MQSeries®

# 使用 Java™

SC40-0625-06

MQSeries®

# 使用 Java™

SC40-0625-06

**附註！**

在使用本資訊及其支援的產品之前，請務必閱讀第357頁的『附錄F. 注意事項』下面的一般資訊。

**第 7 版（2001 年 1 月）**

本修訂版適用於 IBM® MQSeries Class for Java 第 5.2.0 版及 MQSeries Class for Java 訊息服務第 5.2 版和所有後續版次和修訂，直到新修訂版中另有說明爲止。

# 目錄

# 圖

# 表

# 關於本書

本書說明：

- MQSeries Class for Java，可用來存取 MQSeries 系統
- MQSeries Class for Java 訊息服務，可用來存取 Java 訊息服務 (JMS) 和 MQSeries 應用程式

**註:** 這份文件只有產品及 MQSeries 系列網站所提供的電子書格式（PDF 和 HTML），網址如下：

`http://www.ibm.com/software/mqseries/`

您**無法**訂購印刷版。

## 本書使用的縮寫

本書使用下列縮寫：

**MQ Java** MQSeries Class for Java 和 MQSeries Class for Java 訊息服務組合

**MQ base Java** MQSeries Class for Java

**MQ JMS** MQSeries Class for Java 訊息服務

## 本書對象

這份資訊專爲瞭解 *MQSeries Application Programming Guide* 中所說明之程序化 MQSeries 應用程式程式設計介面的程式設計師而寫，同時也說明如何轉化這項知識成爲使用 MQ Java 程式設計介面時的生產力。

## 使用者須知

您應該具備：

- Java 程式設計語言的知識
- 瞭解 *MQSeries Application Programming Guide* 中的「訊息佇列介面 (MQI)」相關章節及 *MQSeries Application Programming Reference* 一書中的「呼叫說明」相關章節中所說明的「訊息佇列介面」用途。
- 一般的 MQSeries 程式經驗，或熟悉其它 MQSeries 書籍的內容

要在 CICS® Transaction Server for OS/390® 中使用 MQ base Java 的使用者也應該熟悉下列各項：

- 「客戶資訊控制系統 (CICS)」概念
- 使用 CICS Java 應用程式程式設計介面 (API)
- 從 CICS 內執行 Java 程式

## 關於本書

要利用 VisualAge® for Java 來開發 OS/390 UNIX® System Services High Performance Java (HPJ) 應用程式的使用者應該熟悉 Enterprise Toolkit for OS/390（VisualAge for Java for OS/390 企業版第 2 版所提供）。

## 如何使用本書

本書的第 1 篇說明如何使用 MQ base Java 和 MQ JMS，第 2 篇協助程式設計師使用 MQ base Java，第 3 篇協助程式設計師使用 MQ JMS。

請閱讀第 1 篇的各章，它們會向您介紹 MQ base Java 和 MQ JMS。之後，再利用第 2 或 3 篇的程式設計指南，瞭解如何利用這些類別在要使用的環境中傳送和接收 MQSeries 訊息。

本書最後有一份名詞解釋和參考書目。

# 變更摘要

本節說明 *MQSeries 使用 Java* 這個修訂版中的變更。前一修訂版之後的變更，左側會有垂直線標示。

## 本版變更 (SC40-0625-06)

這個修訂版包括 MQ Java 第 5.2 版所引進的新功能。其中包括：

- 安裝程序更新。請參閱第7頁的『第2章 安裝程序』。
- 連線儲存池支援，用以改進使用多重連線來連接 MQSeries 佇列管理程式的應用程式和中介軟體之效能。請參閱：
  - 第62頁的『連線儲存池』
  - 第86頁的『MQEnvironment』
  - 第121頁的『MQPoolServices』
  - 第122頁的『MQPoolServicesEvent』
  - 第124頁的『MQPoolToken』
  - 第138頁的『MQQueueManager』
  - 第148頁的『MQSimpleConnectionManager』
  - 第152頁的『MQConnectionManager』
  - 第151頁的『MQPoolServicesEventListener』
  - 第159頁的『ManagedConnection』
  - 第162頁的『ManagedConnectionFactory』
  - 第164頁的『ManagedConnectionMetaData』
- 新的訂閱者佇列配置選項，用以提供發佈/訂閱應用程式的多重佇列和共用佇列方式。請參閱：
  - 第36頁的『內容』
  - 第183頁的『配置基本訂閱者佇列』
  - 第313頁的『Topic』
  - 第317頁的『TopicConnectionFactory』
- 新訂閱者清除公用程式，用以避免因訂閱者物件不正常關閉而造成的任何問題。請參閱第186頁的『訂閱者清除公用程式』。
- 應用程式伺服器機能支援，也就是訊息的並行處理。請參閱：
  - 第205頁的『第13章 MQ JMS 應用程式伺服器機能』
  - 第239頁的『ConnectionConsumer』
  - 第284頁的『QueueConnection』
  - 第297頁的『Session』
  - 第315頁的『TopicConnection』
- 更新 LDAP 伺服器配置資訊。請參閱第347頁的『附錄C. Java 物件的 LDAP 伺服器配置』。

變更

- 使用 X/Open XA 通信協定的分散式異動之支援。也就是說，MQ JMS 包括 XA 類別，因而 MQ JMS 能夠參與適當異動管理程式所協調的兩段式確定。請參閱：
  - 第351頁的『附錄E. 使用 WebSphere 的 JMS JTA/XA 介面』
  - 第329頁的『XAConnection』
  - 第330頁的『XAConnectionFactory』
  - 第331頁的『XAQueueConnection』
  - 第332頁的『XAQueueConnectionFactory』
  - 第334頁的『XAQueueSession』
  - 第335頁的『XASession』
  - 第337頁的『XATopicConnection』
  - 第338頁的『XATopicConnectionFactory』
  - 第340頁的『XATopicSession』

## 第 6 版變更 (SC40-0625-05)

包括 Linux 支援。

## 第 5 版變更 (SC40-0625-04)

### WebSphere™和 MQSeries Integrator 第 2 版之支援

MQ base Java 第 5.1.2 版現在可當作一項產品延伸來使用。它提供下列功能：

- 連接至 MQSeries Integrator for Windows NT® 第 2.0 版，以提供發佈/訂閱支援。請參閱第349頁的『附錄D. 連接至 MQSeries Integrator 第 2 版』，以取得進一步的詳細資料。
- 使用 WebSphere 的 CosNaming JNDI 服務提供者。請參閱第30頁的『配置』，以取得進一步的詳細資料。

# 第1篇 使用者指引

# 第1章 入門

本章總覽 MQSeries Class for Java 和 MQSeries Class for Java 訊息服務及其用法。

## 什麼是 MQSeries Class for Java？

MQSeries Class for Java (MQ base Java) 可讓利用 Java 程式設計語言所撰寫的程式執行下列動作：

- 作爲 MQSeries 從屬站而連接至 MQSeries
- 直接連接至 MQSeries 伺服器

它使 Java Applet、應用程式和 Servlet 能夠呼叫和查詢 MQSeries。這提供了存取大型電腦和舊型應用程式的功能，通常是透過網際網路，從屬站機器中不需要有任何其它 MQSeries 程式碼。透過 MQ base Java，網際網路終端機的使用者可以成爲異動的眞正參與者，而不只是資訊的提供和接收者而已。

## 什麼是 MQSeries Class for Java 訊息服務？

MQSeries Class for Java 訊息服務 (MQ JMS) 是一組 Java 類別，它們實作 Sun 的 Java 訊息服務 (JMS) 介面，使 JMS 程式能夠存取 MQSeries 系統。 JMS 的點對點和發佈/訂閱兩種模型都有支援。

將 MQ JMS 當作 API 來撰寫 MQSeries 應用程式有許多好處。部份好處來自 JMS 是有多重實作的開放標準。其它好處來自 MQ JMS（而不是 MQ base Java）所提供的特性。

使用開放標準所帶來的好處包括：

- 在技術和應用程式碼這兩方面提供投資保護
- 熟悉 JMS 應用程式設計的人員比較容易取得
- 外掛不同 JMS 實作來配合不同需求的功能

如果需要 JMS API 有什麼好處的詳細資訊，請造訪 Sun 的網址，網址如下：`http://java.sun.com`。

透過 MQ base Java 而額外提供的功能有：

- 非同步傳訊遞送
- 訊息選取元
- 支援發佈/訂閱傳訊
- 結構化訊息類別

## 誰應該使用 MQ Java？

如果您的企業符合下列中的任何實務，使用 MQSeries Class for Java 和 MQSeries Class for Java 訊息服務可爲您帶來非常大的好處：

- 引進內部網路型主從式解決方案的中大型企業。 在這裡，網際網路技術提供成本又低、又容易進行的全球通信存取功能， 而 MQSeries 連線則提供具有可靠遞送能力和時間獨立性的高度整合功能。

## 誰應該使用 MQ Java

- 需要有與夥伴企業間可靠的企業對企業通信的中大型企業。 在這裡，網際網路同樣提供成本又低、又容易進行的全球通信存取功能， 而 MQSeries 連線則提供具有可靠遞送能力和時間獨立性的高度整合功能。
- 想要透過公開的網際網路，供他人存取其企業應用程式的中大型企業。 在這裡，網際網路提供成本又低、又可以遍及全球的觸角， 而 MQSeries 連線則透過佇列作業的典範來提供高度整合功能。 除了成本低之外，企業也可以因爲提供 24 小時的服務、回應快速及越來越好的精確度，而提高客戶的滿意度。
- 網際網路服務提供者，或其他加值型網路提供者。 這些公司可以利用網際網路所提供的低成本又容易進行的通信功能。他們也可以利用 MQSeries 連線所提供的高度整合功能來附加價值。 利用 MQSeries 的網際網路服務程式提供者可以即時認可收到 Web 瀏覽器送來的輸入資料、保證遞送的有效， 及提供簡單的方法讓 Web 瀏覽器的使用者能監視訊息的狀態。

MQSeries 和 MQSeries Class for Java 訊息服務提供可用來存取企業應用程式及開發複合式 Web 應用程式的最佳基礎架構。 Web 瀏覽器所發出的服務要求可以放入佇列中，等可能之時再繼續處理，這樣，不論系統的負荷如何，都能即時將回應傳回給一般使用者。用網路詞彙來說，藉著將這個佇列放在使用者的「近處」，網路的負載便不會衝擊到回應的即時性。另外，MQSeries 的異動性質也表示在異動方式中，瀏覽器所發出的簡單要求可以放心展開成一系列的個別後端程序。

MQSeries Class for Java 也使應用程式開發人員能夠利用 Java 程式設計語言的力量來建立 Applet 和應用程式，以執行於任何支援 Java 執行環境的平台上。這些因素組合起來，便大幅縮減了多平台 MQSeries 應用程式的開發時間。另外，如果未來有 Applet 的加強功能的話，一般使用者也可以隨著 Applet 程式碼的下載而自動取得它們。

---

## 連線選項

可程式選項使 MQ Java 能夠採用下列中的任何方式來連接到 MQSeries：

- 作爲使用傳輸控制通信協定/網際網路通信協定 (TCP/IP) 的 MQSeries 從屬站
- 採用連結模式，直接連到 MQSeries

Windows NT 中的 MQ base Java 也可以利用 VisiBroker for Java 來進行連接。 第5頁的表1.顯示每個平台能夠使用的連接模式。

*表 1. 平台和連線模式*

| 伺服器平台 | 連線模式 | | |
|---|---|---|---|
| | 從屬站 | | 連結 |
| | 標準 | VisiBroker | |
| Windows NT | 是 | 是 | 是 |
| Windows® 2000 | 是 | 否 | 是 |
| AIX® | 是 | 否 | 是 |
| Sun OS（第 4.1.4 版或更早的版本） | 是 | 否 | 否 |
| Sun Solaris（第 2.6、2.8、7 版或 SunOS 第 5.6、5.7 版） | 是 | 否 | 是 |
| OS/2® | 是 | 否 | 是 |
| OS/400® | 是 | 否 | 是 |
| HP-UX | 是 | 否 | 是 |
| AT&T GIS UNIX | 是 | 否 | 否 |
| SINIX 和 DC/OSx | 是 | 否 | 否 |
| OS/390 | 否 | 否 | 是 |
| Linux | 是 | 否 | 否 |

**註:**

1. HP-UX Java 連結支援只適用於執行 POSIX draft10 pthreaded 版 MQSeries 的 HP-UX 第 11 版系統。 您也需要 HP-UX Developer's Kit for Java 1.1.7 (JDK™)，版次 C.01.17.01 或以上。
2. 在 HP-UX 第 10.20 版、Linux、Windows 95 和 Windows 98 中， 只支援 TCP/IP 從屬站連線。

下列各節會更詳細說明這些選項。

## 從屬站連線

如果要使用 MQ Java 作爲 MQSeries 從屬站，您可以將它安裝在可以有 Web 伺服器的 MQSeries 伺服器機器中，或安裝在個別機器中。如果您將 MQ Java 安裝在 Web 伺服器的相同機器中，好處在於即使沒有在本端環境內安裝 MQ Java 的機器，也可以下載和執行 MQSeries 從屬站應用程式。

不論您選擇在哪裡安裝從屬站，您都可以採用三種不同的模式來執行它：

**從任何具有 Java 功能的 Web 瀏覽器**

在這個模式中，可存取的 MQSeries 佇列管理程式位置可能會受限於使用的瀏覽器之安全限制。

**使用 Applet Viewer**

如果要使用這個方法，您的從屬站機器必須安裝有 Java Developer's Kit (JDK) 或 Java Runtime Environment (JRE)。

**作為獨立的 Java 程式，或在 Web 應用程式伺服器中**

如果要使用這個方法，您的從屬站機器必須安裝有 Java Developer's Kit (JDK) 或 Java Runtime Environment (JRE)。

### 使用 VisiBroker for Java

在 Windows 平台中，使用 VisiBroker 的連線是使用標準 MQSeries 從屬站通信協定的替代方案。這項支援是 VisiBroker for Java 結合 Netscape Navigator 所提供的，它需要 VisiBroker for Java 和 MQSeries 伺服器機器中的 MQSeries 物件伺服器。 MQ base Java 提供有適用的物件伺服器。

### 連結連線

當採用連結模式時，MQ Java 會使用 Java 原生介面 (JNI) 來直接呼叫現有的佇列管理程式 API，而不會透過網路來通信。這個方式所提供的 MQSeries 應用程式效能比使用網路連線好。和從屬站模式不同，使用連結模式來撰寫的應用程式不能當作 Applet 來下載。

如果要使用連結連線，MQ Java 必須安裝在 MQSeries 伺服器中。

## 必備條件

如果要執行 MQ base Java，您需要下列軟體：

- 您要使用的伺服器平台所適用的 MQSeries。
- 伺服器平台所適用的 Java Developers Kit (JDK)。
- Java Developers Kit， 或 Java Runtime Environment (JRE)， 或從屬站平台具有 Java 功能的 Web 瀏覽器。（請參閱第5頁的『從屬站連線』。）

  **註:** 如果要在 Web 瀏覽器中執行 MQ base Java Applet（比方說，安裝驗證程式），您需要可執行 Java 1.1.6 Applet 的瀏覽器。 Sun System 的 HotJava™、Netscape Navigator 4 及 Microsoft® Internet Explorer 4 都是符合這項需求的瀏覽器範例。
- VisiBroker for Java（只有在執行於有 VisiBroker 連線的 Windows 時才需要）。
- (OS/390) OS/390 第 2.5 版，含 UNIX System Services。
- (OS/400) AS/400® Developer Kit for Java, 5769-JV1，及 Qshell Interpreter，OS/400 (5769-SS1) Option 30。

如果要使用 MQ JMS 管理工具（請參閱第29頁的『第5章 使用 MQ JMS 管理工具』），您需要下列其它軟體：

- 至少下列其中一個服務提供者套件：
  - 輕裝備目錄存取通信協定 (LDAP) - `ldap.jar`、`providerutil.jar`。
  - 檔案系統 - `fscontext.jar`、`providerutil.jar`。
- Java 命名和目錄服務 (JNDI) 服務提供者。這是儲存管理物件之實體表示的資源。 MQ JMS 的使用者也許可以利用 LDAP 伺服器來做到這一點， 但這個工具也支援使用檔案系統環境定義服務提供者。 如果使用 LDAP 伺服器的話，必須配置它來儲存 JMS 物件。 如果需要這項配置的協助資訊，請參閱第347頁的『附錄C. Java 物件的 LDAP 伺服器配置』。

如果要使用 MQ JMS 的 XOpen/XA 機能，您需要 MQSeries 第 5.2 版。

# 第2章 安裝程序

本章說明如何安裝 MQSeries Class for Java 和 MQSeries Class for Java 訊息服務產品。

## 安裝 MQSeries Class for Java 和 MQSeries Class for Java 訊息服務

這個產品適用於 AIX、AS/400、HP-UX、Linux、Sun Solaris 和 Windows 平台。它含有：

- MQSeries Class for Java (MQ base Java) 第 5.2 版
- MQSeries Class for Java 訊息服務 (MQ JMS) 第 5.2 版（不是 AS/400）

| 關於各特定平台所能使用的連線，請參閱第4頁的『連線選項』。

產品以壓縮格式檔案來提供，您可以從 MQSeries 網站取得這些檔案，網址如下：`http://www.ibm.com/software/mqseries/`。

**註:** 對於 OS/390 而言， MQ base Java 是作爲 MQSeries SupportPac™ 來提供的，您可以從 `http://www.ibm.com/software/mqseries/` 中下載它。

如果只需要 MQ base Java 類別的最新版本，您可以只安裝 MQ base Java 第 5.2.0 版。如果要使用 MQ JMS 應用程式，您必須安裝 MQ base Java 和 MQ JMS（合稱爲 MQ Java）。

MQ base Java 內含於下列 Java `.jar` 檔中：

| | |
|---|---|
| **com.ibm.mq.jar** | 這個程式碼包含所有連線選項的支援。 |
| **com.ibm.mq.iiop.jar** | 這個程式碼只支援 VisiBroker 連線。只有在 Windows 平台中有提供這個程式碼。 |
| **com.ibm.mqbind.jar** | 這個程式碼只支援連結連線，在所有平台中，都不提供或支援它。建議您不要在任何新應用程式中使用它。 |

MQ JMS 內含於下列 Java `.jar` 檔：

**com.ibm.mqjms.jar**

Sun Microsystems 所提供的下列 Java 程式庫會檢附在 MQ JMS 產品中：

| | |
|---|---|
| **connector.jar** | 第 1.0 版公開初稿 |
| **fscontext.jar** | 早期存取第 4 版次 |
| **jms.jar** | 第 1.0.2 版 |
| **jndi.jar** | 第 1.1.2 版 |
| **ldap.jar** | 第 1.0.3 版 |
| **providerutil.jar** | 第 1.0 版 |

## 安裝 MQ base Java 和 MQ JMS

**註:** 在 OS/390 中，只提供 **com.ibm.mq.jar** 檔。這個檔案支援從 UNIX System Services 和 CICS Transaction Server for OS/390 通往 MQSeries 的連結連線。

如果需要安裝示指，請參閱所需平台的相關章節：

| | |
|---|---|
| **AIX、HP-UX 和 Sun Solaris** | 『在 UNIX 中安裝』 |
| **AS/400** | 第9頁的『在 AS/400 中安裝』 |
| **Linux** | 第9頁的『在 Linux 中安裝』 |
| **Windows** | 第10頁的『在 Windows 中安裝』 |

安裝好之後，檔案和範例會安裝在第10頁的『安裝目錄』所顯示的位置中。

安裝好之後，您必須如第10頁的『環境變數』所示來更新您的環境變數。

**註:** 如果您安裝產品的話，請小心，之後，再安裝或重新安裝基礎 MQSeries。請確定您沒有安裝 MQ base Java 第 5.1 版，因爲您的 MQSeries Java 支援會倒退一個層次。

## 在 UNIX 中安裝

這一節說明如何在 AIX、HP-UX 和 Sun Solaris 中安裝 MQ Java。如果需要在 Linux 中安裝 MQ base Java 的相關資訊，請參閱第9頁的『在 Linux 中安裝』。

**註:** 如果這是從屬站專用的安裝結構（也就是說，沒有安裝 MQSeries 伺服器），您必須設定群組和使用者 ID mqm。如果需要詳細資訊，請參閱平台所適用的「MQSeries 快速入門」手冊。

1. 以 root 身份登入。
2. 以二進位格式複製 `ma88_xxx.tar.Z` 檔，將它儲存在 /tmp 目錄中，其中 *xxx* 是適當的平台識別碼：
   - `aix` AIX
   - `hp10` HP-UXv10
   - `hp11` HP-UXv11
   - `sol` Sun Solaris
3. 輸入下列指令（其中 *xxx* 是適當的平台識別碼）：

   ```
   uncompress -fv /tmp/ma88_xxx.tar.Z
   tar -xvf /tmp/ma88_xxx.tar
   rm /tmp/ma88_xxx.tar
   ```

   這些指令會建立必要的檔案和目錄。
4. 使用每個平台所適用的安裝工具：
   - 如果是 AIX，請使用 `smitty`，並且：
     a. 解除安裝所有開頭爲 mqm.java 的元件。
     b. 從 /tmp 目錄中安裝各個元件。
   - 如果是 HP-UX，請利用 `sam`，並依適當情況從 `ma88_hp10` 或 `ma88_hp11` 檔中進行安裝。

**註:** Java 不支援字碼頁 1051（這是 HP-UX 的預設值）。如果要在 HP-UX 中執行發佈/訂閱分配管理程式，您可能需要將分配管理程式之佇列管理程式的 CCSID 改成替代值，如 819。

- 如果是 Sun Solaris，請輸入下列指令，並選取您需要的選項：

  ```
  pkgadd -d /tmp mqjava
  ```

  之後，再輸入下列指令：

  ```
  rm -R /tmp/mqjava
  ```

## 在 AS/400 中安裝

這一節說明如何在 AS/400 中安裝 MQ base Java。

1. 將 ma88_400.zip 檔複製到 PC 中的某目錄中。
2. 利用 InfoZip 的 Unzip 公用程式，將檔案解壓縮。

   這時會建立 ma88_400.sav 檔。
3. 在 AS/400 中的適當程式庫中，建立一個稱爲 MA88 的儲存檔，比方說，QGPL 程式庫：

   ```
   CRTSAVF FILE(QGPL/MA88)
   ```
4. 以二進位影像的方式，將 ma88_400.sav 轉送到這個儲存檔中。如果您要利用 FTP 來執行這個動作，put 指令應該類似於：

   ```
   PUT C:\TEMP\MA88_400.SAV QGPL/MA88
   ```
5. 利用 RSTLICPGM 安裝 MQSeries Class for Java，產品 ID 5648C60：

   ```
   RSTLICPGM LICPGM(5648C60) DEV(*SAVF) SAVF(QGPL/MA88)
   ```
6. 刪除步驟『3.』所建立的儲存檔：

   ```
   DLTF FILE(QGPL/MA88)
   ```

## 在 Linux 中安裝

這一節說明如何在 Linux 中安裝 MQ Java。

Linux 有兩個可用的安裝檔：ma88_linux.tgz 和 MQSeriesJava-5.2.0-1.noarch.rpm。每個檔案都提供相同的安裝結構。

如果您有目標系統的最高存取權，或您利用 Red Hat Package Manager (RPM) 資料庫來安裝套裝軟體，請使用 MQSeriesJava-5.2.0-1.noarch.rpm。

如果您沒有目標系統的最高存取權，或目標系統沒有安裝 RPM，請使用 ma88_linux.tgz。

如果要使用 ma88_linux.tgz 來安裝：

1. 選取一個產品的安裝目錄（如 /opt）。

   如果這個目錄不在您的起始目錄中，您可能需要以 root 身份登入。
2. 將 ma88_linux.tgz 檔複製到您的起始目錄中。
3. 將目錄切換至您選取的安裝目錄，例如：

   ```
   cd /opt
   ```
4. 輸入下列指令：

   ```
   tar -xpzf ~/ma88_linux.tgz
   ```

這時會在現行目錄（如 /opt）中建立和移入名稱爲 mqm 的目錄。

如果要使用 MQSeriesJava-5.2.0-1.noarch.rpm 來安裝：

1. 以 root 身份登入。
2. 將 MQSeriesJava-5.2.0-1.noarch.rpm 複製到工作目錄中。
3. 輸入下列指令：

```
rpm -i MQSeriesJava-5.2.0-1.noarch.rpm
```

這時會將產品安裝到 /opt/mqm/ 中。您也可以安裝到不同的路徑中（請參閱 RPM 文件，以取得詳細資料）。

## 在 Windows 中安裝

這一節說明如何在 Windows 中安裝 MQ Java。

1. 建立一個稱爲 tmp 的空目錄，使它成爲現行目錄。
2. 將 `ma88_win.zip` 檔案複製到這個目錄中。
3. 利用 InfoZip 的 Unzip 公用程式，將 `ma88_win.zip` 解壓縮。
4. 從這個目錄中執行 `setup.exe`，再遵循結果視窗中的提示。

**註:** 如果您只要安裝 MQ base Java，請在這個階段選取相關的選項。

## 安裝目錄

MQ Java 第 5.2 版檔案安裝在表2.所顯示的目錄中。

*表 2. 產品安裝目錄*

| 平台 | 目錄 |
|---|---|
| AIX | usr/mqm/java/ |
| AS/400 | /QIBM/ProdData/mqm/java/ |
| HP-UX 和 Sun Solaris | opt/mqm/java/ |
| Linux | *install_dir*/mqm/java/ |
| Windows 95、98、2000 和 NT | *install_dir*\ |
| **註:** *install_dir* 是產品的安裝目錄。在 Linux 中，這可能是 /opt。 | |

## 環境變數

安裝好之後，您必須更新 CLASSPATH 環境變數來併入 MQ base Java 程式碼及範例目錄。 表3.顯示不同平台的一般 CLASSPATH 設定。

*表 3. 產品的範例 CLASSPATH 陳述式*

| 平台 | 範例 CLASSPATH |
|---|---|
| AIX | CLASSPATH=*jdk_dir*/lib/classes.zip:<br>/usr/mqm/java/lib/com.ibm.mq.jar:<br>/usr/mqm/java/lib/connector.jar:<br>/usr/mqm/java/lib:<br>/usr/mqm/java/samples/base: |

*表 3. 產品的範例 CLASSPATH 陳述式 (繼續)*

| 平台 | 範例 CLASSPATH |
|---|---|
| HP-UX 和 Sun Solaris | CLASSPATH=*jdk_dir*/lib/classes.zip:<br>/opt/mqm/java/lib/com.ibm.mq.jar:<br>/opt/mqm/java/lib/connector.jar:<br>/opt/mqm/java/lib:<br>/opt/mqm/java/samples/base: |
| Windows 95、98、2000 和 NT | CLASSPATH=C:*jdk_dir*\lib\classes.zip;<br>*install_dir*\lib\com.ibm.mq.jar;<br>*install_dir*\lib\com.ibm.mq.iiop.jar;<br>*install_dir*\lib\connector.jar;<br>*install_dir*\lib\;<br>*install_dir*\samples\base\; |
| AS/400 | CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar:<br>/QIBM/ProdData/mqm/java/lib/connector.jar:<br>/QIBM/ProdData/mqm/java/lib:<br>/QIBM/ProdData/mqm/java/samples/base: |
| Linux | CLASSPATH=*jdk_dir*/lib/classes.zip:<br>*install_dir*/mqm/java/lib/com.ibm.mq.jar:<br>*install_dir*/mqm/java/lib/connector.jar:<br>*install_dir*/mqm/java/lib:<br>*install_dir*/mqm/java/samples/base: |
| **註:**<br>1. *jdk_dir* 是 JDK 的安裝目錄<br>2. *install_dir* 是產品的安裝目錄 | |

如果要使用 MQ JMS，您必須在類別路徑中併入其它 jar 檔。這些都列在第19頁的『後置安裝設定』中。

如果有現存的應用程式相依於已更換的連結套件 `com.ibm.mqbind`，您也必須新增 `com.ibm.mqbind.jar` 檔到您的類別路徑中。

如表4.所示，在部份平台中，您必須更新其它環境變數。

*表 4. 產品的環境變數*

| 平台 | 環境變數 |
|---|---|
| AIX | LD_LIBRARY_PATH=/usr/mqm/java/lib |
| HP_UX | SHLIB_PATH=/opt/mqm/java/lib |
| Sun Solaris | LD_LIBRARY_PATH=/opt/mqm/java/lib |
| Windows 95、98、2000 和 NT | PATH=*install_dir*\lib |
| **註:** *install_dir* 是產品的安裝目錄 | |

**註:**

1. 如果要在 OS/400 中使用 MQSeries Bindings for Java，請確定您的程式庫清單中有 QMQMJAVA 程式庫。

2. 確定您附加了 MQSeries 變數，且沒有改寫任何現存的系統環境變數。如果您改寫了現存的系統環境變數，在編譯或執行時期，應用程式可能會失效。

## Web 伺服器配置

如果您在 Web 伺服器中安裝 MQSeries Java，您可以在沒有於本端環境內安裝 MQSeries Java 的機器中，下載和執行 MQSeries Java 應用程式。如果要使 Web 伺服器能夠存取 MQSeries Java 檔，您必須設定您的 Web 伺服器配置來指向從屬站的安裝目錄。請參閱 Web 伺服器文件，以取得如何進行這項配置的詳細資料。

**註:** 在 OS/390 中，安裝的類別不支援從屬站連線，且無法很有效下載到從屬站中。不過，其它平台的 JAR 檔可以轉送到 OS/390 中，再提供給從屬站。

# 第3章 使用 MQSeries Class for Java (MQ base Java)

本章說明：

- 如何配置系統來執行範例 Applet 及應用程式，以驗證所安裝的 MQ base Java
- 如何修改執行自己的程式的程序

程序取決於要使用的連線選項。請遵循區段中符合您的需求的指示。

## 使用範例 Applet 驗證 TCP/IP 從屬站

MQ base Java 包括一個安裝驗證 Applet `mqjavac.html`。您可以利用這個 Applet 來驗證 TCP/IP 連接的 MQ base Java 從屬站模式。（另請參閱第15頁的『使用範例應用程式進行驗證』。）

Applet 會連接到給定的佇列管理程式，進行所有 MQSeries 呼叫，如果有任何失敗的話，會產生診斷訊息。

您可以利用 JDK 所提供的 Applet Viewer 來執行 Applet。 Applet Viewer 可以存取任何主電腦中的佇列管理程式。

不論任何情況，如果 Applet 沒有順利完成的話，請遵循診斷訊息中所提供的建議，再重新執行 Applet。

### 在 AS/400 中使用範例 Applet

OS/400 作業系統沒有原生圖形使用者介面 (GUI)。如果要執行範例 Applet，您必須在具有圖形能力的硬體中使用 Remote Abstract Window Toolkit for Java (AWT) 或 Class Broker for Java (CBJ)。您也可以從指令行驗證從屬站（請參閱第15頁的『使用範例應用程式進行驗證』）。

### 配置佇列管理程式接受從屬站連線

請利用下列程序來配置佇列管理程式接受從屬站所發出的送入連線要求。

#### TCP/IP 從屬站

1. 利用下列程序定義伺服器連線通道：

   **AS/400 以外的平台：**

   a. 利用 strmqm 指令啓動您的佇列管理程式。

   b. 輸入下列指令來啓動 runmqsc 程式：

      ```
      runmqsc
      ```

   c. 輸入下列指令來定義稱爲 JAVA.CHANNEL 的範例通道：

      ```
      DEF CHL('JAVA.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ') +
      DESCR('Sample channel for MQSeries Client for Java')
      ```

**AS/400 平台：**

a. 利用 STRMQM 指令啓動您的佇列管理程式。

b. 輸入下列指令來定義稱爲 JAVA.CHANNEL 的範例通道：

```
CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
MCAUSERID(SOMEUSERID) TEXT('Sample channel for MQSeries Client for Java')
```

其中 QMGRNAME 是佇列管理程式的名稱， SOMEUSERID 是有 MQSeries 資源之適當權限的 AS/400 使用者 ID。

2. 利用下列指令啓動一個接聽器程式：

**OS/2 和 NT 作業系統：**

發出下列指令：

```
runmqlsr -t tcp [-m QMNAME] -p 1414
```

**註:** 如果您使用預設佇列管理程式，您可以略過 -m 選項。

**在 Windows NT 作業系統中使用 VisiBroker for Java：**

利用下列指令啓動 IIOP (Internet Inter-ORB Protocol) 伺服器：

```
java com.ibm.mq.iiop.Server
```

**註:** 如果要停止 IIOP 伺服器，請發出下列指令：

```
java com.ibm.mq.iiop.samples.AdministrationApplet shutdown
```

**UNIX 作業系統：**

配置 inetd 常駐程式，使 inetd 啓動 MQSeries 通道。請參閱 *MQSeries 從屬站*，以取得如何執行這個動作的指示。

**OS/400 作業系統：**

發出下列指令：

```
STRMQMLSR MQMNAME(QMGRNAME)
```

其中 QMGRNAME 是佇列管理程式的名稱。

## 從 Applet Viewer 執行

如果要使用這個方法，您的機器必須安裝有 Java Developer's Kit (JDK)。

**本端安裝程序**

1. 切換至您的語言的範例目錄。
2. 輸入：

```
appletviewer mqjavac.html
```

**Web 伺服器安裝程序：**

請輸入指令：

```
appletviewer http://Web.server.host/MQJavaclient/mqjavac.html
```

**註:**

1. 在部份平台上，這個指令是 'applet'，不是 'appletviewer'。
2. 在某些平台中，您可能需要從畫面左上方的 'Applet' 功能表中選取「內容」，再將「網路存取」設爲「無限制」。

藉著使用這項技術，您應該可以連接到可利用 TCP/IP 來存取的任何主電腦其中的任何佇列管理程式。

## 自訂驗證 Applet

mqjavac.html 檔含有一些選用參數。這些參數可讓您修改 Applet 來配合您的需求。每個參數都是用 HTML 的一行來定義，外觀如下：

```
<!PARAM name="xxx" value="yyy">
```

如果要指定一個參數值，請移除開頭的驚歎號，再依需要來編輯值。您可以指定下列參數：

**hostname** 主電腦名稱編輯框內最初顯示的值。

**port** 埠號編輯框內最初顯示的值。

**channel** 通道編輯框內最初顯示的值。

**queueManager**
佇列管理程式編輯框內最初顯示的值。

**userID** 在連接到佇列管理程式時使用指定的使用者 ID。

**password** 在連接到佇列管理程式時使用指定的密碼。

**trace** 使 MQ base Java 寫入追蹤日誌。請只在 IBM 的服務指示下，才使用這個選項。

## 使用範例應用程式進行驗證

MQ base Java 提供有安裝驗證程式 MQIVP。您可以利用這個應用程式來測試 MQ base Java 的所有連線模式。這個程式會嘗試許多選項及其它資料來確定您要驗證的連線模式。請利用下列程序來進行安裝驗證：

1. 如果要測試從屬站連線：
   a. 依照第13頁的『配置佇列管理程式接受從屬站連線』所說明來配置您的佇列管理程式。
   b. 在從屬站機器中完成這個程序的其餘部份。

   如果要測試連結連線，請在 MQSeries 伺服器機器中完成這個程序的其餘部份。
2. 切換至範例目錄。
3. 輸入：

   ```
   java MQIVP
   ```

   這時程式會試圖：
   a. 連接具名佇列管理程式，然後切斷其連線。
   b. 開啓、放置、取得和關閉系統預設本端佇列。
   c. 如果作業順利完成的話，即傳回一則訊息。
4. 在提示 (1) 中，保留預設值 'MQSeries'。
5. 在提示 (2) 中：
   - 如果要使用 TCP/IP 連線，請輸入 MQSeries 伺服器主電腦名稱。
   - 如果要使用原生連線（連結模式），請保留欄位空白。（請勿輸入名稱。）

以下是您可能會見到的提示和回應範例。實際的提示和您的回應，會隨著您的 MQSeries 網路而不同。

```
Please enter the type of connection (MQSeries)              : (MQSeries)(1)
Please enter the IP address of the MQSeries server          : myhost(2)
Please enter the port to connect to                         : (1414)(3)
Please enter the server connection channel name             : JAVA.CHANNEL(3)
Please enter the queue manager name                         :
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager

Tests complete -
SUCCESS: This transport is functioning correctly.
Press Enter to continue...
```

**註:**

1. 如果您選擇伺服器連線，便不會出現標示 (3) 的提示。
2. 在 OS/390 中，您不會見到提示 (1)、(2) 或 (3)。
3. 在 OS/400 中，您只能從 Qshell 交談式介面來執行 `java MQIVP` 指令 （Qshell 是 OS/400, 5769-SS1 的選項 30）。另外，您也可以利用 CL 指令 `RUNJVA CLASS(MQIVP)` 來執行應用程式。
4. 如果要將 MQSeries 連結用在 OS/400 中的 Java，您必須確定 QMQMJAVA 程式庫在您的程式庫清單中。

### 使用 VisiBroker 連線

如果您使用 VisiBroker，便不需要第13頁的『配置佇列管理程式接受從屬站連線』中所說明的程序。

如果要測試使用 VisiBroker 的安裝結構，請使用第15頁的『使用範例應用程式進行驗證』中所說明的程序，但在提示 (1) 中輸入 `VisiBroker`，大小寫要完全相符。

### 使用 CICS Transaction Server for OS/390

1. 向 CICS 定義範例應用程式。
2. 定義一項異動來執行範例應用程式。
3. 將佇列管理程式名稱放入標準輸入所用的檔案中。
4. 執行異動。

程式輸出會放在標準和錯誤輸出所用的檔案中。

請參閱 CICS 文件，以取得執行 Java 程式及設定輸入和輸出檔的詳細資訊。

## 執行您自己的 MQ base Java 程式

如果要執行您自己的 Java Applet 或應用程式，請使用所說明的驗證程式的程序，請用您的應用程式名稱來取代 'mqjavac.html' 或 'MQIVP'。

如果需要撰寫 MQ base Java 應用程式及 Applet 的相關資訊，請參閱第43頁的『第2篇 MQ base Java 的程式設計』。

## 解決 MQ base Java 問題

如果有程式無法順利完成，請執行安裝驗證 Applet 或安裝驗證程式，再遵循診斷訊息中所提供的建議。請參閱第13頁的『第3章 使用 MQSeries Class for Java (MQ base Java)』，以取得這兩個程式的說明。

如果問題繼續存在，您需要洽詢 IBM 服務團隊，他們可能會要求您打開追蹤機能。執行這個動作的方法會隨著您是採用從屬站模式或連結模式而不同。請參閱下列各段，以取得系統的適當程序。

### 追蹤範例 Applet

如果要用範例 Applet 來執行追蹤，請編輯 mqjavac.html 檔。請找出下這行：

```
<!PARAM name="trace" value="1">
```

請移除問號，並根據所需要的詳細層次，將值 1 改成 1-5 中的一個數字。（數字越大，收集的資訊越多。）之後，這行應該是：

```
<PARAM name="trace" value="n">
```

其中 'n' 是 1 和 5 之間的一個數字。

追蹤輸出會出現在 Java 主控台或 Web 瀏覽器的 Java 日誌檔中。

### 追蹤範例應用程式

如果要追蹤 MQIVP 程式，請輸入下列字串：

```
java MQIVP -trace n
```

其中 'n' 是 1 和 5 之間的一個數字，隨著所需詳細層次而不同。（數字越大，收集的資訊越多。）

| 如果需要如何使用追蹤的詳細資訊，請參閱第68頁的『追蹤 MQ base Java 程式』。

### 使用 CICS Transaction Server for OS/390 進行追蹤

當您使用 CICS Transaction Server for OS/390 時，不可能將指令行引數直接提供給程式。您需要撰寫小的外層程式，以適當的引數來呼叫 MQIVP.main()。

## 錯誤訊息

這裡是您可能見到的一些常見錯誤訊息：

**無法識別本端主電腦 IP 位址**

伺服器沒有連接到網路中。

*建議動作*：將伺服器連接到網路中，再重試一次。

**無法載入 gatekeeper.ior 檔**

當 `gatekeeper.ior` 檔不在正確位置時，部署 VisiBroker Applet 的 Web 伺服器可能會出現這項失敗。

*建議動作*：請從 Applet 的部署目錄中，重新啓動 VisiBroker Gatekeeper。`gatekeeper` 檔會寫入這個目錄中。

**失敗：遺漏軟體，可能是 MQSeries，也可能是 VBROKER_ADM 變數**

如果 Java 軟體環境不完整的話，MQIVP 範例程式可能會出現這項失敗。

*建議動作*：在從屬站中，確定 VBROKER_ADM 環境變數已設爲 VisiBroker for Java 管理 (`adm`) 目錄的位址，再重試一次。

在伺服器中，確定已安裝了最新版的 MQ base Java，再重試一次。

**NO_IMPLEMENT**

發生包含 VisiBroker Smart Agent 在內的通信問題。

*建議動作*：請參閱 VisiBroker 文件。

**COMM_FAILURE**

發生包含 VisiBroker Smart Agent 在內的通信問題。

*建議動作*：所有 VisiBroker Smart Agent 都採用相同埠號，再重試一次。請參閱 VisiBroker 文件。

**MQRC_ADAPTER_NOT_AVAILABLE**

如果您試圖使用 VisiBroker 時出現這個錯誤，可能是在 CLASSPATH 中找不到 JAVA 類別 org.omg.CORBA.ORB。

*建議動作*：請確定您的 CLASSPATH 陳述式含有通往 VisiBroker `vbjorb.jar` 和 `vbjapp.jar` 檔的路徑。

**MQRC_ADAPTER_CONN_LOAD_ERROR**

如果在 OS/390 中執行時出現這個錯誤，請確定 STEPLIB 陳述式中有 MQSeries SCSQANLE 和 SCSQAUTH 資料集。

# 第4章 使用 MQSeries Class for Java 訊息服務 (MQ JMS)

本章說明下列作業：

- 如何設定您的系統來使用測試和範例程式
- 如何執行點對點「安裝驗證測試 (IVT)」程式來驗證您的 MQSeries Class for Java 訊息服務安裝
- 如何執行範例「發佈/訂閱安裝驗證測試 (PSIVT)」程式來驗證您的「發佈/訂閱」安裝
- 如何執行您自己的程式

## 後置安裝設定

如果要安裝 MQ JMS 程式所能使用的所有必要資源，您需要更新下列系統變數：

**類別路徑**

JMS 程式的順利作業需要 JVM 能夠使用若干 Java 套件。在您取得及安裝好必要的套件之後，必須在類別路徑中指定它們。

新增下列 .jar 檔到類別路徑中：

- com.ibm.mq.jar
- com.ibm.mqjms.jar
- connector.jar
- jms.jar
- jndi.jar
- jta.jar
- ldap.jar
- providerutil.jar

**環境變數**

在 MQ JMS 安裝的 `bin` 子目錄中有若干 Script。這些 Script 的用途是要作爲若干常用動作的捷徑。其中的許多 Script 假設已定義了環境變數 `MQ_JAVA_INSTALL_PATH`，且它指向 MQ JMS 安裝其中的目錄。您不需要設定這個變數，但如果您沒有設定它，您必須據此來編輯 `bin` 目錄中的 Script。

在 Windows NT 中，您可以利用**系統內容**的**環境**標籤來設定類別路徑和新環境變數。在 UNIX 中，這些通常是從使用者的登入 Script 中設定的。在任何平台中，您都可以利用 Script，爲不同專案維護不同的類別路徑及其它環境變數。

## 發佈/訂閱模式的其它設定

在可以使用 JMS 發佈/訂閱的 MQ JMS 實作之前，還有一些其它的必要設定：

### 確定分配管理程式在執行中

要確認 MQSeries 發佈/訂閱分配管理程式已經安裝且在執行中，請利用下列指令：

```
dspmqbrk -m MY.QUEUE.MANAGER
```

其中 MY.QUEUE.MANAGER 是分配管理程式執行其中的佇列管理程式。如果分配管理程式在執行中，會出現一則如下所示的訊息：

```
佇列管理程式 MY.QUEUE.MANAGER 的 MQSeries 訊息分配管理程式在執行中。
```

如果作業系統報告它無法執行 dspmqbrk 指令，請確定已正確安裝 MQSeries 發佈/訂閱分配管理程式。

如果作業系統報告分配管理程式不在作用中，請利用下列指令來啓動它：

```
strmqbrk -m MY.QUEUE.MANAGER
```

### 建立 MQ JMS 系統佇列

如果要讓 MQ JMS 發佈/訂閱實作正確運作，您必須建立若干系統佇列。 MQ JMS 安裝結構的 bin 子目錄中有一份 Script，可協助您進行這項作業。如果要使用這份 Script，請輸入下列指令：

```
runmqsc MY.QUEUE.MANAGER < MQJMS_PSQ.mqsc
```

如果發生錯誤，請檢查輸入的佇列管理程式名稱正不正確，以及佇列管理程式在不在執行中。

## 需要非專用使用者授權的佇列

非專用使用者需要 JMS 所用佇列的存取授權。如果需要在 MQSeries 中之存取控制的詳細資料，請參閱 *MQSeries System Administration* 中有關保護 MQSeries 物件的章節。

JMS 點對點模式的存取控制問題和 MQSeries Class for Java 的存取控制問題相類似：

- QueueSender 所用的佇列需要 put 權限。
- QueueReceiver 和 QueueBrowser 所用的佇列需要 get、inq 和 browse 權限。
- QueueSession.createTemporaryQueue 方法需要存取 QueueConnectionFactory temporaryModel 欄位中所定義的模型佇列（依預設，這會是 SYSTEM.DEFAULT.MODEL.QUEUE）。

對於 JMS 發佈/訂閱模式而言，會使用下列系統佇列：

SYSTEM.JMS.ADMIN.QUEUE

SYSTEM.JMS.REPORT.QUEUE

SYSTEM.JMS.MODEL.QUEUE

SYSTEM.JMS.PS.STATUS.QUEUE

SYSTEM.JMS.ND.SUBSCRIBER.QUEUE

SYSTEM.JMS.D.SUBSCRIBER.QUEUE

SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE

SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE

SYSTEM.BROKER.CONTROL.QUEUE

另外，任何發佈訊息的應用程式都需要存取使用的主題連線 Factory 中所指定的 STREAM 佇列。這一項的預設值是：

SYSTEM.BROKER.DEFAULT.STREAM

## 執行點對點 IVT

這一節說明 MQ JMS 所提供的點對點安裝驗證測試程式 (IVT)。

IVT 會試圖利用連結模式的 MQ JMS 來連接到本端機器串的預設佇列管理程式，以進行安裝驗證。之後，它會傳送一則訊息到 SYSTEM.DEFAULT.LOCAL.QUEUE 佇列中，再將它重新讀回。

您可以採用兩種可能的模式來執行程式。

**使用管理物件的 JNDI 查閱**

JNDI 模式會強迫程式從 JNDI 名稱空間中取得它的管理物件，這是 JMS 從屬站應用程式的預期作業。（請參閱第33頁的『管理 JMS 物件』，以取得管理物件的詳細資料。）這個呼叫方法和管理工具有相同的必備項目（請參閱第29頁的『第5章 使用 MQ JMS 管理工具』）。

**不使用管理物件的 JNDI 查閱**

如果您不要使用 JNDI，您可以採用非 JNDI 模式來執行 IVT，在執行時期建立各個管理物件。由於 JNDI 型儲存庫的設定比較複雜，因此，我們建議先執行 IVT，而不執行 JNDI。

### 不使用 JNDI 的點對點驗證

UNIX 中名稱爲 IVTRun 的 Script，或 Windows NT 中名稱爲 IVTRun.bat 的 Script，專供執行 IVT 之用。這個檔案安裝在安裝結構的 bin 子目錄中。

如果執行測試時不要使用 JNDI，請發出下列指令：

```
IVTRun -nojndi
```

在從屬站模式中，如果執行測試時不要使用 JNDI，請發出下列指令：

```
IVTRun -nojndi -client -m <qmgr> -host <hostname> [-port <port>]
               [-channel <channel>]
```

其中：

**qmgr** 是您要連接的佇列管理程式的名稱

**hostname** 是佇列管理程式執行其中的主電腦

**port** 是佇列管理程式接聽器所執行的 TCP/IP 埠（預設值是 1414）

**channel** 是從屬站連線通道（預設值 SYSTEM.DEF.SVRCONN）

如果測試順利完成，您應該會見到如下所示的輸出：

```
5648-C60 (c) Copyright IBM Corp. 1999. All Rights Reserved.
MQSeries Classes for Java(tm) Message Service - Installation Verification Test

Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message: Message Class:   jms_text          JMSType:          null
JMSDeliveryMode: 2                              JMSExpiration:    0
JMSPriority:     4                              JMSMessageID:     ID:414d5120716
d31202020202020202020203000c43713400000
JMSTimestamp:    935592657000                   JMSCorrelationID: null
JMSDestination:  queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo:      null
JMSRedelivered:  false
JMS_IBM_Format:MQSTR            JMS_IBM_PutApplType:11
JMSXGroupSeq:1                  JMSXDeliveryCount:0
JMS_IBM_MsgType:8               JMSXUserID:kingdon
JMSXApplID:D:\jdk1.1.8\bin\java.exe
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
```

## 使用 JNDI 的點對點驗證

如果要使用 JNDI 來執行 IVT，LDAP 伺服器必須在執行中，且必須配置來接受 Java 物件。如果出現下列訊息，它表示有 LDAP 伺服器連線，但目前沒有配置伺服器：

```
無法連結到物件
```

這個訊息表示伺服器沒有儲存 Java 物件，或物件許可權或字尾不正確。請參閱第347頁的『檢查 LDAP 伺服器配置』。

另外，下列管理物件必須能夠從 JNSI 名稱空間中擷取出來：

- MQQueueConnectionFactory
- MQQueue

UNIX 中名稱爲 `IVTSetup` 的 Script，或 Windows NT 中名稱爲 `IVTSetup.bat` 的 Script，專供自動建立這些物件之用。請輸入指令：

```
IVTSetup
```

Script 會呼叫 MQ JMS 管理工具（請參閱第29頁的『第5章 使用 MQ JMS 管理工具』），且會在 JNDI 名稱空間內建立物件。

MQQueueConnectionFactory 連結在 ivtQCF 名稱之下（LDAP 是 cn=ivtQCF）。所有內容都採用預設值：

```
TRANSPORT(BIND)
PORT(1414)
HOSTNAME(localhost)
CHANNEL(SYSTEM.DEF.SVRCONN)
VERSION(1)
CCSID(819)
TEMPMODEL(SYSTEM.DEFAULT.MODEL.QUEUE)
QMANAGER()
```

MQQueue 連結在 ivtQ 之下 (cn=ivtQ)。 QUEUE 內容值會成爲 QUEUE(SYSTEM.DEFAULT.LOCAL.QUEUE)。所有其它內容都有預設值：

```
PERSISTENCE(APP)
QUEUE(SYSTEM.DEFAULT.LOCAL.QUEUE)
EXPIRY(APP)
TARGCLIENT(JMS)
ENCODING(NATIVE)
VERSION(1)
CCSID(1208)
PRIORITY(APP)
QMANAGER()
```

在 JNDI 名稱空間內建立好管理物件之後，請利用下列指令來執行 IVTRun （Windows NT 是 IVTRun.bat）：

```
IVTRun [ -t ] [ -url <"providerURL"> [ -icf <initCtxFact> ] ]
```

其中：

**-t** 表示開啓追蹤（預設值是關閉追蹤）

**providerURL** 是管理物件的 JNDI 位置。如果預設起始環境定義 Factory 在使用中，則這是下列格式的 LDAP URL：

```
ldap://hostname.company.com/contextName
```

如果使用檔案系統服務提供者（請參閱下面的 initCtxFact），則 URL 採用下列格式：

```
file://directorySpec
```

**註:** 用引號 (″) 括住 *providerURL* 字串。

**initCtxFact** 是起始環境定義 Factory 的類別名稱。預設值是 LDAP 服務提供者，其值爲：

```
com.sun.jndi.ldap.LdapCtxFactory
```

如果使用檔案系統服務提供者，請將這個參數設爲：

```
com.sun.jndi.fscontext.RefFSContextFactory
```

如果測試順利完成，除了 'create' QueueConnectionFactory 和 Queue 行會指出從 JNDI 中擷取物件之外，輸出會和非 JNDI 輸出相類似。下列程式碼片段顯示一個範例。

```
5648-C60 (c) Copyright IBM Corp. 1999. All Rights Reserved.
MQSeries Classes for Java(tm) Message Service - Installation Verification Test

Using administered objects, please ensure that these are available

Retrieving a QueueConnectionFactory from JNDI
Creating a Connection
```

```
Creating a Session
Retrieving a Queue from JNDI
Creating a QueueSender
 ...
 ...
```

雖然不是嚴格必要，但您最好從 JNDI 名稱空間中移除 IVTSetup Script 所建立的物件。名稱爲 IVTTidy 的 Script（Windows NT 是 IVTTidy.bat）專供這個目的之用。

## IVT 錯誤復原

如果測試沒有順利完成，下列附註可能會有幫助：

- 如果需要含有類別路徑的任何錯誤訊息的協助，請依照第19頁的『後置安裝設定』中所說明來檢查類別路徑的設定正不正確。
- IVT 有可能會失敗，並出現「無法建立 MQQueueManager」的訊息，以及含有號碼 2059 的其它訊息。這表示 MQSeries 無法連接到 IVT 所執行機器其中的預設本端佇列管理程式。請檢查佇列管理程式在不在執行中，且它是否有標示爲預設佇列管理程式。
- 「無法開啓 MQ 佇列」的訊息表示 MQSeries 已連接到預設佇列管理程式，但無法開啓 'SYSTEM.DEFAULT.LOCAL.QUEUE'。這可能表示佇列不在預設佇列管理程式中，或沒有啓用佇列來執行 PUT 和 GET。請在測試期間，新增或啓用佇列。

表5.列出 IVT 所測試的類別及其來源套件：

*表 5. IVT 所測試的類別*

| 類別 | JAR 檔 |
|---|---|
| MQSeries JMS 類別 | com.ibm.mqjms.jar |
| com.ibm.mq.MQMessage | com.ibm.mq.jar |
| javax.jms.Message | jms.jar |
| javax.naming.InitialContext | jndi.jar |
| javax.resource.cci.Connection | connector.jar |
| javax.transaction.xa.XAException | jta.jar |
| com/sun/jndi/toolkit/ComponentDirContext | providerutil.jar |
| com.sun.jndi.ldap.LdapCtxFactory | ldap.jar |

# 發佈/訂閱安裝驗證測試

「發佈/訂閱安裝驗證測試 (PSIVT)」程式只提供有編譯格式。它在 com.ibm.mq.jms 套件中。

PSIVT 會試圖：

1. 建立一個發佈者 p，發出 MQJMS/PSIVT/資訊主題
2. 建立一個訂閱者 s，訂閱 MQJMS/PSIVT/資訊主題
3. 使用 p 來發佈簡單的文字訊息
4. 使用 s 來接收等在輸入佇列中的訊息

當您執行 PSIVT 時，發佈者會發佈訊息，訂閱者會接收和顯示訊息。發佈者會發佈到分配管理程式的預設串流。訂閱者不可延續，不會選擇訊息，會接受本端連線所提供的訊息。它會執行同步接收，等待訊息的到達最多等 5 秒鐘。

您可以如 IVT 一般，採用 JNDI 模式或獨立模式來執行 PSIVT。 JNDI 模式會利用 JNDI，從 JNDI 名稱空間中擷取 TopicConnectionFactory 和 Topic。如果沒有使用 JNDI 的話，會在執行時期建立這些物件。

## 不使用 JNDI 的發佈/訂閱驗證

名稱爲 PSIVTRun（Windows NT 是 PSIVTRun.bat) 的 'PSIVTRun' Script 專供執行 PSIVT。這個檔案在安裝結構的 bin 子目錄中。

如果執行測試時不要使用 JNDI，請發出下列指令：

```
PSIVTRun -nojndi [-m <qmgr>] [-t]
```

在從屬站模式中，如果執行測試時不要使用 JNDI，請發出下列指令：

```
PSIVTRun -nojndi -client -m <qmgr> -host <hostname> [-port <port>]
                [-channel <channel>] [-t]
```

其中：

**-nojndi** 表示沒有管理物件的 JNDI 查閱

**qmgr** 是您要連接的佇列管理程式的名稱

**hostname** 是佇列管理程式執行其中的主電腦

**port** 是佇列管理程式接聽器所執行的 TCP/IP 埠（預設值是 1414）

**channel** 是從屬站連線通道（預設值 SYSTEM.DEF.SVRCONN）

**-t** 表示開啓追蹤（預設值是關閉）

如果測試順利完成，應該會有如下輸出：

```
5648-C60 (c) Copyright IBM Corp. 1999. All Rights Reserved.
MQSeries Classes for Java(tm) Message Service
Publish/Subscribe Installation Verification Test
Creating a TopicConnectionFactory
Creating a Topic
Creating a Connection
Creating a Session
Creating a TopicPublisher
Creating a TopicSubscriber
Creating a TextMessage
Adding Text
Publishing the message to topic://MQJMS/PSIVT/Information
Waiting for a message to arrive...

Got message:

JMS Message class: jms_text
  JMSType:         null
  JMSDeliveryMode: 2
  JMSExpiration:   0
  JMSPriority:     4
  JMSMessageID:    ID:414d5120514d2e504f4c415249532e4254b7dc3753700000
  JMSTimestamp:    937232048000
  JMSCorrelationID:ID:414d515800000000000000000000000000000000000000000
  JMSDestination:  topic
  ://MQJMS/PSIVT/Information
```

```
  JMSReplyTo:      null
  JMSRedelivered:  false
  JMS_IBM_Format:MQSTR
  UNIQUE_CONNECTION_ID:937232047753
  JMS_IBM_PutApplType:26
  JMSXGroupSeq:1
  JMSXDeliveryCount:0
  JMS_IBM_MsgType:8
  JMSXUserID:hollingl
  JMSXApplID:QM.POLARIS.BROKER
A simple text message from the MQJMSPSIVT program

Reply string equals original string
Closing TopicSubscriber
Closing TopicPublisher
Closing Session
Closing Connection
PSIVT completed OK
PSIVT finished
```

## 使用 JNDI 的發佈/訂閱驗證

如果要採 JNDI 模式來執行 PSIVT，必須能夠從 JNSI 名稱空間中擷取出兩個管理物件：

- 連結在 ivtTCF 名稱下的 TopicConnectionFactory
- 連結在 ivtT 名稱下的 Topic

您可以利用「MQ JMS 管理工具」（請參閱第29頁的『第5章 使用 MQ JMS 管理工具』）及利用下列指令來定義這些物件：

```
DEFINE TCF(ivtTCF)
```

這個指令會定義 TopicConnectionFactory。

```
DEFINE T(ivtT) TOPIC(MQJMS/PSIVT/Information)
```

這個指令會定義 Topic。

這些定義假設可以使用分配管理程式執行其中的預設佇列管理程式。如果需要配置這些物件來使用非預設佇列管理程式的詳細資料，請參閱第33頁的『管理 JMS 物件』。這些物件應該常駐於以下說明的 -url 指令行參數所指向的環境定義。

如果要採 JNDI 模式來執行測試，請輸入下列指令：

```
PSIVTRun -url <purl> [-icf <initcf>] [-t]
```

其中：

**-t** 表示開啓追蹤（預設值是關閉追蹤）

**-url <purl>** 是管理物件常駐其中之 JNDI 位置的 URL。

**-icf <initcf>** 是 JNDI [com.sun.jndi.ldap.LdapCtxFactory] 的 initialContextFactory

如果測試順利完成，除了 'create' QueueConnectionFactory 和 Queue 行會指出從 JNDI 中擷取物件之外，輸出會和非 JNDI 輸出相類似。

## PSIVT 錯誤復原

如果測試沒有順利完成，下列附註可能會有幫助：

- 如果出現下列訊息：

```
*** 分配管理程式不在執行中！請利用 'strmqbrk' 來重新啓動它 ***
```

這表示分配管理程式已安裝在目標佇列管理程式中，但它的控制佇列含有部份未完成處理的訊息。這表示分配管理程式不在執行中。如果要啓動它，請利用 strmqbrk 指令。（請參閱第20頁的『發佈/訂閱模式的其它設定』。）

- 如果出現下列訊息：

  ```
  無法連接到佇列管理程式：<default>
  ```

  請確定您的 MQSeries 系統已配置了預設佇列管理程式。

- 如果出現下列訊息：

  ```
  無法連接到佇列管理程式...
  ```

  請確定 PSIVT 所用被管理的 TopicConnectionFactory 配置了有效的佇列管理程式名稱。另外，如果您使用 -nojndi 選項，請確定您提供了有效的佇列管理程式（使用 -m 選項）。

- 如果出現下列訊息：

  ```
  無法在佇列管理程式中存取分配管理程式控制佇列...
  請確定分配管理程式已安裝在這個佇列管理程式中
  ```

  請確定 PSIVT 所用被管理的 TopicConnectionFactory 配置了分配管理程式安裝其中的佇列管理程式名稱。如果您使用 -nojndi 選項，請確定您提供了佇列管理程式名稱（使用 -m 選項）。

## 執行您自己的 MQ JMS 程式

如果需要執行您自己的 MQ JMS 程式的相關資訊，請參閱第167頁的『第10章 撰寫MQ JMS 程式』。

MQ JMS 含有一個公用程式檔案 runjms（Windows NT 中的 runjms.bat），用以協助您執行提供的程式及您已撰寫的程式。

這個公用程式提供追蹤和日誌檔的預設位置，可讓您新增應用程式所需要的任何應用程式執行時期參數。提供的 Script 假設環境變數 MQ_JAVA_INSTALL_PATH 設爲 MQ JMS 安裝其中的目錄。這份 Script 也假設追蹤和日誌輸出分別使用這個目錄內的 trace 和 log 子目錄。這些只是建議採用的位置，您可以編輯 Script 來使用您選取的任何目錄。

請利用下列指令來執行您的應用程式：

```
runjms <classname of application> [application-specific arguments]
```

如果需要撰寫 MQ JMS 應用程式及 Applet 的相關資訊，請參閱第165頁的『第3篇 MQ JMS 的程式設計』。

## 解決問題

如果有程式無法順利完成，請執行第19頁的『第4章 使用 MQSeries Class for Java 訊息服務 (MQ JMS)』中所說明的安裝驗證程式，再遵循診斷訊息中所提供的建議。

## 執行 MQ JMS 追蹤

### 追蹤程式

提供 MQ JMS 追蹤機能是爲了要協助 IBM 工作人員診斷客戶的問題。

依預設，會停用追蹤，因爲輸出會快速增長，且在正常情況下沒有用。

當要求您提供追蹤輸出時，您可以將 Java 內容 MQJMS_TRACE_LEVEL 設爲下列其中一個值來啓用它：

**on** 只追蹤 MQ JMS 呼叫

**base** 追蹤 MQ JMS 呼叫和基礎 MQ base Java 呼叫

例如：

```
java -DMQJMS_TRACE_LEVEL=base MyJMSProg
```

如果要停用追蹤，請將 MQJMS_TRACE_LEVEL 設爲 **off**。

依預設，追蹤會輸出到現行工作目錄內名稱爲 mqjms.trc 的檔案中。您可以使用 Java 內容 MQJMS_TRACE_DIR，將它重新導向至不同的目錄。

例如：

```
java -DMQJMS_TRACE_LEVEL=base -DMQJMS_TRACE_DIR=/somepath/tracedir MyJMSProg
```

runjms 公用程式 Script 會依照下列方式，利用環境變數 MQJMS_TRACE_LEVEL 和 MQ_JAVA_INSTALL_PATH 來設定這些內容：

```
java -DMQJMS_LOG_DIR=%MQ_JAVA_INSTALL_PATH%\log
-DMQJMS_TRACE_DIR=%MQ_JAVA_INSTALL_PATH%\trace
-DMQJMS_TRACE_LEVEL=%MQJMS_TRACE_LEVEL% %1 %2 %3 %4 %5 %6 %7 %8 %9
```

這只是一項建議，您可以依需要來修改它。

### 日誌記載

提供 MQ JMS 日誌機能是爲了要報告嚴重問題，尤其是可能指出配置錯誤的嚴重問題，而不是程式設計錯誤。依預設，日誌輸出會傳送到 System.err 串流，它通常會出現在執行 JVM 之主控台的 stderr 中。

您可以利用指定新位置的 Java 內容，將輸出重新導向到檔案中，例如：

```
java -DMQJMS_LOG_DIR=/mydir/forlogs MyJMSProg
```

MQ JMS 安裝結構 bin 目錄中的公用程式 Script runjms 會將這個內容設爲：

```
<MQ_JAVA_INSTALL_PATH>/log
```

其中 MQ_JAVA_INSTALL_PATH 是通往 MQ JMS 安裝結構的路徑。這是一項建議，您可以修改它來配合需求。

當日誌重新導向到某檔案時，會採用二進位格式來輸出它。如果要檢視日誌，您可以利用所提供的 formatLog 公用程式（Windows NT 中的 formatLog.bat），它會將檔案轉換成純文字格式。這個公用程式儲存在 MQ JMS 安裝結構的 bin 目錄中。請依照下列方式來執行轉換：

```
formatLog <inputfile> <outputfile>
```

# 第5章 使用 MQ JMS 管理工具

管理者可以利用管理工具來定義 8 種類型的 MQ JMS 物件內容，以及將它們儲存在 JNDI 名稱空間內。之後，JMS 從屬站會利用 JNDI 從名稱空間中擷取這些被管理的物件來使用。

以下是您可以利用工具來管理的 JMS 物件：

- MQQueueConnectionFactory
- MQTopicConnectionFactory
- MQQueue
- MQTopic
- MQXAQueueConnectionFactory
- MQXATopicConnectionFactory
- JMSWrapXAQueueConnectionFactory
- JMSWrapXATopicConnectionFactory

如果需要這些物件的詳細資訊，請參閱第33頁的『管理 JMS 物件』。

**註:** JMSWrapXAQueueConnectionFactory 和 JMSWrapXATopicConnectionFactory 是 WebSphere 專用的類別。它們在 **com.ibm.ejs.jms.mq** 套件中。

管理者也可以利用這個工具來操作 JNDI 內的目錄名稱空間子環境定義。請參閱第32頁的『操作子環境定義』。

## 呼叫管理工具

管理工具有一個指令行介面。您可以採交談方式來使用它，或利用它來啓動批次程序。交談模式提供一個指令提示，讓您輸入管理指令。在批次模式中，啓動工具的指令包括含有管理指令 Script 之檔案的名稱。

如果要採交談模式來啓動工具，請輸入指令：

```
JMSAdmin [-t] [-v] [-cfg config_filename]
```

其中：

| | |
|---|---|
| **-t** | 啓用追蹤（預設值是關閉追蹤） |
| **-v** | 產生冗長輸出（預設值是精簡輸出） |
| **-cfg config_filename** | 替代配置檔的名稱（請參閱第30頁的『配置』） |

這時會出現指令提示，指出工具已準備好接受管理指令。開始時，這個提示會呈現爲：

```
InitCtx>
```

指出現行環境定義（也就是所有命名和目錄作業目前指向的 JNDI 環境定義）是 PROVIDER_URL 配置參數中所定義的起始環境定義（請參閱第30頁的『配置』）。

當您遍訪目錄名稱空間時，提示會反映變更，因而永遠顯示現行環境定義。

### 呼叫管理工具

如果要採批次模式來啓動工具，請輸入指令：

```
JMSAdmin <test.scp
```

其中 *test.scp* 是含有管理指令的 Script 檔（請參閱第32頁的『管理指令』）。檔案中的最後一個指令必須是 `END` 指令。

## 配置

您必須以下面這三個參數值來配置管理工具：

**INITIAL_CONTEXT_FACTORY**

這指出工具所用的服務提供者。這個內容目前有三個支援的值：

- com.sun.jndi.ldap.LdapCtxFactory (LDAP)
- com.sun.jndi.fscontext.RefFSContextFactory（檔案系統環境定義）
- com.ibm.ejs.ns.jndi.CNInitialContextFactory（使用 WebSphere 的 CosNaming 儲存庫）

**PROVIDER_URL**

這指出階段作業起始環境定義的 URL，也就是工具所完成的所有 JNDI 作業的根。這個內容有三種支援的格式：

- ldap://hostname/contextname (LDAP)
- file:[drive:]/pathname（檔案系統上下文）
- iiop://hostname[:port] /[?TargetContext=ctx]（存取「基本」 WebSphere CosNaming 名稱空間）

**SECURITY_AUTHENTICATION**

這指出 JNDI 是否傳遞安全憑證給您的服務提供者。只有在使用 LDAP 服務提供者時，才使用這個參數。這個內容目前可採用三個值：

- none（匿名鑑別）
- simple（簡式鑑別）
- CRAM-MD5（CRAM-MD5 鑑別機制）

如果沒有提供有效值，內容預設值爲「無」。請參閱第31頁的『安全』，以取得管理工具安全的詳細資料。

這些參數是在配置檔中設定的。當您呼叫工具時，您可以依照第29頁的『呼叫管理工具』中的說明，利用 `-cfg` 指令行參數來指定這個配置。如果您沒有指定配置檔名稱，工具會試圖載入預設配置檔 (`JMSAdmin.config`)。它會先在現行目錄中，再到 `<MQ_JAVA_INSTALL_PATH>/bin` 目錄中尋找這個檔案。（其中 <MQ_JAVA_INSTALL_PATH> 是通往 MQ JMS 安裝結構的路徑。）

配置檔是一組由 '=' 分隔的鍵/值配對組成的純文字檔。下列範例中可見到這種內容：

```
#設定服務供應站
    INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
#設定起始環境定義
    PROVIDER_URL=ldap://polaris/o=ibm_us,c=us
#設定鑑別類型
    SECURITY_AUTHENTICATION=none
```

（行開頭的 '#' 表示這行是註解或不用。）

安裝結構檢附有一個範例配置檔，稱爲 JMSAdmin.config，您可以在 <MQ_JAVA_INSTALL_PATH>/bin 目錄中找到它。請編輯這個檔案來配合您的系統設定。

## WebSphere 配置

如果要讓管理工具（或需要執行後續查閱的任何從屬站應用程式）使用 WebSphere 的 CosNaming 儲存庫，您需要有下列配置：

- CLASSPATH 必須包括 WebSphere 的 JNDI 相關 JAR 檔：
  - WebSphere 第 3.5 版：

    ```
    <WSAppserver>\lib\ejs.jar
    ```

- WebSphere 第 3.5 版的 PATH 必須包括：

  ```
  <WSAppserver>\jdk\jre\bin
  ```

  其中 <WSAppserver> 是 WebSphere 的安裝路徑。

## 安全

管理者必須瞭解第30頁的『配置』中所說明的 SECURITY_AUTHENTICATION 內容的作用。

- 如果這個參數設爲 *none* 的話，JNDI 不會傳遞任何安全憑證給服務提供者，且會執行「匿名鑑別」。
- 如果參數設爲 simple 或 CRAM-MD5 的話，安全憑證會透過 JNDI 而傳遞給基礎服務提供者。 這些安全憑證採用使用者識別名稱 (User DN) 和密碼的形式。

如果需要安全憑證的話，當工具起始設定時，會出現提示要求使用者提供這些內容。

**註:** 輸入的文字會回應在螢幕中，其中包括密碼。 因此，請小心避免未獲授權的使用者見到密碼。

工具本身不會進行任何鑑別；這項作業委派給 LDAP 伺服器來進行。 LDAP 伺服器管理者要負責設定和維護目錄不同部份的存取專用權。如果鑑別失敗，工具會顯示一則適當的錯誤訊息，並終止作業。

如果需要安全和 JNDI 的詳細資訊，請參閱 Sun 的 Java 網站 (http://java.sun.com) 所提供的文件。

## 管理指令

當出現指令提示時，表示工具已準備好，可以開始接受指令。管理指令通常會有下列形式：

```
verb [param]*
```

其中 *verb* 是表6.中所列出的管理動詞。所有有效指令都至少含有一個（且只有一個）動詞，它會出現在指令的開頭，不論標準格式或簡短格式都一樣。

不同的動詞，能採用的參數也不同。比方說，END 動詞不能有任何參數，但 DEFINE 動詞可以有 1 至 20 個參數。本章的稍後各節會討論至少有一個參數之動詞的詳細資料。

*表 6. 管理動詞*

| 動詞 | | 說明 |
|---|---|---|
| 標準 | 簡短 | |
| ALTER | ALT | 變更給定管理物件的至少一個內容 |
| DEFINE | DEF | 建立和儲存一個管理物件，或建立一個新的子環境定義 |
| DISPLAY | DIS | 顯示一或多個儲存的管理物件的內容或現行環境定義的內涵 |
| DELETE | DEL | 移除名稱空間內的一或多個物件，或移除空的子環境定義 |
| CHANGE | CHG | 變更現行環境定義，讓使用者能夠遍訪起始環境定義下的任何目錄名稱空間（擱置安全清除） |
| COPY | CP | 建立儲存的管理物件副本，用替代名稱來儲存它 |
| MOVE | MV | 改變用來儲存被管理物件的名稱 |
| END | | 關閉管理工具 |

動詞名稱不區分大小寫。

通常，您會按回車鍵來終止指令。不過，您也可以換一個方式，直接在回車之前輸入 '+' 號。這樣，您便可以如下列範例所示來輸入多行指令：

```
DEFINE Q(BookingsInputQueue) +
       QMGR(QM.POLARIS.TEST) +
       QUEUE(BOOKINGS.INPUT.QUEUE) +
       PORT(1415) +
       CCSID(437)
```

開頭爲 *、# 或 / 的行會被當作備註或忽略的行來處理。

## 操作子環境定義

您可以利用 CHANGE、DEFINE、DISPLAY 和 DELETE 等動詞來操作目錄名稱空間子環境定義。請參閱表7，以取得它們的用法。

*表 7. 用來操作子環境定義之指令的語法和說明*

| 指令語法 | 說明 |
|---|---|
| DEFINE CTX(ctxName) | 嘗試建立現行環境定義的新子環境定義，名稱爲 ctxName。如果安全違規、子環境定義已經存在或提供的名稱無效的話，這個動作便告失敗。 |
| DISPLAY CTX | 顯示現行環境定義的內涵。管理物件會附註 'a'，子環境定義會附註 '[D]'。每個物件的 Java 類型也會顯示出來。 |

*表 7. 用來操作子環境定義之指令的語法和說明 (繼續)*

| 指令語法 | 說明 |
|---|---|
| DELETE CTX(ctxName) | 嘗試刪除現行環境定義之名稱爲 ctxName 的子環境定義。如果找不到這個子環境定義、這個子環境定義不是空的，或安全違規，這個動作便告失敗。 |
| CHANGE CTX(ctxName) | 改變現行環境定義，使它能指向名稱爲 ctxName 的子環境定義。可能會提供下面這兩個 ctxName 特殊值的其中之一：<br>**=UP** 移到現行環境定義的母項<br>**=INIT** 直接移到起始環境定義<br><br>如果指定的環境定義不存在或安全違規，這個動作便告失敗。 |

