝置，有了 MediaLocator 物件，我們便可以使用他建立 DataSource 物件，如此一來，DataSource 便封裝了麥克風位址，並將得到的語音資料儲存起來，建立 DataSource 的方法為呼叫 Manager 的 createDataSource() method，最後有了 DataSource 物件，我們便可以產生 Processor 物件處理 DataSource，產生 Processor 的方式則為呼叫 Manager 的 createProcessor() method，藉由 Processor 物件，我們可以針對 DataSource 進行改變格式等的處理，而存在於 Processor 裡的 TrackControl 物件便可幫助我們完成格式變更的工作，對 DataSource 物件的格式進行處理。

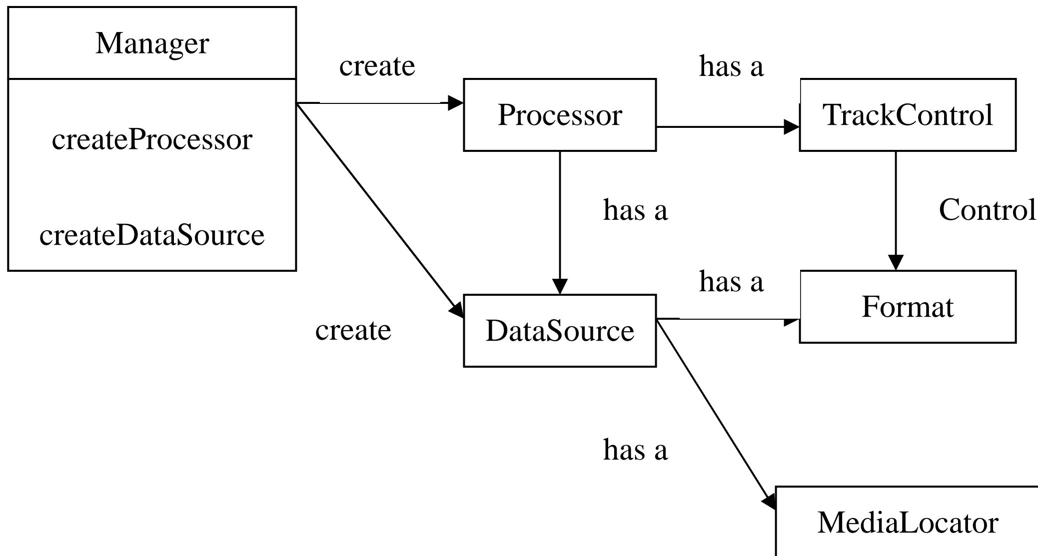


圖 4.11 建立 Processor

------------------------------------------------------------------------------------------------

**private String createProcessor() {**

**try {**

//在這裡我們使用了 MediaLocator 物件 (mediaLocator)包裝了 DataSource 的位

//置，MediaLocator 有可能是位於網路上的位址，或是個人電腦裡的檔案，也可

//能是擷取裝置,例如麥克風的位置,當我們要指定麥克風裝置時,指定的值 //

為 ”javasound:/8000”

