第三章　JAIN SIP

在這一章裡，我們將介紹 JAIN SIP 這套 API，包括 JAIN SIP 的起源、開發目的、JAIN SIP 的架構以及 JAIN SIP 裡物件和物件的相關聯性，最後則透過範例程式的介紹，讓使用者能更簡單的了解 JAIN SIP，並達成實作 SIP 協定的目的。

3.1 JAIN SIP 概述

Java APIs for Integrated Networks (JAIN)屬於 Java Community Process 裡活動的一部分，而 Java Community Process 則是負責致力於建立有關電話服務的 APIs，一開始，JAIN 全名為 Java APIs for Intelligent Network，後來則更名為 Java APIs for Integrated Networks 以更符合此論文廣大的範圍，JAIN 包含了許多專家群組，每一個群組則負責開發不同的 API 規格。

JAIN APIs 的主要目的在於抽象化底層網路，如此一來，我們便可以建立獨立於網路之上的服務。

JAIN SIP 包含在 JAIN 的活動之下，是一套針對 SIP 訊號協定所開發的 Java 標準介面，其中包含標準化以下的項目：

1. 標準化堆疊介面
2. 標準化訊息介面

3. 標準化事件和事件的語法

4. 應用程式的可攜性

JAIN SIP 提供開發者一套可以簡單實作 SIP 協定的工具，藉由他，我們可以更了解 SIP 協定，並實作 SIP 協定裡的各種實體，例如 user agent, proxy server, registrar server 等。

本論文使用 JAIN SIP API 協助開發 SIP 協定，以下介紹 JAIN SIP API 的介面內容，以及使用方法。

3.2 JAIN SIP Packages

JAIN SIP 裡包含了四個主要的 packages，以下介紹這些 packages 的主要內容。

1. General package- javax.sip

此 package 定義了用來建構 JAIN SIP 協定的堆疊、應用程式以及如何接收傳送 JAIN SIP 訊息的介面，並分成 Application Developer 和 Stack Vendors 兩種觀點，以 Application Developer 的觀點來說，開發者關心的地方在於實作 SipListener 的部份，SipListener 可以視為整個應用程式，而開發者必須實作

SipListener 所應提供的 method，SipListener 的定義和功能下面會詳細介紹，另外 Stack Vendors 則負責實作除了 SipListener 之外的所有 interface，其中最重要的包含 SipStack 和 SipProvider 兩個主要部份。

2. Address package- javax.sip.address

此 package 包含了代表 SIP 協定的 Addressing 部分，它定義了 URI 介面，而通稱為 URI 的介面之下又可分為 SipURI 和 TelURI。

除此之外，address package 還包含了 AddressFactory 類別，此類別定義了用來建立 address 的 method 以幫助建立 address 物件。

3. Message package- javax.sip.message

此 package 包含了代表 SIP 訊息的介面，通稱為 message 的介面之下又可分成 Request 和 Response 兩種。

除此之外，message package 還包含了 MessageFactory 類別，此類別定義了用來建立新的 request 和 response 的 method 以幫助建立 message 物件。

4. Header package- javax.sip.header

此 package 包含了所有定義在 SIP 協定裡的檔頭 interface，JAIN SIP 規格將每一個不同類型的檔頭定義為介面，而目前 JAIN SIP 支援所有定義在 RFC3261 裡的 Header。

除此之外，Header package 還包含了 HeaderFactory 類別，此類別定義了用來建立新 Header 的 method 以幫助建立 Header 物件。

3.3 JAIN SIP 物件介紹和物件間關係

1. 以 SipFactory 為中心產生其他介面和類別

在 JAIN 的主架構中，大量運用了物件導向設計的 Factory Pattern，使得應用程式的開發和 JAIN 的實作類別架構無關，增加應用程式的可攜性。而在 JAIN 的架構裡，應用程式是以 SipFactory 為中心產生其他各相關類別或介面，其中包含了 AddressFactory、HeaderFactory、MessageFactory 以及重要的 SipStack，如圖 3.1 所示，而我們可以藉由使用 getInstance() method 獲得一個獨一無二的 SipFactory。

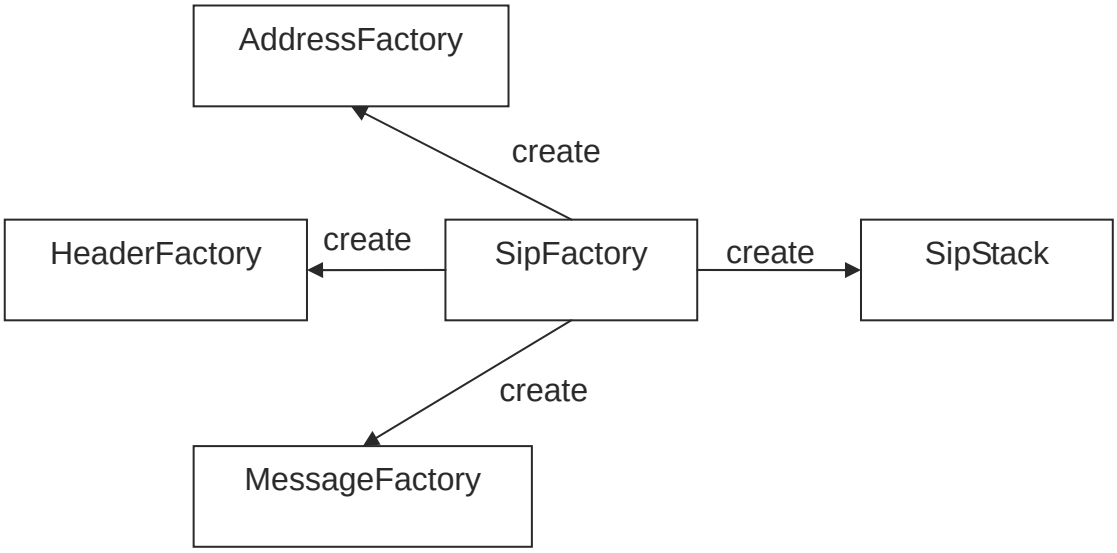

圖 3.1 以 SipFactory 為中心產生其他介面和類別

2. JAIN SIP 的主要物件架構

當使用 SipFactory 產生其他的介面或類別時，其中最重要的則是 SipStack 介面，我們可以將 SipStack 視為與外界溝通的網路卡，此介面可用來接收網路上傳來的訊息或物件，並且傳給內部的應用程式作處理，或者是將內部的訊息或物件傳送到網路上，每一張網路卡都會有一組屬於自己的 IP 位址，所以在同一個應用程式中，一個 IP 位址也只能對應一個 SipStack。