## 管理 JMS 物件

這一節說明管理工具所能處理的 8 個物件類型。其中包括各物件類型之可配置內容的詳細資料，以及可操作它們的動詞。

### 物件類型

表8.顯示管理物件的 8 個類型。「關鍵字」直欄顯示可取代第34頁的表9.所顯示指令之 *TYPE* 的字串。

*表 8. 管理工具所處理的 JMS 物件類型。*

| 物件類型 | | 說明 |
|---|---|---|
| **Java** | **關鍵字** | |
| MQQueueConnectionFactory | QCF | JMS QueueConnectionFactory 介面的 MQSeries 實作。這代表在 JMS 的點對點領域中建立連線的 Factory 物件。 |
| MQTopicConnectionFactory | TCF | JMS TopicConnectionFactory 介面的 MQSeries 實作。這代表在 JMS 的發佈/訂閱領域中建立連線的 Factory 物件。 |
| MQQueue | Q | JMS Queue 介面的 MQSeries 實作。這代表在 JMS 的點對點領域中之訊息的目的地。 |
| MQTopic | T | JMS Topic 介面的 MQSeries 實作。這代表在 JMS 的發佈/訂閱領域中之訊息的目的地。 |
| MQXAQueueConnectionFactory[1] | XAQCF | JMS XAQueueConnectionFactory 介面的 MQSeries 實作。這代表在使用 XA 版 JMS 類別之 JMS 的點對點領域中建立連線的 Factory 物件。 |
| MQXATopicConnectionFactory[1] | XATCF | JMS XATopicConnectionFactory 介面的 MQSeries 實作。這代表在使用 XA 版 JMS 類別之 JMS 的發佈/訂閱領域中建立連線的 Factory 物件。 |

*表 8. 管理工具所處理的 JMS 物件類型。 (繼續)*

| 物件類型 | | 說明 |
|---|---|---|
| **Java** | **關鍵字** | |
| JMSWrapXAQueueConnectionFactory[2] | WSQCF | JMS QueueConnectionFactory 介面的 MQSeries 實作。這代表在搭配 WebSphere 使用 XA 版 JMS 類別之 JMS 的點對點領域中建立連線的 Factory 物件。 |
| JMSWrapXATopicConnectionFactory[2] | WSTCF | JMS TopicConnectionFactory 介面的 MQSeries 實作。這代表在搭配 WebSphere 使用 XA 版 JMS 類別之 JMS 的發佈/訂閱領域中建立連線的 Factory 物件。 |

1. 提供這些類別，是爲了給應用程式伺服器的供應商使用。 對應用程式設計師而言，它們可能沒有直接的用途。
2. 如果您的 JMS 階段作業要參與 WebSphere 所協調的整體異動， 請使用這個類型的 ConnectionFactory。

## JMS 物件使用的動詞

您可以利用 ALTER、DEFINE、DISPLAY、DELETE、COPY 和 MOVE 等動詞，在目錄名稱空間內操作管理物件。 表9.是它們的用法摘要。 請依第33頁的表8.所示，用代表所需管理物件的關鍵字來取代 *TYPE*。

*表 9. 用來操作管理物件之指令的語法和說明*

| 指令語法 | 說明 |
|---|---|
| ALTER *TYPE*(name) [property]* | 嘗試以提供的內容來更新給定管理物件的內容。如果安全違規、找不到指定的物件或提供的新內容無效，這個動作便告失敗。 |
| DEFINE *TYPE*(name) [property]* | 嘗試以提供的內容建立一個 *TYPE* 類型的管理物件，並試著利用 name 名稱將它儲存在現行環境定義下。如果安全違規、提供的名稱無效或已存在，或提供的內容無效的話，這個動作便告失敗。 |
| DISPLAY *TYPE*(name) | 顯示連結在現行環境定義 name 名稱下的 *TYPE* 類型之管理物件的內容。如果物件不存在或安全違規，這個動作便告失敗。 |
| DELETE *TYPE*(name) | 嘗試從現行環境定義中移除名稱爲 name、類型爲 *TYPE* 的管理物件。如果物件不存在或安全違規，這個動作便告失敗。 |
| COPY *TYPE*(nameA) *TYPE*(nameB) | 建立名稱爲 nameA、類型爲 *TYPE* 之管理物件的副本，將副本命名爲 nameB。這全部發生在現行環境定義的範圍內。如果要複製的物件不存在、已有名稱爲 nameB 的物件存在，或安全違規，這個動作便告失敗。 |
| MOVE *TYPE*(nameA) *TYPE*(nameB) | 將名稱爲 nameA、類型爲 *TYPE* 之管理物件移至（重新命名爲）nameB。這全部發生在現行環境定義的範圍內。如果要移動的物件不存在、已有名稱爲 nameB 的物件存在，或安全違規，這個動作便告失敗。 |

## 建立物件

物件是利用下列指令語法而建立和儲存在 JNDI 名稱空間內：

```
DEFINE TYPE(name) [property]*
```

也就是說，DEFINE 動詞，後面接著 *TYPE*(name) 管理物件參照，再接著零或多個內容（請參閱第36頁的『內容』）。

### LDAP 命名考量

如果要將您的物件儲存在 LDAP 環境中，它們的名稱必須符合某些慣例。其中一個是物件和子環境定義名稱必須含有字首，如 cn=（一般名稱）或 ou=（組織單位）。

管理工具簡化了 LDAP 服務提供者的使用方式，您不需要使用字首，便能參照物件和環境定義名稱。如果您沒有提供字首，工具會自動新增預設字首（目前爲 cn=）到您提供的名稱中。

下列範例中顯示這個情況：

```
InitCtx> DEFINE Q(testQueue)

InitCtx> DISPLAY CTX

    Contents of InitCtx

      a  cn=testQueue              com.ibm.mq.jms.MQQueue

     1 Object(s)
       0 Context(s)
       1 Binding(s), 1 Administered
```

請注意，雖然提供的物件名稱 (testQueue) 沒有字首，但工具會自動新增一個，以確保能夠符合 LDAP 命名慣例。同樣，提出 DISPLAY Q(testQueue) 指令也會新增這個字首。

您可能需要配置您的 LDAP 伺服器來儲存 Java 物件。請參閱第347頁的『附錄C. Java 物件的 LDAP 伺服器配置』，以取得協助您進行這項配置的資訊。

## 內容

內容由下列格式的名稱/值配對組成：

```
PROPERTY_NAME(property_value)
```

內容名稱不區分大小寫，以表10.所顯示的辨識名稱集爲限。這份表格也顯示每個內容的有效內容值。

*表 10. 內容名稱和有效值*

| 內容 | | 有效值（預設值為粗體） |
|---|---|---|
| **標準** | **簡短** | |
| DESCRIPTION | DESC | 任何字串 |
| TRANSPORT | TRAN | • **BIND** - 連線使用 MQSeries 連結。<br>• CLIENT - 使用從屬站連線。 |
| CLIENTID | CID | 任何字串 |
| QMANAGER | QMGR | 任何字串 |
| HOSTNAME | HOST | 任何字串 |
| PORT | | 任何正整數 |
| CHANNEL | CHAN | 任何字串 |
| CCSID | CCS | 任何正整數 |
| RECEXIT | RCX | 任何字串 |
| RECEXITINIT | RCXI | 任何字串 |
| SECEXIT | SCX | 任何字串 |
| SECEXITINIT | SCXI | 任何字串 |
| SENDEXIT | SDX | 任何字串 |
| SENDXITINIT | SDXI | 任何字串 |
| TEMPMODEL | TM | 任何字串 |
| MSGRETENTION | MRET | • **Yes** - 不要的訊息保留在輸入佇列中。<br>• No - 不要的訊息依其處理選項來加以處理。 |

*表 10. 內容名稱和有效值 (繼續)*

| 內容 | | 有效值（預設值為粗體） |
|---|---|---|
| **標準** | **簡短** | |
| BROKERVER | BVER | **V1** - 目前唯一容許使用的值。 |
| BROKERPUBQ | BPUB | 任何字串（預設值是 **SYSTEM.BROKER.DEFAULT.STREAM**） |
| BROKERSUBQ | BSUB | 任何字串（預設值是 **SYSTEM.JMS.ND.SUBSCRIPTION.QUEUE**） |
| BROKERDURSUBQ | BDSUB | 任何字串（預設值是 **SYSTEM.JMS.D.SUBSCRIPTION.QUEUE**） |
| BROKERCCSUBQ | CCSUB | 任何字串（預設值是 **SYSTEM.JMS.ND.CC.SUBSCRIPTION.QUEUE**） |
| BROKERCCDSUBQ | CCDSUB | 任何字串（預設值是 **SYSTEM.JMS.D.CC.SUBSCRIPTION.QUEUE**） |
| BROKERQMGR | BQM | 任何字串 |
| BROKERCONQ | BCON | 任何字串 |
| EXPIRY | EXP | • **APP** - 期限可由 JMS 應用程式來定義。<br>• UNLIM - 沒有期限。<br>• 任何代表期限的正整數（以毫秒爲單位）。 |
| PRIORITY | PRI | • **APP** - 優先順序可由 JMS 應用程式來定義。<br>• QDEF - 優先順序採用佇列預設值。<br>• 範圍在 0-9 之間的任何整數。 |
| PERSISTENCE | PER | • **APP** - 持續性可由 JMS 應用程式來定義。<br>• QDEF - 持續性採用佇列預設值。<br>• PERS - 訊息具有持續性。<br>• NON - 訊息不具持續性。 |
| TARGCLIENT | TC | • **JMS** - 訊息目標是 JMS 應用程式。<br>• MQ - 訊息目標是非 JMS 的傳統 MQSeries 應用程式。 |
| ENCODING | ENC | 請參閱第39頁的『ENCODING 內容』。 |
| QUEUE | QU | 任何字串 |
| TOPIC | TOP | 任何字串 |

許多內容都只關聯於物件類型的特定子集。 表11.顯示有效的內容/物件類型組合，並提供每個內容的簡要說明。

*表 11. 內容和物件類型的有效組合*

| 內容 | 有效物件類型 | | | | | | 說明 |
|---|---|---|---|---|---|---|---|
| | **QCF** | **TCF** | **Q** | **T** | **WSQCF XAQCF** | **WSTCF XATCF** | |
| DESCRIPTION | Y | Y | Y | Y | Y | Y | 所儲存物件的說明 |
| TRANSPORT | Y | Y | | | Y[1] | Y[1] | 連線要使用「MQ 連結」或從屬站連線 |
| CLIENTID | Y | Y | | | Y | Y | 從屬站的字串識別碼 |
| QMANAGER | Y | Y | Y | | Y | Y | 要連接的佇列管理程式名稱 |
| PORT | Y | Y | | | | | 佇列管理程式用來接聽的埠 |
| HOSTNAME | Y | Y | | | | | 佇列管理程式常駐其中之主電腦的名稱 |

*表 11. 內容和物件類型的有效組合 (繼續)*

| 內容 | 有效物件類型 | | | | | | 說明 |
|---|---|---|---|---|---|---|---|
| | QCF | TCF | Q | T | WSQCF XAQCF | WSTCF XATCF | |
| CHANNEL | Y | Y | | | | | 使用的從屬站連線通道的名稱 |
| CCSID | Y | Y | Y | Y | | | 連線要使用的編碼字集 ID |
| RECEXIT | Y | Y | | | | | 使用中之接收跳出程式的完整類別名稱 |
| RECEXITINIT | Y | Y | | | | | 接收跳出程式起始設定字串 |
| SECEXIT | Y | Y | | | | | 使用中之安全跳出程式的完整類別名稱 |
| SECEXITINIT | Y | Y | | | | | 安全跳出程式起始設定字串 |
| SENDEXIT | Y | Y | | | | | 使用中之傳送跳出程式的完整類別名稱 |
| SENDEXITINIT | Y | Y | | | | | 傳送跳出程式起始設定字串 |
| TEMPMODEL | Y | | | | Y | | 從中建立暫時佇列的模型佇名稱。 |
| MSGRETENTION | Y | | | | Y | | 連線消費者是否要在輸入佇列中保留不要的訊息 |
| BROKERVER | | Y | | | | Y | 所用分配管理程式的版本 |
| BROKERPUBQ | | Y | | | | Y | 分配管理程式輸入佇列（串流佇列）的名稱 |
| BROKERSUBQ | | Y | | | | Y | 從中擷取不可延續之訂閱訊息的佇列名稱 |
| BROKERDURSUBQ | | | | Y | | | 從中擷取可延續之訂閱訊息的佇列名稱 |
| BROKERCCSUBQ | | Y | | | | Y | 從中擷取 ConnectionConsumer 的不可延續訂閱訊息的佇列名稱 |
| BROKERCCDSUBQ | | | | Y | | | 從中擷取 ConnectionConsumer 的可延續訂閱訊息的佇列名稱 |
| BROKERQMGR | | Y | | | | Y | 分配管理程式執行其中的佇列管理程式 |
| BROKERCONQ | | Y | | | | Y | 分配管理程式的控制佇列名稱 |
| EXPIRY | | | Y | Y | | | 訊息在目的地上到期之前所經歷的時間 |
| PRIORITY | | | Y | Y | | | 傳送至目的地之訊息的優先順序 |
| PERSISTENCE | | | Y | Y | | | 傳送至目的地之訊息的持續性 |
| TARGCLIENT | | | Y | Y | | | 欄位指出是否要使用 MQSeries RFH2 格式與目標應用程式交換資訊 |
| ENCODING | | | Y | Y | | | 這個目的地所用的編碼架構 |
| QUEUE | | | Y | | | | 代表這個目的地之佇列的基礎名稱 |
| TOPIC | | | | Y | | | 代表這個目的地之主題的基礎名稱 |

**註:**

1. 對於 WSTCF、WSQCF、XATCF 和 XAQCF 物件而言，只能使用 BIND 傳輸類型。
2. 第343頁的『附錄A. 管理工具內容和可程式內容之間的對映』顯示工具與可程式內容所設定之內容的關係。
3. TARGCLIENT 內容指出是否要用 MQSeries RFH2 格式與目標應用程式交換資訊。

   MQJMS_CLIENT_JMS_COMPLIANT 常數指出要用 RFH2 格式來傳送訊息。 使用 MQ JMS 的應用程式能瞭解 RFH2 格式。 當您要與目標 MQ JMS 應用程式交換資訊時，您應該設定 MQJMS_CLIENT_JMS_COMPLIANT 常數。

MQJMS_CLIENT_NONJMS_MQ 常數指出不用 RFH2 格式來傳送訊息。 這個值通常會用於現有的 MQSeries 應用程式（也就是說，不處理 RFH2 的應用程式）。

## 內容相依關係

有些內容彼此之間會有相依關係。這可能表示，除非將另一內容設爲特定值，否則，提供某個內容並沒有意義。從屬站內容和跳出程式起始設定字串是可能有這種情況的兩個特定內容群組。

### 從屬站內容

如果連線 Factory 沒有明確設定 TRANSPORT(CLIENT) 內容，則 Factory 提供的連線所用的傳輸爲 MQ 連結。因而，這個連線 Factory 的任何從屬站連線都會無法配置。它們有：

- HOST
- PORT
- CHANNEL
- CCSID
- RECEXIT
- RECEXITINIT
- SECEXIT
- SECEXITINIT
- SENDEXIT
- SENDEXITINIT

如果您試圖設定其中的任何內容，而沒有將 TRANSPORT 內容設爲 CLIENT，這時會發生錯誤。

### 跳出程式起始設定字串

除非已提供了對應的跳出程式名稱，否則，設定任何跳出程式起始設定都無效。以下是跳出程式起始設定內容：

- RECEXITINIT
- SECEXITINIT
- SENDEXITINIT

例如，指定 RECEXITINIT(myString) 而沒有指定 RECEXIT(some.exit.classname) 會造成錯誤。

## ENCODING 內容

ENCODING 內容可採用的有效值比其餘內容複雜。編碼內容是從三個子內容中建構起來的：

| | |
|---|---|
| **整數編碼** | 這可能是一般編碼，也可能是反向編碼 |
| **十進位編碼** | 這可能是一般編碼，也可能是反向編碼 |
| **浮點編碼** | 這可能是 IEEE 一般編碼、IEEE 反向編碼，或 System/390® |

ENCODING 以下列語法的三字元字串來表示：

```
{N|R}{N|R}{N|R|3}
```

在這個字串中：

- N 表示一般編碼
- R 表示反向編碼
- 3 表示 System/390
- 第一個字元代表*整數編碼*
- 第二個字元代表*十進位編碼*
- 第三個字元代表*浮點編碼*

這爲 ENCODING 內容提供一組 12 個可能的值。

另外還有一個值，也就是 NATIVE 字串，它會設定 Java 平台所適用的編碼值。

下列範例顯示 ENCODING 的有效組合：

```
ENCODING(NNR)
ENCODING(NATIVE)
ENCODING(RR3)
```

## 範例錯誤狀況

這一節提供建立物件時所可能產生之錯誤狀況的範例。

### 內容不明

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PIZZA(ham and mushroom)
  無法建立有效物件，請檢查提供的參數
  不明類容：PIZZA
```

### 物件內容無效

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PRIORITY(4)
  無法建立有效物件，請檢查提供的參數
  無效的 QCF 內容：PRI
```

### 內容值的無效類型

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) CCSID(english)
  無法建立有效物件，請檢查提供的參數
  CCS 內容值無效：英文
```

### 內容值超出有效範圍

```
InitCtx/cn=Trash> DEFINE Q(testQ) PRIORITY(12)
  無法建立有效物件，請檢查提供的參數
  PRI 內容值無效：12
```

### 內容衝突 - 從屬站/連結

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) HOSTNAME(polaris.hursley.ibm.com)
  無法建立有效物件，請檢查提供的參數
  這個環境定義中的內容無效：從屬站/連結屬性衝突
```

### 內容衝突 - 跳出程式起始設定

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) SECEXITINIT(initStr)
  無法建立有效物件，請檢查提供的參數
  這個環境定義中的內容無效：提供的 ExitInit 字串
  沒有跳出程式字串
```

# 第2篇 MQ base Java 的程式設計

# 第6章 提供給程式設計師的簡介

本章含有提供給程式設計師的一般資訊。如果需要撰寫應用程式的詳細資料，請參閱第49頁的『第7章 撰寫 MQ base Java 程式』。

## 為什麼要使用 Java 介面？

如果您是 MQSeries 應用程式的開發者， MQSeries Class for Java 程式設計介面可爲您帶來許多 Java 的好處：

- Java 程式設計語言**很容易使用**。

  它不需要用到標題檔、指標、結構、聯集及運算子超載。 和對等的 C 及 C++ 相較，用 Java 來撰寫的程式比較容易開發和除錯。

- Java 是**物件導向**的。

  Java 的物件導向特性可以和 C++ 的物件導向特性相比較，但它沒有多重繼承。 相反地，Java 採用了介面的概念。

- Java 先天上就是**分散**的。

  Java 類別程式庫有一個用來應付 HTTP 和 FTP 這類 TCP/IP 通信協定的常式程式庫。 Java 程式可以如同存取檔案系統一般簡單來存取 URL。

- Java 是**強韌**的。

  Java 非常強調可能問題的早期檢查、動態（執行時期）檢查， 及排除有錯誤傾向的狀況。 Java 使用參照的概念，排除了改寫記憶體及毀損資料的可能性。

- Java 是**安全**的。

  Java 本來就是爲了在網路環或分散式環境中執行的， 有許多安全上的重點考量。 Java 程式無法超限執行它們的執行時期堆疊， 也無法在其程序空間之外毀損記憶體。 當 Java 程式從網際網路下載而來時， 它們甚至不能讀取或寫入本端檔案。

- Java 程式是**可攜**的。

  Java 規格沒有任何實作上的相依性。 Java 編譯器會產生架構中立的物件檔格式。 只要有 Java 執行時期系統，便可以在許多處理器上執行編譯的程式碼。

如果您利用 MQSeries Class for Java 來撰寫您的應用程式，使用者可以從網際網路中下載您的程式的 Java 二進位碼（稱爲 *Applet*）。使用者可以在他們自己的機器中執行這些 Applet。這表示有權存取您的 Web 伺服器的使用者可以載入及執行您的應用程式，而不需要事先將程式安裝在他們的機器中。

當程式需要更新時，您可以更新 Web 伺服器中的副本。下次使用者存取 Applet 時，他們會自動收到最新的版本。這可以大幅縮減安裝和更新傳統從屬站應用程式的成本；在傳統從屬站應用程式中會使用到許多桌面。

如果您將 Applet 放在從公司防火牆之外也能存取的 Web 伺服器中，這時網際網路中的任何人都可以下載和使用您的應用程式。這表示您可以從網際網路的任何位置上，將訊息放到您的 MQSeries 系統中。這便開啓了一扇大門，可讓您建置一組可從網際網路存取的全新服務、支援及電子商務。

## MQSeries Class for Java 介面

程序化的 MQSeries 應用程式設計介面是環繞著下列動詞來建置的：

```
MQBACK, MQBEGIN, MQCLOSE, MQCMIT, MQCONN, MQCONNX,
       MQDISC, MQGET, MQINQ, MQOPEN, MQPUT, MQPUT1, MQSET
```

這些動詞會以參數來取得它們所要操作的 MQSeries 物件的控點。由於 Java 是物件導向的，因此，Java 程式設計介面改變了這個情況。您的程式由一組 MQSeries 物件組成，您可以如下列範例所示，呼叫這些物件的方法來處理這些物件：

當您使用程序化介面時，切斷佇列管理程式連線要使用 MQDISC(Hconn, CompCode, Reason) 呼叫，其中 *Hconn* 是佇列管理程式的一個控點。

在 Java 介面中，佇列管理程式由 MQQueueManager 類別的物件來表示。在切斷佇列管理程式的連線時，您要呼叫這個類別的 disconnect() 方法。

```
// 宣告類型佇列管理程式的物件
MQQueueManager queueManager=new MQQueueManager();
...
// 執行某些動作...
...
// 切斷與佇列管理程式的連接
queueManager.disconnect();
```

## Java Development Kit

在編譯您撰寫的任何 Applet 或應用程式之前，您必須能夠存取您的開發平台所適用的 Java Development Kit (JDK)。 JDK 含有 MQSeries Class for Java 所相依的所有標準 Java 類別、變數、建構子和介面。它也含有在每個支援的平台中編譯和執行 Applet 及程式所需要的工具。

| 您可以在全球資訊網中，從 IBM Software Download Catalog 下載 JDK，網址如下：

```
http://www.ibm.com/software/download
```

您也可以利用 IBM VisualAge for Java 的整合開發環境所包含的 JDK 來開發應用程式。

如果要在 AS/400 平台中編譯 Java 應用程式，您必須先安裝：
- AS/400 Developer Kit for Java (5769-JV1)
- Qshell Interpreter OS/400 (5769-SS1) Option 30

## MQSeries Class for Java 類別程式庫

MQSeries Class for Java 是一組使 Java Applet 和應用程式能夠與 MQSeries 互動的 Java 類別。

以下是提供的類別：
- MQChannelDefinition
- MQChannelExit
- MQDistributionList
- MQDistributionListItem
- MQEnvironment
- MQException
- MQGetMessageOptions

- MQManagedObject
- MQMessage
- MQMessageTracker
- MQPoolServices
- MQPoolServicesEvent
- MQPoolToken
- MQPutMessageOptions
- MQProcess
- MQQueue
- MQQueueManager
- MQSimpleConnectionManager

以下是提供的 Java 介面：

- MQC
- MQPoolServicesEventListener
- MQReceiveExit
- MQSecurityExit
- MQSendExit

另外，也提供了下列 Java 介面的實作。不過，這些介面不是爲了要直接給應用程式使用：

- MQConnectionManager
- javax.resource.spi.ManagedConnection
- javax.resource.spi.ManagedConnectionFactory
- javax.resource.spi.ManagedConnectionMetaData

在 Java 中，*套件*是一種將相關類別集分成一組的機制。 MQSeries 類別和介面都放在稱爲 com.ibm.mq 的 Java 套件中。如果要將 MQSeries Class for Java 套件併入您的程式中，請新增下面這行到您的原始檔的頂端：

```
import com.ibm.mq.*;
```

# 第7章 撰寫 MQ base Java 程式

如果要利用 MQSeries Class for Java 來存取 MQSeries 佇列，您要撰寫 Java 程式，其中含有在 MQSeries 佇列中放入和取得訊息的呼叫。這些程式可採用 Java *Applet*、Java *Servlet* 或 Java *應用程式*的形式。

這一章提供的資訊可協助您撰寫 Java Applet、Servlet 及應用程式來與 MQSeries 系統互動。如果需要個別類別的詳細資料，請參閱第77頁的『第9章 MQ base Java 類別和介面』。

## 我應該撰寫 Applet 或應用程式？

您要撰寫 Applet、Servlet 或應用程式，取決於您要用什麼連線及要從哪裡執行程式。

Applet 和應用程式的主要差異如下：

- Applet 是利用 Applet Viewer 或在 Web 瀏覽器中執行，Servlet 是在 Web 應用程式伺服器中執行，應用程式則是獨立地執行。
- Applet 可以從 Web 伺服器下載到 Web 瀏覽器機器，但應用程式和 Servlet 不行。

以下是一般的適用規則：

- 如果您要從沒有在本端環境內安裝 MQSeries Class for Java 的機器中執行程式，您應該撰寫 Applet。
- MQSeries Class for Java 的原生連結模式不支援 Applet。因此，如果程式要用在包括原生連結模式的所有連線模式中，您必須撰寫 Servlet 或應用程式。

## 連線差異

MQSeries Class for Java 的程式設計方式與要使用的連線模式有些相依關係。

### 從屬站連線

當 MQSeries Class for Java 要作爲從屬站來使用時，它類似於 MQSeries C 從屬站，但有下列差異：

- 它只支援 TCP/IP。
- 它不支援連線表。
- 它不會在啓動時讀取任何 MQSeries 環境變數。
- 儲存在通道定義和環境變數中的資訊會儲存在 MQEnvironment 類別中。另外，這項資訊也可以在建立連線時，當作參數來傳遞。
- 錯誤和異常狀況會寫入 MQException 類別指定的日誌中。預設的錯誤目的地是 Java 主控台。

MQSeries Class for Java 從屬站不支援 MQBEGIN 動詞或快速連結。

如果需要 MQSeries 從屬站的一般資訊，請參閱 *MQSeries 從屬站*一書。

### 連線差異

**註:** 當您使用 VisiBroker 連線時，MQEnvironment 中的使用者 ID 和密碼設定不會轉遞到 MQSeries 伺服器。有效的使用者 ID 是 IIOP 伺服器所適用的使用者 ID。

### 連結模式

以下是 MQSeries Class for Java 連結模式不同於從屬站模式之處：

- 會忽略 MQEnvironment 類別所提供的大部份參數
- 連結支援 MQBEGIN 動詞和快速連結到 MQSeries 佇列管理程式中

**註:** MQSeries for AS/400 不支援利用 MQBEGIN 來起始佇列管理程式所協調的廣域工作單元。

### 定義要使用的連線

連線是藉由設定 MQEnvironment 類別中的變數來確定。

**MQEnvironment.properties**

這可包括下列鍵/值配對：

- 從屬站和連結連線：

```
MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES
```

- VisiBroker 連線：

```
MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_VISIBROKER
MQC.ORB_PROPERTY, orb
```

**MQEnvironment.hostname**

請依照下列方式設定這個變數的值：

- 如果是從屬站連線，請將這個變數設爲要連線的 MQSeries 伺服器的主電腦名稱
- 如果是連結模式，請將這個變數設爲 null

---

## 範例程式碼片段

這一節包含兩個範例程式碼片段；第51頁的圖1.和第54頁的圖2。每個都使用特定連線，且含有說明使用替代連線所需要之變更的附註。

### 範例 Applet 程式碼

下列程式碼片段示範利用 TCP/IP 連線來執行下列動作的 Applet：

1. 連接到佇列管理程式
2. 將訊息放入 SYSTEM.DEFAULT.LOCAL.QUEUE
3. 取回訊息

```
// ============================================================================
//
// Licensed Materials - Property of IBM
//
// 5639-C34
//
// (c) Copyright IBM Corp. 1995,1999
//
// ============================================================================
// MQSeries Client for Java 範例 Applet
//
// 這個範例使用 Applet Viewer 和 HTML 檔以 Applet 方法執行，
// 使用的指令：
//            appletviewer MQSample.html
// 輸出為指令行，而不是 Applet Viewer 視窗。
//
// 注意：如果您收到 MQSeries 錯誤 2 原因 2059，且您確定您的
// MQSeries 和 TCP/IP 設定是正確的，
// 則您應該按一下 Applet Viewer 視窗中的 "Applet" 選項
// 選取內容，並將「網路存取」變更為「沒有限制」。
import com.ibm.mq.*;            // 併入 MQSeries Class for Java 套件

public class MQSample extends java.applet.Applet
{

  private String hostname = "your_hostname";      // 定義您要連接的您的
                                                  // 主電腦名稱，
  private String channel  = "server_channel";     // 來定義要使用的
                                                  // 從屬站通道
                                                  // 注意；假設 MQSeries 伺服器
                                                  // 正在預設的 TCP/IP 的 1414 埠
                                                  // 上接聽
  private String qManager = "your_Q_manager";     // 定義要連接的
                                                  // 佇列管理程式
                                                  // 物件名稱。

  private MQQueueManager qMgr;                    // 定義佇列管理程式物件

  // 當呼叫類別時，會先執行這項起始設定作業。

  public void init()
  {
     // 設定 MQSeries 環境
     MQEnvironment.hostname = hostname;           // 將主電腦名稱和
                                                  // 通道字串直接
     MQEnvironment.channel  = channel;            // 放在這裡！

     MQEnvironment.properties.put(MQC.TRANSPORT_PROPERTY,//設定 TCP/IP 或伺服器
                                  MQC.TRANSPORT_MQSERIES);//連線

  } // 結束起始設定
```

*圖 1. MQSeries Class for Java 範例 Applet (1/3)*

## 範例程式碼

```
  public void start()
  {

    try {
      // 建立與佇列管理程式的連線
      qMgr = new MQQueueManager(qManager);

      // 設定我們要開啓的佇列之選項...
      // 注意：在 Java 中所有的 MQSeries 選項的字首都是 MQC。
       int openOptions = MQC.MQOO_INPUT_AS_Q_DEF |
                         MQC.MQOO_OUTPUT ;
      // 現在，指定我們要開啓的佇列，並開啓選項...

      MQQueue system_default_local_queue =
              qMgr.accessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                               openOptions);
|
      // 定義單一 MQSeries 訊息，並以 UTF 格式撰寫某些文字...

      MQMessage hello_world = new MQMessage();
      hello_world.writeUTF("Hello World!");

      // 指定訊息選項...

      MQPutMessageOptions pmo = new MQPutMessageOptions();  // 接受預設值，
                                                            // 如同
                                                            // MQPMO_DEFAULT
                                                            // 常數
      // 將訊息放在佇列上

      system_default_local_queue.put(hello_world,pmo);

      // 再將訊息取回...
      // 先定義 MQSeries 訊息緩衝區來接收訊息至...

      MQMessage retrievedMessage = new MQMessage();
      retrievedMessage.messageId = hello_world.messageId;

      // 設定取得訊息選項..

      MQGetMessageOptions gmo = new MQGetMessageOptions();  // 接受預設值
                                                            // 如同
                                                            // MQGMO_DEFAULT
      // 取得佇列訊息..

      system_default_local_queue.get(retrievedMessage, gmo);

      // 透過顯示 UTF 訊息文字來探測我們擁有的訊息

      String msgText = retrievedMessage.readUTF();
      System.out.println("The message is: " + msgText);

      // 關閉佇列

      system_default_local_queue.close();

      // 切斷與佇列管理程式的連線

      qMgr.disconnect();

    }

    // 如果以上發生錯誤，請試著找出錯誤所在。
    // 是 MQSeries 的錯誤嗎？
```

*圖 1. MQSeries Class for Java 範例 Applet (2/3)*

```
    catch (MQException ex)
    {
      System.out.println("An MQSeries error occurred : Completion code " +
                         ex.completionCode +
                         " Reason code " + ex.reasonCode);
    }
    // 是 Java 緩衝空間的錯誤嗎？
    catch (java.io.IOException ex)
    {
      System.out.println("An error occurred whilst writing to the
      message buffer: " + ex);
    }

  } // 結束啓動

} // 範例結尾
```

*圖 1. MQSeries Class for Java 範例 Applet (3/3)*

## 變更連線以使用 VisiBroker for Java

請將下面這行：

```
MQEnvironment.properties.put (MQC.TRANSPORT_PROPERTY,
                              MQC.TRANSPORT_MQSERIES);
```

改成：

```
MQEnvironment.properties.put (MQC.TRANSPORT_PROPERTY,
                              MQC.TRANSPORT_VISIBROKER);
```

並加入下面這幾行來起始設定 ORB（物件要求分配管理程式）：

```
ORB orb=ORB.init(this,null);
MQEnvironment.properties.put(MQC.ORB_PROPERTY,orb);
```

您也必須新增下列 import 陳述式到檔案的開頭：

```
import org.omg.CORBA.ORB;
```

如果您使用 VisiBroker 的話，您不需要指定埠號或通道。

## 範例應用程式碼

下列程式碼片段示範使用連結模式來執行下列動作的簡式應用程式：

1. 連接到佇列管理程式
2. 將訊息放入 SYSTEM.DEFAULT.LOCAL.QUEUE
3. 重新取回訊息

```
// ============================================================================
// Licensed Materials - Property of IBM
// 5639-C34
// (c) Copyright IBM Corp. 1995, 1999
// ============================================================================
// MQSeries Classes for Java 範例應用程式
//
// 這個範例使用 java MQSample 指令以 Java 應用程式方式執行

import com.ibm.mq.*;            // 併入 MQSeries classes for Java 套件
|
public class MQSample
{
  private String qManager = "your_Q_manager";  // 定義要連接的
                                               // 佇列管理程式名稱。
  private MQQueueManager qMgr;                 // 定義佇列管理程式
                                               // 物件
  public static void main(String args[]) {
     new MQSample();
  }

  public MQSample() {
   try {
|
      // 建立與佇列管理程式的連線

      qMgr = new MQQueueManager(qManager);

      // 設定我們要開啓的佇列之選項...
      // 注意：在 Java 中所有的 MQSeries 選項的字首都是 MQC。

      int openOptions = MQC.MQOO_INPUT_AS_Q_DEF |
                        MQC.MQOO_OUTPUT ;

      // 現在，指定我們要開啓的佇列，
      // 並開啓選項...

      MQQueue system_default_local_queue =
              qMgr.accessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                               openOptions);
|
      // 定義單一 MQSeries 訊息，並以 UTF 格式撰寫某些文字..

      MQMessage hello_world = new MQMessage();
      hello_world.writeUTF("Hello World!");

      // 指定訊息選項...

      MQPutMessageOptions pmo = new MQPutMessageOptions(); // 接受 // 預設值，
                                                           // 如同 MQPMO_DEFAULT
```

*圖 2. MQSeries Class for Java 範例應用程式 (1/2)*

```
      // 將訊息放在佇列上

       system_default_local_queue.put(hello_world,pmo);

       // 再將訊息取回...
       // 先定義 MQSeries 訊息緩衝區來接收訊息至...

       MQMessage retrievedMessage = new MQMessage();
       retrievedMessage.messageId = hello_world.messageId;

       // 設定取得訊息選項...

       MQGetMessageOptions gmo = new MQGetMessageOptions();  // 接受預設值
                                                             // 如同  MQGMO_DEFAULT
       // 取得佇列訊息...

       system_default_local_queue.get(retrievedMessage, gmo);

       // 透過顯示 UTF 訊息文字來探測我們擁有的訊息

       String msgText = retrievedMessage.readUTF();
       System.out.println("The message is: " + msgText);
       // 關閉佇列...
       system_default_local_queue.close();
       // 切斷與佇列管理程式的連線

       qMgr.disconnect();
     }
       // 如果以上發生錯誤，請試著找出錯誤所在
       // 是 MQSeries 的錯誤嗎？
     catch (MQException ex)
     {
       System.out.println("An MQSeries error occurred : Completion code " +
                          ex.completionCode + " Reason code " + ex.reasonCode);
     }
     // 是 Java 緩衝空間的錯誤嗎？
     catch (java.io.IOException ex)
     {
       System.out.println("An error occurred whilst writing to the message buffer: " + ex);
     }
   }
} // 範例結尾
```

*圖 2. MQSeries Class for Java 範例應用程式 (2/2)*

# 佇列管理程式作業

這一節說明如何利用 MQSeries Class for Java 來連接和切斷佇列管理程式。

## 設定 MQSeries 環境

**註:** 在連結模式中使用 MQSeries Class for Java 不需要這個步驟。這時請直接移到『連接佇列管理程式』。在利用從屬站連線來連接佇列管理程式之前，您必須先小心設定 MQEnvironment。

以 "C" 爲基礎的 MQSeries 從屬站依賴環境變數來控制 MQCONN 呼叫的行爲。由於 Java Applet 沒有環境變數的存取權，因此，Java 程式設計介面含有 MQEnvironment 類別。您可以利用這個類別來指定試圖建立連線時所用的下列詳細資料：

- 通道名稱
- 主電腦名稱
- 埠號
- 使用者 ID
- 密碼

如果要指定通道名稱和主電腦名稱，請使用下列程式碼：

```
MQEnvironment.hostname = "host.domain.com";
MQEnvironment.channel  = "java.client.channel";
```

這相當於下列項目的 MQSERVER 環境變數設定：

```
"java.client.channel/TCP/host.domain.com"
```

依預設，Java 從屬站會嘗試連接到埠 1414 的 MQSeries 接聽器。如果要指定不同的埠，請採用下列程式碼：

```
MQEnvironment.port = nnnn;
```

使用者 ID 和密碼預設爲空白。如果要指定不是空白的使用者 ID 或密碼，請使用下列程式碼：

```
MQEnvironment.userID   = "uid";  // 與 env var MQ_USER_ID 相等
MQEnvironment.password = "pwd";  // 與 env var MQ_PASSWORD 相等
```

**註:** 如果您利用 VisiBroker for Java 來建立連線，請參閱第53頁的『變更連線以使用 VisiBroker for Java』。

## 連接佇列管理程式

您已準備好，可以開始建立 MQQueueManager 類別的新案例來連接佇列管理程式：

```
MQQueueManager queueManager = new MQQueueManager("qMgrName");
```

如果要切斷佇列管理程式的連線，請呼叫佇列管理程式的 disconnect() 方法：

```
queueManager.disconnect();
```

如果您呼叫 disconnect 方法，所有通過這個佇列管理程式而存取的開放佇列和程序都會關閉。不過，用完這些資源之後確實關閉它們，是比較好的程式設計習慣。如果要做到這一點，請使用 close() 方法。

佇列管理程式的 commit() 和 backout() 方法會取代程序介面所用的 MQCMIT 和 MQBACK 呼叫。

## 存取佇列和程序

如果要存取佇列和程序，您可以使用 MQQueueManager 類別。 MQOD（物件描述子結構）會收合到這些方法的參數中。比方說，如果要開啓 ″queueManager″ 佇列管理程式中的佇列，請使用下列程式碼：

```
MQQueue queue = queueManager.accessQueue("qName",
                                         MQC.MQOO_OUTPUT,
                                         "qMgrName",
                                         "dynamicQName",
                                         "altUserId");
```

*選項*參數和 MQOPEN 呼叫中的「選項」參數相同。

accessQueue 方法會傳回 MQQueue 類別的新物件。

用完佇列之後，請如下列範例所示，利用 close() 方法來關閉它：

```
queue.close();
```

您也可以利用 MQQueue 建構子，透過 MQSeries Class for Java 來建立佇列。除了加上一個佇列管理程式參數之外，參數完全和 accessQueue 方法相同。例如：

```
MQQueue queue = new MQQueue(queueManager,
                            "qName",
                            MQC.MQOO_OUTPUT,
                            "qMgrName",
                            "dynamicQName",
                            "altUserId");
```

用這個方法來建構佇列物件，您便可以撰寫自己的 MQQueue 超類別。

如果要存取程序，請利用 accessProcess 方法來取代 accessQueue。這個方法不會有*動態佇列名稱*參數，因爲這不適用於程序。

accessProcess 方法會傳回 MQProcess 類別的新物件。

用完程序物件之後，請如下列範例所示，利用 close() 方法來關閉它：

```
process.close();
```

您也可以利用 MQProcess 建構子，透過 MQSeries Class for Java 來建立程序。除了加上一個佇列管理程式參數之外，參數完全和 accessProcess 方法相同。用這個方法來建構程序物件，您便可以撰寫自己的 MQProcess 超類別。

## 處理訊息

您利用 MQQueue 類別的 put() 方法將訊息放入佇列中。您利用 MQQueue 類別的 get() 方法取出佇列中的訊息。這不同於程序介面，程序介面中的 MQPUT 和 MQGET 會放入和取出位元組陣列， Java 程式設計語言則會放入和取出 MQMessage 類別的案例。MQMessage 類別封裝了含有實際訊息資料的資料緩衝區及說明這項訊息的所有 MQMD（訊息描述子）參數。

如果要建置新訊息，請建立一個新的 MQMessage 類別案例，再利用 writeXXX 方法將資料放入訊息緩衝區中。

## 訊息處理

當建立新的訊息案例時，所有 MQMD 參數都會如 *MQSeries Application Programming Reference* 所定義，自動設爲預設值。 MQQueue 案例的 put() 方法也會以 MQPutMessageOptions 類別的案例爲參數。這個類別代表 MQPMO 結構。下列範例會建立一則訊息，並將它放入佇列中：

```
// 建立包含在 my 名稱之後的 my 年齡的新訊息
MQMessage myMessage = new MQMessage();
myMessage.writeInt(25);

String name = "Wendy Ling";
myMessage.writeInt(name.length());
myMessage.writeBytes(name);

// 使用預設放置訊息選項...
MQPutMessageOptions pmo = new MQPutMessageOptions();

// 放置訊息！
queue.put(myMessage,pmo);
```

MQQueue 的 get() 方法會傳回代表剛從佇列中取出之訊息的新 MQMessage 案例。它也會以 MQGetMessageOptions 類別的案例爲參數。這個類別代表 MQGMO 結構。

您不需要指定訊息大小上限，因爲 get() 方法會自動調整內部緩衝區的大小來配合內收的訊息。請利用 MQMessage 類別的 readXXX 方法來存取必要訊息中的資料。

下列範例顯示如何從佇列中取得訊息：

```
// 取得佇列的訊息
MQMessage theMessage    = new MQMessage();
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.get(theMessage,gmo);  // 有預設值

// 取出訊息資料
int age = theMessage.readInt();
int strLen = theMessage.readInt();
byte[] strData = new byte[strLen];
theMessage.readFully(strData,0,strLen);
String name = new String(strData,0);
```

您可以設定 *encoding* 成員變數來改變 read 和 write 方法所用的數字格式。

您可以設定 *characterSet* 成員變數來改變讀取和寫入字串所用的字集。

請參閱第100頁的『MQMessage』，以取得詳細資料。

**註:** MQMessage 的 writeUTF() 方法會自動爲字串長度及所包含的 Unicode 位元組編碼。當其它 Java 程式讀取您的訊息（利用 readUTF()）時，這是傳送字串資訊最簡單的方法。

## 處理錯誤

Java 介面中的方法不會傳回完成碼和原因碼。相反地，每當 MQSeries 呼叫所產生的完成碼和原因碼沒有全爲零時，它們會擲出異常狀況。這簡化了程式的邏輯，不必在每次呼叫 MQSeries 之後，都要檢查回覆碼。您可以決定要在程式的哪些點上處理可能發生的失敗。在這些點上，您可以如下列範例所示，用 'try' 和 'catch' 區塊來括住程式碼：

```
try {
myQueue.put(messageA,putMessageOptionsA);
myQueue.put(messageB,putMessageOptionsB);
```

```
}
catch (MQException ex) {
// 這塊程式碼只在下列其中一種情況下才執行：
// 兩個放置方法提供 rise 給非零
// 完成碼或原因碼。
System.out.println("An error occurred during the put operation:" +
                           "CC = " + ex.completionCode +
                           "RC = " + ex.reasonCode);
}
```

## 取得和設定屬性值

對於許多常用的屬性，MQManagedObject、MQQueue、MQProcess 和 MQQueueManager 等類別都含有 getXXX() 和 setXXX() 方法。您可以利用這些方法來取得和設定它們的屬性值。請注意，在 MQQueue 中，當您開啓佇列時，必須指定適當的 'inquire' 和 'set' 旗號，這些方法才能運作。

對於不常用的屬性，MQQueueManager、MQQueue 和 MQProcess 類別都會繼承 MQManagedObject 類別。這個類別定義 inquire() 和 set() 介面。

當您利用 *new* 運算子來建立新的佇列管理程式物件時，會自動就 'inquiry' 來開啓它。當您使用 accessProcess() 方法來存取程序物件時，會自動就 'inquiry' 來開啓這個物件。當您使用 accessQueue() 方法來存取佇列物件時，不會自動就 'inquiry' 或 'set' 作業來開啓這個物件。因爲自動加入這些選項，有可能使遠端佇列的某些類型發生問題。如果要就佇列來使用 inquire、set、getXXX 和 setXXX 等方法，您必須在 accessQueue() 方法的 openOptions 參數中指定適當的 'inquire' 和 'set' 旗號。

inquire 和 set 方法有三個參數：

- selectors 陣列
- intAttrs 陣列
- charAttrs 陣列

您不需要在 MQINQ 中找到的 SelectorCount、IntAttrCount 和 CharAttrLength 參數，因爲 Java 中的陣列長度是已知的。下列範例顯示如何進行佇列查詢：

```
// 查詢佇列
final static int MQIA_DEF_PRIORITY = 6;
final static int MQCA_Q_DESC = 2013;
final static int MQ_Q_DESC_LENGTH = 64;

int[] selectors  = new int[2];
int[] intAttrs   = new int[1];
byte[] charAttrs = new byte[MQ_Q_DESC_LENGTH]

selectors[0] = MQIA_DEF_PRIORITY;
selectors[1] = MQCA_Q_DESC;

queue.inquire(selectors,intAttrs,charAttrs);

System.out.println("Default Priority = " + intAttrs[0]);
System.out.println("Description : " + new String(charAttrs,0));
```

## 多重執行緒程式

在 Java 中很難避免多重執行緒的程式。請考慮一個連接到佇列管理程式且在啓動時開啓佇列的簡單程式。這個程式會在螢幕中顯示單一按鈕。當使用者按下這個按鈕時，程式會從佇列中提取一則訊息。

## 多重執行緒程式

Java 執行環境先天上就是多重執行緒的。因此，應用程式起始設定會採用一個執行緒，回應按下按鈕的程式碼會採用另一個執行緒來執行（使用者介面執行緒）。

當使用以 C 語言爲基礎的 MQSeries 從屬站時，這會造成問題，因爲控點不能由多個執行緒來共用。 MQSeries Class for Java 放寬了這項限制，它讓佇列管理程式物件（及其相關佇列和程序物件）能夠由多個執行緒來共用。

實作 MQSeries Class for Java 可確保在給定的連線 （MQQueueManager 物件案例）上，對於目標 MQSeries 佇列管理程式的所有存取活動都會同步化。因此，在這個連線所有進行中的呼叫都完成之前，無法執行要呼叫佇列管理程式的執行緒。 如果您需要從程式內的多重執行緒同時存取相同的佇列管理程式，請爲需要同時存取的每個執行緒各建立一個新的 MQQueueManager 物件。（這相當於爲每個執行緒發出個別的 MQCONN 呼叫。）

**註:** 在 CICS Transaction Server for OS/390 環境中，只有主要（第一個）執行緒可以發出 CICS 或 MQSeries 呼叫。因此，不可能在這個環境中，讓執行緒共用 MQQueueManager 或 MQQueue，也不可能就子項執行緒建立新的 MQQueueManager。

## 撰寫使用者跳出程式

MQSeries Class for Java 可讓您提供自己的傳送、接收和安全跳出程式。

如果要實作跳出程式，您要定義一個新的 Java 類別來實作適當的介面。 MQSeries 套件中定義了三個跳出程式介面：

- MQSendExit
- MQReceiveExit
- MQSecurityExit

下列範例定義完整實作了三個介面的類別：

```
class MyMQExits implements MQSendExit, MQReceiveExit, MQSecurityExit {

  // 這個方法來自傳送跳出程式
  public byte[] sendExit(MQChannelExit channelExitParms,
                         MQChannelDefinition channelDefParms,
                         byte agentBuffer[])
  {
     // 在這裡填入傳送跳出程式的主體
  }


  // 這個方法來自接收跳出程式
  public byte[] receiveExit(MQChannelExit channelExitParms,
                            MQChannelDefinition channelDefParms,
                            byte agentBuffer[])
  {
     // 在這裡填入接收跳出程式的主體
  }


  // 這個方法來自安全跳出程式
  public byte[] securityExit(MQChannelExit channelExitParms,
                             MQChannelDefinition channelDefParms,
                             byte agentBuffer[])
  {
     // 在這裡填入安全跳出程式的主體
  }

}
```

每個跳出程式都會收到一個 MQChannelExit 和一個 MQChannelDefinition 物件案例。這些物件代表程序介面中所定義的 MQCXP 和 MQCD 結構。

對傳送跳出程式，*agentBuffer* 參數含有即將傳送的資料。對於接收跳出程式或安全跳出程式，*agentBuffer* 參數含有剛收到的資料。您不需要長度參數，因爲 agentBuffer.length 表示式會指出陣列長度。

對於傳送和安全跳出程式，您的跳出程式碼應該傳回要傳給伺服器的位元組陣列。對於接收跳出程式，您的跳出程式碼應該傳回 MQSeries Class for Java 所要解譯的修改過的資料。

以下是最簡單的跳出程式主體：

```
{
  return agentBuffer;
}
```

### 撰寫使用者跳出程式

如果您的程式要採下載 Java Applet 的方式來執行，適用的安全限制會使您無法讀取或寫入任何本端檔案。如果您的跳出程式需要配置檔，您可以將檔案放在 Web 中，再利用 java.net.URL 類別下載它及檢查它的內容。

## 連線儲存池

MQSeries Class for Java 第 5.2 版爲處理多重 MQSeries 佇列管理程式連線的應用程式提供了其它支援。當不再需要某個連線時，不會將連線毀損，而會將它放入儲存池中，供稍後使用。這可以實質改進會序列連接任意佇列管理程式的應用程式和中介軟體的效能。

MQSeries 提供一個預設的連線儲存池。應用程式可以利用 MQEnvironment 類別來登錄或取消登錄記號，以啓動這個連線儲存池或予以取消啓動。如果儲存池在作用中，當 MQ base Java 建構 MQQueueManager 物件時，它會搜尋這個預設儲存池，及重複使用任何適用的連線。當發生 MQQueueManager.disconnect() 呼叫時，會將基礎連線傳回儲存池中。

另外，應用程式也可以爲特定用途而建構一個 MQSimpleConnectionManager 連線儲存池。之後，應用程式可以在建構 MQQueueManager 物件時指定這個儲存池，也可以將這個儲存池傳遞給 MQQueueManager，以作爲預設連線儲存池。

另外，MQ base Java 也提供了 Java 2 Platform Enterprise Edition (J2EE) Connector Architecture 的局部實作。在含 JAAS 1.0（Java 鑑別和授權服務）的 Java 2 第 1.3 版 JVM 之下執行的應用程式可以實作 **javax.resource.spi.ConnectionManager** 介面來提供自己的連線儲存池。同樣，這個介面可由 MQQueueManager 建構子來指定，也可以指定成預設連線儲存池。

### 控制預設連線儲存池

請思考下面這個範例應用程式，MQApp1：

```
import com.ibm.mq.*;
public class MQApp1
{
      public static void main(String[] args) throws MQException
      {
         for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            :
            : (使用 qmgr 執行某些動作)
            :
            qmgr.disconnect();
         }
      }
}
```

MQApp1 會從指令行取出一份本端佇列管理程式清單，依次連接到各個本端佇列管理程式，並執行某些作業。不過，當指令行多次列出相同的佇列管理程式時，只連接一次、但重複使用連線多次，會比較有效。

MQ base Java 提供一個預設連線儲存池，您可以利用它來做到這一點。如果要啓用儲存池，請利用 MQEnvironment.addConnectionPoolToken() 方法中的其中一個。如果要停用儲存池，請利用 MQEnvironment.removeConnectionPoolToken()。

下面這個範例應用程式 MQApp2 的功能相當於 MQApp1，但只連接到每個佇列管理程式一次。

```
import com.ibm.mq.*;
public class MQApp2
{
      public static void main(String[] args) throws MQException
      {
         MQPoolToken token=MQEnvironment.addConnectionPoolToken();

         for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            :
            : (使用 qmgr 執行某些動作)
            :
            qmgr.disconnect();
         }

         MQEnvironment.removeConnectionPoolToken(token);

      }
}
```

第一粗體行會在 MQEnvironment 中登錄一個 MQPoolToken 物件來啓動預設的連線儲存池。

現在，MQQueueManager 建構子會搜尋這個儲存池來找出適當的連線，如果找不到現存連線的話，會建立一個連線來連接佇列管理程式。 qmgr.disconnect() 呼叫會將連線傳回儲存池，供稍後重複使用。這些 API 呼叫和範例應用程式 MQApp1 相同。

第二個強調顯示的行會停用預設的連線儲存池，它會毀損儲存池所儲存的任何佇列管理程式連線。這非常重要，因爲沒有這樣的話，應用程式會因儲存池中多個存活著的佇列管理程式連線而終止。這個情況可能會造成出現在佇列管理程式日誌中的錯誤。

預設連線儲存池會儲存最多 10 個沒有使用的連線，沒有使用的連線最多維持 5 分鐘作用中的狀態。應用程式可加以改變（如果需要詳細資料，請參閱第65頁的『提供不同的連線儲存池』）。

應用程式不必利用 MQEnvironment 來提供 MQPoolToken，它可以建構自己的 MQPoolToken：

```
MQPoolToken token=new MQPoolToken();
MQEnvironment.addConnectionPoolToken(token);
```

有些應用程式或中介軟體供應商可能會提供 MQPoolToken 的子類別，以將資訊傳遞到自訂連線儲存池中。它們可以利用這個方式來建構並傳遞至 addConnectionPoolToken() 中，因而可以將額外的資訊傳遞到連線儲存池。

## 預設連線儲存池和多重元件

MQEnvironment 保留有一個靜態的登錄 MQPoolToken 物件集。如果要在這個集中新增或移除 MQPoolToken，請使用下列方法：

- MQEnvironment.addConnectionPoolToken()
- MQEnvironment.removeConnectionPoolToken()

應用程式可由許多獨立存在的元件組成，以及利用佇列管理程式來執行工作。在這種應用程式中，每個元件都應新增一個 MQPoolToken 到 MQEnvironment 集中來代表它的存活時間。

比方說，範例應用程式 MQApp3 會建立 10 個執行緒，並啓動其中每一個執行緒。每個執行緒都會登錄它自己的 MQPoolToken，等待一段時間，再連接到佇列管理程式。等執行緒切斷連接之後，它會移除它自己的 MQPoolToken。

當 MQPoolToken 集中至少有一個記號時，預設連線儲存池會保留在作用中，以便在這個應用程式的持續期間，它能維持作用中的狀態。這個應用程式不需要在執行緒的整體控制中保留主要物件。

```
import com.ibm.mq.*;
public class MQApp3
{
      public static void main(String[] args)
      {
         for (int i=0; i<10; i++) {
            MQApp3_Thread thread=new MQApp3_Thread(i*60000);
            thread.start();
         }
      }
}

class MQApp3_Thread extends Thread
{
      long time;

      public MQApp3_Thread(long time)
      {
         this.time=time;
      }

      public synchronized void run()
      {
         MQPoolToken token=MQEnvironment.addConnectionPoolToken();
         try {
            wait(time);
            MQQueueManager qmgr=new MQQueueManager("my.qmgr.1");
            :
            : (使用 qmgr 執行某些動作)
            :
            qmgr.disconnect();
         }
         catch (MQException mqe) {System.err.println("Error occurred!");}
         catch (InterruptedException ie) {}

         MQEnvironment.removeConnectionPoolToken(token);
      }
}
```

## 提供不同的連線儲存池

這一節說明如何使用 **com.ibm.mq.MQSimpleConnectionManager** 類別來提供不同的連線儲存池。這個類別提供連線儲存池的基本機能，應用程式可以使用這個類別來自訂儲存池的行爲。

在建立好案例之後，便可以在 MQQueueManager 建構子中指定 MQSimpleConnectionManager。之後，MQSimpleConnectionManager 會管理建構好的 MQQueueManager 的基礎連線。如果 MQSimpleConnectionManager 含有適用的儲存池連線，便會重複使用這個連線，且會在 MQQueueManager.disconnect() 呼叫之後，將它送回 MQSimpleConnectionManager。

下列程式碼片段示範這個行爲：

```
MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
myConnMan.setActive(MQSimpleConnectionManager.MODE_ACTIVE);
MQQueueManager qmgr=new MQQueueManager("my.qmgr.1", myConnMan);
:
: (使用 qmgr 執行某些動作)
:
qmgr.disconnect();

MQQueueManager qmgr2=new MQQueueManager("my.qmgr.1", myConnMan);
:
: (使用 qmgr2 執行某些動作)
:
qmgr2.disconnect();
myConnMan.setActive(MQSimpleConnectionManager.MODE_INACTIVE);
```

在 qmgr.disconnect() 呼叫之後，第一個 MQQueueManager 建構子期間所造出的連線會儲存到 myConnMan 中。之後，在第二次呼叫 MQQueueManager 建構子期間，會重複使用這個連線。

第二行啓用 MQSimpleConnectionManager。最後一行停用 MQSimpleConnectionManager，毀損儲存池中所保留的任何連線。依預設，MQSimpleConnectionManager 會在 MODE_AUTO 中，這一節稍後會加以說明。

MQSimpleConnectionManager 會以「最近最常使用」爲基礎來配置連線，以「最近最不常使用」爲基礎來毀損連線。依預設，如果連線沒有使用的時間達 5 分鐘，或儲存池中有超出 10 個沒有使用的連線，便會毀損連線。您可以利用下列方式來改變這些值：

- MQSimpleConnectionManager.setTimeout()
- MQSimpleConnectionManager.setHighThreshold()

您也可以設定 MQSimpleConnectionManager 來作爲預設連線儲存池，以便在 MQQueueManager 建構子沒有提供連線管理程式時使用。

下列應用程式示範這一點：

```
import com.ibm.mq.*;
public class MQApp4
{
      public static void main(String[] args)
      {
         MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
         myConnMan.setActive(MQSimpleConnectionManager.MODE_AUTO);
         myConnMan.setTimeout(3600000);
         myConnMan.setHighThreshold(50);
         MQEnvironment.setDefaultConnectionManager(myConnMan);
         MQApp3.main(args);
      }
}
```

粗體的幾行會設定一個 MQSimpleConnectionManager。這項設定：

- 毀損已一個小時沒有使用的連線
- 將儲存池所保留沒有使用的連線數目限制爲 50
- 設爲 MODE_AUTO（實際上是預設值）。這表示只有在預設連線儲存池的情況下，且 MQEnvironment 所保留的 MQPoolToken 集中至少有一個記號，儲存池才會在作用中。

之後，新的 MQSimpleConnectionManager 會設爲預設連線儲存池。

在最後一行中，應用程式呼叫 MQApp3.main()。這會執行若干執行緒，其中每個執行緒皆獨立使用 MQSereis。現在，當這些執行緒造出連線時，它們會使用 myConnMan。

## 提供您自己的 ConnectionManager

在安裝了 JAAS 1.0 的 Java 2 第 1.3 版之下，應用程式和中介軟體提供者可以提供替代的連線儲存池實作。 MQ base Java 提供有 J2EE Connector Architecture 的部份實作。 **javax.resource.spi.ConnectionManager** 的實作可用來作爲預設連線管理程式，也可以在 MQQueueManager 建構子中加以指定。

MQ base Java 符合 J2EE Connector Architecture 的連線管理合約。請閱讀這一節及 J2EE Connector Architecture 的連線管理合約（請參閱 Sun 的網站，網址如下：http://java.sun.com）。

ConnectionManager 介面只定義一個方法：

```
package javax.resource.spi;
public interface ConnectionManager {
      Object allocateConnection(ManagedConnectionFactory mcf,
                                ConnectionRequestInfo cxRequestInfo);
}
```

MQQueueManager 建構子會在適當的 ConnectionManager 中呼叫 allocateConnection。它會將適當的 ManagedConnectionFactory 和 ConnectionRequestInfo 實作當作參數來傳遞，以說明所需要的連線。

ConnectionManager 會搜尋儲存池來尋找用相同的 ManagedConnectionFactory 和 ConnectionRequestInfo 物件建立的 javax.resource.spi.ManagedConnection 物件。如果 ConnectionManager 找到任何適用的 ManagedConnection 物件，它會建立一個含有候選 ManagedConnection 的 java.util.Set。之後，ConnectionManager 會呼叫：

```
ManagedConnection mc=mcf.matchManagedConnections(connectionSet, subject,
cxRequestInfo);
```

ManagedConnectionFactory 的 MQSeries 實作會忽略 subject 參數。這個方法會從這個集中選取和傳回適當的 ManagedConnection，如果找不到適當的 ManagedConnection 的話，會傳回空值。如果儲存池中沒有適當的 ManagedConnection，ConnectionManager 可以利用下列方式來建立一個：

```
ManagedConnection mc=mcf.createManagedConnection(subject, cxRequestInfo);
```

同樣，這個 subject 參數也會被忽略。這個方法會連接到 MQSeries 佇列管理程式，並傳回代表新造出之連線的 javax.resource.spi.ManagedConnection 實作。在 ConnectionManager 取得 ManagedConnection（從儲存池中或重新建立）之後，它會利用下列方式來建立連線控點：

```
Object handle=mc.getConnection(subject, cxRequestInfo);
```

這個連線控點可以從 allocateConnection() 中傳回。

ConnectionManager 應該利用下列方式登錄它需要 ManagedConnection：

```
mc.addConnectionEventListener()
```

如果連線發生嚴重錯誤或呼叫 MQQueueManager.disconnect()，這時 ConnectionEventListener 會獲得通知。當呼叫 MQQueueManager.disconnect() 時，ConnectionEventListener 可以執行下列中的任何動作：

- 利用 mc.cleanup() 呼叫來重設 ManagedConnection，再將 ManagedConnection 傳回儲存池
- 利用 mc.destroy() 呼叫來毀損 ManagedConnection

如果這個 ConnectionManager 要作爲預設的 ConnectionManager，它也可以登錄它需要 MQEnvironment 管理的 MQPoolToken 集的狀態。如果要這麼做，請先建構一個 MQPoolService 物件，再向 MQPoolService 物件登錄 MQPoolServicesEventListener 物件：

```
MQPoolServices mqps=new MQPoolServices();
mqps.addMQPoolServicesEventListener(listener);
```

當在集中新增或移除 MQPoolToken 時，或在預設的 ConnectionManager 改變時，接聽器都會獲得通知。 MQPoolService 物件也提供一個查詢 MQPoolToken 集現行大小的方法。

## 編譯和測試 MQ base Java 程式

在編譯 MQ base Java 程式之前，您必須確定您的 MQSeries Class for Java 安裝目錄在 CLASSPATH 環境變數中，如第7頁的『第2章 安裝程序』所說明。

如果要編譯 ″MyClass.java″ 類別，請利用下列指令：

```
javac MyClass.java
```

## 執行 MQ base Java Applet

如果您撰寫 Applet（java.applet.Applet 的子類別），您必須先建立一個參照您的類別的 HTML 檔，才能執行它。範例 HTML 檔所可能呈現的外觀：

### 執行 MQ base Java Applet

```
<html>
<body>
<applet code="MyClass.class" width=200 height=400>
</applet>
</body>
</html>
```

請將這個 HTML 檔載入具有 Java 功能的 Web 瀏覽器中，或使用 Java Development Kit (JDK) 所檢附的 Applet Viewer，來執行您的 Applet。

如果要使用 Applet Viewer，請輸入下列指令：

```
appletviewer myclass.html
```

## 執行 MQ base Java 應用程式

如果您利用從屬站或連結模式來撰寫應用程式（含有 main() 方法的類別），請利用 Java 直譯器來執行您的程式。請使用下列指令：

```
java MyClass
```

**註:** 類別名稱中的 '.class' 副檔名會略過。

## 在 CICS Transaction Server for OS/390 下執行 MQ base Java 應用程式

如果要將 Java 應用程式當作 CICS 之下的異動來執行，您必須：

1. 利用提供的 CEDA 異動，向 CICS 定義這個應用程式和異動。
2. 確定 MQSeries CICS 配接器已安裝在您的 CICS 系統中。（請參閱 *MQSeries for OS/390 System Management Guide*，以取得詳細資料。）
3. 確定 CICS 啓動 JCL（工作控制語言）中的 DHFJVM 參數中所指定的 JVM 環境含有適當的 CLASSPATH 和 LIBPATH 項目。
4. 利用您的任何正常程序來起始異動。

如果需要執行 CICS Java 異動的詳細資訊，請參閱 CICS 系統文件。

## 追蹤 MQ base Java 程式

MQ base Java 包括一種追蹤機能，當您懷疑程式碼可能有問題時，可用來產生診斷訊息。（您通常只有在要求 IBM 服務時，才會需要這項機能。）

追蹤是由 MQEnvironment 類別的 enableTracing 和 disableTracing 方法來控制的。例如：

```
MQEnvironment.enableTracing(2);    // 層次 2 的追蹤
 ...                               // 會追蹤這些指令
MQEnvironment.disableTracing();    // 再關閉追蹤
```

追蹤會寫入 Java 主控台 (System.err)。

如果您的程式是應用程式，或您利用 appletviewer 指令從本端磁碟來執行它，您也可以將追蹤輸出重新導向至您選擇的檔案中。下列程式碼片段顯示如何將追蹤輸出重新導向到稱爲 myapp.trc 的檔案中：

```
import java.io.*;

try {
  FileOutputStream
  traceFile = new FileOutputStream("myapp.trc");
  MQEnvironment.enableTracing(2,traceFile);
}
```

```
catch (IOException ex) {
  // 不要開啓這個檔案，
  // 改追蹤 System.err
  MQEnvironment.enableTracing(2);
}
```

追蹤有五個不同的層次：

1. 提供進入、跳出和異常狀況追蹤
2. 除了 1 以外，還提供參數資訊
3. 除了 2 以外，還提供傳輸和接收的 MQSeries 標題及資料區塊
4. 除了 3 以外，還提供傳輸和接收的使用者訊息資料
5. 除了 4 以外，還提供 Java 虛擬機器中的方法追蹤

如果要利用追蹤層次 5 來追蹤 Java 虛擬機器中的方法：

- 如果是應用程式，請發出 `java_g` 指令（而不是 `java`）
- 如果是 Applet，請發出 `appletviewer_g` 指令（而不是 `appletviewer`）

**註:**

1. 不支援在 OS/390 上的高效能 Java (HPJ) 應用程式中使用 `java_g`。
2. 不支援在 OS/400 中使用 `java_g`，但在 RUNJVA 指令中使用 `OPTION(*VERBOSE)` 可提供類似的功能。

## 追蹤 MQ base Java 程式

# 第8章 環境相依行為

這一章說明 Java 類別在各種使用環境內的行爲。您可以利用這些 MQSeries Class for Java 類別來建立使用於下列環境的應用程式：

1. 連接到 UNIX 或 Windows 平台中的 MQSeries 第 2.x 版伺服器的 MQSeries Client for Java
2. 連接到 UNIX 或 Windows 平台中的 MQSeries 第 5 版伺服器的 MQSeries Client for Java
3. 執行於 UNIX 或 Windows 平台中之 MQSeries 第 5 版伺服器的 MQSeries Bindings for Java
4. 執行於 MQSeries for MVS/ESA™ 伺服器的 MQSeries Bindings for Java
5. 執行於含 CICS Transaction Server for OS/390 第 1.3 版之 MQSeries for MVS/ESA 伺服器的 MQSeries Bindings for Java

在所有情況中，MQSeries Class for Java 程式碼都會利用基礎 MQSeries 伺服器所提供的服務。在函數的層次上，有些差異存在（比方說，MQSeries 第 5 版提供有第 2 版函數的超集）。部份 API 呼叫和選項的行爲也有些差異。多數行爲差異都是次要的，其中的大部份都是在 OS/390 (MQSeries for MVS/ESA) 伺服器和其它平台的伺服器之間。

MQSeries Class for Java 有一個類別的「基核」，它在所有環境中提供一致的函數和行爲。它也提供有專爲了在2和3環境中使用而設計的第 5 版延伸規格。下列各節依次說明基核和這些延伸規格。

## 基核詳細資料

MQSeries Class for Java 含有下列基核類別，可使用於所有的環境，些許的差異已列示於第72頁的『基核類別的限制和差異』。

- MQEnvironment
- MQException
- MQGetMessageOptions

  排除：
  - MatchOptions
  - GroupStatus
  - SegmentStatus
  - Segmentation
- MQManagedObject

  排除：
  - inquire()
  - set()
- MQMessage

  排除：
  - groupId
  - messageFlags
  - messageSequenceNumber

  - offset
  - originalLength
- MQPoolServices
- MQPoolServicesEvent
- MQPoolServicesEventListener
- MQPoolToken
- MQPutMessageOptions

  排除：
  - knownDestCount
  - unknownDestCount
  - invalidDestCount
  - recordFields
- MQProcess
- MQQueue
- MQQueueManager

  排除：
  - begin()
  - accessDistributionList()
- MQSimpleConnectionManager
- MQC

**註:**

1. 部份常數沒有併在基核中（請參閱『基核類別的限制和差異』，以取得詳細資料），在完全可攜的程式中，不應該使用它們。
2. 部份平台不支援所有連線模式。在這些平台中，您只能使用基核類別及支援的模式所關聯的選項。（請參閱第5頁的表1。）

## 基核類別的限制和差異

雖然在所有環境中，基核類別的行爲大體一致，但它們仍有些些許的限制和差異，請參閱表12，以取得相關說明。

除了所說明的這些差異之外，基核類別在所有環境上都提供一致的行爲，即使對等的 MQSeries 類別通常會有環境差異也一樣。大體上，行爲會相同，如同在2和3環境中。

*表 12. 基核類別限制和差異*

| 類別或元素 | 限制和差異 |
| --- | --- |
| MQGMO_LOCK<br>MQGMO_UNLOCK<br>MQGMO_BROWSE_MSG_UNDER_CURSOR | 當在4或5環境中使用時，會造成 MQRC_OPTIONS_ERROR。 |
| MQPMO_NEW_MSG_ID<br>MQPMO_NEW_CORREL_ID<br>MQPMO_LOGICAL_ORDER | 除了在2和3環境之外，都會造成錯誤。（請參閱第 5 版延伸規格。） |
| MQGMO_LOGICAL_ORDER<br>MQGMO_COMPLETE_MESSAGE<br>MQGMO_ALL_MSGS_AVAILABLE<br>MQGMO_ALL_SEGMENTS_AVAILABLE | 除了在2和3環境之外，都會造成錯誤。（請參閱第 5 版延伸規格。） |
| MQGMO_SYNCPOINT_IF_PERSISTENT | 在1環境中會造成錯誤。（請參閱第 5 版延伸規格。） |

*表 12. 基核類別限制和差異 (繼續)*

| 類別或元素 | 限制和差異 |
|---|---|
| MQGMO_MARK_SKIP_BACKOUT | 除了在4和5環境中，都會造成 MQRC_OPTIONS_ERROR。 |
| MQCNO_FASTPATH_BINDING | 只在3環境中有支援。（請參閱第 5 版延伸規格。） |
| MQPMRF_* fields | 只有在2和3環境中有支援。 |
| 使用 MQQueue.priority > MaxPriority 來放置訊息 | 在4和5環境中，會遭到拒絕，出現 MQCC_FAILED 和 MQRC_PRIORITY_ERROR。<br><br>其它環境會接受它，但會出現 MQCC_WARNING 和 MQRC_PRIORITY_EXCEEDS_MAXIMUM 警告，且處理訊息時，會將它視爲以 MaxPriority 來放置。 |
| BackoutCount | 4和5環境會傳回回計數上限 255，即使訊息已回復超出 255 次也一樣。 |
| 預設值動態佇列名稱 | CSQ.*，適用於4和5環境。 AMQ.*，適用於其它環境。 |
| MQMessage.report 選項：<br>MQRO_EXCEPTION_WITH_FULL_DATA<br>MQRO_EXPIRATION_WITH_FULL_DATA<br>MQRO_COA_WITH_FULL_DATA<br>MQRO_COD_WITH_FULL_DATA<br>MQRO_DISCARD_MSG | 雖然所有環境都可以設定它們，但如果報告訊息是 OS/390 佇列管理程式所產生的，則不支援。這個問題會影響到所有 Java 環境，因爲 OS/390 佇列管理程式可能會離 Java 應用程式很遠。如果有可能呼叫 OS/390 佇列管理程式的話，請避免依賴任何這些選項。 |
| MQQueueManager.commit() 和 MQQueueManager.backout() | 在5環境中，這些方法會傳回 MQRC_ENVIRONMENT_ERROR。在這個環境中，應用程式應該使用 JCICS 作業同步化方法：com.ibm.cics.server.Task.commit()，以及 com.ibm.cics.server.Task.rollback()。 |
| MQQueueManager 建構子 | 在 4 和 5 環境中，如果 MQEnvironment 中的選項（及選用的內容引數）隱含從屬站連線，建構子會失敗，並出現 MQRC_ENVIRONMENT_ERROR。<br><br>在 4 和 5 環境中，建構子也可以傳回 MQRC_CHAR_CONVERSION_ERROR。請確定已安裝 OS/390 Language Environment® 的國家語言資源元件。特別是要確定可在 IBM-1047 和 ISO8859-1 字碼頁之間進行轉換。<br><br>在 4 和 5 環境中，建構子也可以傳回 MQRC_UCS2_CONVERSION_ERROR。 MQSeries Classes for Java 會試圖從 Unicode 轉換成佇列管理程式字碼頁，如果無法使用特定字碼頁的話，會預設爲 IBM-500。請確定您有適當的 Unicode 轉換表，它應該安裝成 OS/390 C/C++ 選用特性的一部份，另外，也請確定 Language Environment 能夠找到這些表格。請參閱 *OS/390 C/C++ Programming Guide*, SC09-2362，以取得啓用 UCS-2 轉換的詳細資訊。 |

## 在其它環境中運作的第 5 版延伸規格

MQSeries Class for Java 含有專爲了使用 MQSeries 第 5 版所引進之 API 延伸規格而設計的下列函數。這些函數的運作如同只在 2 和 3 中設計一般。這個主題說明它們在其它環境中的運作方式。

### MQQueueManager 建構子選項

MQQueueManager 建構子包括選用的整數引數。這會對映到 MQI 的 MQCNO.options 欄位，可用來切換一般連線和捷徑連線。如果使用的選項只有 MQCNO_STANDARD_BINDING 或 MQCNO_FASTPATH_BINDING 的話，所有環境都能接受這個延伸的建構子形式。任何其它選項都會使建構子失敗，並出現 MQRC_OPTIONS_ERROR。只有用在 MQSeries 第 5 版連線（3 環境）時，才會接受捷徑選項 MQC.MQCNO_FASTPATH_BINDING。如果在任何其它環境中使用這個選項，都會予以忽略。

### MQQueueManager.begin() 方法

這只適用於 3 環境。在任何其它環境中，它都會失敗，出現 MQRC_ENVIRONMENT_ERROR。 MQSeries for AS/400 不支援使用 `begin()` 方法來起始佇列管理程式所協調的廣域工作單元。

### MQPutMessageOptions 選項

在任何環境中，都可以將下列旗號設到 MQPutMessageOptions 選項欄位中。不過，如果要在 2 或 3 以外的任何環境中，搭配後續的 MQQueue.put() 來使用這些旗號，put() 會失敗，並出現 MQRC_OPTIONS_ERROR。

- MQPMO_NEW_MSG_ID
- MQPMO_NEW_CORREL_ID
- MQPMO_LOGICAL_ORDER

### MQGetMessageOptions 選項

在任何環境中，都可以將下列旗號設到 MQGetMessageOptions 選項欄位中。不過，如果要在 2 或 3 以外的任何環境中，搭配後續的 MQQueue.put() 來使用這些旗號，get() 會失敗，並出現 MQRC_OPTIONS_ERROR。

- MQGMO_LOGICAL_ORDER
- MQGMO_COMPLETE_MESSAGE
- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_ALL_SEGMENTS_AVAILABLE

在任何環境中，都可以將下列旗號設到 MQGetMessageOptions 選項欄位中。不過，如果要在 1 環境中搭配後續的 MQQueue.get() 來使用這些旗號，get() 會失敗，並出現 MQRC_OPTIONS_ERROR。

- MQGMO_SYNCPOINT_IF_PERSISTENT

### MQGetMessageOptions 欄位

不論環境爲何，值都可以設到下列欄位中。不過，當執行於 2 或 3 以外的任何環境時，如果後續的 MQQueue.get() 中所用的 MQGetMessageOption 含有非預設值，get() 會失敗，並出現 MQRC_GMO_ERROR。這表示在 2 或 3 以外的環境中，在每次順利 get() 之後，這些欄位永遠會設爲它們的起始值。

- MatchOptions
- GroupStatus
- SegmentStatus
- Segmentation

### 分送清單

下列類別用來建立分送清單：

- MQDistributionList
- MQDistributionListItem
- MQMessageTracker

您可以在任何環境中建立和移入 MQDistributionList 和 MQDistributionListItem，但您只能在 2 和 3 環境中順利建立和開啓 MQDistributionList。當您試圖在任何其它環境中建立和開啓 MQDistributionList 時，都會遭到拒絕，出現 MQRC_OD_ERROR。

### MQPutMessageOptions 欄位

在 MQPutMessageOption 類別中，MQPMO 中的四個欄位會轉換成下列成員變數：

- knownDestCount
- unknownDestCount
- invalidDestCount
- recordFields

雖然主要用途是搭配分送清單使用，但 MQSeries 第 5 版伺服器在 MQPUT 到單一佇列之後，也會填入 DestCount 欄位中。比方說，如果佇列解析成本端佇列，knownDestCount 會設爲 1，另兩個欄位會設爲 0。在 2 和 3 環境中，會在 MQPutMessageOption 類別中傳回第 5 版伺服器所設的值。在其它環境中，會依照下列方式來模擬回覆值：

- 如果 put() 順利完成，unknownDestCount 會設爲 1，其它欄位會設爲 0。
- 如果 put() 失敗，invalidDestCount 會設爲 1，其它欄位會設爲 0。

recordFields 要搭配分送清單使用。不論在什麼環境中，值隨時都可以寫入 recordField。不過，如果 MQPutMessage 選項用在後續的 MQQueue.put() 中，而不是 MQDistributionList.put()，則會予以忽略。

### MQMD 欄位

下列 MQMD 欄位與訊息的分段關係非常密切：

- GroupId
- MsgSeqNumber
- Offset MsgFlags
- OriginalLength

如果應用程式將任何這些 MQMD 欄位設成非預設值，之後，又在 2 或 3 以外的環境中執行 put() 或 get()，這時 put() 或 get() 會產生異常狀況 (MQRC_MD_ERROR)。在 2 或 3 以外的環境中順利進行 put() 或 get()，永遠會使新的 MQMD 欄位設爲它們的預設值。分組或分段的訊息不應正常傳送

至就 MQSeries 第 5 版或以上版本之佇列管理程式而執行的 Java 應用程式。如果這樣的應用程式發出 get，而要擷取的實際訊息是分組或分段訊息的一部份（MQMD 欄位有非預設值），這時會進行擷取，不會有任何錯誤。不過，MQMessage 中的 MQMD 欄位不會更新。 MQMessage 格式內容會設爲 MQFMT_MD_EXTENSION，眞的訊息資料前面會附加含有新欄位值的 MQMDE 結構。

# 第9章 MQ base Java 類別和介面

本章說明所有 MQSeries Class for Java 類別和介面。其中包括每個類別和介面的變數、建構子和方法的詳細資料。

以下是所說明的類別：

- MQChannelDefinition
- MQChannelExit
- MQDistributionList
- MQDistributionListItem
- MQEnvironment
- MQException
- MQGetMessageOptions
- MQManagedObject
- MQMessage
- MQMessageTracker
- MQPoolServices
- MQPoolServicesEvent
- MQPoolToken
- MQPutMessageOptions
- MQProcess
- MQQueue
- MQQueueManager
- MQSimpleConnectionManager

以下是所說明的介面：

- MQC
- MQPoolServicesEventListener
- MQConnectionManager
- MQReceiveExit
- MQSecurityExit
- MQSendExit
- ManagedConnection
- ManagedConnectionFactory
- ManagedConnectionMetaData

# MQChannelDefinition