//有了 MediaLocator 物件,我們便可使用 Manager 的 creaeDataSource() method 建

//立新的 DataSource，建立時必須給定 DataSource 的位置，也就是我們上述介紹

//的 MediaLocator 物件

**DataSource ds = javax.media.Manager.createDataSource(new**

**MediaLocator("javasound:/8000"));**

**} catch (Exception e) {**

//如果發生錯誤則無法正確獲得 DataSource 物件

**return "Couldn't create DataSource";**

**}**

//有了 DataSource 物件之後，接下來則是產生用來處理此物件的 Processor

**try {**

//使用 Manager 的 createProcessor() method 產生新的 Processor 物件，產生時必須

//給定相對應的 DataSource 物件，也就是剛剛我們所建立的 DataSource 物件

**Processor processor = javax.media.Manager.createProcessor(ds);**

**} catch (NoProcessorException npe) {**

//若發生錯誤則回傳無法產生 Processor 訊息

**return "Couldn't create processor";**

**}**

//獲得存在於 Processor 裡的每一個 TrackControl 物件，以針對每一段音軌進行//

處理，並以此物件以陣列的方式儲存

**TrackControl[] tracks = processor.getTrackControls();**

//建立兩個 Format 物件，其中一個儲存可支援的格式 (supported)，另一個則儲存

//選擇傳送的格式 (chosen)

```
        Format supported[];
        Format chosen;
//開始針對每一段音軌進行處理
        for (int i = 0; i < tracks.length; i++) {
//列出所有此音軌的可支援編碼格式
            supported = tracks[i].getSupportedFormats();
//判斷是否有一個以上的可支援編碼格式
            if (supported.length > 0) {
//如果有的話，在這裡我們選擇第一個編碼格式，其編碼格式為 dvi/rtp，其中 index
//為 1 的編碼格式為 G.711/rtp、index 為 2 的編碼格式為 g.723/rtp、index 為 3 的//
編碼格式為 gsm/rtp，共支援四種編碼格式
                chosen = supported[0];
//設定選取的編碼格式
                tracks[i].setFormat(chosen);
        }
}
//所有的值都設定好之後，取得 Processor 的 DataSource 物件，並將它儲存在
//DataOutput 物件以供未來 RTP 傳送使用
        DataSource dataOutput = processor.getDataOutput();
```

//回傳 null 訊息代表無錯誤發生

```
        return null;
        }
```

---------------------------------------------------------------------------------------------------

3. 將 Processor 處理過後的 DataSource 使用 RTP 協定傳送到網路上

以上程式已介紹如何成功建立處理這些媒體資料的 Processor 物件，接下來的範例程式則是如何將這些媒體位元流利用 RTP 協定傳送到網路上，建立 RTP 會議最重要的物件為 RTPManager，而在建立 RTPManager 之前，我們必須知道該建立多少個 RTPManager 以供使用，RTPManager 的個數是根據位元流的數量而定的，所以在這裡我們必須先將上述 Processor 所產生的 DataSource 物件轉變為 Stream 物件，由圖 4.12 可知，我們可以先將 DataSource 物件轉變成 PushBufferDataSource 物件，接下來則呼叫 getStream() method，獲得屬於 PushBufferDataSource 的每一個 PushBufferStream，並將這些位元流儲存在陣列裡，有了這個 PushBufferStream 陣列之後，我們便可計算陣列的數量，並依照這個數量建立 RTPManager。

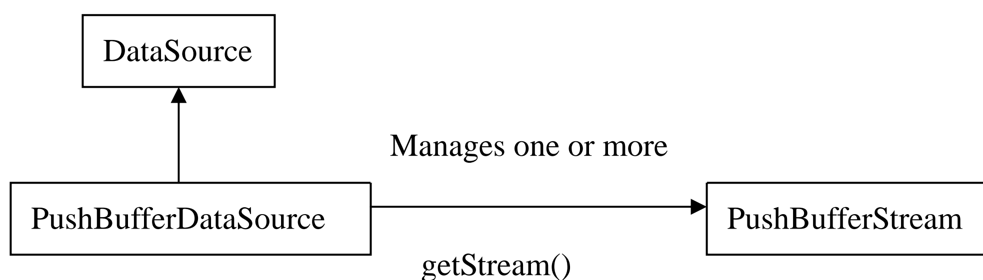

圖 4.12 獲得 PushBufferStream

當我們有了 RTPManager 物件之後，接下來則是 RTPManager 物件的相關設定，如圖 4.13 所示，首先我們必須擁有兩個 SessionAddress 物件，其中一個紀錄本地位址另一個則紀錄遠端位址，SessionAddress 物件的建立必須包含 InetAddress 物件和 Port Number，有了代表本地位址的 SessionAddress 之後，我們便可以呼叫 RTPManager 的 initialize() method，讓 RTPManager 知道會議的本地端位址，另外則呼叫 addTarger() method，並傳入代表遠端位址的 SessionAddress 物件，讓 RTPManager 知道媒體位元流的遠端傳送位址，如此一來便成功設定好 RTPManager，最後則輸出 SendStream 物件，如此便可開始傳送媒體位元流，以下為範例程式的部份原始程式碼。

