9.3

開發 *IBM MQ* 的應用程式

附註

使用本資訊及其支援的產品之前，請先閱讀第 1089 頁的『注意事項』中的資訊。

除非新版中另有指示，否則此版本適用於 IBM® MQ 9.5.3 版及所有後續版本與修訂版。

當您將資訊傳送至 IBM 時，您授與 IBM 非專屬權利，以任何其認為適當的方式使用或散佈資訊，而無需對您負責。

© Copyright International Business Machines Corporation 2007, 2024.

目錄

開發適用於 IBM MQ 的應用程式

您可以開發應用程式來傳送及接收訊息，以及管理佇列管理程式和相關資源。 IBM MQ 支援以許多不同語言及架構撰寫的應用程式。

開發 IBM MQ 應用程式的新手?

若要瞭解如何開發 IBM MQ 的應用程式，請造訪 IBM Developer:

- IBM MQ Developer Essentials (瞭解基本觀念、執行展示、編寫應用程式、採用更進階的指導教學)
- IBM MQ Downloads for Developers (包括免費開發人員版本及試用版)

如果您熟悉下列各節中說明的概念，也可能會發現開發應用程式更容易:

- 第 6 頁的『應用程式開發概念』
- 第 41 頁的『IBM MQ 應用程式的設計考量』

支援物件導向語言及架構

IBM MQ 為以下列語言及架構開發的應用程式提供核心支援:

- JMS
- Java
- C++
- .NET

另請參閱第 14 頁的『物件導向應用程式』。

.NET 支援以多種語言開發的應用程式。 為了說明如何使用 .NET 的 IBM MQ 類別來存取 IBM MQ 佇列， MQ 產品說明文件包含下列語言的資訊:

- C# 範例程式碼 和 範例應用程式
- C++ 範例應用程式
- Visual Basic 範例應用程式

請參閱 第 484 頁的『撰寫及部署 IBM MQ .NET 程式』。

IBM MQ 支援 .NET Core ， 適用於來自 IBM MQ 9.1.1 的 Windows 環境中的應用程式，以及來自 IBM MQ 9.1.2 的 Linux® 環境中的應用程式。 如需相關資訊，請參閱第 465 頁的『正在安裝 IBM MQ classes for .NET』。

ALW IBM MQ 也支援實作 OASIS AMQP 1.0 通訊協定的 AMQP 用戶端。

MQ Light、 Apache Qpid 用戶端 (例如 Apache Qpid Proton 及 Apache Qpid JMS API) 基於此通訊協定。

MQ Light API 位於 IBM MQ Light。

Apache Qpid 用戶端可在 QPid Proton 取得。

下列語言連結依現狀提供:

- Go 連結
- JavaScript API 實作，與 Node.js 應用程式搭配使用

支援程式化 REST API

IBM MQ 提供下列程式化 REST API 的支援，以傳送及接收訊息:

- IBM MQ messaging REST API

© Copyright IBM Corp. 2007, 2024

- z/OS IBM z/OS Connect EE
- IBM Integration Bus
- IBM DataPower 閘道

請參閱 第 588 頁的『使用 IBM MQ 開發 REST 應用程式』，並在 IBM Developer 的 IBM MQ 區域中， 開始使用 IBM MQ 傳訊 REST API 指導教學。 本指導教學包含下列語言的範例 (依現狀提供)， 以與 IBM MQ messaging REST API 搭配使用:

- 使用 MQ 傳訊 REST API 的範例
- 使用 HTTPS 模組的 Node.js 範例
- 含有 Promise 模組的 Node.js 範例

支援程序化程式設計語言

IBM MQ 提供下列程序化程式設計語言所開發的應用程式支援:

- C
- Windows Visual Basic (僅限 Windows 系統)
- COBOL
- z/OS Assembler (僅限 IBM MQ for z/OS)
- z/OS PL/I (僅限 IBM MQ for z/OS)
- IBM i RPG (僅限 IBM MQ for IBM i)

這些語言使用訊息佇列介面 (MQI) 來存取訊息佇列作業服務。 請參閱 第 600 頁的『使用 IBM MQ 開發 MQI 應用程式』。 請注意，物件導向語言及架構所使用的「 IBM MQ 物件模型」提供使用 MQI 之程序化語言無法使用的其他功能。

指定應用程式名稱

ALW

在 IBM MQ 9.1.2 之前，您可以在 Java 或 JMS 用戶端應用程式上指定應用程式名稱。 從 IBM MQ 9.1.2 開始，您也可以在其他程式設計語言上指定應用程式名稱。 如需相關資訊，請參閱 第 44 頁的『以支援的程式設計語言指定應用程式名稱』。

相關工作

第 1044 頁的『開發適用於 MQ Telemetry 的應用程式』

第 1063 頁的『使用 IBM MQ 開發 Microsoft Windows Communication Foundation 應用程式』
IBM MQ 的 Microsoft Windows Communication Foundation (WCF) 自訂通道會在 WCF 用戶端與服務之間傳送及接收訊息。

相關參考

第 1025 頁的『開發適用於 Managed File Transfer 的應用程式』
指定要與 Managed File Transfer 搭配執行的程式、搭配使用 Apache Ant 與 Managed File Transfer、搭配使用自訂 Managed File Transfer 與使用者結束程式，以及透過將訊息放置在代理程式指令佇列上來控制 Managed File Transfer 。

應用程式開發概念

您可以使用選擇的程序化或物件導向語言來撰寫 IBM MQ 應用程式。 在開始設計及撰寫 IBM MQ 應用程式之前，請先熟悉基本 IBM MQ 概念。

如需您可以針對 IBM MQ 撰寫之應用程式類型的相關資訊，請參閱 第 5 頁的『開發適用於 IBM MQ 的應用程式』 及 第 7 頁的『應用程式可以執行的動作』。

相關概念

第 41 頁的『IBM MQ 應用程式的設計考量』

當您決定應用程式如何利用可供您使用的平台及環境時，您需要決定如何使用 IBM MQ 所提供的特性。

應用程式可以執行的動作

您可以開發應用程式，以傳送及接收支援商業程序所需的訊息。 您也可以開發應用程式來管理佇列管理程式及相關資源。

應用程式可以在 IBM MQ for Multiplatforms 上執行的動作

在 多平台上，您可以撰寫執行下列動作的應用程式:

- 將訊息傳送至在相同作業系統下執行的其他應用程式。 應用程式可以位於相同或另一個系統上。
- 將訊息傳送至在其他 IBM MQ 平台上執行的應用程式。
- 針對下列系統，從 CICS 內使用訊息佇列作業:
  - 

TXSeries for AIX
  - 

IBM i
  - 

Windows
- 將 Encina 內的訊息佇列作業用於下列系統:
  - 

AIX
  - 

Windows
- 針對下列系統使用 Tuxedo 內的訊息佇列作業:
  - 

AIX
  - AT&T
  - 

Windows
- 使用 IBM MQ 作為交易管理程式，在 IBM MQ 工作單元內協調外部資源管理程式所做的更新。 下列外部資源管理程式受支援且符合 X/OPEN XA 介面
  - Db2
  - Informix
  - Oracle
  - Sybase
- 將數個訊息作為單一工作單元一起處理，可確定或取消。
- 從完整 IBM MQ 環境執行，或從 IBM MQ 用戶端環境執行。

應用程式可以在 IBM MQ for z/OS 上執行的動作

在 z/OS 上，您可以撰寫執行下列動作的應用程式:

- 在 CICS 或 IMS 內使用訊息佇列作業。
- 在批次、 CICS 及 IMS 應用程式之間傳送訊息，為每一個功能選取最適當的環境。
- 將訊息傳送至在其他 IBM MQ 平台上執行的應用程式。
- 將數個訊息作為單一工作單元一起處理，可確定或取消。
- 透過 IMS 橋接器，將訊息傳送至 IMS 應用程式，並與之互動。

- 參與由 RRS 協調的工作單位。

z/OS 內的每一個環境都有自己的性質、優點及缺點。 IBM MQ for z/OS 的優點是應用程式不會關聯於任何一個環境，但可以配送以利用每一個環境的好處。 例如，您可以使用 TSO 或 CICS 開發一般使用者介面，您可以在 z/OS 批次中執行處理密集模組，並且可以在 IMS 或 CICS 中執行資料庫應用程式。 無論如何，應用程式的各個部分都可以使用訊息和佇列進行通訊。

IBM MQ 應用程式的設計者必須瞭解這些環境所強加的差異和限制。 例如：

- IBM MQ 提供容許佇列管理程式之間交互通訊的機能 (這稱為 分散式佇列作業)。
- 在批次和 CICS 環境之間，確定和取消變更的方法各不相同。
- IBM MQ for z/OS 在 IMS 環境中提供線上訊息處理程式 (MPP)、互動式捷徑程式 (IFP) 及批次訊息處理程式 (BMP) 的支援。 如果您要撰寫批次 DL/I 程式，請遵循主題 (例如 第 859 頁的『建置 z/OS 批次應用程式』 及 第 613 頁的『z/OS 批次考量』 for z/OS 批次程式) 中提供的指引。
- 雖然單一 z/OS 系統上可以存在多個 IBM MQ for z/OS 實例，但 CICS 區域一次只能連接至一個佇列管理程式。 不過，多個 CICS 區域可以連接至相同的佇列管理程式。 在 IMS 及 z/OS 批次環境中，程式可以連接至多個佇列管理程式。
- IBM MQ for z/OS 容許由一組佇列管理程式共用本端佇列，以提高傳輸量及可用性。 這類佇列稱為 共用佇列，且佇列管理程式會形成 佇列共用群組，可處理相同共用佇列上的訊息。 透過指定佇列共用群組名稱而非特定佇列管理程式名稱，批次應用程式可以連接至佇列共用群組內數個佇列管理程式的其中一個。 這稱為 群組批次連接，或更簡單的 群組連接。 請參閱 共用佇列及佇列共用群組。

z/OS 第 745 頁的『在 IBM MQ for z/OS 上使用及撰寫應用程式』中進一步說明受支援環境之間的差異及其限制。

相關概念

第 6 頁的『應用程式開發概念』
您可以使用選擇的程序化或物件導向語言來撰寫 IBM MQ 應用程式。 在開始設計及撰寫 IBM MQ 應用程式之前，請先熟悉基本 IBM MQ 概念。

第 41 頁的『IBM MQ 應用程式的設計考量』
當您決定應用程式如何利用可供您使用的平台及環境時，您需要決定如何使用 IBM MQ 所提供的特性。

第 603 頁的『撰寫佇列作業的程序化應用程式』
請利用這項資訊來瞭解如何撰寫佇列作業應用程式、連接佇列管理程式、發佈/訂閱，以及開啟和關閉物件。

第 765 頁的『撰寫用戶端程序化應用程式』
使用程序化語言在 IBM MQ 上撰寫用戶端應用程式所需的資訊。

第 68 頁的『使用 IBM MQ classes for JMS/Jakarta Messaging』
IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 是 IBM MQ 隨附的 Java 傳訊提供者。 除了實作 JMS 和 Jakarta Messaging 規格中定義的介面，這些傳訊提供者也會將兩組延伸新增至 Java 傳訊 API。

第 290 頁的『使用 IBM MQ classes for Java』
在 Java 環境中使用 IBM MQ 。 IBM MQ classes for Java 可讓 Java 應用程式以 IBM MQ 用戶端身分連接至 IBM MQ ，或直接連接至 IBM MQ 佇列管理程式。

第 464 頁的『開發 .NET 應用程式』
IBM MQ classes for .NET 容許 .NET 應用程式以 IBM MQ MQI client 身分連接至 IBM MQ ，或直接連接至 IBM MQ 伺服器。

第 438 頁的『開發 C++ 應用程式』
IBM MQ 提供相當於 IBM MQ 物件的 C++ 類別，以及相當於陣列資料類型的其他類別。 它提供許多無法透過 MQI 使用的特性。

第 837 頁的『建置程序化應用程式』
您可以使用數種程序化語言之一來撰寫 IBM MQ 應用程式，並在數個不同的平台上執行該應用程式。

相關工作

第 887 頁的『使用 IBM MQ 範例程序化程式』
這些範例程式是以程序化語言撰寫，並示範「訊息佇列介面 (MQI)」的一般用法。 不同平台上的 IBM MQ 程式。

第 1063 頁的『使用 IBM MQ 開發 Microsoft Windows Communication Foundation 應用程式』
IBM MQ 的 Microsoft Windows Communication Foundation (WCF) 自訂通道會在 WCF 用戶端與服務之間傳送及接收訊息。

保護安全

Multi 應用程式、應用程式名稱及應用程式實例

在開始設計及撰寫應用程式之前，請先熟悉應用程式、應用程式名稱及應用程式實例的基本概念。

應用程式

Multi

如果佇列管理程式的連線提供相同的 應用程式名稱，則會視為來自相同的 應用程式 。 應用程式名稱會顯示為 DISPLAY CONN (*) TYPE CONN 指令的 APPLTAG 屬性。

附註:

1. 對於使用早於 IBM MQ 9.1.2 的 IBM MQ client 版本的應用程式， IBM MQ client 會自動設定應用程式名稱。 其值取決於應用程式設計語言，以及應用程式執行所在的平台。 如需相關資訊，請參閱 PutAppl 名稱 。
2. 對於使用 IBM MQ client (位於 IBM MQ 9.1.2 或更新版本) 的 IBM MQ client 應用程式，可以將應用程式名稱設為特定值。 在大部分情況下，這不需要變更應用程式碼或重新編譯應用程式。 如需進一步資訊，請參閱第 45 頁的『在支援的程式設計語言中使用應用程式名稱』。

應用程式實例

Multi

連線進一步細分為 應用程式實例。 應用程式實例是一組密切相關的連線，為該應用程式提供一個「執行單元」。 一般而言，這是單一作業系統處理程序，可以有一些執行緒及相關聯的 IBM MQ 連線。

在 IBM MQ for Multiplatforms 上，應用程式實例與特定的 連線標籤相關聯。 當佇列管理程式可以看到新連線與現有應用程式實例相關時，它會自動將新連線與現有應用程式實例相關聯。

附註:

- 如果使用用戶端連線，這些處理程序可能會透過一或多個執行中通道連接至佇列管理程式。
- 在 JMS 應用程式中，應用程式實例會對映至特定的 JMS 連線及所有相關聯的 JMS 階段作業。

當使用統一叢集 自動應用程式平衡時，應用程式實例在 IBM MQ for Multiplatforms 上特別重要。 在 IBM MQ for Multiplatforms 平台上，您可以使用 DISPLAY APSTATUS 指令來檢視目前連接的應用程式實例。

在某些情況下，佇列管理程式無法正確地執行與應用程式實例關聯的連線，特別是:

- 如果在來自相同程序的共用交談上建立多個連線，則使用不同的應用程式名稱。
- 如果使用較舊層次用戶端程式庫。 例如， IBM MQ 9.1.2 及更早版本的 IBM MQ JMS 用戶端安裝。

在這些狀況下，如果應用程式未將自己定義為可重新連接，則允許這樣做，但部分應用程式實例分組可能不正確。 如果有任何連線宣告為 MQCNO_RECONNECT ，則這會對應用程式平衡產生明顯的負面影響; 因此，MQCONN 呼叫會因 MQCNO_RECONNECT_INCOMPATENT 而遭到拒絕。

相關概念
第 44 頁的『以支援的程式設計語言指定應用程式名稱』
在 IBM MQ 9.2.0 之前，您可以在 Java 或 JMS 用戶端應用程式上指定應用程式名稱。 從 IBM MQ 9.2.0 ，此特性已延伸至 IBM MQ for Multiplatforms 上的其他程式設計語言。

使用 MQI 的應用程式

IBM MQ 應用程式需要特定物件才能順利執行。

第 10 頁的圖 1 顯示應用程式從佇列中移除訊息、處理訊息，然後將部分結果傳送至相同佇列管理程式上的另一個佇列。

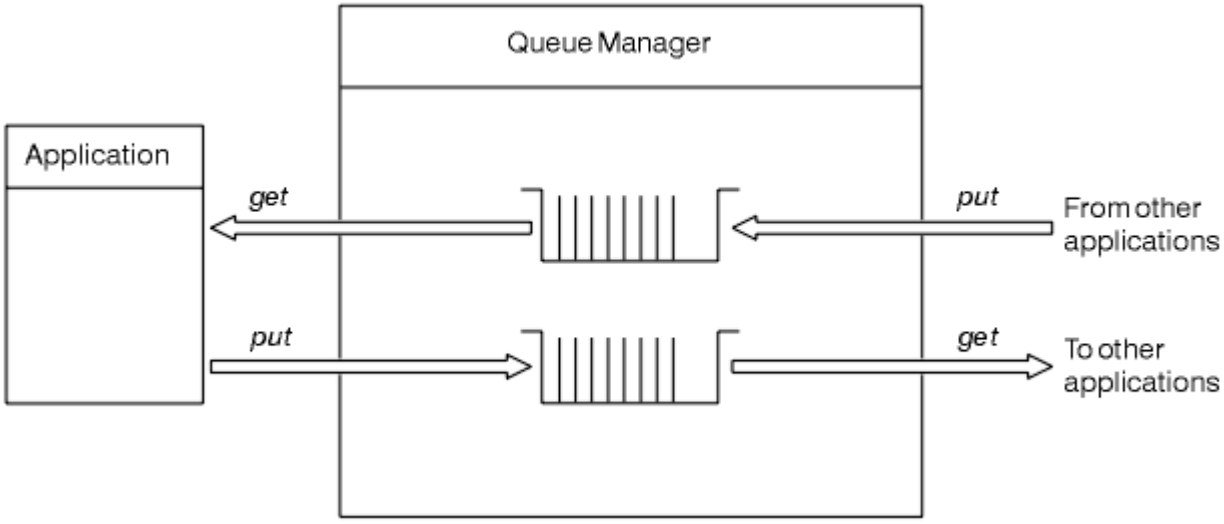


圖 *1:* 佇列、訊息及應用程式

應用程式可以將訊息放入本端或遠端佇列 (使用 MQPUT)，但只能直接從本端佇列 (使用 MQGET) 取得訊息。

必須滿足下列條件，才能執行此應用程式:

- 佇列管理程式必須存在且在執行中。
- 必須定義要從中移除訊息的第一個應用程式佇列。
- 也必須定義應用程式放置訊息的第二個佇列。
- 應用程式必須能夠連接至佇列管理程式。 若要執行此動作，它必須鏈結至 IBM MQ。 請參閱第 837 頁的『建置程序化應用程式』。
- 將訊息放置在第一個佇列上的應用程式也必須連接至佇列管理程式。 如果它們位於遠端，則也必須使用傳輸佇列及通道來設定它們。 第 10 頁的圖 1 中未顯示這個系統組件。

使用用戶端連線來連接多個 IBM MQ 佇列管理程式

可以配置用戶端連接的應用程式，以連接至多個佇列管理程式 (基於負載平衡或服務可用性原因)。

在 IBM MQ 用戶端中達成此目的的主要機制是使用用戶端通道定義表，請參閱 配置用戶端通道定義表或連線清單。

也可以使用外部負載平衡產品，或在 'stub' 中包裝可重新導向主機名稱或 IP 位址的 IBM MQ 連線程式碼，來達到類似的行為。

其中每一個技術都有一些限制，且可能或多或少適合特定應用程式需求。 下列各節雖然並非詳盡無遺，但說明您應該考量的特定層面，以及這些不同方法對這些層面的影響。

IBM MQ 統一叢集，請參閱 關於統一叢集，提供功能強大的機制，可讓應用程式在多個佇列管理程式之間進行水平調整，而這些佇列管理程式是以 CCDT 的基本機制來建置，以提供多個目的地。 統一叢集可以提供超出可能使用外部負載平衡器而不知道基礎 IBM MQ 通訊協定的功能，並避免下面討論的部分問題，因此請考量優先使用統一叢集，而不是其他技術 (如果適用的話)。

小心: 請小心使用使用 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 的應用程式 (包括使用其中一個「IBM MQ 資源配接器」的應用程式)，這些應用程式會使用負載平衡技術連接至佇列管理程式。 如果您遇到問題，請重建那些問題，而不嘗試使用負載平衡。

涉及多個問題，所有這些問題都意味著這類連線在最好的情況下是有問題的，而且在最壞的情況下完全不可靠:

- 當連接任何使用任何形式的負載平衡建立多個佇列管理程式連線的應用程式時，需要特別小心。 這包括使用 IBM MQ Classes for JMS/Jakarta Messaging 的所有應用程式，因為這些會在一般情況下建立多個 IBM MQ 連線。 如果使用外部負載平衡器或自訂程式碼 Stub，則必須隨時將連線從相同的應用程式實例遞送至相同的佇列管理程式。

- 使用 XA 交易管理或 JTA (Java 交易 API) 取決於一致地連接至相同佇列管理程式的能力-實際上，這在任何形式的負載平衡中都不太可能實際。
- -統一叢集管理需要能夠指示用戶端重新連接至特定佇列管理程式而不受干擾。 不建議嘗試結合外部負載平衡與使用「統一叢集」

您應該使用 IBM MQ 統一叢集功能，在多個佇列管理程式之間實現應用程式的水平調整，而不是外部負載平衡技術。 如需統一叢集的相關資訊，包括如何建立及使用統一叢集，請參閱 配置統一叢集及下列主題。

本資訊中使用的術語

CCDT-多重 QMGR

表示包含具有相同群組 (即佇列管理程式名稱用戶端連線 (QMNAME CLNTCONN) 屬性) 之多個用戶端連線 (CLNTCONN) 通道的 CCDT 檔案，其中不同的 CLNTCONN 項目會解析至不同的佇列管理程式。

這不同於包含多個 CLNTCONN 項目的 CCDT 檔案，這些項目只是相同多重實例佇列管理程式的不同 IP 位址或主機名稱，這是您可以選擇與程式碼 Stub 結合的方法。

如果您選擇 CCDT 多重佇列管理程式方法，則需要選擇是設定項目優先順序，還是具有隨機化工作量管理 (WLM):

設定威脅

搭配使用多個按字母順序排序的項目與 CLNTWGHT (1) 及 AFFINITY (PREFERRED) 屬性，以記住最後一個良好連線。

隨機化

請使用 CLNTWGHT (1) 和 AFFINITY (NONE) 屬性。 您可以透過調整 CLNTWGHT ，在不同調整大小的 IBM MQ 伺服器之間調整 WLM 加權

註: 您應該避免在通道之間的 CLNTWGHT 中有較大的差異。

負載平衡器

表示網路應用裝置，其「虛擬 IP 位址 (VIP)」已配置多個 IBM MQ 佇列管理程式之 TCP/IP 接聽器的埠監視。 如何在網路應用裝置中配置 VIP 取決於您正在使用的網路應用裝置。

下列選項僅與傳送訊息或起始同步要求及回覆傳訊的應用程式相關。 處理那些訊息及要求的應用程式考量，例如，接聽器完全分開，並在「將訊息接聽器連接至佇列」中詳細討論。

連接至單一佇列管理程式的現有應用程式所需的程式碼變更規模

CONNAME 清單、CCDT multi-QMGR 及負載平衡器

MQCONN ("QMNAME") 至 MQCONN ("*QMNAME")

佇列管理程式名稱可能位於 Java Platform, Enterprise Edition (Java EE) 應用程式的「 Java 命名和目錄介面 (JNDI)」配置中。 否則，這需要變更一個字元碼。

程式碼 Stub

將現有的 JMS 或 MQI 連線邏輯取代為程式碼 Stub。

支援不同的 WLM 策略

CONNAME 清單

僅已設定優先順序。

這可能會對程式碼產生負面影響。

CCDT 多重 QMGR

已設定優先順序或隨機。

這可能不會對程式碼產生任何影響。

負載平衡器

任何，包括所有訊息的每一個連線。

這可能會對程式碼產生正面影響。

程式碼 Stub
任何，包括所有訊息的每一個訊息。

這可能會對程式碼產生正面影響。

主要佇列管理程式無法使用時的效能額外負擔

CONNAME 清單
一律在清單中先嘗試。

這可能會對程式碼產生負面影響。

CCDT 多重 QMGR
記住前次良好的連線。

這可能會對程式碼產生正面影響。

負載平衡器
埠監視可避免不正確的佇列管理程式。

這可能會對程式碼產生正面影響。

程式碼 Stub
可以記住前次良好的連線，並以智慧方式重試。

這可能會對程式碼產生正面影響。

XA 交易支援

CONNAME 清單、CCDT multi-QMGR 及負載平衡器
交易管理程式需要儲存重新連接至相同佇列管理程式資源的回復資訊。

解析為不同佇列管理程式的 MQCONN 呼叫通常會使此失效。 例如，在 Java EE 中，當使用 XA 時，單一 Connection Factory 應該會解析成單一佇列管理程式。

這可能會對程式碼產生負面影響。

程式碼 Stub
程式碼 Stub 可以符合交易管理程式的 XA 需求，例如多個 Connection Factory。

這可能會對程式碼產生正面影響。

管理彈性可隱藏應用程式的基礎架構變更

CONNAME 清單
僅限 DNS。

這可能會對程式碼產生負面影響。

CCDT 多重 QMGR
DNS 及共用檔案系統，或共用檔案系統，或 CCDT 檔案推送。

這可能不會對程式碼產生任何影響。

負載平衡器
動態虛擬 IP 位址 (VIP)。

這可能會對程式碼產生正面影響。

程式碼 Stub
DNS 或單一佇列管理程式 CCDT 項目。

這可能不會對程式碼產生任何影響。

避免因計劃性維護而中斷

您需要考量並規劃另一個狀況，即如何避免在計劃的佇列管理程式維護期間毀壞應用程式，例如一般使用者可見的錯誤及逾時。 避免毀壞的最佳方法是在佇列管理程式停止之前將所有工作從佇列管理程式中移除。

考量要求及回覆實務範例。 您想要完成所有進行中要求，並由應用程式處理回覆，但不想要將任何其他工作提交至系統。 只是靜止佇列管理程式無法滿足此需求，因為編碼完整的應用程式在接收進行中要求的回覆訊息之前，會收到回覆碼 RC2161 MQRC_Q_MGR_QUIESCING 異常狀況。

您可以在用來提交工作的要求佇列上設定 PUT (DISABLED) ，同時保留回覆佇列 PUT (ENABLED) 和 GET (ENABLED)。 如此一來，您可以監視要求、傳輸及回覆佇列的深度。 一旦它們都穩定，即進行中要求完成或逾時，您可以停止佇列管理程式。

不過，在發出要求的應用程式中需要良好的編碼，才能處理 PUT (DISABLED) 要求佇列，這會在嘗試傳送訊息時導致回覆碼 RC2051 MQRC_PUT_INHIBITED 錯誤。

請注意，在建立與 IBM MQ 的連線或開啟要求佇列時，不會發生異常狀況。 只有在嘗試使用 MQPUT 呼叫實際傳送訊息時，才會發生異常狀況。

針對要求及回覆實務範例建置包含此錯誤處理邏輯的程式碼 Stub ，並要求您的應用程式團隊在未來使用此類程式碼 Stub ，可協助您開發具有一致行為的應用程式。

V 9.3.0 開發彈性且可擴充的用戶端應用程式

為了容錯及可調整性，將支援連線選項的用戶端應用程式部署至統一叢集，可讓應用程式實例在佇列管理程式之間重新平衡。

如需統一叢集的概觀，請參閱 關於統一叢集 。

理想情況下，此重新平衡對應用程式是不可見的，但只有特定類型的應用程式才適合此類型的部署，而且在應用程式設計中可能需要一些考量。

這些考量分為兩個主要種類:

- 少數 錯誤路徑 可能已存在可重新連接的應用程式，但在部署至統一叢集時變得更有可能。 例如，在重新連接之後，會取消任何進行中的工作單元，並重設瀏覽游標。 對於您在其現行環境中可重新連接的應用程式而言，這些可能是罕見的事件，因此應用程式碼無法盡可能徹底處理。 檢閱應用程式邏輯，以確保對這類狀況有適當的處理，有助於避免發生非預期的問題。
- 對特定佇列管理程式的 親緣性 。 如果您知道應用程式必須一律連接回相同或特定佇列管理程式，則應該將應用程式配置成重新連接至該佇列管理程式，或不啟用其與該佇列管理程式的連線。 不過，這些親緣性可能是暫時的，例如等待回應訊息。 下一節將討論影響平衡演算法以說明應用程式碼中的這些親緣性。 如需這些選項的詳細資料，以及如何透過配置而非應用程式碼來達成類似方法，請參閱 在統一叢集中影響應用程式重新平衡。

影響 MQI 中的重新連線選項

如需 MQCNO_RECONNECT 的相關資訊，請參閱 重新連接選項 。

如果您知道應用程式必須一律連接回相同或特定佇列管理程式，則應該將它配置為 MQCNO_RECONNECT_Q_MGR 或 MQCNO_RECONNECT_DISABLED。

影響 MQI 中的平衡演算法

不過，您可能想要控制或影響重新平衡行為，以符合特定應用程式類型的需求; 例如，將進行中交易的中斷減至最少，或確保要求者應用程式在移動之前收到其回應。

在 在統一叢集中影響應用程式重新平衡中採用並討論某些預設想要的行為。 您也可以透過該主題所討論的 client.ini 檔，來影響特定應用程式在配置或部署期間的行為。

在其他情況下，讓平衡行為和需求成為應用程式邏輯的一部分可能更有意義。 在這些情況下，以稱為 MQBNO (平衡選項) 的結構連接至 MQCONNX 呼叫上的佇列管理程式時，可以將應用程式的相同相關性質提供給 IBM MQ 。

如果您提供 MQBNO 結構，則必須提供 IBM MQ 所需的所有資訊，以決定應該如何及何時要求應用程式重新連接至不同的佇列管理程式。

您必須提供:

- 應用程式的 **Type**

- 不論狀態為何，都會重新平衡實例的 **Timeout**
- 任何特殊 **BalanceOptions**

不過，如果需要的話，您可以將逾時保留為 MQBNO_TIMEOUT_DEFAULT。 在此情況下，逾時會解析為 client.ini 檔案、應用程式或廣域段落中的任何值 (如果有提供的話)，並使其失敗，基本預設值為 10 秒。

如需此結構格式的詳細資料，請參閱 MQBNO 。

若為 .NET 應用程式，請參閱 在 .NET 中影響應用程式重新平衡，以取得進一步資訊。

物件導向應用程式

IBM MQ 提供 JMS、 Java、C++ 及 .NET 的支援。 這些語言及架構使用「IBM MQ 物件模型」，其提供的類別提供與 IBM MQ 呼叫及結構相同的功能。

使用「IBM MQ 物件模型」的部分語言及架構提供當您搭配使用程序化語言與訊息佇列介面 (MQI) 時無法使用的其他功能。

如需此模型所提供類別、方法及內容的詳細資料，請參閱 第 14 頁的『IBM MQ 物件模型』。

JMS

IBM MQ 提供實作 Jakarta Messaging 3.0 和 Java Message Service 2.0 規格的類別。 如需 IBM MQ classes for JMS 的詳細資料，請參閱 使用 IBM MQ classes for JMS。 如需 IBM MQ classes for Java 與 IBM MQ classes for JMS 之間的差異相關資訊，以協助您決定要使用的項目，請參閱 第 67 頁的『開發 JMS/Jakarta Messaging 和 Java 應用程式』。

IBM MQ Message Service Client (XMS) for C/C++ 和 IBM MQ Message Service Client (XMS) for .NET 提供稱為 XMS 的應用程式設計介面 (API)，其具有與 Java Message Service (JMS) 相同的介面集 API。 如需相關資訊，請參閱 第 516 頁的『開發 XMS .NET 應用程式』。

Java

如需在 Java 中使用 IBM MQ 物件模型撰寫程式的相關資訊，請參閱 使用 IBM MQ classes for Java 。

Stabilized IBM 將不會進一步加強 IBM MQ classes for Java，而且其功能層次將穩定為 IBM MQ 8.0 隨附的層次。 如需 IBM MQ classes for Java 與 IBM MQ classes for JMS 之間差異的相關資訊，以協助您決定要使用的項目，請參閱 第 67 頁的『開發 JMS/Jakarta Messaging 和 Java 應用程式』。

C++

IBM MQ 提供相當於 IBM MQ 物件的 C++ 類別，以及相當於陣列資料類型的其他類別。 它提供許多無法透過 MQI 使用的特性。 如需使用 IBM MQ Object Model in C++ 的相關資訊，請參閱 使用 C++ 。 Message Service Client for C/C++ 及 .NET 提供稱為 XMS 的應用程式設計介面 (API)，其具有與 Java Message Service (JMS) 相同的介面集。 API。

.NET

如需使用 IBM MQ .NET 類別編碼 .NET 程式的相關資訊，請參閱 開發 .NET 應用程式 。 Message Service Client for C/C++ 及 .NET 提供稱為 XMS 的應用程式設計介面 (API)，其具有與 Java Message Service (JMS) 相同的介面集 API。

相關概念

第 600 頁的『使用 IBM MQ 開發 MQI 應用程式』
IBM MQ 提供 C、Visual Basic、COBOL、Assembler、RPG、 pTAL 及 PL/I 的支援。 這些程序化語言使用訊息佇列介面 (MQI) 來存取訊息佇列作業服務。

技術概觀

第 6 頁的『應用程式開發概念』
您可以使用選擇的程序化或物件導向語言來撰寫 IBM MQ 應用程式。 在開始設計及撰寫 IBM MQ 應用程式之前，請先熟悉基本 IBM MQ 概念。

相關參考

開發應用程式參照

IBM MQ 物件模型

「IBM MQ 物件模型」由類別、方法及內容組成。

IBM MQ 物件模型包含:

- 類別 代表熟悉的 IBM MQ 概念，例如佇列管理程式、佇列及訊息。
- 每一個類別上對應於 MQI 呼叫的 方法 。
- 每個類別上對應於 IBM MQ 物件屬性的 內容 。

使用「IBM MQ 物件模型」建立 IBM MQ 應用程式時，您會在應用程式中建立這些類別的實例。 物件導向程式設計中的類別實例稱為 物件。 建立物件之後，您可以透過檢查或設定物件內容的值 (相當於發出 MQINQ 或 MQSET 呼叫) ， 以及對物件進行方法呼叫 (相當於發出其他 MQI 呼叫) ， 來與物件互動。

類別

「IBM MQ 物件模型」提供下列基本類別集。

模型的實際實作在不同的受支援物件導向環境之間略有不同。

MQQueueManager
MQQueueManager 類別的物件代表與佇列管理程式的連線。 它具有 Connect ()、Disconnect ()、Commit () 及 Backout () 的方法 (相當於 MQCONN 或 MQCONNX、MQDISC、MQCMIT 及 MQBACK)。 它具有對應於佇列管理程式屬性的內容。 如果尚未連接，則存取佇列管理程式屬性內容會隱含地連接至佇列管理程式。 毀損 MQQueueManager 物件會隱含地切斷與佇列管理程式的連線。

MQQUEUE
MQQueue 類別的物件代表佇列。 它具有在佇列中來回放置 () 及取得 () 訊息的方法 (相當於 MQPUT 及 MQGET)。 它具有對應於佇列屬性的內容。 存取佇列屬性內容，或發出 Put () 或 Get () 方法呼叫，會隱含地開啟佇列 (相當於 MQOPEN)。 毀損 MQQueue 物件會隱含地關閉佇列 (相當於 MQCLOSE)。

MQTopic
MQTopic 類別的物件代表主題。 它具有與主題 (相等於 MQPUT 及 MQGET) 來回的 Put () (發佈) 及 Get () (接收或訂閱) 訊息的方法。 它具有對應於主題屬性的內容。 MQTopic 物件只能存取發佈或訂閱，不能同時存取兩者。 當用於接收訊息時，可以使用未受管理或受管理的訂閱來建立 MQTopic 物件，並作為可延續或不可延續訂閱者-針對這些不同的實務範例提供多個超載建構子。

MQMessage
MQMessage 類別的物件代表要放置在佇列上或從佇列取得的訊息。 它包含緩衝區，並封裝應用程式資料及 MQMD。 它具有對應於 MQMD 欄位的內容，以及可讓您在緩衝區來回寫入及讀取不同類型 (例如，字串、長整數、短整數、單一位元組) 的使用者資料的方法。

MQPutMessage 選項
MQPutMessageOptions 類別的物件代表 MQPMO 結構。 它具有對應於 MQPMO 欄位的內容。

MQGetMessage 選項
MQGetMessageOptions 類別的物件代表 MQGMO 結構。 它具有對應於 MQGMO 欄位的內容。

MQProcess
MQProcess 類別的物件代表程序定義 (與觸發一起使用)。 它具有代表程序定義屬性的內容。

Multi **MQDistributionList**
MQDistributionList 類別的物件代表一個配送清單 (用來以單一 MQPUT 傳送多個訊息)。 它包含 MQDistributionListItem 物件的清單。

Multi **MQDistributionList 項目**
MQDistributionList 項目類別的物件代表單一配送清單目的地。 它封裝 MQOR、MQRR 及 MQPMR 結構，並具有對應於這些結構欄位的內容。

物件參照

在使用 MQI 的 IBM MQ 程式中， IBM MQ 會將連線控點及物件控點傳回給程式。

在後續的 IBM MQ 呼叫中，這些控點必須以參數形式傳遞。 使用「IBM MQ 物件模型」， 會對應用程式隱藏這些控點。 相反地，從類別建立物件會導致將物件參照傳回應用程式。 當對物件進行方法呼叫和內容存取時，會使用這個物件參照。

回覆碼

發出方法呼叫或設定內容值會導致設定回覆碼。

這些回覆碼是完成碼和原因碼，它們本身是物件的內容。 完成碼及原因碼的值與為 MQI 定義的值相同，且具有特定於物件導向環境的一些額外值。

IBM MQ 訊息

IBM MQ 訊息由訊息內容及應用程式資料組成。 訊息佇列訊息描述子 (MQMD) 包含在傳送端與接收端應用程式之間傳送訊息時，伴隨應用程式資料的控制資訊。

訊息組件

IBM MQ 訊息由兩個部分組成:

- 訊息內容
- 應用程式資料

第 16 頁的圖 2 代表訊息，並顯示它在邏輯上如何劃分為訊息內容及應用程式資料。

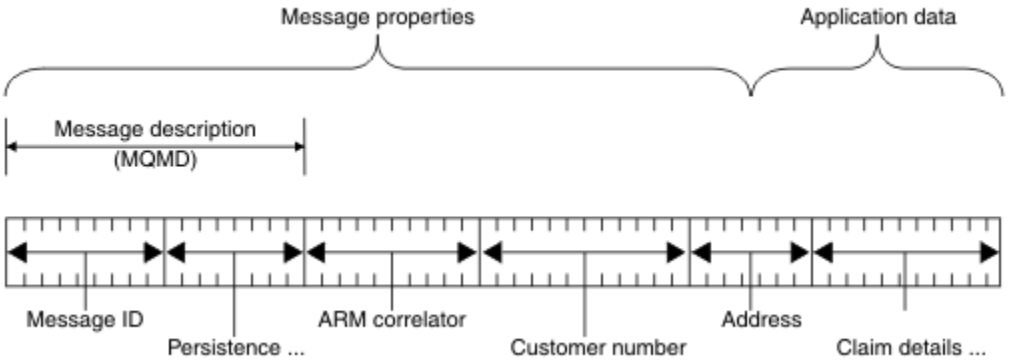


圖 *2:* 訊息的表示法

除非對 IBM MQ 訊息執行資料轉換，否則佇列管理程式不會變更該訊息中所載的應用程式資料。 此外，IBM MQ 不會對此資料的內容設定任何限制。 每一則訊息中的資料長度不能超出佇列及佇列管理程式的 **MaxMsgLength** 屬性值。

ALW 在 AIX, Linux, and Windows 上，佇列管理程式及佇列的 *MaxMsgLength* 屬性預設為 4 MB (4 194 304 位元組)，必要的話，最多可以變更為 100 MB (104 857 600 位元組)。

IBM i 在 IBM i 上，佇列管理程式及佇列的 *MaxMsgLength* 屬性預設為 4 MB (4 194 304 位元組)，必要的話，最多可以變更為 100 MB (104 857 600 位元組)。 如果您想要在 IBM i 上使用大於 15 MB 的 IBM MQ 訊息，請參閱 第 843 頁的『在 IBM i 上建置程序化應用程式』。

z/OS 在 z/OS 上，佇列管理程式的 **MaxMsgLength** 屬性固定為 100 MB，且佇列的 **MaxMsgLength** 屬性預設為 4 MB (4 194 304 個位元組)，必要的話，您最多可以變更為 100 MB。

在某些情況下，讓您的訊息比 **MaxMsgLength** 屬性值略短。 如需相關資訊，請參閱 第 634 頁的『訊息中的資料』。

當您使用 MQPUT 或 MQPUT1 MQI 呼叫時，會建立訊息。 作為這些呼叫的輸入，您可以提供控制資訊 (例如訊息的優先順序及回覆佇列的名稱) 和您的資料，然後呼叫會將訊息放置在佇列上。 如需這些呼叫的相關資訊，請參閱 MQPUT 和 MQPUT1。

訊息描述子

您可以使用 MQMD 結構 (定義 訊息描述子) 來存取訊息控制資訊。

如需 MQMD 結構的完整說明，請參閱 MQMD-訊息描述子。

如需如何使用 MQMD 內包含訊息來源相關資訊之欄位的說明，請參閱 第 40 頁的『訊息環境定義』。

訊息描述子有不同版本。 訊息分組及分段的其他資訊 (請參閱 第 37 頁的『訊息群組』) 在訊息描述子 (或 MQMDE) 第 2 版中提供。 這與第 1 版訊息描述子相同，但有額外欄位。 這些欄位在 MQMDE-訊息描述子延伸中有說明。

訊息類型

IBM MQ 定義四種類型的訊息。

這四個訊息如下:

- 資料包
- 要求訊息
- 回覆訊息
- 報告訊息
  - 報告訊息類型
  - 報告訊息選項

應用程式可以使用前三種類型的訊息，在它們之間傳遞資訊。 第四種類型 (報告) 是供應用程式及佇列管理程式用來報告事件 (例如發生錯誤) 的相關資訊。

每一種類型的訊息都由 MQMT_ * 值識別。 您也可以定義自己的訊息類型。 如需您可以使用的值範圍，請參閱 MsgType。

資料包

當您不需要接收訊息 (即從佇列取得訊息) 的應用程式回覆時，請使用 資料封包 。

可能使用資料包的應用程式範例是在機場休息室內顯示航班資訊的應用程式。 訊息可能包含整個航班資訊畫面的資料。 這類應用程式不太可能要求訊息的確認通知，因為如果訊息未遞送，可能並不重要。 應用程式會在短時間之後傳送更新訊息。

要求訊息

當您想要來自接收訊息之應用程式的回覆時，請使用 要求訊息 。

可以使用要求訊息的應用程式範例是顯示支票帳戶餘額的應用程式。 要求訊息可能包含帳戶號碼，而回覆訊息會包含帳戶餘額。

如果您要將回覆訊息與要求訊息鏈結在一起，有兩個選項:

- 讓處理要求訊息的應用程式負責確保將資訊放入與要求訊息相關的回覆訊息中。
- 使用要求訊息之訊息描述子中的報告欄位，以指定回覆訊息的 *MsgId* 及 *CorrelId* 欄位的內容:
  - 您可以要求將原始訊息的 *MsgId* 或 *CorrelId* 複製到回覆訊息的 *CorrelId* 欄位 (預設動作是複製 *MsgId*)。
  - 您可以要求針對回覆訊息產生新的 *MsgId* ，或將原始訊息的 *MsgId* 複製到回覆訊息的 *MsgId* 欄位 (預設動作是產生新的訊息 ID)。

回覆訊息

當您回覆另一則訊息時，請使用 回覆訊息 。

當您建立回覆訊息時，請遵循您要回覆之訊息的訊息描述子中所設定的任何選項。 報告選項指定訊息 ID (*MsgId*) 及相關性 ID (*CorrelId*) 欄位的內容。 這些欄位可讓接收回覆的應用程式將回覆與其原始要求產生關聯。

報告訊息

報告訊息 會通知應用程式有關事件，例如在處理訊息時發生錯誤。

它們可以由下列產生:

- 佇列管理程式,
- 訊息通道代理程式 (例如，如果無法遞送訊息)，或
- 應用程式 (例如，如果無法使用訊息中的資料)。

報告訊息可以隨時產生，當您的應用程式不預期它們時，可能會抵達佇列。

報告訊息的類型

當您將訊息放置在佇列上時，您可以選取接收:

- 異常狀況報告訊息。 這會傳送以回應已設定異常狀況旗標的訊息。 它是由訊息通道代理程式 (MCA) 或應用程式所產生。
- 期限報告訊息。 這指出應用程式嘗試擷取已達到到期臨界值的訊息; 訊息會標示為捨棄。 此類型的報告由佇列管理程式產生。
- 確認到達 *(COA)* 報告訊息。 這指出訊息已到達其目標佇列。 它是由佇列管理程式所產生。
- 確認遞送 *(COD)* 報告訊息。 這指出接收端應用程式已擷取訊息。 它是由佇列管理程式所產生。
- 正面動作通知 *(PAN)* 報告訊息。 這指出已順利處理要求 (亦即，已順利執行訊息中所要求的動作)。 此類型的報告由應用程式產生。
- 負面動作通知 *(NAN)* 報告訊息。 這指出未順利處理要求 (亦即，訊息中所要求的動作未順利執行)。 此類型的報告由應用程式產生。

註: 每一種類型的報告訊息都包含下列其中一項:

- 整個原始訊息
- 原始訊息中資料的前 100 個位元組
- 沒有來自原始訊息的資料

將訊息放入佇列時，您可以要求多種類型的報告訊息。 如果您選取遞送確認報告訊息和異常狀況報告訊息選項，如果無法遞送訊息，您會收到異常狀況報告訊息。 不過，如果您只選取遞送確認報告訊息選項，且無法遞送訊息，則不會收到異常狀況報告訊息。

當符合產生特定訊息的準則時，您所要求的報告訊息是您收到的唯一報告訊息。

報告訊息選項

您可以在發生異常狀況之後 捨棄 訊息。 如果您選取捨棄選項，且已要求異常狀況報告訊息，則報告訊息會移至 *ReplyToQ* 及 *ReplyToQMgr*，並捨棄原始訊息。

註: 其好處是您可以減少進入無法傳送郵件之佇列的訊息數。 不過，它確實表示您的應用程式除非只傳送資料包訊息，否則必須處理傳回的訊息。 當產生異常狀況報告訊息時，它會繼承原始訊息的持續性。

如果無法遞送報告訊息 (例如，如果佇列已滿)，則會將報告訊息放置在無法傳送郵件的佇列上。

如果您想要接收報告訊息，請在 *ReplyToQ* 欄位中指定回覆目的地佇列的名稱; 否則原始訊息的 MQPUT 或 MQPUT1 會因 MQRC_MISSING_REPLY_TO_Q 而失敗。

您可以在訊息的訊息描述子 (MQMD) 中使用其他報告選項，來指定針對訊息所建立之任何報告訊息的 *MsgId* 及 *CorrelId* 欄位內容:

- 您可以要求將原始訊息的 *MsgId* 或 *CorrelId* 複製到報告訊息的 *CorrelId* 欄位。 預設動作是複製訊息 ID。 請使用 MQRO_COPY_MSG_ID_TO_CORRELID，因為它可讓訊息傳送者將回覆或報告訊息與原始訊息產生關聯。 回覆或報告訊息的相關性 ID 與原始訊息的訊息 ID 相同。
- 您可以要求為報告訊息產生新的 *MsgId*，或將原始訊息的 *MsgId* 複製到報告訊息的 *MsgId* 欄位。 預設動作是產生新的訊息 ID。 請使用 MQRO_NEW_MSG_ID，因為它可確保系統中的每一則訊息都有不同的訊息 ID，並且可以與系統中的所有其他訊息明確區分。
- 特殊化應用程式可能需要使用 MQRO_PASS_MSG_ID 或 MQRO_PASS_CORREL_ID。 不過，您需要設計從佇列讀取訊息的應用程式，以確保在佇列包含多個具有相同訊息 ID 的訊息時正確運作。

  伺服器應用程式必須檢查要求訊息中這些旗標的設定，並適當地設定回覆或報告訊息中的 *MsgId* 和 *CorrelId* 欄位。

作為要求者應用程式與伺服器應用程式之間的媒介的應用程式不需要檢查這些旗標的設定。 這是因為這些應用程式通常需要將訊息轉遞至伺服器應用程式，且 *MsgId*、 *CorrelId* 及 *Report* 欄位保持不變。 這可讓伺服器應用程式從回覆訊息的 *CorrelId* 欄位中的原始訊息複製 *MsgId* 。

當產生訊息的相關報告時，伺服器應用程式必須進行測試，以查看是否已設定任何這些選項。

如需如何使用報告訊息的相關資訊，請參閱 報告。

為了指出報告的本質，佇列管理程式會使用一系列的回饋碼。 它們會將這些代碼放置在報告訊息之訊息描述子的 *Feedback* 欄位中。 佇列管理程式也可以在 *Feedback* 欄位中傳回 MQI 原因碼。 IBM MQ 定義供應用程式使用的回饋碼範圍。

如需意見和原因碼的相關資訊，請參閱 意見。

可以使用回饋碼的程式範例是監視處理佇列之其他程式的工作量。 如果有多個程式實例負責處理佇列，且到達佇列的訊息數不再是合理的，則這類程式可以將報告訊息 (具有回饋碼 MQFB_QUIT) 傳送至其中一個服務程式，以指出程式應該終止其活動。 (監視程式可以使用 MQINQ 呼叫來找出負責處理佇列的程式數目。)

Multi 報告及分段訊息

在 IBM MQ for z/OS 上不受支援。

如果訊息已分段，且您要求產生報告，則您所收到的報告可能比未分段訊息的報告還多。

如需分段訊息的說明，請參閱 第 662 頁的『訊息分段』。

針對 IBM MQ 所產生的報告

如果您將訊息分段或容許佇列管理程式這樣做，則只有一個案例可預期接收整個訊息的單一報告。 這是當您只要求 COD 報告，且已在取得應用程式上指定 MQGMO_COMPLETE_MSG 時。

在其他情況下，您必須準備好處理數個報告; 通常每一個區段一個報告。

註: 如果您將訊息分段，且只需要傳回原始訊息資料的前 100 個位元組，請變更報告選項的設定，以針對偏移為 100 或以上的區段要求沒有資料的報告。 如果您不這麼做，且離開設定，讓每一個區段要求 100 個位元組的資料，並以單一 MQGET 指定 MQGMO_COMPLETE_MSG 來擷取報告訊息，則報告會在每一個適當偏移處組合成包含 100 個位元組讀取資料的大型訊息。 如果發生這種情況，您需要較大的緩衝區，或您需要指定 MQGMO_ACCEPT_TRUNCATED_MSG。

針對應用程式所產生的報告

如果您的應用程式產生報告，請一律將原始訊息資料開頭的 IBM MQ 標頭複製到報告訊息資料。

然後將無、100 位元組或所有原始訊息資料 (或您通常包括的任何其他數量) 新增至報告訊息資料。

您可以透過查看連續的「格式」名稱 (從 MQMD 開始並透過任何呈現的標頭繼續進行)，來辨識必須複製的 IBM MQ 標頭。 下列 `Format` 名稱指出這些 IBM MQ 標頭:

- MQMDE
- MQDLH
- MQXQH
- MQIIH
- MQH*

MQH* 表示任何以 MQH 字元開頭的名稱。

`Format` 名稱出現在 MQDLH 和 MQXQH 的特定位置，但其他 IBM MQ 標頭則出現在相同位置。 標頭長度包含在欄位中，該欄位也會出現在 MQMDE、 MQIMS 及所有 MQH* 標頭的相同位置。

如果您是使用第 1 版 MQMD，且正在報告區段、群組中的訊息或容許分段的訊息，則報告資料必須以 MQMDE 開頭。 將 *OriginalLength* 欄位設為原始訊息資料的長度，但不包括您找到的任何 IBM MQ 標頭的長度。

擷取報告

如果您要求 COA 或 COD 報告，您可以使用 MQGMO_COMPLETE_MSG 來要求重新組合它們。

當佇列上有足夠的報告訊息 (單一類型，例如 COA，且具有相同的 *GroupId*) 代表一個完整原始訊息時，即滿足 MQGET 與 MQGMO_COMPLETE_MSG。 即使報告訊息本身不包含完整原始資料，也是如此; 即使資料本身不存在，每則報告訊息中的 *OriginalLength* 欄位也會提供該報告訊息所代表的原始資料長度。

即使佇列上存在數個不同的報告類型 (例如， COA 及 COD)，您也可以使用此技術，因為只有在 MQGET 與 MQGMO_COMPLETE_MSG 具有相同的 *Feedback* 程式碼時，它們才會重新組合報告訊息。 不過，您通常無法將此技術用於異常狀況報告，因為一般而言，這些具有不同的 *Feedback* 代碼。

您可以使用此技術來取得整個訊息已到達的正面指示。 不過，在大部分情況下，您需要滿足部分區段到達而其他區段可能產生異常狀況 (或到期，如果您允許的話) 的可能性。 在此情況下，您無法使用 MQGMO_COMPLETE_MSG，因為一般而言，您可能會針對不同的區段取得不同的 *Feedback* 程式碼，而且您可能會針對一個區段取得多個報告。 不過，您可以使用 MQGMO_ALL_SEGMENTS_AVAILABLE。

若要容許這樣做，您可能需要在報告到達時擷取報告，並在應用程式中建立原始訊息所發生的情況的圖片。 您可以使用報告訊息中的 *GroupId* 欄位，將報告與原始訊息的 *GroupId* 產生關聯，並使用 *Feedback* 欄位來識別每一個報告訊息的類型。 您執行此動作的方式視您的應用程式需求而定。

一種辦法如下:

- 要求 COD 報告和異常狀況報告。
- 在特定時間之後，請使用 MQGMO_COMPLETE_MSG 檢查是否收到一組完整的 COD 報告。 如果是這樣，您的應用程式會知道已處理整個訊息。
- 如果沒有，且存在與此訊息相關的異常狀況報告，請處理未分段訊息的問題，但確保您在某個點清除孤立區段。
- 如果有區段沒有任何類型的報告，則原始區段 (或報告) 可能正在等待通道重新連接，或網路在某個點可能超載。 如果完全未收到異常狀況報告 (或您認為您所擁有的報告可能只是暫時的)，則您可能決定讓應用程式稍等一下。

  與之前一樣，這與您在處理未分段訊息時的考量類似，但您也必須考量清除孤立區段的可能性。

如果原始訊息不嚴重 (例如，如果它是查詢，或稍後可以重複的訊息)，請設定到期時間，以確保移除孤立區段。

前一層次佇列管理程式

當報告由支援分段的佇列管理程式產生，但在不支援分段的佇列管理程式上接收到報告時，除了訊息中的零、100 位元組或所有原始資料之外， MQMDE 結構 (識別報告所代表的 *Offset* 和 *OriginalLength*) 一律包括在報告資料中。

不過，如果訊息區段通過不支援分段的佇列管理程式，如果在該處產生報告，則會將原始訊息中的 MQMDE 結構純粹視為資料。 因此，如果已要求零位元組的原始資料，則不會將它併入報告資料中。 如果沒有 MQMDE，報告訊息可能沒有用。

如果訊息可能通過前一層次佇列管理程式，請要求報告中至少 100 個位元組的資料。

訊息控制資訊及訊息資料的格式

佇列管理程式只對訊息內控制資訊的格式感興趣，而處理訊息的應用程式則對控制資訊及資料的格式感興趣。

訊息控制資訊的格式

訊息描述子的字串欄位中的控制資訊必須在佇列管理程式所使用的字集中。

佇列管理程式物件的 **CodedCharSetId** 屬性定義此字集。 控制資訊必須在此字集中，因為當應用程式將訊息從一個佇列管理程式傳遞至另一個佇列管理程式時，傳輸訊息的訊息通道代理程式會使用此屬性值來決定要執行的資料轉換。

訊息資料的格式

您可以指定下列任何項目:

- 應用程式資料的格式
- 字元資料的字集
- 數值資料的格式

若要執行此動作，請使用下列欄位:

Format
這會向訊息接收端指出訊息中應用程式資料的格式。

當佇列管理程式建立訊息時，在某些情況下，會使用 *Format* 欄位來識別該訊息的格式。 例如，當佇列管理程式無法遞送訊息時，它會將訊息放在無法傳送的郵件 (未遞送的訊息) 佇列中。 它會將標頭 (包含更多控制項資訊) 新增至訊息，並變更 *Format* 欄位以顯示此訊息。

佇列管理程式有許多 內建格式 ，名稱以 MQ 開頭，例如 MQFMT_STRING。 如果這些不符合您的需求，您可以定義自己的格式 (使用者定義的格式) ，但不得針對這些格式使用以 MQ 開頭的名稱。

當您建立並使用自己的格式時，必須撰寫資料轉換結束程式，以支援使用 MQGMO_CONVERT 取得訊息的程式。

CodedCharSetId
這會定義訊息中字元資料的字集。 如果要將此字集設為佇列管理程式的字集，您可以將此欄位設為常數 MQCCSI_Q_MGR 或 MQCCSI_INHERIT。

當您從佇列取得訊息時，請比較 *CodedCharSetId* 欄位的值與應用程式預期的值。 如果這兩個值不同，您可能需要轉換訊息中的任何字元資料，或使用資料轉換訊息結束程式 (如果有的話)。

Encoding
這說明包含二進位整數、聚集十進位整數及浮點數字的數值訊息資料格式。 通常會根據執行佇列管理程式的特定機器來編碼。

當您將訊息放入佇列時，通常會在 *Encoding* 欄位中指定常數 MQENC_NATIVE。 這表示訊息資料的編碼與應用程式執行所在機器的編碼相同。

當您從佇列取得訊息時，請比較訊息描述子中的 *Encoding* 欄位值與機器上的常數 MQENC_NATIVE 值。 如果這兩個值不同，您可能需要轉換訊息中的任何數值資料，或使用資料轉換訊息結束程式 (如果有的話)。

應用程式資料轉換

應用程式資料可能需要轉換成字集，以及不同平台所涉及的另一個應用程式所需要的編碼。

它可以在傳送端佇列管理程式或接收端佇列管理程式中轉換。 如果內建格式的程式庫不符合您的需求，您可以自行定義。 轉換類型取決於訊息描述子 MQMD 的格式欄位中指定的訊息格式。

註: 不會轉換指定 MQFMT_NONE 的訊息。

傳送端佇列管理程式的轉換

如果您需要傳送訊息通道代理程式 (MCA) 來轉換應用程式資料，請將 CONVERT 通道屬性設為 YES。

如果提供適當的使用者結束程式，則會在傳送端佇列管理程式執行特定內建格式及使用者定義格式的轉換。

內建格式

這些包括:

- 所有字元的訊息 (使用格式名稱 MQFMT_STRING)
- IBM MQ 定義的訊息，例如「可程式指令格式」

  IBM MQ 針對管理訊息和事件使用「可程式化指令格式」訊息 (在此情況下使用的格式名稱是 MQFMT_ADMIN)。 您可以對自己的訊息使用相同的格式 (使用格式名稱 MQFMT_PCF) ，並利用內建資料轉換。

佇列管理程式內建格式都有以 MQFMT 開頭的名稱。 它們以 格式列出並說明。

應用程式定義的格式

對於使用者定義格式，應用程式資料轉換必須由資料轉換跳出程式執行 (如需相關資訊，請參閱 第 823 頁的『寫入資料-轉換結束程式』)。 在主從架構環境中，會在伺服器載入結束程式，並在該處進行轉換。

接收端佇列管理程式的轉換

接收端佇列管理程式可以針對內建及使用者定義格式來轉換應用程式訊息資料。

如果您指定 MQGMO_CONVERT 選項，則會在處理 MQGET 呼叫期間執行轉換。 如需詳細資料，請參閱 選項

編碼字集

IBM MQ 產品支援基礎作業系統所提供的編碼字集。

當您建立佇列管理程式時，所使用的佇列管理程式編碼字集 ID (CCSID) 是以基礎環境的編碼字集 ID (CCSID) 為基礎。 如果這是混合字碼頁， IBM MQ 會使用混合字碼頁的 SBCS 部分作為佇列管理程式 CCSID。

對於一般資料轉換，如果基礎作業系統支援 DBCS 字碼頁， IBM MQ 可以使用它。

請參閱作業系統的說明文件，以取得其支援之編碼字集的詳細資料。

在撰寫跨越多個平台的應用程式時，您需要考量應用程式資料轉換、格式名稱及使用者結束程式。 如需呼叫及寫入資料轉換結束程式的相關資訊，請參閱 第 823 頁的『寫入資料-轉換結束程式』 。

訊息優先順序

您可以將訊息的優先順序設為數值，或讓訊息採用佇列的預設優先順序。

將訊息放入佇列時，您可以設定訊息的優先順序 (在 MQMD 結構的 *Priority* 欄位中)。 您可以設定優先順序的數值，也可以讓訊息採用佇列的預設優先順序。

佇列的 **MsgDeliverySequence** 屬性決定佇列上的訊息是依 FIFO (先進先出) 順序儲存，還是依優先順序以 FIFO 順序儲存。 如果此屬性設為 MQMDS_PRIORITY ，則會以訊息描述子的 *Priority* 欄位中指定的優先順序將訊息放入佇列; 但如果設為 MQMDS_FIFO ，則會以佇列的預設優先順序將訊息放入佇列。 同等優先順序的訊息會依到達順序儲存在佇列上。

佇列的 **DefPriority** 屬性會設定放置在該佇列上之訊息的預設優先順序值。 此值是在建立佇列時設定，但之後可以變更。 別名佇列及遠端佇列的本端定義可以與它們所解析的基本佇列具有不同的預設優先順序。 如果解析路徑中有多個佇列定義 (請參閱 第 623 頁的『名稱解析』) ，則會從 open 指令中所指定佇列的 **DefPriority** 屬性值 (在執行 put 作業時) 取得預設優先順序。

佇列管理程式的 **MaxPriority** 屬性值是您可以指派給該佇列管理程式所處理之訊息的優先順序上限。 您無法變更此屬性的值。 在 IBM MQ 中，屬性值為 9; 您可以建立優先順序介於 0 (最低) 和 9 (最高) 之間的訊息。

訊息內容

使用訊息內容可讓應用程式選取要處理的訊息，或擷取訊息的相關資訊，而不存取 MQMD 或 MQRFH2 標頭。 它們也有助於 IBM MQ 與 JMS 應用程式之間的通訊。

訊息內容是與訊息相關聯的資料，由文字名稱和特定類型的值組成。 訊息選取器會使用訊息內容來過濾主題的發佈，或選擇性地從佇列取得訊息。 訊息內容可用來併入商業資料或狀態資訊，而不需要將它儲存在應用程式資料中。 應用程式不需要存取 MQ 訊息描述子 (MQMD) 或 MQRFH2 標頭中的資料，因為可以使用「訊息佇列介面 (MQI)」函數呼叫來存取這些資料結構中的欄位作為訊息內容。

在 IBM MQ 中使用訊息內容會模擬在 JMS 中使用內容。 這表示您可以在 JMS 應用程式中設定內容，並在程序化 IBM MQ 應用程式或以另一種方式來擷取它們。 若要讓內容可供 JMS 應用程式使用，請將字首 "usr" 指派給它; 然後它可作為 JMS 訊息使用者內容使用 (不含字首)。 例如， JMS 應用程式可以使用 JMS 呼叫 `message.getStringProperty('myproperty')` 來存取 IBM MQ 內容 *usr.myproperty* (字串)。 請注意，如果 JMS 應用程式包含兩個以上 U+002E (".") ，則無法存取字首為 "usr" 的內容。 字元。 不含字首且

沒有 U+002E (".") 的內容 字元會被視為具有字首 "usr"。 相反地，透過新增 "usr." ，可以在 IBM MQ 應用程式中存取 JMS 應用程式中設定的使用者內容。 MQINQMP 呼叫中所查詢之內容名稱的字首。

訊息內容及訊息長度

使用佇列管理程式屬性 *MaxPropertiesLength* 來控制可與 IBM MQ 佇列管理程式中任何訊息一起傳送的內容大小。

一般而言，當您使用 MQSETMP 來設定內容時，內容的大小是內容名稱的長度 (以位元組為單位) ，加上傳入 MQSETMP 呼叫的內容值長度 (以位元組為單位)。 在將訊息傳輸至其目的地期間，可以變更內容名稱及內容值的字集，因為這些可以轉換為 Unicode; 在此情況下，內容的大小可能會變更。

在 MQPUT 或 MQPUT1 呼叫上，訊息內容不會計入佇列及佇列管理程式的訊息長度，但會計入佇列管理程式所察覺內容的長度 (不論是否使用訊息內容 MQI 呼叫來設定)。

如果內容大小超出內容長度上限，則會以 MQRC_PROPERTIES_TOO_BIG 拒絕訊息。 因為內容的大小取決於其表示法，所以您應該在總層次設定內容長度上限。

如果緩衝區包含內容，應用程式可以順利放置緩衝區大於 *MaxMsg* 長度值的訊息。 這是因為即使以 MQRFH2 元素表示，訊息內容也不會計入訊息的長度。 只有在包含一或多個資料夾且標頭中的每個資料夾都包含內容時， MQRFH2 標頭欄位才會新增至內容長度。 如果一或多個資料夾包含在 MQRFH2 標頭中，且任何資料夾不包含內容，則 MQRFH2 標頭欄位會改為計入訊息長度。

在 MQGET 呼叫中，就佇列和佇列管理程式而言，訊息的內容不會計入訊息的長度。 不過，因為內容是個別計算的，所以 MQGET 呼叫所傳回的緩衝區可能大於 *MaxMsgLength* 屬性的值。

請不要讓應用程式查詢 *MaxMsg* 長度 的值，然後在呼叫 MQGET 之前配置此大小的緩衝區; 而是配置您認為足夠大的緩衝區。 如果 MQGET 失敗，請依 *DataLength* 參數的大小來配置緩衝區。

如果 MQGMO 結構中未指定訊息控點， MQGET 呼叫的 *DataLength* 參數會傳回應用程式資料的長度 (以位元組為單位) ，以及您所提供之緩衝區中所傳回的任何內容。

MQPUT 呼叫的 緩衝區 參數包含要傳送的應用程式訊息資料，以及訊息資料中代表的任何內容。

訊息內容的長度限制為 100 MB ，不包括每則訊息的訊息描述子或延伸。

內容在其內部表示法中的大小是名稱的長度，加上其值的大小，加上內容的部分控制資料。 在訊息中新增一個內容之後，該內容集也會有一些控制資料。

內容名稱

內容名稱是字串。 某些限制適用於其長度及可使用的字元集。

內容名稱是區分大小寫的字串，除非環境定義另有限制，否則限制為 +4095 個字元。 此限制包含在 MQ_MAX_PROPERTY_NAME_LENGTH 常數中。

如果您在使用訊息內容 MQI 呼叫時超出此長度上限，則呼叫會失敗，原因碼為 MQRC_PROPERTY_NAME_LENGTH_ERR。

因為 JMS 中沒有內容名稱長度上限，所以當儲存在 MQRFH2 結構中時， JMS 應用程式可以設定有效的 JMS 內容名稱，但該內容名稱不是有效的 IBM MQ 內容名稱。

在此情況下，當剖析時，只會使用內容名稱的前 4095 個字元; 下列字元會被截斷。 這可能導致使用選取元的應用程式無法符合選取字串，或在未預期時無法符合字串，因為多個內容可能會截斷為相同的名稱。 截斷內容名稱時， WebSphereMQ 會發出錯誤日誌訊息。

所有內容名稱都必須遵循 Java Java ID 的語言規格所定義的規則，但 Unicode 字元 U+002E (.) 可作為名稱的一部分，但不能作為開頭。 Java ID 的規則等於內容名稱的 JMS 規格中包含的那些規則。

禁止使用空格字元及比較運算子。 內容名稱中容許內含空值，但不建議使用。 如果您使用內嵌空值，當與 MQCHARV 結構一起使用來指定可變長度字串時，這會防止使用 MQVS_NULL_TERMINATED 常數。

讓內容名稱保持簡單，因為應用程式可以根據內容名稱以及名稱與選取元的字集之間的轉換來選取訊息，可能會導致選取非預期地失敗。

IBM MQ 內容名稱使用字元 U+002E (.) 來進行內容的邏輯分組。 這會劃分內容的名稱空間。 具有下列字首的內容 (任何小寫或大寫的混合) 會保留供產品使用:

- mcd

- jms
- usr
- mq
- sib
- wmq
- Root
- Body
- Properties

避免名稱衝突的一個好方法是確保所有應用程式都以其網際網路網域名稱作為其訊息內容的字首。 例如，如果您使用網域名稱 ourcompany.com 來開發應用程式，則可以使用字首 com.ourcompany 來命名所有內容。 此命名慣例也容許輕鬆選取內容; 例如，應用程式可以查詢啟動 com.ourcompany.%的所有訊息內容。

如需使用內容名稱的進一步相關資訊，請參閱 內容名稱限制 。

內容名稱限制
當您命名內容時，必須遵守特定規則。

下列限制適用於內容名稱:

1. 內容不得以下列字串開頭:
   - "JMS"-保留供 IBM MQ classes for JMS 使用。
   - "usr.JMS"-無效。

   唯一例外是下列內容提供 JMS 內容的同義字:

內容	同義字
JMSCorrelationID	Root.MQMD.CorrelId 或 jms.Cid
JMSDeliveryMode	Root.MQMD.Persistence 或 jms.Dlv
JMSDestination	jms.Dst
JMSExpiration	Root.MQMD.Expiry 或 jms.Exp
JMSMessageID	Root.MQMD.MsgId
JMSPriority	Root.MQMD.Priority 或 jms.Pri
JMSRedelivered	Root.MQMD.BackoutCount
JMSReplyTo (編碼為 URI 的字串)	Root.MQMD.ReplyToQ 或 Root。MQMD.ReplyToQMgr 或 jms.Rto
JMSTimestamp	Root.MQMD.PutDate 或 Root。MQMD.PutTime 或 jms.Tms
JMSType	mcd.Type 或 mcd.Set 或 mcd.Fmt
JMSXAppID	Root.MQMD.PutApplName
JMSXDeliveryCount	Root.MQMD.BackoutCount
JMSXGroupID	Root.MQMD.GroupId 或 jms.Gid
JMSXGroupSeq	Root.MQMD.MsgSeqNumber 或 jms.Seq
JMSXUserID	Root.MQMD.UserIdentifier

   這些同義字容許 MQI 應用程式以類似於 IBM MQ classes for JMS 用戶端應用程式的方式來存取 JMS 內容。 在這些內容中，只能使用 MQI 來設定 JMSCorrelationID、 JMSReplyTo、JMSType、JMSXGroupID 及 JMSXGroupSeq 。

請注意，使用 MQI 無法使用 IBM MQ classes for JMS 內提供的 JMS_IBM_ * 內容。 MQI 應用程式可以其他方式來存取 JMS_IBM_ * 內容所參照的欄位。

2. 在任何大小寫混合的情況下，不得呼叫內容 "NULL"、"TRUE"、"FALSE"、"NOT"、"AND"、"OR"、"之間"、"LIKE"、"IN"、"IS" 及 "ESCAPE"。 這些是選擇字串中使用的 SQL 關鍵字名稱。
3. 以 " 開頭的內容名稱 mq " 在任何小寫或大寫的混合中，且不是以 "mq_usr" 開頭的 "mq_usr" 只能包含一個 "." 字元 (U+002E)。 多個 "." 在具有這些字首的內容中不容許使用個字元。
4. 兩個 "." 字元之間必須包含其他字元; 階層中不能有空點。 同樣地，內容名稱不能以 "." 結尾。 來區隔。
5. 如果應用程式設定內容 "a.b" ，然後設定內容 "a.b.c" ，則不清楚在階層 "b" 中是否包含值或其他邏輯分組。 此類階層是「混合內容」，且不受支援。 不容許設定導致混合內容的內容。

這些限制由驗證機制強制執行，如下所示:

- 如果在建立訊息控點時要求驗證，當使用 MQSETMP-設定訊息內容 呼叫來設定內容時，會驗證內容名稱。 如果嘗試驗證內容，但由於內容名稱規格中的錯誤而失敗，則完成碼為 MQCC_FAILED ，原因為:
  - MQRC_PROPERTY_NAME_ERROR ，原因為 1-4。
  - MQRC_MIXED_CONTENT_NOT_ALLOWED ，原因為 5。
- 不保證 MQPUT 呼叫會驗證直接指定為 MQRFH2 元素的內容名稱。

訊息描述子欄位作為內容
大部分訊息描述子欄位都可以視為內容。 內容名稱是透過將字首新增至訊息描述子欄位的名稱來建構。

如果 MQI 應用程式想要識別訊息描述子欄位 (例如，在選取元字串中或使用訊息內容 API) 中包含的訊息內容，請使用下列語法:

內容名稱	訊息描述子欄位
Root.MQMD。欄位	欄位

指定 *Field* ，其大小寫與 C 語言宣告中 MQMD 結構欄位的大小寫相同。 例如，內容名稱 `Root.MQMD.AccountingToken` 會存取訊息描述子的 `AccountingToken` 欄位。

無法使用所顯示的語法來存取訊息描述子的 `StrucId` 及 `Version` 欄位。

對於其他內容，在 MQRFH2 標頭中絕不會呈現訊息描述子欄位。

如果訊息資料以佇列管理程式允許使用的 MQMDE 開頭，則可以使用所說明的 `Root.MQMD.`*`Field`* 表示法來存取 MQMDE 欄位。 在此情況下，會從內容視景將 MQMDE 欄位視為 MQMD 的邏輯部分。 請參閱 MQMDE 概觀。

內容資料類型和值

內容可以是布林、位元組字串、字串或浮點或整數。 除非環境定義另有限制，否則內容可以儲存資料類型範圍內的任何有效值。

內容值的資料類型必須是下列其中一個值:

- MQBOOL
- MQBYTE []
- MQCHAR []
- MQFLOAT32
- MQFLOAT64
- MQINT8
- MQINT16
- MQINT32
- MQINT64

內容可以存在，但沒有已定義的值; 它是空值內容。 空值內容不同於位元組內容 (MQBYTE []) 或字串內容 (MQCHAR []) ，因為它具有已定義但空白的值，即具有零長度值的值。

位元組字串在 JMS 或 XMS 中不是有效的內容資料類型。 建議您不要使用 *usr* 資料夾中的位元組字串內容。

從佇列中選取訊息

您可以在 MQGET 呼叫上使用 MsgId 及 CorrelId 欄位，或在 MQOPEN 或 MQSUB 呼叫上使用 SelectionString，從佇列中選取訊息。

選取器

訊息選取器是一個可變長度字串，由應用程式用來僅登錄其感興趣的訊息，這些訊息具有選項字串所代表的「結構化查詢語言 (SQL)」查詢的內容。

使用 MQSUB 和 MQOPEN 函數呼叫的選項

您可以使用 *SelectionString*(類型為 MQCHARV 的結構)，以使用 MQSUB 及 MQOPEN 呼叫進行選擇。

SelectionString 結構用來將可變長度選取字串傳遞至佇列管理程式。

與選取元字串相關聯的 CCSID 是透過 MQCHARV 結構的 VSCCSID 欄位來設定。 所使用的值必須是選取元字串所支援的 CCSID。 如需受支援字碼頁的清單，請參閱 字碼頁轉換 。

指定沒有 IBM MQ 支援 Unicode 轉換的 CCSID 會導致 MQRC_SOURCE_CCSID_ERROR 錯誤。 當選取元呈現給佇列管理程式時，即在 MQSUB、MQOPEN 或 MQPUT1 呼叫上，會傳回此錯誤。

VSCCSID 欄位的預設值是 MQCCSI_APPL，指出選取字串的 CCSID 等於佇列管理程式 CCSID，或用戶端 CCSID (如果透過用戶端連接)。 不過， MQCCSI_APPL 常數可以在編譯之前由重新定義它的應用程式置換。

如果 MQCHARV 選取器代表空值字串，則不會對該訊息消費者進行任何選擇，且會如同未使用選取器一樣遞送訊息。

選取字串的長度上限僅受限於 MQCHARV 欄位 *VSLength* 可以說明的內容。

如果您已提供緩衝區，且 VSBufSize 中有正的緩衝區長度，則會在 MQSUB 呼叫使用 MQSO_RESUME SUBSCRIBE 選項的輸出上傳回 SelectionString。 如果您未提供緩衝區， MQCHARV 的 VSLength 欄位中只會傳回選取字串的長度。 如果提供的緩衝區小於傳回欄位所需的空間，則在提供的緩衝區中只會傳回 VSBufSize 個位元組。

應用程式必須先關閉佇列的控點 (適用於 MQOPEN) 或訂閱 (適用於 MQSUB)，才能變更選項字串。 然後可以在後續的 MQOPEN 或 MQSUB 呼叫上指定新的選擇字串。

MQOPEN
使用 MQCLOSE 來關閉開啟的控點，然後在後續的 MQOPEN 呼叫上指定新的選取字串。

MQSUB
使用 MQCLOSE 來關閉傳回的訂閱控點 (hSub)，然後在後續的 MQSUB 呼叫上指定新的選取字串。

第 27 頁的圖 3 顯示使用 MQSUB 呼叫進行選取的程序。

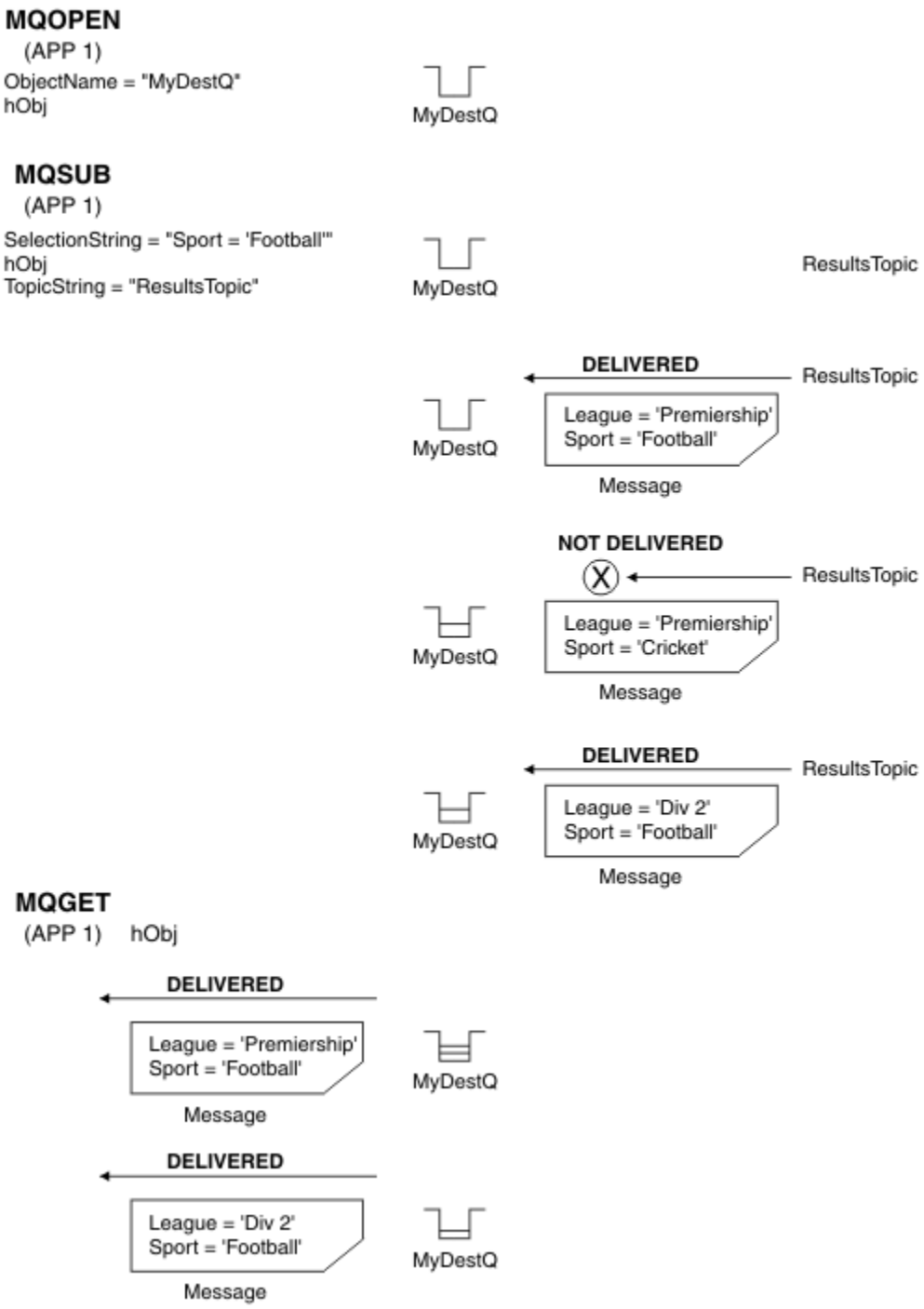


圖 *3:* 使用 *MQSUB* 呼叫的選項

可以使用 MQSD 結構中的 *SelectionString* 欄位，在呼叫 MQSUB 時傳入選取元。 在 MQSUB 上傳入選取元的效果是，只有已發佈至所訂閱主題且符合所提供選取字串的那些訊息，才會在目的地佇列上提供。

第 28 頁的圖 4 顯示使用 MQOPEN 呼叫進行選取的程序。

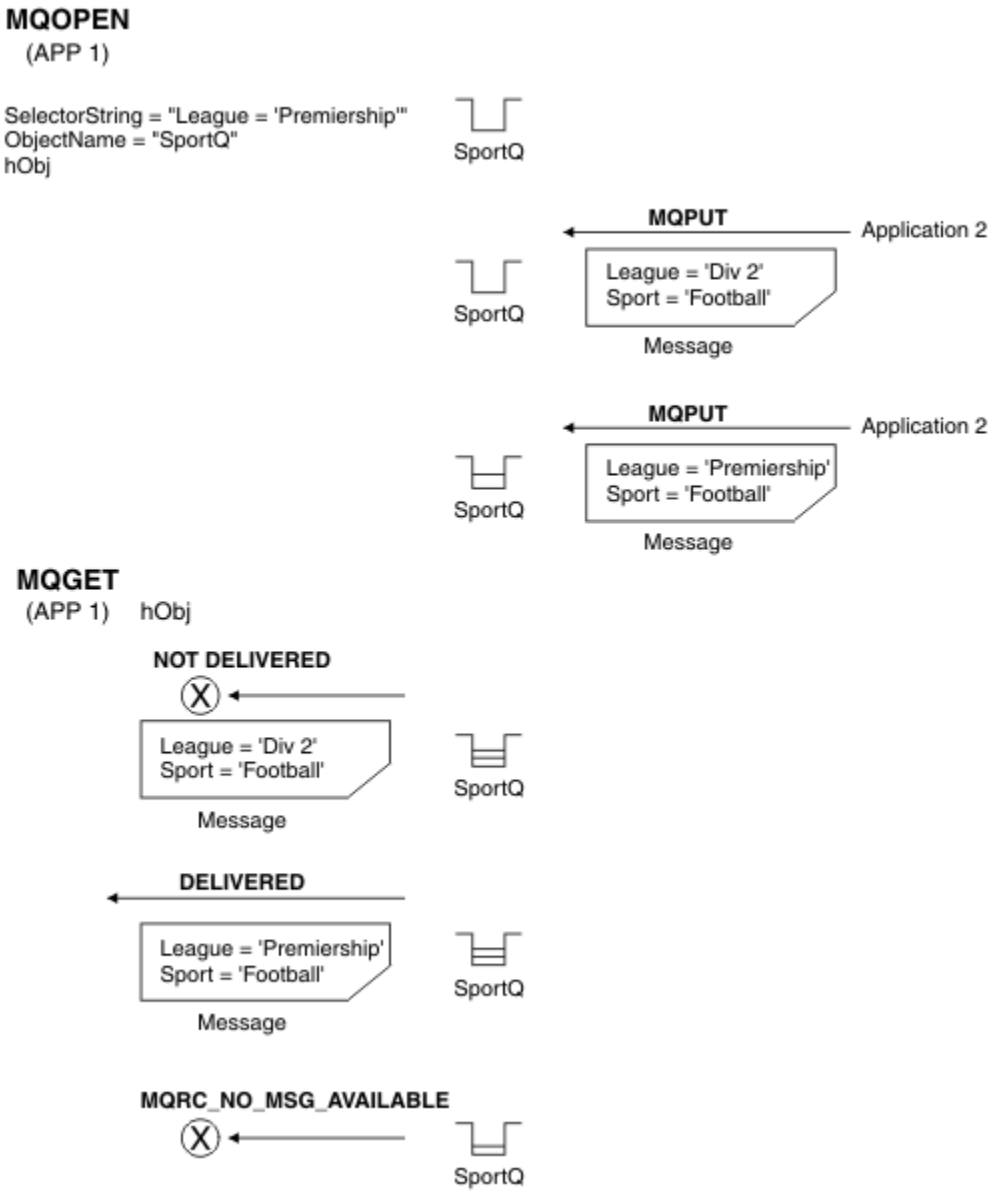


圖 *4:* 使用 *MQOPEN* 呼叫的選項

使用 MQOD 結構中的 *SelectionString* 欄位，可以在 MQOPEN 呼叫時傳入選取元。 在 MQOPEN 呼叫上傳入選取元的效果是，只有開啟佇列上符合選取元的那些訊息才會遞送至訊息消費者。

MQOPEN 呼叫上選取元的主要用途是在點對點情況下，應用程式可以選擇只接收佇列上符合選取元的那些訊息。 前一個範例顯示一個簡式實務範例，其中兩則訊息放置在 MQOPEN 所開啟的佇列中，但取得它的應用程式只會收到其中一則訊息，因為它是唯一符合選取器的訊息。

請注意，後續 MQGET 呼叫會導致 MQRC_NO_MSG_AVAILABLE ，因為佇列中不存在符合給定選取器的進一步訊息。

相關概念

第 34 頁的『選取字串規則及限制』
請熟悉下列規則: 如何解譯選取字串及字元限制，以避免在使用選取元時發生潛在問題。

選取行為

IBM MQ 選取行為的概觀。

如果 MQMD ，則 MQMDE 結構中的欄位會被視為對應訊息描述子內容的訊息內容:

- 格式為 MQFMT_MD_EXTENSION
- 後面緊接著有效的 MQMDE 結構
- 是第 1 版或只包含預設第 2 版欄位

在對訊息內容進行任何比對之前，可以將選取字串解析為 TRUE 或 FALSE。 例如，如果選取字串設為 "TRUE <> FALSE"，則可能是這種情況。 只有在選取字串中沒有訊息內容參照時，才保證會進行這類早期評估。

如果選擇字串在考量任何訊息內容之前解析為 TRUE，則會遞送發佈至消費者所訂閱主題的所有訊息。 如果在考量任何訊息內容之前，選擇字串解析為 FALSE，則會在呈現選取器的函數呼叫上傳回原因碼 MQRC_SELECTOR_ALWAYS_FALSE 及完成碼 MQCC_FAILED。

即使訊息未包含任何訊息內容 (除了標頭內容之外)，它仍然可以進行選取。 如果選取字串所參照的訊息內容不存在，則會假設此內容具有 NULL 或「不明」值。

例如，訊息可能仍然滿足 'Color IS NULL'之類的選取字串，其中 'Color' 不存在作為訊息中的訊息內容。

除非有延伸訊息選擇提供者可用，否則只能對與訊息相關聯的內容執行選擇，而不能對訊息本身執行選擇。 只有在延伸訊息選取提供者可用時，才能對訊息有效負載執行選取。

每一個訊息內容都有一個相關聯的類型。 當您執行選項時，必須確定表示式中用來測試訊息內容的值類型正確。 如果發生類型不符，則有問題的表示式會解析為 FALSE。

您有責任確保選取字串和訊息內容使用相容類型。

繼續代表非作用中可延續訂閱者套用選取準則，因此只會保留符合原始提供之選取字串的訊息。

使用變更 (MQSO_ALTER) 回復可延續訂閱時，選擇字串不可變更。 如果在可延續訂閱者回復活動時呈現不同的選擇字串，則 MQRC_SELECTOR_NOT_ALTERABLE 會傳回給應用程式。

如果佇列上沒有符合選取準則的訊息，應用程式會收到回覆碼 MQRC_NO_MSG_AVAILABLE。

如果應用程式已指定包含內容值的選取字串，則只有包含相符內容的那些訊息才適合進行選取。 例如，訂閱者指定選取字串 "a = 3"，且發佈的訊息不包含任何內容，或其中 'a' 不存在或不等於 3 的內容。 訂閱者不會將該訊息接收到其目的地佇列。

傳訊效能

從佇列中選取訊息需要 IBM MQ 循序檢查佇列上的每一個訊息。 會檢查訊息，直到找到符合選取準則的訊息，或沒有其他要檢查的訊息為止。 因此，如果在深層佇列上使用訊息選項，則傳訊效能會受到影響。

當選取是以 JMSCorrelationID 或 JMSMessageID 為基礎時，如果要最佳化深度佇列上的訊息選取，請使用下列格式的選取字串:

- JMSCorrelationID = 'ID:*correlation_id*'
- JMSMessageID= 'ID:*message_id*'

其中:

- *correlation_id* 是包含標準 IBM MQ 相關性 ID 的字串。
- *message_id* 是包含標準 IBM MQ 訊息 ID 的字串。

註: 選取元應該只參照其中一個內容。 使用具有下列其中一種格式的選取器，可在選取 JMSCorrelationID 時大幅提高效能，並提供 JMSMessageID 的邊際效能改善。 如需相關資訊，請參閱 第 122 頁的『JMS 中的訊息選取器』。

使用複式選取器

選取元可以包含許多元件，例如:

a 及 b 或 c 及 d 或 e 及 f 或 g 及 h 或 i 及 j... 或 y 和 z

使用此類複式選取器可能具有嚴重的效能影響及過多的資源需求。 因此， IBM MQ 將透過無法處理可能導致系統資源短缺的過度複雜選取器來保護系統。 在包含超過 100 個測試的選取字串上，或當 IBM MQ 偵測到

接近作業系統堆疊的大小限制時，可能會進行保護。 您應該在適當的平台上徹底嘗試並測試使用具有許多元件的選取字串，以確保未達到保護限制。

透過使用其他括弧來結合元件，可以簡化選取元的效能及複雜性。 例如：

(a 及 b 或 c 及 d) 或 (e 及 f 或 g 及 h) 或 (i 及 j) ...

相關概念

第 34 頁的『選取字串規則及限制』
請熟悉下列規則: 如何解譯選取字串及字元限制，以避免在使用選取元時發生潛在問題。

訊息選取器語法

IBM MQ 訊息選取器是一個字串，其語法是以 SQL92 條件式表示式語法的子集為基礎。

在優先順序層次內，訊息選取器的評估順序是從左到右。 您可以使用括弧來變更此順序。 這裡會以大寫撰寫預先定義的選取元文字及運算子名稱; 不過，它們不區分大小寫。

如果透過 API 提供選取元， IBM MQ 會在呈現訊息選取元時驗證訊息選取元的語法正確性。 如果選取字串的語法不正確或內容名稱無效，且無法使用延伸訊息選取提供者，則 MQRC_SELECTION_NOT_AVAILABLE 會傳回給應用程式。 當回復訂閱時，如果選取字串的語法不正確或內容名稱無效，則會將 MQRC_SELECTOR_SYNTAX_ERROR 傳回給應用程式。 如果在設定內容時已停用內容名稱驗證 (透過設定 MQCMHO_NONE 而非 MQCMHO_VALIDATE) ，且應用程式隨後將具有無效內容名稱的訊息放置在無效內容名稱中，則永不會選取此訊息。

如果 IBM MQ 判定管理定義的訂閱選取元正在使用延伸訊息語法 (如具有值 EXTENDED 的 **DISPLAY SUB** 參數 **SELTYPE** 所指示) ，則在呈現選取元時不會傳回任何錯誤。 在此情況下，選取字串的語法檢查會延遲到發佈時間 (請參閱 MQRC_SELECTION_NOT_AVAILABLE)。

選取元可以包含:

- 文字:
  - 字串文字以單引號括住。 兩個連續單引號代表一個單引號。 範例為 'literal' 及 'literal'。 與 Java 字串文字一樣，這些文字使用 Unicode 字元編碼。 您無法使用雙引號來含括字串文字。 任何位元組序列都可以在單引號之間使用。
  - 位元組字串是一對以上以雙引號括住且字首為 0x 的十六進位字元。 例如 "0x2F1C" 或 "0XD43A"。 位元組字串的長度必須至少為一個位元組。 如果選取元位元組字串符合 MQTYPE_BYTE_STRING 類型的訊息內容，則不會對前導或尾端零採取任何特殊動作。 位元組被視為另一個字元。 也不考慮排列法。 選取元及內容位元組字串的長度必須相等，且位元組順序必須相同。

    符合下列情況的位元組字串選擇範例 (假設 *myBytes* = 0AFC23):

    - "myBytes = "0x0AFC23"" = TRUE

    下列字串選項不相符:

    - "myBytes = "0xAFC23"" = MQRC_SELECTOR_SYNTAX_ERROR (因為位元組數不是 2 的倍數)
    - "myBytes = "0x0AFC2300"" = FALSE (因為尾端零在比較中是顯著的)
    - "myBytes = "0x000AFC23"" = FALSE (因為前導零在比較中是顯著的)
    - "myBytes = "0x23FC0A"" = FALSE (因為未考量排列法)
  - 十六進位數字以零開頭，後面接著大寫或小寫 x。 文字的其餘部分包含一個以上有效的十六進位字元。 範例有 0xA、 0xAF、 0X2020。
  - 前導零後面接著 0-7 範圍內的一或多個數字，一律會解譯為八進位數字的開頭。 您無法代表字首為零的十進位數，例如， 09 會傳回語法錯誤，因為 9 不是有效的八進位數字。 八進位數字的範例為 0177、 0713。
  - 確切數值文字是不含小數點的數值，例如 57、 -957 及 +62。 確切數值文字可以有尾端大寫或小寫 L; 這不會影響數字的儲存或解譯方式。 IBM MQ 支援 -9,223,372,036,854,775,808 至 9,223,372,036,854,775,807 範圍內的確切數字。
  - 近似數值文字是採用科學記號表示法的數值 (例如 7E3 或 -57.9E2) ，或具有小數的數值 (例如 7.、 -95.7 或 +6.2)。 IBM MQ 支援 -1.797693134862315E+308 至 1.797693134862315E+308 範圍內的數字。

顯著性應該遵循選用的符號字元 (+ 或 -)。 顯著性應該是整數或分數。 顯著性的小數部分，且不需要有前導數字。

大寫或小寫 E 表示選用指數的開頭。 指數具有十進位基數，且指數的數字部分可以以選用符號字元作為字首。

近似數值文字可以由 F 或 D 字元終止 (不區分大小寫)。 此語法是為了支援跨語言方法來標記單一或倍精準度數字。 這些字元是選用的，不會影響近似數值文字的儲存或處理方式。 這些數字一律使用倍精準度來儲存及處理。

– 布林文字 TRUE 和 FALSE。

註: 選取字串中不支援無限 IEEE-754 表示法，例如 NaN、 +Infinity、 -Infinity 。 因此，無法在表示式中使用這些值作為運算元。 在數學運算中，負零會被視為正零。

- ID:

ID 是可變長度字元序列，必須以有效 ID 起始字元開頭，後面接著零個以上有效 ID 部分字元。 ID 名稱的規則與訊息內容名稱的規則相同，如需相關資訊，請參閱 第 23 頁的『內容名稱』 及 第 24 頁的『內容名稱限制』 。

註: 只有在延伸訊息選取提供者可用時，才能對訊息有效負載執行選取。

ID 是標頭欄位參照或內容參照。 訊息選取器中的內容值類型必須對應於用來設定內容的類型，雖然在可能的情況下會執行數值促銷。 如果發生類型不符，則表示式的結果為 FALSE。 如果參照不存在於訊息中的內容，則其值為 NULL。

在訊息選取元表示式中使用內容時，適用於內容的 get 方法的類型轉換不適用。 例如，如果您將內容設為字串值，然後使用選取器將其查詢為數值，則表示式會傳回 FALSE。

JMS 欄位及對映至內容名稱或 MQMD 欄位名稱的內容名稱也是選取字串中的有效 ID。 IBM MQ 會將已辨識的 JMS 欄位及內容名稱對映至訊息內容值。 如需相關資訊，請參閱第 122 頁的『JMS 中的訊息選取器』。 例如，在現行訊息的 jms 資料夾中找到的 Pri 內容上，選取字串 "JMSPriority >=" 會選取。

- 溢位/下溢:

對於十進位數和近似數值，未定義下列條件:

– 指定超出定義範圍的數字

– 指定會導致溢位或下溢的算術表示式

不會針對這些條件執行任何檢查。

- 空格:

定義為空格、換頁、換行、歸位、水平定位點或垂直定位點。 下列 Unicode 字元可辨識為空格:

– \u0009 to \u000D

– \u0020

– \u001C

– \u001D

– \u001E

– \u001F

– \u1680

– \u180E

– \u2000 至 \u200A

– \u2028

– \u2029

– \u202F

– \u205F

– \u3000

- 表示式:

  - 選取元是條件式表示式。 評估為 true 相符項的選取元; 評估為 false 或不明的選取元不相符。
  - 算術表示式由自己、算術運算、ID (ID 值被視為數值文字) 和數值文字組成。
  - 條件式表示式由其本身、比較運算及邏輯運算組成。
- 支援標準括弧 () ，以設定評估表示式的順序。
- 依優先順序的邏輯運算子: NOT、 AND、 OR。
- 比較運算子: =、 >、 >=、 <、 <=、 <> (不等於)。
  - 只有在字串長度相同且位元組順序相等時，兩個位元組字串才相等。
  - 只能比較相同類型的值。 有一個例外是比較確切數值和近似數值是有效的 (所需的類型轉換是由 Java 數值促銷活動的規則所定義)。 如果嘗試比較不同的類型，則選取元一律為 false。
  - 字串和布林比較限制為 = 和 <>。 只有在兩個字串包含相同的字元序列時，它們才相等。
- 依優先順序的算術運算子:
  - +、 - 單元。
  - * 乘法和 / 除法。
  - + 加法及 - 減法。
  - 不支援空值上的算術運算。 如果嘗試它們，則完整選取元一律為 false。
  - 算術運算必須使用 Java 數值促銷。
- arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 和 arithmetic-expr3 比較運算子:
  - `Age BETWEEN 15 and 19` 相當於 `age>= 15 AND age <= 19`。
  - `Age NOT BETWEEN 15 and 19` 相當於 `age < 15 OR age > 19`。
  - 如果 BETWEEN 作業的任何表示式為 NULL，則作業的值為 false。 如果 `NOT BETWEEN` 運算的任何表示式為 NULL，則運算的值為 true。
- `ID [NOT] IN (string-literal1, string-literal2,...)` 比較運算子，其中 ID 具有字串或 NULL 值。
  - `Country IN ('UK', 'US', 'France')` 若為 `'UK'`，則為 true; 若為 `'Peru'`，則為 false。 它相當於表示式 `(Country = 'UK') OR (Country = 'US') OR (Country = 'France')`。
  - `Country NOT IN ('UK', 'US', 'France')` 若為 `'UK'`，則為 false; 若為 `'Peru'`，則為 true。 它相當於表示式 `NOT ((Country = 'UK') OR (Country = 'US') OR (Country = 'France'))`。
  - 如果 IN 或 `NOT IN` 作業的 ID 是空值，則作業的值不明。
- `identifier [NOT] LIKE` *`pattern-value`* `[ESCAPE` *`escape-character`* `]` 比較運算子，其中 `identifier` 具有字串值。 *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)。 所有其他角色都代表自己。 選用 *escape-character* 是單一字串文字，用來跳出 *pattern-value* 中 _ 及 % 的特殊意義。 LIKE 運算子只能用來比較兩個字串值。
  - `phone LIKE '12%3'` 對於 123 和 12993 是 true，對於 1234 是 false。
  - `word LIKE 'l_se'` 對於 lose 為 true，對於 loose 為 false。
  - `underscored LIKE '\_%' ESCAPE '\'` 若為 `_foo`，則為 true; 若為 bar，則為 false。
  - `phone NOT LIKE '12%3'` is false for 123 and 12993 and true for 1234.
  - 如果 LIKE 或 `NOT LIKE` 作業的 ID 是 NULL，則作業的值不明。

  註: 必須使用 LIKE 運算子來比較兩個字串值。 `Root.MQMD.CorrelId` 的值是 24 位元組位元組陣列，不是字串。 剖析器接受語法有效的選取器字串 `Root.MQMD.CorrelId LIKE 'ABC%'`，但會評估為 false。 當您比較位元組陣列與字串時，無法使用 LIKE 。
- `identifier IS NULL` 比較運算子會測試 NULL 標頭欄位值或遺漏內容值。
- `identifier IS NOT NULL` 比較運算子會測試非空值標頭欄位值或內容值是否存在。
- 空值

  在摘要中，包含 NULL 值的選取元表示式的評估由 SQL 92 NULL 語意定義:

– SQL 會將 NULL 值視為不明。
– 具有不明值的比較或算術一律會產生不明值。
– IS NULL 和 IS NOT NULL 運算子會將不明值轉換為 TRUE 和 FALSE 值。

布林運算子使用三值邏輯 (T=TRUE、 F=FALSE、 U=UNKNOWN)

表 *1:* 邏輯為 *A AND B* 時的布林運算子結果值

操作員 A	**運算子 B**	**結果 (A 和 B)**
T	F	F
T	U	U
T	T	T
F	T	F
F	U	F
F	F	F
U	T	U
U	U	U
U	F	F

表 *2:* 邏輯為 *A OR B* 時的布林運算子結果值

操作員 A	**運算子 B**	**結果 (A OR B)**
T	F	T
T	U	T
T	T	T
F	T	T
F	U	U
F	F	F
U	T	T
U	U	U
U	F	U

表 *3:* 邏輯為 *NOT A* 時的布林運算子結果值

操作員 A	**結果 (NOT A)**
T	F
F	T
U	U

下列訊息選取器會選取訊息類型為 car、顏色為藍色且加權大於 2500 磅的訊息:

```
"JMSType = 'car' AND color = 'blue' AND weight > 2500"
```

雖然 SQL 支援固定十進位比較及算術，但訊息選取元不支援。 這就是為何精確數值文字限制為沒有小數的文字。 這也是為何會有數字以十進位作為近似數值的替代表示法。

不支援 SQL 註解。

相關概念

第 22 頁的『訊息內容』
使用訊息內容可讓應用程式選取要處理的訊息，或擷取訊息的相關資訊，而不存取 MQMD 或 MQRFH2 標頭。 它們也有助於 IBM MQ 與 JMS 應用程式之間的通訊。

相關參考

MsgHandle

MQBUFMH-將緩衝區轉換成訊息控點

選取字串規則及限制

請熟悉下列規則: 如何解譯選取字串及字元限制，以避免在使用選取元時發生潛在問題。

- 發佈/訂閱傳訊的訊息選擇發生在發佈者所傳送的訊息上。 請參閱 選取字串。
- 等值是使用單一等於字元進行測試; 例如， `a = b` 是正確的，而 `a == b` 是不正確的。
- 許多程式設計語言用來代表「不等於」的運算子是 `!=`。 此表示法不是 `<>` 的有效同義字; 例如， `a <> b` 有效，而 `a != b` 無效。
- 只有在 ' 使用 (U+0027) 字元。 同樣地，雙引號 (只有在用來含括位元組字串時才有效) 必須使用 " (U+0022) 字元。
- 符號 &、 &&、 | 和 || 不是邏輯連結/聯集的同義字; 例如， `a && b` 必須指定為 `a AND b`。
- 萬用字元 * 和 ? 不是 % 和 _ 的同義字。
- 包含複合表示式 (例如 `20 < b < 30`) 的選取元無效。 剖析器會從左至右評估具有相同優先順序的運算子。 因此，範例會變成 `(20 < b) < 30`，這沒有意義。 相反地，表示式必須撰寫為 `(b > 20) AND (b < 30)`。
- 位元組字串必須以雙引號括住; 如果使用單引號，則位元組字串會被當作字串文字。 `0x` 後面的字元數 (不是字元代表的數字) 必須是 2 的倍數。
- 關鍵字 IS 不是等號字元的同義字。 因此，選取字串 `a IS 3` 和 `b IS 'red'` 無效。 IS 關鍵字只有在支援 `IS NULL` 和 `IS NOT NULL` 觀察值時才存在。

相關概念

第 28 頁的『選取行為』
IBM MQ 選取行為的概觀。

相關參考

選取字串

使用訊息選取器時的 *UTF-8* 及 *Unicode* 考量

構成選取字串之保留關鍵字的字元 (未以單引號括住) 必須以「基本拉丁文 Unicode」輸入 (範圍從字元 U+0000 到 U+0007F)。 使用其他英數字元的字碼點表示法無效。 例如，數字 1 必須以 Unicode U+0031 表示，無法使用 Fullwidth Digit 對等項目 U+FF11 或阿拉伯文對等項目 U+0661。

可以使用任何有效的 Unicode 字元序列來指定訊息內容名稱。 即使訊息內容名稱包含多位元組字元，也會驗證以 UTF-8 編碼的選取字串內所包含的訊息內容名稱。 多位元組 UTF-8 的驗證嚴格，您必須確定訊息內容名稱使用有效的 UTF-8 序列。 訊息內容名稱中不支援以代理字碼點 (X'D800'至 X'DFF') 或 UTF-8 中四個位元組來代表 UTF-16 的「Unicode 基本多語言平面」(U + FFFF 以上) 以外的字元。

比較相等性時，不會對內容名稱或值執行額外處理。 例如，這表示不發生前置/解除組合，且連字沒有任何特殊意義。 例如，預先編製的母音字元 U+00FC 不視為相等於 U+0075 + U+0308 ，而字元序列 ff 則不視為相等於 Unicode U+FB00 (Latin SMALL LIGATURE FF)

以單引號括住的內容資料可以用任何位元組序列來表示，且不會驗證。

在訊息內容上選取

可以根據訊息有效負載內容的選項 (也稱為內容過濾) 來訂閱，但 IBM MQ 無法直接執行應該將哪些訊息遞送至這類訂閱的決策; 而是需要延伸訊息選取提供者 (例如 IBM Integration Bus) 來處理訊息。

當應用程式在主題字串上發佈時，其中一個以上訂閱者已在訊息內容上選取選取字串， IBM MQ 會要求延伸訊息選取提供者剖析發佈，並通知 IBM MQ 發佈是否符合具有內容過濾器之每一個訂閱者指定的選取準則。

如果延伸訊息選擇提供者判定發佈符合訂閱者的選擇字串，則訊息將繼續遞送至訂閱者。

如果延伸訊息選擇提供者判定發佈不相符，則訊息不會遞送至訂閱者。 這可能會導致 MQPUT 或 MQPUT1 呼叫失敗，原因碼為 MQRC_PUBLICATION_FAILURE。 如果延伸訊息選取提供者無法剖析發佈，則會傳回原因碼 MQRC_CONTENT_ERROR ，且 MQPUT 或 MQPUT1 呼叫會失敗。

如果延伸訊息選取提供者無法使用或無法判斷訂閱者是否應該接收發佈，則會傳回原因碼 MQRC_SELECTION_NOT_AVAILABLE ，且 MQPUT 或 MQPUT1 呼叫會失敗。

使用內容過濾器建立訂閱且無法使用延伸訊息選取提供者時， MQSUB 呼叫會失敗，原因碼為 MQRC_SELECTION_NOT_AVAILABLE。 如果正在回復具有內容過濾器的訂閱，且延伸訊息選取提供者無法使用，則 MQSUB 呼叫會傳回 MQRC_SELECTION_NOT_AVAILABLE 警告，但容許回復訂閱。

相關參考

選取字串

非同步使用 IBM MQ 訊息

非同步使用會使用一組「訊息佇列介面 (MQI)」延伸規格， MQI 呼叫 MQCB 及 MQCTL 可讓 MQI 應用程式寫入來使用來自一組佇列的訊息。 透過呼叫 'unit of code' (由傳遞訊息的應用程式識別) 或代表訊息的記號，將訊息遞送至應用程式。

在最直接明確的應用程式環境中，程式碼單元由函數指標定義，但在其他環境中，程式碼單元可以由程式或模組名稱定義。

在訊息的非同步使用中，會使用下列術語:

訊息消費者
一種程式設計建構，可讓您定義當符合應用程式需求的訊息變成可用時，要以訊息來呼叫的程式或函數。

事件處理程式
一種程式設計建構，可讓您定義發生非同步事件 (例如佇列管理程式靜止) 時要呼叫的程式或函數。

回呼
用來參照「訊息消費者」或「事件處理程式」常式的一般術語。

非同步使用可以簡化新應用程式的設計和實作，特別是處理多個輸入佇列或訂閱的應用程式。 不過，如果您使用多個輸入佇列，並且以優先順序順序處理訊息，則會在每一個佇列內獨立觀察優先順序: 您可能會從一個佇列取得低優先順序訊息，然後從另一個佇列取得高優先順序訊息。 不保證跨多個佇列的訊息順序。 另請注意，如果您使用 API 結束程式，則可能需要將它們變更為包含 MQCB 和 MQCTL 呼叫。

下列圖解提供如何使用此功能的範例。

第 36 頁的圖 5 顯示多執行緒應用程式從兩個佇列耗用訊息。 此範例顯示遞送至單一函數的所有訊息。

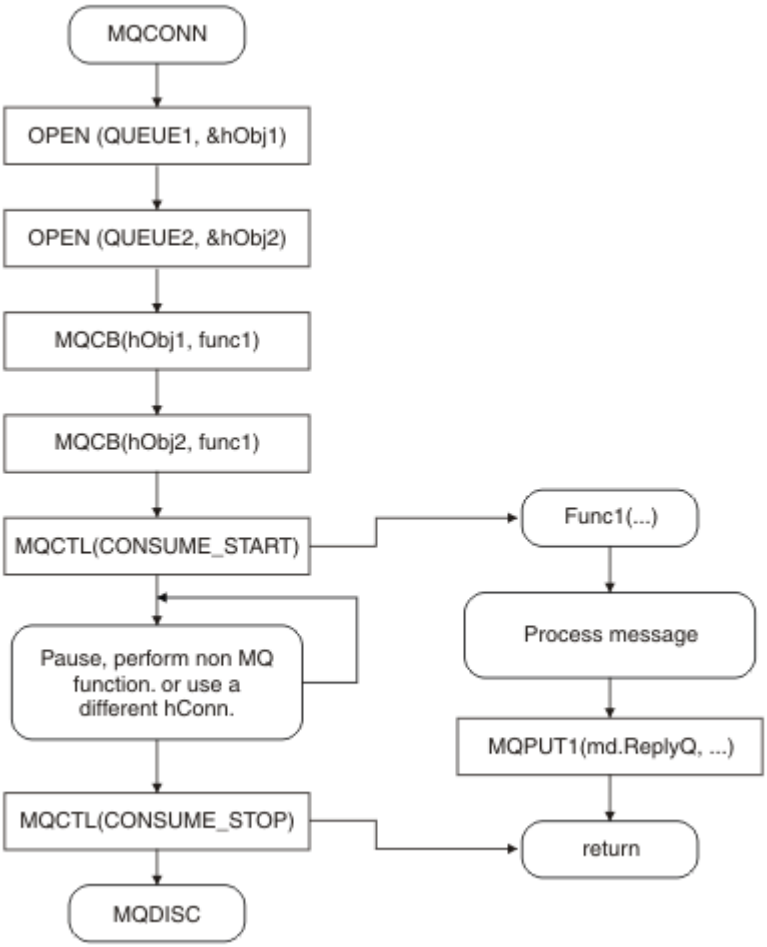


圖 *5:* 從兩個佇列耗用的標準訊息驅動應用程式

z/OS 在 z/OS 上，主要控制執行緒必須在結束之前發出 MQDISC 呼叫。 這可讓任何回呼執行緒結束並釋放系統資源。

第 37 頁的圖 6 此範例流程顯示單一執行緒應用程式耗用來自兩個佇列的訊息。 此範例顯示遞送至單一函數的所有訊息。

與非同步情況不同的是，在所有消費者都已自行取消啟動之前，控制不會回到 MQCTL 的發出者; 亦即，有一個消費者已發出 MQCTL STOP 要求或佇列管理程式靜止。

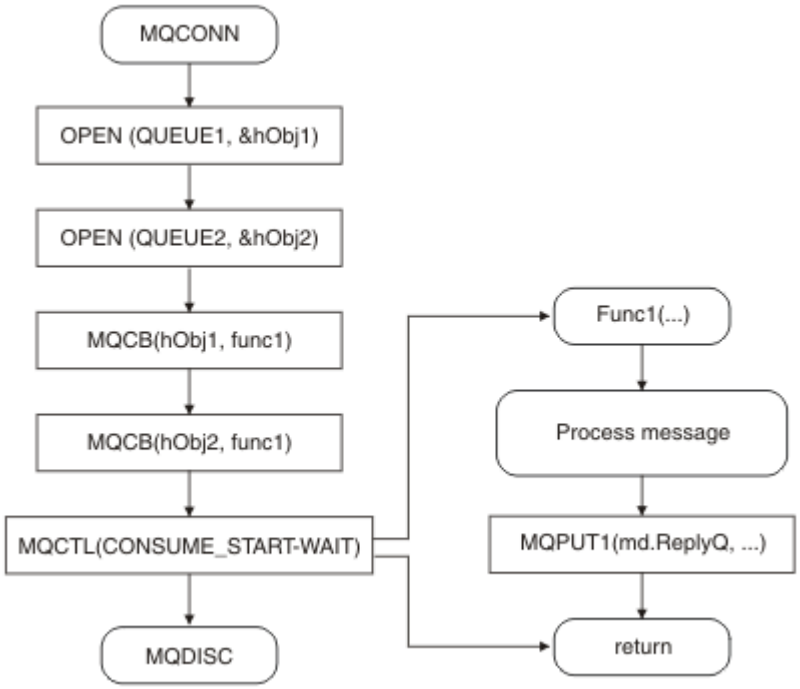


圖 *6:* 從兩個佇列耗用的單一執行緒訊息驅動應用程式

訊息群組

訊息可以在群組內出現，以容許排序訊息。

訊息群組容許將多個訊息標示為彼此相關，並將邏輯順序套用至群組 (請參閱 第 648 頁的『邏輯和實體排序』)。 在 多平台上， 訊息分段 可將大型訊息分成較小的區段。 放置到主題時，您無法使用分組或分段訊息。

群組內的階層如下:

群組
這是階層中的最高層次，由 *GroupId* 識別。 它包含一個以上包含相同 *GroupId* 的訊息。 這些訊息可以儲存在佇列上的任何位置。

註: 這裡使用術語 *message* 來表示佇列上的一個項目，例如由未指定 MQGMO_COMPLETE_MSG 的單一 MQGET 傳回。

第 37 頁的圖 7 顯示邏輯訊息群組:

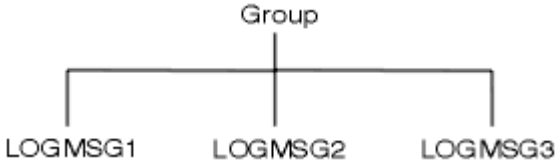


圖 *7:* 邏輯訊息群組

透過開啟佇列並指定 MQOO_BIND_ON_GROUP，您可以強制將群組中傳送至此佇列的所有訊息傳送至佇列的相同實例。 如需 BIND_ON_GROUP 選項的相關資訊，請參閱 處理訊息親緣性。

邏輯訊息
群組內的邏輯訊息由 *GroupId* 和 *MsgSeqNumber* 欄位識別。 對於群組內的第一個訊息，*MsgSeqNumber* 從 1 開始，如果訊息不在群組中，則欄位值為 1。

使用群組內的邏輯訊息來執行下列動作:

- 確定排序 (如果在傳輸訊息的情況下不保證這樣做)。
- 容許應用程式將類似訊息分組 (例如，必須全部由相同伺服器實例處理的訊息)。

除非分割成區段，否則群組內的每一則訊息都由一則實體訊息組成。 每一個訊息在邏輯上都是個別訊息，只有 MQMD 中的 *GroupId* 和 *MsgSeqNumber* 欄位需要與群組中的其他訊息產生任何關係。 MQMD 中的其他欄位是獨立的; 群組中所有訊息的部分欄位可能相同，而其他訊息則可能不同。 例如，群組中的訊息可以具有不同的格式名稱、CCSID 及編碼。

區段

區段是用來處理對於放置或取得應用程式或佇列管理程式 (包括訊息傳遞所透過的中間佇列管理程式) 而言太大的訊息。 如需相關資訊，請參閱第 662 頁的『訊息分段』。

個別訊息分成較小的訊息，稱為 區段。 訊息的區段由 *GroupId*、 *MsgSeqNumber* 和 *Offset* 欄位識別。 對於訊息內的第一個區段， *Offset* 欄位從零開始。

每一個區段由可能屬於某個群組的一個實體訊息組成 (第 38 頁的圖 8 顯示群組內訊息的範例)。 區段在邏輯上是單一訊息的一部分，因此 MQMD 中只有 *MsgId*、 *Offset* 和 *MsgFlags* 欄位應該在相同訊息的個別區段之間不同。 如果區段無法抵達，則會適當地傳回原因碼 MQRC_INCOMPLETE_GROUP 或 MQRC_INCOMPLETE_MSG 。

第 38 頁的圖 8 顯示一組邏輯訊息，其中部分已分段:

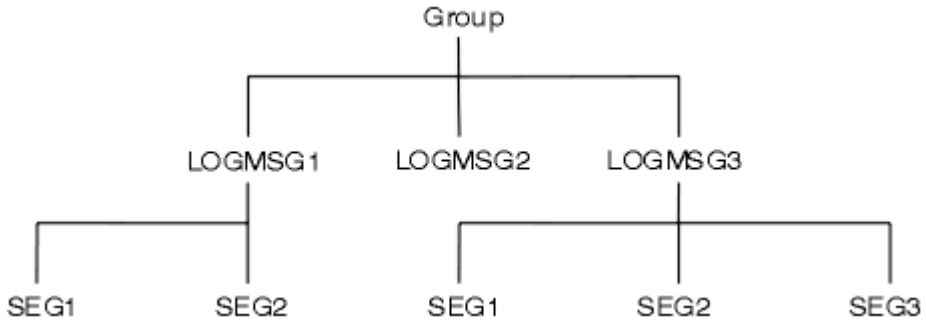


圖 *8:* 分段訊息

z/OS IBM MQ for z/OS 不支援分段。

您無法將分段或分組的訊息與「發佈/訂閱」搭配使用。

相關概念

第 662 頁的『訊息分段』
使用此資訊來瞭解分段訊息的相關資訊。 IBM MQ for z/OS 或使用 IBM MQ classes for JMS 的應用程式不支援此特性。

相關參考

第 648 頁的『邏輯和實體排序』
佇列上的訊息可以 實體 或 邏輯 順序發生 (在每一個優先順序層次內)。

MQMD-訊息描述子

訊息持續性

持續訊息會寫入日誌及佇列資料檔。 如果佇列管理程式在失敗之後重新啟動，它會根據需要從記載的資料回復這些持續訊息。 如果佇列管理程式停止，則不論停止是由於操作員指令或部分系統失敗，都會捨棄非持續的訊息。

z/OS 儲存在 z/OS 上連結機能 (CF) 中的非持續訊息是此情況的例外。 只要 CF 仍然可用，它們就會持續存在。

當您建立訊息時，如果您使用預設值來起始設定訊息描述子 (MQMD) ，則會從 MQOPEN 指令中所指定佇列的 **DefPersistence** 屬性取得訊息的持續性。 或者，您可以使用 MQMD 結構的 *Persistence* 欄位來設定訊息的持續性，以將訊息定義為持續性或非持續性。

當您使用持續訊息時，應用程式的效能會受到影響; 效果的範圍取決於機器 I/O 子系統的效能性質，以及您在每一個平台上使用同步點選項的方式:

- 在現行工作單元之外，每次放置及取得作業都會將持續訊息寫入磁碟。 請參閱 第 716 頁的『確定及取消工作單元』。
- z/OS ALW對於 IBM i 以外的所有平台，只有在確定工作單元時，才會記載現行工作單元內的持續訊息，且工作單元可以包含許多佇列作業。

非持續訊息可用於快速傳訊。 如需快速訊息的進一步相關資訊，請參閱 訊息安全 。

註: 在工作單元內寫入持續訊息，以及在單元或工作外部寫入持續訊息的組合，可能會對應用程式造成潛在的嚴重效能問題。 當兩個作業都使用相同的目標佇列時，尤其如此。

無法遞送的訊息

當佇列管理程式無法將訊息放置在佇列上時，您有各種選項。

您可以:

- 再次嘗試將訊息放置在佇列上。
- 要求將訊息傳回給傳送者。
- 將訊息放置在無法傳送郵件的佇列上。

如需相關資訊，請參閱第 871 頁的『處理程序化程式錯誤』。

已取消的訊息

在工作單元的控制下處理來自佇列的訊息時，工作單元可以由一或多個訊息組成。 如果發生取消，則會在佇列上恢復從佇列擷取的訊息，並且可以在另一個工作單元中重新處理這些訊息。 如果特定訊息的處理導致問題，則會再次取消工作單元。 這可能會導致處理迴圈。 已放入佇列中的訊息會從佇列中移除。

應用程式可以透過測試 MQMD 的 *BackoutCount* 欄位來偵測在此類迴圈中捕捉到的訊息。 應用程式可以更正狀況，或向操作員發出警告。

Multi取消計數一律在佇列管理程式重新啟動之後仍然存在。 **HardenGetBackout** 屬性的任何變更都會被忽略。

z/OS對於共用佇列，取消計數一律在重新啟動佇列管理程式之後仍然存在。 對於 z/OS 上的所有其他配置，若要確保專用佇列的取消計數在佇列管理程式重新啟動之後仍然存在，請將 *HardenGetBackout* 屬性設為 MQQA_BACKOUT_HARDENED; 否則，如果佇列管理程式必須重新啟動，它不會維護每一則訊息的精確取消計數。 以此方式設定屬性會增加額外處理的成本。

如需確定及取消訊息的相關資訊，請參閱 第 716 頁的『確定及取消工作單元』。

回覆目的地佇列及佇列管理程式

有時您可能會收到訊息以回應您傳送的訊息:

- 回應要求訊息的回覆訊息
- 關於非預期事件或期限的報告訊息
- 關於 COA (確認抵達) 或 COD (確認遞送) 事件的報告訊息
- 關於 PAN (正面動作通知) 或 NAN (負面動作通知) 事件的報告訊息

使用 MQMD 結構，在 *ReplyToQ* 欄位中指定您要將回覆及報告訊息傳送至其中的佇列名稱。 在 *ReplyToQMgr* 欄位中指定擁有回覆目的地佇列的佇列管理程式名稱。

如果您將 *ReplyToQMgr* 欄位保留空白，佇列管理程式會在佇列的訊息描述子中設定下列欄位的內容:

ReplyToQ
如果 *ReplyToQ* 是遠端佇列的本端定義，則 *ReplyToQ* 欄位會設為遠端佇列的名稱; 否則不會變更此欄位。

ReplyToQMgr
如果 *ReplyToQ* 是遠端佇列的本端定義，則 *ReplyToQMgr* 欄位會設為擁有遠端佇列的佇列管理程式名稱; 否則， *ReplyToQMgr* 欄位會設為應用程式所連接的佇列管理程式名稱。

註: 您可以要求佇列管理程式不只一次嘗試遞送訊息，也可以要求在訊息失敗時捨棄該訊息。 在無法遞送之後，如果不會捨棄訊息，遠端佇列管理程式會將訊息放到其無法傳送的郵件 (未遞送的訊息) 佇列中 (請參閱第 874 頁的『使用無法傳送的郵件 (無法遞送的訊息) 佇列』)。

訊息環境定義

訊息環境定義 資訊可讓擷取訊息的應用程式找出訊息的發送端。

擷取應用程式可能想要:

- 請檢查傳送端應用程式是否具有正確的權限層次
- 執行一些會計功能，以便它可以針對它必須執行的任何工作向傳送端應用程式收費
- 保留其使用的所有訊息的審核追蹤

當您使用 MQPUT 或 MQPUT1 呼叫將訊息放置在佇列上時，您可以指定佇列管理程式將部分預設環境定義資訊新增至訊息描述子。 具有適當權限層次的應用程式可以新增額外環境定義資訊。 如需如何指定環境定義資訊的相關資訊，請參閱第 635 頁的『控制訊息環境定義資訊』。

當產生下列類型的報告訊息時，佇列管理程式會使用使用者環境定義:

- 遞送時確認
- 期限

產生這些報告訊息時，會檢查使用者環境定義對報告目的地的 + put 及 + passid 權限。 如果使用者環境定義權限不足，則會將報告訊息放置在無法傳送郵件的佇列 (如果已定義的話)。 如果沒有無法傳送郵件的佇列，則會捨棄報告訊息。

所有環境定義資訊都儲存在訊息描述子的環境定義欄位中。 資訊的類型落在身分、原點及使用者環境定義資訊中。

身分環境定義 (identity context)

身分環境定義 資訊可識別第一次將訊息放置在佇列上的應用程式使用者。 適當授權的應用程式可以設定下列欄位:

- 佇列管理程式會在 *UserIdentifier* 欄位中填入識別使用者的名稱; 佇列管理程式可以執行此動作的方式，取決於應用程式執行所在的環境。
- 佇列管理程式會在 *AccountingToken* 欄位中填入它從放置訊息的應用程式所決定的記號或號碼。
- 應用程式可以使用 *ApplIdentityData* 欄位來取得他們想要包含的任何關於使用者的額外資訊 (例如，已加密密碼)。

當在 IBM MQ for Windows 下建立訊息時， Windows 系統安全 ID (SID) 會儲存在 *AccountingToken* 欄位中。 SID 可用來補充 *UserIdentifier* 欄位，以及建立使用者的認證。

如需佇列管理程式如何填入 *UserIdentifier* 及 *AccountingToken* 欄位的相關資訊，請參閱 UserIdentifier 及 AccountingToken 中這些欄位的說明。

將訊息從一個佇列管理程式傳遞至另一個佇列管理程式的應用程式也應該傳遞身分環境定義資訊，讓其他應用程式知道訊息發送端的身分。

原始環境定義

原始環境定義 資訊說明將訊息放置在目前儲存訊息之佇列上的應用程式。 訊息描述子包含原始環境定義資訊的下列欄位:

- *PutApplType* 定義放置訊息的應用程式類型 (例如， CICS 交易)。
- *PutApplName* 定義放置訊息的應用程式名稱 (例如，工作或交易的名稱)。
- *PutDate* 定義將訊息放入佇列的日期。

- *PutTime* 定義將訊息放入佇列的時間。
- *ApplOriginData* 定義應用程式想要包含的任何關於訊息來源的額外資訊。 例如，它可以由適當授權的應用程式設定，以指出身分資料是否受信任。

原始環境定義資訊通常由佇列管理程式提供。 「格林威治標準時間 (GMT)」用於 *PutDate* 和 *PutTime* 欄位。 請參閱 PutDate 和 PutTime 中這些欄位的說明。

具有足夠權限的應用程式可以提供自己的環境定義。 當單一使用者在處理其產生的訊息的每一個系統上具有不同的使用者 ID 時，這可保留帳戶資訊。

IBM MQ 物件

此資訊提供 IBM MQ 物件的詳細資料，包括: 佇列管理程式、佇列共用群組、佇列、管理主題物件、名稱清單、程序定義、鑑別資訊物件、通道、儲存類別、接聽器及服務。

佇列管理程式會定義這些物件的內容 (稱為屬性)。 這些屬性的值會影響 IBM MQ 處理這些物件的方式。 從應用程式中，您可以使用「訊息佇列介面 (MQI)」來控制這些物件。 從程式定址時，由 物件描述子 (MQOD) 來識別物件。

當您使用 IBM MQ 指令來定義、變更或刪除物件時，例如，佇列管理程式會檢查您是否具有執行這些作業所需的權限層次。 同樣地，當應用程式使用 MQOPEN 呼叫來開啟物件時，佇列管理程式會先檢查應用程式是否具有必要的權限層次，然後才容許存取該物件。 會對正在開啟的物件名稱進行檢查。

相關概念

第 635 頁的『控制訊息環境定義資訊』
當您使用 MQPUT 或 MQPUT1 呼叫將訊息放置在佇列上時，您可以指定佇列管理程式將部分預設環境定義資訊新增至訊息描述子。 具有適當權限層次的應用程式可以新增額外環境定義資訊。 您可以使用 MQPMO 結構中的選項欄位來控制環境定義資訊。

相關參考

第 628 頁的『與訊息環境定義相關的 MQOPEN 選項』
如果您想要在將環境定義資訊放入佇列時能夠將它與訊息相關聯，則必須在開啟佇列時使用其中一個訊息環境定義選項。

Windows 準備及執行 Microsoft Transaction Server 應用程式

若要準備 MTS 應用程式以作為 IBM MQ MQI client 應用程式執行，請針對您的環境適當遵循下列指示。

如需如何開發存取 IBM MQ 資源之 Microsoft Transaction Server (MTS) 應用程式的一般資訊，請參閱 IBM MQ 說明中心的 MTS 一節。

若要準備 MTS 應用程式以作為 IBM MQ MQI client 應用程式來執行，請針對應用程式的每一個元件執行下列其中一項:

- 如果元件使用 MQI 的 C 語言連結，請遵循 第 853 頁的『在 Windows 中準備 C 程式』 中的指示，但將元件與程式庫 mqicxa.lib 鏈結，而不是 mqic.lib。
- 如果元件使用 IBM MQ C++ 類別，請遵循 第 460 頁的『在 Windows 上建置 C++ 程式』 中的指示，但將元件鏈結至程式庫 imqx23vn.lib ，而不是 imqc23vn.lib。
- 如果元件使用 MQI 的 Visual Basic 語言連結，請遵循 第 856 頁的『在 Windows 中準備 Visual Basic 程式』 中的指示，但當您定義 Visual Basic 專案時，請在 **條件式編譯引數** 欄位中鍵入 MqType=3 。

IBM MQ 應用程式的設計考量

當您決定應用程式如何利用可供您使用的平台及環境時，您需要決定如何使用 IBM MQ 所提供的特性。

設計 IBM MQ 應用程式時，請考量下列問題和選項:

應用程式類型

您申請的目的是什麼? 如需您可以開發的不同應用程式類型的相關資訊，請參閱下列鏈結:

- 伺服器
- 用戶端

- 發佈/訂閱
- Web 服務
- 使用者結束程式、API 結束程式及可安裝的服務

此外，您也可以撰寫自己的應用程式，以自動管理 IBM MQ。 如需相關資訊，請參閱 IBM MQ 管理介面 (MQAI) 及 自動化管理作業。

程式設計語言

IBM MQ 支援多種不同的程式設計語言來撰寫應用程式。 如需相關資訊，請參閱 第 5 頁的『開發適用於 IBM MQ 的應用程式』。

多個平台的應用程式

您的應用程式會在多個平台上執行嗎? 您是否有策略可移至與您今天使用的平台不同的平台? 如果上述任一問題的答案是「是」，請確保為您的程式撰寫平台獨立性的程式碼。

例如，如果您使用 C，則在 ANSI 標準 C 中使用程式碼。 使用標準 C 程式庫函數，而不是相等的平台專用函數，即使平台專用函數更快速或更有效率也一樣。 例外是當程式碼中的效率至關重要時，當您應該使用 #ifdef 為這兩種狀況撰寫程式碼時。 例如:

```
#ifdef _AIX
    AIX specific code
#else
    generic code
#endif
```

佇列類型

您是要在每次需要時建立佇列，還是要使用已設定的佇列? 您要在完成使用佇列時刪除佇列，還是要再次使用它? 您要使用別名佇列來取得應用程式獨立性嗎? 若要查看支援哪些類型的佇列，請參閱 佇列。

z/OS 使用共用佇列、佇列共用群組及佇列共用群組叢集 (僅限 IBM MQ for z/OS)

當您將共用佇列與佇列共用群組搭配使用時，您可能想要利用增加的可用性、可調整性及工作量平衡。 如需相關資訊，請參閱 共用佇列及佇列共用群組 。

您也可能想要預估平均及尖峰訊息流程，並考量使用佇列共用群組叢集來分散工作量。 如需相關資訊，請參閱 共用佇列及佇列共用群組 。

使用佇列管理程式叢集

您可能想要利用簡化系統管理，以及使用叢集時可能增加的可用性、可調整性及工作量平衡。

訊息類型

您可能想要針對簡式訊息使用資料包，但針對其他狀況要求訊息 (您預期會有回覆)。 您可能想要將不同的優先順序指派給部分訊息。 如需設計訊息的相關資訊，請參閱 第 49 頁的『訊息的設計技術』。

使用發佈/訂閱或點對點傳訊

使用發佈/訂閱傳訊，傳送端應用程式會將想要在 IBM MQ 訊息中共用的資訊傳送至 IBM MQ 發佈? 訂閱所管理的標準目的地，並讓 IBM MQ 處理該資訊的配送。 目標應用程式不需要知道它所接收資訊的來源，它只會登錄一個以上主題的興趣，並在可用時接收該資訊。 如需發佈/訂閱傳訊的相關資訊，請參閱發佈/訂閱傳訊。

使用點對點傳訊，傳送端應用程式會將訊息傳送至特定佇列，並從中得知接收端應用程式將擷取該訊息。 接收端應用程式會從特定佇列取得訊息，並處理其內容。 應用程式通常會同時作為傳送端和接收端，將查詢傳送至另一個應用程式，並接收回應。

控制 IBM MQ 程式

您可能想要自動啟動部分程式，或讓程式等待特定訊息到達佇列 (使用 IBM MQ 觸發 功能，請參閱 第 725 頁的『使用觸發程式啟動 IBM MQ 應用程式』)。 或者，當佇列上的訊息處理速度不夠快時，您可能想要啟動另一個應用程式實例 (使用 IBM MQ 檢測事件 特性，如 檢測事件 中所述)。

在 IBM MQ 用戶端上執行應用程式

在用戶端環境中支援完整 MQI，而且幾乎任何以程序化語言撰寫的 IBM MQ 應用程式都可以重新鏈結，以在 IBM MQ MQI client 上執行。 將 IBM MQ MQI client 上的應用程式鏈結至 MQIC 程式庫，而不是 MQI 程式庫。 z/OS 不支援 z/OS 上的取得 (信號)。

註: 在 IBM MQ 用戶端上執行的應用程式可以同時連接至多個佇列管理程式，或在 MQCONN 或 MQCONNX 呼叫中使用具有星號 (*) 的佇列管理程式名稱。 如果您要鏈結至佇列管理程式檔案庫而非用戶端檔案庫，請變更應用程式，因為此功能將無法使用。

如需相關資訊，請參閱 第 771 頁的『在 IBM MQ MQI client 環境中執行應用程式』。

應用程式效能

設計決策可能會影響您的應用程式效能，如需加強 IBM MQ 應用程式效能的建議，請參閱 第 50 頁的『應用程式設計和效能考量』 IBM i 及 第 53 頁的『IBM i 應用程式的設計和效能考量』。

進階 IBM MQ 技術

對於更進階的應用程式，您可能想要使用一些進階 IBM MQ 技術，例如關聯回覆，以及產生和傳送 IBM MQ 環境定義資訊。 如需相關資訊，請參閱 第 51 頁的『進階應用程式的設計技術』。

保護資料安全並維護其完整性

您可以使用隨訊息一起傳遞的環境定義資訊，來測試訊息是否已從可接受的來源傳送。 您可以使用 IBM MQ 或作業系統所提供的同步點機能，以確保您的資料與其他資源保持一致 (如需進一步詳細資料，請參閱 第 716 頁的『確定及取消工作單元』)。 您可以使用 IBM MQ 訊息的 持續性 特性來確保遞送重要訊息。

測試 IBM MQ 應用程式

IBM MQ 程式的應用程式開發環境與任何其他應用程式的應用程式開發環境沒有不同，因此您可以使用相同的開發工具及 IBM MQ 追蹤機能。

z/OS 使用 IBM MQ for z/OS 來測試 CICS 應用程式時，您可以使用 CICS 執行診斷機能 (CEDF)。 CEDF 會設陷每一個 MQI 呼叫的進入及結束，以及對所有 CICS 服務的呼叫。 此外，在 CICS 環境中，您可以撰寫跨 API 結束程式，以在每次 MQI 呼叫之前及之後提供診斷資訊。 如需如何執行此作業的相關資訊，請參閱第 745 頁的『在 IBM MQ for z/OS 上使用及撰寫應用程式』。

IBM i 測試 IBM i 應用程式時，您可以使用標準 Debugger。 若要啟動此作業，請使用 STRDBG 指令。

處理異常狀況和錯誤

您需要考量如何處理無法遞送的訊息，以及如何解決佇列管理程式向您報告的錯誤狀況。 對於部分報告，您必須在 MQPUT 上設定報告選項。

相關概念

IBM MQ 技術概觀

第 54 頁的『z/OS 應用程式的設計和效能考量』
應用程式設計是影響效能的最重要因素之一。 請利用這個主題來瞭解效能所涉及的一些設計因素。

第 5 頁的『開發適用於 IBM MQ 的應用程式』
您可以開發應用程式來傳送及接收訊息，以及管理佇列管理程式和相關資源。 IBM MQ 支援以許多不同語言及架構撰寫的應用程式。

第 6 頁的『應用程式開發概念』
您可以使用選擇的程序化或物件導向語言來撰寫 IBM MQ 應用程式。 在開始設計及撰寫 IBM MQ 應用程式之前，請先熟悉基本 IBM MQ 概念。

第 603 頁的『撰寫佇列作業的程序化應用程式』
請利用這項資訊來瞭解如何撰寫佇列作業應用程式、連接佇列管理程式、發佈/訂閱，以及開啟和關閉物件。

第 765 頁的『撰寫用戶端程序化應用程式』
使用程序化語言在 IBM MQ 上撰寫用戶端應用程式所需的資訊。

第 464 頁的『開發 .NET 應用程式』
IBM MQ classes for .NET 容許 .NET 應用程式以 IBM MQ MQI client 身分連接至 IBM MQ，或直接連接至 IBM MQ 伺服器。

第 438 頁的『開發 C++ 應用程式』
IBM MQ 提供相當於 IBM MQ 物件的 C++ 類別，以及相當於陣列資料類型的其他類別。 它提供許多無法透過 MQI 使用的特性。

第 68 頁的『使用 IBM MQ classes for JMS/Jakarta Messaging』

IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 是 IBM MQ 隨附的 Java 傳訊提供者。除了實作 JMS 和 Jakarta Messaging 規格中定義的介面，這些傳訊提供者也會將兩組延伸新增至 Java 傳訊 API。

第 290 頁的『使用 IBM MQ classes for Java』
在 Java 環境中使用 IBM MQ 。 IBM MQ classes for Java 可讓 Java 應用程式以 IBM MQ 用戶端身分連接至 IBM MQ ，或直接連接至 IBM MQ 佇列管理程式。

以支援的程式設計語言指定應用程式名稱

在 IBM MQ 9.2.0 之前，您可以在 Java 或 JMS 用戶端應用程式上指定應用程式名稱。 從 IBM MQ 9.2.0 ，此特性已延伸至 IBM MQ for Multiplatforms 上的其他程式設計語言。

如何使用應用程式名稱

應用程式名稱來自:

- runmqsc DISPLAY CONN APPLTAG
- runmqsc DISPLAY QSTATUS TYPE (HANDLE) APPLTAG
- runmqsc DISPLAY CHSTATUS RAPPLTAG
- MQMD.PutApplName
- 應用程式活動追蹤

當配置應用程式活動追蹤時，也會使用應用程式名稱。 非 Java 應用程式的預設應用程式名稱是執行檔的截斷名稱，但 Windows 和 IBM i 除外。

Windows 在 Windows 上，預設名稱是完整執行檔名稱，在左側截斷為 28 個字元。

IBM i 在 IBM i 上，預設名稱是工作名稱。

對於 Java 應用程式，它是類別名稱，其字首是在左側截斷為 28 個字元的套件名稱。

如需相關資訊，請參閱 PutAppl 名稱。

從 IBM MQ 9.2.0 開始， IBM MQ for Multiplatforms 上的應用程式可以透過管理方式或使用各種程式設計方法來設定其應用程式名稱。 當您配置應用程式活動追蹤或從各種 **runmqsc** 指令輸出時，這可讓應用程式提供更有意義的獨立式平台名稱。

從 IBM MQ 9.2.0 開始，您可以在統一叢集中重新平衡應用程式。 會使用有意義的應用程式名稱來達成此目的。

支援的字元

如需如何指定應用程式名稱的相關資訊，請參閱 第 45 頁的『建議的應用程式名稱字元』 。

程式設計語言

如需應用程式解析成 C 及其他程式設計語言之 IBM MQ 程式庫如何提供應用程式名稱的相關資訊，請參閱 第 46 頁的『程式設計語言連線』 。

受管理 .NET 應用程式

如需受管理 .NET 應用程式如何提供應用程式名稱的相關資訊，請參閱 第 47 頁的『受管理 .NET 應用程式』 。

XMS 應用程式

如需 XMS 應用程式如何提供應用程式名稱的相關資訊，請參閱 第 48 頁的『XMS 應用程式』 。

Java 和 JMS 連結應用程式

ALW

如需 Java 和 JMS 應用程式如何提供應用程式名稱的相關資訊，請參閱 第 48 頁的『Java 和 JMS 連結應用程式』。

相關概念

應用程式活動追蹤

關於統一叢集

相關參考

MQCNO

IBM i 上的 MQCNO

在支援的程式設計語言中使用應用程式名稱

使用此資訊來瞭解如何以 IBM MQ 支援的各種語言來選取應用程式名稱。

建議的應用程式名稱字元

應用程式名稱必須採用佇列管理程式欄位的 **CodedCharSetId** 屬性所提供的字集。 如需此屬性的相關資訊，請參閱 佇列管理程式的屬性。

不過，如果應用程式以 IBM MQ MQI client 身分執行，則應用程式名稱必須採用用戶端的字集及編碼。

若要確保在佇列管理程式之間順利轉移應用程式名稱，並容許透過資源監視主題進行應用程式資源監視，應用程式名稱應該只包含單位元組可列印字元。

附註:

- You should also avoid the use of forward slash and ampersand characters in application names.
- V 9.3.0 You should avoid use of the ampersand character in application names. System topic STATAPP metrics for application names containing an ampersand will not be produced.

這會將名稱限制為:

- 英數字元: `A-Z`、 `a-z` 及 `0-9`

  註: 在使用 EBCDIC Katakana 的系統上，您不應在應用程式名稱中使用小寫 `a-z` 字元。
- 空格字元
- EBCDIC 中的可列印字元不變: `+ < = > % * ' ( ) , _ - . : ; ?`
- V 9.3.0 /字元。 When subscribing to activity trace or STATAPP system topic metrics for an application whose name contains a forward slash, you must replace any forward slash characters with an ampersand character. 例如，若要接收稱為 "DEPT1/APPS/STOCKQUOTE" 之應用程式的 STATAPP 度量值，您必須訂閱主題字串 "$SYS/MQ/INFO/QMGR/QMBASIC/Monitor/STATAPP/DEPT1&APPS&STOCKQUOTE/INSTANCE"。 The amqsact and amqsrua sample applications will automatically convert forward slash characters to ampersands when creating their subscriptions.

如何設定字元

下表彙總以 IBM MQ 支援的各種語言來選擇應用程式名稱的方法。 選擇名稱所使用的方法依優先順序排列，最高優先。

	C 連結和用戶端	Java 連結和用戶端	JMS 連結和用戶端	受管理 .NET 用戶端	未受管理的 .NET 連結和用戶端	受管理 XMS 用戶端	未受管理。XMS 連結和用戶端
連線內容置換		Java 連線內容置換		。NET 連線內容置換	。NET 連線內容置換		
已置換內容		Java 置換內容		。NET 置換內容	。NET 置換內容		
MQEnvironment		Java MQEnvironment		。NET MQEnvironment	。NET MQEnvironment		
Connection Factory 內容			Connection Factory 內容			Connection Factory 內容	Connection Factory 內容
JMSAdmin			JMSAdmin			JMSAdmin	JMSAdmin
MQCNO	連線選項						
環境變數	環境變數				環境變數		環境變數
mqclient.ini (僅適用於用戶端連線)	用戶端連線				用戶端連線		用戶端連線
Java 類別名稱		Java 類別名稱	Java 類別名稱				
預設名稱	預設名稱			。NET 預設名稱	。NET 預設名稱	。NET 預設名稱	。NET 預設名稱

註: C 連結和用戶端直欄也適用於下列程式設計語言:

- COBOL
- Assembler
- Visual Basic
- RPG

程式設計語言連線

以 C 及其他程式設計語言解析至 IBM MQ 程式庫的應用程式，可以透過下列方式提供應用程式名稱。連線方法以優先順序列出，從最高開始。

Multi **連線選項**

- ALW MQCNO

註: z/OS 當連接至 IBM MQ for z/OS 佇列管理程式時，您只能使用用戶端模式連線或使用 IBM MQ classes for JMS 或 IBM MQ classes for Java 應用程式來設定應用程式名稱。

- IBM i IBM i 上的 MQCNO

ALW 環境變數

如果您尚未選取應用程式名稱，則可以使用 **MQAPPLNAME** 環境變數來識別與佇列管理程式的連線。 例如:

```
export MQAPPLNAME=ExampleApplName
```

請注意，只會使用前 28 個字元，且這些字元不能全為空白或空值。

註: 此屬性僅適用於支援的程式設計語言、未受管理的 .NET 及未受管理的 XMS 連線。

ALW 用戶端配置檔

如果您尚未選取應用程式名稱，且連線是用戶端連線，則可以在用戶端配置檔 (例如， `mqclient.ini`) 中指定下列資訊，以識別與佇列管理程式的連線。

```
Connection:
    ApplName=ExampleApplName
```

附註:

1. 只會使用前 28 個字元，且這些字元不能全為空白或空值。
2. 此屬性僅適用於受支援程式設計語言、未受管理 .NET 及未受管理 XMS 連線上的用戶端連線。

如需相關資訊，請參閱 IBM MQ MQI client 配置檔 `mqclient.ini`。

預設名稱

如果您仍未選擇應用程式名稱，則會繼續使用預設名稱，其包含的路徑和執行檔名稱與作業系統顯示的一樣多。 如需相關資訊，請參閱 PutAppl 名稱。

受管理 .NET 應用程式

受管理 .NET 應用程式可以透過下列方式提供應用程式名稱。

連線方法以優先順序列出，從最高開始。

連線內容置換

您可以使用下列方式，將連線詳細資料置換檔案提供給應用程式:

```
<appSettings>
     <add key="overrideConnectionDetails" value="true" />
     <add key="overrideConnectionDetailsFile" value="<location>" />
</appSettings>
```

`overrideConnectionDetailsFile` 指定的檔案包含以 `mqj` 作為字首的內容清單。 應用程式需要定義 **`mqj.APPNAME`** 內容，其中 **`mqj.APPNAME`** 內容的值指定用來識別佇列管理程式連線的名稱。

只會使用名稱的前 28 個字元。 例如:

```
mqj.APPNAME=ExampleApplName
```

已置換內容

已使用值 *APPNAME* 定義常數 **MQC.APPNAME_PROPERTY**。 您現在可以只使用名稱的前 28 個字元，將此內容傳遞至 **MQQueueManager** 建構子。 例如：

```
Hashtable properties = new Hashtable();
properties.Add( MQC.APPNAME_PROPERTY, "ExampleApplName" );
MQQueueManager qMgr = new MQQueueManager("qmgrname", properties);
```

如需相關資訊，請參閱 第 546 頁的『.NET 中的受管理及未受管理作業』。

MQEnvironment

AppName 內容會新增至 **MQEnvironment** 類別，且只會使用前 28 個字元。 例如：

```
MQEnvironment.AppName = "ExampleApplName";
```

預設名稱

如果您未使用上述文字中說明的任何方法來提供應用程式名稱，則應用程式名稱會自動設為執行檔名稱 (以及符合的路徑)。

XMS 應用程式

連線方法以優先順序列出，從最高開始。

Connection Factory 內容

XMS 應用程式可以使用 **XMSC.WMQ_APPLICATIONNAME** 內容 ("*XMSC_WMQ_APPNAME*") 在 Connection Factory 上提供應用程式名稱 以類似於 JMS 的方式。 您最多可以指定 28 個字元。

如需相關資訊，請參閱第 553 頁的『XMS .NET 建立受管理物件』和第 560 頁的『XMS 訊息的內容』。

JMSAdmin

在管理工具中，此內容簡稱為 "**APPLICATIONNAME**" 或 "**APPNAME**"。

Java 和 JMS 連結應用程式

連線方法以優先順序列出，從最高開始。

ALW Java 和 JMS 用戶端應用程式已可以指定應用程式名稱，且已利用 MQCNO **ApplName** 欄位在 IBM MQ for Multiplatforms 上延伸至連結應用程式。

連線內容置換

Application name 內容已新增至您可以置換的連線內容清單。 如需相關資訊，請參閱 使用 IBM MQ 連線內容置換。

小心: IBM MQ classes for Java 和 .NET 的連線內容和使用「連線內容置換」檔案的方式相同。

已置換內容

已使用值 *APPNAME* 定義常數 **MQC.APPNAME_PROPERTY**。 您現在可以只使用名稱的前 28 個字元，將此內容傳遞至 **MQQueueManager** 建構子。 如需相關資訊，請參閱 在 IBM MQ classes for Java 中使用連線內容置換。

MQEnvironment

AppName 內容會新增至 **MQEnvironment** 類別，且只會使用前 28 個字元。

如需相關資訊，請參閱 第 314 頁的『為 IBM MQ classes for Java 設定 IBM MQ 環境』。

Java 類別名稱

如果您未以前述文字中的任何方法提供應用程式名稱，則應用程式名稱衍生自主要類別名稱。

如需相關資訊，請參閱 第 314 頁的『為 IBM MQ classes for Java 設定 IBM MQ 環境』。

小心: IBM i 在 IBM i 上，無法查詢 main 類別名稱，因此改用 IBM MQ client for Java 。

相關概念

第 314 頁的『為 IBM MQ classes for Java 設定 IBM MQ 環境』
若要讓應用程式以用戶端模式連接至佇列管理程式，應用程式必須指定通道名稱、主機名稱及埠號。

相關參考

MQCNO
IBM i 上的 MQCNO

訊息的設計技術

協助您設計訊息的考量，包括選取元和訊息內容的考量。

在設計階段要考慮的事項

當您使用 MQI 呼叫將訊息放入佇列時，會建立訊息。 作為呼叫的輸入，您可以在 訊息描述子 (MQMD) 中提供部分控制資訊，以及您要傳送至另一個程式的資料。 但在設計階段，您需要考量下列各項，因為它們會影響您建立訊息的方式:

要使用的訊息類型

您是否設計可傳送訊息的簡式應用程式，然後不採取進一步動作? 還是你要求回答一個問題? 如果您是詢問問題，則可以在訊息描述子中包括您要接收回覆的佇列名稱。

您要要求和回覆訊息同步嗎? 這意味著您設定回覆的逾時期間來回答您的要求，如果您未在該期間內收到回覆，則會將它視為錯誤。

或者，您是否偏好非同步工作，以便您的處理程序不必取決於特定事件 (例如一般計時信號) 的發生?

另一個考量是您是否在工作單元內具有所有訊息。

指派不同的優先順序給訊息

您可以指派優先順序值給每一則訊息，並定義佇列，讓它依其優先順序來維護其訊息。 如果您這麼做，當另一個程式從佇列擷取訊息時，它一律會取得具有最高優先順序的訊息。 如果佇列未依優先順序維護其訊息，則從佇列擷取訊息的程式會依訊息新增至佇列的順序來擷取訊息。

程式也可以使用佇列管理程式在將訊息放入佇列時所指派的 ID 來選取訊息。 或者，您可以為每一則訊息產生自己的 ID。

重新啟動佇列管理程式對訊息的影響

佇列管理程式會保留所有持續訊息，並在重新啟動時，必要時從 IBM MQ 日誌檔回復它們。 不會保留非持續訊息及暫時動態佇列。 您不想要捨棄的任何訊息在建立時必須定義為持續的訊息。 在 AIX and Linux 系統上撰寫 IBM MQ for Windows 或 IBM MQ 的應用程式時，請確定您知道如何在日誌檔配置方面設定系統，以減少設計將執行至日誌檔限制的應用程式的風險。

z/OS 因為共用佇列上的訊息 (僅適用於 IBM MQ for z/OS) 會保留在連結機能 (CF) 中，只要 CF 仍然可用，在佇列管理程式重新啟動時，就會保留非持續訊息。 如果 CF 失敗，則會遺失非持續訊息。

將您自己的相關資訊提供給訊息收件者

通常，佇列管理程式會設定使用者 ID ，但適當授權的應用程式也可以設定此欄位，以便您可以併入自己的使用者 ID ， 以及接收程式可用於帳戶或安全目的的其他資訊。

接收佇列數量

Multi 如果訊息可能需要放置在數個佇列上，您可以發佈至主題或配送清單。

z/OS 如果訊息可能需要放置在數個佇列上，您可以發佈至主題。

選取元和訊息內容

訊息可以與主要訊息有效負載一起具有相關聯的 meta 資料。 這些訊息內容在提供其他資料時非常有用。

瞭解這個額外資料有兩個方面很重要:

- 這些內容不受 Advanced Message Security (AMS) 保護。 如果您要使用 AMS 來保護資料，請將它放置在有效負載中，而不是訊息內容中。
- 這些內容可用來執行訊息的選取。

請務必注意，使用選取元會岔斷先進先出的標準訊息慣例。 由於佇列管理程式已針對此工作量進行最佳化，基於效能原因，不建議提供複式選取器。 佇列管理程式不會儲存訊息內容的索引，因此搜尋訊息必須是線性搜尋。 佇列越深，選取元越複雜，符合訊息的選取元會對效能產生負面影響的機率越低。

如果需要複雜的選擇，建議使用任何應用程式或處理引擎 (例如 IBM Integration Bus) 來過濾訊息至不同的目的地。 或者，使用主題階層可能很有用。

註: IBM MQ classes for Java 不支援使用選取元，如果您想要使用選取元，則應該透過 JMS API 來執行這些動作。

應用程式設計和效能考量

有許多方式不良的程式設計會影響效能。 這些可能難以偵測，因為程式本身可能表現良好，但會影響其他作業的效能。 本主題說明 IBM MQ 呼叫的程式特有的數個問題。

以下幾個構想可協助您設計有效率的應用程式:

- 設計您的應用程式，以便處理程序與使用者的思考時間平行進行:
  - 顯示畫面，並容許使用者在應用程式仍在起始設定時開始鍵入。
  - 從不同伺服器平行取得您需要的資料。
- 如果您要重複使用連線和佇列，而不是反覆地開啟和關閉、連接及中斷連線，請保持連線和佇列開啟。
- 不過，只放置一則訊息的伺服器應用程式應該使用 MQPUT1。
- 佇列管理程式已針對大小介於 4 KB 到 100 KB 之間的訊息進行最佳化。 非常大的訊息沒有效率; 最好傳送 100 則訊息 (每則 1 MB)，而不是單一 100 MB 訊息。 非常小的訊息也沒有效率。 佇列管理程式對單位元組訊息執行的工作量與對 4 KB 訊息執行的工作量相同。
- 將您的訊息保存在工作單元內，以便可以同步確定或取消它們。
- 對於不需要可回復的訊息，請使用非持續性選項。
- 如果您需要將訊息傳送至許多目標佇列，請考慮使用配送清單。

訊息長度的效果

訊息中的資料量可能會影響處理訊息之應用程式的效能。 若要從應用程式達到最佳效能，請只傳送訊息中的必要資料。 例如，在銀行帳戶借項的要求中，唯一可能需要從用戶端傳遞至伺服器應用程式的資訊是帳戶號碼及借項金額。

訊息持續性的效果

通常會記載持續訊息。 記載訊息會降低應用程式的效能，因此僅針對必要資料使用持續訊息。 如果在佇列管理程式停止或失敗時可以捨棄訊息中的資料，請使用非持續訊息。

z/OS 當回復日誌空間不足以記錄作業時，持續訊息的 MQPUT 及 MQGET 作業將會封鎖。 這類狀況在佇列管理程式工作日誌中由訊息 CSQJ110E 及 CSQJ111A 指出。 確保有適當的監視處理程序，以便管理及避免這類狀況。

搜尋特定訊息

MQGET 呼叫通常會從佇列中擷取第一個訊息。 如果您使用訊息描述子中的訊息及相關性 ID (*MsgId* 及 *CorrelId*) 來指定特定訊息，則佇列管理程式必須搜尋佇列，直到找到該訊息為止。 以此方式使用 MQGET 呼叫會影響應用程式的效能。

包含不同長度訊息的佇列

如果您的應用程式無法使用固定長度的訊息，請動態增加並縮減緩衝區，以符合一般訊息大小。 如果應用程式發出因緩衝區太小而失敗的 MQGET 呼叫，則會傳回訊息資料的大小。 將程式碼新增至應用程式，以便相應地調整緩衝區大小，並重新發出 MQGET 呼叫。

註: 如果您未明確設定 **MaxMsgLength** 屬性，則會預設為 4 MB，如果使用此屬性來影響應用程式緩衝區大小，可能會非常無效。

同步點的頻率

在同步點內發出大量 MQPUT 或 MQGET 呼叫而未確定它們的程式可能會導致效能問題。 受影響的佇列可能填滿目前無法存取的訊息，而其他作業可能正在等待取得這些訊息。 這涉及儲存體，以及與嘗試取得訊息之作業相關聯的執行緒。

使用 MQPUT1 呼叫

僅當您有單一訊息放置在佇列上時，才使用 MQPUT1 呼叫。 如果您要放置多個訊息，請使用 MQOPEN 呼叫，後面接著一系列 MQPUT 呼叫和單一 MQCLOSE 呼叫。

使用中的執行緒數目

Windows 對於 IBM MQ for Windows，應用程式可能需要大量執行緒。 每一個佇列管理程式處理程序都會配置容許的應用程式執行緒數目上限。

應用程式可能使用太多執行緒。 請考量應用程式是否考量此可能性，以及它是否採取動作來停止或報告此類型的出現項目。

將持續訊息置於同步點之下

持續訊息應該置於同步點之下。 這是因為在同步點之外取得持續訊息時，如果取得失敗，應用程式無法知道訊息是否已從佇列取得，以及如果已取得訊息，是否也會遺失。 在同步點下取得持續訊息時，如果有任何失敗，則會回復交易，且持續訊息不會遺失，因為它仍在佇列上。

同樣地，放置持續訊息時，請將它們置於同步點之下。 將持續訊息置於同步點及取得持續訊息的另一個原因是 IBM MQ 中的持續訊息碼已針對同步點進行大量最佳化。 因此，在同步點下放置及取得持續訊息比在同步點之外放置及取得持續訊息更快。

如果您的應用程式將持續訊息放在同步點之外，佇列管理程式會檢查它是否可以代表應用程式建立隱含的同步點。 如果佇列管理程式可以這麼做，它會包含在該同步點內的放置，並自動確定它。 如需更詳細的說明，請參閱 第 722 頁的『Multiplatforms 上的隱含同步點』。

不過，在同步點外部放置及取得非持續訊息會更快，因為 IBM MQ 中的非持續程式碼已針對在同步點外部進行最佳化。 放置及取得持續訊息會以磁碟速度執行，因為持續訊息會持續保存至磁碟。 不過，放置及取得非持續訊息會以 CPU 速度執行，因為即使使用同步點也不會涉及磁碟寫入。

如果應用程式正在取得訊息，且事先不知道它們是否持續存在，則可以使用 GMO 選項 MQGMO_SYNCPOINT_IF_PERSISTENT。

進階應用程式的設計技術

設計更進階的應用程式時，您可能想要考量一些技術，例如等待訊息、關聯回覆、設定及使用環境定義資訊、自動啟動應用程式、產生報告，以及在使用叢集作業時移除訊息親緣性。

對於簡式 IBM MQ 應用程式，您需要決定要在應用程式中使用哪些 IBM MQ 物件，以及要使用哪些類型的訊息。 對於更進階的應用程式，您可能想要使用下列各節中引進的部分技術。

等待訊息

負責處理佇列的程式可以等待下列訊息:

- 等待訊息到達或指定的時間間隔到期 (請參閱 第 666 頁的『等待訊息』)。

- z/OS 僅在 IBM MQ for z/OS 上，設定信號，以便在訊息到達時通知程式。 如需相關資訊，請參閱 第 667 頁的『信號 (signaling)』。
- 建立訊息到達時要驅動的回呼結束程式; 請參閱 第 35 頁的『非同步使用 IBM MQ 訊息』。
- 在佇列上定期呼叫以查看訊息是否已到達 (*polling*)。 通常不建議這樣做，因為它可能會影響效能。

關聯回覆

在 IBM MQ 應用程式中，當程式收到要求它執行某些工作的訊息時，通常會將一或多個回覆訊息傳送給要求者。

為了協助要求者將這些回覆與其原始要求相關聯，應用程式可以在每一個訊息的描述子中設定 相關性 *ID* 欄位。 然後，程式會將要求訊息的訊息 ID 複製到其回覆訊息的相關性 ID 欄位。

設定及使用環境定義資訊

環境定義資訊 用於將訊息與產生訊息的使用者相關聯，以及用於識別產生訊息的應用程式。 這類資訊有助於安全、會計、審核及問題判斷。

當您建立訊息時，您可以指定一個選項，要求佇列管理程式讓預設環境定義資訊與您的訊息產生關聯。

如需使用及設定環境定義資訊的相關資訊，請參閱 第 40 頁的『訊息環境定義』。

自動啟動 IBM MQ 程式

使用 IBM MQ 觸發 ，在訊息到達佇列時自動啟動程式。

您可以在佇列上設定觸發條件，讓程式開始處理該佇列:

- 每次訊息到達佇列時
- 當第一個訊息到達佇列時
- 當佇列上的訊息數達到預先定義的數目時

如需觸發的相關資訊，請參閱 第 725 頁的『使用觸發程式啟動 IBM MQ 應用程式』。 觸發只是自動啟動程式的一種方式。 例如，您可以使用非 IBM MQ 機能，在計時器上自動啟動程式。

Multi 在 多平台上， IBM MQ 可以定義服務物件，以在佇列管理程式啟動時啟動 IBM MQ 程式; 請參閱 服務物件。

產生 IBM MQ 報告

您可以在應用程式內要求下列報告:

- 異常狀況報告
- 到期報告
- 確認到達 (COA) 報告
- 確認交付 (COD) 報告
- 正面動作通知 (PAN) 報告
- 負面動作通知 (NAN) 報告

說明請見第 17 頁的『報告訊息』。

叢集和訊息親緣性

在開始使用具有多個相同佇列定義的叢集之前，請檢查您的應用程式，以查看是否有任何需要交換相關訊息的應用程式。

在叢集內，訊息可以遞送至管理適當佇列實例的任何佇列管理程式。 因此，具有訊息親緣性的應用程式的邏輯可能會不設定。

例如，您可能有兩個應用程式依賴以問答形式在它們之間流動的一系列訊息。 請務必將所有問題傳送至相同的佇列管理程式，並將所有回答傳回至其他佇列管理程式。 在此狀況下，工作量管理常式務必不要將訊息傳送至剛管理適當佇列實例的任何佇列管理程式。

可能的話，請移除親緣性。 移除訊息親緣性可改善應用程式的可用性及可調整性。

如需相關資訊，請參閱 處理訊息親緣性。

IBM i 應用程式的設計和效能考量

使用此資訊來瞭解應用程式設計、執行緒及儲存體如何影響效能。

此資訊分成兩個區段:

- 第 53 頁的『應用程式設計考量』
- 第 54 頁的『特定的效能問題』

應用程式設計考量

有許多方式不良的程式設計會影響效能。 這些問題可能難以偵測，因為程式看起來執行良好，同時會影響其他作業的效能。 下列各節說明發出 IBM MQ for IBM i 呼叫的程式特有的數個問題。

如需應用程式設計的相關資訊，請參閱 第 41 頁的『IBM MQ 應用程式的設計考量』。

訊息長度的效果
雖然 IBM MQ for IBM i 容許訊息保留最多 100 MB 的資料，但訊息中的資料量會影響處理訊息之應用程式的效能。 若要從應用程式取得最佳效能，請僅傳送訊息中的基本資料; 例如，在要求借記銀行帳戶的要求中，唯一可能需要從用戶端傳遞至伺服器應用程式的資訊是帳號及借記金額。

訊息持續性的效果
持續訊息會被登載。 日誌登載訊息會降低應用程式的效能，因此僅針對必要資料使用持續訊息。 如果在佇列管理程式停止或失敗時可以捨棄訊息中的資料，請使用非持續訊息。

搜尋特定訊息
MQGET 呼叫通常會從佇列中擷取第一個訊息。 如果您使用訊息描述子中的訊息及相關性 ID (*MsgId* 及 *CorrelId*) 來指定特定訊息，則佇列管理程式必須搜尋佇列，直到找到該訊息為止。 以此方式使用 MQGET 呼叫會影響應用程式的效能。

包含不同長度訊息的佇列
如果佇列上的訊息長度不同，為了判斷訊息大小，您的應用程式可以使用 MQGET 呼叫，並將 *BufferLength* 欄位設為零，以便即使呼叫失敗，也會傳回訊息資料的大小。 然後，應用程式可以重複呼叫，指定它在其第一次呼叫中所測量的訊息 ID，以及正確大小的緩衝區。 不過，如果有其他應用程式負責處理相同的佇列，您可能會發現應用程式的效能會降低，因為它的第二個 MQGET 呼叫會花費時間來搜尋另一個應用程式在兩次呼叫之間擷取的訊息。

如果您的應用程式無法使用固定長度的訊息，則此問題的另一個解決方案是使用 MQINQ 呼叫來尋找佇列可接受的訊息大小上限，然後在 MQGET 呼叫中使用此值。 佇列的訊息大小上限儲存在佇列的 **MaxMsgLen** 屬性中。 不過，此方法可能會使用大量儲存體，因為此佇列屬性的值可以是 IBM MQ for IBM i 容許的上限 (可能大於 2 GB)。

同步點的頻率
在同步點內發出許多 MQPUT 呼叫而未確定它們的程式可能會導致效能問題。 受影響的佇列可能填滿目前無法使用的訊息，而其他作業可能正在等待取得這些訊息。 此問題涉及儲存體，以及與嘗試取得訊息之作業相關的執行緒。

使用 MQPUT1 呼叫
僅當您有單一訊息放置在佇列上時，才使用 MQPUT1 呼叫。 如果您要放置多個訊息，請使用 MQOPEN 呼叫，後面接著一系列 MQPUT 呼叫和單一 MQCLOSE 呼叫。

使用中的執行緒數目
應用程式可能需要許多執行緒。 每個佇列管理程式處理程序都會配置容許的執行緒數目上限。 如果有些應用程式很麻煩，可能是因為它們的設計使用太多執行緒。 請考量應用程式是否考量此可能性，以及它是否採取動作來停止或報告此類型的出現項目。 IBM i 容許的執行緒數目上限為 4,095。 不過，預設值是 64。 IBM MQ 最多可提供 63 個執行緒給其處理程序。

特定的效能問題

本節說明儲存體及效能不佳的問題。

儲存體問題

如果您收到系統訊息 CPF0907. Serious storage condition may exist，則可能是您正在填滿與 IBM MQ for IBM i 佇列管理程式相關聯的空間。

您的應用程式或 IBM MQ for IBM i 執行緩慢嗎?

如果您的應用程式執行緩慢，則可能表示它處於迴圈中，或正在等待無法使用的資源。 這個執行緩慢也可能是效能問題所造成。 可能是因為您的系統運作接近其容量限制。 這種類型的問題可能在尖峰系統負載時間最差，通常是在上午中及下午中。 (如果您的網路延伸到多個時區，您可能會覺得在其他時間發生尖峰系統負載。)

如果您發現效能降低並不取決於系統負載，但有時會在系統負載較輕時發生，則設計不良的應用程式很可能是原因之一。 此問題可能將本身視為只有在存取特定佇列時才會發生的問題。

QTOTJOB 及 QADLTOTJ 是值得調查的系統值。

下列症狀可能指出 IBM MQ for IBM i 執行緩慢:

- 如果您的系統回應 MQSC 指令的速度緩慢。
- 如果重複顯示佇列深度，表示針對您預期會有大量佇列活動的應用程式，正在緩慢處理佇列。
- IBM MQ 追蹤是否在執行中?

Linux Linux on Power Systems - Little Endian 應用程式的設計考量

由於 Linux on Power Systems - Little Endian 僅支援 64 位元應用程式，因此 IBM MQ 中不支援 32 位元應用程式。

相關概念

第 41 頁的『IBM MQ 應用程式的設計考量』

當您決定應用程式如何利用可供您使用的平台及環境時，您需要決定如何使用 IBM MQ 所提供的特性。

z/OS z/OS 應用程式的設計和效能考量

應用程式設計是影響效能的最重要因素之一。 請利用這個主題來瞭解效能所涉及的一些設計因素。

有許多方式不良的程式設計會影響效能。 這些問題可能難以偵測，因為程式看起來執行良好，同時會影響其他作業的效能。 下列各節會示範數個特定於進行 MQI 呼叫之程式的問題。

如需應用程式設計的相關資訊，請參閱 第 41 頁的『IBM MQ 應用程式的設計考量』。

訊息長度的效果

雖然 IBM MQ for z/OS 容許訊息保留最多 100 MB 的資料，但訊息中的資料量會影響處理訊息之應用程式的效能。 若要從應用程式達到最佳效能，請只傳送訊息中的必要資料。 例如，在借記銀行帳戶的要求中，唯一可能需要從用戶端傳遞至伺服器應用程式的資訊是要借記的帳號及金額。

訊息持續性的效果

會記載持續訊息。 記載訊息會降低應用程式的效能，因此僅針對必要資料使用持續訊息。 如果在佇列管理程式停止或失敗時可以捨棄訊息中的資料，請使用非持續訊息。

持續訊息的資料會寫入日誌緩衝區。 在下列情況下，這些緩衝區會寫入日誌資料集:

- 確定發生
- 訊息已取得或已超出同步點
- WRTHRSH 緩衝區已填滿

在一個工作單元中處理許多訊息可能會導致較少輸入/輸出，如果針對每一個工作單元處理一個訊息，或超出同步點，則會產生較少的輸入/輸出。

搜尋特定訊息

MQGET 呼叫通常會從佇列中擷取第一個訊息。 如果您使用訊息和相關性 ID (**MsgId** 和 **CorrelId**) 在訊息描述子中指定特定訊息，佇列管理程式會搜尋佇列，直到找到該訊息為止。 以此方式使用 MQGET 會影響應用程式的效能，因為若要尋找特定訊息， IBM MQ 可能必須掃描整個佇列。

您可以使用 **IndexType** 佇列屬性，指定您要佇列管理程式維護可用來增加佇列上 MQGET 作業速度的索引。 不過，維護索引會有少量效能降低，因此只有在您需要使用它時才會產生一個。 您可以選擇建置訊息 ID 或相關性 ID 的索引，也可以選擇不建置循序擷取訊息之佇列的索引。 嘗試具有許多不同的索引鍵值，而不是具有相同值的批次。 例如 Balance1、 Balance2 及 Balance3，而不是 3 (含 Balance)。 對於共用佇列，您必須具有正確的 **IndexType**。 如需 **IndexType** 佇列屬性的詳細資料，請參閱 IndexType。

為了避免使用索引佇列來影響佇列管理程式重新啟動時間，請在 CSQ6SYSP 巨集中使用 QINDXBLD (NOWAIT) 參數。 這可讓佇列管理程式重新啟動完成，而不等待佇列索引建置完成。

如需 **IndexType** 屬性及其他物件屬性的完整說明，請參閱 物件屬性。

包含不同長度訊息的佇列

使用符合預期訊息大小的緩衝區大小來取得訊息。 如果您收到回覆碼，指出訊息太長，請取得更大的緩衝區。 以這種方式取得失敗時，傳回的資料長度是未轉換訊息資料的大小。 如果您在 MQGET 呼叫上指定 MQGMO_CONVERT ，且資料在轉換期間展開，則可能仍無法放入緩衝區中，在此情況下，您需要進一步增加緩衝區的大小。

如果您發出緩衝區長度為零的 MQGET ，它會傳回訊息大小，然後應用程式可以取得此大小的緩衝區，並重新發出 get。 如果您有多個應用程式在處理佇列，當原始應用程式重新發出 get 時，另一個應用程式可能已處理訊息。 如果您偶爾有大型訊息，則可能只需要針對這些訊息取得大型緩衝區，並在處理訊息之後釋放它。 如果所有應用程式都有大型緩衝區，這應該有助於減少虛擬儲存體問題。

如果您的應用程式無法使用固定長度的訊息，則此問題的另一個解決方案是使用 MQINQ 呼叫來尋找佇列可接受的訊息大小上限，然後在 MQGET 呼叫中使用此值。 佇列的訊息大小上限儲存在佇列的 **MaxMsgL** 屬性中。 不過，此方法可能會使用大量儲存體，因為 **MaxMsgL** 的值可能高達 100 MB (IBM MQ for z/OS 容許的上限)。

註: 將大型訊息放入佇列之後，您可以降低 **MaxMsgL** 參數。 例如，您可以放置 100 MB 訊息，然後將 **MaxMsgL** 設為 50 位元組。 這表示仍有可能取得比應用程式預期還大的訊息。

同步點的頻率

在同步點內發出許多 MQPUT 呼叫而未確定它們的程式可能會導致效能問題。 受影響的佇列可能填滿目前無法使用的訊息，而其他作業可能正在等待取得這些訊息。 這會影響儲存體，以及與嘗試取得訊息之作業相關聯的執行緒。

一般而言，如果您有多個應用程式正在處理佇列，當您具有下列任一項時，通常會取得最佳效能:

- 100 則簡訊 (小於 1 KB) ，或
- 一個訊息代表較大的訊息 (100 KB)

每一個同步點。 如果只有一個應用程式處理佇列，則每一個工作單元必須有更多訊息。

您可以使用 **MAXUMSGS** 佇列管理程式屬性來限制作業可以在單一回復單元內取得或放置的訊息數。 如需此屬性的相關資訊，請參閱 MQSC 指令中的 **ALTER QMGR** 指令。

MQPUT1 呼叫的優點

僅當您有單一訊息放置在佇列上時，才使用 MQPUT1 呼叫。 如果您要放置多個訊息，請使用 MQOPEN 呼叫，後面接著一系列 MQPUT 呼叫和單一 MQCLOSE 呼叫。

佇列管理程式可以包含多少訊息

本端佇列

佇列管理程式可以保留的本端訊息數基本上是頁集的大小。 您最多可以有 100 個頁集 (雖然建議頁集 0 和頁集 1 用於系統相關物件和佇列)。 您可以使用具有延伸格式的頁集，並增加頁集的容量。

共用佇列

共用佇列的容量取決於連結機能 (CF) 的大小。 IBM MQ 使用 CF 清單結構，其中基本儲存單元是項目和元素。 每一則訊息會儲存為 1 個項目，以及包含相關聯 MQMD 及其他訊息資料的多個元素。 單一訊息所耗用的元素數目取決於訊息的大小，以及 CFLEVEL (5) 在 MQPUT 時生效的卸載規則。 當訊息資料卸載至 Db2 或 SMDS 時，需要較少的元素。 當訊息已卸載時，訊息資料存取速度較慢。 如需與訊息卸載相關聯的效能及 CPU 額外負擔的進一步比較，請參閱 Performance Supportpac MP1H 。

影響效能的因素

效能可以表示處理訊息的速度，也可以表示每個訊息需要多少 CPU。

影響訊息處理速度的因素

對於持續訊息，最大影響是日誌資料集的速度。 日誌資料集的速度取決於它們所使用的 DASD。 因此，請小心將日誌資料集放置在低使用磁區上，以減少競用。 當每個 I/O 寫入多個頁面時，分段 MQ 日誌可增進日誌效能。 當 I/O 子系統忙碌時， z High Performance Fibre 連線 (zHPF) 對於 I/O 回應時間也有顯著的效能。

當有取得及放置訊息的要求時，在要求期間會鎖定佇列的存取權，以保留佇列的完整性。 基於規劃目的，請考量鎖定整個要求的佇列。 因此，如果放置時間是 100 微秒，且您每秒有超過 10,000 個要求，則可能會遇到延遲。 實際上，您可能取得比這更好的結果，但這是一個良好的一般規則。 您可以使用不同的佇列來增進效能。

可能的原因如下:

- 使用每一個 CICS 交易使用的一般回覆佇列
- 每一個 CICS 交易都有唯一的回覆目的地佇列
- 對 CICS 區域的佇列的回覆，以及 CICS 區域中的所有交易都使用此佇列。

答案取決於每秒的要求數，以及要求的回應時間。

如果必須從頁集讀取訊息，則與訊息位於緩衝池時相比會較慢。 如果您有超出緩衝池的訊息數目，則它們會溢出至磁碟。 因此，您需要確保緩衝池夠大，足以容納您的短暫存活訊息。 如果您有幾個小時之後處理的訊息，則這些訊息可能會溢出至磁碟，因此您應該預期這些訊息的取得速度會比緩衝池中的訊息來得慢。

對於共用佇列，訊息的速度取決於「連結機能」的速度。 實體處理器內的 CF 可能比外部 CF 更快。 CF 回應時間取決於 CF 忙碌的程度。 例如，在 Hursley 系統上，當 CF 為 17% 忙碌時，回應時間為 14 微秒。 當 CF 為 95% 忙碌時，回應時間為 45 微秒。

如果 MQ 要求使用大量 CPU ，這會影響處理訊息的速度。 因為如果邏輯分割區 (LPAR) 受限於 CPU ，應用程式將延遲等待 CPU。

每個訊息的 CPU 數量

一般而言，較大的訊息會使用較多的 CPU ，因此請盡可能避免大量 (x MB) 訊息。

從佇列中取得特定訊息時，應該為佇列編製索引，以便佇列管理程式可以直接移至訊息 (因此可避免佇列的整個掃描)。 如果佇列未編製索引，則會從頭開始掃描佇列以尋找訊息。 如果佇列上有 1000 則訊息，則可能必須掃描所有 1000 則訊息。 結果是大量不必要的 CPU 使用率。

由於訊息加密，使用 TLS 的通道會有額外成本。

在 MQ V7 中，除了 **CORRELID** 或 **MSGID** 之外，您還可以透過選取元字串來選取訊息。 每個訊息都必須在中查看，因此如果佇列中有許多訊息，這會很昂貴。

應用程式執行 OPEN PUT PUT CLOSE 比 PUT1 PUT1 更有效率。

在 CICS 中觸發

當觸發佇列的訊息到達速率低時，首先使用觸發程式是有效率的。 當訊息到達速率每秒超過 10 則訊息時，觸發第一個交易，然後讓交易處理訊息並取得下一個訊息等會更有效率。 如果訊息在短時間內 (例如在 0.1 和 1 秒之間) 未到達，則交易結束。 在高傳輸量時，您可能需要執行多個交易來處理訊息，以及防止建立訊息。 對於每個產生的觸發訊息，這需要放置及取得觸發訊息，這實際上會使訊息成本加倍。

支援多少連線或並行使用者

每一個連線都會使用佇列管理程式內的虛擬儲存體，因此並行使用者越多，所使用的儲存體就越多。 如果您需要非常大的緩衝池和大量使用者，則您可能會受到虛擬儲存體的限制，而且您可能需要減少緩衝池的大小。

如果使用安全，佇列管理程式會長時間快取佇列管理程式內的資訊。 佇列管理程式內使用的虛擬儲存體數量會受到影響。

CHINIT 最多可以支援大約 10,000 個連線。 這受虛擬儲存體限制。 如果連線使用更多儲存體 (例如由 TLS 使用)，則每個連線的儲存體會增加，因此表示 **CHINIT** 可以支援較少的連線。 如果您正在處理大型訊息，則在 **CHINIT** 中需要更多的緩衝區儲存體，因此 **CHINIT** 可以支援較少的訊息。

與遠端佇列管理程式的連線比用戶端連線更有效率。 例如，每個 MQ 用戶端要求都需要兩個網路流程 (一個用於要求，一個用於回應)。 使用遠端佇列管理程式的通道，在回應回應之前可能有 50 個透過網路傳送。 如果您考慮使用大型用戶端網路，則在分散式方框上使用集中器佇列管理程式可能會更有效率，而且會有一個通道進出集中器。

影響效能的其他事項

日誌資料集大小至少應該為 1000 個磁柱。 如果日誌小於此值，則檢查點活動可能太頻繁。 在忙碌系統上，檢查點通常應該是每 15 分鐘或更長，其傳輸量非常高，可能小於此值。 發生檢查點時，會掃描緩衝池，並將「舊」訊息及變更的頁面寫入磁碟。 如果檢查點太頻繁，這可能會影響效能。 LOGLOAD 的值也會影響檢查點頻率。 如果佇列管理程式異常結束，則在重新啟動時可能必須讀回 3 個檢查點。 最佳檢查點間隔是取得檢查點時的活動與佇列管理程式重新啟動時可能需要讀取的日誌資料量之間的平衡。

啟動通道時產生大量額外負擔。 通常最好啟動通道並保持連線，而不是頻繁啟動及停止通道。

相關資訊

MP1K: IBM MQ for z/OS 9.0 效能報告

z/OS IBM MQ for z/OS 上的 IMS 及 IMS 橋接器應用程式

此資訊可協助您使用 IBM MQ 撰寫 IMS 應用程式。

- 如果要在 IMS 應用程式中使用同步點和 MQI 呼叫，請參閱 第 58 頁的『使用 IBM MQ 撰寫 IMS 應用程式』。
- 若要撰寫使用 IBM MQ - IMS 橋接器的應用程式，請參閱 第 61 頁的『撰寫 IMS 橋接器應用程式』。

使用下列鏈結，以進一步瞭解 IBM MQ for z/OS 上 IMS 及 IMS 橋接器應用程式的相關資訊:

- 第 58 頁的『使用 IBM MQ 撰寫 IMS 應用程式』
- 第 61 頁的『撰寫 IMS 橋接器應用程式』

相關概念

第 604 頁的『訊息佇列介面概觀』
瞭解「訊息佇列介面 (MQI)」元件。

第 616 頁的『連接至佇列管理程式並切斷與佇列管理程式的連線』
若要使用 IBM MQ 程式設計服務，程式必須具有與佇列管理程式的連線。 使用此資訊來瞭解如何連接至佇列管理程式，以及與佇列管理程式中斷連線。

第 621 頁的『開啟及關閉物件』

此資訊提供開啟及關閉 IBM MQ 物件的見解。

第 630 頁的『將訊息放置在佇列上』
使用此資訊來瞭解如何將訊息放置在佇列上。

第 643 頁的『從佇列取得訊息』
使用此資訊來瞭解從佇列取得訊息的相關資訊。

第 713 頁的『查詢及設定物件屬性』
屬性是定義 IBM MQ 物件性質的內容。

第 716 頁的『確定及取消工作單元』
此資訊說明如何確定及取消工作單元中發生的任何可回復取得及放置作業。

第 725 頁的『使用觸發程式啟動 IBM MQ 應用程式』
瞭解觸發程式以及如何使用觸發程式來啟動 IBM MQ 應用程式。

第 741 頁的『使用 MQI 及叢集』
對於與叢集作業相關的呼叫和回覆碼，有一些特殊選項。

第 745 頁的『在 IBM MQ for z/OS 上使用及撰寫應用程式』
IBM MQ for z/OS 應用程式可以由在許多不同環境中執行的程式組成。 這表示他們可以利用多個環境中可用的設施。

z/OS 使用 IBM MQ 撰寫 IMS 應用程式

在 IMS 應用程式中使用 IBM MQ 時還有進一步的考量。這些包括可以使用哪些 MQ API 呼叫，以及用於同步點的機制。

使用下列鏈結，以進一步瞭解如何在 IBM MQ for z/OS 上撰寫 IMS 應用程式:

- 第 58 頁的『IMS 應用程式中的同步點』
- 第 59 頁的『IMS 應用程式中的 MQI 呼叫』

限制

使用 IMS 配接器的應用程式可以使用哪些 IBM MQ API 呼叫有一些限制。

在使用 IMS 配接器的應用程式內不支援下列 IBM MQ API 呼叫:

- MQCB
- MQCB_XX_ENCODE_CASE_ONE function
- MQCTL

相關概念

第 61 頁的『撰寫 IMS 橋接器應用程式』
本主題包含撰寫應用程式以使用 IBM MQ - IMS 橋接器的相關資訊。

z/OS *IMS* 應用程式中的同步點

在 IMS 應用程式中，您可以使用對 IOPCB 及 CHKP (檢查點) 的 GU (取得唯一) 之類的 IMS 呼叫來建立同步點。

若要回復自前一個檢查點以來的所有變更，您可以使用 IMS ROLB (回復) 呼叫。 如需相關資訊，請參閱 IMS 說明文件中的 ROLB 呼叫 。

佇列管理程式是兩段式確定通訊協定中的參與者; IMS 同步點管理程式是協調程式。

IMS 配接器會在同步點關閉所有開啟控點 (批次或非訊息驅動 BMP 環境除外)。 這是因為不同的使用者可以起始下一個工作單元，而 IBM MQ 安全檢查是在發出 MQCONN、MQCONNX 及 MQOPEN 呼叫時執行，而不是在發出 MQPUT 或 MQGET 呼叫時執行。

不過，在「等待輸入 (WFI)」或虛擬「等待輸入 (PWFI)」環境中， IMS 不會通知 IBM MQ 關閉控點，直到下一個訊息到達或 QC 狀態碼傳回應用程式為止。 如果應用程式正在 IMS 區域中等待，且任何這些控點都屬於已觸發的佇列，則不會發生觸發，因為佇列已開啟。 因此，在 WFI 或 PWFI 環境中執行的應用程式應該在對 IOPCB 執行 GU 以取得下一則訊息之前，明確 MQCLOSE 佇列控點。

如果 IMS 應用程式 (BMP 或 MPP) 發出 MQDISC 呼叫，則會關閉開啟佇列，但不會採用隱含的同步點。 如果應用程式正常結束，則會關閉任何開啟的佇列，並進行隱含的確定。 如果應用程式異常結束，則會關閉任何開啟的佇列，並發生隱含的取消。

z/OS *IMS 應用程式中的 MQI 呼叫*

使用此資訊來瞭解在「伺服器」應用程式及「查詢」應用程式上使用 MQI 呼叫的相關資訊。

本節涵蓋在下列類型的 IMS 應用程式中使用 MQI 呼叫:

- 第 59 頁的『伺服器應用程式』
- 第 61 頁的『查詢應用程式』

伺服器應用程式

以下是 MQI 伺服器應用程式模型的大綱:

```
Initialize/Connect
.
Open queue for input shared
.
Get message from IBM MQ queue
.
Do while Get does not fail
.
If expected message received
Process the message
Else
Process unexpected message
End if
.
Commit
.
Get next message from IBM MQ queue
.
End do
.
Close queue/Disconnect
.
END
```

範例程式 CSQ4ICB3 顯示在 C/370 中使用此模型的 BMP 實作。 程式會先建立與 IMS 的通訊，然後再建立與 IBM MQ 的通訊:

```
main()
----
Call InitIMS
If IMS initialization successful
Call InitMQM
If IBM MQ initialization successful
Call ProcessRequests
Call EndMQM
End-if
End-if

Return
```

IMS 起始設定會判定是否以訊息驅動或批次導向 BMP 來呼叫程式，並相應地控制 IBM MQ 佇列管理程式連線及佇列控點:

```
InitIMS
-------
Get the IO, Alternate and Database PCBs
Set MessageOriented to true

Call ctdli to handle status codes rather than abend
If call is successful (status code is zero)
While status code is zero
Call ctdli to get next message from IMS message queue
If message received
Do nothing
```

```
Else if no IOPBC
Set MessageOriented to false
Initialize error message
Build 'Started as batch oriented BMP' message
Call ReportCallError to output the message
End-if
Else if response is not 'no message available'
Initialize error message
Build 'GU failed' message
Call ReportCallError to output the message
Set return code to error
End-if
End-if
End-while
Else
Initialize error message
Build 'INIT failed' message
Call ReportCallError to output the message
Set return code to error
End-if

Return to calling function
```

IBM MQ 起始設定會連接至佇列管理程式，並開啟佇列。 在訊息驅動 BMP 中，這會在取得每一個 IMS 同步點之後呼叫; 在批次導向 BMP 中，這只會在程式啟動期間呼叫:

```
InitMQM
-------
Connect to the queue manager
If connect is successful
Initialize variables for the open call
Open the request queue
If open is not successful
Initialize error message
Build 'open failed' message
Call ReportCallError to output the message
Set return code to error
End-if
Else
Initialize error message
Build 'connect failed' message
Call ReportCallError to output the message
Set return code to error
End-if

Return to calling function
```

MPP 中伺服器模型的實作受到 MPP 每次呼叫處理單一工作單元的影響。 這是因為當採用同步點 (GU) 時，會關閉連線和佇列控點，並遞送下一個 IMS 訊息。 下列其中一項可部分克服此限制:

- **在單一工作單元內處理許多訊息**

  這涉及:

  – 讀取訊息
  – 正在處理必要的更新項目
  – 放置回覆

  迴圈中，直到已處理所有訊息，或直到已處理設定的訊息數目上限為止，此時會採用同步點。

  以這種方式只能處理特定類型的應用程式 (例如，簡式資料庫更新或查詢)。 雖然 MQI 回覆訊息可以放置在所處理 MQI 訊息的發送端權限下，但需要小心處理任何 IMS 資源更新項目的安全含意。

- **每次呼叫 MPP 時處理一則訊息，並確保多重排程 MPP 以處理所有可用的訊息**。

  當 IBM MQ 佇列上有訊息且沒有負責處理 MPP 交易的應用程式時，請使用 IBM MQ IMS 觸發監視器程式 (CSQQTRMN) 來排定 MPP 交易。

  如果觸發監視器啟動 MPP ，則佇列管理程式名稱及佇列名稱會傳遞至程式，如下列 COBOL 程式碼擷取所示:

```
* Data definition extract
01 WS-INPUT-MSG.
```

```
05 IN-LL1         PIC S9(3) COMP.
05 IN-ZZ1         PIC S9(3) COMP.
05 WS-STRINGPARM      PIC X(1000).
01 TRIGGER-MESSAGE.
COPY CMQTMC2L.
*
* Code extract
GU-IOPCB SECTION.
MOVE SPACES TO WS-STRINGPARM.
CALL 'CBLTDLI' USING GU,
IOPCB,
WS-INPUT-MSG.
IF IOPCB-STATUS = SPACES
MOVE WS-STRINGPARM TO MQTMC.
*  ELSE handle error
*
* Now use the queue manager and queue names passed
DISPLAY 'MQTMC-QMGRNAME     ='
MQTMC-QMGRNAME OF MQTMC '='.
DISPLAY 'MQTMC-QNAME       ='
MQTMC-QNAME  OF MQTMC '='.
```

雖然無法使用 CSQQTRMN 觸發 BMP，但在批次處理區域中最好支援伺服器模型 (預期是長時間執行的作業)。

查詢應用程式

一般 IBM MQ 應用程式起始查詢或更新的運作方式如下:

- 從使用者收集資料
- 放置一或多則 IBM MQ 訊息
- 取得回覆訊息 (您可能必須等待它們)
- 提供回應給使用者

因為放置在 IBM MQ 佇列中的訊息在確定之前不會變成可供其他 IBM MQ 應用程式使用，所以必須將它們置於同步點之外，或 IMS 應用程式必須分割成兩個交易。

如果查詢涉及放置單一訊息，您可以使用 無同步點 選項; 不過，如果查詢更複雜，或涉及資源更新，如果發生失敗且未使用同步點，則可能會發生一致性問題。

若要克服此問題，您可以使用 MQI 呼叫 (使用程式對程式訊息切換參數) 來分割 IMS MPP 交易; 如需相關資訊，請參閱 *IMS 跨系統通訊 (ISC)* 。 這容許在 MPP 中實作查詢程式:

```
Initialize first program/Connect
.
Open queue for output
.
Put inquiry to IBM MQ queue
.
Switch to second IBM MQ program, passing necessary data in save
pack area (this commits the put)
.
END
.
.
Initialize second program/Connect
.
Open queue for input shared
.
Get results of inquiry from IBM MQ queue
.
Return results to originator
.
END
```

z/OS 撰寫 IMS 橋接器應用程式

本主題包含撰寫應用程式以使用 IBM MQ - IMS 橋接器的相關資訊。

如需 IBM MQ - IMS 橋接器的相關資訊，請參閱 IMS 橋接器。

請使用下列鏈結，以進一步瞭解如何在 IBM MQ for z/OS 上撰寫 IMS 橋接器應用程式:

- 第 62 頁的『IMS 橋接器如何處理訊息』
- 第 763 頁的『透過 IBM MQ 撰寫 IMS 交易程式』

相關概念

第 58 頁的『使用 IBM MQ 撰寫 IMS 應用程式』
在 IMS 應用程式中使用 IBM MQ 時還有進一步的考量。這些包括可以使用哪些 MQ API 呼叫，以及用於同步點的機制。

z/OS *IMS* 橋接器如何處理訊息

當您使用 IBM MQ - IMS 橋接器將訊息傳送至 IMS 應用程式時，需要以特殊格式建構訊息。

您還必須將訊息放置在已定義儲存類別 (指定目標 IMS 系統的 XCF 群組及成員名稱) 的 IBM MQ 佇列上。 這些稱為 MQ-IMS 橋接器佇列，或只是 **橋接器** 佇列。

如果橋接器佇列以 QSGDISP (QMGR) 定義，或以 QSGDISP (SHARED) 搭配 NOSHARE 選項一起定義，則 IBM MQ-IMS 橋接器需要對橋接器佇列的互斥輸入存取 (MQOO_INPUT_EXCLUSIVE)。

在將訊息傳送至 IMS 應用程式之前，使用者不需要登入 IMS 。 MQMD 結構的 *UserIdentifier* 欄位中的使用者 ID 用於安全檢查。 檢查層次是在 IBM MQ 連接至 IMS 時決定，並在 IMS 橋接器的應用程式存取控制中說明。 這可讓您實作虛擬登入。

IBM MQ - IMS 橋接器接受下列類型的訊息:

- 包含 IMS 交易資料及 MQIIH 結構 (在 MQIIH 中說明) 的訊息:

```
MQIIH LLZZ<trancode><data>[LLZZ<data>][LLZZ<data>]
```

註:

1. 方括弧 [] 代表選用的多區段。
2. 將 MQMD 結構的 *Format* 欄位設為 MQFMT_IMS ， 以使用 MQIIH 結構。

- 包含 IMS 交易資料但無 MQIIH 結構的訊息:

```
LLZZ<trancode><data> \
[LLZZ<data>][LLZZ<data>]
```

IBM MQ 會驗證訊息資料，以確保 LL 位元組加上 MQIIH (如果存在的話) 長度的總和等於訊息長度。

當 IBM MQ - IMS 橋接器從橋接器佇列取得訊息時，它會如下處理訊息:

- 如果訊息包含 MQIIH 結構，則橋接器會驗證 MQIIH (請參閱 MQIIH)、建置 OTMA 標頭，並將訊息傳送至 IMS。 交易碼指定在輸入訊息中。 如果這是邏輯終端機， IMS 會以 DFS1288E 訊息回覆。 如果交易碼代表指令，則 IMS 會執行該指令; 否則訊息會排入 IMS 佇列以進行交易。
- 如果訊息包含 IMS 交易資料，但沒有 MQIIH 結構，則 IMS 橋接器會做出下列假設:
  - 交易碼以 5 到 12 個位元組的使用者資料為單位
  - 交易處於非交談模式
  - 交易處於確定模式 0 (commit-then-send)
  - MQMD 中的 *Format* 用作 *MFSMapName* (輸入時)
  - 安全模式是 MQISS_CHECK

  也會建置不含 MQIIH 結構的回覆訊息，從 IMS 輸出的 *MFSMapName* 中取得 MQMD 的 *Format* 。

IBM MQ - IMS 橋接器會針對每一個 IBM MQ 佇列使用一或兩個 Tp 管道:

- 使用「確定」模式 0 (COMMIT_THEN_SEND) (這些在 IMS /DIS TMEMBER 用戶端 TPIPE xxxx 指令的狀態欄位中顯示 SYN) 的所有訊息會使用同步化 Tpipe
- 使用「確定」模式 1 (SEND_THEN_COMMIT) 的所有訊息都使用非同步化 Tpipe

第一次使用 Tp 管道時， IBM MQ 會建立 Tp 管道。 在 IMS 重新啟動之前，會存在未同步的 Tpipe。 已同步的 Tp 管道存在，直到 IMS 冷啟動為止。 您無法自行刪除這些 Tp 管道。

如需 IBM MQ - IMS 橋接器如何處理訊息的相關資訊，請參閱下列主題:

- 第 63 頁的『將 IBM MQ 訊息對映至 IMS 交易類型』
- 第 63 頁的『如果訊息無法放入 IMS 佇列』
- 第 64 頁的『IMS 橋接器回饋碼』
- 第 64 頁的『來自 IMS 橋接器的訊息中的 MQMD 欄位』
- 第 65 頁的『來自 IMS 橋接器的訊息中的 MQIIH 欄位』
- 第 66 頁的『來自 IMS 的回覆訊息』
- 第 66 頁的『在 IMS 交易中使用替代回應 PCB』
- 第 66 頁的『從 IMS 傳送自發的訊息』
- 第 66 頁的『訊息分段』
- 第 67 頁的『與 IMS 橋接器之間的訊息資料轉換』

相關概念

第 763 頁的『透過 IBM MQ 撰寫 IMS 交易程式』

透過 IBM MQ 處理 IMS 交易所需的編碼取決於 IMS 交易所需的訊息格式，以及它可以傳回的回應範圍。 不過，當您的應用程式處理 IMS 畫面格式化資訊時，有幾點需要考量。

z/OS 將 *IBM MQ* 訊息對映至 *IMS* 交易類型

此表格說明 IBM MQ 訊息至 IMS 交易類型的對映。

表 *4: IBM MQ* 訊息如何對映至 *IMS* 交易類型

IBM MQ 訊息類型	**Commit-then-send (模式 0)-使用已同步的 IMS Tp 管道**	**Send-then-commit (模式 1)-使用非同步化 IMS Tp 管道**
持續 IBM MQ 訊息	• 可回復的完整功能交易 • IMS 拒絕無法復原的交易	• 捷徑交易 • 交談式交易 • 完整功能交易
非持續 IBM MQ 訊息	• 無法復原的完整功能交易 • IMS V8 及 APAR PQ61404 以及 IMS 的所有更新版本允許可回復交易	• 捷徑交易 • 交談式交易 • 完整功能交易

註: IMS 指令無法使用具有確定模式 0 的持續 IBM MQ 訊息。 如需相關資訊，請參閱 確定模式 (commitMode) 。

z/OS 如果訊息無法放入 *IMS* 佇列

瞭解無法將訊息放入 IMS 佇列時要採取的動作。

如果訊息無法放入 IMS 佇列， IBM MQ 會採取下列動作:

- 如果因為訊息無效而無法將訊息放置到 IMS ，則會將訊息放置到無法傳送郵件的佇列，並將訊息傳送到系統主控台。
- 如果訊息有效，但遭到 IMS 拒絕，則 IBM MQ 會將錯誤訊息傳送至系統主控台，該訊息包括 IMS 感應碼，且 IBM MQ 訊息會放置在無法傳送郵件的佇列中。 如果 IMS 感應碼為 001A， IMS 會將包含失敗原因的 IBM MQ 訊息傳送至回覆目的地佇列。

  註: 在先前列出的情況下，如果 IBM MQ 因任何原因而無法將訊息放入無法傳送郵件的佇列，則會將訊息傳回原始 IBM MQ 佇列。 錯誤訊息會傳送至系統主控台，且不會從該佇列傳送進一步訊息。

  若要重新傳送訊息，請執行下列 **其中一個** :

- 停止並重新啟動 IMS 中對應於佇列的 Tp 管道
- 將佇列變更為 GET (DISABLED)，並再次變更為 GET (ENABLED)
- 停止並重新啟動 IMS 或 OTMA
- 停止並重新啟動 IBM MQ 子系統

- 如果 IMS 拒絕訊息以外的任何其他訊息錯誤，則 IBM MQ 訊息會傳回原始佇列， IBM MQ 會停止處理佇列，並將錯誤訊息傳送至系統主控台。

  如果需要異常狀況報告訊息，橋接器會以發送端的權限將它放入回覆目的地佇列。 如果無法將訊息放入佇列，則會以橋接器的權限將報告訊息放入無法傳送郵件的佇列。 如果無法將它放入 DLQ，則會捨棄它。

z/OS *IMS* 橋接器回饋碼

IMS 感應碼通常在 IBM MQ 主控台訊息中以十六進位格式輸出，例如 CSQ2001I (例如，感應碼 0x001F)。 在放入無法傳送郵件之訊息的無法傳送郵件的標頭中看到的 IBM MQ 回饋碼是十進位數。

IMS 橋接器回饋碼的範圍為 301 至 399，或 600 至 855 (若為 NACK 感應碼 0x001A)。 它們是從 IMS-OTMA 感應碼對映，如下所示:

1. IMS-OTMA 感應碼會從十六進位數轉換成十進位數。
2. 300 會新增至 1 中計算所產生的數字，並提供 IBM MQ *Feedback* 代碼。
3. IMS-OTMA 感應碼 0x001A，十進位 26 是特殊情況。 即會產生範圍 600-855 內的 *Feedback* 程式碼。
   a. IMS-OTMA 原因碼會從十六進位數轉換成十進位數。
   b. 600 會新增至 a 中計算所產生的數字，並提供 IBM MQ *Feedback* 代碼。

如需 IMS-OTMA 感應碼的相關資訊，請參閱 NAK 訊息的 OTMA 感應碼。

z/OS 來自 *IMS* 橋接器的訊息中的 *MQMD* 欄位

瞭解來自 IMS 橋接器之訊息中的 MQMD 欄位。

IMS 會在 OTMA 標頭的「使用者資料」區段中攜帶原始訊息的 MQMD。 如果訊息源自 IMS，則由 IMS 目的地解決方案結束程式建置。 從 IMS 接收的訊息 MQMD 建置如下:

StrucID
"MD"

版本
MQMD_VERSION_1

報告
MQRO_NONE

MsgType
MQMT_REPLY

期限
如果 MQIIH 的旗標欄位中設定了 MQIH_PASS_EXPIRATION，則此欄位會包含剩餘到期時間，否則它會設為 MQEI_UNQUALITED

意見
MQFB_NONE

編碼
MQENC.Native (z/OS 系統的編碼)

CodedCharSetId
MQCCSI_Q_MGR (z/OS 系統的 CodedCharSetID)

格式
MQFMT_IMS (如果 MQMD.Format 為 MQFMT_IMS，否則為 IOPCB.MODNAME

優先順序
MQMD.Priority

持續性
視確定模式而定: 如果 CM-1; 持續性符合 IMS 訊息的可回復性 (如果 CM-0)， 則輸入訊息的 MQMD.Persistence

MsgId
MQMD.MsgId 如果 MQRO_PASS_MSG_ID， 則為 MQRO_PASS_MSG_ID， 否則為新的 MsgId (預設值)

CorrelId
MQMD.CorrelId， 否則為 MQMD.MsgId (預設值)

BackoutCount
0

ReplyToQ
空白

回覆目的地佇列管理程式
空白 (在 MQPUT 期間由佇列管理程式設為本端佇列管理程式名稱)

UserIdentifier
MQMD.UserIdentifier

AccountingToken
MQMD.AccountingToken

ApplIdentityData
MQMD.ApplIdentityData

PutApplType
如果沒有錯誤，則為 MQAT_XCF， 否則為 MQAT_BRIDGE

PutApplName
<XCFgroupName><XCFmemberName> if no error, otherwise QMGR name

PutDate
放置訊息的日期

PutTime
放置訊息的時間

ApplOriginData
空白

z/OS 來自 *IMS* 橋接器的訊息中的 *MQIIH* 欄位
瞭解來自 IMS 橋接器的訊息中的 MQIIH 欄位。

從 IMS 接收的訊息 MQIIH 建置如下:

StrucId
"IIH"

版本
1

StrucLength
84

編碼
MQENC_NATIVE

CodedCharSetId
MQCCSI_Q_MGR

格式
MQIIH.ReplyToFormat， 如果 MQIIH.ReplyToFormat 不是空白，則為輸入訊息，否則為 IOPCB.MODNAME

旗標
0

LTermOverride
OTMA 標頭中的 LTERM 名稱 (Tpipe)

MFSMapName
OTMA 標頭中的對映名稱

ReplyTo 格式
空白

鑑別程式
輸入訊息的 MQIIH.Authenticator，如果將回覆訊息放入 MQ-IMS 橋接器佇列，則為空白。

TranInstanceID
來自 OTMA 標頭的交談 ID/伺服器記號 (如果在交談中)。 在 V14 之前的 IMS 版本中，如果不在交談中，則此欄位一律為空值。 從 IMS V14 開始，即使不在交談中， IMS 也可能會設定此欄位。

TranState
如果在交談中，則為 "C"，否則為空白

CommitMode
OTMA 標頭的確定模式 ("0" 或 "1")

SecurityScope
Blank

已保留
Blank

z/OS 來自 *IMS* 的回覆訊息

當 IMS 交易 ISRT 至其 IOPCB 時，訊息會遞送回原始 LTERM 或 TPIPE。

這些在 IBM MQ 中視為回覆訊息。 來自 IMS 的回覆訊息會放入原始訊息中指定的回覆目的地佇列。 如果無法將訊息放入回覆目的地佇列，則會使用橋接器的權限將訊息放入無法傳送郵件的佇列。 如果無法將訊息放入無法傳送郵件的佇列，則會將負面確認通知傳送至 IMS，以指出無法接收訊息。 然後，訊息的責任會回到 IMS。 如果您使用確定模式 0，則來自該 Tpipe 的訊息不會傳送至橋接器，且會保留在 IMS 佇列上; 亦即，在重新啟動之前不會傳送進一步訊息。 如果您使用確定模式 1，則其他工作可以繼續。

如果回覆具有 MQIIH 結構，則其格式類型為 MQFMT_IMS; 否則，其格式類型由插入訊息時使用的 IMS MOD 名稱指定。

z/OS 在 *IMS* 交易中使用替代回應 *PCB*

當 IMS 交易使用替代回應 PCB (ALTPCB 的 ISRT，或對可修改的 PCB 發出 CHNG 呼叫) 時，會呼叫預先遞送結束程式 (DFSYPRX0)，以判定是否應該重新遞送訊息。

如果要重新遞送訊息，則會呼叫目的地解析結束程式 (DFSYDRU0) 來確認目的地並準備標頭資訊，如需這些結束程式的相關資訊，請參閱 在 IMS 中使用 OTMA 結束程式 及 預先遞送結束程式 DFSYPRX0 。

除非在結束程式中採取動作，否則從 IBM MQ 佇列管理程式起始之 IMS 交易的所有輸出 (不論是 IOPCB 或 ALTPCB) 都會回到相同的佇列管理程式。

z/OS 從 *IMS* 傳送自發的訊息

若要將訊息從 IMS 傳送至 IBM MQ 佇列，您需要呼叫 ISRT 至 ALTPCB 的 IMS 交易。

您需要撰寫預先遞送及目的地解析結束程式，以從 IMS 遞送自發的訊息並建置 OTMA 使用者資料，以便能夠正確建置訊息的 MQMD。 如需這些結束程式的相關資訊，請參閱 預先遞送結束程式 DFSYPRX0 及 目的地解析使用者結束程式 。

註: IBM MQ - IMS 橋接器不知道它收到的訊息是回覆還是自發訊息。 在每一種情況下，它都會以相同方式處理訊息，並根據隨訊息送達的 OTMA UserData 來建置回覆的 MQMD 及 MQIIH

自發的訊息可以建立新的 Tp 管道。 例如，如果現有 IMS 交易切換至新的邏輯終端機 (例如 PRINT01)，但實作需要透過 OTMA 遞送輸出，則會建立新的 Tpipe (在此範例中稱為 PRINT01)。 依預設，這是未同步的 Tpipe。 如果實作需要訊息可回復，請設定目的地解析結束程式輸出旗標。 如需相關資訊，請參閱 *IMS* 自訂作業手冊 。

z/OS 訊息分段

您可以將 IMS 交易定義為預期單一或多區段輸入。

原始 IBM MQ 應用程式必須將 MQIIH 結構後面的使用者輸入建構為一或多個 LLZZ 資料區段。 IMS 訊息的所有區段都必須包含在以單一 MQPUT 傳送的單一 IBM MQ 訊息中。

LLZZ 資料區段的長度上限由 IMS/OTMA (32767 位元組) 定義。 IBM MQ 訊息長度總計是 LL 位元組加上 MQIIH 結構長度的總和。

回覆的所有區段都包含在單一 IBM MQ 訊息中。

對於 MQFMT_IMS_VAR_STRING 格式的訊息，有進一步的 32 KB 限制。 當 ASCII 混合 CCSID 訊息中的資料轉換成 EBCDIC 混合 CCSID 訊息時，每次 SBCS 與 DBCS 字元之間有轉移時，都會新增移入位元組或移出位元組。 32 KB 限制適用於訊息的大小上限。 亦即，因為訊息中的 LL 欄位不能超過 32 KB，所以訊息不得超過 32 KB，包括所有移入及移出字元。 建置訊息的應用程式必須容許此情況。

z/OS 與 *IMS* 橋接器之間的訊息資料轉換

資料轉換是由分散式佇列機能 (可能呼叫任何必要的結束程式) 或內部群組佇列代理程式 (不支援使用結束程式) 在將訊息放入目的地佇列時執行，該目的地佇列已針對其儲存類別定義 XCF 資訊。 當發佈/訂閱將訊息遞送至佇列時，不會進行資料轉換。

在 CSQXLIB DD 陳述式所參照的資料集中，任何需要的結束程式都必須可供分散式佇列機能使用。 這表示您可以從任何 IBM MQ 平台使用 IBM MQ - IMS 橋接器，將訊息傳送至 IMS 應用程式。

如果有轉換錯誤，則會將訊息放入未轉換的佇列; 這最終會導致 IBM MQ - IMS 橋接器將其視為錯誤，因為橋接器無法辨識標頭格式。 如果發生轉換錯誤，則會將錯誤訊息傳送至 z/OS 主控台。

如需一般資料轉換的詳細資訊，請參閱 第 823 頁的『寫入資料-轉換結束程式』。

將訊息傳送至 IBM MQ - IMS 橋接器

若要確保正確執行轉換，您必須告知佇列管理程式訊息的格式。

如果訊息具有 MQIIH 結構，則 MQMD 中的 *Format* 必須設為內建格式 MQFMT_IMS，且 MQIIH 中的 *Format* 必須設為說明訊息資料的格式名稱。 如果沒有 MQIIH，請將 MQMD 中的 *Format* 設為您的格式名稱。

如果您的資料 (LLZZ 除外) 是所有字元資料 (MQCHAR)，請使用內建格式 MQFMT_IMS_VAR_STRING 作為您的格式名稱 (在 MQIIH 或 MQMD 中，視情況而定)。 否則，請使用您自己的格式名稱，在此情況下，您也必須提供格式的資料轉換結束程式。 除了資料本身之外，結束程式還必須處理訊息中 LLZZ 的轉換 (但它不需要在訊息開始時處理任何 MQIIH)。

如果您的應用程式使用 *MFSMapName*，則可以改為搭配使用訊息與 MQFMT_IMS，並在 MQIIH 的 MFSMapName 欄位中定義傳遞至 IMS 交易的對映名稱。

從 IBM MQ - IMS 橋接器接收訊息

如果您要傳送至 IMS 的原始訊息上存在 MQIIH 結構，則回覆訊息上也會存在 MQIIH 結構。

若要確保正確轉換您的回覆，請執行下列動作:

- 如果您在原始訊息上具有 MQIIH 結構，請在原始訊息的 MQIIH *ReplytoFormat* 欄位中指定您想要的回覆訊息格式。 此值位於回覆訊息的 MQIIH *Format* 欄位中。 This is particularly useful if all your output data is of the form LLZZ<character data>.
- 如果原始訊息沒有 MQIIH 結構，請在 IMS 應用程式的 ISRT 中將回覆訊息的格式指定為 IOPCB 的 MFS MOD 名稱。

開發 JMS/Jakarta Messaging 和 Java 應用程式

IBM MQ 提供三個 Java 語言介面: IBM MQ classes for Jakarta Messaging、 IBM MQ classes for JMS 及 IBM MQ classes for Java。

關於這項作業

JM 3.0 V 9.3.0 V 9.3.0 **IBM MQ classes for Jakarta Messaging**

IBM MQ classes for Jakarta Messaging 是一個 Jakarta Messaging 提供者，將 IBM MQ 的 Jakarta Messaging 介面實作為傳訊系統。 Jakarta Connectors Architecture 提供將在 Jakarta EE 環境中執行的應用程式連接至「企業資訊系統 (EIS)」(例如 IBM MQ 或 Db2) 的標準方式。

如需相關資訊，請參閱第 69 頁的『為何應該使用 IBM MQ classes for Jakarta Messaging?』和第 72 頁的『從 Java 存取 IBM MQ -API 的選項』。

JMS 2.0 **IBM MQ classes for JMS**

IBM MQ classes for JMS 是一個 JMS 提供者，將 IBM MQ 的 JMS 介面實作為傳訊系統。 Java Platform, Enterprise Edition Connector Architecture (JCA) 提供將在 Java EE 環境中執行的應用程式連接至「企業資訊系統 (EIS)」(例如 IBM MQ 或 Db2) 的標準方式。

如需相關資訊，請參閱第 70 頁的『為何應該使用 IBM MQ classes for JMS?』和第 72 頁的『從 Java 存取 IBM MQ -API 的選項』。

IBM MQ classes for Java

IBM MQ classes for Java 可讓您在 Java 環境中使用 IBM MQ 。 IBM MQ classes for Java 可讓 Java 應用程式以 IBM MQ 用戶端身分連接至 IBM MQ ，或直接連接至 IBM MQ 佇列管理程式。

IBM MQ classes for Java 會封裝「訊息佇列介面 (MQI)」(原生 IBM MQ API) ，並使用與其他物件導向介面相同的物件模型，而 IBM MQ classes for JMS 及 IBM MQ classes for Jakarta Messaging 會分別從 Oracle 及 Java Community Process 實作 Java 傳訊介面。

如需相關資訊，請參閱第 292 頁的『為何應該使用 IBM MQ classes for Java?』和第 72 頁的『從 Java 存取 IBM MQ -API 的選項』。

註:

Stabilized IBM 將不會進一步加強 IBM MQ classes for Java，而且其功能層次將穩定為 IBM MQ 8.0 隨附的層次。 繼續完全支援使用 IBM MQ classes for Java 的現有應用程式，但不會新增特性，且會拒絕加強功能要求。 完全支援表示問題報告修正將與「IBM MQ 系統需求」變更而引發的任何必要變更一併進行。

IMS 中不支援 IBM MQ classes for Java 。

WebSphere Liberty 中不支援 IBM MQ classes for Java 。 它們不得與 IBM MQ Liberty 傳訊特性或一般 JCA 支援一起使用。 如需相關資訊，請參閱 在 J2EE/JEE 環境中使用 WebSphere MQ Java 介面。

使用 IBM MQ classes for JMS/Jakarta Messaging

IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 是 IBM MQ 隨附的 Java 傳訊提供者。除了實作 JMS 和 Jakarta Messaging 規格中定義的介面，這些傳訊提供者也會將兩組延伸新增至 Java 傳訊 API。

V 9.3.0 JM 3.0 V 9.3.0 從 IBM MQ 9.3.0 開始，支援 Jakarta Messaging 3.0 開發新的應用程式。 IBM MQ 9.3.0 繼續支援 JMS 2.0 現有的應用程式。 不支援在同一應用程式中同時使用 JMS 2.0 API 和 Jakarta Messaging 3.0 API。

註: 對於 Jakarta Messaging 3.0， JMS 規格的控制項會從 Oracle 移至 Java Community Process。 不過，Oracle 會保留 "javax" 名稱的控制權，該名稱用於尚未移至 Java Community Process 的其他 Java 技術中。因此，雖然 Jakarta Messaging 3.0 在功能上等同於 JMS 2.0 ，但在命名方面仍有一些差異:

- 3.0 版的正式名稱是 Jakarta Messaging ，而不是 Java Message Service。
- 套件及常數名稱會以 `jakarta` 而非 `javax` 作為字首。 例如，在 JMS 2.0 中，傳訊提供者的起始連線是 `javax.jms.Connection` 物件，在 Jakarta Messaging 3.0 中是 `jakarta.jms.Connection` 物件。

JMS 2.0 javax.jms 套件定義 JMS 介面， JMS 提供者會針對特定的傳訊產品實作這些介面。 IBM MQ classes for JMS 是一個 JMS 提供者，用於實作 IBM MQ 的 JMS 介面。

JM 3.0 jakarta.jms 套件定義 Jakarta Messaging 介面， Jakarta Messaging 提供者會針對特定的傳訊產品實作這些介面。 IBM MQ classes for Jakarta Messaging 是一個 Jakarta Messaging 提供者，用於實作 IBM MQ 的 Jakarta Messaging 介面。

JMS 和 Jakarta Messaging 規格預期 ConnectionFactory 和「目的地」物件是受管理物件。 管理者會在中央儲存庫中建立及維護受管理物件，而 JMS 或 Jakarta Messaging 應用程式會使用 Java Naming Directory Interface (JNDI) 來擷取這些物件。

JMS 2.0 對於 JMS 2.0，管理者可以使用 IBM MQ JMS 管理工具 **JMSAdmin** 或 IBM MQ Explorer，在中央儲存庫中建立及維護受管理物件。

JM 3.0 對於 Jakarta Messaging 3.0，您無法使用 IBM MQ Explorer 來管理 JNDI。 **JMSAdmin** 的 Jakarta Messaging 3.0 變式 **JMS30Admin** 支援 JNDI 管理。

因為 JMS 和 Jakarta Messaging 有許多共同之處，本主題中對 JMS 的進一步參照可以視為同時參照兩者。必要時會強調顯示任何差異。

IBM MQ classes for JMS 也提供兩組 JMS API 延伸。 這些延伸的主要焦點關注在執行時期動態地建立及配置 Connection Factory 和目的地，但這些延伸也提供與傳訊不直接相關的功能，如問題判斷的功能。

IBM MQ JMS 延伸
IBM MQ classes for JMS 包含在物件 (例如 MQConnectionFactory、MQQueue 及 MQTopic 物件) 中實作的延伸。 這些物件具有特定於 IBM MQ 的內容及方法。 物件可以是受管理物件，或應用程式可以在執行時期動態地建立物件。 這些延伸稱為 IBM MQ JMS 延伸。

IBM JMS 延伸
IBM MQ classes for JMS 也為 JMS API 提供一組更通用的延伸，這些延伸不是 IBM MQ 作為傳訊系統所特有，或 Java 作為所使用的程式設計語言。 這些延伸稱為 IBM JMS 延伸，具有下列廣泛目標:

- 在 IBM JMS 提供者之間提供更高層次的一致性。
- 更容易在兩個 IBM 傳訊系統之間撰寫橋接器應用程式。
- 更容易將應用程式從一個 IBM JMS 提供者移轉至另一個提供者。

延伸提供的功能類似於 IBM MQ Message Service Client (XMS) for C/C++ 和 IBM MQ Message Service Client (XMS) for .NET 中提供的功能。

相關概念
IBM MQ Java 語言介面
相關工作
第 118 頁的『撰寫 IBM MQ classes for JMS/Jakarta Messaging 應用程式』
在簡要介紹 JMS 模型之後，本節提供如何撰寫 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 應用程式的詳細指引。

JM 3.0 V 9.3.0 V 9.3.0 為何應該使用 IBM MQ classes for Jakarta Messaging?

使用 IBM MQ classes for Jakarta Messaging 有許多優點，包括能夠在組織中重複使用任何現有的 Jakarta Messaging 技能，以及應用程式更獨立於 Jakarta Messaging 提供者及基礎 IBM MQ 配置。

使用 IBM MQ classes for Jakarta Messaging 的優點摘要

使用 IBM MQ classes for Jakarta Messaging 可讓您重複使用現有的 Jakarta Messaging 技能，並提供應用程式獨立性。

- 您可以重複使用 Jakarta Messaging 技能。

  IBM MQ classes for Jakarta Messaging 是一個 Jakarta Messaging 提供者，將 IBM MQ 的 Jakarta Messaging 介面實作為傳訊系統。 如果您的組織是 IBM MQ 的新手，但已具備 Jakarta Messaging (或 JMS) 應用程式開發技能，您可能會發現使用熟悉的 Jakarta Messaging API 來存取 IBM MQ 資源比使用 IBM MQ 所提供的其他 API 更容易。
- Jakarta Messaging 是 Jakarta EE 的一部分。

  Jakarta Messaging 是在 Jakarta EE 平台上用於傳訊的自然 API。 每一個符合 Jakarta EE 標準的應用程式伺服器都必須包含 Jakarta Messaging 提供者。 您可以在應用程式用戶端、Servlet、 Java Server Pages (JSP)、Enterprise Java Bean (EJB) 及訊息驅動 Bean (MDB) 中使用 Jakarta Messaging 。 請特別注意，

Jakarta EE 應用程式會以非同步方式使用 MDB 來處理訊息，且所有訊息都會以 Jakarta Messaging 訊息形式遞送至 MDB。

- Connection Factory 及目的地可以儲存為中央儲存庫中的 Jakarta Messaging 受管理物件，而不是寫在應用程式中。

  管理者可以在中央儲存庫中建立及維護 Jakarta Messaging 受管理物件，且 IBM MQ classes for Jakarta Messaging 應用程式可以使用 Java Naming Directory Interface (JNDI) 來擷取這些物件。 Jakarta Messaging Connection Factory 和目的地會封裝 IBM MQ 特定資訊，例如佇列管理程式名稱、通道名稱、連線選項、佇列名稱和主題名稱。 如果 Connection Factory 和目的地儲存成受管理物件，這項資訊不會寫在應用程式中。 因此，此安排提供應用程式與基礎 IBM MQ 配置的獨立程度。

- Jakarta Messaging 是一種業界標準 API，可提供應用程式可攜性。

  Jakarta Messaging 應用程式可以使用 JNDI 來擷取儲存為受管理物件的 Connection Factory 及目的地，並僅使用 `jakarta.jms` (Jakarta Messaging 3.0) 套件中定義的介面來執行傳訊作業。 然後，應用程式會完全獨立於任何 Jakarta Messaging 提供者 (例如 IBM MQ classes for Jakarta Messaging)，且可以從一個 Jakarta Messaging 提供者移轉至另一個提供者，而無需對應用程式進行任何變更。

  如果在特定應用程式環境中無法使用 JNDI，則 IBM MQ classes for Jakarta Messaging 應用程式可以使用 Jakarta Messaging API 的延伸，在執行時期動態建立及配置 Connection Factory 和目的地。 然後，應用程式會完全自行包含，但與作為 Jakarta Messaging 提供者的 IBM MQ classes for Jakarta Messaging 相關聯。

- 透過使用 Jakarta Messaging，橋接器應用程式可能更容易撰寫。

  橋接器應用程式是從一個傳訊系統接收訊息，並將它們傳送至另一個傳訊系統的應用程式。 使用產品特定的 API 和訊息格式，撰寫橋接器應用程式可能會很複雜。 相反地，您可以使用兩個 Jakarta Messaging 提供者來撰寫橋接器應用程式，每個傳訊系統一個提供者。 然後應用程式只會使用一個 API，即 Jakarta Messaging API，且只會處理 Jakarta Messaging 訊息。

可部署的環境

為提供與 Jakarta EE 應用程式伺服器的整合，Jakarta EE 標準需要傳訊提供者以供應資源配接器。 遵循 Jakarta Connectors Architecture 規格， IBM MQ 提供一個資源配接器，利用 Jakarta Messaging 在任何經過認證的 Jakarta EE 環境內提供傳訊功能。 如需相關資訊，請參閱第 370 頁的『Liberty 和 IBM MQ 資源配接器』。

註: WebSphere Application Server traditional 目前不支援 Jakarta EE。

在 Jakarta EE 環境的外部，提供了 OSGi 及 JAR 檔，可讓您更輕鬆地僅取得 IBM MQ classes for Jakarta Messaging。 這些 JAR 檔更容易部署在獨立式或軟體管理架構 (例如 Maven) 內。 如需相關資訊，請參閱第 107 頁的『個別取得 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging』。

相關概念

IBM MQ Classes for Jakarta Messaging: 概觀

第 72 頁的『從 Java 存取 IBM MQ -API 的選項』

IBM MQ 提供三個 Java 語言介面。

JMS 2.0 為何應該使用 IBM MQ classes for JMS?

使用 IBM MQ classes for JMS 有許多優點，包括能夠在組織中重複使用任何現有的 JMS 技能，以及應用程式更獨立於 JMS 提供者及基礎 IBM MQ 配置。

使用 IBM MQ classes for JMS 的優點摘要

使用 IBM MQ classes for JMS 可讓您重複使用現有的 JMS 技能，並提供應用程式獨立性。

V 9.3.0 V 9.3.0

註: JMS 2.0 已由 Jakarta Messaging 取代。 IBM MQ classes for JMS 繼續支援 JMS 2.0 標準，但 Java 傳訊的未來加強功能只會出現在 Jakarta Messaging 中，因此會出現在 IBM MQ classes for Jakarta Messaging

中。 僅建議使用 IBM MQ classes for JMS 來維護及延伸現有的 JMS 2.0 應用程式。 IBM MQ classes for Jakarta Messaging 應該是新開發的偏好技術。

- 您可以重複使用 JMS 技能。

  IBM MQ classes for JMS 是一個 JMS 提供者，將 IBM MQ 的 JMS 介面實作為傳訊系統。 如果您的組織不熟悉 IBM MQ，但已具備 JMS 應用程式開發技能，您可能會發現使用熟悉的 JMS API 來存取 IBM MQ 資源比使用 IBM MQ 所提供的其中一個其他 API 更容易。

- JMS 是 Java Platform, Enterprise Edition (Java EE) 的一部分。

  JMS 是在 Java EE 平台上用於傳訊的自然 API。 每一個符合 Java EE 標準的應用程式伺服器都必須包含 JMS 提供者。 您可以在應用程式用戶端、Servlet、 Java Server Pages (JSP)、Enterprise Java Bean (EJB) 及訊息驅動 Bean (MDB) 中使用 JMS 。 請特別注意， Java EE 應用程式會以非同步方式使用 MDB 來處理訊息，且所有訊息都會以 JMS 訊息形式遞送至 MDB。

- Connection Factory 及目的地可以儲存為中央儲存庫中的 JMS 受管理物件，而不是寫在應用程式中。

  管理者可以在中央儲存庫中建立及維護 JMS 受管理物件，且 IBM MQ classes for JMS 應用程式可以使用 Java Naming Directory Interface (JNDI) 來擷取這些物件。 JMS Connection Factory 和目的地會封裝 IBM MQ 特定資訊，例如佇列管理程式名稱、通道名稱、連線選項、佇列名稱和主題名稱。 如果 Connection Factory 和目的地儲存成受管理物件，這項資訊不會寫在應用程式中。 因此，此安排提供應用程式與基礎 IBM MQ 配置的獨立程度。

- JMS 是一種業界標準 API ，可提供應用程式可攜性。

  JMS 應用程式可以使用 JNDI 來擷取儲存為受管理物件的 Connection Factory 及目的地，並僅使用 `javax.jms` 套件中定義的介面來執行傳訊作業。 然後，應用程式會完全獨立於任何 JMS 提供者 (例如 IBM MQ classes for JMS) ，且可以從一個 JMS 提供者移轉至另一個提供者，而無需對應用程式進行任何變更。

  如果在特定應用程式環境中無法使用 JNDI ，則 IBM MQ classes for JMS 應用程式可以使用 JMS API 的延伸，在執行時期動態建立及配置 Connection Factory 和目的地。 然後，應用程式會完全自行包含，但與作為 JMS 提供者的 IBM MQ classes for JMS 相關聯。

- 透過使用 JMS，橋接器應用程式可能更容易撰寫。

  橋接器應用程式是從一個傳訊系統接收訊息，並將它們傳送至另一個傳訊系統的應用程式。 使用產品特定的 API 和訊息格式，撰寫橋接器應用程式可能會很複雜。 相反地，您可以使用兩個 JMS 提供者來撰寫橋接器應用程式，每個傳訊系統一個提供者。 然後應用程式只會使用一個 API ，即 JMS API ，且只會處理 JMS 訊息。

可部署的環境

為提供與 Java EE 應用程式伺服器的整合，Java EE 標準需要傳訊提供者以供應資源配接器。 透過遵循 Java EE Connector Architecture (JCA) 規格，IBM MQ 提供了資源配接器，資源配接器使用 JMS 在任何認證的 Java EE 環境內提供傳訊功能。

雖然可以在 Java EE 內使用 IBM MQ classes for Java ，但此 API 未針對此目的進行工程或最佳化。 如需 Java EE 內 IBM MQ classes for Java 考量的相關資訊，請參閱 第 292 頁的『在 Java EE 內執行 IBM MQ classes for Java 應用程式』。

在 Java EE 環境的外部，提供了 OSGi 及 JAR 檔，可讓您更輕鬆地僅取得 IBM MQ classes for JMS。 這些 JAR 檔更容易部署在獨立式或軟體管理架構 (例如 Maven) 內。 如需相關資訊，請參閱 第 107 頁的『個別取得 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging』。

相關概念

V 9.3.0 V 9.3.0 IBM MQ Classes for Jakarta Messaging: 概觀

第 69 頁的『為何應該使用 IBM MQ classes for Jakarta Messaging?』
使用 IBM MQ classes for Jakarta Messaging 有許多優點，包括能夠在組織中重複使用任何現有的 Jakarta Messaging 技能，以及應用程式更獨立於 Jakarta Messaging 提供者及基礎 IBM MQ 配置。

第 72 頁的『從 Java 存取 IBM MQ -API 的選項』
IBM MQ 提供三個 Java 語言介面。

從 Java 存取 IBM MQ -API 的選項

IBM MQ 提供三個 Java 語言介面。

- V 9.3.0 IBM MQ classes for Jakarta Messaging
- IBM MQ classes for JMS
- IBM MQ classes for Java

IBM MQ classes for Jakarta Messaging

V 9.3.0 IBM MQ classes for Jakarta Messaging 容許使用 Jakarta Messaging 3.0 API 撰寫的應用程式利用 IBM MQ 作為傳訊提供者。

Jakarta Messaging 是 Java 應用程式中傳訊的策略方向。

Jakarta Messaging 3.0 在功能上等同於 JMS 2.0，因此如需相關資訊，請參閱 第 68 頁的『使用 IBM MQ classes for JMS/Jakarta Messaging』。

IBM MQ classes for JMS

IBM MQ classes for JMS 容許使用 JMS 2.0 API 撰寫的應用程式利用 IBM MQ 作為傳訊提供者。

作為 Jakarta Messaging 接替 JMS，建議在現有應用程式或不支援 Jakarta Messaging 的環境 (例如，WebSphere Application Server) 中使用 IBM MQ classes for JMS 。

不支援在相同應用程式中同時使用 IBM MQ classes for Jakarta Messaging 和 IBM MQ classes for JMS 。

如需相關資訊，請參閱 第 68 頁的『使用 IBM MQ classes for JMS/Jakarta Messaging』。

IBM MQ classes for Java

Stabilized Java 應用程式可用來存取 IBM MQ 資源的另一個 API 是 IBM MQ classes for Java，它提供 IBM MQ 導向 API，供程式使用 IBM MQ 作為傳訊提供者。 不過， IBM MQ classes for Java 在功能上已穩定在 IBM MQ 8.0 隨附的層次。 如需相關資訊，請參閱第 292 頁的『為何應該使用 IBM MQ classes for Java?』。 雖然繼續完全支援使用 IBM MQ classes for Java 的現有應用程式，但新的應用程式應該使用 IBM MQ classes for Jakarta Messaging。

IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 的一般特性

IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 可讓您同時存取 IBM MQ 的點對點和發佈/訂閱傳訊特性。 應用程式不僅傳送可為 JMS 標準傳訊模型提供支援的 JMS 訊息，還可傳送及接收沒有其他標頭的訊息，因此可以與其他 IBM MQ 應用程式（例如，C MQI 應用程式）互相操作。 可以完全控制 MQMD 和 MQ 訊息有效負載。

還提供了進一步 IBM MQ 特性，如訊息多媒體串流、非同步放置及報告訊息。

使用提供的 PCF Helper 類別， IBM MQ PCF 管理訊息可以透過 JMS API 傳送及接收，並可用來管理佇列管理程式。

最近新增至 IBM MQ 的特性 (例如非同步使用及自動重新連線) 在 IBM MQ classes for Java 中無法使用，但在 IBM MQ classes for JMS 及 IBM MQ classes for Jakarta Messaging 中可以使用。

要求加強功能

如果您需要存取無法透過 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 提供的 IBM MQ 特性，您可以提出構想。

然後， IBM 可以建議是否可以在 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 實作中進行實作，或者是否可以遵循最佳作法。

對於其他傳訊特性，由於 IBM 是開放式標準的貢獻者，因此可以在 JCP 程序中提出這些特性。 這些只適用於 Jakarta Messaging。

相關資訊
歡迎使用 IBM Ideas 入口網站
JMS Java 規格檢閱程序
使用 JMS 來傳送 PCF 訊息

JM 3.0 V 9.3.0 V 9.3.0 IBM MQ classes for Jakarta Messaging 的必備項目

本主題告訴您在使用 IBM MQ classes for Jakarta Messaging 之前需要知道的內容。 若要開發及執行 IBM MQ classes for Jakarta Messaging 應用程式，您需要某些軟體元件作為必要條件。

如需 IBM MQ classes for Jakarta Messaging 必要條件的相關資訊，請參閱 IBM MQ 的系統需求。

若要開發 IBM MQ classes for Jakarta Messaging 應用程式，您需要 Java SE Software Development Kit (SDK)。 如需作業系統所支援之 JDK 的詳細資料，請參閱 IBM MQ 的系統需求。

若要執行 IBM MQ classes for Jakarta Messaging 應用程式，您需要下列軟體元件:

- IBM MQ 佇列管理程式。
- Java runtime environment (JRE) ，適用於您執行應用程式的每一個系統。
- IBM i 若為 IBM i，則為 Qshell ，這是作業系統的選項 30。
- z/OS 若為 z/OS， z/OS UNIX System Services (z/OS UNIX)。

IBM JSSE 提供者包含 FIPS 認證的加密提供者，因此可以透過程式設計方式配置以符合 FIPS 140-2 標準，備妥可立即使用。 因此，可以直接從 IBM MQ classes for Jakarta Messaging 支援 FIPS 140-2 相符性。

Oracle 的 JSSE 提供者可以在其中配置 FIPS 認證的加密提供者，但這尚未備妥可供立即使用，無法用於程式化配置。 因此，在此情況下， IBM MQ classes for Jakarta Messaging 無法直接啟用 FIPS 140-2 相符性。 您可能可以手動啟用這類相符性，但 IBM 目前無法提供這方面的指引。

如果 Java 虛擬機器 (JVM) 及作業系統上的 TCP/IP 實作支援 IPv6 位址，您可以在 IBM MQ classes for Jakarta Messaging 應用程式中使用 Internet Protocol 第 6 版 (IPv6) 位址。 IBM MQ Jakarta Messaging 管理工具 **JMS30Admin** 也接受 IPv6 位址。 如需此工具的相關資訊，請參閱 使用管理工具來配置 JMS 及 Jakarta Messaging 物件。

IBM MQ JMS 管理工具和 IBM MQ Explorer 使用 Java Naming Directory Interface (JNDI) 來存取儲存受管理物件的目錄服務。 IBM MQ classes for Jakarta Messaging 應用程式也可以使用 JNDI ，從目錄服務擷取受管理物件。

註: 對於 Jakarta Messaging 3.0，您無法使用 IBM MQ Explorer 來管理 JNDI。 **JMSAdmin** 的 Jakarta Messaging 3.0 變式 **JMS30Admin** 支援 JNDI 管理。

服務提供者是透過將 JNDI 呼叫對映至目錄服務來提供目錄服務存取權的程式碼。 IBM MQ classes for Jakarta Messaging 提供了檔案 `fscontext.jar` 和 `providerutil.jar` 中的檔案系統服務提供者。 檔案系統服務提供者可讓您根據本端檔案系統來存取目錄服務。

如果您想要使用基於 LDAP 伺服器的目錄服務，則必須安裝並配置 LDAP 伺服器，或具有現有 LDAP 伺服器的存取權。 特別是，您必須配置 LDAP 伺服器來儲存 Java 物件。 如需如何安裝及配置 LDAP 伺服器的相關資訊，請參閱伺服器隨附的說明文件。

JMS 2.0 IBM MQ classes for JMS 的必備項目

本主題告訴您在使用 IBM MQ classes for JMS 之前需要知道的內容。 若要開發及執行 IBM MQ classes for JMS 應用程式，您需要某些軟體元件作為必要條件。

如需 IBM MQ classes for JMS 必要條件的相關資訊，請參閱 IBM MQ 的系統需求。

若要開發 IBM MQ classes for JMS 應用程式，您需要 Java SE Software Development Kit (SDK)。 如需作業系統所支援之 JDK 的詳細資料，請參閱 IBM MQ 的系統需求。

若要執行 IBM MQ classes for JMS 應用程式，您需要下列軟體元件:

- IBM MQ 佇列管理程式。
- Java runtime environment (JRE)，適用於您執行應用程式的每一個系統。
- IBM i 若為 IBM i，則為 Qshell，這是作業系統的選項 30。
- z/OS 若為 z/OS， z/OS UNIX System Services (z/OS UNIX)。

IBM JSSE 提供者包含 FIPS 認證的加密提供者，因此可以透過程式設計方式配置以符合 FIPS 140-2 標準，備妥可立即使用。 因此，可以直接從 IBM MQ classes for Java 和 IBM MQ classes for JMS 支援 FIPS 140-2 相符性。

Oracle 的 JSSE 提供者可以在其中配置 FIPS 認證的加密提供者，但這尚未備妥可供立即使用，無法用於程式化配置。 因此，在此情況下， IBM MQ classes for Java 和 IBM MQ classes for JMS 無法直接啟用 FIPS 140-2 相符性。 您可能可以手動啟用這類相符性，但 IBM 目前無法提供這方面的指引。

如果 Java 虛擬機器 (JVM) 及作業系統上的 TCP/IP 實作支援 IPv6 位址，您可以在 IBM MQ classes for JMS 應用程式中使用 Internet Protocol 第 6 版 (IPv6) 位址。 IBM MQ JMS 管理工具 (請參閱 使用管理工具來配置 JMS 物件) 也接受 IPv6 位址。

IBM MQ JMS 管理工具和 IBM MQ Explorer 使用 Java Naming Directory Interface (JNDI) 來存取儲存受管理物件的目錄服務。 IBM MQ classes for JMS 應用程式也可以使用 JNDI，從目錄服務擷取受管理物件。 服務提供者是透過將 JNDI 呼叫對映至目錄服務來提供目錄服務存取權的程式碼。 IBM MQ classes for JMS 提供了檔案 `fscontext.jar` 和 `providerutil.jar` 中的檔案系統服務提供者。 檔案系統服務提供者可讓您根據本端檔案系統來存取目錄服務。

如果您想要使用基於 LDAP 伺服器的目錄服務，則必須安裝並配置 LDAP 伺服器，或具有現有 LDAP 伺服器的存取權。 特別是，您必須配置 LDAP 伺服器來儲存 Java 物件。 如需如何安裝及配置 LDAP 伺服器的相關資訊，請參閱伺服器隨附的說明文件。

安裝及配置 IBM MQ classes for JMS/Jakarta Messaging

本節說明安裝 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 時所建立的目錄和檔案，並告訴您如何在安裝之後配置 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging。

相關概念

第 365 頁的『使用 IBM MQ 資源配接器』
資源配接器容許在應用程式伺服器中執行的應用程式存取 IBM MQ 資源。 它支援入埠和出埠通訊。

針對 *IBM MQ classes for JMS* 安裝的內容

當您安裝 IBM MQ classes for JMS 時，會建立一些檔案和目錄。 在 Windows 上，透過自動設定環境變數，在安裝期間執行部分配置。 在其他平台及某些 Windows 環境中，您必須先設定環境變數，然後才能執行 IBM MQ classes for JMS 應用程式。

對於大部分作業系統，當您安裝 IBM MQ 時， IBM MQ classes for JMS 會安裝為選用元件。

如需安裝 IBM MQ 的相關資訊，請參閱:

- Multi 安裝 IBM MQ
- z/OS 安裝 IBM MQ for z/OS

重要: 除了 第 76 頁的『IBM MQ classes for JMS/Jakarta Messaging 可重新定位 JAR 檔』中說明的可再定位 JAR 檔之外，不支援將 IBM MQ classes for JMS JAR 檔或原生程式庫複製到其他機器，或複製到已安裝 IBM MQ classes for JMS 之機器上的不同位置。

安裝目錄

第 75 頁的表 5 顯示每一個平台上安裝 IBM MQ classes for JMS 檔案的位置。

表 5: *IBM MQ classes for JMS* 安裝目錄	
平台	**目錄**
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*/java
Windows Windows	*MQ_INSTALLATION_PATH*\java
IBM i IBM i	/QIBM/ProdData/mqm/java
z/OS z/OS	*MQ_INSTALLATION_PATH*/java

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

安裝目錄包括下列內容:

- IBM MQ classes for JMS JAR 檔 (包括可重新定位的 JAR 檔) 位於 *MQ_INSTALLATION_PATH*\java\lib 目錄中。
- IBM MQ 原生程式庫，由使用 Java 原生介面的應用程式使用。

  32 位元原生程式庫會安裝到 *MQ_INSTALLATION_PATH*\java\lib 目錄中， 64 位元原生程式庫可以在 *MQ_INSTALLATION_PATH*\java\lib64 目錄中找到。

  如需 IBM MQ 原生程式庫的相關資訊，請參閱 第 81 頁的『配置 Java 原生介面 (JNI) 程式庫』。
- 第 104 頁的『IBM MQ classes for JMS/Jakarta Messaging 隨附的 Script』中說明的其他 Script。 這些 Script 位於 *MQ_INSTALLATION_PATH*\java\bin 目錄中。
- IBM MQ classes for JMS API 的規格。 Javadoc 工具已用來產生包含 API 規格的 HTML 頁面。

  HTML 頁面位於 *MQ_INSTALLATION_PATH*\java\doc\WMQJMSClasses 目錄中:

  – ALW 在 AIX, Linux, and Windows 上，此子目錄包含個別 HTML 頁面。

  – IBM i 在 IBM i 上， HTML 頁面位於稱為 wmqjms_javadoc.jar 的檔案中。

  – z/OS 在 z/OS 上， HTML 頁面位於稱為 wmqjms_javadoc.jar 的檔案中。
- 支援 OGSi。 OSGi 軟體組安裝在 java\lib\OSGi 目錄中，並在 第 105 頁的『支援 OSGi 與 IBM MQ classes for JMS』中說明。
- IBM MQ 資源配接器，可部署至任何符合 Java Platform, Enterprise Edition 7 (Java EE 7) 或 Jakarta EE 標準的應用程式伺服器。

  IBM MQ 資源配接器位於 *MQ_INSTALLATION_PATH*\java\lib\jca 目錄中; 如需相關資訊，請參閱 第 365 頁的『使用 IBM MQ 資源配接器』
- Windows 在 Windows 上，可用來除錯的符號安裝在 *MQ_INSTALLATION_PATH*\java\lib\symbols 目錄中。

安裝目錄也包括屬於其他 IBM MQ 元件的部分檔案。

範例應用程式

JMS 2.0 部分範例應用程式隨 IBM MQ classes for JMS 一起提供。 第 75 頁的表 6 顯示範例應用程式在每一個平台上的安裝位置。

JM 3.0 對於 IBM MQ classes for Jakarta Messaging，正在準備新的範例。

JMS 2.0

表 6: *IBM MQ classes for JMS* 的範例目錄	
平台	**目錄**
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*/samp/jms
Windows Windows	*MQ_INSTALLATION_PATH*\tools\jms
IBM i IBM i	/QIBM/ProdData/mqm/java/samples/jms
z/OS z/OS	*MQ_INSTALLATION_PATH*/java/samples/jms

在此表格中， *MQ_INSTALLATION_PATH* 代表 IBM MQ 安裝所在的高階目錄。

安裝之後，您可能需要執行一些配置作業來編譯及執行應用程式。

第 78 頁的『設定 IBM MQ classes for JMS/Jakarta Messaging 的環境變數』 說明執行範例 IBM MQ classes for JMS 應用程式所需的類別路徑。 本主題也說明在特殊情況下需要參照的其他 JAR 檔，以及您必須設定以執行 IBM MQ classes for JMS 隨附的 Script 的環境變數。

若要控制內容 (例如應用程式的追蹤及記載) ，您需要提供配置內容檔。 第 83 頁的『IBM MQ classes for JMS/Jakarta Messaging 配置檔』中說明 IBM MQ classes for JMS 配置內容檔。

相關概念

部署資源配接器時發生問題

相關工作

第 101 頁的『使用 IBM MQ classes for JMS 範例應用程式』
IBM MQ classes for JMS 範例應用程式提供 JMS API 的一般特性概觀。 您可以使用它們來驗證安裝及傳訊伺服器的設定，並協助您建置自己的應用程式。

IBM MQ classes for JMS/Jakarta Messaging 可重新定位 *JAR* 檔

可重新定位的 JAR 檔可以移至需要執行 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 的系統。

重要:

- 除了 可再定位 JAR 檔中說明的可再定位 JAR 檔之外，不支援將 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging JAR 檔或原生程式庫複製到其他機器，或複製到已安裝 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 的機器上的不同位置。
- 在部署至 Java EE 應用程式伺服器 (例如 WebSphere Application Server 或 WebSphere Liberty) 的應用程式內，請勿包含可重新定位的 JAR 檔。 在這些環境中，應該部署並改用 IBM MQ 資源配接器。 請注意，WebSphere Application Server 會內嵌 IBM MQ 資源配接器，因此不需要手動將它部署到這個環境中。
- 為了避免類別載入器衝突，不建議將可重新定位的 JAR 檔組合在相同 Java 執行時期內的多個應用程式內。 在此實務範例中，在 Java 執行時期的類別路徑上提供 IBM MQ 可重新定位 JAR 檔。
- 如果您要組合應用程式內的可再定位 JAR 檔，請確保包含 可再定位 JAR 檔中說明的所有必備 JAR 檔。 在應用程式維護過程中，您也應該確定有適當的程序來更新組合 JAR 檔，以確保 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 仍然是最新的，且會重新調解已知問題。

可再定位 JAR 檔

在企業內，下列檔案可以移至需要執行 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 的系統:

- V 9.3.0 V 9.3.0 bcpkix-jdk15to18.jar 第 77 頁的『4』
- V 9.3.5 bcpkix-jdk18on.jar 第 77 頁的『3』
- V 9.3.0 V 9.3.0 bcprov-jdk15to18.jar 第 77 頁的『4』

- V 9.3.5 bcprov-jdk18on.jar 第 77 頁的『3』
- V 9.3.0 V 9.3.0 bcutil-jdk15to18.jar 第 77 頁的『4』
- V 9.3.5 bcutil-jdk18on.jar 第 77 頁的『3』
- JMS 2.0 com.ibm.mq.allclient.jar 第 77 頁的『1』
- JM 3.0 V 9.3.0 V 9.3.0 com.ibm.mq.jakarta.client.jar 第 77 頁的『2』
- V 9.3.3 Removed com.ibm.mq.traceControl.jar
- fscontext.jar
- V 9.3.3 jackson-annotations.jar
- V 9.3.3 jackson-core.jar
- V 9.3.3 jackson-databind.jar
- jakarta.jms-api.jar
- jms.jar
- org.json.jar
- providerutil.jar

附註:

1. JMS 2.0 及 JMS 1.1
2. Jakarta Messaging 3.0
3. Continuous Delivery from IBM MQ 9.3.5
4. Long Term Support 及 Continuous Delivery 之前 IBM MQ 9.3.5

JMS JAR 檔

V 9.3.0 V 9.3.0 jms.jar 包含 JMS 1.1 和 JMS 2.0 介面-這些介面命名為 javax.jms.*。

JM 3.0 V 9.3.0 V 9.3.0 jakarta.jms-api.jar 包含 Jakarta Messaging 3.0 介面-這些介面命名為 jakarta.jms.*。

fscontext.jar 和 providerutil.jar

如果您的應用程式使用檔案系統環境定義來執行 JNDI 查閱，則需要 fscontext.jar 和 providerutil.jar 檔案。

Bouncy Castle 安全提供者和 CMS 支援 JAR 檔

需要 Bouncy Castle 安全提供者和 CMS 支援 JAR 檔。 如需相關資訊，請參閱 支援非 IBM 具備 AMS 的 JRE。

V 9.3.5 對於 IBM MQ 9.3.5 中的 Continuous Delivery，需要下列 JAR 檔:

- bcpkix-jdk18on.jar
- bcprov-jdk18on.jar
- bcutil-jdk18on.jar

LTS 對於 Long Term Support 和 IBM MQ 9.3.5 之前的 Continuous Delivery，需要下列 JAR 檔:

- bcpkix-jdk15to18.jar
- bcprov-jdk15to18.jar

- bcutil-jdk15to18.jar

org.json.jar

如果您的 IBM MQ classes for JMS 應用程式使用 JSON 格式的 CCDT，則需要 org.json.jar 檔案。

com.ibm.mq.allclient.jar 和 com.ibm.mq.jakarta.client.jar

檔案 com.ibm.mq.allclient.jar 和 com.ibm.mq.jakarta.client.jar 包含 IBM MQ classes for JMS、 IBM MQ classes for Jakarta Messaging、 IBM MQ classes for Java 以及 PCF 和標頭類別。 如果您將這些 JAR 檔移至新位置，請確定您已採取步驟，以使用新的「IBM MQ 修正套件」來維護這個新位置。 此外，如果您取得臨時修正程式，請確定「IBM 支援中心」已知道這些檔案的使用。

若要判定檔案 com.ibm.mq.allclient.jar 及 com.ibm.mq.jakarta.client.jar 的版本，請使用下列指令:

JM 3.0 V 9.3.0 V 9.3.0

```
java -jar com.ibm.mq.jakarta.client.jar
```

JMS 2.0

```
java -jar com.ibm.mq.allclient.jar
```

下列範例顯示此指令的部分範例輸出:

```
C:\Program Files\IBM\MQ_1\java\lib>java -jar com.ibm.mq.allclient.jar
Name:       Java Message Service Client
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       WebSphere MQ classes for Java Message Service
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       WebSphere MQ JMS Provider
Version:    9.3.0.0
Level:      p000-L140428.1 mqjbnd=p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       Common Services for Java Platform, Standard Edition
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar
```

jackson-annotations.jar、 jackson-core.jar 和 jackson-databind.jar

V 9.3.3

如果您的 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 應用程式建立與佇列管理程式的安全 TLS 連線，則需要三個 Jackson JAR 檔。

設定 *IBM MQ classes for JMS/Jakarta Messaging* 的環境變數
在編譯及執行 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 應用程式之前，**CLASSPATH** 環境變數的設定必須包含 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging Java 保存檔 (JAR)。 視您的需求而定，您可能需要將其他 JAR 檔新增至類別路徑。 若要執行 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 隨附的 Script，必須設定其他環境變數。

開始之前

JM 3.0 V 9.3.0 V 9.3.0 從 IBM MQ 9.3.0 開始，支援 Jakarta Messaging 3.0 開發新的應用程式。 IBM MQ 9.3.0 繼續支援 JMS 2.0 現有的應用程式。 不支援在同一應用程式中同時使用 Jakarta Messaging 3.0 API 和 JMS 2.0 API。 如需相關資訊，請參閱 使用 IBM MQ 類別進行 JMS/Jakarta 傳訊。

重要: 不支援將 Java 選項 *-Xbootclasspath* 設為包含 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 。

關於這項作業

若要編譯並執行 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 應用程式，請使用平台及 Java 傳訊版本的 **CLASSPATH** 設定，如下表所示。 或者，您可以在 **java** 指令上指定類別路徑，而不使用環境變數。

JMS 2.0 對於 IBM MQ classes for JMS，此設定包括 samples 目錄，因此您可以編譯並執行 IBM MQ classes for JMS 範例應用程式。

JM 3.0 對於 IBM MQ classes for Jakarta Messaging，正在準備新的範例。

JM 3.0

表 *7:* ***CLASSPATH*** 設定，供 *Jakarta Messaging 3.0* 編譯及執行 *IBM MQ classes for Jakarta Messaging* 應用程式

平台	CLASSPATH 設定
AIX AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jakarta.client.jar:
Linux Linux	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jakarta.client.jar:
IBM i IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar:
Windows Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\java\lib\com.ibm.mq.jakarta.client.jar;
z/OS z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jakarta.client.jar;

JMS 2.0

表 *8:* ***CLASSPATH*** 設定，供 *JMS 2.0* 編譯及執行 *IBM MQ classes for JMS* 應用程式，包括範例應用程式

平台	CLASSPATH 設定
AIX AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.allclient.jar: *MQ_INSTALLATION_PATH*/samp/jms/samples:
Linux Linux	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.allclient.jar: *MQ_INSTALLATION_PATH*/samp/jms/samples:
IBM i IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar: /QIBM/ProdData/mqm/java/samples/jms/samples:
Windows Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\java\lib\com.ibm.mq.allclient.jar; *MQ_INSTALLATION_PATH*\tools\jms\samples;

表 8: ***CLASSPATH*** 設定，供 *JMS 2.0* 編譯及執行 *IBM MQ classes for JMS* 應用程式，包括範例應用程式 (繼續)	
平台	**CLASSPATH 設定**
z/OS z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.allclient.jar: *MQ_INSTALLATION_PATH*/java/samples/jms/samples:

在這些表格中， *MQ_INSTALLATION_PATH* 代表 IBM MQ 安裝所在的高階目錄。

JAR 檔 `com.ibm.mq.jakarta.client.jar` 或 `com.ibm.mq.allclient.jar` 的資訊清單包含 IBM MQ classes for JMS 應用程式所需的大部分其他 JAR 檔的參照，因此您不需要將這些 JAR 檔新增至類別路徑。 這些 JAR 檔包括使用「Java Naming Directory Interface (JNDI)」從目錄服務擷取受管理物件的應用程式，以及使用「Java 交易 API (JTA)」的應用程式所需要的那些 JAR 檔。

不過，在下列情況下，您必須在類別路徑中包含其他 JAR 檔:

- 如果您使用通道結束程式類別來實作 `com.ibm.mq` 套件中定義的通道結束程式介面，而不是 `com.ibm.mq.exits` 套件中定義的通道結束程式介面，則必須將 IBM MQ classes for Java JAR 檔 `com.ibm.mq.jar` 新增至類別路徑。
- 如果您的應用程式使用 JNDI 從目錄服務擷取受管理物件，您也必須將下列 JAR 檔新增至類別路徑:
  - `fscontext.jar`
  - `providerutil.jar`
- 如果您的應用程式使用 JTA，您也必須將 `jta.jar` 新增至類別路徑。

註: 只有編譯應用程式，而不是執行應用程式，才需要這些額外的 JAR 檔。

IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 隨附的 Script 使用下列環境變數:

MQ_JAVA_DATA_PATH
此環境變數指定日誌和追蹤輸出的目錄。

MQ_JAVA_INSTALL_PATH
此環境變數指定 IBM MQ classes for JMS 的安裝目錄。

MQ_JAVA_LIB_PATH
此環境變數指定 IBM MQ classes for JMS 程式庫的儲存目錄，如先前表格中所示。

程序

- Windows
  在 Windows 上，安裝 IBM MQ 之後，請執行指令 **setmqenv**。
  如果您沒有先執行這個指令，當您發出 **dspmqver** 指令時，可能會出現下列錯誤訊息:
  ```
  AMQ8351: IBM MQ Java 環境
  正確，或尚未安裝 IBM MQ JRE 特性。
  ```
  註: 如果您未安裝 IBM MQ Java runtime environment (JRE)，則預期會出現此訊息 (請參閱 其他 Windows 特性必備項目檢查)。
- Linux AIX
  在 AIX and Linux 系統上，自行設定環境變數:

  JMS 2.0 對於 JMS 2.0，請使用下列其中一個 Script 來設定環境變數:
  - 如果您使用 32 位元 JVM，請使用 Script `setjmsenv`。
  - 如果您在 AIX 或 Linux 系統上使用 64 位元 JVM，請使用 Script `setjmsenv64`。

  JM 3.0 對於 Jakarta Messaging 3.0，請使用下列其中一個 Script 來設定環境變數:
  - 如果您使用 32 位元 JVM，請使用 Script `setjms30env`。
  - 如果您使用 64 位元 JVM，請使用 Script `setjms30env64`。

這些 Script 位於 *MQ_INSTALLATION_PATH*/java/bin 目錄中，其中 *MQ_INSTALLATION_PATH* 代表 IBM MQ 安裝所在的高階目錄。

您可以用各種方式來使用這些 Script。 您可以使用 Script 作為設定所需環境變數的基礎 (如表格中所示)，或使用文字編輯器將它們新增至 .profile 。 如果您有非一般設定，請視需要編輯 Script 內容。 或者，您可以在每一個要從中執行 JMS 啟動 Script 的階段作業中執行 Script。 如果您選擇此選項，則在 JMS 驗證處理程序期間，您需要在您啟動的每個 Shell 視窗中執行 Script:

- JMS 2.0 若為 JMS 2.0，請鍵入 . ./setjmsenv 或 ../setjmsenv64。
- JM 3.0 若為 Jakarta Messaging 3.0，請鍵入 . ./setjms30env 或 . ./setjms30env64。

IBM i 在 IBM i 上，您必須將環境變數 **QIBM_MULTI_THREADED** 設為 Y。 然後，您可以使用與執行單一執行緒應用程式相同的方式來執行多執行緒應用程式。 如需相關資訊，請參閱 使用 Java 和 JMS 來設定 IBM MQ。

相關工作

第 101 頁的『使用 IBM MQ classes for JMS 範例應用程式』
IBM MQ classes for JMS 範例應用程式提供 JMS API 的一般特性概觀。 您可以使用它們來驗證安裝及傳訊伺服器的設定，並協助您建置自己的應用程式。

相關參考

第 104 頁的『IBM MQ classes for JMS/Jakarta Messaging 隨附的 Script』
提供了一些 Script 來協助處理使用 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 時需要執行的一般作業。

配置 *Java* 原生介面 *(JNI)* 程式庫
IBM MQ classes for JMS 應用程式 (使用連結傳輸連接至佇列管理程式，或使用用戶端傳輸連接至佇列管理程式，並使用以非 Java 語言撰寫的通道結束程式) 需要在容許存取 Java 原生介面 (JNI) 程式庫的環境中執行。

開始之前

如需使用 WebSphere Application Server 環境的相關資訊，請參閱 使用原生程式庫資訊來配置 IBM MQ 傳訊提供者 。

關於這項作業

若要設定此環境，您必須配置環境的程式庫路徑，讓 Java Virtual Machine (JVM) 可以在啟動 IBM MQ classes for JMS 應用程式之前載入 mqjbnd 程式庫。

IBM MQ 提供兩個 Java 原生介面 (JNI) 程式庫:

mqjbnd
使用連結傳輸連接至佇列管理程式的應用程式會使用此程式庫。 它提供 IBM MQ classes for JMS 與佇列管理程式之間的介面。 隨 IBM MQ 9.3 一起安裝的 mqjbnd 程式庫可用來連接至任何 IBM MQ 9.3 (或更早版本) 佇列管理程式。

mqjexitstub02
當應用程式使用用戶端傳輸連接至佇列管理程式，並使用以非 Java 語言撰寫的通道結束程式時， IBM MQ classes for JMS 會載入 mqjexitstub02 程式庫。

在特定平台上， IBM MQ 會安裝這些 JNI 程式庫的 32 位元及 64 位元版本。 每一個平台的程式庫位置如 表 1 所示。

表 9: 每一個平台的 *IBM MQ classes for JMS* 程式庫位置	
平台	**包含 IBM MQ classes for JMS 程式庫的目錄**
AIX Linux (POWER、 x86-64 及 zSeries s390x 平台)	*MQ_INSTALLATION_PATH*/java/lib (32 位元程式庫) *MQ_INSTALLATION_PATH*/java/lib64 (64 位元程式庫)
Windows	*MQ_INSTALLATION_PATH*\java\lib (32 位元程式庫) *MQ_INSTALLATION_PATH*\java\lib64 (64 位元程式庫)
z/OS	*MQ_INSTALLATION_PATH*/java/lib (31 位元及 64 位元檔案庫)

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

註: z/OS 在 z/OS 上，您可以使用 31 位元或 64 位元 Java Virtual Machine (JVM)。 您不需要指定要使用哪些 JNI 程式庫; IBM MQ classes for JMS 可以自行決定要載入哪些 JNI 程式庫。

程序

1. 配置 JVM 的 **java.library.path** 內容，可透過兩種方式來完成:
   - 指定 JVM 引數，如下列範例所示:

     ```
     -Djava.library.path=path_to_library_directory
     ```

     Linux 例如，若為 Linux 上的 64 位元 JVM ，若為預設位置安裝，請指定:

     ```
     -Djava.library.path=/opt/mqm/java/lib64
     ```

   - 透過配置 Shell 的環境， JVM 將會設定自己的 `java.library.path`。 此路徑會因平台及您安裝 IBM MQ 的位置而有所不同。 例如，若為 64 位元 JVM 及預設 IBM MQ 安裝位置，您可以使用下列設定:

     AIX
     ```
     export LIBPATH=/usr/mqm/java/lib64:$LIBPATH
     ```

     Linux
     ```
     export LD_LIBRARY_PATH=/opt/mqm/java/lib64:$LD_LIBRARY_PATH
     ```

     Windows
     ```
     set PATH=C:\Program Files\IBM\MQ\java\lib64;%PATH%
     ```

   未正確配置環境時您看到的異常狀況堆疊範例如下:

```
原因: com.ibm.mq.jmqi.local.LocalMQ$4: CC=2;RC=2495;
AMQ8598: 無法載入 WebSphere MQ 原生 JNI 程式庫: 'mqjbnd '。
    at com.ibm.mq.jmqi.local.LocalMQ.loadLib(LocalMQ.java:1268)
    at com.ibm.mq.jmqi.local.LocalMQ$1.run(LocalMQ.java:309)
    位於 java.security.AccessController.doPrivileged(AccessController.java:400)
    at com.ibm.mq.jmqi.local.LocalMQ.initialise_inner(LocalMQ.java:259)
    at com.ibm.mq.jmqi.local.LocalMQ.initialise(LocalMQ.java:221)
    at com.ibm.mq.jmqi.local.LocalMQ.<init>(LocalMQ.java:1350)
    at com.ibm.mq.jmqi.local.LocalServer.<init>(LocalServer.java:230)
    at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
    at
sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:86)

    at
sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.jav
```

```
a:58)
    at java.lang.reflect.Constructor.newInstance(Constructor.java:542)
    at com.ibm.mq.jmqi.JmqiEnvironment.getInstance(JmqiEnvironment.java:706)
    at com.ibm.mq.jmqi.JmqiEnvironment.getMQI(JmqiEnvironment.java:640)
    at
com.ibm.msg.client.wmq.factories.WMQConnectionFactory.createV7ProviderConnection(WMQConnectionF
actory.java:8437)
    ... 7 more
原因: java.lang.UnsatisfiedLinkError: mqjbnd (在 java.library.path 中找不到)
    at java.lang.ClassLoader.loadLibraryWithPath(ClassLoader.java:1235)
    at java.lang.ClassLoader.loadLibraryWithClassLoader(ClassLoader.java:1205)
    at java.lang.System.loadLibrary(System.java:534)
    at com.ibm.mq.jmqi.local.LocalMQ.loadLib(LocalMQ.java:1240)
    ... 20 個以上
```

2. 設定 32 位元或 64 位元環境之後，請使用下列指令啟動 IBM MQ classes for JMS 應用程式:

```
java application-name
```

其中 *application-name* 是要執行的 IBM MQ classes for JMS 應用程式名稱。

在下列情況下， IBM MQ classes for JMS 會擲出包含 IBM MQ 原因碼 2495 (MQRC_MODULE_NOT_FOUND) 的異常狀況:

- IBM MQ classes for JMS 應用程式在 32 位元 Java runtime environment 中執行，且已針對 IBM MQ classes for JMS 設定 64 位元環境，因為 32 位元 Java runtime environment 無法載入 64 位元 Java 原生程式庫。
- IBM MQ classes for JMS 應用程式在 64 位元 Java runtime environment 中執行，並且已針對 IBM MQ classes for JMS 設定 32 位元環境，因為 64 位元 Java runtime environment 無法載入 32 位元 Java 原生程式庫。

IBM MQ classes for JMS/Jakarta Messaging 配置檔

IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 配置檔指定用來配置 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 的內容。

註: 配置檔中定義的內容也可以設為 JVM 系統內容。 如果在配置檔中同時將內容設定為系統內容，則會優先使用系統內容。 因此，必要的話，您可以置換配置檔中的任何內容，方法是將它指定為 **java** 指令上的系統內容。

IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 配置檔的格式是標準 Java 內容檔的格式。 IBM MQ classes for JMS 安裝目錄的 `bin` 子目錄中提供稱為 `jms.config` 的範例配置檔。 此檔案記載所有支援的內容及其預設值。

您可以選擇 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 配置檔的名稱和位置。 當您啟動應用程式時，請使用具有下列格式的 **java** 指令:

```
java -Dcom.ibm.msg.client.config.location= config_file_url application_name
```

在指令中， *config_file_url* 是統一資源定址器 (URL) ，指定 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 配置檔的名稱和位置。 支援下列類型的 URL :http、file、ftp 及 jar。

以下是 **java** 指令的範例:

```
java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/myjms.config MyAppClass
```

這個指令會將 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 配置檔識別為本端 Windows 系統上的 `D:\mydir\mjms.config` 檔案。

當應用程式啟動時， IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 會讀取配置檔的內容，並將指定的內容儲存在內部內容儲存庫中。 如果 **java** 指令未識別配置檔，或找不到配置檔，則 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 會對所有內容使用預設值。

IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 配置檔可以與應用程式與佇列管理程式或分配管理系統之間任何支援的傳輸搭配使用。

置換 IBM MQ MQI client 配置檔中指定的內容

IBM MQ MQI client 配置檔也可以指定用來配置 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 的內容。 不過，只有在應用程式以用戶端模式連接至佇列管理程式時， IBM MQ MQI client 配置檔中指定的內容才適用。

必要的話，您可以置換 IBM MQ MQI client 配置檔中的任何屬性，方法是將它指定為 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 配置檔中的內容。 若要置換 IBM MQ MQI client 配置檔中的屬性，請在 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 配置檔中使用具有下列格式的項目:

```
com.ibm.mq.cfg. stanza. propName = propValue
```

項目中的變數具有下列意義:

段落 (stanza)
IBM MQ MQI client 配置檔中包含屬性的段落名稱

propName
IBM MQ MQI client 配置檔中指定的屬性名稱

propValue
此內容的值會置換 IBM MQ MQI client 配置檔中指定的屬性值

或者，您可以在 **java** 指令上指定內容作為系統內容，以置換 IBM MQ MQI client 配置檔中的屬性。 請使用前述格式，將內容指定為系統內容。

IBM MQ MQI client 配置檔中只有下列屬性與 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 相關。 如果您指定或置換其他屬性，則它沒有作用。 具體而言，請注意，不會使用 用戶端配置檔的 CHANNELS 段落 中的 `ChannelDefinitionFile` 和 `ChannelDefinitionDirectory`。 如需如何搭配使用 CCDT 與 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 的詳細資料，請參閱 第 239 頁的『搭配使用用戶端通道定義表與 IBM MQ classes for JMS』。

表 *10:* 用戶端配置檔的哪個段落包含哪個屬性

段落 (stanza)	**屬性**
用戶端配置檔的 CHANNELS 段落	Put1DefaultAlwaysSync
用戶端配置檔的 CHANNELS 段落	DefRecon
用戶端配置檔的 CHANNELS 段落	ReconDelay
用戶端配置檔的 CHANNELS 段落	PasswordProtection
ClientExit 用戶端配置檔的路徑段落	ExitsDefaultPath
ClientExit 用戶端配置檔的路徑段落	ExitsDefaultPath64
ClientExit 用戶端配置檔的路徑段落	JavaExitsClasspath
用戶端配置檔的 JMQI 段落	useMQCSPauthentication
用戶端配置檔的 MessageBuffer 段落	MaximumSize
用戶端配置檔的 MessageBuffer 段落	PurgeTime
用戶端配置檔的 MessageBuffer 段落	UpdatePercentage
用戶端配置檔的 TCP 段落	ClntRcvBufSize
用戶端配置檔的 TCP 段落	ClntSndBufSize
用戶端配置檔的 TCP 段落	連線逾時
用戶端配置檔的 TCP 段落	KeepAlive

如需 IBM MQ MQI client 配置的進一步詳細資料，請參閱 IBM MQ MQI client 配置檔: `mqclient.ini`

使用 *Java* 標準環境追蹤來配置 *JMS* 追蹤

使用「Java 標準環境追蹤設定」段落來配置 IBM MQ classes for JMS 及 IBM MQ classes for Jakarta Messaging 追蹤機能。

com.ibm.msg.client.commonservices.trace.outputName = *traceOutput* 名稱

traceOutputName 是追蹤輸出傳送至其中的目錄及檔名。

依預設，追蹤資訊會寫入應用程式現行工作目錄中的追蹤檔。 追蹤檔的名稱視應用程式執行所在的環境而定:

- JM 3.0 V 9.3.0 V 9.3.0 從 IBM MQ 9.3.0 開始，如果應用程式已從可再定位 JAR 檔 `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0) 載入 IBM MQ classes for Jakarta Messaging，或從可再定位 JAR 檔 `com.ibm.mq.allclient.jar` (JMS 2.0) 載入 IBM MQ classes for JMS，則會將追蹤寫入稱為 `mqjavaclient_%PID%.cl%u.trc` 的檔案。
- 從 IBM MQ 9.1.5 和 IBM MQ 9.1.0 Fix Pack 5:
  - 如果應用程式已從可再定位 JAR 檔 `com.ibm.mq.allclient.jar` 載入 IBM MQ classes for JMS，則會將追蹤寫入稱為 `mqjavaclient_%PID%.cl%u.trc` 的檔案。
  - 如果應用程式已從 JAR 檔 `com.ibm.mqjms.jar` 載入 IBM MQ classes for JMS，追蹤會寫入稱為 `mqjava_%PID%.cl%u.trc` 的檔案。
- 從 IBM MQ 9.0.0 Fix Pack 2:
  - 如果應用程式已從可再定位 JAR 檔 `com.ibm.mq.allclient.jar` 載入 IBM MQ classes for JMS，則會將追蹤寫入稱為 `mqjavaclient_%PID%.trc` 的檔案。
  - 如果應用程式已從 JAR 檔 `com.ibm.mqjms.jar` 載入 IBM MQ classes for JMS，追蹤會寫入稱為 `mqjava_%PID%.trc` 的檔案。
- 若為 IBM MQ classes for JMS for IBM MQ 9.0.0 Fix Pack 1 或更早版本，追蹤會寫入稱為 `mqjms_%PID%.trc` 的檔案。

其中 *%PID%* 是所追蹤應用程式的處理程序 ID，而 *%u* 是唯一數字，用來區分在不同 Java 類別載入器下執行追蹤的執行緒之間的檔案。

如果程序 ID 無法使用，則會產生一個亂數，並以字母 `f` 作為字首。 若要在您指定的檔名中包括程序 ID，請使用字串 %PID%。

如果您指定替代目錄，則它必須存在，且您必須具有此目錄的寫入權。 如果您沒有寫入權，追蹤輸出會寫入 `System.err`。

com.ibm.msg.client.commonservices.trace.include = *includeList*

includeList 是所追蹤套件及類別的清單，或特殊值 `ALL` 或 `NONE`。

以分號 `;` 區隔套件或類別名稱。 *includeList* 預設為 `ALL`，並追蹤 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 中的所有套件和類別。

註: 您可以包括套件，然後排除該套件的子套件。 例如，如果您包括套件 `a.b` 及排除套件 `a.b.x`，則追蹤會包括 `a.b.y` 及 `a.b.z` 中的所有項目，但不會包括 `a.b.x` 或 `a.b.x.1`。

com.ibm.msg.client.commonservices.trace.exclude = *excludeList*

excludeList 是未追蹤的套件和類別清單，或特殊值 `ALL` 或 `NONE`。

以分號 `;` 區隔套件或類別名稱。 *excludeList* 預設為 `NONE`，因此不會追蹤 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 中的任何套件和類別。

註: 您可以排除套件，然後併入該套件的子套件。 比方說，如果您排除套件 `a.b` 並包含套件 `a.b.x`，則追蹤會包含 `a.b.x` 和 `a.b.x.1` 中的所有項目，但不會包含 `a.b.y` 或 `a.b.z`。