```
java.lang.Object
  |
  └ com.ibm.mq.MQChannelDefinition
```

public class **MQChannelDefinition**
extends **Object**

MQChannelDefinition 類別用來傳遞佇列管理程式連線的相關資訊到傳送、接收和安全跳出程式中。

**註:** 在採用連結模式直接連到 MQSeries 時，不適合採用這個類別。

## 變數

### channelName

```
public String channelName
```

用來建立連線的通道名稱。

### queueManagerName

```
public String queueManagerName
```

連線所連接的佇列管理程式名稱。

### maxMessageLength

```
public int maxMessageLength
```

可傳送到佇列管理程式的訊息長度上限。

### securityUserData

```
public String securityUserData
```

要使用的安全跳出程式的儲存區。放在這裡的資訊會保留下來而跨越不同的安全跳出程式呼叫，且可供傳送和接收跳出程式使用。

### sendUserData

```
public String sendUserData
```

要使用的傳送跳出程式的儲存區。放在這裡的資訊會保留下來而跨越不同的傳送跳出程式呼叫，且可供安全和接收跳出程式使用。

### receiveUserData

```
public String receiveUserData
```

要使用的接收跳出程式的儲存區。放在這裡的資訊會保留下來而跨越不同的接收跳出程式呼叫，且可供傳送和安全跳出程式使用。

### connectionName

```
public String connectionName
```

佇列管理程式常駐其中之機器的 TCP/IP 主電腦名稱。

### remoteUserId

```
public String remoteUserId
```

用來建立連線的使用者 ID。

### remotePassword

```
public String remotePassword
```

用來建立連線的密碼。

## 建構子

### MQChannelDefinition

```
public MQChannelDefinition()
```

# MQChannelExit

```
java.lang.Object
   |
   +-- com.ibm.mq.MQChannelExit
```

public class **MQChannelExit**
extends **Object**

這個類別定義在呼叫傳送、接收和安全跳出程式時傳遞給這些跳出程式的環境定義資訊。 exitResponse 成員變數應該由跳出程式來設定，以指出 MQSeries Client for Java 應該接著採取什麼動作。

**註:** 在採用連結模式直接連到 MQSeries 時，不適合採用這個類別。

## 變數

**MQXT_CHANNEL_SEC_EXIT**

```
public final static int MQXT_CHANNEL_SEC_EXIT
```

**MQXT_CHANNEL_SEND_EXIT**

```
public final static int MQXT_CHANNEL_SEND_EXIT
```

**MQXT_CHANNEL_RCV_EXIT**

```
public final static int MQXT_CHANNEL_RCV_EXIT
```

**MQXR_INIT**

```
public final static int MQXR_INIT
```

**MQXR_TERM**

```
public final static int MQXR_TERM
```

**MQXR_XMIT**

```
public final static int MQXR_XMIT
```

**MQXR_SEC_MSG**

```
public final static int MQXR_SEC_MSG
```

**MQXR_INIT_SEC**

```
public final static int MQXR_INIT_SEC
```

**MQXCC_OK**

```
public final static int MQXCC_OK
```

**MQXCC_SUPPRESS_FUNCTION**

```
public final static int MQXCC_SUPPRESS_FUNCTION
```

**MQXCC_SEND_AND_REQUEST_SEC_MSG**

```
public final static int MQXCC_SEND_AND_REQUEST_SEC_MSG
```

**MQXCC_SEND_SEC_MSG**

```
public final static int MQXCC_SEND_SEC_MSG
```

**MQXCC_SUPPRESS_EXIT**

```
public final static int MQXCC_SUPPRESS_EXIT
```

**MQXCC_CLOSE_CHANNEL**

```
public final static int MQXCC_CLOSE_CHANNEL
```

**exitID** `public int exitID`

已呼叫的跳出程式類型。對於 MQSecurityExit 而言，這永遠是 MQXT_CHANNEL_SEC_EXIT。對於 MQSendExit 而言，這永遠是 MQXT_CHANNEL_SEND_EXIT，對於 MQReceiveExit 而言，這永遠是 MQXT_CHANNEL_RCV_EXIT。

**exitReason**

`public int exitReason`

呼叫跳出程式的原因。可能的值如下：

**MQXR_INIT**
跳出程式起始設定；在協議好通道連線狀況之後、傳送任何安全串流之前呼叫。

**MQXR_TERM**
跳出程式終止；在傳送切斷連線串流之後、毀損 Socket 連線之前呼叫。

**MQXR_XMIT**
對於傳送跳出程式，指出資料要傳輸到佇列管理程式。

對於接收跳出程式，指出已收到佇列管理程式所提供的資料。

**MQXR_SEC_MSG**
告訴安全跳出程式已收到佇列管理程式所提供的安全訊息。

**MQXR_INIT_SEC**
指出跳出程式要以佇列管理程式來起始安全對話框。

**exitResponse**

`public int exitResponse`

由跳出程式設定，用以指出 MQSeries Class for Java 應該接著採取什麼動作。有效值如下：

**MQXCC_OK**
由安全跳出程式設定，用以指出安全交換已經完成。

由傳送跳出程式設定，用以指出傳回的資料要傳輸給佇列管理程式。

由接收跳出程式設定，用以指出 MQSeries Client for Java 可以處理傳回的資料。

**MQXCC_SUPPRESS_FUNCTION**
由安全跳出程式設定，用以指出應該關閉與佇列管理程式的通信。

**MQXCC_SEND_AND_REQUEST_SEC_MSG**
由安全跳出程式設定，用以指出傳回的資料要傳輸給佇列管理程式，且預期佇列管理程式會有回應。

**MQXCC_SEND_SEC_MSG**
由安全跳出程式設定，用以指出傳回的資料要傳輸給佇列管理程式，且預期不會有回應。

**MQXCC_SUPPRESS_EXIT**
由任何跳出程式設定，用以指出不應該再呼叫它。

**MQXCC_CLOSE_CHANNEL**
由任何跳出程式設定，用以指出應該關閉與佇列管理程式的連線。

**maxSegmentLength**

```
public int maxSegmentLength
```

任何指向佇列管理程式的傳輸之長度上限。

如果跳出程式傳回要送到佇列管理程式的資料，傳回資料的長度不應超出這個值。

**exitUserArea**

```
public byte exitUserArea[]
```

跳出程式可以使用的儲存區。

MQSeries Client for Java 會保留放在 exitUserArea 中的任何資料，以跨越相同 exitID 的跳出程式呼叫。（也就是說，傳送、接收和安全跳出程式都有自己獨立的使用者區域。）

**capabilityFlags**

```
public static final int capabilityFlags
```

指出佇列管理程式的性能。

只支援 MQC.MQCF_DIST_LISTS 旗號。

**fapLevel**

```
public static final int fapLevel
```

協議格式和通信協定 (FAP) 層次。

## 建構子

**MQChannelExit**

```
public MQChannelExit()
```

# MQDistributionList

```
java.lang.Object
   |
   +-- com.ibm.mq.MQManagedObject
          |
          +-- com.ibm.mq.MQDistributionList
```

public class **MQDistributionList**
extends **MQManagedObject**（請參閱頁 97。）

**註:** 只有在連接到 MQSeries 第 5 版（或以上）佇列管理程式時，才可以使用這個類別。

MQDistributionList 是利用 MQDistributionList 建構子或 MQQueueManager 的 accessDistributionList 方法建立的。

分送清單代表可利用單一 put() 方法呼叫將訊息送往的一組開放佇列。（請參閱 *MQSeries Application Programming Guide* 中的「分送清單」。）

## 建構子

### MQDistributionList

```
public MQDistributionList(MQQueueManager qMgr,
              MQDistributionListItem[] litems,
              int openOptions,
              String alternateUserId)
                        throws MQException
```

qMgr 是要開啓清單的佇列管理程式。

litems 是分送清單中所併入的項目。

請參閱第146頁的『accessDistributionList』，以取得其餘參數的詳細資訊。

## 方法

### put

```
public synchronized void put(MQMessage message,
             MQPutMessageOptions putMessageOptions )
                       throws MQException
```

將訊息放入分送清單中的佇列中。

#### 參數

*message*
含有訊息描述子資訊及傳回的訊息資料之輸入/輸出參數。

*putMessageOptions*
控制 MQPUT 動作的選項。（請參閱第127頁的『MQPutMessageOptions』，以取得詳細資料。）

如果 put 失敗的話，會擲出 MQException。

## MQDistributionList

### getFirstDistributionListItem

```
public MQDistributionListItem getFirstDistributionListItem()
```

傳回分送清單中的第一個項目，如果清單是空的，則傳回*空值*。

### getValidDestinationCount

```
public int getValidDestinationCount()
```

傳回順利開啓的分送清單其中的項目數。

### getInvalidDestinationCount

```
public int getInvalidDestinationCount()
```

傳回無法開啓的分送清單其中的項目數。

# MQDistributionListItem

```
java.lang.Object
   |
   +-- com.ibm.mq.MQMessageTracker
          |
          +-- com.ibm.mq.MQDistributionListItem
```

public class **MQDistributionListItem**
extends **MQMessageTracker**（請參閱頁 119。）

**註:** 只有在連接到 MQSeries 第 5 版（或以上）佇列管理程式時，才可以使用這個類別。

MQDistributionListItem 代表分送清單內的單一項目（佇列）。

## 變數

### completionCode

```
public int completionCode
```

這個項目的前次作業所產生的完成碼。如果這是 MQDistributionList 的建構，完成碼會關聯於佇列的開啓動作。如果是放置作業的話，完成碼會關聯於將訊息放入這個佇列的嘗試。

起始值是 ″0″。

### queueName

```
public String queueName
```

要搭配分送清單使用的佇列名稱。這個通道不能是模型佇列的名稱。

起始值是 ″″。

### queueManagerName

```
public String queueManagerName
```

定義佇列的佇列管理程式名稱。

起始值是 ″″。

### reasonCode

```
public int reasonCode
```

這個項目的前次作業所產生的原因碼。如果這是 MQDistributionList 的建構，原因碼會關聯於佇列的開啓動作。如果是放置作業的話，原因碼會關聯於將訊息放入這個佇列的嘗試。

起始值是 ″0″。

## 建構子

### MQDistributionListItem

```
public MQDistributionListItem()
```

建構一個新的 MQDistributionListItem 物件。

# MQEnvironment

```
java.lang.Object
   |
   +-- com.ibm.mq.MQEnvironment
```

public class **MQEnvironment**
extends **Object**

**註:** 這個類別的所有方法和屬性都適用於 MQSeries Class for Java 從屬站連線，但只有 enableTracing、disableTracing、properties 和 version_notice 適用於連結連線。

MQEnvironment 含有控制建構 MQQueueManager 物件（及其指向 MQSeries 的對應連線）之環境的靜態成員變數。

MQEnvironment 類別中的值集會在呼叫 MQQueueManager 建構子時生效，因此，您應該先設定 MQEnvironment 類別中的值，之後，才建構 MQQueueManager 案例。

## 變數

**註:** 在採用連結模式直接連到 MQSeries 時，不適合採用有 * 標示的變數。

### version_notice

```
public final static String version_notice
```

MQSeries Class for Java 的現行版本。

### securityExit*

```
public static MQSecurityExit securityExit
```

安全跳出程式可讓您自訂試圖連接佇列管理程式時所發生的安全串流。

如果要提供自己的安全跳出程式，請定義一個類別來實作 MQSecurityExit 介面，並指派一個 securityExit 給這個類別的案例。否則，您可以將 securityExit 設爲空值，這時不會呼叫任何安全跳出程式。

另請參閱第155頁的『MQSecurityExit』。

### sendExit*

```
public static MQSendExit sendExit
```

傳送跳出程式可讓您檢查傳送到佇列管理程式的資料，也可以改變它。它通常要搭配在佇列管理程式的對應接收跳出程式。

如果要提供自己的傳送跳出程式，請定義一個類別來實作 MQSendExit 介面，並指派一個 sendExit 給這個類別的案例。否則，您可以將 sendExit 設爲空值，這時不會呼叫任何傳送跳出程式。

另請參閱第157頁的『MQSendExit』。

### receiveExit*

```
public static MQReceiveExit receiveExit
```

接收跳出程式可讓您檢查從佇列管理程式接收來的資料，也可以改變它。它通常要搭配在佇列管理程式的對應傳送跳出程式。

如果要提供自己的接收跳出程式，請定義一個類別來實作 MQReceiveExit 介面，並指派一個 receiveExit 給這個類別的案例。否則，您可以將 receiveExit 設爲空值，這時不會呼叫任何接收跳出程式。

另請參閱第153頁的『MQReceiveExit』。

**hostname***

```
public static String hostname
```

MQSeries 伺服器常駐其中之機器的 TCP/IP 主電腦名稱。如果沒有設定主電腦名稱，也沒有設定置換內容，這時會利用連結模式來連接本端佇列管理程式。

**port*** `public static int port`

要連接的埠。這是 MQSeries 伺服器用來接聽內收連線要求的埠。預設值是 1414。

**channel***

```
public static String channel
```

連接到目標佇列管理程式的通道名稱。您*必須*先設定這個成員變數或對應的內容，才能建構從屬站模式中所用的 MQQueueManager 案例。

**userID***

```
public static String userID
```

相當於 MQSeries 環境變數 MQ_USER_ID。

如果沒有定義這個從屬站的安全跳出程式，會將使用者 ID 值傳給伺服器，當呼叫伺服器安全跳出程式時，可以使用這個值。這個值可用來驗證 MQSeries 從屬站的身份。

預設值是 ""。

**password***

```
public static String password
```

這相當於 MQSeries 環境變數 MQ_PASSWORD。

如果沒有定義這個從屬站的安全跳出程式，會將密碼值傳給伺服器，當呼叫伺服器安全跳出程式時，可以使用這個值。這個值可用來驗證 MQSeries 從屬站的身份。

預設值是 ""。

**properties**

```
public static java.util.Hashtable properties
```

一組定義 MQSeries 環境的鍵/值配對。

這個雜湊表可讓您將環境內容設爲鍵/值配對，而不是個別的變數。

內容也可以作爲雜湊表而在 MQQueueManager 建構子的參數中傳遞。建構子所傳遞的內容優先於這個內容變數所設定的值，否則，它們可以互換。以下是尋找內容的優先順序：

1. MQQueueManager 建構子中的 `properties` 參數
2. MQEnvironment.properties
3. 其它 MQEnvironment 變數
4. 常數預設值

下表中顯示可能的鍵/值配對：

| 鍵 | 值 |
|---|---|
| MQC.CCSID_PROPERTY | Integer（置換 MQEnvironment.CCSID） |
| MQC.CHANNEL_PROPERTY | String（置換 MQEnvironment.channel） |
| MQC.CONNECT_OPTIONS_PROPERTY | Integer，預設值爲 MQC.MQCNO_NONE |
| MQC.HOST_NAME_PROPERTY | String（置換 MQEnvironment.hostname） |
| MQC.ORB_PROPERTY | org.omg.CORBA.ORB（選用） |
| MQC.PASSWORD_PROPERTY | String（置換 MQEnvironment.password） |
| MQC.PORT_PROPERTY | Integer（置換 MQEnvironment.port） |
| MQC.RECEIVE_EXIT_PROPERTY | MQReceiveExit（置換 MQEnvironment.receiveExit） |
| MQC.SECURITY_EXIT_PROPERTY | MQSecurityExit（置換 MQEnvironment.securityExit） |
| MQC.SEND_EXIT_PROPERTY | MQSendExit（置換 MQEnvironment.sendExit.） |
| MQC.TRANSPORT_PROPERTY | MQC.TRANSPORT_MQSERIES_BINDINGS<br>或<br>MQC.TRANSPORT_MQSERIES_CLIENT<br>或<br>MQC.TRANSPORT_VISIBROKER<br>或<br>MQC.TRANSPORT_MQSERIES（預設值，根據 ″hostname″ 值來選取連結或從屬站。） |
| MQC.USER_ID_PROPERTY | String（置換 MQEnvironment.userID） |

### CCSID*

```
public static int CCSID
```

從屬站所用的 CCSID。

變更這個值會影響連接的佇列管理程式在 MQSeries 標題中轉換資訊的方式。 MQSeries 標題中的所有資料都會取自 ASCI 字碼集的不變部份，但 MQMessage 類別的 applicationIdData 和 putApplicationName 欄位中的資料除外。（請參閱第100頁的『MQMessage』。）

如果避免在這兩個欄位中使用 ASCII 字碼集可變部份的字元，您便可以放心將 CCSID 819 改成任何其它 ASCII 字碼集。

如果您將從屬站的 CCSID 改成與所連接之佇列管理程式相同的 CCSID，佇列管理程式的效能會更好，因爲它不需要轉換訊息標題。

預設值是 819。

## 建構子

### MQEnvironment

```
public MQEnvironment()
```

## 方法

### disableTracing

```
public static void disableTracing()
```

關閉 MQSeries Client for Java 追蹤機能。

### enableTracing

```
public static void enableTracing(int level)
```

開啓 MQSeries Client for Java 追蹤機能。

**參數**

*level* 需要的追蹤層次，1-5（5 最詳盡）。

### enableTracing

```
public static void enableTracing(int level,
                                 OutputStream stream)
```

開啓 MQSeries Client for Java 追蹤機能。

**參數：**

*level* 需要的追蹤層次，1-5（5 最詳盡）。

*stream* 追蹤寫入其中的串流。

### setDefaultConnectionManager

```
public static void setDefaultConnectionManager(MQConnectionManager cxManager)
```

將提供的 MQConnectionManager 設爲預設的 ConnectionManager。當 MQQueueManager 建構子中沒有指定 ConnectionManager 時，會使用預設 ConnectionManager。這個方法也會將 MQPoolTokens 子集清空。

**參數：**

*cxManager*
要作爲預設 ConnectionManager 的 MQConnectionManager。

### setDefaultConnectionManager

```
public static void setDefaultConnectionManager
                    (javax.resource.spi.ConnectionManager cxManager)
```

設定預設 ConnectionManager，將 MQPoolToken 集清空。當 MQQueueManager 建構子中沒有指定 ConnectionManager 時，會使用預設 ConnectionManager。

這個方法需要 Java 2 第 1.3 版或以上的 JVM，且必須安裝有 JAAS 1.0 或以上。

**參數：**

*cxManager*
預設 ConnectionManager（實作 javax.resource.spi.ConnectionManager 介面）。

### getDefaultConnectionManager

```
public static javax.resource.spi.ConnectionManager
                                  getDefaultConnectionManager()
```

傳回預設的 ConnectionManager。如果預設 ConnectionManager 實際上是 MQConnectionManager，會傳回空值。

### addConnectionPoolToken

```
public static void addConnectionPoolToken(MQPoolToken token)
```

新增提供的 MQPoolToken 到記號集內。預設的 ConnectionManager 可將此當作提示，通常，只有在集中至少有一個記號時，才會啓用它。

**參數：**

*token* 要新增到記號集內的 MQPoolToken。

### addConnectionPoolToken

```
public static MQPoolToken addConnectionPoolToken()
```

建構一個 MQPoolToken，將它新增到記號集中。 MQPoolToken 會傳回給應用程式，稍後再傳到 removeConnectionPoolToken() 中。

### removeConnectionPoolToken

```
public static void removeConnectionPoolToken(MQPoolToken token)
```

從記號集內移除指定的 MQPoolToken。如果這個 MQPoolToken 沒有在集內的話，不會有任何動作。

**參數：**

*token* 要從記號集中移除的 MQPoolToken。

# MQException

```
java.lang.Object
   |
   +-- java.lang.Throwable
          |
          +-- java.lang.Exception
                 |
                 +-- com.ibm.mq.MQException
```

public class **MQException**
extends **Exception**

每當發生 MQSeries 錯誤時，都會擲出 MQException. 您可以設定 MQException.log 的值來變更所記錄的異常狀況輸出字串。預設值是 System.err。這個類別含有完成碼和錯誤碼常數的定義。開頭爲 MQCC_ 的常數是 MQSeries 完成碼，開頭爲 MQRC_ 的常數是 MQSeries 原因碼。 *MQSeries Application Programming Reference* 含有這些錯誤及其可能原因的完整說明。

## 變數

**log** `public static java.io.outputStreamWriter log`

異常狀況要記錄到其中的串流。（預設值是 System.err。）如果您將它設爲空值，便不會記錄任何內容。

**completionCode**

`public int completionCode`

產生錯誤的 MQSeries 完成碼。可能的值如下：

- MQException.MQCC_WARNING
- MQException.MQCC_FAILED

**reasonCode**

`public int reasonCode`

說明錯誤的 MQSeries 原因碼。如果需要原因碼的完整說明，請參閱 *MQSeries Application Programming Reference*。

**exceptionSource**

`public Object exceptionSource`

擲出異常狀況的物件案例。當您判斷錯誤原因時，這可以作爲診斷的一部份。

## 建構子

**MQException**

```
public MQException(int completionCode,
                   int reasonCode,
                   Object source)
```

建構一個新的 MQException 物件。

**參數**

*completionCode*
MQSeries 完成碼。

*reasonCode*
MQSeries 原因碼。

*source* 發生錯誤的物件。

## MQGetMessageOptions

```
java.lang.Object
   |
   +-- com.ibm.mq.MQGetMessageOptions
```

public class **MQGetMessageOptions**
extends **Object**

這個類別含有控制 MQQueue.get() 行爲的選項。

**註:** 這個類別中某些可用選項的行爲會隨著使用它們的環境而不同。這些元素會有標出 *****。請參閱第71頁的『第8章 環境相依行爲』，以取得詳細資訊。

### 變數

#### options

```
public int options
```

控制 MQQueue.get 動作的選項。您可以指定下列中的任何值，或不指定下列中的任何值。如果需要多個選項，您可以利用按位元的 OR 運算子，將這些值加起來或組合起來。

**MQC.MQGMO_NONE**

**MQC.MQGMO_WAIT**
等待訊息到達。

**MQC.MQGMO_NO_WAIT**
如果沒有適合的訊息，便立即傳回。

**MQC.MQGMO_SYNCPOINT**
在同步點控制之下取得訊息；訊息會標示爲其它應用程式無法使用，但只有在工作單元已確定之後，才會從佇列中刪除。如果回復工作單元的話，可以重新使用訊息。

**MQC.MQGMO_NO_SYNCPOINT**
取得不含同步點控制的訊息。

**MQC.MQGMO_BROWSE_FIRST**
從佇列開頭開始瀏覽。

**MQC.MQGMO_BROWSE_NEXT**
從佇列中的現行位置開始瀏覽。

**MQC.MQGMO_BROWSE_MSG_UNDER_CURSOR***
瀏覽「瀏覽游標」下的訊息。

**MQC.MQGMO_MSG_UNDER_CURSOR**
取得瀏覽游標下的訊息。

**MQC.MQGMO_LOCK***
鎖定瀏覽的訊息。

**MQC.MQGMO_UNLOCK***
解除鎖定先前鎖定的訊息。

**MQC.MQGMO_ACCEPT_TRUNCATED_MSG**
容許截斷訊息資料。

**MQC.MQGMO_FAIL_IF_QUIESCING**
如果佇列管理程式在靜止中，便告失敗。

**MQC.MQGMO_CONVERT**
要求在應用程式資料複製到訊息緩衝區之前，加以轉換，以符合 MQMessage 的 characterSet 及 encoding 屬性。從訊息緩衝區中擷取資料時也會套用資料轉換，因此，應用程式不會常常設定這個選項。

**MQC.MQGMO_SYNCPOINT_IF_PERSISTENT***
如果訊息會持續存在，便取得具有同步點控制的訊息。

**MQC.MQGMO_MARK_SKIP_BACKOUT***
容許在不復原佇列中的訊息的情況下，回復工作單元。

**分段和分組** MQSeries 訊息可以當作單一實體來傳送或接收，可以分割成多個傳送和接收的區段，也可以鏈結到群組中的其它訊息。

所傳送的每個資料片段都稱爲*實體*訊息，它可以是完整的*邏輯*訊息，也可以是較長的邏輯訊息區段。

每個實體訊息通常都會使用不同的 `MsgId`。單一邏輯訊息的所有區段都會有相同的 `groupId` 值及 `MsgSeqNumber` 值，但每個區段的 `Offset` 值都不同。 `Offset` 欄位提供在實體訊息中資料與邏輯訊息起點的偏移。區段通常會有不同的 `MsgId` 值，因爲它們是個別的實體訊息。

形成群組部份的各邏輯訊息會有相同的 `groupId` 值，但群組中的每個訊息都會有不同的 `MsgSeqNumber` 值。群組中的訊息也可以分區段。

下列選項可用來處理分段或分組的訊息：

**MQC.MQGMO_LOGICAL_ORDER***
依邏輯次序傳回群組中的訊息及邏輯訊息的區段。

**MQC.MQGMO_COMPLETE_MSG***
只擷取完整的邏輯訊息。

**MQC.MQGMO_ALL_MSGS_AVAILABLE***
只有在群組中的所有訊息都能取得時，才擷取群組中的訊息。

**MQC.MQGMO_ALL_SEGMENTS_AVAILABLE***
只有在群組中的所有區段都能取得時，才擷取邏輯訊息中的區段。

**waitInterval**

```
public int waitInterval
```

MQQueue.get 呼叫等待適當訊息到達的時間上限（秒數）（搭配 MQC.MQGMO_WAIT 使用）。 MQC.MQWI_UNLIMITED 值表示需要無限等待。

**resolvedQueueName**

```
public String resolvedQueueName
```

這是一個輸出欄位，由佇列管理程式將它設成從中擷取訊息之佇列的本端名稱。如果是開啓別名佇列或模型佇列的話，這會不同於開啓佇列時所用的名稱。

**matchOptions***

```
public int matchOptions
```

判斷要擷取什麼訊息的選取準則。以下是可設定的符合選項：

**MQC.MQMO_MATCH_MSG_ID**
要符合的訊息 ID。

**MQC.MQMO_MATCH_CORREL_ID**
要符合的相互關係 ID。

**MQC.MQMO_MATCH_GROUP_ID**
要符合的群組 ID。

**MQC.MQMO_MATCH_MSG_SEQ_NUMBER**
符合訊息序號。

**MQC.MQMO_NONE**
不需要符合。

**groupStatus***

```
public char groupStatus
```

這是一個輸出欄位，指出擷取的訊息在不在群組中，如果是的話，則指出它是不是群組中的最後一個。可能的值如下：

**MQC.MQGS_NOT_IN_GROUP**
訊息不在群組中。

**MQC.MQGS_MSG_IN_GROUP**
訊息在群組中，但不是群組中的最後一項。

**MQC.MQGS_LAST_MSG_IN_GROUP**
訊息是群組中的最後一項。這也是群組只含有一則訊息時所傳回的值。

**segmentStatus***

```
public char segmentStatus
```

這是一個輸出欄位，指出擷取的訊息是不是邏輯訊息的區段。如果訊息是一個區段的話，旗號會指出它是不是最後一個區段。可能的值如下：

**MQC.MQSS_NOT_A_SEGMENT**
訊息不是一個區段。

**MQC.MQSS_SEGMENT**
訊息是一個區段，但不是邏輯訊息中的最後一個區段。

**MQC.MQSS_LAST_SEGMENT**
訊息是邏輯訊息中的最後一個區段。這也是邏輯訊息只含有一個區段時所傳回的值。

### segmentation*

```
public char segmentation
```

這是一個輸出欄位，指出本身是邏輯訊息之區段的擷取訊息是否能分段。可能的值如下：

**MQC.MQSEG_INHIBITED**
不容許分段。

**MQC.MQSEG_ALLOWED**
容許分段。

## 建構子

### MQGetMessageOptions

```
public MQGetMessageOptions()
```

建構一個新的 MQGetMessageOptions 物件，選項設爲 MQC.MQGMO_NO_WAIT、等待間隔設爲零，且解析的佇列名稱空白。

# MQManagedObject

```
java.lang.Object
   |
   +-- com.ibm.mq.MQManagedObject
```

public class **MQManagedObject**
extends **Object**

MQManagedObject 是 MQQueueManager、MQQueue 和 MQProcess 的超類別。它提供查詢和設定這些資源的屬性的能力。

## 變數

### alternateUserId

```
public String alternateUserId
```

開啓這項資源時所指定的替代使用者 ID（如果有的話）。設定這個屬性不會有作用。

### name

```
public String name
```

這項資源的名稱（存取方法提供的名稱或佇列管理程式配置給動態佇列的名稱）。設定這個屬性不會有作用。

### openOptions

```
public int openOptions
```

開啓這項資源時所指定的選項。設定這個屬性不會有作用。

### isOpen

```
public boolean isOpen
```

指出這項資源目前是否已開啓。這個屬性*已更換*，設定它不會有作用。

### connectionReference

```
public  MQQueueManager connectionReference
```

這項資源所屬的佇列管理程式。設定這個屬性不會有作用。

### closeOptions

```
public int closeOptions
```

請設定這個屬性來控制資源的關閉方式。預設值是 MQC.MQCO_NONE，這是永久動態佇列以外的所有資源，以及建立暫時動態佇列的物件所存取之暫時動態佇列，唯一的可許可值。以下是這些佇列其它的可許可值：

**MQC.MQCO_DELETE**
如果沒有訊息的話，即刪除佇列。

**MQC.MQCO_DELETE_PURGE**
刪除佇列，除去其中的任何訊息。

## 建構子

### MQManagedObject

```
protected MQManagedObject()
```

建構子方法。

## MQManagedObject

### 方法

#### getDescription

```
public String getDescription()
```

擲出 MQException。

傳回佇列管理程式所保留的這項資源之說明。

如果在資源關閉之後呼叫這個方法，會擲出 MQException。

#### inquire

```
public void inquire(int selectors[],
                    int intAttrs[],
                    byte charAttrs[])
```

throws MQException.

傳回一個整數陣列和一組含有物件（佇列、程序或佇列管理程式）屬性的字串。

要查詢的屬性由選取元陣列來指定。請參閱 *MQSeries Application Programming Reference*，以取得可許可的選取元及其對應整數值的詳細資料。

請注意，許多更常見的屬性可利用 MQManagedObject、MQQueue、MQQueueManager 和 MQProcess 中所定義的 getXXX() 方法來查詢。

**參數**

*selectors*
指出要查詢其值的屬性之整數陣列。

*intAttrs*
傳回整數屬性值的陣列。整數屬性值的傳回次序和選取元陣列中的整數屬性選取元次序相同。

*charAttrs*
在其中連接傳回之字元屬性的緩衝區。字元屬性的傳回次序和選取元陣列中的字元屬性選取元次序相同。每個屬性的字串長度都是固定的。

如果查詢失敗的話，會擲出 MQException。

#### isOpen

```
public boolean isOpen()
```

傳回 `isOpen` 變數的值。

#### set

```
public synchronized void set(int selectors[],
                             int intAttrs[],
                             byte charAttrs[])
```

throws MQException.

設定選取元向量中所定義的屬性。

要設定的屬性由選取元陣列來指定。請參閱 *MQSeries Application Programming Reference*，以取得可許可的選取元及其對應整數值的詳細資料。

請注意，有些佇列屬性可以利用 MQQueue 中所定義 setXXX() 方法來設定。

**參數**

*selectors*
指出要設定其值的屬性之整數陣列。

*intAttrs* 要設定的整數屬性值陣列。這些值的次序必須和選取元陣列中的整數屬性選取元次序相同。

*charAttrs*
在其中連接要設定之字元屬性的緩衝區。這些值的次序必須和選取元陣列中的字元屬性選取元次序相同。每個字元屬性的長度都是固定的。

如果設定失敗的話，會擲出 MQException。

## close

```
public synchronized void close()
```

throws MQException.

關閉物件。在呼叫這個方法之後，不能再就這項資源進行任何作業。您可以設定 closeOptions 屬性來改變 close 方法的行爲。

如果 MQSeries 呼叫失敗的話，會擲出 MQException。

# MQMessage

```
java.lang.Object
   |
   +-- com.ibm.mq.MQMessage
```

public class **MQMessage**
implements **DataInput**, **DataOutput**

MQMessage 代表訊息描述子和 MQSeries 訊息的資料。 readXXX 方法群組用來讀取訊息中的資料，writeXXX 方法群組用來將資料寫入訊息中。這些讀取和寫入方法所用的數字和字串格式可由 encoding 和 characterSet 成員變數來控制。其餘成員變數含有訊息在傳送和接收端應用程式之流動時，附隨於應用程式訊息資料的控制資訊。應用程式可在訊息放入佇列之前，將值設到成員變數中，也可以在擷取某佇列中的訊息之後讀取值。

## 變數

**report** `public int report`

報告是有關另一則訊息的訊息。這個成員變數使應用程式能夠傳送原始訊息來指定需要哪些報告訊息、應用程式訊息資料是否要併入其中，以及如何在報告和回答中設定訊息和相互關係識別碼。您可以要求下列中的任何或所有報告類型，或全部不要求：

- 異常狀況
- 期限
- 在到達時確認
- 在遞送時確認

每個類型都應根據應用程式訊息資料是否要併入報告訊息中，而僅指定下列三個對應值中的一個。

**註:** 下列清單標出 **** 的值，MVS 佇列管理程式不支援它們，如果您的應用程式可能存取 MVS 佇列管理程式的話，不論應用程式執行於什麼平台，您都不應該使用它們。

有效值如下：

- MQC.MQRO_EXCEPTION
- MQC.MQRO_EXCEPTION_WITH_DATA
- MQC.MQRO_EXCEPTION_WITH_FULL_DATA**
- MQC.MQRO_EXPIRATION
- MQC.MQRO_EXPIRATION_WITH_DATA
- MQC.MQRO_EXPIRATION_WITH_FULL_DATA**
- MQC.MQRO_COA
- MQC.MQRO_COA_WITH_DATA
- MQC.MQRO_COA_WITH_FULL_DATA**
- MQC.MQRO_COD
- MQC.MQRO_COD_WITH_DATA
- MQC.MQRO_COD_WITH_FULL_DATA**

您可以指定下列其中一項來控制報告或回答訊息之訊息 ID 的產生方式：

- MQC.MQRO_NEW_MSG_ID

- MQC.MQRO_PASS_MSG_ID

您可以指定下列其中一項來控制報告或回答訊息之相互關係 ID 的設定方式：
- MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQC.MQRO_PASS_CORREL_ID

您可以指定下列其中一項來控制原始訊息不能遞送到目的地佇列時，要如何處理它：
- MQC.MQRO_DEAD_LETTER_Q
- MQC.MQRO_DISCARD_MSG **

如果沒有指定報告選項的話，預設值爲：

```
MQC.MQRO_NEW_MSG_ID |
MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID |
MQC.MQRO_DEAD_LETTER_Q
```

您可以指定下列中的一或兩項來要求接收端應用程式傳送正面動作或負面動作報告訊息。
- MQRO_PAN
- MQRO_NAN

**messageType**

```
public int messageType
```

指出訊息類型：以下是系統目前所設定的值：
- MQC.MQMT_DATAGRAM
- MQC.MQMT_REQUEST
- MQC.MQMT_REPLY
- MQC.MQMT_REPORT

您也可以使用應用程式定義的值。這些都應該在 MQC.MQMT_APPL_FIRST 至 MQC.MQMT_APPL_LAST 的範圍內。

這個欄位的預設值是 MQC.MQMT_DATAGRAM。

**expiry** `public int expiry`

期限時間以十分之一秒來表示，由放置訊息的應用程式來設定。在過了訊息的期限時間之後，佇列管理程式便可以將它捨棄。如果訊息指定了某個 MQC.MQRO_EXPIRATION 旗號，當捨棄訊息時，會產生一份報告。

預設值是 MQC.MQEI_UNLIMITED，表示訊息永遠不會到期。

**feedback**

```
public int feedback
```

這用來搭配 MQC.MQMT_REPORT 類型的訊息指出報告的性質。以下是系統所定義的傳回訊息碼：
- MQC.MQFB_EXPIRATION
- MQC.MQFB_COA
- MQC.MQFB_COD
- MQC.MQFB_QUIT
- MQC.MQFB_PAN
- MQC.MQFB_NAN
- MQC.MQFB_DATA_LENGTH_ZERO
- MQC.MQFB_DATA_LENGTH_NEGATIVE

- MQC.MQFB_DATA_LENGTH_TOO_BIG
- MQC.MQFB_BUFFER_OVERFLOW
- MQC.MQFB_LENGTH_OFF_BY_ONE
- MQC.MQFB_IIH_ERROR

應用程式所定義、在 MQC.MQFB_APPL_FIRST 至 MQC.MQFB_APPL_LAST 範圍的傳回訊息也可以使用。

這個欄位的預設值是 MQC.MQFB_NONE，表示不提供任何傳回的訊息。

### encoding

```
public int encoding
```

這個成員變數會指定應用程式訊息資料的數值所用的表示法；這適用於二進位、聚集十進位及浮點資料。這些數值格式的讀取和寫入方法的行爲也會據此而改變。

以下是定義好的二進位整數編碼：

**MQC.MQENC_INTEGER_NORMAL**
大序排列法整數，如在 Java 中。

**MQC.MQENC_INTEGER_REVERSED**
小序排列法整數，如 PC 所使用。

以下是定義好的聚集十進位整數編碼：

**MQC.MQENC_DECIMAL_NORMAL**
大序排列法聚集十進位，如 System/390 所使用。

**MQC.MQENC_DECIMAL_REVERSED**
小序排列法聚集十進位。

以下是定義好的浮點數編碼：

**MQC.MQENC_FLOAT_IEEE_NORMAL**
大序排列法浮點數，如在 Java 中。

**MQC.MQENC_FLOAT_IEEE_REVERSED**
小序排列法 IEEE 浮點數，如 PC 所用。

**MQC.MQENC_FLOAT_S390**
System/390 格式浮點。

編碼欄位值的建構方式應該是三個區段各新增其中一值，將它們合起來（或使用按位元的 OR 運算子）。預設值是：

```
MQC.MQENC_INTEGER_NORMAL |
MQC.MQENC_DECIMAL_NORMAL |
MQC.MQENC_FLOAT_IEEE_NORMAL
```

爲了方便，這個值也可以由 MQC.MQENC_NATIVE 來表示。這個設定會使 writeInt() 寫入一個大序排列法設定，使 readInt() 讀取一個大序排列法整數。如果改設了 MQC.MQENC_INTEGER_REVERSED 旗號，writeInt() 會寫入小序排列法整數， readInt() 會讀取小序排列法整數。

請注意，當您從 IEEE 格式的浮點轉換成 System/390 格式的浮點時，可能會失去精準度。

### characterSet

```
public int characterSet
```

這指定應用程式訊息資料中之字元資料的編碼字集識別碼。 readString、readLine 和 writeString 方法的行爲也據此而改變。

這個欄位的預設值是 MQC.MQCCSI_Q_MGR，它指定應用程式訊息資料中的字元資料是在佇列管理程式的字集中。 表13.中所顯示的其它字集值也有支援。

*表 13. 字集識別碼*

| characterSet | 說明 |
|---|---|
| 819 | iso-8859-1 / 拉丁文 1 / ibm819 |
| 912 | iso-8859-2 / 拉丁文 2 / ibm912 |
| 913 | iso-8859-3 / 拉丁文 3 / ibm913 |
| 914 | iso-8859-4 / 拉丁文 4 / ibm914 |
| 915 | iso-8859-5 / 斯拉夫文 / ibm915 |
| 1089 | iso-8859-6 / 阿拉伯文 / ibm1089 |
| 813 | iso-8859-7 / 希臘文 / ibm813 |
| 916 | iso-8859-8 / 希伯來文 / ibm916 |
| 920 | iso-8859-9 / 拉丁文 5 / ibm920 |
| 37 | ibm037 |
| 273 | ibm273 |
| 277 | ibm277 |
| 278 | ibm278 |
| 280 | ibm280 |
| 284 | ibm284 |
| 285 | ibm285 |
| 297 | ibm297 |
| 420 | ibm420 |
| 424 | ibm424 |
| 437 | ibm437 / PC 原始語言 |
| 500 | ibm500 |
| 737 | ibm737 / PC 希臘文 |
| 775 | ibm775 / PC 波羅的海語系 |
| 838 | ibm838 |
| 850 | ibm850 / PC 拉丁文 1 |
| 852 | ibm852 / PC 拉丁文 2 |
| 855 | ibm855 / PC 斯拉夫文 |
| 856 | ibm856 |
| 857 | ibm857 / PC 土耳其文 |
| 860 | ibm860 / PC 葡萄牙文 |
| 861 | ibm861 / PC 冰島文 |
| 862 | ibm862 / PC 希伯來文 |
| 863 | ibm863 / PC 加拿大法文 |
| 864 | ibm864 / PC 阿拉伯文 |
| 865 | ibm865 / PC 斯堪地那維亞文 |
| 866 | ibm866 / PC 俄文 |
| 868 | ibm868 |
| 869 | ibm869 / PC 現代希臘文 |
| 870 | ibm870 |
| 871 | ibm871 |
| 874 | ibm874 |
| 875 | ibm875 |
| 918 | ibm918 |
| 921 | ibm921 |
| 922 | ibm922 |

表 13. 字集識別碼 (繼續)

| characterSet | 說明 |
|---|---|
| 930 | ibm930 |
| 933 | ibm933 |
| 935 | ibm935 |
| 937 | ibm937 |
| 939 | ibm939 |
| 942 | ibm942 |
| 948 | ibm948 |
| 949 | ibm949 |
| 950 | ibm950 / Big 5 繁體中文 |
| 964 | ibm964 / CNS 11643 繁體中文 |
| 970 | ibm970 |
| 1006 | ibm1006 |
| 1025 | ibm1025 |
| 1026 | ibm1026 |
| 1097 | ibm1097 |
| 1098 | ibm1098 |
| 1112 | ibm1112 |
| 1122 | ibm1122 |
| 1123 | ibm1123 |
| 1124 | ibm1124 |
| 1381 | ibm1381 |
| 1383 | ibm1383 |
| 2022 | JIS |
| 932 | PC 日文 |
| 954 | EUCJIS |
| 1250 | Windows 拉丁文 2 |
| 1251 | Windows 斯拉夫文 |
| 1252 | Windows 拉丁文 1 |
| 1253 | Windows 希臘文 |
| 1254 | Windows 土耳其文 |
| 1255 | Windows 希伯來文 |
| 1256 | Windows 阿拉伯文 |
| 1257 | Windows 波羅的海語系 |
| 1258 | Windows 越南文 |
| 33722 | ibm33722 |
| 5601 | ksc-5601 韓文 |
| 1200 | Unicode |
| 1208 | UTF-8 |

### format

```
public String format
```

訊息傳送端用來告訴接收端訊息中之資料性質的格式名稱。您可以使用自己的格式名稱，但開頭爲 ″MQ″ 字母的名稱由佇列管理程式來定義其意義。以下是佇列管理程式內建格式：

**MQC.MQFMT_NONE**
沒有格式名稱。

**MQC.MQFMT_ADMIN**
指令伺服器要求/回答訊息。

**MQC.MQFMT_COMMAND_1**
類型 1 指令回覆訊息。

**MQC.MQFMT_COMMAND_2**
類型 2 指令回覆訊息。

**MQC.MQFMT_DEAD_LETTER_HEADER**
無法傳送的郵件標題。

**MQC.MQFMT_EVENT**
事件訊息。

**MQC.MQFMT_PCF**
採用可程式指令格式的使用者定義訊息。

**MQC.MQFMT_STRING**
完全由字元組成的訊息。

**MQC.MQFMT_TRIGGER**
觸發程式訊息

**MQC.MQFMT_XMIT_Q_HEADER**
傳輸佇列標題。

預設值是 MQC.MQFMT_NONE。

### priority

```
public int priority
```

訊息優先順序。在外送訊息中，也可以指定特殊值 MQC.MQPRI_PRIORITY_AS_Q_DEF，這時候，訊息的優先順序會取自目的地佇列的預設優先順序屬性。

預設值是 MQC.MQPRI_PRIORITY_AS_Q_DEF。

### persistence

```
public int persistence
```

訊息持續性。定義的值如下：

- MQC.MQPER_PERSISTENT
- MQC.MQPER_NOT_PERSISTENT
- MQC.MQPER_PERSISTENCE_AS_Q_DEF

預設值是 MQC.MQPER_PERSISTENCE_AS_Q_DEF，表示訊息持續性應該取自目的地佇列的預設持續性屬性。

## MQMessage

### messageId

```
public byte messageId[]
```

如果是 MQQueue.get() 呼叫，這個欄位會指定要擷取之訊息的訊息識別碼。通常，佇列管理程式傳回的第一個訊息會含有符合所指定者的訊息識別碼及相互關係識別碼。特殊值 MQC.MQMI_NONE 可以符合*任何*訊息識別碼。

如果是 MQQueue.put() 呼叫，這會指定要使用的訊息識別碼。如果指定 MQC.MQMI_NONE 的話，佇列管理程式會在放置訊息時產生唯一的訊息識別碼。在放置之後，會更新這個成員變數的值來指示所用的訊息識別碼。

預設值是 MQC.MQMI_NONE。

### correlationId

```
public byte correlationId[]
```

如果是 MQQueue.get() 呼叫，這個欄位會指定要擷取之訊息的相互關係識別碼。通常，佇列管理程式傳回的第一個訊息會含有符合所指定者的訊息識別碼及相互關係識別碼。特殊值 MQC.MQCI_NONE 可以符合*任何*相互關係識別碼。

如果是 MQQueue.put() 呼叫，這會指定要使用的相互關係識別碼。

預設值是 MQC.MQCI_NONE。

### backoutCount

```
public int backoutCount
```

在工作單元中，MQQueue.get() 呼叫先前傳回訊息又加以回復的次數。

預設值是零。

### replyToQueueName

```
public String replyToQueueName
```

發出取得訊息之要求的應用程式應該將 MQC.MQMT_REPLY 和 MQC.MQMT_REPORT 訊息送往的訊息佇列名稱。

預設值是 ″″。

### replyToQueueManagerName

```
public String replyToQueueManagerName
```

回答或報告訊息應該送往的佇列管理程式名稱。

預設值是 ″″。

如果在 MQQueue.put() 呼叫中的值是 ″″，QueueManager 會填入這個值。

### userId

```
public String userId
```

訊息的身份環境定義的一部份；它會指出產生這則訊息的使用者。

預設值是 ″″。

### accountingToken

```
public byte accountingToken[]
```

訊息的身份環境定義的一部份；應用程式可以利用它，使工作隨著需要適當付費的訊息而完成。

預設值是 ″MQC.MQACT_NONE″。

**applicationIdData**

```
public String applicationIdData
```

訊息的身份環境定義的一部份；它是應用程式套件所定義的資訊，可用來提供訊息或其發訊者的其它相關資訊。

預設值是 ""。

**putApplicationType**

```
public int putApplicationType
```

放置訊息之應用程式的類型。這可能是系統定義的值，也可能是使用者定義的值。以下是系統所定義的值：

- MQC.MQAT_AIX
- MQC.MQAT_CICS
- MQC.MQAT_DOS
- MQC.MQAT_IMS
- MQC.MQAT_MVS
- MQC.MQAT_OS2
- MQC.MQAT_OS400
- MQC.MQAT_QMGR
- MQC.MQAT_UNIX
- MQC.MQAT_WINDOWS
- MQC.MQAT_JAVA

預設值是特殊值 MQC.MQAT_NO_CONTEXT，表示訊息中沒有環境定義資訊。

**putApplicationName**

```
public String putApplicationName
```

放置訊息之應用程式的名稱。預設值是 ""。

**putDateTime**

```
public GregorianCalendar putDateTime
```

放置訊息的日期和時間。

**applicationOriginData**

```
public String applicationOriginData
```

應用程式所定義、可用來提供訊息起源之其它相關資訊的資訊。

預設值是 ""。

**groupId**

```
public byte[] groupId
```

識別實際訊息所屬之訊息群組的位元組字串。

預設值是 "MQC.MQGI_NONE"。

**messageSequenceNumber**

```
public int messageSequenceNumber
```

群組內之邏輯訊息的序號。

**offset** `public int offset`

在分段的訊息中，資料在實體訊息中與邏輯訊息起點的偏移。

## MQMessage

### messageFlags

```
public int messageFlags
```

控制區段和訊息狀態的旗號。

### originalLength

```
public int originalLength
```

已分段之訊息的原始長度。

## 建構子

### MQMessage

```
public MQMessage()
```

用預設訊息描述子資訊和空的訊息緩衝區來建立新訊息。

## 方法

### getTotalMessageLength

```
public int getTotalMessageLength()
```

從中擷取（或試圖擷取）這個訊息的訊息佇列其中所儲存之訊息的位元組總數。當 MQQueue.get() 方法失敗而出現訊息截斷錯誤碼時，這個方法會告訴您佇列中的訊息總大小。

另請參閱第131頁的『MQQueue.get』。

### getMessageLength

```
public int getMessageLength
```

擲出 IOException。

這個 MQMessage 物件中的訊息資料位元組數。

### getDataLength

```
public int getDataLength()
```

擲出 MQException。

剩餘要讀取的訊息資料之位元組數目。

### seek

```
public void seek(int pos)
```

擲出 IOException。

將游標移到 *pos* 所提供的訊息緩衝區中之絕對位置。後續的讀取和寫入會作用於緩衝區中的這個位置。

如果 pos 在訊息資料長度之外，會擲出 EOFException。

### setDataOffset

```
public void setDataOffset(int offset)
```

擲出 IOException。

將游標移到訊息緩衝區中的絕對位置。這個方法是 seek() 的同義字，專供與其它 MQSeries API 的跨語言相容性之用。

### getDataOffset

```
public int getDataOffset()
```

擲出 IOException。

傳回訊息資料內的現行游標位置（讀取和寫入作業生效的點）。

### clearMessage

```
public void clearMessage()
```

擲出 IOException。

捨棄訊息緩衝區中的任何資料，將資料偏移重設爲零。

### getVersion

```
public int getVersion()
```

傳回使用中之結構的版本。

### resizeBuffer

```
public void resizeBuffer(int size)
```

擲出 IOException。

提供給 MQMessage 物件有關後續 get 作業所需要之緩衝區大小的提示。如果訊息目前含有訊息資料，且新大小小於現行大小，這時會截斷訊息資料。

### readBoolean

```
public boolean readBoolean()
```

擲出 IOException。

從訊息緩衝區中的現行位置讀取一個（帶正負號）位元組。

### readChar

```
public char readChar()
```

擲出 IOException、EOFException。

從訊息緩衝區中的現行位置讀取一個 Unicode 字元。

### readDouble

```
public double readDouble()
```

擲出 IOException、EOFException。

從訊息緩衝區中的現行位置讀取一個倍整數。 encoding 成員變數的值決定了這個方法的行爲。

MQC.MQENC_FLOAT_IEEE_NORMAL 和 MQC.MQENC_FLOAT_IEEE_REVERSED 的值分別會讀取大序排列和小序排列格式的 IEEE 標準浮點數。

MQC.MQENC_FLOAT_S390 值會讀取 System/390 格式浮點數。

### readFloat

```
public float readFloat()
```

擲出 IOException、EOFException。

從訊息緩衝區中的現行位置讀取一個浮點數。 encoding 成員變數的值決定了這個方法的行爲。

MQC.MQENC_FLOAT_IEEE_NORMAL 和 MQC.MQENC_FLOAT_IEEE_REVERSED 的值分別會讀取大序排列和小序排列格式的 IEEE 標準浮點數。

MQC.MQENC_FLOAT_S390 值會讀取 System/390 格式浮點數。

### readFully

```
public void readFully(byte b[])
```

擲出 Exception、 EOFException。

以訊息緩衝區中的資料填入位元組陣列 b。

### readFully

```
public void readFully(byte b[],
                      int off,
                      int len)
```

擲出 IOException、EOFException。

從 *off* 偏移開始，以訊息緩衝區中的資料填入位元組陣列 b 的 *len* 元素。

### readInt

```
public int readInt()
```

擲出 IOException、EOFException。

從訊息緩衝區中的現行位置讀取一個整數。 encoding 成員變數的值決定了這個方法的行爲。

MQC.MQENC_INTEGER_NORMAL 值會讀取大序排列法整數，MQC.MQENC_INTEGER_REVERSED 值會讀取小序排列法整數。

### readInt4

```
public int readInt4()
```

擲出 IOException、EOFException。

readInt() 的同義字，專供跨語言 MQSeries API 相容性之用。

### readLine

```
public String readLine()
```

擲出 IOException。

將 characterSet 成員變數中所識別的字碼集轉換成 Unicode，之後，再讀取以 \n、\r、\r\n 或 EOF 爲終止行。

### readLong

```
public long readLong()
```

擲出 IOException、EOFException。

從訊息緩衝區中的現行位置讀取一個長整數。 encoding 成員變數的值決定了這個方法的行爲。

MQC.MQENC_INTEGER_NORMAL 值會讀取大序排列法長整數，MQC.MQENC_INTEGER_REVERSED 值會讀取小序排列法長整數。

### readInt8

```
public long readInt8()
```

擲出 IOException、EOFException。

readLong() 的同義字，專供跨語言 MQSeries API 相容性之用。

### readObject

```
public Object readObject()
```

擲出 OptionalDataException、 ClassNotFoundException、IOException。

從訊息緩衝區中讀取一個物件。這會讀取物件類別、類別簽章及類別的非暫時和非靜態欄位值。

### readShort

```
public short readShort()
```

擲出 IOException、EOFException。

### readInt2

```
public short readInt2()
```

擲出 IOException、EOFException。

readShort() 的同義字，專供跨語言 MQSeries API 相容性之用。

### readUTF

```
public String readUTF()
```

擲出 IOException。

從訊息緩衝區中的現行位置讀取一個 UTF 字串，字首爲 2 位元組長度欄位。

### readUnsignedByte

```
public int readUnsignedByte()
```

擲出 IOException、EOFException。

從訊息緩衝區中的現行位置讀取一個無正負號位元組。

### readUnsignedShort

```
public int readUnsignedShort()
```

擲出 IOException、EOFException。

從訊息緩衝區中的現行位置讀取一個無正負號短整數。 encoding 成員變數的值決定了這個方法的行爲。

MQC.MQENC_INTEGER_NORMAL 值會讀取大序排列法無正負號短整數，MQC.MQENC_INTEGER_REVERSED 值會讀取小序排列法無正負號短整數。

### readUInt2

```
public int readUInt2()
```

擲出 IOException、EOFException。

readUnsignedShort() 的同義字，專供跨語言 MQSeries API 相容性之用。

### readString

```
public String readString(int length)
```

擲出 IOException、EOFException。

以 characterSet 成員變數所識別的字碼集讀取一個字串，再將它轉換成 Unicode。

**參數：**

*length* 要讀取的字元數（根據字碼集，可能會不同於位元組數，因爲有些字碼集每個字元使用不只一個位元組）。

### readDecimal2

```
public short readDecimal2()
```

擲出 IOException、EOFException。

讀取 2 位元組聚集十進位數 (-999..999)。這個方法的行爲由 encoding 成員變數的值來控制。 MQC.MQENC_DECIMAL_NORMAL 值會讀取大序排列法聚集十進位數，MQC.MQENC_DECIMAL_REVERSED 值會讀取小序排列法聚集十進位數。

### readDecimal4

```
public int readDecimal4()
```

擲出 IOException、EOFException。

讀取 4 位元組聚集十進位數 (-9999999..9999999)。這個方法的行爲由 encoding 成員變數的值來控制。 MQC.MQENC_DECIMAL_NORMAL 值會讀取大序排列法聚集十進位數，MQC.MQENC_DECIMAL_REVERSED 值會讀取小序排列法聚集十進位數。

### readDecimal8

```
public long readDecimal8()
```

擲出 IOException、EOFException。

讀取 8 位元組聚集十進位數 (-999999999999999 至 999999999999999)。這個方法的行爲由 encoding 成員變數來控制。 MQC.MQENC_DECIMAL_NORMAL 值會讀取大序排列法聚集十進位數，MQC.MQENC_DECIMAL_REVERSED 會讀取小序排列法聚集十進位數。

### setVersion

```
public void setVersion(int version)
```

指定要使用結構的哪個版本。可能的值如下：

- MQC.MQMD_VERSION_1
- MQC.MQMD_VERSION_2

除非在連接到能夠處理第 2 版結構的佇列管理程式時，您要強迫從屬站使用第 1 版結構，否則，您通常應該不需要呼叫這個方法。在所有其它情況中，從屬站會查詢佇列管理程式的功能來判斷要使用的正確結構版本。

### skipBytes

```
public int skipBytes(int n)
```

擲出 IOException、EOFException。

在訊息緩衝區中，向前移動 n 位元組。

這個方法會暫時停用，直到出現下列中的情況爲止：
- 略過所有位元組
- 偵測到訊息緩衝區尾端
- 擲出異常狀況

傳回略過的位元組數目，它永遠是 n。

### write

```
public void write(int b)
```

擲出 IOException。

將一個位元組寫入訊息緩衝區的現行位置。

### write

```
public void write(byte b[])
```

擲出 IOException。

將一個位元組陣列寫入訊息緩衝區的現行位置。

### write

```
public void write(byte b[],
                  int off,
                  int len)
```

擲出 IOException。

將一個位元組序列寫入訊息緩衝區的現行位置。從 b 陣列 *off* 偏移中取出，而寫入 *len* 位元組。

### writeBoolean

```
public void writeBoolean(boolean v)
```

擲出 IOException。

將一個 Boolean 寫入訊息緩衝區的現行位置。

### writeByte

```
public void writeByte(int v)
```

擲出 IOException。

將一個位元組寫入訊息緩衝區的現行位置。

### writeBytes

```
public void writeBytes(String s)
```

擲出 IOException。

將一個字串當作位元組序列寫到訊息緩衝區中。字串中的每個字元都會藉由捨棄其高址的八位元而循序寫出。

### writeChar

```
public void writeChar(int v)
```

擲出 IOException。

將一個 Unicode 字元寫入訊息緩衝區的現行位置。

### writeChars

```
public void writeChars(String s)
```

擲出 IOException。

將一個字串當作 Unicode 字元序列寫入訊息緩衝區的現行位置。

### writeDouble

```
public void writeDouble(double v)
```

擲出 IOException

將一個倍整數寫入訊息緩衝區的現行位置。 encoding 成員變數的值決定了這個方法的行爲。

MQC.MQENC_FLOAT_IEEE_NORMAL 和
MQC.MQENC_FLOAT_IEEE_REVERSED 的值分別會寫入大序排列和小序排列格式的 IEEE 標準浮點數。

MQC.MQENC_FLOAT_S390 值會寫入 System/390 格式浮點數。請注意，IEEE Double 的範圍大於 S/390® 倍精準浮點數，因此，無法轉換非常大的數字。

### writeFloat

```
public void writeFloat(float v)
```

擲出 IOException。

將一個浮點數寫入訊息緩衝區的現行位置。 encoding 成員變數的值決定了這個方法的行爲。

MQC.MQENC_FLOAT_IEEE_NORMAL 和
MQC.MQENC_FLOAT_IEEE_REVERSED 的值分別會寫入大序排列和小序排列格式的 IEEE 標準浮點數。

MQC.MQENC_FLOAT_S390 值會寫入 System/390 格式浮點數。

### writeInt

```
public void writeInt(int v)
```

擲出 IOException。

將一個整數寫入訊息緩衝區的現行位置。 encoding 成員變數的值決定了這個方法的行爲。

MQC.MQENC_INTEGER_NORMAL 值會寫入大序排列法整數，MQC.MQENC_INTEGER_REVERSED 值會寫入小序排列法整數。

### writeInt4

```
public void writeInt4(int v)
```

擲出 IOException。

writeInt() 的同義字，專供跨語言 MQSeries API 相容性之用。

### writeLong

```
public void writeLong(long v)
```

擲出 IOException。

將一個長整數寫入訊息緩衝區的現行位置。 encoding 成員變數的值決定了這個方法的行爲。

MQC.MQENC_INTEGER_NORMAL 值會寫入大序排列法長整數，MQC.MQENC_INTEGER_REVERSED 值會寫入小序排列法長整數。

### writeInt8

```
public void writeInt8(long v)
```

擲出 IOException。

writeLong() 的同義字，專供跨語言 MQSeries API 相容性之用。

### writeObject

```
public void writeObject(Object obj)
```

擲出 IOException。

將指定的物件寫入訊息緩衝區中。這會寫入物件類別、類別簽章及類別的非暫時和非靜態欄位值及其所有子類型。

### writeShort

```
public void writeShort(int v)
```

擲出 IOException。

將一個短整數寫入訊息緩衝區的現行位置。 encoding 成員變數的值決定了這個方法的行爲。

MQC.MQENC_INTEGER_NORMAL 值會寫入大序排列法短整數，MQC.MQENC_INTEGER_REVERSED 值會寫入小序排列法短整數。

### writeInt2

```
public void writeInt2(int v)
```

擲出 IOException。

writeShort() 的同義字，專供跨語言 MQSeries API 相容性之用。

### writeDecimal2

```
public void writeDecimal2(short v)
```

擲出 IOException。

將一個 2 位元組的聚集十進位格式數字寫入訊息緩衝區的現行位置。 encoding 成員變數的值決定了這個方法的行爲。

MQC.MQENC_DECIMAL_NORMAL 值會寫入大序排列法聚集十進位數，MQC.MQENC_DECIMAL_REVERSED 值會寫入小序排列法聚集十進位數。

#### 參數

*v* 可在 -999 至 999 的範圍內。

### writeDecimal4

```
public void writeDecimal4(int v)
```

擲出 IOException。

將一個 4 位元組的聚集十進位格式數字寫入訊息緩衝區的現行位置。 encoding 成員變數的值決定了這個方法的行爲。

MQC.MQENC_DECIMAL_NORMAL 值會寫入大序排列法聚集十進位數，MQC.MQENC_DECIMAL_REVERSED 值會寫入小序排列法聚集十進位數。

#### 參數

*v* 可在 -9999999 至 9999999 的範圍內。

### writeDecimal8

```
public void writeDecimal8(long v)
```

擲出 IOException。

將一個 8 位元組的聚集十進位格式數字寫入訊息緩衝區的現行位置。 encoding 成員變數的值決定了這個方法的行爲。

MQC.MQENC_DECIMAL_NORMAL 值會寫入大序排列法聚集十進位數，MQC.MQENC_DECIMAL_REVERSED 值會寫入小序排列法聚集十進位數。

**參數：**

*v* 可在 -999999999999999 至 999999999999999 的範圍內。

### writeUTF

```
public void writeUTF(String str)
```

擲出 IOException。

在訊息緩衝區的現行位置中寫入一個 UTF 字串，字首爲 2 位元組長度欄位。

### writeString

```
public void writeString(String str)
```

擲出 IOException。

將一個字串寫入訊息緩衝區的現行位置，將它轉換成 characterSet 成員變數所識別的字集。

# MQMessageTracker

```
java.lang.Object
   |
   +-- com.ibm.mq.MQMessageTracker
```

public abstract class **MQMessageTracker**
extends **Object**

**註:** 只有在連接到 MQSeries 第 5 版（或以上）佇列管理程式時，才可以使用這個類別。

MQDistributionListItem（頁 85）繼承這個類別，利用它來調整分送清單中的給定目的地之訊息參數。

## 變數

### feedback

```
public int feedback
```

這用來搭配 MQC.MQMT_REPORT 類型的訊息指出報告的性質。以下是系統所定義的傳回訊息碼：

- MQC.MQFB_EXPIRATION
- MQC.MQFB_COA
- MQC.MQFB_COD
- MQC.MQFB_QUIT
- MQC.MQFB_PAN
- MQC.MQFB_NAN
- MQC.MQFB_DATA_LENGTH_ZERO
- MQC.MQFB_DATA_LENGTH_NEGATIVE
- MQC.MQFB_DATA_LENGTH_TOO_BIG
- MQC.MQFB_BUFFER_OVERFLOW
- MQC.MQFB_LENGTH_OFF_BY_ONE
- MQC.MQFB_IIH_ERROR

應用程式所定義、在 MQC.MQFB_APPL_FIRST 至 MQC.MQFB_APPL_LAST 範圍的傳回訊息也可以使用。

這個欄位的預設值是 MQC.MQFB_NONE，表示不提供任何傳回的訊息。

### messageId

```
public byte messageId[]
```

這指定放置訊息時所用的訊息識別碼。如果指定 MQC.MQMI_NONE 的話，佇列管理程式會在放置訊息時產生唯一的訊息識別碼。在放置之後，會更新這個成員變數的值來指示所用的訊息識別碼。

預設值是 MQC.MQMI_NONE。

## MQMessageTracker

### correlationId

```
public byte correlationId[]
```

這指定放置訊息時所用的交互識別碼。

預設值是 MQC.MQCI_NONE。

### accountingToken

```
public byte accountingToken[]
```

這是訊息的身份環境定義的一部份。應用程式可以利用它，使工作隨著需要適當付費的訊息而完成。

預設值是 MQC.MQACT_NONE。

### groupId

```
public byte[] groupId
```

識別實際訊息所屬之訊息群組的位元組字串。

預設值是 MQC.MQGI_NONE。

# MQPoolServices

```
java.lang.Object
    |
    └ com.ibm.mq.MQPoolServices
```

public class **MQPoolServices**
extends **Object**

**註:** 應用程式通常不會用到這個類別。

要作爲 MQSeries 連線之預設 ConnectionManager 的 ConnectionManager 實作可以使用 MQPoolService 類別。

ConnectionManager 可以建構一個 MQPoolService 物件，並透過它來登錄接聽器。這個接聽器會接收 MQEnvironment 所管理的 MQPoolToken 集的相關事件。ConnectionManager 可以利用這項資訊來執行任何必要的啓動或清除工作。

另請參閱第122頁的『MQPoolServicesEvent』和
第151頁的『MQPoolServicesEventListener』。

## 建構子

### MQPoolServices

```
public MQPoolServices()
```

建構一個新的 MQPoolServices 物件。

## 方法

### addMQPoolServicesEventListener

```
public void addMQPoolServicesEventListener
                                (MQPoolServicesEventListener listener)
```

新增一個 MQPoolServicesEventListener。每當從 MQEnvironment 所控制的 MQPoolToken 集中新增或移除記號時，或每當預設 ConnectionManager 改變時，接聽器都會收到一則事件。

### removeMQPoolServicesEventListener

```
public void removeMQPoolServicesEventListener
                                  (MQPoolServicesEventListener listener)
```

移除一個 MQPoolServicesEventListener。

### getTokenCount

```
public int getTokenCount()
```

傳回 MQEnvironment 目前所登錄的 MQPoolToken 數目。

# MQPoolServicesEvent

```
java.lang.Object
   |
   └ java.util.EventObject
           |
           └ com.ibm.mq.MQPoolServicesEvent
```

**註:** 應用程式通常不會用到這個類別。

每次在 MQEnvironment 所控制的記號集中新增或移除 MQPoolToken 時，都會產生一個 MQPoolServicesEvent。當變更預設 ConnectionManager 時，也會產生一則事件。

另請參閱第121頁的『MQPoolServices』和
第151頁的『MQPoolServicesEventListener』。

## 變數

### TOKEN_ADDED

```
public static final int TOKEN_ADDED
```

當 MQPoolToken 新增到 MQPoolToken 集時所用的事件 ID。

### TOKEN_REMOVED

```
public static final int TOKEN_REMOVED
```

從 MQPoolToken 集中移除 MQPoolToken 時所用的事件 ID。

### DEFAULT_POOL_CHANGED

```
public static final int DEFAULT_POOL_CHANGED
```

預設 ConnectionManager 變更時所用的事件 ID。

### ID

```
protected int ID
```

事件 ID。有效值如下：

TOKEN_ADDED

TOKEN_REMOVED

DEFAULT_POOL_CHANGED

### token

```
protected MQPoolToken token
```

記號。當事件 ID 是 DEFAULT_POOL_CHANGED 時，這是空值。

## 建構子

### MQPoolServicesEvent

```
public MQPoolServicesEvent(Object source, int eid, MQPoolToken token)
```

根據事件 ID 和記號建構一個 MQPoolServicesEvent。

### MQPoolServicesEvent

```
public MQPoolServicesEvent(Object source, int eid)
```

根據事件 ID 建構一個 MQPoolServicesEvent。

## 方法

**getId** `public int getId()`

取得事件 ID。

**傳回**

含有下列值之一的事件 ID：

TOKEN_ADDED

TOKEN_REMOVED

DEFAULT_POOL_CHANGED

**getToken**

`public MQPoolToken getToken()`

傳回在集中新增或移除的記號。如果事件 ID 是 DEFAULT_POOL_CHANGED，這便是空值。

# MQPoolToken

```
java.lang.Object
   |
   +-- com.ibm.mq.MQPoolToken
```

public class **MQPoolToken**
extends **Object**

MQPoolToken 可用來啓用預設的連線儲存池。在應用程式元件連接到 MQSeries 之前，MQPoolToken 要先在 MQEnvironment 類別中進行登錄。之後，當元件用完 MQSeries 時，它們再取消登錄。通常當登錄的 MQPoolToken 集不是空的，預設的 ConnectionManager 會在作用中。

MQPoolToken 不提供任何方法或變數。 ConnectionManager 提供者可選擇繼承 MQPoolToken，使提示能夠傳遞給 ConnectionManager。

請參閱第90頁的『MQEnvironment.addConnectionPoolToken』和第90頁的『MQEnvironment.removeConnectionPoolToken』。

## 建構子

### MQPoolToken

```
public MQPoolToken()
```

建構一個新的 MQPoolToken 物件。

# MQProcess

```
java.lang.Object
   |
   +-- com.ibm.mq.MQManagedObject
          |
          +-- com.ibm.mq.MQProcess
```

public class **MQProcess**
extends **MQManagedObject**（請參閱頁 97。）

MQProcess 提供有 MQSeries 程序的查詢作業。

## 建構子

### MQProcess

```
public MQProcess(MQQueueManager qMgr,
                 String processName,
                 int openOptions,
                 String queueManagerName,
                 String alternateUserId)
          throws MQException
```

存取 qMgr 佇列管理程式中的程序。請參閱第138頁的『MQQueueManager』 中的 accessProcess，以取得其餘參數的詳細資料。

## 方法

### getApplicationId