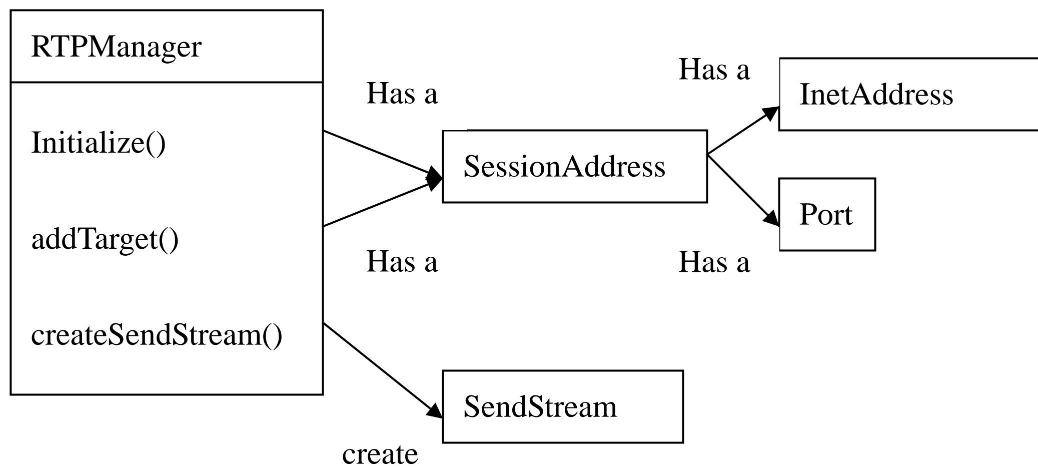


圖 4.14 建立設定 RTPManager

---------------------------------------------------------------------------------------------

/**

* 在以下的程式裡我們將會介紹如何透過 RTPManager API 針對每一個

```
    * Processor 所產生的 Stream 建立兩端點之間的 session
    */
   private String createTransmitter() {
//在這裡我們將上一段程式裡 Processor 最後產生的 DataSource (dadaOutput)物件
//改變成以 Buffer 為傳送單位的 PushBufferDataSource，以方便我們使用 RTP 協
//定傳送
        PushBufferDataSource pbds = (PushBufferDataSource)dataOutput;
//使用 PushBufferDataSource 的 getStrems() method 獲得每一段位元流，並以陣列
//方式儲存
        PushBufferStream[] pbss = pbds.getStreams();
//根據位元流的數量建立相同數目的 RTPManager，以用來處理傳送媒體位元流
        RTPManager[] rtpMgrs = new RTPManager[pbss.length];
//建立用來儲存本地 (localAddr)和遠端位址 (desAddr)的 SessionAddressj 物件，
//以供 RTPManager 使用
        SessionAddress localAddr, destAddr;
//建立包含在 SessionAddress 裡的 InetAddress 物件
        InetAddress ipAddr;
//建立包含在 SessionAddress 物件裡的 port number
        int port;
```

```
//開始針對每一個]PushBufferStream 建立設定 RTPManager
        for (int i = 0; i < pbss.length; i++) {
        try {
//設定傳送此 PushBufferStream 物件的 port number，每一次增加 2
            port = portBase + 2*i;
//根據遠端的 IP Address 建立 InetAddress 物件，以供 SessionAddress 使用
            ipAddr = InetAddress.getByName(ipAddress);
//建立本地的 SessionAddress，建立時必須給定代表本地位址的 InetAddress 物件
//以及 port number
            localAddr = new SessionAddress( InetAddress.getLocalHost(),
                port);
//建立遠端的 SessionAddress，建立時必須給定代表遠端位址的 InetAddress 物件
//以及 port number，本地和遠端的 port number 必須相同
            destAddr = new SessionAddress( ipAddr, port);
//開始建立 RTPManager 建立的方式為使用 RTPManager 物件的 newInstance()
//method，獲得一個新的 RTPManager
            rtpMgrs[i] = RTPManager.newInstance();
//使用 RTPManager 的 initialize() method 初始化 session，使用時必須給定本地的
//SessionAddress 物件(localAddr)，此 method 只能被呼叫一次
```

**rtpMgrs[i].initialize( localAddr);**

//藉由使用 RTPManager 的 addTarget() method 讓 RTPManager 獲得媒體位元流的遠端位址，並開啟此會議，此 method 必須在 initialize() method 之後呼叫