會併入在相同層次所指定的任何套件或類別 (已併入及已排除)。

com.ibm.msg.client.commonservices.trace.maxBytes = *maxArray* 位元組

maxArrayBytes 是從任何位元組陣列追蹤的位元組數上限。

如果 *maxArrayBytes* 設為正整數，它會限制位元組陣列中寫入追蹤檔的位元組數。 它會在寫出 *maxArrayBytes* 之後截斷位元組陣列。 設定 *maxArrayBytes* 會減少產生的追蹤檔大小，並減少追蹤對應用程式效能的影響。

此內容的值 0 表示任何位元組陣列的內容都不會傳送至追蹤檔。

預設值為 -1，它會移除對位元組陣列中傳送至追蹤檔的位元組數的任何限制。

com.ibm.msg.client.commonservices.trace.limit = *maxTrace* 位元組
maxTraceBytes 是寫入追蹤輸出檔的位元組數上限。

maxTraceBytes 與 *traceCycles* 搭配使用。 如果寫入的追蹤位元組數接近限制，則會關閉檔案，並啟動新的追蹤輸出檔。

值 0 表示追蹤輸出檔長度為零。 預設值為 -1，表示要寫入追蹤輸出檔的資料量無限制。

com.ibm.msg.client.commonservices.trace.count = *traceCycles*
traceCycles 是要輪流選取的追蹤輸出檔數目。

如果現行追蹤輸出檔達到 *maxTraceBytes* 指定的限制，則會關閉該檔案。 後續追蹤輸出會依序寫入至下一個追蹤輸出檔。 每一個追蹤輸出檔都以附加至檔名的數字字尾來識別。 現行或最新的追蹤輸出檔是 mqjms.trc.0，下一個最新的追蹤輸出檔是 mqjms.trc.1。 較舊的追蹤檔遵循相同的編號型樣，直到達到限制為止。

traceCycles 的預設值為 1。 如果 *traceCycles* 為 1，則當現行追蹤輸出檔達到其大小上限時，會關閉並刪除該檔案。 啟動同名的新追蹤輸出檔。 因此，一次只會有一個追蹤輸出檔。

com.ibm.msg.client.commonservices.trace.parameter = *traceParameters*
traceParameters 控制方法參數和回覆值是否包含在追蹤中。

traceParameters 預設為 TRUE。 如果 *traceParameters* 設為 FALSE，則只會追蹤方法簽章。

com.ibm.msg.client.commonservices.trace.startup = *startup*
IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 有一個起始設定階段，在此期間會配置資源。 在資源配置階段期間會起始設定主要追蹤機能。

如果 *startup* 設為 TRUE，則會使用啟動追蹤。 追蹤資訊會立即產生，並包括所有元件的設定，包括追蹤機能本身。 啟動追蹤資訊可用來診斷配置問題。 啟動追蹤資訊一律寫入 System.err。

startup 預設為 FALSE。

在起始設定完成之前，會先檢查 *startup* 。 因此，請只將指令行上的內容指定為 Java 系統內容。 請勿在 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 配置檔中指定它。

com.ibm.msg.client.commonservices.trace.compress = *compressedTrace*
將 *compressedTrace* 設為 TRUE ， 以壓縮追蹤輸出。

compressedTrace 的預設值為 FALSE。

如果 *compressedTrace* 設為 TRUE，則會壓縮追蹤輸出。 預設追蹤輸出檔名稱的副檔名為 .trz。 如果壓縮設為 FALSE(預設值) ，則檔案副檔名為 .trc ， 表示它未經壓縮。 不過，如果已在 *traceOutputName* 中指定追蹤輸出的檔名，則會改用該名稱; 不會將字尾套用至檔案。

壓縮追蹤輸出小於未壓縮。 因為 I/O 較少，所以可以比未壓縮追蹤更快地寫出它。 與未經壓縮的追蹤相比，壓縮追蹤對 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 效能的影響較小。

如果設定 *maxTraceBytes* 和 *traceCycles* ，則會建立多個壓縮追蹤檔來取代多個純文字檔。

如果 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 以不受控制的方式結束，則壓縮追蹤檔可能無效。 因此，只有在 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 以受控制的方式關閉時，才必須使用追蹤壓縮。 只有在所調查的問題不會導致 JVM 本身非預期地停止時，才使用追蹤壓縮。 在診斷可能導致 System.Halt() 關機或異常、不受控制的 JVM 終止的問題時，請勿使用追蹤壓縮。

com.ibm.msg.client.commonservices.trace.level = *traceLevel*
traceLevel 指定追蹤的過濾層次。 定義的追蹤層次如下:

- TRACE_NONE: 0
- TRACE_EXCEPTION: 1
- TRACE_WARNING: 3
- TRACE_INFO: 6

- TRACE_ENTRYEXIT: 8
- TRACE_DATA: 9
- TRACE_ALL: Integer.MAX_VALUE

每一個追蹤層次都包含所有較低層次。 比方說，如果在 TRACE_INFO 設定追蹤層次，則任何已定義層次為 TRACE_EXCEPTION、 TRACE_WARNING 或 TRACE_INFO 的追蹤點都會寫入追蹤。 會排除所有其他追蹤點。

com.ibm.msg.client.commonservices.trace.standalone = *standaloneTrace*

standaloneTrace 控制是否在 WebSphere Application Server 環境中使用 IBM MQ JMS 用戶端追蹤服務。

如果 *standaloneTrace* 設為 TRUE，則會使用 IBM MQ JMS 用戶端追蹤內容來決定追蹤配置。

如果 *standaloneTrace* 設為 FALSE，且 IBM MQ JMS 用戶端在 WebSphere Application Server 儲存器中執行，則會使用 WebSphere Application Server 追蹤服務。 產生的追蹤資訊取決於應用程式伺服器的追蹤設定。

standaloneTrace 的預設值為 FALSE。

記載段落
使用 Logging 段落來配置 IBM MQ classes for JMS 日誌機能。

下列內容可以包含在「記載」段落中:

com.ibm.msg.client.commonservices.log.outputName = path
IBM MQ classes for JMS 日誌機能使用的日誌檔名稱。 預設值為 mqjms.log，它會寫入 IBM MQ classes for JMS 執行所在之「Java 執行時期環境」的現行工作目錄中。

內容可以採用下列其中一個值:

- 單一路徑名稱
- 以逗點區隔的路徑名稱清單 (所有資料都會記載至所有檔案)

每一個路徑名稱可以是絕對或相對路徑名稱，或:

"stderr" 或 "System.err"
代表標準錯誤串流。

"stdout" 或 "System.out"
代表標準輸出串流。

com.ibm.msg.client.commonservices.log.maxBytes
從任何日誌訊息資料呼叫記載的位元組數上限。

正整數
每次日誌呼叫都會將資料寫入至該位元組值。

0
未寫入任何資料。

-1
寫入無限制資料 (預設值)。

com.ibm.msg.client.commonservices.log.limit
寫入任何 1 日誌檔的位元組數上限 (預設值為 262144)。

正整數
資料會寫入至每個日誌檔的該位元組值。

0
未寫入任何資料。

-1
寫入無限制資料。

com.ibm.msg.client.commonservices.log.count
要循環經歷的日誌檔數目。 當每一個檔案到達 com.ibm.msg.client.commonservices.trace.limit 追蹤時，將在下一個檔案中開始，預設值為 3。

正整數
要循環的檔案數。

0
單一檔案。

Java SE 特性段落
使用 Java SE Spec 細節段落來配置在 Java Standard Edition 環境中使用 IBM MQ classes for JMS 時所使用的內容。

com.ibm.msg.client.commonservices.j2se.produceJavaCore = TRUE|FALSE
決定是否在 IBM MQ classes for JMS 產生 FDC 檔案之後立即寫入 Java 核心檔案。 如果設為 TRUE，則會在 IBM MQ classes for JMS 執行所在之「Java 執行時期環境」的工作目錄中產生 Java 核心檔案。

TRUE
根據 Java 執行時期環境的能力來產生 JavaCore。

FALSE
不產生 JavaCore; 這是預設值。

IBM MQ 內容段落
使用 IBM MQ Properties 段落來設定內容，這些內容會影響 IBM MQ classes for JMS 與 IBM MQ 互動的方式。

com.ibm.msg.client.wmq.compat.base.internal.MQQueue.smallMsgsBufferReductionThreshold
當使用 IBM MQ classes for JMS 的應用程式使用 IBM MQ 傳訊提供者移轉模式連接至 IBM MQ 佇列管理程式時， IBM MQ classes for JMS 會在接收訊息時使用預設緩衝區大小 4 KB。 如果應用程式嘗試取得的訊息大於 4 KB，則 IBM MQ classes for JMS 會調整緩衝區大小，使其足以容納該訊息。 然後會在收到後續訊息時使用較大的緩衝區大小。

此內容控制何時將緩衝區大小縮減回 4 KB。 依預設，當收到 10 個連續小於較大緩衝區大小的訊息時，緩衝區大小會縮減回 4 KB。 若要在每次收到訊息時將緩衝區大小重設回 4 KB，請將內容設為值 0。

0

緩衝區一律重設為預設大小。

10

這是預設值。 將在第十則訊息之後調整緩衝區大小。

com.ibm.msg.client.wmq.receiveConversionCCSID
當使用 IBM MQ classes for JMS 的應用程式使用 IBM MQ 傳訊提供者標準模式連接至 IBM MQ 佇列管理程式時，可以設定 receiveConversionCCSID 內容來置換 MQMD 結構中用來從佇列管理程式接收訊息的預設 CCSID 值。 依預設， MQMD 包含設為 1208 的 CCSID 欄位，但如果例如佇列管理程式無法將訊息轉換為此字碼頁，則可以變更此欄位。

有效值為任何有效的 CCSID 號碼或下列其中一個值:

-1
使用平台預設值。

1208
這是預設值。

用戶端模式特性段落
使用「用戶端模式特性」段落來指定當「IBM MQ classes for JMS」連接至使用 CLIENT 傳輸的佇列管理程式時所使用的內容。

com.ibm.mq.polling.RemoteRequestEntry
指定當 IBM MQ classes for JMS 等待佇列管理程式的回應時，用來檢查是否有中斷連線的輪詢間隔。

正整數
在檢查之前等待的毫秒數。 預設值為 10000 或 10 秒。 下限值為 3000 ，而下限值的處理方式與此下限值相同。

用來配置 *JMS* 用戶端行為的內容
使用這些內容來配置 JMS 用戶端的行為。

com.ibm.mq.jms.SupportMQExtensions TRUE|FALSE
JMS 2.0 規格引進某些行為運作方式的變更。 IBM MQ 8.0 包含內容 `com.ibm.mq.jms.SupportMQExtensions`(可設為 *TRUE*) ，以將這些變更的行為回復為先前的實作。 回復已變更的行為對於 JMS 2.0 應用程式可能是必要的，對於使用 JMS 1.1 API 但針對 IBM MQ 8.0 IBM MQ classes for JMS 執行的部分應用程式也可能是必要的。

TRUE
將 SupportMQExtensions 設為 TRUE，可回復下列三個功能區域:

訊息優先順序
訊息可以指派優先順序 *0* - *9*。 在 JMS 2.0 之前，訊息也可以使用值 *-1*，指出使用佇列的預設優先順序。 JMS 2.0 不容許設定訊息優先順序 *-1* 。 開啟 `SupportMQExtensions` 容許使用 *-1* 的值。

用戶端 ID
JMS 2.0 規格要求在建立連線時檢查非空值用戶端 ID 的唯一性。 開啟 `SupportMQExtensions`，表示不處理此需求，且可以重複使用用戶端 ID。

NoLocal
JMS 2.0 規格要求在開啟此常數時，消費者無法接收由相同用戶端 ID 發佈的訊息。 在 JMS 2.0 之前，已在訂閱者上設定此屬性，以防止它接收由自己的連線發佈的訊息。 開啟 `SupportMQExtensions` 會將此行為回復為先前的實作。

FALSE
會保留行為的變更。

com.ibm.msg.client.jms.ByteStreamReadOnlyAfterSend= TRUE|FALSE
從 IBM MQ 8.0.0 Fix Pack 2 開始，在應用程式傳送「位元組」或「串流」訊息之後， IBM MQ classes for JMS 可以將剛傳送的訊息狀態設為唯讀或唯寫。

TRUE
物件在傳送之後會設為唯讀。 設定此值可維護與 JMS 2.0 規格的相容性

FALSE
物件設定為僅在傳送之後寫入。 這是預設值。

相關概念
第 274 頁的『SupportMQExtensions 內容』
JMS 2.0 規格引進了某些行為運作方式的變更。 IBM MQ 8.0 以及更新版本包括內容 **com.ibm.mq.jms.SupportMQExtensions**，可以設定為 TRUE 以將這些變更的行為回復為先前的實作。

z/OS 上 *IBM MQ classes for JMS* 的 *STEPLIB* 配置
在 z/OS 上，執行時期使用的 STEPLIB 必須包含 IBM MQ SCSQAUTH 和 SCSQANLE 程式庫。 在啟動 JCL 中或使用 `.profile` 檔案來指定這些程式庫。

從 z/OS UNIX System Services 開始，您可以使用 `.profile` 中的一行來新增這些，如下列程式碼 Snippet 所示，將 `thlqual` 取代為您在安裝 IBM MQ 時選擇的高階資料集限定元:

```
export STEPLIB=thlqual.SCSQAUTH:thlqual.SCSQANLE:$STEPLIB
```

在其他環境中，您通常需要編輯啟動 JCL ，以在 STEPLIB 連結上包含 SCSQAUTH 和 SCSQANLE:

```
STEPLIB DD DSN=thlqual.SCSQAUTH,DISP=SHR
        DD DSN=thlqual.SCSQANLE,DISP=SHR
```

IBM MQ classes for JMS 和軟體管理工具

軟體管理工具 (例如 Apache Maven) 可以與 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 搭配使用。

許多大型開發組織使用這些工具來集中管理協力廠商程式庫的儲存庫。

IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 由一些 JAR 檔組成。 當您使用此 API 來開發 Java 語言應用程式時，需要在開發應用程式的機器上安裝 IBM MQ Server、 IBM MQ Client 或 IBM MQ Client SupportPac 。

如果您想要使用這類工具，並將組成 IBM MQ classes for JMS 的 JAR 檔新增至集中管理的儲存庫，則必須觀察下列要點:

- 儲存庫或儲存器只能供組織內的開發人員使用。 不允許在組織外部進行任何配送。
- 儲存庫需要包含一組來自單一 IBM MQ 版本或修正套件的完整且一致的 JAR 檔。
- 您負責以「IBM 支援中心」提供的任何維護來更新儲存庫。

下列 JAR 檔需要安裝至儲存庫:

- JMS 2.0 如果您使用 IBM MQ classes for JMS，則需要 `com.ibm.mq.allclient.jar` 和 `jms.jar` 。
- JM 3.0 如果您使用 IBM MQ classes for Jakarta Messaging，則需要 `com.ibm.mq.jakarta.client.jar` 和 `jakarta.jms-api.jar` 。
- 如果您使用 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging ，且正在存取儲存在檔案系統 JNDI 環境定義中的 JMS 受管理物件，則需要 `fscontext.jar` 。
- 如果您使用 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging ，且正在存取儲存在檔案系統 JNDI 環境定義中的 JMS 受管理物件，則需要 `providerutil.jar` 。
- 需要 Bouncy Castle 安全提供者及 CMS 支援 JAR 檔，才能支援非 IBM JRE。 如需相關資訊，請參閱 支援非 IBM JRE。

在 *Java security manager* 下執行 *IBM MQ classes for JMS* 應用程式

IBM MQ classes for JMS 可以在啟用 Java security manager 的情況下執行。 若要在啟用 Java security manager 的情況下順利執行應用程式，您必須使用適當的原則配置檔來配置 Java Virtual Machine (JVM)。

建立適合原則定義檔的最簡單方法是變更 Java runtime environment (JRE) 所提供的原則配置檔。 在大部分系統上，此檔案位於相對於 JRE 目錄的 `lib/security/java.policy` 目錄中。 您可以使用偏好的編輯器，或使用 JRE 隨附的原則工具程式，來編輯原則配置檔。

原則配置檔範例

以下是容許 IBM MQ classes for JMS 在預設安全管理程式下順利執行的原則配置檔範例。 需要自訂此檔案，以指定特定檔案及目錄的位置: *MQ_INSTALLATION_PATH* 代表 IBM MQ 安裝所在的高階目錄，*MQ_DATA_DIRECTORY* 代表 MQ 資料目錄的位置， *QM_NAME* 是為其配置存取權的佇列管理程式名稱。

```
grant codeBase "file:MQ_INSTALLATION_PATH/java/lib/*" {
  //We need access to these properties, mainly for tracing
  permission java.util.PropertyPermission "user.name","read";
  permission java.util.PropertyPermission "os.name","read";
  permission java.util.PropertyPermission "user.dir","read";
  permission java.util.PropertyPermission "line.separator","read";
  permission java.util.PropertyPermission "path.separator","read";
  permission java.util.PropertyPermission "file.separator","read";
  permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.log.*","read";
  permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.trace.*","read";
  permission java.util.PropertyPermission "Diagnostics.Java.Errors.Destination.Filename","read";
  permission java.util.PropertyPermission "com.ibm.mq.commonservices","read";
  permission java.util.PropertyPermission "com.ibm.mq.cfg.*","read";

  //Tracing - we need the ability to control java.util.logging
  permission java.util.logging.LoggingPermission "control";
  // And access to create the trace file and read the log file - assumed to be in the current
directory
  permission java.io.FilePermission "*","read,write";
```

```
   // We'd like to set up an mBean to control trace
   permission javax.management.MBeanServerPermission "createMBeanServer";
   permission javax.management.MBeanPermission "*","*";

   // We need to be able to read manifests etc from the jar files in the installation directory
   permission java.io.FilePermission "MQ_INSTALLATION_PATH/java/lib/-","read";

   //Required if mqclient.ini/mqs.ini configuration files are used
   permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqclient.ini","read";
   permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqs.ini","read";

   //For the client transport type.
   permission java.net.SocketPermission "*","connect,resolve";

   //For the bindings transport type.
   permission java.lang.RuntimePermission "loadLibrary.*";

   //For applications that use CCDT tables (access to the CCDT AMQCLCHL.TAB)
   permission java.io.FilePermission "MQ_DATA_DIRECTORY/qmgrs/QM_NAME/@ipcc/AMQCLCHL.TAB","read";

   //For applications that use User Exits
   permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits/*","read";
   permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits64/*","read";
   permission java.lang.RuntimePermission "createClassLoader";

   //Required for the z/OS platform
   permission java.util.PropertyPermission "com.ibm.vm.bitmode","read";

   // Used by the internal ConnectionFactory implementation
   permission java.lang.reflect.ReflectPermission "suppressAccessChecks";

   // Used for controlled class loading
   permission java.lang.RuntimePermission "setContextClassLoader";

   // Used to default the Application name in Client mode connections
   permission java.util.PropertyPermission "sun.java.command","read";

   // Used by the IBM JSSE classes
   permission java.util.PropertyPermission "com.ibm.crypto.provider.AESNITrace","read";

   //Required to determine if an IBM Java Runtime is running in FIPS mode,
   //and to modify the property values status as required.
   permission java.util.PropertyPermission "com.ibm.jsse2.usefipsprovider","read,write";
   permission java.util.PropertyPermission "com.ibm.jsse2.JSSEFIPS","read,write";
   //Required if an IBM FIPS provider is to be used for SSL communication.
   permission java.security.SecurityPermission "insertProvider.IBMJCEFIPS";

   // Required for non-IBM Java Runtimes that establish secure client
   // transport mode connections using mutual TLS authentication
   permission java.util.PropertyPermission "javax.net.ssl.keyStore","read";
   permission java.util.PropertyPermission "javax.net.ssl.keyStorePassword","read";
};
```

在此範例中， `grant` 陳述式包含 IBM MQ classes for JMS 所需的許可權。 若要在原則配置檔中使用這些授權陳述式，您可能需要根據安裝 IBM MQ classes for JMS 的位置及儲存應用程式的位置來修改路徑名稱。

IBM MQ classes for JMS 隨附的範例應用程式以及用來執行它們的 Script ，不會啟用安全管理程式。

重要:

IBM MQ classes for JMS 追蹤機能需要進一步的許可權，因為它會執行系統內容的其他查詢，以及進一步的檔案系統作業。

在 IBM MQ 安裝架構的 `samples/wmqjava` 目錄中，提供了在啟用追蹤的安全管理程式下執行的適當範本安全原則檔 `example.security.policy`。

IBM MQ classes for JMS 應用程式的後置安裝設定

本主題告訴您 IBM MQ classes for JMS 應用程式存取佇列管理程式資源所需的權限。 它也引進連線模式，並說明如何配置佇列管理程式，讓應用程式可以在用戶端模式下連接。

請記得檢查 IBM MQ Readme 檔。 它可能包含取代本主題中資訊的資訊。

JMS 所使用且需要非特許使用者授權的物件

非特許使用者需要授權才能存取 JMS 所使用的佇列。 每個 JMS 應用程式都需要其使用之佇列管理程式的授權。

如需 IBM MQ 中存取控制的詳細資料，請參閱 設定安全。

IBM MQ classes for JMS 應用程式需要佇列管理程式的 **連接** 及 inq 權限。 您可以使用 **setmqaut** 控制指令來設定適當的授權，例如:

```
setmqaut -m QM1 -t qmgr -g jmsappsgroup +connect +inq
```

對於點對點網域，需要下列權限:

- MessageProducer 物件使用的佇列需要 put 權限。
- MessageConsumer 和 QueueBrowser 物件使用的佇列需要 get、 inq 和 **瀏覽** 權限。
- QueueSession.createTemporaryQueue () 方法需要存取 QueueConnectionFactory 物件的 TEMPMODEL 內容所指定的模型佇列。 依預設，此模型佇列為 SYSTEM.TEMP.MODEL.QUEUE。

如果其中任何佇列是別名佇列，則其目標佇列需要查詢權限。 如果目標佇列是叢集佇列，它也需要瀏覽權限。

對於發佈/訂閱網域，如果 IBM MQ classes for JMS 以 IBM MQ 傳訊提供者移轉模式連接至 IBM MQ 佇列管理程式，則會使用下列佇列:

- 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

如需 IBM MQ 傳訊提供者移轉模式的進一步資訊，請參閱 配置 JMS **PROVIDERVERSION** 內容

此外，如果「 IBM MQ classes for JMS 」以此模式連接至佇列管理程式，則任何發佈訊息的應用程式都需要存取 TopicConnectionFactory 或主題物件所指定的串流佇列。 依預設，此佇列為 SYSTEM.BROKER.DEFAULT.STREAM。

如果您使用 ConnectionConsumer、 IBM MQ Resource Adapter 或 WebSphere Application Server IBM MQ 傳訊提供者，則可能需要其他授權。

要由 ConnectionConsumer 讀取的佇列必須具有 get、 inq 及 **瀏覽** 權限。 系統無法傳送郵件的佇列，以及 ConnectionConsumer 所使用的任何重新放回佇列或報告佇列必須具有 put 和 passall 權限。

當應用程式使用 IBM MQ 傳訊提供者標準模式來執行發佈/訂閱傳訊時，應用程式會利用佇列管理程式所提供的整合發佈/訂閱功能。 如需保護所使用主題及佇列安全的相關資訊，請參閱 發佈/訂閱安全 。

IBM MQ classes for JMS 的連線模式

IBM MQ classes for JMS 應用程式可以在用戶端或連結模式下連接至佇列管理程式。 在用戶端模式中， IBM MQ classes for JMS 會透過 TCP/IP 連接至佇列管理程式。 在連結模式中， IBM MQ classes for JMS 會使用「 Java 原生介面 (JNI)」直接連接至佇列管理程式。

在 z/OS 上的 WebSphere Application Server 中執行的應用程式可以在連結或用戶端模式下連接至佇列管理程式，但在 z/OS 上的任何其他環境中執行的應用程式只能在連結模式下連接至佇列管理程式。 在任何其他平台上執行的應用程式可以在連結或用戶端模式下連接至佇列管理程式。

您可以將現行或任何舊版受支援的 IBM MQ classes for JMS 與現行佇列管理程式搭配使用，也可以將現行或舊版受支援的佇列管理程式與現行版本的 IBM MQ classes for JMS 搭配使用。 如果您混合不同的版本，則功能會限制為舊版的層次。

下列各節更詳細地說明每一種連線模式。

CLIENT 模式

若要以用戶端模式連接至佇列管理程式， IBM MQ classes for JMS 應用程式可以在執行佇列管理程式的相同系統上執行，也可以在不同的系統上執行。 在每一個情況下，「IBM MQ classes for JMS」都會透過 TCP/IP 連接至佇列管理程式。

BINDINGS 模式

若要以連結模式連接至佇列管理程式， IBM MQ classes for JMS 應用程式必須在佇列管理程式執行所在的相同系統上執行。

「IBM MQ classes for JMS」會使用「Java 原生介面 (JNI)」直接連接至佇列管理程式。 如果要使用連結傳輸， IBM MQ classes for JMS 必須在可存取 IBM MQ Java 原生介面程式庫的環境中執行; 如需進一步資訊，請參閱 第 81 頁的『配置 Java 原生介面 (JNI) 程式庫』 。

IBM MQ classes for JMS 支援 *ConnectOption* 的下列值:

- MQCNO_FASTPATH_BINDING
- MQCN_STANDARD_BINDING
- MQCNO_SHARED_BINDING
- MQCNO_ISOLATED_BINDING
- MQCNO_RESTRICT_CONN_TAG_QSG
- MQCNO_RESTRICT_CONN_TAG_Q_MGR

若要變更 IBM MQ classes for JMS 所使用的連線選項，請修改 Connection Factory 內容 CONNOPT。

如需連線選項的進一步資訊，請參閱 第 618 頁的『使用 MQCONNX 呼叫連接至佇列管理程式』

如果要使用連結傳輸，所使用的「Java 執行時期環境」必須支援 IBM MQ classes for JMS 所連接之佇列管理程式的「編碼字集 ID (CCSID)」。

如需如何判斷「Java 執行時期環境」支援哪些 CCSID 的詳細資料，請參閱 IBM MQ 使用適用於 Java 的 IBM MQ V7 類別或適用於 JMS 的 IBM MQ V7 類別時所產生具有探測 ID 21 的 FDC。

配置佇列管理程式，讓 *IBM MQ classes for JMS* 應用程式可以在用戶端模式下連接
若要配置佇列管理程式，讓 IBM MQ classes for JMS 應用程式可以在用戶端模式下連接，您必須建立伺服器連線通道定義並啟動接聽器。

建立伺服器連線通道定義

在所有平台上，您可以使用 MQSC 指令 DEFINE CHANNEL 來建立伺服器連線通道定義。 請參閱下列範例:

```
DEFINE CHANNEL(JAVA.CHANNEL) CHLTYPE(SVRCONN) TRPTYPE(TCP)
```

IBM i 在 IBM i 上，您可以改用 CL 指令 CRTMQMCHL ，如下列範例所示:

```
CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN)
TRPTYPE(*TCP)
MQMNAME(QMGRNAME)
```

在此指令中， *QMGRNAME* 是佇列管理程式的名稱。

Windows Linux 在 Linux 和 Windows 上，您也可以使用 IBM MQ Explorer 來建立伺服器連線通道定義。

z/OS 在 z/OS 上，您可以使用作業及控制台來建立伺服器連線通道定義。

通道 (JAVA.CHANNEL 必須與應用程式用來連接至佇列管理程式之 Connection Factory 的 CHANNEL 內容所指定的通道名稱相同。 CHANNEL 內容的預設值是 SYSTEM.DEF.SVRCONN。

啟動接聽器

您必須啟動佇列管理程式的接聽器 (如果尚未啟動的話)。

Multi 在 多平台上，您可以在第一次使用 MQSC 指令 DEFINE LISTENER 來建立接聽器物件之後，使用 MQSC 指令 START LISTENER 來啟動接聽器，如下列範例所示:

```
DEFINE LISTENER(LISTENER.TCP) TRPTYPE(TCP) PORT(1414)
START LISTENER(LISTENER.TCP)
```

z/OS 在 z/OS 上，您只能使用 START LISTENER 指令，如下列範例所示，但請注意，必須先啟動通道起始程式位址空間，才能啟動接聽器:

```
START LISTENER TRPTYPE(TCP) PORT(1414)
```

IBM i 在 IBM i 上，您也可以使用 CL 指令 STRMQMLSR 來啟動接聽器，如下列範例中所示:

```
STRMQMLSR PORT(1414) MQMNAME(QMGRNAME)
```

在此指令中， *QMGRNAME* 是佇列管理程式的名稱。

ALW 在 AIX, Linux, and Windows 上，您也可以使用控制指令 **runmqlsr** 來啟動接聽器，如下列範例中所示:

```
runmqlsr -t tcp -p 1414 -m QMgrName
```

在此指令中， *QMgrName* 是佇列管理程式的名稱。

Windows Linux 在 Linux 和 Windows 上，您也可以使用 IBM MQ Explorer 來啟動接聽器。

z/OS 在 z/OS 上，您也可以使用作業及控制台來啟動接聽器。

接聽器接聽所在的埠號必須與應用程式用來連接至佇列管理程式之 Connection Factory 的 PORT 內容所指定的埠號相同。 PORT 內容的預設值為 1414。

IBM MQ classes for JMS 的點對點 *IVT*

IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 隨附點對點安裝驗證測試 (IVT) 程式。程式會以連結或用戶端模式連接至佇列管理程式，並將訊息傳送至稱為 SYSTEM.DEFAULT.LOCAL.QUEUE，然後從佇列接收訊息。 程式可以在執行時期動態建立及配置它所需要的所有物件，也可以使用 JNDI 從目錄服務擷取受管理物件。

在不先使用 JNDI 的情況下執行安裝驗證測試，因為測試自行包含且不需要使用目錄服務。 如需受管理物件的說明，請參閱 使用管理工具來配置 JMS 物件。

不使用 JNDI 的點對點安裝驗證測試

在此測試中， IVT 程式會建立並配置在執行時期動態需要且不使用 JNDI 的所有物件。

Multi 在多平台上，會提供 Script 來執行 IVT 程式。 此 Script 在 AIX and Linux 系統上稱為 **IVTRun** ，在 Windows 上稱為 **IVTRun.bat** 。 Script 位於 IBM MQ classes for JMS 安裝目錄的 `bin` 子目錄中。 類別路徑必須包含 `com.ibm.mqjms.jar`。

如果要在連結模式下執行測試，請輸入下列指令:

```
IVTRun -nojndi [-m qmgr ] [-v providerVersion ] [-t]
```

如果要以用戶端模式執行測試，請先依照 第 897 頁的『配置佇列管理程式以接受 Multiplatforms 上的用戶端連線』中的說明來設定佇列管理程式。 請注意，要使用的通道預設為 **SYSTEM.DEF.SVRCONN**，而要使用的佇列是 **SYSTEM.DEFAULT.LOCAL.QUEUE**，然後輸入下列指令:

```
IVTRun -nojndi -client -m qmgr -host hostname [-port port ] [-channel channel ]
[-v providerVersion ] [-ccsid ccsid ] [-t]
```

z/OS 在 z/OS 系統上未提供對等 Script。 相反地，您可以直接呼叫 Java 類別，以連結模式執行 IVT。 在 z/OS 上，您可以在 IVT 程式的兩個功能相同的實例之間進行選擇:

- `com.ibm.mq.jms.MQJMSIVT`，隨附於 IBM MQ classes for JMS (JMS 2.0)。 如果要使用這個程式，類別路徑必須包含 `com.ibm.mqjms.jar` 或 `com.ibm.mq.allclient.jar`。
- `com.ibm.mq.jakarta.jms.MQJMSIVT`，隨附於 IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0)。 如果要使用這個程式，類別路徑必須包含 `com.ibm.mq.jakarta.client.jar`。

如果要在 z/OS 上以連結模式執行測試，請輸入下列指令:

```
java com.ibm.mq.jms.MQJMSIVT -nojndi [-m qmgr ] [-v providerVersion ] [-t]
```

指令上的參數具有下列意義:

-m *qmgr*
IVT 程式所連接的佇列管理程式名稱。 如果您以連結模式執行測試，並省略此參數，則 IVT 程式會連接至預設佇列管理程式。

-host *hostname*
執行佇列管理程式之系統的主機名稱或 IP 位址。

-port *port*
佇列管理程式的接聽器接聽所在的埠號。 預設值為 1414。

-channel *channel*
IVT 程式用來連接佇列管理程式的 MQI 通道名稱。 預設值為 `SYSTEM.DEF.SVRCONN`。

-v *providerVersion*
IVT 程式預期連接之佇列管理程式的版次。

此參數用來設定 MQQueueConnectionFactory 物件的 PROVIDERVERSION 內容，其有效值與 PROVIDERVERSION 內容的有效值相同。 因此，如需此參數的相關資訊 (包括其有效值)，請參閱 JMS: 對 PROVIDERVERSION 內容的變更，以及 IBM MQ classes for JMS 物件的內容中 PROVIDERVERSION 內容的說明。

預設值為 `unspecified`。

-ccsid *ccsid*
連線要使用之編碼字集或字碼頁的 ID (CCSID)。 預設值為 819。

-t
已啟用追蹤。 依預設，會停用追蹤。

成功測試會產生類似下列範例輸出的輸出:

```
5724-H72, 5655-R36, 5724-L26, 5655-L82 (c) Copyright IBM Corp. 2008, 2024. All
Rights Reserved.
WebSphere MQ classes for Java(tm) Message Service 7.0
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
JMSMessage class: jms_text
JMSType:     null
JMSDeliveryMode: 2
JMSExpiration:  0
JMSPriority:   4
JMSMessageID:   ID:414d5120514d5f6d627720202020202001edb14620005e03
JMSTimestamp:   1187170264000
JMSCorrelationID: null
JMSDestination:  queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo:    null
JMSRedelivered:  false
JMSXUserID: mwhite
JMS_IBM_Encoding: 273
JMS_IBM_PutApplType: 28
JMSXAppID: IBM MQ Client for Java
JMSXDeliveryCount: 1
JMS_IBM_PutDate: 20070815
JMS_IBM_PutTime: 09310400
JMS_IBM_Format: MQSTR
JMS_IBM_MsgType: 8
A simple text message from the MQJMSIVT
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
```

使用 JNDI 的點對點安裝驗證測試

在此測試中， IVT 程式會使用 JNDI 從目錄服務擷取受管理物件。

您必須先配置以輕量型目錄存取通訊協定 (LDAP) 伺服器或本端檔案系統為基礎的目錄服務，才能執行測試。 您也必須配置 IBM MQ JMS 管理工具，讓它可以使用目錄服務來儲存受管理物件。 如需這些必要條件的相關資訊，請參閱 第 73 頁的『IBM MQ classes for JMS 的必備項目』。 如需如何配置 IBM MQ JMS 管理工具的相關資訊，請參閱 配置 JMS 管理工具。

IVT 程式必須能夠使用 JNDI 從目錄服務中擷取 MQQueueConnectionFactory 物件和 MQQueue 物件。 提供 Script 為您建立這些受管理物件。 該 Script 在 AIX and Linux 系統上稱為 IVTSetup ，在 Windows 上稱為 IVTSetup.bat ，位於 IBM MQ classes for JMS 安裝目錄的 bin 子目錄中。 若要執行 Script ，請輸入下列指令:

```
IVTSetup
```

該 Script 會呼叫 IBM MQ JMS 管理工具來建立受管理物件。

MQQueueConnectionFactory 物件會以名稱 ivtQCF 連結，並以其所有內容的預設值建立，這表示 IVT 程式會以連結模式執行，並連接至預設佇列管理程式。 如果您想要 IVT 程式以用戶端模式執行，或連接至預設佇列管理程式以外的佇列管理程式，則必須使用 IBM MQ JMS 管理工具或 IBM MQ Explorer 來變更 MQQueueConnectionFactory 物件的適當內容。 如需如何使用 IBM MQ Explorer JMS 管理工具的相關資訊，請參閱 使用管理工具來配置 JMS 物件。 如需如何使用 IBM MQ Explorer 的相關資訊，請參閱 IBM MQ Explorer 簡介 或 IBM MQ Explorer 提供的說明。

MQQueue 物件會以名稱 ivtQ 連結，並以其所有內容的預設值建立，但 QUEUE 內容除外，其值為 SYSTEM.DEFAULT.LOCAL.QUEUE。

建立受管理物件之後，您可以執行 IVT 程式。 如果要使用 JNDI 來執行測試，請輸入下列指令:

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

指令上的參數具有下列意義:

-url "*providerURL*"
目錄服務的統一資源定址器 (URL)。 URL 可以具有下列其中一種格式:

- `ldap://`*`hostname`*`/`*`contextName`* ，適用於以 LDAP 伺服器為基礎的目錄服務
- `file:/`*`directoryPath`* ，適用於基於本端檔案系統的目錄服務

請注意，您必須用引號 (") 括住 URL 。

-icf *initCtx* 事實
起始環境定義 Factory 的類別名稱，必須是下列其中一個值:

- `com.sun.jndi.ldap.LdapCtxFactory`，適用於以 LDAP 伺服器為基礎的目錄服務。 這是預設值。
- `com.sun.jndi.fscontext.RefFSContextFactory`，適用於以本端檔案系統為基礎的目錄服務。

-t
已啟用追蹤。 依預設，會停用追蹤。

成功測試會產生類似於成功測試而不使用 JNDI 的輸出。 主要差異是輸出指出測試正在使用 JNDI 來擷取 MQQueueConnectionFactory 物件和 MQQueue 物件。

雖然並非嚴格必要，但最好在測試之後，刪除 IVTSetup Script 所建立的受管理物件來進行清理。 為此目的提供了 Script。 此 Script 在 AIX and Linux 系統上稱為 IVTTidy ， 在 Windows 上稱為 IVTTidy.bat ， 位於 IBM MQ classes for JMS 安裝目錄的 `bin` 子目錄中。

點對點安裝驗證測試的問題判斷

安裝驗證測試可能因下列原因而失敗:

- 如果 IVT 程式寫入訊息指出找不到類別，請檢查是否已正確設定類別路徑，如 第 78 頁的『設定 IBM MQ classes for JMS/Jakarta Messaging 的環境變數』中所述。
- 測試可能會失敗，並出現下列訊息:

```
Failed to connect to queue manager ' qmgr ' with connection mode ' connMode '
and host name ' hostname '
```

以及相關聯的原因碼 2059。 訊息中的變數具有下列意義:

qmgr
IVT 程式嘗試連接的佇列管理程式名稱。 如果 IVT 程式嘗試以連結模式連接至預設佇列管理程式，則此訊息插入為空白。

connMode
連線模式，即 `Bindings` 或 `Client`。

hostname
執行佇列管理程式之系統的主機名稱或 IP 位址。

此訊息表示 IVT 程式嘗試連接的佇列管理程式無法使用。 請檢查佇列管理程式是否在執行中，如果 IVT 程式正在嘗試連接至預設佇列管理程式，請確定佇列管理程式定義為系統的預設佇列管理程式。

- 測試可能會失敗，並出現下列訊息:

```
Failed to open MQ queue 'SYSTEM.DEFAULT.LOCAL.QUEUE'
```

此訊息表示佇列 SYSTEM.DEFAULT.LOCAL.QUEUE 不存在於 IVT 程式所連接的佇列管理程式上。 或者，如果佇列確實存在，則 IVT 程式無法開啟佇列，因為它未啟用放置及取得訊息。 請檢查佇列是否存在，以及是否已啟用以放置及取得訊息。

- 測試可能會失敗，並出現下列訊息:

```
Unable to bind to object
```

此訊息表示有與 LDAP 伺服器的連線，但 LDAP 伺服器未正確配置。 未配置 LDAP 伺服器來儲存 Java 物件，或對物件或字尾的許可權不正確。 如需此狀況的相關說明，請參閱 LDAP 伺服器的說明文件。

- 測試可能會失敗，並出現下列訊息:

```
The security authentication was not valid that was supplied for
QueueManager ' qmgr ' with connection mode 'Client' and host name ' hostname '
```

此訊息表示佇列管理程式未正確設定接受來自系統的用戶端連線。 如需詳細資料，請參閱 第 897 頁的『配置佇列管理程式以接受 Multiplatforms 上的用戶端連線』。

IBM MQ classes for JMS 的發佈/訂閱 *IVT*

IBM MQ classes for JMS 隨附發佈/訂閱安裝驗證測試 (IVT) 程式。 程式會以連結或用戶端模式連接至佇列管理程式，訂閱主題，發佈關於主題的訊息，然後接收它剛剛發佈的訊息。 程式可以在執行時期動態建立及配置它所需要的所有物件，也可以使用 JNDI 從目錄服務擷取受管理物件。

在不先使用 JNDI 的情況下執行安裝驗證測試，因為測試自行包含且不需要使用目錄服務。 如需受管理物件的說明，請參閱 使用管理工具來配置 JMS 物件。

不使用 JNDI 的發佈/訂閱安裝驗證測試

在此測試中， IVT 程式會建立並配置在執行時期動態需要且不使用 JNDI 的所有物件。

提供 Script 來執行 IVT 程式。 此 Script 在 AIX and Linux 系統上稱為 PSIVTRun ， 在 Windows 上稱為 PSIVTRun.bat ， 位於 IBM MQ classes for JMS 安裝目錄的 `bin` 子目錄中。

如果要在連結模式下執行測試，請輸入下列指令:

```
PSIVTRun -nojndi [-m qmgr ] [-bqm brokerQmgr ] [-v providerVersion ] [-t]
```

如果要以用戶端模式執行測試，請先依照 第 897 頁的『配置佇列管理程式以接受 Multiplatforms 上的用戶端連線』 中的說明來設定佇列管理程式，並指出要使用的通道預設為 SYSTEM.DEF.SVRCONN，然後輸入下列指令:

```
PSIVTRun -nojndi -client -m qmgr -host hostname [-port port ] [-channel channel ]
[-bqm brokerQmgr ] [-v providerVersion ] [-ccsid ccsid ] [-t]
```

指令上的參數具有下列意義:

-m *qmgr*
IVT 程式所連接的佇列管理程式名稱。 如果您以連結模式執行測試，並省略此參數，則 IVT 程式會連接至預設佇列管理程式。

-host *hostname*
執行佇列管理程式之系統的主機名稱或 IP 位址。

-port *port*
佇列管理程式的接聽器接聽所在的埠號。 預設值為 1414。

-channel *channel*
IVT 程式用來連接佇列管理程式的 MQI 通道名稱。 預設值為 SYSTEM.DEF.SVRCONN。

-bqm *brokerQmgr*
分配管理系統執行所在的佇列管理程式名稱。 預設值是 IVT 程式所連接的佇列管理程式名稱。

此參數與佇列管理程式版本號碼 v 7 或以上無關。

-v *providerVersion*
IVT 程式預期連接之佇列管理程式的版次。

此參數用來設定 MQTopicConnectionFactory 物件的 PROVIDERVERSION 內容，其有效值與 PROVIDERVERSION 內容的有效值相同。 因此，如需此參數的相關資訊 (包括其有效值)，請參閱 IBM MQ classes for JMS 物件的內容中 PROVIDERVERSION 內容的說明。

預設值為 unspecified。

-ccsid *ccsid*
連線要使用之編碼字集或字碼頁的 ID (CCSID)。 預設值為 819。

-t
已啟用追蹤。 依預設，會停用追蹤。

成功測試會產生類似下列範例輸出的輸出:

```
5724-H72, 5655-R36, 5724-L26, 5655-L82 (c) Copyright IBM Corp. 2008, 2024. All
Rights Reserved.
IBM MQ classes for Java Message Service 7.0
Publish/Subscribe Installation Verification Test

Creating a TopicConnectionFactory
Creating a Connection
Creating a Session
Creating a Topic
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 [5 secs max]...

Got message:
JMSMessage class: jms_text
JMSType:     null
JMSDeliveryMode: 2
JMSExpiration:  0
JMSPriority:   4
JMSMessageID:   ID:414d5120514d5f6d627720202020202001edb14620006706
JMSTimestamp:   1187182520203
JMSCorrelationID: ID:414d5120514d5f6d627720202020202001edb14620006704
JMSDestination:  topic://MQJMS/PSIVT/Information
JMSReplyTo:    null
JMSRedelivered:  false
JMSXUserID: mwhite
JMS_IBM_Encoding: 273
JMS_IBM_PutApplType: 26
JMSXAppID: QM_mbw
JMSXDeliveryCount: 1
JMS_IBM_PutDate: 20070815
JMS_IBM_ConnectionID: 414D5143514D5F6D627720202020202001EDB14620006601
JMS_IBM_PutTime: 12552020
JMS_IBM_Format: MQSTR
JMS_IBM_MsgType: 8
A simple text message from the MQJMSPSIVT program
Reply string equals original string
Closing TopicSubscriber
Closing TopicPublisher
Closing Session
Closing Connection
PSIVT finished
```

使用 JNDI 的發佈/訂閱安裝驗證測試

在此測試中， IVT 程式會使用 JNDI 從目錄服務擷取受管理物件。

您必須先配置以輕量型目錄存取通訊協定 (LDAP) 伺服器或本端檔案系統為基礎的目錄服務，才能執行測試。 您也必須配置 IBM MQ JMS 管理工具，讓它可以使用目錄服務來儲存受管理物件。 如需這些必要條件的相關資訊，請參閱 第 73 頁的『IBM MQ classes for JMS 的必備項目』。 如需如何配置 IBM MQ JMS 管理工具的相關資訊，請參閱 配置 JMS 管理工具。

IVT 程式必須能夠使用 JNDI 從目錄服務中擷取 MQTopicConnectionFactory 物件和 MQTopic 物件。 提供 Script 為您建立這些受管理物件。 該 Script 在 AIX and Linux 系統上稱為 IVTSetup，在 Windows 上稱為 IVTSetup.bat，位於 IBM MQ classes for JMS 安裝目錄的 bin 子目錄中。 若要執行 Script，請輸入下列指令:

```
IVTSetup
```

該 Script 會呼叫 IBM MQ JMS 管理工具來建立受管理物件。

MQTopicConnectionFactory 物件以名稱 ivtTCF 連結，並以其所有內容的預設值建立，這表示 IVT 程式以連結模式執行，連接至預設佇列管理程式，並使用內嵌的發佈/訂閱功能。 如果您想要 IVT 程式以用戶端模式執行，連接至預設佇列管理程式以外的佇列管理程式，或使用 IBM Integration Bus 而非內嵌式發佈/訂閱功能，則必須使用 IBM MQ JMS 管理工具或「IBM MQ 探險家」來變更 MQTopicConnectionFactory 物件的適當內容。 如需如何使用 IBM MQ JMS 管理工具的相關資訊，請參閱 使用管理工具來配置 JMS 物件。 如需如何使用 IBM MQ Explorer 的相關資訊，請參閱 IBM MQ Explorer 提供的說明。

MQTopic 物件與名稱 ivtT 連結，並使用其所有內容的預設值建立，但 TOPIC 內容除外，其值為 MQJMS/PSIVT/Information。

建立受管理物件之後，您可以執行 IVT 程式。 如果要使用 JNDI 來執行測試，請輸入下列指令:

```
PSIVTRun -url "providerURL" [-icf initCtxFact ] [-t]
```

指令上的參數具有下列意義:

-url "*providerURL*"
目錄服務的統一資源定址器 (URL)。 URL 可以具有下列其中一種格式:

- `ldap://hostname/contextName` ，適用於以 LDAP 伺服器為基礎的目錄服務
- `file:/directoryPath` ，適用於基於本端檔案系統的目錄服務

請注意，您必須用引號 (") 括住 URL 。

-icf *initCtx* 事實
起始環境定義 Factory 的類別名稱，必須是下列其中一個值:

- `com.sun.jndi.ldap.LdapCtxFactory`，適用於以 LDAP 伺服器為基礎的目錄服務。 這是預設值。
- `com.sun.jndi.fscontext.RefFSContextFactory`，適用於以本端檔案系統為基礎的目錄服務。

-t
已啟用追蹤。 依預設，會停用追蹤。

成功測試會產生類似於成功測試而不使用 JNDI 的輸出。 主要差異是輸出指出測試正在使用 JNDI 來擷取 MQTopicConnectionFactory 物件和 MQTopic 物件。

雖然並非嚴格必要，但最好在測試之後，刪除 IVTSetup Script 所建立的受管理物件來進行清理。 為此目的提供了 Script。 此 Script 在 AIX and Linux 系統上稱為 IVTTidy，在 Windows 上稱為 IVTTidy.bat，位於 IBM MQ classes for JMS 安裝目錄的 bin 子目錄中。

發佈/訂閱安裝驗證測試的問題判斷

安裝驗證測試可能因下列原因而失敗:

- 如果 IVT 程式寫入訊息指出找不到類別，請檢查是否已正確設定類別路徑，如 第 78 頁的『設定 IBM MQ classes for JMS/Jakarta Messaging 的環境變數』中所述。
- 測試可能會失敗，並出現下列訊息:

```
Failed to connect to queue manager ' qmgr ' with
connection mode ' connMode ' and host name ' hostname '
```

以及相關聯的原因碼 2059。 訊息中的變數具有下列意義:

qmgr
IVT 程式嘗試連接的佇列管理程式名稱。 如果 IVT 程式嘗試以連結模式連接至預設佇列管理程式，則此訊息插入為空白。

connMode
連線模式，即 `Bindings` 或 `Client`。

hostname
執行佇列管理程式之系統的主機名稱或 IP 位址。

此訊息表示 IVT 程式嘗試連接的佇列管理程式無法使用。 請檢查佇列管理程式是否在執行中，如果 IVT 程式正在嘗試連接至預設佇列管理程式，請確定佇列管理程式定義為系統的預設佇列管理程式。

- 測試可能會失敗，並出現下列訊息:

```
Unable to bind to object
```

此訊息表示有與 LDAP 伺服器的連線，但 LDAP 伺服器未正確配置。 未配置 LDAP 伺服器來儲存 Java 物件，或對物件或字尾的許可權不正確。 如需此狀況的相關說明，請參閱 LDAP 伺服器的說明文件。

- 測試可能會失敗，並出現下列訊息:

```
The security authentication was not valid that was supplied for
QueueManager 'qmgr' with connection mode 'Client' and host name 'hostname'
```

此訊息表示未正確設定佇列管理程式來接受來自系統的用戶端連線。 如需相關資訊，請參閱 第 897 頁的『配置佇列管理程式以接受 Multiplatforms 上的用戶端連線』。

JMS 2.0 使用 *IBM MQ classes for JMS* 範例應用程式

IBM MQ classes for JMS 範例應用程式提供 JMS API 的一般特性概觀。 您可以使用它們來驗證安裝及傳訊伺服器的設定，並協助您建置自己的應用程式。

關於這項作業

如果您需要協助來建立自己的應用程式，您可以使用範例應用程式作為起點。 會為每一個應用程式提供原始檔及已編譯版本。 請檢閱範例原始碼，並識別建立應用程式 (ConnectionFactory、「連線」、「階段作業」、「目的地」及/或「生產者」或「消費者」) 的每一個必要物件的主要步驟，以及設定指定應用程式運作方式所需的任何特定內容。 如需相關資訊，請參閱 第 118 頁的『撰寫 IBM MQ classes for JMS/Jakarta Messaging 應用程式』。 這些範例可能會在 IBM MQ 的未來版本中變更。

對於 JMS 2.0， 第 101 頁的表 11 顯示 IBM MQ classes for JMS 範例應用程式在每一個平台上的安裝位置。

註:

JM 3.0 對於 IBM MQ classes for Jakarta Messaging，正在準備新的範例。

表 *11: IBM MQ classes for JMS* 範例應用程式的安裝目錄

平台	目錄
AIX AIX Linux Linux	*MQ_INSTALLATION_PATH*`/samp/jms/samples`
Windows Windows	*MQ_INSTALLATION_PATH*`\tools\jms\samples`
IBM i IBM i	`/qibm/proddata/mqm/java/samples/jms/samples`
z/OS z/OS	*MQ_INSTALLATION_PATH*`/java/samples/jms`

在此目錄內，有子目錄包含一或多個範例應用程式，如 第 102 頁的表 12 所示。

表 *12: IBM MQ classes for JMS* 範例應用程式

範例名稱	**說明**
JmsBrowser.java	JMS 佇列瀏覽器應用程式，以消費者應用程式接收訊息的順序查看具名佇列上所有可用的訊息，而不移除它們。
JmsConsumer.java	JMS 佇列瀏覽器應用程式，透過在起始環境定義中查閱 Connection Factory 實例及目的地實例 (此範例僅支援檔案系統環境定義)，依消費者應用程式接收訊息的順序查看具名佇列上的所有可用訊息，而不移除它們。
JmsJndiConsumer.java	透過在起始環境定義中查閱 Connection Factory 實例和目的地實例 (此範例僅支援檔案系統環境定義)，從指名目的地 (佇列或主題) 接收訊息的 JMS 消費者 (接收端或訂閱者) 應用程式。
JmsJndiProducer.java	JMS 生產者 (傳送端或發佈者) 應用程式，透過在起始環境定義中查閱 Connection Factory 實例和目的地實例，將簡式訊息傳送至指名的目的地 (佇列或主題) (此範例僅支援檔案系統環境定義)。
JmsProducer.java	這是一個 JMS 生產者 (傳送端或發佈者) 應用程式，用來將簡式訊息傳送至指名的目的地 (佇列或主題)。
/interactive/	
SampleConsumerJava.java	從主題/佇列接收訊息。
SampleProducerJava.java	將訊息傳送至主題/佇列。
/interactive/helper/	
BaseOptions.java	可延伸以提供使用者選項功能的抽象類別。
IsValidType.java	有效性檢查程式類別的抽象類別。
JmsApp.java	可延伸以提供消費者/生產者功能的抽象類別。
Keys.java	一組索引鍵，用於定義範例應用程式的選項。
Literals.java	一組常數文字。
MyContext.java	呈現選項的環境定義。
Options.java	提供使用者選項的功能。
OptionsPresenter.java	呈現現行選項的環境定義。
/simple/	
SimpleAsyncPutPTP.java	點對點傳訊的簡式應用程式; 非同步傳送訊息 (也稱為 *fire-and-*遺忘 傳訊)。 未收到任何訊息。
SimpleDurableSub.java	示範可延續訂閱機能的簡式應用程式。

表 12: IBM MQ classes for JMS 範例應用程式 (繼續)	
範例名稱	**說明**
SimpleJNDILookup.java	一個最小且簡單的應用程式，示範使用起始環境定義來查閱 JMS 物件。 不會建立與佇列管理程式的連線，也不會傳送或接收任何訊息。
SimpleMQMDRead.java	一個簡式應用程式，示範 JMS 應用程式如何利用 MQ 訊息描述子 (MQMD) 欄位作為 JMS 訊息內容。 不會傳送任何訊息; 會假設使用中的佇列已移入部分訊息。
SimpleMQMDWrite.java	一個簡式應用程式，示範 JMS 應用程式如何撰寫 MQ 訊息描述子 (MQMD) 欄位。 未收到任何訊息。
SimplePTP.java	點對點傳訊的最小及簡式應用程式。
SimplePubSub.java	發佈/訂閱傳訊的最小及簡式應用程式。
SimpleReadAheadPTP.java	用於點對點傳訊的簡式應用程式; 訊息從佇列管理程式 (也稱為先讀機能) 串流。 不會傳送任何訊息; 會假設使用中的佇列已移入部分訊息。
SimpleRequestor.java	使用要求者傳送要求訊息，然後等待並接收回覆的簡式應用程式。 附註: 假設其他應用程式會處理要求訊息並傳送回覆訊息。
SimpleResponder.java	在目的地接聽訊息，然後將回覆傳送至訊息的 replyTo 目的地的簡式應用程式。 應用程式撰寫為與 SimpleRequestor 範例一起運作。
SimpleRetainedPub.java	示範保留發佈資訊的簡式應用程式。 未收到任何訊息。
SimpleWMQJMSPTP.java	點對點傳訊的最小及簡式應用程式。
SimpleWMQJMSPubSub.java	發佈/訂閱傳訊的最小簡式應用程式。

IBM MQ classes for JMS 提供一個稱為 `runjms` 的 Script，可用來執行範例應用程式。 此 Script 會設定 IBM MQ 環境，以容許您執行 IBM MQ classes for JMS 範例應用程式。

第 103 頁的表 13 顯示每一個平台上 Script 的位置:

表 13: `runjms` Script 的位置	
平台	**目錄**
AIX AIX Linux Linux	*MQ_INSTALLATION_PATH*`/java/bin/runjms`
Windows Windows	*MQ_INSTALLATION_PATH*`\java\bin\runjms.bat`

IBM i IBM i	`/qibm/proddata/mqm/java/bin/runjms`
	or
	`/qibm/proddata/mqm/java/bin/runjms64`

z/OS z/OS	*MQ_INSTALLATION_PATH*`/java/bin/runjms`

若要使用 `runjms` Script 來呼叫範例應用程式，請完成下列步驟:

程序

1. 啟動命令提示字元，並導覽至包含您要執行之範例應用程式的目錄。
2. 輸入下列指令：

```
Path to the runjms script/runjms sample_application_name
```

   範例應用程式會顯示它需要的參數清單。
3. 輸入下列指令，以使用這些參數來執行範例：

```
Path to the runjms script/runjms sample_application_name parameters
```

範例

Linux 例如，若要在 Linux 上執行 `JmsBrowser` 範例，請輸入下列指令：

```
cd /opt/mqm/samp/jms/samples
/opt/mqm/java/bin/runjms JmsBrowser -m QM1 -d LQ1
```

相關概念

第 74 頁的『針對 IBM MQ classes for JMS 安裝的內容』
當您安裝 IBM MQ classes for JMS 時，會建立一些檔案和目錄。 在 Windows 上，透過自動設定環境變數，在安裝期間執行部分配置。 在其他平台及某些 Windows 環境中，您必須先設定環境變數，然後才能執行 IBM MQ classes for JMS 應用程式。

IBM MQ classes for JMS/Jakarta Messaging 隨附的 *Script*

提供了一些 Script 來協助處理使用 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 時需要執行的一般作業。

第 104 頁的表 14 列出所有 Script 及其用法。 這些 Script 位於 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 安裝目錄的 `bin` 子目錄中。

表 *14: IBM MQ classes for JMS* 和 *IBM MQ classes for Jakarta Messaging* 隨附的 *Script*

效用	使用
清除 [1]	為了與舊版相容而維護此 Script，但不執行任何功能。 不再需要手動清除訂閱資訊。
DefaultConfiguration	在 Windows 以外的平台上執行預設配置應用程式。
formatLog [1]	為了與舊版相容而維護此 Script，但不執行任何功能。 現在以可讀取文字產生日誌輸出。
IVTRun [1] IVTSetup [1] IVTTidy [1]	用於點對點安裝驗證測試，如 第 94 頁的『IBM MQ classes for JMS 的點對點 IVT』中所述。
JM 3.0 JMS30Admin [1]	執行 IBM MQ Jakarta Messaging 管理工具，如 啟動管理工具中所述。
JM 3.0 JMS30Admin.config	IBM MQ Jakarta Messaging 管理工具的配置檔，如 配置 JMS 管理工具中所述。
JMS 2.0 JMSAdmin [1]	執行 IBM MQ JMS 管理工具，如 啟動管理工具中所述。
JMS 2.0 JMSAdmin.config	IBM MQ JMS 管理工具的配置檔，如 配置 JMS 管理工具中所述。
PSIVTRun [1]	執行發佈/訂閱安裝驗證測試程式，如 第 98 頁的『IBM MQ classes for JMS 的發佈/訂閱 IVT』中所述。

表 14: *IBM MQ classes for JMS* 和 *IBM MQ classes for Jakarta Messaging* 隨附的 *Script (繼續)*	
效用	**使用**
PSReportDump.class	此類別是為了與舊版相容而維護，但不執行任何功能。
JM 3.0 V 9.3.0 V 9.3.0 setjms30env 第 105 頁的『2』	Linux AIX 對於 Jakarta Messaging 3.0，設定環境變數以在 AIX and Linux 系統上的 32 位元 Java 虛擬機器 (JVM) 中執行 IBM MQ classes for JMS 應用程式，如 第 78 頁的『設定 IBM MQ classes for JMS/Jakarta Messaging 的環境變數』中所述。
JMS 2.0 setjmsenv 第 105 頁的『2』	Linux AIX 對於 JMS 2.0，設定環境變數以在 AIX and Linux 系統上的 32 位元 Java 虛擬機器 (JVM) 中執行 IBM MQ classes for JMS 應用程式，如 第 78 頁的『設定 IBM MQ classes for JMS/Jakarta Messaging 的環境變數』中所述。
JM 3.0 V 9.3.0 V 9.3.0 setjms30env64 第 105 頁的『2』	Linux AIX 對於 Jakarta Messaging 3.0，設定環境變數以在 AIX and Linux 系統上的 64 位元 JVM 中執行 IBM MQ classes for JMS 應用程式，如 第 78 頁的『設定 IBM MQ classes for JMS/Jakarta Messaging 的環境變數』中所述。
JMS 2.0 setjmsenv64 第 105 頁的『2』	Linux AIX 對於 JMS 2.0，設定環境變數以在 AIX and Linux 系統上的 64 位元 JVM 中執行 IBM MQ classes for JMS 應用程式，如 第 78 頁的『設定 IBM MQ classes for JMS/Jakarta Messaging 的環境變數』中所述。

註:

1. 在 Windows 上，檔名具有副檔名 `.bat`。
2. 這些 Script 只能在 AIX and Linux 上使用。 在 Windows 上，安裝 IBM MQ 之後，請執行指令 **setmqenv**。 如需相關資訊，請參閱 第 78 頁的『設定 IBM MQ classes for JMS/Jakarta Messaging 的環境變數』。

支援 *OSGi* 與 *IBM MQ classes for JMS*

OSGi 提供一個架構，可支援將應用程式部署成軟體組。 OSGi 軟體組作為 IBM MQ classes for JMS 的一部分提供。

IBM MQ classes for JMS 包含下列 OSGi 軟體組。

com.ibm.msg.client.osgi.jms*version_number*.jar
IBM MQ classes for JMS 中的一般程式碼層。 如需 IBM MQ classes for JMS 的分層架構相關資訊，請參閱 IBM MQ classes for JMS 架構。

com.ibm.msg.client.osgi.jms.prereq_*version_number*.jar
共用層的必備 Java 保存檔 (JAR)。

com.ibm.msg.client.osgi.commonservices.j2se_*version_number*.jar
Java Platform, Standard Edition (Java SE) 應用程式的共用服務。

com.ibm.msg.client.osgi.nls_*version_number*.jar
共用層的訊息。

com.ibm.msg.client.osgi.wmq_*version_number*.jar
IBM MQ classes for JMS 中的 IBM MQ 傳訊提供者。 如需 IBM MQ classes for JMS 的分層架構相關資訊，請參閱 IBM MQ for JMS 架構類別。

com.ibm.msg.client.osgi.wmq.prereq_*version_number*.jar
IBM MQ 傳訊提供者的必備 JAR 檔。

com.ibm.msg.client.osgi.wmq.nls_*version_number*.jar
IBM MQ 傳訊提供者的訊息。

com.ibm.mq.jakarta.osgi.allclient_*version_number*.jar

JM 3.0 V 9.3.0 V 9.3.0 對於 Jakarta Messaging 3.0，這個 JAR 檔可讓應用程式同時使用 IBM MQ classes for JMS 和 IBM MQ classes for Java，也包含用來處理 PCF 訊息的程式碼。

com.ibm.mq.jakarta.osgi.allclientprereqs_*version_number*.jar

JM 3.0 V 9.3.0 V 9.3.0 對於 Jakarta Messaging 3.0，此 JAR 檔提供 com.ibm.mq.jakarta.osgi.allclient_*version_number*.jar 的必要條件。

com.ibm.mq.osgi.allclient_*version_number*.jar

JMS 2.0 對於 JMS 2.0，這個 JAR 檔可讓應用程式同時使用 IBM MQ classes for JMS 和 IBM MQ classes for Java，也包含用來處理 PCF 訊息的程式碼。

com.ibm.mq.osgi.allclientprereqs_*version_number*.jar

JMS 2.0 對於 JMS 2.0，此 JAR 檔提供 com.ibm.mq.osgi.allclient_*version_number*.jar 的必要條件。

其中 *version_number* 是所安裝 IBM MQ 的版本號碼。

軟體組會安裝至 IBM MQ 安裝架構的 java/lib/OSGi 子目錄，或 Windows 上的 java\lib\OSGi 資料夾。

從 IBM MQ 8.0 中，針對任何新的應用程式使用組合 com.ibm.mq.osgi.allclient_8.0.0.0.jar 及 com.ibm.mq.osgi.allclientprereqs_8.0.0.0.jar。 使用這些軟體組會移除無法在相同 OSGi 架構內同時執行 IBM MQ classes for JMS 及 IBM MQ classes for Java 的限制，但所有其他限制仍適用。

組合 com.ibm.mq.osgi.java*version_number*.jar 也安裝在 IBM MQ 安裝架構的 java/lib/OSGi 子目錄中，或 Windows 上的 java\lib\OSGi 資料夾中，是 IBM MQ classes for Java 的一部分。 此軟體組不得載入至已載入 IBM MQ classes for JMS 的 OSGi 執行時期環境。

IBM MQ classes for JMS 的 OSGi 組合已寫入 OSGi 第 4 版規格。 它們在 OSGi 第 3 版環境中無法運作。

您必須正確設定系統路徑或程式庫路徑， OSGi 執行時期環境才能找到任何必要的 DLL 檔案或共用程式庫。

如果您將 OSGi 軟體組用於 IBM MQ classes for JMS，則暫時主題無法運作。 此外，由於在多類別載入器環境 (例如 OSGi) 中載入類別的固有問題，因此不支援以 Java 撰寫的通道結束程式類別。 使用者軟體組可以知道 IBM MQ classes for JMS 軟體組，但 IBM MQ classes for JMS 軟體組不知道任何使用者軟體組。 因此，IBM MQ classes for JMS 組合中使用的類別載入器無法載入使用者組合中的通道結束程式類別。

如需 OSGi 的相關資訊，請參閱 OSGi 聯盟 網站。

z/OS MQ Adv. VUE JMS/Jakarta Messaging 用戶端與在 z/OS 上執行之批次應用程式的連線功能

在特定狀況下， z/OS 上的 IBM MQ classes for JMS/Jakarta Messaging 應用程式可以使用用戶端連線連接至 z/OS 上的佇列管理程式。 使用用戶端連線可以簡化 IBM MQ 拓蹼。

透過使用用戶端連線，如果 IBM MQ classes for JMS/Jakarta Messaging 應用程式在批次環境中執行，且適用下列其中一個條件，則應用程式可以連接至遠端 z/OS 佇列管理程式:

- LTS V 9.3.4 IBM MQ classes for JMS/Jakarta Messaging 程式碼位於 IBM MQ 9.3.4 或更新版本，或 Long Term Support 已套用 APAR PH56722。 佇列管理程式可以是任何支援的版本。
- 正在連接的佇列管理程式正在執行 IBM MQ Advanced for z/OS Value Unit Edition 授權，因此將 **ADVCAP** 參數設為 ENABLED。 佇列管理程式可以是任何受支援的版本。

如需 IBM MQ Advanced for z/OS Value Unit Edition 的相關資訊，請參閱 IBM MQ 產品 ID 及匯出資訊。

如需 **ADVCAP** 的相關資訊，請參閱 DISPLAY QMGR ; 如需 **QMGRPROD** 的相關資訊，請參閱 START QMGR 。

請注意，批次是唯一受支援的環境; 不支援 JMS/Jakarta Messaging for CICS 或 JMS/Jakarta Messaging for IMS。

z/OS 上的 IBM MQ classes for JMS/Jakarta Messaging 應用程式無法使用用戶端模式連線來連接未在 z/OS 的佇列管理程式。

如果 z/OS 上的 IBM MQ classes for JMS/Jakarta Messaging 應用程式嘗試使用用戶端模式進行連接，且不容許這樣做，則會發出異常狀況訊息 JMSFMQ0005 。

Advanced Message Security (AMS) 支援

IBM MQ classes for JMS/Jakarta Messaging 用戶端應用程式在連接至遠端 z/OS 佇列管理程式時可以使用 AMS ，但必須遵守本主題先前說明的條件。

若要以此方式使用 AMS ，用戶端應用程式必須在 `keystore.conf` 中使用金鑰儲存庫類型 `jceracfks` ，其中:

- 內容名稱字首為 `jceracfks` ，且此名稱字首不區分大小寫。
- 金鑰儲存庫是 RACF 金鑰環。
- 不需要密碼，將予以忽略。 這是因為 RACF 金鑰環不使用密碼。
- 如果您指定提供者，提供者必須是 `IBMJCE`。

當您搭配使用 `jceracfks` 與 AMS 時，金鑰儲存庫必須採用下列格式: `safkeyring://`*`user`*`/`*`keyring`*，其中:

- `safkeyring` 是文字，且此名稱不區分大小寫
- *`user`* 是擁有金鑰環的 RACF 使用者 ID
- *`keyring`* 是 RACF 金鑰環的名稱，且金鑰環的名稱區分大小寫

下列範例使用使用者 `JOHNDOE` 的標準 AMS 金鑰環:

```
jceracfks.keystore=safkeyring://JOHNDOE/drq.ams.keyring
```

相關概念

第 311 頁的『Java 用戶端與在 z/OS 上執行之批次應用程式的連線功能』
在特定條件下， z/OS 上的 IBM MQ classes for Java 應用程式可以使用用戶端連線連接至 z/OS 上的佇列管理程式。 使用用戶端連線可以簡化 IBM MQ 拓蹼。

個別取得 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging

使用 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 來執行應用程式所需的程式庫及公用程式，可在您從 Fix Central 下載的自行解壓縮 JAR 檔中找到。 如果您只想取得這些檔案 (例如，部署至軟體管理工具或與獨立式用戶端應用程式搭配使用) ，請執行此動作。

開始之前

在開始這項作業之前，請確定您已在機器上安裝 Java runtime environment (JRE) ，且 JRE 已新增至系統路徑。

在此安裝程序中使用的 Java 安裝程式不需要以 root 或任何特定使用者身分執行。 唯一的需求是執行它的使用者對您要檔案進入的目錄具有寫入權。

JM 3.0 V 9.3.0 V 9.3.0 從 IBM MQ 9.3.0 開始，支援 Jakarta Messaging 3.0 開發新的應用程式。 IBM MQ 9.3.0 繼續支援 JMS 2.0 現有的應用程式。 不支援在同一應用程式中同時使用 Jakarta Messaging 3.0 API 和 JMS 2.0 API。 如需相關資訊，請參閱 使用 IBM MQ 類別進行 JMS/Jakarta 傳訊。

關於這項作業

會使用自行解壓縮的 JAR 檔，將下載的大小及執行擷取所花費的時間縮至最小。 這個 JAR 檔的確切內容，以及它將檔案解壓縮到其中的子目錄，取決於 IBM MQ 的版本。

當您執行自行解壓縮的 JAR 檔時，它會顯示必須接受的 IBM MQ 授權合約。 它也可讓您變更擷取的上層目錄。

對於 IBM MQ 9.3，自行解壓縮的 JAR 檔會將檔案解壓縮至下列目錄結構:

wmq/JavaEE
IBM MQ 資源配接器 EAR 和 RAR 檔。

JMS 2.0 下列檔案與 JMS 2.0 及 JMS 1.1 物件搭配使用:

- `wmq.jmsra.ivt.ear`
- `wmq.jmsra.rar`

JM 3.0 也有一個與 Jakarta Messaging 3.0 物件搭配使用的對等集:

- `wmq.jakarta.jmsra.ivt.ear`。 包含「安裝驗證測試」檔案。
- `wmq.jakarta.jmsra.rar`。 包含資源配接器檔案。

wmq/JavaSE

wmq/JavaSE/bin
JMSAdmin 和 **JMS30Admin** 工具。 用來定義代表 JMS 或 Jakarta Messaging 物件的 JNDI 實體。

JMS 2.0 下列檔案與 JMS 2.0 及 JMS 1.1 物件搭配使用:

- `JMSAdmin.bat`
- `JMSAdmin`
- `JMSAdmin.config`

JM 3.0 也有一個與 Jakarta Messaging 3.0 物件搭配使用的對等集:

- `JMS30Admin.bat`。 用來在 Windows 上啟動工具的檔案。
- `JMS30Admin`。 在 Linux 和 UNIX 平台上用來啟動工具的 Script。
- `JMS30Admin.config`。 工具的範例配置檔。

註:

- 在 **JMSAdmin** 工具新增至自行解壓縮的 JAR 檔之前，此目錄中的檔案位於上層目錄 `wmq/JavaSE` 中。
- 使用自行解壓縮 JAR 檔安裝的用戶端可以使用 **JMSAdmin** 或 **JMS30Admin** 工具，在檔案系統環境定義 (`.bindings` 檔案) 內建立 Java 傳訊受管理物件。 用戶端也可以查閱及使用這些受管理物件。
- JMS 2.0 與 JMS 2.0 和 JMS 1.1 物件搭配使用的 **JMSAdmin** 工具已新增至位於 IBM MQ 9.2.0 Fix Pack 2 和 IBM MQ 9.2.2 的自行解壓縮 JAR 檔。
- JM 3.0 與 Jakarta Messaging 3.0 物件搭配使用的 **JMS30Admin** 工具已新增至 IBM MQ 9.3.0 中自行解壓縮的 JAR 檔。

wmq/JavaSE/lib
Advanced Message Security 使用下列開放程式碼 Bouncy Castle 套件來支援加密訊息語法 (CMS)。 請參閱 支援具有 AMS 的非 IBM JRE。

V 9.3.5 若為 IBM MQ 9.3.5 中的 Continuous Delivery :

- `bcpkix-jdk18on.jar`
- `bcprov-jdk18on.jar`
- `bcutil-jdk18on.jar`

LTS 若為 Long Term Support 及 IBM MQ 9.3.5 之前的 Continuous Delivery :

- `bcpkix-jdk15on.jar`
- `bcprov-jdk15on.jar`
- `bcutil-jdk15on.jar`

下列檔案各包含其特定 JMS 或 Jakarta Messaging 層次的類別:

- JMS 2.0 `com.ibm.mq.allclient.jar` (JMS 2.0 和 JMS 1.1)
- JM 3.0 `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0)

其他必備 JAR 檔:

- V 9.3.3 Removed `com.ibm.mq.traceControl.jar`。 用來動態控制 IBM MQ classes for JMS 應用程式的追蹤。
- `fscontext.jar`。 如果您的應用程式使用檔案系統環境定義來執行 JNDI 查閱，則為必要。
- V 9.3.3 `jackson-annotations.jar`、`jackson-core.jar`、`jackson-databind.jar`: 包含在建立與佇列管理程式的安全 TLS 連線時，用來執行 CipherSuite 及 CipherSpec 對映的類別。
- JM 3.0 `jakarta.jms-api.jar`。 包含 Jakarta Messaging 3.0 介面和異常狀況定義。
- JMS 2.0 `jms.jar`。 包含 JMS 2.0 介面和異常狀況定義。
- `org.json.jar`。 包含容許 IBM MQ classes for JMS 解譯 JSON 格式 CCDT 檔案的類別。
- `providerutil.jar`。 如果您的應用程式使用檔案系統環境定義來執行 JNDI 查閱，則為必要。

註: Stabilized `com.ibm.mq.allclient.jar` 和 `com.ibm.mq.jakarta.client.jar` 都包含 IBM MQ classes for Java 的副本。 不過，在 IBM MQ 9.0 中，這些類別在 IBM MQ 8.0 隨附的層次宣告為功能穩定。 請參閱 IBM MQ 9.0 中的淘汰、穩定和移除。

wmq/OSGi

IBM MQ OSGi 用戶端軟體組:

- JM 3.0 `com.ibm.mq.jakarta.osgi.allclient_V.R.M.F.jar`
- JM 3.0 `com.ibm.mq.jakarta.osgi.allclientprereqs_V.R.M.F.jar`
- JMS 2.0 `com.ibm.mq.osgi.allclient_V.R.M.F.jar`
- JMS 2.0 `com.ibm.mq.osgi.allclientprereqs_V.R.M.F.jar`

其中 *V.R.M.F* 是版本、版次、修正層次及修正套件號碼。

程序

1. 從 Fix Central 下載 IBM MQ Java / JMS 用戶端 JAR 檔。
   a) 按一下此鏈結: IBM MQ Java / JMS 用戶端。
   b) 在顯示的可用修正程式清單中，尋找您 IBM MQ 版本的用戶端。
   例如:

   ```
   release level: 9.3.0.0-IBM-MQ-Install-Java-All
   Long Term Support: 9.3.0.0 IBM MQ JMS and Java 'All Client'
   ```

   然後按一下用戶端檔名，並遵循下載處理程序。
2. 從您將檔案下載至其中的目錄開始解壓縮。
   若要開始擷取，請以下列格式輸入指令:

   ```
   java -jar V.R.M.F-IBM-MQ-Install-Java-All.jar
   ```

   其中 *V.R.M.F* 是產品版本號碼，例如 `9.3.0.0`，而 *V.R.M.F*`-IBM-MQ-Install-Java-All.jar` 是從 Fix Central 下載的檔案名稱。
   例如，若要解壓縮 IBM MQ 9.3.0 版本的 JMS 用戶端，您將使用下列指令:

   ```
   java -jar 9.3.0.0-IBM-MQ-Install-Java-All.jar
   ```

註: 若要執行此安裝，您必須在機器上安裝 JRE 並新增至系統路徑。

當您輸入指令時，會顯示下列資訊:

```
在使用、解壓縮或安裝 IBM MQ V9.3 之前，您必須先接受
1 的條款。 IBM 國際授權合約-評估
方案 2. IBM 國際程式授權合約及其他
授權資訊。 請仔細閱讀下列授權合約。

授權合約可使用來個別檢視
--viewLicenseAgreement 選項。

請按 Enter 鍵立即顯示授權條款，或按 'x' 跳過。
```

3. 檢閱並接受授權條款:
   a) 若要顯示授權，請按 Enter 鍵。
      或者，按 x 以跳過授權的顯示畫面。
      顯示授權之後，如果您按下 x，則會立即顯示下列訊息:

      ```
      其他授權資訊可使用來個別檢視
      --viewLicenseInfo 選項。

      請按 Enter 鍵立即顯示其他授權資訊，或按 'x' 跳過。
      ```

   b) 若要顯示其他授權條款，請按 Enter 鍵。
      或者，按 x 以跳過顯示其他授權條款。
      在顯示其他授權條款之後，如果您按下 x，則會立即顯示下列訊息:

      ```
      選擇下面的「我同意」選項，即表示您同意
      授權合約及非 IBM 條款（如果適用的話）。 如果您未
      同意，選取「我不同意」。

      選取 [1] 我同意，或 [2] 我不同意:
      ```

   c) 若要接受授權合約並繼續選取安裝目錄，請選取 1。
      或者，選取 2 以立即結束安裝。
      如果您選取 1，則會顯示類似下列訊息的訊息:

      ```
      輸入產品檔案的目錄，或保留空白以接受預設值。
      預設目標目錄為 H: \downloads

      產品檔案的目標目錄?
      ```

4. 指定擷取的上層目錄。

   預設位置是現行目錄。

   - 如果您要將產品檔案解壓縮至預設位置，請按 Enter 鍵而不指定值。
   - 如果您要將產品檔案解壓縮至不同位置，請指定您要將檔案解壓縮至其中的目錄名稱，然後按 Enter 鍵以開始解壓縮。

   您指定的目錄名稱必須尚未存在，否則，當您啟動解壓縮時，會報告錯誤，且不會安裝任何檔案。
   如果它尚未存在，則會建立指定的目錄，並將程式檔案解壓縮至此目錄。 在安裝期間，會在您指定的上層目錄內建立名稱為 wmq 的新目錄。

   在 wmq 目錄中建立三個子目錄 (JavaEE、 JavaSE 和 OSGi)，其內容如下:

   JavaEE

   JM 3.0 wmq.jakarta.jmsra.ivt.ear

   JM 3.0 wmq.jakarta.jmsra.rar

   JMS 2.0 wmq.jmsra.ivt.ear

   JMS 2.0 wmq.jmsra.rar

   JavaSE
   此目錄包含下列子目錄及檔案:

   JavaSE/lib

   V 9.3.5 bcpkix-jdk18on.jar

LTS bcpkix-jdk15on.jar

V 9.3.5 bcprov-jdk18on.jar

LTS bcprov-jdk15on.jar

V 9.3.5 bcutil-jdk18on.jar

LTS bcutil-jdk15on.jar

JMS 2.0 com.ibm.mq.allclient.jar

JM 3.0 com.ibm.mq.jakarta.client.jar

V 9.3.3 Removed com.ibm.mq.traceControl.jar

fscontext.jar

V 9.3.3 jackson-annotations.jar

V 9.3.3 jackson-core.jar

V 9.3.3 jackson-databind.jar

jms.jar

org.json.jar

providerutil.jar

JavaSE/bin

JMSAdmin.bat

JMSAdmin

JMSAdmin.config

OSGi

JM 3.0 com.ibm.mq.jakarta.osgi.allclient_*V.R.M.F*.jar

JM 3.0 com.ibm.mq.jakarta.osgi.allclientprereqs_*V.R.M.F*.jar

JMS 2.0 com.ibm.mq.osgi.allclient_*V.R.M.F*.jar

JMS 2.0 com.ibm.mq.osgi.allclientprereqs_*V.R.M.F*.jar

當擷取完成時，會顯示確認訊息，如下列範例所示:

```
將檔案解壓縮至 H: \downloads\wmq
已順利擷取所有產品檔案。
```

IBM MQ classes for JMS/Jakarta Messaging 中的允許清單

Java 物件序列化和解除序列化機制已識別為潛在安全風險。 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 中的允許清單提供一些針對部分序列化風險的保護。

關於這項作業

Java 物件序列化和解除序列化機制已識別為潛在安全風險，因為解除序列化會實例化任意 Java 物件，其中可能會惡意傳送資料來導致各種問題。 序列化的一個值得注意的應用程式是在 Jakarta Messaging 3.0 和 Java Message Service 2.0 ObjectMessages 中，使用序列化來封裝及傳送任意物件。

序列化允許清單是對序列化所造成的部分風險的潛在緩解。 透過明確指定哪些類別可以封裝在 ObjectMessages 中，以及從 ObjectMessages 中擷取這些類別，允許清單可提供一些保護來避免某些序列化風險。

相關概念

第 90 頁的『在 Java security manager 下執行 IBM MQ classes for JMS 應用程式』

IBM MQ classes for JMS 可以在啟用 Java security manager 的情況下執行。 若要在啟用 Java security manager 的情況下順利執行應用程式，您必須使用適當的原則配置檔來配置 Java Virtual Machine (JVM)。

列入允許清單的概念

在 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 中，支援在 JMS ObjectMessage 介面的實作中將類別列入允許清單。 這可針對可能與 Java 物件序列化和解除序列化機制相關的部分安全風險，提供潛在的緩解。

IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 中的允許清單

重要:

可能的情況下，術語 *allowlist* 已替代術語 *whitelist*. 對於 IBM MQ 9.0 以及更新版本，這包括本主題中提及的部分 Java 系統內容名稱。 不需要變更任何現有的配置。 先前的系統內容名稱也會繼續運作。

IBM MQ classes for JMS (JMS 2.0) V 9.3.0 V 9.3.0 和 IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0) 在 JMS ObjectMessage 介面的實作中支援類別的允許清單。

- JMS 2.0 對於 IBM MQ classes for JMS，相關內容名稱為 **com.ibm.mq.jms.allowlist.***。
- JM 3.0 對於 IBM MQ classes for Jakarta Messaging，相關內容名稱為 **com.ibm.mq.jakarta.jms.allowlist.***

允許清單定義哪些 Java 類別可以使用 ObjectMessage.setObject() 序列化，並使用 ObjectMessage.getObject() 解除序列化。

- JMS 2.0 嘗試使用 ObjectMessage 來序列化或解除序列化未包含在允許清單中的類別實例，會導致擲出 javax.jms.MessageFormatException ，並以 java.io.InvalidClassException 作為其原因。
- JM 3.0 嘗試使用 ObjectMessage 將未包含在允許清單中的類別實例序列化或解除序列化，會導致擲出 jakarta.jms.MessageFormatException ，並以 java.io.InvalidClassException 作為其原因。

產生允許清單

重要: IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 無法與允許清單一起配送。 使用 ObjectMessages 來選擇要傳送的類別是應用程式設計選項， IBM MQ 無法先占。

因此，列入允許清單機制容許兩種作業模式:

探索
在此模式中，該機制會產生完整類別名稱的清單，並報告在 ObjectMessages 中觀察到要序列化或解除序列化的所有類別。

強制執行
在此模式中，機制會施行列入允許清單，拒絕將不在允許清單中的類別序列化或解除序列化的嘗試。

如果您想要使用此機制，則一開始必須在 DISCOVERY 模式下執行，以收集目前序列化及解除序列化類別的清單，檢閱清單並將其用作允許清單的基礎。 甚至可能適合使用未變更的清單，但在您決定執行此動作之前，必須先檢閱清單。

控制列入名單機制

有三個系統內容可用來控制列入允許清單的機制:

com.ibm.mq.jms.allowlist (JMS 2.0) 和 com.ibm.mq.jakarta.jms.allowlist (Jakarta Messaging 3.0)
可以使用下列其中一種方式來指定此內容:

- 包含允許清單的檔案路徑名稱，採用檔案 URI 格式 (亦即，以 `file:` 開頭)。 在 DISCOVERY 模式中，此檔案是由允許清單機制寫入。 檔案不得存在。 如果檔案確實存在，機制會擲出異常狀況，而不是改寫它。 在 ENFORCEMENT 模式中，允許清單機制會讀取此檔案。
- 組成允許清單的完整類別名稱 (以逗點區隔)。

如果取消設定此內容，則容許清單機制為非作用中。

如果您使用 Java security manager，則必須確保 IBM MQ classes for JMS JAR 檔具有此檔案的讀取及寫入權。

com.ibm.mq.jms.allowlist.discover (JMS 2.0) 和 com.ibm.mq.jakarta.jms.allowlist.discover (Jakarta Messaging 3.0)

- 如果此內容已取消設定或設為 false，則容許清單機制會以 ENFORCEMENT 模式執行。
- 如果此內容設為 true，且容許清單已指定為檔案 URI，則容許清單機制會以 DISCOVERY 模式執行。
- 如果此內容設為 true，且容許清單已指定為類別名稱清單，則容許清單機制會擲出適當的異常狀況。
- 如果此內容設為 true，且尚未使用 com.ibm.mq.jms.allowlist 或 com.ibm.mq.jakarta.jms.allowlist 內容指定允許清單，則允許清單機制是非作用中。
- 如果此內容設為 true，且容許清單檔已存在，則容許清單機制會擲出 java.io.InvalidClassException，且項目不會新增至檔案。

com.ibm.mq.jms.allowlist.mode (JMS 2.0) 和 com.ibm.mq.jakarta.jms.allowlist.mode (Jakarta Messaging 3.0)

此字串內容可以用三種方式中的任何一種來指定:

- 如果此內容設為 SERIALIZE，則 ENFORCEMENT 模式只會對 ObjectMessage.setObject() 方法執行允許清單驗證。
- 如果此內容設為 DESERIALIZE，則 ENFORCEMENT 模式只會對 ObjectMessage.getObject() 方法執行允許清單驗證。
- 如果此內容未設定或設為任何其他值，則 ENFORCEMENT 模式會對 ObjectMessage.getObject() 及 ObjectMessage.setObject() 方法執行允許清單驗證。

允許清單檔案的格式

以下是容許清單檔格式的主要特性:

- 允許清單檔採用預設平台檔案編碼，且具有適合平台的行尾。

  註: 如果正在使用容許清單檔，則一律會使用 JVM 的預設檔案編碼來寫入及讀取該檔案。

  如果允許清單檔以下列任何方式產生，這就很好:

  - z/OS 由在 z/OS 上執行的獨立式應用程式所產生，並由也在 z/OS 上執行的其他獨立式應用程式所使用。
  - 由在任何平台上 WebSphere Application Server 內部執行的應用程式所產生，並由另一個 WebSphere Application Server 實例使用。
  - Multi 由在 IBM MQ for Multiplatforms 上執行的獨立式應用程式所產生，並由在 IBM MQ for Multiplatforms 上執行的其他獨立式應用程式所使用，或由在任何平台上 WebSphere Application Server 內執行的應用程式所產生。

  不過，由於 WebSphere Application Server 使用 ASCII，且獨立式 JVM 使用 EBCDIC，因此如果以下列其中一種方式產生容許清單檔，則會有檔案編碼問題:

  - 在 z/OS 上產生，然後由 z/OS 或 WebSphere Application Server 以外的平台上執行的獨立式應用程式使用。
  - 由 WebSphere Application Server 或在 z/OS 以外的平台上執行的獨立式應用程式產生，然後由 z/OS 上的獨立式應用程式使用。
- 每一個非空行都包含完整類別名稱。 空行會被忽略。
- 可以包含註解-忽略行尾 '#' 字元之後的任何內容。
- 有一個非常基本的萬用字元機制:
  - '*' 可以是類別名稱的 **last** 元素。
  - '*' 符合類別名稱的 **single** 元素，即類別，但不符合套件的任何部分。

因此 com.ibm.mq.* 符合 com.ibm.mq.MQMessage，但不符合 com.ibm.mq.jmqi.remote.api.RemoteFAP。

萬用字元不適用於預設套件中沒有明確套件名稱之類別的類別，因此會拒絕類別名稱 "*"。

- 格式不正確的允許清單檔 (例如，包含 com.ibm.mq.*.Message 之類的項目的檔案，其中萬用字元不是最後一個元素) 會導致擲出 java.lang.IllegalArgumentException 。
- 空的允許清單檔具有完全停用使用 ObjectMessage 的效果。

允許清單的格式 (以逗點區隔清單)

相同的萬用字元機制可用於容許清單 (以逗點區隔的清單)。

- 如果在指令行上或 Shell Script 或批次檔中指定，則作業系統可以展開 '*' ，因此可能需要特殊處理。
- 只有在指定檔案時， '#' 註解字元才適用。 如果將允許清單指定為以逗點區隔的類別名稱清單，則假設作業系統或 Shell 不會處理它，因為它是許多 AIX and Linux Shell 中的預設註解字元，它會被視為一般字元。

何時會列入允許名單?

當應用程式第一次執行 ObjectMessage setMessage() 或 getMessage() 方法時，即會起始允許清單。

系統內容會進行評估，允許清單檔會開啟，且在 ENFORCEMENT 模式下，當起始設定機制時，會載入允許清單類別的清單。 此時，會將項目寫入應用程式的 IBM MQ JMS 日誌檔。

當起始設定機制時，其參數可能不會變更。 因為起始設定的時間並不容易預測，因為它取決於應用程式行為。 因此，從啟動應用程式開始，應該將系統內容設定及允許清單檔案內容視為已修正。 在應用程式執行時，請勿變更允許清單檔的內容或內容，因為不保證會有結果。

要考量的點

減輕 Java 序列化機制本身風險的最佳方法是探索資料傳送的替代方法，例如使用 JSON 而非 ObjectMessage。 使用 Advanced Message Security (AMS) 機制可以確保訊息來自授信來源，從而增加進一步的安全。

如果您將 Java security manager 機制與應用程式搭配使用，則必須授與下列許可權:

- 在您使用的任何允許清單檔上 FilePermission ，具有 ENFORCEMENT 模式的讀取權，以及 DISCOVER 模式的寫入權。
- JMS 2.0 **com.ibm.mq.jms.allowlist**、 **com.ibm.mq.jms.allowlist.discover** 和 **com.ibm.mq.jms.allowlist.mode** 內容上的 PropertyPermission (讀取)。
- JM 3.0 **com.ibm.mq.jakarta.jms.allowlist**、 **com.ibm.mq.jakarta.jms.allowlist.discover** 和 **com.ibm.mq.jakarta.jms.allowlist.mode** 內容上的 PropertyPermission (讀取)。

相關資訊

如需允許清單的相關資訊，請參閱 第 114 頁的『設定及使用 JMS 或 Jakarta Messaging 允許清單』 及 第 116 頁的『WebSphere Application Server 中的允許清單』 。

相關概念

第 90 頁的『在 Java security manager 下執行 IBM MQ classes for JMS 應用程式』
IBM MQ classes for JMS 可以在啟用 Java security manager 的情況下執行。 若要在啟用 Java security manager 的情況下順利執行應用程式，您必須使用適當的原則配置檔來配置 Java Virtual Machine (JVM)。

設定及使用 *JMS* 或 *Jakarta Messaging* 允許清單

此資訊告訴您允許清單的運作方式，以及如何使用 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 中包含的功能來設定允許清單檔，其中包含應用程式可以處理的 ObjectMessages 類型清單。

開始之前

重要:

可能的情況下，術語 *allowlist* 已替代術語 *whitelist*. 對於 IBM MQ 9.0 以及更新版本，這包括本主題中提及的部分 Java 系統內容名稱。 不需要變更任何現有的配置。 先前的系統內容名稱也會繼續運作。

在開始這項作業之前，請確定您已閱讀並瞭解 第 112 頁的『列入允許清單的概念』

關於這項作業

因為 JMS 和 Jakarta Messaging 有許多共同之處，本主題中對 JMS 的進一步參照可以視為同時參照兩者。必要時會強調顯示任何差異。

當您已啟用容許清單功能時， IBM MQ classes for JMS 會以下列方式使用該功能:

- 當應用程式想要傳送 ObjectMessage 時，它可以透過下列兩種方式之一來建立它:
  - Session.createObjectMessage(可序列化) 方法，傳入要包含在訊息內的物件。
  - Session.createObjectMessage() 方法，用來建立空白 ObjectMessage，然後呼叫 ObjectMessage.setObject(可序列化) ，以儲存要在 ObjectMessage 內傳送的物件。

  當呼叫 Session.createObjectMessage(Serializable) 或 ObjectMessage.setObject(Serializable) 方法時，JMS 的類別會檢查傳入的物件是否屬於允許清單中提及的類型。

  如果它是所提及的類型，則會序列化物件並儲存在 ObjectMessage 內。 不過，如果物件的類型不在允許清單中，則 IBM MQ classes for JMS 會擲出包含訊息的 JMSException:

  ```
  JMSCC0052: 序列化物件時發生異常狀況:
  'java.io.InvalidClassException: <object class>; The class may not be serialized
  or deserialized as it has not been included in the allowlist '<allowlist>'.
  ```

  回到應用程式。

  重要: 如果從 Session.createObjectMessage(Serializable) 方法擲出異常狀況，則不會建立 ObjectMessage 。 同樣地，如果從 ObjectMessage.setObject(Serializable) 方法擲出 JMSException ，物件將不會新增至 ObjectMessage。
- 如果應用程式收到 ObjectMessage，它會呼叫 ObjectMessage.getObject() 方法來取得其中包含的物件。當呼叫此方法時， IBM MQ classes for JMS 會檢查 ObjectMessage 中包含的物件類型，以查看該物件是否屬於容許清單中指定的類型。

  如果是的話，會將物件解除序列化並傳回至應用程式。 不過，如果物件的類型不在允許清單中，則 IBM MQ classes for JMS 會擲出包含訊息的 JMSException:

  ```
  JMSCC0053: 解除序列化訊息時發生異常狀況:
  'java.io.InvalidClassException: <object class>; The class may not be
  序列化或解除序列化，因為它尚未併入
  allowlist '<allowlist>'.'.
  ```

  回到應用程式。

例如，假設您的應用程式包含下列程式碼，以傳送包含類型 java.net.URI: 物件的 ObjectMessage :

```
java.net.URL testURL = new java.net.URL("https://www.ibm.com/");
ObjectMessage msg = session.createObjectMessage(testURL);
sender.send(msg);
```

由於未啟用允許清單，因此應用程式能夠順利將訊息放置到所需的目的地。

如果您建立一個稱為 `C:\allowlist.txt` 的檔案，其中包含單一項目 `java.net.URL`，且您使用 Java 系統內容集來重新啟動應用程式:

```
-Dcom.ibm.mq.jms.allowlist=file:/C:/allowlist.txt
```

已啟用允許清單功能。 應用程式仍然可以建立並傳送 ObjectMessage ，其中包含 java.net.URI 類型的物件，因為該類型指定在允許清單中。

不過，如果您變更 allowlist.txt 檔案，使檔案包含單一項目 java.util.Calendar，當應用程式呼叫時，仍會啟用允許清單功能:

```
ObjectMessage msg = session.createObjectMessage(testURL);
```

IBM MQ classes for JMS 會檢查允許清單，並發現它不包含 java.net.URI 的項目。

因此，會擲出包含 JMSCC0052 訊息的 JMSException。

同樣地，假設您有另一個應用程式使用下列程式碼來接收 ObjectMessages :

```
ObjectMessage message = (ObjectMessage)receiver.receive(30000);
if (message != null) {
    Object messageBody = objectMessage.getObject();
    if (messageBody instanceof java.net.URI) {
      :    :    :    :    :    :    :
```

如果未啟用允許清單，則應用程式能夠接收包含任何類型物件的 ObjectMessages 。 然後，在執行適當的處理程序之前，應用程式會檢查物件類型是否為 java.net.URL 。

如果您現在使用 Java 系統內容來啟動應用程式:

```
-Dcom.ibm.mq.jms.allowlist=java.net.URL
```

，則會開啟允許清單功能。 當應用程式呼叫時:

```
Object messageBody = objectMessage.getObject();
```

ObjectMessage.getObject() 方法只會傳回 java.net.URL 類型的物件。

如果 ObjectMessage 中包含的物件不是此類型，則 ObjectMessage.getObject() 方法會擲出包含 JMSCC0053 訊息的 JMSException。 然後，應用程式需要決定如何處理訊息; 例如，訊息可以移至該佇列管理程式的無法傳送郵件的佇列。

只有在 ObjectMessage 內的物件類型為 java.net.URL 時，應用程式才會正常傳回。

程序

1. 執行處理 ObjectMessages 的應用程式，並指定下列 Java 系統內容:

   ```
   -Dcom.ibm.mq.jms.allowlist.discover=true
   -Dcom.ibm.mq.jms.allowlist=file:/<path to your allowlist file>
   ```

   當應用程式執行時， IBM MQ classes for JMS 會建立一個檔案，其中包含應用程式所處理的物件類型。
2. 在一段時間內，應用程式已處理 ObjectMessages 的代表性範例之後，請停止它。

   現在，容許清單檔包含應用程式在執行時所處理的 ObjectMessages 內所有物件類型的清單。

   如果您已執行應用程式足夠時間，則此清單包括應用程式可能處理的 ObjectMessages 內包含的所有可能類型物件。
3. 使用下列系統內容集來重新啟動應用程式:

   ```
   -Dcom.ibm.mq.jms.allowlist=file:/<path to your allowlist file>
   ```

   這會啟用允許清單，如果 IBM MQ classes for JMS 偵測不在允許清單中的類型 ObjectMessage ，則會擲出包含 JMSCC0052 或 JMSCC0053 訊息的 JMSException。

WebSphere Application Server 中的允許清單

如何在 WebSphere Application Server 中使用 IBM MQ classes for JMS 允許清單。

重要:

可能的情況下，術語 *allowlist* 已替代術語 *whitelist*. 對於 IBM MQ 9.0 以及更新版本，這包括本主題中提及的部分 Java 系統內容名稱。 不需要變更任何現有的配置。 先前的系統內容名稱也會繼續運作。

您必須確定 WebSphere Application Server 安裝架構包含支援列入允許清單的 IBM MQ 資源配接器版本。

如需使用這兩個產品的進一步資訊，請參閱 第 415 頁的『同時使用 IBM MQ 和 WebSphere Application Server』 。

IBM MQ 9.0.0 Fix Pack 1 及以上版本包括適當的功能。

更新應用程式伺服器之後，您可以使用 Java 系統內容:

- `-Dcom.ibm.mq.jms.allowlist`
- `-Dcom.ibm.mq.jms.allowlist.discover`

第 114 頁的『設定及使用 JMS 或 Jakarta Messaging 允許清單』中說明。

註: 您需要在用來執行應用程式伺服器的 Java Virtual Machine 上，將 Java 系統內容設為通用 JVM 引數，並重新啟動應用程式伺服器，使變更生效。

如需相關資訊，請參閱 Java 虛擬機器設定 中的 通用 *JVM* 引數 一節。

若要設定內容，請跳至 程序定義 中的 Java Virtual Machine 視窗，並輸入適當的引數。

下列設定:

```
-Dcom.ibm.mq.jms.allowlist=<youruserId>_MyObject
```

導致應用程式伺服器使用允許清單 *youruserId_MyObject*。 應用程式伺服器只會處理該類型的物件。

下列設定:

```
-Dcom.ibm.mq.jms.allowlist.discover=true
-Dcom.ibm.mq.jms.allowlist=file:C/:allowlist.txt
```

將應用程式伺服器配置成使用 探索 模式，並將應用程式伺服器處理的 JMS ObjectMessages 詳細資料記錄到檔案 `C:\allowlist.txt`

下列設定:

```
-Dcom.ibm.mq.jms.allowlist=file:C/:allowlist.txt
```

會使應用程式伺服器載入檔案 `C:/allowlist.txt`，並使用該檔案中的資訊來決定允許清單。

相關概念

第 90 頁的『在 Java security manager 下執行 IBM MQ classes for JMS 應用程式』
IBM MQ classes for JMS 可以在啟用 Java security manager 的情況下執行。 若要在啟用 Java security manager 的情況下順利執行應用程式，您必須使用適當的原則配置檔來配置 Java Virtual Machine (JVM)。

IBM MQ classes for JMS 中的字串轉換

IBM MQ classes for JMS 直接使用 CharsetEncoders 和 CharsetDecoders 來進行字串轉換。 字串轉換的預設行為可以配置兩個系統內容。 包含不可對映字元的訊息處理可以透過訊息內容來配置，以設定 UnmappableCharacter 動作及取代位元組。

在 IBM MQ 8.0 之前， IBM MQ classes for JMS 中的字串轉換是透過呼叫 `java.nio.charset.Charset.decode(ByteBuffer)` 和 `Charset.encode(CharBuffer)` 方法來完成。

使用上述任一方法會導致預設取代 (REPLACE) 形態異常或無法翻譯的資料。 此行為可能會遮蔽應用程式中的錯誤，並在已翻譯的資料中導致非預期的字元 (例如 ?)。

從 IBM MQ 8.0 開始，為了更早且更有效地偵測這類問題， IBM MQ classes for JMS 會直接使用 CharsetEncoders 和 CharsetDecoders ，並明確地配置處理形態異常和不可翻譯的資料。 預設行為是 REPORT 擲出適當的 `MQException` 來發出這類問題。

配置

從 UTF-16 (Java 中使用的字元表示法) 轉換為原生字集 (例如 UTF-8) ， 稱為 編碼，而反向轉換則稱為 解碼。

解碼採用 CharsetDecoders 的預設行為，透過擲出異常狀況來報告錯誤。

其中一項設定是用來指定 java.nio.charset.CodingErrorAction，以控制編碼和解碼兩者的錯誤處理。 在編碼時，會使用另一個設定來控制取代位元組。 解碼作業將使用預設 Java 取代字串。

UnmappableCharacter IBM MQ classes for JMS 中的動作及取代位元組設定

從 IBM MQ 8.0 開始，下列兩個內容可用於設定 UnmappableCharacter 動作及取代位元組。 適當的常數定義位於 com.ibm.msg.client.wmq.WMQConstants 中。

JMS_IBM_UNMAPPABLE_ACTION
在編碼或解碼作業中無法對映字元時，設定或取得要套用的 CodingErrorAction。

您應該將此設為 CodingErrorAction.{REPLACE|REPORT|IGNORE}.toString()，如下所示:

```
public static final String JMS_IBM_UNMAPPABLE_ACTION = "JMS_IBM_Unmappable_Action";
```

JMS_IBM_UNMAPPABLE_REPLACEMENT
在編碼作業中無法對映字元時，設定或取得要套用的取代位元組。

解碼作業會使用預設 Java 取代字串。

```
public static final String JMS_IBM_UNMAPPABLE_REPLACEMENT = "JMS_IBM_Unmappable_Replacement";
```

JMS_IBM_UNMAPPABLE_ACTION 和 JMS_IBM_UNMAPPABLE_REPLACEMENT 內容可以在目的地或訊息上設定。 在訊息上設定的值會置換在訊息傳送目的地上設定的值。

請注意， JMS_IBM_UNMAPPABLE_REPLACEMENT 必須設為單位元組。

設定系統預設值的系統內容

從 IBM MQ 8.0 開始，下列兩個 Java 系統內容可用來配置有關字串轉換的預設行為。

com.ibm.mq.cfg.jmqi.UnmappableCharacterAction
指定要對編碼和解碼的不可翻譯資料採取的動作。 值可以是 REPORT、 REPLACE 或 IGNORE。

com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement
在編碼作業中無法對映字元時，設定或取得要套用的取代位元組。在解碼作業中使用預設 Java 取代字串。

為了避免 Java 字元與原生位元組表示法之間的混淆，您應該指定 com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement 作為十進位數，代表原生字集中的取代位元組。

例如，如果原生字集是 ASCII 型 (例如 ISO-8859-1)，則 ?作為原生位元組的十進位值為 63，而如果原生字集是 EBCDIC，則為 111。

註: 請注意，如果 MQMD 或 MQMessage 物件已設定 **unmappableAction** 或 **unMappableReplacement** 欄位，則這些欄位的值優先於 Java 系統內容。 必要的話，這容許針對每一個訊息置換 Java 系統內容指定的值。

相關概念

第 294 頁的『IBM MQ classes for Java 中的字串轉換』
IBM MQ classes for Java 直接使用 CharsetEncoders 和 CharsetDecoders 來進行字串轉換。 字串轉換的預設行為可以配置兩個系統內容。 包含無法對映之字元的訊息處理可以透過 com.ibm.mq.MQMD 來配置。

撰寫 IBM MQ classes for JMS/Jakarta Messaging 應用程式

在簡要介紹 JMS 模型之後，本節提供如何撰寫 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 應用程式的詳細指引。

關於這項作業

JM 3.0 V 9.3.0 V 9.3.0 從 IBM MQ 9.3.0 開始，支援 Jakarta Messaging 3.0 開發新的應用程式。 IBM MQ 9.3.0 繼續支援 JMS 2.0 現有的應用程式。 不支援在同一應用程式中同時使用 Jakarta Messaging 3.0 API 和 JMS 2.0 API。 如需相關資訊，請參閱 使用 IBM MQ 類別進行 JMS/Jakarta 傳訊。

相關概念

IBM MQ classes for Jakarta Messaging: 概觀

JMS 和 *Jakarta Messaging* 模型

JMS 和 Jakarta Messaging 模型定義一組介面，供 Java 應用程式用來執行傳訊作業。 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 都是傳訊提供者。 它們定義 JMS 和 Jakarta Messaging 物件如何與 IBM MQ 概念相關。 JMS 和 Jakarta Messaging 規格預期某些 JMS 和 Jakarta Messaging 物件是受管理物件。

JMS 2.0 IBM MQ 8.0 新增了對 JMS 標準 JMS 2.0 版本的支援，該標準引進了簡化的 API，同時保留 JMS 1.1 中的典型 API。

V 9.3.0 JM 3.0 V 9.3.0 從 IBM MQ 9.3.0 開始，支援 Jakarta Messaging 3.0 開發新的應用程式。 IBM MQ 9.3.0 繼續支援 JMS 2.0 現有的應用程式。 不支援在同一應用程式中同時使用 JMS 2.0 API 和 Jakarta Messaging 3.0 API。

註: 對於 Jakarta Messaging 3.0， JMS 規格的控制項會從 Oracle 移至 Java Community Process。 不過，Oracle 會保留 "javax" 名稱的控制權，該名稱用於尚未移至 Java Community Process 的其他 Java 技術中。因此，雖然 Jakarta Messaging 3.0 在功能上等同於 JMS 2.0，但在命名方面仍有一些差異:

- 3.0 版的正式名稱是 Jakarta Messaging，而不是 Java Message Service。
- 套件及常數名稱會以 `jakarta` 而非 `javax` 作為字首。 例如，在 JMS 2.0 中，傳訊提供者的起始連線是 `javax.jms.Connection` 物件，在 Jakarta Messaging 3.0 中是 `jakarta.jms.Connection` 物件。

JMS 2.0 javax.jms 套件定義 JMS 介面， JMS 提供者會針對特定的傳訊產品實作這些介面。 IBM MQ classes for JMS 是一個 JMS 提供者，用於實作 IBM MQ 的 JMS 介面。

JM 3.0 jakarta.jms 套件定義 Jakarta Messaging 介面， Jakarta Messaging 提供者會針對特定的傳訊產品實作這些介面。 IBM MQ classes for Jakarta Messaging 是一個 Jakarta Messaging 提供者，用於實作 IBM MQ 的 Jakarta Messaging 介面。

因為 JMS 和 Jakarta Messaging 有許多共同之處，本主題中對 JMS 的進一步參照可以視為同時參照兩者。必要時會強調顯示任何差異。

簡化的 API

JMS 2.0 引進了簡化的 API，同時保留 JMS 1.1 中的網域特定及網域無關介面。 簡化的 API 減少了傳送及接收訊息所需的物件數，且包含下列介面:

ConnectionFactory
ConnectionFactory 是 JMS 用戶端用於建立連線的受管理物件。 此介面也用於典型 API 中。

JMS 環境定義
此物件結合典型 API 的 Connection 及 Session 物件。 您可以使用重複的基礎連線，從其他 JMSContext 物件建立 JMSContext 物件。

JMS 生產者
JMSProducer 由 JMSContext 建立，且用於將訊息傳送至佇列或主題。 JMSProducer 物件會導致建立傳送訊息所需的物件。

JMS 消費者
JMSConsumer 由 JMSContext 建立，且用於接收來自主題或佇列的訊息。

簡化的 API 具有許多影響:

- JMSContext 物件會一律自動啟動基礎連線。

- 使用訊息的 getBody 方法，JMSProducers 及 JMSConsumers 現在可以直接處理訊息內文，而無需取得整個訊息物件。
- 在傳送「內文」（訊息內容）之前，使用方法鏈在 JMSProducer 物件上設定訊息內容。 JMSProducer 將處理傳送訊息所需的所有物件的建立作業。 使用 JMS 2.0，可以設定內容，且傳送的訊息如下：

```
context.createProducer().
setProperty("foo", "bar").
setTimeToLive(10000).
setDeliveryMode(NON_PERSISTENT).
setDisableMessageTimestamp(true).
send(dataQueue, body);
```

JMS 2.0 也引進了共用訂閱，其中可以在多個消費者之間共用訊息。 所有 JMS 1.1 訂閱視為非共用訂閱閱。

典型 API

下列清單彙總典型 API 的主要 JMS 介面：

目的地
目的地是應用程式傳送訊息的地方，及（或）應用程式從中接收訊息的來源。

ConnectionFactory
ConnectionFactory 物件封裝連線的一組配置內容。 應用程式使用 Connection Factory 來建立連線。

連線
Connection 物件封裝傳訊伺服器的應用程式的作用中連線。 應用程式使用連線建立階段作業。

Session
階段作業是用於傳送及接收訊息的單一執行緒環境定義。 應用程式使用階段作業建立訊息、訊息產生者及訊息消費者。 階段作業已交易或未交易。

訊息
Message 物件封裝應用程式傳送或接收的訊息。

MessageProducer
應用程式使用訊息產生者將訊息傳送至目的地。

MessageConsumer
應用程式使用訊息消費者接收已傳送至目的地的訊息。

第 120 頁的圖 9 顯示這些物件及其關係。

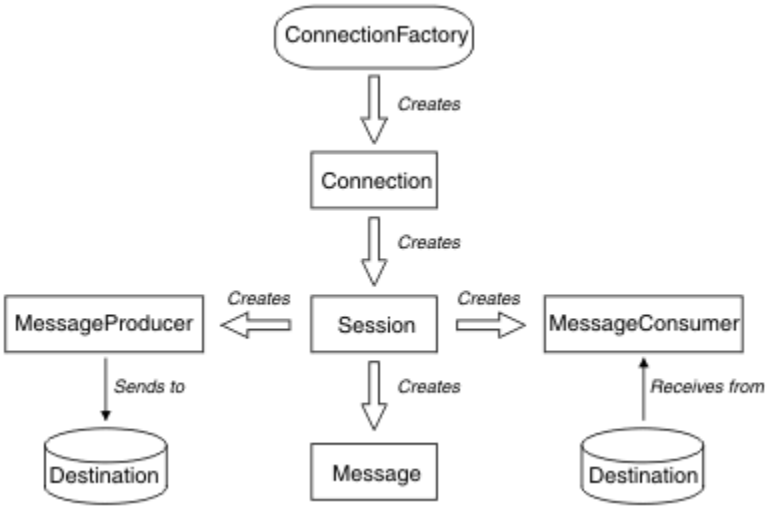


圖 *9: JMS* 物件及其關係

Destination、ConnectionFactory 或 Connection 物件可以同時供多執行緒應用程式的不同執行緒使用，但 Session、MessageProducer 或 MessageConsumer 物件不能同時供不同的執行緒使用。 確保不同時使用 Session、MessageProducer 或 MessageConsumer 的最簡單的方法是為每個執行緒建立個別 Session 物件。

傳訊網域

JMS 支援兩種傳訊樣式:

- 點對點傳訊
- 發佈/訂閱傳訊

這些傳訊樣式亦稱為傳訊網域，您可以將這兩種傳訊樣式結合在一個應用程式中。 在點對點網域中，目的地是佇列；在發佈/訂閱網域中，目的地是主題。

在 JMS 1.1 之前的 JMS 版本中，點對點網域的程式設計會使用一組介面和方法，而發佈/訂閱網域的程式設計則會使用另一組介面和方法。 兩組類似，但彼此獨立。 從 JMS 1.1 開始，您可以使用支援這兩種傳訊網域的一組共用介面及方法。 共用介面提供了每個傳訊網域的網域無關視圖。 第 121 頁的表 15 列出 JMS 網域無關介面及其對應的網域專用介面。

表 *15: JMS* 網域無關及網域專用介面

網域無關介面	**點對點網域的網域專用介面**	**發佈/訂閱網域的網域專用介面**
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
連線	QueueConnection	TopicConnection
目的地	佇列	主題
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver QueueBrowser	TopicSubscriber

JMS 2.0 IBM MQ classes for JMS 2.0 同時支援舊版 JMS 1.1 網域特定介面及 JMS 2.0 的簡化 API。 因此， IBM MQ classes for JMS 2.0 可用來維護現有應用程式，包括在現有應用程式中開發新功能。

JM 3.0 IBM MQ classes for Jakarta Messaging 3.0 支援相同介面的 Jakarta Messaging 版本，建議用於新的應用程式開發。

在 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 中， JMS 物件以下列方式與 IBM MQ 概念相關:

- Connection 物件包含的內容衍生自 Connection Factory 內容，用於建立連線。 這些內容控制應用程式連接至佇列管理程式的方式。 這些內容的範例是佇列管理程式的名稱，對於在用戶端模式中連接至佇列管理程式的應用程式，則是佇列管理程式正在其中執行的系統的主機名稱或 IP 位址。
- Session 物件封裝 IBM MQ 連線控點，因此定義階段作業的交易式範圍。
- MessageProducer 物件及 MessageConsumer 物件各自封裝 IBM MQ 物件控點。

使用 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 時，會套用 IBM MQ 的所有一般規則。 請特別注意，應用程式可以將訊息傳送至遠端佇列，但它只能接收來自應用程式所連接佇列管理程式擁有的佇列的訊息。

JMS 規格預期 ConnectionFactory 及 Destination 物件是受管理物件。 管理者可在中央儲存庫中建立及維護受管理物件，且 JMS 應用程式使用「Java 命名和目錄介面 (JNDI)」擷取這些物件。

在 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 中，「目的地」介面的實作是「佇列」和「主題」的抽象超類別，因此「目的地」實例是「佇列」物件或「主題」物件。 網域無關介面將佇列或主題視為目的地。 MessageProducer 或 MessageConsumer 物件的傳訊網域由目的地是佇列還是主題來判斷。

因此，在 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 中，下列類型的物件可以是受管理物件:

- ConnectionFactory

- QueueConnectionFactory
- TopicConnectionFactory
- 佇列
- 主題
- XAConnectionFactory
- XAQueueConnectionFactory
- XATopicConnectionFactory

相關概念

IBM MQ Java 語言介面

第 175 頁的『建立及配置 Connection Factory 和目的地』

IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 應用程式可以建立 Connection Factory 及目的地，方法是從「Java 命名和目錄介面 (JNDI)」名稱空間、使用 IBM JMS 延伸，或使用 IBM MQ JMS 延伸，將它們擷取為受管理物件。 應用程式也可以使用 IBM JMS 延伸或 IBM MQ JMS 延伸來設定 Connection Factory 和目的地的內容。

JMS 訊息

JMS 訊息由標頭、內容及內文組成。 JMS 定義五種類型的訊息內文。

JMS 訊息由下列組件組成:

標頭
所有訊息都支援相同的標頭欄位集。 標頭欄位包含用戶端和提供者用來識別及遞送訊息的值。

內容
每一個訊息都包含內建機能，以支援應用程式定義的內容值。 內容提供有效率的機制來過濾應用程式定義的訊息。

內文
JMS 定義五種訊息內文類型，涵蓋目前使用中的大部分傳訊樣式:

串流
Java 初始值的串流。 它會依序填入及讀取。

對映
一組名稱/值配對，其中名稱是字串，值是 Java 初始類型。 可以依名稱循序或隨機存取項目。 未定義項目的順序。

文字
包含 java.lang.String 的訊息。

物件
包含可序列化 Java 物件的訊息

位元組
未解譯位元組的串流。 此訊息類型用於文字編碼內文以符合現有訊息格式。

JMSCorrelationID 標頭欄位用來鏈結一則訊息與另一則訊息。 它通常會鏈結回覆訊息及其要求訊息。 JMSCorrelationID 可以保留提供者特定的訊息 ID、應用程式特定的字串或提供者原生位元組 [] 值。

JMS 中的訊息選取器

訊息可以包含應用程式定義的內容值。 應用程式可以使用訊息選取器來具有 JMS 提供者過濾器訊息。

訊息包含內建機能，可支援應用程式定義的內容值。 實際上，這提供一種機制，可將應用程式特定的標頭欄位新增至訊息。 內容可讓應用程式 (使用訊息選取器) 讓 JMS 提供者使用應用程式特定的準則，代表其選取或過濾訊息。 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 ID 的規則。
- 內容值可以是 Boolean、byte、short、int、long、float、double 及 String。
- JMSX 和 JMS_ 名稱字首是保留的。

在傳送訊息之前會先設定內容值。 當用戶端接收訊息時，訊息內容是唯讀的。 如果用戶端此時嘗試設定內容，則會擲出 MessageNotWriteableException 。 如果呼叫 clearProperties ，現在可以讀取及寫入內容。

內容值可能會複製訊息內文中的值。 JMS 不會為可能變成內容的內容定義原則。 不過，應用程式開發人員必須注意， JMS 提供者處理訊息內文中的資料可能比訊息內容中的資料更有效率。 為了取得最佳效能，只有在應用程式需要自訂訊息標頭時，它們才必須使用訊息內容。 執行此動作的主要原因是支援自訂訊息選擇。

JMS 訊息選取器可讓用戶端使用訊息標頭來指定其感興趣的訊息。 只會遞送標頭符合選取元的訊息。

訊息選取元無法參照訊息內文值。

當訊息標頭欄位和內容值取代選取器中的對應 ID 時，當選取器評估為 true 時，訊息選取器會比對訊息。

訊息選取元是一個「字串」，其語法是以 SQL92 條件式表示式語法的子集為基礎。 在優先順序層次內，訊息選取器的評估順序是從左到右。 您可以使用括弧來變更此順序。 這裡會以大寫撰寫預先定義的選取元文字及運算子名稱; 不過，它們不區分大小寫。

訊息選取器的內容

訊息選取元可以包含:

- 文字值
  - 字串文字以引號括住。 雙引號代表引號。 範例為 'literal' 及 'literal'。 與 Java 字串文字一樣，這些文字使用 Unicode 字元編碼。
  - 確切數值文字是不含小數點的數值，例如 57、-957 及 +62。 支援 Java 長整數範圍內的數字。
  - 近似數值文字是科學記號表示法中的數值 (例如 7E3 或 -57.9E2) ，或具有小數的數值 (例如 7.、 -95.7 或 +6.2)。 支援 Java 倍精準數範圍內的數字。
  - 布林文字 TRUE 和 FALSE。
- ID:
  - ID 是無限制長度的 Java 字母及 Java 數字序列，其中第一個必須是 Java 字母。 字母是 Character.isJava 方法傳回 true 的任何字元。 這包括 _ 和 $。 字母或數字是 Character.isJavaLetterOrDigit 方法傳回 true 的任何字元。
  - ID 不能是名稱 NULL、TRUE 或 FALSE。
  - ID 不能是 NOT、AND、OR、之間、LIKE、IN 或 IS。
  - ID 是標頭欄位參照或內容參照。
  - ID 區分大小寫。
  - 訊息標頭欄位參照限制為:
    - JMSDeliveryMode
    - JMSPriority
    - JMSMessageID
    - JMSTimestamp
    - JMSCorrelationID
    - JMSType

    JMSMessageID、JMSTimestamp、 JMSCorrelationID 和 JMSType 值可以是空值，如果是，則會被視為空值。
  - 任何以 JMSX 開頭的名稱都是 JMS 定義的內容名稱。
  - 任何以 JMS_ 開頭的名稱都是提供者特定的內容名稱。
  - 任何不是以 JMS 開頭的名稱都是應用程式特定的內容名稱。 如果訊息中沒有內容的參照，則其值為 NULL。 如果它確實存在，則其值是對應的內容值。
- 空格與針對 Java 定義的空格相同: 空格、水平定位點、換頁和行終止符號。
- 表示式:
  - 選取元是條件式表示式。 評估為 true 相符項的選取元; 評估為 false 或不明的選取元不相符。
  - 算術表示式是由本身、算術運算、ID (具有被視為數值文字的值) 及數值文字所組成。

  - 條件式表示式由其本身、比較運算及邏輯運算組成。
- 支援標準括弧 () ，以設定評估表示式的順序。
- 依優先順序的邏輯運算子: NOT、AND、OR。
- Comparison operators: =, >, >=, <, <=, <> (not equal).
  - 只能比較相同類型的值。 一個例外是比較確切數值和近似數值是有效的。 (所需的類型轉換是由 Java 數值促銷活動的規則所定義。) 如果嘗試比較不同的類型，則選取元一律為 false。
  - String and Boolean comparison is restricted to = and <>. Two strings are equal only if they contain the same sequence of characters.
- 依優先順序的算術運算子:
  - + ， -單元。
  - *、/、乘法及除法。
  - +、-、加法及減法。
  - 不支援空值上的算術運算。 如果嘗試它們，則完整選取元一律為 false。
  - 算術運算必須使用 Java 數值促銷。
- arithmetic-expr1 [NOT] 介於 arithmetic-expr2 與 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 ETWEEN 運算的任何表示式為 NULL ，則運算的值為 false。 如果 NOT ETWEEN 運算的任何表示式是 NULL ，則運算的值為 true。
- identifier [NOT] IN (string-literal1， string-literal2， ...) 比較運算子，其中 ID 具有字串或空值。
  - 國家 IN ('UK '，' US '，' France ') 對於' UK '是 true ，對於' Peru ' 是 false。 它相當於表示式 (Country = 'UK ') OR (Country = 'US') OR (Country = 'France ')。
  - 國家或地區 NOT IN ('UK '，' US '，' France ') 若為' UK '則為 false ，若為' Peru ' 則為 true。 它相當於表示式 NOT ((Country = 'UK ') OR (Country = 'US') OR (Country = 'France '))。
  - 如果 IN 或 NOT IN 作業的 ID 是空值，則作業的值不明。
- identifier [NOT] LIKE pattern-value [ESCAPE ESCAPE-character] 比較運算子，其中 identifier 具有字串值。 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)。 所有其他角色都代表自己。 選用的跳出字元是單一字串文字，具有用來跳出 pattern-value 中 _ 及% 的特殊意義的字元。
  - phone LIKE '12%3' is true for 123 and 12993 and false for 1234.
  - word LIKE 'l_se' 是 true (代表 "lose") ，而 false (代表 "loose")。
  - 對於 "_foo" ， LIKE '_ %' ESCAPE' \ ' 為 true ，而對於 "bar" ，則為 false。
  - phone NOT LIKE '12%3' 對於 123 和 12993 是 false ，對於 1234 是 true。
  - 如果 LIKE 或 NOT LIKE 作業的 ID 是 NULL ，則作業的值不明。
- ID IS NULL 比較運算子會測試空值標頭欄位值或遺漏內容值。
  - prop_name IS NULL。
- ID IS NOT NULL 比較運算子會測試是否存在非空值標頭欄位值或內容值。
  - prop_name IS NOT NULL。

訊息選取器範例

下列訊息選取器會選取訊息類型為 car、顏色為藍色且加權大於 2500 磅的訊息:

```
"JMSType = 'car' AND color = 'blue' AND weight > 2500"
```

空值內容值

如前述清單中所述，內容值可以是空值。 包含空值的選取元表示式的評估由 SQL 92 空值語意定義。 下列清單提供這些語意的簡要說明:

- SQL 將空值視為不明。
- 具有不明值的比較或算術一律會產生不明值。
- IS NULL 運算子將不明值轉換為 TRUE 值。
- IS NOT NULL 運算子將不明值轉換為 FALSE 值。

JMSMessageID 和 JMSCorrelationID 的特殊行為

從基於 JMSMessageID 或 JMSCorrelationID 的佇列中選取訊息時， JMS 的 IBM MQ 類別包含最佳化。

如果應用程式指定表單的選取器:

JMSMessageID= 'ID:*message_id*'

其中 *message_id* 是包含標準 IBM MQ 訊息 ID 的字串，然後 JMS 的 IBM MQ 類別使用 **MatchOption** MQMO_MATCH_MSG_ID 來取得具有指定訊息 ID 的訊息。

例如，若要從佇列取得訊息 ID 為 414D51207061756C745639314C545320C57C1A5F25ECE602 的訊息，應用程式應該使用下列訊息選取器:

```
JMSMessageID='ID:414D51207061756C745639314C545320C57C1A5F25ECE602'
```

同樣地，如果應用程式指定具有下列格式的選取器:

JMSCorrelationID = 'ID:*correlation_id*'

其中 *correlation_id* 是包含標準 IBM MQ 相關性 ID 的字串， JMS 的 IBM MQ 類別會使用 **MatchOption** MQMO_MATCH_CORREL_ID 從佇列中取得具有指定相關性 ID 的訊息。

在下列範例中，會使用訊息選取器來取得相關性 ID 為 414D51207061756C745639314C545320846E5B5F25B1CC02 的訊息:

```
JMSCorrelationID='ID:414D51207061756C745639314C545320846E5B5F25B1CC02'
```

如果訊息選取器針對 *message_id* 或 *correlation_id* 包含全部為零的值，則它會符合佇列上的任何訊息。 例如，如果應用程式正在使用選取器:

```
JMSMessageID='ID:000000000000000000000000000000000000000000000000'
```

則會將佇列上的任何訊息視為相符，並傳回給應用程式。

如需 MQMO_MATCH_MSG_ID 和 MQMO_MATCH_CORREL_ID **MatchOptions** 的相關資訊，請參閱 MatchOptions (MQLONG)。

限制

雖然 SQL 支援固定十進位比較及算術，但 JMS 訊息選取元不支援。 這就是為何精確數值文字限制為沒有小數的文字。 這也是為何會有數字以十進位作為近似數值的替代表示法。

不支援 SQL 註解。

將 *JMS* 訊息對映至 *IBM MQ* 訊息

IBM MQ 訊息由「訊息描述子」、選用的 MQRFH2 標頭及內文組成。 JMS 訊息的內容部分對映並部分複製到 IBM MQ 訊息。

本主題說明如何將本節第一部分中說明的 JMS 訊息結構對映至 IBM MQ 訊息。 想要在 JMS 與傳統 IBM MQ 應用程式之間傳輸訊息的程式設計師會感興趣。 對於想要操作在兩個 JMS 應用程式之間傳輸的訊息 (例如，在 IBM Integration Bus 實作中) 的人員也有興趣。

如果應用程式使用與分配管理系統的即時連線，則此區段不適用。 當應用程式使用即時連線時，會直接透過 TCP/IP 執行所有通訊; 不涉及任何 IBM MQ 佇列或訊息。

IBM MQ 訊息由三個元件組成:

- IBM MQ 訊息描述子 (MQMD)
- IBM MQ MQRFH2 標頭
- 訊息內文。

MQRFH2 是選用項目，其包含在送出訊息中由 JMS 目的地類別中的 TARGCLIENT 旗標控管。 您可以使用 IBM MQ JMS 管理工具來設定此旗標。 因為 MQRFH2 包含 JMS 特定資訊，當傳送端知道接收端目的地是 JMS 應用程式時，請一律將它包含在訊息中。 通常，將訊息直接傳送至非 JMS 應用程式時，會省略 MQRFH2 。 這是因為此類應用程式在其 IBM MQ 訊息中不預期 MQRFH2 。

如果送入訊息沒有 MQRFH2 標頭，依預設，訊息的 JMSReplyTo 標頭欄位所衍生的「佇列」或「主題」物件會設定此旗標，因此傳送至佇列或主題的回覆訊息也不會有 MQRFH2 標頭。 只有在原始訊息具有 MQRFH2 標頭時，您才可以將 Connection Factory 的 TARGCLIENTMATCHING 內容設為 NO ，來關閉將 MQRFH2 標頭併入回覆訊息的這個行為。

第 126 頁的圖 10 顯示如何將 JMS 訊息的結構轉換為 IBM MQ 訊息，然後重新轉換:

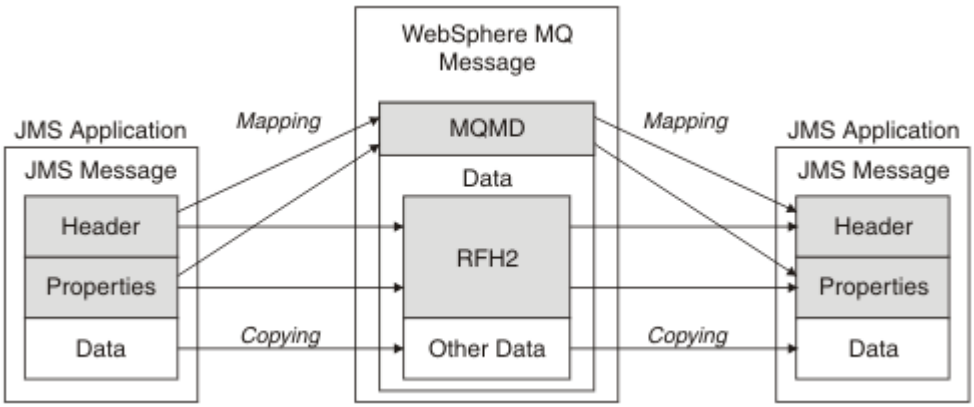


圖 *10:* 如何使用 *MQRFH2* 標頭在 *JMS* 與 *IBM MQ* 之間轉換訊息

結構有兩種轉換方式:

對映
如果 MQMD 包括相等於 JMS 欄位的欄位，則 JMS 欄位會對映至 MQMD 欄位。 其他 MQMD 欄位公開為 JMS 內容，因為 JMS 應用程式在與非 JMS 應用程式通訊時可能需要取得或設定這些欄位。

正在複製
如果沒有 MQMD 對等項目，則會傳遞 JMS 標頭欄位或內容，可能將其轉換為 MQRFH2 內的欄位。

MQRFH2 標頭和 *JMS*
此主題集合說明 MQRFH 第 2 版標頭，其中包含與訊息內容相關聯的 JMS 特定資料。 MQRFH 第 2 版標頭是可延伸的，也可以包含與 JMS 沒有直接關聯的其他資訊。 不過，本節只涵蓋 JMS 所使用的內容。如需完整說明，請參閱 MQRFH2 -規則和格式化標頭 2。

標頭有兩個部分，固定部分和可變部分。

固定部分
固定部分在 標準 IBM MQ 標頭型樣上建模，由下列欄位組成:

StrucId (MQCHAR4)
結構 ID。

必須是 MQRFH_STRUC_ID (值: "RFH ") (起始值)。

MQRFH_STRUC_ID_ARRAY (值: "R"、 "F"、 "H"、 " ") 也已定義。

版本 (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，包括 NameValue 資料欄位。

設為 StrucLength 的值必須是 4 的倍數 (NameValue 資料欄位中的資料可以填補空格字元以達到此目的)。

編碼 (MQLONG)
資料編碼。

MQRFH2 之後的訊息部分 (下一個標頭，或此標頭之後的訊息資料) 中任何數值資料的編碼。

CodedCharSetId (MQLONG)
編碼字集 ID。

MQRFH2 之後的訊息部分 (下一個標頭，或此標頭之後的訊息資料) 中任何字元資料的表示法。

格式 (MQCHAR8)
格式名稱。

MQRFH2 之後的訊息部分的格式名稱。

旗標 (MQLONG)
旗子

MQRFH_NO_FLAGS = 0。 未設定旗標。

NameValueCCSID (MQLONG)
此標頭所包含之 NameValue 資料字串的編碼字集 ID (CCSID)。 NameValue 資料可以編碼為字集，不同於標頭中包含的其他字串 (StrucID 和格式)。

如果 NameValueCCSID 是 2 位元組 Unicode CCSID (1200、13488 或 17584) ，則 Unicode 的位元組順序與 MQRFH2 中數值欄位的位元組排序相同。 (例如， Version、 StrucLength 及 NameValueCCSID 本身。)

表 *16: NameValueCCSID* 欄位的可能值

CCSID	**意義**
1200	UTF-16: 支援最新 Unicode 版本
13488	UTF-16:Unicode 2.0 版子集
17584	UTF-16Unicode 版本 3.0 子集 (包括歐元符號)
1208	UTF-8，支援最新 Unicode 版本

變數部分
可變部分跟隨在固定部分之後。 變數部分包含變數數目的 MQRFH2 資料夾。 每一個資料夾都包含可變數目的元素或內容。 資料夾群組相關內容。 JMS 所建立的 MQRFH2 標頭可以包含下列任何資料夾:

mcd 資料夾

mcd 包含說明訊息格式的內容。 例如，訊息服務網域 Msd 內容將 JMS 訊息識別為 JMSTextMessage、JMSBytesMessage、JMSStreamMessage、JMSMapMessage、JMSObjectMessage 或空值。

mcd 資料夾一律存在於包含 MQRFH2 的 JMS 訊息中。

它一律存在於包含從 IBM Integration Bus 傳送之 MQRFH2 的訊息中。 它會說明訊息的網域、格式、類型及訊息集。

表 *17: mcd* 內容名稱、同義字、資料類型及資料夾

內容同義字	**內容名稱**	**資料類型**	**資料夾**
	mcd.Msd	string	<mcd><Msd>*messageDomain*</Msd></mcd>

表 *17: mcd* 內容名稱、同義字、資料類型及資料夾 *(繼續)*			
內容同義字	**內容名稱**	**資料類型**	**資料夾**
	mcd.Set	string	<mcd><Set>*messageDomain*</Set></mcd>
	mcd.Type	string	<mcd><Type>*messageDomain*</Type></mcd>
	mcd.Fmt	string	<mcd><Fmt>*messageDomain*</Fmt></mcd>

請勿在 mcd 資料夾中新增您自己的內容。

jms 資料夾

jms 包含 JMS 標頭欄位，以及無法在 MQMD 中完整表達的 JMSX 內容。 jms 資料夾一律存在於 JMS MQRFH2 中。

usr 資料夾

usr 包含與訊息相關聯的應用程式定義 JMS 內容。 僅當應用程式已設定應用程式定義的內容時，才會呈現 usr 資料夾。

mqext 資料夾

mqext 包含下列類型的內容:

- 僅供 WebSphere Application Server 使用的內容。
- 與訊息延遲遞送相關的內容。

如果應用程式至少設定了其中一個 IBM 定義的內容，或已使用遞送延遲，則會存在該資料夾。

表 *18: mqext* 內容名稱、同義字、資料類型及資料夾			
內容同義字	**內容名稱**	**資料類型**	**資料夾**
JMSArmCorrelator	mqext.Arm	string	<mqext><Arm>*armCorrelator*</Arm></mqext>
JMSRMCorrelator	mqext.Wrm	string	<mqext><Wrm>*wrmCorrelator*</Wrm></mqext>
JMSDeliveryTime	mqext.Dlt	i8	<mqext><Dlt>*DeliveryTime*</Dlt></mqext>
JMSDeliveryDelay	mqext.Dly	i8	<mqext><Dly>*DeliveryTime*</Dly></mqext>

請勿在 mqext 資料夾中新增您自己的內容。

mqps 資料夾

mqps 包含僅供 IBM MQ 發佈/訂閱使用的內容。 只有在應用程式已設定至少其中一個整合的發佈/訂閱內容時，該資料夾才存在。

表 *19: mqps* 內容名稱、同義字、資料類型及資料夾			
內容同義字	**內容名稱**	**資料類型**	**資料夾**
MQTopicString	mqps.Top	string	<mqps><Top>*topicString*</Top></mqps>
MQSubUserData	mqps.Sud	string	<mqps><Sud>*subscriberUserData...*</Sud></mqps>

表 *19: mqps* 內容名稱、同義字、資料類型及資料夾 *(*繼續*)*			
內容同義字	**內容名稱**	**資料類型**	**資料夾**
MQIsRetained	mqps.Ret	boolean	<mqps><Ret>*isRetained*</Ret></mqps>
MQPubOptions	mqps.Pub	i8	<mqps><Pub>*publicationOptions*</Pub></mqps>
MQPubLevel	mqps.Pbl	i8	<mqps><Pbl>*publicationLevel*</Pbl></mqps>
MQPubTime	mqpse.Pts	string	<mqps><Pts>*publicationTime*</Pts></mqps>
MQPubSeqNum	mqpse.Seq	i8	<mqps><Seq>*publicationSequenceNumber*</Seq></mqps>
MQPubStrIntData	mqpse.Sid	string	<mqps><Sid>*publicationData*</Sid></mqps>
MQPubFormat	mqpse.Pfmt	i8	<mqps><Pfmt>*messageFormat*</Pfmt></mqps>

請勿在 mqps 資料夾中新增您自己的內容。

第 129 頁的表 20 顯示內容名稱的完整清單。

表 *20: MQRFH2* 資料夾和 *JMS* 使用的內容				
JMS 欄位名稱	**Java type**	**MQRFH2 資料夾名稱**	**內容名稱**	**類型/值**
JMSDestination	目的地	jms	dst	字串
JMSExpiration	長整數	jms	匯出	i8
JMSPriority	整數	jms	PRI	i4
JMSDeliveryMode	整數	jms	德爾夫	i4
JMSCorrelationID	字串	jms	CID	字串
JMSReplyTo	目的地	jms	Rto	字串
JMSTimestamp	長整數	jms	Tms	i8
JMSType	字串	MCD	類型、集、Fmt	字串
JMSXGroupID	字串	jms	GID	字串
JMSXGroupSeq	整數	jms	順序	i4
xxx (使用者定義)	任意	USR	xxx	全部
		MCD	MSD	jms_none jms_text jms_bytes jms_map jms_stream jms_object

NameValue 長度 (MQLONG)
緊接在此長度欄位後面的 NameValue 資料字串的長度 (以位元組為單位) (不包括其自己的長度)。

NameValue 資料 (MQCHARn)

單一字串，其長度 (以位元組為單位) 由之前的 NameValue 長度欄位提供。 它包含一個資料夾，其中包含一系列內容。 每一個內容都是名稱/類型/值三元組，包含在名稱為資料夾名稱的 XML 元素中，如下所示:

```
<foldername>
triplet1 triplet2 .....  tripletn </foldername>
```

結尾 </foldername> 標籤後面可以接著空格作為填補字元。 每一個三元組都使用類似 XML 的語法進行編碼:

```
<name dt='datatype'>value</name>
```

dt='datatype' 元素是選用的，對於許多內容會省略，因為資料類型是預先定義的。 如果包含它，則必須在 dt= 標籤之前包含一個以上空格字元。

name
是內容的名稱; 請參閱 第 129 頁的表 20。

datatype
在摺疊之後，必須符合 第 130 頁的表 21 中列出的其中一個資料類型。

value
是要傳送之值的字串表示法，使用 第 130 頁的表 21 中的定義。

使用下列語法來編碼空值:

```
<name dt='datatype' xsi:nil='true'></name>
```

請勿使用 xsi:nil='false'。

表 *21:* 內容資料類型

資料類型	**確立**
字串	任何字元序列，但 < 和 & 除外
布林值	字元 0 或 1 (0 = false ， 1 = true)
bin.hex	代表八位元組的十六進位數字
i1	數字，以數字 0..9 表示，具有選用符號 (無分數或指數)。 必須位於 -128 至 127 (含) 的範圍內
i2	數字，以數字 0..9 表示，具有選用符號 (無分數或指數)。 必須在 -32768 至 32767 (含) 的範圍內
i4	數字，以數字 0..9 表示，具有選用符號 (無分數或指數)。 必須位於 -2147483648 至 2147483647 (含) 的範圍內
i8	數字，以數字 0..9 表示，具有選用符號 (無分數或指數)。 必須位於 -9223372036854775808 至 9223372036854750807 (含) 範圍內
整數	數字，以數字 0..9 表示，具有選用符號 (無分數或指數)。 必須位於與 i8 相同的範圍內。 如果寄件者不想將特定精準度與內容相關聯，則可以使用此選項來取代其中一個 i * 類型
r4	Floating point number, magnitude <= 3.40282347E+38,>= 1.175E-37 expressed using digits 0..9, optional sign, optional fractional digits, optional exponent
r8	Floating point number, magnitude <= 1.7976931348623E+308,>= 2.225E-307 expressed using digits 0..9, optional sign, optional fractional digits, optional exponent

字串值可以包含空格。 您必須在字串值中使用下列 ESC 序列:

- & 代表 & 字元
- < 代表 < 字元

您可以使用下列 ESC 序列，但它們不是必要項目:

- > 代表 > 字元
- ' 代表 ' 字元
- " 代表 " 字元

JMS 欄位及具有對應 *MQMD* 欄位的內容
這些表格顯示相等於 JMS 標頭欄位、 JMS 內容及 JMS 提供者特定內容的 MQMD 欄位。

第 131 頁的表 22 列出 JMS 標頭欄位， 第 131 頁的表 23 列出直接對映至 MQMD 欄位的 JMS 內容。 第 131 頁的表 24 列出提供者特定的內容及其對映至的 MQMD 欄位。

表 *22: JMS* 標頭欄位對映至 *MQMD* 欄位

JMS 標頭欄位	**Java type**	**MQMD 欄位**	**合併類型**
JMSDeliveryMode	整數	持續性	MQLONG
JMSExpiration	長整數	期限	MQLONG
JMSPriority	整數	優先順序	MQLONG
JMSMessageID	字串	MsgID	MQBYTE24
JMSTimestamp	長整數	PutDate PutTime	MQCHAR8 MQCHAR8
JMSCorrelationID	字串	CorrelId	MQBYTE24

表 *23: JMS* 內容對映至 *MQMD* 欄位

JMS 內容	**Java type**	**MQMD 欄位**	**合併類型**
JMSXUserID	字串	UserIdentifier	MQCHAR12
JMSXAppID	字串	PutApplName	MQCHAR28
JMSXDeliveryCount	整數	BackoutCount	MQLONG
JMSXGroupID	字串	GroupId	MQBYTE24
JMSXGroupSeq	整數	MsgSeqNumber	MQLONG

表 *24: JMS* 提供者特定的內容對映至 *MQMD* 欄位

JMS 提供者特定的內容	**Java type**	**MQMD 欄位**	**合併類型**
JMS_IBM_Report_Exception	整數	報告	MQLONG
JMS_IBM_Report_Expiration	整數	報告	MQLONG
JMS_IBM_Report_COA	整數	報告	MQLONG
JMS_IBM_Report_COD	整數	報告	MQLONG
JMS_IBM_Report_PAN	整數	報告	MQLONG

表 *24: JMS* 提供者特定的內容對映至 *MQMD* 欄位 *(*繼續*)*			
JMS 提供者特定的內容	**Java type**	**MQMD 欄位**	**合併類型**
JMS_IBM_Report_NAN	整數	報告	MQLONG
JMS_IBM_Report_Pass_Msg_ID	整數	報告	MQLONG
JMS_IBM_Report_Pass_Correl_ID	整數	報告	MQLONG
JMS_IBM_Report_Discard_Msg	整數	報告	MQLONG
JMS_IBM_MsgType	整數	MsgType	MQLONG
JMS_IBM_Feedback	整數	意見	MQLONG
JMS_IBM_Format	字串	格式 第 132 頁的『1』	MQCHAR8
JMS_IBM_PutAppl 類型	整數	PutApplType	MQLONG
JMS_IBM_Encoding	整數	編碼	MQLONG
JMS_IBM_Character_Set	字串	CodedCharacterSetId 第 132 頁的『2』	MQLONG
JMS_IBM_PutDate	字串	PutDate	MQCHAR8
JMS_IBM_PutTime	字串	PutTime	MQCHAR8
JMS_IBM_Last_Msg_In_Group	布林值	MsgFlags	MQLONG

註:

1. JMS_IBM_Format 代表訊息內文的格式。 這可以由應用程式設定訊息的 JMS_IBM_Format 內容來定義 (請注意，有 8 個字元限制) ，也可以預設為適用於 JMS 訊息類型之訊息內文的 IBM MQ 格式。 只有在訊息未包含 RFH 或 RFH2 區段時， JMS_IBM_Format 才會對映至 MQMD 格式欄位。 在一般訊息中，它會對映至緊接在訊息內文前面的 RFH2 的「格式」欄位。
2. JMS_IBM_Character_Set 內容值是一個「字串」值，包含與數值 CodedCharacterSetId 值相等的 Java 字集。 MQMD 欄位 CodedCharacterSetId 是一個數值，其中包含 JMS_IBM_Character_Set 內容所指定 Java 字集字串的對等項目。

將 *JMS* 欄位對映至 *IBM MQ* 欄位 *(*送出訊息*)*
這些表格顯示 JMS 標頭和內容欄位在 send () 或 publish () 時如何對映至 MQMD 和 MQRFH2 欄位。

第 132 頁的表 25 顯示如何在 send () 或 publish () 時間將 JMS 標頭欄位對映至 MQMD/RFH2 欄位。 第 133 頁的表 26 顯示如何在 send () 或 publish () 時間將 JMS 內容對映至 MQMD/RFH2 欄位。 第 133 頁的表 27 顯示 JMS 提供者特定內容如何在 send () 或 publish () 時間對映至 MQMD 欄位。

對於標示由訊息物件設定的欄位，傳輸的值是緊接在 send () 或 publish () 作業之前保留在 JMS 訊息中的值。 作業會維持 JMS 訊息中的值不變。

對於標示為由傳送方法設定的欄位，當執行 send () 或 publish () 時，會指派一個值 (系統不處理 JMS 訊息中保留的任何值)。 JMS 訊息中的值會更新，以顯示所使用的值。

標示為「僅接收」的欄位不會傳輸，在訊息中由 send () 或 publish () 維持不變。

表 *25:* 送出訊息欄位對映			
JMS 標頭欄位名稱	**用於傳輸的 MQMD 欄位**	**標頭**	**設定依據**
JMSDestination		MQRFH2	傳送方法
JMSDeliveryMode	持續性	MQRFH2	傳送方法
JMSExpiration	期限	MQRFH2	傳送方法

表 *25:* 送出訊息欄位對映 *(繼續)*			
JMS 標頭欄位名稱	**用於傳輸的 MQMD 欄位**	**標頭**	**設定依據**
JMSPriority	優先順序	MQRFH2	傳送方法
JMSMessageID	MsgID		傳送方法
JMSTimestamp	PutDate/PutTime		傳送方法
JMSCorrelationID	CorrelId	MQRFH2	訊息物件
JMSReplyTo	ReplyToQ/ReplyTo 佇列管理程式	MQRFH2	訊息物件
JMSType		MQRFH2	訊息物件
JMSRedelivered			僅接收

註:

1. MQMD 欄位 CodedCharacterSetId 是一個數值，其中包含 JMS_IBM_Character_Set 內容所指定 Java 字集字串的對等項目。

表 *26:* 送出訊息 *JMS* 內容對映			
JMS 內容名稱	**用於傳輸的 MQMD 欄位**	**標頭**	**設定依據**
JMSXUserID	UserIdentifier		傳送方法
JMSXAppID	PutApplName		傳送方法
JMSXDeliveryCount			僅接收
JMSXGroupID	GroupId	MQRFH2	訊息物件
JMSXGroupSeq	MsgSeqNumber	MQRFH2	訊息物件

註:

這些內容由 JMS 規格定義為唯讀，並由 JMS 提供者設定 (在某些情況下選擇性地)。

在 IBM MQ classes for JMS 中，應用程式可以置換其中兩個內容。 如果要這麼做，請設定下列內容，確定已適當配置目的地:

1. 將內容 `WMQConstants.WMQ_MQMD_MESSAGE_CONTEXT` 設為 `WMQConstants.WMQ_MDCTX_SET_ALL_CONTEXT`。
2. 將內容 `WMQConstants.WMQ_MQMD_WRITE_ENABLED` 設為 `true`。

應用程式可以置換下列內容:

JMSXAppID
可以透過在訊息上設定內容 `WMQConstants.JMS_IBM_MQMD_PUTAPPLNAME` 來置換此內容-值應該是 Java 字串。

JMSXGroupID
可以透過在訊息上設定內容 `WMQConstants.JMS_IBM_MQMD_GROUPID` 來置換此內容-值應該是位元組陣列。

表 *27:* 送出訊息 *JMS* 提供者特定的內容對映			
JMS 提供者特定的內容名稱	**用於傳輸的 MQMD 欄位**	**標頭**	**設定依據**
JMS_IBM_Report_Exception	報告		訊息物件
JMS_IBM_Report_Expiration	報告		訊息物件
JMS_IBM_Report_COA/COD	報告		訊息物件

表 27: 送出訊息 JMS 提供者特定的內容對映 (繼續)			
JMS 提供者特定的內容名稱	**用於傳輸的 MQMD 欄位**	**標頭**	**設定依據**
JMS_IBM_Report_NAN/PAN	報告		訊息物件
JMS_IBM_Report_Pass_Msg_ID	報告		訊息物件
JMS_IBM_Report_Pass_Correl_ID	報告		訊息物件
JMS_IBM_Report_Discard_Msg	報告		訊息物件
JMS_IBM_MsgType	MsgType		訊息物件
JMS_IBM_Feedback	意見		訊息物件
JMS_IBM_Format	格式		訊息物件
JMS_IBM_PutAppl 類型	PutApplType		傳送方法
JMS_IBM_Encoding	編碼		訊息物件
JMS_IBM_Character_Set	CodedCharacterSetId		訊息物件
JMS_IBM_PutDate	PutDate		傳送方法
JMS_IBM_PutTime	PutTime		傳送方法
JMS_IBM_Last_Msg_In_Group	MsgFlags		訊息物件

對映 *send ()* 或 *publish ()* 的 *JMS* 標頭欄位
這些附註與 send () 或 publish () 的 JMS 欄位對映相關。

JMSDestination 至 MQRFH2
這會儲存為字串，將目的地物件的顯著性質序列化，以便接收端 JMS 可以重新建構相等的目的地物件。 MQRFH2 欄位編碼為 URI (如需 URI 表示法的詳細資料，請參閱 第 190 頁的『統一資源識別碼 (URI)』)。

JMSReplyTo 至 MQMD.ReplyToQ、 ReplyTo 佇列管理程式、 MQRFH2
佇列名稱會複製到 MQMD.ReplyToQ 欄位，佇列管理程式名稱會複製到 ReplyTo 佇列管理程式欄位。 目的地延伸資訊 (保留在目的地物件中的其他有用詳細資料) 會複製到 MQRFH2 欄位。 MQRFH2 欄位編碼為 URI (如需 URI 表示法的詳細資料，請參閱 第 190 頁的『統一資源識別碼 (URI)』)。

JMSDeliveryMode 至 MQMD.Persistence
JMSDeliveryMode 值由 send () 或 publish () Method 或 MessageProducer 設定，除非「目的地物件」置換它。 JMSDeliveryMode 值對映至 MQMD.Persistence 欄位如下:

- JMS 值 PERSISTENT 相當於 MQPER_PERSISTENT
- JMS 值 NON_PERSISTENT 相當於 MQPER_NOT_PERSISTENT

如果 MQQueue 持續性內容未設為 WMQConstants.WMQ_PER_QDEF，遞送模式值也會編碼在 MQRFH2 中。

與 MQMD.Expiry， MQRFH2
JMSExpiration 會儲存到期時間 (現行時間與存活時間的總和) ，而 MQMD 會儲存存活時間。 此外，JMSExpiration 是以毫秒為單位，但為 MQMD.Expiry 以十分之一秒為單位。

- 如果 send () 方法設定無限制的存活時間，則為 MQMD.Expiry 設為 MQEI_UNLIMITED ，且不會在 MQRFH2 中編碼任何 JMSExpiration。
- 如果 send () 方法設定的存活時間小於 214748364.7 秒 (大約 7 年) ，則存活時間會儲存在 MQMD.Expiry 及有效期限 (毫秒) 在 MQRFH2 中編碼為 i8 值。
- 如果 send () 方法設定的存活時間大於 214748364.7 秒，則為 MQMD.Expiry 設為 MQEI_UNLIMITED。 實際有效期限 (毫秒) 在 MQRFH2 中編碼為 i8 值。

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 設為非預設值，則優先順序層次也會編碼在 MQRFH2 中。

MQMD.MessageID 中的 JMSMessageID
從 JMS 傳送的所有訊息都具有由 IBM MQ 指派的唯一訊息 ID。 指派的值會在 MQMD.MessageId 欄位會傳回給 JMSMessageID 欄位中的應用程式。 IBM MQ messageId 是 24 位元組二進位值，而 JMSMessageID 是字串。 JMSMessageID 由轉換成一連串 48 個十六進位字元的二進位 messageId 值組成，字首為字元 ID:。 JMS 提供可設為停用產生訊息 ID 的提示。 系統會忽略此提示，並在所有情況下指派唯一 ID。 在 send () 之前設定給 JMSMessageID 欄位的任何值都會被改寫。

如果您確實需要指定 MQMD.MessageID，您可以使用 第 208 頁的『從 IBM MQ classes for JMS 應用程式讀取及寫入訊息描述子』中說明的其中一個 IBM MQ JMS 延伸來執行此動作。

JMSTimestamp 至 MQRFH2
在傳送期間，會根據 JVM 的時鐘來設定 JMSTimestamp 欄位。 此值設定為 MQRFH2。 在 send () 之前設定給 JMSTimestamp 欄位的任何值都會被改寫。 另請參閱 JMS_IBM_PutDate 和 JMS_IBM_PutTime 內容。

JMSType 至 MQRFH2
此字串會設為 MQRFH2 mcd.Type 欄位。 如果它是 URI 格式，也可能會影響 mcd.Set 和 mcd.Fmt 欄位。

JMSCorrelationID 至 MQMD.CorrelId， MQRFH2
JMSCorrelationID 可以保留下列其中一項:

提供者特定的訊息 ID
這是來自先前傳送或接收之訊息的訊息 ID，因此應該是字首為 `ID:` 的 48 個小寫十六進位數字的字串。會移除字首，其餘字元會轉換成二進位，然後將它們設為 MQMD.CorrelId 欄位。

提供者原生位元組 [] 值
該值會複製到 MQMD.CorrelId 欄位-以空值填補，或在必要時截斷為 24 個位元組。 MQRFH2 中未編碼任何 CorrelId 值。

應用程式特定的字串
該值會複製到 MQRFH2。 字串的前 24 個位元組 (UTF8 格式) 會寫入 MQMD.CorrelID。

對映 *JMS* 內容欄位
這些附註參照 IBM MQ 訊息中 JMS 內容欄位的對映。

JMSXUserID (來自 MQMD UserIdentifier)
JMSXUserID 是在從傳送呼叫傳回時設定。

來自 MQMD PutAppl 名稱的 JMSXAppID
從傳送呼叫返回時設定 JSMXAppID 。

JMSXGroupID 至 MQRFH2 (點對點)
對於點對點訊息， JMSXGroupID 會複製到 MQMD GroupID 欄位。 如果 JMSXGroupID 以字首 ID 開頭，則會轉換成二進位。 否則，它會編碼為 UTF8 字串。 必要的話，會填補或截斷該值，長度為 24 個位元組。 已設定 MQMF_MSG_IN_GROUP 旗標。

JMSXGroupID 至 MQRFH2 (發佈/訂閱)
對於發佈/訂閱訊息， JMSXGroupID 會以字串形式複製到 MQRFH2 。

JMSXGroupSeq MQMD MsgSeq 號碼 (點對點)
對於點對點訊息， JMSXGroupSeq 會複製到 MQMD MsgSeq 號碼欄位。 已設定 MQMF_MSG_IN_GROUP 旗標。

JMSXGroupSeq MQMD MsgSeq 號碼 (發佈/訂閱)
對於發佈/訂閱訊息， JMSXGroupSeq 會以 i4 形式複製到 MQRFH2 。

對映 *JMS* 提供者特定的欄位
下列附註參照 JMS 提供者特定欄位至 IBM MQ 訊息的對映。

JMS_IBM_Report_*XXX* 至 MQMD 報告
JMS 應用程式可以使用下列 JMS_IBM_Report_*XXX* 內容來設定 MQMD 報告選項。 單一 MQMD 對映至數個 JMS_IBM_Report_*XXX* 內容。

JMS_IBM_Report_*XXX* 常數位於 com.ibm.msg.client.jakarta.wmq.WMQConstants 或 com.ibm.msg.client.wmq.WMQConstants 中。

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
MQ Ro_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

MQRO 值位於 com.ibm.mq.constants.CMQC 中。

JMS_IBM_MsgType 至 MQMD MsgType
值直接對映至 MQMD MsgType。 如果應用程式未設定明確值 JMS_IBM_MsgType，則會使用預設值。 此預設值決定如下:

- 如果 JMSReplyTo 設為 IBM MQ 佇列目的地，則 MSGType 會設為值 MQMT_REQUEST
- 如果未設定 JMSReplyTo ，或設為 IBM MQ 佇列目的地以外的任何其他值，則 MsgType 會設為 MQMT_DATAGRAM 值

JMS_IBM_Feedback 至 MQMD Feedback
值直接對映至 MQMD 意見回饋。

JMS_IBM_Format 至 MQMD 格式
值直接對映至 MQMD 格式。

JMS_IBM_Encoding 至 MQMD 編碼
如果設定，此內容會置換「目的地佇列」或「主題」的數值編碼。

JMS_IBM_Character_Set 至 MQMD CodedCharacterSetId
如果設定，則此內容會置換「目的地佇列」或「主題」的編碼字集內容。

來自 MQMD PutDate 的 JMS_IBM_PutDate
此內容的值在傳送期間直接從 MQMD 中的 PutDate 欄位設定。 在改寫傳送之前設定給 JMS_IBM_PutDate 內容的任何值。 此欄位是 8 個字元的字串，採用 IBM MQ 日期格式 YYYYMMDD。 此內容可與 JMS_IBM_PutTime 內容搭配使用，以決定根據佇列管理程式放置訊息的時間。

來自 MQMD PutTime 的 JMS_IBM_PutTime
在傳送期間，會直接從 MQMD 的 PutTime 欄位設定此內容的值。 在改寫傳送之前設定給 JMS_IBM_PutTime 內容的任何值。 此欄位是 8 個字元的字串，採用 IBM MQ 時間格式 HHMMSSTH。 此內容可與 JMS_IBM_PutDate 內容搭配使用，以決定根據佇列管理程式放置訊息的時間。

JMS_IBM_Last_Msg_In_Group 至 MQMD MsgFlags
對於點對點傳訊，此布林值對映至 MQMD MsgFlags 欄位中的 MQMF_LAST_MSG_IN_GROUP 旗標。 通常與 JMSXGroupID 及 JMSXGroupSeq 內容搭配使用，以向舊式 IBM MQ 應用程式指出此訊息是群組中的最後一則。 發佈/訂閱傳訊會忽略這個內容。

將 *IBM MQ* 欄位對映至 *JMS* 欄位 (送入訊息)
這些表格顯示如何在 get () 或 receive () 時間將 JMS 標頭及內容欄位對映至 MQMD 及 MQRFH2 欄位。

第 137 頁的表 28 顯示如何在 get () 或 receive () 時間將 JMS 標頭欄位對映至 MQMD/MQRFH2 欄位。 第 137 頁的表 29 顯示如何在 get () 或 receive () 時間將 JMS 內容欄位對映至 MQMD/MQRFH2 欄位。 第 138 頁的表 30 顯示如何對映 JMS 提供者特定的內容。

表 *28:* 送入訊息 *JMS* 標頭欄位對映

JMS 標頭欄位名稱	**MQMD 欄位擷取自**	**MQRFH2 欄位擷取自**
JMSDestination		jms.Dst 或 mqps.Top 第 137 頁的『1』
JMSDeliveryMode	持續性第 137 頁的『2』	jms.Dlv 第 137 頁的『2』
JMSExpiration		jms.Exp
JMSPriority	優先順序	
JMSMessageID	MsgID	
JMSTimestamp	PutDate 第 137 頁的『2』 PutTime 第 137 頁的『2』	jms.Tms 第 137 頁的『2』
JMSCorrelationID	CorrelId 第 137 頁的『2』	jms.Cid 第 137 頁的『2』
JMSReplyTo	回覆目的地佇列第 137 頁的『2』 回覆目的地佇列管理程式第 137 頁的『2』	jms.Rto 第 137 頁的『2』
JMSType		mcd.Type, mcd.Set, mcd.Fmt
JMSRedelivered	BackoutCount	

註:

1. 如果同時設定 jms.Dst 和 mqps.Top ，則會使用 jms.Dst 中的值。
2. 對於可以從 MQRFH2 或 MQMD 擷取值的內容，如果兩者都可用，則會使用 MQRFH2 中的設定。
3. JMS_IBM_Character_Set 內容值是一個「字串」值，包含與數值 CodedCharacterSetId 值相等的 Java 字集。

表 *29:* 送入訊息內容對映

JMS 內容名稱	**MQMD 欄位擷取自**	**MQRFH2 欄位擷取自**
JMSXUserID	UserIdentifier	
JMSXAppID	PutApplName	
JMSXDeliveryCount	BackoutCount	
JMSXGroupID	GroupId 第 138 頁的『1』	jms.Gid 第 138 頁的『1』

表 *29:* 送入訊息內容對映 *(繼續)*		
JMS 內容名稱	**MQMD 欄位擷取自**	**MQRFH2 欄位擷取自**
JMSXGroupSeq	MsgSeq 號碼 第 138 頁的『1』	jms.Seq 第 138 頁的『1』

註:

1. 對於可以從 MQRFH2 或 MQMD 擷取值的內容，如果兩者都可用，則會使用 MQRFH2 中的設定。 只有在設定 MQMF_MSG_IN_GROUP 或 MQMF_LAST_MSG_IN_GROUP 訊息旗標時，才會從 MQMD 值設定內容。

表 *30:* 送入訊息提供者特定的 *JMS* 內容對映		
JMS 內容名稱	**MQMD 欄位擷取自**	**MQRFH2 欄位擷取自**
JMS_IBM_Report_Exception	報告	
JMS_IBM_Report_Expiration	報告	
JMS_IBM_Report_COA	報告	
JMS_IBM_Report_COD	報告	
JMS_IBM_Report_PAN	報告	
JMS_IBM_Report_NAN	報告	
JMS_IBM_Report_ Pass_Msg_ID	報告	
JMS_IBM_Report_Pass_Correl_ID	報告	
JMS_IBM_Report_Discard_Msg	報告	
JMS_IBM_MsgType	MsgType	
JMS_IBM_Feedback	意見	
JMS_IBM_Format	格式	
JMS_IBM_PutAppl 類型	PutApplType	
JMS_IBM_Encoding 第 138 頁的『1』	編碼	
JMS_IBM_Character_Set 第 138 頁的『1』	CodedCharacterSetId	
JMS_IBM_PutDate	PutDate	
JMS_IBM_PutTime	PutTime	
JMS_IBM_Last_Msg_In_Group	MsgFlags	

1. 只有在送入訊息是「位元組訊息」時才會設定。

在 *JMS* 應用程式與傳統 *IBM MQ* 應用程式之間交換訊息
本主題說明當 JMS 應用程式與無法處理 MQRFH2 標頭的傳統 IBM MQ 應用程式交換訊息時所發生的情況。

第 139 頁的圖 11 顯示對映。

管理者透過將目的地的 TARGCLIENT 內容設為 *MQ*，指出 JMS 應用程式正在與傳統 IBM MQ 應用程式進行通訊。 這表示不會產生 MQRFH2 標頭。 如果未這樣做，接收端應用程式必須能夠處理 MQRFH2 標頭。

以傳統 IBM MQ 應用程式為目標，從 JMS 至 MQMD 的對映與以 JMS 應用程式為目標，從 JMS 至 MQMD 的對映相同。 如果 IBM MQ classes for JMS 收到 IBM MQ 訊息，且 MQMD *Format* 欄位設為 MQFMT_RFH2 以外的任何值，則會從非 JMS 應用程式接收資料。 如果格式為 MQFMT_STRING ，則會以 JMS 文字訊息接收訊息。 否則，會將它當作 JMS 位元組訊息來接收。 因為沒有 MQRFH2，所以只能還原在 MQMD 中傳輸的那些 JMS 內容。

如果 IBM MQ classes for JMS 收到沒有 MQRFH2 標頭的訊息，依預設，從訊息的 JMSReplyTo 標頭欄位衍生之「佇列」或「主題」物件的 TARGCLIENT 內容會設為 MQ 。 這表示傳送至佇列或主題的回覆訊息也沒有 MQRFH2 標頭。 只有在原始訊息具有 MQRFH2 標頭時，您才可以將 Connection Factory 的 TARGCLIENTMATCHING 內容設為 NO ，來關閉將 MQRFH2 標頭併入回覆訊息的這個行為。

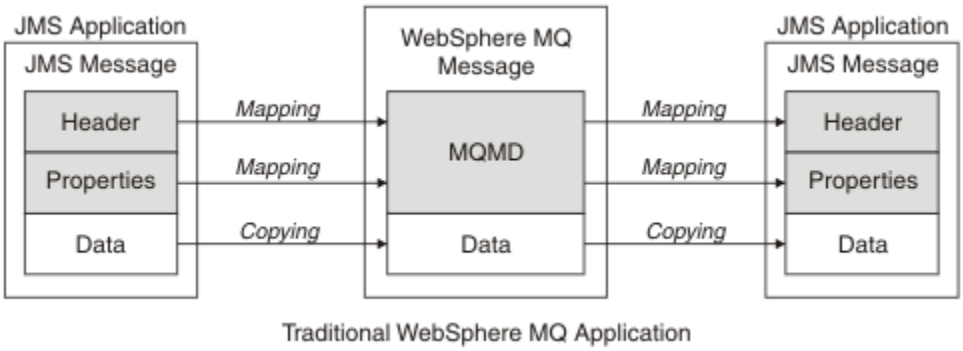


圖 *11:* 如何將 *JMS* 訊息轉換成沒有 *MQRFH2* 標頭的 *IBM MQ* 訊息

JMS 訊息內文
本主題包含訊息內文本身的編碼相關資訊。 編碼取決於 JMS 訊息的類型。

ObjectMessage
ObjectMessage 是由「 Java 執行時期」以一般方式序列化的物件。

TextMessage
TextMessage 是編碼字串。 對於送出訊息，字串會以目的地物件所提供的字集編碼。 這預設為 UTF8 編碼 (UTF8 編碼從訊息的第一個字元開始; 開頭沒有長度欄位)。 不過，可以指定 IBM MQ classes for JMS 支援的任何其他字集。 這類字集主要是在您將訊息傳送至非 JMS 應用程式時使用。

如果字集是雙位元組集 (包括 UTF16) ，目的地物件的整數編碼規格會決定位元組的順序。

送入訊息會使用訊息本身中指定的字集及編碼來解譯。 這些規格位於最後一個 IBM MQ 標頭中 (如果沒有標頭，則為 MQMD)。 對於 JMS 訊息，最後一個標頭通常是 MQRFH2。

BytesMessage
依預設， BytesMessage 是 JMS 1.0.2 規格及相關聯 Java 文件所定義的一連串位元組。

對於應用程式本身所組合的送出訊息，目的地物件的編碼內容可用來置換訊息中包含的整數和浮點欄位的編碼。 例如，您可以要求以 S/390 而非 IEEE 格式儲存浮點值。

送入訊息會使用訊息本身中指定的數值編碼來解譯。 此規格位於最後一個 IBM MQ 標頭中 (如果沒有標頭，則為 MQMD)。 對於 JMS 訊息，最後一個標頭通常是 MQRFH2。

如果接收到 BytesMessage ，且未修改即重新傳送，則會以位元組方式傳輸其內文。 目的地物件的編碼內容對主體沒有影響。 唯一可以在 BytesMessage 中明確傳送的字串型實體是 UTF8 字串。 這是以 Java UTF8 格式編碼，並以 2 位元組長度欄位開頭。 目的地物件的字集內容對送出的 BytesMessage 編碼沒有影響。 送入 IBM MQ 訊息中的字集值不會影響將該訊息解譯為 JMS BytesMessage。

非 Java 應用程式不太可能辨識 Java UTF8 編碼。 因此，若要讓 JMS 應用程式傳送包含文字資料的 BytesMessage ，應用程式本身必須將其字串轉換為位元組陣列，並將這些位元組陣列寫入 BytesMessage。

MapMessage
MapMessage 是包含 XML 名稱/類型/值三元組的字串，編碼為:

```
<map>
  <elt name="elementname1" dt="datatype1">value1</elt>
  <elt name="elementname2" dt="datatype2">value2</elt>
  ...
</map>
```

其中 `datatype` 是 第 130 頁的表 21 中列出的其中一個資料類型。 預設資料類型是 `string`，因此字串元素會省略屬性 `dt="string"` 。

用來編碼或解譯形成對映訊息主體之 XML 字串的字集，是根據套用至文字訊息的規則來決定。

早於 5.3 的 IBM MQ classes for JMS 版本會以下列格式對對映訊息內文進行編碼:

```
<map>
  <elementname1 dt="datatype1">value1</elementname1>
  <elementname2 dt="datatype2">value2</elementname2>
  ...
</map>
```

IBM MQ classes for JMS 5.3 以及更新版本可以解譯任一格式，但 5.3 之前的 IBM MQ classes for JMS 版本無法解譯現行格式。

如果應用程式需要將對映訊息傳送至另一個使用 5.3 之前的 IBM MQ classes for JMS 版本的應用程式，則傳送端應用程式必須呼叫 Connection Factory 方法 `setMapNameStyle(WMQConstants.WMQ_MAP_NAME_STYLE_COMPATIBLE)`，以指定以先前的格式傳送對映訊息。 依預設，會以現行格式傳送所有對映訊息。

StreamMessage
StreamMessage 類似於對映訊息，但沒有元素名稱:

```
<stream>
  <elt dt="datatype1">value1</elt>
  <elt dt="datatype2">value2</elt>
  ...
</stream>
```

其中 `datatype` 是 第 130 頁的表 21 中列出的其中一個資料類型。 預設資料類型是 `string`，因此字串元素會省略屬性 `dt="string"`。

用來編碼或解譯組成 StreamMessage 主體之 XML 字串的字集，是遵循適用於 TextMessage 的規則來決定。

MQRFH2.format 欄位設定如下:

MQFMT_NONE
針對 ObjectMessage、 BytesMessage 或沒有內文的訊息。

MQFMT_STRING
適用於 TextMessage、 StreamMessage 或 MapMessage。

JMS 訊息轉換
在傳送及接收訊息時，會執行 JMS 中的訊息資料轉換。 IBM MQ 會自動執行大部分資料轉換。 在 JMS 應用程式之間傳送訊息時，它會轉換文字及數值資料。 在 JMS 應用程式與 IBM MQ 應用程式之間交換 `JMSTextMessage` 時，會轉換文字。

如果您計劃執行更複雜的訊息交換，您會感興趣下列主題。 複式訊息交換包括:

- 在 IBM MQ 應用程式與 JMS 應用程式之間傳送非文字訊息。
- 以位元組格式交換文字資料。
- 轉換應用程式中的文字。

JMS 訊息資料

即使在兩個 JMS 應用程式之間，也必須進行資料轉換，才能在應用程式之間交換文字和數值資料。 必須對文字和數字的內部表示法進行編碼，以便可以在訊息中傳送它們。 編碼會強制決定如何代表數字和文字。 IBM MQ 管理 JMS 訊息中文字及數字的編碼，但 `JMSObjectMessage` 除外，請參閱 第 146 頁的『JMSObjectMessage』。 它使用三個訊息屬性。 這三個屬性分別是 `CodedCharacterSetId`、`Encoding` 和 `Format`。

這三個訊息屬性通常儲存在 JMS 訊息的 JMS 標頭 `MQRFH2` 欄位中。 如果訊息類型是 `MQ`，而不是 `JMS` 訊息類型，則屬性會儲存在訊息描述子 `MQMD` 中。 這些屬性用來轉換 JMS 訊息資料。 在 IBM MQ 訊息的訊息資料部分中傳送 JMS 訊息資料。

JMS 訊息內容

除非已在沒有 MQRFH2 的情況下傳送訊息，否則 JMS 訊息內容 (例如 JMS_IBM_CHARACTER_SET) 會在 JMS 訊息的 MQRFH2 標頭部分中交換。 在沒有 MQRFH2 的情況下，只能傳送 JMSTextMessage 和 JMSBytesMessage 。 如果 JMS 內容在訊息描述子 MQMD 中儲存為 IBM MQ 訊息內容，則會在 MQMD 轉換過程中進行轉換。 如果 JMS 內容儲存在 MQRFH2 中，則它會儲存在 MQRFH2.NameValueCCSID 指定的字集中。 當傳送或接收訊息時，訊息內容會與 JVM 中的內部表示法來回轉換。 轉換是與訊息描述子或 MQRFH2.NameValueCCSID 的字集之間的轉換。 數值資料會轉換為文字。

JMS 訊息轉換

下列主題包含範例及作業，如果您計劃交換需要轉換的更複雜訊息，則這些範例及作業非常有用。

JMS 訊息轉換方法
JMS 應用程式設計人員可以使用許多資料轉換方法。 這些方法並非互斥; 有些應用程式可能會使用這些方法的組合。 如果您的應用程式只交換文字，或只與其他 JMS 應用程式交換訊息，則通常不會考量資料轉換。 IBM MQ 會自動為您執行資料轉換。

您可以詢問一些關於如何處理訊息轉換的問題:

是否完全需要考慮訊息轉換?
在某些情況下 (例如 JMS 至 JMS 訊息傳送，以及與 IBM MQ 程式交換文字訊息) ， IBM MQ 會自動為您執行必要的轉換。 基於效能原因，您可能想要控制資料轉換，或者您可能正在交換具有預先定義格式的複雜訊息。 在這些情況下，您必須瞭解訊息轉換，並閱讀下列主題。

有甚麼轉變呢?
有四種主要類型的轉換，在下列各節中說明:

1. 第 141 頁的『JMS 用戶端資料轉換』
2. 第 142 頁的『應用程式資料轉換』
3. 第 142 頁的『佇列管理程式資料轉換』
4. 第 143 頁的『訊息通道資料轉換』

應在何處執行轉換?
第 143 頁的『選擇訊息轉換方法: receiver ma 之好』一節說明 "receiver 製造"的一般方法。 "接收端良好" 也適用於 JMS 資料轉換。

JMS 用戶端資料轉換

JMS 用戶端 [1] 資料轉換是將 JMS 訊息傳送至目的地時，將 Java 基本元素及物件轉換成位元組，並在收到時重新轉換回。 JMS 用戶端資料轉換使用 JMSMessage 類別的方法。 這些方法依 第 144 頁的表 31 中的 JMSMessage 類別類型列出。

對於 read、get、set 和 write 方法，會執行數字和文字的內部 JVM 表示法來回轉換。 當傳送訊息時，以及在已收到的訊息上呼叫任何 read 或 get 方法時，會執行轉換。

用來寫入或設定訊息內容的字碼頁和數值編碼定義為目的地的屬性。 可以透過管理方式變更目的地字碼頁及數值編碼。 應用程式也可以透過設定控制寫入或設定訊息內容的訊息內容，來置換目的地字碼頁及編碼。

如果您想要在將 JMSBytesMessage 訊息傳送至未定義為 Native 編碼的目的地時轉換數字編碼，則必須在傳送訊息之前設定訊息內容 JMS_IBM_ENCODING 。 如果您遵循 "接收端建立良好的" 型樣，或如果您在 JMS 應用程式之間交換訊息，則應用程式不需要設定 JMS_IBM_ENCODING。 在大部分情況下，您可以將 Encoding 內容保留為 Native。

對於 JMSStreamMessage、 JMSMapMessage 和 JMSTextMessage 訊息，會使用目的地的字集 ID 內容。 傳送時會忽略編碼，因為數字會以文字格式寫出。 如果要套用目的地字集內容，則在傳送訊息之前，JMS 用戶端應用程式不需要設定 JMS_IBM_CHARACTER_SET 。

若要取得訊息中的資料，應用程式會呼叫 JMS 訊息 read 或 get 方法。 這些方法會參照前一個訊息標頭中所定義的字碼頁和編碼，以正確建立 Java 基本元素和物件。

[1] "JMS 用戶端" 是指實作 JMS 介面的 IBM MQ classes for JMS ， 以用戶端或連結模式執行。

JMS 用戶端資料轉換符合大部分 JMS 應用程式的需求，這些應用程式在一個 JMS 用戶端與另一個用戶端之間交換訊息。 您不撰寫任何明確資料轉換的程式碼。 您不使用 `java.nio.charset.Charset` 類別，通常在將文字寫入檔案時使用。 `writeString` 和 `setString` 方法會為您執行轉換。

如需 JMS 用戶端資料轉換的詳細資料，請參閱 第 152 頁的『JMS 用戶端訊息轉換及編碼』。

應用程式資料轉換

JMS 用戶端應用程式可以使用 `java.nio.charset.Charset` 類別來執行明確字元資料轉換; 請參閱 第 145 頁的圖 14 和 第 145 頁的圖 15 中的範例。 字串資料會使用 `getBytes` 方法轉換成位元組，並以位元組形式傳送。 使用採用位元組陣列及 `Charset` 的 `String` 建構子，將位元組轉換回文字。 使用 `encode` 和 `decode` `Charset` 方法來轉換字元資料。 一般而言，訊息會以 `JMSBytesMessage` 來傳送或接收，因為 `JMSBytesMessage` 的訊息組件不包含應用程式所寫入資料以外的任何內容 [2]. 您也可以使用 `JMSStreamMessage`、 `JMSMapMessage` 或 `JMSObjectMessage` 來傳送及接收位元組。

沒有 Java 方法可編碼及解碼包含以不同編碼格式表示之數值資料的位元組。 數值資料會使用數值 `JMSMessage` 讀寫方法自動編碼及解碼。 read 和 write 方法使用訊息資料的 `JMS_IBM_ENCODING` 屬性值。

應用程式資料轉換的一般用途是 JMS 用戶端從非 JMS 應用程式傳送或接收格式化訊息。 格式化訊息包含依資料欄位長度組織的文字、數值及位元組資料。 除非非 JMS 應用程式已將訊息格式指定為 "MQSTR "，否則會將訊息建構為 `JMSBytesMessage`。 若要在 `JMSBytesMessage` 中接收格式化訊息資料，您必須呼叫一系列方法。 必須以欄位寫入訊息的相同順序來呼叫方法。 如果欄位是數值，您必須知道數值資料的編碼和長度。 如果任何欄位包含位元組或文字資料，您必須知道訊息中任何位元組資料的長度。 有兩種方法可以將格式化訊息轉換成易於使用的 Java 物件。

1. 建構對應於記錄的 Java 類別，以封裝讀取和寫入訊息。 記錄中資料的存取權是使用類別的 get 和 set 方法。
2. 延伸 `com.ibm.mq.headers` 類別，以建構對應於記錄的 Java 類別。 對類別中資料的存取權具有 `getStringValue(`*`fieldname`*`);` 格式的特定類型專用存取元

請參閱 第 159 頁的『與非 JMS 應用程式交換格式化記錄』。

佇列管理程式資料轉換

當 JMS 用戶端程式取得訊息時，佇列管理程式可以執行字碼頁轉換。 轉換與針對 C 程式執行的轉換相同。 C 程式會將 `MQGMO_CONVERT` 設為 `MQGET GetMsgOpts` 參數選項; 請參閱 第 145 頁的圖 13。 如果 `WMQ_RECEIVE_CONVERSION` 目的地內容設為 `WMQ_RECEIVE_CONVERSION_QMGR`，則佇列管理程式會對接收訊息的 JMS 用戶端程式執行轉換。 JMS 用戶端程式也可以設定目的地內容; 請參閱 第 142 頁的圖 12。

```
((MQDestination)destination).setIntProperty(
              WMQConstants.WMQ_RECEIVE_CONVERSION,
              WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

或

```
((MQDestination)destination).setReceiveConversion
              (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

圖 *12:* 啟用佇列管理程式資料轉換

與非 JMS 應用程式交換訊息時，佇列管理程式轉換的主要好處。 如果已定義訊息中的 `Format` 欄位，且目標字集或編碼與訊息不同，則在應用程式要求時，佇列管理程式會執行目標應用程式的資料轉換。 佇列管理程式會根據其中一個預先定義的 IBM MQ 訊息類型 (例如 CICS bridge 標頭 (`MQCIH`)) 來轉換格式化的訊息資料。 如果 `Format` 欄位是使用者定義的，佇列管理程式會以 `Format` 欄位中提供的名稱來尋找資料轉換結束程式。

[2] 一個例外: 使用 `writeUTF` 寫入的資料以 2 位元組長度欄位開頭

佇列管理程式資料轉換是用來對 "receiver 讓" 設計型樣發揮最佳效果。 傳送端 JMS 用戶端不需要執行轉換。 非 JMS 接收端程式依賴轉換結束程式，以確保以必要的字碼頁和編碼來遞送訊息。 使用傳送端 JMS 用戶端及非 JMS 接收端，此範例適用於 IBM MQ。

您可以使用資料轉換結束程式公用程式 **crtmqcvx** 來建立資料轉換結束程式，以讓佇列管理程式轉換您自己的記錄格式資料。 您可以建置自己的記錄格式，使用 `com.ibm.mq.headers` 作為 Java 類別來存取它，並使用您自己的轉換結束程式來轉換它。 在 z/OS 上，公用程式稱為 **CSQUCVX**，在 IBM i 上，稱為 **CVTMQMDTA**。 請參閱第 159 頁的『與非 JMS 應用程式交換格式化記錄』。

訊息通道資料轉換

IBM MQ 傳送端、伺服器、叢集接收端及叢集傳送端通道具有訊息轉換選項 CONVERT。 傳送訊息時，可以選擇性地轉換訊息的內容。 轉換在通道傳送端進行。 叢集接收端定義用來自動定義對應的叢集傳送端通道。

如果無法使用其他轉換形式，通常會使用訊息通道的資料轉換。

選擇訊息轉換方法: "receiver ma 之好"

在 IBM MQ 應用程式設計中，程式碼轉換的一般方法是 "receiver 製造"。 "接收端良好" 會減少訊息轉換的數目。 在訊息傳送期間，如果部分中間佇列管理程式上的訊息轉換失敗，也可以避免非預期通道錯誤的問題。 "receiver make 良好" 規則只有在有某種原因導致接收端無法良好時才會岔斷。 例如，接收端平台可能沒有正確的字集。

"接收端良好" 也是 JMS 用戶端應用程式的良好一般指引。 但在特定情況下，在來源處轉換至正確的字集可能會更有效率。 當傳送包含文字或數值類型的訊息時，必須從 JVM 內部表示法進行轉換。 轉換為接收端所需的字集 (如果接收端不是 JMS 用戶端) 可能會不需要非 JMS 接收端執行轉換。 如果收件者是 JMS 用戶端，則無論如何都會重新轉換，以解碼訊息資料並建立 Java 基本元素及物件。

JMS 用戶端應用程式與以語言 (例如 C) 撰寫的應用程式之間的差異在於 Java 必須執行資料轉換。 Java 應用程式必須將數字和文字從其內部表示法轉換成訊息中使用的編碼格式。

透過設定目的地或訊息內容，您可以設定 IBM MQ 用來編碼訊息中數字及文字的字集及編碼。 通常，您會將字集保留為 `1208` ，並將編碼保留為 `Native`。

IBM MQ 不會轉換位元組陣列。 如果要將字串和字元陣列編碼成位元組陣列，請使用 `java.nio.charset` 套件。 `Charset` 指定用來將字串或字元陣列轉換成位元組陣列的字集。 您也可以使用 `Charset`，將位元組陣列解碼成字串或字元陣列。 在編碼字串和字元陣列時，最好不要依賴 `java.nio.charset.Charset.defaultCodePage` 。 預設 `Charset` 通常是 `windows-1252` (在 Windows 上) 和 `UTF-8` (在 AIX and Linux 上)。 `windows-1252` 是單位元組字集，而 `UTF-8` 是多位元組字集。

與其他 JMS 應用程式交換訊息時，通常會將目的地字集及編碼內容保留為其預設值 `UTF-8` 及 `Native` 。 如果您要與 JMS 應用程式交換包含數字或文字的訊息，請選擇符合您目的的 `JMSTextMessage`、`JMSStreamMessage`、 `JMSMapMessage` 或 `JMSObjectMessage` 訊息類型之一。 沒有其他轉換作業可執行。

如果您要與使用記錄格式的非 JMS 應用程式交換訊息，則會更複雜。 除非整個記錄包含文字且可以作為 `JMSTextMessage` 傳送，否則您必須對應用程式中的文字進行編碼及解碼。 將目的地訊息類型設為 `MQ`，並使用 `JMSBytesMessage` 以避免 IBM MQ classes for JMS 將其他標頭及標記資訊新增至訊息資料。 使用 `JMSBytesMessage` 方法來寫入數字和位元組， `Charset` 類別會將文字明確轉換成位元組陣列。 許多因素可能會影響您選擇的字集:

- 效能: 您可以透過將文字轉換為在最大數目伺服器上使用的字集來減少轉換數目嗎?
- 一致性: 傳送相同字集的所有訊息。
- 豐富性: 哪些字集具有應用程式必須使用的所有字碼點?
- 簡單: 單位元組字集比可變長度及多位元組字集更容易使用。

請參閱 第 159 頁的『與非 JMS 應用程式交換格式化記錄』。 以取得轉換與非 JMS 應用程式交換之訊息的範例。

範例

訊息類型及轉換類型的表格

表 *31:* 訊息類型和轉換類型

	轉換類型			
訊息類型	**文字**	**數字**	**其他**	**無**
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			
JMSBytesMessage	readUTF writeUTF	readDouble readFloat readInt readLong readShort readUnsignedShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readUnsignedByte readBytes readChar writeByte writeBytes writeChar
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

```
gmo.Options = MQGMO_WAIT         /* wait for new messages            */
            | MQGMO_NO_SYNCPOINT /* no transaction                   */
            | MQGMO_CONVERT;     /* convert if necessary             */

     while (CompCode != MQCC_FAILED) {
     buflen = sizeof(buffer) - 1; /* buffer size available for GET   */
     memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
     memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
     md.Encoding       = MQENC_NATIVE;
     md.CodedCharSetId = MQCCSI_Q_MGR;

           MQGET(Hcon,          /* connection handle                */
           Hobj,                /* object handle                    */
           &md,                 /* message descriptor               */
           &gmo,                /* get message options              */
           buflen,              /* buffer length                    */
           buffer,              /* message buffer                   */
           &messlen,            /* message length                   */
           &CompCode,           /* completion code                  */
           &Reason);            /* reason code                      */
```

圖 *13:* 來自 *amqsget0.c* 的程式碼 *Snippet*

在 JMSBytesMessage 中傳送及接收文字

第 145 頁的圖 14 中的程式碼會傳送 BytesMessage 中的字串。 為了簡單起見，此範例會傳送 JMSTextMessage 更適合的單一字串。 若要接收包含混合類型的字串 (以位元組為單位) 訊息，您必須知道第 145 頁的圖 15 中稱為 *TEXT_LENGTH* 的字串長度 (以位元組為單位)。 即使是具有固定字元數的字串，位元組表示法的長度也可能較長。

```
BytesMessage bytes = session.createBytesMessage();
String codePage = CCSID.getCodepage(((MQDestination) destination)
                        .getIntProperty(WMQConstants.WMQ_CCSID));
bytes.writeBytes("In the destination code page".getBytes(codePage));
producer.send(bytes);
```

圖 *14:* 在 *JMSBytesMessage* 中傳送 *String*

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength())).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

圖 *15:* 從 *JMSBytesMessage* 接收 *String*

相關概念

JMS 用戶端訊息轉換及編碼
會列出您用來執行 JMS 用戶端訊息轉換及編碼的方法，以及每一種轉換類型的程式碼範例。

佇列管理程式資料轉換
佇列管理程式資料轉換一律適用於從 JMS 用戶端接收訊息的非 JMS 應用程式。 接收訊息的 JMS 用戶端也會使用佇列管理程式資料轉換 (選用)。

相關工作

與非 JMS 應用程式交換格式化記錄

遵循此作業中建議的步驟，以設計並建置資料轉換結束程式，以及可使用 JMSBytesMessage 與非 JMS 應用程式交換訊息的 JMS 用戶端應用程式。 不論是否呼叫資料轉換結束程式，都可以與非 JMS 應用程式交換格式化訊息。

相關參考

JMS 訊息類型及轉換
訊息類型的選擇會影響您的訊息轉換方法。 針對 JMS 訊息類型 JMSObjectMessage、JMSTextMessage、 JMSMapMessage、 JMSStreamMessage 及 JMSBytesMessage，會說明訊息轉換與訊息類型的互動。

JMS 訊息類型及轉換
訊息類型的選擇會影響您的訊息轉換方法。 針對 JMS 訊息類型 JMSObjectMessage、JMSTextMessage、 JMSMapMessage、 JMSStreamMessage 及 JMSBytesMessage，會說明訊息轉換與訊息類型的互動。

JMSObjectMessage

JMSObjectMessage 包含一個物件，以及它所參照的任何物件，由 JVM 序列化成位元組串流。 文字會序列化至 UTF-8，並限制為不超過 65534 個位元組的字串或字元陣列。 JMSObjectMessage 的優點是只要應用程式只使用物件的方法和屬性，就不會涉及任何資料轉換問題。 JMSObjectMessage 提供複式物件的資料轉換，而無需應用程式設計師考量如何對訊息中的物件進行編碼。 使用 JMSObjectMessage 的缺點是只能與其他 JMS 應用程式交換。 透過選擇其中一個其他 JMS 訊息類型，可以與非 JMS 應用程式交換 JMS 訊息。

第 148 頁的『傳送及接收 JMSObjectMessage』 顯示在訊息中交換的 String 物件。

JMS 用戶端應用程式只能在具有 JMS 樣式內文的訊息中接收 JMSObjectMessage 。 目的地必須指定 JMS 樣式主體。

JMSTextMessage

JMSTextMessage 包含單一字串。 傳送文字訊息時，文字 Format 會設為 "MQSTR ", WMQConstants.MQFMT_STRING。 文字的 CodedCharacterSetId 會設為為其目的地定義的編碼字集 ID。 IBM MQ 會將文字編碼成 CodedCharacterSetId 。 CodedCharacterSetId 和 Format 欄位在訊息描述子 MQMD 中設定，或在 MQRFH2 中的 JMS 欄位中設定。 如果訊息定義為具有 WMQ_MESSAGE_BODY_MQ 訊息內文樣式，或未指定內文樣式，但目標目的地是 WMQ_TARGET_DEST_MQ，則會設定訊息描述子欄位。 否則，訊息會有 JMS RFH2 ，且欄位會設定在 MQRFH2 的固定組件中。

應用程式可以置換定義給目的地的編碼字集 ID。 它必須將訊息內容 JMS_IBM_CHARACTER_SET 設為編碼字集 ID; 請參閱 第 148 頁的『傳送及接收 JMSTextmessage』中的範例。

當 JMS 用戶端呼叫時， consumer.receive 方法佇列管理程式轉換是選用的。 將目的地內容 WMQ_RECEIVE_CONVERSION 設為 WMQ_RECEIVE_CONVERSION_QMGR，即可啟用佇列管理程式轉換。 在將訊息傳送至 JMS 用戶端之前，佇列管理程式會從指定給訊息的 JMS_IBM_CHARACTER_SET 轉換文字訊息。 除非目的地具有不同的 WMQ_RECEIVE_CCSID，否則已轉換訊息的字集是 1208， UTF-8。 參照 JMSTextMessage 的訊息中的 CodedCharacterSetId 會更新為目標字集 ID。 getText 方法會將文字從目標字集解碼成 Unicode; 請參閱 第 148 頁的『傳送及接收 JMSTextmessage』中的範例。

JMSTextMessage 可以在沒有 JMS MQRFH2 標頭的 MQ 樣式訊息內文中傳送。 除非應用程式置換，否則目的地屬性 WMQ_MESSAGE_BODY 及 WMQ_TARGET_DEST 的值會決定訊息內文樣式。 應用程式可以透過呼叫 destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ) 或 destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ)來置換在目的地上設定的值。

如果您傳送具有 MQ 樣式主體的 JMSTextMessage ， 並將它傳送至 WMQ_MESSAGE_BODY 設為 WMQ_MESSAGE_BODY_MQ 的目的地，則無法從相同目的地接收它作為 JMSTextMessage 。 從 WMQ_MESSAGE_BODY 設為 WMQ_MESSAGE_BODY_MQ 的目的地接收的所有訊息，都會以 JMSBytesMessage 形式接收。 如果您嘗試以 JMSTextMessage 方式接收訊息，則會導致異常狀況 ClassCastException: com.ibm.jms.JMSBytesMessage cannot be cast to jakarta (or javax).jms.TextMessage。

註: JMS 用戶端不會轉換 JMSBytesMessage 中的文字。 用戶端只能以位元組陣列形式接收訊息中的文字。 如果啟用佇列管理程式轉換，則佇列管理程式會轉換文字，但 JMS 用戶端仍必須在 JMSBytesMessage 中以位元組陣列形式接收該文字。

通常最好使用 WMQ_TARGET_DEST 內容來控制 JMSTextMessage 是以 MQ 或 JMS 主體樣式傳送。 然後，您可以從 WMQ_TARGET_DEST 設為 WMQ_TARGET_DEST_MQ 或 WMQ_TARGET_DEST_JMS 的目的地接收訊息。 WMQ_TARGET_DEST 對接收端沒有影響。

JMSMapMessage 及 JMSStreamMessage

這兩種 JMS 訊息類型類似。 您可以使用以 DataInputStream 和 DataOutputStream 介面為基礎的方法，在訊息中讀取和寫入初始類型; 請參閱 第 151 頁的『訊息類型及轉換類型的表格』。 詳細資料在 第 152 頁的『JMS 用戶端訊息轉換及編碼』中說明。 會標記每一個基本元素; 請參閱 第 139 頁的『JMS 訊息內文』。

數值資料會讀取並寫入編碼為 XML 文字的訊息。 不會參照目的地內容 JMS_IBM_ENCODING。 文字資料的處理方式與 JMSTextMessage 中的文字相同。 如果您要查看 第 149 頁的圖 20 中的範例所建立的訊息內容，則所有訊息資料都會以 EBCDIC 格式傳送，因為它是以字集值 37 來傳送。

您可以在 JMSMapMessage 或 JMSStreamMessage 中傳送多個項目。

您可以從 JMSMapMessage 依名稱或從 JMSStreamMessage 依位置來擷取個別資料項目。 當使用訊息中儲存的 CodedCharacterSetId 值來呼叫 get 或 read 方法時，會將每一個項目解碼。 如果用來擷取項目的方法所傳回的類型與所傳送的類型不同，則會轉換類型。 如果無法轉換類型，則會擲出異常狀況。 如需詳細資料，請參閱 類別 JMSStreamMessage 。 第 149 頁的『在 JMSStreamMessage 和 JMSMapMessage 中傳送資料』 中的範例說明類型轉換，以及讓 JMSMapMessage 內容失序。

JMSMapMessage 和 JMSStreamMessage 的 MQRFH2.format 欄位設為 "MQSTR "。 如果目的地內容 WMQ_RECEIVE_CONVERSION 設為 WMQ_RECEIVE_CONVERSION_QMGR，則佇列管理程式會先轉換訊息資料，再將訊息資料傳送至 JMS 用戶端。 訊息的 MQRFH2.CodedCharacterSetId 是目的地的 WMQ_RECEIVE_CCSID 。 MQRFH2.Encoding 是 Native。 如果 WMQ_RECEIVE_CONVERSION 是 WMQ_RECEIVE_CONVERSION_CLIENT_MSG , 則 MQRFH2 的 CodedCharacterSetId 和 Encoding 是傳送者設定的值。

JMS 用戶端應用程式只能在具有 JMS 樣式主體的訊息中接收 JMSMapMessage 或 JMSStreamMessage , 以及從未指定 MQ 樣式主體的目的地接收。

JMSBytesMessage

JMSBytesMessage 可以包含多個初始類型。 您可以使用以 DataInputStream 和 DataOutputStream 介面為基礎的方法，在訊息中讀取和寫入初始類型; 請參閱 第 151 頁的『訊息類型及轉換類型的表格』。 詳細資料在 第 146 頁的『JMS 訊息類型及轉換』中說明。

訊息中數值資料的編碼是由在將數值資料寫入 JMSBytesMessage 之前所設定的 JMS_IBM_ENCODING 值所控制。 應用程式可以設定訊息內容 JMS_IBM_ENCODING 來置換定義給 JMSBytesMessage 的預設 Native 編碼。

文字資料可以在 UTF-8 中使用 readUTF 和 writeUTF 來讀取和寫入，或在 Unicode 中使用 readChar 和 writeChar 方法來讀取和寫入。 沒有使用 CodedCharacterSetId 的方法。 或者， JMS 用戶端可以使用 Charset 類別將文字編碼及解碼成位元組。 它會在 JVM 與訊息之間傳送位元組，而不 IBM MQ classes for JMS 執行任何轉換; 請參閱 第 149 頁的『在 JMSBytesMessage 中傳送及接收文字』。

傳送至 MQ 應用程式的 JMSBytesMessage 通常會在 MQ 樣式訊息內文中傳送，不含 JMS MQRFH2 標頭。 如果它傳送至 JMS 應用程式，則訊息內文樣式通常是 JMS。 除非應用程式置換，否則目的地屬性 WMQ_MESSAGE_BODY 及 WMQ_TARGET_DEST 的值會決定訊息內文樣式。 應用程式可以透過呼叫 destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ) 或 destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ) 來置換在目的地上設定的值。

如果您傳送含有 MQ 樣式內文的 JMSBytesMessage , 您可以從定義 MQ 或 JMS 訊息內文樣式的目的地接收訊息。 如果您傳送含有 JMS 樣式內文的 JMSBytesMessage , 則必須從定義 JMS 訊息內文樣式的目的地接收訊息。 如果沒有，則會將 MQRFH2 視為使用者訊息資料的一部分，這可能不是您所預期的。

無論訊息具有 MQ 或 JMS 主體樣式，都不會受到設定 WMQ_TARGET_DEST 的影響。

如果為訊息資料提供 Format，且已啟用佇列管理程式資料轉換，則稍後佇列管理程式可能會轉換訊息。 請勿將格式欄位用於指定訊息資料格式以外的任何項目，或將它保留空白， MQConstants.MQFMT_NONE

您可以在 JMSBytesMessage 中傳送多個項目。 當使用針對訊息定義的編碼來傳送訊息時，會轉換每一個數值項目。

您可以從 JMSBytesMessage 擷取個別資料項目。 呼叫 read 方法的順序與呼叫 write 方法來建立訊息的順序相同。 當使用訊息中儲存的 Encoding 值來呼叫訊息時，會轉換每一個數值項目。

與 JMSMapMessage 和 JMSStreamMessage 不同， JMSBytesMessage 只包含應用程式所寫入的資料。 沒有其他資料儲存在訊息資料中，例如用來在 JMSMapMessage 和 JMSStreamMessage 中定義項目的 XML 標籤。 因此，請使用 JMSBytesMessage 來傳送針對其他應用程式所格式化的訊息。

在 JMSBytesMessage 與 DataInputStream 和 DataOutputStream 之間轉換在部分應用程式中很有用。 需要以範例 第 150 頁的『使用 DataInputStream 及 DataOutputStream 來讀取及寫入訊息』為基礎的程式碼，才能搭配使用 com.ibm.mq.header 套件與 JMS。

範例

傳送及接收 JMSObjectMessage

```
ObjectMessage omo = session.createObjectMessage();
omo.setObject(new String("A string"));
producer.send(omo);
...
ObjectMessage omi = (ObjectMessage)consumer.receive();
System.out.println((String)omi.getObject());
...
A string
```

圖 *16:* 傳送及接收 *JMSObjectMessage*

傳送及接收 JMSTextmessage

文字訊息不能包含不同字集的文字。 此範例顯示在兩個不同訊息中傳送的不同字集的文字。

```
TextMessage tmo = session.createTextMessage();
tmo.setText("Sent in the character set defined for the destination");
producer.send(tmo);
```

圖 *17:* 以目的地定義的字集傳送文字訊息

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText("Sent in EBCDIC character set 37");
producer.send(tmo);
```

圖 *18:* 在 *ccsid 37* 中傳送文字訊息

```
TextMessage tmi = (TextMessage)consumer.receive();
System.out.println(tmi.getText());
...
Sent in the character set defined for the destination
```

圖 *19:* 接收文字訊息

在 JMSStreamMessage 和 JMSMapMessage 中傳送資料

```
StreamMessage smo = session.createStreamMessage();
smo.writeString("256");
smo.writeInt(512);
smo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
producer.send(smo);
...
MapMessage mmo = session.createMapMessage();
mmo.setString("First", "256");
mmo.setInt("Second", 512);
mmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
producer.send(mmo);
...
StreamMessage smi = (StreamMessage)consumer.receive();
System.out.println("Stream: First as float "   + smi.readFloat() +
                           " Second as String " + smi.readString());
...
Stream: First as float: 256.0, Second as String: 512
...
MapMessage mmi = (MapMessage)consumer.receive();
System.out.println("Map: Second as String " + mmi.getString("Second") +
                        " First as double "  + mmi.getDouble("First"));
...
Map: Second as String: 512, First as double: 256.0
```

圖 *20:* 在 *JMSStreamMessage* 和 *JMSMapMessage* 中傳送資料

在 JMSBytesMessage 中傳送及接收文字

第 149 頁的圖 21 中的程式碼會傳送 BytesMessage 中的字串。 為了簡單起見，此範例會傳送 JMSTextMessage 更適合的單一字串。 若要接收包含混合類型的字串 (以位元組為單位) 訊息，您必須知道 第 150 頁的圖 22 中稱為 *TEXT_LENGTH* 的字串長度 (以位元組為單位)。 即使是具有固定字元數的字串，位元組表示法的長度也可能較長。

```
BytesMessage bytes = session.createBytesMessage();
String codePage = CCSID.getCodepage(((MQDestination) destination)
                      .getIntProperty(WMQConstants.WMQ_CCSID));
bytes.writeBytes("In the destination code page".getBytes(codePage));
producer.send(bytes);
```

圖 *21:* 在 *JMSBytesMessage* 中傳送 *String*

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength())).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

圖 *22:* 從 *JMSBytesMessage* 接收 *String*

使用 DataInputStream 及 DataOutputStream 來讀取及寫入訊息

第 150 頁的圖 23 中的程式碼會使用 DataOutputStream 來建立 JMSBytesMessage。

```
ByteArrayOutputStream bout = new ByteArrayOutputStream();
DataOutputStream dout = new DataOutputStream(bout);
BytesMessage messageOut = prod.session.createBytesMessage();
// messageOut.setIntProperty(WMQConstants.JMS_IBM_ENCODING,
//                           ((MQDestination) (prod.destination)).getIntProperty
//                           (WMQConstants.WMQ_ENCODING));
int ccsidOut = (((MQDestination)prod.destination).getIntProperty(WMQConstants.WMQ_CCSID));
String codePageOut = CCSID.getCodepage(ccsidOut);
dout.writeInt(ccsidOut);
dout.write(codePageOut.getBytes(codePageOut));
messageOut.writeBytes(bout.toByteArray());
producer.send(messageOut);
```

圖 *23:* 使用 *DataOutputStream* 傳送 *JMSBytesMessage*

已註銷設定 JMS_IBM_ENCODING 內容的陳述式。 如果直接寫入 JMSBytesMessage，則陳述式有效，但在寫入 DataOutputStream 時無效。 寫入 DataOutputStream 的數字會以 Native 編碼來編碼。 設定 JMS_IBM_ENCODING 沒有作用。

第 150 頁的圖 24 中的程式碼會使用 DataInputStream 來接收 JMSBytesMessage。

```
static final int ccsidIn_SIZE = (Integer.SIZE)/8;
...
connection.start();
BytesMessage messageIn = (BytesMessage) consumer.receive();
int messageLength = new Long(messageIn.getBodyLength()).intValue();
byte [] bin = new byte[messageLength];
messageIn.readBytes(bin, messageLength);
DataInputStream din = new DataInputStream(new ByteArrayInputStream(bin));
int ccsidIn = din.readInt();
byte [] codePageByte = new byte[messageLength - ccsidIn_SIZE];
din.read(codePageByte, 0, codePageByte.length);
System.out.println("CCSID " + ccsidIn + " code page " + new String(codePageByte,
                    messageIn.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET)));
```

圖 *24:* 使用 *DataInputStream* 接收 *JMSBytesMessage*

使用輸入訊息資料 JMS_IBM_CHARACTER_SET 的字碼頁內容來印出字碼頁。 在輸入上，JMS_IBM_CHARACTER_SET 是 Java 字碼頁，不是數值編碼字集 ID。

訊息類型及轉換類型的表格

表 *32:* 訊息類型和轉換類型

	轉換類型			
訊息類型	**文字**	**數字**	**其他**	**無**
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			
JMSBytesMessage	readUTF writeUTF	readDouble readFloat readInt readLong readShort readUnsignedShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readUnsignedByte readBytes readChar writeByte writeBytes writeChar
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

相關概念

JMS 訊息轉換方法

JMS 應用程式設計人員可以使用許多資料轉換方法。 這些方法並非互斥; 有些應用程式可能會使用這些方法的組合。 如果您的應用程式只交換文字，或只與其他 JMS 應用程式交換訊息，則通常不會考量資料轉換。 IBM MQ 會自動為您執行資料轉換。

JMS 用戶端訊息轉換及編碼

會列出您用來執行 JMS 用戶端訊息轉換及編碼的方法，以及每一種轉換類型的程式碼範例。

佇列管理程式資料轉換
佇列管理程式資料轉換一律適用於從 JMS 用戶端接收訊息的非 JMS 應用程式。 接收訊息的 JMS 用戶端也會使用佇列管理程式資料轉換 (選用)。

相關工作

與非 JMS 應用程式交換格式化記錄
遵循此作業中建議的步驟，以設計並建置資料轉換結束程式，以及可使用 JMSBytesMessage 與非 JMS 應用程式交換訊息的 JMS 用戶端應用程式。 不論是否呼叫資料轉換結束程式，都可以與非 JMS 應用程式交換格式化訊息。

JMS 用戶端訊息轉換及編碼
會列出您用來執行 JMS 用戶端訊息轉換及編碼的方法，以及每一種轉換類型的程式碼範例。

在 JMS 訊息中讀取或寫入 Java 基本元素或物件時，會進行轉換及編碼。 轉換稱為 JMS 用戶端資料轉換，以區別它與佇列管理程式資料轉換及應用程式資料轉換。 在 JMS 訊息中讀取或寫入資料時，嚴格會進行轉換。 文字會轉換成內部 16 位元 Unicode 表示法，以及從內部 16 位元 Unicode 表示法轉換成文字 [3] 用於訊息中文字的字集。 數值資料會轉換為訊息定義的編碼，並將 Java 初始數值類型轉換為訊息定義的編碼。 是否執行轉換，以及執行的轉換類型，取決於 JMS 訊息類型及讀取或寫入作業。

第 152 頁的表 33 會依所執行的轉換類型來分類不同 JMS 訊息類型的讀取和寫入方法。 轉換類型在表格後面的文字中說明。

表 *33:* 訊息類型和轉換類型

	轉換類型			
訊息類型	**文字**	**數字**	**其他**	**無**
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			
JMSBytesMessage	readUTF writeUTF	readDouble readFloat readInt readLong readShort readUnsignedShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readUnsignedByte readBytes readChar writeByte writeBytes writeChar

[3] 部分 Unicode 表示法需要超過 16 位元。 請參閱 Java SE 參照。

表 *33:* 訊息類型和轉換類型 *(*繼續*)*				
	轉換類型			
訊息類型	**文字**	**數字**	**其他**	**無**
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

文字

目的地的預設 CodedCharacterSetId 是 1208 UTF-8。 依預設，文字會從 Unicode 進行轉換，並以 UTF-8 字串形式傳送。 接收時，文字會從用戶端所接收訊息中的編碼字集轉換成 Unicode。

setText 和 writeString 方法將文字從 Unicode 轉換為針對目的地定義的字集。 應用程式可以設定訊息內容 JMS_IBM_CHARACTER_SET 來置換目的地字集。 JMS_IBM_CHARACTER_SET，當傳送訊息時，必須是數值編碼字集 ID[4].

第 155 頁的『傳送及接收 JMSTextmessage』 中的程式碼 Snippet 會傳送兩則訊息。 一個以定義給目的地的字集傳送，另一個以應用程式定義的字集 37 傳送。

getText 和 readString 方法會將訊息中的文字從訊息中定義的字集轉換成 Unicode。 這些方法使用訊息內容 JMS_IBM_CHARACTER_SET 中所定義的字碼頁。 除非訊息是 MQ 類型的訊息，且沒有 MQRFH2，否則會從 MQRFH2.CodedCharacterSetId 對映字碼頁。 如果訊息是 MQ 類型的訊息 (不含 MQRFH2)，則會從 MQMD.CodedCharacterSetId 對映字碼頁。

第 156 頁的圖 29 中的程式碼 Snippet 會接收已傳送至目的地的訊息。 訊息中的文字會從字碼頁 IBM037 轉換回 Unicode。

註: 檢查文字是否轉換為編碼字集 37 的簡單方法是使用 IBM MQ Explorer。 在擷取訊息之前，請先瀏覽佇列並顯示訊息的內容。

請對照 第 155 頁的圖 28 中的程式碼 Snippet 與 第 154 頁的圖 25 中的不正確程式碼 Snippet。 在不正確的 Snippet 中，字串會轉換兩次，一次由應用程式轉換，一次由 IBM MQ 轉換。

[4] 當接收訊息 JMS_IBM_CHARACTER_SET 是 Java Charset 字碼頁名稱。

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText(new String("Sent in EBCDIC character set 37".getBytes(CCSID.getCodepage(37))));
producer.send(tmo);
```

圖 *25:* 字碼頁轉換不正確

writeUTF 方法會將文字從 Unicode 轉換為 1208 (UTF-8)。 字串前面有 2 個位元組的長度。 字串的長度上限為 65534 個位元組。 readUTF 方法會讀取 writeUTF 方法所寫入訊息中的項目。 它會確切讀取 writeUTF 方法所寫入的位元組數。

數字

目的地的預設數值編碼是 Native。 Java 的 Native 編碼常數值為 273 x'00000111'，所有平台都相同。 接收時，訊息中的數字會正確轉換為數值 Java 基本元素。 轉換會使用訊息中所定義的編碼，以及 read 方法所傳回的類型。

傳送方法會將 set 和 write 新增至訊息的數字轉換為針對目的地定義的數值編碼。 應用程式可以設定訊息內容 JMS_IBM_ENCODING 來置換訊息的目的地編碼; 例如:

```
message.setIntProperty(WMQConstants.JMS_IBM_ENCODING,
WMQConstants.WMQ_ENCODING_INTEGER_REVERSED);
```

get 和 read 數值方法會從訊息中定義的數值編碼轉換訊息中的數字。 它們會將數字轉換為 read 或 get 方法指定的類型; 請參閱 ENCODING 內容。 這些方法使用 JMS_IBM_ENCODING 中定義的編碼。 除非訊息是 MQ 類型的訊息，且沒有 MQRFH2，否則會從 MQRFH2.Encoding 對映編碼。 如果訊息是 MQ 類型的訊息 (不含 MQRFH2) ，則方法會使用 MQMD.Encoding 中定義的編碼。

第 156 頁的圖 30 中的範例顯示應用程式以目的地格式編碼數字，並以 JMSStreamMessage 傳送該數字。 比較 第 156 頁的圖 30 中的範例與 第 156 頁的圖 31 中的範例。 差異在於必須在 JMSBytesMessage 中設定 JMS_IBM_ENCODING 。

註: 檢查數字是否正確編碼的簡單方法是使用「IBM MQ 檔案總管」。 在耗用訊息之前瀏覽佇列並顯示訊息的內容。

其他

在 JMSByteMessage、 JMSStreamMessage 和 JMSMapMessage 中， boolean 方法會將 true 和 false 編碼成 x'01' 和 x'00' 。

UTF 方法會將 Unicode 編碼及解碼為 UTF-8 字串。 字串限制為小於 65536 個字元，且前面有 2 位元組長度欄位。

Object 方法會將初始類型包裝成物件。 數值和文字類型會編碼或轉換，就像使用數值和文字方法來讀取或寫入初始類型一樣。

無

readByte、 readBytes、 readUnsignedByte、 writeByte 及 writeBytes 方法會在應用程式與訊息之間取得或放置單一位元組或位元組陣列，而不進行轉換。 readChar 及 writeChar 方法會在應用程式與訊息之間取得並放置 2 位元組 Unicode 字元，而不進行轉換。

使用 readBytes 和 writeBytes 方法，應用程式可以執行自己的字碼點轉換，如 第 156 頁的『在 JMSBytesMessage 中傳送及接收文字』中所示。

IBM MQ 不會在用戶端中執行任何字碼頁轉換，因為訊息是 JMSBytesMessage，而且因為使用 readBytes 和 writeBytes 方法。 不過，如果位元組代表文字，請確定應用程式使用的字碼頁符合目的地的編碼字集。 佇列管理程式轉換結束程式可能會再次轉換訊息。 另一種可能性是接收端 JMS 用戶端程式可能遵循慣例，使用訊息中的 JMS_IBM_CHARACTER_SET 內容，將訊息中代表文字的任何位元組陣列轉換成字串或字元。

在此範例中，用戶端使用目的地編碼字集進行其轉換:

```
bytes.writeBytes("In the destination code page".getBytes(
CCSID.getCodepage(((MQDestination) destination)
.getIntProperty(WMQConstants.WMQ_CCSID))));
```

或者，用戶端可能已選擇字碼頁，然後在訊息的 JMS_IBM_CHARACTER_SET 內容中設定對應的編碼字集。 IBM MQ classes for Java 使用 JMS_IBM_CHARACTER_SET 來設定 MQRFH2 或訊息描述子 MQMD 中 JMS 內容的 CodedCharacterSetId 欄位:

```
String codePage = CCSID.getCodepage(37);
message.setIntProperty(wMQConstants.JMS_IBM_CHARACTER_SET, codePage);
5
```

如果位元組陣列寫入 JMSStringMessage 或 JMSMapMessage，則 IBM MQ classes for JMS 不會執行資料轉換，因為位元組鍵入為十六進位資料，而不是 JMSStringMessage 及 JMSMapMessage 中的文字。

如果位元組代表應用程式中的字元，您必須考量要讀取及寫入訊息的字碼點。 第 155 頁的圖 26 中的程式碼遵循使用目的地編碼字集的慣例。 如果您使用 JVM 的預設字集來建立字串，則位元組內容取決於平台。 Windows 上的 JVM 通常具有預設 Charset windows-1252，而 AIX and Linux 具有 UTF-8。 對於 Windows 與 AIX and Linux 之間的交換，您必須選取明確的字碼頁來交換文字 (以位元組為單位)。

```
StreamMessage smo = producer.session.createStreamMessage();
smo.writeBytes("123".getBytes(CCSID.getCodepage(((MQDestination) destination)
.getIntProperty(WMQConstants.WMQ_CCSID))));
```

圖 *26:* 使用目的地字集在 *JMSStreamMessage* 中寫入代表字串的位元組

範例

傳送及接收 JMSTextmessage

文字訊息不能包含不同字集的文字。 此範例顯示在兩個不同訊息中傳送的不同字集的文字。

```
TextMessage tmo = session.createTextMessage();
tmo.setText("Sent in the character set defined for the destination");
producer.send(tmo);
```

圖 *27:* 以目的地定義的字集傳送文字訊息

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText("Sent in EBCDIC character set 37");
producer.send(tmo);
```

圖 *28:* 在 *ccsid 37* 中傳送文字訊息

[5] SetStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage) currently accepts only numeric character set identifiers.

```
TextMessage tmi = (TextMessage)consumer.receive();
System.out.println(tmi.getText());
...
Sent in the character set defined for the destination
```

圖 *29:* 接收文字訊息

編碼範例

顯示以目的地定義的編碼所傳送的數字的範例。 請注意，您必須將 JMSBytesMessage 的 JMS_IBM_ENCODING 內容設為指定給目的地的值。

```
StreamMessage smo = session.createStreamMessage();
smo.writeInt(256);
producer.send(smo);
...
StreamMessage smi = (StreamMessage)consumer.receive();
System.out.println(smi.readInt());
...
256
```

圖 *30:* 在 *JMSStreamMessage* 中使用目的地編碼來傳送數字

```
BytesMessage bmo = session.createBytesMessage();
bmo.writeInt(256);
int encoding = ((MQDestination) (destination)).getIntProperty
    (WMQConstants.WMQ_ENCODING)
bmo.setIntProperty(WMQConstants.JMS_IBM_ENCODING, encoding);
producer.send(bmo);
...
BytesMessage bmi = (BytesMessage)consumer.receive();
System.out.println(bmi.readInt());
...
256
```

圖 *31:* 在 *JMSBytesMessage* 中使用目的地編碼來傳送數字

在 JMSBytesMessage 中傳送及接收文字

第 156 頁的圖 32 中的程式碼會傳送 BytesMessage 中的字串。 為了簡單起見，此範例會傳送 JMSTextMessage 更適合的單一字串。 若要接收包含混合類型的字串 (以位元組為單位) 訊息，您必須知道 第 157 頁的圖 33 中稱為 *TEXT_LENGTH* 的字串長度 (以位元組為單位)。 即使是具有固定字元數的字串，位元組表示法的長度也可能較長。

```
BytesMessage bytes = session.createBytesMessage();
String codePage = CCSID.getCodepage(((MQDestination) destination)
                      .getIntProperty(WMQConstants.WMQ_CCSID));
bytes.writeBytes("In the destination code page".getBytes(codePage));
producer.send(bytes);
```

圖 *32:* 在 *JMSBytesMessage* 中傳送 *String*

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength())).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

圖 *33:* 從 *JMSBytesMessage* 接收 *String*

相關概念

JMS 訊息轉換方法

JMS 應用程式設計人員可以使用許多資料轉換方法。 這些方法並非互斥; 有些應用程式可能會使用這些方法的組合。 如果您的應用程式只交換文字，或只與其他 JMS 應用程式交換訊息，則通常不會考量資料轉換。 IBM MQ 會自動為您執行資料轉換。

佇列管理程式資料轉換

佇列管理程式資料轉換一律適用於從 JMS 用戶端接收訊息的非 JMS 應用程式。 接收訊息的 JMS 用戶端也會使用佇列管理程式資料轉換 (選用)。

相關工作

與非 JMS 應用程式交換格式化記錄

遵循此作業中建議的步驟，以設計並建置資料轉換結束程式，以及可使用 JMSBytesMessage 與非 JMS 應用程式交換訊息的 JMS 用戶端應用程式。 不論是否呼叫資料轉換結束程式，都可以與非 JMS 應用程式交換格式化訊息。

相關參考

JMS 訊息類型及轉換

訊息類型的選擇會影響您的訊息轉換方法。 針對 JMS 訊息類型 JMSObjectMessage、 JMSTextMessage、 JMSMapMessage、 JMSStreamMessage 及 JMSBytesMessage，會說明訊息轉換與訊息類型的互動。

佇列管理程式資料轉換

佇列管理程式資料轉換一律適用於從 JMS 用戶端接收訊息的非 JMS 應用程式。 接收訊息的 JMS 用戶端也會使用佇列管理程式資料轉換 (選用)。

佇列管理程式可以使用為訊息資料設定的 CodedCharacterSetId、 Encoding 及 Format 值，來轉換訊息資料中的字元及數值資料。 對於非 JMS 應用程式，透過設定 GetMessage **選項**，一律可以使用轉換功能 GMO_CONVERT。

佇列管理程式可以轉換傳送至 JMS 用戶端的訊息。 佇列管理程式轉換是透過將目的地內容 WMQ_RECEIVE_CONVERSION 設為 WMQ_RECEIVE_CONVERSION_QMGR 或 WMQ_RECEIVE_CONVERSION_CLIENT_MSG 來控制。 應用程式可以變更目的地設定:

```
((MQDestination)destination).setIntProperty(
              WMQConstants.WMQ_RECEIVE_CONVERSION,
              WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

或

```
((MQDestination)destination).setReceiveConversion
              (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

圖 *34:* 啟用佇列管理程式資料轉換

當用戶端呼叫 consumer.receive 方法時，會進行 JMS 用戶端的佇列管理程式資料轉換。 依預設，文字資料會轉換為 UTF-8 (1208)。 後續讀取及取得方法會將從 UTF-8 收到的資料中的文字解碼，並以其內部

Unicode 編碼建立 Java 文字基本元素。 UTF-8 不是佇列管理程式資料轉換的唯一目標字集。 您可以設定 WMQ_RECEIVE_CCSID 目的地內容來選擇不同的 CCSID。

應用程式也可以變更目的地設定，例如將它設為 437 ， DOS-US:

```
((MQDestination)destination).setIntProperty
(WMQConstants.WMQ_RECEIVE_CCSID, 437);
```

或

```
((MQDestination)destination).setReceiveCCSID(437);
```

圖 *35:* 設定佇列管理程式轉換的目標編碼字集

變更 WMQ_RECEIVE_CCSID 的原因是特殊化的; 選擇的 CCSID 與在 JVM 中建立的文字物件沒有差異。 不過，在某些平台上，某些 JVM 可能無法處理從訊息中文字的 CCSID 轉換成 Unicode。 此選項可讓您對訊息中遞送至用戶端的任何文字選擇 CCSID。 部分 JMS 用戶端平台在以 UTF-8 遞送訊息文字時發生問題。

JMS 程式碼相當於 第 158 頁的圖 36 中 C 程式碼的粗體字。

```
gmo.Options = MQGMO_WAIT         /* wait for new messages           */
            | MQGMO_NO_SYNCPOINT /* no transaction                  */
            | MQGMO_CONVERT;     /* convert if necessary            */

     while (CompCode != MQCC_FAILED) {
     buflen = sizeof(buffer) - 1; /* buffer size available for GET   */
     memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
     memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
     md.Encoding       = MQENC_NATIVE;
     md.CodedCharSetId = MQCCSI_Q_MGR;

           MQGET(Hcon,          /* connection handle               */
           Hobj,                /* object handle                   */
           &md,                 /* message descriptor              */
           &gmo,                /* get message options             */
           buflen,              /* buffer length                   */
           buffer,              /* message buffer                  */
           &messlen,            /* message length                  */
           &CompCode,           /* completion code                 */
           &Reason);            /* reason code                     */
```

圖 *36:* 來自 *amqsget0.c* 的程式碼 *Snippet*

註:

只會對具有已知 IBM MQ 格式的訊息資料執行佇列管理程式轉換。 MQSTR， 或 MQCIH 是預先定義的已知格式範例。 只要您已提供資料轉換結束程式，已知格式也可以是使用者定義的格式。

建構為 JMSTextMessage、 JMSMapMessage 及 JMSStreamMessage 的訊息具有 MQSTR 格式，且可以由佇列管理程式進行轉換。

相關概念

JMS 訊息轉換方法
JMS 應用程式設計人員可以使用許多資料轉換方法。 這些方法並非互斥; 有些應用程式可能會使用這些方法的組合。 如果您的應用程式只交換文字，或只與其他 JMS 應用程式交換訊息，則通常不會考量資料轉換。 IBM MQ 會自動為您執行資料轉換。

JMS 用戶端訊息轉換及編碼
會列出您用來執行 JMS 用戶端訊息轉換及編碼的方法，以及每一種轉換類型的程式碼範例。

第 824 頁的『呼叫資料轉換結束程式』
資料轉換結束程式是使用者撰寫的結束程式，在處理 MQGET 呼叫期間接收控制。

相關工作

與非 JMS 應用程式交換格式化記錄
遵循此作業中建議的步驟，以設計並建置資料轉換結束程式，以及可使用 JMSBytesMessage 與非 JMS 應用程式交換訊息的 JMS 用戶端應用程式。 不論是否呼叫資料轉換結束程式，都可以與非 JMS 應用程式交換格式化訊息。

相關參考

JMS 訊息類型及轉換
訊息類型的選擇會影響您的訊息轉換方法。 針對 JMS 訊息類型 JMSObjectMessage、JMSTextMessage、 JMSMapMessage、 JMSStreamMessage 及 JMSBytesMessage，會說明訊息轉換與訊息類型的互動。

與非 *JMS* 應用程式交換格式化記錄
遵循此作業中建議的步驟，以設計並建置資料轉換結束程式，以及可使用 JMSBytesMessage 與非 JMS 應用程式交換訊息的 JMS 用戶端應用程式。 不論是否呼叫資料轉換結束程式，都可以與非 JMS 應用程式交換格式化訊息。

開始之前

您可能可以設計更簡單的解決方案，以使用 JMSTextMessage 與非 JMS 應用程式交換訊息。 在遵循此作業中的步驟之前，消除該可能性。

關於這項作業

如果 JMS 用戶端未涉及格式化與其他 JMS 用戶端交換的 JMS 訊息的詳細資料，則它更容易撰寫。 只要訊息類型是 JMSTextMessage、 JMSMapMessage、 JMSStreamMessage 或 JMSObjectMessage， IBM MQ 就會查看格式化訊息的詳細資料。 IBM MQ 處理不同平台上的字碼頁和數值編碼差異。

您可以使用這些訊息類型，與非 JMS 應用程式交換訊息。 若要這樣做，您必須瞭解 IBM MQ classes for JMS 如何建構這些訊息。 您可能可以修改非 JMS 應用程式來解譯訊息; 請參閱 第 125 頁的『將 JMS 訊息對映至 IBM MQ 訊息』。

使用其中一種訊息類型的優點是 JMS 用戶端程式設計並不取決於它與之交換訊息的應用程式類型。 缺點是它可能需要修改另一個程式，而您可能無法變更另一個程式。

替代方法是撰寫可處理現有訊息格式的 JMS 用戶端應用程式。 通常現有訊息是固定格式，包含未格式化資料、文字及數字的混合。 使用此作業中的步驟，以及 第 162 頁的『撰寫類別以在 JMSBytesMessage 中封裝記錄佈置』中的範例 JMS 用戶端，作為建置 JMS 用戶端的起點，該用戶端可以與非 JMS 應用程式交換格式化記錄。

程序

1. 定義記錄佈置，或使用其中一個預先定義的 IBM MQ 標頭類別。

   如需處理預先定義的 IBM MQ 標頭，請參閱 處理 IBM MQ 訊息標頭。

   第 160 頁的圖 37 是使用者定義的固定長度記錄佈置範例，可由資料轉換公用程式處理。
2. 建立資料轉換結束程式。

   遵循 寫入資料轉換結束程式 中的指示，以寫入資料轉換結束程式。

   若要嘗試 第 162 頁的『撰寫類別以在 JMSBytesMessage 中封裝記錄佈置』中的範例，請將資料轉換結束程式命名為 MYRECORD。
3. 撰寫 Java 類別以封裝記錄佈置，以及傳送和接收記錄。 您可能採取的兩種方法如下:

   - 將類別寫入讀取及寫入包含記錄的 JMSBytesMessage ; 請參閱 第 162 頁的『撰寫類別以在 JMSBytesMessage 中封裝記錄佈置』。
   - 撰寫延伸 com.ibm.mq.header.Header 的類別以定義記錄的資料結構; 請參閱 建立新標頭類型的類別。
4. 決定要在其中交換訊息的編碼字集。

   請參閱 選擇訊息轉換的方法 :receiver 使其良好。

5. 配置目的地以交換不含 JMS MQRFH2 標頭的 MQ 類型訊息。

   傳送和接收目的地都必須配置成交換 MQ-type 訊息。 您可以對傳送和接收使用相同的目的地。

   應用程式可以置換目的地訊息內文內容:

   ```
   ((MQDestination)destination).setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
   ```

   第 162 頁的『撰寫類別以在 JMSBytesMessage 中封裝記錄佈置』 中的範例會置換目的地訊息內文內容，以確保傳送 MQ 樣式訊息。
6. 使用 JMS 及非 JMS 應用程式來測試解決方案

   測試資料轉換結束程式的有用工具如下:

   - amqsgetc0.c 範例程式有助於測試接收 JMS 用戶端所傳送的訊息。 請參閱 第 161 頁的圖 38 中的建議修改，以使用範例標頭 RECORD.h。 修改之後， amqsgetc0.c 會接收範例 JMS 用戶端 TryMyRecord.java 所傳送的訊息; 請參閱 第 162 頁的『撰寫類別以在 JMSBytesMessage 中封裝記錄佈置』。
   - 範例 IBM MQ 瀏覽程式 amqsbcg0.c 有助於檢查訊息標頭、 JMS 標頭、 MQRFH2 及訊息內容的內容。
   - 先前在 SupportPac IH03 中提供的 **rfhutil** 程式容許擷取測試訊息並儲存在檔案中，然後用來驅動「訊息流程」。 輸出訊息也可以讀取並以各種格式顯示。 格式包括兩種類型的 XML ， 以及與 COBOL 記錄定義檔進行比對。 資料可以是 EBCDIC 或 ASCII。 在傳送訊息之前，可以將 RFH2 標頭新增至訊息。

   如果您嘗試使用已修改的 amqsgetc0.c 範例程式來接收訊息，並取得原因碼為 2080 的錯誤，請檢查訊息是否具有 MQRFH2。 修改會假設訊息已傳送至未指定 MQRFH2 的目的地。

範例

```
struct RECORD { MQCHAR StrucID[4];
                MQLONG Version;
                                MQLONG StructLength;
                                MQLONG Encoding;
                                MQLONG CodeCharSetId;
                                MQCHAR Format[8];
                                MQLONG Flags;
                                MQCHAR RecordData[32];
        };
```

圖 *37: RECORD.h*

- 宣告 RECORD.h 資料結構

```
struct tagRECORD {
   MQCHAR4    StrucId;
   MQLONG     Version;
   MQLONG     StrucLength;
   MQLONG     Encoding;
   MQLONG     CCSID;
   MQCHAR8    Format;
   MQLONG     Flags;
   MQCHAR32   RecordData;
 };
typedef struct tagRECORD RECORD;
   typedef RECORD MQPOINTER PRECORD;
   RECORD record;
   PRECORD pRecord = &(record);
```

- 修改 MQGET 呼叫以使用 RECORD,

  1. 修改之前:

```
    MQGET(Hcon,            /* connection handle     */
          Hobj,            /* object handle         */
          &md,             /* message descriptor    */
          &gmo,            /* get message options   */
          buflen,          /* buffer length         */
          buffer,          /* message buffer        */
          &messlen,        /* message length        */
          &CompCode,       /* completion code       */
          &Reason);        /* reason code           */
```

  2. 修改之後:

```
    MQGET(Hcon,            /* connection handle     */
          Hobj,            /* object handle         */
          &md,             /* message descriptor    */
          &gmo,            /* get message options   */
          sizeof(RECORD),  /* buffer length         */
          pRecord,         /* message buffer        */
          &messlen,        /* message length        */
          &CompCode,       /* completion code       */
          &Reason);        /* reason code           */
```

- 變更列印陳述式,

  1. 自:

```
buffer[messlen] = '\0';            /* add terminator          */
printf("message <%s>\n", buffer);
```

  2. 收件者:

```
/* buffer[messlen] = '\0';            add terminator          */
printf("ccsid <%d>, flags <%d>, message <%32.32s>\n \0",
        md.CodedCharSetId, record.Flags, record.RecordData);
```

圖 *38:* 修改 *amqsget0.c*

相關概念

JMS 訊息轉換方法

JMS 應用程式設計人員可以使用許多資料轉換方法。 這些方法並非互斥; 有些應用程式可能會使用這些方法的組合。 如果您的應用程式只交換文字，或只與其他 JMS 應用程式交換訊息，則通常不會考量資料轉換。 IBM MQ 會自動為您執行資料轉換。

JMS 用戶端訊息轉換及編碼

會列出您用來執行 JMS 用戶端訊息轉換及編碼的方法，以及每一種轉換類型的程式碼範例。

佇列管理程式資料轉換
佇列管理程式資料轉換一律適用於從 JMS 用戶端接收訊息的非 JMS 應用程式。 接收訊息的 JMS 用戶端也會使用佇列管理程式資料轉換 (選用)。

用於建立轉換-結束碼的公用程式

相關參考

JMS 訊息類型及轉換
訊息類型的選擇會影響您的訊息轉換方法。 針對 JMS 訊息類型 JMSObjectMessage、 JMSTextMessage、 JMSMapMessage、 JMSStreamMessage 及 JMSBytesMessage，會說明訊息轉換與訊息類型的互動。

撰寫類別以在 *JMSBytesMessage* 中封裝記錄佈置
此作業的目的是依範例探索如何在 JMSBytesMessage 中結合資料轉換與固定記錄佈置。 在作業中，您可以建立一些 Java 類別，以交換 JMSBytesMessage 中的範例記錄結構。 您可以修改範例以撰寫類別來交換其他記錄結構。

JMSBytesMessage 是 JMS 訊息類型的最佳選擇，可與非 JMS 程式交換混合資料類型記錄。 JMS 提供者未將其他資料插入訊息內文。 因此，如果 JMS 用戶端程式與現有的 IBM MQ 程式交互作業，則這是最佳的訊息類型選擇。 使用 JMSBytesMessage 的主要挑戰是符合其他程式所預期的編碼和字集。 解決方案是建立封裝記錄的類別。 針對特定記錄類型，封裝讀取及寫入 JMSBytesMessage 的類別可讓您更容易在 JMS 程式中傳送及接收固定格式記錄。 透過擷取抽象類別中介面的一般層面，許多解決方案可以針對不同的記錄格式重複使用。 在延伸抽象通用類別的類別中，可以實作不同的記錄格式。

另一種方法是延伸 com.ibm.mq.headers.Header 類別。 Header 類別有一些方法 (例如 addMQLONG)，以更宣告的方式來建置記錄格式。 使用 Header 類別的缺點是取得及設定屬性使用更複雜的解釋性介面。 這兩種方法會產生相同數量的應用程式碼。

除非每一個記錄使用相同的格式、編碼字集及編碼，否則除了 MQRFH2 之外， JMSBytesMessage 只能將單一格式封裝在一個訊息中。 JMSBytesMessage 的格式、編碼及字集是 MQRFH2 之後所有訊息的內容。 此範例是在假設 JMSBytesMessage 只包含一筆使用者記錄的情況下撰寫的。

開始之前

1. 您的技能水準: 您必須熟悉 Java 程式設計和 JMS。 未提供設定 Java 開發環境的相關指示。 撰寫程式以交換 JMSTextMessage、 JMSStreamMessage 或 JMSMapMessage 是很有利的。 然後，您可以查看使用 JMSBytesMessage 交換訊息的差異。
2. 此範例需要 IBM WebSphere MQ 7.0。
3. 此範例是使用 Eclipse 工作台的 Java 視景所建立。 它需要 JRE 6.0 或更新版本。 您可以在 IBM MQ Explorer 中使用 Java 視景，來開發及執行 Java 類別。 或者，使用您自己的 Java 開發環境。
4. 使用 IBM MQ Explorer 可讓您設定測試環境及除錯，比使用指令行公用程式更簡單。

關於這項作業

系統會引導您建立兩個類別: RECORD 和 MyRecord。 這兩個類別一起封裝固定格式的記錄。 它們具有取得及設定屬性的方法。 get 方法會從 JMSBytesMessage 讀取記錄，而 put 方法會將記錄寫入至 JMSBytesMessage。

此作業的目的不是建立您可以重複使用的正式作業品質類別。 您可以選擇使用作業中的範例，以開始使用您自己的類別。 此作業的目的是為您提供指引附註，主要是在使用 JMSBytesMessage 時使用字集、格式及編碼。 系統會說明建立類別的每一個步驟，並說明使用 JMSBytesMessage 的各個層面 (有時會忽略這些層面)。

RECORD 類別是抽象的，並定義使用者記錄的部分一般欄位。 一般欄位在標準 IBM MQ 標頭佈置上建模，該佈置具有眼睛捕捉器、版本及長度欄位。 會省略在許多 IBM MQ 標頭中找到的編碼、字集及格式欄位。 另一個標頭無法遵循使用者定義的格式。 延伸 RECORD 類別的 MyRecord 類別實際上是透過使用其他使用者欄位來延伸記錄來執行此動作。 由類別建立的 JMSBytesMessage 可以由佇列管理程式資料轉換結束程式處理。

第 168 頁的『用來執行範例的類別』 包含 RECORD 和 MyRecord 的完整清單。 它也包括額外 "scaffolding" 類別的清單，以測試 RECORD 和 MyRecord。 額外類別如下:

TryMyRecord
測試 RECORD 和 MyRecord 的主程式。

EndPoint
將 JMS 連線、目的地和階段作業封裝在單一類別中的抽象類別。 它的介面只符合測試 RECORD 和 MyRecord 類別的需求。 它不是用於撰寫 JMS 應用程式的已建立設計型樣。

註: 在建立目的地之後， Endpoint 類別會包含這一行程式碼:

```
((MQDestination)destination).setReceiveConversion
                            (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

在 V7.0 中，從 V7.0.1.5 開始，需要開啟佇列管理程式轉換。 依預設會予以停用。 在 V7.0 中，依預設會啟用最多 V7.0.1.4 佇列管理程式轉換，這一行程式碼會導致錯誤。

MyProducer 及 MyConsumer
延伸 EndPoint 並建立 MessageConsumer 和 MessageProducer 的類別，已連接並準備好接受要求。

所有類別一起組成您可以建置並實驗的完整應用程式，以瞭解如何在 JMSBytesMessage 中使用資料轉換。

程序

1. 建立抽象類別，以使用預設建構子來封裝 IBM MQ 標頭中的標準欄位。 稍後，您可以延伸類別，以自訂標頭來符合您的需求。

```
public abstract class RECORD implements Serializable {
   private static final long serialVersionUID = -1616617232750561712L;
   protected final static int UTF8 = 1208;
   protected final static int MQLONG_LENGTH = 4;
   protected final static int RECORD_STRUCT_ID_LENGTH = 4;
   protected final static int RECORD_VERSION_1 = 1;
   protected final String RECORD_STRUCT_ID = "BLNK";
   protected final String RECORD_TYPE = "BLANK   ";
   private String structID = RECORD_STRUCT_ID;
   private int version = RECORD_VERSION_1;
   private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
   private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
   private String headerCharset = "UTF-8";
   private String headerFormat = RECORD_TYPE;

   public RECORD() {
       super();
   }
```

註:

a. 屬性 structID 至 nextFormat 會依它們在標準 IBM MQ 訊息標頭中的佈置順序列出。

b. 屬性 format、 messageEncoding 及 messageCharset 說明標頭本身，且不是標頭的一部分。

c. 您必須決定是否儲存記錄的編碼字集 ID 或字集。 Java 使用字集，而 IBM MQ 訊息使用編碼字集 ID。 程式碼範例使用字集。

d. int 由 IBM MQ 序列化至 MQLONG 。 MQLONG 是 4 個位元組。

2. 建立專用屬性的 getter 和 setter。

a) 建立或產生 getter:

```
    public String getHeaderFormat() { return headerFormat; }
   public int getHeaderEncoding() { return headerEncoding;    }
   public String getMessageCharset() { return headerCharset; }
   public int getMessageEncoding() { return headerEncoding; }
   public String getStructID() { return structID; }
   public int getStructLength() { return structLength; }
   public int getVersion() { return version; }
```

b) 建立或產生 setter:

```
    public void setHeaderCharset(String charset) {
       this.headerCharset = charset; }
   public void setHeaderEncoding(int encoding) {
       this.headerEncoding = encoding; }
   public void setHeaderFormat(String headerFormat) {
       this.headerFormat = headerFormat; }
   public void setStructID(String structID) {
       this.structID = structID; }
   public void setStructLength(int structLength) {
       this.structLength = structLength; }
   public void setVersion(int version) {
       this.version = version; }
}
```

3. 建立建構子以從 JMSBytesMessage 建立 RECORD 實例。

```
public RECORD(BytesMessage message) throws JMSException, IOException,
       MQDataException {
   super();
   setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
   setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
   byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
   message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
   setStructID(new String(structID, getMessageCharset()));
   setVersion(message.readInt());
   setStructLength(message.readInt());
}
```

註:

a. messageCharset 和 messageEncoding 會從訊息內容中擷取，因為它們會置換設定給目的地的值。 format 未更新。 此範例不執行錯誤檢查。 如果呼叫 Record(BytesMessage) 建構子，則會假設 JMSBytesMessage 是 RECORD 類型訊息。 這條線 "setStructID(new String(structID, getMessageCharset()))" 會設定醒目標示。

b. 完成方法的程式碼行會依序將訊息中的欄位解除序列化，並更新 RECORD 實例中設定的預設值。

4. 建立 put 方法以將標頭欄位寫入至 JMSBytesMessage。

```
protected BytesMessage put(MyProducer myProducer) throws IOException,
       JMSException, UnsupportedEncodingException {
   setHeaderEncoding(myProducer.getEncoding());
   setHeaderCharset(myProducer.getCharset());
   myProducer.setMQClient(true);
   BytesMessage bytes = myProducer.session.createBytesMessage();
   bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
   bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
   bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                        myProducer.getCCSID());
   bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                    + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                    .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
   bytes.writeInt(getVersion());
   bytes.writeInt(getStructLength());
   return bytes;
}
```

註:

a. MyProducer 會將 JMS Connection、 Destination、 Session 和 MessageProducer 封裝在單一類別中。 MyConsumer(稍後使用) 會將 JMS Connection、 Destination、 Session 和 MessageConsumer 封裝在單一類別中。

b. 對於 JMSBytesMessage，如果編碼不是 Native，則必須在訊息中設定編碼。 目的地編碼會複製到訊息編碼屬性 JMS_IBM_CHARACTER_SET，並儲存為 RECORD 類別的屬性。

i) "setMessageEncoding(myProducer.getEncoding());" 呼叫 "(((MQDestination) destination).getIntProperty(WMQConstants.WMQ_ENCODING));" 以取得目的地編碼。

ii) "Bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getMessageEncoding());" 設定訊息編碼。

c. 用來將文字轉換成位元組的字集是從目的地取得，並儲存為 RECORD 類別的屬性。 它未設定在訊息中，因為 IBM MQ classes for JMS 在寫入 JMSBytesMessage 時不會使用它。

"messageCharset = myProducer.getCharset();" 呼叫

```
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());
}
```

它會從編碼字集 ID 取得 Java 字集。

" CCSID.getCodepage(ccsid) " 位於套件 com.ibm.mq.headers 中。 ccsid 是從 MyProducer 中的另一個方法取得，該方法會查詢目的地:

```
    public int getCCSID() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_CCSID));
    }
```

d. "myProducer.setMQClient(true);" 會置換用戶端類型的目的地設定，將它強制設為 IBM MQ MQI client。 您可能偏好省略這一行程式碼，因為它會遮蔽管理配置錯誤。

"myProducer.setMQClient(true);" 呼叫:

```
((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ); }
if (!getMQDest()) setMQBody();
```

如果 IBM MQ 主體樣式必須置換 JMS 設定，則程式碼會產生副作用，將其設為未指定。

註:

IBM MQ classes for JMS 會將訊息的格式、編碼及字集 ID 寫入訊息描述子 MQMD 或 JMS 標頭 MQRFH2。 這取決於訊息是否具有 IBM MQ 樣式主體。 請勿手動設定 MQMD 欄位。

已存在手動設定訊息描述子內容的方法。 它使用 JMS_IBM_MQMD_* 內容。 您必須設定目的地內容 WMQ_MQMD_WRITE_ENABLED ，以設定 JMS_IBM_MQMD_* 內容:

```
((MQDestination)destination).setMQMDWriteEnabled(true);
```

您必須設定目的地內容 WMQ_MQMD_READ_ENABLED，才能讀取內容。

只有在您完全控制整個訊息有效負載時，才使用 JMS_IBM_MQMD_* 。 與 JMS_IBM_* 內容不同，JMS_IBM_MQMD_* 內容不會控制 IBM MQ classes for JMS 如何建構 JMS 訊息。 可以建立與 JMS 訊息內容衝突的訊息描述子內容。

e. 完成方法的程式碼行會將類別中的屬性序列化為訊息中的欄位。

字串屬性會以空白填補。 字串會使用定義給記錄的字集轉換成位元組，並截斷成訊息欄位的長度。

5. 透過新增匯入項目來完成類別。

```
 package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
```

```
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;
```

6. 建立類別來擴充 RECORD 類別，以包含其他欄位。 包含預設建構子。

```
public class MyRecord extends RECORD {
   private static final long serialVersionUID = -370551723162299429L;
   private final static int FLAGS = 1;
   private final static String STRUCT_ID = "MYRD";
   private final static int DATA_LENGTH = 32;
   private final static String FORMAT = "MYRECORD";
   private int flags = FLAGS;
   private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

   public MyRecord() {
       super();
       super.setStructID(STRUCT_ID);
       super.setHeaderFormat(FORMAT);
       super.setStructLength(super.getStructLength() + MQLONG_LENGTH
               + DATA_LENGTH);
   }
```

註:

a. RECORD 子類別 MyRecord 會自訂標頭的眼睛捕捉器、格式及長度。

7. 建立或產生 getter 和 setter。

a) 建立 getter:

```
    public int getFlags() { return flags; }
   public String getRecordData() { return recordData; } .
```

b) 建立 setter:

```
    public void setFlags(int flags) {
       this.flags = flags; }
   public void setRecordData(String recordData) {
       this.recordData = recordData; }
}
```

8. 建立建構子以從 JMSBytesMessage 建立 MyRecord 實例。

```
    public MyRecord(BytesMessage message) throws JMSException, IOException,
           MQDataException {
       super(message);
       setFlags(message.readInt());
       byte[] recordData = new byte[DATA_LENGTH];
       message.readBytes(recordData, DATA_LENGTH);
       setRecordData(new String(recordData, super.getMessageCharset()));
   }
```

註:

a. RECORD 類別會先讀取組成標準訊息範本的欄位。

b. 使用訊息的字集內容將 recordData 文字轉換為 String 。

9. 建立靜態方法以從消費者取得訊息，並建立新的 MyRecord 實例。

```
    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
           MQDataException, IOException {
       BytesMessage message = (BytesMessage) myConsumer.receive();
       return new MyRecord(message);
   }
```

註:

a. 在此範例中，為了簡潔起見，會從 static get 方法呼叫 MyRecord(BytesMessage) 建構子。 一般而言，您可以將接收訊息與建立新的 MyRecord 實例分開。

10. 建立 put 方法，以將客戶欄位附加至包含訊息標頭的 JMSBytesMessage 。

```
 public BytesMessage put(MyProducer myProducer) throws JMSException,
        IOException {
    BytesMessage bytes = super.put(myProducer);
    bytes.writeInt(getFlags());
    bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                    + DATA_LENGTH + "s",getRecordData())
                    .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
    myProducer.send(bytes);
    return bytes;
}
```

註:

a. 程式碼中的方法會將 MyRecord 類別中的屬性序列化為訊息中的欄位。

- recordData String 屬性會以空白填補，使用定義給記錄的字集轉換成位元組，並截斷成 RecordData 欄位的長度。

11. 透過新增 include 陳述式來完成類別。

```
 package com.ibm.mq.id;
import java.io.IOException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.headers.MQDataException;
```

結果

- 執行 TryMyRecord 類別的結果:
  - 以編碼字集 37 傳送訊息，並使用佇列管理程式轉換結束程式:

```
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
In flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 273 CCSID UTF-8
```

  - 以編碼字集 37 傳送訊息，且不使用佇列管理程式轉換結束程式:

```
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
In flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID IBM037
```

- 修改 TryMyRecord 類別的結果不會接收訊息，而是使用已修改的 amqsget0.c 範例來接收訊息。 已修改的範例接受格式化記錄; 請參閱 第 159 頁的『與非 JMS 應用程式交換格式化記錄』中的 第 161 頁的圖 38 。
  - 以編碼字集 37 傳送訊息，並使用佇列管理程式轉換結束程式:

```
Sample AMQSGET0 start
ccsid <850>, flags <1>, message <ABCDEFGHIJKLMNOPQRSTUVWXYZ012345>
no more messages
Sample AMQSGET0 end
```

  - 以編碼字集 37 傳送訊息，且不使用佇列管理程式轉換結束程式:

```
Sample AMQSGET0 start
MQGET ended with reason code 2110
ccsid <37>, flags <1>, message <--+-+ãÃ++ÐÊËÈiÐÎÐ+ÔÒõõµþÞÚ-±=¾¶§>
```

```
no more messages
Sample AMQSGET0 end
```

嘗試使用不同的字碼頁和資料轉換結束程式來嘗試範例和實驗。 建立 Java 類別，配置 IBM MQ，並執行主程式 TryMyRecord ; 請參閱 第 168 頁的『#unique_196/unique_196_Connect_42_Try』。

1. 配置 IBM MQ 和 JMS 以執行範例。 這些指示用於在 Windows 上執行範例。

   a. 建立佇列管理程式

   ```
   crtmqm -sa -u SYSTEM.DEAD.LETTER.QUEUE QM1
   strmqm QM1
   ```

   b. 建立佇列

   ```
   echo DEFINE QL('Q1') REPLACE | runmqsc QM1
   ```

   c. 建立 JNDI 目錄

   ```
   cd c:\
   md JNDI-Directory
   ```

   d. 切換至 JMS bin 目錄

   必須從這裡執行「JMS 管理」程式。 路徑為 *MQ_INSTALLATION_PATH*\java\bin。

   e. 在稱為 JMSQM1Q1.txt 的檔案中建立下列 JMS 定義

   ```
   DEF CF(QM1) PROVIDERVERSION(7) QMANAGER(QM1)
   DEF Q(Q1) CCSID(37) ENCODING(RRR) MSGBODY(MQ) QMANAGER(QM1) QUEUE(Q1) TARGCLIENT(MQ)
   VERSION(7)
   END
   ```

   f. 執行 JMSAdmin 程式以建立 JMS 資源

   ```
   JMSAdmin < JMSQM1Q1.txt
   ```

2. 您可以使用「IBM MQ 檔案總管」來建立、變更及瀏覽您已建立的定義。
3. 執行 TryMyRecord。

用來執行範例的類別

下列程式碼區塊中列出的類別也可以在壓縮檔中使用。 下載 jm25529_.zip 或 jm25529_.tar.gz。

TryMyRecord

```
package com.ibm.mq.id;
public class TryMyRecord {
    public static void main(String[] args) throws Exception {
        MyProducer producer = new MyProducer();
        MyRecord outrec = new MyRecord();
        System.out.println("Out flags " + outrec.getFlags() + " text "
                + outrec.getRecordData() + " Encoding "
                + producer.getEncoding() + " CCSID " + producer.getCCSID()
                + " MQ " + producer.getMQDest());
        outrec.put(producer);
        System.out.println("Out flags " + outrec.getFlags() + " text "
                + outrec.getRecordData() + " Encoding "
                + producer.getEncoding() + " CCSID " + producer.getCCSID()
                + " MQ " + producer.getMQDest());
        MyRecord inrec = MyRecord.get(new MyConsumer());
        System.out.println("In  flags " + inrec.getFlags() + " text "
                + inrec.getRecordData() + " Encoding "
                + inrec.getMessageEncoding() + " CCSID "
                + inrec.getMessageCharset());
    }
}
```

RECORD

JM 3.0 V 9.3.0 V 9.3.0

```
package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;

public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK   ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }

    public RECORD(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super();
        setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
        setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
        byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
        message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
        setStructID(new String(structID, getMessageCharset()));
        setVersion(message.readInt());
        setStructLength(message.readInt());
    }

    public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }


    protected BytesMessage put(MyProducer myProducer) throws IOException,
            JMSException, UnsupportedEncodingException {
        setHeaderEncoding(myProducer.getEncoding());
        setHeaderCharset(myProducer.getCharset());
        myProducer.setMQClient(true);
        BytesMessage bytes = myProducer.session.createBytesMessage();
        bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
        bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
        bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                             myProducer.getCCSID());
        bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                         + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                         .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
        bytes.writeInt(getVersion());
        bytes.writeInt(getStructLength());
        return bytes;
    }

    public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
```

```
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK   ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }
    public RECORD(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super();
        setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
        setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
        byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
        message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
        setStructID(new String(structID, getMessageCharset()));
        setVersion(message.readInt());
        setStructLength(message.readInt());
    }

    public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }

    protected BytesMessage put(MyProducer myProducer) throws IOException,
            JMSException, UnsupportedEncodingException {
        setHeaderEncoding(myProducer.getEncoding());
        setHeaderCharset(myProducer.getCharset());
        myProducer.setMQClient(true);
        BytesMessage bytes = myProducer.session.createBytesMessage();
        bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
        bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
        bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                             myProducer.getCCSID());
        bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                         + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                         .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
        bytes.writeInt(getVersion());
        bytes.writeInt(getStructLength());
        return bytes;
    }

    public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
```

```
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

MyRecord

JM 3.0 V 9.3.0 V 9.3.0

```
package com.ibm.mq.id;
import java.io.IOException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.headers.MQDataException;

public class MyRecord extends RECORD {
    private static final long serialVersionUID = -370551723162299429L;
    private final static int FLAGS = 1;
    private final static String STRUCT_ID = "MYRD";
    private final static int DATA_LENGTH = 32;
    private final static String FORMAT = "MYRECORD";
    private int flags = FLAGS;
    private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

    public MyRecord() {
        super();
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
                + DATA_LENGTH);
    }

    public MyRecord(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super(message);
        setFlags(message.readInt());
        byte[] recordData = new byte[DATA_LENGTH];
        message.readBytes(recordData, DATA_LENGTH);
        setRecordData(new String(recordData, super.getMessageCharset()));
    }

    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
            MQDataException, IOException {
        BytesMessage message = (BytesMessage) myConsumer.receive();
        return new MyRecord(message);
    }

    public int getFlags() { return flags; }
    public String getRecordData() { return recordData; } .


    public BytesMessage put(MyProducer myProducer) throws JMSException,
            IOException {
        BytesMessage bytes = super.put(myProducer);
        bytes.writeInt(getFlags());
        bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                        + DATA_LENGTH + "s",getRecordData())
                        .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }

    public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import java.io.IOException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.headers.MQDataException;
public class MyRecord extends RECORD {
    private static final long serialVersionUID = -370551723162299429L;
    private final static int FLAGS = 1;
    private final static String STRUCT_ID = "MYRD";
    private final static int DATA_LENGTH = 32;
```

```
    private final static String FORMAT = "MYRECORD";
    private int flags = FLAGS;
    private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

    public MyRecord() {
        super();
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
                + DATA_LENGTH);
    }
    public MyRecord(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super(message);
        setFlags(message.readInt());
        byte[] recordData = new byte[DATA_LENGTH];
        message.readBytes(recordData, DATA_LENGTH);
        setRecordData(new String(recordData, super.getMessageCharset()));
    }
    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
            MQDataException, IOException {
        BytesMessage message = (BytesMessage) myConsumer.receive();
        return new MyRecord(message);
    }
    public int getFlags() { return flags; }
    public String getRecordData() { return recordData; } .

    public BytesMessage put(MyProducer myProducer) throws JMSException,
            IOException {
        BytesMessage bytes = super.put(myProducer);
        bytes.writeInt(getFlags());
        bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                        + DATA_LENGTH + "s",getRecordData())
                        .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }
    public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}
```

EndPoint

JM 3.0 V 9.3.0 V 9.3.0

```
package com.ibm.mq.id;
import java.io.UnsupportedEncodingException;
import jakarta.jms.Connection;
import jakarta.jms.ConnectionFactory;
import jakarta.jms.Destination;
import jakarta.jms.JMSException;
import jakarta.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.ibm.mq.headers.CCSID;
import com.ibm.mq.jms.MQDestination;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class EndPoint {
    public Context ctx;
    public ConnectionFactory cf;
    public Connection connection;
    public Destination destination;
    public Session session;
    protected EndPoint() throws NamingException, JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup("QM1");
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup("Q1");
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); }
    protected EndPoint(String cFactory, String dest) throws NamingException,
            JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
```

```
                          "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup(cFactory);
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup(dest);
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);  }
    public int getCCSID() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_CCSID));  }
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());  }
    public int getEncoding() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_ENCODING));  }
    public boolean getMQDest() throws JMSException {
        if ((((MQDestination) destination).getMessageBodyStyle()
               == WMQConstants.WMQ_MESSAGE_BODY_MQ)
            || ((((MQDestination) destination).getMessageBodyStyle()
                   == WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED)
               && (((MQDestination) destination).getTargetClient()
                   == WMQConstants.WMQ_TARGET_DEST_MQ)))
            return true;
        else
            return false; }
    public void setCCSID(int ccsid) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_CCSID,
                ccsid); }
    public void setEncoding(int encoding) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_ENCODING,
                encoding); }
    public void setMQBody() throws JMSException {
        ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);  }
    public void setMQBody(boolean mqbody) throws JMSException {
        if (mqbody)  ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
        else         ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_JMS);  }
    public void setMQClient(boolean mqclient) throws JMSException {
       if (mqclient){
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
            if (!getMQDest()) setMQBody();
        }
       else
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_JMS); }
    }
```

JMS 2.0

```
package com.ibm.mq.id;
import java.io.UnsupportedEncodingException;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.ibm.mq.headers.CCSID;
import com.ibm.mq.jms.MQDestination;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class EndPoint {
    public Context ctx;
    public ConnectionFactory cf;
    public Connection connection;
    public Destination destination;
    public Session session;
    protected EndPoint() throws NamingException, JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup("QM1");
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup("Q1");
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

```
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); }
    protected EndPoint(String cFactory, String dest) throws NamingException,
            JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup(cFactory);
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup(dest);
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);  }
    public int getCCSID() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_CCSID));  }
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());  }
    public int getEncoding() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_ENCODING));  }
    public boolean getMQDest() throws JMSException {
        if ((((MQDestination) destination).getMessageBodyStyle()
               == WMQConstants.WMQ_MESSAGE_BODY_MQ)
            || ((((MQDestination) destination).getMessageBodyStyle()
                   == WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED)
               && (((MQDestination) destination).getTargetClient()
                   == WMQConstants.WMQ_TARGET_DEST_MQ)))
            return true;
        else
            return false; }
    public void setCCSID(int ccsid) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_CCSID,
                ccsid); }
    public void setEncoding(int encoding) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_ENCODING,
                encoding); }
    public void setMQBody() throws JMSException {
        ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);  }
    public void setMQBody(boolean mqbody) throws JMSException {
        if (mqbody)  ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
        else         ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_JMS);  }
    public void setMQClient(boolean mqclient) throws JMSException {
       if (mqclient){
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
            if (!getMQDest()) setMQBody();
        }
       else
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_JMS); }
    }
```

MyProducer

JM 3.0 V 9.3.0 V 9.3.0

```
package com.ibm.mq.id;
import jakarta.jms.JMSException;
import jakarta.jms.Message;
import jakarta.jms.MessageProducer;
import javax.naming.NamingException;
public class MyProducer extends EndPoint {
    public MessageProducer producer;
    public MyProducer() throws NamingException, JMSException {
        super();
        producer = session.createProducer(destination); }
    public MyProducer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        producer = session.createProducer(destination); }
    public void send(Message message) throws JMSException {
        producer.send(message); }
}
```

JMS 2.0

```
package com.ibm.mq.id;
```

```
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageProducer;
import javax.naming.NamingException;
public class MyProducer extends EndPoint {
    public MessageProducer producer;
    public MyProducer() throws NamingException, JMSException {
        super();
        producer = session.createProducer(destination); }
    public MyProducer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        producer = session.createProducer(destination); }
    public void send(Message message) throws JMSException {
        producer.send(message); }
}
```

MyConsumer

JM 3.0 V 9.3.0 V 9.3.0

```
package com.ibm.mq.id;
import jakarta.jms.JMSException;
import jakarta.jms.Message;
import jakarta.jms.MessageConsumer;
import javax.naming.NamingException;
public class MyConsumer extends EndPoint {
    public MessageConsumer consumer;
    public MyConsumer() throws NamingException, JMSException {
        super();
        consumer = session.createConsumer(destination);
        connection.start();  }
    public MyConsumer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        consumer = session.createConsumer(destination);
        connection.start();  }
    public Message receive() throws JMSException {
        return consumer.receive();  }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.naming.NamingException;
public class MyConsumer extends EndPoint {
    public MessageConsumer consumer;
    public MyConsumer() throws NamingException, JMSException {
        super();
        consumer = session.createConsumer(destination);
        connection.start();  }
    public MyConsumer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        consumer = session.createConsumer(destination);
        connection.start();  }
    public Message receive() throws JMSException {
        return consumer.receive();  }
}
```

建立及配置 *Connection Factory* 和目的地

IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 應用程式可以建立 Connection Factory 及目的地，方法是從「Java 命名和目錄介面 (JNDI)」名稱空間、使用 IBM JMS 延伸，或使用 IBM MQ JMS 延伸，將它們擷取為受管理物件。 應用程式也可以使用 IBM JMS 延伸或 IBM MQ JMS 延伸來設定 Connection Factory 和目的地的內容。

在 JMS 或 Jakarta Messaging 應用程式的邏輯流程中， Connection Factory 和目的地是起始點。 應用程式使用 ConnectionFactory 物件來建立傳訊伺服器的連線，並使用「佇列」或「主題」物件作為目標，將訊息傳送至或從中接收訊息的來源。 因此，應用程式至少需要建立一個 Connection Factory 及一或多個目的地。 建立 Connection Factory 或目的地之後，應用程式可能需要設定物件的一或多個內容來配置物件。

總而言之，應用程式可以利用下列方式來建立及配置 Connection Factory 和目的地:

使用 JNDI 來擷取受管理物件

管理者可以使用 IBM MQ JMS 管理工具 (如 使用管理工具來配置 JMS 和 Jakarta 傳訊物件中所述) 或 IBM MQ Explorer (如 使用「IBM MQ 探險家」來配置 JMS 2.0 物件中所述)，在 JNDI 名稱空間中建立並配置 Connection Factory 和目的地作為受管理物件。 然後應用程式可以從 JNDI 名稱空間擷取受管理物件。 在擷取受管理物件之後，應用程式可以在必要時使用 IBM JMS 延伸或 IBM MQ JMS 延伸來設定或變更其一或多個內容。

註: JM 3.0 V 9.3.0 V 9.3.0 對於 Jakarta Messaging 3.0，您無法使用 IBM MQ Explorer 來管理 JNDI。 **JMSAdmin** 的 Jakarta Messaging 3.0 變式 **JMS30Admin** 支援 JNDI 管理。

使用 IBM JMS 延伸

應用程式可以使用 IBM JMS 延伸，在執行時期動態建立 Connection Factory 和目的地。 應用程式會先建立 JmsFactoryFactory 物件，然後使用此物件的方法來建立 Connection Factory 和目的地。 建立 Connection Factory 或目的地之後，應用程式可以使用繼承自 JmsProperty 環境定義介面的方法來設定其內容。 或者，應用程式可以在建立目的地時使用統一資源識別碼 (URI) 來指定目的地的一個以上內容。

使用 IBM MQ JMS 延伸

應用程式也可以使用 IBM MQ JMS 延伸，在執行時期動態建立 Connection Factory 和目的地。 應用程式使用提供的建構子來建立 Connection Factory 和目的地。 建立 Connection Factory 或目的地之後，應用程式可以使用物件的方法來設定其內容。 或者，應用程式可以在建立目的地時使用 URI 來指定目的地的一或多個內容。

相關工作

配置 JMS 和 Jakarta 傳訊資源

使用 *JNDI* 來擷取 *JMS* 或 *Jakarta Messaging* 應用程式中的受管理物件

若要從「Java 命名和目錄介面 (JNDI)」名稱空間擷取受管理物件， JMS 或 Jakarta Messaging 應用程式必須建立起始環境定義，然後使用 lookup () 方法來擷取物件。

在應用程式可以從 JNDI 名稱空間擷取受管理物件之前，管理者必須先建立受管理物件。

JMS 2.0 對於 JMS 2.0，管理者可以使用 IBM MQ JMS 管理工具、 **JMSAmin** 或 IBM MQ Explorer 來建立及維護 JNDI 名稱空間中的受管理物件。 如需相關資訊，請參閱 在 JNDI 名稱空間中配置 Connection Factory 和目的地。

JM 3.0 V 9.3.0 V 9.3.0 對於 Jakarta Messaging 3.0，您無法使用 IBM MQ Explorer 來管理 JNDI。 **JMSAdmin** 的 Jakarta Messaging 3.0 變式 **JMS30Admin** 支援 JNDI 管理。

應用程式伺服器通常會為受管理物件提供其自己的儲存庫，以及其自己用來建立及維護物件的工具。

如果要從 JNDI 名稱空間擷取受管理物件，應用程式必須先建立起始環境定義，如下列範例所示:

JM 3.0 V 9.3.0 V 9.3.0

```
import jakarta.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
.
.
.
String url = "ldap://server.company.com/o=company_us,c=us";
String icf = "com.sun.jndi.ldap.LdapCtxFactory";
.
java.util.Hashtable environment = new java.util.Hashtable();
environment.put(Context.PROVIDER_URL, url);
environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
Context ctx = new InitialDirContext(environment);
```

JMS 2.0

```
import javax.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
.
.
.
String url = "ldap://server.company.com/o=company_us,c=us";
```

```
String icf = "com.sun.jndi.ldap.LdapCtxFactory";
.
java.util.Hashtable environment = new java.util.Hashtable();
environment.put(Context.PROVIDER_URL, url);
environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
Context ctx = new InitialDirContext(environment);
```

在此程式碼中，字串變數 `url` 和 `icf` 具有下列意義:

url
目錄服務的統一資源定址器 (URL)。 URL 可以具有下列其中一種格式:

- `ldap://`*`hostname`*`/`*`contextName`* ，適用於以 LDAP 伺服器為基礎的目錄服務
- `file:/`*`directoryPath`* ，適用於基於本端檔案系統的目錄服務

ICF
起始環境定義 Factory 的類別名稱，可以是下列其中一個值:

- `com.sun.jndi.ldap.LdapCtxFactory`，適用於以 LDAP 伺服器為基礎的目錄服務
- `com.sun.jndi.fscontext.RefFSContextFactory`，適用於基於本端檔案系統的目錄服務

請注意， JNDI 套件和「輕量型目錄存取通訊協定 (LDAP)」服務提供者的部分組合可能會導致發生 LDAP 錯誤 84。 若要解決此問題，請在呼叫 InitialDirContext () 之前插入下列程式碼行:

```
environment.put(Context.REFERRAL, "throw");
```

取得起始環境定義之後，應用程式可以使用 lookup () 方法從 JNDI 名稱空間擷取受管理物件，如下列範例所示:

```
ConnectionFactory factory;
Queue queue;
Topic topic;
.
.
.
factory = (ConnectionFactory)ctx.lookup("cn=myCF");
queue = (Queue)ctx.lookup("cn=myQ");
topic = (Topic)ctx.lookup("cn=myT");
```

此程式碼會從 LDAP 型名稱空間擷取下列物件:

- 名稱為 myCF 的 ConnectionFactory 物件
- 名稱為 myQ 的佇列物件
- 名稱為 myT 的 Topic 物件

如需使用 JNDI 的相關資訊，請參閱 Oracle 公司提供的 JNDI 文件。

相關工作
使用 IBM MQ 探險家來配置 JMS 2.0 物件
使用管理工具來配置 JMS 和 Jakarta 傳訊物件
在 WebSphere Application Server 中配置 JMS 2.0 資源

使用 *IBM JMS* 延伸
IBM MQ classes for JMS (JMS 2.0) 和 IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0) 分別包含一組功能相同的 JMS API 延伸，稱為 IBM JMS 延伸。 應用程式可以使用這些延伸，在執行時期動態建立 Connection Factory 和目的地，以及設定 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 物件的內容。 延伸可以與任何傳訊提供者搭配使用。

IBM JMS 延伸是下列套件中的一組介面和類別:

- com.ibm.msg.client.jms
- com.ibm.msg.client.services

對於 Jakarta Messaging 3.0，這些套件位於 `com.ibm.jakarta.client.jar` 中。

JMS 2.0 若為 JMS 2.0，這些套件位於 com.ibm.mqjms.jar 或 com.ibm.mq.allclient.jar 中。

這些延伸提供下列功能:

- 以 Factory 為基礎的機制，可在執行時期動態建立 Connection Factory 和目的地，而不是從「Java 命名和目錄介面 (JNDI)」名稱空間擷取它們作為受管理物件
- 一組用來設定 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 物件內容的方法
- 一組異常狀況類別，含有取得問題詳細資訊的方法
- 一組控制追蹤的方法
- 一組方法，用於取得 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 的版本相關資訊

為了在執行時期動態建立 Connection Factory 和目的地，以及設定和取得其內容， IBM JMS 延伸提供 IBM MQ JMS 延伸的替代介面集。 不過，雖然 IBM MQ JMS 延伸專屬於 IBM MQ 傳訊提供者，但 IBM JMS 延伸並不專屬於 IBM MQ ，可以與 IBM MQ 適用於 JMS 架構的類別中所說明的分層架構內的任何傳訊提供者一起使用。

介面 com.ibm.msg.client.wmq.WMQConstants (JMS 2.0) 或 com.ibm.msg.jakarta.client.wmq.WMQConstants (Jakarta Messaging 3.0) 包含應用程式在使用 IBM JMS 延伸設定 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 物件的內容時可使用的常數定義。 介面包含 IBM MQ 傳訊提供者的常數，以及與任何傳訊提供者無關的 JMS 常數。

下列程式碼範例假設 Java 類別中包含下列 import 陳述式:

JM 3.0 V 9.3.0 V 9.3.0
```
import com.ibm.msg.jakarta.client.jms.*;
import com.ibm.msg.jakarta.client.services.*;
import com.ibm.msg.jakarta.client.wmq.WMQConstants;
```

JMS 2.0
```
import com.ibm.msg.client.jms.*;
import com.ibm.msg.client.services.*;
import com.ibm.msg.client.wmq.WMQConstants;
```

建立 Connection Factory 和目的地

應用程式必須先建立 JmsFactoryFactory 物件，才能使用 IBM JMS 延伸來建立 Connection Factory 和目的地。 如果要建立 JmsFactoryFactory 物件，應用程式會呼叫 JmsFactoryFactory 類別的 getInstance() 方法，如下列範例所示:

JM 3.0 V 9.3.0 V 9.3.0
```
JmsFactoryFactory ff = JmsFactoryFactory.getInstance(JmsConstants.JAKARTA_WMQ_PROVIDER);
```

JMS 2.0
```
JmsFactoryFactory ff = JmsFactoryFactory.getInstance(JmsConstants.WMQ_PROVIDER);
```

getInstance() 呼叫的參數是常數，用來將 IBM MQ 傳訊提供者識別為所選的傳訊提供者。 然後應用程式可以使用 JmsFactoryFactory 物件來建立 Connection Factory 和目的地。

如果要建立 Connection Factory ，應用程式會呼叫 JmsFactoryFactory 物件的 createConnectionFactory () 方法，如下列範例所示:

```
JmsConnectionFactory factory = ff.createConnectionFactory();
```

此陳述式會建立一個 JmsConnectionFactory 物件，其所有內容都具有預設值，這表示應用程式會以連結模式連接至預設佇列管理程式。 如果您想要應用程式以用戶端模式連接，或連接至預設佇列管理程式以外的佇列管理程式，則在建立連線之前，應用程式必須設定 JmsConnectionFactory 物件的適當內容。 如需如何執行此作業的相關資訊，請參閱第 179 頁的『設定 IBM MQ classes for JMS 物件的內容』。

JmsFactoryFactory 類別也包含建立下列類型的 Connection Factory 的方法:

- JmsQueueConnectionFactory
- JmsTopicConnectionFactory
- JmsXAConnectionFactory
- JmsXAQueueConnectionFactory
- JmsXATopicConnectionFactory