```
public String getApplicationId()
```

指出要啓動之應用程式的字串。這項資訊專供處理起始佇列中之訊息的觸發監視器應用程式使用；這項資訊會作爲觸發訊息的一部份而傳送到起始佇列中。

如果您關閉程序之後呼叫這個方法，便會擲出 MQException。

### getApplicationType

```
public int getApplicationType()
```

擲出 MQException（請參閱 91 頁）。

這會指出要啓動以回應收到的觸發訊息之程式的性質。應用程式類型可採用任何值，但標準類型建議採用下列各值：

- MQC.MQAT_AIX
- MQC.MQAT_CICS
- MQC.MQAT_DOS
- MQC.MQAT_IMS
- MQC.MQAT_MVS
- MQC.MQAT_OS2
- MQC.MQAT_OS400
- MQC.MQAT_UNIX
- MQC.MQAT_WINDOWS
- MQC.MQAT_WINDOWS_NT
- MQC.MWQAT_USER_FIRST（使用者定義應用程式類型的最低值）
- MQC.MQAT_USER_LAST（使用者定義應用程式類型的最高值）

## MQProcess

### getEnvironmentData

```
public String getEnvironmentData()
```

擲出 MQException。

含有附屬於要啓動之應用程式的環境相關資訊的字串。

### getUserData

```
public String getUserData()
```

擲出 MQException。

含有要啓動之應用程式的相關使用者資訊的字串。

### close

```
public synchronized void close()
```

擲出 MQException。

置換 第99頁的『MQManagedObject.close』。

# MQPutMessageOptions

```
java.lang.Object
   |
   +-- com.ibm.mq.MQPutMessageOptions
```

public class **MQPutMessageOptions**
extends **Object**

這個類別含有控制 MQQueue.put() 行爲的選項。

**註:** 這個類別中某些可用選項的行爲會隨著使用它們的環境而不同。這些元素會有標出 *****。請參閱第74頁的『在其它環境中運作的第 5 版延伸規格』，以取得詳細資訊。

## 變數

### options

```
public int options
```

控制 MQQueue.put 動作的選項。您可以指定下列中的任何值，或不指定下列中的任何值。如果需要多個選項，您可以利用按位元的 OR 運算子，將這些值加起來或組合起來。

**MQC.MQPMO_SYNCPOINT**
放置含同步點控制的訊息。在確定工作單元之前，無法在工作單元之外見到訊息。如果回復工作單元的話，會刪除訊息。

**MQC.MQPMO_NO_SYNCPOINT**
放置不含同步點控制的訊息。請注意，如果沒有指定同步點控制的話，會採用預設值 'no syncpoint'。這適用於所有支援的平台，其中包括 OS/390。

**MQC.MQPMO_NO_CONTEXT**
這個訊息沒有關聯於任何環境定義。

**MQC.MQPMO_DEFAULT_CONTEXT**
建立預設環境定義和訊息的關聯。

**MQC.MQPMO_SET_IDENTITY_CONTEXT**
從應用程式設定身份識別環境定義。

**MQC.MQPMO_SET_ALL_CONTEXT**
從應用程式設定所有環境定義。

**MQC.MQPMO_FAIL_IF_QUIESCING**
如果佇列管理程式在靜止中，便告失敗。

**MQC.MQPMO_NEW_MSG_ID***
就每個傳送的訊息而產生一個新的訊息 ID。

**MQC.MQPMO_NEW_CORREL_ID***
就每個傳送的訊息而產生一個新的相互關係 ID。

**MQC.MQPMO_LOGICAL_ORDER***
將訊息群組中的邏輯訊息和區段放入它們的邏輯次序中。

**MQC.MQPMO_NONE**
沒有指定任何選項。請勿結合其它選項。

**MQC.MQPMO_PASS_IDENTITY_CONTEXT**
從輸入佇列控點中傳遞身份識別環境定義。

**MQC.MQPMO_PASS_ALL_CONTEXT**
從輸入佇列控點中傳遞所有環境定義。

**contextReference**

```
public MQQueue ContextReference
```

這是一個指出環境定義資訊的輸入欄位。

如果選項欄位包含 MQC.MQPMO_PASS_IDENTITY_CONTEXT 或 MQC.MQPMO_PASS_ALL_CONTEXT，請設定這個欄位來參照應該是環境定義資訊來源的 MQQueue。

這個欄位的起始值是空值。

**recordFields ***

```
public int recordFields
```

指出在訊息放入分送清單時有哪些欄位是每個佇列都要自訂的旗號。您可以指定下列中的一或多個旗號：

**MQC.MQPMRF_MSG_ID**
請使用 MQDistributionListItem 中的 messageId 屬性。

**MQC.MQPMRF_CORREL_ID**
請使用 MQDistributionListItem 中的 correlationId 屬性。

**MQC.MQPMRF_GROUP_ID**
請使用 MQDistributionListItem 中的 groupId 屬性。

**MQC.MQPMRF_FEEDBACK**
請使用 MQDistributionListItem 中的 feedback 屬性。

**MQC.MQPMRF_ACCOUNTING_TOKEN**
請使用 MQDistributionListItem 中的 accountingToken 屬性。

特殊值 MQC.MQPMRF_NONE 指出沒有要自訂的欄位。

**resolvedQueueName**

```
public String resolvedQueueName
```

這是一個輸出欄位，由佇列管理程式將它設成訊息放置其中的佇列名稱。如果開啓的佇列是別名或模型佇列，這可能會不同於開啓佇列時所用的名稱。

**resolvedQueueManagerName**

```
public String resolvedQueueManagerName
```

這是一個輸出欄位，由佇列管理程式將它設成擁有遠端佇列名稱指定的佇列之佇列管理程式的名稱。如果佇列是遠端佇列，這可能會不同於處理佇列之佇列管理程式的名稱。

**knownDestCount ***

```
public int knownDestCount
```

這是一個輸出欄位，由佇列管理程式將它設成現行呼叫已順利傳送至解析成本端佇列之佇列的訊息數目。當開啓不在分送清單中的單一佇列時，也會設定這個欄位。

**unknownDestCount ***

```
public int unknownDestCount
```

這是一個輸出欄位，由佇列管理程式將它設成現行呼叫已順利傳送至解析成遠端佇列之佇列的訊息數目。當開啓不在分送清單中的單一佇列時，也會設定這個欄位。

**invalidDestCount ***

```
public int invalidDestCount
```

這是一個輸出欄位，由佇列管理程式將它設成無法傳送至分送清單中之佇列的訊息數目。計數中包括無法開啓的佇列及順利開啓但放置作業失敗的佇列。當開啓不在分送清單中的單一佇列時，也會設定這個欄位。

## 建構子

**MQPutMessageOptions**

```
public MQPutMessageOptions()
```

建構一個不含選項集的新 MQPutMessageOptions 物件，以及一個空白的 resolvedQueueName 和 resolvedQueueManagerName。

# MQQueue

```
java.lang.Object
   |
   +-- com.ibm.mq.MQManagedObject
          |
          +-- com.ibm.mq.MQQueue
```

public class **MQQueue**
extends **MQManagedObject**（請參閱頁 97。）

MQQueue 提供 MQSeries 佇列的 inquire、set、put 和 get 等作業。 inquire 和 set 功能繼承自 MQ.MQManagedObject。

另請參閱第143頁的『MQQueueManager.accessQueue』。

## 建構子

### MQQueue

```
public MQQueue(MQQueueManager qMgr, String queueName, int openOptions,
               String queueManagerName, String dynamicQueueName,
               String alternateUserId )
               throws MQException
```

存取 qMgr 佇列管理程式中的佇列。

請參閱第143頁的『MQQueueManager.accessQueue』，以取得其餘參數的詳細資訊。

## 方法

### get

```
public synchronized void get(MQMessage message,
                             MQGetMessageOptions getMessageOptions,
                             int MaxMsgSize)
```

擲出 MQException。

從佇列中擷取一則訊息，不得超出指定的訊息大小上限。

這個方法會將 MQMessage 物件當作參數。它會利用物件中的某些欄位作爲輸入參數 - 尤其是 messageId 和 correlationId，因此，請務必確定這些的設定都符合需要。（請參閱第258頁的『Message』。）

如果 get 失敗，MQMessage 物件不會改變。如果順利成功的話，訊息描述子（成員變數）和 MQMessage 的訊息資料部份會完全改成內收訊息的訊息描述子及訊息資料。

請注意，並非給定的 MQQueueManager 所發出的 MQSeries 呼叫都是同步的。因此，如果您執行 get with wait，所有使用相同 MQQueueManager 的其它執行緒都會暫時無法再呼叫 MQSeries，直到 get 完成爲止。如果您需要多個執行緒來同時存取 MQSeries，每個執行緒都必須建立它自己的 MQQueueManager 物件。

### 參數

*message*
含有訊息描述子資訊及傳回的訊息資料之輸入/輸出參數。

*getMessageOptions*
控制 get 動作的選項。（請參閱第93頁的『MQGetMessageOptions』。）

*MaxMsgSize*
這個呼叫能夠接收的最大訊息。如果佇列中的訊息超出這個大小，會發生下兩種情況的其中一種：

1. 如果 MQGetMessageOption 物件的選項成員變數設了 MQC.MQGMO_ACCEPT_TRUNCATED_MSG 旗號，訊息會填入指定緩衝區大小範圍內的訊息資料，且會擲出完成碼爲 MQException.MQCC_WARNING、原因碼爲 MQException.MQRC_TRUNCATED_MSG_ACCEPTED 的異常狀況。
2. 如果沒有設定 MQC.MQGMO_ACCEPT_TRUNCATED_MSG 旗號的話，訊息會留在佇列中，且會產生一則 MQException，完成碼爲 MQException.MQCC_WARNING、原因碼爲 MQException.MQRC_TRUNCATED_MSG_FAILED。

如果 get 失敗的話，會擲出 MQException。

## get

```
public synchronized void get(MQMessage message,
                             MQGetMessageOptions getMessageOptions)
```

擲出 MQException。

從佇列中擷取一則訊息，而不論訊息的大小爲何。如果是大型訊息，get 方法可能需要替您發出兩個 MQSeries 呼叫，一個用來建立必要的緩衝區大小，另一個用來取得訊息資料本身。

這個方法會將 MQMessage 物件當作參數。它會利用物件中的某些欄位作爲輸入參數 - 尤其是 messageId 和 correlationId，因此，請務必確定這些的設定都符合需要。（請參閱第258頁的『Message』。）

如果 get 失敗，MQMessage 物件不會改變。如果順利成功的話，訊息描述子（成員變數）和 MQMessage 的訊息資料部份會完全改成內收訊息的訊息描述子及訊息資料。

請注意，並非給定的 MQQueueManager 所發出的 MQSeries 呼叫都是同步的。因此，如果您執行 get with wait，所有使用相同 MQQueueManager 的其它執行緒都會暫時無法再呼叫 MQSeries，直到 get 完成爲止。如果您需要多個執行緒來同時存取 MQSeries，每個執行緒都必須建立它自己的 MQQueueManager 物件。

### 參數

*message*
含有訊息描述子資訊及傳回的訊息資料之輸入/輸出參數。

*getMessageOptions*
控制 get 動作的選項。（請參閱第93頁的『MQGetMessageOptions』，以取得詳細資料。）

如果 get 失敗的話，會擲出 MQException。

## get

```
public synchronized void get(MQMessage message)
```

這是先前所說明之 get 方法的簡化版本。

### 參數

*MQMessage*
含有訊息描述子資訊及傳回的訊息資料之輸入/輸出參數。

這個方法會使用 MQGetMessageOptions 的預設案例來執行 get。所用的訊息選項是 MQGMO_NOWAIT。

## put

```
public synchronized void put(MQMessage message,
              MQPutMessageOptions putMessageOptions )
```

擲出 MQException。

將訊息放入佇列中。

這個方法會將 MQMessage 物件當作參數。這個物件的訊息描述子內容可能會因這個方法的結果而改變。它們在這個方法剛完成時所擁有的值，是放入 MQSeries 佇列的值。

在 put 完成之後修改 MQMessage 物件不會影響 MQSeries 佇列中的實際訊息。

put 會更新 messageId 和 correlationId。當您要利用相同的 MQMessage 物件來進一步呼叫 put/get 時，必須考慮這一點。此外，呼叫 put 不會清除訊息資料，因此：

```
msg.writeString("a");
q.put(msg,pmo);
msg.writeString("b");
q.put(msg,pmo);
```

會放置兩個訊息。第一個含有 ″a″，第二個含有 ″ab″。

**參數**

*message*
含有訊息描述子資料和要傳送的訊息的訊息緩衝區。

*putMessageOptions*
控制 put 動作的選項。（請參閱第127頁的『MQPutMessageOptions』）

如果 put 失敗的話，會擲出 MQException。

### put

```
public synchronized void put(MQMessage message)
```

這是先前所說明之 put 方法的簡化版本。

**參數**

*MQMessage*
含有訊息描述子資料和要傳送的訊息的訊息緩衝區。

這個方法會使用 MQPutMessageOption 的預設案例來執行 put。

**註:** 如果您在關閉佇列之後呼叫這個方法，所有下列方法都會擲出 MQException。

### getCreationDateTime

```
public GregorianCalendar getCreationDateTime()
```

擲出 MQException。

這個佇列的建立日期和時間。

### getQueueType

```
public int getQueueType()
```

擲出 MQException

**傳回** 含有下列其中一值的這個佇列的類型：
- MQC.MQQT_ALIAS
- MQC.MQQT_LOCAL
- MQC.MQQT_REMOTE
- MQC.MQQT_CLUSTER

### getCurrentDepth

```
public int getCurrentDepth()
```

擲出 MQException。

取得目前在佇列中的訊息數目。在 put 呼叫期間及在 get 呼叫的回復期間，這個值會增加。在非瀏覽的 get 期間及 put 呼叫的回復期間，這個值會減少。

### getDefinitionType

```
public int getDefinitionType()
```

擲出 MQException。

指出如何定義佇列。

**傳回** 下列其中一項：

- MQC.MQQDT_PREDEFINED
- MQC.MQQDT_PERMANENT_DYNAMIC
- MQC.MQQDT_TEMPORARY_DYNAMIC

### getMaximumDepth

```
public int getMaximumDepth()
```

擲出 MQException。

在任何時候，佇列中所能存在的訊息數目上限。試圖將訊息放入已有這麼多訊息的佇列中會失敗，並出現原因碼 MQException.MQRC_Q_FULL。

### getMaximumMessageLength

```
public int getMaximumMessageLength()
```

擲出 MQException。

這是這個佇列上的每個訊息中所能有的應用程式資料長度上限。試圖放置長度超出這個值的訊息會失敗，並出現原因碼 MQException.MQRC_MSG_TOO_BIG_FOR_Q。

### getOpenInputCount

```
public int getOpenInputCount()
```

擲出 MQException。

目前從佇列中移除訊息的有效控點數。這是本端佇列管理程式所識別的這些控點的總數，而不是 MQSeries Class for Java 所建立的控點（利用 accessQueue）。

### getOpenOutputCount

```
public int getOpenOutputCount()
```

擲出 MQException。

目前在佇列中新增訊息的有效控點數。這是本端佇列管理程式所識別的這些控點的總數，而不是 MQSeries Class for Java 所建立的控點（利用 accessQueue）。

### getShareability

```
public int getShareability()
```

擲出 MQException。

指出是否能開啓佇列來進行多次的輸入。

**傳回** 下列其中一項：
- MQC.MQQA_SHAREABLE
- MQC.MQQA_NOT_SHAREABLE

### getInhibitPut

```
public int getInhibitPut()
```

擲出 MQException。

指出這個佇列是否能進行 put 作業。

**傳回** 下列其中一項：
- MQC.MQQA_PUT_INHIBITED
- MQC.MQQA_PUT_ALLOWED

### setInhibitPut

```
public void setInhibitPut(int inhibit)
```

擲出 MQException。

控制這個佇列是否能進行 put 作業。可用的值如下：
- MQC.MQQA_PUT_INHIBITED
- MQC.MQQA_PUT_ALLOWED

### getInhibitGet

```
public int getInhibitGet()
```

擲出 MQException。

指出這個佇列是否能進行 get 作業。

**傳回** 可能的值如下：
- MQC.MQQA_GET_INHIBITED
- MQC.MQQA_GET_ALLOWED

### setInhibitGet

```
public void setInhibitGet(int inhibit)
```

擲出 MQException。

控制這個佇列是否能進行 get 作業。可用的值如下：
- MQC.MQQA_GET_INHIBITED
- MQC.MQQA_GET_ALLOWED

### getTriggerControl

```
public int getTriggerControl()
```

擲出 MQException。

指出是否要將觸發訊息寫入起始佇列中，以便能啓動應用程式來服務佇列。

**傳回** 可能的值如下：
- MQC.MQTC_OFF
- MQC.MQTC_ON

### setTriggerControl

```
public void setTriggerControl(int trigger)
```

擲出 MQException。

控制是否要將觸發訊息寫入起始佇列中，以便能啓動應用程式來服務佇列。可用的值如下：
- MQC.MQTC_OFF
- MQC.MQTC_ON

### getTriggerData

```
public String getTriggerData()
```

擲出 MQException。

佇列管理程式在訊息到達這個佇列時插入觸發訊息的自由格式資料，會使觸發訊息寫入起始佇列中。

### setTriggerData

```
public void setTriggerData(String data)
```

擲出 MQException。

設定佇列管理程式在訊息到達這個佇列時插入觸發訊息的自由格式資料，會使觸發訊息寫入起始佇列中。 MQC.MQ_TRIGGER_DATA_LENGTH 提供字串長度的許可上限。

### getTriggerDepth

```
public int getTriggerDepth()
```

擲出 MQException。

當觸發類型設爲 MQC.MQTT_DEPTH 時，在寫入觸發訊息之前，必須存在於佇列中的訊息數目。

### setTriggerDepth

```
public void setTriggerDepth(int depth)
```

擲出 MQException。

設定當觸發類型設爲 MQC.MQTT_DEPTH 時，在寫入觸發訊息之前，必須存在於佇列中的訊息數目。

### getTriggerMessagePriority

```
public int getTriggerMessagePriority()
```

擲出 MQException。

這是一個訊息優先順序，低於這個優先順序的訊息不會影響到觸發訊息的產生與否（也就是在決定應不應產生觸發時，佇列管理程式會忽略這些訊息）。零值會使所有訊息都會影響到觸發訊息的產生。

### setTriggerMessagePriority

```
public void setTriggerMessagePriority(int priority)
```

擲出 MQException。

設定一個訊息優先順序，低於這個優先順序的訊息不會影響到觸發訊息的產生與否（也就是在決定應不應產生觸發時，佇列管理程式會忽略這些訊息）。零值會使所有訊息都會影響到觸發訊息的產生。

### getTriggerType

```
public int getTriggerType()
```

擲出 MQException。

在訊息到達這個佇列時，會造成寫入觸發訊息的狀況。

**傳回** 可能的值如下：

- MQC.MQTT_NONE
- MQC.MQTT_FIRST
- MQC.MQTT_EVERY
- MQC.MQTT_DEPTH

### setTriggerType

```
public void setTriggerType(int type)
```

擲出 MQException。

設定在訊息到達這個佇列時，會造成寫入觸發訊息的狀況。可能的值如下：

- MQC.MQTT_NONE
- MQC.MQTT_FIRST
- MQC.MQTT_EVERY
- MQC.MQTT_DEPTH

### close

```
public synchronized void close()
```

擲出 MQException。

置換 第99頁的『MQManagedObject.close』。

# MQQueueManager

```
java.lang.Object
   |
   +-- com.ibm.mq.MQManagedObject
          |
          +-- com.ibm.mq.MQQueueManager
```

public class **MQQueueManager**
extends **MQManagedObject**（請參閱頁 97。）

**註:** 這個類別中某些可用選項的行爲會隨著使用它們的環境而不同。這些元素會有標出 *****。請參閱第71頁的『第8章 環境相依行爲』以取得詳細資訊。

## 變數

### isConnected

```
public boolean isConnected
```

如果佇列管理程式連線仍在開啓狀態中，即爲 True。

## 建構子

### MQQueueManager

```
public MQQueueManager(String queueManagerName)
```

擲出 MQException。

建立一個連線來通往具名的佇列管理程式。

**註:** 當使用 MQSeries Class for Java 時，連線要求期間所用的主電腦名稱、通道名稱及埠都由 MQEnvironment 類別來指定。這必須在呼叫這個建構子*之前*完成。

下列範例顯示通往主電腦名稱爲 fred.mq.com 之機器中所執行的佇列管理程式 ″MYQM″ 的連線。

```
MQEnvironment.hostname = "fred.mq.com";  // 要連接的主電腦
MQEnvironment.port     = 1414;           // 要連接的埠。
                                         // 如果我不設定這個，
                                         // 則預設值為 1414
                                         // (預設值的 MQSeries 埠)
MQEnvironment.channel  = "channel.name"; // 佇列管理程式上的
                                         //   SVR CONN 通道的
                                         // 「區分大小寫」
                                         // 名稱
MQQueueManager qMgr    = new MQQueueManager("MYQM");
```

如果佇列管理程式名稱保留空白（空值或 ″″），這時會建立一個連線來通往預設佇列管理程式。

另請參閱第86頁的『MQEnvironment』。

### MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      MQConnectionManager cxManager)
```

擲出 MQException。

這個建構子會利用 MQEnvironment 中的內容來連接到指定的佇列管理程式。指定的 MQConnectionManager 負責管理連線。

### MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      ConnectionManager cxManager)
```

擲出 MQException。

這個建構子會利用 MQEnvironment 中的內容來連接到指定的佇列管理程式。指定的 ConnectionManager 負責管理連線。

這個方法需要 Java 2 第 1.3 版或以上的 JVM，且安裝有 JAAS 1.0 或以上。

### MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      int options)
```

擲出 MQException。

這個版本的建構子專供連結模式使用，它利用延伸的連線 API (MQCONNX) 來連接佇列管理程式。 *options* 參數可讓您選擇快速或一般連結。可能的值如下：

- MQC.MQCNO_FASTPATH_BINDING，代表快速連結 *****。
- MQC.MQCNO_STANDARD_BINDING，代表一般連結。

### MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      int options,
                      MQConnectionManager cxManager)
```

擲出 MQException。

這個建構子會傳遞提供的選項來執行 MQCONNX。指定的 MQConnectionManager 負責管理連線。

### MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      int options,
                      ConnectionManager cxManager)
```

擲出 MQException。

這個建構子會傳遞提供的選項來執行 MQCONNX。指定的 ConnectionManager 負責管理連線。

這個方法需要 Java 2 第 1.3 版或以上的 JVM，且安裝有 JAAS 1.0 或以上。

### MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      java.util.Hashtable properties)
```

properties 參數會採用一系列說明這個特定佇列管理程式之 MQSeries 環境的鍵/值配對。這些內容在指定之後，會取代 MQEnvironment 類別所設定的值，且容許以佇列管理程式爲基礎，在佇列管理程式上設定個別的內容。請參閱第87頁的『"MQEnvironment.properties"』。

### MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      Hashtable properties,
                      MQConnectionManager cxManager)
```

擲出 MQException。

這個建構子會利用提供的內容 Hashtable 來置換 MQEnvironment 中的內容，以連接到指定的佇列管理程式。指定的 MQConnectionManager 負責管理連線。

### MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      Hashtable properties,
                      ConnectionManager cxManager)
```

擲出 MQException。

這個建構子會利用提供的內容 Hashtable 來置換 MQEnvironment 中的內容，以連接到指定的佇列管理程式。指定的 ConnectionManager 負責管理連線。

這個方法需要 Java 2 第 1.3 版或以上的 JVM，且安裝有 JAAS 1.0 或以上。

## 方法

### getCharacterSet

```
public int getCharacterSet()
```

擲出 MQException。

傳回佇列管理程式字碼集的 CCSID（編碼字集識別碼）。這會定義佇列管理程式用於應用程式設計介面中之所有字串欄位的字集。

如果您在切斷佇列管理程式的連線之後呼叫這個方法，便會擲出 MQException。

### getMaximumMessageLength

```
public int getMaximumMessageLength()
```

擲出 MQException。

傳回佇列管理程式所能處理的訊息（位元組）長度上限。沒有任何佇列可以採用大於這個值的訊息長度上限來定義。

如果您在切斷佇列管理程式的連線之後呼叫這個方法，便會擲出 MQException。

### getCommandLevel

```
public int getCommandLevel()
```

擲出 MQException。

指出佇列管理程式所支援的系統控制指令層次。特定指令層次所對應的系統控制指令集會隨著佇列管理程式執行其中之平台的架構而不同。請參閱您的平台所適用的 MQSeries 文件，來取得進一步的詳細資料。

如果您在切斷佇列管理程式的連線之後呼叫這個方法，便會擲出 MQException。

**傳回** MQC.MQCMDL_LEVEL_xxx 常數之一

### getCommandInputQueueName

```
public String getCommandInputQueueName()
```

擲出 MQException。

傳回佇列管理程式所定義之指令輸入佇列的名稱。這是應用程式獲授權時，可將指令送往的佇列。

如果您在切斷佇列管理程式的連線之後呼叫這個方法，便會擲出 MQException。

### getMaximumPriority

```
public int getMaximumPriority()
```

擲出 MQException。

傳回佇列管理程式所支援的訊息優先順序上限。優先順序範圍從零（最低）至這個值。

如果您在切斷佇列管理程式的連線之後呼叫這個方法，便會擲出 MQException。

### getSyncpointAvailability

```
public int getSyncpointAvailability()
```

擲出 MQException。

指出佇列管理程式支不支援工作單元及與 MQQueue.get 和 MQQueue.put 方法之間的同步點。

**傳回**

- 如果可以使用同步點的話，即爲 MQC.MQSP_AVAILABLE。
- 如果不能使用同步點的話，即爲 MQC.MQSP_NOT_AVAILABLE。

如果您在切斷佇列管理程式的連線之後呼叫這個方法，便會擲出 MQException。

### getDistributionListCapable

```
public boolean getDistributionListCapable()
```

指出佇列管理程式支不支援分送清單。

### disconnect

```
public synchronized void disconnect()
```

擲出 MQException。

終止與佇列管理程式的連線。這個佇列管理程式所存取的所有開啓的佇列及程序都會關閉，因而會成爲無法使用。當您切斷與佇列管理程式的連線時，重新連接的唯一方法是建立一個新的 MQQueueManager 物件。

通常，工作單元中所執行的工作都會被確定。不過，如果這個連線是由 ConnectionManager 來管理，而不是由 MQConnectionManager 來管理，便可能會回復工作單元。

### commit

```
public synchronized void commit()
```

擲出 MQException。

呼叫這個方法會告訴佇列管理程式應用程式已到達同步點，且上一個同步點之後所發生的所有訊息取得和放置動作都會成爲永久的。其它應用程式便可以使用工作單元中所放置的訊息（在 MQPutMessageOption 的選項欄位中設定 MQC.MQPMO_SYNCPOINT 旗號）。在工作單元中擷取的訊息（在 MQGetMessageOption 的選項欄位中設定 MQC.MQGMO_SYNCPOINT 旗號）會被刪除。

另請參閱後面的 ″backout″ 的說明。

### backout

```
public synchronized void backout()
```

擲出 MQException。

呼叫這個方法會告訴佇列管理程式應用程式上一個同步點之後所發生的所有訊息取得和放置動作都會回復。工作單元中所放置的訊息（在 MQPutMessageOption 的選項欄位中設定 MQC.MQPMO_SYNCPOINT 旗號）都會刪除；在工作單元中擷取的訊息（在 MQGetMessageOption 的選項欄位中設定 MQC.MQGMO_SYNCPOINT 旗號）會保留在佇列中。

另請參閱上述 ″commit″ 的說明。

## accessQueue

```
public synchronized MQQueue accessQueue
                        (
                        String queueName, int openOptions,
                        String queueManagerName,
                        String dynamicQueueName,
                        String alternateUserId
                        )
```

擲出 MQException。

建立這個佇列管理程式中之 MQSeries 佇列的存取，以取得或瀏覽訊息、放置訊息、查詢佇列屬性或設定佇列屬性。

如果具名佇列是模型佇列，則會建立一個動態本端佇列。您可以檢查傳回的 MQQueue 物件的 name 屬性來判斷所建立佇列的名稱。

### 參數

*queueName*

要開啓的佇列名稱。

*openOptions*

控制佇列之開啓的選項。有效選項如下：

**MQC.MQOO_BROWSE**

開啓以瀏覽訊息。

**MQC.MQOO_INPUT_AS_Q_DEF**

開啓以利用佇列定義的預設值取得訊息。

**MQC.MQOO_INPUT_SHARED**

開啓以利用共用存取瀏覽訊息。

**MQC.MQOO_INPUT_EXCLUSIVE**

開啓以利用互斥存取瀏覽訊息。

**MQC.MQOO_OUTPUT**

開啓以放置訊息。

**MQC.MQOO_INQUIRE**

開啓以進行查詢 - 如果您要查詢內容的話，便會需要。

**MQC.MQOO_SET**

開啓以設定屬性。

**MQC.MQOO_SAVE_ALL_CONTEXT**

在擷取訊息時儲存環境定義*。

**MQC.MQOO_SET_IDENTITY_CONTEXT**

容許設定身份環境定義。

**MQC.MQOO_SET_ALL_CONTEXT**

容許設定所有環境定義。

**MQC.MQOO_ALTERNATE_USER_AUTHORITY**

以指定的使用者識別碼來進行驗證。

**MQC.MQOO_FAIL_IF_QUIESCING**

如果佇列管理程式在靜止中，便告失敗。

**MQC.MQOO_BIND_AS_QDEF**
使用佇列的預設連結。

**MQC.MQOO_BIND_ON_OPEN**
當開啓佇列時，將控點連結至目的地。

**MQC.MQOO_BIND_NOT_FIXED**
不連結至特定目的地。

**MQC.MQOO_PASS_ALL_CONTEXT**
容許傳遞所有環境定義。

**MQC.MQOO_PASS_IDENTITY_CONTEXT**
容許傳遞身份環境定義。

如果需要多個選項，您可以利用按位元的 OR 運算子，將這些值加起來或組合起來。請參閱MQSeries *MQSeries Application Programming Reference*，以取得這些選項的完整說明。

*queueManagerName*
佇列定義其中的佇列管理程式名稱。完全空白或空值的名稱表示這個 MQQueueManager 物件所連接的佇列管理程式。

*dynamicQueueName*
除非 queueName 指定模型佇列的名稱，否則，會忽略這個參數。如果指定的話，這個參數會指定要建立的動態佇列的名稱。如果 queueName 指定模型佇列的名稱，則空白或空值名稱無效。如果名稱中最後一個非空白字元是星號 (*)，佇列管理程式會將星號改成字串，且會確保在這個佇列管理程式中，產生給這個佇列的名稱是唯一的。

*alternateUserId*
如果在 openOption 參數中指定了 MQOO_ALTERNATE_USER_AUTHORITY，這個參數會指定用來進行開啓鑑別檢查的替換使用者識別碼。如果沒有指定 MQOO_ALTERNATE_USER_AUTHORITY 的話，這個參數可以保留空白（或空值）。

**傳回** 已順利開啓的 MQQueue。

如果 open 失敗的話，會擲出 MQException。

另請參閱第145頁的『″accessProcess″』。

### accessQueue

```
public synchronized MQQueue accessQueue
                        (
                        String queueName,
                        int openOptions
                        )
```

如果您在切斷佇列管理程式的連線之後呼叫這個方法，便會擲出 MQException。

#### 參數

*queueName*
要開啓的佇列名稱

*openOptions*
控制佇列之開啓的選項。

請參閱第143頁的『MQQueueManager.accessQueue』，以取得參數的詳細資訊。

*queueManagerName*、*dynamicQueueName* 和 *alternateUserId* 都設爲 *""*。

### accessProcess

```
public synchronized MQProcess accessProcess
                         (
                        String processName,
                        int openOptions,
                        String queueManagerName,
                        String alternateUserId
                        )
```

擲出 MQException。

建立這個佇列管理程式中之 MQSeries 程序的存取，以查詢程序屬性。

#### 參數

*processName*
要開啓的程序名稱。

*openOptions*
控制程序之開啓的選項。指定的選項會自動加入查詢，因而不需要明確地指定它。

有效選項如下：

**MQC.MQOO_ALTERNATE_USER_AUTHORITY**
以指定的使用者 ID 來進行驗證

**MQC.MQOO_FAIL_IF_QUIESCING**
如果佇列管理程式在靜止中，便告失敗

如果需要多個選項，您可以利用按位元的 OR 運算子，將這些值加起來或組合起來。請參閱 *MQSeries Application Programming Reference*，以取得這些選項的完整說明。

*queueManagerName*
程序定義其中的佇列管理程式名稱。應用程式應該將這個參數保留空白或空值。

*alternateUserId*
如果在 openOption 參數中指定了 MQOO_ALTERNATE_USER_AUTHORITY，這個參數會指定用來進行開啓鑑別檢查的替換使用者識別碼。如果沒有指定 MQOO_ALTERNATE_USER_AUTHORITY 的話，這個參數可以保留空白（或空值）。

**傳回** 已順利開啓的 MQProcess。

如果 open 失敗的話，會擲出 MQException。

另請參閱第143頁的『MQQueueManager.accessQueue』。

### accessProcess

這是先前所說明之 AccessProcess 方法的簡化版本。

```
public synchronized MQProcess accessProcess
                      (
                     String processName,
                     int openOptions
                     )
```

這是先前所說明之 AccessQueue 方法的簡化版本。

**參數**

*processName*
要開啓的程序名稱。

*openOptions*
控制程序之開啓的選項。

請參閱第145頁的『"accessProcess"』，以取得選項的詳細資訊。

*queueManagerName* 和 *alternateUserId* 都設爲 ""。

### accessDistributionList

```
public synchronized MQDistributionList accessDistributionList
          (
           MQDistributionListItem[] litems, int openOptions,
           String alternateUserId
          )
```

擲出 MQException。

**參數**

*litems* 要併入分送清單中的項目。

*openOptions*
控制分送清單之開啓的選項。

*alternateUserId*
如果在 openOption 參數中指定了 MQOO_ALTERNATE_USER_AUTHORITY，這個參數會指定用來進行開啓鑑別檢查的替換使用者識別碼。如果沒有指定 MQOO_ALTERNATE_USER_AUTHORITY 的話，這個參數可以保留空白（或空值）。

**傳回** 已開啓且準備執行放置作業的新建立的 MQDistributionList。

如果 open 失敗的話，會擲出 MQException。

另請參閱第143頁的『MQQueueManager.accessQueue』。

### accessDistributionList

這是先前所說明之 AccessDistributionList 方法的簡化版本。

```
public synchronized MQDistributionList accessDistributionList
          (
           MQDistributionListItem[] litems,
           int openOptions,
          )
```

**參數**

*litems* 要併入分送清單中的項目。

*openOptions*
控制分送清單之開啓的選項。

請參閱第146頁的『accessDistributionList』，以取得參數的詳細資訊。

*alternateUserId* 設爲 *""*。

### begin* (bindings connection only)

```
public synchronized void begin()
```

擲出 MQException。

這個方法只在連結模式下受到 MQSeries Class for Java 的支援，它會發出信號給佇列管理程式，指出正在啓動新工作單元。

請不要將這個方法用在使用本端一段式異動的應用程式上。

### isConnected

```
public boolean isConnected()
```

傳回 isConnected 變數的值。

# MQSimpleConnectionManager

```
java.lang.Object      com.ibm.mq.MQConnectionManager
   |                          |
   +- com.ibm.mq.MQSimpleConnectionManager
```

public class **MQSimpleConnectionManager**
implements **MQConnectionManager**（請參閱頁 152。）

MQSimpleConnectionManager 提供基本連線儲存池功能。您可以利用 MQSimpleConnectionManager 來作爲預設 ConnectionManager，或作爲 MQQueueManager 建構子的一個參數。當建構 MQQueueManager 時，會使用儲存池中最近使用的連線。

當連線在指定的時間內沒有使用時，或連線超出儲存池中未用連線的指定數目時，會毀損連線（由個別執行緒進行）。您可以指定逾時期間及未用連線數目上限。

## 變數

### MODE_AUTO

```
public static final int MODE_AUTO
```
請參閱『setActive』。

### MODE_ACTIVE

```
public static final int MODE_ACTIVE
```
請參閱『setActive』。

### MODE_INACTIVE

```
public static final int MODE_INACTIVE
```
請參閱『setActive』。

## 建構子

### MQSimpleConnectionManager

```
public MQSimpleConnectionManager()
```

建構一個 MQSimpleConnectionManager。

## 方法

### setActive

```
public void setActive(int mode)
```

設定連線儲存池的作用模式。

**參數**

*mode* 所需要的連線儲存池作用模式。有效值如下：

**MODE_AUTO**
當連線管理程式是預設連線管理程式且 MQEnvironment 所保留的 MQPoolToken 集中至少有一個記號，這時連線儲存池會在作用中。這是預設模式。

**MODE_ACTIVE**
連線儲存池永遠在作用中。當呼叫 MQQueueManager.disconnect() 時，會將基礎連線放在儲存池中，下次建構 MQQueueManager 物件時，可能會重複使用它。如果連線的未用時間超出逾時期間，或儲存池大小超出 HighThreshold，便會由個別執行緒來毀損連線。

**MODE_INACTIVE**
連線儲存池永遠不在作用中。當進入這個模式時，會清除通往 MQSeries 的連線儲存池。當呼叫 MQQueueManager.disconnect() 時，會毀損任何作用中之 MQQueueManager 物件的基礎連線。

### getActive

```
public int getActive()
```

取得連線儲存池的模式。

**傳回**

含有下列值之一的現行作用中連線儲存池模式（請參閱第148頁的『setActive』）：

MODE_AUTO

MODE_ACTIVE

MODE_INACTIVE

### setTimeout

```
public void setTimeout(long timeout)
```

設定逾時值，在經過這段時間之後仍沒有使用的連線，由個別執行緒來予以毀損。

**參數**

*timeout* 逾時值（以毫秒爲單位）。

### getTimeout

```
public long getTimeout()
```

傳回逾時值。

### setHighThreshold

```
public void setHighThreshold(int threshold)
```

設定 HighThreshold。如果儲存池中未用連線的數目超出這個值，會毀損儲存池中最舊的未用連線。

**參數**

*threshold*
儲存池中未用連線的數目上限。

### getHighThreshold

```
public int getHighThreshold ()
```

傳回 HighThreshold 值。

# MQC

public interface **MQC**
extends **Object**

MQC 介面會定義 MQ Java 程式設計介面使用的所有常數（除了完成碼常數和錯誤碼常數）。如果您的程式中要參照這些常數，請在常數名稱前面附加 "MQC." 字首。比方說，您可以依下列方式來設定佇列的 close 選項：

```
MQQueue queue;
 ...
queue.closeOptions = MQC.MQCO_DELETE; // 當關閉它時，
                                      // 刪除
                                      // 這個佇列
 ...
```

請參閱 *MQSeries Application Programming Reference*，找到這些常數的完整說明。

完成碼和錯誤碼常數由 MQException 類別來定義。請參閱第91頁的『MQException』。

## MQPoolServicesEventListener

public interface **MQPoolServicesEventListener**
extends **Object**

**註:** 應用程式通常不會用到這個介面。

MQPoolServicesEventListener 專供預設 ConnectionManager 的提供者實作。當 MQPoolServicesEventListener 在 MQPoolService 物件中登錄時，每次在 MQEnvironment 所管理的 MQPoolToken 集中新增或移除 MQPoolToken 時，事件接聽器都會收到一個事件。每次預設 ConnectionManager 改變時，它也會收到一個事件。

另請參閱第121頁的『MQPoolServices』和
第122頁的『MQPoolServicesEvent』。

### 方法

#### tokenAdded

```
public void tokenAdded(MQPoolServicesEvent event)
```

當 MQPoolToken 新增到 MQPoolToken 的集中，會呼叫這個方法。

#### tokenRemoved

```
public void tokenRemoved(MQPoolServicesEvent event)
```

當從 MQPoolToken 的集中刪除 MQPoolToken 時，會呼叫這個方法。

#### defaultConnectionManagerChanged

```
public void defaultConnectionManagerChanged(MQPoolServicesEvent event)
```

當設定預設 ConnectionManager 時，會呼叫這個方法。這時將會清除 MQPoolTokens 的集。

## MQConnectionManager

這是一個專用介面，應用程式無法實作它。 MQSeries Class for Java 提供這個介面 (MQSimpleConnectionManager) 的實作，您可以利用 MQQueueManager 建構子或 MQEnvironment.setDefaultConnectionManager 來指定它。

請參閱第148頁的『MQSimpleConnectionManager』。

要提供自己的 ConnectionManager 的應用程式或中介軟體應該實作 javax.resource.spi.ConnectionManager。這需要安裝了 JAAS 1.0 的 Java 2 第 1.3 版。

# MQReceiveExit

public interface **MQReceiveExit**
extends **Object**

您可以利用接收跳出程式介面來檢查 MQSeries Class for Java 從佇列管理程式中收到的資料，也可以改變它。

**註:** 在採用連結模式直接連到 MQSeries 時，不適合採用這個介面。

如果要提供自己的接收跳出程式，請定義一個類別來實作這個介面。在建構 MQQueueManager 物件之前，請先建立新的類別案例，指派 MQEnvironment.receiveExit 變數給它。例如：

```
// 在 MyReceiveExit.java 中
class MyReceiveExit implements MQReceiveExit {
  // 您必須提供 receiveExit 方法的
  // 實作
  public byte[] receiveExit(
   MQChannelExit       channelExitParms,
   MQChannelDefinition channelDefinition,
   byte[]              agentBuffer)
  {
   // 您的出口碼來到這裡...
  }
}
// 在您的主要程式中...
MQEnvironment.receiveExit = new MyReceiveExit();
 ...    // 其它起始設定
MQQueueManager qMgr        = new MQQueueManager("");
```

## 方法

### receiveExit

```
public abstract byte[] receiveExit(MQChannelExit channelExitParms,
                                   MQChannelDefinition channelDefinition,
                                   byte agentBuffer[])
```

您的類別必須提供的接收跳出程式。每當 MQSeries Class for Java 從佇列管理程式收到一些資料時，都會呼叫這個方法。

#### 參數

*channelExitParms*
含有跳出程式在其中呼叫的環境定義的相關資訊。 exitResponse 成員變數是用來告訴 MQSeries Class for Java 接下來要採取什麼動作的輸出參數。請參閱第80頁的『MQChannelExit』，以取得進一步的詳細資料。

*channelDefinition*
包含通道的詳細資料，所有與佇列管理程式的通信，皆透過此通道進行。

*agentBuffer*
如果 channelExitParms.exitReason 是 MQChannelExit.MQXR_XMIT，agentBuffer 會含有從佇列管理程式收到的資料；否則，agentBuffer 爲空值。

### 傳回

如果設定跳出程式回應碼（在 channelExitParms 中），使 MQSeries Class for Java 能現在處理資料 (MQXCC_OK)，您的接收跳出程式方法必須傳回要處理的資料。因此，最簡單的接收跳出程式只有 ″return agentBuffer;″ 一行。

另請參閱：
- 第150頁的『MQC』
- 第78頁的『MQChannelDefinition』

## MQSecurityExit

public interface **MQSecurityExit**
extends **Object**

安全跳出程式介面可讓您自訂試圖連接佇列管理程式時所進行的安全串流。

**註:** 在採用連結模式直接連到 MQSeries 時，不適合採用這個介面。

如果要提供自己的安全跳出程式，請定義一個類別來實作這個介面。在建構 MQQueueManager 物件之前，請先建立新的類別案例，指派 MQEnvironment.securityExit 變數給它。例如：

```
// 在 MySecurityExit.java 中
class MySecurityExit implements MQSecurityExit {
  // 您必須提供 securityExit 方法的
  // 實作
  public byte[] securityExit(
    MQChannelExit       channelExitParms,
    MQChannelDefinition channelDefinition,
    byte[]              agentBuffer)
  {
   // 您的出口碼來到這裡...
  }
}
// 在您的主要程式中...
MQEnvironment.securityExit = new MySecurityExit();
 ...    // 其它起始設定
MQQueueManager qMgr        = new MQQueueManager("");
```

### 方法

#### securityExit

```
public abstract byte[] securityExit(MQChannelExit channelExitParms,
                                    MQChannelDefinition channelDefinition,
                                    byte agentBuffer[])
```

您的類別必須提供的安全跳出程式。

**參數**

*channelExitParms*
含有跳出程式在其中呼叫的環境定義的相關資訊。 exitResponse 成員變數是用來告訴 MQSeries Client for Java 接下來要採取什麼動作的輸出參數。請參閱第80頁的『MQChannelExit』，以取得進一步的詳細資料。

*channelDefinition*
包含通道的詳細資料，所有與佇列管理程式的通信，皆透過此通道進行。

*agentBuffer*
如果 channelExitParms.exitReason 是 MQChannelExit.MQXR_SEC_MSG，agentBuffer 會含有從佇列管理程式收到的安全訊息；否則，agentBuffer 爲空值。

**傳回**

## MQSecurityExit

如果設定跳出程式回應碼（在 channelExitParms 中），使訊息能傳輸給佇列管理程式，您的安全跳出程式方法必須傳回要傳輸的資料。

另請參閱：
- 第150頁的『MQC』
- 第78頁的『MQChannelDefinition』

# MQSendExit

public interface **MQSendExit**
extends **Object**

您可以利用傳送跳出程式介面來檢查 MQSeries Client for Java 傳送給佇列管理程式的資料，也可以改變它。

**註:** 在採用連結模式直接連到 MQSeries 時，不適合採用這個介面。

如果要提供自己的傳送跳出程式，請定義一個類別來實作這個介面。在建構 MQQueueManager 物件之前，請先建立新的類別案例，指派 MQEnvironment.sendExit 變數給它。例如：

```
// 在 MySendExit.java 中
class MySendExit implements MQSendExit {
  // 您必須提供 sendExit 方法的實作
  public byte[] sendExit(
    MQChannelExit       channelExitParms,
    MQChannelDefinition channelDefinition,
    byte[]              agentBuffer)
  {
   // 您的出口碼來到這裡...
  }
}
// 在您的主要程式中...
MQEnvironment.sendExit = new MySendExit();
 ...    // 其它起始設定
MQQueueManager qMgr        = new MQQueueManager("");
```

## 方法

### sendExit

```
public abstract byte[] sendExit(MQChannelExit channelExitParms,
                                MQChannelDefinition channelDefinition,
                                byte agentBuffer[])
```

您的類別必須提供的傳送跳出程式。每當 MQSeries Class for Java 要傳輸某些資料給佇列管理程式時，都會呼叫這個方法。

#### 參數

*channelExitParms*

含有跳出程式在其中呼叫的環境定義的相關資訊。 exitResponse 成員變數是用來告訴 MQSeries Class for Java 接下來要採取什麼動作的輸出參數。請參閱第80頁的『MQChannelExit』，以取得進一步的詳細資料。

*channelDefinition*

包含通道的詳細資料，所有與佇列管理程式的通信，皆透過此通道進行。

*agentBuffer*

如果 channelExitParms.exitReason 是 MQChannelExit.MQXR_XMIT，agentBuffer 會含有要傳輸給佇列管理程式的資料；否則，agentBuffer 爲空值。

### 傳回

如果設定跳出程式回應碼（在 channelExitParms 中），使訊息能傳輸給佇列管理程式 (MQXCC_OK)，您的傳送跳出程式方法必須傳回要傳輸的資料。因此，最簡單的傳送跳出程式只有 ″return agentBuffer;″ 一行。

另請參閱：
- 第150頁的『MQC』
- 第78頁的『MQChannelDefinition』

# ManagedConnection

public interface **javax.resource.spi.ManagedConnection**

**註:** 應用程式通常不會用到這個類別；它是要給 ConnectionManager 的實作使用。

MQSeries Class for Java 提供有從 ManagedConnectionFactory.createManagedConnection 傳回的 ManagedConnection 的實作。這個物件代表 MQSeries 佇列管理程式的連線。

## 方法

### getConnection

```
public Object getConnection(javax.security.auth.Subject subject,
                            ConnectionRequestInfo cxRequestInfo)
```

擲出 ResourceException。

就 ManagedConnection 物件所代表的實體連線建立一個新的連線控點。如果是 MQSeries Class for Java，這會傳回 MQQueueManager 物件。 ConnectionManager 通常會從 allocateConnection 傳回這個物件。

subject 參數會略過。如果 cxRequestInfo 參數不適用的話，會擲出 ResourceException。每個單一 ManagedConnection 都可以同時使用多個連線控點。

### destroy

```
public void destroy()
```

擲出 ResourceException。

毀損通往 MQSeries 佇列管理程式的實體連線。任何擱置中的本端異動都會確定。如果需要詳細的資料，請參閱第160頁的『getLocalTransaction』。

### cleanup

```
public void cleanup()
```

擲出 ResourceException。

關閉所有開啓的連線控點，並且將實體連線重設爲準備放入儲存池的起始狀態。任何擱置中的本端異動都會回復。如果需要詳細的資料，請參閱第160頁的『getLocalTransaction』。

### associateConnection

```
public void associateConnection(Object connection)
```

擲出 ResourceException。

MQSeries Class for Java 目前不支援這個方法。這個方法會擲出 javax.resource.NotSupportedException。

## ManagedConnection

### addConnectionEventListener

```
public void addConnectionEventListener(ConnectionEventListener listener)
```

新增一個 ConnectionEventListener 到 ManagedConnection 案例中。

如果 ManagedConnection 發生嚴重錯誤，或在呼叫這個 ManagedConnection 所關聯之連線控點的 MQQueueManager.disconnect() 方法時，會通知接聽器。本端異動事件不會通知接聽器（請參閱『getLocalTransaction』）。

### removeConnectionEventListener

```
public void removeConnectionEventListener(ConnectionEventListener listener)
```

移除已登錄的 ConnectionEventListener。

### getXAResource

```
public javax.transaction.xa.XAResource getXAResource()
```

擲出 ResourceException。

MQSeries Class for Java 目前不支援這個方法。這個方法會擲出 javax.resource.NotSupportedException。

### getLocalTransaction

```
public LocalTransaction getLocalTransaction()
```

MQSeries Class for Java 目前不支援這個方法。這個方法會擲出 javax.resource.NotSupportedException。

目前，ConnectionManager 無法管理 MQSeries 本端異動，本端異動所關聯的事件不會通知給登錄的 ConnectionEventListener。當發生 cleanup() 時，會回復任何進行中的工作單元。當發生 destroy() 時，會確定任何進行中的工作單元。

現存的 API 行爲是在 MQQueueManager.disconnect() 確定進行中的工作單元。只有當 MQConnectionManager（不是 ConnectionManager）管理連線時，才會保留這個現存的行爲。

### getMetaData

```
public ManagedConnectionMetaData getMetaData()
```

擲出 ResourceException。

取得基礎佇列管理程式的 meta 資料資訊。請參閱 第164頁的『ManagedConnectionMetaData』。

### setLogWriter

```
public void setLogWriter(java.io.PrintWriter out)
```

擲出 ResourceException。

設定這個 ManagedConnection 的日誌寫出器。在建立了 ManagedConnection 之後，它會從 ManagedConnectionFactory 中繼承日誌寫出器。

MQSeries Class for Java 目前不使用日誌寫出器。 請參閱第91頁的『MQException.log』，以取得日誌記載的詳細資訊。

### getLogWriter

```
public java.io.PrintWriter getLogWriter()
```

擲出 ResourceException。

傳回這個 ManagedConnection 的日誌寫出器。

MQSeries Class for Java 目前不使用日誌寫出器。 請參閱第91頁的『MQException.log』，以取得日誌記載的詳細資訊。

# ManagedConnectionFactory

public interface **javax.resource.spi.ManagedConnectionFactory**

**註:** 應用程式通常不會用到這個類別。

MQSeries Class for Java 爲 ConnectionManager 提供了一個這個介面的實作。ManagedConnectionFactory 用來建構 ManagedConnection，以及從一組候選中選取一個適用的 ManagedConnection。如果需要這個介面的詳細資料，請參閱 J2EE Connector Architecture 規格（請造訪 Sun 的網站，網址如下：http://java.sun.com）。

## 方法

### createConnectionFactory

```
public Object createConnectionFactory()
```

擲出 ResourceException。

MQSeries Class for Java 目前不支援 createConnectionFactory 方法。這個方法會擲出
javax.resource.NotSupportedException。

### createConnectionFactory

```
public Object createConnectionFactory(ConnectionManager cxManager)
```

擲出 ResourceException。

MQSeries Class for Java 目前不支援 createConnectionFactory 方法。這個方法會擲出
javax.resource.NotSupportedException。

### createManagedConnection

```
public ManagedConnection createManagedConnection
                                (javax.security.auth.Subject subject,
                                 ConnectionRequestInfo cxRequestInfo)
```

擲出 ResourceException。

建立一個新的實體連線來通往 MQSeries 佇列管理程式，及傳回代表這個連線的 ManagedConnection 物件。 MQSeries 會忽略 subject 參數。

### matchManagedConnection

```
public ManagedConnection matchManagedConnection
                                (java.util.Set connectionSet,
                                 javax.security.auth.Subject subject,
                                 ConnectionRequestInfo cxRequestInfo)
```

擲出 ResourceException。

搜尋所提供的候選 ManagedConnection 集，以找出適當的 ManagedConnection。可能傳回空值，也可能從符合連線準則的集中傳回適用的 ManagedConnection。

### setLogWriter

```
public void setLogWriter(java.io.PrintWriter out)
```

擲出 ResourceException。

設定這個 ManagedConnectionFactory 的日誌寫出器。在建立了 ManagedConnection 之後，它會從 ManagedConnectionFactory 中繼承日誌寫出器。

MQSeries Class for Java 目前不使用日誌寫出器。請參閱第91頁的『MQException.log』，以取得日誌記載的詳細資訊。

### getLogWriter

```
public java.io.PrintWriter getLogWriter()
```

擲出 ResourceException。

傳回這個 ManagedConnectionFactory 的日誌寫出器。

MQSeries Class for Java 目前不使用日誌寫出器。請參閱第91頁的『MQException.log』，以取得日誌記載的詳細資訊。

### hashCode

```
public int hashCode()
```

傳回這個 ManagedConnectionFactory 的 hashCode。

### equals

```
public boolean equals(Object other)
```

檢查這個 ManagedConnectionFactory 是否等於另一個 ManagedConnectionFactory。如果兩個 ManagedConnectionFactory 說明相同的目標佇列管理程式，便會傳回 true。

# ManagedConnectionMetaData

public interface **javax.resource.spi.ManagedConnectionMetaData**

**註:** 應用程式通常不會用到這個類別；它是要給 ConnectionManager 的實作使用。

ConnectionManager 可以利用這個類別來接收佇列管理程式之基礎實體連線所關聯的 meta 資料。這個類別的實作是從 ManagedConnection.getMetaData() 傳回的。

## 方法

### getEISProductName

```
public String getEISProductName()
```

擲出 ResourceException。

傳回 “IBM MQSeries”。

### getProductVersion

```
public String getProductVersion()
```

擲出 ResourceException。

傳回一個字串來說明 ManagedConnection 所連接之 MQSeries 佇列管理程式的指令層次。

### getMaxConnections

```
public int getMaxConnections()
```

擲出 ResourceException。

傳回 0。

### getUserName

```
public String getUserName()
```

擲出 ResourceException。

如果 ManagedConnection 代表通往佇列管理程式的從屬站連線，這會傳回連線所用的使用者 ID。否則，它會傳回空字串。

# 第3篇 MQ JMS 的程式設計

# 第10章 撰寫MQ JMS 程式

這一章提供的資訊，可協助您撰寫 MQ JMS 應用程式。它提供 JMS 模型的簡介及應用程式可能需要執行的部份常用作業的程式設計詳細資訊。

## JMS 模型

JMS 會定義一個訊息傳遞服務的通用檢視。您務必瞭解這個檢視畫面以及它如何映射至基礎的 MQSeries 傳輸。

通用 JMS 模型以 Sun 的 javax.jms 套件所定義的下列介面爲基礎：

**Connection**
提供基礎傳輸的存取功能，用來建立 **Sessions**。

**Session**
提供產生和消費訊息的環境定義，其中包括用來建立 **MessageProducers** 和 **MessageConsumers** 的方法。

**MessageProducer**
用來傳送訊息。

**MessageConsumer**
用來接收訊息。

請注意，**Connection** 具有執行緒安全性，但 **Session**、**MessageProducer** 和 **MessageConsumer** 不具執行緒安全性。建議的策略是每個應用程式執行緒使用一個 Session。

在 MQSeries 詞彙中：

**Connection**
提供暫時佇列的範圍。另外，它也提供一個位置來保留控制 MQSeries 之連接方式的參數。佇列管理程式的名稱便是這些參數的範例，如果您使用 MQSeries Java 從屬站連線的話，遠端主電腦的名稱也是這些參數的範例。

**Session**
含有一個 HCONN，並因而定義了一個異動範圍。

**MessageProducer 和 MessageConsumer**
含有一個定義要寫入或讀取之特定佇列的 HOBJ。

請注意，一般 MQSeries 規則套用：

- 不論在任何時候，每個 HCONN 都只能有一個進行中的單一作業。因此，不能同時呼叫 Session 所關聯的 MessageProducer 或 MessageConsumer。這與每個 Session 有一個單一執行緒的 JMS 限制相一致。
- PUT 可以使用遠端佇列，但 GET 只適用於本端佇列管理程式中的佇列。

通用 JMS 介面會延伸爲適用於「點對點」和「發佈/訂閱」行爲的更特定的版本。

以下是點對點的版本：

- QueueConnection
- QueueSession
- QueueSender
- QueueReceiver

JMS 中的主要觀念是您可以（而且最好）撰寫只使用 javax.jms 中之介面參照的應用程式。所有供應商特定資訊都封裝在下列各項的實作中：

- QueueConnectionFactory
- TopicConnectionFactory
- Queue
- Topic

這些都稱爲「管理物件」，也就是說，可利用供應商提供的管理工具來建置且可儲存在 JNDI 名稱空間內。 JMS 應用程式不需要知道實作的提供廠商爲何，便可以從名稱空間擷取這些物件及利用它們。

## 建置連線

連線不是直接建立，而是利用連線 Factory 來建置的。 Factory 物件可儲存在 JNDI 名稱空間中，因而將 JMS 應用程式和提供者特定資訊分離開。如何建立和儲存 Factory 物件的詳細資料，請參閱第29頁的『第5章 使用 MQ JMS 管理工具』。

如果您沒有可用的 JNDI 名稱空間，請參閱第169頁的『在執行時期建立 Factory』。

### 從 JNDI 擷取 Factory

如果要從 JNDI 名稱空間中擷取 Factory，必須設定好起始環境定義，如下列取自 IVTRun 範例檔的片段所示：

```
import javax.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
 .
 .
 .
  java.util.Hashtable environment = new java.util.Hashtable();
  environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
  environment.put(Context.PROVIDER_URL, url);
  Context ctx = new InitialDirContext( environment );
```

其中：

**icf** 定義起始環境定義的 Factory 類別

**url** 定義環境定義特定 URL

如果需要 JNDI 用法的詳細資料，請參閱 Sun 的 JNDI 文件。

**註:** JNDI 套件和 LDAP 服務提供者的某些組合可能會造成 LDAP 錯誤 84。如果要解決這個問題，請在呼叫 `InitialDirContext` 之前插入下面這行。

```
environment.put(Context.REFERRAL, "throw");
```

在取得起始環境定義之後，再利用 `lookup()` 方法從名稱空間中擷取物件。下列程式碼會從 LDAP 型名稱空間中擷取名稱爲 `ivtQCF` 的 QueueConnectionFactory：

```
QueueConnectionFactory factory;
factory = (QueueConnectionFactory)ctx.lookup("cn=ivtQCF");
```

## 使用 Factory 建立連線

Factory 物件的 `createQueueConnection()` 方法用來建立 'Connection'，如下列程式碼所示：

```
QueueConnection connection;
connection = factory.createQueueConnection();
```

## 在執行時期建立 Factory

如果無法使用 JNDI 名稱空間，您可以在執行時期建立 Factory 物件。不過，使用這個方法會降低 JMS 應用程式的可攜性，因爲它需要參照 MQSeries 特定類別。

下列程式碼以所有預設值來建立一個 QueueConnectionFactory：

```
factory = new com.ibm.mq.jms.MQQueueConnectionFactory();
```

（如果您改匯入 `com.ibm.mq.jms` 套件的話，您可以略過 `com.ibm.mq.jms.` 字首。）

從上述 Factory 建立的連線會用 Java 連結來連接本端機器中的預設佇列管理程式。 表 14.中所示的 `set` 方法可用來自訂含 MQSeries 特定資訊的 Factory。

### 啓動連線

JMS 規格會定義哪些連線應該在「停止」狀態中建立。在連線啓動之前，連線所關聯的 MessageConsumer 無法接收到任何訊息。如果要啓動連線，請發出下列指令：

```
connection.start();
```

*表 14. 設定 MQQueueConnectionFactory 中的方法*

| 方法 | 說明 |
|---|---|
| setCCSID(int) | 用來設定 `MQEnvironment.CCSID` 內容 |
| setChannel(String) | 從屬站連線的通道名稱 |
| setHostName(String) | 從屬站連線的主電腦名稱 |
| setPort(int) | 從屬站連線的埠 |
| setQueueManager(String) | 佇列管理程式的名稱 |
| setTemporaryModel(String) | 用來產生暫時目的地作爲 `QueueSession.createTemporaryQueue()` 呼叫之結果的模型佇列名稱。我們建議以此爲暫時動態佇列的名稱，不要作爲永久動態佇列的名稱。 |
| setTransportType(int) | 指定如何連接到 MQSeries。以下是目前可用的選項：<br>• JMSC.MQJMS_TP_BINDINGS_MQ（預設值）<br>• JMSC.MQJMS_TP_CLIENT_MQ_TCPIP。<br>`JMSC` 在 `com.ibm.mq.jms` 套件中 |

*表 14. 設定 MQQueueConnectionFactory 中的方法 (繼續)*

| 方法 | 說明 |
|---|---|
| setReceiveExit(String)<br>setSecurityExit(String)<br>setSendExit(String)<br>setReceiveExitInit(String)<br>setSecutityExitInit(String)<br>setSendExitInit(String) | 這些方法可供使用基礎 MQSeries Classes for Java 所提供的傳送、接收和安全跳出程式。 `set*Exit` 方法會採用實作相關跳出程式方法之類別的名稱。（請參閱 MQSeries 5.1 產品文件，以取得詳細資料。）<br><br>另外，類別必須實作含單一 `String` 參數的建構子。這個字串提供跳出程式可能會需要的任何起始設定資料，它設定成對應的 `set*ExitInit` 方法所提供的值。 |

### 選擇從屬站或連結傳輸

MQ JMS 可以利用從屬站或連結傳輸來與 MQSeries 通信。如果您使用 Java 連結，JMS 應用程式和 MQSeries 佇列管理程式必須在相同的機器中。如果您使用從屬站，佇列管理程式和應用程式可在不同的機器中。

連線 Factory 物件的內容決定了要用哪個傳輸。 第29頁的『第5章 使用 MQ JMS 管理工具』說明如何定義 Factory 物件來使用從屬站或連結傳輸。

下列程式碼片段說明如何在應用程式內定義傳輸：

```
String HOSTNAME = "machine1";
String QMGRNAME = "machine1.QM1";
String CHANNEL = "SYSTEM.DEF.SVRCONN";

factory = new MQQueueConnectionFactory();
factory.setTransportType(JMSC.MQJMS_TP_CLIENT_MQ_TCPIP);
factory.setQueueManager(QMGRNAME);
factory.setHostName(HOSTNAME);
factory.setChannel(CHANNEL);
```

## 取得階段作業

建立好連線之後，請利用 QueueConnection 中的 `createQueueSession` 方法來取得階段作業。

這個方法需要兩個參數：

1. 決定階段作業「要異動」或「不異動」的 Boolean。
2. 決定「認可」模式的參數。

最簡單的例子是含 AUTO_ACKNOWLEDGE 的「不異動」階段作業，如下列程式碼片段所示：

```
QueueSession session;

boolean transacted = false;
session = connection.createQueueSession(transacted,
                                        Session.AUTO_ACKNOWLEDGE);
```

**註：** 連線具有執行緒安全性，但階段作業（及從它們建立的物件）不具執行緒安全性。建議多重執行緒的應用程式要每個執行緒都使用個別的階段作業。

## 傳送訊息

訊息是利用 MessageProducer 來傳送的。如果是點對點的話，這是利用 QueueSession 的 createSender 方法來建立的 QueueSender。 QueueSender 通常是就某特定佇列來建立的，因此，所有利用這個傳送端來傳送的訊息都會傳到相同目的地。目的地是利用 Queue 物件來指定的。 Queue 物件可以在執行時期建立，也可建置和儲存在 JNDI 名稱空間中。

Queue 物件是利用下列方式，從 JNDI 中擷取出來的：

```
Queue ioQueue;
ioQueue = (Queue)ctx.lookup( qLookup );
```

MQ JMS 在 com.ibm.mq.jms.MQQueue 中提供 Queue 的實作。它含有控制 MQSeries 特定行爲細節的內容，但在許多情況下，都可能會使用預設內容。 JMS 會定義一個標準方法來指定將應用程式中的 MQSeries 特定碼縮到最小之目的地。這個機制會用到採 string 參數來說明目的地之 QueueSession.createQueue 方法。字串本身仍採向量特定格式，但這個方式比直接參照供應商類別更具彈性。

MQ JMS 接受 createQueue() 兩種格式的 string 參數。

- 第一個是如下列片段所示的 MQSeries 佇列名稱，它取自 samples 目錄中的 IVTRun 程式：

  ```
  public static final String QUEUE = "SYSTEM.DEFAULT.LOCAL.QUEUE" ;
   .
   .
   .
    ioQueue = session.createQueue( QUEUE );
  ```

- 第二個格式以「統一資源識別碼 (URI)」爲基礎，它的功能更強。您可以利用這個格式來指定遠端佇列（所在的佇列管理程式不是您連接的佇列管理程式之佇列）。您也可以利用它來設定 com.ibm.mq.jms.MQQueue 物件所包含的其它內容。

  序列的 URI 開頭是 queue://，後面接著序列所在的佇列管理程式名稱。後面再接一個 '/'、佇列名稱，以及（選用）設定其餘 Queue 內容的名稱/值配對清單。比方說，前一個範例的 URI 對等項是：

  ```
  ioQueue = session.createQueue("queue:///SYSTEM.DEFAULT.LOCAL.QUEUE");
  ```

  請注意，佇列管理程式的名稱會被忽略。這會解譯成擁有的 QueueConnection 在使用 Queue 物件時所連接的佇列管理程式。

下列範例會連接到 'HOST1.QM1' 佇列管理程式中的 'Q1' 佇列，並會使所有訊息都以優先順序爲 5 且不具持續性的方式來傳送：

```
ioQueue = session.createQueue("queue://HOST1.QM1/Q1?persistence=1&priority=5");
```

表15.列出 URI 的名稱/值配對中所使用的名稱。這個格式的缺點是不支援值的符號名稱，因此，在適當情況下，表格也會指出「特殊」值。請注意，這些特殊值可能會改變。（請參閱『以 'set' 方法設定內容』，以取得設定內容的替代方法。）

*表 15. 佇列 URI 的內容名稱*

| 內容 | 說明 | 值 |
|---|---|---|
| expiry | 訊息的存活時間（以毫秒爲單位） | 0 代表無限，正整數代表逾時值（毫秒） |
| priority | 訊息的優先順序 | 0 到 9、-1=QDEF、-2=APP |
| persistence | 訊息是否要「強制」寫入磁碟中 | 1 = 不具持續性、2 = 具持續性、-1=QDEF、-2=APP |
| CCSID | 目的地之字集 | integers - 基礎 MQSeries 文件所列出的有效值 |
| targetClient | 接收端應用程式是否符合 JMS 標準 | 0=JMS、1=MQ |
| encoding | 如何表示數值欄位 | 如基礎 MQSeries 文件所說明的整數值 |
| **QDEF** - 表示內容應該由 MQSeries 佇列配置來決定的特殊值。<br>**APP** - 表示由 JMS 應用程式控制這個內容的特殊值。 | | |

取得 Queue 物件（如上所述利用 `createQueue`，或從 JNDI 中取得）之後，它必須傳遞到 `createSender` 方法中，以建立 QueueSender：

```
QueueSender queueSender = session.createSender(ioQueue);
```

產生的 `queueSender` 物件用來以 `send` 方法傳送訊息：

```
queueSender.send(outMessage);
```

## 以 'set' 方法設定內容

您可以利用預設建構子，先建立一個 `com.ibm.mq.jms.MQQueue` 案例，來設定 Queue 內容。之後，您可以利用各 public `set` 方法來填入所需要的值。這個方法表示您可以利用符號名稱來作爲內容值。不過，由於這些值都是供應商特定值，且都內嵌在程式碼中，因而應用程式比較不具可攜性。

下列程式碼片段顯示使用 `set` 方法的佇列內容設定。

```
com.ibm.mq.jms.MQQueue q1 = new com.ibm.mq.jms.MQQueue();
      q1.setBaseQueueManagerName("HOST1.QM1");
      q1.setBaseQueueName("Q1");
      q1.setPersistence(DeliveryMode.NON_PERSISTENT);
      q1.setPriority(5);
```

表16.顯示 MQ JMS 提供來搭配 `set` 方法使用的符號內容值。

*表 16. 佇列內容的符號值*

| 內容 | 管理工具關鍵字 | 值 |
|---|---|---|
| expiry | UNLIM<br>APP | JMSC.MQJMS_EXP_UNLIMITED<br>JMSC.MQJMS_EXP_APP |

*表 16. 佇列內容的符號值 (繼續)*

| 內容 | 管理工具關鍵字 | 值 |
|---|---|---|
| priority | APP | JMSC.MQJMS_PRI_APP |
| | QDEF | JMSC.MQJMS_PRI_QDEF |
| persistence | APP | JMSC.MQJMS_PER_APP |
| | QDEF | JMSC.MQJMS_PER_QDEF |
| | PERS | JMSC.MQJMS_PER_PER |
| | NON | JMSC.MQJMS_PER_NON |
| targetClient | JMS | JMSC.MQJMS_CLIENT_JMS_COMPLIANT |
| | MQ | JMSC.MQJMS_CLIENT_NONJMS_MQ |
| encoding | Integer(N) | JMSC.MQJMS_ENCODING_INTEGER_NORMAL |
| | Integer(R) | JMSC.MQJMS_ENCODING_INTEGER_REVERSED |
| | Decimal(N) | JMSC.MQJMS_ENCODING_DECIMAL_NORMAL |
| | Decimal(R) | JMSC.MQJMS_ENCODING_DECIMAL_REVERSED |
| | Float(N) | JMSC.MQJMS_ENCODING_FLOAT_IEEE_NORMAL |
| | Float(R) | JMSC.MQJMS_ENCODING_FLOAT_IEEE_REVERSED |
| | Native | JMSC.MQJMS_ENCODING_NATIVE |

請參閱第39頁的『ENCODING 內容』，以取得編碼的相關討論。

### 訊息類型

JMS 提供有許多訊息類型，每種訊息類型都具體表現了其內容的部份知識。如果要避免參照供應商特定訊息類型的類別名稱，建立訊息的 Session 物件中提供有各種方法。

在範例程式中，採用下列方式來建立文字訊息：

```
System.out.println( "Creating a TextMessage" );
TextMessage outMessage = session.createTextMessage();
System.out.println("Adding Text");
outMessage.setText(outString);
```

可用的訊息類型如下：

- BytesMessage
- MapMessage
- ObjectMessage
- StreamMessage
- TextMessage

請參閱第223頁的『第14章 JMS 介面和類別』，以取得這些類型的詳細資料。

## 接收訊息

利用 QueueReceiver 來接收訊息。這是利用 `createReceiver()` 方法，從 Session 中建立的。這個方法採用 Queue 參數來定義訊息的接收來源。請參閱第171頁的『傳送訊息』，以取得如何建立 Queue 物件的詳細資料。

範例程式利用下列程式碼來建立接收端及讀回測試訊息：

```
QueueReceiver queueReceiver = session.createReceiver(ioQueue);
Message inMessage = queueReceiver.receive(1000);
```

### 接收訊息

接收呼叫中的參數是一個逾時值（以毫秒爲單位）。這個參數定義如果沒有立即可用的訊息，方法應該要等多久。您可以略過這個參數，這時候，呼叫會無限地暫停執行。如果您不要任何延遲，請使用 `receiveNoWait()` 方法。

接收方法會傳回適當類型的訊息。比方說，如果 `TextMessage` 放在佇列中，當收到訊息時，傳回的物件是一個 `TextMessage` 案例。

如果要從訊息主體中擷取內容，便必須從通用 Message 類別（接收方法的宣告傳回類型）強制轉型成更特定的子類別，如 `TextMessage`。如果接收的訊息類型不明，您可以使用 'instanceof' 運算子來判斷它的類型。先測試訊息類別再強制轉型絕對是最恰當的方法，因爲它可以依正常程序來處理非預期的錯誤。

下列程式碼示範 'instanceof' 的使用，以及從 TextMessage 中取出內容：

```
if (inMessage instanceof TextMessage) {
  String replyString = ((TextMessage) inMessage).getText();
  .
  .
  .
} else {
  // 如果 Message 不是 TextMessage，則會列印錯誤訊息。
  System.out.println("Reply message was not a TextMessage");
}
```

### 訊息選取元

JMS 提供一種在佇列中選取訊息子類別的機制，使接收呼叫能傳回這個子集。當建立 QueueReceiver 時，可提供一個含有 SQL（結構化查詢語言）表示式的字串，來判斷要擷取哪些訊息。選取元可以參照 JMS 訊息標題中的欄位及訊息內容中的欄位（實際上，這些是應用程式定義的標題欄位）。請參閱第187頁的『第12章 JMS 訊息』，以取得標題欄位名稱及 SQL 選取元語法的詳細資料。

下列範例顯示如何就名稱爲 `myProp` 的使用者定義內容來進行選取：

```
queueReceiver = session.createReceiver(ioQueue, "myProp = 'blue'");
```

**註:** JMS 規格不容許變更接收端所關聯的選取元。建立好接收端之後，在這個接收端的存活時間內，選取元是固定的。這表示如果您需要不同的選取元，您必須建立新的接收端。

### 非同步遞送

呼叫 QueueReceiver.receive() 的替代方法是登錄一個在有適當的可用訊息時會自動呼叫的方法。下列片段示範這個機制：

```
import javax.jms.*;

public class MyClass implements MessageListener
{
  // 當有訊息可用時，JMS 會呼叫
  // 這個方法。
  public void onMessage(Message message)
  {
    System.out.println("message is "+message);

    // 應用程式特定處理程序在此
    .
    .
    .
  }
}

 .
 .
 .
  // 在主要程式中（其它類別的可能性）
  MyClass listener = new MyClass();
  queueReceiver.setMessageListener(listener);