**rtpMgrs[i].addTarget( destAddr);**

//最後使用 RTPManager 的 createSendStream() method 建立 SendStream 物件

**SendStream sendStream =**

**rtpMgrs[i].createSendStream(dataOutput, i);**

//使用 SendStream 的 start() method 開始傳送媒體位元流

**sendStream.start();**

**} catch (Exception e) {**

**return e.getMessage();**

**}**

**}**

//代表程式成功執行完成無錯誤發生

**return null;**

**}**

----------------------------------------------------------------------------------------------------

4.開始傳送媒體位元流

在這個範例程式裡，我們將會呼叫上述的 createProcessor()和 createTransmitter() method，以用來開啟傳送端程式，並將 Prcocessor 的狀態設成 Start，開始傳出從麥克風擷取的媒體位元流物件，以供 RTPManager 使用，以下為開啟傳送端程式的 Start() method。

```
----------------------------------------------------------------------------------------------------
public synchronized String start() {
//呼叫 createProcessor() method，以建立 Processor 物件
    createProcessor();
//建立 RTPManager 物件，以用來傳送 Processor 物件所產生 DataSource
    createTransmitter();
//將 Processor 物件的狀態設成 Start，開始傳送媒體
    processor.start();
//若程式成功執行完成，則回傳 null 字串
    return null;
}
----------------------------------------------------------------------------------------------------
```

## 4.6.2 實作接收端範例程式

介紹完傳送端範例程式，接下來則是介紹如何實做接收端範例程式，在接收端範例程式裡，我們必須實作兩種 Listener，其中包括有：

1. ReceiveStreamListener：負責接收 ReceiveStreamEvent，並根據接收到的事件不同作相對應的處理，藉由此介面可讓 RTPManager 獲得媒體位元流的狀態，實作的方法為實作 update( ReceiveStreamEvent event) method。

2. SessionListener：負責接收 SessionEvent，並根據接收到的事件進行相對應的處理，藉由此介面可獲得有關 Session 的各種資訊，實作的方法為實作 update( SessionEvent event) method。

所以在我們的接收端範例程式裡包含四個部份：

1. import package
2. 實作 ReceiveStreamListener
3. 實作 SessionListener
4. 初始化 RTPManager

首先是第一個部份

1. import package

在此範例程式裡需要用到四種 Package，其中包括有

a. java.awt.* package

定義所有建立 java GUI 介面的 interface，藉由此封包我們可以建立一個簡單的 Player 視窗介面

b. java.net.* package

定義所有有關網路物件的 API，例如 InetAddress 等物件

c. java.util.Vector package

定義所有有關 Vector 物件的 package

d. import javax.media.*

包含了所有 JMF API 的 package

Import package 的原始程式碼：

```
----------------------------------------------------------------------------------------------------
import java.awt.*;
import java.net.*;
import java.util.Vector;
import javax.media.*;
----------------------------------------------------------------------------------------------------
```

2. 實作 ReceiveStreamListener

在我們的 ReceiveStreamListener 裡所收到的 ReceiveStreamEvent 包含四個種類，並針對每一個不同種類進行處理，如圖 4.15 所示：

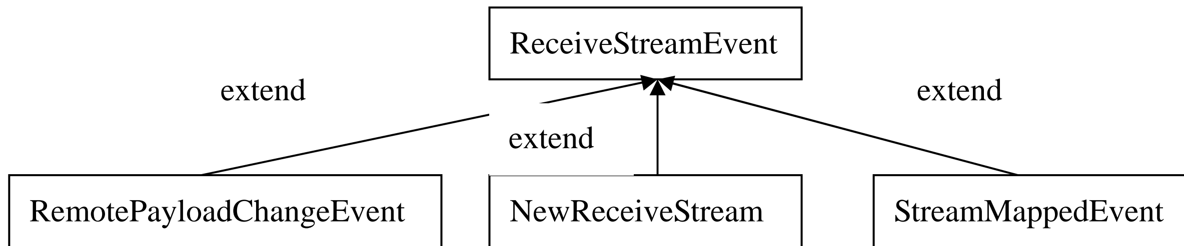


圖 4.15 ReceiveStreamEvent

a. RemotePayloadChangeEvent

代表遠端傳送過來的媒體位元流內容已改變，在我們的範例程式裡不對內容改變的媒體位元流進行處理，所以當程式收到此事件則將停止接收媒體位元流，並將程式停止，以下是處理此物件的原始程式碼：