如果要建立 Queue 物件，應用程式會呼叫 JmsFactoryFactory 物件的 createQueue() 方法，如下列範例所示:

```
JmsQueue q1 = ff.createQueue("Q1");
```

此陳述式會使用其所有內容的預設值來建立 JmsQueue 物件。 此物件代表屬於本端佇列管理程式的 IBM MQ 佇列 Q1 。 此佇列可以是本端佇列、別名佇列或遠端佇列定義。

createQueue() 方法也可以接受佇列統一資源識別碼 (URI) 作為參數。 佇列 URI 是一個字串，用來指定 IBM MQ 佇列的名稱，以及選擇性地指定擁有佇列之佇列管理程式的名稱，以及 JmsQueue 物件的一或多個內容。 下列陳述式包含佇列 URI 的範例:

```
JmsQueue q2 = ff.createQueue("queue://QM2/Q2?persistence=2&priority=5");
```

此陳述式所建立的 JmsQueue 物件代表佇列管理程式 QM2 擁有的 IBM MQ 佇列 Q2 ，且傳送至此目的地的所有訊息都持續且優先順序為 5。 如需佇列 URI 的相關資訊，請參閱 第 190 頁的『統一資源識別碼 (URI)』。 如需設定 JmsQueue 物件內容的替代方式，請參閱 第 179 頁的『設定 IBM MQ classes for JMS 物件的內容』。

如果要建立 Topic 物件，應用程式可以使用 JmsFactoryFactory 物件的 createTopic() 方法，如下列範例所示:

```
JmsTopic t1 = ff.createTopic("Sport/Football/Results");
```

此陳述式會使用其所有內容的預設值來建立 JmsTopic 物件。 此物件代表稱為 Sport/Football/Results 的主題。

createTopic() 方法也可以接受主題 URI 作為參數。 主題 URI 是一個字串，指定主題的名稱，以及選擇性地指定 JmsTopic 物件的一個以上內容。 下列陳述式包含主題 URI 的範例:

```
String s1 = "topic://Sport/Tennis/Results?persistence=1&priority=0";
JmsTopic t2 = ff.createTopic(s1);
```

這些陳述式所建立的 JmsTopic 物件代表稱為 Sport/Tennis/Results 的主題，且傳送至此目的地的所有訊息都是非持續性且優先順序為 0。 如需主題 URI 的相關資訊，請參閱 第 190 頁的『統一資源識別碼 (URI)』。 如需設定 JmsTopic 物件內容的替代方式，請參閱 第 179 頁的『設定 IBM MQ classes for JMS 物件的內容』。

在應用程式建立 Connection Factory 或目的地之後，該物件只能與選取的傳訊提供者搭配使用。

設定 IBM MQ classes for JMS 物件的內容

若要使用 IBM JMS 延伸來設定 IBM MQ classes for JMS 物件的內容，應用程式會使用 com.ibm.msg.client.JmsPropertyContext 介面的方法。 同樣地，如果要使用 IBM JMS 延伸來設定 IBM MQ classes for Jakarta Messaging 物件的內容，應用程式會使用 com.ibm.msg.jakarta.client.JmsPropertyContext 介面的方法。

對於每一種 Java 資料類型， JmsProperty 環境定義介面包含用來設定具有該資料類型之內容值的方法，以及用來取得具有該資料類型之內容值的方法。 例如，應用程式呼叫 setIntProperty () 方法來設定具有整數值的內容，並呼叫 getIntProperty () 方法來取得具有整數值的內容。

com.ibm.mq.jms 和 com.ibm.mq.jakarta.jms 套件中的類別實例會繼承對應 JmsProperty 環境定義介面的方法。 因此，應用程式可以使用這些方法來設定 MQConnectionFactory、MQQueue 及 MQTopic 物件的內容。

當應用程式建立 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 物件時，會自動設定任何具有預設值的內容。 當應用程式設定內容時，新值會取代該內容先前具有的任何值。 在設定內容之後，無法刪除它，但可以變更它的值。

如果應用程式嘗試將內容設為對內容無效的值， IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 會擲出 JMSException 異常狀況。 如果應用程式嘗試取得尚未設定的內容，則行為如 JMS 規格中所述。 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 對初始資料類型擲出 NumberFormat 異常狀況，對參照的資料類型傳回空值。

除了 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 物件的預先定義內容之外，應用程式還可以設定自己的內容。 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 會忽略這些應用程式定義的內容。

如需 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 物件內容的相關資訊，請參閱 IBM MQ classes for JMS 物件的內容。

下列程式碼是如何使用 IBM JMS 延伸來設定內容的範例。 程式碼會設定 Connection Factory 的五個內容。

```
factory.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE,
WMQConstants.WMQ_CM_CLIENT);
factory.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM1");
factory.setStringProperty(WMQConstants.WMQ_HOST_NAME, "HOST1");
factory.setIntProperty(WMQConstants.WMQ_PORT, 1415);
factory.setStringProperty(WMQConstants.WMQ_CHANNEL, "QM1.SVR");
factory.setStringProperty(WMQConstants.WMQ_APPLICATIONNAME, "My Application");
```

設定這些內容的效果是應用程式會使用稱為 QM1.SVR 的 MQI 通道，以用戶端模式連接至佇列管理程式 QM1 。 佇列管理程式正在主機名稱為 HOST1 的系統上執行，佇列管理程式的接聽器正在埠號 1415 中接聽。 此連線及其他與其下階段作業相關聯的佇列管理程式連線，具有與它們相關聯的應用程式名稱「我的應用程式」。

註: 在 z/OS 平台上執行的佇列管理程式不支援設定應用程式名稱，因此會忽略此設定。

JmsProperty 環境定義介面也包含 setObjectProperty () 方法，應用程式可用來設定內容。 方法的第二個參數是封裝內容值的物件。 例如，下列程式碼會建立一個 Integer 物件來封裝整數 1415 ，然後呼叫 setObjectProperty () ，將 Connection Factory 的 PORT 內容設為值 1415:

```
Integer port = new Integer(1415);
factory.setObjectProperty(WMQConstants.WMQ_PORT, port);
```

因此，此程式碼相當於下列陳述式:

```
factory.setIntProperty(WMQConstants.WMQ_PORT, 1415);
```

相反地， getObjectProperty () 方法會傳回封裝內容值的物件。

將內容值從一個資料類型隱含轉換為另一個資料類型

當應用程式使用 JmsProperty 環境定義介面的方法來設定或取得 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 物件的內容時，該內容的值可以隱含地從一種資料類型轉換為另一種資料類型。

例如，下列陳述式會設定 JmsQueue 物件 q1:

```
q1.setStringProperty(WMQConstants.WMQ_PRIORITY, "5");
```

PRIORITY 內容具有整數值，因此 setStringProperty () 呼叫會將字串 "5" (來源值) 隱含地轉換為整數 5 (目標值) ，然後它會變成 PRIORITY 內容的值。

相反地，下列陳述式會取得 JmsQueue 物件 q1: 的 PRIORITY 內容:

```
String s1 = q1.getStringProperty(WMQConstants.WMQ_PRIORITY);
```

getStringProperty () 呼叫會將整數 5 (來源值) (即 PRIORITY 內容的值) 隱含地轉換為字串 "5" (目標值)。

第 181 頁的表 34 中顯示 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 支援的轉換。

表 *34:* 支援從一種資料類型到另一種資料類型的轉換

來源資料類型	**支援的目標資料類型**
布林值	字串
位元組	int、long、short、String
char	字串
倍精準數	字串
浮點數	double、String
整數	long、String
長整數	字串
短	int、long、String
字串	boolean、byte、double、float、int、long、short

控管支援轉換的一般規則如下:

- 數值可以從一種資料類型轉換成另一種資料類型，前提是在轉換期間不會遺失任何資料。 例如，資料類型為 `int` 的值可以轉換為資料類型為 `long` 的值，但無法轉換為資料類型為 `short` 的值。
- 任何資料類型的值都可以轉換成字串。
- 字串可以轉換為任何其他資料類型的值 (`char` 除外) 如果字串是正確的轉換格式。 如果應用程式嘗試轉換格式不正確的字串， IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 會擲出 NumberFormat 異常狀況。
- 如果應用程式嘗試不受支援的轉換， IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 會擲出 MessageFormat 異常狀況。

將值從一種資料類型轉換成另一種資料類型的特定規則如下:

- 將布林值轉換為字串時，值 `true` 會轉換為字串 "true"，而值 `false` 會轉換為字串 "false"。
- 將字串轉換為布林值時，會將字串 "true" (不區分大小寫) 轉換為 `true`，並將字串 "false" (不區分大小寫) 轉換為 `false`。 任何其他字串都會轉換為 `false`。
- 將字串轉換為資料類型為 `byte`、 `int`、 `long` 或 `short` 的值時，字串必須具有下列格式:

  [空白] [*sign*] *digits*

  字串元件的意義如下:

  空白
  選用前導空白字元。

  符號
  選用加號 (+) 或減號 (-)。

  數字
  連續的數字序列 (0-9)。 至少必須有一個數字。

  在數字序列之後，字串可以包含不是數字的其他字元，但一旦到達這些字元的第一個字元，轉換就會停止。 假設字串代表十進位整數。

  如果字串格式不正確， IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 會擲出 NumberFormat 異常狀況。

- 將字串轉換為資料類型為 `double` 或 `float` 的值時，字串必須具有下列格式:

  [空白] [*sign*] *digits* [*e_char* [*e_sign*] *e_digits*]

  字串元件的意義如下:

  空白
  選用前導空白字元。

  符號
  選用加號 (+) 或減號 (-)。

  數字
  連續的數字序列 (0-9)。 至少必須有一個數字。

  字元
  指數字元，可以是 *E* 或 *e*。

  _ 簽署
  指數的選用加號 (+) 或減號 (-)。

  數字 *(_D)*
  指數的連續數字序列 (0-9)。 如果字串包含指數字元，則至少必須存在一個數字。

  在數字序列或代表指數的選用字元之後，字串可以包含非數字的其他字元，但一旦到達這些字元的第一個字元，就會停止轉換。 假設字串代表具有 10 次方的指數的十進位浮點數字。

  如果字串格式不正確， IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 會擲出 NumberFormat 異常狀況。
- 轉換數值時 (包括資料類型為 `byte` 的值) 對於字串，該值會轉換為該值的字串表示法 (以十進位數表示) ，而不是包含該值的 ASCII 字元的字串。 例如，整數 65 會轉換為字串 "65"，而不是字串 "A"。

在單一呼叫中設定多個內容

JmsProperty 環境定義介面也包含 setBatchProperties () 方法，應用程式可以使用該方法在單一呼叫中設定多個內容。 方法的參數是封裝一組內容名稱/值配對的 Map 物件。

例如，下列程式碼使用 setBatchProperties () 方法來設定 Connection Factory 的相同五個內容，如 第 179 頁的『設定 IBM MQ classes for JMS 物件的內容』所示。 程式碼會建立 HashMap 類別的實例，以實作 Map 介面。

```
HashMap batchProperties = new HashMap();
batchProperties.put(WMQConstants.WMQ_CONNECTION_MODE,
                    new Integer(WMQConstants.WMQ_CM_CLIENT));
batchProperties.put(WMQConstants.WMQ_QUEUE_MANAGER, "QM1");
batchProperties.put(WMQConstants.WMQ_WMQ_HOST_NAME, "HOST1");
batchProperties.put(WMQConstants.WMQ_PORT, "1414");
batchProperties.put(WMQConstants.WMQ_CHANNEL, "QM1.SVR");
factory.setBatchProperties(batchProperties);
```

請注意， Map.put() 方法的第二個參數必須是物件。 因此，具有初始資料類型的內容值必須封裝在物件內，或由字串代表，如範例所示。

setBatchProperties () 方法會驗證每一個內容。 如果 setBatchProperties () 方法無法設定內容，例如，因為其值無效，則不會設定任何指定內容。

內容名稱和值

如果應用程式使用適當 JmsProperty 環境定義介面的方法來設定及取得 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 物件的內容，則應用程式可以使用下列任何方式來指定內容的名稱和值。 每一個隨附範例都顯示如何設定 JmsQueue 物件 q1 的 PRIORITY 內容，以便傳送至佇列的訊息具有在 send () 呼叫上指定的優先順序。

使用在 com.ibm.msg.client.wmq.WMQConstants 介面中定義為常數的內容名稱和值
下列陳述式是如何以這種方式指定內容名稱及值的範例:

```
q1.setIntProperty(WMQConstants.WMQ_PRIORITY, WMQConstants.WMQ_PRI_APP);
```

使用可在佇列和主題統一資源識別碼 (URI) 中使用的內容名稱和值

下列陳述式是如何以這種方式指定內容名稱及值的範例:

```
q1.setIntProperty("priority", -2);
```

只能以這種方式指定目的地內容的名稱和值。

使用 IBM MQ JMS 管理工具所辨識的內容名稱及值

下列陳述式是如何以這種方式指定內容名稱及值的範例:

```
q1.setStringProperty("PRIORITY", "APP");
```

內容名稱的簡短格式也可接受，如下列陳述式所示:

```
q1.setStringProperty("PRI", "APP");
```

當應用程式取得內容時，傳回的值取決於應用程式指定內容名稱的方式。 例如，如果應用程式指定常數 WMQConstants.WMQ_PRIORITY 作為內容名稱，則傳回的值為整數 -2:

```
int n1 = getIntProperty(WMQConstants.WMQ_PRIORITY);
```

如果應用程式指定字串 "priority" 作為內容名稱，則會傳回相同的值:

```
int n2 = getIntProperty("priority");
```

不過，如果應用程式指定字串 "PRIORITY" 或 "PRI" 作為內容名稱，則傳回的值是字串 "APP":

```
String s1 = getStringProperty("PRI");
```

在內部， IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 會將內容名稱和值儲存為相符 WMQConstants 介面中定義的文字值。 這是內容名稱和值的已定義標準格式。 作為一般規則，如果應用程式使用另外兩種指定內容名稱和值的方式之一來設定內容，則 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 必須將名稱和值從指定的輸入格式轉換成標準格式。 同樣地，如果應用程式使用另兩種指定內容名稱和值的方式之一來取得內容，則 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 必須將指定輸入格式的名稱轉換成標準格式，並將標準格式的值轉換成所需的輸出格式。 必須執行這些轉換可能會影響效能。

異常狀況、追蹤檔或 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 日誌中所傳回的內容名稱和值一律採用標準格式。

使用地圖介面

JmsProperty 環境定義介面延伸 java.util.Map 介面。 因此，應用程式可以使用「對映」介面的方法來存取 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 物件的內容。

例如，下列程式碼會印出 Connection Factory 所有內容的名稱和值。 程式碼只會使用「對映」介面的方法來取得內容的名稱和值。

```
// Get the names of all the properties
Set propNames = factory.keySet();

// Loop round all the property names and get the property values
Iterator iterator = propNames.iterator();
while (iterator.hasNext()){
    String pName = (String)iterator.next();
```

```
    System.out.println(pName+"="+factory.get(pName));
}
```

使用「對映」介面的方法不會略過任何內容驗證或轉換。

使用 *IBM MQ JMS* 延伸
IBM MQ classes for JMS 包含 JMS API 的一組延伸，稱為 IBM MQ JMS 延伸。 應用程式可以使用這些延伸，在執行時期動態建立 Connection Factory 和目的地，以及設定 Connection Factory 和目的地的內容。

IBM MQ classes for JMS 包含 com.ibm.jms 及 com.ibm.mq.jms 套件中的一組類別。 這些類別會實作 JMS 介面，並包含 IBM MQ JMS 延伸。 下列程式碼範例假設下列陳述式已匯入這些套件:

```
import com.ibm.jms.*;
import com.ibm.mq.jms.*;
import com.ibm.msg.client.wmq.WMQConstants;
```

應用程式可以使用 IBM MQ JMS 延伸來執行下列功能:

- 在執行時期動態建立 Connection Factory 和目的地，而不是從「Java 命名和目錄介面 (JNDI)」名稱空間擷取它們作為受管理物件
- 設定 Connection Factory 和目的地的內容

建立 Connection Factory

如果要建立 Connection Factory ，應用程式可以使用 MQConnectionFactory 建構子，如下列範例所示:

```
MQConnectionFactory factory = new MQConnectionFactory();
```

此陳述式會建立一個 MQConnectionFactory 物件，其所有內容都具有預設值，這表示應用程式會以連結模式連接至預設佇列管理程式。 如果您想要應用程式以用戶端模式連接，或連接至預設佇列管理程式以外的佇列管理程式，則在建立連線之前，應用程式必須設定 MQConnectionFactory 物件的適當內容。 如需如何執行此作業的相關資訊，請參閱第 184 頁的『設定 Connection Factory 的內容』。

應用程式可以用類似的方式來建立下列類型的 Connection Factory:

- MQQueueConnectionFactory
- MQTopicConnectionFactory
- MQXAConnectionFactory
- MQXAQueueConnectionFactory
- MQXATopicConnectionFactory

設定 Connection Factory 的內容

應用程式可以呼叫 Connection Factory 的適當方法來設定 Connection Factory 的內容。 Connection Factory 可以是受管理物件或在執行時期動態建立的物件。

請考量下列程式碼，例如:

```
MQConnectionFactory factory = new MQConnectionFactory();
.
factory.setTransportType(WMQConstants.WMQ_CM_CLIENT);
factory.setQueueManager("QM1");
factory.setHostName("HOST1");
factory.setPort(1415);
factory.setChannel("QM1.SVR");
```

此程式碼會建立 MQConnectionFactory 物件，然後設定物件的五個內容。 設定這些內容的效果是應用程式會使用稱為 QM1.SVR 的 MQI 通道，以用戶端模式連接至佇列管理程式 QM1 。 佇列管理程式正在主機名稱為 HOST1 的系統上執行，佇列管理程式的接聽器正在埠號 1415 中接聽。

使用分配管理系統即時連線的應用程式只能使用傳訊的發佈/訂閱樣式。 它無法使用點對點樣式的傳訊。

只有 Connection Factory 的某些內容組合才有效。 如需哪些組合有效的相關資訊，請參閱 IBM MQ classes for JMS 物件內容之間的相依關係。

如需 Connection Factory 的內容以及用來設定其內容之方法的相關資訊，請參閱 IBM MQ classes for JMS 物件的內容。

建立目的地

若要建立 Queue 物件，應用程式可以使用 MQQueue 建構子，如下列範例所示:

```
MQQueue q1 = new MQQueue("Q1");
```

此陳述式會使用其所有內容的預設值來建立 MQQueue 物件。 此物件代表屬於本端佇列管理程式的 IBM MQ 佇列 Q1 。 此佇列可以是本端佇列、別名佇列或遠端佇列定義。

MQQueue 建構子的替代形式有兩個參數，如下列範例所示:

```
MQQueue q2 = new MQQueue("QM2", "Q2");
```

此陳述式所建立的 MQQueue 物件代表佇列管理程式 QM2 擁有的 IBM MQ 佇列，稱為 Q2 。 以這種方式識別的佇列管理程式可以是本端佇列管理程式或遠端佇列管理程式。 如果它是遠端佇列管理程式，則必須配置 IBM MQ ，以便在應用程式將訊息傳送至此目的地時， WebSphere MQ 可以將訊息從本端佇列管理程式遞送至遠端佇列管理程式。

MQQueue 建構子也可以接受佇列統一資源識別碼 (URI) 作為單一參數。 佇列 URI 是一個字串，指定 IBM MQ 佇列的名稱，並選擇性地指定擁有佇列之佇列管理程式的名稱，以及 MQQueue 物件的一或多個內容。 下列陳述式包含佇列 URI 的範例:

```
MQQueue q3 = new MQQueue("queue://QM3/Q3?persistence=2&priority=5");
```

此陳述式所建立的 MQQueue 物件代表佇列管理程式 QM3 擁有的 IBM MQ 佇列 Q3 ，且傳送至此目的地的所有訊息都是持續且優先順序為 5。 如需佇列 URI 的相關資訊，請參閱 第 190 頁的『統一資源識別碼 (URI)』。 如需設定 MQQueue 物件內容的替代方式，請參閱 第 185 頁的『設定目的地的內容』。

若要建立 Topic 物件，應用程式可以使用 MQTopic 建構子，如下列範例所示:

```
MQTopic t1 = new MQTopic("Sport/Football/Results");
```

此陳述式會使用其所有內容的預設值來建立 MQTopic 物件。 此物件代表稱為 Sport/Football/Results 的主題。

MQTopic 建構子也可以接受主題 URI 作為參數。 主題 URI 是一個字串，指定主題的名稱，以及選擇性地指定 MQTopic 物件的一或多個內容。 下列陳述式包含主題 URI 的範例:

```
MQTopic t2 = new MQTopic("topic://Sport/Tennis/Results?persistence=1&priority=0");
```

此陳述式所建立的 MQTopic 物件代表稱為 Sport/Tennis/Results 的主題，且傳送至此目的地的所有訊息都是非持續性且優先順序為 0。 如需主題 URI 的相關資訊，請參閱 第 190 頁的『統一資源識別碼 (URI)』。 如需設定 MQTopic 物件內容的替代方式，請參閱 第 185 頁的『設定目的地的內容』。

設定目的地的內容

應用程式可以呼叫目的地的適當方法來設定目的地的內容。 目的地可以是受管理物件或在執行時期動態建立的物件。

請考量下列程式碼，例如:

```
MQQueue q1 = new MQQueue("Q1");
.
```

```
q1.setPersistence(WMQConstants.WMQ_PER_PER);
q1.setPriority(5);
```

此程式碼會建立 MQQueue 物件，然後設定物件的兩個內容。 設定這些內容的效果是傳送至目的地的所有訊息都是持續性的，且優先順序為 5。

應用程式可以用類似的方式來設定 MQTopic 物件的內容，如下列範例所示:

```
MQTopic t1 = new MQTopic("Sport/Football/Results");
.
t1.setPersistence(WMQConstants.WMQ_PER_NON);
t1.setPriority(0);
```

此程式碼會建立 MQTopic 物件，然後設定物件的兩個內容。 設定這些內容的效果是傳送至目的地的所有訊息都是非持續性且優先順序為 0。

如需目的地內容以及用來設定其內容之方法的相關資訊，請參閱 IBM MQ classes for JMS 物件的內容。

Linux AIX 從 *JMS* 應用程式連接至 *IBM MQ*

為了建置連線， JMS 應用程式會使用 **ConnectionFactory** 物件來建立 **Connection** 物件，然後啟動連線。

對於 JMS 2.0 以及更新版本，應用程式通常會使用 **ConnectionFactory** 物件和 `createContext()` 方法來連接至傳訊提供者。

在舊版 JMS 中，您必須先使用 `createConnection` 來建立 **Connection** 物件，然後啟動連線呼叫 `getSession()` 來建立可執行傳訊作業的 **Session** 物件。

JMSContext 物件會有效地封裝 **Connection** 和 **Session** 物件。 如果您想要使用傳統方法並直接建立連線和階段作業物件，請參閱 第 186 頁的『在 JMS 應用程式中建置連線』 和 第 187 頁的『在 JMS 應用程式中建立階段作業』。

若要建立 **JMSContext** 物件，應用程式會使用 **ConnectionFactory** 物件的 `createContext()` 方法，如下列範例所示:

```
ConnectionFactory factory;
Connection connection;
.
.
.
connection = factory.createContext();
```

建立 JMS 連線時， IBM MQ classes for JMS 會建立連線控點 (Hconn) ，並啟動與佇列管理程式的交談。

註: 請注意，應用程式程序 ID 會作為要傳遞至佇列管理程式的預設使用者身分。 如果應用程式在用戶端傳輸模式中執行，則伺服器上必須存在具有相關授權的這個程序 ID。 如果您想要使用不同的身分，請使用 `createConnection(username,password)` 方法。

V 9.3.5 這個機制也可以用來提供鑑別記號，請參閱 從您選擇的記號發證者取得鑑別記號。

JMS 1.0 在 *JMS* 應用程式中建置連線

為了 在 JMS 1.0 中建置連線， JMS 應用程式會使用 ConnectionFactory 物件來建立 Connection 物件，然後啟動連線。

如果要建立 Connection 物件，應用程式會使用 ConnectionFactory 物件的 createConnection() 方法，如下列範例所示:

```
ConnectionFactory factory;
Connection connection;
.
.
.
connection = factory.createConnection();
```

建立 JMS 連線時， IBM MQ classes for JMS 會建立連線控點 (Hconn) ，並啟動與佇列管理程式的交談。

QueueConnectionFactory 介面和 TopicConnectionFactory 介面各繼承來自 ConnectionFactory 介面的 createConnection() 方法。 因此，您可以使用 createConnection() 方法來建立網域特定的物件，如下列範例所示:

```
QueueConnectionFactory qcf;
Connection connection;
.
.
.
connection = qcf.createConnection();
```

此程式碼片段會建立 QueueConnection 物件。 應用程式現在可以在此物件上執行與網域無關的作業，或只適用於點對點網域的作業。 不過，如果應用程式嘗試執行只適用於發佈/訂閱網域的作業，則會擲出 IllegalState 異常狀況，並出現下列訊息:

```
JMSMQ1112: Operation for a domain specific object was not valid.
           Operation createProducer() is not valid for type com.ibm.mq.jms.MQTopic
```

這是因為連線是從網域特定的 Connection Factory 建立的。

註: 請注意，應用程式程序 ID 會作為要傳遞至佇列管理程式的預設使用者身分。 如果應用程式在用戶端傳輸模式中執行，則伺服器上必須存在具有相關授權的這個程序 ID。 如果您想要使用不同的身分，請使用 createConnection(username , password) 方法。

JMS 規格指出以 stopped 狀態建立連線。 在連線啟動之前，與連線相關聯的訊息消費者無法接收任何訊息。 若要啟動連線，應用程式會使用 Connection 物件的 start () 方法，如下列範例所示:

```
connection.start();
```

V 9.3.5 這個機制也可以用來提供鑑別記號，請參閱 從您選擇的記號發證者取得鑑別記號。

JMS 1.0 在 *JMS* 應用程式中建立階段作業

若要建立階段作業 在 JMS 1.0 中， JMS 應用程式會使用 Connection 物件的 createSession() 方法。

createSession() 方法有兩個參數:

1. 指定階段作業是否進行交易的參數
2. 指定階段作業確認模式的參數

例如，下列程式碼會建立未交易的階段作業，且確認模式為 AUTO_ACKNOWLEDGE:

```
Session session;
.
boolean transacted = false;
session = connection.createSession(transacted, Session.AUTO_ACKNOWLEDGE);
```

建立 JMS 階段作業時， IBM MQ classes for JMS 會建立連線控點 (Hconn) ， 並啟動與佇列管理程式的交談。

Session 物件以及從它建立的任何 MessageProducer 或 MessageConsumer 物件，無法由多執行緒應用程式的不同執行緒同時使用。 要確保這些物件不會同時使用，最簡單的方法是為每一個執行緒建立個別的 Session 物件。

V 9.3.5 這個機制也可以用來提供鑑別記號，請參閱 從您選擇的記號發證者取得鑑別記號。

JMS 應用程式中的交易式階段作業

JMS 應用程式可以先建立交易式階段作業來執行區域交易。 應用程式可以確定或回復交易。

JMS 應用程式可以執行區域交易。 區域交易是只涉及變更應用程式所連接之佇列管理程式資源的交易。 若要執行區域交易，應用程式必須先透過呼叫 Connection 物件的 createSession() 方法，並指定作為交易階段作業的參數，來建立交易階段作業。 隨後，在階段作業內傳送及接收的所有訊息會分組成一連串交易。 當應用程式確定或回復自交易開始以來所傳送及接收的訊息時，交易會結束。

如果要確定交易，應用程式會呼叫 Session 物件的 commit () 方法。 當確定交易時，在交易內傳送的所有訊息都會變成可供遞送給其他應用程式，且會確認交易內收到的所有訊息，以便傳訊伺服器不會再次嘗試將它們遞送給應用程式。 在點對點網域中，傳訊伺服器也會從其佇列中移除收到的訊息。

如果要回復交易，應用程式會呼叫 Session 物件的 rollback () 方法。 當回復交易時，傳訊伺服器會捨棄交易內所傳送的所有訊息，且交易內所接收的所有訊息會重新可供遞送。 在點對點網域中，接收到的訊息會放回其佇列中，並再次對其他應用程式可見。

當應用程式建立交易式階段作業或呼叫 commit () 或 rollback () 方法時，新交易會自動啟動。 因此，交易式階段作業一律具有作用中交易。

當應用程式關閉交易式階段作業時，會進行隱含回復。 當應用程式關閉連線時，所有連線的交易階段作業都會進行隱含回復。

如果應用程式結束而未關閉連線，則所有連線的交易階段作業也會發生隱含回復。

交易完全包含在交易式階段作業內。 交易無法跨越階段作業。 這表示應用程式無法在兩個以上交易階段作業中傳送及接收訊息，然後將所有這些動作確定或回復為單一交易。

JMS 階段作業的確認模式

每一個未交易的階段作業都有確認通知模式，可決定如何確認應用程式收到的訊息。 有三種確認模式可用，而選擇確認模式會影響應用程式的設計。

如果階段作業未進行交易，則由階段作業的確認模式決定應用程式所收到訊息的確認方式。 下列段落說明三種確認模式:

自動確認

階段作業會自動確認應用程式收到的每一則訊息。

如果訊息同步遞送至應用程式，階段作業會在每次「接收」呼叫順利完成時確認接收訊息。 如果非同步遞送訊息，則每次呼叫訊息接聽器的 onMessage() 方法順利完成時，階段作業都會確認收到訊息。

如果應用程式順利接收訊息，但失敗阻止發生確認通知，則訊息會再次變成可供遞送。 因此，應用程式必須能夠處理重新遞送的訊息。

Dups_ok_acknowledge

階段作業會確認應用程式在選取時所收到的訊息。

使用此確認模式可減少階段作業必須執行的工作量，但阻止訊息確認的失敗可能會導致多個訊息重新可供遞送。 因此，應用程式必須能夠處理重新遞送的訊息。

限制: 在 AUTO_ANCHACK 和 DUPS_OK_ANCHACK 模式中， JMS 不支援應用程式在訊息接聽器中擲出無法處理的異常狀況。 這表示不論訊息接聽器是否順利處理，一律會在訊息接聽器傳回時確認訊息 (前提是任何失敗都不嚴重，且不會阻止應用程式繼續執行)。 如果您需要更細微的訊息確認控制，請使用 CLIENT_ACKNOWLEDGE 或交易模式，這可讓應用程式完全控制確認功能。

用戶端確認

應用程式透過呼叫 Message 類別的 Acknowledge 方法來確認它收到的訊息。

應用程式可以個別確認接收每一個訊息，也可以接收一批訊息，並僅針對它接收的最後一則訊息呼叫 Acknowledge 方法。 當呼叫 Acknowledge 方法時，會確認自前次呼叫方法以來收到的所有訊息。

與任何這些確認模式一起使用時，應用程式可以透過呼叫「階段作業」類別的「回復」方法，停止並重新啟動階段作業中的訊息遞送。 已接收但先前未確認的訊息會重新遞送。 不過，它們可能不會以先前遞送的相同順序來遞送。 同時，較高優先順序的訊息可能已到達，部分原始訊息可能已過期。 在點對點網域中，部分原始訊息可能已由另一個應用程式使用。

應用程式可以檢查訊息的 JMSRedelivered 標頭欄位內容，來判斷是否正在重新遞送訊息。 應用程式會呼叫 Message 類別的 getJMSRedelivered() 方法來執行此動作。

在 *JMS* 應用程式中建立目的地

JMS 應用程式可以在執行時期使用階段作業來動態建立目的地，而不是從「 Java 命名和目錄介面 (JNDI)」名稱空間擷取目的地作為受管理物件。 應用程式可以使用統一資源識別碼 (URI) 來識別 IBM MQ 佇列或主題，並選擇性地指定 Queue 或 Topic 物件的一個以上內容。

使用階段作業來建立佇列物件

若要建立 Queue 物件，應用程式可以使用 Session 物件的 createQueue() 方法，如下列範例所示:

```
Session session;
.
Queue q1 = session.createQueue("Q1");
```

此程式碼會使用其所有內容的預設值來建立「佇列」物件。 此物件代表屬於本端佇列管理程式的 IBM MQ 佇列 Q1 。 此佇列可以是本端佇列、別名佇列或遠端佇列定義。

createQueue() 方法也接受佇列 URI 作為參數。 佇列 URI 是一個字串，指定 IBM MQ 佇列的名稱，並選擇性地指定擁有佇列的佇列管理程式名稱，以及佇列物件的一或多個內容。 下列陳述式包含佇列 URI 的範例:

```
Queue q2 = session.createQueue("queue://QM2/Q2?persistence=2&priority=5");
```

此陳述式所建立的「佇列」物件代表名為 Q2 的 IBM MQ 佇列，由名為 QM2 的佇列管理程式所擁有，且傳送至此目的地的所有訊息皆持續且優先順序為 5。 以這種方式識別的佇列管理程式可以是本端佇列管理程式或遠端佇列管理程式。 如果它是遠端佇列管理程式，則必須配置 IBM MQ ，以便在應用程式將訊息傳送至此目的地時， WebSphere MQ 可以將訊息從本端佇列管理程式遞送至佇列管理程式 QM2。 如需 URI 的相關資訊，請參閱 第 190 頁的『統一資源識別碼 (URI)』。

請注意， createQueue() 方法上的參數包含提供者特定的資訊。 因此，使用 createQueue() 方法來建立「佇列」物件，而不是從 JNDI 名稱空間中擷取「佇列」物件作為受管理物件，可能會使您的應用程式缺乏可攜性。

應用程式可以使用 Session 物件的 createTemporaryQueue () 方法來建立 TemporaryQueue 物件，如下列範例所示:

```
TemporaryQueue q3 = session.createTemporaryQueue();
```

雖然階段作業是用來建立暫時佇列，但暫時佇列的範圍是用來建立階段作業的連線。 任何連線的階段作業都可以建立暫時佇列的訊息產生者和訊息消費者。 暫時佇列會保留到連線結束或應用程式使用 TemporaryQueue.delete () 方法明確刪除暫時佇列 (取兩者中較早的者) 為止。

當應用程式建立暫時佇列時， IBM MQ classes for JMS 會在應用程式所連接的佇列管理程式中建立動態佇列。 Connection Factory 的 TEMPMODEL 內容指定用來建立動態佇列的模型佇列名稱，而 Connection Factory 的 TEMPQPREFIX 內容指定用來形成動態佇列名稱的字首。

使用階段作業來建立主題物件

若要建立 Topic 物件，應用程式可以使用 Session 物件的 createTopic() 方法，如下列範例所示:

```
Session session;
.
Topic t1 = session.createTopic("Sport/Football/Results");
```

此程式碼會使用其所有內容的預設值來建立 Topic 物件。 此物件代表稱為 Sport/Football/Results 的主題。

createTopic() 方法也接受主題 URI 作為參數。 主題 URI 是一個字串，用來指定主題的名稱，以及選擇性地指定 Topic 物件的一或多個內容。 下列程式碼包含主題 URI 的範例:

```
String uri = "topic://Sport/Tennis/Results?persistence=1&priority=0";
Topic t2 = session.createTopic(uri);
```

此代碼所建立的主題物件代表稱為 Sport/Tennis/Results 的主題，且傳送至此目的地的所有訊息都是非持續性且優先順序為 0。 如需主題 URI 的相關資訊，請參閱 第 190 頁的『統一資源識別碼 (URI)』。

請注意， createTopic() 方法上的參數包含提供者特定資訊。 因此，使用 createTopic() 方法來建立 Topic 物件，而不是從 JNDI 名稱空間中擷取 Topic 物件作為受管理物件，可能會使您的應用程式更缺乏可攜性。

應用程式可以使用 Session 物件的 createTemporaryTopic () 方法來建立 TemporaryTopic 物件，如下列範例所示:

```
TemporaryTopic t3 = session.createTemporaryTopic();
```

雖然階段作業是用來建立暫時主題，但暫時主題的範圍是用來建立階段作業的連線。 任何連線的階段作業都可以建立暫時主題的訊息產生者和訊息消費者。 在連線結束或應用程式使用 TemporaryTopic.delete () 方法明確刪除暫時主題之前，暫時主題仍會保留。

當應用程式建立暫時主題時， IBM MQ classes for JMS 會建立名稱以 TEMP/*tempTopicPrefix* 字元開頭的主題，其中 *tempTopicPrefix* 是 Connection Factory 的 TEMPTOPICPREFIX 內容值。

統一資源識別碼 (URI)

佇列 URI 是一個字串，用來指定 IBM MQ 佇列的名稱，以及選擇性地指定擁有佇列之佇列管理程式的名稱，以及應用程式所建立之「佇列」物件的一或多個內容。 主題 URI 是一個字串，指定主題的名稱，以及選擇性地指定應用程式所建立 Topic 物件的一或多個內容。

佇列 URI 具有下列格式:

```
queue://[ qMgrName ]/qName [? propertyName1 = propertyValue1
& propertyName2 = propertyValue2
&...]
```

主題 URI 具有下列格式:

```
topic://topicName [? propertyName1 = propertyValue1
& propertyName2 = propertyValue2
&...]
```

這些格式的變數具有下列意義:

QmgrName
擁有 URI 所識別之佇列的佇列管理程式名稱。

佇列管理程式可以是本端佇列管理程式或遠端佇列管理程式。 如果它是遠端佇列管理程式，則必須配置 IBM MQ ， 以便在應用程式將訊息傳送至佇列時， WebSphere MQ 可以將訊息從本端佇列管理程式遞送至遠端佇列管理程式。

如果未指定名稱，則會採用本端佇列管理程式。

qName
IBM MQ 佇列的名稱。

佇列可以是本端佇列、別名佇列或遠端佇列定義。

如需建立佇列名稱的規則，請參閱 命名 IBM MQ 物件的規則。

topicName
主題的名稱。

如需建立主題名稱的規則，請參閱 命名 IBM MQ 物件的規則。 避免使用萬用字元 +、#、* 及? 在主題名稱中。 當您訂閱這些字元時，包含這些字元的主題名稱可能會導致非預期的結果。 請參閱 結合主題字串。

propertyName1, propertyName2, ...
應用程式所建立「佇列」或「主題」物件的內容名稱。 第 191 頁的表 35 列出可在 URI 中使用的有效內容名稱。

如果未指定任何內容，則「佇列」或「主題」物件具有其所有內容的預設值。

propertyValue1, propertyValue2, ...
應用程式所建立「佇列」或「主題」物件的內容值。 第 191 頁的表 35 列出可在 URI 中使用的有效內容值。

方括弧 ([]) 表示選用元件，省略符號 (...) 表示內容名稱/值配對清單 (如果存在的話) 可以包含一或多個名稱/值配對。

第 191 頁的表 35 列出可在佇列和主題 URI 中使用的有效內容名稱和有效值。 雖然 IBM MQ JMS 管理工具會將符號常數用於內容值，但 URI 不能包含符號常數。

表 *35:* 在佇列和主題 *URI* 中使用的內容名稱和有效值

內容名稱	**說明**	**有效值**
CCSID	當 IBM MQ classes for JMS 將訊息轉遞至目的地時，如何表示訊息內文中的字元資料	• IBM MQ 支援的任何編碼字集 ID。
編碼	當 IBM MQ classes for JMS 將訊息轉遞至目的地時，如何呈現訊息內文中的數值資料	• IBM MQ 訊息描述子中 編碼 欄位的任何有效值。
到期	傳送至目的地之訊息的存活時間	• -2-如 send () 呼叫中所指定，或如果未在 send () 呼叫中指定，則為訊息產生者的預設存活時間。 • 0-傳送至目的地的訊息永不到期。 • 指定存活時間 (毫秒) 的正整數。
多重播送	使用與分配管理系統的即時連線時主題的多重播送設定	下列清單包含有效值。 與每一個值相關聯的是在 IBM MQ JMS 管理工具中使用的多重播送內容的對應值。 如需多重播送內容及其有效值的說明，請參閱 IBM MQ classes for JMS 物件的內容。 • -1-ASCF • 0 - 停用 • 3-NOTR • 5-可靠 • 7-已啟用
持續性	傳送至目的地之訊息的持續性	• -2-如 send () 呼叫上所指定，或如果未在 send () 呼叫上指定，則為訊息產生者的預設持續性。 • -1-如 IBM MQ 佇列或主題的 *DefPersistence* 屬性所指定。 • 1-非持續性。 • 2-持續。 • 3-等於 IBM MQ JMS 管理工具中使用的 PERSISTENCE 內容值 HIGH。 如需此值的說明，請參閱 第 215 頁的『JMS 持續訊息』。
priority	傳送至目的地的訊息優先順序	• -2-如 send () 呼叫中所指定，或如果未在 send () 呼叫中指定，則為訊息產生者的預設優先順序。 • -1-如 IBM MQ 佇列或主題的 *DefPriority* 屬性所指定。 • 0-9 範圍內的整數，指定傳送至目的地的訊息優先順序。

表 35: 在佇列和主題 *URI* 中使用的內容名稱和有效值 (繼續)		
內容名稱	**說明**	**有效值**
targetClient	傳送至目的地的訊息是否包含 MQRFH2 標頭	• 0-訊息包含 MQRFH2 標頭。 • 1-訊息不包含 MQRFH2 標頭。

例如，下列 URI 識別稱為 Q1 且由本端佇列管理程式所擁有的 IBM MQ 佇列。 使用此 URI 所建立的「佇列」物件具有其所有內容的預設值。

```
queue:///Q1
```

下列 URI 會識別稱為 Q2 的 IBM MQ 佇列，該佇列由稱為 QM2 的佇列管理程式所擁有。 所有傳送至此目的地的訊息都具有優先順序 6。 使用此 URI 所建立之「佇列」物件的其餘內容具有其預設值。

```
queue://QM2/Q2?priority=6
```

下列 URI 會識別稱為「運動/運動員/結果」的主題。 傳送至此目的地的所有訊息都是非持續性，且優先順序為 0。 使用這個 URI 所建立之 Topic 物件的其餘內容會有其預設值。

```
topic://Sport/Athletics/Results?persistence=1&priority=0
```

在 *JMS* 應用程式中傳送訊息

JMS 應用程式必須先為目的地建立 MessageProducer 物件，才能將訊息傳送至目的地。 為了將訊息傳送至目的地，應用程式會建立「訊息」物件，然後呼叫 MessageProducer 物件的 send () 方法。

應用程式使用 MessageProducer 物件來傳送訊息。 應用程式通常會為特定目的地 (可以是佇列或主題) 建立 MessageProducer 物件，以便所有使用訊息產生者傳送的訊息都傳送至相同目的地。 因此，應用程式必須先建立「佇列」或「主題」物件，才能建立 MessageProducer 物件。 如需如何建立「佇列」或「主題」物件的相關資訊，請參閱下列主題:

- 第 176 頁的『使用 JNDI 來擷取 JMS 或 Jakarta Messaging 應用程式中的受管理物件』
- 第 177 頁的『使用 IBM JMS 延伸』
- 第 184 頁的『使用 IBM MQ JMS 延伸』
- 第 188 頁的『在 JMS 應用程式中建立目的地』

如果要建立 MessageProducer 物件，應用程式會使用 Session 物件的 createProducer() 方法，如下列範例所示:

```
MessageProducer producer = session.createProducer(destination);
```

參數 `destination` 是應用程式先前建立的「佇列」或「主題」物件。

應用程式必須先建立訊息物件，然後才能傳送訊息。 訊息內文包含應用程式資料，且 JMS 定義五種類型的訊息內文:

- 位元組
- 對映
- 物件
- 串流
- 文字

每一種類型的訊息內文都有自己的 JMS 介面 (即「訊息」介面的子介面)，以及「階段作業」介面中用來建立具有該類型內文之訊息的方法。 例如，文字訊息的介面稱為 TextMessage，且應用程式使用 Session 物件的 createTextMessage () 方法來建立文字訊息，如下列陳述式所示:

```
TextMessage outMessage = session.createTextMessage(outString);
```

如需訊息及訊息內文的相關資訊，請參閱 第 122 頁的『JMS 訊息』。

如果要傳送訊息，應用程式會使用 MessageProducer 物件的 send () 方法，如下列範例所示:

```
producer.send(outMessage);
```

應用程式可以使用 send () 方法，在任一傳訊網域中傳送訊息。 目的地的本質會決定使用哪一個傳訊網域。 不過， TopicPublisher 是特定於發佈/訂閱網域之 MessageProducer 的子介面，也有一個 publish () 方法，可用來取代 send () 方法。 這兩種方法功能相同。

應用程式可以建立沒有指定目的地的 MessageProducer 物件。 在此情況下，應用程式必須在呼叫 send () 方法時指定目的地。

如果應用程式在交易內傳送訊息，則在確定交易之前不會將訊息遞送至其目的地。 這表示應用程式無法在相同交易內傳送訊息並接收訊息的回覆。

可以配置目的地，以便在應用程式傳送訊息給它時， IBM MQ classes for JMS 會轉遞訊息並將控制權傳回給應用程式，而不會判斷佇列管理程式是否已安全地接收訊息。 這有時稱為 非同步放置。 如需相關資訊，請參閱 第 268 頁的『在 IBM MQ classes for JMS 中非同步放置訊息』。

在 *JMS* 應用程式中接收訊息

應用程式使用訊息消費者來接收訊息。 可延續主題訂閱者是訊息消費者，可接收傳送至目的地的所有訊息，包括在消費者非作用中時所傳送的訊息。 應用程式可以使用訊息選取器來選取要接收的訊息，並且可以使用訊息接聽器來非同步接收訊息。

應用程式使用 MessageConsumer 物件來接收訊息。 應用程式會為特定目的地 (可以是佇列或主題) 建立 MessageConsumer 物件，以便從相同目的地接收使用訊息消費者接收的所有訊息。 因此，應用程式必須先建立「佇列」或「主題」物件，才能建立 MessageConsumer 物件。 如需如何建立「佇列」或「主題」物件的相關資訊，請參閱下列主題:

- 第 176 頁的『使用 JNDI 來擷取 JMS 或 Jakarta Messaging 應用程式中的受管理物件』
- 第 177 頁的『使用 IBM JMS 延伸』
- 第 184 頁的『使用 IBM MQ JMS 延伸』
- 第 188 頁的『在 JMS 應用程式中建立目的地』

若要建立 MessageConsumer 物件，應用程式會使用 Session 物件的 createConsumer() 方法，如下列範例所示:

```
MessageConsumer consumer = session.createConsumer(destination);
```

參數 `destination` 是應用程式先前建立的「佇列」或「主題」物件。

然後應用程式會使用 MessageConsumer 物件的 receive () 方法，從目的地接收訊息，如下列範例所示:

```
Message inMessage = consumer.receive(1000);
```

receive () 呼叫的參數指定如果沒有立即可用的訊息，方法等待適當訊息到達的時間 (毫秒)。 如果您省略此參數，則會無限期地封鎖呼叫，直到適當的訊息到達為止。 如果您不想要應用程式等待訊息，請改用 receiveNoWait () 方法。

receive () 方法會傳回特定類型的訊息。 例如，當應用程式接收文字訊息時， receive () 呼叫所傳回的物件是 TextMessage 物件。

不過， receive () 呼叫所傳回之物件的宣告類型是「訊息」物件。 因此，為了從剛收到的訊息內文擷取資料，應用程式必須從 Message 類別強制轉型為更具體的子類別，例如 TextMessage。 如果訊息類型不明，

應用程式可以使用 instanceof 運算子來判斷類型。 在強制轉型之前，應用程式最好先判斷訊息類型，以便能循序處理錯誤。

下列程式碼使用 instanceof 運算子，並顯示如何從文字訊息內文擷取資料:

```
if (inMessage instanceof TextMessage) {
  String replyString = ((TextMessage) inMessage).getText();
  .
  .
  .
} else {
  // Print error message if Message was not a TextMessage.
  System.out.println("Reply message was not a TextMessage");
}
```

如果應用程式在交易內傳送訊息，則在確定交易之前不會將訊息遞送至其目的地。 這表示應用程式無法在相同交易內傳送訊息並接收訊息的回覆。

如果訊息消費者從配置成先讀的目的地接收訊息，當應用程式結束時，會捨棄先讀緩衝區中的任何非持續訊息。

在發佈/訂閱網域中， JMS 會識別兩種類型的訊息消費者: 不可延續主題訂閱者和可延續主題訂閱者，這在下列兩節中有說明。

不可延續主題訂閱者

不可延續的主題訂閱者只會接收訂閱者處於作用中狀態時所發佈的那些訊息。 當應用程式建立不可延續的主題訂閱者時，即會啟動不可延續的訂閱，當應用程式關閉訂閱者時，或當訂閱者落在範圍之外時，即會結束不可延續的訂閱。 作為 IBM MQ classes for JMS 中的延伸，不可延續主題訂閱者也會接收保留的發佈。

如果要建立不可延續的主題訂閱者，應用程式可以使用與網域無關的 createConsumer() 方法，並指定 Topic 物件作為目的地。 或者，應用程式可以使用網域特定的 createSubscriber() 方法，如下列範例所示:

```
TopicSubscriber subscriber = session.createSubscriber(topic);
```

參數 topic 是應用程式先前建立的「主題」物件。

可延續主題訂閱者

限制: 使用分配管理系統的即時連線時，應用程式無法建立可延續主題訂閱者。

可延續主題訂閱者會接收在可延續訂閱生命期間發佈的所有訊息。 這些訊息包括訂閱者處於非作用中狀態時所發佈的所有訊息。 作為 IBM MQ classes for JMS 中的延伸，可延續主題訂閱者也會接收保留的發佈資訊。

如果要建立可延續主題訂閱者，應用程式會使用 Session 物件的 createDurableSubscriber () 方法，如下列範例所示:

```
TopicSubscriber subscriber = session.createDurableSubscriber(topic, "D_SUB_000001");
```

在 createDurableSubscriber () 呼叫上，第一個參數是應用程式先前建立的 Topic 物件，第二個參數是用來識別可延續訂閱的名稱。

用來建立可延續主題訂閱者的階段作業必須有相關聯的用戶端 ID。 與階段作業相關聯的用戶端 ID 與用來建立階段作業之連線的用戶端 ID 相同。 您可以設定 ConnectionFactory 物件的 CLIENTID 內容來指定用戶端 ID。 或者，應用程式可以呼叫 Connection 物件的 setClientID () 方法來指定用戶端 ID。

用來識別可延續訂閱的名稱只能在用戶端 ID 內是唯一的，因此用戶端 ID 會構成可延續訂閱完整唯一 ID 的一部分。 若要繼續使用先前建立的可延續訂閱，應用程式必須使用與可延續訂閱具有相同用戶端 ID 的階段作業，並使用相同的訂閱名稱來建立可延續主題訂閱者。

當應用程式使用目前不存在可延續訂閱的用戶端 ID 和訂閱名稱來建立可延續主題訂閱者時，即會啟動可延續訂閱。 不過，當應用程式關閉可延續主題訂閱者時，可延續訂閱不會結束。 如果要結束可延續訂閱，應

用程式必須呼叫 Session 物件的 unsubscribe () 方法，其用戶端 ID 與可延續訂閱相關聯的用戶端 ID 相同。 unsubscribe () 呼叫上的參數是訂閱名稱，如下列範例所示:

```
session.unsubscribe("D_SUB_000001");
```

可延續訂閱的範圍是佇列管理程式。 如果某個佇列管理程式上存在可延續訂閱，且連接至另一個佇列管理程式的應用程式會建立具有相同用戶端 ID 及訂閱名稱的可延續訂閱，則這兩個可延續訂閱完全獨立。

訊息選取器

應用程式可以指定連續的 receive () 呼叫只傳回那些符合特定準則的訊息。 建立 MessageConsumer 物件時，應用程式可以指定「結構化查詢語言 (SQL)」表示式，以決定要擷取哪些訊息。 此 SQL 表示式稱為 訊息選取器。 訊息選取器可以包含 JMS 訊息標頭欄位及訊息內容的名稱。 如需如何建構訊息選取器的相關資訊，請參閱 第 122 頁的『JMS 中的訊息選取器』。

下列範例顯示應用程式如何根據稱為 myProp 的使用者定義內容來選取訊息:

```
MessageConsumer consumer;
.
consumer = session.createConsumer(destination, "myProp = 'blue'");
```

JMS 規格不容許應用程式變更訊息消費者的訊息選取器。 在應用程式建立具有訊息選取器的訊息消費者之後，訊息選取器會在該消費者的生命期限內保留。 如果應用程式需要多個訊息選取器，應用程式必須為每一個訊息選取器建立一個訊息消費者。

請注意，當應用程式連接至第 7 版佇列管理程式時， Connection Factory 的 MSGSELECTION 內容沒有作用。 為了最佳化效能，所有訊息選擇都由佇列管理程式完成。

暫停本端發佈

應用程式可以建立訊息消費者，以忽略在消費者自己連線上發佈的發佈。 應用程式可以透過將 createConsumer() 呼叫上的第三個參數設為 `true` 來執行此動作，如下列範例所示:

```
MessageConsumer consumer = session.createConsumer(topic, null, true);
```

在 createDurableSubscriber () 呼叫中，應用程式會將第四個參數設為 `true` 來執行這個動作，如下列範例所示:

```
String selector = "company = 'IBM'";
TopicSubscriber subscriber = session.createDurableSubscriber(topic, "D_SUB_000001",
                                                              selector, true);
```

非同步遞送訊息

應用程式可以向訊息消費者登錄訊息接聽器，以非同步方式接收訊息。 訊息接聽器有一個稱為 onMessage 的方法，當有適當的訊息可用且其目的是處理訊息時，會非同步呼叫此方法。 下列程式碼說明機制:

JM 3.0 V 9.3.0 V 9.3.0

```
import jakarta.jms.*;

public class MyClass implements MessageListener
{
  // The method that is called asynchronously when a suitable message is available
  public void onMessage(Message message)
  {
    System.out.println("Message is "+message);

    // The code to process the message
    .
    .
    .
  }
```

```
}
.
.
.
// Main program (possibly in another class)
.
// Creating the message listener
MyClass listener = new MyClass();

// Registering the message listener with a message consumer
consumer.setMessageListener(listener);

// The main program now continues with other processing
```

JMS 2.0

```
import javax.jms.*;

public class MyClass implements MessageListener
{
  // The method that is called asynchronously when a suitable message is available
  public void onMessage(Message message)
  {
    System.out.println("Message is "+message);

    // The code to process the message
    .
    .
    .
  }
}
.
.
.
// Main program (possibly in another class)
.
// Creating the message listener
MyClass listener = new MyClass();

// Registering the message listener with a message consumer
consumer.setMessageListener(listener);

// The main program now continues with other processing
```

應用程式可以使用階段作業來使用 receive () 呼叫同步接收訊息，或使用訊息接聽器非同步接收訊息，但不能同時使用這兩者。 如果應用程式需要同步及非同步接收訊息，則必須建立個別階段作業。

一旦將階段作業設定為非同步接收訊息，就無法在該階段作業或從該階段作業建立的物件上呼叫下列方法:

- MessageConsumer.receive ()
- MessageConsumer.receive (long)
- MessageConsumer.receiveNoWait ()
- Session.acknowledge()
- MessageProducer.send (Destination , Message)
- MessageProducer.send (Destination , Message , int , int , long)
- MessageProducer.send (Message)
- MessageProducer.send (Message , int , int , long)
- MessageProducer.send (Destination , Message , CompletionListener)
- MessageProducer.send (Destination , Message , int , int , long , CompletionListener)
- MessageProducer.send (Message , CompletionListener)
- MessageProducer.send (Message , int , int , long , CompletionListener)
- Session.commit()
- Session.createBrowser(佇列)
- Session.createBrowser(Queue , String)
- Session.createBytesMessage()

- Session.createConsumer(目的地)
- Session.createConsumer(Destination , String , boolean)
- Session.createDurableSubscriber(Topic , String)
- Session.createDurableSubscriber(Topic , String , String , boolean)
- Session.createMapMessage()
- Session.createMessage()
- Session.createObjectMessage()
- Session.createObjectMessage(可序列化)
- Session.createProducer(目的地)
- Session.createQueue(字串)
- Session.createStreamMessage()
- Session.createTemporaryQueue()
- Session.createTemporaryTopic()
- Session.createTextMessage()
- Session.createTextMessage(字串)
- Session.createTopic()
- Session.getAcknowledgeMode()
- Session.getMessageListener()
- Session.getTransacted()
- Session.rollback()
- Session.unsubscribe(String)

如果呼叫下列任何方法，則為包含訊息的 JMSException:

```
JMSCC0033: 非同步使用階段作業時，不允許同步方法呼叫: 'method name '
```

會擲出。

接收有害訊息

應用程式可以接收無法處理的訊息。 可能有數個無法處理訊息的原因，例如訊息的格式可能不正確。 這類訊息說明為有害訊息，需要特殊處理，以防止遞迴處理訊息。

如需如何處理有害訊息的詳細資料，請參閱 第 199 頁的『在 IBM MQ classes for JMS 中處理有害訊息』。

調整緩衝區大小以符合收到的訊息

當非 JMS 應用程式從 IBM MQ 收到訊息時，應用程式必須提供訊息緩衝區，才能將訊息寫入其中。 JMS 應用程式不需要手動建立緩衝區。 IBM MQ classes for JMS 會自動建立並調整訊息緩衝區大小，以符合所接收訊息的大小。 對於大部分應用程式，自動受管理緩衝區為應用程式開發人員提供適當的效能與便利平衡。 在某些情況下，最好手動指定訊息緩衝區的起始大小。 IBM MQ JMS 接收緩衝區的預設起始大小為 4 KB。 如果應用程式一律會接收大小為 256 KB 的訊息，則最好將起始緩衝區大小配置為 256 KB。 這可以避免 IBM MQ classes for JMS 在將訊息重新調整為 256 KB 並順利接收之前嘗試並無法將訊息接收到 4 KB 緩衝區。 對於用戶端連接的應用程式，這可以避免在 IBM MQ classes for JMS 判定要使用的正確緩衝區大小時可能浪費的網路來回轉換。

透過將 com.ibm.mq.jmqi.defaultMaxMsgSize Java 內容設定為您選擇的值 (以位元組為單位) ，可以配置起始緩衝區大小。 請注意，此內容會影響在 Java Virtual Machine 內執行的所有 IBM MQ JMS 應用程式，因此請小心不要對接收不同大小的訊息的其他訊息消費者產生不利影響。

如果收到數則小於所配置大小的訊息， IBM MQ classes for JMS 仍會嘗試自動減少緩衝區大小。 依預設，如果收到 10 則訊息都小於緩衝區大小，則會發生此情況。 例如，如果在大小為 128 KB 的列中接收 10 則訊息，則緩衝區會從 256 KB 減少至 128 KB。 然後，當接收到較大的訊息時，它會再次增加。 可以配置在緩衝區大小減少之前必須接收的訊息數。 例如，如果已知應用程式會接收五個大型訊息，接著接收 10 個較小

的訊息，然後再接收另外五個大型訊息，則這可能很有用。 使用預設值，在收到 10 則較小的訊息之後，會減少緩衝區，而對於較大的訊息，則需要再次增加。 Java 系統內容 com.ibm.mq.jmqi.smallMsgBufferReductionThreshold 可以設為在減少緩衝區大小之前必須接收的訊息數。 在此範例中，可以將它設為 20 ，以防止 10 則較小的訊息減少緩衝區大小。

內容可以彼此獨立設定。 例如，您可以選擇將起始緩衝區大小保留為其預設值 4 KB ，但增加 com.ibm.mq.jmqi.smallMsgBufferReductionThreshold 的值，以便一旦增加緩衝區大小，它會保留該大小更長。

如果在 MQI 統計資料記錄中看到 JMS 應用程式的大量 MQRC_TRUNCATED_MSG_FAILED (2080) 回覆碼，則表示您可以為那些應用程式配置較高的起始緩衝區大小，或減少緩衝區大小的頻率。 不過，請務必注意，對於長時間執行的應用程式，您可能只會看到很少的 MQRC_TRUNCATED_MSG_FAILED 回覆碼。 這是因為通常會在收到第一個大型訊息之後立即將緩衝區增加至正確的大小，而且除非收到一些較小的訊息，否則不會減少大小。 因此，大量 MQRC_TRUNCATED_MSG_FAILED 可能指出其他不良應用程式作法，例如連接至 IBM MQ 以在中斷連線之前只接收一或兩則訊息。

擷取訂閱使用者資料

如果 IBM MQ classes for JMS 應用程式從佇列耗用的訊息是由管理定義的可延續訂閱所放置，則應用程式需要存取與訂閱相關聯的使用者資料資訊。 此資訊會以內容形式新增至訊息。

從包含 RFH2 標頭及 MQPS 資料夾的佇列中耗用訊息時，與 Sud 索引鍵相關聯的值 (如果存在的話) 會以「字串」內容新增至傳回 IBM MQ classes for JMS 應用程式的 JMS 訊息物件。 若要啟用從訊息擷取此內容， JmsConstants 介面中的常數 JMS_IBM_SUBSCRIPTION_USER_DATA 可以與下列方法搭配使用，以取得訂閱使用者資料:

- JM 3.0 V 9.3.0 V 9.3.0 jakarta.jms.Message.getStringProperty(java.lang.String)
- JMS 2.0 javax.jms.Message.getStringProperty(java.lang.String)

在下列範例中，使用 MQSC 指令 **DEFINE SUB** 來定義管理可延續訂閱:

```
DEFINE SUB('MY.SUBCRIPTION') TOPICSTR('PUBLIC') DEST('MY.SUBSCRIPTION.Q')
USERDATA('Administrative durable subscription to put message to the queue MY.SUBSCRIPTION.Q')
```

發佈至主題字串 PUBLIC 的訊息副本會放入佇列 MY.SUBSCRIPTION.Q。 然後，與可延續訂閱相關聯的使用者資料會作為內容新增至訊息，該訊息儲存在具有索引鍵 Sud 之 RFH2 標頭的 MQPS 資料夾中。

IBM MQ classes for JMS 應用程式可以呼叫:

JM 3.0 V 9.3.0 V 9.3.0
```
jakarta.jms.Message.getStringProperty(JmsConstants.JMS_IBM_SUBSCRIPTION_USER_DATA);
```

JMS 2.0 `javax.jms.Message.getStringProperty(JmsConstants.JMS_IBM_SUBSCRIPTION_USER_DATA);`

然後會傳回下列字串:

```
Administrative durable subscription to put message to the queue MY.SUBSCRIPTION.Q
```

相關概念

第 126 頁的『MQRFH2 標頭和 JMS』

相關工作

定義管理訂閱

相關參考

DEFINE SUB

JmsConstants 介面

關閉 *IBM MQ classes for JMS* 應用程式

IBM MQ classes for JMS 應用程式在停止之前明確關閉某些 JMS 物件是很重要的。 可能不會呼叫終止程式，因此請不要依賴它們來釋放資源。 不容許應用程式以作用中的壓縮追蹤來終止。

僅記憶體回收無法及時釋放所有 IBM MQ classes for JMS 和 IBM MQ 資源，尤其是當應用程式在階段作業層次或更低層次建立許多短期存活 JMS 物件時。 因此，應用程式必須關閉不再需要的「連線」、「階段作業」、 MessageConsumer 或 MessageProducer 物件。

如果應用程式結束而未關閉「連線」，則會對所有連線的交易階段作業進行隱含回復。 若要確保應用程式所做的任何變更都已確定，請在關閉應用程式之前明確關閉「連線」。

請勿在應用程式中使用終止程式來關閉 JMS 物件。 因為可能未呼叫終止程式，所以可能不會釋放資源。 當關閉「連線」時，它會關閉從它建立的所有「階段作業」。 同樣地，當「階段作業」關閉時，會關閉從「階段作業」建立的 MessageConsumers 和 MessageProducers 。 不過，請考量明確關閉「階段作業」、 MessageConsumers 及 MessageProducers ， 以確保及時釋放資源。

如果啟動追蹤壓縮，則 System.Halt() 關機及異常、不受控制的 JVM 終止可能會導致追蹤檔毀損。 可能的話，當您已收集所需的追蹤資訊時，請關閉追蹤機能。 如果您要追蹤應用程式直到異常結束，請使用未經壓縮的追蹤輸出。

註: 為了切斷與佇列管理程式的連線， JMS 應用程式會對連線物件呼叫 close () 方法。

在 *IBM MQ classes for JMS* 中處理有害訊息

有害訊息是指接收端應用程式無法處理的訊息。 如果有害訊息遞送至應用程式並回復指定次數，則 IBM MQ classes for JMS 可以將它移至取消佇列。

有害訊息是指接收端應用程式無法處理的訊息。 訊息可能具有非預期的類型，或包含應用程式邏輯無法處理的資訊。 如果有毒訊息遞送至應用程式，應用程式將無法處理它，且會將它回復到其來源的佇列。 依預設， IBM MQ classes for JMS 會反覆地將訊息重新遞送至應用程式。 這可能會導致應用程式在迴圈中持續嘗試處理有毒訊息並回復。

為了防止發生這種情況， IBM MQ classes for JMS 可以偵測有毒訊息，並將它們移至替代目的地。 若要這麼做， IBM MQ classes for JMS 會利用下列內容:

- 已偵測到訊息的 MQMD 內 BackoutCount 欄位的值。
- 包含訊息之輸入佇列的 IBM MQ 佇列屬性 **BOTHRESH** (取消臨界值) 及 **BOQNAME** (取消重新排入佇列佇列)。

每當應用程式回復訊息時，佇列管理程式會自動增加訊息的 BackoutCount 欄位值。

當 IBM MQ classes for JMS 偵測到 BackoutCount 大於零的訊息時，它們會比較 BackoutCount 的值與 **BOTHRESH** 屬性的值。

- 如果 BackoutCount 小於 **BOTHRESH** 屬性的值，則 IBM MQ classes for JMS 會將它遞送至應用程式以進行處理。
- 不過，如果 BackoutCount 大於或等於 **BOTHRESH**，則會將訊息視為有害訊息。 在此狀況下， IBM MQ classes for JMS 會將訊息移至 **BOQNAME** 屬性指定的佇列。 如果無法將訊息放入取消佇列，則視訊息的報告選項而定，會將訊息移至佇列管理程式的無法傳送的郵件佇列或捨棄。

註:

- 如果 **BOTHRESH** 屬性保留其預設值 0，則會停用有害訊息處理。 這表示任何有害訊息都會放回輸入佇列。
- 另一件要注意的事是第一次偵測到 BackoutCount 大於零的訊息時， IBM MQ classes for JMS 會查詢佇列的 **BOTHRESH** 及 **BOQNAME** 屬性。 然後會快取這些屬性的值，並在 IBM MQ classes for JMS 遇到 BackoutCount 大於零的訊息時使用。

配置系統以執行有毒訊息處理

IBM MQ classes for JMS 在查詢 **BOTHRESH** 和 **BOQNAME** 屬性時使用的佇列取決於所執行傳訊的樣式:

- 對於點對點傳訊，這是基礎本端佇列。 當 JMS 應用程式使用別名佇列或叢集佇列中的訊息時，這很重要。
- 對於發佈/訂閱傳訊，會建立受管理佇列來保留應用程式的訊息。 IBM MQ classes for JMS 會查詢受管理佇列，以判定 **BOTHRESH** 及 **BOQNAME** 屬性的值。

  受管理佇列是從與應用程式已訂閱之「主題」物件相關聯的模型佇列建立，並從模型佇列繼承 **BOTHRESH** 及 **BOQNAME** 屬性的值。 使用的模型佇列取決於接收端應用程式是否已取出可延續或不可延續訂閱:

- 用於可延續訂閱的模型佇列由主題的 **MDURMDL** 屬性指定。 此屬性的預設值為 SYSTEM.DURABLE.MODEL.QUEUE。
- 對於不可延續訂閱，使用的模型佇列由 **MNDURMDL** 屬性指定。 **MNDURMDL** 屬性的預設值是 SYSTEM.NDURABLE.MODEL.QUEUE。

查詢 **BOTHRESH** 和 **BOQNAME** 屬性時， IBM MQ classes for JMS:

- 開啟本端佇列或別名佇列的目標佇列。
- 查詢 **BOTHRESH** 及 **BOQNAME** 屬性。
- 關閉本端佇列或別名佇列的目標佇列。

開啟本端佇列或別名佇列的目標佇列時所使用的開啟選項，視所使用的 IBM MQ classes for JMS 版本而定:

- 若為 IBM MQ 9.1.0 Fix Pack 1 及更早版本中的 IBM MQ classes for JMS ，或 IBM MQ 9.1.1，如果本端佇列或別名佇列的目標佇列是叢集佇列，則 IBM MQ classes for JMS 會使用 MQOO_INPUT_AS_Q_DEF、MQOO_INQUIRE 及 MQOO_FAIL_IF_QUIESCING 選項開啟佇列。 這表示執行接收端應用程式的使用者必須具有叢集佇列本端實例的查詢及存取權。

  IBM MQ classes for JMS 會使用開啟選項 MQOO_INQUIRE 及 MQOO_FAIL_IF_QUIESCING 來開啟所有其他類型的本端佇列。 為了讓 IBM MQ classes for JMS 查詢屬性值，執行接收端應用程式的使用者必須具有本端佇列的查詢存取權。
- 在 IBM MQ 9.1.0 Fix Pack 2 及更新版本中使用 IBM MQ classes for JMS 時，或在 IBM MQ 9.1.2 及更新版本中使用時，執行接收端應用程式的使用者必須對本端佇列具有查詢存取權，而不論佇列類型為何。

若要將有害訊息移至取消佇列或佇列管理程式的無法傳送郵件佇列，您必須授與執行應用程式 put 及 passall 權限的使用者。

處理同步應用程式的有害訊息

如果應用程式透過呼叫下列其中一種方法來同步接收訊息，則 IBM MQ classes for JMS 會在應用程式嘗試取得訊息時，將作用中工作單元內的有害訊息重新排入佇列:

- JMSConsumer.receive()
- JMSConsumer.receive(長逾時)
- JMSConsumer.receiveBody(Class<T> c)
- JMSConsumer.receiveBody(Class<T> c, long timeout)
- JMSConsumer.receiveBodyNoWait Class<T> c)
- JMSConsumer.receiveNoWait()
- MessageConsumer.receive ()
- MessageConsumer.receive (長逾時)
- MessageConsumer.receiveNoWait ()
- QueueReceiver.receive ()
- QueueReceiver.receive (long timeout)
- QueueReceiver。receiveNoWait ()
- TopicSubscriber.receive ()
- TopicSubscriber.receive (長逾時)
- TopicSubscriber.receiveNoWait ()

這表示如果應用程式正在使用交易式 JMS 環境定義或階段作業，則在確定交易之前，不會確定將訊息移至取消佇列。

如果 **BOTHRESH** 屬性設為零以外的值，則也應該設定 **BOQNAME** 屬性。 如果 **BOTHRESH** 設為大於零的值，且尚未設定 **BOQNAME** ，則行為由訊息的報告選項決定:

- 如果訊息已設定報告選項 MQRO_DISCARD_MSG ，則會捨棄該訊息。

- 如果訊息已指定報告選項 MQRO_DEAD_LETTER_Q，則 IBM MQ classes for JMS 會嘗試將訊息移至佇列管理程式的無法傳送郵件佇列。
- 如果訊息未設定 MQRO_DISCARD_MSG 或 MQRO_DEAD_LETTER_Q，則 IBM MQ classes for JMS 會嘗試將訊息放入佇列管理程式的無法傳送郵件的佇列。

如果嘗試將訊息放置到無法傳送的郵件佇列因某些原因而失敗，則會根據接收端應用程式是否使用交易式或非交易式 JMS 環境定義或階段作業來決定訊息的狀況:

- 如果接收端應用程式正在使用交易式 JMS 環境定義或階段作業，且已確定交易，則會捨棄訊息。
- 如果接收端應用程式使用交易式 JMS 環境定義或階段作業，並回復交易，則會將訊息傳回給輸入佇列。
- 如果接收端應用程式已建立非交易式 JMS 環境定義或階段作業，則會捨棄訊息。

處理非同步應用程式的有害訊息

如果應用程式透過 MessageListener 非同步接收訊息，則 IBM MQ classes for JMS 會將有害訊息重新排入佇列，而不會影響訊息遞送。 重新排入佇列程序是在與實際訊息遞送至應用程式相關聯的任何工作單元之外進行。

如果 **BOTHRESH** 設為大於零的值，且尚未設定 **BOQNAME**，則行為由訊息的報告選項決定:

- 如果訊息已設定報告選項 MQRO_DISCARD_MSG，則會捨棄該訊息。
- 如果訊息已指定報告選項 MQRO_DEAD_LETTER_Q，則 IBM MQ classes for JMS 會嘗試將訊息移至佇列管理程式的無法傳送郵件佇列。
- 如果訊息未設定 MQRO_DISCARD_MSG 或 MQRO_DEAD_LETTER_Q，則 IBM MQ classes for JMS 會嘗試將訊息放入佇列管理程式的無法傳送郵件的佇列。

如果嘗試將訊息放置到無法傳送的郵件佇列失敗，則 IBM MQ classes for JMS 會將訊息傳回到輸入佇列。

如需啟動規格和 ConnectionConsumers 如何處理有害訊息的相關資訊，請參閱 在 ASF 中從佇列移除訊息。

將訊息移至取消佇列時發生的狀況

當有害訊息重新排入取消重新排入佇列時， IBM MQ classes for JMS 會在其中新增 RFH2 標頭 (如果它還沒有的話)，並更新訊息描述子 (MQMD) 內的部分欄位。

如果有害訊息包含 RFH2 標頭 (例如，因為它是 JMS 訊息)，當將訊息移至取消重新排入佇列時， IBM MQ classes for JMS 會變更 MQMD 內的下列欄位:

- BackoutCount 欄位會重設為零。
- 訊息的「期限」欄位會更新，以反映 JMS 應用程式收到有害訊息時剩餘的期限。

如果有害訊息不包含 RFH2 標頭，則 IBM MQ classes for JMS 會在 MQMD 中新增一個並更新下列欄位，以作為取消處理的一部分:

- BackoutCount 欄位會重設為零。
- 訊息的「期限」欄位會更新，以反映 JMS 應用程式收到有害訊息時剩餘的期限。
- 訊息的「格式」欄位會變更為 MQHRF2。
- CCSID 欄位變更為 1208。
- 「編碼」欄位修改為 273。

此外，有害訊息中的 CCSID 及「編碼」欄位會複製到 RFH2 標頭的 CCSID 及「編碼」欄位，以確保取消重新排入佇列中的訊息標頭鏈結正確。

相關概念

第 282 頁的『處理 ASF 中的有害訊息』

在「應用程式伺服器機能」內，有毒訊息處理的處理方式與 IBM MQ classes for JMS 中的其他位置略有不同。

IBM MQ classes for JMS 中的異常狀況

IBM MQ classes for JMS 應用程式必須處理 JMS API 呼叫所擲出或遞送至異常狀況處理程式的異常狀況。

IBM MQ classes for JMS 會透過擲出異常狀況來報告執行時期問題。 擲出的異常狀況類型，以及必須處理這些異常狀況的方式，取決於應用程式所使用的 JMS 規格版本:

- 在 JMS 1.1 及更早版本中定義的介面上擲出已檢查異常狀況的方法。 這些異常狀況的基礎類別是 JMSException。 如需如何處理已檢查異常狀況的相關資訊，請參閱 第 202 頁的『處理已檢查的異常狀況』。
- 在 JMS 2.0 中新增的介面上的方法會擲出未檢查的異常狀況。 這些異常狀況的基礎類別是 JMSRuntimeException。 如需如何處理未檢查異常狀況的相關資訊，請參閱 第 205 頁的『處理未檢查的異常狀況』。

您也可以向 JMS Connection 或 JMSContext 登錄 ExceptionListener。 然後，如果偵測到與佇列管理程式的連線有問題，或嘗試非同步遞送訊息時發生問題，則 JMS 的 MQ 類別會通知 ExceptionListener。 如需相關資訊，請參閱 第 208 頁的『ExceptionListeners』。

相關概念

IBM MQ classes for JMS

相關參考

ASYNCEXCEPTION

處理已檢查的異常狀況

在 JMS 1.1 或更早版本中定義的介面上擲出已檢查異常狀況的方法。 這些異常狀況的基礎類別是 JMSException。 因此，捕捉 JMSExceptions 提供一般方式來處理這些類型的異常狀況。

每一個 JMSException 都會封裝下列資訊:

- 提供者特定的異常狀況訊息，您的應用程式可以透過呼叫 Throwable.getMessage() 方法來取得該訊息。
- 提供者特定的錯誤碼，您的應用程式可以透過呼叫 JMSException.getErrorCode() 方法來取得該錯誤碼。
- 鏈結的異常狀況。 JMS 1.1 API 呼叫所擲出的異常狀況，通常是因為另一個鏈結至此異常狀況的異常狀況所報告的較低層次問題。 您的應用程式可以呼叫 JMSException.getLinkedException() 方法或 Throwable.getCause() 方法來取得鏈結的異常狀況。

當您使用 JMS 1.1 API 時， IBM MQ classes for JMS 所擲出的大部分異常狀況都是 JMSException 的子類別實例。 這些子類別實作 com.ibm.msg.client.jms.JmsExceptionDetail 介面，其提供下列其他資訊:

- 異常狀況訊息的說明。 您的應用程式可以呼叫 JmsExceptionDetail.getExplanation() 方法來取得此訊息。
- 異常狀況的建議使用者回應。 您的應用程式可以呼叫 JmsExceptionDetail.getUserAction() 方法來取得此訊息。
- 在異常狀況訊息中插入訊息的索引鍵。 您的應用程式可以透過呼叫 JmsExceptionDetail.getKeys() 方法來取得所有索引鍵的反覆運算子。
- 訊息會插入異常狀況訊息中。 例如，訊息插入可能是導致異常狀況的佇列名稱，而且您的應用程式存取該名稱可能很有用。 您的應用程式可以呼叫 JmsExceptionDetail.getValue() 方法，以取得對應於指定金鑰的訊息插入。

如果沒有可用的詳細資料， JmsExceptionDetail 介面中的所有方法都會傳回空值。

例如，如果應用程式嘗試為不存在的 IBM MQ 佇列建立訊息產生者，則會擲出含有下列資訊的異常狀況:

```
Message : JMSWMQ2008: Failed to open MQ queue 'Q_test'.
Class : class com.ibm.msg.client.jms.DetailedInvalidDestinationException
Error Code : JMSWMQ2008
Explanation : JMS attempted to perform an MQOPEN, but IBM MQ reported an
              error.
User Action : Use the linked exception to determine the cause of this error. Check
              that the specified queue and queue manager are defined correctly.
```

擲出的異常狀況 com.ibm.msg.client.jms.DetailedInvalidDestinationException 是下列類別的子類別，並實作 com.ibm.msg.client.jms.JmsExceptionDetail 介面。

- JM 3.0 V 9.3.0 V 9.3.0 `jakarta.jms.InvalidDestinationException`
- JMS 2.0 `javax.jms.InvalidDestinationException`

鏈結的異常狀況

鏈結的異常狀況提供執行時期問題的進一步相關資訊。 因此，對於所擲出的每一個 `JMSException`，應用程式應該檢查鏈結的異常狀況。

鏈結的異常狀況本身可能有另一個鏈結的異常狀況，因此鏈結的異常狀況會形成鏈結，導致回到原始基礎問題。 鏈結異常狀況是利用 `java.lang.Throwable` 類別的鏈結異常狀況機制來實作，您的應用程式可以呼叫 `Throwable.getCause()` 方法來取得鏈結異常狀況。 若為 `JMSException`，`getLinkedException()` 方法會委派給 `Throwable.getCause()` 方法。

例如，如果應用程式在連接至佇列管理程式時指定不正確的埠號，則異常狀況會形成下列鏈結:

```
com.ibm.msg.client.jms.DetailIllegalStateException
|
+--->
    com.ibm.mq.MQException
    |
    +--->
        com.ibm.mq.jmqi.JmqiException
        |
        +--->
            com.ibm.mq.jmqi.JmqiException
            |
            +--->
                java.net.ConnectionException
```

通常會從程式碼中的不同層擲出鏈中的每一個異常狀況。 例如，前一個鏈中的異常狀況由下列層擲出:

- 第一個異常狀況 (`JMSException` 子類別的實例) 由 IBM MQ classes for JMS 中的一般層擲出。
- IBM MQ 傳訊提供者會擲出下一個異常狀況，即 `com.ibm.mq.MQException` 實例。
- 「Java 訊息佇列介面 (JMQI)」會擲出接下來的兩個異常狀況，這兩個異常狀況都是 `com.ibm.mq.jmqi.JmqiException` 的實例。 JMQI 是「IBM MQ classes for JMS」用來與佇列管理程式進行通訊的元件。
- Java 類別庫會擲出最終異常狀況 (`java.net.ConnectionException` 的實例)。

如需 IBM MQ classes for JMS 之分層架構的相關資訊，請參閱 IBM MQ for JMS 架構類別。

您可以撰寫應用程式來反覆運算此鏈結，以擷取所有適當的資訊，如下列範例所示:

JM 3.0 V 9.3.0 V 9.3.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import jakarta.jms.JMSException;
.
.
.
catch (JMSException je) {
  System.err.println("Caught JMSException");
  // Check for linked exceptions in JMSException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSException) {
      JMSException je1 = (JMSException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
```

```
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

JMS 2.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import javax.jms.JMSException;
.
.
.
catch (JMSException je) {
  System.err.println("Caught JMSException");
  // Check for linked exceptions in JMSException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSException) {
      JMSException je1 = (JMSException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

請注意，您的應用程式應該一律檢查鏈中每一個異常狀況的類型，因為異常狀況的類型可能有所不同，且不同類型的異常狀況會封裝不同的資訊。

取得問題的 IBM MQ 特定相關資訊

`com.ibm.mq.MQException` 和 `com.ibm.mq.jmqi.JmqiException` 的實例會封裝 IBM MQ 問題的特定相關資訊。

`MQException` 會封裝下列資訊:

- 完成碼，您的應用程式可以透過呼叫 `getCompCode()` 方法來取得。
- 原因碼，您的應用程式可以透過呼叫 `getReason()` 方法來取得。

如需如何使用這些方法的範例，請參閱 鏈結的異常狀況中的範例程式碼。

JmqiException 也會封裝完成碼和原因碼。 此外， JmqiException 還包含 AMQ *nnnn* 或 CSQ *nnnn* 訊息 (如果有與異常狀況相關聯的話) 中的資訊。 您的應用程式可以呼叫下列方法來取得此訊息的各種元件:

- getWmqMsgExplanation() 方法會傳回 AMQ *nnnn* 或 CSQ *nnnn* 訊息的說明。
- getWmqMsgSeverity() 方法會傳回 AMQ *nnnn* 或 CSQ *nnnn* 訊息的嚴重性。
- getWmqMsgSummary() 方法會傳回 AMQ *nnnn* 或 CSQ *nnnn* 訊息的摘要。
- getWmqMsgUserResponse() 方法會傳回與 AMQ *nnnn* 或 CSQ *nnnn* 訊息相關聯的使用者回應。

處理未檢查的異常狀況
定義在 JMS 2.0 中的介面上的方法會擲出未檢查的異常狀況。 這些異常狀況的基礎類別是 JMSRuntimeException。 因此，捕捉 JMSRuntimeExceptions 提供一般方式來處理這些類型的異常狀況。

每一個 JMSRuntimeException 都會封裝下列資訊:

- 提供者特定的異常狀況訊息，您的應用程式可以透過呼叫 JMSRuntimeException.getMessage() 方法來取得該訊息。
- 提供者特定的錯誤碼，您的應用程式可以透過呼叫 JMSRuntimeException.getErrorCode() 方法來取得該錯誤碼。
- 鏈結的異常狀況。 JMS 2.0 API 呼叫所擲出的異常狀況，通常是鏈結至此異常狀況的另一個異常狀況所報告的較低層次問題所導致。 您的應用程式可以呼叫 JMSRuntimeException.getCause() 方法來取得鏈結的異常狀況。

當您在 JMS 2.0 API 所提供的介面上呼叫方法時， IBM MQ classes for JMS 所擲出的大部分異常狀況是 JMSRuntimeException 的子類別實例。 這些子類別實作 com.ibm.msg.client.jms.JmsExceptionDetail 介面，其提供下列其他資訊:

- 異常狀況訊息的說明。 您的應用程式可以呼叫 JmsExceptionDetail.getExplanation() 方法來取得此訊息。
- 異常狀況的建議使用者回應。 您的應用程式可以呼叫 JmsExceptionDetail.getUserAction() 方法來取得此訊息。
- 在異常狀況訊息中插入訊息的索引鍵。 您的應用程式可以透過呼叫 JmsExceptionDetail.getKeys() 方法來取得所有索引鍵的反覆運算子。
- 訊息會插入異常狀況訊息中。 例如，訊息插入可能是導致異常狀況的佇列名稱，而且您的應用程式存取該名稱可能很有用。 您的應用程式可以呼叫 JmsExceptionDetail.getValue() 方法，以取得對應於指定金鑰的訊息插入。

如果沒有可用的詳細資料， JmsExceptionDetail 介面中的所有方法都會傳回空值。

例如，如果應用程式嘗試為不存在的 IBM MQ 佇列建立 JMSProducer，則會擲出具有下列資訊的異常狀況:

```
Message : JMSWMQ2008: Failed to open MQ queue 'Q_test'.
Class : class com.ibm.msg.client.jms.DetailedInvalidDestinationException
Error Code : JMSWMQ2008
Explanation : JMS attempted to perform an MQOPEN, but IBM MQ reported an
              error.
User Action : Use the linked exception to determine the cause of this error. Check
              that the specified queue and queue manager are defined correctly.
```

擲出的異常狀況 com.ibm.msg.client.jms.DetailedInvalidDestinationException 是下列類別的子類別，並實作 com.ibm.msg.client.jms.JmsExceptionDetail 介面。

- JM 3.0 V 9.3.0 V 9.3.0 jakarta.jms.InvalidDestinationException
- JMS 2.0 javax.jms.InvalidDestinationException

鏈結的異常狀況

一般而言，異常狀況是由其他異常狀況所造成。 因此，對於所擲出的每一個 JMSRuntimeException ， 您的應用程式應該檢查鏈結的異常狀況。

JMSRuntimeException 的原因可能是另一個異常狀況。 這些異常狀況會形成鏈結，導致回到原始基礎問題。 異常狀況的原因是利用 java.lang.Throwable 類別的鏈結異常狀況機制來實作，您的應用程式可以呼叫 Throwable.getCause() 方法來取得鏈結異常狀況。

例如，如果應用程式在連接至佇列管理程式時指定不正確的埠號，則異常狀況會形成下列鏈結:

```
com.ibm.msg.client.jms.DetailIllegalStateException
|
+--->
    com.ibm.mq.MQException
    |
    +--->
        com.ibm.mq.jmqi.JmqiException
        |
        +--->
            com.ibm.mq.jmqi.JmqiException
            |
            +--->
                java.net.ConnectionException
```

通常會從程式碼中的不同層擲出鏈中的每一個異常狀況。 例如，前一個鏈中的異常狀況由下列層擲出:

- 第一個異常狀況 (JMSRuntimeException 子類別的實例) 由 IBM MQ classes for JMS 中的一般層擲出。
- IBM MQ 傳訊提供者會擲出下一個異常狀況，即 com.ibm.mq.MQException 實例。
- 「Java 訊息佇列介面 (JMQI)」會擲出接下來的兩個異常狀況，這兩個異常狀況都是 com.ibm.mq.jmqi.JmqiException 的實例。 JMQI 是「IBM MQ classes for JMS 」用來與佇列管理程式進行通訊的元件。
- Java 類別庫會擲出最終異常狀況 (java.net.ConnectionException 的實例)。

如需 IBM MQ classes for JMS 之分層架構的相關資訊，請參閱 IBM MQ for JMS 架構類別。

您可以撰寫應用程式來反覆運算此鏈結，以擷取所有適當的資訊，如下列範例所示:

JM 3.0 V 9.3.0 V 9.3.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import jakarta.jms.JMSRuntimeException;
.
.
.
catch (JMSRuntimeException je) {
  System.err.println("Caught JMSRuntimeException");
  // Check for linked exceptions in JMSRuntimeException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSRuntimeException) {
      JMSRuntimeException je1 = (JMSRuntimeException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
```

```
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

JMS 2.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import javax.jms.JMSRuntimeException;
.
.
.
catch (JMSRuntimeException je) {
  System.err.println("Caught JMSRuntimeException");
  // Check for linked exceptions in JMSRuntimeException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSRuntimeException) {
      JMSRuntimeException je1 = (JMSRuntimeException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

請注意，您的應用程式應該一律檢查鏈中每一個異常狀況的類型，因為異常狀況的類型可能有所不同，且不同類型的異常狀況會封裝不同的資訊。

取得問題的 IBM MQ 特定相關資訊

`com.ibm.mq.MQException` 和 `com.ibm.mq.jmqi.JmqiException` 的實例會封裝 IBM MQ 問題的特定相關資訊。

`MQException` 會封裝下列資訊:

- 完成碼，您的應用程式可以透過呼叫 `getCompCode()` 方法來取得。
- 原因碼，您的應用程式可以透過呼叫 `getReason()` 方法來取得。

如需如何使用這些方法的範例，請參閱 鏈結異常狀況中的範例程式碼。

`JmqiException` 也會封裝完成碼和原因碼。 此外， `JmqiException` 還包含 AMQ *nnnn* 或 CSQ *nnnn* 訊息 (如果有與異常狀況相關聯的話) 中的資訊。 您的應用程式可以呼叫下列方法來取得此訊息的各種元件:

- `getWmqMsgExplanation()` 方法會傳回 AMQ *nnnn* 或 CSQ *nnnn* 訊息的說明。
- `getWmqMsgSeverity()` 方法會傳回 AMQ *nnnn* 或 CSQ *nnnn* 訊息的嚴重性。

- `getWmqMsgSummary()` 方法會傳回 AMQ *nnnn* 或 CSQ *nnnn* 訊息的摘要。
- `getWmqMsgUserResponse()` 方法會傳回與 AMQ *nnnn* 或 CSQ *nnnn* 訊息相關聯的使用者回應。

ExceptionListeners
JMS `Connection` 和 `JMSContext` 物件具有與佇列管理程式相關聯的連線。 您的應用程式可以向 JMS `Connection` 或 `JMSContext` 登錄 `ExceptionListener`。 如果發生問題，導致與 `Connection` 或 `JMSContext` 相關聯的連線無法使用，則 IBM MQ classes for JMS 會透過呼叫其 `onException()` 方法，將異常狀況遞送至 `ExceptionListener`。 然後您的應用程式就有機會重新建立連線。

如果在嘗試非同步遞送訊息時發生問題， IBM MQ classes for JMS 也可以將異常狀況遞送至異常狀況接聽器。

異常狀況接聽器

從 IBM MQ 8.0.0 Fix Pack 2 開始，為了維護配置 JMS `MessageListener` 和 JMS `ExceptionListener` 之現行 JMS 應用程式的行為，並確保 IBM MQ classes for JMS 與 JMS 規格一致， `ConnectionFactory` 內容 ASYNCEXCEPTION 的預設值會變更為 ASYNC_EXCEPTIONS_CONNECTIONBROKEN。 因此，只會將對應於中斷連線錯誤碼的異常狀況遞送至應用程式的 `ExceptionListener`。

APAR IT14820(從 IBM MQ 9.0.0 Fix Pack 1 併入) 會更新 IBM MQ classes for JMS， 以便:

- 不論應用程式是使用同步或非同步訊息消費者，都會對任何連線中斷異常狀況呼叫應用程式所登錄的 `ExceptionListener`。
- 當應用程式使用非同步訊息消費者，且應用程式所使用的 JMS ConnectionFactory 將 ASYNC_EXCEPTION 內容設為值 ASYNC_EXCEPTIONS_ALL 時，在訊息遞送期間產生的非連線中斷異常狀況 (例如 MQRC_GET_INHIBITED) 會遞送至應用程式的 `ExceptionListener`。

註: 對於連線中斷異常狀況，即使兩個 TCP/IP 連線 (一個由 JMS 連線使用，另一個由 JMS 階段作業使用) 中斷，也只會呼叫 `ExceptionListener` 一次。

對於任何其他類型的問題，現行 JMS API 呼叫會擲出異常狀況。 擲出的異常狀況類型取決於應用程式正在使用的 JMS API 版本:

- 如果應用程式使用 JMS 1.1 規格所提供的介面，則例外是 `JMSException`。 如需如何處理這些異常狀況的相關資訊，請參閱 第 202 頁的『處理已檢查的異常狀況』。
- 如果應用程式使用 JMS 2.0 介面，則異常狀況為 `JMSRuntimeException`。 如需如何處理這些異常狀況的相關資訊，請參閱 第 205 頁的『處理未檢查的異常狀況』。

如果應用程式未向 `Connection` 或 `JMSContext` 登錄異常狀況接聽器，則會將任何將遞送至異常狀況接聽器的異常狀況寫入 IBM MQ classes for JMS 日誌。

從 *IBM MQ classes for JMS* 應用程式存取 *IBM MQ* 特性

IBM MQ classes for JMS 提供機能來利用 IBM MQ 的許多特性。

小心: 這些特性在 JMS 規格之外，或在某些情況下違反 JMS 規格。 如果您使用它們，則您的應用程式不可能與其他 JMS 提供者相容。 那些不符合 JMS 規格的特性會標上「注意」注意事項。

從 *IBM MQ classes for JMS* 應用程式讀取及寫入訊息描述子

您可以透過在「目的地」及「訊息」上設定內容，來控制存取訊息描述子 (MQMD) 的能力。

部分 IBM MQ 應用程式需要在傳送至它們的訊息 MQMD 中設定特定值。 IBM MQ classes for JMS 提供訊息屬性，可讓 JMS 應用程式設定 MQMD 欄位，因此讓 JMS 應用程式「驅動」 IBM MQ 應用程式。

您必須將目的地物件內容 WMQ_MQMD_WRITE_ENABLED 設為 true， MQMD 內容的設定才會有任何效果。 然後，您可以使用訊息的內容設定方法 (例如 setStringProperty)，將值指派給 MQMD 欄位。 除了 StrucId 和版本之外，所有 MQMD 欄位都公開; BackoutCount 可以讀取，但無法寫入。

此範例會導致將訊息放入具有 MQMD.UserIdentifier 設為 "JoeBloggs"。

```
// Create a ConnectionFactory, connection, session, producer, message
// ...

// Create a destination
```

```
// ...

// Enable MQMD write
dest.setBooleanProperty(WMQConstants.WMQ_MQMD_WRITE_ENABLED, true);

// Optionally, set a message context if applicable for this MD field
dest.setIntProperty(WMQConstants.WMQ_MQMD_MESSAGE_CONTEXT,
  WMQConstants.WMQ_MDCTX_SET_IDENTITY_CONTEXT);

// On the message, set property to provide custom UserId
msg.setStringProperty("JMS_IBM_MQMD_UserIdentifier", "JoeBloggs");

// Send the message
// ...
```

在設定 JMS_IBM_MQMD_UserIdentifier 之前，必須先設定 WMQ_MQMD_MESSAGE_CONTEXT。 如需使用 WMQ_MQMD_MESSAGE_CONTEXT 的相關資訊，請參閱 第 210 頁的『JMS 訊息物件內容』。

同樣地，您可以在接收訊息之前將 WMQ_MQMD_READ_ENABLED 設為 true ， 然後使用訊息的 get 方法 (例如 getString 內容) 來擷取 MQMD 欄位的內容。 任何收到的內容都是唯讀的。

此範例會導致 *value* 欄位保留 MQMD.ApplIdentityData 欄位。

```
// Create a ConnectionFactory, connection, session, consumer
// ...

// Create a destination
// ...

// Enable MQMD read
dest.setBooleanProperty(WMQConstants.WMQ_MQMD_READ_ENABLED, true);

// Receive a message
// ...

// Get MQMD field value using a property
String value = rcvMsg.getStringProperty("JMS_IBM_MQMD_ApplIdentityData");
```

JMS 目的地物件內容

「目的地」物件的兩個內容會控制從 JMS 對 MQMD 的存取，第三個內容會控制訊息環境定義。

表 *36:* 內容名稱和說明

內容	**簡短形式**	**說明**
已啟用 WMQ_MQMD_WRITE_ENABLED	MDW	JMS 應用程式是否可以設定 MQMD 欄位的值
WMQ_MQMD_READ_ENABLED	MDR	JMS 應用程式是否可以擷取 MQMD 欄位的值
WMQ_MQMD_MESSAGE_CONTEXT	MDCTX	要由 JMS 應用程式設定的訊息環境定義層次。 應用程式必須以適當的環境定義權限執行，此內容才會生效

表 *37:* 內容名稱、值及 *set* 方法

內容	**管理工具中的有效值 (以粗體顯示預設值)**	**程式中的有效值**	**設定方法**
WMQ_MQMD_WRITE_ENABLED	• **否** 會忽略所有 JMS_IBM_MQMD* 內容，且其值不會複製到基礎 MQMD 結構。 • YES 會處理 JMS_IBM_MQMD* 內容。 其值會複製到基礎 MQMD 結構。	• **否** • True	setMQMDWrite 已啟用

表 37: 內容名稱、值及 *set* 方法 (繼續)

內容	管理工具中的有效值 (以粗體顯示預設值)	程式中的有效值	設定方法
WMQ_MQMD_READ_ENABLED	• **否** 傳送訊息時，不會更新已傳送訊息上的 JMS_IBM_MQMD* 內容，以反映 MQMD 中已更新的欄位值。 接收訊息時，接收的訊息上沒有任何可用的 JMS_IBM_MQMD* 內容，即使寄件者已設定部分或全部內容都一樣。 • YES 傳送訊息時，會更新已傳送訊息上的所有 JMS_IBM_MQMD* 內容，以反映 MQMD 中已更新的欄位值，包括寄件者未明確設定的那些欄位值。 接收訊息時，接收的訊息上有所有 JMS_IBM_MQMD* 內容，其中包括使寄件者未明確設定的那些內容。	• **否** • True	setMQMDRead 已啟用
WMQ_MQMD_MESSAGE_CONTEXT	• **Default** MQOPEN API 呼叫和 MQPMO 結構未指定明確訊息環境定義選項 • SET_IDENTITY_CONTEXT MQOPEN API 呼叫指定訊息環境定義選項 MQOO_SET_IDENTITY_CONTEXT，MQPMO 結構指定 MQPMO_SET_IDENTITY_CONTEXT • 環境 _ALL_CONTEXT MQOPEN API 呼叫指定訊息環境定義選項 MQOO_SET_ALL_CONTEXT，MQPMO 結構指定 MQPMO_SET_ALL_CONTEXT	• **WMQ_MDCTTX_DEFAULT** • WMQ_MDCTX_SET_IDENTITY_CONTEXT • WMQ_MDCTX_SET_ALL_CONTEXT	setMQMDMessage 環境定義

JMS 訊息物件內容
字首為 JMS_IBM_MQMD 的訊息物件內容可讓您設定或讀取對應的 MQMD 欄位。

傳送訊息

除了 StrucId 和 Version 之外的所有 MQMD 欄位都已呈現。 這些內容僅參照 MQMD 欄位; 如果內容同時出現在 MQMD 和 MQRFH2 標頭中，則不會設定或擷取 MQRFH2 中的版本。

可以設定任何這些內容，但 JMS_IBM_MQMD_BackoutCount 除外。 會忽略為 JMS_IBM_MQMD_BackoutCount 設定的任何值。

如果內容具有長度上限，且您提供的值太長，則會截斷該值。

對於某些內容，您還必須在目的地物件上設定 WMQ_MQMD_MESSAGE_CONTEXT 內容。 應用程式必須以適當的環境定義權限執行，才能使此內容生效。 如果您未將 WMQ_MQMD_MESSAGE_CONTEXT 設為適當的值，則會忽略內容值。 如果您將 WMQ_MQMD_MESSAGE_CONTEXT 設為適當的值，但您沒有足夠的佇列管理程式環境定義權限，則會發出 JMSException。 需要 WMQ_MQMD_MESSAGE_CONTEXT 特定值的內容如下。

下列內容需要將 WMQ_MQMD_MESSAGE_CONTEXT 設為 WMQ_MDCTX_SET_IDENTITY_CONTEXT 或 WMQ_MDCTTX_SET_ALL_CONTEXT:

- JMS_IBM_MQMD_UserIdentifier
- JMS_IBM_MQMD_AccountingToken
- JMS_IBM_MQMD_ApplIdentity 資料

下列內容需要將 WMQ_MQMD_MESSAGE_CONTEXT 設為 WMQ_MDCTTX_SET_ALL_CONTEXT:

- JMS_IBM_MQMD_PutAppl 類型
- JMS_IBM_MQMD_PutAppl 名稱
- JMS_IBM_MQMD_PutDate
- JMS_IBM_MQMD_PutTime
- JMS_IBM_MQMD_ApplOrigin 資料

接收訊息

如果 WMQ_MQMD_READ_ENABLED 內容設為 true，則不論生產端應用程式已設定的實際內容為何，所有這些內容都可以在收到的訊息上使用。 除非先根據 JMS 規格清除所有內容，否則應用程式無法修改所接收訊息的內容。 可以在不修改內容的情況下轉遞接收到的訊息。

小心: 如果您的應用程式從 WMQ_MQMD_READ_ENABLED 內容設為 true 的目的地接收訊息，並將它轉遞至 WMQ_MQMD_WRITE_ENABLED 設為 true 的目的地，這會導致將所接收訊息的所有 MQMD 欄位值複製到轉遞的訊息。

內容表

此表格列出代表 MQMD 欄位的 Message 物件的內容。 如需欄位及其容許值的完整說明，請參閱鏈結。

表 *38:* 內容名稱、說明及類型

內容	說明	Java 類型	鏈結至完整說明
JMS_IBM_MQMD_Report	報告訊息的選項	整數	報告
JMS_IBM_MQMD_MsgType	訊息類型	整數	MsgType
JMS_IBM_MQMD_Expiry	訊息期限	整數	期限
JMS_IBM_MQMD_Feedback	回饋碼或原因碼	整數	意見
JMS_IBM_MQMD_Encoding	訊息資料的數字編碼	整數	編碼
JMS_IBM_MQMD_CodedCharSetId	訊息資料的字集 ID	整數	CodedCharSetId
JMS_IBM_MQMD_Format	訊息資料的格式名稱	字串	格式
JMS_IBM_MQMD_Priority [1]	訊息優先順序	整數	優先順序
JMS_IBM_MQMD_Persistence	訊息持續性	整數	持續性
JMS_IBM_MQMD_MsgId [2]	訊息 ID	物件 (byte []) [4]	MsgId
JMS_IBM_MQMD_CorrelId [3]	相關性 ID	物件 (byte []) [4]	CorrelId
JMS_IBM_MQMD_BackoutCount	取消計數器	整數	BackoutCount
JMS_IBM_MQMD_ReplyToQ	回覆佇列的名稱	字串	ReplyToQ
JMS_IBM_MQMD_ReplyTo 佇列管理程式	回覆佇列管理程式的名稱	字串	回覆目的地佇列管理程式
JMS_IBM_MQMD_UserIdentifier	使用者 ID	字串	UserIdentifier

表 38: 內容名稱、說明及類型 (繼續)			
內容	**說明**	**Java 類型**	**鏈結至完整說明**
JMS_IBM_MQMD_AccountingToken	帳戶記號	物件 (byte []) [4]	AccountingToken
JMS_IBM_MQMD_ApplIdentity 資料	與身分相關的應用程式資料	字串	ApplIdentityData
JMS_IBM_MQMD_PutAppl 類型	放置訊息的應用程式類型	整數	PutApplType
JMS_IBM_MQMD_PutAppl 名稱	放置訊息的應用程式名稱	字串	PutApplName
JMS_IBM_MQMD_PutDate	放置訊息的日期	字串	PutDate
JMS_IBM_MQMD_PutTime	放置訊息的時間	字串	PutTime
JMS_IBM_MQMD_ApplOrigin 資料	與出處相關的應用程式資料	字串	ApplOriginData
JMS_IBM_MQMD_GroupId	群組 ID	物件 (byte []) [4]	GroupId
JMS_IBM_MQMD_MsgSeq 號碼	群組內的邏輯訊息的序號	整數	MsgSeqNumber
JMS_IBM_MQMD_Offset	從邏輯訊息開始的實體訊息中的資料偏移	整數	偏移
JMS_IBM_MQMD_MsgFlags	訊息旗標	整數	MsgFlags
JMS_IBM_MQMD_OriginalLength	原始訊息的長度	整數	OriginalLength

1. **小心:** 如果您指派值給不在 0-9 範圍內的 JMS_IBM_MQMD_Priority，則會違反 JMS 規格。
2. **小心:** JMS 規格指出訊息 ID 必須由 JMS 提供者設定，且必須是唯一或空值。 如果您指派值給 JMS_IBM_MQMD_MsgId，則會將此值複製到 JMSMessageID。 因此，它不是由 JMS 提供者所設定，且可能不是唯一的: 這違反 JMS 規格。
3. **小心:** 如果您指派值給以字串 'ID:' 開頭的 JMS_IBM_MQMD_CorrelId，則會違反 JMS 規格。
4. **小心:** 在訊息上使用位元組陣列內容違反 JMS 規格。

使用 *IBM MQ classes for JMS* 從應用程式存取 *IBM MQ* 訊息資料
您可以使用 IBM MQ classes for JMS 來存取應用程式內的完整 IBM MQ 訊息資料。 若要存取所有資料，訊息必須是 JMSBytesMessage。 JMSBytesMessage 的內文包括任何 MQRFH2 標頭、任何其他 IBM MQ 標頭，以及下列訊息資料。

將目的地的 WMQ_MESSAGE_BODY 內容設為 WMQ_MESSAGE_BODY_MQ，以接收 JMSBytesMessage 中的所有訊息內文資料。

如果 WMQ_MESSAGE_BODY 設為 WMQ_MESSAGE_BODY_JMS 或 WMQ_MESSAGE_BODY_UNSPECIFIED，則會傳回不含 JMS MQRFH2 標頭的訊息內文，且 JMSBytesMessage 的內容會反映在 RFH2 中設定的內容。

部分應用程式無法使用本主題中說明的功能。 如果應用程式連接至 IBM MQ V6 佇列管理程式，或已將 PROVIDERVERSION 設為 6，則無法使用這些功能。

傳送訊息
傳送訊息時，目的地內容 WMQ_MESSAGE_BODY 優先於 WMQ_TARGET_CLIENT。

如果 WMQ_MESSAGE_BODY 設為 WMQ_MESSAGE_BODY_JMS，則 IBM MQ classes for JMS 會根據 JMSMessage 內容及標頭欄位的設定自動產生 MQRFH2 標頭。

如果 WMQ_MESSAGE_BODY 設為 WMQ_MESSAGE_BODY_MQ，則不會將其他標頭新增至訊息內文

如果 WMQ_MESSAGE_BODY 設為 WMQ_MESSAGE_BODY_UNSPECIFIED，則 IBM MQ classes for JMS 會傳送 MQRFH2 標頭，除非 WMQ_TARGET_CLIENT 設為 WMQ_TARGET_DEST_MQ。 在接收時，將 WMQ_TARGET_CLIENT 設為 WMQ_TARGET_DEST_MQ 會導致從訊息內文中移除任何 MQRFH2 。

註: JMSBytesMessage 和 JMSTextMessage 不需要 MQRFH2，而 JMSStreamMessage、JMSMapMessage 和 JMSObjectMessage 則需要。

WMQ_MESSAGE_BODY_UNSPECIFIED 是 WMQ_MESSAGE_BODY 的預設值， WMQ_TARGET_DEST_JMS 是 WMQ_TARGET_CLIENT 的預設值。

如果您傳送 JMSBytesMessage，則可以在建構 IBM MQ 訊息時置換 JMS 訊息內文的預設值。 使用下列內容:

- JMS_IBM_Format 或 JMS_IBM_MQMD_Format: 此內容指定 IBM MQ 標頭或應用程式有效負載的格式，會啟動 JMS 訊息內文 (如果之前沒有 WebSphere MQ 標頭的話)。
- JMS_IBM_Character_Set 或 JMS_IBM_MQMD_CodedCharSetId: 此內容會指定 IBM MQ 標頭或應用程式有效負載的 CCSID，如果沒有前置 WebSphere MQ 標頭，則會啟動 JMS 訊息內文。
- JMS_IBM_Encoding 或 JMS_IBM_MQMD_Encoding: 此內容會指定在沒有前置 WebSphere MQ 標頭的情況下，啟動 JMS 訊息內文之 IBM MQ 標頭或應用程式有效負載的編碼。

如果同時指定這兩種類型的內容，則只要目的地內容 WMQ_MQMD_WRITE_ENABLED 設為 true，JMS_IBM_MQMD_* 內容就會置換對應的 JMS_IBM_* 內容。

使用 JMS_IBM_MQMD_* 和 JMS_IBM_* 來設定訊息內容之間的效果差異非常顯著:

1. JMS_IBM_MQMD_* 內容特定於 IBM MQ JMS 提供者。
2. JMS_IBM_MQMD_* 內容僅在 MQMD 中設定。 只有在訊息沒有 MQRFH2 JMS 標頭時，才會在 MQMD 中設定 JMS_IBM_* 內容。 否則，它們會設定在 JMS RFH2 標頭中。
3. JMS_IBM_MQMD_* 內容不會影響寫入 JMSMessage 的文字和數字的編碼。

   接收端應用程式可能會假設 MQMD.Encoding 及 MQMD.CodedCharSetId 的值對應於訊息內文中數字及文字的編碼及字集。 如果使用 JMS_IBM_MQMD_* 內容，則由傳送端應用程式負責這樣做。 訊息內文中數字及文字的編碼及字集是由 JMS_IBM_* 內容所設定。

   第 213 頁的圖 39 中編碼錯誤的 Snippet 會傳送以字集 1208 編碼的訊息，且 MQMD.CodedCharSetId 設為 37。

   a. 傳送錯誤編碼的訊息

   ```
   TextMessage tmo = session.createTextMessage();
   ((MQDestination) destination).setMessageBodyStyle
                   (WMQConstants.WMQ_MESSAGE_BODY_MQ);
   ((MQDestination)destination).setMQMDWriteEnabled(true);
   tmo.setIntProperty(WMQConstants.JMS_IBM_MQMD_CODEDCHARSETID, 37);
   tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 1208);
   tmo.setText("String one");
   producer.send(tmo);
   ```

   b. 根據 MQMD.CodedCharSetId 值所設定的 JMS_IBM_CHARACTER_SET 值，來接收訊息:

   ```
   TextMessage tmi = (TextMessage) cons.receive();
   System.out.println("Message is \"" + tmi.getText() + "\"");
   ```

   c. 產生的輸出:

   ```
   Message is "éÈÊ'>...??>?"
   ```

   圖 *39:* 編碼不一致的 *MQMD* 及訊息資料

第 214 頁的圖 40 中的程式碼 Snippet 會導致將訊息放入佇列或主題，其內文包含應用程式有效負載，但未新增自動產生的 MQRFH2 標頭。

1. 設定 WMQ_MESSAGE_BODY_MQ:

```
((MQDestination) destination).setMessageBodyStyle
                (WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

2. 設定 WMQ_TARGET_DEST_MQ:

```
((MQDestination) destination).setMessageBodyStyle
                (WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);
((MQDestination) destination).
                setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
```

圖 *40:* 傳送含有 *MQ* 訊息內文的訊息。

接收訊息

如果 WMQ_MESSAGE_BODY 設為 WMQ_MESSAGE_BODY_JMS，則入埠 JMS 訊息類型及內文由接收的 WebSphere MQ 訊息內容決定。 訊息類型和內文由 MQRFH2 標頭中的欄位決定，如果沒有 MQRFH2，則由 MQMD 中的欄位決定。

如果 WMQ_MESSAGE_BODY 設為 WMQ_MESSAGE_BODY_MQ，則入埠 JMS 訊息類型為 JMSBytesMessage。 JMS 訊息內文是基礎 MQGET API 呼叫所傳回的訊息資料。 訊息內文的長度是 MQGET 呼叫所傳回的長度。 訊息內文中資料的字集及編碼是由 MQMD 的 CodedCharSetId 及 Encoding 欄位所決定。 訊息內文中資料的格式由 MQMD 的 **格式** 欄位決定

如果 WMQ_MESSAGE_BODY 設為 WMQ_MESSAGE_BODY_UNSPECIFIED，則預設值 IBM MQ classes for JMS 會將它設為 WMQ_MESSAGE_BODY_JMS。

當您收到 JMSBytesMessage 時，可以參照下列內容來將它解碼:

- JMS_IBM_Format 或 JMS_IBM_MQMD_Format: 此內容指定 IBM MQ 標頭或應用程式有效負載的格式，會啟動 JMS 訊息內文 (如果之前沒有 WebSphere MQ 標頭的話)。
- JMS_IBM_Character_Set 或 JMS_IBM_MQMD_CodedCharSetId: 此內容會指定 IBM MQ 標頭或應用程式有效負載的 CCSID，如果沒有前置 WebSphere MQ 標頭，則會啟動 JMS 訊息內文。
- JMS_IBM_Encoding 或 JMS_IBM_MQMD_Encoding: 此內容會指定在沒有前置 WebSphere MQ 標頭的情況下，啟動 JMS 訊息內文之 IBM MQ 標頭或應用程式有效負載的編碼。

下列程式碼 Snippet 會產生收到的訊息，即 JMSBytesMessage。 不論所接收訊息的內容及所接收 MQMD 的格式欄位為何，該訊息都是 JMSBytesMessage。

```
  ((MQDestination)destination).setMessageBodyStyle
                              (WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

目的地內容 *WMQ_MESSAGE_BODY*
WMQ_MESSAGE_Body 可判定 JMS 應用程式是否將 IBM MQ 訊息的 MQRFH2 作為訊息有效負載的一部分 (亦即，作為 JMS 訊息內文的一部分) 處理。

表 *39:* 內容名稱和說明

內容	**簡短形式**	**說明**
WMQ_MESSAGE_Body	MBODY	JMS 應用程式是否將 IBM MQ 訊息的 MQRFH2 作為訊息有效負載的一部分 (亦即，作為 JMS 訊息內文的一部分) 處理。

表 *40:* 內容名稱、值及 *set* 方法

內容	管理工具中的有效值 (以粗體顯示預設值)	程式中的有效值	設定方法
WMQ_MESSAGE_BODY	• **未指定** 傳送時，視 WMQ_TARGET_CLIENT 的值而定， IBM MQ classes for JMS 不一定會產生並包含 MQRFH2 標頭。 接收時，作為值 JMS。 • JMS 傳送時， IBM MQ classes for JMS 會自動產生 MQRFH2 標頭，並將它包含在 IBM MQ 訊息中。 接收時， IBM MQ classes for JMS 會根據 MQRFH2 (如果有的話) 中的值來設定 JMS 訊息內容; 它不會將 MQRFH2 呈現為 JMS 訊息內文的一部分。 • MQ 傳送時， IBM MQ classes for JMS 不會產生 MQRFH2。 接收時， IBM MQ classes for JMS 會將 MQRFH2 呈現為 JMS 訊息內文的一部分。	• **WMQ_MESSAGE_ 未指定的 BODY_UNSPECIFIED** • WMQ_MESSAGE_BODY_JMS • WMQ_MESSAGE_BODY_MQ	setMessageBodyStyle

JMS 持續訊息

IBM MQ classes for JMS 應用程式可以使用 **NonPersistentMessageClass** 佇列屬性，為 JMS 持續訊息提供更好的效能，但代價是部分可靠性。

IBM MQ 佇列具有稱為 **NonPersistentMessageClass** 的屬性。 此屬性的值決定當佇列管理程式重新啟動時是否捨棄佇列上的非持續訊息。

您可以使用 IBM MQ Script (MQSC) 指令 DEFINE QLOCAL ，搭配下列其中一個參數來設定本端佇列的屬性:

NPMCLASS (NORMAL)
當佇列管理程式重新啟動時，會捨棄佇列上的非持續訊息。 這是預設值。

NPMCLASS (HIGH)
當佇列管理程式在靜止或立即關閉之後重新啟動時，不會捨棄佇列上的非持續訊息。 不過，在強制關機或故障之後，可能會捨棄非持續訊息。

本主題說明 IBM MQ classes for JMS 應用程式如何使用此佇列屬性，為 JMS 持續訊息提供更好的效能。

「佇列」或「主題」物件的 PERSISTENCE 內容可以具有值 HIGH。 您可以使用 IBM MQ JMS 管理工具來設定此值，或者應用程式可以呼叫 Destination.setPersistence() 方法，以傳遞值 WMQConstants.WMQ_PER_NPHIGH 作為參數。

如果應用程式將 JMS 持續訊息或 JMS 非持續訊息傳送至 PERSISTENCE 內容值為 HIGH 且基礎 IBM MQ 佇列設為 NPMCLASS (HIGH) 的目的地，則訊息會以 IBM MQ 非持續訊息形式放置在佇列上。 如果目的地的 PERSISTENCE 內容沒有值 HIGH ，或基礎佇列設為 NPMCLASS (NORMAL) ，則會將 JMS 持續訊息作為 IBM MQ 持續訊息放置在佇列上，並將 JMS 非持續訊息作為 IBM MQ 非持續訊息放置在佇列上。

如果 JMS 持續訊息以 IBM MQ 非持續訊息形式放置在佇列上，且您想要確保在佇列管理程式靜止或立即關閉之後不會捨棄該訊息，則可能透過其遞送訊息的所有佇列都必須設為 NPMCLASS (HIGH)。 在發佈/訂閱網

域中，這些佇列包括訂閱者佇列。 作為施行此配置的輔助工具，如果應用程式嘗試為 PERSISTENCE 內容值為 HIGH 且基礎 IBM MQ 佇列設為 NPMCLASS (NORMAL) 的目的地建立訊息消費者，則 IBM MQ classes for JMS 會擲出 InvalidDestination 異常狀況。

將目的地的 PERSISTENCE 內容設為 HIGH，並不會影響從該目的地接收訊息的方式。 以 JMS 持續訊息形式傳送的訊息會以 JMS 持續訊息形式接收，以 JMS 非持續訊息形式傳送的訊息會以 JMS 非持續訊息形式接收。

When an application sends the first message to a destination where the PERSISTENCE property has the value HIGH, or when an application creates the first message consumer for a destination where the PERSISTENCE property has the value HIGH, IBM MQ classes for JMS issues an MQINQ call to determine whether NPMCLASS(HIGH) is set on the underlying IBM MQ queue. 因此，應用程式必須具有查詢佇列的權限。 此外， IBM MQ classes for JMS 會保留 MQINQ 呼叫的結果，直到刪除目的地為止，且不會發出更多 MQINQ 呼叫。 因此，當應用程式仍在使用目的地時，如果您變更基礎佇列上的 NPMCLASS 設定， IBM MQ classes for JMS 不會注意到新的設定。

透過容許將 JMS 持續訊息作為 IBM MQ 非持續訊息放置在 IBM MQ 佇列上，您會以犧牲部分可靠性來獲得效能。 如果您需要 JMS 持續訊息的可靠性上限，請勿將訊息傳送至 PERSISTENCE 內容值為 HIGH 的目的地。

JMS 層可以使用 SYSTEM.JMS.TEMPQ.MODEL，而不是 SYSTEM.DEFAULT.MODEL.QUEUE。 SYSTEM.JMS.TEMPQ.MODEL 會建立接受持續訊息的永久動態佇列，因為 SYSTEM.DEFAULT.MODEL.QUEUE 無法接受持續訊息。 若要使用暫時佇列來接受持續訊息，您必須使用 SYSTEM.JMS.TEMPQ.MODEL，或將模型佇列變更為您選擇的替代佇列。

搭配使用 *TLS* 與 *IBM MQ classes for JMS*

IBM MQ classes for JMS 應用程式可以使用「傳輸層安全 (TLS)」加密。 若要這樣做，他們需要 JSSE 提供者。

使用 TRANSPORT (CLIENT) 的 IBM MQ classes for JMS 連線支援 TLS 加密。 TLS 提供通訊加密、鑑別及訊息完整性。 它通常用來保護網際網路或企業內部網路中任何兩個對等節點之間的通訊安全。

IBM MQ classes for JMS 使用 Java Secure Socket Extension (JSSE) 來處理 TLS 加密，因此需要 JSSE 提供者。 JSE v1.4 JVM 已內建 JSSE 提供者。 如何管理及儲存憑證的詳細資料可能因提供者而異。 如需相關資訊，請參閱 JSSE 提供者的說明文件。

本節假設您的 JSSE 提供者已正確安裝和配置，且已安裝適當的憑證並可供 JSSE 提供者使用。 現在您可以使用 JMSAdmin 來設定一些管理內容。

如果 IBM MQ classes for JMS 應用程式使用用戶端通道定義表 (CCDT) 來連接至佇列管理程式，請參閱 第 239 頁的『搭配使用用戶端通道定義表與 IBM MQ classes for JMS』。

SSLCIPHERSUITE 物件內容

設定 SSLCIPHERSUITE 以在 ConnectionFactory 物件上啟用 TLS 加密。

如果要在 ConnectionFactory 物件上啟用 TLS 加密，請使用 JMSAdmin 將 SSLCIPHERSUITE 內容設為 JSSE 提供者所支援的 CipherSuite 。 這必須符合目標通道上設定的 CipherSpec 。 不過， CipherSuites 不同於 CipherSpecs ，因此具有不同的名稱。 第 219 頁的『IBM MQ classes for JMS 中的 TLS CipherSpecs 及 CipherSuites』 包含一個表格，將 IBM MQ 支援的 CipherSpecs 對映至 JSSE 已知的對等 CipherSuites 。 如需 IBM MQ 的 CipherSpecs 及 CipherSuites 的相關資訊，請參閱 保護 IBM MQ 安全。

例如，若要設定 ConnectionFactory 物件 (可用來透過啟用 TLS 且 CipherSpec 為 TLS_RSA_WITH_AES_128_CBC_SHA 的 MQI 通道建立連線) ，請向 JMSAdmin 發出下列指令:

```
ALTER CF(my.cf) SSLCIPHERSUITE(SSL_RSA_WITH_AES_128_CBC_SHA)
```

這也可以從應用程式使用 MQConnectionFactory 物件上的 setSSLCipherSuite () 方法來設定。

為了方便起見，如果在 SSLCIPHERSUITE 內容上指定 CipherSpec ， JMSAdmin 會嘗試將 CipherSpec 對映至適當的 CipherSuite ，並發出警告。 如果應用程式指定內容，則不會嘗試對映。

或者，使用「用戶端通道定義表 (CCDT)」。 如需相關資訊，請參閱 第 239 頁的『搭配使用用戶端通道定義表與 IBM MQ classes for JMS』。

SSLFIPSREQUIRED 物件內容
如果您需要連線才能使用 IBM Java JSSE FIPS 提供者 (IBMJSSEFIPS) 支援的 CipherSuite，請將 Connection Factory 的 SSLFIPSREQUIRED 內容設為 YES。

註: 在 AIX, Linux, and Windows 上，IBM MQ 透過 IBM Crypto for C (ICC) 加密模組提供 FIPS 140-2 相符性。此模組的憑證已移至「歷程」狀態。客戶應該檢視 IBM Crypto for C (ICC) 憑證，並注意 NIST 提供的任何建議。目前正在進行取代 FIPS 140-3 模組，您可以在 處理程序清單中的 NIST CMVP 模組中搜尋它，以檢視其狀態。

此內容的預設值為 NO，表示連線可以使用 IBM MQ 支援的任何 CipherSuite。

如果應用程式使用多個連線，則當應用程式建立第一個連線時所使用的 SSLFIPSREQUIDED 值會決定當應用程式建立任何後續連線時所使用的值。這表示會忽略用來建立後續連線之 Connection Factory 的 SSLFIPSREQUIRED 內容值。如果您想要使用不同的 SSLFIPSREQUIRED 值，則必須重新啟動應用程式。

應用程式可以呼叫 ConnectionFactory 物件的 setSSLFipsRequired () 方法來設定這個內容。如果未設定 CipherSuite，則會忽略此內容。

相關工作
指定在執行時期於 MQI 用戶端上僅使用 FIPS 認證的 CipherSpecs
相關參考
AIX, Linux, and Windows 的聯邦資訊存取安全標準 (FIPS)

SSLPEERNAME 物件內容
使用 SSLPEERNAME 來指定識別名稱型樣，以確保 JMS 應用程式連接至正確的佇列管理程式。

JMS 應用程式可以指定識別名稱 (DN) 型樣，以確保它連接至正確的佇列管理程式。只有在佇列管理程式呈現符合型樣的 DN 時，連線才會成功。如需此型樣格式的詳細資料，請參閱相關主題。

DN 是使用 ConnectionFactory 物件的 SSLPEERNAME 內容來設定。例如，下列 JMSAdmin 指令會設定 ConnectionFactory 物件，以預期佇列管理程式會使用以 `QMGR.` 字元開頭的「通用名稱」以及至少兩個「組織單位」名稱 (第一個名稱必須是 IBM，第二個名稱必須是 WEBSPHERE) 來識別自己:

```
ALTER CF(my.cf) SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

檢查不區分大小寫，且可以使用分號來取代逗點。您也可以在 MQConnectionFactory 物件上使用 setSSLPeerName () 方法，從應用程式設定 SSLPEERNAME。如果未設定此內容，則不會對佇列管理程式提供的「識別名稱」執行任何檢查。如果未設定 CipherSuite，則會忽略此內容。

SSLCERTSTORES 物件內容
使用 SSLCERTSTORES 來指定要用於憑證撤銷清冊 (CRL) 檢查的 LDAP 伺服器清單。

通常會使用憑證撤銷清冊 (CRL) 來識別不再信任的憑證。CRL 通常是在 LDAP 伺服器上管理。JMS 容許在 Java 2 v1.4 或更新版本下指定 LDAP 伺服器以進行 CRL 檢查。下列 JMSAdmin 範例會指示 JMS 使用在 LDAP 伺服器 crl1.ibm.com:

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com)
```

註: 若要順利搭配使用 CertStore 與 LDAP 伺服器上管理的 CRL，請確定 Java Software Development Kit (SDK) 與 CRL 相容。部分 SDK 需要 CRL 符合 RFC 2587，這會定義 LDAP v2 的綱目。大部分 LDAP v3 伺服器改用 RFC 2256。

如果您的 LDAP 伺服器不是在預設埠 389 上執行，您可以在主機名稱中附加冒號 (:) 及埠號來指定埠。如果佇列管理程式所提供的憑證存在於 crl1.ibm.com 上所管理的 CRL 中，則連線不會完成。為了避免單一失敗點，JMS 提供以空格字元定界的 LDAP 伺服器清單，以容許提供多部 LDAP 伺服器。以下是範例:

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com ldap://crl2.ibm.com)
```

當指定多個 LDAP 伺服器時，JMS 會依序嘗試每一部伺服器，直到找到可用來順利驗證佇列管理程式憑證的伺服器為止。每一部伺服器必須包含相同的資訊。

此格式的字串可由應用程式在 MQConnectionFactory.setSSLCertStores () 方法上提供。 或者，應用程式可以建立一或多個 java.security.cert.CertStore 物件，將這些物件放在適當的 Collection 物件中，並將這個 Collection 物件提供給 setSSLCertStores () 方法。 如此一來，應用程式就可以自訂 CRL 檢查。 如需建構及使用 CertStore 物件的詳細資料，請參閱 JSSE 文件。

設定連線時佇列管理程式所提供的憑證驗證如下:

1. sslCertStores 所識別「集合」中的第一個 CertStore 物件用來識別 CRL 伺服器。
2. 嘗試聯絡 CRL 伺服器。
3. 如果嘗試成功，則會搜尋伺服器以找出憑證的相符項。
   a. 如果發現要撤銷憑證，則搜尋程序會結束，且連線要求會失敗，原因碼為 MQRC_SSL_CERTIFICATE_REVOKED。
   b. 如果找不到憑證，則會結束搜尋程序，並容許繼續進行連線。
4. 如果嘗試聯絡伺服器失敗，則會使用下一個 CertStore 物件來識別 CRL 伺服器，且處理程序會從步驟 2 重複。

   如果這是「集合」中最後一個 CertStore ，或如果「集合」未包含任何 CertStore 物件，則搜尋程序會失敗，且連線要求會失敗，原因碼為 MQRC_SSL_CERT_STORE_ERROR。

Collection 物件會決定 CertStores 的使用順序。

如果您的應用程式使用 setSSLCertStores () 來設定 CertStore 物件的集合，則無法再將 MQConnectionFactory 連結至 JNDI 名稱空間。 嘗試這樣做會導致異常狀況。 如果未設定 sslCert 商店內容，則不會對佇列管理程式提供的憑證執行撤銷檢查。 如果未設定 CipherSuite ，則會忽略此內容。

SSLRESETCOUNT 物件內容
此內容代表在重新協議用於加密的秘密金鑰之前，連線所傳送及接收的位元組總數。

傳送的位元組數是加密之前的數目，而接收的位元組數是解密之後的數目。 位元組數也包括 IBM MQ classes for JMS 所傳送及接收的控制資訊。

比方說，如果要配置 ConnectionFactory 物件，以用來透過啟用 TLS 的 MQI 通道建立連線，並使用在傳送 4 MB 資料之後重新協議的秘密金鑰，請向 JMSAdmin 發出下列指令:

```
ALTER CF(my.cf) SSLRESETCOUNT(4194304)
```

應用程式可以透過呼叫 ConnectionFactory 物件的 setSSLResetCount () 方法來設定此內容。

如果此內容的值為零 (預設值) ，則永不重新協議秘密金鑰。 如果未設定 CipherSuite ，則會忽略此內容。

SSLSocketFactory 物件內容
若要為應用程式自訂 TLS 連線的其他方面，請建立 SSLSocketFactory 並配置 JMS 以使用它。

您可能想要為應用程式自訂 TLS 連線的其他方面。 例如，您可能想要起始設定加密硬體，或變更使用中的金鑰儲存庫及信任儲存庫。 若要這樣做，應用程式必須先建立相應地自訂的 javax.net.ssl.SSLSocketFactory 物件。 如需如何執行此動作的相關資訊，請參閱 JSSE 說明文件，因為可自訂特性因提供者而異。 取得適當的 SSLSocketFactory 物件之後，請使用 MQConnectionFactory.setSSLSocketFactory () 方法來配置 JMS ，以使用自訂的 SSLSocketFactory 物件。

如果您的應用程式使用 setSSLSocketFactory () 方法來設定自訂 SSLSocketFactory 物件，則 MQConnectionFactory 物件無法再連結至 JNDI 名稱空間。 嘗試這樣做會導致異常狀況。 如果未設定此內容，則會使用預設 SSLSocketFactory 物件。 如需預設 SSLSocketFactory 物件行為的詳細資料，請參閱 JSSE 文件。 如果未設定 CipherSuite ，則會忽略此內容。

重要: 當從本身不安全的 JNDI 名稱空間擷取 ConnectionFactory 物件時，請勿假設使用 SSL 內容會確保安全。 具體而言， JNDI 的標準 LDAP 實作並不安全。 攻擊者可以模仿 LDAP 伺服器，誤導 JMS 應用程式連接錯誤伺服器而不注意。 在適當的安全安排下， JNDI 的其他實作 (例如 fscontext 實作) 是安全的。

變更 *JSSE* 金鑰儲存庫或信任儲存庫
如果您對金鑰儲存庫或信任儲存庫進行變更，則必須採取某些動作，變更才會生效。

如果您變更 JSSE 金鑰儲存庫或信任儲存庫的內容，或變更金鑰儲存庫或信任儲存庫檔案的位置，則當時執行的 IBM MQ classes for JMS 應用程式不會自動挑選變更。 若要讓變更生效，必須執行下列動作:

- 應用程式必須關閉其所有連線，並毀損連線儲存區中任何未用的連線。
- 如果 JSSE 提供者從金鑰儲存庫和信任儲存庫快取資訊，則必須重新整理此資訊。

執行這些動作之後，應用程式就可以重建其連線。

視您設計應用程式的方式以及 JSSE 提供者所提供的功能而定，可能可以執行這些動作，而不需要停止並重新啟動應用程式。 不過，停止並重新啟動應用程式可能是最簡單的解決方案。

IBM MQ classes for JMS 中的 *TLS CipherSpecs* 及 *CipherSuites*

IBM MQ classes for JMS 應用程式建立佇列管理程式連線的能力，取決於在 MQI 通道伺服器端指定的 CipherSpec 及在用戶端指定的 CipherSuite 。

下表列出 IBM MQ 支援的 CipherSpecs 及其對等的 CipherSuites。

Deprecated 您應該檢閱 已淘汰 CipherSpecs 主題，以查看下列表格中列出的任何 CipherSpecs 是否已被 IBM MQ 淘汰，如果已淘汰，則會在其中更新 CipherSpec 。

重要: 列出的 CipherSuites 是 IBM MQ 隨附的 IBM Java 執行時期環境 (JRE) 所支援的那些 CipherSuite。 列出的 CipherSuites 包括 Oracle Java JRE 所支援的那些 CipherSuite。 如需將應用程式配置成使用 Oracle Java JRE 的相關資訊，請參閱 將應用程式配置成使用 IBM Java 或 Oracle Java CipherSuite 對映。

此表格也指出用於通訊的通訊協定，以及 CipherSuite 是否符合 FIPS 140-2 標準。

註: 在 AIX, Linux, and Windows 上， IBM MQ 透過 IBM Crypto for C (ICC) 加密模組提供 FIPS 140-2 相符性。 此模組的憑證已移至「歷程」狀態。 客戶應該檢視 IBM Crypto for C (ICC) 憑證 ，並注意 NIST 提供的任何建議。 目前正在進行取代 FIPS 140-3 模組，您可以在 處理程序清單中的 NIST CMVP 模組中搜尋它，以檢視其狀態。

如果應用程式尚未配置為施行 FIPS 140-2 相符性，但如果已針對應用程式配置 FIPS 140-2 相符性 (請參閱下列配置注意事項) ，則只能配置標示為 FIPS 140-2 相容的 CipherSuites ; 嘗試使用其他 CipherSuites 會導致錯誤。

註: 每一個 JRE 都可以有多個加密安全提供者，每一個提供者都可以提供相同 CipherSuite 的實作。 不過，並非所有安全提供者都經過 FIPS 140-2 認證。 如果應用程式未施行 FIPS 140-2 相符性，則可能使用未經認證的 CipherSuite 實作。 未經認證的實作可能不符合 FIPS 140-2 標準運作，即使 CipherSuite 理論上符合標準所需的最低安全層次也一樣。 如需在 IBM MQ JMS 應用程式中配置 FIPS 140-2 強制執行的相關資訊，請參閱下列注意事項。

如需 CipherSpecs 及 CipherSuites 的 FIPS 140-2 及 Suite-B 相符性的相關資訊，請參閱 指定 CipherSpecs。 您也可能需要注意有關美國 聯邦資訊存取安全標準的資訊。

若要使用完整的 CipherSuites 集，並使用經認證的 FIPS 140-2 及/或 Suite-B 標準來操作，需要適合的 JRE。 IBM Java 7 Service Refresh 4 Fix Pack 2 或更高層次的 IBM JRE 提供對 第 220 頁的表 41 中列出的 TLS 1.2 CipherSuites 的適當支援。

若要能夠使用 TLS 1.3 密碼，執行應用程式的 JRE 必須支援 TLS 1.3。

註: 若要使用部分 CipherSuites，需要在 JRE 中配置「未限定」原則檔。 如需如何在 SDK 或 JRE 中設定原則檔的詳細資料，請參閱您所使用版本的 *Security Reference for IBM SDK* , *Java Technology Edition* 中的 *IBM SDK* 原則檔 主題。

表 *41: IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs*

CipherSpec 第 234 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
ECDHE_ECDSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	yes
ECDHE_ECDSA_AES_128_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS 1.2	yes

表 41: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (*繼續*)*

CipherSpec 第 234 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
ECDHE_ECDSA_AES_128_GCM_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS 1.2	yes

表 *41: IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (*繼續*)*				
CipherSpec 第 234 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
ECDHE_ECDSA_AES_256_CBC_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS 1.2	yes

表 41: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (*繼續*)*

CipherSpec 第 234 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
ECDHE_ECDSA_AES_256_GCM_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS 1.2	yes
ECDHE_ECDSA_NULL_SHA256	SSL_ECDHE_ECDSA_WITH_NULL_SHA	TLS_ECDHE_ECDSA_WITH_NULL_SHA	TLS 1.2	否

表 41: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (*繼續*)*

CipherSpec 第 234 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS 1.2	否
ECDHE_RSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	yes

表 41: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (*繼續*)*

CipherSpec 第 234 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
ECDHE_RSA_AES_128_CBC_SHA256	SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	yes
ECDHE_RSA_AES_128_GCM_SHA256	SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	yes

表 *41: IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (*繼續*)*				
CipherSpec 第 234 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
ECDHE_RSA_AES_256_CBC_SHA384	SSL_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS 1.2	yes
ECDHE_RSA_AES_256_GCM_SHA384	SSL_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	yes

表 41: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (*繼續*)*

CipherSpec 第 234 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
ECDHE_RSA_NULL_SHA256	SSL_ECDHE_RSA_WITH_NULL_SHA	TLS_ECDHE_RSA_WITH_NULL_SHA	TLS 1.2	否
ECDHE_RSA_RC4_128_SHA256	SSL_ECDHE_RSA_WITH_RC4_128_SHA	TLS_ECDHE_RSA_WITH_RC4_128_SHA	TLS 1.2	否

表 41: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (繼續)*

CipherSpec 第 234 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
TLS_RSA_WITH_3DES_EDE_CBC_SHA 第 235 頁的『2』	SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	否 第 235 頁的『4』
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	否 第 235 頁的『4』

表 41: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (*繼續*)*

CipherSpec 第 234 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	否 第 235 頁的『4』
TLS_RSA_WITH_AES_128_GCM_SHA256	SSL_RSA_WITH_AES_128_GCM_SHA256	TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	否 第 235 頁的『4』

表 41: IBM MQ 及其對等的 CipherSuites 支援 CipherSpecs (繼續)

CipherSpec 第 234 頁的『1』	等效 CipherSuite (IBM JRE)	等效 CipherSuite (Oracle JRE)	通訊協定	FIPS 140-2 相容
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	否 第 235 頁的『4』
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	否 第 235 頁的『4』

表 41: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (*繼續*)*

CipherSpec 第 234 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
TLS_RSA_WITH_AES_256_GCM_SHA384	SSL_RSA_WITH_AES_256_GCM_SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	否 第 235 頁的『4』
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	TLS 1.0	否

表 41: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (*繼續*)*

CipherSpec 第 234 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	TLS_RSA_WITH_NULL_SHA256	TLS 1.2	否
TLS_RSA_WITH_RC4_128_SHA256	SSL_RSA_WITH_RC4_128_SHA	SSL_RSA_WITH_RC4_128_SHA	TLS 1.2	否
ANY_TLS12	*TLS12	*TLS12	TLS 1.2	yes
TLS_AES_128_GCM_SHA256 第 235 頁的『3』	TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS 1.3 版	否

表 41: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (*繼續*)*

CipherSpec 第 234 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
TLS_AES_256_GCM_SHA384 第 235 頁的『3』	TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS 1.3 版	否
TLS_CHACHA20_POLY1305_SHA256 第 235 頁的『3』	TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS 1.3 版	否
TLS_AES_128_CCM_SHA256 第 235 頁的『3』	TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS 1.3 版	否

表 41: IBM MQ 及其對等的 CipherSuites 支援 CipherSpecs (繼續)				
CipherSpec 第 234 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
TLS_AES_128_CCM_8_SHA256 第 235 頁的『3』	TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS 1.3 版	否
任意第 235 頁的『3』	*ANY	*ANY	多重	否
ANY_TLS13 第 235 頁的『3』	*TLS13	*TLS13	TLS V13	否
ANY_TLS12_OR_HIGHER 第 235 頁的『3』	*TLS12ORHIGHER	*TLS12ORHIGHER	TLS 1.2 及以上版本	否
ANY_TLS13_OR_HIGHER 第 235 頁的『3』	*TLS13ORHIGHER	*TLS13ORHIGHER	TLS 1.3 及以上版本	否

附註:

1. 這是在 IBM MQ 中的通道上配置的值，包括在 CCDT 中 (二進位或 JSON)。

2. Deprecated CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA 已淘汰。 不過，在連線因錯誤 AMQ9288 而終止之前，它仍可用來傳送最多 32 GB 的資料。 若要避免此錯誤，您需要避免使用三重 DES 演算法，或在使用此 CipherSpec 時啟用秘密金鑰重設。
3. 若要能夠使用 TLS v1.3 密碼，執行應用程式的 Java runtime environment (JRE) 必須支援 TLS v1.3。
4. V 9.3.0.17 V 9.3.5.1 從 IBM MQ 9.3.5 CSU 1 和 IBM MQ 9.3.0 CSU 17 開始，當以 FIPS 模式運作時， IBM Java 8 JRE 不再支援 RSA 金鑰交換。

在 IBM MQ classes for JMS 應用程式中配置密碼組合及 FIPS 相符性

- 使用 IBM MQ classes for JMS 的應用程式可以使用下列兩種方法之一來設定連線的 CipherSuite :
  - 呼叫 ConnectionFactory 物件的 setSSLCipherSuite 方法。
  - 使用 IBM MQ JMS 管理工具來設定 ConnectionFactory 物件的 SSLCIPHERSUITE 內容。
- 使用 IBM MQ classes for JMS 的應用程式可以使用下列兩種方法之一來施行 FIPS 140-2 相符性:
  - 呼叫 ConnectionFactory 物件的 setSSLFips 必要方法。
  - 使用 IBM MQ JMS 管理工具來設定 ConnectionFactory 物件的 SSLFIPSREQUIRED 內容。

將應用程式配置成使用 IBM Java 或 Oracle Java CipherSuite 對映

註: V 9.3.3 對於 IBM MQ 9.3.3 中的 Continuous Delivery， 會從產品中移除 Java 系統內容 `com.ibm.mq.cfg.useIBMCipherMappings`(控制使用哪些對映)。 從 IBM MQ 9.3.3 開始， 「密碼」可以定義為 CipherSpec 或 CipherSuite 名稱，並由 IBM MQ 正確處理。 如果您的 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 應用程式建立與佇列管理程式的安全 TLS 連線，則需要下列三個 Jackson JAR 檔:

- `jackson-annotations.jar`
- `jackson-core.jar`
- `jackson-databind.jar`

重要: `com.ibm.mq.cfg.useIBMCipherMappings` 的下列相關資訊僅適用於 Long Term Support 及 Continuous Delivery 之前 IBM MQ 9.3.3 。

您可以配置應用程式是使用預設 IBM Java CipherSuite 至 IBM MQ CipherSpec 對映，還是使用 Oracle CipherSuite 至 IBM MQ CipherSpec 對映。 因此，不論您的應用程式使用 IBM JRE 或 Oracle JRE， 都可以使用 TLS CipherSuites 。 Java 系統內容 `com.ibm.mq.cfg.useIBMCipherMappings` 控制使用哪些對映。 內容可以是下列其中一個值:

true
使用 IBM Java CipherSuite 至 IBM MQ CipherSpec 對映。

此值為預設值。

false
使用 Oracle CipherSuite 至 IBM MQ CipherSpec 對映。

如需使用 IBM MQ Java 和 TLS 密碼的相關資訊，請參閱 MQdev 部落格文章 MQ Java、TLS 密碼、非 IBM JRE 及 APAR IT06775、 IV66840、 IT09423、 IT10837。

交互作業能力限制

視使用中的通訊協定而定，某些 CipherSuites 可能與多個 IBM MQ CipherSpec 相容。 不過， 僅支援使用表 1 中所指定 TLS 版本的 CipherSuite/CipherSpec 組合。 嘗試使用不受支援的 CipherSuites 及 CipherSpecs 組合將因適當的異常狀況而失敗。 使用任何這些 CipherSuite/CipherSpec 組合的安裝應該移至受支援的組合。

下表顯示套用此限制的 CipherSuites 。

表 42: CipherSuites 及其受支援及不受支援的 CipherSpecs		
CipherSuite	**支援的 TLS CipherSpec**	**不受支援的 SSL CipherSpec**
SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA 第 236 頁的『1』	TRIPLE_DES_SHA_US
SSL_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA	DES_SHA_EXPORT
SSL_RSA_WITH_RC4_128_SHA	TLS_RSA_WITH_RC4_128_SHA256	RC4_SHA_US

註:

1. Deprecated 此 CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA 已淘汰。 不過，在連線因錯誤 AMQ9288 而終止之前，它仍可用來傳送最多 32 GB 的資料。 若要避免此錯誤，您需要避免使用三重 DES 演算法，或在使用此 CipherSpec 時啟用秘密金鑰重設。

在 *Java for IBM MQ classes for JMS* 中寫入通道結束程式

您可以透過定義實作指定介面的 Java 類別來建立通道結束程式。

如需安全結束程式的簡介，請從 通道安全結束程式 主題開始。

com.ibm.mq.exits 套件中定義了三個介面:

- WMQSendExit，適用於傳送結束程式
- WMQReceiveExit，適用於接收結束程式
- WMQSecurityExit，適用於安全結束程式

下列範例程式碼定義實作所有三個介面的類別:

```
public class MyMQExits implements
WMQSendExit, WMQReceiveExit, WMQSecurityExit {
    // Default constructor
  public MyMQExits(){
  }
    // This method implements the send exit interface
  public ByteBuffer channelSendExit(
                                    MQCXP channelExitParms,
                                    MQCD channelDefinition,
                                    ByteBuffer agentBuffer)
  {
    // Complete the body of the send exit here
  }
    // This method implements the receive exit interface
  public ByteBuffer channelReceiveExit(
                                    MQCXP channelExitParms,
                                    MQCD channelDefinition,
                                    ByteBuffer agentBuffer)
  {
    // Complete the body of the receive exit here
  }
    // This method implements the security exit interface
  public ByteBuffer channelSecurityExit(
                                    MQCXP channelExitParms,
                                    MQCD channelDefinition,
                                    ByteBuffer agentBuffer)
  {
    // Complete the body of the security exit here
  }
}
```

每一個結束程式都會以參數形式接收 MQCXP 物件和 MQCD 物件。 這些物件代表程序化介面中定義的 MQCXP 和 MQCD 結構。

呼叫傳送結束程式時， agentBuffer 參數會包含即將傳送至伺服器佇列管理程式的資料。 不需要長度參數，因為表示式 agentBuffer.limit () 提供資料的長度。 傳送結束程式會傳回要傳送至伺服器佇列管理程式的資料作為其值。 不過，如果傳送結束程式不是一連串傳送結束程式中的最後一個傳送結束程式，則傳回的資料會

改為依序傳遞至下一個傳送結束程式。 傳送結束程式可以傳回 agentBuffer 參數中所接收資料的已修改版本，也可以傳回未變更的資料。 因此，最簡單的結束程式主體為:

```
{ return agentBuffer; }
```

當呼叫接收結束程式時， agentBuffer 參數會包含已從伺服器佇列管理程式收到的資料。 接收結束程式會傳回要由 IBM MQ classes for JMS 傳遞至應用程式的資料作為其值。 不過，如果接收結束程式不是一連串接收結束程式中的最後一個接收結束程式，則傳回的資料會改為依序傳遞至下一個接收結束程式。

呼叫安全結束程式時， agentBuffer 參數會包含安全流程中從連線伺服器端安全結束程式收到的資料。 安全結束程式會傳回要在安全流程中傳送至伺服器安全結束程式的資料作為其值。

通道結束程式是以具有支持陣列的緩衝區來呼叫。 為了取得最佳效能，結束程式應該傳回具有支持陣列的緩衝區。

呼叫通道結束程式時，最多可以將 32 個字元的使用者資料傳遞至通道結束程式。 結束程式會呼叫 MQCXP 物件的 getExitData () 方法來存取使用者資料。 雖然結束程式可以透過呼叫 setExitData () 方法來變更使用者資料，但每次呼叫結束程式時都會重新整理使用者資料。 因此，對使用者資料所做的任何變更都會遺失。 不過，結束程式可以使用 MQCXP 物件的結束程式使用者區域，將資料從一個呼叫傳遞至下一個呼叫。 結束程式會呼叫 getExitUserArea() 方法來參照存取結束程式使用者區域。

每個結束程式類別都必須有建構子。 建構子可以是預設建構子 (如上述範例所示) ，或具有字串參數的建構子。 會呼叫建構子，以針對類別中定義的每一個結束程式建立結束程式類別的實例。 因此，在前一個範例中，會為傳送結束程式建立 MyMQExits 類別的實例，為接收結束程式建立另一個實例，並為安全結束程式建立第三個實例。 當呼叫含有字串參數的建構子時，參數會包含傳遞至要建立實例之通道結束程式的相同使用者資料。 如果結束類別同時具有預設建構子和單一參數建構子，則會優先使用單一參數建構子。

請勿從通道結束程式內關閉連線。

當資料傳送至連線的伺服器端時，會在呼叫任何通道結束程式 之後 執行 TLS 加密。 同樣地，從連線的伺服器端接收資料時，會在呼叫任何通道結束程式 之前 執行 TLS 解密。

在早於 IBM WebSphere MQ 7.0 的 IBM MQ classes for JMS 版本中，通道結束程式是使用介面 MQSendExit、 MQReceiveExit 及 MQSecurityExit 來實作。 您仍然可以使用這些介面，但為了改良功能及效能，建議使用新的介面。

配置 *IBM MQ classes for JMS* 以使用通道結束程式

IBM MQ classes for JMS 應用程式可以在應用程式連接至佇列管理程式時所啟動的 MQI 通道上使用通道安全、傳送及接收結束程式。 應用程式可以使用以 Java、C 或 C++ 撰寫的結束程式。 應用程式也可以使用一連串連續執行的傳送或接收結束程式。

下列內容用來指定 JMS 連線所使用的傳送結束程式或一連串傳送結束程式:

- MQConnectionFactory 物件的 **SENDEXIT** 內容。
- IBM MQ 資源配接器用於入埠通訊之啟動規格的 **sendexit** 內容，
- IBM MQ 資源配接器用於輸出通訊之 ConnectionFactory 物件上的 **sendexit** 內容。

內容的值是包含一個以上以逗點區隔的項目的字串。 每一個項目都會以下列其中一種方式來識別傳送結束程式:

- 針對以 Java 撰寫的傳送結束程式，實作 `WMQSendExit` 介面的類別名稱。
- 以 C 或 C++ 撰寫之傳送結束程式的字串，格式為 *libraryName* (*entryPointName*)。

以類似方式， 下列內容指定連線所使用的接收結束程式或接收結束程式順序:

- MQConnectionFactory 物件的 **RECEXIT** 內容。
- IBM MQ 資源配接器用於入埠通訊之啟動規格的 **receiveexit** 內容，
- IBM MQ 資源配接器用於輸出通訊之 ConnectionFactory 物件上的 **receiveexit** 內容。

下列內容指定連線所使用的安全結束程式:

- MQConnectionFactory 物件的 **SECEXIT** 內容。
- IBM MQ 資源配接器用於入埠通訊之啟動規格的 **securityexit** 內容，

- IBM MQ 資源配接器用於輸出通訊之 ConnectionFactory 物件上的 **securityexit** 內容。

對於 MQConnectionFactories，您可以使用 IBM MQ JMS 管理工具或 IBM MQ Explorer 來設定 **SENDEXIT**、**RECEXIT** 和 **SECEXIT** 內容。 或者，應用程式可以透過呼叫 `setSendExit()`、 `setReceiveExit()`及 `setSecurityExit()` 方法來設定內容。

通道結束程式由其自己的類別載入器載入。 為了尋找通道結束程式，類別載入器會以指定的順序搜尋下列位置。

1. 在 IBM MQ 用戶端配置檔的 Channels 段落中，由內容 **com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath** 或 **JavaExitsClassPath** 屬性指定的類別路徑。
2. Deprecated Java 系統內容 **com.ibm.mq.exitClasspath** 指定的類別路徑。 請注意，此內容現在已淘汰。
3. IBM MQ 會結束目錄，如 第 238 頁的表 43 所示。 類別載入器會先在目錄中搜尋未包裝在 Java 保存檔 (JAR) 中的類別檔。 如果找不到通道結束程式，類別載入器會在目錄中搜尋 JAR 檔。

表 *43: IBM MQ* 結束程式目錄

平台	目錄
Linux AIX AIX and Linux	`/var/mqm/exits` (32 位元通道結束程式) `/var/mqm/exits64` (64 位元通道結束程式)
Windows Windows	*install_data_dir*\exits 其中 *install_data_dir* 是您在安裝期間為 IBM MQ 資料檔案選擇的目錄。 預設目錄為 C:\ProgramData\IBM\MQ。

註: 如果通道結束程式存在於多個位置，則 IBM MQ classes for JMS 會載入它找到的第一個實例。

類別載入器的母項是用來載入 IBM MQ classes for JMS 的類別載入器。 因此，如果在上述任何位置中都找不到通道結束程式，則母類別載入器可以載入通道結束程式。 不過，當您在 JEE 應用程式伺服器之類的環境中使用 IBM MQ classes for JMS 時，您可能無法影響母類別載入器的選擇，因此應該在應用程式伺服器上設定 Java 系統內容 **com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath** 來配置類別載入器。

如果您的應用程式是在啟用 Java security manager 的情況下執行，則執行應用程式的 Java 執行時期環境所使用的原則配置檔必須具有載入通道結束程式類別的許可權。 如需如何執行此動作的相關資訊，請參閱 在 Java 安全管理程式下執行適用於 JMS 應用程式的 IBM MQ 類別。

仍支援 IBM WebSphere MQ 7.0 之前的版本所提供的 `MQSendExit`、 `MQReceiveExit` 及 `MQSecurityExit` 介面。 如果您使用實作這些介面的通道結束程式，則 `com.ibm.mq.jar` 必須存在於類別路徑中。

如需如何在 C 中寫入通道結束程式的相關資訊，請參閱 第 805 頁的『傳訊通道的通道結束程式』。 您必須將以 C 或 C++ 撰寫的通道結束程式儲存在 第 238 頁的表 43 所顯示的目錄中。

如果您的應用程式使用用戶端通道定義表 (CCDT) 來連接佇列管理程式，請參閱 第 239 頁的『搭配使用用戶端通道定義表與 IBM MQ classes for JMS』。

指定使用 *IBM MQ classes for JMS* 時要傳遞至通道結束程式的使用者資料

呼叫通道結束程式時，最多可以將 32 個字元的使用者資料傳遞至通道結束程式。

MQConnectionFactory 物件的 SENDEXITINIT 內容指定在呼叫每一個傳送結束程式時傳遞給它的使用者資料。 內容的值是一個字串，包含以逗點區隔的一或多個使用者資料項目。 字串內每一個使用者資料項目的位置會決定傳送結束程式的哪一個傳送結束程式，在一連串傳送結束程式中，會將使用者資料傳遞至哪個傳送結束程式。 例如，字串中使用者資料的第一個項目會以一連串傳送結束程式傳遞至第一個傳送結束程式。

您可以使用 IBM MQ JMS 管理工具或 IBM MQ Explorer 來設定 SENDEXITINIT 內容。 或者，應用程式可以透過呼叫 setSendExitInit() 方法來設定內容。

同樣地， ConnectionFactory 物件的 RECEXITINIT 內容會指定傳給每一個接收結束程式的使用者資料，而 SECEXITINIT 內容會指定傳給安全結束程式的使用者資料。 您可以使用 IBM MQ JMS 管理工具或 IBM MQ

Explorer 來設定這些內容。 或者，應用程式可以透過呼叫 setReceiveExitInit() 及 setSecurityExitInit() 方法來設定內容。

指定傳遞至通道結束程式的使用者資料時，請注意下列規則:

- 如果字串中使用者資料的項目數超過序列中結束程式的數目，則會忽略使用者資料的多餘項目。
- 如果字串中使用者資料的項目數小於序列中結束程式的數目，則使用者資料的每一個未指定項目都會設為空字串。 字串內連續兩個逗點或字串開頭的逗點也表示使用者資料的未指定項目。

如果應用程式使用用戶端通道定義表 (CCDT) 來連接至佇列管理程式，當呼叫用戶端連線通道定義中指定的任何使用者資料時，會將它們傳遞至通道結束程式。 如需使用用戶端通道定義表的相關資訊，請參閱 第 239 頁的『搭配使用用戶端通道定義表與 IBM MQ classes for JMS』。

搭配使用用戶端通道定義表與 *IBM MQ classes for JMS*

IBM MQ classes for JMS 應用程式可以使用儲存在用戶端通道定義表 (CCDT) 中的用戶端連線通道定義。 您將 ConnectionFactory 物件配置成使用 CCDT。 使用它有一些限制。

除了透過設定 ConnectionFactory 物件的某些內容來建立用戶端連線通道定義之外， IBM MQ classes for JMS 應用程式也可以使用儲存在用戶端通道定義表中的用戶端連線通道定義。 這些定義是由 IBM MQ Script (MQSC) 指令或 IBM MQ 可程式化指令格式 (PCF) 指令所建立。 當應用程式建立 Connection 物件時， IBM MQ classes for JMS 會在用戶端通道定義表中搜尋適當的用戶端連線通道定義，並使用通道定義來啟動 MQI 通道。 如需用戶端通道定義表及如何建構一個通道定義表的相關資訊，請參閱 用戶端通道定義表。

如果要使用用戶端通道定義表， ConnectionFactory 物件的 CCDTURL 內容必須設為 URL 物件。 IBM MQ classes for JMS 不會從 IBM MQ MQI client 配置檔讀取 CCDT 的相關資訊，但會從該處使用其他一些值 (請參閱 第 83 頁的『IBM MQ classes for JMS/Jakarta Messaging 配置檔』，以取得適用的值)。 URL 物件封裝統一資源定址器 (URL)，可識別包含用戶端通道定義表之檔案的名稱及位置，並指定如何存取檔案。 您可以使用 IBM MQ JMS 管理工具來設定 CCDTURL 內容，或者應用程式可以透過建立 URL 物件並呼叫 ConnectionFactory 物件的 setCCDTURL() 方法來設定內容。

例如，如果檔案 ccdt1.tab 包含用戶端通道定義表，且儲存在應用程式執行所在的相同系統上，則應用程式可以使用下列方式來設定 CCDTURL 內容:

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
factory.setCCDTURL(chanTab1);
```

另舉一例，假設檔案 ccdt2.tab 包含用戶端通道定義表，並儲存在與執行應用程式的系統不同的系統上。 如果可以使用 FTP 通訊協定來存取檔案，應用程式可以使用下列方式來設定 CCDTURL 內容:

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
factory.setCCDTURL(chanTab2);
```

除了設定 ConnectionFactory 物件的 CCDTURL 內容之外，相同物件的 QMANAGER 內容還必須設為下列其中一個值:

- 佇列管理程式的名稱
- 星號 (*) 後接佇列管理程式群組的名稱

這些值與使用「訊息佇列介面 (MQI)」的用戶端應用程式發出的 MQCONN 呼叫中，可用於 **QMgrName** 參數的值相同。 如需這些值意義的相關資訊，請參閱 MQCONN。 您可以使用 IBM MQ JMS 管理工具或 IBM MQ Explorer 來設定 QMANAGER 內容。 另外，應用程式也可以呼叫 ConnectionFactory 物件的 setQueueManager () 方法來設定內容。

如果應用程式接著從 ConnectionFactory 物件建立 Connection 物件，則 IBM MQ classes for JMS 會存取 CCDTURL 內容所識別的用戶端通道定義表，使用 QMANAGER 內容來搜尋表格中是否有適當的用戶端連線通道定義，然後使用通道定義來啟動佇列管理程式的 MQI 通道。

請注意，當應用程式呼叫 createConnection() 方法時，無法同時設定 ConnectionFactory 物件的 CCDTURL 和 CHANNEL 內容。 如果同時設定這兩個內容，方法會擲出異常狀況。 如果 CCDTURL 或 CHANNEL 內容的值不是空值、空字串或包含所有空白字元的字串，則會將 CCDTURL 或 CHANNEL 內容視為已設定。

當 IBM MQ classes for JMS 在用戶端通道定義表中找到適當的用戶端連線通道定義時，它只會使用從表格中擷取的資訊來啟動 MQI 通道。 ConnectionFactory 物件的任何通道相關內容都會被忽略。

特別是，如果您使用 TLS，請注意下列要點:

- 只有在從用戶端通道定義表擷取的通道定義指定 IBM MQ classes for JMS 支援的 CipherSpec 名稱時，MQI 通道才會使用 TLS。
- 用戶端通道定義表也包含保留憑證撤銷清冊 (CRL) 之「輕量型目錄存取通訊協定 (LDAP)」伺服器位置的相關資訊。 IBM MQ classes for JMS 僅使用此資訊來存取保留 CRL 的 LDAP 伺服器。
- 用戶端通道定義表也可以包含 OCSP 回應端的位置。 「IBM MQ classes for JMS」無法在用戶端通道定義表檔案中使用 OCSP 資訊。 不過，您可以依照 Java 及 JMS 用戶端應用程式中的線上憑證狀態通訊協定 (OCSP)小節中的說明來配置 OCSP。

如需搭配使用 TLS 與用戶端通道定義表的相關資訊，請參閱 搭配使用延伸交易式用戶端與 TLS 通道。

如果您使用通道結束程式，也請注意下列要點:

- MQI 通道只會使用通道結束程式，以及從用戶端通道定義表擷取的通道定義所指定的相關聯使用者資料。
- 從用戶端通道定義表擷取的通道定義可以指定在 Java 中寫入的通道結束程式。 例如，這表示 DEFINE CHANNEL 指令上用來建立用戶端連線通道定義的 SCYEXIT 參數可以指定實作 WMQSecurityExit 介面的類別名稱。 同樣地， SENDEXIT 參數可以指定實作 WMQSendExit 介面的類別名稱，而 RCVEXIT 參數可以指定實作 WMQReceiveExit 介面的類別名稱。 如需如何在 Java 中寫入通道結束程式的相關資訊，請參閱 第 236 頁的『在 Java for IBM MQ classes for JMS 中寫入通道結束程式』。

  也支援使用以非 Java 語言撰寫的通道結束程式。 如需如何針對以另一種語言撰寫的通道結束程式，在 DEFINE CHANNEL 指令上指定 SCYEXIT、SENDEXIT 及 RCVEXIT 參數的相關資訊，請參閱 DEFINE CHANNEL。

自動 *JMS* 用戶端重新連線

配置 JMS 用戶端在網路、佇列管理程式或伺服器故障之後自動重新連接。

通常，如果獨立式 IBM MQ classes for JMS 應用程式使用用戶端傳輸連接至佇列管理程式，且佇列管理程式因某些原因而無法使用 (例如，由於網路中斷、佇列管理程式失敗或佇列管理程式正在停止)，則下次應用程式嘗試與佇列管理程式進行通訊時， IBM MQ classes for JMS 會擲出 JMSException。 應用程式必須捕捉 JMSException，並嘗試重新連接佇列管理程式。 您可以透過啟用自動用戶端重新連線來簡化應用程式的設計。 當佇列管理程式變成無法使用時，「IBM MQ classes for JMS」會代表應用程式自動嘗試重新連接至佇列管理程式。 這表示應用程式不需要包含邏輯即可重新連接。

在 Java Platform Enterprise Edition 應用程式伺服器內，不支援使用這項自動用戶端重新連線實作。 如需替代實作，請參閱 第 245 頁的『在 Java EE 環境中使用自動用戶端重新連線』。

使用自動 *JMS* 用戶端重新連線

如果獨立式 IBM MQ classes for JMS 應用程式使用已設定 CONNECTIONNAMELIST 或 CCDTURL 內容的 Connection Factory，則應用程式有資格使用自動用戶端重新連線。

自動用戶端重新連線可用來重新連接至佇列管理程式，包括那些屬於高可用性 (HA) 配置的佇列管理程式。 HA 配置包括 IBM MQ 應用裝置上的多重實例佇列管理程式、RDQM 佇列管理程式或 HA 佇列管理程式。

IBM MQ classes for JMS 提供的自動用戶端重新連線功能的行為取決於下列內容:

JMS Connection Factory 內容 TRANSPORT (簡稱 TRAN)

TRANSPORT 指定使用 Connection Factory 的應用程式如何連接佇列管理程式。 此內容必須設為值 CLIENT，才能使用自動用戶端重新連線。 如果應用程式連接的佇列管理程式使用的 Connection Factory 已將 TRANSPORT 內容設為 BIND、DIRECT 或 DIRECTTP，則無法使用自動用戶端重新連線。

JMS Connection Factory 內容 QMANAGER (簡稱 QMGR)

QMANAGER 內容會指定 Connection Factory 所連接的佇列管理程式名稱。

JMS Connection Factory 內容 CONNECTIONNAMELIST (簡稱 CRHOSTS)

CONNECTIONNAMELIST 內容是以逗點區隔的清單，其中每一個項目包含主機名稱及埠的相關資訊，當您使用 CLIENT 傳輸時，會使用這些資訊來連接至 QMANAGER 內容指定的佇列管理程式。 清單具有下列格式: 主機名稱 (埠)、主機名稱 (埠)。

JMS Connection Factory 內容 CCDTURL (簡稱 CCDT)

CCDTURL 內容會指向用戶端通道定義表，供 IBM MQ classes for JMS 使用 CCDT 連接至佇列管理程式時使用。

JMS Connection Factory 內容 CLIENTRECONNECTOPTIONS (簡稱 CROPT)

CLIENTRECONNECTOPTIONS 控制在佇列管理程式變成可用時， IBM MQ classes for JMS 是否會嘗試代表應用程式自動連接至佇列管理程式。

用戶端配置檔的「通道」段落中的 DefRecon 屬性

DefRecon 屬性提供一個管理選項，可讓所有應用程式自動重新連接，或針對寫入以自動重新連接的應用程式停用自動重新連線。

只有在應用程式順利連接至佇列管理程式時，才能使用自動用戶端重新連線。

當應用程式連接至使用 CLIENT 傳輸的佇列管理程式時，如果應用程式所連接的佇列管理程式變成無法使用， IBM MQ classes for JMS 會使用 Connection Factory 內容 CLIENTRECONNECTOPTIONS 的值來決定是否要使用自動用戶端重新連線。 表 1 顯示 CLIENTRECONNECTOPTIONS 內容的可能值， 以及 IBM MQ classes for JMS 對下列每一個值的行為:

表 *44:* 可能的 *CLIENTRECCECTOPTIONS* 內容值。

CLIENTRECONNECTOPTIONS	**IBM MQ classes for JMS 的行為**
ANY	如果已設定 CONNECTIONNAMELIST ， 請使用 CONNECTIONNAMELIST 內容的值來開啟與主機名稱及埠組合的連線，並連接至任何佇列管理程式。 若要使用此自動用戶端重新連線選項，必須將 QMANAGER 內容設為預設值或 "*"。 如果已設定 CCDTURL ， 請開啟 CCDTURL 內容指定的用戶端通道定義表，在表格中挑選項目，然後使用該項目來啟動佇列管理程式的用戶端連線通道。 若要使用此自動用戶端重新連線選項，必須將 QMANAGER 內容設為下列任一: • 星號 (*) • 星號 (*) 後接佇列管理程式群組的名稱 • 空字串，或包含所有空白字元的字串
ASDEF	請使用 DefRecon 的值來判斷是否可以使用自動用戶端重新連線。
已停用	請勿執行任何自動用戶端重新連線，並將 JMSException 傳回給應用程式。
QMGR	指定用戶端必須重新連接至相同的佇列管理程式。 此選項必須用於高可用性解決方案，其中需要重新連線至相同佇列管理程式的另一個實例。 如果已設定 CONNECTIONNAMELIST ， 請使用 CONNECTIONNAMELIST 內容的值來開啟與主機名稱及埠組合的連線，並連接至 QMANAGER 內容指定的佇列管理程式。 如果已設定 CCDTURL ， 請開啟 CCDTURL 內容所指定的用戶端通道定義表，在表格中尋找符合 QMANAGER 內容所指定佇列管理程式名稱的項目，然後使用那些項目來啟動與該佇列管理程式的用戶端連線通道。

如果設定 CONNECTIONNAMELIST，當您執行自動用戶端重新連線時， IBM MQ classes for JMS 會使用 Connection Factory 內容 CONNNECTIONNAMELIST 中的資訊來決定要重新連接的系統。

IBM MQ classes for JMS 一開始會嘗試使用 CONNECTIONNAMELIST 中第一個項目中指定的主機名稱及埠來重新連接。 如果建立連線，則「 IBM MQ classes for JMS 」會嘗試連接至具有 QMANAGER 內容中所指定名稱的佇列管理程式。 如果可以建立與佇列管理程式的連線，則 IBM MQ classes for JMS 會重新開啟應用程式在自動用戶端重新連線之前已開啟的所有 IBM MQ 物件，並繼續像之前一樣執行。

如果無法使用 CONNECTIONNAMELIST 中的第一個項目來建立與所需佇列管理程式的連線，則 IBM MQ classes for JMS 會嘗試 CONNECTIONNAMELIST 中的第二個項目，依此類推。

當 IBM MQ classes for JMS 已嘗試 CONNECTIONNAMELIST 中的所有項目時，它們會等待一段時間，然後再嘗試重新連接。 若要執行新的重新連線嘗試， IBM MQ classes for JMS 會從 CONNECTIONNAMELIST 中的第一個項目開始。 然後依次嘗試 CONNECTIONNAMELIST 中的每一個項目，直到發生重新連線或到達 CONNECTIONNAMELIST 結尾為止，此時 IBM MQ classes for JMS 會等待一段時間，然後再試一次。

如果設定 CCDTURL，當執行自動用戶端重新連線時， IBM MQ classes for JMS 會使用 CCDTURL 內容中指定的用戶端通道定義表來決定要重新連接的系統。

IBM MQ classes for JMS 一開始會剖析用戶端通道定義表，並尋找符合 QMANAGER 內容值的適當項目。 找到項目時，「 IBM MQ classes for JMS 」會嘗試使用該項目重新連接至所需的佇列管理程式。 如果可以建立與佇列管理程式的連線，則 IBM MQ classes for JMS 會重新開啟應用程式在自動用戶端重新連線之前已開啟的所有 IBM MQ 物件，並繼續像之前一樣執行。

如果無法建立與所需佇列管理程式的連線， IBM MQ classes for JMS 會在用戶端通道定義表中尋找另一個適當的項目，並嘗試使用該項目，依此類推。

當 IBM MQ classes for JMS 已嘗試用戶端通道定義表中所有適當的項目時，它們會等待一段時間，然後再重試重新連接。 為了執行新的重新連線嘗試， IBM MQ classes for JMS 會重新剖析用戶端通道定義表，並嘗試第一個適當的項目。 然後，他們會依序嘗試用戶端通道定義表中每一個適當的項目，直到重新連線發生或嘗試用戶端通道定義表中最後一個適當的項目為止，此時 IBM MQ classes for JMS 會等待一段時間，然後再試一次。

無論使用 CONNECTIONNAMELIST 或 CCDTURL，自動用戶端重新連線的程序都會繼續，直到 IBM MQ classes for JMS 順利重新連接至 QMANAGER 內容指定的佇列管理程式為止。

依預設，會依下列間隔進行重新連線嘗試:

- 第一次嘗試是在起始延遲 1 秒之後，加上隨機元素，最多 250 毫秒。
- 第二次嘗試會在第一次嘗試失敗之後 2 秒加上隨機間隔 (最多 500 毫秒)。
- 在第二次嘗試失敗之後，第三次嘗試會進行 4 秒，外加隨機間隔最多 1 秒。
- 在第三次嘗試失敗之後，會進行第四次嘗試 8 秒，外加隨機間隔最多 2 秒。
- 在第四次嘗試失敗之後，會進行第五次嘗試 16 秒，外加隨機間隔最多 4 秒。
- 第六次嘗試及所有後續嘗試都會進行 25 秒，加上前一次嘗試失敗之後最多 6 秒及 250 毫秒的隨機間隔。

重新連線嘗試會因部分固定及部分隨機的間隔而延遲。 這是為了防止所有已連接至不再可用的佇列管理程式的 IBM MQ classes for JMS 應用程式同步重新連接。

如果您需要增加預設值，以更精確地反映佇列管理程式回復或待命佇列管理程式變成作用中所需的時間量，請修改用戶端配置檔的通道段落中的 ReconDelay 屬性，如需相關資訊，請參閱 用戶端配置檔的通道段落。

IBM MQ classes for JMS 應用程式在自動重新連接之後是否繼續正確運作取決於其設計。 請閱讀相關主題，以瞭解如何設計應用程式可以使用自動重新連線功能。

指出佇列管理程式不再可用的原因碼

哪些原因碼指出在嘗試自動 IBM MQ classes for JMS 重新連線時，佇列管理程式不再可用或無法呼叫到。

第 240 頁的『自動 JMS 用戶端重新連線』 提供 JMSExceptions 及應用程式如何自動重新啟動的概觀， 第 240 頁的『使用自動 JMS 用戶端重新連線』 中的資訊詳述自動用戶端重新連線的需求。

下列資訊列出應用程式應該檢查的 IBM MQ 原因碼:

RC2009
MQRC_CONNECTION_BROKEN

RC2059
MQRC_Q_MGR_NOT_AVAILABLE

RC2161
MQRC_Q_MGR_QUIESCING

RC2162
MQRC_Q_MGR_STOPPING

RC2202
MQRC_CONNECTION_QUIESCING

RC2203
MQRC_CONNECTION_STOPPING

RC2223
MQRC_Q_MGR_NOT_ACTIVE

RC2279
MQ RC_CHANNEL_STOPPED_BY_USER

RC2537
MQRC_CHANNEL_NOT_AVAILABLE

RC2538
MQRC_HOST_NOT_AVAILABLE

擲出回企業應用程式的大部分 JMSExceptions 包含鏈結 MQException，其保留原因碼。 如果要實作上述清單中原因碼的重試邏輯，您的企業應用程式應該使用類似下列範例的程式碼來檢查這個鏈結的異常狀況:

```
} catch (JMSException ex) {
        Exception linkedEx = ex.getLinkedException();
        if (ex.getLinkedException() != null) {
            if (linkedEx instanceof MQException) {
                MQException mqException = (MQException) linkedEx;
                int reasonCode = mqException.reasonCode;
                // Handle the reason code accordingly
            }
        }
    }
```

相關概念
IBM MQ classes for JMS

在 *Java SE* 和 *Java EE* 環境中使用自動用戶端重新連線
您可以使用 IBM MQ 自動用戶端重新連線，以協助 Java SE 及 Java EE 環境中的各種高可用性 (HA) 及災難回復 (DR) 解決方案。

在不同平台上提供各種 HA 和 DR 解決方案:

- Multi 多重實例佇列管理程式是在不同伺服器上配置之相同佇列管理程式的實例 (請參閱 多重實例佇列管理程式)。 佇列管理程式的其中一個實例會定義為作用中實例，另一個實例會定義為待命實例。 如果作用中實例失敗，多重實例佇列管理程式會在待命伺服器上自動重新啟動。

  作用中及待命佇列管理程式都具有相同的佇列管理程式 ID (QMID)。 連接至多重實例佇列管理程式的 IBM MQ 用戶端應用程式可以配置為使用自動用戶端重新連線來自動重新連接至佇列管理程式的待命實例。

- Linux RDQM (抄寫的資料佇列管理程式) 是可在 Linux 平台上使用的高可用性解決方案 (請參閱 RDQM 高可用性)。 RDQM 配置由高可用性 (HA) 群組中所配置的三部伺服器組成，每一部伺服器都有一個佇列管理程式實例。 其中一個實例是執行中的佇列管理程式，它會同步將其資料抄寫至另外兩個實例。 如果執行此佇列管理程式的伺服器失敗，則另一個佇列管理程式實例會啟動並具有現行資料可操作。 佇列管理程式的三個實例會共用浮動 IP 位址，因此用戶端只需要配置單一 IP 位址。 連接至 RDQM 佇列管理程式的用戶端應用程式可以配置為使用自動用戶端重新連線來自動重新連接至佇列管理程式的待命實例。

- MQ Appliance HA 解決方案也可以由一對 IBM MQ 軟體驅動裝置提供 (請參閱 IBM MQ Appliance 說明文件中的 高可用性 和 災難回復)。 HA 佇列管理程式在其中一個應用裝置上執行，同時同步將資料抄寫至另一個應用裝置上佇列管理程式的待命實例。 如果主要應用裝置失敗，佇列管理程式會自動啟動並在另一個應用裝置上執行。 佇列管理程式的兩個實例可以配置為共用浮動 IP 位址，因此用戶端只需要配置單一 IP 位

址。 連接至「IBM MQ 應用裝置」上 HA 佇列管理程式的用戶端應用程式可以配置為使用自動用戶端重新連線來自動重新連接至佇列管理程式的待命實例。

註: 在 Java EE 環境 (例如 WebSphere Application Server) 內，不支援使用 IBM MQ classes for JMS 提供的功能來自動用戶端與啟動規格重新連線。 如果啟動規格所連接的佇列管理程式變成無法使用， IBM MQ 資源配接器會提供自己的機制來重新連接啟動規格。 如需相關資訊，請參閱 第 245 頁的『在 Java EE 環境中支援自動用戶端重新連線』。

相關概念

多重實例佇列管理程式

自動用戶端重新連線

相關參考

rdqm 高可用性

在 *Java SE* 環境中使用自動用戶端重新連線

使用在 Java SE 環境中執行之 IBM MQ classes for JMS 的應用程式可以透過 Connection Factory 內容 **CLIENTRECONNECTOPTIONS** 使用自動用戶端重新連線功能。

Connection Factory 內容 **CLIENTRECONNECTOPTIONS** 會使用另外兩個 Connection Factory 內容 **CONNECTIONNAMELIST** 和 **CCDTURL**，來決定如何連接執行佇列管理程式的伺服器。

CONNECTIONNAMELIST 內容

CONNECTIONNAMELIST 內容是以逗點區隔的清單，其中包含在用戶端模式中用於連接至佇列管理程式的主機名稱及埠資訊。 此內容可與 **QMANAGER** 及 **CHANNEL** 值搭配使用。 在應用程式使用 **CONNECTIONNAMELIST** 內容來建立用戶端連線時，IBM MQ classes for JMS 會嘗試以清單順序連接至每個主機。 如果第一個佇列管理程式主機無法使用，IBM MQ classes for JMS 會嘗試連接至清單中的下一個主機。 如果到達連線名稱清單的結尾而未建立連線，IBM MQ classes for JMS 會擲出 MQRC_QMGR_NOT_AVAILABLE IBM MQ 原因碼。

如果應用程式所連接的佇列管理程式失敗，則使用 **CONNECTIONNAMELIST** 連接至該佇列管理程式的任何應用程式皆會收到異常狀況，指出該佇列管理程式無法使用。 應用程式必須捕捉此異常狀況，並清除它正在使用的任何資源。 若要建立連線，應用程式必須使用 Connection Factory。 Connection Factory 會嘗試再次以清單順序連接至每個主機，但失敗的佇列管理程式現在將無法使用。 因此 Connection Factory 會嘗試連接至清單中的其他主機。

CCDTURL 內容

CCDTURL 內容包含指向「用戶端通道定義表 (CCDT)」的「統一資源定址器 (URL)」，此內容可與 **QMANAGER** 內容搭配使用。 CCDT 包含用於連接至 IBM MQ 系統上所定義佇列管理程式的用戶端通道清單。 如需 IBM MQ classes for JMS 如何使用 CCDT 的相關資訊，請參閱 第 239 頁的『搭配使用用戶端通道定義表與 IBM MQ classes for JMS』。

使用 CLIENTRECONNECTOPTIONS 內容在 IBM MQ classes for JMS 內啟用自動用戶端重新連線

使用 **CLIENTRECONNECTOPTIONS** 內容可在 IBM MQ classes for JMS 內啟用自動用戶端重新連線。 此內容可能的值如下:

ASDEF
自動用戶端重新連線行為，將由 IBM MQ 用戶端配置檔 (mqclient.ini) 的通道段落中指定的預設值定義。

DISABLED
停用自動用戶端重新連線。

QMGR
IBM MQ classes for JMS 會嘗試使用下列任一選項，連接至佇列管理程式 ID 與其所連接佇列管理程式相同的佇列管理程式:

- **CONNECTIONNAMELIST** 內容以及 **CHANNEL** 內容中定義的通道。

- **CCDTURL** 內容中定義的 CCDT。

任何

IBM MQ classes for JMS 會嘗試使用 **CONNECTIONNAMELIST** 內容或 **CCDTURL**，重新連接至具有相同名稱的佇列管理程式。

相關資訊

用戶端配置檔的 CHANNELS 段落

在 *Java EE* 環境中使用自動用戶端重新連線

IBM MQ 資源配接器可部署至 Java EE (Java Platform, Enterprise Edition) 環境， WebSphere Application Server IBM MQ 傳訊提供者會使用 IBM MQ classes for JMS 來與 IBM MQ 佇列管理程式通訊。 IBM MQ 資源配接器和 WebSphere Application Server IBM MQ 傳訊提供者提供一些機制，可讓啟動規格、WebSphere Application Server 接聽器埠和在用戶端儲存器內執行的應用程式自動重新連接至佇列管理程式。 Enterprise JavaBeans (EJB) 及 Web 型應用程式需要實作自己的重新連線邏輯。

註: 不支援使用 IBM MQ classes for JMS 所提供的功能，將自動用戶端與啟動規格重新連線 (請參閱 第 240 頁的『自動 JMS 用戶端重新連線』)。 如果啟動規格所連接的佇列管理程式變成無法使用， IBM MQ 資源配接器會提供自己的機制來重新連接啟動規格。

資源配接器提供的機制是由下列項目所控制:

- IBM MQ 資源配接器內容 **reconnectionRetryCount**。
- IBM MQ 資源配接器內容 **reconnectionRetryInterval**。
- 啟動規格內容 **connectionNameList**。

如需這些內容的相關資訊，請參閱 第 377 頁的『ResourceAdapter 物件內容的配置』。

不支援在訊息驅動 Bean 應用程式的 onMessage() 方法內使用自動用戶端重新連線，或在 Java Platform, Enterprise Edition 環境內執行的任何其他應用程式。 如果應用程式所連接的佇列管理程式變成無法使用，則應用程式需要實作自己的重新連線邏輯。 如需相關資訊，請參閱 第 251 頁的『在 Java EE 應用程式中實作重新連線邏輯』。

在 *Java EE* 環境中支援自動用戶端重新連線

在 Java EE 環境 (例如 WebSphere Application Server) 內， IBM MQ 資源配接器和 WebSphere Application Server IBM MQ 傳訊提供者提供許多機制，可讓應用程式自動重新連接至佇列管理程式。 但是，在部分情況下，此支援有一些限制。

可以部署到 Java EE 環境和 WebSphere Application Server IBM MQ 傳訊提供者中的 IBM MQ 資源配接器，會使用 IBM MQ classes for JMS 來與 IBM MQ 佇列管理程式通訊。

下表彙總 IBM MQ 資源配接器及 WebSphere Application Server IBM MQ 傳訊提供者，針對自動用戶端重新連線所提供的支援。

表 *45: Java EE* 環境中自動用戶端重新連線選項的支援摘要

自動重新連線的選項	CONNECTIONNAMELIST 內容	CCDTURL 內容	CLIENTRECONNECTOPTIONS 內容	自動用戶端重新連線的替代方法
啟動規格	受支援但有一些限制	受支援但有一些限制	不支援	Java EE 環境和啟動規格提供自己的重新連線機制
WebSphere Application Server 接聽器埠	受支援但有一些限制	受支援但有一些限制	不支援	WebSphere Application Server 可提供自己的重新連線機制
Enterprise JavaBean 及 Web 型應用程式	受支援但有一些限制	受支援但有一些限制	不支援	應用程式必須實作自己的重新連線邏輯
在用戶端儲存器內執行的應用程式	支援	支援	支援	不適用

安裝在 Java EE 環境中的訊息驅動 Bean 應用程式 (例如 IBM MQ classes for JMS) 可以使用啟動規格來處理 IBM MQ 系統上的訊息。 啟動規格用來偵測抵達 IBM MQ 系統的訊息，並將它們遞送至訊息驅動 Bean 以進行處理。 訊息驅動 Bean 也可以從其 **onMessage()** 方法內部建立與 IBM MQ 系統的更多連線。 如需這些連線如何才能使用自動用戶端重新連線的相關資訊，請參閱 Enterprise JavaBean 及 Web 型應用程式。

啟動規格
對於啟動規格，**CONNECTIONNAMELIST** 及 **CCDTURL** 內容受支援但有一些限制，**CLIENTRECONNECTOPTIONS** 內容則不受支援。

安裝在 Java EE 環境 (例如 WebSphere Application Server) 中的訊息驅動 Bean (MDB) 應用程式可以利用啟動規格來處理 IBM MQ 系統上的訊息。

啟動規格可用來偵測到達 IBM MQ 系統的訊息，然後將其遞送至 MDB 以進行處理。 本節將討論啟動規格如何監視 IBM MQ 系統。

MDB 也可以從其 onMessage() 方法內部建立與 IBM MQ 系統的其他連線。

在 第 249 頁的『Enterprise JavaBean 及 Web 型應用程式』中，可以找到這些連線如何才能使用自動用戶端重新連線的相關詳細資料。

CONNECTIONNAMELIST 內容

在啟動時，啟動規格會嘗試使用下列資訊來連接至佇列管理程式：

- **QMANAGER** 內容中指定的佇列管理程式
- **CHANNEL** 內容中提及的通道
- **CONNECTIONNAMELIST** 內第一個項目中的主機名稱及埠資訊

如果啟動規格無法使用清單中的第一個項目連接至佇列管理程式，則會移至第二個項目，依此類推，直到已建立與佇列管理程式的連線或已到達清單結尾為止。

如果啟動規格無法使用 **CONNECTIONNAMELIST** 中的任何項目連接至指定的佇列管理程式，則啟動規格會停止且必須重新啟動。

一旦啟動規格正在執行中，啟動規格即會從 IBM MQ 系統取得訊息，然後將這些訊息遞送至 MDB 以進行處理。

如果佇列管理程式在處理訊息時失敗，Java EE 環境會偵測到失敗並嘗試重新連接啟動規格。

啟動規格在執行重新連線嘗試時，會如同之前一樣使用 **CONNECTIONNAMELIST** 內容中的資訊。

如果啟動規格嘗試 **CONNECTIONNAMELIST** 中的所有項目，但仍無法連接至佇列管理程式，則會等待由 IBM MQ 資源配接器內容 **reconnectionRetryInterval** 指定的一段時間，然後再重試。

IBM MQ 資源配接器內容 **reconnectionRetryCount**，會定義在啟動規格停止並需要手動重新啟動之前，將會進行的連續重新連線嘗試次數

一旦啟動規格重新連接至 IBM MQ 系統，Java EE 環境即會執行所需的任何交易式清理，並繼續將訊息遞送至 MDB 以進行處理。

為了使交易式清理正確運作，Java EE 環境必須能夠存取失敗佇列管理程式的日誌。

如果啟動規格是與參與 XA 交易的交易式 MDB 搭配使用，且正在連接至多重實例佇列管理程式，則 **CONNECTIONNAMELIST** 必須包含作用中及待命佇列管理程式實例的項目。

這表示如果 Java EE 環境需要執行交易回復，無論該環境在發生失敗狀況之後重新連接至哪個佇列管理程式，皆可存取佇列管理程式日誌。

如果交易式 MDB 與獨立式佇列管理程式搭配使用，則 **CONNECTIONNAMELIST** 內容必須包含單一項目，以確保啟動規格在失敗之後一律重新連接至在相同系統上執行的相同佇列管理程式。

CCDTURL 內容

啟動時，啟動規格會使用用戶端通道定義表 (CCDT) 中的第一個項目，嘗試連接至 **QMANAGER** 內容中指定的佇列管理程式。

如果啟動規格無法使用表格中的第一個項目連接至佇列管理程式，則會移至第二個項目，依此類推，直到已建立與佇列管理程式的連線或已到達表格結尾為止。

如果啟動規格無法使用 CCDT 中的任何項目連接至指定的佇列管理程式，則啟動規格會停止且必須重新啟動。

一旦啟動規格正在執行中，啟動規格即會從 IBM MQ 系統取得訊息，然後將這些訊息遞送至 MDB 以進行處理。

如果佇列管理程式在處理訊息時失敗，Java EE 環境會偵測到失敗並嘗試重新連接啟動規格。

啟動規格在執行重新連線嘗試時，會如同之前一樣使用 CCDT 內容中的資訊。

如果啟動規格嘗試 CCDT 中的所有項目，但仍無法連接至佇列管理程式，則會等待由 IBM MQ 資源配接器內容 **reconnectionRetryInterval** 指定的一段時間，然後再重試。

IBM MQ 資源配接器內容 **reconnectionRetryCount**，會定義在啟動規格停止並需要手動重新啟動之前，將會進行的連續重新連線嘗試次數

一旦啟動規格重新連接至 IBM MQ 系統，Java EE 環境即會執行所需的任何交易式清理，並繼續將訊息遞送至 MDB 以進行處理。

為了使交易式清理正確運作，Java EE 環境必須能夠存取失敗佇列管理程式的日誌。

如果啟動規格是與參與 XA 交易的交易式 MDB 搭配使用，且正在連接至多重實例佇列管理程式，則 CCDT 必須包含作用中及待命佇列管理程式實例的項目。

這表示如果 Java EE 環境需要執行交易回復，無論該環境在發生失敗狀況之後重新連接至哪個佇列管理程式，皆可存取佇列管理程式日誌。

如果交易式 MDB 是與獨立式佇列管理程式搭配使用，則 CCDT 必須包含單一項目，以確保啟動規格在發生失敗狀況之後，一律重新連接至在相同系統上執行的相同佇列管理程式。

請確保已針對與啟動規格搭配使用的 CCDT 上的 **AFFINITY** 內容設定預設值 *PREFERRED*，以便建立與相同作用中佇列管理程式的連線。

CLIENTRECONNECTOPTIONS 內容

啟動規格可提供自己的重新連線功能。 如果這些規格所連接的佇列管理程式失敗，則提供的功能將容許這些規格自動重新連接至 IBM MQ 系統。

因此，不支援 IBM MQ classes for JMS 提供的自動用戶端重新連線功能。

您必須針對 Java EE 中使用的所有啟動規格，將 **CLIENTRECONNECTOPTIONS** 內容設為 *DISABLED* 。

WebSphere Application Server 接聽器埠
安裝在 WebSphere Application Server 中的訊息驅動 Bean (MDB) 應用程式，亦可使用接聽器埠來處理 IBM MQ 系統上的訊息。

接聽器埠可用來偵測到達 IBM MQ 系統的訊息，然後將其遞送至 MDB 以進行處理。 本主題說明接聽器埠如何監視 IBM MQ 系統。

MDB 也可以從其 onMessage() 方法內部建立與 IBM MQ 系統的其他連線。

如需這些連線如何使用自動用戶端重新連線的相關資訊，請參閱 第 249 頁的『Enterprise JavaBean 及 Web 型應用程式』

對於 WebSphere Application Server 接聽器埠：

- 支援 **CONNECTIONNAMELIST** 及 **CCDTURL** 但有一些限制
- 不支援 **CLIENTRECONNECTOPTIONS**

CONNECTIONNAMELIST 內容

當連接至 IBM MQ 時，接聽器埠會使用 JMS 連線儲存區，因此會受限於使用連線儲存區的含意。 如需進一步資訊，請參閱第 246 頁的『啟動規格』。

如果沒有可用連線，且尚未從此 Connection Factory 建立連線數目上限，則會使用 **CONNECTIONNAMELIST** 內容來嘗試建立與 IBM MQ 的新連線。

如果無法存取 **CONNECTIONNAMELIST** 中的所有 IBM MQ 系統，則接聽器埠會停止。

然後，接聽器埠會等待由訊息接聽器服務自訂內容 **RECOVERY.RETRY.INTERVAL** 指定的一段時間，並再次嘗試重新連接。

此重新連線嘗試會檢查連線儲存區中是否有任何可用連線，以防萬一在兩次連線嘗試之間傳回可用連線。 如果沒有可用連線，接聽器埠會如同之前一樣使用 **CONNECTIONNAMELIST**。

一旦接聽器埠重新連接至 IBM MQ 系統，Java EE 環境即會執行所需的任何交易式清理，並繼續將訊息遞送至 MDB 以進行處理。

為了使交易式清理正確運作，Java EE 環境必須能夠存取失敗佇列管理程式的日誌。

如果接聽器埠是與參與 XA 交易的交易式 MDB 搭配使用，且正在連接至**多重實例佇列管理程式**，則 **CONNECTIONNAMELIST** 必須包含作用中及待命佇列管理程式實例的項目。

這表示如果 Java EE 環境需要執行交易回復，無論該環境在發生失敗狀況之後重新連接至哪個佇列管理程式，皆可存取佇列管理程式日誌。

如果交易式 MDB 與獨立式佇列管理程式搭配使用，則 **CONNECTIONNAMELIST** 內容必須包含單一項目，以確保啟動規格在失敗之後一律重新連接至在相同系統上執行的相同佇列管理程式。

CCDTURL 內容

啟動時，接聽器埠會使用 CCDT 中的第一個項目，嘗試連接至 **QMANAGER** 內容中指定的佇列管理程式。

如果接聽器埠無法使用表格中的第一個項目連接至佇列管理程式，則會移至第二個項目，依此類推，直到已建立與佇列管理程式的連線或已到達表格結尾為止。

如果接聽器埠無法使用 CCDT 中的任何項目連接至指定的佇列管理程式，則接聽器埠會停止。

然後，接聽器埠會等待由訊息接聽器服務自訂內容 **RECOVERY.RETRY.INTERVAL** 指定的一段時間，並再次嘗試重新連接。

此重新連線嘗試會如同之前一樣試用 CCDT 中的所有項目。

一旦接聽器埠正在執行中，它即會從 IBM MQ 系統取得訊息，然後將這些訊息遞送至 MDB 以進行處理。

如果佇列管理程式在處理訊息時失敗，Java EE 環境會偵測到失敗並嘗試重新連接此接聽器埠。 在執行重新連線嘗試時，接聽器埠會使用 CCDT 中的資訊。

如果接聽器埠嘗試 CCDT 中的所有項目，但仍無法連接至佇列管理程式，則會等待由 **RECOVERY.RETRY.INTERVAL** 內容指定的一段時間，然後再重試。

訊息接聽器服務內容 **MAX.RECOVERY.RETRIES**，會定義在接聽器埠停止並需要手動重新啟動之前，將會進行的連續重新連線嘗試次數。

一旦接聽器埠重新連接至 IBM MQ 系統，Java EE 環境即會執行所需的任何交易式清理，並繼續將訊息遞送至 MDB 以進行處理。

為了使交易式清理正確運作，Java EE 環境必須能夠存取失敗佇列管理程式的日誌。

如果接聽器埠是與參與 XA 交易的交易式 MDB 搭配使用，且正在連接至多重實例佇列管理程式，則 CCDT 必須包含作用中及待命佇列管理程式實例的項目。

這表示如果 Java EE 環境需要執行交易回復，無論該環境在發生失敗狀況之後重新連接至哪個佇列管理程式，皆可存取佇列管理程式日誌。

如果交易式 MDB 是與獨立式佇列管理程式搭配使用，則 CCDT 必須包含單一項目，以確保接聽器埠在發生失敗狀況之後，一律重新連接至在相同系統上執行的相同佇列管理程式。

請確保已針對與接聽器搭配使用的 CCDT 上的 **AFFINITY** 內容設定預設值 *PREFERRED*，以便建立與相同作用中佇列管理程式的連線。

CLIENTRECONNECTOPTIONS 內容

接聽器埠可提供自己的重新連線功能。 如果這些接聽器埠所連接的佇列管理程式失敗，則提供的功能可讓這些接聽器埠自動重新連接至 IBM MQ 系統。

因此，不支援 IBM MQ classes for JMS 提供的自動用戶端重新連線功能。

對於 Java EE 中使用的所有接聽器埠，您必須將 **CLIENTRECONNECTOPTIONS** 內容設為 *DISABLED* 。

Enterprise JavaBean 及 *Web* 型應用程式
Enterprise JavaBean (EJB) 應用程式以及在 Web 儲存器內執行的應用程式（例如 Servlet）， 會使用 JMS Connection Factory 來建立與 IBM MQ 佇列管理程式的連線。

下列限制適用於 EJB 及 Web 型應用程式:

- 支援 **CONNECTIONNAMELIST** 及 **CCDTURL** 但有一些限制
- 不支援 **CLIENTRECONNECTOPTIONS**

CONNECTIONNAMELIST 內容

如果 Java EE 環境為 JMS 連線提供連線儲存區，請參閱第 250 頁的『使用連線儲存區中的 CONNECTIONNAMELIST 或 CCDT』，以取得這將如何影響 **CONNECTIONNAMELIST** 內容之行為的相關資訊。

如果 Java EE 環境不提供 JMS 連線儲存區， 應用程式使用 **CONNECTIONNAMELIST** 內容的方式與 Java SE 應用程式相同。

如果應用程式是與參與 XA 交易的交易式 MDB 搭配使用，且正在連接至多重實例佇列管理程式，則 **CONNECTIONNAMELIST** 必須包含作用中及待命佇列管理程式實例的項目。

這表示如果 Java EE 環境需要執行交易回復，無論該環境在發生失敗狀況之後重新連接至哪個佇列管理程式，皆可存取佇列管理程式日誌。

如果應用程式與獨立式佇列管理程式搭配使用，則 **CONNECTIONNAMELIST** 內容必須包含單一項目，以確保在失敗之後，應用程式一律重新連接至在相同系統上執行的相同佇列管理程式。

CCDTURL 內容

如果 Java EE 環境為 JMS 連線提供連線儲存區，請參閱第 250 頁的『使用連線儲存區中的 CONNECTIONNAMELIST 或 CCDT』，以取得這將如何影響 **CCDTURL** 內容之行為的相關資訊。

如果 Java EE 環境不提供 JMS 連線儲存區， 應用程式使用 **CCDTURL** 內容的方式與 Java SE 應用程式相同。

如果應用程式是與參與 XA 交易的交易式 MDB 搭配使用，且正在連接至多重實例佇列管理程式，則 CCDT 必須包含作用中及待命佇列管理程式實例的項目。

這表示如果 Java EE 環境需要執行交易回復，無論該環境在發生失敗狀況之後重新連接至哪個佇列管理程式，皆可存取佇列管理程式日誌。

如果應用程式是與獨立式佇列管理程式搭配使用，則 CCDT 必須包含單一項目，以確保啟動規格在發生失敗狀況之後，一律重新連接至在相同系統上執行的相同佇列管理程式。

CLIENTRECONNECTOPTIONS 內容

對於在 Web 儲存器中執行的 EJB 或應用程式所使用的所有 JMS Connection Factory ， 您必須將 **CLIENTRECONNECTOPTIONS** 內容設為 *DISABLED* 。

在所使用的佇列管理程式失敗時需要自動重新連接至新佇列管理程式的應用程式，需要實作自己的重新連線邏輯。 如需相關資訊，請參閱 第 251 頁的『在 Java EE 應用程式中實作重新連線邏輯』。

實務範例: WebSphere Application Server 搭配 IBM MQ
實務範例: WebSphere Application Server Liberty 設定檔與 IBM MQ

在用戶端儲存器內執行的應用程式
部分 Java EE 環境（例如 WebSphere Application Server），提供可用於執行 Java SE 應用程式的用戶端儲存器。

在這些環境內執行的應用程式，會使用 JMS Connection Factory 來連接至 IBM MQ 佇列管理程式。

對於在用戶端儲存器內執行的應用程式：

- 完全支援 **CONNECTIONNAMELIST** 及 **CCDTURL**
- 完全支援 **CLIENTRECONNECTOPTIONS**

CONNECTIONNAMELIST 內容

如果 Java EE 環境為 JMS 連線提供連線儲存區，請參閱第 250 頁的『使用連線儲存區中的 CONNECTIONNAMELIST 或 CCDT』，以取得這將如何影響 **CONNECTIONNAMELIST** 內容之行為的相關資訊。

如果 Java EE 環境不提供 JMS 連線儲存區， 應用程式使用 **CONNECTIONNAMELIST** 內容的方式與 Java SE 應用程式相同。

CCDTURL 內容

如果 Java EE 環境為 JMS 連線提供連線儲存區，請參閱第 250 頁的『使用連線儲存區中的 CONNECTIONNAMELIST 或 CCDT』，以取得這將如何影響 **CCDTURL** 內容之行為的相關資訊。

如果 Java EE 環境不提供 JMS 連線儲存區， 應用程式使用 **CCDTURL** 內容的方式與 Java SE 應用程式相同。

使用連線儲存區中的 *CONNECTIONNAMELIST* 或 *CCDT*
部分 Java EE 環境 (例如 WebSphere Application Server) 提供 JMS 連線儲存區。 可用來執行 Java SE 應用程式的儲存器。

使用已在 Java EE 環境中定義的 Connection Factory 來建立連線的應用程式，會從此 Connection Factory 的連線儲存區中取得現有可用連線，如果此連線儲存區中沒有適合的連線，則會取得新連線。

如果已配置 Connection Factory 並已定義 **CONNECTIONNAMELIST** 或 **CCDTURL** 內容，則這可能會有一些影響。

第一次使用 Connection Factory 來建立連線時， Java EE 環境會使用 **CONNECTIONNAMELIST**。 或 **CCDTURL**，來建立與 IBM MQ 系統的新連線。 當此連線不再需要時，即會將其傳回連線儲存區，在連線儲存區中，此連線將變為可供重複使用。

如果其他物件從 Connection Factory 建立連線，Java EE 環境會從連線儲存區傳回此連線，而不會使用 **CONNECTIONNAMELIST** 或 **CCDTURL** 內容來建立新連線。

如果在佇列管理程式實例失敗時正在使用某條連線，則會捨棄此連線。 但是，可能不會捨棄連線儲存區的內容，這表示該儲存區可能仍包含與再也不在執行中的佇列管理程式之連線。

在此狀況下，下次要求從 Connection Factory 建立連線時，將會傳回與失敗佇列管理程式的連線。 使用此連線的任何嘗試皆會失敗，因為此佇列管理程式再也不在執行中，從而導致捨棄此連線。

只有在連線儲存區為空的時，Java EE 環境才會使用 **CONNECTIONNAMELIST** 或 **CCDTURL** 內容，來建立與 IBM MQ 的新連線。

由於使用 **CONNECTIONNAMELIST** 及 CCDT 來建立 JMS 連線的方式，所擁有的連線儲存區還可能包含與不同 IBM MQ 系統的連線。

例如，假設已配置 Connection Factory 並將 **CONNECTIONNAMELIST** 內容設定為下列值：

```
CONNECTIONNAMELIST = hostname1(port1), hostname2(port2)
```

假設在某個應用程式第一次嘗試從此 Connection Factory 建立與獨立式佇列管理程式的連線時，在 `hostname1(port1)` 系統上執行的此佇列管理程式無法存取。 這表示此應用程式會以與 `hostname2(port2)` 上執行的佇列管理程式建立連線而結束。

現在有另一個應用程式並且從相同 Connection Factory 建立 JMS 連線。 現在可以使用 hostname1(port1) 上的佇列管理程式，因此會建立與此 IBM MQ 系統的新 JMS 連線，並傳回給應用程式。

當這兩個應用程式皆已完成時，它們會關閉其 JMS 連線，從而導致將這些連線傳回連線儲存區。

結果便造成 Connection Factory 的連線儲存區現在包含下列兩條 JMS 連線：

- 一條是與 hostname1(port1) 上執行的佇列管理程式的連線
- 一條是與 hostname2(port2) 上執行的佇列管理程式的連線

這可能會導致與交易回復相關的問題。 如果 Java EE 系統需要回復交易，則它必須可以連接至有權存取交易日誌的佇列管理程式。

在 *Java EE* 應用程式中實作重新連線邏輯
如果佇列管理程式失敗，要自動重新連接的 Enterprise JavaBeans 及 Web 型應用程式需要實作自己的重新連線邏輯。

下列選項提供如何達成此目的的相關資訊。

容許應用程式失敗

此方法不需要進行應用程式變更，但需要對 Connection Factory 定義進行管理重新配置以包含 **CONNECTIONNAMELIST** 內容。 但是，此方法需要呼叫程式能夠適當地處理失敗。 請注意，這對於與連線失敗不相關的失敗（例如 MQRC_Q_FULL）也是必要的。

此程序的程式碼範例：

```
public class SimpleServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
         try {
 // get connection factory/ queue
 InitialContext ic = new InitialContext();
 ConnectionFactory cf =
           (ConnectionFactory)ic.lookup("java:comp/env/jms/WMQCF");
 Queue q = (Queue) ic.lookup("java:comp/env/jms/WMQQueue");

 // send a message
 Connection c = cf.createConnection();
 Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
 MessageProducer p = s.createProducer(q);
 Message m = s.createTextMessage();
 p.send(m);

 // done, release the connection
 c.close();
 }
 catch (JMSException je) {
 // process exception
   }
 }
}
```

上述程式碼假設此 Servlet 使用的 Connection Factory 已定義 **CONNECTIONNAMELIST** 內容。

當 Servlet 第一次處理時，會使用 **CONNECTIONNAMELIST** 內容來建立新連線，並假設其他連接至相同佇列管理程式的應用程式沒有可用的儲存連線。

在進行 close() 呼叫之後釋放此連線時，即會將此連線傳回儲存區並在 Servlet 下次執行時重複使用此連線（而不參照 **CONNECTIONNAMELIST**），直到發生連線失敗為止，而在此時則會產生 CONNECTION_ERROR_OCCURRED 事件。 此事件會提示儲存區毀損失敗連線。

在應用程式下次執行時，由於沒有可用的儲存連線，因此會使用 **CONNECTIONNAMELIST** 來連接至第一個可用的佇列管理程式。 如果已進行佇列管理程式失效接手（例如，此失敗不是暫時網路失敗），則一旦備用實例可用，Servlet 即會連接至備用實例。

如果應用程式中涉及其他資源（例如資料庫），則適當的做法可能是指出應用程式伺服器應該回復交易。

在應用程式內處理重新連線

如果呼叫程式無法從 Servlet 處理失敗，則必須在應用程式內處理重新連線。 如下範例所示，若要在應用程式內處理重新連線，應用程式需要要求新連線，如此便能快取其透過 JNDI 查閱的 Connection Factory，以及處理 JMSException（例如，JMSCMQ0001: WebSphere MQ 呼叫失敗，compcode 為'2' ('MQCC_FAILED')，reason 為 '2009' ('MQRC_CONNECTION_BROKEN')）。

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

  // get connection factory/ queue
  InitialContext ic = new InitialContext();
  ConnectionFactory cf = (ConnectionFactory)
              ic.lookup("java:comp/env/jms/WMQCF");
  Destination destination = (Destination) ic.lookup("java:comp/env/jms/WMQQueue");

  setupResources();

  // loop sending messages
  while (!sendComplete) {
    try {
      // create the next message to send
      msg.setText("message sent at "+new Date());
      // and send it
      producer.send(msg);
    }
    catch (JMSException je) {
        // drive reconnection
        setupResources();
    }
  }
```

在下列範例中，setupResources() 會建立 JMS 物件，並包含用於處理非即時重新連線的休眠及重試迴圈。 實際上，此方法會阻止許多重新連接嘗試。 請注意，為了清楚說明，已省略結束條件。

```
 private void setupResources() {

    boolean connected = false;
    while (!connected) {
      try {
        connection = cf.createConnection(); // cf cached from JNDI lookup
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        msg = session.createTextMessage();
        producer = session.createProducer(destination); // destination cached from JNDI lookup
        // no exception? then we connected ok
        connected = true;
      }
      catch (JMSException je) {
        // sleep and then have another attempt
        try {Thread.sleep(30*1000);} catch (InterruptedException ie) {}
      }
    }
```

如果應用程式管理重新連線，則應用程式釋放保留給其他資源（無論這些資源是其他 IBM MQ 佇列管理程式，還是資料庫等其他後端服務）的任何連線，是非常重要的。 完成與新的 IBM MQ 佇列管理程式實例的重新連線後，您必須重新建立這些連線。 如果您未重新建立連線，則會在嘗試重新連線期間不必要地保留應用程式伺服器資源，而且它們可能在要重複使用前已逾時。

使用 WorkManager

對於處理時間大於幾十秒的長時間應用（例如批次處理），可以使用 WebSphere Application Server WorkManager。 WebSphere Application Server 的程式碼片段範例如下所示：

```
public class BatchSenderServlet extends HttpServlet  {

  private WorkManager workManager = null;
  private MessageSender sender; // background sender WorkImpl

  public void init() throws ServletException {
    InitialContext ctx = new InitialContext();
```

```
    workManager = (WorkManager)ctx.lookup(java:comp/env/wm/default);
    sender = new MessageSender(5000);
    workManager.startWork(sender);
  }

  public void destroy() {
    sender.halt();
  }

  public void doGet(HttpServletRequest req, HttpServletResponse res)
                                 throws ServletException, IOException {
    res.setContentType("text/plain");
    PrintWriter out = res.getWriter();
    if (sender.isRunning()) {
      out.println(sender.getStatus());
    }
}
```

其中 web.xml 包含:

```
<resource-ref>
      <description>WorkManager</description>
      <res-ref-name>wm/default</res-ref-name>
      <res-type>com.ibm.websphere.asynchbeans.WorkManager</res-type>
      <res-auth>Container</res-auth>
      <res-sharing-scope>Shareable</res-sharing-scope>
   </resource-ref>
```

現在將透過工作介面來實作批次:

```
import com.ibm.websphere.asynchbeans.Work;

public class MessageSender implements Work {

  public MessageSender(int messages) {numberOfMessages = messages;}

  public void run() {
    // get connection factory/ queue
    InitialContext ic = new InitialContext();
    ConnectionFactory cf = (ConnectionFactory)
               ic.lookup("java:comp/env/jms/WMQCF");
    Destination destination = (Destination) ic.lookup("jms/WMQQueue");

    setupResources();

    // loop sending messages
    while (!sendComplete) {
      try {
        // create the next message to send
        msg.setText("message sent at "+new Date());
        // and send it
        producer.send(msg);
        // are we finished?
        if (sendCount == numberOfMessages) {sendComplete = true);
      }
      catch (JMSException je) {
          // drive reconnection
          setupResources();
      }
  }

  public boolean isRunning() {return !sendComplete;}

  public void release() {sendComplete = true;}
```

如果批次處理花費較長時間執行，例如大型訊息、慢速網路或大量資料庫存取（尤其是在伴隨著慢速失效接手時），則伺服器會開始輸出當掉的執行緒警告（類似於下列範例）：

WSVR0605W: 執行緒 "WorkManager.DefaultWorkManager : 0" (00000035) 已處於作用中狀態長達 694061 毫秒，因此可能已當掉。 伺服器中總計有 1 個執行緒可能已當掉。

減少批次大小或增加當掉的執行緒逾時值，即可將這些警告減至最少。 但是，在 EJB（適用於批次傳送）或訊息驅動 Bean（適用於消費或消費及回覆）處理中實作此處理，通常會更好。

請注意，應用程式管理的重新連線不會提供處理執行時期錯誤的一般解決方案，因此應用程式仍必須處理與連線失敗不相關的錯誤。

例如，嘗試將訊息放置到已滿的佇列 (2053 MQRC_Q_FULL)，或嘗試使用無效的安全認證來連接至佇列管理程式 (2035 MQRC_NOT_AUTHORIZED)。

應用程式還必須處理在進行失效接手時沒有任何實例立即可用的 2059 MQRC_Q_MGR_NOT_AVAILABLE 錯誤。 應用程式可以透過在發生這些錯誤時報告 JMS 異常狀況（而非以無聲自動方式嘗試重新連接）來達成此目的。

IBM MQ classes for JMS 物件儲存區
在 Java EE 外部使用連線儲存區的形式有助於減少整體負載，例如，從使用架構或部署至雲端環境的部分獨立式應用程式，以及從更多用戶端連線至 `QueueManagers`，導致應用程式及佇列管理程式的伺服器合併增加。

在 Java EE 程式設計模型內，有各種使用中物件的明確定義生命週期。 訊息驅動 Bean (MDB) 受到最大限制，而 Servlet 提供更多自由。 因此， Java EE 伺服器內可用的儲存區作業選項符合所使用的各種程式設計模型。

使用 Java SE (或使用另一個架構，例如 Spring)，程式設計模型非常靈活。 因此，單一合併策略並不適合所有方案。 您應該考量是否有適當的架構可執行任何形式的儲存區作業，例如 Spring。

要使用的儲存區策略視應用程式執行所在的環境而定。

Java EE 環境中的物件儲存區作業
Java EE 應用程式伺服器提供可供訊息驅動 Bean 應用程式、Enterprise Java Beans 和 Servlet 使用的連線儲存區作業功能。

WebSphere Application Server 會維護 JMS 提供者的連線儲存區以提高效能。 在應用程式建立 JMS 連線時，應用程式伺服器會判斷可用連線儲存區中是否已經存在連線。 如果存在，則會將連線傳回應用程式；否則會建立新連線。

第 254 頁的圖 41 顯示啟動規格及接聽器埠在標準模式中如何建立 JMS 連線，並使用該連線來監視訊息目的地。

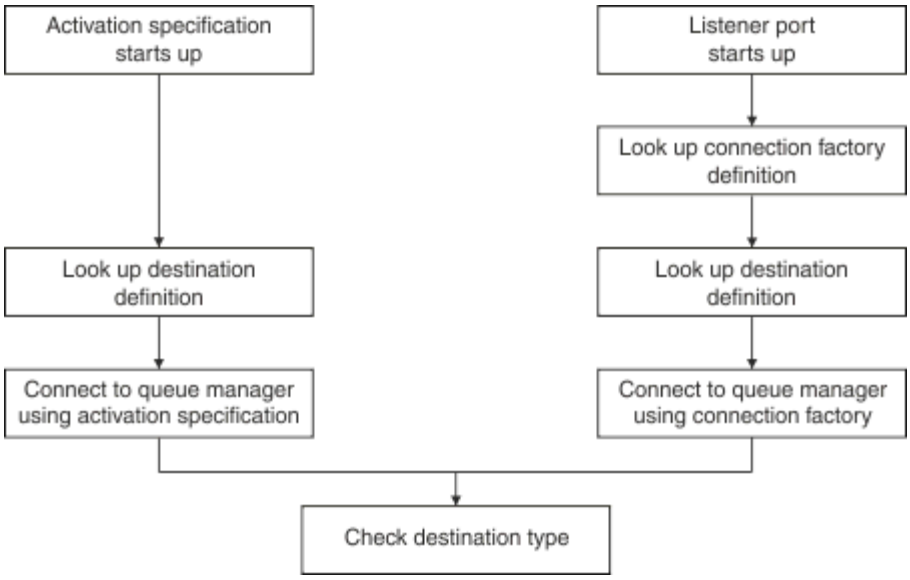


圖 *41:* 標準模式

當使用 IBM MQ 傳訊提供者時，執行出埠傳訊的應用程式 (例如 Enterprise Java Beans 和 Servlet) 以及訊息驅動 Bean 接聽器埠元件，可以利用這些連線儲存區。

IBM MQ 傳訊提供者啟動規格會使用 IBM MQ 資源配接器提供的連線儲存區功能。 如需相關資訊，請參閱 Configuring properties for the WebSphere MQ resource adapter。

第 258 頁的『使用連線儲存區的範例』說明執行出埠傳訊的應用程式及接聽器埠，在建立 JMS 連線時如何使用可用儲存區。

第 260 頁的『可用連線儲存區維護執行緒』說明在應用程式或接聽器埠完成連線時，這些連線將會發生的情況。

第 261 頁的『儲存區維護執行緒範例』說明如何清除可用連線儲存區，以防止 JMS 連線過時。

WebSphere Application Server 對可以從 Factory 建立的連線數目具有限制，此限制是由 Connection Factory 的 *maximumconnections* 內容指定。 此內容的預設值為 10，這表示在任何時刻最多可以從 Factory 建立 10 條連線。

每個 Factory 皆具有相關聯的可用連線儲存區。 在應用程式伺服器啟動時，可用連線儲存區是空的。 Factory 的可用儲存區中可以存在的連線數目上限，也是由 Maximum connections 內容指定。

提示: 對於 JMS 2.0，可以使用 Connection Factory 來建立這兩種連線和環境定義。 因此，連線儲存區可以與包含混合連線和環境定義的 Connection Factory 相關聯。 建議僅將一個 Connection Factory 用於建立連線或建立環境定義。 這可確保該 Connection Factory 的連線儲存區只包含一種類型的物件，從而使該儲存區更為有效率。

如需連線儲存區在 WebSphere Application Server 中如何運作的相關資訊，請參閱 配置 JMS 連線的連線儲存區。 若為其他應用程式伺服器，請參閱適當的應用程式伺服器文件。

如何使用連線儲存區

每個 JMS Connection Factory 皆具有與其相關聯的連線儲存區，而連線儲存區又會包含零條以上的 JMS 連線。 每條 JMS 連線皆具有相關聯的 JMS 階段作業儲存區，而每個 JMS 階段作業儲存區又會包含零個以上的 JMS 階段作業。

第 255 頁的圖 42 顯示這些物件之間的關係。

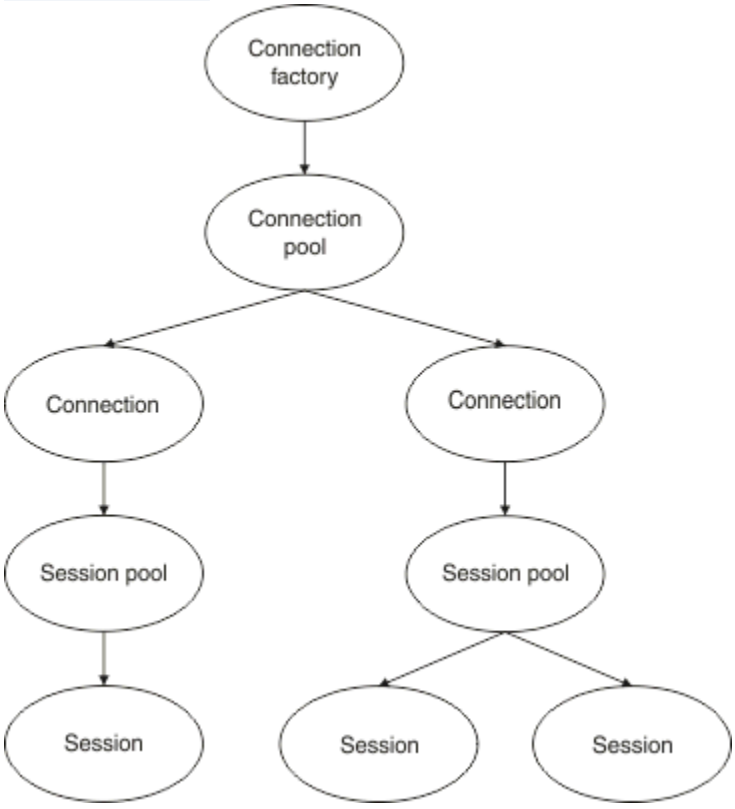


圖 *42:* 連線儲存區及階段作業儲存區

在接聽器埠啟動時，或是要執行出埠傳訊的應用程式使用 Factory 來建立連線時，該埠或應用程式會呼叫下列其中一個方法:

- **connectionFactory.createConnection()**
- **ConnectionFactory.createConnection(String, String)**
- **QueueConnectionFactory.createQueueConnection()**

- **QueueConnectionFactory.createQueueConnection(String, String)**
- **TopicConnectionFactory.createTopicConnection()**
- **TopicConnectionFactory.createTopicConnection(String, String)**

WebSphere Application Server 連線管理程式會嘗試從此 Factory 的可用儲存區中取得連線，並將其傳回應用程式。

如果儲存區中沒有可用連線，且從此 Factory 建立的連線數目尚未達到該 Factory 的 *maximum connections* 內容中所指定的限制，則連線管理程式會建立新連線以供應用程式使用。

但是，如果應用程式嘗試建立連線，但從此 Factory 建立的連線數目已經等於該 Factory 的 *maximum connections* 內容值，則應用程式會等待連線變成可用（將要放回可用儲存區）。

應用程式等待的時間是在連線儲存區的 *connection timeout* 內容中指定，其預設值為 180 秒。 如果在此 180 秒期間內將連線放回可用儲存區，則連線管理程式會立即將其再次從儲存區中取出並傳遞給應用程式。 但是，如果逾時期間過去，則會擲出 *ConnectionWaitTimeoutException*。

在應用程式完成連線並透過呼叫下列方法來關閉連線時：

- **Connection.close()**
- **QueueConnection.close()**
- **TopicConnection.close()**

此連線實際上會保持開啟，並且會傳回可用儲存區以便其他應用程式可以重複使用。 因此，您可以在 WebSphere Application Server 與 JMS 提供者之間開啟連線，即使應用程式伺服器上未執行任何 JMS 應用程式。

進階連線儲存區內容
許多進階內容可用來控制 JMS 連線儲存區的行為。

突波保護

第 259 頁的『執行出埠傳訊的應用程式如何使用連線儲存區』將說明如何使用併入 `connectionFactory.createConnection()` 的 `sendMessage()` 方法。

請考量如下狀況：您擁有全部是從相同 Connection Factory 建立 JMS 連線的 50 個 EJB 作為其 `ejbCreate()` 方法的一部分。

如果所有這些 Bean 都是同時建立，且 Factory 的可用連線儲存區中沒有任何連線，則應用程式伺服器會嘗試同時建立 50 條 JMS 與相同 JMS 提供者的連線。 其結果會對 WebSphere Application Server 及 JMS 提供者造成重大負載。

突波保護內容可以限制在任何時刻可從 Connection Factory 建立的 JMS 連線數目，並錯開建立其他連線，從而防止發生此狀況。

使用下列兩個內容，即可限制在任何時刻建立的 JMS 連線數目：

- Surge threshold
- Surge creation interval。

在 EJB 應用程式嘗試從 Connection Factory 建立 JMS 連線時，連線管理程式會檢查要建立的連線數目。 如果該數目小於或等於 `surge threshold` 內容的值，連線管理程式會繼續開啟新的連線。

不過，如果要建立的連線數目超出 `surge threshold` 內容，則在建立及開啟新連線之前，連線管理程式會等待 `surge creation interval` 內容指定的時段。

停留連線

如果 JMS 應用程式使用該連線將要求傳送至 JMS 提供者，且提供者未在特定時間量內回應，則會將 JMS 連線視為 `stuck`。

WebSphere Application Server 可提供偵測 `stuck` JMS 連線的方法。若要使用此功能，您必須設定下列三個內容：

- Stuck Time Timer
- Stuck Time
- Stuck Threshold

第 261 頁的『儲存區維護執行緒範例』會說明儲存區維護執行緒如何定期執行，以及如何檢查 Connection Factory 的可用儲存區內容，以尋找在某段時間內未用的連線或已經存在太久的連線。

為了偵測停留連線，應用程式伺服器還會管理停留連線執行緒，用於檢查從 Connection Factory 建立的所有作用中連線狀態，以瞭解是否有任何連線正在等待 JMS 提供者的回覆。

停留連線執行緒的執行時間是由 `Stuck time timer` 內容決定。 此內容的預設值是零，表示絕不執行停留連線偵測。

如果執行緒找到正在等待回應的連線，則會判斷它已經等待的時間，並將此時間與 `Stuck time` 內容值進行比較。

如果 JMS 提供者進行回應所花費的時間超出 `Stuck time` 內容指定的時間，應用程式伺服器即會將此 JMS 連線標示為 stuck。

例如，假設 Connection Factory `jms/CF1` 將 `Stuck time timer` 內容設為 10，並將 `Stuck time` 內容設為 15。

停留連線執行緒每隔 10 秒便會變成作用中狀態，並且會檢查從 `jms/CF1` 建立的任何連線，是否已等待 IBM MQ 的回應超過 15 秒。

假設 EJB 使用 `jms/CF1` 來建立與 IBM MQ 的 JMS 連線，然後嘗試透過呼叫 `Connection.createSession()` 來建立使用該連線的 JMS 階段作業。

但是，某個情況將阻止 JMS 提供者回應該要求。 可能是機器已凍結，也可能是 JMS 提供者上執行的程序已死鎖，從而阻止處理任何新工作：

在 EJB 呼叫 `Connection.createSession()` 10 秒之後，停留連線計時器會變成作用中狀態，並檢查從 `jms/CF1` 建立的作用中連線。

假設只有一條作用中連線，例如，稱為 `c1`。 第一個 EJB 已等待 10 秒以接收對其向下傳送至 `c1` 的要求之回應，由於此時間小於 `Stuck time` 的值，因此停留連線計時器會忽略此連線並變成非作用中狀態。

10 秒之後，停留連線執行緒再次變成作用中狀態，並檢查 `jms/CF1` 的作用中連線。 如同之前一樣，假設只有 `c1` 一條連線。

自第一個 EJB 呼叫 `createSession()` 以來已有 20 秒，且 EJB 仍在等待回應。 20 秒比 `Stuck time` 內容中指定的時間長，因此停留連線執行緒會將 `c1` 標示為停留。

5 秒之後，如果 IBM MQ 最終做出回應，並容許第一個 EJB 建立 JMS 階段作業，則此連線會回到使用中狀態。

應用程式伺服器會計算從 Connection Factory 建立的停留 JMS 連線數目。 在應用程式使用該 Connection Factory 來建立新的 JMS 連線，且該 Factory 的可用儲存區中沒有任何可用連線時，連線管理程式即會將停留連線數目與 `Stuck threshold` 內容值進行比較。

如果停留連線數目小於為 `Stuck threshold` 內容設定的值，則連線管理程式會建立新的連線，並將它提供給應用程式。

不過，如果停留連線數目等於 `Stuck threshold` 內容的值，應用程式會發生資源異常狀況。

儲存區分割區

WebSphere Application Server 會提供下列兩個內容，讓您分割 Connection Factory 的可用連線儲存區：

- `Number of free pool partitions` 會告知應用程式伺服器需要將可用連線儲存區分為多少個分割區。
- `Free pool distribution table size` 會決定如何為分割區編製索引。

除非「IBM 支援中心」要求您進行變更，否則請將這些內容保留為其預設值零。

請注意，WebSphere Application Server 具有一個稱為 `Number of shared partitions` 的額外進階連線儲存區內容。 此內容會指定用於儲存共用連線的分割區數目。 但是，由於 JMS 連線一律不會共用，所以此內容並不適用。

使用連線儲存區的範例
訊息驅動 Bean 接聽器埠元件以及執行出埠傳訊的應用程式，會使用 JMS 連線儲存區。

第 258 頁的圖 43 顯示 WebSphere Application Server 7.5 版及 8.0 版的連線儲存區如何運作。

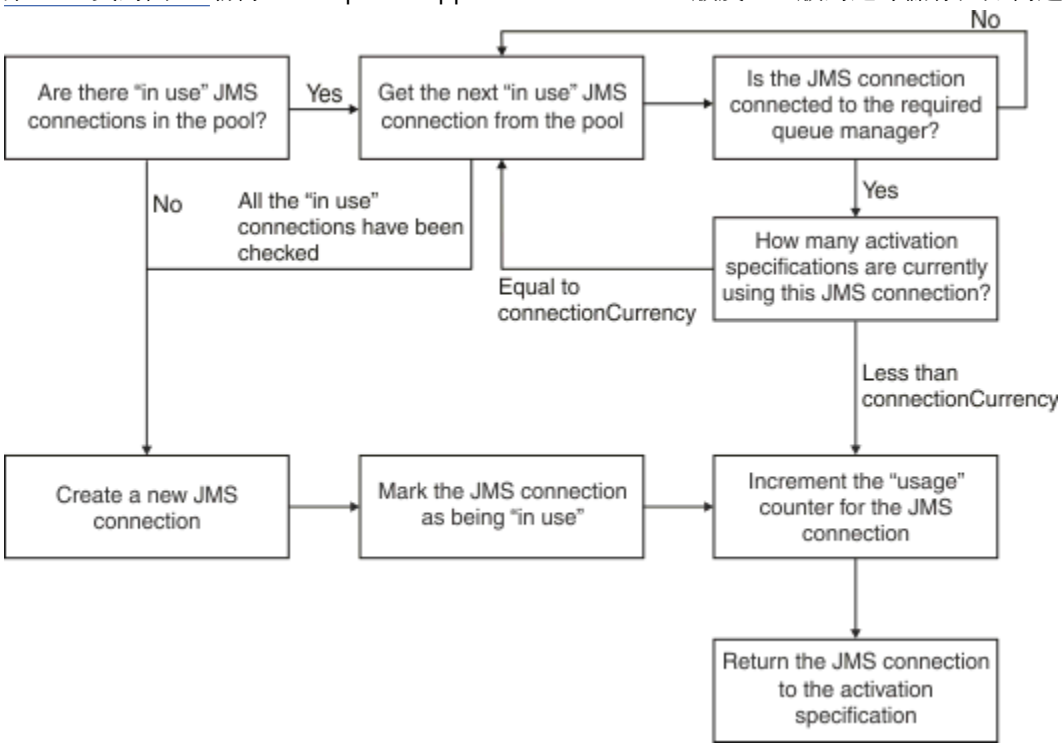


圖 *43: WebSphere Application Server 7.5* 版及 *8.0* 版 - 連線儲存區如何運作

第 258 頁的圖 44 顯示 WebSphere Application Server 8.5 版的連線儲存區如何運作。

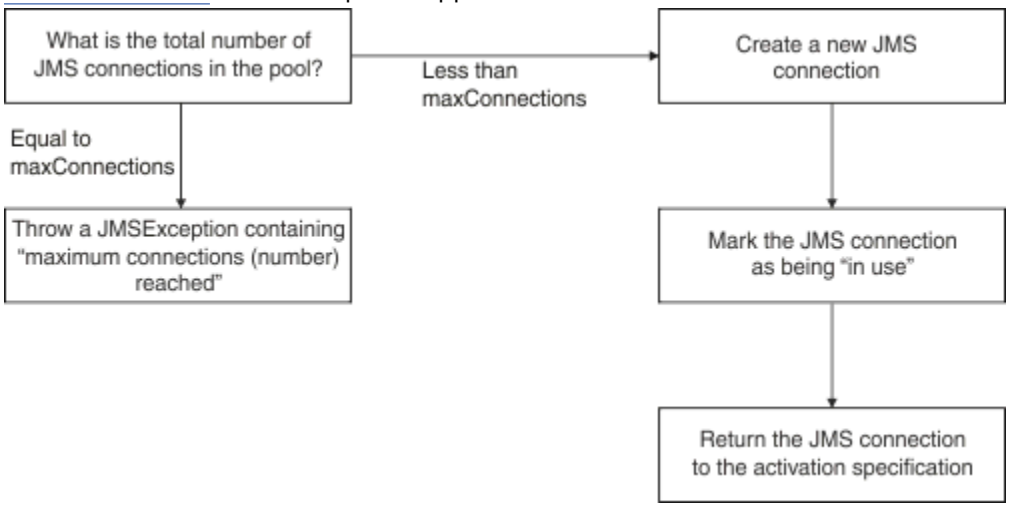


圖 *44: WebSphere Application Server 8.5* 版 - 連線儲存區如何運作

MDB 接聽器埠如何使用連線儲存區

假設您在使用 IBM MQ 作為 JMS 提供者的 WebSphere Application Server Network Deployment 系統上部署 MDB。 此 MDB 是針對使用某個 Connection Factory（例如，稱為 `jms/CF1`）的接聽器埠而部署，而該 Connection Factory 的 *maximum connections* 內容設定為 2，這表示在任何時刻皆只能從此 Factory 建立兩條連線。

當接聽器埠啟動時，該埠會嘗試使用 jms/CF1 Connection Factory 來建立與 IBM MQ 的連線。

為了這樣做，該埠會從連線管理程式要求連線。 因為這是第一次使用 jms/CF1 Connection Factory，所以 jms/CF1 可用連線儲存區中沒有任何連線，因此連線管理程式會建立一條新連線（例如，稱為 c1）。 請注意，此連線在接聽器埠的整個生命週期皆會存在。

現在，請考量您使用 WebSphere Application Server 管理主控台來停止接聽器埠的狀況。 在此情況下，連線管理程式會取得連線，並將其放回可用儲存區。 但是，與 IBM MQ 的連線仍會保持開啟。

如果您重新啟動接聽器埠，該埠會再次向管理程式要求取得與佇列管理程式的連線。 因為您在可用儲存區中現已擁有連線 (c1)，所以連線管理程式會從儲存區中取出此連線，並使其可供接聽器埠使用。

現在，假設您擁有已部署至應用程式伺服器的第二個 MDB，但它是使用不同的接聽器埠。

假設您隨後嘗試啟動第三個接聽器埠，該埠亦配置為使用 jms/CF1 Connection Factory。 第三個接聽器埠從連線管理程式要求連線，但連線管理程式查看 jms/CF1 的可用儲存區並發現它是空的。 然後，連線管理程式會檢查已從 jms/CF1 Factory 建立的連線數目。

因為 jms/CF1 的 maximum connections 內容設定為 2，且您已從此 Factory 建立兩條連線，所以連線管理程式會等待 180 秒（connection timeout 內容的預設值）以讓連線變成可用。

但是，如果您停止第一個接聽器埠，則其連線 c1 會放入 jms/CF1 的可用儲存區。 連線管理程式隨後會擷取此連線，並將其提供給第三個接聽器。

如果您現在嘗試重新啟動第一個接聽器，則此接聽器必須等待其他接聽器埠的其中一個接聽器埠停止，然後第一個接聽器才能重新啟動。 如果沒有任何執行中的接聽器埠在 180 秒內停止，則第一個接聽器會收到 ConnectionWaitTimeoutException 錯誤並停止。

執行出埠傳訊的應用程式如何使用連線儲存區

對於此選項，假設在應用程式伺服器中已安裝單一 EJB（例如，稱為 EJB1）。 此 Bean 會透過下列方式來實作稱為 sendMessage() 的方法:

- 使用 connectionFactory.createConnection()，從 Factory jms/CF1 建立與 IBM MQ 的 JMS 連線。
- 從此連線建立 JMS 階段作業。
- 從此階段作業建立訊息產生者。
- 傳送訊息。
- 關閉產生者。
- 關閉階段作業。
- 透過呼叫 connection.close() 來關閉連線。

假設 Factory jms/CF1 的可用儲存區是空的。 第一次呼叫 EJB 時， Bean 會嘗試從 Factory jms/CF1 建立與 IBM MQ 的連線。 因為此 Factory 的可用儲存區是空的，所以連線管理程式會建立新連線，然後將其提供給 EJB1。

在此方法剛要結束之前，它會呼叫 connection.close()。 但連線管理程式不是關閉 c1，而是會取得此連線並將其放回 jms/CF1 可用儲存區。

下次呼叫 sendMessage() 時， connectionFactory.createConnection() 方法會將 c1 傳回給應用程式。

假設您有第二個 EJB 實例與第一個實例同時執行。 在這兩個實例呼叫 sendMessage() 時，會從 jms/CF1 Connection Factory 建立兩條連線。

現在假設已建立第三個 Bean 實例。 在第三個 Bean 呼叫 sendMessage() 時，此方法會呼叫 connectionFactory.createConnection() 以從 jms/CF1 建立連線。

但是，目前已從 jms/CF1 建立兩條連線，該數目等於此 Factory 的 maximum connections 值。 因此，createConnection() 方法會等待 180 秒 (連線逾時內容的預設值)，讓連線變成可用。

但是，如果第一個 EJB 的 sendMessage() 方法呼叫 connection.close() 並結束，則它所使用的連線 c1 會放回可用連線儲存區。 連線管理程式會從可用儲存區中取出此連線，並將其提供給第三個 EJB。 從該

Bean 至 connectionFactory.createConnection() 的呼叫隨後即會傳回，以讓 sendMessage() 方法完成。

使用相同連線儲存區的 MDB 接聽器埠及 EJB

上述兩個範例顯示接聽器埠及 EJB 如何才能獨立地使用連線儲存區。 但是，您可以讓接聽器埠及 EJB 在相同應用程式伺服器內執行，並使用相同 Connection Factory 來建立 JMS 連線。

您需要考量此狀況的影響

要切記的關鍵事項是將會在接聽器埠與 EJB 之間共用 Connection Factory。

例如，假設您擁有同時執行的接聽器及 EJB。 兩者皆使用 jms/CF1 Connection Factory，這表示已達到該 Factory 的 maximum connections 內容所指定的連線限制。

如果您嘗試啟動另一個接聽器埠或另一個 EJB 實例，則兩者皆必須等待連線傳回 jms/CF1 的可用連線儲存區。

可用連線儲存區維護執行緒
與每個可用連線儲存區相關聯的是儲存區維護執行緒，此執行緒會監視可用儲存區以確定其中的連線仍然有效。

如果儲存區維護執行緒決定需要捨棄可用儲存區中的連線，則執行緒會實際關閉 IBM MQ 的 JMS 連線。

儲存區維護執行緒如何運作

儲存區維護執行緒的行為，是由連線儲存區的下列四個內容的值決定：

Aged timeout
連線保持開啟的時間量。

Minimum connections
連線管理程式在 Connection Factory 的可用儲存區中保留的連線數目下限。

Reap time
儲存區維護執行緒的執行頻率。

Unused timeout
連線在關閉之前保留在可用儲存區中的時間。

依預設，儲存區維護的執行緒會每隔 180 秒執行一次，但設定連線儲存區的 **Reap time** 內容即可變更此值。

維護執行緒會檢查儲存區中的每條連線、檢查它已處於儲存區中的時間，以及自它建立和前次使用它以來的經歷時間。

如果連線未使用的時間超過連線儲存區的 **Unused timeout** 內容值，維護執行緒會檢查目前在可用儲存區中的連線數目。 如果該數目：

- 大於 **Minimum connections** 的值，連線管理程式即會關閉該連線。
- 等於 **Minimum connections** 的值，該連線不會關閉且會保留在可用儲存區中。

Minimum connections 內容的預設值為 *1*，這表示為了效能原因，連線管理程式一律會嘗試在可用儲存區中至少保留一條連線。

Unused timeout 內容的預設值為 1800 秒。 依預設，如果連線放回可用儲存區且再次未使用的時間長達至少 1800 秒，則只要關閉該連線後可用儲存區中至少保留一條連線，即會關閉該連線。

此程序可防止未用的連線過時。 若要關閉此特性，請將 **Unused timeout** 內容設定為零。

如果連線位於可用儲存區中，且自其建立以來的經歷時間大於連線儲存區的 **Aged timeout** 內容值，則無論自前次使用它以來有多久，皆會關閉該連線。

依預設，**Aged timeout** 內容會設定為零，這表示維護執行緒絕不執行此檢查。 無論可用儲存區中將會保留多少連線，皆會捨棄存在時間長於 **Aged timeout** 內容的連線。 請注意，**Minimum connections** 內容在此狀況下不起作用。

停用儲存區維護執行緒

從上述說明中，您可以看出儲存區維護執行緒在處於作用中時會執行大量工作，尤其是在 Connection Factory 的可用儲存區中有大量連線時。

例如，假設有三個 JMS Connection Factory，每個 Factory 的 **Maximum connections** 內容皆設定為 10。每隔 180 秒，三個儲存區維護執行緒便會變成作用中狀態，並分別掃描每個 Connection Factory 的可用儲存區。 如果可用儲存區具有許多連線，維護執行緒便會有許多工作要執行，這可能會嚴重影響效能。

將個別可用連線儲存區的 **Reap time** 內容設定為零，即可為其停用儲存區維護執行緒。

停用維護執行緒即表示絕不會關閉連線，即使 **Unused timeout** 時間已過亦如此。 但是，如果 **Aged timeout** 時間已過，則仍可能會關閉連線。

當應用程式完成連線時，連線管理程式會檢查連線存在的時間長度，如果該期間超過 **Aged timeout** 內容的值，則連線管理程式會關閉連線，而不是將它傳回至可用儲存區。

Aged timeout 的交易式影響

如上一節中所述，**Aged timeout** 內容會指定在連線管理程式關閉與 JMS 提供者的連線之前，該連線保持開啟的時間。

Aged timeout 內容的預設值為零，這表示絕不會因連線太舊而關閉它。 您應該將 **Aged timeout** 內容保留為此值，因為在 EJB 內使用 JMS 時，啟用 **Aged timeout** 可能會有交易式含意。

在 JMS 中，交易單元是從 JMS 連線 建立的 JMS 階段作業。 參與交易的是 JMS 階段作業，而不是 JMS 連線。

由於應用程式伺服器的設計， JMS 連線可以關閉，因為 **Aged timeout** 已經過，即使交易中涉及從該連線建立的 JMS 階段作業也一樣。

如 JMS 規格中所述，關閉 JMS 連線會導致回復 JMS 階段作業中所有未執行的交易式工作。 但是，應用程式伺服器並不知道從該連線建立的 JMS 階段作業不再有效。 在伺服器嘗試使用此階段作業來確定或回復交易時，將會出現 `IllegalStateException`。

重要: 如果要在 EJB 內將 **Aged timeout** 與 JMS 連線搭配使用，在執行 JMS 作業的 EJB 方法結束之前，請確保在 JMS 階段作業中明確地確定所有 JMS 工作。

儲存區維護執行緒範例

使用 Enterprise JavaBean (EJB) 範例來瞭解儲存區維護執行緒的運作方式。 請注意，您亦可使用「訊息驅動 Bean (MDB)」及接聽器埠，只要能夠取得可用儲存區中的連線即可。

如需 `sendMessage()` 方法的進一步詳細資料，請參閱 第 259 頁的『執行出埠傳訊的應用程式如何使用連線儲存區』。

您已使用下列值來配置 Connection Factory：

- **Reap time** 設定為其預設值（180 秒）
- **Aged timeout** 設定為其預設值（零秒）
- **Unused timeout** 設為 300 秒

在應用程式伺服器啟動之後，即會呼叫 `sendMessage()` 方法。

此方法會使用 Factory `jms/CF1` 來建立連線（例如，稱為 `c1`），並使用該 Factory 來傳送訊息，然後呼叫 `connection.close()`（這會導致將 `c1` 放回可用儲存區）。

在 180 秒之後，儲存區維護執行緒會啟動，並檢查 `jms/CF1` 可用連線儲存區。 由於此儲存區中存在可用連線 `c1`，因此維護執行緒會檢查放回該連線的時間，並將此時間與現行時間進行比較。

自將連線放入可用儲存區以來已過 180 秒，但此值小於 `jms/CF1` 的 **Unused timeout** 內容值。 因此，維護執行緒不會對此連線進行任何處理。

180 秒之後，儲存區維護執行緒會再次執行。 維護執行緒找到連線 `c1`，並判斷此連線已處於儲存區中 360 秒，而這長於所設定的 **Unused timeout** 值，因此連線管理程式會關閉此連線。

如果您現在再次執行 sendMessage() 方法，則在應用程式呼叫 connectionFactory.createConnection() 時，連線管理程式會建立與 IBM MQ 的新連線，因為 Connection Factory 的可用連線儲存區是空的。

前述範例顯示在 **Aged timeout** 內容設定為零時，維護執行緒如何使用 **Reap time** 及 **Unused timeout** 內容來防止過時的連線。

Aged timeout 內容如何運作?

在下列範例中，假設您已將:

- **Aged timeout** 內容設定為 300 秒
- **Unused timeout** 內容設定為零。

您呼叫 sendMessage() 方法，此方法嘗試從 jms/CF1 Connection Factory 建立連線。

因為此 Factory 的可用儲存區是空的，所以連線管理程式會建立新連線 c1，然後將其傳回應用程式。 在 sendMessage() 呼叫 connection.close() 時，會將 c1 放回可用連線儲存區。

180 秒之後，儲存區維護執行緒即會執行。 執行緒在可用連線儲存區中找到 c1，並檢查它是多久以前建立的。 此連線已存在 180 秒，但此值小於 **Aged timeout**，因此儲存區維護執行緒不會對此連線進行任何處理，並且會回到休眠狀態。

60 秒之後，再次呼叫 sendMessage()。 在此方法呼叫 connectionFactory.createConnection() 時，這次連線管理程式探索到 jms/CF1 的可用儲存區中存在連線 c1。 連線管理程式會從可用儲存區中取出 c1，並將該連線提供給應用程式。

在 sendMessage() 結束時，此連線即會傳回可用儲存區。 120 秒之後，儲存區維護執行緒再次起動、掃描 jms/CF1 的可用儲存區內容並探索到 c1。

雖然僅在 120 秒以前使用此連線，但儲存區維護執行緒仍會關閉此連線，因為此連線已存在總計 360 秒，而這長於您為 **Aged timeout** 內容設定的值（300 秒）。

Minimum connections 內容如何影響儲存區維護執行緒

再次使用 第 258 頁的『MDB 接聽器埠如何使用連線儲存區』範例，並假設您在應用程式伺服器中已部署兩個 MDB，且各自使用不同的接聽器埠。

每個接聽器埠皆配置為使用您已按下列方式配置的 jms/CF1 Connection Factory:

- **Unused timeout** 內容設定為 120 秒
- **Reap time** 內容設定為 180 秒
- **Minimum connections** 內容設定為 1

假設第一個接聽器已停止，且其連線 c1 已放回可用儲存區。 180 秒之後，儲存區維護執行緒會起動、掃描 jms/CF1 的可用儲存區內容，並探索到 c1 處於可用儲存區的時間長於 Connection Factory 的 **Unused timeout** 內容值。

但是，在關閉 c1 之前，儲存區維護執行緒會檢查在捨棄此連線時儲存區中將會保留的連線數目。 由於 c1 是可用連線儲存區中的唯一連線，因此連線管理程式不會關閉它，因為這樣做會使可用儲存區中保留的連線數目小於為 **Minimum connections** 設定的值。

現在，假設第二個接聽器已停止。 可用連線儲存區現在包含兩條可用連線 - c1 及 c2。

180 秒之後，儲存區維護執行緒會再次執行。 此時，c1 已處於可用連線儲存區 360 秒，c2 已處於可用連線儲存區 180 秒。

儲存區維護執行緒會檢查 c1，並探索到它處於儲存區的時間長於 **Unused timeout** 內容值。

然後，執行緒會檢查可用儲存區中的連線數目，並將此值與 **Minimum connections** 內容值進行比較。 因為此儲存區包含兩條連線，且 **Minimum connections** 設定為 1，所以連線管理程式會關閉 c1。

現在，維護執行緒會檢查 c2。 這在可用連線儲存區中的時間也超過 **Unused timeout** 內容的值。 不過，因為關閉 c2 會讓可用連線儲存區保留小於其中設定的連線數下限，所以連線管理程式會單獨離開 c2 。

JMS 連線及 *IBM MQ*
使用 IBM MQ 作為 JMS 提供者的相關資訊。

使用連結傳輸

如果 Connection Factory 已配置成使用連結傳輸，則每一個 JMS 連線都會與 IBM MQ 建立交談 (也稱為 **hconn**)。 此交談會使用交互程序通訊（或共用記憶體）與佇列管理程式進行通訊。

使用用戶端傳輸

當 IBM MQ 傳訊提供者 Connection Factory 已配置成使用用戶端傳輸時，從該 Factory 建立的每一個連線都會建立與 IBM MQ 的新交談 (也稱為 **hconn**)。

對於使用 IBM MQ 傳訊提供者標準模式連接至佇列管理程式的 Connection Factory，從 Connection Factory 建立的多條 JMS 連線，可以共用與 IBM MQ 的 TCP/IP 連線。 如需相關資訊，請參閱 第 265 頁的『在 IBM MQ classes for JMS 中共用 TCP/IP 連線』。

若要判斷 JMS 連線在任何時刻使用的用戶端通道數目上限，請將指向相同佇列管理程式的所有 Connection Factory 的 *Maximum connections* 內容值相加。

例如，假設您擁有 `jms/CF1` 及 `jms/CF2` 兩個 Connection Factory，它們皆已配置為使用相同 IBM MQ 通道連接至相同 IBM MQ 佇列管理程式。

這些 Factory 使用的是預設連線儲存區內容，這表示 *Maximum connections* 設定為 10。 如果同時使用來自 jms/CF1 及 jms/CF2 的所有連線，則在應用程式伺服器與 IBM MQ 之間將會存在 20 個交談。

如果 Connection Factory 使用 IBM MQ 傳訊提供者標準模式連接至佇列管理程式，則在應用程式伺服器與佇列管理程式之間可以存在的這些 Connection Factory 的 TCP/IP 連線數目上限為:

```
20/the value of SHARECNV for the IBM MQ channel
```

如果已將 Connection Factory 配置為使用 IBM MQ 傳訊提供者移轉模式，則應用程式伺服器與 IBM MQ 之間的這些 Connection Factory 的 TCP/IP 連線數目上限將為 20（兩個 Factory 的連線儲存區中每條 JMS 連線各一條）。

相關概念
第 68 頁的『使用 IBM MQ classes for JMS/Jakarta Messaging』
IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 是 IBM MQ 隨附的 Java 傳訊提供者。 除了實作 JMS 和 Jakarta Messaging 規格中定義的介面，這些傳訊提供者也會將兩組延伸新增至 Java 傳訊 API。

Java SE 環境中的物件儲存區作業
使用 Java SE (或使用另一個架構，例如 Spring)，程式設計模型非常靈活。 因此，單一合併策略並不適合所有方案。 您應該考量是否有架構可執行任何形式的儲存區作業，例如 Spring。

否則，應用程式邏輯可能會處理此問題。 問自己應用程式本身有多複雜? 最好瞭解應用程式，以及它從傳訊系統的連線功能所需要的內容。 應用程式通常也會在其自己的封套程式碼內撰寫基本 JMS API。

雖然這可以是一個非常明智的方法，並且可以隱藏複雜性，但值得注意的是它會帶來問題。 例如，經常呼叫的一般 `getMessage()` 方法不應只開啟和關閉消費者。

您應該考量的點:

- 應用程式需要存取 IBM MQ 多久? 一直或只是偶爾
- 多久傳送一次訊息? 頻率越低，與 IBM MQ 的單一連線可以共用的越多。
- 連線中斷異常狀況通常表示需要重建儲存連線。 關於:
  - 安全異常狀況或主機無法使用
  - 佇列已滿異常狀況
- 如果發生連線中斷異常狀況，儲存區中的其他可用連線應該怎麼辦? 是否應該關閉並重建它們?
- 例如，如果正在使用 TLS，您希望單一連線保持開啟多久?