  // 主要程式現在可以延續其它應用程式特定的
  // 行為。
```

**註:** 在 QueueReceiver 中使用非同步遞送，會將整個 Session 標示成非同步。明確呼叫使用非同步遞送之 Session 所關聯的 QueueReceiver 的 receive 方法，是一項錯誤。

## 關閉

記憶體回收本身無法及時釋出所有 MQSeries 資源。當應用程式需要在 Session 層次或更低層次建立許多存活時間很短的 JMS 物件時，尤其如此。因此，當資源不再需要時，呼叫各類別（QueueConnection、QueueSession、QueueSender 和 QueueReceiver）的 close() 方法非常重要。

### 關閉時 Java 虛擬機器當掉

如果 MQ JMS 應用程式完成作業，而沒有呼叫 Connection.close()，有些 JVM 可能會當掉。如果發生這個問題，請編輯應用程式來呼叫 Connection.close()，或使用 Ctrl-C 鍵來終止 JVM。

## 處理錯誤

JMS 應用程式中的任何執行時期錯誤都由異常狀況來報告。 JMS 的大部份方法都會擲出 JMSExceptions 來指出錯誤。捕捉這些異常狀況，將它們顯示到適合的輸出中，是好的程式設計作法。

JMSException 不像一般的 Java 異常狀況，它可能還含有其它內嵌的異常狀況。對 JMS 而言，這可能是非常有價值的方法，可用來傳遞基礎傳輸中非常重要的詳細資料。在 MQ JMS 中，當 MQSeries 產生 MQException 時，這個異常狀況通常會併成 JMSException 中的內嵌異常狀況。

JMSException 的實作不會在其 toString() 方法的輸出中併入內嵌的異常狀況。因此，您必須依下列片段所示，明確檢查是否有內嵌的異常狀況，並將它印出來：

```
try {
  .
  . code which may throw a JMSException
  .
} catch (JMSException je) {
  System.err.println("caught "+je);
  Exception e = je.getLinkedException();
  if (e != null) {
    System.err.println("linked exception: "+e);
  }
}
```

## 異常狀況接聽器

對於非同步傳訊遞送，應用程式碼無法捕捉因接收訊息失敗而產生的異常狀況。這是因爲應用程式碼不會明確呼叫 receive() 方法。如果要克服這個狀況，或許可以登錄一個 ExceptionListener，它是實作 onException() 方法的類別案例。當發生嚴重錯誤時，會傳遞 JMSException 作爲唯一參數來呼叫這個方法。如果需要詳細資料，請參閱 Sun 的 JMS 文件。

# 第11章 設計發佈/訂閱應用程式

這一節介紹撰寫發佈/訂閱應用程式來使用 MQSeries Classes for Java 訊息服務的程式設計模型。

## 撰寫簡式發佈/訂閱應用程式

這一節提供簡式 MQ JMS 應用程式的簡單實務。

### 匯入必要的套件

MQSeries Class for Java 訊息服務應用程式的開頭是一些 import 陳述式，其中至少要有下列各項：

```
import javax.jms.*;                // JMS 介面
import javax.naming.*;             // 使用管理物件的
import javax.naming.directory.*;   //    JNDI 查閱
```

### 取得或建立 JMS 物件

下個步驟是取得或建立一些 JMS 物件：

1. 取得 TopicConnectionFactory
2. 建立 TopicConnection
3. 建立 TopicSession
4. 從 JNDI 中取得 Topic
5. 建立 TopicPublishers 和 TopicSubscribers

其中有許多程序類似如下所示點對點所用的程序：

#### 取得 TopicConnectionFactory

要做到這一點，比較好的方法是使用 JNDI 查閱，以便能夠維護應用程式碼的可攜性。下列程式碼會起始設定 JNDI 環境定義：

```
String CTX_FACTORY = "com.sun.jndi.ldap.LdapCtxFactory";
String INIT_URL    = "ldap://server.company.com/o=company_us,c=us";

Java.util.Hashtable env = new java.util.Hashtable();
env.put( Context.INITIAL_CONTEXT_FACTORY, CTX_FACTORY );
env.put( Context.PROVIDER_URL,            INIT_URL );
env.put( Context.REFERRAL,                "throw" );

Context ctx = null;
try {
   ctx = new InitialDirContext( env );
} catch( NamingException nx ) {
   // 新增程式碼以處理連接至 JNDI 環境定義的能力
}
```

**註:** CTX_FACTORY 和 INIT_URL 變數需要自訂， 以配合您的安裝結構及您的 JNDI 服務提供者。

JNDI 起始設定所需要的內容在傳遞給 InitialDirContext 建構子的一份雜湊表中。如果這個連線失敗，會擲出一個異常狀況，指出無法提供稍後應用程式會用到的管理物件。

現在，請利用管理者所定義的查閱鍵來取得 TopicConnectionFactory：

```
TopicConnectionFactory factory;
factory = (TopicConnectionFactory)lookup("cn=sample.tcf");
```

如果無法使用 JNDI 名稱空間，您可以在執行時期建立 TopicConnectionFactory。您可以採用類似於第169頁的『在執行時期建立 Factory』中所說明適用於 QueueConnectionFactory 的方法，來建立新的 com.ibm.mq.jms.MQTopicConnectionFactory。

### 建立 TopicConnection

這是從 TopicConnectionFactory 物件建立的。連線永遠在 stop 狀態中起始設定，且必須用下列程式碼來啓動：

```
TopicConnection conn;
conn = factory.createTopicConnection();
conn.start();
```

### 建立 TopicSession

這是利用 TopicConnection 來建立的。這個方法需要兩個參數；一個表示階段作業是否進行異動處理，一個指定認可模式：

```
TopicSession session = conn.createTopicSession( false,
                                                Session.AUTO_ACKNOWLEDGE );
```

### 取得 Topic

這個物件可以從 JNDI 中取得，以搭配稍後建立的 TopicPublisher 和 TopicSubscriber 使用。下列程式碼會擷取 Topic：

```
Topic topic = null;
 try {
    topic = (Topic)ctx.lookup( "cn=sample.topic" );
 } catch( NamingException nx ) {
    // 新增程式碼以處理擷取 JNDI 中 Topic 的能力
 }
```

如果無法使用 JNDI 名稱空間，您可以依照第180頁的『在執行時期建立主題』中所說明，在執行時期建立 Topic。

### 建立發佈資訊的消費者和產生者

您必須根據您撰寫的 JMS 從屬站應用程式的性質來建立訂閱者和/或發佈者。請依照下列方式來使用 createPublisher 和 createSubscriber 方法：

```
// 建立發佈者，在給定的主題中發佈
 TopicPublisher pub = session.createPublisher( topic );
// 建立訂閱者，在給定的主題中訂閱
 TopicSubscriber sub = session.createSubscriber( topic );
```

### 發佈訊息

TopicPublisher 物件 pub 用來發佈訊息，如同點對點領域中所用的 QueueSender。下列片段會利用階段作業來建立 TextMessage，再發佈訊息：

```
// 建立 TextMessage 並將某些資料放在其中
TextMessage outMsg = session.createTextMessage();
outMsg.setText( "This is a short test string!" );

// 使用發佈者來發佈訊息
pub.publish( outMsg );
```

### 接收訂閱

訂閱者必須能夠讀取遞送給他們的訂閱，如下列程式碼所示：

```
// 擷取下一個等候的訂閱
TextMessage inMsg = (TextMessage)sub.receive();

// 取得訊息內容
String payload = inMsg.getText();
```

這個程式碼片段執行 'get-with-wait'，這表示在有可用訊息之前，會暫停執行接收呼叫。您也可以其它版本的接收呼叫（如 'receiveNoWait'）。如果需要詳細資料，請參閱第328頁的『TopicSubscriber』。

### 關閉不要的資源

當發佈/訂閱應用程式終止時，釋出它使用的所有資源是非常重要的。請在可關閉的物件上（訂閱者、發佈者、階段作業及連線）使用 close() 方法：

```
// 關閉發佈者和訂閱者
pub.close();
sub.close();

// 關閉階段作業和連線
session.close();
conn.close();
```

## 使用主題

這一節討論如何在 MQSeries Class for Java 訊息服務應用程式中使用 JMS Topic 物件。

### 主題名稱

這一節說明如何在 MQSeries Class for Java 訊息服務內使用主題名稱。

**註:** JMS 規格沒有指定使用和維護主題階層的確實詳細資料。 因此，這個區域會隨著提供者而不同。

MQ JMS 中的主題名稱用樹狀階層來安排，第180頁的圖3.顯示一個這種範例。

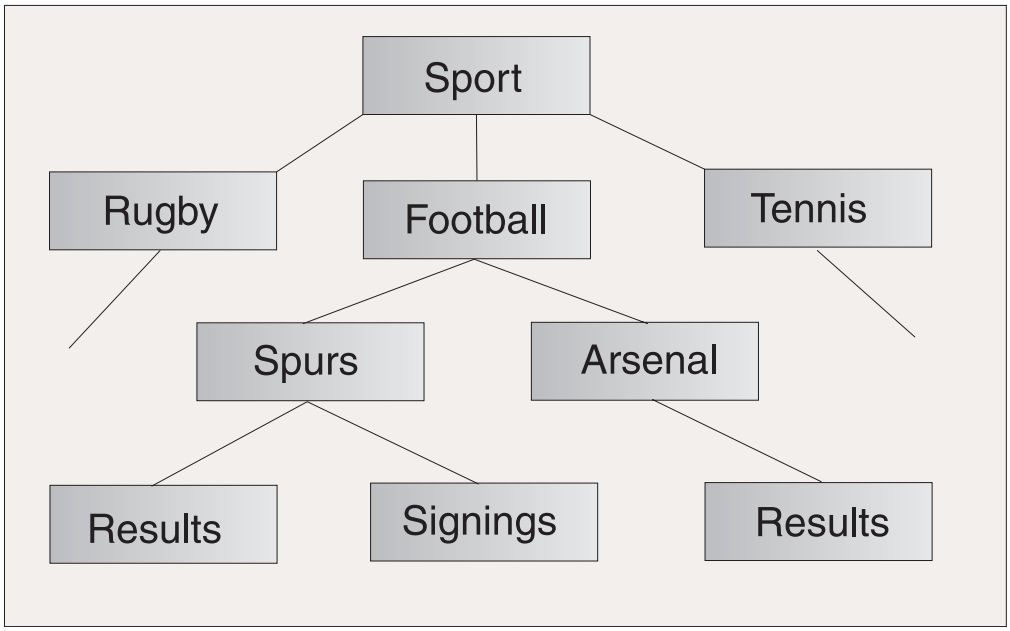


*圖 3. 主題名稱階層*

在主題名稱中，樹狀結構中的各層次用 '/' 字元來分開。這表示 'Signings' 節點由這個主題名稱來表示：

```
Sport/Football/Spurs/Signings
```

在 MQSeries Class for Java 訊息服務中，使用萬用字元是主題系統很強的一個特性。這使得訂閱者能夠同時訂閱多個主題。萬用字元 '*' 對應於零或多個字元，萬用字元 '?' 對應於單一字元。

如果訂閱者訂閱下列主題名稱所代表的 Topic：

```
Sport/Football/*/Results
```

它會接收包括下列各項在內的主題發佈資訊：

- Sport/Football/Spurs/Results
- Sport/Football/Arsenal/Results

如果訂閱主題是：

```
Sport/Football/Spurs/*
```

它會接收包括下列各項在內的主題發佈資訊：

- Sport/Football/Spurs/Results
- Sport/Football/Spurs/Signings

系統的分配管理程式端所用的主題階層並不需要明確的管理。在有了給定主題的第一個發佈者或訂閱者時，分配管理程式會自動建立目前所發佈或訂閱之主題的狀態。

**註:** 發佈者要發佈的主題名稱不能含有萬用字元。

## 在執行時期建立主題

在執行時期建立 Topic 物件的方法有四種：

1. 利用單引數的 `MQTopic` 建構子來建構主題

2. 利用預設 MQTopic 建構子，再呼叫 setBaseTopicName(..) 方法來建構主題
3. 利用階段作業的 createTopic(..) 方法
4. 利用階段作業的 createTemporaryTopic() 方法

**方法 1：使用 MQTopic(..)**

這個方法需要參照 JMS Topic 介面的 MQSeries 實作，因此，會將程式碼轉換成不具可攜性。

建構子採用一個引數，它應該是統一資源識別碼 (URI)。對於 MQSeries Class for Java 訊息服務 Topic 而言，這應該具有如下格式：

```
topic://TopicName[?property=value[&property=value]*]
```

如果需要 URI 及許可名稱/值配對的詳細資料，請參閱第171頁的『傳送訊息』。

下列程式碼會產生不具持續性且優先順序爲 5 的訊息主題：

```
// 建立 Topic，使用一個引數的 MQTopic 建構子
String tSpec = "Sport/Football/Spurs/Results?persistence=1&priority=5";
Topic rtTopic = new MQTopic( "topic://" + tSpec );
```

**方法 2：使用 MQTopic()，再使用 setBaseTopicName(..)**

這個方法利用預設的 MQTopic 建構子，因此，會將程式碼轉換成不具可攜性。

建立好物件之後，請利用 setBaseTopicName 方法，傳入必要的主題名稱，來設定 baseTopicName 內容。

**註:** 這裡所用的主題名稱採非 JRI 格式，不能併入名稱/值配對。請依照第172頁的『以 'set' 方法設定內容』中所說明，利用 'set' 方法來進行這些設定。下列程式碼利用這個方法來建立主題：

```
// 建立 Topic，使用預設的 MQTopic 建構子
Topic rtTopic = new MQTopic();

// 設定物件內容，使用設定者方法
((MQTopic)rtTopic).setBaseTopicName( "Sport/Football/Spurs/Results" );
((MQTopic)rtTopic).setPersistence(1);
((MQTopic)rtTopic).setPriority(5);
```

**方法 3：使用 session.createTopic(..)**

Topic 物件也可以利用 TopicSession 的 createTopic 方法來建立，它依照下列方式來採用主題 URI：

```
// 建立 Topic，使用階段作業■Factory■方法
Topic rtTopic = session.createTopic( "topic://Sport/Football/Spurs/Results" );
```

**方法 4：使用 session.createTemporaryTopic()**

TemporaryTopic 是只有相同 TopicConnection 所建立的訂閱者才能消費的 Topic。 TemporaryTopic 是依照下列方式來建立的：

```
// 建立 TemporaryTopic，使用階段作業 Factory 方法
Topic rtTopic = session.createTemporaryTopic();
```

## 訂閱者選項

JMS 訂閱者的使用方式有許多種。這一節說明它們的一部份使用範例。

JMS 提供兩類型的訂閱者：

## 訂閱者選項

**不可延續的訂閱者**

只有當訊息發佈時訂閱者在作用中，這些訂閱者才能接收所選主題的訊息。

**可延續的訂閱者**

這些訂閱者會收到某主題的所有發佈訊息，其中包括訂閱者不作用時所發佈的訊息。

### 建立不可延續的訂閱者

第178頁的『建立發佈資訊的消費者和產生者』中所建立的訂閱者是不可延續的訂閱者，它們是利用下列程式碼來建立的：

```
// 建立訂閱者，在給定的主題中訂閱
 TopicSubscriber sub = session.createSubscriber( topic );
```

### 建立可延續的訂閱者

建立可延續的訂閱者非常類似於建立不可延續的訂閱者，不過，您也必須提供用來唯一識別訂閱者的名稱：

```
// 建立可延續的訂閱者，提供唯一的識別名稱
TopicSubscriber sub = session.createDurableSubscriber( topic, "D_SUB_000001" );
```

不可延續的訂閱者會在呼叫它們的 `close()` 方法（當它們不在範圍內）時，自動登錄它們本身。不過，如果您要終止可延續的訂閱，您必須明確通知系統。如果要做到這一點，請利用階段作業的 `unsubscribe()` 方法，再傳入建立訂閱者的唯一名稱。

```
// 取消訂閱以上建立之可延續的訂閱者
session.unsubscribe( "D_SUB_000001" );
```

| 可延續的訂閱者建立在 MQTopicConnectionFactory 佇列管理程式參數所指定的佇列管理
| 程式。如果後來嘗試在不同的佇列管理程式建立同名的可延續訂閱者，這時會傳回全
| 新且完全獨立的可延續訂閱者。

### 使用訊息選取元

您可以利用訊息選取元來過濾出不符合給定基準的訊息。如果需要訊息選取元的詳細資料，請參閱第174頁的『訊息選取元』。訊息選取元會依下列方式而關聯於訂閱者：

```
// 建立訊息選取元與不可延續之訂閱者的關聯性
String selector = "company = 'IBM'";
TopicSubscriber sub = session.createSubscriber( topic, selector, false );
```

### 抑制本端發佈資訊

您可以建立一個會忽略本身的連線中所發佈之發佈資訊的訂閱者。請依照下列方式，將 `createSubscriber` 呼叫的第三個參數設爲 `true`：

```
// 建立不可延續之訂閱者，使用 noLocal 選項集
TopicSubscriber sub = session.createSubscriber( topic, null, true );
```

## 組合訂閱者選項

您可以組合訂閱者差異，因而能在想要之時，建立套用選取元而忽略本端發佈資訊的可延續訂閱者。下列程式碼片段顯示組合選項的使用：

```
// 建立可延續、noLocal 訂閱者，使用適用的選取元
String selector = "company = 'IBM'";
TopicSubscriber sub = session.createDurableSubscriber( topic, "D_SUB_000001",
                                                        selector, true );
```

## 配置基本訂閱者佇列

在 MQ JMS 第 5.2 版中，有兩種方法可用來配置訂閱者：

- 多重佇列的方式

  每個訂閱者都指派有可從中擷取所有訊息的專用佇列。 JMS 會爲每個訂閱者各建立一個新佇列。 這是 MQ JMS 第 1.1 版所提供的唯一方法。

- 共用佇列的方式

  訂閱者使用共用佇列，它與其它訂閱者都從這個共用佇列中擷取其訊息。 這個方式只需要一個佇列來爲多個訂閱者提供服務。 這是 MQ JMS 第 5.2 版所用的預設方法。

在 MQ JMS 第 5.2 版中，您可以選擇要用哪個方法以及配置要使用的佇列。

大體上，共用佇列的方式有適度的效能優點。對於通訊量很高的系統而言，在架構和管理方面的好處也非常大，因爲它大幅減少了所需的佇列數目。

但在某些情況下，仍有恰當原因可讓您使用多重佇列的方式：

- 訊息儲存體在理論上的實體容量比較大。

  MQSeries 佇列不能保留超出 640000 則訊息，在共用佇列的方法中， 這個數目要分攤給共用佇列的所有訂閱者。 對於可延續的訂閱者而言，這個問題比較重要，因爲可延續訂閱者的存活時間通常比不可延續的訂閱者長。 因此，可延續的訂閱者可能會累積比較多訊息。

- 訂閱佇列的外部管理比較容易。

  對某些應用程式類型而言，管理者可能會想監視特定訂閱者佇列的狀態和深度。 當訂閱者和佇列之間是一對一的對映時，這項作業會比較容易進行。

### 預設配置

預設配置使用下列共用訂閱佇列：

- SYSTEM.JMS.ND.SUBSCRIPTION.QUEUE，用於不可延續的訂閱
- SYSTEM.JMS.D.SUBSCRIPTION.QUEUE，用於可延續的訂閱

當您執行 MQJMS_PSQ.MQSC Script 時，會自動建立這些項目。

必要的話，您可以指定替代的實體佇列。您也可以變更配置來使用多重佇列的方式。

### 配置不可延續的訂閱者

您可以採用下列方式之一，來設定不可延續的訂閱者佇列名稱內容：

- 使用 MQ JMS 管理工具（用於 JNDI 擷取的物件）來設定 BROKERSUBQ 內容
- 使用程式中的 setBrokerSubQueue() 方法

對於不可延續的訂閱而言，您提供的佇列名稱開頭應該是下列字元：

SYSTEM.JMS.ND.

如果要選取共用佇列的方式，請指定一個明確的佇列名稱，在這裡，具名佇列是共用佇列所用的佇列。在您建立訂閱之前，您指定的佇列必須已經實際存在。

如果要選取多重佇列的方式，請指定一個結尾爲 * 字元的佇列名稱。之後，用這個佇列名稱建立的每個訂閱者都會建立一個適當的動態佇列，專供該特定訂閱者使用。 MQ JMS 會利用它自己的內部模型佇列來建立這種佇列。 因此，在使用多重佇列的方式時，會動態建立所有必要的佇列。

當使用多重佇列的方式時，您不能指定明確的佇列名稱。不過，您可以指定佇列字首。這讓您能夠建立不同的訂閱者佇列領域。比方說，您可以使用：

SYSTEM.JMS.ND.MYDOMAIN.*

在 * 字元前面的字元會用作字首，因此，這項訂閱所關聯的所有動態佇列都會有開頭爲 SYSTEM.JMS.ND.MYDOMAIN 的佇列名稱。

## 配置可延續的訂閱者

如先前所討論，仍有恰當原因可讓您在可延續的訂閱上使用多重佇列的方式。可延續的訂閱很可能會有較長的生命期限，因而佇列中也比較可能累積大量未擷取的訊息。

因此，可延續的訂閱者佇列名稱內容要設定在 Topic 物件中（也就是說，超出 TopicConnectionFactory 的可管理層次）。這使您能夠指定許多不同的訂閱者佇列名稱，而不需要從 TopicConnectionFactory 開始，重新建立多個物件。

您可以採用下列方式之一，來設定可延續的訂閱者佇列名稱：

- 使用 MQ JMS 管理工具（用於 JNDI 擷取的物件）來設定 BROKERDURSUBQ 內容
- 使用程式中的 setBrokerDurSubQueue() 方法：

```
// 設定 MQTopic 可延續之訂閱者佇列名稱，使用
// 多重佇列方法
sportsTopic.setBrokerDurSubQueue("SYSTEM.JMS.D.FOOTBALL.*");
```

起始設定好 Topic 物件之後，它會傳遞到 TopicSession createDurableSubscriber() 方法中，來建立指定的訂閱：

```
// 建立可延續之訂閱者，使用我們稍早的 Topic
TopicSubscriber sub = new session.createDurableSubscriber
                                          (sportsTopic, "D_SUB_SPORT_001");
```

對於可延續的訂閱而言，您提供的佇列名稱開頭應該是下列字元：

SYSTEM.JMS.D.

如果要選取共用佇列的方式，請指定一個明確的佇列名稱，在這裡，具名佇列是共用佇列所用的佇列。在您建立訂閱之前，您指定的佇列必須已經實際存在。

如果要選取多重佇列的方式，請指定一個結尾爲 * 字元的佇列名稱。之後，用這個佇列名稱建立的每個訂閱者都會建立一個適當的動態佇列，專供該特定訂閱者使用。 MQ JMS 會利用它自己的內部模型佇列來建立這種佇列。 因此，在使用多重佇列的方式時，會動態建立所有必要的佇列。

當使用多重佇列的方式時，您不能指定明確的佇列名稱。不過，您可以指定佇列字首。這讓您能夠建立不同的訂閱者佇列領域。比方說，您可以使用：

SYSTEM.JMS.D.MYDOMAIN.*

在 * 字元前面的字元會用作字首，因此，這項訂閱所關聯的所有動態佇列都會有開頭爲 SYSTEM.JMS.D.MYDOMAIN 的佇列名稱。

### 可延續訂閱者的重建和移轉問題

對於可延續的訂閱者，在刪除訂閱者之前，請勿嘗試重新配置訂閱者佇列名稱。也就是說，先執行 unsubscribe()，再重新建立佇列（請記住，任何舊的訂閱者訊息都會被刪除）。

不過，如果您利用 MQ JMS 第 1.1 版建立了訂閱者，當您移轉至新層次時，會辨識出這個訂閱者。您不需要刪除訂閱。訂閱會繼續採用多重佇列的方式來運作。

## 解決發佈/訂閱問題

這一節說明部署使用發佈/訂閱領域的 JMS 從屬站應用程式時，所可能發生的某些問題。請注意，這一節所討論的問題針對發佈/訂閱領域。請參閱第175頁的『處理錯誤』和第27頁的『解決問題』，以取得其它一般疑難排解指引。

### 發佈/訂閱關閉不完整

當 JMS 從屬站應用程式終止時，它們要歸還所有外部資源，這一點非常重要。如果要做到這一點，請呼叫在不需要之後可予以關閉的所有物件的 `close()` 方法。如果是發佈/訂閱領域，這些物件有：

- TopicConnection
- TopicSession
- TopicPublisher
- TopicSubscriber

MQSeries Class for Java 訊息服務實作藉著使用「聯動式關閉」來簡化這項作業。在使用這個程序時，呼叫某 TopicConnection 的 'close'，結果也會連帶呼叫它所建立的每個 TopicSession 的 'close'。這又會連帶呼叫這些階段作業所建立的所有 TopicSubscriber 和 TopicPublisher 的 'close'。

因此，爲了要確保正確釋出外部資源，務必要呼叫應用程式所建立的每個連線的 `connection.close()`。

在某些情況下，這個 'close' 程序可能會不完整。其中包括：

- 遺失 MQSeries 從屬站連線
- 應用程式非預期的終止

在這些情況下，不會呼叫 `close()`，且外部資源會爲了終止的應用程式而維持開啓狀態。其主要結果如下：

**分配管理程式狀態不一致**

MQSeries 訊息分配管理程式可能含有已不存在的訂閱者和發佈者的登錄資訊。這表示分配管理程式可能會繼續將訊息轉遞給永遠不會收到訊息的訂閱者。

**遺留訂閱者訊息和佇列**

訂閱者取消登錄程序的一部份是移除訂閱者訊息。如果適當的話，也會移除用

來接收訂閱的基礎 MQSeries 佇列。如果沒有正常關閉的話，這些訊息和佇列會留下來。如果分配管理程式狀態不一致的話，這些佇列會繼續填入永遠不會被讀取的訊息。

## 訂閱者清除公用程式

爲了要避免沒有正常關閉訂閱者物件所關聯的問題，MQ JMS 包含一個訂閱者清除公用程式。 當第一個使用某實體佇列管理程式的 TopicConnection 起始設定時，便會就這個佇列管理程式而執行這個公用程式。如果給定佇列管理程式的所有 TopicConnection 都已關閉，當就著這個佇列管理程式而起始設定下個 TopicConnection 時，便會重新執行這個公用程式。

清除公用程式會試圖偵測出其它應用程式所帶來的任何較早的 MQ JMS 發佈/訂閱問題。如果偵測到問題的話，它會利用下列方式來消除相關的資源：

- 就 MQSeries 訊息分配管理程式來進行取消登錄
- 清除訂閱所關聯的任何未擷取的訊息及佇列

清除公用程式以透通方式執行於背景中，且只會持續一小段時間。它不應該影響到其它 MQ JMS 應用程式。如果就給定的佇列管理程式而偵測出許多問題，在清除好資源之後，起始設定時可能會有一小段延遲時間。

**註:** 我們仍強力建議您儘可能正常關閉任何訂閱者物件，以免造成訂閱者問題。

## 處理分配管理程式報告

MQ JMS 實作會使用分配管理程式的訊息來確認登錄和取消登錄指令。這些報告通常都是由 MQSeries Class for Java 訊息服務實作來使用，但在某些錯誤狀況下，它們會保留在佇列中。這些訊息會傳送到本端佇列管理程式的 `SYSTEM.JMS.REPORT.QUEUE` 佇列中。

MQSeries Class for Java 訊息服務提供有 `PSReportDump`，它是一個 Java 應用程式，會採用純文字格式來傾出這個佇列的內容。之後，可由使用者或 IBM 支援人員來分析這項資訊。您也可以在診斷或修正過問題之後，利用這個應用程式來清除訊息佇列。

已編譯格式的工具安裝在 `<MQ_JAVA_INSTALL_PATH>/bin` 目錄中。如果要呼叫工具，請切換至這個目錄，再使用下列指令：

```
java PSReportDump [-m queueManager] [-clear]
```

其中：

**-m queueManager**
= 指定要使用的佇列管理程式名稱

**-clear** = 在傾出內容之後，清除訊息佇列

輸出會送到螢幕中，您也可以將它重新導向到檔案中。

# 第12章 JMS 訊息

JMS 訊息由下列各部份組成：

**標題** 所有訊息都支援同一組標題欄位。標題欄位含有從屬站和提供者用來識別和遞送訊息的值。

**內容** 每個訊息都有一項支援應用程式定義的內容值之內建機能。內容提供一種有效的機制來過濾應用程式定義的訊息。

**主體** JMS 定義許多種訊息主體類型，其中涵蓋了目前在使用中的大多數訊息樣式。

JMS 定義的訊息主體有五種類型：

**串流** Java 的基本值串流。它會循序填入和讀取。

**對映** 一組名稱/值配對，名稱是 Strings，值是 Java 基本類型。這些項目可以循序存取，也可以依名稱而隨機存取。項目次序沒有定義。

**文字** 含有 java.util.String 的訊息。

**物件** 含有可序列化 Java 物件的訊息

**位元組** 未解譯之位元組的串流。這個訊息類型專用於主體的實際編碼，以符合現有的訊息格式。

JMSCorrelationID 標題欄位用來鏈結不同的訊息。它通常會鏈結回覆訊息和它的要求訊息。 JMSCorrelationID 可保留一個提供者特定訊息 ID、一個應用程式特定 String，或一個提供者原生 byte[] 值。

## 訊息選取元

訊息含有支援應用程式定義的內容值之內建機能。事實上，這提供了新增應用程式特定標題欄位到訊息中的機制。內容可讓應用程式透過訊息選取元，使 JMS 提供者利用應用程式特定基準來代表它選取或過濾訊息。應用程式定義的內容必須遵循下列規則：

- 內容名稱必須遵循訊息選取元識別碼的規則。
- 內容值可以是 boolean、byte、short、int、long、float、double 和 string。
- 保留下列名稱字首：JMSX、JMS_。

內容值是在傳送訊息之前設定的。當從屬站收到訊息時，訊息內容是唯讀的。如果從屬站試圖在這時設定內容，這時便會擲出 MessageNotWriteableException。如果呼叫 clearProperties 的話，這時可以讀取和寫入內容。

內容值可以複製或不複製訊息主體中的值。 JMS 不會定義是否應該在內容中做什麼的政策。不過，應用程式開發人員應該記住，JMS 提供者在處理訊息主體中的資料時，比訊息內容中的資料有效。如果要有最佳效能，應用程式應該只在需要自訂訊息標題時才使用訊息內容。這麼做的主要原因，是支援自訂的訊息選項。

## 訊息選取元

JMS 訊息選取元可讓從屬站利用訊息標題來指定它想要的訊息。只有標題符合選取元的訊息才會遞送。

訊息選取元不能參照訊息主體的值。

當訊息標題欄位和內容值代換了選取元內的對應識別碼且選取元得出 true 值時，訊息選取元便符合了某一則訊息。

訊息選取元是一個 String，它的語法基礎是 SQL92 條件表示式語法的一個子集。訊息選取元的求值以優先順序層次內由左而右的順序來進行。您可以利用括弧來變更要順序。預先定義的選取元文字和運算子名稱是用大寫在這裡寫入的；不過，它們並不區分大小寫。

選取元可包含：

- 文字
  - 括在單引號內的字串文字。兩個單引號代表一個單引號。 例如 'literal' 和 'literal''s'。如同 Java 字串文字，這些都使用 Unicode 字元編碼。
  - 完全正確的數值文字是不含小數點的數值，如 57、-957、+62。 Java long 範圍內的數字都有支援。
  - 概略的數值文字是科學記號表示法的數值，如 7E3 或 -57.9E2，或有小數點的數值，如 7.、-95.7 或 +6.2。 Java double 範圍內的數字都有支援。
  - Boolean 文字 TRUE 和 FALSE。
- 識別碼：
  - 識別碼是沒有長度限制的 Java 字母和 Java 數字序列， 其中的第一個必須是 Java 字母。 字母是 Character.isJavaLetter 方法會傳回 true 的任何字元。 其中包括 '_' 和 '$'。 字母或數字是 Character.isJavaLetterOrDigit 方法會傳回 true 的任何字元。
  - 識別碼不能是 NULL、TRUE 或 FALSE 等名稱。
  - 識別碼不能是 NOT、AND、OR、BETWEEN、LIKE、IN 和 IS。
  - 識別碼是標題欄位參照或內容參照。
  - 識別碼會區分大小寫。
  - 訊息標題欄位參照限制爲：
    - JMSDeliveryMode
    - JMSPriority
    - JMSMessageID
    - JMSTimestamp
    - JMSCorrelationID
    - JMSType

    JMSMessageID、JMSTimestamp、JMSCorrelationID 和 JMSType 值可以是空值，如果是空值的話，會當作 NULL 值來處理。
  - 任何開頭爲 'JMSX' 的名稱都是 JMS 定義的內容名稱。
  - 任何開頭爲 'JMS_' 的名稱都是提供者特定內容名稱。
  - 任何開頭不是 'JMS' 的名稱都是應用程式特定內容名稱。 如果訊息中有某內容的參照不存在，它的值即爲 NULL。 如果它不存在，它的值爲對應的內容值。
- 空白和 Java 所定義者相同：空格、水平欄標、換頁及行尾。

- 表示式：
  - 選取元是一個條件表示式。 其值爲 true 的選取元符合，其值爲 false 或不明的選取元不符合。
  - 算術表示式包含它們本身、算術運算、識別碼（其值作爲數值文字來處理）及數值文字。
  - 條件表示式包含它們本身、比較運算及邏輯運算。
- 支援標準括弧 ()，用以設定表示式的求值順序。
- 依照優先順序的邏輯運算子：NOT、AND、OR。
- 比較運算子：=、>、>=、<、<=、<>（不等於）。
  - 類型相同的值才能比較。 但完全正確的數值和概略數值的比較除外，這個比較有效。 （需要的類型轉換由 Java 數值提升規則定義。） 如果試圖比較不同類型的話，選取元永遠爲 false。
  - 字串和 Boolean 比較限制於 = 和 <>。 如果兩個字串含有相同的字元序列，且唯有如此，這時兩個字串相等。
- 依照優先順序的算術運算子：
  - +、-：單運算元。
  - *、/：乘和除。
  - +、-：加和減。
  - 不支援 NULL 值的算術運算。 如果嘗試這種運算的話，完成選取元永遠爲 false。
  - 算術運算必須使用 Java 數值提升。
- arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 和 arithmetic-expr3 比較運算子：
  - age BETWEEN 15 and 19 is equivalent to age >= 15 AND age <= 19.
  - age NOT BETWEEN 15 and 19 is equivalent to age < 15 OR age > 19.
  - 如果 BETWEEN 運算的任何表示式是 NULL，運算值即爲 false。 如果 NOT BETWEEN 運算的任何表示式是 NULL，運算值即爲 true。
- identifier [NOT] IN (string-literal1, string-literal2,...) 比較運算子， 其中 identifier 有 String 或 NULL 值。
  - Country IN (' UK', 'US', 'France')，'UK' 是 true，'Peru' 是 false。 它相當於 (Country = ' UK') OR (Country = ' US') OR (Country = ' France') 這個表示式。
  - Country NOT IN (' UK', 'US', 'France')，'UK' 是 false，'Peru' 是 true。 它相當於 NOT ((Country = ' UK') OR (Country = ' US') OR (Country = ' France')) 這個表示式。
  - 如果 IN 或 NOT IN 運算的識別碼是 NULL，則運算值爲不明的值。
- identifier [NOT] LIKE pattern-value [ESCAPE escape-character] 比較運算子，其中 identifier 有 String 值。 pattern-value 是字串文字，其中 '_' 代表任何單一字元，'%' 代表任何字元序列（包括空序列）。 所有其它字元都代表它們自己。 選用的 escape-character 是一個單字元的字串文字，它的字元用來採 pattern-value 的方式跳出 '_' 和 '%' 的特殊意義。
  - phone LIKE '12%3'，'123' '12993' 是 true，'1234' 是 false。
  - word LIKE 'l_se'，'lose' 是 true，'loose' 是 false。
  - underscored LIKE '_%' ESCAPE '\'，'_foo' 是 true，'bar' 是 false。
  - phone NOT LIKE '12%3'，'123' '12993' 是 false，'1234' 是 true。

  - 如果 LIKE 或 NOT LIKE 運算的 identifier 是 NULL，則運算值爲 unknown。
- identifier IS NULL，測試空標題欄位值或遺漏內容值的比較運算子。
  - prop_name IS NULL.
- identifier IS NOT NULL，測試不是空的標題欄位值或內容值之存在的比較運算子。
  - prop_name IS NOT NULL.

下列訊息選取元會選取訊息類型爲 char、顏色爲 blue，重量超出 2500 磅的訊息：

```
"JMSType = 'car' AND color = 'blue' AND weight > 2500"
```

如上所述，內容值可以是 NULL。含有 NULL 值之選取元表示式的運算由 SQL 92 NULL 語意來定義。以下是這些語意的簡要說明：

- SQL 會將 NULL 值視爲 unknown。
- 含不明的值的比較或運算表示式永遠會產生不明的值。
- IS NULL 和 IS NOT NULL 運算子會將不明的值轉換成 TRUE 和 FALSE 值。

雖然 SQL 支援固定小數點的比較和運算表示式，但 JMS 訊息選取元不會。這是完全正確的數值文字會限制於不含小數點的原因。這也是含小數點的數值會作爲概略數值之替代表示法的原因。

不支援 SQL 備註。

## 對映 JMS 訊息至 MQSeries 訊息

這一節說明本章第一部份說明的 JMS 訊息結構如何對映至 MQSeries 訊息。對於想要進行 JMS 和傳統 MQSeries 應用程式間之訊息轉換的程式設計師而言，這很有用。對於想要操作兩個 JMS 應用程式間之訊息傳輸（例如，訊息分配管理程式實作）的人而言，這也很有用。

MQSeries 訊息由下面三種元件組成：
- MQSeries 訊息描述子 (MQMD)
- MQSeries MQRFH2 標題
- 訊息主體。

MQRFH2 是選用的，在外送訊息併入它，由 JMS Destination 類別中的一個旗號來支配。您可以利用 MQSeries JMS 管理工具來設定這個旗號。由於 MQRFH2 會攜帶 JMS 特定資訊，因而在傳送者知道接收目的地是 JMS 應用程式時，永遠要將它併在訊息中。通常在直接傳送訊息到非 JMS 應用程式（MQSeries 原生應用程式）時，會略過 MQRFH2。這是因爲這種應用程式不預期它的 MQSeries 訊息中會有 MQRFH2。 圖4. 顯示結構的轉換：

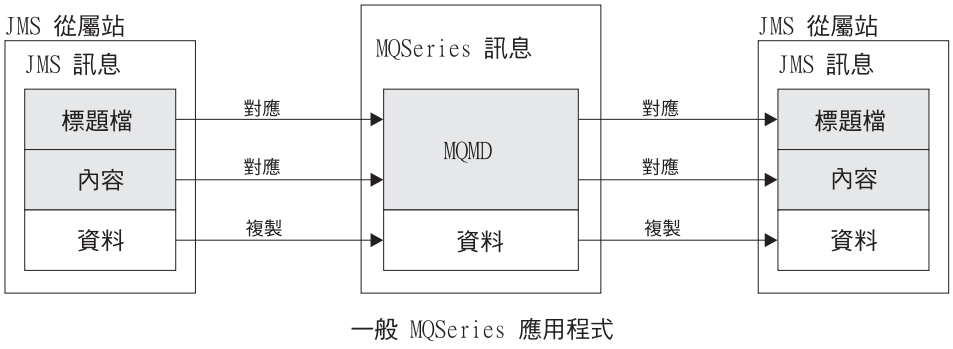


*圖 4. JMS 至 MQSeries 對映模型*

這些結構會採兩種方式來轉換：

**對映** 當 MQMD 併入一個相當於 JMS 欄位的欄位時，JMS 欄位會對映至 MQMD 欄位。其它 MQMD 欄位會陳列成 JMS 內容，因爲與非 JMS 應用程式通信時，JMS 應用程式可能需要取得或設定這些欄位。

**複製** 當沒有 MQMD 對等項時，會傳遞（可能會轉換）一個 JMS 標題欄位或內容來作爲 MQRFH2 內的一個欄位。

## MQRFH2 標題

這一節說明 MQRFH 第 2 版標題，它會攜帶與訊息內容相關的 JMS 特定資料。MQRFH2 第 2 版是可延伸的標題，也可以攜帶不直接關聯於 JMS 的其它資訊。不過，這一節只涵蓋 JMS 所使用者。

標題有兩部份，一個固定部份，一個可變部份。

### 固定部份

固定部份以「標準」MQSeries 標題型樣爲基礎來建立模型，由下列欄位組成：

**StrucId (MQCHAR4)**

結構識別碼。

必須是 MQRFH_STRUC_ID（值：″RFH ″）（起始值）。

MQRFH_STRUC_ID_ARRAY（值：'R','F','H',' '）也用通常的方法來定義。

**Version (MQLONG)**

結構版本號碼。

必須是 MQRFH_VERSION_2（值：2）（起始值）。

**StrucLength (MQLONG)**

MQRFH2 的總長度，其中包括 NameValueData 欄位。

設爲 StrucLength 的值必須是 4 的倍數（NameValueData 欄位中的資料可填補空格字元以符合這一點）。

**Encoding (MQLONG)**

資料編碼。

MQRFH2 後面的訊息部份（下個標題，或這個標題後面的訊息資料）中之任何數值資料的編碼。

**CodedCharSetId (MQLONG)**

編碼字集識別碼。

MQRFH2 後面的訊息部份（下個標題，或這個標題後面的訊息資料）中之任何字元資料的表示法。

**Format (MQCHAR8)**

格式名稱。

MQRFH2 後面的訊息部份的格式名稱。

**Flags (MQLONG)**

旗號。

MQRFH_NO_FLAGS =0。不設定旗號。

**NameValueCCSID (MQLONG)**

NameValueCCSID 字串的編碼字集識別碼 (CCSID) 包含在這個標題中。NameValueData 的編碼所採用的字集可不同於標題所包含的其它字串（StrucID 和 Format）。

如果 NameValueCCSID 是 2 位元組的 Unicode CCSID（1200、13488 或 17584），則 Unicode 的位元組次序和 MQRFH2 中的數值欄位位元組次序相同。（如 Version、StrucLength、NameValueCCSID 本身。）

NameValueCCSID 只能採用下列清單中的值：

| | |
|---|---|
| 1200 | UCS2 開放結尾 |
| 1208 | UTF8 |
| 13488 | UCS2 2.0 子集 |
| 17584 | UCS2 2.1 子集（包括歐元符號） |

### 可變部份

可變部份在固定部份後面。可變部份含有可變的 MQRFH2 資料夾數目。每個資料夾都含有可變的元素或內容數目。資料夾會將相關的內容分組到一起。 JMS 建立的 MQRFH2 標題可含有最多三個資料夾：

**<mcd> 資料夾**

其中包含說明訊息的 'shape' 或 'format' 的內容。比方說，msd 內容會將訊息識別爲 Text、Bytes、Stream、Map、Object 或 'Null'。這個資料夾永遠會出現在 JMS MQRFH2 中。

**<jms> 資料夾**

這用來傳輸 JMS 標題欄位及 MQMD 所無法完整表示的 JMSX 內容。這個資料夾永遠會出現在 JMS MQRFH2 中。

**<usr> 資料夾**

這用來傳輸訊息所關聯的任何應用程式定義的內容。這個資料夾只有在應用程式設定一些應用程式定義的內容時，才會出現。

表17.顯示內容名稱的完整清單。

*表 17. JMS 所用的 MQRFH2 資料夾和內容*

| JMS 欄位 | | MQRFH2 欄位 | | |
|---|---|---|---|---|
| 名稱 | Java 類型 | 資料夾名稱 | 內容名稱 | 類型/值 |
| JMSDestination | Destination | jms | Dst | string |
| JMSExpiration | long | jms | Exp | i8 |
| JMSPriority | int | jms | Pri | i4 |
| JMSDeliveryMode | int | jms | Dlv | i4 |
| JMSCorrelationID | String | jms | Cid | string |
| JMSReplyTo | Destination | jms | Rto | string |
| JMSType | String | mcd | Type | string |
| JMSXGroupID | String | jms | Gid | string |
| JMSXGroupSeq | int | jms | Seq | i4 |
| xxx（使用者定義的） | Any | usr | xxx | any |
| | | mcd | Msd | jms_none<br>jms_text<br>jms_bytes<br>jms_map<br>jms_stream<br>jms_object |

以下是可變部份中用來表示內容的語法：

**NameValueLength (MQLONG)**

緊接在這個長度欄位之後的 NameValueData 字串的位元組長度（不包括它本身的長度）。設爲 NameValueLength 的值永遠是 4 的倍數（NameValueData 欄位會填補空格字元以符合這一點）。

**NameValueData (MQCHARn)**

位元組長度由前面的 NameValueLength 欄位來提供的單一字串。其中含有包留「內容」序列的「資料夾」。每個內容都是一個「名稱/類型/值」三聯組，包含在其名稱爲資料夾名稱的 XML 元素中，如下所示：

```
<foldername> triplet1 triplet2 .....   tripletn </foldername>
```

結尾的 `</foldername>` 標示之後可接著空格作爲填補字元。每個三聯組都用 XML 型的語法來編碼：

```
<name dt='datatype'>value</name>
```

`dt='datatype'` 元素是選用的，許多內容都可以忽略它，因爲它們的資料類型都是預先定義的。如果包含它的話，在 `dt=` 標示之前必須併入一或多個空格字元。

`name` 是內容的名稱 - 請參閱第193頁的表17。

`datatype` 在收合之後，必須符合表18.中的一個文字值。

`value` 是要傳遞之值的字串表示法，如表18.所示。

空值利用下列語法來編碼：

```
<name/>
```

*表 18. 內容資料類型和值*

| 資料類型 | 值 |
|---|---|
| string | 排除 < 和 & 的任何字元序列 |
| boolean | 字元 0 或 1 (1 = ″true″) |
| bin.hex | 代表八進位的十六進位數 |
| i1 | 用 0..9 位數來表示的數字，含選用的正負號（沒有小數或指數）。必須在 -128 至 127 的範圍內，頭尾包括在內。 |
| i2 | 用 0..9 位數來表示的數字，含選用的正負號（沒有小數或指數）。必須在 -32768 至 32767 的範圍內，頭尾包括在內。 |
| i4 | 用 0..9 位數來表示的數字，含選用的正負號（沒有小數或指數）。必須在 -2147483648 至 2147483647 的範圍內，頭尾包括在內。 |
| i8 | 用 0..9 位數來表示的數字，含選用的正負號（沒有小數或指數）。必須在 -9223372036854775808 至 9223372036854775807 的範圍內，頭尾包括在內。 |
| int | 用 0..9 位數來表示的數字，含選用的正負號（沒有小數或指數）。範圍和 'i8' 相同。如果傳送者不要將特定精準度關聯於內容的話，這可用來取代其中一個 'i*' 類型。 |
| r4 | 浮點數字，長度 <= 3.40282347E+38，>= 1.175E-37，利用 0..9 位數來表示，含選用的正負號、選用的小數、選用的指數。 |
| r8 | 浮點數字，長度 <= 1.7976931348623E+308，>= 2.225E-307，利用 0..9 位數來表示，含選用的正負號、選用的小數、選用的指數。 |

字串值可含有空格。您必須在字串值中使用下列跳離序列：

`&`，代表 & 字元

| <，代表 < 字元

您可以利用下列跳離序列，但它們不是必要的：

| &g;，代表 > 字元
&apos，代表 ' 字元
"，代表 " 字元

## JMS 欄位與含對應 MQMD 欄位的內容

表19.列出直接對映至 MQMD 欄位的內容。

*表 19. 對映至 MQMD 欄位的 JMS 內容*

| JMS 欄位 | | MQMD 欄位 | |
|---|---|---|---|
| 標題 | Java 類型 | 欄位 | C 類型 |
| JMSDeliveryMode | int | Persistence | MQLONG |
| JMSExpiration | long | Expiry | MQLONG |
| JMSPriority | int | Priority | MQLONG |
| JMSMessageID | String | MessageID | MQBYTE24 |
| JMSTimestamp | long | PutDate<br>PutTime | MQCHAR8<br>MQCHAR8 |
| JMSCorrelationID | String | CorrelId | MQBYTE24 |
| 內容 | | | |
| JMSXUserID | String | UserIdentifier | MQCHAR12 |
| JMSXAppID | String | PutApplName | MQCHAR28 |
| JMSXDeliveryCount | int | BackoutCount | MQLONG |
| JMSXGroupID | String | GroupId | MQBYTE24 |
| JMSXGroupSeq | int | MsgSeqNumber | MQLONG |
| 提供者特定 | | | |
| JMS_IBM_Report_Exception | int | Report | MQLONG |
| JMS_IBM_Report_Expiration | int | Report | MQLONG |
| JMS_IBM_Report_COA | int | Report | MQLONG |
| JMS_IBM_Report_COD | int | Report | MQLONG |
| JMS_IBM_Report_PAN | int | Report | MQLONG |
| JMS_IBM_Report_NAN | int | Report | MQLONG |
| JMS_IBM_Report_Pass_Msg_ID | int | Report | MQLONG |
| JMS_IBM_Report_Pass_Correl_ID | int | Report | MQLONG |
| JMS_IBM_Report_Discard_Msg | int | Report | MQLONG |
| JMS_IBM_MsgType | int | MsgType | MQLONG |
| JMS_IBM_Feedback | int | Feedback | MQLONG |
| JMS_IBM_Format | String | Format | MQCHAR8 |
| JMS_IBM_PutApplType | int | PutApplType | MQLONG |
| JMS_IBM_Encoding | int | Encoding | MQLONG |
| JMS_IBM_Character_Set | String | CodedCharacterSetId | MQLONG |

## 對映 JMS 訊息

### 對映 JMS 欄位至 MQSeries 欄位（外送訊息）

表20.顯示在 send() 或 publish() 時對映至 MQMD/RFH2 欄位的標題/內容欄位。

如果是標出「訊息物件設定」的欄位，傳輸的值即爲緊接在 send/publish() 前面的 JMS 訊息所保留的值。 send/publish() 會將 JMS 訊息中的值保留不變。

如果是標出「傳送方法設定」的欄位，便會在執行 send/publish() 時指派一個值（會忽略 JMS 訊息所保留的任何值）。 JMS 訊息中的值會更新顯示所用的值。

如果是標出「僅限接收」的欄位，send() 或 publish() 不會予以傳輸，它們會在訊息中原封不動。

*表 20. 外送訊息欄位對映*

| JMS 欄位 | 傳入 | | 設定者 |
|---|---|---|---|
| 名稱 | MQMD 欄位 | 標題 | |
| JMSDestination | | MQRFH2 | 傳送方法 |
| JMSDeliveryMode | Persistence | MQRFH2 | 傳送方法 |
| JMSExpiration | Expiry | MQRFH2 | 傳送方法 |
| JMSPriority | Priority | MQRFH2 | 傳送方法 |
| JMSMessageID | MessageID | | 傳送方法 |
| JMSTimestamp | PutDate/PutTime | | 傳送方法 |
| JMSCorrelationID | CorrelId | MQRFH2 | 訊息物件 |
| JMSReplyTo | ReplyToQ/ReplyToQMgr | MQRFH2 | 訊息物件 |
| JMSType | | MQRFH2 | 訊息物件 |
| JMSRedelivered | | | 僅限接收 |
| 內容 | | | |
| JMSXUserID | UserIdentifier | | 傳送方法 |
| JMSXAppID | PutApplName | | 傳送方法 |
| JMSXDeliveryCount | | | 僅限接收 |
| JMSXGroupID | GroupId | MQRFH2 | 訊息物件 |
| JMSXGroupSeq | MsgSeqNumber | MQRFH2 | 訊息物件 |
| 提供者特定 | | | |
| JMS_IBM_Report_Exception | Report | | 訊息物件 |
| JMS_IBM_Report_Expiration | Report | | 訊息物件 |
| JMS_IBM_Report_COA/COD | Report | | 訊息物件 |
| JMS_IBM_Report_NAN/PAN | Report | | 訊息物件 |
| JMS_IBM_Report_Pass_Msg_ID | Report | | 訊息物件 |
| JMS_IBM_Report_Pass_Correl_ID | Report | | 訊息物件 |
| JMS_IBM_Report_Discard_Msg | Report | | 訊息物件 |
| JMS_IBM_MsgType | MsgType | | 訊息物件 |
| JMS_IBM_Feedback | Feedback | | 訊息物件 |
| JMS_IBM_Format | Format | | 訊息物件 |
| JMS_IBM_PutApplType | PutApplType | | 傳送方法 |
| JMS_IBM_Encoding | Encoding | | 訊息物件 |

*表 20. 外送訊息欄位對映 (繼續)*

| JMS 欄位 | 傳入 | | 設定者 |
|---|---|---|---|
| 名稱 | MQMD 欄位 | 標題 | |
| JMS_IBM_Character_Set | CodedCharacterSetId | | 訊息物件 |

## 在 send()/publish() 對映 JMS 標題欄位

以下是在 send()/publish() 對映 JMS 欄位的相關附註：

- **JMS Destination 至 MQRFH2**： 這會儲存成序列化目的地物件之無聲特性的字串，因此，接收端的 JMS 可以重新建構對等的目的地物件。 MQRFH2 欄位會編碼成 URI（請參閱第171頁的統一資源識別碼，以取得 URI 表示法的詳細資料）。
- **JMSReplyTo 至 MQMD ReplyToQ、ReplyToQMgr、MQRFH2**： Queue 和 QueueManager 名稱會分別複製到 MQMD ReplyToQ 和 ReplyToQMgr 欄位。 目的地延伸資訊（目的地物件中所保留其它「有用的」詳細資料）會複製到 MQRFH2 欄位中。 MQRFH2 欄位會編碼成 URI（請參閱第171頁的統一資源識別碼，以取得 URI 表示法的詳細資料）。

- **JMSDeliveryMode 至 MQMD Persistence**： 除非 Destination 物件予以置換，否則， JMSDeliveryMode 值由 send/publish() 方法或 MessageProducer 來設定。 JMSDeliveryMode 值會依下列方式對映至 MQMD Persistence：
  - JMS 值 PERSISTENT 相當於 MQPER_PERSISTENT
  - JMS 值 NON_PERSISTENT 相當於 MQPER_NOT_PERSISTENT

  如果 JMSDeliveryMode 設爲非預設值，這時也會在 MQRFH2 中編碼遞送模式值。
- **JMSExpiration 至/來自 MQMD Expiry、MQRFH2:** JMSExpiration 會儲存到期時間（現行時間和存活時間的總和），MQMD 會儲存存活時間。 另外，JMSExpiration 以毫秒爲單位，但 MQMD.expiry 以百分之一秒爲單位。
  - 如果 send() 方法設定無限的存活時間，MQMD Expiry 會設定 to MQEI_UNLIMITED， 且 MQRFH2 中不會進行 JMSExpiration 的編碼。
  - 如果 send() 方法設定小於 214748364.7 秒的存活時間（約 7 年），存活時間便會儲存在 MQMD 中。 在 MQRFH2 中，以 i8 值來進行期限和期限時間（以毫秒爲單位）的編碼。
  - 如果 send() 方法將存活時間設爲大於 214748364.7 秒，MQMD.Expiry 會設爲 MQEI_UNLIMITED。 在 MQRFH2 中，會以 i8 值來進行眞正期限時間（以毫秒爲單位）的編碼。
- **JMSPriority 至 MQMD Priority**： 將 JMSPriority 值 (0-9) 直接對映至 MQMD 優先順序值 (0-9)。 如果 JMSPriority 設爲非預設值，這時也會在 MQRFH2 中編碼優先順序值。
- **JMSMessageID 來自 MQMD MessageID**： JMS 送來的所有訊息都有 MQSeries 所指派的唯一訊息識別碼。 在 MQPUT 呼叫之後，會在 MQMD messageId 欄位中傳回指派的值，且會在 JMSMessageID 欄位中傳回給應用程式。 MQSeries messageId 是 24 位元組的二進位值，JMSMessageID 則是 String。 JMSMessageID 的內容包括轉換成 48 個十六進位字元之序列的二進位 messageId，前面再加上 'ID:' 字元。 JMS 提供有可設定來停止產生訊息識別碼的提示。 這個提示會被忽略，在所有情況中，都會指派一個唯一識別碼。 在 send() 之前設定於 JMSMessageId 欄位中的任何值都會被改寫。
- **JMSTimestamp 來自 MQMD PutDate、PutTime**： 在傳送之後，JMSTimestamp 欄位會設成等於 MQMD PutDate 和 PutTime 欄位所提供日期/時間值。 在 send() 之前設定於 JMSMessageId 欄位中的任何值都會被改寫。
- **JMSType 至 MQRFH2**： 這個字串會設於 MQRFH2 中。
- **JMSCorrelationID 至 MQMD CorrelId、MQRFH2**： JMSCorrelationID 可以保留下列其中一項：
  - **提供者特定訊息 ID**： 這是來自先前所傳送或接收之訊息的訊息 ID， 因此，應該是有 48 個十六進位數且字首爲 'ID:' 的字串。 這個字首會移除，其餘的字元會轉換成二進位，之後，它們會設定於 MQMD CorrelId 欄位中。 在 MQRFH2 中不會進行 correlid 值的編碼。
  - **提供者原生 byte[] 值**： 這個值會複製到 MQMD CorrelId 欄位中 - 必要時，會填補空值或截斷成 24 位元組。 在 MQRFH2 中不會進行 correlid 值的編碼。
  - **應用程式特定字串**： 這個值會複製到 MQRFH2 中。 字串的前 24 個位元組（UTF8 格式）會寫入 MQMD CorrelID 中。

## 對映 JMS 內容欄位

這些附註參照 MQSeries 訊息中之 JMS 內容欄位的對映：

- **JMSXUserID 來自 MQMD UserIdentifier**： JMSXUserID 在從傳送呼叫傳回之時設定。
- **JMSXAppID 來自 MQMD PutApplName**： JMSXAppID 在從傳送呼叫傳回之時設定。
- **JMSXGroupID 至 MQRFH2（點對點）**：對於點對點訊息，JMSXGroupID 會複製到 MQMD GroupID 欄位中。 如果 JMSXGroupID 起始於 'ID:' 字首，它會轉換成二進位。 否則，它會編碼成 UTF8 字串。 必要時，這個值會填補或截斷成 24 位元組。 這會設定 MQF_MSG_IN_GROUP 旗號。
- **JMSXGroupID 至 MQRFH2（發佈/訂閱）**：對於發佈/訂閱訊息，JMSXGroupID 會複製到 MQRFH2 中，成爲一個字串。
- **JMSXGroupSeq MQMD MsgSeqNumber（點對點）**： 對於點對點訊息，JMSXGroupSeq 會複製到 MQMD MsgSeqNumber 欄位中。 這會設定 MQF_MSG_IN_GROUP 旗號。
- **JMSXGroupSeq MQMD MsgSeqNumber（發佈/訂閱）**： 對於發佈/訂閱訊息，JMSXGroupSeq 會複製到 MQRFH2 中，成爲一個 i4。

## 對映 JMS 提供者特定欄位

下列附註參照從 JMS 提供者特定欄位至 MQSeries 訊息的對映：

- **JMS_IBM_Report_<name> 至 MQMD Report**： JMS 應用程式可以利用下列 JMS_IBM_Report_XXX 內容來設定 MQMD Report 選項。 單一 MQMD 會對映至多個 JMS_IBM_Report_XXX 內容。 應用程式應該將這些內容的值設爲標準 MQSeries MQRO_ 常數（包括在 com.ibm.mq.MQC 中）。因此，比方說，如果要要求完整資料的 COD，應用程式應該將 JMS_IBM_Report_COD 設爲 MQC.MQRO_COD_WITH_FULL_DATA 值。

  **JMS_IBM_Report_Exception**

  MQRO_EXCEPTION 或
  MQRO_EXCEPTION_WITH_DATA 或
  MQRO_EXCEPTION_WITH_FULL_DATA

  **JMS_IBM_Report_Expiration**

  MQRO_EXPIRATION 或
  MQRO_EXPIRATION_WITH_DATA 或
  MQRO_EXPIRATION_WITH_FULL_DATA

  **JMS_IBM_Report_COA**

  MQRO_COA 或
  MQRO_COA_WITH_DATA 或
  MQRO_COA_WITH_FULL_DATA

  **JMS_IBM_Report_COD**

  MQRO_COD 或
  MQRO_COD_WITH_DATA 或
  MQRO_COD_WITH_FULL_DATA

  **JMS_IBM_Report_PAN**
  MQRO_PAN

**JMS_IBM_Report_NAN**
MQRO_NAN

**JMS_IBM_Report_Pass_Msg_ID**
MQRO_PASS_MSG_ID

**JMS_IBM_Report_Pass_Correl_ID**
MQRO_PASS_CORREL_ID

**JMS_IBM_Report_Discard_Msg**
MQRO_DISCARD_MSG

- **JMS_IBM_MsgType 至 MQMD MsgType**： 值直接對映至 MQMD MsgType。 如果應用程式沒有明確設定 JMS_IBM_MsgType 值，便會使用預設值。 這個預設值的決定方式如下：
  - 如果 JMSReplyTo 設爲 MQSeries 佇列目的地的話，MSGType 設爲 MQMT_REQUEST 值
  - 如果 JMSReplyTo 沒有設定或設爲 MQSeries 佇列目的地以外的任何東西， 則 MsgType 設爲 MQMT_DATAGRAM 值
- **JMS_IBM_Feedback 至 MQMD Feedback**： 值直接對映至 MQMD Feedback。
- **JMS_IBM_Format 至 MQMD Format**： 值直接對映至 MQMD Format。
- **JMS_IBM_Encoding 至 MQMD Encoding**： 如果設定的話，這個內容會置換 Destination Queue 或 Topic 的數值編碼。
- **JMS_IBM_Character_Set 至 MQMD CodedCharacterSetId**： 如果設定的話，這個內容會置換 Destination Queue 或 Topic 的編碼字集內容。

## 對映 MQSeries 欄位至 JMS 欄位（內收訊息）

表21.顯示在 send() 或 publish() 時標題/內容欄位如何對映至 MQMD/MQRFH2 欄位。

*表 21. 內收訊息欄位對映*

| JMS 欄位 | 擷取來源 | |
|---|---|---|
| 名稱 | MQMD 欄位 | MQRFH2 |
| **JMS 標題** | | |
| JMSDestination | | jms.Dst |
| JMSDeliveryMode | Persistence | |
| JMSExpiration | | jms.Exp |
| JMSPriority | Priority | |
| JMSMessageID | MessageID | |
| JMSTimestamp | PutDate PutTime | |
| JMSCorrelationID | CorrelId | jms.Cid |
| JMSReplyTo | ReplyToQ ReplyToQMgr | jms.Rto |
| JMSType | | mcd.Type |
| JMSRedelivered | BackoutCount | |
| **JMS 內容** | | |
| JMSXUserID | UserIdentifier | |
| JMSXAppID | PutApplName | |
| JMSXDeliveryCount | BackoutCount | |

*表 21. 內收訊息欄位對映 (繼續)*

| JMS 欄位 | 擷取來源 | |
|---|---|---|
| 名稱 | MQMD 欄位 | MQRFH2 |
| JMSXGroupID | GroupId | jms.Gid |
| JMSXGroupSeq | MsgSeqNumber | jms.Seq |
| JMS 提供者特定 | | |
| JMS_IBM_Report_Exception | Report | |
| JMS_IBM_Report_Expiration | Report | |
| JMS_IBM_Report_COA | Report | |
| JMS_IBM_Report_COD | Report | |
| JMS_IBM_Report_PAN | Report | |
| JMS_IBM_Report_NAN | Report | |
| JMS_IBM_Report_ Pass_Msg_ID | Report | |
| JMS_IBM_Report_Pass_Correl_ID | Report | |
| JMS_IBM_Report_Discard_Msg | Report | |
| JMS_IBM_MsgType | MsgType | |
| JMS_IBM_Feedback | Feedback | |
| JMS_IBM_Format | Format | |
| JMS_IBM_PutApplType | PutApplType | |
| JMS_IBM_Encoding[1] | Encoding | |
| JMS_IBM_Character_Set[1] | CodedCharacterSetId | |
| 1. 只有當內收訊息是 Bytes Message 時才設定。 | | |

## 對映 JMS 至原生 MQSeries 應用程式

這一節說明從 JMS 從屬站應用程式傳送一則訊息至不懂 MQRFH2 標題的傳統 MQSeries 應用程式時，會發生什麼。 第202頁的圖5.是對映的圖解。

管理者將 MQSeries Destination 的 TargetClient 值設成 JMSC.MQJMS_CLIENT_NONJMS_MQ，指出 JMS 從屬站與這種應用程式在通信。這表示不會產生任何 MQRFH2 欄位。

從 JMS 至目標爲原生 MQSeries 應用程式之 MQMD 的對映和從 JMS 至目標爲眞正 JMS 從屬站的對映相同。如果 JMS 接收了 MQSeries 訊息，且訊息中含有 MQMD 格式欄位，而不是 MQFMT_RFH2，我們會知道接收的資料來自非 JMS 應用程式。如果 Format 是 MQFMT_STRING，這時會將訊息當作 JMS Text Message 來接收。否則，會將它當作 JMS Bytes 訊息來接收。由於沒有 MQRFH2，因而只會還原 MQMD 中所傳輸的 JMS 內容。

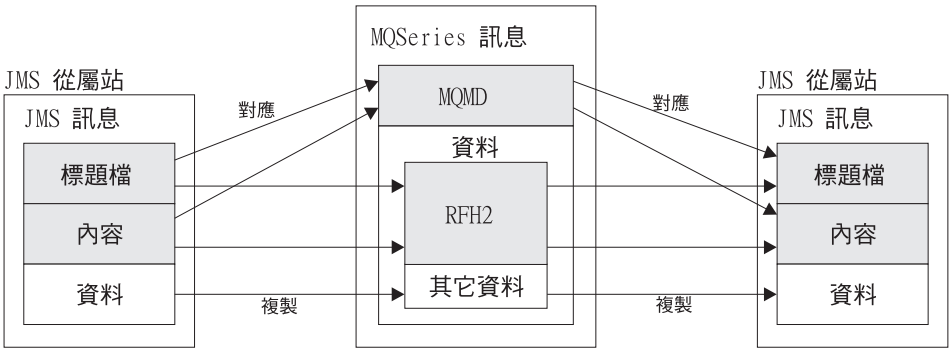


*圖 5. JMS 至 MQSeries 對映模型*

## 訊息主體

這一節討論訊息主體本身的編碼。編碼會隨著 JMS 訊息類型而不同：

**ObjectMessage**

是 Java Runtime 依正常方式來序列化的物件。

**TextMessage**

是編碼字串。如果是外送訊息，字串會以 Destination 物件所提供的字集來編碼。這會預設爲 UTF8 編碼（UTF8 編碼起始於訊息的第一個字元 - 開始時沒有長度欄位）。不過，您也可以指定 MQ Java 所支援的任何其它字集。這些字集主要在您傳送訊息給非 JMS 應用程式之時使用。

如果字集是雙位元組字集（包括 UTF16），Destination 物件的整數編碼規格會決定位元組的次序。

內收訊息利用訊息本身所指定的字集和編碼來解譯。這些規格在最右側的 MQSeries 標題（如果沒有標題的話，則爲 MQMD）。對於 JMS 訊息而言，最右側標題通常是 MQRFH2。

**BytesMessage**

依預設，是 JMS 1.0.2 規格及相關 Java 文件所定義的位元組序列。

對於應用程式本身所組譯的外送訊息而言，Destination 物件的編碼內容可用來置換訊息中所包含的整數及浮點欄位編碼。比方說，您可以要求將這個浮點值儲存在 S/390，而不儲存成 IEEE 格式）。

內收訊息利用訊息本身所指定的數值編碼來解譯。這個規格在最右側的 MQSeries 標題（如果沒有標題的話，則爲 MQMD）。對於 JMS 訊息而言，最右側標題通常是 MQRFH2。

如果收到 BytesMessage，且沒有修改便又傳送出去，它的主體會依照它所接收者而進行位元組對位元組的傳輸。 Destination 物件的編碼內容對主體沒有作用。 BytesMessage 唯一能明確傳送的字串型實體是 UTF8 字串。這採用 Java UTF8 格式來編碼，起始於 2 位元組的長度欄位。 Destination 目的地之字集內容對送出 BytesMessage 的編碼沒有作用。送入 MQSeries 訊息中的字集值對於將這個訊息解譯成 JMS BytesMessage 沒有作用。

非 Java 應用程式不太可能認得 Java UTF8 編碼。因此，對於要傳送含文字資料的 BytesMessage 之 JMS 應用程式而言，應用程式本身必須將它的字串轉換成位元組陣列，並將位元組陣列寫入 BytesMessage 中。

**MapMessage**

是含有「XML 名稱/類型/值」三聯組的字串，編碼爲：

```
<map><elementName1 dt='datatype'>value</elementName1>
<elementName2 dt='datatype'>value</elementName2>.....
</map>
```

其中：

`datatype` 可採用第194頁的表18.中所說明的其中一個值。
`string` 是預設資料類型，因此，會忽略 `dt='string'`。

用來編碼或解譯組成 MapMessage 主體的 XML 字串之字集，由適用於 TextMessage 的規則來決定。

**StreamMessage**

如同對映，但沒有元素名稱 ：

```
<stream><elt dt='datatype'>value</elt>
<elt dt='datatype'>value</elt>.....</stream>
```

每個元素都用相同標示名稱 (elt) 來傳送的。預設類型是字串，因此，會忽略字串元素的 `dt='string'`。

用來編碼或解譯組成 StreamMessage 主體的 XML 字串之字集，由適用於 TextMessage 的規則來決定。

MQRFH2 格式欄位會設定如下：

**MQFMT_NONE**

適用於 ObjectMessage、BytesMessage 或沒有主體的訊息。

**MQFMT_STRING**

適用於 TextMessage、StreamMessage 或 MapMessage。

# 第13章 MQ JMS 應用程式伺服器機能

MQ JMS 第 5.2 版支援 Java 訊息服務 1.0.2 規格（請參閱 Sun 的 Java，網址如下：http://java.sun.com）所指定的「應用程式伺服器機能 (ASF)」。這個規格指出了這個程式設計模型內三個角色：

- **JMS 提供者**，提供 ConnectionConsumer 和進階的 Session 功能。
- **應用程式伺服器**，提供 ServerSessionPool 和 ServerSession 功能。
- **從屬站應用程式**，使用 JMS 提供者和應用程式伺服器所提供的功能。

下列各節含有 MQ JMS 如何實作 ASF 的詳細資料：

- 『ASF 類別和功能』，說明 MQ JMS 如何實作 ConnectionConsumer 類別和 Session 類別中的進階功能。
- 第211頁的『應用程式伺服器範例程式碼』，說明 MQ JMS 所提供的範例 ServerSessionPool 和 ServerSession 程式碼。
- 第214頁的『ASF 使用範例』，說明提供的 ASF 範例及從屬站應用程式觀點下的 ASF 使用範例。

**註:** ASF 的 Java 訊息服務 1.0.2 規格也說明使用 X/Open XA 通信協定的分散式異動 JMS 支援。如果需要 MQ JMS 提供的 XA 支援的詳細資料，請參閱第351頁的『附錄E. 使用 WebSphere 的 JMS JTA/XA 介面』。

## ASF 類別和功能

MQ JMS 會實作 ConnectionConsumer 類別和 Session 類別中的進階功能。如果需要詳細資料，請參閱：

- 第121頁的『MQPoolServices』
- 第122頁的『MQPoolServicesEvent』
- 第124頁的『MQPoolToken』
- 第151頁的『MQPoolServicesEventListener』
- 第239頁的『ConnectionConsumer』
- 第284頁的『QueueConnection』
- 第297頁的『Session』
- 第315頁的『TopicConnection』

### ConnectionConsumer

JMS 規格使應用程式伺服器能夠利用 ConnectionConsumer 介面而與 JMS 實作密切整合起來。這個特性提供訊息的並行處理功能。通常，應用程式伺服器會建立一個執行緒儲存池，JMS 實作使這些執行緒能夠得到這些訊息。具有 JMS 感應功能的應用程式伺服器可以利用這個特性來提供高階的傳訊功能，如訊息處理 Bean。

一般應用程式不會用到 ConnectionConsumer，但專家級的 JMS 從屬站可能會用到它。ConnectionConsumer 可爲這些從屬站提供高效能的方法來同時遞送若干訊息到執行緒儲

存池中。當訊息到達某佇列或主題時，JMS 會從儲存池中選取一個執行緒，遞送一批訊息給它。爲了做到這一點，JMS 會執行相關的 MessageListener 的 `onMessage()` 方法。

您可以建構多個 Session 和 MessageConsumer 物件，每個物件都有一個登錄的 MessageListener，來得到相同的結果。不過，ConnectionConsumer 的效能比較好，資源用得比較少，彈性也比較大。特別是，需要的 Session 物件比較少。

爲協助您開發使用 ConnectionConsumers 的應用程式，MQ JMS 提供了儲存池的全功能範例實作。您可以原封不動使用這個實作，或調整它來配合應用程式的特定需求。

## 規劃應用程式

### 點對點傳訊的一般原則

當應用程式從 QueueConnection 物件中建立起 ConnectionConsumer 時，它會指定一個 JMS Queue 物件和一個選取元字串。之後，ConnectionConsumer 會開始接收訊息（更精確地說，提供訊息給相關 ServerSessionPool 中的 Session）。訊息會送達佇列，如果符合選取元的話，會將它們遞送給相關 ServerSessionPool 中的 Session。

用 MQSeries 的話來說，Queue 物件指本端佇列管理程式中的 QLOCAL 或 QALIAS。如果它是 QALIAS，這個 QALIAS 必須指向 QLOCAL。完整解析過的 MQSeries QLOCAL 稱爲 *基礎 QLOCAL*。如果 ConnectionConsumer 沒有關閉且已啓動它的母項 QueueConnection 的話，便稱這個 ConnectionConsumer 是在*作用中*。

多個 ConnectionConsumer 有可能就著相同的基礎 QLOCAL 來執行，每個都有不同的選取元。爲了要維護效能，不要的訊息不應積累在佇列中。不要的訊息是任何作用中的 ConnectionConsumer 都沒有相符選取元的訊息。您可以設定 QueueConnectionFactory，使佇列移除這些不要的訊息（如果需要詳細資料，請參閱第208頁的『從佇列中移除訊息』）。您可以採用下列方式之一來設定這個行爲：

- 利用 JMS 管理工具，將 QueueConnectionFactory 設爲 MRET(NO)。
- 在您的程式中，使用：

  ```
  MQQueueConnectionFactory.setMessageRetention(JMSC.MQJMS_MRET_NO)
  ```

如果您沒有變更這個設定，預設值是將這些不要的訊息保留在佇列中。

從多個 QueueConnection 物件中，有可能建立起多個將目標設爲相同基礎 QLOCAL 的 ConnectionConsumer。不過，爲了效能，建議您不要讓多個 JVM 就相同的基礎 QLOCAL 來建立多個 ConnectionConsumer。

當您設定 MQSeries 佇列管理程式時，請考慮下面這幾點：

- 必須啓用共用輸入的基礎 QLOCAL。 如果要做到這一點，請使用下列 MQSC 方法：

  ```
  ALTER QLOCAL(your.qlocal.name) SHARE GET(ENABLED)
  ```

- 您的佇列管理程式必須有一個已啓用的無法傳送的郵件佇列。 如果 ConnectionConsumer 將訊息放入無法傳送的郵件佇列時發生問題， 基礎 QLOCAL 發出的訊息遞送會停止作業。 如果要定義無法傳送的郵件佇列，請使用：