---

```
//首先判斷接收到的 ReceiveStreamEvent 是否為 RemotePayloadChangeEvent
    if (ReceiveStreamEvent instanceof RemotePayloadChangeEvent)
//如果是的話則印出無法處理此事件的錯誤訊息
        System.err.println("    - Received an RTP PayloadChangeEvent.");
        System.err.println("Sorry, cannot handle payload change.");
//將程式正常結束
        System.exit(0);
    }
```

---

b. NewReceiveStreamEvent

代表接收到新的媒體位元流，當我們收到此事件時，程式必須獲得位於此事件裡的 DataSource 物件，並產生相對應的 Player 物件以用來處理播放此媒體位元流，處理的流程如圖 4.16 所示：

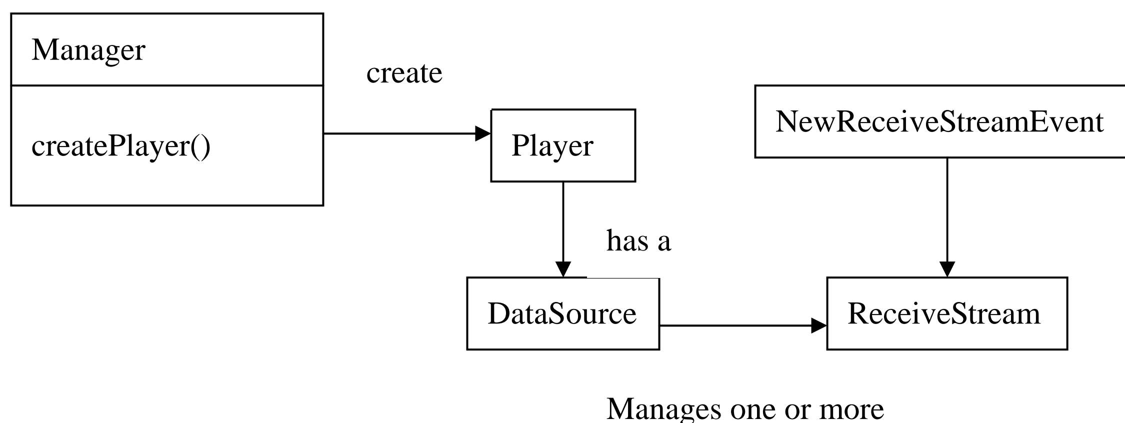


圖 4.16 NewReceiveStreamEvent

首先藉由收到的 NewReceiveStreamEvent 獲得 ReceiveStream 物件，ReceiveStream 物件包含在 DataSource 物件裡，所以我們可以獲得屬於此位元流的 DataSource，最後則是利用 Manager 的 createPlayer() method 建立相對應的 Player 物件，以下為接收處理 NewReceiveStreamEvent 的原始程式碼。

```
----------------------------------------------------------------------------------------------------
//首先判斷接收到的位元流事件是否為 NewReceiveStreamEvent
    if (ReceiveStreamEvent instanceof NewReceiveStreamEvent) {
```

```
        try {
            ReceiveStream stream =
                ((NewReceiveStreamEvent)evt).getReceiveStream();
//獲得位於此事件的媒體位元流
            DataSource ds = stream.getDataSource();
//每一段媒體位元流都包含在 DataSource 物件裡，所以在這裡我們獲得包含此位
//元流的 DataSource 物件
        }
//使用 Manager 的 createPlayer() method 建立相對應於 DataSorce (ds)物件的
//Player
        Player p = javax.media.Manager.createPlayer(ds);
//最後呼叫 Player 的 realize() method，使 Player 的狀態轉變為 realize
        p.realize();
        } catch (Exception e) {
//所發生錯誤則印出錯誤訊息
        System.err.println("NewReceiveStreamEvent exception " +
            e.getMessage());
        return;
        }
```