產生 SipStack 物件時必須先給定屬於此物件的屬性(Property)，屬性的內容包括有 IP Address、Stack Name、Outbound Proxy 等參數，並且藉由 SipFactory 的 createSipStack(Properties) method 建立。

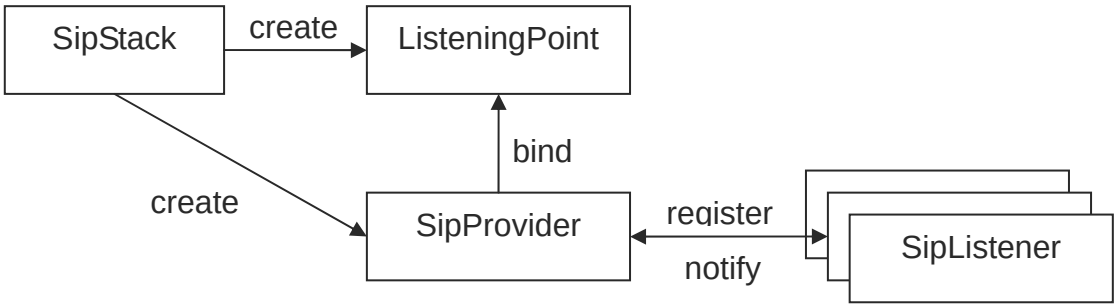

圖 3.2 JAIN SIP 的主要物件架構

擁有了 SipStack 介面之後，接下來我們必須為應用程式建立 SipProvider 介面，如圖三所示，SipProvider 是 JAIN SIP Event Model 的中心，藉由他，我們可以處理從 SipStack 傳送過來的事件，並且呼叫相對應的函式處理，每一個 SipStack 可能會有一個以上的 SipProvider，而每產生一個 SipProvider 都須給定一個 port number，所以不同 port number 的 SipProvider 則負責處理接收屬於自己 port number 傳來的訊息，我們可以藉由 SipStack 的 createSipProvider(ListeningPoint) method 建立新的 SipProvider。在 JAIN SIP 裡，Port Number 是用 ListeningPoint 介面代表。

有了 SipStack 和 SipProvider 之後，最後一個 JAIN SIP 裡的重要介面則為 SipListener，SipListener 必須向 SipProvider 註冊，以接收從 SipProvider 傳送過來的事件，而每一個 SipListener 可以向多個 SipProvider 註冊，如同可以接收不同 port 所傳送過來的事件，我們可以將 SipListener 介面視為開發者所實作的應用程式，而實作的 SipListener 介面時必須包含 processRequest()、processResponse()以

及 processTimeout()三種 method，分別處理接收到的 request、response 和 timeout 事件。

當 SipProvider 接收到來自 SipStack 的 Message 時 (Request 或 Response)，SipProvider 會將該 Request 或 Response 加入 Event 物件內，並透過對應的 Event Handler 通知 SipListener。例如，當接收到 Resquest 時，SipProvider 會呼叫已登錄 SipListener 的 processRequest()方法，此時應用程式便可作適當的處理。

3. 使用 MessageFactory 產生的 Message 物件

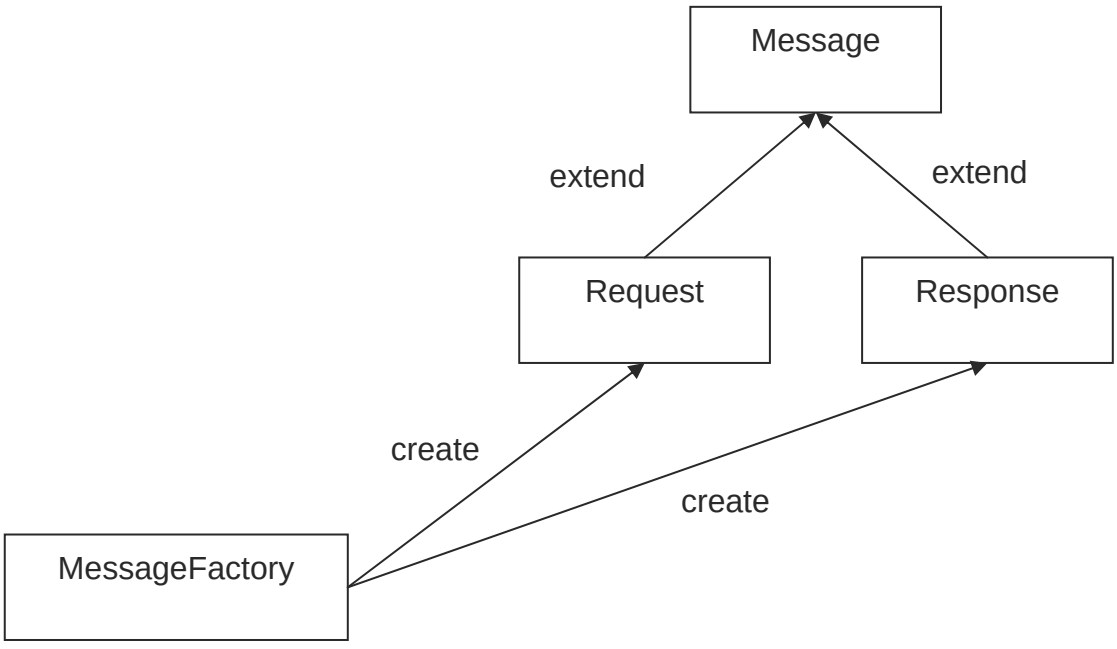

圖 3.3 使用 MessageFactory 產生的 Message 物件

當我們介紹 SipFactory 介面的時候提到 SipFactory 可以幫助我們產生

MessageFactory 介面，而 MessageFactory 則可以建立 SIP Protocol 裡的 Request 和 Response 訊息物件。