  ```
  ALTER QMGR DEADQ(your.dead.letter.queue.name)
  ```

- 執行 ConnectionConsumer 的使用者必須有權以 MQOO_SAVE_ALL_CONTEXT 和 MQOO_PASS_ALL_CONTEXT 來執行 MQOPEN。 如果需要詳細資料，請參閱特定平台所適用的 MQSeries 文件。

- 如果不要的訊息保留在佇列中，它們會造成系統效能的退化。 因此，請規劃您的訊息選取元，使 ConnectionConsumer 會在它們之間移除佇列中的所有訊息。

如果需要 MQSC 指令的詳細資料，請參閱 *MQSeries MQSC 指令參考手冊*。

## 發佈/訂閱傳訊的一般原則

當應用程式從 TopicConnection 物件中建立起 ConnectionConsumer 時，它會指定一個 Topic 物件和一個選取元字串。之後，ConnectionConsumer 會開始接收符合選取元之 Topic 的相關訊息。

另外，應用程式也可以建立關聯於特定名稱的可延續 ConnectionConsumer。這個 ConnectionConsumer 會接收在可延續的 ConnectionConsumer 前次作用之後，又就這個 Topic 發佈的訊息。它會接收這個 Topic 的所有符合選取元的這些訊息。

就不可延續的訂閱而言，ConnectionConsumer 訂閱會使用個別的佇列。TopicConnectionFactory 的 CCSUB 可配置選項會指定要使用的佇列。通常，CCSUB 應該指定單一佇列供所有使用相同 TopicConnectionFactory 的 ConnectionConsumer 使用。不過，指定佇列名稱字首後面再接著 '*'，也可能使每個 ConnectionConsumer 都產生一個暫時佇列。

就可延續的訂閱而言，Topic 的 CCDSUB 內容會指定要使用的佇列。同樣，這可以是已存在的佇列，或後面接著 '*' 的佇列名稱字首。如果您指定已存在的佇列的話，所有訂閱這個 Topic 的可延續 ConnectionConsumer 都會使用這個佇列。如果您指定後面接著 '*' 的佇列名稱字首的話，在第一次以給定名稱建立可延續的 ConnectionConsumer 時會產生一個佇列。後來在用相同名稱建立可延續的 ConnectionConsumer 時，會重複使用這個佇列。

當您設定 MQSeries 佇列管理程式時，請考慮下面這幾點：

- 您的佇列管理程式必須有一個已啓用的無法傳送的郵件佇列。 如果 ConnectionConsumer 將訊息放入無法傳送的郵件佇列時發生問題， 基礎 QLOCAL 發出的訊息遞送會停止作業。 如果要定義無法傳送的郵件佇列，請使用：

  ```
  ALTER QMGR DEADQ(your.dead.letter.queue.name)
  ```

- 執行 ConnectionConsumer 的使用者必須有權以 MQOO_SAVE_ALL_CONTEXT 和 MQOO_PASS_ALL_CONTEXT 來執行 MQOPEN。 如果需要詳細資料，請參閱特定平台所適用的 MQSeries 文件。
- 您可以建立個別專用的佇列給個別的 ConnectionConsumer，以最佳化它的效能。 代價是要耗用額外的資源。

## 處理有害訊息

有時佇列會收到格式不正確的訊息。這種訊息會使接收端應用程式失效，並回復收到的訊息。這時候，可能會反覆收到這個訊息及傳回給佇列。這些訊息稱爲*有害訊息*。ConnectionConsumer 必須能夠偵測出有害訊息，將它們重新遞送到另一個目的地。

當應用程式使用 ConnectionConsumer 時，訊息回復的情況會隨著應用程式伺服器提供的 Session 而不同：

- 如果是含 AUTO_ACKNOWLEDGE 或 DUPS_OK_ACKNOWLEDGE 的無異動 Session， 只會在發生系統錯誤或應用程式非預期地終止時回復訊息。
- 如果是含 CLIENT_ACKNOWLEDGE 的無異動 Session，呼叫 `Session.recover()` 的應用程式伺服器可回復未識別的訊息。

MessageListener 的從屬站實作或應用程式伺服器通常會呼叫 Message.acknowledge()。Message.acknowledge() 會認可階段作業在此之前所遞送的所有訊息。

- 如果是異動的 Session，應用程式伺服器通常會確定 Session。 如果應用程式伺服器偵測到錯誤，它可能會選擇回復一或多個訊息。
- 如果應用程式伺服器提供 XASession，這時會根據分散式異動來確定或回復訊息。 應用程式伺服器要負責完成異動。

MQSeries 佇列管理程式會保留每個訊息回復的次數記錄。當這個數字到達可配置的臨界值時，ConnectionConsumer 會重新將訊息放入名稱爲 Backout Queue 的佇列中。如果重新放入佇列因任何原因而失敗，便會從佇列中移除訊息，並重新放入無法傳送的郵件佇列中或予以捨棄。請參閱『從佇列中移除訊息』，以取得詳細資料。

在大多數平台中，臨界值和重入佇列都是 MQSeries QLOCAL 的內容。如果是點對點傳訊，這應該是基礎 QLOCAL。如果是傳訊傳訊，這是 TopicConnectionFactory 所定義的 CCSUB 佇列，或是 Topic 所定義的 CCSUB 佇列。如果要設定臨界值和重入佇列的 Queue 內容，請發出下列 MQSC 指令：

```
ALTER QLOCAL(your.queue.name) BOTHRESH(threshold) BOQUEUE(your.requeue.queue.name)
```

如果是發佈/訂閱傳訊，且您的系統爲每項訂閱各建立一個動態佇列，便會從 MQ JMS 模型佇列中取得這些設定。如果要變更這些設定，您可以利用：

```
ALTER QMODEL(SYSTEM.JMS.MODEL.QUEUE) BOTHRESH(threshold) BOQUEUE(your.requeue.queue.name)
```

如果臨界值是零，便會停止處理有害訊息，有害訊息會保留在輸入佇列中。否則，當回復計數到達臨界值時，訊息會送到用來重入佇列的具名佇列中。如果回復計數到達臨界值，但訊息無法進入用來重入佇列的佇列中，訊息會送到無法傳送的郵件佇列中或予以捨棄。如果沒有定義用來重入佇列的佇列，或 ConnectionConsumer 無法傳送訊息給用來重入佇列的佇列，便會發生這個狀況。在某些平台中，您不能指定臨界值和用來重入佇列的佇列等內容。在這些平台中，當回復計數到達 20 時，訊息會送到無法傳送的郵件佇列中，或予以捨棄。請參閱『從佇列中移除訊息』，以取得進一步的詳細資料。

## 從佇列中移除訊息

當應用程式使用 ConnectionConsumer 時，在若干情況中，JMS 可能需要從佇列中移除訊息：

**格式錯誤的訊息**

可能會送來 JMS 無法處理的訊息。

**有害訊息**

訊息可能到達回復臨界值，但 ConnectionConsumer 無法在回復佇列中讓它重入佇列。

**不想要的 ConnectionConsumer**

在點對點傳訊中，QueueConnectionFactory 設定成不會保留不要的訊息，任何一個 ConnectionConsumer 都不接受不要的訊息。

在這些狀況中，ConnectionConsumer 會試圖從佇列中移除訊息。訊息 MQMD 報告欄位中的處理選項會設定正確的行爲。以下是這些選項：

**MQRO_DEAD_LETTER_Q**

訊息會重入佇列，放到佇列管理程式無法傳送的郵件佇列中。此爲預設值。

**MQRO_DISCARD_MSG**
捨棄訊息。

ConnectionConsumer 也會產生一則報告訊息，而這也相依於訊息 MQMD 的報告欄位。這個訊息會傳送給訊息在 ReplyToQmgr 中的 ReplyToQ。如果傳送報告訊息時發生錯誤，訊息會送往無法傳送的郵件佇列中。訊息 MQMD 報告欄位中的異常狀況報告選項會設定報告訊息的詳細資料。以下是這些選項：

**MQRO_EXCEPTION**
產生含有原始訊息之 MQMD 的報告訊息。它不包含任何訊息主體資料。

**MQRO_EXCEPTION_WITH_DATA**
產生含有 MQMD、任何 MQ 標題及 100 位元組主體資料的報告訊息。

**MQRO_EXCEPTION_WITH_FULL_DATA**
產生含有原始訊息所有資料的報告訊息。

**default**
不產生任何報告訊息。

當產生報告訊息時，會接受下列選項：

- MQRO_NEW_MSG_ID
- MQRO_PASS_MSG_ID
- MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQRO_PASS_CORREL_ID

如果 ConnectionConsumer 沒有遵循訊息 MQMD 中的處理選項或異常狀況報告選項，它的動作會相依於訊息的持續性。如果訊息不具持續性，這時會捨棄訊息，且不會產生任何報告訊息。如果訊息具持續性，則 QLOCAL 送出的所有訊息都會停止遞送。

因此，定義無法傳送的郵件及定期檢查它以確定沒有發生問題，非常重要。尤其是要確保無法傳送的郵件佇列不會到達它的深度上限，且訊息大小上限夠大，足以容納所有訊息。

當訊息重入佇列而放到無法傳送的郵件佇列時，它前面會有 MQSeries 無法傳送的郵件標題 (MQDLH)。請參閱 *MQSeries Application Programming Reference*，以取得 MQDLH 格式的詳細資料。您可以利用下列欄位來識別 ConnectionConsumer 放在無法傳送的郵件佇列中的訊息，或 ConnectionConsumer 已產生的報告訊息：

- PutApplType 是 `MQAT_JAVA (0x1C)`
- PutApplName 是 “`MQ JMS ConnectionConsumer`”

這些欄位都在無法傳送的郵件佇列中之訊息的 MQDLH 及報告訊息的 MQMD 中。MQMD 傳回的訊息欄位及 MQDLH 的「原因」欄位含有說明錯誤的代碼。如果需要這些代碼的詳細資料，請參閱第210頁的『錯誤處理』。請它欄位的說明，請參閱 *MQSeries Application Programming Reference*。

## 錯誤處理

### 錯誤狀況復原

如果 ConnectionConsumer 遇到嚴重問題，遞送給所有想要相同 QLOCAL 的 ConnectionConsumer 之訊息都會停止。通常，如果 ConnectionConsumer 無法讓訊息重入佇列進入無法傳送的郵件佇列中，或 ConnectionConsumer 在讀取 QLOCAL 中的訊息而發生問題時，會發生這個情況。

當發生這個情況時，會利用下列方式來通知應用程式和應用程式伺服器：

- 通知受影響之 Connection 所登錄的任何 ExceptionListener。

您可以利用這些來識別出問題的原因。在某些情況下，系統管理者必須介入，才能解決問題。

應用程式從這些錯誤狀況中復原的方式有兩種：

- 呼叫所有受影響的 ConnectionConsumer 的 `close()`。 只有在所有受影響的 ConnectionConsumer 都已關閉且已解決了任何系統問題之後， 應用程式才能建立新的 ConnectionConsumer。
- 呼叫所有受影響的 Connection 的 `stop()`。 所有 Connection 都已停止且任何系統問題都已解決之後，應用程式應該能夠順利 `start()` 所有 Connectoin。

### 原因碼和傳回的訊息碼

如果要判斷錯誤原因，您可以利用：

- 任何報告訊息中之傳回的訊息碼
- 無法傳送的郵件佇列中任何訊息的 MQDLH 中之原因碼

ConnectionConsumer 會產生下列原因碼。

**MQRC_BACKOUT_THRESHOLD_REACHED (0x93A; 2362)**

**原因** 訊息到達 QLOCAL 所定義的「回復臨界值」，但沒有定義「回復佇列」。

在無法定義「回復佇列」的平台中，訊息會到達 JMS 定義的回復臨界值 20。

**動作** 如果要避免這個狀況，請確定使用佇列的 ConnectionConsumer 會提供一組選取元來處理訊息，或提供一組 QueueConnectionFactory 來保留訊息。

另外，也要探究訊息的來源。

**MQRC_MSG_NOT_MATCHED (0x93B; 2363)**

**原因** 在點對點傳訊中，有一則訊息不符合監視佇列的 ConnectionConsumer 的任何選取元。爲了維護效能，訊息已重入佇列，放到無法傳送的郵件佇列中。

**動作** 如果要避免這個狀況，請確定使用佇列的 ConnectionConsumer 會提供一組選取元來處理訊息，或提供一組 QueueConnectionFactory 來保留訊息。

另外，也要探究訊息的來源。

**MQRC_JMS_FORMAT_ERROR (0x93C; 2364)**

**原因** JMS 無法解譯佇列中的訊息。

**動作** 探究訊息來源。 JMS 通常會以 BytesMessage 或 TextMessage 來遞送非預期格式的訊息。有時候，如果訊息格式嚴重錯誤的話，這會失敗。

這些欄位中所出現的其它代碼，原因是試圖讓訊息重入佇列到「回復佇列」中，但卻失敗。在這個狀況中，代碼會說明重入佇列失敗的原因。如果要診斷這些錯誤的原因，請參閱 *MQSeries Application Programming Reference*。

如果報告訊息無法放到 ReplyToQ，它會放到無法傳送的郵件佇列中。在這個狀況下，會依照上述說明來填入 MQMD 傳回的訊息欄位。 MQDLH 中的原因欄位說明報告訊息無法放入 ReplyToQ 的原因。

## 應用程式伺服器範例程式碼

第212頁的圖6.是 ServerSessionPool 和 ServerSession 功能的摘要。

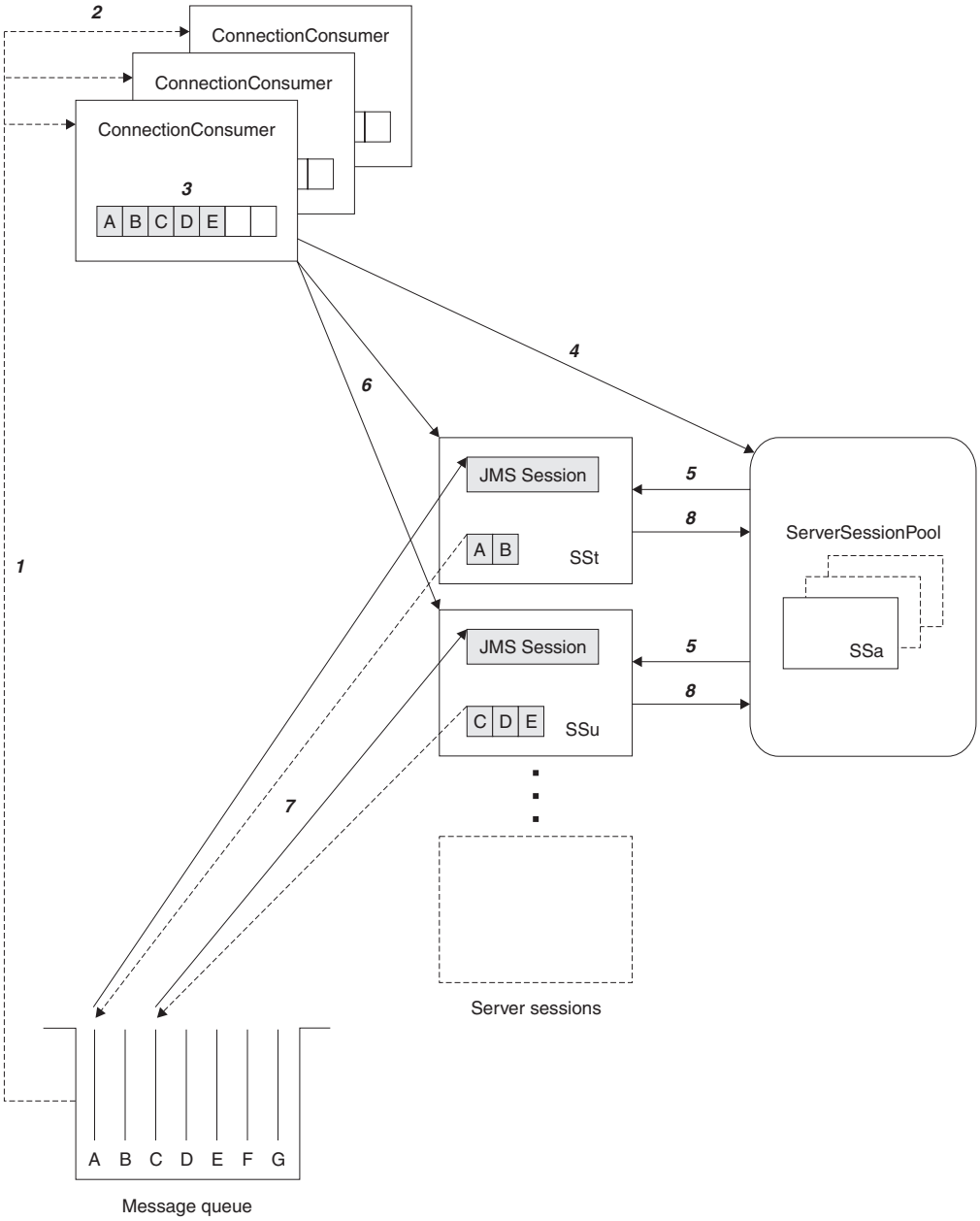


*圖 6. ServerSessionPool 和 ServerSession 功能*

1. ConnectionConsumer 從佇列中取得訊息參照。
2. 每個 ConnectionConsumer 都選取特定的訊息參照。
3. ConnectionConsumer 緩衝區保留選取的訊息參照。
4. ConnectionConsumer 從 ServerSessionPool 中要求一或多個 ServerSession。
5. ServerSessions 從 ServerSessionPool 來進行配置。
6. ConnectionConsumer 將訊息參照指派給 ServerSession，並啓動執行 ServerSession 執行緒。
7. 每個 ServerSession 都從佇列中擷取它的被參照訊息。 它會從 JMS Session 所關聯的 MessageListener 中，將它們傳遞到 `onMessage` 方法中。
8. 處理完成之後，ServerSession 會回到儲存池中。

通常，應用程式伺服器會提供 ServerSessionPool 和 ServerSession 功能。不過，MQ JMS 會隨著程式原始碼而取得這些介面的簡單實作。這些範例在下列目錄中，其中 <install_dir> 是 MQ JMS 的安裝目錄：

```
<install_dir>/samples/jms/asf
```

這些範例可讓您在獨立環境中使用 MQ JMS ASF（也就是說，您不需要有適當的應用程式伺服器）。另外，它們也提供如何實作這些介面及利用 MQ JMS ASF 的範例。這些範例是要協助 MQ JMS 使用者及其它應用程式伺服器的供應商。

## MyServerSession.java

這個類別實作 `javax.jms.ServerSession` 介面。它的基本功能是建立執行緒與 JMS Session 的關聯。 ServerSessionPool 會將這個類別的案例放在儲存池中（請參閱『MyServerSessionPool.java』）。作爲一個 ServerSession，它必須實作下面這兩個方法：

- `getSession()`，這會傳回這個 ServerSession 所關聯的 JMS Session
- `start()`，這會啓動這個 ServerSession 的執行緒，結果會呼叫 JMS Session 的 `run()` 方法

MyServerSession 也會實作 `Runnable` 介面。因此，ServerSession 執行緒的建立可以根據這個類別，不需要個別的類別。

這個類別會使用以 ready 和 quit 這兩個 Boolean 旗號值爲基礎的 `wait()-notify()` 機制。這個機制表示 ServerSession 會在其建構期間建立和啓動它的相關執行緒。不過，它不會自動執行 `run()` 方法的主體。只有當 `start()` 方法將 ready 旗號設爲 true 時，才會執行 `run()` 方法的主體。 ASF 會在需要遞送訊息到相關的 JMS Session 時，呼叫 `start()` 方法。

在遞送時，會呼叫 JMS Session 的 `run()` 方法。這時，MQ JMS ASF 已隨訊息而載入了 `run()` 方法。

遞送好之後，ready 旗號會重設爲 false，擁有的 ServerSessionPool 會收到遞送已經完成的通知。之後，ServerSession 會保留在等待狀態中，直到重新呼叫 `start()` 方法爲止，或直到呼叫 `close()` 方法並結束這個 ServerSession 的執行緒爲止。

## MyServerSessionPool.java

這個類別實作 `javax.jms.ServerSessionPool` 介面，作用是建立和控制 ServerSession 儲存池的存取。

在這個簡單的實作中，儲存池由建構儲存池時所建立的一個靜態 ServerSession 物件陣列組成。下面這四個參數會傳遞到建構子中：

- `javax.jms.Connection` *`connection`*

  用來建立 JMS Session 的連線。
- `int` *`capacity`*

  MyServerSession 物件的陣列大小。
- `int` *`ackMode`*

  所需要的 JMS Session 認可模式。
- `MessageListenerFactory` *`mlf`*

建立提供給 JMS Session 之訊息接聽器的 MesssageListenerFactory。請參閱『MessageListenerFactory.java』。

儲存池建構子會利用這些參數來建立 MyServerSession 物件的陣列。提供的連線用來建立給定認可模式和正確領域的 JMS Session（點對點 QueueSession，發佈/訂閱是 TopicSession）。這些 Session 的提供會附隨訊息接聽器。最後，會根據 JMS Session 來建立 ServerSession 物件。

這個範例實作是一個靜態模型。也就是說，當建立儲存池時，會建立儲存池中的所有 ServerSession，之後，儲存池便無法增加或縮減。這個方式只是爲了簡單。如果需要的話，ServerSessionPool 也可以使用複雜的演算法來動態建立 ServerSession。

MyServerSessionPool 會維護一個稱爲 inUse 的 Boolean 值陣列，來保留目前有哪些 ServerSession 在使用中的記錄。這些 Boolean 都會起始設定成 false。當呼叫 `getServerSession` 方法而要求儲存池中的 ServerSession 時，會搜尋 inUse 陣列找出第一個 false 值。找到之後，Boolean 會設爲 true，且會傳回對應的 ServerSession。如果 inUse 陣列中沒有 false，`getServerSession` 方法必須 `wait()`，直到獲得通知爲止。

通知會在下列情況下發生：

- 呼叫儲存池的 `close()` 方法，指出應該關閉儲存池。
- 目前使用中的 ServerSession 完成它的工作量並呼叫 `serverSessionFinished` 方法。`serverSessionFinished` 方法會將 ServerSession 送回儲存池中，並將對應的 inUse 旗號設爲 false。 之後，便可以重複使用 ServerSession。

### MessageListenerFactory.java

這個範例將每個 ServerSessionPool 案例關聯於一個訊息接聽器 Factory 物件。MessageListenerFactory 類別代表一個非常簡單的介面，用來取得實作 `javax.jms.MessageListener` 介面之類別的案例。這個類別只含有一個方法：

```
javax.jms.MessageListener createMessageListener();
```

當建構 ServerSessionPool 時，會提供這個介面的實作。這個物件用來建立訊息接聽器給在儲存池中備份 ServerSession 的個別 JMS Session。這個架構表示 `MessageListenerFactory` 介面的每個個別實作都必須有它自己的 ServerSessionPool。

MQ JMS 含有一個範例 MessageListenerFactory 實作，請參閱第216頁的『CountingMessageListenerFactory.java』，找到它的相關討論。

## ASF 使用範例

`<install_dir>/samples/jms/asf`（其中 <install_dir> 是 MQ JMS 的安裝目錄）目錄中有一組類別及其原始檔。這些類別會在第211頁的『應用程式伺服器範例程式碼』中所說明的範例獨立式應用程式伺服器環境中，使用第205頁的『ASF 類別和功能』中所說明的 MQ JMS 應用程式伺服器機能。

這些範例由從屬站應用程式的角度來提供三個 ASF 使用範例：

- 簡單點對點範例使用下列各項：
  - ASFClient1.java
  - Load1.java
  - CountingMessageListenerFactory.java

- 比較複雜的點對點範例使用下列各項：
  - ASFClient2.java
  - Load2.java
  - CountingMessageListenerFactory.java
  - LoggingMessageListenerFactory.java
- 簡單發佈/訂閱範例使用下列各項：
  - ASFClient3.java
  - TopicLoad.java
  - CountingMessageListenerFactory.java
- 比較複雜的發佈/訂閱範例使用下列各項：
  - ASFClient4.java
  - TopicLoad.java
  - CountingMessageListenerFactory.java
  - LoggingMessageListenerFactory.java

下列各節依次說明每個類別。

## Load1.java

這個類別是一個簡式通用 JMS 應用程式，它會載入含有若干訊息的給定佇列，之後，再予以終止。它可以從 JNDI 名稱空間中擷取必要的管理物件，也可以利用實作這些介面的 MQ JMS 類別來明確建立它們。必要的管理物件有 QueueConnectionFactory 和 Queue。您可以利用指令行選項來設定載入佇列時所含的訊息數，以及個別訊息放置作業之間的休眠時間。

這個應用程式有兩個版本的指令行語法。

如果要搭配 JNDI 來使用，語法如下：

```
java Load1 [-icf jndiICF] [-url jndiURL] [-qcfLookup qcfLookup]
           [-qLookup qLookup] [-sleep sleepTime] [-msgs numMsgs]
```

如果不要搭配 JNDI 來使用，語法如下：

```
java Load1 -nojndi [-qm qMgrName] [-q qName]
                   [-sleep sleepTime] [-msgs numMsgs]
```

表22 說明各參數及其預設值。

*表 22. Load1 參數和預設值*

| 參數 | 意義 | 預設值 |
|---|---|---|
| jndiICF | JNDI 所用的起始環境定義 Factory 類別 | com.sun.jndi.ldap.LdapCtxFactory |
| jndiURL | JNDI 所用的提供者 URL | ldap://localhost/o=ibm,c=us |
| qcfLookup | QueueConnectionFactory 所用的 JNDI 查閱鍵 | cn=qcf |
| qLookup | Queue 所用的 JNDI 查閱鍵 | cn=q |
| qMgrName | 要連接的佇列管理程式名稱 | ""（使用預設佇列管理程式） |
| qName | 要載入的佇列名稱 | SYSTEM.DEFAULT.LOCAL.QUEUE |

*表 22. Load1 參數和預設值 (繼續)*

| 參數 | 意義 | 預設值 |
|---|---|---|
| sleepTime | 訊息放置作業之間的暫停時間（以毫秒爲單位） | 0（不暫停） |
| numMsgs | 要放置的訊息數目 | 1000 |

如果有任何錯誤的話，會出現一則錯誤訊息，且應用程式會終止。

您可以利用這個應用程式來模擬 MQSeries 佇列中的訊息載入。之後，這個訊息載入又會觸發下列各節所說明具有 ASF 功能的應用程式。放入佇列中的訊息是簡單的 JMS TextMessage 物件。這些物件不含使用者定義的訊息內容，當使用不同的訊息接聽器時，這可能非常有用。提供原始程式可讓您在必要時修改這個載入應用程式。

## CountingMessageListenerFactory.java

這個檔案含有兩個類別的定義：

- CountingMessageListener
- CountingMessageListenerFactory

CountingMessageListener 是一個非常簡單的 `javax.jms.MessageListener` 介面實作。它保留了它的 `onMessage` 方法的呼叫次數記錄，但與傳送的訊息無關。

CountingMessageListenerFactory 是 CountingMessageListener 的 Factory 類別。它是 第214頁的『MessageListenerFactory.java』 中所說明之 `MessageListenerFactory` 介面的實作。這個 Factory 會保留它產生的所有訊息接聽器的記錄。它也含有一個 `printStats()` 方法，會顯示其中每個接聽器的使用統計。

## ASFClient1.java

這個應用程式要作爲 MQ JMS ASF 的從屬站。它會設定單一 ConnectionConsumer 來使用單一 MQSeries 佇列中的訊息。它會顯示所用的每個訊息接聽器的通訊量統計，且會在 1 分鐘之後終止作業。

這個應用程式可以從 JNDI 名稱空間中擷取必要的管理物件，也可以利用實作這些介面的 MQ JMS 類別來明確建立它們。必要的管理物件有 QueueConnectionFactory 和 Queue。

這個應用程式有兩個版本的指令行語法：

如果要搭配 JNDI 來使用，語法如下：

```
java ASFClient1 [-icf jndiICF] [-url jndiURL] [-qcfLookup qcfLookup]
                [-qLookup qLookup] [-poolSize poolSize] [-batchSize batchSize]
```

如果不要搭配 JNDI 來使用，語法如下：

```
java ASFClient1 -nojndi [-qm qMgrName] [-q qName]
                        [-poolSize poolSize] [-batchSize batchSize]
```

表23 說明各參數及其預設值。

*表 23. ASFClient1 參數和預設值*

| 參數 | 意義 | 預設值 |
|---|---|---|
| jndiICF | JNDI 所用的起始環境定義 Factory 類別 | com.sun.jndi.ldap.LdapCtxFactory |
| jndiURL | JNDI 所用的提供者 URL | ldap://localhost/o=ibm,c=us |
| qcfLookup | QueueConnectionFactory 所用的 JNDI 查閱鍵 | cn=qcf |
| qLookup | Queue 所用的 JNDI 查閱鍵 | cn=q |
| qMgrName | 要連接的佇列管理程式名稱 | ""（使用預設佇列管理程式） |
| qName | 使用的來源佇列名稱 | SYSTEM.DEFAULT.LOCAL.QUEUE |
| poolSize | 使用的 ServerSessionPool 中所建立的 ServerSession 數目 | 5 |
| batchSize | 可同時指派給 ServerSession 的訊息數目上限。 | 10 |

這個應用程式會從 QueueConnectionFactory 中取得 QueueConnection。

形式爲 MyServerSessionPool 的 ServerSessionPool 是採用下列各項來建構的：

- 先前所建立的 QueueConnection
- 必要的 poolSize
- 認可模式 AUTO_ACKNOWLEDGE
- CountingMessageListenerFactory 的案例，如第216頁的『CountingMessageListenerFactory.java』中所說明

之後，會呼叫連線的 createConnectionConsumer 方法，傳入：

- 先前所取得的 Queue
- 空的訊息選取元（表示應該接受所有訊息）
- 剛建立的 ServerSessionPool
- 必要的 batchSize

之後，再透過呼叫連線的 start() 方法來使用訊息。

這個從屬站應用程式會顯示所用的每個訊息接聽器的通訊量統計，且會每 10 秒鐘顯示一次統計。等 1 分鐘之後，連線會關閉，伺服器階段作業儲存池會停止，應用程式會終止。

## Load2.java

這個類別是一個 JMS 應用程式，它會載入含有若干訊息的給定佇列，之後，再採用類似於 Load1.java 的方法予以終止。指令行語法也類似於 Load1.java 的語法（用 Load2 來取代語法中的 Load1）。如果需要詳細資料，請參閱第215頁的『Load1.java』。

不同之處在於每個訊息都含有一個使用者內容呼叫值，採用隨機選取的 0-100 之間的整數值。這個內容表示您可以將選取選取元套用到訊息上。之後，訊息可以由『ASFClient2.java』中說明從屬站應用程式所建立的兩個消費者來共用。

## LoggingMessageListenerFactory.java

這個檔案含有兩個類別的定義：

- LoggingMessageListener
- LoggingMessageListenerFactory

LoggingMessageListener 是 javax.jms.MessageListener 介面的實作。它會取用傳給它的訊息，並在日誌檔中寫入一個項目。預設日誌檔是 ./ASFClient2.log。您可以檢查這個檔案，以及檢查傳送給使用這個訊息接聽器之連線消費者的訊息。

LoggingMessageListenerFactory 是 LoggingMessageListener 的 Factory 類別。它是第214頁的『MessageListenerFactory.java』中所說明之 MessageListenerFactory 介面的實作。

## ASFClient2.java

ASFClient2.java 是一個比 ASFClient1.java 複雜些的從屬站應用程式。它會建立兩個使用相同佇列，但套用不同訊息選取元的 ConnectionConsumer。這個應用程式將 CountingMessageListenerFactory 用於一個消費者，將 LoggingMessageListenerFactory 用於另一消費者。使用兩種不同的訊息接聽器 Factory 表示每個消費者都必須有它自己的伺服器階段作業儲存池。

這個應用程式會在一個畫面中顯示一個 ConnectionConsumer 的相關統計，而將另一 ConnectionConsumer 的相關統計寫入日誌檔中。

指令行語法類似於第216頁的『ASFClient1.java』的語法（用 ASFClient2 來取代語法中的 ASFClient1）。兩個伺服器階段作業儲存池都含有 poolSize 參數所設定的 ServerSession 數目。

訊息的分散應該不平均。 Load2 載入來源佇列的訊息含有一個使用者內容，其值應該隨機而平均分散在 0 和 100 之間。訊息選取元 value>75 套用到 highConnectionConsumer，訊息選取元 value≤75 套用到 normalConnectionConsumer。highConnectionConsumer 的訊息（大約總載入量的 25%）會傳送到 LoggingMessageListener。 normalConnectionConsumer 的訊息（大約總載入量的 75%）會傳送到 CountingMessageListener。

當從屬站應用程式執行時，normalConnectionConsumer 的相關統計及其相關 CountingMessageListenerFactory 會每 10 秒列印到畫面一次。 highConnectionConsumer 的相關統計及其相關 LoggingMessageListenerFactory 會寫入日誌檔中。

您可以檢視畫面和日誌檔來查看訊息的眞正目的地。請加入每個 CountingMessageListener 的總數。只有在使用所有訊息之前，從屬站應用程式沒有終止，這應該含有載入量的 75%。日誌檔項目數應該含有載入量的其餘部份。（如果從屬站應用程式在使用所有訊息之前便告終止，您可以增加應用程式的逾時值。）

## TopicLoad.java

這個類別是一個 JMS 應用程式，它是 第218頁的『Load2.java』 所說明的 Load2 佇列載入器的發佈/訂閱版。它會在給定主題下，發佈必要的訊息數目，然後終止。每個訊息都含有一個使用者內容呼叫值，採用隨機選取的 0-100 之間的整數值。

如果要使用這個應用程式，請確定分配管理程式在執行中，且必要的設定已經完成。如果需要詳細資料，請參閱第20頁的『發佈/訂閱模式的其它設定』。

這個應用程式有兩個版本的指令行語法。

如果要搭配 JNDI 來使用，語法如下：

```
java TopicLoad [-icf jndiICF] [-url jndiURL] [-tcfLookup tcfLookup]
               [-tLookup tLookup] [-sleep sleepTime] [-msgs numMsgs]
```

如果不要搭配 JNDI 來使用，語法如下：

```
java TopicLoad -nojndi [-qm qMgrName] [-t tName]
                       [-sleep sleepTime] [-msgs numMsgs]
```

表24 說明各參數及其預設值。

*表 24. TopicLoad 參數和預設值*

| 參數 | 意義 | 預設值 |
|---|---|---|
| jndiICF | JNDI 所用的起始環境定義 Factory 類別 | com.sun.jndi.ldap.LdapCtxFactory |
| jndiURL | JNDI 所用的提供者 URL | ldap://localhost/o=ibm,c=us |
| tcfLookup | TopicConnectionFactory 所用的 JNDI 查閱鍵 | cn=tcf |
| tLookup | Topic 所用的 JNDI 查閱鍵 | cn=t |
| qMgrName | 要連接的佇列管理程式名稱，及發佈訊息所要送往的分配管理程式佇列管理程式 | ""（使用預設佇列管理程式） |
| tName | 要發佈的主題名稱 | MQJMS/ASF/TopicLoad |

*表 24. TopicLoad 參數和預設值 (繼續)*

| 參數 | 意義 | 預設值 |
|---|---|---|
| sleepTime | 訊息放置作業之間的暫停時間（以毫秒爲單位） | 0（不暫停） |
| numMsgs | 要放置的訊息數目 | 200 |

如果有任何錯誤的話，會出現一則錯誤訊息，且應用程式會終止。

## ASFClient3.java

ASFClient3.java 是一個從屬站應用程式，它是 第216頁的『ASFClient1.java』 的發佈/訂閱版。它會設定單一 ConnectionConsumer 來使用所發佈的單一 Topic 訊息。它會顯示所用的每個訊息接聽器的通訊量統計，且會在 1 分鐘之後終止作業。

這個應用程式有兩個版本的指令行語法。

如果要搭配 JNDI 來使用，語法如下：

```
java ASFClient3 [-icf jndiICF] [-url jndiURL] [-tcfLookup tcfLookup]
                [-tLookup tLookup] [-poolsize poolSize] [-batchsize batchSize]
```

如果不要搭配 JNDI 來使用，語法如下：

```
java ASFClient3 -nojndi [-qm qMgrName] [-t tName]
                        [-poolsize poolSize] [-batchsize batchSize]
```

表25 說明各參數及其預設值。

*表 25. ASFClient3 參數和預設值*

| 參數 | 意義 | 預設值 |
|---|---|---|
| jndiICF | JNDI 所用的起始環境定義 Factory 類別 | com.sun.jndi.ldap.LdapCtxFactory |
| jndiURL | JNDI 所用的提供者 URL | ldap://localhost/o=ibm,c=us |
| tcfLookup | TopicConnectionFactory 所用的 JNDI 查閱鍵 | cn=tcf |
| tLookup | Topic 所用的 JNDI 查閱鍵 | cn=t |
| qMgrName | 要連接的佇列管理程式名稱，及發佈訊息所要送往的分配管理程式佇列管理程式 | ""（使用預設佇列管理程式） |
| tName | 使用的來源主題名稱 | MQJMS/ASF/TopicLoad |
| poolSize | 使用的 ServerSessionPool 中所建立的 ServerSession 數目 | 5 |
| batchSize | 可同時指派給 ServerSession 的訊息數目上限。 | 10 |

和 ASFClient1 一樣，這個從屬站應用程式會顯示所用的每個訊息接聽器的通訊量統計，且會每 10 秒鐘顯示一次統計。等 1 分鐘之後，連線會關閉，伺服器階段作業儲存池會停止，應用程式會終止。

## ASFClient4.java

ASFClient4.java 是一個比較複雜的發佈/訂閱從屬站應用程式。它會建立三個 ConnectionConsumer，它們都需要相同的主題，但分別套用不同的訊息選取元。

前兩個消費者使用 'high' 和 'normal' 的訊息選取元，使用方式如同第218頁的『ASFClient2.java』應用程式。第三個消費者不使用任何訊息選取元。這個應用程式在兩個選取元型的消費者上使用兩個 CountingMessageListenerFactory，在第三個消費者上使用 LoggingMessageListenerFactory。由於應用程式使用不同的訊息接聽器 Factory，因而每個消費者都必須有它自己的伺服器階段作業儲存池。

應用程式會在畫面中顯示兩個選取元型消費者的相關統計。它會將第三個 ConnectionConsumer 的相關統計寫入日誌檔中。

指令行語法類似於 第220頁的『ASFClient3.java』 的語法（用 ASFClient4 來取代語法中的 ASFClient3）。三個伺服器階段作業儲存池都含有 poolSize 參數所設定的 ServerSession 數目。

當從屬站應用程式執行時，normalConnectionConsumer 和 highConnectionConsumer 的相關統計及其相關 CountingMessageListenerFactory 會每 10 秒列印到畫面一次。第三個 ConnectionConsumer 的相關統計及其相關 LoggingMessageListenerFactory 會寫入日誌檔中。

您可以檢視畫面和日誌檔來查看訊息的眞正目的地。請加入每個 CountingMessageListener 的總數，並檢查日誌檔項目數。

訊息的分送應該不同於得自相同應用程式 (ASFClient2.java) 之點對點版本的分送。這是因爲在發佈/訂閱領域中，主題的每一則發佈訊息，主題的每個消費者都會取得一份自己的副本。在這個應用程式中，就給定主題的量而言，'high' 和 'normal' 消費者分別會收到大約 25% 和 75% 的量。第三個消費者仍會收到 100% 的數量。因此，收到的訊息總數會大於主題原始發佈量的 100%。

# 第14章 JMS 介面和類別

MQSeries Class for Java 訊息服務由若干 Java 類別和介面組成，這些類別和介面以 Sun java.jms 套件中的介面和類別爲基礎。從屬站應該利用下列中的 Sun 介面和類別來撰寫，以下各節會有這些介面和類別的詳細說明。實作 Sun 介面和類別之 MQSeries 物件的名稱字首爲 'MQ'（物件說明中另有指示者除外）。這些說明包括 MQSeries 物件任何偏離標準 JMS 定義之處的詳細資料。這些偏離之處會標出 '*'。

## Sun Java 訊息服務類別和介面

下表列出 **javax.jms** 套件所包含的 JMS 物件。 有 '*' 標示的介面由應用程式來實作。有 '**' 標示的介面由應用程式伺服器來實作。

*表 26. 介面摘要*

| 介面 | 說明 |
|---|---|
| **BytesMessage** | BytesMessage 用來傳送含有未解譯之位元組串流的訊息。 |
| **Connection** | JMS Connection 是從屬站通往其 JMS 提供者的作用連線。 |
| **ConnectionConsumer** | Connection 爲應用程式伺服器提供一個用來建立 ConnectionConsumer 的特殊機能。 |
| **ConnectionFactory** | ConnectionFactory 封裝一組管理者已定義的連線配置參數。 |
| **ConnectionMetaData** | ConnectionMetaData 提供說明 Connection 的資訊。 |
| **DeliveryMode** | JMS 所支援的遞送模式。 |
| **Destination** | Queue 和 Topic 的母項介面。 |
| **ExceptionListener*** | 用來接收 Connection 非同步遞送執行緒所擲出之異常狀況的異常狀況接聽器。 |
| **MapMessage** | MapMessage 用來傳送一組名稱/值配對，名稱是 Strings，值是 Java 基本類型。 |
| **Message** | Message 介面是所有 JMS 訊息的根介面。 |
| **MessageConsumer** | 所有訊息消費者的母項介面。 |
| **MessageListener*** | MessageListener 用來接收非同步遞送的訊息。 |
| **MessageProducer** | 從屬站利用 MessageProducer 來傳送訊息到 Destination。 |
| **ObjectMessage** | ObjectMessage 用來傳送含有可序列化之 Java 物件的訊息。 |
| **Queue** | Queue 物件會封裝特定提供者專用的佇列名稱。 |
| **QueueBrowser** | 從屬站利用 QueueBrowser 來查看佇列中的訊息，但不移除它們。 |
| **QueueConnection** | QueueConnection 是通往 JMS 點對點提供者的作用中連線。 |
| **QueueConnectionFactory** | 從屬站利用 QueueConnectionFactory，以 JMS 點對點提供者來建立 QueueConnections。 |
| **QueueReceiver** | 從屬站利用 QueueReceiver 來接收已遞送到佇列的訊息。 |
| **QueueSender** | 從屬站利用 QueueSender 來傳送訊息到佇列。 |
| **QueueSession** | QueueSession 提供建立 QueueReceivers、QueueSenders、QueueBrowsers 和 TemporaryQueues 的方法。 |

*表 26. 介面摘要 (繼續)*

| 介面 | 說明 |
|---|---|
| **ServerSession **** | ServerSession 不是應用程式伺服器所實作的物件。 |
| **ServerSessionPool **** | ServerSessionPool 是應用程式伺服器所實作，用來提供 ServerSession 儲存池以處理 ConnectionConsumer 之訊息的物件。 |
| **Session** | JMS Session 是一個產生和消費訊息的單執行緒環境。 |
| **StreamMessage** | StreamMessage 用來傳送 Java 基本類型的串流。 |
| **TemporaryQueue** | TemporaryQueue 是專爲 QueueConnection 的持續期間而建立的唯一 Queue 物件。 |
| **TemporaryTopic** | TemporaryTopic 是專爲 TopicConnection 的持續期間而建立的唯一 Topic 物件。 |
| **TextMessage** | TextMessage 用來傳送含有 `java.lang.String` 的訊息。 |
| **Topic** | Topic 物件封裝特定提供者專用的主題名稱。 |
| **TopicConnection** | TopicConnection 是通往 JMS 發佈/訂閱提供者的作用中連線。 |
| **TopicConnectionFactory** | 從屬站利用 TopicConnectionFactory，以 JMS 發佈/訂閱提供者來建立 TopicConnection。 |
| **TopicPublisher** | 從屬站利用 TopicPublisher 來發佈主題訊息。 |
| **TopicSession** | TopicSession 提供建立 TopicPublishers、TopicSubscribers 和 TemporaryTopics 的方法。 |
| **TopicSubscriber** | 從屬站利用 TopicSubscriber 來接收已就某主題而發佈的訊息。 |
| **XAConnection** | XAConnection 提供 XASession 來延伸連線功能。 |
| **XAConnectionFactory** | 有些應用程式伺服器支援將具有 Java 異動服務 (JTS) 能力的資源使用分組到分散式異動中。 |
| **XAQueueConnection** | XAQueueConnection 提供和 QueueConnection 相同的建立選項。 |
| **XAQueueConnectionFactory** | XAQueueConnectionFactory 提供和 QueueConnectionFactory 相同的建立選項。 |
| **XAQueueSession** | XAQueueSession 提供一個可用來建立 QueueReceivers、QueueSenders 和 QueueBrowsers 的正規 QueueSession。 |
| **XASession** | XASession 新增 JMS 提供者的 Java 異動 API (JTA) 支援存取來延伸 Session 的功能。 |
| **XATopicConnection** | XATopicConnection 提供和 TopicConnection 相同的建立選項。 |
| **XATopicConnectionFactory** | XATopicConnectionFactory 提供和 TopicConnectionFactory 相同的建立選項。 |
| **XATopicSession** | XATopicSession 提供可用來建立 TopicSubscribers 和 TopicPublishers 的正規 TopicSession。 |

*表 27. 類別摘要*

| 類別 | 說明 |
| --- | --- |
| **QueueRequestor** | JMS 提供一個 QueueRequestor Help 類別來簡化發出服務要求的程序。 |
| **TopicRequestor** | JMS 提供一個 TopicRequestor Help 類別來簡化發出服務要求的程序。 |

## MQSeries JMS 類別

下表列出含有實作 Sun 介面之 MQSeries 類別的 **com.ibm.mq.jms** 和 **com.ibm.jms** 套件。

*表 28. 'com.ibm.mq.jms' 套件類別摘要*

| 類別 | 實作 |
|---|---|
| MQConnection | Connection |
| MQConnectionConsumer | ConnectionConsumer |
| MQConnectionFactory | ConnectionFactory |
| MQConnectionMetaData | ConnectionMetaData |
| MQDestination | Destination |
| MQMessageConsumer | MessageConsumer |
| MQMessageProducer | MessageProducer |
| MQQueue | Queue |
| MQQueueBrowser | QueueBrowser |
| MQQueueConnection | QueueConnection |
| MQQueueConnectionFactory | QueueConnectionFactory |
| MQQueueEnumeration | QueueBrowser 中的 java.util.Enumeration |
| MQQueueReceiver | QueueReceiver |
| MQQueueSender | QueueSender |
| MQQueueSession | QueueSession |
| MQSession | Session |
| MQTemporaryQueue | TemporaryQueue |
| MQTemporaryTopic | TemporaryTopic |
| MQTopic | Topic |
| MQTopicConnection | TopicConnection |
| MQTopicConnectionFactory | TopicConnectionFactory |
| MQTopicPublisher | TopicPublisher |
| MQTopicSession | TopicSession |
| MQTopicSubscriber | TopicSubscriber |
| MQXAConnection | XAConnection |
| MQXAConnectionFactory | XAConnectionFactory |
| MQXAQueueConnection | XAQueueConnection |
| MQXAQueueConnectionFactory | XAQueueConnectionFactory |
| MQXAQueueSession | XAQueueSession |
| MQXASession | XASession |
| MQXATopicConnection | XATopicConnection |
| MQXATopicConnectionFactory | XATopicConnectionFactory |
| MQXATopicSession | XATopicSession |

*表 29. 套件 'com.ibm.jms' 類別摘要*

| 類別 | 實作 |
|---|---|
| JMSBytesMessage | BytesMessage |
| JMSMapMessage | MapMessage |
| JMSMessage | Message |
| JMSObjectMessage | ObjectMessage |
| JMSStreamMessage | StreamMessage |
| JMSTextMessage | TextMessage |

這個版次的 MQSeries Class for Java 訊息服務有下列 JMS 介面的範例實作。

- ServerSession
- ServerSessionPool

請參閱第211頁的『應用程式伺服器範例程式碼』。

# BytesMessage

public interface **BytesMessage**
extends **Message**

MQSeries 類別：**JMSBytesMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSBytesMessage
```

BytesMessage 用來傳送含有未解譯之位元組串流的訊息。它繼承 **Message**，並加入了一個位元組訊息主體。訊息的接收端負責解譯位元組。

**註:** 這個訊息類型專供從屬站的現存訊息格式編碼之用。如果可能的話，應該改用其它自行定義的訊息類型。

另請參閱：**MapMessage**、**Message**、**ObjectMessage**、**StreamMessage** 和 **TextMessage**。

## 方法

### readBoolean

```
public boolean readBoolean() throws JMSException
```

從位元組訊息中讀取 Boolean。

**傳回：** 讀取的 Boolean 值。

**擲出：**

- MessageNotReadableException - 如果訊息採用唯寫模式的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。
- MessageEOFException - 如果它是訊息位元組結尾的話。

### readByte

```
public byte readByte() throws JMSException
```

從位元組訊息中讀取帶正負號的 8 位元值。

**傳回：** 位元組訊息中下個位元組，作爲帶正負號的 8 位元 byte。

**擲出：**

- MessageNotReadableException - 如果訊息採用唯寫模式的話。
- MessageEOFException - 如果它是訊息位元組結尾的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。

### readUnsignedByte

```
public int readUnsignedByte() throws JMSException
```

從位元組訊息中讀取無正負號的 8 位元數字。

**傳回：** 位元組訊息中下個位元組，解譯爲無正負號的 8 位元數字。

**擲出：**

- MessageNotReadableException - 如果訊息採用唯寫模式的話。
- MessageEOFException - 如果它是訊息位元組結尾的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。

### readShort

```
public short readShort() throws JMSException
```

從位元組訊息中讀取帶正負號的 16 位元數字。

**傳回：** 位元組訊息中下兩個位元組，解譯爲帶正負號的 16 位元數字。

**擲出：**

- MessageNotReadableException - 如果訊息採用唯寫模式的話。
- MessageEOFException - 如果它是訊息位元組結尾的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。

### readUnsignedShort

```
public int readUnsignedShort() throws JMSException
```

從位元組訊息中讀取無正負號的 16 位元數字。

**傳回：** 位元組訊息中下兩個位元組，解譯爲無正負號的 16 位元整數。

**擲出：**

- MessageNotReadableException - 如果訊息採用唯寫模式的話。
- MessageEOFException - 如果它是訊息位元組結尾的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。

### readChar

```
public char readChar() throws JMSException
```

從位元組訊息中讀取 Unicode 字元值。

**傳回：** 位元組訊息中下個位元組，作爲一個 Unicode 字元。

**擲出：**

- MessageNotReadableException - 如果訊息採用唯寫模式的話。
- MessageEOFException - 如果它是訊息位元組結尾的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。

### readInt

```
public int readInt() throws JMSException
```

從位元組訊息中讀取帶正負號的 32 位元整數。

**傳回：** 位元組訊息中下四個位元組，解譯爲一個 `int`。

**擲出：**

- MessageNotReadableException - 如果訊息採用唯寫模式的話。
- MessageEOFException - 如果它是訊息位元組結尾的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。

### readLong

```
public long readLong() throws JMSException
```

從位元組訊息中讀取帶正負號的 64 位元整數。

**傳回：** 位元組訊息中下八個位元組，解譯爲一個 `long`。

**擲出：**

- MessageNotReadableException - 如果訊息採用唯寫模式的話。
- MessageEOFException - 如果它是訊息位元組結尾的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。

### readFloat

```
public float readFloat() throws JMSException
```

從位元組訊息中讀取一個 `float`。

**傳回：** 位元組訊息中下四個位元組，解譯爲一個 `float`。

**擲出：**

- MessageNotReadableException - 如果訊息採用唯寫模式的話。
- MessageEOFException - 如果它是訊息位元組結尾的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。

### readDouble

```
public double readDouble() throws JMSException
```

從位元組訊息中讀取一個 double。

**傳回：** 位元組訊息中下八個位元組，解譯爲一個 double。

**擲出：**

- MessageNotReadableException - 如果訊息採用唯寫模式的話。
- MessageEOFException - 如果它是訊息位元組結尾的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。

### readUTF

```
public java.lang.String readUTF() throws JMSException
```

從位元組訊息中讀取一個已利用修改過的 UTF-8 格式編碼過的字串。前兩個位元組會解譯成 2 位元組長度欄位。

**傳回：** 位元組訊息中的一個 Unicode 字串。

**擲出：**

- MessageNotReadableException - 如果訊息採用唯寫模式的話。
- MessageEOFException - 如果它是訊息位元組結尾的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。

### readBytes

```
public int readBytes(byte[] value) throws JMSException
```

從位元組訊息中讀取一個位元組陣列。如果串流中仍有足夠的位元組，便會填滿整個緩衝區，否則會局部填入緩衝區。

**參數：** value - 資料讀入其中的緩衝區。

**傳回：** 讀入緩衝區的位元數，如果已到達位元組尾端而不再有資料時，則爲 -1。

**擲出：**

- MessageNotReadableException - 如果訊息採用唯寫模式的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。

### readBytes

```
public int readBytes(byte[] value, int length)
              throws JMSException
```

讀取位元組訊息的一部份。

**參數：**

- value - 資料讀入其中的緩衝區。
- length - 要讀取的位元組數目。

**傳回：** 讀入緩衝區的位元數，如果已到達位元組尾端而不再有資料時，則爲 -1。

**擲出：**

- MessageNotReadableException - 如果訊息採用唯寫模式的話。

- IndexOutOfBoundsException - 如果 length 是負的或小於陣列 value 的長度的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。

### writeBoolean

```
public void writeBoolean(boolean value) throws JMSException
```

將一個 Boolean 當作 1 位元組值來寫入位元組訊息中。 true 值會寫成 (byte)1 值； false 值會寫成 (byte)0 值。

**參數：** value - 要寫入的 Boolean 值。

**擲出：**

- MessageNotWriteableException - 如果訊息採用唯讀模式的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。

### writeByte

```
public void writeByte(byte value) throws JMSException
```

將一個 Byte 當作 1 位元組值來寫入位元組訊息中。

**參數：** value - 要寫入的 byte 值。

**擲出：**

- MessageNotWriteableException - 如果訊息採用唯讀模式的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。

### writeShort

```
public void writeShort(short value) throws JMSException
```

將一個 Short 當作兩個位元組寫入位元組訊息中。

**參數：** value - 要寫入的 Short。

**擲出：**

- MessageNotWriteableException - 如果訊息採用唯讀模式的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。

### writeChar

```
public void writeChar(char value) throws JMSException
```

將一個 char 當作 2 位元組值來寫入位元組訊息中，高址位元組先處理。

**參數：** value - 要寫入的 char 值。

**擲出：**

- MessageNotWriteableException - 如果訊息採用唯讀模式的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。

### writeInt

```
public void writeInt(int value) throws JMSException
```

將一個 int 當作四位元組寫入位元組訊息中。

**參數：** value - 要寫入的 int。

**擲出：**

- MessageNotWriteableException - 如果訊息採用唯讀模式的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。

### writeLong

```
public void writeLong(long value) throws JMSException
```

將一個 long 當作八位元組寫入位元組訊息中。

**參數：** value - 要寫入的 long。

**擲出：**

- MessageNotWriteableException - 如果訊息採用唯讀模式的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。

### writeFloat

```
public void writeFloat(float value) throws JMSException
```

利用 Float 類別中的 floatToIntBits 方法，將浮點引數轉換成 int，再將這個 int 當作四位元組數量寫入位元組訊息中。

**參數：** value - 要寫入的 float 值。

**擲出：**

- MessageNotWriteableException - 如果訊息採用唯讀模式的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。

## BytesMessage

### writeDouble

```
public void writeDouble(double value) throws JMSException
```

利用 Double 類別中的 doubleToLongBits 方法，將 double 引數轉換成 long，再將這個 long 當作 8 位元組數量寫入位元組訊息中。

**參數：** value - 要寫入的 double 值。

**擲出：**

- MessageNotWriteableException - 如果訊息採用唯讀模式的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。

### writeUTF

```
public void writeUTF(java.lang.String value)
                              throws JMSException
```

採與機器無關的方式，利用 UTF-8 編碼將字串寫入位元組訊息中。寫入緩衝區的 UTF-8 字串開頭爲 2 位元組長度的欄位。

**參數：** value - 要寫入的 String 值。

**擲出：**

- MessageNotWriteableException - 如果訊息採用唯讀模式的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。

### writeBytes

```
public void writeBytes(byte[] value) throws JMSException
```

將位元組陣列寫入位元組訊息。

**參數：** value - 要寫入的位元組陣列。

**擲出：**

- MessageNotWriteableException - 如果訊息採用唯讀模式的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。

### writeBytes

```
public void writeBytes(byte[] value,
                       int length) throws JMSException
```

將位元組陣列的一部份寫入位元組訊息中。

**參數：**

- value - 要寫入的位元組陣列值。
- offset - 位元組陣列內的起始偏移。
- length - 要使用的位元組數目。

**擲出：**

- MessageNotWriteableException - 如果訊息採用唯讀模式的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。

### writeObject

```
public void writeObject(java.lang.Object value)
                                    throws JMSException
```

將 Java 物件寫入位元組訊息中。

**註:** 這個方法只適用於基本物件類型（如 Integer、Double 和 Long）、String 及位元組陣列。

**參數：** `value` - 要寫入的 Java 物件。

**擲出：**

- MessageNotWriteableException - 如果訊息採用唯讀模式的話。
- MessageFormatException - 如果物件類型無效的話。
- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。

### reset

```
public void reset() throws JMSException
```

將訊息主體置於唯讀模式，並將位元組的位元組重新定位到起始位置。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法重設訊息的話。
- MessageFormatException - 如果訊息格式無效的話。

# Connection

public interface **Connection**

子介面：**QueueConnection**、**TopicConnection**、**XAQueueConnection** 和 **XATopicConnection**

MQSeries 類別：**MQConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection
```

JMS Connection 是從屬站通往其 JMS 提供者的作用連線。

另請參閱：**QueueConnection**、**TopicConnection**、**XAQueueConnection** 和 **XATopicConnection**

## 方法

### getClientID

```
public java.lang.String getClientID()
                                throws JMSException
```

取得這個連線的從屬站識別碼。從屬站識別碼可以由管理者預先配置在 ConnectionFactory 中，也可以呼叫 setClientId 來指派。

**傳回：** 唯一從屬站識別碼。

**擲出：** JMSException - 如果 JMS 實作因內部錯誤而無法傳回這個 Connection 的從屬站 ID。

### setClientID

```
public void setClientID(java.lang.String clientID)
                                            throws JMSException
```

設定這個連線的從屬站識別碼。

**註:** 這會忽略點對點連線的從屬站識別碼。

**參數：** clientID - 唯一從屬站識別碼。

**擲出：**

- JMSException - 如果 JMS 實作因內部錯誤而無法設定這個 Connection 的從屬站 ID。
- InvalidClientIDException - 如果 JMS 從屬站指定了無效或重複的從屬站 ID。
- IllegalStateException - 如果試圖在錯誤時間設定連線的從屬站識別碼，或先前已爲它進行了管理配置。

### getMetaData

```
public ConnectionMetaData getMetaData() throws JMSException
```

取得這個連線的 meta 資料。

**傳回：** 連線 meta 資料。

**擲出：** JMSException - 如果 JMS 實作無法取得這個 Connection 的 Connection meta 資料的話，爲一般異常狀況。

**另請參閱：**

**ConnectionMetaData**

### getExceptionListener

```
public ExceptionListener getExceptionListener()
            throws JMSException
```

取得這個 Connection 的 ExceptionListener。

**傳回：** 這個 Connection 的 ExceptionListener

**擲出：** JMSException - 如果 JMS 實作無法取得這個 Connection 的 Exception 接聽器的話，爲一般異常狀況。

### setExceptionListener

```
public void setExceptionListener(ExceptionListener listener)
            throws JMSException
```

設定這個連線的異常狀況接聽器。

**參數：** handler - 異常狀況接聽器。

**擲出：** JMSException - 如果 JMS 實作無法設定這個 Connection 的 Exception 接聽器的話，爲一般異常狀況。

### start

```
public void start() throws JMSException
```

啓動（或重新啓動）Connectoin 對內收訊息的遞送。啓動已啓動的階段作業會被忽略，不做處理。

**擲出：** JMSException - 如果 JMS 實作因內部錯誤而無法啓動訊息遞送的話。

**另請參閱：**

停止

### stop

```
public void stop() throws JMSException
```

用來暫時停止 Connection 對內收訊息的遞送。利用它的 `start` 方法可以重新啓動它。當停止時，會禁止遞送給 Connection 的所有訊息消費者。同步接收會暫停執行，訊息也不會遞送給訊息接聽器。

停止階段作業不會影響到它傳送訊息的能力。停止已停止的階段作業會被忽略，不做處理。

**擲出：** JMSException - 如果 JMS 實作因內部錯誤而無法停止訊息遞送的話。

**另請參閱：**

start

### close

```
public void close() throws JMSException
```

## Connection

由於提供者可在 JVM 之外代表 Connection 來配置某些資源，因此，當不需要它們時，從屬站應該關閉它們。您終究不能依賴記憶體回收來收回這些資源，因爲記憶體回收的進行可能不夠快。已關閉的連線，其階段作業、產生者和消費者不需要關閉。

關閉連線會回復其階段作業之任何進行中的異動。在階段作業的工作由外部異動管理程式來協調的情況下，當使用 XASession 時，不會使用階段作業的確定和回復方法，稍後再由異動管理程式來判斷已關閉的階段作業之工作結果。關閉連線不會強制認可從屬站所認可的階段作業。

MQ JMS 會保留一個 MQSeries hConns 儲存池，供 Sessions 使用。在某些情況下，Connection.close() 會清除這個儲存池。如果應用程式循序使用多個 Connection，可能會強制儲存池在兩個 JMS Connection 之間維持在作用中。如果要做到這一點，請在 JMS 應用程式的存活時間中，在 com.ibm.mq.MQEnvironment 中登錄 MQPoolToken。如果需要詳細資料，請參閱第62頁的『連線儲存池』和第86頁的『MQEnvironment』。

**擲出：** JMSException - 如果 JMS 實作因內部錯誤而無法關閉連線的話。無法釋出資源或無法關閉 Socket 連線都是範例。

# ConnectionConsumer

public interface **ConnectionConsumer**

MQSeries 類別：**MQConnectionConsumer**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionConsumer
```

Connection 爲應用程式伺服器提供一個用來建立 ConnectionConsumer 的特殊機能。Destination 和 PropertySelector 會指定它要使用的訊息。另外，ConnectionConsumer 也必須得到用來處理它的訊息的 ServerSessionPool。

另請參閱：**QueueConnection** 和 **TopicConnection**。

## 方法

### close()

```
public void close() throws JMSException
```

由於提供者可在 JVM 之外代表 ConnectionConsumer 來配置某些資源，因此，當不需要它們時，從屬站應該關閉它們。您終究不能依賴記憶體回收來收回這些資源，因爲記憶體回收的進行可能不夠快。

**擲出：** JMSException - 如果 JMS 實作無法代表 ConnectionConsumer 釋出資源或無法關閉連線消費者的話。

### getServerSessionPool()

```
public ServerSessionPool getServerSessionPool()
                                        throws JMSException
```

取得這個連線消費者所關聯的伺服器階段作業。

**傳回：** 這個連線消費者所使用的伺服器階段作業儲存池。

**擲出：** JMSException - 如果 JMS 實作因內部錯誤而無法取得這個連線消費者所關聯的伺服器階段作業儲存池。

# ConnectionFactory

public interface **ConnectionFactory**
子介面：**QueueConnectionFactory**、**TopicConnectionFactory**、
**XAQueueConnectionFactory** 和 **XATopicConnectionFactory**

MQSeries 類別：**MQConnectionFactory**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory
```

ConnectionFactory 封裝一組管理者已定義好的連線配置參數。從屬站利用它來透過 JMS 提供者建立 Connection。

另請參閱：**QueueConnectionFactory**、**TopicConnectionFactory**、
**XAQueueConnectionFactory** 和 **XATopicConnectionFactory**

## MQSeries 建構子

### MQConnectionFactory

```
public MQConnectionFactory()
```

## 方法

### setDescription *

```
public void setDescription(String x)
```

物件的摘要說明。

### getDescription *

```
public String getDescription()
```

擷取物件說明。

### setTransportType *

```
public void setTransportType(int x) throws JMSException
```

設定要使用的傳輸類型。它可以是 JMSC.MQJMS_TP_BINDINGS_MQ 或 JMSC.MQJMS_TP_CLIENT_MQ_TCPIP。

### getTransportType *

```
public int getTransportType()
```

擷取傳輸類型。

### setClientId *

```
public void setClientId(String x)
```

設定利用這個 Connection 來建立的所有連線所用的從屬站識別碼。

**getClientId ***

```
public String getClientId()
```

取得利用這個 ConnectionFactory 建立的所有連線所用的從屬站識別碼。

**setQueueManager ***

```
public void setQueueManager(String x) throws JMSException
```

設定要連接之佇列管理程式的名稱。

**getQueueManager ***

```
public String getQueueManager()
```

取得佇列管理程式的名稱。

**setHostName ***

```
public void setHostName(String hostname)
```

限從屬站使用，要連接的主電腦名稱。

**getHostName ***

```
public String getHostName()
```

擷取主電腦名稱。

**setPort ***

```
public void setPort(int port) throws JMSException
```

設定從屬站連線的埠。

**參數：** port - 要使用的新值。

**擲出：** JMSException，如果埠是負的。

**getPort ***

```
public int getPort()
```

限從屬站連線，取得埠號。

**setChannel ***

```
public void setChannel(String x) throws JMSException
```

限從屬站使用，設定要使用的通道。

**getChannel ***

```
public String getChannel()
```

限從屬站使用，取得所用的通道。

**setCCSID ***

```
public void setCCSID(int x) throws JMSException
```

設定連接佇列管理程式時所要使用的字集。請參閱第103頁的表13，以取得可用值的清單。建議您在大部份情況下都使用預設值 (819)。

## ConnectionFactory

### getCCSID *

```
public int getCCSID()
```

取得佇列管理程式的字集。

### setReceiveExit *

```
public void setReceiveExit(String receiveExit)
```

實作接收跳出程式的類別名稱。

### getReceiveExit *

```
public String getReceiveExit()
```

取得接收跳出程式類別的名稱。

### setReceiveExitInit *

```
public void setReceiveExitInit(String x)
```

傳遞給接收跳出程式類別的建構子之起始設定字串。

### getReceiveExitInit *

```
public String getReceiveExitInit()
```

取得傳遞給接收跳出程式類別的起始設定字串。

### setSecurityExit *

```
public void setSecurityExit(String securityExit)
```

實作安全跳出程式的類別名稱。

### getSecurityExit *

```
public String getSecurityExit()
```

取得安全跳出程式類別的名稱。

### setSecurityExitInit *

```
public void setSecurityExitInit(String x)
```

傳遞給安全跳出程式建構子的起始設定字串。

### getSecurityExitInit *

```
public String getSecurityExitInit()
```

取得安全跳出程式起始設定字串。

### setSendExit *

```
public void setSendExit(String sendExit)
```

實作傳送跳出程式之類別名稱。

### getSendExit *

```
public String getSendExit()
```

取得傳送跳出程式類別的名稱。

### setSendExitInit *

```
public void setSendExitInit(String x)
```

傳遞給傳送跳出程式建構子的起始設定字串。

### getSendExitInit *

```
public String getSendExitInit()
```

取得傳送跳出程式起始設定字串。

# ConnectionMetaData

public interface **ConnectionMetaData**

MQSeries 類別：**MQConnectionMetaData**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionMetaData
```

ConnectionMetaData 提供說明連線的資訊。

## MQSeries 建構子

### MQConnectionMetaData

```
public MQConnectionMetaData()
```

## 方法

### getJMSVersion

```
public java.lang.String getJMSVersion() throws JMSException
```

取得 JMS 版本。

**傳回：** JMS 版本。

**擲出：** JMSException - 如果 meta 資料擷取期間，JMS 實作中發生內部錯誤的話。

### getJMSMajorVersion

```
public int getJMSMajorVersion() throws JMSException
```

取得 JMS 主要版本號碼。

**傳回：** JMS 主要版本號碼。

**擲出：** JMSException - 如果 meta 資料擷取期間，JMS 實作中發生內部錯誤的話。

### getJMSMinorVersion

```
public int getJMSMinorVersion() throws JMSException
```

取得 JMS 次要版本號碼。

**傳回：** JMS 次要版本號碼。

**擲出：** JMSException - 如果 meta 資料擷取期間，JMS 實作中發生內部錯誤的話。

### getJMSXPropertyNames

```
public java.util.Enumeration getJMSXPropertyNames()
                throws JMSException
```

取得這個連線所支援的 JMSX 內容名稱列舉。

**傳回：** JMSX 內容名稱列舉。

**擲出：** JMSException - 如果內容名稱擷取期間，JMS 實作中發生內部錯誤的話。

### getJMSProviderName

```
public java.lang.String getJMSProviderName()
              throws JMSException
```

取得 JMS 提供者名稱。

**傳回：** JMS 提供者名稱。

**擲出：** JMSException - 如果 meta 資料擷取期間，JMS 實作中發生內部錯誤的話。

### getProviderVersion

```
public java.lang.String getProviderVersion()
              throws JMSException
```

取得 JMS 提供者版本。

**傳回：** JMS 提供者版本。

**擲出：** JMSException - 如果 meta 資料擷取期間，JMS 實作中發生內部錯誤的話。

### getProviderMajorVersion

```
public int getProviderMajorVersion() throws JMSException
```

取得 JMS 提供者主要版本號碼。

**傳回：** JMS 提供者主要版本號碼。

**擲出：** JMSException - 如果 meta 資料擷取期間，JMS 實作中發生內部錯誤的話。

### getProviderMinorVersion

```
public int getProviderMinorVersion() throws JMSException
```

取得 JMS 提供者次要版本號碼。

**傳回：** JMS 提供者次要版本號碼。

**擲出：** JMSException - 如果 meta 資料擷取期間，JMS 實作中發生內部錯誤的話。

### toString *

```
public String toString()
```

**置換：** Object 類別中的 toString。

## DeliveryMode

public interface **DeliveryMode**

JMS 所支援的遞送模式。

### 欄位

#### NON_PERSISTENT

```
public static final int NON_PERSISTENT
```

這是成本最低的遞送模式，因爲它不需要將訊息記錄到穩定的儲存體中。

#### PERSISTENT

```
public static final int PERSISTENT
```

這個模式指示 JMS 提供者將訊息記錄到穩定的儲存體中，作爲從屬站傳送作業的一部份。

# Destination

public interface **Destination**

子介面：**Queue**、**TemporaryQueue**、**TemporaryTopic** 和 **Topic**

MQSeries 類別：**MQDestination**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
```

Destination 物件封裝特定提供者專用的位址。

另請參閱：**Queue**、**TemporaryQueue**、**TemporaryTopic** 和 **Topic**

## MQSeries 建構子

### MQDestination

```
public MQDestination()
```

## 方法

### setDescription *

```
public void setDescription(String x)
```

物件的摘要說明。

### getDescription *

```
public String getDescription()
```

取得物件的說明。

### setPriority *

```
public void setPriority(int priority) throws JMSException
```

用來置換傳送到這個目的地之所有訊息的優先順序。

### getPriority *

```
public int getPriority()
```

取得置換優先順序值。

### setExpiry *

```
public void setExpiry(int expiry) throws JMSException
```

用來置換傳送到這個目的地之所有訊息的期限。

### getExpiry *

```
public int getExpiry()
```

取得這個目的地的期限值。

### setPersistence *

```
public void setPersistence(int persistence)
                                  throws JMSException
```

## Destination

用來置換傳送到這個目的地之所有訊息的持續性。

### getPersistence *

```
public int getPersistence()
```

取得這個目的地的持續性值。

### setTargetClient *

```
public void setTargetClient(int targetClient)
                                    throws JMSException
```

指出遠端應用程式是否符合 JMS 標準的旗號。

### getTargetClient *

```
public int getTargetClient()
```

取得 JMS 標準指示器旗號。

### setCCSID *

```
public void setCCSID(int x) throws JMSException
```

傳送到這個目的地之訊息文字字串用來編碼的字集。請參閱第103頁的表13，以取得可用值的清單。 預設值是 1208 (UTF8)。

### getCCSID *

```
public int getCCSID()
```

取得這個目的地所用字集的名稱。

### setEncoding *

```
public void setEncoding(int x) throws JMSException
```

指定傳送到這個目的地之訊息中的數值欄位所用的編碼。請參閱第103頁的表13，以取得可用值的清單。

### getEncoding *

```
public int getEncoding()
```

取得這個目的地所用的編碼。

# ExceptionListener

public interface **ExceptionListener**

如果 JMS 提供者偵測到連線的嚴重問題，且連線有登錄過的 ExceptionListener，這時 JMS 提供者會通知這個 ExceptoinListener。它會呼叫接聽器的 onException() 方法，傳給它一個說明問題的 MNSException，來完成這個動作。

這使得從屬站能夠非同步得到問題的通知。有些連線只會使用訊息，因此沒有任何其它方法可得知連線已經失效。

在下列情況下，會送出異常狀況：
- 接收非同步訊息失敗
- 訊息擲出執行時期異常狀況

## 方法

### onException

```
public void onException(JMSException exception)
```

通知使用者發生 JMS 異常狀況。

**參數：** exception - JMS 異常狀況。這些是從非同步訊息遞送中產生的異常狀況。它們通常表示接收佇列管理程式中的訊息時所發生的問題，也可能是 JMS 實作的內部錯誤。

# MapMessage

public interface **MapMessage**
extends **Message**

MQSeries 類別：**JMSMapMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSMapMessage
```

MapMessage 用來傳送一組名稱/值配對，名稱是 Strings，值是 Java 基本類型。這些項目可以循序存取，也可以依名稱而隨機存取。項目次序沒有定義。

另請參閱：**BytesMessage**、**Message**、**ObjectMessage**、**StreamMessage** 和 **TextMessage**

## 方法

### getBoolean

```
public boolean getBoolean(java.lang.String name)
                                                throws JMSException
```

傳回含給定名稱的 Boolean 值。

**參數：** name - Boolean 的名稱。

**傳回：** 含給定名稱的 Boolean 值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。
- MessageFormatException - 如果這個類型轉換無效的話。

### getByte

```
public byte getByte(java.lang.String name)
                                       throws JMSException
```

傳回含給定名稱的 Byte 值。

**參數：** name - Byte 的名稱。

**傳回：** 給定名稱的 Byte 值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。
- MessageFormatException - 如果這個類型轉換無效的話。

### getShort

```
public short getShort(java.lang.String name) throws JMSException
```

傳回含給定名稱的 Short 值。

**參數：** name - Short 的名稱。

**傳回：** 含給定名稱的 Short 值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。
- MessageFormatException - 如果這個類型轉換無效的話。

### getChar

```
public char getChar(java.lang.String name)
                                        throws JMSException
```

傳回含給定名稱的 Unicode 字元值。

**參數：** name - Unicode 字元的名稱。

**傳回：** 含給定名稱的 Unicode 字元值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。
- MessageFormatException - 如果這個類型轉換無效的話。