- 聯合排存的連線如何識別本身，以便佇列管理程式管理者可以找出連線並加以追蹤。

您應該考量將所有 JMS 物件用於儲存區作業，並在可能時將該物件儲存起來。 物件包括:

- JMS 連線
- Session
- 環境定義
- 所有不同類型的生產者和消費者

使用用戶端傳輸時， JMS 連線、階段作業及環境定義會在與 IBM MQ 佇列管理程式通訊時使用 Socket。 透過儲存這些物件，可節省到佇列管理程式的送入 IBM MQ 連線數 (hConns)，並減少通道實例數。

使用佇列管理程式的連結傳輸會完全移除網路層。 不過，許多應用程式會使用用戶端傳輸來提供更高可用性及工作量平衡的配置。

JMS 生產者和消費者會開啟佇列管理程式上的目的地。 如果開啟的佇列或主題數目較少，且應用程式的多個組件正在使用這些物件，則將這些物件合併起來可能很有用。

從 IBM MQ 角度來看，此處理程序會儲存一系列 MQOPEN 和 MQCLOSE 作業。

連線、階段作業及環境定義

這些物件都封裝佇列管理程式的 IBM MQ 連線控點，並從 ConnectionFactory 產生。 您可以將邏輯新增至應用程式，以將從單一 Connection Factory 建立的連線數及其他物件數限制為特定數目。

您可以在應用程式中使用簡式資料結構來包含所建立的連線。 需要使用其中一個資料結構的應用程式碼可以移出 要使用的物件。

請考慮下列因素:

- 何時應該從儲存區中移除連線? 通常，在連線上建立異常狀況接聽器。 當呼叫該接聽器來處理異常狀況時，您應該重建連線，以及從該連線建立的任何階段作業。
- 如果 CCDT 是用於工作量平衡，則連線可以連接至不同的佇列管理程式。 這可能適用於儲存區需求。

請記住， JMS 規格指出多個執行緒同時存取階段作業或環境定義是程式設計錯誤。 IBM MQ JMS 程式碼會嘗試嚴格處理執行緒。 不過，您應該將邏輯新增至應用程式，以確保一次只有一個執行緒使用階段作業或環境定義物件。

生產者和消費者

所建立的每一個生產者和消費者都會在佇列管理程式上開啟一個目的地。 如果將相同的目的地用於各種作業，則保持消費者或生產者物件開啟是有意義的。 只有在完成所有工作時，才關閉物件。

雖然開啟和關閉目的地是短暫的作業，但如果經常這樣做，所花費的時間可能會加起來。

這些物件的範圍是在其建立來源的階段作業或環境定義內，因此它們需要保留在該範圍內。 一般來說，應用程式的撰寫方式是這樣，這樣做是相當直接的。

監視

應用程式如何監視其物件儲存區? 這個問題的答案很大程度上取決於所實施的匯集解決方案的複雜性。

如果您考量 JavaEE 儲存區作業實作，則有大量選項，包括:

- 儲存區的現行大小
- 物件在其中花費的時間
- 清理儲存區
- 重新整理連線

您也應該考量單一重複使用的階段作業如何出現在佇列管理程式上。 有一些 Connection Factory 內容可識別可能有用的應用程式 (例如 appName)。

第 68 頁的『使用 IBM MQ classes for JMS/Jakarta Messaging』

IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 是 IBM MQ 隨附的 Java 傳訊提供者。除了實作 JMS 和 Jakarta Messaging 規格中定義的介面，這些傳訊提供者也會將兩組延伸新增至 Java 傳訊 API。

在 *IBM MQ classes for JMS* 中共用 *TCP/IP* 連線
可以建立 MQI 通道的多個實例，以共用單一 TCP/IP 連線。

在相同 Java 執行時期環境內執行的應用程式，以及使用 IBM MQ classes for JMS 或 IBM MQ 資源配接器使用 CLIENT 傳輸來連接至佇列管理程式的應用程式，可以用來共用通道實例。

如果通道定義時 **SHARECNV** 參數設為大於 1 的值，則該交談數可以共用通道實例。 如果要啟用 Connection Factory 或啟動規格來使用這個函數，請將 **SHARECONVALLOWED** 內容設為 YES。

JMS 應用程式所建立的每一個 JMS 連線和 JMS 階段作業，都會建立自己與佇列管理程式的交談。

當啟動規格啟動時， IBM MQ 資源配接器會啟動與佇列管理程式的交談，以供啟動規格使用。 伺服器階段作業儲存區中與啟動規格相關聯的每個伺服器階段作業也會啟動與佇列管理程式的交談。

SHARECNV 屬性是連線共用的最佳努力方法。 因此，當 **SHARECNV** 值大於 0 與 IBM MQ classes for JMS 搭配使用時，不保證新的連線要求一律會共用已建立的連線。

如何共用 TCP/IP 連線

有兩種策略可用於共用 TCP/IP 連線:

GLOBAL 策略
此策略是共用 TCP/IP 連線的預設策略。 任何 JMS 連線或階段作業都可以在任何適當的 TCP/IP 連線上使用交談。 適用性取決於主機位址、埠號、使用者 ID 和密碼，以及 TLS/SSL 參數等因素。

這種共用 TCP/IP 連線的方法會將使用中的通道實例數減至最少，但代價是競用 TCP/IP 連線的廣域儲存區存取權。

CONNECTION 策略
使用此策略，通道實例只會在相關 JMS 物件之間共用。 具體而言，當建立 JMS 連線時，會為其建立通道實例，且該通道實例上的其他交談僅適用於該 JMS 連線所建立的 JMS 階段作業。

如果建立的交談數超過 SHARECNV 屬性指定的數目，則會建立新的通道實例，該實例只能由原始 JMS 連線所建立的 JMS 階段作業使用。

這種共用通道實例的方法可減少交談的競用，但代價是可能需要大幅增加通道實例。

明確指定通道實例共用策略

V 9.3.2

依預設，如果應用程式無法重新連接，則會使用 GLOBAL 策略。 可重新連接的應用程式一律使用 CONNECTION 策略。

對於使用 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 的應用程式，可以針對整個應用程式的不可重新連接應用程式啟用 CONNECTION 策略。 您可以透過將系統內容 `com.ibm.mq.jms.channel.sharing` 設為值 `CONNECTION` 來啟用 CONNECTION 策略。 此值不區分大小寫，且會忽略 `CONNECTION` 以外的任何值。

您可以使用下列其中一種方式來設定系統內容 `com.ibm.mq.jms.channel.sharing`:

- 使用 "-D" 指令行選項，將內容設為 JVM 起始設定的一部分:

```
-Dcom.ibm.mq.jms.channel.sharing=CONNECTION
```

- 使用 `System.setProperty()` 在任何使用 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 之前設定內容

計算 GLOBAL 共用策略的通道實例數

請使用下列公式來判定應用程式所建立的通道實例數目上限:

啟動規格

通道實例數 = (*maxPoolDepth_value* + 1)/ *SHARECNV_value*

其中 *maxPoolDepth_value* 是 **maxPoolDepth** 內容的值，而 *SHARECNV_value* 是啟動規格所使用通道上 **SHARECNV** 內容的值。

其他 JMS 應用程式

通道實例數 = (*jms_connections* + *jms_sessions*)/ *SHARECNV_value*

其中 *jms_connections* 是應用程式所建立的連線數， *jms_sessions* 是應用程式所建立的 JMS 階段作業數，而 *SHARECNV_value* 是啟動規格所使用通道上 **SHARECNV** 內容的值。

計算 CONNECTION 共用策略的通道實例數

通道實例數取決於應用程式中 JMS 連線之間的 JMS 階段作業分佈。

容許 JMS 連線的一個交談，以及該 JMS 連線下每一個 JMS 階段作業的一個交談，然後除以 **SHARECNV** 值 (四捨五入)。 此計算會提供該 JMS 連線所需的通道實例。

相同的原則可以套用至啟動規格。 將啟動規格視為 JMS 連線，並將 **maxPoolDepth** 內容視為 JMS 階段作業數目。

範例

下列範例顯示如何使用公式來計算應用程式使用 IBM MQ classes for JMS 或 IBM MQ 資源配接器在佇列管理程式上建立的通道實例數。

JMS 應用程式範例

JMS 應用程式連線會使用 CLIENT 傳輸連接至佇列管理程式，並建立 JMS 連線及三個 JMS 階段作業。 應用程式用來連接佇列管理程式的通道將 **SHARECNV** 內容設為值 10。 當應用程式執行時，應用程式與佇列管理程式之間會有四個交談，以及一個通道實例。 這四個交談都共用通道實例。

啟動規格範例

啟動規格會使用 CLIENT 傳輸來連接至佇列管理程式。 啟動規格是配置成將 **maxPoolDepth** 內容設為 10。 啟動規格配置成使用的通道將 **SHARECNV** 內容設為 10。 當啟動規格同時執行並處理 10 則訊息時，啟動規格與佇列管理程式之間的交談數為 11 (伺服器階段作業為 10 個交談，啟動規格為 1 個交談)。 啟動規格所使用的通道實例數為 2。

啟動規格範例

啟動規格會使用 CLIENT 傳輸來連接至佇列管理程式。 啟動規格是配置成將 **maxPoolDepth** 內容設為 5。 啟動規格配置成使用的通道將 **SHARECNV** 內容設為 0。 當啟動規格正在執行且同時處理 5 則訊息時，啟動規格與佇列管理程式之間的交談數為 6 (伺服器階段作業為 5 個交談，啟動規格為 1 個交談)。 啟動規格所使用的通道實例數是 6 ，因為通道上的 **SHARECNV** 內容設為 0 ， 每一次交談都會使用自己的通道實例。

相關工作

第 417 頁的『決定從 WebSphere Application Server 到 IBM MQ 所建立的 TCP/IP 連線數』
使用共用交談特性，多個交談可以共用 MQI 通道實例，這也稱為 TCP/IP 連線。

在 *IBM MQ classes for JMS* 中指定用戶端連線的埠範圍
使用 LOCALADDRESS 內容來指定應用程式可以連結的埠範圍。

當 IBM MQ classes for JMS 應用程式嘗試以用戶端模式連接至 IBM MQ 佇列管理程式時，防火牆可能只容許源自指定埠或埠範圍的那些連線。 在此情況下，您可以使用 ConnectionFactory、QueueConnectionFactory 或 TopicConnectionFactory 物件的 LOCALADDRESS 內容來指定應用程式可連結的埠或埠範圍。

您可以使用 IBM MQ JMS 管理工具，或在 JMS 應用程式中呼叫 setLocalAddress () 方法，來設定 LOCALADDRESS 內容。 以下是從應用程式內設定內容的範例:

```
mqConnectionFactory.setLocalAddress("192.0.2.0(2000,3000)");
```

當應用程式隨後連接至佇列管理程式時，應用程式會連結至 192.0.2.0(2000) 至 192.0.2.0(3000) 範圍內的本端 IP 位址及埠號。

在具有多個網路介面的系統中，您也可以使用 LOCALADDRESS 內容來指定必須用於連線的網路介面。

對於與分配管理系統的即時連線，只有在使用多重播送時， LOCALADDRESS 內容才相關。 在此情況下，您可以使用內容來指定哪些本端網路介面必須用於連線，但內容的值不得包含埠號或埠號範圍。

如果您限制埠範圍，則可能會發生連線錯誤。 如果發生錯誤，則會擲出包含內嵌 MQException 的 JMSException ，其中包含 IBM MQ 原因碼 MQRC_Q_MGR_NOT_AVAILABLE 及下列訊息:

```
由於 LOCAL_ADDRESS_PROPERTY 限制，已拒絕 Socket 連線嘗試
```

如果指定範圍內的所有埠都在使用中，或指定的 IP 位址、主機名稱或埠號無效 (例如，負數埠號) ，則可能會發生錯誤。

因為 IBM MQ classes for JMS 可能建立應用程式所需要以外的連線，所以請一律考量指定埠範圍。 一般而言，應用程式所建立的每個階段作業都需要一個埠，而 IBM MQ classes for JMS 可能需要三個或四個額外埠。 如果發生連線錯誤，請增加埠範圍。

依預設在 IBM MQ classes for JMS 中使用的連線儲存區可能會影響埠可重複使用的速度。 因此，在釋放埠時可能會發生連線錯誤。

IBM MQ classes for JMS 中的通道壓縮

IBM MQ classes for JMS 應用程式可以使用 IBM MQ 機能來壓縮訊息標頭或資料。

壓縮在 IBM MQ 通道上流動的資料可以改善通道效能並減少網路資料流量。 使用 IBM MQ 提供的功能，您可以壓縮在訊息通道及 MQI 通道上流動的資料。 在任一類型的通道上，您可以彼此獨立壓縮標頭資料和訊息資料。 依預設，通道上不會壓縮任何資料。

IBM MQ classes for JMS 應用程式透過建立 java.util.Collection 物件，指定可用於壓縮連線上的標頭或訊息資料的技術。 每一個壓縮技術都是集合中的「整數」物件，而應用程式將壓縮技術新增至集合的順序，是當應用程式建立連線時，與佇列管理程式協議壓縮技術的順序。 然後，應用程式可以呼叫 setHdrCompList() 方法 (若為標頭資料) 或 setMsgCompList() 方法 (若為訊息資料) ，將集合傳遞至 ConnectionFactory 物件。 當應用程式備妥時，它可以建立連線。

下列程式碼片段說明所述的方法。 第一個程式碼片段顯示如何實作標頭資料壓縮:

```
Collection headerComp = new Vector();
headerComp.add(new Integer(WMQConstants.WMQ_COMPHDR_SYSTEM));
.
.
.
((MQConnectionFactory) cf).setHdrCompList(headerComp);
.
.
.
connection = cf.createConnection();
```

第二個程式碼片段顯示如何實作訊息資料壓縮:

```
Collection msgComp = new Vector();
msgComp.add(new Integer(WMQConstants.WMQ_COMPMSG_RLE));
msgComp.add(new Integer(WMQConstants.WMQ_COMPMSG_ZLIBHIGH));
.
.
.
((MQConnectionFactory) cf).setMsgCompList(msgComp);
.
.
.
connection = cf.createConnection();
```

在第二個範例中，當建立連線時，會依序 RLE 和 ZLIBHIGH 協議壓縮技術。 在 Connection 物件的生命期限期間，無法變更選取的壓縮技術。 若要在連線上使用壓縮，必須在建立 Connection 物件之前呼叫 setHdrCompList() 和 setMsgCompList() 方法。

在 *IBM MQ classes for JMS* 中非同步放置訊息

通常，當應用程式將訊息傳送至目的地時，應用程式必須等待佇列管理程式確認它已處理要求。 在某些情況下，您可以選擇以非同步方式放置訊息，以增進傳訊效能。 當應用程式非同步放置訊息時，佇列管理程式不會傳回每一個呼叫的成功或失敗，但您可以改為定期檢查錯誤。

目的地是否將控制權交還給應用程式，而不判斷佇列管理程式是否已安全接收訊息，取決於下列內容:

JMS 目的地內容 PUTASYNCALLOWED (簡稱-PAALD)。
PUTASYNCALLOWED 控制如果基礎佇列或 JMS 目的地代表的主題容許此選項， JMS 應用程式是否可以非同步放置訊息。

IBM MQ 佇列或主題內容 DEFPRESP (預設放置回應類型)。
DEFPRESP 指定將訊息放入佇列或將訊息發佈至主題的應用程式是否可以使用非同步放置功能。

下表顯示 PUTASYNCALLOWED 和 DEFPRESP 內容的可能值，以及用來啟用非同步放置功能的值組合:

表 *46:* 如何結合 *PUTASYNCALLOWED* 與 *DEFPRESP* 內容，以決定是否非同步將訊息放置到目的地。

IBM MQ 佇列內容	**PUTASYNCALLOWED = NO**	**PUTASYNCALLOWED = YES**	**已啟用非同步放置功能**
DEFPRESP=SYNC	未啟用非同步放置功能	已啟用非同步放置功能	PUTASYNCALLOWED = AS_DEST 或 AS_Q_DEF 或 AS_T_DEF
DEFPRESP=ASYNC	未啟用非同步放置功能	已啟用非同步放置功能	PUTASYNCALLOWED = AS_DEST 或 AS_Q_DEF 或 AS_T_DEF

您可以依照表格所示，指定 IBM MQ-JMS Destination 內容來指出 "NO" 或 "YES" ， 以變更行為，但也可以針對整個 Java 虛擬機器使用 JVM **SystemProperty** 及值來置換它:

```
com.ibm.mq.cfg.Channels.Put1DefaultAlwaysSync=Y
```

對於在交易式階段作業中傳送的訊息，應用程式最終會判斷佇列管理程式在呼叫 `commit()`時是否安全地接收訊息。

如果應用程式在交易式階段作業內傳送持續訊息，且未安全地接收一或多個訊息，則交易無法確定並產生異常狀況。 不過，如果應用程式在交易式階段作業內傳送非持續訊息，且未安全地接收一或多個訊息，則交易會順利確定。 應用程式不會收到非持續訊息未安全送達的任何意見。

對於在未交易的階段作業中傳送的非持續訊息， *ConnectionFactory* 物件的 SENDCHECKCOUNT 內容會指定要傳送的訊息數，然後 IBM MQ classes for JMS 會檢查佇列管理程式是否已安全接收訊息。

如果檢查發現未安全接收一或多個訊息，且應用程式已向連線登錄異常狀況接聽器，則 IBM MQ classes for JMS 會呼叫異常狀況接聽器的 `onException()` 方法，以將 JMS 異常狀況傳遞至應用程式。

JMS 異常狀況的錯誤碼為 JMSWMQ0028 ，且此程式碼會顯示下列訊息:

```
At least one asynchronous put message failed or gave a warning.
```

JMS 異常狀況也有一個鏈結的異常狀況，提供更多詳細資料。 SENDCHECKCOUNT 內容的預設值為零，表示不進行此類檢查。

對於以用戶端模式連接至佇列管理程式，且需要快速連續傳送一連串訊息，但不需要佇列管理程式針對所傳送的每一則訊息立即提供意見的應用程式而言，此最佳化最有好處。 不過，即使應用程式以連結模式連接至佇列管理程式，也仍然可以使用此最佳化，但預期的效能好處沒有那麼大。

註: 如果您使用無法識別的 **MessageProducer** 在交易下傳送訊息，則依預設會使用非同步放置機制將訊息放置到佇列中。

因為 JMS API 容許建立 **MessageProducer** ，而不使用下列語法來指定「目的地」:

```
javax.jms.MessageProducer messageProducer =  javax.jms.Session.createProducer(null);
  messageProducer.send(Destination destination, Message message, int deliveryMode, int priority, long
```

```
timeToLive);
```

在此實務範例中，會在傳送訊息時提供 JMS 目的地，而不是在建構 **MessageProducer** 時提前提供。 就 IBM MQ API 而言，這會導致發出 MQPUT1 ，將訊息放入佇列。

如果您在 IBM MQ 同步點下執行此動作，這表示 (在 JMS 術語中) 使用交易式 JMS 階段作業，或透過使用 XASession IBM MQ classes for JMS API 切換至使用非同步放置，將訊息置於交易之下。

搭配使用先讀與 *IBM MQ classes for JMS*

IBM MQ 提供的先讀功能容許在應用程式要求非持續訊息之前，將在交易之外接收到的非持續訊息傳送至 IBM MQ classes for JMS 。 IBM MQ classes for JMS 會將訊息儲存在內部緩衝區中，並在應用程式要求時將訊息傳遞至應用程式。

使用 MessageConsumers 或 MessageListeners 從交易之外的目的地接收訊息的 IBM MQ classes for JMS 應用程式可以使用先讀功能。 使用先讀可讓使用這些物件的應用程式在接收訊息時受益於改良的效能。

使用 MessageConsumers 或 MessageListeners 的應用程式是否可以使用先讀，取決於下列內容:

JMS 目的地內容 READAHHEADALLOWED (簡稱-RAALD)。
READAHEADALLOWED 控制如果 JMS 目的地代表的基礎佇列或主題容許此選項，在取得或瀏覽交易之外的非持續訊息時， JMS 應用程式是否可以使用先讀。

IBM MQ 佇列或主題內容 DEFREADA (預設先讀)。
DEFREADA 指定在交易之外接收或瀏覽非持續訊息的應用程式是否可以使用先讀。

下表顯示 READAheadallowed 和 DEFREADA 內容的可能值，以及用來啟用先讀功能的值組合:

表 *47:* 如何結合 *READAHEADALLOWED* 和 *DEFREADA* 內容，以決定在接收或瀏覽交易之外的非持續訊息時，是否使用先讀。

IBM MQ 佇列內容	**READAHEADALLOWED = YES**	**READAHEADALLOWED = NO**	**AS_DEST 或 AS_Q_DEF 或 AS_T_DEF**
DEFREADA = NO	已啟用先讀功能	未啟用先讀功能	未啟用先讀功能
DEFREADA = YES	已啟用先讀功能	未啟用先讀功能	已啟用先讀功能
DEFREADA = DISABLED	未啟用先讀功能	未啟用先讀功能	未啟用先讀功能

如果啟用先讀功能，當應用程式建立 MessageConsumer 或 MessageListener 時， IBM MQ classes for JMS 會為 MessageConsumer 或 MessageListener 所監視的目的地建立內部緩衝區。 每一個 MessageConsumer 或 MessageListener 都有一個內部緩衝區。 當應用程式呼叫下列其中一種方法時，佇列管理程式會開始將非持續訊息傳送至 IBM MQ classes for JMS :

- MessageConsumer.receive()
- MessageConsumer.receive(long timeout)
- MessageConsumer.receiveNoWait()
- Session.setMessageListener(MessageListener listener)

IBM MQ classes for JMS 會透過應用程式所進行的方法呼叫，自動將第一個訊息傳回給應用程式。 其他非持續訊息由 IBM MQ classes for JMS 儲存在為目的地建立的內部緩衝區中。 當應用程式要求處理下一個訊息時， IBM MQ classes for JMS 會在內部緩衝區中傳回下一個訊息。

當內部緩衝區是空的時， IBM MQ classes for JMS 會從佇列管理程式要求更多非持續訊息。

當應用程式關閉 MessageConsumer 或與 MessageListener 相關聯的 JMS 階段作業時，會刪除 IBM MQ classes for JMS 所使用的內部緩衝區。

對於 MessageConsumers，會遺失內部緩衝區中任何未處理的訊息。

使用 MessageListeners 時，內部緩衝區中的訊息會發生什麼情況，取決於 JMS 目的地內容 READAHHEADCLOSEPOLICY (簡稱-RACP)。 此內容的預設值為 DELIVER_ALL ，這表示在將內部緩衝區中的所有訊息遞送至應用程式之前，不會關閉用來建立 MessageListener 的 JMS 階段作業。 如果內容設為

DELIVER_CURRENT，則在應用程式處理現行訊息之後，會關閉 JMS 階段作業，並捨棄內部緩衝區中的所有剩餘訊息。

IBM MQ classes for JMS 中保留的發佈資訊
IBM MQ classes for JMS 用戶端可以配置成使用保留的發佈資訊。

發佈者可以指定必須保留發佈的副本，以便將它傳送給未來登錄主題感興趣的訂閱者。 在 IBM MQ classes for JMS 中，將整數內容 JMS_IBM_RETAIN 設為值 1 即可完成此動作。 已在 com.ibm.msg.client.jms.JmsConstants 介面中定義這些值的常數。 例如，如果您已建立訊息 *msg*，若要將它設為保留的發佈資訊，請使用下列程式碼:

```
// set as a retained publication
msg.setIntProperty(JmsConstants.JMS_IBM_RETAIN, JmsConstants.RETAIN_PUBLICATION);
```

您現在可以正常傳送訊息。 JMS_IBM_RETAIN 也可以在收到的訊息中查詢。 因此，可以查詢收到的訊息是否為保留的發佈資訊。

IBM MQ classes for JMS 中的 *XA* 支援

JMS 在連結及用戶端模式中支援符合 XA 標準的交易，並在 JEE 儲存器內具有受支援的交易管理程式。

如果您在應用程式伺服器環境中需要 XA 功能，則必須適當地配置應用程式。 如需如何配置應用程式以使用分散式交易的相關資訊，請參閱應用程式伺服器自己的文件。

IBM MQ 佇列管理程式無法作為 JMS 的交易管理程式。

JMS 訊息的遞送延遲

對於 JMS 2.0 或更新版本，您可以指定傳送訊息時的遞送延遲。 在經歷指定的遞送延遲之後，佇列管理程式才會遞送訊息。

應用程式可以使用 `MessageProducer.setDeliveryDelay(long deliveryDelay)` 或 `JMSProducer.setDeliveryDelay(long deliveryDelay)`來指定傳送訊息時的遞送延遲 (毫秒)。 此值會新增至傳送訊息的時間，並提供任何其他應用程式可以取得該訊息的最早時間。

遞送延遲是使用單一內部暫置佇列來實作。 具有非零遞送延遲的訊息會放置在此佇列上，其標頭指出遞送延遲及目標佇列的相關資訊。 稱為遞送延遲處理器的佇列管理程式元件會監視暫置佇列上的訊息。 當訊息的遞送延遲完成時，會從暫置佇列中取出訊息，並放在目標佇列中。

傳訊用戶端

只有在您使用 JMS 用戶端時，才能使用遞送延遲的 IBM MQ 實作。 如果您對 IBM MQ 使用遞送延遲，則下列限制適用。 這些限制同樣適用於 `MessageProducers` 和 `JMSProducers`，但若為 `JMSProducers`，則會擲出 `JMSRuntimeExceptions`。

- 當連接至早於 IBM MQ 8.0 的佇列管理程式時，如果嘗試以非零值呼叫 `MessageProducer.setDeliveryDelay`，則會導致 `JMSException` 出現 MQRC_FUNCTION_NOT_SUPPORTED 訊息。
- 對於其 **DEFBIND** 值不是 MQBND_BIND_NOT_FIXED 的叢集目的地，不支援遞送延遲。 如果 `MessageProducer` 已設定非零遞送延遲，且嘗試傳送至不符合此需求的目的地，則呼叫會產生具有 MQRC_OPTIONS_ERROR 訊息的 `JMSException`。
- 任何嘗試設定的存活時間值小於先前指定的非零遞送延遲 (反之亦然) 會導致 `JMSException` 出現 MQRC_EXPIRY_ERROR 訊息。 此檢查是在呼叫 `setTimeToLive` 或 `setDeliveryDelay` 或 `send` 方法時執行，視所選擇的確切作業集而定。
- 不支援使用保留的發佈及遞送延遲。 如果使用 `msg.setIntProperty(JmsConstants.JMS_IBM_RETAIN, JmsConstants.RETAIN_PUBLICATION)` 將訊息標示為保留，則嘗試發佈具有遞送延遲的訊息會產生具有 MQRC_OPTIONS_ERROR 訊息的 `JMSException`。
- 不支援遞送延遲及訊息分組，任何嘗試使用此組合都會導致 `JMSException` 出現 MQRC_OPTIONS_ERROR 訊息。

如果無法傳送具有遞送延遲的訊息，則會導致用戶端擲出含有適當錯誤訊息 (例如佇列已滿) 的 `JMSException`。 在某些情況下，錯誤訊息可能適用於目標目的地及/或暫置佇列。

註: IBM MQ 可讓將訊息放入工作單元的應用程式重新取得相同的訊息，即使工作單元尚未確定。 此技術不適用於遞送延遲，因為在確定工作單元之前不會將訊息放置在暫置佇列上，因此不會將訊息傳送至目標目的地。

授權

當應用程式傳送具有非零遞送延遲的訊息時， IBM MQ 會對原始目標目的地執行授權檢查。 如果應用程式未獲授權，則傳送會失敗。 當佇列管理程式偵測到訊息的遞送延遲已完成時，它會開啟目標佇列。 此時不會執行授權檢查。

SYSTEM.DDELAY.LOCAL.QUEUE

系統佇列 SYSTEM.DDELAY.LOCAL.QUEUE，用於實作遞送延遲。

- Multi 在 多平台上， SYSTEM.DDELAY.LOCAL.QUEUE 存在。 必須變更系統佇列，使其 MAXMSGL 及 MAXDEPTH 屬性足以應付預期的負載。
- z/OS 在 IBM MQ for z/OS 上， SYSTEM.DDELAY.LOCAL.QUEUE 用來作為訊息的暫置佇列，這些訊息會以遞送延遲方式傳送至本端及共用佇列。 在 z/OS 上，必須建立並定義佇列，使其 MAXMSGL 及 MAXDEPTH 屬性足以應付預期負載。

建立此佇列時，必須確保其安全，以便儘可能少的使用者可以存取它。 對佇列的存取權必須僅用於維護及監視目的。

當 JMS 應用程式傳送具有非零遞送延遲的訊息時，會以新的訊息 ID 將訊息放入此佇列。 原始訊息 ID 會放在訊息的相關性 ID 中。 此相關性 ID 可讓應用程式在必要時從暫置佇列擷取訊息，例如，如果錯誤地使用了大量遞送延遲。

z/OS 的考量

z/OS

如果您的系統是在 z/OS 上執行，當您想要使用遞送延遲時，還有其他考量要考量。

如果要使用遞送延遲，則系統佇列 SYSTEM.DDELAY.LOCAL.QUEUE。 它必須以足以應付預期載入的儲存類別來定義，且必須指定 INDXTYPE (NONE) 及 MSGDLVSQ (FIFO)。 系統佇列的範例定義在 CSQ4INSG JCL 中提供並註銷。

共用佇列

支援將訊息傳送至共用佇列的遞送延遲。 不過，不論目標佇列是否共用，都只會使用單一專用暫置佇列。 擁有該專用佇列的佇列管理程式必須在執行中，才能在延遲完成時將延遲訊息傳送至其目標共用佇列。

註: 如果將具有遞送延遲的非持續訊息放置到共用佇列，且擁有暫置佇列的佇列管理程式關閉，則原始訊息會遺失。 因此，傳送至共用佇列且具有遞送延遲的非持續訊息比傳送至共用佇列且沒有遞送延遲的非持續訊息更有可能遺失。

目標目的地解析

如果訊息傳送至佇列，則會兩次驅動解析; 一次由 JMS 應用程式驅動，另一次由佇列管理程式驅動，當它將訊息從暫置佇列移出並傳送至目標佇列時。

當 JMS 應用程式呼叫傳送方法時，會比對發佈的目標訂閱。

如果根據佇列定義以持續性或優先順序傳送訊息，則會在第一個解析上設定值，而不是在第二個解析上設定值。

到期間隔

遞送延遲會保留期限內容 **MQMD.Expiry** 的行為。 例如，如果從到期間隔為 20,000 毫秒且遞送延遲為 5,000 毫秒的 JMS 應用程式放置訊息，並在經歷時間 10,000 毫秒之後取得，則 MQMD.expiry 欄位的值可能大約是 50 十分之一秒。 此值指出從放置訊息的時間到取得訊息的時間已經過 15 秒。

如果訊息在暫置佇列上到期，且已設定其中一個 MQRO_EXPIRATION_ * 選項，則產生的報告是針對應用程式所傳送的原始訊息，則會移除用來包含遞送延遲資訊的標頭。

停止並啟動遞送延遲處理器

z/OS 在 z/OS 上，遞送延遲處理器會整合至佇列管理程式 MSTR 位址空間。 當佇列管理程式啟動時，也會啟動遞送延遲處理器。 如果暫置佇列可用，則它會開啟佇列，並等待訊息到達該佇列以進行處理。 如果尚未定義暫置佇列，或已停用取得，或發生另一個錯誤，則會關閉遞送延遲處理器。 如果稍後定義暫置佇列，或變更為啟用暫置佇列，則遞送延遲處理器會重新啟動。 如果遞送延遲處理器因任何其他原因而關閉，則可以透過將暫置佇列的 **PUT** 屬性從 ENABLED 變更為 **已停用** ，然後重新回到 ENABLED ，來重新啟動它。 如果您因為任何原因而需要停止遞送延遲處理器，請將暫置佇列的 PUT 屬性設為 DISABLED。

Multi 在 多平台上，延遲處理器會以佇列管理程式啟動，並在發生可回復失敗時自動重新啟動。

無法放入目標佇列

如果延遲訊息在其延遲完成之後無法放入目標佇列，則會依照其報告選項中的指示來處理訊息: 它會被捨棄或傳送至無法傳送的郵件佇列。 如果此動作失敗，則稍後會嘗試放置訊息。 如果動作成功，如果要求報告，則會產生異常狀況報告並傳送至指定的佇列。 如果無法傳送報告訊息，則會將報告訊息傳送至無法傳送的郵件佇列。 如果將報告傳送至無法傳送的郵件佇列失敗且訊息持續存在，則會捨棄所有變更，且原始訊息會回復並稍後重新遞送。 如果訊息是非持續性，則會捨棄報告訊息，但會確定其他變更。 如果因為訂閱者已取消訂閱而無法遞送延遲發佈，或如果是不可延續訂閱者，因為它已斷線，則會無聲自動捨棄訊息。 仍如先前所述產生報告訊息。

如果延遲發佈無法遞送至訂閱者，而是放置到無法傳送的郵件佇列，且放置到無法傳送的郵件佇列失敗，則會捨棄訊息。

為了減少在遞送延遲完成之後放置到目標佇列失敗的可能性，當 JMS 用戶端傳送具有非零遞送延遲的訊息時，佇列管理程式會執行一些基本檢查。 這些檢查包括佇列是否已停用、訊息是否大於容許的訊息長度上限，以及佇列是否已滿。

發佈/訂閱

當 JMS 應用程式傳送具有非零遞送延遲的訊息時，會進行發佈與可用訂閱的比對。 會將每一個相符訂閱者的訊息放置到 SYSTEM.DDELAY.LOCAL.QUEUE 佇列，它會保留到遞送延遲完成為止。 如果其中一個訂閱者是另一個佇列管理程式的 Proxy 訂閱，則在遞送延遲完成之後，會在該佇列管理程式上展開。 這可能會導致其他佇列管理程式上的訂閱者接收在訂閱之前最初發佈的發佈。 這是與 JMS 2.0 或更新版本規格的偏差。

只有在目標主題已配置 (N) PMSGDLV = ALLAVAIL 時，才支援具有發佈/訂閱的遞送延遲。 嘗試使用任何其他值會導致 MQRC_PUBLICATION_FAILURE 錯誤。 如果遞送延遲處理器在將訊息放置到目標佇列時失敗，則結果如「無法放置到目標佇列」區段中所述。

報告訊息

遞送處理器會支援並處理所有報告選項，但下列選項除外，這些選項會被忽略，但在將訊息傳送至目標佇列時在訊息上傳遞:

- MQRO_COA*
- MQRO_COD*
- MQRO_PAN/MQRO_NAN
- MQRO_ACTIVITY

複製及共用訂閱

在 IBM MQ 8.0 或更新版本中，有兩種方法可讓多個消費者存取相同的訂閱。 這兩種方法是使用複製的訂閱或使用共用訂閱。

複製的訂閱

複製的訂閱是 IBM MQ 延伸。 複製的訂閱可讓不同 Java 虛擬機器 (JVM) 中的多個消費者並行存取訂閱。 您可以將 ConnectionFactory 物件上的 **CLONESUPP** 內容設為 Enabled，來使用這個行為。 依預設，**CLONESUPP** 是 **已停用**。 複製的訂閱只能在可延續訂閱上啟用。 如果啟用 **CLONESUPP**，則會複製使用這個 ConnectionFactory 所建立的每一個後續連線。

如果建立一或多個消費者來接收來自該訂閱的訊息 (亦即，它們是指定相同的訂閱名稱來建立的)，則可延續訂閱可視為已複製。 只有在建立消費者的連線已在 MQConnectionFactory 上 **CLONESUPP** 設為 **已啟用** 時，才能執行此動作。 在訂閱主題上發佈訊息時，該訊息的副本會傳送至訂閱。 訊息可供任何消費者使用，但只有一個消費者收到。

註: 啟用複製的訂閱會延伸 JMS 規格。

共用訂閱

JMS 2.0 規格所引進的共用訂閱可讓來自主題訂閱的訊息在多個消費者之間共用。 來自訂閱的每一則訊息只會遞送給該訂閱上的其中一個消費者。 JMS 2.0 或更新版本 API 的相關呼叫會啟用共用訂閱。

您可以使用下列其中一種方式來呼叫 API:

- 從 Java SE 應用程式 (或 Java EE 用戶端儲存器)。
- 從 Servlet 或 MDB 的實作。

JMS 2.0 或更新版本規格未定義從共用訂閱驅動 MDB 的任何標準方式，因此 IBM MQ 8.0 或更新版本會為此目的提供 **sharedSubscription** 啟動規格內容。 如需此內容的相關資訊，請參閱 第 379 頁的『配置入埠通訊的資源配接器』 和 第 391 頁的『如何定義 sharedSubscription 內容的範例』。

如果已啟用共用訂閱，則無法取消共用。

共用訂閱可以建立為可延續或不可延續訂閱。 除了一般 JMS 配置之外，不需要在佇列管理程式端個別建立任何物件。 任何需要的物件都會動態建立。

決定共用或複製的訂閱

當您決定要使用共用或複製的訂閱時，請考量兩者的好處。 可能的話，請使用共用訂閱，因為它是規格定義的行為，而不是 IBM MQ 特定的延伸。

下表包含在您決定共用與複製的訂閱時要考量的一些要點:

表 *48:* 共用訂閱與複製訂閱的考量比較

共用訂閱	**複製的訂閱**
共用訂閱是 JMS 2.0 或更新版本規格的標準部分。	複製的訂閱是 IBM MQ 特定的延伸。
共用訂閱是使用明確 API 方法呼叫來建立。	複製的訂閱在 ConnectionFactory 層次以管理方式進行控制。
共用訂閱可以是可延續或不可延續。	複製的訂閱只能可延續。
共用訂閱是以個別訂閱為基礎明確建立。	複製的訂閱用於已啟用功能之連線下的任何可延續訂閱。
如果將訂閱建立為共用，則稍後無法變更為未共用，反之亦然。	如果擁有連線的 **CLONESUPP** 內容已變更，則每次重新開啟訂閱時，可以將訂閱從複製變更為取消複製。

相關概念

訂閱者和訂閱

訂閱延續性

相關工作

使用 JMS 2.0 共用訂閱

相關參考

第 391 頁的『如何定義 sharedSubscription 內容的範例』
您可以在 WebSphere Liberty `server.xml` 檔內定義啟動規格的 sharedSubscription 內容。 或者，您可以使用註釋來定義訊息驅動 Bean (MDB) 內的內容。

CLONESUPP

SupportMQExtensions 內容

JMS 2.0 規格引進了某些行為運作方式的變更。 IBM MQ 8.0 以及更新版本包括內容 **com.ibm.mq.jms.SupportMQExtensions**，可以設定為 TRUE 以將這些變更的行為回復為先前的實作。

JM 3.0 V 9.3.0 V 9.3.0 IBM MQ classes for Jakarta Messaging 支援 **com.ibm.mq.jakarta.jms.SupportMQExtensions** 內容 (Jakarta Messaging 3.0)，其在 `com.ibm.mq.jakarta.client.jar` 中提供。

JMS 2.0 IBM MQ classes for JMS 支援 **com.ibm.mq.jms.SupportMQExtensions** (JMS 2.0) 內容，可在 `com.ibm.mq.allclient.jar` 或 `com.ibm.mqjms.jar` 中找到。

將 **SupportMQExtensions** 設為 `True` 會回復三個功能區域:

訊息優先順序
: 訊息可以指派 0-9 的優先順序。 在 JMS 2.0 之前，訊息也可以使用值 -1，指出使用佇列的預設優先順序。 JMS 2.0 以及更新版本不容許設定訊息優先順序 -1。 開啟 **SupportMQExtensions** 容許使用值 -1。

用戶端 ID
: JMS 2.0 或更新版本規格要求在建立連線時檢查非空值用戶端 ID 的唯一性。 開啟 **SupportMQExtensions** 表示不處理此需求，且可以重複使用用戶端 ID。

NoLocal
: JMS 2.0 或更新版本規格要求當開啟這個常數時，消費者無法接收相同用戶端 ID 所發佈的訊息。 在 JMS 2.0 之前，已在訂閱者上設定此屬性，以防止它接收由自己的連線發佈的訊息。 開啟 **SupportMQExtensions** 會將此行為回復為先前的實作。

此內容可以設定如下:

```
java -Dcom.ibm.mq.jms.SupportMQExtensions=true
```

此內容可以設為 **java** 指令上的標準 JVM 系統內容，或包含在 IBM MQ classes for JMS 配置檔內。

相關概念

第 83 頁的『IBM MQ classes for JMS/Jakarta Messaging 配置檔』
IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 配置檔指定用來配置 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 的內容。

相關參考

第 89 頁的『用來配置 JMS 用戶端行為的內容』
使用這些內容來配置 JMS 用戶端的行為。

在 *JMS* 應用程式中使用共用訂閱

使用共用訂閱，單一訂閱會在多個消費者之間共用，且只有一個消費者在任何時間點會收到發佈。

從 JMS 2.0 開始，可以使用共用訂閱。 當您針對 IBM MQ 8.0 或更新版本開發 JMS 應用程式時，因此可能需要考量此功能對佇列管理程式的影響。

共用訂閱背後的構想是在多個消費者之間共用負載。 可延續訂閱也可以在多個消費者之間共用。

例如，假設有:

- 訂閱 SUB，訂閱主題 FIFA2014/UPDATES 以接收足球比賽更新項目，由三個消費者 C1、 C2 及 C3 共用

- FIFA2014/UPDATES 主題上的生產者 P1 發佈

當在 FIFA2014/UPDATES 上製作出版品時，只有三個消費者 (C1、 C2 或 C3) 中的一個會收到該出版品，但並非全部。

下列範例示範共用訂閱的使用情形，也示範 JMS 2.0(`Message.receiveBody()`) 中的其他 API 使用情形，以便只擷取訊息內文。

範例會建立三個訂閱者執行緒，分別建立 FIFA2014/UPDATES 主題的共用訂閱，以及一個發佈者執行緒。

JM 3.0 V 9.3.0 V 9.3.0

```
package mqv91Samples;

import jakarta.jms.JMSException;

import com.ibm.msg.client.jms.JmsConnectionFactory;
import com.ibm.msg.client.jms.JmsFactoryFactory;
import com.ibm.msg.client.wmq.WMQConstants;

import jakarta.jms.JMSContext;
import jakarta.jms.Topic;
import jakarta.jms.Queue;
import jakarta.jms.JMSConsumer;
import jakarta.jms.Message;
import jakarta.jms.JMSProducer;

/*
* Implements both Subscriber and Publisher
*/
class SharedNonDurableSubscriberAndPublisher implements Runnable {
   private Thread t;
   private String threadName;

   SharedNonDurableSubscriberAndPublisher( String name){
       threadName = name;
       System.out.println("Creating Thread:" +  threadName );
   }

   /*
    * Demonstrates shared non-durable subscription in JMS 2.0 and later
    */
    private void sharedNonDurableSubscriptionDemo(){
        JmsConnectionFactory cf = null;
        JMSContext msgContext = null;

        try {
            // Create Factory for WMQ JMS provider
            JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);
            // Create connection factory
            cf = ff.createConnectionFactory();
            // Set MQ properties
            cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
            cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);
            // Create message context
            msgContext = cf.createContext();

            // Create a topic destination
            Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

            // Create a consumer. Subscription name specified, required for sharing of subscription.
            JMSConsumer msgCons = msgContext.createSharedConsumer(fifaScores, "FIFA2014SUBID");

            // Loop around to receive publications
            while(true){

                String msgBody=null;

                // Use JMS 2.0 and later receiveBody method as we are interested in message body only.
                msgBody = msgCons.receiveBody(String.class);

                if(msgBody != null){
                    System.out.println(threadName + " : " + msgBody);
                }
            }
        }catch(JMSException jmsEx){
            System.out.println(jmsEx);
        }
    }
```

JMS 2.0

```
package mqv91Samples;

import javax.jms.JMSException;

import com.ibm.msg.client.jms.JmsConnectionFactory;
import com.ibm.msg.client.jms.JmsFactoryFactory;
import com.ibm.msg.client.wmq.WMQConstants;

import javax.jms.JMSContext;
import javax.jms.Topic;
import javax.jms.Queue;
import javax.jms.JMSConsumer;
import javax.jms.Message;
import javax.jms.JMSProducer;

/*
* Implements both Subscriber and Publisher
*/
class SharedNonDurableSubscriberAndPublisher implements Runnable {
   private Thread t;
   private String threadName;

   SharedNonDurableSubscriberAndPublisher( String name){
       threadName = name;
       System.out.println("Creating Thread:" +  threadName );
   }

   /*
    * Demonstrates shared non-durable subscription in JMS 2.0 and later
    */
    private void sharedNonDurableSubscriptionDemo(){
        JmsConnectionFactory cf = null;
        JMSContext msgContext = null;

        try {
            // Create Factory for WMQ JMS provider
            JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);
            // Create connection factory
            cf = ff.createConnectionFactory();
            // Set MQ properties
            cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
            cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);
            // Create message context
            msgContext = cf.createContext();

            // Create a topic destination
            Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

            // Create a consumer. Subscription name specified, required for sharing of subscription.
            JMSConsumer msgCons = msgContext.createSharedConsumer(fifaScores, "FIFA2014SUBID");

            // Loop around to receive publications
            while(true){

                String msgBody=null;

                // Use JMS 2.0 and later receiveBody method as we are interested in message body only.
                msgBody = msgCons.receiveBody(String.class);

                if(msgBody != null){
                    System.out.println(threadName + " : " + msgBody);
                }
            }
        }catch(JMSException jmsEx){
            System.out.println(jmsEx);
        }
    }
```

```
 /*
        * Publisher publishes match updates like current attendance in the stadium, goal score and ball
possession by teams.
        */
        private void matchUpdatePublisher(){
            JmsConnectionFactory cf = null;
            JMSContext msgContext = null;
            int nederlandsGoals = 0;
            int chileGoals = 0;
            int stadiumAttendence = 23231;
```

```
            int switchIndex = 0;
            String msgBody = "";
            int nederlandsHolding = 60;
            int chileHolding = 40;

            try {
                // Create Factory for WMQ JMS provider
                JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);

                // Create connection factory
                cf = ff.createConnectionFactory();
                // Set MQ properties
                cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
                cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);

                // Create message context
                msgContext = cf.createContext();

                // Create a topic destination
                Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

                // Create publisher to publish updates from stadium
                JMSProducer msgProducer = msgContext.createProducer();

                while(true){
                    // Send match updates
                    switch(switchIndex){
                        // Attendance
                    case 0:
                        msgBody ="Stadium Attendence " + stadiumAttendence;
                        stadiumAttendence += 314;
                        break;

                        // Goals
                    case 1:
                        msgBody ="SCORE: The Netherlands: " + nederlandsGoals + " - Chile:" + chileGoals;
                        break;

                        // Ball possession percentage
                    case 2:
                        msgBody ="Ball possession: The Netherlands: " + nederlandsHolding + "% - Chile:
" + chileHolding + "%";
                        if((nederlandsHolding > 60) && (nederlandsHolding < 70)){
                            nederlandsHolding -= 2;
                            chileHolding += 2;
                        }else{
                            nederlandsHolding += 2;
                            chileHolding -= 2;
                        }
                        break;
                    }

                    // Publish and wait for two seconds to publish next update
                    msgProducer.send (fifaScores, msgBody);
                    try{
                        Thread.sleep(2000);
                    }catch(InterruptedException iex){

                    }

                    // Increment and reset the index if greater than 2
                    switchIndex++;
                    if(switchIndex > 2)
                        switchIndex = 0;
                }
            }catch(JMSException jmsEx){
                System.out.println(jmsEx);
            }
        }

    /*
     * (non-Javadoc)
     * @see java.lang.Runnable#run()
     */
    public void run() {
        // If this is a publisher thread
        if(threadName == "PUBLISHER"){
            matchUpdatePublisher();
        }else{
            // Create subscription and start receiving publications
            sharedNonDurableSubscriptionDemo();
        }
```

```
    }

       // Start thread
   public void start (){
      System.out.println("Starting " +  threadName );
      if (t == null)
      {
         t = new Thread (this, threadName);
         t.start ();
      }
   }
}
```

```
/*
* Demonstrate JMS 2.0 and later simplified API using IBM MQ 91 JMS Implementation
*/
public class Mqv91jms2Sample {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // Create first subscriber and start
        SharedNonDurableSubscriberAndPublisher subOne = new
SharedNonDurableSubscriberAndPublisher( "SUB1");
        subOne.start();

        // Create second subscriber and start
        SharedNonDurableSubscriberAndPublisher subTwo = new
SharedNonDurableSubscriberAndPublisher( "SUB2");
        subTwo.start();

        // Create third subscriber and start
        SharedNonDurableSubscriberAndPublisher subThree = new
SharedNonDurableSubscriberAndPublisher( "SUB3");
        subThree.start();

        // Create publisher and start
        SharedNonDurableSubscriberAndPublisher publisher = new
SharedNonDurableSubscriberAndPublisher( "PUBLISHER");
        publisher.start();
    }
}
```

相關概念

IBM MQ Java 語言介面

V 9.3.2 配置模組化應用程式以使用 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging

V 9.3.2 您可以透過模組化方式來使用 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging，方法是在應用程式內要求適當的模組，並在模組路徑中包括適當的目錄。

模組化包裝

IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 的統一 JAR 檔提供自動模組名稱，以取代衍生自 JAR 檔名稱的預設名稱。

- IBM MQ classes for JMS (`com.ibm.mq.allclient.jar`) 隨附模組名稱 `com.ibm.mq.javax`。
- IBM MQ classes for Jakarta Messaging (`com.ibm.mq.jakarta.client.jar`) 隨附模組名稱 `com.ibm.mq.jakarta`。

預設 `MQ_HOME/java/lib` 目錄不適合模組化使用，因為模組不能包含相同的套件，且預設目錄在多個 JAR 中包含相同的套件。 因此，只有包含所需 JAR 檔的新目錄可供使用， JAR 之間沒有重複的套件。 這些目錄適合併入 `module-path` 中。

註: 如果您有應用程式根據預設模組名稱，在模組環境定義中使用可用的 JAR 檔，您必須更新應用程式，以要求新的模組名稱。 預設模組名稱衍生自 JAR 檔名稱。

配置模組化應用程式以使用 IBM MQ classes for JMS

您可以完成下列步驟，將模組化應用程式配置成使用 IBM MQ classes for JMS (`com.ibm.mq.allclient.jar`):

- 將應用程式配置成需要 `com.ibm.mq.javax` 模組。
- 將應用程式配置成在模組路徑中包含 `MQ_HOME/java/lib/modules/javax` 目錄。

配置模組化應用程式以使用 IBM MQ classes for Jakarta Messaging

您可以完成下列步驟，將模組化應用程式配置成使用 IBM MQ classes for Jakarta Messaging (`com.ibm.mq.jakarta.client.jar`):

- 將應用程式配置成需要 `com.ibm.mq.jakarta` 模組。
- 將應用程式配置成在模組路徑中包含 `MQ_HOME/java/lib/modules/jakarta` 目錄。

配置模組化應用程式以使用 IBM MQ classes for Java

如果要從模組應用程式使用 IBM MQ classes for Java ，您可以使用 IBM MQ classes for JMS 的配置或 IBM MQ classes for Jakarta Messaging 的配置，因為這兩個用戶端 JAR 檔都支援 IBM MQ classes for Java。 不過，您的應用程式只能使用其中一項配置，不能同時使用兩者。

IBM MQ classes for JMS Application Server 機能

這個主題說明 IBM MQ classes for JMS 如何在 Session 類別中實作 ConnectionConsumer 類別和進階功能。 它也會彙總伺服器階段作業儲存區的功能。

重要: 本資訊僅供參考。 應用程式不得撰寫成使用這個介面: 它在 IBM MQ 資源配接器內用來連接 Java EE 伺服器。 如需實際連線資訊，請參閱 第 365 頁的『使用 IBM MQ 資源配接器』。

IBM MQ classes for JMS 支援在 *Java Message Service* 規格 中指定的「應用程式伺服器機能 (ASF)」(請參閱 Oracle Technology Network for Java Developers)。 此規格識別此程式設計模型內的三個角色:

- **JMS 提供者** 提供 ConnectionConsumer 和進階階段作業功能。
- **應用程式伺服器** 提供 ServerSession 儲存區和 ServerSession 功能。
- **用戶端應用程式** 使用 JMS 提供者和應用程式伺服器所提供的功能。

如果應用程式使用與分配管理系統的即時連線，則本主題中的資訊不適用。

JMS ConnectionConsumer

ConnectionConsumer 介面提供高效能方法，可將訊息同時遞送至執行緒儲存區。

JMS 規格可讓應用程式伺服器使用 `ConnectionConsumer` 介面，與 JMS 實作緊密整合。 此特性提供訊息的並行處理。 一般而言，應用程式伺服器會建立執行緒儲存區，而 JMS 實作會使訊息可供這些執行緒使用。 可察覺 JMS 的應用程式伺服器 (例如 WebSphere Application Server) 可以使用此特性來提供高階傳訊功能，例如訊息驅動 Bean。

一般應用程式不會使用 ConnectionConsumer，但專家級 JMS 用戶端可能會使用它。 對於這類用戶端，ConnectionConsumer 提供高效能方法將訊息同時遞送至執行緒儲存區。 當訊息到達佇列或主題時， JMS 會從儲存區中選取一個執行緒，並將一批訊息遞送給它。 為此， JMS 會執行相關聯的 MessageListener 的 `onMessage()` 方法。

您可以透過建構多個 Session 和 MessageConsumer 物件來達到相同的效果，每一個物件都具有已登錄的 MessageListener。 不過， ConnectionConsumer 提供更好的效能、較少使用資源，以及更大的彈性。 尤其是需要較少的「階段作業」物件。

使用 *ASF* 規劃應用程式

本節說明如何規劃應用程式，包括:

- 第 280 頁的『使用 ASF 進行點對點傳訊的一般原則』
- 第 280 頁的『使用 ASF 發佈/訂閱傳訊的一般原則』

- 第 281 頁的『在 ASF 中從佇列移除訊息』
- 在 ASF 中處理有毒訊息。 請參閱 第 199 頁的『在 IBM MQ classes for JMS 中處理有害訊息』。

使用 *ASF* 進行點對點傳訊的一般原則
如需使用 ASF 之點對點傳訊的一般資訊，請使用本主題。

當應用程式從 QueueConnection 物件建立 ConnectionConsumer 時，它會指定 JMS 佇列物件和選取元字串。 然後 ConnectionConsumer 會開始提供訊息給相關聯 ServerSession 儲存區中的階段作業。 訊息抵達佇列，如果符合選取器，則會遞送至相關聯 ServerSession 儲存區中的階段作業。

就 IBM MQ 而言，佇列物件是指本端佇列管理程式上的 QLOCAL 或 QALIAS。 如果它是 QALIAS ，則該 QALIAS 必須參照 QLOCAL。 完整解析的 IBM MQ QLOCAL 稱為 基礎 *QLOCAL*。 如果 ConnectionConsumer 未關閉且其母項 QueueConnection 已啟動，則會被認為是 作用中 。

多個 ConnectionConsumers(每一個都有不同的選取元) 可以針對相同的基礎 QLOCAL 執行。 若要維護效能，不要的訊息不得累積在佇列上。 不想要的訊息是沒有作用中 ConnectionConsumer 具有相符選取器的訊息。 您可以設定 QueueConnectionFactory ，以便將這些不想要的訊息從佇列中移除 (如需詳細資料，請參閱 第 281 頁的『在 ASF 中從佇列移除訊息』)。 您可以使用下列兩種方式之一來設定此行為:

- 使用 JMS 管理工具，將 QueueConnectionFactory 設為 MRET (NO)。
- 在您的程式中，使用:

```
MQQueueConnectionFactory.setMessageRetention(WMQConstants.WMQ_MRET_NO)
```

如果您不變更此設定，預設值是在佇列中保留這類不想要的訊息。

當您設定 IBM MQ 佇列管理程式時，請考量下列要點:

- 必須針對共用輸入啟用基礎 QLOCAL。 若要執行此動作，請使用下列 MQSC 指令:

```
ALTER QLOCAL( your.qlocal.name ) SHARE GET(ENABLED)
```

- 您的佇列管理程式必須已啟用無法傳送郵件的佇列。 如果 ConnectionConsumer 在將訊息放入無法傳送郵件的佇列時發生問題，則會停止從基礎 QLOCAL 遞送訊息。 若要定義無法傳送郵件的佇列，請使用:

```
ALTER QMGR DEADQ( your.dead.letter.queue.name )
```

- 執行 ConnectionConsumer 的使用者必須具有使用 MQOO_SAVE_ALL_CONTEXT 及 MQOO_PASS_ALL_CONTEXT 執行 MQOPEN 的權限。 如需詳細資料，請參閱特定平台的 IBM MQ 文件。
- 如果將不想要的訊息留在佇列中，則會降低系統效能。 因此，請規劃您的訊息選取器，以便在它們之間，ConnectionConsumers 會移除佇列中的所有訊息。

如需 MQSC 指令的詳細資料，請參閱 MQSC 指令。

使用 *ASF* 發佈/訂閱傳訊的一般原則
ConnectionConsumers 會接收指定主題的訊息。 ConnectionConsumer 可以是可延續或不可延續。 您必須指定 ConnectionConsumer 使用的佇列或佇列。

當應用程式從 TopicConnection 物件建立 ConnectionConsumer 時，它會指定 Topic 物件和選取元字串。 然後， ConnectionConsumer 會開始接收符合該主題上選取器的訊息，包括所訂閱主題的任何保留發佈。

或者，應用程式可以建立與特定名稱相關聯的可延續 ConnectionConsumer 。 這個 ConnectionConsumer 會接收自可延續 ConnectionConsumer 前次作用中以來已在主題上發佈的訊息。 它會接收符合「主題」上選取元的所有這類訊息。 不過，如果 ConnectionConsumer 使用先讀，當它關閉時，可能會遺失用戶端緩衝區中的非持續訊息。

如果 IBM MQ classes for JMS 處於 IBM MQ 傳訊提供者移轉模式，則會針對不可延續的 ConnectionConsumer 訂閱使用個別佇列。TopicConnectionFactory 上的 CCSUB 可配置選項指定要使用的佇列。 一般而言， CCSUB 會指定單一佇列，供所有使用相同 TopicConnectionFactory 的 ConnectionConsumers 使用。 不過，您可以指定後面接著星號 (*) 的佇列名稱字首，讓每一個 ConnectionConsumer 產生暫時佇列。

如果 IBM MQ classes for JMS 處於 IBM MQ 傳訊提供者移轉模式，則主題的 CCDSUB 內容會指定用於可延續訂閱的佇列。 同樣地，這可以是已存在的佇列，或後面接著星號 (*) 的佇列名稱字首。 如果您指定已存在的佇列，訂閱主題的所有可延續 ConnectionConsumers 都會使用這個佇列。 如果您指定佇列名稱字首，後面接著星號 (*)，當第一次以特定名稱建立可延續 ConnectionConsumer 時，會產生佇列。 稍後當建立同名的可延續 ConnectionConsumer 時，會重複使用這個佇列。

當您設定 IBM MQ 佇列管理程式時，請考量下列要點:

- 您的佇列管理程式必須已啟用無法傳送郵件的佇列。 如果 ConnectionConsumer 在將訊息放入無法傳送郵件的佇列時發生問題，則會停止從基礎 QLOCAL 遞送訊息。 若要定義無法傳送郵件的佇列，請使用:

```
ALTER QMGR DEADQ( your.dead.letter.queue.name )
```

- 執行 ConnectionConsumer 的使用者必須具有使用 MQOO_SAVE_ALL_CONTEXT 及 MQOO_PASS_ALL_CONTEXT 執行 MQOPEN 的權限。 如需詳細資料，請參閱適用於您平台的 IBM MQ 文件。
- 您可以為個別 ConnectionConsumer 建立個別的專用佇列，使其效能最佳化。 這將以額外資源使用的成本為代價。

在 *ASF* 中從佇列移除訊息
當應用程式使用 ConnectionConsumers 時，在許多情況下 JMS 可能需要從佇列中移除訊息。

這些狀況如下:

訊息格式不正確
JMS 可能無法剖析的訊息到達。

有害訊息
訊息可能達到取消臨界值，但 ConnectionConsumer 無法將它重新排入取消佇列。

沒有感興趣的 ConnectionConsumer
對於點對點傳訊，當設定 QueueConnectionFactory 使其不會保留不想要的訊息時，任何 ConnectionConsumers 都會傳回不想要的訊息。

在這些狀況下， ConnectionConsumer 會嘗試從佇列中移除訊息。 訊息 MQMD 的報告欄位中的處置選項會設定確切的行為。 這兩個選項是:

MQRO_DEAD_LETTER_Q
訊息會重新排入佇列管理程式的無法傳送郵件的佇列。 這是預設值。

MQRO_DISCARD_MSG
已捨棄訊息。

ConnectionConsumer 也會產生報告訊息，這也取決於訊息 MQMD 的報告欄位。 此訊息會傳送至 ReplyTo 佇列管理程式上訊息的 ReplyToQ。 如果在傳送報告訊息時發生錯誤，則會改為將訊息傳送至無法傳送郵件的佇列。 訊息的 MQMD 報告集報告訊息詳細資料的報告欄位中的異常狀況報告選項。 這兩個選項是:

MQRO_Exception
即會產生報告訊息，其中包含原始訊息的 MQMD。 它不包含任何訊息內文資料。

MQRO_EXCEPTION_WITH_DATA
會產生報告訊息，其中包含 MQMD、任何 MQ 標頭，以及 100 個位元組的內文資料。

MQRO_EXCEPTION_WITH_FULL_DATA
即會產生報告訊息，其中包含原始訊息中的所有資料。

預設值
未產生報告訊息。

產生報告訊息時，允許使用下列選項:

- MQRO_NEW_MSG_ID
- MQRO_PASS_MSG_ID
- MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQRO_PASS_CORREL_ID

如果有毒訊息無法重新排入佇列，可能是因為無法傳送郵件的佇列已滿，或錯誤地指定授權，發生的情況取決於訊息的持續性。 如果訊息是非持續性，則會捨棄該訊息，且不會產生任何報告訊息。 如果訊息持續存在，則會停止將訊息遞送至在該目的地上接聽的所有連線消費者。 這類連線消費者必須先關閉並解決問題，然後才能重建它們並重新啟動訊息遞送。

請務必定義無法傳送郵件的佇列，並定期檢查它，以確保不會發生任何問題。 特別是確保無法傳送郵件的佇列未達到其深度上限，且其訊息大小上限足以容納所有訊息。

將訊息重新排入無法傳送郵件的佇列時，會在訊息之前加上 IBM MQ 無法傳送郵件的標頭 (MQDLH)。 如需 MQDLH 格式的詳細資料，請參閱 MQDLH-無法傳送的郵件標頭 。 您可以利用下列欄位來識別 ConnectionConsumer 已放在無法傳送郵件的佇列中的訊息，或報告 ConnectionConsumer 已產生的訊息:

- PutAppl 類型為 `MQAT_JAVA （0x1C）`
- PutAppl 名稱為 `"MQ JMS ConnectionConsumer"`

這些欄位位於無法傳送郵件的佇列上訊息的 MQDLH 中，以及報告訊息的 MQMD 中。 MQMD 的回饋欄位及 MQDLH 的原因欄位包含說明錯誤的代碼。 如需這些代碼的詳細資料，請參閱 第 283 頁的『ASF 中的原因及回饋碼』。 其他欄位如 MQDLH-無法傳送的郵件標頭中所述。

處理 *ASF* 中的有害訊息
在「應用程式伺服器機能」內，有毒訊息處理的處理方式與 IBM MQ classes for JMS 中的其他位置略有不同。

如需 IBM MQ classes for JMS 中有毒訊息處理的相關資訊，請參閱 第 199 頁的『在 IBM MQ classes for JMS 中處理有害訊息』。

當您使用「應用程式伺服器機能 (ASF)」時， ConnectionConsumer(而非 MessageConsumer) 會處理有毒訊息。 ConnectionConsumer 會根據佇列的 BackoutThreshold 及 BackoutRequeue 完整名稱內容來要求訊息。

當應用程式使用 ConnectionConsumers 時，取消訊息的情況取決於應用程式伺服器提供的階段作業:

- 當階段作業非交易式時，若有 AUTO_ANKACK 或 DUPS_OK_ANKACK ，則只有在系統錯誤之後，或應用程式非預期地終止時，才會取消訊息。
- 當階段作業與 CLIENT_ACKNOWLEDGE 非交易時，呼叫 Session.recover() 的應用程式伺服器可以取消未確認的訊息。

  一般而言， MessageListener 或應用程式伺服器的用戶端實作會呼叫 Message.acknowledge()。 Message.acknowledge() 確認到目前為止在階段作業上遞送的所有訊息。
- 當交易階段作業時，呼叫 Session.rollback() 的應用程式伺服器可以取消未確認的訊息。
- 如果應用程式伺服器提供 XASession ，會根據分散式交易來確定或取消訊息。 應用程式伺服器負責完成交易。

相關概念
第 199 頁的『在 IBM MQ classes for JMS 中處理有害訊息』
有害訊息是指接收端應用程式無法處理的訊息。 如果有害訊息遞送至應用程式並回復指定次數，則 IBM MQ classes for JMS 可以將它移至取消佇列。

錯誤處理

本節涵蓋錯誤處理的各個層面，包括 第 282 頁的『從 ASF 中的錯誤狀況回復』 和 第 283 頁的『ASF 中的原因及回饋碼』。

從 *ASF* 中的錯誤狀況回復
如果 ConnectionConsumer 遇到嚴重錯誤，則會停止將訊息遞送給所有與相同 QLOCAL 有興趣的 ConnectionConsumers 。 發生此情況時，會通知向受影響連線登錄的任何 ExceptionListener 。 有兩種方法可讓應用程式從這些錯誤狀況中回復。

一般而言，如果 ConnectionConsumer 無法將訊息重新排入無法傳送郵件的佇列，或從 QLOCAL 讀取訊息時遇到錯誤，就會發生這種性質的嚴重錯誤。

因為會通知向受影響連線登錄的任何 ExceptionListener ，您可以使用它們來識別問題的原因。 在某些情況下，系統管理者必須介入來解決問題。

請使用下列其中一項技術，從這些錯誤狀況中回復:

- 對所有受影響的 ConnectionConsumers 呼叫 `close()`。 只有在關閉所有受影響的 ConnectionConsumers 並解決任何系統問題之後，應用程式才能建立新的 ConnectionConsumers 。
- 在所有受影響的連線上呼叫 `stop()`。 在所有連線都已停止且任何系統問題都已解決之後，應用程式可以順利 `start()` 其連線。

ASF 中的原因及回饋碼
請使用原因碼和回饋碼來判斷錯誤的原因。 這裡提供 ConnectionConsumer 所產生的一般原因碼。

若要判斷錯誤的原因，請使用下列資訊:

- 任何報告訊息中的意見代碼
- 無法傳送郵件之佇列中任何訊息的 MQDLH 原因碼

ConnectionConsumers 會產生下列原因碼。

MQRC_BACKOUT_THRESHOLD_REALED (0x93A; 2362)

原因
訊息已達到 QLOCAL 上定義的「取消臨界值」，但未定義「取消佇列」。

在無法定義「取消佇列」的平台上，訊息已達到 JMS 定義的取消臨界值 20。

動作
如果不想要這樣做，請定義相關 QLOCAL 的「取消佇列」。 也請尋找多個取消的原因。

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之 ConnectionConsumers 的任何選取器。 為了維護效能，訊息會重新排入無法傳送郵件的佇列。

動作
如果要避免這種狀況，請確定使用佇列的 ConnectionConsumers 提供一組處理所有訊息的選取元，或設定 QueueConnectionFactory 來保留訊息。

或者，調查訊息來源。

MQRC_JMS_FORMAT_ERROR (0x93C; 2364)

原因
JMS 無法解譯佇列上的訊息。

動作
調查訊息來源。 JMS 通常會以 BytesMessage 或 TextMessage 來遞送非預期格式的訊息。 有時，如果訊息格式非常錯誤，這會失敗。

出現在這些欄位中的其他代碼是由於嘗試將訊息重新排入「取消佇列」失敗所導致。 在此狀況下，程式碼會說明重新排入佇列失敗的原因。 若要診斷這些錯誤的原因，請參閱 API> API 完成及原因碼。

如果無法將報告訊息放置在 ReplyToQ 上，則會將它放置在無法傳送郵件的佇列上。 在此狀況下， MQMD 的回饋欄位已完成，如本主題所述。 MQDLH 中的原因欄位說明無法在 ReplyToQ 上放置報告訊息的原因。

AFS 中伺服器階段作業儲存區的功能
本主題彙總伺服器階段作業儲存區的功能。

第 284 頁的圖 45 彙總 ServerSession 儲存區和 ServerSession 功能的原則。

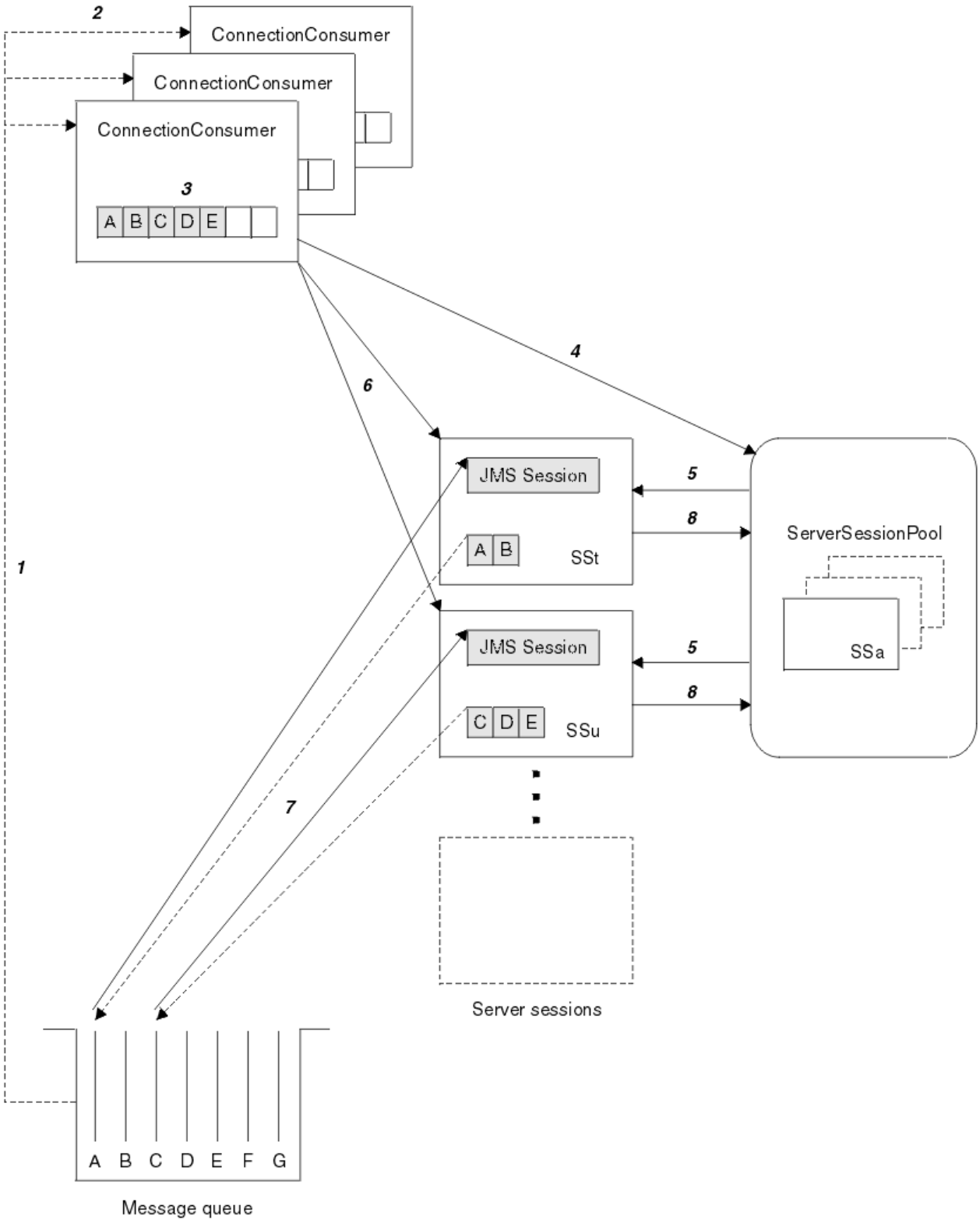


圖 *45: ServerSession* 儲存區和 *ServerSession* 功能

1. ConnectionConsumers 會從佇列取得訊息參照。
2. 每一個 ConnectionConsumer 都會選取特定的訊息參照。
3. ConnectionConsumer 緩衝區會保留選取的訊息參照。
4. ConnectionConsumer 會從 ServerSession 儲存區要求一或多個 ServerSessions 。

5. 從 ServerSession 儲存區配置 ServerSessions 。
6. ConnectionConsumer 會將訊息參照指派給 ServerSessions ， 並啟動執行中的 ServerSession 執行緒。
7. 每一個 ServerSession 都會從佇列中擷取其參照的訊息。 它會從與 JMS 階段作業相關聯的 MessageListener ， 將它們傳遞至 onMessage 方法。
8. 在完成其處理之後， ServerSession 會回到儲存區。

應用程式伺服器通常會提供 ServerSessionPool 和 ServerSession 功能。

z/OS 在 CICS Liberty JVM 伺服器中使用 IBM MQ classes for JMS

從 IBM MQ 9.1.0，在 CICS Liberty JVM 伺服器中執行的 Java 程式可以使用 IBM MQ classes for JMS 來存取 IBM MQ。

您必須使用 IBM MQ 9.1.0 或更新版本的 IBM MQ 資源配接器。 您可以從 Fix Central 取得資源配接器 (請參閱 第 373 頁的『在 Liberty 中安裝資源配接器』)。

CICS 5.3 以及更新版本中可用的 Liberty 設定檔 JVM 有兩個特性， IBM MQ 可能的連線類型限制如下:

CICS Liberty 標準

- IBM MQ 資源配接器可以在 CLIENT 模式下連接至 IBM MQ 的任何使用中版本
- 當相同 CICS 區域中沒有與相同佇列管理程式的 CICS 連線 (作用中 CICS MQCONN 資源定義) 時， IBM MQ 資源配接器可以以 BINDINGS 模式連接至任何使用中版本的 IBM MQ for z/OS 。

CICS Liberty 整合性

- IBM MQ 資源配接器可以在 CLIENT 模式中連接至 IBM MQ 的任何使用中版本。
- 不支援 BINDINGS 模式連線。

如需設定及配置系統的詳細資料，請參閱 CICS 說明文件中的 在 Liberty JVM 伺服器中使用 IBM MQ classes for JMS 。

在 IMS 中使用 IBM MQ classes for JMS/ Jakarta Messaging

IMS 環境內的標準型傳訊支援是透過使用 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 來提供。

請檢查您企業使用之 IMS 系統的系統需求。 如需進一步資訊，請參閱 IMS 15.2 。

這組主題說明如何在 IMS 環境中設定 IBM MQ classes for JMS ， 以及使用標準 (JMS 1.1) 和簡化 (JMS 2.0) 介面時所適用的 API 限制。 如需 API 特定資訊的清單，請參閱 第 289 頁的『JMS API 限制』 。

註: 類似的限制適用於舊式 (JMS 1.0.2) 網域專用介面，但這裡並未特別說明它們。

JM 3.0 V 9.3.0 V 9.3.0 從 IBM MQ 9.3.0 開始，支援 Jakarta Messaging 3.0 開發新的應用程式。 IBM MQ 9.3.0 繼續支援 JMS 2.0 現有的應用程式。 不支援在同一應用程式中同時使用 Jakarta Messaging 3.0 API 和 JMS 2.0 API。 如需相關資訊，請參閱 使用 IBM MQ 類別進行 JMS/Jakarta 傳訊。

支援的 IMS 相依區域

支援下列相依區域類型:

- MPR
- BMP
- IFP
- JMP 31 及 64 位元 Java 虛擬機器 (JVM)
- JBP 31 和 64 位元 JVM

除非在下列主題中特別提及，否則 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 在所有區域類型中的行為都相同。

支援的 Java 虛擬機器

IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 需要 IBM Runtime Environment Java Technology Edition 8。 不支援 IBM Semeru Runtime Certified Edition for z/OS 第 11 版。

其他限制

在 IMS 環境中使用 IBM MQ classes for JMS 時， 下列限制適用:

- 不支援用戶端模式連線。
- 只有使用 IBM MQ 傳訊提供者 `Normal` 模式的 IBM MQ 8.0 佇列管理程式才支援連線。

  Connection Factory 的 **PROVIDERVERSION** 屬性必須未指定，或是大於或等於 7 的值。
- 不支援使用任何 XA Connection Factory， 例如 `com.ibm.mq.jms.MQXAConnectionFactory`。

相關工作

將 IBM MQ 定義為 IMS

設定 *IMS* 配接器以與 *IBM MQ classes for JMS/Jakarta Messaging* 搭配使用

IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 使用其他程式設計語言所使用的相同 IBM MQ-IMS 配接器。 此配接器使用 IMS 外部子系統連接機能 (ESAF)。

開始之前

在完成下列程序之前，您必須為相關佇列管理程式及 IMS 控制項和相依區域配置 IMS 配接器，如 設定 IMS 配接器中所述。

小心: 您不需要執行說明建置動態 Stub 的步驟，除非您需要動態 Stub 作為其他用途。

配置 IMS 配接器之後，請執行下列程序。

程序

1. 更新相依區域 JCL (例如 DFSJVMEV) 中環境參數所參照之 IMS PROCLIB 成員中的 LIBPATH 變數，使其包含 IBM MQ classes for JMS 原生程式庫。

   亦即，包含 `libmqjims.so` 的 zFS 目錄。 例如， DFSJVMEV 可能如下所示，其中最後一行是包含 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 原生程式庫的目錄:

```
LIBPATH=>
/java/latest/bin/j9vm:>
/java/latest/bin:>
/ims/latest/dbdc/imsjava/classic/lib:>
/ims/latest/dbdc/imsjava/lib:>
/mqm/latest/java/lib
```

2. 透過更新 `java.class.path` 選項，將 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 新增至 JVM 的類別路徑 (由 IMS 相依區域使用)。

   請遵循 IMS PROCLIB 資料集的 DFSJVMMS 成員中的指示來執行此動作。

   例如，您可以使用下列，其中粗體行指出更新:

JM 3.0 V 9.3.0 V 9.3.0

```
-Djava.class.path=/ims/latest/dbdc/imsjava/imsutm.jar:/ims/latest/dbdc/imsjava/imsudb.jar:
/mqm/latest/java/lib/com.ibm.mq.jakarta.client.jar
```

JMS 2.0

```
-Djava.class.path=/ims/latest/dbdc/imsjava/imsutm.jar:/ims/latest/dbdc/imsjava/imsudb.jar:
/mqm/latest/java/lib/com.ibm.mq.allclient.jar
```

註: 雖然包含 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 的目錄中有許多不同的 Jar 檔可用，但您只需要 `com.ibm.mq.allclient.jar` (JMS 2.0) 或 `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0)。

3. 停止並重新啟動將使用 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 的任何 IMS 相依區域。

下一步

建立及配置 Connection Factory 和目的地。

有三種可能的方法可將 Connection Factory 和目的地的 IBM MQ 實作實例化。 請參閱第 175 頁的『建立及配置 Connection Factory 和目的地』，以取得詳細資料。

請注意，這三種方法都在 IMS 環境中有效。

相關概念

設定 IMS 配接器

將 IBM MQ 定義為 IMS

交易式行為

IMS 環境中 IBM MQ classes for JMS 所傳送及接收的訊息一律與現行作業上作用中的 IMS 工作單元 (UOW) 相關聯。

只有在 `com.ibm.ims.dli.tm.Transaction` 物件的實例上呼叫確定或回復方法，或 IMS 作業正常結束 (在此情況下會隱含地確定 UOW)，才能完成該 UOW。 如果 IMS 作業異常結束，則會回復 UOW。

因此，當呼叫任何 `Connection.createSession` 或 `ConnectionFactory.createContext` 方法時，會忽略 **transacted** 和 **acknowledgeMode** 引數的值。 此外，不支援下列方法。 在階段作業案例中，呼叫下列任何方法皆會導致 `IllegalStateException`:

- `javax.jms.Session.commit()`
- `javax.jms.Session.recover()`
- `javax.jms.Session.rollback()`

以及 JMS 環境定義案例中的 `IllegalStateRuntimeSession` :

- `javax.jms.JMSContext.commit()`
- `javax.jms.JMSContext.recover()`
- `javax.jms.JMSContext.rollback()`

此行為有一個例外。 如果使用下列其中一種機制來建立階段作業或 JMS 環境定義:

- `Connection.createSession(false, Session.AUTO_ACKNOWLEDGE)`
- `Connection.createSession(Session.AUTO_ACKNOWLEDGE)`
- `ConnectionFactory.createContext(JMSContext.AUTO_ACKNOWLEDGE)`

則該階段作業或 JMS 環境定義的行為如下:

- 任何已傳送的訊息都會在 IMS UOW 之外傳送。 也就是說，它們將立即在目標目的地上可用，或在提供的遞送延遲間隔已完成時可用。
- 除非已在建立階段作業或 JMS 環境定義的 Connection Factory 上指定 SYNCPOINTALLGETS 內容，否則將在 IMS UOW 之外接收任何非持續訊息。
- 持續訊息一律在 IMS UOW 內接收。

例如，如果您想要將審核訊息寫入佇列 (即使 UOW 回復也一樣)，則這可能很有用。

IMS 同步點的含意

IBM MQ classes for JMS 建置在利用 ESAF 的現有 IBM MQ 配接器支援上。 這表示所記載的行為適用，包括發生同步點時由 IMS 配接器關閉的所有開啟控點。

如需相關資訊，請參閱 第 58 頁的『IMS 應用程式中的同步點』。

為了說明這一點，請考量在 JMP 環境中執行下列程式碼。 第二次呼叫 mp.send() 會導致 JMSException，因為 messageQueue.getUnique(inputMessage) 程式碼會導致所有開啟的 IBM MQ 連線及物件控點關閉。

如果 getUnique() 呼叫取代為 Transaction.commit()，則會觀察到類似的行為，但如果使用 Transaction.rollback() 則不會。

```
//Create a connection to queue manager MQ21.
MQConnectionFactory cf = new MQConnectionFactory();
cf.setQueueManager("MQ21");

Connection c = cf.createConnection();
Session s = c.createSession();

//Send a message to MQ queue Q1.
Queue q = new MQQueue("Q1");
MessageProducer mp = s.createProducer(q);
TextMessage m = s.createTextMessage("Hello world!");
mp.send(m);

//Get a message from an IMS message queue. This results in a GU call
//which results in all MQ handles being closed.
Application a = ApplicationFactory.createApplication();
MessageQueue messageQueue = a.getMessageQueue();
IOMessage inputMessage = a.getIOMessage(MESSAGE_CLASS_NAME);
messageQueue.getUnique(inputMessage);

//This attempt to send another message will result in a JMSException containing a
//MQRC_HCONN_ERROR as the connection/handle has been closed.
mp.send(m);
```

在此實務範例中使用的正確程式碼如下。 在此情況下， IBM MQ 的連線會在呼叫 getUnique()之前關閉。 然後會重建連線及階段作業，以傳送另一個訊息。

```
//Create a connection to queue manager MQ21.
MQConnectionFactory cf = new MQConnectionFactory();
cf.setQueueManager("MQ21");

Connection c = cf.createConnection();
Session s = c.createSession();

//Send a message to MQ queue Q1.
Queue q = new MQQueue("Q1");
MessageProducer mp = s.createProducer(q);
TextMessage m = s.createTextMessage("Hello world!");
mp.send(m);

//Close the connection to MQ, which closes all MQ object handles.
//The send of the message will be committed by the subsequent GU call.
c.close();
c = null;
s = null;
mp = null;

//Get a message from an IMS message queue. This results in a GU call.
Application a = ApplicationFactory.createApplication();
MessageQueue messageQueue = a.getMessageQueue();
IOMessage inputMessage = a.getIOMessage(MESSAGE_CLASS_NAME);
messageQueue.getUnique(inputMessage);

//Re-create the connection to MQ and send another message;
c = cf.createConnection();
s = c.createSession();
mp = s.createProducer(q);
m = s.createTextMessage("Hello world 2!");
mp.send(m);
```

使用 *IMS* 配接器時的考量

您需要注意下列限制。 每一個佇列管理程式只能有一個連線控點。 同時使用 JMS 和原生程式碼時，與 IBM MQ 互動會有一些含意。 連線鑑別和授權有一些限制。

每一個佇列管理程式一個連線控點

在 IMS 相依區域中，一次只容許一個與特定佇列管理程式的連線控點。 後續任何連接至相同佇列管理程式的嘗試都會重複使用現有的控點。

雖然此行為應該不會在只使用 IBM MQ classes for JMS 的應用程式中造成任何問題，但此行為可能會在與 IBM MQ 互動的應用程式中造成問題，當同時使用 IBM MQ classes for JMS 及以原生程式碼撰寫的 MQI (例如 COBOL 或 C) 時。

同時使用 JMS 和原生程式碼時與 IBM MQ 互動的含意

交織同時使用 IBM MQ 功能的 Java 程式碼和原生程式碼時，以及在離開原生或 Java 程式碼之前未關閉與 IBM MQ 的連線時，可能會發生問題。

例如，在下列虛擬程式碼中，最初使用 IBM MQ classes for JMS 在 Java 程式碼中建立佇列管理程式的連線控點。 連線控點在 COBOL 程式碼中重複使用，並因 MQDISC 呼叫而失效。

下次 IBM MQ classes for JMS 使用 `JMSException` 連線控點時，原因碼為 MQRC_HCONN_ERROR 結果。

```
COBOL code running in message processing region
    Use the Java Native Interface (JNI) to call Java code
       Create MQ connection and session - this creates an MQ connection handle
       Send message to MQ queue
       Store connection and session in static variable
       Return to COBOL code

    MQCONN - picks up MQ connection handle established in Java code
    MQDISC - invalidates connection handle

    Use the Java Native Interface (JNI) to call Java code
       Get session from static variable
       Create a message consumer - fails as connection handle invalidated
```

還有其他類似的用法型樣可能會導致 MQRC_HCONN_ERROR。

雖然通常可以在原生程式碼與 Java 程式碼之間共用 IBM MQ 連線控點 (例如，如果沒有 MQDISC 呼叫，則前一個範例會運作)，但最佳作法是在從 Java 變更為原生程式碼之前關閉任何連線控點，或以另一種方式來回切換。

連線鑑別和授權

JMS 規格可讓您在建立連線或 JMS 環境定義物件時，指定使用者名稱和密碼來進行鑑別和授權。

這在 IMS 環境中不受支援。 嘗試在指定使用者名稱和密碼時建立連線會導致擲出 `JMS Exception`。 嘗試在指定使用者名稱和密碼時建立 JMS 環境定義，會導致擲出 `JMSRuntimeException`。

相反地，當從 IMS 環境連接至 IBM MQ 時，必須使用現有的鑑別和授權機制。

如需相關資訊，請參閱在 z/OS 上設定安全。 特別請參閱用於安全檢查的使用者 ID，其中說明可以使用的使用者 ID。

相關工作

在 z/OS 上設定安全

JMS API 限制

從 JMS 規格視景中， IBM MQ classes for JMS 會將 IMS 視為符合 Java EE 或 Jakarta EE 標準的應用程式伺服器，且一律會進行 JTA 交易。

例如，您無法在 IMS 中呼叫 `javax.jms.Session.commit()`，因為 JMS 規格指出在 JTA 交易進行期間，您無法在 JEE EJB 或 Web 儲存器中呼叫它。

除了 第 287 頁的『交易式行為』中說明的限制之外，還會對 JMS API 產生下列限制。

典型 API 限制

- javax.jms.Connection.createConnectionConsumer(javax.jms.Destination, String, javax.jms.ServerSessionPool, int) 一律會擲出 JMSException。
- javax.jms.Connection.createDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int) 一律會擲出 JMSException。
- 如果連線已具有作用中的現有階段作業，則 javax.jms.Connection.createSession 的所有三個變式一律會擲出 JMSException。
- javax.jms.Connection.createSharedConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int) 一律會擲出 JMSException。
- javax.jms.Connection.createSharedDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int) 一律會擲出 JMSException。
- javax.jms.Connection.setClientID() 一律會擲出 JMSException。
- javax.jms.Connection.setExceptionListener(javax.jms.ExceptionListener) 一律會擲出 JMSException。
- javax.jms.Connection.stop() 一律會擲出 JMSException。
- javax.jms.MessageConsumer.setMessageListener(javax.jms.MessageListener) 一律會擲出 JMSException。
- javax.jms.MessageConsumer.getMessageListener() 一律會擲出 JMSException。
- javax.jms.MessageProducer.send(javax.jms.Destination, javax.jms.Message,javax.jms.CompletionListener) 一律會擲出 JMSException。
- javax.jms.MessageProducer.send(javax.jms.Destination, javax.jms.Message, int, int, long, javax.jms.CompletionListener) 一律會擲出 JMSException。
- javax.jms.MessageProducer.send(javax.jms.Message, int, int, long, javax.jms.CompletionListener) 一律會擲出 JMSException。
- javax.jms.MessageProducer.send(javax.jms.Message,javax.jms.CompletionListener) 一律會擲出 JMSException。
- javax.jms.Session.run() 一律會擲出 JMSRuntimeException。
- javax.jms.Session.setMessageListener(javax.jms.MessageListener) 一律會擲出 JMSException。
- javax.jms.Session.getMessageListener() 一律會擲出 JMSException。

簡化的 API 限制

- javax.jms.JMSContext.createContext(int) 一律會擲出 JMSRuntimeException。
- javax.jms.JMSContext.setClientID(String) 一律會擲出 JMSRuntimeException。
- javax.jms.JMSContext.setExceptionListener(javax.jms.ExceptionListener) 一律會擲出 JMSRuntimeException。
- javax.jms.JMSContext.stop() 一律會擲出 JMSRuntimeException。
- javax.jms.JMSProducer.setAsync(javax.jms.CompletionListener) 一律會擲出 JMSRuntimeException。

使用 IBM MQ classes for Java

在 Java 環境中使用 IBM MQ 。 IBM MQ classes for Java 可讓 Java 應用程式以 IBM MQ 用戶端身分連接至 IBM MQ ，或直接連接至 IBM MQ 佇列管理程式。

註:

Stabilized IBM 將不會進一步加強 IBM MQ classes for Java，而且其功能層次將穩定為 IBM MQ 8.0 隨附的層次。 繼續完全支援使用 IBM MQ classes for Java 的現有應用程式，但不會新增特性，且會拒絕加強功能要求。 完全支援表示問題報告修正將與「IBM MQ 系統需求」變更而引發的任何必要變更一併進行。

IMS 中不支援 IBM MQ classes for Java 。

WebSphere Liberty 中不支援 IBM MQ classes for Java 。 它們不得與 IBM MQ Liberty 傳訊特性或一般 JCA 支援一起使用。 如需相關資訊，請參閱 在 J2EE/JEE 環境中使用 WebSphere MQ Java 介面。

IBM MQ classes for Java 是 Java 應用程式可用來存取 IBM MQ 資源的三個替代 API 之一。 其他 API 如下:

- JM 3.0 V 9.3.0 V 9.3.0 IBM MQ classes for Jakarta Messaging
- JMS 2.0 IBM MQ classes for JMS

如需相關資訊，請參閱 第 72 頁的『從 Java 存取 IBM MQ -API 的選項』。

從 IBM MQ 9.3 開始， IBM MQ classes for Java 是使用 Java 8 來建置。 Java 8 執行時期環境支援執行舊版類別檔。

IBM MQ classes for Java 會封裝「訊息佇列介面 (MQI)」(原生 IBM MQ API) ， 並使用與 IBM MQC++ 及 .NET 介面類似的物件模型。

可程式化選項可讓 IBM MQ classes for Java 以下列其中一種方式連接至 IBM MQ :

- 在 用戶端模式 中，使用傳輸控制通訊協定/Internet Protocol (TCP/IP) 作為 IBM MQ MQI client
- 在 連結模式中，使用 Java 原生介面 (JNI) 直接連接至 IBM MQ

註: IBM MQ classes for Java 不支援自動用戶端重新連線。

用戶端模式連線

IBM MQ classes for Java 應用程式可以使用用戶端模式來連接至任何支援的佇列管理程式。

若要以用戶端模式連接至佇列管理程式， IBM MQ classes for Java 應用程式可以在執行佇列管理程式的相同系統上執行，也可以在不同的系統上執行。 在每一個情況下，「 IBM MQ classes for Java 」都會透過 TCP/IP 連接至佇列管理程式。

如需如何撰寫應用程式以使用用戶端模式連線的相關資訊，請參閱 第 313 頁的『IBM MQ classes for Java 連線模式』。

連結模式連線

在連結模式中使用時， IBM MQ classes for Java 會使用 Java 原生介面 (JNI) 直接呼叫現有的佇列管理程式 API ，而不是透過網路進行通訊。 在大部分環境中，透過避免 TCP/IP 通訊成本，以連結模式連接會為 IBM MQ classes for Java 應用程式提供比以用戶端模式連接更好的效能。

使用「 IBM MQ classes for Java 」以連結模式連接的應用程式，必須在與其所連接的佇列管理程式相同的系統上執行。

用來執行 IBM MQ classes for Java 應用程式的「 Java 執行時期環境」必須配置成載入 IBM MQ classes for Java 程式庫; 如需進一步資訊，請參閱 第 300 頁的『IBM MQ classes for Java 位於檔案庫中』 。

如需如何撰寫應用程式以使用連結模式連線的相關資訊，請參閱 第 313 頁的『IBM MQ classes for Java 連線模式』。

相關概念

IBM MQ Java 語言介面

第 68 頁的『使用 IBM MQ classes for JMS/Jakarta Messaging』

IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 是 IBM MQ 隨附的 Java 傳訊提供者。 除了實作 JMS 和 Jakarta Messaging 規格中定義的介面，這些傳訊提供者也會將兩組延伸新增至 Java 傳訊 API。

相關工作

追蹤 IBM MQ classes for Java 應用程式

疑難排解 Java 和 JMS 問題

為何應該使用 IBM MQ classes for Java?

Java 應用程式可以使用 IBM MQ classes for Java 或 IBM MQ classes for JMS 來存取 IBM MQ 資源。

註: V 9.3.0 V 9.3.0 雖然仍然完全支援使用 IBM MQ classes for Java 的現有應用程式，但新的應用程式應該使用 IBM MQ classes for Jakarta Messaging。 最近新增至 IBM MQ 的特性 (例如非同步使用及自動重新連線) 在 IBM MQ classes for Java 中無法使用，但在 IBM MQ classes for JMS 及 IBM MQ classes for Jakarta Messaging 中可以使用。 如需相關資訊，請參閱第 70 頁的『為何應該使用 IBM MQ classes for JMS?』及第 69 頁的『為何應該使用 IBM MQ classes for Jakarta Messaging?』。

註:

Stabilized IBM 將不會進一步加強 IBM MQ classes for Java，而且其功能層次將穩定為 IBM MQ 8.0 隨附的層次。 繼續完全支援使用 IBM MQ classes for Java 的現有應用程式，但不會新增特性，且會拒絕加強功能要求。 完全支援表示問題報告修正將與「IBM MQ 系統需求」變更而引發的任何必要變更一併進行。

IMS 中不支援 IBM MQ classes for Java 。

WebSphere Liberty 中不支援 IBM MQ classes for Java 。 它們不得與 IBM MQ Liberty 傳訊特性或一般 JCA 支援一起使用。 如需相關資訊，請參閱 在 J2EE/JEE 環境中使用 WebSphere MQ Java 介面。

相關概念

第 72 頁的『從 Java 存取 IBM MQ -API 的選項』
IBM MQ 提供三個 Java 語言介面。

IBM MQ classes for Java 的必備項目

若要使用 IBM MQ classes for Java，您需要某些其他軟體產品。

如需 IBM MQ classes for Java 必備項目的相關資訊，請參閱 IBM MQ 的系統需求 網頁。

若要開發 IBM MQ classes for Java 應用程式，您需要 Java Development Kit (JDK)。 您可以在 IBM MQ 的系統需求 資訊中找到作業系統所支援之 JDK 的詳細資料。

若要執行 IBM MQ classes for Java 應用程式，您需要下列軟體元件:

- IBM MQ 佇列管理程式，適用於連接至佇列管理程式的應用程式
- Java 執行時期環境 (JRE) ，適用於您執行應用程式的每一個系統。 IBM MQ 隨附適當的 JRE。
- IBM i 若為 IBM i， QShell ，這是作業系統的選項 30
- z/OS 若為 z/OS， z/OS UNIX System Services (z/OS UNIX)

如果您需要 TLS 連線才能使用已通過 FIPS 140-2 認證的加密模組，則需要 IBM Java JSSE FIPS 提供者 (IBMJSSEFIPS)。 每一個 1.4.2 版或更新版本的 IBM JDK 和 JRE 都包含 IBMJSSEFIPS。

您可以在 Java 虛擬機器 (JVM) 及作業系統上的 TCP/IP 實作所支援的 IBM MQ classes for Java 應用程式 如果 IPv6 是 中使用 Internet Protocol 第 6 版 (IPv6) 位址。

在 Java EE 內執行 IBM MQ classes for Java 應用程式

在 Java EE 中使用 IBM MQ classes for Java 之前，必須考量某些限制和設計考量。

在 Java Platform, Enterprise Edition (Java EE) 環境內使用時， IBM MQ classes for Java 有一些限制。 在設計、實作及管理在 Java EE 環境內執行的 IBM MQ classes for Java 應用程式時，也必須考量其他事項。 下列各節概述這些限制和考量。

JTA 交易限制

對於使用 IBM MQ classes for Java 的應用程式，唯一支援的交易管理程式是 IBM MQ 本身。 雖然 JTA 控制下的應用程式可以使用 IBM MQ classes for Java，但透過這些類別執行的任何工作都不受 JTA 工作單元控制。 相反地，它們會透過 JTA 介面，與應用程式伺服器所管理的工作單元分開形成本端工作單元。 尤其是 JTA 交易的任何回復都不會導致回復任何已傳送或已接收的訊息。 這項限制適用於應用程式或 Bean 管理的

交易，以及儲存器管理的交易和所有 Java EE 儲存器。 如果要直接使用應用程式伺服器協調交易內的 IBM MQ 來執行傳訊工作，必須改用 IBM MQ classes for JMS 。

建立執行緒

IBM MQ classes for Java 會在內部為各種作業建立執行緒。 例如，以 BINDINGS 模式執行以直接在本端佇列管理程式上呼叫時，會在 IBM MQ classes for Java 內部建立的 'worker' 執行緒上進行呼叫。 可以在內部建立其他執行緒，例如從連線儲存區清除未用的連線，或移除已終止發佈/訂閱應用程式的訂閱。

部分 Java EE 應用程式 (例如在 EJB 和 Web 儲存器中執行的應用程式) 不得建立新的執行緒。 而是必須在應用程式伺服器所管理的主要應用程式執行緒上執行所有工作。 當應用程式使用 IBM MQ classes for Java 時，應用程式伺服器可能無法區分應用程式碼與 IBM MQ classes for Java 程式碼，因此先前說明的執行緒會導致應用程式不符合儲存器規格。 IBM MQ classes for JMS 不會岔斷這些 Java EE 規格，因此可以改用。

安全限制

應用程式伺服器所實作的安全原則可能會阻止 IBM MQ classes for Java API 所執行的某些作業，例如建立及操作新的控制執行緒 (如前述各節所述)。

例如，依預設，應用程式伺服器通常在停用「Java 安全」的情況下執行，並容許透過部分應用程式伺服器特定的配置來啟用它 (部分應用程式伺服器也容許對 Java Security 內使用的原則進行更詳細的配置)。 當啟用 Java Security 時， IBM MQ classes for Java 可能會岔斷定義給應用程式伺服器的 Java Security 原則執行緒作業規則，且 API 可能無法建立它運作所需的所有執行緒。 為了防止執行緒管理發生問題，在啟用「Java 安全」的環境中不支援使用 IBM MQ classes for Java 。

應用程式隔離考量

在 Java EE 環境內執行應用程式的預期好處是應用程式隔離。 IBM MQ classes for Java 的設計及實作早於 Java EE 環境。 IBM MQ classes for Java 的使用方式不支援應用程式隔離的概念。 這方面的具體考慮例子包括:

- 在 MQEnvironment 類別內使用靜態 (JVM 處理程序層面) 設定，例如:
  - 要用於連線識別及鑑別的使用者 ID 及密碼
  - 用於用戶端連線的主機名稱、埠及通道
  - 安全用戶端連線的 TLS 配置

  為了一個應用程式的好處而修改任何 MQEnvironment 內容，也會影響使用相同內容的其他應用程式。 在多重應用程式環境 (例如 Java EE) 中執行時，每一個應用程式都必須透過建立具有一組特定內容的 MQQueueManager 物件來使用自己的獨特配置，而不是預設為處理程序層面 MQEnvironment 類別中所配置的內容。
- MQEnvironment 類別引進一些靜態方法，這些方法在相同 JVM 程序內使用 IBM MQ classes for Java 的所有應用程式上廣域運作，無法針對特定應用程式置換此行為。 範例包括:
  - 配置 TLS 內容，例如金鑰儲存庫的位置
  - 配置用戶端通道結束程式
  - 啟用或停用診斷追蹤
  - 管理用來最佳化佇列管理程式連線使用的預設連線儲存區

  呼叫這類方法會影響在相同 Java EE 環境中執行的所有應用程式。
- 啟用連線儲存區以最佳化對相同佇列管理程式建立多個連線的處理程序。 預設連線儲存區管理程式是整個處理程序，並由多個應用程式共用。 連線儲存區配置的變更，例如使用 MQEnvironment.setDefaultConnectionManager() 方法取代某個應用程式的預設連線管理程式，因此會影響在相同 Java EE 應用程式伺服器中執行的其他應用程式。
- 使用 MQEnvironment 類別及 MQQueueManager 物件內容，為使用 IBM MQ classes for Java 的應用程式配置 TLS。 它未與應用程式伺服器本身的受管理安全配置整合。 您必須確保適當地配置 IBM MQ classes for Java ，以提供所需的安全層次，並且不使用應用程式伺服器配置。

連結模式限制

IBM MQ 和 WebSphere Application Server 可以安裝在相同機器上，以便 WebSphere Application Server 中所提供的佇列管理程式和 IBM MQ 資源配接器 (RA) 的主要版本不同。

如果佇列管理程式和資源配接器主要版本不同，則無法使用連結連線。 從 WebSphere Application Server 到使用資源配接器的佇列管理程式的任何連線都必須使用用戶端類型連線。 如果版本相同，則可以使用連結連線。

IBM MQ classes for Java 中的字串轉換

IBM MQ classes for Java 直接使用 CharsetEncoders 和 CharsetDecoders 來進行字串轉換。 字串轉換的預設行為可以配置兩個系統內容。 包含無法對映之字元的訊息處理可以透過 `com.ibm.mq.MQMD` 來配置。

在 IBM MQ 8.0 之前， IBM MQ classes for Java 中的字串轉換是透過呼叫 `java.nio.charset.Charset.decode(ByteBuffer)` 和 `Charset.encode(CharBuffer)` 方法來完成。

使用上述任一方法會導致預設取代 (REPLACE) 形態異常或無法翻譯的資料。 此行為可能會遮蔽應用程式中的錯誤，並在已翻譯的資料中導致非預期的字元 (例如 ?)。

從 IBM MQ 8.0 開始，為了更早且更有效地偵測這類問題， IBM MQ classes for Java 會直接使用 CharsetEncoders 和 CharsetDecoders ， 並明確地配置處理形態異常和不可翻譯的資料。 預設行為是 REPORT 擲出適當的 `MQException` 來發出這類問題。

配置

從 UTF-16 (Java 中使用的字元表示法) 轉換為原生字集 (例如 UTF-8) ， 稱為 編碼，而反向轉換則稱為 解碼。

解碼採用 `CharsetDecoders` 的預設行為，透過擲出異常狀況來報告錯誤。

其中一項設定是用來指定 `java.nio.charset.CodingErrorAction` ， 以控制編碼和解碼兩者的錯誤處理。 在編碼時，會使用另一個設定來控制取代位元組。 解碼作業將使用預設 Java 取代字串。

IBM MQ classes for Java 中不可翻譯資料處理的配置

從 IBM MQ 8.0 開始， `com.ibm.mq.MQMD` 包括下列兩個欄位:

byte [] unMappableReplacement
如果輸入字元無法轉換，且您已指定 REPLACE，將寫入編碼字串的位元組順序。

預設值: "?"。getBytes()
解碼作業會使用預設 Java 取代字串。

java.nio.charset.CodingErrorAction unmappableAction
指定要對編碼和解碼中無法翻譯的資料採取的動作:

預設值: CodingErrorAction.REPORT;

設定系統預設值的系統內容

從 IBM MQ 8.0 開始， 下列兩個 Java 系統內容可用來配置有關字串轉換的預設行為。

com.ibm.mq.cfg.jmqi.UnmappableCharacterAction
指定要對編碼和解碼的不可翻譯資料採取的動作。 值可以是 REPORT、 REPLACE 或 IGNORE。

com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement
在編碼作業中無法對映字元時，設定或取得要套用的取代位元組。在解碼作業中使用預設 Java 取代字串。

為了避免 Java 字元與原生位元組表示法之間的混淆，您應該指定 `com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement` 作為十進位數，代表原生字集中的取代位元組。

例如，如果原生字集是 ASCII 型 (例如 ISO-8859-1)，則 ?作為原生位元組的十進位值為 63，而如果原生字集是 EBCDIC，則為 111。

註: 請注意，如果 MQMD 或 MQMessage 物件已設定 **unmappableAction** 或 **unMappableReplacement** 欄位，則這些欄位的值優先於 Java 系統內容。 必要的話，這容許針對每一個訊息置換 Java 系統內容指定的值。

相關概念

第 117 頁的『IBM MQ classes for JMS 中的字串轉換』
IBM MQ classes for JMS 直接使用 CharsetEncoders 和 CharsetDecoders 來進行字串轉換。 字串轉換的預設行為可以配置兩個系統內容。 包含不可對映字元的訊息處理可以透過訊息內容來配置，以設定 UnmappableCharacter 動作及取代位元組。

安裝與配置 IBM MQ classes for Java

本節說明安裝 IBM MQ classes for Java 時所建立的目錄和檔案，並告訴您如何在安裝之後配置 IBM MQ classes for Java 。

針對 *IBM MQ classes for Java* 安裝的內容

最新版本的 IBM MQ classes for Java 會隨 IBM MQ 一起安裝。 您可能需要置換預設安裝選項，以確保完成此作業。

如需安裝 IBM MQ 的相關資訊，請參閱:

- Multi 安裝 IBM MQ
- z/OS 安裝 IBM MQ for z/OS 產品

IBM MQ classes for Java 包含在 Java 保存檔 (JAR)、 `com.ibm.mq.jar` 和 `com.ibm.mq.jmqi.jar` 中。

標準訊息標頭 (例如「可程式化指令格式 (PCF)」) 的支援包含在 JAR 檔 `com.ibm.mq.headers.jar` 中。

「可程式化指令格式 (PCF)」的支援包含在 JAR 檔 `com.ibm.mq.pcf.jar` 中。

註: 不建議在應用程式伺服器內使用 IBM MQ classes for Java 。 如需在此環境中執行時適用之限制的相關資訊，請參閱 第 292 頁的『在 Java EE 內執行 IBM MQ classes for Java 應用程式』。 如需相關資訊，請參閱在 J2EE/JEE 環境中使用 WebSphere MQ Java 介面。

重要: 除了 第 295 頁的『IBM MQ classes for Java 可重新定位 JAR 檔』中說明的可再定位 JAR 檔之外，不支援將 IBM MQ classes for Java JAR 檔或原生程式庫複製到其他機器，或複製到已安裝 IBM MQ classes for Java 之機器上的不同位置。

IBM MQ classes for Java 可重新定位 *JAR* 檔

可再定位 JAR 檔可以移至需要執行 IBM MQ classes for Java 的系統。

重要:

- 除了 可再定位 JAR 檔中說明的可再定位 JAR 檔之外，不支援將 IBM MQ classes for Java JAR 檔或原生程式庫複製到其他機器，或複製到已安裝 IBM MQ classes for Java 的機器上的不同位置。
- 為了避免類別載入器衝突，不建議將可重新定位的 JAR 檔組合在相同 Java 執行時期內的多個應用程式內。 在此實務範例中，請考量在 Java 執行時期的類別路徑上提供 IBM MQ 可重新配置 JAR 檔。
- 在部署至 Java EE 應用程式伺服器 (例如 WebSphere Application Server) 的應用程式內，請勿包含可重新定位的 JAR 檔。 在這些環境中，應該部署並改用 IBM MQ 資源配接器，因為這包含 IBM MQ classes for Java。 請注意， WebSphere Application Server 會內嵌 IBM MQ 資源配接器，因此不需要手動將它部署到這個環境中。 此外， WebSphere Liberty 中不支援 IBM MQ classes for Java 。 如需相關資訊，請參閱第 370 頁的『Liberty 和 IBM MQ 資源配接器』。
- 如果您要組合應用程式內的可再定位 JAR 檔，請確保包含 可再定位 JAR 檔中說明的所有必備 JAR 檔。 在應用程式維護過程中，您也應該確保您具有適當的程序來更新組合 JAR 檔，以確保 IBM MQ classes for Java 保持最新，並重新調解已知問題。

可再定位 JAR 檔

在企業內，可以將下列檔案移至需要執行 IBM MQ classes for Java 應用程式的系統:

- JMS 2.0 com.ibm.mq.allclient.jar 第 296 頁的『1』
- JM 3.0 V 9.3.0 V 9.3.0 com.ibm.mq.jakarta.client.jar 第 296 頁的『2』
- V 9.3.3 Removed com.ibm.mq.traceControl.jar
- V 9.3.5 bcpkix-jdk18on.jar 第 296 頁的『3』
- V 9.3.0 V 9.3.0 bcpkix-jdk15to18.jar 第 296 頁的『4』
- V 9.3.5 bcprov-jdk18on.jar 第 296 頁的『3』
- V 9.3.0 V 9.3.0 bcprov-jdk15to18.jar 第 296 頁的『4』
- V 9.3.5 bcutil-jdk18on.jar 第 296 頁的『3』
- V 9.3.0 V 9.3.0 bcutil-jdk15to18.jar 第 296 頁的『4』
- V 9.3.3 jackson-annotations.jar
- V 9.3.3 jackson-core.jar
- V 9.3.3 jackson-databind.jar
- org.json.jar

附註:

1. JMS 2.0 及 JMS 1.1
2. Jakarta Messaging 3.0
3. Continuous Delivery from IBM MQ 9.3.5
4. Long Term Support 及 Continuous Delivery 之前 IBM MQ 9.3.5

Bouncy Castle 安全提供者和 CMS 支援 JAR 檔

需要 Bouncy Castle 安全提供者和 CMS 支援 JAR 檔。 如需相關資訊，請參閱 支援非 IBM 具備 AMS 的 JRE。

V 9.3.5 對於 IBM MQ 9.3.5 中的 Continuous Delivery，需要下列 JAR 檔:

- bcpkix-jdk18on.jar
- bcprov-jdk18on.jar
- bcutil-jdk18on.jar

LTS 對於 Long Term Support 和 IBM MQ 9.3.5 之前的 Continuous Delivery，需要下列 JAR 檔:

- bcpkix-jdk15to18.jar
- bcprov-jdk15to18.jar
- bcutil-jdk15to18.jar

org.json.jar

如果您的 IBM MQ classes for Java 應用程式使用 JSON 格式的 CCDT，則需要 org.json.jar 檔案。

com.ibm.mq.allclient.jar 和 com.ibm.mq.jakarta.client.jar

檔案 com.ibm.mq.allclient.jar 和 com.ibm.mq.jakarta.client.jar 包含 IBM MQ classes for JMS、 IBM MQ classes for Java 以及 PCF 和標頭類別。 如果您將這些檔案移至新位置，請確保採取步驟以使用新的 IBM MQ 修正套件來維護此新位置。 此外，如果您取得臨時修正程式，請確定「IBM 支援中心」已知道這些檔案的使用。

若要判定 com.ibm.mq.allclient.jar 檔案或 com.ibm.mq.jakarta.client.jar 檔案的版本，請使用下列指令:

JM 3.0 V 9.3.0 V 9.3.0

```
java -jar com.ibm.mq.jakarta.client.jar
```

JMS 2.0

```
java -jar com.ibm.mq.allclient.jar
```

下列範例顯示此指令的部分範例輸出:

```
C:\Program Files\IBM\MQ_1\java\lib>java -jar com.ibm.mq.allclient.jar
Name:       Java Message Service Client
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       IBM MQ classes for Java Message Service
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       IBM MQ JMS Provider
Version:    9.3.0.0
Level:      p000-L140428.1 mqjbnd=p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       Common Services for Java Platform, Standard Edition
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar
```

jackson-annotations.jar、 jackson-core.jar 和 jackson-databind.jar

V 9.3.3

如果您的 IBM MQ classes for Java 應用程式建立與佇列管理程式的安全 TLS 連線，則需要這三個 Jackson JAR 檔。

IBM MQ classes for Java 的安裝目錄
IBM MQ classes for Java 檔案和範例根據平台安裝在不同位置。 隨 IBM MQ 一起安裝的「Java 執行時期環境 (JRE)」位置也會隨著平台而不同。

IBM MQ classes for Java 檔案的安裝目錄

第 297 頁的表 49 顯示 IBM MQ classes for Java 檔案的安裝位置。

表 *49: IBM MQ classes for Java* 安裝目錄

平台	目錄
AIX AIX	*MQ_INSTALLATION_PATH*/java/lib
	/QIBM/ProdData/mqm/java/lib

表 49: IBM MQ classes for Java 安裝目錄 (繼續)	
平台	**目錄**
Linux Linux	*MQ_INSTALLATION_PATH*/java/lib
Windows Windows	*MQ_INSTALLATION_PATH*\java\lib
z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java /lib

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

範例的安裝目錄

部分範例應用程式 (例如「安裝驗證程式 (IVP)」) 隨附於 IBM MQ。 第 298 頁的表 50 顯示範例應用程式的安裝位置。 IBM MQ classes for Java 範例位於稱為 `wmqjava` 的子目錄中。 PCF 範例位於稱為 `pcf` 的子目錄中。

表 50: 範例目錄	
平台	**目錄**
AIX AIX	*MQ_INSTALLATION_PATH*/samp/wmqjava/
IBM i IBM i	/QIBM/ProdData/mqm/java/samples
Linux Linux	*MQ_INSTALLATION_PATH*/samp/wmqjava/
Windows Windows	*MQ_INSTALLATION_PATH*\tools\wmqjava\
z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/samples

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

JRE 的安裝目錄

IBM MQ classes for JMS 需要 Java 7 (或更新版本) Java 執行時期環境 (JRE)。 適當的 JRE 隨 IBM MQ 一起安裝。 第 298 頁的表 51 顯示此 JRE 的安裝位置。 若要執行 Java 程式 (例如提供的範例)，請使用此 JRE，明確地呼叫 *JRE_LOCATION*`/bin/java` 或將 *JRE_LOCATION*`/bin` 新增至平台的 PATH 環境 (或對等項目)，其中 *JRE_LOCATION* 是 第 298 頁的表 51 中提供的目錄。

表 51: JRE 目錄	
平台	**目錄**
AIX AIX	*MQ_INSTALLATION_PATH*/java/jre
IBM i IBM i	/QIBM/ProdData/mqm/java/jre
Linux Linux	*MQ_INSTALLATION_PATH*/java/jre
Windows Windows	*MQ_INSTALLATION_PATH*\java\jre
z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/jre

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

與 *IBM MQ classes for Java* 相關的環境變數

如果您想要執行 IBM MQ classes for Java 應用程式，其類別路徑必須包含 IBM MQ classes for Java 和 samples 目錄。

如果要執行 IBM MQ classes for Java 應用程式，其類別路徑必須包含適當的 IBM MQ classes for Java 目錄。 如果要執行範例應用程式，類別路徑也必須包含適當的範例目錄。 此資訊可以在 Java 呼叫指令或 **CLASSPATH** 環境變數中提供。

重要: 不支援將 Java 選項 `-Xbootclasspath` 設為包含 IBM MQ classes for Java 。

第 299 頁的表 52 顯示在每一個平台上用來執行 IBM MQ classes for Java 應用程式的適當 **CLASSPATH** 設定，包括範例應用程式。

表 *52:* ***CLASSPATH*** 設定，以執行 *IBM MQ classes for Java* 應用程式，包括 *IBM MQ classes for Java* 範例應用程式

平台	**CLASSPATH** 設定
AIX AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
IBM i IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: /QIBM/ProdData/mqm/java/samples/wmqjava/samples:
Linux Linux	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
Windows Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\Java\lib\com.ibm.mq.jar; *MQ_INSTALLATION_PATH*\tools\wmqjava\samples;
z/OS z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/mqm/V9R3M0/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/mqm/V9R3M0/java/samples/wmqjava: *MQ_INSTALLATION_PATH*/mqm/V9R3M0/java/samples/pcf

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

如果您使用 `-Xlint` 選項編譯，可能會看到一則訊息警告您 `com.ibm.mq.ese.jar` 不存在。 您可以忽略警告。 只有在您已安裝 Advanced Message Security 時，才會呈現這個檔案。

IBM MQ classes for JMS 隨附的 Script 使用下列環境變數:

MQ_JAVA_DATA_PATH
此環境變數指定日誌和追蹤輸出的目錄。

MQ_JAVA_INSTALL_PATH
此環境變數指定 IBM MQ classes for Java 的安裝目錄，如 IBM MQ classes for Java 安裝目錄中所示。

MQ_JAVA_LIB_PATH
此環境變數指定儲存 IBM MQ classes for Java 程式庫的目錄，如 每一個平台的 IBM MQ classes for Java 程式庫位置中所示。 IBM MQ classes for Java 隨附的部分 Script (例如 IVTRun) 會使用此環境變數。

Windows 在 Windows 上，安裝期間會自動設定所有環境變數。

Linux AIX 在 AIX and Linux 上，您可以使用 Script `setjmsenv` (如果您使用 32 位元 JVM) 或 `setjmsenv64` (如果您使用 64 位元 JVM) 來設定環境變數。 這些 Script 位於 *MQ_INSTALLATION_PATH*/java/bin 目錄中。

IBM i 在 IBM i 上，環境變數 **QIBM_MULTI_THREADED** 必須設為 Y。 然後，您可以使用與執行單一執行緒應用程式相同的方式來執行多執行緒應用程式。 如需相關資訊，請參閱 使用 Java 和 JMS 設定 IBM MQ。

IBM MQ classes for Java 需要 Java 7 Java 執行時期環境 (JRE)。 如需隨 IBM MQ 一起安裝之適當 JRE 的位置相關資訊，請參閱 第 297 頁的『IBM MQ classes for Java 的安裝目錄』。

IBM MQ classes for Java 位於檔案庫中
IBM MQ classes for Java 程式庫的位置會因平台而異。 當您啟動應用程式時，請指定此位置。

若要指定 Java 原生介面 (JNI) 程式庫的位置，請使用具有下列格式的 **java** 指令來啟動應用程式:

```
java -Djava.library.path= library_path application_name
```

其中 *library_path* 是 IBM MQ classes for Java 的路徑，包括 JNI 程式庫。 第 300 頁的表 53 顯示每一個平台的 IBM MQ classes for Java 程式庫位置。 在此表格中， *MQ_INSTALLATION_PATH* 代表 IBM MQ 安裝所在的高階目錄。

表 *53:* 每一個平台的 *IBM MQ classes for Java* 程式庫位置

平台	**包含 IBM MQ classes for Java 程式庫的目錄**
AIX	*MQ_INSTALLATION_PATH*/java/lib (32 位元程式庫) *MQ_INSTALLATION_PATH*/java/lib64 (64 位元程式庫)
Linux (x86 平台)	*MQ_INSTALLATION_PATH*/java/lib
Linux (POWER、 x86-64 及 zSeries s390x 平台)	*MQ_INSTALLATION_PATH*/java/lib (32 位元程式庫) *MQ_INSTALLATION_PATH*/java/lib64 (64 位元程式庫)
Windows	*MQ_INSTALLATION_PATH*\Java\lib (32 位元程式庫) *MQ_INSTALLATION_PATH*\Java\lib64 (64 位元程式庫)
z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/lib (32 位元及 64 位元檔案庫)

註:

1. Linux AIX 在 AIX 或 Linux (Power 平台) 上，使用 32 位元程式庫或 64 位元程式庫。 只有在 64 位元平台上的 64 位元 Java 虛擬機器 (JVM) 中執行應用程式時，才使用 64 位元程式庫。 否則，請使用 32 位元程式庫。
2. Windows 在 Windows 上，您可以使用 PATH 環境變數來指定 IBM MQ classes for Java 程式庫的位置，而不是在 **java** 指令上指定它們的位置。
3. IBM i 若要在 IBM i 上以連結模式使用 IBM MQ classes for Java ，請確保程式庫 QMQMJAVA 位於程式庫清單中。
4. z/OS 在 z/OS 上，您可以使用 32 位元或 64 位元 Java 虛擬機器 (JVM)。 您不需要指定要使用哪些程式庫; IBM MQ classes for Java 可以自行決定要載入哪些 JNI 程式庫。

相關概念

使用 IBM MQ classes for Java
安裝 IBM MQ classes for Java 之後，您可以配置安裝，以便執行您自己的應用程式。

支援 *OSGi* 與 *IBM MQ classes for Java*
OSGi 提供一個架構，可支援將應用程式部署成軟體組。 在 IBM MQ classes for Java 中提供了三個 OSGi 軟體組。

OSGi 提供一般用途、安全及受管理的 Java 架構，支援部署以軟體組形式提供的應用程式。 OSGi 相容裝置可以下載並安裝軟體組，並在不再需要時移除軟體組。 架構會以動態及可調式方式來管理軟體組的安裝及更新。

IBM MQ classes for Java 包含下列 OSGi 軟體組。

com.ibm.mq.osgi.java_*version_number*.jar
容許應用程式使用 IBM MQ classes for Java 的 JAR 檔。

com.ibm.mq.jakarta.osgi.allclient_*version_number*.jar
JM 3.0 V 9.3.0 V 9.3.0 對於 Jakarta Messaging 3.0，這個 JAR 檔可讓應用程式同時使用 IBM MQ classes for JMS 和 IBM MQ classes for Java，也包含用來處理 PCF 訊息的程式碼。

com.ibm.mq.osgi.allclient_*version_number*.jar
JMS 2.0 對於 JMS 2.0，這個 JAR 檔可讓應用程式同時使用 IBM MQ classes for JMS 和 IBM MQ classes for Java，也包含用來處理 PCF 訊息的程式碼。

com.ibm.mq.jakarta.osgi.allclientprereqs_*version_number*.jar
JM 3.0 V 9.3.0 V 9.3.0 對於 Jakarta Messaging 3.0，此 JAR 檔提供 `com.ibm.mq.jakarta.osgi.allclient_version_number.jar` 的必要條件。

com.ibm.mq.osgi.allclientprereqs_*version_number*.jar
JMS 2.0 對於 JMS 2.0，此 JAR 檔提供 `com.ibm.mq.osgi.allclient_version_number.jar` 的必要條件。

其中 *version_number* 是所安裝 IBM MQ 的版本號碼。

軟體組會安裝至 IBM MQ 安裝架構的 `java/lib/OSGi` 子目錄，或 Windows 上的 `java\lib\OSGi` 資料夾。

從 IBM MQ 8.0 中，針對任何新的應用程式使用組合 `com.ibm.mq.osgi.allclient_8.0.0.0.jar` 及 `com.ibm.mq.osgi.allclientprereqs_8.0.0.0.jar`。 使用這些軟體組會移除無法在相同 OSGi 架構內同時執行 IBM MQ classes for JMS 和 IBM MQ classes for Java 的限制。 不過，所有其他限制仍然適用。 對於 IBM MQ 8.0 之前的 IBM MQ 版本，適用使用 IBM MQ classes for JMS 或 IBM MQ classes for Java 的限制。

其他九個軟體組也會安裝到 IBM MQ 安裝架構的 `java/lib/OSGi` 子目錄中，或 Windows 上的 `java\lib\OSGi` 資料夾中。 這些軟體組是 IBM MQ classes for JMS 的一部分，不得載入至已載入 IBM MQ classes for Java 軟體組的 OSGi 執行時期環境。 如果 IBM MQ classes for Java OSGi 軟體組載入 OSGi 執行時期環境中，且也載入 IBM MQ classes for JMS 軟體組，當執行使用 IBM MQ classes for Java 軟體組或 IBM MQ classes for JMS 軟體組的應用程式時，會發生下列範例所示的錯誤:

```
java.lang.ClassCastException: com.ibm.mq.MQException incompatible with com.ibm.mq.MQException
```

IBM MQ classes for Java 的 OSGi 組合已寫入 OSGi 第 4 版規格; 它在 OSGi 第 3 版環境中無法運作。

您必須正確設定系統路徑或程式庫路徑， OSGi 執行時期環境才能找到任何必要的 DLL 檔案或共用程式庫。

如果您將 OSGi 軟體組用於 IBM MQ classes for Java，則不支援在 Java 中撰寫的通道結束程式類別，因為在多類別載入器環境 (例如 OSGi) 中載入類別時會發生固有問題。 使用者軟體組可以知道 IBM MQ classes for Java 軟體組，但 IBM MQ classes for Java 軟體組不知道任何使用者軟體組。 因此， IBM MQ classes for Java 組合中使用的類別載入器無法載入使用者組合中的通道結束程式類別。

如需 OSGi 的相關資訊，請參閱 OSGi 聯盟 網站。

z/OS 安裝 *IBM MQ classes for Java on z/OS*
在 z/OS 上，執行時期使用的 STEPLIB 必須包含 IBM MQ SCSQAUTH 和 SCSQANLE 程式庫。

從 z/OS UNIX System Services 開始，您可以使用 `.profile` 中的一行來新增這些程式庫，如下列範例所示，將 `thlqual` 取代為您在安裝 IBM MQ 時選擇的高階資料集限定元:

```
export STEPLIB=thlqual.SCSQAUTH:thlqual.SCSQANLE:$STEPLIB
```

在其他環境中，您通常需要編輯啟動 JCL，以在 STEPLIB 連結上包含 SCSQAUTH:

```
STEPLIB DD DSN=thlqual.SCSQAUTH,DISP=SHR
        DD DSN=thlqual.SCSQANLE,DISP=SHR
```

IBM MQ classes for Java 配置檔
IBM MQ classes for Java 配置檔指定用來配置 IBM MQ classes for Java 的內容。

IBM MQ classes for Java 配置檔的格式是標準 Java 內容檔的格式。

IBM MQ classes for Java 安裝目錄的 `bin` 子目錄中提供範例配置檔 `mqjava.config`。 此檔案記載所有支援的內容及其預設值。

註: 當 IBM MQ 安裝升級至未來修正套件時，會改寫範例配置檔。 因此，建議您製作範例配置檔的副本，以與您的應用程式搭配使用。

您可以選擇 IBM MQ classes for Java 配置檔的名稱和位置。 當您啟動應用程式時，請使用具有下列格式的 **java** 指令:

```
java -Dcom.ibm.msg.client.config.location=config_file_url application_name
```

在指令中， *config_file_url* 是統一資源定址器 (URL)，指定 IBM MQ classes for Java 配置檔的名稱和位置。 支援下列類型的 URL: `http`、 `file`、 `ftp` 和 `jar`。

下列範例顯示 **java** 指令:

```
java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/mqjava.config MyAppClass
```

這個指令會將 IBM MQ classes for Java 配置檔識別為本端 Windows 系統上的 `D:\mydir\mqjava.config` 檔案。

當應用程式啟動時， IBM MQ classes for Java 會讀取配置檔的內容，並將指定的內容儲存在內部內容儲存庫中。 如果 **java** 指令未識別配置檔，或找不到配置檔，則 IBM MQ classes for Java 會對所有內容使用預設值。 必要的話，您可以將配置檔中的任何內容指定為 **java** 指令上的系統內容，以置換配置檔中的任何內容。

IBM MQ classes for Java 配置檔可與應用程式與佇列管理程式或分配管理系統之間的任何受支援傳輸搭配使用。

置換 IBM MQ MQI client 配置檔中指定的內容

IBM MQ MQI client 配置檔也可以指定用來配置 IBM MQ classes for Java 的內容。 不過，只有在應用程式以用戶端模式連接至佇列管理程式時， IBM MQ MQI client 配置檔中指定的內容才適用。

必要的話，您可以置換 IBM MQ MQI client 配置檔中的任何屬性，方法是將它指定為 IBM MQ classes for Java 配置檔中的內容。 若要置換 IBM MQ MQI client 配置檔中的屬性，請在 IBM MQ classes for Java 配置檔中使用具有下列格式的項目:

```
com.ibm.mq.cfg.stanza.propName=propValue
```

項目中的變數具有下列意義:

段落 (stanza)
IBM MQ MQI client 配置檔中包含屬性的段落名稱。

propName
在 IBM MQ MQI client 配置檔中指定的屬性名稱。

propValue
此內容的值會置換 IBM MQ MQI client 配置檔中指定的屬性值。

或者，您可以在 **java** 指令上指定內容作為系統內容，以置換 IBM MQ MQI client 配置檔中的屬性。 請使用前述格式，將內容指定為系統內容。

只有 IBM MQ MQI client 配置檔中的下列屬性與 IBM MQ classes for Java 相關。 如果您指定或置換其他屬性，則它沒有作用。 具體而言，請注意不會使用 用戶端配置檔的 CHANNELS 段落 中的 `ChannelDefinitionFile` 和 `ChannelDefinitionDirectory`。 如需如何搭配使用 CCDT 與 IBM MQ classes for Java 的詳細資料，請參閱 第 316 頁的『搭配使用用戶端通道定義表與 IBM MQ classes for Java』。

表 *54:* 用戶端配置檔的哪個段落包含哪個屬性

段落 (stanza)	**屬性**
用戶端配置檔的 CHANNELS 段落	Put1DefaultAlwaysSync
用戶端配置檔的 CHANNELS 段落	PasswordProtection
ClientExit 用戶端配置檔的路徑段落	ExitsDefaultPath
ClientExit 用戶端配置檔的路徑段落	ExitsDefaultPath64
ClientExit 用戶端配置檔的路徑段落	JavaExits 類別路徑
用戶端配置檔的 JMQI 段落	useMQCSPauthentication
用戶端配置檔的 MessageBuffer 段落	MaximumSize
用戶端配置檔的 MessageBuffer 段落	PurgeTime
用戶端配置檔的 MessageBuffer 段落	UpdatePercentage
用戶端配置檔的 TCP 段落	ClntRcvBuffSize
用戶端配置檔的 TCP 段落	ClntSndBuffSize
用戶端配置檔的 TCP 段落	連線逾時
用戶端配置檔的 TCP 段落	KeepAlive

如需 IBM MQ MQI client 配置的相關資訊，請參閱 IBM MQ MQI client 配置檔 `mqclient.ini`。

相關工作

追蹤 Java 應用程式的 IBM MQ 類別

使用 *Java* 標準環境追蹤來配置 *Java* 追蹤

使用「Java 標準環境追蹤設定」段落來配置 IBM MQ classes for Java 追蹤機能。

com.ibm.msg.client.commonservices.trace.outputName = *traceOutput* 名稱

traceOutputName 是追蹤輸出傳送至其中的目錄及檔名。

依預設，追蹤資訊會寫入應用程式現行工作目錄中的追蹤檔。 追蹤檔的名稱視應用程式執行所在的環境而定:

- 從 IBM MQ 9.1.5 和 IBM MQ 9.1.0 Fix Pack 5:
  - 如果應用程式已從可再定位 JAR 檔 `com.ibm.mq.allclient.jar` 載入 IBM MQ classes for Java，則會將追蹤寫入稱為 `mqjavaclient_%PID%.cl%u.trc` 的檔案。
  - 如果應用程式已從 JAR 檔 `com.ibm.mq.jar` 載入 IBM MQ classes for Java，追蹤會寫入稱為 `mqjava_%PID%.cl%u.trc` 的檔案。
- 從 IBM MQ 9.0.0 Fix Pack 2:
  - 如果應用程式已從可再定位 JAR 檔 `com.ibm.mq.allclient.jar` 載入 IBM MQ classes for Java，則會將追蹤寫入稱為 `mqjavaclient_%PID%.trc` 的檔案。
  - 如果應用程式已從 JAR 檔 `com.ibm.mq.jar` 載入 IBM MQ classes for Java，追蹤會寫入稱為 `mqjava_%PID%.trc` 的檔案。
- 若為 IBM MQ classes for Java for IBM MQ 9.0.0 Fix Pack 1 或更早版本，追蹤會寫入稱為 `mqjms_%PID%.trc` 的檔案。

其中 *%PID%* 是所追蹤應用程式的處理程序 ID，而 *%u* 是唯一數字，用來區分在不同 Java 類別載入器下執行追蹤的執行緒之間的檔案。

如果程序 ID 無法使用，則會產生一個亂數，並以字母 `f` 作為字首。 若要在您指定的檔名中包括程序 ID，請使用字串 `%PID%`。

如果您指定替代目錄，則它必須存在，且您必須具有此目錄的寫入權。 如果您沒有寫入權，追蹤輸出會寫入 `System.err`。

com.ibm.msg.client.commonservices.trace.include = *includeList*
includeList 是所追蹤套件及類別的清單，或特殊值 `ALL` 或 `NONE`。

以分號 `;` 區隔套件或類別名稱。 *includeList* 預設為 `ALL`，並追蹤 IBM MQ classes for Java 中的所有套件和類別。

註: 您可以包括套件，然後排除該套件的子套件。 例如，如果您包括套件 `a.b` 及排除套件 `a.b.x`，則追蹤會包括 `a.b.y` 及 `a.b.z` 中的所有項目，但不會包括 `a.b.x` 或 `a.b.x.1`。

com.ibm.msg.client.commonservices.trace.exclude = *excludeList*
excludeList 是未追蹤的套件和類別清單，或特殊值 `ALL` 或 `NONE`。

以分號 `;` 區隔套件或類別名稱。 *excludeList* 預設為 `NONE`，因此不會追蹤 IBM MQ classes for JMS 中的任何套件和類別。

註: 您可以排除套件，然後併入該套件的子套件。 比方說，如果您排除套件 `a.b` 並包含套件 `a.b.x`，則追蹤會包含 `a.b.x` 和 `a.b.x.1` 中的所有項目，但不會包含 `a.b.y` 或 `a.b.z`。

會併入在相同層次所指定的任何套件或類別 (已併入及已排除)。

com.ibm.msg.client.commonservices.trace.maxBytes = *maxArray* 位元組
maxArrayBytes 是從任何位元組陣列追蹤的位元組數上限。

如果 *maxArrayBytes* 設為正整數，它會限制位元組陣列中寫入追蹤檔的位元組數。 它會在寫出 *maxArrayBytes* 之後截斷位元組陣列。 設定 *maxArrayBytes* 會減少產生的追蹤檔大小，並減少追蹤對應用程式效能的影響。

此內容的值 `0` 表示任何位元組陣列的內容都不會傳送至追蹤檔。

預設值為 `-1`，它會移除對位元組陣列中傳送至追蹤檔的位元組數的任何限制。

com.ibm.msg.client.commonservices.trace.limit = *maxTrace* 位元組
maxTraceBytes 是寫入追蹤輸出檔的位元組數上限。

maxTraceBytes 與 *traceCycles* 搭配使用。 如果寫入的追蹤位元組數接近限制，則會關閉檔案，並啟動新的追蹤輸出檔。

值 `0` 表示追蹤輸出檔長度為零。 預設值為 `-1`，表示要寫入追蹤輸出檔的資料量無限制。

com.ibm.msg.client.commonservices.trace.count = *traceCycles*
traceCycles 是要輪流選取的追蹤輸出檔數目。

如果現行追蹤輸出檔達到 *maxTraceBytes* 指定的限制，則會關閉該檔案。 後續追蹤輸出會依序寫入至下一個追蹤輸出檔。 每一個追蹤輸出檔都以附加至檔名的數字字尾來識別。 現行或最新的追蹤輸出檔是 `mqjms.trc.0`，下一個最新的追蹤輸出檔是 `mqjms.trc.1`。 較舊的追蹤檔遵循相同的編號型樣，直到達到限制為止。

traceCycles 的預設值為 `1`。 如果 *traceCycles* 為 `1`，則當現行追蹤輸出檔達到其大小上限時，會關閉並刪除該檔案。 啟動同名的新追蹤輸出檔。 因此，一次只會有一個追蹤輸出檔。

com.ibm.msg.client.commonservices.trace.parameter = *traceParameters*
traceParameters 控制方法參數和回覆值是否包含在追蹤中。

traceParameters 預設為 `TRUE`。 如果 *traceParameters* 設為 `FALSE`，則只會追蹤方法簽章。

com.ibm.msg.client.commonservices.trace.startup = *startup*
IBM MQ classes for Java 有一個起始設定階段，在此期間會配置資源。 在資源配置階段期間會起始設定主要追蹤機能。

如果 *startup* 設為 TRUE，則會使用啟動追蹤。 追蹤資訊會立即產生，並包括所有元件的設定，包括追蹤機能本身。 啟動追蹤資訊可用來診斷配置問題。 啟動追蹤資訊一律寫入 System.err。

startup 預設為 FALSE。

在起始設定完成之前，會先檢查 *startup* 。 因此，請只將指令行上的內容指定為 Java 系統內容。 請勿在 IBM MQ classes for Java 配置檔中指定它。

com.ibm.msg.client.commonservices.trace.compress = *compressedTrace*
將 *compressedTrace* 設為 TRUE ， 以壓縮追蹤輸出。

compressedTrace 的預設值為 FALSE。

如果 *compressedTrace* 設為 TRUE，則會壓縮追蹤輸出。 預設追蹤輸出檔名稱的副檔名為 .trz。 如果壓縮設為 FALSE(預設值) ，則檔案副檔名為 .trc ， 表示它未經壓縮。 不過，如果已在 *traceOutputName* 中指定追蹤輸出的檔名，則會改用該名稱; 不會將字尾套用至檔案。

壓縮追蹤輸出小於未壓縮。 因為 I/O 較少，所以可以比未壓縮追蹤更快地寫出它。 與未經壓縮的追蹤相比，壓縮追蹤對 IBM MQ classes for Java 效能的影響較小。

如果設定 *maxTraceBytes* 和 *traceCycles* ，則會建立多個壓縮追蹤檔來取代多個純文字檔。

如果 IBM MQ classes for Java 以不受控制的方式結束，則壓縮追蹤檔可能無效。 因此，只有在 IBM MQ classes for Java 以受控制的方式關閉時，才必須使用追蹤壓縮。 只有在所調查的問題不會導致 JVM 本身非預期地停止時，才使用追蹤壓縮。 在診斷可能導致 System.Halt() 關機或異常、不受控制的 JVM 終止的問題時，請勿使用追蹤壓縮。

com.ibm.msg.client.commonservices.trace.level = *traceLevel*
traceLevel 指定追蹤的過濾層次。 定義的追蹤層次如下:

- TRACE_NONE: 0
- TRACE_EXCEPTION: 1
- TRACE_WARNING: 3
- TRACE_INFO: 6
- TRACE_ENTRYEXIT: 8
- TRACE_DATA: 9
- TRACE_ALL: Integer.MAX_VALUE

每一個追蹤層次都包含所有較低層次。 比方說，如果在 TRACE_INFO 設定追蹤層次，則任何已定義層次為 TRACE_EXCEPTION、 TRACE_WARNING 或 TRACE_INFO 的追蹤點都會寫入追蹤。 會排除所有其他追蹤點。

com.ibm.msg.client.commonservices.trace.standalone = *standaloneTrace*

standaloneTrace 控制是否在 WebSphere Application Server 環境中使用 IBM MQ classes for Java 用戶端追蹤服務。

如果 *standaloneTrace* 設為 TRUE，則會使用 IBM MQ classes for Java 用戶端追蹤內容來決定追蹤配置。

如果 *standaloneTrace* 設為 FALSE，且 IBM MQ classes for Java 用戶端在 WebSphere Application Server 儲存器中執行，則會使用 WebSphere Application Server 追蹤服務。 產生的追蹤資訊取決於應用程式伺服器的追蹤設定。

standaloneTrace 的預設值為 FALSE。

IBM MQ classes for Java 和軟體管理工具
軟體管理工具 (例如 Apache Maven) 可以與 IBM MQ classes for Java 搭配使用。

許多大型開發組織使用這些工具來集中管理協力廠商程式庫的儲存庫。

IBM MQ classes for Java 由許多 JAR 檔組成。 當您使用此 API 來開發 Java 語言應用程式時，需要在開發應用程式的機器上安裝 IBM MQ Server、 IBM MQ Client 或 IBM MQ Client SupportPac 。

如果您想要使用軟體管理工具，並將組成 IBM MQ classes for Java 的 JAR 檔新增至集中管理的儲存庫，則必須觀察下列要點:

- 儲存庫或儲存器只能供組織內的開發人員使用。 不允許在組織外部進行任何配送。
- 儲存庫需要包含一組來自單一 IBM MQ 版本或修正套件的完整且一致的 JAR 檔。
- 您負責以「IBM 支援中心」提供的任何維護來更新儲存庫。

從 IBM MQ 8.0 開始， `com.ibm.mq.allclient.jar` JAR 檔需要安裝到儲存庫中。

從 IBM MQ 9.0 開始，需要 Bouncy Castle 安全提供者和 CMS 支援 JAR 檔。 如需相關資訊，請參閱 第 295 頁的『IBM MQ classes for Java 可重新定位 JAR 檔』 及 支援非 IBM JRE。

IBM MQ classes for Java 應用程式的後置安裝設定

安裝 IBM MQ classes for Java 之後，您可以配置安裝，以便執行您自己的應用程式。

請記得檢查 IBM MQ 產品 Readme 檔，以取得最新資訊或您環境的特定相關資訊。 IBM MQ、 WebSphere MQ 及 MQSeries 產品 Readme 網頁上提供產品 Readme 檔的最新版本。

在嘗試以連結模式執行 IBM MQ classes for Java 應用程式之前，請確定您已依照 配置中的說明來配置 IBM MQ 。

配置佇列管理程式以接受來自 *IBM MQ classes for Java* 的用戶端連線

若要配置佇列管理程式來接受來自用戶端的送入連線要求，請定義並允許使用伺服器連線通道，並啟動接聽器程式。

請參閱第 897 頁的『配置佇列管理程式以接受 Multiplatforms 上的用戶端連線』，以取得詳細資料。

在 *Java security manager* 下執行 *IBM MQ classes for Java* 應用程式

IBM MQ classes for Java 可以在啟用 Java security manager 的情況下執行。 若要在啟用 Java security manager 的情況下順利執行應用程式，您必須使用適當的原則定義檔來配置 Java Virtual Machine (JVM)。

建立適當原則定義檔的最簡單方法是變更 Java runtime environment (JRE) 提供的原則檔。 在大部分系統上，此檔案儲存在相對於 JRE 目錄的 `path lib/security/java.policy` 中。 您可以使用偏好的編輯器或使用 JRE 隨附的 policytool 程式來編輯原則檔。

您必須提供 `com.ibm.mq.jmqi.jar` 檔案的權限，以便它可以:

- 建立 Socket (以用戶端模式)
- 載入原生程式庫 (以連結模式)
- 從環境讀取各種內容

在 Java security manager 下執行時， IBM MQ classes for Java 必須可以使用系統內容 **`os.name`** 。

如果 Java 應用程式使用 Java security manager，您必須將下列許可權新增至應用程式所使用的 `java.security.policy` 檔，否則會向應用程式擲出異常狀況:

```
permission java.lang.RuntimePermission "modifyThread";
```

在管理透過 TCP/IP 連線至佇列管理程式之多工交談的指派及關閉過程中，用戶端需要此 RuntimePermission 。

原則檔項目範例

以下是容許 IBM MQ classes for Java 在預設安全管理程式下順利執行的原則檔項目範例。 將此範例中的字串 *MQ_INSTALLATION_PATH* 取代為系統上安裝 IBM MQ classes for Java 的位置。

```
grant codeBase "file: MQ_INSTALLATION_PATH/java/lib/*" {
//We need access to these properties, mainly for tracing
permission java.util.PropertyPermission "user.name","read";
permission java.util.PropertyPermission "os.name","read";
permission java.util.PropertyPermission "user.dir","read";
permission java.util.PropertyPermission "line.separator","read";
permission java.util.PropertyPermission "path.separator","read";
permission java.util.PropertyPermission "file.separator","read";
permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.log.*","read";
```

```
permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.trace.*","read";
permission java.util.PropertyPermission "Diagnostics.Java.Errors.Destination.Filename","read";
permission java.util.PropertyPermission "com.ibm.mq.commonservices","read";
permission java.util.PropertyPermission "com.ibm.mq.cfg.*","read";

//Tracing - we need the ability to control java.util.logging
permission java.util.logging.LoggingPermission "control";
// And access to create the trace file and read the log file - assumed to be in the current
directory
permission java.io.FilePermission "*","read,write";

// Required to allow a trace file to be written to the filesystem.
// Replace 'TRACE_FILE_DIRECTORY' with the directory name where trace is to be written to
permission java.io.FilePermission "TRACE_FILE_DIRECTORY","read,write";
permission java.io.FilePermission "TRACE_FILE_DIRECTORY/*","read,write";

// We'd like to set up an mBean to control trace
permission javax.management.MBeanServerPermission "createMBeanServer";
permission javax.management.MBeanPermission "*","*";

// We need to be able to read manifests etc from the jar files in the installation directory
permission java.io.FilePermission "MQ_INSTALLATION_PATH/java/lib/-","read";

//Required if mqclient.ini/mqs.ini configuration files are used
permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqclient.ini","read";
permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqs.ini","read";

//For the client transport type.
permission java.net.SocketPermission "*","connect,resolve";

//For the bindings transport type.
permission java.lang.RuntimePermission "loadLibrary.*";

//For applications that use CCDT tables (access to the CCDT AMQCLCHL.TAB)
permission java.io.FilePermission "MQ_DATA_DIRECTORY/qmgrs/QM_NAME/@ipcc/AMQCLCHL.TAB","read";

//For applications that use User Exits
permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits/*","read";
permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits64/*","read";
permission java.lang.RuntimePermission "createClassLoader";

//Required for the z/OS platform
permission java.util.PropertyPermission "com.ibm.vm.bitmode","read";

// Used by the internal ConnectionFactory implementation
permission java.lang.reflect.ReflectPermission "suppressAccessChecks";

// Used for controlled class loading
permission java.lang.RuntimePermission "setContextClassLoader";

// Used to default the Application name in Client mode connections
permission java.util.PropertyPermission "sun.java.command","read";

// Used by the IBM JSSE classes
permission java.util.PropertyPermission "com.ibm.crypto.provider.AESNITrace","read";

//Required to determine if an IBM Java Runtime is running in FIPS mode,
//and to modify the property values status as required.
permission java.util.PropertyPermission "com.ibm.jsse2.usefipsprovider","read,write";
permission java.util.PropertyPermission "com.ibm.jsse2.JSSEFIPS","read,write";
//Required if an IBM FIPS provider is to be used for SSL communication.
permission java.security.SecurityPermission "insertProvider.IBMJCEFIPS";

// Required for non-IBM Java Runtimes that establish secure client
// transport mode connections using mutual TLS authentication
permission java.util.PropertyPermission "javax.net.ssl.keyStore","read";
permission java.util.PropertyPermission "javax.net.ssl.keyStorePassword","read";

// Required for Java applications that use the Java Security Manager
permission java.lang.RuntimePermission "modifyThread";
};
```

此原則檔範例可讓 IBM MQ classes for Java 在安全管理程式下正確運作，但您可能仍需要啟用自己的程式碼，才能在應用程式運作之前正確執行。

IBM MQ classes for Java 隨附的範例程式碼未明確啟用與安全管理程式搭配使用; 不過， IVT 測試會與此原則檔及預設安全管理程式一起執行。

重要:

IBM MQ classes for Java 追蹤機能需要進一步的許可權，因為它會執行系統內容的其他查詢，以及進一步的檔案系統作業。

在 IBM MQ 安裝架構的 samples/wmqjava 目錄中，提供了在啟用追蹤的安全管理程式下執行的適當範本安全原則檔 example.security.policy。

若為預設安裝， example.security.policy 檔案位於:

Windows
輸入 C:\Program Files\IBM\MQ\Tools\wmqjava\samples\example.security.policy

Linux
輸入 /opt/mqm/samp/wmqjava/samples/example.security.policy

Solaris
輸入 /opt/mqm/samp/wmqjava/samples/example.security.policy

AIX
輸入 /usr/mqm/samp/wmqjava/samples/example.security.policy

在 *CICS Transaction Server* 下執行 *IBM MQ classes for Java* 應用程式
IBM MQ classes for Java 應用程式可以在 CICS Transaction Server 下作為交易執行。

若要在 CICS Transaction Server for z/OS 下以交易形式執行 IBM MQ classes for Java 應用程式，請執行下列步驟:

1. 使用提供的 CEDA 交易，將應用程式及交易定義給 CICS 。
2. 確保 IBM MQ CICS 配接器已安裝在 CICS 系統中。 z/OS (如需詳細資料，請參閱 搭配使用 IBM MQ 與 CICS 。)
3. 請確定 CICS 中指定的 JVM 環境包含適當的 CLASSPATH 和 LIBPATH 項目。
4. 使用任何一般程序來起始交易。

如需執行 CICS Java 交易的相關資訊，請參閱 CICS 系統文件。

驗證 *IBM MQ classes for Java* 安裝

IBM MQ classes for Java 隨附安裝驗證程式 MQIVP。 您可以使用此程式來測試 IBM MQ classes for Java 的所有連線模式。

程式會提示您輸入一些選項及其他資料，以決定您要驗證的連線模式。 請使用下列程序來驗證安裝:

1. 如果您要以用戶端模式執行程式，請依照 第 897 頁的『配置佇列管理程式以接受 Multiplatforms 上的用戶端連線』中的說明來配置佇列管理程式。 要使用的佇列是 SYSTEM.DEFAULT.LOCAL.QUEUE
2. 如果您要以用戶端模式執行程式，另請參閱 第 290 頁的『使用 IBM MQ classes for Java』。

   在您要執行程式的系統上執行此程序的其餘步驟。
3. 請確定您已根據 第 299 頁的『與 IBM MQ classes for Java 相關的環境變數』中的指示更新 CLASSPATH 環境變數。
4. 將目錄切換至 *MQ_INSTALLATION_PATH*/mqm/samp/wmqjava/samples，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 安裝的路徑。 然後在命令提示字元中輸入:

   ```
   java -Djava.library.path= library_path MQIVP
   ```

   其中 *library_path* 是 IBM MQ classes for Java 程式庫的路徑 (請參閱 第 300 頁的『IBM MQ classes for Java 位於檔案庫中』)。

   在提示標示 (1) 時:

   - 若要使用 TCP/IP 連線，請輸入 IBM MQ 伺服器主機名稱。
   - 如果要使用原生連線 (連結模式) ，請將欄位保留空白 (請勿輸入名稱)。

   程式會嘗試:

   - 1. 連接至佇列管理程式

- 2. 開啟佇列 SYSTEM.DEFAULT.LOCAL.QUEUE，將訊息放置在佇列上，從佇列中取得訊息，然後關閉佇列
- 3. 切斷與佇列管理程式的連線
- 4. 如果作業成功，則傳回訊息

以下是您可能會看到的提示和回應範例。 實際提示和您的回應視您的 IBM MQ 網路而定。

```
Please enter the IP address of the MQ server    : ipaddress(1)
Please enter the port to connect to             : (1414) (2)
Please enter the server connection channel name : channelname (2)
Please enter the queue manager name             : qmname
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 MQ Transport is functioning correctly.
Press Enter to continue ...
```

註:

1. z/OS 在 z/OS 上，在標示 (1)的提示上，將欄位留白。
2. 如果您選擇伺服器連線，則不會看到標示 (2)的提示。
3. IBM i 在 IBM i 上，您只能從 QShell 發出 `java MQIVP` 指令。 或者，您可以使用 CL 指令 `RUNJVA CLASS(MQIVP)` 來執行應用程式。

使用 *IBM MQ classes for Java* 範例應用程式

IBM MQ classes for Java 範例應用程式提供 IBM MQ classes for Java API 一般特性的概觀。 您可以使用它們來驗證安裝及傳訊伺服器的設定，並協助您建置自己的應用程式。

關於這項作業

如果您需要協助來建立自己的應用程式，您可以使用範例應用程式作為起點。 會為每一個應用程式提供原始檔及已編譯版本。 檢閱範例原始碼，並識別關鍵步驟以建立應用程式的每一個必要物件 (MQQueueManager、MQConstants、MQMessage、 MQPutMessage 選項及 MQDestination)，以及設定任何特定內容以指定應用程式的運作方式。 如需相關資訊，請參閱 第 312 頁的『撰寫 IBM MQ classes for Java 應用程式』。 這些範例可能會在 IBM MQ Java 的未來版本中變更。

第 309 頁的表 55 顯示 IBM MQ classes for Java 範例應用程式在每一個平台上的安裝位置:

表 *55: IBM MQ classes for Java* 範例應用程式的安裝目錄	
平台	**目錄**
AIX AIX Linux Linux	*MQ_INSTALLATION_PATH*`/samp/wmqjava/samples`
Windows Windows	*MQ_INSTALLATION_PATH*`\tools\wmqjava\samples`
IBM i IBM i	`/qibm/proddata/mqm/java/samples/wmqjava/samples`
z/OS z/OS	*MQ_INSTALLATION_PATH*`/java/samples/wmqjava`

第 310 頁的表 56 顯示 IBM MQ classes for Java 隨附的範例應用程式集。

表 56: *IBM MQ classes for Java* 範例應用程式	
範例名稱	**說明**
IMSBridgeSample.java	簡式程式，示範如何搭配使用 IMS Bridge 與 IBM MQ classes for Java。
MQIVP.java	IBM MQ Java 安裝驗證程式。
MQMessagePropertiesSample.java	示範訊息內容 API 的使用。
MQPubSubApiSample.java	示範使用發佈/訂閱 API。
MQSample.java	簡式程式，示範從佇列放置及取得訊息。
MQSampleMessageManager.java	IBM MQ 基本 Java 範例中用於訊息處理的公用程式類別。
mqjcivp.properties	此資源組合包含 IBM MQ classes for Java 安裝驗證程式 (`MQIVP.java`) 所使用的訊息。

IBM MQ classes for Java 提供一個稱為 `runjms` 的 Script，可用來執行範例應用程式。 此 Script 會設定 IBM MQ 環境，以容許您執行 IBM MQ classes for Java 範例應用程式。

第 310 頁的表 57 顯示每一個平台上 Script 的位置:

表 57: *runjms Script* 的位置	
平台	**目錄**
AIX AIX Linux Linux	*MQ_INSTALLATION_PATH*`/java/bin/runjms`
Windows Windows	*MQ_INSTALLATION_PATH*`\java\bin\runjms.bat`

IBM i IBM i	`/qibm/proddata/mqm/java/bin/runjms`
	or
	`/qibm/proddata/mqm/java/bin/runjms64`

z/OS z/OS	*MQ_INSTALLATION_PATH*`java/bin/runjms`

若要使用 `runjms` Script 來呼叫範例應用程式，請完成下列步驟:

程序

1. 啟動命令提示字元，並導覽至包含您要執行之範例應用程式的目錄。
2. 輸入下列指令:

   ```
   Path to the runjms script/runjms sample_application_name
   ```

   範例應用程式會顯示它需要的參數清單。
3. 輸入下列指令，以使用這些參數來執行範例:

   ```
   Path to the runjms script/runjms sample_application_name parameters
   ```

範例

Linux 例如，若要在 Linux 上執行 MQIVP 範例，請輸入下列指令:

```
cd /opt/mqm/samp/wmqjava/samples
/opt/mqm/java/bin/runjms MQIVP
```

相關概念

第 74 頁的『針對 IBM MQ classes for JMS 安裝的內容』

當您安裝 IBM MQ classes for JMS 時，會建立一些檔案和目錄。 在 Windows 上，透過自動設定環境變數，在安裝期間執行部分配置。 在其他平台及某些 Windows 環境中，您必須先設定環境變數，然後才能執行 IBM MQ classes for JMS 應用程式。

解決 *IBM MQ classes for Java* 問題

一開始，請執行安裝驗證程式。 您也可能必須使用追蹤機能。

如果應用程式未順利完成，請執行安裝驗證程式，並遵循診斷訊息中提供的建議。 安裝驗證程式在 第 308 頁的『驗證 IBM MQ classes for Java 安裝』中說明。

如果問題繼續存在，且您需要聯絡 IBM 服務團隊，則可能會要求您開啟追蹤機能。 如下列範例所示執行此動作。

若要追蹤 MQIVP 程式，請執行下列動作:

- 建立 `com.ibm.mq.commonservices` 內容檔 (請參閱 使用 com.ibm.mq.commonservices。
- 輸入下列指令:

```
java -Dcom.ibm.mq.commonservices=commonservices_properties_file java
-Djava.library.path= library_path MQIVP -trace
```

其中:

- *commonservices_properties_file* 是 `com.ibm.mq.commonservices` 內容檔的路徑 (包括檔名)。
- *library_path* 是 IBM MQ classes for Java 程式庫的路徑 (請參閱 第 300 頁的『IBM MQ classes for Java 位於檔案庫中』)。

如需如何使用追蹤的相關資訊，請參閱 追蹤 IBM MQ classes for Java 應用程式。

z/OS MQ Adv. VUE Java 用戶端與在 z/OS 上執行之批次應用程式的連線功能

在特定條件下， z/OS 上的 IBM MQ classes for Java 應用程式可以使用用戶端連線連接至 z/OS 上的佇列管理程式。 使用用戶端連線可以簡化 IBM MQ 拓蹼。

透過使用用戶端連線，如果 IBM MQ classes for Java 應用程式在批次環境中執行，且適用下列其中一個條件，則應用程式可以連接至遠端 z/OS 佇列管理程式:

- LTS V 9.3.4 IBM MQ classes for Java 程式碼位於 IBM MQ 9.3.4 或更新版本，或 Long Term Support 已套用 APAR PH56722 。 佇列管理程式可以是任何支援的版本。
- 正在連接的佇列管理程式正在執行 IBM MQ Advanced for z/OS Value Unit Edition 授權，因此將 **ADVCAP** 參數設為 ENABLED。 佇列管理程式可以是任何受支援的版本。

如需 IBM MQ Advanced for z/OS Value Unit Edition 的相關資訊，請參閱 IBM MQ 產品 ID 及匯出資訊。

如需 **ADVCAP** 的相關資訊，請參閱 DISPLAY QMGR ; 如需 **QMGRPROD** 的相關資訊，請參閱 START QMGR 。

z/OS 上的 IBM MQ classes for Java 應用程式無法使用用戶端模式連線來連接未在 z/OS

如果 z/OS 上的 IBM MQ classes for Java 應用程式嘗試使用用戶端模式進行連接，但不容許這樣做，則會傳回 MQRC_ENVIRONMENT_ERROR 。

Advanced Message Security (AMS) 支援

IBM MQ classes for Java 用戶端應用程式在連接至遠端 z/OS 佇列管理程式時可以使用 AMS，但必須遵守本主題先前說明的條件。

若要以此方式使用 AMS，用戶端應用程式必須在 `keystore.conf` 中使用金鑰儲存庫類型 `jceracfks`，其中:

- 內容名稱字首為 `jceracfks`，且此名稱字首不區分大小寫。
- 金鑰儲存庫是 RACF 金鑰環。
- 不需要密碼，將予以忽略。 這是因為 RACF 金鑰環不使用密碼。
- 如果您指定提供者，提供者必須是 `IBMJCE`。

當您搭配使用 `jceracfks` 與 AMS 時，金鑰儲存庫必須採用下列格式: `safkeyring://`*`user`*`/`*`keyring`*，其中:

- `safkeyring` 是文字，且此名稱不區分大小寫
- *`user`* 是擁有金鑰環的 RACF 使用者 ID
- *`keyring`* 是 RACF 金鑰環的名稱，且金鑰環的名稱區分大小寫

下列範例使用使用者 `JOHNDOE` 的標準 AMS 金鑰環:

```
jceracfks.keystore=safkeyring://JOHNDOE/drq.ams.keyring
```

相關概念

第 106 頁的『JMS/Jakarta Messaging 用戶端與在 z/OS 上執行之批次應用程式的連線功能』
在特定狀況下， z/OS 上的 IBM MQ classes for JMS/Jakarta Messaging 應用程式可以使用用戶端連線連接至 z/OS 上的佇列管理程式。 使用用戶端連線可以簡化 IBM MQ 拓蹼。

撰寫 IBM MQ classes for Java 應用程式

此主題集合提供資訊來協助撰寫 Java 應用程式，以與 IBM MQ 系統互動。

若要使用 IBM MQ classes for Java 來存取 IBM MQ 佇列，您可以撰寫 Java 應用程式，其中包含將訊息放入 IBM MQ 佇列並從中取得訊息的呼叫。 如需個別類別的詳細資料，請參閱 IBM MQ classes for Java。

註: IBM MQ classes for Java 不支援自動用戶端重新連線。

IBM MQ classes for Java 介面

程序化 IBM MQ 應用程式設計介面使用動詞來處理物件。 Java 程式設計介面會使用您透過呼叫方法來處理的物件。

程序化 IBM MQ 應用程式設計介面是以動詞為建置基礎，例如:

```
MQBACK, MQBEGIN, MQCLOSE, MQCONN, MQDISC,
MQGET, MQINQ, MQOPEN, MQPUT, MQSET, MQSUB
```

這些動詞全部都採用它們要在其上操作之 IBM MQ 物件的控點作為參數。 您的程式由一組 IBM MQ 物件組成，您可以對這些物件呼叫方法來處理這些物件。

當您使用程序化介面時，您可以使用呼叫 MQDISC (Hconn， CompCode， Reason) 來切斷佇列管理程式的連線，其中 *Hconn* 是佇列管理程式的控點。

在 Java 介面中，佇列管理程式由類別 MQQueueManager 的物件代表。 您可以在該類別上呼叫 disconnect () 方法來切斷佇列管理程式的連線。

```
// declare an object of type queue manager
MQQueueManager queueManager=new MQQueueManager();
...
// do something...
...
```

```
// disconnect from the queue manager
queueManager.disconnect();
```

IBM MQ classes for Java 連線模式

IBM MQ classes for Java 的程式方式與您要使用的連線模式有一些相依關係。

如果您使用用戶端連線，則與 IBM MQ MQI client 有一些差異，但概念上類似。 如果您使用連結模式，則可以使用捷徑連結，並且可以發出 MQBEGIN 指令。 您可以透過在 MQEnvironment 類別中設定變數來指定要使用的模式。

IBM MQ classes for Java 用戶端連線

當使用 IBM MQ classes for Java 作為用戶端時，它類似於 IBM MQ MQI client，但有一些差異。

如果您是針對 *IBM MQ classes for Java* 進行程式設計以用作用戶端，請注意下列差異:

- 它僅支援 TCP/IP。
- 它在啟動時不會讀取任何 IBM MQ 環境變數。
- 儲存在通道定義及環境變數中的資訊可以儲存在稱為「環境」的類別中。 或者，此資訊可以在建立連線時作為參數傳遞。
- 錯誤和異常狀況會寫入 `MQException` 類別中指定的日誌。 預設錯誤目的地是 Java 主控台。
- 只有 IBM MQ 用戶端配置檔中的下列屬性與 IBM MQ classes for Java 相關。 如果您指定其他屬性，則它們無效。

段落 (stanza)	**屬性**
ClientExit 用戶端配置檔的路徑段落	ExitsDefaultPath
ClientExit 用戶端配置檔的路徑段落	ExitsDefaultPath64
ClientExit 用戶端配置檔的路徑段落	JavaExitsClasspath
用戶端配置檔的 MessageBuffer 段落	MaximumSize
用戶端配置檔的 MessageBuffer 段落	PurgeTime
用戶端配置檔的 MessageBuffer 段落	UpdatePercentage
用戶端配置檔的 TCP 段落	ClntRcvBuffSize
用戶端配置檔的 TCP 段落	ClntSndBuffSize
用戶端配置檔的 TCP 段落	連線逾時
用戶端配置檔的 TCP 段落	KeepAlive

- 如果連接至需要轉換字元資料的佇列管理程式，則在佇列管理程式無法執行轉換時， V7 Java 用戶端現在可以執行轉換。 用戶端 JVM 必須支援用戶端的 CCSID 與佇列管理程式的 CCSID 之間的轉換。
- IBM MQ classes for Java 不支援自動重新連接用戶端。

以用戶端模式使用時， *IBM MQ classes for Java* 不支援 MQBEGIN 呼叫。

IBM MQ classes for Java BINDINGS 模式

IBM MQ classes for Java 的連結模式與用戶端模式有三種主要不同。

在連結模式中使用時， IBM MQ classes for Java 會使用 Java 原生介面 (JNI) 直接呼叫現有的佇列管理程式 API，而不是透過網路進行通訊。

依預設，在連結模式下使用 IBM MQ classes for Java 的應用程式會使用 *ConnectOption*(MQCNO_STANDARD_BINDINGS) 連接至佇列管理程式。

IBM MQ classes for Java 支援下列 *ConnectOptions*:

- MQCNO_FASTPATH_BINDING
- MQCN_STANDARD_BINDING

- MQCNO_SHARED_BINDING
- MQCNO_ISOLATED_BINDING

如需 *ConnectOptions* 的進一步資訊，請參閱 第 618 頁的『使用 MQCONNX 呼叫連接至佇列管理程式』。

連結模式支援 MQBEGIN 呼叫，可在 IBM MQ for IBM i 及 IBM MQ for z/OS 以外的所有平台上起始由佇列管理程式協調的廣域工作單元。

`MQEnvironment` 類別所提供的大部分參數與連結模式無關，因此會被忽略。

定義要使用的 *IBM MQ classes for Java* 連線
要使用的連線類型取決於 MQEnvironment 類別中的變數設定。

使用兩個變數:

MQEnvironment.properties
連線類型由與金鑰名稱 `CMQC.TRANSPORT_PROPERTY` 相關聯的值決定。 可能的值如下:

CMQC.TRANSPORT_MQSERIES_BINDINGS
以連結模式連接

CMQC.TRANSPORT_MQSERIES_CLIENT
以用戶端模式連接

CMQC.TRANSPORT_MQSERIES
連線模式由 *hostname* 內容的值決定

MQEnvironment.hostname
設定此變數的值，如下所示:

- 若為用戶端連線，請將此變數的值設為您要連接之 IBM MQ 伺服器的主機名稱
- 對於連結模式，請勿設定此變數，或將其設為空值

佇列管理程式上的作業

這個主題集合說明如何使用「IBM MQ classes for Java」來連接至佇列管理程式，以及與佇列管理程式中斷連線。

為 *IBM MQ classes for Java* 設定 *IBM MQ* 環境
若要讓應用程式以用戶端模式連接至佇列管理程式，應用程式必須指定通道名稱、主機名稱及埠號。

註: 僅當您的應用程式以用戶端模式連接至佇列管理程式時，本主題中的資訊才相關。 如果它以連結模式連接，則它不相關。 請參閱: 第 92 頁的『IBM MQ classes for JMS 的連線模式』

您可以以下列兩種方式之一指定通道名稱、主機名稱及埠號: 作為 MQEnvironment 類別中的欄位，或作為 MQQueueManager 物件的內容。

如果您在 MQEnvironment 類別中設定欄位，則它們會套用至整個應用程式，但內容雜湊表會置換它們的位置除外。 若要在 MQEnvironment 中指定通道名稱及主機名稱，請使用下列程式碼:

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

這相當於設定 **MQSERVER** 環境變數:

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

依預設， Java 用戶端會嘗試連接至埠 1414 上的 IBM MQ 接聽器。 若要指定不同的埠，請使用下列程式碼:

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

其中 nnnn 是必要的埠號

如果您在建立時將內容傳遞至佇列管理程式物件，則它們只會套用至該佇列管理程式。 在 Hashtable 物件中建立索引鍵為 **hostname**、 **channel** 及 (選用) **port** 的項目，並使用適當的值。 若要使用預設埠 1414，您可以省略 **port** 項目。 使用接受內容雜湊表的建構子來建立 MQQueueManager 物件。

透過設定應用程式名稱來識別與佇列管理程式的連線

應用程式可以設定名稱來識別其與佇列管理程式的連線。 此應用程式名稱由 **DISPLAY CONN MQSC/PCF** 指令顯示 (其中欄位稱為 **APPLTAG**) 或在 IBM MQ 瀏覽器 **應用程式連線** 顯示畫面中 (其中欄位稱為 **App name**)。

應用程式名稱限制為 28 個字元，因此會截斷較長的名稱。 如果未指定應用程式名稱，則會提供預設值。 預設名稱基於呼叫 (main) 類別，但如果無法使用此資訊，則會使用文字 `IBM MQ Client for Java`。

如果使用呼叫類別的名稱，必要的話，會移除前導套件名稱來調整它以適合。 例如，如果呼叫端類別是 `com.example.MainApp`，則會使用完整名稱，但如果呼叫端類別是 `com.example.dictionaryAndThesaurus.multilingual.mainApp`，則會使用名稱 `multilingual.mainApp` ，因為它是最適合可用長度的類別名稱與最右邊套件名稱的最長組合。

如果類別名稱本身長度超過 28 個字元，則會截斷以適合。 例如，`com.example.mainApplicationForSecondTestCase` 會變成 `mainApplicationForSecondTest`。

若要在 MQEnvironment 類別中設定應用程式名稱，請使用下列程式碼，將名稱新增至 MQEnvironment.properties 雜湊表 (索引鍵為 **MQConstants.APPNAME_PROPERTY**):

```
MQEnvironment.properties.put(MQConstants.APPNAME_PROPERTY, "my_application_name");
```

若要在傳遞至 MQQueueManager 建構子的內容雜湊表中設定應用程式名稱，請將名稱新增至索引鍵為 **MQConstants.APPNAME_PROPERTY** 的內容雜湊表。

置換 IBM MQ 用戶端配置檔中指定的內容

IBM MQ 用戶端配置檔也可以指定用來配置 IBM MQ classes for Java 的內容。 不過，只有在應用程式以用戶端模式連接至佇列管理程式時， IBM MQ MQI client 配置檔中指定的內容才適用。

必要的話，您可以使用下列任何方式來置換 IBM MQ 配置檔中的任何屬性。 選項依優先順序顯示。

- 設定配置內容的 Java 系統內容。
- 在 MQEnvironment.properties 對映中設定內容。
- 在 Java5 以及更新版本上，設定系統環境變數。

只有 IBM MQ 用戶端配置檔中的下列屬性與 IBM MQ classes for Java 相關。 如果您指定或置換其他屬性，則它沒有作用。

段落 (stanza)	屬性
ClientExit 用戶端配置檔的路徑段落	ExitsDefaultPath
ClientExit 用戶端配置檔的路徑段落	ExitsDefaultPath64
ClientExit 用戶端配置檔的路徑段落	JavaExitsClasspath
用戶端配置檔的 MessageBuffer 段落	MaximumSize
用戶端配置檔的 MessageBuffer 段落	PurgeTime
用戶端配置檔的 MessageBuffer 段落	UpdatePercentage
用戶端配置檔的 TCP 段落	ClntRcvBufSize
用戶端配置檔的 TCP 段落	ClntSndBufSize
用戶端配置檔的 TCP 段落	連線逾時

段落 (stanza)	屬性
用戶端配置檔的 TCP 段落	KeepAlive

在「*IBM MQ classes for Java* 」中連接至佇列管理程式
建立新的 MQQueueManager 類別實例，以連接至佇列管理程式。 呼叫 disconnect () 方法來切斷佇列管理程式的連線。

現在，您已準備好建立 MQQueueManager 類別的新實例，以連接至佇列管理程式:

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

如果要切斷佇列管理程式的連線，請在佇列管理程式上呼叫 disconnect () 方法:

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

如果您呼叫斷線方法，則會關閉您透過該佇列管理程式存取的所有開啟佇列及處理程序。 不過，當您完成使用這些資源時，最好是明確地關閉這些資源。 如果要這麼做，請在相關物件上使用 close () 方法。

佇列管理程式上的 commit () 和 backout () 方法相當於與程序化介面搭配使用的 MQCMIT 和 MQBACK 呼叫。

搭配使用用戶端通道定義表與 *IBM MQ classes for Java*
IBM MQ classes for Java 用戶端應用程式可以使用儲存在用戶端通道定義表 (CCDT) 中的用戶端連線通道定義。

除了透過在 `MQEnvironment` 類別中設定某些欄位及環境內容，或將它們傳遞至內容雜湊表中的 `MQQueueManager`，來建立用戶端連線通道定義之外， IBM MQ classes for Java 用戶端應用程式還可以使用儲存在用戶端通道定義表中的用戶端連線通道定義。 這些定義是由 IBM MQ Script (MQSC) 指令或 IBM MQ 可程式化指令格式 (PCF) 指令所建立，或使用 IBM MQ Explorer 來建立。

當應用程式建立 `MQQueueManager` 物件時， IBM MQ classes for Java 用戶端會在用戶端通道定義表中搜尋適當的用戶端連線通道定義，並使用通道定義來啟動 MQI 通道。 如需用戶端通道定義表及如何建構一個通道定義表的相關資訊，請參閱 用戶端通道定義表。

若要使用用戶端通道定義表，應用程式必須先建立 URL 物件。 URL 物件封裝統一資源定址器 (URL)，可識別包含用戶端通道定義表之檔案的名稱及位置，並指定如何存取檔案。

例如，如果 `ccdt1.tab` 檔案包含用戶端通道定義表，且儲存在應用程式執行所在的相同系統上，則應用程式可以使用下列方式建立 URL 物件:

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
```

另一個範例是，假設檔案 `ccdt2.tab` 包含用戶端通道定義表，且儲存在不同於應用程式執行所在的系統上。 如果可以使用 FTP 通訊協定來存取檔案，應用程式可以使用下列方式來建立 URL 物件:

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
```

在應用程式建立 URL 物件之後，應用程式可以使用其中一個建構子來建立 `MQQueueManager` 物件，以 URL 物件作為參數。 以下是範例:

```
MQQueueManager mars = new MQQueueManager("MARS", chanTab2);
```

此陳述式會導致 IBM MQ classes for Java 用戶端存取由 URL 物件 chanTab2 所識別的用戶端通道定義表，搜尋表格中是否有適當的用戶端連線通道定義，然後使用通道定義來啟動佇列管理程式 MARS 的 MQI 通道。

如果應用程式使用用戶端通道定義表，請注意下列要點:

- 當應用程式使用以 URL 物件作為參數的建構子來建立 MQQueueManager 物件時， MQEnvironment 類別中不得設定通道名稱作為欄位或環境內容。 如果設定通道名稱， IBM MQ classes for Java 用戶端會擲出 MQException。 如果指定通道名稱的欄位或環境內容值不是空值、空字串或包含所有空白字元的字串，則會將該欄位或環境內容視為已設定。
- MQQueueManager 建構子上的 **queueManagerName** 參數可以具有下列其中一個值:
  - 佇列管理程式的名稱
  - 星號 (*) 後接佇列管理程式群組的名稱
  - 星號 (*)
  - 空值、空字串或包含所有空白字元的字串

  這些值與使用「訊息佇列介面 (MQI)」的用戶端應用程式發出的 MQCONN 呼叫中，可用於 **QMgrName** 參數的值相同。 如需這些值意義的相關資訊，請參閱第 604 頁的『訊息佇列介面概觀』。

  如果您的應用程式使用連線儲存區，請參閱第 333 頁的『控制 IBM MQ classes for Java 中的預設連線儲存區』。
- 當 IBM MQ classes for Java 用戶端在用戶端通道定義表中找到適當的用戶端連線通道定義時，它只會使用從這個通道定義擷取的資訊來啟動 MQI 通道。 系統會忽略應用程式可能在 MQEnvironment 類別中設定的任何通道相關欄位或環境內容。

  如果您使用「傳輸層安全 (TLS)」，請特別注意下列要點:
  - 只有在從用戶端通道定義表擷取的通道定義指定 IBM MQ classes for Java 用戶端支援的 CipherSpec 名稱時， MQI 通道才會使用 TLS。
  - 用戶端通道定義表也包含保留憑證撤銷清冊 (CRL) 之「輕量型目錄存取通訊協定 (LDAP)」伺服器位置的相關資訊。 IBM MQ classes for Java 用戶端只會使用此資訊來存取保留 CRL 的 LDAP 伺服器。
  - 用戶端通道定義表也可以包含 OCSP 回應端的位置。 「IBM MQ classes for Java」無法在用戶端通道定義表檔案中使用 OCSP 資訊。 不過，您可以依照 使用線上憑證通訊協定 一節中的說明來配置 OCSP

  如需搭配使用 TLS 與用戶端通道定義表的相關資訊，請參閱 指定 MQI 通道使用 TLS。

  如果您使用通道結束程式，也請注意下列要點:
  - MQI 通道會優先使用從用戶端通道定義表中擷取的通道定義所指定的通道結束程式及相關聯使用者資料，而不使用其他方法所指定的通道結束程式及資料。
  - 從用戶端通道定義表擷取的通道定義可以指定以 Java、C 或 C++ 撰寫的通道結束程式。如需如何在 Java 中寫入通道結束程式的相關資訊，請參閱第 328 頁的『在 IBM MQ classes for Java 中建立通道結束程式』。 如需如何以其他語言撰寫通道結束程式的相關資訊，請參閱 第 331 頁的『將未在 Java 中寫入的通道結束程式與 IBM MQ classes for Java 搭配使用』。

指定 *IBM MQ classes for Java* 用戶端連線的埠範圍

您可以用兩種方式之一來指定應用程式可連結的埠或埠範圍。

當 IBM MQ classes for Java 應用程式嘗試以用戶端模式連接至 IBM MQ 佇列管理程式時，防火牆可能只容許源自指定埠或埠範圍的那些連線。 在此狀況下，您可以指定應用程式可連結的埠或埠範圍。 您可以使用下列方式來指定埠:

- 您可以在 MQEnvironment 類別中設定 localAddress 設定欄位。 以下是範例:

```
MQEnvironment.localAddressSetting = "192.0.2.0(2000,3000)";
```

- 您可以設定環境內容 CMQC.LOCAL_ADDRESS_PROPERTY。 以下是範例:

```
(MQEnvironment.properties).put(CMQC.LOCAL_ADDRESS_PROPERTY,
                               "192.0.2.0(2000,3000)");
```

- 當您可以建構 MQQueueManager 物件時，您可以傳遞包含 LOCAL_ADDRESS_PROPERTY 且值為 "192.0.2.0(2000,3000)" 的內容雜湊表

在每一個範例中，當應用程式稍後連接至佇列管理程式時，應用程式會連結至 192.0.2.0(2000) 至 192.0.2.0(3000) 範圍內的本端 IP 位址及埠號。

在具有多個網路介面的系統中，您也可以使用 localAddress 設定欄位或環境內容 CMQC.LOCAL_ADDRESS_PROPERTY，指定連線必須使用的網路介面。

如果您限制埠範圍，則可能會發生連線錯誤。 如果發生錯誤，則會擲出 MQException ，其中包含 IBM MQ 原因碼 MQRC_Q_MGR_NOT_AVAILABLE 及下列訊息:

```
Socket connection attempt refused due to LOCAL_ADDRESS_PROPERTY restrictions
```

如果指定範圍內的所有埠都在使用中，或指定的 IP 位址、主機名稱或埠號無效 (例如，負數埠號) ，則可能會發生錯誤。

在 *IBM MQ classes for Java* 中存取佇列、主題及處理程序

若要存取佇列、主題及處理程序，請使用 MQQueueManager 類別的方法。 MQOD (物件描述子結構) 會收合到這些方法的參數中。

佇列

若要開啟佇列，您可以使用 MQQueueManager 類別的 accessQueue 方法。 例如，在稱為 queueManager 的佇列管理程式上，使用下列程式碼:

```
MQQueue queue = queueManager.accessQueue("qName",CMQC.MQOO_OUTPUT);
```

accessQueue 方法會傳回 MQQueue 類別的新物件。

當您完成使用佇列時，請使用 close () 方法來關閉它，如下列範例所示:

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

您也可以使用 MQQueue 建構子來建立佇列。 這些參數與 accessQueue 方法的參數完全相同，加上佇列管理程式參數。 例如:

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

您可以在建立佇列時指定一些選項。 如需這些的詳細資料，請參閱 Class.com.ibm.mq.MQQueue。 以此方式建構佇列物件可讓您撰寫自己的 MQQueue 子類別。

主題

同樣地，您可以使用 MQQueueManager 類別的 accessTopic 方法來開啟主題。 例如，在稱為 queueManager 的佇列管理程式上，使用下列程式碼來建立訂閱者和發佈者:

```
MQTopic subscriber =
      queueManager.accessTopic("TOPICSTRING","TOPICNAME",
      CMQC.MQTOPIC_OPEN_AS_SUBSCRIPTION, CMQC.MQSO_CREATE);
```

```
MQTopic publisher =
      queueManager.accessTopic("TOPICSTRING","TOPICNAME",
      CMQC.MQTOPIC_OPEN_AS_PUBLICATION, CMQC.MQOO_OUTPUT);
```

當您完成使用主題時，請使用 close () 方法來關閉它。

您也可以使用 MQTopic 建構子來建立主題。 這些參數與 accessTopic 方法的參數完全相同，新增了佇列管理程式參數。 例如:

```
MQTopic subscriber = new
      MQTopic(queueManager,"TOPICSTRING","TOPICNAME",
      CMQC.MQTOPIC_OPEN_AS_SUBSCRIPTION, CMQC.MQSO_CREATE);
```

您可以在建立主題時指定一些選項。 如需這些詳細資料，請參閱 類別 com.ibm.mq.MQTopic。 以此方式建構主題物件可讓您撰寫自己的 MQTopic 子類別。

必須開啟主題以進行發佈或訂閱。 MQQueueManager 類別有八個 accessTopic 方法，且 Topic 類別有八個建構子。 在每一種情況下，其中四個具有 **destination** 參數，四個具有 **subscriptionName** 參數 (包括兩個同時具有這兩個參數)。 這些只能用來開啟訂閱的主題。 其餘兩個方法都有 **openAs** 參數，視 **openAs** 參數的值而定，可以開啟主題來進行發佈或訂閱。

若要建立主題作為可延續訂閱者，請使用 MQQueueManager 類別的 accessTopic 方法，或接受訂閱名稱且在任一情況下設定 CMQC.MQSO_DURABLE 選項。

Processes

若要存取處理程序，請使用 MQQueueManager 的 accessProcess 方法。 例如，在稱為 queueManager 的佇列管理程式上，使用下列程式碼來建立 MQProcess 物件:

```
MQProcess process =
queueManager.accessProcess("PROCESSNAME",
CMQC.MQOO_FAIL_IF_QUIESCING);
```

若要存取處理程序，請使用 MQQueueManager 的 accessProcess 方法。

accessProcess 方法會傳回 MQProcess 類別的新物件。

當您完成使用程序物件時，請使用 close () 方法來關閉它，如下列範例所示:

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

您也可以使用 MQProcess 建構子來建立程序。 這些參數與 accessProcess 方法的參數完全相同，並新增了佇列管理程式參數。 例如:

```
MQProcess process =
      new MQProcess(queueManager,"PROCESSNAME",
      CMQC.MQOO_FAIL_IF_QUIESCING);
```

以此方式建構處理程序物件可讓您撰寫自己的 MQProcess 子類別。

處理 *IBM MQ classes for Java* 中的訊息

訊息由 MQMessage 類別代表。 您可以使用 MQDestination 類別 (具有 MQQueue 及 MQTopic 子類別) 的方法來放置及取得訊息。

使用 MQDestination 類別的 put () 方法，將訊息放入佇列或主題。 您可以使用 MQDestination 類別的 get () 方法，從佇列或主題取得訊息。 與 MQPUT 及 MQGET 放置及取得位元組陣列的程序化介面不同， Java 程式設計語言會放置及取得 MQMessage 類別的實例。 MQMessage 類別會封裝包含實際訊息資料的資料緩衝區，以及說明該訊息的所有 MQMD (訊息描述子) 參數和訊息內容。

若要建置新訊息，請建立 MQMessage 類別的新實例，並使用 writeXXX 方法將資料放入訊息緩衝區。

建立新的訊息實例時，所有 MQMD 參數都會自動設為其預設值，如 MQMD 的起始值及語言宣告中所定義。 MQDestination 的 put () 方法也會採用 MQPutMessageOptions 類別的實例作為參數。 此類別代表 MQPMO 結構。 下列範例會建立訊息並將它放入佇列:

```
// Build a new message containing my age followed by my name
MQMessage myMessage = new MQMessage();
myMessage.writeInt(25);

String name = "Charlie Jordan";
myMessage.writeInt(name.length());
```

```
myMessage.writeBytes(name);

// Use the default put message options...
MQPutMessageOptions pmo = new MQPutMessageOptions();

// put the message
!queue.put(myMessage,pmo);
```

MQDestination 的 get () 方法會傳回新的 MQMessage 實例，其代表剛從佇列取得的訊息。 它也會採用 MQGetMessageOptions 類別的實例作為參數。 此類別代表 MQGMO 結構。

您不需要指定訊息大小上限，因為 get () 方法會自動調整其內部緩衝區的大小，以符合送入訊息。 使用 MQMessage 類別的 readXXX 方法來存取所傳回訊息中的資料。

下列範例顯示如何從佇列取得訊息:

```
// Get a message from the queue
MQMessage theMessage    = new MQMessage();
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.get(theMessage,gmo); // has default values

// Extract the message data
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，來變更 read 和 write 方法使用的數字格式。

您可以透過設定 *characterSet* 成員變數，變更字集以用於讀取及寫入字串。

如需相關資訊，請參閱 MQMessage 類別 。

註: MQMessage 的 writeUTF() 方法會自動對字串長度及其包含的 Unicode 位元組進行編碼。 當另一個 Java 程式 (使用 readUTF()) 讀取您的訊息時，這是傳送字串資訊最簡單的方式。

改善 *IBM MQ classes for Java* 中非持續訊息的效能
若要改善從用戶端應用程式瀏覽訊息或耗用非持續訊息時的效能，您可以使用 先讀。 使用 MQGET 或非同步耗用的用戶端應用程式，在瀏覽訊息或耗用非持續訊息時，會因效能改善而受益。

如需先讀機能的一般資訊，請參閱相關主題。

在 IBM MQ classes for Java 中，您使用 CMQC.MQSO_READ_AHEAD 和 CMQC.MQSO_NO_READ_AHEAD 內容，以判定是否容許訊息消費者及佇列瀏覽器對該物件使用先讀。

使用 *IBM MQ classes for Java* 非同步放置訊息
若要非同步放置訊息，請設定 MQPMO_ASYNC_RESPONSE。

您可以使用 MQDestination 類別的 put () 方法，將訊息放置在佇列或主題上。 若要非同步放置訊息 (即容許作業完成而不等待佇列管理程式的回應) ，您可以在 MQPutMessage 選項的選項欄位中設定 MQPMO_ASYNC_RESPONSE。 若要判定非同步放置的成功或失敗，請使用 MQQueueManager.getAsyncStatus 呼叫。

在 *IBM MQ classes for Java* 中發佈/訂閱

在 IBM MQ classes for Java 中，主題由 MQTopic 類別代表，您可以使用 MQTopic.put() 方法向其發佈。

如需 IBM MQ 發佈/訂閱的一般資訊，請參閱 發佈/訂閱傳訊。

使用 *IBM MQ classes for Java* 處理 *IBM MQ* 訊息標頭

提供的 Java 類別代表不同類型的訊息標頭。 也提供兩個 Helper 類別。

MQHeader 介面

標頭物件由 MQHeader 介面說明，該介面提供一般用途方法來存取標頭欄位以及讀取和寫入訊息內容。 每一個標頭類型都有自己的類別來實作 MQHeader 介面，並新增個別欄位的 getter 和 setter 方法。 例如，MQRFH2 標頭類型由 MQRFH2 類別代表; MQDLH 標頭類型由 MQDLH 類別代表，依此類推。 標頭類別會自動執行任何必要的資料轉換，並且可以使用任何指定的數值編碼或字集 (CCSID) 來讀取或寫入資料。

重要: MQRFH2 標頭類別會將訊息視為隨機存取檔案，這表示游標必須位於訊息開頭。 在使用內部訊息標頭類別 (例如 MQRFH、 MQRFH2、MQCIH、MQDEAD、MQIIH 或 MQXMIT) 之前，請確定您將訊息的游標位置更新至正確的位置，然後再將訊息傳遞至類別。

Helper 類別

兩個 Helper 類別 (MQHeaderIterator 和 MQHeaderList) 可協助讀取及解碼 (剖析) 訊息中的標頭內容:

- MQHeaderIterator 類別的運作方式類似於 java.util.Iterator。 只要訊息中有更多標頭，則 next () 方法會傳回 true ， 而 nextHeader() 或 next () 方法會傳回下一個標頭物件。
- MQHeaderList 的運作方式類似於 java.util.List。 與 MQHeaderIterator 類似，它會剖析標頭內容，但也可讓您搜尋特定標頭、新增標頭、移除現有標頭、更新標頭欄位，然後將標頭內容寫回訊息。 或者，您可以建立空的 MQHeaderList，然後將標頭實例移入其中，並將它寫入訊息一次或反覆。

MQHeaderIterator 及 MQHeaderList 類別使用 MQHeaderRegistry 中的資訊來知道哪些 IBM MQ 標頭類別與特定訊息類型及格式相關聯。 MQHeaderRegistry 已配置為具備所有現行 IBM MQ 格式和標頭類型及其實作類別的知識，您也可以登錄自己的標頭類型。

提供下列常用 IBM MQ 標頭的支援

- MQRFH-規則和格式化標頭
- MQRFH2 -類似 MQRFH ， 用來在屬於 IBM Integration Bus 的訊息分配管理系統之間傳遞訊息。 也用來包含訊息內容
- MQCIH- CICS 橋接器
- MQDLH-無法傳送的郵件標頭
- MQIIH- IMS 資訊標頭
- MQRMH-參照訊息標頭
- MQSAPH- SAP 標頭
- MQWIH-工作資訊標頭
- MQXQH-傳輸佇列標頭
- MQDH-Distribution 標頭
- MQEPH 封裝的 PCF 標頭

您也可以定義類別來代表您自己的標頭。

若要使用 MQHeaderIterator 來取得 RFH2 標頭，請在 GetMessage 選項中設定 MQGMO_PROPERTIES_FORCE_MQRFH2 ， 或將佇列內容 PROPCTL 設為 FORCE。

使用 *IBM MQ classes for Java* 列印訊息中的所有標頭
在此範例中， MQHeaderIterator 實例會剖析已從佇列收到的 MQMessage 中的標頭。 從 nextHeader() 方法傳回的 MQHeader 物件會在呼叫其 toString 方法時顯示其結構及內容。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeader;
import com.ibm.mq.headers.MQHeaderIterator;
...
MQMessage message = ... // Message received from a queue.
MQHeaderIterator it = new MQHeaderIterator (message);

while (it.hasNext ())
{
    MQHeader header = it.nextHeader ();
```

```
    System.out.println ("Header type " + header.type () + ": " + header);
}
```

使用 *IBM MQ classes for Java* 跳過訊息中的標頭
在此範例中， MQHeaderIterator 的 skipHeaders() 方法會將訊息讀取游標緊接在最後一個標頭之後。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderIterator;
...
MQMessage message = ... // Message received from a queue.
MQHeaderIterator it = new MQHeaderIterator (message);

it.skipHeaders ();
```

使用 *IBM MQ classes for Java* 在無法傳送郵件的訊息中尋找原因碼
在此範例中， read 方法會從訊息讀取來移入 MQDLH 物件。 在讀取作業之後，訊息讀取游標會緊接在 MQDLH 標頭內容之後。

佇列管理程式無法傳送郵件的佇列上的訊息會以無法傳送郵件的標頭 (MQDLH) 作為字首。 若要決定如何處理這些訊息 (例如，決定是重試還是捨棄它們) ，無法傳送郵件的處理應用程式必須查看 MQDLH 中包含的原因碼。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;
...
MQMessage message = ... // Message received from the dead-letter queue.
MQDLH dlh = new MQDLH ();

dlh.read (message);

System.out.println ("Reason: " + dlh.getReason ());
```

所有標頭類別也提供便利建構子，可在單一步驟中直接從訊息起始設定它們自己。 因此，此範例中的程式碼可以簡化如下:

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;
...
MQMessage message = ... // Message received from the dead-letter queue.
MQDLH dlh = new MQDLH (message);

System.out.println ("Reason: " + dlh.getReason ());
```

使用 *IBM MQ classes for Java* 從無法傳送郵件的訊息讀取及移除標頭
在此範例中，使用 MQDLH 從無法傳送郵件的訊息中移除標頭。

如果無法傳送郵件的處理應用程式的原因碼指出暫時性錯誤，則通常會重新提交已拒絕的訊息。 在重新提交訊息之前，它必須移除 MQDLH 標頭。

此範例會執行下列步驟 (請參閱範例程式碼中的註解):

1. MQHeaderList 會讀取整個訊息，訊息中發現的每一個標頭都會變成清單中的項目。
2. 無法傳送郵件的訊息包含 MQDLH 作為其第一個標頭，因此可以在標頭清單的第一個項目中找到此訊息。 建置 MQHeaderList 時已從訊息移入 MQDLH ，因此不需要呼叫其讀取方法。
3. 使用 MQDLH 類別提供的 getReason() 方法擷取原因碼。
4. 已檢查原因碼，並指出適合重新提交訊息。 使用 MQHeaderList remove () 方法來移除 MQDLH。
5. MQHeaderList 會將其剩餘內容寫入新的訊息物件。 新訊息現在包含原始訊息 (MQDLH 除外) 中的所有內容，並且可以寫入佇列。 建構子及寫入方法的 **true** 引數指出訊息內文將保留在 MQHeaderList 內，並再次寫出。
6. 新訊息的訊息描述子中的格式欄位現在包含先前在 MQDLH 格式欄位中的值。 訊息資料符合訊息描述子中設定的數值編碼和 CCSID。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;
import com.ibm.mq.headers.MQHeaderList;
...
MQMessage message = ... // Message received from the dead-letter queue.
MQHeaderList list = new MQHeaderList (message, true); // Step 1.
MQDLH dlh = (MQDLH) list.get (0); // Step 2.
int reason = dlh.getReason (); // Step 3.
...
list.remove (dlh); // Step 4.

MQMessage newMessage = new MQMessage ();

list.write (newMessage, true); // Step 5.
newMessage.format = list.getFormat (); // Step 6.
```

使用 *IBM MQ classes for Java* 列印訊息內容
此範例使用 MQHeaderList 來印出訊息的內容 (包括其標頭)。

輸出包含所有標頭內容以及訊息內文的視圖。 MQHeaderList 類別會一次解碼所有標頭，而 MQHeaderIterator 則會在應用程式控制下一次逐步執行一個標頭。 在撰寫 WebSphere MQ 應用程式時，您可以使用這項技術來提供簡式除錯工具。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderList;
...
MQMessage message = ... // Message received from a queue.

System.out.println (new MQHeaderList (message, true));
```

此範例也會使用 MQMD 類別來印出訊息描述子欄位。 com.ibm.mq.headers.MQMD 類別的 copyFrom() 方法會從 MQMessage 的訊息描述子欄位移入標頭物件，而不是透過讀取訊息內文來移入。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQMD;
import com.ibm.mq.headers.MQHeaderList;
...
MQMessage message = ...
MQMD md = new MQMD ();
...
md.copyFrom (message);
System.out.println (md + "\n" + new MQHeaderList (message, true));
```

使用 *IBM MQ classes for Java* 在訊息中尋找特定類型的標頭
此範例使用 MQHeaderList 的 indexOf(String) 方法來尋找訊息中的 MQRFH2 標頭 (如果有的話)。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderList;
import com.ibm.mq.headers.MQRFH2;
...
MQMessage message = ...
MQHeaderList list = new MQHeaderList (message);
int index = list.indexOf ("MQRFH2");

if (index >= 0)
{
    MQRFH2 rfh = (MQRFH2) list.get (index);
        ...
}
```

使用 *IBM MQ classes for Java* 分析 *MQRFH2* 標頭
此範例顯示如何使用 MQRFH2 類別來存取具名資料夾中的已知欄位值。

MQRFH2 類別提供許多方法，不僅可以存取結構固定部分中的欄位，還可以存取 NameValue 資料欄位中所包含的 XML 編碼資料夾內容。 此範例顯示如何存取具名資料夾中的已知欄位值-在此實例中為 jms 資料夾中的 Rto 欄位，代表 MQ JMS 訊息中的回覆佇列名稱。

```
MQRFH2 rfh = ...

String value = rfh.getStringFieldValue ("jms", "Rto");
```

若要探索 MQRFH2 的內容 (而不是直接要求特定欄位)，您可以使用 getFolders 方法來傳回 MQRFH2.Element，代表可包含欄位及其他資料夾之資料夾的結構。 將欄位或資料夾設為空值會從 MQRFH2 中移除它。 當您以這種方式操作 NameValueData 資料夾內容時，會相應地自動更新 StrucLength 欄位。

使用 *IBM MQ classes for Java* 讀取及寫入非 *MQMessage* 物件的位元組串流

當資料來源不是 MQMessage 物件時，這些範例會使用標頭類別來剖析及操作 IBM MQ 標頭內容。

即使資料來源不是 MQMessage 物件，您也可以使用標頭類別來剖析及操作 IBM MQ 標頭內容。 每個標頭類別所實作的 MQHeader 介面會提供方法 `int read (java.io.DataInput message, int encoding, int characterSet)` 及 `int write (java.io.DataOutput message, int encoding, int characterSet)`。 com.ibm.mq.MQMessage 類別會實作 java.io.DataInput 和 java.io.DataOutput 介面。 這表示您可以使用兩個 MQHeader 方法來讀取及寫入 MQMessage 內容，以置換訊息描述子中指定的編碼及 CCSID。 這對於包含不同編碼的標頭鏈的訊息很有用。

您也可以從其他資料串流 (例如檔案或 Socket 串流) 或 JMS 訊息中所包含的位元組陣列，取得 DataInput 和 DataOutput 物件。 java.io.DataInputStream 類別實作 DataInput，而 java.io.DataOutputStream 類別實作 DataOutput。 此範例會從位元組陣列讀取 IBM MQ 標頭內容:

```
import java.io.*;
import com.ibm.mq.headers.*;
...
byte [] bytes = ...
DataInput in = new DataInputStream (new ByteArrayInputStream (bytes));
MQHeaderIterator it = new MQHeaderIterator (in, CMQC.MQENC_NATIVE,
  CMQC.MQCCSI_DEFAULT);
```

以 MQHeaderIterator 開頭的行可以取代為

```
MQDLH dlh = new MQDLH (in, CMQC.MQENC_NATIVE, CMQC.MQCCSI_DEFAULT);
// or any other header type
```

此範例使用 DataOutput 串流寫入位元組陣列:

```
MQHeader header = ... // Could be any header type
ByteArrayOutputStream out = new ByteArrayOutputStream ();

header.write (new DataOutputStream (out), CMQC.MQENC_NATIVE, CMQC.MQCCSI_DEFAULT);
byte [] bytes = out.toByteArray ();
```

以這種方式使用串流時，請小心對編碼和 characterSet 引數使用正確的值。 讀取標頭時，請指定最初用來寫入位元組內容的編碼及 CCSID。 撰寫標頭時，請指定您要產生的編碼和 CCSID。 資料轉換由標頭類別自動執行。

使用 *IBM MQ classes for Java* 來建立新標頭類型的類別

您可以為 IBM MQ classes for Java 未提供的標頭類型建立 Java 類別。

若要新增代表新標頭類型的 Java 類別，您可以使用與隨 IBM MQ classes for Java 提供的任何標頭類別相同的方式來使用，您可以建立實作 MQHeader 介面的類別。 最簡單的方法是延伸 com.ibm.mq.headers.impl.Header 類別。 此範例會產生代表 MQTM 標頭結構的完整功能類別。 您不需要為每一個欄位新增個別的 getter 和 setter 方法，但對標頭類別的使用者來說，這是很方便的。 採用欄位名稱字串的通用 getValue 和 setValue 方法將適用於標頭類型中定義的所有欄位。 繼承的讀取、寫入及大小方法可讓新標頭類型的實例讀取及寫入，並根據其欄位定義正確計算標頭大小。 類型定義只會建立一次，不過會建立此標頭類別的許多實例。 若要讓新的標頭定義可使用 MQHeaderIterator 或 MQHeaderList 類別進行解碼，您可以使用 MQHeaderRegistry 來登錄它。 不過請注意， MQTM 標頭類別實際上已在此套件中提供，並登錄在預設登錄中。

```
import com.ibm.mq.headers.impl.Header;
import com.ibm.mq.headers.impl.HeaderField;
import com.ibm.mq.headers.CMQC;

public class MQTM extends Header {
    final static HeaderType TYPE = new HeaderType ("MQTM");
    final static HeaderField StrucId = TYPE.addMQChar ("StrucId", CMQC.MQTM_STRUC_ID);
    final static HeaderField Version = TYPE.addMQLong ("Version", CMQC.MQTM_VERSION_1);
    final static HeaderField QName = TYPE.addMQChar ("QName", CMQC.MQ_Q_NAME_LENGTH);
    final static HeaderField ProcessName = TYPE.addMQChar ("ProcessName",
      CMQC.MQ_PROCESS_NAME_LENGTH);
    final static HeaderField TriggerData = TYPE.addMQChar ("TriggerData",
      CMQC.MQ_TRIGGER_DATA_LENGTH);
    final static HeaderField ApplType = TYPE.addMQLong ("ApplType");
    final static HeaderField ApplId = TYPE.addMQChar ("ApplId", 256);
    final static HeaderField EnvData = TYPE.addMQChar ("EnvData", 128);
    final static HeaderField UserData = TYPE.addMQChar ("UserData", 128);

    protected MQTM (HeaderType type){
        super (type);
    }
    public String getStrucId () {
        return getStringValue (StrucId);
    }
    public int getVersion () {
        return getIntValue (Version);
    }
    public String getQName () {
        return getStringValue (QName);
    }
    public void setQName (String value) {
        setStringValue (QName, value);
    }
    // ...Add convenience getters and setters for remaining fields in the same way.
}
```

使用 *IBM MQ classes for Java* 處理 *PCF* 訊息

提供 Java 類別以建立及剖析 PCF 結構化訊息，並協助傳送 PCF 要求及收集 PCF 回應。

類別 PCFMessage 及 MQCFGR 代表 PCF 參數結構的陣列。 它們提供便利方法來新增及擷取 PCF 參數。

PCF 參數結構由類別 MQCFH、MQCFIN、 MQCFIN64、MQCFST、MQCFBS、MQCFIL、 MQCFIL64 MQCFSL 及 MQCFGR 表示。 這些共用基本作業介面:

- 讀取及寫入訊息內容的方法 :read ()、write () 及 size ()
- 操作參數的方法: getValue ()、 setValue ()、 getParameter () 及其他
- 列舉元方法。nextParameter () ，用於剖析 MQMessage 中的 PCF 內容

在 inquire 指令中使用 PCF 過濾參數，以提供過濾函數。 它封裝在下列類別中:

- MQCFIF-整數過濾器
- MQCFSF-字串過濾器
- MQCFBF 位元組過濾器

提供兩個代理程式類別 PCFAgent 及 PCFMessageAgent ，以管理與佇列管理程式、指令伺服器佇列及相關聯回應佇列的連線。 PCFMessageAgent 延伸 PCFAgent ，通常應該優先使用。 PCFMessageAgent 類別會轉換收到的 MQMessages ，並以 PCFMessage 陣列形式將它們傳回給呼叫者。 PCFAgent 會傳回 MQMessages 陣列，您必須先剖析才能使用。

處理 *IBM MQ classes for Java* 中的訊息內容

在 IBM MQ classes for Java 中，用來處理訊息控點的函數呼叫沒有對等項目。 若要設定、傳回或刪除訊息控點內容，請使用 MQMessage 類別的方法。

如需訊息內容的一般資訊，請參閱 第 23 頁的『內容名稱』。

在 IBM MQ classes for Java 中，對訊息的存取權是透過 MQMessage 類別。 因此，在 Java 環境中未提供訊息控點，且沒有相等於 IBM MQ 函數呼叫 MQCRTMH、MQDLTMH、MQMHBUF 及 MQBUFMH 的函數。

若要在程序化介面中設定訊息控點內容，請使用呼叫 MQSETMP。 在 IBM MQ classes for Java 中，使用 MQMessage 類別的適當方法:

- setBoolean 內容
- setByte 內容
- setBytes 內容
- setShort 內容
- setInt 內容
- setInt2Property
- setInt4Property
- setInt8Property
- setLong 內容
- setFloat 內容
- setDouble 內容
- setString 內容
- setObject 內容

這些有時統稱為 *set*property* 方法。

若要在程序化介面中傳回訊息控點內容的值，請使用呼叫 MQINQMP。 在 IBM MQ classes for Java 中，使用 MQMessage 類別的適當方法:

- getBoolean 內容
- getByte 內容
- getBytes 內容
- getShort 內容
- getInt 內容
- getInt2Property
- getInt4Property
- getInt8Property
- getLong 內容
- getFloat 內容
- getDouble 內容
- getString 內容
- getObject 內容

這些有時統稱為 *get*property* 方法。

若要刪除程序化介面中訊息控點內容的值，請使用呼叫 MQDLTMP。 在 IBM MQ classes for Java 中，使用 MQMessage 類別的 deleteProperty 方法。

處理 *IBM MQ classes for Java* 中的錯誤

使用 Java `try` 和 `catch` 區塊來處理 IBM MQ classes for Java 產生的錯誤。

Java 介面中的方法不會傳回完成碼和原因碼。 相反地，每當 IBM MQ 呼叫所產生的完成碼和原因碼都不是零時，它們就會擲出異常狀況。 這可簡化程式邏輯，讓您在每次呼叫 IBM MQ 之後都不需要檢查回覆碼。 您可以決定要在程式中的哪些點處理失敗的可能性。 在這些點，您可以用 `try` 和 `catch` 區塊括住程式碼，如下列範例所示:

```
try {
    myQueue.put(messageA,putMessageOptionsA);
    myQueue.put(messageB,putMessageOptionsB);
}
catch (MQException ex) {
```

```
    // This block of code is only executed if one of
    // the two put methods gave rise to a non-zero
    // completion code or reason code.
    System.out.println("An error occurred during the put operation:" +
                       "CC = " + ex.completionCode +
                       "RC = " + ex.reasonCode);
    System.out.println("Cause exception:" + ex.getCause() );
}
```

API 完成碼和原因碼中記載了在 Java z/OS 異常狀況中回報的 IBM MQ 呼叫原因碼。

執行 IBM MQ classes for Java 應用程式時擲出的異常狀況也會寫入日誌中。 不過，應用程式可以呼叫 MQException.logExclude() 方法，以防止記載與特定原因碼相關聯的異常狀況。 在您預期會擲出許多與特定原因碼相關聯的異常狀況，且您不想要日誌中填入這些異常狀況的情況下，您可能想要執行此動作。 例如，如果您的應用程式每次反覆運算迴圈時都嘗試從佇列中取得訊息，並且在大部分嘗試中，您預期佇列上不會有適當的訊息，則您可能想要防止記載與原因碼 MQRC_NO_MSG_AVAILABLE 相關聯的異常狀況。 如果應用程式先前已阻止記載與特定原因碼相關聯的異常狀況，則可以呼叫方法 MQException.logInclude() 來容許重新記載這些異常狀況。

有時原因碼不會傳達與錯誤相關聯的所有詳細資料。 對於所擲出的每一個異常狀況，應用程式應該檢查鏈結的異常狀況。 鏈結的異常狀況本身可能有另一個鏈結的異常狀況，因此鏈結的異常狀況會形成鏈結，導致回到原始基礎問題。 鏈結異常狀況是利用 java.lang.Throwable 類別的鏈結異常狀況機制來實作，應用程式會呼叫 Throwable.getCause() 方法來取得鏈結異常狀況。 從屬於 MQException 實例的異常狀況中，MQException.getCause() 會擷取 com.ibm.mq.jmqi.JmqiException 的基礎實例，而從此異常狀況中 getCause 會擷取導致錯誤的基礎 java.lang.Exception 。

在 *IBM MQ classes for Java* 中取得及設定屬性值

getXXX() 和 setXXX() 方法適用於許多共同屬性。 其他可以使用一般 inquire () 和 set () 方法來存取。

對於許多共同屬性，類別 MQManagedObject、MQDestination、MQQueue、MQTopic、MQProcess 及 MQQueueManager 包含 getXXX() 及 setXXX() 方法。 這些方法可讓您取得並設定其屬性值。 請注意，對於 MQDestination、MQQueue 及 MQTopic ，只有在您開啟物件時指定適當的查詢及設定旗標時，這些方法才會運作。

對於較不常見的屬性， MQQueueManager、MQDestination、MQQueue、MQTopic 及 MQProcess 類別都繼承自稱為 MQManagedObject 的類別。 這個類別定義 inquire () 和 set () 介面。

當您使用 *new* 運算子來建立新的佇列管理程式物件時，會自動開啟該物件以進行查詢。 當您使用 accessProcess() 方法來存取程序物件時，會自動開啟該物件進行查詢。 當您使用 accessQueue() 方法來存取佇列物件時，不會自動開啟該物件進行查詢或設定作業。 這是因為自動新增這些選項可能會導致部分類型的遠端佇列發生問題。 若要在佇列上使用 inquire、set、 getXXX 及 setXXX 方法，您必須在 accessQueue() 方法的 openOptions 參數中指定適當的 inquire 及 set 旗標。 目的地和主題物件也是如此。

inquire 及 set 方法採用三個參數:

- 選取元陣列
- intAttrs 陣列
- charAttrs 陣列

您不需要 MQINQ 中找到的 SelectorCount、 IntAttrCount 及 CharAttrLength 參數，因為 Java 中陣列的長度一律是已知的。 下列範例顯示如何對佇列進行查詢:

```
// inquire on a queue
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));
```

Java 中的多執行緒程式

Java 執行時期環境本質上是多執行緒。 IBM MQ classes for Java 容許多個執行緒共用佇列管理程式物件，但可確保對目標佇列管理程式的所有存取權都已同步。

在 Java 中很難避免使用多執行緒程式。 請考量連接至佇列管理程式並在啟動時開啟佇列的簡式程式。 程式會在畫面上顯示單一按鈕。 當使用者按一下該按鈕時，程式會從佇列中提取訊息。

Java 執行時期環境本質上是多執行緒。 因此，您的應用程式起始設定會在一個執行緒中進行，而為了回應按鈕而執行的程式碼會在個別執行緒 (使用者介面執行緒) 中執行。

使用 C 型 IBM MQ MQI client 時，這會導致問題，因為多個執行緒共用控點有一些限制。 IBM MQ classes for Java 會放寬此限制，容許多個執行緒共用佇列管理程式物件 (及其相關聯的佇列、主題及處理程序物件)。

IBM MQ classes for Java 的實作可確保針對特定連線 (MQQueueManager 物件實例) ，對目標 IBM MQ 佇列管理程式的所有存取權都會同步。 會封鎖想要對佇列管理程式發出呼叫的執行緒，直到該連線的所有其他進行中呼叫都完成為止。 如果您需要從程式內的多個執行緒同時存取相同的佇列管理程式，請為每一個需要並行存取的執行緒建立新的 MQQueueManager 物件。 (這相當於對每一個執行緒發出個別 MQCONN 呼叫。)

註: 類別 `com.ibm.mq.MQGetMessageOptions` 的實例不得在同時要求訊息的執行緒之間共用。 在對應的 MQGET 要求期間，會以資料來更新這個類別的實例，當多個執行緒同時在物件的相同實例上運作時，可能會產生非預期的結果。

在 *IBM MQ classes for Java* 中使用通道結束程式

如何在使用 IBM MQ classes for Java 的應用程式中使用通道結束程式的概觀。

下列主題說明如何在 Java 中撰寫通道結束程式、如何指派它，以及如何將資料傳遞給它。 然後說明如何使用以 C 撰寫的通道結束程式，以及如何使用一連串通道結束程式。

您的應用程式必須具有正確的安全許可權，才能載入通道結束程式類別。

在 *IBM MQ classes for Java* 中建立通道結束程式

您可以定義實作適當介面的 Java 類別，以提供您自己的通道結束程式。

若要實作結束程式，您可以定義新的 Java 類別來實作適當的介面。 com.ibm.mq.exits 套件中定義了三個結束程式介面:

- WMQSendExit
- WMQReceiveExit
- WMQSecurityExit

註: 只有用戶端連線才支援通道結束程式; 連結連線不支援通道結束程式。 例如，如果您使用以 C 撰寫的用戶端應用程式，則無法在 IBM MQ classes for Java 外部使用 Java 通道結束程式。

在 已呼叫 傳送及安全結束程式之後，會執行為連線定義的任何 TLS 加密。 同樣地，在 呼叫 接收及安全結束程式之前，會先執行解密。

下列範例定義實作所有三個介面的類別:

```
public class MyMQExits implements
WMQSendExit, WMQReceiveExit, WMQSecurityExit {
    // Default constructor
  public MyMQExits(){
  }
    // This method comes from the send exit interface
  public ByteBuffer channelSendExit(
MQCXP channelExitParms,
                                    MQCD channelDefinition,
                                    ByteBuffer agentBuffer)
  {
    // Fill in the body of the send exit here
  }
    // This method comes from the receive exit interface
  public ByteBuffer channelReceiveExit(
```

```
MQCXP channelExitParms,
                                        MQCD channelDefinition,
                                        ByteBuffer agentBuffer)
  {
    // Fill in the body of the receive exit here
  }
    // This method comes from the security exit interface
  public ByteBuffer channelSecurityExit(
MQCXP channelExitParms,
                                         MQCD channelDefinition,
                                         ByteBuffer agentBuffer)
  {
    // Fill in the body of the security exit here
  }
}
```

每一個結束程式都會傳遞一個 MQCXP 物件及一個 MQCD 物件。 這些物件代表程序化介面中定義的 MQCXP 和 MQCD 結構。

您撰寫的任何結束程式類別都必須有建構子。 這可以是預設建構子或採用字串引數的建構子。 如果採用字串，則使用者資料會在建立時傳遞至結束程式類別。 如果結束類別同時包含預設建構子和單一引數建構子，則單一引數建構子具有優先順序。

對於傳送及安全結束程式，您的結束碼必須傳回您要傳送至伺服器的資料。 對於接收結束程式，您的結束碼必須傳回您要 IBM MQ 解譯的已修改資料。

最簡單的可能結束主體為:

```
{ return agentBuffer; }
```

請勿從通道結束程式內關閉佇列管理程式。

使用現有的通道結束程式類別

在早於 7.0 的 IBM MQ 版本中，您將使用介面 MQSendExit、 MQReceiveExit 及 MQSecurityExit 來實作這些結束程式，如下列範例所示。 此方法仍然有效，但為了改良功能及效能，偏好使用新方法。

```
public class MyMQExits implements MQSendExit, MQReceiveExit, MQSecurityExit {
    // Default constructor
  public MyMQExits(){
  }
    // This method comes from the send exit
  public byte[] sendExit(MQChannelExit channelExitParms,
                         MQChannelDefinition channelDefParms,
                         byte agentBuffer[])
  {
    // Fill in the body of the send exit here
  }
    // This method comes from the receive exit
  public byte[] receiveExit(MQChannelExit channelExitParms,
                            MQChannelDefinition channelDefParms,
                            byte agentBuffer[])
  {
    // Fill in the body of the receive exit here
  }
    // This method comes from the security exit
  public byte[] securityExit(MQChannelExit channelExitParms,
                             MQChannelDefinition channelDefParms,
                             byte agentBuffer[])
  {
    // Fill in the body of the security exit here
  }
}
```

在 *IBM MQ classes for Java* 中指派通道結束程式

您可以使用 IBM MQ classes for Java 來指派通道結束程式。

IBM MQ classes for Java 中沒有直接相等於 IBM MQ 通道的通道。 通道結束程式會指派給 MQQueueManager。 例如，定義了實作 WMQSecurityExit 介面的類別之後，應用程式可以四種方式之一來使用安全結束程式:

- 在建立 MQQueueManager 物件之前，將類別的實例指派給 MQEnvironment.channelSecurityExit 欄位
- 在建立 MQQueueManager 物件之前，將 MQEnvironment.channelSecurityExit 欄位設為代表安全結束程式類別的字串
- 透過在傳遞至 MQQueueManager 的內容雜湊表中建立鍵值組，並指定 CMQC.SECURITY_EXIT_PROPERTY 索引鍵
- 使用用戶端通道定義表 (CCDT)

任何透過將 MQEnvironment.channelSecurityExit 欄位設為字串、在內容雜湊表中建立鍵/值配對或使用 CCDT 來指派的結束程式，都必須以預設建構子撰寫。 視應用程式而定，指派為類別實例的結束程式不需要預設建構子。

應用程式可以用類似的方式來使用傳送或接收結束程式。 例如，下列程式碼片段顯示如何使用在先前使用 MQEnvironment 定義的類別 MyMQExits 中實作的安全、傳送及接收結束程式:

```
MyMQExits myexits = new MyMQExits();
MQEnvironment.channelSecurityExit = myexits;
MQEnvironment.channelSendExit = myexits;
MQEnvironment.channelReceiveExit = myexits;
:
MQQueueManager jupiter = new MQQueueManager("JUPITER");
```

如果使用多個方法來指派通道結束程式，則優先順序如下:

1. 如果 CCDT 的 URL 傳遞至 MQQueueManager，則 CCDT 的內容會決定要使用的通道結束程式，並忽略 MQEnvironment 或內容雜湊表中的任何結束程式定義。
2. 如果未傳遞任何 CCDT URL，則會合併 MQEnvironment 和雜湊表中的結束程式定義
   - 如果 MQEnvironment 和雜湊表中都定義了相同的結束類型，則會使用雜湊表中的定義。
   - 如果指定相等的舊及新類型結束程式 (例如 sendExit 欄位 (只能用於 IBM WebSphere MQ 7.0 之前版本中使用的結束程式類型)，以及 channelSend 結束程式欄位 (可以用於任何傳送結束程式))，則會使用新結束程式 (channelSend 結束程式) 而非舊結束程式。

如果您已將通道結束程式宣告為字串，則必須啟用 IBM MQ 才能找到通道結束程式。 您可以根據應用程式執行所在的環境，以及通道結束程式的包裝方式，以各種方式來執行。

- 對於在應用程式伺服器中執行的應用程式，您必須將檔案儲存在 第 330 頁的表 58 所顯示的目錄中，或儲存在 **exitClasspath** 所參照的 JAR 檔中。
- 對於未在應用程式伺服器中執行的應用程式，下列規則適用:
  - 如果通道結束程式類別包裝在個別 JAR 檔中，則這些 JAR 檔必須包含在 **exitClasspath** 中。
  - 如果您的通道結束程式類別未包裝在 JAR 檔中，類別檔可以儲存在 第 330 頁的表 58 所顯示的目錄中，或儲存在 JVM 系統類別路徑或 **exitClasspath** 的任何目錄中。

您可以用四種方式來指定 **exitClasspath** 內容。 依優先順序，這些方式如下:

1. 系統內容 com.ibm.mq.exitClasspath (使用 -D 選項在指令行上定義)
2. `mqclient.ini` 檔案的 exitPath 段落
3. 具有索引鍵 CMQC.EXIT_CLASSPATH_PROPERTY
4. MQEnvironment 變數 **exitClasspath**

使用 java.io.File.pathSeparator 字元來區隔多個路徑。

表 *58:* 通道結束程式的目錄

平台	**目錄**
AIX Linux	`/var/mqm/exits` (32 位元通道結束程式) `/var/mqm/exits64` (64 位元通道結束程式)
Windows	*install_data_dir*\exits

註: *install_data_dir* 是您在安裝期間為 IBM MQ 資料檔案選擇的目錄。 預設目錄為 `C:\ProgramData\IBM\MQ`。

將資料傳遞至 *IBM MQ classes for Java* 中的通道結束程式
您可以將資料傳遞至通道結束程式，並將資料從通道結束程式傳回至應用程式。

agentBuffer 參數

對於傳送結束程式， *agentBuffer* 參數包含即將傳送的資料。 對於接收結束程式或安全結束程式， *agentBuffer* 參數包含剛收到的資料。 您不需要長度參數，因為表示式 agentBuffer.limit () 指出陣列的長度。

對於傳送及安全結束程式，您的結束碼必須傳回您要傳送至伺服器的資料。 對於接收結束程式，您的結束碼必須傳回您要 IBM MQ 解譯的已修改資料。

最簡單的可能結束主體為:

```
{ return agentBuffer; }
```

通道結束程式是以具有支持陣列的緩衝區來呼叫。 為了取得最佳效能，結束程式應該傳回具有支持陣列的緩衝區。

使用者資料

如果應用程式透過設定 channelSecurity 結束程式、 channelSend 結束程式或 channelReceive 結束程式來連接至佇列管理程式，則可以使用 channelSecurityExitUser 資料、 channelSendExitUser 資料或 channelReceiveExitUser 資料欄位，將 32 個位元組的使用者資料傳遞至適當的通道結束程式類別。 此使用者資料可供通道結束程式類別使用，但每次呼叫結束程式時都會重新整理。 因此，對通道結束程式中的使用者資料所做的任何變更都會遺失。 如果您想要對通道結束程式中的資料進行持續變更，請使用 MQCXP exitUser 區域。 在呼叫結束程式之間會維護此欄位中的資料。

如果應用程式設定 securityExit、 sendExit 或 receiveExit，則無法將使用者資料傳遞至這些通道結束程式類別。

如果應用程式使用用戶端通道定義表 (CCDT) 來連接至佇列管理程式，則在呼叫用戶端連線通道定義中指定的任何使用者資料時，都會傳遞至通道結束程式類別。 如需使用用戶端通道定義表的相關資訊，請參閱 第 316 頁的『搭配使用用戶端通道定義表與 IBM MQ classes for Java』。

將未在 *Java* 中寫入的通道結束程式與 *IBM MQ classes for Java* 搭配使用
如何使用從 Java 應用程式以 C 撰寫的通道結束程式。

在 IBM MQ 中，您可以將以 C 撰寫的通道結束程式名稱指定為傳遞至 MQEnvironment 物件或內容 Hashtable 中的 channelSecurityExit、 channelSendExit 或 channelReceiveExit 欄位的字串。 不過，您無法在以另一種語言撰寫的應用程式中使用以 Java 撰寫的通道結束程式。

以 `library(function)` 格式指定結束程式名稱，並確保依照 結束程式路徑中的說明來指定結束程式的位置。

如需如何在 C 中寫入通道結束程式的相關資訊，請參閱 第 805 頁的『傳訊通道的通道結束程式』。

在 *IBM MQ classes for Java* 中使用一系列通道傳送或接收結束程式
IBM MQ classes for Java 應用程式可以使用一連串連續執行的通道傳送或接收結束程式。

若要使用一連串傳送結束程式，應用程式可以建立包含傳送結束程式的「清單」或「字串」。 如果使用「清單」，則「清單」的每一個元素可以是下列任何一項:

- 實作 WMQSendExit 介面之使用者定義類別的實例
- 實作 MQSendExit 介面的使用者定義類別實例 (適用於以 Java 撰寫的傳送結束程式)
- MQExternalSendExit 類別的實例 (適用於未在 Java 中寫入的傳送結束程式)
- MQSendExitChain 類別的實例
- String 類別的實例

清單不能包含另一個清單。

應用程式可以用類似的方式來使用一系列接收結束程式。

如果使用「字串」，它必須由一或多個以逗點區隔的結束程式定義組成，每一個定義都可以是 Java 類別的名稱，或格式為 `library(function)` 的 C 程式。

然後，應用程式會在建立 MQQueueManager 物件之前，將 List 或 String 物件指派給 MQEnvironment.channelSendExit 欄位。

傳遞至結束程式之資訊的環境定義僅在結束程式的網域內。 例如，如果鏈結 Java 結束程式和 C 結束程式，則 Java 結束程式的存在對 C 結束程式沒有影響。

使用結束鏈類別

在 IBM WebSphere MQ 7.0 之前的版本中，提供了兩個類別以容許結束序列:

- MQSendExit 鏈，實作 MQSendExit 介面
- MQReceiveExit 鏈，實作 MQReceiveExit 介面

使用這些類別仍然有效，但建議使用新方法。 使用 IBM MQ Classes for Java 介面表示您的應用程式仍然具有與 `com.ibm.mq.jar` 的相依關係如果使用 `com.ibm.mq.exits` 套件中的新介面集，則沒有與 `com.ibm.mq.jar` 的相依關係。

為了使用一連串傳送結束程式，應用程式建立了物件清單，其中每一個物件都是下列其中一項:

- 實作 MQSendExit 介面的使用者定義類別實例 (適用於以 Java 撰寫的傳送結束程式)
- MQExternalSendExit 類別的實例 (適用於未在 Java 中寫入的傳送結束程式)
- MQSendExitChain 類別的實例

應用程式已建立「MQSendExit 鏈結」物件，方法是在建構子上傳遞此物件清單作為參數。 然後，應用程式會在建立 MQQueueManager 物件之前，將 MQSendExitChain 物件指派給 MQEnvironment.sendExit 欄位。

IBM MQ classes for Java 中的通道壓縮

壓縮通道上流動的資料可以改善通道效能並減少網路資料流量。 IBM MQ classes for Java 使用 IBM MQ 內建的壓縮功能。

使用 IBM MQ 提供的功能，您可以壓縮在訊息通道及 MQI 通道上流動的資料，而且在任一類型通道上，您可以彼此獨立壓縮標頭資料及訊息資料。 依預設，通道上不會壓縮任何資料。 如需通道壓縮的完整說明 (包括在 IBM MQ 中實作的方式)，請參閱 資料壓縮 (COMPMSG) 及 標頭壓縮 (COMPHDR)。

IBM MQ classes for Java 應用程式會建立 java.util.Collection 物件，來指定可用來壓縮用戶端連線上的標頭或訊息資料的技術。 每一個壓縮技術都是集合中的「整數」物件，應用程式將壓縮技術新增至集合的順序是用戶端連線啟動時與佇列管理程式協議壓縮技術的順序。 然後，應用程式可以將集合指派給 MQEnvironment 類別中的 hdrComp 清單欄位 (若為標頭資料) 或 msgComp 清單欄位 (若為訊息資料)。 當應用程式備妥時，它可以透過建立 MQQueueManager 物件來啟動用戶端連線。

下列程式碼片段說明所述的方法。 第一個程式碼片段顯示如何實作標頭資料壓縮:

```
Collection headerComp = new Vector();
headerComp.add(new Integer(CMQXC.MQCOMPRESS_SYSTEM));
:
MQEnvironment.hdrCompList = headerComp;
:
MQQueueManager qMgr = new MQQueueManager(QM);
```

第二個程式碼片段顯示如何實作訊息資料壓縮:

```
Collection msgComp = new Vector();
msgComp.add(new Integer(CMQXC.MQCOMPRESS_RLE));
msgComp.add(new Integer(CMQXC.MQCOMPRESS_ZLIBHIGH));

:
MQEnvironment.msgCompList = msgComp;
```

```
:
MQQueueManager qMgr = new MQQueueManager(QM);
```

在第二個範例中，當用戶端連線啟動時，會依序以 RLE、ZLIBHIGH 協議壓縮技術。 在 MQQueueManager 物件的生命期限內，無法變更所選的壓縮技術。

用戶端連線上的用戶端及佇列管理程式所支援的標頭及訊息資料壓縮技術，會以 MQChannelDefinition 物件的 hdrCompList 及 msgCompList 欄位中的集合形式傳遞至通道結束程式。 目前用於壓縮用戶端連線上的標頭及訊息資料的實際技術，會傳遞至 MQChannelExit 物件的 CurHdr 壓縮及 CurMsg 壓縮欄位中的通道結束程式。

如果在用戶端連線上使用壓縮，則在處理任何通道傳送結束程式之前會壓縮資料，並在處理任何通道接收結束程式之後擷取資料。 因此，傳送至傳送及接收結束程式的資料會處於壓縮狀態。

如需指定壓縮技術及可用的壓縮技術的相關資訊，請參閱 類別 com.ibm.mq.MQEnvironment 及 介面 com.ibm.mq.MQC。

在 *IBM MQ classes for Java* 中共用 *TCP/IP* 連線

可以建立 MQI 通道的多個實例，以共用單一 TCP/IP 連線。

在 IBM MQ classes for Java 中，您可以使用 MQEnvironment.sharingConversations 變數來控制可以共用單一 TCP/IP 連線的交談數。

SHARECNV 屬性是連線共用的最佳努力方法。 因此，當 SHARECNV 值大於 0 與 IBM MQ classes for Java 搭配使用時，不保證新的連線要求一律會共用已建立的連線。

IBM MQ classes for Java 中的連線儲存區

IBM MQ classes for Java 容許將備用連線儲存起來，以便重複使用。

IBM MQ classes for Java 為處理與 IBM MQ 佇列管理程式的多個連線的應用程式提供其他支援。 當不再需要連線時，可以將它儲存起來，稍後再重複使用，而不是將它毀損。 這可以為循序連接至任意佇列管理程式的應用程式及中介軟體提供大量效能加強功能。

IBM MQ 提供預設連線儲存區。 應用程式可以透過 MQEnvironment 類別來登錄及取消登錄記號，以啟動或取消啟動此連線儲存區。 當 IBM MQ classes for Java 建構 MQQueueManager 物件時，如果儲存區處於作用中，則它會搜尋此預設儲存區，並重複使用任何適當的連線。 當發生 MQQueueManager.disconnect () 呼叫時，基礎連線會回到儲存區。

或者，應用程式可以針對特定用途建構 MQSimpleConnectionManager 連線儲存區。 然後，應用程式可以在建構 MQQueueManager 物件期間指定該儲存區，或將該儲存區傳遞至 MQEnvironment 以用作預設連線儲存區。

若要防止連線使用太多資源，您可以限制 MQSimpleConnectionManager 物件可以處理的連線總數，並且可以限制連線儲存區的大小。 如果 JVM 內的連線有衝突需求，則設定限制很有用。

依預設， getMaxConnections () 方法會傳回零值，這表示 MQSimpleConnectionManager 物件可以處理的連線數沒有限制。 您可以使用 setMaxConnections () 方法來設定限制。 如果您設定限制並達到此限制，則進一步連線的要求可能會導致擲出 MQException ，原因碼為 MQRC_MAX_CONNS_LIMIT_REACHED。

控制 *IBM MQ classes for Java* 中的預設連線儲存區

此範例顯示如何使用預設連線儲存區。

請考量下列範例應用程式 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]);
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }
```

```
      }
}
```

MQApp1 會從指令行取得本端佇列管理程式的清單，依序連接至每一個佇列管理程式，並執行一些作業。 不過，當指令行多次列出相同的佇列管理程式時，只連接一次以及重複使用該連線會更有效率。

IBM MQ classes for Java 提供您可以用來執行此動作的預設連線儲存區。 若要啟用儲存區，請使用其中一個 MQEnvironment.addConnectionPoolToken() 方法。 若要停用儲存區，請使用 MQEnvironment.removeConnectionPoolToken()。

下列範例應用程式 MQApp2 在功能上與 MQApp1 相同，但只會連接至每一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]);
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }

         MQEnvironment.removeConnectionPoolToken(token);

      }
}
```

第一個粗體行會透過向 MQEnvironment 登錄 MQPoolToken 物件來啟動預設連線儲存區。

MQQueueManager 建構子現在會在此儲存區中搜尋適當的連線，而且只有在找不到現有連線時，才會建立與佇列管理程式的連線。 qmgr.disconnect() 呼叫會將連線傳回儲存區，供稍後重複使用。 這些 API 呼叫與範例應用程式 MQApp1 相同。

第二個強調顯示行會取消啟動預設連線儲存區，這會毀損儲存在儲存區中的任何佇列管理程式連線。 這很重要，因為否則應用程式會以儲存區中的一些即時佇列管理程式連線來終止。 此狀況可能會導致佇列管理程式日誌中出現錯誤。

如果應用程式使用用戶端通道定義表 (CCDT) 來連接至佇列管理程式，則 MQQueueManager 建構子會先搜尋表格，以找出適合的用戶端連線通道定義。 如果找到一個，建構子會在預設連線儲存區中搜尋可用於通道的連線。 如果建構子在儲存區中找不到適當的連線，它會在用戶端通道定義表中搜尋下一個適當的用戶端連線通道定義，並依照先前的說明繼續進行。 如果建構子完成用戶端通道定義表的搜尋，但在儲存區中找不到任何適當的連線，則建構子會啟動表格的第二次搜尋。 在這項搜尋期間，建構子會依序嘗試為每一個適當的用戶端連線通道定義建立新連線，並使用它所管理的第一個連線來建立。

預設連線儲存區會儲存最多 10 個未使用的連線，並將未使用的連線保持在作用中狀態最多 5 分鐘。 應用程式可以變更此項 (如需詳細資料，請參閱 第 335 頁的『在 IBM MQ classes for Java 中提供不同的連線儲存區』)。

應用程式可以建構自己的下列項目，而不是使用 MQEnvironment 來提供 MQPoolToken:

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

部分應用程式或中介軟體供應商提供 MQPoolToken 的子類別，以將資訊傳遞至自訂連線儲存區。 可以用這種方式來建構它們並傳遞至 addConnectionPoolToken() ，以便將額外資訊傳遞至連線儲存區。

IBM MQ classes for Java 中的預設連線儲存區及多個元件

此範例顯示如何從已登錄 MQPoolToken 物件的靜態集新增或移除 MQPoolTokens 。

MQEnvironment 會保留已登錄 MQPoolToken 物件的靜態集。 若要在此集中新增或移除 MQPoolTokens ，請使用下列方法:

- MQEnvironment.addConnectionPoolToken()

- MQEnvironment.removeConnectionPoolToken()

應用程式可能包含許多獨立存在的元件，並使用佇列管理程式來執行工作。 在這類應用程式中，每一個元件都應該將 MQPoolToken 新增至其生命期限所設定的 MQEnvironment。

例如，範例應用程式 MQApp3 會建立十個執行緒，並啟動每一個執行緒。 每一個執行緒都會登錄自己的 MQPoolToken，等待一段時間，然後連接至佇列管理程式。 在執行緒中斷連線之後，它會移除自己的 MQPoolToken。

當 MQPoolTokens 集中至少有一個記號時，預設連線儲存區會維持作用中，因此在這個應用程式的期間內，它會維持作用中。 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");
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }
         catch (MQException mqe) {System.err.println("Error occurred!");}
         catch (InterruptedException ie) {}

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

在 *IBM MQ classes for Java* 中提供不同的連線儲存區
此範例顯示如何使用類別 **com.ibm.mq.MQSimpleConnectionManager** 來提供不同的連線儲存區。

這個類別提供連線儲存區的基本機能，應用程式可以使用這個類別來自訂儲存區的行為。

實例化之後，可以在 MQQueueManager 建構子上指定 MQSimpleConnectionManager。 然後，MQSimpleConnectionManager 會管理建構的 MQQueueManager 基礎下的連線。 如果 MQSimpleConnectionManager 包含適合的儲存區連線，則會重複使用該連線，並在 MQQueueManager.disconnect () 呼叫之後傳回給 MQSimpleConnectionManager。

下列程式碼片段示範此行為:

```
   MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
   myConnMan.setActive(MQSimpleConnectionManager.MODE_ACTIVE);
   MQQueueManager qmgr=new MQQueueManager("my.qmgr.1", myConnMan);
   :
   : (do something with qmgr)
   :
   qmgr.disconnect();

   MQQueueManager qmgr2=new MQQueueManager("my.qmgr.1", myConnMan);
   :
```

```
    : (do something with qmgr2)
    :
    qmgr2.disconnect();
    myConnMan.setActive(MQSimpleConnectionManager.MODE_INACTIVE);
```

在第一個 MQQueueManager 建構子期間偽造的連線，儲存在 qmgr.disconnect() 呼叫之後的 myConnMan 中。 然後，在第二次呼叫 MQQueueManager 建構子期間會重複使用連線。

第二行會啟用 MQSimpleConnectionManager。 最後一行會停用 MQSimpleConnectionManager ，並毀損儲存區中保留的任何連線。 依預設， MQSimpleConnectionManager 位於 MODE_AUTO 中，本節稍後會說明。

MQSimpleConnectionManager 會以最近使用的基礎來配置連線，並以最近使用的基礎來毀損連線。 依預設，如果連線已 5 分鐘未使用，或儲存區中有超過 10 個未用連線，則會毀損該連線。 您可以呼叫 MQSimpleConnectionManager.setTimeout() 來變更這些值。

您也可以設定 MQSimpleConnectionManager 作為預設連線儲存區，以便在 MQQueueManager 建構子上未提供 Connection Manager 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.setMaxConnections(75);
         myConnMan.setMaxUnusedConnections(50);
         MQEnvironment.setDefaultConnectionManager(myConnMan);
         MQApp3.main(args);
      }
}
```

粗體行會建立並配置 MQSimpleConnectionManager 物件。 配置會執行下列動作:

- 結束一小時未使用的連線
- 將 myConnMan 管理的連線數目限制為 75
- 將儲存區中未用連線的數目限制為 50
- 設定 MODE_AUTO ，這是預設值。 這表示只有在儲存區是預設連線管理程式，且 MQEnvironment 所保留的 MQPoolTokens 集中至少有一個記號時，儲存區才會處於作用中。

然後，新的 MQSimpleConnectionManager 會設為預設連線管理程式。

在最後一行中，應用程式會呼叫 MQApp3.main()。 這會執行一些執行緒，其中每一個執行緒都獨立使用 IBM MQ 。 這些執行緒在偽造連線時使用 myConnMan。

JTA/JDBC 協調，使用 *IBM MQ classes for Java*

IBM MQ classes for Java 支援 MQQueueManager.begin () 方法，可讓 IBM MQ 作為資料庫的協調程式，以提供符合 JDBC 第 2 類或 JDBC 第 4 類標準的驅動程式。

並非所有平台都提供此支援。 若要檢查哪些平台支援 JDBC 協調，請參閱 IBM MQ 的系統需求。

若要使用 XA-JTA 支援，您必須使用特殊 JTA 交換器程式庫。 使用此程式庫的方法會因您是使用 Windows 或其他平台之一而有所不同。

在 *Windows* 上配置 *JTA/JDBC* 協調

XA 程式庫以 DLL 形式提供，名稱格式為 `jdbcxxx.dll`。

針對 IBM MQ for Windows 伺服器安裝，所提供的 `jdbcora12.dll` 提供與 Oracle 12C 的相容性。

在 Windows 系統上， XA 程式庫是以完整 DLL 提供。 此 DLL 的名稱是 `jdbcxxx.dll` ，其中 `xxx` 指出已編譯交換器檔案庫的資料庫。 此程式庫位於 IBM MQ classes for Java 安裝架構的 `java\lib\jdbc` 或 `java\lib64\jdbc` 目錄中。 您必須將 XA 程式庫 (也說明為切換載入檔) 宣告給佇列管理程式。 使用 IBM

MQ Explorer。 在 XA 資源管理程式下的佇列管理程式內容畫面中，指定交換器載入檔的詳細資料。 您必須只提供檔案庫的名稱。 例如：

若為 Db2 資料庫，請將 SwitchFile 欄位設為: dbcdb2

若為 Oracle 資料庫，請將 SwitchFile 欄位設為: jdbcora

附註:

1. Oracle 12C 僅在 IBM MQ for Windows 上受 IBM MQ classes for Java 支援。
2. 支援的 Oracle 12C 版本為 12.1.0.1.0 Enterprise Edition 及未來修正套件。
3. Oracle 64 位元資料庫 Windows 需要 32 位元 Oracle 用戶端。
4. 使用 IBM MQ classes for Java， IBM MQ 可以充當交易協調程式。 不過，無法參與 JTA 樣式交易。

在 *Windows* 以外的平台上配置 *JTA/JDBC* 協調
提供物件檔。 請使用所提供的 make 檔來鏈結適當的 make 檔，並使用配置檔將它宣告至佇列管理程式。

對於每一個資料庫管理系統， IBM MQ 提供兩個物件檔。 您必須鏈結一個物件檔以建立 32 位元切換檔案庫，並鏈結另一個物件檔以建立 64 位元切換檔案庫。 對於 Db2，每一個物件檔的名稱是 jdbcdb2.o，而對於 Oracle ，每一個物件檔的名稱是 jdbcora.o。

您必須使用 IBM MQ 隨附的適當 make 檔來鏈結每一個物件檔。 交換器媒體庫需要其他媒體庫，這些媒體庫可能儲存在不同系統上的不同位置。 不過，交換器程式庫無法使用程式庫路徑環境變數來尋找這些程式庫，因為交換器程式庫是由在 setuid 環境中執行的佇列管理程式所載入。 因此，提供的 make 檔可確保交換器程式庫包含這些程式庫的完整路徑名稱。

若要建立交換器程式庫，請輸入具有下列格式的 **make** 指令。 若要建立 32 位元交換器程式庫，請在 IBM MQ 安裝的 /java/lib/jdbc 目錄中輸入指令。 若要建立 64 位元切換程式庫，請在 /java/lib64/jdbc 目錄中輸入指令。

```
make DBMS
```

其中 *DBMS* 是您要為其建立交換器檔案庫的資料庫管理系統。 有效值為 db2 (若為 Db2) 及 oracle (若為 Oracle)。

註:

- 若要執行 32 位元應用程式，您必須為所使用的每一個資料庫管理系統建立 32 位元及 64 位元切換檔案庫。 若要執行 64 位元應用程式，您只需要建立 64 位元切換檔案庫。 對於 Db2，每一個交換器媒體庫的名稱都是 jdbcdb2 ，而對於 Oracle，每一個交換器媒體庫的名稱都是 jdbcora。 make 檔可確保 32 位元和 64 位元切換程式庫儲存在不同的 IBM MQ 目錄中。 32 位元交換器程式庫儲存在 /java/lib/jdbc 目錄中， 64 位元交換器程式庫儲存在 /java/lib64/jdbc 目錄中。
- 因為您可以在系統上的任何位置安裝 Oracle ，所以 make 檔會使用 **ORACLE_HOME** 環境變數來尋找 Oracle 的安裝位置。
- 如果 IBM MQ 安裝至預設位置以外的位置，請變更 make 檔中的 **MQ_INSTALLATION_PATH** 值。

建立 Db2 及/或 Oracle 的交換器程式庫之後，您必須向佇列管理程式宣告它們。 如果佇列管理程式配置檔 (qm.ini) 已包含 Db2 或 Oracle 資料庫的 XAResourceManager 段落，則必須將每一個段落中的 SwitchFile 項目取代為下列其中一項:

若為 Db2 資料庫

```
SwitchFile=jdbcdb2
```

若為 Oracle 資料庫

```
SwitchFile=jdbcora
```

請勿指定 32 位元或 64 位元切換檔案庫的完整路徑名稱。 僅指定檔案庫的名稱。

如果佇列管理程式配置檔尚未包含 Db2 或 Oracle 資料庫的 XAResourceManager 段落，或您想要新增其他 XAResourceManager 段落，請參閱 管理 IBM MQ ，以取得如何建構 XAResourceManager 段落的相關資訊。 不過，新 XAResourceManager 段落中的每一個 SwitchFile 項目必須與先前針對 Db2 或 Oracle 資料庫所說明的完全相同。 您也必須包括項目 ThreadOfControl=PROCESS。

在更新佇列管理程式配置檔並確定已設定所有適當的資料庫環境變數之後，您可以重新啟動佇列管理程式。

使用 *JTA/JDBC* 協調
依照所提供的範例來撰寫 API 呼叫的程式碼。

使用者應用程式的 API 呼叫基本順序如下:

```
qMgr = new MQQueueManager("QM1")
Connection con = qMgr.getJDBCConnection( xads );
qMgr.begin()

< Perform MQ and DB operations to be grouped in a unit of work >

qMgr.commit() or qMgr.backout();
con.close()
qMgr.disconnect()
```

getJDBCConnection 呼叫中的 xads 是 XADataSource 介面的資料庫特定實作，定義要連接之資料庫的詳細資料。 請參閱資料庫的文件，以判斷如何建立適當的 XADataSource 物件，以傳遞至 getJDBCConnection。

您也必須以適當的資料庫特定 Jar 檔來更新類別路徑，以執行 JDBC 工作。

如果您必須連接至多個資料庫，則必須多次呼叫 getJDBCConnection ，以跨數個不同的連線執行交易。

getJDBCConnection 有兩種形式，反映兩種形式的 XADataSource。getXAConnection:

```
public java.sql.Connection getJDBCConnection(javax.sql.XADataSource xads)
  throws MQException, SQLException, Exception

public java.sql.Connection getJDBCConnection(XADataSource dataSource,
                                             String userid, String password)
  throws MQException, SQLException, Exception
```

這些方法在其 throws 子句中宣告「異常狀況」，以避免未使用 JTA 函數之客戶的 JVM 驗證器發生問題。 擲出的實際異常狀況是 javax.transaction.xa.XAException ，它需要將 jta.jar 檔新增至先前不需要它的程式的類別路徑。

如果要使用 JTA/JDBC 支援，您必須在應用程式中包含下列陳述式:

```
MQEnvironment.properties.put(CMQC.THREAD_AFFINITY_PROPERTY, new Boolean(true));
```

JTA/JDBC 協調的已知問題和限制
JTA/JDBC 支援的部分問題和限制取決於使用中的資料庫管理系統，例如，在應用程式執行期間關閉資料庫時，已測試的 JDBC 驅動程式會有不同的行為。 如果應用程式正在使用的資料庫連線中斷，則應用程式可以執行一些步驟來重新建立與佇列管理程式及資料庫的新連線，以便它可以使用那些新連線來執行所需的交易式工作。

因為 JTA/JDBC 支援中心會呼叫 JDBC 驅動程式，這些 JDBC 驅動程式的實作可能會對系統行為產生重大影響。 特別是在應用程式執行期間關閉資料庫時，受測試的 JDBC 驅動程式會有不同的行為。

重要: 當有應用程式保留開啟的資料庫連線時，請一律避免突然關閉資料庫。

註: IBM MQ classes for Java 應用程式必須使用連結模式來連接， IBM MQ 才能作為資料庫協調程式。

多個 XAResourceManager 段落
不支援在佇列管理程式配置檔 qm.ini 中使用多個 XAResourceManager 段落。 除了第一個以外的任何 XAResourceManager 段落都會被忽略。

Db2

有時 Db2 會傳回 SQL0805N 錯誤。 可以使用下列 CLP 指令來解決此問題:

```
DB2 bind @db2cli.lst blocking all grant public
```

如需相關資訊，請參閱 Db2 文件。

XAResourceManager 段落必須配置成使用 ThreadOfControl=PROCESS。 對於 Db2 8.1 及更高版本，這不符合 Db2 的控制項設定的預設執行緒，因此必須在 XA 開放式字串中指定 toc=p。 具有 JTA/JDBC 協調的 Db2 的 XAResourceManager 段落範例如下:

```
XAResourceManager:
     Name=jdbcdb2
     SwitchFile=jdbcdb2
     XAOpenString=uid=userid,db=dbalias,pwd=password,toc=p
     ThreadOfControl=PROCESS
```

這不會阻止使用 JTA/JDBC 協調的 Java 應用程式本身成為多執行緒。

Oracle

在 MQQueueManager.disconnect () 之後呼叫 JDBC Connection.close() 方法會產生 SQLException。 在 MQQueueManager.disconnect () 之前呼叫 Connection.close() ，或省略 Connection.close() 的呼叫。

處理資料庫連線的問題

當 IBM MQ classes for Java 應用程式使用 IBM MQ 所提供的 JTA/JDBC 支援時，它通常會執行下列步驟:

1. 建立新的 MQQueueManager 物件，以代表與將作為交易管理程式之佇列管理程式的連線。
2. 建構 XADataSource 物件，其中包含如何連接至將列入交易中的資料庫的詳細資料。
3. 呼叫 MQQueueManager 方法。getJDBCConnection(XADataSource) 會傳入先前建立的 XADataSource 。 這會導致 IBM MQ classes for Java 建立與資料庫的連線。
4. 呼叫 MQQueueManager.begin () 方法來啟動 XA 交易。
5. 執行傳訊及資料庫工作。
6. 當所有必要的工作都已完成時，會呼叫 MQQueueManager.commit () 方法。 這會完成 XA 交易。
7. 如果此時需要新的 XA 交易，應用程式可以重複步驟 4、5 和 6。
8. 當應用程式完成時，它應該關閉在步驟 3 建立的資料庫連線，然後呼叫 MQQueueManager.disconnect () 方法來切斷與佇列管理程式的連線。

IBM MQ classes for Java 會維護應用程式呼叫 MQQueueManager.getJDBCConnection(XADataSource) 時所建立之所有資料庫連線的內部清單。 如果佇列管理程式在處理 XA 交易期間需要與資料庫通訊，則會進行下列處理:

1. 佇列管理程式會呼叫 IBM MQ classes for Java，並傳入需要傳遞至資料庫之 XA 呼叫的詳細資料。
2. 然後， IBM MQ classes for Java 會在清單中查閱適當的連線，然後使用該連線將 XA 呼叫傳送至資料庫。

如果在此處理期間的任何時間點失去與資料庫的連線，則應用程式應該:

1. 呼叫 MQQueueManager.backout () 方法，以取消在交易下完成的任何現有工作。
2. 關閉資料庫連線。 這應該會導致 IBM MQ classes for Java 從其內部清單中移除已中斷資料庫連線的詳細資料。
3. 呼叫 MQQueueManager.disconnect () 方法來切斷佇列管理程式的連線。
4. 建構新的 MQQueueManager 物件，以建立與佇列管理程式的新連線。
5. 透過呼叫方法 MQQueueManager.getJDBCConnection(XADataSource) 來建立新的資料庫連線。
6. 重新執行交易式工作。

這可讓應用程式重新建立與佇列管理程式及資料庫的新連線，然後使用那些連線來執行所需的交易式工作。

IBM MQ classes for Java 中的傳輸層安全 *(TLS)* 支援

IBM MQ classes for Java 用戶端應用程式支援 TLS 加密。 您需要 JSSE 提供者才能使用 TLS 加密。

使用 TRANSPORT (CLIENT) 的 IBM MQ classes for Java 用戶端應用程式支援 TLS 加密。 TLS 提供通訊加密、鑑別及訊息完整性。 它通常用來保護網際網路或企業內部網路中任何兩個對等節點之間的通訊安全。

IBM MQ classes for Java 使用 Java Secure Socket Extension (JSSE) 來處理 TLS 加密，因此需要 JSSE 提供者。 JSE v1.4 JVM 具有內建的 JSSE 提供者。 如何管理及儲存憑證的詳細資料可能因提供者而異。 如需相關資訊，請參閱 JSSE 提供者的說明文件。

本節假設您的 JSSE 提供者已正確安裝和配置，且已安裝適當的憑證並可供 JSSE 提供者使用。

如果您的 IBM MQ classes for Java 用戶端應用程式使用用戶端通道定義表 (CCDT) 來連接佇列管理程式，請參閱 第 316 頁的『搭配使用用戶端通道定義表與 IBM MQ classes for Java』。

在 *IBM MQ classes for Java* 中啟用 *TLS*

若要啟用 TLS，請指定 CipherSuite。 有兩種方法可以指定 CipherSuite。

只有用戶端連線才支援 TLS。 若要啟用 TLS，您必須指定與佇列管理程式通訊時要使用的 CipherSuite，且此 CipherSuite 必須符合目標通道上設定的 CipherSpec。 此外， JSSE 提供者必須支援指名的 CipherSuite。 不過， CipherSuites 不同於 CipherSpecs，因此具有不同的名稱。 第 343 頁的『IBM MQ classes for Java 中的 TLS CipherSpecs 及 CipherSuites』 包含一個表格，將 IBM MQ 支援的 CipherSpecs 對映至 JSSE 已知的對等 CipherSuites。

若要啟用 TLS，請使用 MQEnvironment 的 sslCipher 套組靜態成員變數來指定 CipherSuite。 下列範例連接至名為 SECURE.SVRCONN.CHANNEL 的 SVRCONN 通道，該通道已設定為需要使用 CipherSpec 為 TLS_RSA_WITH_AES_128_CBC_SHA256: 的 TLS

```
MQEnvironment.hostname       = "your_hostname";
MQEnvironment.channel        = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.sslCipherSuite = "SSL_RSA_WITH_AES_128_CBC_SHA256";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

雖然通道的 CipherSpec 為 TLS_RSA_WITH_AES_128_CBC_SHA256，但 Java 應用程式必須將 CipherSuite 指定為 SSL_RSA_WITH_AES_128_CBC_SHA256。 如需 CipherSpecs 與 CipherSuites 之間的對映清單，請參閱 第 343 頁的『IBM MQ classes for Java 中的 TLS CipherSpecs 及 CipherSuites』。

應用程式也可以透過設定環境內容 CMQC.SSL_CIPHER_SUITE_PROPERTY 來指定 CipherSuite。

或者，使用「用戶端通道定義表 (CCDT)」。 如需相關資訊，請參閱 第 316 頁的『搭配使用用戶端通道定義表與 IBM MQ classes for Java』

如果您需要用戶端連線才能使用 IBM Java JSSE FIPS 提供者 (IBMJSSEFIPS) 所支援的 CipherSuite，應用程式可以將 MQEnvironment 類別中的 sslFips「必要」欄位設為 `true`。 或者，應用程式可以設定環境內容 CMQC.SSL_FIPS_REQUIRED_PROPERTY。 預設值為 `false`，表示用戶端連線可以使用 IBM MQ 支援的任何 CipherSuite。

如果應用程式使用多個用戶端連線，當應用程式建立第一個用戶端連線時所使用的 sslFips 必要欄位值會決定應用程式建立任何後續用戶端連線時所使用的值。 因此，當應用程式建立後續用戶端連線時，會忽略 sslFips 必要欄位的值。 如果您想要對 sslFips「必要」欄位使用不同的值，則必須重新啟動應用程式。

若要使用 TLS 順利連接，必須使用憑證管理中心主要憑證來設定 JSSE 信任儲存庫，可以從中鑑別佇列管理程式所提供的憑證。 同樣地，如果 SVRCONN 通道上的 SSLClientAuth 已設為 MQSSL_CLIENT_AUTH_REQUIRED， JSSE 金鑰儲存庫必須包含佇列管理程式所信任的識別憑證。

相關參考

AIX, Linux, and Windows 的聯邦資訊存取安全標準 (FIPS)

在 *IBM MQ classes for Java* 中使用佇列管理程式的識別名稱

佇列管理程式會使用包含識別名稱 (DN) 的 TLS 憑證來識別自己。 IBM MQ classes for Java 用戶端應用程式可以使用此 DN 來確保它與正確的佇列管理程式通訊。

DN 型樣是使用 MQEnvironment 的 sslPeerName 變數來指定。 例如，設定:

```
MQEnvironment.sslPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

只有在佇列管理程式呈現「通用名稱」開頭為 QMGR 的憑證時，連線才會成功。 及至少兩個組織單位名稱，第一個必須是 IBM ，第二個必須是 WebSphere。

如果設定 sslPeer 名稱，則只有在設為有效型樣且佇列管理程式呈現相符憑證時，連線才會成功。

應用程式也可以透過設定環境內容 CMQC.SSL_PEER_NAME_PROPERTY 來指定佇列管理程式的識別名稱。 如需識別名稱的相關資訊，請參閱 識別名稱。

在 *IBM MQ classes for Java* 中使用憑證撤銷清冊

透過 java.security.cert.CertStore 類別指定要使用的憑證撤銷清冊。 然後 IBM MQ classes for Java 會根據指定的 CRL 來檢查憑證。

憑證撤銷清冊 (CRL) 是已由簽發憑證管理中心或本端組織撤銷的一組憑證。 CRL 通常是在 LDAP 伺服器上管理。 使用 Java 2 v1.4，可以在連接時指定 CRL 伺服器，並在容許連線之前根據 CRL 來檢查佇列管理程式提供的憑證。 如需憑證撤銷清冊和 IBM MQ 的相關資訊，請參閱 使用憑證撤銷清冊和權限撤銷清冊 ，以及 使用 IBM MQ classes for Java 和 IBM MQ classes for JMS 來存取 CRL 和 ARL。

註: 若要順利搭配使用 CertStore 與 LDAP 伺服器上管理的 CRL ，請確定 Java Software Development Kit (SDK) 與 CRL 相容。 部分 SDK 需要 CRL 符合 RFC 2587 ，這會定義 LDAP v2 的綱目。 大部分 LDAP v3 伺服器改用 RFC 2256。

要使用的 CRL 是透過 java.security.cert.CertStore 類別來指定。 如需如何取得 CertStore 實例的完整詳細資料，請參閱此類別的文件。 若要根據 LDAP 伺服器建立 CertStore ，請先建立「 LDAPCertStore 參數」實例，並以要使用的伺服器和埠設定來起始設定。 例如：

```
import java.security.cert.*;
CertStoreParameters csp = new LDAPCertStoreParameters("crl_server", 389);
```

建立「 CertStore 參數」實例之後，請使用 CertStore 上的靜態建構子來建立 LDAP 類型的 CertStore :

```
CertStore cs = CertStore.getInstance("LDAP", csp);
```

也支援其他 CertStore 類型 (例如， Collection)。 通常會有數個 CRL 伺服器設定相同的 CRL 資訊，以提供備援。 當您有這些 CRL 伺服器的 CertStore 物件時，請將它們全部放在適當的集合中。 下列範例顯示放置在 ArrayList 中的 CertStore 物件:

```
import java.util.ArrayList;
Collection crls = new ArrayList();
crls.add(cs);
```

在連接以啟用 CRL 檢查之前，可以將此集合設為 MQEnvironment 靜態變數 sslCertStores:

```
MQEnvironment.sslCertStores = crls;
```

設定連線時佇列管理程式所提供的憑證驗證如下:

1. sslCertStores 所識別「集合」中的第一個 CertStore 物件用來識別 CRL 伺服器。
2. 嘗試聯絡 CRL 伺服器。
3. 如果嘗試成功，則會搜尋伺服器以找出憑證的相符項。
   a. 如果發現要撤銷憑證，則搜尋程序會結束，且連線要求會失敗，原因碼為 MQRC_SSL_CERTIFICATE_REVOKED。
   b. 如果找不到憑證，則會結束搜尋程序，並容許繼續進行連線。
4. 如果嘗試聯絡伺服器失敗，則會使用下一個 CertStore 物件來識別 CRL 伺服器，且處理程序會從步驟 2 重複。

如果這是「集合」中的最後一個 CertStore，或如果「集合」未包含 CertStore 物件，則搜尋程序會失敗，且連線要求會失敗，原因碼為 MQRC_SSL_CERT_STORE_ERROR。

Collection 物件會決定 CertStores 的使用順序。

也可以使用 CMQC.SSL_CERT_STORE_PROPERTY 來設定 CertStores 的集合。 為了方便起見，此內容也容許指定單一 CertStore，而不作為「集合」的成員。

如果 sslCert 儲存庫設為空值，則不會執行 CRL 檢查。 如果未設定 sslCipher 套組，則會忽略此內容。

在 *IBM MQ classes for Java* 中重新協議秘密金鑰

IBM MQ classes for Java 用戶端應用程式可以根據傳送及接收的位元組總數，控制何時重新協議用戶端連線上用於加密的秘密金鑰。

應用程式可以使用下列其中一種方式來執行此動作: 如果應用程式使用多個這些方式，則適用一般優先順序規則。

- 透過在 MQEnvironment 類別中設定 sslReset 計數欄位。
- 透過在 Hashtable 物件中設定環境內容 MQC.SSL_RESET_COUNT_PROPERTY。 然後應用程式會將雜湊表指派給 MQEnvironment 類別中的 `properties` 欄位，或將雜湊表傳遞給其建構子上的 MQQueueManager 物件。

sslReset 計數欄位或環境內容 MQC.SSL_RESET_COUNT_PROPERTY 的值代表在重新協議秘密金鑰之前，IBM MQ classes for Java 用戶端程式碼所傳送及接收的位元組總數。 傳送的位元組數是加密之前的數目，而接收的位元組數是解密之後的數目。 位元組數也包括 IBM MQ classes for Java 用戶端所傳送及接收的控制資訊。

如果重設計數為零 (預設值)，則永不重新協議秘密金鑰。 如果未指定 CipherSuite，則會忽略重設計數。

在 *IBM MQ classes for Java* 中提供自訂的 *SSLSocketFactory*

如果您使用自訂的 JSSE Socket Factory，請將 MQEnvironment.sslSocketFactory 設為自訂的 Factory 物件。 不同 JSSE 實作之間的詳細資料有所不同。

不同的 JSSE 實作可以提供不同的特性。 例如，特殊化 JSSE 實作可能容許配置特定加密硬體型號。 此外，部分 JSSE 提供者容許程式自訂金鑰儲存庫及信任儲存庫，或容許變更金鑰儲存庫中的身分憑證選擇。 在 JSSE 中，所有這些自訂作業都會抽象成 Factory 類別 javax.net.ssl.SSLSocketFactory。

如需如何建立自訂 SSLSocketFactory 實作的詳細資料，請參閱 JSSE 文件。 各提供者的詳細資料各不相同，但一般步驟順序可能如下:

1. 在 SSLContext 上使用 static 方法來建立 SSLContext 物件
2. 以適當的 KeyManager 和 TrustManager 實作 (從自己的 Factory 類別建立) 來起始設定這個 SSLContext
3. 從 SSLContext 建立 SSLSocketFactory

當您有 SSLSocketFactory 物件時，請將 MQEnvironment.sslSocketFactory 設為自訂的 Factory 物件。 例如:

```
javax.net.ssl.SSLSocketFactory sf = sslContext.getSocketFactory();
MQEnvironment.sslSocketFactory = sf;
```

IBM MQ classes for Java 使用此 SSLSocketFactory 來連接 IBM MQ 佇列管理程式。 也可以使用 CMQC.SSL_SOCKET_FACTORY_PROPERTY 來設定此內容。 如果 sslSocketFactory 設為空值，則會使用 JVM 的預設 SSLSocketFactory。 如果未設定 sslCipher 套組，則會忽略此內容。

當您使用自訂 SSLSocketFactories 時，請考量 TCP/IP 連線共用的影響。 如果可以共用連線，則不會要求所提供 SSLSocketFactory 的新 Socket，即使在後續連線要求的環境定義中，所產生的 Socket 在某些方面會有所不同。 例如，如果要在後續連線上呈現不同的用戶端憑證，則不得容許連線共用。

在 *IBM MQ classes for Java* 中變更 *JSSE* 金鑰儲存庫或信任儲存庫

如果您變更 JSSE 金鑰儲存庫或信任儲存庫，則必須執行某些動作，變更才會生效。

如果您變更 JSSE 金鑰儲存庫或信任儲存庫的內容，或變更金鑰儲存庫或信任儲存庫檔案的位置，則當時執行的 IBM MQ classes for Java 應用程式不會自動挑選變更。 若要讓變更生效，必須執行下列動作:

- 應用程式必須關閉其所有連線，並毀損連線儲存區中任何未用的連線。
- 如果 JSSE 提供者從金鑰儲存庫和信任儲存庫快取資訊，則必須重新整理此資訊。

執行這些動作之後，應用程式就可以重建其連線。

視您設計應用程式的方式以及 JSSE 提供者所提供的功能而定，可能可以執行這些動作，而不需要停止並重新啟動應用程式。 不過，停止並重新啟動應用程式可能是最簡單的解決方案。

搭配使用 *TLS* 與 *IBM MQ classes for Java* 時的錯誤處理

使用 TLS 連接至佇列管理程式時， IBM MQ classes for Java 可以發出許多原因碼。

這些在下列清單中說明:

MQRC_SSL_NOT_ALLOWED
已設定 sslCipher 套組內容，但已使用連結連接。 只有用戶端連接才支援 TLS。

MQRC_JSSE_ERROR
JSSE 提供者報告了 IBM MQ 無法處理的錯誤。 這可能是因為 JSSE 的配置問題，或因為無法驗證佇列管理程式所提供的憑證。 可以在 MQException 上使用 getCause() 方法來擷取 JSSE 所產生的異常狀況。

MQRC_SSL_INITIALIZATION_ERROR
已發出 MQCONN 或 MQCONNX 呼叫並指定 TLS 配置選項，但在 TLS 環境起始設定期間發生錯誤。

MQRC_SSL_PEER_NAME_MISMATCH
sslPeer 名稱內容中指定的 DN 型樣不符合佇列管理程式所提供的 DN。

MQRC_SSL_PEER_NAME_ERROR
sslPeer 名稱內容中指定的 DN 型樣無效。

MQRC_UNSUPPORTED_CIPHER_SUITE
JSSE 提供者無法辨識 sslCipherSuite 中指定的 CipherSuite 。 JSSE 提供者支援的 CipherSuites 完整清單可由程式使用 SSLSocketFactory 取得。getSupportedCipherSuites() 方法。 可以在 第 343 頁的『IBM MQ classes for Java 中的 TLS CipherSpecs 及 CipherSuites』中找到可用來與 IBM MQ 通訊的 CipherSuites 清單。

MQRC_SSL_CERTIFICATE_REVOKED
在使用 sslCertStores 內容指定的 CRL 中找到佇列管理程式所提供的憑證。 更新佇列管理程式以使用授信憑證。

MQRC_SSL_CERT_STORE_ERROR
無法在任何提供的 CertStores 中搜尋佇列管理程式所提供的憑證。 MQException.getCause() 方法傳回搜尋第一次嘗試 CertStore 時發生的錯誤。 如果原因異常狀況是 NoSuchElementException、ClassCastException 或 NullPointerException ，請檢查 sslCertStores 內容上指定的「集合」是否至少包含一個有效的 CertStore 物件。

IBM MQ classes for Java 中的 *TLS CipherSpecs* 及 *CipherSuites*

IBM MQ classes for Java 應用程式建立佇列管理程式連線的能力，取決於在 MQI 通道伺服器端指定的 CipherSpec 及在用戶端指定的 CipherSuite 。

下表列出 IBM MQ 支援的 CipherSpecs 及其對等的 CipherSuites。

Deprecated 您應該檢閱 已淘汰 CipherSpecs 主題，以查看下列表格中列出的任何 CipherSpecs 是否已被 IBM MQ 淘汰，如果已淘汰，則會在其中更新 CipherSpec 。

重要: 列出的 CipherSuites 是 IBM MQ 隨附的 IBM Java 執行時期環境 (JRE) 所支援的那些 CipherSuite。 列出的 CipherSuites 包括 Oracle Java JRE 所支援的那些 CipherSuite。 如需將應用程式配置成使用 Oracle Java JRE 的相關資訊，請參閱 第 360 頁的『將應用程式配置成使用 IBM Java 或 Oracle Java CipherSuite 對映』。

此表格也指出用於通訊的通訊協定，以及 CipherSuite 是否符合 FIPS 140-2 標準。

註: 在 AIX, Linux, and Windows 上， IBM MQ 透過 IBM Crypto for C (ICC) 加密模組提供 FIPS 140-2 相符性。 此模組的憑證已移至「歷程」狀態。 客戶應該檢視 IBM Crypto for C (ICC) 憑證 ，並注意 NIST 提供的任何建議。 目前正在進行取代 FIPS 140-3 模組，您可以在 處理程序清單中的 NIST CMVP 模組中搜尋它，以檢視其狀態。

如果應用程式尚未配置為施行 FIPS 140-2 相符性，但如果已針對應用程式配置 FIPS 140-2 相符性 (請參閱下列配置注意事項)，則只能配置標示為 FIPS 140-2 相容的 CipherSuites；嘗試使用其他 CipherSuites 會導致錯誤。

註: 每一個 JRE 都可以有多個加密安全提供者，每一個提供者都可以提供相同 CipherSuite 的實作。 不過，並非所有安全提供者都經過 FIPS 140-2 認證。 如果應用程式未施行 FIPS 140-2 相符性，則可能使用未經認證的 CipherSuite 實作。 未經認證的實作可能不符合 FIPS 140-2 標準運作，即使 CipherSuite 理論上符合標準所需的最低安全層次也一樣。 如需在 IBM MQ Java 應用程式中配置 FIPS 140-2 強制執行的相關資訊，請參閱下列注意事項。

如需 CipherSpecs 及 CipherSuites 的 FIPS 140-2 及 Suite-B 相符性的相關資訊，請參閱 指定 CipherSpecs。 您也可能需要注意有關美國 聯邦資訊存取安全標準的資訊。

若要使用完整的 CipherSuites 集，並使用經認證的 FIPS 140-2 及/或 Suite-B 標準來操作，需要適合的 JRE。 IBM Java 7 Service Refresh 4 Fix Pack 2 或更高層次的 IBM JRE 提供對 第 344 頁的表 59 中列出的 TLS 1.2 CipherSuites 的適當支援。

若要能夠使用 TLS 1.3 密碼，執行應用程式的 JRE 必須支援 TLS 1.3。

註: 若要使用部分 CipherSuites，需要在 JRE 中配置「未限定」原則檔。 如需如何在 SDK 或 JRE 中設定原則檔的詳細資料，請參閱您所使用版本的 *Security Reference for IBM SDK*， *Java Technology Edition* 中的 *IBM SDK* 原則檔 主題。

表 *59: IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs*

CipherSpec 第 359 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
ECDHE_ECDSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	yes

表 59: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (*繼續*)*

CipherSpec 第 359 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
ECDHE_ECDSA_AES_128_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS 1.2	yes

表 59: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (繼續)*

CipherSpec 第 359 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
ECDHE_ECDSA_AES_128_GCM_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS 1.2	yes

表 59: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (*繼續*)*

CipherSpec 第 359 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
ECDHE_ECDSA_AES_256_CBC_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS 1.2	yes

表 59: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (繼續)*

CipherSpec 第 359 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
ECDHE_ECDSA_AES_256_GCM_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS 1.2	yes
ECDHE_ECDSA_NULL_SHA256	SSL_ECDHE_ECDSA_WITH_NULL_SHA	TLS_ECDHE_ECDSA_WITH_NULL_SHA	TLS 1.2	否

表 59: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (*繼續*)*

CipherSpec 第 359 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS 1.2	否
ECDHE_RSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	yes

表 59: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (繼續)*

CipherSpec 第 359 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
ECDHE_RSA_AES_128_CBC_SHA256	SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	yes
ECDHE_RSA_AES_128_GCM_SHA256	SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	yes

表 59: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (繼續)*

CipherSpec 第 359 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
ECDHE_RSA_AES_256_CBC_SHA384	SSL_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS 1.2	yes
ECDHE_RSA_AES_256_GCM_SHA384	SSL_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	yes

表 59: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (繼續)*

CipherSpec 第 359 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
ECDHE_RSA_NULL_SHA256	SSL_ECDHE_RSA_WITH_NULL_SHA	TLS_ECDHE_RSA_WITH_NULL_SHA	TLS 1.2	否
ECDHE_RSA_RC4_128_SHA256	SSL_ECDHE_RSA_WITH_RC4_128_SHA	TLS_ECDHE_RSA_WITH_RC4_128_SHA	TLS 1.2	否

表 59: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (繼續)*

CipherSpec 第 359 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
TLS_RSA_WITH_3DES_EDE_CBC_SHA 第 360 頁的『2』	SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	否 第 360 頁的『4』
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	否 第 360 頁的『4』

表 59: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (繼續)*

CipherSpec 第 359 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	否 第 360 頁的『4』
TLS_RSA_WITH_AES_128_GCM_SHA256	SSL_RSA_WITH_AES_128_GCM_SHA256	TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	否 第 360 頁的『4』

表 59: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (繼續)*

CipherSpec 第 359 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	否 第 360 頁的『4』
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	否 第 360 頁的『4』

表 59: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (繼續)*

CipherSpec 第 359 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
TLS_RSA_WITH_AES_256_GCM_SHA384	SSL_RSA_WITH_AES_256_GCM_SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	否 第 360 頁的『4』
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	TLS 1.0	否

表 59: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (繼續)*

CipherSpec 第 359 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	TLS_RSA_WITH_NULL_SHA256	TLS 1.2	否
TLS_RSA_WITH_RC4_128_SHA256	SSL_RSA_WITH_RC4_128_SHA	SSL_RSA_WITH_RC4_128_SHA	TLS 1.2	否
ANY_TLS12	*TLS12	*TLS12	TLS 1.2	yes
TLS_AES_128_GCM_SHA256 第 360 頁的『3』	TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS 1.3 版	否

表 59: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (*繼續*)*

CipherSpec 第 359 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
TLS_AES_256_GCM_SHA384 第 360 頁的『3』	TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS 1.3 版	否
TLS_CHACHA20_POLY1305_SHA256 第 360 頁的『3』	TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS 1.3 版	否
TLS_AES_128_CCM_SHA256 第 360 頁的『3』	TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS 1.3 版	否

表 59: *IBM MQ* 及其對等的 *CipherSuites* 支援 *CipherSpecs (繼續)*

CipherSpec 第 359 頁的『1』	**等效 CipherSuite (IBM JRE)**	**等效 CipherSuite (Oracle JRE)**	**通訊協定**	**FIPS 140-2 相容**
TLS_AES_128_CCM_8_SHA256 第 360 頁的『3』	TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS 1.3 版	否
任意第 360 頁的『3』	*ANY	*ANY	多重	否
ANY_TLS13 第 360 頁的『3』	*TLS13	*TLS13	TLS V13	否
ANY_TLS12_OR_HIGHER 第 360 頁的『3』	*TLS12ORHIGHER	*TLS12ORHIGHER	TLS 1.2 及以上版本	否
ANY_TLS13_OR_HIGHER 第 360 頁的『3』	*TLS13ORHIGHER	*TLS13ORHIGHER	TLS 1.3 及以上版本	否

附註:

1. 這是在 IBM MQ 中的通道上配置的值，包括在 CCDT 中 (二進位或 JSON)。

2. Deprecated CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA 已淘汰。 不過，在連線因錯誤 AMQ9288 而終止之前，它仍可用來傳送最多 32 GB 的資料。 若要避免此錯誤，您需要避免使用三重 DES 演算法，或在使用此 CipherSpec 時啟用秘密金鑰重設。
3. 若要能夠使用 TLS v1.3 密碼，執行應用程式的 Java runtime environment (JRE) 必須支援 TLS v1.3。
4. V 9.3.0.17 V 9.3.5.1 從 IBM MQ 9.3.5 CSU 1 和 IBM MQ 9.3.0 CSU 17 開始，當以 FIPS 模式運作時， IBM Java 8 JRE 不再支援 RSA 金鑰交換。

在 IBM MQ classes for Java 應用程式中配置密碼組合及 FIPS 相符性

- 使用 IBM MQ classes for Java 的應用程式可以使用下列兩種方法之一來設定連線的 CipherSuite :
  - 將 MQEnvironment 類別中的 sslCipher 套組欄位設為 CipherSuite 名稱。
  - 在傳遞至 MQQueueManager 建構子的內容雜湊表中，將 CMQC.SSL_CIPHER_SUITE_PROPERTY 內容設為 CipherSuite 名稱。
- 使用 IBM MQ classes for Java 的應用程式可以使用下列兩種方法之一來施行 FIPS 140-2 相符性:
  - 在 MQEnvironment 類別中，將 sslFips「必要」欄位設為 true。
  - 將內容 CMQC.SSL_FIPS_REQUIRED_PROPERTYin 傳遞至 MQQueueManager 建構子的內容雜湊表設為 true。

將應用程式配置成使用 IBM Java 或 Oracle Java CipherSuite 對映

註:

V 9.3.3 對於 IBM MQ 9.3.3 中的 Continuous Delivery ， 會從產品中移除 Java 系統內容 `com.ibm.mq.cfg.useIBMCipherMappings`(控制使用哪些對映)。 從 IBM MQ 9.3.3 開始， 「密碼」可以定義為 CipherSpec 或 CipherSuite 名稱，並由 IBM MQ 正確處理。 如果您的 IBM MQ classes for Java 應用程式建立與佇列管理程式的安全 TLS 連線，則需要下列三個 Jackson JAR 檔:

- `jackson-annotations.jar`
- `jackson-core.jar`
- `jackson-databind.jar`

重要: `com.ibm.mq.cfg.useIBMCipherMappings` 的下列相關資訊僅適用於 Long Term Support 及 Continuous Delivery 之前 IBM MQ 9.3.3 。

您可以配置應用程式是使用預設 IBM Java CipherSuite 至 IBM MQ CipherSpec 對映，還是使用 Oracle CipherSuite 至 IBM MQ CipherSpec 對映。 因此，不論您的應用程式使用 IBM JRE 或 Oracle JRE ， 都可以使用 TLS CipherSuites 。 Java 系統內容 `com.ibm.mq.cfg.useIBMCipherMappings` 控制使用哪些對映。 內容可以是下列其中一個值:

true
使用 IBM Java CipherSuite 至 IBM MQ CipherSpec 對映。

此值為預設值。

false
使用 Oracle CipherSuite 至 IBM MQ CipherSpec 對映。

如需使用 IBM MQ Java 和 TLS 密碼的相關資訊，請參閱 MQdev 部落格文章 MQ Java、TLS 密碼、非 IBM JRE 及 APAR IT06775、 IV66840、 IT09423、 IT10837。

交互作業能力限制

視使用中的通訊協定而定，某些 CipherSuites 可能與多個 IBM MQ CipherSpec 相容。 不過， 僅支援使用表 1 中所指定 TLS 版本的 CipherSuite/CipherSpec 組合。 嘗試使用不受支援的 CipherSuites 及 CipherSpecs 組合將因適當的異常狀況而失敗。 使用任何這些 CipherSuite/CipherSpec 組合的安裝應該移至受支援的組合。

下表顯示套用此限制的 CipherSuites 。

表 60: CipherSuites 及其受支援及不受支援的 CipherSpecs		
CipherSuite	**支援的 TLS CipherSpec**	**不受支援的 SSL CipherSpec**
SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA 第 361 頁的『1』	TRIPLE_DES_SHA_US
SSL_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA	DES_SHA_EXPORT
SSL_RSA_WITH_RC4_128_SHA	TLS_RSA_WITH_RC4_128_SHA256	RC4_SHA_US

註:

1. Deprecated 此 CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA 已淘汰。 不過，在連線因錯誤 AMQ9288 而終止之前，它仍可用來傳送最多 32 GB 的資料。 若要避免此錯誤，您需要避免使用三重 DES 演算法，或在使用此 CipherSpec 時啟用秘密金鑰重設。

執行 *IBM MQ classes for Java* 應用程式

如果您使用用戶端或連結模式來撰寫應用程式 (包含 main () 方法的類別) ，請使用 Java 直譯器來執行您的程式。

執行下列指令:

```
java -Djava.library.path= library_path MyClass
```

其中 *library_path* 是 IBM MQ classes for Java 程式庫的路徑。 如需相關資訊，請參閱 第 300 頁的『IBM MQ classes for Java 位於檔案庫中』。

相關工作

追蹤 IBM MQ classes for Java 應用程式

追蹤 IBM MQ 資源配接器

IBM MQ classes for Java 環境相依行為

IBM MQ classes for Java 可讓您建立可針對不同版本的 IBM MQ 執行的應用程式。 此主題集合說明相依於這些不同版本之 Java 類別的行為。

IBM MQ classes for Java 提供類別的核心，可在所有環境中提供一致的功能及行為。 此核心以外的特性取決於應用程式所連接之佇列管理程式的功能。

除非這裡另有說明，否則所展現的行為會如適用於佇列管理程式的 MQI 應用程式參照 中所說明。

IBM MQ classes for Java 中的核心類別

IBM MQ classes for Java 包含一組核心類別，可在所有環境中使用。

下列類別集被視為核心類別，且只能在具有 第 362 頁的『IBM MQ classes for Java 核心類別的限制和變異』中所列出之次要變異的所有環境中使用。

- MQEnvironment
- MQException
- MQGetMessage 選項

  排除:

  - MatchOptions
  - GroupStatus
  - SegmentStatus
  - 分區段
- MQManagedObject

排除:

- inquire ()
- set ()

- MQMessage

  排除:

  - groupId
  - messageFlags
  - messageSequence 號碼
  - 偏移
  - originalLength
- MQPoolServices
- MQPoolServices 事件
- MQPoolServicesEventListener
- MQPoolToken
- MQPutMessage 選項

  排除:

  - KnownDestCount
  - UnknownDestCount
  - InvalidDestCount
  - recordFields
- MQProcess
- MQQUEUE
- MQQueueManager

  排除:

  - begin ()
  - accessDistribution 清單 ()
- MQSimpleConnection 管理程式
- MQTopic
- MQC

註:

1. 部分常數未包含在核心中 (如需詳細資料，請參閱 第 362 頁的『IBM MQ classes for Java 核心類別的限制和變異』); 請勿在完全可攜式程式中使用它們。
2. 部分平台不支援所有連線模式。 在這些平台上，您只能使用與受支援模式相關的核心類別和選項。

IBM MQ classes for Java 核心類別的限制和變異

核心類別通常會在所有環境之間一致地運作，即使對等的 MQI 呼叫通常會有環境差異。 此行為如同使用 AIX、 Linux 或 Windows 佇列管理程式，但下列次要限制及變異除外。

IBM MQ classes for Java 中 *MQGMO_ * 值的限制*

並非所有佇列管理程式都支援某些 MQGMO_ * 值。

使用下列 MQGMO_ * 值可能會導致從 MQQueue.get() 擲出 MQException:

MQGMO_SYNCPOINT_IF_PERSISTENT
MQGMO_MARK_SKIP_BACKOUT
MQGMO_BROWSE_MSG_UNDER_CURSOR

MQGMO_LOCK
MQGMO_UNLOCK
MQ GMO_LOGICAL_ORDER
MQGMO_COMPLETE_MESSAGE
MQGMO_ALL_MSGS_AVAILABLE
MQGMO_ALL_SEGMENTS_AVAILABLE
MQGMO_UNMARKED_BROWSE_MSG
MQ GMO_MARK_BROWSE_HANDLE
MQGMO_MARK_BROWSE_CO_OP
MQ GMO_UNMARK_BROWSE_HANDLE
MQGMO_UNMARK_BROWSE_CO_OP

此外，從 Java 使用時，不支援 MQGMO_SET_SIGNAL。

IBM MQ classes for Java 中 *MQPMRF_ * 值的限制
只有在將訊息放入配送清單時，才會使用這些訊息，且只有支援配送清單的佇列管理程式才支援這些訊息。 例如， z/OS 佇列管理程式不支援配送清單。

IBM MQ classes for Java 中 *MQPMO_ * 值的限制
並非所有佇列管理程式都支援某些 MQPMO_ * 值

使用下列 MQPMO_ * 值可能會導致從 MQQueue.put() 或 MQQueueManager.pput () 擲出 MQException:

MQPMO_LOGICAL_ORDER
MQPMO_NEW_CORREL_ID
MQPMO_NEW_MESSAGE_ID
MQPMO_RESOLVE_LOCAL_Q

IBM MQ classes for Java 中 *MQCNO_ * 值的限制及變異
不支援某些 MQCNO_* 值。

- IBM MQ classes for Java 不支援自動用戶端重新連接。 無論您設定任何值 MQCNO_RECONNECT_* ，連線會繼續像您設定 MQCNO_RECONNECT_DISABLED 一樣運作。
- 在不支援 MQCNO_FASTPATH 的佇列管理程式上，會忽略 MQCNO_FASTPATH 。 用戶端連線也會忽略它。

IBM MQ classes for Java 中 *MQRO_ * 值的限制
可以設定下列報告選項。

MQRO_EXCEPTION_WITH_FULL_DATA
MQRO_EXPIRATION_WITH_FULL_DATA
MQ Ro_COA_WITH_FULL_DATA
MQRO_COD_WITH_FULL_DATA
MQRO_DISCARD_MSG
MQRO_PASS_DISCARD_AND_EXPIRY

如需相關資訊，請參閱 報告。

z/OS 上 *IBM MQ classes for Java* 與其他平台之間的細項差異
在某些區域中， IBM MQ for z/OS 與其他平台上的 IBM MQ 行為不同。

BackoutCount
即使訊息已取消超過 255 次， z/OS 佇列管理程式仍會傳回最多 255 個 BackoutCount 。

預設動態佇列字首
使用連結連線連接至 z/OS 佇列管理程式時，預設動態佇列字首為 CSQ. *。 否則，預設動態佇列字首為 AMQ. *。

MQQueueManager 建構子
z/OS 不支援用戶端連接。 嘗試使用用戶端選項進行連接會導致 MQException 出現 MQCC_FAILED 和 MQRC_ENVIRONMENT_ERROR。

MQQueueManager 建構子也可能會失敗，並產生 MQRC_CHAR_CONVERSION_ERROR (如果無法起始設定 IBM-1047 與 ISO8859-1 字碼頁之間的轉換) ，或 MQRC_UCS2_CONVERSION_ERROR (如果無法起始設定佇列管理程式字碼頁與 Unicode 之間的轉換)。 如果您的應用程式失敗，並出現下列其中一個原因碼，請確定已安裝 Language Environment 的「國家語言資源」元件，並確定有正確的轉換表可用。

Unicode 的轉換表會安裝為 z/OS C/C++ 選用特性的一部分。 如需啟用 UCS-2 轉換的相關資訊，請參閱 *z/OS C/C++* 程式設計手冊(SC09-4765)。

IBM MQ classes for Java 核心類別以外的特性

IBM MQ classes for Java 包含某些函數，這些函數專門設計用來使用並非所有佇列管理程式都支援的 API 延伸。 這個主題集合說明它們在使用不支援它們的佇列管理程式時的行為。

MQQueueManager 建構子選項中的變異

部分 MQQueueManager 建構子包含選用的整數引數。 並非所有平台都接受此引數的部分值。

如果 MQQueueManager 建構子包含選用的整數引數，則會對映至 MQI 的 MQCNO 選項欄位，並用來在正常與捷徑連線之間切換。 如果唯一使用的選項是 MQCNO_STANDARD_BINDING 或 MQCNO_FASTPATH_BINDING ，則在所有環境中都接受這種延伸形式的建構子。 任何其他選項都會導致建構子失敗，並產生 MQRC_OPTIONS_ERROR。 捷徑選項 CMQC.MQCNO_FASTPATH_BINDING 的佇列管理程式的連結連線才允許使用它。 在其他環境中，會忽略它。

MQQueueManager.begin () 方法的限制

此方法只能在連結模式下針對 AIX, Linux, and Windows 系統上的 IBM MQ 佇列管理程式使用。 否則，它會因 MQRC_ENVIRONMENT_ERROR 而失敗。

如需詳細資料，請參閱 第 336 頁的『JTA/JDBC 協調，使用 IBM MQ classes for Java』 。

MQGetMessage 選項欄位中的變異

部分佇列管理程式不支援第 2 版 MQGMO 結構，因此您必須將部分欄位設為其預設值。

使用不支援第 2 版 MQGMO 結構的佇列管理程式時，請保留下列欄位設為其預設值:

GroupStatus
SegmentStatus
分區段

此外， MatchOptions 欄位僅支援 MQMO_MATCH_MSG_ID 和 MQMO_MATCH_CORREL_ID。 如果您在這些欄位中放置不受支援的值，則後續的 MQDestination.get() 會失敗，並產生 MQRC_GMO_ERROR。 如果佇列管理程式不支援第 2 版 MQGMO 結構，則在順利完成 MQDestination.get() 之後，不會更新這些欄位。

IBM MQ classes for Java 中配送清單的限制

並非所有佇列管理程式都容許您開啟 MQDistributionList。

下列類別用來建立配送清單:

MQDistributionList
MQDistributionList 項目
MQMessageTracker

您可以在任何環境中建立並移入 MQDistributionLists 和 MQDistributionList 項目，但並非所有佇列管理程式都容許您開啟 MQDistributionList。 特別是 z/OS 佇列管理程式不支援配送清單。 當使用這類佇列管理程式時，嘗試開啟 MQDistributionList 會導致 MQRC_OD_ERROR。

MQPutMessage 選項欄位中的變異

如果佇列管理程式不支援配送清單，則會以不同方式處理某些 MQPMO 欄位。

MQPMO 中的四個欄位會呈現為 MQPutMessage 選項類別中的下列成員變數:

KnownDestCount
UnknownDestCount
InvalidDestCount

recordFields

這些欄位主要用於與配送清單搭配使用。 不過，在 MQPUT 至單一佇列之後，支援配送清單的佇列管理程式也會填入 DestCount 欄位。 例如，如果佇列解析為本端佇列，則 knownDest 計數會設為 1 ，而其他兩個計數欄位則會設為 0。

如果佇列管理程式不支援配送清單，則會模擬下列值:

- 如果 put () 成功，則 unknownDest 計數會設為 1 ，而其他計數會設為 0。
- 如果 put () 失敗，則 invalidDest 計數會設為 1 ，而其他計數會設為 0。

recordFields 變數與配送清單搭配使用。 不論環境為何，都可以隨時將值寫入 recordFields 。 如果在後續的 MQDestination.put() 或 MQQueueManager.pput () 而非 MQDistributionList.pput () 上使用 MQPutMessageOptions 物件，則會忽略此參數。

使用 *IBM MQ classes for Java* 的 *MQMD* 欄位中的限制
當使用不支援分段的佇列管理程式時，某些與訊息分段相關的 MQMD 欄位應該保留其預設值。

下列 MQMD 欄位主要關注訊息分段:

GroupId
MsgSeqNumber
偏移
MsgFlags
OriginalLength

如果應用程式將任何這些 MQMD 欄位設為預設值以外的值，然後在不支援這些值的佇列管理程式上執行 put () 或 get () ，則 put () 或 get () 會引發 MQRC_MD_ERROR 的 MQException。 具有這類佇列管理程式的成功 put () 或 get () 一律會將 MQMD 欄位設為其預設值。 請勿將已分組或已分段的訊息傳送至針對不支援訊息分組及分段的佇列管理程式執行的 Java 應用程式。

如果 Java 應用程式嘗試從不支援這些欄位的佇列管理程式取得 () 訊息，且要擷取的實體訊息是一組分段訊息的一部分 (亦即，它具有 MQMD 欄位的非預設值) ，則會擷取而不會發生錯誤。 不過， MQMessage 中的 MQMD 欄位不會更新， MQMessage 格式內容會設為 MQFMT_MD_EXTENSION ，且 true 訊息資料會以包含新欄位值的 MQMDE 結構作為字首。

CICS Transaction Server 下 *IBM MQ classes for Java* 的限制

在 CICS Transaction Server for z/OS 環境中，只容許主要 (第一個) 執行緒發出 CICS 或 IBM MQ 呼叫。

請注意，不支援在 CICS Java 應用程式中使用 IBM MQ JMS 類別。

因此，無法在此環境中的執行緒之間共用 MQQueueManager 或 MQQueue 物件，也無法在子執行緒上建立新的 MQQueueManager 。

z/OS 第 363 頁的『z/OS 上 IBM MQ classes for Java 與其他平台之間的細項差異』 識別針對 z/OS 佇列管理程式執行時，適用於 IBM MQ classes for Java 的部分限制及變異。 此外，在 CICS 下執行時，不支援 MQQueueManager 上的交易控制方法。 應用程式使用 JCICS 作業同步化方法 Task.commit() 和 Task.rollback() ，而不是發出 MQQueueManager.commit () 或 MQQueueManager.backout ()。 作業類別由 com.ibm.cics.server 套件中的 JCICS 提供。

使用 IBM MQ 資源配接器

資源配接器容許在應用程式伺服器中執行的應用程式存取 IBM MQ 資源。 它支援入埠和出埠通訊。

資源配接器包含的內容

V 9.3.0 V 9.3.0 從 IBM MQ 9.3.0 開始，支援 Jakarta Messaging 3.0 開發新的應用程式。 IBM MQ 9.3.0 繼續支援 JMS 2.0 現有的應用程式。 除了支援 Java EE 和 JMS 2.0 的資源配接器之外， IBM MQ 9.3.0 還提供支援 Jakarta Messaging 的資源配接器。

JM 3.0 **Jakarta Messaging 的 IBM MQ 資源配接器**

Jakarta Connectors Architecture 提供將在 Jakarta EE 環境中執行的應用程式連接至「企業資訊系統 (EIS)」(例如 IBM MQ 或 Db2) 的標準方式。 Jakarta Messaging 的 IBM MQ 資源配接器會實作 Jakarta Connectors 2.0.0 介面，並包含 IBM MQ classes for Jakarta Messaging。 它可讓在應用程式伺服器中執行的 Jakarta Messaging 應用程式和訊息驅動 Bean (MDB) 存取 IBM MQ 佇列管理程式的資源。 資源配接器同時支援點對點網域和發佈/訂閱網域。

JMS 2.0 **JMS 2.0 的 IBM MQ 資源配接器**

Java Platform, Enterprise Edition Connector Architecture (JCA) 提供將在 Java EE 環境中執行的應用程式連接至「企業資訊系統 (EIS)」(例如 IBM MQ 或 Db2) 的標準方式。 JMS 2.0 的 IBM MQ 資源配接器會實作 JCA 1.7 介面，並包含 IBM MQ classes for JMS。 它容許在應用程式伺服器中執行的 JMS 應用程式和訊息驅動 Bean (MDB) 存取 IBM MQ 佇列管理程式的資源。 資源配接器同時支援點對點網域和發佈/訂閱網域。

IBM MQ 資源配接器支援應用程式與佇列管理程式之間兩種類型的通訊:

出埠通訊

應用程式會啟動與佇列管理程式的連線，然後以同步方式將 JMS 訊息傳送至 JMS 目的地，並從 JMS 目的地接收 JMS 訊息。

入埠通訊

到達 JMS 目的地的 JMS 訊息會遞送至 MDB ， 以非同步方式處理訊息。

資源配接器也包含 IBM MQ classes for Java。 這些類別會自動可供在資源配接器已部署至其中的應用程式伺服器中執行的應用程式使用，並容許在該應用程式伺服器中執行的應用程式在存取 IBM MQ 佇列管理程式的資源時使用 IBM MQ classes for Java API。

支援在 Java EE 環境內使用 IBM MQ classes for Java ， 但有一些限制。 如需這些限制的相關資訊，請參閱 第 292 頁的『在 Java EE 內執行 IBM MQ classes for Java 應用程式』。

要使用的資源配接器版本

V 9.3.0 V 9.3.0 您使用的資源配接器版本取決於您要將它部署到支援 Jakarta EE 或 Java EE 的應用程式伺服器:

JM 3.0 V 9.3.0 V 9.3.0 **Jakarta EE**

從 IBM MQ 9.3.0 開始，支援 Jakarta Messaging 3.0 。 Jakarta Messaging 的 IBM MQ 資源配接器必須部署在支援 Jakarta EE 的應用程式伺服器內。

Java EE 7

IBM MQ 8.0 以及更新版本的資源配接器支援 JCA v1.7 ， 並提供 JMS 2.0 支援。 這個資源配接器需要部署在 Java EE 7 以及更新版本的應用程式伺服器內 (請參閱 第 367 頁的『支援的 IBM MQ 資源配接器陳述式』)。

您可以在任何經認證符合 Java Platform, Enterprise Edition 7 規格的應用程式伺服器上安裝 IBM MQ 8.0 或更新版本資源配接器。 使用 IBM MQ 8.0 或更新版本的資源配接器，應用程式可以使用 BINDINGS 或 CLIENT 傳輸來連接至佇列管理程式。

重要: IBM MQ 8.0 或更新版本的資源配接器只能部署至支援 JMS 2.0 的應用程式伺服器。

搭配使用資源配接器與 WebSphere Application Server traditional

從 IBM MQ 9.0 開始， IBM MQ 資源配接器已預先安裝在 WebSphere Application Server traditional 9.0 或更新版本內。 因此，不需要安裝新的資源配接器。

JM 3.0 WebSphere Application Server traditional 目前不支援 Jakarta EE。 請參閱 IBM MQ 資源配接器支援陳述式。

註: IBM MQ 9.0 或更新版本的資源配接器可以 CLIENT 或 BINDINGS 傳輸模式連接至任何使用中的 IBM MQ 佇列管理程式。

搭配使用資源配接器與 WebSphere Liberty

如果要從 WebSphere Liberty 連接至 IBM MQ，您必須使用 IBM MQ 資源配接器。 由於 Liberty 不包含 IBM MQ 資源配接器，您必須從 Fix Central 個別取得它。

V 9.3.0 V 9.3.0 您使用的資源配接器版本取決於您將它部署到支援 Jakarta EE 或 Java EE 的 Liberty 版本。

如需如何下載及安裝資源配接器的相關資訊，請參閱 第 373 頁的『在 Liberty 中安裝資源配接器』。

相關概念

第 379 頁的『配置入埠通訊的資源配接器』
如果要配置入埠通訊，請定義一或多個 ActivationSpec 物件的內容。

第 392 頁的『配置資源配接器以進行出埠通訊』
如果要配置出埠通訊，請定義 ConnectionFactory 物件和受管理目的地物件的內容。

第 68 頁的『使用 IBM MQ classes for JMS/Jakarta Messaging』
IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 是 IBM MQ 隨附的 Java 傳訊提供者。 除了實作 JMS 和 Jakarta Messaging 規格中定義的介面，這些傳訊提供者也會將兩組延伸新增至 Java 傳訊 API。

第 290 頁的『使用 IBM MQ classes for Java』
在 Java 環境中使用 IBM MQ。 IBM MQ classes for Java 可讓 Java 應用程式以 IBM MQ 用戶端身分連接至 IBM MQ，或直接連接至 IBM MQ 佇列管理程式。

相關參考

將應用程式伺服器配置成使用最新的資源配接器維護層次
IBM MQ 資源配接器的問題判斷

WebSphere Application Server 主題

維護 IBM MQ 資源配接器
將 JMS 應用程式部署至 Liberty 以使用 IBM MQ 傳訊提供者

支援的 IBM MQ 資源配接器陳述式

您必須用於應用程式與佇列管理程式之間通訊的 IBM MQ 資源配接器，取決於您是使用 Jakarta Messaging 3.0 API 還是 JMS 2.0 API。

JMS 2.0 IBM MQ 8.0 或更新版本隨附實作 JMS 2.0 規格的資源配接器。 它只能部署至符合 Java Platform, Enterprise Edition 7 (Java EE 7) 標準的應用程式伺服器，因此支援 JMS 2.0。 Java Platform, Enterprise Edition 的認證應用程式伺服器清單在 Oracle 的網站上進行維護。

JM 3.0 V 9.3.0 V 9.3.0 從 IBM MQ 9.3.0 開始，支援 Jakarta Messaging 3.0 開發新的應用程式。 IBM MQ 9.3.0 繼續支援 JMS 2.0 現有的應用程式。 除了支援 Java EE 和 JMS 2.0 的資源配接器之外， IBM MQ 9.3.0 還提供支援 Jakarta Messaging 的資源配接器。 不支援在同一應用程式中同時使用 Jakarta Messaging 3.0 API 和 JMS 2.0 API。 如需相關資訊，請參閱使用 IBM MQ classes for JMS。

在 WebSphere Liberty 內部署

WebSphere Liberty 8.5.5 Fix Pack 6 及更新版本，以及 WebSphere Application Server Liberty 9.0 及更新版本都是 Java EE 7 認證的應用程式伺服器，因此 IBM MQ 9.0 資源配接器可以部署至其中。

V 9.3.0 V 9.3.0 如果要將 Jakarta Messaging 的 IBM MQ 資源配接器與 Liberty 搭配使用，您必須使用支援 Jakarta EE 的 Liberty 版本。

WebSphere Liberty 具有下列特性可用來使用資源配接器:

- JM 3.0 V 9.3.0 V 9.3.0 messaging-3.0 特性可讓您使用 Jakarta Messaging 3.0 資源配接器。
- wmqJmsClient-2.0 特性可容許使用 JMS 2.0 資源配接器。
- wmqJmsClient-1.1 特性可容許使用 JMS 1.1 資源配接器。

重要:

- JM 3.0 V 9.3.0 V 9.3.0 Jakarta Messaging 的 IBM MQ 資源配接器必須部署至支援 Jakarta EE 的 Liberty 版本。 這個資源配接器無法與支援舊版 Java EE 規格而非 Jakarta EE 的 Liberty 版本一起使用。
- JMS 2.0 必須使用 wmqJmsClient-2.0 特性來部署支援 JMS 2.0 的 IBM MQ 8.0 或更新版本資源配接器。

在 WebSphere Application Server traditional 內部署

WebSphere Application Server traditional 9.0 隨附已安裝的 IBM MQ 9.0 資源配接器。 因此，不需要安裝新的資源配接器。 已安裝的資源配接器可以 CLIENT 或 BINDINGS 傳輸模式連接至在受支援 IBM MQ 版本上執行的任何佇列管理程式。 如需相關資訊，請參閱第 368 頁的『與 IBM MQ 8.0 或更新版本佇列管理程式的連線功能』。

重要:

- IBM MQ 9.0 資源配接器無法部署至 WebSphere Application Server traditional IBM MQ 9.0 之前的版本，因為這些版本未經 Java EE 7 認證。
- JM 3.0 WebSphere Application Server traditional 目前不支援 Jakarta EE。

如需 WebSphere Application Server 隨附的資源配接器版本的相關資訊，請參閱 Technote WebSphere Application Server 隨附了哪一個 WebSphere MQ 資源配接器 (RA) 版本?