}

c. StreamMappedEvent

當我們收到 StreamMappedEvent 時，代表此位元流為之前已接收到的位元流，此時我們可以不做任何動作，以下為處理此事件的原始程式碼：

```
-----------------------------------------------------------------------------------------------------
//判斷接收到的位元流是 StreamMappedEvent
    if (ReceiveStreamEvent instanceof StreamMappedEvent) {
//如果是的話則印出此位元流為之前尚未確認的位元流
            System.err.println("    - The previously unidentified stream ");
    }
-----------------------------------------------------------------------------------------------------
```

3. 實作 SessionListener

SessionListener 是用來接收 SessionEvent 的，在我們的範例程式裡只處理 SessionEvent 中的 NewParticipantEvent，其中處理的方式如圖 4.17 所示，在每一個 NewParticipantEvent 中都有 Participant 物件，而我們只須獲得此物件並印出 Participant 的名稱即可，實作 SessionListener 時，必須實作 SessionListener 的 Updata(SessionEvent) method，以下為實作的範例程式。

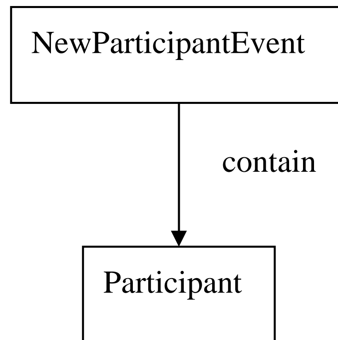


圖 4.17 NewParticipantEvent

---

```
//實作 SessionListner 的 update(SessionEvent) method
public synchronized void update(SessionEvent evt) {
//判斷事件類型是否為 NewParticipantEvent
    if (SessionEvent instanceof NewParticipantEvent) {
//獲得 NewPartcipantEvent 裡的 Participant 物件
        Participant p = ((NewParticipantEvent)evt).getParticipant();
//印出 Participant 名稱
        System.err.println("    - A new participant had just joined: " +
p.getCNAME());
    }
}
```

---

4. 初始化 RTPManager

在傳送端的範例程式裡我們曾經談到如何利用 RTPManager 傳送 RTP 媒體位元流，在接收端的程式裡，我們一樣需要 RTPManager 幫助我們處理接收到的媒體位元流，如圖 4.8 所示，經由 RTPManager 收到的媒體位元流則交給我們之前實作的 Listener 做處理，在這裡我們建立一個 initialize() method 實作所有建立 RTPManager 的流程，實作的流程如圖 4.18 所示，首先我們必須擁有兩個 SessionAddress 物件，其中一個紀錄本地位址另一個則紀錄遠端位址，SessionAddress 物件的建立必須包含 InetAddress 物件和 Port Number，有了代表本地位址的 SessionAddress 之後，我們便可以呼叫 RTPManager 的 initialize() method，讓 RTPManager 知道會議的本地端位址，另外則呼叫 addTarger() method，並傳入代表遠端位址的 SessionAddress 物件，讓 RTPManager 知道媒體位元流的遠端傳送位址，如此一來便成功設定好 RTPManager，接下來利用 RTPManager 的 addSessionListener()和 addReceiveStreamListener()，加入接收端程式，如此一來，RTPManager 便可將從網路接收到的媒體位元流交給 Listener 做進一步處理，以下是我們建立 RTPManager 的原始範例程式。