JAIN SIP 裡的 Message 物件可視為 Response 和 Request 的泛型 (Generic Type)，目的為將 Request 及 Response 兩者共用的方法集在同一界面上。而 Request 與 Reponse 對應 SIP Protocol 中的各類型 Request 與 Reponse，利用 MessageFactory 產生時，應用程式要給定所需的類型資訊 (例如 INVITE, ACK …或 180, 200 …)。

4.使用 AddressFactory 產生的 Address 物件

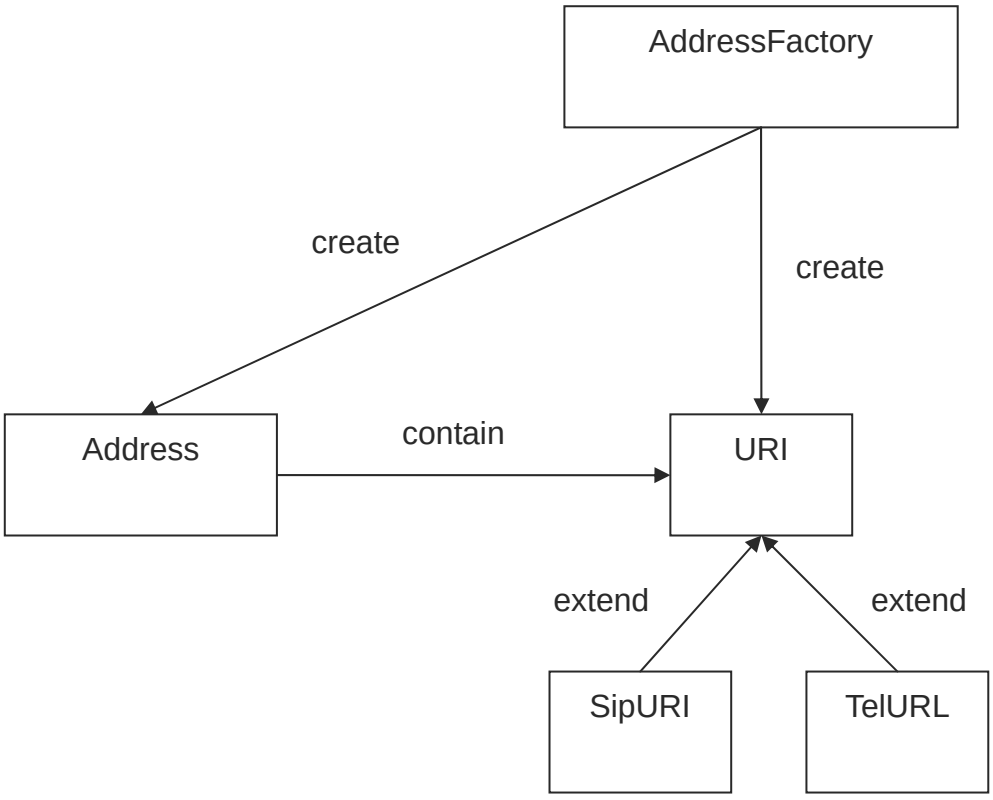

圖 3.4 使用 AddressFactory 產生的 Address 物件

AddressFactory 由 SipFactory 產生，產生後應用程式利用此物件產生 Sip Protocol 的各種 URI 相關物件，其中 URI 為 SipURI 與 TelURL 的泛型(Generic Type)，目的為將 SipURI 及 TelURL 兩者共用的方法集在同一界面上；以及預留日後擴充之用。另外值得注意的是，Address 物件除了包含 URI 物件之外，還包含了用來顯示使用者名稱的 display name，display name 通常會顯示在使用者介面上，以方便使用者判斷來電者為何。

5.使用 HeaderFactory 產生的 Header 物件

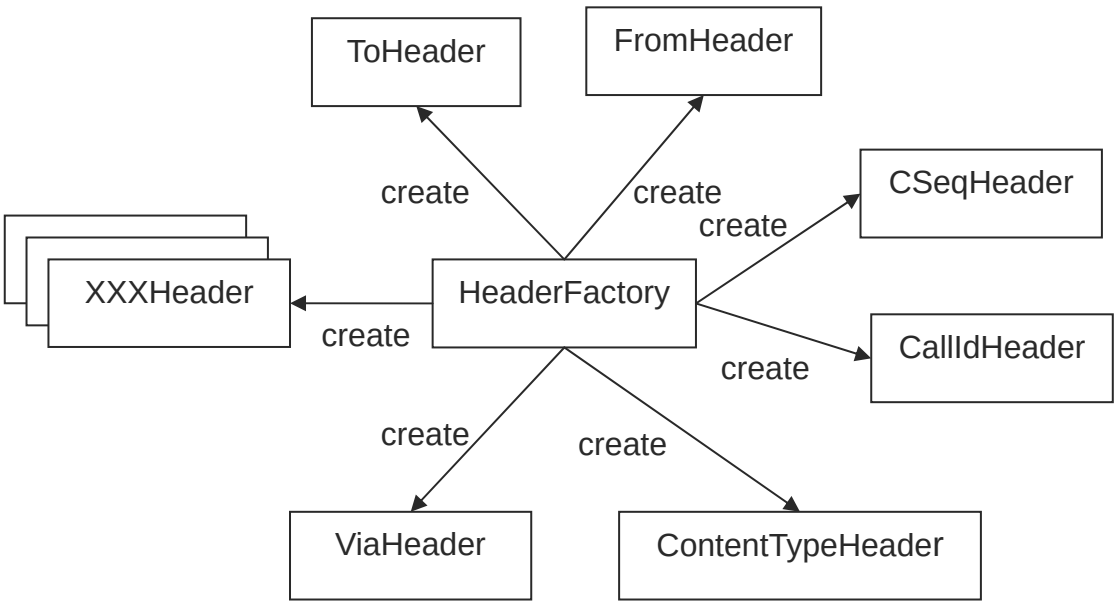

圖 3.5 使用 HeaderFactory 產生的 Header 物件

HeaderFactory 由 SipFactory 產生，產生後應用程式利用此物件產生 Sip

Protocol 的所有 Header 相關物件，例如 ToHeader、FromHeader、CseqHeader 等。JAIN SIP 規格將每一個不同類型的檔頭定義為物件，並且所有在 RFC3261 中定義的 Header 在 API 中都有相對應的類別。