將資源配接器與其他應用程式伺服器搭配使用

對於所有其他符合 Java EE 7 或 Jakarta EE 標準的應用程式伺服器，在順利完成 IBM MQ 資源配接器 安裝驗證測試 (IVT) 之後發生的問題，可以向 IBM 報告，以調查 IBM MQ 產品追蹤及其他 IBM MQ 診斷資訊。 如果 IBM MQ 資源配接器 IVT 無法順利執行，任何發現的問題都可能是部署不正確或應用程式伺服器特定的資源定義不正確所造成，必須使用應用程式伺服器說明文件及該應用程式伺服器的支援組織來調查這些問題。

Java 執行時期

用來執行應用程式伺服器的 Java 執行時期 (JRE) 必須是 IBM MQ 9.0 或更新版本用戶端所支援的執行時期 (JRE)。 如需相關資訊，請參閱 IBM MQ 的系統需求。 (選取您要查看的版本及作業系統或元件報告，然後遵循 **支援的軟體** 標籤下列出的 **Java** 鏈結。)

與 IBM MQ 8.0 或更新版本佇列管理程式的連線功能

當使用已部署至 Java EE 7 認證應用程式伺服器的資源配接器來連接 IBM MQ 8.0 或更新版本佇列管理程式時，可以使用完整範圍的 JMS 2.0 功能。 如果要利用 JMS 2.0 功能，資源配接器需要利用 IBM MQ 傳訊提供者標準模式來連接佇列管理程式。 如需相關資訊，請參閱 配置 JMS **PROVIDERVERSION** 內容。

JM 3.0 V 9.3.0 V 9.3.0 當使用已部署至 Jakarta EE 認證應用程式伺服器的資源配接器來連接 IBM MQ 9.3 佇列管理程式時，可以使用完整範圍的 Jakarta Messaging 3.0 功能。

MQ 延伸

JMS 2.0 規格引進某些行為運作方式的變更。 因為 IBM MQ 8.0 或更新版本實作此規格，所以 IBM MQ 8.0 與更新版本及舊版產品之間的行為會有所變更。 在 IBM MQ 8.0 或更新版本中， IBM MQ classes for JMS 包括對 Java 系統內容 `com.ibm.mq.jms.SupportMQExtensions` 的支援，當設為 TRUE 時，會導致這些 IBM MQ 版本將這些行為回復為 IBM WebSphere MQ 7.5 或更早版本的行為。 此內容的預設值為 FALSE。

IBM MQ 9.0 或更新版本的資源配接器也包含一個稱為 `supportMQExtensions` 的資源配接器內容，其效果和預設值與 `com.ibm.mq.jms.SupportMQExtensions` Java 系統內容相同。 依預設，在 `ra.xml` 中，這個資源配接器內容會設為 false。

如果同時設定資源配接器內容和 Java 系統內容，則系統內容具有優先順序。

請注意，在已部署在 WebSphere Application Server traditional 9.0 內的資源配接器內，這個內容會自動設為 TRUE ， 以協助移轉。

如需相關資訊，請參閱 第 274 頁的『SupportMQExtensions 內容』。

一般問題

不支援階段作業交錯

部分應用程式伺服器提供稱為階段作業交錯的功能，其中相同的 JMS 階段作業可以在多個交易中使用，雖然一次只會列入一個階段作業。 IBM MQ 資源配接器不支援這項功能，這可能導致下列問題:

嘗試將訊息放入 MQ 佇列失敗，原因碼為 2072 (MQRC_SYNCPOINT_NOT_AVAILABLE)。

呼叫 xa_close () 失敗，原因碼為 -3 (XAER_PROTO) ，並在從應用程式伺服器存取的 IBM MQ 佇列管理程式上產生探測 ID 為 AT040010 的 FDC。 如需如何停用此功能的相關資訊，請參閱應用程式伺服器說明文件。

關於 XA 交易回復如何回復 XA 資源的 Java Transaction API (JTA) 規格

JTA 規格的 3.4.8 小節未定義用來重建 XA 資源以執行 XA 交易式回復的特定機制。 因此，要如何回復 XA 交易所涉及的 XA 資源，由每一個個別交易管理程式 (以及應用程式伺服器) 來決定。 對於某些應用程式伺服器， IBM MQ 9.0 資源配接器可能不會實作用來執行 XA 交易式回復的應用程式伺服器特定機制。

ManagedConnectionFactory 中的相符連線

應用程式伺服器可以在 IBM MQ 資源配接器提供的 ManagedConnectionFactory 實例上呼叫 matchManaged 連線方法。 只有在方法找到同時符合應用程式伺服器傳遞給方法之 **javax.security.auth.Subject** 和 **javax.resource.spi.ConnectionRequestInfo** 引數的方法時，才會傳回 ManagedConnection 。

IBM MQ 資源配接器的限制

所有 IBM MQ 平台都支援 IBM MQ 資源配接器。 不過，當您使用 IBM MQ 資源配接器時， IBM MQ 的某些特性無法使用或受到限制。

IBM MQ 資源配接器有下列限制:

- 從 IBM MQ 8.0 開始，資源配接器是提供 JMS 2.0 功能的 Java Platform, Enterprise Edition 7 (Java EE 7) 資源配接器。 因此， IBM MQ 8.0 或更新版本的資源配接器必須安裝在 Java EE 7 或更新版本的認證應用程式伺服器中。 它可以在用戶端或連結傳輸模式下連接至任何使用中的佇列管理程式。
- 在 WebSphere Liberty 應用程式伺服器內執行時，不支援已穩定的 IBM MQ classes for Java 。 在其他應用程式伺服器內，不建議使用 IBM MQ classes for Java 。 如需 Java EE 內 IBM MQ classes for Java 考量的詳細資料，請參閱 IBM Technote Using WebSphere MQ Java Interfaces in J2EE/JEE Environments 。
- 在 z/OS 上的 WebSphere Liberty 應用程式伺服器內執行時，必須使用 wmqJmsClient-2.0 特性。 z/OS 不可能提供一般 JCA 支援。
- IBM MQ 資源配接器不支援以 Java 以外的語言撰寫的通道結束程式。
- 當應用程式伺服器在執行中，所有 JCA 資源的 sslFipsRequired 內容值必須為 true ，所有 JCA 資源的值必須為 false。 即使未同時使用 JCA 資源，這是一項需求。 如果 sslFipsRequired 內容針對不同的 JCA 資源具有不同的值， IBM MQ 會發出原因碼 MQRC_UNSUPPORTED_CIPHER_SUITE ，即使未使用 TLS 連線也一樣。
- 您無法為應用程式伺服器指定多個金鑰儲存庫。 如果與多個佇列管理程式建立連線，則所有連線都必須使用相同的金鑰儲存庫。 此限制不適用於 WebSphere Application Server。
- 如果您使用的用戶端通道定義表 (CCDT) 有多個適合的用戶端連線通道定義，一旦失敗，資源配接器可能會選取不同的通道定義，因此會從 CCDT 選取不同的佇列管理程式，這會造成交易回復問題。 資源配接器不會採取任何動作來防止使用這類配置，您必須負責避免可能導致交易回復問題的配置。
- 在 Java EE 儲存器 (EJB/Servlet) 中執行時，出埠連線不支援連線重試功能。 當在 JEE 儲存器環境定義中使用配接器時，不論交易配置或非交易用途，完全不支援出埠 JMS 的連線重試。
- 不支援 JMS 連線的重新鑑別 (如 Java EE Connector Architecture 1.7 版規格的 9.1.9 小節中所定義)。 IBM MQ 資源新增程式內的 `ra.xml` 檔案必須將稱為 **reauthentication-support** 的內容設為值 `false`。 應用程式伺服器嘗試重新鑑別 JMS 連線，導致 IBM MQ 資源配接器以 MQJCA1028 訊息碼擲出 javax.resource.spi.SecurityException 。

相關工作

指定在執行時期於 MQI 用戶端上僅使用 FIPS 認證的 CipherSpecs

相關參考

AIX, Linux, and Windows 的聯邦資訊存取安全標準 (FIPS)

WebSphere Application Server 和 IBM MQ 資源配接器

IBM MQ 資源配接器供在 WebSphere Application Server 中使用 IBM MQ 傳訊提供者來執行 JMS 傳訊的應用程式使用。

重要: 請勿將 IBM MQ 資源配接器與 WebSphere Application Server 6.0 或 WebSphere Application Server 6.1 搭配使用。

WebSphere Application Server traditional 9.0 包含 IBM MQ 9.0 資源配接器的版本。 IBM MQ 9.0 或更新版本的資源配接器無法部署至舊版 WebSphere Application Server，因為這些版本未經 Java EE 7 認證。

JM 3.0 V 9.3.0 V 9.3.0 WebSphere Application Server traditional 目前不支援 Jakarta EE。 請參閱 IBM MQ 資源配接器支援陳述式。

如果您想要使用 JMS 應用程式，從 WebSphere Application Server 內存取 IBM MQ 佇列管理程式的資源，請使用 WebSphere Application Server 中的 IBM MQ 傳訊提供者。 IBM MQ 傳訊提供者包含 IBM MQ classes for JMS 的版本。 如需相關資訊，請參閱 Technote 哪個版本的 WebSphere MQ Resource Adapter (RA) 隨附於 WebSphere Application Server?。

重要: 請勿在應用程式內包含任何 IBM MQ classes for JMS 或 IBM MQ classes for Java JAR 檔。 這樣做可能會導致 ClassCast 異常狀況，且難以維護。

Liberty 和 IBM MQ 資源配接器

您可以利用 Liberty 特性，將 IBM MQ 資源配接器安裝在 WebSphere Liberty 中。 您使用的特性取決於您要安裝的資源配接器版本。 另外，您也可以遵循某些限制，利用一般 Java Platform, Enterprise Edition Connector Architecture (Java EE JCA) 支援來安裝資源配接器。

將資源配接器安裝到 Liberty 時的一般限制

當使用 wmqJmsClient-1.1 或 wmqJmsClient-2.0 特性以及使用一般 JCA 支援時，下列限制適用於資源配接器:

- Liberty 中不支援 IBM MQ classes for Java 。 它們不得與 IBM MQ Liberty 傳訊特性或一般 JCA 支援一起使用。 如需相關資訊，請參閱 在 J2EE/JEE 環境中使用 WebSphere MQ Java 介面。
- IBM MQ 資源配接器的傳輸類型為 BINDINGS_THEN_CLIENT。 IBM MQ Liberty 傳訊特性不支援這個傳輸類型。
- 在 IBM MQ 9.0 之前， Advanced Message Security (AMS) 特性未包含在 IBM MQ Liberty 傳訊特性中。 不過， IBM MQ 9.0 或更新版本的資源配接器支援 AMS 。

註: 在大於 IBM MQ 9.0.0.6 和 IBM MQ 9.1.0.1 的 IBM MQ 版本上，您應該使用 transportSecurity-1.0 特性，而不是 ssl-1.0 特性。

如需相關資訊，請參閱:

在 Liberty 中啟用 SSL 通訊
Liberty 中的 SSL 預設值
傳輸安全 1.0

使用 Liberty 特性時的限制

如果將 WebSphere Liberty 8.5.5 Fix Pack 2 併入 WebSphere Liberty 8.5.5 Fix Pack 5 ，則只能使用 wmqJmsClient-1.1 特性，且只能使用 JMS 1.1 。 WebSphere Liberty 8.5.5 Fix Pack 6 已新增 wmqJmsClient-2.0 特性，因此可以使用 JMS 2.0 。

JM 3.0 V 9.3.0 V 9.3.0 從 IBM MQ 9.3.0 開始，支援 Jakarta Messaging 3.0 。 如果要將 Jakarta Messaging 的 IBM MQ 資源配接器與 Liberty 搭配使用，您必須使用支援 Jakarta EE 的 Liberty 版本。 您必須搭配使用 Jakarta Messaging 的資源配接器與 Liberty generic messaging-3.0 特性。

您必須使用的特性取決於您使用的資源配接器版本:

- IBM MQ 8.0.0 Fix Pack 3 以及更新版本 IBM MQ 8.0 資源配接器只能與 wmqJmsClient-2.0 特性搭配使用。
- IBM MQ 9.0 資源配接器只能與 wmqJmsClient-2.0 特性搭配使用。
- JM 3.0 V 9.3.0 V 9.3.0 messaging-3.0 特性容許使用 Jakarta Messaging 3.0 資源配接器。

使用一般 JCA 支援時的限制

如果您使用一般 JCA 支援，則適用下列限制:

- 使用一般 JCA 支援時，您必須指定 JMS 的層次。 JMS 2.0 和 JCA 1.7 只能與 IBM MQ 8.0.0 Fix Pack 3 以及更新版本 IBM MQ 8.0 資源配接器搭配使用。
- 無法使用一般 JCA 支援在 z/OS 上執行 IBM MQ 資源配接器。 為了在 z/OS 上執行 IBM MQ 資源配接器，必須使用 wmqJmsClient-1.1 或 wmqJmsClient-2.0 特性來執行。
- 資源配接器的位置是利用下列 xml 元素來指定:

JM 3.0
```
<resourceAdapter id="mqJms" location="${server.config.dir}/
wmq.jakarta.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

JMS 2.0
```
<resourceAdapter id="mqJms" location="${server.config.dir}/wmq.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

重要: 對於 wmqJms， ID 標籤的值可以是任何 EXCEPT。 如果您使用 wmqJms 作為 ID ，則 Liberty 無法適當地載入資源配接器。 這是因為 wmqJms 是內部用來參照 IBM MQ 特定特性的 ID。 它實際上會建立 NullPointer 異常狀況。

下列範例顯示執行 JMS 2.0 時來自 `server.xml` 檔案的部分 Snippet:

```
<!-- Enable features -->
  <featureManager>
    <feature>servlet-3.1</feature>
    <feature>jndi-1.0</feature>
    <feature>jca-1.7</feature>
    <feature>jms-2.0</feature>
  </featureManager>
```

提示: 請注意使用 jca-1.7 和 jms-2.0 特性，以及缺少 wmqJmsClient-2.0 特性。

```
<resourceAdapter id="mqJms" location="${server.config.dir}/wmq.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

提示: 請注意使用 mqJms 作為 ID ， 這是偏好的作法。 請勿使用 wmqJms。

```
<application id="WMQHTTP" location="${server.config.dir}/apps/WMQHTTP.war"
name="WMQHTTP" type="war">
    <classloader apiTypeVisibility="spec, ibm-api, api, third-party"
classProviderRef="mqJms"/>
</application>
```

提示: 請注意 classloaderProvider 透過 ID mqJms 傳回資源配接器的參照; 這是允許載入 IBM MQ 特定的類別。

使用一般 JCA 支援進行追蹤時的限制

追蹤及記載未整合在 Liberty 追蹤系統內。 相反地，必須使用 Java 系統內容或 IBM MQ classes for JMS 配置檔來啟用 IBM MQ 資源配接器追蹤，如 追蹤 IBM MQ classes for JMS 應用程式中所述。 如需如何在 Liberty 中設定 Java 系統內容的詳細資料，請參閱 WebSphere Liberty 文件。

比方說，如果要在 Liberty 19.0.0.9 中啟用 IBM MQ 資源配接器的追蹤，請在 Liberty 檔 `jvm.options` 中新增一個項目:

1. 建立名為 `jvm.options` 的文字檔。
2. 插入下列 JVM 選項，以在此檔案中啟用追蹤 (每行一個):

   ```
   -Dcom.ibm.msg.client.commonservices.trace.status=ON
   -Dcom.ibm.msg.client.commonservices.trace.outputName=C:\Trace\MQRA-WLP_%PID%.trc
   ```

3. 若要將這些設定套用至單一伺服器，請將 `jvm.options` 儲存在:

   ```
   ${server.config.dir}/jvm.options
   ```

   若要將這些變更套用至所有 Liberty，請在下列位置儲存 `jvm.options`:

   ```
   ${wlp.install.dir}/etc/jvm.options
   ```

   這將對沒有本端定義的 `jvm.options` 檔案的所有 JVM 生效。
4. 重新啟動伺服器以啟用變更。

這會導致將追蹤寫入 `<path_to_trace_to>`目錄中稱為 `MQRA-WLP_<process identifier>.trc` 的追蹤檔。

使用用戶端通道定義表的完整 Liberty XA 支援

使用 WebSphere Liberty 18.0.0.2 及 IBM MQ 9.2.0 或更新版本時，您可以在與 XA 交易一起使用用戶端通道定義表 (CCDT) 內的佇列管理程式群組。 這表示現在可以使用佇列管理程式群組所提供的工作量配送及可用性，同時維護交易完整性。

如果佇列管理程式的連線功能發生錯誤，則佇列管理程式需要再次變成可用，才能解決交易。 交易回復由 Liberty 管理，且您可能需要配置交易管理程式，以便容許適當的時段讓佇列管理程式重新變成可用。 如需相關資訊，請參閱 WebSphere Liberty 產品說明文件中的 交易管理程式 (交易) 。

這是用戶端特性，亦即您需要 IBM MQ 9.2.0 或更新版本的資源配接器，而不是 IBM MQ 9.2.0 或更新版本的佇列管理程式。

安裝 IBM MQ 資源配接器

IBM MQ 資源配接器以資源保存檔 (RAR) 形式提供。 在應用程式伺服器中安裝 RAR 檔。 您可能需要將目錄新增至系統路徑。

關於這項作業

IBM MQ 資源配接器以資源保存檔 (RAR) 形式提供:

- JM 3.0 V 9.3.0 V 9.3.0 對於 Jakarta Messaging 3.0，此檔案稱為 `wmq.jakarta.jmsra.rar`。 RAR 檔包含 IBM MQ classes for Jakarta Messaging 和 Jakarta Connectors Architecture (JCA) 介面的 IBM MQ 實作。
- JMS 2.0 對於 JMS 2.0，此檔案稱為 `wmq.jmsra.rar`。 RAR 檔包含 Java EE Connector Architecture (JCA) 介面的 IBM MQ classes for JMS 和 IBM MQ 實作。

當您在 IBM MQ 產品安裝過程中安裝資源配接器時， RAR 檔會與 IBM MQ classes for JMS 一起安裝在 第 373 頁的表 61 所示的目錄中。

表 61: *IBM MQ* 目錄，包含每一個平台的 *RAR* 檔	
平台	**目錄**
AIX and Linux	*MQ_INSTALLATION_PATH*/java/lib/jca
IBM i	/QIBM/ProdData/mqm/java/lib/jca
Windows	*MQ_INSTALLATION_PATH*\java\lib\jca
z/OS	*MQ_INSTALLATION_PATH*/java/lib/jca

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

您必須使用 IBM MQ 資源配接器，從應用程式伺服器連接至 IBM MQ 。 視您使用的應用程式伺服器而定，資源配接器可能已預先安裝，或者您可能需要自行安裝它。

表 62: 在應用程式伺服器中安裝資源配接器	
應用程式伺服器	**已預先安裝或需要安裝?**
WebSphere Application Server traditional 9.0	IBM MQ 9.0 資源配接器預先安裝在 WebSphere Application Server traditional 9.0 內。 因此，您不需要在 WebSphere Application Server traditional 9.0 中安裝新的資源配接器。
WebSphere Liberty	WebSphere Liberty 不包含 IBM MQ 資源配接器，因此您必須從 Fix Central 個別取得它。
其他 Java EE 或 Jakarta EE 應用程式伺服器	從 Fix Central 分開取得資源配接器，就像 WebSphere Liberty 一樣。

程序

- 如果您要從 WebSphere Liberty 或另一個 Java EE 或 Jakarta EE 應用程式伺服器連接至 IBM MQ ，請下載並安裝 IBM MQ 資源配接器，如 第 373 頁的『在 Liberty 中安裝資源配接器』中所述。
- Linux AIX

  對於 AIX and Linux 系統上的連結連線，請確保包含 Java 原生介面 (JNI) 程式庫的目錄位於系統路徑中。 如需此目錄 (也包含 IBM MQ classes for JMS 程式庫) 的位置，請參閱 第 81 頁的『配置 Java 原生介面 (JNI) 程式庫』。

  Windows 在 Windows 上，在 IBM MQ classes for JMS 安裝期間，此目錄會自動新增至系統路徑。

  提示: 除了設定系統路徑之外， IBM MQ 資源配接器還有一個稱為 nativeLibrary 路徑的內容，可用來指定 JNI 程式庫的位置。 例如，在 WebSphere Liberty 中，這會如下列範例所示配置:

  ```
  <wmqJmsClient nativeLibraryPath="/opt/mqm/java/lib64"/>
  ```

  在用戶端和連結模式下都支援交易。

在 *Liberty* 中安裝資源配接器

如果要從 WebSphere Liberty 或其他 Java EE 或 Jakarta EE 應用程式伺服器連接至 IBM MQ ，您必須使用 IBM MQ 資源配接器。 由於 Liberty 不包含 IBM MQ 資源配接器，您必須從 Fix Central 個別取得它。

開始之前

註: 本主題中的資訊不適用於 WebSphere Application Server traditional 9.0。 IBM MQ 9.0 資源配接器已預先安裝在 WebSphere Application Server traditional 9.0 內。 因此，在此情況下，不需要安裝新的資源配接器。

在開始這項作業之前，請確定您已在機器上安裝 Java runtime environment (JRE)，且 JRE 已新增至系統路徑。

在此安裝程序中使用的 Java 安裝程式不需要以 root 或任何特定使用者身分執行。 唯一的需求是執行它的使用者對您要檔案進入的目錄具有存取權。

對於直至 WebSphere Liberty 8.5.5 Fix Pack 1 的 Liberty 版本，如果僅使用 `ejb-jar.xml` 內的配置來部署 EJB，則 Liberty 設定檔所使用的 WebSphere Application Server 版本必須已套用 APAR PM89890。 這個配置方法用於資源配接器的 安裝驗證程式 (IVT)，因此需要此 APAR， IVT 才能執行。

JM 3.0 V 9.3.0 V 9.3.0 從 IBM MQ 9.3.0 開始，支援 Jakarta Messaging 3.0。 如果要將 Jakarta Messaging 的 IBM MQ 資源配接器與 Liberty 搭配使用，您必須使用支援 Jakarta EE 的 Liberty 版本。 例如，您可以使用 Liberty generic messaging-3.0 特性。

關於這項作業

您可以從 Fix Central 下載之資源配接器的 JAR 檔是可執行的。 當您執行這個執行檔時，它會顯示必須接受的 IBM MQ 授權合約。 它會要求在其中安裝 IBM MQ 資源配接器的目錄。 然後資源配接器 RAR 檔和安裝驗證測試 (IVT) 程式會安裝在該目錄中。 您可以接受預設值，或指定另一個目錄 (可能是應用程式伺服器的資源配接器目錄)，或您系統上的任何其他目錄。 如果目錄不存在，則會在安裝過程中建立該目錄。

在 IBM MQ 9.0 之前，要下載的檔案名稱採用 *V.R.M.F*`-WS-MQ-Java-InstallRA.jar` 格式，例如 `8.0.0.6-WS-MQ-Java-InstallRA.jar`。 從 IBM MQ 9.0 開始，檔名的格式為 *V.R.M.F*`-IBM-MQ-Java-InstallRA.jar`，例如 `9.0.0.0-IBM-MQ-Java-InstallRA.jar`。

在您下載並安裝資源配接器之後，您已準備好在 WebSphere Liberty 中配置它。

程序

1. 從 Fix Central 下載 IBM MQ 資源配接器。
   a) 按一下此鏈結: IBM MQ 資源配接器。
   b) 在顯示的可用修正程式清單中，尋找您 IBM MQ 版本的資源配接器。
   例如:

   ```
   release level: 9.1.4.0-IBM-MQ-Java-InstallRA
   Continuous Delivery Release: 9.1.4 IBM MQ Resource Adapter for use with Application
   Servers
   ```

   然後按一下資源配接器檔名，並遵循下載程序。
2. 從您下載檔案的目錄中輸入下列指令，以開始安裝。
   從 IBM MQ 9.0 開始，指令的格式如下:

   ```
   java -jar V.R.M.F-IBM-MQ-Java-InstallRA.jar
   ```

   其中 *V.R.M.F* 是版本、版次、修正層次及修正套件號碼，而 *V.R.M.F*`-IBM-MQ-Java-InstallRA.jar` 是從 Fix Central 下載的檔案名稱。
   比方說，如果要安裝 IBM MQ 9.1.4 版本的 IBM MQ 資源配接器，您可以使用下列指令:

   ```
   java -jar 9.1.4.0-IBM-MQ-Java-InstallRA.jar
   ```

   註: 若要執行此安裝，您必須在機器上安裝 JRE 並新增至系統路徑。

   當您輸入指令時，會顯示下列資訊:

   ```
   在您可以使用、解壓縮或安裝 IBM MQ 9.1 之前，必須先接受
   1 的條款。 IBM 國際授權合約-評估
   方案 2. IBM 國際程式授權合約及其他
   授權資訊。 請仔細閱讀下列授權合約。

   授權合約可使用來個別檢視
   ```

```
--viewLicenseAgreement 選項。
請按 Enter 鍵立即顯示授權條款，或按 'x' 跳過。
```

3. 檢閱並接受授權條款:
   a) 若要顯示授權，請按 Enter 鍵。
      或者，按 x 會跳過授權的顯示畫面。
      在顯示授權之後，或在選取 x 之後立即出現下列訊息，告訴您您可以選擇顯示其他授權條款:

      ```
      其他授權資訊可使用來個別檢視
      --viewLicenseInfo 選項。
      請按 Enter 鍵立即顯示其他授權資訊，或按 'x' 跳過。
      ```

   b) 若要顯示其他授權條款，請按 Enter 鍵。
      或者，按 x 會跳過其他授權條款的顯示畫面。
      在顯示其他授權條款之後，或在選取 x 之後立即顯示下列訊息，要求您接受授權合約:

      ```
      選擇下面的「我同意」選項，即表示您同意
      授權合約及非 IBM 條款 (如果適用的話)。 如果您未
      同意，選取「我不同意」。

      選取 [1] 我同意，或 [2] 我不同意:
      ```

   c) 若要接受授權合約並繼續選取安裝目錄，請選取 1。
      或者，如果您選取 2，則安裝會立即終止。
      如果您選取 1，則會出現下列訊息，要求您選取目標安裝目錄:

      ```
      輸入產品檔案的目錄，或保留空白以接受預設值。
      預設目標目錄是 H: \Liberty\WMQ
      產品檔案的目標目錄?
      ```

4. 指定資源配接器的安裝目錄:
   - 如果您要在預設位置安裝資源配接器，請按 Enter 鍵而不指定值。
   - 如果您要將資源配接器安裝在不同於預設值的位置，請指定您要在其中安裝資源配接器的目錄名稱，然後按 Enter 鍵。

   在選取的位置中安裝檔案之後，會顯示確認訊息，如下列範例所示:

   ```
   將檔案解壓縮至 H: \Liberty\WMQ \wmq
   已順利擷取所有產品檔案。
   ```

   在安裝期間，會在選取的安裝目錄中建立名為 wmq 的新目錄，然後在 wmq 目錄中安裝下列檔案:
   - 安裝驗證測試程式 `wmq.jakarta.jmsra.ivt` (Jakarta Messaging 3.0) 或 `wmq.jmsra.ivt` (JMS 2.0)。
   - IBM MQ RAR 檔 `wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0 或 `wmq.jmsra.rar` (JMS 2.0))。

5. JMS 2.0
   選擇性的: 在 WebSphere Liberty Profile 中配置 Java EE 7 (JMS 2.0) 資源配接器。

   在 Liberty 中配置資源配接器所必須採取的步驟如下。 如需相關資訊，請參閱 WebSphere Application Server 產品說明文件。

   a) 將 wmqJmsClient-2.0 特性新增至 `server.xml` 檔，以容許使用 IBM MQ 資源配接器。
      如需相關資訊，請參閱 第 366 頁的『要使用的資源配接器版本』。
   b) 將參照新增至您已安裝的 `wmq.jmsra.rar` (JMS 2.0) 檔案。

   使用 JNDI 來支援 Servlet 及 MDB 的範例配置可能如下所示:

   ```
   <featureManager>
      <feature>wmqJmsClient-2.0</feature>
      <feature>servlet-3.0</feature>
      <feature>jmsMdb-3.1</feature>
      <feature>jndi-1.0</feature>
      </featureManager>

   <variable name="wmqJmsClient.rar.location"
         value="H:\Liberty\WMQ\wmq\wmq.jmsra.rar"/>
   ```

6. JM 3.0

選擇性的: 在 WebSphere Liberty Profile 中配置 Jakarta EE 9 (Jakarta Messaging 3.0) 資源配接器。

在 Liberty 中配置資源配接器所必須採取的步驟如下。 如需相關資訊，請參閱 WebSphere Application Server 產品說明文件。

a) 將 wmqJmsClient-3.0 特性新增至 `server.xml` 檔案，以容許使用 IBM MQ 資源配接器。
如需相關資訊，請參閱 第 366 頁的『要使用的資源配接器版本』。

b) 將參照新增至您已安裝的 `wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0) 檔案。

使用 JNDI 來支援 Servlet 及 MDB 的範例配置可能如下所示:

```
 <featureManager>
    <feature>wmqJmsClient-3.0</feature>
    <feature>servlet-3.0</feature>
    <feature>jmsMdb-3.1</feature>
    <feature>jndi-1.0</feature>
    </featureManager>

<variable name="wmqJmsClient.rar.location"
      value="H:\Liberty\WMQ\wmq\wmq.jmsra.rar"/>
```

註: 如果您是使用 Open Liberty 而非 WebSphere Liberty Profile，則需要使用通用資源配接器支援特性 "messagingClient-3.0" 來取代 "wmqJmsClient-3.0" ，且配置的其他層面將會不同。 如需詳細資料，請參閱 Open Liberty 說明文件。

配置 IBM MQ 資源配接器

如果要配置 IBM MQ 資源配接器，您可以定義各種 Java Platform, Enterprise Edition Connector Architecture (JCA) 資源，以及選擇性地定義系統內容。 您也必須將資源配接器配置成執行安裝驗證測試 (IVT) 程式。 這很重要，因為 IBM 服務可能需要執行此程式，以指出已正確配置任何非 IBM 應用程式伺服器。

開始之前

這項作業假設您已熟悉 JMS 和 IBM MQ classes for JMS。 用來配置 IBM MQ 資源配接器的許多內容相當於 IBM MQ classes for JMS 物件的內容，且具有相同的功能。

關於這項作業

每個應用程式伺服器都提供自己的一組管理介面。 部分應用程式伺服器提供圖形使用者介面來定義 JCA 資源，但其他應用程式伺服器則需要管理者撰寫 XML 部署計劃。 因此，提供如何為每一部應用程式伺服器配置 IBM MQ 資源配接器的相關資訊，已超出本文件的範圍。

因此，下列步驟僅著重於您需要配置的項目。 如需如何配置 JCA 資源配接器的相關資訊，請參閱應用程式伺服器隨附的文件。

程序

在下列種類中定義 JCA 資源:

- 定義 ResourceAdapter 物件的內容。
  第 377 頁的『ResourceAdapter 物件內容的配置』中說明這些內容，這些內容代表資源配接器的廣域內容，例如診斷追蹤層次。
- 定義 ActivationSpec 物件的內容。
  這些內容決定如何啟動 MDB 來進行入埠通訊。 如需相關資訊，請參閱 第 379 頁的『配置入埠通訊的資源配接器』。
- 定義 ConnectionFactory 物件的內容。
  應用程式伺服器會使用這些內容來建立 JMS ConnectionFactory 物件，以進行出埠通訊。 如需相關資訊，請參閱 第 392 頁的『配置資源配接器以進行出埠通訊』。
- 定義受管理目的地物件的內容。

應用程式伺服器會使用這些內容來建立 JMS Queue 物件或 JMS Topic 物件，以進行出埠通訊。 如需相關資訊，請參閱 第 392 頁的『配置資源配接器以進行出埠通訊』。

- 選擇性的: 定義資源配接器的部署計劃。

  IBM MQ 資源配接器 RAR 檔含有一個稱為 `META-INF/ra.xml` 的檔案，其中含有資源配接器的部署描述子。 這個部署描述子是由位於 https://xmlns.jcp.org/xml/ns/javaee/connector_1_7.xsd 的 XML 綱目所定義，且包含資源配接器及其所提供服務的相關資訊。 應用程式伺服器也可能需要資源配接器的部署計劃。 此部署計劃特定於應用程式伺服器。

視需要指定 JVM 系統內容:

- 如果您使用「傳輸層安全 (TLS)」，請將金鑰儲存庫檔和信任儲存庫檔的位置指定為 JVM 系統內容，如下列範例中所示:

  ```
  java ... -Djavax.net.ssl.keyStore=
  key_store_location
           -Djavax.net.ssl.trustStore=trust_store_location
           -Djavax.net.ssl.keyStorePassword=key_store_password
  ```

  這些內容不能是 ActivationSpec 或 ConnectionFactory 物件的內容，且您不能為應用程式伺服器指定多個金鑰儲存庫。 這些內容適用於整個 JVM，因此如果在應用程式伺服器中執行的其他應用程式正在使用 TLS 連線，則可能會影響應用程式伺服器。 應用程式伺服器也可能將這些內容重設為不同的值。 如需搭配使用 TLS 與 IBM MQ classes for JMS 的相關資訊，請參閱 第 216 頁的『搭配使用 TLS 與 IBM MQ classes for JMS』。

- 選擇性的: 必要的話，請配置資源配接器，將警告訊息記載至應用程式伺服器的標準輸出日誌。

  資源配接器日誌、警告和錯誤訊息使用與 IBM MQ classes for JMS 相同的機制。 如需相關資訊，請參閱 IBM MQ classes for JMS 的記載錯誤。 這表示依預設，訊息會移至稱為 `mqjms.log` 的檔案。 如果要將資源配接器配置成額外將警告訊息記載至應用程式伺服器的標準輸出日誌，請為應用程式伺服器設定下列 JVM 系統內容:

  ```
  -Dcom.ibm.msg.client.commonservices.log.outputName=mqjms.log,stdout
  ```

  此內容與用來控制 IBM MQ classes for JMS 追蹤的內容相同。 與 IBM MQ classes for JMS 一樣，也可以使用指向 `jms.config` 檔的系統內容 (請參閱 第 83 頁的『IBM MQ classes for JMS/Jakarta Messaging 配置檔』)。 如需如何設定 JVM 系統內容的相關資訊，請參閱應用程式伺服器說明文件。

配置資源配接器以執行安裝驗證測試

- 將資源配接器配置成執行 IBM MQ 資源配接器所提供的安裝驗證測試 (IVT) 程式。

  如需執行 IVT 程式所需配置內容的相關資訊，請參閱 第 407 頁的『驗證資源配接器安裝架構』。

  這很重要，因為 IBM 服務可能需要執行此程式，以指出已正確配置任何非 IBM 應用程式伺服器。

  重要: 您必須先配置資源配接器，才能執行程式。

ResourceAdapter 物件內容的配置

ResourceAdapter 物件封裝 IBM MQ 資源配接器的廣域內容，例如診斷追蹤層次。 如果要定義這些內容，請依照應用程式伺服器所提供的文件說明，使用資源配接器的機能。

ResourceAdapter 物件有兩組內容:

- 與診斷追蹤相關聯的內容
- 與資源配接器所管理的連線儲存區相關聯的內容

您定義這些內容的方式取決於應用程式伺服器提供的管理介面。 如果您使用 WebSphere Application Server traditional，請參閱 第 379 頁的『WebSphere Application Server traditional 配置』；如果您使用 WebSphere Liberty，請參閱 第 379 頁的『WebSphere Liberty 配置』。 若為其他應用程式伺服器，請參閱應用程式伺服器的產品說明文件。

如需定義與診斷追蹤相關聯之內容的相關資訊，請參閱 追蹤 IBM MQ 資源配接器

資源配接器管理 JMS 連線的內部連線儲存區，這些連線用來將訊息遞送至 MDB。 第 378 頁的表 63 列出與連線儲存區相關聯的 ResourceAdapter 物件的內容。

表 63: 與連線儲存區相關聯的 *ResourceAdapter* 物件的內容			
內容名稱	**類型**	**預設值**	**說明**
maxConnections	字串	50	IBM MQ 佇列管理程式的連線數上限，以及已部署的 MDB 數目上限。
connectionConcurrency	字串	1	共用 JMS 連線的 MDB 數目上限。 無法共用連線，此內容一律具有值 1。
reconnectionRetry 計數	字串	5	當連線失敗時，資源配接器嘗試重新連接 IBM MQ 佇列管理程式的次數上限。
reconnectionRetry 間隔	字串	300 000	在嘗試重新連接 IBM MQ 佇列管理程式之前，資源配接器等待的時間 (毫秒)。
startupRetry 計數	字串	0	啟動時嘗試連接 MDB 的預設次數 (如果在啟動應用程式伺服器時佇列管理程式不在執行中)。
startupRetry 間隔	字串	30 000	啟動連線嘗試之間的預設休眠時間 (毫秒)。
supportMQExtensions	字串	false	將 IBM MQ JMS 行為回復為 JMS 2.0 之前的行為。 如需相關資訊，請參閱第 274 頁的『SupportMQExtensions 內容』。
nativeLibrary 路徑	字串	<empty>	用來載入 IBM MQ JNI 程式庫以允許連結模式連線的路徑。 Windows 在 Windows 上，系統路徑也需要包含相符 IBM MQ 安裝的位置。

當 MDB 部署在應用程式伺服器中，只要未超出 maxConnection 內容指定的連線數目上限，就會建立新的 JMS 連線，並以佇列管理程式啟動交談。 因此， MDB 數目上限等於連線數目上限。 如果已部署的 MDB 數目達到此上限，則任何部署另一個 MDB 的嘗試都會失敗。 如果 MDB 已停止，另一個 MDB 可以使用它的連線。

一般而言，如果要部署許多 MDB ，您必須增加 maxConnections 內容的值。

reconnectionRetry「計數」和 reconnectionRetry「間隔」內容控管當 IBM MQ 佇列管理程式的連線因為網路失敗而失敗時，資源配接器的行為。 當連線失敗時，在 reconnectionRetry「間隔」內容指定的間隔內，資源配接器會暫停將訊息遞送至該連線所提供的所有 MDB。 然後，資源配接器會嘗試重新連接佇列管理程式。 如果嘗試失敗，資源配接器會依照 reconnectionRetry「間隔」內容所指定的間隔，進一步嘗試重新連接，直到達到 reconnectionRetry「計數」內容所強制的限制為止。 如果所有嘗試都失敗，則會永久停止遞送，直到手動重新啟動 MDB 為止。

一般而言， ResourceAdapter 物件不需要管理。 不過，例如，若要在 AIX and Linux 系統上啟用診斷追蹤，您可以設定下列內容:

```
traceEnabled:    true
traceLevel:      10
```

如果資源配接器尚未啟動 (例如，當使用 IBM MQ 資源的應用程式只在用戶端儲存器中執行時) ，則這些內容沒有作用。 在此狀況下，您可以將診斷追蹤的內容設為 Java 虛擬機器 (JVM) 系統內容。 您可以在 **java** 指令上使用 -D 旗標來設定內容，如下列範例所示:

```
java ... -DtraceEnabled=true -DtraceLevel=6
```

您不需要定義 ResourceAdapter 物件的所有內容。 任何未指定的內容都會採用其預設值。 在受管理環境中，最好不要混合兩種指定內容的方式。 如果您混合使用它們， JVM 系統內容的優先順序會高於 ResourceAdapter 物件的內容。

WebSphere Application Server traditional 配置

在 WebSphere Application Server traditional 中，資源配接器可以使用相同的內容，但應該在資源配接器的內容畫面內設定它們 (請參閱 WebSphere Application Server traditional 產品說明文件中的 JMS 提供者設定)。 追蹤是由 WebSphere Application Server traditional 配置的診斷區段所控制。 如需相關資訊，請參閱 WebSphere Application Server traditional 產品說明文件中的 使用診斷提供者 。

WebSphere Liberty 配置

資源配接器是使用 `server.xml` 檔中的 XML 元素來配置，如下列範例所示:

JM 3.0

```
<featureManager>
...
    <feature>messaging-3.0</feature>
...
</featureManager>
    <variable name="wmqJmsClient.rar.location"
        value="F:/_rtc_wmq8005/_build/ship/lib/jca/wmq.jakarta.jmsra.rar"/>
...
        <wmqJmsClient supportMQExtensions="true" logWriterEnabled="true"/>
```

JMS 2.0

```
<featureManager>
...
    <feature>wmqJmsClient-2.0</feature>
...
</featureManager>
    <variable name="wmqJmsClient.rar.location"
        value="F:/_rtc_wmq8005/_build/ship/lib/jca/wmq.jmsra.rar"/>
...
        <wmqJmsClient supportMQExtensions="true" logWriterEnabled="true"/>
```

透過新增下列 XML 元素來啟用追蹤:

```
<logging traceSpecification="JMSApi=all:WAS.j2c=all:"/>
```

配置入埠通訊的資源配接器

如果要配置入埠通訊，請定義一或多個 ActivationSpec 物件的內容。

ActivationSpec 物件的內容會決定訊息驅動 Bean (MDB) 如何從 IBM MQ 佇列接收 JMS 訊息。 MDB 的交易式行為定義在其部署描述子中。

ActivationSpec 物件有兩組內容:

- 用來建立與 IBM MQ 佇列管理程式的 JMS 連線的內容
- 用來建立 JMS 連線消費者的內容，當訊息到達指定的佇列時，會以非同步方式遞送訊息

您定義 ActivationSpec 物件內容的方式，取決於應用程式伺服器所提供的管理介面。

用來建立與 IBM MQ 佇列管理程式的 JMS 連線的內容

第 380 頁的表 64 中的所有內容都是選用的。

表 64: 用來建立 *JMS* 連線之 *ActivationSpec* 物件的內容

內容名稱	類型	有效值 (預設值以粗體顯示)	說明
applicationName	字串	• 呼叫端類別名稱 (如果可用的話) 已調整為不超過 28 個字元。 如果無法使用，則會使用字串 `WebSphere MQ Client for Java`。	用來向佇列管理程式登錄應用程式的名稱。 此應用程式名稱由 **DISPLAY CONN MQSC/PCF** 指令顯示 (其中欄位稱為 **APPLTAG**) 或在 IBM MQ Explorer **應用程式連線** 顯示畫面中 (其中欄位稱為 **App name**)。
brokerCCDurSubQueue [1]	字串	• **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE** • 佇列名稱	連線消費者從中接收可延續訂閱訊息的佇列名稱
brokerCCSub 佇列 [1]	字串	• **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE** • 佇列名稱	連線消費者從中接收不可延續訂閱訊息的佇列名稱
brokerControl 佇列 [1]	字串	• **SYSTEM.BROKER.CONTROL.QUEUE** • 佇列名稱	分配管理系統控制佇列的名稱
brokerQueue 管理程式 [1]	字串	• **"" (空字串)** • 佇列管理程式名稱	分配管理系統執行所在的佇列管理程式名稱
brokerSub 佇列 [1]	字串	• **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE** • 佇列名稱	不可延續訊息消費者從中接收訊息的佇列名稱
brokerVersion [1]	字串	• **unspecified** -在分配管理系統從 V6 移轉至 V7 之後，請設定此內容，以便不再使用 RFH2 標頭。 移轉之後，此內容不再相關。 • **V1** -如果要使用 IBM MQ 發佈/訂閱 broker.This 值是預設值。 • **V2** -以原生模式使用 IBM Integration Bus 的分配管理系統。 如果 TRANSPORT 設為 DIRECT 或 DIRECTHTTP，則此值是預設值。	所使用分配管理系統的版本
ccdtURL	字串	• **NULL** • 統一資源定址器 (URL)	URL，識別包含用戶端通道定義表 (CCDT) 之檔案的名稱和位置，並指定如何存取檔案
CCSID	字串	• **819** • Java 虛擬機器 (JVM) 支援的編碼字集 ID	連線的編碼字集 ID
channel	字串	• **SYSTEM.DEF.SVRCONN** • MQI 通道的名稱	要使用的 MQI 通道名稱
cleanupInterval [1]	整數	• **3 600 000** • 正整數	發佈/訂閱清理公用程式的背景執行間隔 (毫秒)

表 64: 用來建立 *JMS* 連線之 *ActivationSpec* 物件的內容 (繼續)

內容名稱	類型	有效值 (預設值以粗體顯示)	說明
cleanupLevel [1]	字串	• **安全** • 無 • 進階安全 • 強制 • NONDUR	分配管理系統型訂閱儲存庫的清理層次
clientID	字串	• **NULL** • 用戶端 ID	連線的用戶端 ID
cloneSupport	字串	• **DISABLED** -一次只能執行一個可延續主題訂閱者的實例。 • ENABLED-相同可延續主題訂閱者的兩個以上實例可以同步執行，但每一個實例必須在個別 Java 虛擬機器 (JVM) 中執行。	相同可延續主題訂閱者的兩個以上實例是否可以同步執行
connectionFactory 查閱	字串	• **NULL** • ConnectionFactory 物件的 JNDI 名稱	如果設定此內容，ActivationSpec 會在應用程式伺服器的 JNDI 名稱空間中查閱具有指定 JNDI 名稱的 JMS ConnectionFactory 物件，然後使用該物件的內容來建立與 IBM MQ 佇列管理程式的 JMS 連線，只有一個例外。 建立 JMS 連線時將使用的 ActivationSpec 的唯一內容是 clientID。 如需相關資訊，請參閱 第 389 頁的『ActivationSpec connectionFactory 查閱和 destinationLookup 內容』。
ConnectionNameList	字串	• **localhost (1414)** • 由以逗點區隔的項目組成的字串，其中每一個項目都採用下列格式: `HOSTNAME(PORT)` 其中 *HOSTNAME* 是 DNS 名稱或 IP 位址。	用於入埠通訊的 TCP/IP 連線名稱清單。 指定時，**connectionNameList** 會取代 **hostname** 和 **port** 內容。 這個內容用來重新連接多重實例佇列管理程式。 **connectionNameList** 在形式上類似於 **localAddress**，但不得與它混淆。 **localAddress** 指定本端通訊的性質，而 **connectionNameList** 指定如何呼叫到遠端佇列管理程式。

表 64: 用來建立 *JMS* 連線之 *ActivationSpec* 物件的內容 (繼續)

內容名稱	類型	有效值 (預設值以粗體顯示)	說明
V 9.3.0 V 9.3.0 dynamicallyBalanced [4]	布林	• **false** • true	在統一叢集中的應用程式平衡中，是否可以要求這個 MDB 從不同的佇列管理程式接收訊息。
FAILIFQUIESCE	布林	• **true** • false	如果佇列管理程式處於靜止狀態，對特定方法的呼叫是否失敗
headerCompression	字串	• **NONE** • SYSTEM-執行 RLE 訊息標頭壓縮	可用來壓縮連線上的標頭資料的技術清單
hostName	字串	• **localhost** • 主機名稱 • IP 位址	佇列管理程式所在系統的主機名稱或 IP 位址。 當指定 **connectionNameList** 內容時，它會取代 **hostname** 和 **port** 內容。
localAddress	字串	• **NULL** • 下列格式的字串: `[ host_name ][(low_port [, high_port ])]` 其中 *host_name* 是主機名稱或 IP 位址，*low_port* 和 *high_port* 是 TCP 埠號，方括弧表示選用元件	對於佇列管理程式的連線，此內容會指定下列其中一項或兩項: • 要使用的本端網路介面 • 要使用的本端埠或本端埠範圍 **localAddress** 在形式上類似於 **connectionNameList**，但不得與它混淆。**localAddress** 指定本端通訊的性質，而 **connectionNameList** 指定如何呼叫到遠端佇列管理程式。
messageCompression	字串	• **NONE** • 下列一或多個值的清單，以空白字元區隔: RLE ZLIBFAST ZLIBHIGH	可用來壓縮連線上訊息資料的技術清單
messageRetention [1]	布林	• **true** -輸入佇列中仍有不想要的訊息 • false-根據不想要的訊息的處置選項來處理不想要的訊息	連線消費者是否將不想要的訊息保留在輸入佇列上
messageSelection [1]	字串	• **用戶端** • 分配管理系統	決定訊息選擇是由 IBM MQ classes for JMS 還是由分配管理系統完成。 當 brokerVersion 值為 1 時，不支援分配管理系統選取訊息。

表 64: 用來建立 *JMS* 連線之 *ActivationSpec* 物件的內容 (繼續)

內容名稱	類型	有效值 (預設值以粗體顯示)	說明
密碼	字串	• **NULL** • 密碼	建立與佇列管理程式的連線時要使用的預設密碼
pollingInterval [1]	整數	• **5000** • 任何正整數	如果階段作業內每個訊息接聽器的佇列上都沒有適當的訊息，這個值便是每個訊息接聽器在重新嘗試從它的佇列取得訊息之前，所經歷的間隔上限（毫秒）。 如果經常發生階段作業中的任何訊息接聽器都沒有可用的適當訊息，請考量增加此內容的值。 只有在 TRANSPORT 具有 BIND 或 CLIENT 值時，此內容才相關。
port	整數	• **1414** • TCP 埠號	佇列管理程式接聽所在的埠。 當指定 **connectionNameList** 內容時，它會取代 **hostname** 和 **port** 內容。
providerVersion	字串	• **未指定** • 下列其中一種格式的字串 – V.R.M.F – V.R.M – V.R – V 其中 V、R、M 和 F 是大於或等於零的整數值。	MDB 打算連接之佇列管理程式的版本、版次、修正層次及修正套件。
queueManager	字串	• **"" (空字串)** • 佇列管理程式名稱	要連接的佇列管理程式名稱
receiveExit [3]	字串	• **NULL** • 包含一個以上項目 (以逗點區隔) 的字串，其中每個項目都是實作 IBM MQ classes for Java 介面的類別完整名稱 MQReceiveExit	識別通道接收結束程式，或一連串要連續執行的接收結束程式
receiveExit 起始設定	字串	• **NULL** • 包含一或多個使用者資料項目的字串，以逗點區隔	當呼叫通道接收結束程式時，傳給通道接收結束程式的使用者資料

表 64: 用來建立 JMS 連線之 ActivationSpec 物件的內容 (繼續)			
內容名稱	**類型**	**有效值 (預設值以粗體顯示)**	**說明**
rescanInterval [1]	整數	• **5000** • 任何正整數	當點對點網域中的訊息消費者使用訊息選取器來選取要接收的訊息時， IBM MQ classes for JMS 會依佇列的 **MsgDeliverySequence** 屬性所決定的順序，在 IBM MQ 佇列中搜尋適當的訊息。 當 IBM MQ classes for JMS 找到適當的訊息並將它遞送給消費者時， IBM MQ classes for JMS 會從它在佇列中的現行位置回復搜尋下一個適當的訊息。 IBM MQ classes for JMS 會以這種方式繼續搜尋佇列，直到它到達佇列結尾，或直到此內容值所決定的時間間隔 (毫秒) 過期為止。 在每一種情況下， IBM MQ classes for JMS 會回到佇列開頭以繼續搜尋，並開始新的時間間隔。
securityExit [3]	字串	• **NULL** • 實作 IBM MQ classes for Java 介面的類別完整名稱 MQSecurityExit	識別通道安全結束程式
securityExit 起始設定	字串	• **NULL** • 使用者資料的字串	呼叫通道安全結束程式時傳遞給通道安全結束程式的使用者資料
sendExit [3]	字串	• **NULL** • 包含一個以上項目 (以逗點區隔) 的字串，其中每個項目都是實作 IBM MQ classes for Java 介面之類別的完整名稱 MQSendExit	識別通道傳送結束程式，或一連串要連續執行的傳送結束程式
SENDEXITINIT	字串	• **NULL** • 包含一或多個使用者資料項目的字串，以逗點區隔	當呼叫通道傳送結束程式時，傳給通道傳送結束程式的使用者資料
SHARECONVALLOWED	布林	• NO-用戶端連線無法共用其 Socket。 • **YES** -用戶端連線可以共用其 Socket。	如果通道定義相符，用戶端連線是否可以與從相同處理程序至相同佇列管理程式的其他最上層 JMS 連線共用其 Socket
sparseSubscriptions [1]	布林	• **false** -訂閱會接收經常相符的訊息。 • true-訂閱接收不頻繁的相符訊息。 此值需要可以開啟訂閱佇列以進行瀏覽。	控制 TopicSubscriber 物件的訊息擷取原則

表 64: 用來建立 JMS 連線之 ActivationSpec 物件的內容 (繼續)

內容名稱	類型	有效值 (預設值以粗體顯示)	說明
sslCert 儲存庫	字串	• **NULL** • 一個以上 LDAP URL 的字串，以空白區隔。每一個 LDAP URL 都具有下列格式: `ldap://host_name [: port ]` 其中 *host_name* 是主機名稱或 IP 位址，*port* 是 TCP 埠號，方括弧 ([]) 表示選用元件。	保留憑證撤銷清冊 (CRL) 以在 TLS 連線上使用的「輕量型目錄存取通訊協定 (LDAP)」伺服器
SSLCipherSuite	字串	• **NULL** • CipherSuite 的名稱	用於 TLS 連線的 CipherSuite
sslFips 必要 [2]	布林	• **false** • true	TLS 連線是否必須使用 IBM Java JSSE FIPS 提供者 (IBMJSSEFIPS) 支援的 CipherSuite
SSLPeerName	字串	• **NULL** • 識別名稱的範本	對於 TLS 連線，用於檢查佇列管理程式所提供數位憑證中的識別名稱的範本
SSLResetCount	整數	• **0** • 0-999 999 999 範圍內的整數	在重新協議 TLS 所使用的秘密金鑰之前， TLS 連線所傳送及接收的位元組總數
sslSocketFactory	字串	代表類別完整類別名稱的字串，提供 javax.net.ssl.SSLSocketFactory 介面的實作。選擇性地包括要傳遞至建構子方法的引數，以括弧括住。	在受管理物件範圍內建立的任何連線，都會使用從這個 SSLSocketFactory 介面實作取得的 Socket。
statusRefresh 間隔 [1]	整數	• **60000** • 任何正整數	長時間執行交易的重新整理間隔 (毫秒) ， 會偵測訂閱者何時失去與佇列管理程式的連線。 僅當 **subscriptionStore** 具有值 QUEUE 時，此內容才相關。
subscriptionStore [1]	字串	• **分配管理系統** • 移轉 • 佇列	決定 IBM MQ classes for JMS 儲存作用中訂閱相關持續資料的位置

表 64: 用來建立 JMS 連線之 ActivationSpec 物件的內容 (繼續)			
內容名稱	**類型**	**有效值 (預設值以粗體顯示)**	**說明**
transportType	字串	• **用戶端** • BINDINGS • BINDINGS_THEN_CLIENT	佇列管理程式的連線是使用用戶端模式還是連結模式。 如果指定值 BINDINGS_THEN_CLIENT，資源配接器會先嘗試以連結模式建立連線。 如果這個連線嘗試失敗，資源配接器會嘗試建立用戶端模式連線。 z/OS 如果在 WebSphere Application Server for z/OS 系統上執行的啟動規格已配置成使用 BINDINGS_THEN_CLIENT 傳輸模式，且先前建立的連線已中斷，則啟動規格會先嘗試使用 BINDINGS 傳輸模式來嘗試重新連線。 如果 BINDINGS 傳輸模式連線嘗試失敗，啟動規格隨後會嘗試 CLIENT 傳輸模式連線。
username	字串	• **NULL** • 使用者名稱	建立與佇列管理程式的連線時要使用的預設使用者名稱
wildcardFormat	字串	• CHAR-僅辨識在分配管理系統第 1 版中使用的字元萬用字元 • **TOPIC** -僅辨識主題層次萬用字元，如分配管理系統第 2 版中所使用	要使用的萬用字元語法版本

附註:

1. 此內容可以與 IBM MQ classes for JMS 第 70 版搭配使用。
2. 如需使用 sslFips 必要內容的相關重要資訊，請參閱 第 369 頁的『IBM MQ 資源配接器的限制』。
3. 如需如何配置資源配接器以便找到結束程式的相關資訊，請參閱 第 237 頁的『配置 IBM MQ classes for JMS 以使用通道結束程式』。
4. V 9.3.0 不支援 dynamicallyBalanced 內容與 XA 交易支援一起使用。 如果 dynamicallyBalanced 是 "true"，則 MDB 必須配置成停用 XA 交易。

用來建立 JMS 連線消費者的內容

註: 必須明確定義 **destination** 和 **destinationType** 。 第 386 頁的表 65 中的所有其他內容都是選用的。

表 65: 用來建立 JMS 連線消費者之 ActivationSpec 物件的內容			
內容名稱	**類型**	**有效值 (預設值以粗體顯示)**	**說明**
destination	字串	目的地名稱	從中接收訊息的目的地。 **useJNDI** 內容會決定如何解譯此內容的值。

表 65: 用來建立 *JMS* 連線消費者之 *ActivationSpec* 物件的內容 *(*繼續*)*

內容名稱	類型	有效值 (預設值以粗體顯示)	說明
destinationLookup	字串	• **NULL** • 目的地物件的 JNDI 名稱	如果設定此內容，則 ActivationSpec 會在應用程式伺服器的 JNDI 名稱空間中查閱具有指定 JNDI 名稱的 JMS Destination 物件，然後使用該物件的內容來建立 JMS 連線消費者，優先於 ActivationSpec 上指定的其他內容。 如需相關資訊，請參閱 第 389 頁的『ActivationSpec connectionFactory 查閱和 destinationLookup 內容』。
destinationType	字串	• V 9.3.0 V 9.3.0 jakarta.jms.Queue (Jakarta Messaging 3.0) • V 9.3.0 V 9.3.0 jakarta.jms.Topic (Jakarta Messaging 3.0) • javax.jms.Queue (JMS 2.0) • javax.jms.Topic (JMS 2.0)	目的地、佇列或主題的類型
maxMessages	整數	• **1** • 正整數	一次可以指派給伺服器階段作業的訊息數目上限。 如果啟動規格在 XA 交易中遞送訊息至 MDB，不論這個內容的設定為何，都會使用值 1。
maxPool 深度	整數	• **10** • 正整數	連線消費者所使用伺服器階段作業儲存區中的伺服器階段作業數目上限
messageSelector	字串	• **NULL** • SQL92 訊息選取元表示式	指定要遞送哪些訊息的訊息選取器表示式
nonASFTimeout	整數	• **0** • 正整數	正值表示使用非 ASF 遞送。 此值是取得要求等待可能尚未到達之訊息 (含等待呼叫的取得) 的時間 (毫秒)。 預設值 0 表示使用 ASF 遞送。 在下列情況下，此參數有效: • 應用程式正在 WebSphere Application Server 7.0 或更新版本上執行。 • 應用程式正在 WebSphere Liberty 中使用適當層次的 wmqJms 用戶端特性執行。 如需相關資訊，請參閱 第 370 頁的『Liberty 和 IBM MQ 資源配接器』。
nonASFRollback 已啟用	布林	• **false** -即使 MDB 失敗也會耗用訊息 • true-MDB 內的失敗會導致訊息回復至佇列。	如果 MDB 非交易式，訊息遞送是否在 IBM MQ 同步點內。 如果 MDB 已交易，或 **nonASFTimeout** 設為 0，則會忽略。

表 65: 用來建立 *JMS* 連線消費者之 *ActivationSpec* 物件的內容 (繼續)

內容名稱	類型	有效值 (預設值以粗體顯示)	說明
poolTimeout	整數	• **300000** • 正整數	由於閒置而在關閉之前，未使用的伺服器階段作業在伺服器階段作業儲存區中保持開啟的時間 (毫秒)
READAHEADALLOWED	整數	• **DESTINATION** -透過參照佇列或主題定義來判定是否容許先讀。 • DISABLED-不容許先讀。 • ENABLED-容許先讀。 • QUEUE-藉由參照佇列定義來判斷是否容許先讀。 • TOPIC-透過參照主題定義來判定是否容許先讀。	在處理伺服器階段作業以進行破壞性使用之前，是否容許啟動規格瀏覽執行緒使用先讀來從目的地瀏覽多則訊息至內部緩衝區。 **註:** 啟用先讀可能會導致 JMSCC0108 訊息增加或降低效能，如果 MDB 處理速率無法跟上從目的地瀏覽訊息的速率，也會導致這兩者的效能降低。
readAheadClosePolicy	整數	• **ALL** -內部先讀緩衝區中的所有訊息都會在停止之前遞送至 MDB。 • CURRENT-只有現行 MDB 呼叫完成，可能會將訊息留在內部先讀緩衝區中，然後捨棄這些訊息。	當管理者停止 MDB 時，內部先讀緩衝區中的訊息會如何處理。
receiveCCSID	整數	• **0** -使用 JVM `Charset.defaultCharset` • 1208- UTF-8 • 支援的編碼字集 ID	設定佇列管理程式訊息轉換的目標 CCSID 的目的地內容。 除非 **`receiveConversion`** 設為 `QMGR`，否則會忽略此值。
receiveConversion	字串	• **CLIENT_MSG** • QMGR	決定佇列管理程式是否要執行資料轉換的目的地內容。
sharedSubscription	布林	• **False** -MDB 不應該將訂閱開啟為共用訂閱。 • True-MDB 應該將訂閱開啟為共用訂閱 (具有 JMS 2.0 隱含的規則，請參閱 Java.net 中的 JMS 2.0 規格)。	控制如何從共用訂閱驅動 MDB。 如需如何使用此內容的相關資訊，請參閱 第 391 頁的『如何定義 sharedSubscription 內容的範例』。
startTimeout	整數	• **10,000** • 正整數	在排定遞送訊息的工作之後，必須開始將訊息遞送至 MDB 的時間 (毫秒)。 如果超過此時間，則會將訊息回復至佇列。
subscriptionDurability	字串	• **NonDurable** -不可延續訂閱用來將訊息遞送至訂閱主題的 MDB。 • 可延續-可延續訂閱是用來將訊息遞送至訂閱主題的 MDB。	是否使用可延續或不可延續訂閱將訊息遞送至訂閱主題的 MDB
subscriptionName	字串	• **"" (空字串)** • 訂閱名稱	可延續訂閱的名稱

表 65: 用來建立 JMS 連線消費者之 ActivationSpec 物件的內容 (繼續)			
內容名稱	**類型**	**有效值 (預設值以粗體顯示)**	**說明**
useJNDI	布林	• **false** -稱為 destination 的內容會解譯為 IBM MQ 佇列或主題的名稱。 • true-稱為 destination 的內容會解譯為應用程式伺服器 JNDI 名稱空間中下列其中一個物件的名稱: – V 9.3.0 V 9.3.0 jakarta.jms.Queue (Jakarta Messaging 3.0) – V 9.3.0 V 9.3.0 jakarta.jms.Topic (Jakarta Messaging 3.0) – javax.jms.Queue (JMS 2.0) – javax.jms.Topic (JMS 2.0)	Deprecated 決定如何解譯稱為 destination 之內容的值 **註:** 此內容在 IBM MQ 9.0 中已淘汰。 應該改用 destinationLookup 內容 。

內容衝突和相依關係

ActivationSpec 物件可能有衝突的內容。 例如，您可以在連結模式中指定連線的 TLS 內容。 在此情況下，行為由傳輸類型和傳訊網域決定，它是由 **destinationType** 內容決定的點對點或發佈/訂閱。 任何不適用於指定傳輸類型或傳訊網域的內容都會被忽略。

如果您定義的內容需要定義其他內容，但未定義這些其他內容，在 MDB 部署期間呼叫其 validate () 方法時， ActivationSpec 物件會擲出 InvalidProperty 異常狀況。 異常狀況會以相依於應用程式伺服器的方式，向應用程式伺服器的管理者報告。 例如，如果您將 subscriptionDurability 內容設為可延續，指出您想要使用可延續訂閱，則也必須定義 **subscriptionName** 內容。

如果同時定義名稱為 **ccdtURL** 和 **channel** 的內容，則會擲出 InvalidProperty 異常狀況。 不過，如果您只定義 **ccdtURL** 內容，則將稱為 **channel** 的內容保留為其預設值 SYSTEM.DEF.SVRCONN 不會擲出異常狀況，且會使用 **ccdtURL** 內容所識別的用戶端通道定義表來啟動 JMS 連線。

ActivationSpec connectionFactory 查閱和 destinationLookup 內容

JMS 2.0 規格引進了兩個新的 ActivationSpec 內容。 connectionFactory 查閱和 destinationLookup 內容可以提供受管理物件的 JNDI 名稱，以便優先於其他 ActivationSpec 內容使用。

例如，如果在 JNDI 中定義 Connection Factory ，且在啟動規格的 connectionFactory 查閱內容中指定該物件的 JNDI 名稱，則會優先使用 JNDI 中定義的 Connection Factory 的所有內容，而不使用 第 380 頁的表 64 中的內容。

如果在 JNDI 中定義目的地，且在 ActivationSpec 的 destinationLookup 內容中設定 JNDI 名稱，則會優先使用 第 386 頁的表 65 中的值。 如需如何使用這兩個內容的相關資訊，請參閱 第 389 頁的『ActivationSpec connectionFactory 查閱和 destinationLookup 內容』。

這兩個內容可用來指定 ConnectionFactory 和 Destination 物件的 JNDI 名稱，這些物件優先於 第 380 頁的表 64 和 第 386 頁的表 65 中定義的 ActivationSpec 內容。

請務必注意下列要點，以詳細說明這些內容的運作方式。

connectionFactory 查閱

從 JNDI 查閱的 ConnectionFactory 會作為 第 380 頁的表 64 中所列內容的來源。 ConnectionFactory 物件不會用來實際建立任何 JMS 連線，只會查詢物件的內容。 ConnectionFactory 中的這些內容會置換 ActivationSpec 上定義的任何內容。 這有單一異常狀況。 如果 ActivationSpec 已設定 **ClientID** 內容，則這個內容的值會置換 ConnectionFactory 中指定的值。 這是因為一般實務是使用具有多個

ActivationSpecs 的單一 ConnectionFactory 。 這可簡化管理。 不過， JMS 2.0 規格指出每一個從 ConnectionFactory 建立的 JMS Connection 都應該有唯一的 **ClientID**。 因此， ActivationSpecs 必須能夠置換 ConnectionFactory 上設定的任何值。 如果在 ActivationSpec 上未設定 **ClientID** ，則會使用 Connection Factory 上的任何值。

destinationLookup
Destination 和 **UseJndi** 內容定義在 ActivationSpec 上。 如果 **UseJndi** 旗標設為 `true`，則會將目的地內容中指定的文字視為 JNDI 名稱，並從 JNDI 查閱具有該 JNDI 名稱的目的地物件。

destinationLookup 內容的行為方式完全相同。 如果已設定，則會從 JNDI 查閱內容所指定具有 JNDI 名稱的目的地物件。 此內容優先於 **useJNDI** 內容。

Deprecated useJNDI 內容已在 IBM MQ 9.0 中淘汰，因為 **destinationLookup** 內容是執行相同功能的 JMS 2.0 規格或更新版本。

ActivationSpec 內容，在 IBM MQ classes for JMS 中沒有對等項目

ActivationSpec 物件的大部分內容相當於 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 物件的內容，或 IBM MQ classes for JMS IBM MQ classes for Jakarta Messaging 方法的參數。 不過，三個調整內容及一個可用性內容在 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 中沒有對等項目:

startTimeout
在資源配接器排定工作物件將訊息遞送至 MDB 之後，應用程式伺服器的工作管理員等待資源變成可用的時間 (毫秒)。 如果在開始遞送訊息之前已過了此時間，則「工作」物件會逾時，訊息會回復到佇列上，然後資源配接器可以再次嘗試遞送訊息。 警告會寫入診斷追蹤 (如果已啟用的話) ，但不會影響遞送訊息的程序。 您可能預期只有在應用程式伺服器遇到非常高的負載時，才會發生此狀況。 如果狀況定期發生，請考量增加此內容的值，以讓工作管理員更長時間排定訊息遞送。

maxPool 深度
連線消費者所使用伺服器階段作業儲存區中的伺服器階段作業數目上限。 建立伺服器階段作業時，它會啟動與佇列管理程式的交談。 連線消費者使用伺服器階段作業將訊息遞送至 MDB。 較大的儲存區深度容許在大量狀況下同時遞送更多訊息，但會使用更多應用程式伺服器資源。 如果要部署許多 MDB ，請考量縮小儲存區深度，以便在可管理的層次維護應用程式伺服器上的負載。 每一個連線消費者都使用自己的伺服器階段作業儲存區，因此這個內容不會定義所有連線消費者可用的伺服器階段作業總數。

poolTimeout
由於閒置而關閉之前，未使用的伺服器階段作業在伺服器階段作業儲存區中保持開啟的時間 (毫秒)。 訊息工作量的暫時性增加會導致建立其他伺服器階段作業以配送負載，但在訊息工作量回復正常之後，其他伺服器階段作業會保留在儲存區中且不會使用。

每次使用伺服器階段作業時，都會標示時間戳記。 清除器執行緒會定期檢查每一個伺服器階段作業是否已在此內容指定的期間內使用。 如果尚未使用伺服器階段作業，則會關閉伺服器階段作業，並從伺服器階段作業儲存區中移除它。 在過了指定的期間之後，可能不會立即關閉伺服器階段作業，這個內容代表移除之前的閒置期間下限。

useJNDI
如需此內容的說明，請參閱 第 386 頁的表 65。

部署 MDB

如果要部署 MDB ，請先定義 ActivationSpec 物件的內容，並指定 MDB 所需的內容。 下列範例是您可以明確定義的一般內容集:

JM 3.0 V 9.3.0 V 9.3.0

```
channel:           SYSTEM.DEF.SVRCONN
destination:       SYSTEM.DEFAULT.LOCAL.QUEUE
destinationType:   jakarta.jms.Queue
hostName:          192.168.0.42
messageSelector:   color='red'
port:              1414
queueManager:      ExampleQM
transportType:     CLIENT
```

JMS 2.0

```
channel:           SYSTEM.DEF.SVRCONN
destination:       SYSTEM.DEFAULT.LOCAL.QUEUE
destinationType:   javax.jms.Queue
hostName:          192.168.0.42
messageSelector:   color='red'
port:              1414
queueManager:      ExampleQM
transportType:     CLIENT
```

應用程式伺服器會使用這些內容來建立 ActivationSpec 物件，然後與 MDB 相關聯。 ActivationSpec 物件的內容會決定如何將訊息遞送至 MDB。 如果 MDB 需要分散式交易，但資源配接器不支援分散式交易，則 MDB 部署會失敗。 如需如何安裝資源配接器以支援分散式交易的相關資訊，請參閱 第 372 頁的『安裝 IBM MQ 資源配接器』。

如果有多個 MDB 從相同目的地接收訊息，則在點對點網域中傳送的訊息只會由一個 MDB 接收，即使其他 MDB 也有資格接收該訊息。 特別是，如果兩個 MDB 使用不同的訊息選取器，且送入訊息符合兩個訊息選取器，則只有其中一個 MDB 會接收訊息。 選擇接收訊息的 MDB 未定義，您無法依賴接收訊息的特定 MDB。 所有合格 MDB 都會接收在發佈/訂閱網域中傳送的訊息。

在某些情況下，遞送至 MDB 的訊息可能會回復到 IBM MQ 佇列。 例如，如果在隨後回復的工作單元內遞送訊息，則可能會發生此回復。 已回復的訊息會重新遞送，但格式不正確的訊息可能會反覆地導致 MDB 失敗，因此無法遞送。 這類訊息稱為有害訊息。 您可以配置 IBM MQ，讓 IBM MQ classes for JMS 自動將有害訊息傳送至另一個佇列，以進一步調查或捨棄訊息。

如需如何處理有害訊息的詳細資料，請參閱 第 199 頁的『在 IBM MQ classes for JMS 中處理有害訊息』。

相關概念

指定在執行時期於 MQI 用戶端上僅使用 FIPS 認證的 CipherSpecs

AIX, Linux, and Windows 的聯邦資訊存取安全標準 (FIPS)

相關工作

在 WebSphere Application Server 中配置 JMS 資源

如何定義 *sharedSubscription* 內容的範例

您可以在 WebSphere Liberty `server.xml` 檔內定義啟動規格的 sharedSubscription 內容。 或者，您可以使用註釋來定義訊息驅動 Bean (MDB) 內的內容。

範例: 在 Liberty `server.xml` 檔案內定義

在 WebSphere Liberty `server.xml` 檔案內，您可以定義啟動規格，如下列範例所示。 此範例會在 localhost/port 1490 上建立佇列管理程式的可延續共用訂閱。

```
<jmsActivationSpec id="SubApp/SubscribingEJB/SubscribingMDB" authDataRef="JMSConnectionAlias">
<properties.wmqJms hostName="localhost" port="1490" maxPoolDepth="5"
subscriptionName="MySubName"
subscriptionDurability="DURABLE" sharedSubscription="true"/>
</jmsActivationSpec>
```

範例: 在 MDB 內定義

您也可以使用註釋在 MDB 內定義 sharedSubscription 內容，如下列範例所示:

```
@ActioncationConfigProperty(propertyName ="sharedSubscription",
propertyValue = "true")
```

下列範例顯示使用註釋方法的 MDB 程式碼片段:

JM 3.0 V 9.3.0 V 9.3.0

```
/**
 * Message-Driven Bean example using Annotations for configuration
 */
@MessageDriven(
```

```
  activationConfig = {
     @ActivationConfigProperty(
        propertyName = "destinationType", propertyValue = "jakarta.jms.Topic"),
     @ActivationConfigProperty(
        propertyName = "sharedSubscription", propertyValue = "TRUE"),
     @ActivationConfigProperty(
        propertyName = "destination", propertyValue = "JNDI_TOPIC_NAME")
   },
   mappedName = "Stock/IBM")
public class SubscribingMDB implements MessageListener {

   // Default constructor.
   public SubscribingMDB() {
   }

   // @see MessageListener#onMessage(Message)
   public void onMessage(Message message) {
      // implement business logic here
   }

}
```

JMS 2.0

```
/**
 * Message-Driven Bean example using Annotations for configuration
 */
@MessageDriven(
  activationConfig = {
     @ActivationConfigProperty(
        propertyName = "destinationType", propertyValue = "javax.jms.Topic"),
     @ActivationConfigProperty(
        propertyName = "sharedSubscription", propertyValue = "TRUE"),
     @ActivationConfigProperty(
        propertyName = "destination", propertyValue = "JNDI_TOPIC_NAME")
   },
   mappedName = "Stock/IBM")
public class SubscribingMDB implements MessageListener {

   // Default constructor.
   public SubscribingMDB() {
   }

   // @see MessageListener#onMessage(Message)
   public void onMessage(Message message) {
      // implement business logic here
   }

}
```

相關概念

訂閱者和訂閱

訂閱延續性

第 273 頁的『複製及共用訂閱』

在 IBM MQ 8.0 或更新版本中，有兩種方法可讓多個消費者存取相同的訂閱。 這兩種方法是使用複製的訂閱或使用共用訂閱。

配置資源配接器以進行出埠通訊

如果要配置出埠通訊，請定義 ConnectionFactory 物件和受管理目的地物件的內容。

使用出埠通訊的範例

使用出埠通訊時，在應用程式伺服器中執行的應用程式會啟動與佇列管理程式的連線，然後以同步方式將訊息傳送至其佇列並從其佇列接收訊息。 例如，下列 Servlet 方法 doGet() 使用出埠通訊:

JMS 3.0 V 9.3.0 V 9.3.0

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {

...

// Look up ConnectionFactory and Queue objects from the JNDI namespace
```

```
     InitialContext ic = new InitialContext();
     ConnectionFactory cf = (jakarta.jms.ConnectionFactory) ic.lookup("myCF");
     Queue q = (jakarta.jms.Queue) ic.lookup("myQueue");

// Create and start a connection

     Connection c = cf.createConnection();
     c.start();

// Create a session and message producer

     Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
     MessageProducer pr = s.createProducer(q);

// Create and send a message

     Message m = s.createTextMessage("Hello, World!");
     pr.send(m);

// Create a message consumer and receive the message just sent

     MessageConsumer co = s.createConsumer(q);
     Message mr = co.receive(5000);

// Close the connection

      c.close();
}
```

JMS 2.0

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {

...

// Look up ConnectionFactory and Queue objects from the JNDI namespace

     InitialContext ic = new InitialContext();
     ConnectionFactory cf = (javax.jms.ConnectionFactory) ic.lookup("myCF");
     Queue q = (javax.jms.Queue) ic.lookup("myQueue");

// Create and start a connection

     Connection c = cf.createConnection();
     c.start();

// Create a session and message producer

     Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
     MessageProducer pr = s.createProducer(q);

// Create and send a message

     Message m = s.createTextMessage("Hello, World!");
     pr.send(m);

// Create a message consumer and receive the message just sent

     MessageConsumer co = s.createConsumer(q);
     Message mr = co.receive(5000);

// Close the connection

      c.close();
}
```

當 Servlet 收到 HTTP GET 要求時，它會從 JNDI 名稱空間擷取 ConnectionFactory 物件和「佇列」物件，並使用這些物件將訊息傳送至 IBM MQ 佇列。 然後， Servlet 會接收它已傳送的訊息。

出埠通訊所需的資源

若要配置出埠通訊，請定義下列種類的 Java EE Connector Architecture (JCA) 資源:

- ConnectionFactory 物件的內容，應用程式伺服器用來建立 JMS ConnectionFactory 物件。
- 受管理目的地物件的內容，應用程式伺服器用來建立 JMS Queue 物件或 JMS Topic 物件。

您定義這些內容的方式取決於應用程式伺服器所提供的管理介面。 應用程式伺服器所建立的 ConnectionFactory、「佇列」和「主題」物件會連結至 JNDI 名稱空間，應用程式可以從中擷取它們。

一般而言，您會針對應用程式可能需要連接的每一個佇列管理程式，各定義一個 ConnectionFactory 物件。 您為應用程式在點對點網域中可能需要存取的每一個佇列定義一個「佇列」物件。 並且您為應用程式可能想要發佈或訂閱的每一個主題定義一個「主題」物件。 ConnectionFactory 物件可以獨立於網域。 或者，它可以是網域特定的 QueueConnectionFactory 物件 (用於點對點網域) 或 TopicConnectionFactory 物件 (用於發佈/訂閱網域)。

提示: 對於 JMS 2.0，可以使用 Connection Factory 來建立這兩種連線和環境定義。 因此，連線儲存區可以與包含混合連線和環境定義的 Connection Factory 相關聯。 建議僅將一個 Connection Factory 用於建立連線或建立環境定義。 這可確保該 Connection Factory 的連線儲存區只包含一種類型的物件，從而使該儲存區更為有效率。

ConnectionFactory 物件的內容

第 394 頁的表 66 列出 ConnectionFactory 物件的內容。 應用程式伺服器會使用這些內容來建立 JMS ConnectionFactory 物件。

表 *66: ConnectionFactory* 物件的內容

內容名稱	類型	有效值 (預設值以粗體顯示)	說明
applicationName	字串	• 呼叫端類別名稱 (如果可用的話) 已調整為不超過 28 個字元。 如果無法使用，則會使用字串 `WebSphere MQ Client for Java`。	用來向佇列管理程式登錄應用程式的名稱。 此應用程式名稱由 **`DISPLAY CONN MQSC/PCF`** 指令 (其中欄位稱為 **APPLTAG**) 或在「IBM MQ 瀏覽器 **應用程式連線**」顯示畫面中 (其中欄位稱為 **App name**) 顯示。
brokerCCSub 佇列	字串	• **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE** • 佇列名稱	連線消費者從中接收不可延續訂閱訊息的佇列名稱。
brokerControl 佇列	字串	• **SYSTEM.BROKER.CONTROL.QUEUE** • 佇列名稱	分配管理系統控制佇列的名稱。
brokerPub 佇列	字串	• **SYSTEM.BROKER.DEFAULT.STREAM** • 佇列名稱	傳送已發佈訊息的佇列名稱 (串流佇列)。
brokerQueue 管理程式	字串	• **"" (空字串)** • 佇列管理程式名稱	分配管理系統執行所在的佇列管理程式名稱。
brokerSub 佇列	字串	• **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE** • 佇列名稱	不可延續訊息消費者從中接收訊息的佇列名稱。 如需相關資訊，請參閱 BROKERSUBQ 內容。

表 66: *ConnectionFactory* 物件的內容 (繼續)

內容名稱	類型	有效值 (預設值以粗體顯示)	說明
brokerVersion	字串	• **unspecified** -在分配管理系統從 V6 移轉至 V7 之後，請設定此內容，以便不再使用 RFH2 標頭。 移轉之後，此內容不再相關。 • **V1** -使用「 IBM MQ 發佈/訂閱」分配管理系統。 如果 TRANSPORT 設為 BIND 或 CLIENT ，則此值是預設值。 • **V2** -以原生模式使用 IBM Integration Bus 的分配管理系統。 如果 TRANSPORT 設為 DIRECT 或 DIRECTHTTP ，則此值是預設值。	正在使用的分配管理系統版本。
ccdtURL	字串	• **NULL** • 統一資源定址器 (URL)	URL ，識別包含用戶端通道定義表 (CCDT) 之檔案的名稱和位置，並指定如何存取檔案。
CCSID	字串	• **819** • Java 虛擬機器 (JVM) 支援的編碼字集 ID	連線的編碼字集 ID。
channel	字串	• **SYSTEM.DEF.SVRCONN** • MQI 通道的名稱	要使用的 MQI 通道名稱。
cleanupInterval	整數	• **3 600 000** • 正整數	發佈/訂閱清理公用程式背景執行之間的間隔 (毫秒)。
cleanupLevel	字串	• **安全** • 無 • 進階安全 • 強制 • NONDUR	分配管理系統型訂閱儲存庫的清理層次。
clientID	字串	• **NULL** • 用戶端 ID	連線的用戶端 ID。
cloneSupport	字串	• **DISABLED** -一次只能執行一個可延續主題訂閱者的實例。 • ENABLED-相同可延續主題訂閱者的兩個以上實例可以同步執行，但每一個實例必須在個別 Java 虛擬機器 (JVM) 中執行。	相同可延續主題訂閱者的兩個以上實例是否可以同步執行。

表 66: *ConnectionFactory* 物件的內容 (繼續)

內容名稱	類型	有效值 (預設值以粗體顯示)	說明
ConnectionNameList	字串	• **localhost (1414)** • 由以逗點區隔的項目組成的字串，其中每一個項目都採用下列格式: `HOSTNAME(PORT)` 其中 *HOSTNAME* 是 DNS 名稱或 IP 位址。	用於出埠通訊的 TCP/IP 連線名稱清單。 **connectionNameList** 取代 **hostname** 和 **port** 內容。 這個內容用來重新連接多重實例佇列管理程式。 **connectionNameList** 在形式上類似於 **localAddress**，但不得與它混淆。 **localAddress** 指定本端通訊的性質，而 **connectionNameList** 指定如何呼叫到遠端佇列管理程式。
FAILIFQUIESCE	布林	• **true** • false	如果佇列管理程式處於靜止狀態，則對特定方法的呼叫是否失敗。
headerCompression	字串	• **NONE** • 執行 SYSTEM-RLE 訊息標頭壓縮。	可用來壓縮連線上標頭資料的技術清單。
hostName	字串	• **localhost** • 主機名稱 • IP 位址	佇列管理程式所在系統的主機名稱或 IP 位址。 當指定 **connectionNameList** 內容時，它會取代 **hostname** 和 **port** 內容。
localAddress	字串	• **NULL** • 下列格式的字串: `[ host_name ][(low_port [, high_port ])]` 其中 *host_name* 是主機名稱或 IP 位址， *low_port* 和 *high_port* 是 TCP 埠號，方括弧表示選用元件	對於佇列管理程式的連線，此內容指定下列其中一項或兩項: • 要使用的本端網路介面 • 要使用的本端埠或本端埠範圍 **localAddress** 在形式上類似於 **connectionNameList**，但不得與它混淆。 **localAddress** 指定本端通訊的性質，而 **connectionNameList** 指定如何呼叫到遠端佇列管理程式。
messageCompression	字串	• **NONE** • 下列一或多個值的清單，以空白字元區隔: RLE ZLIBFAST ZLIBHIGH	可用來壓縮連線上訊息資料的技術清單。
messageSelection	字串	• **用戶端** • 分配管理系統	決定訊息選擇是由 IBM MQ classes for JMS 還是由分配管理系統完成。 當 **brokerVersion** 值為 1 時，不支援分配管理系統選取訊息。

表 66: *ConnectionFactory* 物件的內容 (繼續)			
內容名稱	**類型**	**有效值 (預設值以粗體顯示)**	**說明**
密碼	字串	• **NULL** • 密碼	建立與佇列管理程式的連線時要使用的預設密碼。
pollingInterval	整數	• **5000** • 任何正整數	如果階段作業內每個訊息接聽器的佇列上都沒有適當的訊息，這個值便是每個訊息接聽器在重新嘗試從它的佇列取得訊息之前，所經歷的間隔上限（毫秒）。 如果經常發生階段作業中的任何訊息接聽器都沒有可用的適當訊息，請考量增加此內容的值。 僅當 **TRANSPORT** 具有值 BIND 或 CLIENT 時，此內容才相關。
port	整數	• **1414** • TCP 埠號	佇列管理程式接聽所在的埠。 當指定 **connectionNameList** 內容時，它會取代 **hostname** 和 **port** 內容。
providerVersion	字串	• **未指定** • 下列其中一種格式的字串 – V.R.M.F – V.R.M – V.R – V 其中 V、R、M 和 F 是大於或等於零的整數值。	應用程式準備連接的佇列管理程式的版本、版次、修正層次和修正套件。
pubAck 間隔	整數	• **25** • 正整數	在 IBM MQ classes for JMS 要求分配管理系統的確認通知之前，發佈者已發佈的訊息數。
queueManager	字串	• **"" (空字串)** • 佇列管理程式名稱	要連接的佇列管理程式的名稱。
receiveExit [3]	字串	• **NULL** • 包含一個以上項目 (以逗點區隔) 的字串，其中每個項目都是實作 IBM MQ classes for Java 介面的類別完整名稱 MQReceiveExit	識別通道接收結束程式，或一連串要連續執行的接收結束程式。
receiveExit 起始設定	字串	• **NULL** • 包含一或多個使用者資料項目的字串，以逗點區隔	當呼叫通道接收結束程式時，傳給通道接收結束程式的使用者資料。

表 66: *ConnectionFactory* 物件的內容 (繼續)

內容名稱	類型	有效值 (預設值以粗體顯示)	說明
rescanInterval	整數	• **5000** • 任何正整數	當點對點網域中的訊息消費者使用訊息選取器來選取要接收的訊息時， IBM MQ classes for JMS 會依佇列的 **MsgDeliverySequence** 屬性所決定的順序，在 IBM MQ 佇列中搜尋適當的訊息。 當 IBM MQ classes for JMS 找到適當的訊息並將它遞送給消費者時， IBM MQ classes for JMS 會從它在佇列中的現行位置回復搜尋下一個適當的訊息。 IBM MQ classes for JMS 會以這種方式繼續搜尋佇列，直到它到達佇列結尾，或直到此內容值所決定的時間間隔 (毫秒) 過期為止。 在每一種情況下， IBM MQ classes for JMS 會回到佇列開頭以繼續搜尋，並開始新的時間間隔。
securityExit [3]	字串	• **NULL** • 實作 IBM MQ classes for Java 介面的類別完整名稱 MQSecurityExit	識別通道安全結束程式。
securityExit 起始設定	字串	• **NULL** • 使用者資料的字串	呼叫通道安全結束程式時傳遞給它的使用者資料。
SENDCHECKCOUNT	整數	• **0** • 任何正整數	在單一非交易式 JMS 階段作業內檢查非同步放置錯誤之間容許的傳送呼叫數。
sendExit [3]	字串	• **NULL** • 包含一個以上項目 (以逗點區隔) 的字串，其中每個項目都是實作 IBM MQ classes for Java 介面之類別的完整名稱 MQSendExit	識別通道傳送結束程式，或一連串要連續執行的傳送結束程式。
SENDEXITINIT	字串	• **NULL** • 包含一或多個使用者資料項目的字串，以逗點區隔	當呼叫通道傳送結束程式時，傳給通道傳送結束程式的使用者資料。
SHARECONVALLOWED	布林	• NO-用戶端連線無法共用其 Socket。 • **YES** -用戶端連線可以共用其 Socket。	在通道定義符合時，用戶端連線能否與其他頂層 JMS 連線（從相同程序到相同佇列管理程式）共用它的 Socket。
sparseSubscriptions	布林	• **false** -訂閱會接收經常相符的訊息。 • true-訂閱接收不頻繁的相符訊息。 此值需要可以開啟訂閱佇列以進行瀏覽。	控制 TopicSubscriber 物件的訊息擷取原則。

表 66: *ConnectionFactory* 物件的內容 (繼續)			
內容名稱	**類型**	**有效值 (預設值以粗體顯示)**	**說明**
sslCert 儲存庫	字串	• **NULL** • 一個以上 LDAP URL 的字串，以空白區隔。 每一個 LDAP URL 都具有下列格式: `ldap://host_name [: port ]` 其中 *host_name* 是主機名稱或 IP 位址， *port* 是 TCP 埠號，方括弧 ([]) 表示選用元件。	保留憑證撤銷清冊 (CRL) 以在 TLS 連線上使用的「輕量型目錄存取通訊協定 (LDAP)」伺服器。
SSLCipherSuite	字串	• **NULL** • CipherSuite 的名稱	用於 TLS 連線的 CipherSuite 。
sslFips 必要 [2]	布林	• **false** • true	TLS 連線是否必須使用 IBM Java JSSE FIPS 提供者 (IBMJSSEFIPS) 支援的 CipherSuite 。
SSLPeerName	字串	• **NULL** • 識別名稱的範本	對於 TLS 連線，這是用來檢查佇列管理程式所提供數位憑證中的識別名稱的範本。
SSLResetCount	整數	• **0** • 0-999 999 999 範圍內的整數	在重新協議 TLS 所使用的秘密金鑰之前， TLS 連線所傳送及接收的位元組總數。
sslSocketFactory	字串	代表類別的完整類別名稱的字串，提供 javax.net.ssl.SSLSocketFactory 介面的實作，可選擇性地包括要傳遞給建構子方法的引數 (以括弧括住)。	在受管理目的地物件範圍中建立的任何連線，都會使用從 SSLSocketFactory 介面的這個實作所取得的 Socket。
statusRefresh 間隔	整數	• **60000** • 任何正整數	長時間執行交易的重新整理間隔 (毫秒) ，會偵測訂閱者何時失去與佇列管理程式的連線。 僅當 **SUBSTORE** 具有值 QUEUE 時，此內容才相關。
subscriptionStore	字串	• **分配管理系統** • 移轉 • 佇列	決定 IBM MQ classes for JMS 儲存作用中訂閱相關持續資料的位置。
targetClient 比對	布林	• **true** • false	傳送至送入訊息的 JMSReplyTo 標頭欄位所識別之佇列的回覆訊息，是否只有在送入訊息具有 MQRFH2 標頭時，才會有 MQRFH2 標頭。 您也可以為啟動規格配置這個內容。 如需相關資訊，請參閱 第 406 頁的『配置啟動規格的 targetClient 比對內容』。

表 66: *ConnectionFactory* 物件的內容 (繼續)			
內容名稱	**類型**	**有效值 (預設值以粗體顯示)**	**說明**
temporaryModel	字串	• **SYSTEM.DEFAULT.MODEL.QUEUE** • SYSTEM.JMS.TEMPQ.MODEL • 任何字串	從中建立 JMS 暫時佇列的模型佇列名稱。 使用 SYSTEM.DEFAULT.MODEL.QUEUE : • 您的應用程式使用將接受非持續訊息的暫時佇列。 • 一次只有一個應用程式會在 ConnectionFactory 指向的佇列管理程式上建立暫時佇列。 請注意 SYSTEM.DEFAULT.MODEL.QUEUE 一次只能由一個應用程式開啟。 使用 SYSTEM.JMS.TEMPQ.MODEL 處於下列狀況: • 當您的應用程式使用將接受持續訊息的暫時佇列時。 • 如果多個應用程式可以連接至 ConnectionFactory 所指向的佇列管理程式，且那些應用程式需要同時建立暫時佇列。 在下列狀況中，定義新的模型佇列，並將 **DEFPSIST** 屬性設為 YES，並將 **DEFSOPT** 屬性設為 SHARED : • 當您的應用程式使用將接受非持續訊息的暫時佇列，且多個應用程式將連接至 ConnectionFactory 所指向的佇列管理程式，且那些應用程式需要同時建立暫時佇列。 建立新的模型佇列時，請將 **temporaryModel** 內容設為新模型佇列的名稱。
tempQPrefix	字串	• **"" (空字串)** • 可用來形成 IBM MQ 動態佇列名稱的字首。 構成字首的規則與構成 IBM MQ 物件描述子 (結構 MQOD) 中 **DynamicQName** 欄位內容的規則相同，但最後一個非空白字元必須是星號 (*)。 如果內容的值是空字串，則 IBM MQ classes for JMS 會使用值 AMQ.* 建立動態佇列時。	用來形成 IBM MQ 動態佇列名稱的字首。

表 66: *ConnectionFactory* 物件的內容 (繼續)			
內容名稱	**類型**	**有效值 (預設值以粗體顯示)**	**說明**
TEMPTOPICPREFIX	字串	任何非空值字串，僅包含 IBM MQ 主題字串的有效字元	建立暫時主題時， JMS 會產生格式為 "TEMP/*TEMPtopicprefix*/*unique_id* " 的主題字串，或者如果此內容保留預設值，則只會產生 "TEMP/*unique_id*"。 指定非空白 **TEMPTOPICPREFIX** 可定義特定的模型佇列，以針對在此連線下建立的暫時主題建立訂閱者的受管理佇列。
transportType	字串	• **用戶端** • BINDINGS • BINDINGS_THEN_CLIENT	佇列管理程式的連線是使用用戶端模式還是連結模式。 如果指定值 BINDINGS_THEN_CLIENT，資源配接器會先嘗試以連結模式建立連線。 如果這個連線嘗試失敗，資源配接器會嘗試建立用戶端模式連線。 z/OS 如果在 WebSphere Application Server for z/OS 系統上執行的啟動規格已配置成使用 BINDINGS_THEN_CLIENT 傳輸模式，且先前建立的連線已中斷，則啟動規格會先嘗試使用 BINDINGS 傳輸模式來嘗試重新連線。 如果 BINDINGS 傳輸模式連線嘗試失敗，啟動規格隨後會嘗試 CLIENT 傳輸模式連線。
username	字串	• **NULL** • 使用者名稱	建立與佇列管理程式的連線時要使用的預設使用者名稱。
wildcardFormat	整數	• CHAR-僅辨識在分配管理系統第 1 版中使用的字元萬用字元 • TOPIC-僅辨識主題層次萬用字元，如分配管理系統第 2 版中所使用	要使用的萬用字元語法版本。

附註:

1. 如需使用 sslFips 必要內容的相關重要資訊，請參閱 第 369 頁的『IBM MQ 資源配接器的限制』。
2. 如需如何配置資源配接器以便找到結束程式的相關資訊，請參閱 第 237 頁的『配置 IBM MQ classes for JMS 以使用通道結束程式』。

下列範例顯示 ConnectionFactory 物件的一般內容集:

```
channel:        SYSTEM.DEF.SVRCONN
hostName:       192.168.0.42
port:           1414
queueManager:   ExampleQM
transportType:  CLIENT
```

受管理目的地物件的內容

應用程式伺服器會使用受管理目的地物件的內容來建立 JMS Queue 物件或 JMS Topic 物件。

第 402 頁的表 67 列出 Queue 物件和 Topic 物件共用的內容。

表 *67:* 佇列物件和主題物件共用的內容

內容名稱	**類型**	**有效值 (預設值以粗體顯示)**	**說明**
CCSID	字串	• **1208** • Java 虛擬機器 (JVM) 支援的編碼字集 ID	目的地的編碼字集 ID。
編碼	字串	• **native** • 三個字元的字串: – 第一個字元指定二進位整數的表示法: - *N* 表示正常編碼。 - *R* 表示反向編碼。 – 第二個字元指定聚集十進位整數的表示法: - *N* 表示正常編碼。 - *R* 表示反向編碼。 – 第三個字元指定浮點數字的表示法: - *N* 表示標準 IEEE 編碼。 - *R* 表示反向 IEEE 編碼。 - *3* 表示 zSeries 編碼。 NATIVE 相當於字串 NNN。	目的地的二進位整數、聚集十進位整數及浮點數字的表示法。
到期	字串	• **APP** -訊息的到期時間由訊息產生者決定。 • UNLIM-訊息永不到期。 • 0-訊息永不到期。 • 代表訊息到期時間 (毫秒) 的正整數。	傳送至目的地之訊息的到期時間。
FAILIFQUIESCE	字串	• **true** • false	如果佇列管理程式處於靜止狀態，則嘗試存取目的地是否失敗。

表 67: 佇列物件和主題物件共用的內容 (繼續)

內容名稱	類型	有效值 (預設值以粗體顯示)	說明
messageBody 樣式	字串	• **未指定** • JMS • MQ	您可以在 JMS 佇列及主題上設定 **messageBodyStyle** 內容: UNSPECIFIED (預設值) • 傳送時，視 WMQ_TARGET_CLIENT 的值而定， IBM MQ classes for JMS 會產生並併入 MQRFH2 標頭。 • 接收時， IBM MQ classes for JMS 會根據 MQRFH2 中的值來設定 JMS 訊息內容 (如果有的話)。 MQRFH2 不會呈現為 JMS 訊息內文的一部分。 JMS • 傳送時， IBM MQ classes for JMS 會自動產生 MQRFH2 標頭，並在 IBM MQ 訊息中包含該標頭。 • 接收時， IBM MQ classes for JMS 會根據 MQRFH2 中的值來設定 JMS 訊息內容 (如果有的話)。 MQRFH2 不會呈現為 JMS 訊息內文的一部分。 MQ • 傳送時， IBM MQ classes for JMS 不會產生 MQRFH2。 • 接收時， IBM MQ classes for JMS 會將 MQRFH2 呈現為 JMS 訊息內文的一部分。
持續性	字串	• **APP** -訊息的持續性由訊息產生者決定。 • QDEF-訊息的持續性由 IBM MQ 佇列的 **DefPersistence** 屬性決定。 • PERS-訊息持續存在。 • 非持續性訊息。 • 高-訊息的持續性由 IBM MQ 佇列的 **NonPersistentMessageClass** 屬性根據 第 215 頁的『JMS 持續訊息』中的說明來決定。	傳送至目的地之訊息的持續性。
priority	字串	• **APP** -訊息的優先順序由訊息產生者決定。 • QDEF-訊息的優先順序由 IBM MQ 佇列的 **DefPriority** 屬性決定。 • 範圍 0 (最低優先順序) 到 9 (最高優先順序) 的整數。	傳送至目的地之訊息的優先順序。

表 67: 佇列物件和主題物件共用的內容 (繼續)			
內容名稱	**類型**	**有效值 (預設值以粗體顯示)**	**說明**
PUTASYNCALLOWED	字串	• QUEUE-藉由參照佇列定義來判斷是否容許非同步放置。 • TOPIC-透過參照主題定義來判定是否容許非同步放置。 • DESTINATION-透過參照佇列或主題定義來判定是否容許非同步放置。 • DISABLED-不容許非同步放置。 • ENABLED-容許非同步放置。	是否容許訊息產生者使用非同步放置來傳送訊息至這個目的地。
READAHEADALLOWED	整數	• **DESTINATION** -透過參照佇列或主題定義來判定是否容許先讀。 • DISABLED-不容許先讀。 • ENABLED-容許先讀。 • QUEUE-藉由參照佇列定義來判斷是否容許先讀。 • TOPIC-透過參照主題定義來判定是否容許先讀。	在接收非持續訊息之前，是否容許訊息消費者及佇列瀏覽器使用先讀，將非持續訊息從目的地取得至內部緩衝區。
receiveCCSID	整數	• **0** -使用 JVM `Charset.defaultCharset` • 1208- UTF-8 • 支援的編碼字集 ID	設定佇列管理程式訊息轉換的目標 CCSID 的目的地內容。除非 **receiveConversion** 設為 QMGR，否則會忽略此值。
receiveConversion	字串	• **CLIENT_MSG** • QMGR	決定佇列管理程式是否要執行資料轉換的目的地內容。
targetClient	字串	• **JMS** -訊息的目標是 JMS 應用程式。 • MQ -訊息的目標是非 JMS IBM MQ 應用程式。	傳送至目的地的訊息目標是否為 JMS 應用程式。 目標為 JMS 應用程式的訊息包含 MQRFH2 標頭。

第 404 頁的表 68 列出「佇列」物件特定的內容。

表 68: 特定於佇列物件的內容			
內容名稱	**類型**	**有效值 (預設值以粗體顯示)**	**說明**
baseQueueManagerName	字串	• **"" (空字串)** • 佇列管理程式名稱	擁有基礎 IBM MQ 佇列的佇列管理程式名稱。
baseQueue 名稱	字串	• **"" (空字串)** • 佇列名稱	基礎 IBM MQ 佇列的名稱。

第 405 頁的表 69 列出 Topic 物件特定的內容。

表 69: 主題物件特定的內容			
內容名稱	**類型**	**有效值 (預設值以粗體顯示)**	**說明**
baseTopic 名稱	字串	• **"" (空字串)** • 主題名稱	基礎主題的名稱。
brokerCCDurSubQueue >	字串	• **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE** • 佇列名稱	連線消費者從中接收可延續訂閱訊息的佇列名稱。
brokerDurSubQueue	字串	• **SYSTEM.JMS.D.SUBSCRIBER.QUEUE** • 佇列名稱	可延續主題訂閱者從中接收訊息的佇列名稱。 如需相關資訊，請參閱 IBM MQ Explorer 文件中的 **BROKEDURRSUBQ** 內容。
brokerPub 佇列	字串	• **未設定** • 佇列名稱	傳送已發佈訊息的佇列名稱 (串流佇列)。 這個內容的值會置換 ConnectionFactory 物件的 **brokerPubQueue** 內容值。 不過，如果您沒有設定這個內容的值，則會改用 ConnectionFactory 物件的 **brokerPubQueue** 內容值。
brokerPubQueueManager	字串	• **"" (空字串)** • 佇列管理程式名稱	擁有佇列的佇列管理程式名稱，在此佇列中傳送針對主題發佈的訊息。
brokerVersion	字串	• **未設定** • 1 • 2	正在使用的分配管理系統版本。 這個內容的值會置換 ConnectionFactory 物件的 **brokerVersion** 內容值。 不過，如果您沒有設定這個內容的值，則會改用 ConnectionFactory 物件的 **brokerVersion** 內容值。

下列範例顯示 Queue 物件的一組內容:

```
expiry:               UNLIM
persistence:          QDEF
baseQueueManagerName: ExampleQM
baseQueueName:        SYSTEM.JMS.TEMPQ.MODEL
```

下列範例顯示 Topic 物件的一組內容:

```
expiry:               UNLIM
persistence:          NON
baseTopicName:        myTestTopic
```

相關工作

指定在執行時期於 MQI 用戶端上僅使用 FIPS 認證的 CipherSpecs

在 WebSphere Application Server 中配置 JMS 資源

相關參考

AIX, Linux, and Windows 的聯邦資訊存取安全標準 (FIPS)

配置啟動規格的 *targetClient* 比對內容

您可以配置啟動規格的 **targetClientMatching** 內容，以便在要求訊息不包含 MQRFH2 標頭時，在回覆訊息中包含 MQRFH2 標頭。 這表示在傳送訊息時，會併入應用程式在回覆訊息上定義的任何訊息內容。

關於這項作業

如果訊息驅動 Bean (MDB) 應用程式透過 IBM MQ JCA 資源配接器啟動規格來耗用不包含 MQRFH2 標頭的訊息，隨後又將回覆訊息傳送至從要求訊息的 JMSReplyTo 欄位建立的 JMS 目的地，則回覆訊息必須包含 MQRFH2 標頭，即使要求訊息不包含，否則也會遺失應用程式在回覆訊息上所定義的任何訊息內容。

targetClientMatching 內容定義只有在送入訊息具有 MQRFH2 標頭時，傳送至送入訊息的 JMSReplyTo 標頭欄位所識別佇列的回覆訊息是否具有 MQRFH2 標頭。 您可以在 WebSphere Application Server traditional 和 WebSphere Liberty 中配置啟動規格的這個內容。

如果您將 **targetClientMatching** 內容的值設為 `false`，則 MQRFH2 標頭可以包含在傳送至 JMS 目的地的回覆訊息中，該 JMS 目的地是從不包含 MQRFH2 的送入要求訊息的 JMSReplyTo 標頭所建立。 這是因為 JMS 目的地上的 **targetClient** 內容設為值 `0`，這表示訊息包含 MQRFH2 標頭。 在出埠訊息中存在 MQRFH2 標頭，允許在傳送至 IBM MQ 佇列時儲存訊息上的使用者定義訊息內容。

如果 **targetClientMatching** 內容設為 `true` ， 且要求訊息不包含 MQRFH2 標頭，則回覆訊息中不會包含 MQRFH2 標頭。

程序

- 在 WebSphere Application Server traditional 中，使用管理主控台將 **targetClientMatching** 內容定義為 IBM MQ 啟動規格上的自訂內容:
  a) 在導覽窗格中，按一下 **資源-> JMS-> 啟動規格**。
  b) 選取您要檢視或變更的啟動規格名稱。
  c) 按一下 **自訂內容-> 新建** ，然後輸入新自訂內容的詳細資料。
  將內容名稱設為 `targetClientMatching`，將類型設為 `java.lang.Boolean` ， 並將值設為 `false`。
- 在 WebSphere Liberty 中，對 `server.xml` 內啟動規格的定義指定 **targetClientMatching** 內容。
  例如:

```
<jmsActivationSpec id="SimpleMDBApplication/SimpleEchoMDB/SimpleEchoMDB">
<properties.wmqJms destinationRef="MDBRequestQ"
queueManager="MY_QMGR" transportType="BINDINGS" targetClientMatching="false"/>
<authData password="********" user="tom"/>
</jmsActivationSpec>
```

相關概念

第 188 頁的『在 JMS 應用程式中建立目的地』
JMS 應用程式可以在執行時期使用階段作業來動態建立目的地，而不是從「 Java 命名和目錄介面 (JNDI)」名稱空間擷取目的地作為受管理物件。 應用程式可以使用統一資源識別碼 (URI) 來識別 IBM MQ 佇列或主題，並選擇性地指定 Queue 或 Topic 物件的一個以上內容。

第 392 頁的『配置資源配接器以進行出埠通訊』
如果要配置出埠通訊，請定義 ConnectionFactory 物件和受管理目的地物件的內容。

IBM MQ WebSphere Liberty 中的訊息驅動 *Bean* 暫停

啟動規格的 **maxSequentialDeliveryFailures** 內容定義在暫停 MDB 之前，資源配接器所容忍的訊息驅動 Bean (MDB) 實例的循序訊息遞送失敗次數上限。

開始之前

您需要注意可能導致 MDB 在 WebSphere Liberty 中暫停的事件集。 資源配接器會將下列任何一項視為訊息遞送失敗:

- MDB 的 **onMessage** 方法擲出未檢查的異常狀況。

- 在將訊息遞送至 MDB 之前，處理資源配接器時發生的 JMSException 。
- 在處理資源配接器時發生的 JMSException ，將訊息遞送至 MDB。
- XA 交易或區域交易，用來耗用要回復的訊息。
- 應用程式伺服器中沒有可用的執行緒，無法將訊息遞送至 MDB。

關於這項作業

maxSequentialDeliveryFailures 內容的預設值是 *-1*，表示 MDB 永不暫停。 任何其他負值都會被視為與 *-1* 相同。 值為:

- *0* 表示 MDB 會在第一個錯誤時暫停
- *1* 表示 MDB 會因兩個循序錯誤而暫停
- *2* 表示 MDB 會因三個循序錯誤而暫停，依此類推。

您只能在 WebSphere Liberty 中，以及當 Liberty 的層次是 18.0.0.4 或更高版本時，為啟動規格配置這個內容。

小心: 如果您在 Liberty 以外的任何應用程式伺服器環境中將屬性設為非預設值，則會忽略該值，並將一則警告訊息寫入日誌。

此外，也可以將 IBM MQ 資源配接器安裝到 WebSphere Liberty 作為一般資源配接器。 這樣做會停用所有 IBM MQ 和 WebSphere Application Server 整合功能，並使資源配接器無法偵測到它正在 Liberty 中執行。 因此，不支援將 **maxSequentialDeliveryFailures** 設為大於或等於 *0* ，並在日誌中產生警告訊息。

程序

- 在 WebSphere Liberty 中，對 server.xml 內啟動規格的定義指定 **maxSequentialDeliveryFailures** 內容。
  例如:

```
<jmsActivationSpec>
     <properties.wmqJms destinationRef="jndi/MDBQ"
                        transportType="BINDINGS"
                        queueManager="MQ21"
                        maxSequentialDeliveryFailures="1"/>
</jmsActivationSpec>
```

相關概念

第 392 頁的『配置資源配接器以進行出埠通訊』
如果要配置出埠通訊，請定義 ConnectionFactory 物件和受管理目的地物件的內容。

驗證資源配接器安裝架構

IBM MQ 資源配接器的安裝驗證測試 (IVT) 程式是以 EAR 檔提供。 若要使用程式，您必須部署它並將部分物件定義為 JCA 資源。

關於這項作業

安裝驗證測試 (IVT) 程式以稱為 wmq.jakarta.jmsra.ivt.ear (Jakarta Messaging 3.0) 或 wmq.jmsra.ivt.ear (JMS 2.0) 的企業保存檔 (EAR) 提供。 這個檔案與 IBM MQ classes for JMS 安裝在 IBM MQ 資源配接器 RAR 檔、 wmq.jakarta.jmsra.rar (Jakarta Messaging 3.0) 或 wmq.jmsra.rar (JMS 2.0) 的相同目錄中。 如需這些檔案安裝位置的相關資訊，請參閱 第 372 頁的『安裝 IBM MQ 資源配接器』。

您必須在應用程式伺服器上部署 IVT 程式。 IVT 程式包含 Servlet 和 MDB ，測試是否可以將訊息傳送至 IBM MQ 佇列，以及從該佇列接收訊息。 您可以使用 IVT 程式來驗證 IBM MQ 資源配接器已正確配置為支援分散式交易。 如果您要在非 IBM 應用程式伺服器中部署 IBM MQ 資源配接器， IBM 服務可能會要求您示範 IVT 運作，以驗證您的應用程式伺服器已正確配置。

在執行 IVT 程式之前，您必須將 ConnectionFactory 物件、「佇列」物件及可能的「啟動規格」物件定義為 JCA 資源，並確保應用程式伺服器從這些定義建立 JMS 物件，並將它們連結至 JNDI 名稱空間。 您可以選擇物件的內容，以符合您自己的 QueueManager 的主機及埠設定，但下列內容集是簡單的範例:

```
ConnectionFactory object:
channel:               SYSTEM.DEF.SVRCONN
hostName:              localhost
port:                  1550
queueManager:          QM1
transportType:         CLIENT
Queue object:
baseQueueManagerName: QM1
baseQueueName:         TEST.QUEUE
```

用來定義 ConnectionFactory、「佇列」和「啟動規格」物件的機制會因您的應用程式伺服器而異。 例如，若要在 WebSphere Liberty 內設定這些內容，請將下列項目新增至應用程式伺服器的 `server.xml` 檔:

JMS 3.0
```
<!-- IVT Connection factory -->
<jmsQueueConnectionFactory connectionManagerRef="ConMgrIVT" jndiName="IVTCF">
   <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostname="localhost" port="1550"
transportType="CLIENT"/>
</jmsQueueConnectionFactory>
<connectionManager id="ConMgrIVT" maxPoolSize="10"/>

<!-- IVT Queues -->
<jmsQueue id="IVTQueue" jndiName="IVTQueue">
   <properties.wmqJms baseQueueName="TEST.QUEUE"/>
</jmsQueue>

<!-- IVT Activation Spec -->
<jmsActivationSpec id="wmq.jakarta.jmsra.ivt/WMQ_IVT_MDB/WMQ_IVT_MDB">
   <properties.wmqJms destinationRef="IVTQueue"
transportType="CLIENT"
queueManager="QM1"
hostName="localhost"
port="1550"
maxPoolDepth="1"/>
</jmsActivationSpec>
```

JMS 2.0
```
<!-- IVT Connection factory -->
<jmsQueueConnectionFactory connectionManagerRef="ConMgrIVT" jndiName="IVTCF">
   <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostname="localhost" port="1550"
transportType="CLIENT"/>
</jmsQueueConnectionFactory>
<connectionManager id="ConMgrIVT" maxPoolSize="10"/>

<!-- IVT Queues -->
<jmsQueue id="IVTQueue" jndiName="IVTQueue">
   <properties.wmqJms baseQueueName="TEST.QUEUE"/>
</jmsQueue>

<!-- IVT Activation Spec -->
<jmsActivationSpec id="wmq.jmsra.ivt/WMQ_IVT_MDB/WMQ_IVT_MDB">
   <properties.wmqJms destinationRef="IVTQueue"
transportType="CLIENT"
queueManager="QM1"
hostName="localhost"
port="1550"
maxPoolDepth="1"/>
</jmsActivationSpec>
```

依預設， IVT 程式預期在 JNDI 名稱空間中連結名為 jms/ivt/IVTCF 的 ConnectionFactory 物件，並連結名為 jms/ivt/IVTQueue 的 Queue 物件。 您可以使用不同的名稱，但如果這麼做，則必須在 IVT 程式的起始頁面上輸入物件的名稱，並適當地修改 EAR 檔。

部署 IVT 程式且應用程式伺服器已建立 JMS 物件並將它們連結至 JNDI 名稱空間之後，您可以完成下列步驟來啟動 IVT 程式。

程序

1. 在 Web 瀏覽器中輸入下列格式的 URL ， 以啟動 IVT 程式:

```
http://app_server_host: port/WMQ_IVT/
```

其中 *app_server_host* 是執行應用程式伺服器之系統的 IP 位址或主機名稱， *port* 是應用程式伺服器接聽所在的 TCP 埠號。 以下是範例：

```
http://localhost:9080/WMQ_IVT/
```

以下是 IVT 程式所顯示的起始頁面範例。

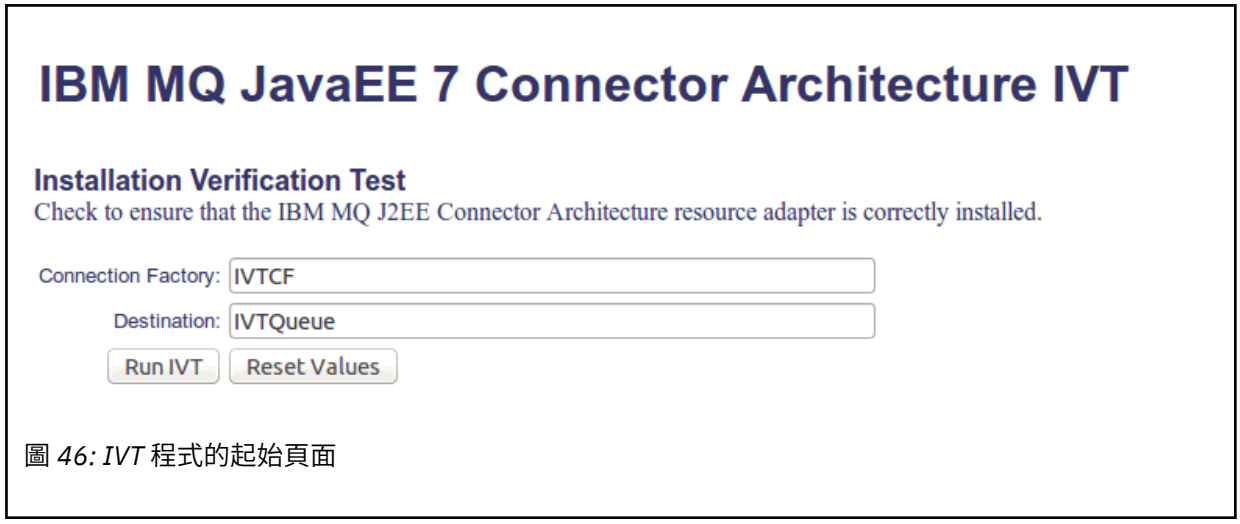


圖 *46: IVT* 程式的起始頁面

2. 如果要執行測試，請按一下 **執行 IVT**。

以下是 IVT 成功時顯示的頁面範例。

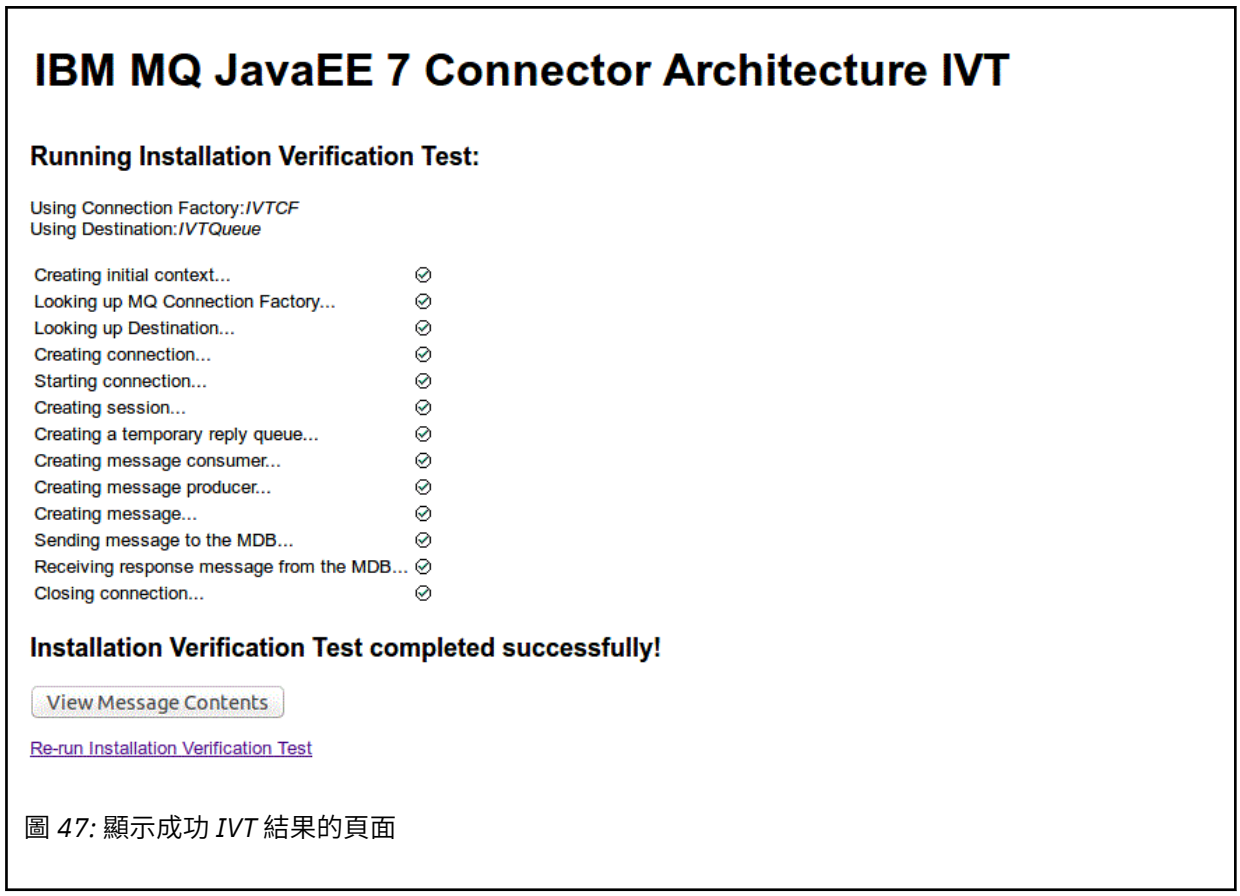


圖 *47:* 顯示成功 *IVT* 結果的頁面

以下是 IVT 失敗時顯示的頁面範例。 若要取得失敗原因的進一步資訊，請按一下 **檢視堆疊追蹤**。

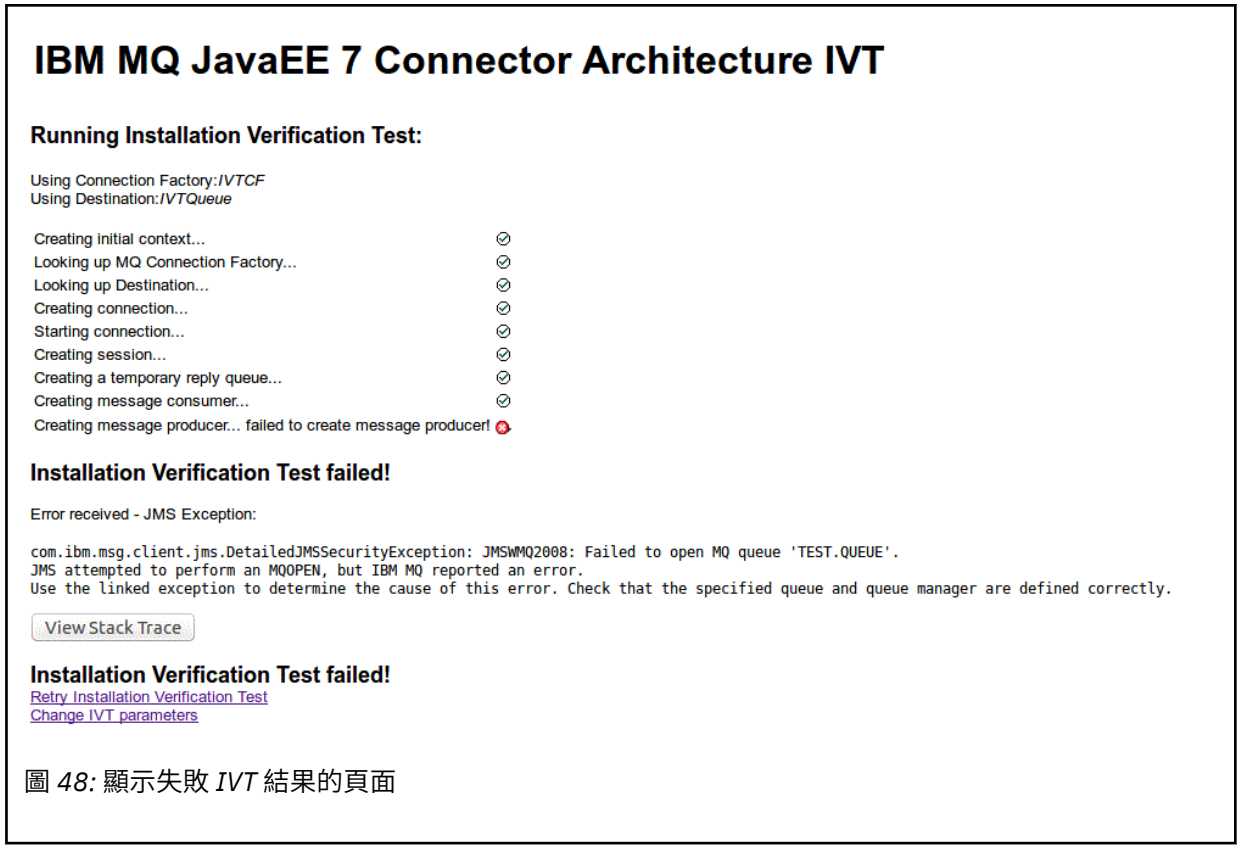


圖 *48:* 顯示失敗 *IVT* 結果的頁面

Windows 在 GlassFish Server 中安裝及測試資源配接器

如果要在 Windows 作業系統上的 GlassFish 伺服器中安裝 IBM MQ 資源配接器，您必須先建立並啟動網域。 然後您可以部署和配置資源配接器，以及部署和執行安裝驗證測試 (IVT) 應用程式。

開始之前

- 這些指示適用於 GlassFish Server 第 4 版。
- 此版本的 GlassFish 伺服器不支援 Jakarta EE。

關於這項作業

此作業假設您具有執行中的 GlassFish Server 應用程式伺服器，並且您熟悉它的標準管理作業。 這項作業也假設您在本端系統上已安裝 IBM MQ ，且您熟悉標準管理作業。

註: 若要完成下列作業步驟，您必須具有正常運作的 IBM MQ 安裝架構，並配置下列物件:

- 在埠 1414 上啟動且使用通道 SYSTEM.DEF.SVRCONN，且使用「用戶端」傳輸來連接。
- 稱為 Q1 的佇列。

程序

1. 啟動 GlassFish Server **asadmin** Shell 程式。
   a) 開啟 Windows 指令行，並導覽至 *GlassFish*/bin 目錄，其中 *GlassFish* 是 GlassFish Server 第 4 版的安裝目錄。
   b) 在指令行中輸入指令 **asadmin** 。
      asadmin 指令會在指令行中開啟 Shell 程式，可讓您建立新網域。
   系統上已啟動 GlassFish 伺服器第 4 版。
2. 建立然後啟動網域。

a) 使用 **create-domain** 指令 (指定埠和網域名稱) 來建立新網域。 在指令行上輸入下列指令:

```
create-domain --adminport port domain_name
```

其中 *port* 是埠號，而 *domain_name* 是您要網域使用的名稱。

註: create-domain 指令具有許多與其相關聯的選用參數。 不過，對於這項作業，您只需要 `--adminport` 參數。 如需相關資訊，請參閱 GlassFish Server 第 4 版的產品說明文件。

如果您指定的埠在使用中，則會出現下列訊息:

domain_name port **的埠正在使用中**

如果您指定的網域名稱正在使用中，則您會收到一則訊息，指出您指定的名稱已在使用中，以及目前無法使用的所有網域名稱清單。

b) 當系統提示輸入使用者名稱和密碼時，請輸入用來透過 Web 瀏覽器登入應用程式伺服器的認證。

如果指令順利完成，則會在指令行上顯示彙總網域建立的訊息，包括訊息 `Command create-domain executed successfully.`

您已順利建立網域。

c) 在指令行中輸入下列指令，以啟動您的網域:

```
start-domain domain_name
```

其中 *domain_name* 是您先前指定的網域名稱。

3. 使用 Web 瀏覽器來存取 GlassFish 應用程式伺服器。

a) 在 Web 瀏覽器的位址列中，輸入下列指令:

```
localhost:port
```

其中 *port* 是您先前在建立網域時指定的埠。

即會顯示「GlassFish 主控台」。

b) 當 GlassFish 主控台已載入且系統提示您輸入使用者名稱及密碼時，請輸入您在步驟 2b 中指定的認證。

4. 將資源配接器上傳至 GlassFish 伺服器 4。

a) 在工具列 一**般作業** 上，選取 **應用程式** 功能表項目，以顯示「**應用程式**」頁面。

b) 按一下 **部署** 按鈕，以開啟「**部署應用程式或模組**」頁面。

c) 按一下 **瀏覽** 按鈕，然後導覽至 `wmq.jmsra.rar` 檔的位置。 選取檔案，然後按一下 **確定**。

5. 建立連線儲存區。

a) 在工具列上的 **資源**下，選取 **連接器** 功能表項目。

b) 然後選取 **連接器連線儲存區** 功能表項目，以開啟「**連接器連線儲存區**」頁面。

c) 按一下 **新建**，以開啟「**新建連接器連線儲存區 (步驟 1/2)**」頁面。

d) 在「**新建連接器連線儲存區 (步驟 1/2)**」頁面上，於 **儲存區名稱** 欄位中輸入儲存區名稱 jms/ivt/IVTCF-Connection-Pool。

e) 在 **資源配接器** 欄位中，選取 `wmq.jmsra`。

f) 在 **連線定義** 欄位中，輸入 `javax.jms.ConnectionFactory`。

g) 選取 **下一步**，然後選取 **完成**。

6. 建立連接器資源。

a) 在工具列上的 **連接器** 功能表下，選取 **連接器資源** 選項，以開啟「**連接器資源**」頁面。

b) 選取 **新建**，以開啟「**新建連接器資源**」頁面。

c) 在 **JNDI 名稱** 欄位中，輸入 `IVTCF`。

d) 在 **儲存區名稱** 欄位中，輸入 `jms/ivt/IVTCF-Connection-Pool`。

e) 將所有其他欄位保留空白。

f) 針對下列每一個內容/值配對，按一下 **新增內容**，然後輸入內容名稱及值，如下列範例所示:

- name: host; value: localhost
- name: port; value 1414
- name: channel; value: SYSTEM.DEF.SVRCONN
- 名稱: queueManager; 值 :QM
- name: transportType; value: CLIENT

註: 請確定您對自己的配置設定使用正確的值，這可能不同於此範例中顯示的值。

g) 在工具列的 **連接器**下，選取 **管理物件資源** 功能表項目，以開啟「 **管理物件資源** 」頁面。

h) 在「 **管理物件資源** 」頁面中，按一下 **新建** 以開啟「 **新建管理物件資源** 」頁面。

i) 在 **JNDI 名稱** 欄位中，輸入 `IVTQueue`。

j) 在 **資源配接器** 欄位中，輸入 `wmq.jmsra`。

k) 在 **資源類型** 欄位中，輸入 `javax.jms.Queue`。

l) 保留 **類別名稱** 欄位的現狀。

m) 針對下列每一個內容/值配對，按一下 **新增內容**，然後輸入內容名稱及值，如下列範例所示:

- name: name; value: IVTQueue
- 名稱: baseQueueManagerName; 值 QM
- 名稱: baseQueue 名稱; 值: Q1

註: 請確定您對自己的配置設定使用正確的值，這可能不同於此範例中顯示的值。

n) 按一下**確定**。

o) 選取 **已啟用** 勾選框，然後按一下 **啟用**。

7. 將 EAR 檔 `wmq.jmsra.ivt.ear` 部署至 GlassFish Server。

a) 按一下工具列中的 **應用程式** 選項，以顯示「 **應用程式** 」頁面。

b) 按一下 **部署** 以新增 IVT 應用程式。

c) 在 **位置** 欄位中，導覽至並選取 `wmq.jmsra.ivt.ear`。

d) 在 **Virtual Servers** 欄位中，選取 **伺服器**，然後按一下 **確定**。

8. 啟動 IVT 程式。

a) 按一下工具列中的 **應用程式** 選項，以顯示「 **應用程式** 」頁面。

b) 按一下「已部署的應用程式」表格中的 `wmq.jmsra.ivt` 。

c) 在「模組和元件」表格中，按一下 **啟動** 按鈕。

d) 選取 http: 鏈結。

e) 按一下 **執行 IVT**。

您已啟動 IVT 程式，如果成功，則會顯示下列輸出:

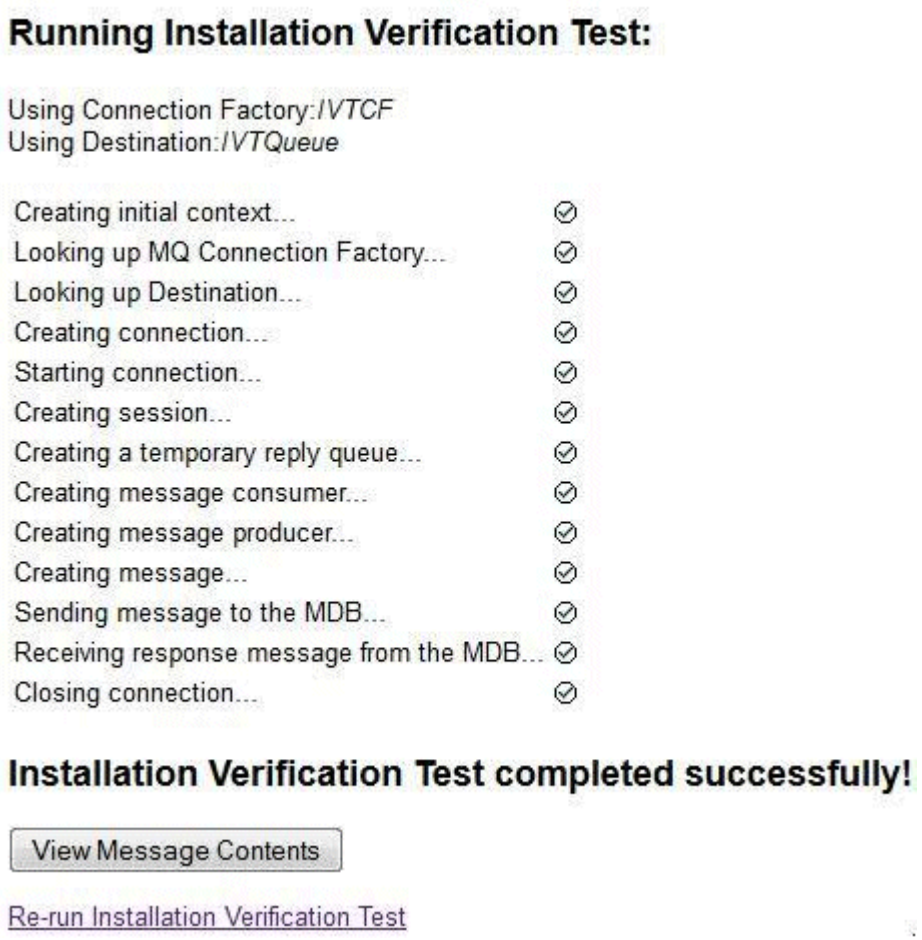


圖 *49:* 順利完成 *IVT* 輸出

在 WildFly 中安裝及測試資源配接器

如果您要在 WildFly V10 中安裝 IBM MQ 資源配接器，您必須先進行一些配置檔變更，以新增 IBM MQ 資源配接器的子系統定義。 然後，您可以安裝並執行安裝驗證測試 (IVT) 應用程式，來部署資源配接器並測試它。

開始之前

- 這些指示適用於 WildFly V10。
- 此 WildFly 版本不支援 Jakarta EE。

關於這項作業

此作業假設您具有執行中 WildFly 應用程式伺服器，且您熟悉它的標準管理作業。 這項作業也假設您有 IBM MQ 安裝架構，且您熟悉標準管理作業。

程序

1. 建立稱為 ExampleQM 的 IBM MQ 佇列管理程式，並依照 第 897 頁的『配置佇列管理程式以接受 Multiplatforms 上的用戶端連線』中的說明來設定它。
   設定佇列管理程式時，請注意下列要點:
   - 必須在埠 1414 上啟動接聽器。
   - 要使用的通道稱為 SYSTEM.DEF.SVRCONN。
   - IVT 應用程式所使用的佇列名稱為 TEST.QUEUE。

模型佇列 SYSTEM.DEFAULT.MODEL.QUEUE 也需要獲得 DSP 和 PUT 權限，此應用程式才能建立暫時回覆佇列。

2. 編輯配置檔 *WildFly_Home*/standalone/configuration/standalone-full.xml，並新增下列子系統:

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:4.0">
    <resource-adapters>
        <resource-adapter id="wmq.jmsra">
            <archive>
                wmq.jmsra.rar
            </archive>
            <transaction-support>NoTransaction</transaction-support>
            <connection-definitions>
                <connection-definition class-
name="com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl"
                                       jndi-name="java:jboss/jms/ivt/IVTCF" enabled="true"
use-java-context="true"
                                       pool-name="IVTCF">
                    <config-property name="channel">SYSTEM.DEF.SVRCONN
                    </config-property>
                    <config-property
                                      name="hostName">localhost
                    </config-property>
                    <config-property name="transportType">
                        CLIENT
                    </config-property>
                    <config-property name="queueManager">
                        ExampleQM
                    </config-property>
                    <config-property name="port">
                        1414
                    </config-property>
                </connection-definition>
                <connection-definition class-
name="com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl"
                                       jndi-name="java:jboss/jms/ivt/JMS2CF" enabled="true"
use-java-context="true"
                                       pool-name="JMS2CF">
                    <config-property name="channel">
                        SYSTEM.DEF.SVRCONN
                    </config-property>
                    <config-property name="hostName">
                        localhost
                    </config-property>
                    <config-property name="transportType">
                        CLIENT
                    </config-property>
                    <config-property name="queueManager">
                        ExampleQM
                    </config-property>
                    <config-property name="port">
                        1414
                    </config-property>
                </connection-definition>
            </connection-definitions>
        <admin-objects>
            <admin-object class-name="com.ibm.mq.connector.outbound.MQQueueProxy"
                          jndi-name="java:jboss/jms/ivt/IVTQueue" pool-name="IVTQueue">
                <config-property name="baseQueueName">
                    TEST.QUEUE
                </config-property>
                </admin-object>
            </admin-objects>
        </resource-adapter>
    </resource-adapters>
</subsystem>
```

3. 將 wmq.jmsra.rar 檔複製到 *WildFly_Home*/standalone/deployments 目錄，將資源配接器部署到伺服器。
4. 將 wmq.jmsra.ivt.ear 檔案複製到 *WildFly_Home*/standalone/deployments 目錄，以部署 IVT 應用程式。
5. 啟動應用程式伺服器，方法是啟動命令提示字元，導覽至 *WildFly_Home*/bin 目錄並執行指令:

```
standalone.bat -c standalone-full.xml
```

6. 執行 IVT 應用程式。

如需相關資訊，請參閱 第 407 頁的『驗證資源配接器安裝架構』。 對於 WildFly，預設 URL 為 `http://localhost:8080/WMQ_IVT/`。

同時使用 IBM MQ 和 WebSphere Application Server

透過 WebSphere Application Server 中的 IBM MQ 傳訊提供者， Java Message Service (JMS) 傳訊應用程式可以使用 IBM MQ 系統作為 JMS 傳訊資源的外部提供者。

關於這項作業

以 Java 撰寫且在 WebSphere Application Server 下執行的應用程式可以使用 Java Message Service (JMS) 規格來執行傳訊。 此環境中的傳訊可以由 IBM MQ 佇列管理程式提供。

使用 IBM MQ 佇列管理程式的好處是連接 JMS 應用程式可以完全參與 IBM MQ 網路的功能，這可讓應用程式與在許多平台上執行的佇列管理程式交換訊息。

應用程式可以對 Queue Connection Factory 物件使用 用戶端傳輸 或 連結傳輸 。 對於連結傳輸，佇列管理程式必須存在於需要連線的應用程式本端。

依預設， IBM MQ 佇列上保留的 JMS 訊息會使用 MQRFH2 標頭來保留部分 JMS 訊息標頭資訊。 許多舊式 IBM MQ 應用程式無法處理具有這些標頭的訊息，且需要自己的性質標頭，例如 MQCIH for CICS Bridge , 或 MQWIH for IBM MQ Workflow 應用程式。 如需這些特殊考量的相關資訊，請參閱 將 JMS 訊息對映至 IBM MQ 訊息。

相關工作

在 WebSphere Application Server 中配置 JMS 資源

將應用程式伺服器配置成使用最新的資源配接器維護層次

將 WebSphere Application Server 與 IBM MQ 搭配使用

IBM MQ 及 IBM MQ for z/OS，可以與 WebSphere Application Server 隨附的預設傳訊提供者搭配使用，或是用作其替代方案。

IBM MQ 傳訊提供者會作為 WebSphere Application Server 的一部分來安裝。 這包含 IBM MQ 資源配接器的某個版本，以及容許佇列管理程式參與應用程式伺服器所管理 XA 交易的 IBM MQ Extended Transactional Client 功能。 透過使用資源配接器，即可將訊息驅動 Bean 配置為使用啟動規格或接聽器埠。

如果要支援應用程式伺服器，必須將 IBM MQ 資源配接器安裝驗證測試程式 部署至應用程式伺服器，並順利執行。 在順利執行 IBM MQ 資源配接器安裝驗證測試程式之後， IBM MQ 資源配接器可以連接至任何支援的 IBM MQ 佇列管理程式。

JMS 從 WebSphere Application Server 到 IBM MQ 的連線

在考量可與 WebSphere Application Server 搭配使用的 IBM MQ 層次之前，請務必瞭解在應用程式伺服器內執行的 Java Message Service (JMS) 應用程式如何可以連接至 IBM MQ 佇列管理程式。

需要存取 IBM MQ 佇列管理程式資源的 JMS 應用程式可以使用下列其中一種傳輸類型來執行此動作:

BINDINGS

當應用程式伺服器及佇列管理程式安裝在相同機器及作業系統映像檔中時，即可使用此傳輸。 使用 BINDINGS 模式時，兩個產品之間的所有通訊皆是使用「交互程序通訊 (IPC)」來執行。

IBM MQ 傳訊提供者不包含以 BINDINGS 模式連接至 IBM MQ 佇列管理程式所需的原生程式庫。 為了使用 BINDINGS 模式連線，必須將 IBM MQ 安裝在與應用程式伺服器相同的機器中，且必須將資源配接器的原生程式庫路徑配置為指向這些程式庫所處的 IBM MQ 目錄。 如需相關資訊，請參閱 WebSphere Application Server 產品說明文件:

- 若為 WebSphere Application Server traditional，請參閱 使用原生程式庫來配置 IBM MQ 傳訊提供者。
- 對於 WebSphere Liberty，請參閱 將 JMS 應用程式部署至 Liberty 以使用 IBM MQ 傳訊提供者。

z/OS 在 z/OS 上，如果您想要以連結模式將 WebSphere Application Server Connection Factory 連接至 IBM MQ 佇列管理程式，您必須在 WebSphere Application Server STEPLIB 連結中指定正確的

IBM MQ 程式庫。 如需相關資訊，請參閱 WebSphere Application Server 產品說明文件中的 IBM MQ 程式庫及 WebSphere Application Server for z/OS STEPLIB 。

用戶端

用戶端傳輸會使用 TCP/IP 在 WebSphere Application Server 與 IBM MQ 之間通訊。 除了可在應用程式伺服器及佇列管理程式位於不同機器中時使用以外，當這兩個產品安裝在相同機器及作業系統映像檔中時，亦可使用 CLIENT 模式。

JMS 應用程式亦可指定 BINDINGS_THEN_CLIENT 傳輸類型。 使用此傳輸類型時，應用程式最初會嘗試使用 BINDINGS 模式來連接至佇列管理程式 - 如果無法執行此作業，則會嘗試 CLIENT 傳輸。

如何尋找安裝在 WebSphere Application Server 內的 IBM MQ 資源配接器版本

如需 WebSphere Application Server 內安裝的 IBM MQ 資源配接器版本的相關資訊，請參閱 Technote WebSphere Application Server 隨附的 WebSphere MQ 資源配接器 (RA) 版本。

您可以使用下列 Jython 及 JACL 指令，來判斷 WebSphere Application Server 目前使用的資源配接器層次:

Jython

```
wmqInfoMBeansUnsplit = AdminControl.queryNames("WebSphere:type=WMQInfo,*")
wmqInfoMBeansSplit = AdminUtilities.convertToList(wmqInfoMBeansUnsplit)
for wmqInfoMBean in wmqInfoMBeansSplit: print wmqInfoMBean; print
AdminControl.invoke(wmqInfoMBean, 'getInfo', '')
```

註: 輸入此指令之後，您需要按兩下**傳回**才能執行它。

JACL

```
set wmqInfoMBeans [$AdminControl queryNames WebSphere:type=WMQInfo,*]
foreach wmqInfoMBean $wmqInfoMBeans {
puts $wmqInfoMBean;
puts [$AdminControl invoke $wmqInfoMBean getInfo [] []]
}
```

更新資源配接器

隨應用程式伺服器安裝之 IBM MQ 資源配接器的更新項目包含在 WebSphere Application Server 修正套件中。 使用 **更新資源配接器 ...** 來更新 IBM MQ 資源配接器 不建議使用「WebSphere Application Server 管理主控台」中的機能，因為這樣做將表示「WebSphere Application Server 修正套件」中提供的更新項目沒有作用。

MQ_INSTALL_ROOT 變數

從 WebSphere Application Server 7.0 開始， MQ_INSTALL_ROOT 僅用於尋找原生程式庫，並由資源配接器上配置的任何原生程式庫路徑置換。

從 WebSphere Application Server 連接至 IBM MQ

小心:

1. 任何受支援版本的 WebSphere Application Server 都可以使用隨附的 IBM MQ 資源配接器，來連接至任何受支援版本的 IBM MQ。
2. 如果使用連結模式， WebSphere Application Server 中的某些程式庫需要符合它所連接的佇列管理程式版本:
   - WebSphere Application Server 必須配置成載入 IBM MQ 9.3 所提供的原生程式庫。 如需相關資訊，請參閱第 81 頁的『配置 Java 原生介面 (JNI) 程式庫』。
   - z/OS 在 z/OS 上，您必須在 WebSphere Application Server STEPLIB 連結中指定正確的 IBM MQ 程式庫。

如需所需 IBM MQ 程式庫的詳細資料，請參閱 IBM MQ 程式庫及 WebSphere Application Server for z/OS STEPLIB 。

如果您在 LINKLIST (LINKLST) 中有一個 IBM MQ 版本的程式庫，您可以使用 STEPLIB 來置換程式庫，以連接至不同版本的 IBM MQ 。

3. 「IBM MQ 資源配接器」版本與佇列管理程式安裝所提供的原生 (共用) 程式庫版本無關。

例如，具有 IBM MQ 8.0 資源配接器的 WebSphere Application Server 8.5 仍然可以使用 IBM MQ 9.0 原生程式庫來管理與 IBM MQ 9.0 佇列管理程式的連結連線。

如需相關資訊，請參閱 第 367 頁的『支援的 IBM MQ 資源配接器陳述式』。

BINDINGS 及 CLIENT 傳輸類型可用來從任何 WebSphere Application Server 版本連接至 IBM MQ 。 對於 BINDINGS 傳輸類型，適用下列限制:

- IBM MQ 必須安裝在與應用程式伺服器相同的機器上。
- WebSphere Application Server 必須配置成載入 IBM MQ 所提供的原生程式庫。
- z/OS 在 z/OS 上，如果您想要以連結模式將 WebSphere Application Server Connection Factory 連接至 IBM MQ 佇列管理程式，則必須在 WebSphere Application Server STEPLIB 連結中指定正確的 IBM MQ 程式庫。

下表顯示支援在其中執行每一個 IBM MQ 資源配接器版本的 WebSphere Application Server 版本。

表 *70:* 將 *WebSphere Application Server* 版本對映至 *IBM MQ* 資源配接器版本。

IBM MQ 資源配接器的版本	**這個資源配接器版本可以在哪一個 WebSphere Application Server 版本中執行?**
IBM MQ 9.0 以及更新版本	資源配接器可以在下列項目中執行: • WebSphere Liberty 的任何 Java EE 7 相容版本。 • WebSphere Application Server traditional 9.0
IBM MQ 8.0	資源配接器可以在任何符合 Java EE 7 標準的 WebSphere Liberty 版本中執行 不支援 IBM MQ 8.0 資源配接器在 WebSphere Application Server traditional 中執行。 已安裝在 WebSphere Application Server traditional 中的資源配接器應該用來連接 IBM MQ 8.0 佇列管理程式。

相關概念

第 367 頁的『支援的 IBM MQ 資源配接器陳述式』
您必須用於應用程式與佇列管理程式之間通訊的 IBM MQ 資源配接器，取決於您是使用 Jakarta Messaging 3.0 API 還是 JMS 2.0 API。

相關資訊

IBM MQ 的系統需求

決定從 WebSphere Application Server 到 IBM MQ 所建立的 TCP/IP 連線數

使用共用交談特性，多個交談可以共用 MQI 通道實例，這也稱為 TCP/IP 連線。

關於這項作業

在 WebSphere Application Server 7 和 WebSphere Application Server 8 內執行且使用 IBM MQ 傳訊提供者標準模式的應用程式會自動使用此特性。 這意味著在相同應用程式伺服器實例內執行的多個應用程式（連接至相同的 IBM MQ 佇列管理程式）可以共用相同的通道實例。

可以透過單一通道實例共用的交談數由 IBM MQ 通道內容 **SHARECNV** 判斷。 伺服器連線通道的此內容預設值是 10。

透過查看 WebSphere Application Server 7 和 WebSphere Application Server 8 所建立的交談數，您可以判定所建立的通道實例數。

如需 IBM MQ 傳訊提供者模式的相關資訊，請參閱 PROVIDERVERSION 標準模式。

相關概念

使用共用交談

在允許共用交談的環境中，交談可以共用 MQI 通道實例。

第 265 頁的『在 IBM MQ classes for JMS 中共用 TCP/IP 連線』

可以建立 MQI 通道的多個實例，以共用單一 TCP/IP 連線。

JMS Connection Factory

在 WebSphere Application Server 內執行的應用程式 (使用 IBM MQ 傳訊提供者 Connection Factory 來建立連線和階段作業)，對於從 Connection Factory 建立的每一個 JMS 連線，以及從 JMS 連線建立的每一個 JMS 階段作業，都有作用中的交談。

已從 Connection Factory 建立的每個 JMS 連線都有一個交談

每個 JMS Connection Factory 具有關聯的連線儲存區，分為兩部分：可用儲存區及作用中的儲存區。 這兩個儲存區最初是空的。

應用程式從 Connection Factory 建立 JMS 連線時，WebSphere Application Server 會查看可用儲存區中是否存在 JMS 連線。 如果存在，它會移至作用中的儲存區，並提供給應用程式。 否則，會建立新的 JMS 連線，放入作用中的儲存區及回到應用程式。 可從 Connection Factory 建立的連線數上限由 Connection Factory Connection 儲存區內容 **Maximum connections** 指定。 此內容的預設值是 10。

應用程式完成 JMS 連線並關閉它之後，連線會從作用中的儲存區移至可用儲存區，從而可以重複使用。 連線儲存區內容 **Unused timeout** 定義 JMS 連線在中斷之前可以保留在可用儲存區中的時間。 此內容的預設值是 1800 秒（30 分鐘）。

首次建立 JMS 連線時，WebSphere Application Server 與 IBM MQ 之間的交談會啟動。 當超出可用儲存區的 **Unused timeout** 內容值時，除非關閉連線，否則交談會保持作用中。

一次用於已從 JMS 連線建立的每個 JMS 階段作業的交談

從 IBM MQ 傳訊提供者 Connection Factory 建立的每一個 JMS 連線都有相關聯的 JMS 階段作業儲存區。 這些階段作業儲存區以與連線儲存區相同的方式工作。 可從單一 JMS 連線建立的 JMS 階段作業數上限由 Connection Factory 階段作業儲存區內容 **Maximum connections** 決定。 此內容的預設值是 10。

交談會在第一次建立 JMS 階段作業時開始，該交談會保持作用中狀態，直到關閉 JMS 階段作業為止，因為它在可用儲存區中的停留時間已超過階段作業儲存區的 **Unused timeout** 內容值。

計算 SHARECNV 內容的值

您可以使用下列公式計算從單一 Connection Factory 至 IBM MQ 的交談數上限：

```
Maximum number of conversations =
    connection Pool Maximum Connections +
   (connection Pool Maximum Connections * Session Pool Maximum Connections)
```

您可以使用下列計算，來計算允許進行此數目的交談而建立的通道實例數：

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

您可以對此計算中的任何餘數四捨五入。

對於使用連線儲存區 **Maximum connections** 及階段作業儲存區 **Maximum connections** 內容預設值的簡式 Connection Factory，此 Connection Factory 的 WebSphere Application Server 與 IBM MQ 之間可以存在的交談數上限為:

```
Maximum number of conversations =
   connection Pool Maximum Connections +
  (connection Pool Maximum Connections * Session Pool Maximum Connections)
```

例如:

```
        = 10 + (10 * 10)
        = 10 + 100
        = 110
```

如果此 Connection Factory 正在使用 **SHARECNV** 內容設為 10 的通道連接至 IBM MQ，則將為此 Connection Factory 建立的通道實例數上限為:

```
Maximum number of channel instances = Maximum number of conversations / SHARECNV for the
channel being used
```

例如:

```
        = 110 / 10
        = 11 (rounded up to nearest connection)
```

啟動規格

配置為使用啟動規格的訊息驅動 Bean 應用程式對於下列項目會使交談處於作用中狀態：用於監視 JMS 目的地的啟動規格，以及用於執行訊息驅動 Bean 實例以處理訊息的每個伺服器階段作業。

下列交談對於配置為使用啟動規格的訊息驅動 Bean 應用程式處於作用中狀態:

- 一次用於監視 JMS 目的地是否有適合訊息的啟動規格的交談。 啟動規格一啟動，此交談就處於啟動狀態，在啟動規格停止之前會一直保留作用中狀態。
- 一次用於執行訊息驅動 Bean 實例以處理訊息的每個伺服器階段作業的交談。

啟動規格進階內容 **Maximum server sessions** 指定針對給定的啟動規格，在任何一次可以處於作用中的伺服器階段作業數目上限。 此內容的預設值是 10。 伺服器階段作業在需要時建立，在啟動規格進階內容 **Server session pool timeout** 指定的時段閒置時關閉。 此內容的預設值是 300000 毫秒（5 分鐘）。

交談在伺服器階段作業建立時啟動，在啟動規格停止時或在伺服器階段作業逾時停止。

這意味著，您可以使用下列公式計算從單一啟動規格至 IBM MQ 的交談數上限:

```
Maximum number of conversations = Maximum server sessions + 1
```

您可以使用下列計算，來找到允許進行此數目的交談而建立的通道實例數:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

您可以對此計算中的任何餘數四捨五入。

對於使用 **Maximum server sessions** 內容預設值的簡式啟動規格，此啟動規格的 WebSphere Application Server 與 IBM MQ 之間可以存在的交談數上限計算為:

```
Maximum number of conversations = Maximum server sessions + 1
```

例如:

```
    = 10 + 1
    = 11
```

如果這個啟動規格是使用 **SHARECNV** 內容設為 10 的通道來連接至 IBM MQ，則所建立的通道實例數目會計算為:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

例如:

```
    = 11 / 10
    = 2 (rounded up to nearest connection)
```

在應用程式伺服器機能 *(ASF)* 模式中執行的接聽器埠

在 ASF 模式中執行的且由訊息驅動 Bean 應用程式使用的接聽器埠會為每個伺服器階段作業建立交談。 一次交談監視目的地是否有適合訊息，另一次交談執行訊息驅動 Bean 實例以處理訊息。 您可以透過階段作業數上限計算每個接聽器埠的交談數。

依預設，接聽器埠將在 ASF 模式中作為 1.1 規格的一部分執行，以定義應用程式伺服器應用於偵測訊息及將訊息遞送至訊息驅動 Bean 以進行處理的機制。 設定為在作業的此預設模式中使用接聽器埠的訊息驅動 Bean 應用程式會建立交談:

一次用於監視目的地是否有適合訊息的接聽器埠的交談
配置為使用 JMS Connection Factory 的接聽器埠。 接聽器埠啟動時，從 Connection Factory 可用儲存區發出 JMS 連線的要求。 停止接聽器埠時，連線會回到可用儲存區。 如需如何使用連線儲存區及它如何影響 IBM MQ 的交談數的相關資訊，請參閱 第 418 頁的『JMS Connection Factory』。

一次用於執行訊息驅動 Bean 實例以處理訊息的每個伺服器階段作業的交談
接聽器埠內容 **Maximum sessions** 指定在給定接聽器埠的任何一次可以處於作用中的伺服器階段作業數目上限。 此內容的預設值是 10。 伺服器階段作業在需要時建立，並利用從階段作業儲存區（與接聽器埠使用的 JMS 連線相關聯）取得的 JMS 階段作業。

如果伺服器階段作業在「訊息接聽器服務」自訂內容 **SERVER.SESSION.POOL.UNUSED.TIMEOUT** 指定的時段閒置，則階段作業會關閉，且使用的 JMS 階段作業會回到階段作業儲存區的可用儲存區。 JMS 階段作業會一直保留在階段作業儲存區的可用儲存區中，直到需要它，或會因為它在可用儲存區中閒置的時間超過階段作業儲存區的 **Unused timeout** 內容的值而關閉。

如需如何使用階段作業儲存區及如何管理 WebSphere Application Server 與 IBM MQ 之間的交談的相關資訊，請參閱 第 418 頁的『JMS Connection Factory』。

如需「訊息接聽器服務」自訂內容 **SERVER.SESSION.POOL.UNUSED.TIMEOUT**，請參閱 WebSphere Application Server 產品說明文件中的 監視接聽器埠的伺服器階段作業儲存區 。

計算從單一接聽器埠至 IBM MQ 的交談數上限

您可以使用下列公式計算從單一接聽器埠至 IBM MQ 的交談數上限:

```
Maximum number of conversations = Maximum sessions + 1
```

您可以使用下列計算，來計算允許進行此數目的交談而建立的通道實例數:

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

您可以對此計算中的任何餘數四捨五入。

對於使用 **Maximum sessions** 內容預設值的簡式接聽器埠，此接聽器埠在 WebSphere Application Server 與 IBM MQ 之間可以存在的交談數上限計算為:

```
Maximum number of conversations = Maximum sessions + 1
```

例如：

```
    = 10 + 1
    = 11
```

如果此接聽器埠使用 **SHARECNV** 內容設定為 10 的通道連接至 IBM MQ，則將建立的通道實例數計算為：

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

例如：

```
    = 11 / 10
    = 2 (rounded up to nearest connection)
```

在非應用程式伺服器機能（非 *ASF*）模式中執行的接聽器埠

在非 ASF 模式中執行的接聽器埠可以配置為使用伺服器階段作業監視佇列目的地及主題目的地。 伺服器階段作業可以具有多個交談，在每個案例中可以計算交談數上限。

接聽器埠可以配置為在非 ASF 模式中執行，這會變更接聽器埠監視 JMS 目的地的方式。 在作業的非 ASF 模式中使用接聽器埠的訊息驅動 Bean 應用程式，會為用於執行訊息驅動 Bean 實例以處理訊息的每個伺服器階段作業建立交談。 接聽器埠內容 **maximum sessions** 為給定的接聽器埠指定任何一次可以處於作用中狀態的伺服器階段作業數上限。 此內容的預設值是 10。

在非 ASF 模式中執行時，監視佇列目的地的接聽器埠將會自動建立接聽器埠內容 **Maximum sessions** 指定的伺服器階段作業數。 所有這些伺服器階段作業都利用從階段作業儲存區（與接聽器埠使用的 JMS 連線相關聯）取得的 JMS 階段作業，並不斷監視 JMS 目的地是否有適合訊息。

如果接聽器埠配置為監視主題目的地，則會忽略 **Maximum sessions** 的值，並使用單一伺服器階段作業。

在非 ASF 模式中執行的接聽器埠使用的伺服器階段作業在接聽器埠停止之前會一直保留作用中狀態，在此時使用的 JMS 階段作業會回到接聽器埠所使用 JMS 連線的階段作業儲存區的可用儲存區。

如需如何使用階段作業儲存區及如何管理 WebSphere Application Server 與 IBM MQ 之間的交談的相關資訊，請參閱 第 418 頁的『JMS Connection Factory』。

如需 WebSphere Application Server 作業的 ASF 及非 ASF 模式的相關資訊，以及如何將接聽器埠配置為使用非 ASF 模式，請參閱 ASF 模式及非 ASF 模式中的訊息處理。

監視佇列目的地時計算交談數上限

您可以使用下列公式計算從單一接聽器埠（在非 ASF 模式中執行並監視佇列目的地）到 IBM MQ 的交談數上限：

```
Maximum number of conversations = Maximum sessions
```

您可以使用下列計算，來找到允許進行此數目的交談而建立的通道實例數：

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

您可以對此計算中的任何餘數四捨五入。

對於使用 **Maximum sessions** 內容預設值及監視佇列目的地的簡式接聽器埠（在非 ASF 模式中執行），此接聽器埠的交談（可能存在於 WebSphere Application Server 與 IBM MQ 之間）數上限是：

```
Maximum number of conversations = Maximum sessions
```

例如：

```
     = 10
```

如果此接聽器埠使用 **SHARECNV** 內容設為 10 的通道連接至 IBM MQ，則所建立的通道實例數計算為:

```
Maximum number of channel instances =
   Maximum number of conversations / SHARECNV for the channel being used
```

例如:

```
= 10 / 10
= 1
```

監視主題目的地時計算交談數上限

對於在非 ASF 模式中執行且配置為監視主題目的地的接聽器埠，從接聽器埠至 IBM MQ 的交談數為:

```
Maximum number of conversations = 1
```

您可以使用下列計算，來找到允許進行此數目的交談而建立的通道實例數:

```
Maximum number of channel instances =
   Maximum number of conversations / SHARECNV for the channel being used
```

您可以對此計算中的任何餘數四捨五入。

對於使用 **Maximum sessions** 內容預設值及監視主題目的地的簡式接聽器埠（在非 ASF 模式中執行），此接聽器埠的交談（可能存在於 WebSphere Application Server 與 IBM MQ 之間）數上限是:

```
Maximum number of conversations = Maximum sessions
```

例如:

```
= 10
```

如果此接聽器埠使用 **SHARECNV** 內容設為 10 的通道連接至 IBM MQ，則所建立的通道實例數計算為:

```
Maximum number of channel instances =
   Maximum number of conversations / SHARECNV for the channel being used
```

例如:

```
= 10 / 10
= 1
```

配置鑑別別名以保護 IBM MQ 的 WebSphere Application Server 連線安全

鑑別別名會對映至使用者名稱及密碼組合，該組合可用於保護 WebSphere Application Server 與 IBM MQ 的連線。 您可以使用鑑別別名來配置 Connection Factory。

將鑑別別名與企業應用程式搭配使用

當在 WebSphere Application Server 內執行的企業應用程式嘗試建立與 IBM MQ 的 JMS 連線時，應用程式會從應用程式伺服器的 Java Naming Directory Interface (JNDI) 儲存庫中查閱 IBM MQ 傳訊提供者 Connection Factory 定義。

當 IBM MQ 傳訊提供者 Connection Factory 定義位於應用程式伺服器的 JNDI 儲存庫內時，會呼叫下列其中一個方法:

- `ConnectionFactory.createConnection()`
- `ConnectionFactory.createConnection(String username, String password)`

如果已使用所定義的 J2C 鑑別別名來配置 Connection Factory，則在使用 Connection Factory 建立連線時，鑑別別名中的使用者名稱及密碼可能會向下傳送至 IBM MQ。

Connection Factory 及鑑別別名

IBM MQ 傳訊提供者 Connection Factory，包含如何連接至 IBM MQ 佇列管理程式的相關資訊。 在 WebSphere Application Server 內執行的企業應用程式，可以使用這些 Connection Factory 來建立 JMS 與 IBM MQ 的連線。

WebSphere Application Server 會將 Connection Factory 定義儲存在可以使用 JNDI 存取的儲存庫中。 建立 Connection Factory 時，將會為 Connection Factory 指定可在定義它的應用程式伺服器範圍（Cell、「節點」或「伺服器」範圍）內唯一識別它的 JNDI 名稱。

例如，在 WebSphere Application Server Cell 範圍定義的 IBM MQ 傳訊提供者 Connection Factory 包含如何使用 BINDINGS 傳輸來連接佇列管理程式 (myQM) 的相關資訊。 為了唯一識別此 Connection Factory，將為其指定 JNDI 名稱 `jms/myCF`。

Connection Factory 亦可配置為使用鑑別別名。 鑑別別名會對映至使用者名稱及密碼組合。 視 Connection Factory 的使用方式而定，在建立 JMS 連線時，鑑別別名中的使用者名稱和密碼可能會或不會向下傳送至 IBM MQ 。

重要: 在 IBM MQ 8.0 之前，預設 IBM MQ「物件權限管理程式 (OAM)」執行授權檢查僅僅是為了確保在建立連線時，向下傳送至 IBM MQ 的使用者名稱有權存取佇列管理程式。

沒有為驗證已指定的密碼而進行任何檢查。 為了執行鑑別檢查並驗證使用者 ID 及密碼相符，您需要撰寫 IBM MQ 通道安全結束程式。 如需如何執行此動作的詳細資料，請參閱 第 812 頁的『通道安全結束程式』。

從 IBM MQ 8.0 起，除了使用者名稱以外，佇列管理程式還會檢查密碼。

使用 Connection Factory

下列主題包含利用直接及間接查閱來使用 Connection Factory 的相關資訊：

- 第 425 頁的『透過直接查閱來使用 Connection Factory』
- 第 426 頁的『透過間接查閱來使用 Connection Factory』

使用 CLIENT 傳輸

配置為使用 CLIENT 傳輸的 Connection Factory，必須指定將要用來連接至佇列管理程式的 IBM MQ 伺服器連線通道 (SVRCONN)。

對於已將 Connection Factory 配置為要使用的通道，如果 IBM MQ 通道代理程式使用者 ID (MCAUSER) 內容保留空白，則可將 Connection Factory 與直接查閱或間接查閱搭配使用。

如果將 MCAUSER 內容設定為某個使用者 ID，則不論企業應用程式使用的是直接查閱還是間接查閱，在使用 Connection Factory 來建立與 IBM MQ 的連線時，皆會將此使用者 ID 向下傳送至 IBM MQ。

摘要表格

下列表格彙總在分別使用 BINDINGS 及 CLIENT 傳輸時，將會向下傳送至 IBM MQ 的使用者 ID：

表 *71: BINDINGS* 模式

配置	**應用程式呼叫 ConnectionFactory.createConnection()**	**應用程式呼叫 ConnectionFactory.createConnection(String username, String password)**
應用程式的部署描述子不包含 Connection Factory 的「資源參照」	應用程式伺服器程序的使用者 ID 會向下傳送至 IBM MQ。	傳入 `ConnectionFactory.createConnection(String username, String password)` 方法的使用者 ID 及密碼，會向下傳送至 IBM MQ。
應用程式的部署描述子包含 Connection Factory 的「資源參照」，且 **res-auth** 內容設定為 "Application"	應用程式伺服器程序的使用者 ID 會向下傳送至 IBM MQ。	傳入 `ConnectionFactory.createConnection(String username, String password)` 方法的使用者 ID 及密碼，會向下傳送至 IBM MQ。
應用程式的部署描述子包含 Connection Factory 的「資源參照」，且 **res-auth** 內容設定為 "Container"	在鑑別別名中為 Connection Factory 指定的使用者 ID 及密碼，會向下傳送至 IBM MQ。	在鑑別別名中為 Connection Factory 指定的使用者 ID 及密碼，會向下傳送至 IBM MQ。
應用程式的部署描述子包含 **res-auth** 內容設定為 "Container" 的 Connection Factory 之「資源參照」，且已使用鑑別別名來配置應用程式	在應用程式已配置為要使用的鑑別別名中指定的使用者 ID 及密碼，會向下傳送至 IBM MQ。	在應用程式已配置為要使用的鑑別別名中指定的使用者 ID 及密碼，會向下傳送至 IBM MQ。

表 *72: CLIENT* 模式

配置	**應用程式呼叫 ConnectionFactory.createConnection()**	**應用程式呼叫 ConnectionFactory.createConnection(String username, String password)**
應用程式的部署描述子不包含 Connection Factory 的「資源參照」，且 Connection Factory 配置成使用已取消設定 MCAUSER 內容的 IBM MQ 通道	應用程式伺服器程序的使用者 ID 會向下傳送至 IBM MQ。	傳入 `ConnectionFactory.createConnection(String username, String password)` 方法的使用者 ID 及密碼，會向下傳送至 IBM MQ。
應用程式的部署描述子不包含 Connection Factory 的「資源參照」，且 Connection Factory 配置成使用 MCAUSER 內容設為使用者 ID 的 IBM MQ 通道	由 Connection Factory 配置為要使用的 IBM MQ 通道上 MCAUSER 內容所指定的使用者 ID，會向下傳送至 IBM MQ。	由 Connection Factory 配置為要使用的 IBM MQ 通道上 MCAUSER 內容所指定的使用者 ID，會向下傳送至 IBM MQ。
應用程式的部署描述子包含 Connection Factory 的「資源參照」，其 **res-auth** 內容設為 應用程式，且 Connection Factory 配置為使用已取消設定 MCAUSER 內容的 IBM MQ 通道	應用程式伺服器程序的使用者 ID 會向下傳送至 IBM MQ。	傳入 `ConnectionFactory.createConnection(String username, String password)` 方法的使用者 ID 及密碼，會向下傳送至 IBM MQ。

表 72: *CLIENT* 模式 (繼續)		
配置	**應用程式呼叫 ConnectionFactory.createConnection()**	**應用程式呼叫 ConnectionFactory.createConnection(String username, String password)**
應用程式的部署描述子包含 **res-auth** 內容設為 應用程式 之 Connection Factory 的「資源參照」，且 Connection Factory 配置為使用 MCAUSER 內容設為使用者 ID 的 IBM MQ 通道	由 Connection Factory 配置為要使用的 IBM MQ 通道上 MCAUSER 內容所指定的使用者 ID，會向下傳送至 IBM MQ。	由 Connection Factory 配置為要使用的 IBM MQ 通道上 MCAUSER 內容所指定的使用者 ID，會向下傳送至 IBM MQ。
應用程式的部署描述子包含 Connection Factory 的「資源參照」，其 **res-auth** 內容設為「儲存器」，且 Connection Factory 配置為使用已取消設定 MCAUSER 內容的 IBM MQ 通道	在鑑別別名中為 Connection Factory 指定的使用者 ID 及密碼，會向下傳送至 IBM MQ。	在鑑別別名中為 Connection Factory 指定的使用者 ID 及密碼，會向下傳送至 IBM MQ。
應用程式的部署描述子包含 Connection Factory 的「資源參照」，其 **res-auth** 內容設為 "*Container*，且 Connection Factory 配置為使用 MCAUSER 內容設為使用者 ID 的 IBM MQ 通道	由 Connection Factory 配置為要使用的 IBM MQ 通道上 MCAUSER 內容所指定的使用者 ID，會向下傳送至 IBM MQ。	由 Connection Factory 配置為要使用的 IBM MQ 通道上 MCAUSER 內容所指定的使用者 ID，會向下傳送至 IBM MQ。
應用程式的部署描述子包含 Connection Factory 的「資源參照」，其 **res-auth** 內容設為「儲存器」，且應用程式已配置鑑別別名，且 Connection Factory 配置為使用已取消設定 MCAUSER 內容的 IBM MQ 通道	在應用程式已配置為要使用的鑑別別名中指定的使用者 ID 及密碼，會向下傳送至 IBM MQ。	在應用程式已配置為要使用的鑑別別名中指定的使用者 ID 及密碼，會向下傳送至 IBM MQ。
應用程式的部署描述子包含 Connection Factory 的「資源參照」，其 **res-auth** 內容設定為儲存器，且應用程式已配置鑑別別名，且 Connection Factory 配置為使用 MCAUSER 設定為使用者 ID 的 IBM MQ 通道	由 Connection Factory 配置為要使用的 IBM MQ 通道上 MCAUSER 內容所指定的使用者 ID，會向下傳送至 IBM MQ。	由 Connection Factory 配置為要使用的 IBM MQ 通道上 MCAUSER 內容所指定的使用者 ID，會向下傳送至 IBM MQ。

透過直接查閱來使用 *Connection Factory*

定義 IBM MQ 傳訊提供者 Connection Factory 之後，企業應用程式可以查閱 Connection Factory 定義，並使用它來建立指向 IBM MQ 佇列管理程式的 JMS 連線。 透過直接查閱即可執行此作業。

為了使用直接查閱，企業應用程式會進行下列方法呼叫，以連接至應用程式伺服器的 JNDI 儲存庫：

```
InitialContext ctx = new InitialContext();
```

一旦連接至 JNDI 儲存庫，企業應用程式即會如下所示，使用 Connection Factory 的 JNDI 名稱來識別 Connection Factory 定義：

```
ConnectionFactory cf = (ConnectionFactory) ctx.lookup("jms/myCF");
```

附註：

- 開發企業應用程式時，應用程式開發者需要知道所需 Connection Factory 的 JNDI 名稱。 因為 JNDI 名稱是寫在應用程式內，如果 JNDI 名稱變更，您便需要重新撰寫並重新部署應用程式。
- 以此方式使用 Connection Factory 定義時，在鑑別別名（Connection Factory 已配置為要使用該別名）中指定的使用者名稱及密碼，不會向下傳送至 IBM MQ。 這是為了防止未獲授權的應用程式識別 Connection Factory 並可使用它來連接至安全 IBM MQ 系統。

向下傳送至 IBM MQ 的使用者名稱及密碼，取決於用來從 Connection Factory 建立 JMS 連線的方法。

如果應用程式使用下列方法建立 JMS 連線：

```
ConnectionFactory.createConnection()
```

則會將預設使用者身分向下傳遞至 IBM MQ。 這是啟動在其中執行企業應用程式的應用程式伺服器之使用者名稱及密碼。

應用程式亦可透過呼叫下列方法來建立 JMS 連線：

```
ConnectionFactory.createConnection(String username, String password)
```

如果應用程式已對 Connection Factory 執行直接查閱，並在隨後呼叫此方法，則會將傳入 `createConnection()` 方法的使用者名稱及密碼，向下傳送至 IBM MQ。

重要: 在 IBM MQ 8.0 之前，IBM MQ 處理授權檢查僅僅是為了確保向下傳送的使用者名稱有權存取佇列管理程式。

沒有對密碼進行檢查。 為了執行鑑別檢查並驗證使用者名稱及密碼有效，則必須撰寫 IBM MQ 通道安全結束程式。 如需如何執行此動作的詳細資料，請參閱 第 812 頁的『通道安全結束程式』。

從 IBM MQ 8.0 起，除了使用者名稱以外，佇列管理程式還會檢查密碼。

透過間接查閱來使用 *Connection Factory*

撰寫企業應用程式時，如果 Connection Factory 的 JNDI 名稱不明，或是要利用不同的 JNDI 名稱（視要在其中安裝此應用程式的應用程式伺服器而定）在使用不同 Connection Factory 的不同應用程式伺服器上安裝此應用程式，則可使用資源參照來查閱 Connection Factory。 透過間接查閱即可執行此作業。

範例

企業應用程式不是使用 `jms/myCF` 直接查閱 Connection Factory，而是包含本端 JNDI 名稱為 `jms/myResourceReferenceCF` 的資源參照。

為了使用此 JNDI 名稱，應用程式會以如同執行直接查閱的相同方式，連接至應用程式伺服器的 JNDI 儲存庫：

```
InitialContext ctx = new InitialContext();
```

現在，應用程式不是直接識別 `jms/myCF`，而是識別資源參照的 JNDI 名稱：

```
ConnectionFactory cf = (ConnectionFactory) ctx.lookup("java:comp/env/jms/
myResourceReferenceCF");
```

您需要本端 JNDI 名稱的 `java:comp/env` 字首，以告知應用程式伺服器企業應用程式正在執行間接查閱。

部署應用程式時，使用者會將資源參照的 JNDI 名稱 `jms/myResourceReferenceCF`，對映至應用程式已經建立的 Connection Factory 的 JNDI 名稱：`jms/myCF`。

當應用程式執行時，它會利用應用程式伺服器所對映的本端 JNDI 名稱來查閱 JMS Connection Factory: `jms/myCF`。 然後，應用程式會使用此 Connection Factory 來建立與 IBM MQ 的連線。

鑑別別名及間接查閱

資源參照還容許定義其他內容，用於變更所提供的 Connection Factory 之行為。 資源參照的其中一個內容是 **res-auth**。 此內容的值指定在建立與 IBM MQ 的連線時 (如果已定義鑑別別名)，企業應用程式是否應使用資源參照所對映之 Connection Factory 的鑑別別名，或應用程式是否指定自己的使用者名稱和密碼。

此內容的預設值是 *Application*。 這表示在建立 JMS 連線時，向下傳送至佇列管理程式的使用者名稱及密碼，將由應用程式自己決定。 而不會使用資源參照所對映的 Connection Factory 之鑑別別名。

應用程式可以使用下列其中一種方法來建立 JMS 連線：

- `ConnectionFactory.createConnection()`
- `ConnectionFactory.createConnection(String username, String password)`

如果應用程式使用 `ConnectionFactory.createConnection()`，且 **res-auth** 設定為 *Application*，則會將預設使用者身分向下傳送至 IBM MQ。 這是啟動在其中執行企業應用程式的應用程式伺服器之使用者名稱及密碼。

如果應用程式使用 `ConnectionFactory.createConnection(String username, String password)`，且 **res-auth** 設為 應用程式，則傳入方法的使用者名稱及密碼會向下傳送至 IBM MQ。

若要使用在建立連線時資源參照所對映的 Connection Factory 上定義的鑑別別名，您需要將 **res-auth** 內容設定為 *Container* 值。 在應用程式建立 JMS 連線時，即使 `createConnection` 呼叫指定了使用者名稱及密碼，仍會使用鑑別別名詳細資料。

使用間接查閱時置換鑑別別名

如果應用程式使用 **res-auth** 內容設定為 *Container* 的資源參照，您可以置換在建立 JMS 連線時使用的鑑別別名。

若要置換鑑別別名，資源參照需要包含稱為 **authDataAlias** 的額外內容，該內容會對映至已在將要部署應用程式的應用程式伺服器環境中建立的現有鑑別別名。 您可以在使用 IBM 所提供的 Rational 工具所建立的任何資源參照上指定此內容。

透過使用此方法，在使用已經間接查閱的 JMS Connection Factory 時，您可以使用不同的鑑別別名。 如果指定的鑑別別名不存在，則可在安裝企業應用程式之後指定新的鑑別別名。 如需相關資訊，請參閱 WebSphere Application Server 產品說明文件中的 資源參照 。

WebSphere Application Server 8.5.5 的相關資訊

資源參照

WebSphere Application Server 8.0 的相關資訊

資源參照

WebSphere Application Server 7.0 的相關資訊

資源參照

使用 WebSphere Application Server 叢集時訊息驅動 Bean 的工作量平衡

使用在 WebSphere Application Server 7.0 及 WebSphere Application Server 8.0 叢集中部署，且配置為在 IBM MQ 傳訊提供者標準模式中執行的訊息驅動 Bean 應用程式時，其中一個叢集成員會處理大多數訊息。 您可以平衡叢集成員的工作量，以分散處理多個叢集成員之間的訊息。

IBM MQ 包含稱為 **Asynchronous consume** 的特性，可讓應用程式使用稱為 **MQCB** 及 **MQCTL** 的 API，以非同步方式使用來自佇列的訊息。

在 WebSphere Application Server 7.0 及 WebSphere Application Server 8.0 內執行的訊息驅動 Bean 應用程式（使用 IBM MQ 傳訊提供者標準模式）將自動使用此特性。 當應用程式啟動時，他們會呼叫 **MQCB**，在已配置要監視的 JMS 目的地上設定非同步消費者。 然後，呼叫 **MQCTL** API 以指出應用程式已準備好從 JMS 目的地接收訊息。

訊息驅動 Bean 應用程式已部署在 WebSphere Application Server 叢集中時，每個叢集成員將為訊息驅動 Bean 正在監視訊息的 JMS 目的地設定非同步消費者。 然後，當 JMS 目的地上有適當的訊息可供叢集成員處理時，管理 JMS 目的地的 IBM MQ 佇列管理程式會負責通知叢集成員。

當 WebSphere Application Server 連接至 IBM MQ 佇列管理程式時，到達 JMS 目的地的訊息將更平均地配送至已在該 JMS 目的地上登錄的所有非同步消費者。 對於在 WebSphere Application Server 7.0 及 WebSphere Application Server 8.0 叢集內部署的訊息驅動 Bean 應用程式，意味著這些訊息將更均勻地分散在叢集成員之間。

相關工作

配置 JMS **PROVIDERVERSION** 內容

使用 IBM MQ Headers 套件

IBM MQ Headers 套件提供一組 Helper 介面及類別，您可以用來操作訊息的 IBM MQ 標頭。 通常，您會使用 IBM MQ Headers 套件，因為您想要使用指令伺服器來執行管理服務 (使用「可程式化指令格式 (PCF)」訊息)。

關於這項作業

IBM MQ 標頭套件位於 `com.ibm.mq.headers` 和 `com.ibm.mq.headers.pcf` 套件中。 您可以將此機能用於 IBM MQ 提供用於 Java 應用程式的兩個替代 API:

- IBM MQ classes for Java (也稱為 IBM MQ 基本 Java)。
- IBM MQ classes for Java Message Service (IBM MQ classes for JMS，也稱為 IBM MQ JMS)。

IBM MQ Base Java 應用程式通常會操作 MQMessage 物件， Headers 支援類別可以直接與這些物件互動，因為它們原生瞭解 IBM MQ Base Java 介面。

在 IBM MQ JMS 中，訊息的有效負載通常是「字串」或位元組陣列物件，可使用 DataInput 和 DataOutput 串流來操作。 IBM MQ 標頭套件可用來與這些資料串流互動，且適用於操作 IBM MQ JMS 應用程式所傳送及接收的任何 MQ 訊息。

因此，雖然 IBM MQ Headers 套件包含 IBM MQ Base Java 套件的參照，它也適用於 IBM MQ JMS 應用程式，而且適用於 Java Platform, Enterprise Edition (Java EE) 環境。

您可以使用 IBM MQ Headers 套件的典型方法是以「可程式化指令格式 (PCF)」來操作管理訊息，例如，基於下列任何原因:

- 存取 IBM MQ 資源的詳細資料。
- 監視佇列的深度。
- 禁止存取佇列。

透過搭配使用 PCF 訊息與 IBM MQ JMS API ，可以從 Java EE 應用程式內執行這種以應用程式為中心的資源管理，而無需使用 IBM MQ 基本 Java API。

程序

- 若要使用 IBM MQ Headers 套件來操作 IBM MQ classes for Java 的訊息標頭，請參閱 第 428 頁的『與 IBM MQ classes for Java 搭配使用』。
- 若要使用 IBM MQ Headers 套件來操作 IBM MQ classes for JMS 的訊息標頭，請參閱 第 429 頁的『與 IBM MQ classes for JMS 搭配使用』。

與 IBM MQ classes for Java 搭配使用

IBM MQ classes for Java 應用程式通常會操作 MQMessage 物件，而且 Headers 支援類別可以直接與這些物件互動，因為它們原生瞭解 IBM MQ classes for Java 介面。

關於這項作業

IBM MQ 提供一些範例應用程式，示範如何搭配使用 IBM MQ Headers 套件與 IBM MQ Base Java API (IBM MQ classes for Java)。

範例顯示兩件事:

- 如何建立 PCF 訊息以執行管理動作及剖析回應訊息。

- 如何使用 IBM MQ classes for Java 傳送此 PCF 訊息。

視您使用的平台而定，這些範例會安裝在 IBM MQ 安裝的 `samples` 或 `tools` 目錄中的 `pcf` 目錄下 (請參閱 第 297 頁的『IBM MQ classes for Java 的安裝目錄』)。

程序

1. 建立 PCF 訊息以執行管理動作並剖析回應訊息。
2. 使用 IBM MQ classes for Java 傳送此 PCF 訊息。

相關概念

第 320 頁的『使用 IBM MQ classes for Java 處理 IBM MQ 訊息標頭』
提供的 Java 類別代表不同類型的訊息標頭。 也提供兩個 Helper 類別。

第 325 頁的『使用 IBM MQ classes for Java 處理 PCF 訊息』
提供 Java 類別以建立及剖析 PCF 結構化訊息，並協助傳送 PCF 要求及收集 PCF 回應。

與 IBM MQ classes for JMS 搭配使用

若要將 IBM MQ 標頭與 IBM MQ classes for JMS 搭配使用，您可以執行與 IBM MQ classes for Java 相同的重要步驟。 可以使用 IBM MQ Headers 套件及與 IBM MQ classes for Java 相同的範例程式碼，以完全相同的方式來建立 PCF 訊息及剖析回應。

關於這項作業

若要使用 IBM MQ API 來傳送 PCF 訊息，必須將訊息有效負載寫入 JMS Bytes Message 中，並使用標準 JMS API 來傳送。 唯一考量是訊息不得包含 JMS RFH2 或 MQMD 中具有特定值的任何其他標頭。

若要傳送 PCF 訊息，請完成下列步驟。 建立 PCF 訊息及從回應訊息擷取資訊的方式與 IBM MQ classes for Java 相同 (請參閱 第 428 頁的『與 IBM MQ classes for Java 搭配使用』)。

程序

1. 建立代表 SYSTEM.ADMIN.COMMAND.QUEUE 的 JMS 佇列目的地。

   IBM MQ JMS 應用程式會將 PCF 訊息傳送至 SYSTEM.ADMIN.COMMAND.QUEUE，且需要存取代表此佇列的 JMS 目的地物件。 「目的地」必須已設定下列內容:

   ```
   WMQ_MQMD_WRITE_ENABLED = YES
   WMQ_MESSAGE_BODY = MQ
   ```

   如果您使用 WebSphere Application Server，則必須將這些內容定義為目的地上的自訂內容。

   如果要從應用程式內以程式化方式建立目的地，請使用下列程式碼:

   ```
   Queue q1 = session.createQueue("SYSTEM.ADMIN.COMMAND.QUEUE");
   ((MQQueue) q1).setIntProperty(WMQConstants.WMQ_MESSAGE_BODY,
       WMQConstants.WMQ_MESSAGE_BODY_MQ);
   ((MQQueue) q1).setMQMDWriteEnabled(true);
   ```

2. 將 PCF 訊息轉換為包含正確 MQMD 值的 JMS 位元組訊息。

   需要建立 JMS 位元組訊息，並將 PCF 訊息寫入其中。 需要建立回應佇列，但這不需要具有特定設定。

   下列範例程式碼 Snippet 顯示如何建立 JMS Bytes Message ， 並將 com.ibm.mq.headers, pcf.PCFMessage 物件寫入其中。 先前已使用 IBM MQ Headers 套件建置 PCFMessage 物件 (pcfCmd)。 (請注意，載入 PCFMessage 的套件是 com.ibm.mq.headers.pcf.PCFMessage)。

   ```
    // create the JMS Bytes Message
   final BytesMessage msg = session.createBytesMessage();

   // Create the wrapping streams to put the bytes into the message payload
   ByteArrayOutputStream baos = new ByteArrayOutputStream();
   DataOutput dataOutput = new DataOutputStream(baos);

   // Set the JMSReplyTo so the answer comes back
   msg.setJMSReplyTo(new MQQueue("adminResp"));
   ```

```
// write the pcf into the stream
pcfCmd.write(dataOutput);
baos.flush();
msg.writeBytes(baos.toByteArray());

// we have taken control of the MD, so need to set all
// flags in the MD that we require - main one is the format
msg.setJMSPriority(4);
msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_PERSISTENCE,
  CMQC.MQPER_NOT_PERSISTENT);
msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_EXPIRY, 300);
msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_REPORT,
    CMQC.MQRO_PASS_CORREL_ID);
msg.setStringProperty(WMQConstants.JMS_IBM_MQMD_FORMAT, "MQADMIN");

// and send the message
sender.send(msg);
```

3. 傳送訊息，並使用標準 JMS API 接收回應。
4. 將回應訊息轉換成 PCF 訊息以進行處理。

   若要擷取回應訊息並將其作為 PCF 訊息處理，請使用下列程式碼:

```
 // Get the message back
BytesMessage msg = (BytesMessage) consumer.receive();

// get the size of the bytes message & read into an array
int bodySize = (int) msg.getBodyLength();
byte[] data = new byte[bodySize];
msg.readBytes(data);

// Read into Stream and DataInput Stream
ByteArrayInputStream bais = new ByteArrayInputStream(data);
DataInput dataInput = new DataInputStream(bais);

// Pass to PCF Message to process
PCFMessage response = new PCFMessage(dataInput);
```

相關概念

第 122 頁的『JMS 訊息』

JMS 訊息由標頭、內容及內文組成。 JMS 定義五種類型的訊息內文。

IBM i 在 IBM i 上使用 Java 和 JMS 來設定 IBM MQ

此主題集合提供如何使用 CL 指令或 qshell 環境，在 IBM i 上使用 Java 及 JMS 來設定及測試 IBM MQ 的概觀。

註:

- 從 IBM MQ 8.0、 `ldap.jar`、 `jndi.jar` 和 `jta.jar` 開始，都是 JDK 的一部分。
- JM 3.0 V 9.3.0 V 9.3.0 從 IBM MQ 9.3.0 開始，支援 Jakarta Messaging 3.0 開發新的應用程式。 IBM MQ 9.3.0 繼續支援 JMS 2.0 現有的應用程式。 不支援在同一應用程式中同時使用 Jakarta Messaging 3.0 API 和 JMS 2.0 API。 如需相關資訊，請參閱 使用 IBM MQ 類別進行 JMS/Jakarta 傳訊。

使用 CL 指令

您所設定的 CLASSPATH 是用來測試 MQ 基本 Java、JMS (含 JNDI) 及 JMS (不含 JNDI)。

如果您不使用 `/home/Userprofile` 目錄下的 `.profile` 檔案，則需要在下面設定系統層次環境變數。 您可以檢查是否使用 **WRKENVVAR** 指令來設定它們。

1. 若要檢視整個系統的環境變數，請發出下列指令: **WRKENVVAR LEVEL (*SYS)**
2. 若要檢視工作特定的環境變數，請發出下列指令: **WRKENVVAR LEVEL (*JOB)**
3. 如果未設定 CLASSPATH ，請發出下列指令:

JM 3.0

```
ADDENVVAR ENVVAR(CLASSPATH)
       VALUE('.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar
```

```
       :/QIBM/ProdData/mqm/java/lib/connector.jar:/QIBM/ProdData/mqm/java/lib
       :/QIBM/ProdData/mqm/java/samples/base
       :/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar
       :/QIBM/ProdData/mqm/java/lib/jms.jar
       :/QIBM/ProdData/mqm/java/lib/providerutil.jar
       :/QIBM/ProdData/mqm/java/lib/fscontext.jar:') LEVEL(*SYS)
```

JMS 2.0

```
ADDENVVAR ENVVAR(CLASSPATH)
       VALUE('.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar
       :/QIBM/ProdData/mqm/java/lib/connector.jar:/QIBM/ProdData/mqm/java/lib
       :/QIBM/ProdData/mqm/java/samples/base
       :/QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar
       :/QIBM/ProdData/mqm/java/lib/jms.jar
       :/QIBM/ProdData/mqm/java/lib/providerutil.jar
       :/QIBM/ProdData/mqm/java/lib/fscontext.jar:') LEVEL(*SYS)
```

4. 如果未設定 QIBM_MULTI_THREADING，請發出下列指令:

```
ADDENVVAR ENVVAR(QIBM_MULTI_THREADED) VALUE('Y') LEVEL(*SYS)
```

5. 如果未設定 QIBM_USE_DESCRIPTOR_STDIO，請發出下列指令:

```
ADDENVVAR ENVVAR(QIBM_USE_DESCRIPTOR_STDIO) VALUE('I') LEVEL(*SYS)
```

6. 如果未設定 QSH_REDIRECTION_TEXTDATA，請發出下列指令:

```
ADDENVVAR ENVVAR(QSH_REDIRECTION_TEXTDATA) VALUE('Y') LEVEL(*SYS)
```

使用 qshell 環境

如果您使用 QSHELL 環境，則可以在 `/home/Userprofile` 目錄中設定 `.profile`。 如需相關資訊，請參閱 Qshell 直譯器 (qsh) 文件。

在 `.profile` 中指定下列。 請注意， CLASSPATH 陳述式必須在單行上，或使用 \ 字元分隔至不同行，如下所示。

JM 3.0

```
CLASSPATH=.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: \
/QIBM/ProdData/mqm/java/lib/connector.jar: \
/QIBM/ProdData/mqm/java/lib: \
/QIBM/ProdData/mqm/java/samples/base: \
/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar: \
/QIBM/ProdData/mqm/java/lib/jms.jar: \
/QIBM/ProdData/mqm/java/lib/providerutil.jar: \
/QIBM/ProdData/mqm/java/lib/fscontext.jar:
HOME=/home/XXXXX
LOGNAME=XXXXX
PATH=/usr/bin:
QIBM_MULTI_THREADED=Y QIBM_USE_DESCRIPTOR_STDIO=I
QSH_REDIRECTION_TEXTDATA=Y
TERMINAL_TYPE=5250
```

JMS 2.0

```
CLASSPATH=.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: \
/QIBM/ProdData/mqm/java/lib/connector.jar: \
/QIBM/ProdData/mqm/java/lib: \
/QIBM/ProdData/mqm/java/samples/base: \
/QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar: \
/QIBM/ProdData/mqm/java/lib/jms.jar: \
/QIBM/ProdData/mqm/java/lib/providerutil.jar: \
/QIBM/ProdData/mqm/java/lib/fscontext.jar:
HOME=/home/XXXXX
LOGNAME=XXXXX
PATH=/usr/bin:
QIBM_MULTI_THREADED=Y QIBM_USE_DESCRIPTOR_STDIO=I
```

```
QSH_REDIRECTION_TEXTDATA=Y
TERMINAL_TYPE=5250
```

發出指令 **DSPLIBL**，確定 QMQMJAVA 程式庫位於程式庫清單中。

如果 QMQMJAVA 程式庫不在清單中，請使用下列指令來新增它: **ADDLIBLE LIB (QMQMJAVA)**

IBM i 使用 Java 在 IBM i 上測試 IBM MQ

如何使用 MQIVP 範例程式來使用 Java 測試 IBM MQ 。

測試 IBM MQ 基本 Java

執行下列程序:

1. 發出下列指令，驗證佇列管理程式是否已啟動，以及佇列管理程式的狀態是否為 ACTIVE:

   ```
   WRKMQM MQMNAME(QMGRNAME)
   ```

2. 驗證 JAVA.CHANNEL 伺服器連線通道:

   ```
   WRKMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
   ```

   a. 如果 JAVA.CHANNEL 不存在，請發出下列指令:

   ```
   CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
   ```

3. 透過發出 **WRKMQMLSR** 指令，驗證佇列管理程式接聽器是否正在執行埠 1414 或您正在使用的埠。

   a. 如果未啟動佇列管理程式的接聽器，請發出下列指令:

   ```
   STRMQMLSR PORT(xxxx) MQMNAME(QMGRNAME)
   ```

執行 MQIVP 範例測試程式

1. 從指令行發出指令 STRQSH 來啟動 qshell
2. 發出 **export** 指令，驗證已設定正確的 CLASSPATH ，然後發出 **cd** 指令，如下所示:

   ```
   cd /qibm/proddata/mqm/java/samples/wmqjava/samples
   ```

3. 發出下列指令來執行 **java** 程式:

   ```
   java MQIVP
   ```

當提示您輸入下列指令時，您可以按 ENTER 鍵:

- 連線類型
- IP 位址
- 佇列管理程式名稱

以使用預設值。 這會驗證產品連結 (可在 QMQMJAVA 程式庫中找到)。

您會收到類似下列範例的輸出。 請注意，版權聲明視您使用的產品版本而定。

```
> java MQIVP
MQSeries for Java Installation Verification Program
5724-H72 (C) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
============================================================

Please enter the IP address of the MQ server :>
Please enter the queue manager name :>
Attaching Java program to QIBM/ProdData/mqm/java/lib/connector.JAR.
```

```
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 MQ Transport is functioning correctly.
Press Enter to continue ...>
$
```

測試 IBM MQ Java 用戶端連線

您必須指定:

- 連線類型
- IP 位址
- 埠
- 伺服器連線通道
- 佇列管理程式

您會收到類似下列範例的輸出。 請注意，版權聲明視您使用的產品版本而定。

```
> java MQIVP
MQSeries for Java Installation Verification Program
5724-H72 (C) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
============================================================

Please enter the IP address of the MQ server :> x.xx.xx.xx
Please enter the port to connect to : (1414)> 1470
Please enter the server connection channel name :> JAVA.CHANNEL
Please enter the queue manager name :> KAREN01
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 MQ Transport is functioning correctly.
Press Enter to continue ...>
$
```

IBM i 使用 JMS 在 IBM i 上測試 IBM MQ

如何使用含及不含 JNDI 的 JMS 來測試 IBM MQ

使用 IVTRun 範例在沒有 JNDI 的情況下測試 JMS

執行下列程序:

1. 發出下列指令，驗證佇列管理程式是否已啟動，以及佇列管理程式的狀態是否為 ACTIVE:

   ```
   WRKMQM MQMNAME(QMGRNAME)
   ```

2. 從指令行發出 **STRQSH** 指令來啟動 qshell。
3. 使用 **cd** 指令來切換目錄，如下所示:

   ```
   cd /qibm/proddata/mqm/java/bin
   ```

4. 執行 Script 檔:

```
IVTRun -nojndi [-m qmgrname]
```

您會收到類似下列範例的輸出。 請注意，著作權聲明視您使用的產品版本而定:

```
 IVTRun -nojndi -m ELCRTP19

Attaching Java program to
/QIBM/ProdData/mqm/java/lib/com.ibm.mqjms.JAR.
Attaching Java program to
/QIBM/ProdData/mqm/java/lib/jms.JAR.

5724-H72, 5724-B41, 5655-F10 (c) Copyright IBM Corp. 2011, 2024.
All Rights Reserved.
WebSphere MQ classes for Java Message Service 5.300
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:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000c012
JMSTimestamp: 1020273404500
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_PutDate:20040326
JMSXAppID:QP0ZSPWT STANLEY 170302
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:8
JMS_IBM_MsgType:8
JMSXUserID:STANLEY
JMS_IBM_PutTime:13441354
JMSXDeliveryCount:1
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
$>
$
```

在沒有 JNDI 的情況下測試 IBM MQ JMS 用戶端模式

執行下列程序:

1. 發出下列指令，驗證佇列管理程式是否已啟動，以及佇列管理程式的狀態是否為 ACTIVE:

   ```
   WRKMQM MQMNAME(QMGRNAME)
   ```

2. 發出下列指令，驗證已建立伺服器連線通道:

   ```
   WRKMQMCHL CHLNAME( SYSTEM.DEF.SVRCONN ) CHLTYPE(*SVRCN)
   MQMNAME(QMGRNAME)
   ```

3. 透過發出 **WRKMQMLSR** 指令，驗證已針對正確埠啟動接聽器

4. 從指令行發出 **STRQSH** 指令來啟動 qshell。
5. 透過發出 **export** 指令，驗證 CLASSPATH 是否正確。
6. 使用 **cd** 指令來切換目錄，如下所示:

```
cd /qibm/proddata/mqm/java/bin
```

7. 執行 Script 檔:

```
IVTRun -nojndi -client -m QMgrName -host hostname [-port port] [-channel channel]
```

您會收到類似下列範例的輸出。 請注意，著作權聲明視您使用的產品版本而定。

```
> IVTRun -nojndi -client -m ELCRTP19 -host ELCRTP19 -port 1414 -channel SYSTEM.DEF.SVRCONN

5724-H72, 5724-B41, 5655-F10 (c) Copyright IBM Corp. 2011, 2024.
All Rights Reserved.
WebSphere MQ classes for Java Message Service 5.300
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:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000d012
JMSTimestamp: 1020274009970
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_PutDate:20040326
JMSXAppID:MQSeries Client for Java
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:28
JMS_IBM_MsgType:8
JMSXUserID:QMQM
JMS_IBM_PutTime:14085237
JMSXDeliveryCount:1
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 測試 IBM MQ JMS

發出下列指令，驗證佇列管理程式是否已啟動，以及佇列管理程式的狀態是否為 ACTIVE:

```
WRKMQM MQMNAME(QMGRNAME)
```

使用 IVTRun 範例測試 Script

執行下列程序:

1. 對 JMSAdmin.config 檔案進行適當的變更。 若要編輯此檔案，請從 IBM i 指令行使用 **EDTF** (編輯檔案) 指令

```
EDTF '/qibm/proddata/mqm/java/bin/JMSAdmin.config'
```

.

   a. 若要使用 LDAP for Weblogic ， 請從中移除註解:

```
INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
```

   b. 若要使用 LDAP for WebSphere Application Server，請從下列中移除註解:

```
INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
```

   c. 若要測試檔案系統，請從中移除註解:

```
INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
```

   d. 請從適當的行中移除註解，以確定您已選取正確的 PROVIDER_URL。

   e. 使用 # 符號註銷所有其他行。

   f. 完成所有變更之後，請按 **F2=Save** 及 **F3=Exit**。

2. 從指令行發出 **STRQSH** 指令來啟動 qshell。
3. 透過發出 **export** 指令，驗證 CLASSPATH 是否正確。
4. 使用 **cd** 指令來切換目錄，如下所示:

```
cd /qibm/proddata/mqm/java/bin
```

5. 透過發出 **IVTSetup** 指令，啟動 **IVTSetup** Script 以建立受管理物件 (*MQQueueConnectionFactory* 及 *MQQueue*)。
6. 發出下列指令來執行 IVTRun Script:

```
IVTRun -url providerURL [-icf initCtxFact]
```

您會收到類似下列範例的輸出。 請注意，著作權聲明視您使用的產品版本而定。

```
> IVTSetup
+ Creating script for object creation within JMSAdmin
+ Calling JMSAdmin in batch mode to create objects
Ignoring unknown flag: -i

5724-H72 (c) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
Starting WebSphere MQ classes for Java Message Service Administration

InitCtx>
InitCtx>
InitCtx>
InitCtx>
InitCtx>
Stopping MQSeries classes for Java Message Service Administration

+ Administration done; tidying up files
+ Done!
$
> IVTRun -url file:////tmp/mqjms -icf com.sun.jndi.fscontext.RefFSContextFactory

5724-H72 (c) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
MQSeries classes for Java 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
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000e012
JMSTimestamp: 1020274903770
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:8
JMSXDeliveryCount:1
JMS_IBM_MsgType:8
JMSXUserID:STANLEY
JMSXAppID:QP0ZSPWT STANLEY 170308
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 Maven 儲存庫進行 Java 應用程式開發

開發 IBM MQ 的 Java 應用程式時，透過使用 Maven 儲存庫自動安裝相依關係，您不需要在使用 IBM MQ 介面之前明確安裝任何項目。

Maven Central 儲存庫

Maven 是建置應用程式的工具，也提供儲存庫來保留您應用程式可能想要存取的構件。

Maven 儲存庫 (或「中央儲存庫」) 具有一種結構，可讓檔案 (例如 JAR 檔) 具有不同的版本，然後使用已知的命名機制輕鬆探索這些版本。 然後建置工具可以使用這些名稱來動態取回應用程式的相依關係。 在應用程式的定義中 (當使用 Maven 作為建置工具時，稱為 POM 檔案) ， 您可以命名相依關係，建置程序會知道從該處執行的動作。

IBM MQ 用戶端檔案

IBM MQ Java 用戶端介面的副本可在 `com.ibm.mq GroupId` 下的「中央儲存庫」中找到。 您可以找到 `com.ibm.mq.jakarta.client.jar` 檔案 (Jakarta Messaging 3.0) 和 `com.ibm.mq.allclient.jar` 檔案 (JMS 2.0)。 這些檔案通常用於獨立式程式。 您也可以找到 `wmq.jakarta.jmsra.rar` 檔 (Jakarta Messaging 3.0) 和 `wmq.jmsra.rar` 檔 (JMS 2.0) ， 供 Java EE 應用程式伺服器使用。 `jakarta.client.jar` 和 `allclient.jar` 都包含 IBM MQ classes for JMS 和 IBM MQ classes for Java。

重要: 不支援使用 Apache Maven 組合外掛程式 *jar-with-dependencies* 格式來建置包含 IBM MQ 可再定位 JAR 檔的應用程式。

在 maven 指令所處理的 `pom.xml` 檔案中，您可以新增這些 JAR 檔的相依關係，如下列範例所示:

- JM 3.0 V 9.3.0 V 9.3.0 若要顯示應用程式碼與 `com.ibm.mq.jakarta.client.jar` 之間的關係，請執行下列動作:

```
<dependency>
        <groupId>com.ibm.mq</groupId>
```

```
        <artifactId>com.ibm.mq.jakarta.client</artifactId>
        <version>9.3.0.0</version>
    </dependency>
```

- JMS 2.0 若要顯示應用程式碼與 `com.ibm.mq.allclient.jar` 之間的關係，請執行下列動作:

```
<dependency>
        <groupId>com.ibm.mq</groupId>
        <artifactId>com.ibm.mq.allclient</artifactId>
        <version>9.2.2.0</version>
    </dependency>
```

- JM 3.0 V 9.3.0 V 9.3.0 如果要使用 Jakarta EE 資源配接器:

```
<dependency>
     <groupId>com.ibm.mq</groupId>
     <artifactId>wmq.jakarta.jmsra</artifactId>
     <version>9.3.0.0</version>
 </dependency>
```

- JMS 2.0 如果要使用 JMS 2.0 Java EE 資源配接器:

```
<dependency>
     <groupId>com.ibm.mq</groupId>
     <artifactId>wmq.jmsra</artifactId>
     <version>9.2.2.0</version>
 </dependency>
```

如需 Eclipse 中執行 JMS 專案的簡式專案範例，請參閱 IBM Developer 文章 使用 Maven 來開發 MQ 的 Java 應用程式變得更容易。

開發 C++ 應用程式

IBM MQ 提供相當於 IBM MQ 物件的 C++ 類別，以及相當於陣列資料類型的其他類別。 它提供許多無法透過 MQI 使用的特性。

IBM WebSphere MQ 7.0， IBM MQ 程式設計介面的加強功能未套用至 C++ 類別。

IBM MQ C++ 提供下列特性:

- 自動起始設定 IBM MQ 資料結構。
- 即時佇列管理程式連線及佇列開啟。
- 隱含的佇列關閉及佇列管理程式斷線。
- 無法傳送郵件的標頭傳輸及回執。
- IMS 橋接器標頭傳輸及回執。
- 參照訊息標頭傳輸及回執。
- 觸發訊息回執。
- CICS bridge 標頭傳輸和回執。
- 工作標頭傳輸及回執。
- 用戶端通道定義。

下列 Booch 類別圖顯示所有類別大致平行於程序化 MQI (例如使用 C) 中具有控點或資料結構的那些 IBM MQ 實體。 所有類別都繼承自 ImqError 類別 (請參閱 ImqError C++ 類別)， 它容許錯誤條件與每一個物件相關聯。

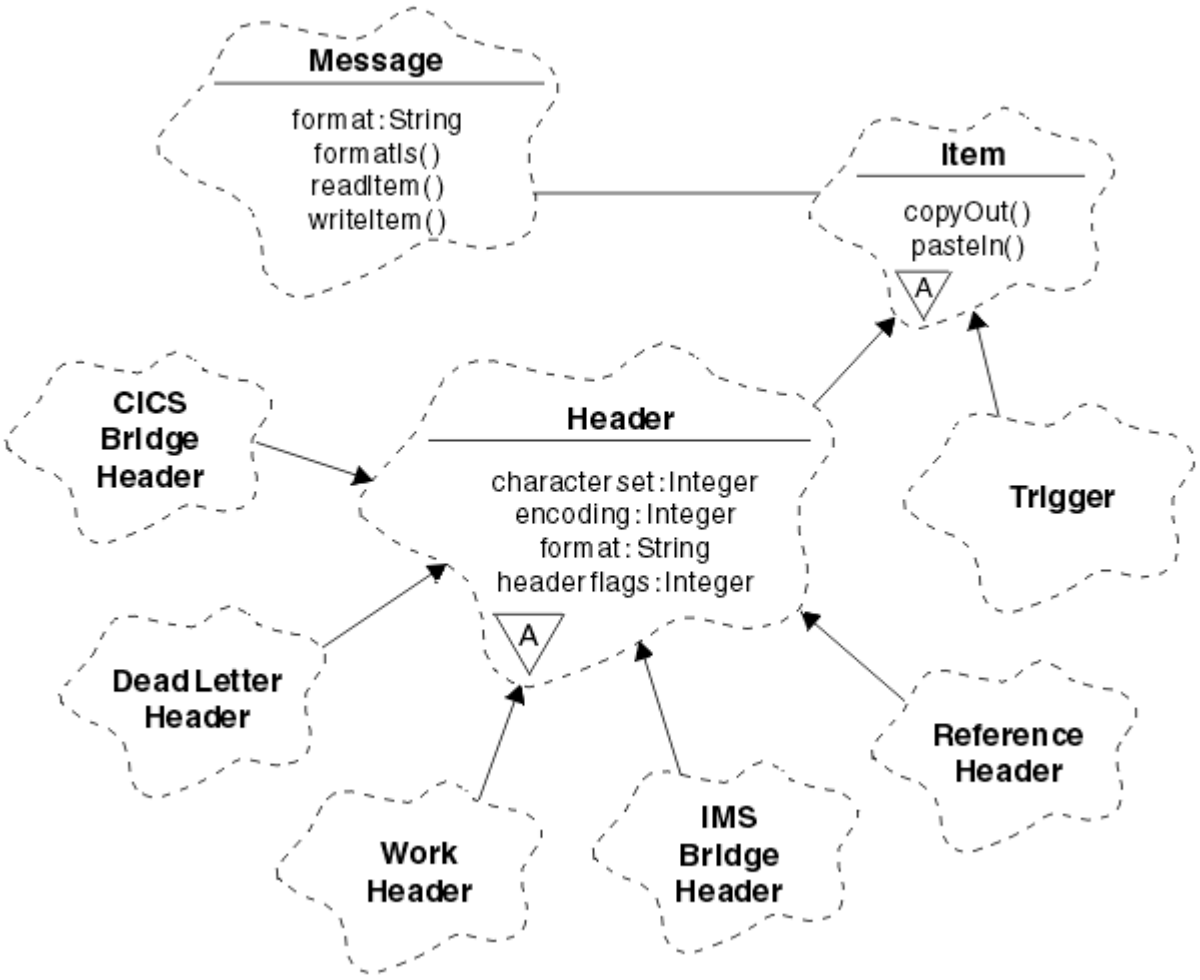


圖 *50: IBM MQ C++* 類別 *(*項目處理*)*

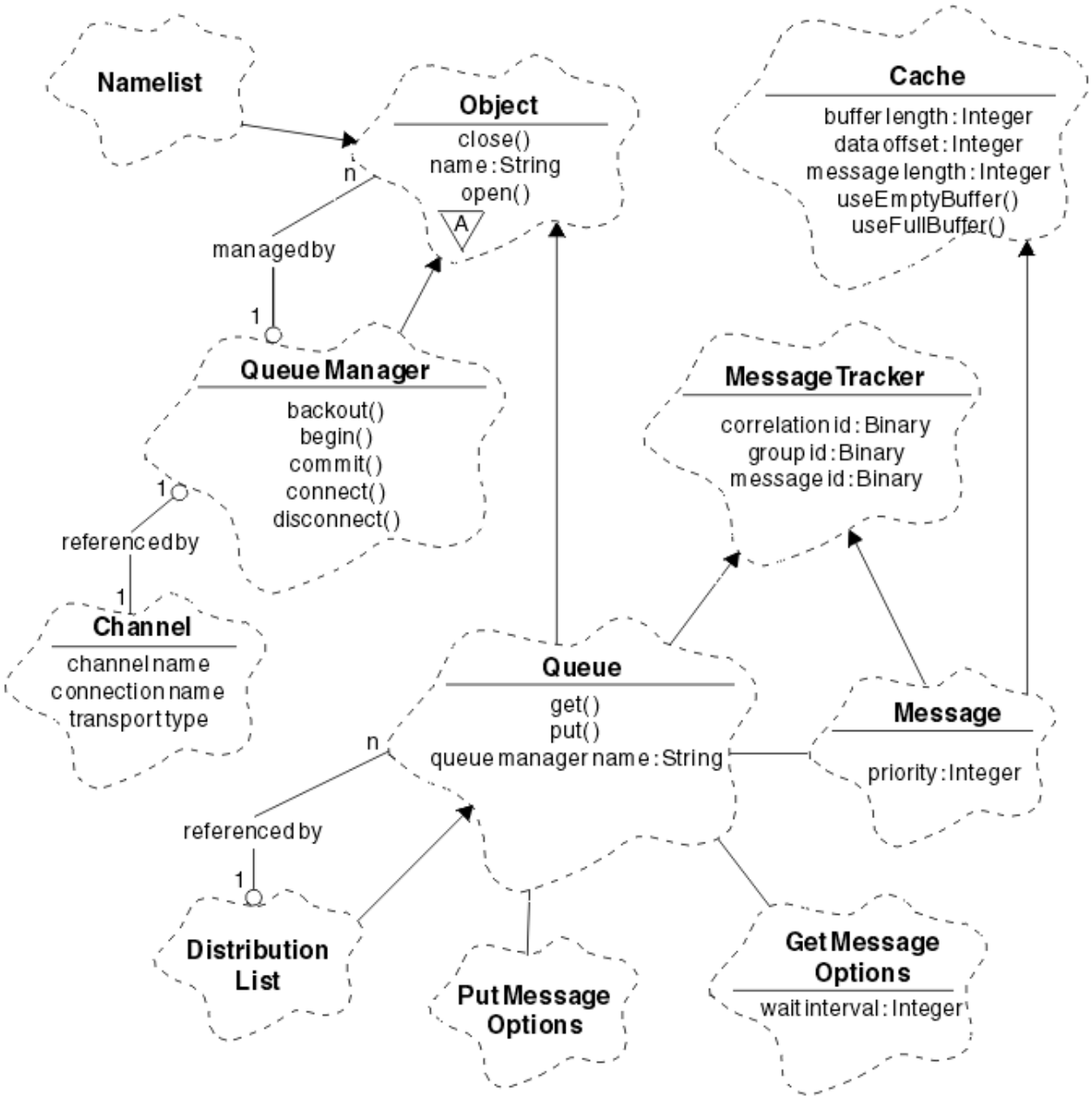


圖 *51: IBM MQ C++* 類別 *(佇列管理)*

若要正確解譯 Booch 類別圖，請注意下列使用慣例:

- 方法和特殊注意事項屬性顯示在 類別 名稱下方。
- 雲端內的小三角形表示 抽象類別。
- 繼承 以母類別的箭頭表示。
- 雲端之間未裝飾的線條表示類別之間的 合作關係 。
- 以數字裝飾的線條表示兩個類別之間的 參照關係 。 此數字指出在任何一次可以參與特定關係的物件數。

在佇列管理類別的 C++ 方法簽章中使用下列類別及資料類型 (請參閱 第 440 頁的圖 51) 及項目處理類別 (請參閱 第 439 頁的圖 50):

- ImqBinary 類別 (請參閱 ImqBinary C++ 類別)，其封裝位元組陣列，例如 MQBYTE24。
- ImqBoolean 資料類型，定義為 **typedef unsigned char ImqBoolean**。
- ImqString 類別 (請參閱 ImqString C++ 類別)，它會封裝字元陣列，例如 MQCHAR64。

具有資料結構的實體會納入適當的物件類別中。 個別資料結構欄位 (請參閱 C++ 及 MQI 交互參照) 使用方法來存取。

具有控點的實體位於 ImqObject 類別階層下 (請參閱 ImqObject C++ 類別) 並提供 MQI 的封裝介面。 這些類別的物件會呈現智慧型行為，可減少相對於程序化 MQI 所需的方法呼叫數目。 例如，您可以視需要建立並捨棄佇列管理程式連線，也可以使用適當的選項開啟佇列，然後關閉它。

ImqMessage 類別 (請參閱 ImqMessage C++ 類別) 封裝 MQMD 資料結構，同時充當使用者資料和 項目 的保留點 (請參閱 第 449 頁的『在 C++ 中讀取訊息』) 提供快取的緩衝區機能。 您可以為使用者資料提供固定長度緩衝區，並多次使用該緩衝區。 緩衝區中呈現的資料量可能因不同用途而有所不同。 或者，系統可以提供並管理彈性長度的緩衝區。 緩衝區大小 (可用於接收訊息的數量) 和實際使用的數量 (傳輸的位元組數或實際接收的位元組數) 都成為重要考量。

相關概念

技術概觀

第 441 頁的『C++ 範例程式』
提供四個範例程式，以示範取得及放置訊息。

第 445 頁的『C++ 語言考量』
此主題集合詳細說明在撰寫使用「訊息佇列介面 (MQI)」的應用程式時，您必須考量的 C++ 語言使用情形及使用慣例的各個層面。

第 448 頁的『在 C++ 中準備訊息資料』
訊息資料準備在緩衝區中，可由系統或應用程式提供。 任何一種方法都有優點。 提供使用緩衝區的範例。

第 5 頁的『開發適用於 IBM MQ 的應用程式』
您可以開發應用程式來傳送及接收訊息，以及管理佇列管理程式和相關資源。 IBM MQ 支援以許多不同語言及架構撰寫的應用程式。

相關參考

第 454 頁的『建置 IBM MQ C++ 程式』
會列出受支援編譯器的 URL，以及用來在 IBM MQ 平台上編譯、鏈結及執行 C++ 程式和範例的指令。

C++ 及 MQI 交互參照

IBM MQ C++ 類別

C++ 範例程式

提供四個範例程式，以示範取得及放置訊息。

範例程式如下:

- Hello World (imqwrld.cpp)
- SPUT (imqsput.cpp)
- SGET (imqsget.cpp)
- DPUT (imqdput.cpp)

範例程式位於 第 441 頁的表 73 中顯示的目錄。

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

表 *73:* 範例程式的位置

環境	包含來源的目錄	包含建置的目錄 程式集
AIX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/ia
AIX V 9.3.5 AIX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/ca (請參閱附註 第 442 頁的『1』)

表 73: 範例程式的位置 (繼續)		
環境	**包含來源的目錄**	**包含建置的目錄 程式集**
IBM i IBM i	/QIBM/ProdData/mqm/samp/	(請參閱附註 第 442 頁的『2』)
Linux Linux	*MQ_INSTALLATION_PATH*/samp	無
Windows Windows	*MQ_INSTALLATION_PATH*\tools\cplus\samples	*MQ_INSTALLATION_PATH*\tools\cplus\範例 \bin\vn (請參閱附註 第 442 頁的『3』)
z/OS z/OS	thlqual.SCSQCPPS	

附註:

1. AIX V 9.3.5 使用 XLC 17 編譯器建置的程式位於 "ca" 資料夾中，而使用 XLC 16 編譯器建置的程式位於 "ia" 資料夾中。
2. IBM i 使用 ILE C++ 編譯器針對 IBM i 建置的程式位於檔案庫 QMQM 中。 原始檔位於 /QIBM/ProdData/mqm/samp 中。
3. Windows 使用 Microsoft Visual Studio Visual Studio 建置的程式可在 *MQ_INSTALLATION_PATH*\tools\cplus\samples\bin\vn 中找到。 如需這些編譯器的進一步相關資訊，請參閱 第 460 頁的『在 Windows 上建置 C++ 程式』。

範例程式 HELLO WORLD (imqwrld.cpp)

此 C++ 範例程式顯示如何使用 ImqMessage 類別放置及取得一般資料包 (C 結構)。

此程式顯示如何使用 ImqMessage 類別來放置及取得一般資料封包 (C 結構)。 此範例使用少數方法呼叫，並利用隱含的方法呼叫，例如 **open**、 **close** 及 **disconnect**。

在 z/OS 以外的所有平台上

如果您使用與 IBM MQ 的伺服器連線，請遵循下列其中一個程序:

- 若要使用現有的預設佇列，請使用 SYSTEM.DEFAULT.LOCAL.QUEUE，執行程式 **imqwrlds** 而不傳遞任何參數
- 若要使用暫時動態指派的佇列，請執行 **imqwrlds** ，傳遞預設模型佇列 SYSTEM.DEFAULT.MODEL.QUEUE。

如果您是使用 IBM MQ 的用戶端連線，請遵循下列其中一個程序:

- 設定 MQSERVER 環境變數 (如需相關資訊，請參閱 *MQSERVER*) 並執行 **imqwrldc**，或
- 執行 **imqwrldc** 作為參數傳遞 **queue-name**、 **queue-manager-name** 及 **channel-definition**，其中一般 **channel-definition** 可能是 SYSTEM.DEF.SVRCONN/TCP/*hostname* (1414)

開啟 z/OS

z/OS

使用範例 JCL **imqwrldr** 來建構並執行批次工作。

如需相關資訊，請參閱 z/OS 批次、RRS 批次和 CICS 。

範例程式碼

```
extern "C" {
#include <stdio.h>
}

#include <imqi.hpp> // IBM MQ C++

#define EXISTING_QUEUE "SYSTEM.DEFAULT.LOCAL.QUEUE"

#define BUFFER_SIZE 12

static char gpszHello[ BUFFER_SIZE ] = "Hello world" ;
int main ( int argc, char * * argv ) {
  ImqQueueManager manager ;
  int iReturnCode = 0 ;

  // Connect to the queue manager.
  if ( argc > 2 ) {
    manager.setName( argv[ 2 ] );
  }
  if ( manager.connect( ) ) {
    ImqQueue * pqueue = new ImqQueue ;
    ImqMessage * pmsg = new ImqMessage ;

    // Identify the queue which will hold the message.
    pqueue -> setConnectionReference( manager );
    if ( argc > 1 ) {
      pqueue -> setName( argv[ 1 ] );


      // The named queue can be a model queue, which will result in
      // the creation of a temporary dynamic queue, which will be
      // destroyed as soon as it is closed. Therefore we must ensure
      // that such a queue is not automatically closed and reopened.
      // We do this by setting open options which will avoid the need
      // for closure and reopening.
      pqueue -> setOpenOptions( MQOO_OUTPUT | MQOO_INPUT_SHARED |
                                MQOO_INQUIRE );
    } else {
      pqueue -> setName( EXISTING_QUEUE );

      // The existing queue is not a model queue, and will not be
      // destroyed by automatic closure and reopening. Therefore we
      // will let the open options be selected on an as-needed basis.
      // The queue will be opened implicitly with an output option
      // during the "put", and then implicitly closed and reopened
      // with the addition of an input option during the "get".
    }

    // Prepare a message containing the text "Hello world".
    pmsg -> useFullBuffer( gpszHello , BUFFER_SIZE );
    pmsg -> setFormat( MQFMT_STRING );

    // Place the message on the queue, using default put message
    // Options.
    // The queue will be automatically opened with an output option.
    if ( pqueue -> put( * pmsg ) ) {
      ImqString strQueue( pqueue -> name( ) );

      // Discover the name of the queue manager.
      ImqString strQueueManagerName( manager.name( ) );
      printf( "The queue manager name is %s.\n",
              (char *)strQueueManagerName );

      // Show the name of the queue.
      printf( "Message sent to %s.\n", (char *)strQueue );

      // Retrieve the data message just sent ("Hello world" expected)
      // from the queue, using default get message options. The queue
      // is automatically closed and reopened with an input option
      // if it is not already open with an input option. We get the
      // message just sent, rather than any other message on the
      // queue, because the "put" will have set the ID of the message
      // so, as we are using the same message object, the message ID
      // acts as in the message object, a filter which says that we
      // are interested in a message only if it has this
      // particular ID.

      if ( pqueue -> get( * pmsg ) ) {
```

```
        int iDataLength = pmsg -> dataLength( );

        // Show the text of the received message.
        printf( "Message of length %d received, ", iDataLength );

        if ( pmsg -> formatIs( MQFMT_STRING ) ) {
          char * pszText = pmsg -> bufferPointer( );

          // If the last character of data is a null, then we can
          // assume that the data can be interpreted as a text
          // string.
          if ( ! pszText[ iDataLength - 1 ] ) {
            printf( "text is \"%s\".\n", pszText );
          } else {
            printf( "no text.\n" );
          }

        } else {
          printf( "non-text message.\n" );
        }
      } else {
        printf( "ImqQueue::get failed with reason code %ld\n",
                pqueue -> reasonCode( ) );
        iReturnCode = (int)pqueue -> reasonCode( );
      }

    } else {
      printf( "ImqQueue::open/put failed with reason code %ld\n",
              pqueue -> reasonCode( ) );
      iReturnCode = (int)pqueue -> reasonCode( );
    }

    // Deletion of the queue will ensure that it is closed.
    // If the queue is dynamic then it will also be destroyed.
    delete pqueue ;
    delete pmsg ;

  } else {
    printf( "ImqQueueManager::connect failed with reason code %ld\n"
            manager.reasonCode( ) );
    iReturnCode = (int)manager.reasonCode( );
  }

  // Destruction of the queue manager ensures that it is
  // disconnected.  If the queue object were still available
  // and open (which it is not), the queue would be closed
  // prior to disconnection.

  return iReturnCode ;
}
```

程式範例 SPUT (imqsput.cpp) 和 SGET (imqsget.cpp)

這些 C++ 程式會將訊息放置在具名佇列中，並從該佇列擷取訊息。

這些範例顯示使用下列類別:

- ImqError (請參閱 ImqError C++ 類別)
- ImqMessage (請參閱 ImqMessage C++ 類別)
- ImqObject (請參閱 ImqObject C++ 類別)
- ImqQueue (請參閱 ImqQueue C++ 類別)
- ImqQueue 管理程式 (請參閱 ImqQueueManager C++ 類別)

請遵循適當的指示來執行程式。

在 z/OS 以外的所有平台上

1. 執行 **imqsputs** *queue-name*。
2. 在主控台鍵入文字行。 這些行會作為訊息放置在指定的佇列上。
3. 輸入空值行以結束輸入。
4. 執行 **imqsgets** *queue-name* 以擷取所有行，並在主控台顯示它們。

z/OS 如需相關資訊，請參閱第 462 頁的『在 z/OS Batch、RRS Batch 和 CICS 上建置 C++ 程式』。

開啟 z/OS

z/OS

1. 使用範例 JCL **imqsputr** 來建構並執行批次工作。 從 SYSIN 資料集讀取訊息。
2. 使用範例 JCL **imqsgetr** 來建構並執行批次工作。 訊息會從佇列擷取並傳送至 SYSPRINT 資料集。

範例程式 DPUT (imqdput.cpp)

此 C++ 範例程式會將訊息放入由兩個佇列組成的配送清單中。

DPUT 顯示使用 ImqDistributionList 類別 (請參閱 ImqDistributionList C++ 類別)。 z/OS 不支援此範例。

1. 執行 **imqdput** *queue-name-1 queue-name-2* ，將訊息放在兩個指名的佇列上。
2. 執行 **imqsgets** *queue-name-1* 及 **imqsgets** *queue-name-2* ， 以從那些佇列擷取訊息。

C++ 語言考量

此主題集合詳細說明在撰寫使用「訊息佇列介面 (MQI)」的應用程式時，您必須考量的 C++ 語言使用情形及使用慣例的各個層面。

C++ 標頭檔

標頭檔作為 MQI 定義的一部分提供，可協助您以 C++ 語言撰寫 IBM MQ 應用程式。

下表彙總這些標頭檔。

表 *74: C/C++* 標頭檔	
檔名	**內容**
IMQI.HPP	C++ MQI 類別 (包括 CMQC.H 和 IMQTYPE.H)
IMQTYPE.H	定義 **ImqBoolean** 資料類型
CMQC.H	MQI 資料結構和資訊清單常數

若要改進應用程式的可攜性，請在 **#include** 前置處理器指引上以小寫形式撰寫標頭檔的名稱:

```
#include <imqi.hpp> // C++ classes
```

C++ 方法及屬性

方法名稱大小寫混合。 各種考量適用於參數和回覆值。 屬性是使用 set 和 get 方法來存取 (視適用情況而定)。

const 方法的參數僅適用於輸入。 Parameters with signatures including a pointer (*) or a reference (&) are passed by reference. 不包含指標或參照的回覆值會依值傳遞; 在傳回物件的情況下，這些是成為呼叫者責任的新實體。

部分方法簽章包括採用預設值的項目 (如果未指定的話)。 此類項目一律位於簽章結尾，並以等號 (=) 表示; 等號後面的值指出省略項目時適用的預設值。

這些類別中的所有方法名稱都是大小寫混合，從小寫開始。 除了方法名稱內的第一個單字之外，每一個單字都以大寫字母開頭。 除非廣泛理解縮寫的含義，否則不會使用縮寫。 所用的縮寫包括 *id* (代表身分) 和 *sync* (代表同步化)。

使用 set 和 get 方法來存取物件屬性。 set 方法以單字 *set* 開始; get 方法沒有字首。 如果屬性是 唯讀，則沒有 set 方法。

在物件建構期間，屬性會起始設定為有效狀態，且物件的狀態一律一致。

C++ 中的資料類型

所有資料類型都由 C **typedef** 陳述式定義。

類型 **ImqBoolean** 在 IMQTYPE.H 中定義為 **unsigned char** ，並且可以具有值 TRUE 和 FALSE。 您可以使用 **ImqBinary** 類別物件來取代 **MQBYTE** 陣列，並使用 **ImqString** 類別物件來取代 **char ***。 許多方法會傳回物件而非 **char** 或 **MQBYTE** 指標，以簡化儲存體管理。 所有回覆值都會變成呼叫者的責任，而且在傳回物件的情況下，可以使用刪除來刪除儲存體。

在 C++ 中操作二進位字串

二進位資料的字串宣告為 **ImqBinary** 類別的物件。 此類別的物件可以使用熟悉的 C 運算子來複製、比較及設定。 提供程式碼範例。

下列程式碼範例顯示二進位字串的作業:

```
#include <imqi.hpp> // C++ classes

ImqMessage message ;
ImqBinary id, correlationId ;
MQBYTE24 byteId ;

correlationId.set( byteId, sizeof( byteId ) ); // Set.
id = message.id( );                            // Assign.
if ( correlationId == id ) {                   // Compare.
...
```

在 C++ 中操作字串

通常會在 **ImqString** 類別物件中傳回字元資料，可使用轉換運算子強制轉型為 **char *** 。 ImqString 類別包含協助處理字串的方法。

使用 MQI C++ 方法接受或傳回字元資料時，字元資料一律以空值結尾，且可以任意長度。 不過， IBM MQ 會強制某些限制，可能導致資訊被截斷。 為了簡化儲存體管理，通常會在 **ImqString** 類別物件中傳回字元資料。 在許多需要 **char *** 的狀況下，可以使用所提供的轉換運算子將這些物件強制轉型為 **char *** ，並用於唯讀 目的。

註: 來自 **ImqString** 類別物件的 **char *** 轉換結果可能是空值。

雖然可以在 **char ***上使用 C 函數，但 **ImqString** 類別有一些特殊方法更適合; **operator length** () 相當於 **strlen** 和 **storage** () 指出配置給字元資料的記憶體。

C++ 中物件的起始狀態

所有物件都有其屬性所反映的一致起始狀態。 起始值定義在類別說明中。

使用 C++ 中的 C

當您從 C++ 程式使用 C 函數時，請包含適當的標頭。

下列範例顯示 C++ 程式中包含的 `string.h` :

```
extern "C" {
#include <string.h>
}
```

C++ 符號慣例

這個範例顯示如何呼叫方法及宣告參數。

此程式碼範例使用方法及參數 **ImqBoolean ImqQueue**:: **取得** (**ImqMessage &** 訊息)

宣告並使用參數，如下所示:

```
ImqQueueManager * pmanager ;    // Queue manager
ImqQueue * pqueue ;             // Message queue
ImqMessage msg ;                // Message
char szBuffer[ 100 ];          // Buffer for message data

pmanager = new ImqQueueManager ;
pqueue = new ImqQueue ;
pqueue -> setName( "myreplyq" );
pqueue -> setConnectionReference( pmanager );

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );

if ( pqueue -> get( msg ) ) {
  long lDataLength = msg.dataLength( );

...
}
```

C++ 中的隱含作業

即時可以隱含地執行數個作業，以滿足順利執行方法的必要條件。 這些隱含作業為連接、開啟、重新開啟、關閉及中斷連線。 您可以使用類別屬性來控制連接及開啟隱含行為。

連接

針對導致對 MQI 進行任何呼叫的任何方法，會自動連接「ImqQueue 管理程式」物件 (請參閱 C++ 及 MQI 交互參照)。

開啟

針對任何導致 MQGET、MQINQ、MQPUT 或 MQSET 呼叫的方法，會自動開啟 ImqObject 物件。 使用 **openFor** 方法來指定一或多個相關 **開啟選項** 值。

重新開啟

會針對導致 MQGET、MQINQ、MQPUT 或 MQSET 呼叫的任何方法自動重新開啟 ImqObject ，其中已開啟物件，但現有的 **開啟選項** 不足以讓 MQI 呼叫成功。 使用暫時 **關閉選項** 值 MQCO_NONE 暫時關閉物件。 使用 **openFor** 方法來新增相關的**開啟選項**。

在特定情況下，重新開啟可能會導致問題:

- 暫時動態佇列會在關閉時毀損，且永遠無法重新開啟。
- 在關閉及重新開啟期間，其他人可能會在商機視窗中存取針對專用輸入開啟的佇列 (明確或依預設)。
- 當佇列關閉時，瀏覽游標位置會遺失。 此狀況不會阻止關閉及重新開啟，但會阻止後續使用游標，直到再次使用 MQGMO_BROWSE_FIRST 為止。
- 當佇列關閉時，所擷取的最後一則訊息的環境定義會遺失。

如果發生或可以預見其中任何情況，請避免重新開啟，方法是在開啟物件之前明確設定適當的 **開啟選項** (明確或隱含)。

針對複式佇列處理狀況明確設定 **開啟選項** ，會產生更好的效能，並避免與使用重新開啟相關聯的問題。

關閉

在物件狀態不再可行的任何點，例如，如果 ImqObject 連線參照已中斷，或 ImqObject 物件已毀損，則會自動關閉 ImqObject 。

斷線

在連線不再可行的任何點，例如，如果 ImqObject 連線參照已中斷，或 ImqQueue 管理程式物件已毀損，則會自動中斷「ImqQueue 管理程式」的連線。

C++ 中的二進位和字串

ImqString 類別會封裝傳統 *char* * 資料格式。 ImqBinary 類別會封裝二進位位元組陣列。 某些設定字元資料的方法可能會截斷資料。

設定字元的方法 (**char ***) 資料一律會取得資料副本，但部分方法可能會截斷副本，因為 IBM MQ 會強制某些限制。

ImqString 類別 (請參閱 ImqString C++ 類別) 封裝傳統 **char *** ， 並提供下列支援:

- 比較
- 連結
- 正在複製
- 整數至文字及文字至整數轉換
- 記號 (單字) 擷取
- 大寫轉換

ImqBinary 類別 (請參閱 ImqBinary C++ 類別) 封裝任意大小的二進位位元組陣列。 特別是用來保留下列屬性:

- **帳戶記號** (MQBYTE32)
- **連線標籤** (MQBYTE128)
- **相關性 ID** (MQBYTE24)
- **機能記號** (MQBYTE8)
- **群組 ID** (MQBYTE24)
- **實例 ID** (MQBYTE24)
- **訊息 ID** (MQBYTE24)
- **訊息記號** (MQBYTE16)
- **交易實例 ID** (MQBYTE16)

其中這些屬性屬於下列類別的物件:

- ImqCICSBridgeHeader (請參閱 ImqCICSBridgeHeader C++ 類別)
- ImqGetMessageOptions (請參閱 ImqGetMessageOptions C++ 類別)
- ImqIMSBridgeHeader (請參閱 ImqIMSBridgeHeader C++ 類別)
- ImqMessage 追蹤器 (請參閱 ImqMessageTracker C++ 類別)
- ImqQueue 管理程式 (請參閱 ImqQueueManager C++ 類別)
- ImqReference 標頭 (請參閱 ImqReference 標頭 C++ 類別)
- ImqWork 標頭 (請參閱 ImqWork 標頭 C++ 類別)

ImqBinary 類別也支援比較和複製。

C++ 中不受支援的函數

IBM MQ C++ 類別及方法與 IBM MQ 平台無關。 因此，它們可能提供在特定平台上不受支援的部分功能。

如果您嘗試在不支援某個函數的平台上使用該函數，則 IBM MQ 會偵測該函數，但 C++ 語言連結不會偵測該函數。 IBM MQ 會向您的程式報告錯誤，如同任何其他 MQI 錯誤一樣。

C++ 中的傳訊

這個主題集合詳細說明如何在 C++ 中準備、讀取及寫入訊息。

在 C++ 中準備訊息資料

訊息資料準備在緩衝區中，可由系統或應用程式提供。 任何一種方法都有優點。 提供使用緩衝區的範例。

當您傳送訊息時，首先會在 ImqCache 物件所管理的緩衝區中準備訊息資料 (請參閱 ImqCache C++ 類別)。 緩衝區與每一個 ImqMessage 物件相關聯 (透過繼承) (請參閱 ImqMessage C++ 類別): 它可以由應用程式提供 (使用 **useEmptyBuffer** 或 **useFullBuffer** 方法) 或由系統自動提供。 應用程式提供訊息緩衝區的優點是在許多情況下不需要複製資料，因為應用程式可以直接使用準備好的資料區。 缺點是所提供的緩衝區是固定長度。

透過在傳輸之前使用 **setMessageLength** 方法，可以重複使用緩衝區，且每次都可以改變傳輸的位元組數。

當系統自動提供時，系統會管理可用的位元組數，並且可以使用例如 ImqCache **write** 方法或 ImqMessage **writeItem** 方法，將資料複製到訊息緩衝區。 訊息緩衝區會根據需要成長。 隨著緩衝區成長，不會遺失先前寫入的資料。 可以循序寫入大型或多組件訊息。

下列範例顯示簡化的訊息傳送。

1. 在使用者提供的緩衝區中使用備妥資料

```
char szBuffer[ ] = "Hello world" ;

msg.useFullBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
```

2. 在使用者提供的緩衝區中使用備妥資料，其中緩衝區大小超出資料大小

```
char szBuffer[ 24 ] = "Hello world" ;

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
msg.setMessageLength( 12 );
```

3. 將資料複製到使用者提供的緩衝區

```
char szBuffer[ 12 ];

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
msg.write( 12, "Hello world" );
```

4. 將資料複製到系統提供的緩衝區

```
msg.setFormat( MQFMT_STRING );
msg.write( 12, "Hello world" );
```

5. 使用物件 (物件設定訊息格式及內容) 將資料複製到系統提供的緩衝區

```
ImqString strText( "Hello world" );

msg.writeItem( strText );
```

在 C++ 中讀取訊息

應用程式或系統可以提供緩衝區。 資料可以直接從緩衝區存取或循序讀取。 每一種訊息類型都有一個相等的類別。 提供範例程式碼。

當接收資料時，應用程式或系統可以提供適當的訊息緩衝區。 對於特定 ImqMessage 物件的多重傳輸和多重接收，可以使用相同的緩衝區。 如果自動提供訊息緩衝區，則它會成長以容納任何收到的資料長度。 不過，應用程式提供的訊息緩衝區可能不夠大，無法保留收到的資料。 然後可能會發生截斷或失敗，視用於訊息接收的選項而定。

可以直接從訊息緩衝區存取送入資料，在此情況下，資料長度會指出送入資料的總量。 或者，可以從訊息緩衝區循序讀取送入的資料。 在此情況下，資料指標會定址送入資料的下一個位元組，而且每次讀取資料時都會更新資料指標及資料長度。

項目 是訊息的片段，全部都在訊息緩衝區的使用者區域中，需要循序個別處理。 除了一般使用者資料之外，項目可能是無法傳送郵件的標頭或觸發訊息。 項目一律與訊息格式相關聯; 訊息格式 **並非** 一律與項目相關聯。

每一個項目都有一個物件類別，對應於可辨識的 IBM MQ 訊息格式。 有一個用於無法傳送郵件的標頭，一個用於觸發訊息。 沒有使用者資料的物件類別。 亦即，一旦已用盡可辨識的格式，則會將剩餘的處理程序留給應用程式。 使用者資料的類別可以透過特殊處理 ImqItem 類別來寫入。

下列範例顯示在虛數狀況中，考慮在使用者資料之前的許多潛在項目的訊息回執。 非項目使用者資料定義為可在可識別項目之後發生的任何項目。 自動緩衝區 (預設值) 用來保留任意數量的訊息資料。

```
ImqQueue queue ;
ImqMessage msg ;

if ( queue.get( msg ) ) {

  /* Process all items of data in the message buffer. */
  do while ( msg.dataLength( ) ) {
    ImqBoolean bFormatKnown = FALSE ;
    /* There remains unprocessed data in the message buffer. */

    /* Determine what kind of item is next. */

    if ( msg.formatIs( MQFMT_DEAD_LETTER_HEADER ) ) {
      ImqDeadLetterHeader header ;
      /* The next item is a dead-letter header.                */
      /* For the next statement to work and return TRUE,       */
      /* the correct class of object pointer must be supplied. */
      bFormatKnown = TRUE ;

      if ( msg.readItem( header ) ) {
        /* The dead-letter header has been extricated from the */
        /* buffer and transformed into a dead-letter object.   */
        /* The encoding and character set of the dead-letter   */
        /* object itself are MQENC_NATIVE and MQCCSI_Q_MGR.    */
        /* The encoding and character set from the dead-letter */
        /* header have been copied to the message attributes   */
        /* to reflect any remaining data in the buffer.        */

        /* Process the information in the dead-letter object.  */
        /* Note that the encoding and character set have       */
        /* already been processed.                             */
        ...
      }
      /* There might be another item after this, */
      /* or just the user data.                  */
    }
    if ( msg.formatIs( MQFMT_TRIGGER ) ) {
      ImqTrigger trigger ;
      /* The next item is a trigger message.                   */
      /* For the next statement to work and return TRUE,       */
      /* the correct class of object pointer must be supplied. */
      bFormatKnown = TRUE ;
      if ( msg.readItem( trigger ) ) {

        /* The trigger message has been extricated from the */
        /* buffer and transformed into a trigger object.    */
        /* Process the information in the trigger object. */
        ...
      }

      /* There is usually nothing after a trigger message. */
    }

    if ( msg.formatIs( FMT_USERCLASS ) ) {
      UserClass object ;
      /* The next item is an item of a user-defined class.     */
      /* For the next statement to work and return TRUE,       */
      /* the correct class of object pointer must be supplied. */
      bFormatKnown = TRUE ;

      if ( msg.readItem( object ) ) {
        /* The user-defined data has been extricated from the */
        /* buffer and transformed into a user-defined object. */

        /* Process the information in the user-defined object. */
        ...
      }
```

```
      /* Continue looking for further items. */
    }
    if ( ! bFormatKnown ) {
      /* There remains data that is not associated with a specific*/
      /* item class.                                               */
      char * pszDataPointer = msg.dataPointer( );       /* Address.*/
      int iDataLength = msg.dataLength( );              /* Length. */

      /* The encoding and character set for the remaining data are */
      /* reflected in the attributes of the message object, even   */
      /* if a dead-letter header was present.                      */
      ...

    }

  }
}
```

在此範例中， FMT_USERCLASS 是一個常數，代表與類別 UserClass 的物件相關聯的 8 個字元格式名稱，並由應用程式定義。

UserClass 衍生自 ImqItem 類別 (請參閱 ImqItem C++ 類別)， 並從該類別實作虛擬 **copyOut** 及 **pasteIn** 方法。

接下來的兩個範例顯示來自 ImqDeadLetterHeader 類別的程式碼 (請參閱 ImqDeadLetterHeader C++ 類別)。 第一個範例顯示自訂封裝的訊息- *writing* 程式碼。

```
// Insert a dead-letter header.
// Return TRUE if successful.
ImqBoolean ImqDeadLetterHeader :: copyOut ( ImqMessage & msg ) {
  ImqBoolean bSuccess ;
  if ( msg.moreBytes( sizeof( omqdlh ) ) ) {
    ImqCache cacheData( msg ); // Preserve original message content.
    // Note original message attributes in the dead-letter header.
    setEncoding( msg.encoding( ) );
    setCharacterSet( msg.characterSet( ) );
    setFormat( msg.format( ) );

    // Set the message attributes to reflect the dead-letter header.
    msg.setEncoding( MQENC_NATIVE );
    msg.setCharacterSet( MQCCSI_Q_MGR );
    msg.setFormat( MQFMT_DEAD_LETTER_HEADER );
    // Replace the existing data with the dead-letter header.
    msg.clearMessage( );
    if ( msg.write( sizeof( omqdlh ), (char *) & omqdlh ) ) {
      // Append the original message data.
      bSuccess = msg.write( cacheData.messageLength( ),
                            cacheData.bufferPointer( ) );
    } else {
      bSuccess = FALSE ;
    }
  } else {
    bSuccess = FALSE ;
  }
  // Reflect and cache error in this object.
  if ( ! bSuccess ) {
    setReasonCode( msg.reasonCode( ) );
    setCompletionCode( msg.completionCode( ) );
  }

  return bSuccess ;
}
```

第二個範例顯示自訂封裝訊息- *reading* 程式碼。

```
// Read a dead-letter header.
// Return TRUE if successful.
ImqBoolean ImqDeadLetterHeader :: pasteIn ( ImqMessage & msg ) {
  ImqBoolean bSuccess = FALSE ;

  // First check that the eye-catcher is correct.
  // This is also our guarantee that the "character set" is correct.
  if ( ImqItem::structureIdIs( MQDLH_STRUC_ID, msg ) ) {
    // Next check that the "encoding" is correct, as the MQDLH
    // contains numeric data.
```

```
    if ( msg.encoding( ) == MQENC_NATIVE ) {

      // Finally check that the "format" is correct.
      if ( msg.formatIs( MQFMT_DEAD_LETTER_HEADER ) ) {
        char * pszBuffer = (char *) & omqdlh ;
        // Transfer the MQDLH from the message and move pointer on.
        if ( bSuccess = msg.read( sizeof( omdlh ), pszBuffer ) ) {
          // Update the encoding, character set and format of the
          // message to reflect the remaining data.
          msg.setEncoding( encoding( ) );
          msg.setCharacterSet( characterSet( ) );
          msg.setFormat( format( ) );
        } else {

          // Reflect the cache error in this object.
          setReasonCode( msg.reasonCode( ) );
          setCompletionCode( msg.completionCode( ) );
        }
      } else {
        setReasonCode( MQRC_INCONSISTENT_FORMAT );
        setCompletionCode( MQCC_FAILED );
      }
    } else {
      setReasonCode( MQRC_ENCODING_ERROR );
      setCompletionCode( MQCC_FAILED );
    {
  } else {
    setReasonCode( MQRC_STRUC_ID_ERROR );
    setCompletionCode( MQCC_FAILED );
  }

  return bSuccess ;
}
```

使用自動緩衝區時，緩衝區儲存體是 *volatile*。 也就是說，在每一個 **get** 方法呼叫之後，緩衝區資料可能會保留在不同的實體位置。 因此，每次參照緩衝區資料時，請使用 **bufferPointer** 或 **dataPointer** 方法來存取訊息資料。

您可能想要程式設定固定區域來接收訊息資料。 在此情況下，在使用 **get** 方法之前，請先呼叫 **useEmptyBuffer** 方法。

使用固定且非自動區域會將訊息限制為大小上限，因此請務必考量 ImqGetMessageOptions 物件的 MQGMO_ACCEPT_TRUNCATED_MSG 選項。 如果未指定此選項 (預設值) ，則預期 MQRC_TRUNCATED_MSG_FAILED 原因碼。 如果指定此選項，則視應用程式的設計而定，可能預期 MQRC_TRUNCATED_MSG_ACCEPTED 原因碼。

下一個範例顯示如何使用固定儲存體區域來接收訊息:

```
char * pszBuffer = new char[ 100 ];

msg.useEmptyBuffer( pszBuffer, 100 );
gmo.setOptions( MQGMO_ACCEPT_TRUNCATED_MSG );
queue.get( msg, gmo );

delete [ ] pszBuffer ;
```

在此程式碼片段中，緩衝區一律可以使用 *pszBuffer* 直接定址，而不是使用 **bufferPointer** 方法。 不過，最好使用 **dataPointer** 方法來進行一般用途存取。 應用程式 (不是 ImqCache 類別物件) 必須捨棄使用者定義 (非自動) 緩衝區。

注意: 使用 **useEmpty 緩衝區** 指定空值指標和零長度，並不會如預期般指定長度為零的固定長度緩衝區。 此組合會解譯為要求忽略任何先前使用者定義的緩衝區，並改為回復為使用自動緩衝區。

在 C++ 中將訊息寫入無法傳送郵件的佇列

將訊息寫入無法傳送郵件的佇列的程式程式碼範例。

多組件訊息的典型案例是包含無法傳送郵件的標頭。 無法處理之訊息中的資料會附加至無法傳送郵件的標頭。

```
ImqQueueManager mgr ;        // The queue manager.
```

```
ImqQueue queueIn ;           // Incoming message queue.
ImqQueue queueDead ;         // Dead-letter message queue.
ImqMessage msg ;             // Incoming and outgoing message.
ImqDeadLetterHeader header ; // Dead-letter header information.

// Retrieve the message to be rerouted.
queueIn.setConnectionReference( mgr );
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Set up the dead-letter header information.
header.setDestinationQueueManagerName( mgr.name( ) );
header.setDestinationQueueName( queueIn.name( ) );
header.setPutApplicationName( /* ? */ );
header.setPutApplicationType( /* ? */ );
header.setPutDate( /* TODAY */ );
header.setPutTime( /* NOW */ );
header.setDeadLetterReasonCode( FB_APPL_ERROR_1234 );

// Insert the dead-letter header information. This will vary
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );

// Send the message to the dead-letter queue.
queueDead.setConnectionReference( mgr );
queueDead.setName( mgr.deadLetterQueueName( ) );
queueDead.put( msg );
```

以 C++ 將訊息寫入 IMS 橋接器

將訊息寫入 IMS 橋接器的程式碼範例。

傳送至 IBM MQ - IMS 橋接器的訊息可能使用特殊標頭。 IMS 橋接器標頭會以一般訊息資料為字首。

```
ImqQueueManager    mgr;         // The queue manager.
ImqQueue           queueBridge; // IMS bridge message queue.
ImqMessage         msg;         // Outgoing message.
ImqIMSBridgeHeader header;      // IMS bridge header.

// Set up the message.
//
// Here we are constructing a message with format
// MQFMT_IMS_VAR_STRING, and appropriate data.
//
msg.write( 2,       /* ? */ );         // Total message length.
msg.write( 2,       /* ? */ );         // IMS flags.
msg.write( 7,       /* ? */ );         // Transaction code.
msg.write( /* ? */, /* ? */ );         // String data.
msg.setFormat( MQFMT_IMS_VAR_STRING ); // The format attribute.

// Set up the IMS bridge header information.
//
// The reply-to-format is often specified.
// Other attributes can be specified, but all have default values.
//
header.setReplyToFormat( /* ? */ );

// Insert the IMS bridge header into the message.
//
// This will:
// 1) Insert the header into the message buffer, before the existing
//    data.
// 2) Copy attributes out of the message descriptor into the header,
//    for example the IMS bridge header format attribute will now
//    be set to MQFMT_IMS_VAR_STRING.
// 3) Set up the message attributes to describe the header, in
//    particular setting the message format to MQFMT_IMS.
//
msg.writeItem( header );

// Send the message to the IMS bridge queue.
//
queueBridge.setConnectionReference( mgr );
queueBridge.setName( /* ? */ );
queueBridge.put( msg );
```

以 C++ 將訊息寫入 CICS bridge

將訊息寫入 CICS bridge 的程式程式碼範例。

使用 CICS bridge 傳送至 IBM MQ for z/OS 的訊息需要特殊標頭。 CICS bridge 標頭會以一般訊息資料為字首。

```
ImqQueueManager mgr ;          // The queue manager.
ImqQueue queueIn ;             // Incoming message queue.
ImqQueue queueBridge ;         // CICS bridge message queue.
ImqMessage msg ;               // Incoming and outgoing message.
ImqCicsBridgeHeader header ;   // CICS bridge header information.

// Retrieve the message to be forwarded.
queueIn.setConnectionReference( mgr );
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Set up the CICS bridge header information.
// The reply-to format is often specified.
// Other attributes can be specified, but all have default values.
header.setReplyToFormat( /* ? */ );

// Insert the CICS bridge header information.  This will vary
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );

// Send the message to the CICS bridge queue.
queueBridge.setConnectionReference( mgr );
queueBridge.setName( /* ? */ );
queueBridge.put( msg );
```

以 C++ 撰寫含有工作標頭的訊息

用於寫入目的地為 z/OS Workload Manager 所管理之佇列的訊息的程式碼範例。

傳送至 IBM MQ for z/OS 的訊息 (以 z/OS Workload Manager 所管理的佇列為目的地) 需要特殊標頭。 工作標頭會以一般訊息資料為字首。

```
ImqQueueManager mgr ;              // The queue manager.
ImqQueue queueIn ;                 // Incoming message queue.
ImqQueue queueWLM ;                // WLM managed queue.
ImqMessage msg ;                   // Incoming and outgoing message.
ImqWorkHeader header ;             // Work header information

// Retrieve the message to be forwarded.
queueIn.setConnectionReference( mgr );
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Insert the Work header information.  This will vary
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );

// Send the message to the WLM managed queue.
queueWLM.setConnectionReference( mgr );
queueWLM.setName( /* ? */ );
queueWLM.put( msg );
```

建置 IBM MQ C++ 程式

會列出受支援編譯器的 URL，以及用來在 IBM MQ 平台上編譯、鏈結及執行 C++ 程式和範例的指令。

如需每一個受支援平台及 IBM MQ 版本的編譯器清單，請參閱 IBM MQ 的系統需求。

編譯及鏈結 IBM MQ C++ 程式所需的指令取決於您的安裝及需求。 下列範例顯示在許多平台上使用預設 IBM MQ 安裝的部分編譯器的一般編譯及鏈結指令。

AIX 在 AIX 上建置 C++ 程式

使用 XL C Enterprise Edition 編譯器在 AIX 上建置 IBM MQ C++ 程式。

V 9.3.5 如需 XLC 16 與 XLC 17 編譯器之間不同編譯器選項對映的相關資訊，請參閱 選項對映。

Deprecated V 9.3.5 IBM MQ 9.3.5 已淘汰對 AIX 上 XL C/C++ for AIX 16 編譯器的支援。

用戶端

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

32 位元無執行緒應用程式

```
xlC -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ia -limqb23ia -lmqic
```

32 位元執行緒應用程式

```
xlC_r -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ia_r -limqb23ia_r -lmqic_r
```

64 位元無執行緒應用程式

```
xlC -q64 -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ia -limqb23ia -lmqic
```

64 位元執行緒應用程式

```
xlC_r -q64 -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ia_r -limqb23ia_r -lmqic_r
```

V 9.3.5 **32 位元無執行緒應用程式 (XLC 17)**

```
ibm-clang++_r -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ca -limqb23ca -lmqic
```

V 9.3.5 **32 位元執行緒應用程式 (XLC 17)**

```
ibm-clang++_r -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ca_r -limqb23ca_r -lmqic_r
```

V 9.3.5 **64 位元無執行緒應用程式 (XLC 17)**

```
ibm-clang++_r -m64 -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ca -limqb23ca -lmqic
```

V 9.3.5 **64 位元執行緒應用程式 (XLC 17)**

```
ibm-clang++_r -m64 -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ca_r -limqb23ca_r -lmqic_r
```

伺服器

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

32 位元無執行緒應用程式

```
xlC -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ia -limqb23ia -lmqm
```

32 位元執行緒應用程式

```
xlC_r -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ia_r -limqb23ia_r -lmqm_r
```

64 位元無執行緒應用程式

```
xlC -q64 -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ia -limqb23ia -lmqm
```

64 位元執行緒應用程式

```
xlC_r -q64 -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ia_r -limqb23ia_r -lmqm_r
```

V 9.3.5 **32 位元無執行緒應用程式 (XLC 17)**

```
ibm-clang++_r -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ca -limqb23ca -lmqm
```

V 9.3.5 **32 位元執行緒應用程式 (XLC 17)**

```
ibm-clang++_r -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ca_r -limqb23ca_r -lmqm_r
```

V 9.3.5 **64 位元無執行緒應用程式 (XLC 17)**

```
ibm-clang++_r -m64 -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ca -limqb23ca -lmqm
```

V 9.3.5 **64 位元執行緒應用程式 (XLC 17)**

```
ibm-clang++_r -m64 -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ca_r -limqb23ca_r -lmqm_r
```

IBM i 在 IBM i 上建置 C++ 程式

使用 ILE C++ 編譯器在 IBM i 上建置 IBM MQ C++ 程式。

IBM ILE C++ for IBM i 是 C++ 程式的原生編譯器。 下列指示說明如何使用此編譯器，以使用 *Hello World!* 來建立 IBM MQ C++ 應用程式。 IBM MQ 範例程式作為範例。

1. 依照 先讀我*!* 中的指示，安裝 ILE C++ for IBM i 編譯器。 產品隨附的手冊。
2. 請確定 QCXXN 檔案庫在您的檔案庫清單中。
3. 建立 HELLO WORLD 範例程式:

   a. 建立模組:

   ```
   CRTCPPMOD MODULE(MYLIB/IMQWRLD) +
   SRCSTMF('/QIBM/ProdData/mqm/samp/imqwrld.cpp') +
   INCDIR('/QIBM/ProdData/mqm/inc') DFTCHAR(*SIGNED) +
   TERASPACE(*YES)
   ```

   C++ 範例程式的來源可以在 `/QIBM/ProdData/mqm/samp` 中找到，並可以在 `/QIBM/ProdData/mqm/inc` 中找到併入檔。

或者，可以在程式庫 SRCFILE(QCPPSRC/LIB) SRCMBR(IMQWRLD)中找到來源。

b. 將此與 IBM MQ 提供的服務程式連結，以產生程式物件:

```
CRTPGM PGM(MYLIB/IMQWRLD) MODULE(MYLIB/IMQWRLD) +
BNDSRVPGM(QMQM/IMQB23I4 QMQM/IMQS23I4)
```

若要建置執行緒應用程式，請使用重新進入的服務程式:

```
CRTPGM PGM(MYLIB/IMQWRLD) MODULE(MYLIB/IMQWRLD) +
BNDSRVPGM(QMQM/IMQB23I4[_R] QMQM/IMQS23I4[_R])
```

c. 使用 SYSTEM.DEFAULT.LOCAL.QUEUE:

```
CALL PGM(MYLIB/IMQWRLD)
```

Linux 在 Linux 上建置 C++ 程式

使用 GNU g + + 編譯器在 Linux 上建置 IBM MQ C++ 程式。

System p

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

用戶端: System p

32 位元無執行緒應用程式

```
g++ -m32 -o imqsputc_32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl
-limqb23gl -lmqic
```

32 位元執行緒應用程式

```
g++ -m32 -o imqsputc_r32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r
-limqb23gl_r -lmqic_r
```

64 位元無執行緒應用程式

```
g++ -m64 -o imqsputc_64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

64 位元執行緒應用程式

```
g++ -m64 -o imqsputc_r64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r
```

伺服器： System p

32 位元無執行緒應用程式

```
g++ -m32 -o imqsput_32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl
-limqb23gl -lmqm
```

32 位元執行緒應用程式

```
g++ -m32 -o imqsput_r32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r
-limqb23gl_r -lmqm_r
```

64 位元無執行緒應用程式

```
g++ -m64 -o imqsput_64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

64 位元執行緒應用程式

```
g++ -m64 -o imqsput_r64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r
```

IBM Z

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

用戶端: IBM Z

32 位元無執行緒應用程式

```
g++ -m31 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

32 位元執行緒應用程式

```
g++ -m31 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r -limqb23gl_r -lmqic_r
-lpthread
```

64 位元無執行緒應用程式

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

64 位元執行緒應用程式

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

伺服器： IBM Z

32 位元無執行緒應用程式

```
g++ -m31 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl -limqb23gl -lmqm
```

32 位元執行緒應用程式

```
g++ -m31 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

64 位元無執行緒應用程式

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

64 位元執行緒應用程式

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

x86-64 (32 位元)

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

用戶端: x86-64 (32 位元)

32 位元無執行緒應用程式

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L
MQ_INSTALLATION_PATH/lib -Wl,
-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqc23gl -limqb23gl -lmqic
```

32 位元執行緒應用程式

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

64 位元無執行緒應用程式

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl -limqb23gl
-lmqic
```

64 位元執行緒應用程式

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

伺服器: x86-64 (32 位元)

32 位元無執行緒應用程式

```
g++ -m32 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl -limqb23gl -lmqm
```

32 位元執行緒應用程式

```
g++ -m32 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

64 位元無執行緒應用程式

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

64 位元執行緒應用程式

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

Windows 在 Windows 上建置 C++ 程式

使用 Microsoft Visual Studio C++ 編譯器在 Windows 上建置 IBM MQ C++ 程式。

小心: IBM MQ 隨附的程式庫是動態程式庫，而不是靜態程式庫。 IBM MQ 提供稱為 "`import libraries`" 的項目，您只能在編譯時期使用。 對於執行時期，您必須使用動態程式庫。

從 IBM MQ 8.0.0 Fix Pack 4 開始， IBM MQ 提供可重新配送的用戶端，其中包含執行 IBM MQ 應用程式所需的程式庫。 這些程式庫可以與用戶端應用程式一起包裝及重新配送。 如需相關資訊，請參閱 Windows 上可重新配送的用戶端。

與 32 位元應用程式搭配使用的程式庫 (.lib) 檔案和 dll 檔案安裝在 *MQ_INSTALLATION_PATH*`/Tools/Lib` 中。 與 64 位元應用程式搭配使用的檔案安裝在 *MQ_INSTALLATION_PATH*`/Tools/Lib64` 中。 *MQ_INSTALLATION_PATH* 代表 IBM MQ 安裝所在的高階目錄。

用戶端

```
cl -MD imqsput.cpp /Feimqsputc.exe imqb23vn.lib imqc23vn.lib
```

伺服器

```
cl -MD imqsput.cpp /Feimqsput.exe imqb23vn.lib imqs23vn.lib
```

安裝通用 C 執行時期

如果您使用 Windows 8.1 或 Windows Server 2012 R2，則必須從 Microsoft 安裝通用 C 執行時期更新 (Universal CRT)。 此執行時期包含為 Windows 10 及 Windows Server 2016 的一部分。

通用 CRT 更新為 Microsoft update KB3118401。 您可以在 `C:\Windows\System32` 目錄中搜尋稱為 `ucrtbase.dll` 的檔案，以查看您是否有這項更新。 如果沒有，您可以從下列 Microsoft 頁面下載更新項目: https://www.catalog.update.microsoft.com/Search.aspx?q=kb3118401。

在未安裝執行時期的情況下，嘗試執行 IBM MQ 程式或您使用 Microsoft Visual Studio 2017 自行編譯的程式會導致下列錯誤:

```
The program can't start because api-ms-win-crt-runtime-|1-1-0.dll
is missing from your computer. Try reinstalling the program to
fix this problem.
```

提供 Microsoft Visual Studio 2012 程式的執行時期

如果您使用 Microsoft Visual Studio 2012 來編譯 IBM MQ 程式，請注意 IBM MQ 安裝程式不會安裝 Microsoft Visual Studio 2012 C/C++ 執行時期。 如果舊版 IBM MQ 已安裝在同一部電腦上，則該安裝架構會提供 Microsoft Visual Studio 2012 執行時期。

不過，如果您使用使用 Microsoft Visual Studio 2012 建置的程式，且未安裝舊版 IBM MQ ，則必須執行下列其中一項:

- 從 Microsoft 下載並安裝 **Microsoft Visual C++ Redistributable for VisualStudio 2017 (32 and 64-bit versions)** 。
- 使用 Microsoft Visual Studio 2017 或另一個安裝執行時期的 Microsoft Visual Studio 層次來重新編譯您的程式。

使用 Microsoft Visual Studio 2015 編譯器建置的 C++ 用戶端程式庫

IBM MQ 提供使用 Microsoft Visual Studio 2015 C++ 編譯器及 Microsoft Visual Studio 2017 C++ 編譯器建置的 C++ 用戶端程式庫。

同時提供 32 位元及 64 位元版本的 IBM MQ C++ 程式庫。 32 位元檔案庫安裝在 `bin\vs2015` 資料夾下，64 位元檔案庫安裝在 `bin64\vs2015` 資料夾下。

依預設， IBM MQ 會配置成使用 Microsoft Visual Studio 2017 程式庫。 若要使用 Microsoft Visual Studio 2015 程式庫，您必須在安裝 IBM MQ 之前，或在使用 **setmqenv** 或 **setmqinst** 指令之前，將 `MQ_PREFIX_VS_LIBRARIES` 環境變數設為 `MQ_PREFIX_VS_LIBRARIES=vs2015` 。

使用不同名稱的 IBM MQ C++ 程式庫

IBM MQ 提供一些以不同方式命名的其他 C++ 用戶端程式庫。 這些程式庫是使用 Microsoft Visual Studio 2015 及 Microsoft Visual Studio 2017 C++ 編譯器所建置。 除了使用 Microsoft Visual Studio 2017 C++ 編譯器建置的現有 C++ 程式庫之外，還提供了這些程式庫。 由於這些額外的 IBM MQ C++ 程式庫具有不同的名稱，因此您可以執行 IBM MQ C++ 應用程式，這些應用程式使用 IBM MQ C++ 建置，並在相同電腦上使用 Microsoft Visual Studio 2017 及舊版產品進行編譯。

其他 Microsoft Visual Studio 2017 程式庫具有下列名稱:

- `imqb23vnvs2017.dll`
- `imqc23vnvs2017.dll`
- `imqs23vnvs2017.dll`
- `imqx23vnvs2017.dll`

其他 Microsoft Visual Studio 2015 程式庫具有下列名稱:

- `imqb23vnvs2015.dll`
- `imqc23vnvs2015.dll`
- `imqs23vnvs2015.dll`
- `imqx23vnvs2015.dll`

同時提供這些程式庫的 32 位元及 64 位元版本。 32 位元程式庫安裝在 `bin` 資料夾下， 64 位元程式庫安裝在 `bin64` 資料夾下。 對應的匯入程式庫安裝在 `Tools\lib` 和 `Tools\lib64` 目錄下。

如果您的應用程式使用 `imq*vs2015.lib` 檔，您必須使用 Microsoft Visual Studio 2015 編譯器來編譯它。 若要執行使用 Microsoft Visual Studio 2015 編譯的 IBM MQ C++ 應用程式，或在相同電腦上使用舊版產品編譯的應用程式， PATH 環境變數必須具有字首，如下列範例所示:

- 若為 32 位元應用程式:

```
SET PATH=installation folder\bin\vs2015;%PATH%
```

- 若為 64 位元應用程式:

```
SET PATH=installation folder\bin64\vs2015;%PATH%
```

相關概念

Windows: 來自 IBM MQ 8.0 的變更

z/OS 在 z/OS Batch、RRS Batch 和 CICS 上建置 C++ 程式

在 z/OS 上針對批次、RRS 批次或 CICS 環境建置 IBM MQ C++ 程式，並執行範例程式。

您可以針對 IBM MQ for z/OS 支援的三個環境撰寫 C++ 程式:

- 批次
- RRS 批次
- CICS

編譯、前置鏈結及鏈結

透過編譯、預先鏈結及鏈結編輯 C++ 原始碼來建立 z/OS 應用程式。

IBM MQ C++ for z/OS 實作為 IBM C++ for z/OS 語言的 z/OS DLL。 使用 DLL，您可以在前置鏈結時間將提供的定義與編譯器輸出連結在一起。 這可讓鏈結器檢查您對 IBM MQ C++ 成員函數的呼叫。

註: 這三個環境各有三組側邊卡片組。

若要建置 IBM MQ for z/OS C++ 應用程式，請建立並執行 JCL。 請使用下列程序:

1. 如果您的應用程式在 CICS 下執行，請使用 CICS 提供的程序來轉換程式中的 CICS 指令。

   此外，對於 CICS 應用程式，您還需要:

   a. 將 SCSQLOAD 程式庫新增至 DFHRPL 連結。

   b. 使用 SCSQPROC 程式庫中的成員 IMQ4B100 來定義 CSQCAT1 CEDA 群組。

   c. 安裝 CSQCAT1。
2. 編譯程式以產生物件程式碼。 用於編譯的 JCL 必須包括讓產品資料定義檔可供編譯器使用的陳述式。 下列 IBM MQ for z/OS 程式庫中提供資料定義:

   - **thlqual**.SCSQC370
   - **thlqual**.SCSQHPPS

   請務必指定 /cxx 編譯器選項。

   註: 名稱 **thlqual** 是 z/OS 上 IBM MQ 安裝程式庫的高階限定元。
3. 預先鏈結在步驟 第 462 頁的『2』 中建立的物件程式碼，包括 **thlqual**.SCSQDEFS 中提供的下列定義側面:

   a. 批次: imqs23dm 和 imqb23dm

   b. RRS 批次的 imqs23dr 和 imqb23dr

   c. imqs23dc 和 imqb23dc for CICS

   這些是對應的 DLL。

   a. imqs23im 和 imqb23im 用於批次

   b. imqs23ir 及 imqb23ir for RRS 批次

   c. imqs23ic 和 imqb23ic，適用於 CICS
4. 鏈結-編輯在步驟 第 462 頁的『3』 中建立的物件程式碼，以產生載入模組，並將它儲存在應用程式載入程式庫中。

若要執行批次或 RRS 批次程式，請將程式庫 **thlqual**.SCSQAUTH 及 **thlqual**.SCSQLOAD 併入 STEPLIB 或 JOBLIB 資料集連結中。

若要執行 CICS 程式，請先讓系統管理者將它定義為 CICS 作為 IBM MQ 程式及交易。 然後您可以用平常的方式來執行它。

執行範例程式

這些程式在 第 441 頁的『C++ 範例程式』中有說明。

範例應用程式僅以來源表單提供。 這些檔案如下:

表 *75: z/OS* 範例程式檔案		
範例	**來源程式 (在程式庫 thlqual.SCSQCPPS 中)**	**JCL (在程式庫 thlqual.SCSQPROC 中)**
Hello World	imqwrld	imqwrldr
SPUT	imqsput	imqsputr
SGET	imqsget	imqsgetr

若要執行範例，請編譯並鏈結編輯它們，如同使用任何 C++ 程式一樣 (請參閱 第 462 頁的『在 z/OS Batch、RRS Batch 和 CICS 上建置 C++ 程式』)。 使用提供的 JCL 來建構及執行批次工作。 您一開始必須遵循 JCL 隨附的註解來自訂 JCL。

z/OS 在 z/OS UNIX System Services 上建置 C++ 程式

在 z/OS UNIX System Services 上建置 IBM MQ C++ 程式 (z/OS UNIX)。

如果要在 z/OS UNIX Shell 下建置應用程式，您必須提供編譯器存取權給 IBM MQ 併入檔 (位於 `thlqual.SCSQC370` 和 `hlqual.SCSQHPPS` 中) ， 以及針對兩個 DLL sidedecks (位於 `thlqual.SCSQDEFS` 中) 的鏈結。 在執行時期，應用程式需要存取 IBM MQ 資料集 `thlqual.SCSQLOAD`、 `thlqual.SCSQAUTH` 及其中一個語言特定資料集，例如 `thlqual.SCSQANLE`[6].

編譯中

1. 使用 TSO **oput** 指令或使用 FTP 將範例複製到檔案系統。 此範例的其餘部分假設您已將範例複製到名為 `/u/fred/sample` 的目錄中，並將它命名為 `imqwrld.cpp`。
2. 登入 z/OS UNIX Shell ， 並切換至您放置範例的目錄。
3. 設定 C++ 編譯器，讓它可以接受 DLL sidedeck 及 .cpp 檔案作為輸入:

```
/u/fred/sample:> export _CXX_EXTRA_ARGS=1
/u/fred/sample:> export _CXX_CXXSUFFIX="cpp"
```

4. 編譯並鏈結範例程式。 下列指令會鏈結程式與批次側邊卡片組; 可以改用 RRS 批次側邊卡片組。 \ 字元用來將指令分割成多行。 請勿輸入此字元; 請以單行輸入指令:

```
/u/fred/sample:> c++ -o imqwrld -I "//'thlqual.SCSQC370'" \
-I "//'thlqual.SCSQHPPS'" imqwrld.cpp \
"//'thlqual.SCSQDEFS(IMQS23DM)'" "//'thlqual.SCSQDEFS(IMQB23DM)'"
```

如需 TSO **oput** 指令的相關資訊，請參閱 z/OS UNIX Command Reference。

您也可以使用 make 公用程式來簡化建置 C++ 程式。 以下是建置 HELLO WORLD C++ 範例程式的範例 make 檔。 它會區隔編譯及鏈結階段。 在執行 make 之前，請如步驟 第 463 頁的『3』 中所示設定環境。

```
flags = -I "//'thlqual.SCSQC370'" -I "//'thlqual.SCSQHPPS'"
decks = "//'thlqual.SCSQDEFS(IMQS23DM)'" "//'thlqual.SCSQDEFS(IMQB23DM)'"

imqwrld: imqwrld.o
```

[6] 您可以與 「預先鏈結物件程式碼」中列出的任何側邊欄鏈結，以在三個環境中的任何一個環境中執行 z/OS UNIX : 第 462 頁的『在 z/OS Batch、RRS Batch 和 CICS 上建置 C++ 程式』

```
    c++ -o imqwrld imqwrld.o $(decks)

imqwrld.o: imqwrld.cpp
    c++ -c -o imqwrld $(flags) imqwrld.cpp
```

如需使用 make 的相關資訊，請參閱 z/OS UNIX System Services 程式設計工具 。

執行中

1. 登入 z/OS UNIX Shell ， 並切換至您建置範例的目錄。
2. 設定 STEPLIB 環境變數以包含 IBM MQ 資料集:

```
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQLOAD
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQAUTH
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQANLE
```

3. 執行範例:

```
/u/fred/sample:> ./imqwrld
```

開發 .NET 應用程式

IBM MQ classes for .NET 容許 .NET 應用程式以 IBM MQ MQI client 身分連接至 IBM MQ ， 或直接連接至 IBM MQ 伺服器。

如果您有應用程式使用 Microsoft .NET Framework ， 且想要利用 IBM MQ 的機能，則必須使用 IBM MQ classes for .NET。 如需相關資訊，請參閱 第 469 頁的『正在安裝 IBM MQ classes for .NET Framework』。

從 IBM MQ 9.1.1 開始， IBM MQ 支援 Windows 環境中應用程式的 .NET Core 。 如需相關資訊，請參閱 第 465 頁的『正在安裝 IBM MQ classes for .NET』。

從 IBM MQ 9.1.2 開始， IBM MQ 支援 Linux 環境中應用程式的 .NET Core 。

從 IBM MQ 9.1.4 開始， IBM MQ .NET 受管理應用程式可以自動平衡叢集佇列管理程式之間的連線。 同時支援 .NET Framework 和 .NET Standard 程式庫。 如需相關資訊，請參閱 關於統一叢集 及 自動應用程式平衡。

物件導向 IBM MQ .NET 介面不同於 MQI 介面，因為它使用物件方法，而不是使用 MQI 動詞。

程序化 IBM MQ 應用程式設計介面是以動詞 (例如下列清單中的動詞) 為建置基礎:

```
       MQCONN, MQDISC, MQOPEN, MQCLOSE,
       MQINQ, MQSET, MQGET, MQPUT, MQSUB
```

這些動詞全部都採用它們要在其上操作之 IBM MQ 物件的控點作為參數。 因為 .NET 是物件導向，所以 .NET 程式設計介面會開啟這一輪。 您的程式由一組 IBM MQ 物件組成，您可以對這些物件呼叫方法來處理這些物件。 您可以使用 .NET 支援的任何語言撰寫程式。

當您使用程序化介面時，您可以使用呼叫 MQDISC (*Hconn*， CompCode， Reason) 來切斷佇列管理程式的連線，其中 *Hconn* 是佇列管理程式的控點。

在 .NET 介面中，佇列管理程式由類別 MQQueueManager 的物件代表。 您可以在該類別上呼叫 Disconnect () 方法來切斷佇列管理程式的連線。

```
// declare an object of type queue manager
MQQueueManager queueManager=new MQQueueManager();
...
// do something...
...
// disconnect from the queue manager
queueManager.Disconnect();
```

IBM MQ classes for .NET 是一組類別，可讓 .NET 應用程式與 IBM MQ 互動。 它們代表您應用程式使用的 IBM MQ 的各種元件，例如佇列管理程式、佇列、通道及訊息。 如需這些類別的詳細資料，請參閱 IBM MQ .NET 類別和介面。

您必須先安裝 .NET Framework，才能編譯您所撰寫的任何應用程式。 如需安裝 IBM MQ classes for .NET 及 .NET Framework 的指示，請參閱 第 469 頁的『正在安裝 IBM MQ classes for .NET Framework』。

相關概念

技術概觀

第 5 頁的『開發適用於 IBM MQ 的應用程式』
您可以開發應用程式來傳送及接收訊息，以及管理佇列管理程式和相關資源。 IBM MQ 支援以許多不同語言及架構撰寫的應用程式。

相關工作

與 IBM 支援中心聯絡

疑難排解 IBM MQ .NET 問題

第 1063 頁的『使用 IBM MQ 開發 Microsoft Windows Communication Foundation 應用程式』
IBM MQ 的 Microsoft Windows Communication Foundation (WCF) 自訂通道會在 WCF 用戶端與服務之間傳送及接收訊息。

第 516 頁的『開發 XMS .NET 應用程式』
IBM MQ Message Service Client (XMS) for .NET (XMS .NET) 提供稱為 XMS 的應用程式設計介面 (API)，其具有與 Java Message Service (JMS) 相同的介面集 API。 IBM MQ Message Service Client (XMS) for .NET 包含 XMS 的完全受管理實作，可供任何符合 .NET 標準的語言使用。

Windows Linux 正在安裝 IBM MQ classes for .NET

IBM MQ classes for .NET(包括範例) 與 IBM MQ 一起安裝在 Windows 和 Linux 上

必備項目及安裝

從 IBM MQ 9.2.0 開始，使用 .NET Standard 建置的 IBM MQ .NET 用戶端程式庫可在 Windows 和 Linux 上使用。 若要執行 IBM MQ classes for .NET Standard，您必須安裝 Microsoft .NET Core。 Microsoft.NET Core 3.1 是執行 IBM MQ classes for .NET Standard 所需的最低版本。

V 9.3.0 V 9.3.0 從 IBM MQ 9.3.0 開始， IBM MQ 支援使用 IBM MQ classes for .NET Standard 的 .NET 6 應用程式。 如果您使用 .NET Core 3.1 應用程式，則可以在 `csproj` 檔案中進行小型編輯來執行此應用程式，將 `targetframeworkversion` 設定為 `"net6.0"`，而不需要任何重新編譯。

V 9.3.1 IBM MQ 9.3.1 提供針對 .NET 6 建置的 IBM MQ .NET 用戶端程式庫作為目標架構。 從 IBM MQ 9.3.1 開始， Microsoft .NET 6.0 是使用 IBM MQ 程式庫 (使用 .NET 6 作為目標架構來建置) 來執行應用程式所需的最低版本。

V 9.3.1 從 IBM MQ 9.3.1 開始，使用 .NET Standard 建置的 IBM MQ .NET 用戶端程式庫可在新資料夾 `netstandard2.0` 下使用，而使用 .NET 6 建置的 IBM MQ .NET 用戶端程式庫可在 Windows 上的 *MQ_INSTALLATION_PATH*`/bin` 及 Linux 上的 *MQ_INSTALLATION_PATH*`/lib64` 下使用。

依預設，最新版本的 IBM MQ classes for .NET 會隨 *Java* 和 *.NET* 傳訊及 *Web* 服務 特性的標準 IBM MQ 安裝一起安裝。

Windows 如需 Windows 上必備項目及安裝的相關資訊:

- 請參閱 IBM MQ classes for .NET 的需求，以取得執行 IBM MQ classes for .NET 的必備軟體。
- 如需安裝指示，請參閱 在 Windows 上安裝 IBM MQ 伺服器 或 在 Windows 系統上安裝 IBM MQ 用戶端 。

Linux 如需 Linux 上必備項目及安裝的相關資訊:

- 請參閱 IBM MQ classes for .NET 的需求，以取得執行 IBM MQ classes for .NET 的必備軟體。
- 如需 rpm 安裝指示，請參閱 在 Linux 系統上安裝 IBM MQ 用戶端。

- 若為 Linux Ubuntu，使用 Debian 套件，請參閱 在 Linux 系統上安裝 IBM MQ 用戶端。

IBM MQ classes for .NET Standard 程式庫 amqmdnetstd.dll 可從 NuGet 儲存庫下載。 如需相關資訊，請參閱第 468 頁的『從 NuGet 儲存庫下載 IBM MQ classes for .NET』。

amqmdnetstd.dll 磁帶庫

V 9.3.1 從 IBM MQ 9.3.1 開始， amqmdnetstd.dll 媒體庫位於下列位置

使用 .NET Standard 2.0 作為目標架構建置的程式庫

- Windows 在 Windows 上: *MQ_INSTALLATION_PATH*\bin\netstandard2.0。
- Linux 在 Linux 上: *MQ_INSTALLATION_PATH*\lib64\netstandard2.0。

Deprecated 這些程式庫已淘汰， IBM 打算在未來版本中移除它們。

使用 .NET 6 作為目標架構建置的程式庫

- Windows 在 Windows 上: *MQ_INSTALLATION_PATH*\bin。 範例應用程式安裝在 *MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base 中。
- Linux 在 Linux 上: *MQ_INSTALLATION_PATH*\lib64。 .NET 範例位於 *MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base 中。

LTS 對於 IBM MQ 9.3.0 Long Term Support， amqmdnetstd.dll 媒體庫位於下列位置:

- Windows 在 Windows 上: *MQ_INSTALLATION_PATH*\bin。 範例應用程式安裝在 *MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base 中。
- Linux 在 Linux 上: *MQ_INSTALLATION_PATH*/lib64 path。 .NET 範例位於 *MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base 中。

小心: V 9.3.1 Deprecated 從 IBM MQ 9.3.1 開始，使用 .NET Standard 2.0 作為目標架構建置的 IBM MQ .NET 用戶端程式庫已淘汰，且在編譯時期，參照這些程式庫的應用程式會擲出警告 CS0618 。

LTS Stabilized 仍提供 .NET Framework 的 amqmdnet.dll 程式庫，但此程式庫已穩定; 也就是說，不會在其中引進任何新特性。 對於任何最新特性，您必須移轉至 amqmdnetstd.dll 程式庫。 不過，您可以在 IBM MQ 9.1 或更新版本 Long Term Support 或 Continuous Delivery 版次上繼續使用 amqmdnet.dll 程式庫。

V 9.3.1 如果使用 amqmdnetstd.dll 或 amqmxmsstd.dll 從低於 IBM MQ 9.3.1 的版本編譯 .NET Framework 應用程式，且使用 .NET 6 型 IBM MQ 用戶端程式庫執行相同的應用程式，則 .NET 會擲出下列 FileLoad **異常狀況** 類型的異常狀況:

```
捕捉到異常狀況: System.IO.FileLoadException: 無法載入檔案或組件
'amqmdnetstd , Version =x.x.x.x , Culture=neutral , PublicKeyToken=23d6cb914eeaac0e' 或
其中一個相依關係。 所找到組件的資訊清單定義不符合
組件參照。 (來自 HRESULT 的異常狀況: 0x80131040)

檔名: 'amqmdnetstd , Version =x.x.x.x , Culture=neutral ,
PublicKeyToken=23d6cb914eeaac0e'
```

若要解決此錯誤，必須將 *MQ_INSTALLATION_PATH*/bin/netstandard2.0 中存在的程式庫複製到 .NET Framework 應用程式執行所在的目錄。

dspmqver 指令

您可以使用 **dspmqver** 指令來顯示 .NET Core 元件的版本和建置資訊。

IBM MQ classes for .NET Framework 與 IBM MQ classes for .NET (.NET Standard 與 .NET 6 程式庫) 之間的特性比較

下表列出相較於 IBM MQ classes for .NET (.NET Standard 和 .NET 6 程式庫) 的特性， IBM MQ classes for .NET Framework 的特性。

表 *76: IBM MQ classes for .NET Framework* 與 *IBM MQ classes for .NET (.NET Standard* 與 *.NET 6* 程式庫) 之間的差異

特性	**IBM MQ classes for .NET Framework**	**IBM MQ classes for .NET (.NET Standard 和 .NET 6 程式庫)**
類別名稱 (API)	所有類別都在每個網路中保持相同。	所有類別都在每個網路中保持相同。
作業系統	Windows	Windows Docker 化的儲存器 Linux macOS
`app.config` 檔案（用於在可重新配送用戶端中啟用追蹤的配置檔案）	`app.config` 檔案用來啟用可重新配送套件及獨立式 IBM MQ .NET 用戶端的追蹤。 如需您用於追蹤之變數的相關資訊，請參閱 使用應用程式配置檔追蹤 IBM MQ classes for .NET Framework 用戶端，包括 **MQTRACEPATH** 及 **MQTRACELEVEL**。	不支援 `app.config`。 使用環境變數。
追蹤	若為 IBM MQ 的完整用戶端安裝，您可以使用 **strmqtrc** 指令來啟用 IBM MQ classes for .NET Framework 的追蹤。 對於可重新配送的用戶端，也會使用 `app.config` 檔案來啟用追蹤。 如需相關資訊，請參閱 追蹤 IBM MQ .NET 應用程式。 V 9.3.3 從 IBM MQ 9.3.3 開始，您可以使用 `mqclient.ini` 檔案並設定「追蹤」段落的適當內容，來啟用及停用追蹤。 您也可以使用 `mqclient.ini` 檔案來動態啟用及停用追蹤。 如需相關資訊，請參閱 使用 mqclient.ini 追蹤 IBM MQ .NET 應用程式。	環境變數 **MQDOTNET_TRACE_ON** 用來啟用可重新配送用戶端的追蹤。 等於及小於 0 的值不會啟用追蹤。 值 1 會啟用預設層次追蹤。 大於 1 的值會啟用詳細追蹤。 將此環境變數設為任何其他值 (如字串) 不會啟用追蹤。 請參閱 使用環境變數追蹤 IBM MQ .NET 應用程式。 **MQDOTNET_TRACE_ON** 環境變數會檢查 IBM MQ 追蹤目錄是否可用。 如果追蹤目錄可用，則會在追蹤目錄中產生追蹤檔。 不過，如果未安裝 IBM MQ，則會將追蹤檔複製到現行工作目錄。 其他用於 IBM MQ classes for .NET Framework 的環境變數 (包括 **MQERRORPATH**、 **MQLOGLEVEL**、 **MQSERVER** 等) 也可以相同方式使用及運作。 V 9.3.3 從 IBM MQ 9.3.3 開始，您可以使用 `mqclient.ini` 檔案並設定「追蹤」段落的適當內容，來啟用及停用追蹤。 您也可以使用 `mqclient.ini` 檔案來動態啟用及停用追蹤。 如需相關資訊，請參閱 使用 mqclient.ini 追蹤 IBM MQ .NET 應用程式。

表 76: *IBM MQ classes for .NET Framework* 與 *IBM MQ classes for .NET (.NET Standard* 與 *.NET 6* 程式庫) 之間的差異 (繼續)

特性	IBM MQ classes for .NET Framework	IBM MQ classes for .NET (.NET Standard 和 .NET 6 程式庫)
傳輸模式	受管理、未受管理和連結	受管理的
TLS	Windows 金鑰儲存庫用於儲存憑證。	Windows 在 Windows 上，金鑰儲存庫必須用於儲存憑證。 允許的值為 *USER 或 *SYSTEM。 根據輸入，IBM MQ .NET 用戶端會查看現行使用者或系統層面的 Windows 金鑰儲存庫。 Linux 在 Linux 上，建議使用 X509Store 類別來安裝憑證，.NET Core 憑證安裝到下列位置：".dotnet/corefx/cryptography/x509stores"。
CCDT	支援	受支援，而且 CCDT 路徑的設定和 .NET Framework Framework 類別相同。
用戶端自動重新連接	支援	支援
分散式交易	支援	不支援
將動態鏈結程式庫 (dll) 安裝到廣域組合語言快取 (GAC) 中	將 Dll 作為 IBM MQ 安裝的一部分安裝在 GAC 中。	Dll 不作為 IBM MQ 安裝的一部分安裝在 GAC 中。

註: Windows Windows 安全 ID (SID):

IBM MQ classes for .NET (.NET Standard 和 .NET 6 程式庫) 不支援網域層次鑑別。 登入的使用者 ID 用於鑑別。

在 macOS 上開發 IBM MQ .NET Core 應用程式

macOS

IBM MQ .NET Core 應用程式可以在 macOS 上開發。

IBM MQ .NET 程式庫未隨附於 macOS 工具箱，因此您必須將它們從 Windows 或 Linux IBM MQ 用戶端複製到 macOS。 然後，您可以使用這些程式庫，在 macOS 上開發 IBM MQ .NET Core 應用程式。

開發之後，可以在 Windows 或 Linux 環境上支援執行這些應用程式。

相關概念

第 469 頁的『正在安裝 IBM MQ classes for .NET Framework』
IBM MQ classes for .NET Framework(包括範例) 與 IBM MQ 一起安裝。 Windows 上具有 Microsoft.NET Framework 的必要條件。

第 519 頁的『正在安裝 IBM MQ classes for XMS .NET』
IBM MQ classes for XMS .NET(包括範例) 與 IBM MQ 一起安裝在 Windows 和 Linux 上。

Windows Linux 從 NuGet 儲存庫下載 IBM MQ classes for .NET

IBM MQ classes for .NET 可從 NuGet 儲存庫下載，因此 .NET 開發人員可以輕鬆使用它們。

關於這項作業

NuGet 是 Microsoft 開發平台的套件管理程式，包括 .NET。 NuGet 用戶端工具提供產生及耗用套件的能力。 NuGet 套件是具有 `.nupkg` 副檔名的單一壓縮檔，其中包含已編譯的程式碼 (DLL)、與該程式碼相關的其他檔案，以及包含套件版本號碼之類資訊的敘述性資訊清單。

您可以從 NuGet Gallery 下載包含 `amqmdnetstd.dll` 檔案庫的 `IBMMQDotnetClient` NuGet 套件，這是所有套件作者及消費者使用的中央套件儲存庫。

註: V 9.3.1 從 IBM MQ 9.3.1 開始， NuGet 套件包含使用 .NET Standard 2.0 和 .NET 6 作為目標架構所建置的程式庫。 .NET Standard 2.0 的程式庫位於 `netstandard2.0` 資料夾之下， .NET 6 的程式庫位於 `net6.0` 資料夾之下。

下載 `IBMMQDotnetClient` 套件有三種方式:

- 使用 Microsoft Visual Studio。 NuGet 是作為 Microsoft Visual Studio 延伸來配送。 從 Microsoft Visual Studio 2012 開始，依預設會預先安裝 NuGet 。
- 從指令行使用 NuGet Package Manager 或 .NET CLI。
- 使用 Web 瀏覽器。

對於可重新配送的套件，您可以使用環境變數 **MQDOTNET_TRACE_ON** 來啟用追蹤。

程序

- 若要使用 Microsoft Visual Studio 中的「套件管理程式」使用者介面來下載 `IBMMQDotnetClient` 套件，請完成下列步驟:

  a) 用滑鼠右鍵按一下 .NET 專案，然後按一下 **管理 Nuget 套件**。

  b) 按一下 **瀏覽** 標籤並搜尋 `"IBMMQDotnetClient"`。

  c) 選取套件，然後按一下 **安裝**。

  在安裝期間，「套件管理程式」會以主控台陳述式的形式提供進度資訊。
- 若要從指令行下載 `IBMMQDotnetClient` 套件，請選擇下列其中一個選項:
  - 使用 NuGet Package Manager ，輸入下列指令:

    ```
    Install-Package IBMMQDotnetClient -Version 9.1.4.0
    ```

    在安裝期間，「套件管理程式」會以主控台陳述式的形式提供進度資訊。 您可以將輸出重新導向至日誌檔。
  - 使用 .NET CLI ，輸入下列指令:

    ```
    dotnet add package IBMMQDotnetClient --version 9.1.4
    ```
- 使用 Web 瀏覽器，從 https://www.nuget.org/packages/IBMMQDotnetClient 下載 `IBMMQDotnetClient` 套件。

相關概念

IBM MQ Client for .NET 授權資訊

相關工作

第 522 頁的『從 NuGet 儲存庫下載 IBM MQ classes for XMS .NET』

IBM MQ classes for XMS .NET 可從 NuGet 儲存庫下載，因此 .NET Developers 可以輕鬆使用它們。

Windows 正在安裝 IBM MQ classes for .NET Framework

IBM MQ classes for .NET Framework(包括範例) 與 IBM MQ 一起安裝。 Windows 上具有 Microsoft.NET Framework 的必要條件。

依預設，在 *Java* 和 *.NET* 傳訊及 *Web* 服務 特性中安裝標準 IBM MQ 時，會安裝最新版本的 IBM MQ classes for .NET Framework 。 如需安裝指示，請參閱 在 Windows 上安裝 IBM MQ 伺服器 或 在 Windows 系統上安裝 IBM MQ 用戶端。

V 9.3.0 V 9.3.0 從 IBM MQ 9.3.0 開始，若要執行 IBM MQ classes for .NET Framework ，您必須安裝 Microsoft.NET Framework V4.7.2 或更新版本。 這是從 IBM MQ 9.2 開始的變更，其中最低必要版本為 V4.6.2。

V 9.3.0 V 9.3.0 透過在應用程式的 `app.config` 檔案中新增下列標籤，可以執行使用 Microsoft.NET Framework V3.5 編譯的現有應用程式，而無需重新編譯:

```
<configuration>
 <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2"/>
 </startup>
</configuration>
```

註: 如果在安裝 IBM MQ 之前未安裝 Microsoft .NET Framework V4.7.2 或更高版本，則 IBM MQ 產品安裝會繼續進行，且不會發生錯誤，但 IBM MQ classes for .NET 無法使用。 如果在安裝 IBM MQ 之後安裝.NET Framework ，則必須透過執行 *WMQInstallDir*`\bin\amqiRegisterdotNet.cmd` Script 來登錄 IBM MQ.NET 組件，其中 *WMQInstallDir* 是 IBM MQ 的安裝目錄。 此 Script 會在「廣域組件快取 (GAC)」中安裝必要的組件。 會在 %TEMP% 目錄中建立一組記錄所採取動作的 `amqi*.log` 檔案。 如果 .NET 從舊版 (例如，從 .NET V3.5) 升級至 V4.7.2 或更高版本，則不需要重新執行 `amqiRegisterdotNet.cmd` Script。

在多重安裝環境中，如果您先前已安裝 IBM MQ classes for .NET 作為支援套件，則除非先解除安裝支援套件，否則無法安裝 IBM MQ 。 隨 IBM MQ 一起安裝的 IBM MQ classes for .NET 特性包含與支援套件相同的功能。

也會提供範例應用程式 (包括原始檔); 請參閱 第 471 頁的『.NET 的範例應用程式』。

如需搭配使用 Microsoft WCF 的 IBM MQ 自訂通道與 .NET 的相關資訊，請參閱 第 1063 頁的『使用 IBM MQ 開發 Microsoft Windows Communication Foundation 應用程式』

相關概念

第 465 頁的『正在安裝 IBM MQ classes for .NET』
IBM MQ classes for .NET(包括範例) 與 IBM MQ 一起安裝在 Windows 和 Linux 上

相關工作

追蹤 IBM MQ .NET 應用程式

用於將 IBM MQ classes for .NET 連接至佇列管理程式的選項

將 IBM MQ classes for .NET 連接至佇列管理程式有三種模式。 請考量最適合您需求的連線類型。

用戶端連結連線

若要使用 IBM MQ classes for .NET 作為 IBM MQ MQI client，您可以使用 IBM MQ MQI client 在 IBM MQ 伺服器機器上或在個別機器上安裝它。 用戶端連結連線可以使用 XA 或非 XA 交易

伺服器連結連線

在伺服器連結模式中使用時， IBM MQ classes for .NET 會使用佇列管理程式 API ，而不是透過網路進行通訊。 這會為 IBM MQ 應用程式提供比使用網路連線更好的效能。

如果要使用連結連線，您必須在 IBM MQ 伺服器上安裝 IBM MQ classes for .NET 。

受管理用戶端連線

在此模式下建立的連線會以 IBM MQ 用戶端身分連接至在本端或遠端機器上執行的 IBM MQ 伺服器。

在此模式下連接的 IBM MQ classes for .NET 會保留在 .NET 受管理程式碼中，且不會呼叫原生服務。 如需受管理程式碼的相關資訊，請參閱 Microsoft 文件。

使用受管理用戶端有一些限制。 如需這些的相關資訊，請參閱 第 484 頁的『受管理用戶端連線』。

.NET 的範例應用程式

若要執行您自己的 .NET 應用程式，請使用驗證程式的指示，以您的應用程式名稱取代範例應用程式。

提供下列範例應用程式:

- 放置訊息應用程式
- 取得訊息應用程式
- 'hello world' 應用程式
- 發佈/訂閱應用程式
- 使用訊息內容的應用程式

所有這些範例應用程式都以 C# 語言提供，部分也以 C++ 及 Visual Basic 提供。 您可以使用 .NET 支援的任何語言來撰寫應用程式。

"Put message" 程式 SPUT (nmqsput.cs， mmqsput.cpp， vmqsput.vb)
此程式顯示如何將訊息放入指名的佇列。 程式有三個參數:

- 佇列的名稱 (必要)，例如 SYSTEM.DEFAULT.LOCAL.QUEUE
- 佇列管理程式的名稱 (選用)
- 通道的定義 (選用)，例如 SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

如果未提供佇列管理程式名稱，則佇列管理程式會預設為預設本端佇列管理程式。 如果已定義通道，則其格式與 MQSERVER 環境變數相同。

"Get message" 程式 SGET (nmqsget.cs， mmqsget.cpp， vmqsget.vb)
此程式顯示如何從具名佇列取得訊息。 程式有三個參數:

- 佇列的名稱 (必要)，例如 SYSTEM.DEFAULT.LOCAL.QUEUE
- 佇列管理程式的名稱 (選用)
- 通道的定義 (選用)，例如 SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

如果未提供佇列管理程式名稱，則佇列管理程式會預設為預設本端佇列管理程式。 如果已定義通道，則其格式與 MQSERVER 環境變數相同。

"Hello World" 程式 (nmqwrld.cs， mmqwrld.cpp， vmqwrld.vb)
此程式顯示如何放置及取得訊息。 程式有三個參數:

- 佇列的名稱 (選用)，例如 SYSTEM.DEFAULT.LOCAL.QUEUE 或 SYSTEM.DEFAULT.MODEL.QUEUE
- 佇列管理程式的名稱 (選用)
- 通道定義 (選用)，例如 SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

如果未提供佇列名稱，則名稱預設為 SYSTEM.DEFAULT.LOCAL.QUEUE。 如果未提供佇列管理程式名稱，則佇列管理程式會預設為預設本端佇列管理程式。

"Publish/subscribe" 程式 (MQPubSubSample.cs)
此程式顯示如何使用 IBM MQ 發佈/訂閱。 它僅在 C# 中提供。 程式有兩個參數:

- 佇列管理程式的名稱 (選用)
- 通道定義 (選用)

「訊息內容」程式 (MQMessagePropertiesSample.cs)
此程式顯示如何使用訊息內容。 它僅在 C# 中提供。 程式有兩個參數:

- 佇列管理程式的名稱 (選用)
- 通道定義 (選用)

您可以編譯並執行這些應用程式來驗證安裝。

安裝位置

範例應用程式會根據撰寫它們的語言，安裝至下列位置。 *MQ_INSTALLATION_PATH* 代表 IBM MQ 安裝所在的高階目錄。

C#

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\nmqswrld.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\nmqsput.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\nmqsget.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\MQPubSubSample.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\MQMessagePropertiesSample.cs

受管理 C++

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\mmqswrld.cpp

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\mmqsput.cpp

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\mmqsget.cpp

Visual Basic

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\vmqswrld.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\vmqsput.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\vmqsget.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\xmqswrld.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\xmqsput.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\xmqsget.vb

建置範例應用程式

為了建置範例應用程式，會提供每一種語言的批次檔。

C#

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\bldcssamp.bat

bldcssamp.bat 檔會針對每一個範例包含一行，建置此範例程式只需要這一行:

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib: MQ_INSTALLATION_PATH\bin
/out:nmqwrld.exe nmqwrld.cs
```

受管理 C++

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\bldmcpsamp.bat

bldmcpsamp.bat 檔針對每一個範例各包含一行，這是建置此範例程式所需要的一切:

```
cl /clr:oldsyntax MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

如果您想要在 Microsoft Visual Studio 2003/.NET SDKv1.1 上編譯這些應用程式，請取代編譯指令:

```
cl /clr:oldsyntax MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

具有

```
cl /clr MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

Visual Basic

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\bldvbsamp.bat

bldvbsamp.bat 檔會針對每一個範例包含一行，這是建置此範例程式所需要的一切:

```
vbc /r:System.dll /r: MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:vmqwrld.exe vmqwrld.vb
```

IBM MQ 與 Microsoft .NET Core 搭配使用的範例

從 IBM MQ 9.2.0 開始， IBM MQ 在 Windows 環境中支援 .NET Core for IBM MQ .NET 應用程式。 依預設，IBM MQ classes for .NET Standard (包括範例) 會隨標準 IBM MQ 安裝一起安裝。

IBM MQ .NET 的範例應用程式安裝在 &MQINSTALL_PATH&/samp/dotnet/samples/cs/core/base 中。 也會提供 Script，可用來編譯範例。

您可以使用提供的 build.bat 檔案來建置範例。 Windows 上下列位置中的每一個範例都有一個 build.bat：

- MQ\tools\dotnet\samples\cs\core\base\SimpleGet
- MQ\tools\dotnet\samples\cs\core\base\SimplePut

Linux 從 IBM MQ 9.2.0 開始， IBM MQ 也支援 Linux 環境中應用程式的 Core。

如需使用 IBM MQ 與 Microsoft .NET Core 搭配的相關資訊，請參閱 第 465 頁的『正在安裝 IBM MQ classes for .NET』。

配置佇列管理程式以接受 TCP/IP 用戶端連線

配置佇列管理程式以接受來自用戶端的送入連線要求。

關於這項作業

此作業說明配置佇列管理程式以接受 TCP/IP 用戶端連線的基本步驟。 若為正式作業系統，在配置佇列管理程式時，您也必須考量安全含意。

程序

1. 定義伺服器連線通道:
   a. 啟動佇列管理程式。
   b. 定義稱為 NET.CHANNEL:

```
DEF CHL('NET.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ') +
DESCR('Sample channel for IBM MQ classes for .NET')
```

   重要: 此範例僅適用於沙盤推演環境，因為它不包含安全含意的任何考量。 若為正式作業系統，請考量使用 TLS 或安全結束程式。 如需相關資訊，請參閱 保護 IBM MQ 安全。
2. 啟動接聽器:

```
runmqlsr -t tcp [-m qmnqme ] [-p portnum ]
```

   註: 方括弧指出選用參數; 預設佇列管理程式不需要 *qmname*，如果您使用預設值 (1414)，則不需要埠號 *portnum*。

.NET 中的分散式交易

分散式交易或廣域交易可讓用戶端應用程式在一個交易中包含兩個以上網路系統上的數個不同資料來源。

在分散式交易中，交易管理程式會協調及管理兩個以上資源管理程式之間的交易。

交易可以是單階段或兩階段確定程序。 單一階段確定是一個處理程序，其中只有一個資源管理程式參與交易，而兩個階段確定處理程序是有多個資源管理程式參與交易的。 在兩段式確定程序中，交易管理程式會傳送準備呼叫，以檢查所有資源管理程式是否都已準備好要確定。 當它收到所有資源管理程式的確認通知時，即會發出確定呼叫。 否則，會對整個交易進行回復。 如需詳細資料，請參閱 交易管理及支援 。 資源管理程

式應將其參與交易的情況通知交易管理人員。 當資源管理程式通知交易管理程式其參與時，當交易即將確定或回復時，資源管理程式會從交易管理程式取得回呼。

IBM MQ .NET 類別已在未受管理及伺服器連結模式連線中支援分散式交易。 在這些模式中， IBM MQ .NET 類別會將其所有呼叫委派給 C 延伸交易用戶端，該用戶端代表 .NET 管理交易處理。

IBM MQ.NET 類別現在支援受管理模式中的分散式交易，其中 IBM MQ .NET 類別使用分散式交易支援的 System.Transactions 名稱空間。 System.Transactions 基礎架構支援在所有資源管理程式 (包括 IBM MQ) 中起始的交易，使交易式程式設計變得簡單而有效率。 IBM MQ .NET 應用程式可以使用 .NET 隱含交易程式設計或明確交易程式設計模型來放置及取得訊息。 在隱含交易中，由應用程式建立交易界限，決定何時確定、回復 (針對明確交易) 或完成交易。 在明確交易中，您必須明確指定是否要確定、回復及完成交易。

IBM MQ.NET 使用 Microsoft 分散式交易協調程式 (MS DTC) 作為交易管理程式，可協調及管理多個資源管理程式之間的交易。 IBM MQ 用來作為資源管理程式。 請注意，您無法將 TLS 與 XA 交易搭配使用。 您必須使用 CCDT。 如需相關資訊，請參閱 搭配使用延伸交易式用戶端與 TLS 通道。

IBM MQ.NET 遵循「X/Open 分散式交易處理 (DTP)」模型。 「X/Open 分散式交易處理」模型是由供應商聯盟「開放式群組」所提出的分散式交易處理模型。 此模型是交易處理及資料庫網域中大部分商業供應商的標準。 大部分商業交易管理產品支援 X/DTP 模型。

交易模式

- 第 475 頁的『.NET 受管理模式中的分散式交易』
- 未受管理模式的分散式交易

在各種實務中協調交易

- 連線可能參與數個交易，但在任何時間點只有一個交易處於作用中。
- 在交易期間， MQQueueManager。允許中斷通話。 在此情況下，會要求交易回復。
- 在交易期間，允許使用 MQQueue.Close 或 MQTopic.Close 呼叫。 在此情況下，會要求交易回復。
- 交易界限由應用程式建立，決定何時確定、回復 (針對明確交易) 或完成 (針對隱含交易) 交易。
- 在對佇列或主題呼叫發出 Put 或 Get 呼叫之前，如果用戶端應用程式在交易期間中斷，且發生非預期的錯誤，則會回復交易，並擲出 MQException。
- 如果在佇列或主題呼叫「放置」或「取得」呼叫期間傳回 MQCC_FAILED 原因碼，則會擲出 MQException ，原因碼為，並捲動交易。 如果交易管理程式已發出準備呼叫，則 IBM MQ .NET 會強制回復交易，以傳回準備要求。 然後，交易管理程式 DTC 會導致現行工作回復現行環境交易中的所有資源管理程式。
- 在涉及多個資源管理程式的交易期間，如果某個環境原因導致「放置」或「取得」呼叫無限期當掉，則交易管理程式會等待到規定的時間。 在逾時之後，它會導致回復現行環境交易中所有資源管理程式的所有現行工作。 如果在準備階段期間發生此無限期等待，則交易管理程式可能會逾時，或對資源發出不確定的呼叫，在此情況下會回復交易。
- 使用交易的應用程式必須在 SYNC_POINT 下放置或取得訊息。 如果在非 SYNC_POINT 下的交易式環境定義下發出訊息 Put 或 Get 呼叫，則呼叫會失敗，原因碼為 MQRC_UNIT_OF_WORK_NOT_STARTED。

使用 Microsoft.NET System.Transactions 名稱空間的「受管理用戶端」與「未受管理用戶端」交易支援之間的行為差異

巢狀交易在另一個 TransactionScope 內具有 TransactionScope

- IBM MQ .NET 完全受管理用戶端不支援巢狀 TransactionScope
- IBM MQ .NET 未受管理的用戶端不支援巢狀 TransactionScope

來自 System.Transactions 的相依交易

- IBM MQ .NET 完全受管理用戶端不支援 System.Transactions 所提供的相依交易機能。
- IBM MQ .NET 未受管理的用戶端不支援 System.Transactions 所提供的相依交易機能。

產品範例

產品範例 SimpleXAPut 和 SimpleXAGet 可在 WebSphere MQ\tools\dotnet\samples\cs\base 下取得。 範例是 C# 應用程式，示範在分散式交易 (使用 SystemTransactions 名稱空間) 下使用 MQPUT 及 MQGET。 如需這些範例的相關資訊，請參閱 第 477 頁的『在 TransactionScope 內建立簡式放置及取得訊息』。

.NET 受管理模式中的分散式交易

對於受管理模式中的分散式交易支援， IBM MQ .NET 類別使用 System.Transactions 名稱空間。 在受管理模式中， MS DTC 會協調及管理在交易中列入的所有伺服器之間的分散式交易。

IBM MQ .NET 類別提供以 System.Transactions.Transaction 類別為基礎的明確程式設計模型，以及使用 System.Transactions.TransactionScope 類別 (其中交易由基礎架構自動管理) 的隱含程式設計模型。

隱含交易

下列程式碼片段說明 IBM MQ .NET 應用程式如何使用 .NET 隱含交易程式設計來放置訊息。

```
Using (TransactionScope scope = new TransactionScope ())
{
    Q.Put (putMsg,pmo);
    scope.Complete ();
}

Q.close();
qMgr.Disconect();}
```

隱含交易的程式碼流程說明

程式碼會建立 *TransactionScope* ，並將訊息置於範圍下。 然後，它會呼叫 完成 ，以通知交易協調程式完成交易。 現在，交易協調程式會發出 *prepare* 及 *commit* ，以完成交易。 如果偵測到問題，則會呼叫 回復 。

明確交易

下列程式碼說明 IBM MQ .NET 應用程式如何使用 .NET 明確交易程式設計模型來放置訊息。

```
MQQueueManager qMgr = new MQQueuemanager ("MQQM);
MQQueue Q = QMGR.AccessQueue("Q", MQC.MQOO_OUTPUT+MQC.MQOO_INPUT_SHARED);
MQPutMessageOptions pmo = new MQPutMessageOptions();
pmo.Options = MQC.MQPMO_SYNCPOINT;
MQMessage putMsg1 = new MQMessage();
Using(CommittableTransaction tx = new CommittableTransaction()){
Transaction.Current = tx;
    try
    {
    Q.Put(MSG,pmo);
    tx.commit();
    }
    catch(Exception)
    {tx.rollback();}
    }

Q.close();
qMgr.Disconnect();
}
```

明確交易的程式碼流程說明

程式碼片段會使用 *CommitableTransaction* 類別來建立交易。 它會將訊息放在該範圍之下，然後明確呼叫 *commit* 來完成交易。 如果有任何問題，則會呼叫 *rollback* 。

處於 .NET 未受管理模式的分散式交易

IBM MQ.NET 類別支援使用延伸交易用戶端及 COM + /MTS 作為交易協調程式的未受管理連線 (用戶端) ，並使用隱含或明確的交易程式設計模型。 在未受管理模式中， IBM MQ .NET 類別會將其所有呼叫委派給代表 .NET 管理交易處理的 C 延伸交易用戶端。

交易處理由外部交易管理程式控制，在交易管理程式的 API 控制下協調廣域工作單元。 MQBEGIN、MQCMIT 及 MQBACK 動詞無法使用。 IBM MQ .NET 類別透過其未受管理的傳輸模式 (C 用戶端) 來公開此支援。 請參閱 配置符合 XA 標準的交易管理程式

MTS 發展為交易處理 (TP) 系統，以在 Windows NT 上提供 CICS、Tuxedo 及其他平台所提供的相同特性。 安裝 MTS 時，會將個別服務新增至 Windows NT，稱為 Microsoft 分散式交易協調程式 (MSDTC)。 MSDTC 會協調跨越個別資料儲存庫或資源的交易。 若要運作，它需要每一個資料儲存庫實作自己的專有資源管理程式。

透過實作介面 (專有資源管理程式介面)， IBM MQ 可在其中管理將 DTC XA 呼叫對映至 IBM MQ(X/Open) 呼叫，從而與 MSDTC 相容。 IBM MQ 扮演資源管理程式的角色。

當 COM + 之類的元件要求存取 IBM MQ 時，如果需要交易， COM 通常會檢查適當的 MTS 環境定義物件。 如果需要交易， COM 會通知 DTC，並自動啟動此作業的整數 IBM MQ 交易。 然後 COM 會透過 MQMTS 軟體處理資料，視需要放置及取得訊息。 在資料上的所有動作都結束之後，從 COM 取得的物件實例會呼叫 SetComplete 或 SetAbort 方法。 當應用程式發出 SetComplete 時，呼叫會向 DTC 發出信號，指出應用程式已完成交易，且 DTC 可以繼續進行兩階段確定程序。 然後 DTC 會向 MQMTS 發出呼叫，而 MQMTS 又會向 IBM MQ 發出呼叫，以確定或回復交易。

使用未受管理的用戶端撰寫 IBM MQ .NET 應用程式

如果要在 COM + 環境定義內執行， .NET 類別必須繼承自 System.EnterpriseServices.ServicedComponent。 建立使用已維修元件之組件的規則及建議如下:

註: 僅當您使用 System.EnterpriseServices 模式時，下列步驟才相關。

- 在 COM + 中啟動的類別和方法都必須是 public (沒有內部類別，也沒有 protected 或 static 方法)。
- 類別及方法屬性: TransactionOption 屬性指定類別的交易層次，即交易是已停用、受支援還是必要。 如果未擲出未處理的異常狀況， ExecuteUOW() 方法的 AutoComplete 屬性會指示 COM + 確定交易。
- 強式命名組件: 組件必須強式命名，並在「廣域組件快取 (GAC)」中登錄。 組件在 GAC 中登錄之後，會明確或延遲登錄在 COM + 中。
- 在 COM + 中登錄組件: 準備向 COM 用戶端公開組件。 然後使用組合登錄工具 regasm.exe 來建立類型程式庫。

```
regasm UnmanagedToManagedXa.dll
```

- 將組件登錄至 GAC `gacutil /i UnmanagedToManagedXa.dll`。
- 使用 .NET 服務安裝程式工具 regsvcs.exe，在 COM + 中登錄組件。 請參閱 regasm.exe: 所建立的類型程式庫:

```
Regsvcs /appname:UnmanagedToManagedXa /tlb:UnmanagedToManagedXa.tlb  UnmanagedToManagedXa.dll
```

- 組件會部署至 GAC，稍後會透過延遲登錄在 COM + 中登錄它。 第一次執行程式碼之後， .NET 架構會處理登錄。

下列各節說明使用 System.EnterpriseServices 模型及 System.Transactions with COM + 的程式碼流程範例:

使用 System.EnterpriseServices 模型的程式碼流程範例

```
using System;
using IBM.WMQ;
using IBM.WMQ.Nmqi;
using System.Transactions;
using System.EnterpriseServices;

namespace UnmanagedToManagedXa
{

[ComVisible(true)]
[System.EnterpriseServices.Transaction(System.EnterpriseServices.TransactionOption.Required)]
    public class MyXa : System.EnterpriseServices.ServicedComponent
    {

        public MQQueueManager QMGR = null;
        public MQQueueManager QMGR1 = null;
        public MQQueue QUEUE = null;
```

```
        public MQQueue QUEUE1 = null;
        public MQPutMessageOptions pmo = null;
        public MQMessage MSG = null;

        public MyXa()
        {
        }

        [System.EnterpriseServices.AutoComplete()]
        public void ExecuteUOW()
        {
            QMGR = new MQQueueManager("usemq");

            QUEUE = QMGR.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                                     MQC.MQOO_INPUT_SHARED +
                                     MQC.MQOO_OUTPUT +
                                     MQC.MQOO_BROWSE);

            pmo = new MQPutMessageOptions();
            pmo.Options = MQC.MQPMO_SYNCPOINT;
            MSG = new MQMessage();
            QUEUE.Put(MSG, pmo);
            QMGR.Disconnect();
        }
    }
}


public void RunNow()
{
   MyXa xa = new MyXa();
   xa.ExecuteUOW();
}
```

使用 System.Transactions 與 COM + 互動的程式碼流程範例

```
[STAThread]
public void ExecuteUOW()
{
Hashtable t1 = new Hashtable();
t1.Add(MQC.CHANNEL_PROPERTY, "SYSTEM.DEF.SVRCONN");
t1.Add(MQC.HOST_NAME_PROPERTY, "localhost");
t1.Add(MQC.PORT_PROPERTY, 1414);
t1.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_CLIENT);
TransactionOptions opts = new TransactionOptions();

using(TransactionScope scope = new  TransactionScope(TransactionScopeOption.RequiresNew,
                                opts,     EnterpriseServicesInteropOption.Full)
  {

        QMGR = new MQQueueManager("usemq", t1);
        QUEUE = QMGR.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                                          MQC.MQOO_INPUT_SHARED +
                                          MQC.MQOO_OUTPUT +
                                          MQC.MQOO_BROWSE);

         pmo = new MQPutMessageOptions();
         pmo.Options = MQC.MQPMO_SYNCPOINT;
         MSG = new MQMessage();
         QUEUE.Put(MSG, pmo);
         scope.Complete();
  }
 QMGR.Disconnect();
}
```

在 TransactionScope 內建立簡式放置及取得訊息

產品範例 C# 應用程式可在 IBM MQ 內取得。 這些簡式應用程式示範在 TransactionScope 內放置及取得訊息。 在作業結束時，您將能夠從佇列或主題中放置及取得訊息。

開始之前

對於 XA 交易， MSDTC 服務必須在執行中且已啟用。

關於這項作業

範例是簡式應用程式 SimpleXAPut 和 SimpleXAGet。 程式 SimpleXAPut 和 SimpleXAGet 是 IBM MQ 內可用的 C# 應用程式。 SimpleXAPut 示範使用 MQPUT，在「分散式交易」下使用 SystemTransactions 名稱空間。 SimpleXAGet 示範使用 MQGET，在「分散式交易」下使用 SystemTransactions 名稱空間。

SimpleXAPut 位於 MQ\tools\dotnet\samples\cs\base

程序

可以使用 tools\dotnet\samples\cs\base\bin 中的指令行參數來執行應用程式

```
SimpleXAPut.exe -d destinationURI [-h host -p port -l channel -tx transaction -tm mode -n
numberOfMsgs]
```

```
SimpleXAGet.exe -d destinationURI [-h host -p port -l channel -tx transaction -tm mode -n
numberOfMsgs]
```

其中參數為:

-destinationURI

這可以是佇列或主題。 若為佇列，請指定為 queue://queueName，若為主題，請指定為 topic://topicName。

-host

這可以是主機名稱，例如 localhost 或 IP 位址。

-port

執行佇列管理程式的埠。

-channel

正在使用的連線通道。 預設值為 SYSTEM.DEF.SVRCONN

-transaction

交易結果，例如確定或回復。

-mode

傳輸模式，例如受管理或未受管理。

-numberOfMsgs

訊息數。 預設值是 1。

範例

```
SimpleXAPut -d topic://T01 -h localhost -p 2345 -tx rollback -tm unmanaged
```

```
SimpleXAGet -d queue://Q01 -h localhost -p 2345 -tx rollback -tm unmanaged
```

在 IBM MQ .NET 中回復交易

本節說明在 IBM MQ .NET XA 中使用受管理模式回復交易的處理程序。

關於這項作業

在分散式交易處理中，交易可以順利完成，但在某些情況下，交易可能會因許多原因而失敗。 這些原因可能包括系統故障、硬體故障、網路錯誤、資料不正確或無效、應用程式錯誤或自然或人為災難。 無法防止交易

失敗。 分散式交易系統必須能夠處理這些失敗。 當錯誤發生時，它必須能夠偵測並更正錯誤。 此處理程序稱為「交易回復」。

「分散式交易處理」的重要層面是回復不完整或不確定的交易。 必須執行回復，因為特定交易的「工作單元」部分會保持鎖定，直到回復為止。 Microsoft.NET 從其 System.Transactions 類別庫提供選項來回復不完整/不確定的交易。 這項回復支援預期 Resource Manager 會維護交易日誌，並在需要時執行回復。

在 Microsoft .NET 交易回復模型中，「交易管理程式」(System.Transactions 及/或 Microsoft Distributed Transaction Coordinator (MS DTC)) 會起始、協調及控制交易回復。 OLE Tx 通訊協定 (Microsoft XA 通訊協定) 型「資源管理程式」提供選項，可讓您配置 DTC 來驅動、協調及控制它們的回復。 若要這樣做，「資源管理程式」必須使用原生介面向 MS DTC 登錄 XA_Switch。

XA_Switch 在 Resource Manager 中提供 XA 函數 (例如 xa_start、xa_end 及 xa_recover) 的進入點給「分散式交易協調程式」。

使用 Microsoft 分散式交易協調程式 (DTC) 進行回復:

Microsoft 分散式交易協調程式提供兩種回復處理程序。

冷回復

當 XA 資源管理程式連線開啟時，如果交易管理程式處理程序失敗，則會執行冷回復。 當交易管理程式重新啟動時，它會讀取交易管理程式日誌，並重新建立與 XA 資源管理程式的連線，然後起始回復。

熱回復

當交易管理程式與 XA 資源管理程式之間的連線因 XA 資源管理程式或網路失敗而失敗時，如果交易管理程式仍維持啟動，則會執行熱回復。 失敗之後，交易管理程式會定期嘗試重新連接 XA 資源管理程式。 當重新建立連線時，交易管理程式會起始 XA 回復。

System.Transactions 名稱空間提供以 MS DTC 作為交易管理程式之「分散式」交易的受管理實作。 它提供與 MS DTC 原生介面類似的功能，但在完全受管理的環境中。 唯一的差異在於交易回復。 System.Transactions 預期「資源管理程式」會自行驅動回復，然後與「交易管理程式 (MS DTC)」協調。 Resource Manager 必須要求回復特定未完成的交易，然後「交易管理程式」會接受它，並根據該特定交易的實際結果進行協調。

IBM MQ .NET 的交易回復程序

本節說明如何使用 IBM MQ .NET 類別來回復分散式交易。

概觀

若要回復不完整的交易，需要回復資訊。 交易回復資訊必須由資源管理程式記載至儲存體。 IBM MQ .NET 類別遵循類似的路徑。 交易回復資訊會記載至稱為 SYSTEM.DOTNET.XARECOVERY.QUEUE。

IBM MQ .NET 中的交易回復是兩個階段的程序:

1. 在 SYSTEM.DOTNET.XARECOVERY.QUEUE。
2. 使用 XA 監視器應用程式 WmqDotnetXAMonitor 來回復交易。

SYSTEM.DOTNET.XARECOVERY.QUEUE

SYSTEM.DOTNET.XARECOVERY.QUEUE 是一個系統佇列，用於保留不完整交易的交易回復資訊。 當建立佇列管理程式時，會建立此佇列。

對於每個交易，在準備階段期間，會將包含回復資訊的持續訊息新增至 SYSTEM.DOTNET.XARECOVERY.QUEUE。 如果確定呼叫成功，則會刪除訊息。

註: 您不得刪除 SYSTEM.DOTNET.XARECOVERY.QUEUE 佇列。

WMQDotnetXAMonitor 應用程式

IBM MQ .NET XA 監視器應用程式 WmqDotnetXAMonitor 是一個 .NET 受管理應用程式，可監視佇列管理程式並處理 SYSTEM.DOTNET.XARECOVERY.QUEUE 並回復未完成的交易

如果訊息通道代理程式 (MCA) 無法將訊息放置到目的地佇列，它會產生包含原始訊息的異常狀況報告，並將它放置在傳輸佇列中，以傳送到原始訊息中指定的回覆目的地佇列。 (如果回覆目的地佇列位於與 MCA 相同的佇列管理程式上，則訊息會直接放置在該佇列中，而不是放置在傳輸佇列中。)

下列交易被視為未完成且已回復:

- 如果交易已備妥，但 COMMIT 未在逾時期間內完成。
- 如果交易已備妥，但 IBM MQ 佇列管理程式已關閉。
- 如果交易已備妥，但「交易管理程式」已關閉。

XA 監視器應用程式必須從 IBM MQ .NET 用戶端應用程式執行所在的系統執行。 如果有應用程式在多個系統上執行並連接至相同的佇列管理程式，則必須從所有系統執行 WmqDotnetXAMonitor 應用程式。 雖然每一個用戶端機器都有一個執行中的 XA 監視器應用程式實例來回復應用程式，但每一個 XA 監視器實例應該能夠識別對應於現行 XA 監視器本端 MS DTC 所協調之交易的訊息，以便它可以重新列入並完成。

相關概念

第 480 頁的『IBM MQ .NET 的交易回復使用案例』
可能需要從數個不同的使用案例回復交易。

相關工作

第 481 頁的『使用 WMQDotnetXAMonitor 應用程式』
IBM MQ .NET 用戶端提供 XA 監視器應用程式 WmqDotnetXAMonitor ，可用來回復任何未完成的分散式交易。 WmqDotnetXAMonitor 應用程式會建立與交易不確定的佇列管理程式的連線，然後根據您設定的參數來解析交易。

IBM MQ .NET 的交易回復使用案例

可能需要從數個不同的使用案例回復交易。

- **IBM MQ 使用單一 DTC 及單一佇列管理程式實例的應用程式:** 在此使用案例中，當您連接至佇列管理程式並在交易下執行「工作單元」(UoW) 時，如果交易失敗並變成不完整，則 XA 監視器應用程式會回復交易並完成它。

  在此使用案例中，將有 XA 監視器應用程式的單一實例在執行中，因為單一佇列管理程式與交易相關聯。
- **使用單一 DTC 及單一佇列管理程式實例的多個 IBM MQ 應用程式:** 在此使用案例中，單一 DTC 下有多個 IBM MQ 應用程式，且全都連接至相同的佇列管理程式，並在交易下執行 UoW 。

  如果交易失敗並變成不完整， XA 監視器應用程式會回復它們，並完成與所有應用程式相關的交易。

  在此使用案例中， XA 監視器應用程式的單一實例會執行，因為在交易中使用一個佇列管理程式。
- **多個 IBM MQ 應用程式、多個 DTC、不同的佇列管理程式實例:** 在此使用案例中，在不同的 DTC 下 (亦即，每一個應用程式都在不同機器上執行) 有多個 IBM MQ 應用程式，並連接至不同的佇列管理程式。

  如果發生失敗且交易變成不完整，則監視器應用程式會檢查訊息中的 TransactionManagerWherearets ，以判定 DTC 位址。 如果 TransactionManagerWhereatos 值符合監視器執行所在的 DTC 位址，則它會完成回復，否則它會繼續搜尋，直到找到對應於其 DTC 的訊息為止。

  在此使用案例中，每個用戶端 (使用者或電腦) 只會執行一個 XA 監視器應用程式實例，因為每一個用戶端在交易中都有自己的佇列管理程式。
- **多個 IBM MQ 應用程式、多個 DTC、多個相同的佇列管理程式實例:** 在這個使用案例中，在不同的 DTC (亦即，每一個應用程式都在不同機器上執行) 下，有多個 IBM MQ 應用程式連接至相同的佇列管理程式。

  如果發生失敗且交易變成不完整，則監視器應用程式會驗證訊息中的 TransactionManagerWhereaos ，以檢查 DTC 位址及值是否與執行監視器的 DTC 相符。 如果這兩個值都相符，則它會完成回復，否則會繼續搜尋，直到找到對應於其 DTC 的訊息為止。

  在此使用案例中，每個用戶端 (使用者或電腦) 只會執行 XA 監視器應用程式的單一實例，因為每個用戶端在交易中都有自己的佇列管理程式關聯。
- **多個 IBM MQ 應用程式、單一 DTC、不同佇列管理程式實例:** 在此使用案例中，單一 DTC 下有多個 IBM MQ 應用程式 (亦即，在電腦上，有多個 IBM MQ 應用程式在執行中) ，並連接至不同的佇列管理程式。

  如果交易失敗且變成不完整，則監視器應用程式會回復交易。

在此使用案例中，監視器應用程式的實例數與連接至的佇列管理程式數一樣多，因為每一個應用程式都有自己的佇列管理程式用於交易中，且每一個應用程式都必須回復。

註: 如果 XA 監視器應用程式未在背景中執行，您可以啟動它。

相關概念

第 479 頁的『IBM MQ .NET 的交易回復程序』
本節說明如何使用 IBM MQ .NET 類別來回復分散式交易。

相關工作

第 481 頁的『使用 WMQDotnetXAMonitor 應用程式』
IBM MQ .NET 用戶端提供 XA 監視器應用程式 WmqDotnetXAMonitor，可用來回復任何未完成的分散式交易。 WmqDotnetXAMonitor 應用程式會建立與交易不確定的佇列管理程式的連線，然後根據您設定的參數來解析交易。

使用 *WMQDotnetXAMonitor* 應用程式

IBM MQ .NET 用戶端提供 XA 監視器應用程式 WmqDotnetXAMonitor，可用來回復任何未完成的分散式交易。 WmqDotnetXAMonitor 應用程式會建立與交易不確定的佇列管理程式的連線，然後根據您設定的參數來解析交易。

關於這項作業

必須手動執行 WMQDotnetXAMonitor 應用程式。 它可以隨時啟動。 當您看到 SYSTEM.DOTNET.XARECOVERY.QUEUE 或您可以在使用 IBM MQ .NET 類別撰寫的應用程式執行任何交易式工作之前，保持它在背景中執行。

您可以透過指令行或使用應用程式配置檔來設定 WMQDotnetXAMonitor 的參數值。 透過應用程式配置檔提供的值優先於透過指令行設定的值。

在 IBM MQ 9.3.0 之前， WMQDotnetXAMonitor 建立的連線是未受保護的連線。

V 9.3.0 從 IBM MQ 9.3.0 開始，您可以選擇透過設定 WMQDotnetXAMonitor 的其他參數來建立與佇列管理程式的安全連線。

程序

- 若要使用應用程式配置檔提供輸入給 WmqDotNETXAMonitor，請參閱 第 483 頁的『WmqDotNETXAMonitor 應用程式配置檔設定』。
- 若要從指令行啟動 WMQDotnetXAMonitor 應用程式，請搭配使用下列指令與您需要的參數:

  之前 IBM MQ 9.3.0:

  ```
  WmqDotnetXAMonitor.exe -m QueueManagerName -n ConnectionName -c ChannelName -i
  ```

  V 9.3.0 從 IBM MQ 9.3.0:

  ```
  WmqDotnetXAMonitor.exe -m QueueManagerName -n ConnectionName -c ChannelName -i -k SSL Key
  Repository -s Cipher Spec
  ```

  您可以指定的參數如下:

  – **-m *QueueManager* 名稱**
  佇列管理程式名稱。

  選用

  -n *ConnectionName*
  主機 (埠) 格式的連線名稱。 *ConnectionName* 可以包含多個連線名稱。 必須以逗點區隔清單提供多個連線名稱，例如 `localhost (1414), localhost (1415), localhost (1416)`。 WMQDotnetXAMonitor 應用程式會針對逗點區隔清單中指定的每一個連線名稱執行回復。

-c *ChannelName*
通道名稱。

-i
啟發式分支完成。

選用

V 9.3.0 **-k *SSL* 金鑰儲存庫**
SSL 金鑰儲存庫的名稱。 支援的值如下:

- *SYSTEM (這是預設值)
- *USER

選用

V 9.3.0 **-s 密碼規格**
您設定的 CipherSpec 必須是受支援版本的其中一個 CipherSpecs，它最好可以與 Windows 群組原則中指定的 CipherSpec 相同。 如需相關資訊，請參閱 第 501 頁的『受管理 .NET 用戶端的 CipherSpec 支援』。

對於建立與佇列管理程式的安全連線而言是必要的。

V 9.3.0 **-dn *SSLPeer* 名稱**
用來檢查同層級佇列管理程式中憑證的「識別名稱 (DN)」的 SSL 同層級名稱。

選用

V 9.3.0 **-cl 憑證標籤**
識別憑證的標籤名稱。

選用

V 9.3.0 **-sn *OutboundSNI***
在起始 TLS 連線時，是否應該將「伺服器名稱指示 (SNI)」設為遠端系統的目標 IBM MQ 通道名稱，或設為主機名稱。 此選項支援的值如下:

- CHANNEL (這是預設值)
- HOSTNAME
- *

如果未設定任何值，則會使用預設值 CHANNEL。

選用

V 9.3.0 **-cr 憑證撤銷檢查**
是否要執行憑證撤銷檢查。 此選項支援的值如下:

- true
- false (這是預設值)

選用

V 9.3.0 **-kr *KeyReset* 計數**
在重新協議用於加密的秘密金鑰之前，在通道上傳送及接收的未加密位元組總數。

預設值 0 表示永不重新協議秘密金鑰

選用

WMQDotnetXAMonitor 應用程式會執行下列動作:

1. 檢查 SYSTEM.DOTNET.XARECOVERY.QUEUE。
2. 如果佇列深度大於零，請瀏覽佇列中的訊息，並檢查訊息是否滿足不完整的交易準則。
3. 如果訊息滿足不完整的交易準則，則會拉出訊息，並擷取交易回復資訊。

4. 判定回復資訊是否與本端 Microsoft 分散式交易協調程式 (MS DTC) 相關。 如果是這種情況，則 WMQDotnetXAMonitor 會繼續回復交易，否則它會返回以瀏覽下一則訊息。
5. 呼叫佇列管理程式以回復不完整的交易。

WmqDotNETXAMonitor 應用程式配置檔設定
您可以使用應用程式配置檔，提供輸入給 IBM MQ .NET XA 監視器應用程式 WmqDotNETXAMonitor。 IBM MQ .NET 隨附了範例應用程式配置檔。 您可以根據需求來修改此範例檔。

透過應用程式配置檔提供的輸入值具有最高優先順序。 如果您在指令行及應用程式配置檔中提供輸入值，如第 481 頁的『使用 WMQDotnetXAMonitor 應用程式』中所述，則應用程式配置檔中的值優先。

IBM MQ 9.3.0 之前的應用程式配置檔範例。

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<configSections>
<sectionGroup name="IBM.WMQ">
<section name="dnetxa" type="System.Configuration.NameValueFileSectionHandler" />
</sectionGroup>
</configSections>
<IBM.WMQ>
<dnetxa>
<add key="ConnectionName" value=""/>
<add key="ChannelName" value="" />
<add key="QueueManagerName" value="" />
<add key="UserId" value="" />
<add key="SecurityExit" value="" />
<add key="SecurityExitUserData" value = "">
</dnetxa>
</dnetxa>
</configuration>
```

V 9.3.0 IBM MQ 9.3.0 中的應用程式配置檔範例。

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<configSections>
<sectionGroup name="IBM.WMQ">
<section name="dnetxa" type="System.Configuration.NameValueFileSectionHandler" />
</sectionGroup>
</configSections>
<IBM.WMQ>
<dnetxa>
<add key="ConnectionName" value=""/>
<add key="ChannelName" value="" />
<add key="QueueManagerName" value="" />
<add key="UserId" value="" />
<add key="SecurityExit" value="" />
<add key="SecurityExitUserData" value = "">
<add key="SSLKeyRepository" value="" />
<add key="SSLCipherSpec" value="" />
<add key="SSLPeerName" value="" />
<add key="SSLKeyResetCount" value="" />
<add key="SSLCertRevocationCheck" value="" />
<add key="CertificateLabel" value="" />
<add key="OutboundSNI" value="" />
</dnetxa>
</dnetxa>
</configuration>
```

WmqDotNetXAMonitor 應用程式日誌
「監視器應用程式」會在應用程式目錄中建立日誌檔，以記載「監視器」的進度及交易回復狀態。 記載會從連線名稱和通道詳細資料開始，以顯示正在執行回復的現行佇列管理程式。

一旦開始回復，將會記載交易回復訊息的 MessageId 、不完整交易的 TransactionId ，以及交易根據「交易管理程式協調」的實際結果。

範例日誌檔:

```
Time|ProcessId|ThreadId|WMQ .NET XA Recovery Monitor, Running now for
```

```
ConnectionName:xxxx, Time|ProcessId|ThreadId|Channel=xxxx
Time|ProcessId|ThreadId|Current QueueDepth = n
Time|ProcessId|ThreadId|Current MessageId = xxxx
Time|ProcessId|ThreadId|Current Incomplete Transaction being recovered = xxxxx
Time|ProcessId|ThreadId|Actual Outcome of the transaction(as per DTC)= Commit/Roll back
Time|ProcessId|ThreadId|Recovery Completed for TransactionId= xxxxx
Time|ProcessId|ThreadId|Current QueueDepth = n
Time|ProcessId|ThreadId|Current MessageId = xxxx
Time|ProcessId|ThreadId|Current Incomplete Transaction being recovered = xxxxx
Time|ProcessId|ThreadId|Actual Outcome of the transaction(as per DTC)= Commit/Roll back
Time|ProcessId|ThreadId| Recovery Completed for TransactionId= xxxxx
```

撰寫及部署 IBM MQ .NET 程式

若要使用 IBM MQ classes for .NET 來存取 IBM MQ 佇列，您可以使用 .NET 所支援的任何語言撰寫程式，其中包含將訊息放入 IBM MQ 佇列並從中取得訊息的呼叫。

IBM MQ 文件只包含 C#、C++ 及 Visual Basic 語言的相關資訊。

這個主題集合提供資訊來協助撰寫應用程式，以便與 IBM MQ 系統互動。 如需個別類別的詳細資料，請參閱 IBM MQ .NET 類別和介面。

連線差異

您為 IBM MQ.NET 撰寫程式的方式與您要使用的連線模式有一些相依關係。

當使用 IBM MQ classes for .NET 作為受管理用戶端時，與標準 IBM MQ MQI client 有一些差異，因為部分特性無法供受管理用戶端使用。

IBM MQ.NET 會從您指定給連線名稱、通道名稱、自訂值 NMQ_MQ_LIB 及內容 MQC.TRANSPORT_PROPERTY。

受管理用戶端連線

當使用 IBM MQ classes for .NET 作為受管理用戶端時，與標準 IBM MQ MQI client 有一些差異。

受管理用戶端無法使用下列特性:

- 通道壓縮
- 通道結束程式鏈結

如果您嘗試將這些特性與受管理用戶端搭配使用，則會傳回 MQException。 如果在連線的用戶端偵測到錯誤，則會使用原因碼 MQRC_ENVIRONMENT_ERROR。 如果在伺服器端偵測到它，則會使用伺服器傳回的原因碼。

針對未受管理用戶端寫入的通道結束程式無法運作。 您必須特別為受管理用戶端撰寫新的結束程式。 請檢查用戶端通道定義表 (CCDT) 中沒有指定無效的通道結束程式。

受管理通道結束程式的名稱最長可達 999 個字元。 不過，如果您使用 CCDT 來指定通道結束程式名稱，則限制為 128 個字元。

僅透過 TCP/IP 支援通訊。

當您使用 **endmqm** 指令停止佇列管理程式時，與 .NET 受管理用戶端的伺服器連線通道關閉時間可能比與其他用戶端的伺服器連線通道關閉時間更長。

如果您已將 *NMQ_MQ_LIB* 設為 **受管理** ，以便使用受管理 IBM MQ 問題診斷程式，則不支援 **strmqtrc** 指令的 -i、-p、-s、-b 或 -c 參數。

使用 XA 交易的受管理 .NET 應用程式將無法使用 z/OS 佇列管理程式。 嘗試連接至 z/OS 佇列管理程式的受管理 .NET 用戶端在 MQOPEN 呼叫上發生錯誤 MQRC_UOW_ENLISTMENT_ERROR (mqrc=2354) 失敗。 不過，使用 XA 交易的受管理 .NET 應用程式將使用分散式佇列管理程式。

定義要使用的連線類型

連線類型取決於連線名稱、通道名稱、自訂值 NMQ_MQ_LIB 及內容 MQC.TRANSPORT_PROPERTY。

您可以指定連線名稱，如下所示:

- 在 MQQueueManager 建構子上明確:

```
public MQQueueManager(String queueManagerName, MQLONG Options, string Channel,
string ConnName)
```

```
public MQQueueManager(String queueManagerName, string Channel, string ConnName)
```

- 透過在 MQQueueManager 建構子的雜湊表項目中設定內容 MQC.HOST_NAME_PROPERTY 及選擇性地設定 MQC.PORT_PROPERTY :

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- 作為明確 MQEnvironment 值

```
MQEnvironment.Hostname
```

`MQEnvironment.Port` (選用)。

- 透過在 MQEnvironment.properties 雜湊表中設定內容 MQC.HOST_NAME_PROPERTY 及選擇性地設定 MQC.PORT_PROPERTY 。

您可以指定通道名稱，如下所示:

- 在 MQQueueManager 建構子上明確:

```
public MQQueueManager(String queueManagerName, MQLONG Options, string Channel,
string ConnName)
```

```
public MQQueueManager(String queueManagerName, string Channel, string ConnName)
```

- 透過在 MQQueueManager 建構子上的雜湊表項目中設定內容 MQC.CHANNEL_PROPERTY :

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- 作為明確的 MQEnvironment 值

```
MQEnvironment.Channel
```

- 透過在 MQEnvironment.properties 雜湊表中設定內容 MQC.CHANNEL_PROPERTY 。

您可以指定傳輸內容，如下所示:

- 透過在 MQQueueManager 建構子上的雜湊表項目中設定內容 MQC.TRANSPORT_PROPERTY :

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- 透過在 MQEnvironment.properties 雜湊表中設定內容 MQC.TRANSPORT_PROPERTY 。

使用下列其中一個值來選取您需要的連線類型:

MQC.TRANSPORT_MQSERIES_BINDINGS -以伺服器身分連接
MQC.TRANSPORT_MQSERIES_CLIENT -以非 XA 用戶端連接
MQC.TRANSPORT_MQSERIES_XACLIENT -以 XA 用戶端連接
MQC.TRANSPORT_MQSERIES_MANAGED -連接為非 XA 受管理用戶端

您可以設定自訂值 NMQ_MQ_LIB ，以明確選擇連線類型，如下表所示。

NMQ_MQ_LIB 值	**連線類型**
mqic.dll	以非 XA 用戶端連接

NMQ_MQ_LIB 值	連線類型
mqicxa.dll	以 XA 用戶端連接
mqm.dll	以伺服器或非 XA 用戶端連接
管理	以非 XA 受管理用戶端身分連接

註: 為了與舊版相容，接受 mqic32.dll 和 mqic32xa.dll 的值作為 mqic.dll 和 mqicxa.dll 的同義字。 不過，從 IBM WebSphere MQ 7.1 開始， mqm.dll 和 mqm.pdb 僅是用戶端套件的一部分。

如果您選擇環境中無法使用的連線類型，例如您指定 mqic32xa.dll 且沒有 XA 支援，則 IBM MQ.NET 會擲出異常狀況。

將 NMQ_MQ_LIB 設為「受管理」會導致用戶端使用受管理 IBM MQ 問題診斷測試、 .NET 資料轉換及其他受管理低階 IBM MQ 功能。

NMQ_MQ_LIB 的所有其他值會導致 .NET 處理程序使用未受管理的 IBM MQ 問題診斷測試及資料轉換，以及其他未受管理的低階 IBM MQ 函數 (假設 IBM MQ MQI client 或伺服器已安裝在系統上)。

IBM MQ.NET 會選擇連線類型，如下所示:

1. 若為 MQC.TRANSPORT_PROPERTY ，它會根據 MQC.TRANSPORT_PROPERTY。

   不過請注意，該設定 MQC.TRANSPORT_PROPERTY 至 MQC.TRANSPORT_MQSERIES_MANAGED 不保證用戶端處理程序會執行受管理。 即使使用此設定，在下列情況下也不會管理用戶端:

   - 如果處理程序中的另一個執行緒已連接 MQC.TRANSPORT_PROPERTY 設為 MQC.TRANSPORT_MQSERIES_MANAGED。
   - 如果 NMQ_MQ_LIB 未設為「受管理」，則不會完全管理問題診斷測試、資料轉換及其他低階函數 (假設系統上已安裝 IBM MQ MQI client 或伺服器)。
2. 如果指定連線名稱時未指定通道名稱，或指定通道名稱時未指定連線名稱，則會擲出錯誤。
3. 如果同時指定了連線名稱和通道名稱:
   - 如果 NMQ_MQ_LIB 設為 mqic32xa.dll，則它會作為 XA 用戶端連接。
   - 如果 NMQ_MQ_LIB 設為受管理，則會以受管理用戶端的身分連接。
   - 否則，它會以非 XA 用戶端連接。
4. 如果指定 NMQ_MQ_LIB ，則它會根據 NMQ_MQ_LIB 的值進行連接。
5. 如果已安裝 IBM MQ 伺服器，則它會作為伺服器進行連接。
6. 如果已安裝 IBM MQ MQI client ，則它會作為非 XA 用戶端進行連接。
7. 否則，它會以受管理用戶端的身分連接。

Windows 使用 IBM MQ .NET 專案範本

IBM MQ .NET 用戶端可讓您使用專案範本來協助您開發 .NET Core 應用程式。

開始之前

您必須在系統上具有 Microsoft Visual Studio 2017 或更新版本，以及 .NET Core 2.1 。

您必須從下列項目複製 .NET 範本:

&MQ_INSTALL_ROOT&\tools\dotnet\samples\cs\core\base\ProjectTemplates\IBMMQ.NETClientApp.zip

目錄至

&USER_HOME_DIRECTORY&\Documents\&Visual_Studio_Version&\Templates\ProjectTemplates

目錄，其中:

- *&MQ_INSTALL_ROOT* 是安裝的根目錄
- *&USER_HOME_DIRECTORY* 是您的起始目錄。

您必須停止並重新啟動 Microsoft Visual Studio，才能挑選範本。

關於這項作業

.NET 專案範本包含一些通用代碼，可用來協助開發應用程式。使用內建程式碼，您可以連接至 IBM MQ 佇列管理程式，並只需修改內建程式碼中的內容，即可執行 put 或 get 作業。

程序

1. 開啟 Microsoft Visual Studio。
2. 按一下 **檔案**，接著按一下 **新建**，然後按一下 **專案**。
3. 在「建立新專案」視窗中，選取 `IBM MQ .NET Client App (.NET Core)`，然後按 **下一步**。
4. 在「配置新專案」視窗中，變更專案的 專案名稱 (如果您想要的話)，然後按一下 **建立** 以建立 .NET 專案。

   `MQDotnetApp.cs` 是與專案檔一起建立的檔案。此檔案包含連接至佇列管理程式並執行 put 及 get 作業的程式碼。

   連線內容會設為預設值:

   - MQC.CONNECTION_NAME_PROPERTY 已設為 *localhost (1414)*
   - MQC.CHANNEL_PROPERTY 設為 *DOTNET.SVRCONN*

   佇列設為 *Q1*，您可以相應地修改這些內容。
5. 編譯並執行應用程式。

相關概念

IBM MQ 元件和特性

.NET 應用程式執行時期-僅限 Windows

IBM MQ classes for .NET 的配置檔

.NET 用戶端應用程式可以使用 IBM MQ MQI client 配置檔，如果您使用受管理連線類型，也可以使用 .NET 應用程式配置檔。應用程式配置檔中的設定具有優先順序。

用戶端配置檔

IBM MQ classes for .NET 用戶端應用程式可以採用與任何其他 IBM MQ MQI client 相同的方式來使用用戶端配置檔。此檔案通常稱為 `mqclient.ini`，但您可以指定不同的檔名。如需用戶端配置檔的相關資訊，請參閱 IBM MQ MQI client 配置檔 `mqclient.ini`。

只有 IBM MQ MQI client 配置檔中的下列屬性與 IBM MQ classes for .NET 相關。如果您指定其他屬性，則它沒有作用。

表 *77:* 與 *IBM MQ classes for .NET* 相關的用戶端配置檔屬性

段落 (stanza)	**屬性**
通道	CCSID
通道	ChannelDefinition 目錄
通道	ChannelDefinition 檔案
通道	ReconDelay
通道	DefRecon
通道	MQReconnectTimeout

表 77: 與 *IBM MQ classes for .NET* 相關的用戶端配置檔屬性 (繼續)	
段落 (stanza)	**屬性**
通道	ServerConnection 參數
通道	Put1DefaultAlwaysSync
通道	PasswordProtection
ClientExit 路徑	ExitsDefaultPath
ClientExit 路徑	ExitsDefaultPath64
MessageBuffer	MaximumSize
MessageBuffer	PurgeTime
MessageBuffer	UpdatePercentage
V 9.3.0 V 9.3.0 安全	MQIInitialKey 檔案
V 9.3.0 V 9.3.0 SSL	SSLKeyRepository
V 9.3.0 V 9.3.0 SSL	SSLKeyRepository 密碼
TCP	ClntRcvBufSize
TCP	ClntSndBufSize
TCP	IPAddressVersion
TCP	KeepAlive

您可以使用適當的環境變數來置換任何這些屬性。

應用程式配置檔

如果您使用受管理連線類型執行，則也可以使用 .NET 應用程式配置檔來置換 IBM MQ 用戶端配置檔及對等環境變數。

只有在以受管理連線類型執行時，才會處理 .NET 應用程式配置檔設定，其他連線類型會忽略這些設定。

.NET 應用程式配置檔及其格式由 Microsoft 定義，以在 .NET 架構內一般使用，但本文件中所提及的特定區段名稱、索引鍵及值是 IBM MQ 特有的。

.NET 應用程式配置檔的格式是一些 區段。 每一個區段包含一或多個 索引鍵，且每一個索引鍵都有相關聯的值。 下列範例顯示 .NET 應用程式配置檔中用來控制 TCP/IP KeepAlive 內容的區段、索引鍵和值:

```
<configuration>
  <configSections>
    <section name="TCP" type="System.Configuration.NameValueSectionHandler"/>
  </configSections>
  <TCP>
    <add key="KeepAlive" value="true"></add>
  </TCP>
<configuration>
```

.NET 應用程式配置檔區段名稱及索引鍵中使用的關鍵字，完全符合用戶端配置檔中定義的段落及屬性的關鍵字。

區段 `<configSections>` 必須是 `<configuration>` 元素的第一個子元素。

請參閱 Microsoft 文件以取得進一步資訊。

與 .NET 搭配使用的 C# 程式碼片段範例

C# 程式碼片段，示範應用程式連接至佇列管理程式，將訊息放入佇列並接收回覆。

下列 C# 程式碼片段示範執行三個動作的應用程式:

1. 連接至佇列管理程式
2. 將訊息放置到 SYSTEM.DEFAULT.LOCAL.QUEUE
3. 取回訊息

它也會顯示如何變更連線類型。

```
// ============================================================================
// Licensed Materials - Property of IBM
// 5724-H72
// (c) Copyright IBM Corp. 2003, 2024
// ============================================================================
using System;
using System.Collections;

using IBM.WMQ;

class MQSample
{
  // The type of connection to use, this can be:-
  // MQC.TRANSPORT_MQSERIES_BINDINGS for a server connection.
  // MQC.TRANSPORT_MQSERIES_CLIENT for a non-XA client connection
  // MQC.TRANSPORT_MQSERIES_XACLIENT for an XA client connection
  // MQC.TRANSPORT_MQSERIES_MANAGED for a managed client connection
  const String connectionType = MQC.TRANSPORT_MQSERIES_CLIENT;

  // Define the name of the queue manager to use (applies to all connections)
  const String qManager = "your_Q_manager";

  // Define the name of your host connection (applies to client connections only)
  const String hostName = "your_hostname";

  // Define the name of the channel to use (applies to client connections only)
  const String channel = "your_channelname";



  /// <summary>
  /// Initialise the connection properties for the connection type requested
  /// </summary>
  /// <param name="connectionType">One of the MQC.TRANSPORT_MQSERIES_ values</param>
  static Hashtable init(String connectionType)
  {
    Hashtable connectionProperties = new Hashtable();

    // Add the connection type
    connectionProperties.Add(MQC.TRANSPORT_PROPERTY, connectionType);

    // Set up the rest of the connection properties, based on the
    // connection type requested
    switch(connectionType)
    {
      case MQC.TRANSPORT_MQSERIES_BINDINGS:
        break;
      case MQC.TRANSPORT_MQSERIES_CLIENT:
      case MQC.TRANSPORT_MQSERIES_XACLIENT:
      case MQC.TRANSPORT_MQSERIES_MANAGED:
        connectionProperties.Add(MQC.HOST_NAME_PROPERTY, hostName);
        connectionProperties.Add(MQC.CHANNEL_PROPERTY, channel);
        break;
    }

    return connectionProperties;
  }
  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [STAThread]
  static int Main(string[] args)
  {
    try
    {
```

```
      Hashtable connectionProperties = init(connectionType);

      // Create a connection to the queue manager using the connection
      // properties just defined
      MQQueueManager qMgr = new MQQueueManager(qManager, connectionProperties);

      // Set up the options on the queue we want to open
      int openOptions = MQC.MQOO_INPUT_AS_Q_DEF | MQC.MQOO_OUTPUT;

      // Now specify the queue that we want to open,and the open options
      MQQueue system_default_local_queue =
        qMgr.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE", openOptions);

      // Define an IBM MQ message, writing some text in UTF format
      MQMessage hello_world = new MQMessage();
      hello_world.WriteUTF("Hello World!");

      // Specify the message options
      MQPutMessageOptions pmo = new MQPutMessageOptions(); // accept the defaults,
                                                           // same as MQPMO_DEFAULT

      // Put the message on the queue
      system_default_local_queue.Put(hello_world, pmo);



      // Get the message back again

      // First define an IBM MQ message buffer to receive the message
      MQMessage retrievedMessage =new MQMessage();
      retrievedMessage.MessageId =hello_world.MessageId;

      // Set the get message options
      MQGetMessageOptions gmo =new MQGetMessageOptions(); //accept the defaults
                                                          //same as MQGMO_DEFAULT

      // Get the message off the queue
      system_default_local_queue.Get(retrievedMessage,gmo);

      // Prove we have the message by displaying the UTF message text
      String msgText = retrievedMessage.ReadUTF();
      Console.WriteLine("The message is: {0}", msgText);

      // Close the queue
      system_default_local_queue.Close();

      // Disconnect from the queue manager
      qMgr.Disconnect();
    }

    //If an error has occurred,try to identify what went wrong.

    //Was it an IBM MQ error?
    catch (MQException ex)
    {
      Console.WriteLine("An IBM MQ error occurred: {0}", ex.ToString());
    }

    catch (System.Exception ex)
    {
      Console.WriteLine("A System error occurred: {0}", ex.ToString());
    }

    return 0;
  }//end of start
}//end of sample
```

設定 IBM MQ 環境

在使用用戶端連線來連接佇列管理程式之前，您必須先設定 IBM MQ 環境。

註: 以伺服器連結模式使用 IBM MQ classes for .NET 時，不需要此步驟。

.NET 程式設計介面可讓您使用 NMQ_MQ_LIB 自訂值，但也包括類別 MQEnvironment。 此類別可讓您指定要在連線嘗試期間使用的詳細資料，例如下列清單中的那些詳細資料:

- 通道名稱

- 主機名稱
- 埠號
- 通道結束程式
- SSL 參數
- 使用者 ID 和密碼

如需 MQEnvironment 類別的完整資訊，請參閱 MQEnvironment.NET 類別

若要指定通道名稱及主機名稱，請使用下列程式碼:

```
MQEnvironment.Hostname = "host.domain.com";
MQEnvironment.Channel  = "client.channel";
```

依預設，用戶端會嘗試連接至埠 1414 的 IBM MQ 接聽器。 若要指定不同的埠，請使用下列程式碼:

```
MQEnvironment.Port = nnnn;
```

連接至佇列管理程式並切斷與佇列管理程式的連線

當您已配置 IBM MQ 環境時，已準備好連接至佇列管理程式。

若要連接至佇列管理程式，請建立新的 MQQueueManager 類別實例:

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

如果要切斷佇列管理程式的連線，請在佇列管理程式上呼叫 Disconnect 方法:

```
queueManager.Disconnect();
```

嘗試連接至佇列管理程式時，您必須對佇列管理程式具有查詢 (inq) 權限。 如果沒有查詢權限，連線嘗試會失敗。

如果您呼叫 Disconnect 方法，則會關閉您透過該佇列管理程式存取的所有開啟佇列及處理程序。 不過，當您完成使用這些資源時，最好是明確地關閉這些資源。 若要關閉資源，請在與每一個資源相關聯的物件上使用 Close 方法。

佇列管理程式上的 Commit 及 Backout 方法會取代與程序化介面搭配使用的 MQCMIT 及 MQBACK 呼叫。

存取佇列及主題

您可以使用 MQQueueManager 的方法或適當的建構子來存取佇列和主題。

若要存取佇列，請使用 MQQueueManager 類別的方法。 MQOD (物件描述子結構) 會收合到這些方法的參數中。 例如，若要在稱為 queueManager 的 MQQueueManager 物件所代表的佇列管理程式上開啟佇列，請使用下列程式碼:

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

options 參數與 MQOPEN 呼叫中的 Options 參數相同。

AccessQueue 方法會傳回 MQQueue 類別的新物件。

當您完成使用佇列時，請使用 Close () 方法來關閉它，如下列範例所示:

```
queue.Close();
```

使用 IBM MQ .NET，您也可以使用 MQQueue 建構子來建立佇列。 這些參數與 accessQueue 方法的參數完全相同，其中新增了佇列管理程式參數，指定要使用的已實例化 MQQueueManager 物件。 例如：

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

以此方式建構佇列物件可讓您撰寫自己的 MQQueue 子類別。

同樣地，您也可以使用 MQQueueManager 類別的方法來存取主題。 使用 AccessTopic() 方法來開啟主題。這會傳回 MQTopic 類別的新物件。 當您完成使用主題時，請使用 MQTopic 的 Close () 方法來關閉它。

您也可以使用 MQTopic 建構子來建立主題。 主題有許多建構子; 如需相關資訊，請參閱 MQTopic.NET 類別。

處理訊息

訊息是使用佇列或主題類別的方法來處理。 若要建置新訊息，請建立新的 MQMessageobject。

使用 MQQueue 或 MQTopic 類別的 Put () 方法，將訊息放入佇列或主題。 使用 MQQueue 或 MQTopic 類別的 Get () 方法，從佇列或主題取得訊息。 與程序化介面不同， MQPUT 及 MQGET 會放置及取得位元組陣列，而 IBM MQ classes for .NET 會放置及取得 MQMessage 類別的實例。 MQMessage 類別會封裝包含實際訊息資料的資料緩衝區，以及說明該訊息的所有 MQMD (訊息描述子) 參數。

若要建置新訊息，請建立 MQMessage 類別的新實例，並使用 WriteXXX 方法將資料放入訊息緩衝區。

建立新的訊息實例時，所有 MQMD 參數都會自動設為其預設值，如 MQMD 的起始值及語言宣告中所定義。MQQueue 的 Put () 方法也會採用 MQPutMessageOptions 類別的實例作為參數。 此類別代表 MQPMO 結構。 下列範例會建立訊息並將它放入佇列:

```
// Build a new message containing my age followed by my name
MQMessage myMessage = new MQMessage();
myMessage.WriteInt(25);

String name = "Charlie Jordan";
myMessage.WriteUTF(name);

// Use the default put message options...
MQPutMessageOptions pmo = new MQPutMessageOptions();

// put the message
!queue.Put(myMessage,pmo);
```

MQQueue 的 Get () 方法會傳回新的 MQMessage 實例，代表剛從佇列取得的訊息。 它也會採用 MQGetMessageOptions 類別的實例作為參數。 此類別代表 MQGMO 結構。

您不需要指定訊息大小上限，因為 Get () 方法會自動調整其內部緩衝區的大小，以符合送入訊息。 使用 MQMessage 類別的 ReadXXX 方法來存取所傳回訊息中的資料。

下列範例顯示如何從佇列取得訊息:

```
// Get a message from the queue
MQMessage theMessage    = new MQMessage();
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.Get(theMessage,gmo);  // has default values

// Extract the message data
int age = theMessage.ReadInt();
String name1 = theMessage.ReadUTF();
```

您可以設定 *encoding* 成員變數，來變更 read 和 write 方法使用的數字格式。

您可以透過設定 *characterSet* 成員變數，變更字集以用於讀取及寫入字串。

如需詳細資料，請參閱 MQMessage.NET 類別 。

註: MQMessage 的 WriteUTF() 方法會自動對字串長度及其包含的 Unicode 位元組進行編碼。 當另一個 .NET 程式 (使用 ReadUTF()) 讀取您的訊息時，這是傳送字串資訊最簡單的方式。

處理訊息內容

訊息內容可讓您選取訊息，或擷取訊息的相關資訊，而不存取其標頭。 MQMessage 類別包含取得及設定內容的方法。

您可以使用訊息內容來容許應用程式選取要處理的訊息，或在不存取 MQMD 或 MQRFH2 標頭的情況下擷取訊息的相關資訊。 它們也有助於 IBM MQ 與 JMS 應用程式之間的通訊。 如需 IBM MQ 中訊息內容的相關資訊，請參閱 訊息內容。

MQMessage 類別會根據內容的資料類型，提供一些方法來取得及設定內容。 get 方法的名稱格式為 Get * Property ， 而 set 方法的名稱格式為 Set * Property ， 其中星號 (*) 代表下列其中一個字串:

- 布林
- 位元組
- 位元組
- 雙線
- 浮點數
- 整數
- Int2
- Int4
- Int8
- 長
- 物件
- 短期
- 字串

例如，若要取得 IBM MQ 內容 myproperty (字串) ， 請使用呼叫 `message.GetStringProperty('myproperty')`。 您可以選擇性地傳遞內容描述子， IBM MQ 將完成該內容描述子。

處理錯誤

使用 `try` 和 `catch` 區塊來處理 IBM MQ classes for .NET 產生的錯誤。

.NET 介面中的方法不會傳回完成碼和原因碼。 相反地，每當 IBM MQ 呼叫所產生的完成碼和原因碼都不是零時，它們就會擲出異常狀況。 這可簡化程式邏輯，讓您在每次呼叫 IBM MQ 之後都不需要檢查回覆碼。 您可以決定要在程式中的哪些點處理失敗的可能性。 在這些點，您可以用 `try` 和 `catch` 區塊括住程式碼，如下列範例所示:

```
try
{
  myQueue.Put(messageA,PutMessageOptionsA);
  myQueue.Put(messageB,PutMessageOptionsB);
}
catch (MQException ex)
{
  // This block of code is only executed if one of
  // the two put methods gave rise to a non-zero
  // completion code or reason code.
  Console.WriteLine("An error occurred during the put operation:" +
                        "CC = " + ex.CompletionCode +
                        "RC = " + ex.ReasonCode);
  Console.WriteLine("Cause exception:" + ex );
}
```

取得及設定屬性值

類別 MQManagedObject、MQQueue 及 MQQueueManager 包含方法，可讓您取得並設定其屬性值。 請注意，對於 MQQueue ，只有在您開啟佇列時指定適當的查詢並設定旗標時，這些方法才會運作。

對於共同屬性， MQQueueManager 和 MQQueue 類別會繼承自稱為 MQManagedObject 的類別。 這個類別定義 Inquire () 和 Set () 介面。

當您使用 *new* 運算子來建立新的佇列管理程式物件時，會自動開啟該物件以進行查詢。 當您使用 AccessQueue() 方法來存取佇列物件時，查詢或設定作業不會自動開啟該物件，這可能會導致部分類型的遠端佇列發生問題。 若要使用「查詢及設定」方法，以及在佇列上設定內容，您必須在 AccessQueue() 方法的 openOptions 參數中指定適當的查詢及設定旗標。

inquire 及 set 方法採用三個參數:

- 選取元陣列
- intAttrs 陣列
- charAttrs 陣列

您不需要 MQINQ 中找到的 SelectorCount、 IntAttrCount 及 CharAttrLength 參數，因為陣列的長度一律是已知的。 下列範例顯示如何對佇列進行查詢:

```
//inquire on a queue
int [ ] selectors = new int [2] ;
int [ ] intAttrs = new int [1] ;
byte [ ] charAttrs = new byte [MQC.MQ_Q_DESC_LENGTH];
selectors [0] = MQC.MQIA_DEF_PRIORITY;
selectors [1] = MQC.MQCA_Q_DESC;
queue.Inquire(selectors,intAttrs,charAttrs);
ASCIIEncoding enc = new ASCIIEncoding();
String s1 = "";
s1 = enc.GetString(charAttrs);
```

可以對這些物件的所有屬性進行查詢。 屬性子集會公開為物件的內容。 如需物件屬性的清單，請參閱 物件屬性。 如需物件內容，請參閱適當的類別說明。

多執行緒程式

.NET 執行時期環境本質上是多執行緒。 IBM MQ classes for .NET 容許在多個執行緒之間共用佇列管理程式物件，但可確保對目標佇列管理程式的所有存取權都已同步。

請考量連接至佇列管理程式並在啟動時開啟佇列的簡式程式。 程式會在畫面上顯示單一按鈕。 當使用者按一下該按鈕時，程式會從佇列中提取訊息。 在此狀況下，應用程式起始設定會發生在一個執行緒中，而回應按鈕而執行的程式碼會在個別執行緒 (使用者介面執行緒) 中執行。

IBM MQ .NET 的實作可確保針對特定連線 (MQQueueManager 物件實例) ，對目標 IBM MQ 佇列管理程式的所有存取權都會同步。 預設行為是會封鎖想要對佇列管理程式發出呼叫的執行緒，直到該連線的所有其他進行中呼叫都完成為止。 如果您需要從程式內的多個執行緒同時存取相同的佇列管理程式，請為每一個需要並行存取的執行緒建立新的 MQQueueManager 物件。 (這相當於對每一個執行緒發出個別 MQCONN 呼叫。)

如果 MQC.MQCNO_HANDLE_SHARE_NONE 或 MQC.MQCNO_SHARE_NO_BLOCK 則佇列管理程式不再同步。

搭配使用用戶端通道定義表與 .NET

您可以將用戶端通道定義表 (CCDT) 與 IBM MQ classes for .NET 搭配使用。 您可以根據使用受管理或未受管理的連線，以不同方式指定 CCDT 的位置。

非 XA 或 XA 未受管理用戶端連線類型

使用未受管理的連線類型，您可以使用兩種方式來指定 CCDT 的位置:

- 使用環境變數 MQCHLLIB 來指定表格所在的目錄，並使用 MQCHLTAB 來指定表格的檔名。

- 使用用戶端配置檔。 在 CHANNELS 段落中，使用屬性 **ChannelDefinitionDirectory** 來指定表格所在的目錄，並使用 **ChannelDefinitionFile** 來指定檔名。

如果在用戶端配置檔中以及使用環境變數來指定位置，則環境變數會優先處理。 您可以使用此特性在用戶端配置檔中指定標準位置，並在必要時使用環境變數來置換它。

受管理用戶端連線類型

使用受管理連線類型，您可以使用三種方式來指定 CCDT 的位置:

- 使用 .NET 應用程式配置檔。 在 CHANNELS 區段中，使用索引鍵 **ChannelDefinitionDirectory** 來指定表格所在的目錄，並使用 **ChannelDefinitionFile** 來指定檔名。
- 使用環境變數 MQCHLLIB 來指定表格所在的目錄，並使用 MQCHLTAB 來指定表格的檔名。
- 使用用戶端配置檔。 在 CHANNELS 段落中，使用屬性 **ChannelDefinitionDirectory** 來指定表格所在的目錄，並使用 **ChannelDefinitionFile** 來指定檔名。

如果以多種方式指定位置，則環境變數優先於用戶端配置檔，而 .NET 應用程式配置檔優先於其他兩種方法。 您可以使用此特性在用戶端配置檔中指定標準位置，並在必要時使用環境變數或應用程式配置檔來置換它。

在 IBM MQ 9.3.0 之前，使用具有佇列管理程式分組的 CCDT 時，受管理 .NET 用戶端與 IBM MQ Java 及 C 用戶端之間的行為有所不同。 當 CCDT 檔案包含由三個佇列管理程式及三個明確 CLNTCONNs 組成的佇列管理程式群組時，如果應用程式提供 "*" 作為佇列管理程式， C 及 Java 用戶端會傳回 MQRC_Q_MGR_NAME_ERROR。 不過，受管理 .NET 用戶端會使用第一個可用的 CLNTCONN ，如果沒有可用的 CLNTCONN ，則會使用佇列管理程式分組的 CLNTCONN。

V 9.3.0 V 9.3.0 從 IBM MQ 9.3.0 開始，當使用 CCDT 與佇列管理程式分組搭配時， .NET 用戶端的行為方式與 C 及 Java 用戶端相同，並傳回 MQRC_Q_MGR_NAME_ERROR 。

.NET 應用程式如何決定要使用的通道定義

在 IBM MQ .NET 用戶端環境中，可以使用多種不同方式來指定要使用的通道定義。 通道定義可以有多個規格。 應用程式會從一或多個來源衍生通道定義。

如果存在多個通道定義，則會以下列優先順序選取所使用的通道定義:

1. 在 MQQueueManager 建構子上指定的內容 (明確或包括 *MQC.CHANNEL_PROPERTY*
2. MQEnvironment.properties 雜湊表中的內容 *MQC.CHANNEL_PROPERTY*
3. MQEnvironment 中的 通道 內容
4. .NET 應用程式配置檔，區段名稱 CHANNELS ， 索引鍵 ServerConnection 參數 (僅適用於受管理連線)
5. *MQSERVER* 環境變數
6. 用戶端配置檔、段落 CHANNELS、屬性 ServerConnection 參數
7. 用戶端通道定義表 (CCDT)。 CCDT 的位置指定在 .NET 應用程式配置檔中 (僅適用於受管理連線)
8. 用戶端通道定義表 (CCDT)。 CCDT 的位置是使用環境變數 *MQCHLIB* 及 *MQCHLTAB* 來指定
9. 用戶端通道定義表 (CCDT)。 使用用戶端配置檔來指定 CCDT 的位置

對於項目 1-3 ， 從應用程式提供的值逐欄位建置通道定義。 這些值可以使用不同的介面來提供，且每一個值可以有多個值。 欄位值會依照給定的優先順序新增至通道定義:

1. MQQueueManager 建構子上的 *connName* 值
2. MQQueueManager.properties 雜湊表中的內容值
3. MQEnvironment.properties 雜湊表中的內容值
4. 設為 MQEnvironment 欄位的值 (例如， MQEnvironment.Hostname、 MQEnvironment.Port)

對於項目 4-6 ， 會提供整個通道定義作為值。 通道定義上的未指定欄位採用系統預設值。 其他定義通道方法及其欄位的值不會與這些規格合併。

對於項目 7-9 ， 整個通道定義取自 CCDT。 定義通道時未明確指定的欄位會採用系統預設值。 其他定義通道方法及其欄位的值不會與這些規格合併。

在 IBM MQ .NET 中使用通道結束程式

如果您使用用戶端連結，則可以使用通道結束程式，如同任何其他用戶端連線一樣。 如果您使用受管理連結，則必須撰寫實作適當介面的結束程式。

用戶端連結

如果您使用用戶端連結，您可以依照 通道結束程式中的說明來使用通道結束程式。 您無法使用針對受管理連結所撰寫的通道結束程式。

受管理連結

如果您使用受管理連線來實作結束程式，您可以定義新的 .NET 類別來實作適當的介面。 IBM MQ 套件中定義了三個結束程式介面:

- MQSendExit
- MQReceiveExit
- MQSecurityExit

註: 在未受管理的環境中，不支援使用這些介面撰寫的使用者結束程式作為通道結束程式。

下列範例定義實作這三種方法的類別:

```
class MyMQExits : MQSendExit, MQReceiveExit, MQSecurityExit
{
   // This method comes from the send exit
  byte[] SendExit(MQChannelExit       channelExitParms,
                  MQChannelDefinition channelDefinition,
                  byte[]              dataBuffer
                  ref int             dataOffset
                  ref int             dataLength
                  ref int             dataMaxLength)
  {
     // complete the body of the send exit here
  }


  // This method comes from the receive exit
  byte[] ReceiveExit(MQChannelExit       channelExitParms,
                     MQChannelDefinition channelDefinition,
                     byte[]              dataBuffer
                     ref int             dataOffset
                     ref int             dataLength
                     ref int             dataMaxLength)
  {
     // complete the body of the receive exit here
  }


  // This method comes from the security exit
  byte[] SecurityExit(MQChannelExit       channelExitParms,
                      MQChannelDefinition channelDefParms,
                      byte[]              dataBuffer
                      ref int             dataOffset
                      ref int             dataLength
                      ref int             dataMaxLength)
  {
     // complete the body of the security exit here
  }

}
```

每個結束程式都會傳遞一個 MQChannelExit 和一個 MQChannelDefinition 物件實例。 這些物件代表程序化介面中定義的 MQCXP 和 MQCD 結構。

要由傳送結束程式傳送的資料，以及在安全或接收結束程式中接收的資料，是使用結束程式的參數來指定。

進入時，位元組陣列 *dataBuffer* 中長度為 *dataLength* 的偏移 *dataOffset* 的資料是即將由傳送結束程式傳送的資料，以及在安全或接收結束程式中接收的資料。 參數 *dataMaxLength* 提供長度上限 (從 *dataOffset* 開始) 可供 *dataBuffer* 中的結束程式使用。 附註: 對於安全結束程式，如果這是第一次呼叫結束程式，或選取夥伴結束程式不傳送任何資料，則 dataBuffer 可能是空值。

傳回時， *dataOffset* 及 *dataLength* 的值應該設為指向 .NET 類別隨後應使用之所傳回位元組陣列內的偏移及長度。 對於傳送結束程式，這指出它應該傳送的資料，對於安全或接收結束程式，則指出應該解譯的資料。結束程式通常應該傳回位元組陣列; 異常狀況是安全結束程式，可選擇不傳送任何資料，以及以 INIT 或 TERM 原因呼叫的任何結束程式。 因此，可以寫入的最簡單結束形式是只會傳回 dataBuffer 的結束形式:

最簡單的可能結束主體為:

```
{
  return dataBuffer;
}
```

MQChannelDefinition 類別

與受管理 .NET 用戶端應用程式一起指定的使用者 ID 及密碼，會設定在傳遞至用戶端安全結束程式的 IBM MQ .NET MQChannelDefinition 類別中。 安全結束程式會將使用者 ID 和密碼複製到 MQCD.RemoteUserIdentifier 和 MQCD.RemotePassword 欄位 (請參閱 第 816 頁的『寫入安全結束程式』)。

指定通道結束程式 (受管理用戶端)

如果您在建立 MQQueueManager 物件 (在 MQEnvironment 中或在 MQQueueManager 建構子上) 時指定通道名稱及連線名稱，則可以透過兩種方式指定通道結束程式。

依優先順序，這些是:

1. 在 MQQueueManager 建構子上傳遞雜湊表內容 MQC.SECURITY_EXIT_PROPERTY、MQC.SEND_EXIT_PROPERTY 或 MQC.RECEIVE_EXIT_PROPERTY 。
2. 設定 MQEnvironment SecurityExit、 SendExit 或 ReceiveExit 內容。

如果您未指定通道名稱和連線名稱，則要使用的通道結束程式來自從用戶端通道定義表 (CCDT) 挑選的通道定義。 無法置換儲存在通道定義中的值。 如需通道定義表的相關資訊，請參閱 用戶端通道定義表 及 第 494 頁的『搭配使用用戶端通道定義表與 .NET』 。

在每一種情況下，規格都採用下列格式的字串形式:

```
Assembly_name(Class_name)
```

類別名稱 是實作 IBM.WMQ.MQSecurityExit、 IBM.WMQ.MQSendExit 或 IBM.WMQ.MQReceiveExit 介面 (適當的話) 之 .NET 類別的完整名稱 (包括名稱空間規格)。 *Assembly_name* 是包含類別之組件的完整位置 (包括副檔名)。 如果您使用 MQEnvironment 或 MQQueueManager 的內容，該字串的長度限制為 999 個字元。 不過，如果在 CCDT 中指定通道結束程式名稱，則限制為 128 個字元。 必要時， .NET 用戶端程式碼會透過剖析字串規格來載入並建立指定類別的實例。

指定通道結束程式使用者資料 (受管理用戶端)

通道結束程式可以有相關聯的使用者資料。 如果您在建立 MQQueueManager 物件 (在 MQEnvironment 中或在 MQQueueManager 建構子上) 時指定通道名稱及連線名稱，則可以透過兩種方式來指定使用者資料。

依優先順序，這些是:

1. 在 MQQueueManager 建構子上傳遞雜湊表內容 MQC.SECURITY_USERDATA_PROPERTY、MQC.SEND_USERDATA_PROPERTY 或 MQC.RECEIVE_USERDATA_PROPERTY 。
2. 設定 MQEnvironment SecurityUser 資料、 SendUser 資料或 ReceiveUser 資料內容。

如果未指定通道名稱及連線名稱，則要使用的結束程式使用者資料值來自從用戶端通道定義表 (CCDT) 中挑選的通道定義。 無法置換儲存在通道定義中的值。 如需通道定義表的相關資訊，請參閱 用戶端通道定義表 及 第 494 頁的『搭配使用用戶端通道定義表與 .NET』 。

在每一種情況下，規格都是字串，限制為 32 個字元。

.NET 中的自動用戶端重新連線

您可以在非預期的連線中斷期間，讓用戶端自動重新連接至佇列管理程式。

例如，如果佇列管理程式停止或網路或伺服器失敗，則用戶端可能會非預期地從佇列管理程式斷線。

如果沒有自動用戶端重新連線，當連線失敗時，會產生錯誤。 您可以使用錯誤碼來協助您重新建立連線。

使用自動用戶端重新連線機能的用戶端稱為可重新連接的用戶端。 若要建立可重新連接的用戶端，請在連接至佇列管理程式時指定稱為重新連接選項的特定選項。

如果用戶端應用程式是 IBM MQ .NET 用戶端，當您使用 MQQueueManager 類別來建立佇列管理程式時，可以選擇為 CONNECT_OPTIONS_PROPERTY 指定適當的值，以取得自動用戶端重新連線。 如需 CONNECT_OPTIONS_PROPERTY 值的詳細資料，請參閱 重新連接選項 。

您可以選取用戶端應用程式是否一律連接及重新連接至相同名稱的佇列管理程式、相同佇列管理程式，或用戶端連線表格中以相同 QMNAME 定義的任何佇列管理程式集 (如需詳細資料，請參閱 CCDT 中的佇列管理程式群組)。

.NET 的傳輸層安全 (TLS) 支援

IBM MQ classes for .NET 用戶端應用程式支援傳輸層安全 (TLS) 加密。 TLS 通訊協定提供透過網際網路的通訊安全，並容許主從式應用程式以機密且可靠的方式進行通訊。

相關概念

IBM MQ.NET 受管理用戶端 TLS 支援

加密安全通訊協定: TLS

未受管理 *.NET* 用戶端的 *TLS* 支援

未受管理 .NET 用戶端的 TLS 支援基於 C MQI 及 IBM Global Security Kit (GSKit)。 C MQI 會處理 TLS 作業，且 GSKit 會實作 TLS 安全 Socket 通訊協定。

針對未受管理的 *.NET* 用戶端啟用 *TLS*

只有用戶端連線才支援 TLS。 若要啟用 TLS ， 您必須指定與佇列管理程式通訊時要使用的 CipherSpec ， 且這必須符合目標通道上設定的 CipherSpec 。

若要啟用 TLS ， 請使用 MQEnvironment 的 SSLCipherSpec 靜態成員變數來指定 CipherSpec 。 下列範例會連接至名為 SECURE.SVRCONN.CHANNEL，已設定為使用 TLS_RSA_WITH_AES_128_CBC_SHA: 的 CipherSpec 來要求 TLS:

V 9.3.0 V 9.3.0

```
MQEnvironment.Hostname         = "your_hostname";
MQEnvironment.Channel          = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.SSLCipherSpec    = "TLS_RSA_WITH_AES_128_CBC_SHA256";
MQEnvironment.SSLKeyRepository = "C:\mqm\key.kdb";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

如需 CipherSpecs 的清單，請參閱 指定 CipherSpecs 。

SSLCipherSpec 內容也可以使用連線內容雜湊表中的 MQC.SSL_CIPHER_SPEC_PROPERTY 來設定。

若要使用 TLS 順利連接，用戶端金鑰儲存庫必須設定「憑證管理中心」主要憑證鏈，可從中鑑別佇列管理程式所提供的憑證。 同樣地，如果 SVRCONN 通道上的 SSLClientAuth 已設為 MQSSL_CLIENT_AUTH_REQUIRED ，則用戶端金鑰儲存庫必須包含佇列管理程式所信任的識別個人憑證。

使用佇列管理程式的識別名稱

佇列管理程式會使用包含 識別名稱 (DN) 的 TLS 憑證來識別自己。

IBM MQ .NET 用戶端應用程式可以使用此 DN 來確保它與正確的佇列管理程式通訊。 DN 型樣是使用 MQEnvironment 的 sslPeerName 變數來指定。 例如，設定:

```
MQEnvironment.SSLPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

只有在佇列管理程式呈現「通用名稱」開頭為 QMGR 的憑證時，連線才會成功。 以及至少兩個組織單位名稱，第一個名稱必須是 IBM ， 第二個名稱必須是 WEBSPHERE。

也可以使用連線內容雜湊表中的 MQC.SSL_PEER_NAME_PROPERTY 來設定 SSLPeerName 內容。 如需「識別名稱」及設定對等節點名稱之規則的相關資訊，請參閱 保護 IBM MQ 安全。

如果設定 SSLPeerName ，則只有在設為有效型樣且佇列管理程式呈現相符憑證時，連線才會成功。

使用 *TLS* 時發生錯誤處理

使用 TLS 連接至佇列管理程式時， IBM MQ classes for .NET 可能會發出下列原因碼:

MQRC_SSL_NOT_ALLOWED
已設定 SSLCipherSpec 內容，但已使用連結連接。 只有用戶端連接才支援 TLS。

MQRC_SSL_PEER_NAME_MISMATCH
SSLPeerName 內容中指定的 DN 型樣不符合佇列管理程式所提供的 DN。

MQRC_SSL_PEER_NAME_ERROR
SSLPeerName 內容中指定的 DN 型樣無效。

V 9.3.0 V 9.3.0 **MQRC_KEY_REPOSITORY_ERROR**
金鑰儲存庫的位置未指定、無效或無法存取。

受管理 *.NET* 用戶端的 *TLS* 支援

受管理 .NET 用戶端使用 Microsoft .NET Framework 程式庫來實作 TLS 安全 Socket 通訊協定。 Microsoft System.Net.SecuritySslStream 類別以串流形式透過連接的 TCP Socket 來運作，並透過該 Socket 連線來傳送及接收資料。

所需的最低 .NET Framework 層次是 .NET Framework v3.5。 「密碼演算法」支援的層次基於應用程式使用的 .NET Framework 層次:

- 對於以 .NET Framework 層次 3.5 和 4.0 為基礎的應用程式，可用的安全 Socket 通訊協定為 SSL 3.0 和 TSL 1.0。
- 對於以 .NET Framework 層次 4.5 為基礎的應用程式，可用的安全 Socket 通訊協定為 SSL 3.0、TLS 1.1 及 TLS 1.2。

您可能需要將預期更高 TLS 通訊協定支援的應用程式移至 .NET Framework 中針對 Microsoft Security 支援所定義的較新架構版本。

受管理 .NET 用戶端的 TLS 支援主要特性如下:

TLS 通訊協定支援
.NET 受管理用戶端的 TLS 支援是透過 .NET SSLStream 類別來定義，且取決於應用程式所使用的 .NET 架構。 如需相關資訊，請參閱 第 501 頁的『受管理 .NET 用戶端的 TLS 通訊協定支援』。

CipherSpec 支援
.NET 受管理用戶端的 TLS 設定與 Microsoft.NET TLS steams 一樣。 如需相關資訊，請參閱 第 501 頁的『受管理 .NET 用戶端的 CipherSpec 支援』 和 第 502 頁的『受管理 .NET 用戶端的 CipherSpec 對映』。

金鑰儲存庫
用戶端上的金鑰儲存庫是 Windows 金鑰儲存庫。 伺服器端儲存庫是儲存庫的「加密訊息語法 (CMS)」類型。 如需相關資訊，請參閱 第 504 頁的『受管理 .NET 用戶端的金鑰儲存庫』。

憑證
您可以使用自簽 TLS 憑證來實作用戶端與佇列管理程式之間的交互鑑別。 如需相關資訊，請參閱 第 504 頁的『使用受管理 .NET 用戶端的憑證』。

SSLPEERNAME
在 .NET 中，應用程式可以使用選用 SSLPEERNAME 屬性來指定「識別名稱 (DN)」型樣。 如需相關資訊，請參閱 第 505 頁的『SSLPEERNAME』。

符合 FIPS 標準
Microsoft.NET 安全程式庫不支援以程式設計方式啟用 FIPS。 FIPS 啟用由 Windows 群組原則設定控制。

NSA Suite B 相符性
IBM MQ 實作 RFC 6460。 NSA 套組 B 的 Microsoft.NET 實作是 5430。 從 .NET 架構 3.5 開始，支援這樣做。

秘密金鑰重設或重新協議

雖然 SSLStream 類別不支援秘密金鑰重設或重新協議，為了與其他 IBM MQ 用戶端保持一致， .NET 受管理用戶端容許應用程式設定 `SSLKeyResetCount`。 如需相關資訊，請參閱 第 505 頁的『受管理 .NET 用戶端的秘密金鑰重設或重新協議』。

撤銷檢查

SSLStream 類別支援憑證撤銷檢查，憑證撤銷檢查由憑證鏈引擎自動完成。 如需相關資訊，請參閱 第 506 頁的『撤銷檢查』。

IBM MQ 安全結束程式支援

SSLStream 類別提供 IBM MQ 安全結束程式的有限支援。 可以查詢本端及遠端憑證以取得 `SSLPeerNamePtr`(主體 DN) 及 `SSLRemCertIssNamePtr` (發證者 DN)，因為 Microsoft.NET 中支援這樣做。 不過，不支援取得 DNQ、UNSTRUCTUREDNAME 及 UNSTRUCTUREDADDRESS 等屬性，因此無法使用結束程式來擷取這些值。

加密硬體支援

受管理 .NET 用戶端不支援加密硬體。

在受管理 IBM MQ .NET 及 XMS .NET 用戶端上支援 TLS1.3

V 9.3.2

從 IBM MQ 9.3.2 開始， IBM MQ .NET 及 XMS .NET 用戶端支援 TLS1.3，但前提是作業系統支援 TLS1.3。

受管理 .NET 用戶端使用 Microsoft .NET Framework 程式庫來實作 TLS 安全 Socket 通訊協定。 Microsoft `System.Net.SecuritySslStream` 類別透過連接的 TCP Socket 以串流方式運作，並透過該 Socket 連線傳送及接收資料。

在 Windows 上， .NET 使用 SCHANNEL，在 Linux .NET 上，使用 OpenSSL 進行 SSL 通訊。

Windows 對於在 Windows 上執行的 IBM MQ .NET 用戶端應用程式

Microsoft 已宣佈依預設 Windows 11 和 Windows Server 2022 支援 TSL1.3 密碼。

依預設，在兩個 Windows 版本上都會啟用 TLS_AES_128_GCM_SHA256 及 TLS_AES_256_GCM_SHA384 密碼組合。

小心:

- TLS_CHACHA20_POLY1305_SHA256 依預設未啟用密碼組合，但受支援。
- 對於已啟用 TLS1.3 的 IBM MQ .NET 用戶端，若要順利連接至佇列管理程式， IBM Global Security Kit (GSKit) 8.0.55.29 版是佇列管理程式端所需的最低版本。

Linux 對於在 Linux 上執行的 IBM MQ .NET 用戶端應用程式

因為 .NET 使用 Linux 上的 OpenSSL 進行 SSL 通訊，所以若要使用 TLS1.3， OpenSSL v1.1.1 是最低需求。

此外，由於 .NET 在 Linux 上使用 OpenSSL， OpenSSL 支援的所有密碼也應該適用於 .NET。

OpenSSL 支援下列 CipherSpecs for TLS1.3:

- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256
- TLS_AES_128_GCM_SHA256
- TLS_AES_128_CCM_8_SHA256
- TLS_AES_128_CCM_SHA256

相關概念

第 502 頁的『受管理 .NET 用戶端的 CipherSpec 對映』
IBM MQ.NET 介面會維護 IBM MQ 至 Microsoft.NET 對照表，該對照表可用來判斷受管理用戶端建立與佇列管理程式的安全連線所需的 TLS 通訊協定版本。

受管理 *.NET* 用戶端的 *TLS* 通訊協定支援
IBM MQ.NET TLS 支援基於 .NET SSLStream 類別。

註: 受管理 .NET 用戶端的 TLS 通訊協定支援取決於應用程式正在使用的 .NET Framework 層次。 如需相關資訊，請參閱 第 499 頁的『受管理 .NET 用戶端的 TLS 支援』。

若要讓 Microsoft.NET SSLStream 類別起始設定 TLS 並對佇列管理程式執行信號交換，您必須設定的其中一個必要參數是 **SSLProtocol**，其中必須指定 TLS 版本號碼，其必須是下列其中一個值:

- SSL3.0
- TLS1.0
- TLS1.2

此參數的值與偏好的 CipherSpec 所屬的「通訊協定」系列緊密連結。 當 SSLStream 啟動與伺服器 (佇列管理程式) 的 TLS 信號交換時，它會使用 **SSLProtocol** 中指定的 TLS 版本來識別要用於協議的 CipherSpecs 清單。

IBM MQ.NET 不會使任何內容可供應用程式用來設定此值。 相反地， IBM MQ 會使用對映表，在內部將 CipherSpec 集對映至「通訊協定」系列，並識別要使用的 SSLProtocol 版本。 此表格顯示 Microsoft.NET 與 IBM MQ 之間每一個受支援 CipherSpec 的對映，以及它們所屬的通訊協定版本。 如需相關資訊，請參閱第 502 頁的『受管理 .NET 用戶端的 CipherSpec 對映』。

受管理 *.NET* 用戶端的 *CipherSpec* 支援
在與伺服器信號交換期間，會使用應用程式的 CipherSpec 設定。

IBM MQ 用戶端可讓您設定與佇列管理程式信號交換期間使用的 CipherSpec 值。 IBM MQ 用戶端應該為要建立的安全連線設定有效的 CipherSpec ，最好是 Windows 群組原則中指定的 CipherSpec 。 將此欄位保留空白表示 Socket 上沒有任何安全保護的純文字通道。

對於 IBM MQ.NET 受管理用戶端， TLS 設定適用於 Microsoft.NET SSLStream 類別。 對於 SSLStream ，只能在 Windows 群組原則中設定 CipherSpec 或 CipherSpecs 的喜好設定清單，這是電腦層面的設定。 然後在與伺服器信號交換期間， SSLStream 會使用指定的 CipherSpec 或喜好設定清單。 如果是其他 IBM MQ 用戶端，則可以在應用程式中的 IBM MQ 通道定義上設定 CipherSpec 內容，並將相同的設定用於 TLS 協議。 由於此限制，不論 IBM MQ 通道配置中指定什麼， TLS 信號交換可能會協議任何支援的 CipherSpec 。 因此，這可能會導致佇列管理程式上發生錯誤 AMQ9631 。 若要避免此錯誤，請設定與您在應用程式中設定的 CipherSpec 相同的 CipherSpec ，以作為 Windows 群組原則中的 TLS 配置。

新的 IBM MQ.NET TLS 用戶端程式碼只會檢查是否已協議正確的通訊協定版本。 TLS 通訊協定版本衍生自應用程式所設定的 CipherSpec ，並用於與伺服器 (佇列管理程式) 進行 TLS 信號交換。 因此，設計需要在 IBM MQ.NET 受管理用戶端應用程式中設定 CipherSpec 。 如果 IBM MQ 用戶端所設定的 CipherSpec 不是來自 SSL 3.0、TLS 1.0 及 TLS 1.2 通訊協定的任何其他密碼，則 IBM MQ 受管理 .NET 用戶端依預設會與來自 SSL 3.0 或 TLS 1.0 通訊協定的任何密碼協議，且不會報告錯誤。

註: 如果應用程式提供的 CipherSpec 值不是 IBM MQ 已知的 CipherSpec ，則 IBM MQ 受管理 .NET 用戶端會將它視為不處理，並根據 Windows 系統的群組原則來協議連線。

設定 CipherSpec

有三種方式可以設定 CipherSpec:

MQEnvironment .NET 類別
下列範例顯示如何使用 MQEnvironment 類別來設定 CipherSpec 。

```
MQEnvironment.SSLKeyRepository = "*USER";
MQEnvironment.ConnectionName = connectionName;
MQEnvironment.Channel = channelName;
MQEnvironment.properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
MQEnvironment.SSLCipherSpec = "TLS_RSA_WITH_AES_128_CBC_SHA";
```

TLS CipherSpec 內容
下列範例顯示如何透過將雜湊表參數新增至 MQQueueManager 建構子來設定 CipherSpec 。

```
properties = new Hashtable();
properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
properties.Add(MQC.HOST_NAME_PROPERTY, hostName);
properties.Add(MQC.PORT_PROPERTY, port);
properties.Add(MQC.CHANNEL_PROPERTY, channelName);
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, cipherSpec);
properties.Add(MQC.SSL_PEER_NAME_PROPERTY, sslPeerName);
properties.Add(MQC.SSL_RESET_COUNT_PROPERTY, keyResetCount);
queueManager = new MQQueueManager(queueManagerName, properties);
```

Windows 群組原則

透過 Windows 群組原則管理主控台配置「密碼組合」清單時， SVRCONN 通道定義必須指定相符的 CipherSpec。 相符的 CipherSpec 可以是一般值 (例如 "ANY_TLS12_OR_HIGHER") ，或是對映至將從排序清單協議的最高「密碼組合」的特定值。 建議使用一般 CipherSpec 值與 .NET 用戶端搭配使用，因為如果用戶端清單的順序變更，它會避免需要變更 SVRCONN CipherSpec 配置。

CCDT 用法

IBM MQ.NET 只支援本端電腦上的「用戶端通道定義表」(.TAB 檔案)。 已設定 CipherSpec 值的現有 CCDT 檔案可用於 IBM MQ.NET 連線。 不過，在用戶端連線通道上設定的 CipherSpec 值會決定 TLS 通訊協定版本，而且必須符合在 Windows 群組原則中設定的 CipherSpec 。

相關概念

第 490 頁的『設定 IBM MQ 環境』
在使用用戶端連線來連接佇列管理程式之前，您必須先設定 IBM MQ 環境。

第 499 頁的『受管理 .NET 用戶端的 TLS 支援』
受管理 .NET 用戶端使用 Microsoft .NET Framework 程式庫來實作 TLS 安全 Socket 通訊協定。 Microsoft System.Net.SecuritySslStream 類別以串流形式透過連接的 TCP Socket 來運作，並透過該 Socket 連線來傳送及接收資料。

相關工作

指定 CipherSpecs

相關參考

MQEnvironment .NET 類別

受管理 *.NET* 用戶端的 *CipherSpec* 對映

IBM MQ.NET 介面會維護 IBM MQ 至 Microsoft.NET 對照表，該對照表可用來判斷受管理用戶端建立與佇列管理程式的安全連線所需的 TLS 通訊協定版本。

如果在 SVRCONN 通道上指定 CipherSpec ，則在 TLS 信號交換完成之後，佇列管理程式會嘗試將該 CipherSpec 與用戶端應用程式正在使用的協議 CipherSpec 進行比對。 如果佇列管理程式找不到相符的 CipherSpec，則通訊會失敗，並發生錯誤 AMQ9631。

IBM MQ.NET 介面會維護 IBM MQ 至 Microsoft.NET CipherSpec 對照表。 此表格用來判定用戶端要用來建立與佇列管理程式之安全 Socket 連線的 TLS 通訊協定版本。 根據 SSLCipherSpec 值， SSLProtocol 版本可以是 TLS 1.0 或 TLS 1.2，視您使用的 Microsoft.NET Framework 版本而定。

請確定您提供正確的 SSLCipherSpec 值，因為指定不正確的值可能會導致使用 SSL 3.0 或 TLS 1.0 通訊協定。

表 *78: IBM MQ* 和 *Microsoft.NET* 對照表

IBM MQ CipherSpec	**Microsoft.NET CipherSpec**	**TLS 版本**
TLS_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0
TLS_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0
TLS_RSA_WITH_3DES_EDE_CBC_SHA [1]	TLS_RSA_WITH_3DES_EDE_CBC_SHA [1]	TLS 1.0

表 78: *IBM MQ* 和 *Microsoft.NET* 對照表 (繼續)

IBM MQ CipherSpec	**Microsoft.NET CipherSpec**	**TLS 版本**
TLS_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2
TLS_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P256	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P384	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P256	TLS 1.2
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P384	TLS 1.2
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P384	TLS 1.2
ECDHE_ECDSA_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P521	TLS 1.2
ECDHE_RSA_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2
ECDHE_RSA_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P256	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P384	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P384	TLS 1.2
ECDHE_ECDSA_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P521	TLS 1.2
V 9.3.2 TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS 1.3
V 9.3.2 TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS 1.3
V 9.3.2 TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS 1.3

表 78: *IBM MQ* 和 *Microsoft.NET* 對照表 (繼續)		
IBM MQ CipherSpec	**Microsoft.NET CipherSpec**	**TLS 版本**
V 9.3.2 TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS 1.3
V 9.3.2 TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS 1.3

附註:

1. Deprecated 此 CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA 已淘汰。 不過，在連線因錯誤 AMQ9288 而終止之前，它仍可用來傳送最多 32 GB 的資料。 若要避免此錯誤，您需要避免使用三重 DES 演算法，或在使用此 CipherSpec 時啟用秘密金鑰重設。

相關概念

第 499 頁的『受管理 .NET 用戶端的 TLS 支援』

受管理 .NET 用戶端使用 Microsoft .NET Framework 程式庫來實作 TLS 安全 Socket 通訊協定。 Microsoft System.Net.SecuritySslStream 類別以串流形式透過連接的 TCP Socket 來運作，並透過該 Socket 連線來傳送及接收資料。

受管理 *.NET* 用戶端的金鑰儲存庫

受管理 .NET 用戶端使用的金鑰儲存庫是 Windows 金鑰儲存庫。 憑證和私密金鑰必須在使用者或系統金鑰儲存庫中可用，用戶端應用程式才能在 TLS 信號交換期間同時用於身分和信任。

用戶端

在應用程式中，您可以為金鑰儲存庫設定下列任一值:

- "*USER": IBM MQ.NET 會存取現行使用者的憑證儲存庫，以擷取用戶端憑證。
- "*SYSTEM": IBM MQ.NET 會存取本端電腦帳戶以擷取憑證。

用戶端的憑證必須儲存在使用者或電腦帳戶的「我的憑證」儲存庫中。 所有伺服器 (CA) 憑證都必須儲存在憑證儲存庫的根目錄中。

註: 您可以使用下列格式，將多個憑證儲存在單一檔案中:

- 個人資訊交換-PKCS #12 (.PFX， .P12)
- 加密訊息語法標準-PKCS #7 憑證 (.P7B)
- Microsoft 序列化憑證儲存庫 (.SST)

使用受管理 *.NET* 用戶端的憑證

對於用戶端憑證， IBM MQ 受管理 .NET 用戶端會存取 Windows 金鑰儲存庫，並載入所有符合憑證標籤或符合字串的用戶端憑證。

選取要使用的憑證時， IBM MQ 受管理 .NET 用戶端一律使用第一個相符憑證來進行 SSLStream TLS 信號交換。

依憑證標籤比對憑證

如果您設定憑證標籤， IBM MQ 受管理 .NET 用戶端會搜尋具有給定標籤名稱的 Windows 憑證儲存庫，以識別用戶端憑證。 它會載入所有相符憑證，並使用清單上的第一個憑證。 設定憑證標籤有兩個選項:

- 憑證標籤可以在存取 MQEnvironment.CertificateLabel 的 MQEnvironment 類別上設定。
- 憑證標籤也可以在雜湊表內容中設定，如下列範例所示，這些內容以 MQQueueManager 建構子作為輸入參數提供。

```
Hashtable properties = new Hashtable();
properties.Add("CertificateLabel", "mycert");
```

名稱 ("CertificateLabel") 且值區分大小寫。

依字串比對憑證

如果未設定憑證標籤，則會搜尋並使用符合字串 "ibmwebspheremq" 及現行登入使用者 (小寫) 的憑證。

相關工作

將用戶端安全連接至佇列管理程式

相關參考

MQEnvironment .NET 類別

SSLPEERNAME

SSLPEERNAME 屬性用來檢查同層級佇列管理程式中憑證的「識別名稱 (DN)」。

在 IBM MQ.NET 中，應用程式可以使用 SSLPEERNAME 來指定識別名稱型樣，如下列範例所示。

```
SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

至於其他 IBM MQ 用戶端， SSLPEERNAME 是選用參數。

如果未設定 SSLPEERNAME 值，則 IBM MQ.NET 受管理用戶端不會執行任何「遠端 (伺服器)」憑證驗證，且受管理用戶端只會依現狀接受「遠端 (/server)」憑證。

您設定 SSLPEERNAME 的方式取決於您使用的 IBM MQ 堆疊供應項目。

IBM MQ classes for .NET

有三個選項如下。

1. 在 MQEnvironment 類別中設定 MQEnvironment.SSLPeerName 。
2. `MQEnvironment.properties.Add(MQC.SSL_PEER_NAME_PROPERTY, value)`
3. 使用佇列管理程式建構子 `MQQueueManager (String queueManagerName, Hashtable properties)`。 在 `Hashtable properties` 中提供選項 2 的 SSLPEERNAME。

XMS .NET

在 Connection Factory 中設定 SSL 同層級名稱:

```
ConnectionFactory.SetStringProperty(XMSC.WMQ_SSL_PEER_NAME, value);
```

WCF

在 URI 中以分號區隔欄位包含 SslPeer 名稱。

相關參考

MQEnvironment .NET 類別

受管理 *.NET* 用戶端的秘密金鑰重設或重新協議

SSLStream 類別不支援秘密金鑰重設/重新協議。 不過，為了與其他 IBM MQ 用戶端一致， IBM MQ 受管理 .NET 用戶端容許應用程式設定 **SSLKeyResetCount**。

當達到限制時， IBM MQ.NET 會中斷與佇列管理程式的連線，且會通知應用程式，原因碼為 MQRC_CONNECTION_BROKEN 異常狀況。 應用程式可以選擇處理異常狀況並重新建立連線，或啟用 MQCNO_RECONNECT 選項，讓 IBM MQ.NET 自動重新連接佇列管理程式。

啟用自動用戶端重新連線機能表示當達到金鑰重設計數時，會關閉所有現有的連線，且 IBM MQ.NET 用戶端會重新建立所有連線。 如需自動用戶端重新連線的相關資訊，請參閱 自動用戶端重新連線。

相關概念

重設 SSL 和 TLS 秘密金鑰

撤銷檢查
SSLStream 類別支援憑證撤銷檢查。

憑證鏈引擎會自動完成撤銷檢查。 這適用於「線上憑證狀態通訊協定 (OCSP)」及「憑證撤銷清冊 (CRL)」。 SSLStream 類別使用的憑證撤銷只使用憑證中指定的伺服器，亦即伺服器由憑證本身指定。 HTTP CDP 延伸及 OCSP HTTP 可透過 HTTP Proxy 伺服器對 Proxy 提出要求。

您設定撤銷檢查的方式，取決於您使用的 IBM MQ 堆疊供應項目。

IBM MQ.NET
可以透過存取 `MQEnvironment.cs` 類別檔上的 **MQEnvironment.SSLCertRevocationCheck** 內容來設定撤銷檢查。

XMS .NET
可以在 Connection Factory 內容環境定義上設定撤銷檢查，如下列範例所示。

```
ConnectionFactory.SetBooleanProperty(XMSC.WMQ_SSL_CERT_REVOCATION_CHECK, true);
```

WCF
您可以使用下列命名慣例，在 URI 上設定撤銷檢查。

```
"SslCertRevocationCheck=true"
```

配置受管理 *IBM MQ .NET* 的 *TLS*
為受管理 IBM MQ .NET 配置 TLS 包括建立簽章者憑證，然後配置伺服器端、用戶端及應用程式。

關於這項作業

若要配置 TLS，您必須先建立適當的簽章者憑證。 簽章者憑證可以是自簽憑證，也可以是憑證管理中心所提供的憑證。 雖然可以在開發、測試或前置正式作業系統上使用自簽憑證，但請勿在正式作業系統上使用它們。 在正式作業系統上，使用您從授信外部憑證管理中心 (CA) 取得的憑證。

程序

1. 建立簽章者憑證。
   a) 若要建立自簽憑證，請使用 IBM MQ 隨附的下列其中一個工具:
      使用 **strmqikm** GUI，或從指令行使用 **runmqckm** 或 **runmqakm**。 如需使用這些工具的相關資訊，請參閱 使用 **runmqckm**、 **runmqakm** 及 **strmqikm** 來管理數位憑證。
   b) 若要從憑證管理中心 (CA) 取得佇列管理程式及用戶端的憑證，請遵循 從憑證管理中心取得個人憑證 中的指示。
2. 配置伺服器端。
   a) 使用 IBM Global Security Kit (GSKit)，在佇列管理程式上配置 TLS，如 安全地將用戶端連接至佇列管理程式中所述。
   b) 設定 SVRCONN 通道 TLS 屬性:
      - 將 **SSLCAUTH** 設為 `"REQUIRED/OPTIONAL"`。
      - 將 **SSLCIPH** 設為適當的 CipherSpec。

      如需相關資訊，請參閱 第 498 頁的『針對未受管理的 .NET 用戶端啟用 TLS』。
3. 配置用戶端。
   a) 將用戶端憑證匯入至 Windows 憑證儲存庫 (在使用者/電腦帳戶下)。
      IBM MQ .NET 會從 Windows 憑證儲存庫存取用戶端憑證，因此您必須將憑證匯入 Windows 憑證儲存庫，以建立與 IBM MQ 的安全 Socket 連線。 如需如何存取 Windows 金鑰儲存庫及匯入用戶端憑證的相關資訊，請參閱 匯入或匯出憑證及私密金鑰。
   b) 提供 CertificateLabel，如 將用戶端安全連接至佇列管理程式中所述。

c) 必要的話，請編輯 Windows 群組原則以設定 CipherSpec，然後重新啟動電腦，讓 Windows 群組原則更新生效。

4. 配置應用程式。

a) 設定 MQEnvironment 或 SSLCipherSpec 值，以將連線表示為安全連線。

您指定的值用來識別所使用的通訊協定 (TLS)。 CipherSpec 集應該是受支援 SSLProtocol 版本的其中一個 CipherSpecs ，它最好與 Windows 群組原則中指定的 CipherSpec 相同。 (支援的 SSLProtocol 版本取決於使用的 .NET 架構。 SSLProtocol 版本可以是 TLS 1.0 或 TLS 1.2，視您使用的 Microsoft .NET Framework 版本而定。)

註: 如果應用程式提供的 CipherSpec 值不是 IBM MQ 已知的 CipherSpec ，則 IBM MQ 受管理 .NET 用戶端會將它視為不處理，並根據 Windows 系統的群組原則來協議連線。

b) 將 SSLKeyRepository 內容設為 "*SYSTEM" 或 "*USER"。

c) 選擇性的: 將 SSLPEERNAME 設為伺服器憑證的識別名稱 (DN)。

d) 提供 CertificateLabel ，如 將用戶端安全連接至佇列管理程式中所述。

e) 設定您需要的任何進一步選用參數，例如 KeyReset 計數、 CertificationRevocation 檢查並啟用 FIPS。

如何設定 TLS 通訊協定及 TLS 金鑰儲存庫的範例

對於基本 .NET，您可以透過 MQEnvironment 類別來設定 TLS 通訊協定及 TLS 金鑰儲存庫，如下列範例所示:

```
MQEnvironment.SSLCipherSpec = "TLS_RSA_WITH_AES_128_CBC_SHA256";
MQEnvironment.SSLKeyRepository = "*USER";

MQEnvironment.properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, "TLS_RSA_WITH_AES_128_CBC_SHA256")
```

或者，您可以透過提供雜湊表作為 MQQueueManager 建構子的一部分，來設定 TLS 通訊協定及 TLS 金鑰儲存庫，如下列範例所示。

```
Hashtable properties = new Hashtable();
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, "TLS_RSA_WITH_AES_128_CBC_SHA256")
```

下一步

如需開始開發 IBM MQ .NET 受管理 TLS 應用程式的相關資訊，請參閱第 507 頁的『撰寫簡式應用程式』。

相關參考

MQEnvironment .NET 類別

KeyReset 計數 (MQLONG)

AIX, Linux, and Windows 的聯邦資訊存取安全標準 (FIPS)

撰寫簡式應用程式

撰寫簡式 IBM MQ 受管理 .NET TLS 應用程式的提示，包括下列範例: 設定 Connection Factory 的 SSL 內容、建立佇列管理程式實例、連線、階段作業及目的地，以及傳送測試訊息。

開始之前

您必須先為受管理 IBM MQ.NET 配置 TLS ，如 第 506 頁的『配置受管理 IBM MQ .NET 的 TLS』中所述。

對於基本 .NET 中的應用程式配置，請使用 MQEnvironment 類別或提供雜湊表作為 MQQueueManager 建構子的一部分來設定 SSL 內容。

對於 XMS .NET 中的應用程式配置，您可以在 Connection Factory 的內容環境定義中設定 SSL 內容。

程序

1. 設定 Connection Factory 的 SSL 內容，如下列範例所示。

   IBM MQ.NET 的範例

   ```
   properties = new Hashtable();
   properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
   properties.Add(MQC.HOST_NAME_PROPERTY, hostName);
   properties.Add(MQC.PORT_PROPERTY, port);
   properties.Add(MQC.CHANNEL_PROPERTY, channelName);
   properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
   properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, cipherSpec);
   properties.Add(MQC.SSL_PEER_NAME_PROPERTY, sslPeerName);
   properties.Add(MQC.SSL_RESET_COUNT_PROPERTY, keyResetCount);
   properties.Add("CertificateLabel", "ibmwebspheremq");
   MQEnvironment.SSLCertRevocationCheck = sslCertRevocationCheck;
   ```

   XMS .NET 的範例

   ```
   cf.SetStringProperty(XMSC.WMQ_SSL_KEY_REPOSITORY, "sslKeyRepository");
   cf.SetStringProperty(XMSC.WMQ_SSL_CIPHER_SPEC, cipherSpec);
   cf.SetStringProperty(XMSC.WMQ_SSL_PEER_NAME, sslPeerName);
   cf.SetIntProperty(XMSC.WMQ_SSL_KEY_RESETCOUNT, keyResetCount);
   cf.SetBooleanProperty(XMSC.WMQ_SSL_CERT_REVOCATION_CHECK, true);
   ```

2. 建立佇列管理程式實例、連線、階段作業及目的地，如下列範例所示。

   MQ .NET 的範例

   ```
   queueManager = new MQQueueManager(queueManagerName, properties);
   Console.WriteLine("done");

   // accessing queue
   Console.Write("Accessing queue " + queueName + ".. ");
   queue = queueManager.AccessQueue(queueName, MQC.MQOO_OUTPUT +
   MQC.MQOO_FAIL_IF_QUIESCING);
   Console.WriteLine("done");
   ```

   XMS .NET 的範例

   ```
   connectionWMQ = cf.CreateConnection();
   // Create session
   sessionWMQ = connectionWMQ.CreateSession(false, AcknowledgeMode.AutoAcknowledge);

   // Create destination
   destination = sessionWMQ.CreateQueue(destinationName);

   // Create producer
   producer = sessionWMQ.CreateProducer(destination);
   ```

3. 如下列範例所示傳送訊息。

   MQ .NET 的範例

   ```
   // creating a message object
   message = new MQMessage();
   message.WriteString(messageString);

   // putting messages continuously
   for (int i = 1; i <= numberOfMsgs; i++)
   {
   Console.Write("Message " + i + " <" + messageString + ">.. ");
   queue.Put(message);
   Console.WriteLine("put");
   }
   ```

   XMS .NET 的範例

   ```
   textMessage = sessionWMQ.CreateTextMessage();
   ```

```
textMessage.Text = simpleMessage;
producer.Send(textMessage);
```

4. 請驗證 TLS 連線。

   請檢查通道狀態，以驗證 TLS 連線已建立且正常運作。

配置 *SSLStream* 的追蹤
若要擷取與 SSLStream 類別相關的追蹤事件及訊息，您必須將系統診斷程式的配置區段新增至應用程式的應用程式配置檔。

關於這項作業

註:

此作業僅適用於 IBM MQ classes for .NET Framework 。 IBM MQ classes for .NET (.NET Standard 和 .NET 6 程式庫) 不支援應用程式配置檔。

如果您未將系統診斷程式的配置區段新增至應用程式配置檔，則 IBM MQ 受管理 .NET 用戶端將不會擷取任何與 TLS 及 SSLStream 類別相關的事件、追蹤資料或除錯點。

註: 使用 **strmqtrc** 啟動 IBM MQ 追蹤不會擷取所有必要的 TLS 追蹤。

程序

1. 建立應用程式專案的應用程式配置 (App.Config) 檔案。
2. 新增系統診斷程式配置區段，如下列範例所示。

```
<system.diagnostics>
           <sources>
                 <source name="System.Net" tracemode="includehex">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Net.Sockets">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Net.Cache">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Net.Security">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Security">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
           </sources>
           <switches>
                 <add name="System.Net" value="Verbose"/>
                 <add name="System.Net.Sockets" value="Verbose"/>
                 <add name="System.Net.Cache" value="Verbose"/>
                 <add name="System.Security" value="Verbose"/>
                 <add name="System.Net.Security" value="Verbose"/>
           </switches>

           <sharedListeners>
                 <add name="ExternalSourceTrace" type="IBM.WMQ.ExternalSourceTrace,
amqmdnet, Version=n.n.n.n, Culture=neutral, PublicKeyToken=dd3cb1c9aae9ec97" />
           </sharedListeners>
         <trace autoflush="true"/>
     </system.diagnostics>
```

小心: add name 項目的 Version 欄位必須是所使用 .net amqmdnet.dll 檔案的任何版本。

相關工作

使用應用程式配置檔來追蹤 IBM MQ classes for .NET Framework 用戶端

在受管理 *.NET* 中實作 *TLS* 的範例應用程式

提供範例應用程式，以顯示 IBM MQ classes for .NET、 XMS .NET 及 IBM MQ WCF 自訂通道中受管理 .NET 的 TLS 實作。

下表顯示範例應用程式的位置。 *MQ_INSTALLATION_PATH* 代表 IBM MQ 安裝所在的高階目錄。

表 *79:* 在受管理 *.NET* 中實作 *TLS* 的範例應用程式位置

IBM MQ.NET 堆疊供應項目	**範例的位置**
基底.NET	*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\base\SimplePut\SimplePut.cs *MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\base\SimpleGet\SimpleGet.cs
XMS .NET	*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\xms\simple\wmq\SimpleProducer\SimpleProducer.cs *MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\xms\simple\wmq\SimpleConsumer\SimpleConsumer.cs
WCF 的 IBM MQ 自訂通道	*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\MQMessagingOneWayService.cs

Windows 使用 .NET 監視器

.NET 監視器是類似於 IBM MQ 觸發監視器的應用程式。

重要: 如需重要資訊，請參閱 只能與 Windows 上的主要安裝搭配使用的特性 。

您可以建立 .NET 元件，每當在受監視佇列上收到訊息時，即會實例化這些元件，然後處理該訊息。 「.NET 監視器」由 **runmqdnm** 指令啟動，並由 **endmqdnm** 指令停止。 如需這些指令的詳細資料，請參閱 runmqdnm 及 endmqdnm。

若要使用 .NET Monitor ， 您可以撰寫實作 IMQObjectTrigger 介面的元件，該介面定義在 amqmdnm.dll 中。

元件可以是交易式或非交易式。 交易式元件必須繼承自 System.EnterpriseServices.ServicedComponent ，並登錄為 RequiresTransaction 或 SupportsTransaction。 它不得登錄為 RequiresNew ， 因為 .NET Monitor 已起始交易。

元件會從 **runmqdnm** 接收 MQQueueManager、MQQueue 及 MQMessage 物件。 當 runmqdnm 啟動時，如果已使用 *-u* 指令行選項指定「使用者參數」字串，則它也可能會收到該字串。 請注意，您的元件會接收到達 MQMessage 物件中受監視佇列之訊息的內容。 它不需要連接至佇列管理程式、開啟佇列或取得訊息本身。 然後，元件必須適當地處理訊息，並將控制權交還給「.NET 監視器」。

如果您的元件已撰寫為交易式元件，它會使用 System.EnterpriseServices.ServicedComponent 所提供的機能來登錄以確定或回復交易。

當元件接收 MQQueueManager 和 MQQueue 物件以及訊息時，就會提供該訊息的完整環境定義資訊，例如，在相同佇列管理程式上開啟另一個佇列，而不需要個別連接至 IBM MQ。

Windows 程式碼片段範例

本主題包含兩個元件範例，這些元件從「.NET 監視器」取得並列印訊息，一個使用交易式處理程序，另一個非交易式處理程序。 第三個範例顯示共用公用程式常式，適用於前兩個範例。 所有範例都在 C# 中。

範例 1: 交易式處理程序

```
/*****************************************************************/
/* Licensed materials, property of IBM                           */
/* 63H9336                                                       */
/* (C) Copyright IBM Corp. 2005, 2024.                           */
/*****************************************************************/
using System;
using System.EnterpriseServices;

using IBM.WMQ;
using IBM.WMQMonitor;

[assembly: ApplicationName("dnmsamp")]

// build:
//
// csc -target:library -reference:amqmdnet.dll;amqmdnm.dll TranAssembly.cs
//
// run (with dotnet monitor)
//
// runmqdnm -m QMNAME -q QNAME -a dnmsamp.dll -c Tran

namespace dnmsamp
{
  [TransactionAttribute(TransactionOption.Required)]
  public class Tran : ServicedComponent, IMQObjectTrigger
  {
    Util util = null;

    [AutoComplete(true)]
    public void Execute(MQQueueManager qmgr, MQQueue queue,
        MQMessage message, string param)
    {
      util = new Util("Tran");

      if (param != null)
        util.Print("PARAM: '" +param.ToString() + "'");

      util.PrintMessage(message);

      //System.Console.WriteLine("SETTING ABORT");
      //ContextUtil.MyTransactionVote = TransactionVote.Abort;

      System.Console.WriteLine("SETTING COMMIT");
      ContextUtil.SetComplete();
      //ContextUtil.MyTransactionVote = TransactionVote.Commit;
    }
  }
}
```

範例 2: 非交易式處理

```
/*****************************************************************/
/* Licensed materials, property of IBM                           */
/* 63H9336                                                       */
/* (C) Copyright IBM Corp. 2005, 2024.                           */
/*****************************************************************/

using System;

using IBM.WMQ;
using IBM.WMQMonitor;

// build:
//
// csc -target:library -reference:amqmdnet.dll;amqmdnm.dll NonTranAssembly.cs
//
// run (with dotnet monitor)
//
// runmqdnm -m QMNAME -q QNAME -a dnmsamp.dll -c NonTran
namespace dnmsamp
{
  public class NonTran : IMQObjectTrigger
  {
    Util util = null;
```

```
    public void Execute(MQQueueManager qmgr, MQQueue queue,
        MQMessage message, string param)
    {
      util = new Util("NonTran");

      try
      {
        util.PrintMessage(message);
      }

      catch (Exception ex)
      {
        System.Console.WriteLine(">>> NonTran\n{0}", ex.ToString());
      }
    }
  }
}
```

範例 3: 共用常式

```
/*********************************************************************/
/* Licensed materials, property of IBM                               */
/* 63H9336                                                           */
/* (C) Copyright IBM Corp. 2005, 2024.                               */
/*********************************************************************/

using System;

using IBM.WMQ;

namespace dnmsamp
{
    /// <summary>
    /// Summary description for Util.
    /// </summary>
    public class Util
    {
    /* ---------------------------------------------------------------- */
    /* Default prefix string of the namespace.                           */
    /* ---------------------------------------------------------------- */
    private string prefixText = "dnmsamp";

    /* ---------------------------------------------------------------- */
    /* Constructor that takes the replacement prefix string to use.     */
    /* ---------------------------------------------------------------- */
    public Util(String text)
    {
      prefixText = text;
    }

    /* ---------------------------------------------------------------- */
    /* Display an arbitrary string to the console.                      */
    /* ---------------------------------------------------------------- */
    public void Print(String text)
    {
      System.Console.WriteLine("{0} {1}\n", prefixText, text);
    }


 /* ---------------------------------------------------------------- */
    /* Display the content of the message passed to the console.        */
    /* ---------------------------------------------------------------- */
    public void PrintMessage(MQMessage message)
    {
      if (message.Format.CompareTo(MQC.MQFMT_STRING) == 0)
      {
        try
        {
          string messageText = message.ReadString(message.MessageLength);

          Print(messageText);
        }

        catch(Exception ex)
        {
          Print(ex.ToString());
        }
```

```
      }
      else
      {
        Print("UNRECOGNISED FORMAT");
      }
    }

    /* ------------------------------------------------------------------ */
    /* Convert the byte array into a hex string.                          */
    /* ------------------------------------------------------------------ */
    static public string ToHexString(byte[] byteArray)
    {
      string hex = "0123456789ABCDEF";

      string retString = "";

      for(int i = 0; i < byteArray.Length; i++)
      {
        int h = (byteArray[i] & 0xF0)>>4;
        int l = (byteArray[i] & 0x0F);

        retString += hex.Substring(h,1) + hex.Substring(l,1);
      }

      return retString;
    }
  }
}
```

編譯 IBM MQ .NET 程式

編譯以各種語言撰寫的 .NET 應用程式的範例指令。

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

若要使用 IBM MQ classes for .NET 來建置 C# 應用程式，請使用下列指令:

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib: MQ_INSTALLATION_PATH\bin /out:MyProg.exe
MyProg.cs
```

如果要使用 IBM MQ classes for .NET 來建置 Visual Basic 應用程式，請使用下列指令:

```
vbc /r:System.dll /r: MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:MyProg.exe MyProg.vb
```

若要使用 IBM MQ classes for .NET 來建置受管理 C++ 應用程式，請使用下列指令:

```
cl /clr MQ_INSTALLATION_PATH\bin Myprog.cpp
```

如需其他語言，請參閱語言供應商提供的文件。

使用獨立式 IBM MQ .NET 用戶端

IBM MQ .NET 用戶端可讓您包裝及部署 IBM MQ .NET 組件，而不需要在正式作業系統上使用完整 IBM MQ 用戶端安裝架構來執行應用程式。

開始之前

V 9.3.1 從 IBM MQ 9.3.1 開始，安裝在預設位置的 `amqmdnetstd.dll` 用戶端程式庫是以 .NET 6 為基礎。 基於 .NET Standard 的 `amqmdnetstd.dll` 用戶端程式庫已移至 IBM MQ 用戶端安裝套件中的新位置，現在位於下列位置:

- Windows 在 Windows 上: *MQ_INSTALLATION_PATH*`\bin\netstandard2.0`
- Linux 在 Linux 上: *MQ_INSTALLATION_PATH*`\lib64\netstandard2.0`

LTS 對於 IBM MQ 9.3.0 Long Term Support， `amqmdnetstd.dll` 媒體庫位於下列位置:

- Windows 在 Windows 上: *MQ_INSTALLATION_PATH*\bin。 範例應用程式安裝在 *MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base 中。
- Linux 在 Linux 上: *MQ_INSTALLATION_PATH*/lib64 path。 .NET 範例位於 *MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base 中。

LTS Stabilized 仍提供 amqmdnet.dll 程式庫，但此程式庫已穩定; 也就是說，不會在其中引進任何新特性。 對於任何最新特性，您必須移轉至 amqmdnetstd.dll 程式庫。 不過，您可以在 IBM MQ 9.1 Long Term Support 或 Continuous Delivery 版次上繼續使用 amqmdnet.dll 程式庫。

關於這項作業

您可以在已安裝完整 IBM MQ 用戶端的機器上建置 IBM MQ .NET 應用程式，稍後再將 IBM MQ .NET 組件 (即 amqmdnetstd.dll) 與應用程式包裝在一起，並將它部署在正式作業系統上。

您建置並部署的應用程式可以是傳統 .NET 應用程式、服務或 Microsoft Azure Web/工作者應用程式

在這類部署中， IBM MQ .NET 用戶端只支援佇列管理程式連線功能的受管理模式。 伺服器連結和未受管理的用戶端模式連線功能無法使用，因為這兩種模式需要完整的 IBM MQ 用戶端安裝。 任何嘗試使用這兩種其他模式都會導致應用程式異常狀況。

程序

在應用程式中參照 IBM MQ .NET 用戶端組件

- 以您對舊版所做的相同方式來參照應用程式中的 amqmdnetstd.dll 組件。

  將 amqmdnetstd.dll 組件的 **CopyLocal** 內容設為 True ， 以確保 amqmdnetstd.dll 組件複製到應用程式的 bin 目錄。 設定此內容也可協助應用程式包裝工具包裝所需的二進位檔，以部署在正式作業系統及 Microsoft Azure PaaS 雲端環境上。

新增廣域交易支援

- 確保您的應用程式將監視器應用程式 WMQDotnetXAMonitor 部署在機器上，以及應用程式本身。

  如果應用程式使用 IBM MQ .NET 受管理廣域交易特性，則它還必須在機器上連同應用程式本身一起部署 WMQDotnetXAMonitor 。 回復任何不確定的交易需要此公用程式。

啟動及停止追蹤

- 僅限 IBM MQ classes for .NET Framework ， 若要使用應用程式配置檔及 IBM MQ 特定追蹤配置檔來啟動及停止追蹤，請參閱 使用應用程式配置檔追蹤 IBM MQ for .NET Framework 用戶端類別。

  您必須使用應用程式配置檔和 IBM MQ 特定的追蹤配置檔，因為由於沒有完整的 IBM MQ 用戶端安裝架構，因此無法使用用來啟動和停止追蹤的標準工具 **strmqtrc** 和 **endmqtrc**。

  附註:

  - 這種產生追蹤的方式適用於 .NET 可重新配送的受管理用戶端以及獨立式 .NET 用戶端。 請參閱 .NET 應用程式執行時期-僅限 Windows。
  - IBM MQ classes for .NET (.NET Standard 和 .NET 6 程式庫) 不支援應用程式配置檔。 若要啟用 IBM MQ classes for .NET (.NET Standard 和 .NET 6 程式庫) 的追蹤，您可以使用 **MQDOTNET_TRACE_ON** 環境變數。 請參閱 使用環境變數追蹤 IBM MQ .NET 應用程式。

- V 9.3.3

  使用 mqclient.ini 檔案並設定「追蹤」段落的適當內容，以啟動及停止追蹤。

  請參閱 使用 mqclient.ini 追蹤 IBM MQ .NET 應用程式。

  從 IBM MQ 9.3.3 開始，您可以使用 mqclient.ini 檔案並設定「追蹤」段落的適當內容來配置追蹤。 您也可以使用 mqclient.ini 檔案來動態啟用及停用追蹤。

在應用程式配置檔中啟用連結重新導向

- To enable compile time binding reference of the IBM MQ .NET assembly to a later version of the assembly, add the <dependentAssembly> property to the application configuration file.

app.config 檔案中的下列範例 Snippet 會重新導向使用 IBM MQ .NET 組件的 IBM MQ 8.0.0 Fix Pack 2 (8.0.0.2) 版本所編譯的應用程式，但稍後會將已更新 IBM MQ.NET 組件的修正套件 IBM MQ 8.0.0 Fix Pack 3 套用至 8.0.0.3。

```
<runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
    <!-- amqmdnet related binding redirect -->
    <dependentAssembly>
      <assemblyIdentity name="amqmdnet"
                        publicKeyToken="dd3cb1c9aae9ec97"
                        culture="neutral" />
      <codeBase version="8.0.0.2"
            href="file:///amqmdnet.dll"/>
      <bindingRedirect oldVersion="1.0.0.3-8.0.0.2"
                       newVersion="8.0.0.3"/>
      <publisherPolicy apply="no" />
    </dependentAssembly>
  </assemblyBinding>
</runtime>
```

相關概念

第 465 頁的『正在安裝 IBM MQ classes for .NET』
IBM MQ classes for .NET(包括範例) 與 IBM MQ 一起安裝在 Windows 和 Linux 上

可重新配送的用戶端

.NET 應用程式執行時期-僅限 Windows

相關工作

第 481 頁的『使用 WMQDotnetXAMonitor 應用程式』
IBM MQ .NET 用戶端提供 XA 監視器應用程式 WmqDotnetXAMonitor，可用來回復任何未完成的分散式交易。 WmqDotnetXAMonitor 應用程式會建立與交易不確定的佇列管理程式的連線，然後根據您設定的參數來解析交易。

追蹤 IBM MQ .NET 應用程式

V 9.3.0 OutboundSNI 內容

您可以使用內容或環境變數，在應用程式中設定 **OutboundSNI** 內容。

從 IBM MQ 9.3.0 開始，您可以設定 MQC.OUTBOUND_SNI_PROPERTY，在使用 MQQueueManager 類別連接至佇列管理程式時使用雜湊表。

MQC.OUTBOUND_SNI_PROPERTY 採用下列值:

- MQC.OUTBOUND_SNI_CHANNEL，對映至 "CHANNEL"
- MQC.OUTBOUND_SNI_HOSTNAME，其對映至 "HOSTNAME"
- MQC.OUTBOUND_SNI_ASTERISK，其對映至 "*"

此外，您可以使用 MQOUTBOUND_SNI 環境變數來設定 **OutboundSNI** 內容，其採用下列值:

- CHANNEL
- HOSTNAME
- *

並在 App.config 檔案中設定 **OutboundSNI** 值，如同任何其他 mqclient.ini 內容一樣。

註: 此內容預設為 MQC.OUTBOUND_SNI_CHANNEL。

在受管理節點中設定 **OutboundSNI** 內容的優先順序如下:

1. 應用程式層次內容
2. 環境變數

對於未受管理節點中的 **OutboundSNI** 內容，僅支援 mqclient.ini。

App.config 檔案中設定的內容僅適用於 .NET Framework 應用程式。

如果您提供在應用程式層次或 App.config 檔案中無效的值，則會發出 MQRC_OUTBOUND_SNI_NOT_VALID 回覆碼。

如果您設定無效的環境變數，或在 mqclient.ini 檔案中提供無效的值，則會使用預設值 CHANNEL 。

OutboundSNI 及多個憑證

IBM MQ 使用 SNI 標頭來提供多個憑證功能。 如果應用程式連接至透過 CERTLABL 欄位配置為使用不同憑證的 IBM MQ 通道，則應用程式必須使用 CHANNEL **OutboundSNI** 設定進行連接。

如果 **OutboundSNI** 設定不是 CHANNEL 的應用程式連接至已配置憑證標籤的通道，則會拒絕該應用程式，並在佇列管理程式錯誤日誌中列印 AMQ9673 訊息。

如需 IBM MQ 如何提供多個憑證功能的相關資訊，請參閱 IBM MQ 如何提供多個憑證功能 。

開發 XMS .NET 應用程式

IBM MQ Message Service Client (XMS) for .NET (XMS .NET) 提供稱為 XMS 的應用程式設計介面 (API) ，其具有與 Java Message Service (JMS) 相同的介面集 API。 IBM MQ Message Service Client (XMS) for .NET 包含 XMS 的完全受管理實作，可供任何符合 .NET 標準的語言使用。

關於這項作業

XMS 支援:

- 點對點傳訊
- 發佈/訂閱傳訊
- 同步訊息遞送
- 非同步訊息遞送

XMS 應用程式可以與下列類型的應用程式交換訊息:

- XMS 應用程式
- IBM MQ classes for JMS 應用程式
- 原生 IBM MQ 應用程式
- 使用 IBM MQ 預設傳訊提供者的 JMS 應用程式

XMS 應用程式可以連接及使用下列任何傳訊伺服器的資源:

IBM MQ 佇列管理程式 (queue manager)
應用程式可以在連結或用戶端模式下連接。

WebSphere Application Server service integration bus
應用程式可以使用直接 TCP/IP 連線，也可以使用透過 TCP/IP 的 HTTP。

IBM Integration Bus
使用 WebSphere MQ Real-Time Transport 在應用程式與分配管理系統之間傳輸訊息。 可以使用 WebSphere MQ Multicast Transport 將訊息遞送至應用程式。

透過連接至 IBM MQ 佇列管理程式， XMS 應用程式可以使用 WebSphere MQ Enterprise Transport 來與 IBM Integration Bus 進行通訊。 或者， XMS 應用程式可以透過連接至 IBM MQ 來發佈及訂閱。

從 IBM MQ 9.1.1 開始， IBM MQ 支援 Windows 環境中應用程式的 .NET Core 。 如需相關資訊，請參閱 第 519 頁的『正在安裝 IBM MQ classes for XMS .NET』。

從 IBM MQ 9.1.2 開始， IBM MQ 支援 Linux 環境中應用程式的 .NET Core 。

從 IBM MQ 9.1.4 開始， XMS .NET 受管理應用程式可以自動平衡叢集佇列管理程式之間的連線。 同時支援 .NET Framework 和 .NET Standard 程式庫。 如需相關資訊，請參閱 關於統一叢集 及 自動應用程式平衡。

相關工作
與 IBM 支援中心聯絡

疑難排解 XMS .NETproblems 問題

XMS 支援的傳訊樣式

XMS 支援傳訊的點對點和發佈/訂閱樣式。

傳訊樣式也稱為傳訊網域。

點對點傳訊

點對點傳訊的一般形式使用佇列作業。 在最簡單的情況下，應用程式會隱含或明確地識別目的地佇列，以將訊息傳送至另一個應用程式。 基礎傳訊及佇列作業系統會從傳送端應用程式接收訊息，並將訊息遞送至其目的地佇列。 然後，接收端應用程式可以從佇列擷取訊息。

如果基礎傳訊及佇列作業系統包含 IBM Integration Bus， IBM Integration Bus 可能會將訊息及遞送訊息副本抄寫至不同的佇列。 因此，多個應用程式可以接收訊息。 IBM Integration Bus 也可以轉換訊息並將資料新增至其中。

點對點傳訊的主要性質是應用程式在傳送訊息時將訊息放在本端佇列上。 基礎傳訊及佇列作業系統會決定訊息傳送至哪個目的地佇列。 接收端應用程式會從目的地佇列擷取訊息。

發佈/訂閱傳訊

在發佈/訂閱傳訊中，有兩種類型的應用程式: 發佈者和訂閱者。

發佈者 以發佈訊息的形式提供資訊。 當發佈者發佈訊息時，它會指定主題，以識別訊息內資訊的主旨。

訂閱者 是已發佈資訊的消費者。 訂閱者透過建立訂閱來指定其感興趣的主題。

發佈/訂閱系統會從發佈者接收發佈，並從訂閱者接收訂閱。 它會將發佈遞送給訂閱者。 訂閱者只會接收其訂閱的那些主題的發佈。

發佈/訂閱傳訊的主要性質是發佈者在發佈訊息時識別主題。 它不會識別訂閱者。 如果在沒有訂閱者的主題上發佈訊息，則沒有應用程式會收到訊息。

應用程式可以同時是發佈者和訂閱者。

XMS 物件模型

XMS API 是物件導向介面。 XMS 物件模型基於 JMS 1.1 物件模型。

主要 XMS 類別

主要 XMS 類別或物件類型如下:

ConnectionFactory
ConnectionFactory 物件會封裝連線的一組參數。 應用程式使用 ConnectionFactory 來建立連線。 應用程式可以在執行時期提供參數，並建立 ConnectionFactory 物件。 或者，連線參數可以儲存在受管理物件的儲存庫中。 應用程式可以從儲存庫擷取物件，並從中建立 ConnectionFactory 物件。

連線
Connection 物件封裝從應用程式到傳訊伺服器的作用中連線。 應用程式使用連線建立階段作業。

目的地
應用程式使用 Destination 物件來傳送訊息或接收訊息。 在發佈/訂閱網域中， Destination 物件會封裝主題，而在點對點網域中， Destination 物件會封裝佇列。 應用程式可以提供參數，以在執行時期建立 Destination 物件。 或者，它可以從儲存在受管理物件儲存庫中的物件定義建立 Destination 物件。

Session
Session 物件是用於傳送及接收訊息的單一執行緒環境定義。 應用程式使用 Session 物件來建立 Message、 MessageProducer 及 MessageConsumer 物件。

訊息
Message 物件會封裝應用程式使用 MessageProducer 物件傳送或使用 MessageConsumer 物件接收的 Message 物件。

MessageProducer
應用程式使用 MessageProducer 物件將訊息傳送至目的地。

MessageConsumer
應用程式使用 MessageConsumer 物件來接收傳送至目的地的訊息。

XMS 物件及其關係

第 518 頁的圖 52 顯示 XMS 物件的主要類型: ConnectionFactory、連線、階段作業、 MessageProducer、MessageConsumer、訊息及目的地。 應用程式使用 Connection Factory 建立連線，使用連線建立階段作業。 然後，應用程式可以使用階段作業建立訊息、訊息產生者及訊息消費者。 應用程式使用訊息產生者將訊息傳送至目的地，使用訊息消費者接收已傳送至目的地的訊息。

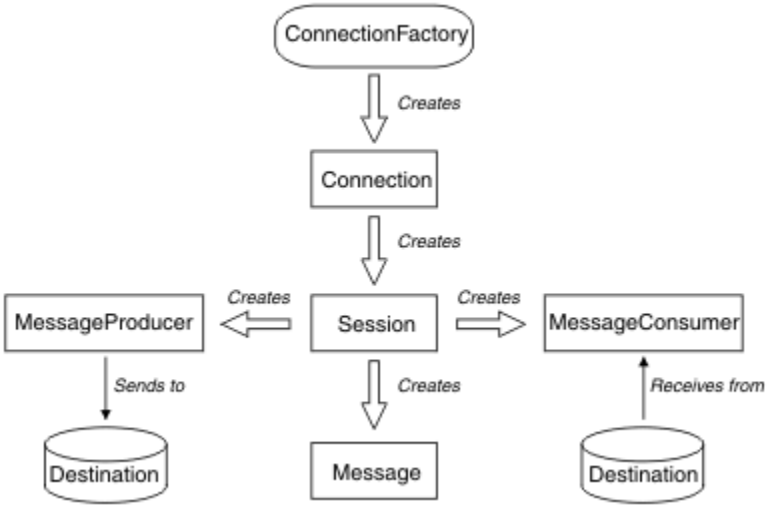


圖 *52: XMS* 物件及其關係

在 XMS .NET 中， XMS 類別定義為一組 .NET 介面。 當您撰寫 XMS .NET 應用程式的程式碼時，只需要宣告的介面。

XMS 物件模型基於 Java Message Service Specification 1.1 中說明的與網域無關的介面。 未提供網域特定的類別，例如 Topic、 TopicPublisher 及 TopicSubscriber。

XMS 物件的屬性及內容

XMS 物件可以具有以不同方式實作的屬性及內容，這些屬性及內容是物件的性質:

屬性
一律呈現並佔用儲存體的物件性質，即使屬性沒有值也是如此。 在這方面，屬性類似於固定長度資料結構中的欄位。 屬性的一個區別特性是每一個屬性都有自己的方法來設定及取得其值。

內容
物件的內容存在，且僅在設定其值之後才會佔用儲存體。 在設定內容值之後，無法刪除內容或回復其儲存體。 您可以變更其值。 XMS 提供一組一般方法來設定及取得內容值。

受管理物件

使用受管理物件，您可以管理要從中央儲存庫管理的用戶端應用程式所使用的連線設定。 應用程式會從中央儲存庫擷取物件定義，並使用它們來建立 ConnectionFactory 及 Destination 物件。 使用受管理物件，您可以將應用程式與它們在執行時期使用的資源取消連結。

例如，可以使用參照測試環境中一組連線及目的地的受管理物件來撰寫及測試 XMS 應用程式。 部署應用程式時，可以變更受管理物件，以將應用程式配置為參照正式作業環境中的連線及目的地。

XMS 支援兩種類型的受管理物件:

- `ConnectionFactory` 物件，供應用程式用來建立伺服器的起始連線。
- `Destination` 物件，由應用程式用來指定要傳送之訊息的目的地，以及要接收之訊息的來源。 目的地是伺服器上應用程式所連接的主題或佇列。

管理工具 **JMSAdmin** 隨 IBM MQ 一起提供。 它用來在受管理物件的中央儲存庫中建立及管理受管理物件 。

IBM MQ classes for JMS 和 XMS 應用程式可以使用儲存庫中的受管理物件。 XMS 應用程式可以使用 `ConnectionFactory` 及 `Destination` 物件來連接至 IBM MQ 佇列管理程式。 管理者可以變更保留在儲存庫中的物件定義，而不會影響應用程式碼。

下圖顯示 XMS 應用程式通常如何使用受管理物件。 圖表左側顯示的儲存庫包含使用管理主控台管理的 ConnectionFactory 和「目的地」物件定義。 圖表右側顯示的 XMS 應用程式會在儲存庫中查閱物件定義，然後在連接至傳訊伺服器時使用這些物件定義。

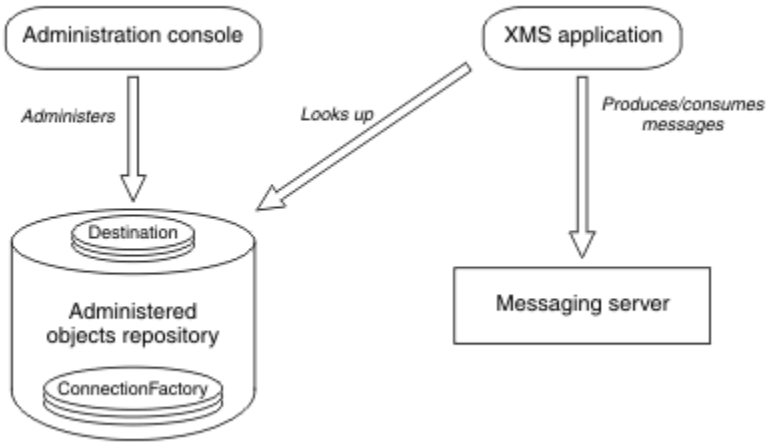


圖 *53: XMS* 應用程式一般使用受管理物件

XMS 訊息模型

XMS 訊息模型與 IBM MQ classes for JMS 訊息模型相同。

特別是， XMS 會實作 IBM MQ classes for JMS 所實作的相同訊息標頭欄位及訊息內容:

- JMS 標頭欄位。 這些欄位具有以 JMS 字首開頭的名稱。
- JMS 定義的內容。 這些欄位具有名稱以字首 JMSX 開頭的內容。
- IBM 定義的內容。 這些欄位具有名稱以字首 JMS_IBM_ 開頭的內容。

因此， XMS 應用程式可以與 IBM MQ classes for JMS 應用程式交換訊息。 在每一則訊息中，部分標頭欄位及內容由應用程式設定，其他則由 XMS 或 IBM MQ classes for JMS 設定。 XMS 或 IBM MQ classes for JMS 所設定的部分欄位會在傳送訊息時設定，而其他欄位則會在收到訊息時設定。 在適當的情況下，標頭欄位和內容會透過傳訊伺服器隨訊息一起傳播。 它們可供接收訊息的任何應用程式使用。

相關概念

IBM MQ classes for JMS

Windows Linux 正在安裝 IBM MQ classes for XMS .NET

IBM MQ classes for XMS .NET(包括範例) 與 IBM MQ 一起安裝在 Windows 和 Linux 上。

從 IBM MQ 9.2.0 開始， Microsoft.NET Core 3.1 是執行 IBM MQ classes for XMS .NET Standard 所需的最低版本。

V 9.3.0 V 9.3.0 從 IBM MQ 9.3.0 開始， IBM MQ 支援使用 IBM MQ classes for XMS .NET Standard 的 .NET 6 應用程式。 如果您使用 .NET Core 3.1 應用程式，則可以在 `csproj` 檔案中進行小型編輯來執行此應用程式，將 `targetframeworkversion` 設定為 `"net6.0"`，而不需要任何重新編譯。

V 9.3.1 IBM MQ 9.3.1 提供針對 .NET 6 建置的 XMS .NET 用戶端程式庫作為目標架構。 從 IBM MQ 9.3.1 開始， Microsoft .NET 6.0 是使用 IBM MQ 程式庫 (使用 .NET 6 作為目標架構來建置) 來執行應用程式所需的最低版本。

V 9.3.1 從 IBM MQ 9.3.1 開始，使用 .NET Standard 建置的 XMS .NET 用戶端程式庫可在新資料夾 `netstandard2.0` 下使用，而使用 .NET 6 建置的 XMS .NET 用戶端程式庫可在 Windows 上的 *MQ_INSTALLATION_PATH*`/bin` 及 Linux 上的 *MQ_INSTALLATION_PATH*`/lib64` 下使用。

amqmxmsstd.dll 磁帶庫

V 9.3.1 從 IBM MQ 9.3.1 開始， `amqmxmsstd.dll` 媒體庫位於下列位置

使用 .NET Standard 2.0 作為目標架構建置的程式庫

- Windows 在 Windows 上: *MQ_INSTALLATION_PATH*`\bin\netstandard2.0`。
- Linux 在 Linux 上: *MQ_INSTALLATION_PATH*`\lib64\netstandard2.0`。

Deprecated 這些程式庫已淘汰， IBM 打算在未來版本中移除它們。

使用 .NET 6 作為目標架構建置的程式庫

- Windows 在 Windows 上: *MQ_INSTALLATION_PATH*`\bin`。 範例應用程式安裝在 *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base` 中。
- Linux 在 Linux 上: *MQ_INSTALLATION_PATH*`\lib64`。 .NET 範例位於 *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base` 中。

LTS 對於 IBM MQ 9.3.0 Long Term Support， IBM MQ classes for XMS .NET Standard 媒體庫 `amqmxmsstd.dll` 位於下列位置:

- Windows 在 Windows 上: *MQ_INSTALLATION_PATH*`\bin`。 範例應用程式安裝在 *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/xms` 中。
- Linux 在 Linux 上: *MQ_INSTALLATION_PATH*`/lib64 path`。 .NET 範例位於 *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/xms` 中。

如需相關資訊，請參閱 第 465 頁的『正在安裝 IBM MQ classes for .NET』。

小心: V 9.3.1 Deprecated 從 IBM MQ 9.3.1 開始，使用 .NET Standard 2.0 作為目標架構建置的 IBM MQ .NET 用戶端程式庫已淘汰，且在編譯時期，參照這些程式庫的應用程式會擲出警告 CS0618 。

Stabilized 仍會提供所有 `IBM.XMS.*` 程式庫，但這些程式庫已穩定; 也就是說，不會在其中引進任何新特性。 對於任何最新特性，您必須移轉至 `amqmxmsstd.dll` 程式庫。 不過，您可以繼續使用 IBM MQ 9.1 Long Term Support 或 Continuous Delivery 版次上的現有檔案庫。

V 9.3.1 如果使用 `amqmdnetstd.dll` 或 `amqmxmsstd.dll` 從低於 IBM MQ 9.3.1 的版本編譯 .NET Framework 應用程式，且使用 .NET 6 型 IBM MQ 用戶端程式庫執行相同的應用程式，則 .NET 會擲出下列 `FileLoad` **異常狀況** 類型的異常狀況:

```
捕捉到異常狀況: System.IO.FileLoadException: 無法載入檔案或組件
'amqmdnetstd , Version =x.x.x.x , Culture=neutral , PublicKeyToken=23d6cb914eeaac0e' 或
其中一個相依關係。 所找到組件的資訊清單定義不符合
組件參照。 (來自 HRESULT 的異常狀況: 0x80131040)

檔名: 'amqmdnetstd , Version =x.x.x.x , Culture=neutral ,
PublicKeyToken=23d6cb914eeaac0e'
```

若要解決此錯誤，必須將 *MQ_INSTALLATION_PATH*/bin/netstandard2.0 中存在的程式庫複製到 .NET Framework 應用程式執行所在的目錄。

從 IBM MQ 9.2.0，可以從 NuGet 儲存庫下載 IBM MQ classes for XMS .NET Standard 。 NuGet 套件同時包含 amqmxmsstd.dll 程式庫及 amqmdnetstd.dll 程式庫。 amqmxmsstd.dll 相依於 amqmdnetstd.dll ， 在包裝 XMS .NET Core 應用程式時， amqmxmsstd.dll 和 amqmdnetstd.dll 應該與 XMS .NET Core 應用程式一起包裝。 如需相關資訊，請參閱 第 522 頁的『從 NuGet 儲存庫下載 IBM MQ classes for XMS .NET』。

dspmqver 指令

您可以使用 **dspmqver** 指令來顯示 .NET Core 元件的版本和建置資訊。

IBM MQ classes for XMS .NET Framework 與 IBM MQ classes for XMS .NET (.NET Standard 與 .NET 6 程式庫) 之間的比較

下表列出相較於 IBM MQ classes for XMS .NET (.NET Standard 及 .NET 6 程式庫) 的特性， IBM MQ classes for XMS .NET Framework 的特性。

表 *80: IBM MQ classes for XMS .NET Framework* 與 *IBM MQ classes for XMS .NET (.NET Standard* 與 *.NET 6* 程式庫*)* 之間的差異

特性	IBM MQ classes for XMS .NET Framework	IBM MQ classes for XMS .NET (.NET Standard 和 .NET 6 程式庫)
類別名稱 (API)	所有類別都在每個網路中保持相同。	所有類別都在每個網路中保持相同。
作業系統	Windows	Windows Docker 化的儲存器 Linux macOS
app.config 檔案（用於在可重新配送用戶端中啟用追蹤的配置檔案）	app.config 檔用來啟用可重新配送套件的追蹤。	不支援 app.config 。 使用環境變數。
追蹤	若要追蹤 XMS .NET 用戶端，您可以使用現有的環境變數，例如用來啟用追蹤的環境變數 **XMS_TRACE_ON** 。 如需相關資訊，請參閱 使用 XMS 環境變數來配置 XMS .NET 追蹤。 對於可重新配送的用戶端，可以使用 app.config 檔案來啟用追蹤。	若要追蹤 XMS .NET 用戶端，您可以使用現有的環境變數，例如用來啟用追蹤的環境變數 **XMS_TRACE_ON** 。 如需相關資訊，請參閱 使用 XMS 環境變數來配置 XMS .NET 追蹤。
傳輸模式	受管理、未受管理和連結	受管理的
TLS	Windows 金鑰儲存庫用於儲存憑證。	在 Windows 上，金鑰儲存庫必須用於儲存憑證。 允許的值為 *USER 或 *SYSTEM。 根據輸入，IBM MQ .NET 用戶端會查看現行使用者或系統層面的 Windows 金鑰儲存庫。 在 Linux 上，建議使用 X509Store 類別來安裝憑證，.NET Core 憑證安裝到下列位置：".dotnet/corefx/cryptography/x509stores"。
CCDT	支援	受支援，而且 CCDT 路徑的設定和 .NET Framework Framework 類別相同。

表 80: IBM MQ classes for XMS .NET Framework 與 IBM MQ classes for XMS .NET (.NET Standard 與 .NET 6 程式庫) 之間的差異 (繼續)		
特性	**IBM MQ classes for XMS .NET Framework**	**IBM MQ classes for XMS .NET (.NET Standard 和 .NET 6 程式庫)**
用戶端自動重新連接	支援	支援
分散式交易	支援	不支援
將動態鏈結程式庫 (dll) 安裝到廣域組合語言快取 (GAC) 中	將 Dll 作為 IBM MQ 安裝的一部分安裝在 GAC 中。	Dll 不作為 IBM MQ 安裝的一部分安裝在 GAC 中。
支援 WMQ、WPM 及 RTT 連線類型	支援 WMQ、WPM 及 RTT 連線類型	僅支援 WMQ
JNDI 受管理物件	支援 LDAP 及 FileSystem	僅支援 FileSystem

V 9.3.0 V 9.3.0 從 IBM MQ 9.3.0 開始，若要執行 IBM MQ classes for XMS .NET Framework，您必須安裝 Microsoft.NET Framework V4.7.2 或更新版本。

相關工作

第 527 頁的『使用 XMS 範例應用程式』

XMS .NET 範例應用程式提供每一個 API 的一般特性概觀。 您可以使用它們來驗證安裝及傳訊伺服器設定，並協助您建置自己的應用程式。

Windows Linux 從 NuGet 儲存庫下載 IBM MQ classes for XMS .NET

IBM MQ classes for XMS .NET 可從 NuGet 儲存庫下載，因此 .NET Developers 可以輕鬆使用它們。

關於這項作業

NuGet 是 Microsoft 開發平台的套件管理程式，包括 .NET。 NuGet 用戶端工具提供產生及耗用套件的能力。 NuGet 套件是具有 `.nupkg` 副檔名的單一壓縮檔，其中包含已編譯的程式碼 (DLL)、與該程式碼相關的其他檔案，以及包含套件版本號碼之類資訊的敘述性資訊清單。

您可以從 NuGet Gallery 下載 `IBMXMSDotnetClient` NuGet 套件，其中包含 `amqmdnetstd.dll` 檔案庫及 `amqmxmsstd.dll` 檔案庫，這是所有套件作者及消費者使用的中央套件儲存庫。

註: V 9.3.1 從 IBM MQ 9.3.1 開始， NuGet 套件包含使用 .NET Standard 2.0 和 .NET 6 作為目標架構所建置的程式庫。 .NET Standard 2.0 的程式庫位於 `netstandard2.0` 資料夾之下， .NET 6 的程式庫位於 `net6.0` 資料夾之下。

下載 `IBMXMSDotnetClient` 套件有三種方式:

- 使用 Microsoft Visual Studio。 NuGet 是作為 Microsoft Visual Studio 延伸來配送。 從 Microsoft Visual Studio 2012 開始，依預設會預先安裝 NuGet 。
- 從指令行使用 NuGet Package Manager 或 .NET CLI。
- 使用 Web 瀏覽器。

對於可重新配送的套件，您可以使用環境變數 **XMS_TRACE_ON** 來啟用追蹤。

程序

- 若要使用 Microsoft Visual Studio 中的「套件管理程式」使用者介面來下載 `IBMXMSDotnetClient` 套件，請完成下列步驟:
  a) 用滑鼠右鍵按一下 .NET 專案，然後按一下 **管理 Nuget 套件**。
  b) 按一下 **瀏覽** 標籤並搜尋 "`IBMXMSDotnetClient`"。
  c) 選取套件，然後按一下 **安裝**。

在安裝期間，「套件管理程式」會以主控台陳述式的形式提供進度資訊。

- 若要從指令行下載 IBMXMSDotnetClient 套件，請選擇下列其中一個選項:
  - 使用 NuGet Package Manager，輸入下列指令:

    ```
    Install-Package IBMXMSDotnetClient -Version 9.1.4.0
    ```

    在安裝期間，「套件管理程式」會以主控台陳述式的形式提供進度資訊。 您可以將輸出重新導向至日誌檔。
  - 使用 .NET CLI，輸入下列指令:

    ```
    dotnet add package IBMXMSDotnetClient --version 9.1.4
    ```
- 使用 Web 瀏覽器，從 https://www.nuget.org/packages/IBMXMSDotnetClient 下載 IBMXMSDotnetClient 套件。

相關概念

第 465 頁的『正在安裝 IBM MQ classes for .NET』
IBM MQ classes for .NET(包括範例) 與 IBM MQ 一起安裝在 Windows 和 Linux 上

IBM MQ Client for .NET 授權資訊

相關工作

第 468 頁的『從 NuGet 儲存庫下載 IBM MQ classes for .NET』
IBM MQ classes for .NET 可從 NuGet 儲存庫下載，因此 .NET 開發人員可以輕鬆使用它們。

設定傳訊伺服器環境

本節中的主題說明如何設定傳訊伺服器環境，以容許 XMS 應用程式連接至伺服器。

關於這項作業

對於連接至 IBM MQ 佇列管理程式的應用程式，需要 IBM MQ 用戶端 (或連結模式的佇列管理程式)。

對於使用分配管理系統即時連線的應用程式，目前沒有必要條件。

在執行任何 XMS 應用程式之前，您必須先設定傳訊伺服器環境，包括 XMS 隨附的範例應用程式。

本節包含下列主題:

- 第 525 頁的『為連接至 IBM MQ 佇列管理程式的應用程式配置佇列管理程式及分配管理系統』
- 第 519 頁的『正在安裝 IBM MQ classes for XMS .NET』
- 第 526 頁的『為使用分配管理系統即時連線的應用程式配置分配管理系統』
- 第 527 頁的『為連接至 WebSphere Application Server 的應用程式配置服務整合匯流排』

XMS .NET 中的訊息接聽器

訊息接聽器用來非同步接收訊息。 與 MessageConsumer.receive() 呼叫不同，訊息接聽器不會封鎖呼叫端執行緒，而是會將訊息遞送至應用程式指定的回呼方法，通常是 onMessage 方法。

一旦呼叫 Connection.Start() 方法，即會啟動訊息遞送。 您可以分別使用 Connection.Stop() 和 Connection.Start() 方法，隨時停止並回復訊息遞送。

在將訊息接聽器設為階段作業中至少一個消費者之後呼叫 Connection.Start() 方法之後，該階段作業會變成非同步階段作業。 一旦階段作業變成非同步，就無法呼叫任何 XMS .NET 同步方法。 例如，MessageProducer.Send()。 這樣做會導致異常狀況，原因碼為 IBM MQ MQRC_HCONN_ASYNC_ACTIVE (2500)。

非同步階段作業中的同步呼叫

Session.Close 是非同步階段作業中唯一容許的同步呼叫。 應用程式也可以使用訊息接聽器回呼方法 (即 onMessage 方法) 進行同步呼叫 (Session.Close 除外)。

除了這兩個選項之外，您必須使用 Connection.Stop() 方法來停止連線，應用程式才能進行任何同步呼叫。 進行呼叫之後，您必須使用 Connection.Start() 方法重新回復連線。 這會重新啟動訊息遞送。

階段作業可以有多少個非同步訊息消費者?

階段作業可以有多個非同步訊息消費者。 但在任何時間，訊息只會遞送給一個消費者。 這實際上意味著，當 XMS .NET 已呼叫消費者的 onMessage() 方法來遞送第一個訊息時，當第二個訊息到達時，在 onMessage() 方法傳回之前，不會將第二個訊息遞送至階段作業中的消費者。

只有在 onMessage() 方法傳回之後，第二則訊息才會在階段作業中遞送給消費者。 這是因為階段作業只使用一個執行緒來管理對消費者的訊息遞送。 這表示一次只能遞送一個訊息，而消費者可以是任何一個。

如果應用程式需要並行訊息遞送 (亦即，所有消費者必須同時接收訊息)，則應用程式必須建立多個階段作業，且每個階段作業必須有一個非同步訊息消費者。

下列範例更清楚地顯示此特性。

在第一個範例中，階段作業中有多個非同步訊息消費者。 階段作業 S 有三個非同步訊息消費者: AMC1、AMC2 及 AMC3，它們會從三個不同的目的地 Q1、 Q2 及 Q3 接收訊息。

因為只有一個階段作業 S，所以只有訊息遞送執行緒可將訊息遞送至消費者 AMC1、 AMC2 及 AMC3。 當階段作業將訊息遞送至 AMC1 時，其他兩個消費者 AMC2 和 AMC3 會等待，即使 Q2 和 Q3 中有訊息已備妥可供遞送。

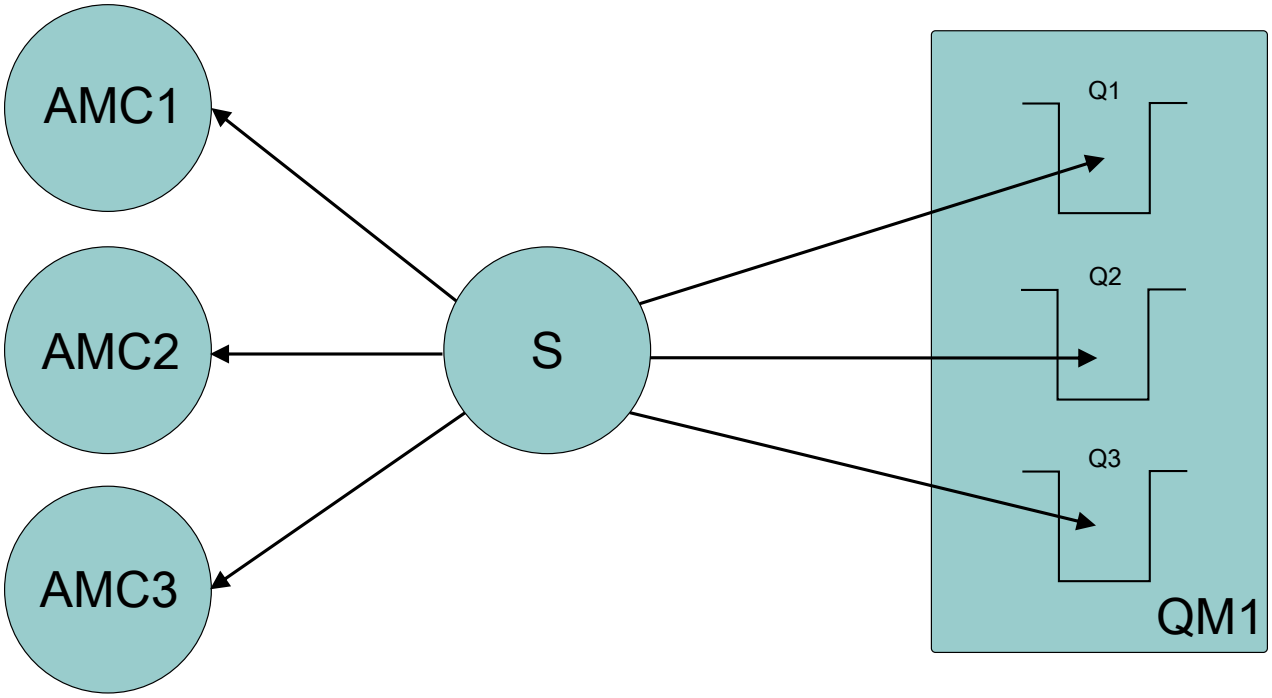


圖 *54:* 一個具有三個非同步訊息消費者的階段作業

在第二個案例中，有多個階段作業 S1、 S2 和 S3，分別有一個非同步訊息消費者 AMC1、 AMC2 和 AMC3。 因為每一個階段作業都有一個消費者，所以會同時將訊息遞送給消費者。

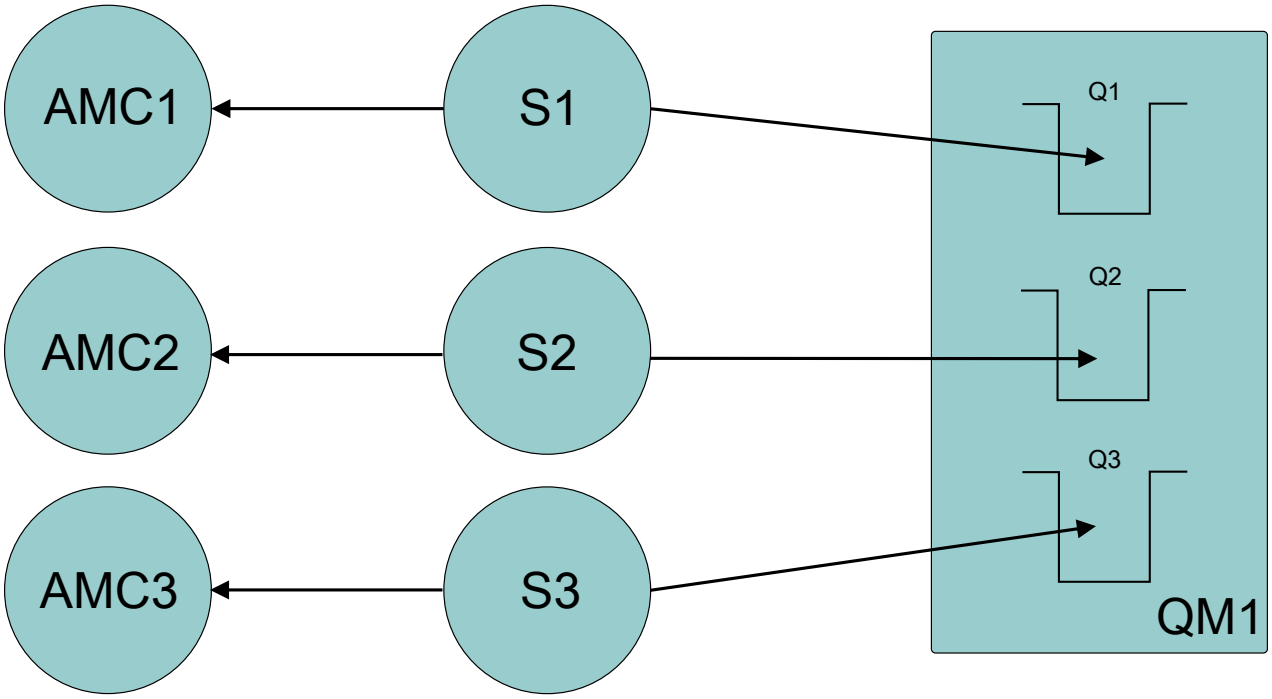


圖 *55:* 多個階段作業，每個階段作業都有一個非同步訊息消費者

這顯示如果您需要並行訊息遞送，則需要多個階段作業。

為連接至 IBM MQ 佇列管理程式的應用程式配置佇列管理程式及分配管理系統

本節假設您使用 IBM WebSphere MQ 7.0.1 或更新版本。 您必須先配置佇列管理程式，才能執行連接至 IBM MQ 佇列管理程式的應用程式。 對於發佈/訂閱應用程式，如果您使用排入佇列的發佈/訂閱介面，則需要一些額外的配置。

開始之前

XMS 與 IBM Integration Bus 或 WebSphere Message Broker 6.1 或更新版本搭配使用

在啟動此作業之前，請執行下列步驟:

- 請確定您的應用程式有權存取執行中的佇列管理程式。
- 如果您的應用程式是發佈/訂閱應用程式，且使用排入佇列的發佈/訂閱介面，請確定佇列管理程式上的 **PSMODE** 屬性設為 ENABLED 。
- 請確定您的應用程式使用其內容已適當設定的 Connection Factory ，以連接至佇列管理程式。 如果您的應用程式是發佈/訂閱應用程式，請確定已設定適當的 Connection Factory 內容來使用分配管理系統。 如需 Connection Factory 內容的相關資訊，請參閱 ConnectionFactory 的內容。

關於這項作業

您可以配置佇列管理程式及分配管理系統來執行 XMS 應用程式，方法與您配置佇列管理程式及已排入佇列的發佈/訂閱介面來執行 IBM MQ JMS 應用程式相同。 下列步驟彙總您需要執行的動作。

程序

1. 在佇列管理程式上，建立應用程式需要的佇列。

   如需如何建立佇列的概觀，請參閱 定義佇列。

   如果您的應用程式是發佈/訂閱應用程式，並使用需要存取 IBM MQ classes for JMS 系統佇列的排入佇列發佈/訂閱介面，請等到步驟 4a 之後再建立佇列。
2. 授與與應用程式相關聯的使用者 ID 連接至佇列管理程式的權限，以及存取佇列的適當權限。

如需授權的概觀，請參閱 保護。 如果您的應用程式以用戶端模式連接至佇列管理程式，另請參閱 用戶端及伺服器。

3. 如果您的應用程式以用戶端模式連接至佇列管理程式，請確定已在佇列管理程式中定義伺服器連線通道，且已啟動接聽器。

   您不需要對連接至佇列管理程式的每一個應用程式執行此步驟。 一個伺服器連線通道定義和一個接聽器可以支援以用戶端模式連接的所有應用程式。

4. 如果您的應用程式是發佈/訂閱應用程式，並使用佇列發佈/訂閱介面，請執行下列步驟。

   a) 在佇列管理程式上，執行 IBM MQ 隨附的 MQSC 指令 Script ，以建立 IBM MQ classes for JMS 系統佇列。 請確定與 IBM Integration Bus 或 WebSphere Message Broker 相關聯的使用者 ID 具有存取佇列的權限。

   如需在何處尋找 Script 以及如何執行 Script 的相關資訊，請參閱 使用 IBM MQ classes for Java。

   僅針對佇列管理程式執行一次此步驟。 同一組 IBM MQ classes for JMS 系統佇列可支援所有連接至佇列管理程式的 XMS 及 IBM MQ classes for JMS 應用程式。

   b) 授與與應用程式相關聯的使用者 ID 存取 IBM MQ classes for JMS 系統佇列的權限。

   如需使用者 ID 需要哪些權限的相關資訊，請參閱 使用 IBM MQ classes for JMS。

   c) 若為 IBM Integration Bus 或 WebSphere Message Broker 的分配管理系統，請建立並部署訊息流程，以處理應用程式在其中傳送所發佈訊息的佇列。

   基本訊息流程包含用於讀取已發佈訊息的 MQInput 訊息處理節點，以及用於發佈訊息的 Publication 訊息處理節點。

   如需如何建立及部署訊息流程的相關資訊，請參閱 IBM Integration Bus 產品說明文件檔案庫網頁中提供的 IBM Integration Bus 或 WebSphere Message Broker 產品說明文件。

   如果已在分配管理系統上部署適當的訊息流程，則不需要執行此步驟。

結果

您現在可以啟動應用程式。

為使用分配管理系統即時連線的應用程式配置分配管理系統

您必須先配置分配管理系統，才能執行使用即時連線至分配管理系統的應用程式。

開始之前

在啟動此作業之前，請執行下列步驟:

- 請確定您的應用程式可以存取執行中的分配管理系統。
- 請確定您的應用程式使用 Connection Factory ，其內容已適當地設定，可即時連接至分配管理系統。 如需 Connection Factory 內容的相關資訊，請參閱 ConnectionFactory 的內容。

關於這項作業

配置分配管理系統以執行 XMS 應用程式的方式，與配置分配管理系統以執行 IBM MQ classes for JMS 應用程式的方式相同。 下列步驟彙總您需要執行的動作:

程序

1. 建立並部署訊息流程，以從分配管理系統接聽及發佈訊息的 TCP/IP 埠讀取訊息。

   您可以利用下列任何一種方式來執行這項作業:

   - 建立包含 **Real-timeOptimizedFlow** 訊息處理節點的訊息流程。
   - 建立包含 **Real-timeInput** 訊息處理節點及 Publication 訊息處理節點的訊息流程。

   您必須將 **Real-timeOptimizedFlow** 或 **Real-timeInput** 節點配置成在用於即時連線的埠上接聽。 在 XMS 中，即時連線的預設埠號是 1506。

如果已在分配管理系統上部署適當的訊息流程，則不需要執行此步驟。

2. 如果您需要使用 IBM MQ classes for JMS 將訊息遞送至應用程式，請配置分配管理系統以啟用多重播送。 配置必須啟用多重播送的主題，為那些需要可靠多重播送的主題指定可靠的服務品質。
3. 如果您的應用程式在連接分配管理系統時提供使用者 ID 和密碼，且您想要分配管理系統使用此資訊來鑑別您的應用程式，請配置使用者名稱伺服器和分配管理系統，以進行簡單的 Telnet 型密碼鑑別。

結果

您現在可以啟動應用程式。

為連接至 WebSphere Application Server 的應用程式配置服務整合匯流排

您必須以配置服務整合匯流排來執行使用預設傳訊提供者的 JMS 應用程式的相同方式，來配置服務整合，才能執行連接至 WebSphere Application Server service integration technologies 服務整合匯流排的應用程式。

開始之前

在啟動此作業之前，您必須執行下列步驟:

- 請確定已建立傳訊匯流排，且您的伺服器已新增至匯流排作為匯流排成員。
- 請確定您的應用程式能夠存取至少包含一個執行中傳訊引擎的服務整合匯流排。
- 如果需要 HTTP 作業，則必須定義 HTTP 傳訊引擎入埠傳輸通道。 依預設，在伺服器安裝期間會定義 SSL 和 TCP 的通道。
- 請確定您的應用程式使用 Connection Factory ，且其內容已適當設定，可利用引導伺服器來連接服務整合匯流排。 所需的資訊下限為:
  - 提供者端點，說明在與傳訊伺服器 (亦即，透過引導伺服器) 協議連線時要使用的位置和通訊協定。 以最簡單的形式，對於以預設值安裝的伺服器，提供端點可以設為伺服器的主機名稱。
  - 用來傳送訊息的匯流排名稱。

  如需 Connection Factory 內容的相關資訊，請參閱 ConnectionFactory 的內容。

關於這項作業

必須定義您需要的任何佇列或主題空間。 依預設，在伺服器安裝期間會定義稱為 Default.Topic.Space 的主題空間，但如果您需要進一步的主題空間，則必須自行建立這些主題空間。 您不需要預先定義主題空間內的個別主題，因為伺服器會根據需要動態實例化這些個別主題。

下列步驟彙總您需要執行的動作。

程序

1. 建立應用程式點對點傳訊所需的佇列。
2. 建立應用程式用於發佈/訂閱傳訊所需的任何其他主題空間。

結果

您現在可以啟動應用程式。

使用 XMS 範例應用程式

XMS .NET 範例應用程式提供每一個 API 的一般特性概觀。 您可以使用它們來驗證安裝及傳訊伺服器設定，並協助您建置自己的應用程式。

關於這項作業

如果您需要協助來建立自己的應用程式，您可以使用範例應用程式作為起點。 會為每一個應用程式提供原始檔及已編譯版本。 請檢閱範例原始碼，並識別建立應用程式 (ConnectionFactory、「連線」、「階段作業」、「目的地」及/或「生產者」或「消費者」) 的每一個必要物件的主要步驟，以及設定指定應用程式運

作方式所需的任何特定內容。 如需相關資訊，請參閱 第 530 頁的『撰寫 XMS 應用程式』。 這些範例在未來的 XMS 版本中可能會變更。

下表顯示 XMS 隨附的範例應用程式集 (每一個 API 各一個)。

表 *81: XMS .NET* 的範例應用程式	
範例名稱	**說明**
SampleConsumerCS	從佇列取得訊息或訂閱主題的訊息消費者應用程式。
SampleProducerCS	一種訊息產生者應用程式，可將訊息產生至佇列或主題。
SampleConfigCS	可用來建立檔案型受管理物件儲存庫的配置應用程式。 應用程式包含特定連線設定的 Connection Factory 和目的地。 然後，此受管理物件儲存庫可以與每一個範例消費者及生產者應用程式搭配使用。

在各種 API 中支援相同功能的範例具有語法差異。

- 訊息消費者和生產者應用程式範例都支援下列功能:
  - 與 IBM MQ、 IBM Integration Bus (使用與分配管理系統的即時連線) 及 WebSphere Application Server service integration bus 的連線
  - 使用起始環境定義介面來查閱受管理物件儲存庫
  - 與佇列 (IBM MQ 和 WebSphere Application Server service integration bus) 及主題 (IBM MQ、與分配管理系統的即時連線及 WebSphere Application Server service integration bus) 的連線
  - 基本、位元組、對映、物件、串流及文字訊息
- 範例訊息消費者應用程式支援同步及非同步接收模式，以及 SQL Selector 陳述式。
- 範例訊息產生者應用程式支援持續性和非持續性遞送模式。

範例可以兩種模式之一運作:

簡式模式
　您可以使用最少使用者輸入來執行範例。

進階模式
　您可以更精細地自訂範例的操作方式。

所有範例都是相容的，因此可以跨語言運作。

Windows 從 IBM MQ 9.1.1 開始， IBM MQ 在 Windows 環境中支援 .NET Core for XMS .NET 應用程式。 依預設， IBM MQ classes for .NET Standard (包括範例) 會隨標準 IBM MQ 安裝一起安裝。

Linux 從 IBM MQ 9.1.2 開始， IBM MQ 也支援 Linux 環境中應用程式的 .NET Core。

XMS .NET 的範例應用程式安裝在 `&MQINSTALL_PATH&/samp/dotnet/samples/cs/core/xms` 中。

如需相關資訊，請參閱 第 519 頁的『正在安裝 IBM MQ classes for XMS .NET』。

執行 .NET 範例應用程式

您可以在簡式或進階模式下以互動方式執行 .NET 範例應用程式，或使用自動產生或自訂的回應檔以非互動方式執行。

開始之前

在執行任何提供的範例應用程式之前，您必須先設定傳訊伺服器環境，讓應用程式可以連接伺服器。 請參閱 第 523 頁的『設定傳訊伺服器環境』。

程序

若要執行 .NET 範例應用程式，請完成下列步驟:

提示: 當您執行範例應用程式時，請鍵入? 隨時可以獲得接下來要做什麼的協助。

1. 選取您要執行範例應用程式的模式。

   鍵入 Advanced 或 Simple。

2. 回答問題。

   若要選取預設值 (顯示在問題結尾的方括弧中)，請按 Enter 鍵。 若要選取不同的值，請鍵入適當的值，然後按 Enter 鍵。

   以下是範例問題:

   ```
   Enter connection type [wpm]:
   ```

   在此情況下，預設值為 wpm (與 WebSphere Application Server service integration bus 的連線)。

結果

當您執行範例應用程式時，會在現行工作目錄中自動產生回應檔。 回應檔名稱的格式為 *connection_type-sample_type*.rsp; 例如 wpm-producer.rsp。 必要的話，您可以使用產生的回應檔，以相同的選項重新執行範例應用程式，以便您不必重新輸入選項。

相關工作

建置 .NET 範例應用程式
當您建置範例 .NET 應用程式時，會建立所選範例的執行檔版本。

建置您自己的應用程式
您可以建置自己的應用程式，就像您建置範例應用程式一樣。

建置 .NET 範例應用程式

當您建置範例 .NET 應用程式時，會建立所選範例的執行檔版本。

開始之前

安裝適當的編譯器。 這項作業假設您已安裝 Microsoft Visual Studio 2012，且您熟悉如何使用它。

程序

若要建置 .NET 範例應用程式，請完成下列步驟:

1. 按一下 .NET 範例隨附的 Samples.sln 解決方案檔案。
2. 在「解決方案瀏覽器」視窗中，用滑鼠右鍵按一下解決方案 **範例**，然後選取 **建置解決方案**。

結果

視您選擇的配置而定，會在範例的適當子資料夾 (bin/Debug 或 bin/Release) 中建立可執行程式。 此程式具有與資料夾相同的名稱，字尾為 CS。 例如，如果您要建置訊息產生者範例應用程式的 C# 版本，則會在 SampleProducer 資料夾中建立 SampleProducerCS.exe 。

相關工作

執行 .NET 範例應用程式
您可以在簡式或進階模式下以互動方式執行 .NET 範例應用程式，或使用自動產生或自訂的回應檔以非互動方式執行。

建置您自己的應用程式
您可以建置自己的應用程式，就像您建置範例應用程式一樣。

第 529 頁的『建置您自己的應用程式』
您可以建置自己的應用程式，就像您建置範例應用程式一樣。

建置您自己的應用程式

您可以建置自己的應用程式，就像您建置範例應用程式一樣。

開始之前

安裝適當的編譯器。 這項作業假設您已安裝 Microsoft Visual Studio 2012 ， 且您熟悉如何使用它。

程序

- 建置 .NET 應用程式，如 第 529 頁的『建置 .NET 範例應用程式』中所述。
  如需如何建置您自己的應用程式的其他指引，請使用為每一個範例應用程式提供的 make 檔。

  提示: 如果要在失敗時協助診斷問題，您可能會發現編譯包含符號的應用程式會很有用。

相關工作

執行 .NET 範例應用程式
您可以在簡式或進階模式下以互動方式執行 .NET 範例應用程式，或使用自動產生或自訂的回應檔以非互動方式執行。

建置 .NET 範例應用程式
當您建置範例 .NET 應用程式時，會建立所選範例的執行檔版本。

撰寫 XMS 應用程式

本節中的主題提供資訊，以協助您撰寫一般的 XMS 應用程式。

關於這項作業

本節包含撰寫 XMS 應用程式的一般概念。 如需建立 XMS .NET 應用程式的特定資訊，另請參閱 第 545 頁的『撰寫 XMS .NET 應用程式』。

從 IBM MQ 9.2.0 開始，指 XMS.NET 動態鏈結程式庫已大幅減少，總計五個。 這五個動態鏈結程式庫如下:

- `IBM.XMS.dll` -包括所有國家語言訊息
- `IBM.XMS.Comms.RMM.dll`
- 三個原則動態鏈結程式庫:
  - `policy.8.0.IBM.XMS.dll`
  - `policy.9.0.IBM.XMS.dll`
  - `policy.9.1.IBM.XMS.dll`

V 9.3.0 Removed V 9.3.0 XMS .NET 多重播送傳訊（使用 RMM）已從 IBM MQ 9.2 中淘汰，並在 IBM MQ 9.3 中移除。

本節包含下列主題:

- 第 531 頁的『執行緒作業模型』
- 第 532 頁的『ConnectionFactories 和 Connection 物件』
- 第 533 頁的『會議』
- 第 535 頁的『目的地』
- 第 537 頁的『訊息產生者』
- 第 538 頁的『訊息消費者』
- 第 540 頁的『佇列瀏覽器』
- 第 541 頁的『要求者』
- 第 541 頁的『物件刪除』
- 第 542 頁的『XMS 初始類型』
- 第 542 頁的『將內容值從一個資料類型隱含轉換為另一個資料類型』
- 第 544 頁的『反覆運算子』
- 第 544 頁的『編碼字集 ID』
- 第 544 頁的『XMS 錯誤碼和異常狀況碼』

- 第 529 頁的『建置您自己的應用程式』

Windows 使用 IBM MQ XMS .NET 專案範本

IBM MQ XMS .NET 用戶端可讓您使用專案範本來協助您開發 XMS .NET Core 應用程式。

開始之前

您必須在系統上具有 Microsoft Visual Studio 2017 或更新版本，以及 .NET Core 2.1 。

您必須從下列項目複製 XMS .NET 範本:

&MQ_INSTALL_ROOT&\tools\dotnet\samples\cs\core\xms\ProjectTemplates\IBMXMS.NETClientApp.zip

目錄至

&USER_HOME_DIRECTORY&\Documents\&Visual_Studio_Version&\Templates\ProjectTemplates

目錄，其中:

- *&MQ_INSTALL_ROOT* 是安裝的根目錄
- *&USER_HOME_DIRECTORY* 是您的起始目錄。

您必須停止並重新啟動 Microsoft Visual Studio ，才能挑選範本。

關於這項作業

XMS .NET 專案範本包含一些通用代碼，可用來協助開發應用程式。 使用內建程式碼，您可以連接至 IBM MQ 佇列管理程式，並只需修改內建程式碼中的內容，即可執行 put 或 get 作業。

程序

1. 開啟 Microsoft Visual Studio。
2. 按一下 **檔案**，接著按一下 **新建** ，然後按一下 **專案**。
3. 在 「建立新專案」視窗中，選取 IBM XMS .NET Client App (.NET Core) ，然後按 **下一步**。
4. 在「 配置新專案 」視窗中，變更專案的 專案名稱 (如果您想要的話) ，然後按一下 **建立** 以建立 XMS .NET 專案。

   XMSDotnetApp.cs 是與專案檔一起建立的檔案。 此檔案包含連接至佇列管理程式並執行傳送及接收作業的程式碼。

   連線內容會設為預設值:

   - WMQ_CONNECTION_NAME_LIST 設為 *localhost (1414)*
   - XMSC.WMQ_CHANNEL 設為 *DOTNET.SVRCONN*

   佇列設為 *Q1*，您可以相應地修改這些內容。
5. 編譯並執行應用程式。

相關概念

IBM MQ 元件和特性

.NET 應用程式執行時期-僅限 Windows

執行緒作業模型

一般規則控管多執行緒應用程式如何使用 XMS 物件。

- 在不同執行緒上只能同時使用下列類型的物件:
  - ConnectionFactory

  - 連線
  - ConnectionMeta 資料
  - 目的地
- Session 物件一次只能在單一執行緒上使用。

在 IBM Message Service Client for .NET 參照中，方法的介面定義中標示為 "執行緒環境定義" 的項目會指出這些規則的異常狀況。

ConnectionFactories 和 Connection 物件

ConnectionFactory 物件提供應用程式用來建立 Connection 物件的範本。 應用程式使用 Connection 物件來建立 Session 物件。

對於 .NET， XMS 應用程式會先使用 XMSFactoryFactory 物件來取得 ConnectionFactory 物件的參照，該物件適用於所需的通訊協定類型。 然後，這個 ConnectionFactory 物件只能產生該通訊協定類型的連線。

XMS 應用程式可以建立多個連線，而多執行緒應用程式可以在多個執行緒上同時使用單一 Connection 物件。 Connection 物件會封裝應用程式與傳訊伺服器之間的通訊連線。

連線有數個用途:

- 當應用程式建立連線時，可以鑑別應用程式。
- 應用程式可以將唯一用戶端 ID 與連線相關聯。 用戶端 ID 用來支援發佈/訂閱網域中的可延續訂閱。 有兩種方式可以設定用戶端 ID:

  指派連線用戶端 ID 的偏好方式，是使用內容在用戶端特定的 ConnectionFactory 物件中配置，並將它透通地指派給它所建立的連線。

  指派用戶端 ID 的替代方式是使用在 Connection 物件上設定的提供者特定值。 此值不會置換已透過管理方式配置的 ID。 它是針對不存在管理指定 ID 的情況所提供。 如果管理上指定的 ID 確實存在，嘗試以提供者特定值置換它會導致擲出異常狀況。 如果應用程式明確設定 ID ，則必須在建立連線之後，並在對連線採取任何其他動作之前立即執行; 否則，會擲出異常狀況。

XMS 應用程式通常會建立連線、一個以上階段作業，以及一些訊息產生者和訊息消費者。

就系統資源而言，建立連線相當昂貴，因為它涉及建立通訊連線，而且也可能涉及鑑別應用程式。

連線已啟動和已停止模式

連線可以在已啟動或已停止模式下運作。

當應用程式建立連線時，連線會處於已停止模式。 當連線處於已停止模式時，應用程式可以起始設定階段作業，它可以傳送訊息，但無法同步或非同步接收它們。

應用程式可以呼叫 `Start Connection` 方法來啟動連線。 當連線處於已啟動模式時，應用程式可以傳送及接收訊息。 然後，應用程式可以透過呼叫「停止連線」及 `Start Connection` 方法來停止並重新啟動連線。

連線關閉

應用程式會呼叫「關閉連線」方法來關閉連線。 當應用程式關閉連線時， XMS 會執行下列動作:

- 它會關閉與連線相關聯的所有階段作業，並刪除與這些階段作業相關聯的特定物件。 如需刪除哪些物件的相關資訊，請參閱 第 541 頁的『物件刪除』。 同時， XMS 會回復階段作業內目前進行中的任何交易。
- 它會結束與傳訊伺服器的通訊連線。
- 它會釋放連線所使用的記憶體及其他內部資源。

在關閉連線之前， XMS 不會確認在階段作業期間收到任何它無法確認的訊息。 如需確認接收訊息的相關資訊，請參閱 第 534 頁的『訊息確認通知』。

例外處理

XMS .NET 異常狀況全都衍生自 System.Exception。 如需相關資訊，請參閱 第 548 頁的『.NET 中的錯誤處理』。

服務整合匯流排的連線

XMS 應用程式可以使用直接 TCP/IP 連線或透過 TCP/IP 使用 HTTP 來連接至 WebSphere Application Server 服務整合匯流排。

在無法使用直接 TCP/IP 連線的情況下，可以使用 HTTP 通訊協定。 一個常見的狀況是透過防火牆進行通訊時，例如當兩個企業交換訊息時。 使用 HTTP 透過防火牆進行通訊通常稱為 *HTTP* 通道作業。 不過， HTTP 通道作業本質上比使用直接 TCP/IP 連線來得慢，因為 HTTP 標頭會大幅增加所傳送的資料量，而且因為 HTTP 通訊協定需要比 TCP/IP 更多的通訊流程。

若要建立 TCP/IP 連線，應用程式可以使用其 XMSC_WPM_TARGET_TRANSPORT_CHAIN 內容設為 XMSC_WPM_TARGET_TRANSPORT_CHAIN_BASIC 的 Connection Factory。 這是內容的預設值。 如果順利建立連線，連線的 XMSC_WPM_CONNECTION_PROTOCOL 內容會設為 XMSC_WPM_CP_TCP。

若要建立使用 HTTP 的連線，應用程式必須使用其 XMSC_WPM_TARGET_TRANSPORT_CHAIN 內容設為入埠傳輸鏈 (配置為使用 HTTP 傳輸通道) 名稱的 Connection Factory。 如果順利建立連線，連線的 XMSC_WPM_CONNECTION_PROTOCOL 內容會設為 XMSC_WPM_CP_HTTP。 如需如何配置傳輸鏈的相關資訊，請參閱 WebSphere Application Server 產品說明文件中的 配置傳輸鏈 。

連接至引導伺服器時，應用程式具有類似的通訊協定選項。 Connection Factory 的 XMSC_WPM_PROVIDER_ENDPOINTS 內容是引導伺服器的一或多個端點位址序列。 每一個端點位址的引導傳輸鏈元件可以是 XMSC_WPM_BOOTSTRAP_TCP (對於引導伺服器的 TCP/IP 連線) 或 XMSC_WPM_BOOTSTRAP_HTTP (對於使用 HTTP 的連線)。

會議

階段作業是用於傳送及接收訊息的單一執行緒環境定義。

應用程式可以使用階段作業來建立訊息、訊息產生者、訊息消費者、佇列瀏覽器及暫時目的地。 應用程式也可以使用階段作業來執行區域交易。

應用程式可以建立多個階段作業，其中每一個階段作業會產生及耗用與其他階段作業無關的訊息。 如果在個別階段作業 (或甚至在相同階段作業中) 中有兩個訊息消費者訂閱相同的主題，則每一個都會收到針對該主題發佈的任何訊息副本。

不同於 Connection 物件， Session 物件不能同時在不同的執行緒上使用。 只能從「階段作業」物件當時正在使用的執行緒以外的執行緒呼叫「階段作業」物件的「關閉階段作業」方法。 「關閉階段作業」方法會結束階段作業，並釋放配置給階段作業的任何系統資源。

如果應用程式必須在多個執行緒上同時處理訊息，應用程式必須在每一個執行緒上建立一個階段作業，然後在該執行緒內的任何傳送或接收作業中使用該階段作業。

交易式階段作業

XMS 應用程式可以執行區域交易。 區域交易 是只涉及變更應用程式所連接之佇列管理程式或服務整合匯流排的資源的交易。

只有在應用程式連接至 IBM MQ 佇列管理程式或 WebSphere Application Server 服務整合匯流排時，本主題中的資訊才相關。 此資訊與分配管理系統的即時連線無關。

如果要執行區域交易，應用程式必須先呼叫 Connection 物件的「建立階段作業」方法來建立交易式階段作業，並指定作為交易式階段作業的參數。 隨後，在階段作業內傳送及接收的所有訊息會分組成一連串交易。 當應用程式確定或回復自交易開始以來所傳送及接收的訊息時，交易會結束。

為了確定交易，應用程式會呼叫「階段作業」物件的「確定」方法。 當確定交易時，在交易內傳送的所有訊息都會變成可供遞送給其他應用程式，且會確認交易內收到的所有訊息，以便傳訊伺服器不會再次嘗試將它們遞送給應用程式。 在點對點網域中，傳訊伺服器也會從其佇列中移除收到的訊息。

為了回復交易，應用程式會呼叫「階段作業」物件的「回復」方法。 當回復交易時，傳訊伺服器會捨棄交易內所傳送的所有訊息，且交易內所接收的所有訊息會重新可供遞送。 在點對點網域中，接收到的訊息會放回其佇列中，並再次對其他應用程式可見。

當應用程式建立交易式階段作業或呼叫「確定」或「回復」方法時，新交易會自動啟動。 因此，交易式階段作業一律具有作用中交易。

當應用程式關閉交易式階段作業時，會進行隱含回復。 當應用程式關閉連線時，所有連線的交易階段作業都會進行隱含回復。

交易完全包含在交易式階段作業內。 交易無法跨越階段作業。 這表示應用程式無法在兩個以上交易階段作業中傳送及接收訊息，然後將所有這些動作確定或回復為單一交易。

相關概念

訊息確認通知
每一個未交易的階段作業都有確認通知模式，可決定如何確認應用程式收到的訊息。 有三種確認模式可用，而選擇確認模式會影響應用程式的設計。

訊息傳遞
XMS 支援訊息遞送的持續及非持續模式，以及訊息的非同步及同步遞送。

訊息確認通知

每一個未交易的階段作業都有確認通知模式，可決定如何確認應用程式收到的訊息。 有三種確認模式可用，而選擇確認模式會影響應用程式的設計。

只有在應用程式連接至 IBM MQ 佇列管理程式或 WebSphere Application Server 服務整合匯流排時，本主題中的資訊才相關。 此資訊與分配管理系統的即時連線無關。

XMS 使用相同的機制來確認 JMS 使用的訊息接收。

如果階段作業未進行交易，則由階段作業的確認模式決定應用程式所收到訊息的確認方式。 下列段落說明三種確認模式:

XMSC_AUTO_認可
階段作業會自動確認應用程式收到的每一則訊息。

如果訊息同步遞送至應用程式，階段作業會在每次「接收」呼叫順利完成時確認接收訊息。

如果應用程式順利接收訊息，但失敗阻止發生確認通知，則訊息會再次變成可供遞送。 因此，應用程式必須能夠處理重新遞送的訊息。

XMSC_DUPS_OK_認可
階段作業會確認應用程式在選取時所收到的訊息。

使用此確認模式可減少階段作業必須執行的工作量，但阻止訊息確認的失敗可能會導致多個訊息重新可供遞送。 因此，應用程式必須能夠處理重新遞送的訊息。

XMSC_CLIENT_認可
應用程式透過呼叫 Message 類別的 Acknowledge 方法來確認它收到的訊息。

應用程式可以個別確認接收每一個訊息，也可以接收一批訊息，並僅針對它接收的最後一則訊息呼叫 Acknowledge 方法。 當呼叫 Acknowledge 方法時，會確認自前次呼叫方法以來收到的所有訊息。

與任何這些確認模式一起使用時，應用程式可以透過呼叫「階段作業」類別的「回復」方法，停止並重新啟動階段作業中的訊息遞送。 重新遞送其回執先前未確認的訊息。 不過，它們可能不會以先前遞送的相同順序來遞送。 同時，較高優先順序的訊息可能已到達，部分原始訊息可能已過期。 在點對點網域中，部分原始訊息可能已由另一個應用程式使用。

應用程式可以檢查訊息的 JMSRedelivered 標頭欄位內容，來判斷是否正在重新遞送訊息。 應用程式透過呼叫 Message 類別的 Get JMSRedelivered 方法來執行此動作。

相關概念

交易式階段作業
XMS 應用程式可以執行區域交易。 區域交易 是只涉及變更應用程式所連接之佇列管理程式或服務整合匯流排的資源的交易。

訊息傳遞

XMS 支援訊息遞送的持續及非持續模式，以及訊息的非同步及同步遞送。

訊息傳遞

XMS 支援訊息遞送的持續及非持續模式，以及訊息的非同步及同步遞送。

訊息遞送模式

XMS 支援兩種訊息遞送模式:

持續
持續訊息會遞送一次。 傳訊伺服器會採取特殊預防措施 (例如記載訊息)，以確保持續訊息不會在傳輸中遺失，即使發生失敗也一樣。

非持續性
非持續訊息只會遞送一次。 非持續訊息比持續訊息更不可靠，因為一旦失敗，它們可能會在傳輸中遺失。

遞送模式的選擇是在可靠性與效能之間進行取捨。 非持續訊息的傳輸速度通常比持續訊息快。

非同步訊息遞送

XMS 使用一個執行緒來處理階段作業的所有非同步訊息遞送。 這表示一次只能執行一個訊息接聽器函數或一個 `onMessage()` 方法。

如果階段作業中有多個訊息消費者非同步接收訊息，且訊息接聽器功能或 `onMessage()` 方法正在將訊息遞送至訊息消費者，則等待相同訊息的任何其他訊息消費者都必須繼續等待。 等待遞送至階段作業的其他訊息也必須繼續等待。

如果應用程式需要並行遞送訊息，請建立多個階段作業，讓 XMS 使用多個執行緒來處理非同步訊息遞送。 如此一來，就可以同時執行多個訊息接聽器函數或 `onMessage()` 方法。

將訊息接聽器指派給消費者，不會使階段作業變成非同步。 只有在呼叫 `Connection.Start` 方法時，階段作業才會變成非同步。 在呼叫 `Connection.Start` 方法之前，允許所有同步呼叫。 當呼叫 `Connection.Start` 時，會開始將訊息遞送至消費者。

如果必須在非同步階段作業上進行同步呼叫 (例如建立消費者或生產者)，則必須呼叫 `Connection.Stop` 。 您可以呼叫 `Connection.Start` 方法來開始遞送訊息，以回復階段作業。 唯一例外是「階段作業」訊息遞送執行緒，它是將訊息遞送至回呼函數的執行緒。 此執行緒可以對訊息回呼函數中的階段作業 (「關閉」呼叫除外) 進行任何呼叫。

註: 在未受管理模式中， IBM MQ .NET 用戶端不支援回呼函數內的 MQDISC 呼叫。 因此，在「非同步接收」模式下，用戶端應用程式無法在 MessageListener 回呼內建立或關閉階段作業。 在 `MessageListener` 方法之外建立並刪除階段作業。

同步訊息遞送

如果應用程式使用 MessageConsumer 物件的「接收」方法，則訊息會同步遞送至應用程式。

使用「接收」方法，應用程式可以等待指定的訊息一段時間，也可以無限期地等待。 或者，如果應用程式不想等待訊息，則可以使用「不等待接收」方法。

相關概念
交易式階段作業
XMS 應用程式可以執行區域交易。 區域交易 是只涉及變更應用程式所連接之佇列管理程式或服務整合匯流排的資源的交易。

訊息確認通知
每一個未交易的階段作業都有確認通知模式，可決定如何確認應用程式收到的訊息。 有三種確認模式可用，而選擇確認模式會影響應用程式的設計。

目的地

XMS 應用程式使用「目的地」物件來指定所傳送訊息的目的地，以及所接收訊息的來源。

XMS 應用程式可以在執行時期建立「目的地」物件，或從受管理物件的儲存庫取得預先定義的目的地。

如同 ConnectionFactory， XMS 應用程式指定目的地的最靈活方式是將它定義為受管理物件。 使用此方法，以 C、C++ 及 .NET 語言及 Java 撰寫的應用程式可以共用目的地的定義。 可以在不變更任何程式碼的情況下變更受管理目的地物件的內容。

對於 .NET 應用程式，您可以使用 CreateTopic 或 CreateQueue 方法來建立目的地。 在 .NET API 的 ISession 和 XMSFactoryFactory 物件中提供這兩種方法。 如需相關資訊，請參閱第 547 頁的『.NET 中的目的地』及../refdev/sapidest.dita#sapidest。

主題統一資源識別碼

主題統一資源識別碼 (URI) 指定主題的名稱; 它也可以為它指定一或多個內容。

主題的 URI 以序列 topic: // 開頭，後面接著主題的名稱及 (選用) 設定其餘主題內容的名稱/值配對清單。 主題名稱不能是空的。

以下是 .NET 程式碼片段中的範例:

```
topic = session.CreateTopic("topic://Sport/Football/Results?multicast=7");
```

如需主題內容 (包括可在 URI 中使用的名稱及有效值) 的相關資訊，請參閱 目的地的內容。

指定要在訂閱中使用的主題 URI 時，可以使用萬用字元。 這些萬用字元的語法視連線類型及分配管理系統版本而定; 可以使用下列選項:

- WebSphere Application Server 服務整合匯流排

WebSphere Application Server 服務整合匯流排

WebSphere Application Server 服務整合匯流排使用下列萬用字元:

* to match any characters at one level in the hierarchy
// to match 0 or more levels
//. to match 0 or more levels (at the end of a Topic expression)

第 536 頁的表 82 提供一些如何使用這個萬用字元架構的範例。

表 *82: WebSphere Application Server* 服務整合匯流排使用萬用字元架構的 *URI* 範例

統一資源識別碼 (URI)	符合	範例
"topic://Sport/ * ball/Results"	Sport 與 Results 之間具有單一階層式層次名稱以 "ball" 結尾的所有主題	"topic://Sport/Football/Results" 及 "topic://Sport/Netball/Results"
"topic://Sport//Results"	以 "Sport/" 開頭且以 "/Results" 結尾的所有主題	"topic://Sport/Football/Results" 及 "topic://Sport/Hockey/National/Div3/Results"
"topic://Sport/Fooball//."	以 "Sport/Football/" 開頭的所有主題	"topic://Sport/Fooball/Results" 及 "topic://Sport/Fooball/TeamNews/Signings/Managerial"
"topic://Sport/ * ball//Results//."	主題	"topic://Sport/Football/Results" 及 "topic://Sport/Netball/National/Div3/Results/2002/November"

相關概念

佇列統一資源識別碼
佇列的 URI 指定佇列的名稱; 它也可以指定佇列的一或多個內容。

暫時目的地
XMS 應用程式可以建立及使用暫時目的地。

佇列統一資源識別碼

佇列的 URI 指定佇列的名稱; 它也可以指定佇列的一或多個內容。

佇列的 URI 以順序 queue://開頭，後面接著佇列的名稱; 它也可能包含設定其餘佇列內容的名稱/值配對清單。

對於 IBM MQ 佇列 (但不適用於 WebSphere Application Server 預設傳訊提供者佇列) ，佇列所在的佇列管理程式可以在佇列之前指定，並以/區隔佇列管理程式名稱與佇列名稱。

如果指定佇列管理程式，則必須是 XMS 使用此佇列直接連接的佇列管理程式，或者必須可從這個佇列存取。 僅支援遠端佇列管理程式從佇列擷取訊息，不支援將訊息放入佇列。 如需完整資料，請參閱 IBM MQ 佇列管理程式文件。

如果未指定佇列管理程式，則額外/分隔字元是選用的，且其存在與否與佇列的定義沒有任何差異。

在稱為 QM_A 的佇列管理程式上，下列佇列定義都相等於 XMS 直接連接的 IBM MQ 佇列 QB:

```
queue://QB
queue:///QB
queue://QM_A/QB
```

相關概念

主題統一資源識別碼
主題統一資源識別碼 (URI) 指定主題的名稱; 它也可以為它指定一或多個內容。

暫時目的地
XMS 應用程式可以建立及使用暫時目的地。

暫時目的地

XMS 應用程式可以建立及使用暫時目的地。

應用程式通常會使用暫時目的地來接收要求訊息的回覆。 為了指定要傳送要求訊息之回覆的目的地，應用程式會呼叫代表要求訊息之訊息物件的 Set JMSReplyTo 方法。 呼叫上指定的目的地可以是暫時目的地。

雖然階段作業是用來建立暫時目的地，但暫時目的地的範圍實際上是用來建立階段作業的連線。 任何連線的階段作業都可以建立暫時目的地的訊息產生者和訊息消費者。 暫時目的地會保留到明確刪除或連線結束為止 (以先發生的情況為準)。

當應用程式建立暫時佇列時，會在應用程式所連接的傳訊伺服器中建立佇列。 如果應用程式連接至佇列管理程式，則會從模型佇列建立動態佇列，其名稱由 XMSC_WMQ_TEMPORARY_MODEL 內容指定，且用來形成動態佇列名稱的字首由 XMSC_WMQ_TEMP_Q_PREFIX 內容指定。 如果應用程式連接至服務整合匯流排，則會在匯流排中建立暫時佇列，且 XMSC_WPM_TEMP_Q_PREFIX 內容會指定用來形成暫時佇列名稱的字首。

當連接至服務整合匯流排的應用程式建立暫時主題時， XMSC_WPM_TEMP_TOPIC_PREFIX 內容會指定用來形成暫時主題名稱的字首。

相關概念

主題統一資源識別碼
主題統一資源識別碼 (URI) 指定主題的名稱; 它也可以為它指定一或多個內容。

佇列統一資源識別碼
佇列的 URI 指定佇列的名稱; 它也可以指定佇列的一或多個內容。

訊息產生者

在 XMS 中，可以使用有效的目的地或沒有相關聯的目的地來建立訊息產生者。 建立具有空值目的地的訊息產生者時，在傳送訊息時必須指定有效的目的地。

具有相關聯目的地的訊息產生者

在此實務範例中，會使用有效的目的地來建立訊息產生者。 在傳送作業期間，不需要指定目的地。

沒有相關聯目的地的訊息產生者

在 XMS .NET 中，可以使用空值目的地來建立訊息產生者。

當使用 .NET API 時，如果要建立沒有關聯目的地的訊息產生者，必須將 NULL 當作參數傳遞至 ISession 物件的 `CreateProducer()` 方法 (例如 `session.CreateProducer(null)`)。 不過，在傳送訊息時必須指定有效的目的地。

訊息消費者

訊息消費者可以分類為可延續及不可延續訂閱者，以及同步及非同步訊息消費者。

延續訂閱者

可延續訂閱者是一種訊息消費者，可接收針對主題發佈的所有訊息，包括訂閱者處於非作用中狀態時發佈的訊息。

只有在應用程式連接至 IBM MQ 佇列管理程式或 WebSphere Application Server 服務整合匯流排時，本主題中的資訊才相關。 此資訊與分配管理系統的即時連線無關。

為了建立主題的可延續訂閱者，應用程式會呼叫 Session 物件的 Create Durable Subscriber 方法，並將識別可延續訂閱的名稱和代表主題的 Destination 物件指定為參數。 應用程式可以使用或不使用訊息選取器來建立可延續訂閱者，並且它可以指定可延續訂閱者是否要接收其自己的連線所發佈的訊息。

用來建立可延續訂閱者的階段作業必須有相關聯的用戶端 ID。 用戶端 ID 與用來建立階段作業的連線相關聯的用戶端 ID 相同; 其指定方式如 第 532 頁的『ConnectionFactories 和 Connection 物件』中所述。

識別可延續訂閱的名稱在用戶端 ID 內必須是唯一的，因此用戶端 ID 會構成可延續訂閱完整唯一 ID 的一部分。 傳訊伺服器會維護可延續訂閱的記錄，並確保在主題上發佈的所有訊息都會保留下來，直到可延續訂閱者確認它們或它們到期為止。

即使在可延續訂閱者關閉之後，傳訊伺服器仍會繼續維護可延續訂閱的記錄。 若要重複使用先前建立的可延續訂閱，應用程式必須建立可延續訂閱者，並指定相同的訂閱名稱，並使用與可延續訂閱相關聯且具有相同用戶端 ID 的階段作業。 一次只能有一個階段作業具有特定可延續訂閱的可延續訂閱者。

可延續訂閱的範圍是維護訂閱記錄的傳訊伺服器。 如果連接至不同傳訊伺服器的兩個應用程式各使用相同的訂閱名稱及用戶端 ID 來建立可延續訂閱者，則會建立兩個完全獨立的可延續訂閱。

若要刪除可延續訂閱，應用程式會呼叫「階段作業」物件的「取消訂閱」方法，並將識別可延續訂閱的名稱指定為參數。 與階段作業相關聯的用戶端 ID 必須與與可延續訂閱相關聯的用戶端 ID 相同。 傳訊伺服器會刪除它所維護的可延續訂閱記錄，且不會再傳送任何訊息給可延續訂閱者。

若要變更現有訂閱，應用程式可以使用相同的訂閱名稱及用戶端 ID 來建立可延續訂閱者，但指定不同的主題及/或訊息選取器。 變更可延續訂閱相當於刪除訂閱並建立新的訂閱。

對於連接至 IBM MQ 佇列管理程式的應用程式， XMS 會管理訂閱者佇列。 因此，應用程式不需要指定訂閱者佇列。 XMS 將忽略訂閱者佇列 (如果指定的話)。

請注意，您無法變更可延續訂閱的訂閱者佇列。 變更訂閱者佇列的唯一方法是刪除訂閱並建立新的訂閱。

對於連接至服務整合匯流排的應用程式，每一個可延續訂閱者必須具有指定的可延續訂閱起始目錄。 如果要指定使用相同連線之所有可延續訂閱者的可延續訂閱起始目錄，請設定用來建立連線之 ConnectionFactory 物件的 XMSC_WPM_DUR_SUB_HOME 內容。 若要指定個別主題的可延續訂閱起始目錄，請設定代表主題之 Destination 物件的 XMSC_WPM_DUR_SUB_HOME 內容。 必須先指定連線的可延續訂閱起始目錄，應用程式才能建立使用連線的可延續訂閱者。 指定給目的地的任何值都會置換指定給連線的值。

同步及非同步訊息消費者

同步訊息消費者會同步接收來自佇列的訊息，而非同步訊息消費者會非同步接收來自佇列的訊息。

同步訊息消費者

同步訊息消費者一次接收一則訊息。 當使用 `Receive(wait interval)` 方法時，呼叫只會等待訊息的指定時段 (毫秒) ，或直到訊息消費者關閉為止。

如果使用 ReceiveNoWait () 方法，則同步訊息消費者會接收訊息而不會延遲; 如果下一則訊息可用，則會立即接收它，否則會傳回空值訊息物件的指標。

非同步訊息消費者

每當佇列上有新的訊息可用時，就會呼叫應用程式所登錄的訊息接聽器。

XMS 中的有害訊息

有害訊息是指無法由接收端 MDB 應用程式處理的訊息。 如果發現有害訊息， XMS MessageConsumer 物件可以根據兩個佇列內容 (BOQUEUE 和 BOTHRESH) 將它重新排入佇列。

在某些情況下，遞送至 MDB 的訊息可能會回復到 IBM MQ 佇列。 例如，如果在後續回復的工作單元內遞送訊息，則會發生此情況。 通常會重新遞送已回復的訊息，但格式不正確的訊息可能會反覆地導致 MDB 失敗，因此無法遞送。 這類訊息稱為有害訊息。 您可以配置 IBM MQ ，讓有害訊息自動傳送至另一個佇列，以進一步調查或捨棄。 如需如何以這種方式配置 IBM MQ 的相關資訊，請參閱 處理 ASF 中的有害訊息。

有時，格式不正確的訊息會到達佇列。 在此環境定義中，格式不正確表示接收端應用程式無法正確處理訊息。 這類訊息會導致接收端應用程式失敗，並退出這個格式不正確的訊息。 然後，訊息可以反覆地遞送至輸入佇列，並由應用程式反覆地取消。 這些訊息稱為有害訊息。 XMS MessageConsumer 物件會偵測有毒訊息，並將它們重新遞送至替代目的地。

IBM MQ 佇列管理程式會記錄每一則訊息已取消的次數。 當此數目達到可配置的臨界值時，訊息消費者會將訊息重新排入指名的取消佇列。 如果此重新排入佇列因任何原因而失敗，則會從輸入佇列中移除訊息，並重新排入無法傳送郵件的佇列，或捨棄該訊息。

XMS ConnectionConsumer 物件會以相同的方式並使用相同的佇列內容來處理有毒訊息。 如果多個連線消費者正在監視相同的佇列，則在重新排入佇列之前，有害訊息可能遞送至應用程式的次數可能會超過臨界值。 此行為是由於個別連線消費者監視佇列及重新排入佇列有害訊息的方式。

臨界值及回復佇列的名稱是 IBM MQ 佇列的屬性。 屬性名稱為 BackoutThreshold 及 BackoutRequeue 完整名稱。 它們適用的佇列如下:

- 對於點對點傳訊，這是基礎本端佇列。 當訊息消費者和連線消費者使用佇列別名時，這很重要。
- 對於 IBM MQ 傳訊提供者標準模式中的發佈/訂閱傳訊，它是從中建立主題受管理佇列的模型佇列。
- 對於 IBM MQ 傳訊提供者移轉模式中的發佈/訂閱傳訊，它是在 TopicConnectionFactory 物件上定義的 CCSUB 佇列，或在 Topic 物件上定義的 CCDSUB 佇列。

若要設定 BackoutThreshold 及 BackoutRequeue 完整名稱屬性，請發出下列 MQSC 指令:

```
ALTER QLOCAL(your.queue.name) BOTHRESH(threshold value)
BOQUEUE(your.backout.queue.name)
```

對於發佈/訂閱傳訊，如果您的系統為每一個訂閱建立動態佇列，則會從 IBM MQ classes for JMS 模型佇列 SYSTEM.JMS.MODEL.QUEUE。 若要變更這些設定，請使用:

```
ALTER QMODEL(SYSTEM.JMS.MODEL.QUEUE) BOTHRESH(threshold value)
BOQUEUE(your.backout.queue.name)
```

如果取消臨界值為零，則會停用有害訊息處理，且有害訊息會保留在輸入佇列上。 否則，當取消計數達到臨界值時， 會將訊息傳送至指名的取消佇列。

如果取消計數達到臨界值，但訊息無法跳至取消佇列，則會將訊息傳送至無法傳送郵件的佇列，或如果訊息是非持續性，則會捨棄該訊息。

如果未定義取消佇列，或如果 MessageConsumer 物件無法將訊息傳送至取消佇列，則會發生此狀況。

配置系統以執行有毒訊息處理

XMS .NET 在查詢 **BOTHRESH** 和 **BOQNAME** 屬性時所使用的佇列，取決於所執行的傳訊樣式:

- 對於點對點傳訊，這是基礎本端佇列。 當 XMS .NET 應用程式使用別名佇列或叢集佇列中的訊息時，這很重要。
- 對於發佈/訂閱傳訊，會建立受管理佇列來保留應用程式的訊息。 XMS .NET 會查詢受管理佇列，以判定 **BOTHRESH** 及 **BOQNAME** 屬性的值。

受管理佇列是從與應用程式已訂閱之「主題」物件相關聯的模型佇列建立，並從模型佇列繼承 **BOTHRESH** 及 **BOQNAME** 屬性的值。 使用的模型佇列取決於接收端應用程式是否已取出可延續或不可延續訂閱:

- 用於可延續訂閱的模型佇列由主題的 **MDURMDL** 屬性指定。 此屬性的預設值為 SYSTEM.DURABLE.MODEL.QUEUE。
- 對於不可延續訂閱，使用的模型佇列由 **MNDURMDL** 屬性指定。 **MNDURMDL** 屬性的預設值是 SYSTEM.NDURABLE.MODEL.QUEUE。

查詢 **BOTHRESH** 和 **BOQNAME** 屬性時， XMS .NET:

- 開啟本端佇列或別名佇列的目標佇列。
- 查詢 **BOTHRESH** 及 **BOQNAME** 屬性。
- 關閉本端佇列或別名佇列的目標佇列。

開啟本端佇列或別名佇列的目標佇列時所使用的開啟選項，視所使用的 IBM MQ 版本而定:

- 若為 IBM MQ 9.1.0 Fix Pack 4 Long Term Support 及更早版本，以及 IBM MQ 9.1.4 Continuous Delivery 及更早版本: 如果本端佇列或別名佇列的目標佇列是叢集佇列，則 XMS .NET 會使用 MQOO_INPUT_AS_Q_DEF、 MQOO_INQUIRE 及 MQOO_FAIL_IF_QUIESCING 選項來開啟佇列。 這表示執行接收端應用程式的使用者必須具有叢集佇列本端實例的查詢及存取權。

  XMS .NET 會使用開啟選項 MQOO_INQUIRE 及 MQOO_FAIL_IF_QUIESCING 開啟所有其他類型的本端佇列。 為了讓 XMS .NET 查詢屬性的值，執行接收端應用程式的使用者必須對本端佇列具有查詢存取權。
- 使用 IBM MQ 9.1.5 和 IBM MQ 9.1.0 Fix Pack 5 中的 XMS .NET 時，不論佇列類型為何，執行接收端應用程式的使用者必須對本端佇列具有查詢存取權。

若要將有害訊息移至取消佇列或佇列管理程式的無法傳送郵件佇列，您必須授與執行應用程式 put 及 passall 權限的使用者。

處理 *ASF* 中的有害訊息
當您使用「應用程式伺服器機能 (ASF)」時， ConnectionConsumer(而非 MessageConsumer) 會處理有毒訊息。 ConnectionConsumer 會根據佇列的 BackoutThreshold 及 BackoutRequeue 完整名稱內容來要求訊息。

當應用程式使用 ConnectionConsumers 時，取消訊息的情況取決於應用程式伺服器提供的階段作業:

- 當階段作業非交易式時，若有 AUTO_ANKACK 或 DUPS_OK_ANKACK ，則只有在系統錯誤之後，或應用程式非預期地終止時，才會取消訊息。
- 當階段作業與 CLIENT_ACKNOWLEDGE 非交易時，呼叫 Session.recover()的應用程式伺服器可以取消未確認的訊息。

  一般而言， MessageListener 或應用程式伺服器的用戶端實作會呼叫 Message.acknowledge()。 Message.acknowledge() 確認到目前為止在階段作業上遞送的所有訊息。
- 當交易階段作業時，呼叫 Session.rollback()的應用程式伺服器可以取消未確認的訊息。

佇列瀏覽器

應用程式使用佇列瀏覽器來瀏覽佇列上的訊息，而不移除它們。

若要建立佇列瀏覽器，應用程式會呼叫 ISession 物件的「建立佇列瀏覽器」方法，並將目的地物件指定為參數，以識別要瀏覽的佇列。 應用程式可以使用或不使用訊息選取器來建立佇列瀏覽器。

建立佇列瀏覽器之後，應用程式可以呼叫 IQueueBrowser 物件的 GetEnumerator 方法，以取得佇列上的訊息清單。 此方法會傳回封裝「訊息」物件清單的列舉元。 清單中「訊息」物件的順序與從佇列擷取訊息的順序相同。 然後，應用程式可以依序使用列舉元來瀏覽每一個訊息。

當訊息放置在佇列上並從佇列中移除時，會動態更新列舉元。 每次應用程式呼叫 IEnumerator.MoveNext() 來瀏覽佇列上的下一個訊息時，該訊息會反映佇列的現行內容。

應用程式可以針對給定的佇列瀏覽器多次呼叫 GetEnumerator 方法。 每一個呼叫都會傳回新的列舉元。 因此，應用程式可以使用多個列舉元來瀏覽佇列上的訊息，並在佇列內維護多個位置。

應用程式可以使用佇列瀏覽器來搜尋要從佇列中移除的適當訊息，然後使用具有訊息選取器的訊息消費者來移除訊息。 訊息選取器可以根據 JMSMessageID 標頭欄位的值來選取訊息。 如需此及其他 JMS 訊息標頭欄位的相關資訊，請參閱 第 559 頁的『XMS 訊息中的標頭欄位』。

要求者

應用程式使用要求者來傳送要求訊息，然後等待及接收回覆。

許多傳訊應用程式都是以傳送要求訊息然後等待回覆的演算法為基礎。 XMS 提供稱為 Requestor 的類別，以協助開發這種應用程式樣式。

為了建立要求者，應用程式會呼叫 Requestor 類別的 Create Requestor 建構子，並指定 Session 物件和 Destination 物件作為參數，以識別要傳送要求訊息的位置。 階段作業不得交易，也不得具有 XMSC_CLIENT_ACKNOWLEDGE 確認模式。 建構子會自動建立要傳送回覆訊息的暫時佇列或主題。

建立要求者之後，應用程式可以呼叫 Requestor 物件的 Request 方法來傳送要求訊息，然後等待並接收來自接收要求訊息之應用程式的回覆。 呼叫會等待直到收到回覆或直到階段作業結束 (以先出現者為準)。 對於每一個要求訊息，要求者只需要一個回覆。

當應用程式關閉要求者時，會刪除暫時佇列或主題。 不過，相關聯的階段作業不會關閉。

物件刪除

當應用程式刪除它所建立的 XMS 物件時， XMS 會釋放已配置給該物件的內部資源。

當應用程式建立 XMS 物件時， XMS 會將記憶體及其他內部資源配置給物件。 XMS 會保留這些內部資源，直到應用程式透過呼叫物件的關閉或刪除方法明確刪除物件為止，此時 XMS 會釋放內部資源。 如果應用程式嘗試刪除已刪除的物件，則會忽略呼叫。

當應用程式刪除「連線」或「階段作業」物件時， XMS 會自動刪除某些相關聯的物件，並釋放其內部資源。 這些物件是由「連線」或「階段作業」物件所建立，且沒有與物件無關的功能。 這些物件顯示在 第 541 頁的表 83 中。

註: 如果應用程式關閉與相依階段作業的連線，則也會刪除所有相依於那些階段作業的物件。 只有「連線」或「階段作業」物件可以有相依物件。

表 *83:* 自動刪除的物件

已刪除物件	**方法**	**自動刪除的相依物件**
連線	關閉連線	ConnectionMeta 資料和階段作業物件
Session	關閉階段作業	MessageConsumer、 MessageProducer、 QueueBrowser 及 Requestor 物件

透過 XMS 管理 IBM MQ XA 交易

受管理 IBM MQ XA 交易可以透過 XMS 使用。

若要透過 XMS 使用 XA 交易，必須建立交易式階段作業。 當使用 XA 交易時，交易控制是透過「分散式交易協調程式 (DTC)」廣域交易，而不是透過 XMS 階段作業。 使用 XA 交易時，無法在 XMS 階段作業上發出 `Session.commit` 或 `Session.rollback`。 請改用 `Transscope.Commit` 或 `Transscope.Rollback` DTC 方法來確定或回復交易。 如果階段作業用於 XA 交易，則使用階段作業建立的生產者或消費者必須是 XA 交易的一部分。 它們無法用於 XA 交易範圍以外的任何作業。 它們無法用於 XA 交易外部的作業，例如 `Producer.send` 或 `Consumer.receive`。

在下列情況下，會擲出 `IllegalStateException` 異常狀況物件:

- XA 交易式階段作業用於 `Session.commit` 或 `Session.rollback`。
- XA 交易階段作業中使用過一次的生產者或消費者物件會在 XA 交易範圍之外使用。

非同步消費者不支援 XA 交易。

註:

1. 在 XA 交易確定之前，可能會在 Producer、 Consumer、 Session 或 Connection 物件上發出關閉。 在這種情況下，會回復交易中的訊息。 同樣地，如果在 XA 交易確定之前中斷連線，則會回復交易中的所有訊息。 對於 Producer 物件，回復表示訊息不會放置在佇列上。 對於 Consumer 物件，回復表示訊息保留在佇列上。
2. 如果 Producer 物件將具有 TimeToLive 的訊息放置在 TransactionScope 中，並且在經歷時間之後發出 commit，則該訊息可能會在發出 commit 之前到期。 在此情況下，訊息無法供 Consumer 物件使用。
3. 執行緒之間不支援 Session 物件。 不支援使用與執行緒之間共用的 Session 物件的交易。

XMS 初始類型

XMS 提供八個 Java 初始類型 (byte、short、int、long、float、double、char 及 boolean) 的對等項目。 這容許在 XMS 與 JMS 之間交換訊息，而不會失去或毀損資料。

第 542 頁的表 84 列出 Java 相等的資料類型、大小，以及每一個 XMS 初始類型的最小值和最大值。

表 *84: XMS* 資料類型及其 *Java* 對等項目

XMS 資料類型	**相容 Java 資料類型**	**大小**	**最小值**	**最大值**
System.Boolean	布林值	32 位元	false	true
System.SBYTE	位元組	8 位元	-2^7 (-128)	2^7-1 (127)
System.BYTE	位元組	8 位元	-2^7 (-128)	2^7-1 (127)
System.CHAR	位元組	8 位元	-2^7 (-128)	2^7-1 (127)
System.Int16	短	16 位元	-2^{15} (-32768)	2^{15}-1 (32767)
System.Int32	整數	32 位元	-2^{31} (-2147483648)	2^{31}-1 (2147483647)
System.Int64	長整數	64 位元	-2^{63} (-9223372036854775808)	2^{63}-1 (9223372036854775807)
System.Single	浮點數	32 位元	-3.402823E-38 (至 7 位數精準度)	3.402823E+38 (至 7 位數精準度)
System.Double	倍精準數	64 位元	-1.79769313486231E-308 (至 15 位數精準度)	1.79769313486231E+308 (到 15 位數精準度)

將內容值從一個資料類型隱含轉換為另一個資料類型

當應用程式取得內容的值時， XMS 可以將該值轉換為另一個資料類型。 許多規則會控管支援哪些轉換，以及 XMS 如何執行轉換。

物件的內容具有名稱和值; 值具有相關聯的資料類型，其中內容的值也稱為 內容類型。

應用程式使用 PropertyContext 類別的方法來取得和設定物件的內容。 為了取得內容的值，應用程式會呼叫適用於內容類型的方法。 例如，若要取得整數內容的值，應用程式通常會呼叫 GetIntProperty 方法。

不過，當應用程式取得內容的值時， XMS 可以將該值轉換成另一個資料類型。 例如，若要取得整數內容的值，應用程式可以呼叫 GetStringProperty 方法，它會以字串形式傳回內容的值。 第 542 頁的表 85 顯示 XMS 支援的轉換。

表 *85:* 支援從內容類型到其他資料類型的轉換

內容類型	**支援的目標資料類型**
System.String	System.Boolean, System.Double, System.Float, System.Int32, System.Int64, System.SByte, System.Int16
System.Boolean	System.String, System.SByte, System.Int32, System.Int64, System.Int16

表 *85:* 支援從內容類型到其他資料類型的轉換 *(*繼續*)*	
內容類型	**支援的目標資料類型**
System.Char	System.String
System.Double	System.String
System.Float	System.String, System.Double
System.Int32	System.String, System.Int64
System.Int64	System.String
System.SByte	System.String, System.Int32, System.Int64, System.Int16
System.SByte 陣列	System.String
System.Int16	System.String, System.Int32, System.Int64

下列一般規則控管支援的轉換:

- 如果在轉換期間未遺失任何資料，則可以將數值內容值從一個資料類型轉換為另一個資料類型。 例如，資料類型為 System.Int32 的內容值可以轉換成資料類型為 System.Int64 的值，但無法轉換成資料類型為 System.Int16 的值。
- 任何資料類型的內容值都可以轉換成字串。
- 字串內容值可以轉換為任何其他資料類型，前提是字串已正確格式化以進行轉換。 如果應用程式嘗試轉換未正確格式化的字串內容值， XMS 可能會傳回錯誤。
- 如果應用程式嘗試不支援的轉換， XMS 可能會傳回錯誤。

當內容值從一種資料類型轉換為另一種資料類型時，會套用下列規則:

- 將布林內容值轉換成字串時，值 true 會轉換成字串 "true"，值 false 會轉換成字串 "false"。
- 將布林內容值轉換為數值資料類型 (包括 System.SByte) 時，值 true 會轉換為 1 ，而值 false 會轉換為 0。
- 將字串內容值轉換成布林值時，字串 "true" (不區分大小寫) 或 "1" 會轉換成 true ，而字串 "false" (不區分大小寫) 或 "0" 會轉換成 false。 無法轉換所有其他字串。
- 將字串內容值轉換成資料類型為 System.Int32、 System.Int64、 System.SByte 或 System.Int16 的值時，字串必須具有下列格式:

  [空白] [*sign*]*digits*

  字串元件定義如下:

  空白
  選用前導空白字元。

  符號
  選用加號 (+) 或減號 (-) 字元。

  數字
  一連串連續的數字字元 (0-9)。 至少必須呈現一個數字字元。

  在一連串數字字元之後，字串可以包含非數字字元的其他字元，但一旦達到這些字元的第一個字元，轉換就會停止。 假設字串代表十進位整數。

  如果字串格式不正確， XMS 可能會傳回錯誤。
- 當將字串內容值轉換成資料類型為 System.Double 或 System.Float 的值時，字串必須具有下列格式:

  [空白] [*sign*] [*digits*] [*point*[*d_digits*]] [*e_char*[*e_sign*]*e_digits*]

  字串元件定義如下:

  空白
  (選用) 前導空白字元。

符號
(選用) 加號 (+) 或減號 (-) 字元。

數字
一連串連續的數字字元 (0-9)。 數字 或 *d_digits* 中必須至少存在一個數字字元。

點
(選用) 小數點 (.)。

數字
一連串連續的數字字元 (0-9)。 數字 或 *d_digits* 中必須至少存在一個數字字元。

字元
指數字元，可以是 *E* 或 *e*。

_ 簽署
(選用) 指數的加號 (+) 或減號 (-) 字元。

數字 *(_D)*
指數的連續數字字元 (0-9)。 如果字串包含指數字元，則至少必須存在一個數字字元。

在一連串數字字元或代表指數的選用字元之後，字串可以包含不是數字字元的其他字元，但一旦達到這些字元的第一個字元，就會停止轉換。 假設字串代表具有 10 次方的指數的十進位浮點數字。

如果字串格式不正確， XMS 可能會傳回錯誤。

- 將數值內容值轉換為字串 (包括資料類型為 System.SByte 的內容值) 時，該值會轉換為以十進位數表示的值字串，而不是包含該值的 ASCII 字元的字串。 例如，整數 65 會轉換為字串 "65"，而不是字串 "A"。
- 將位元組陣列內容值轉換為字串時，每個位元組都會轉換為代表位元組的 2 個十六進位字元。 例如，位元組陣列 {0xF1, 0x12, 0x00, 0xFF} 會轉換為字串 "F11200FF"。

Property 和 PropertyContext 類別的方法都支援從內容類型轉換為其他資料類型。

反覆運算子

反覆運算子會封裝物件清單，以及維護清單中現行位置的游標。 「反覆運算子」的概念 (如 IBM MQ Message Service Client (XMS) for C/C++中所提供) 是使用 IBM MQ Message Service Client (XMS) for .NET 中的 IEnumerator 介面來實作。

建立反覆運算子時，游標的位置位於第一個物件之前。 應用程式會使用反覆運算子來依序擷取每一個物件。

IBM MQ Message Service Client (XMS) for C/C++ 的「反覆運算子」類別相當於 Java 中的「列舉元」類別。 IBM MQ Message Service Client (XMS) for .NET 類似於 Java ， 並使用 IEnumerator 介面。

應用程式可以使用 IEnumerator 來執行下列作業:

- 取得訊息的內容
- 取得對映訊息內文中的名稱/值配對
- 瀏覽佇列上的訊息
- 取得連線支援的 JMS 定義訊息內容名稱

編碼字集 ID

在 XMS .NET 中，會使用原生 .NET 字串來傳遞所有字串。 由於這是固定編碼，因此不需要進一步資訊來解譯它。 因此， XMS .NET 應用程式不需要 XMSC_CLIENT_CCSID 內容。

XMS 錯誤碼和異常狀況碼

XMS 使用一系列錯誤碼來指出失敗。 這些錯誤碼未在本文件中明確列出，因為它們可能會因不同版本而不同。 只會記載 XMS 異常狀況碼 (格式為 XMS_X_ ...) ， 因為它們在 XMS 的不同版本之間仍然相同。

透過 XMS 自動重新連線 IBM MQ 用戶端

將 XMS 用戶端配置成在使用 IBM WebSphere MQ 7.1 用戶端以及更新版本作為訊息提供者時，在網路、佇列管理程式或伺服器失敗之後自動重新連接。

使用 MQConnectionFactory 類別的 WMQ_CONNECTION_NAME_LIST 及 WMQ_CLIENT_RECONNECT_OPTIONS 內容，將用戶端連線配置為自動重新連接。 自動用戶端重新連線會在連線失敗之後重新連接用戶端，或在停止佇列管理程式之後作為選項重新連線。 部分用戶端應用程式的設計使得它們不適合自動重新連線。

建立連線之後，可自動重新連接的用戶端連線會變成可重新連接。

註: 也可以透過「用戶端通道定義表 (CCDT)」或透過 mqclient.ini 檔啟用用戶端重新連線，來設定內容 **用戶端重新連接選項**、 **用戶端重新連接逾時**及 **連線名單** 。

註: 如果在 ConnectionFactory 物件上以及 CCDT 中設定重新連線內容，則優先順序規則如下所示。 如果在 ConnectionFactory 物件中設定連線名稱清單內容的預設值，則 CCDT 優先。 如果連線名稱清單未設為其預設值，則 ConnectionFactory 物件中設定的內容值優先。 連線名單的預設值為 localhost(1414)。

撰寫 XMS .NET 應用程式

本節中的主題提供在撰寫 XMS .NET 應用程式時協助您的資訊。

關於這項作業

本節提供撰寫 XMS .NET 應用程式的特定資訊。 如需撰寫 XMS 應用程式的一般資訊，請參閱 第 530 頁的『撰寫 XMS 應用程式』。

本節包含下列主題:

.NET 的資料類型

XMS .NET 支援 System.Boolean、 System.Byte、 System.SByte、 System.Char、 System.String、 System.Single、 System.Double、 System.Decimal、 System.Int16、 System.Int32、 System.Int64、 System.UInt16、 System.UInt32、 System.UInt64、 及 System.Object。 XMS .NET 的資料類型不同於 XMS C/C++ 的資料類型。您可以使用本主題來識別對應的資料類型。

下表顯示對應的 XMS .NET 及 XMS C/C++ 資料類型，並簡要說明它們。

表 *86: XMS .NET* 和 *XMS C/C++* 的資料類型

XMS .NET 類型	**XMS C/C++ 類型**	**說明**
System.SByte	xmsSBYTE xmsINT8	帶正負號的 8 位元值
System.Byte	xmsBYTE xmsUINT8	不帶正負號的 8 位元值
System.Int16	xmsINT16 xmsSHORT	帶正負號的 16 位元值
System.UInt16	xmsUINT16 xmsUSHORT	不帶正負號的 16 位元值

表 *86: XMS .NET* 和 *XMS C/C++* 的資料類型 *(*繼續*)*		
XMS .NET 類型	**XMS C/C++ 類型**	**說明**
System.Int32	xmsINT32 xmsINT	帶正負號的 32 位元值
System.UInt32	xmsUINT32 xmsUINT	不帶正負號的 32 位元值
System.Int64	xmsLONG xmsINT64	帶正負號的 64 位元值
System.UInt64	xmsULONG xmsUINT64	不帶正負號的 64 位元值
System.Char	xmsCHAR16	不帶正負號的 16 位元字元 (Unicode for .NET)
System.Single	xmsFLOAT	IEEE 32 位元浮點數
System.Double	xmsDOUBLE	IEEE 64 位元浮點數
System.Boolean	xmsBOOL	True/False 值
不適用	xmsCHAR	已簽署或未簽署的 8 位元值 (已簽署或未簽署視平台而定)
System.Decimal	不適用	96 位元帶正負號整數乘以 10^0 到 10^{28}
System.Object	不適用	所有類型的基礎
System.String	不適用	字串類型

.NET 中的受管理及未受管理作業

受管理程式碼專門在 .NET 一般語言執行時期環境內執行，完全相依於該執行時期所提供的服務。 如果應用程式的任何部分在 .NET 一般語言執行時期環境之外執行或呼叫服務，則應用程式會分類為未受管理。

受管理 .NET 環境目前無法支援某些進階功能。

如果您的應用程式需要在完全受管理環境中目前不支援的功能，則您可以將應用程式變更為使用未受管理的環境，而不需要對應用程式進行重大變更。 不過，您應該注意，在進行這項選取時， XMS 堆疊會利用未受管理的程式碼。

與 IBM MQ 佇列管理程式的連線

WMQ_CM_CLIENT 的受管理連線不支援非 TCP 通訊及通道壓縮。 不過，可以使用未受管理的連線 (WMQ_CM_CLIENT_UNMANAGED) 來支援這些連線。 如需相關資訊，請參閱 第 464 頁的『開發 .NET 應用程式』。

如果您在未受管理環境中從受管理物件建立 Connection Factory ，則必須手動將連線模式的值變更為 XMSC_WMQ_CM_CLIENT_UNMANAGED。

WebSphere Application Server 服務整合匯流排傳訊引擎的連線

目前不支援連線至需要使用 SSL 通訊協定 (包括 HTTPS) 的 WebSphere Application Server 服務整合匯流排傳訊引擎作為受管理程式碼。

.NET 中的目的地

在 .NET 中，目的地是根據通訊協定類型來建立，且只能在為其建立目的地的通訊協定類型上使用。

提供兩個功能來建立目的地，一個用於主題，另一個用於佇列:

- `IDestination CreateTopic(String topic);`
- `IDestination CreateQueue(String queue);`

這些函數可在 API 中的下列兩個物件上使用:

- ISession
- XMSFactoryFactory

在這兩種情況下，這些方法都可以接受下列格式的 URI 樣式字串 (可包含參數):

```
"topic://some/topic/name?priority=5"
```

或者，這些方法只能接受目的地名稱，即沒有 topic: // 或 queue: // 字首且沒有參數的名稱。

這表示下列 URI 樣式字串:

```
CreateTopic("topic://some/topic/name");
```

會產生與下列目的地名稱相同的結果:

```
CreateTopic("some/topic/name");
```

對於 WebSphere Application Server 服務整合匯流排 JMS，也可以速記形式指定主題，其中包括 *topicname* 和 *topicspace* ，但不能包括參數:

```
CreateTopic("topicspace:topicname");
```

.NET 中的內容

.NET 應用程式使用 PropertyContext 介面中的方法來取得和設定物件的內容。

PropertyContext 介面會封裝取得和設定內容的方法。 這些方法由下列類別直接或間接繼承:

- BytesMessage
- 連線
- ConnectionFactory
- ConnectionMeta 資料
- 目的地
- MapMessage
- 訊息
- MessageConsumer
- MessageProducer
- ObjectMessage
- QueueBrowser
- 階段作業
- StreamMessage
- TextMessage

如果應用程式設定內容的值，新值會取代內容先前的任何值。

如需 XMS 內容的相關資訊，請參閱 XMS 物件的內容。

為了方便使用， XMS 中的 XMS 內容名稱和值預先定義為稱為 XMSC 的結構中的公用常數。 這些常數的名稱格式為 XMSC.*constant*; 例如， XMSC.USERID (內容名稱常數) 和 XMSC.DELIVERY_AS_APP (值常數)。

此外，您可以使用 IBM.XMS.MQC 結構來存取 IBM MQ 常數。 如果是 IBM.XMS 名稱空間，您可以 MQC.*constant* 格式來存取這些內容的值。 例如， MQC.MQRO_COA_WITH_FULL_DATA。

此外，如果您有混合式應用程式同時使用 .NET 的 XMS .NET 和 IBM MQ 類別，且同時匯入 IBM.XMS 和 IBM.WMQ 名稱空間，然後您必須完整限定 MQC 結構名稱空間，以確保每一個出現項目都是唯一的。

受管理 .NET 環境目前不支援部分進階功能。 如需詳細資料，請參閱 第 546 頁的『.NET 中的受管理及未受管理作業』。

.NET 中不存在的內容處理

XMS .NET 中不存在內容的處理與 JMS 規格以及 XMS 的 C 和 C++ 實作大致一致。

在 JMS 中，當方法嘗試將不存在 (空值) 的值轉換成所需的類型時，存取不存在的內容可能會導致 Java 系統異常狀況。 如果內容不存在，則會發生下列異常狀況:

- getString 內容和 getObject 內容傳回空值
- getBoolean 內容傳回 false ，因為 Boolean.valueOf(null) 傳回 false
- getInt 內容 etc.throw java.lang.NumberFormatException ，因為 Integer.valueOf(null) 擲出異常狀況

如果內容不存在於 XMS .NET 中，則會發生下列異常狀況:

- GetString 內容和 GetObject 內容 (以及 GetBytes 內容) 會傳回空值 (與 Java 相同)
- GetBoolean 內容擲出 System.NullReferenceException
- GetInt 內容等會擲出 System.NullReferenceException

此實作與 Java 不同，但與 JMS 規格以及 XMS C 和 C++ 介面大致一致。 如同 Java 實作， XMS .NET 會將 System.Convert 呼叫中的任何異常狀況傳播給呼叫端。 不過，與 Java 不同， XMS 會明確擲出 NullReference 異常狀況，而不只是透過將空值傳遞至系統轉換常式來使用 .NET 架構的原生行為。 如果應用程式將內容設為 "abc" 之類的字串，並呼叫 GetInt 內容，則 Convert.ToInt32("abc") 會擲出 System.FormatException 會延伸到呼叫程式，其與 Java 一致。 MessageFormat 只有在用於 setProperty 和 getProperty 的類型不相容時，才會擲出異常狀況。 此行為也與 Java 一致。

.NET 中的錯誤處理

XMS .NET 異常狀況全都衍生自 System.Exception。 XMS 方法呼叫可能會擲出特定的 XMS 異常狀況，例如 MessageFormat 異常狀況、一般 XMSExceptions ，或系統異常狀況，例如 NullReference 異常狀況。

撰寫應用程式以根據應用程式需求，在特定 catch 區塊或一般 `System.Exception` catch 區塊中捕捉任何這些錯誤。

在 .NET 中使用訊息和異常狀況接聽器

.NET 應用程式使用訊息接聽器來非同步接收訊息，並使用異常狀況接聽器來非同步通知連線發生問題。

關於這項作業

對於 .NET 和 C++ ，訊息和異常狀況接聽器的功能是相同的。不過，有一些小小的實作差異。

程序

- 若要設定訊息接聽器以非同步接收訊息，請完成下列步驟:
  a) 定義符合訊息接聽器委派簽章的方法。
  您定義的方法可以是靜態或實例方法，並且可以在任何可存取類別中定義。 委派簽章如下:

```
public delegate void MessageListener(IMessage msg);
```

因此您可以將方法定義為:

```
void SomeMethodName(IMessage msg);
```

b) 使用與下列範例類似的項目，將此方法實例化為委派:

```
MessageListener OnMsgMethod = new MessageListener(SomeMethodName)
```

c) 將委派設為消費者的 MessageListener 內容，以向一或多個消費者登錄:

```
consumer.MessageListener = OnMsgMethod;
```

您可以將 MessageListener 設回空值，以移除委派:

```
consumer.MessageListener = null;
```

- 若要設定異常狀況接聽器，請完成下列步驟。

  異常狀況接聽器的運作方式與訊息接聽器相同，但具有不同的委派定義，且會指派給連線，而不是訊息消費者。 這與 C++ 的相同。

  a) 定義方法。

  委派簽章如下:

```
public delegate void ExceptionListener(Exception ex);
```

因此所定義的方法可能如下:

```
void SomeMethodName(Exception ex);
```

b) 使用與下列範例類似的項目，將此方法實例化為委派:

```
ExceptionListener OnExMethod = new ExceptionListener(SomeMethodName)
```

c) 透過設定其 ExceptionListener 內容，向連線登錄委派:

```
connection.ExceptionListener = OnExMethod ;
```

您可以將 ExceptionListener 重設為:

```
null:  connection.ExceptionListener = null;
```

使用 XMS .NET 受管理物件

本節中的主題提供受管理物件的相關資訊。 XMS 應用程式可以從中央受管理物件儲存庫擷取物件定義，並使用它們來建立 Connection Factory 及目的地。

關於這項作業

本節提供資訊來協助建立及管理受管理物件，並說明 XMS 支援的受管理物件儲存庫類型。 本節也說明 XMS 應用程式如何建立與受管理物件儲存庫的連線，以擷取必要的受管理物件。

本節包含下列主題:

- 第 550 頁的『XMS .NET 支援的受管理物件儲存庫類型』
- 第 550 頁的『受管理物件的 XMS .NET 內容對映』
- 第 552 頁的『XMS .NET 受管理 ConnectionFactory 物件的必要內容』

- 第 553 頁的『受管理目的地物件的 XMS .NET 必要內容』
- 第 553 頁的『XMS .NET 建立受管理物件』
- 第 554 頁的『XMS .NET 建立 InitialContext 物件』
- 第 554 頁的『XMS .NET InitialContext 內容』
- 第 555 頁的『XMS 起始環境定義的 URI 格式』
- 第 556 頁的『XMS .NET 的 JNDI 查閱 Web 服務』
- 第 556 頁的『XMS .NET 擷取受管理物件』

XMS .NET 支援的受管理物件儲存庫類型

「檔案系統」及 LDAP 受管理物件可以用來連接至 IBM MQ 及 WebSphere Application Server，而 COS 命名只能用來連接至 WebSphere Application Server。

「檔案系統」物件目錄採用序列化 Java Naming Directory Interface (JNDI) 物件的形式。 LDAP 物件目錄是包含 JNDI 物件的目錄。 「檔案系統」和 LDAP 物件目錄可以由 IBM MQ Explorer(隨 IBM MQ 以及更新版本提供) 來管理。 透過集中化 IBM MQ Connection Factory 和目的地，可以使用檔案系統和 LDAP 物件目錄來管理用戶端連線。 網路管理者可以部署多個應用程式，這些應用程式會參照相同的中央儲存庫，且會自動更新以反映中央儲存庫中所做的連線設定變更。

COS 命名目錄包含 WebSphere Application Server service integration bus Connection Factory 及目的地，並且可以使用 WebSphere Application Server 管理主控台來管理。 若要讓 XMS 應用程式從 COS 命名目錄擷取物件，必須部署 JNDI 查閱 Web 服務。 此 Web 服務並非在所有 WebSphere Application Server service integration technologies 上都可用。 如需詳細資料，請參閱產品說明文件。

註: 重新啟動應用程式連線，使物件目錄的變更生效。

受管理物件的 XMS .NET 內容對映

為了讓 XMS .NET 應用程式能夠使用 IBM MQ JMS 和 WebSphere Application Server Connection Factory 及目的地物件定義，從這些定義擷取的內容必須對映至可以在 XMS Connection Factory 和目的地上設定的對應 XMS 內容。

比方說，如果要建立 XMS Connection Factory，且其中含有從 IBM MQ JMS Connection Factory 擷取的內容，則必須在這兩者之間對映這些內容。

會自動執行所有內容對映。

第 550 頁的表 87 示範 Connection Factory 和目的地最常見的部分內容之間的對映。 此表格中顯示的內容只是一小組範例，並非所有顯示的內容都與所有連線類型及伺服器相關。

表 *87: Connection Factory* 和目的地內容的名稱對映範例

IBM MQ JMS 內容名稱	**XMS 內容名稱**	**WebSphere Application Server service integration bus 內容名稱**
持續性 (每個)	XMSC_DELIVERY_MODE	
到期 (EXP)	XMSC_TIME_TO_LIVE	
優先順序 (PRI)	XMSC_PRIORITY	
	XMSC_WPM_HOST_NAME	serverName
	XMSC_WPM_BUS_NAME	busName
	XMSC_WPM_TOPIC_SPACE	topicName

註: 第 551 頁的表 88 中顯示的內容適用於 JMS 以及 XMS .NET。

表 88: XMS .NET 內容					
內容	**物件類型**				
	CF	**QCF**	**TCF**	**佇列**	**主題**
APPLICATIONNAME	Y	Y	Y	N/A	N/A
非同步異常狀況	Y	Y	Y	N/A	N/A
CCDTURL	Y	Y	Y	N/A	N/A
渠道	Y	Y	Y	N/A	N/A
ConnectionNameList	Y	Y	Y	N/A	N/A
CLIENTRECONNECTOPTIONS	Y	Y	Y	N/A	N/A
CLIENTRECONNECTTIMEOUT	Y	Y	Y	N/A	N/A
clientId	N/A	Y	N/A	N/A	N/A
COMPHDR 第 552 頁的『1』	Y	N/A	Y	N/A	N/A
COMPMSG 第 552 頁的『1』	Y	Y	Y	N/A	N/A
CONNOPT 第 552 頁的『1』	Y	Y	Y	N/A	N/A
CONNTAG 第 552 頁的『1』	Y	Y	Y	N/A	N/A
說明第 552 頁的『1』	N/A	Y	N/A	Y	Y
期限 第 552 頁的『1』	N/A	N/A	N/A	Y	Y
FAILIFQUIESCE	Y	Y	Y	Y	Y
HOSTNAME	N/A	Y	N/A	N/A	N/A
LOCALADDRESS	N/A	Y	N/A	N/A	N/A
持續保存	N/A	N/A	N/A	Y	Y
埠號	N/A	Y	N/A	N/A	N/A
優先順序第 552 頁的『1』	N/A	N/A	N/A	Y	Y
PROVIDERVERSION 第 552 頁的『1』	N/A	Y	N/A	N/A	N/A
QMANAGER	Y	Y	Y	Y	N/A
佇列第 552 頁的『1』	N/A	N/A	N/A	Y	N/A

表 88: XMS .NET 內容 (繼續)					
內容	**物件類型**				
	CF	**QCF**	**TCF**	**佇列**	**主題**
SHARECONVALLOWED	Y	Y	Y	N/A	N/A
TOPIC 第 552 頁的『1』	N/A	N/A	N/A	N/A	Y
Transport 第 552 頁的『1』	N/A	Y	N/A	N/A	N/A

註:

1. 這些內容沒有應用程式層次內容，但可以選擇性地使用受管理內容來設定。

OutboundSNI 內容

V 9.3.0

從 IBM MQ 9.3.0 開始，您可以設定 XMSC_WMQ_OUTBOUND_SNI 內容，這會在應用程式中設定 **OutboundSNI** 內容。

XMSC_WMQ_OUTBOUND_SNI_PROPERTY 採用下列值:

- XMSC_WMQ_OUTBOUND_SNI_CHANNEL ， 對映至 "CHANNEL"
- XMSC_WMQ_OUTBOUND_SNI_HOSTNAME ， 其對映至 "HOSTNAME"
- XMSC_WMQ_OUTBOUND_SNI_ASTERISK ， 其對映至 "*"

此外，您可以使用 MQOUTBOUND_SNI 環境變數來設定 **OutboundSNI** 內容，其採用下列值:

- CHANNEL
- HOSTNAME
- *

註: 如果未設定特定值，則此內容預設為 XMSC_WMQ_OUTBOUND_SNI_CHANNEL。

在受管理節點中設定 **OutboundSNI** 內容的優先順序如下:

1. 應用程式層次內容
2. 環境變數

對於未受管理節點中的 **OutboundSNI** 內容，僅支援 `mqclient.ini` 。

XMS .NET 受管理 ConnectionFactory 物件的必要內容

當應用程式建立 Connection Factory 時，必須定義一些內容來建立傳訊伺服器的連線。

下表中列出的內容是應用程式設定以建立傳訊伺服器連線所需的最低內容。 如果您想要自訂建立連線的方式，您的應用程式可以視需要設定 ConnectionFactory 物件的任何其他內容。 如需相關資訊，請參閱 ConnectionFactory 的內容。 包括可用內容的完整清單。

IBM MQ 佇列管理程式的連線

表 89: IBM MQ 佇列管理程式連線之受管理 ConnectionFactory 物件的內容設定	
必要 XMS 內容	**需要同等的 IBM MQ JMS 內容**
XMSC_CONNECTION_TYPE	XMS 會從 Connection Factory 類別名稱和 TRANSPORT (TRAN) 內容來解決這個問題。

表 *89: IBM MQ* 佇列管理程式連線之受管理 *ConnectionFactory* 物件的內容設定 *(繼續)*	
必要 XMS 內容	**需要同等的 IBM MQ JMS 內容**
XMSC_WMQ_HOST_NAME	HOSTNAME (HOST)
XMSC_WMQ_PORT	PORT
XMSC_WMQ_QUEUE_MANAGER	佇列管理程式的名稱

與分配管理系統的即時連線

表 *90:* 即時連線至分配管理系統之受管理 *ConnectionFactory* 物件的內容設定	
必要 XMS	**需要同等的 IBM MQ JMS 內容**
XMSC_CONNECTION_TYPE	XMS 會從 Connection Factory 類別名稱和 TRANSPORT (TRAN) 內容來解決這個問題。
XMSC_RTT_HOST_NAME	HOSTNAME (HOST)
XMSC_RTT_PORT	PORT

與 WebSphere Application Server service integration bus 的連線

表 *91: WebSphere Application Server service integration bus* 連線的受管理 *ConnectionFactory* 物件的內容設定	
XMS 內容	**說明**
XMSC_CONNECTION_TYPE	應用程式所連接傳訊伺服器的類型。。 這是由 Connection Factory 類別名稱來決定。
XMSC_WPM_BUS_NAME	對於 Connection Factory，則應用程式所連接的服務整合匯流排的名稱，或者對於目的地，則為目的地所在的服務整合匯流排的名稱。

受管理目的地物件的 XMS .NET 必要內容

建立目的地的應用程式必須在受管理的目的地物件上設定應用程式的數個內容。

表 *92:* 受管理目的地物件的內容設定		
連線類型	**內容**	**說明**
IBM MQ 佇列管理程式 (queue manager)	佇列 (QU) 主題 (頂端)	您要連接的佇列 應用程式用作目的地的主題
與分配管理系統的即時連線	主題 (頂端)	應用程式用作目的地的主題
WebSphere Application Server service integration bus	topicName queueName	如果您的應用程式正在連接至主題 如果您的應用程式正在連接至佇列

XMS .NET 建立受管理物件

必須使用適當的管理工具來建立 XMS 應用程式建立傳訊伺服器連線所需的 ConnectionFactory 和「目的地」物件定義。

開始之前

如需 XMS 支援的不同受管理物件儲存庫類型的進一步詳細資料，請參閱 第 550 頁的『XMS .NET 支援的受管理物件儲存庫類型』。

關於這項作業

如果要建立 IBM MQ 的受管理物件，請使用 IBM MQ Explorer 或 IBM MQ JMS 管理 (JMSAdmin) 工具。

若要為 IBM MQ 或 IBM Integration Bus 建立受管理物件，請使用 IBM MQ JMS 管理 (JMSAdmin) 工具。

如果要建立「WebSphere Application Server service integration bus」的受管理物件，請使用 WebSphere Application Server 管理主控台。

在管理工具中，此內容簡稱為 **APPLICATIONNAME** 或 **APPNAME** 。

註: 您無法使用 JMSAdmin 來設定 TRANSPORT (UNMANAGED)。 因此，若要使用管理上選擇的應用程式名稱來取得未受管理的 XMS 用戶端，您需要輸入下列指令:

```
cf.SetIntProperty(XMSC.WMQ_CONNECTION_MODE, XMSC.WMQ_CM_CLIENT_UNMANAGED);
```

下列步驟彙總您在建立受管理物件時所執行的動作。

程序

1. 建立 Connection Factory 並定義必要的內容，以建立從應用程式到所選伺服器的連線。
   XMS 建立連線所需的最少內容定義在 第 552 頁的『XMS .NET 受管理 ConnectionFactory 物件的必要內容』中。
2. 在您應用程式所連接的傳訊伺服器上建立必要的目的地:
   - 若要建立與 IBM MQ 佇列管理程式的連線，請建立佇列或主題。
   - 如需與分配管理系統的即時連線，請建立主題。
   - 若要建立與 WebSphere Application Server service integration bus 的連線，請建立佇列或主題。

   XMS 建立連線所需的最少內容定義在 第 553 頁的『受管理目的地物件的 XMS .NET 必要內容』中。

XMS .NET 建立 InitialContext 物件

應用程式必須建立起始環境定義，以用來建立受管理物件儲存庫的連線，以擷取必要的受管理物件。

關於這項作業

InitialContext 物件會封裝與儲存庫的連線。 XMS API 提供方法來執行下列作業:

- 建立 InitialContext 物件
- 在受管理物件儲存庫中查閱受管理物件。

程序

- 如需建立 InitialContext 物件的進一步詳細資料，請參閱 InitialContext for .NET 和 InitialContext 的內容。

XMS .NET InitialContext 內容

InitialContext 建構子的參數包括受管理物件的儲存庫位置，以統一資源指示器 (URI) 形式提供。 為了讓應用程式建立與儲存庫的連線，可能需要提供比 URI 中包含的資訊更多的資訊。

在 XMS 的 JNDI 及 .NET 實作中，會在建構子的環境 Hashtable 中提供其他資訊。

受管理物件儲存庫的位置定義在 XMSC_IC_URL 內容中。 此內容通常在「建立」呼叫上傳遞，但可以修改為在查閱之前連接至不同的命名目錄。 對於 FileSystem 或 LDAP 環境定義，此內容定義目錄的位址。 對於 COS 命名，這是使用這些內容來連接至 JNDI 目錄的 Web 服務位址。

下列內容會以未經修改的方式傳遞至 Web 服務，而 Web 服務會使用這些內容來連接至 JNDI 目錄。

- XMSC_IC_PROVIDER_URL
- XMSC_IC_SECURITY_CREDENTIALS
- XMSC_IC_SECURITY_AUTHENTICATION
- XMSC_IC_SECURITY_PRINCIPAL
- XMSC_IC_SECURITY_PROTOCOL

XMS 起始環境定義的 URI 格式

受管理物件的儲存庫位置以統一資源指示器 (URI) 提供。 URI 的格式視環境定義類型而定。

FileSystem 環境定義

對於 FileSystem 環境定義， URL 提供檔案系統型目錄的位置。 URL 的結構由 RFC 1738 統一資源定址器 *(URL)*所定義: URL 具有字首 `file://`，且此字首後面的語法是可以在執行 XMS 的系統上開啟之檔案的有效定義。

此語法可以是平台專用，並且可以使用 '/分隔字元或' \ ' 分隔字元。 如果您使用 '\' ，則需要使用其他 '\' 來跳出每一個分隔字元。 這可防止 .NET 架構嘗試將分隔字元解譯為後續字元的跳出字元。

這些範例說明此語法:

```
file://myBindings
file:///admin/.bindings
file://\\admin\\.bindings
file://c:/admin/.bindings
file://c:\\admin\\.bindings
file://\\\\madison\\shared\\admin\\.bindings
file:///usr/admin/.bindings
```

LDAP 環境定義

對於 LDAP 環境定義， URL 的基本結構由 RFC 2255 定義: *The LDAP URL Format*，字首不區分大小寫 `ldap://`

下列範例說明精確的語法:

```
LDAP://[Hostname][:Port]["/"[DistinguishedName]]
```

此語法如 RFC 中所定義，但不支援任何屬性、範圍、過濾器或延伸。

此語法的範例包括:

```
ldap://madison:389/cn=JMSData,dc=IBM,dc=UK
ldap://madison/cn=JMSData,dc=IBM,dc=UK
LDAP:///cn=JMSData,dc=IBM,dc=UK
```

WSS 環境定義

對於 WSS 環境定義， URL 採用 Web 服務端點的形式，字首為 `http://`。

或者，您也可以使用字首 `cosnaming://` 或 `wsvc://`,

這兩個字首會解譯為您使用 WSS 環境定義與透過 http 存取的 URL 搭配，這可讓起始環境定義類型輕鬆直接從 URL 衍生。

此語法的範例包括下列:

```
http://madison.ibm.com:9080/xmsjndi/services/JndiLookup
cosnaming://madison/jndilookup
```

XMS .NET 的 JNDI 查閱 Web 服務

若要從 XMS 存取 COS 命名目錄，必須在 WebSphere Application Server service integration bus 伺服器上部署 JNDI Lookup Web 服務。 此 Web 服務會將 Java 資訊從 COS 命名服務轉換成 XMS 應用程式可以讀取的表單。

Web 服務在安裝目錄內的企業保存檔 SIBXJndiLookupEAR.ear 中提供。 對於現行版本的 IBM MQ Message Service Client (XMS) for .NET，可在 *install_dir*\java\lib 目錄中找到 SIBXJndiLookupEAR.ear 。 您可以使用管理主控台或 wsaadmin Scripting 工具，將它安裝在 WebSphere Application Server service integration bus 伺服器內。 如需部署 Web 服務應用程式的進一步資訊，請參閱產品說明文件。

如果要在 XMS 應用程式內定義 Web 服務，您只需要將 InitialContext 物件的 XMSC_IC_URL 內容設為 Web 服務端點 URL。 例如，如果 Web 服務部署在稱為 MyServer 的伺服器主機上， Web 服務端點 URL 的範例:

```
wsvc://MyHost:9080/SIBXJndiLookup/services/JndiLookup
```

設定 XMSC_IC_URL 內容可讓 InitialContext 查閱呼叫在定義的端點上呼叫 Web 服務，進而從 COS 命名服務查閱所需的受管理物件。

.NET 應用程式可以使用 Web 服務。 XMS C、/C++ 及 XMS .NET 的伺服器端部署相同。 XMS .NET 會直接透過 Microsoft .NET Framework 來呼叫 Web 服務。

XMS .NET 擷取受管理物件

XMS 會使用建立 InitialContext 物件時所提供的位址，或在 InitialContext 內容中，從儲存庫擷取受管理物件。

要擷取的物件可以具有下列類型的名稱:

- 說明 Destination 物件的簡稱，例如，稱為 SalesOrders 的佇列目的地
- 複合名稱，可以由 SubContexts 組成，以 ' /' 區隔，且必須以物件名稱結尾。 複合名稱的範例為 "Warehouse/PickLists/DispatchQueue2" ，其中 Warehouse 和 Picklists 是命名目錄中的 SubContexts ，而 DispatchQueue2 是目的地物件的名稱。

防止應用程式使用較新的 XMS 版本

依預設，當安裝較新的 XMS 版本時，使用舊版的應用程式會自動切換至較新的版本，而無需 recompile.However。不過，您可以在應用程式配置檔中設定屬性來防止應用程式使用較新的版本。

關於這項作業

多重版本共存性特性可確保較新 XMS 版本的安裝不會改寫先前的 XMS 版本。 相反地，相似 XMS .NET 組件的多個實例同時存在於「廣域組件快取 (GAC)」中，但具有不同的版本號碼。 在內部， GAC 會使用原則檔，將應用程式呼叫遞送至最新版本的 XMS。 應用程式無需重新編譯即可執行，並且可以使用較新 XMS .NET 版本中提供的新特性。

程序

- 如果需要應用程式使用舊版 XMS .NET ，請在應用程式配置檔中，將 `publisherpolicy` 屬性設為 `no` 。

  註: 應用程式配置檔是一個檔案，其名稱是由檔案相關的可執行程式名稱所組成，字尾為 .config。 例如， text.exe 的應用程式配置檔將具有名稱 text.exe.config。

  不過，在任何時候，系統的所有應用程式都會使用相同版本的 XMS .NET。

保護 XMS 應用程式的通訊安全

本節提供設定安全通訊的相關資訊，可讓 XMS 應用程式透過 Secure Sockets Layer (SSL) 連接至 WebSphere Application Server service integration bus 傳訊引擎或 IBM MQ 佇列管理程式。

關於這項作業

本節包含下列主題:

IBM MQ 佇列管理程式的安全連線

如果要讓 XMS .NET 應用程式能夠建立與 IBM MQ 佇列管理程式的安全連線，必須在 ConnectionFactory 物件中定義相關內容。

加密協議中使用的通訊協定可以是 Secure Sockets Layer (SSL) 或傳輸層安全 (TLS)，視您在 ConnectionFactory 物件中指定的 CipherSuite 而定。

下表顯示使用 SSL 連接至 IBM MQ 佇列管理程式之連線的 ConnectionFactory 內容，以及簡要說明:

表 *93: ConnectionFactory* 的內容，用於透過 *SSL* 連線至 *IBM MQ* 佇列管理程式

內容名稱	說明
XMSC_WMQ_SSL_CERT_STORES	保留給佇列管理程式建立 SSL 連線時所使用憑證撤銷清冊 (CRL) 的伺服器位置。
XMSC_WMQ_SSL_CIPHER_SPEC	建立佇列管理程式的安全連線時所使用的 CipherSpec 名稱。
XMSC_WMQ_SSL_CIPHER_SUITE	建立佇列管理程式的 TLS 連線時所使用的 CipherSuite 名稱。 在協議安全連線時使用的通訊協定取決於指定的 CipherSuite。
XMSC_WMQ_SSL_CRYPTO_HW	用戶端系統所連接的加密硬體的配置詳細資料。
XMSC_WMQ_SSL_FIPS_REQUIRED	此內容的值決定應用程式是否可以使用不符合 FIPS 標準的密碼組合。 如果此內容設為 true，則只將 FIPS 演算法用於用戶端/伺服器連線。
XMSC_WMQ_SSL_KEY_REPOSITORY	在其中儲存金鑰和憑證的金鑰資料庫檔的位置。
XMSC_WMQ_SSL_KEY_RESETCOUNT	KeyResetCount 代表在重新協議秘密金鑰之前 SSL 交談中傳送和收到的未加密位元組總數。
XMSC_WMQ_SSL_PEER_NAME	建立佇列管理程式的 SSL 連線時所使用的對等節點名稱。

IBM MQ 佇列管理程式之 *XMS* 連線的 *CipherSuite* 及 *CipherSpec* 名稱對映

InitialContext 會在 JMSAdmin Connection Factory 內容 SSLCIPHERSUITE 與 XMS 接近對等的 XMSC_WMQ_SSL_CIPHER_SPEC 之間進行轉換。 如果您指定 XMSC_WMQ_SSL_CIPHER_SUITE 的值，但省略 XMSC_WMQ_SSL_CIPHER_SPEC 的值，則需要類似的轉換。

第 558 頁的表 94 列出可用的 CipherSpecs 及其 JSSE CipherSuite 對等項目。

表 94: 可用的 *CipherSpecs* 及其 *JSSE CipherSuite* 對等項目	
CipherSpec	**等效 JSSE CipherSuite**
TLS_RSA_WITH_3DES_EDE_CBC_SHA	SSL_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA

註: Deprecated TLS_RSA_WITH_3DES_EDE_CBC_SHA 已淘汰。 不過，在連線因錯誤 AMQ9288 而終止之前，它仍可用來傳送最多 32 GB 的資料。 若要避免此錯誤，您需要避免使用三重 DES 演算法，或在使用此 CipherSpec 時啟用秘密金鑰重設。

WebSphere Application Server service integration bus 傳訊引擎的安全連線

如果要讓 XMS .NET 應用程式能夠建立與 WebSphere Application Server service integration bus 傳訊引擎的安全連線，必須在 ConnectionFactory 物件中定義相關內容。

XMS 為 WebSphere Application Server service integration bus 的連線提供 SSL 及 HTTPS 支援。 SSL 和 HTTPS 提供安全連線，以進行鑑別和機密性。

類似於 WebSphere 安全， XMS 安全是根據 JSSE 安全標準及命名慣例來配置，其中包括使用 CipherSuites 來指定協議安全連線時所使用的演算法。 加密協議中使用的通訊協定可以是 SSL 或 TLS ， 視您在 ConnectionFactory 物件中指定的 CipherSuite 而定。

第 558 頁的表 95 列出必須在 ConnectionFactory 物件中定義的內容。

表 95: *ConnectionFactory* 的內容，用於 *WebSphere Application Server service integration bus* 傳訊引擎的安全連線	
內容名稱	**說明**
XMSC_WPM_SSL_CIPHER_SUITE	要在 WebSphere Application Server service integration bus 傳訊引擎的 TLS 連線上使用的 CipherSuite 名稱。 在協議安全連線時使用的通訊協定取決於指定的 CipherSuite。
XMSC_WPM_SSL_KEYRING_LABEL	向伺服器鑑別時使用的憑證。

以下是 WebSphere Application Server service integration bus 傳訊引擎安全連線的 ConnectionFactory 內容範例:

```
cf.setStringProperty(XMSC_WPM_PROVIDER_ENDPOINTS, host_name:port_number:chain_name);
cf.setStringProperty(XMSC_WPM_SSL_KEY_REPOSITORY, key_repository_pathname);
cf.setStringProperty(XMSC_WPM_TARGET_TRANSPORT_CHAIN, transport_chain);
cf.setStringProperty(XMSC_WPM_SSL_CIPHER_SUITE, cipher_suite);
cf.setStringProperty(XMSC_WPM_SSL_KEYRING_STASH_FILE, stash_file_pathname);
```

其中 chain_name 應該設為 BootstrapTunneledSecureMessaging 或 BootstrapSecureMessaging ， 而 port_number 是引導伺服器用來接聽送入要求的埠號。

以下是插入範例值之 WebSphere Application Server service integration bus 傳訊引擎安全連線的 ConnectionFactory 內容範例:

```
  /* CF properties needed for an SSL connection */
    cf.setStringProperty(XMSC_WPM_PROVIDER_ENDPOINTS,"localhost:7286:BootstrapSecureMessaging");
    cf.setStringProperty(XMSC_WPM_TARGET_TRANSPORT_CHAIN,"InboundSecureMessaging");
    cf.setStringProperty(XMSC_WPM_SSL_KEY_REPOSITORY,"C:\\Program Files\\IBM\\gsk7\\bin\
\XMSkey.kdb");
    cf.setStringProperty(XMSC_WPM_SSL_KEYRING_STASH_FILE,"C:\\Program Files\\IBM\\gsk7\\bin\
\XMSkey.sth");
    cf.setStringProperty(XMSC_WPM_SSL_CIPHER_SUITE,"SSL_RSA_EXPORT_WITH_RC4_40_MD5");
```

WebSphere Application Server service integration bus 連線的 *CipherSuite* 及 *CipherSpec* 名稱對映

因為 IBM Global Security Kit (GSKit) 使用 CipherSpecs 而非 CipherSuites，所以 XMSC_WPM_SSL_CIPHER_SUITE 內容中指定的 JSSE 樣式 CipherSuite 名稱必須對映至 GSKit-style CipherSpec 名稱。

第 559 頁的表 96 列出每一個已辨識 CipherSuite 的對等 CipherSpec 。

表 *96:* 可用的 *CipherSuites* 及其對等 *CipherSpecs*

CipherSuite	**CipherSpec 對等項目**
TLS_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA

註: Deprecated TLS_RSA_WITH_3DES_EDE_CBC_SHA 已淘汰。 不過，在連線因錯誤 AMQ9288 而終止之前，它仍可用來傳送最多 32 GB 的資料。 若要避免此錯誤，您需要避免使用三重 DES 演算法，或在使用此 CipherSpec 時啟用秘密金鑰重設。

XMS 訊息

本節說明 XMS 訊息的結構和內容，並說明應用程式如何處理 XMS 訊息。

本節包含下列主題:

- 第 559 頁的『XMS 訊息的組件』
- 第 559 頁的『XMS 訊息中的標頭欄位』
- 第 560 頁的『XMS 訊息的內容』
- 第 562 頁的『XMS 訊息的內文』
- 第 565 頁的『訊息選取器』
- 第 566 頁的『將 XMS 訊息對映至 IBM MQ 訊息』

XMS 訊息的組件

XMS 訊息包含一個標頭、一組內容及一個內文。

標頭
訊息的標頭包含欄位，而所有訊息都包含相同的標頭欄位集。 XMS 及應用程式使用標頭欄位的值來識別及遞送訊息。 如需標頭欄位的相關資訊，請參閱 第 559 頁的『XMS 訊息中的標頭欄位』。

內容集
訊息的內容指定訊息的其他相關資訊。 雖然所有訊息都有一組相同的標頭欄位，但每一則訊息可以有一組不同的內容。 如需相關資訊，請參閱第 560 頁的『XMS 訊息的內容』。

內文
訊息內文包含應用程式資料。 如需相關資訊，請參閱第 562 頁的『XMS 訊息的內文』。

應用程式可以選取要接收的訊息。 使用訊息選取元來指定選取準則。 準則可以根據特定標頭欄位的值，以及訊息任何內容的值。 如需訊息選取器的相關資訊，請參閱 第 565 頁的『訊息選取器』。

XMS 訊息中的標頭欄位

為了容許 XMS 應用程式與 WebSphere JMS 應用程式交換訊息， XMS 訊息的標頭包含 JMS 訊息標頭欄位。

這些標頭欄位的名稱以字首 JMS 開始。 如需 JMS 訊息標頭欄位的說明，請參閱 *Java Message Service* 規格。

XMS 會將 JMS 訊息標頭欄位實作為訊息物件的屬性。 每一個標頭欄位都有自己的方法來設定及取得其值。 如需這些方法的說明，請參閱 IMessage。 標頭欄位一律可讀取及寫入。

第 560 頁的表 97 列出 JMS 訊息標頭欄位，並指出如何為傳輸的訊息設定每一個欄位的值。 當應用程式傳送訊息時，或在 JMSRedelivered 的情況下，當應用程式接收訊息時， XMS 會自動設定部分欄位。

表 *97: JMS* 訊息標頭欄位.]

JMS 訊息標頭欄位的名稱	**如何設定所傳輸訊息的值 (格式為 *method* [*class*])**
JMSCorrelationID	設定 JMSCorrelationID [訊息]
JMSDeliveryMode	傳送 [MessageProducer]
JMSDestination	傳送 [MessageProducer]
JMSExpiration	傳送 [MessageProducer]
JMSMessageID	傳送 [MessageProducer]
JMSPriority	傳送 [MessageProducer]
JMSRedelivered	接收 [MessageConsumer]
JMSReplyTo	設定 JMSReplyTo [訊息]
JMSTimestamp	傳送 [MessageProducer]
JMSType	設定 JMSType [訊息]

XMS 訊息的內容

XMS 支援三種類型的訊息內容: JMS 定義的內容、 IBM 定義的內容，以及應用程式定義的內容。

XMS 應用程式可以與 WebSphere JMS 應用程式交換訊息，因為 XMS 支援「訊息」物件的下列預先定義內容:

- WebSphere JMS 支援的相同 JMS 定義內容。 這些內容的名稱以字首 JMSX 開頭。
- WebSphere JMS 支援的相同 IBM 定義內容。 這些內容的名稱以字首 JMS_IBM_ 開頭。

每一個預先定義的內容都有兩個名稱:

- JMS 名稱 (若為 JMS 定義的內容) 或 WebSphere JMS 名稱 (若為 IBM 定義的內容)。

  這是在 JMS 或 WebSphere JMS 中用來識別內容的名稱，也是與具有此內容的訊息一起傳輸的名稱。 XMS 應用程式使用此名稱來識別訊息選取元表示式中的內容。
- XMS 名稱，用於在所有狀況中識別內容，但訊息選取器表示式除外。 每一個 XMS 名稱都定義為 `IBM.XMS.XMSC` 類別中的已命名常數。 已命名常數的值是對應的 JMS 或 WebSphere JMS 名稱。

除了預先定義的內容之外， XMS 應用程式還可以建立並使用自己的訊息內容集。 這些內容稱為 應用程式定義的內容。

在應用程式建立訊息之後，訊息的內容是可讀取和可寫入的。 在應用程式傳送訊息之後，內容仍可讀取及寫入。 當應用程式接收訊息時，訊息的內容是唯讀的。 當訊息的內容是唯讀時，如果應用程式呼叫 Message 類別的 `Clear Properties` 方法，則內容會變成可讀取及可寫入。 此方法也會清除內容。

在清除訊息內容之後轉遞收到的訊息時，其行為將與轉遞標準 WMQ XMS for .NET BytesMessage 的行為一致，並清除訊息內容。

不過，不建議這樣做，因為下列內容將會遺失:

- JMS_IBM_Encoding 內容值，暗示無法對訊息資料進行有意義的解碼。
- JMS_IBM_Format 內容值，暗示 (MQMD 或新的 MQRFH2) 訊息標頭與現有標頭之間的標頭鏈會中斷。

若要判定訊息所有內容的值，應用程式可以呼叫「訊息」類別的「取得內容」方法。 此方法會建立一個反覆運算子來封裝「內容」物件的清單，其中每一個「內容」物件代表訊息的內容。 然後，應用程式可以使用

Iterator 類別的方法依序擷取每一個 Property 物件，並且可以使用 Property 類別的方法來擷取每一個內容的名稱、資料類型及值。

訊息的 *JMS* 定義內容

XMS 和 WebSphere JMS 都支援訊息的數個 JMS 定義內容。

第 561 頁的表 98 列出 XMS 和 WebSphere JMS 支援之訊息的 JMS 定義內容。 如需 JMS 定義內容的說明，請參閱 *Java Message Service* 規格。 JMS 定義的內容對分配管理系統的即時連線無效。

此表格指定每一個內容的資料類型，並指出如何設定所傳輸訊息的內容值。 當應用程式傳送訊息時，或在 JMSXDeliveryCount 的情況下，當應用程式接收訊息時， XMS 會自動設定部分內容。

表 *98:* 訊息的 *JMS* 定義內容

JMS 定義內容的 XMS 名稱	**JMS 名稱**	**資料類型**	**如何設定所傳輸訊息的值 (格式為 *method* [*class*])**
JMSX_APPID	JMSXAppID	System.String	傳送 [MessageProducer]
JMSX_DELIVERY_COUNT	JMSXDeliveryCount	System.Int32	接收 [MessageConsumer]
JMSX_GROUPID	JMSXGroupID	System.String	設定字串內容 [PropertyContext]
JMSX_GROUPSEQ	JMSXGroupSeq	System.Int32	設定整數內容 [PropertyContext]
JMSX_USERID	JMSXUserID	System.String	傳送 [MessageProducer]

訊息的 *IBM* 定義內容

XMS 和 WebSphere JMS 支援訊息的數個 IBM 定義內容。

第 561 頁的表 99 列出 XMS 和 WebSphere JMS 支援之訊息的 IBM 定義內容。 如需 IBM 定義內容的相關資訊，請參閱 IBM MQ 或 WebSphere Application Server 產品說明文件。

此表格指定每一個內容的資料類型，並指出如何設定所傳輸訊息的內容值。 當應用程式傳送訊息時， XMS 會自動設定部分內容。

表 *99: IBM* 定義的訊息內容

IBM 定義內容的 XMS 名稱	**WebSphere JMS 名稱**	**資料類型**	**如何設定所傳輸訊息的值 (格式為 *method* [*class*])**
JMS_IBM_CHARACTER_SET	JMS_IBM_Character_Set	System.Int32	設定整數內容 [PropertyContext]
JMS_IBM_Encoding	JMS_IBM_Encoding	System.Int32	設定整數內容 [PropertyContext]
JMS_IBM_EXCEPTIONMESSAGE	JMS_IBM_ExceptionMessage	System.String	接收 [MessageConsumer]
JMS_IBM_EXCEPTIONREASON	JMS_IBM_ExceptionReason	System.Int32	接收 [MessageConsumer]
JMS_IBM_EXCEPTIONTIMESTAMP	JMS_IBM_ExceptionTimestamp	System.Int64	接收 [MessageConsumer]
JMS_IBM_EXCEPTIONPROBLEM 目的地	JMS_IBM_ExceptionProblem 目的地	System.String	接收 [MessageConsumer]
JMS_IBM_FEEDBACK	JMS_IBM_Feedback	System.Int32	設定整數內容 [PropertyContext]

表 99: *IBM* 定義的訊息內容 *(*繼續*)*

IBM 定義內容的 XMS 名稱	WebSphere JMS 名稱	資料類型	如何設定所傳輸訊息的值 (格式為 *method* [*class*])
JMS_IBM_FORMAT	JMS_IBM_Format	System.String	設定字串內容 [PropertyContext]
JMS_IBM_LAST_MSG_IN_GROUP	JMS_IBM_Last_Msg_In_Group	System.Boolean	設定整數內容 [PropertyContext]
JMS_IBM_MSGTYPE	JMS_IBM_MsgType	System.Int32	設定整數內容 [PropertyContext]
JMS_IBM_PUTAPPLTYPE	JMS_IBM_PutAppl 類型	System.Int32	傳送 [MessageProducer]
JMS_IBM_PUTDATE	JMS_IBM_PutDate	System.String	傳送 [MessageProducer]
JMS_IBM_PUTTIME	JMS_IBM_PutTime	System.String	傳送 [MessageProducer]
JMS_IBM_REPORT_COA	JMS_IBM_Report_COA	System.Int32	設定整數內容 [PropertyContext]
JMS_IBM_REPORT_COD	JMS_IBM_Report_COD	System.Int32	設定整數內容 [PropertyContext]
JMS_IBM_REPORT_DISCARD_MSG	JMS_IBM_Report_Discard_Msg	System.Int32	設定整數內容 [PropertyContext]
JMS_IBM_REPORT_EXCEPTION	JMS_IBM_Report_Exception	System.Int32	設定整數內容 [PropertyContext]
JMS_IBM_REPORT_EXPIRATION	JMS_IBM_Report_Expiration	System.Int32	設定整數內容 [PropertyContext]
JMS_IBM_REPORT_NAN	JMS_IBM_Report_NAN	System.Int32	設定整數內容 [PropertyContext]
JMS_IBM_REPORT_PAN	JMS_IBM_Report_PAN	System.Int32	設定整數內容 [PropertyContext]
JMS_IBM_REPORT_PASS_CORREL_ ID	JMS_IBM_Report_Pass_Correl_ID	System.Int32	設定整數內容 [PropertyContext]
JMS_IBM_REPORT_PASS_MSG_ID	JMS_IBM_Report_Pass_Msg_ID	System.Int32	設定整數內容 [PropertyContext]
JMS_IBM_SYSTEM_MESSAGEID	JMS_IBM_System_MessageID	System.String	傳送 [MessageProducer]

訊息的應用程式定義內容

XMS 應用程式可以建立及使用自己的訊息內容集。 當應用程式傳送訊息時，這些內容也會隨訊息一起傳輸。 然後，接收端應用程式 (使用訊息選取器) 可以根據這些內容的值來選取它想要接收的訊息。

若要容許 WebSphere JMS 應用程式選取及處理 XMS 應用程式所傳送的訊息，應用程式定義內容的名稱必須符合在訊息選取元表示式中形成 ID 的規則。 如需相關資訊，請參閱第 122 頁的『JMS 中的訊息選取器』。 應用程式定義內容的值必須具有下列其中一種資料類型: System.Boolean、 System.SByte、 System.Int16、 System.Int32、 System.Int64、 System.Float、 System.Double 或 System.String。

XMS 訊息的內文

訊息內文包含應用程式資料。 不過，訊息可以沒有內文，且只包含標頭欄位和內容。

XMS 支援五種類型的訊息內文:

位元組
主體包含位元組串流。 具有此類型內文的訊息稱為 位元組訊息。 IBytesMessage 介面包含用來處理位元組訊息內文的方法。

對映
內文包含一組名稱/值配對，其中每一個值都有相關聯的資料類型。 具有此類型內文的訊息稱為 對映訊息。 IMapMessage 介面包含用來處理對映訊息內文的方法。

物件
主體包含序列化 Java 或 .NET 物件。 具有此類型內文的訊息稱為 物件訊息。 IObjectMessage 介面包含用來處理物件訊息內文的方法。

串流
內文包含值串流，其中每一個值都有相關聯的資料類型。 具有此類型內文的訊息稱為 串流訊息。 IStreamMessage 介面包含用來處理串流訊息內文的方法。

文字
內文包含字串。 具有此類型內文的訊息稱為 文字訊息。 ITextMessage 介面包含處理文字訊息內文的方法。

IMessage 介面是所有訊息物件的母項，可在傳訊功能中用來代表任何 XMS 訊息類型。

如需每一種資料類型的大小及最大值和最小值的相關資訊，請參閱 第 542 頁的表 84。

位元組訊息

位元組訊息的主體包含位元組串流。 主體只包含實際資料，傳送及接收應用程式負責解譯此資料。

如果 XMS 應用程式需要與未使用 XMS 或 JMS 應用程式設計介面的應用程式交換訊息，則位元組訊息很有用。

在應用程式建立位元組訊息之後，訊息內文是唯寫的。 應用程式透過針對 .NET 呼叫 IBytesMessage 介面的適當寫入方法，將應用程式資料組合到主體中。 每次應用程式將值寫入位元組訊息串流時，都會在應用程式所寫入的前一個值之後立即組合該值。 XMS 會維護內部游標，以記住組合的最後一個位元組的位置。

當應用程式傳送訊息時，訊息內文會變成唯讀。 在此模式中，應用程式可以反覆地傳送訊息。

當應用程式接收位元組訊息時，訊息內文是唯讀的。 應用程式可以使用 IBytesMessage 介面的適當 read 方法來讀取位元組訊息串流的內容。 應用程式會依序讀取位元組，且 XMS 會維護內部游標，以記住所讀取最後一個位元組的位置。

當位元組訊息的內文可寫入時，如果應用程式呼叫 IBytesMessage 介面的 Reset 方法，則內文會變成唯讀。此方法也會將游標重新定位在位元組訊息串流的開頭。

當位元組訊息的內文是唯讀時，如果應用程式呼叫 .NET 之 IMessage 介面的 `Clear Body` 方法，則內文會變成可寫入。 此方法也會清除主體。

對映訊息

對映訊息的內文包含一組名稱/值配對，其中每一個值都有相關聯的資料類型。

在每一個名稱/值配對中，名稱是識別值的字串，值是應用程式資料的元素，具有 第 565 頁的表 100 中列出的其中一個 XMS 資料類型。 未定義名稱/值配對的順序。 MapMessage 類別包含設定及取得名稱/值配對的方法。

應用程式可以透過指定名稱來隨機存取名稱/值配對。

.NET 應用程式可以使用 MapNames 內容來取得對映訊息內文中的名稱列舉。

當應用程式取得名稱/值配對的值時， XMS 可以將該值轉換為另一個資料類型。 例如，若要從對映訊息內文中取得整數，應用程式可以呼叫 MapMessage 類別的 GetString 方法，它會以字串形式傳回整數。 支援的轉換與 XMS 將內容值從一種資料類型轉換為另一種資料類型時支援的轉換相同。 如需受支援轉換的相關資訊，請參閱 第 542 頁的『將內容值從一個資料類型隱含轉換為另一個資料類型』。

在應用程式建立對映訊息之後，訊息的內文是可讀取及可寫入的。 在應用程式傳送訊息之後，內文仍可讀取及寫入。 當應用程式接收對映訊息時，訊息內文是唯讀的。 當對映訊息的主體是唯讀時，如果應用程式呼叫 Message 類別的 Clear Body 方法，則主體會變成可讀取及可寫入。 此方法也會清除主體。

物件訊息

物件訊息的主體包含序列化 Java 或 .NET 物件。

XMS 應用程式可以接收物件訊息，變更其標頭欄位和內容，然後將它傳送至另一個目的地。 應用程式也可以複製物件訊息的內文，並使用它來形成另一個物件訊息。 XMS 會將物件訊息的內文視為位元組陣列。

在應用程式建立物件訊息之後，訊息內文是可讀取及可寫入的。 在應用程式傳送訊息之後，內文仍可讀取及寫入。 當應用程式接收物件訊息時，訊息內文是唯讀的。 當物件訊息的內文是唯讀時，如果應用程式呼叫 .NET IMessage 介面的 `Clear Body` 方法，則內文會變成可讀取及可寫入。 此方法也會清除主體。

串流訊息

串流訊息的內文包含值串流，其中每一個值都有相關聯的資料類型。

值的資料類型是 第 565 頁的表 100 中列出的其中一個 XMS 資料類型。

在應用程式建立串流訊息之後，訊息內文是可寫入的。 應用程式會呼叫 .NET 的 IStreamMessage 介面的適當寫入方法，將應用程式資料組合到主體中。 每次應用程式將值寫入訊息串流時，該值及其資料類型會緊接在應用程式所寫入的前一個值之後組合。 XMS 會維護內部游標，以記住組合的最後一個值的位置。

當應用程式傳送訊息時，訊息內文會變成唯讀。 在此模式中，應用程式可以多次傳送訊息。

當應用程式接收串流訊息時，訊息內文是唯讀的。 應用程式可以使用適用於 .NET 的 IStreamMessage 介面的適當讀取方法，來讀取訊息串流的內容。 應用程式依序讀取值，且 XMS 會維護內部游標，以記住所讀取最後一個值的位置。

當應用程式從訊息串流讀取值時， XMS 可以將值轉換為另一個資料類型。 例如，若要從訊息串流讀取整數，應用程式可以呼叫 ReadString 方法，以字串形式傳回整數。 支援的轉換與 XMS 將內容值從一種資料類型轉換為另一種資料類型時支援的轉換相同。 如需受支援轉換的相關資訊，請參閱 第 542 頁的『將內容值從一個資料類型隱含轉換為另一個資料類型』。

如果應用程式嘗試從訊息串流讀取值時發生錯誤，則游標不會進階。 應用程式可以從錯誤中回復，方法是嘗試將值讀取為另一個資料類型。

當串流訊息的內文是唯讀時，如果應用程式針對 XMS 呼叫 IStreamMessage 介面的 Reset 方法，則內文會變成唯讀。 此方法也會將游標重新定位在訊息串流的開頭。

當串流訊息的內文是唯讀時，如果應用程式呼叫 XMS 之 IMessage 介面的 `Clear Body` 方法，則內文會變成唯寫。 此方法也會清除主體。

文字訊息

文字訊息的內文包含字串。

在應用程式建立文字訊息之後，訊息內文是可讀取及可寫入的。 在應用程式傳送訊息之後，內文仍可讀取及寫入。 當應用程式收到文字訊息時，訊息內文是唯讀的。 當文字訊息的內文是唯讀時，如果應用程式呼叫 .NET 的 IMessage 介面的「清除內文」方法，則內文會變成可讀取及可寫入。 此方法也會清除主體。

應用程式資料元素的資料類型

為了確保 XMS 應用程式可以與 IBM MQ classes for JMS 應用程式交換訊息，這兩個應用程式必須能夠以相同方式解譯訊息內文中的應用程式資料。

因此， XMS 應用程式寫入訊息內文中的每一個應用程式資料元素，都必須具有 第 565 頁的表 100 中列出的其中一個資料類型。 對於每一種資料類型，表格會顯示相容的 Java 資料類型。 XMS 提供僅使用這些資料類型寫入應用程式資料元素的方法。

表 *100:* 與 *Java* 資料類型相容的 *XMS* 資料類型		
XMS 資料類型	**代表**	**相容 Java 資料類型**
System.Boolean	布林值 true 或 false	布林值
System.Char16	雙位元組字元	char
System.SByte	帶正負號的 8 位元整數	位元組
System.Int16	帶正負號的 16 位元整數	短
System.Int32	帶正負號的 32 位元整數	整數
System.Int64	帶正負號的 64 位元整數	長整數
System.Float	帶正負號的浮點數字	浮點數
System.Double	帶正負號的倍精準度浮點數字	倍精準數
System.String	字串	字串

如需每一種資料類型的大小、最大值及最小值的相關資訊，請參閱 第 542 頁的『XMS 初始類型』。

訊息選取器

XMS 應用程式使用訊息選取器來選取它要接收的訊息。

當應用程式建立訊息消費者時，它可以建立訊息選取元表示式與消費者的關聯。 訊息選取元表示式指定選取準則。

當應用程式連接至 IBM WebSphere MQ 7.0 佇列管理程式時，會在佇列管理程式端完成訊息選取。 XMS 不會執行任何選擇，只會遞送它從佇列管理程式收到的訊息，從而提供更好的效能。

應用程式可以建立多個訊息消費者，每一個都有自己的訊息選取元表示式。 如果送入訊息符合多個訊息消費者的選取準則，則 XMS 會將訊息遞送至其中每一個消費者。

訊息選取元表示式可以參照訊息的下列內容:

- JMS 定義的內容
- IBM 定義的內容
- 應用程式定義的內容

它也可以參照下列訊息標頭欄位:

- JMSCorrelationID
- JMSDeliveryMode
- JMSMessageID
- JMSPriority
- JMSTimestamp
- JMSType

不過，訊息選取元表示式無法參照訊息內文中的資料。

以下是訊息選取元表示式的範例:

```
JMSPriority > 3 AND manufacturer = 'Jaguar' AND model in ('xj6','xj12')
```

只有在訊息的優先順序大於 3 時， XMS 才會使用此訊息選取元表示式將訊息遞送給訊息消費者; 應用程式定義的內容 (製造商) ，其值為 `Jaguar;` ; 另一個應用程式定義的內容 (模型) ，其值為 `xj6` 或 `xj12.`

在 XMS 中形成訊息選取器表示式的語法規則與在 IBM MQ classes for JMS 中形成訊息選取器表示式的語法規則相同。 如需如何建構訊息選取元表示式的相關資訊，請參閱 IBM MQ 產品說明文件。請注意，在訊息選

取元表示式中， JMS 定義內容的名稱必須是 JMS 名稱，而 IBM 定義內容的名稱必須是 IBM MQ classes for JMS 名稱。 您無法在訊息選取器表示式中使用 XMS 名稱。

將 XMS 訊息對映至 IBM MQ 訊息

XMS 訊息的 JMS 標頭欄位和內容會對映至 IBM MQ 訊息的標頭結構中的欄位。

當 XMS 應用程式連接至 IBM MQ 佇列管理程式時，在類似情況下，傳送至佇列管理程式的訊息會以 IBM MQ classes for JMS 訊息對映至 IBM MQ 訊息的相同方式，對映至 IBM MQ 訊息。

如果「目的地」物件的 XMSC_WMQ_TARGET_CLIENT 內容設為 XMSC_WMQ_TARGET_DEST_JMS ，傳送至目的地之訊息的 JMS 標頭欄位及內容會對映至 IBM MQ 訊息的 MQMD 及 MQRFH2 標頭結構中的欄位。 以此方式設定 XMSC_WMQ_TARGET_CLIENT 內容，會假設接收訊息的應用程式可以處理 MQRFH2 標頭。 因此，接收端應用程式可能是另一個 XMS 應用程式、 IBM MQ classes for JMS 應用程式，或設計用來處理 MQRFH2 標頭的原生 IBM MQ 應用程式。

如果「目的地」物件的 XMSC_WMQ_TARGET_CLIENT 內容設為 XMSC_WMQ_TARGET_DEST_MQ ，則傳送至目的地之訊息的 JMS 標頭欄位及內容會對映至 IBM MQ 訊息的 MQMD 標頭結構中的欄位。 訊息不包含 MQRFH2 標頭，且會忽略無法對映至 MQMD 標頭結構中的欄位的任何 JMS 標頭欄位和內容。 因此，接收訊息的應用程式可以是原生 IBM MQ ，其設計目的不是處理 MQRFH2 標頭。

從佇列管理程式接收到的 IBM MQ 訊息對映至 XMS 訊息的方式，與在類似情況下 IBM MQ 訊息對映至 IBM MQ classes for JMS 訊息的方式相同。

如果送入的 IBM MQ 訊息具有 MQRFH2 標頭，則產生的 XMS 訊息具有一個內文，其類型由 MQRFH2 標頭的 `mcd` 資料夾中包含的 **Msd** 內容值決定。 如果 MQRFH2 標頭中沒有 **Msd** 內容，或如果 IBM MQ 訊息沒有 MQRFH2 標頭，則產生的 XMS 訊息會有一個主體，其類型是由 MQMD 標頭中 *Format* 欄位的值所決定。 如果 *Format* 欄位設為 MQFMT_STRING ，則 XMS 訊息是文字訊息。 否則， XMS 訊息是位元組訊息。 如果 IBM MQ 訊息沒有 MQRFH2 標頭，則只會設定那些 JMS 標頭欄位，以及可從 MQMD 標頭中的欄位衍生的內容。

如需將 IBM MQ classes for JMS 訊息對映至 IBM MQ 訊息的相關資訊，請參閱 第 125 頁的『將 JMS 訊息對映至 IBM MQ 訊息』。

從 *IBM MQ Message Service Client (XMS) for .NET* 應用程式讀取及寫入訊息描述子

您可以存取 IBM MQ 訊息的所有訊息描述子 (MQMD) 欄位，但 StrucId 和版本除外; BackoutCount 可以讀取，但無法寫入。

IBM MQ Message Service Client (XMS) for .NET 提供的訊息屬性可協助 XMS 應用程式設定 MQMD 欄位，以及驅動 IBM WebSphere MQ 應用程式。

使用發佈/訂閱傳訊時，會有一些限制。 例如，將忽略諸如 MsgID 和 CorrelId 之類的 MQMD 欄位 (如果已設定的話)。

當 **PROVIDERVERSION** 內容設為 6 時，也無法使用此功能。

從 *IBM MQ Message Service Client (XMS) for .NET* 應用程式存取 *IBM MQ* 訊息資料

您可以存取完整 IBM MQ 訊息資料，包括 IBM MQ Message Service Client (XMS) for .NET 應用程式內的 MQRFH2 標頭 (如果有的話) 及任何其他 IBM MQ 標頭 (如果有的話) 作為 JMSBytesMessage 內文。

只有在連接至 IBM WebSphere MQ 7.0 或更新版本的佇列管理程式，且 IBM MQ 傳訊提供者處於標準模式時，才能使用本主題中說明的功能。

目的地物件內容決定 XMS 應用程式如何存取整個 IBM MQ 訊息 (包括 MQRFH2 標頭，如果存在的話) 作為 JMSBytesMessage 的內文。

ALW 開發 AMQP 用戶端應用程式

AMQP API 的 IBM MQ 支援可讓 IBM MQ 管理者建立 AMQP 通道。 啟動時，此通道會定義埠號，以接受來自 AMQP 用戶端應用程式的連線。

您可以在 AIX, Linux, and Windows 系統上安裝 AMQP 通道; 它在 IBM i 或 z/OS 上無法使用。

AMQP 1.0 用戶端應用程式可以透過 AMQP 通道連接至佇列管理程式。

使用 Apache Qpid JMS 程式庫開發應用程式

簡介

Apache Qpid JMS 程式庫使用 AMQP 1.0 通訊協定來提供 JMS 2 規格的實作。

Apache Qpid JMS 會以不同於 MQ Light 傳訊 API 的方式使用 AMQP 1.0 通訊協定的部分層面。 IBM MQ 9.2 新增對 IBM MQ AMQP 通道的支援，以便 Apache Qpid JMS 應用程式能夠連接至 IBM MQ 並執行發佈/訂閱傳訊，包括使用共用訂閱。

V 9.3.0 IBM MQ 9.3 新增了對 IBM MQ AMQP 通道的進一步支援，以便 Apache Qpid JMS 應用程式能夠連接至 IBM MQ 並執行點對點傳訊。 如需相關資訊，請參閱第 571 頁的『AMQP 通道上的點對點支援』。

V 9.3.0 IBM MQ 9.3.0 新增 IBM MQ AMQP 通道的進一步佇列瀏覽支援，因此 Apache Qpid JMS 應用程式可以連接至 IBM MQ ，並從佇列中執行訊息瀏覽。 如需相關資訊，請參閱第 571 頁的『AMQP 通道上的點對點支援』。

V 9.3.0 IBM MQ 9.3.0 會為 AMQP 通道新增兩個額外的通道屬性: TMPMODEL 及 TMPQPRFX。 這些屬性適用於模型佇列，以及建立暫時佇列時要使用的暫時佇列字首。

與其他 IBM MQ 應用程式的交互通訊

可以在 Apache Qpid JMS 應用程式與其他 IBM MQ 應用程式之間傳送訊息。 例如， Apache Qpid 應用程式可以發佈主題的訊息，且 MQ Light 應用程式可以透過建立訂閱來接收訊息。

Apache Qpid JMS 應用程式也可以發佈傳統 IBM MQ 應用程式所耗用的訊息，例如使用 MQSUB API 呼叫來訂閱相同主題。

同樣地， Apache Qpid JMS 應用程式可以訂閱傳統 IBM MQ 應用程式在其上發佈訊息的 IBM MQ 主題。

只要兩個用戶端指定相同的共用名稱和主題型樣， Apache Qpid JMS 應用程式也可以與 MQ Light 應用程式共用訂閱。

請注意，為了執行此動作， Apache Qpid JMS 應用程式不得與用戶端 ID 連接。 這可確保兩個應用程式所使用的 IBM MQ 訂閱名稱相同。

小心: Apache Qpid JMS 應用程式無法與 IBM MQ JMS 應用程式共用訂閱。

Apache Qpid JMS 限制

支援下列 JMS 功能:

- 用戶端確認、自動確認及 dups 正常確認模式 (DUPS_OK_ACKNOWLEDGE)
  - 使用或不使用認證進行連接
  - 在主題目的地上建立消費者
  - 在主題目的地上建立可延續消費者
  - 在主題目的地上建立共用消費者
  - 在主題目的地上建立共用可延續消費者
  - 用戶端確認及自動確認模式
  - 訊息確認和階段作業確認
  - 取消訂閱可延續訂閱
  - V 9.3.0 建立暫時佇列
  - V 9.3.0 在佇列或暫時佇列目的地上建立消費者
  - V 9.3.0 JMS MessageListeners
  - V 9.3.0 接收內文的 JMS 消費者; 稱為 `Consumer.receiveBody()` 的 JMS 2.0 方法

  - V 9.3.0 支援下列 JMS 訊息類型:
    - BytesMessage
    - MapMessage
    - ObjectMessage
    - StreamMessage
    - TextMessage
  - V 9.3.0 從佇列瀏覽訊息

AMQP 用戶端不支援下列 JMS 功能:

- 使用交易式階段作業和交易式 JMSContexts
  - 使用訊息選取器
  - 使用 **nolocal** 屬性
  - 交易式階段作業的使用
  - 使用遞送延遲
  - 在 IBM MQ 9.3.0 中，從佇列瀏覽訊息。
  - 建立多個具有相同用戶端 ID 和主題的可延續訂閱或消費者
  - V 9.3.0 JMS 暫時主題
  - 不支援 AMQP 過濾器。

V 9.3.3 從 IBM MQ 9.3.3 開始，下列附註不再適用於 Continuous Delivery 使用者。

小心: V 9.3.0 用戶端確認: 如果要使用任何未解決的 AMQP 訊息傳送，亦即，當需要用戶端確認訊息時，則在使用用戶端確認模式時，必須及時傳送訊息確認通知，其中任一用戶端必須及時解決，或考量將佇列管理程式內容 MARKINT (**MsgMarkBrowseInterval**) 設為較高的值。

MsgMarkBrowseInterval 的預設值是 5 秒。 如果應用程式未在此預設值內穩定，則可能會看到重複訊息。 為了避免訊息重複，您必須相應地增加 **MsgMarkBrowseInterval** 的值，最好將它設為 NOLIMIT ，以代表無限制的時間間隔。 在確定訊息之前，如果應用程式損毀或中斷連線，則訊息可供另一個應用程式使用。

如需相關資訊，請參閱 **MsgMarkBrowseInterval** 。 因為這是佇列管理程式內容，所以您設定的值將套用至連接至該佇列管理程式的所有應用程式。

在 AMQP 上， **MsgMarkBrowseInterval** 僅適用於佇列，不適用於訂閱。

下載範例 AMQP 用戶端

IBM MQ 不會隨附 AMQP 用戶端，但您可以下載 MQ Light 用戶端，或根據 Apache Qpid 程式庫下載開放程式碼 AMQP 用戶端。 如需相關資訊，請參閱 IBM MQ Light 及 Apache Qpid。

您也可以根據 Apache Qpid 程式庫下載其他開放程式碼 AMQP 用戶端。 如需相關資訊，請參閱 https://qpid.apache.org/index.html。

小心: IBM 支援中心無法提供這些用戶端套件的配置或問題報告支援，任何使用問題或程式碼問題報告都應該導向個別專案。

將 AMQP 用戶端部署至 IBM MQ

當應用程式備妥可部署時，它需要其他企業應用程式的所有監視、可靠性及安全功能。 它也可以與其他企業應用程式交換資料。

部署 AMQP 用戶端之後，您可以與 IBM MQ 應用程式交換訊息。 例如，如果您使用 AMQP 用戶端來傳送 JavaScript 字串訊息，則 IBM MQ 應用程式會收到 MQ 訊息，其中 MQMD 的格式欄位設為 MQSTR。

管理 AMQP 通道

AMQP 通道的管理方式與其他 MQ 通道相同。 您可以使用 MQSC 指令、PCF 指令訊息或 IBM MQ Explorer 來定義、啟動、停止及管理通道。 在 建立及使用 AMQP 通道中，提供範例指令來定義用戶端並開始將用戶端連接至佇列管理程式。

啟動 AMQP 通道時，您可以透過連接 AMQP 1.0 用戶端來測試它。 例如， MQ Light、 Apache Qpid Proton 或 Apache Qpid JMS。

相關工作

建立及使用 AMQP 通道

保護 AMQP 用戶端安全

ALW MQ Light、 Apache Qpid JMS 及 AMQP (進階訊息佇列作業通訊協定)

MQ Light 用戶端、 Apache Qpid 用戶端 (例如 Apache Proton) 及 Apache Qpid JMS API 基於 OASIS 標準 AMQP 1.0 佈線通訊協定。 AMQP 指定如何在傳送端與接收端之間傳送訊息。 當應用程式將訊息傳送至訊息分配管理系統 (例如 IBM MQ) 時，應用程式會作為傳送端。 IBM MQ 會在傳送訊息至 AMQP 應用程式時充當傳送端。

AMQP 的部分好處如下:

- 開放式標準化通訊協定
- 與其他開放程式碼 AMQP 1.0 用戶端的相容性
- 許多開放程式碼用戶端實作可用

雖然任何 AMQP 1.0 用戶端都可以連接至 AMQP 通道，但部分 AMQP 特性不受支援，例如交易或多個階段作業。

如需相關資訊，請參閱 AMQP.org 網站 及 OASIS Standard AMQP 1.0 PDF。

MQ Light 和 Apache Qpid JMS API 具有下列傳訊特性:

- 最多一次訊息遞送
- 至少一次訊息遞送
- 主題字串目的地定址
- 訊息及目的地延續性
- 共用目的地容許多個訂閱者共用工作量
- 用戶端接管以輕鬆解決當掉的用戶端
- 可配置先讀訊息
- 訊息的可配置確認通知

如需 Apache Qpid JMS API 的完整文件，請參閱 Qpid JMS。

相關工作

建立及使用 AMQP 通道

保護 AMQP 用戶端安全

ALW AMQP 1.0 支援

AMQP 通道提供對 AMQP 1.0-compliant 應用程式的支援層次。

AMQP 通道支援 AMQP 1.0 通訊協定的子集。 您可以將 AMQP 1.0 相容用戶端連接至 IBM MQ AMQP 通道。 若要使用 AMQP 通道所支援的所有傳訊特性，您必須正確地設定特定 AMQP 1.0 欄位的值。

本資訊概述 AMQP 欄位必須格式化的方式，並列出 AMQP 通道不支援之 AMQP 1.0 規格的特性。

AMQP 1.0 規格的下列特性不受支援或限制使用:

ATTACH 框架

V 9.3.0

AMQP 通道預期 ATTACH 訊框中的功能包含下列其中一項:

```
topic
temporary queue
queue
shared
```

功能暗示物件類型，在多重功能的情況下，選取功能的優先順序為 topic、temporary-queue、queue。

如果功能不包含期望值，則預設功能是 topic。 任何其他功能都會被忽略。

註: 部分 AMQP 用戶端不會設定這些功能，將取得發佈/訂閱的 IBM MQ 預設行為。 例如， Quarkus Reactive Messaging AMQP 1.0 連接器僅從 2.8.0CR1 版開始設定功能。

V 9.3.0 對於來源或目標， AMQP 通道預期 ATTACH 訊框上的 `distribution-Mode` 包含下列其中一項:

- **移動**
- **複製**

其中 `move` 表示破壞性取得，而 `copy` 表示瀏覽器。

註: 如果未設定 `distribution-Mode` ，或設為 `copy` 以外的任何其他值，則會假設 `move` 。

鏈結名稱

AMQP 通道預期 AMQP 鏈結名稱遵循下列 五種 格式之一:

- 一般主題 (用於發佈及訂閱)
  - 發佈訊息: 一般主題字串 (例如，鏈結名稱 `"/sports/football"`) 會導致在 `/sports/football` 主題上發佈訊息。
  - 訂閱主題以接收訊息: 一般主題字串 (例如，鏈結名稱 `"/sports/football"` 會導致在 `/sports/football` 主題上定義訂閱。
- 專用詳細主題 (用於訂閱)
  - 說明專用訂閱的詳細主題字串，格式如下: `"private:`*`topic string`*`"` (例如: `"private:/sports/football"`)。 行為與一般主題字串相同。 `private` 宣告會區分特定 AMQP 用戶端特定的訂閱與用戶端之間共用的訂閱。
- 共用詳細主題 (用於訂閱)
  - 說明共用訂閱的詳細主題字串，格式如下: `"share:`*`share name`*`:`*`topic string`*`"` (例如: `"share:bbc:/sports/football"`)。
- V 9.3.0 佇列 (用於生產者和消費者的點對點傳訊)
  - 傳送訊息的生產者; 佇列名稱字串會使生產者在佇列上傳送訊息。
  - 接收訊息的消費者; 佇列名稱字串會使消費者從佇列接收訊息。
- V 9.3.0 空白 (針對暫時佇列上的點對點傳訊)
  - 生產者在暫時佇列上傳送訊息; 空白會使生產者在暫時佇列上傳送訊息。
  - 消費者在暫時佇列上接收訊息; 空白會導致消費者從暫時佇列接收訊息。

如需 AMQP 訊息與 IBM MQ 訊息之間的對映方式的相關資訊，請參閱 第 575 頁的『將 AMQP 欄位對映至 IBM MQ 欄位 (送入訊息)』。

主題字串、共用名稱及用戶端 ID 的長度上限

主題字串、共用名稱及用戶端 ID 必須包含在 10237 個位元組內。 此外，用戶端 ID 的長度上限為 256 個字元。

這些長度上限表示您可以具有下列其中一項:

- 非常長的主題字串，前提是共用名稱很短
- 長共用名稱，但簡短主題字串

儲存器 ID

AMQP 通道預期 AMQP Open 的儲存器 ID 包含唯一的 AMQP 用戶端 ID。 AMQP 用戶端 ID 的長度上限為 256 個字元， ID 可以包含英數字元、百分比符號 (%)、斜線 (/)、句點 (.) 及底線 (_)。

會議

AMQP 通道僅支援單一 AMQP 階段作業。 嘗試建立多個 AMQP 階段作業的 AMQP 用戶端會收到錯誤訊息，且會與通道中斷連線。

交易

AMQP 通道不支援 AMQP 交易。 嘗試協調新交易的 AMQP 連接訊框或嘗試宣告新交易的 AMQP 傳送訊框遭到拒絕，並出現錯誤訊息。

遞送狀態

AMQP 通道僅支援處置訊框「已接受」、「已發行」或「已修改」的遞送狀態。 請注意，在使用「已修改」狀態時， AMQP 通道不支援無法遞送-here 選項。

相關工作

建立及使用 AMQP 通道

保護 AMQP 用戶端安全

V 9.3.0 ALW AMQP 通道上的點對點支援

IBM MQ AMQP 通道支援將訊息傳送至佇列及從佇列接收訊息。

當傳送 AMQP 連接訊框時， Apache Qpid™ JMS 程式庫之類的 AMQP 用戶端會要求 queue 或 `temporary-queue` 功能。 功能可讓 AMQP 通道將物件識別為佇列、暫時佇列或主題。 如果沒有佇列或暫時佇列功能，甚至沒有任何功能，則會假設要求是用於主題。

IBM MQ AMQP 通道提供下列的佇列類型支援:

佇列接收及傳送
訊息可以傳送至佇列，並從佇列耗用。 對於耗用訊息，支援同步和非同步模式。

V 9.3.0 佇列瀏覽訊息
除了將訊息放入佇列及從佇列取得訊息之外，也可以從佇列瀏覽訊息。

暫時佇列支援
訊息可以傳送至暫時佇列，並從暫時佇列耗用。 請注意，如果用來建立暫時佇列的相同暫時佇列物件也用來刪除暫時佇列，則支援刪除暫時佇列。

SYSTEM.DEFAULT.MODEL.QUEUE ，且暫時佇列的字首會是 `AMQP.*`。

依預設， SYSTEM.DEFAULT.MODEL.QUEUE 是暫時動態佇列，但您可以使用 SYSTEM.DEFAULT.MODEL.QUEUE 佇列上的 **Definition type** 內容，將佇列變更為永久動態佇列。

永久動態佇列 (permanent dynamic queue)
當 AMQP 用戶端 (例如 Apache Qpid JMS 程式庫) 傳送具有 `detach` 訊框且 **closed** 屬性設為 *true* 的要求時，會刪除永久動態佇列。

重要:

Qpid JMS 行為:
您必須呼叫 Qpid JMS API 指令，例如 `javax.jms.TemporaryQueue.delete()` 方法，以在使用之後毀損佇列，且此處理程序也會清除佇列上呈現的訊息。

如果您沒有發出這類指令，當連線關閉時，佇列仍會保留任何仍存在的訊息。

-

暫時動態佇列 (temporary dynamic queue)
當 AMQP 用戶端關閉連線時，會刪除暫時動態佇列。

重要:

Qpid JMS 行為:
如果您呼叫 Qpid JMS API 指令 (例如， `javax.jms.TemporaryQueue.delete()` 方法、關閉 JMS 連線或連線岔斷)，則會刪除佇列並遺失任何訊息。

關閉 JMS 階段作業本身不會導致刪除暫時佇列，即使暫時佇列可能已使用 `javax.jms.Session.createTemporaryQueue()` 方法建立。

-

相關工作
建立及使用 AMQP 通道
保護 AMQP 用戶端安全

ALW 對映 AMQP 和 IBM MQ 訊息欄位

AMQP 訊息由標頭、遞送註釋、訊息註釋、內容、應用程式內容、內文及標底組成。

AMQP 訊息由下列組件組成:

標頭
選用標頭包含訊息的五個固定屬性:

- **durable** -指定延續性需求
- **priority** -相對訊息優先順序
- **ttl** -存活時間 (毫秒)
- **first-acquirer** -如果這是 true，則任何其他鏈結都未獲得訊息
- **delivery-count** -先前未順利完成的遞送嘗試次數。

遞送-註釋
選用項目。 針對不同的預期對象，指定訊息的非標準標頭屬性。 遞送註釋會將資訊從傳送端對等節點傳送至接收端對等節點。

訊息註釋
選用項目。 針對不同的預期對象，指定訊息的非標準標頭屬性。 訊息註釋區段用於訊息的內容，這些內容以基礎架構為目標，且應該在每個遞送步驟之間延伸。

內容
選用項目。 此組件相當於 MQ 訊息描述子。 它包含下列固定欄位:

- **message-id** -應用程式訊息 ID
- **user-id** -建立使用者的 ID
- **to** -訊息目的地節點的位址
- **subject** -訊息的主旨
- **reply-to** -傳送回覆的節點
- **correlation-id** -應用程式相關性 ID
- **content-type** -MIME 內容類型
- **content-encoding** -MIME 內容類型。 用作內容類型的修飾元。
- **absolute-expiry-time** -將此訊息視為過期的時間

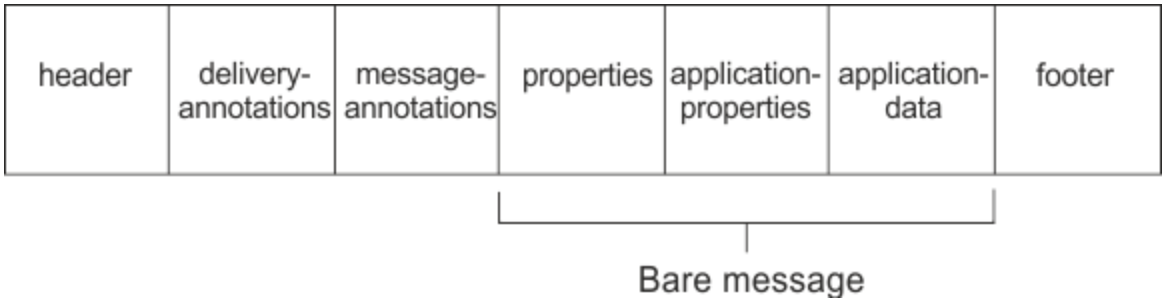
- **creation-time** -建立此訊息的時間
- **group-id** -此訊息所屬的群組
- **group-sequence** -此訊息在其群組內的序號
- **reply-to-group-id** -回覆訊息所屬的群組

應用程式-內容
相當於 MQ 訊息內容。

內文
相當於 MQ 使用者有效負載。

頁腳
選用項目。 標底用於僅在建構或查看整個裸機訊息 (例如，訊息雜湊、HMAC、簽章及加密詳細資料) 之後才能計算或評估的訊息或遞送的相關詳細資料。

下圖說明 AMQP 訊息格式:



內容、應用程式內容及應用程式資料組件稱為 "裸機訊息"。 這是寄件者所傳送的訊息，且不可變。 接收端會看到整個訊息，包括標頭、標底、遞送註釋及訊息註釋。

如需 AMQP 1.0 訊息格式的完整說明，請參閱 OASIS 標準，網址為 https://docs.oasis-open.org/amqp/core/v1.0/amqp-core-complete-v1.0.pdf。

相關工作
建立及使用 AMQP 通道
保護 AMQP 用戶端安全

ALW 將 IBM MQ 欄位對映至 AMQP 欄位 (送出訊息)

當發佈 IBM MQ 訊息並 IBM MQ 將其傳送至 AMQP 消費者時，它會將 IBM MQ 訊息的部分屬性延伸至相等的 AMQP 訊息屬性。

標頭 (header)

只有在標頭中的五個欄位之一包含非預設值時，才會包含標頭。 標頭中只會包含具有非預設值的欄位。 這五個標頭欄位最初衍生自對等的 `mq_amqp.Hdr` 內容 (如果已設定的話)，然後如下表所示進行修改:

表 *101:* 標頭欄位對映

欄位	預設值	值
可延續	false	如果 `MQMD.Persistence` 設為 `MQPER_PERSISTENT`，則為 true，否則為 false。
priority	4	從 `mq_amqp.Hdr.Pri`(若有設定)，或從 `MQMD.Priority`(若有設定)。 如果都未設定，請設為 4。
ttl	不適用	`MQMD.Expiry` (毫秒)。 如果 `MQMD.Expiry` 的值是 `MQEI_UNLIMITED`，則會設為 AMQP ttl 欄位的上限值

表 *101:* 標頭欄位對映 *(繼續)*		
欄位	**預設值**	**值**
第一收單銀行	false	從 mq_amqp.Hdr.Fac，如果已設定，則為 false，否則為 false。
遞送-計數	0	從 mq_amqp.Hdr.Dct，如果已設定，則為 0，否則為 0。

遞送註釋

依需要由 AMQP 通道設定。

訊息註釋

不包括。

內容

如果設定 **內容**，則會從對等的 mq_amqp.Prp 內容中取消修改這些內容。 如果訊息最初不是 AMQP 訊息 (亦即， PutAppl 類型不是 MQAT_AMQP)，則會產生內容區段，如下表中所述:

表 *102:* 內容欄位對映	
名稱	**值**
Message-ID	MQMD.MsgId 設為二進位。
使用者 ID	以網路位元組順序將 MQMD.UserIdentifier 的 UTF-8 格式設為二進位。
收件者	取得訊息的來源佇列，或發佈資訊的主題字串。
主旨	未設定。
回覆目的地	如果不是空白，則為 MQMD.ReplyToQ，否則為未設定。
相關性 ID	如果不是空白，則 MQMD.CorrelId 會設為二進位，否則不會設定。
content-type	未設定。
內容編碼	未設定。
absolute-expiry-time	未設定。
建立時間	MQMD.PutDate 和 MQMD.PutTime 欄位用來產生時間戳記。
group-id	未設定。
group-sequence	未設定。
reply-to-group-id	未設定。

application-properties

"usr" 群組中的所有 IBM MQ 內容都會新增為 **application-properties**。

主體

AMQP 通道會使用轉換來執行 get，以將 IBM MQ 有效負載轉換為 UTF-8。

如果 IBM MQ 有效負載不包含 AMQP 訊息，則會在內文中將 IBM MQ 有效負載設為格式 MQFMT_STRING 的單一字串資料區段 (提供的 UTF-8 轉換成功)，或設為單一二進位資料區段。

如果包含 AMQP 格式訊息，則會設為內文。 如果內文是 AMQP 序列，則在 AMQP 訊息之前的任何 IBM MQ 標頭 (不包括在訊息控點中傳回的訊息內容) 會以二進位值附加到前面。 否則會捨棄 IBM MQ 標頭。

標底

不包括標底。

相關工作

建立及使用 AMQP 通道

保護 AMQP 用戶端安全

相關參考

MQMD-訊息描述子

ALW 將 AMQP 欄位對映至 IBM MQ 欄位 (送入訊息)

當 AMQP 通道接收訊息並將其放置到 IBM MQ 時，它會將 AMQP 訊息的部分屬性傳播至對等的 IBM MQ 訊息屬性。

對映送入 AMQP 訊息時，適用下列限制:

- 如果內容組件中的 `message-id` 或 `correlation-id` 欄位是 UUID 或 ulong，則會拒絕訊息。
- 任何 `message-annotations` 都會導致拒絕訊息。
- 容許 `delivery-annotations` 和 `footer` 區段，但不會延伸到 IBM MQ 訊息。

下列子區段顯示 AMQP 訊息的 IBM MQ 表示式。

訊息描述子

表 *103: AMQP* 訊息的訊息描述子

欄位	**值**
StrucId	MQMD_STRUC_ID
版本	MQMD_VERSION_1
報告	MQRO_NONE
MsgType	MQMT_DATAGRAM
期限	取自 AMQP 訊息標頭中 `ttl` 欄位的值
意見	MQFB_NONE
編碼	MQENC_NORMAL
CodedCharSetId	1208 (UTF-8)
格式	請參閱有效負載
優先順序	取自 AMQP 訊息標頭中 `priority` 欄位的值。 如果已設定，則限制為上限為 9。 如果未設定，則採用預設值 4。
持續性	如果 AMQP 訊息標頭中的 `durable` 欄位設為 true，請設為 MQPER_PERSISTENT。 否則，請設為 MQPER_NOT_PERSISTENT。
MagId	佇列管理程式會配置唯一的 24 位元組 MsgId。
科雷爾德	從 AMQP 內容中的 `correlation-id` 欄位取得的值 (如果已設定)。 設為 24 位元組二進位值。 否則，請設為 MQCI_NONE/。
BackoutCount	0

表 *103: AMQP* 訊息的訊息描述子 *(繼續)*	
欄位	**值**
ReplyToQ	V 9.3.0 從 AMQP 內容中的 `reply-to` 欄位取得的值 (如果已設定)。 否則設為 ""。
回覆目的地佇列管理程式	""
V 9.3.0 報告	衍生自 AMQP 應用程式內容中的任何 JMS IBM 報告內容集的值。
UserIdentifier	設為連接至 AMQP 通道之已鑑別使用者的 ID
AccountingToken	MQACT_NONE
ApplIdentityData	十六進位字串。 設為 AMQP 通道 MQ 連線 ID 的最後 8 個位元組。
PutApplType	MQAT_AMQP
PutApplName	
PutDate	從 AMQP 內容的 `creation-time` 欄位中取得的值 (如果已設定)。 否則設為現行日期。
PutTime	從 AMQP 內容的 `creation-time` 欄位中取得的值 (如果已設定)。 否則，請設為現行時間。
ApplOriginData	""

訊息內容

設定訊息內容有兩個原因:

- 容許 AMQP 訊息的部分流經佇列管理程式，而不會影響訊息的有效負載。
- 容許選取 `application-properties`。

下表顯示從 AMQP 訊息設定的內容:

表 *104: AMQP* 訊息內容			
內容名稱	**MQRFH2 名稱**	**類型**	**說明**
AMQPListener	mq_amqp.Lis	MQTYPE_STRING	AMQP 通道的識別字串。 它是用來產生訊息，以便有興趣的人可以辨別放置訊息的版本 (例如，服務團隊在診斷問題時)。 此值未由佇列管理程式驗證，且不得在外部記載。
AMQPVersion	mq_amqp.Ver	MQTYPE_STRING	AMQP 訊息的版本。 如果不存在，則假設為 "1.0"。 佇列管理程式不會驗證此值。
AMQPClient	mq_amqp.Cli	MQTYPE_STRING	API 的識別字串。 它是用來將 AMQP 訊息傳送至通道，以便相關方可以辨別訊息放置的版本 (例如，服務團隊在診斷問題時)。 此值未由佇列管理程式驗證，且不得在外部記載。
AMQPDurable	mq_amqp.Hdr.Dur	MQ 類型 _ 布林	AMQP 訊息標頭中 `durable` 欄位的值 (如果已設定)。
AMQPPriority	mq_amqp.Hdr.Pri	MQTYPE_INT32	AMQP 訊息標頭中 `priority` 欄位的值 (如果已設定)。

表 *104: AMQP* 訊息內容 *(*繼續*)*			
內容名稱	**MQRFH2 名稱**	**類型**	**說明**
AMQPTtl	mq_amqp.Hdr.Ttl	MQTYPE_INT64	AMQP 訊息標頭中 `ttl` 欄位的值 (如果已設定)。
AMQPFirstAcquirer	mq_amqp.Hdr.Fac	MQ 類型 _ 布林	AMQP 訊息標頭中 `first-acquirer` 欄位的值 (如果已設定)。
AMQPDeliveryCount	mq_amqp.Hdr.Dct	MQTYPE_INT64	AMQP 訊息標頭中 `delivery-count` 欄位的值 (如果已設定)。
AMQPMsgId	mq_amqp.Prp.Mid	MQTYPE_STRING	AMQP 內容中 `message-id` 欄位的值 (如果設定為字串)。
		MQTYPE_BYTE_STRING	AMQP 內容中 `message-id` 欄位的值 (如果設為位元組字串)。
AMQPUserId	mq_amqp.Prp.Uid	MQTYPE_BYTE_STRING	AMQP 內容中 `user-id` 欄位的值 (如果已設定)。
AMQPTo	mq_amqp.Prp.To	MQTYPE_STRING	AMQP 內容中 `to` 欄位的值 (如果已設定)。
AMQPSubject	mq_amqp.Prp.Sub	MQTYPE_STRING	AMQP 內容中 `subject` 欄位的值 (如果已設定)。
AMQPReplyTo	mq_amqp.Prp.Rto	MQTYPE_STRING	AMQP 內容中 `reply-to` 欄位的值 (如果已設定)。
AMQPCorrelationId	mq_amqp.Prp.Cid	MQTYPE_STRING	AMQP 內容中 `correlation-id` 欄位的值 (如果設為字串)。
		MQTYPE_BYTE_STRING	AMQP 內容中 `correlation-id` 欄位的值 (如果設為位元組字串)。
AMQPContentType	mq_amqp.Prp.Cnt	MQTYPE_STRING	AMQP 內容中 `content-type` 欄位的值 (如果已設定)。
AMQPContentEncoding	mq_amqp.Prp.Cneing	MQTYPE_STRING	AMQP 內容中 `content-encoding` 欄位的值 (如果已設定)。
AMQPAbsoluteExpiry 時間	mq_amqp.Prp.Aet	MQTYPE_STRING	AMQP 內容中 `absolute-expiry-time` 欄位的值 (如果已設定)。
AMQPCreationTime	mq_amqp.Prp.Crt	MQTYPE_STRING	AMQP 內容中 `creation-time` 欄位的值 (如果已設定)。
AMQPGroupId	mq_amqp.Prp.Gid	MQTYPE_STRING	AMQP 內容中 `group-id` 欄位的值 (如果已設定)。
AMQPGroupSequence	mq_amqp.Prp.Gsq	MQTYPE_INT64	AMQP 內容中 `group-sequence` 欄位的值 (如果已設定)。
AMQPReplyToGroupId	mq_amqp.Prp.Rtg	MQTYPE_STRING	AMQP 內容中 `reply-to-group-id` 欄位的值 (如果已設定)。

AMQP 訊息中的每一個應用程式內容都設為 IBM MQ 訊息內容。 `application-properties` 區段必須以相同位元組為單位重新組成，因此適用下列限制:

- 如果 MQSETMP 驗證碼拒絕應用程式內容，則會拒絕訊息。 例如:
  - 內容名稱的長度限制為 MQ_MAX_PROPERTY_NAME_LENGTH。
  - 內容名稱必須遵循 Java ID 的 Java 語言規格所定義的規則。

  - 內容名稱不得以 JMS 或 usr.JMS 開頭，但可設定所記載的 JMS 內容除外。
  - 內容名稱不能是 SQL 關鍵字。
- 包含 Unicode 字元 U+002E (".") 的應用程式內容 會導致拒絕訊息。 內容必須可在 JMS 使用的 "usr" 內容群組中表示。
- 僅支援 null、boolean、byte、short、int、long、float、double、binary 及 string 內容。 具有任何其他類型的應用程式內容將導致拒絕訊息。

V 9.3.0 您可以使用 application-properties 來設定下列 JMS 內容:

- JMS_IBM_REPORT_EXCEPTION
- JMS_IBM_REPORT_EXPIRATION
- JMS_IBM_REPORT_COA
- JMS_IBM_REPORT_COD
- JMS_IBM_REPORT_PAN
- JMS_IBM_REPORT_NAN
- JMS_IBM_REPORT_PASS_MSG_ID
- JMS_IBM_REPORT_PASS_CORREL_ID
- JMS_IBM_REPORT_DISCARD_MSG

請注意，內容名稱和值與對等的 第 135 頁的『對映 JMS 提供者特定的欄位』 詳細資料一致，且會忽略無效的值。

有效負載

- 對於具有單一二進位資料區段的 AMQP body ， 二進位資料 (不包括 AMQP 位元) 會放置為 IBM MQ 有效負載，格式為 MQFMT_NONE。
- 對於具有單一字串資料區段的 AMQP body ， 會將字串資料 (不包括 AMQP 位元) 放置為 IBM MQ 有效負載，其格式為 MQFMT_STRING。
- 否則， AMQP body 會依現狀使用 MQFMT_AMQP 格式來形成有效負載。

相關工作

建立及使用 AMQP 通道

保護 AMQP 用戶端安全

ALW 訊息遞送可靠性

本節比較 MQ Light API 和 Apache Qpid JMS 的可靠性特性。

相關工作

建立及使用 AMQP 通道

保護 AMQP 用戶端安全

ALW MQ Light 訊息可靠性

MQ Light API 有四個特性可讓您控制與 AMQP 應用程式之間的訊息遞送可靠性。

它們是:

- 第 579 頁的『訊息服務品質 (QOS)』
- 第 579 頁的『訂閱者自動確認』
- 第 579 頁的『訂閱存活時間』
- 第 580 頁的『訊息持續性』

訊息服務品質 (QOS)

MQ Light API 提供兩種服務品質:

- 至多一次
- 至少一次

您可以選擇要發佈者和訂閱者使用的服務品質。

如果您使用 MQ Light 用戶端，請將用戶端或訂閱 **qos** 選項設為 *QOS_AT_MOST_ONCE* 或 *QOS_AT_LEAST_ONCE*。

如果您使用不同的 AMQP 用戶端，請視您要達到的服務品質而定，將傳送訊框 (適用於發佈者) 或處置訊框 (適用於訂閱者) 的 **settled** 屬性設為 *true* 或 *false*。

服務品質會決定何時從交談的 sending 端捨棄訊息。

發佈

如果發佈者選擇 **QOS** *0* (最多一次)，在捨棄其訊息副本之前，發佈者不會等待來自佇列管理程式的確認通知。

如果在傳送完成之前與佇列管理程式的連線失敗，訂閱者可能不會收到訊息。

如果發佈者選擇 **QOS** *1* (至少一次)，發佈者會等待佇列管理程式確認訊息已寫入訂閱者佇列，然後再捨棄其訊息副本。

如果在傳送期間與佇列管理程式的連線失敗，發佈者會在重新連接至佇列管理程式之後重新傳送訊息。

訂閱

如果訂閱者選擇 **QOS** *0*，佇列管理程式在捨棄其訊息副本之前不會等待來自訂閱者的確認通知。

如果訂閱者的連線在訂閱者收到訊息之前失敗，則該訊息可能會遺失。

如果訂閱者選擇 **QOS** *1*，佇列管理程式會先等待來自訂閱者的確認通知，再捨棄其訊息副本。

V 9.3.3 從 IBM MQ 9.3.3，會以批次方式移除已確認的訊息，以增進效能。 如需相關資訊，請參閱第 581 頁的『以批次方式從佇列中移除已確認的 AMQP 訊息』。

如果訂閱者的連線在訂閱者收到訊息之前失敗，佇列管理程式會保留該訊息。 當佇列管理程式重新連接時，佇列管理程式會將訊息重新傳送給訂閱者，如果訂閱是共用的，則會重新傳送給另一個訂閱者。

訂閱者自動確認

如果訂閱者選擇 **QOS** *1* (至少一次)，則必須確認在佇列管理程式捨棄其副本之前收到每一則訊息。 訂閱者可以決定何時確認訊息。

當 **auto-confirm** 設為 *true* 時，一旦用戶端透過網路順利接收訊息， MQ Light 用戶端就會自動確認每一個訊息的遞送。

這可確保如果網路失敗，訊息會重新遞送至應用程式。 不過，如果應用程式在確認訊息的 MQ Light 用戶端與處理訊息的應用程式之間失敗，應用程式仍有可能遺失訊息。

當 **auto-confirm** 設為 *false* 時， MQ Light 用戶端不會自動確認訊息的遞送，但會將它留給應用程式來決定何時應該確認。

這可讓應用程式在向佇列管理程式確認訊息現在已處理且可以捨棄之前，先更新外部資源 (例如資料庫或檔案)。

訂閱存活時間

當應用程式訂閱時，它會選擇在應用程式中斷連線之後是否繼續存在訂閱，以及儲存該訂閱之訊息的目的地。

MQ Light 訂閱選項 **ttl** 用來指定在應用程式中斷連線之後，訂閱繼續存在的時間 (毫秒)。 如果應用程式在此時間之前重新連接，則會回復訂閱，且應用程式可以繼續耗用來自該訂閱的訊息。

如果在沒有應用程式重新連接的情況下過了存活時間期間，則會移除訂閱，且會遺失儲存在其目的地的任何訊息，即使它們是持續訊息也一樣。

如果不要遺失訊息很重要，您必須指定應用程式的存活時間值，該值足以確保訊息在中斷期間不會遺失。

訊息持續性

訊息的持續性是由發佈和訂閱應用程式以及 IBM MQ 主題物件的配置所控制。

如果 AMQP 訂閱者使用 **QOS** *0* (最多一次) 並建立不可延續訂閱，則不論下列文字中說明的其他選項為何，AMQP 通道一律會將非持續訊息放入訂閱者佇列。

請注意，如果佇列管理程式已停止，則訂閱及訊息都會遺失。

如果 AMQP 發佈者將 AMQP **durable** 標頭設為 *true*，則 AMQP 通道會將持續訊息放入訂閱者佇列。

如果佇列管理程式因任何原因而停止，當佇列管理程式重新啟動時，訊息仍可供訂閱者使用。

如果未設定 **durable** 標頭， AMQP 通道會根據相關 IBM MQ 主題物件的 **DEFPSIST** 屬性，選擇已發佈訊息的持續性。

依預設，這是 SYSTEM.BASE.TOPIC，使用 **DEFPSIST** 屬性 *NO* (非持續性)。

小心: 更新版本的 MQ Light 用戶端不支援設定 AMQP 可延續標頭。

相關工作

建立及使用 AMQP 通道

保護 AMQP 用戶端安全

ALW Apache Qpid JMS 訊息可靠性

Apache Qpid™ JMS 程式庫有四個特性，可讓您控制與 AMQP 應用程式之間訊息遞送的可靠性。

這些適用於:

- 第 580 頁的『發佈』 V 9.3.0 /Producer for point-to point messaging
  - 訊息有效期限
  - 訊息持續性
- 第 580 頁的『訂閱』
  - 訂閱延續性
  - 階段作業確認模式 V 9.3.0 (也適用於消費者點對點傳訊)

發佈

訊息有效期限

設定 JMS 生產者的存活時間值，會影響提供給該訊息生產者所發佈訊息的到期時間。

確保 JMS 生產者的存活時間值足夠大，以在訊息到期之前耗用它們。

或者，取消設定存活時間值可防止訂閱佇列中的訊息到期。

訊息持續性

設定 JMS 訊息產生者的遞送模式會設定發佈至指定主題之 IBM MQ 訊息的持續性。

請確定您使用 **DeliveryMode**。持續保存在佇列管理程式結束或中斷時必須保留的訊息。

訂閱

訂閱延續性

AMQP 通道支援使用 JMS 建立消費者方法的可延續版本來建立可延續訂閱:

- **createDurableConsumer()**
- **createSharedDurableConsumer()**

階段作業確認模式

若要保證已耗用的訊息在從 IBM MQ 訂閱佇列中移除之前已完全處理，請使用 **Session** 建立 JMS 階段作業。CLIENT_ACKNOWLEDGE 模式，並使用 **message.acknowledge()** 方法來確認此訊息及先前在此階段作業上收到的任何其他訊息。

相關概念

開發 AMQP 用戶端應用程式

AMQP API 的 IBM MQ 支援可讓 IBM MQ 管理者建立 AMQP 通道。 啟動時，此通道會定義埠號，以接受來自 AMQP 用戶端應用程式的連線。

V 9.3.3 以批次方式從佇列中移除已確認的 AMQP 訊息

如果 AMQP 應用程式正在使用 QOS_AT_LEAST_ONCE (1) 訊息遞送，則 AMQP 服務會等待來自應用程式的確認通知，然後它會捨棄將該訊息傳送至應用程式之後所保留的訊息副本。 從 IBM MQ 9.3.3 開始，已確認的訊息會以批次方式從佇列中移除，而不是個別移除，以改善效能。

關於這項作業

若為 Long Term Support 及 Continuous Delivery 之前 IBM MQ 9.3.3，則會個別從佇列中移除每一則訊息。

從 IBM MQ 9.3.3 開始，您可以使用兩個系統內容 **com.ibm.mq.AMQP.BATCHSZ** 和 **com.ibm.mq.AMQP.BATCHINT** 來細部調整批次中的確認通知處理，以增進效能:

com.ibm.mq.AMQP.BATCHSZ
此屬性定義 AMQP 服務移除訊息之前要接收的確認通知數上限。 數字可以在 1 到 9999 的範圍內。 如果設定無效的數字，或指定的數字超出範圍，則會使用預設值 50。

批次大小不會影響傳送訊息的方式。 訊息一律個別傳送，然後在 AMQP 服務收到確認通知之後以批次方式移除。 批次的實際大小可以小於 **com.ibm.mq.AMQP.BATCHINT** 指定的值。 例如，如果 **com.ibm.mq.AMQP.BATCHINT** 屬性所設定的期間到期，則批次會完成。

com.ibm.mq.AMQP.BATCHINT
此屬性定義 AMQP 服務在佇列上保留已確認訊息的時間量 (毫秒)。 如果批次未滿，則會在此期間之後清除批次。 您可以指定從 1 到 999 999 999 999 的任何毫秒數。 預設值為 50。 如果您未指定此屬性的值，則會使用預設值 50。

附註:

1. AMQP 服務是否在捨棄訊息之前等待確認通知，取決於應用程式用於訊息遞送的下列兩種服務品質中的哪一種:
   - QOS for QOS_AT_MOST_ONCE (0)

     如果 AMQP 應用程式正在使用此服務品質，則它不會確認訊息，因此 AMQP 服務會在將訊息傳送至應用程式之後捨棄訊息，而不等待確認通知。
   - QOS_AT_LEAST_ONCE (1)

     如果 AMQP 應用程式正在使用此服務品質，則它會確認訊息，因此 AMQP 服務會在將每一則訊息傳送至應用程式之後保留該訊息的副本，直到它收到來自應用程式的確認通知為止。 如果應用程式在確認訊息之前中斷或中斷連線，則該訊息可供其他應用程式使用。 AMQP 服務在確認之前不會從佇列中移除訊息。
2. MQ Appliance **com.ibm.mq.AMQP.BATCHSZ** 和 **com.ibm.mq.AMQP.BATCHINT** 系統內容不適用於 IBM MQ Appliance。 在 IBM MQ Appliance 上使用預設值 50。

.

程序

使用 **com.ibm.mq.AMQP.BATCHSZ** 和 **com.ibm.mq.AMQP.BATCHINT** 系統內容來細部調整批次中的確認通知處理。

從 IBM MQ 9.3.3 開始，當建立佇列管理程式時， amqp_java.properties 檔會包含系統內容的下列預設值:

```
-Dcom.ibm.mq.AMQP.BATCHSIZE=50
-Dcom.ibm.mq.AMQP.BATCHINT=50
```

視耗用的訊息速率而定，您可以細部調整批次中確認通知的處理，以增進效能。 已移轉的佇列管理程式在 amqp_java.properties 檔中沒有這些內容。 因此，對於已移轉的佇列管理程式，或如果未設定內容，則會使用預設值。 您可以新增這些內容來細部調整值，以達到最佳化效能。

當符合下列其中一個條件時，會批次移除已確認的訊息:

- 已確認訊息數達到 **com.ibm.mq.AMQP.BATCHSZ**。
- 在批次開始之後已超出 **com.ibm.mq.AMQP.BATCHINT** 。
- 在滿足兩個先前的條件之前，應用程式會中斷或關閉佇列或主題。

ALW 使用 IBM MQ 之 AMQP 用戶端的拓蹼

協助您開發 AMQP 用戶端以使用 IBM MQ 的拓蹼範例。

相關工作

建立及使用 AMQP 通道

保護 AMQP 用戶端安全

ALW 透過 IBM MQ 進行通訊的 AMQP 用戶端

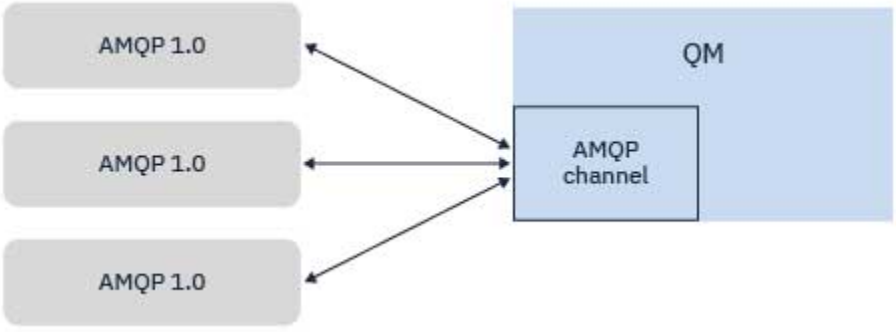
您可以使用 IBM MQ 作為任何符合 AMQP 1.0 的應用程式的傳訊提供者。 雖然任何 AMQP 1.0 用戶端都可以連接至 AMQP 通道，但部分 AMQP 特性不受支援，例如交易或多個階段作業。

透過定義一個以上 AMQP 通道， AMQP 1.0 用戶端可以連接至佇列管理程式，並將訊息傳送至主題字串。用戶端也可以訂閱主題型樣，以接收符合該型樣的訊息。

在下列實務範例中，傳送及接收訊息的應用程式只有 AMQP 1.0 應用程式。

應用程式可以選擇是否持續保存訂閱主題字串所建立的目的地，以便在應用程式暫時失去與佇列管理程式的連線時不會遺失訊息。

應用程式也可以選擇從目的地清除訊息之前要保留的時間長度。



相關工作

建立及使用 AMQP 通道

保護 AMQP 用戶端安全

ALW AMQP 用戶端與 IBM MQ 應用程式交換訊息

透過定義及啟動 AMQP 通道， AMQP 1.0 應用程式可以發佈現有 MQ 應用程式所接收的訊息。 透過 AMQP 通道發佈的訊息全都傳送至 MQ 主題，而不是 MQ 佇列。 已使用 MQSUB API 呼叫建立訂閱的 MQ 應用程式會接收 AMQP 1.0 應用程式發佈的訊息，前提是 MQ 應用程式使用的主題字串或主題物件符合 AMQP 用戶端發佈的主題字串。

AMQP 訊息資料、屬性及內容設定在 MQ 應用程式所接收的 MQ 訊息上。 如需 AMQP 至 MQ 訊息對映的相關資訊，請參閱 第 575 頁的『將 AMQP 欄位對映至 IBM MQ 欄位 (送入訊息)』。

如果 MQ 應用程式已建立可延續的訂閱，則 AMQP 應用程式發佈的訊息會儲存在支援訂閱的佇列上。 然後，當 MQ 應用程式回復其訂閱時，該應用程式會收到這些訊息。 如果 AMQP 應用程式指定訊息存活時間，且 MQ 應用程式未在存活時間內重新連接，則佇列中的訊息已過期。

AMQP 1.0 應用程式也可以耗用現有 MQ 應用程式發佈的訊息。 MQ 應用程式發佈至 MQ 主題或主題字串的訊息由 AMQP 1.0 應用程式接收，前提是應用程式已訂閱符合已發佈主題字串的主題型樣。

如果 AMQP 1.0 應用程式指定訂閱的存活時間值，且 AMQP 應用程式中斷連線超過存活時間，則佇列管理程式會使訂閱過期，且會遺失儲存在訂閱佇列上的任何訊息。

MQMD 欄位、訊息內容及應用程式資料設定在 AMQP 應用程式所接收的 AMQP 訊息上。 如需 MQ 至 AMQP 訊息對映的相關資訊，請參閱 第 573 頁的『將 IBM MQ 欄位對映至 AMQP 欄位 (送出訊息)』。

相關工作

建立及使用 AMQP 通道

保護 AMQP 用戶端安全

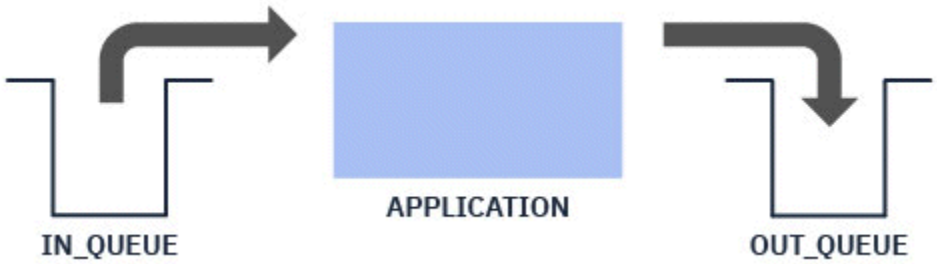
ALW 配置 AMQP 用戶端以直接與 IBM MQ 佇列上的應用程式互動

V 9.3.0 IBM MQ AMQP 實作支援發佈/訂閱及點對點。 對於任何不支援點對點的 AMQP 用戶端，請使用下列步驟將訊息傳送至佇列，或從佇列接收訊息。

概觀

例如，假設有一個應用程式從輸入佇列 IN_QUEUE 取得訊息，並將這些訊息放入輸出佇列 OUT_QUEUE。 AMQP 用戶端可以將訊息放置到 IN_QUEUE，並從 OUT_QUEUE 取得訊息

註: 應用程式本身不需要任何變更。



若要讓 AMQP 發佈者將訊息放置到佇列中，您需要為 AMQP 用戶端正在發佈至其中的主題字串建立管理訂閱，並具有預期佇列的目的地; 請參閱 第 583 頁的『將訊息傳送至應用程式:』。

若要讓 AMQP 訂閱者從佇列取得訊息，您需要將佇列取代為相同名稱的別名佇列，並以主題物件 (代表 AMQP 用戶端所訂閱的主題字串) 的目標; 請參閱 第 584 頁的『從應用程式取得訊息:』 ;

將訊息傳送至應用程式:

應用程式已從 IN_QUEUE 中挑選訊息，且您想要 AMQP 用戶端能夠發佈訊息，以便它們進入此佇列供應用程式處理。

若要這樣做，您可以建立新的管理訂閱，其中此訂閱從其中接收訊息的主題字串，是 AMQP 用戶端發佈至其中的主題字串。 此訂閱的目的地佇列是應用程式 IN_QUEUE 的輸入佇列。

任何發佈至該管理訂閱的已定義主題字串的訊息都會遞送至已定義的目的地，在本例中為 IN_QUEUE。

假設 AMQP 用戶端發佈至主題字串 /application/in，您可以使用下列 MQSC 指令建立管理訂閱 APP_IN:

```
DEF SUB(APP_IN) TOPICSTR('/application/in') DEST('IN_QUEUE')
```

當您定義此物件時，會發佈至 /application/in 的所有訊息都會遞送至目的地 IN_QUEUE，應用程式會以其他應用程式放入此佇列的任何其他訊息的相同方式來挑選它們。

從應用程式取得訊息:

應用程式正在將訊息放置到 OUT_QUEUE，其他用戶端可以在其中挑選並處理這些訊息。

不過，在此情況下，您想要改為將訊息遞送至 AMQP 用戶端，但 AMQP 用戶端僅使用發佈/訂閱，且無法直接從佇列中挑選訊息。

若要以訂閱 AMQP 用戶端取代先前接收訊息的用戶端，您需要為 AMQP 用戶端訂閱的主題字串及別名佇列建立主題物件。

小心: 如果您定義別名佇列，然後在任何 AMQP 用戶端有機會訂閱之前啟動生產端應用程式，則生產端應用程式傳送至「佇列」(現在是主題) 的訊息將會遺失，因為沒有訂閱者。

此文字中說明的變更僅以訂閱 AMQP 用戶端取代先前接收訊息的用戶端。 若要使用 AMQP 與其他用戶端的組合來取得訊息，需要進行更廣泛的變更。

假設 AMQP 用戶端訂閱主題字串 /application/out，您可以使用下列 MQSC 指令來定義主題物件 APP_OUT :

```
DEF TOPIC(APP_OUT) TOPICSTR('/application/out')
```

遞送至此主題物件的任何訊息都會遞送至訂閱相同主題字串的 AMQP 用戶端。

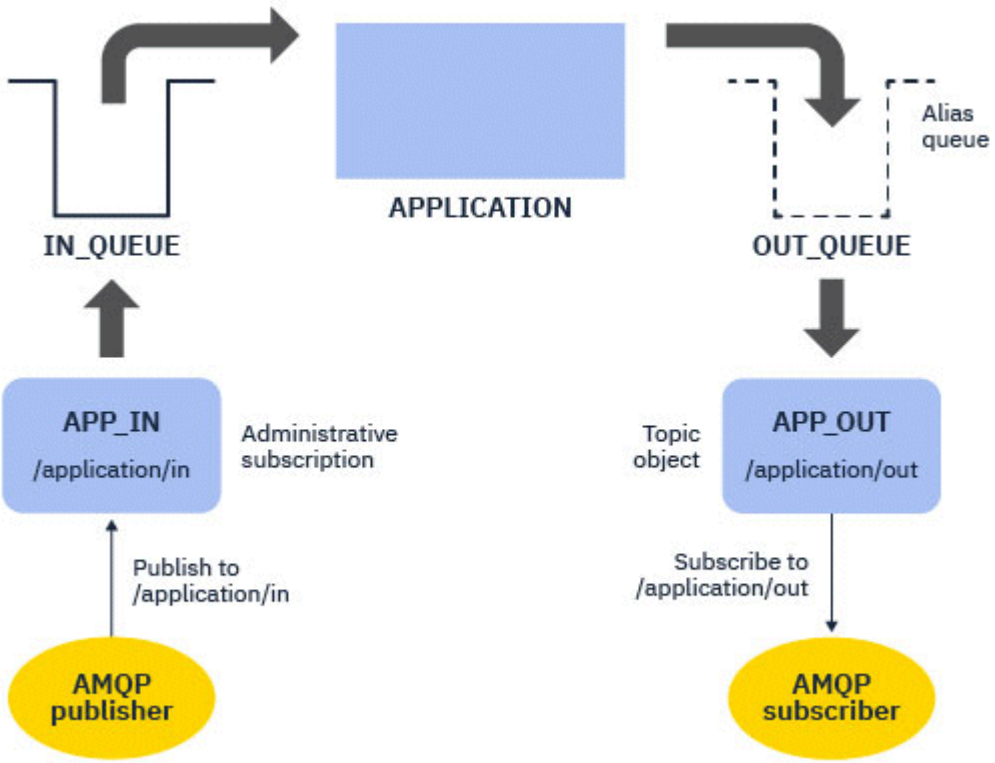
然後，您需要確保將應用程式放置到 OUT_QUEUE 的訊息遞送到這個新的主題物件，以便將它們傳送到訂閱用戶端。

若要這麼做，請使用下列 MQSC 指令，將現有佇列 OUT_QUEUE 取代為相同名稱的別名佇列，取代為剛建立之主題物件的目標類型:

```
DEF QALIAS(OUT_QUEUE) TARGTYPE(TOPIC) TARGET(APP_OUT)
```

現在，應用程式放置到 OUT_QUEUE 的訊息不會等待在佇列上挑選; 而是會遞送到這個別名佇列的目標，即新的主題物件 APP_OUT。

AMQP 用戶端會訂閱此主題物件 /application/out 所代表的主題字串，然後會從別名佇列接收任何傳送至此主題物件的訊息。



相關工作

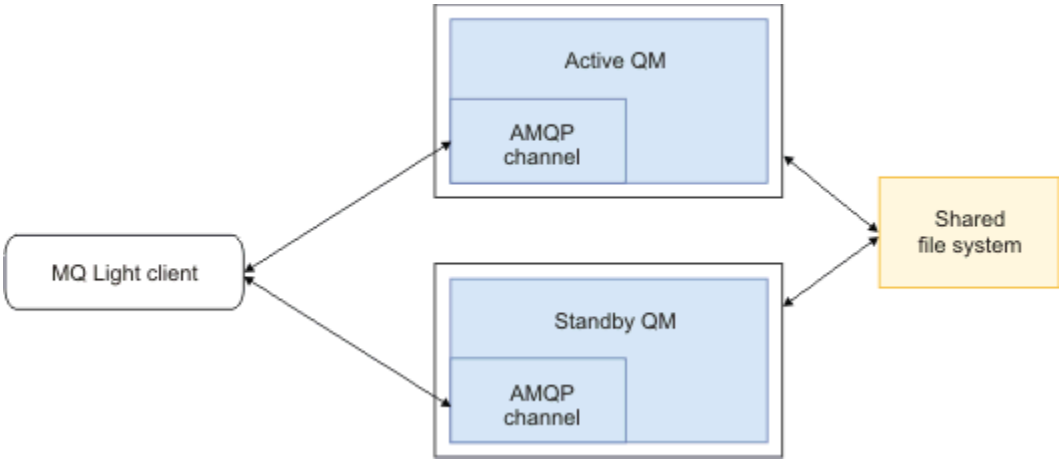
建立及使用 AMQP 通道

保護 AMQP 用戶端安全

ALW 配置 AMQP 用戶端以取得高可用性

您可以配置 AMQP 1.0 應用程式，以連接至 IBM MQ 多重實例佇列管理程式的作用中實例，並以高可用性 (HA) 配對失效接手至多重實例佇列管理程式的待命實例。 若要這樣做，請使用兩個 IP 位址及埠配對來配置 AMQP 應用程式。

您可以使用自訂函數來配置 AMQP 用戶端 API，此函數會在用戶端失去與伺服器的連線時呼叫。 此功能可以連接至替代 IP 位址，例如待命 IBM MQ 佇列管理程式或原始 IP 位址。 對於其他 AMQP 用戶端，如果用戶端支援配置多個連線端點，請使用兩個主機-埠配對來配置應用程式，並使用 AMQP 程式庫提供的重新連接特性來切換至待命佇列管理程式。



相關工作

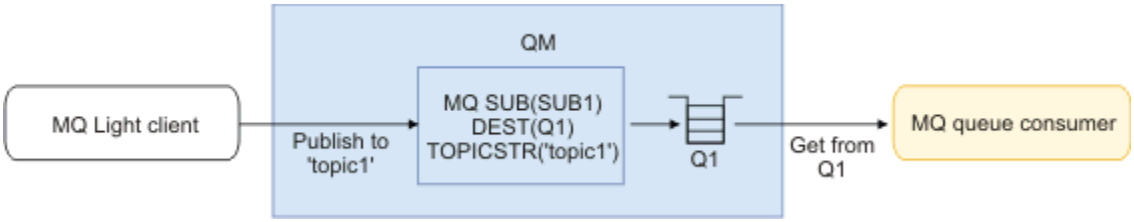
建立及使用 AMQP 通道

保護 AMQP 用戶端安全

ALW 配置 AMQP 用戶端的發佈/訂閱

AMQP 用戶端可以發佈至具有 IBM MQ 訂閱的主題，該訂閱會將訊息遞送至現有應用程式所讀取的 IBM MQ 佇列。 如果您想要 AMQP 1.0 應用程式將訊息傳送至配置為從佇列讀取的現有 IBM MQ 應用程式，您必須在佇列管理程式上定義受管理 IBM MQ 訂閱。

將訂閱配置為使用符合 AMQP 應用程式所使用的主題字串的主題型樣。 將訂閱目的地設為 IBM MQ 應用程式從中取得或瀏覽訊息的佇列名稱。



相關工作

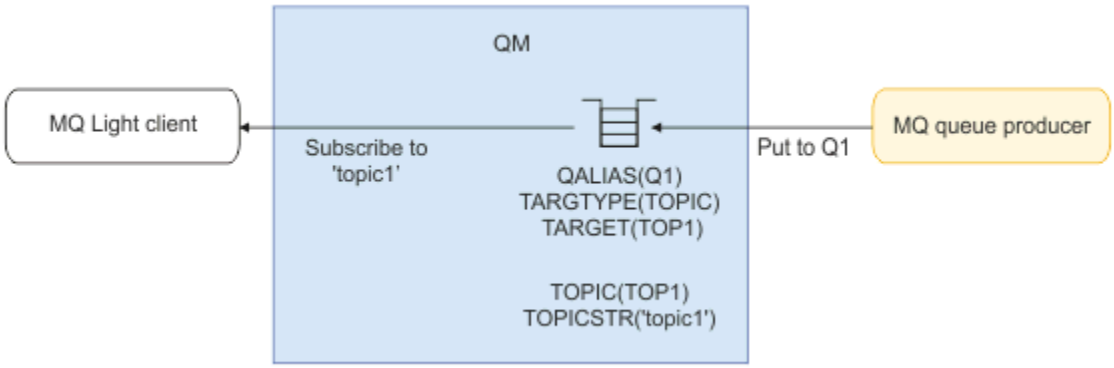
建立及使用 AMQP 通道

保護 AMQP 用戶端安全

ALW 使用佇列別名從 IBM MQ 應用程式接收訊息的 AMQP 用戶端

AMQP 用戶端可以訂閱主題，並接收 IBM MQ 應用程式放入別名佇列的訊息。 如果您想要 AMQP 1.0 應用程式從配置為將訊息放置在佇列上的現有 IBM MQ 應用程式接收訊息，您必須在佇列管理程式上定義佇列別名 (QALIAS)。

佇列別名必須與 IBM MQ 應用程式開啟以放置的佇列同名。 佇列別名必須指定 TOPIC 的基本類型，以及具有主題字串且符合 AMQP 應用程式所訂閱主題型樣的 IBM MQ 主題物件的基本物件。



相關工作

建立及使用 AMQP 通道

保護 AMQP 用戶端安全

ALW AMQP 用戶端向應用程式伺服器提交要求並耗用來自應用程式伺服器的回應

AMQP 用戶端可以向在應用程式伺服器中執行的訊息驅動 Bean 提交要求，並耗用來自回覆主題的回應。 IBM MQ 支援 AMQP 1.0 應用程式在 IBM MQ 發佈的訊息中設定回覆目的地主題。 在已設定 reply-to 屬性的情況下發佈 AMQP 訊息時，會將 reply-to 欄位的值設為 JMS 內容，以供 JMS 消費者接收。 此設定可讓 JMS 消費者從訊息讀取回覆主題，並將回應訊息傳回 AMQP 用戶端。

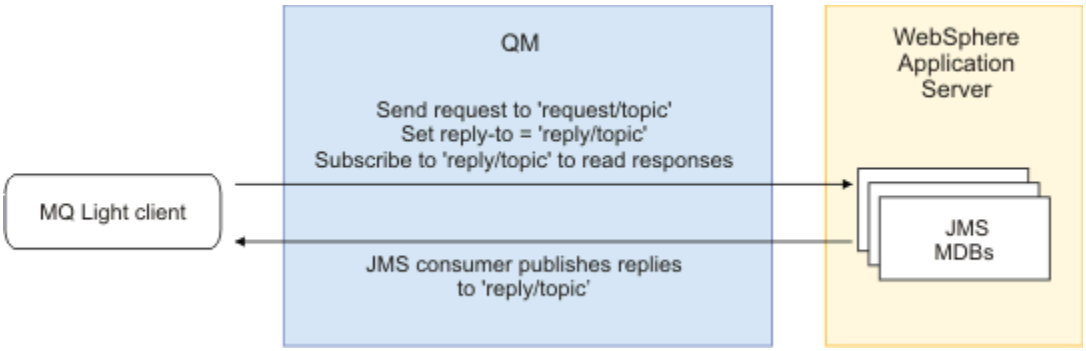
JMS 內容是 **JMSReplyTo**。 AMQP 回覆目的地字串必須是下列其中一種類型:

- 主題字串。 例如: `'reply/topic'`

- AMQP 位址 URL，格式為 `amqp://host:port/[topic-string]`。 例如， `amqp://localhost:5672/reply/topic`

如果您指定 AMQP 位址 URL 作為回覆目的地欄位，則在設定 **JMSReplyTo** 內容之前，會移除 URL 結尾的 topic-string 以外的所有項目。

如需從 AMQP 回覆位址到 **JMSReplyTo** 內容之對映的相關資訊，請參閱 第 575 頁的『將 AMQP 欄位對映至 IBM MQ 欄位 (送入訊息)』



相關工作

建立及使用 AMQP 通道

保護 AMQP 用戶端安全

ALW MQ Light 與 Apache Qpid JMS 應用程式之間的交互作業能力

MQ Light 和 Apache Qpid JMS 應用程式以類似方式運作，當訂閱主題時，請建立遵循相同命名慣例的 IBM MQ 訂閱。

專用、非共用訂閱

應用程式所建立的 IBM MQ 訂閱名稱是 `:private:<clientid>:<topicstring>`。

使用不同用戶端 ID 的應用程式無法存取其他應用程式所建立的訂閱，因為訂閱名稱會自動產生並包含 AMQP 用戶端 ID。

Apache Qpid JMS 和 MQ Light 應用程式都使用專用訂閱的這個命名慣例。

廣域共用訂閱

AMQP 用戶端所建立的廣域共用 IBM MQ 訂閱名稱為 `:share:<sharename>:<topicstring>`。

如果數個具有不同 AMQP 用戶端 ID 的應用程式指定相同的共用名稱及主題字串，則它們會共用單一訂閱，並且可以一起處理該訂閱的訊息。 如果您想要調整從訂閱中排除訊息的工作者應用程式數目，則可以使用此型樣。

Apache Qpid JMS 和 MQ Light 應用程式都會對廣域共用訂閱使用這個命名慣例。 如果是 Apache Qpid JMS，這需要 JMS 連線沒有指定用戶端 ID。

Apache Qpid JMS 程式庫會自動產生 AMQP 用戶端 ID，但此用戶端 ID 不會用於 IBM MQ 訂閱命名目的。

註: 廣域共用訂閱仍以個別佇列管理程式為範圍。

專用共用訂閱

AMQP 用戶端所建立之專用共用 IBM MQ 訂閱的名稱為 `:privateshare:<clientid>:<sharename>:<topicstring>`。

如果來自單一 Apache Qpid JMS 應用程式的數個執行緒使用相同的共用名稱和主題字串，且已在 JMS 連線上配置用戶端 ID，則這些執行緒會共用相同的 IBM MQ 訂閱物件。

不過，其他 Apache Qpid JMS 連線無法共用訂閱，因為它們必須使用不同的用戶端 ID。

MQ Light 用戶端不支援專用共用訂閱的概念，且無法耗用 Apache Qpid JMS 應用程式所建立之專用共用訂閱的訊息。

IBM MQ JMS 訂閱

IBM MQ JMS 訂閱使用與 AMQP 通道不同的命名方法。 MQ Light 或 Apache Qpid JMS 應用程式無法與 IBM MQ JMS 應用程式共用訂閱。

相關概念

開發 AMQP 用戶端應用程式

AMQP API 的 IBM MQ 支援可讓 IBM MQ 管理者建立 AMQP 通道。 啟動時，此通道會定義埠號，以接受來自 AMQP 用戶端應用程式的連線。

ALW IBM MQ AMQP 接聽器控制項內容

若要在多執行緒應用程式中取得更好的效能，您可以在 AMQP 內容檔中配置內容，以調整 AMQP 服務應該使用的工作程式執行緒數目。

您可以在下列內容檔中配置 AMQP 接聽器服務內容:

- Windows 在 Windows 系統上: `amqp_win.properties` 。
- Linux AIX 在 AIX and Linux 系統上: `amqp_unix.properties` 。

您可以配置的內容如下:

表 *105: AMQP* 接聽器服務內容

內容	說明
com.ibm.mq.MQXR.Workers	AMQP 接聽器服務建立的伺服器工作程式執行緒數目。 如果未指定此值，則預設為等於系統上的邏輯處理器數目。
MQIBindType	AMQP 服務的連結類型: FASTPATH、 SHARED 或 **隔離**。 預設值為 FASTPATH。

AMQP 接聽器服務會在多個工作程式執行緒之間平衡用戶端連線工作量。 可以使用 **com.ibm.mq.MQXR.Workers** 內容來指定 AMQP 服務應該使用的工作程式執行緒數。

IBM MQ 佇列管理程式管理者可以調整工作程式執行緒數目，以在多執行緒應用程式中取得更好的效能。 一般而言，當工作程式執行緒數目符合系統上邏輯處理器數目時，會達到最佳效能。 不過，對於某些機器配置和用戶端負載性質，不一定都是如此，因此可能需要調整元素來尋找工作程式執行緒數目的最佳值。

在調整之前，請確定您完全瞭解用戶端應用程式及其工作量的本質。 測量具有不同執行緒計數及評比之應用程式的效能，應該有助於判斷工作程式執行緒數目的最佳值。

註: MQ Appliance 這些內容不適用於 IBM MQ Appliance。 在 IBM MQ Appliance 上使用預設值。

使用 IBM MQ 開發 REST 應用程式

您可以開發 REST 應用程式來傳送及接收訊息。 視平台和功能而定， IBM MQ 支援不同的 REST API。

下列選項是 IBM MQ 支援的選項，您可以從中選擇將訊息傳送至 IBM MQ，以及從中接收訊息:

- IBM MQ messaging REST API
- IBM z/OS Connect EE
- IBM Integration Bus
- DataPower

IBM MQ messaging REST API

您可以使用 messaging REST API 來傳送、接收及瀏覽純文字格式的 IBM MQ 訊息。 依預設會啟用 messaging REST API 。

支援許多不同的 HTTP 標頭，可用來設定一般訊息內容。

messaging REST API 與 IBM MQ 安全完全整合。 若要使用 messaging REST API，使用者必須向 mqweb 伺服器進行鑑別，並且必須是 MQWebUser 角色的成員。

如需進一步資訊，請參閱第 590 頁的『使用 REST API 進行傳訊』。 另請參閱 指導教學: 在 IBM Developer 上開始使用 IBM MQ 傳訊 REST API，其中包括使用傳訊 REST API 的 Go 和 Node.js 範例。

IBM z/OS Connect EE

IBM z/OS Connect EE 是一項 z/OS 產品，可讓您在現有的 z/OS 資產 (例如 CICS 或 IMS 交易，以及 IBM MQ 佇列和主題) 上建置 REST API。 現有 z/OS 資產對使用者隱藏。 這可讓您 REST 啟用資產，而不變更它們或任何使用它們的現有應用程式。

IBM z/OS Connect EE 提供自動資料轉換，以在 REST API 所使用的 JSON 資料與許多大型主機應用程式所預期的較傳統語言結構 (例如 COBOL) 之間進行轉換。

Eclipse 型 IBM z/OS Connect EE API 工具箱可用來建置綜合性的 RESTful API，利用查詢參數和 URL 路徑區段，並在流經 IBM z/OS Connect EE 執行時期時操作 JSON 格式。

IBM z/OS Connect EE 可用來透過 IBM MQ 服務提供者，將 IBM MQ 佇列和主題公開成 RESTful API。 支援兩種不同的服務類型:

- 單向服務: 這些提供 REST API，容許對佇列或主題執行單一 IBM MQ 作業。 根據確切的配置， HTTP 要求可能導致訊息傳送至佇列或發佈至主題; 或 HTTP 要求可能導致從佇列破壞性接收訊息
- 雙向服務: 這些在後端要求/回應樣式應用程式所使用的一對佇列頂端提供 REST API。 呼叫者向雙向服務發出 HTTP 要求。 HTTP 要求有效負載會從 JSON 轉換成傳統語言結構，並放在要求佇列中，由後端應用程式處理它，以及放在回應佇列中的回應。 此回應由服務擷取，從傳統語言結構轉換為 JSON，並傳回給呼叫者作為 POST 回應內文。

如需 IBM z/OS Connect EE 的相關資訊，請參閱 z/OS Connect EE。

如需 IBM MQ 服務提供者的相關資訊，請參閱 使用 IBM MQ 服務提供者。

IBM Integration Bus

IBM Integration Bus 是 IBM 的先進整合技術，可用來將應用程式及系統連接在一起，而不論它們支援何種訊息格式及通訊協定。

IBM Integration Bus 一律支援 IBM MQ，並提供 *HTTPInput* 及 *HTTPRequest* 節點，這些節點可用來在 IBM MQ 及許多其他系統 (例如資料庫) 上建構 RESTful 介面。

IBM Integration Bus 可以用來執行比在 IBM MQ 上提供簡式 REST 介面更多的動作。 其功能可用來提供進階有效負載操作、有效負載強化及許多其他加強功能，作為 REST API 的一部分。

如需進一步資訊，請參閱 技術範例，它會在預期 XML 有效負載的 IBM MQ 應用程式之上公開 JSON over REST 介面。

DataPower

DataPower 閘道是單一多通道閘道，可協助提供對各種系統 (包括 IBM MQ) 的安全、控制、整合及最佳化存取。 它同時出現在硬體和虛擬表單係數中。

DataPower 提供的其中一項服務是多重通訊協定閘道，它可以在一個通訊協定中取得輸入，並在不同的通訊協定中產生輸出。 特別是 DataPower 可以配置為接受 HTTP(S) 資料，並透過用戶端連線將它遞送至 IBM MQ，這可用來在 IBM MQ 上建置 REST 介面。 其他 DataPower 服務 (例如轉換) 也可以用來加強 REST 介面。

如需進一步資訊，請參閱 多重通訊協定閘道。

使用 REST API 進行傳訊

您可以使用 messaging REST API 來執行簡式點對點和發佈傳訊。 您可以將訊息發佈至主題、將訊息傳送至佇列、瀏覽佇列上的訊息，以及破壞性地從佇列取得訊息。 資訊會以純文字格式傳送至 messaging REST API，並從其中接收。

開始之前

註:

- 依預設會啟用 messaging REST API。 您可以停用 messaging REST API，以防止所有傳訊。 如需啟用或停用 messaging REST API 的相關資訊，請參閱 配置 messaging REST API。
- messaging REST API 與 IBM MQ 安全整合。 若要使用 messaging REST API，使用者必須向 mqweb 伺服器進行鑑別，並且必須是 MQWebUser 角色的成員。 使用者也必須獲得授權，才能存取指定的佇列或主題。 如需 REST API 安全的相關資訊，請參閱 IBM MQ Console 和 REST API 安全。
- 如果您將 Advanced Message Security (AMS) 與 messaging REST API 搭配使用，請注意所有訊息都是使用 mqweb 伺服器的環境定義來加密，而不是使用公佈訊息之使用者的環境定義來加密。
- 接收或瀏覽訊息時，只支援 IBM MQ `MQSTR` 或 `JMS TextMessage` 格式化訊息。 隨後，會在同步點下破壞性地接收所有訊息，並將任何未處理的訊息留在佇列上。 IBM MQ 佇列可以配置成將這些有害訊息移至替代目的地。 如需進一步資訊，請參閱第 199 頁的『在 IBM MQ classes for JMS 中處理有害訊息』。
- messaging REST API 不會提供您僅一次遞送具有交易式支援的訊息。 如果在用戶端收到 HTTP 回應之前發出 HTTP POST 且連線失敗，則用戶端無法立即判斷訊息是否已傳送至指定的佇列，或是否已發佈至指定的主題。 如果在用戶端收到 HTTP 回應之前發出 HTTP DELETE，且連線失敗，則訊息可能已從佇列破壞性取得並遺失，因為無法回復破壞性取得。
- V 9.3.0 從 IBM MQ 9.3.0 開始， HTTP POST 作業不再移除送入字串中的新行。 使用舊版的 REST 應用程式不應在使用 REST API 傳送或發佈的訊息中使用換行，因為它們將會遺失。

程序

- 第 590 頁的『「messaging REST API」入門』
- 第 593 頁的『使用 messaging REST API』
- REST API 錯誤處理
- REST API 探索
- REST API 國家語言支援

相關參考

傳訊 REST API 參照

相關資訊

指導教學: 開始使用 IBM MQ 傳訊 REST API

「messaging REST API」入門

快速開始使用 messaging REST API，並使用 cURL 來試用一些範例指令。

開始之前

為了讓您開始使用 messaging REST API，此作業中的範例具有下列需求:

- 這些範例使用 cURL 來傳送 REST 要求，以從佇列放置及取得訊息。 因此，若要完成此作業，您需要在系統上安裝 cURL。
- 範例使用佇列管理程式 `QM1`。 請建立同名的佇列管理程式，或替換系統上現有的佇列管理程式。 佇列管理程式必須與 mqweb 伺服器位於同一部機器上。
- 若要完成此作業，您必須是具有某些特權的使用者，如此才能使用 **dspmqweb** 指令:
  - z/OS 在 z/OS 上，您必須有權執行 **dspmqweb** 指令，並且具有 `mqwebuser.xml` 檔的寫入權。

– Multi 在所有其他作業系統上，必須是 特許使用者。

IBM i 在 IBM i 上，指令應該在 QSHELL 中執行。

程序

1. 確定已針對 messaging REST API 配置 mqweb 伺服器:
   - 確保您已配置 mqweb 伺服器以供 administrative REST API、 administrative REST API for MFT、 messaging REST API 或 IBM MQ Console 使用。如需使用基本登錄來配置 mqweb 伺服器的相關資訊，請參閱 mqweb 伺服器的基本配置。
   - 如果已配置 mqweb 伺服器，請確定您已在 mqweb 伺服器的基本配置的步驟 5 中新增適當的使用者來啟用傳訊。
     – 在 mqweb 伺服器配置中，如果 **mqRestMessagingAdoptWebUserContext** 設為 `true` ，則 messaging REST API 的使用者必須是 `MQWebUser` 角色的成員。 `MQWebAdmin` 和 `MQWebAdminRO` 角色不適用於 messaging REST API。 使用者還必須獲得授權，才能存取用於透過 OAM 或 RACF 進行傳訊的佇列及主題。
     – 如果在 mqweb 伺服器配置中將 **mqRestMessagingAdoptWebUserContext** 設為 `false` ，則用來啟動 mqweb 伺服器的使用者 ID 必須獲得授權，才能透過 OAM 或 RACF 存取用於傳訊的佇列。
2. z/OS
   在 z/OS 上，設定 WLP_USER_DIR 環境變數，以便您可以使用 **dspmqweb** 指令。 透過輸入下列指令，將變數設定為指向您的 mqweb 伺服器配置:

   ```
   export WLP_USER_DIR=WLP_user_directory
   ```

   , 其中 *WLP_user_directory* 是傳遞至 `crtmqweb` 的目錄名稱。 例如:

   ```
   export WLP_USER_DIR=/var/mqm/web/installation1
   ```

   如需相關資訊，請參閱建立 mqweb 伺服器。
3. 輸入下列指令來決定 REST API URL:

   ```
   dspmqweb status
   ```

   下列步驟中的範例假設您的 REST API URL 是預設 URL `https://localhost:9443/ibmmq/rest/v2/`。 如果您的 URL 不同於預設值，請在下列步驟中使用您的 URL 替換預設值。
4. 在佇列管理程式 `QM1` 上建立佇列 `MSGQ`。 此佇列用於傳訊。 使用下列其中一種方法:
   - 在 administrative REST API 的 `mqsc` 資源上使用 POST 要求，並以 `mqadmin` 使用者身分進行鑑別:

     ```
     curl -k https://localhost:9443/ibmmq/rest/v2/admin/action/qmgr/QM1/mqsc -X POST -u
     mqadmin:mqadmin -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: application/json" --
     data "{\"type\": \"runCommandJSON\",\"command\": \"define\", \"qualifier\":
     \"qlocal\",\"name\": \"MSGQ\"}"
     ```

   - 使用 MQSC 指令:

     z/OS 在 z/OS 上，請使用 2CR 來源，而非 **runmqsc** 指令。 如需相關資訊，請參閱 您可以從中在 IBM MQ for z/OS 上發出 MQSC 及 PCF 指令的來源。

     a. 透過輸入下列指令，啟動佇列管理程式的 **runmqsc** :

     ```
     runmqsc QM1
     ```

     b. 使用 **DEFINE QLOCAL** MQSC 指令來建立佇列:

     ```
     DEFINE QLOCAL(MSGQ)
     ```

c. 輸入下列指令，以結束 **runmqsc**：

```
end
```

5. 將權限授與您在 mqweb 伺服器的基本配置 的步驟 5 中新增至 mqwebuser.xml 的使用者，以存取佇列 MSGQ。 替換使用 myuser 的使用者:

- z/OS 在 z/OS 上:

a. 授與使用者對佇列的存取權:

```
RDEFINE MQQUEUE hlq.MSGQ UACC(NONE)
PERMIT hlq.MSGQ CLASS(MQQUEUE) ID(MYUSER) ACCESS(UPDATE)
```

b. 授與 mqweb 啟動型作業使用者 ID 存取權，以設定佇列上的所有環境定義:

```
RDEFINE MQADMIN hlq.CONTEXT.MSGQ UACC(NONE)
PERMIT hlq.CONTEXT.MSGQ CLASS(MQADMIN) ID(mqwebStartedTaskID) ACCESS(CONTROL)
```

- Multi 在所有其他作業系統上，如果您的使用者是在 mqm 群組中，則已授與權限。 否則，請輸入下列指令:

a. 透過輸入下列指令，啟動佇列管理程式的 **runmqsc**：

```
runmqsc QM1
```

b. 使用 **SET AUTHREC** MQSC 指令，提供使用者對佇列的瀏覽、查詢、取得及放置權限:

```
SET AUTHREC PROFILE(MSGQ) OBJTYPE(QUEUE) +
PRINCIPAL(myuser) AUTHADD(BROWSE, INQ, GET, PUT)
```

c. 輸入下列指令，以結束 **runmqsc**：

```
end
```

6. 在 message 資源上使用 POST 要求，將內容為 Hello World! 的訊息放置在佇列管理程式 QM1 的佇列 MSGQ 上。 將 mqwebuser.xml 中的使用者 ID 和密碼替換為 myuser 和 mypassword:

使用基本鑑別，並在 cURL REST 要求中設定具有任意值的 ibm-mq-rest-csrf-token HTTP 標頭。 POST、PATCH 及 DELETE 要求需要此額外標頭。

```
curl -k https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/MSGQ/message -X POST
-u myuser:mypassword -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: text/
plain;charset=utf-8" --data "Hello World!"
```

7. 在 message 資源上使用 DELETE 要求，以破壞性方式從佇列 MSGQ 佇列管理程式 QM1 上的佇列 Hello World! 取得訊息。 將 mqwebuser.xml 中的使用者 ID 和密碼替換為 myuser 和 mypassword:

```
curl -k https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/MSGQ/message -X DELETE
-u myuser:mypassword -H "ibm-mq-rest-csrf-token: value"
```

即會傳回訊息 Hello World!。

下一步

- 這些範例使用基本鑑別來保護要求。 您可以改為使用記號型鑑別或用戶端型鑑別。 如需相關資訊，請參閱使用用戶端憑證鑑別與 REST API 及 IBM MQ Console 及 使用記號型鑑別與 REST API。
- 進一步瞭解使用 messaging REST API 及使用查詢參數來建構 URL: 第 593 頁的『使用 messaging REST API』。
- 當您使用 messaging REST API 時，會儲存佇列管理程式的連線，以最佳化效能。 您可以配置儲存區大小上限，以及當儲存區中的所有連線都在使用中時所採取的動作: 配置 messaging REST API。
- 瀏覽可用的 messaging REST API 資源及所有可用的查詢參數的參照資訊: messaging REST API reference。

- 探索 administrative REST API，這是 IBM MQ 管理的 RESTful 介面: 使用 REST API 進行管理。
- 探索 IBM MQ Console(瀏覽器型 GUI): 使用 IBM MQ Console 進行管理。

使用 messaging REST API

當您使用 messaging REST API 時，您可以對 URL 呼叫 HTTP 方法來傳送及接收 IBM MQ 訊息。 HTTP 方法 (例如 POST) 代表要對 URL 所代表的物件執行的動作類型。 動作的進一步相關資訊可以在查詢參數中編碼。 執行動作之結果的相關資訊可能會以 HTTP 回應的內文傳回。

開始之前

在使用 messaging REST API 之前，請考量下列事項:

- 您必須向 mqweb 伺服器進行鑑別，才能使用 messaging REST API。 您可以使用 HTTP 基本鑑別、用戶端憑證鑑別或記號型鑑別進行鑑別。 如需如何使用這些鑑別方法的相關資訊，請參閱 IBM MQ Console 和 REST API 安全。
- REST API 區分大小寫。 例如，如果佇列管理程式稱為 `qmgr1`，則下列 URL 上的 HTTP POST 會導致錯誤。

```
/ibmmq/rest/v2/messaging/qmgr/QMGR1/queue/Q1/message
```

- V 9.3.3 如果您使用「messaging REST API」連接遠端佇列管理程式，則必須使用佇列管理程式連線的唯一名稱，而非佇列管理程式名稱。
- 並非 IBM MQ 物件名稱中可以使用的所有字元都可以直接編碼在 URL 中。 如果要正確編碼這些字元，您必須使用適當的 URL 編碼:
  - 正斜線必須編碼為 %2F。
  - 百分比符號必須編碼為 %25。
  - 句點必須編碼為 %2E。
  - 問號必須編碼為 %3F。
- 當接收或瀏覽訊息時，只支援 IBM MQ `MQSTR` 和 `JMS TextMessage` 格式化訊息。 隨後，會在同步點下破壞性地接收所有訊息，並將任何未處理的訊息留在佇列上。 IBM MQ 佇列可以配置成將這些有害訊息移至替代目的地。 如需進一步資訊，請參閱第 199 頁的『在 IBM MQ classes for JMS 中處理有害訊息』。

關於這項作業

當您使用 REST API 來對 IBM MQ 佇列物件執行傳訊動作時，首先需要建構 URL 來代表該物件。 每一個 URL 都以字首開頭，說明要將要求傳送至哪一個主機名稱和埠。 URL 的其餘部分說明特定物件，或該物件的路徑，稱為資源。

要在資源上執行的傳訊動作會定義 URL 是否需要查詢參數。 它也會定義所使用的 HTTP 方法，以及其他資訊是傳送至 URL，還是從它傳回。 其他資訊可能形成 HTTP 要求的一部分，或作為 HTTP 回應的一部分傳回。

建構 URL 之後，您可以將 HTTP 要求傳送至 IBM MQ。 您可以使用內建在您選擇的程式設計語言中的 HTTP 實作來傳送要求。 您也可以使用指令行工具 (例如 cURL) 或 Web 瀏覽器或 Web 瀏覽器附加程式來傳送要求。

重要: 您至少必須執行步驟 第 593 頁的『1.a』 和 第 593 頁的『1.b』。

程序

1. 建構 URL:

   a) 輸入下列指令，以判定字首 URL :

   ```
   dspmqweb status
   ```

   您要使用的 URL 包括 `/ibmmq/rest/` 詞組。

   b) 將用於傳訊的佇列及相關聯佇列管理程式資源新增至 URL 路徑。

在傳訊參照中，變數區段可以在 URL 中以含括它的大括弧 *{ }*來識別。如需進一步資訊，請參閱 /messaging/qmgr/{qmgrName}/queue/{queueName}/message。

例如，若要與與佇列管理程式 *QM1* 相關聯的佇列 *Q1* 互動，請將 /qmgr 及 /queue 新增至字首 URL，以建立下列 URL:

```
https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/Q1/message
```

提示: V 9.3.3 如果佇列管理程式是遠端佇列管理程式，您必須使用佇列管理程式的唯一名稱來取代佇列管理程式名稱。 必須先配置遠端佇列管理程式，才能與 messaging REST API 搭配使用。 如需相關資訊，請參閱 第 594 頁的『設定遠端佇列管理程式以與 messaging REST API 搭配使用』。

c) 選擇性的: 將選用查詢參數新增至 URL。

新增問號，?， 查詢參數、等號 = 及 URL 的值。

例如，若要等待最多 30 秒，讓下一個訊息變成可用，請建立下列 URL:

```
https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/Q1/message?wait=30000
```

d) 選擇性的: 將進一步選用查詢參數新增至 URL。

Add an ampersand, &, to the URL, and then repeat 步驟 1c.

2. 在 URL 上呼叫相關的 HTTP 方法。 指定任何選用訊息有效負載，並提供適當的安全認證以進行鑑別。 例如:
   - 使用所選程式設計語言的 HTTP/REST 實作。
   - 使用 REST 用戶端瀏覽器附加程式或 cURL 之類的工具。

V 9.3.3 設定遠端佇列管理程式以與 messaging REST API 搭配使用

您可以使用「messaging REST API」來連接遠端佇列管理程式，以進行傳訊。 您必須先設定遠端佇列管理程式配置，才能連接至遠端佇列管理程式。 然後，您可以使用配置資訊中定義的唯一名稱來連接遠端佇列管理程式。

開始之前

- 確保您已配置 mqweb 伺服器以供 administrative REST API、 administrative REST API for MFT、messaging REST API 或 IBM MQ Console 使用。如需使用基本登錄來配置 mqweb 伺服器的相關資訊，請參閱 mqweb 伺服器的基本配置。
- 如果已配置 mqweb 伺服器，請確定您已在 mqweb 伺服器的基本配置的步驟 5 中新增適當的使用者來啟用傳訊。 messaging REST API 的使用者必須是 `MQWebUser` 角色的成員。 `MQWebAdmin` 和 `MQWebAdminRO` 角色不適用於 messaging REST API。
  - 如果在 mqweb 伺服器配置中將 **`mqRestMessagingAdoptWebUserContext`** 設為 `true`，則必須授權 `MQWebUser` 角色中的使用者存取用於透過 OAM 或 RACF 進行傳訊的佇列及主題。
  - 如果在 mqweb 伺服器配置中， **`mqRestMessagingAdoptWebUserContext`** 設為 `false`，則用來啟動 mqweb 伺服器的使用者 ID 必須獲得授權，才能透過 OAM 或 RACF 存取用於傳訊的佇列及主題。
- 請確定 messaging REST API 已配置成連接至遠端佇列管理程式。 如需相關資訊，請參閱 配置 messaging REST API 的連線模式。

關於這項作業

您可以使用「messaging REST API」來連接遠端佇列管理程式。 遠端佇列管理程式可以是另一個系統上的佇列管理程式、另一個安裝中的佇列管理程式，或與 mqweb 伺服器相同安裝中的佇列管理程式。

若要連接至遠端佇列管理程式，您必須完成下列配置步驟:

- 配置伺服器連線通道和接聽器。
- 將存取佇列管理程式的權限提供給適當的使用者。

- 建立包含佇列管理程式連線資訊的 CCDT 檔案。
- 使用 **setmqweb remote** 指令，將連線資訊新增至 messaging REST API 。

然後，您可以在資源 URL 中提供唯一名稱來取代佇列管理程式名稱，以使用遠端佇列管理程式。

您也可以將遠端佇列管理程式配置成佇列管理程式群組的一部分。 如需相關資訊，請參閱 第 597 頁的『設定佇列管理程式群組以與傳訊 REST API 搭配使用』。

程序

1. 在遠端佇列管理程式上，建立伺服器連線通道以容許遠端連線至佇列管理程式。 您可以在指令行上使用 **DEFINE CHANNEL** MQSC 指令來建立伺服器連線通道。
   例如，若要為佇列管理程式 QM1 建立伺服器連線通道 QM1.SVRCONN ，請輸入下列指令:

   ```
   DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
   ```

   如需 **DEFINE CHANNEL** 及可用選項的相關資訊，請參閱 DEFINE CHANNEL。
2. 請確定適當的使用者已獲授權存取佇列管理程式。 此使用者也必須獲得授權，才能存取您用於傳訊的任何佇列或主題。 使用者在佇列管理程式上需要 connect、 inquire、 alternate user 及 set context 權限。 在 UNIX, Linux, and Windows 上，請在指令行上使用 **setmqaut** 控制指令。 在 z/OS 上，定義 RACF 設定檔，以授與授權使用者對佇列管理程式的存取權。
   例如，在 UNIX, Linux, and Windows 上，若要授權使用者 exampleUser 存取佇列管理程式 QM1，請輸入下列指令:

   ```
   setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +altusr +setall
   ```

   如需哪些使用者需要獲得授權的相關資訊，請參閱 第 599 頁的『決定 messaging REST API 所使用的安全主體』。
3. ALW
   如果遠端佇列管理程式上沒有接聽器，請在指令行上使用 **DEFINE LISTENER** MQSC 指令建立接聽器以接受送入的網路連線。
   例如，若要在埠 1414 上建立遠端佇列管理程式 QM1 的接聽器 REMOTE.LISTENER ，請輸入下列指令:

   ```
   runmqsc QM1
   DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
   end
   ```
4. 透過在指令行上使用 **START LISTENER** MQSC 指令，確保接聽器正在執行中:

   ALW例如，在 AIX, Linux, and Windows 上，若要啟動佇列管理程式 QM1 的接聽器 REMOTE.LISTENER ，請輸入下列指令:

   ```
   runmqsc QM1
   START LISTENER(REMOTE.LISTENER)
   end
   ```

   z/OS例如，在 z/OS 上，若要啟動接聽器，請輸入下列指令:

   ```
   runmqsc QM1
   START LISTENER TRPTYPE(TCP) PORT(1414)
   end
   ```

   必須先啟動通道起始程式位址空間，然後才能在 z/OS 上啟動接聽器。
5. 在管理 messaging REST API 的 mqweb 伺服器執行所在的系統上，建立或更新包含佇列管理程式連線資訊的 JSON CCDT 檔案。

   CCDT 檔案必須包含 name、 clientConnection 及 type 資訊。 您可以選擇性地包括其他資訊，例如 transmissionSecurity 資訊。 如需所有 CCDT 通道屬性定義的相關資訊，請參閱 CCDT 通道屬性定義的完整清單。

下列範例顯示遠端佇列管理程式連線的基本 JSON CCDT 檔案。 它會將通道名稱設為與步驟 第 595 頁的『1』中建立的範例伺服器連線通道相同的名稱。 連線埠設為與接聽器所使用埠相同的值。 連線主機設為遠端佇列管理程式 QM1 執行所在系統的主機名稱。

```
{
  "channel": [{
      "name": "QM1.SVRCONN",
      "clientConnection": {
        "connection": [{
            "host": "example.com",
            "port": 1414
          }],
        "queueManager": "QM1"
      },
      "type": "clientConnection"
    }]
}
```

6. 從執行管理 messaging REST API 之 mqweb 伺服器的安裝架構中，使用 **setmqweb remote** 指令將遠端佇列管理程式資訊新增至 mqweb 伺服器配置。

   您至少必須指定下列參數:

   - **-qmgrName**，您在其中指定佇列管理程式的名稱。
   - **-ccdtURL**，您在其中指定佇列管理程式的 CCDT URL 。
   - **-uniqueName**，您可以在其中指定佇列管理程式的唯一名稱。 唯一名稱用來區分可能具有相同名稱的遠端佇列管理程式，因此不得存在以識別另一個佇列管理程式。

   您可以指定數個其他選項，例如用於遠端佇列管理程式連線的使用者名稱及密碼，或信任儲存庫及金鑰儲存庫的詳細資料。 如需可以使用 **setmqweb remote** 指令指定之參數的完整清單，請參閱 setmqweb remote。

   例如，若要使用範例 CCDT 檔案新增遠端佇列管理程式 QM1，請輸入下列指令:

```
setmqweb remote add -uniqueName "RemoteQM1" -qmgrName "QM1" -ccdtURL "c:\myccdts\ccdt.json"
```

結果

遠端佇列管理程式可以與 messaging REST API 搭配使用，方法是使用資源 URL 中的唯一名稱來取代佇列管理程式名稱。

範例

下列範例設定佇列管理程式 QM1 的遠端佇列管理程式連線。 根據提供給使用者 exampleUser 的授權，獲授權管理佇列管理程式的 IBM MQ Console 。 當使用 **setmqweb remote** 來配置佇列管理程式連線時，會將此使用者的認證提供給 IBM MQ Console 。

1. 在遠端佇列管理程式 QM1 所在的系統上，會建立伺服器連線通道及接聽器。 即會啟動接聽器，並授權使用者 `exampleUser` 連接至佇列管理程式，以及存取用於傳訊的佇列:

```
runmqsc QM1
#Define the server connection channel that will accept connections from the Console
DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
# Define the listener to use for the connection from the Console
DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
# Start the listener
START LISTENER(REMOTE.LISTENER)
end

#Set mq authorization for exampleUser to access the queue manager and a queue for messaging
setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +setall +dsp
setmqaut -m QM1 -t queue -p exampleUser -n EXAMPLEQ +put +get +browse +inq
```

2. 在 mqweb 伺服器執行所在的系統上，會使用下列連線資訊建立 `QM1_ccdt.json` 檔案:

```
{
  "channel": [{
      "name": "QM1.SVRCONN",
      "clientConnection": {
        "connection": [{
```

```
            "host": "example.com",
            "port": 1414
          }],
        "queueManager": "QM1"
      },
      "type": "clientConnection"
    }]
}
```

3. 在 mqweb 伺服器執行所在的系統上，佇列管理程式 QM1 的連線資訊會新增至 mqweb 伺服器。 exampleUser 的認證包含在連線資訊中:

```
setmqweb remote add -uniqueName "MACHINEAQM1" -qmgrName "QM1" -ccdtURL
"c:\myccdts\QM1_ccdt.json" -username "exampleUser" -password "password"
```

4. 「messaging REST API」可以連接至遠端佇列管理程式 QM1，方法是使用佇列管理程式連線的唯一名稱來取代資源 URL 中的佇列管理程式名稱:

```
curl -k https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/MACHINEAQM1/queue/EXAMPLEQ/
message -X POST -u myuser:mypassword -H "ibm-mq-rest-csrf-token: value" -H "Content-Type:
text/plain;charset=utf-8" --data "Hello World!"
```

V 9.3.3 設定佇列管理程式群組以與傳訊 REST API 搭配使用

您可以使用「messaging REST API」來連接至佇列管理程式群組，以進行傳訊。 您必須先設定群組的遠端佇列管理程式配置，才能連接至佇列管理程式群組。 然後，您可以使用配置資訊中定義的唯一名稱來連接至佇列管理程式群組。

開始之前

- 確保您已配置 mqweb 伺服器以供 administrative REST API、 administrative REST API for MFT、 messaging REST API 或 IBM MQ Console 使用。如需使用基本登錄來配置 mqweb 伺服器的相關資訊，請參閱 mqweb 伺服器的基本配置。
- 如果已配置 mqweb 伺服器，請確定您已在 mqweb 伺服器的基本配置的步驟 5 中新增適當的使用者來啟用傳訊。 messaging REST API 的使用者必須是 MQWebUser 角色的成員。 MQWebAdmin 和 MQWebAdminRO 角色不適用於 messaging REST API。
  - 如果在 mqweb 伺服器配置中將 **mqRestMessagingAdoptWebUserContext** 設為 true，則必須授權 MQWebUser 角色中的使用者存取用於傳訊的佇列及主題。 您可以透過 OAM 或 RACF 來授權這些使用者。
  - 如果 **mqRestMessagingAdoptWebUserContext** 在 mqweb 伺服器配置中設為 false，則啟動 mqweb 伺服器的使用者 ID 必須獲得授權，才能存取用於傳訊的佇列及主題。 您可以透過 OAM 或 RACF 來授權此使用者。
- 請確定 messaging REST API 已配置成連接至遠端佇列管理程式。 如需相關資訊，請參閱 配置 messaging REST API 的連線模式

關於這項作業

佇列管理程式群組可讓您將應用程式連接至群組內的任何佇列管理程式。 群組定義為用戶端通道定義表 (CCDT) 中的一組連線。 當您使用 MQCONN 或 MQCONNX 呼叫時，您可以在佇列管理程式名稱前面加上星號來參照群組。 使用 messaging REST API，您可以使用與佇列管理程式群組相關聯的唯一名稱來參照群組。 唯一名稱包含在資源 URL 中，以取代佇列管理程式名稱。 如需佇列管理程式群組的相關資訊，請參閱 第 773 頁的『CCDT 中的佇列管理程式群組』。

您也可以個別配置遠端佇列管理程式。 如需相關資訊，請參閱 第 594 頁的『設定遠端佇列管理程式以與 messaging REST API 搭配使用』。

程序

1. 在群組中的每一個遠端佇列管理程式上，建立伺服器連線通道以容許與佇列管理程式的遠端連線。 您可以在指令行上使用 **DEFINE CHANNEL** MQSC 指令來建立伺服器連線通道。

例如，若要為佇列管理程式 QM1 建立伺服器連線通道 QM1.SVRCONN，請輸入下列指令:

```
DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
```

如需 **DEFINE CHANNEL** 及可用選項的相關資訊，請參閱 DEFINE CHANNEL。

2. 在群組中的每一個遠端佇列管理程式上，確定適當的使用者已獲授權存取佇列管理程式。 此使用者也必須獲得授權，才能存取您用於傳訊的任何佇列或主題。 使用者在佇列管理程式上需要 connect、inquire、 alternate user 及 set context 權限。 在 UNIX, Linux, and Windows 上，請在指令行上使用 **setmqaut** 控制指令。 在 z/OS 上，定義 RACF 設定檔，以授與授權使用者對佇列管理程式的存取權。
例如，在 UNIX, Linux, and Windows 上，輸入下列指令以授權使用者 exampleUser 存取佇列管理程式 QM1:

```
setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +altusr +setall
```

如需哪些使用者需要獲得授權的相關資訊，請參閱 第 599 頁的『決定 messaging REST API 所使用的安全主體』。

3. ALW
如果群組中每一個遠端佇列管理程式上都沒有接聽器，請建立接聽器以接受送入的網路連線。 您可以在指令行上使用 **DEFINE LISTENER** MQSC 指令來建立接聽器。
例如，若要在埠 1414 上建立遠端佇列管理程式 QM1 的接聽器 REMOTE.LISTENER，請輸入下列指令:

```
runmqsc QM1
DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
end
```

4. 在群組中的每一個遠端佇列管理程式上，確保接聽器正在執行，方法是在指令行上使用 **START LISTENER** MQSC 指令。

ALW 例如，在 AIX, Linux, and Windows 上，若要啟動佇列管理程式 QM1 的接聽器 REMOTE.LISTENER，請輸入下列指令:

```
runmqsc QM1
START LISTENER(REMOTE.LISTENER)
end
```

z/OS 例如，在 z/OS 上，若要啟動接聽器，請輸入下列指令:

```
runmqsc QM1
START LISTENER TRPTYPE(TCP) PORT(1414)
end
```

必須先啟動通道起始程式位址空間，然後才能在 z/OS 上啟動接聽器。

5. 在管理 messaging REST API 的 mqweb 伺服器執行所在的系統上，建立 JSON CCDT 檔案。 此 JSON 檔案包含群組中每一個佇列管理程式的連線資訊。

CCDT 檔案必須包含每一個佇列管理程式連線的 name、 clientConnection 及 type 資訊。 您可以選擇性地包括其他資訊，例如 transmissionSecurity 資訊。 如需所有 CCDT 通道屬性定義的相關資訊，請參閱 CCDT 通道屬性定義的完整清單。

下列範例顯示兩個佇列管理程式連線的基本 JSON CCDT 檔案。 第一個連線是針對佇列管理程式 QM1。它具有伺服器連線通道 QM1.SVRCONN(埠 1414 上的接聽器)，並在主機 QM1.example.com 上執行。第二個連線適用於佇列管理程式 QM2。 它具有伺服器連線通道 QM2.SVRCONN(埠 1415 上的接聽器)，並在主機 QM2.example.com 上執行。 不過，因為連線是佇列管理程式群組 QMGRP 的一部分，所以這兩個連線的 **queueManager** 欄位都會設為佇列管理程式群組的名稱。

```
{
  "channel": [{
      "name": "QM1.SVRCONN",
      "clientConnection": {
        "connection": [{
            "host": "QM1.example.com",
            "port": 1414
          }],
```

```
        "queueManager": "QMGRP"
      },
      "type": "clientConnection"
   }],
   "channel": [{
      "name": "QM2.SVRCONN",
      "clientConnection": {
        "connection": [{
            "host": "QM2.example.com",
            "port": 1415
          }],
        "queueManager": "QMGRP"
      },
      "type": "clientConnection"
    }]
}
```

6. 從執行管理 messaging REST API 之 mqweb 伺服器的安裝中，使用 **setmqweb remote** 指令將佇列管理程式群組新增至 mqweb 伺服器配置。

   您至少必須指定下列參數:

   - **-qmgrName**，您在其中指定佇列管理程式群組的群組名稱。
   - **-ccdtURL**，您在其中指定佇列管理程式的 CCDT URL 。
   - **-uniqueName**，您可以在其中指定唯一名稱來識別佇列管理程式群組。
   - **-group**，將佇列管理程式資訊設為群組的資訊。

   您可以指定數個其他選項，例如用於連線的使用者名稱及密碼，或信任儲存庫及金鑰儲存庫的詳細資料。 如需可以使用 **setmqweb remote** 指令指定之參數的完整清單，請參閱 setmqweb remote。

   例如，若要使用範例 CCDT 檔案來新增佇列管理程式群組 QMGRP ，請輸入下列指令:

```
setmqweb remote add  -uniqueName "MyQMGRP" -qmgrName "QMGRP" -ccdtURL
"c:\myccdts\group_ccdt.json" -group
```

結果

遠端佇列管理程式群組可以與 messaging REST API 搭配使用，方法是使用資源 URL 中的唯一名稱。 會選取群組中的佇列管理程式來完成要求，並在回應標頭 `ibm-mq-resolved-qmgr` 中傳回哪個佇列管理程式已完成要求的相關資訊。

V 9.3.3 決定 messaging REST API 所使用的安全主體

當您使用 messaging REST API 時，必須授權適當的使用者存取您要連接以進行傳訊的佇列管理程式、佇列及主題。 需要授權的使用者取決於 mqweb 伺服器的配置方式，以及您是否搭配使用遠端佇列管理程式與 messaging REST API。

依預設，用來授權存取佇列管理程式的安全主體是啟動執行 messaging REST API 的 mqweb 伺服器的使用者。 用來授權存取佇列及主題的安全主體是登入 messaging REST API 的使用者。 不過，您可以配置 mqweb 伺服器或遠端佇列管理程式連線，以使用不同的安全主體。

判斷用來連接佇列管理程式的安全主體

對於本端佇列管理程式連線，用來連接至佇列管理程式的安全主體是啟動執行 messaging REST API 的 mqweb 伺服器的使用者。 對於遠端佇列管理程式連線， messaging REST API 會依優先順序使用下列安全主體來授權存取佇列管理程式。 也就是說，如果在遠端佇列管理程式配置內以多種方式指定使用者，則會使用清單中的第一個來進行授權。

1. 安全主體是來自安全結束程式的採用使用者環境定義。
2. 安全主體是伺服器連線通道上用來連接遠端佇列管理程式的 CHLAUTH 規則中採用的使用者環境定義。
3. 安全主體是 messaging REST API 的遠端佇列管理程式配置中所包含的使用者 ID。 當您使用 **setmqweb remote** 指令新增佇列管理程式時，此使用者 ID 會選擇性地包含在佇列管理程式連線資訊中。
4. 安全主體是啟動執行 messaging REST API 的 mqweb 伺服器的使用者。

如需設定遠端佇列管理程式以與 messaging REST API 搭配使用的相關資訊，請參閱 第 594 頁的『設定遠端佇列管理程式以與 messaging REST API 搭配使用』。

決定用來連接佇列及主題的安全主體

您可以在 mqweb 伺服器配置中設定內容，以判定當您使用 messaging REST API 時，使用哪個安全主體來授權佇列及主題的連線。 此內容是 **mqRestMessagingAdoptWebUserContext** 內容。 您可以使用 **dspmqweb properties** 指令來檢視此內容的設定。

- 如果 **mqRestMessagingAdoptWebUserContext** 設為 true，則 messaging REST API 會使用登入 messaging REST API 之使用者的使用者 ID 進行授權。 因此，存在於 mqweb 伺服器配置中以與 messaging REST API 搭配使用的使用者 ID 或使用者 ID，是必須授權才能存取佇列及主題的安全主體。
- 如果 **mqRestMessagingAdoptWebUserContext** 設為 false，則 messaging REST API 會使用啟動管理 messaging REST API 之 mqweb 伺服器的使用者 ID 進行授權。 因此，與啟動管理 messaging REST API 之 mqweb 伺服器的使用者 ID 相同的使用者 ID 必須獲得授權，才能存取佇列及主題。

  如果您的佇列和主題位於遠端佇列管理程式上，則用於授權的安全主體可能由佇列管理程式配置中的設定來決定。 可以按優先順序使用下列安全主體:

  1. 安全主體是來自安全結束程式的採用使用者環境定義。
  2. 安全主體是伺服器連線通道上用來連接遠端佇列管理程式的 CHLAUTH 規則中採用的使用者環境定義。 例如，您可以在伺服器連線通道上配置 CHLAUTH 規則，以使用 MCAUSER 參數。 然後，所有連線都會對映至獲授權使用佇列管理程式的使用者 ID。
  3. 安全主體是來自佇列管理程式 AUTHINFO 的採用使用者環境定義。 如果佇列管理程式的 CONNAUTH 屬性所參照的 AUTHINFO 物件配置為使用 **ADOPTCTX(yes)**，則用來授權佇列管理程式連線的安全主體也會用來授權佇列及主題。 例如，此安全主體可能是遠端佇列管理程式連線資訊中包含的使用者 ID，作為 **setmqweb remote** 指令的一部分。

相關資訊

CHLAUTH

CONNAUTH

dspmqweb 內容

使用 IBM MQ 開發 MQI 應用程式

IBM MQ 提供 C、Visual Basic、COBOL、Assembler、RPG、 pTAL 及 PL/I 的支援。 這些程序化語言使用訊息佇列介面 (MQI) 來存取訊息佇列作業服務。

如需如何以您選擇的語言撰寫應用程式的詳細資訊，請參閱子主題。

如需程序化語言的呼叫介面概觀，請參閱 呼叫說明。 本主題包含 MQI 呼叫的清單，且每一個呼叫都顯示如何對每一種語言的呼叫進行編碼。

IBM MQ 提供資料定義檔來協助您撰寫應用程式。 如需完整說明，請參閱 第 601 頁的『IBM MQ 資料定義檔』。

為了協助您選擇要撰寫程式的程序化語言，請考量程式將處理的訊息長度上限。 如果您的程式只會處理已知長度上限的訊息，您可以用任何支援的語言來撰寫它們的程式碼。 如果您不知道程式將必須處理的訊息長度上限，則您選擇的語言將視您是撰寫 CICS、 IMS 還是批次應用程式而定:

IMS 和批次
: 以 C、PL/I 或組譯語言撰寫程式，以使用這些語言提供的機能來取得及釋放任意數量的記憶體。 或者，您可以在 COBOL 中撰寫程式的程式碼，但使用組譯語言、PL/I 或 C 子常式來取得及釋放儲存體。

CICS
: 以 CICS 支援的任何語言撰寫程式的程式碼。 必要的話， EXEC CICS 介面會提供用於管理記憶體的呼叫。

相關概念

第 14 頁的『物件導向應用程式』

IBM MQ 提供 JMS、 Java、C++ 及 .NET 的支援。 這些語言及架構使用「 IBM MQ 物件模型」，其提供的類別提供與 IBM MQ 呼叫及結構相同的功能。

技術概觀
第 6 頁的『應用程式開發概念』
您可以使用選擇的程序化或物件導向語言來撰寫 IBM MQ 應用程式。 在開始設計及撰寫 IBM MQ 應用程式之前，請先熟悉基本 IBM MQ 概念。

相關參考
開發應用程式參照

IBM MQ 資料定義檔

IBM MQ 提供資料定義檔來協助您撰寫應用程式。

資料定義檔也稱為:

語言	資料定義
C	併入檔案或標頭檔
Visual Basic	模組檔案 (僅限 32 位元版本)
COBOL	複製檔案
Assembler	巨集
PL/I	包括檔案

IBM MQ COPY、標頭、併入和模組檔案中說明了協助您寫入通道結束程式的資料定義檔。

第 784 頁的『使用者結束程式、API 結束程式及 IBM MQ 可安裝服務』中說明了協助您撰寫可安裝服務結束程式的資料定義檔。

如需 C++ 上支援的資料定義檔，請參閱 使用 C++。

IBM i

如需 RPG 支援的資料定義檔，請參閱 IBM i Application Programming Reference (ILE/RPG)。

資料定義檔的名稱具有字首 CMQ ，以及由程式設計語言決定的字尾:

後綴	語言
a	組譯語言
b	Visual Basic
c	C
l	COBOL (沒有起始設定值)
p	PL/I
v	COBOL (已設定預設值)

安裝程式庫

名稱 **thlqual** 是 z/OS 上安裝程式庫的高階限定元。

本主題介紹 IBM MQ 資料定義檔，位於下列標題下:

- 第 602 頁的『C 語言併入檔』
- 第 602 頁的『Visual Basic 模組檔案』
- 第 602 頁的『COBOL 複製檔案』
- z/OS 第 603 頁的『System/390 組譯語言巨集』
- z/OS 第 603 頁的『PL/I 併入檔』

C 語言併入檔

IBM MQ C 併入檔列在 C 標頭檔中。 它們安裝在下列目錄或檔案庫中:

平台	安裝目錄或程式庫
IBM i IBM i	QMQM/H
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*/inc/
Windows Windows	*MQ_INSTALLATION_PATH*\Tools\c\include
z/OS z/OS	**thlqual**.SCSQC370

其中 *MQ_INSTALLATION_PATH* 代表 IBM MQ 安裝所在的高階目錄。

註: 對於 AIX and Linux，併入檔以符號方式鏈結至 `/usr/include`。

如需目錄結構的相關資訊，請參閱 規劃檔案系統支援。

Visual Basic 模組檔案

IBM MQ for Windows 提供四個 Visual Basic 模組檔案。

它們列在 Visual Basic 模組檔案 中，並安裝在

```
MQ_INSTALLATION_PATH\Tools\Samples\VB\Include
```

COBOL 複製檔案

對於 COBOL， IBM MQ 提供包含具名常數的個別副本檔案，以及每一個結構的兩個副本檔案。

每一個結構都有兩個複製檔案，因為每一個都提供有及沒有起始值:

- 在 COBOL 程式的 WORKING-STORAGE SECTION 中，使用將結構欄位起始設定為預設值的檔案。 這些結構定義在副本檔中，其名稱以字母 V (值) 為字尾。
- 在 COBOL 程式的 LINKAGE SECTION 中，使用不含起始值的結構。 這些結構定義在名稱以字母 L (鏈結) 為字尾的副本檔中。

IBM i 針對使用 MQI 原型化呼叫的 ILE COBOL 程式，提供包含 IBM i 資料及介面定義的副本檔案。檔案存在於 QMQM/QCBLLESRC 中，其成員名稱具有字尾 L (適用於沒有起始值的結構) 或字尾 V (適用於具有起始值的結構)。

IBM MQ COBOL 副本檔案列在 COBOL COPY 檔案中。 它們安裝在下列目錄中:

平台	安裝目錄或程式庫
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*/inc/
IBM i IBM i	QMQM/QCBLLESRC
Windows Windows	*MQ_INSTALLATION_PATH*\Tools\cobol\copybook (適用於 Micro Focus COBOL) *MQ_INSTALLATION_PATH*\Tools\cobol\copybook\VAcobol (適用於 IBM VisualAge COBOL)
z/OS z/OS	**thlqual**.SCSQCOBC

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

只在您的程式中包含您需要的檔案。 在 level-01 宣告之後使用一或多個 COPY 陳述式執行此動作。 這表示必要的話，您可以在程式中併入多個結構版本。 請注意， CMQV 是一個大型檔案。

以下是要包含 CMQMDV 副本檔案的 COBOL 程式碼範例:

```
01 MQM-MESSAGE-DESCRIPTOR.
COPY CMQMDV.
```

每一個結構宣告都以 level-01 項目開始; 您可以透過編碼 level-01 宣告，然後再編碼 COPY 陳述式，以在結構宣告的其餘部分中複製，來宣告結構的數個實例。 若要參照適當的實例，請使用 IN 關鍵字。

以下是 COBOL 程式碼的範例，其中包含 CMQMDV 的兩個實例:

```
* Declare two instances of MQMD
01 MY-CMQMD.
COPY CMQMDV.
01 MY-OTHER-CMQMD.
COPY CMQMDV.
*
* Set MSGTYPE field in MY-OTHER-CMQMD
MOVE MQMT-REQUEST TO MQMD-MSGTYPE IN MY-OTHER-CMQMD.
```

在 4 位元組界限上對齊結構。 如果您使用 COPY 陳述式在不是 level-01 項目的項目之後併入結構，請確保該結構是從 level-01 項目開始算起 4 個位元組的倍數。 如果您不這麼做，可能會降低應用程式的效能。

這些結構在 MQI 中使用的資料類型中有說明。 結構中的欄位說明會顯示沒有字首的欄位名稱。 在 COBOL 程式中，以結構名稱加上連字號作為欄位名稱的字首，如 COBOL 宣告中所示。 結構副本檔案中的欄位會以這種方式作為字首。

結構副本檔案中宣告的欄位名稱是大寫。 您可以改用大小寫混合或小寫。 例如，在 COBOL 宣告及複製檔案中， MQGMO 結構的欄位 *StrucId* 會顯示為 MQGMO-STRUCID。

V 字尾結構會以所有欄位的起始值來宣告，因此您只需要設定所需值不同於起始值的那些欄位。

System/390 組譯語言巨集

z/OS

IBM MQ for z/OS 提供兩個包含具名常數的組譯語言巨集，以及一個用來產生每一個結構的巨集。

它們列在 z/OS Assembler COPY 檔案 中，並安裝在 **thlqual**.SCSQMACS 中。

使用如下所示的程式碼來呼叫這些巨集:

```
MY_MQMD CMQMDA EXPIRY=0,MSGTYPE=MQMT_DATAGRAM
```

PL/I 併入檔

z/OS

IBM MQ for z/OS 提供併入檔，其中包含您在 PL/I 中撰寫 IBM MQ 應用程式時所需的所有定義。

這些檔案列在 PL/I 併入檔 中，並安裝在 **thlqual**.SCSQPLIC 目錄中:

如果您要將 IBM MQ Stub 鏈結至程式，請將這些檔案包含在程式中 (請參閱 第 858 頁的『準備要執行的程式』)。 如果您想要動態鏈結 IBM MQ 呼叫，請只包含 CMQP (請參閱 第 864 頁的『動態呼叫 IBM MQ Stub』)。 只能對批次和 IMS 程式執行動態鏈結。

撰寫佇列作業的程序化應用程式

請利用這項資訊來瞭解如何撰寫佇列作業應用程式、連接佇列管理程式、發佈/訂閱，以及開啟和關閉物件。

請使用下列鏈結，以進一步瞭解撰寫應用程式的相關資訊:

- 第 604 頁的『訊息佇列介面概觀』
- 第 616 頁的『連接至佇列管理程式並切斷與佇列管理程式的連線』
- 第 621 頁的『開啟及關閉物件』
- 第 630 頁的『將訊息放置在佇列上』
- 第 643 頁的『從佇列取得訊息』
- 第 676 頁的『撰寫發佈/訂閱應用程式』
- 第 713 頁的『查詢及設定物件屬性』
- 第 716 頁的『確定及取消工作單元』
- 第 725 頁的『使用觸發程式啟動 IBM MQ 應用程式』
- 第 741 頁的『使用 MQI 及叢集』
- z/OS 第 745 頁的『在 IBM MQ for z/OS 上使用及撰寫應用程式』
- z/OS 第 57 頁的『IBM MQ for z/OS 上的 IMS 及 IMS 橋接器應用程式』

相關概念

第 6 頁的『應用程式開發概念』
您可以使用選擇的程序化或物件導向語言來撰寫 IBM MQ 應用程式。 在開始設計及撰寫 IBM MQ 應用程式之前，請先熟悉基本 IBM MQ 概念。

第 5 頁的『開發適用於 IBM MQ 的應用程式』
您可以開發應用程式來傳送及接收訊息，以及管理佇列管理程式和相關資源。 IBM MQ 支援以許多不同語言及架構撰寫的應用程式。

第 41 頁的『IBM MQ 應用程式的設計考量』
當您決定應用程式如何利用可供您使用的平台及環境時，您需要決定如何使用 IBM MQ 所提供的特性。

第 765 頁的『撰寫用戶端程序化應用程式』
使用程序化語言在 IBM MQ 上撰寫用戶端應用程式所需的資訊。

第 837 頁的『建置程序化應用程式』
您可以使用數種程序化語言之一來撰寫 IBM MQ 應用程式，並在數個不同的平台上執行該應用程式。

第 871 頁的『處理程序化程式錯誤』
此資訊說明與應用程式 MQI 呼叫相關聯的錯誤，當它進行呼叫時，或當它的訊息遞送至其最終目的地時。

相關工作

第 887 頁的『使用 IBM MQ 範例程序化程式』
這些範例程式是以程序化語言撰寫，並示範「訊息佇列介面 (MQI)」的一般用法。 不同平台上的 IBM MQ 程式。

訊息佇列介面概觀

瞭解「訊息佇列介面 (MQI)」元件。

「訊息佇列介面」由下列項目組成:

- 呼叫 可讓程式存取佇列管理程式及其機能
- 結構，程式用來將資料傳遞至佇列管理程式，以及從佇列管理程式取得資料
- 基本資料類型，用於將資料傳遞至佇列管理程式，以及從佇列管理程式取得資料

z/OS IBM MQ for z/OS 也提供:

- z/OS 批次程式可透過兩個額外呼叫來確定及取消變更。
- 資料定義檔 (有時稱為複製檔案、巨集、併入檔及標頭檔)，定義 IBM MQ for z/OS 提供的常數值。
- *Stub* 程式 : 將編輯鏈結至您的應用程式。
- 範例程式套組，示範如何在 z/OS 平台上使用 MQI。 如需這些範例的進一步相關資訊，請參閱 第 975 頁的『使用 z/OS 的程式範例』。

IBM i IBM MQ for IBM i 也提供:

- 資料定義檔 (有時稱為複製檔案、巨集、併入檔及標頭檔)，定義 IBM MQ for IBM i 提供的常數值。
- 三個 Stub 程式，可鏈結編輯至 ILE C、ILE COBOL 及 ILE RPG 應用程式。
- 範例程式套組，示範如何在 IBM i 平台上使用 MQI。

AIX, Linux, and Windows 系統也提供:

- IBM MQ for AIX, Linux, and Windows 系統程式可以用來確定及取消變更的呼叫。
- 併入檔，定義在這些平台上提供的常數值。
- 檔案庫檔案，以鏈結您的應用程式。
- 範例程式套組，示範如何在這些平台上使用 MQI。 如需這些範例的進一步相關資訊，請參閱 第 888 頁的『在 Multiplatforms 上使用範例程式』。
- 外部交易管理程式連結的範例程式碼和執行碼。

請使用下列鏈結，以進一步瞭解 MQI:

- 第 606 頁的『MQI 呼叫』
- 第 606 頁的『同步點呼叫』
- 第 607 頁的『資料轉換、資料類型、資料定義及結構』
- 第 608 頁的『IBM MQ Stub 程式和程式庫檔案』
- 第 612 頁的『所有呼叫共用的參數』
- 第 613 頁的『指定緩衝區』
- z/OS 第 613 頁的『z/OS 批次考量』
- 第 614 頁的『AIX and Linux 信號處理』

相關概念

第 616 頁的『連接至佇列管理程式並切斷與佇列管理程式的連線』
若要使用 IBM MQ 程式設計服務，程式必須具有與佇列管理程式的連線。 使用此資訊來瞭解如何連接至佇列管理程式，以及與佇列管理程式中斷連線。

第 621 頁的『開啟及關閉物件』
此資訊提供開啟及關閉 IBM MQ 物件的見解。

第 630 頁的『將訊息放置在佇列上』
使用此資訊來瞭解如何將訊息放置在佇列上。

第 643 頁的『從佇列取得訊息』
使用此資訊來瞭解從佇列取得訊息的相關資訊。

第 713 頁的『查詢及設定物件屬性』
屬性是定義 IBM MQ 物件性質的內容。

第 716 頁的『確定及取消工作單元』
此資訊說明如何確定及取消工作單元中發生的任何可回復取得及放置作業。

第 725 頁的『使用觸發程式啟動 IBM MQ 應用程式』
瞭解觸發程式以及如何使用觸發程式來啟動 IBM MQ 應用程式。

第 741 頁的『使用 MQI 及叢集』
對於與叢集作業相關的呼叫和回覆碼，有一些特殊選項。

第 745 頁的『在 IBM MQ for z/OS 上使用及撰寫應用程式』
IBM MQ for z/OS 應用程式可以由在許多不同環境中執行的程式組成。 這表示他們可以利用多個環境中可用的設施。

第 57 頁的『IBM MQ for z/OS 上的 IMS 及 IMS 橋接器應用程式』
此資訊可協助您使用 IBM MQ 撰寫 IMS 應用程式。

MQI 呼叫

使用此資訊來瞭解「訊息佇列介面 (MQI)」中的呼叫。

MQI 中的呼叫可以分組如下:

MQCONN、MQCONNX 及 MQDISC
使用這些呼叫，將程式連接至 (不論是否使用選項)，並從佇列管理程式中斷程式的連線。 如果您為 z/OS 撰寫 CICS 程式，則不需要使用這些呼叫。 不過，如果您想要將應用程式移轉至其他平台，建議您使用它們。

MQOPEN 和 MQCLOSE
使用這些呼叫來開啟及關閉物件，例如佇列。

MQPUT 和 MQPUT1
使用這些呼叫將訊息放置在佇列上。

MQGET
使用此呼叫來瀏覽佇列上的訊息，或從佇列中移除訊息。

MQSUB、MQSUBRQ
使用這些呼叫來登錄主題的訂閱，以及要求符合訂閱的發佈。

MQINQ
使用此呼叫來查詢物件的屬性。

MQSET
使用此呼叫來設定佇列的部分屬性。 您無法設定其他類型物件的屬性。

MQBEGIN、MQCMIT 及 MQBACK
當 IBM MQ 是工作單元的協調程式時，請使用這些呼叫。 MQBEGIN 啟動工作單元。 MQCMIT 和 MQBACK 會結束工作單元，確定或回復在工作單元期間所做的更新。 IBM i IBM i 確定控制器用來協調 IBM MQ for IBM i 上的廣域工作單元。 使用原生啟動確定控制、確定及回復指令。

MQCRTMH、MQBUFMH、MQMHBUF、MQDLTMH
使用這些呼叫來建立訊息控點、將訊息控點轉換為緩衝區或將緩衝區轉換為訊息控點，以及刪除訊息控點。

MQSETMP、MQINQMP、MQDLTMP
請利用這些呼叫來設定訊息控點的訊息內容、查詢訊息內容，以及從訊息控點中刪除內容。

MQCB、MQCB_FUNCTION、MQCTL
使用這些呼叫來登錄及控制回呼函數。

MQSTAT
請利用這個呼叫來擷取先前非同步放置作業的狀態資訊。

如需 MQI 呼叫的說明，請參閱 呼叫說明 。

同步點呼叫

使用此資訊來瞭解不同平台上的同步點呼叫。

同步點呼叫提供如下:

IBM MQ for z/OS 呼叫

z/OS

IBM MQ for z/OS 提供 MQCMIT 和 MQBACK 呼叫。

在 z/OS 批次程式中使用這些呼叫，以告知佇列管理程式自前次同步點以來的所有 MQGET 及 MQPUT 作業都將成為永久 (已確定) 或取消。 如果要在其他環境中確定及取消變更，請執行下列動作:

CICS
Use commands such as EXEC CICS SYNCPOINT and EXEC CICS SYNCPOINT ROLLBACK.

IMS
使用 IMS 同步點機能，例如 IOPCB、CHKP (檢查點) 及 ROLB (回復) 呼叫的 GU (取得唯一)。

RRS

適當地使用 MQCMIT 和 MQBACK 或 SRRCMIT 和 SRRBACK。 （請參閱第 719 頁的『交易管理和可回復的資源管理程式服務』。）

註: SRRCMIT 和 SRRBACK 是原生 RRS 指令，它們不是 MQI 呼叫。

IBM i 呼叫

IBM i

IBM MQ for IBM i 提供 MQCMIT 和 MQBACK 指令。 您也可以使用 IBM i COMMIT 及 ROLLBACK 指令，或任何其他起始 IBM i 確定控制機能的指令或呼叫 (例如 EXEC CICS SYNCPOINT)。

AIX, Linux, and Windows 平台上的 IBM MQ 呼叫

ALW

IBM MQ for AIX, Linux, and Windows 提供 MQCMIT 和 MQBACK 呼叫。

在程式中使用同步點呼叫，以告知佇列管理程式自前次同步點以來的所有 MQGET 及 MQPUT 作業都將變成永久 (已確定) 或取消。 To commit and back out changes in the CICS environment, use commands such as EXEC CICS SYNCPOINT and EXEC CICS SYNCPOINT ROLLBACK.

資料轉換、資料類型、資料定義及結構

使用此資訊來瞭解使用「訊息佇列介面」時的資料轉換、基本資料類型、 IBM MQ 資料定義及結構。

資料轉換

MQXCNVC (轉換字元) 呼叫會將訊息字元資料從一個字集轉換成另一個字集。 除了 IBM MQ for z/OS 之外，只會從資料轉換結束程式使用此呼叫。

請參閱 MQXCNVC-轉換字元，以取得與 MQXCNVC 呼叫搭配使用的語法，以及第 823 頁的『寫入資料-轉換結束程式』，以取得寫入及呼叫資料轉換結束程式的指引。

基本資料類型

對於支援的程式設計語言， MQI 提供基本資料類型或非結構化欄位。

這些資料類型在 基本資料類型中有完整說明。

IBM MQ 資料定義

z/OS IBM MQ for z/OS 以 COBOL 複製檔案、組合語言巨集、單一 PL/I 併入檔、單一 C 語言併入檔及 C++ 語言併入檔的形式提供資料定義。

IBM i IBM MQ for IBM i 以 COBOL 複製檔案、RPG 複製檔案、C 語言併入檔及 C++ 語言併入檔的形式提供資料定義。

IBM MQ 隨附的資料定義檔包含:

- 所有 IBM MQ 常數及回覆碼的定義
- IBM MQ 結構及資料類型的定義
- 起始設定結構的常數定義
- 每一個呼叫的函數原型 (僅適用於 PL/I 和 C 語言)

如需 IBM MQ 資料定義檔的完整說明，請參閱第 601 頁的『IBM MQ 資料定義檔』。

結構

在每一種支援的程式設計語言的資料定義檔中，會提供與第 606 頁的『MQI 呼叫』中列出的 MQI 呼叫搭配使用的結構。 IBM i z/OS IBM MQ for z/OS 及 IBM MQ for IBM i 提供包含常數的檔案，供您在完成這些結構的部分欄位時使用。 如需這些的相關資訊，請參閱 IBM MQ 資料定義。

如需結構的摘要，請參閱 結構資料類型 。

IBM MQ Stub 程式和程式庫檔案

這裡針對每一個平台列出所提供的 Stub 程式和程式庫檔案。

如需如何在建置可執行應用程式時使用 Stub 程式和程式庫檔案的相關資訊，請參閱 第 837 頁的『建置程序化應用程式』。 如需鏈結至 C++ 程式庫檔案的相關資訊，請參閱 使用 C++ *IBM MQ* 使用 *C++*。

AIX *IBM MQ for AIX* 程式庫檔案

在 IBM MQ for AIX 上，除了作業系統所提供的 MQI 程式庫檔案之外，您還必須將程式鏈結至在其中執行應用程式的環境所提供的 MQI 程式庫檔案。

在非執行緒應用程式中，鏈結至下列其中一個檔案庫:

表 *106:* 非執行緒 *AIX* 應用程式的檔案庫檔案

程式庫檔案	**環境**
libmqm.a	適用於 C 的伺服器
libmqic.a 和 libmqm.a	適用於 C 的用戶端
libmqmzf.a	C 的可安裝服務結束程式
libmqmxa.a	伺服器 XA 介面
libmqmxa64.a	伺服器替代 XA 介面
libmqcxa.a	用戶端 XA 介面
libmqcxa64.a	用戶端替代 XA 介面
libmqmcbrt.o	Micro Focus COBOL 支援的 IBM MQ 執行時期程式庫
libmqmcb.a	COBOL 的伺服器
libmqicb.a	COBOL 的用戶端
libimqc23ia.a	Client for C++ (XLC 16)
libimqs23ia.a	Server for C++ (XLC 16)
V 9.3.5 libimqc23ca.a	Client for C++ (XLC 17)
V 9.3.5 libimqs23ca.a	Server for C++ (XLC 17)

V 9.3.5 包含 "ia" 的程式庫已使用 XLC 16 編譯器建置，而名稱中包含 "ca" 的程式庫已使用 XLC 17 編譯器建置。

在執行緒應用程式中，鏈結至下列其中一個程式庫:

表 *107:* 含執行緒作業的 *AIX* 應用程式的檔案庫檔案.

兩欄式表格，列出檔案庫檔案及每一個檔案庫檔案的環境。

程式庫檔案	**環境**
libmqm_r.a	適用於 C 的伺服器
libmqic_r.a 和 libmqm_r.a	適用於 C 的用戶端
libmqmzf_r.a	C 的可安裝服務結束程式
libmqmxa_r.a	伺服器 XA 介面
libmqmxa64_r.a	伺服器替代 XA 介面

表 *107:* 含執行緒作業的 *AIX* 應用程式的檔案庫檔案. 兩欄式表格，列出檔案庫檔案及每一個檔案庫檔案的環境。 *(繼續)*	
程式庫檔案	**環境**
libmqcxa_r.a	用戶端 XA 介面
libmqcxa64_r.a	用戶端替代 XA 介面
libimqc23ia_r.a	Client for C++ (XLC 16)
libimqs23ia_r.a	Server for C++ (XLC 16)
V 9.3.5 libimqc23ca_r.a	Client for C++ (XLC 17)
V 9.3.5 libimqs23ca_r.a	Server for C++ (XLC 17)

V 9.3.5 包含 "ia" 的程式庫已使用 XLC 16 編譯器建置，而名稱中包含 "ca" 的程式庫已使用 XLC 17 編譯器建置。

註: 您無法鏈結至多個檔案庫。 亦即，您無法同時鏈結至執行緒及非執行緒檔案庫。

IBM i *IBM MQ for IBM i* 程式庫檔案

在 IBM MQ for IBM i 中，除了作業系統提供的 MQI 程式庫檔案之外，還會將您的程式鏈結至您在其中執行應用程式的環境所提供的 MQI 程式庫檔案。

若為非執行緒應用程式:

表 *108:* 非執行緒 *IBM i* 應用程式的檔案庫檔案	
程式庫檔案	**環境**
LIBMQM	伺服器及用戶端服務程式
LIBMQIC	用戶端服務程式
IMQB23I4	C++ 基本服務程式
IMQS23I4	C++ 伺服器服務程式
LIBMQMZF	C 的可安裝結束程式

在執行緒化應用程式中:

表 *109:* 含執行緒作業的 *IBM i* 應用程式的檔案庫檔案	
程式庫檔案	**環境**
LIBMQM_R	伺服器與用戶端服務程式
IMQB23I4_R	C++ 基本服務程式
IMQS23I4_R	C++ 伺服器服務程式
LIBMQMZF_R	C 的可安裝結束程式
LIBMQIC_R	用戶端服務程式

在 IBM MQ for IBM i 上，您可以使用 C++ 撰寫應用程式。若要查看如何鏈結 C++ 應用程式，以及如需使用 C++ 的所有層面的完整詳細資料，請參閱 使用 C++。

Linux *IBM MQ for Linux* 程式庫檔案

在 IBM MQ for Linux 上，除了作業系統提供的 MQI 程式庫檔案之外，您還必須將程式鏈結至在其中執行應用程式的環境所提供的 MQI 程式庫檔案。

在非執行緒應用程式中，鏈結至下列其中一個檔案庫:

表 *110:* 非執行緒 *Linux* 應用程式的檔案庫檔案	
程式庫檔案	**環境**
libmqm.so	適用於 C 的伺服器
libmqic.so 和 libmqm.so	適用於 C 的用戶端
libmqmzf.so	C 的可安裝服務結束程式
libmqmxa.so	伺服器 XA 介面
libmqmxa64.so	伺服器替代 XA 介面
libmqcxa.so	用戶端 XA 介面
libmqcxa64.so	用戶端替代 XA 介面
libimqc23gl.so	適用於 C++ 的用戶端
libimqs23gl.so	C++ 的伺服器

在執行緒應用程式中，鏈結至下列其中一個程式庫:

表 *111:* 含執行緒作業的 *Linux* 應用程式的檔案庫檔案	
程式庫檔案	**環境**
libmqm_r.so	適用於 C 的伺服器
libmqic_r.so 和 libmqm_r.so	適用於 C 的用戶端
libmqmzf_r.so	C 的可安裝服務結束程式
libmqmxa_r.so	伺服器 XA 介面
libmqmxa64_r.so	伺服器替代 XA 介面
libmqcxa_r.so	用戶端 XA 介面
libmqcxa64_r.so	用戶端替代 XA 介面
libimqc23gl_r.so	適用於 C++ 的用戶端
libimqs23gl_r.so	C++ 的伺服器

註: 您無法鏈結至多個檔案庫。 亦即，您無法同時鏈結至執行緒及非執行緒檔案庫。

Windows *IBM MQ for Windows* 程式庫檔案

在 IBM MQ for Windows 上，除了作業系統提供的 MQI 程式庫檔案之外，您還必須將程式鏈結至在其中執行應用程式的環境所提供的 MQI 程式庫檔案:

表 *112: Windows* 應用程式的程式庫檔案	
程式庫檔案	**環境**
MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib	Server for C (32 位元)
MQ_INSTALLATION_PATH\Tools\Lib\mqic.lib	Client for C (32 位元)
MQ_INSTALLATION_PATH\Tools\Lib\mqmxa.lib	適用於 C (32 位元) 的伺服器 XA 介面

表 112: Windows 應用程式的程式庫檔案 (繼續)	
程式庫檔案	**環境**
MQ_INSTALLATION_PATH\Tools\Lib\mqcxa.lib	適用於 C 的用戶端 XA 介面 (32 位元)
MQ_INSTALLATION_PATH\Tools\Lib\mqicxa.lib	用戶端 MTS for C (32 位元)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcics4.lib32	伺服器 TXSeries CICS 支援 C (32 位元)
MQ_INSTALLATION_PATH\Tools\Lib\mqccics4.lib32	C (32 位元) 的用戶端 TXSeries CICS 支援
MQ_INSTALLATION_PATH\Tools\Lib\mqmzf.lib	C (32 位元) 的可安裝服務結束程式
MQ_INSTALLATION_PATH\Tools\Lib\mqmcbb.lib	Server for IBM COBOL (32 位元)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcb.lib	Server for Micro Focus COBOL (32 位元)
MQ_INSTALLATION_PATH\Tools\Lib\mqiccbb.lib	適用於 IBM COBOL (32 位元) 的用戶端
MQ_INSTALLATION_PATH\Tools\Lib\mqiccb.lib	Micro Focus COBOL 用戶端 (32 位元)
MQ_INSTALLATION_PATH\Tools\Lib\imqs23vn.lib	Server for C++ (32 位元)
MQ_INSTALLATION_PATH\Tools\Lib\imqc23vn.lib	Client for C++ (32 位元)
MQ_INSTALLATION_PATH\Tools\Lib\imqb23vn.lib	C++ 基本程式 (32 位元)
MQ_INSTALLATION_PATH\Tools\Lib\imqx23vn.lib	Client MTS for C++ (32 位元)
MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib	Server for C (64 位元)
MQ_INSTALLATION_PATH\Tools\Lib64\mqic.lib	Client for C (64 位元)
MQ_INSTALLATION_PATH\Tools\Lib64\mqmxa.lib	Server XA interface for C (64 位元)
MQ_INSTALLATION_PATH\Tools\Lib64\mqcxa.lib	適用於 C 的用戶端 XA 介面 (64 位元)
MQ_INSTALLATION_PATH\Tools\Lib64\mqicxa.lib	用戶端 MTS for C (64 位元)
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcbb.lib	Server for IBM COBOL (64 位元)
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcb.lib	Server for Micro Focus COBOL (64 位元)
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccbb.lib	Client for IBM COBOL (64 位元)
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccb.lib	Client for Micro Focus COBOL (64 位元)
MQ_INSTALLATION_PATH\Tools\Lib64\imqs23vn.lib	Server for C++ (64 位元)
MQ_INSTALLATION_PATH\Tools\Lib64\imqc23vn.lib	Client for C++ (64 位元)
MQ_INSTALLATION_PATH\Tools\Lib64\imqb23vn.lib	Base for C++ (64 位元)
MQ_INSTALLATION_PATH\Tools\Lib64\imqx23vn.lib	Client MTS for C++ (64 位元)

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

使用 `amqmdnet.dll` 來編譯 .NET 程式。 如需相關資訊，請參閱 第 464 頁的『開發 .NET 應用程式』 一節內的 第 513 頁的『編譯 IBM MQ .NET 程式』 。

這些檔案是為了與舊版相容而提供:

```
mqic32.lib
```

```
mqic32xa.lib
```

z/OS *IBM MQ for z/OS Stub* 程式

在執行以 IBM MQ for z/OS 撰寫的程式之前，您必須針對您執行應用程式的環境，將它鏈結至 IBM MQ for z/OS 所提供的 Stub 程式。

Stub 程式提供第一個階段來處理對 IBM MQ for z/OS 可處理之要求的呼叫。

IBM MQ for z/OS 提供下列 Stub 程式:

CSQBSTUB
z/OS 批次程式的 Stub 程式

CSQBRRSI
透過 MQI 使用 RRS 的 z/OS 批次程式的 Stub 程式

CSQBRSTB
直接使用 RRS 的 z/OS 批次程式的 Stub 程式

CSQCSTUB
CICS 程式的 Stub 程式

CSQQSTUB
IMS 程式的 Stub 程式

CSQXSTUB
分散式佇列非 CICS 結束程式的 Stub 程式

CSQASTUB
資料轉換結束程式的 Stub 程式

小心: 如果您使用針對特定環境所列出的 Stub 程式以外的 Stub 程式，它可能會有無法預期的結果。

註: 如果您使用 CSQBRSTB Stub 程式，請使用 SYS1.CSSLIB。 (SYS1.CSSLIB 也稱為 可呼叫服務程式庫)。如需 RRS 的相關資訊，請參閱 第 719 頁的『交易管理和可回復的資源管理程式服務』。

或者，您可以從程式內動態呼叫 Stub。 此技術在 第 864 頁的『動態呼叫 IBM MQ Stub』中說明。

在 IMS 中，您也可能需要使用 IBM MQ 所提供的特殊語言介面模組。

請勿執行使用相同 IMS MPP 區域中的 CSQBSTUB 和 CSQQSTUB 來鏈結編輯的應用程式。 這可能會導致問題，例如 DFS3607I 或 CSQQ005E 訊息。 位址空間中的第一個 MQCONN 呼叫會決定使用哪個介面，因此 CSQQSTUB 和 CSQBSTUB 交易必須在不同的 IMS 訊息區域中執行。

所有呼叫共用的參數

所有呼叫共有兩種類型的參數: 控點及回覆碼。

使用控點

所有 MQI 呼叫都使用一或多個 控點。 這些可識別適用於呼叫的佇列管理程式、佇列或其他物件、訊息或訂閱。

若要讓程式與佇列管理程式進行通訊，程式必須具有唯一 ID ，以用來識別該佇列管理程式。 此 ID 稱為 連線控點，有時稱為 *Hconn*。 對於 CICS 程式，連線控點一律為零。 對於所有其他平台或程式樣式，當程式連接至佇列管理程式時， MQCONN 或 MQCONNX 呼叫會傳回連線控點。 當程式使用其他呼叫時，會傳遞連線控點作為輸入參數。

若要讓程式使用 IBM MQ 物件，該程式必須具有用來識別該物件的唯一 ID。 此 ID 稱為 物件控點，有時稱為 *Hobj*。 當程式開啟物件以使用它時， MQOPEN 呼叫會傳回控點。 當程式使用後續的 MQPUT、MQGET、MQINQ、MQSET 或 MQCLOSE 呼叫時，會傳遞物件控點作為輸入參數。

同樣地， MQSUB 呼叫會傳回 訂閱控點 或 *Hsub*，用來識別後續 MQGET、MQCB 或 MQSUBRQ 呼叫中的訂閱，且處理訊息內容的某些呼叫會使用 訊息控點 或 *Hmsg*。

瞭解回覆碼

每一個呼叫都會傳回完成碼和原因碼作為輸出參數。 這些統稱為 回覆碼。

若要顯示呼叫是否成功，當呼叫完成時，每一個呼叫都會傳回 完成碼 。 完成碼通常是 MQCC_OK (指示成功) 或 MQCC_FAILED (指示失敗)。 部分呼叫可能傳回中間狀態 MQCC_WARNING ，指出局部成功。

每一個呼叫也會傳回 原因碼 ，其中顯示呼叫失敗或局部成功的原因。 有許多原因碼，涵蓋佇列已滿、佇列不容許取得作業，以及佇列管理程式未定義特定佇列等情況。 程式可以使用原因碼來決定如何繼續進行。 例如，他們可以提示使用者變更其輸入資料，然後重新進行呼叫，或者他們可以傳回錯誤訊息給使用者。

當完成碼為 MQCC_OK 時，原因碼一律為 MQRC_NONE。

每一個呼叫的完成碼和原因碼都會與該呼叫的說明一起列出。 請參閱 通話說明 ，並從清單中選取適當的通話。

如需更詳細的資訊，包括更正動作的構想，請參閱:

- z/OS IBM MQ for z/OS 訊息、完成及原因碼 for IBM MQ for z/OS
- 訊息及原因碼 ，適用於所有其他 IBM MQ 平台

指定緩衝區

只有在需要緩衝區時，佇列管理程式才會參照緩衝區。 如果您在呼叫時不需要緩衝區，或緩衝區長度為零，您可以對緩衝區使用空值指標。

在指定您需要的緩衝區大小時，一律使用資料長度。

當您使用緩衝區來保留呼叫的輸出 (例如，保留 MQGET 呼叫的訊息資料，或 MQINQ 呼叫所查詢的屬性值) 時，如果您指定的緩衝區無效或在唯讀儲存體中，佇列管理程式會嘗試傳回原因碼。 不過，它可能無法一律傳回原因碼。

z/OS *z/OS* 批次考量

呼叫 MQI 的 z/OS 批次程式可以處於監督者或問題狀態。

不過，它們必須符合下列條件:

- 它們必須處於作業模式，而不是服務要求區塊 (SRB) 模式。
- 它們必須處於主要位址空間控制 (ASC) 模式 (非存取登錄 ASC 模式)。
- 它們不得處於跨記憶體模式。 主要位址空間號碼 (ASN) 必須等於次要 ASN 及起始 ASN。
- 它們不得用作強制性公積金 (MPF) 跳出程式。
- 無法保留任何 z/OS 鎖定。
- 在 FRR 堆疊上不能有函數回復常式 (FRR)。
- 任何程式狀態單字 (PSW) 索引鍵都可以對 MQCONN 或 MQCONNX 呼叫有效 (前提是索引鍵與 TCB 索引鍵中的儲存體相容)，但後續呼叫會使用 MQCONN 或 MQCONNX 所傳回的連線控點:
  - 必須具有在 MQCONN 或 MQCONNX 呼叫中使用的相同 PSW 索引鍵
  - 必須在相同的 PSW 金鑰下具有可存取的參數 (適用於寫入，如果適當的話)
  - 必須在相同作業 (TCB) 下發出，但不在作業的任何子作業中
- 它們可以是 24 位元或 31 位元定址模式。 不過，如果有效的 24 位元定址模式，則必須將參數位址解譯為有效的 31 位元位址。

如果不符合其中任何條件，則可能會發生程式檢查。 在某些情況下，呼叫會失敗，且會傳回原因碼。

Linux AIX *AIX and Linux* 考量

開發 AIX and Linux 應用程式時需要注意的考量。

Linux AIX *AIX and Linux* 系統中的分出系統呼叫

在 IBM MQ 應用程式中使用分出系統呼叫時，請注意下列考量。

如果您的應用程式想要使用 fork，則該應用程式的母項處理程序應該先呼叫 fork，然後再進行任何 IBM MQ 呼叫 (例如， MQCONN)，或使用 **ImqQueueManager** 來建立 IBM MQ 物件。

如果您的應用程式想要在進行任何 IBM MQ 呼叫之後建立子程序，則應用程式碼必須搭配使用 fork() 與 exec()，以確保子項是新實例，而不是母項的確切副本。

如果您的應用程式不使用 exec()，則在子程序內所進行的 IBM MQ API 呼叫會傳回 MQRC_ENVIRONMENT_ERROR。

Linux AIX *AIX and Linux* 信號處理

一般而言， AIX and Linux 系統已從非執行緒 (處理程序) 環境移至多執行緒環境。 在許多情況下，信號和信號處理雖然受到支援，但並不適合多執行緒環境，且存在各種限制。

一般而言， AIX and Linux 系統已從非執行緒 (處理程序) 環境移至多執行緒環境。 在非執行緒環境中，部分函數只能使用信號來實作，但大部分應用程式不需要知道信號及信號處理。 在多執行緒環境中，執行緒型基本元素支援使用信號在非執行緒環境中實作的部分函數。

在許多情況下，信號和信號處理雖然受到支援，但並不適合多執行緒環境，且存在各種限制。 在多執行緒環境中，當您將應用程式碼與不同的中介軟體程式庫 (作為應用程式的一部分執行) 整合時，這可能會有問題，其中每一個都嘗試處理信號。 儲存及還原信號處理程式 (每個處理程序定義) 的傳統方法，在處理程序內只有一個執行緒執行時有效，在多執行緒環境中無法運作。 這是因為許多執行執行緒可能嘗試儲存及還原整個處理程序的資源，導致無法預期的結果。

Linux AIX 無執行緒應用程式

每一個 MQI 函數都會為信號設定自己的信號處理程式。 在 MQI 函數呼叫期間，會取代使用者的處理程式。 其他信號可以由使用者撰寫的處理程式以正常方式捕捉。

每一個 MQI 函數都會針對信號設定自己的信號處理程式:

SIGALRM
SIGBUS
SIGFPE
SIGSEGV
SIGILL

在 MQI 函數呼叫期間，會取代使用者的處理程式。 其他信號可以由使用者撰寫的處理程式以正常方式捕捉。 如果您未安裝處理程式，則預設動作 (例如，忽略、核心傾出或結束) 會保留在原處。

在 IBM MQ 處理同步信號 (SIGSEGV、SIGBUS、SIGFPE、SIGILL) 之後，它會嘗試將信號傳遞至任何已登錄的信號處理程式，然後再進行 MQI 函數呼叫。

Linux AIX 執行緒應用程式

在 MQDISC 之前，會將執行緒視為從 MQCONN (或 MQCONNX) 連接至 IBM MQ 。

同步信號

在特定執行緒中產生同步信號。

AIX and Linux 系統安全地容許在整個處理程序中設定這類信號的信號處理程式。 不過，當任何執行緒連接至 IBM MQ 時， IBM MQ 會在應用程式程序中為下列信號設定自己的處理程式:

SIGBUS
SIGFPE
SIGSEGV
SIGILL

如果您是撰寫多執行緒應用程式，則每一個信號只有一個處理程序層面的信號處理程式。 當 IBM MQ 設定自己的同步信號處理程式時，它會儲存每一個信號先前已登錄的任何處理程式。 在 IBM MQ 處理其中一個列出的信號之後， IBM MQ 會嘗試呼叫在程序內第一個 IBM MQ 連線時生效的信號處理程式。 當所有應用程式執行緒都已切斷與 IBM MQ 的連線時，會還原先前登錄的處理程式。

因為信號處理程式是由 IBM MQ 儲存及還原，所以應用程式執行緒不得針對這些信號建立信號處理程式，而同一處理程序的另一個執行緒也可能連接至 IBM MQ。

註: 當執行緒連接至 IBM MQ 時，當應用程式或中介軟體程式庫 (作為應用程式的一部分執行) 建立信號處理程式時，應用程式的信號處理程式必須在處理該信號期間呼叫對應的 IBM MQ 處理程式。

建立及還原信號處理程式時，一般原則是要儲存的最後一個信號處理程式必須是要還原的第一個:

- 當應用程式在連接至 IBM MQ 之後建立信號處理程式時，必須先還原先前的信號處理程式，然後應用程式才會從 IBM MQ 中斷連線。
- 當應用程式在連接至 IBM MQ 之前建立信號處理程式時，應用程式必須先中斷與 IBM MQ 的連線，然後再還原其信號處理程式。

註: 如果不遵守一般原則，最後一個要儲存的信號處理程式必須是第一個要還原的信號處理程式，可能會導致應用程式中發生非預期的信號處理，並可能導致應用程式遺失信號。

非同步信號

IBM MQ 不會在執行緒應用程式中使用任何非同步信號，除非它們是用戶端應用程式。

執行緒用戶端應用程式的其他考量

在伺服器 I/O 期間， IBM MQ 會處理下列信號。 這些信號由通訊堆疊定義。 當執行緒連接至佇列管理程式時，應用程式不得建立這些信號的信號處理程式:

SIGPIPE (適用於 TCP/IP)

Linux AIX 在 *MQI* 中使用 *AIX and Linux* 信號處理時的其他考量

在 AIX and Linux 上使用 MQI 進行信號處理時，對於捷徑應用程式、信號處理程式內的 MQI 函數呼叫、MQI 呼叫期間的信號、使用者結束程式及可安裝的服務，以及 VMS 結束程式處理程式，還有其他考量。

捷徑 (授信) 應用程式

捷徑應用程式在與 IBM MQ 相同的處理程序中執行，因此在多執行緒環境中執行。

在此環境中， IBM MQ 會處理同步信號 SIGSEGV、SIGBUS、SIGFPE 及 SIGILL。 當 Fastpath 應用程式連接至 IBM MQ 時，不得將所有其他信號遞送至該應用程式。 相反地，應用程式必須封鎖或處理它們。 如果 Fastpath 應用程式截取這類事件，則必須停止並重新啟動佇列管理程式，或它可能處於未定義的狀態。 如需 MQCONNX 下 Fastpath 應用程式之限制的完整清單，請參閱 第 618 頁的『使用 MQCONNX 呼叫連接至佇列管理程式』。

信號處理程式內的 MQI 函數呼叫

當您在信號處理程式中時，請勿呼叫 MQI 函數。

如果您嘗試在另一個 MQI 函數作用中時從信號處理程式呼叫 MQI 函數，則會傳回 MQRC_CALL_IN_PROGRESS。 如果您嘗試從信號處理程式呼叫 MQI 函數，但沒有其他 MQI 函數作用中，則可能會在作業期間失敗，因為作業系統限制只能從處理程式發出選擇性呼叫，或在處理程式內發出選擇性呼叫。

對於 C++ 解構子方法 (在程式結束期間可能自動呼叫) ，您可能無法停止呼叫 MQI 函數。 忽略有關 MQRC_CALL_IN_PROGRESS 的任何錯誤。 如果信號處理程式呼叫 exit () ， IBM MQ 會照常取消同步點中未確定的訊息，並關閉任何開啟的佇列。

MQI 呼叫期間的信號

MQI 函數不會傳回程式碼 EINTR 或任何與應用程式相等的程式碼。

如果在 MQI 呼叫期間發生信號，且處理程式呼叫 傳回，則該呼叫會繼續執行，如同未發生信號一樣。 尤其是 MQGET 無法被信號岔斷，無法立即將控制項傳回給應用程式。 如果您想要中斷 MQGET ，請將佇列設為 GET_DISABLED; 或者，在呼叫 MQGET 時使用迴圈，並設定有限時間期限 (MQGMO_WAIT ，並設定

gmo.WaitInterval)，然後使用信號處理程式 (在非執行緒環境中) 或執行緒環境中的對等函數來設定可中斷迴圈的旗標。

AIX 在 AIX 環境中， IBM MQ 需要重新啟動信號所中斷的系統呼叫。 使用 sigaction (2) 建立您自己的信號處理程式時，請在新動作結構的 sa_flags 欄位中設定 SA_RESTART 旗標，否則 IBM MQ 可能無法完成信號所岔斷的任何呼叫。

使用者結束程式和可安裝的服務

在多執行緒環境中作為 IBM MQ 處理程序的一部分執行的使用者結束程式及可安裝服務，與捷徑應用程式的限制相同。 請將這些視為永久連接至 IBM MQ ，因此不使用信號或非安全執行緒作業系統呼叫。

連接至佇列管理程式並切斷與佇列管理程式的連線

若要使用 IBM MQ 程式設計服務，程式必須具有與佇列管理程式的連線。 使用此資訊來瞭解如何連接至佇列管理程式，以及與佇列管理程式中斷連線。

建立此連線的方式取決於平台及程式運作所在的環境:

Multi **IBM MQ for Multiplatforms**
在這些環境中執行的程式可以使用 MQCONN MQI 呼叫來連接至佇列管理程式，以及使用 MQDISC 呼叫來中斷與佇列管理程式的連線。 或者，程式可以使用 MQCONNX 呼叫。

z/OS **IBM MQ for z/OS 批次**
在此環境中執行的程式可以使用 MQCONN MQI 呼叫來連接至佇列管理程式，以及使用 MQDISC 呼叫來中斷與佇列管理程式的連線。 或者，程式可以使用 MQCONNX 呼叫。

z/OS 批次程式可以連續或同時連接至相同 TCB 上的多個佇列管理程式。

z/OS **IMS**
當 IMS 控制區域啟動時，它會連接至一或多個佇列管理程式。 此連線由 IMS 指令控制。 如需如何在 z/OS 上控制 IMS 配接器的相關資訊，請參閱 管理 IBM MQ for z/OS。 不過，訊息佇列作業 IMS 程式的寫出器必須使用 MQCONN MQI 呼叫來指定它們要連接的佇列管理程式。 他們可以使用 MQDISC 呼叫來中斷與該佇列管理程式的連線。

在建立同步點的 IMS 呼叫之後，在處理另一個使用者的訊息之前， IMS 配接器會確保應用程式關閉控點並切斷與佇列管理程式的連線。 請參閱第 718 頁的『IMS 應用程式中的同步點』。

IMS 程式可以連續或同時連接至相同 TCB 上的多個佇列管理程式。

z/OS **CICS Transaction Server for z/OS**
CICS 程式不需要執行任何工作來連接至佇列管理程式，因為 CICS 系統本身已連接。 此連線通常在起始設定時自動建立，但您也可以使用 IBM MQ for z/OS 隨附的 CKQC 交易。 如需 CKQC 的相關資訊，請參閱 管理 IBM MQ for z/OS。

CICS 作業只能連接至 CICS 區域所連接的佇列管理程式。

CICS 程式也可以使用 MQI 連接及斷線呼叫 (MQCONN 及 MQDISC)。 您可能想要這樣做，以便您可以將這些應用程式移轉至具有最少重新編碼的非 CICS 環境。 不過，在 CICS 環境中，這些呼叫 一律 順利完成。 這表示回覆碼可能不會反映佇列管理程式連線的真實狀態。

TXSeries for Windows and Open Systems
這些程式不需要執行任何工作來連接至佇列管理程式，因為 CICS 系統本身已連接。 因此，一次只支援一個連線。 CICS 應用程式必須發出 MQCONN 呼叫來取得連線控點，並在結束之前發出 MQDISC 呼叫。

請使用下列鏈結，以進一步瞭解與佇列管理程式連接及中斷連線的相關資訊:

- 第 617 頁的『使用 MQCONN 呼叫連接至佇列管理程式』
- 第 618 頁的『使用 MQCONNX 呼叫連接至佇列管理程式』
- 第 621 頁的『使用 MQDISC 切斷程式與佇列管理程式的連線』

相關概念
第 604 頁的『訊息佇列介面概觀』
瞭解「訊息佇列介面 (MQI)」元件。

第 621 頁的『開啟及關閉物件』
此資訊提供開啟及關閉 IBM MQ 物件的見解。

第 630 頁的『將訊息放置在佇列上』
使用此資訊來瞭解如何將訊息放置在佇列上。

第 643 頁的『從佇列取得訊息』
使用此資訊來瞭解從佇列取得訊息的相關資訊。

第 713 頁的『查詢及設定物件屬性』
屬性是定義 IBM MQ 物件性質的內容。

第 716 頁的『確定及取消工作單元』
此資訊說明如何確定及取消工作單元中發生的任何可回復取得及放置作業。

第 725 頁的『使用觸發程式啟動 IBM MQ 應用程式』
瞭解觸發程式以及如何使用觸發程式來啟動 IBM MQ 應用程式。

第 741 頁的『使用 MQI 及叢集』
對於與叢集作業相關的呼叫和回覆碼，有一些特殊選項。

第 745 頁的『在 IBM MQ for z/OS 上使用及撰寫應用程式』
IBM MQ for z/OS 應用程式可以由在許多不同環境中執行的程式組成。 這表示他們可以利用多個環境中可用的設施。

第 57 頁的『IBM MQ for z/OS 上的 IMS 及 IMS 橋接器應用程式』
此資訊可協助您使用 IBM MQ 撰寫 IMS 應用程式。

使用 *MQCONN* 呼叫連接至佇列管理程式

使用此資訊來瞭解如何使用 MQCONN 呼叫連接至佇列管理程式。

一般而言，您可以連接至特定佇列管理程式或預設佇列管理程式:

- 對於 IBM MQ for z/OS，在批次環境中，會在 CSQBDEFV 模組中指定預設佇列管理程式。
- 對於 IBM MQ for Multiplatforms，預設佇列管理程式指定在 mqs.ini 檔案中。

或者，在 z/OS MVS 批次、TSO 和 RRS 環境中，您可以連接至佇列共用群組內的任何一個佇列管理程式。 MQCONN 或 MQCONNX 要求會選取群組的任何一個作用中成員。

當您連接至佇列管理程式時，它必須是作業的本端。 它必須屬於與 IBM MQ 應用程式相同的系統。

在 IMS 環境中，佇列管理程式必須連接至 IMS 控制區域及程式使用的相依區域。 安裝 IBM MQ for z/OS 時，會在 CSQQDEFV 模組中指定預設佇列管理程式。

使用 TXSeries CICS 環境，以及 TXSeries for Windows 和 AIX，佇列管理程式必須定義為 CICS 的 XA 資源。

若要連接至預設佇列管理程式，請呼叫 MQCONN ，並指定完全由空白組成或以空值 (X'00 ') 字元開頭的名稱。

應用程式必須獲得授權，才能順利連接至佇列管理程式。 如需相關資訊，請參閱保護。

MQCONN 的輸出為:

- 連線控點 (**Hconn**)
- 完成碼
- 原因碼

在後續的 MQI 呼叫中使用連線控點。

如果原因碼指出應用程式已連接至該佇列管理程式，則傳回的連線控點與應用程式第一次連接時所傳回的連線控點相同。 在此狀況下，應用程式不得發出 MQDISC 呼叫，因為呼叫端應用程式預期保持連接。

連線控點的範圍與物件控點的範圍相同 (請參閱 第 623 頁的『使用 MQOPEN 呼叫開啟物件』)。

參數的說明在 MQCONN 中 MQCONN 呼叫的說明中提供。

當您發出呼叫時，如果佇列管理程式處於靜止狀態，或佇列管理程式正在關閉，則 MQCONN 呼叫會失敗。

MQCONN 或 MQCONNX 的範圍

MQCONN 或 MQCONNX 呼叫的範圍通常是發出它的執行緒。 也就是說，從呼叫傳回的連線控點僅在發出呼叫的執行緒內有效。 使用控點一次只能進行一次呼叫。 如果是從不同的執行緒使用它，則會因為無效而拒絕它。 如果您應用程式中有多個執行緒，且每一個都想要使用 IBM MQ 呼叫，則每一個都必須發出 MQCONN 或 MQCONNX。

當處理程序發出多個 MQCONN 呼叫時，不需要對相同的佇列管理程式進行每一次呼叫。 不過，一次只能從一個執行緒建立一個 IBM MQ 連線。 或者，請考量 第 619 頁的『與 MQCONNX 共用 (與執行緒無關) 連線』，以容許從單一執行緒使用多個 IBM MQ 連線，以及從任何執行緒使用 IBM MQ 連線。[7]

如果您的應用程式以用戶端身分執行，它可以連接至執行緒內的多個佇列管理程式。

使用 *MQCONNX* 呼叫連接至佇列管理程式

MQCONNX 呼叫類似於 MQCONN 呼叫，但包括用來控制呼叫運作方式的選項。

作為 MQCONNX 的輸入，您可以提供佇列管理程式名稱 z/OS，或 z/OS 共用佇列系統上的佇列共用群組名稱。 用來控制如何以稱為 MQCNO 的結構來建立佇列管理程式連線的選項。

MQCONNX 的輸出如下:

- 連線控點 (Hconn)
- 完成碼
- 原因碼

您可以在後續的 MQI 呼叫中使用連線控點。

MQCNO 結構的 *Options* 欄位中設定的連接選項容許控制連線的數個屬性。 特別值得注意的是下列選項群組:

- 連結選項容許建立 授信應用程式 。 授信應用程式暗示 IBM MQ 應用程式與本端佇列管理程式代理程式變成相同的處理程序。 因為代理程式處理程序不再需要使用介面來存取佇列管理程式，所以這些應用程式會變成佇列管理程式的延伸。 透過指定 MQCNO_FASTPATH_BINDING 選項來要求此行為。 如需適用於授信應用程式之限制的相關資訊，請參閱 第 618 頁的『授信應用程式的限制』。
- 控點共用選項容許建立共用連線。 共用連線可以在相同處理程序內的不同執行緒之間共用控點。 如需共用連線的相關資訊，請參閱 第 619 頁的『與 MQCONNX 共用 (與執行緒無關) 連線』。

MQCNO 也可讓應用程式控制如何鑑別佇列管理程式的連線。 鑑別認證可以在 MQCNO 結構所參照的 MQCSP 結構中指定。

如需 MQCONNX 呼叫參數及可控制之連線屬性的完整說明，請參閱 MQCONNX-Connect 佇列管理程式 (延伸)。

授信應用程式的限制

適用於授信應用程式的限制。 部分限制適用於所有平台，其他則是平台專用。

T

- 您必須明確切斷授信應用程式與佇列管理程式的連線。
- 在使用 **endmqm** 指令結束佇列管理程式之前，您必須先停止授信應用程式。
- 您不得搭配使用非同步信號和計時器岔斷 (例如 `sigkill`) 與 MQCNO_FASTPATH_BINDING。
- 在所有平台上，當相同程序中的另一個執行緒連接至不同的佇列管理程式時，授信應用程式內的執行緒無法連接至佇列管理程式。
- Linux AIX 在 AIX and Linux 系統上，您必須使用 mqm 作為所有 MQI 呼叫的有效 userID 及 groupID 。 您可以在發出需要鑑別的非 MQI 呼叫 (例如，開啟檔案) 之前變更這些 ID ，但必須在發出下一個 MQI 呼叫之前，將它變回 mqm。

[7] 將多執行緒應用程式與 IBM MQ for AIX or Linux 系統搭配使用時，您需要確保應用程式具有足夠的執行緒堆疊大小。 當多執行緒應用程式自行或搭配其他信號處理程式 (例如 CICS) 來發出 MQI 呼叫時，請考慮使用 256 KB 或更大的堆疊大小。

- IBM i 在 IBM i 上:
  1. 授信應用程式必須在 QMQM 使用者設定檔下執行。 使用者設定檔是 QMQM 群組的成員或程式採用 QMQM 權限是不夠的。 可能無法使用 QMQM 使用者設定檔來登入互動式工作，或在工作說明中指定執行授信應用程式的工作。 在此情況下，一種方法是使用 IBM i 設定檔交換 API 函數、QSYGETPH、QWTSETP 及 QSYRLSPH，以在 IBM MQ 程式執行時暫時將工作的現行使用者變更為 QMQM。 IBM i 應用程式設計介面 說明文件的 安全 API 小節中提供這些函數的詳細資料及其使用範例。
  2. 請勿使用「系統要求選項 2」來取消授信應用程式，或使用 ENDJOB 來結束其執行所在的工作。
- ALW 在 AIX, Linux, and Windows 系統上，不支援授信 32 位元應用程式。 如果您嘗試執行授信 32 位元應用程式，則會將它降級至標準連結連線。

與 *MQCONNX* 共用 (與執行緒無關) 連線

使用此資訊來瞭解與 MQCONNX 的共用連線，以及要考量的一些使用注意事項。

註: 在 IBM MQ for z/OS 上不受支援。

在 IBM MQ for z/OS 以外的 IBM MQ 平台上，只有建立連線的執行緒才能使用與 MQCONN 建立的連線。 MQCONNX 呼叫的選項可讓您建立可由程序中所有執行緒共用的連線。 如果您的應用程式是在需要在相同執行緒上發出 MQI 呼叫的交易式環境中執行，則必須使用下列預設選項:

MQCNO_HANDLE_SHARE_NONE
建立非共用連線。

在大部分其他環境中，您可以使用下列其中一個與執行緒無關的共用連線選項:

MQCNO_HANDLE_SHARE_BLOCK
建立共用連線。 在 MQCNO_HANDLE_SHARE_BLOCK 連線上，如果另一個執行緒上的 MQI 呼叫目前正在使用連線，則 MQI 呼叫會等待直到現行 MQI 呼叫完成為止。

MQCNO_HANDLE_SHARE_NO_BLOCK
建立共用連線。 在 MQCNO_HANDLE_SHARE_NO_BLOCK 連線上，如果另一個執行緒上的 MQI 呼叫目前正在使用連線，則 MQI 呼叫會立即失敗，原因為 MQRC_CALL_IN_PROGRESS。

除了 MTS (Microsoft Transaction Server) 環境之外，預設值為 MQCNO_HANDLE_SHARE_NONE。 在 MTS 環境中，預設值為 MQCNO_HANDLE_SHARE_BLOCK。

從 MQCONNX 呼叫傳回連線控點。 處理程序中任何執行緒的後續 MQI 呼叫可以使用此控點，並將這些呼叫與從 MQCONNX 傳回的控點相關聯。 使用單一共用控點的 MQI 呼叫會在執行緒之間序列化。

例如，共用控點可以使用下列活動序列:

1. 執行緒 1 發出 MQCONNX 並取得共用控點 *h1*
2. 執行緒 1 會開啟佇列，並使用 *h1* 發出取得要求
3. 執行緒 2 使用 *h1* 發出放置要求
4. 執行緒 3 使用 *h1* 發出放置要求
5. 執行緒 2 問題 MQDISC 使用 *h1*

當任何執行緒正在使用控點時，其他執行緒無法存取連線。 在可接受執行緒等待來自另一個執行緒的任何先前呼叫完成的情況下，請搭配使用 MQCONNX 與選項 MQCNO_HANDLE_SHARE_BLOCK。

不過，封鎖可能會造成困難。 假設在步驟 第 619 頁的『2』中，執行緒 1 會發出 get 要求，等待可能尚未到達的訊息 (get with wait)。 在此情況下，執行緒 2 和 3 也會保留等待 (封鎖) 狀態，只要執行緒 1 上的取得要求所採用的時間。 如果您偏好在控點上已執行另一個 MQI 呼叫時， MQI 呼叫會傳回錯誤，請搭配使用 MQCONNX 與選項 MQCNO_HANDLE_SHARE_NO_BLOCK。

共用連線使用注意事項

1. 開啟物件所建立的任何物件控點 (Hobj) 都會與 Hconn 相關聯; 因此對於共用 Hconn ， 任何使用 Hconn 的執行緒也會共用及使用 Hobjs。 同樣地，在 Hconn 下啟動的任何工作單元都會與該 Hconn 相關聯; 因此，這也會與共用的 Hconn 在執行緒之間共用。

2. 任何 執行緒都可以呼叫 MQDISC 來切斷共用 Hconn 的連線，而不只是呼叫對應 MQCONNX 的執行緒。 MQDISC 會終止 Hconn ，讓所有執行緒都無法使用它。
3. 單一執行緒可以循序使用多個共用 Hconn ，例如使用 MQPUT 將一個訊息放在一個共用 Hconn 之下，然後使用另一個共用 Hconn 來放置另一個訊息，每一個作業都在不同的本端工作單元之下。
4. 共用 Hconns 無法在廣域工作單元內使用。

Multi 搭配 *MQ_CONNECT_TYPE* 使用 *MQCONNX* 呼叫選項

使用此資訊可瞭解不同的 MQCONNX 呼叫選項以及它們如何與 **MQ_CONNECT_TYPE** 環境變數搭配使用。

註: MQ_CONNECT_TYPE 只對 STANDARD 連結有任何作用。 對於其他連結，會忽略 **MQ_CONNECT_TYPE** 。

在 IBM MQ for Multiplatforms 上，您可以將環境變數 **MQ_CONNECT_TYPE** 與 MQCONNX 呼叫所用 MQCNO 結構的 Options 欄位中指定的連結類型一起使用。

表 *113: MQCONNX* 呼叫選項如何與 ***MQ_CONNECT_TYPE*** 環境變數搭配使用

MQCONNX 呼叫選項	**MQ_CONNECT_TYPE 環境變數**	**結果**
STANDARD	未定義	STANDARD
STANDARD	STANDARD	STANDARD
STANDARD	Fastpath	STANDARD
STANDARD	用戶端	用戶端
STANDARD	本端	STANDARD

如果未指定 MQCNO_STANDARD_BINDING ，您可以使用 MQCNO_NONE ，其預設為 MQCNO_STANDARD_BINDING。

MQCONN 及 *MQCONNX* 的鑑別及身分

使用此作業來瞭解應用程式如何提供在連接至 IBM MQ 時用於鑑別的認證。

預設使用者身分

當應用程式使用訊息佇列介面 (MQI) 來連接至具有 MQCONN 或 MQCONNX 的 IBM MQ 時，一律會建立使用者身分並與連線相關聯。

依預設，起始使用者身分一律是應用程式執行所在之作業系統程序的使用者身分。 對於本端連結或授信應用程式連線而言，這個起始身分可能已足夠。

當應用程式使用 MQCONN 呼叫連接至佇列管理程式時，應用程式無法修改預設使用者 ID。 不過，下列機制可以變更與連線相關聯的使用者 ID:

- 用戶端或伺服器端安全結束程式。
- 佇列管理程式上的通道鑑別規則。
- 在 TLS 交互鑑別期間建立的用戶端使用者 ID。

使用 MQCONNX 來提供認證

MQCONNX 可讓應用程式更能控制與連線相關聯的身分。 應用程式可以將 MQCSP 結構作為 **ConnectOpts** 參數中指定的連接選項的一部分提供給 MQCONNX。 MQCSP 結構可以包含用來建立使用者身分的認證。 IBM MQ 支援 MQCSP 結構中的下列認證:

- 使用者 ID 和密碼。
- V 9.3.4 從 IBM MQ 9.3.4 開始，如果應用程式連接至在 AIX 或 Linux 系統上執行的佇列管理程式，則為鑑別記號。

佇列管理程式的連線鑑別和通道鑑別配置會控制如何處理應用程式所提供的認證。 例如，此配置會影響下列層面:

- MQCSP 結構中的認證是否已驗證，以及如何驗證。
- MQCSP 結構中認證的使用者 ID 是否對映至另一個使用者 ID。
- 然後是否採用已鑑別使用者作為應用程式的環境定義。

如需連線鑑別的相關資訊，請參閱 連線鑑別。 如需通道鑑別的相關資訊，請參閱通道鑑別記錄。

以 C 撰寫且使用 MQI 的數個範例程式，示範如何使用 MQCSP 結構來提供鑑別認證。 如需相關資訊，請參閱下列範例程式:

- 第 916 頁的『Get 範例程式』
- 第 927 頁的『放置範例程式』
- 第 905 頁的『瀏覽器範例程式』
- 第 941 頁的『TLS 範例程式』

相關資訊

使用 MQCSP 結構來識別及鑑別使用者
MQCSP-安全參數
識別及鑑別使用者

使用 *MQDISC* 切斷程式與佇列管理程式的連線

使用此資訊來瞭解如何使用 MQDISC 切斷程式與佇列管理程式的連線。

當已使用 MQCONN 或 MQCONNX 呼叫連接至佇列管理程式的程式已完成與佇列管理程式的所有互動時，它會中斷使用 MQDISC 呼叫的連線，但下列情況除外:

- 在 CICS Transaction Server for z/OS 應用程式上，除非使用 MQCONNX 且您想要在應用程式結束之前捨棄連線標籤，否則呼叫是選用的。
- 在 IBM MQ for IBM i 上，當您從作業系統登出時，會發出隱含的 MQDISC 呼叫。

作為 MQDISC 呼叫的輸入，您必須提供 MQCONN 或 MQCONNX 在連接至佇列管理程式時所傳回的連線控點 (Hconn)。

除了 z/OS 上的 CICS 之外，在呼叫 MQDISC 之後，連線控點 (Hconn) 不再有效，您必須等到重新呼叫 MQCONN 或 MQCONNX 之後，才能發出任何進一步的 MQI 呼叫。 MQDISC 會對仍使用此控點開啟的任何物件執行隱含 MQCLOSE。

z/OS 對於連接至 z/OS 的用戶端，當發出 MQDISC 呼叫時，會進行隱含確定，但在通道實際結束之前，不會關閉任何仍開啟的佇列控點。

如果您在 IBM MQ for z/OS 上使用 MQCONNX 進行連接， MQDISC 也會結束 MQCONNX 所建立的連線標籤範圍。 不過，在 CICS、 IMS 或 RRS 應用程式中，如果有作用中的回復單元與連線標籤相關聯，則會拒絕 MQDISC ，原因碼為 MQRC_CONN_TAG_NOT_RELEARED。

參數的說明在 MQDISC 中 MQDISC 呼叫的說明中提供。

當未發出 MQDISC 時

當建立執行緒終止時，會清除標準非共用連線 (Hconn)。 只有在整個程序終止時，才會隱含地取消及中斷共用連線。 如果建立共用 Hconn 的執行緒在 Hconn 仍然存在時終止，則 Hconn 仍可使用。

權限檢查

MQCLOSE 及 MQDISC 呼叫通常不會執行權限檢查。

在正常事件過程中，有權開啟或連接至 IBM MQ 物件的工作會關閉或切斷與該物件的連線。 即使撤銷已連接或開啟 IBM MQ 物件之工作的權限，也會接受 MQCLOSE 及 MQDISC 呼叫。

開啟及關閉物件

此資訊提供開啟及關閉 IBM MQ 物件的見解。

若要執行下列任何作業，您必須先 開啟 相關的 IBM MQ 物件:

- 將訊息放置在佇列上
- 從佇列取得 (瀏覽或擷取) 訊息
- 設定物件的屬性
- 查詢任何物件的屬性

使用 MQOPEN 呼叫來開啟物件，並使用呼叫的選項來指定您要對物件執行的動作。 唯一例外是如果您要將單一訊息放置在佇列上，然後立即關閉佇列。 在此情況下，您可以使用 MQPUT1 呼叫來略過 開啟 階段 (請參閱 第 637 頁的『使用 MQPUT1 呼叫將一則訊息放置在佇列上』)。

在使用 MQOPEN 呼叫開啟物件之前，您必須將程式連接至佇列管理程式。 對於 第 616 頁的『連接至佇列管理程式並切斷與佇列管理程式的連線』中的所有環境，會詳細說明這一點。

您可以開啟四種類型的 IBM MQ 物件:

- 佇列
- 名稱清單
- 程序定義
- 佇列管理程式

您可以使用 MQOPEN 呼叫以類似方式開啟所有這些物件。 如需 IBM MQ 物件的相關資訊，請參閱 物件類型。

您可以多次開啟相同的物件，且每次取得新的物件控點時都可以開啟。 您可能想要使用一個控點來瀏覽佇列上的訊息，並使用另一個控點從相同佇列中移除訊息。 這會節省使用資源來關閉並重新開啟相同的物件。 您也可以開啟佇列以同時瀏覽 及 移除訊息。

此外，您可以使用單一 MQOPEN 來開啟多個物件，並使用 MQCLOSE 來關閉它們。 如需如何執行此動作的相關資訊，請參閱 第 638 頁的『分送清單』 。

當您嘗試開啟物件時，佇列管理程式會針對您在 MQOPEN 呼叫中指定的選項，檢查您是否已獲授權開啟該物件。

當程式與佇列管理程式中斷連線時，會自動關閉物件。 在 IMS 環境中，當程式在 GU (取得唯一) IMS 呼叫之後開始處理新使用者時，會強制斷線。 在 IBM i 平台上，當工作結束時，會自動關閉物件。

關閉您已開啟的物件是很好的程式設計作法。 請使用 MQCLOSE 呼叫來執行此動作。

請使用下列鏈結，以進一步瞭解開啟及關閉物件的相關資訊:

- 第 623 頁的『使用 MQOPEN 呼叫開啟物件』
- 第 628 頁的『建立動態佇列』
- 第 629 頁的『開啟遠端佇列』
- 第 629 頁的『使用 MQCLOSE 呼叫來關閉物件』

相關概念

第 604 頁的『訊息佇列介面概觀』
瞭解「訊息佇列介面 (MQI)」元件。

第 616 頁的『連接至佇列管理程式並切斷與佇列管理程式的連線』
若要使用 IBM MQ 程式設計服務，程式必須具有與佇列管理程式的連線。 使用此資訊來瞭解如何連接至佇列管理程式，以及與佇列管理程式中斷連線。

第 630 頁的『將訊息放置在佇列上』
使用此資訊來瞭解如何將訊息放置在佇列上。

第 643 頁的『從佇列取得訊息』
使用此資訊來瞭解從佇列取得訊息的相關資訊。

第 713 頁的『查詢及設定物件屬性』
屬性是定義 IBM MQ 物件性質的內容。

第 716 頁的『確定及取消工作單元』

此資訊說明如何確定及取消工作單元中發生的任何可回復取得及放置作業。

第 725 頁的『使用觸發程式啟動 IBM MQ 應用程式』
瞭解觸發程式以及如何使用觸發程式來啟動 IBM MQ 應用程式。

第 741 頁的『使用 MQI 及叢集』
對於與叢集作業相關的呼叫和回覆碼，有一些特殊選項。

第 745 頁的『在 IBM MQ for z/OS 上使用及撰寫應用程式』
IBM MQ for z/OS 應用程式可以由在許多不同環境中執行的程式組成。 這表示他們可以利用多個環境中可用的設施。

第 57 頁的『IBM MQ for z/OS 上的 IMS 及 IMS 橋接器應用程式』
此資訊可協助您使用 IBM MQ 撰寫 IMS 應用程式。

使用 *MQOPEN* 呼叫開啟物件

使用此資訊來瞭解如何使用 MQOPEN 呼叫開啟物件。

作為 MQOPEN 呼叫的輸入，您必須提供:

- 連線控點。 對於 z/OS 上的 CICS 應用程式，您可以指定常數 MQHC_DEF_HCONN (值為零)，或使用 MQCONN 或 MQCONNX 呼叫所傳回的連線控點。 對於其他程式，一律使用 MQCONN 或 MQCONNX 呼叫傳回的連線控點。
- 您要使用物件描述子結構 (MQOD) 開啟之物件的說明。
- 控制呼叫動作的一或多個選項。

MQOPEN 的輸出為:

- 代表您對物件的存取權的物件控點。 在任何後續 MQI 呼叫的輸入中使用此選項。
- 已修改的物件-描述子結構 (如果您正在建立動態佇列，且在您的平台上受支援)。
- 完成碼。
- 原因碼。

物件控點的範圍

物件控點 (Hobj) 的範圍與連線控點 (Hconn) 的範圍相同。

這涵蓋在 第 618 頁的『MQCONN 或 MQCONNX 的範圍』 和 第 619 頁的『與 MQCONNX 共用 (與執行緒無關) 連線』 中。 不過，在某些環境中還有其他考量:

CICS
在 CICS 程式中，您只能在發出 MQOPEN 呼叫的相同 CICS 作業內使用控點。

IMS 和 z/OS 批次
在 IMS 和批次環境中，您可以在相同作業內使用控點，但不能在任何子作業內使用。

MQOPEN 中提供 MQOPEN 呼叫參數的說明。

下列各節說明您必須提供作為 MQOPEN 輸入的資訊。

識別物件 (MQOD 結構)

使用 MQOD 結構來識別您要開啟的物件。 此結構是 MQOPEN 呼叫的輸入參數。 (當您使用 MQOPEN 呼叫來建立動態佇列時，佇列管理程式會修改此結構。)

如需 MQOD 結構的完整資料，請參閱 MQOD。

如需將 MQOD 結構用於配送清單的相關資訊，請參閱 第 638 頁的『分送清單』 下的 第 639 頁的『使用 MQOD 結構』 。

名稱解析
MQOPEN 呼叫如何解析佇列及佇列管理程式名稱。

註: 佇列管理程式別名是不含 RNAME 欄位的遠端佇列定義。

當您開啟 IBM MQ 佇列時， MQOPEN 呼叫會對您指定的佇列名稱執行名稱解析函數。 這會決定佇列管理程式對哪個佇列執行後續作業。 這表示當您在物件描述子 (MQOD) 中指定別名佇列或遠端佇列的名稱時，呼叫會將名稱解析為本端佇列或傳輸佇列。 如果針對任何類型的輸入、瀏覽或設定開啟佇列，則它會解析為本端佇列 (如果有的話) ，如果沒有的話則會失敗。 只有在開啟非本端佇列僅用於輸出、僅用於查詢或僅用於輸出及僅用於查詢時，它才會解析為非本端佇列。 如需名稱解析程序的概觀，請參閱 第 624 頁的表 114 。 您在 *ObjectQMgrName* 中提供的名稱會在 *ObjectName* 中 之前 解析。

第 624 頁的表 114 也顯示如何使用遠端佇列的本端定義，來定義佇列管理程式名稱的別名。 這可讓您選取在遠端佇列上放置訊息時要使用的傳輸佇列，因此您可以使用單一傳輸佇列，例如，針對指定給許多遠端佇列管理程式的訊息使用單一傳輸佇列。

若要使用下表，請先閱讀標題 **MQOD 的輸入**下的兩個左側直欄，然後選取適當的觀察值。 然後遵循任何指示，在對應列中讀取。 遵循 **解析的名稱** 直欄中的指示，您可以回到 **MQOD 輸入** 直欄並依指示插入值，或者您可以使用提供的結果來結束表格。 例如，您可能需要輸入 *ObjectName*。

表 *114:* 使用 *MQOPEN* 時解析佇列名稱

MQOD 的輸入	**MQOD 的輸入**	**已解析的名稱**	**已解析的名稱**	**已解析的名稱**
ObjectQMgrName	*ObjectName*	*ObjectQMgrName*	*ObjectName*	Transmission queue
空白或本端佇列管理程式	沒有 CLUSTER 屬性的本端佇列	本端佇列管理程式	輸入 *ObjectName*	不適用 (使用本端佇列)
空白佇列管理程式	「具有 CLUSTER 的本端佇列」屬性	工作量管理已選取的叢集佇列管理程式或在 PUT 上選取的特定叢集佇列管理程式	輸入 *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE 及本端佇列 SYSTEM.QSG.TRANSMIT.QUEUE (請參閱附註)
本端佇列管理程式	「具有 CLUSTER 的本端佇列」屬性	本端佇列管理程式	輸入 *ObjectName*	不適用 (使用本端佇列)
空白或本端佇列管理程式	模型佇列	本端佇列管理程式	產生的名稱	不適用 (使用本端佇列)
空白或本端佇列管理程式	具有或不具有 CLUSTER 屬性的別名佇列	在 *ObjectQMgrName* 不變的情況下重新執行名稱解析，並在別名佇列定義物件中輸入 *ObjectName* 設為 *BaseQName* 。 不得解析為本端定義的別名 (在其中指定 *ObjectQMgr* 名稱) ，但可以解析為叢集別名 (在其他佇列管理程式上管理) ，其中 *ObjectQMgr* 名稱 為空白。		
本端佇列管理程式	「具有 CLUSTER 的別名佇列」屬性	別名不能解析為未在本端定義的叢集佇列，或具有與別名相同的 *ObjectName* 叢集佇列。		
空白佇列管理程式	「具有 CLUSTER 的別名佇列」屬性	別名可以解析為與別名具有相同 *ObjectName* 的叢集佇列。		

表 *114:* 使用 *MQOPEN* 時解析佇列名稱 *(*繼續*)*

MQOD 的輸入	**MQOD 的輸入**	**已解析的名稱**	**已解析的名稱**	**已解析的名稱**
空白或本端佇列管理程式	遠端佇列的本端定義 (local definition of a remote queue)	在 *ObjectQMgr* 名稱 設為 *RemoteQMgr* 名稱且 *ObjectName* 設為 *RemoteQName* 的情況下，重新執行名稱解析。 不得解析遠端佇列		*XmitQName* 屬性的名稱 (如果非空白); 否則遠端佇列定義物件中的 *RemoteQMgrName* 。 SYSTEM.QSG.TRANSMIT.QUEUE (請參閱附註)
空白佇列管理程式	找不到相符的本端物件; 找到叢集佇列	工作量管理已選取的叢集佇列管理程式或在 PUT 上選取的特定叢集佇列管理程式	輸入 *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEM.QSG.TRANSMIT.QUEUE (請參閱附註)
空白或本端佇列管理程式	沒有相符的本端物件; 找不到叢集佇列		錯誤，找不到佇列	不適用
與本端佇列管理程式位於相同佇列共用群組中的佇列管理程式名稱	本端共用佇列	本端佇列管理程式	輸入 *ObjectName*	不適用
本端傳輸佇列的名稱	(未解析)	輸入 *ObjectQMgr* 名稱	輸入 *ObjectName*	輸入 *ObjectQMgr* 名稱 SYSTEM.QSG.TRANSMIT.QUEUE (請參閱附註)
佇列管理程式別名定義 (*RemoteQMgr* 名稱 可能是本端佇列管理程式)	(未解析，遠端佇列)	將 *ObjectQMgr* 名稱 設為 *RemoteQMgr* 名稱，重新執行名稱解析。 不得解析為遠端佇列	輸入 *ObjectName*	*XmitQName* 屬性的名稱 (如果非空白); 否則遠端佇列定義物件中的 *RemoteQMgrName* 。 SYSTEM.QSG.TRANSMIT.QUEUE (請參閱附註)
佇列管理程式不是任何本端物件的名稱; 找到叢集佇列管理程式或佇列管理程式別名	(未解析)	在 PUT 上選取 *ObjectQMgr* 名稱 或特定的叢集佇列管理程式	輸入 *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEM.QSG.TRANSMIT.QUEUE (請參閱附註)
佇列管理程式不是任何本端物件的名稱; 找不到叢集物件	(未解析)	輸入 *ObjectQMgr* 名稱	輸入 *ObjectName*	支援 *DefXmit* 完整名稱之佇列管理程式的 *DefXmit* 完整名稱 屬性。 SYSTEM.QSG.TRANSMIT.QUEUE (請參閱附註)

附註:

1. *BaseQName* 是別名佇列定義中的基本佇列名稱。
2. *RemoteQName* 是來自遠端佇列本端定義的遠端佇列名稱。
3. *RemoteQMgrName* 是來自遠端佇列本端定義的遠端佇列管理程式名稱。
4. *XmitQName* 是來自遠端佇列本端定義的傳輸佇列名稱。
5. 使用屬於佇列共用群組 (QSG) 的 IBM MQ for z/OS 佇列管理程式時，可以使用佇列共用群組的名稱，而非 第 624 頁的表 114 中的本端佇列管理程式名稱。

如果本端佇列管理程式無法開啟目標佇列，或將訊息放入佇列，則會透過內部群組佇列作業或 IBM MQ 通道，將訊息傳送至指定的 ObjectQMgr 名稱。

6. 在表格的 *ObjectName* 直欄中， CLUSTER 是指佇列的 CLUSTER 及 CLUSNL 屬性。
7. SYSTEM.QSG.TRANSMIT.QUEUE ; 啟用內部群組佇列作業。
8. 如果您已將不同的叢集傳輸佇列指派給每一個叢集傳送端通道，則為 SYSTEM.CLUSTER.TRANSMIT.QUEUE 可能不是叢集傳輸佇列的名稱。 如需多個叢集傳輸佇列的相關資訊，請參閱 叢集作業: 規劃如何配置叢集傳輸佇列。
9. 在佇列管理程式不是任何本端物件的名稱; 找到叢集佇列管理程式或佇列管理程式別名的情況下。

   當您已使用「**ObjectQMgrName**」提供佇列管理程式名稱，且本端佇列管理程式已知有多個叢集名稱不同的叢集通道會到達該目的地時，不論目的地佇列的叢集名稱為何，都可以使用任何這些通道來移動訊息。

   如果您預期該佇列的訊息只會透過與佇列具有相同叢集名稱的通道來傳送，這可能是非預期的。

   不過，在此情況下， **ObjectQMgrName** 優先，而且叢集工作量平衡會將可能到達該佇列管理程式的所有通道納入考量，而不論它們位於哪個叢集名稱中。

開啟別名佇列也會開啟別名所解析成的基本佇列，開啟遠端佇列也會開啟傳輸佇列。 因此，當另一個佇列開啟時，您無法刪除您指定的佇列或它解析成的佇列。

雖然別名佇列無法解析為另一個本端定義的別名佇列 (是否在叢集中共用) ，但允許解析為遠端定義的叢集別名佇列，因此可以指定為基本佇列。

已解析的佇列名稱及已解析的佇列管理程式名稱儲存在 MQOD 中的 *ResolvedQName* 及 *ResolvedQMgrName* 欄位中。

如需分散式佇列環境中名稱解析的相關資訊，請參閱 何謂佇列名稱解析?。

使用 *MQOPEN* 呼叫的選項

在 MQOPEN 呼叫的 **Options** 參數中，您必須選擇一或多個選項，以控制對所開啟物件提供的存取權。 使用這些選項，您可以:

- 開啟佇列，並指定放入該佇列的所有訊息必須導向至該佇列的相同實例
- 開啟佇列以容許您在其中放置訊息
- 開啟佇列以容許您瀏覽其中的訊息
- 開啟佇列以容許您移除其中的訊息
- 開啟物件以容許您查詢並設定其屬性 (但您只能設定佇列的屬性)
- 開啟主題或主題字串以將訊息發佈至其中
- 建立環境定義資訊與訊息的關聯
- 指定要用於安全檢查的替代使用者 ID
- 如果佇列管理程式處於靜止狀態，則控制呼叫

叢集佇列的 *MQOPEN* 選項

用於佇列控點的連結取自 **DefBind** 佇列屬性，其可以採用值 MQBND_BIND_ON_OPEN、MQBND_BIND_NOT_FIXED 或 MQBND_BIND_ON_GROUP。

若要透過相同的路徑，將所有使用 MQPUT 放入佇列的訊息遞送至相同的佇列管理程式，請在 MQOPEN 呼叫上使用 MQOO_BIND_ON_OPEN 選項。

若要指定在 MQPUT 時選取目的地 (即以訊息為基礎) ，請在 MQOPEN 呼叫上使用 MQOO_BIND_NOT_FIXED 選項。

若要指定將 訊息群組 中所有使用 MQPUT 放入佇列的訊息配置給相同的目的地實例，請在 MQOPEN 呼叫上使用 MQOO_BIND_ON_GROUP 選項。

當搭配使用 訊息群組 與叢集時，必須指定 MQOO_BIND_ON_OPEN 或 MQOO_BIND_ON_GROUP ，以確保在相同目的地處理群組中的所有訊息。

如果您未指定任何這些選項，則會使用預設值 MQOO_BIND_AS_Q_DEF。

如果您在「MQOD」中指定佇列管理程式的名稱，則會選取該佇列管理程式上的佇列。 如果佇列管理程式名稱空白，則可以選取任何實例。 如需相關資訊，請參閱第 742 頁的『MQOPEN 及叢集』。

如果您使用 QALIAS 定義開啟叢集佇列，則部分佇列屬性是由別名佇列而非基本佇列所定義。 叢集屬性是別名佇列所置換之基本佇列定義的屬性之一。 例如，在下列 Snippet 中，叢集佇列是以 MQOO_BIND_NOT_FIXED 開啟，而不是 MQOO_BIND_ON_OPEN 開啟。 叢集佇列定義會在整個叢集中通告，別名佇列定義是在佇列管理程式本端。