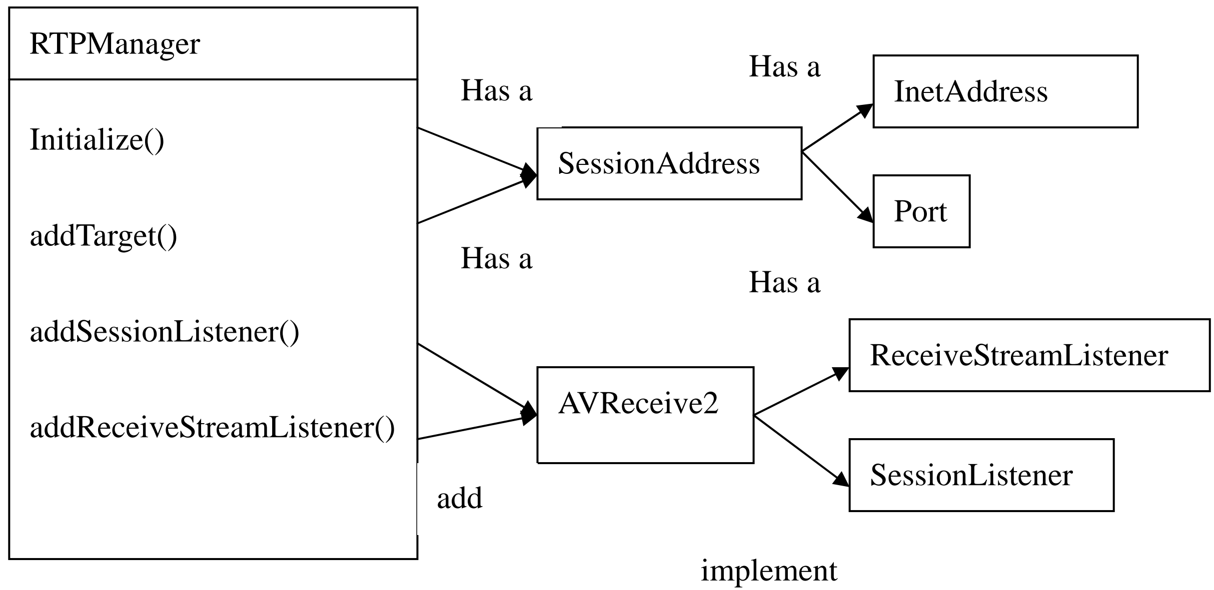


圖 4.18 初始化 RTPManager

---

//初始建立 RTPmanager 物件

**protected boolean initialize()**

**{**

**try {**

//建立記錄遠端位址的 InetAddress 物件

**InetAddress ipAddr;**

//建立新的本地和遠端 SessionAddress 物件以供 RTPManager 使用

**SessionAddress localAddr, destAddr**

//建立新的 RTPManager 物件

**RTPManager mgrs = (RTPManager) RTPManager.newInstance();**

//加入我們在範例程式裡所實作 SessionListener 物件，因為此 listener 實作在我

//們的接收端程式裡，所以傳入的參數為程式名稱(AVReceiver2)

**mgrs.addSessionListener(AVReceive2);**

//加入我們在範例程式裡所實作 ReceiveStreamListenerListener 物件，因為此

//listener 實作在我們的接收端程式裡，所以傳入的參數為程式名稱(AVReceiver2)

**mgrs.addReceiveStreamListener(AVReceive2);**

//建立記錄遠端位址的 InetAddress 物件

**ipAddr = InetAddress. getByName(ipAddress);**

//建立記錄本地位址的 SessionAddress 物件以供 RTPmanager 使用

**localAddr = new SessionAddress(InetAddress.getLocalHost(),**

**session.port);**

//建立記錄遠端位址的 SessionAddress 物件以供 RTPmanager 使用

**destAddr = new SessionAddress(ipAddr, session.port);**

//使用 RTPManager 的 initialize() method 初始化 session，使用時必須給定本地的

//SessionAddress 物件(localAddr)，此 method 只能被呼叫一次

**mgrs.initialize(localAddr);**

//藉由使用 RTPManager 的 addTarget() method 讓 RTPManager 獲得媒體位元流的

//遠端位址，並開啟此會議，此 method 必須在 initialize() method 之後呼叫

```
                mgrs.addTarget(destAddr);
                    }catch (Exception e) {
//若發生錯誤則印出錯誤訊息，並回傳 false
                        System.out.println("Cannot create the RTP Session: "+
                            e);
                        return false;
                    }
//回傳 true，代表已成功建立設定 RTPManager
            return true;
        }
```

------------------------------------------------------------------------------------------------

## 4.6.3 整合 JAIN SIP 和 JMF 實作網路電話

在第三章裡我們介紹了如何利用 JAIN SIP 所提供的 API 實作 SIP 會議，而這一章裡則是利用 JMF 所提供的 API 實作了兩支程式，其中一個為接收端，另一個則為傳送端程式，在圖 2.1 的 SIP 基本呼叫建立和結束流程裡，我們可以很清楚的看到，當 UAC 收到 OK 訊息，且 UAS 收到 ACK 訊息之後，接下來便開始傳送媒體位元流，進行媒體會議，所以當成功建立 SIP 會議之後，接下來我們便可以使用上述的傳送端和接收端程式，進行傳送接收媒體位元流，我們的傳送端程式名稱為 AVTransmit2，接收端名稱則是 AVReceive2，建立 SIP 會議的程式