3.4 實作範例程式介紹

本節將會詳細介紹如何透過 JAIN SIP API 實作 SIP 協定的訊息傳遞過程，以用來成功建立兩端點的連線，達成建立會議的功能。

由 2.3 節可知，SIP 訊息分成由 client 端傳送到 server 端的 request 訊息，以及 server 端傳送到 client 端的 response 訊息，因此我們所實作的 UA 必須具有傳送和接收 Request 和 Response 訊息的功能，由圖 3.6 得知，成功完成一個 transaction 必須包含三個步驟：

1. UAC 傳送 Request (INVITE)訊息給 UAS

2. UAS 收到 Request (INVITE)訊息，且回傳 Response 訊息 (180 Ringing 和 200 OK)

3. UAC 收到 UAS 回傳的 Response (180 Ringing 和 200 OK)訊息，並回傳 Request 訊息 (ACK)

本節將依此三個步驟實作討論範例程式，以達成建立會議的功能。而實作之

範例程式流程如圖 3.7 所示。

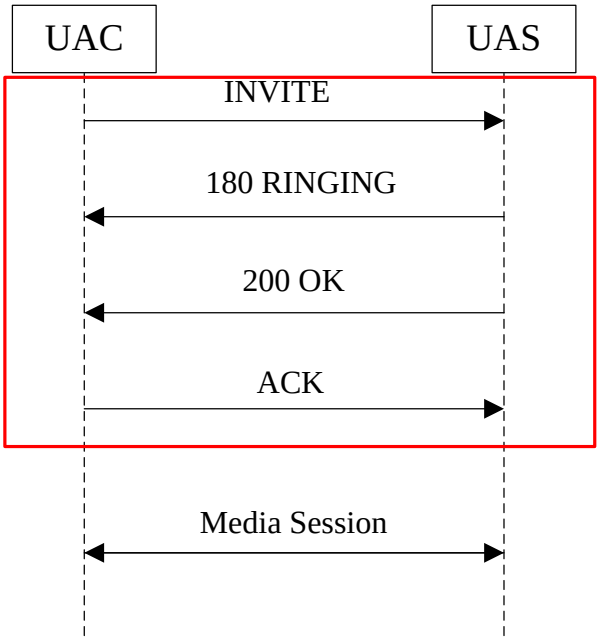

圖 3.6 簡單 SIP 會議例子

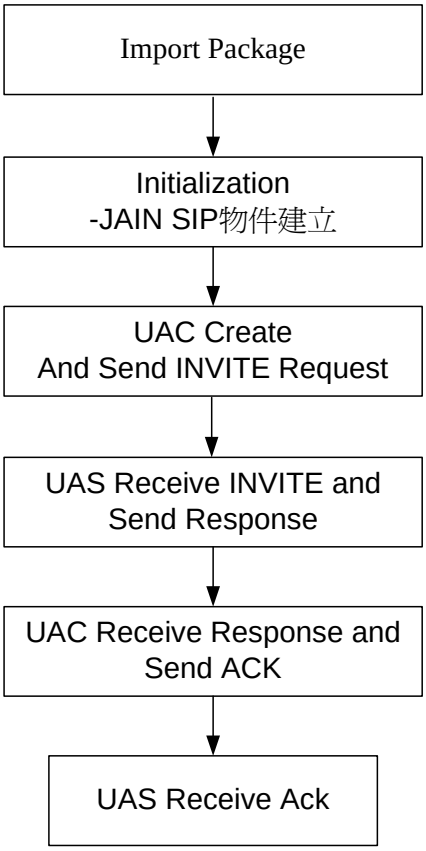

圖 3.7 實作範例程式流程

3.4.1 Import package

在 3.2 節裡我們有提到 JAIN SIP 的主要 package 包含四個部份，其中包括有

1. General package (main architectural interfaces)-javax.sip
2. Address package- javax.sip.address
3. Message package- javax.sip.message
4. Header package- javax.sip.header