### getInt

```
public int getInt(java.lang.String name)
                                          throws JMSException
```

傳回含給定名稱的 Integer 值。

**參數：** name - Integer 的名稱。

**傳回：** 含給定名稱的 Integer 值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。
- MessageFormatException - 如果這個類型轉換無效的話。

### getLong

```
public long getLong(java.lang.String name)
                                        throws JMSException
```

傳回含給定名稱的 Long 值。

**參數：** name - Long 的名稱。

**傳回：** 含給定名稱的 Long 值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。
- MessageFormatException - 如果這個類型轉換無效的話。

### getFloat

```
public float getFloat(java.lang.String name) throws JMSException
```

傳回含給定名稱的 Float 值。

**參數：** name - Float 的名稱。

**傳回：** 含給定名稱的 Float 值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。
- MessageFormatException - 如果這個類型轉換無效的話。

### getDouble

```
public double getDouble(java.lang.String name) throws JMSException
```

傳回含給定名稱的 Double 值。

**參數：** name - Double 的名稱。

**傳回：** 含給定名稱的 Double 值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。
- MessageFormatException - 如果這個類型轉換無效的話。

### getString

```
public java.lang.String getString(java.lang.String name)
                                                  throws JMSException
```

傳回含給定名稱的 String 值。

**參數：** name - String 的名稱。

**傳回：** 含給定名稱的 String 值。如果沒有這個名稱的項目的話，會傳回空值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。
- MessageFormatException - 如果這個類型轉換無效的話。

### getBytes

```
public byte[] getBytes(java.lang.String name) throws JMSException
```

傳回含給定名稱的 Byte Array 值。

**參數：** name - Byte Array 的名稱。

**傳回：** 含給定名稱的 Byte Array 值。如果沒有這個名稱的項目的話，會傳回空值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。
- MessageFormatException - 如果這個類型轉換無效的話。

### getObject

```
public java.lang.Object getObject(java.lang.String name)
                                        throws JMSException
```

傳回含給定名稱的 Java 物件值。這個方法會以物件格式傳回利用 setObject 方法呼叫或對等的基本 set 方法而儲存到 Map 中的值。

**參數：** name - Java 物件的名稱。

**傳回：** 採物件格式，含給定名稱的 Java 物件值副本（如果設爲整數的話，會傳回 Integer）。如果沒有這個名稱的項目的話，會傳回空值。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。

### getMapNames

```
public java.util.Enumeration getMapNames() throws JMSException
```

傳回所有 Map 訊息名稱之列舉。

**傳回：** 這個 Map 訊息中的所有名稱之列舉。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。

### setBoolean

```
public void setBoolean(java.lang.String name,
                       boolean value) throws JMSException
```

將含給定名稱的 Boolean 值設到 Map 中。

**參數：**

- name - Boolean 的名稱。
- value - 要在 Map 中設定的 Boolean 值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

## MapMessage

### setByte

```
public void setByte(java.lang.String name,
                    byte value) throws JMSException
```

將含給定名稱的 byte 值設到 Map 中。

**參數：**

- name - Byte 的名稱。
- value - 要在 Map 中設定的 byte 值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

### setShort

```
public void setShort(java.lang.String name,
                     short value) throws JMSException
```

將含給定名稱的 Short 值設到 Map 中。

**參數：**

- name - Short 的名稱。
- value - 要在 Map 中設定的 Short 值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

### setChar

```
public void setChar(java.lang.String name,
                    char value) throws JMSException
```

將含給定名稱的 Unicode 字元值設到 Map 中。

**參數：**

- name - Unicode 字元的名稱。
- value - 要在 Map 中設定的 Unicode 字元值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

### setInt

```
public void setInt(java.lang.String name,
                   int value) throws JMSException
```

將含給定名稱的 Integer 值設到 Map 中。

**參數：**

- name - Integer 的名稱。
- value - 要在 Map 中設定的 Integer 值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。

- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

### setLong

```
public void setLong(java.lang.String name,
                    long value) throws JMSException
```

將含給定名稱的 Long 值設到 Map 中。

**參數：**

- name - Long 的名稱。
- value - 要在 Map 中設定的 Long 值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

### setFloat

```
public void setFloat(java.lang.String name,
                     float value) throws JMSException
```

將含給定名稱的 Float 值設到 Map 中。

**參數：**

- name - Float 的名稱。
- value - 要在 Map 中設定的 Float 值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

### setDouble

```
public void setDouble(java.lang.String name,
                      double value) throws JMSException
```

將含給定名稱的 Double 值設到 Map 中。

**參數：**

- name - Double 的名稱。
- value - 要在 Map 中設定的 Double 值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

## MapMessage

### setString

```
public void setString(java.lang.String name,
                      java.lang.String value) throws JMSException
```

將含給定名稱的 String 值設到 Map 中。

**參數：**

- name - String 的名稱。
- value - 要在 Map 中設定的 String 值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

### setBytes

```
public void setBytes(java.lang.String name,
                     byte[] value) throws JMSException
```

將含給定名稱的位元組陣列值設到 Map 中。

**參數：**

- name - Byte Array 的名稱。
- value - 要在 Map 中設定的位元組陣列值。
  陣列會經過複製，使對映中的值不會因陣列後續的修改而改變。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

### setBytes

```
public void setBytes(java.lang.String name,
                     byte[] value,
                     int offset,
                     int length) throws JMSException
```

將含給定名稱的位元組陣列值的一部份設到 Map 中。

陣列會經過複製，使對映中的值不會因陣列後續的修改而改變。

**參數：**

- name - Byte Array 的名稱。
- value - 要在 Map 中設定的位元組陣列值。
- offset - 位元組陣列內的起始偏移。
- length - 要複製的位元組數目。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

### setObject

```
public void setObject(java.lang.String name,
                      java.lang.Object value) throws JMSException
```

將含給定名稱的 Java 物件值設到 Map 中。這個方法只適用於物件基本類型（如 Integer、Double、Long）、String 及位元組陣列。

**參數：**

- name - Java 物件的名稱。
- value - 要設在 Map 中的 Java 物件值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。
- MessageFormatException - 如果物件無效的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

### itemExists

```
public boolean itemExists(java.lang.String name)
                                    throws JMSException
```

檢查這個 MapMessage 中是否有項目存在。

**參數：** name - 要測試的項目名稱。

**傳回：** 如果項目存在，即爲 true。

**擲出：** JMSException - 如果發生 JMS 錯誤的話。

# Message

public interface **Message**
子介面：**BytesMessage**、**MapMessage**、**ObjectMessage**、**StreamMessage** 和 **TextMessage**

MQSeries 類別：**JMSMessage**

```
java.lang.Object
  |
  +----com.ibm.jms.MQJMSMessage
```

Message 介面是所有 JMS 訊息的根介面。它會定義 JMS 標題及所有訊息使用的認可方法。

## 欄位

### DEFAULT_DELIVERY_MODE

```
public static final int DEFAULT_DELIVERY_MODE
```

預設遞送模式值。

### DEFAULT_PRIORITY

```
public static final int DEFAULT_PRIORITY
```

預設優先順序值。

### DEFAULT_TIME_TO_LIVE

```
public static final long DEFAULT_TIME_TO_LIVE
```

預設存活時間值。

## 方法

### getJMSMessageID

```
public java.lang.String getJMSMessageID()
                                    throws JMSException
```

取得訊息 ID。

**傳回：** 訊息 ID。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法取得訊息 ID 的話。

**另請參閱：**

setJMSMessageID()

### setJMSMessageID

```
public void setJMSMessageID(java.lang.String id)
                                          throws JMSException
```

設定訊息 ID。

當傳送訊息時，任何利用這個方法設定的值都會遭到忽略，但這個方法可用來變更收到的訊息其中的值。

**參數：** id - 訊息 ID。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法設定訊息 ID 的話。

**另請參閱：**

getJMSMessageID()

### getJMSTimestamp

```
public long getJMSTimestamp() throws JMSException
```

取得訊息時間戳記。

**傳回：** 訊息時間戳記。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法取得時間戳記的話。

**另請參閱：**

setJMSTimestamp()

### setJMSTimestamp

```
public void setJMSTimestamp(long timestamp)
                                        throws JMSException
```

設定訊息時間戳記。

當傳送訊息時，任何利用這個方法設定的值都會遭到忽略，但這個方法可用來變更收到的訊息其中的值。

**參數：** timestamp - 這個訊息的時間戳記。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法設定時間戳記的話。

**另請參閱：**

getJMSTimestamp()

### getJMSCorrelationIDAsBytes

```
public byte[] getJMSCorrelationIDAsBytes()
                                                  throws JMSException
```

以訊息位元組陣列形式取得相互關係 ID。

**傳回：** 採用位元組陣列形式的訊息相互關係 ID。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法取得相互關係 ID 的話。

**另請參閱：**

setJMSCorrelationID()、getJMSCorrelationID()、setJMSCorrelationIDAsBytes()

### setJMSCorrelationIDAsBytes

```
public void setJMSCorrelationIDAsBytes(byte[]
                                       correlationID)
                                              throws JMSException
```

以訊息位元組陣列形式設定相互關係 ID。從屬站可以利用這個呼叫來設定 correlationID 爲等於前一個訊息的 messageID，或等於應用程式特定字串。應用程式特定字串的起始字元不能是 ID:

**參數：** correlationID - 字串形式的 correlation ID，或所參照之訊息的訊息 ID。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法設定相互關係 ID 的話。

**另請參閱：**

setJMSCorrelationID()、getJMSCorrelationID()、getJMSCorrelationIDAsBytes()

### getJMSCorrelationID

```
public java.lang.String getJMSCorrelationID()
                                          throws JMSException
```

取得訊息的相互關係 ID。

**傳回：** String 的訊息相互關係 ID。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法取得相互關係 ID 的話。

**另請參閱：**

setJMSCorrelationID()、getJMSCorrelationIDAsBytes()、setJMSCorrelationIDAsBytes()

### setJMSCorrelationID

```
public void setJMSCorrelationID
                  (java.lang.String correlationID)
                                 throws JMSException
```

設定訊息的相互關係 ID。

從屬站可以利用 JMSCorrelationID 標題欄位來鏈結不同的訊息。標準用法是將回應訊息及其要求訊息鏈結起來。

**註:** JMSCorrelationID 使用的 byte[] 值不具可攜性。

**參數：** correlationID - 所參照之訊息的訊息 ID。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法設定相互關係 ID 的話。

**另請參閱：**

getJMSCorrelationID()、getJMSCorrelationIDAsBytes()、setJMSCorrelationIDAsBytes()

### getJMSReplyTo

```
public Destination getJMSReplyTo() throws JMSException
```

取得這個訊息的回答應該送往的位置。

**傳回：** 這個訊息的回應應該送往的位置。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法取得 ReplyTo Destination 的話。

**另請參閱：**

setJMSReplyTo()

### setJMSReplyTo

```
public void setJMSReplyTo(Destination replyTo)
                                    throws JMSException
```

設定這個訊息的回答應該送往的位置。

**參數：** replyTo - 這個訊息的回應應該送往的位置。空值表示預期沒有回覆。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法設定 ReplyTo Destination 的話。

**另請參閱：**

getJMSReplyTo()

### getJMSDestination

```
public Destination getJMSDestination() throws JMSException
```

取得這個訊息的目的地。

**傳回：** 這個訊息的目的地。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法取得 JMS Destination 的話。

**另請參閱：**

setJMSDestination()

### setJMSDestination

```
public void setJMSDestination(Destination destination)
                                                throws JMSexception
```

設定這個訊息的目的地。

當傳送訊息時，任何利用這個方法設定的值都會遭到忽略，但這個方法可用來變更收到的訊息其中的值。

**參數：** destination - 這個訊息的目的地。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法設定 JMS Destination 的話。

**另請參閱：**

getJMSDestination()

### getJMSDeliveryMode

```
public int getJMSDeliveryMode() throws JMSException
```

取得這個訊息的遞送模式。

**傳回：** 這個訊息的遞送模式。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法取得 JMS DeliveryMode 的話。

**另請參閱：**

setJMSDeliveryMode()、DeliveryMode

## Message

### setJMSDeliveryMode

```
public void setJMSDeliveryMode(int deliveryMode)
                                    throws JMSException
```

設定這個訊息的遞送模式。

當傳送訊息時，任何利用這個方法設定的值都會遭到忽略，但這個方法可用來變更收到的訊息其中的值。

如果要在傳送訊息時改變遞送模式，請使用 QueueSender 或 TopicPublisher 的 setDeliveryMode 方法（這個方法從 MessageProducer 繼承而來）。

**參數：** deliveryMode - 這個訊息的遞送模式。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法設定 JMS DeliveryMode 的話。

**另請參閱：**

getJMSDeliveryMode(), DeliveryMode

### getJMSRedelivered

```
public boolean getJMSRedelivered() throws JMSException
```

取得是否重新遞送這個訊息的指示。

如果從屬站收到設定了重新遞送指示的訊息，很可能（不必然）這個訊息先前已遞送給從屬站，但從屬站先前並沒有認可收到這個訊息。

**傳回：** 如果重新遞送這個訊息的話，即設爲 true。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法取得 JMS Redelivered 旗號的話。

**另請參閱：**

setJMSRedelivered()

### setJMSRedelivered

```
public void setJMSRedelivered(boolean redelivered)
                                        throws JMSException
```

設定指出是否重新遞送這個訊息。

當傳送訊息時，任何利用這個方法設定的值都會遭到忽略，但這個方法可用來變更收到的訊息其中的值。

**參數：** 是否重新遞送這個訊息的指示。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法設定 JMSRedelivered 旗號的話。

**另請參閱：**

getJMSRedelivered()

### getJMSType

```
public java.lang.String getJMSType() throws JMSException
```

取得訊息類型。

**傳回：** 訊息類型。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法取得 JMS 訊息類型的話。

**另請參閱：**

setJMSType()

### setJMSType

```
public void setJMSType(java.lang.String type)
                                          throws JMSException
```

設定訊息類型。

不論應用程式是否使用，JMS 從屬站都應該指派一個類型值。這可以確保會將它正確設定給需要它的提供者。

**參數：** type - 訊息的類別。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法設定 JMS 訊息類型的話。

**另請參閱：**

getJMSType()

### getJMSExpiration

```
public long getJMSExpiration() throws JMSException
```

取得訊息的期限值。

**傳回：** 訊息的到期時間。這是從屬站指定的存活時間值及傳送時的 GMT 的總和。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法取得 JMS 訊息期限的話。

**另請參閱：**

setJMSExpiration()

### setJMSExpiration

```
public void setJMSExpiration(long expiration)
                                           throws JMSException
```

設定訊息的期限值。

當傳送訊息時，任何利用這個方法設定的值都會遭到忽略，但這個方法可用來變更收到的訊息其中的值。

**參數：** expiration - 訊息的期限時間。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法設定 JMS 訊息期限的話。

**另請參閱：**

getJMSExpiration()

### getJMSPriority

```
public int getJMSPriority() throws JMSException
```

取得訊息優先順序。

**傳回：** 訊息優先順序。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法取得 JMS 訊息優先順序的話。

**另請參閱：**

優先順序層次的 setJMSPriority()

### setJMSPriority

```
public void setJMSPriority(int priority)
                                    throws JMSException
```

設定這個訊息的優先順序。

JMS 會定義一個 10 層次的優先順序值，0 是最低優先順序，9 是最高優先順序。另外，從屬站應該將優先順序 0-4 視爲正常優先順序的分級，將優先順序 5-9 視爲特別優先順序的分級。

**參數：** priority - 這個訊息的優先順序。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法設定 JMS 訊息優先順序的話。

**另請參閱：**

getJMSPriority()

### clearProperties

```
public void clearProperties() throws JMSException
```

清除訊息內容。標題欄位和訊息主體不會清除。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法清除 JMS 訊息內容的話。

### propertyExists

```
public boolean propertyExists(java.lang.String name)
                                              throws JMSException
```

檢查內容值存不存在。

**參數：** name - 要測試的內容名稱。

**傳回：** 如果內容存在，即爲 true。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法檢查內容存不存在的話。

### getBooleanProperty

```
public boolean getBooleanProperty(java.lang.String name)
                                                  throws JMSException
```

傳回含給定名稱的 Boolean 內容值。

**參數：** name - Boolean 內容的名稱。

**傳回：** 含給定名稱的 Boolean 內容值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法取得內容的話。
- MessageFormatException - 如果這個類型轉換無效的話

### getByteProperty

```
public byte getByteProperty(java.lang.String name)
                                              throws JMSException
```

傳回含給定名稱的 Byte 內容值。

**參數：** name - Byte 內容的名稱。

**傳回：** 含給定名稱的 Byte 內容值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法取得內容的話。
- MessageFormatException - 如果這個類型轉換無效的話。

## Message

### getShortProperty

```
public short getShortProperty(java.lang.String name)
                                                throws JMSException
```

傳回含給定名稱的 Short 內容值。

**參數：** name - Short 內容的名稱。

**傳回：** 含給定名稱的 Short 內容值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法取得內容的話。
- MessageFormatException - 如果這個類型轉換無效的話。

### getIntProperty

```
public int getIntProperty(java.lang.String name)
                                              throws JMSException
```

傳回含給定名稱的 Integer 內容值。

**參數：** name - Integer 內容的名稱。

**傳回：** 含給定名稱的 Integer 內容值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法取得內容的話。
- MessageFormatException - 如果這個類型轉換無效的話。

### getLongProperty

```
public long getLongProperty(java.lang.String name)
                                               throws JMSException
```

傳回含給定名稱的 Long 內容值。

**參數：** name - Long 內容的名稱。

**傳回：** 含給定名稱的 Long 內容值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法取得內容的話。
- MessageFormatException - 如果這個類型轉換無效的話。

### getFloatProperty

```
public float getFloatProperty(java.lang.String name)
                                                throws JMSException
```

傳回含給定名稱的 Float 內容值。

**參數：** name - Float 內容的名稱。

**傳回：** 含給定名稱的 Float 內容值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法取得內容的話。
- MessageFormatException - 如果這個類型轉換無效的話。

### getDoubleProperty

```
public double getDoubleProperty(java.lang.String name)
                                 throws JMSException
```

傳回含給定名稱的 Double 內容值。

**參數：** name - Double 內容的名稱。

**傳回：** 含給定名稱的 Double 內容值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法取得內容的話。
- MessageFormatException - 如果這個類型轉換無效的話。

### getStringProperty

```
public java.lang.String getStringProperty (java.lang.String name)
                                           throws JMSException
```

傳回含給定名稱的 String 內容值。

**參數：** name - String 內容的名稱

**傳回：** 含給定名稱的 String 內容值。如果沒有這個名稱的內容的話，會傳回空值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法取得內容的話。
- MessageFormatException - 如果這個類型轉換無效的話。

### getObjectProperty

```
public java.lang.Object getObjectProperty (java.lang.String name)
                                           throws JMSException
```

傳回含給定名稱的 Java 物件內容值。

**參數：** name - Java 物件內容的名稱。

**傳回：** 採物件格式，含給定名稱的 Java 物件內容值（例如，如果它設爲 int 的話，會傳回整數。）如果沒有這個名稱的內容的話，會傳回空值。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法取得內容的話。

### getPropertyNames

```
public java.util.Enumeration getPropertyNames()
                                 throws JMSException
```

傳回所有內容名稱之列舉。

**傳回：** 所有內容值名稱之列舉。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法取得內容名稱的話。

### setBooleanProperty

```
public void setBooleanProperty(java.lang.String name,
                               boolean value) throws JMSException
```

將含給定名稱的 Boolean 內容值設到 Message 中。

**參數：**

- name - Boolean 內容的名稱。

- value - 要在 Message 中設定的 Boolean 內容值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法設定 Property 的話。
- MessageNotWriteableException - 如果內容是唯讀的話。

### setByteProperty

```
public void setByteProperty(java.lang.String name,
                            byte value) throws JMSException
```

將含給定名稱的 Byte 內容值設到 Message 中。

**參數：**

- name - Byte 內容的名稱。
- value - 要在 Message 中設定的 Byte 內容值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法設定 Property 的話。
- MessageNotWriteableException - 如果內容是唯讀的話。

### setShortProperty

```
public void setShortProperty(java.lang.String name,
                             short value) throws JMSException
```

將含給定名稱的 Short 內容值設到 Message 中。

**參數：**

- name - Short 內容的名稱。
- value - 要在 Message 中設定的 Short 內容值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法設定 Property 的話。
- MessageNotWriteableException - 如果內容是唯讀的話。

### setIntProperty

```
public void setIntProperty(java.lang.String name,
                           int value) throws JMSException
```

將含給定名稱的 Integer 內容值設到 Message 中。

**參數：**

- name - Integer 內容的名稱。
- value - 要在 Message 中設定的 Integer 內容值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法設定 Property 的話。
- MessageNotWriteableException - 如果內容是唯讀的話。

### setLongProperty

```
public void setLongProperty(java.lang.String name,
                            long value) throws JMSException
```

將含給定名稱的 Long 內容值設到 Message 中。

**參數：**

- name - Long 內容的名稱。
- value - 要在 Message 中設定的 Long 內容值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法設定 Property 的話。
- MessageNotWriteableException - 如果內容是唯讀的話。

### setFloatProperty

```
public void setFloatProperty(java.lang.String name,
                             float value) throws JMSException
```

將含給定名稱的 Float 內容值設到 Message 中。

**參數：**

- name - Float 內容的名稱。
- value - 要在 Message 中設定的 Float 內容值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法設定內容的話。
- MessageNotWriteableException - 如果內容是唯讀的話。

### setDoubleProperty

```
public void setDoubleProperty(java.lang.String name,
                              double value) throws JMSException
```

將含給定名稱的 Double 內容值設到 Message 中。

**參數：**

- name - Double 內容的名稱。
- value - 要在 Message 中設定的 Double 內容值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法設定內容的話。
- MessageNotWriteableException - 如果內容是唯讀的話。

### setStringProperty

```
public void setStringProperty(java.lang.String name,
                              java.lang.String value) throws JMSException
```

將含給定名稱的 String 內容值設到 Message 中。

**參數：**

- name - String 內容的名稱。
- value - 要在 Message 中設定的 String 內容值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法設定內容的話。
- MessageNotWriteableException - 如果內容是唯讀的話。

### setObjectProperty

```
public void setObjectProperty(java.lang.String name,
                              java.lang.Object value) throws JMSException
```

將含給定名稱的內容值設到 Message 中。

**參數：**

- name - Java 物件內容的名稱。
- value - 要在 Message 中設定的 Java 物件內容值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法設定 Property 的話。
- MessageFormatException - 如果物件無效的話。
- MessageNotWriteableException - 如果內容是唯讀的話。

### acknowledge

```
public void acknowledge() throws JMSException
```

認可這個訊息及階段作業先前收到的所有訊息。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法認可的話。

### clearBody

```
public void clearBody() throws JMSException
```

清除訊息主體。訊息的所有其它部份會維持不變。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而失敗的話。

# MessageConsumer

public interface **MessageConsumer**
子介面：**QueueReceiver** 和 **TopicSubscriber**

MQSeries 類別：**MQMessageConsumer**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageConsumer
```

所有訊息消費者的母項介面。 從屬站利用訊息消費者接收來自 Destination 的訊息。

## 方法

### getMessageSelector

```
public java.lang.String getMessageSelector()
                                  throws JMSException
```

取得這個訊息消費者的訊息選取元表示式。

**傳回：** 這個訊息消費者的訊息選取元。

**擲出：** JMSException - 如果 JMS 因 JMS 錯誤而無法取得訊息選取元的話。

### getMessageListener

```
public MessageListener getMessageListener()
                                      throws JMSException
```

取得訊息消費者的 MessageListener。

**傳回：** 訊息消費者的接聽器，如果沒有設定接聽器的話，即爲空值。

**擲出：** JMSException - 如果 JMS 因 JMS 錯誤而無法取得訊息接聽器的話。

**另請參閱：**

setMessageListener

### setMessageListener

```
public void setMessageListener(MessageListener listener)
                                            throws JMSException
```

設定訊息消費者的 MessageListener。

**參數：** messageListener - 遞送到這個接聽器的訊息。

**擲出：** JMSException - 如果 JMS 因 JMS 錯誤而無法設定訊息接聽器的話。

**另請參閱：**

getMessageListener

### receive

```
public Message receive() throws JMSException
```

接收爲了這個訊息消費者而產生的下一個訊息。

**傳回：** 爲了這個訊息消費者而產生的下一個訊息。

**擲出：** JMSException - 如果 JMS 因錯誤而無法接收下個訊息的話。

## MessageConsumer

### receive

```
public Message receive(long timeOut) throws JMSException
```

接收在指定的逾時間隔內到達的下個訊息。逾時值 0 會使呼叫無限等待，直到訊息到達爲止。

**參數：** timeout - 逾時值（以毫秒爲單位）。

**傳回：** 就這個訊息消費者而產生的下個訊息，如果沒有下個訊息的話，即爲空值。

**擲出：** JMSException - 如果 JMS 因錯誤而無法接收下個訊息的話。

### receiveNoWait

```
public Message receiveNoWait() throws JMSException
```

如果立即有下個訊息的話，即接收下個訊息。

**傳回：** 就這個訊息消費者而產生的下個訊息，如果沒有下個訊息的話，即爲空值。

**擲出：** JMSException - 如果 JMS 因錯誤而無法接收下個訊息的話。

### close

```
public void close() throws JMSException
```

由於提供者可在 JVM 之外代表 MessageConsumer 來配置某些資源，因此，當不需要它們時，從屬站應該關閉它們。您終究不能依賴記憶體回收來收回這些資源，因爲記憶體回收的進行可能不夠快。

在進行中的接收或訊息接聽器完成作業之前，這個呼叫不會運作。

**擲出：** JMSException - 如果 JMS 因錯誤而無法關閉消費者的話。

# MessageListener

public interface **MessageListener**

MessageListener 用來接收非同步遞送的訊息。

## 方法

### onMessage

```
public void onMessage(Message message)
```

傳送一則訊息給接聽器。

**參數：** message - 傳遞給接聽器的訊息。

**另請參閱**

Session.setMessageListener

# MessageProducer

public interface **MessageProducer**
子介面：**QueueSender** 和 **TopicPublisher**

MQSeries 類別：**MQMessageProducer**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageProducer
```

從屬站利用訊息產生者來傳送訊息到 Destination。

## MQSeries 建構子

### MQMessageProducer

```
public MQMessageProducer()
```

## 方法

### setDisableMessageID

```
public void setDisableMessageID(boolean value)
                                                throws JMSException
```

設定是否要停用訊息 ID。

依預設，會啓用訊息 ID。

**註:** 在 MQSeries Class for Java 訊息服務實作中，會忽略這個方法。

**參數：** value - 指出是否要停用訊息 ID。

**擲出：** JMSException - 如果 JMS 因內部錯誤而無法設定停用的訊息 ID 的話。

### getDisableMessageID

```
public boolean getDisableMessageID() throws JMSException
```

取得是否要停用訊息 ID 的指示。

**傳回：** 是否要停用訊息 ID 的指示。

**擲出：** JMSException - 如果 JMS 因內部錯誤而無法取得停用的訊息 ID 的話。

### setDisableMessageTimestamp

```
public void setDisableMessageTimestamp(boolean value)
                                                      throws JMSException
```

設定是否要停用訊息時間戳記。

依預設，會啓用訊息時間戳記。

**註:** 在 MQSeries Class for Java 訊息服務實作中，會忽略這個方法。

**參數：** value - 指出是否要停用訊息時間戳記。

**擲出：** JMSException - 如果 JMS 因內部錯誤而無法設定停用的訊息時間戳記的話。

### getDisableMessageTimestamp

```
public boolean getDisableMessageTimestamp()
                                        throws JMSException
```

取得是否要停用訊息時間戳記的指示。

**傳回：** 是否要停用訊息 ID 的指示。

**擲出：** JMSException - 如果 JMS 因內部錯誤而無法取得停用的訊息時間戳記的話。

### setDeliveryMode

```
public void setDeliveryMode(int deliveryMode)
                                         throws JMSException
```

設定產生者的預設遞送模式。

依預設，遞送模式會設爲 PERSISTENT。

**參數：** deliveryMode - 這個訊息產生者的訊息遞送模式。

**擲出：** JMSException - 如果 JMS 因內部錯誤而無法設定遞送模式的話。

**另請參閱：**

getDeliveryMode

### getDeliveryMode

```
public int getDeliveryMode() throws JMSException
```

取得產生者的預設遞送模式。

**傳回：** 這個訊息產生者的訊息遞送模式。

**擲出：** JMSException - 如果 JMS 因內部錯誤而無法取得遞送模式的話。

**另請參閱：**

setDeliveryMode

### setPriority

```
public void setPriority(int priority) throws JMSException
```

設定產生者的預設優先順序。

依預設，優先順序會設爲 4。

**參數：** priority - 這個訊息產生者的訊息優先順序。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法設定優先順序的話。

**另請參閱：**

getPriority

### getPriority

```
public int getPriority() throws JMSException
```

取得產生者的預設優先順序。

**傳回：** 這個訊息產生者的訊息優先順序。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法取得優先順序的話。

**另請參閱：**

setPriority

### setTimeToLive

```
public void setTimeToLive(long timeToLive)
                                    throws JMSException
```

設定從產生的訊息的分派時間開始，訊息系統應該保留這個訊息的預設時間長度（毫秒）。

依預設，「存活時間」會設爲零。

**參數：** timeToLive - 訊息的存活時間（毫秒）；零是無限制。

**擲出：** JMSException - 如果 JMS 因內部錯誤而無法設定存活時間的話。

**另請參閱：**

getTimeToLive

### getTimeToLive

```
public long getTimeToLive() throws JMSException
```

取得從產生的訊息的分派時間開始，訊息系統應該保留這個訊息的預設時間長度（毫秒）。

**傳回：** 訊息的存活時間（毫秒）；零是無限制。

**擲出：** JMSException - 如果 JMS 因內部錯誤而無法取得存活時間的話。

**另請參閱：**

setTimeToLive

## close

```
public void close() throws JMSException
```

由於提供者可在 JVM 之外代表 MessageProducer 來配置某些資源，因此，當不需要它們時，從屬站應該關閉它們。您終究不能依賴記憶體回收來收回這些資源，因爲記憶體回收的進行可能不夠快。

**擲出：** JMSException - 如果 JMS 因錯誤而無法關閉產生者的話。

# MQQueueEnumeration *

public class **MQQueueEnumeration**
extends **Object**
implements **Enumeration**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQQueueEnumeration
```

佇列中的訊息列舉。在 JMS 規格中沒有定義這個類別，它是呼叫 MQQueueBrowser 的 getEnumeration 方法來建立的。這個類別含有控制瀏覽游標的基本 MQQueue 案例。當游標移出佇列尾端之後，便會關閉佇列。

沒有任何方法可以重設這個類別的案例 - 它是一種單次作業的機制。

另請參閱：**MQQueueBrowser**

## 方法

### hasMoreElements

```
public boolean hasMoreElements()
```

指出是否能傳回另一個訊息。

### nextElement

```
public Object nextElement() throws NoSuchElementException
```

傳回現行訊息。

如果 hasMoreElements() 傳回 'true'，nextElement() 永遠會傳回一則訊息。傳回的訊息有可能在 hasMoreElements() 和 nextElement 呼叫之間傳遞它的到期日。

# ObjectMessage

public interface **ObjectMessage**
extends **Message**

MQSeries 類別：**JMSObjectMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSObjectMessage
```

ObjectMessage 用來傳送含有可序列化之 Java 物件的訊息。它繼承 Message，且加入了含有單一 Java 參照的主體。只有可序列化 Java 物件能夠使用。

另請參閱：**BytesMessage**、**MapMessage**、**Message**、**StreamMessage** 和 **TextMessage**

## 方法

### setObject

```
public void setObject(java.io.Serializable object)
              throws JMSException
```

設定含有這個訊息資料的可序列化物件。 ObjectMessage 含有呼叫 setObject() 時的物件 Snapshot。後續的物件修改對 ObjectMessage 主體無效。

**參數：** object - 訊息的資料。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法設定物件的話。
- MessageFormatException - 如果物件序列化失敗的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

### getObject

```
public java.io.Serializable getObject()
              throws JMSException
```

取得含有這個訊息資料的可序列化物件。預設值是空值。

**傳回：** 含有這個訊息資料的可序列化物件。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法取得物件的話。
- MessageFormatException - 如果物件取消序列化失敗的話。

# Queue

public interface **Queue**
extends **Destination**
子介面：**TemporaryQueue**

MQSeries 類別：**MQQueue**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQQueue
```

Queue 物件會封裝特定提供者專用的佇列名稱。它是從屬站將佇列身份指定給 JMS 方法的方式。

## MQSeries 建構子

### MQQueue *

```
public MQQueue()
```

管理工具使用的預設建構子。

### MQQueue *

```
public MQQueue(String URIqueue)
```

建立一個新的 MQQueue 案例。如171 頁所說明，這個字串採用 URI 格式。

### MQQueue *

```
public MQQueue(String queueManagerName,
               String queueName)
```

## 方法

### getQueueName

```
public java.lang.String getQueueName()
                                  throws JMSException
```

取得這個佇列的名稱。

相依於名稱的從屬站不具可攜性。

**傳回：** 佇列名稱

**擲出：** JMSException - 如果 Queue 的 JMS 實作因內部錯誤而無法傳回佇列名稱的話。

### toString

```
public java.lang.String toString()
```

傳回列印優美的佇列名稱版本。

**傳回：** 這個佇列的特定提供者專用身份值。

**置換：** java.lang.Object 類別中的 toString

### getReference *

```
public Reference getReference() throws NamingException
```

建立這個佇列的參照。

**傳回：** 這個物件的參照。

**擲出：** NamingException

### setBaseQueueName *

```
public void setBaseQueueName(String x) throws JMSException
```

設定 MQSeries 佇列名稱的值。

**註:** 這個方法只應該由管理工具來使用。它不會嘗試將 queue:qmgr:queue 格式的字串解碼。

### getBaseQueueName *

```
public String getBaseQueueName()
```

**傳回：** MQSeries 佇列名稱的值。

### setBaseQueueManagerName *

```
public void setBaseQueueManagerName(String x) throws JMSException
```

設定 MQSeries 佇列管理程式名稱的值。

**註:** 這個方法只應該由管理工具來使用。

### getBaseQueueManagerName *

```
public String getBaseQueueManagerName()
```

**傳回：** MQSeries 佇列管理者名稱的值。

# QueueBrowser

public interface **QueueBrowser**

MQSeries 類別：**MQQueueBrowser**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQQueueBrowser
```

從屬站利用 QueueBrowser 來查看佇列中的訊息，但不移除它們。

**註:** MQSeries 類別 **MQQueueEnumeration** 用來控制瀏覽游標。

另請參閱：**QueueReceiver**

## 方法

### getQueue

```
public Queue getQueue() throws JMSException
```

取得這個佇列瀏覽器所關聯的佇列。

**傳回：** 佇列。

**擲出：** JMSException - 如果 JMS 因 JMS 錯誤而無法取得這個瀏覽器所關聯的佇列的話。

### getMessageSelector

```
public java.lang.String getMessageSelector() throws JMSException
```

取得這個佇列瀏覽器的訊息選取元表示式。

**傳回：** 這個佇列瀏覽器的訊息選取元。

**擲出：** JMSException - 如果 JMS 因 JMS 錯誤而無法取得這個瀏覽器的訊息選取元的話。

### getEnumeration

```
public java.util.Enumeration getEnumeration() throws JMSException
```

取得一項列舉，以依序瀏覽現行佇列訊息。

**傳回：** 瀏覽訊息的列舉。

**擲出：** JMSException - 如果 JMS 因 JMS 錯誤而無法取得這個瀏覽器的列舉的話。

**註:** 如果爲不存在的佇列而建立瀏覽器的話，在第一次呼叫 getEnumeration 之前，不會偵測到這個情況。

## close

```
public void close() throws JMSException
```

由於提供者可在 JVM 之外代表 QueueBrowser 來配置某些資源，因此，當不需要它們時，從屬站應該關閉它們。您終究不能依賴記憶體回收來收回這些資源，因爲記憶體回收的進行可能不夠快。

**擲出：** JMSException - 如果 JMS 因 JMS 錯誤而無法關閉這個瀏覽器的話。

# QueueConnection

public interface **QueueConnection**
extends **Connection**
子介面：**XAQueueConnection**

MQSeries 類別：**MQQueueConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection
           |
           +----com.ibm.mq.jms.MQQueueConnection
```

QueueConnection 是通往 JMS 點對點提供者的作用中連線。從屬站利用 QueueConnection 來建立一或多個產生和消費訊息的 QueueSession。

另請參閱：**Connection**、**QueueConnectionFactory** 和 **XAQueueConnection**

## 方法

### createQueueSession

```
public QueueSession createQueueSession(boolean transacted,
                                       int acknowledgeMode)
                                                throws JMSException
```

建立一個 QueueSession。

**參數：**

- transacted - 如果爲 true，會進行階段作業異動。
- acknowledgeMode - 指出消費者或從屬站是否會認可它收到的任何訊息。可能的值如下：
  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

  如果進行階段作業異動的話，會忽略這個參數。

**傳回：** 新建立的佇列階段作業。

**擲出：** JMSException - 如果 JMS Connection 因內部錯誤或沒有特定異動和認可模式之支援而無法建立階段作業的話。

### createConnectionConsumer

```
public ConnectionConsumer createConnectionConsumer
                                    (Queue queue,
                                     java.lang.String messageSelector,
                                     ServerSessionPool sessionPool,
                                     int maxMessages)
                                                throws JMSException
```

建立這個連線的連線消費者。這是不由一般 JMS 從屬站使用的專家級機能。

**參數：**

- queue - 要存取的佇列。
- messageSelector - 只遞送內容符合訊息選取元表示式的訊息。
- sessionPool - 這個連線消費者所關聯的伺服器階段作業儲存池。
- maxMessages - 可同時指派給伺服器階段作業的訊息數目上限。

**傳回：** 連線消費者。

**擲出：**

- JMSException - 如果 JMS Connection 因內部錯誤或 sessionPool 和 messageSelector 的引數無效而無法建立連線消費者的話。
- InvalidSelectorException - 如果訊息選取元無效的話。

**另請參閱：**

ConnectionConsumer

### close *

```
public void close() throws JMSException
```

**置換：** MQConnection 類別中的 close。

# QueueConnectionFactory

public interface **QueueConnectionFactory**
extends **ConnectionFactory**
子介面：**XAQueueConnectionFactory**

MQSeries 類別：**MQQueueConnectionFactory**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory
           |
           +----com.ibm.mq.jms.MQQueueConnectionFactory
```

從屬站利用 QueueConnectionFactory，以 JMS 點對點提供者來建立 QueueConnections。

另請參閱：**ConnectionFactory** 和 **XAQueueConnectionFactory**

## MQSeries 建構子

### MQQueueConnectionFactory

```
public MQQueueConnectionFactory()
```

## 方法

### createQueueConnection

```
public QueueConnection createQueueConnection()
                                          throws JMSException
```

以預設使用者身份建立一個佇列連線。這個連線是採用停止模式建立。在明確呼叫 Connection.start 方法之前，不會遞送任何訊息。

**傳回：** 新建立的佇列連線。

**擲出：**

- JMSException - 如果 JMS 提供者因內部錯誤而無法建立佇列連線的話。
- JMSSecurityException - 如果從屬站鑑別因無效使用者名稱或密碼而失效的話。

### createQueueConnection

```
public QueueConnection createQueueConnection
                                  (java.lang.String userName,
                                   java.lang.String password)
                                                  throws JMSException
```

以指定的使用者身份建立一個佇列連線。

**註：** 這個方法只能使用 JMSC.MQJMS_TP_CLIENT_MQ_TCPIP 傳輸類型（請參閱 ConnectionFactory）。這個連線是採用停止模式建立。在明確呼叫 Connection.start 方法之前，不會遞送任何訊息。

**參數：**

- userName - 呼叫端的使用者名稱。
- password - 呼叫端的密碼。

**傳回：** 新建立的佇列連線。

**擲出：**

- JMSException - 如果 JMS 提供者因內部錯誤而無法建立佇列連線的話。
- JMSSecurityException - 如果從屬站鑑別因無效使用者名稱或密碼而失效的話。

### setTemporaryModel *

```
public void setTemporaryModel(String x) throws JMSException
```

### getTemporaryModel *

```
public String getTemporaryModel()
```

### getReference *

```
public Reference getReference() throws NamingException
```

建立這個佇列連線 Factory 的參照。

**傳回：** 這個物件的參照。

**擲出：** NamingException.

### setMessage Retention*

```
public void setMessageRetention(int x) throws JMSException
```

設定 messageRetention 屬性的方法。

**參數：** 有效值如下：

- JMSC.MQJMS_MRET_YES - 不要的訊息保留在輸入佇列中。
- JMSC.MQJMS_MRET_NO - 根據其處理選項來處理不要的訊息。

### getMessage Retention*

```
public void getMessageRetention()
```

取得 messageRetention 屬性的方法。

**傳回：**

- JMSC.MQJMS_MRET_YES - 不要的訊息保留在輸入佇列中。
- JMSC.MQJMS_MRET_NO - 根據其處理選項來處理不要的訊息。

# QueueReceiver

public interface **QueueReceiver**
extends **MessageConsumer**

MQSeries 類別：**MQQueueReceiver**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageConsumer
           |
           +----com.ibm.mq.jms.MQQueueReceiver
```

從屬站利用 QueueReceiver 來接收已遞送到佇列的訊息。

另請參閱：**MessageConsumer**

這個類別繼承 **MQMessageConsumer** 中的下列方法。

- receive
- receiveNoWait
- close
- getMessageListener
- setMessageListener

## 方法

### getQueue

```
public Queue getQueue() throws JMSException
```

取得這個佇列接收端所關聯的佇列。

**傳回：** 佇列。

**擲出：** JMSException - 如果 JMS 因內部錯誤而無法取得這個接收端的佇列的話。

# QueueRequestor

public class **QueueRequestor**
extends **java.lang.Object**

```
java.lang.Object
    |
    +----javax.jms.QueueRequestor
```

JMS 提供這個 QueueRequestor Help 類別來簡化發出服務要求的程序。 QueueRequestor 建構子會得到非異動 QueueSession 和目的地 Queue。它會建立一個回應的 TemporaryQueue，並提供 request() 方法來傳送要求訊息及等待其回應。使用者可以自由建立更複雜的版本。

另請參閱：**TopicRequestor**

## 建構子

### QueueRequestor

```
public QueueRequestor(QueueSession session,
                      Queue queue)
                              throws JMSException
```

這項實作會假設階段作業參數不是異動式的，認可模式爲 AUTO_ACKNOWLEDGE 或 DUPS_OK_ACKNOWLEDGE。

**參數：**

- session - 佇列所屬的佇列階段作業。
- queue - 執行要求/回覆呼叫的佇列。

**擲出：** JMSException - 如果發生 JMS 錯誤的話。

## 方法

### request

```
public Message request(Message message)
               throws JMSException
```

傳送要求及等待回覆。 replyTo 所用的暫時佇列，每個要求應該只有一個回覆。

**參數：** message - 要傳送的訊息。

**傳回：** 回覆訊息。

**擲出：** JMSException - 如果發生 JMS 錯誤的話。

## QueueRequestor

### close

```
public void close() throws JMSException
```

由於提供者可在 JVM 之外代表 QueueRequestor 來配置某些資源，因此，當不需要它們時，從屬站應該關閉它們。您終究不能依賴記憶體回收來收回這些資源，因爲記憶體回收的進行可能不夠快。

**註:** 這個方法會關閉傳遞給 QueueRequestor 建構子的 Session 物件。

**擲出：** JMSException - 如果發生 JMS 錯誤的話。

# QueueSender

public interface **QueueSender**
extends **MessageProducer**

MQSeries 類別：**MQQueueSender**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageProducer
           |
           +----com.ibm.mq.jms.MQQueueSender
```

從屬站利用 QueueSender 來傳送訊息到佇列。

QueueSender 通常會關聯於特定 Queue。不過，也有可能建立一個未關聯於任何給定 Queue 的未識別的 QueueSender。

另請參閱：**MessageProducer**

## 方法

### getQueue

```
public Queue getQueue() throws JMSException
```

取得這個佇列傳送端所關聯的佇列。

**傳回：** 佇列。

**擲出：** JMSException - 如果 JMS 因內部錯誤而無法取得這個佇列傳送端的佇列的話。

### send

```
public void send(Message message) throws JMSException
```

傳送一則訊息到佇列中。請利用 QueueSender 的預設值遞送模式、存活時間及優先順序。

**參數：** message - 要傳送的訊息。

**擲出：**

- JMSException - 如果 JMS 因錯誤而無法傳送訊息的話。
- MessageFormatException - 如果指定無效訊息的話。
- InvalidDestinationException - 如果從屬站搭配含無效佇列之 Queue 傳送端來使用這個方法的話。

### send

```
public void send(Message message,
                 int deliveryMode,
                 int priority,
                 long timeToLive) throws JMSException
```

傳送指定遞送模式、優先順序及存活時間的訊息到佇列中。

**參數：**

- message - 要傳送的訊息。

- deliveryMode - 要使用的遞送模式。
- priority - 這個訊息的優先順序。
- timeToLive - 訊息的存活時間（以毫秒爲單位）。

**擲出：**

- JMSException - 如果 JMS 因內部錯誤而無法傳送訊息的話。
- MessageFormatException - 如果指定無效訊息的話。
- InvalidDestinationException - 如果從屬站搭配含無效佇列之 Queue 傳送端來使用這個方法的話。

### send

```
public void send(Queue queue,
                 Message message) throws JMSException
```

以 QueueSender 的預設遞送模式、存活時間及優先順序來傳送訊息至指定的佇列中。

**註：** 這個方法只能搭配未識別的 QueueSender 使用。

**參數：**

- queue - 這個訊息應該送往的佇列。
- message - 要傳送的訊息。

**擲出：**

- JMSException - 如果 JMS 因內部錯誤而無法傳送訊息的話。
- MessageFormatException - 如果指定無效訊息的話。
- InvalidDestinationException - 如果從屬站搭配無效佇列來使用這個方法的話。

### send

```
public void send(Queue queue,
                 Message message,
                 int deliveryMode,
                 int priority,
                 long timeToLive) throws JMSException
```

以遞送模式、優先順序及存活時間來傳送訊息到指定的佇列中。

**註：** 這個方法只能搭配未識別的 QueueSender 使用。

**參數：**

- queue - 這個訊息應該送往的佇列。
- message - 要傳送的訊息。
- deliveryMode - 要使用的遞送模式。
- priority - 這個訊息的優先順序。
- timeToLive - 訊息的存活時間（以毫秒爲單位）。

**擲出：**

- JMSException - 如果 JMS 因內部錯誤而無法傳送訊息的話。
- MessageFormatException - 如果指定無效訊息的話。
- InvalidDestinationException - 如果從屬站搭配無效佇列來使用這個方法的話。

### close *

```
public void close() throws JMSException
```

由於提供者可在 JVM 之外代表 QueueSender 來配置某些資源，因此，當不需要它們時，從屬站應該關閉它們。您終究不能依賴記憶體回收來收回這些資源，因爲記憶體回收的進行可能不夠快。

**擲出：** JMSException - 如果 JMS 因錯誤而無法關閉產生者的話。

**置換：** MQMessageProducer 類別中的 close。

# QueueSession

public interface **QueueSession**
extends **Session**

MQSeries 類別：**MQQueueSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQSession
           |
           +----com.ibm.mq.jms.MQQueueSession
```

QueueSession 提供建立 QueueReceivers、QueueSenders、QueueBrowsers 和 TemporaryQueues 的方法。

另請參閱：**Session**

下列方法繼承自 **MQSession**：
- close
- commit
- rollback
- recover

## 方法

### createQueue

```
public Queue createQueue(java.lang.String queueName)
                                                throws JMSException
```

建立一個 Queue，給定一個 Queue 名稱。這可讓您利用特定提供者專用的名稱來建立佇列。如 171 頁所說明，這個字串採用 URI 格式。

**註:** 相依於這種功能的從屬站不具可攜性。

**參數：** queueName - 這個佇列的名稱。

**傳回：** 含給定名稱的 Queue。

**擲出：** JMSException - 如果階段作業因 JMS 錯誤而無法建立佇列的話。

### createReceiver

```
public QueueReceiver createReceiver(Queue queue)
                                                throws JMSException
```

從指定的佇列中建立一個 QueueReceiver 來接收訊息。

**參數：** queue - 要存取的佇列。

**擲出：**

- JMSException - 如果階段作業因 JMS 錯誤而無法建立接收端的話。
- InvalidDestinationException - 如果指定的 Queue 無效的話。

### createReceiver

```
public QueueReceiver createReceiver(Queue queue,
                                    java.lang.String messageSelector)
                                    throws JMSException
```

從指定的佇列中建立一個 QueueReceiver 來接收訊息。

**參數：**

- queue - 要存取的佇列。
- messageSelector - 只遞送內容符合訊息選取元表示式的訊息。

**擲出：**

- JMSException - 如果階段作業因 JMS 錯誤而無法建立接收端的話。
- InvalidDestinationException - 如果指定的 Queue 無效的話。
- InvalidSelectorException - 如果訊息選取元無效的話。

### createSender

```
public QueueSender createSender(Queue queue)
                                        throws JMSException
```

建立一個 QueueSender 來傳送訊息到指定的佇列。

**參數：** queue - 要存取的佇列，如果這要成爲未識別的產生者的話，即爲空值。

**擲出：**

- JMSException - 如果階段作業因 JMS 錯誤而無法建立傳送端的話。
- InvalidDestinationException - 如果指定的 Queue 無效的話。

### createBrowser

```
public QueueBrowser createBrowser(Queue queue)
                                      throws JMSException
```

建立一個 QueueBrowser 來檢視指定佇列中的訊息。

**參數：** queue - 要存取的佇列。

**擲出：**

- JMSException - 如果階段作業因 JMS 錯誤而無法建立瀏覽器的話。
- InvalidDestinationException - 如果指定的 Queue 無效的話。

### createBrowser

```
public QueueBrowser createBrowser(Queue queue,
                                  java.lang.String messageSelector)
                                  throws JMSException
```

建立一個 QueueBrowser 來檢視指定佇列中的訊息。

**參數：**

- queue - 要存取的佇列。
- messageSelector - 只遞送內容符合訊息選取元表示式的訊息。

**擲出：**

- JMSException - 如果階段作業因 JMS 錯誤而無法建立瀏覽器的話。
- InvalidDestinationException - 如果指定的 Queue 無效的話。
- InvalidSelectorException - 如果訊息選取元無效的話。

### createTemporaryQueue

```
public TemporaryQueue createTemporaryQueue()
                                        throws JMSException
```

建立一個暫時佇列。除非提早刪除它，否則，它的存活時間即是 QueueConnection 的存活時間。

**傳回：** 暫時佇列。

**擲出：** JMSException - 如果階段作業因 JMS 錯誤而無法建立暫時佇列的話。

# Session

public interface **Session**
extends **java.lang.Runnable**
子介面：**QueueSession**、**TopicSession**、**XAQueueSession**、**XASession** 和 **XATopicSession**

MQSeries 類別：**MQSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQSession
```

JMS Session 是一個產生和消費訊息的單執行緒環境。

另請參閱：**QueueSession**、**TopicSession**、**XAQueueSession**、**XASession** 和 **XATopicSession**

## 欄位

### AUTO_ACKNOWLEDGE

```
public static final int AUTO_ACKNOWLEDGE
```

當使用這個認可模式時，階段作業會在接收呼叫順利傳回訊息或它呼叫來處理訊息的訊息接聽器順利傳回時，自動認可訊息。

### CLIENT_ACKNOWLEDGE

```
public static final int CLIENT_ACKNOWLEDGE
```

當使用這個認可模式時，從屬站會呼叫訊息的認可方法來認可訊息。

### DUPS_OK_ACKNOWLEDGE

```
public static final int DUPS_OK_ACKNOWLEDGE
```

這個認可模式會指示階段作業慢慢認可訊息的遞送。

## 方法

### createBytesMessage

```
public BytesMessage createBytesMessage()
                                throws JMSException
```

建立一個 BytesMessage。 BytesMessage 用來傳送含有未解譯之位元組串流的訊息。

**擲出：** JMSException - 如果 JMS 因內部錯誤而無法建立這個訊息的話。

## Session

### createMapMessage

```
public MapMessage createMapMessage() throws JMSException
```

建立一個 MapMessage。 MapMessage 用來傳送一組自行定義的名稱/值配對，其中名稱是 String，值是 Java 基本類型。

**擲出：** JMSException - 如果 JMS 因內部錯誤而無法建立這個訊息的話。

### createMessage

```
public Message createMessage() throws JMSException
```

建立一則訊息。 Message 介面是所有 JMS 訊息的根介面。它會保留所有標準訊息標題資訊。當訊息只含標題資訊即已足夠時，便可以傳送它。

**擲出：** JMSException - 如果 JMS 因內部錯誤而無法建立這個訊息的話。

### createObjectMessage

```
public ObjectMessage createObjectMessage()
                             throws JMSException
```

建立一個 ObjectMessage。 ObjectMessage 用來傳送含有可序列化之 Java 物件的訊息。

**擲出：** JMSException - 如果 JMS 因內部錯誤而無法建立這個訊息的話。

### createObjectMessage

```
public ObjectMessage createObjectMessage
                            (java.io.Serializable object)
                             throws JMSException
```

建立一個已起始設定的 ObjectMessage。 ObjectMessage 用來傳送含有可序列化之 Java 物件的訊息。

**參數：** object - 用來起始設定這個訊息的物件。

**擲出：** JMSException - 如果 JMS 因內部錯誤而無法建立這個訊息的話。

### createStreamMessage

```
public StreamMessage createStreamMessage()
                              throws JMSException
```

建立一個 StreamMessage。 StreamMessage 用來傳送 Java 基本類型的自行定義串流。

**擲出：** JMSException - 如果 JMS 因內部錯誤而無法建立這個訊息的話。

### createTextMessage

```
public TextMessage createTextMessage() throws JMSException
```

建立一個 TextMessage。 TextMessage 用來傳送含有 String 的訊息。

**擲出：** JMSException - 如果 JMS 因內部錯誤而無法建立這個訊息的話。

### createTextMessage

```
public TextMessage createTextMessage
                        (java.lang.String string)
                        throws JMSException
```

建立一個已起始設定的 TextMessage。 TextMessage 用來傳送含有 String 的訊息。

**參數：** string - 用來起始設定這個訊息的字串。

**擲出：** JMSException - 如果 JMS 因內部錯誤而無法建立這個訊息的話。

### getTransacted

```
public boolean getTransacted() throws JMSException
```

階段作業採用異動模式嗎？

**傳回：** true - 如果階段作業採用異動模式的話。

**擲出：** JMSException - 如果 JMS 因 JMS 提供者內部錯誤而無法傳回異動模式的話。

### commit

```
public void commit() throws JMSException
```

確定這項異動所進行的所有訊息，及釋出目前所保留的任何鎖定。

**擲出：**

- JMSException - 如果 JMS 實作因內部錯誤而無法確定異動的話。
- TransactionRolledBackException - 如果異動因確定期間的內部錯誤而被回復的話。

### rollback

```
public void rollback() throws JMSException
```

回復這項異動所進行的任何訊息，及釋出目前所保留的任何鎖定。

**擲出：** JMSException - 如果 JMS 實作因內部錯誤而無法回復異動的話。

## Session

### close

```
public void close() throws JMSException
```

由於提供者可在 JVM 之外代表 Session 來配置某些資源，因此，當不需要它們時，從屬站應該關閉它們。您終究不能依賴記憶體回收來收回這些資源，因爲記憶體回收的進行可能不夠快。

關閉異動的階段作業會回復任何進行中的異動。關閉階段作業會自動關閉它的訊息產生者和消費者，因而不需要個別關閉它們。

**擲出：** JMSException - 如果 JMS 實作因內部錯誤而無法關閉 Session 的話。

### recover

```
public void recover() throws JMSException
```

停止這個階段作業中的訊息遞送，及利用最舊而未認可的訊息來重新啓動訊息的傳送。

**擲出：** JMSException - 如果 JMS 實作因內部錯誤而無法停止訊息遞送及重新啓動訊息傳送的話。

### getMessageListener

```
public MessageListener getMessageListener()
                                  throws JMSException
```

傳回階段作業已獲辨認的訊息接聽器。

**傳回：** 這個階段作業所關聯的訊息接聽器。

**擲出：** JMSException - 如果 JMS 因 JMS 提供者內部錯誤而無法取得訊息接聽器的話。

**另請參閱：**

setMessageListener

### setMessageListener

```
public void setMessageListener(MessageListener listener)
                                           throws JMSException
```

設定階段作業已獲辨認的訊息接聽器。設好之後，便不能使用階段作業中任何其它訊息接收格式。不過，仍支援所有訊息傳送格式。

這是不由一般 JMS 從屬站使用的專家級機能。

**參數：** listener - 這個階段作業所關聯的訊息接聽器。

**擲出：** JMSException - 如果 JMS 因 JMS 提供者內部錯誤而無法設定訊息接聽器的話。

**另請參閱：**

getMessageListener、ServerSessionPool、ServerSession

### run

```
public void run()
```

這個方法只供應用程式伺服器使用。

**指定者：**

java.lang.Runnable 介面中的 run

**另請參閱：**

ServerSession

# StreamMessage

public interface **StreamMessage**
extends **Message**

MQSeries 類別：**JMSStreamMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSStreamMessage
```

StreamMessage 用來傳送 Java 基本類型的串流。

另請參閱：**BytesMessage**、**MapMessage**、**Message**、**ObjectMessage** 和 **TextMessage**

## 方法

### readBoolean

```
public boolean readBoolean() throws JMSException
```

從串流訊息中讀取一個 Boolean。

**傳回：** 讀取的 Boolean 值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。
- MessageEOFException - 如果收到訊息串流的結尾端的話。
- MessageFormatException - 如果這個類型轉換無效的話。
- MessageNotReadableException - 如果訊息採用唯寫模式的話。

### readByte

```
public byte readByte() throws JMSException
```

從串流訊息中讀取一個 Byte 值。

**傳回：** 串流訊息中作爲 8 位元之位元組的下個位元組。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。
- MessageEOFException - 如果收到訊息串流的結尾端的話。
- MessageFormatException - 如果這個類型轉換無效的話。
- MessageNotReadableException - 如果訊息採用唯寫模式的話。

### readShort

```
public short readShort() throws JMSException
```

從串流訊息中讀取一個 16 位元的數字。

**傳回：** 串流訊息中的一個 16 位元數字。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。
- MessageEOFException - 如果收到訊息串流的結尾端的話。
- MessageFormatException - 如果這個類型轉換無效的話。
- MessageNotReadableException - 如果訊息採用唯寫模式的話。

### readChar

```
public char readChar() throws JMSException
```

從串流訊息中讀取 Unicode 字元值。

**傳回：** 串流訊息中的一個 Unicode 字元。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。
- MessageEOFException - 如果收到訊息串流的結尾端的話。
- MessageFormatException - 如果這個類型轉換無效的話。
- MessageNotReadableException - 如果訊息採用唯寫模式的話。

### readInt

```
public int readInt() throws JMSException
```

從串流訊息中讀取一個 32 位元整數。

**傳回：** 串流訊息串的一個 32 位元整數值，當作 Int 來解讀。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。
- MessageEOFException - 如果收到訊息串流的結尾端的話。
- MessageFormatException - 如果這個類型轉換無效的話。
- MessageNotReadableException - 如果訊息採用唯寫模式的話。

### readLong

```
public long readLong() throws JMSException
```

從串流訊息中讀取一個 64 位元整數。

**傳回：** 串流訊息串的一個 64 位元整數值，當作 Long 來解讀。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。
- MessageEOFException - 如果收到訊息串流的結尾端的話。
- MessageFormatException - 如果這個類型轉換無效的話。
- MessageNotReadableException - 如果訊息採用唯寫模式的話。

### readFloat

```
public float readFloat() throws JMSException
```

從串流訊息中讀取一個 Float。

**傳回：** 串流訊息中的一個 Float。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。
- MessageEOFException - 如果收到訊息串流的結尾端的話。
- MessageFormatException - 如果這個類型轉換無效的話。
- MessageNotReadableException - 如果訊息採用唯寫模式的話。

### readDouble

```
public double readDouble() throws JMSException
```

從串流訊息中讀取一個 Double。

**傳回：** 串流訊息中的一個 Double。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。
- MessageEOFException - 如果收到訊息串流的結尾端的話。
- MessageFormatException - 如果這個類型轉換無效的話。
- MessageNotReadableException - 如果訊息採用唯寫模式的話。

### readString

```
public java.lang.String readString() throws JMSException
```

從串流訊息中讀取一個 String。

**傳回：** 串流訊息中的一個 Unicode 字串。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。
- MessageEOFException - 如果收到訊息串流的結尾端的話。
- MessageFormatException - 如果這個類型轉換無效的話。
- MessageNotReadableException - 如果訊息採用唯寫模式的話。

### readBytes

```
public int readBytes(byte[] value)
             throws JMSExceptioneam message.
```

將串流訊息中的一個位元組陣列欄位讀到指定的 byte[] 物件（讀取緩衝區）中。如果緩衝區大小小於或等於訊息欄位中的資料大小，應用程式必須進一步呼叫這個方法來擷取資料的其餘部份。呼叫好 byte[] 欄位值中的第一個 readBytes 呼叫之後，必須先讀取完整的欄位值，才能有效讀取下個欄位。試圖在完成這個動作之前讀取下個欄位，會擲出 MessageFormatException。

**參數：** value - 資料讀入其中的緩衝區。

**傳回：** 讀入緩衝區的位元數，如果已到達位元組欄位尾端而不再有資料時，則爲 -1。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。
- MessageEOFException - 如果收到訊息串流的結尾端的話。
- MessageFormatException - 如果這個類型轉換無效的話。
- MessageNotReadableException - 如果訊息採用唯寫模式的話。

### readObject

```
public java.lang.Object readObject() throws JMSException
```

從訊息串流中讀取一個 Java 物件。

**傳回：** 串流訊息中採物件格式的一個 Java 物件（比方說，如果將它設爲 int 的話，會傳回整數）。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。
- MessageEOFException - 如果收到訊息串流的結尾端的話。
- NotReadableException - 如果訊息採用唯寫模式的話。

### writeBoolean

```
public void writeBoolean(boolean value) throws JMSException
```

將一個 Boolean 寫入串流訊息中。

**參數：** value - 要寫入的 Boolean 值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法讀取訊息的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

## StreamMessage

### writeByte

```
public void writeByte(byte value) throws JMSException
```

將一個 Byte 寫到串流訊息中。

**參數：** value - 要寫入的 Byte 值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

### writeShort

```
public void writeShort(short value) throws JMSException
```

將一個 Short 寫入串流訊息。

**參數：** value - 要寫入的 Short。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

### writeChar

```
public void writeChar(char value) throws JMSException
```

將一個 Char 寫入串流訊息。

**參數：** value - 要寫入的 Char 值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

### writeInt

```
public void writeInt(int value) throws JMSException
```

將一個 Int 寫入串流訊息。

**參數：** value - 要寫入的 Int。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

### writeLong

```
public void writeLong(long value) throws JMSException
```

將一個 Long 寫入串流訊息。

**參數：** value - 要寫入的 Long。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

### writeFloat

```
public void writeFloat(float value) throws JMSException
```

將一個 Float 寫入串流訊息。

**參數：** value - 要寫入的 Float 值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

### writeDouble

```
public void writeDouble(double value) throws JMSException
```

將一個 Double 寫入串流訊息。

**參數：** value - 要寫入的 Double 值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

### writeString

```
public void writeString(java.lang.String value)
                                    throws JMSException
```

將一個 String 寫入串流訊息。

**參數：** value - 要寫入的 String 值。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

## StreamMessage

### writeBytes

```
public void writeBytes(byte[] value) throws JMSException
```

將位元組陣列寫入串流訊息。

**參數：** value - 要寫入的位元組陣列。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

### writeBytes

```
public void writeBytes(byte[] value,
                       int offset,
                       int length) throws JMSException
```

將位元組陣列的一部份寫入串流訊息中。

**參數：**

- value - 要寫入的位元組陣列值。
- offset - 位元組陣列內的起始偏移。
- length - 要使用的位元組數目。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

### writeObject

```
public void writeObject(java.lang.Object value)
                                    throws JMSException
```

將 Java 物件寫入串流訊息中。這個方法只適用於物件基本類型（如 Integer、Double、Long）、String 及位元組陣列。

**參數：** value - 要寫入的 Java 物件。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法寫入訊息的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。
- MessageFormatException - 如果物件無效的話。

### reset

```
public void reset() throws JMSException
```

將訊息置於唯讀模式，並將串流重新定位到起始位置。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法重設訊息的話。
- MessageFormatException - 如果訊息格式無效的話。

# TemporaryQueue

public interface **TemporaryQueue**
extends **Queue**

MQSeries 類別：**MQTemporaryQueue**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQQueue
                   |
                   +----com.ibm.mq.jms.MQTemporaryQueue
```

TemporaryQueue 是專爲 QueueConnection 的持續期間而建立的唯一 Queue 物件。

## 方法

### delete

```
public void delete() throws JMSException
```

刪除這個暫時佇列。如果仍有現存的傳送端或接收端在使用它，會擲出 JMSException。

**擲出：** JMSException - 如果 JMS 實作因內部錯誤而無法刪除 TemporaryQueue 的話。

# TemporaryTopic

public interface **TemporaryTopic**
extends **Topic**

MQSeries 類別：**MQTemporaryTopic**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQTopic
                   |
                   +----com.ibm.mq.jms.MQTemporaryTopic
```

TemporaryTopic 是專爲 TopicConnection 的持續期間而建立的唯一主題物件，只有這個連線的消費者可以使用它。

## MQSeries 建構子

### MQTemporaryTopic

```
MQTemporaryTopic() throws JMSException
```

## 方法

### delete

```
public void delete() throws JMSException
```

刪除這個暫時主題。如果仍有現存的發佈者或訂閱者在使用它，會擲出 JMSException。

**擲出：** JMSException - 如果 JMS 實作因內部錯誤而無法刪除 TemporaryTopic 的話。

# TextMessage

public interface **TextMessage**
extends **Message**

MQSeries 類別：**JMSTextMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSTextMessage
```

TextMessage 用來傳送含有 java.lang.String 的訊息。它繼承 Message，且加入了文字訊息主體。

另請參閱：**BytesMessage**、**MapMessage**、**Message**、**ObjectMessage** 和 **StreamMessage**

## 方法

### setText

```
public void setText(java.lang.String string)
                                        throws JMSException
```

設定含有這個訊息資料的字串。

**參數：** string - 含有訊息資料的 String。

**擲出：**

- JMSException - 如果 JMS 因內部 JMS 錯誤而無法設定文字的話。
- MessageNotWriteableException - 如果訊息採用唯讀模式的話。

### getText

```
public java.lang.String getText() throws JMSException
```

取得含有這個訊息資料的字串。預設值是空值。

**傳回：** 含有訊息資料的 String。

**擲出：** JMSException - 如果 JMS 因內部 JMS 錯誤而無法取得文字的話。

# Topic

public interface **Topic**
extends **Destination**
子介面：**TemporaryTopic**

MQSeries 類別：**MQTopic**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQTopic
```

Topic 物件封裝特定提供者專用的主題名稱。它是從屬站將主題身份指定給 JMS 方法的方式。

另請參閱：**Destination**

## MQSeries 建構子

### MQTopic

```
public MQTopic()
public MQTopic(string URItopic)
```

請參閱 **TopicSession.createTopic**。

## 方法

### getTopicName

```
public java.lang.String getTopicName() throws JMSException
```

以 URI 格式取得這個主題的名稱。 （URI 格式，請參閱第180頁的『在執行時期建立主題』。）

**註:** 相依於名稱的從屬站不具可攜性。

**傳回：** 主題名稱。

**擲出：** JMSException - 如果 Topic 的 JMS 實作因內部錯誤而無法傳回主題名稱的話。

### toString

```
public String toString()
```

傳回列印優美的 Topic 名稱版本。

**傳回：** 這個 Topic 的提供者特定身份值。

**置換：** Object 類別中的 toString。

### getReference *

```
public Reference getReference()
```

建立這個主題的參照。

**傳回：** 這個物件的參照。

**擲出：** NamingException.

### setBaseTopicName *

```
public void setBaseTopicName(String x)
```

設定基礎 MQSeries 主題名稱的方法。

### getBaseTopicName *

```
public String getBaseTopicName()
```

取得基礎 MQSeries 主題名稱的方法。

### setBrokerDurSubQueue *

```
public void setBrokerDurSubQueue(String x) throws JMSException
```

設定 brokerDurSubQueue 屬性的方法。

**參數：** brokerDurSubQueue - 要使用的可延續訂閱佇列的名稱。

### getBrokerDurSubQueue *

```
public String getBrokerDurSubQueue()
```

取得 brokerDurSubQueue 屬性的方法。

**傳回：** 要使用的可延續訂閱佇列 (brokerDurSubQueue) 的名稱。

### setBrokerCCDurSubQueue *

```
public void setBrokerCCDurSubQueue(String x) throws JMSException
```

設定 brokerCCDurSubQueue 屬性的方法。

**參數：** brokerCCDurSubQueue - 要用在 ConnectionConsumer 的可延續訂閱佇列的名稱。

### getBrokerCCDurSubQueue *

```
public String getBrokerCCDurSubQueue()
```

取得 brokerCCDurSubQueue 屬性的方法。

**傳回：** 要用在 ConnectionConsumer 的可延續訂閱佇列 (brokerCCDurSubQueue) 的名稱。

## TopicConnection

public interface **TopicConnection**
extends **Connection**
子介面：**XATopicConnection**

MQSeries 類別：**MQTopicConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection
           |
           +----com.ibm.mq.jms.MQTopicConnection
```

TopicConnection 是通往 JMS 發佈/訂閱提供者的作用中連線。

另請參閱：**Connection**、**TopicConnectionFactory** 和 **XATopicConnection**

### 方法

#### createTopicSession

```
public TopicSession createTopicSession(boolean transacted,
                                       int acknowledgeMode)
                                       throws JMSException
```

建立一個 TopicSession。

**參數：**

- transacted - 如果爲 true，會進行階段作業異動。
- acknowledgeMode - 下列項目之一：
  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

  指出消費者或從屬站是否會認可它收到的任何訊息。如果進行階段作業異動的話，會忽略這個參數。

**傳回：** 新建立的主題階段作業。

**擲出：** JMSException - 如果 JMS Connection 因內部錯誤或沒有特定異動和認可模式之支援而無法建立階段作業的話。

#### createConnectionConsumer

```
public ConnectionConsumer createConnectionConsumer
                            (Topic topic,
                             java.lang.String messageSelector,
                             ServerSessionPool sessionPool,
                             int maxMessages)
                                      throws JMSException
```

建立這個連線的連線消費者。這是不由一般 JMS 從屬站使用的專家級機能。

**參數：**

- topic - 要存取的主題。
- messageSelector - 只遞送內容符合訊息選取元表示式的訊息。
- sessionPool - 這個連線消費者所關聯的伺服器階段作業儲存池。
- maxMessages - 可同時指派給伺服器階段作業的訊息數目上限。

**傳回：** 連線消費者。

**擲出：**

- JMSException - 如果 JMS Connection 因內部錯誤或 sessionPool 的引數無效而無法建立連線消費者的話。
- InvalidSelectorException - 如果訊息選取元無效的話。

**另請參閱：**

ConnectionConsumer

### createDurableConnectionConsumer

```
public ConnectionConsumer createDurableConnectionConsumer
                              (Topic topic,
                               java.lang.String subscriptionName
                               java.lang.String messageSelector,
                               ServerSessionPool sessionPool,
                               int maxMessages)
                                        throws JMSException
```

建立這個連線的可延續連線消費者。這是不由一般 JMS 從屬站使用的專家級機能。

**參數：**

- topic - 要存取的主題。
- subscriptionName - 可延續的訂閱名稱。
- messageSelector - 只遞送內容符合訊息選取元表示式的訊息。
- sessionPool - 這個可延續連線消費者所關聯的伺服器階段作業儲存池。
- maxMessages - 可同時指派給伺服器階段作業的訊息數目上限。

**傳回：** 可延續的連線消費者。

**擲出：**

- JMSException - 如果 JMS Connection 因內部錯誤或 sessionPool 和 messageSelector 的引數無效而無法建立連線消費者的話。
- InvalidSelectorException - 如果訊息選取元無效的話。

**另請參閱：**

ConnectionConsumer

# TopicConnectionFactory

public interface **TopicConnectionFactory**
extends **ConnectionFactory**
子介面：**XATopicConnectionFactory**

MQSeries 類別：**MQTopicConnectionFactory**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory
           |
           +----com.ibm.mq.jms.MQTopicConnectionFactory
```

從屬站利用 TopicConnectionFactory，以 JMS 發佈/訂閱提供者來建立 TopicConnection。

另請參閱：**ConnectionFactory** 和 **XATopicConnectionFactory**

## MQSeries 建構子

### MQTopicConnectionFactory

```
public MQTopicConnectionFactory()
```

## 方法

### createTopicConnection

```
public TopicConnection createTopicConnection()
                                        throws JMSException
```

以預設使用者身份建立一個主題連線。這個連線是採用停止模式建立。在明確呼叫 Connection.start 方法之前，不會遞送任何訊息。

**傳回：** 新建立的主題連線。

**擲出：**

- JMSException - 如果 JMS 提供者因內部錯誤而無法建立主題連線的話。
- JMSSecurityException - 如果從屬站鑑別因無效使用者名稱或密碼而失效的話。

### createTopicConnection

```
public TopicConnection createTopicConnection
                                        (java.lang.String userName,
                                        java.lang.String password)
                                        throws JMSException
```

以指定的使用者身份建立一個主題連線。這個連線是採用停止模式建立。在明確呼叫 Connection.start 方法之前，不會遞送任何訊息。

**註:** 這個方法只對 IBM_JMS_TP_CLIENT_MQ_TCPIP 傳輸類型有效。請參閱 ConnectionFactory。

**參數：**

- userName - 呼叫端的使用者名稱。
- password - 呼叫端的密碼。

**傳回：** 新建立的主題連線。

**擲出：**

- JMSException - 如果 JMS 提供者因內部錯誤而無法建立主題連線的話。
- JMSSecurityException - 如果從屬站鑑別因無效使用者名稱或密碼而失效的話。

### setBrokerControlQueue *

```
public void setBrokerControlQueue(String x) throws JMSException
```

設定 brokerControlQueue 屬性的方法。

**參數：** brokerControlQueue - 分配管理程式控制佇列的名稱。

### getBrokerControlQueue *

```
public String getBrokerControlQueue()
```

取得 brokerControlQueue 屬性的方法。

**傳回：** 分配管理程式的控制佇列名稱

### setBrokerQueueManager *

```
public void setBrokerQueueManager(String x) throws JMSException
```

設定 brokerQueueManager 屬性的方法。

**參數：** brokerQueueManager - 分配管理程式之佇列管理程式的名稱。

### getBrokerQueueManager *

```
public String getBrokerQueueManager()
```

取得 brokerQueueManager 屬性的方法。

**傳回：** 分配管理程式的佇列管理程式名稱。

### setBrokerPubQueue *