名稱則是 JainSipPhone，在 JainSipPhone 程式裡開啟媒體位元流的方法為分別建立 startTransmit() 和 startReceiver()兩個 method，以下為建立這兩種 method 的方法。

-----------------------------------------------------------------------------------------------------

```
//開始傳送媒體位元流
public void startTransmit(AVTransmit2 av)
{
//開啟傳送端程式的方法為呼叫我們在傳送端程式所實作的 Start() method，以用
//來建立 Processor 和 RTPmanager 物件，並傳送位元流到網路上
    av.start();
}
//開始接收媒體位元流
protected void startReceiver(AVReceive2 av )
{
//開啟接收端程式的方法為呼叫我們在接收端所實作的 initialize() method，以用
//來建立 RTPmanager，呼叫時必須傳入在 JainSipPhone 程式裡所建立的接收端程
式物//件(AVReceive2)
        av.initialize();
}
```

建立好 startTransmit()和 startReceiver() method 之後，我們便可在 JainSipPhone 程式裡呼叫這兩個 method，首先在第三章的 3.4.5 節 UAC Receive Response and Send ACK 裡，當程式處理完 OK 訊息，並回傳 ACK 訊息之後，便可呼叫 startTransmit()和 startReceiver()，呼叫的方式如下。

---

```
public void JainSipPhone::
    processResponse(ResponseEventresponseReceivedEvent) {
//一開始利用 ResponseEvent 的 getResponse() method 獲得 response 訊息物件
   Response response = (Response) responseReceivedEvent.getResponse();
//使用 ResponseEvent 的 getClientTransaction() method 獲得屬於此 Response 物件 //
的 ClientTransaction，此 ClientTransaction 可幫助傳送回傳的 Request 物件
   Transaction tid = responseReceivedEvent.getClientTransaction();
//判斷此 Response 是否為 200/OK，並且是針對 INVITE 訊息所產生的
   boolean flag = response.getStatusCode() == 200 && ((CSeqHeader)
      response.getHeader(CSeqHeader.NAME)).getMethod().
      equals(Request.INVITE);
//假設判斷式為 True，接下來則需產生回傳 ACK 訊息
      if (flag) {
```

//使用 ClientTransaction 的 getDialog method 獲得屬於此 transaction 的 Dialog

**Dialog dialog = tid.getDialog();**

//利用 Dialog 的 createRequest() method 的建立 ACK request 物件，建立時必須給

//定 Request 名稱

**Request ackRequest = dialog.createRequest(Request.ACK);**

//使用 Dialog 的 sendACK() method 將 ACK request 傳送到網路上

**dialog.sendAck(ackRequest);**

//程式處理完 OK 訊息，開始建立媒體位元流

//呼叫我們實作在 AVTransmit2 程式裡的的 startTransmit() method

**AVTransmit.startTransmit(AVTransmit);**

//呼叫我們實作在 AVReceive2 程式裡的的 startReceiver() method，成功建立可同

//時傳送接收的媒體位元流

**AVReceive.startReceiver(AVReceive);**

}

**}**

**----------------------------------------------------------------------------------------------------**

以上範例程式是當 UAC 接收到 OK Response 訊息之後建立媒體位元流的方法，而在 3.4.6 節裡當 UAS 接收到 ACK 訊息之後建立媒體位元流的方法，也和上述的相同，以下 UAS 處理 ACK 訊息的範例程式：

----------------------------------------------------------------------------------------------------

```
public void JainSipPhone::processRequest(RequestEvent requestEvent) {
//首先先獲得 Request 物件
    Request request = requestEvent.getRequest();
//判斷 request 物件是否等於 ACK
    if (request.getMethod().equals(Request.ACK)) {
//呼叫我們實作在 AVTransmit2 程式裡的的 startTransmit() method
        AVTransmit.startTransmit(AVTransmit);
//呼叫我們實作在 AVReceive2 程式裡的的 startReceiver() method，成功建立可同
//時傳送接收的媒體位元流
        AVReceive.startReceiver(AVReceive);
        }
}
```