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 javax.sip.*;
import javax.sip.address.*;
import javax.sip.header.*;
import javax.sip.message.*;
```

藉由 import 這些封包，我們可以使用 JAIN SIP 提供的 API，幫助我們成功建立 SIP 會議。

3.4.2 Initialization -JAIN SIP 物件建立

在上述的章節裡我們介紹了 JAIN SIP 的主要物件，以及物件和物件之間的關係，如圖 3.8 所示，接下來我們必須了解如何建立初始化這些物件，建立 SIP 協定最主要的物件包括 SipFactory、HeaderFactory、AddressFactory、MessageFactory、SipStack、SipProvider、ListeningPoint，在我們撰寫的 Application 程式裡，首先必須先獲得一個 SIP Factory 物件，藉由 SIP Factory 物件，我們才能產生 SIP Stack 物件，並且產生相對應於 SIP Stack 的 SIP Provider 物件，有了 SIP Provider 物件後，實作的應用程式便可向 SIP Listener 註冊，接收 SIP Provider 傳送過來的訊息。

以下分別介紹初始化這些物件的部份原始程式碼：

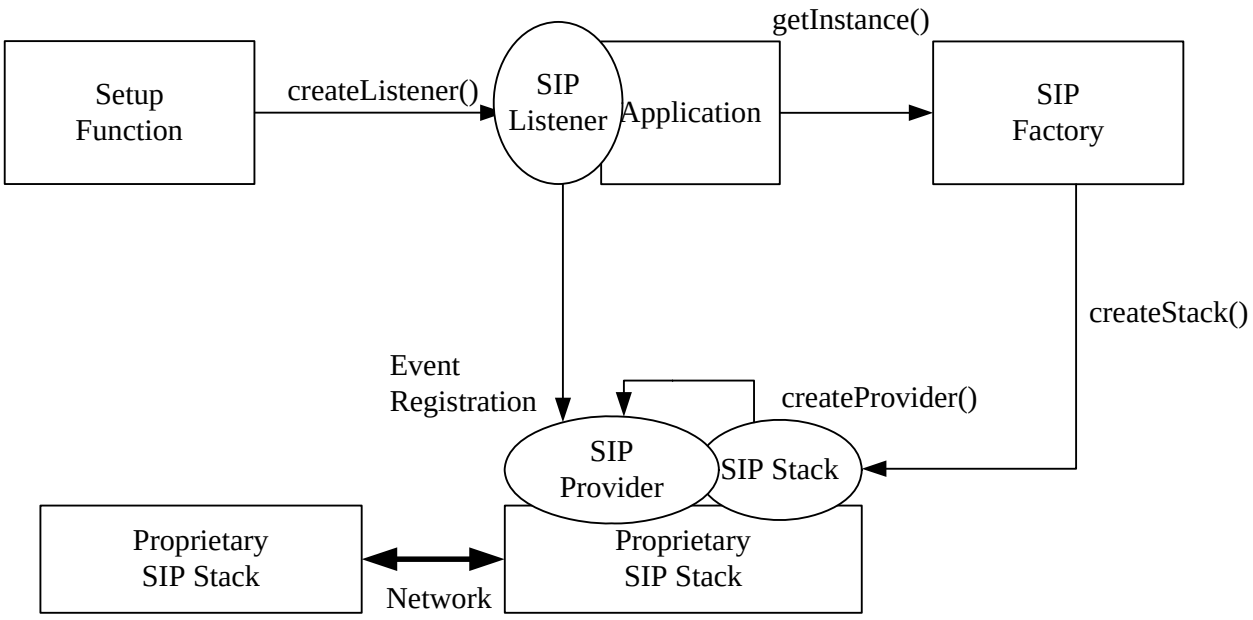

圖 3.8 JAIN SIP 的主要物件架構

1. SipFactory

//首先是建立 SipFactory，在 SipFactory 類別裡的 getInstance() method 定義為

//static，可用來產生 SipFactory 物件，在 Java 的定義裡，static method 可用來幫

//助應用程式直接使用類別產生物件，而不需要實際擁有這個物件才能使用它的

//method。

SipFactory sipFactory = SipFactory.getInstance();

2. SipStack

每一個 SipStack 物件都有屬於自己的名子，IP 位址等屬性，如圖 3.9 所示，所以在建立 SipStack 之前，我們必須建立 Properties 物件以儲存這些屬性，並用這些屬性產生 SipStack，在本範例程式裡我們必須幫 SipStack 建立四種屬性，如表 3.1 所示

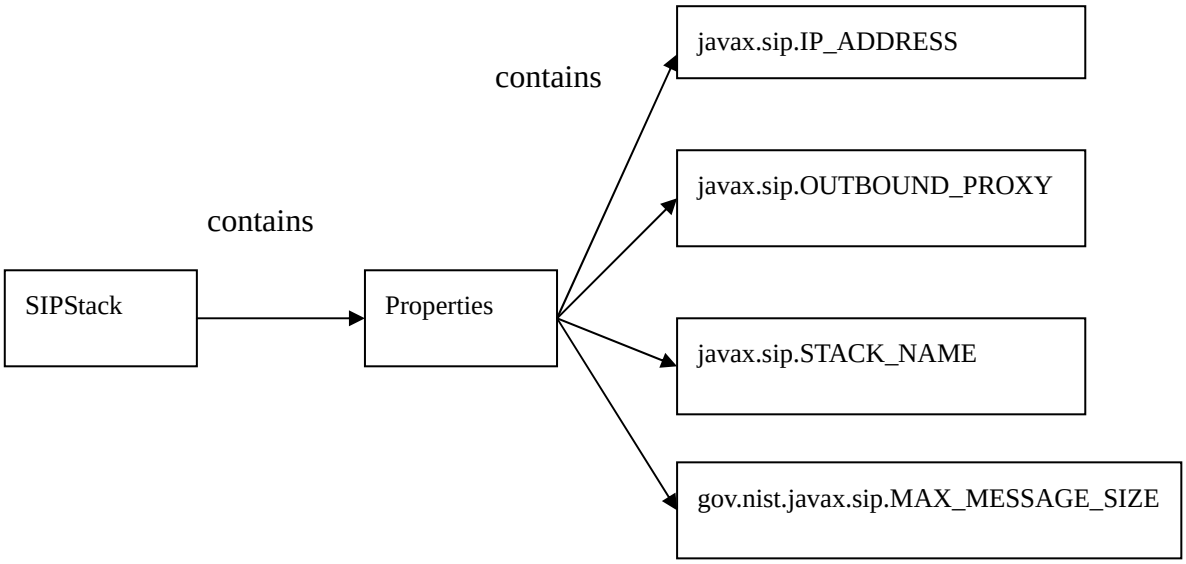

圖 3.9 SIPStack

屬性名稱	描述
javax.sip.IP_ADDRESS	SIP Stack 的 IP 位址
javax.sip.OUTBOUND_PROXY	SIP Stack 對外的 Proxy 位址，如果未透過 Proxy 傳送則直接輸入受話端位址
javax.sip.STACK_NAME	SIP Stack 名稱，可任意輸入

gov.nist.javax.sip.MAX_MESSAGE_SIZE	可接收的最大訊息長度

表 3.1 SIP Stack 的屬性列表

//建立 SipStack 物件之前必須先建立用來儲存 SipStack 屬性的 Properties 物件

//首先是建立新的 properties 物件

Properties properties = new Properties();

//設定代表 IP 位址的 IP_ADDRESS 屬性

properties.setProperty("javax.sip.IP_ADDRESS", myAddress);

//Outbound Proxy 代表 UAS 的位址，此位址有可能是 Proxy Server 或是點對點傳

//送的受話者位址，輸入的格式為「註冊位址 ：port number /使用的傳輸協定」，

//例如：140.122.185.202:5060/udp，在這裡我們將 port number 設為 5060，為一般

//sip 協定使用的標準號碼，使用的 transport 可為 UDP 或 TCP，端看設計者而定

properties.setProperty("javax.sip.OUTBOUND_PROXY",

registrarAddress + ":5060" + "/" + registrarTransport);

//設定 SipStack 的名稱，名稱可任意設定

properties.setProperty("javax.sip.STACK_NAME", "JainSipPhone");

//設定最大的訊息長度，在這裡我們將之設為 2 的 20 次方，若有超過此長度的訊

//息則視為惡意攻擊

properties.setProperty("gov.nist.javax.sip.MAX_MESSAGE_SIZE",

"1048576");

try {

//有了 SipFactory 物件後，我們便可以使用他的 createSipStack() method 建立新的