```
public void setBrokerPubQueue(String x) throws JMSException
```

設定 brokerPubQueue 屬性的方法。

**參數：** brokerPubQueue - 分配管理程式發佈佇列的名稱。

### getBrokerPubQueue *

```
public String getBrokerPubQueue()
```

取得 brokerPubQueue 屬性的方法。

**傳回：** 分配管理程式的發佈佇列名稱。

### setBrokerSubQueue *

```
public void setBrokerSubQueue(String x) throws JMSException
```

設定 brokerSubQueue 屬性的方法。

**參數：** brokerSubQueue - 要使用的不可延續訂閱佇列的名稱。

### getBrokerSubQueue *

```
public String getBrokerSubQueue()
```

取得 brokerSubQueue 屬性的方法。

**傳回：** 要使用的不可延續訂閱佇列的名稱。

### setBrokerCCSubQueue *

```
public void setBrokerCCSubQueue(String x) throws JMSException
```

設定 brokerCCSubQueue 屬性的方法。

**參數：** brokerSubQueue - 要用在 ConnectionConsumer 的不可延續訂閱佇列的名稱。

### getBrokerCCSubQueue *

```
public String getBrokerCCSubQueue()
```

取得 brokerCCSubQueue 屬性的方法。

**傳回：** 要用在 ConnectionConsumer 的不可延續訂閱佇列的名稱。

### setBrokerVersion *

```
public void setBrokerVersion(int x) throws JMSException
```

設定 brokerVersion 屬性的方法。

**參數：** brokerVersion - 分配管理程式的版本號碼。

### getBrokerVersion *

```
public int getBrokerVersion()
```

取得 brokerVersion 屬性的方法。

**傳回：** 分配管理程式的版本號碼。

### getReference *

```
public Reference getReference()
```

傳回這個主題連線 Factory 的參照。

**傳回：** 這個主題連線 Factory 的參照。

**擲出：** NamingException.

# TopicPublisher

public interface **TopicPublisher**
extends **MessageProducer**

MQSeries 類別：**MQTopicPublisher**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageProducer
           |
           +----com.ibm.mq.jms.MQTopicPublisher
```

從屬站利用 TopicPublisher 來發佈主題訊息。 TopicPublisher 是 JMS 訊息消費者的發佈/訂閱差異。

## 方法

### getTopic

```
public Topic getTopic() throws JMSException
```

取得這個發佈者所關聯的主題。

**傳回：** 這個發佈者的主題

**擲出：** JMSException - 如果 JMS 因內部錯誤而無法取得這個主題發佈者的主題的話。

### publish

```
public void publish(Message message) throws JMSException
```

利用主題的預設遞送模式、存活時間及優先順序來發佈訊息到主題中。

**參數：** message - 要發佈的訊息

**擲出：**

- JMSException - 如果 JMS 因內部錯誤而無法發佈訊息的話。
- MessageFormatException - 如果指定無效訊息的話。
- InvalidDestinationException - 如果從屬站搭配主題無效的「主題發佈者」來使用這個方法的話。

### publish

```
public void publish(Message message,
                    int deliveryMode,
                    int priority,
                    long timeToLive) throws JMSException
```

就指定遞送模式、優先順序及存活時間的主題發佈訊息到主題中。

**參數：**

- message - 要發佈的訊息。
- deliveryMode - 要使用的遞送模式。
- priority - 這個訊息的優先順序。
- timeToLive - 訊息的存活時間（以毫秒爲單位）。

**擲出：**

- JMSException - 如果 JMS 因內部錯誤而無法發佈訊息的話。
- MessageFormatException - 如果指定無效訊息的話。
- InvalidDestinationException - 如果從屬站搭配主題主效的「主題發佈者」來使用這個方法的話。

### publish

```
public void publish(Topic topic,
                    Message message) throws JMSException
```

就未識別的訊息產生者來發佈訊息到主題中。請使用主題的預設遞送模式、存活時間及優先順序。

**參數：**

- topic - 這個訊息的發佈主題。
- message - 要傳送的訊息。

**擲出：**

- JMSException - 如果 JMS 因內部錯誤而無法發佈訊息的話。
- MessageFormatException - 如果指定無效訊息的話。
- InvalidDestinationException - 如果從屬站搭配無效主題使用這個方法的話。

### publish

```
public void publish(Topic topic,
                    Message message,
                    int deliveryMode,
                    int priority,
                    long timeToLive) throws JMSException
```

指定遞送模式、優先順序及存活時間，就未識別的訊息產生者而發佈訊息到主題中。

**參數：**

- topic - 這個訊息的發佈主題。
- message - 要傳送的訊息。
- deliveryMode - 要使用的遞送模式。
- priority - 這個訊息的優先順序。
- timeToLive - 訊息的存活時間（以毫秒爲單位）。

**擲出：**

- JMSException - 如果 JMS 因內部錯誤而無法發佈訊息的話。
- MessageFormatException - 如果指定無效訊息的話。
- InvalidDestinationException - 如果從屬站搭配無效主題使用這個方法的話。

### close *

```
public void close() throws JMSException
```

由於提供者可在 JVM 之外代表 TopicRequestor 來配置某些資源，因此，當不需要它們時，從屬站應該關閉它們。您終究不能依賴記憶體回收來收回這些資源，因爲記憶體回收的進行可能不夠快。

**擲出：** JMSException - 如果 JMS 因錯誤而無法關閉產生者的話。

**置換：** MQMessageProducer 類別中的 close。

# TopicRequestor

public class **TopicRequestor**
extends **java.lang.Object**

```
java.lang.Object
    |
    +----javax.jms.TopicRequestor
```

JMS 提供這個 TopicRequestor 類別來協助建立服務要求。

TopicRequestor 建構子會得到非異動 TopicSession 和目的地 Topic。它會建立一個回應的 TemporaryTopic，並提供 request() 方法來傳送要求訊息及等待其回應。使用者可以自由建立更複雜的版本。

## 建構子

### TopicRequestor

```
public TopicRequestor(TopicSession session,
                      Topic topic) throws JMSException
```

TopicRequestor 類別的建構子。這項實作會假設階段作業參數不是異動式的，不論 AUTO_ACKNOWLEDGE 或 DUPS_OK_ACKNOWLEDGE 都一樣。

**參數：**

- session - 主題所屬的主題階段作業。
- topic - 執行要求/回覆呼叫的主題。

**擲出：** JMSException - 如果發生 JMS 錯誤的話。

## 方法

### request

```
public Message request(Message message) throws JMSException
```

傳送要求及等待回覆。

**參數：** message - 要傳送的訊息。

**傳回：** 回覆訊息。

**擲出：** JMSException - 如果發生 JMS 錯誤的話。

### close

```
public void close() throws JMSException
```

由於提供者可在 JVM 之外代表 TopicRequestor 來配置某些資源，因此，當不需要它們時，從屬站應該關閉它們。您終究不能依賴記憶體回收來收回這些資源，因爲記憶體回收的進行可能不夠快。

**註:** 這個方法會關閉傳遞給 TopicRequestor 建構子的 Session 物件。

**擲出：** JMSException - 如果發生 JMS 錯誤的話。

# TopicSession

public interface **TopicSession**
extends **Session**

MQSeries 類別：**MQTopicSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQSession
           |
           +----com.ibm.mq.jms.MQTopicSession
```

TopicSession 提供建立 TopicPublishers、TopicSubscribers 和 TemporaryTopics 的方法。

另請參閱：**Session**

## MQSeries 建構子

### MQTopicSession

```
public MQTopicSession(boolean transacted,
                      int acknowledgeMode) throws JMSException
```

請參閱 **TopicConnection.createTopicSession**。

## 方法

### createTopic

```
public Topic createTopic(java.lang.String topicName)
                                          throws JMSException
```

在得到 URI 格式的 Topic 名稱之後，建立一個 Topic。（URI 格式，請參閱第180頁的『在執行時期建立主題』。）這可讓您利用特定提供者專用的名稱來建立主題。

**註:** 相依於這種功能的從屬站不具可攜性。

**參數：** topicName - 這個主題的名稱。

**傳回：** 含給定名稱的 Topic。

**擲出：** JMSException - 如果階段作業因 JMS 錯誤而無法建立主題的話。

### createSubscriber

```
public TopicSubscriber createSubscriber(Topic topic)
                                      throws JMSException
```

建立指定主題的不可延續 Subscriber。

**參數：** topic - 要訂閱的主題

**擲出：**

- JMSException - 如果階段作業因 JMS 錯誤而無法建立訂閱者的話。
- InvalidDestinationException - 如果指定的 Topic 無效的話。

### createSubscriber

```
public TopicSubscriber createSubscriber
                           (Topic topic,
                            java.lang.String messageSelector,
                            boolean noLocal) throws JMSException
```

建立指定主題的不可延續 Subscriber。

**參數：**

- topic - 要訂閱的主題。
- messageSelector - 只遞送內容符合訊息選取元表示式的訊息。這個值可能是空值。
- noLocal - 如果設定的話，會禁止遞送它本身的連線所發佈的訊息。

**擲出：**

- JMSException - 如果階段作業因 JMS 錯誤或選取元無效而無法建立訂閱者的話。
- InvalidDestinationException - 如果指定的 Topic 無效的話。
- InvalidSelectorException - 如果訊息選取元無效的話。

### createDurableSubscriber

```
public TopicSubscriber createDurableSubscriber
                                  (Topic topic,
                                   java.lang.String name) throws JMSException
```

建立指定主題的可延續 Subscriber。從屬站可以採相同名稱及新主題和/或訊息選取元建立「可延續訂閱者」來變更現有的可延續訂閱。

**參數：**

- topic - 要訂閱的主題。
- name - 用來識別這項訂閱的名稱。

**擲出：**

- JMSException - 如果階段作業因 JMS 錯誤而無法建立訂閱者的話。
- InvalidDestinationException - 如果指定的 Topic 無效的話。

請參閱 **TopicSession.unsubscribe**

### createDurableSubscriber

```
public TopicSubscriber createDurableSubscriber
                                  (Topic topic,
                                   java.lang.String name,
                                   java.lang.String messageSelector,
                                   boolean noLocal)  throws JMSException
```

建立指定主題的可延續 Subscriber。

**參數：**

- topic - 要訂閱的主題。
- name - 用來識別這項訂閱的名稱。
- messageSelector - 只遞送內容符合訊息選取元表示式的訊息。 這個值可能是空值。
- noLocal - 如果設定的話，會禁止遞送它本身的連線所發佈的訊息。

**擲出：**

- JMSException - 如果階段作業因 JMS 錯誤或選取元無效而無法建立訂閱者的話。
- InvalidDestinationException - 如果指定的 Topic 無效的話。
- InvalidSelectorException - 如果訊息選取元無效的話。

### createPublisher

```
public TopicPublisher createPublisher(Topic topic)
                                      throws JMSException
```

建立指定主題的 Publisher。

**參數：** topic - 要發佈的主題，如果這是未識別的產生者，則爲空值。

**擲出：**

- JMSException - 如果階段作業因 JMS 錯誤而無法建立發佈者的話。
- InvalidDestinationException - 如果指定的 Topic 無效的話。

### createTemporaryTopic

```
public TemporaryTopic createTemporaryTopic()
                                       throws JMSException
```

建立暫時主題。除非提早刪除它，否則，它的存活時間即是 TopicConnection 的存活時間。

**傳回：** 暫時主題。

**擲出：** JMSException - 如果階段作業因 JMS 錯誤而無法建立暫時主題的話。

### unsubscribe

```
public void unsubscribe(java.lang.String name)
                                        throws JMSException
```

取消訂閱從屬站所建立的可延續訂閱。

**註:** 當有作用中的訂閱存在時，請勿使用這個方法。您必須先 close() 您的訂閱者。

**參數：** name - 用來識別這項訂閱的名稱。

**擲出：**

- JMSException - 如果 JMS 因 JMS 錯誤而無法取消訂閱可延續訂閱的話。
- InvalidDestinationException - 如果指定的 Topic 無效的話。

# TopicSubscriber

public interface **TopicSubscriber**
extends **MessageConsumer**

MQSeries 類別：**MQTopicSubscriber**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageConsumer
           |
           +----com.ibm.mq.jms.MQTopicSubscriber
```

從屬站利用 TopicSubscriber 來接收已就某主題而發佈的訊息。 TopicSubscriber 是 JMS 訊息消費者的發佈/訂閱差異。

另請參閱：**MessageConsumer** 和 **TopicSession.createSubscriber**

MQTopicSubscriber 繼承 MQMessageConsumer 的下列方法：

close
getMessageListener
receive
receiveNoWait
setMessageListener

## 方法

### getTopic

```
public Topic getTopic() throws JMSException
```

取得這個訂閱者所關聯的主題。

**傳回：** 這個訂閱者的主題。

**擲出：** JMSException - 如果 JMS 因內部錯誤而無法取得這個主題訂閱者的主題的話。

### getNoLocal

```
public boolean getNoLocal() throws JMSException
```

取得這個 TopicSubscriber 的 NoLocal 屬性。這個屬性的預設值是 false。

**傳回：** 如果禁止本端發佈的訊息的話，即設爲 true。

**擲出：** JMSException - 如果 JMS 因內部錯誤而無法取得這個主題訂閱者的 NoLocal 屬性的話。

## XAConnection

public interface **XAConnection**
子介面：**XAQueueConnection** 和 **XATopicConnection**

MQSeries 類別：**MQXAConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXAConnection
```

XAConnection 提供 XASession 來延伸連線功能。請參閱第351頁的『附錄E. 使用 WebSphere 的 JMS JTA/XA 介面』，以取得 MQ JMS 如何利用 XA 類別的詳細資料。

另請參閱：**XAQueueConnection** 和 **XATopicConnection**

## XAConnectionFactory

public interface **XAConnectionFactory**
子介面：**XAQueueConnectionFactory** 和 **XATopicConnectionFactory**

MQSeries 類別：**MQXAConnectionFactory**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXAConnectionFactory
```

有些應用程式伺服器支援將具有 JTS 能力的資源使用分組到分散式異動中。如果要將 JMS 異動併入 JTS 異動中，應用程式伺服器必須有能夠感知 JTS 的 JMS 提供者。JMS 提供者會利用應用程式伺服器用以建立 XASession 的 JMS XAConnectionFactory 來陳列出它的 JTS。如同 ConnectionFactory，XAConnectionFactory 也是 JMS 管理的物件。應用程式伺服器應該利用 JNDI 來找到它們。

請參閱第351頁的『附錄E. 使用 WebSphere 的 JMS JTA/XA 介面』，以取得 MQ JMS 如何利用 XA 類別的詳細資料。

另請參閱：**XAQueueConnectionFactory** 和 **XATopicConnectionFactory**

## XAQueueConnection

public interface **XAQueueConnection**
extends **QueueConnection** and **XAConnection**

MQSeries 類別：**MQXAQueueConnection**

```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQConnection
            |
           +----com.ibm.mq.jms.MQQueueConnection
                    |
                   +----com.ibm.mq.jms.MQXAQueueConnection
```

XAQueueConnection 提供和 QueueConnection 相同的建立選項。唯一的差異是，依定義，會進行 XAConnection 異動。請參閱第351頁的『附錄E. 使用 WebSphere 的 JMS JTA/XA 介面』，以取得 MQ JMS 如何利用 XA 類別的詳細資料。

另請參閱：**XAConnection** 和 **QueueConnection**

### 方法

#### createXAQueueSession

```
public XAQueueSession createXAQueueSession()
```

建立一個 XAQueueSession。

**擲出：** JMSException - 如果 JMS Connection 因內部錯誤而無法建立 XA 佇列階段作業的話。

#### createQueueSession

```
public QueueSession createQueueSession(boolean transacted,
                                       int acknowledgeMode)
                                                throws JMSException
```

建立一個 QueueSession。

**參數：**

- transacted - 如果爲 true，會進行階段作業異動。
- acknowledgeMode - 指出消費者或從屬站是否會認可它收到的任何訊息。可能的值如下：

  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

  如果進行階段作業異動的話，會忽略這個參數。

**傳回：** 新建立的佇列階段作業（請注意，這不是 XA 佇列階段作業）。

**擲出：** JMSException - 如果 JMS Connection 因內部錯誤而無法建立佇列階段作業的話。

# XAQueueConnectionFactory

public interface **XAQueueConnectionFactory**
extends **QueueConnectionFactory** and **XAConnectionFactory**

MQSeries 類別：**MQXAQueueConnectionFactory**

```
java.lang.Object
    |
   +----com.ibm.mq.jms.MQConnectionFactory
            |
           +----com.ibm.mq.jms.MQQueueConnectionFactory
                    |
                   +----com.ibm.mq.jms.MQXAQueueConnectionFactory
```

XAQueueConnectionFactory 提供和 QueueConnectionFactory 相同的建立選項。請參閱第 351頁的『附錄E. 使用 WebSphere 的 JMS JTA/XA 介面』，以取得 MQ JMS 如何利用 XA 類別的詳細資料。

另請參閱：**QueueConnectionFactory** 和 **XAConnectionFactory**

## 方法

### createXAQueueConnection

```
public XAQueueConnection createXAQueueConnection()
                                              throws JMSException
```

利用預設使用者身份建立一個 XAQueueConnection。這個連線是採用停止模式建立。在明確呼叫 Connection.start 之前，不會遞送任何訊息。

**傳回：** 新建立的 XA 佇列連線。

**擲出：**

- JMSException - 如果 JMS 提供者因內部錯誤而無法建立 XA 佇列連線的話。
- JMSSecurityException - 如果從屬站鑑別因無效使用者名稱或密碼而失效的話。

### createXAQueueConnection

```
public XAQueueConnection createXAQueueConnection
                                    (java.lang.String userName,
                                     java.lang.String password)
                                                    throws JMSException
```

利用特定的使用者身份建立一個 XA 佇列連線。這個連線是採用停止模式建立。在明確呼叫 Connection.start 之前，不會遞送任何訊息。

**參數：**

- userName - 呼叫端的使用者名稱。
- password - 呼叫端的密碼。

**傳回：** 新建立的 XA 佇列連線。

**擲出：**

- JMSException - 如果 JMS 提供者因內部錯誤而無法建立 XA 佇列連線的話。
- JMSSecurityException - 如果從屬站鑑別因無效使用者名稱或密碼而失效的話。

# XAQueueSession

public interface **XAQueueSession**
extends **XASession**

MQSeries 類別：**MQXAQueueSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXASession
           |
           +----com.ibm.mq.jms.MQXAQueueSession
```

XAQueueSession 提供一個可用來建立 QueueReceivers、QueueSenders 和 QueueBrowsers 的正規 QueueSession。請參閱第351頁的『附錄E. 使用 WebSphere 的 JMS JTA/XA 介面』，以取得 MQ JMS 如何利用 XA 類別的詳細資料。

QueueSession 所對應的 XAResource 可藉由呼叫從 XASession 繼承來的 getXAResource 方法而取得。

另請參閱：**XASession**

## 方法

### getQueueSession

```
public QueueSession  getQueueSession()
                                 throws JMSException
```

取得這個 XAQueueSession 所關聯的佇列階段作業。

**傳回：** 佇列階段作業物件。

**擲出：** JMSException - 如果發生 JMS 錯誤的話。

# XASession

public interface **XASession**
extends **Session**
子介面：**XAQueueSession** 和 **XATopicSession**

MQSeries 類別：**MQXASession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXASession
```

XASession 新增 JMS 提供者的 JTA 支援存取來延伸 Session 的功能。這項支援採用 javax.transaction.xa.XAResource 物件的形式。這個物件的功能非常類似於標準 X/Open XA 資源介面所定義的功能。

應用程式伺服器會取得 XASession 的 XAResource 來控制 XASession 的異動指派。它利用 XAResource 來指定階段作業給異動，準備及確定異動工作等。

XAResource 提供非常複雜的功能，如多重異動的交錯工作及復原進行中的異動清單。

能夠感應 JTA 的 JMS 提供者必須完整實作這項功能。爲了做到這一點，JMS 提供者可以利用支援 XA 的資料庫服務，也可以從頭開始實作這項功能。

應用程式伺服器的從屬站會得到類似一般的 JMS 階段作業。應用程式伺服器會在背後控制基礎 XASession 的異動管理。

請參閱第351頁的『附錄E. 使用 WebSphere 的 JMS JTA/XA 介面』，以取得 MQ JMS 如何利用 XA 類別的詳細資料。

另請參閱：**XAQueueSession** 和 **XATopicSession**

## 方法

### getXAResource

```
public javax.transaction.xa.XAResource getXAResource()
```

傳回 XA 資源給呼叫端。

**傳回：** XA 資源給呼叫端。

### getTransacted

```
public boolean getTransacted()
                        throws JMSException
```

永遠傳回 true。

**指定者：**
Session 介面中的 getTransacted。

**傳回：** true - 如果階段作業採用異動模式的話。

**擲出：** JMSException - 如果 JMS 因 JMS 提供者內部錯誤而無法傳回異動模式的話。

### commit

```
public void commit()
            throws JMSException
```

不應爲了 XASession 物件而呼叫這個方法。如果呼叫的話，它會擲出 TransactionInProgressException。

**指定者：**

Session 介面中的 commit。

**擲出：** TransactionInProgressException - 如果就 XASession 而呼叫這個方法的話。

### rollback

```
public void rollback()
              throws JMSException
```

不應爲了 XASession 物件而呼叫這個方法。如果呼叫的話，它會擲出 TransactionInProgressException。

**指定者：**

Session 介面中的 rollback。

**擲出：** TransactionInProgressException - 如果就 XASession 而呼叫這個方法的話。

# XATopicConnection

public interface **XATopicConnection**
extends **TopicConnection** and **XAConnection**

MQSeries 類別：**MQXATopicConnection**

```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQConnection
            |
            +----com.ibm.mq.jms.MQTopicConnection
                    |
                    +----com.ibm.mq.jms.MQXATopicConnection
```

XATopicConnection 提供和 TopicConnection 相同的建立選項。唯一的差異是，依定義，會進行 XAConnection 異動。請參閱第351頁的『附錄E. 使用 WebSphere 的 JMS JTA/XA 介面』，以取得 MQ JMS 如何利用 XA 類別的詳細資料。

另請參閱：**TopicConnection** 和 **XAConnection**

## 方法

### createXATopicSession

```
public XATopicSession createXATopicSession()
                                      throws JMSException
```

建立一個 XATopicSession。

**擲出：** JMSException - 如果 JMS Connection 因內部錯誤而無法建立 XA 主題階段作業的話。

### createTopicSession

```
public TopicSession createTopicSession(boolean transacted,
                                       int acknowledgeMode)
                                throws JMSException
```

建立一個 TopicSession。

**指定者：**

TopicConnection 介面中的 createTopicSession。

**參數：**

- transacted - 如果爲 true，會進行階段作業異動。
- acknowledgeMode - 下列項目之一：
  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

  指出消費者或從屬站是否會認可它收到的任何訊息。如果進行階段作業異動的話，會忽略這個參數。

**傳回：** 新建立的主題階段作業（請注意，這不是 XA 主題階段作業）。

**擲出：** JMSException - 如果 JMS Connection 因內部錯誤而無法建立主題階段作業的話。

# XATopicConnectionFactory

public interface **XATopicConnectionFactory**
extends **TopicConnectionFactory** and **XAConnectionFactory**

MQSeries 類別：**MQXATopicConnectionFactory**

```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQConnectionFactory
            |
            +----com.ibm.mq.jms.MQTopicConnectionFactory
                    |
                    +----com.ibm.mq.jms.MQXATopicConnectionFactory
```

XATopicConnectionFactory 提供和 TopicConnectionFactory 相同的建立選項。請參閱第 351頁的『附錄E. 使用 WebSphere 的 JMS JTA/XA 介面』，以取得 MQ JMS 如何利用 XA 類別的詳細資料。

另請參閱：**TopicConnectionFactory** 和 **XAConnectionFactory**

## 方法

### createXATopicConnection

```
public XATopicConnection createXATopicConnection()
                                        throws JMSException
```

利用預設使用者身份建立一個 XA 主題連線。這個連線是採用停止模式建立。在明確呼叫 Connection.start 之前，不會遞送任何訊息。

**傳回：** 新建立的 XA 主題連線。

**擲出：**

- JMSException - 如果 JMS 提供者因內部錯誤而無法建立 XA 主題連線的話。
- JMSSecurityException - 如果從屬站鑑別因無效使用者名稱或密碼而失效的話。

### createXATopicConnection

```
public XATopicConnection createXATopicConnection(java.lang.String userName,
                                                 java.lang.String password)
                                        throws JMSException
```

利用指定的使用者身份建立一個 XA 主題連線。這個連線是採用停止模式建立。在明確呼叫 Connection.start 之前，不會遞送任何訊息。

**參數：**

- userName - 呼叫端的使用者名稱
- password - 呼叫端的密碼

**傳回：** 新建立的 XA 主題連線。

**擲出：**

- JMSException - 如果 JMS 提供者因內部錯誤而無法建立 XA 主題連線的話。

- JMSSecurityException - 如果從屬站鑑別因無效使用者名稱或密碼而失效的話。

# XATopicSession

public interface **XATopicSession**
extends **XASession**

MQSeries 類別：**MQXATopicSession**

```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQXASession
            |
            +----com.ibm.mq.jms.MQXATopicSession
```

XATopicSession 提供可用來建立 TopicSubscribers 和 TopicPublishers 的 TopicSession。請參閱第351頁的『附錄E. 使用 WebSphere 的 JMS JTA/XA 介面』，以取得 MQ JMS 如何利用 XA 類別的詳細資料。

TopicSession 所對應的 XAResource 可藉由呼叫從 XASession 繼承來的 getXAResource 方法而取得。

另請參閱 **TopicSession** 和 **XASession**

## 方法

### getTopicSession

```
public TopicSession getTopicSession()
                             throws JMSException
```

取得這個 XATopicSession 所關聯的主題階段作業。

**傳回：** 主題階段作業物件。

**擲出：**

- JMSException - 如果發生 JMS 錯誤的話。

# 第4篇 附錄與後記

# 附錄A. 管理工具內容和可程式內容之間的對映

MQSeries Classes for Java 訊息服務提供利用 MQ JMS 管理工具或在應用程式中設定和查詢管理物件內容的機能。 表30.顯示管理工具所用的每個內容名稱和它參照的對應成員變數之間的對映。它也顯示儲存池所用的符號內容值及其可程式對等項之間的對映。

*表 30. 管理工具和可程式對等項之間的內容表示法比較。*

| 內容 | 成員變數名稱 | 內容值對映 | |
|---|---|---|---|
| | | 工具 | 程式 |
| DESCRIPTION | description | | |
| TRANSPORT | transportType | • BIND<br>• CLIENT | JMSC.MQJMS_TP_BINDINGS_MQ<br>JMSC.MQJMS_TP_CLIENT_MQ_TCPIP |
| CLIENTID | clientId | | |
| QMANAGER | queueManager* | | |
| HOSTNAME | hostName | | |
| PORT | port | | |
| CHANNEL | channel | | |
| CCSID | CCSID | | |
| RECEXIT | receiveExit | | |
| RECEXITINIT | receiveExitInit | | |
| SECEXIT | securityExit | | |
| SECEXITINIT | securityExitInit | | |
| SENDEXIT | sendExit | | |
| SENDEXITINIT | sendExitInit | | |
| TEMPMODEL | temporaryModel | | |
| MSGRETENTION | messageRetention | • YES<br>• NO | JMSC.MQJMS_MRET_YES<br>JMSC.MQJMS_MRET_NO |
| BROKERVER | brokerVersion | • V1 | JMSC.MQJMS_BROKER_V1 |
| BROKERPUBQ | brokerPubQueue | | |
| BROKERSUBQ | brokerSubQueue | | |
| BROKERDURSUBQ | brokerDurSubQueue | | |
| BROKERCCSUBQ | brokerCCSubQueue | | |
| BROKERCCDSUBQ | brokerCCDurSubQueue | | |
| BROKERQMGR | brokerQueueManager | | |
| BROKERCONQ | brokerControlQueue | | |
| EXPIRY | expiry | • APP<br>• UNLIM | JMSC.MQJMS_EXP_APP<br>JMSC.MQJMS_EXP_UNLIMITED |
| PRIORITY | priority | • APP<br>• QDEF | JMSC.MQJMS_PRI_APP<br>JMSC.MQJMS_PRI_QDEF |

## 內容

*表 30. 管理工具和可程式對等項之間的內容表示法比較。 (繼續)*

| 內容 | 成員變數名稱 | 內容值對映 | |
|---|---|---|---|
| | | 工具 | 程式 |
| PERSISTENCE | persistence | • APP<br>• QDEF<br>• PERS<br>• NON | JMSC.MQJMS_PER_APP<br>JMSC.MQJMS_PER_QDEF<br>JMSC.MQJMS_PER_PER<br>JMSC.MQJMS_PER_NON |
| TARGCLIENT | targetClient | • JMS<br>• MQ | JMSC.MQJMS_CLIENT_JMS_COMPLIANT<br>JMSC.MQJMS_CLIENT_NONJMS_MQ |
| ENCODING | encoding | | |
| QUEUE | baseQueueName | | |
| TOPIC | baseTopicName | | |
| **註:** * 代表 MQQueue 物件，成員變數名稱是 baseQueueManagerName | | | |

# 附錄B. MQSeries Class for Java 訊息服務所提供的 Script

下列檔案提供在 MQ JMS安裝結構的 bin 目錄中。提供這些 Script 是要協助您執行必要的常用作業來安裝或使用 MQ JMS。 表31.列出各 Script 及其用途。

*表 31. MQSeries Class for Java 訊息服務所提供的公用程式*

| 公用程式 | 用途 |
|---|---|
| IVTRun.bat<br>IVTTidy.bat<br>IVTSetup.bat | 用來執行點對點安裝驗證測試程式，說明於第21頁的『執行點對點 IVT』 中。 |
| PSIVTRun.bat | 用來執行發佈/訂閱安裝驗證測試程式，說明於第24頁的『發佈/訂閱安裝驗證測試』中。 |
| formatLog.bat | 用來將二進位日誌檔轉換成純文字，說明於第28頁的『日誌記載』中。 |
| JMSAdmin.bat | 用來執行管理工具，說明於第29頁的『第5章 使用 MQ JMS 管理工具』中。 |
| JMSAdmin.config | 管理工具的配置檔，說明於第30頁的『配置』中。 |
| runjms.bat | 協助執行 JMS 應用程式的公用程式 Script，說明於第27頁的『執行您自己的 MQ JMS 程式』中。 |
| PSReportDump.class | 用來檢視分配管理程式報告訊息，說明於第186頁的『處理分配管理程式報告』中。 |
| **註:** 在 UNIX 系統中，會忽略檔案名稱中的 '.bat' 副檔名。 | |

## Script

# 附錄C. Java 物件的 LDAP 伺服器配置

如果您利用 JNDI 來儲存 MQ JMS 所管理的物件，且您利用 LDAP 伺服器作爲您的 JNDI 服務提供者，伺服器必須是 LDAP 第 3 版（比方說，SecureWay® eNetwork Directory 第 3.1 版），且必須配置它來儲存 Java 物件。

## 檢查 LDAP 伺服器配置

如果要檢查是否有配置好 LDAP 伺服器來接受 Java 物件，請採用 LDAP 模式來執行 MQ JMS 管理工具（請參閱第29頁的『呼叫管理工具』）。

請嘗試利用下列指令來建立和顯示測試物件：

```
DEFINE QCF(ldapTest)
DISPLAY QCF(ldapTest)
```

如果沒有發生異常狀況，表示伺服器已正確配置，您可以繼續儲存 JMS 物件。

如果傳回 'SchemaViolationException'，或出現「無法連結物件」的訊息，表示伺服器的配置不正確。可能是沒有配置伺服器來儲存 Java 物件，也可能是物件的許可權或字尾不正確。下列程序應該可以協助您進行配置作業。

## 配置程序

許多 LDAP 伺服器都提供有管理伺服器的工具。請參閱您的伺服器文件，以取得如何使用這些工具的詳細資料。您應該可以利用這些工具來檢視和更新含有「屬性」和「物件類別」定義的綱要。

請確定綱要含有下列物件類別定義，必要的話，請新增它們：

```
( 1.3.6.1.4.1.42.2.27.4.2.1
  NAME 'javaContainer'
  DESC 'Container for a Java object'
  SUP top
  STRUCTURAL
  MUST ( cn )
)

( 1.3.6.1.4.1.42.2.27.4.2.4
  NAME 'javaObject'
  DESC 'Java object representation'
  SUP top
  ABSTRACT
  MUST ( javaClassName )
  MAY ( javaClassNames $
        javaCodebase $
        javaDoc $
        description )
)
```

```
( 1.3.6.1.4.1.42.2.27.4.2.5
  NAME 'javaSerializedObject'
  DESC 'Java serialized object'
  SUP javaObject
  AUXILIARY
  MUST ( javaSerializedData )
)

( 1.3.6.1.4.1.42.2.27.4.2.7
  NAME 'javaNamingReference'
  DESC 'JNDI reference'
  SUP javaObject
  AUXILIARY
  MAY ( javaReferenceAddress $
        javaFactory )
)
```

另外，也請確定綱要含有下列屬性定義，必要的話，請更新綱要：

```
( 1.3.6.1.4.1.42.2.27.4.1.11
        NAME 'javaReferenceAddress'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

( 1.3.6.1.4.1.42.2.27.4.1.10
        NAME 'javaFactory'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

( 1.3.6.1.4.1.42.2.27.4.1.7
        NAME 'javaCodebase'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

更新完成之後，請停止再重新啓動 LDAP 伺服器，再重複第347頁的『檢查 LDAP 伺服器配置』中所說明的配置檢查程序。

# 附錄D. 連接至 MQSeries Integrator 第 2 版

您可以利用 MQSeries Integrator 第 2 版：

- 作爲 MQ JMS 的發佈/訂閱分配管理程式
- 遞送或轉換 JMS 從屬站應用程式所建立的訊息以及將訊息傳送或發佈到 JMS 從屬站

## 發佈/訂閱

您可以利用 MQSeries Integrator 第 2 版作爲 MQ JMS 的發佈/訂閱分配管理程式。這需要下列設定活動：

- 基礎 MQSeries

  首先，您必須建立一個分配管理程式發佈佇列。這是分配管理程式佇列管理程式中的一個 MQSeries 佇列；它用來提出發佈資訊給分配管理程式。您可以選擇您自己的名稱給這個佇列，但它必須符合 TopicConnectionFactory's BROKERPUBQ 內容中的佇列名稱。依預設，TopicConnectionFactory 的 BROKERPUBQ 內容會設爲 SYSTEM.BROKER.DEFAULT.STREAM 值，除非您要在 TopicConnectionFactory 中配置不同的名稱，您應該將佇列命名爲 SYSTEM.BROKER.DEFAULT.STREAM。

- MQSeries Integrator 第 2 版

  其次，您要在分配管理程式的執行群組內設定一個*訊息流程*。這個訊息之目的是從分配管理程式發佈佇列中讀取訊息。（如果想要的話，您可以設定多個發佈佇列；每個佇列都需要它自己的 TopicConnectionFactory 和訊息流程。）

  基本訊息流程由一個 MQInput 節點組成（配置讀取 SYSTEM.BROKER.DEFAULT.STREAM 佇列），它的輸出會連接到 Publication（或 MQOutput）節點的輸入。

  因此，訊息流程圖解會有如下外觀：


*圖 7. MQSeries Integrator 訊息流程*

部署了這個訊息流程且啓動了分配管理程式之後，從 JMS 應用程式的觀點來看，MQSeries Integrator 第 2 版分配管理程式的行爲會如同 MQSeries 發佈/訂閱分配管理程式。您可以利用 MQSeries Integrator 控制中心來檢視現行訂閱狀態。

**註:**

1. MQSeries Classes for Java 訊息服務不需要修改。
2. MQSeries 發佈/訂閱和 MQSeries Integrator 第 2 版分配管理程式不能共存於相同佇列管理程式中。

3. 如果需要 MQSeries Integrator 第 2 版安裝和設定程序的說明，請參閱 *MQSeries Integrator for Windows NT 第 2.0 版安裝手冊*。

## 轉換和遞送

您可以利用 MQSeries Integrator 第 2 版來遞送或轉換 JMS 從屬站應用程式所建立的訊息，以及將訊息傳送或發佈到 JMS 從屬站。

MQSeries JMS 實作會依照第192頁的『MQRFH2 標題』中所說明，利用 MQRFH2 的 mcd 資料夾來攜帶訊息的相關資訊。依預設，會利用「訊息領域 (Msd)」內容來指出訊息是文字、位元組、串流、對映或物件訊息。

當 JMS 應用程式建立文字或位元組訊息時，應用程式可以置換這個 Msd 內容，且可以設定其它 mcd 資料夾欄位。它進行這項作業的方式，是設定一個「JMS 類型」內容，讓它有特殊的 URI 格式，例如：

```
mcd://domain/set/type[?format=fmt]
```

*domain*、*set*、*type* 和 *fmt* 等欄位中的值（其中 *fmt* 是選用的）會複製到送出的 MQRFH2。這表示應用程式可以將這些欄位設爲 MQSeries Integrator 第 2 版訊息流程能夠辨識的值。

# 附錄E. 使用 WebSphere 的 JMS JTA/XA 介面

MQSeries Class for Java 訊息服務包括 JMS XA 介面。這些介面使得 MQ JMS 能夠參與符合 Java 異動 API (JTA) 標準的異動管理程式所協調的兩段式確定。

這一節說明如何在 WebSphere Application Server 進階版中使用這些特性，使 WebSphere 能夠在整體異動中協調 JMS 傳送和接收作業和資料庫更新。

在 WebSphere 中使用 MQ JMS 和 XA 類別之前，可能會有其它安裝或配置步驟。請參閱 MQSeries 使用 Java SupportPac 網頁中的 `Readme.txt` 檔，以取得最新資訊 (`www.ibm.com/software/ts/mqseries/txppacs/ma88.html`)。

## 在 WebSphere 中使用 JMS

這一節要帶您在 WebSphere Application Server 進階版中使用 JMS 介面。

您必須已經瞭解 JMS 程式、MQSeries 及 EJB Bean 的基本概念。請參閱 JMS 規格、EJB 第 2 版規格（兩者可向 Sun 取得）、這份手冊、MQ JMS 所提供的範例，及 MQSeries 和 WebSphere 的其它手冊，以取得這些詳細資料。

### 管理物件

JMS 利用管理物件來封裝特定廠商的專用資訊。這可以將特定廠商專用細節爲一般使用者應用程式帶來的衝擊縮到最小。管理物件儲存在 JNDI 名稱空間中，在可攜方式下擷取和使用它，並不需要知道特定廠商專用的內容。

對於獨立式的使用，MQ JMS 提供有下列類別：

- MQQueueConnectionFactory
- MQQueue
- MQTopicConnectionFactory
- MQTopic

WebSphere 還提供了另外一對管理物件，使 MQ JMS 能夠與 WebSphere 整合起來：

- JMSWrapXAQueueConnectionFactory
- JMSWrapXATopicConnectionFactory

您利用和 MQQueueConnectionFactory 與 MQTopicConnectionFactory 完全相同的方式來使用這些物件。不過，它們在背後會使用 XA 版的 JMS 類別，且會在 WebSphere 異動中使用 MQ XAResource。

### 儲存器管理和 Bean 管理的異動

儲存器管理的異動是 EJB 儲存器自動在 EJB Bean 定界的異動。 Bean 管理的異動是程式（利用 UserTransaction 介面）自動在 EJB Bean 中定界的異動。

### 兩段式確定和一段式最佳化

WebSphere 只有在特定異動中使用了多個 XAResource 時，才會呼叫眞的兩段式確定。呼叫單一資源的異動會使用一段式最佳化來確定。這可以大幅降低在分散式和非分散式異動中使用不同 ConnectionFactory 的需要。

### 定義管理物件

您可以利用 MQ JMS 管理工具來定義 WebSphere 特定連線 Factory，並將它們儲存在 JNDI 名稱空間中。 *MQ_install_dir*/bin 中的 admin.config 檔應該有下列這幾行：

```
INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
PROVIDER_URL=iiop://hostname/
```

*MQ_install_dir* 是 MQ JMS 的安裝目錄， *hostname* 是執行 WebSphere 之機器的名稱或 IP 位址。

如果要存取 com.ibm.ejs.ns.jndi.CNInitialContextFactory，您必須將 WebSphere lib 目錄中的 ejs.jar 檔加入 CLASSPATH 中。

如果要建立新的 Factory，請使用下面這兩種新類型的定義動詞：

```
def WSQCF(name) [properties]
def WSTCF(name) [properties]
```

這些新類型利用對等的 QCF 或 TCF 類型的相同內容，不過，只能使用 BIND 傳輸類型（因此，不能配置從屬站內容）。如果需要詳細資料，請參閱第33頁的『管理 JMS 物件』。

### 擷取管理物件

在 EJB Bean 中，您可以利用 InitialContext.lookup() 方法來擷取 JMS 管理的物件，比方說：

```
InitialContext ic = new InitialContext();
TopicConnectionFactory tcf = (TopicConnectionFactory) ic.lookup("jms/Samples/TCF1");
```

這些物件可以強制轉型成通用的 JMS 介面，或當作通用 JMS 介面來使用。在應用程式碼中，通常不需要遵循 MQSeries 特定類別來進行程式設計。

## 範例

總共有三個範例要示範在 WebSphere Application Server 進階版中使用 MQ JMS 的基本概念。這些都在 *MQ_install_dir*/samples/ws 的子目錄中，其中 *MQ_install_dir* 是 MQ JMS 的安裝目錄。

- Sample1 示範利用儲存器管理異動，就佇列中的訊息進行簡單的 put 和 get。
- Sample2 示範利用 Bean 管理異動，就佇列中的訊息進行簡單的 put 和 get。
- Sample3 示範發佈/訂閱 API 的使用。

如果需要如何建置和部署 JEB Bean 的詳細資料，請參閱 WebSphere Application Server 文件。

每個範例目錄中的 readme.txt 檔都含有每個 EJB Bean 的範例輸出。提供的 Script 假設本端機器中有預設佇列管理程式。如果您的安裝和預設不同，您可以視需要來編輯這些 Script。

## Sample1

sample1 目錄中的 Sample1EJB.java 定義兩個使用 JMS 的方法：

- putMessage()，會將 TextMessage 傳送給佇列，並傳回所傳送之訊息的 MessageID
- getMessage()，會從佇列中讀取含指定 MessageID 的訊息

在執行範例之前，您必須將兩個管理物件儲存在 WebSphere JNDI 名稱空間中：

**QCF1** WebSphere 特定佇列連線 Factory

**Q1** 一個佇列

這兩個物件都必須連結於 jms/Samples 子環境定義中。

如果要設定管理物件，您可以利用 MQ JMS 管理工具並手動設定它們，您也可以使用所提供的 Script。

MQ JMS 管理工具必須配置爲可以存取 WebSphere 名稱空間。如果需要如何配置管理工具的詳細資料，請參閱第31頁的『WebSphere 配置』。

如果要用一般預設值來設定管理物件，您可以輸入下列指令來執行 admin.scp Script：

```
JMSAdmin < admin.scp
```

Bean 必須利用標示爲 TX_REQUIRED 的 getMessage 和 putMessage 方法來部署。這可以確保儲存器會在進入每個方法之前起始異動，在方法完成之後確定異動。在方法內，您不需要異動狀態所關聯的任何應用程式碼。不過，請記住，從 putMessage 送出的訊息會出現在同步點之下，且在異動確定之前無法使用。

sample1 目錄中有 Sample1Client.java，它是一個呼叫 EJB Bean 的簡單從屬站程式。另外還有一個 Script，runClient，用來簡化這個程式的執行。

這個從屬站程式（或 Script）採用單一參數，這個參數要作爲 TextMessage 的主體，而由 EJB Bean putMessage 方法來傳送這個訊息。之後，再呼叫 getMessage 從佇列中讀回訊息，並將主體傳回給從屬站顯示。 EJB Bean 會傳送進度訊息到應用程式伺服器的標準輸出 (stdout) 中，因此，您可以在執行期間監視輸出。

如果應用程式伺服器在從屬站的遠端機器上，您可能需要編輯 Sample1Client.java。如果您不要使用預設值，您可能需要編輯 runClient Script，以符合部署的 jar 檔的本端安裝路徑和名稱。

## Sample2

在 sample2 目錄中的 Sample2EJB.java 會執行和 sample1 相同的作業，也需要相同的管理物件。但 sample2 又和 sample1 不同，它使用 Bean 管理的異動來控制異動界限。

如果您還沒有執行 sample1，請確定您已依『Sample1』所說明，設定了管理物件 QCF1 和 Q1。

putMessage 方法和 getMessage 方法都是藉由取得 UserTransaction 的案例來啓動。它們利用這個案例，透過 UserTransaction.begin() 方法來建立一項異動。之後，程式碼的主體便和 sample1 相同，直到每個方法結束爲止。在每個方法結束時，由 UserTransaction.commit() 呼叫來完成異動。

sample2 目錄中有 Sample2Client.java，它是一個呼叫 EJB Bean 的簡單從屬站程式。另外還有一個 Script (runClient)，用來簡化這個程式的執行。您可以依第353頁的『Sample1』所說明的相同方式來使用這些東西。

## Sample3

sample3 目錄中的 Sample3EJB.java 示範在 WebSphere 中使用發佈/訂閱 API。發佈訊息和點對點的情況非常類似。不過，在透過 TopicSubscriber 來接收訊息時有不同。

發佈/訂閱程式通常會使用不可延續的訂閱者。這些不可延續的訂閱者只在擁有它們的階段作業的存活時間裡（或在明確關閉訂閱者之前）才存在。另外，它們也只能在這個存活時間裡，接收分配管理程式所送出的訊息。

如果要將 sample1 轉換成發佈/訂閱，您可能會將 putMessage 中的 QueueSender 改成 TopicPublisher，將 getMessage 中的 QueueReceiver 改成不可延續的 TopicPublisher。不過，這會因傳送訊息時分配管理程式並不知道主題的任何訂閱者而失敗。因此，訊息會被捨棄。

解決方案是在發佈訊息之前建立可延續的訂閱者。可延續的訂閱者會以可遞送端點的方式，越過階段作業的存活時間而持續存在。因此，在呼叫 getMessage() 期間，可以擷取訊息。

EJB Bean 包括兩個其它方法：

- createSubscription 會建立可延續的訂閱
- destroySubscription 會刪除可延續的訂閱

這些方法（附隨 putMessage 和 getMessage）必須利用 TX_REQUIRED 屬性來部署。

在執行 sample3 之前，您必須將兩個管理物件儲存在 WebSphere JNDI 名稱空間中：

TCF1

T1

這兩個物件都必須連結於 jms/Samples 子環境定義中。

如果要設定管理物件，您可以利用 MQ JMS 管理工具並手動設定它們，您也可以使用 Script。 sample3 目錄中有 admin.scp Script。

MQ JMS 管理工具必須配置爲可以存取 WebSphere 名稱空間。如果需要如何配置管理工具的詳細資料，請參閱第31頁的『WebSphere 配置』。

如果要用一般預設值來設定管理物件，您可以輸入下列指令來執行 admin.scp Script：

```
JMSAdmin < admin.scp
```

如果您已執行 admin.scp 來設定 sample1 或 sample2 的物件，當您執行 sample3 的 admin.scp 時，會出現錯誤訊息。（當您試圖建立 jms 和 Sample 子環境定義時，會發生這個情況。）您可以放心忽略這些錯誤訊息。

另外，在您執行 sample3 之前，請確定已安裝好 MQSeries 發佈/訂閱分配管理程式 (SupportPac MA0C)，且在執行中。

sample3 目錄中有 Sample3Client.java，它是一個呼叫 EJB Bean 的簡單從屬站程式。另外還有一個 Script，runClient，用來簡化這個程式的執行。您可以依第353頁的『Sample1』所說明的相同方式來使用這些東西。

# 附錄F. 注意事項

本資訊專爲 IBM 在美國提供的產品和服務而開發。在其他國家中，IBM 不一定會提供本文所提及的各項產品、服務或特性。如果需要當地的產品和服務相關資訊，請洽詢當地的 IBM 業務代表。本書在提及 IBM 的產品、程式或服務時，不表示或暗示只能使用 IBM 的產品、程式或服務。只要未侵犯 IBM 的智慧財產權，任何功能相當的產品、程式或服務都可以取代 IBM 的產品、程式或服務。不過，其他非 IBM 產品、程式、或服務在運作上的評價與驗證，其責任屬於使用者。

在本資訊中可能包含著 IBM 所擁有之專利或專利申請案。本資訊使用者並不享有前述專利之任何授權。您可以書面提出授權查詢，查詢請寄：

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

若要查詢有關二位元組 (DBCS) 資訊的特許權限事宜，請聯絡您國家的 IBM 智慧財產部門，或者用書面方式寄到：

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

**下列段落不適用於英國或此類條款和當地法律相抵觸的其他國家：** IBM 僅以「現狀」提供本書，而不提供任何明示或默示之保證（包括但不限於可售性或符合特定效用的保證。）有些地區不允許在某些異動上排除明示或默示之保證，因而這項聲明不一定適合您。

本資訊可能會有技術上或排版印刷上的訛誤。因此，IBM 會定期修訂；並將修訂後的內容納入新版中。同時，IBM 得隨時改進及/或變動本書所提及的產品及/或程式，而不會另作通知。

本資訊中任何對非 IBM 網站的敘述僅供參考，IBM 對該網站並不提供保證。這些網站中的材料不在本 IBM 產品的材料中，您在使用這些網站時，要自行承擔風險。

IBM 得以各種適當的方式使用或散布由 貴客戶提供的任何資訊，而無需對您負責。

本程式之獲授權者若希望取得相關資料，以便使用下列資訊者可洽詢 IBM。其下列資訊指的是： (1) 獨立建立的程式與其他程式 (包括此程式) 之間更換資訊的方式 (2) 相互使用已交換之資訊方法若有任何問題請聯絡：

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England

SO21 2JN.

上述資料之取得有其特殊要件，在某些情況下必須付費方得使用。

IBM 基於雙方之「IBM 客戶合約」、「國際程式授權合約」或任何同等合約之條款，提供本書中所說的授權程式與其所有適用的授權資料。

本書所提及之非 IBM 產品資訊，係一由產品的供應商，或其出版的聲明或其他公開管道取得。 IBM 並未測試過這些產品，也無法確認這些非 IBM 產品的執行效能、相容性、或任何對產品的其他主張是否完全無誤。如果您對非 IBM 產品的性能有任何的疑問，請逕向該產品的供應商查詢。

## 商標

下列詞彙是 International Business Machines Corporation 在美國和/其他國家的商標：

| | | |
|---|---|---|
| AIX | AS/400 | BookManager |
| CICS | IBM | IBMLink |
| Language Environment | MQSeries | MVS/ESA |
| OS/2 | OS/390 | OS/400 |
| SecureWay | SupportPac | System/390 |
| S/390 | VisualAge | VSE/ESA |
| WebSphere | | |

Java、HotJava、JDK 及所有以 Java 爲基礎的商標和標誌都是 Sun Microsystems, Inc. 在美國和（或）其他國家的商標或註冊商標。

Microsoft、Windows 和 Windows NT 是 Microsoft Corporation 在美國及/或其他國家的商標。

UNIX 是 The Open Group 在美國和其他國家的註冊商標。

其他公司、產品和服務名稱可能是第三者的商標或服務標記。

# 詞彙與縮寫名詞解釋

這份名詞解釋說明本書使用的詞彙及意義不同於日常用法的字詞。在某些情況下，在此提供的定義可能不是這個詞彙唯一適合的定義，但就這個字彙在本書使用的情況，此定義確實陳述出它的特別意義。

如果您找不到所需要的詞彙，請參閱索引或 *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994。

## 三劃

**子類別 (subclass).** 子類別是繼承另一類別的類別。子類別會繼承超類別的公開和受保護的方法及變數。

## 四劃

**介面 (interface).** 介面也是類別，但它只含有抽象方法，不含案例變數。介面提供一組共用的方法，供許多不同類別的子類別來實作。

**公開 (public).** 公開類別或介面在任何位置都是可見的。公開方法或變數在所屬類別爲可見的任何位置都是可見的。

**方法 (method).** 方法是一個物件導向程式設計詞彙，它代表函數或程序。

## 六劃

**全球資訊網 (World Wide Web, Web).** 全球資訊網是以一組通用的通信協定爲基礎的網際網路服務，可讓特定配置的伺服器電腦採用標準方式來利用網際網路分送文件。

## 七劃

**佇列 (queue).** 佇列是一個 MQSeries 物件。訊息佇列應用程式可以將訊息放入佇列中，也可以取得佇列中的訊息。

**佇列管理程式 (queue manager).** 佇列管理程式是一個系統程式，負責提供訊息佇列服務給應用程式。

**伺服器 (server).** (1) 在 MQSeries 中，伺服器是提供訊息佇列服務給遠端工作站所執行之從屬站應用程式的佇列管理程式。 (2) 更廣泛地說，伺服器是在主從架構特定兩程式資訊串流模型中回應資訊要求的程式。 (3) 執行伺服器程式的電腦。

**私密 (private).** 私密欄位在本身所屬類別之外不可見。

## 八劃

**物件 (object).** (1) 在 Java 中，物件是一個類別案例。類別是一個事物群組的模型；物件則是該群組特定成員的模型。 (2) 在 MQSeries 中，物件是佇列管理程式、佇列或通道。

**物件管理團體 (Object Management Group, OMG).** 這是設定物件導向程式設計標準的一個國際協會。

## 九劃

**保護 (protected).** 保護欄位只在本身所屬類別、子類別或類別所屬套件內是可見的。

**封裝 (encapsulation).** 封裝是一種物件導向的程式設計技術，它使物件的資料成爲私密或受保護的，讓程式設計師只能透過方法呼叫來存取和操作資料。

## 十劃

**套件 (package).** 在 Java 中，套件是讓一段 Java 程式碼存取一組特定類別的方法。屬於特定套件的 Java 程式碼可以存取套件內的所有類別，也可以存取類別中所有非私密的方法和欄位。

**案例 (instance).** 案例是一個物件。當建立類別案例來產生物件時，我們說，這物件是類別的一個案例。

**訊息 (message).** 在訊息佇列應用程式中，訊息是一項在程式之間傳送的通信。

**訊息佇列 (message queue).** 請參閱「佇列 (queue)」。

**訊息佇列運作 (message queuing).** 這是一種程式設計技術，在這種技術中，應用程式內的每個程式都藉由將訊息放在佇列中，來與其它程式通信。

## 十一劃

**強制轉型 (casting).** 這是 Java 用來說明將某物件或基本類型的值明確轉換成另一類型的詞彙。

**從屬站 (client).** 在 MQSeries 中，從屬站是一個執行時期元件，負責提供本端使用者應用程式存取伺服器之佇列服務的功能。

**統一資源定址器 (Uniform Resource Locator, URL).** 這是代表電腦或網路（如網際網路）中之資訊資源的字元序列。

**通道 (channel).** 請參閱「MQI 通道 (MQI channel)」。

# 十二劃

**超文字標記語言 (Hypertext Markup Language, HTML).** 這是用來定義全球資訊網中所呈現之資訊的一種語言。

**超載 (overloading).** 這是一個識別碼指向相同範圍內的多個項目的情況。在 Java 中，方法可以超載，但變數和運算子不能超載。

**超類別 (superclass).** 超類別是其它類別所繼承的類別。超類別的公開和受保護的方法及變數可以提供給子類別使用。

# 十三劃

**傳輸控制通信協定/網際網路通信協定 (Transmission Control Protocol/Internet Protocol, TCP/IP).** 這是一組支援區域和廣域網路之對等式連線功能的通信協定。

# 十四劃

**網際網路 (Internet).** 網際網路是共用資訊的協同公用網路。在實體上，網際網路使用目前所有現存公用電訊網路整體資源的一部份。在技術上，將網際網路區分出來成爲協同公用網路的因素，是它使用一組稱爲 TCP/IP（傳輸控制通信協定/網際網路通信協定）的通信協定。

**輕裝備目錄存取通信協定 (Lightweight Directory Access Protocol, LDAP).** 這是一種存取目錄服務的主從式通信協定。

# 十七劃

**應用程式設計 (Application Programming Interface, API).** 應用程式設計介面由程式設計師可在其應用程式中使用的函數及變數組成。

# 十九劃

**類別 (class).** 類別是資料及處理資料之方法的封裝集成。類別可以建立案例來產生類別案例物件。

# A

**Abstract Window Toolkit for Java (AWT).** 這是利用元件的原生平台版來實作的圖形使用者介面 (GUI) 集成。

**API.** 這是「應用程式設計介面 (Application Programming Interface)」的縮寫。

**Applet.** 這是專爲了在網頁中執行而設計的 Java 程式。

**AWT.** 這是 Abstract Window Toolkit for Java 的縮寫。

# E

**EJB.** 這是 Enterprise JavaBean 的縮寫。

**Enterprise JavaBean (EJB).** 這是 Sun Microsystems 所發行的伺服器端元件架構，用來撰寫可重複使用的商業邏輯及可攜式企業應用程式。 Enterprise JavaBean 元件完全是用 Java 來撰寫的，可執行於任何符合 EJB 的伺服器上。

# H

**HTML.** 這是「超文字標記語言 (Hypertext Markup Language)」的縮寫。

# I

**IEEE.** 這是「美國電機暨電子工程師學會 (Institute of Electrical and Electronics Engineers)」的縮寫。

**IIOP.** 這是 Internet Inter-ORB Protocol 的縮寫。

**Internet Inter-ORB Protocol (IIOP).** 這是在 ORB 和不同供應商之間的 TCP/IP 通信標準。

# J

**J2EE.** 這是「Java 2 平台企業版 (Java 2 Platform, Enterprise Edition)」的縮寫。

**JAAS.** 這是「Java 鑑別和授權服務 (Java Authentication and Authorization Service)」的簡稱。

**Java 2 平台企業版 (Java 2 Platform, Enterprise Edition, J2EE).** 這是一組服務、API 和通信協定，用以提供開發多層 Web 型應用程式的功能。

**Java Developers Kit (JDK).** 這是 Sun Microsystems 爲 Java 開發人員而發行的軟體套件。它含有 Java 直譯器、Java 類別和 Java 開發工具：編譯器、除錯器、反組合程式、Applet Viewer、Stub 檔產生器及文件產生器。

**Java Runtime Environment (JRE).** 這是 Java Development Kit (JDK) 的一個子集，其中含有基核執行檔和構成標準 Java 平台的檔案。 JRE 包括 Java 虛擬機器、基核類別及支援的檔案。

**Java 命名和目錄服務 (Java Naming and Directory Service, JNDI).** 這是 Java 程式設計語言所指定的一種 API。它提供命名和目錄功能給用 Java 程式設計語言所撰寫的應用程式。

**Java 訊息服務 (Java Message Service, JMS).** 這是 Sun Microsystem 用來從 Java 程式存取企業傳訊系統的 API。

**Java 異動 API (Java Transaction API, JTA).** 這是可讓應用程式和 J2EE 伺服器存取異動的 API。

**Java 異動服務 (Java Transaction Service, JTS).** 這是支援 JTA 和在 API 層次之下實作 OMG Object Transaction Service 1.1 規格的異動管理程式。

**Java 虛擬機器 (Java Virtual Machine, JVM).** 這是執行編譯 Java 程式碼（Applet 和應用程式）的中央處理單元 (CPU) 的軟體實作。

**Java 鑑別和授權服務 (Java Authentication and Authorization Service, JAAS).** 這是提供實體鑑別和存取控制的 Java 服務。

**JDK.** 這是 Java Developers Kit 的縮寫。

**JMS.** 這是「Java 訊息服務 (Java Message Service)」的縮寫。

**JNDI.** 這是「Java 命名和目錄服務 (Java Naming and Directory Service)」的縮寫。

**JRE.** 這是 Java Runtime Environment 的縮寫。

**JTA.** 這是「Java 異動 API (Java Transaction API)」的縮寫。

**JTS.** 這是「Java 異動服務 (Java Transaction Service)」的縮寫。

**JVM.** 這是「Java 虛擬機器 (Java Virtual Machine)」的縮寫。

# L

**LDAP.** 這是「輕裝備目錄存取通信協定 (Lightweight Directory Access Protocol)」的縮寫。

# M

**MQDLH.** 這是指 MQSeries 無法傳送的郵件標題。請參閱 *MQSeries Application Programming Reference*。

**MQI 通道 (MQI channel).** MQI 通道會將 MQSeries 從屬站連接到伺服器系統中之佇列管理程式，且會轉換 MQI 呼叫，並採雙向方式進行回應。

**MQMD.** 這是「MQSeries 訊息描述子 (MQSeries Message Descriptor)」的縮寫。

**MQSC.** 這是「MQSeries 指令(MQSeries commands)」的縮寫。

**MQSeries.** MQSeries 是一系列提供訊息佇列服務的 IBM 授權程式。

**MQSeries 指令 (MQSeries commands, MQSC).** 這是一種人類能夠閱讀的指令，用來操作 MQSeries 物件，且具有跨越所有平台之統一性。

**MQSeries 訊息描述子 (MQSeries Message Descriptor, MQMD).** 這負責控制說明訊息格式及內容的資訊，會攜帶在 MQSeries 訊息中。

# O

**Object Request Broker (ORB).** 這是一種應用系統架構，負責爲不同語言所建置、執行於不同機器、位於異質的分散式環境中的物件，提供交互作業能力。

**OMG.** 這是「物件管理團體 (Object Management Group)」的縮寫。

**ORB.** 這是 Object Request Broker 的縮寫。

# R

**Red Hat 套裝管理程式 (Red Hat Package Manager, RPM).** 這是一種軟體套裝系統，適用於 Red Hat Linux 平台及其它 Linux 和 UNIX 平台。

**RPM.** 這是「Red Hat 套裝管理程式 (Red Hat Package Manager)」的縮寫。

# S

**Servlet.** 這是專爲了在 Web 伺服器中執行而設計的 Java 程式。

# T

**TCP/IP.** 這是「傳輸控制通信協定/網際網路通信協定 (Transmission Control Protocol/Internet Protocol)」的縮寫。

# U

**URL.** 這是「統一資源定址器 (Uniform Resource Locator)」的縮寫。

# V

**VisiBroker for Java.** 這是用 Java 撰寫的 Object Request Broker (ORB)。

# W

**Web.** 請參閱「全球資訊網 (World Wide Web)」。

**Web 瀏覽器 (Web browser).** 這是一種爲全球資訊網所分送的資訊建立格式並加以顯示的程式。

# 參考書目

這一節說明所有現行 MQSeries 產品的可用文件。

## MQSeries 跨平台書籍

這些書籍有時稱爲 MQSeries 『系列』書籍，其中的大部份都適用於所有 MQSeries 層次 2 產品。以下是最新的 MQSeries 層次 2 產品：

- MQSeries AIX 5.2 版
- MQSeries for AS/400 5.2 版
- MQSeries for AT&T GIS UNIX, V2.2
- MQSeries for Compaq (DIGITAL) OpenVMS 2.2.1.1 版
- MQSeries Compaq Tru64 UNIX 5.1 版
- MQSeries HP-UX 5.2 版
- MQSeries Linux 5.2 版
- MQSeries OS/2 Warp 5.1 版
- MQSeries for OS/390 5.2 版
- MQSeries for SINIX 和 DC/OSx 2.2 版
- MQSeries Sun Solaris 5.2 版
- MQSeries Sun Solaris Intel Platform Edition 5.1 版
- MQSeries for Tandem NonStop Kernel 2.2.0.1 版
- MQSeries for VSE/ESA 第 2.1.1 版
- MQSeries for Windows, V2.0
- MQSeries for Windows, V2.1
- MQSeries Windows NT 和 Windows 2000 5.2 版

以下是 MQSeries 跨平台書籍：

- *MQSeries Brochure*, G511-1908
- *An Introduction to Messaging and Queuing*, GC33-0805
- *MQSeries 交互通信*, SC40-0328
- *MQSeries Queue Manager Clusters*, SC34-5349
- *MQSeries 從屬站*, GC40-1554
- *MQSeries System Administration*, SC33-1873
- *MQSeries MQSC 指令參考手冊*, SC40-0327
- *MQSeries Event Monitoring*, SC34-5760
- *MQSeries Programmable System Management*, SC33-1482
- *MQSeries Administration Interface Programming Guide and Reference*, SC34-5390
- *MQSeries 訊息*, GC40-1560
- *MQSeries Application Programming Guide*, SC33-0807
- *MQSeries Application Programming Reference*, SC33-1673
- *MQSeries Programming Interfaces Reference Summary*, SX33-6095
- *MQSeries Using C++* (SC33-1877)
- *MQSeries 使用 Java* (SC40-0625)
- *MQSeries Application Messaging Interface* (SC34-5604)

## MQSeries 特定平台書籍

每個 MQSeries 產品，除了 MQSeries 系列書籍之外，都至少還有一本特定平台專用書籍。

**MQSeries AIX 5.2 版**

*MQSeries AIX 快速入門*, GC40-1555

**MQSeries for AS/400 5.2 版**

*MQSeries for AS/400 Quick Beginnings*, GC34-5557

*MQSeries for AS/400 System Administration*, SC34-5558

*MQSeries for AS/400 Application Programming Reference (ILE RPG)*, SC34-5559

**MQSeries for AT&T GIS UNIX, V2.2**

*MQSeries for AT&T GIS UNIX System Management Guide*, SC33-1642

**MQSeries for Compaq (DIGITAL) OpenVMS 2.2.1.1 版**

*MQSeries for Compaq (DIGITAL) OpenVMS System Management Guide*, GC33-1791

**MQSeries Compaq Tru64 UNIX 5.1 版**

*MQSeries Compaq Tru64 UNIX Quick Beginnings*, GC34-5684

**MQSeries HP-UX 5.2 版**

*MQSeries HP-UX 快速入門*, GC40-1557

**MQSeries Linux 5.2 版**

## 参考書目

| *MQSeries Linux Quick Beginnings*,
| GC34-5691

**MQSeries OS/2 Warp 5.1 版**

*MQSeries OS/2 Warp 快速入門*, GC40-1556

**MQSeries for OS/390 5.2 版**

*MQSeries for OS/390 Concepts and Planning Guide*, GC34-5650

*MQSeries for OS/390 System Setup Guide*, SC34-5651

*MQSeries for OS/390 System Administration Guide*, SC34-5652

*MQSeries for OS/390 Problem Determination Guide*, GC34-5892

*MQSeries for OS/390 Messages and Codes*, GC34-5891

*MQSeries for OS/390 Licensed Program Specifications*, GC34-5893

*MQSeries for OS/390 Program Directory*

**MQSeries link for R/3, Version 1.2**

*MQSeries link for R/3 User's Guide*, GC33-1934

**MQSeries for SINIX 和 DC/OSx 2.2 版**

*MQSeries for SINIX and DC/OSx System Management Guide*, GC33-1768

**MQSeries Sun Solaris 5.2 版**

*MQSeries Sun Solaris 快速入門*, GC40-1558

**MQSeries Sun Solaris Intel Platform Edition 5.1 版**

*MQSeries Sun Solaris Intel Platform Edition Quick Beginnings*, GC34-5851

**MQSeries for Tandem NonStop Kernel 2.2.0.1 版**

*MQSeries for Tandem NonStop Kernel System Management Guide*, GC33-1893

**MQSeries for VSE/ESA 第 2.1.1 版**

*MQSeries for VSE/ESA™ Licensed Program Specifications*, GC34-5365

*MQSeries for VSE/ESA System Management Guide*, GC34-5364

**MQSeries for Windows, V2.0**

*MQSeries for Windows User's Guide*, GC33-1822

**MQSeries for Windows, V2.1**

*MQSeries for Windows User's Guide*, GC33-1965

**MQSeries Windows NT 和 Windows 2000 5.2 版**

*MQSeries Windows NT 快速入門*, GC40-0314

*MQSeries for Windows NT Using the Component Object Model Interface*, SC34-5387

*MQSeries LotusScript Extension*, SC34-5404

## 軟本書籍

MQSeries 的大部份書籍都有軟本和印刷本兩種格式。

### HTML 格式

下列 MQSeries 產品都有相關的 HTML 格式 MQSeries 文件：

- MQSeries AIX 5.2 版
- MQSeries for AS/400 5.2 版
- MQSeries Compaq Tru64 UNIX 5.1 版
- MQSeries HP-UX 5.2 版
- MQSeries Linux 5.2 版
- MQSeries OS/2 Warp 5.1 版
- MQSeries for OS/390 5.2 版
- MQSeries Sun Solaris 5.2 版
- MQSeries Sun Solaris Intel Platform Edition 5.1 版
- MQSeries Windows NT 和 Windows 2000 5.2 版（編譯 HTML）
- MQSeries link for R/3, V1.2

您也可以從 MQSeries 產品系列網站取得 HTML 格式的 MQSeries 書籍，網址如下：

```
http://www.ibm.com/software/mqseries/
```

### 可攜性文件格式 (PDF)

您可以利用 Adobe Acrobat Reader 來檢視和列印 PDF 檔案。

如果您需要取得 Adobe Acrobat Reader 或需要支援 Acrobat Reader 的平台最新的相關資訊，請造訪 Adobe Systems Inc. 網站，網址如下：

```
http://www.adobe.com/
```

下列 MQSeries 產品都有相關 MQSeries 書籍的 PDF 版：

### 參考書目

- MQSeries AIX 5.2 版
- MQSeries for AS/400 5.2 版
- MQSeries Compaq Tru64 UNIX 5.1 版
- MQSeries HP-UX 5.2 版
- MQSeries Linux 5.2 版
- MQSeries OS/2 Warp 5.1 版
- MQSeries for OS/390 5.2 版
- MQSeries Sun Solaris 5.2 版
- MQSeries Sun Solaris Intel Platform Edition 5.1 版
- MQSeries Windows NT 和 Windows 2000 5.2 版
- MQSeries link for R/3, V1.2

您也可以從 MQSeries 產品系列網站中取得所有現行 MQSeries 書籍的 PDF 版本：

```
http://www.ibm.com/software/mqseries/
```

### BookManager® 格式

在各種線上書庫集成套件中，都有 IBM BookManager 格式的 MQSeries 書籍，其中包括 *Transaction Processing and Data* 集成套件 (SK2T-0730)。您可以利用下列 IBM 授權程式來檢視 IBM BookManager 格式的軟本書籍：

BookManager READ/2
BookManager READ/6000
BookManager READ/DOS
BookManager READ/MVS
BookManager READ/VM
BookManager READ for Windows

### PostScript 格式

許多 MQSeries 第 2 版產品都有 PostScript (.PS) 格式的 MQSeries 書庫。 PostScript 格式的書籍可在 PostScript 印表機中加以列印，也可以利用適當的檢視器來檢視。

### Windows 說明格式

MQSeries for Windows 第 2.0 版和 MQSeries for Windows 第 2.1 版都有 Windows 說明格式的 *MQSeries for Windows User's Guide*。

## 網際網路中的 MQSeries 資訊

MQSeries 產品系列網站位於：

```
http://www.ibm.com/software/mqseries/
```

您可以利用這個網站所提供的各個鏈結來完成下列動作：

- 取得 MQSeries 產品系列的最新相關資訊。
- 存取 HTML 和 PDF 格式的 MQSeries 書籍。
- 下載 MQSeries SupportPac。

# 索引

索引順序以中文字，英文字，及特殊符號之次序排列。

## 〔一劃〕



## 〔二劃〕



## 〔三劃〕



## 〔四劃〕



## 〔五劃〕



## 〔六劃〕



## 〔七劃〕



## 〔八劃〕



## 〔九劃〕



## 〔十劃〕

## 〔十一劃〕



## 〔十二劃〕



## 〔十三劃〕

## B



## C



## D



## E



## F



## H



## I



## J



## L



## M

# N



# O



# P



# Q



# R



# S

# T



# U



# V



# W



# X

## 將意見送交 IBM

如果您特別喜歡或不喜歡本書的任何部份，請利用下列任何方法，將您的意見傳送給 IBM。

您可以就所發現的特定錯誤或疏漏之處或本書的精確性、組織方式、主題內容或完整性，自在提出意見，請不要感到拘束。

請將您的意見著眼在本書中的資訊及資訊的呈現方式。

**如果有關於 IBM 產品或系統的意見，請告訴您的 IBM 業務化表或 IBM 授權經銷商。**

當意見送交 IBM 時，您同時也將非專用的權利授與 IBM，同意其採用任何自認爲適當的方式來使用或發佈您的意見，無需向您負責。

您可以採用下列中的任何方式，將意見送交 IBM：

- 郵件，來函請寄到：

  User Technologies Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  SO21 2JN
  United Kingdom
- 傳眞：
  - 英國以外，在您的國際存取碼之後，使用 44–1962–870229
  - 英國以內，使用 01962–870229
- 電子方式，請使用適當的網路 ID：
  - IBM Mail Exchange：GBIBM2Q9 at IBMMAIL
  - IBMLink™: HURSLEY(IDRCF)
  - 網際網路：idrcf@hursley.ibm.com

不論您採用任何方法，請務必併入：

- 書籍標題及訂購號碼
- 意見所適用的主題
- 您的姓名及地址/電話號碼/傳眞號碼/網路 ID。

# 讀者意見表

爲使本書盡善盡美，本公司極需您寶貴的意見；懇請您使用過後，撥冗填寫下表，惠予指教。

請於下表適當空格內，填入記號（∨）；我們會在下一版中，作適當修訂，謝謝您的合作!

| 評估項目 | 評　估　意　見 | | 備　註 |
|---|---|---|---|
| 正確性 | 內容說明與實際程序是否符合 | □是□否 | |
| | 參考書目是否正確 | □是□否 | |
| 一致性 | 文句用語及風格，前後是否一致 | □是□否 | |
| | 實際畫面訊息與本書所提之畫面訊息是否一致 | □是□否 | |
| 完整性 | 是否遺漏您想知道的項目 | □是□否 | |
| | 字句、章節是否有遺漏 | □是□否 | |
| 術語使用 | 術語之使用是否恰當 | □是□否 | |
| | 術語之使用，前後是否一致 | □是□否 | |
| 可讀性 | 文句用語是否通順 | □是□否 | |
| | 有否不知所云之處 | □是□否 | |
| 內容說明 | 內容說明是否詳盡 | □是□否 | |
| | 例題說明是否詳盡 | □是□否 | |
| 排版方式 | 本書的形狀大小，版面安排是否方便使用 | □是□否 | |
| | 字體大小，顏色編排，是否有助於閱讀 | □是□否 | |
| 目錄索引 | 目錄內容之編排，是否便於查考 | □是□否 | |
| | 索引語錄之排定，是否便於查考 | □是□否 | |
| | ※評估意見爲"否"者，請於備註欄說明。 | | |

其他：（篇幅不夠時，請另紙說明。）

上述改正意見，一經採用，本公司有合法之使用及發佈權利，特此聲明。

# 讀者意見表

折疊線

台北市115南港區三重路十九之十一號E棟九樓

軟體國際股份有限公司
臺灣商業機器
國際商業機器
臺灣國際軟體

啟

寄

折疊線

廣告回信
台灣北區郵政管理局
登記證
北台字第0587號
（免貼郵票）

寄件人
姓名：
地址：

Printed in Singapore

SC40-0625-06