//SipStack，建立時必須給定事先設定好的 Property 物件

SipStack sipStack = sipFactory.createSipStack(properties);

} catch (PeerUnavailableException e) {

//如果無法成功建立 SipStack，則印出錯誤訊息

System.err.println(e.getMessage());

}

3. HeaderFactory

//透過 SipFactory 的 createHeaderFactory() method 建立 HeaderFactory

HeaderFactory headerFactory = sipFactory.createHeaderFactory();

4. AddressFactory

//透過 SipFactory 的 method 建立 AddressFactory

AddressFactory addressFactory = sipFactory.createAddressFactory();

5. MessageFactory

//透過 SipFactory 的 method 建立 MessageFactory

messageFactory = sipFactory.createMessageFactory();

6. ListeningPoint

如圖 3.10 所示，我們可以藉由 SIPStack 物件建立新的 ListeningPoint，ListeningPoint 代表了網路卡的 port number，所以建立時必須給定 port number 以及使用的傳輸協定，同樣的，我們也可以藉由 SIPStack 物件建立 SIPProvider 物件，用來接收處理 SIPStack 所傳送過來的事件，建立時必須給定之前所建立的 ListeningPoint，代表此 SipProvder 接收事件時所聆聽的 port number，以下為建立 ListeningPoint 的範例程式。

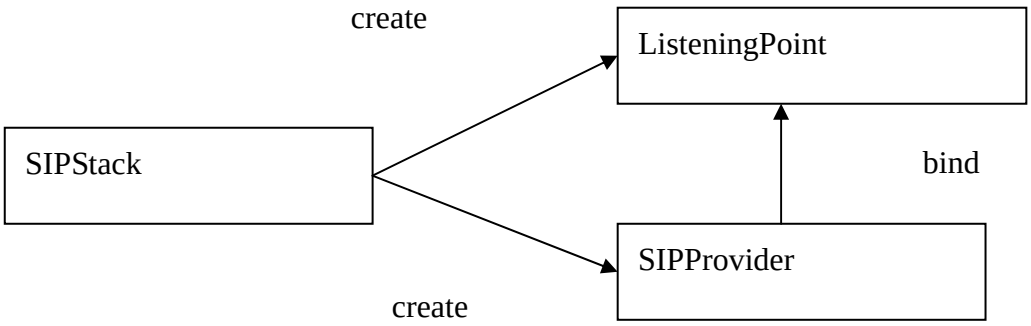

圖 3.10 SIPProvider 和 ListeningPoint

//有了 SipStack 物件之後，我們便可以使用它建立 ListeningPoint，這裡建立了一

//個 transport 為 udp 的 ListeningPoint 物件，同樣我們也可以建立使用 tcp transport

//的 ListeningPoint 物件，myPort 參數則是我們用來傳送訊息的 port number，通

//常為 5060

udpListeningPoint = sipStack.createListeningPoint (myPort, "udp");

7. SipProvider

//藉由 SipStack 的 createSipProvider() method 建立屬於此 SipStack 的 SipProvider

//物件，建立時必須給定之前建立的 ListeningPoint 物件。

sipProvider = sipStack.createSipProvider(udpListeningPoint);

8. ListeningPoint

//從圖 3.8 可知，在實作的應用程式裡我們必須實作 SipListner，實作 SipListener

//的方法在下一節裡將會提到，在這裡假設我們所實做的 SipListener 名稱為

//JainSipPhone，所以我們將 JainSipPhone 指定為 this (代表我們目前撰寫的應用

//程式)

JainSipPhone listener = this;

//從圖 3.8 可知，每一個 SipListener 都可向 SipProvider 註冊，以接收從 SIPProvider

//傳送過來的訊息，JAIN SIP 提供了 SipListener 向 SipProvider 註冊的

//addSipListener() method，如此一來 SipListener 便可接收此 SipProvider 傳送過來

//的訊息

sipProvider.addSipListener(listener);

建立完以上這些物件後，我們才能利用他們幫助傳送訊息以達成建立會議的目的。

3.4.3 UAC Create and Send INVITE Request

成功建立完 JAIN SIP 的基礎物件後，我們便可開始撰寫程式的第一個步驟：UAC 傳送 INVITE 訊息給 UAS，在建立訊息之前我們必須先建立好包含在訊息內的所有 Header，在這裡以 INVITE 訊息為例，如圖 3.11 所示，成功建立一個 INVITE 訊息須包含 CSeqHeader、 FromHeader、ToHeader、ViaHeaders、MaxForwards、CallIDHeader 六個檔頭物件，以下為建立及傳送 INVITE 訊息的原始程式碼。

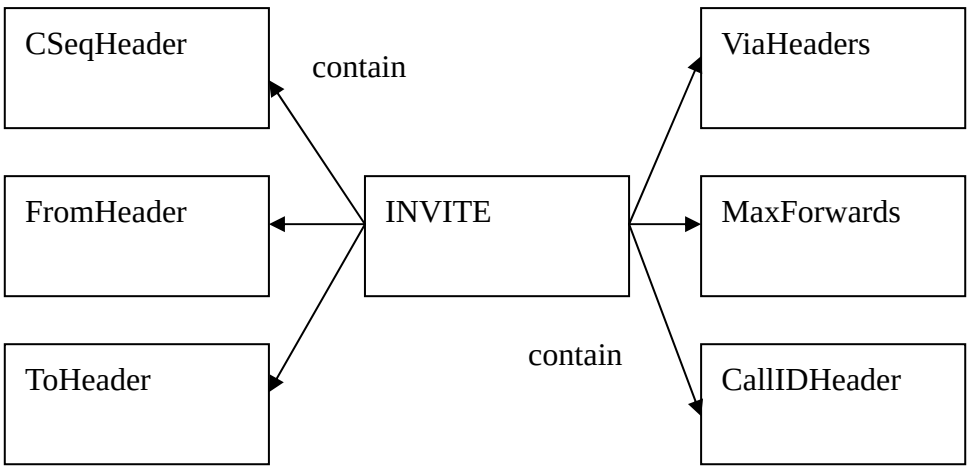

圖 3.11 Message

```
//FromHeader 記錄了呼叫端位址，首先使用 AddressFactory 的 createSipURI()
//method 建立 SipURI 物件，建立時必須給定呼叫端使用者名稱(fromUser)以及使
//用者位址(fromSipAddress)
    SipURI fromURI = (SipURI) addressFactory.createSipURI(fromUser,
        fromSipAddress);
//使用 addressFactory 的 createAddress() method 建立 Address 物件，建立時必須給
//定之前所建立的 SipURI (fromURI)物件
    Address fromAddress = addressFactory.createAddress(fromURI);
//Address 物件和 SipURI 物件的不同點在於 Address 物件包含了使用者的顯示名
//稱(DisplayName)，在這裡利用 Address 物件的 setDisplayName() method 設定
//Display Name
    fromAddress.setDisplayName(fromDisplayName);
//有了 Address 物件(fromAddress)之後，我們便可藉由 HeaderFactory 的
//createFromHeader() method 建立 FromHeader 物件，代表呼叫端的 tag 參數則藉
//由 hashCode() method 亂數產生
    fromHeader = headerFactory.createFromHeader(fromAddress,
        Integer.toString(hashCode()));
```

```
//使用 AddressFactory 的 createSipURI() method 建立 SIPURI 物件，在我們的範例
//裡因為此 INVITE 訊息需透過 Proxy 傳送，所以被呼叫端的位址為 Proxy 位址
//(registrarAddress)，並給定被呼叫端的使用者 ID (toUser)，以供 Proxy 尋找
        SipURI toURI =
            addressFactory.createSipURI(toUser, registrarAddress);
//使用 AddresFactory 的 createAddress() method 建立 Address 物件，並給定建立好
//的 SipURI 物件  (toURI)。
        Address toAddress = addressFactory.createAddress(toURI);
//設定 Address 物件(toAddress)  的 DisplayName
        toAddress.setDisplayName(toDisplayName);
//有了 Address 物件後便可使用 HeaderFactory 的 createToHeader() method 建立新
//的 ToHeader 物件，而代表接收端的 toTag 則須由接收端設定，所以我們先設成
//null
        ToHeader toHeader =
            headerFactory.createToHeader(toNameAddress, null);
//建立代表 request 主題的 requestURI，其中目的地位址為 Proxy 位址
//(registrarAddress)，toUser 則為被呼叫者 ID
        SipURI requestURI =
            addressFactory.createSipURI(toUser, registrarAddress);
```

// 建立 ViaHeader 物件，因為 ViaHeader 通常包含許多經過的位址資訊，所以先新增一個 ArrayList 屬於 ViaHeaders

ArrayList viaHeaders = new ArrayList();

//使用 headerFactory 的 createViaHeader() method 建立 ViaHeader 物件，建立時必須給定本地的 IP 位址 (ipAddress)、port number (listeningPort)、使用的傳輸協定(transport)，branch 會在日後由系統自動設定，在這裡我們先設為 null。

ViaHeader viaHeader =

headerFactory.createViaHeader(ipAddress, listeningPort,

transport, null);

//將新增的 ViaHeader 加入 ViaHeaders 的 ArrayList 裡

viaHeaders.add(viaHeader);

//使用 HeaderFactory 的 createCseqHeader() method 建立新的 CSeqHeader，建立時必須給定 sequence number，其初始值為 1，以及 request 的名稱 (Request.INVITE)

CSeqHeader cSeqHeader

=headerFactory.createCSeqHeader(1,Request.INVITE);

//使用 HeaderFactory 的 createMaxForwardsHeader() method 建立 MaxForwardsHeader，在這裡建議值為 70

MaxForwardsHeader maxForwards =

headerFactory.createMaxForwardsHeader(70);

//透過 SipProvider 的 getNewCallID() method 獲得新的 CallIDHeader，使用

//SipProvider 所提供的 method 可獲得代表此 dialog 的獨一獨二 Call ID，而

//HeaderFactory 提供的 method 則需手動輸入 Call ID 字串，並且不保證獨一性

CallIdHeader callIdHeader = sipProvider.getNewCallId();

//成功建立完 INVITE 訊息所需的 Header 之後，便可利用 MessageFactory 的

//createRequest() method 建立新的 INVITE 訊息

Request request =

messageFactory.createRequest(requestURI, Request.INVITE,

callIdHeader, cSeqHeader, fromHeader, toHeader, viaHeaders,

maxForwardsHeader);

//有了 INVITE 訊息之後，接下來產生新的 ClientTransaction 物件幫助我們將訊息

//傳送到網路上，而 ClientTransaction 物件可透過 SipProvider 獲得

//藉由 SipProvider 的 getNewClientTransaction() method 獲得新的

//ClientTransaction　物件，此物件是根據所給定的 request 訊息產生的，產生後便

//無法處理其他訊息

clientTransaction = sipProvider.getNewClientTransaction(request);

//利用 ClientTransaction 的 sendRequest() method 將訊息傳送到網路上，達成建立

//和傳送 INVITE Request 的目的

clientTransaction.sendRequest();

3.4.4 UAS Receive INVITE and Send Response

在介紹 UAS 端如何處理收到的 INVITE 訊息並回傳 Response 訊息時，我們先介紹 SipProvider 和 SipListener 之間的事件處理關係，如圖 3.12 所示：SipListener 是我們實作應用程式的一部份，而實作 SipListener 的目的則是為了接收從外界傳來的事件，事件的種類包含 Request Event、Response Event 和 Timeout Event，這些事件透過 SipListener 所實作的 method 由 SipProvider 傳送過來，例如 SipProvider 透過 processRequest(RequestEvent) method 傳送 Request 事件，當 SipListener 接收到事件後，則會進行相對應的處理，而我們在這裡所要作的則是撰寫如何處理這些事件的程式碼。

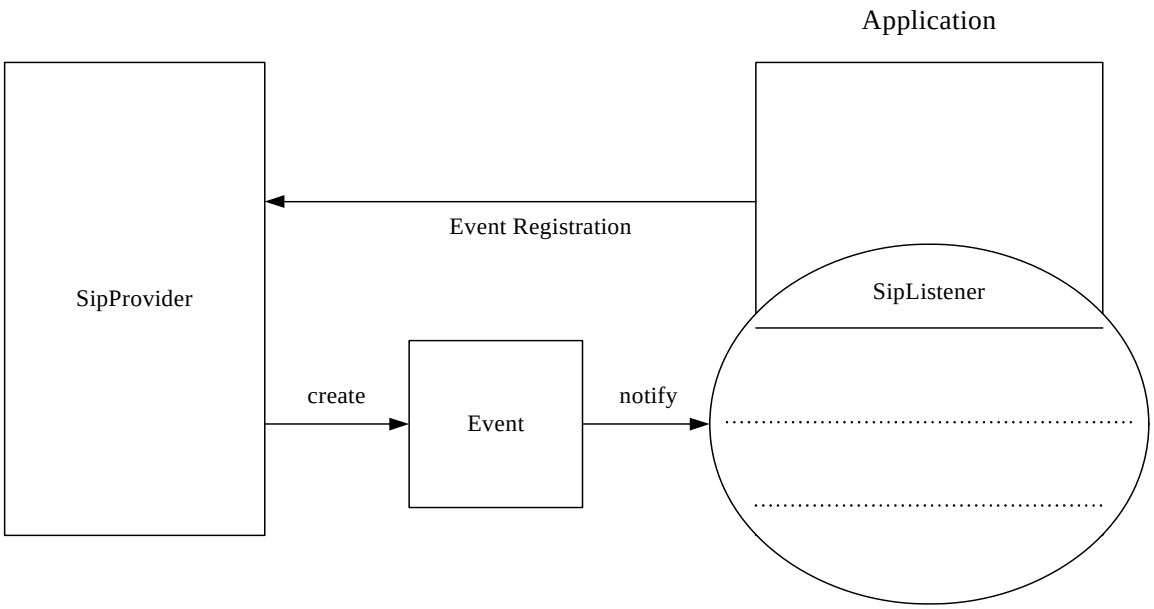

圖 3.12 Event Model Handler

以下介紹 UAS 如何透過 SipListener 實作的 processRequest() method 處理接收到的 INVITE 訊息，並回傳 Response 訊息。

//JainSipPhone 代表我們實作的應用程式

public void JainSipPhone::processRequest(RequestEvent requestEvent) {

//一開始我們必須先獲得接收到的 request 物件，在這裡使用 RequestEvent 的

//getRequest() method 獲得接收到的 request 事件

Request request = requestEvent.getRequest();

//接下來判斷 request 的類型，在這裡我們以 INVITE 訊息為例

if (request.getMethod().equals(Request.INVITE)) {

//當收到 INVITE 訊息之後，應用程式必須回傳 response 訊息給對方，首先是建

//立 180/Ringing 的 response 訊息，response 訊息是根據 request 訊息而來的，所

//以建立時除了給定 response 的 code number (180)之外，還包括相對的 request 物

//件，建立時使用 MessageFactory 的 createResponse() method。

Response response = messageFactory.createResponse(180, request);

//使用 Response 物件的 getHeader() method 獲得 response 的 ToHeader 物件，使用

//時須給定 Header (ToHeader.NAME)名稱

ToHeader toHeader

=(ToHeader)response.getHeader(ToHeader.NAME);

//使用 ToHeader 的 setTag() method 設定 toHeader tag 為 4321

toHeader.setTag("4321");

//有了 response 物件之後，我們便可將它傳送到網路上，首先是使用 SipProvider

//的 getNewServerTransaction() method 獲得屬於此 request 的 server transaction，使

//用時必須給定收到的 request 訊息

ServerTransaction serverTransaction

=sipProvider.getNewServerTransaction(request);

//最後是使用 ServerTransaction 的 sendResponse() method 將 response 傳送到網路

//上，使用時需給定要傳送的 response。

serverTransaction.sendResponse(response);

//傳送完 180/Ringing Response 接下來開始建立 200/Ok 的 response，建立的方法

//和 180/Ringing 大致相同

response = messageFactory.createResponse(200, request);

//獲得 ToHeader 物件

toHeader = (ToHeader) response.getHeader(ToHeader.NAME);

//設定 response toHeader tag=4321

toHeader.setTag("4321");

//在這裡我們使用之前建立的 ServerTransaction 物件將 200/OK 訊息傳送到網路上

serverTransaction.sendResponse(response);

}

}

3.4.5 UAC Receive Response and Send ACK

介紹完 SipListener 如何實作 processRequest()的部份，接下來則是 UAC 如何處理 UAS 所傳送過來的 Response 訊息，並回傳 ACK Request，以完成三方握手協定，處理 Response 的方法則是實作 processResponse() method，在這裡我們以收到 200/OK response 為例，以下為 processResponse()的原始程式碼。

---

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);
                }
}
```

3.4.6 UAS Receive Ack

當 UAS 收到 ACK 訊息之後，代表 SIP 會議已經成功建立，接下來則須開啟媒體位元流，進行媒體會議，我們在這一節裡將討論 UAS 如何透過 SipListener 實作的 processorRequest() method 處理 ACK Request，而接下來的開啟媒體位元流部份則將在第四章進行討論，以下是處理 ACK Request 的原始程式碼：

```
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)) {
//接下來進行 media transmission，此部分將在第四章做詳細的介紹
        …
        }
}
```