```
DEFINE QLOCAL(CLQ1) CLUSTER(MYCLUSTER) DEFBIND(OPEN) REPLACE
DEFINE QALIAS(ACLQ1) TARGET(CLQ1) DEFBIND(NOTFIXED) REPLACE
```

用於放置訊息的 *MQOPEN* 選項
若要開啟佇列或主題以放置訊息，請使用 MQOO_OUTPUT 選項。

用於瀏覽訊息的 *MQOPEN* 選項
若要開啟佇列，以便您可以 瀏覽 其中的訊息，請搭配使用 MQOPEN 呼叫與 MQOO_BROWSE 選項。

這會建立佇列管理程式用來識別佇列上下一則訊息的 瀏覽游標 。 如需相關資訊，請參閱 第 671 頁的『瀏覽佇列上的訊息』。

註:

1. 您無法瀏覽遠端佇列上的訊息; 請勿使用 MQOO_BROWSE 選項開啟遠端佇列。
2. 您無法在開啟配送清單時指定此選項。 如需配送清單的進一步相關資訊，請參閱 第 638 頁的『分送清單』。
3. 如果您使用協同瀏覽，請將 MQOO_CO_OP 與 MQOO_BROWSE 一起使用; 請參閱 選項

用於移除訊息的 *MQOPEN* 選項
三個選項會控制佇列的開啟，以移除其中的訊息。

您只能在任何 MQOPEN 呼叫中使用其中一個。 這些選項定義您的程式是否具有佇列的專用或共用存取權。 互斥存取 表示在您關閉佇列之前， 只有您可以從中移除訊息。 如果另一個程式嘗試開啟佇列以移除訊息，則其 MQOPEN 呼叫會失敗。 共用存取權 表示多個程式可以移除 來自佇列的訊息。

最建議的方法是接受定義佇列時預期用於佇列的存取類型。 佇列定義涉及 **Shareability** 的設定及 **DefInputOpenOption** 屬性。 若要接受此存取權，請使用 MQOO_INPUT_AS_Q_DEF 選項。 請參閱 第 627 頁的表 115 ，以查看這些屬性的設定如何影響當您使用此選項時將提供給您的存取權類型。

表 *115: MQOPEN* 呼叫的佇列屬性及選項如何影響佇列存取權

「佇列」屬性		**使用 MQOPEN 選項的存取類型**		
Shareability	**DefInputOpenOption**	**AS_Q_DEF**	**SHARED**	**EXCLUSIVE**
可共用	SHARED	共用	共用	專用
可共用	EXCLUSIVE	專用	共用	專用
NOT_SHAREABLE*	共用*	專用	專用	專用
NOT_SHAREABLE	EXCLUSIVE	專用	專用	專用

註: * 雖然您可以定義佇列來具有此屬性組合，但 shareability 屬性會置換預設輸入開啟選項。

或者:

- 如果您知道應用程式可以順利運作，即使其他程式可以同時從佇列中移除訊息，請使用 MQOO_INPUT_SHARED 選項。 第 627 頁的表 115 顯示在某些情況下，即使使用此選項，也如何授與您對佇列的互斥存取權。
- 如果您知道只有在防止其他程式同時從佇列中移除訊息時，應用程式才能順利運作，請使用 MQOO_INPUT_EXCLUSIVE 選項。

註:

1. 您無法從遠端佇列移除訊息。 因此，您無法使用任何 MQOO_Input_ * 選項來開啟遠端佇列。
2. 您無法在開啟配送清單時指定此選項。 如需進一步資訊，請參閱第 638 頁的『分送清單』。

MQOPEN 選項，用於設定及查詢屬性
若要開啟佇列以便您可以設定其屬性，請使用 MQOO_SET 選項。

您無法設定任何其他類型物件的屬性 (請參閱 第 713 頁的『查詢及設定物件屬性』)。

若要開啟物件以查詢其屬性，請使用 MQOO_INQUIRE 選項。

註: 您無法在開啟配送清單時指定此選項。

與訊息環境定義相關的 *MQOPEN* 選項
如果您想要在將環境定義資訊放入佇列時能夠將它與訊息相關聯，則必須在開啟佇列時使用其中一個訊息環境定義選項。

這些選項可讓您區分與訊息起源 使用者 相關的環境定義資訊，以及與訊息起源 應用程式 相關的環境定義資訊。 此外，您可以選擇在將訊息放入佇列時設定環境定義資訊，也可以選擇從另一個佇列控點自動取得環境定義。

相關概念
第 40 頁的『訊息環境定義』
訊息環境定義 資訊可讓擷取訊息的應用程式找出訊息的發送端。

第 635 頁的『控制訊息環境定義資訊』
當您使用 MQPUT 或 MQPUT1 呼叫將訊息放置在佇列上時，您可以指定佇列管理程式將部分預設環境定義資訊新增至訊息描述子。 具有適當權限層次的應用程式可以新增額外環境定義資訊。 您可以使用 MQPMO 結構中的選項欄位來控制環境定義資訊。

替代使用者權限的 *MQOPEN* 選項
當您嘗試使用 MQOPEN 呼叫開啟物件時，佇列管理程式會檢查您是否具有開啟該物件的權限。 如果您未獲授權，則通話會失敗。

不過，伺服器程式可能希望佇列管理程式檢查其工作之使用者的授權，而不是伺服器自己的授權。 若要這樣做，他們必須使用 MQOPEN 呼叫的 MQOO_ALTERNATE_USER_AUTHORITY 選項，並在 MQOD 結構的 *AlternateUserId* 欄位中指定替代使用者 ID。 一般而言，伺服器會從它正在處理的訊息中的環境定義資訊取得使用者 ID。

z/OS 佇列管理程式靜止的 *MQOPEN* 選項
如果您在佇列管理程式處於靜止狀態時使用 MQOPEN 呼叫，則視您使用的環境而定，該呼叫可能會失敗。

在 z/OS 上的 CICS 環境中，如果您在佇列管理程式處於靜止狀態時使用 MQOPEN 呼叫，則呼叫一律會失敗。

在其他 z/OS 及 Multiplatforms 環境中，只有在您使用 MQOPEN 呼叫的 MQOO_FAIL_IF_QUIESCING 選項時，佇列管理程式靜止時呼叫才會失敗。

用於解析本端佇列名稱的 *MQOPEN* 選項
當您開啟本端、別名或模型佇列時，會傳回本端佇列。

不過，當您開啟遠端佇列或叢集佇列時， MQOD 結構的 *ResolvedQName* 及 *ResolvedQMgrName* 欄位會填入在遠端佇列定義中找到的遠端佇列及遠端佇列管理程式名稱，或填入所選擇的遠端叢集佇列。

使用 MQOPEN 呼叫的 MQOO_RESOLVE_LOCAL_Q 選項，以開啟的本端佇列名稱填入 MQOD 結構中的 *ResolvedQName* 。 同樣地， *ResolvedQMgrName* 會填入管理本端佇列的本端佇列管理程式名稱。 此欄位僅適用於 MQOD 結構第 3 版; 如果結構小於第 3 版，則會忽略 MQOO_RESOLVE_LOCAL_Q ，而不會傳回錯誤。

如果您在開啟 (例如遠端佇列) 時指定 MQOO_RESOLVE_LOCAL_Q ，則 *ResolvedQName* 是將放置訊息的傳輸佇列名稱。 *ResolvedQMgrName* 是管理傳輸佇列的本端佇列管理程式名稱。

建立動態佇列
當您在應用程式結束之後不需要佇列時，請使用動態佇列。

例如，您可以使用動態佇列作為回覆目的地佇列。 將訊息放入佇列時，您可以在 MQMD 結構的 *ReplyToQ* 欄位中指定回覆目的地佇列的名稱 (請參閱 第 631 頁的『使用 MQMD 結構定義訊息』)。

若要建立動態佇列，您可以搭配使用稱為模型佇列的範本與 MQOPEN 呼叫。 您可以使用 IBM MQ 指令或作業及控制台來建立模型佇列。 您建立的動態佇列會採用模型佇列的屬性。

當您呼叫 MQOPEN 時，請在 MQOD 結構的 *ObjectName* 欄位中指定模型佇列的名稱。 當呼叫完成時，*ObjectName* 欄位會設為所建立動態佇列的名稱。 此外， *ObjectQMgrName* 欄位會設為本端佇列管理程式的名稱。

您可以用三種方式來指定您建立的動態佇列名稱:

- 在 MQOD 結構的 *DynamicQName* 欄位中提供您想要的完整名稱。
- 請指定名稱的字首 (少於 33 個字元) ，並容許佇列管理程式產生剩餘的名稱。 這表示佇列管理程式會產生唯一名稱，但您仍有一些控制權 (例如，您可能想要每一個使用者使用特定字首，或您可能想要對其名稱中具有特定字首的佇列提供特殊安全分類)。 若要使用此方法，請對 *DynamicQName* 欄位的最後一個非空白字元指定星號 (*)。 請勿對動態佇列名稱指定單一星號 (*)。
- 容許佇列管理程式產生完整名稱。 若要使用此方法，請在 *DynamicQName* 欄位的第一個字元位置指定星號 (*)。

如需這些方法的相關資訊，請參閱 DynamicQName 欄位的說明。

動態和模型佇列中有動態佇列的相關資訊。

開啟遠端佇列

遠端佇列是佇列管理程式所擁有的佇列，不是應用程式所連接的佇列管理程式。

若要開啟遠端佇列，請針對本端佇列使用 MQOPEN 呼叫。 您可以指定佇列的名稱，如下所示:

1. 在 MQOD 結構的 *ObjectName* 欄位中，指定 本端 佇列管理程式已知的遠端佇列名稱。

   註: 在此情況下，請將 *ObjectQMgrName* 欄位保留空白。
2. 在 MQOD 結構的 *ObjectName* 欄位中，指定 遠端 佇列管理程式已知的遠端佇列名稱。 在 *ObjectQMgrName* 欄位中，指定下列任一項:
   - 與遠端佇列管理程式同名的傳輸佇列名稱。 名稱和大小寫 (大寫、小寫或混合) 必須 完全符合。
   - 解析為目的地佇列管理程式或傳輸佇列的佇列管理程式別名物件名稱。

   這會告知佇列管理程式訊息的目的地，以及需要放置它才能到達的傳輸佇列。
3. 如果支援 *DefXmitQname* ，請在 MQOD 結構的 *ObjectName* 欄位中，指定 遠端 佇列管理程式已知的遠端佇列名稱。

   註: 將 *ObjectQMgrName* 欄位設為遠端佇列管理程式的名稱 (在此情況下不能保留空白)。

當您呼叫 MQOPEN 時，只會驗證本端名稱; 最後一項檢查是要使用的傳輸佇列是否存在。

這些方法在第 624 頁的表 114 中彙總。

使用 *MQCLOSE* 呼叫來關閉物件

若要關閉物件，請使用 MQCLOSE 呼叫。

如果物件是佇列，請注意下列事項:

- 在關閉暫時動態佇列之前，您不需要先清空它。

  當您關閉暫時動態佇列時，會刪除佇列，以及可能仍在其中的任何訊息。 即使有未確定的 MQGET、MQPUT 或 MQPUT1 呼叫未針對佇列執行，也是如此。
- 在 IBM MQ for z/OS 上，如果您有任何 MQGET 要求針對該佇列具有未完成的 MQGMO_SET_信號選項，則會取消這些要求。
- 如果您使用 MQOO_BROWSE 選項開啟佇列，則會毀損您的瀏覽游標。

關閉與同步點無關，因此您可以在同步點之前或之後關閉佇列。

作為 MQCLOSE 呼叫的輸入，您必須提供:

- 連線控點。 使用用來開啟它的相同連線控點，或者針對 z/OS 上的 CICS 應用程式，您可以指定常數 MQHC_DEF_HCONN (其值為零)。
- 您要關閉之物件的控點。 從 MQOPEN 呼叫的輸出中取得此項目。
- *Options* 欄位中的 MQCO_NONE (除非您要關閉永久動態佇列)。
- 此控制選項可判定佇列管理程式是否應該刪除佇列，即使佇列上仍有訊息 (當關閉永久動態佇列時)。

MQCLOSE 的輸出為:

- 完成碼
- 原因碼
- 物件控點，重設為值 MQHO_UNUSABLE_HOBJ

MQCLOSE 中提供 MQCLOSE 呼叫參數的說明。

將訊息放置在佇列上

使用此資訊來瞭解如何將訊息放置在佇列上。

使用 MQPUT 呼叫將訊息放置在佇列上。 在起始 MQOPEN 呼叫之後，您可以反覆地使用 MQPUT ，將許多訊息放置在相同佇列上。 當您完成將所有訊息放入佇列時，請呼叫 MQCLOSE。

如果您要將單一訊息放置在佇列上，然後立即關閉佇列，則可以使用 MQPUT1 呼叫。 MQPUT1 會執行與下列呼叫序列相同的功能:

- MQOPEN
- MQPUT
- MQCLOSE

不過，一般而言，如果您有多則訊息要放置在佇列上，則使用 MQPUT 呼叫會更有效率。 這取決於訊息的大小以及您正在使用的平台。

請使用下列鏈結，以進一步瞭解將訊息放入佇列的相關資訊:

- 第 631 頁的『使用 MQPUT 呼叫將訊息放置在本端佇列上』
- 第 635 頁的『將訊息放置在遠端佇列上』
- 第 635 頁的『設定訊息的內容』
- 第 635 頁的『控制訊息環境定義資訊』
- 第 637 頁的『使用 MQPUT1 呼叫將一則訊息放置在佇列上』
- 第 638 頁的『分送清單』
- 第 642 頁的『部分放置呼叫失敗的案例』

相關概念

第 604 頁的『訊息佇列介面概觀』
瞭解「訊息佇列介面 (MQI)」元件。

第 616 頁的『連接至佇列管理程式並切斷與佇列管理程式的連線』
若要使用 IBM MQ 程式設計服務，程式必須具有與佇列管理程式的連線。 使用此資訊來瞭解如何連接至佇列管理程式，以及與佇列管理程式中斷連線。

第 621 頁的『開啟及關閉物件』
此資訊提供開啟及關閉 IBM MQ 物件的見解。

第 643 頁的『從佇列取得訊息』
使用此資訊來瞭解從佇列取得訊息的相關資訊。

第 713 頁的『查詢及設定物件屬性』
屬性是定義 IBM MQ 物件性質的內容。

第 716 頁的『確定及取消工作單元』
此資訊說明如何確定及取消工作單元中發生的任何可回復取得及放置作業。

第 725 頁的『使用觸發程式啟動 IBM MQ 應用程式』
瞭解觸發程式以及如何使用觸發程式來啟動 IBM MQ 應用程式。

第 741 頁的『使用 MQI 及叢集』
對於與叢集作業相關的呼叫和回覆碼，有一些特殊選項。

第 745 頁的『在 IBM MQ for z/OS 上使用及撰寫應用程式』
IBM MQ for z/OS 應用程式可以由在許多不同環境中執行的程式組成。 這表示他們可以利用多個環境中可用的設施。

第 57 頁的『IBM MQ for z/OS 上的 IMS 及 IMS 橋接器應用程式』
此資訊可協助您使用 IBM MQ 撰寫 IMS 應用程式。

使用 *MQPUT* 呼叫將訊息放置在本端佇列上

使用此資訊來瞭解如何使用 MQPUT 呼叫將訊息放置在本端佇列上。

作為 MQPUT 呼叫的輸入，您必須提供:

- 連線控點 (Hconn)。
- 佇列控點 (Hobj)。
- 您要放置在佇列上的訊息說明。 這是訊息描述子結構 (MQMD) 的形式。
- 控制資訊，採用放置訊息選項結構 (MQPMO) 的形式。
- 訊息內包含的資料長度 (MQLONG)。
- 訊息資料本身。

MQPUT 呼叫的輸出如下:

- 原因碼 (MQLONG)
- 完成碼 (MQLONG)

如果呼叫順利完成，它也會傳回您的選項結構及訊息描述子結構。 該呼叫會修改您的選項結構，以顯示佇列名稱及訊息傳送至其中的佇列管理程式。 如果您要求佇列管理程式為您所放置訊息的 ID 產生唯一值 (透過在 MQMD 結構的 *MsgId* 欄位中指定二進位零)，則呼叫會在 *MsgId* 欄位中插入該值，然後再將此結構傳回給您。 請重設此值，然後再發出另一個 MQPUT。

MQPUT 中有 MQPUT 呼叫的說明。

如需 MQPUT 呼叫輸入所需資訊的相關說明，請參閱下列鏈結:

- 第 631 頁的『指定控點』
- 第 631 頁的『使用 MQMD 結構定義訊息』
- 第 632 頁的『使用 MQPMO 結構指定選項』
- 第 634 頁的『訊息中的資料』
- 第 635 頁的『放置訊息: 使用訊息控點』

指定控點

對於 z/OS 應用程式上 CICS 中的連線控點 (*Hconn*)，您可以指定常數 MQHC_DEF_HCONN (其值為零)，也可以使用 MQCONN 或 MQCONNX 呼叫所傳回的連線控點。 對於其他應用程式，一律使用 MQCONN 或 MQCONNX 呼叫傳回的連線控點。

不論您在何種環境中工作，請使用 MQOPEN 呼叫所傳回的相同佇列控點 (*Hobj*)。

使用 MQMD 結構定義訊息

訊息描述子結構 (MQMD) 是 MQPUT 及 MQPUT1 呼叫的輸入/輸出參數。 請使用它來定義您要放置在佇列上的訊息。

如果針對訊息指定 MQPRI_PRIORITY_AS_Q_DEF 或 MQPER_PERSISTENCE_AS_Q_DEF，且佇列是叢集佇列，則使用的值是 MQPUT 所解析的佇列值。 如果針對 MQPUT 停用該佇列，則呼叫會失敗。 如需相關資訊，請參閱 配置佇列管理程式叢集 。

註: 在放置新訊息之前使用 MQPMO_NEW_MSG_ID 和 MQPMO_NEW_CORREL_ID，以確保 *MsgId* 和 *CorrelId* 是唯一的。 這些欄位中的值會在成功 MQPUT 時傳回。

MQMD 在 第 16 頁的『IBM MQ 訊息』 中說明之訊息內容的簡介，以及 MQMD 中結構本身的說明。

使用 MQPMO 結構指定選項

使用 MQPMO (放置訊息選項) 結構，將選項傳遞至 MQPUT 及 MQPUT1 呼叫。

下列各節提供您填寫此結構欄位的說明。 MQPMO 中有結構的說明。

結構包括下列欄位:

- *StrucId*
- *Version*
- *Options*
- *Context*
- *ResolvedQName*
- *ResolvedQMgrName*
- *RecsPresent*
- *PutMsgRecsFields*
- *ResponseRecOffset and ResponseRecPtr*
- *OriginalMsgHandle*
- *NewMsgHandle*
- *Action*
- *PubLevel*

這些欄位的內容如下:

StrucId
這會將結構識別為放置訊息選項結構。 這是 4 個字元的欄位。 一律指定 MQPMO_STRUC_ID。

版本
這說明結構的版本號碼。 預設值為 MQPMO_VERSION_1。 如果您輸入 MQPMO_VERSION_2，則可以使用配送清單 (請參閱 第 638 頁的『分送清單』)。 如果您輸入 MQPMO_VERSION_3，則可以使用訊息控點及訊息內容。 如果您輸入 MQPMO_CURRENT_VERSION，則您的應用程式一律設為使用最新層次。

選項
這會控制下列各項:

- 放置作業是否包含在工作單元中
- 與訊息相關聯的環境定義資訊數量
- 環境定義資訊的來源
- 如果佇列管理程式處於靜止狀態，則呼叫是否失敗
- 是否容許分組或分段
- 產生新的訊息 ID 和相關性 ID
- 將訊息及區段放置在佇列上的順序
- 是否解析本端佇列名稱

如果您將 *Options* 欄位設定為預設值 (MQPMO_NONE)，則您放置的訊息具有與其相關聯的預設環境定義資訊。

此外，呼叫以同步點運作的方式由平台決定。 在 z/OS 中，同步點控制項預設值是 yes; 對於其他平台，它不是。

環境定義
這會指出您要從中複製環境定義資訊的佇列控點名稱 (如果在 *Options* 欄位中要求的話)。

如需訊息環境定義的簡介，請參閱 第 40 頁的『訊息環境定義』。 如需使用 MQPMO 結構來控制訊息中環境定義資訊的相關資訊，請參閱 第 635 頁的『控制訊息環境定義資訊』。

ResolvedQName
這包含為了接收訊息而開啟的佇列名稱 (在解析任何別名之後)。 這是輸出欄位。

ResolvedQMgr 名稱
這包含在 *ResolvedQName* 中擁有佇列之佇列管理程式的名稱 (解析任何別名之後)。 這是輸出欄位。

MQPMO 也可以容納配送清單所需的欄位 (請參閱 第 638 頁的『分送清單』)。 如果您想要使用此機能，則會使用 MQPMO 結構第 2 版。 這包括下列欄位:

RecsPresent
此欄位包含配送清單中的佇列數目; 亦即，呈現的「放置訊息記錄 (MQPMR)」及對應的「回應記錄 (MQRR)」數目。

您輸入的值可以與 MQOPEN 提供的「物件記錄」數目相同。 不過，如果值小於 MQOPEN 呼叫上提供的「物件記錄」數目，或您未提供「放置訊息記錄」，則會從訊息描述子提供的預設值取得未定義的佇列值。 此外，如果值大於提供的「物件記錄」數目，則會忽略多餘的「放置訊息記錄」。

建議您執行下列其中一項:

- 如果您想要從每一個目的地接收報告或回覆，請輸入 MQOR 結構中出現的相同值，並使用包含 *MsgId* 欄位的 MQPMR。 將這些 *MsgId* 欄位起始設定為零或指定 MQPMO_NEW_MSG_ID。

  當您將訊息放入佇列時，佇列管理程式所建立的 *MsgId* 值會在 MQPMR 中變成可用; 您可以使用這些值來識別與每一個報告或回覆相關聯的目的地。
- 如果您不想接收報告或回覆，請選擇下列其中一項:
  1. 如果您想要識別立即失敗的目的地，您可能仍想要在 *RecsPresent* 欄位中輸入與 MQOR 結構中顯示的相同值，並提供 MQRR 以識別這些目的地。 請勿指定任何 MQPMR。
  2. 如果您不想識別失敗的目的地，請在 *RecsPresent* 欄位中輸入零，並且不提供 MQPMR 或 MQRR。

註: 如果您使用 MQPUT1，「回應記錄指標」及「回應記錄偏移」的數目必須為零。

如需「放置訊息記錄 (MQPMR)」及「回應記錄 (MQRR)」的完整說明，請參閱 MQPMR 及 MQRR。

PutMsgRecFields
這指出每一個「放置訊息記錄 (MQPMR)」中存在哪些欄位。 如需這些欄位的清單，請參閱 第 641 頁的『使用 MQPMR 結構』。

PutMsgRecOffset* 及 *PutMsgRecPtr
指標 (通常在 C 中) 和偏移 (通常在 COBOL 中) 用來處理「放置訊息記錄」(如需 MQPMR 結構的概觀，請參閱 第 641 頁的『使用 MQPMR 結構』)。

使用 *PutMsgRecPtr* 欄位來指定指向第一個「放置訊息」記錄的指標，或使用 *PutMsgRecOffset* 欄位來指定第一個「放置訊息」記錄的偏移。 這是從 MQPMO 開始的偏移。 視 *PutMsgRecFields* 欄位而定，輸入 *PutMsgRecOffset* 或 *PutMsgRecPtr* 的非空值。

ResponseRec 偏移及 ResponseRecPtr
您也可以使用指標及偏移來處理「回應記錄」(如需「回應記錄」的進一步相關資訊，請參閱 第 640 頁的『使用 MQRR 結構』)。

使用 *ResponseRecPtr* 欄位來指定指向第一個「回應記錄」的指標，或使用 *ResponseRecOffset* 欄位來指定第一個「回應記錄」的偏移。 這是與 MQPMO 結構開頭的偏移。 輸入 *ResponseRecOffset* 或 *ResponseRecPtr* 的非空值。

註: 如果您使用 MQPUT1 將訊息放置到配送清單中，則 *ResponseRecPtr* 必須是空值或零，且 *ResponseRecOffset* 必須是零。

MQPMO 結構第 3 版額外包含下列欄位:

OriginalMsg 控點
您可以使用此欄位，視 動作 欄位的值而定。 如果您要放置具有相關聯訊息內容的新訊息，請將此欄位設為您先前建立並設定內容的訊息控點。 如果您要轉遞、回覆或產生報告以回應先前擷取的訊息，則此欄位包含該訊息的訊息控點。

NewMsg 控點
如果您指定 *NewMsgHandle*，則任何與控點相關聯的內容都會置換與 *OriginalMsgHandle* 相關聯的內容。 如需相關資訊，請參閱 動作 (MQLONG)。

動作
請利用這個欄位來指定要執行的放置類型。 可能的值及其意義如下:

MQACTP_NEW
這是與任何其他訊息無關的新訊息。

MQACTP_FORWARD
先前已擷取此訊息，現在正在轉遞中。

MQACTP_REPLY
此訊息是對先前擷取之訊息的回覆。

MQACTP_REPORT
此訊息是由於先前擷取的訊息而產生的報告。

如需相關資訊，請參閱 動作 (MQLONG)。

PubLevel
如果此訊息是發佈，您可以設定此欄位來決定哪些訂閱會接收它。 只有 *SubLevel* 小於或等於此值的訂閱才會收到此發佈。 預設值是 9 ，這是最高層次，表示具有任何 *SubLevel* 的訂閱可以接收此發佈。

訊息中的資料

在 MQPUT 呼叫的 **Buffer** 參數中提供包含資料之緩衝區的位址。 您可以在訊息的資料中包含任何內容。 不過，訊息中的資料量會影響正在處理它們的應用程式效能。

資料大小上限取決於:

- 佇列管理程式的 **MaxMsgLength** 屬性
- 您要放置訊息之佇列的 **MaxMsgLength** 屬性
- IBM MQ 所新增之任何訊息標頭的大小 (包括無法傳送郵件的標頭、MQDLH 及配送清單標頭 MQDH)

佇列管理程式的 **MaxMsgLength** 屬性會保留佇列管理程式可處理的訊息大小。 對於 V6 或更高版本的所有 IBM MQ 產品，預設值為 100 MB。

若要判定此屬性的值，請對佇列管理程式物件使用 MQINQ 呼叫。 若為大型訊息，您可以變更此值。

佇列的 **MaxMsgLength** 屬性決定您可以放置在佇列上的訊息大小上限。 如果您嘗試放置大小大於此屬性值的訊息，則 MQPUT 呼叫會失敗。 如果您要將訊息放置在遠端佇列上，您可以順利放置的訊息大小上限是由遠端佇列的 **MaxMsgLength** 屬性、訊息放置在其目的地路徑上的任何中間傳輸佇列，以及使用的通道所決定。

對於 MQPUT 作業，訊息大小必須小於或等於佇列及佇列管理程式的 **MaxMsgLength** 屬性。 這些屬性的值是獨立的，但建議您將佇列的 *MaxMsgLength* 設為小於或等於佇列管理程式的值。

在下列情況下， IBM MQ 會將標頭資訊新增至訊息:

- 當您將訊息放入遠端佇列時， IBM MQ 會將傳輸標頭結構 (MQXQH) 新增至訊息。 此結構包括目的地佇列的名稱及其擁有的佇列管理程式。
- 如果 IBM MQ 無法將訊息遞送至遠端佇列，它會嘗試將訊息放置在無法傳送的郵件 (無法遞送的訊息) 佇列上。 它會將 MQDLH 結構新增至訊息。 此結構包括目的地佇列的名稱，以及將訊息放置在無法傳送郵件的佇列上的原因。
- 如果您要將訊息傳送至多個目的地佇列， IBM MQ 會將 MQDH 標頭新增至訊息。 這說明在傳輸佇列上屬於配送清單的訊息中呈現的資料。 在選擇訊息長度上限的最佳值時，請考量此選項。
- 如果訊息是群組中的區段或訊息， IBM MQ 可能會新增 MQMDE。

這些結構在 MQDH 和 MQMDE 中有說明。

如果您的訊息達到這些佇列所容許的大小上限，則新增這些標頭表示放置作業會失敗，因為訊息現在太大。 如果要減少放置作業失敗的可能性，請執行下列動作:

- 使訊息大小小於傳輸及無法傳送郵件的佇列的 **MaxMsgLength** 屬性。 至少容許 MQ_MSG_HEADER_LENGTH 常數的值 (若為大型配送清單，則容許更多值)。
- 請確定無法傳送郵件的佇列 **MaxMsgLength** 屬性設為與擁有無法傳送郵件的佇列之佇列管理程式的 *MaxMsgLength* 相同。

佇列管理程式的屬性中說明佇列管理程式及訊息佇列常數的屬性。

z/OS 如需如何在分散式佇列環境中處理未遞送訊息的相關資訊，請參閱 未遞送/未處理訊息。

放置訊息: 使用訊息控點

MQPMO 結構中有兩個訊息控點: *OriginalMsgHandle* 和 *NewMsgHandle*。 這些訊息控點之間的關係由 MQPMO 動作 欄位的值定義。

如需完整資料，請參閱 動作 (MQLONG)。 放置訊息不一定需要訊息控點。 其目的是將內容與訊息相關聯，因此只有在您使用訊息內容時才需要。

將訊息放置在遠端佇列上

當您想要將訊息放置在遠端佇列 (亦即，佇列管理程式所擁有的佇列，而非應用程式所連接的佇列) 而非本端佇列時，唯一的額外考量是當您開啟佇列時如何指定佇列名稱。這說明於第 629 頁的『開啟遠端佇列』。 對本端佇列使用 MQPUT 或 MQPUT1 呼叫的方式沒有任何變更。

如需使用遠端及傳輸佇列的相關資訊，請參閱 IBM MQ 分散式佇列技術。

設定訊息的內容

針對您要設定的每一個內容，呼叫 MQSETMP。 當您放置訊息時，請設定 MQPMO 結構的訊息控點及動作欄位。

若要將內容與訊息相關聯，訊息必須具有訊息控點。 使用 MQCRTMH 函數呼叫來建立訊息控點。 針對您要設定的每一個內容指定此訊息控點，以呼叫 MQSETMP。 提供範例程式 amqsstma.c 來說明 MQSETMP 的用法。

如果這是新訊息，當您使用 MQPUT 或 MQPUT1 將它放入佇列時，請將 MQPMO 中的 OriginalMsgHandle 欄位設為此訊息控點的值，並將「MQPMO 動作」欄位設為 MQACTP_NEW (這是預設值)。

如果這是您先前擷取的訊息，且您現在正在轉遞或回覆它，或傳送報告以回應它，請將原始訊息控點放置在 MQPMO 的 OriginalMsg 控點欄位中，並將新訊息控點放置在 NewMsg 控點欄位中。 視需要將「動作」欄位設為 MQACTP_FORWARD、MQACTP_REPLY 或 MQACTP_REPORT。

如果您在 MQRFH2 標頭中具有先前所擷取訊息的內容，則可以使用 MQBUFMH 呼叫將它們轉換為訊息控點內容。

如果您要將訊息放入佇列管理程式中早於 IBM WebSphere MQ 7.0 的層次 (無法處理訊息內容)，則可以在通道定義中設定 PropertyControl 參數，以指定如何處理內容。

控制訊息環境定義資訊

當您使用 MQPUT 或 MQPUT1 呼叫將訊息放置在佇列上時，您可以指定佇列管理程式將部分預設環境定義資訊新增至訊息描述子。 具有適當權限層次的應用程式可以新增額外環境定義資訊。 您可以使用 MQPMO 結構中的選項欄位來控制環境定義資訊。

訊息環境定義資訊可讓擷取訊息的應用程式找出訊息的發送端。 所有環境定義資訊都儲存在訊息描述子的環境定義欄位中。 資訊的類型落在身分、原點及使用者環境定義資訊中。

若要控制環境定義資訊，請使用 MQPMO 結構中的 *Options* 欄位。

如果您沒有指定環境定義資訊的任何選項，佇列管理程式會以它為您的訊息所產生的身分和環境定義資訊，來改寫可能已在訊息描述子中的環境定義資訊。 這與指定 MQPMO_DEFAULT_CONTEXT 選項相同。當您建立新訊息時 (例如，從查詢畫面處理使用者輸入時)，您可能需要此預設環境定義資訊。

如果您不想要與訊息相關聯的環境定義資訊，請使用 MQPMO_NO_CONTEXT 選項。 當放置不含環境定義的訊息時，會使用空白使用者 ID 進行 IBM MQ 所進行的任何權限檢查。 無法將 IBM MQ 資源的明確權限指派給空白使用者 ID，但會將其視為特殊群組 'nobody' 的成員。 如需特殊群組 nobody 的詳細資料，請參閱 可安裝服務介面參照資訊。

您可以使用 MQOPEN 執行環境定義設定，然後使用下列各節中指出的 MQOO_ 選項及 MQPMO_ 選項執行 MQPUT。 您也可以只使用 MQPUT1 來執行環境定義設定，在此情況下，您只需要選取下列區段中指出的 MQPMO_ 選項。

本主題的下列各節說明如何使用身分環境定義、使用者環境定義及所有環境定義。

傳遞身分環境定義

一般而言，程式應該在應用程式周圍的訊息之間傳遞身分環境定義資訊，直到資料到達其最終目的地為止。

每次程式變更資料時，都應該變更原始環境定義資訊。 不過，想要變更或設定任何環境定義資訊的應用程式必須具有適當的權限層次。 當應用程式開啟佇列時，佇列管理程式會檢查此權限; 它們必須有權使用 MQOPEN 呼叫的適當環境定義選項。

如果您的應用程式取得訊息，處理訊息中的資料，然後將變更的資料放入另一個訊息中 (可能供另一個應用程式處理)，則應用程式必須將身分環境定義資訊從原始訊息傳遞至新訊息。 您可以容許佇列管理程式建立原始環境定義資訊。

若要儲存原始訊息中的環境定義資訊，請在開啟佇列以取得訊息時使用 MQOO_SAVE_ALL_CONTEXT 選項。 這是您與 MQOPEN 呼叫搭配使用的任何其他選項之外的額外選項。 不過請注意，如果您只瀏覽訊息，則無法儲存環境定義資訊。

當您建立第二則訊息時:

- 使用 MQOO_PASS_IDENTITY_CONTEXT 選項 (除了 MQOO_OUTPUT 選項之外) 開啟佇列。
- 在 put-message 選項結構的 *Context* 欄位中，提供您從中儲存環境定義資訊之佇列的控點。
- 在 put-message 選項結構的 *Options* 欄位中，指定 MQPMO_PASS_IDENTITY_CONTEXT 選項。

傳遞使用者環境定義

您無法選擇僅傳遞使用者環境定義。 若要在放置訊息時傳遞使用者環境定義，請指定 MQPMO_PASS_ALL_CONTEXT。 使用者環境定義中的任何內容都會以與原始環境定義相同的方式來傳遞。

當 MQPUT 或 MQPUT1 發生且正在傳遞環境定義時，會將使用者環境定義中的所有內容從擷取的訊息傳遞至放置訊息。 放置應用程式所變更的任何使用者環境定義內容，都會以其原始值來放置。 放置應用程式已刪除的任何使用者環境定義內容都會還原在放置訊息中。 會保留放置應用程式已新增至訊息的任何使用者環境定義內容。

傳遞所有環境定義

如果您的應用程式取得訊息，並將訊息資料 (未變更) 放入另一個訊息中，則應用程式必須將所有 (身分、原點及使用者) 環境定義資訊從原始訊息傳遞至新訊息。 可能執行此動作的應用程式範例是訊息移轉裝置，它會將訊息從一個佇列移至另一個佇列。

除了您使用 MQOPEN 選項 MQOO_PASS_ALL_CONTEXT 及 put-message 選項 MQPMO_PASS_ALL_CONTEXT 之外，請遵循與傳遞身分環境定義相同的程序。

設定身分環境定義

如果您想要設定訊息的身分環境定義資訊:

- 使用 MQOO_SET_IDENTITY_CONTEXT 選項開啟佇列。

- 指定 MQPMO_SET_IDENTITY_CONTEXT 選項，將訊息放置在佇列上。 在訊息描述子中，指定您需要的任何身分環境定義資訊。

註: 當您使用 MQOO_SET_IDENTITY_CONTEXT 及 MQPMO_SET_IDENTITY_CONTEXT 選項設定部分 (但不是全部) 身分環境定義欄位時，請務必瞭解佇列管理程式不會設定任何其他欄位。

若要修改任何訊息環境定義選項，您必須具有適當的授權才能發出呼叫。 例如，若要使用 MQOO_SET_IDENTITY_CONTEXT 或 MQPMO_SET_IDENTITY_CONTEXT，您必須具有 `+setid` 許可權。

設定使用者環境定義

若要在使用者環境定義中設定內容，請在進行 MQSETMP 呼叫時，將訊息內容描述子 (MQPD) 的「環境定義」欄位設為 MQPD_USER_CONTEXT。

您不需要任何特殊權限，即可在使用者環境定義中設定內容。 使用者環境定義沒有 MQOO_SET_ * 或 MQPMO_SET_ * 環境定義選項。

設定所有環境定義

如果您想要同時設定訊息的身分和原始環境定義資訊:

1. 使用 MQOO_SET_ALL_CONTEXT 選項開啟佇列。
2. 指定 MQPMO_SET_ALL_CONTEXT 選項，將訊息放置在佇列上。 在訊息描述子中，指定您需要的任何身分及原始環境定義資訊。

每一種類型的環境定義設定都需要適當的權限。

相關概念

第 40 頁的『訊息環境定義』
訊息環境定義 資訊可讓擷取訊息的應用程式找出訊息的發送端。

相關參考

第 628 頁的『與訊息環境定義相關的 MQOPEN 選項』
如果您想要在將環境定義資訊放入佇列時能夠將它與訊息相關聯，則必須在開啟佇列時使用其中一個訊息環境定義選項。

使用 *MQPUT1* 呼叫將一則訊息放置在佇列上

當您在佇列上放置單一訊息之後，想要立即關閉佇列時，請使用 MQPUT1 呼叫。 例如，當伺服器應用程式傳送對每一個不同佇列的回覆時，可能會使用 MQPUT1 呼叫。

MQPUT1 在功能上相當於呼叫 MQOPEN，接著呼叫 MQPUT，接著呼叫 MQCLOSE。 MQPUT 和 MQPUT1 呼叫的語法唯一差異是針對 MQPUT 指定物件控點，而針對 MQPUT1 指定 MQOPEN 中定義的物件描述子結構 (MQOD) (請參閱 第 623 頁的『識別物件 (MQOD 結構)』)。 這是因為您需要將其必須開啟之佇列的相關資訊提供給 MQPUT1 呼叫，而當您呼叫 MQPUT 時，該佇列必須已開啟。

作為 MQPUT1 呼叫的輸入，您必須提供:

- 連線控點。
- 您要開啟之物件的說明。 這是物件描述子結構 (MQOD) 的形式。
- 您要放置在佇列上的訊息說明。 這是訊息描述子結構 (MQMD) 的形式。
- 以放置訊息選項結構 (MQPMO) 形式的控制資訊。
- 訊息內包含的資料長度 (MQLONG)。
- 訊息資料的位址。

MQPUT1 的輸出如下:

- 完成碼
- 原因碼

如果呼叫順利完成，它也會傳回您的選項結構及訊息描述子結構。 該呼叫會修改您的選項結構，以顯示佇列名稱及訊息傳送至其中的佇列管理程式。 如果您要求佇列管理程式為您要放置的訊息 ID 產生唯一值 (透過在

MQMD 結構的 *MsgId* 欄位中指定二進位零)，則在將此結構傳回給您之前，呼叫會在 *MsgId* 欄位中插入該值。

註: 您無法將 MQPUT1 與模型佇列名稱搭配使用; 不過，一旦開啟模型佇列，您可以對動態佇列發出 MQPUT1 。

MQPUT1 的六個輸入參數如下:

Hconn
這是連線控點。 對於 CICS 應用程式，您可以指定常數 MQHC_DEF_HCONN (值為零)，或使用 MQCONN 或 MQCONNX 呼叫傳回的連線控點。 對於其他程式，一律使用 MQCONN 或 MQCONNX 呼叫傳回的連線控點。

ObjDesc
這是物件描述子結構 (MQOD)。

在 *ObjectName* 及 *ObjectQMgrName* 欄位中，提供您要放置訊息的佇列名稱，以及擁有此佇列的佇列管理程式名稱。

MQPUT1 呼叫會忽略 *DynamicQName* 欄位，因為它無法使用模型佇列。

如果您要指定替代使用者 ID，以用來測試開啟佇列的權限，請使用 *AlternateUserId* 欄位。

MsgDesc
這是訊息描述子結構 (MQMD)。 與 MQPUT 呼叫一樣，請使用此結構來定義您要放置在佇列上的訊息。

PutMsgOpts
這是放置訊息選項結構 (MQPMO)。 請如同您在 MQPUT 呼叫中使用它一樣 (請參閱 第 632 頁的『使用 MQPMO 結構指定選項』)。

當 *Options* 欄位設為零時，佇列管理程式會在執行測試以取得存取佇列的權限時使用您自己的使用者 ID。 此外，佇列管理程式會忽略 MQOD 結構的 *AlternateUserId* 欄位中提供的任何替代使用者 ID。

BufferLength
這是您訊息的長度。

Buffer
這是包含訊息文字的緩衝區。

當您使用叢集時， MQPUT1 會像 MQOO_BIND_NOT_FIXED 一樣運作。 應用程式必須使用 MQPMO 結構 (而非 MQOD 結構) 中已解析的欄位來判斷訊息的傳送位置。 如需相關資訊，請參閱 配置佇列管理程式叢集 。

MQPUT1 中有 MQPUT1 呼叫的說明。

分送清單

在 IBM MQ for z/OS 上不受支援。 配送清單可讓您在單一 MQPUT 或 MQPUT1 呼叫中，將訊息放置到多個目的地。單一 MQOPEN 呼叫可以開啟多個佇列，然後單一 MQPUT 呼叫可以將訊息放置到其中每一個佇列。 用於此處理程序之 MQI 結構中的部分一般資訊，可以由與配送清單中所包含個別目的地相關的特定資訊所取代。

小心: 發佈清單不支援使用指向主題物件的別名佇列。 如果別名佇列指向配送清單中的主題物件，則 IBM MQ 會傳回 MQRC_ALIAS_BASE_Q_TYPE_ERROR。

發出 MQOPEN 呼叫時，會從「物件描述子 (MQOD)」取得一般資訊。 如果您在 *Version* 欄位中指定 MQOD_VERSION_2，並在 *RecsPresent* 欄位中指定大於零的值，則 *Hobj* 可以定義為清單 (一個以上佇列) 的控點，而不是佇列的控點。 在此情況下，會透過物件記錄 (MQOR) 提供特定資訊，其提供目的地 (即 *ObjectName* 和 *ObjectQMgrName*) 的詳細資料。

物件控點 (*Hobj*) 會傳遞至 MQPUT 呼叫，可讓您將放在清單中，而不是放在單一佇列中。

將訊息放入佇列 (MQPUT) 時，會從「放置訊息選項」結構 (MQPMO) 及「訊息描述子 (MQMD)」中取得一般資訊。 特定資訊以「放置訊息記錄 (MQPMR)」形式提供。

「回應記錄 (MQRR)」可以接收每一個目的地佇列特有的完成碼及原因碼。

第 639 頁的圖 56 顯示配送清單如何運作。

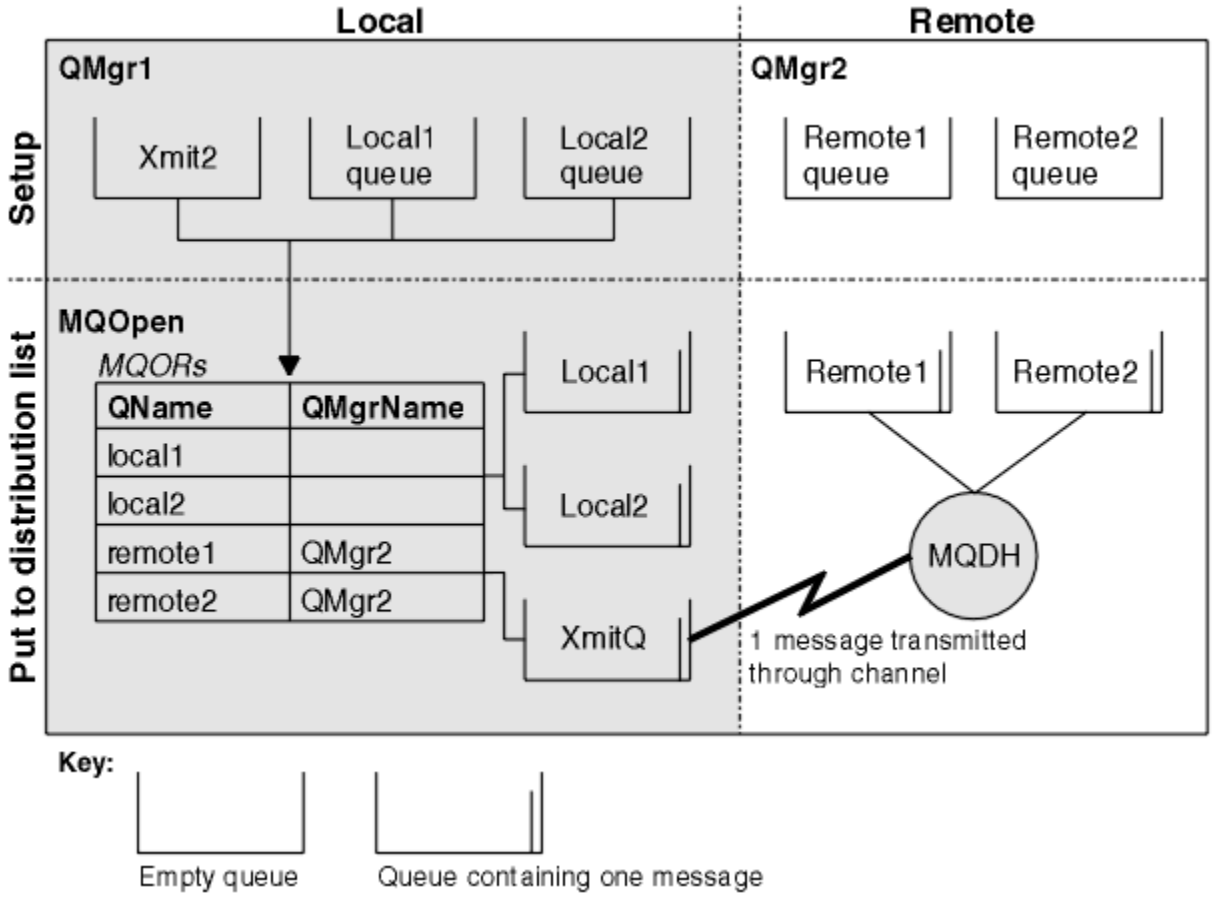


圖 *56:* 配送清單如何運作

開啟配送清單
使用 MQOPEN 呼叫來開啟配送清單，並使用呼叫的選項來指定您要對清單執行的動作。

作為 MQOPEN 的輸入，您必須提供:

- 連線控點 (如需說明，請參閱 第 630 頁的『將訊息放置在佇列上』)
- 物件描述子結構 (MQOD) 中的一般資訊
- 您要使用「物件記錄」結構 (MQOR) 開啟的每一個佇列名稱

MQOPEN 的輸出為:

- 代表您對配送清單的存取權的物件控點
- 一般完成碼
- 一般原因碼
- 回應記錄 (選用)，包含每一個目的地的完成碼及原因

使用 MQOD 結構

使用 MQOD 結構來識別您要開啟的佇列。

若要定義配送清單，您必須在 *Version* 欄位中指定 MQOD_VERSION_2，在 *RecsPresent* 欄位中指定大於零的值，在 *ObjectType* 欄位中指定 MQOT_Q。 如需 MQOD 結構所有欄位的說明，請參閱 MQOD 。

使用 MQOR 結構

提供每一個目的地的 MQOR 結構。

此結構包含目的地佇列及佇列管理程式名稱。 MQOD 中的 *ObjectName* 和 *ObjectQMgrName* 欄位不會用於配送清單。 必須有一或多個物件記錄。 如果 *ObjectQMgrName* 保留空白，則會使用本端佇列管理程式。 如需這些欄位的進一步相關資訊，請參閱 ObjectName 和 ObjectQMgrName 。

您可以使用兩種方式來指定目的地佇列:

- 使用偏移欄位 *ObjectRecOffset*。

  在此情況下，應用程式必須宣告其自己包含 MQOD 結構的結構，後面接著 MQOR 記錄陣列 (具有所需數目的陣列元素) ，並將 *ObjectRecOffset* 設為陣列中第一個元素從 MQOD 開始的偏移。 請確定此偏移是正確的。

  如果在應用程式執行所在的所有環境中都可以使用程式設計語言所提供的內建機能，則建議使用這些內建機能。 下列程式碼說明 COBOL 程式設計語言的這項技術:

```
01  MY-OPEN-DATA.
    02 MY-MQOD.
       COPY CMQODV.
    02 MY-MQOR-TABLE OCCURS 100 TIMES.
       COPY CMQORV.
    MOVE LENGTH OF MY-MQOD TO MQOD-OBJECTRECOFFSET.
```

  或者，如果程式設計語言在所有相關環境中不支援必要的內建機能，請使用常數 MQOD_CURRENT_LENGTH。 下列程式碼說明此技術:

```
01  MY-MQ-CONSTANTS.
    COPY CMQV.
01  MY-OPEN-DATA.
    02 MY-MQOD.
       COPY CMQODV.
    02 MY-MQOR-TABLE OCCURS 100 TIMES.
       COPY CMQORV.
    MOVE MQOD-CURRENT-LENGTH TO MQOD-OBJECTRECOFFSET.
```

  不過，這只有在 MQOD 結構和 MQOR 記錄陣列是連續的時才會正確運作; 如果編譯器在 MQOD 和 MQOR 陣列之間插入位元組，則必須將這些新增至儲存在 *ObjectRecOffset* 中的值。

  對於不支援指標資料類型的程式設計語言，或以不可攜至不同環境 (例如 COBOL 程式設計語言) 的方式來實作指標資料類型的程式設計語言，建議使用 *ObjectRecOffset* 。

- 使用指標欄位 *ObjectRecPtr*。

  在此情況下，應用程式可以與 MQOD 結構分開宣告 MQOR 結構的陣列，並將 *ObjectRecPtr* 設為陣列的位址。 下列程式碼說明 C 程式設計語言的這項技術:

```
MQOD MyMqod;
MQOR MyMqor[100];
MyMqod.ObjectRecPtr = MyMqor;
```

  對於以可攜至不同環境 (例如， C 程式設計語言) 的方式支援指標資料類型的程式設計語言，建議使用 *ObjectRecPtr* 。

無論您選擇何種技術，都必須使用 *ObjectRecOffset* 和 *ObjectRecPtr* ; 如果兩者都是零，或兩者都不是零，則呼叫會失敗，原因碼為 MQRC_OBJECT_RECORDS_ERROR。

使用 MQRR 結構

這些結構是目的地特有的; 每一個「回應記錄」針對配送清單的每一個佇列都包含一個 *CompCode* 和 *Reason* 欄位。 您必須使用此結構，才能識別任何問題所在的位置。

例如，如果您收到原因碼 MQRC_MULTIPLE_REASONS ，且您的配送清單包含五個目的地佇列，則在不使用此結構時，您將不知道問題適用的佇列。 不過，如果您有每一個目的地的完成碼和原因碼，您可以更容易找到錯誤。

如需 MQRR 結構的進一步相關資訊，請參閱 MQRR 。

第 641 頁的圖 57 顯示如何在 C 中開啟配送清單。

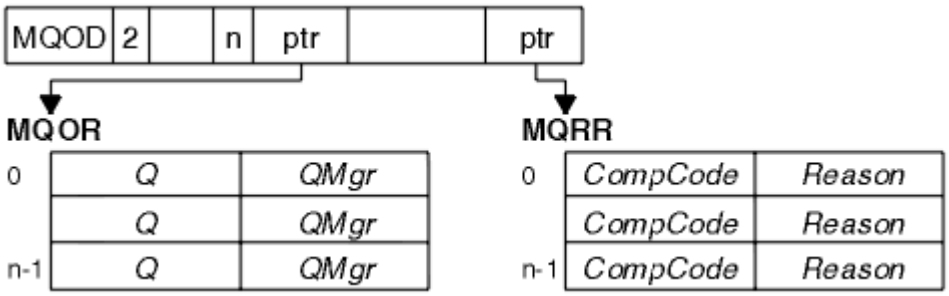


圖 *57:* 在 *C* 中開啟配送清單

第 641 頁的圖 58 顯示如何在 COBOL 中開啟配送清單。

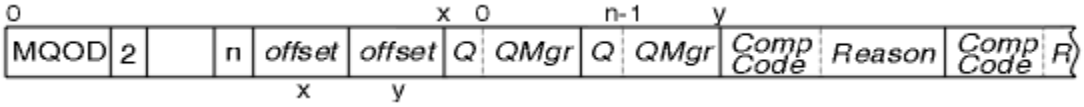


圖 *58:* 在 *COBOL* 中開啟配送清單

使用 MQOPEN 選項

您可以在開啟配送清單時指定下列選項:

- MQOO_OUTPUT
- MQOO_FAIL_IF_QUIESCING (選用)
- MQOO_ALTERNATE_USER_AUTHORITY (選用)
- MQOO_ * _CONTEXT (選用)

如需這些選項的說明，請參閱 第 621 頁的『開啟及關閉物件』 。

將訊息放入配送清單
若要將訊息放入配送清單，您可以使用 MQPUT 或 MQPUT1。

作為輸入，您必須提供:

- 連線控點 (如需說明，請參閱 第 630 頁的『將訊息放置在佇列上』)。
- 物件控點。 如果使用 MQOPEN 開啟配送清單，則 *Hobj* 只容許您放置到清單中。
- 訊息描述子結構 (MQMD)。 如需此結構的說明，請參閱 MQMD 。
- 以放置訊息選項結構 (MQPMO) 形式的控制資訊。 如需完成 MQPMO 結構欄位的相關資訊，請參閱 第 632 頁的『使用 MQPMO 結構指定選項』 。
- 控制資訊，格式為「放置訊息記錄 (MQPMR)」。
- 訊息內包含的資料長度 (MQLONG)。
- 訊息資料本身。

輸出為:

- 完成碼
- 原因碼
- 回應記錄 (選用)

使用 MQPMR 結構

此結構是選用的，並提供部分欄位的目的地特定資訊，您可能想要以不同於 MQMD 中已識別的那些欄位的方式來識別這些欄位。

如需這些欄位的說明，請參閱 MQPMR。

每一筆記錄的內容取決於 MQPMO 的 *PutMsgRecFields* 欄位中提供的資訊。 例如，在範例程式 AMQSPTL0.C (如需說明，請參閱 第 915 頁的『「配送清單」範例程式』)，顯示如何使用配送清單，範例會選擇在 MQPMR 中提供 *MsgId* 和 *CorrelId* 的值。 範例程式的此區段如下所示:

```
typedef struct
{
MQBYTE24 MsgId;
MQBYTE24 CorrelId;
} PutMsgRec;
...
/**********************
MQLONG PutMsgRecFields=MQPMRF_MSG_ID | MQPMRF_CORREL_ID;
```

這表示會為配送清單的每一個目的地提供 *MsgId* 和 *CorrelId* 。 「放置訊息記錄」是以陣列形式提供。

第 642 頁的圖 59 顯示如何將訊息放入 C 中的配送清單。

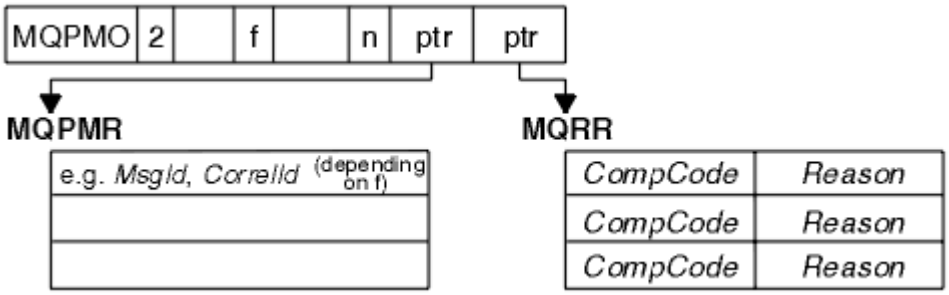


圖 *59:* 將訊息放入 *C* 中的配送清單

第 642 頁的圖 60 顯示如何將訊息放入 COBOL 中的發佈清單。

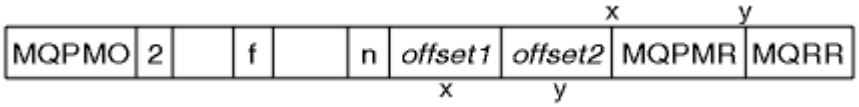


圖 *60:* 將訊息放入 *COBOL* 中的配送清單

使用 MQPUT1

如果您使用 MQPUT1，請考量下列要點:

1. *ResponseRecOffset* 和 *ResponseRecPtr* 欄位的值必須是空值或零。
2. 必要的話，必須從 MQOD 處理「回應記錄」。

部分放置呼叫失敗的案例

在您發出 MQOPEN 與 MQPUT 呼叫之間的間隔期間，如果在指令上使用 FORCE 選項變更佇列的某些屬性，則 MQPUT 呼叫會失敗，並傳回 MQRC_OBJECT_CHANGED 原因碼。

佇列管理程式會將物件控點標示為不再有效。 如果在處理 MQPUT1 呼叫時進行變更，或這些變更套用至佇列名稱所解析的任何佇列，也會發生這種情況。 以這種方式影響控點的屬性會列在 MQOPEN 中 MQOPEN 呼叫的說明中。 如果您的呼叫傳回 MQRC_OBJECT_CHANGED 原因碼，請關閉佇列，重新開啟它，然後重試放置訊息。

如果您嘗試放置訊息的佇列 (或佇列名稱解析的任何佇列) 禁止放置作業，則 MQPUT 或 MQPUT1 呼叫會失敗，並傳回 MQRC_PUT_INHIBITED 原因碼。 如果您稍後嘗試呼叫，且應用程式的設計使得其他程式定期變更佇列的屬性，則您可以順利放置訊息。

Furthemore，如果您嘗試放置訊息的佇列已滿， MQPUT 或 MQPUT1 呼叫會失敗，並傳回 MQRC_Q_FULL。

如果已刪除動態佇列 (暫時或永久)，則使用先前獲得的物件控點的 MQPUT 呼叫會失敗，並傳回 MQRC_Q_DELETED 原因碼。 在此情況下，最好關閉物件控點，因為它不再對您有用。

在配送清單的情況下，單一要求中可能會出現多個完成碼和原因碼。 這些無法僅使用 MQOPEN 和 MQPUT 上的 *CompCode* 和 *Reason* 輸出欄位來處理。

當您使用配送清單將訊息放置到多個目的地時，「回應記錄」包含每一個目的地的特定 *CompCode* 和 *Reason* 。 如果您收到完成碼 MQCC_FAILED ，則不會順利將任何訊息放置在任何目的地佇列上。 如果完成碼為 MQCC_WARNING ，則會順利將訊息放置在一或多個目的地佇列上。 如果您收到回覆碼 MQRC_MULTIPLE_REASONS ，則每個目的地的原因碼並非全部相同。 因此，建議使用 MQRR 結構，以便您可以判斷哪些佇列導致錯誤，以及每一個導致錯誤的原因。

從佇列取得訊息

使用此資訊來瞭解從佇列取得訊息的相關資訊。

您可以透過兩種方式從佇列取得訊息:

1. 您可以從佇列中移除訊息，讓其他程式無法再看到它。
2. 您可以複製訊息，將原始訊息保留在佇列上。 這稱為 瀏覽。 您可以在瀏覽訊息之後移除它。

在這兩種情況下，您都使用 MQGET 呼叫，但應用程式必須先連接至佇列管理程式，且必須使用 MQOPEN 呼叫來開啟佇列 (用於輸入、瀏覽或兩者)。 這些作業在 第 616 頁的『連接至佇列管理程式並切斷與佇列管理程式的連線』 和 第 621 頁的『開啟及關閉物件』中有說明。

當您已開啟佇列時，可以反覆地使用 MQGET 呼叫來瀏覽或移除相同佇列上的訊息。 當您完成從佇列取得您想要的所有訊息時，請呼叫 MQCLOSE。

請使用下列鏈結，以進一步瞭解從佇列取得訊息的相關資訊:

- 第 644 頁的『使用 MQGET 呼叫從佇列取得訊息』
- 第 647 頁的『從佇列擷取訊息的順序』
- 第 657 頁的『取得特定訊息』
- 第 658 頁的『增進非持續訊息的效能』
- z/OS 第 661 頁的『索引類型』
- 第 662 頁的『處理長度大於 4 MB 的訊息』
- 第 666 頁的『等待訊息』
- z/OS 第 667 頁的『信號 (signaling)』
- 第 668 頁的『跳過取消』
- 第 670 頁的『應用程式資料轉換』
- 第 671 頁的『瀏覽佇列上的訊息』
- 第 675 頁的『MQGET 呼叫失敗的部分情況』

相關概念

第 604 頁的『訊息佇列介面概觀』
瞭解「訊息佇列介面 (MQI)」元件。

第 616 頁的『連接至佇列管理程式並切斷與佇列管理程式的連線』
若要使用 IBM MQ 程式設計服務，程式必須具有與佇列管理程式的連線。 使用此資訊來瞭解如何連接至佇列管理程式，以及與佇列管理程式中斷連線。

第 621 頁的『開啟及關閉物件』
此資訊提供開啟及關閉 IBM MQ 物件的見解。

第 630 頁的『將訊息放置在佇列上』
使用此資訊來瞭解如何將訊息放置在佇列上。

第 713 頁的『查詢及設定物件屬性』
屬性是定義 IBM MQ 物件性質的內容。

第 716 頁的『確定及取消工作單元』
此資訊說明如何確定及取消工作單元中發生的任何可回復取得及放置作業。

第 725 頁的『使用觸發程式啟動 IBM MQ 應用程式』
瞭解觸發程式以及如何使用觸發程式來啟動 IBM MQ 應用程式。

第 741 頁的『使用 MQI 及叢集』
對於與叢集作業相關的呼叫和回覆碼，有一些特殊選項。

第 745 頁的『在 IBM MQ for z/OS 上使用及撰寫應用程式』
IBM MQ for z/OS 應用程式可以由在許多不同環境中執行的程式組成。 這表示他們可以利用多個環境中可用的設施。

第 57 頁的『IBM MQ for z/OS 上的 IMS 及 IMS 橋接器應用程式』
此資訊可協助您使用 IBM MQ 撰寫 IMS 應用程式。

使用 *MQGET* 呼叫從佇列取得訊息

MQGET 呼叫會從開啟的本端佇列取得訊息。 它無法從另一個系統上的佇列取得訊息。

作為 MQGET 呼叫的輸入，您必須提供:

- 連線控點。
- 佇列控點。
- 您要從佇列中取得之訊息的說明。 這是訊息描述子 (MQMD) 結構的形式。
- 採用「取得訊息選項 (MQGMO)」結構形式的控制資訊。
- 您指派用來保留訊息的緩衝區大小 (MQLONG)。
- 要在其中放置訊息的儲存體位址。

MQGET 的輸出為:

- 原因碼
- 完成碼
- 您指定的緩衝區中的訊息 (如果呼叫順利完成)
- 您的選項結構，已修改為顯示從中擷取訊息的佇列名稱
- 您的訊息描述子結構，已修改欄位內容來說明所擷取的訊息
- 訊息長度 (MQLONG)

MQGET 中有 MQGET 呼叫的說明。

下列各節說明您必須提供作為 MQGET 呼叫輸入的資訊。

- 第 644 頁的『指定連線控點』
- 第 644 頁的『使用 MQMD 結構和 MQGET 呼叫來說明訊息』
- 第 645 頁的『使用 MQGMO 結構指定 MQGET 選項』
- 第 647 頁的『指定緩衝區的大小』

指定連線控點

z/OS 對於 z/OS 應用程式上的 CICS，您可以指定常數 MQHC_DEF_HCONN (值為零)，或使用 MQCONN 或 MQCONNX 呼叫所傳回的連線控點。 對於其他應用程式，一律使用 MQCONN 或 MQCONNX 呼叫傳回的連線控點。

使用呼叫 MQOPEN 時傳回的佇列控點 (*Hobj*)。

使用 MQMD 結構和 MQGET 呼叫來說明訊息

若要識別您要從佇列取得的訊息，請使用訊息描述子結構 (MQMD)。

這是 MQGET 呼叫的輸入/輸出參數。 MQMD 在 第 16 頁的『IBM MQ 訊息』中說明之訊息內容的簡介，以及 MQMD 中結構本身的說明。

如果您知道要從佇列取得的訊息，請參閱 第 657 頁的『取得特定訊息』。

如果您未指定特定訊息， MQGET 會擷取佇列中的 第一個 訊息。 第 647 頁的『從佇列擷取訊息的順序』說明訊息的優先順序、佇列的 **MsgDeliverySequence** 屬性， 以及 MQGMO_LOGICAL_ORDER 選項如何決定訊息在佇列中的順序。

註: 如果您要多次使用 MQGET (例如，逐步執行佇列中的訊息) ，則必須在每次呼叫之後將此結構的 *MsgId* 及 *CorrelId* 欄位設為空值。 這會清除已擷取訊息 ID 的這些欄位。

不過，如果您要將訊息分組，則 *GroupId* 必須與相同群組中的訊息相同，以便呼叫尋找與前一個訊息具有相同 ID 的訊息，以組成整個群組。

使用 MQGMO 結構指定 MQGET 選項

MQGMO 結構是輸入/輸出變數，用於將選項傳遞至 MQGET 呼叫。 下列各節協助您完成此結構的部分欄位。

MQGMO 中有 MQGMO 結構的說明。

StrucId
StrucId 是 4 個字元的欄位，用來將結構識別為 get-message 選項結構。 一律指定 MQGMO_STRUC_ID。

Version
Version 說明結構的版本號碼。 MQGMO_VERSION_1 是預設值。 如果您想要使用第 2 版欄位或以邏輯順序擷取訊息，請指定 MQGMO_VERSION_2。 如果您想要使用第 3 版欄位或以邏輯順序擷取訊息，請指定 MQGMO_VERSION_3。 MQGMO_CURRENT_VERSION 將應用程式設為使用最新層次。

Options
在程式碼內，您可以依任何順序選取選項; 每一個選項在 *Options* 欄位中以位元表示。

Options 欄位控制項:

- MQGET 呼叫是否在完成之前等待訊息抵達佇列 (請參閱 第 666 頁的『等待訊息』)
- 取得作業是否包含在工作單元中。
- 是否在同步點外部擷取非持續訊息，容許快速傳訊
- z/OS 在 IBM MQ for z/OS 上，擷取的訊息是否標示為跳過取消 (請參閱 第 668 頁的『跳過取消』)
- 是從佇列中移除訊息，還是僅瀏覽訊息
- 使用瀏覽游標或其他選取準則來選取訊息
- 即使訊息長於您的緩衝區，呼叫是否成功
- z/OS 在 IBM MQ for z/OS 上，是否容許呼叫完成。 此選項也會設定信號，以指出您想要在訊息到達時收到通知
- 如果佇列管理程式處於靜止狀態，則呼叫是否失敗
- z/OS 在 IBM MQ for z/OS 上，如果連線處於靜止狀態，則呼叫是否失敗
- 是否需要應用程式訊息資料轉換 (請參閱 第 670 頁的『應用程式資料轉換』)
- 從佇列 z/OS 擷取訊息及區段的順序 (IBM MQ for z/OS 除外)
- 是否完成，僅可擷取邏輯訊息 z/OS (IBM MQ for z/OS 除外)
- 是否只有在群組中的 所有 訊息都可用時，才能擷取群組中的訊息
- 是否只有在邏輯訊息中的 所有 區段都可用時，才能擷取邏輯訊息中的區段 z/OS (IBM MQ for z/OS 除外)

如果您將 *Options* 欄位設為預設值 (MQGMO_NO_WAIT) ， MQGET 呼叫會以下列方式運作:

- 如果佇列上沒有符合您選取準則的訊息，則呼叫不會等待訊息到達，但會立即完成。 z/OS 此外，在 IBM MQ for z/OS 中，當這類訊息到達時，通話不會設定要求通知的信號。
- 呼叫以同步點運作的方式由平台決定:

平台	在同步點控制下
IBM i	否
AIX and Linux 系統	否
z/OS z/OS	是
Windows 系統	否

- z/OS 在 IBM MQ for z/OS 上，擷取的訊息未標示為跳過取消。
- 即會從佇列中移除選取的訊息 (未瀏覽)。
- 不需要應用程式訊息資料轉換。
- 如果訊息長於您的緩衝區，則呼叫會失敗。

WaitInterval
WaitInterval 欄位指定當您使用 MQGMO_WAIT 選項時， MQGET 呼叫等待訊息抵達佇列的時間上限 (毫秒)。 如果在 *WaitInterval* 中指定的時間內沒有任何訊息到達，則呼叫會完成並傳回原因碼，指出沒有訊息符合佇列上的選取準則。

z/OS 在 IBM MQ for z/OS 上，如果您使用 MQGMO_SET_SIGNAL 選項，則 *WaitInterval* 欄位會指定設定信號的時間。

如需這些選項的相關資訊，請參閱 第 666 頁的『等待訊息』 z/OS 和 第 667 頁的『信號 (signaling)』 。

z/OS ***Signal1***
Signal1 僅在 IBM MQ for z/OS 上受支援。

如果您使用 MQGMO_SET_SIGNAL 選項來要求在適當訊息到達時通知您的應用程式，則可以在 *Signal1* 欄位中指定信號類型。 在所有其他平台上的 IBM MQ 中，會保留 *Signal1* 欄位，且其值並不重要。

z/OS 如需相關資訊，請參閱 第 667 頁的『信號 (signaling)』。

Signal2
Signal2 欄位會保留在所有平台上，且其值並不顯著。

z/OS 如需相關資訊，請參閱第 667 頁的『信號 (signaling)』。

ResolvedQName
ResolvedQName 是一個輸出欄位，其中佇列管理程式會傳回從中擷取訊息的佇列名稱 (解析任何別名之後)。

MatchOptions
MatchOptions 控制 MQGET 的選取準則。

GroupStatus
GroupStatus 指出您已擷取的訊息是否在群組中。

SegmentStatus
SegmentStatus 指出您擷取的項目是否為邏輯訊息的區段。

Segmentation
Segmentation 指出擷取的訊息是否容許分段。

MsgToken

MsgToken 可唯一識別訊息。

ReturnedLength
ReturnedLength 是佇列管理程式在其中傳回所傳回訊息資料長度 (以位元組為單位) 的輸出欄位。

MsgHandle
要移入從佇列擷取之訊息內容的訊息控點。 此控點先前已由 MQCRTMH 呼叫建立。 在擷取訊息之前，會先清除已與控點相關聯的任何內容。

指定緩衝區的大小

在 MQGET 呼叫的 **BufferLength** 參數中，指定用來保留您所擷取訊息資料的緩衝區大小。 您可以用三種方式來決定這應該有多大:

1. 您可能已知道此程式預期的訊息長度。 如果是，請指定此大小的緩衝區。

   不過，如果您想要完成 MQGET 呼叫，即使訊息對緩衝區而言太大，也可以在 MQGMO 結構中使用 MQGMO_ACCEPT_TRUNCATED_MSG 選項。 在此情況下:

   - 緩衝區已填入盡可能多的訊息
   - 呼叫會傳回警告完成碼
   - 訊息會從佇列中移除 (捨棄訊息的其餘部分)，或進階瀏覽游標 (如果您正在瀏覽佇列)
   - 在 *DataLength* 中傳回訊息的實際長度

   如果沒有這個選項，呼叫仍會完成但有警告，但不會從佇列中移除訊息 (或前進瀏覽游標)。
2. 預估緩衝區的大小 (甚至指定零位元組的大小)，且 不要 使用 MQGMO_ACCEPT_TRUNCATED_MSG 選項。 如果 MQGET 呼叫失敗 (例如，因為緩衝區太小)，則會在呼叫的 **DataLength** 參數中傳回訊息長度。 (緩衝區仍會盡可能填滿訊息，但呼叫的處理尚未完成。) 儲存此訊息的 *MsgId*，然後重複 MQGET 呼叫，並指定正確大小的緩衝區，以及您從第一次呼叫中記下的 *MsgId*。

   如果您的程式正在處理其他程式也正在處理的佇列，則在您的程式可以發出另一個 MQGET 呼叫之前，其中一個其他程式可能會移除您想要的訊息。 您的程式可能會浪費時間搜尋不再存在的訊息。 若要避免此情況，請先瀏覽佇列，直到找到您想要的訊息，並指定 *BufferLength* 為零並使用 MQGMO_ACCEPT_TRUNCATED_MSG 選項。 這會將瀏覽游標定位在您想要的訊息下。 然後，您可以指定 MQGMO_MSG_UNDER_CURSOR 選項來重新呼叫 MQGET，以擷取訊息。 如果另一個程式移除瀏覽與移除呼叫之間的訊息，則第二個 MQGET 會立即失敗 (不搜尋整個佇列)，因為瀏覽游標下沒有訊息。
3. *MaxMsgLength* 佇列 屬性決定該佇列接受的訊息長度上限; *MaxMsgLength* 佇列管理程式 屬性決定該佇列管理程式接受的訊息長度上限。 如果您不知道預期訊息的長度，則可以查詢 **MaxMsgLength** 屬性 (使用 MQINQ 呼叫)，然後指定此大小的緩衝區。

   嘗試使緩衝區大小盡可能接近實際訊息大小，以避免降低效能。

   如需 **MaxMsgLength** 屬性的進一步相關資訊，請參閱 第 662 頁的『增加訊息長度上限』。

從佇列擷取訊息的順序

您可以控制從佇列擷取訊息的順序。 本節會查看選項。

優先順序

當程式將訊息放入佇列時，可以指派訊息的優先順序 (請參閱 第 22 頁的『訊息優先順序』)。 同等優先順序的訊息會依到達順序儲存在佇列中，而不是它們的確定順序。

佇列管理程式會以嚴格的 FIFO (先進先出) 順序或優先順序內的 FIFO 順序來維護佇列。 這取決於佇列的 **MsgDeliverySequence** 屬性設定。 當訊息到達佇列時，它會緊接在具有相同優先順序的最後一則訊息之後插入。

程式可以從佇列中取得第一個訊息，也可以從佇列中取得特定訊息，而忽略這些訊息的優先順序。 例如，程式可能想要處理先前傳送之特定訊息的回覆。 如需相關資訊，請參閱第 657 頁的『取得特定訊息』。

如果應用程式將一連串訊息放置在佇列上，則另一個應用程式可以依照放置訊息的相同順序來擷取這些訊息，但前提是:

- 這些訊息都具有相同的優先順序
- 訊息全部放在相同的工作單元內，或全部放在工作單元外
- 佇列位於放置應用程式的本端

如果不符合這些條件，且應用程式相依於以特定順序擷取的訊息，則應用程式必須在訊息資料中包括排序資訊，或在傳送下一個訊息之前建立確認接收訊息的方法。

z/OS 在 IBM MQ for z/OS 上，您可以使用佇列屬性 *IndexType* 來增加佇列上 MQGET 作業的速度。 如需相關資訊，請參閱 第 661 頁的『索引類型』。

邏輯和實體排序

佇列上的訊息可以 實體 或 邏輯 順序發生 (在每一個優先順序層次內)。

實體順序是訊息抵達佇列的順序。 邏輯順序是當群組內所有訊息和區段的邏輯順序彼此相鄰，且位於屬於群組之第一個項目的實體位置所決定的位置時。

如需群組、訊息及區段的說明，請參閱 第 37 頁的『訊息群組』。 這些實體和邏輯順序可能不同，因為:

- 群組可以在不同應用程式的類似時間抵達目的地，因此會遺失任何不同的實體順序。
- 即使在單一群組內，由於重新遞送或延遲群組中的部分訊息，訊息也可能不正常。

例如，邏輯順序可能類似於圖 第 648 頁的圖 61:

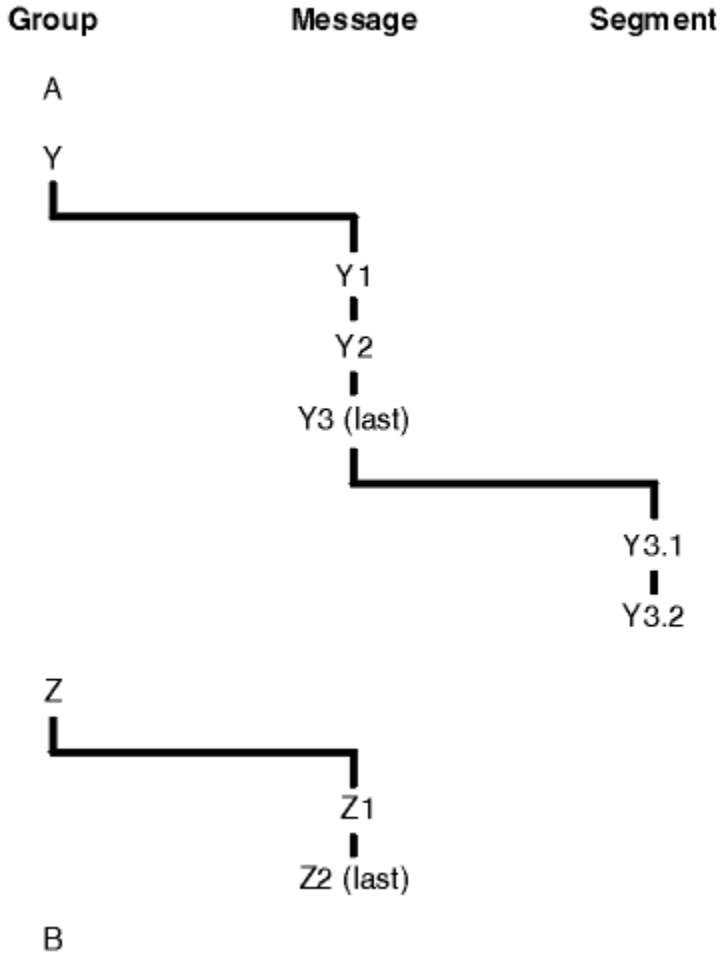


圖 *61:* 佇列上的邏輯順序

這些訊息會以下列邏輯順序在佇列上發生:

1. 訊息 A (不在群組中)
2. 群組 Y 的邏輯訊息 1
3. 群組 Y 的邏輯訊息 2
4. 群組 Y 的 (最後一個) 邏輯訊息 3 的區段 1
5. 群組 Y 的 (最後一個) 邏輯訊息 3 的 (最後一個) 區段 2

6. 群組 Z 的邏輯訊息 1
7. 群組 Z 的 (最後一個) 邏輯訊息 2
8. 訊息 B (不在群組中)

不過，實際順序可能完全不同。 每一個群組內 第一個 項目的實體位置決定整個群組的邏輯位置。 例如，如果群組 Y 和 Z 到達相似時間，且群組 Z 的訊息 2 超過相同群組的訊息 1，則實體順序看起來會類似圖 第 649 頁的圖 62:

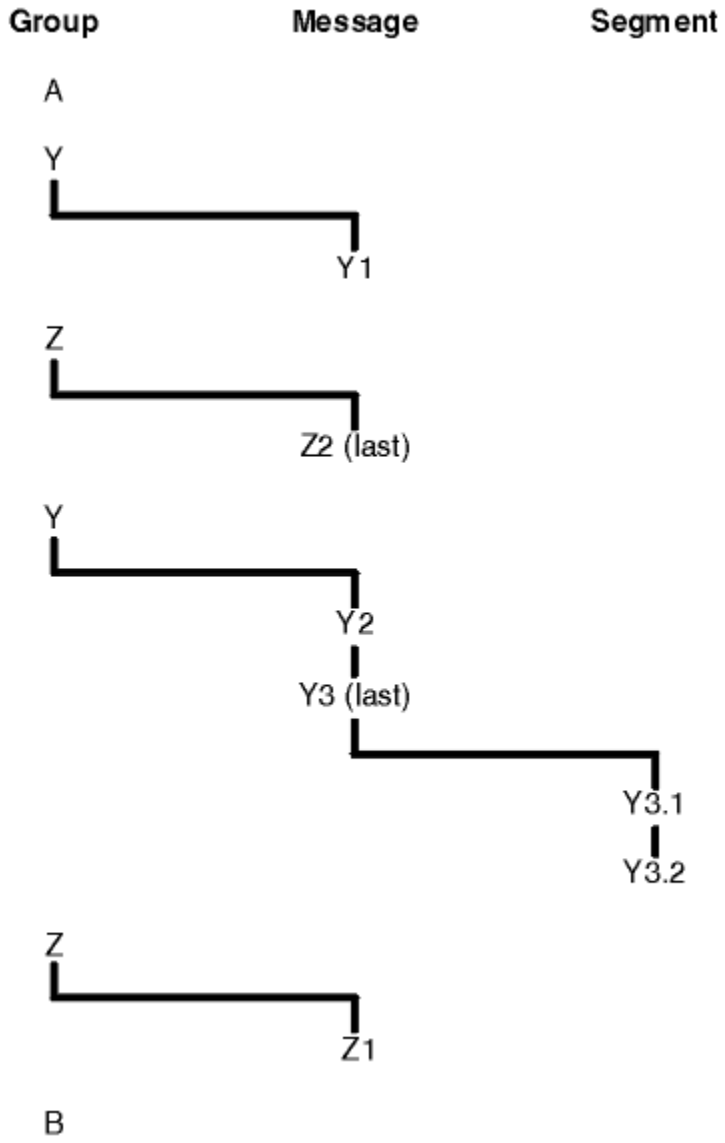


圖 *62:* 佇列上的實體訂單

這些訊息會以下列實體順序出現在佇列上:

1. 訊息 A (不在群組中)
2. 群組 Y 的邏輯訊息 1
3. 群組 Z 的邏輯訊息 2
4. 群組 Y 的邏輯訊息 2
5. 群組 Y 的 (最後一個) 邏輯訊息 3 的區段 1
6. 群組 Y 的 (最後一個) 邏輯訊息 3 的 (最後一個) 區段 2
7. 群組 Z 的邏輯訊息 1
8. 訊息 B (不在群組中)

註: 在 IBM MQ for z/OS 上，如果佇列是由 GROUPID 編製索引，則不保證佇列上訊息的實體順序。

取得訊息時，您可以指定 MQGMO_LOGICAL_ORDER 以邏輯順序而非實體順序來擷取訊息。

如果您使用 MQGMO_BROWSE_FIRST 及 MQGMO_LOGICAL_ORDER 發出 MQGET 呼叫，則使用 MQGMO_BROWSE_NEXT 的後續 MQGET 呼叫也必須指定 MQGMO_LOGICAL_ORDER。 相反地，如果 MQGET (含 MQGMO_BROWSE_FIRST) 未指定 MQGMO_LOGICAL_ORDER ，則下列 MQGET (含 MQGMO_BROWSE_NEXT) 也不能指定。

佇列管理程式針對佇列上瀏覽訊息的 MQGET 呼叫所保留的群組及區段資訊，與佇列管理程式針對從佇列中移除訊息的 MQGET 呼叫所保留的群組及區段資訊不同。 當您指定 MQGMO_BROWSE_FIRST 時，佇列管理程式會忽略用於瀏覽的群組及區段資訊，並掃描佇列，如同沒有現行群組及現行邏輯訊息一樣。

註: 如果沒有指定 MQGMO_LOGICAL_ORDER ，請勿使用 MQGET 呼叫來瀏覽訊息群組 (或不在群組中的邏輯訊息) 的結尾 之後 。 例如，如果群組中的最後一則訊息 位於 佇列上群組中的第一個訊息之前，則使用 MQGMO_BROWSE_NEXT 瀏覽超過群組結尾，並指定 MQMO_MATCH_MSG_SEQ_NUMBER 並將 *MsgSeqNumber* 設為 1 (以尋找下一個群組的第一個訊息) 再次傳回群組中已瀏覽過的第一個訊息。 這可能會立即發生，或稍後會有一些 MQGET 呼叫 (如果有中間群組的話)。

開啟佇列 兩次 進行瀏覽，以避免發生無限迴圈的可能性:

- 使用第一個控點只瀏覽每一個群組中的第一個訊息。
- 使用第二個控點只瀏覽特定群組內的訊息。
- 在瀏覽群組中的訊息之前，請使用 MQMO_ * 選項將第二個瀏覽游標移至第一個瀏覽游標的位置。
- 請勿使用超出群組結尾的 MQGMO_BROWSE_NEXT 瀏覽。

如需此作業的進一步相關資訊，請參閱 MQGET、 MQMD 及 MQI 選項的驗證規則。

對於大部分應用程式，您可能會在瀏覽時選擇邏輯或實體排序。 不過，如果您想要在這些模式之間切換，請記住，當您第一次使用 MQGMO_LOGICAL_ORDER 發出瀏覽時，會建立您在邏輯序列內的位置。

如果此時群組內的第一個項目不存在，則您所在的群組不會被視為邏輯序列的一部分。

一旦瀏覽游標位於群組內，即使移除第一則訊息，它也可以在相同群組內繼續。 一開始，您永遠無法使用 MQGMO_LOGICAL_ORDER 移入第一個項目不存在的群組。

MQPMO_LOGICAL_ORDER

MQPMO 選項會告知佇列管理程式應用程式如何將訊息放入邏輯訊息的群組及區段中。 只能在 MQPUT 呼叫上指定它; 它在 MQPUT1 呼叫上無效。

如果指定 MQPMO_LOGICAL_ORDER，則表示應用程式使用連續的 MQPUT 呼叫來執行以下動作:

1. 按區段偏移的遞增順序（從 0 開始，無間隙），將區段放置在每個邏輯訊息中。
2. 先將所有區段放置在一個邏輯訊息中，然後再將區段放置在下一個邏輯訊息中。
3. 按訊息序號的遞增順序（從 1 開始，無間隙），將邏輯訊息放置在每個訊息群組中。 IBM MQ 將自動遞增訊息序號。
4. 先將所有邏輯訊息放置在一個訊息群組中，然後再將邏輯訊息放置在下一個訊息群組中。

因為應用程式已告知佇列管理程式如何將訊息放入邏輯訊息的群組及區段中，所以應用程式不需要維護及更新每一個 MQPUT 呼叫的群組及區段資訊，因為佇列管理程式會維護及更新此資訊。 具體而言，它表示應用程式不需要在 MQMD 中設定 *GroupId*、 *MsgSeqNumber* 及 *Offset* 欄位，因為佇列管理程式會將這些欄位設為適當的值。 應用程式只能在 MQMD 中設定 *MsgFlags* 欄位，以指出訊息何時屬於邏輯訊息的群組或區段，並指出邏輯訊息的群組或最後區段中的最後一則訊息。

啟動訊息群組或邏輯訊息之後，後續 MQPUT 呼叫必須在 MQMD 的 *MsgFlags* 中指定適當的 MQMF_ * 旗標。 如果應用程式嘗試在有未終止的訊息群組時放置不在群組中的訊息，或在有未終止的邏輯訊息時放置不是區段的訊息，則呼叫會失敗，原因碼為 MQRC_INCOMPLETE_GROUP 或 MQRC_INCOMPLETE_MSG ，視情況而定。 不過，佇列管理程式會保留現行訊息群組或現行邏輯訊息的相關資訊，在重新發出 MQPUT 呼叫以放置不在群組或非區段中的訊息之前，應用程式可以先傳送訊息 (可能沒有應用程式訊息資料) ，並適當地指定 MQMF_LAST_MSG_IN_GROUP 或 MQMF_LAST_SEgment。

第 649 頁的圖 62 顯示有效選項及旗標的組合，以及佇列管理程式在每一個案例中使用的 *GroupId*、*MsgSeqNumber* 及 *Offset* 欄位值。 未顯示在表格中的選項及旗標組合無效。 表格中的直欄具有下列意義; 表示「是」或「否」：

ORD 日誌
是否在呼叫上指定 MQPMO_LOGICAL_ORDER 選項。

MIG
是否在呼叫上指定 MQMF_MSG_IN_GROUP 或 MQMF_LAST_MSG_IN_GROUP 選項。

SEG
是否在呼叫上指定 MQMF_SEGMENT 或 MQMF_LAST_SEGMENT 選項。

SEG 正常
是否在呼叫上指定 MQMF_SEGMENTATION_ALLOWED 選項。

Cur grp
呼叫之前是否存在現行訊息群組。

目前日誌訊息
呼叫之前是否存在現行邏輯訊息。

其他直欄
顯示佇列管理程式使用的值。 前一個表示用於佇列控點之前一個訊息中欄位的值。

表 *116:* 與邏輯訊息群組及區段中的訊息相關的 *MQPUT* 選項

您指定的選項	您指定的選項	您指定的選項	您指定的選項	呼叫之前的群組和日誌訊息狀態	呼叫之前的群組和日誌訊息狀態	佇列管理程式使用的值	佇列管理程式使用的值	佇列管理程式使用的值
ORD 日誌	MIG	SEG	SEG 正常	Cur grp	目前日誌訊息	**GroupId**	**MsgSeqNumber**	**Offset**
是	否	否	否	否	否	MQGI_NONE	1	0
是	否	否	是	否	否	新建群組 ID	1	0
是	否	是	兩者擇一	否	否	新建群組 ID	1	0
是	否	是	兩者擇一	否	是	前一個群組 ID	1	前一個偏移 + 前一個區段長度
是	是	兩者擇一	兩者擇一	否	否	新建群組 ID	1	0
是	是	兩者擇一	兩者擇一	是	否	前一個群組 ID	前一個序號 + 1	0
是	是	是	兩者擇一	是	是	前一個群組 ID	前一個序號	前一個偏移 + 前一個區段長度
否	否	否	否	兩者擇一	兩者擇一	MQGI_NONE	1	0
否	否	否	是	兩者擇一	兩者擇一	如果 MQGI_NONE，則為新群組 ID，否則為欄位中的值	1	0
否	否	是	兩者擇一	兩者擇一	兩者擇一	如果 MQGI_NONE，則為新群組 ID，否則為欄位中的值	1	欄位中的值

表 *116:* 與邏輯訊息群組及區段中的訊息相關的 *MQPUT* 選項 *(*繼續*)*								
您指定的選項	**您指定的選項**	**您指定的選項**	**您指定的選項**	**呼叫之前的群組和日誌訊息狀態**	**呼叫之前的群組和日誌訊息狀態**	**佇列管理程式使用的值**	**佇列管理程式使用的值**	**佇列管理程式使用的值**
否	是	否	兩者擇一	兩者擇一	兩者擇一	如果 MQGI_NONE，則為新群組 ID，否則為欄位中的值	欄位中的值	0
否	是	是	兩者擇一	兩者擇一	兩者擇一	如果 MQGI_NONE，則為新群組 ID，否則為欄位中的值	欄位中的值	欄位中的值

註:

- MQPMO_LOGICAL_ORDER 在 MQPUT1 呼叫中無效。
- 對於 *MsgId* 欄位，如果指定了 MQPMO_NEW_MSG_ID 或 MQMI_NONE，則佇列管理程式會產生新的訊息 ID，否則會使用欄位中的值。
- 對於 *CorrelId* 欄位，如果指定 MQPMO_NEW_CORREL_ID，則佇列管理程式會產生新的相關性 ID，否則會使用欄位中的值。

當您指定 MQPMO_LOGICAL_ORDER 時，佇列管理程式會要求將群組中的所有訊息及邏輯訊息中的區段放置在 MQMD 的 *Persistence* 欄位中，並具有相同的值，亦即，全部必須是持續性，或全部必須是非持續性。 如果未滿足此條件，則 MQPUT 呼叫會失敗，原因碼為 MQRC_INCONSISTENT_PERSISTENCE。

MQPMO_LOGICAL_ORDER 選項會影響工作單元，如下所示:

- 如果將群組或邏輯訊息中的第一個實體訊息放置在工作單元內，則如果使用相同的佇列控點，則必須將群組或邏輯訊息中的所有其他實體訊息放置在工作單元內。 不過，它們不需要放在相同的工作單元內，可讓由許多實體訊息組成的訊息群組或邏輯訊息，在佇列控點的兩個以上連續工作單元之間分割。
- 如果群組或邏輯訊息中的第一個實體訊息未放置在工作單元內，則如果使用相同的佇列控點，則群組或邏輯訊息中的其他實體訊息都無法放置在工作單元內。

如果未滿足這些條件， MQPUT 呼叫會失敗，原因碼為 MQRC_INCONSISTENT_UOW。

指定 MQPMO_LOGICAL_ORDER 時， MQPUT 呼叫上提供的 MQMD 不得小於 MQMD_VERSION_2。 如果未滿足此條件，則呼叫會失敗，原因碼為 MQRC_WRONG_MD_VERSION。

如果未指定 MQPMO_LOGICAL_ORDER，則可以任意順序放置邏輯訊息群組和區段中的訊息，而不需要放置完整訊息群組或完整邏輯訊息。 應用程式負責確保 *GroupId*、 *MsgSeqNumber*、 *Offset* 和 *MsgFlags* 欄位具有適當的值。

在發生系統失敗之後，請使用此技術來重新啟動中間的訊息群組或邏輯訊息。 當系統重新啟動時，應用程式可以將 *GroupId*、 *MsgSeqNumber*、 *Offset*、 *MsgFlags* 及 *Persistence* 欄位設為適當的值，然後發出 MQPUT 呼叫，並視需要設定 MQPMO_SYNCPOINT 或 MQPMO_NO_SYNCPOINT，但不指定 MQPMO_LOGICAL_ORDER。 如果此呼叫成功，佇列管理程式會保留群組和區段資訊，且使用該佇列控點的後續 MQPUT 呼叫可以正常指定 MQPMO_LOGICAL_ORDER。

佇列管理程式針對 MQPUT 呼叫所保留的群組及區段資訊，與它針對 MQGET 呼叫所保留的群組及區段資訊不同。

對於任何給定的佇列控點，應用程式可以將指定 MQPMO_LOGICAL_ORDER 的 MQPUT 呼叫與未指定的 MQPUT 呼叫混合，但請注意下列要點:

- 如果未指定 MQPMO_LOGICAL_ORDER，每一個成功的 MQPUT 呼叫都會導致佇列管理程式將佇列控點的群組和區段資訊設為應用程式指定的值，以取代佇列管理程式保留給佇列控點的現有群組和區段資訊。
- 如果未指定 MQPMO_LOGICAL_ORDER，當有現行訊息群組或邏輯訊息時，呼叫不會失敗; 呼叫可能會成功，並產生 MQCC_WARNING 完成碼。 第 653 頁的表 117 顯示可能產生的各種觀察值。 在這些情況下，如果完成碼不是 MQCC_OK，則原因碼為下列其中一項 (視情況而定):
  - MQRC_INCOMPLETE_GROUP
  - MQRC_INCOMPLETE_MSG
  - MQRC_INCONSISTENT_PERSISTENCE
  - MQRC_INCONSISTENT_UOW

  註: 佇列管理程式不會檢查 MQPUT1 呼叫的群組及區段資訊。

表 *117: MQPUT* 或 *MQCLOSE* 呼叫與群組和區段資訊不一致時的結果

現行呼叫為	前一個呼叫是 MQPUT，含有 MQPMO_LOGICAL_ORDER	前一個呼叫是不含 MQPMO_LOGICAL_ORDER 的 MQPUT
具有 MQPMO_LOGICAL_ORDER 的 MQPUT	MQCC_FAILED	MQCC_FAILED
不含 MQPMO_LOGICAL_ORDER 的 MQPUT	MQCC_WARNING	MQCC_OK
MQCLOSE 具有未終止的群組或邏輯訊息	MQCC_WARNING	MQCC_OK

對於以邏輯順序放置訊息和區段的應用程式，請指定 MQPMO_LOGICAL_ORDER，因為這是最簡單的選項。 此選項可讓應用程式解除管理群組和區段資訊的需求，因為佇列管理程式會管理該資訊。 不過，特殊化應用程式可能需要比 MQPMO_LOGICAL_ORDER 選項所提供的更多控制項，這可以透過不指定該選項來達成; 如果您這樣做，則必須確保在每一個 MQPUT 或 MQPUT1 呼叫之前， MQMD 中的 *GroupId*、 *MsgSeqNumber*、 *Offset* 及 *MsgFlags* 欄位都已正確設定。

例如，如果應用程式想要轉遞所接收的實體訊息，而不管這些訊息是在邏輯訊息的群組還是區段中，則不得指定 MQPMO_LOGICAL_ORDER，原因有兩個:

- 如果擷取訊息並依序放置，則指定 MQPMO_LOGICAL_ORDER 會將新的群組 ID 指派給訊息，這可能導致訊息的發送端難以或無法與訊息群組中產生的任何回覆或報告訊息產生關聯。
- 在傳送與接收佇列管理程式之間有多條路徑的複雜網路中，實體訊息可能不正常送達。 透過在 MQGET 呼叫上不指定 MQPMO_LOGICAL_ORDER 和 MQGMO_LOGICAL_ORDER，轉遞應用程式可以在每則實體訊息到達時立即擷取並轉遞，而無需等待邏輯順序中的下一則實體訊息到達。

在放置報告訊息時，為邏輯訊息群組或區段中的訊息產生報告訊息的應用程式也不得指定 MQPMO_LOGICAL_ORDER。

MQPMO_LOGICAL_ORDER 可以與任何其他 MQPMO_ * 選項一起指定。

將邏輯排序的群組放入叢集佇列 (MQOO_BIND_ON_GROUP)

MQOO_BIND_ON_OPEN 選項可確保將來自此應用程式的所有訊息以及所有群組遞送至單一實例。 這有一個缺點，就是應用程式資料流量未在叢集佇列的多個實例之間進行負載平衡。 為了啟用工作量平衡，同時保持訊息群組完整，您必須設定下列選項:

- MQPUT 呼叫必須指定 MQPMO_LOGICAL_ORDER
- MQOPEN 呼叫必須指定下列兩個選項之一:
  - MQ OO_BIND_ON_GROUP
  - MQOO_BIND_AS_Q_DEF 及佇列定義必須指定 DEFBIND (GROUP)

然後會在訊息群組之間 驅動 工作量平衡，而不需要佇列的 MQCLOSE 及 MQOPEN。 群組之間 表示 MQMF_MSG_IN_GROUP 已設定在 MQMD (v2) 或 MQMDE 中，且沒有局部完整的群組正在進行中。 當群組進行中時，會重複使用物件控點中已解析的佇列管理程式及佇列名稱。

如果前一則訊息是 MQPMO_LOGICAL_ORDER 及/或已設定 MQMF_MSG_IN_GROUP，但現行訊息不是群組的一部分，則 PUT 呼叫會失敗，並出現 MQRC_INCOMPLETE_GROUP。

如果個別 MQPUT 未指定 MQPMO_LOGICAL_ORDER，且沒有作用中的現行群組，則會驅動該訊息的工作量平衡 (如同 MQOPEN 呼叫已指定 MQOO_BIND_NOT_FIXED 一樣)。

對於使用 MQOO_BIND_ON_GROUP 連結至目的地的訊息，不會進行重新配置。 如需重新配置的相關資訊，請參閱 第 37 頁的『訊息群組』。

分組邏輯訊息

在群組中使用邏輯訊息有兩個主要原因:

- 您可能需要以特定順序處理訊息。
- 您可能需要以相關方式處理群組中的每一則訊息。

在任一情況下，請擷取具有相同取得應用程式實例的整個群組。

例如，假設群組由四個邏輯訊息組成。 放置應用程式看起來如下:

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP

MQCMIT
```

取得應用程式針對群組中的第一個訊息指定 MQGMO_ALL_MSGS_AVAILABLE 選項。 這可確保在群組內的所有訊息都到達之前，不會開始處理。 群組內的後續訊息會忽略 MQGMO_ALL_MSGS_AVAILABLE 選項。

擷取群組的第一個邏輯訊息時，您可以使用 MQGMO_LOGICAL_ORDER 來確保依序擷取群組的其餘邏輯訊息。

因此，取得應用程式看起來如下:

```
/* Wait for the first message in a group, or a message not in a group */
GMO.Options = MQGMO_SYNCPOINT | MQGMO_WAIT
            | MQGMO_ALL_MSGS_AVAILABLE | MQGMO_LOGICAL_ORDER
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
   MQGET
   /* Process each remaining message in the group */
   ...

MQCMIT
```

如需分組訊息的進一步範例，請參閱 第 664 頁的『邏輯訊息的應用程式分段』 和 第 654 頁的『放置並取得跨越工作單元的群組』。

小心: 使用發佈/訂閱將訊息傳送至主題 (或將訊息放置至主題別名) 時，不允許進行訊息分組和分段。

因為訂閱可以獨立於發佈活動來建立及移除，所以無法確保訂閱者會收到完整訊息群組或訊息的所有區段; 請參閱 RC2417: MQRC_MSG_NOT_ALLOWED_IN_GROUP。

如需容許應用程式要求將訊息群組全部配置給叢集佇列的相同目的地實例的相關資訊，請參閱 DefBind。

放置並取得跨越工作單元的群組

在前一種情況下，訊息或區段無法開始離開節點 (如果其目的地是遠端) 或開始擷取，直到整個群組已放置並確定工作單元為止。 如果放置整個群組需要很長時間，或節點上的佇列空間受到限制，則這可能不是您想要的。 為了克服此問題，請將小組置於數個工作單元中。

如果將群組放置在多個工作單元內，則即使放置應用程式失敗，部分群組也可以確定。 因此，應用程式必須儲存每一個工作單元所確定的狀態資訊，在重新啟動之後可以使用這些資訊來回復不完整的群組。 記錄此資訊最簡單的位置是在 STATUS 佇列中。 如果已順利放置完整群組，則 STATUS 佇列是空的。

如果涉及分段，則邏輯類似。 在此情況下， **StatusInfo** 必須包括 *Offset*。

以下是將群組放入數個工作單元的範例:

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

/* First UOW */

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
StatusInfo = GroupId,MsgSeqNumber from MQMD
MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
MQCMIT

/* Next and subsequent UOWs */
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
StatusInfo = GroupId,MsgSeqNumber from MQMD
MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
MQCMIT

/* Last UOW */
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
MQCMIT
```

如果已確定所有工作單元，則已順利放置整個群組，且 STATUS 佇列是空的。 如果沒有，則必須在狀態資訊指出的點回復群組。 MQPMO_LOGICAL_ORDER 無法用於第一個放置，但之後可以使用。

重新啟動處理程序看起來如下:

```
MQGET (StatusInfo from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
if (Reason == MQRC_NO_MSG_AVAILABLE)
   /* Proceed to normal processing */
   ...

else
   /* Group was terminated prematurely */
   Set GroupId, MsgSeqNumber in MQMD to values from Status message
   PMO.Options = MQPMO_SYNCPOINT
   MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP

   /* Now normal processing is resumed.
      Assume this is not the last message */
   PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT
   MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
   MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
   StatusInfo = GroupId,MsgSeqNumber from MQMD
   MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
   MQCMIT
```

從取得應用程式中，您可能想要在整個群組到達之前開始處理群組中的訊息。 這可改善群組內訊息的回應時間，也表示整個群組不需要儲存體。 為了實現這些好處，請針對每一個訊息群組使用數個工作單元。 基於回復原因，您必須擷取工作單元內的每一則訊息。

與對應的放置應用程式一樣，這需要在確定每一個工作單元時自動在某處記錄狀態資訊。 同樣地，記錄此資訊最簡單的位置是在 STATUS 佇列上。 如果已順利處理完整群組，則 STATUS 佇列是空的。

註: 對於中間工作單元，您可以透過指定狀態佇列的每一個 MQPUT 是訊息區段 (亦即，透過設定 MQMF_SEGMENT 旗標) ，而不是為每一個工作單元放置完整新訊息，來避免來自 STATUS 佇列的 MQGET 呼叫。 在最後一個工作單元中，會將最終區段放入狀態佇列中並指定 MQMF_LAST_SEGMENT ，然後使用 MQGET 指定 MQGMO_COMPLETE_MSG 來清除狀態資訊。

在重新啟動處理期間，不使用單一 MQGET 來取得可能的狀態訊息，而是瀏覽具有 MQGMO_LOGICAL_ORDER 的狀態佇列，直到您到達最後一個區段為止 (亦即，直到未傳回進一步的區段為止)。 在重新啟動之後的第一個工作單元中，也請在放置狀態區段時明確指定偏移。

在下列範例中，我們只考量群組內的訊息，假設應用程式的緩衝區一律夠大，足以保留整個訊息 (不論訊息是否已分段)。 因此會在每一個 MQGET 上指定 MQGMO_COMPLETE_MSG。 如果涉及分段，則適用相同的原則 (在此情況下， StatusInfo 必須包括 *Offset*)。

為了簡單起見，我們假設在單一 UOW 內最多擷取 4 則訊息:

```
msgs = 0    /* Counts messages retrieved within UOW */
/* Should be no status message at this point */

/* Retrieve remaining messages in the group */
do while ( GroupStatus == MQGS_MSG_IN_GROUP )

   /* Process up to 4 messages in the group */
   GMO.Options = MQGMO_SYNCPOINT | MQGMO_WAIT
               | MQGMO_LOGICAL_ORDER
   do while ( (GroupStatus == MQGS_MSG_IN_GROUP) && (msgs < 4) )
      MQGET
      msgs = msgs + 1
      /* Process this message */
      ...
   /* end while

   /* Have retrieved last message or 4 messages  */
   /* Update status message if not last in group */
   MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
   if ( GroupStatus == MQGS_MSG_IN_GROUP )
      StatusInfo = GroupId,MsgSeqNumber from MQMD
      MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
   MQCMIT
   msgs = 0
/* end while

if ( msgs > 0 )
   /* Come here if there was only 1 message in the group */
   MQCMIT
```

如果已確定所有工作單元，則已順利擷取整個群組，且 STATUS 佇列是空的。 如果沒有，則必須在狀態資訊指出的點回復群組。 MQGMO_LOGICAL_ORDER 無法用於第一次擷取，但隨後可以使用。

重新啟動處理程序看起來如下:

```
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
if (Reason == MQRC_NO_MSG_AVAILABLE)
   /* Proceed to normal processing */
   ...

else
   /* Group was terminated prematurely */
   /* The next message on the group must be retrieved by matching
      the sequence number and group ID with those retrieved from the
      status information. */
   GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT | MQGMO_WAIT
   MQGET GMO.MatchOptions = MQMO_MATCH_GROUP_ID | MQMO_MATCH_MSG_SEQ_NUMBER,
         MQMD.GroupId      = value from Status message,
         MQMD.MsgSeqNumber = value from Status message plus 1
   msgs = 1
   /* Process this message */
   ...

   /* Now normal processing is resumed */
   /* Retrieve remaining messages in the group */
   do while ( GroupStatus == MQGS_MSG_IN_GROUP )

      /* Process up to 4 messages in the group */
      GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT | MQGMO_WAIT
                  | MQGMO_LOGICAL_ORDER
      do while ( (GroupStatus == MQGS_MSG_IN_GROUP) && (msgs < 4) )
         MQGET
         msgs = msgs + 1
         /* Process this message */
         ...
```

```
/* Have retrieved last message or 4 messages  */
/* Update status message if not last in group */
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
if ( GroupStatus == MQGS_MSG_IN_GROUP )
   StatusInfo = GroupId,MsgSeqNumber from MQMD
   MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
MQCMIT
msgs = 0
```

取得特定訊息

有許多方法可從佇列取得特定訊息。 這些包括: 在 MsgId 和 CorrelId 上選取，在 GroupId 上選取 MsgSeq 數字和偏移，並在 MsgToken 上選取。 您也可以在開啟佇列時使用選擇字串。

若要從佇列取得特定訊息，請使用 MQMD 結構的 MsgId 及 CorrelId 欄位。 不過，應用程式可以明確設定這些欄位，因此您指定的值可能無法識別唯一訊息。 第 657 頁的表 118 顯示針對這些欄位的可能設定所擷取的訊息。 如果在 MQGET 呼叫的 **GetMsgOpts** 參數中指定 MQGMO_MSG_UNDER_CURSOR，則輸入時會忽略這些欄位。

表 *118:* 使用訊息及相關性 *ID*

擷取 ...	**MsgId**	**CorrelId**
佇列中的第一個訊息	MQMI_NONE	MQCI_NONE
符合 *MsgId* 的第一個訊息	非零	MQCI_NONE
符合 *CorrelId* 的第一個訊息	MQMI_NONE	非零
同時符合 *MsgId* 和 *CorrelId* 的第一個訊息	非零	非零

在每一種情況下， 第一個 表示第一個滿足選取準則的訊息 (除非指定 MQGMO_BROWSE_NEXT，當它表示序列中滿足選取準則的 下一個 訊息時)。

傳回時， MQGET 呼叫會將 *MsgId* 和 *CorrelId* 欄位設為所傳回訊息的訊息及相關性 ID (如果有的話)。

如果您將 MQMD 結構的 *Version* 欄位設為 2，則可以使用 *GroupId*、 *MsgSeqNumber* 及 *Offset* 欄位。 第 657 頁的表 119 顯示針對這些欄位的可能設定所擷取的訊息。

表 *119:* 使用群組 *ID*

擷取 ...	**符合選項**
佇列中的第一個訊息	MQMO_NONE
符合 MsgId 的第一個訊息	MQMO_MATCH_MSG_ID
符合 CorrelId 的第一個訊息	MQMO_MATCH_CORREL_ID
符合 GroupId 的第一個訊息	MQMO_MATCH_GROUP_ID
符合 MsgSeqNumber 的第一個訊息	MQMO_MATCH_MSG_SEQ_NUMBER
符合 MsgToken 的第一個訊息	MQMO_MATCH_MSG_TOKEN
符合 Offset 的第一個訊息	MQMO_MATCH_OFFSET

附註:

1. MQMO_MATCH_XXX 表示 MQMD 結構中的 *XXX* 欄位設為要比對的值。
2. MQMO 旗標可以組合使用。 例如， MQMO_MATCH_GROUP_ID、MQMO_MATCH_MSG_SEQ_NUMBER 及 MQMO_MATCH_OFFSET 可以一起使用，以提供 GroupId、 MsgSeqNumber 及 Offset 欄位所識別的區段。
3. 如果您指定 MQGMO_LOGICAL_ORDER，則會影響您嘗試擷取的訊息，因為該選項取決於佇列控點所控制的狀態資訊。 如需此作業的相關資訊，請參閱 第 648 頁的『邏輯和實體排序』 及 選項。

MQGET 呼叫通常會從佇列中擷取第一個訊息。 如果您在使用 MQGET 呼叫時指定特定訊息，佇列管理程式必須搜尋佇列，直到找到該訊息為止。 這可能會影響應用程式的效能。

如果您使用 MQGMO 結構第 2 版或更新版本，且未指定 MQMO_MATCH_MSG_ID 或 MQMO_MATCH_CORREL_ID 旗標，則不需要重設 MQGET 之間的 `MsgId` 或 `CorrelId` 欄位。

z/OS 在 IBM MQ for z/OS 上，佇列屬性 `IndexType` 可用來增加佇列上 MQGET 作業的速度。 如需相關資訊，請參閱 第 661 頁的『索引類型』。

您可以從佇列取得特定訊息，方法是在 MQGMO 結構中指定其 MsgToken 及 MatchOption MQMO_MATCH_MSG_TOKEN。 MsgToken 由最初將該訊息放置在佇列上的 MQPUT 呼叫或先前的 MQGET 作業傳回，並且除非重新啟動佇列管理程式，否則保持不變。

如果您只對佇列上的訊息子集感興趣，則可以搭配使用選項字串與 MQOPEN 或 MQSUB 呼叫來指定要處理哪些訊息。 然後 MQGET 會擷取滿足該選取字串的下一則訊息。 如需選取字串的相關資訊，請參閱 第 26 頁的『選取器』。

增進非持續訊息的效能

當用戶端需要來自伺服器的訊息時，它會將要求傳送至伺服器。 它會針對所耗用的每一個訊息傳送個別要求。 若要透過避免必須傳送這些要求訊息來改善耗用非持續訊息之用戶端的效能，可以將用戶端配置為使用先讀。 先讀容許將訊息傳送至用戶端，而無需應用程式要求它們。

當啟用先讀時，訊息會傳送至用戶端上稱為 先讀緩衝區的記憶體緩衝區。 對於已開啟且啟用先讀的每一個佇列，用戶端將具有先讀緩衝區。 先讀緩衝區中的訊息不會持續保存。 用戶端會定期以伺服器所耗用資料量的相關資訊來更新伺服器。

當呼叫 MQOPEN 與 MQOO_READ_AHEAD 時，IBM MQ 用戶端只會在符合特定條件時啟用先讀。 這些條件包括:

- 用戶端應用程式必須與執行緒 IBM MQ MQI 用戶端程式庫進行編譯及鏈結。
- 用戶端通道必須使用 TCP/IP 通訊協定。
- 該通道必須在用戶端和伺服器通道定義中都具有非零 SharingConversations (SHARECNV) 設定。

使用先讀可以改善耗用來自用戶端應用程式的非持續訊息時的效能。 此效能改進同時適用於 MQI 及 JMS 應用程式。 當使用非持續訊息時，使用 MQGET 或非同步使用的用戶端應用程式將受益於效能改進。

並非所有用戶端應用程式設計都適合使用先讀，因為並非所有選項都支援與先讀搭配使用，而且在啟用先讀時，部分選項在 MQGET 呼叫之間必須一致。 如果用戶端在 MQGET 呼叫之間變更其選取準則，則儲存在先讀緩衝區中的訊息仍會停留在用戶端先讀緩衝區中。

如果不再需要具有先前選取準則的滯留訊息待辦事項，則可以在用戶端上設定可配置的清除間隔，以自動從用戶端清除這些訊息。 清除間隔是用戶端所決定的一組先讀調整選項之一。 可以調整這些選項以符合您的需求。

如果用戶端應用程式重新啟動，則先讀緩衝區中的訊息可能會遺失。 相反地，已移至先讀緩衝區的訊息隨後可以從基礎佇列中刪除; 這不會導致將它從緩衝區中移除，因此使用先讀的 MQGET 呼叫可以傳回不再存在的訊息。

只對用戶端連結執行先讀。 所有其他連結都會忽略這個屬性。

先讀對觸發沒有影響。 當用戶端先讀取訊息時，不會產生觸發訊息。 先讀啟用時，不會產生結算及統計資料資訊。

搭配使用先讀與發佈訂閱傳訊

當訂閱應用程式指定要傳送發佈的目的地佇列時，會使用指定佇列的 DEFREADA 值作為預設先讀值。

當訂閱應用程式要求由 IBM MQ 管理發佈資訊傳送目的地時，會根據預先定義的模型佇列，將受管理佇列建立為動態佇列。 它是用作預設先讀值之模型佇列的 DEFREADA 值。 預設模型佇列 SYSTEM.DURABLE.PUBLICATIONS.MODEL 或 SYSTEM.NONDURABLE.PUBLICATIONS.MODEL 。

相關概念

第 660 頁的『在 AIX 上調整非持續訊息的效能』
如果您是使用 AIX V5.3 或更新版本，請考量將調整參數設為對非持續訊息使用完整效能。

相關工作

第 660 頁的『啟用及停用先讀』
依預設，會停用先讀。 您可以在佇列或應用程式層次啟用先讀。

相關參考

第 659 頁的『MQGET 選項及先讀』
當啟用先讀時，並非所有 MQGET 選項都受支援; 部分選項需要在 MQGET 呼叫之間保持一致。

MQGET 選項及先讀

當啟用先讀時，並非所有 MQGET 選項都受支援; 部分選項需要在 MQGET 呼叫之間保持一致。

當呼叫 MQOPEN 與 MQOO_READ_AHEAD 時，IBM MQ 用戶端只會在符合特定條件時啟用先讀。 這些條件包括:

- 用戶端應用程式必須與執行緒 IBM MQ MQI 用戶端程式庫進行編譯及鏈結。
- 用戶端通道必須使用 TCP/IP 通訊協定。
- 該通道必須在用戶端和伺服器通道定義中都具有非零 SharingConversations (SHARECNV) 設定。

下表指出支援搭配先讀使用哪些選項，以及是否可以在 MQGET 呼叫之間變更這些選項。

表 *120: MQGET* 選項及先讀

MQGET 值和選項	**啟用先讀且可在 MQGET 呼叫之間變更時允許 [5]**	**已啟用先讀但無法在 MQGET 呼叫之間變更時允許 [1]**	**啟用先讀時不允許的 MQGET 選項 [2]**
MQGET MQMD 值	MsgId [3] CorrelId [3]	編碼 CodedCharSetId	
MQGET MQGMO 選項	• MQGMO_NO_WAIT • MQGMO_BROWSE_MESSAGE _UNDER_CURSOR • MQGMO_BROWSE_FIRST • MQGMO_BROWSE_NEXT • MQGMO_FAIL_IF_QUIESCING	• MQGMO_SYNCPOINT_IF _PERSISTENT • MQGMO_NO_SYNCPOINT • MQGMO_ACCEPT_TRUNCATED _MSG • MQ GMO_CONVERT	• MQGMO_SET_SIGNAL • MQ GMO_同步點 • MQGMO_MARK_SKIP _BACKOUT • MQGMO_MSG_UNDER _CURSOR [4] • MQGMO_LOCK • MQGMO_UNLOCK • MQ GMO_LOGICAL_ORDER • MQ GMO_COMPLETE_MSG • MQGMO_ALL_MSGS_AVAILABLE • MQGMO_ALL_SEGMENTS_ AVAILABLE

附註:

1. 如果在 MQGET 呼叫之間變更這些選項，則會傳回 MQRC_OPTIONS_CHANGED 原因碼。
2. 如果在第一個 MQGET 呼叫上指定這些選項，則會停用先讀功能。 如果在後續的 MQGET 呼叫上指定了這些選項，則會傳回原因碼 MQRC_OPTIONS_ERROR。
3. 如果用戶端應用程式在 MQGET 呼叫之間變更 MsgId 和 CorrelId 值，則具有先前值的訊息可能已傳送至用戶端，並將保留在用戶端先讀緩衝區中直到耗用 (或自動清除) 為止。
4. MQGMO_MSG_UNDER_CURSOR 不能與先讀功能一起使用。 當開啟佇列時同時指定 MQOO_BROWSE 及 MQOO_INPUT_SHARED 或 MQOO_INPUT_EXCLUSIVE 選項之一時，會停用先讀。
5. 啟用先讀時，第一個 MQGET 會決定是否要從佇列瀏覽或取得訊息。 如果用戶端應用程式接著使用 MQGET 並搭配已變更的選項，例如嘗試在起始取得之後進行瀏覽，或嘗試在起始瀏覽之後進行取得，則會傳回 MQRC_OPTIONS_CHANGED 原因碼。

如果用戶端在 MQGET 呼叫之間變更其選取準則，則用戶端應用程式不會耗用儲存在先讀緩衝區中且符合起始選取準則的訊息，且會留在用戶端先讀緩衝區中。 在用戶端先讀緩衝區包含許多停頓訊息的情況下，會失去與先讀相關聯的好處，且每一個耗用的訊息都需要對伺服器提出個別要求。 若要判定是否有效地使用先讀，您可以使用連線狀態參數 READA。

當應用程式要求時，由於第一個 MQGET 呼叫中指定的選項不相容，可能會禁止先讀。 在此狀況下，連線狀態會顯示禁止先讀。

如果由於 MQGET 的這些限制，您決定用戶端應用程式設計不適合先讀，請指定 MQOPEN 選項 MQOO_READ_AHEAD_NO。 或者，將所開啟佇列的預設先讀值設為 NO 或 DISABLED。

啟用及停用先讀
依預設，會停用先讀。 您可以在佇列或應用程式層次啟用先讀。

關於這項作業

當呼叫 MQOPEN 與 MQOO_READ_AHEAD 時，IBM MQ 用戶端只會在符合特定條件時啟用先讀。 這些條件包括:

- 用戶端應用程式必須與執行緒 IBM MQ MQI 用戶端程式庫進行編譯及鏈結。
- 用戶端通道必須使用 TCP/IP 通訊協定。
- 該通道必須在用戶端和伺服器通道定義中都具有非零 SharingConversations (SHARECNV) 設定。

若要啟用先讀:

- 若要在佇列層次配置先讀，請將佇列屬性 DEFREADA 設為 YES。
- 若要在應用程式層次配置先讀，請執行下列動作:
  - 若要儘可能使用先讀，請在 MQOPEN 函數呼叫上使用 MQOO_READ_AHEAD 選項。 如果 DEFREADA 佇列屬性已設為 DISABLED，則用戶端應用程式無法使用先讀。
  - 若要僅在佇列上啟用先讀時使用先讀，請在 MQOPEN 函數呼叫上使用 MQOO_READ_AHEAD_AS_Q_DEF 選項。

如果用戶端應用程式設計不適合先讀，您可以停用它:

- 在佇列層次上設定佇列屬性，如果您不想要使用先讀 (除非用戶端應用程式要求)，則將 DEFREADA 設為 NO; 如果您不想要使用先讀 (read ahead)，則不論用戶端應用程式是否需要先讀。
- 在 MQOPEN 函數呼叫上使用 MQOO_NO_READ_AHEAD 選項。

兩個 MQCLOSE 選項可讓您配置在佇列關閉時，儲存在先讀緩衝區中的任何訊息會發生什麼情況。

- 使用 MQCO_IMMEDIATE 來捨棄先讀緩衝區中的訊息。
- 使用 MQCO_QUIESCE 可確保在關閉佇列之前，應用程式會耗用先讀緩衝區中的訊息。 當發出具有 MQCO_QUIESCE 的 MQCLOSE 且先讀緩衝區中有剩餘訊息時， MQRC_READ_AHEAD_MSGS 會傳回 MQCC_WARNING。

在 *AIX* 上調整非持續訊息的效能
如果您是使用 AIX V5.3 或更新版本，請考量將調整參數設為對非持續訊息使用完整效能。

若要設定調整參數以使其立即生效，請以 root 使用者身分發出下列指令:

```
/usr/sbin/ioo -o j2_nPagesPerWriteBehindCluster=0
```

若要設定調整參數，使其立即生效並在重新開機時持續保存，請以 root 使用者身分發出下列指令:

```
/usr/sbin/ioo -p -o j2_nPagesPerWriteBehindCluster=0
```

一般而言，非持續訊息只會保留在記憶體中，但在某些情況下， AIX 可以排定將非持續訊息寫入磁碟。 在磁碟寫入完成之前， MQGET 無法使用排定要寫入磁碟的訊息。 建議的調整指令會改變此臨界值; 只有在機器上的實際儲存體接近已滿時，才會進行寫入磁碟，而不是將訊息排定在 16 KB 資料排入佇列時寫入磁碟。 這是廣域變更，可能影響其他軟體元件。

在 AIX 上，當使用多執行緒應用程式時，尤其是在具有多個處理器的機器上執行時，我們強烈建議在啟動應用程式之前，先在 mqm ID `.profile` 中設定 AIXTHREAD_SCOPE=S 或在環境中設定 AIXTHREAD_SCOPE=S，以取得更好的效能及更可靠的排程。 例如：

```
export AIXTHREAD_SCOPE=S
```

設定 AIXTHREAD_SCOPE=S 表示以預設屬性建立的使用者執行緒會置於全系統競爭範圍中。 如果使用者執行緒是使用全系統競用範圍建立的，則它會連結至核心執行緒，並由核心排程。 基礎核心執行緒不會與任何其他使用者執行緒共用。

檔案描述子

執行多執行緒處理程序 (例如代理程式處理程序) 時，您可能會達到檔案描述子的軟性限制。 此限制提供 IBM MQ 原因碼 MQRC_UNEXPECTED_ERROR （2195），以及 IBM MQ FFST™ 檔案 (如果有足夠的檔案描述子)。

若要避免此問題，您可以增加檔案描述子數目的處理程序限制。 若要這樣做，請針對 mqm 使用者 ID 或預設段落將 `/etc/security/limits` 中的 `nofiles` 屬性變更為 10,000。

系統資源限制

在命令提示字元中使用下列指令，將資料區段及堆疊區段的系統資源限制設為無限制：

```
ulimit -d unlimited
ulimit -s unlimited
```

索引類型

佇列屬性 *IndexType* 指定佇列管理程式所維護的索引類型，以增加佇列上 MQGET 作業的速度。

註： 僅在 IBM MQ for z/OS 上受支援。

您有五個選項：

值	說明
無	未維護任何索引。 當循序擷取訊息時，請使用此選項 (請參閱 第 647 頁的『優先順序』)。
GROUPID	會維護群組 ID 的索引。 如果您想要訊息群組的邏輯排序，則必須使用此索引類型 (請參閱 第 648 頁的『邏輯和實體排序』)。
MSGID	會維護訊息 ID 的索引。 使用 *MsgId* 欄位作為 MQGET 呼叫的選取準則來擷取訊息時，請使用此選項 (請參閱 第 657 頁的『取得特定訊息』)。
MsgToken	會維護訊息記號的索引。
CorrelId	會維護相關性 ID 的索引。 使用 *CorrelId* 欄位作為 MQGET 呼叫的選取準則來擷取訊息時，請使用此選項 (請參閱 第 657 頁的『取得特定訊息』)。

註：

1. 如果您使用 MSGID 選項或 CORRELID 選項來編製索引，請在 MQMD 中設定相對 **MsgId** 或 **CorrelId** 參數。 兩者都設定是沒有好處的。
2. 如果佇列符合下列所有條件，則瀏覽會使用索引機制來尋找訊息：
   - 它具有索引類型 MSGID、CORRELID 或 GROUPID
   - 它是使用相同類型的 ID 來瀏覽
   - 它只有一個優先順序的訊息
3. 避免佇列 (由 *MsgId* 或 *CorrelId* 編製索引) 包含數千則訊息，因為這會影響重新啟動時間。 (這不適用於非持續訊息，因為它們會在重新啟動時刪除。)

4. MSGTOKEN 用來定義 z/OS 工作量管理程式所管理的佇列。

如需 **IndexType** 屬性的完整說明，請參閱 IndexType。 如需 **IndexType** 屬性的進一步資訊，請參閱 第 54 頁的『z/OS 應用程式的設計和效能考量』。

處理長度大於 *4 MB* 的訊息

對於應用程式、佇列或佇列管理程式而言，訊息可能太大。 視環境而定， IBM MQ 提供多種方法來處理超過 4 MB 的訊息。

在 V6 或更新版本的所有 IBM MQ 系統上，您可以增加 **MaxMsgLength** 屬性，最多 100 MB。 設定此值以反映使用佇列的訊息大小。 在 IBM MQ for z/OS 以外的 IBM MQ 系統上，您也可以:

1. 使用分段訊息。 (訊息可以由應用程式或佇列管理程式分段。)
2. 使用參照訊息。

本節的其餘部分將說明這些方法中的每一種。

增加訊息長度上限

MaxMsgLength 佇列管理程式屬性定義佇列管理程式可處理的訊息長度上限。 同樣地， **MaxMsgLength** 佇列屬性是佇列可處理的訊息長度上限。 支援的預設訊息長度上限視您工作所在的環境而定。

如果您要處理大型訊息，則可以在 z/OS 以外的平台上獨立變更這些屬性。 您可以設定 32768 個位元組到 100 MB 範圍內的佇列管理程式屬性值。

小心: 在 IBM MQ for z/OS 上，佇列管理程式的 **MaxMsgLength** 屬性會寫在 100 MB 的程式中。

在所有平台上，您可以設定在 0 到 100 MB 範圍內的佇列屬性值。

變更一個或兩個 **MaxMsgLength** 屬性之後，請重新啟動應用程式及通道，以確保變更生效。

進行這些變更時，訊息長度必須小於或等於佇列及佇列管理程式 **MaxMsgLength** 屬性。 不過，現有訊息的長度可能超過任一屬性。

如果訊息對佇列而言太大，則會傳回 MQRC_MSG_TOO_BIG_FOR_Q。 同樣地，如果訊息對佇列管理程式而言太大，則會傳回 MQRC_MSG_TOO_BIG_FOR_Q_MGR。

這種處理大訊息的方法簡單方便。 不過，在使用之前，請考量下列因素:

- 佇列管理程式之間的一致性會降低。 訊息資料的大小上限由將放置訊息之每一個佇列 (包括傳輸佇列) 的 *MaxMsgLength* 決定。 此值通常預設為佇列管理程式的 *MaxMsgLength*，尤其是傳輸佇列。 當訊息要傳送至遠端佇列管理程式時，這會造成難以預測訊息是否太大。
- 系統資源的用量會增加。 例如，應用程式需要較大的緩衝區，而在部分平台上，共用儲存體的用量可能會增加。 只有在實際需要較大訊息時，才應該影響佇列儲存體。
- 通道批次處理受影響。 大型訊息仍只會視為批次計數的一則訊息，但需要較長的傳輸時間，因此會增加其他訊息的回應時間。

Multi 訊息分段

使用此資訊來瞭解分段訊息的相關資訊。 IBM MQ for z/OS 或使用 IBM MQ classes for JMS 的應用程式不支援此特性。

增加訊息長度上限 (如主題 第 662 頁的『增加訊息長度上限』 中所說明) 有一些負面影響。 此外，它仍可能導致訊息對佇列或佇列管理程式而言太大。 在這些情況下，您可以將訊息分段。 如需區段的相關資訊，請參閱 第 37 頁的『訊息群組』。

接下來的區段會查看分段訊息的一般用途。 對於放置及破壞性取得，假設 MQPUT 或 MQGET 呼叫 一律 在工作單元內運作。 請一律考慮使用此技術，以減少網路中呈現不完整群組的可能性。 採用佇列管理程式的單階段確定，但其他協調技術同樣有效。

此外，在取得應用程式中，假設如果有多部伺服器處理相同的佇列，則每一部伺服器都會執行類似的程式碼，因此一部伺服器永遠不會找不到它預期會出現的訊息或區段 (因為先前已指定 MQGMO_ALL_MSGS_AVAILABLE 或 MQGMO_ALL_SEGMENTS_AVAILABLE)。

小心: 使用發佈/訂閱將訊息傳送至主題 (或將訊息放置至主題別名) 時，不允許進行訊息分組和分段。

因為訂閱可以獨立於發佈活動來建立及移除，所以無法確保訂閱者會收到完整訊息群組或訊息的所有區段; 請參閱 RC2417: MQRC_MSG_NOT_ALLOWED_IN_GROUP。

放置並取得跨越工作單元的分段訊息

您可以使用類似於 第 654 頁的『放置並取得跨越工作單元的群組』的方式，放置並取得跨越工作單元的分段訊息。

不過，您無法在廣域工作單元中放置或取得分段的訊息。

Multi 依佇列管理程式進行分段及重組

這是最簡單的實務範例，其中一個應用程式放置要由另一個應用程式擷取的訊息。 訊息可能很大: 對於單一緩衝區中的放置或取得應用程式來說，不會太大，但對於要放置訊息的佇列管理程式或佇列來說，會太大。

這些應用程式唯一需要的變更是讓放置應用程式授權佇列管理程式在必要時執行分段:

```
   PMO.Options = (existing options)
   MD.MsgFlags = MQMF_SEGMENTATION_ALLOWED
   MD.Version = MQMD_VERSION_2
   memcpy(MD.GroupId, MQGI_NONE, MQ_GROUP_ID_LENGTH)
MQPUT
```

並讓取得應用程式要求佇列管理程式重新組合訊息 (如果它已分段的話):

```
   GMO.Options = MQGMO_COMPLETE_MSG | (existing options)
   MQGET
```

在此最簡單的實務範例中，應用程式必須在 MQPUT 呼叫之前將 GroupId 欄位重設為 MQGI_NONE ， 以便佇列管理程式可以為每一則訊息產生唯一群組 ID。 如果未執行此動作，則不相關的訊息可能具有相同的群組 ID ， 這隨後可能會導致不正確的處理。

應用程式緩衝區必須夠大，才能包含重新組合的訊息 (除非您包含 MQGMO_ACCEPT_TRUNCATED_MSG 選項)。

如果要修改佇列的 MAXMSGLEN 屬性以容納訊息分段，請考量:

- 本端佇列上支援的訊息區段下限為 16 個位元組。
- 對於傳輸佇列， MAXMSGLEN 也必須包括標頭所需的空間。 請考慮在任何可放入傳輸佇列的訊息區段中使用至少 4000 個位元組的值，以大於預期的使用者資料長度上限。

如果需要資料轉換，則取得應用程式可能必須透過指定 MQGMO_CONVERT 來執行。 這應該直接明確，因為資料轉換結束程式會呈現完整訊息。 如果訊息已分段，且資料格式使得資料轉換結束程式無法對不完整的資料執行轉換，請不要嘗試轉換傳送端通道中的資料。

Multi 應用程式分段

當佇列管理程式分段不足時，或當應用程式需要具有特定區段界限的資料轉換時，會使用應用程式分段。

使用應用程式分段有兩個主要原因:

1. 因為訊息太大而無法由應用程式在單一緩衝區中處理，所以僅佇列管理程式分段並不足夠。
2. 資料轉換必須由傳送端通道執行，且格式為放置應用程式必須規定區段界限的位置，才能進行個別區段的轉換。

不過，如果資料轉換不是問題，或者如果取得應用程式一律使用 MQGMO_COMPLETE_MSG ，則也可以透過指定 MQMF_SEGMENTATION_ALLOWED 來容許佇列管理程式分段。 在我們的範例中，應用程式會將訊息分段為四個區段:

```
   PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

   MQPUT MD.MsgFlags = MQMF_SEGMENT
```

```
MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_SEGMENT

MQCMIT
```

如果您不使用 MQPMO_LOGICAL_ORDER，應用程式必須設定 *Offset* 及每一個區段的長度。 在此情況下，不會自動維護邏輯狀態。

取得應用程式無法保證具有足夠大的緩衝區來保留任何重新組合的訊息。 因此，它必須準備個別處理客群。

對於分段的訊息，此應用程式不想要開始處理一個區段，直到存在構成邏輯訊息的所有區段為止。 因此，會針對第一個區段指定 MQGMO_ALL_SEGMENTS_AVAILABLE。 如果您指定 MQGMO_LOGICAL_ORDER 且有現行邏輯訊息，則會忽略 MQGMO_ALL_SEGMENTS_AVAILABLE。

擷取邏輯訊息的第一個區段之後，請使用 MQGMO_LOGICAL_ORDER 來確保依序擷取邏輯訊息的其餘區段。

不會考量不同群組內的訊息。 如果發生這類訊息，則會依每則訊息的第一個區段在佇列上的順序來處理它們。

```
GMO.Options = MQGMO_SYNCPOINT | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_SEGMENTS_AVAILABLE | MQGMO_WAIT
do while ( SegmentStatus == MQSS_SEGMENT )
   MQGET
   /* Process each remaining segment of the logical message */
   ...

MQCMIT
```

Multi 邏輯訊息的應用程式分段

訊息必須以群組中的邏輯順序維護，且部分或全部可能太大而需要應用程式分段。

在我們的範例中，將放置一組由四個邏輯訊息組成的群組。 除了第三則訊息以外，所有訊息都很大，需要分段，由放置應用程式執行:

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_LAST_SEGMENT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_LAST_SEGMENT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP

MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_LAST_SEGMENT

MQCMIT
```

在取得應用程式中，在第一個 MQGET 上指定 MQGMO_ALL_MSGS_AVAILABLE。 這表示在整個群組可用之前，不會擷取群組的訊息或區段。 已擷取群組的第一個實體訊息時，會使用 MQGMO_LOGICAL_ORDER 來確保依序擷取群組的區段及訊息:

```
GMO.Options = MQGMO_SYNCPOINT | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_MSGS_AVAILABLE | MQGMO_WAIT

do while ( (GroupStatus   != MQGS_LAST_MSG_IN_GROUP) ||
             (SegmentStatus != MQGS_LAST_SEGMENT) )
   MQGET
   /* Process a segment or complete logical message.  Use the GroupStatus
      and SegmentStatus information to see what has been returned */
   ...

MQCMIT
```

註: 如果您指定 MQGMO_LOGICAL_ORDER 且有現行群組，則會忽略 MQGMO_ALL_MSGS_AVAILABLE。

參照訊息
使用此資訊來進一步瞭解參照訊息。

註: 在 IBM MQ for z/OS 中不受支援。

此方法容許將大型物件從一個節點傳送至另一個節點，而無需將物件儲存在來源節點或目的地節點上的 IBM MQ 佇列上。 當資料以其他形式 (例如，郵件應用程式) 存在時，這會有特別的好處。

若要這麼做，您可以在通道兩端指定訊息結束程式。 如需如何執行此作業的相關資訊，請參閱第 820 頁的『通道訊息結束程式』。

IBM MQ 定義參照訊息標頭 (MQRMH) 的格式。 如需此的說明，請參閱 MQRMH 。 這是以已定義的格式名稱來辨識，後面可能接著實際資料。

若要起始大型物件的傳送，應用程式可以放置由參照訊息標頭組成的訊息，後面沒有資料。 當此訊息離開節點時，訊息結束程式會以適當的方式擷取物件，並將它附加至參照訊息。 然後，它會將訊息 (現在比之前更大) 傳回給傳送端「訊息通道代理程式」，以傳輸至接收端 MCA。

在接收 MCA 時配置另一個訊息結束程式。 當此訊息結束程式接收其中一則訊息時，它會使用附加的物件資料來建立物件，並在 沒有 它的情況下傳遞參照訊息。 應用程式現在可以接收參照訊息，且此應用程式知道已在此節點上建立物件 (或至少此參照訊息所代表的部分)。

傳送訊息結束程式可附加至參照訊息的物件資料量上限，受限於通道的協議訊息長度上限。 對於所傳遞的每一則訊息，結束程式只能傳回單一訊息給 MCA ，因此放置應用程式可以放置數個訊息，以導致傳送一個物件。 每一則訊息必須識別要附加至其中的物件的 邏輯 長度及偏移。 不過，在無法知道物件大小總計或通道所容許的大小上限的情況下，請設計傳送訊息結束程式，以便放置應用程式只會放置單一訊息，而結束程式本身會在將盡可能多的資料附加至已傳遞的訊息時，將下一個訊息放置在傳輸佇列上。

在使用這種處理大型訊息的方法之前，請考量下列要點:

- MCA 和訊息結束程式以 IBM MQ 使用者 ID 執行。 訊息結束程式 (因此，使用者 ID) 需要存取物件，以在傳送端擷取它，或在接收端建立它; 這可能只有在物件可普遍存取的情況下才可行。 這會產生安全問題。
- 如果附加大量資料的參照訊息必須先通過數個佇列管理程式，然後才能到達其目的地，則大量資料會呈現在中間節點的 IBM MQ 佇列上。 不過，在這些情況下，不需要提供特別支援或退出。
- 如果容許重新遞送或無法傳送郵件的佇列作業，則很難設計您的訊息結束程式。 在這些情況下，物件的部分可能不正常到達。
- 當參照訊息到達其目的地時，接收訊息結束程式會建立物件。 不過，這與 MCA 的工作單元不同步，因此如果取消批次，另一則包含物件相同部分的參照訊息會在稍後批次中送達，且訊息結束程式可能會嘗試重建物件的相同部分。 例如，如果物件是一系列資料庫更新項目，則這可能無法接受。 若是如此，訊息結束程式必須保留已套用更新項目的日誌; 這可能需要使用 IBM MQ 佇列。
- 視物件類型的性質而定，訊息結束程式和應用程式可能需要合作維護使用計數，以便在不再需要物件時可以刪除該物件。 也可能需要實例 ID; 會在參照訊息標頭中提供此 ID 的欄位 (請參閱 MQRMH)。
- 如果將參照訊息放置為配送清單，則必須針對該節點上每一個產生的配送清單或個別目的地可擷取物件。 您可能需要維護使用計數。 另請考量節點可能是清單中部分目的地的最終節點，但可能是其他目的地的中間節點。
- 通常不會轉換大量資料。 這是因為在呼叫訊息結束程式 之前 會進行轉換。 基於此原因，不得在原始傳送端通道上要求轉換。 如果參照訊息通過中間節點，則在從中間節點傳送大量資料時 (如果要求的話) 會轉換大量資料。
- 無法分段參照訊息。

使用 MQRMH 和 MQMD 結構

如需參照訊息標頭及訊息描述子中欄位的說明，請參閱 MQRMH 及 MQMD 。

在 MQMD 結構中，將 *Format* 欄位設為 MQFMT_REF_MSG_HEADER。 在 MQGET 上要求 MQHREF 格式時， IBM MQ 會自動轉換 MQHREF 格式以及隨後的任何大量資料。

以下是 MQRMH 的 *DataLogicalOffset* 和 *DataLogicalLength* 欄位的使用範例:

放置應用程式可能會放置含有下列項目的參照訊息:

- 無實體資料
- *DataLogicalLength* = 0 (此訊息代表整個物件)
- *DataLogicalOffset* = 0.

假設物件長度為 70000 個位元組，傳送訊息結束程式會在包含下列內容的參照訊息中，沿著通道傳送前 40000 個位元組:

- MQRMH 之後 40 000 個位元組的實體資料
- *DataLogicalLength* = 40000
- *DataLogicalOffset* = 0 (從物件開始)。

然後，它會將另一則訊息放置在傳輸佇列中，其中包含:

- 無實體資料
- *DataLogicalLength* = 0 (至物件結尾)。 您可以在這裡指定值 30 000。
- *DataLogicalOffset* = 40000 (從此點開始)。

當傳送訊息結束程式看到此訊息結束程式時，會附加剩餘 30,000 個位元組的資料，且欄位會設為:

- MQRMH 之後 30,000 位元組的實體資料
- *DataLogicalLength* = 30000
- *DataLogicalOffset* = 40000 (從此點開始)。

也會設定 MQRMHF_LAST 旗標。

如需提供供參照訊息使用之程式範例的說明，請參閱 第 888 頁的『在 Multiplatforms 上使用範例程式』。

等待訊息

如果您想要程式等待訊息到達佇列，請在 MQGMO 結構的 *Options* 欄位中指定 MQGMO_WAIT 選項。

使用 MQGMO 結構的 *WaitInterval* 欄位來指定 您要 MQGET 呼叫等待訊息抵達佇列的時間上限 (毫秒)。

如果訊息未在此時間內送達，則 MQGET 呼叫會以 MQRC_NO_MSG_AVAILABLE 原因碼完成。

您可以使用常數 MQWI_UNLIMITED 在 *WaitInterval* 欄位中指定無限制等待間隔。 不過，超出您控制的事件可能會導致您的程式等待很長時間，因此請小心使用此常數。 IMS 應用程式不得指定無限制等待間隔，因為這會防止 IMS 系統終止。 (當 IMS 終止時，它需要結束所有相依區域。) 相反地， IMS 應用程式可以指定有限等待間隔; 然後，如果呼叫完成而未在該間隔之後擷取訊息，請使用 wait 選項發出另一個 MQGET 呼叫。

註: 如果有多個程式在相同的共用佇列上等待 移除 訊息，則訊息到達時只會啟動一個程式。 不過，如果有多個程式正在等待瀏覽訊息，則可以啟動所有程式。 如需相關資訊，請參閱 MQGMO 中 MQGMO 結構之 *Options* 欄位的說明。

如果佇列或佇列管理程式的狀態在等待間隔到期之前變更，則會發生下列動作:

- 如果佇列管理程式進入靜止狀態，且您已使用 MQGMO_FAIL_IF_QUIESCING 選項，則會取消等待，並以 MQRC_Q_MGR_QUIESCING 原因碼完成 MQGET 呼叫。 如果沒有此選項，則通話仍在等待中。
- z/OS 在 z/OS 上，如果連線 (適用於 CICS 或 IMS 應用程式) 進入靜止狀態，且您使用 MQGMO_FAIL_IF_QUIESCING 選項，則會取消等待，且 MQGET 呼叫會完成，並傳回 MQRC_CONN_QUIESCING 原因碼。 如果沒有此選項，則通話仍在等待中。
- 如果強制停止或取消佇列管理程式， MQGET 呼叫會完成 MQRC_Q_MGR_STOPPING 或 MQRC_CONNECTION_BROKEN 原因碼。
- 如果佇列 (或佇列名稱解析成的佇列) 的屬性已變更，現在禁止取得要求，則會取消等待，且 MQGET 呼叫會完成，並具有 MQRC_GET_INHIBITED 原因碼。
- 如果以需要 FORCE 選項的方式變更佇列 (或佇列名稱解析成的佇列) 的屬性，則會取消等待，且 MQGET 呼叫會完成，並傳回 MQRC_OBJECT_CHANGED 原因碼。

z/OS 如果您想要應用程式在多個佇列上等待，請使用 IBM MQ for z/OS 的信號機能 (請參閱 第 667 頁的『信號 (signaling)』)。 如需發生這些動作之情況的相關資訊，請參閱 MQGMO。

信號 *(signaling)*

信號僅在 IBM MQ for z/OS 上受支援。

信號是 MQGET 呼叫的選項，容許作業系統通知 (或 信號) 當預期訊息到達佇列時的程式。 這類似於主題 第 666 頁的『等待訊息』 中說明的 *get with wait* 功能，因為它可讓您的程式在等待信號時繼續執行其他工作。 不過，如果您使用信號，則可以釋放應用程式執行緒，並依賴作業系統在訊息到達時通知程式。

設定信號

若要設定信號，請在 MQGET 呼叫中使用的 MQGMO 結構中執行下列動作:

1. 在 *Options* 欄位中設定 MQGMO_SET_SIGNAL 選項。
2. 在 *WaitInterval* 欄位中設定信號的生命期限上限。 這會設定您要 IBM MQ 監視佇列的時間長度 (毫秒)。 請使用 MQWI_UNLIMITED 值來指定無限制生命期限。

   註: IMS 應用程式不得指定無限制等待間隔，因為這會防止 IMS 系統終止。 (當 IMS 終止時，它需要結束所有相依區域。) 相反地， IMS 應用程式可以定期檢查 ECB 的狀態 (請參閱步驟 3)。 程式可以同時在數個佇列控點上設定信號:
3. 在 *Signal1* 欄位中指定 事件控制區塊 (ECB) 的位址。 這會通知您信號的結果。 在佇列關閉之前， ECB 儲存體必須保持可用。

註: 您無法搭配 MQGMO_WAIT 選項使用 MQGMO_SET_信號選項。

訊息到達時

當適當的訊息到達時，會將完成碼傳回給 ECB。

完成碼說明下列其中一項:

- 您為其設定信號的訊息已到達佇列。 訊息未保留給要求信號的程式，因此程式必須重新發出 MQGET 呼叫來取得訊息。

  註: 另一個應用程式可以在您接收信號與發出另一個 MQGET 呼叫之間的時間內取得訊息。
- 您設定的等待間隔已過期，且您設定信號的訊息未到達佇列。 IBM MQ 已取消信號。
- 信號已取消。 例如，如果佇列管理程式停止，或佇列的屬性已變更，則會發生此情況，因此不再容許 MQGET 呼叫。

當佇列上已有適當的訊息時， MQGET 呼叫會以與 MQGET 呼叫相同的方式完成，而不發出信號。 此外，如果立即偵測到錯誤，則會完成呼叫並設定回覆碼。

當接受呼叫且沒有立即可用的訊息時，會將控制權傳回給程式，以便它可以繼續其他工作。 訊息描述子中未設定任何輸出欄位，但 **CompCode** 參數設為 MQCC_WARNING ，且 **Reason** 參數設為 MQRC_SIGNAL_REQUEST_ACCEPTED。

如需 IBM MQ 在使用信號進行 MQGET 呼叫時可以傳回給應用程式的內容相關資訊，請參閱 MQGET。

如果程式在等待歐洲央行公佈時沒有其他工作要做，它可以等待歐洲央行使用:

- 若為 CICS Transaction Server for z/OS 程式， EXEC CICS WAIT EXTERNAL 指令
- 對於批次和 IMS 程式， z/OS WAIT 巨集

如果在設定信號時 (亦即，尚未公佈 ECB) 佇列或佇列管理程式的狀態變更，則會發生下列動作:

- 如果佇列管理程式進入靜止狀態，且您使用 MQGMO_FAIL_IF_QUIESCING 選項，則會取消信號。 ECB 會隨 MQEC_Q_MGR_QUIESCING 完成碼一起公佈。 如果沒有此選項，則信號會保持已設定狀態。
- 如果強制停止或取消佇列管理程式，則會取消信號。 信號隨 MQEC_WAIT_CANCELED 完成碼一起遞送。
- 如果佇列 (或佇列名稱解析的佇列) 的屬性已變更，現在禁止取得要求，則會取消信號。 信號隨 MQEC_WAIT_CANCELED 完成碼一起遞送。

註:

1. 如果多個程式已在相同的共用佇列上設定信號來移除訊息，則訊息到達時只會啟動一個程式。 不過，如果有多個程式正在等待瀏覽訊息，則可以啟動所有程式。 當決定要啟動哪些應用程式時，佇列管理程式所遵循的規則與等待應用程式的規則相同: 如需相關資訊，請參閱 MQGMO-Get-message 選項中 MQGMO 結構之 *Options* 欄位的說明。
2. 如果有多個 MQGET 呼叫在等待相同的訊息，並混合等待及信號選項，則會平均考慮每一個等待中呼叫。 如需相關資訊，請參閱 MQGMO-Get-message 選項中 MQGMO 結構之 *Options* 欄位的說明。
3. 在某些狀況下， MQGET 呼叫可以擷取訊息，也可以遞送信號 (來自相同訊息的到達)。 這表示當您的程式發出另一個 MQGET 呼叫 (因為已遞送信號) 時，可能沒有可用的訊息。 設計您的程式以測試此狀況。

如需如何設定信號的相關資訊，請參閱 Signal1 中 MQGMO_SET_SIGNAL 選項及 *Signal1* 欄位的說明。

跳過取消

您可以在 MQGET 呼叫上指定 **MQGMO_MARK_SKIP_BACKOUT** 選項，以防止應用程式進入 *MQGET-error-backout* 迴圈。

註: 僅在 IBM MQ for z/OS 上受支援。

作為工作單元的一部分，應用程式可以發出一個以上 MQGET 呼叫，以從佇列取得訊息。 如果應用程式偵測到錯誤，它可以退出工作單元。 這會將在該工作單元期間更新的所有資源還原至工作單元啟動之前的狀態，並恢復 MQGET 呼叫所擷取的訊息。

恢復之後，這些訊息可供應用程式所發出的後續 MQGET 呼叫使用。 在許多情況下，這不會導致應用程式發生問題。 不過，如果無法規避導致取消的錯誤，在佇列上恢復訊息可能會導致應用程式進入 *MQGET-error-backout* 迴圈。

若要避免此問題，請在 MQGET 呼叫上指定 MQGMO_MARK_SKIP_BACKOUT 選項。 這會將 MQGET 要求標示為不涉及應用程式起始的取消; 亦即，不得取消。 使用此選項表示當發生取消時，會根據需要取消其他資源的更新，但所標示的訊息會被視為已在新工作單元下擷取。

應用程式必須發出 IBM MQ 呼叫，以確定新的工作單元，或取消新的工作單元。 例如，程式可以執行異常狀況處理，例如通知發送端已捨棄訊息，並確定工作單元，以便從佇列中移除訊息，如果取消 (基於任何原因) 新的工作單元，則會在佇列上恢復訊息。

在工作單元內，只能有一個 MQGET 要求標示為跳過取消; 不過，可能有數個其他訊息未標示為跳過取消。 一旦訊息標示為跳過取消，在工作單元內指定 MQGMO_MARK_SKIP_BACKOUT 的任何進一步 MQGET 呼叫都會失敗，原因碼為 MQRC_SECOND_MARK_NOT_ALLOWED。

註:

1. 只有在應用程式要求取消包含所標示訊息的工作單元時，它才會跳過取消。 如果工作單元因任何其他原因而取消，則會以未標示為跳過取消的相同方式，將訊息取消至佇列。
2. 在參與 RRS 所控制之工作單元的 Db2 儲存程序內，不支援跳過取消。 例如，具有 MQGMO_MARK_SKIP_BACKOUT 選項的 MQGET 呼叫將失敗，原因碼為 MQRC_OPTION_ENVIRONMENT_ERROR。

第 669 頁的圖 63 說明當需要 MQGET 要求來跳過取消時，應用程式可能包含的一般步驟順序。

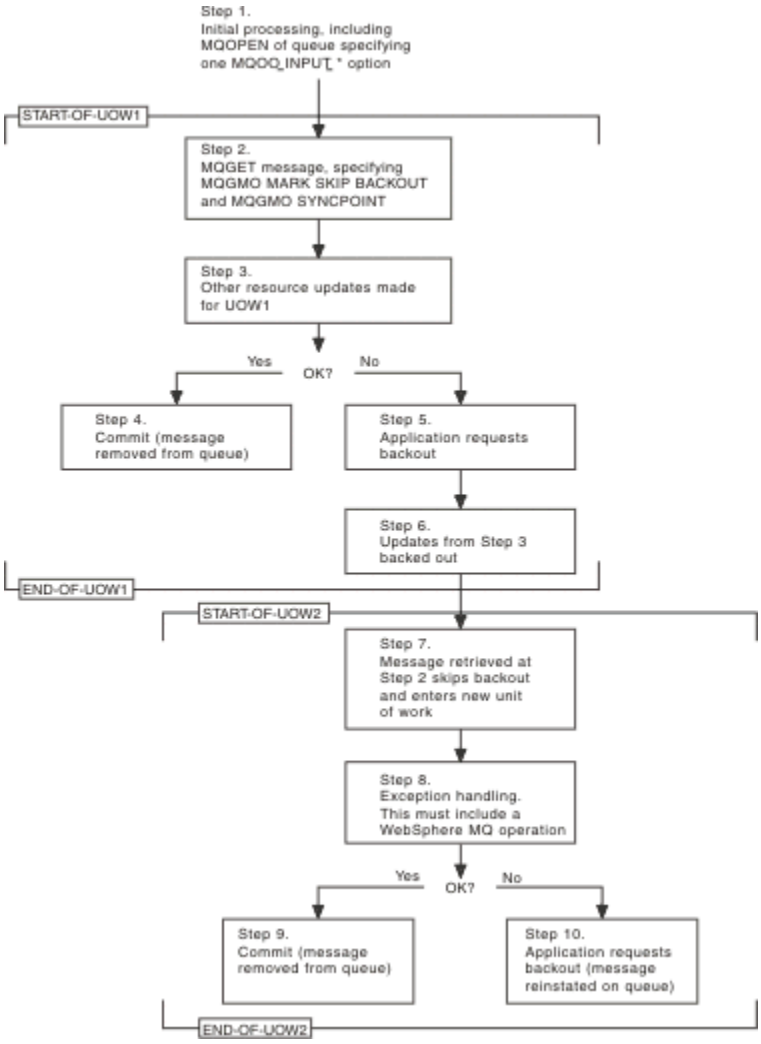


圖 *63:* 使用 *MQGMO_MARK_SKIP_BACKOUT* 跳過取消

第 669 頁的圖 63 中的步驟如下:

步驟 1
在交易內進行起始處理，包括開啟佇列的 MQOPEN 呼叫 (指定其中一個 MQOO_INPUT_ * 選項，以便在步驟 2 中從佇列取得訊息)。

步驟 2
會使用 MQGMO_SYNCPOINT 和 MQGMO_MARK_SKIP_BACKOUT 來呼叫 MQGET。 需要 MQGMO_SYNCPOINT ，因為 MQGET 必須在工作單元內， MQGMO_MARK_SKIP_BACKOUT 才會生效。 在 第 669 頁的圖 63 中，此工作單元稱為 UOW1。

步驟 3
其他資源更新是作為 UOW1 的一部分進行。 這些可以包括進一步的 MQGET 呼叫 (在沒有 MQGMO_MARK_SKIP_BACKOUT 的情況下發出)。

步驟 4
根據需要完成步驟 2 和 3 的所有更新。 應用程式會確定更新項目，且 UOW1 會結束。 從佇列中移除在步驟 2 中擷取的訊息。

步驟 5
步驟 2 和 3 的部分更新未依需要完成。 應用程式要求取消在這些步驟期間所做的更新。

步驟 6
步驟 3 中所做的更新會取消。

步驟 7
在步驟 2 中所提出的 MQGET 要求會跳過取消，並成為新工作單元 UOW2 的一部分。

步驟 8
UOW2 會執行異常狀況處理，以回應取消的 UOW1 。 (例如，對另一個佇列的 MQPUT 呼叫，指出發生問題導致取消 UOW1 。)

步驟 9
步驟 8 會根據需要完成，應用程式會確定活動，且 UOW2 會結束。 因為 MQGET 要求是 UOW2 的一部分 (請參閱步驟 7) ，此確定會導致從佇列中移除訊息。

步驟 10
步驟 8 未依需要完成，應用程式會取消 UOW2。 因為取得訊息要求是 UOW2 的一部分 (請參閱步驟 7) ，所以也會在佇列上取消並恢復它。 現在，它可用於此或另一個應用程式所發出的進一步 MQGET 呼叫 (與佇列上的任何其他訊息相同)。

應用程式資料轉換

必要時， MCA 會將訊息描述子及標頭資料轉換為必要的字集及編碼。 鏈結的任一端 (即本端 MCA 或遠端 MCA) 可以執行轉換。

當應用程式將訊息放入佇列時，本端佇列管理程式會將控制資訊新增至訊息描述子，以協助在佇列管理程式及 MCA 處理訊息時對訊息進行控制。 視環境而定，會在本端系統的字集及編碼中建立訊息標頭資料欄位。

當您在系統之間移動訊息時，有時需要將應用程式資料轉換成接收系統所需的字集和編碼。 可以從接收系統上的應用程式內或由傳送系統上的 MCA 執行此動作。 如果在接收系統上支援資料轉換，請使用應用程式來轉換應用程式資料，而不是取決於傳送系統上已發生的轉換。

當您在傳遞至 MQGET 呼叫之 MQGMO 結構的 *Options* 欄位中指定 MQGMO_CONVERT 選項時，如果 所有下列陳述式都成立，則會在應用程式內轉換應用程式資料:

- 與佇列上訊息相關聯的 MQMD 結構中設定的 *CodedCharSetId* 或 *Encoding* 欄位與 MQGET 呼叫中指定的 MQMD 結構中設定的 *CodedCharSetId* 或 *Encoding* 欄位不同。
- 與訊息相關聯的 MQMD 結構中的 *Format* 欄位不是 MQFMT_NONE。
- MQGET 呼叫中指定的 *BufferLength* 不是零。
- 訊息資料長度不是零。
- 佇列管理程式支援在與訊息及 MQGET 呼叫相關聯的 MQMD 結構中指定的 *CodedCharSetId* 與 *Encoding* 欄位之間進行轉換。 如需受支援編碼字集 ID 及機器編碼的詳細資料，請參閱 CodedCharSetId 及 Encoding 。
- 佇列管理程式支援轉換訊息格式。 如果與訊息相關聯的 MQMD 結構的 *Format* 欄位是其中一種內建格式，則佇列管理程式可以轉換訊息。 如果 *Format* 不是其中一種內建格式，您需要撰寫資料轉換結束程式來轉換訊息。

如果傳送端 MCA 要轉換資料，請在每一個需要轉換的傳送端或伺服器通道定義上指定 CONVERT (YES) 關鍵字。 如果資料轉換失敗，則訊息會傳送至傳送端佇列管理程式上的 DLQ ，且 MQDLH 結構的 *Feedback* 欄位會指出原因。 如果無法將訊息放置在 DLQ 上，則通道會關閉，且未轉換的訊息會保留在傳輸佇列上。 應用程式內的資料轉換，而不是在傳送 MCA 時的資料轉換，可避免這種狀況。

一般而言，訊息中由內建格式或資料轉換結束程式說明為 字元 資料的資料會從訊息所使用的編碼字集轉換為所要求的編碼字集，而 數值 欄位會轉換為所要求的編碼。

如需轉換內建格式時所使用轉換處理慣例的進一步詳細資料，以及撰寫您自己的資料轉換結束程式的相關資訊，請參閱 第 823 頁的『寫入資料-轉換結束程式』 。 如需語言支援表格及受支援機器編碼的相關資訊，另請參閱 國家語言 及 機器編碼 。

EBCDIC 換行字元的轉換

如果您需要確定從 EBCDIC 平台傳送至 ASCII 的資料與您再次收到的資料相同，您必須控制 EBCDIC 換行字元的轉換。

您可以使用平台相依交換器來執行此動作，該交換器會強制 IBM MQ 使用未修改的轉換表格，但您必須瞭解可能會導致的不一致行為。

發生問題的原因是 EBCDIC 換行字元未跨平台或轉換表一致地轉換。 因此，如果資料顯示在 ASCII 平台上，則格式可能不正確。 例如，這會導致難以使用 RUNMQSC 從 ASCII 平台遠端管理 IBM i 系統。

如需將 EBCDIC 格式資料轉換成 ASCII 格式的進一步相關資訊，請參閱 資料轉換 。

瀏覽佇列上的訊息

使用此資訊來瞭解如何使用 MQGET 呼叫來瀏覽佇列上的訊息。

如果要使用 MQGET 呼叫來瀏覽佇列上的訊息，請執行下列動作:

1. 指定 MQOO_BROWSE 選項，以呼叫 MQOPEN 來開啟佇列進行瀏覽。
2. 若要瀏覽佇列上的第一個訊息，請使用 MQGMO_BROWSE_FIRST 選項呼叫 MQGET。 若要尋找您想要的訊息，請使用 MQGMO_BROWSE_NEXT 選項反覆地呼叫 MQGET ，以逐步執行許多訊息。

   在每一個 MQGET 呼叫之後，您必須將 MQMD 結構的 *MsgId* 和 *CorrelId* 欄位設為空值，才能查看所有訊息。
3. 呼叫 MQCLOSE 以關閉佇列。

瀏覽游標

當您開啟 (MQOPEN) 佇列進行瀏覽時，該呼叫會建立瀏覽游標，以與使用其中一個瀏覽選項的 MQGET 呼叫搭配使用。 您可以將瀏覽游標視為位於佇列上第一個訊息之前的邏輯指標。

您可以針對相同佇列發出數個 MQOPEN 要求，讓多個瀏覽游標處於作用中 (來自單一程式)。

當您呼叫 MQGET 進行瀏覽時，請在 MQGMO 結構中使用下列其中一個選項:

MQGMO_BROWSE_FIRST
取得符合 MQMD 結構中所指定條件的第一則訊息的副本。

MQGMO_BROWSE_NEXT
取得滿足 MQMD 結構中所指定條件的下一個訊息副本。

MQGMO_BROWSE_MSG_UNDER_CURSOR
取得游標目前所指向的訊息副本，亦即，前次使用 MQGMO_BROWSE_FIRST 或 MQGMO_BROWSE_NEXT 選項所擷取的訊息副本。

無論如何，訊息都會保留在佇列上。

當您開啟佇列時，瀏覽游標會在邏輯上位於佇列上第一個訊息之前。 這表示如果您在 MQOPEN 呼叫之後立即發出 MQGET 呼叫，您可以使用 MQGMO_BROWSE_NEXT 選項來瀏覽第一個訊息; 您不需要使用 MQGMO_BROWSE_FIRST 選項。

從佇列複製訊息的順序由佇列的 **MsgDeliverySequence** 屬性決定。 (如需相關資訊，請參閱 第 647 頁的『從佇列擷取訊息的順序』。)

- 第 671 頁的『FIFO (先進先出) 順序的佇列』
- 第 671 頁的『依優先順序排列的佇列』
- 第 672 頁的『未確定的訊息』
- 第 672 頁的『變更佇列順序』
- 第 672 頁的『使用佇列的索引』

FIFO (先進先出) 順序的佇列

此順序中佇列的第一個訊息是佇列上最長的訊息。

請使用 MQGMO_BROWSE_NEXT 來循序讀取佇列中的訊息。 當您瀏覽時，您會看到任何放置在佇列中的訊息，因為此順序中的佇列會在結尾放置訊息。 當游標辨識到已到達佇列結尾時，瀏覽游標會停留在它所在的位置，並以 MQRC_NO_MSG_AVAILABLE 傳回。 然後，您可以將它留在那裡等待進一步訊息，或將它重設為具有 MQGMO_BROWSE_FIRST 呼叫的佇列開頭。

依優先順序排列的佇列

此順序佇列中的第一個訊息是佇列上最長且在發出 MQOPEN 呼叫時具有最高優先順序的訊息。

請使用 MQGMO_BROWSE_NEXT 來讀取佇列中的訊息。

瀏覽游標指向下一個訊息，從第一個訊息的優先順序開始，以最低優先順序的訊息完成。 只要訊息的優先順序等於或低於現行瀏覽游標所識別的訊息，它就會在此期間瀏覽放入佇列的任何訊息。

任何放入較高優先順序佇列的訊息只能透過下列方式瀏覽:

- 重新開啟佇列以進行瀏覽，此時會建立新的瀏覽游標
- 使用 MQGMO_BROWSE_FIRST 選項

未確定的訊息

瀏覽絕不會看到未確定的訊息; 瀏覽游標會跳過它。

在確定工作單元之前，無法瀏覽工作單元內的訊息。 除非您使用 MQGMO_BROWSE_FIRST 選項並再次處理佇列，否則訊息在確定時不會變更其在佇列上的位置，因此即使 已 確定，也不會看到未確定的訊息。

變更佇列順序

當佇列中有訊息時，如果訊息遞送順序從優先順序變更為 FIFO ，則不會變更已排入佇列的訊息順序。 稍後新增至佇列的訊息會採用佇列的預設優先順序。

使用佇列的索引

當您瀏覽只包含單一優先順序 (持續或非持續或兩者) 之訊息的索引佇列時，當使用特定形式的瀏覽時，佇列管理程式會使用索引來瀏覽。

註: 僅在 IBM MQ for z/OS 上受支援。

當檢索佇列只包含單一優先順序的訊息時，會使用下列任何形式的瀏覽:

1. 如果依 MSGID 來檢索佇列，則會使用索引來處理在 MQMD 結構中傳遞 MSGID 的瀏覽要求，以尋找目標訊息。
2. 如果佇列是由 CORRELID 編製索引，則會使用索引來處理在 MQMD 結構中傳遞 CORRELID 的瀏覽要求，以尋找目標訊息。
3. 如果佇列是由 GROUPID 編製索引，則會使用索引來處理在 MQMD 結構中傳遞 GROUPID 的瀏覽要求，以尋找目標訊息。

如果瀏覽要求未在 MQMD 結構中傳遞 MSGID、CORRELID 或 GROUPID ，則會檢索佇列並傳回訊息，必須找到訊息的索引項目，以及其中用來更新瀏覽游標的資訊。 如果您使用廣泛的索引值選項，則不會對瀏覽要求新增大量額外處理。

當訊息長度不明時瀏覽訊息

若要在您不知道訊息大小且不想使用 *MsgId*、 *CorrelId* 或 *GroupId* 欄位來尋找訊息時瀏覽訊息，您可以使用 MQGMO_BROWSE_MSG_UNDER_CURSOR 選項:

1. 發出具有下列項目的 MQGET:
   - MQGMO_BROWSE_FIRST 或 MQGMO_BROWSE_NEXT 選項
   - MQGMO_ACCEPT_TRUNCATED_MSG 選項
   - 緩衝區長度零

   註: 如果另一個程式可能取得相同的訊息，也請考慮使用 MQGMO_LOCK 選項。 應傳回 MQRC_TRUNCATED_MSG_ACCEPTED。
2. 使用傳回的 *DataLength* 來配置所需的儲存體。
3. 使用 MQGMO_BROWSE_MSG_UNDER_CURSOR 發出 MQGET。

所指向的訊息是所擷取的最後一個訊息; 瀏覽游標將不會移動。 您可以選擇使用 MQGMO_LOCK 選項來鎖定訊息，或使用 MQGMO_UNLOCK 選項來解除鎖定已鎖定訊息。

如果在開啟佇列之後未順利發出具有 MQGMO_BROWSE_FIRST 或 MQGMO_BROWSE_NEXT 選項的 MQGET ，則呼叫會失敗。

移除您已瀏覽的訊息

您可以從佇列中移除已瀏覽的訊息，但前提是您已開啟佇列來移除訊息及進行瀏覽。 (您必須在 MQOPEN 呼叫上指定 MQOO_INPUT_ * 選項之一，以及 MQOO_BROWSE 選項。)

若要移除訊息，請再次呼叫 MQGET ，但在 MQGMO 結構的 *Options* 欄位中指定 MQGMO_MSG_UNDER_CURSOR。在此情況下， MQGET 呼叫會忽略 MQMD 結構的 *MsgId*、 *CorrelId* 和 *GroupId* 欄位。

在瀏覽步驟與移除步驟之間的時間內，另一個程式可能已從佇列中移除訊息，包括瀏覽游標下的訊息。 在此情況下， MQGET 呼叫會傳回原因碼，指出訊息無法使用。

以邏輯順序瀏覽訊息

第 648 頁的『邏輯和實體排序』 說明佇列上訊息的邏輯順序與實體順序之間的差異。 當瀏覽佇列時，這項區別特別重要，因為一般而言，不會刪除訊息，且瀏覽作業不一定會從佇列開頭開始。

如果應用程式瀏覽一個群組的各種訊息 (使用邏輯順序) ，則應該遵循邏輯順序來到達下一個群組的開頭，這很重要，因為一個群組的最後一則訊息可能實際出現在下一個群組的第一則訊息 之後 。 MQGMO_LOGICAL_ORDER 選項可確保在掃描佇列時遵循邏輯順序。

請小心使用 MQGMO_ALL_MSGS_AVAILABLE (或 MQGMO_ALL_SEGMENTS_AVAILABLE) ，以進行瀏覽作業。 請考量具有 MQGMO_ALL_MSGS_AVAILABLE 的邏輯訊息。 其效果是只有在群組中的所有其餘訊息也存在時，才能使用邏輯訊息。 如果不是，則會傳遞訊息。 這可能表示當後續抵達遺漏的訊息時，瀏覽下一個作業不會注意到這些訊息。

例如，如果存在下列邏輯訊息，

```
Logical message 1 (not last) of group 123
Logical message 1 (not last) of group 456
Logical message 2 (last)     of group 456
```

並發出具有 MQGMO_ALL_MSGS_AVAILABLE 的瀏覽功能，即會傳回群組 456 的第一個邏輯訊息，並將瀏覽游標留在此邏輯訊息上。 如果群組 123 的第二個 (最後一個) 訊息現在到達:

```
Logical message 1 (not last) of group 123
Logical message 2 (last)     of group 123
Logical message 1 (not last) of group 456  <=== browse cursor
Logical message 2 (last)     of group 456
```

並且會發出相同的瀏覽-下一個功能，它不會注意到群組 123 現在已完成，因為此群組的第一個訊息是在瀏覽游標 之前 。

在某些情況下 (例如，當群組完整呈現時，如果以破壞性方式擷取訊息) ，您可以搭配 MQGMO_BROWSE_FIRST 一起使用 MQGMO_ALL_MSGS_AVAILABLE。 否則，您必須重複瀏覽掃描，以記下已遺漏的新送達訊息; 僅與 MQGMO_BROWSE_NEXT 及 MQGMO_ALL_MSGS_AVAILABLE 一起發出 MQGMO_WAIT 並不會考量它們。 (這也會發生在掃描訊息完成之後可能到達的較高優先順序訊息。)

接下來各節會查看處理未分段訊息的瀏覽範例; 分段訊息遵循類似的原則。

瀏覽群組中的訊息

在此範例中，應用程式會以邏輯順序瀏覽佇列上的每一則訊息。

佇列上的訊息可能會分組。 對於分組訊息，應用程式不想要開始處理任何群組，直到它內的所有訊息都已送達為止。 MQGMO_ALL_MSGS_AVAILABLE 因此指定給群組中的第一個訊息; 對於群組中的後續訊息，此選項是不必要的。

在此範例中使用 MQGMO_WAIT。 不過，雖然在新群組到達時可以滿足等待，但基於 第 673 頁的『以邏輯順序瀏覽訊息』中的原因，如果瀏覽游標已通過群組中的第一個邏輯訊息，且剩餘訊息現在已到達，則無法滿足等待。 不過，等待適當的間隔可確保應用程式在等待新訊息或區段時不會持續迴圈。

整個都會使用 MQGMO_LOGICAL_ORDER ，以確保掃描以邏輯順序進行。 這與破壞性 MQGET 範例相反，因為正在移除每一個群組，所以在尋找群組中的第一個 (或唯一) 訊息時不會使用 MQGMO_LOGICAL_ORDER。

不論訊息是否已分段，都會假設應用程式的緩衝區一律夠大，足以保留整個訊息。 因此會在每一個 MQGET 上指定 MQGMO_COMPLETE_MSG。

以下提供瀏覽群組中邏輯訊息的範例:

```
/* Browse the first message in a group, or a message not in a group */
GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
| MQGMO_ALL_MSGS_AVAILABLE | MQGMO_WAIT
MQGET GMO.MatchOptions = MQMO_MATCH_MSG_SEQ_NUMBER, MD.MsgSeqNumber = 1
/* Examine first or only message */
...

GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
   MQGET
   /* Examine each remaining message in the group */
   ...
```

該群組會一直重複，直到傳回 MQRC_NO_MSG_AVAILABLE 為止。

破壞性瀏覽及擷取
在此範例中，應用程式會先瀏覽群組內的每一個邏輯訊息，然後再決定是否破壞性地擷取該群組。

此範例的第一部分與前一個部分類似。 然而，在此情況下，在瀏覽過整個群組之後，我們決定返回並破壞性地擷取它。

在此範例中移除每一個群組時，在尋找群組中的第一個或唯一訊息時不會使用 MQGMO_LOGICAL_ORDER。

下列提供瀏覽然後破壞性擷取的範例:

```
GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_MESSAGES_AVAILABE | MQGMO_WAIT
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
   MQGET
   /* Examine each remaining message in the group (or as many as
      necessary to decide whether to get it destructively) */
   ...

if ( we want to retrieve the group destructively )

   if ( GroupStatus == ' ' )
      /* We retrieved an ungrouped message */
      GMO.Options = MQGMO_MSG_UNDER_CURSOR | MQGMO_SYNCPOINT
      MQGET GMO.MatchOptions = 0
      /* Process the message */
      ...

   else
      /* We retrieved one or more messages in a group.  The browse cursor */
      /* will not normally be still on the first in the group, so we have */
      /* to match on the GroupId and MsgSeqNumber = 1.                    */
      /* Another way, which works for both grouped and ungrouped messages,*/
      /* would be to remember the MsgId of the first message when it was  */
      /* browsed, and match on that.                                      */
      GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT
      MQGET GMO.MatchOptions = MQMO_MATCH_GROUP_ID
                             | MQMO_MATCH_MSG_SEQ_NUMBER,
           (MQMD.GroupId      = value already in the MD)
            MQMD.MsgSeqNumber = 1
      /* Process first or only message */
      ...

      GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT
                  | MQGMO_LOGICAL_ORDER
      do while ( GroupStatus == MQGS_MSG_IN_GROUP )
         MQGET
         /* Process each remaining message in the group */
...
```

避免重複遞送瀏覽的訊息
透過使用某些開啟選項及取得訊息選項，您可以將訊息標示為已瀏覽，以便現行或其他協同作業應用程式不再擷取它們。 訊息可以明確或自動取消標示，讓它們再次可供瀏覽。

如果您瀏覽佇列上的訊息，則可能會以不同於您破壞性取得訊息時擷取訊息的順序來擷取訊息。 特別是，您可以多次瀏覽相同的訊息，如果從佇列中移除，則無法瀏覽相同的訊息。 若要避免此情況，您可以在瀏覽訊息時 標示 訊息，並避免擷取已標示的訊息。 這有時稱為 以標記瀏覽。 若要標示已瀏覽的訊息，請使用取得訊息選項 MQGMO_MARK_BROWSE_HANDLE ，並僅擷取未標示的訊息，請使用 MQGMO_UNMARKED_BROWSE_MSG。 如果您使用選項 MQGMO_BROWSE_FIRST、MQGMO_UNMARKED_BROWSE_MSG 及 MQGMO_MARK_BROWSE_HANDLE 的組合，並發出重複的 MQGET ，則會依序擷取佇列上的每一則訊息。 這可防止重複遞送訊息，即使使用 MQGMO_BROWSE_FIRST 來確保不會跳過訊息。 此選項組合可以由單一常數 MQGMO_BROWSE_HANDLE 來表示。 當佇列上沒有未瀏覽的訊息時，會傳回 MQRC_NO_MSG_AVAILABLE。

如果多個應用程式瀏覽相同的佇列，則可以使用選項 MQOO_CO_OP 及 MQOO_BROWSE 開啟佇列。 每一個 MQOPEN 所傳回的物件控點都視為協同群組的一部分。 指定選項 MQGMO_MARK_BROWSE_CO_OP 的 MQGET 呼叫所傳回的任何訊息都被視為針對這組協同作業控點進行標示。

如果訊息已標示一段時間，佇列管理程式會自動取消標示它，並讓它重新可供瀏覽。 佇列管理程式屬性 MsgMarkBrowseInterval 提供針對協同作業的控點保持標示訊息的時間 (以毫秒為單位)。 MsgMarkBrowseInterval 為 -1 表示永不自動取消標示訊息。

當單一處理程序或一組共同處理程序標示訊息停止時，任何標示的訊息都會變成未標示。

合作瀏覽範例

您可以執行分派器應用程式的多個副本，以瀏覽佇列上的訊息，並根據每一個訊息的內容來起始消費者。 在每一個分派器中，開啟具有 MQOO_CO_OP 的佇列。 這表示分派器正在合作，並將知道彼此的標示訊息。 然後，每一個分派器都會發出重複的 MQGET 呼叫，並指定選項 MQGMO_BROWSE_FIRST、MQGMO_UNMARKED_BROWSE_MSG 及 MQGMO_MARK_BROWSE_CO_OP (您可以使用單一常數 MQGMO_BROWSE_CO_OP 來代表此選項組合)。 然後，每一個分派器應用程式只會擷取尚未由其他協同作業分派器標示的那些訊息。 分派器會起始設定消費者，並將 MQGET 傳回的 MsgToken 傳遞給消費者，這會破壞性地從佇列中取得訊息。 如果消費者取消訊息的 MQGET ，則該訊息可供其中一個瀏覽器重新分派，因為它不再標示。 如果消費者未對訊息執行 MQGET ，則在通過 MsgMarkBrowseInterval 之後，佇列管理程式會針對協同作業的控點集取消標示該訊息，並且可以重新分派該訊息。

相對於相同分派器應用程式的多個副本，您可能有許多不同的分派器應用程式瀏覽佇列，每一個都適合處理佇列上的訊息子集。 在每一個分派器中，開啟具有 MQOO_CO_OP 的佇列。 這表示分派器正在合作，並將知道彼此的標示訊息。

- 如果單一分派器的訊息處理順序很重要，則每一個分派器都會進行重複的 MQGET 呼叫，並指定選項 MQGMO_BROWSE_FIRST、MQGMO_UNMARKED_BROWSE_MSG 及 MQGMO_MARK_BROWSE_HANDLE (或 MQGMO_BROWSE_HANDLE)。 如果瀏覽過的訊息適合此分派器處理，則會發出 MQGET 呼叫，並指定 MQMO_MATCH_MSG_TOKEN、MQGMO_MARK_BROWSE_CO_OP ，以及前一個 MQGET 呼叫所傳回的 MsgToken 。 如果呼叫成功，則分派器會起始設定消費者，並將 MsgToken 傳遞給消費者。
- 如果訊息處理順序不重要，且分派器預期會處理它所遇到的大部分訊息，請使用選項 MQGMO_BROWSE_FIRST、MQGMO_UNMARKED_BROWSE_MSG 及 MQGMO_MARK_BROWSE_CO_OP (或 MQGMO_BROWSE_CO_OP)。 如果分派器瀏覽無法處理的訊息，則會取消標示訊息，方法是呼叫 MQGET 並先前傳回選項 MQGMO_MATCH_MSG_TOKEN、MQGMO_UNMARK_BROWSE_CO_OP 及 MsgToken 。

MQGET 呼叫失敗的部分情況

如果在發出 MQOPEN 和 MQGET 呼叫之間的指令上使用 FORCE 選項變更佇列的某些屬性，則 MQGET 呼叫會失敗，並傳回 MQRC_OBJECT_CHANGED 原因碼。

佇列管理程式會將物件控點標示為不再有效。 如果變更套用至佇列名稱所解析的任何佇列，也會發生這種情況。 以這種方式影響控點的屬性會列在 MQOPEN 中 MQOPEN 呼叫的說明中。 如果您的呼叫傳回 MQRC_OBJECT_CHANGED 原因碼，請關閉佇列，重新開啟它，然後重試取得訊息。

如果您嘗試從中取得訊息的佇列 (或佇列名稱解析的任何佇列) 禁止取得作業，則 MQGET 呼叫會失敗並傳回 MQRC_GET_INHIBITED 原因碼。 即使您使用 MQGET 呼叫來瀏覽，也會發生這種情況。 如果您稍後嘗試 MQGET 呼叫，且應用程式的設計使得其他程式定期變更佇列的屬性，則您可能能夠順利取得訊息。

如果已刪除動態佇列 (暫時或永久)，則使用先前獲得的物件控點的 MQGET 呼叫會失敗，並傳回 MQRC_Q_DELETED 原因碼。

撰寫發佈/訂閱應用程式

開始撰寫發佈/訂閱 IBM MQ 應用程式。

如需發佈/訂閱概念的概觀，請參閱 發佈/訂閱傳訊。

如需撰寫不同類型發佈/訂閱應用程式的相關資訊，請參閱下列主題:

- 第 676 頁的『撰寫發佈者應用程式』
- 第 683 頁的『撰寫訂閱者應用程式』
- 第 699 頁的『發佈/訂閱生命週期』
- 第 702 頁的『發佈/訂閱訊息內容』
- 第 704 頁的『訊息排序』
- 第 704 頁的『截取發佈』
- 第 712 頁的『發佈選項』
- 第 712 頁的『訂閱選項』

相關概念

第 6 頁的『應用程式開發概念』
您可以使用選擇的程序化或物件導向語言來撰寫 IBM MQ 應用程式。 在開始設計及撰寫 IBM MQ 應用程式之前，請先熟悉基本 IBM MQ 概念。

第 5 頁的『開發適用於 IBM MQ 的應用程式』
您可以開發應用程式來傳送及接收訊息，以及管理佇列管理程式和相關資源。 IBM MQ 支援以許多不同語言及架構撰寫的應用程式。

第 41 頁的『IBM MQ 應用程式的設計考量』
當您決定應用程式如何利用可供您使用的平台及環境時，您需要決定如何使用 IBM MQ 所提供的特性。

第 603 頁的『撰寫佇列作業的程序化應用程式』
請利用這項資訊來瞭解如何撰寫佇列作業應用程式、連接佇列管理程式、發佈/訂閱，以及開啟和關閉物件。

第 765 頁的『撰寫用戶端程序化應用程式』
使用程序化語言在 IBM MQ 上撰寫用戶端應用程式所需的資訊。

第 837 頁的『建置程序化應用程式』
您可以使用數種程序化語言之一來撰寫 IBM MQ 應用程式，並在數個不同的平台上執行該應用程式。

第 871 頁的『處理程序化程式錯誤』
此資訊說明與應用程式 MQI 呼叫相關聯的錯誤，當它進行呼叫時，或當它的訊息遞送至其最終目的地時。

相關工作

第 887 頁的『使用 IBM MQ 範例程序化程式』
這些範例程式是以程序化語言撰寫，並示範「訊息佇列介面 (MQI)」的一般用法。 不同平台上的 IBM MQ 程式。

撰寫發佈者應用程式

透過研究兩個範例，開始撰寫發佈者應用程式。 第一個是盡可能在點對點應用程式將訊息放置在佇列上的模型，第二個示範動態建立主題-發佈者應用程式更常見的型樣。

撰寫簡式 IBM MQ 發佈者應用程式就像撰寫 IBM MQ 點對點應用程式，將訊息放入佇列 (第 676 頁的表 121)。 差異是您對主題而非佇列的 MQPUT 訊息。

表 *121:* 點對點與發佈/訂閱 *IBM MQ* 程式型樣。

步驟	**點對點 MQ 呼叫**	**發佈 MQ 呼叫**
連接至佇列管理程式	MQCONN	MQCONN
開啟佇列	MQOPEN	

表 *121:* 點對點與發佈/訂閱 *IBM MQ* 程式型樣。 *(*繼續*)*		
步驟	**點對點 MQ 呼叫**	**發佈 MQ 呼叫**
開啟主題		MQOPEN
放置訊息	MQPUT	MQPUT
關閉主題		MQCLOSE
關閉佇列	MQCLOSE	
切斷與佇列管理程式的連線	MQDISC	MQDISC

為了具體說明這一點，有兩個例子可以說明公佈股票價格的申請。 在第一個範例 (第 677 頁的『範例 1: 發佈者至固定主題』) 中，管理者會以類似建立佇列的方式來建立主題定義。 程式設計師會撰寫 MQPUT 來將訊息寫入主題，而不是將訊息寫入佇列。 在第二個範例 (第 680 頁的『範例 2: 發佈至變數主題』) 中，程式與 IBM MQ 的互動型樣類似。 不同之處在於程式設計師會提供訊息寫入的主題，而不是管理者。 實際上，這通常表示主題字串是已定義的內容，或由另一個來源提供，例如透過瀏覽器的人工輸入。

相關概念

第 683 頁的『撰寫訂閱者應用程式』
透過研究三個範例來開始撰寫訂閱者應用程式: 從佇列耗用訊息的 IBM MQ 應用程式、建立訂閱且不需要佇列作業知識的應用程式，以及最後使用佇列作業及訂閱的範例。

相關參考

DEFINE TOPIC
DISPLAYTOPIC
DISPLAYTPSTATUS

範例 *1:* 發佈者至固定主題
IBM MQ 程式，用於說明發佈至管理定義的主題。

註: 精簡編碼樣式是為了可讀性而非正式作業用途。

請參閱 第 678 頁的圖 65 中的輸出

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    char     topicNameDefault[]   = "IBMSTOCKPRICE";
    char     publicationDefault[] = "129";
    MQCHAR48 qmName = "";

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle                    */
    MQHOBJ   Hobj = MQHO_NONE;            /* object handle sub queue              */
    MQLONG   CompCode = MQCC_OK;          /* completion code                      */
    MQLONG   Reason = MQRC_NONE;          /* reason code                          */
    MQOD     td = {MQOD_DEFAULT};         /* Object descriptor                    */
    MQMD     md = {MQMD_DEFAULT};         /* Message Descriptor                   */
    MQPMO   pmo = {MQPMO_DEFAULT};        /* put message options                  */
    MQCHAR   resTopicStr[151];            /* Returned vale of topic string        */
    char *   topicName   = topicNameDefault;
    char *   publication = publicationDefault;
    memset   (resTopicStr, 0 , sizeof(resTopicStr));

    switch(argc){                         /* replace defaults with args if provided */
        default:
            publication = argv[2];
        case(2):
            topicName = argv[1];
        case(1):
            printf("Optional parameters:  TopicObject Publication\n");
    }
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        td.ObjectType = MQOT_TOPIC;        /* Object is a topic                    */
        td.Version = MQOD_VERSION_4;       /* Descriptor needs to be V4            */
        strncpy(td.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
        td.ResObjectString.VSPtr = resTopicStr;
        td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
        MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
        MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode,
&Reason);
        if (CompCode != MQCC_OK) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    if (CompCode == MQCC_OK)
        printf("Published \"%s\" using topic \"%s\" to topic string \"%s\"\n",
                publication, td.ObjectName, resTopicStr);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

圖 *64:* 簡式 *IBM MQ* 發佈者至固定主題。

```
X:\Publish1\Debug>PublishStock
Optional parameters:  TopicObject Publication
Published "129" using topic "IBMSTOCKPRICE" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

X:\Publish1\Debug>PublishStock IBMSTOCKPRICE 155
Optional parameters:  TopicObject Publication
Published "155" using topic "IBMSTOCKPRICE" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

圖 *65:* 第一個發佈者範例的輸出範例

下列選取的程式碼行說明撰寫 IBM MQ 的發佈者應用程式的層面。

char topicNameDefault[] = "IBMSTOCKPRICE";
預設主題名稱定義在程式中。 您可以提供不同主題物件的名稱作為程式的第一個引數來置換它。

MQCHAR resTopicStr[151];
resTopicStr 由 td.ResObjectString.VSPtr 指向，並由 MQOPEN 用來傳回已解析的主題字串。 使 resTopicStr 的長度大於 td.ResObjectString.VSBufSize 中所傳遞的長度，以提供空值終止的空間。

memset (resTopicStr, 0, sizeof(resTopicStr));
將 resTopicStr 起始設定為空值，以確保 MQCHARV 中傳回的已解析主題字串以空值結尾。

td.ObjectType = MQOT_TOPIC
有一種新的物件類型可供發佈/訂閱: *topic object*。

td.Version = MQOD_VERSION_4;
若要使用物件的新類型，您必須至少使用物件描述子 第 *4* 版 。

strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);
topicName 是主題物件的名稱，有時稱為管理主題物件。 在此範例中，需要事先使用「IBM MQ 探險家」或此 MQSC 指令來建立主題物件，

```
DEFINE TOPIC(IBMSTOCKPRICE) TOPICSTR(NYSE/IBM/PRICE) REPLACE;
```

td.ResObjectString.VSPtr = resTopicStr;
已解析的主題字串會在程式的最終 printf 中回應。 設定 IBM MQ 的 MQCHARV ResObjectString 結構，以將已解析的字串傳回給程式。

MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
開啟輸出主題; 就像開啟輸出佇列一樣。

pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
您希望新訂閱者能夠接收發佈，並且透過在發佈者中指定 MQPMO_RETAIN ，當您啟動訂閱者時，它會接收在訂閱者啟動之前發佈的最新發佈，作為其第一個相符發佈。 替代方案是向訂閱者提供只有在訂閱者啟動之後才發佈的發佈。 此外，訂閱者也可以選擇在其訂閱中指定 MQSO_NEW_PUBLICATIONS_ONLY ，以拒絕接收保留的發佈。

MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
將 1 新增至傳遞至 MQPUT 的字串長度，以將空值終止字元傳遞至 IBM MQ 作為訊息緩衝區的一部分。

第一個範例說明了什麼? 此範例儘量模仿已嘗試並測試過的傳統型樣，以撰寫點對點 IBM MQ 程式。 IBM MQ 程式設計型樣的一個重要特性是程式設計師不關心傳送訊息的位置。 程式設計師的作業是連接至佇列管理程式，並將要配送給收件者的訊息傳遞給它。 在點對點參照範例中，程式設計師會開啟管理者已配置的佇列 (可能是別名佇列)。 別名佇列會將訊息遞送至本端佇列管理程式上的目標佇列，或遞送至遠端佇列管理程式。 當訊息等待遞送時，它們會儲存在來源與目的地之間的佇列中。

在發佈/訂閱型樣中，程式設計師會開啟主題，而不是開啟佇列。 在我們的範例中，主題由管理者與主題字串相關聯。 佇列管理程式會使用佇列，將發佈轉遞至訂閱符合發佈主題字串的本端或遠端訂閱者。 如果保留發佈，佇列管理程式會保留發佈的最新副本，即使它現在沒有訂閱者也一樣。 保留的發佈可供轉遞給未來的訂閱者。 發佈者應用程式在選取發佈資訊或將發佈資訊遞送至目的地時，沒有任何作用; 其作業是建立發佈資訊，並將發佈資訊放置到管理者所定義的主題中。

這個固定主題範例是許多發佈/訂閱應用程式的非典型範例: 它是靜態的。 它需要管理者定義主題字串，並變更在上發佈的主題。 通常發佈/訂閱應用程式需要知道部分或所有主題樹狀結構。 主題可能經常變更，或者雖然主題不會變更太多，但主題組合的數目很大，管理者為可能需要在上發佈的每一個主題字串定義主題節點過於繁重。 可能在發佈之前並不知道主題字串; 發佈者應用程式可能使用發佈內容中的資訊來指定主題字串，或者它可能具有要從另一個來源 (例如來自瀏覽器的人工輸入) 發佈之主題字串的相關資訊。 為了迎合更動態的發佈樣式，下一個範例顯示如何動態建立主題，作為發佈者應用程式的一部分。

主題將發佈者和訂閱者連結在一起。 設計規則或架構以命名主題，並在主題樹狀結構中組織它們，是開發發佈/訂閱解決方案的重要步驟。 請仔細查看主題樹狀結構的組織將發佈者和訂閱者程式連結在一起的程度，並將它們連結至主題樹狀結構的內容。 問自己主題樹狀結構中的變更是否會影響發佈者和訂閱者應用程式，以及如何將影響降到最低。 建置在 IBM MQ 發佈/訂閱模型架構中的是管理主題物件的概念，提供主題的根部分或根子樹狀結構。 主題物件可讓您選擇以管理方式定義主題樹狀結構的根部分，以簡化應用程式設計和

作業，進而增進可維護性。 例如，如果您要部署多個具有隔離主題樹狀結構的發佈/訂閱應用程式，則透過管理方式定義主題樹狀結構的根部分，您可以保證主題樹狀結構的隔離，即使不同應用程式所採用的主題命名慣例沒有一致性也一樣。

實際上，發佈者應用程式僅使用固定主題 (如此範例) 和變數主題 (如下一個範例) 來涵蓋光譜。 第 680 頁的『範例 2: 發佈至變數主題』 也示範如何結合主題和主題字串。

相關概念

第 680 頁的『範例 2: 發佈至變數主題』
WebSphere MQ 程式，說明如何發佈至以程式化方式定義的主題。

第 683 頁的『撰寫訂閱者應用程式』
透過研究三個範例來開始撰寫訂閱者應用程式: 從佇列耗用訊息的 IBM MQ 應用程式、建立訂閱且不需要佇列作業知識的應用程式，以及最後使用佇列作業及訂閱的範例。

範例 *2:* 發佈至變數主題
WebSphere MQ 程式，說明如何發佈至以程式化方式定義的主題。

註: 精簡編碼樣式是為了可讀性而非正式作業用途。

請參閱 第 681 頁的圖 67 中的輸出。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    char     topicNameDefault[]   = "STOCKS";
    char     topicStringDefault[] = "IBM/PRICE";
    char     publicationDefault[] = "130";
    MQCHAR48 qmName = "";

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle           */
    MQHOBJ   Hobj     = MQHO_NONE;   /* object handle sub queue          */
    MQLONG   CompCode = MQCC_OK;     /* completion code                  */
    MQLONG   Reason   = MQRC_NONE;   /* reason code                      */
    MQOD     td = {MQOD_DEFAULT};    /* Object descriptor                */
    MQMD     md = {MQMD_DEFAULT};    /* Message Descriptor               */
    MQPMO   pmo = {MQPMO_DEFAULT};   /* put message options              */
    MQCHAR   resTopicStr[151];       /* Returned value of topic string */
    char *   topicName   = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   publication = publicationDefault;
    memset   (resTopicStr, 0 , sizeof(resTopicStr));

    switch(argc){            /* Replace defaults with args if provided */
        default:
            publication = argv[3];
        case(3):
            topicString = argv[2];
        case(2):
            if (strcmp(argv[1],"/"))  /* "/" invalid = No topic object  */
                topicName = argv[1];
            else
                *topicName = '\0';
        case(1):
            printf("Provide parameters:  TopicObject TopicString Publication\n");
    }

    printf("Publish \"%s\" to topic \"%-.48s\" and topic string \"%s\"\n", publication, topicName,
topicString);
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        td.ObjectType = MQOT_TOPIC;    /* Object is a topic             */
        td.Version = MQOD_VERSION_4;   /* Descriptor needs to be V4     */
        strncpy(td.ObjectName, topicName,  MQ_TOPIC_NAME_LENGTH);
        td.ObjectString.VSPtr = topicString;
        td.ObjectString.VSLength = (MQLONG)strlen(topicString);
        td.ResObjectString.VSPtr = resTopicStr;
        td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
        MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
        MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    if (CompCode == MQCC_OK)
        printf("Published \"%s\" to topic string \"%s\"\n",  publication, resTopicStr);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

圖 *66:* 變數主題的簡式 *IBM MQ* 發佈者。

```
X:\Publish2\Debug>PublishStock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

X:\Publish2\Debug>PublishStock / NYSE/IBM/PRICE 131
Provide parameters: TopicObject TopicString Publication
Publish "131" to topic "" and topic string "NYSE/IBM/PRICE"
Published "131" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

圖 *67:* 第二個發佈者範例的輸出範例

這個範例有幾個要點要注意。

char topicNameDefault[] = "STOCKS";
預設主題名稱 STOCKS 定義主題字串的一部分。 您可以提供此主題名稱作為程式的第一個引數來置換此主題名稱，或提供 / 作為第一個參數來避免使用主題名稱。

char topicString[101] = "IBM/PRICE";
IBM/PRICE 是預設主題字串。 您可以提供這個主題字串作為程式的第二個引數，來置換這個主題字串。

佇列管理程式會結合 STOCKS 主題物件 "NYSE"所提供的主題字串與程式 "IBM/PRICE" 所提供的主題字串，並在兩個主題字串之間插入 "/" 。 結果是已解析的主題字串 "NYSE/IBM/PRICE"。 產生的主題字串與 IBMSTOCKPRICE 主題物件中定義的主題字串相同，且具有完全相同的效果。

與已解析的主題字串相關聯的管理主題物件，不一定與發佈者傳遞至 MQOPEN 的主題物件相同。 IBM MQ 使用已解析主題字串中隱含的樹狀結構，來找出哪個管理主題物件定義與發佈資訊相關聯的屬性。

假設有兩個主題物件 A 和 B， A 定義主題 "a"，而 B 定義主題 "a/b" (第 682 頁的圖 68)。 如果發佈者程式參照主題物件 A 並提供主題字串 "b"(將主題解析為主題字串 "a/b")，則發佈會從主題物件 B 繼承其內容，因為主題符合針對 B 定義的主題字串 "a/b" 。

if (strcmp(argv[1],"/"))
argv[1] 是選擇性地提供的 topicName。 "/" 作為主題名稱無效; 在這裡，它表示沒有主題名稱，且主題字串完全由程式提供。 第 681 頁的圖 67 中的輸出會顯示程式動態提供的整個主題字串。

strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);
對於預設案例，需要使用「IBM MQ 探險家」或此 MQSC 指令預先建立選用 topicName :

```
DEFINE TOPIC(STOCKS) TOPICSTR(NYSE) REPLACE;
```

td.ObjectString.VSPtr = topicString;
主題字串是主題描述子中的 MQCHARV 欄位

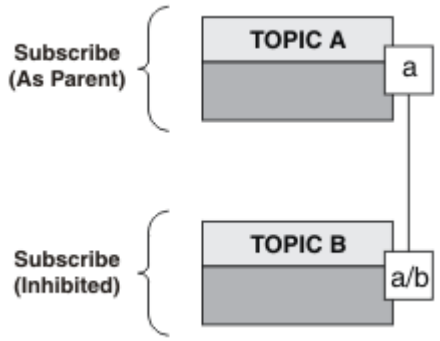


圖 *68:* 主題物件關聯

第二個範例說明了什麼? 雖然程式碼與第一個範例非常相似 (實際上只有兩行差異)，但結果是與第一個範例有明顯不同的程式。 程式設計師控制將發佈傳送至的目的地。 與用來設計訂閱者應用程式的最小管理者輸入一起使用時，不需要預先定義任何主題或佇列，即可將發佈從發佈者遞送至訂閱者。

在點對點傳訊參照範例中，必須先定義佇列，訊息才能流動。 對於發佈/訂閱，雖然 IBM MQ 會使用其基礎佇列系統來實作發佈/訂閱，但它們不會; 發佈/訂閱應用程式會繼承保證遞送、交易方式及與傳訊及佇列作業相關聯之鬆散連結的好處。

設計程式必須決定發佈者和訂閱者是否要知道基礎主題樹狀結構，以及訂閱者程式是否知道佇列作業。 接下來請研究訂閱者範例應用程式。 它們設計為與發佈者範例搭配使用，通常是發佈及訂閱 NYSE/IBM/PRICE。

相關概念
第 677 頁的『範例 1: 發佈者至固定主題』
IBM MQ 程式，用於說明發佈至管理定義的主題。

第 683 頁的『撰寫訂閱者應用程式』
透過研究三個範例來開始撰寫訂閱者應用程式: 從佇列耗用訊息的 IBM MQ 應用程式、建立訂閱且不需要佇列作業知識的應用程式，以及最後使用佇列作業及訂閱的範例。

撰寫訂閱者應用程式

透過研究三個範例來開始撰寫訂閱者應用程式: 從佇列耗用訊息的 IBM MQ 應用程式、建立訂閱且不需要佇列作業知識的應用程式，以及最後使用佇列作業及訂閱的範例。

在 第 683 頁的表 122 中，會列出三種樣式的消費者或訂閱者，以及用來描述它們的 IBM MQ 函數呼叫序列。

1. 第一種樣式 (MQ Publication Consumer) 與僅執行 MQGET 的點對點 MQ 程式相同。 應用程式不知道它正在耗用發佈-它只是從佇列中讀取訊息。 使用「IBM MQ 探險家」或指令，以管理方式建立可將發佈資訊遞送至佇列的訂閱。
2. 第二種樣式是大部分訂閱者應用程式的偏好型樣。 訂閱者應用程式會建立訂閱，然後取得發佈。 佇列管理全部由佇列管理程式執行。 這稱為 受管理訂閱者。
3. 在第三種樣式中，訂閱者應用程式負責指定將用來保留發佈資訊的佇列，開啟及關閉該佇列，並發出訂閱以將發佈資訊填入佇列。 這稱為 未受管理的訂閱者。

瞭解這些樣式的一種方法是針對每一種樣式研究 第 683 頁的表 122 中列出的範例 C 程式。 範例設計為與在 第 676 頁的『撰寫發佈者應用程式』中找到的發佈者範例一起執行。

表 *122:* 點對點與訂閱 *IBM MQ* 程式型樣。

步驟	MQ 訊息消費者	第 683 頁的『範例 1: MQ Publication 消費者』	第 686 頁的『範例 2: 受管理 MQ 訂閱者』	第 691 頁的『範例 3: 未受管理的 MQ 訂閱者』
連接至佇列管理程式	MQCONN	MQCONN	MQCONN	MQCONN
開啟佇列	MQOPEN	MQOPEN		MQOPEN
訂閱			MQSUB	MQSUB
取得訊息	MQGET	MQGET	MQGET	MQGET
關閉佇列	MQCLOSE	MQCLOSE	(MQCLOSE)	MQCLOSE
關閉訂閱			MQCLOSE	MQCLOSE
切斷與佇列管理程式的連線	MQDISC	MQDISC	MQDISC	MQDISC

使用 MQCLOSE 一律是選用項目，可以釋放資源、傳遞 MQCLOSE 選項，或只是為了與 MQOPEN 對稱。 由於在受管理 MQ 訂閱者案例中關閉訂閱佇列，且對稱引數不相關時，您不可能需要指定 MQCLOSE 選項，因此在 範例 2: 受管理 MQ 訂閱者中未明確關閉訂閱佇列。

另一種瞭解發佈/訂閱應用程式型樣的方法是，也要查看所涉及的不同實體之間的互動。 生命線 (或 UML 序列圖) 是研究互動的好方法。 第 699 頁的『發佈/訂閱生命週期』中說明三個生命線範例。

範例 *1: MQ Publication* 消費者

MQ Publication 消費者是不訂閱主題本身的 IBM MQ 訊息消費者。

若要建立此範例的訂閱及發佈佇列，請執行下列指令，或使用「IBM MQ 探險家」定義物件。

```
DEFINE QLOCAL(STOCKTICKER) REPLACE;
DEFINE SUB(IBMSTOCKPRICESUB) DEST(STOCKTICKER) TOPICOBJ(IBMSTOCKPRICE) REPLACE;
```

IBMSTOCKPRICESUB 訂閱會參照針對發佈者範例所建立的 IBMSTOCK 主題物件，以及本端佇列 STOCKTICKER。 主題物件 IBMSTOCK 定義在訂閱 NYSE/IBM/PRICE 中使用的主題字串。 請注意，在建立訂閱之前，必須先定義主題物件及用來接收發佈資訊的佇列。

MQ 發佈消費者型樣有許多有價值的資料類型:

1. 多重處理: 共用讀取出版品的工作。 發佈會全部移至與訂閱主題相關聯的單一佇列。 多個消費者可以使用 MQOO_INPUT_SHARED 來開啟佇列。
2. 集中管理的訂閱。 應用程式不會建構自己的訂閱主題或訂閱; 管理者負責傳送發佈的位置。
3. 訂閱集中: 多個不同的訂閱可以傳送至單一佇列。
4. 訂閱延續性: 不論消費者是否在作用中，佇列都會接收所有發佈。
5. 移轉和共存性: 消費者程式碼同樣適用於點對點和發佈/訂閱實務。

訂閱會在主題字串 NYSE/IBM/PRICE 與佇列 STOCKTICKER 之間建立關係。 從建立訂閱時開始，發佈 (包括任何目前保留的發佈) 會轉遞至 STOCKTICKER 。

以管理方式建立的訂閱可以受管理或未受管理。 受管理訂閱會在建立之後立即生效，就像未受管理的訂閱一樣。 並非所有型樣資料類型都可供受管理訂閱使用。 請參閱第 691 頁的『範例 3: 未受管理的 MQ 訂閱者』

註: 精簡編碼樣式是為了可讀性而非正式作業用途。

結果顯示在 第 685 頁的圖 70 中。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
MQCHAR     publicationBuffer[101];
MQCHAR48 subscriptionQueueDefault = "STOCKTICKER";
MQCHAR48 qmName = "";            /* Use default queue manager  */

MQHCONN Hconn = MQHC_UNUSABLE_HCONN;    /* connection handle      */
MQHOBJ  Hobj = MQHO_NONE;         /* object handle sub queue   */
MQLONG  CompCode = MQCC_OK;        /* completion code       */
MQLONG  Reason = MQRC_NONE;        /* reason code         */
MQLONG  messlen = 0;
MQOD   od = {MQOD_DEFAULT};        /* Unmanaged subscription queue */
MQMD   md = {MQMD_DEFAULT};        /* Message Descriptor      */
MQGMO  gmo = {MQGMO_DEFAULT};       /* Get message options     */
char *  publication=publicationBuffer;
char *  subscriptionQueue = subscriptionQueueDefault;

switch(argc){           /* Replace defaults with args if provided */
default:
subscriptionQueue = argv[1]
case(1):
printf("Optional parameter: subscriptionQueue\n");
}

do {
MQCONN(qmName, &Hconn, &CompCode, &Reason);
if (CompCode != MQCC_OK) break;
strncpy(od.ObjectName, subscriptionQueue, MQ_Q_NAME_LENGTH);
MQOPEN(Hconn, &od, MQOO_INPUT_AS_Q_DEF | MQOO_FAIL_IF_QUIESCING , &Hobj, &CompCode, &Reason);
if (CompCode != MQCC_OK) break;
gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
gmo.WaitInterval = 10000;
printf("Waiting %d seconds for publications from %s\n", gmo.WaitInterval/1000,
subscriptionQueue);
do {
memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
md.Encoding    = MQENC_NATIVE;
md.CodedCharSetId = MQCCSI_Q_MGR;
memset(publication, 0, sizeof(publicationBuffer));
MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1, publication, &messlen,
&CompCode, &Reason);
if (Reason == MQRC_NONE)
printf("Received publication \"%s\"\n", publication);
}
while (CompCode == MQCC_OK);
if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
if (CompCode != MQCC_OK) break;
MQDISC(&Hconn, &CompCode, &Reason);
} while (0);
printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

圖 *69: MQ* 發佈消費者。

```
X:\Subscribe1\Debug>Subscribe1
Optional parameter: subscriptionQueue
Waiting 10 seconds for publications from STOCKTICKER
Received publication "129"
Completion code 0 and Return code 0
```

圖 *70: MQ* 發佈消費者的輸出

有幾個標準 IBM MQ *C* 語言程式設計提示需要注意:

memset(publication, 0, sizeof(publicationBuffer));
確保訊息具有尾端空值，以使用 printf 輕鬆格式化。 發佈者範例透過將 1 新增至 strlen(publication)，在傳遞至 MQPUT 的訊息緩衝區中包括尾端空值。 對於使用緩衝區來儲存字串的 IBM MQ C 程式，將 MQCHAR 緩衝區設為空值是良好的程式設計樣式，確保空值在未完全填滿緩衝區的字元陣列之後。

MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1, publication, &messlen, &CompCode, &Reason);
在訊息緩衝區結尾保留一個空值，以確保傳回的訊息在 if (messlen == strlen(publication)); 為 true 時具有尾端空值。 此提示補充先前的提示，並確保 publicationBuffer 中至少有一個空值未被 publication 內容改寫。

相關概念

第 686 頁的『範例 2: 受管理 MQ 訂閱者』
受管理 MQ 訂閱者是大部分訂閱者應用程式的偏好型樣。 受管理訂閱是 IBM MQ 處理訂閱並為您執行登錄及取消登錄的訂閱。 此範例 不 需要佇列、主題或訂閱的管理定義。

第 691 頁的『範例 3: 未受管理的 MQ 訂閱者』
未受管理的訂閱者是訂閱者應用程式的重要類別。 使用它，您可以將發佈/訂閱的好處與佇列作業及使用發佈的 控制 相結合。 未受管理的訂閱是應用程式負責的位置。 用於指定儲存訂閱的佇列。 此範例示範結合訂閱與佇列的不同方式。

第 676 頁的『撰寫發佈者應用程式』
透過研究兩個範例，開始撰寫發佈者應用程式。 第一個是盡可能在點對點應用程式將訊息放置在佇列上的模型，第二個示範動態建立主題-發佈者應用程式更常見的型樣。

範例 *2:* 受管理 *MQ* 訂閱者

受管理 MQ 訂閱者是大部分訂閱者應用程式的偏好型樣。 受管理訂閱是 IBM MQ 處理訂閱並為您執行登錄及取消登錄的訂閱。 此範例 不 需要佇列、主題或訂閱的管理定義。

這種最簡單的受管理訂閱者通常會使用 不可延續 訂閱。 此範例著重於不可延續訂閱。 訂閱只會持續到 MQSUB 中訂閱控點的生命期限為止。 在訂閱的生命期限內，任何符合主題字串的發佈都會傳送至訂閱佇列 (如果未設定或預設旗標 MQSO_NEW_PUBLICATIONS_ONLY，則可能是保留的發佈，會保留先前符合主題字串的發佈，且發佈會持續或佇列管理程式尚未終止，因為發佈已建立)。

您也可以使用具有此型樣的 可延續 訂閱。 通常，如果使用受管理可延續訂閱，則會基於可靠性原因而執行，而不是建立訂閱，因為在沒有發生任何錯誤的情況下，會使訂閱者存活。 如需與受管理、未受管理、可延續及不可延續訂閱相關聯之不同生命週期的相關資訊，請參閱相關主題小節。

可延續訂閱通常與持續發佈相關聯，而不可延續訂閱則與非持續發佈相關聯，但訂閱延續性與發佈持續性之間沒有必要的關係。 持續性和延續性的所有四種組合都是可能的。

對於考量的受管理不可延續案例，佇列管理程式會建立在佇列關閉時清除並刪除的訂閱佇列。 當關閉不可延續訂閱時，會從佇列中移除發佈。

此程式碼所舉例說明的受管理不可延續型樣的寶貴資料類型如下:

1. 隨需應變訂閱: 訂閱主題字串是動態的。 它在執行時由應用程式提供。
2. 自行管理佇列: 訂閱佇列是自行定義及管理。
3. 自我管理訂閱生命週期: 不可延續 訂閱僅在訂閱者應用程式期間存在。
   - 如果您定義 可延續 受管理訂閱，則會導致永久訂閱佇列及發佈繼續儲存在其中，且沒有作用中的訂閱者程式。 只有在應用程式或管理者選擇刪除訂閱之後，佇列管理程式才會刪除佇列 (並從中清除任何未擷取的發佈)。 可以使用管理指令或使用 MQCO_REMOVE_SUB 選項關閉訂閱來刪除訂閱。
   - 請考量為可延續訂閱設定 SubExpiry，以便停止將發佈資訊傳送至佇列，且訂閱者可以在移除訂閱之前耗用任何剩餘的發佈資訊，並導致佇列管理程式刪除佇列及其中任何剩餘的發佈資訊。
4. 彈性主題字串部署: 使用管理定義的主題來定義訂閱的根部分，以簡化訂閱主題管理。 然後會從應用程式隱藏主題樹狀結構的根部分。 透過隱藏根部分，可以部署應用程式，而無需應用程式意外建立與另一個實例或另一個應用程式所建立的另一個主題樹狀結構重疊的主題樹狀結構。
5. 受管理主題: 使用主題字串 (第一部分符合管理定義的主題物件)，根據主題物件的屬性來管理發佈。

- 例如，如果主題字串的第一部分符合與叢集主題物件相關聯的主題字串，則訂閱可以從叢集的其他成員接收發佈
- 選擇性比對管理定義的主題物件及程式化定義的訂閱，可讓您結合兩者的好處。 管理者提供主題的屬性，程式設計師會動態定義子主題，而不關心主題的管理。
- 它是產生的主題字串，用來比對提供與主題相關聯之屬性的主題物件，但不一定是 sd.Objectname 中所命名的主題物件，雖然它們通常會變成相同。 請參閱 第 680 頁的『範例 2: 發佈至變數主題』。

在此範例中，讓訂閱可延續，在訂閱者使用 MQCO_KEEP_SUB 選項關閉訂閱之後，發佈會繼續傳送至訂閱佇列。 當訂閱者不在作用中時，佇列會繼續接收發佈。 您可以使用 MQSO_PUBLICATIONS_ON_REQUEST 選項來建立訂閱，並使用 MQSUBRQ 來要求保留的發佈資訊，以置換此行為。

稍後可以透過使用 MQCO_RESUME 選項開啟訂閱來回復訂閱。

您可以採用多種方式來使用 MQSUB 所傳回的佇列控點 Hobj。 在範例中使用佇列控點來查詢訂閱佇列的名稱。 使用預設模型佇列 SYSTEM.NDURABLE.MODEL.QUEUE 或 SYSTEM.DURABLE.MODEL.QUEUE 開啟受管理佇列。 您可以置換預設值，方法是在主題上依主題提供您自己的可延續及不可延續模型佇列，作為與訂閱相關聯的主題物件內容。

不論從模型佇列繼承的屬性為何，您都無法重複使用受管理佇列控點來建立額外的訂閱。 您也無法使用傳回的佇列名稱來第二次開啟受管理佇列，以取得受管理佇列的另一個控點。 佇列的行為如同已開啟為專用輸入一樣。

未受管理的佇列比受管理佇列更有彈性。 例如，您可以共用未受管理的佇列，或在一個佇列上定義多個訂閱。 下一個範例，示範如何結合訂閱與未受管理的訂閱佇列。

註: 精簡編碼樣式是為了可讀性而非正式作業用途。

結果顯示在 第 689 頁的圖 73 中。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>

void inquireQname(MQHCONN HConn, MQHOBJ Hobj, MQCHAR48 qName);

int main(int argc, char **argv)
{
    MQCHAR48 topicNameDefault     = "STOCKS";
    char     topicStringDefault[] = "IBM/PRICE";
    MQCHAR48 qmName = "";                 /* Use default queue manager     */
    MQCHAR48 qName = "";                  /* Allocate to query queue name  */
    char     publicationBuffer[101];      /* Allocate to receive messages  */
    char     resTopicStrBuffer[151];   /* Allocate to resolve topic string */

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle             */
    MQHOBJ   Hobj = MQHO_NONE;            /* publication queue handle      */
    MQHOBJ   Hsub = MQSO_NONE;            /* subscription handle           */
    MQLONG   CompCode = MQCC_OK;          /* completion code               */
    MQLONG   Reason = MQRC_NONE;          /* reason code                   */
    MQLONG   messlen = 0;
    MQSD     sd = {MQSD_DEFAULT};         /* Subscription Descriptor       */
    MQMD     md = {MQMD_DEFAULT};         /* Message Descriptor            */
    MQGMO   gmo = {MQGMO_DEFAULT};        /* get message options           */

    char *   topicName   = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   publication = publicationBuffer;
    char *   resTopicStr = resTopicStrBuffer;
    memset(resTopicStr, 0, sizeof(resTopicStrBuffer));

    switch(argc){                /* Replace defaults with args if provided */
        default:
          topicString = argv[2];
        case(2):
          if (strcmp(argv[1],"/"))       /* "/" invalid = No topic object  */
             topicName = argv[1];
          else
             *topicName = '\0';
        case(1):
             printf("Optional parameters: topicName, topicString\nValues \"%s\"  \"%s\"\n",
                   topicName, topicString);
}
```

圖 *71:* 受管理 *MQ* 訂閱者-第 *1* 部分*:* 宣告和參數處理。

在此範例中，有一些關於宣告的其他註解。

MQHOBJ Hobj = MQHO_NONE;
您無法明確開啟不可延續的受管理訂閱佇列來接收發佈，但您確實需要為佇列管理程式在為您開啟佇列時所傳回的物件控點配置儲存體。 請務必將控點起始設定為 MQHO_OBJECT。 這會向佇列管理程式指出它需要將佇列控點傳回訂閱佇列。

MQSD sd = {MQSD_DEFAULT};
在 MQSUB 中使用的新訂閱描述子。

MQCHAR48 qName;
雖然範例不需要訂閱佇列的知識，但範例會查詢訂閱佇列的名稱- MQINQ 連結在 C 語言中有點尷尬，因此您可能會發現範例的這部分很有用。

```
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        strncpy(sd.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
        sd.ObjectString.VSPtr = topicString;
        sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
        sd.Options = MQSO_CREATE | MQSO_MANAGED | MQSO_NON_DURABLE | MQSO_FAIL_IF_QUIESCING ;
        sd.ResObjectString.VSPtr = resTopicStr;
        sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
        MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
        gmo.WaitInterval = 10000;
        inquireQname(Hconn, Hobj, qName);
        printf("Waiting %d seconds for publications matching \"%s\" from \"%-0.48s\"\n",
               gmo.WaitInterval/1000, resTopicStr, qName);
        do {
            memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
            memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
            md.Encoding       = MQENC_NATIVE;
            md.CodedCharSetId = MQCCSI_Q_MGR;
            memset(publicationBuffer, 0, sizeof(publicationBuffer));
            MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1),
                publication, &messlen, &CompCode, &Reason);
            if (Reason == MQRC_NONE)
                printf("Received publication \"%s\"\n", publication);
        }
        while (CompCode == MQCC_OK);
        if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
        MQCLOSE(Hconn, &Hsub, MQCO_REMOVE_SUB, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
    return;
}
void inquireQname(MQHCONN Hconn, MQHOBJ Hobj, MQCHAR48 qName) {
#define _selectors 1
#define _intAttrs 1

    MQLONG select[_selectors] = {MQCA_Q_NAME}; /* Array of attribute selectors */
    MQLONG intAttrs[_intAttrs];                /* Array of integer attributes  */
    MQLONG CompCode, Reason;
    MQINQ(Hconn, Hobj, _selectors, select, _intAttrs, intAttrs, MQ_Q_NAME_LENGTH, qName,
          &CompCode, &Reason);
    if (CompCode != MQCC_OK) {
        printf("MQINQ failed with Condition code %d and Reason %d\n", CompCode, Reason);
        strcpy(qName, "unknown queue");
    }
    return;
}
```

圖 *72:* 受管理 *MQ* 訂閱者-第 *2* 部分*:* 程式碼主體。

```
W:\Subscribe2\Debug>solution2
Optional parameters: topicName, topicString
Values "STOCKS"  "IBM/PRICE"
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from
"SYSTEM.MANAGED.NDURABLE.48A0AC7403300020    "
Received publication "150"
Completion code 0 and Return code 0

W:\Subscribe2\Debug>solution2 / NYSE/IBM/PRICE
Optional parameters: topicName, topicString
Values ""  "NYSE/IBM/PRICE"
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from
"SYSTEM.MANAGED.NDURABLE.48A0AC7403310020    "
Received publication "150"
Completion code 0 and Return code 0
```

圖 *73: MQ* 訂閱者

在此範例中，還有一些關於程式碼的其他註解。

strncpy(sd.ObjectName, topicName, MQ_Q_NAME_LENGTH);
如果 topicName 是空值或空白 *(*預設值*)*，則不會使用主題名稱來計算已解析的主題字串。

sd.ObjectString.VSPtr = topicString;
在此範例中，程式設計師提供由 MQSUB 結合的主題物件和主題字串，而不是僅使用預先定義的主題物件。 請注意，主題字串是 MQCHARV 結構。

sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
設定 MQCHARV 欄位長度的替代方案。

sd.Options = MQSO_CREATE | MQSO_MANAGED | MQSO_NON_DURABLE | MQSO_FAIL_IF_QUIESCING;
在定義主題字串之後， sd.Options 旗標需要最仔細的注意。 有許多選項，範例僅指定最常用的選項。 其他選項使用預設值。

1. 由於訂閱是 不可延續，亦即，它在應用程式中具有開啟訂閱的生命期限，請設定 MQSO_CREATE 旗標。 您也可以設定 *(*預設值*)* MQSO_NON_DURABLE 旗標，以取得可讀性。
2. 補充 MQSO_CREATE 是 MQSO_RESUME。 這兩個旗標可以一起設定; 佇列管理程式會建立新的訂閱或回復現有的訂閱 (以適當的方式)。 不過，如果您指定 MQSO_RESUME ，則即使沒有要回復的訂閱，也必須同時起始設定 sd.SubName 的 MQCHARV 結構。 如果無法起始設定 SubName ，則會從 MQSUB 產生 2440： MQRC_SUB_NAME_ERROR 回覆碼。

   註: 對於不可延續的受管理訂閱，一律會忽略 MQSO_RESUME : 但在未起始設定 sd.SubName 的 MQCHARV 結構的情況下指定它會導致錯誤。
3. 此外，還有第三個旗標會影響訂閱的開啟方式: MQSO_ALTER。 如果提供正確的許可權，則會變更回復訂閱的內容，以符合 MQSUB 中指定的其他屬性。

   註: 必須至少指定其中一個 MQSO_CREATE、 MQSO_RESUME 及 MQSO_ALTER 旗標。 請參閱 選項 (MQLONG)。 第 691 頁的『範例 3: 未受管理的 MQ 訂閱者』中有使用這三個旗標的範例。
4. 設定 MQSO_MANAGED ，讓佇列管理程式自動為您管理訂閱。

sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
選擇性地省略設定以空值結尾的字串的 MQCHARV 長度，並改用空值終止符旗標。

sd.ResObjectString.VSPtr = resTopicStr;
產生的主題字串會在程式的第一個 printf 中回應。 設定 MQCHARV ResObjectString ，讓 IBM MQ 將已解析的字串傳回給程式。

註: 在 memset(resTopicStr, 0, sizeof(resTopicStrBuffer))中，
resTopicStringBuffer 已起始設定為空值。 傳回的主題字串結尾不是尾端空值。

sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
將 sd.ResObjectString 的緩衝區大小設為小於其實際大小。 這會防止在已解析的主題字串填滿整個緩衝區時，改寫所提供的空值終止符。

註: 如果主題字串長於 sizeof(resTopicStrBuffer)-1，則不會傳回任何錯誤。 即使 VSLength > VSBufSiz ， sd.ResObjectString.VSLength 中傳回的長度也是完整字串的長度，不一定是傳回字串的長度。 測試 sd.ResObjectString.VSLength < sd.ResObjectString.VSBufSiz 以確認主題字串已完成。

MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
MQSUB 函數會建立訂閱。 如果不可延續，您可能對其名稱不感興趣，不過您可以在「IBM MQ 檔案總管」中檢查其狀態。 您可以提供 sd.SubName 參數作為輸入，以便您知道要尋找的名稱; 您顯然必須避免與其他訂閱發生名稱衝突。

MQCLOSE(Hconn, &Hsub, MQCO_REMOVE_SUB, &CompCode, &Reason);
關閉訂閱及訂閱佇列是選用的。 在此範例中，訂閱已關閉，但佇列未關閉。 在此情況下， MQCLOSE MQCO_REMOVE_SUB 選項仍然是預設值，因為訂閱不可延續。 使用 MQCO_KEEP_SUB 是一個錯誤。

註: MQSUB 不會關閉訂閱 佇列 ，且其控點 Hobj 會維持有效，直到 MQCLOSE 或 MQDISC 關閉佇列為止。 如果應用程式過早終止，佇列管理程式會在應用程式終止之後某個時間清除佇列及訂閱。

相關概念
第 683 頁的『範例 1: MQ Publication 消費者』
MQ Publication 消費者是不訂閱主題本身的 IBM MQ 訊息消費者。

第 691 頁的『範例 3: 未受管理的 MQ 訂閱者』
未受管理的訂閱者是訂閱者應用程式的重要類別。 使用它，您可以將發佈/訂閱的好處與佇列作業及使用發佈的 控制 相結合。 未受管理的訂閱是應用程式負責的位置。 用於指定儲存訂閱的佇列。 此範例示範結合訂閱與佇列的不同方式。

第 676 頁的『撰寫發佈者應用程式』
透過研究兩個範例，開始撰寫發佈者應用程式。 第一個是盡可能在點對點應用程式將訊息放置在佇列上的模型，第二個示範動態建立主題-發佈者應用程式更常見的型樣。

範例 *3:* 未受管理的 *MQ* 訂閱者

未受管理的訂閱者是訂閱者應用程式的重要類別。 使用它，您可以將發佈/訂閱的好處與佇列作業及使用發佈的 控制 相結合。 未受管理的訂閱是應用程式負責的位置。 用於指定儲存訂閱的佇列。 此範例示範結合訂閱與佇列的不同方式。

與 不可延續相比，未受管理的型樣更常與 可延續 訂閱相關聯。 一般而言，未受管理訂閱者所建立之訂閱的生命週期與訂閱應用程式本身的生命週期無關。 讓訂閱可延續，即使訂閱應用程式不在作用中，訂閱也會接收發佈。

您可以建立可延續的 受管理 訂閱以達到相同的結果，但部分應用程式需要比受管理訂閱更靈活且更能控制佇列及訊息。 對於可延續受管理訂閱，佇列管理程式會為符合訂閱主題的發佈建立永久佇列。 當刪除訂閱時，它會刪除佇列及相關聯的發佈。

如果應用程式及訂閱的生命週期基本上相同，但難以保證，則通常會使用可延續 受管理 訂閱。 讓訂閱可延續，並讓發佈者建立持續發佈，當佇列管理程式或訂閱者提早終止且需要回復時，就不會有遺失的訊息。

對於非 JMS 應用程式或未使用共用訂閱的 JMS 應用程式，佇列管理程式會以無法共用佇列處理的方式，隱含地開啟訂閱者的可延續受管理訂閱佇列。 此外，除非您的應用程式使用 JMS 共用訂閱，否則無法為每一個受管理佇列建立多個訂閱，而且您可能會發現佇列更難管理，因為您對佇列名稱的控制較少。 基於這些原因，請考量 未受管理 MQ 訂閱者是否比 受管理 MQ 訂閱者更適合需要可延續訂閱的應用程式。

第 696 頁的圖 76 中的程式碼示範未受管理的可延續訂閱型樣。 為了方便說明，程式碼也會建立未受管理的不可延續訂閱。 此範例說明下列型樣資料類型:

- 隨需應變訂閱: 訂閱主題字串是動態的。 它們由應用程式在執行時提供。
- 簡化訂閱主題管理: 使用管理定義的主題來定義訂閱主題字串的根部分，以簡化訂閱主題管理。 這會在應用程式中隱藏主題樹狀結構的根部分。 透過隱藏根部分，訂閱者可以部署至不同的主題樹狀結構。
- 彈性訂閱管理: 您可以透過管理方式定義訂閱，或在訂閱者程式中隨需應變建立訂閱。 除了顯示如何建立訂閱的屬性之外，在管理上與以程式化方式建立的訂閱之間沒有差異。 佇列管理程式會自動建立第三種訂閱類型，以配送訂閱。 所有訂閱都會顯示在「IBM MQ 檔案總管」中。
- 訂閱與佇列的彈性關聯: 預先定義的本端佇列由 MQSUB 函數與訂閱相關聯。 有不同的方式可使用 MQSUB 來建立訂閱與佇列的關聯:
  - 將訂閱與 沒有 現有訂閱 `MQSO_CREATE + (Hobj from MQOPEN)`的佇列相關聯。
  - 將 新的 訂閱與具有現有訂閱 `MQSO_CREATE + (Hobj from MQOPEN)`的佇列相關聯。
  - 將現有訂閱移至不同的佇列 `MQSO_ALTER + (Hobj from MQOPEN)`。
  - 回復與現有佇列、 `MQSO_RESUME + (Hobj = MQHO_NONE)`或 `MQSO_RESUME + (Hobj = from MQOPEN of queue with existing subscription)`相關聯的現有訂閱。
  - 透過在不同組合中結合 `MQSO_CREATE | MQSO_RESUME | MQSO_ALTER`，您可以滿足訂閱及佇列的不同輸入狀態，而不需要使用不同的 `sd.Options` 值來撰寫多個 MQSUB 版本的程式碼。
  - 或者，透過撰寫特定選擇的 `MQSO_CREATE | MQSO_RESUME | MQSO_ALTER`，佇列管理程式會傳回錯誤 (第 692 頁的表 123) 如果提供作為 MQSUB 輸入的訂閱及佇列的狀態與 `sd.Options` 的值不一致。 第 698 頁的圖 82 顯示使用 `sd.Options` 旗標的不同個別設定發出 MQSUB for Subscription X 並將三個不同物件控點傳遞給它的結果。

  探索 第 695 頁的圖 75 中範例程式的不同輸入，以熟悉這些不同類型的錯誤。 在表格中列出的案例中未包含的常見錯誤 `RC = 2440` 是訂閱名稱錯誤。 通常是透過 `MQSO_RESUME` 或 `MQSO_ALTER` 傳遞空值或無效的訂閱名稱所造成。
- 多重處理: 您可以在許多消費者之間共用閱讀出版品的工作。 發佈會全部移至與訂閱主題相關聯的單一佇列。 消費者可以選擇使用 MQOPEN 直接開啟佇列，或使用 MQSUB 回復訂閱。

- 訂閱集中: 可以在相同佇列上建立多個訂閱。 請小心使用此功能，因為它可能會導致訂閱重疊，並多次接收相同的發佈。 MQSO_GROUP_SUB 選項可消除重疊訂閱所造成的重複發佈。
- 訂閱者與消費者區隔: 除了範例中所說明的三個消費者模型之外，另一個模型是將消費者與訂閱者區隔。 它是未受管理的 MQ Subscriber 的變異，但不會在相同程式中發出 MQOPEN 和 MQSUB ，而是一個程式會訂閱發佈，而另一個程式會使用它們。 例如，訂閱者可能是發佈/訂閱叢集的一部分，且消費者連接至佇列管理程式叢集外部的佇列管理程式。 消費者透過將訂閱佇列定義為遠端佇列定義，透過標準分散式佇列來接收發佈。

瞭解 MQSO_CREATE | MQSO_RESUME | MQSO_ALTER 的行為非常重要，尤其是當您計劃使用這些選項的組合來簡化程式碼時。 請研究表格 第 692 頁的表 123 ，其中顯示將不同佇列控點傳遞至 MQSUB 的結果，以及將 第 697 頁的圖 77 中所示範例程式執行至 第 698 頁的圖 82 的結果。

用來建構表格的實務範例有一個訂閱 X，以及兩個佇列: A 和 B。 訂閱名稱參數 sd.SubName 設為 X，這是連接至佇列 A 的訂閱名稱。 佇列 B 未連接任何訂閱。

在 第 692 頁的表 123 中， MQSUB 會傳遞訂閱 X 及佇列控點至佇列 A。 訂閱選項的結果如下:

- MQSO_CREATE 失敗，因為佇列控點對應於已訂閱 X 的佇列 A 。 請對照此行為與成功呼叫。 該呼叫成功，因為佇列 B 未附加 X 的訂閱。
- MQSO_RESUME 成功，因為佇列控點對應於已訂閱 X 的佇列 A 。 相反地，當佇列 A 上沒有訂閱 X 時，呼叫會失敗。
- MQSO_ALTER 在開啟訂閱及佇列方面的行為與 MQSO_RESUME 類似。 不過，如果傳遞至 MQSUB 的訂閱描述子所包含的屬性與訂閱的屬性不同，則 MQSO_RESUME 會失敗，而 MQSO_ALTER 只要程式實例具有變更屬性的許可權，就會成功。 請注意，您永遠無法變更訂閱中的主題字串; 但 MQSUB 不會傳回錯誤，而是會忽略訂閱描述子中的主題名稱及主題字串值，並使用現有訂閱中的值。

接下來，請查看 第 692 頁的表 123 ，其中 MQSUB 會傳遞訂閱 X 及佇列控點至佇列 B。 訂閱選項的結果如下:

- MQSO_CREATE 成功並在佇列 B 上建立訂閱 X，因為這是佇列 B 上的新訂閱。
- MQSO_RESUME 失敗。 MQSUB 會在佇列 B 上尋找訂閱 X 並找不到它，但它會傳回 *RC = 2019*-訂閱佇列不符合佇列物件控點，而不是傳回 *RC = 2428*-訂閱 *X* 不存在。 第三個選項 MQSO_ALTER 的行為會建議此非預期錯誤的原因。 MQSUB 預期佇列控點指向具有訂閱的佇列。 它會先檢查此項，然後再檢查 sd.SubName 中所指名的訂閱是否存在。
- MQSO_ALTER 成功，並將訂閱從佇列 A 移至佇列 B。

表格中未顯示的情況是佇列 A 上訂閱的訂閱名稱與 sd.SubName 中的訂閱名稱不符。 該呼叫失敗，且 *RC = 2428*-訂閱 *X* 不存在於佇列 *A* 上。

表 *123:* 來自 *MQSUB* 的錯誤，具有不同的佇列控點及訂閱組合

佇列控點	**佇列 A** **訂閱 X** **佇列 B** **沒有訂閱**	**佇列 A** **沒有訂閱** **佇列 B** **沒有訂閱**
傳遞至 MQSUB 的 佇列 *A* 的 Hobj	**MQSO_CREATE** RC = 2432-佇列 A 上已存在訂閱 X **MQSO_RESUME** 回復佇列 A 上的訂閱 X **MQSO_ALTER** 在佇列 A 上回復訂閱 X 並進行允許的變更	**MQSO_CREATE** 在佇列 A 上建立訂閱 X **MQSO_RESUME** RC = 2428-佇列 A 上不存在訂閱 X **MQSO_ALTER** RC = 2428-佇列 A 上不存在訂閱 X

表 *123:* 來自 *MQSUB* 的錯誤，具有不同的佇列控點及訂閱組合 *(*繼續*)*		
佇列控點	**佇列 A** **訂閱 X** **佇列 B** **沒有訂閱**	**佇列 A** **沒有訂閱** **佇列 B** **沒有訂閱**
傳遞至 MQSUB 的 佇列 *B* 的 Hobj	**MQSO_CREATE** 在佇列 B 上建立新的訂閱 X **MQSO_RESUME** RC = 2019-訂閱佇列不符合佇列物件控點 **MQSO_ALTER** 將訂閱 X 從佇列 A 移至佇列 B	**MQSO_CREATE** 在佇列 B 上建立新的訂閱 X **MQSO_RESUME** RC = 2428-佇列 B 上不存在訂閱 X **MQSO_ALTER** RC = 2428-佇列 B 上不存在訂閱 X
MQHO_NONE 已傳遞至 MQSUB	**MQSO_CREATE** RC = 2019-不當的物件控點: 設定 MQSO_MANAGED 旗標，以建立受管理訂閱及建立受管理佇列 **MQSO_RESUME** 回復佇列 A 上的訂閱 X，並將 Hobj 傳回佇列 A **MQSO_ALTER** 回復佇列 A 上的訂閱 X，將 Hobj 傳回佇列 A，並進行允許的變更	**MQSO_CREATE** RC = 2019-不當的物件控點: 設定 MQSO_MANAGED 旗標，以建立受管理訂閱及建立受管理佇列 **MQSO_RESUME** RC = 2428-無訂閱 X **MQSO_ALTER** RC = 2019-不當的物件控點: 無佇列 A 或 B

註: 精簡編碼樣式是為了可讀性而非正式作業用途。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>

void inquireQname(MQHCONN HConn, MQHOBJ Hobj, MQCHAR48 qName);

int main(int argc, char **argv)
{
    MQCHAR48 topicNameDefault            = "STOCKS";
    char     topicStringDefault[]        = "IBM/PRICE";
    char     subscriptionNameDefault[]   = "IBMSTOCKPRICESUB";
    char     subscriptionQueueDefault[]  = "STOCKTICKER";
    char     publicationBuffer[101];      /* Allocate to receive messages     */
    char     resTopicStrBuffer[151];      /* Allocate to resolve topic string */
    MQCHAR48 qmName = "";                 /* Default queue manager            */
    MQCHAR48 qName = "";                  /* Allocate storage for MQINQ       */

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle                */
    MQHOBJ   Hobj = MQHO_NONE;            /* subscription queue handle        */
    MQHOBJ   Hsub = MQSO_NONE;            /* subscription handle              */
    MQLONG   CompCode = MQCC_OK;          /* completion code                  */
    MQLONG   Reason = MQRC_NONE;          /* reason code                      */
    MQLONG   messlen = 0;
    MQOD     od = {MQOD_DEFAULT};         /* Unmanaged subscription queue     */
    MQSD     sd = {MQSD_DEFAULT};         /* Subscription Descriptor          */
    MQMD     md = {MQMD_DEFAULT};         /* Message Descriptor               */
    MQGMO   gmo = {MQGMO_DEFAULT};        /* get message options              */
    MQLONG   sdOptions = MQSO_CREATE | MQSO_RESUME | MQSO_DURABLE |
MQSO_FAIL_IF_QUIESCING;

    char *   topicName = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   subscriptionName =  subscriptionNameDefault;
    char *   subscriptionQueue = subscriptionQueueDefault;
    char *   publication = publicationBuffer;
    char *   resTopicStr = resTopicStrBuffer;
    memset(resTopicStrBuffer, 0, sizeof(resTopicStrBuffer));
```

圖 *74:* 未受管理的 *MQ* 訂閱者-第 *1* 部分*:* 宣告。

```
    switch(argc){                   /* Replace defaults with args if provided */
default:
        switch((argv[5][0])) {
case('A'):  sdOptions =  MQSO_ALTER  | MQSO_DURABLE | MQSO_FAIL_IF_QUIESCING;
            break;
case('C'):  sdOptions =  MQSO_CREATE | MQSO_DURABLE | MQSO_FAIL_IF_QUIESCING;
            break;
case('R'):  sdOptions =  MQSO_RESUME | MQSO_DURABLE | MQSO_FAIL_IF_QUIESCING;
            break;
default:    ;
        }
case(5):
    if (strcmp(argv[4],"/")) /* "/" invalid = No subscription  */
        subscriptionQueue = argv[4];
    else {
        *subscriptionQueue = '\0';
        if (argc > 5) {
            if (argv[5][0] == 'C') {
                sdOptions = sdOptions + MQSO_MANAGED;
            }
        }
        else
            sdOptions = sdOptions + MQSO_MANAGED;
    }

case(4):
    if (strcmp(argv[3],"/")) /* "/" invalid = No subscription  */
        subscriptionName = argv[3];
    else {
        *subscriptionName = '\0';
        sdOptions = sdOptions - MQSO_DURABLE;
    }
case(3):
    if (strcmp(argv[2],"/")) /* "/" invalid = No topic string  */
        topicString = argv[2];
    else
        *topicString = '\0';
case(2):
    if (strcmp(argv[1],"/")) /* "/" invalid = No topic object  */
        topicName = argv[1];
    else
        *topicName = '\0';
case(1):
    sd.Options = sdOptions;
    printf("Optional parameters: "
    printf("topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|
R(esume)\n");
    printf("Values \"%-.48s\"  \"%s\" \"%s\"  \"%-.48s\" sd.Options=%d\n",
           topicName, topicString, subscriptionName, subscriptionQueue, sd.Options);
}
```

圖 *75:* 未受管理的 *MQ* 訂閱者-第 *2* 部分*:* 參數處理。

在此範例中，有關參數處理的其他註解如下:

switch((argv[5][0]))
您可以選擇在參數 5 中輸入 A lter | C reate | R esume，以測試置換範例中依預設使用的部分 MQSUB 選項設定的效果。 範例所使用的預設值是 MQSO_CREATE | MQSO_RESUME | MQSO_DURABLE。

註: 設定 MQSO_ALTER 或 MQSO_RESUME 而不設定 MQSO_DURABLE 是一個錯誤，必須設定 sd.SubName 並參照可回復或變更的訂閱。

***subscriptionQueue = '\0';**
sdOptions = sdOptions + MQSO_MANAGED;
如果預設訂閱佇列 STOCKTICKER 取代為空字串，則只要設定 MQSO_CREATE，範例就會設定 MQSO_MANAGED 旗標並建立動態訂閱佇列。 如果在第五個參數中設定 Alter or Resume，則範例的行為取決於 subscriptionName 的值。

***subscriptionName = '\0';**
sdOptions = sdOptions - MQSO_DURABLE;
如果預設訂閱 IBMSTOCKPRICESUB 取代為空字串，則範例會移除 MQSO_DURABLE 旗標。 如果您執行範例來提供其他參數的預設值，則會建立指向 STOCKTICKER 的額外暫時訂閱，並接收重複的發佈。 下次執行不含任何參數的範例時，您只會再次收到一個發佈。

```
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        if (strlen(subscriptionQueue)) {
            strncpy(od.ObjectName, subscriptionQueue, MQ_Q_NAME_LENGTH);
            MQOPEN(Hconn, &od, MQOO_INPUT_AS_Q_DEF | MQOO_FAIL_IF_QUIESCING | MQOO_INQUIRE,
                  &Hobj, &CompCode, &Reason);
            if (CompCode != MQCC_OK) break;
        }
        strncpy(sd.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
        sd.ObjectString.VSPtr = topicString;
        sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
        sd.SubName.VSPtr = subscriptionName;
        sd.SubName.VSLength = MQVS_NULL_TERMINATED;
        sd.ResObjectString.VSPtr = resTopicStr;
        sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
        MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
        gmo.WaitInterval = 10000;
        gmo.MatchOptions = MQMO_MATCH_CORREL_ID;
        memcpy(md.CorrelId, sd.SubCorrelId, MQ_CORREL_ID_LENGTH);
        inquireQname(Hconn, Hobj, qName);
        printf("Waiting %d seconds for publications matching \"%s\" from %-0.48s\n",
                gmo.WaitInterval/1000, resTopicStr, qName);
        do {
            memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
            memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
            md.Encoding       = MQENC_NATIVE;
            md.CodedCharSetId = MQCCSI_Q_MGR;
            MQGET(Hconn, Hobj, &md, &gmo, sizeof(publication), publication, &messlen,
&CompCode, &Reason);
            if (Reason == MQRC_NONE)
                printf("Received publication \"%s\"\n", publication);
        }
        while (CompCode == MQCC_OK);
        if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
        MQCLOSE(Hconn, &Hsub, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
void inquireQname(MQHCONN Hconn, MQHOBJ Hobj, MQCHAR48 qName) {
#define _selectors 1
#define _intAttrs 1

    MQLONG select[_selectors] = {MQCA_Q_NAME}; /* Array of attribute selectors    */
    MQLONG intAttrs[_intAttrs];            /* Array of integer attributes           */
    MQLONG CompCode, Reason;
    MQINQ(Hconn, Hobj, _selectors, select, _intAttrs, intAttrs, MQ_Q_NAME_LENGTH, qName,
&CompCode, &Reason);
    if (CompCode != MQCC_OK) {
        printf("MQINQ failed with Condition code %d and Reason %d\n", CompCode, Reason);
        strncpy(qName, "unknown queue", MQ_Q_NAME_LENGTH);
    }
    return;
}
```

圖 *76:* 未受管理的 *MQ* 訂閱者-第 *3* 部分*:* 程式碼主體。

在此範例中，程式碼的其他註解如下:

if (strlen(subscriptionQueue))
如果沒有訂閱佇列名稱，則範例會使用 MQHO_NONE 作為 Hobj 的值。

MQOPEN(...);
即會開啟訂閱佇列，並將佇列控點儲存在 Hobj 中。

MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
使用從 MQOPEN 傳遞的 Hobj 來開啟訂閱 (如果沒有訂閱佇列名稱，則為 MQHO_NONE)。 可以回復未受管理的佇列，而不需要明確使用 MQOPEN 來開啟它。

MQCLOSE(Hconn, &Hsub, MQCO_NONE, &CompCode, &Reason);
使用訂閱控點來關閉訂閱。 視訂閱是否可延續而定，會以隱含的 MQCO_KEEP_SUB 或 MQCO_REMOVE_SUB 來關閉訂閱。 您可以使用 MQCO_REMOVE_SUB 來關閉可延續訂閱，但無法使用 MQCO_KEEP_SUB 來關閉不可延續訂閱。 MQCO_REMOVE_SUB 的動作是移除訂閱，這會停止將任何進一步的發佈傳送至訂閱佇列。

MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
如果訂閱未受管理，則不會採取任何特殊動作。 如果佇列受管理，且訂閱已使用明確或隱含 MQCO_REMOVE_SUB 關閉，則此時會從佇列中清除所有發佈，並刪除佇列。

gmo.MatchOptions = MQMO_MATCH_CORREL_ID;
memcpy(md.CorrelId, sd.SubCorrelId, MQ_CORREL_ID_LENGTH);
請確定收到的訊息是我們訂閱的訊息。

範例中的結果說明發佈/訂閱的層面:

在 第 697 頁的圖 77 中，範例從在 NYSE/IBM/PRICE 主題上發佈 130 開始。

```
W:\Subscribe3\Debug>..\..\Publish2\Debug\publishstock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

圖 *77:* 發佈 *130* 至 *NYSE/IBM/PRICE*

在 第 697 頁的圖 78 執行使用預設參數的範例時，會收到保留的發佈資訊 130。 所提供的主題物件和主題字串會被忽略，如 第 698 頁的圖 82 所示。 提供訂閱物件時，一律會從訂閱物件取得主題物件和主題字串，且主題字串是不可變的。 範例的實際行為取決於 MQSO_CREATE、 MQSO_RESUME 和 MQSO_ALTER 的選項或組合。 在此範例中， MQSO_RESUME 是選取的選項。

```
W:\Subscribe3\Debug>solution3
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8206
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0
```

圖 *78:* 接收保留的發佈

In (第 697 頁的圖 79) 未收到任何發佈，因為可延續訂閱已收到保留的發佈。 在此範例中，透過僅提供不含佇列名稱的訂閱名稱來回復訂閱。 如果已提供佇列名稱，則會先開啟佇列，並將控點傳遞給 MQSUB。

註: 來自 MQINQ 的 2038 錯誤是由於 MQSUB 不包括 MQOO_INQUIRE 選項的 STOCKTICKER 的隱含 MQOPEN 所造成。 透過明確開啟佇列，避免來自 MQINQ 的 2038 回覆碼。

```
W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE IBMSTOCKPRICESUB / Resume
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "IBMSTOCKPRICESUB" "" sd.Options=8204
MQINQ failed with Condition code 2 and Reason 2038
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from unknown queue
Completion code 0 and Return code 0
```

圖 *79:* 回復訂閱

在 第 698 頁的圖 80 中，範例會使用 STOCKTICKER 作為目的地來建立不可延續的未受管理訂閱。 因為這是新的訂閱，所以它會接收保留的發佈。

```
W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE / STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "" "STOCKTICKER" sd.Options=8194
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0
```

圖 *80:* 接收具有新的未受管理不可延續訂閱的保留發佈

在 第 698 頁的圖 81 中，為了示範重疊訂閱，會傳送另一個發佈，變更保留的發佈。 接下來，透過不提供訂閱名稱來建立新的不可延續未受管理的訂閱。 保留的發佈會收到兩次，一次是針對新訂閱，一次是針對 STOCKTICKER 佇列上仍在作用中的可延續 IBMSTOCKPRICESUB 訂閱。 範例是一個圖解，說明它是具有訂閱的佇列，而不是應用程式。 儘管在此應用程式呼叫中未參照 IBMSTOCKPRICESUB 訂閱，應用程式仍會收到發佈兩次: 一次來自以管理方式建立的可延續訂閱，一次來自應用程式本身所建立的不可延續訂閱。

```
W:\Subscribe3\Debug>..\..\Publish2\Debug\publishstock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE / STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "" "STOCKTICKER" sd.Options=8194
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Received publication "130"
Completion code 0 and Return code 0
```

圖 *81:* 重疊訂閱

在 第 698 頁的圖 82 中，範例示範提供新的主題字串及現有訂閱不會導致訂閱變更。

1. 在第一個案例中，如您所預期， `Resume` 會回復現有的訂閱，並忽略已變更的主題字串。
2. 在第二種情況下， `Alter` 會導致錯誤: `RC = 2510, Topic not alterable`。
3. 在第三個範例中， `Create` 會導致錯誤 `RC = 2432, Sub already exists`。

```
W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Resume
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8204
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0

W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Alter
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8201
Completion code 2 and Return code 2510

W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8202
Completion code 2 and Return code 2432
```

圖 *82:* 無法變更訂閱主題

相關概念

第 683 頁的『範例 1: MQ Publication 消費者』
MQ Publication 消費者是不訂閱主題本身的 IBM MQ 訊息消費者。

第 686 頁的『範例 2: 受管理 MQ 訂閱者』
受管理 MQ 訂閱者是大部分訂閱者應用程式的偏好型樣。 受管理訂閱是 IBM MQ 處理訂閱並為您執行登錄及取消登錄的訂閱。 此範例 不 需要佇列、主題或訂閱的管理定義。

第 676 頁的『撰寫發佈者應用程式』

透過研究兩個範例，開始撰寫發佈者應用程式。 第一個是盡可能在點對點應用程式將訊息放置在佇列上的模型，第二個示範動態建立主題-發佈者應用程式更常見的型樣。

發佈/訂閱生命週期

在設計發佈/訂閱應用程式時，請考量主題、訂閱、訂閱者、發佈、發佈者及佇列的生命週期。

物件 (例如訂閱) 的生命週期開始於建立它，結束於刪除它。 它也可能包括它所經歷的其他狀態及變更，例如暫時暫停、具有母項及子項主題、到期及刪除。

傳統 IBM MQ 物件 (例如佇列) 是以管理方式建立，或由使用「可程式化指令格式 (PCF)」的管理程式建立。 發佈/訂閱的不同在於提供 MQSUB 和 MQCLOSE API 動詞來建立及刪除訂閱，具有受管理訂閱的概念，不僅會建立及刪除佇列，還會清除未耗用的訊息，以及在以管理方式建立的主題物件與以程式化方式或以管理方式建立的主題字串之間具有關聯。

這種豐富功能可滿足廣泛的發佈/訂閱需求，同時簡化設計發佈/訂閱應用程式的一些常見型樣。 例如，受管理訂閱可簡化訂閱的程式設計及管理，只要建立訂閱的程式即可。 未受管理的訂閱可簡化在訂閱與耗用發佈之間的連線較鬆散的程式設計。 集中建立的訂閱很有用，其中型樣是根據集中的控制模型將發佈資料流量遞送至消費者的其中一個，例如將航班資訊傳送至自動化閘道，而如果閘道人員負責在閘道輸入航班號碼來訂閱該航班的乘客記錄，則可以使用以程式化方式建立的訂閱。

在此最後一個範例中，受管理可延續訂閱可能是適當的: 受管理，因為經常建立訂閱，而且閘道關閉且可以程式化方式移除訂閱時具有明確的端點; 可延續，以避免因閘道訂閱者程式因某種原因而關閉而遺失乘客記錄 [8] 為了開始向閘口公佈乘客記錄，可能的設計是，閘口申請既使用閘口號碼訂閱乘客記錄，又使用閘口號碼公佈閘口事件。 出版商會透過發佈乘客記錄來回應登機門開啟事件，然後可能還會傳送至其他相關方，例如計費，以記錄正在進行的航班，以及客戶服務，將登機門號碼的文字通知給乘客的行動電話。

集中管理的訂閱可能使用可延續的未受管理模型，使用每個閘道的預先定義佇列將乘客清單遞送至閘道。

下列三個發佈/訂閱生命週期範例說明受管理不可延續、受管理可延續及未受管理可延續訂閱者如何與訂閱、主題、佇列、發佈者及佇列管理程式互動，以及如何在管理與訂閱者程式之間劃分責任。

受管理不可延續訂閱者

第 700 頁的圖 83 顯示應用程式建立受管理的不可延續訂閱，取得兩則發佈至訂閱中所識別主題的訊息，並終止。 以斜體灰色字型加上帶點箭頭來標示的互動是隱含的。

有幾點需要注意。

1. 應用程式會針對已發佈至兩次的主題建立訂閱。 當訂閱者接收其第一個發佈時，它會接收 第二個 發佈，這是目前保留的發佈。
2. 佇列管理程式會建立暫時訂閱佇列，以及建立主題的訂閱。
3. 訂閱已到期。 當訂閱到期時，不會將主題上的其他發佈傳送至這個訂閱，但訂閱者會在訂閱過期之前繼續取得已發佈的訊息。 訂閱期限不會影響發佈期限。
4. 第四個發佈不會放置在訂閱佇列上，因此最後一個 MQGET 不會傳回發佈。
5. 雖然訂閱者會關閉其訂閱，但不會關閉與佇列或佇列管理程式的連線。
6. 在應用程式終止之後不久，佇列管理程式即會清除。 因為訂閱是受管理且不可延續，所以會刪除訂閱佇列。

[8] 當然，發佈者必須傳送乘客記錄作為持續訊息，以避免其他可能的失敗。

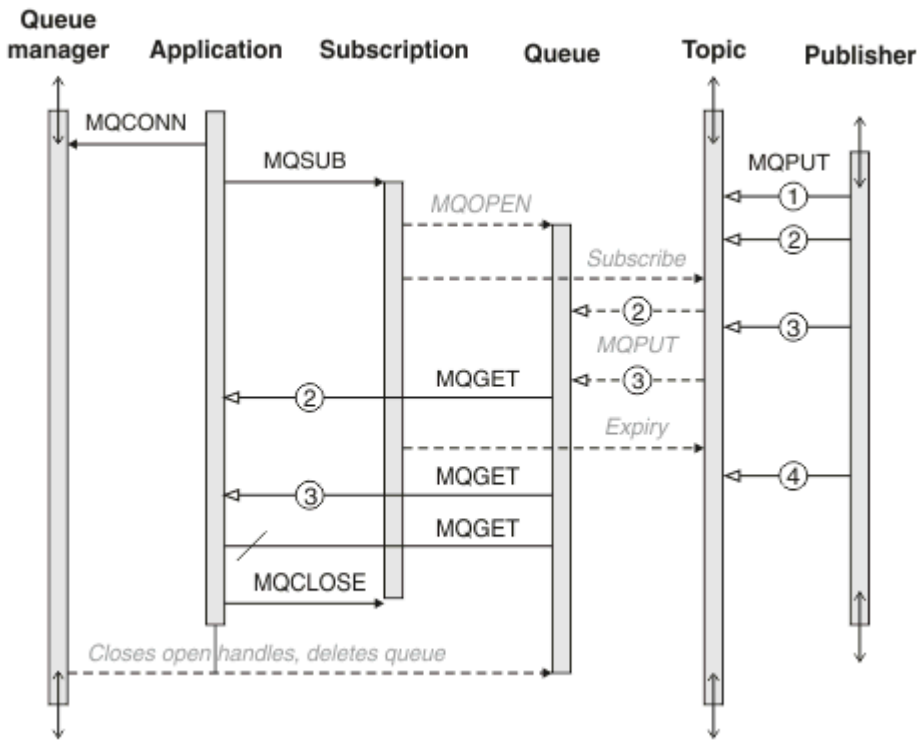


圖 *83:* 受管理不可延續訂閱者生命線

受管理可延續訂閱者

受管理可延續訂閱者進一步採用前一個範例，並顯示在訂閱應用程式終止及重新啟動之後仍存在的受管理訂閱。

有一些新的要點需要注意。

1. 在此範例中，與最後一個不同，發佈主題在訂閱中定義之前並不存在。
2. 訂閱者第一次終止時，會使用選項 MQCO_KEEP_SUB 來關閉訂閱。 這是隱含地關閉受管理可延續訂閱的預設行為。
3. 當訂閱者回復訂閱時，會重新開啟訂閱佇列。
4. 新的發佈 2 在重新開啟之前放置在佇列中，可供 MQGET 使用，即使已移除訂閱也一樣。

   即使訂閱可延續，只有在訂閱可延續且訊息持續 兩者 時，訂閱者才會可靠地接收發佈者所傳送的所有訊息。 訊息持續性取決於發佈者所傳送訊息的 MQMD 中 Persistent 欄位的設定。 訂閱者無法對此進行控制。
5. 使用旗標 MQCO_REMOVE_SUB 關閉訂閱會移除訂閱，並停止將任何其他發佈放置在訂閱佇列上。 當訂閱佇列關閉時，佇列管理程式會移除未讀取的發佈資訊 3，然後刪除佇列。 此動作相當於以管理方式刪除訂閱。

   註: 請勿手動刪除佇列，或使用選項 MQCO_DELETE 或 MQCO_PURGE_DELETE 發出 MQCLOSE 。 受管理訂閱的可見實作詳細資料不是受支援 IBM MQ 介面的一部分。 佇列管理程式管理無法可靠地管理訂閱，除非它具有完全控制。

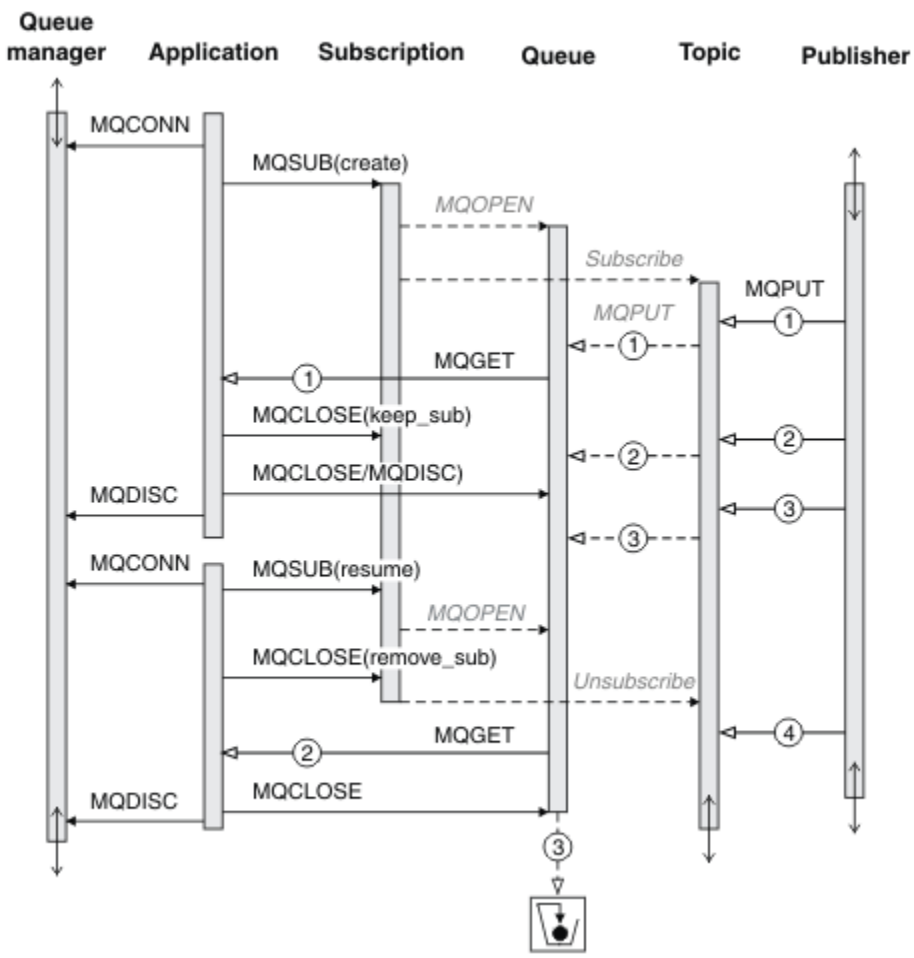


圖 *84:* 受管理可延續訂閱者生命線

未受管理的可延續訂閱者

在第三個範例中新增了管理者: 未受管理的可延續訂閱者。 這是一個很好的範例，顯示管理者如何與發佈/訂閱應用程式互動。

會列出要注意的點。

1. 發佈者會將訊息 1 放置到稍後與用於訂閱的主題物件相關聯的主題。 topic 物件定義的主題字串符合使用萬用字元發佈至的主題。
2. 主題具有保留的發佈資訊。
3. 管理者會建立主題物件、佇列及訂閱。 在訂閱之前，必須先定義主題物件和佇列。
4. 應用程式會開啟與訂閱相關聯的佇列，並傳遞 MQSUB 佇列的控點。 或者，它可以直接開啟訂閱，將佇列控點 MQHO_NONE 傳遞給它。 相反不是 true ，它無法透過只傳遞沒有訂閱名稱的佇列控點來回復訂閱-佇列可能有多個訂閱。
5. 應用程式會使用 MQSO_RESUME 選項來開啟訂閱，即使它是第一次開啟訂閱也一樣。 它正在回復以管理方式建立的訂閱。
6. 訂閱者會接收保留的發佈資訊 1。 發佈 2 雖然在訂閱者收到任何發佈之前發佈，但在訂閱啟動之後發佈，並且是訂閱佇列上的第二個發佈。

   註: 如果保留的發佈資訊未發佈為持續訊息，則在佇列管理程式重新啟動之後會遺失。
7. 在此範例中，訂閱是可延續的。 程式可以建立未受管理的不可延續訂閱; 很明顯這不是管理者可以執行的動作。

8. 選項 MQCO_REMOVE_SUB 在關閉訂閱時的作用是移除訂閱，就像管理者已刪除訂閱一樣。 這會停止任何進一步傳送至佇列的發佈，但不會影響已在佇列上的發佈，即使佇列已關閉也一樣，與 受管理 可延續訂閱不同。
9. 管理者稍後會刪除剩餘訊息 3，並刪除佇列。

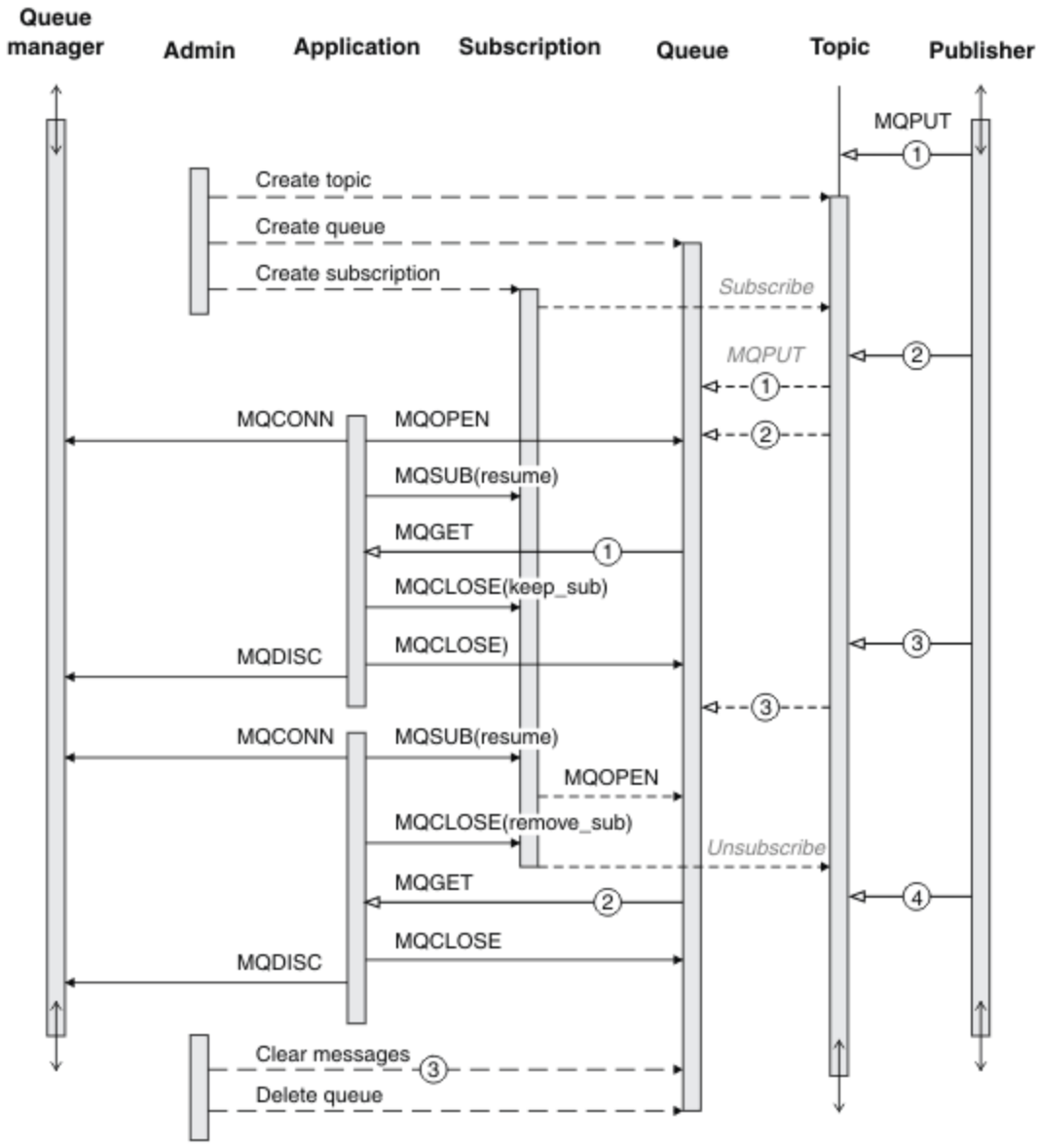


圖 *85:* 未受管理的可延續訂閱者生命線

未受管理訂閱的一般型樣是由管理者執行佇列及訂閱內部管理。 通常不會嘗試模擬受管理訂閱者的行為，並以程式設計方式在應用程式碼中清理佇列及訂閱。 如果您發現自己需要撰寫管理邏輯，請質疑您是否可以使用受管理型樣來達到相同的結果。 不易寫入緊密同步，完全可靠的管理程式碼。 當您可以確定不論訊息、訂閱及佇列的狀態為何，都可以直接刪除訊息、訂閱及佇列時，稍後再手動或使用自動化管理程式更容易進行清理。

發佈/訂閱訊息內容

數個訊息內容與 IBM MQ 發佈/訂閱傳訊相關。

PubAccounting 記號

這是符合此訂閱之所有發佈訊息的「訊息描述子 (MQMD)」的 AccountingToken 欄位中的值。 AccountingToken 是訊息身分環境定義的一部分。 如需訊息環境定義的相關資訊，請參閱 第 40 頁的『訊息環境定義』。 如需 MQMD 中 AccountingToken 欄位的相關資訊，請參閱 AccountingToken。

PubApplIdentityData

這是所有符合此訂閱之發佈訊息的「訊息描述子 (MQMD)」的 ApplIdentity 資料欄位中的值。 ApplIdentity 資料是訊息身分環境定義的一部分。 如需訊息環境定義的相關資訊，請參閱 第 40 頁的『訊息環境定義』。 如需 MQMD 中 ApplIdentity 資料欄位的相關資訊，請參閱 ApplIdentity 資料。

如果未指定選項 MQSO_SET_IDENTITY_CONTEXT，則將在針對此訂閱發佈的每一則訊息中設定的 ApplIdentity 資料為空白，作為預設環境定義資訊。

如果指定選項 MQSO_SET_IDENTITY_CONTEXT，則使用者正在產生 PubApplIdentityData，且此欄位是輸入欄位，其中包含要在此訂閱的每個發佈中設定的 ApplIdentity 資料。

PubPriority

這是符合此訂閱之所有發佈訊息的「訊息描述子 (MQMD)」的「優先順序」欄位中的值。 如需 MQMD 中「優先順序」欄位的相關資訊，請參閱 優先順序。

值必須大於或等於零; 零是最低優先順序。 也可以使用下列特殊值:

- MQPRI_PRIORITY_AS_Q_DEF-當在 MQSUB 呼叫的 Hobj 欄位中提供訂閱佇列時，如果不是受管理控點，則會從這個佇列的 DefProirity 屬性取得訊息的優先順序。 如果如此識別的佇列是叢集佇列，或佇列名稱解析路徑中有多個定義，則會在將發佈訊息放入佇列時決定優先順序，如 MQMD 中的 優先順序 所述。 如果 MQSUB 呼叫使用受管理控點，則會從與所訂閱主題相關聯之模型佇列的 DefPriority 屬性取得訊息的優先順序。
- MQPRI_PRIORITY_AS_PUBLISHED-訊息的優先順序是原始發佈的優先順序。 這是此欄位的起始值。

SubCorrelId

小心: 相關性 ID 只能在發佈/訂閱叢集中的佇列管理程式之間傳遞，不能在階層之間傳遞。

傳送以符合此訂閱的所有發佈將在訊息描述子中包含此相關性 ID。 如果多個訂閱使用相同的佇列來取得其發佈，則依相關性 ID 使用 MQGET 只容許取得特定訂閱的發佈。 此相關性 ID 可以由佇列管理程式或使用者產生。

如果未指定選項 MQSO_SET_CORREL_ID，則佇列管理程式會產生相關性 ID，且此欄位是一個輸出欄位，其中包含將在針對此訂閱所發佈的每一則訊息中設定的相關性 ID。

如果指定選項 MQSO_SET_CORREL_ID，使用者會產生相關性 ID，且此欄位是輸入欄位，其中包含要在此訂閱的每一個發佈中設定的相關性 ID。 在此情況下，如果欄位包含 MQCI_NONE，則將在針對此訂閱發佈的每一則訊息中設定的相關性 ID，將是訊息原始放置所建立的相關性 ID。

如果指定選項 MQSO_GROUP_SUB，且指定的相關性 ID 與使用相同佇列及重疊主題字串的現有分組訂閱相同，則只有群組中最重要的訂閱才會隨附發佈的副本。

SubUserData

這是訂閱使用者資料。 在此欄位中的訂閱上提供的資料將併入作為傳送至此訂閱的每個發佈資訊的 MQSubUser 資料訊息內容。

發佈內容

第 704 頁的表 124 列出發佈訊息隨附的發佈內容。

您可以直接從 **MQRFH2** 資料夾存取這些內容，或使用 MQINQMP 來擷取它們。 MQINQMP 接受內容名稱或 **MQRFH2** 名稱作為要查詢之內容的名稱。

表 *124:* 發佈內容			
內容名稱	**MQRFH2 名稱**	**類型**	**說明**
MQTopicString	mqps.Top	MQTYPE_STRING	主題字串
MQSubUserData	mqps.Sud	MQTYPE_STRING	訂閱者使用者資料
MQIsRetained	mqps.Ret	MQTYPE_BOOLEAN	保留的發佈資訊
MQPubOptions	mqps.Pub	MQTYPE_INT32	發佈選項
MQPubLevel	mqps.Pbl	MQTYPE_INT32	發佈層次
MQPubTime	mqpse.Pts	MQTYPE_STRING	發佈時間
MQPubSeqNum	mqpse.Seq	MQTYPE_INT32	發佈序號
MQPubStrIntData	mqpse.Sid	MQTYPE_STRING	發佈者新增的字串/整數資料
MQPubFormat	mqpse.Pfmt	MQTYPE_INT32	訊息格式: MQRFH1 MQRFH2 PCF

訊息排序

對於特定主題，佇列管理程式會依照從發佈應用程式收到訊息的相同順序來發佈訊息 (根據訊息優先順序來重新排序)。

訊息排序通常表示每一個訂閱者從特定主題上的特定佇列管理程式，以該發佈者發佈訊息的順序接收來自特定發佈者的訊息。

不過，如同所有 IBM MQ 訊息一樣，有時可能會不正常遞送訊息。 在下列狀況中可能會發生這種情況:

- 如果網路中的鏈結關閉，且後續訊息會沿著另一個鏈結重新遞送
- 如果佇列變成暫時已滿或禁止放置，則會將訊息放入無法傳送郵件的佇列，因此會延遲，而後續訊息則會直接傳遞。
- 如果管理者在發佈者和訂閱者仍在運作時刪除佇列管理程式，則會導致將佇列訊息放入無法傳送郵件的佇列，並岔斷訂閱。

如果無法發生這些情況，則一律會依序遞送發佈。

註: 無法搭配使用已分組或分段的訊息與「發佈/訂閱」。

截取發佈

您可以截取發佈，修改它，然後在它到達任何其他訂閱者之前重新發佈它。

您可能想要在發佈到達訂閱者之前截取發佈，以便執行下列其中一個動作:

- 將其他資訊附加至訊息
- 封鎖訊息
- 轉換訊息

您可以對每一個訊息執行相同的作業，或根據訂閱、訊息或訊息標頭來改變作業。

相關參考

MQ_PUBLISH_EXIT-發佈結束程式

訂閱層次
設定訂閱的訂閱層次，以在發佈到達其最終訂閱者之前截取發佈。 截取訂閱者在較高的訂閱層次訂閱，並在較低的發佈層次重新發佈。 建置截取訂閱者的鏈結，以在將訊息遞送至最終訂閱者之前對發佈執行訊息處理。

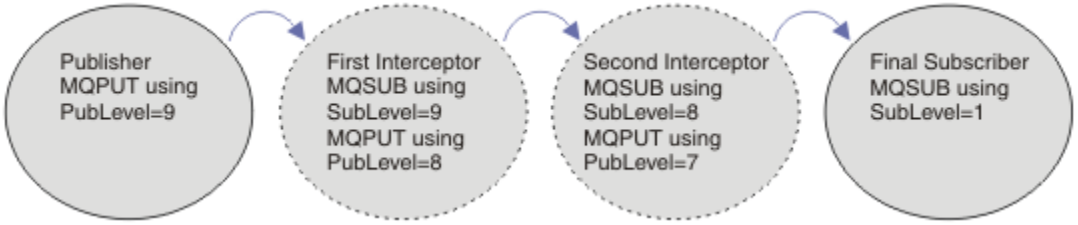


圖 *86:* 截取訂閱者的順序

若要截取發佈資訊，請使用 **MQSD** SubLevel 屬性。 在截取訊息之後，可以透過變更 **MQPMO** PubLevel 屬性來轉換訊息，然後在較低的發佈層次重新發佈。 然後，訊息會傳送至最終訂閱者，或由較低訂閱層次的中間訂閱者再次截取。

截取訂閱者通常會在重新發佈訊息之前先轉換訊息。 一連串截取訂閱者會形成訊息流程。 或者，您可能不會重新發佈所截取的發佈: 較低訂閱層次的訂閱者不會收到訊息。

請確定攔截程式在任何其他訂閱者之前收到發佈。 將攔截程式的訂閱層次設為高於其他訂閱者。 依預設，訂閱者具有 1 的 SubLevel 。 最高值為 9。 發佈必須以至少與最高 SubLevel 一樣高的 PubLevel 開始。 一開始使用 9 的預設 PubLevel 進行發佈。

- 如果您有一個主題的截取訂閱者，請將 SubLevel 設為 9。
- 對於主題上的多個截取應用程式，請針對每一個後續截取訂閱者設定較低的 SubLevel 。
- 您可以實作最多截取 8 個應用程式，訂閱層次從 9 向下至 2 (含)。 訊息的最終收件者具有 1 的 SubLevel 。

訂閱層次最高的攔截程式等於或低於發佈的 PubLevel 會先接收發佈。 針對特定訂閱層次的主題，只配置一個截取訂閱者。 在特定訂閱層次具有多個訂閱者會導致將發佈的多個副本傳送至最終訂閱應用程式集。

使用 SubLevel 為 0 的訂閱者作為擷取並更新。 如果沒有最終訂閱者取得訊息，則它會接收發佈。 SubLevel 為 0 的訂閱者可用來監視未收到任何其他訂閱者的發佈。

對攔截使用者進行程式設計

使用 第 705 頁的表 125 中說明的訂閱選項。

表 *125:* 用於截取訂閱者的訂閱選項

訂閱選項	**附註**
MQSO_SET_CORREL_ID 和 SubCorrelId 設為 MQCI_NONE	保持所截取發佈資訊的 CorrelId 與原始發佈資訊相同。 **註:** 您無法在階層中傳遞發佈的相關性 ID。 佇列管理程式會使用此欄位。
PubPriority 設為 MQPRI_PRIORITY_AS_PUBLISHED	保持所截取發佈資訊的優先順序與原始發佈資訊相同。

第 705 頁的表 125 中的選項必須供所有截取訂閱者使用。 結果是不會從原始發佈者的設定修改相關性 ID 和訊息優先順序。

當截取訂閱者已處理發佈時，它會將訊息重新發佈至其本身訂閱的 PubLevel 低於 SubLevel 的相同主題。 如果截取訂閱者設定 9 的 SubLevel ，則它會以 8 的 PubLevel 重新發佈訊息。

若要正確重新發佈訊息，需要原始發佈資訊中的數個資訊片段。 重複使用與原始訊息中相同的 **MQMD** ，並設定 MQPMO_PASS_ALL_CONTEXT ，以確保將 **MQMD** 中的所有資訊傳遞給下一個訂閱者。 將 第 706 頁的表 126 中所顯示訊息內容的值複製到重新發佈之訊息的對應欄位。 截取訂閱者可以變更這些值。 使用 OR 運算子，將其他值新增至 **MQPMO**。 **選項** 欄位，結合放置訊息選項。

您必須明確開啟發佈佇列，而不是使用受管理發佈佇列。 您無法為受管理佇列設定 MQSO_SET_CORREL_ID。 您也無法在受管理佇列上設定 MQOO_SAVE_ALL_CONTEXT。 請參閱 第 706 頁的『範例』 中列出的程式碼片段。

表 *126:* 重新發佈訊息的 *MQPUT* 值

使用 MQPUT 重新發佈訊息	**發佈訊息中的資訊**
MQOD. ObjectString	訊息內容 MQTopicString
MQPMO. Options	訊息內容 MQPubOptions

最終訂閱者可以選擇以不同方式設定其訂閱選項。 例如，它可能會明確地設定發佈優先順序，而不是 MQPRI_PRIORITY_AS_PUBLISHED。 最終訂閱者的設定只會影響來自鏈中最終截取訂閱者的發佈。

保留的發佈

在截取保留的發佈資訊之後，必須將原始放置訊息選項複製到重新發佈的訊息中，以保留該發佈資訊。

MQPMO_RETAIN 選項由發佈者設定。 每一個截取訂閱者必須將 MQPubOptions 傳送至重新發佈訊息的 put-message 選項，如 第 706 頁的表 126 所示。 複製 put-message 選項會保留原始發佈者所設定的選項，包括是否保留發佈。

當一個出版物完成它向下的截聽使用者鏈，並被傳遞給最終的使用者時，它最終被保留。 在 SubLevel 1 上，要求保留的發佈資訊的新訂閱者會收到它，而不會有任何進一步的截取。 SubLevel 大於 1 的訂閱者不會傳送保留的發佈。 因此，保留的發佈資訊不會被第二次攔截訂閱者的鏈所修改。

範例

範例是可結合以建置截取訂閱者的程式碼片段。 程式碼撰寫為簡短，而不是正式作業品質。

第 706 頁的圖 87 中的前處理器指引定義要從 MQINQMP MQI 呼叫所需的發佈訊息中擷取的兩個內容。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
#define      MQPUBOPTIONS        (MQPTR)(char*) "MQPubOptions",\
                                 0,\
                                 12,\
                                 MQVS_NULL_TERMINATED,\
                                 MQCCSI_APPL
#define      MQTOPICSTRING       (MQPTR)(char*) "MQTopicString",\
                                 0,\
                                 13,\
                                 MQVS_NULL_TERMINATED,\
                                 MQCCSI_APPL
```

圖 *87:* 前處理器指引

第 707 頁的圖 88 列出程式碼片段中使用的宣告。 除了強調顯示的術語之外，宣告是 IBM MQ 應用程式的標準。

已起始設定強調顯示的「放置」及「取得」選項，以傳遞所有環境定義。 強調顯示的 MQTOPICSTRING 及 MQPUBOPTIONS 是前處理器指引中所定義內容名稱的 MQCHARV 起始設定程式。 名稱會傳遞至 MQINQMP。

```
int main(int argc, char **argv) {
    MQLONG   Reason            = MQRC_NONE;
    MQLONG   CompCode          = MQCC_OK;
    MQHCONN  Hcon              = MQHC_UNUSABLE_HCONN;
    MQCHAR   QMName[49]        = " ";
    MQCMHO   CrtMsgHOpts       = {MQCMHO_DEFAULT};
    MQHMSG   Hmsg              = MQHM_NONE;
    MQMD     md                = {MQMD_DEFAULT};
    MQHOBJ   gHobj             = MQHO_NONE;
    MQOD     getOD             = {MQOD_DEFAULT};
    MQGMO    gmo               = {MQGMO_DEFAULT};
    MQLONG   GO_Options        = MQOO_INPUT_AS_Q_DEF
                               | MQOO_FAIL_IF_QUIESCING
                               | MQOO_SAVE_ALL_CONTEXT;
    MQLONG   GC_Options        = MQCO_DELETE_PURGE;
    MQHOBJ   Hsub              = MQHO_NONE;
    MQSD     sd                = {MQSD_DEFAULT};
    MQLONG   SC_Options        = MQCO_NONE;
    MQHOBJ   pHobj             = MQHO_NONE;
    MQOD     putOD             = {MQOD_DEFAULT};
    MQLONG   PO_Options        = MQOO_OUTPUT
                               | MQOO_FAIL_IF_QUIESCING
                               | MQOO_PASS_ALL_CONTEXT;
    MQLONG   PC_Options        = MQCO_NONE;
    MQPMO    pmo               = {MQPMO_DEFAULT};
    MQIMPO   InqPropOpts       = {MQIMPO_DEFAULT};
    MQPD     PropDesc          = {MQPD_DEFAULT};
    MQLONG   Type              = MQTYPE_AS_SET;
    MQCHARV  TopStrProp        = {MQTOPICSTRING};
    MQCHARV  PubOptProp        = {MQPUBOPTIONS};
    MQLONG   DataLength        = 0;
    MQBYTE   buffer[256]       = "";
    MQLONG   buflen            = sizeof(buffer) - 1;
    MQLONG   messlen           = 0;
    char     TopStrBuf[256]    = "Initial value";
    int      i                 = 0;
```

圖 *88:* 宣告

在宣告中不易執行的起始設定會顯示在 第 708 頁的圖 89 中。 強調顯示的值需要說明。

SYSTEM.NDURABLE.MODEL.QUEUE
在此範例中，使用模型佇列 SYSTEM.NDURABLE.MODEL.QUEUE 來建立暫時動態佇列，而不是使用 MQSUB 來開啟受管理不可延續訂閱。 其控點會傳遞至 MQSUB。 透過直接開啟佇列，您可以儲存所有訊息環境定義並設定訂閱選項 MQSO_SET_CORREL_ID。

MQGMO_CURRENT_VERSION
請務必使用大部分 IBM MQ 結構的現行版本。 gmo.MsgHandle 之類的欄位僅在最新版本的控制結構中可用。

MQGMO_PROPERTIES_IN_HANDLE
在原始發佈中設定的主題字串和放置訊息選項，由截取訂閱者利用訊息內容來擷取。 替代方案是直接讀取訊息中的 **MQRFH2** 結構。

MQSO_SET_CORREL_ID
將 MQSO_SET_CORREL_ID 與下列組合使用:

```
memcpy(sd.SubCorrelId, MQCI_NONE, sizeof(sd.SubCorrelId));
```

這些選項的效果是傳遞相關性 ID。 由原始發佈者設定的相關性識別符號被放置在由攔截使用者接收的出版物的相關性識別符號欄位中。 每一個截取訂閱者在相同的相關性 ID 上傳遞。 然後，最終使用者可以選擇接收相同的相關性 ID。

註: 如果透過發佈/訂閱階層傳遞發佈，則永不保留相關性 ID。

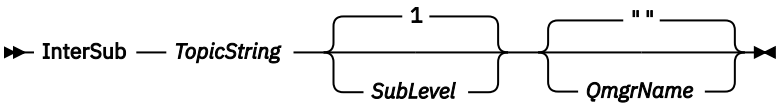
MQPRI_PRIORITY_AS_PUBLISHED
發佈資訊會以發佈資訊的相同訊息優先順序放置在發佈資訊佇列上。

```
strncpy(getOD.ObjectName,    "SYSTEM.NDURABLE.MODEL.QUEUE",
        sizeof(getOD.ObjectName));
gmo.Version                = MQGMO_VERSION_4;
gmo.Options                = MQGMO_WAIT
                           | MQGMO_PROPERTIES_IN_HANDLE
                           | MQGMO_CONVERT;
gmo.WaitInterval           = 30000;
sd.Options                 = MQSO_CREATE
                           | MQSO_FAIL_IF_QUIESCING
                           | MQSO_SET_CORREL_ID;
sd.PubPriority             = MQPRI_PRIORITY_AS_PUBLISHED;
sd.Version                 = MQSD_VERSION_1;
memcpy(sd.SubCorrelId, MQCI_NONE, sizeof(sd.SubCorrelId));
putOD.ObjectType             = MQOT_TOPIC;
putOD.ObjectString.VSPtr     = &TopStrBuf;
putOD.ObjectString.VSBufSize = sizeof(TopStrBuf);
putOD.ObjectString.VSLength  = MQVS_NULL_TERMINATED;
putOD.ObjectString.VSCCSID   = MQCCSI_APPL;
putOD.Version                = MQOD_VERSION_4;
pmo.Version                = MQPMO_VERSION_3;
```

圖 *89:* 起始設定

第 709 頁的圖 90 顯示程式碼片段，以讀取指令行參數、完成起始設定，以及建立截取訂閱。

使用指令執行程式，



為了使錯誤處理盡可能不明顯，每一個 MQI 呼叫的原因碼會儲存在不同的陣列元素中。 在每次呼叫之後，都會測試完成碼，如果值為 MQCC_FAIL，則控制會結束 `do { } while(0)` 程式碼區塊。

這兩行值得注意的程式碼是:

pmo.PubLevel = sd.SubLevel - 1;
將重新發佈訊息的發佈層次設為比截取訂閱者的訂閱層次少 1。

gmo.MsgHandle = Hmsg;
提供訊息控點，讓 MQGET 傳回訊息內容。

```
do {
   printf("Intercepting subscriber start\n");
   if (argc < 2) {
       printf("Required parameter missing - topic string\n");
       exit(99);
   } else {
       sd.ObjectString.VSPtr    = argv[1];
       sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
       printf("TopicString = %s\n", sd.ObjectString.VSPtr);
   }
   if (argc > 2) {
       sd.SubLevel  = atoi(argv[2]);
       pmo.PubLevel = sd.SubLevel - 1;
       printf("SubLevel is %d, PubLevel is %d\n", sd.SubLevel, pmo.PubLevel);
   }
   if (argc > 3)
       strncpy(QMName, argv[3], sizeof(QMName));
   MQCONN(QMName, &Hcon, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQOPEN(Hcon, &getOD, GO_Options, &gHobj, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQSUB(Hcon, &sd, &gHobj, &Hsub, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQCRTMH(Hcon, &CrtMsgHOpts, &Hmsg, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   gmo.MsgHandle = Hmsg;
```

圖 *90:* 準備截取發佈資訊

主要程式碼片段 第 710 頁的圖 91 會從發佈佇列取得訊息。 它會查詢訊息內容，並使用主題字串及原始 **MQPMO** 來重新發佈訊息。 發佈的 **選項** 內容。

在此範例中，不會對發佈執行任何轉換。 重新發佈的發佈資訊主題字串一律符合截取訂閱者訂閱的主題字串。 如果截取訂閱者負責截取傳送至相同發佈佇列的多個訂閱，則可能需要查詢主題字串，以識別符合不同訂閱的發佈。

會強調顯示對 MQINQMP 的呼叫。 主題字串及發佈放置訊息選項內容會直接寫入輸出控制結構。 將 putOD.ObjectString 的 MQCHARV 長度欄位從明確長度變更為以空值結尾的字串的唯一原因是使用 printf 來輸出字串。

```
while (CompCode != MQCC_FAILED) {
   memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
   memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
   md.Encoding       = MQENC_NATIVE;
   md.CodedCharSetId = MQCCSI_Q_MGR;
   printf("MQGET : %d seconds wait time\n", gmo.WaitInterval/1000);
   MQGET(Hcon, gHobj, &md, &gmo, buflen, buffer, &messlen,
       &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   buffer[messlen] = '\0';
   MQINQMP(Hcon, Hmsg,    &InqPropOpts, &TopStrProp, &PropDesc, &Type,
       putOD.ObjectString.VSBufSize, putOD.ObjectString.VSPtr,
       &(putOD.ObjectString.VSLength), &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   memset((void *)((MQLONG)(putOD.ObjectString.VSPtr)
       + putOD.ObjectString.VSLength),'\0',1);
   putOD.ObjectString.VSLength = MQVS_NULL_TERMINATED;
   MQINQMP(Hcon, Hmsg, &InqPropOpts, &PubOptProp, &PropDesc, &Type,
       sizeof(pmo.Options), &(pmo.Options), &DataLength,
       &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQOPEN(Hcon, &putOD, PO_Options, &pHobj, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   printf("Republish message <%s> on topic <%s> with options %d\n",
       buffer, putOD.ObjectString.VSPtr, pmo.Options);
   MQPUT(Hcon, pHobj, &md, &pmo, messlen, buffer, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQCLOSE(Hcon, &pHobj, PC_Options, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
   break;
}
```

圖 *91:* 截取發佈並重新發佈

最終程式碼片段顯示在 第 710 頁的圖 92 中。

```
    } while (0);
    if (CompCode == MQCC_FAILED && Reason != MQRC_NO_MSG_AVAILABLE)
        printf("MQI Call failed with reason code %d\n", Reason);
    if (Hsub!= MQHO_NONE)
        MQCLOSE(Hcon, &Hsub, SC_Options, &CompCode, &Reason);
    if (Hcon!= MQHC_UNUSABLE_HCONN)
        MQDISC(&Hcon, &CompCode, &Reason);
}
```

圖 *92:* 完成

截取發佈及分散式發佈/訂閱

當您將截取訂閱者或發佈結束程式部署至分散式發佈/訂閱拓蹼時，請遵循簡式型樣。 將截取訂閱者部署在與發佈者相同的佇列管理程式上，並將發佈結束程式部署在與最終訂閱者相同的佇列管理程式上。

第 711 頁的圖 93 顯示在發佈訂閱叢集中連接的兩個佇列管理程式。 發佈者會在發佈層次 9 建立叢集主題的發佈資訊。 編號箭頭顯示發佈在流向叢集主題訂閱者時所採取的步驟順序。 訂閱者會截取具有 Sublevel 9 的發佈，並使用 Publevel 8 重新發佈。 訂閱者會在 **子層次** 8 再次截取它。 訂閱者會在 Publevel 7 重新發佈。 佇列管理程式提供的 Proxy 訂閱者會將發佈轉遞至佇列管理程式 B ，其中除了最終訂閱者之外，還部署了「發佈」結束程式。 發佈會先由「發佈」結束程式處理，然後最終由最終訂閱者在 **子層次** 1 接收。 攔截訂閱者和發佈結束程式會顯示中斷的大綱。

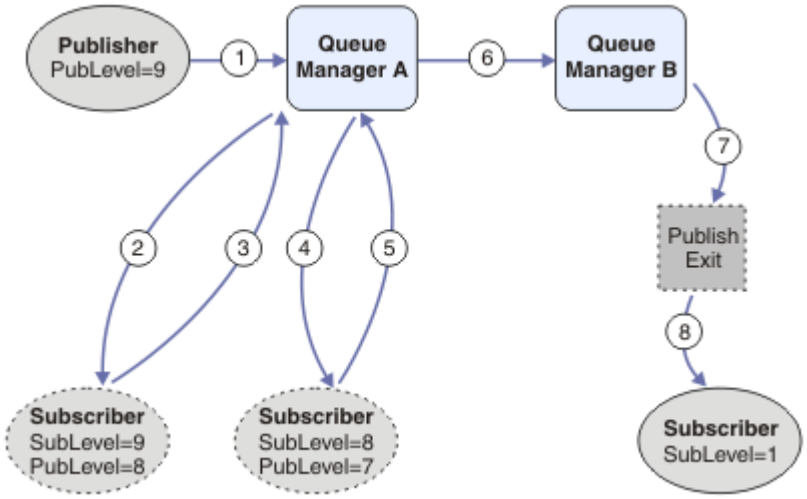


圖 *93:* 叢集中的截取及發佈結束程式

簡式型樣的目標是每個接收發佈資訊的訂閱者接收相同的發佈資訊。 不論訂閱者連接至何處，發佈都會經歷相同的轉換順序。 您可能想要避免轉換順序改變，視發佈者或最終訂閱者的連接位置而定。 合理的例外是自訂最終遞送至每個個別訂閱者的發佈。 使用「發佈」結束程式，根據發佈最終遞送至的佇列來自訂發佈。

您必須仔細考量在分散式發佈/訂閱拓蹼中部署截取訂閱者及「發佈」結束程式的位置。 直接明確型樣會將截取訂閱者部署至與發佈者相同的佇列管理程式，並將「發佈」結束程式部署至與最終訂閱者相同的佇列管理程式。

反型樣

第 711 頁的圖 94 顯示如果您不遵循簡式型樣，則重要事項會如何出錯。 為了使部署複雜化，會將最終訂閱者新增至佇列管理程式 A，並將另外兩個截取訂閱者新增至佇列管理程式 B。

發佈會轉遞至 PubLevel 7 上的佇列管理程式 B，在此由訂閱者在 SubLevel 5 上截取，然後由最終訂閱者在 SubLevel 1 上耗用。 「發佈」結束程式會在發佈傳遞給佇列管理程式 B 中的截取消費者及最終消費者之前，先截取發佈。 發佈會到達佇列管理程式 A 上的最終訂閱者，而不會由「發佈」結束程式處理。

在發佈/訂閱拓蹼中， Proxy 訂閱者會在 SubLevel 1 上訂閱，並傳遞最後一個截取訂閱者所設定的 PubLevel。 在 第 711 頁的圖 94 中，結果是訂閱者不會在佇列管理程式 B 上使用 SubLevel 9 來截取發佈。

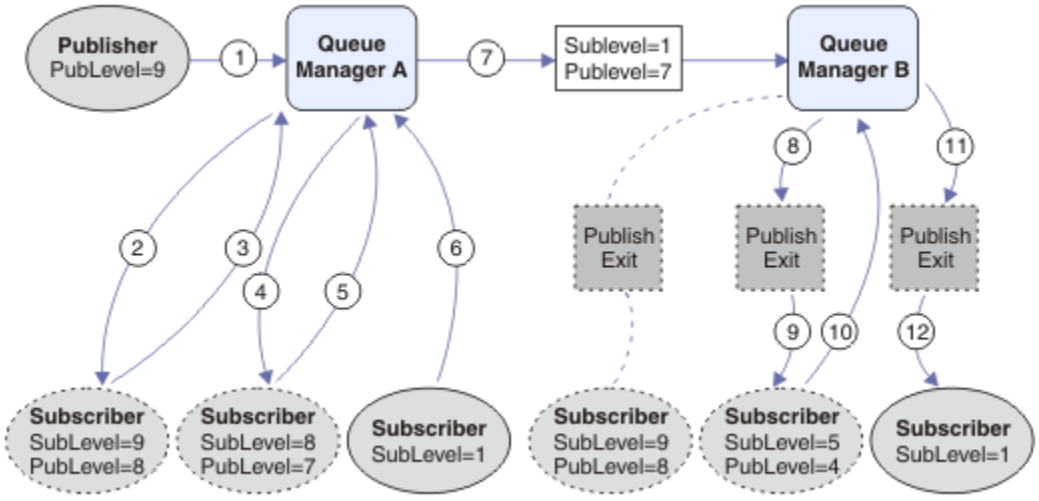


圖 *94:* 攔截使用者的複雜部署

發佈選項
有數個選項可用來控制發佈訊息的方式。

向訂閱者拒絕回覆資訊
如果您不希望訂閱者能夠回覆他們所接收的發佈，則可以使用 MQPMO_SUPPRESS_REPLYTO 放置訊息選項，來保留 MQMD 的 ReplyToQ 及 ReplyTo 佇列管理程式欄位中的資訊。 如果使用此選項，當佇列管理程式在將發佈轉遞至任何訂閱者之前收到發佈時，會從 MQMD 移除該資訊。

此選項無法與需要 ReplyToQ 的報告選項一起使用，如果嘗試呼叫失敗且 MQRC_MISSING_REPLY_TO_Q。

發佈層次
使用發佈層次是控制哪些訂閱者接收發佈的方法。 發佈層次表示發佈所設定的訂閱層次。 只有最高訂閱層次小於或等於發佈的發佈層次的訂閱才會接收發佈。 此值必須在 0 到 9 的範圍內; 零是最低發佈層次。 此欄位的起始值為 9。 發佈和訂閱層次的其中一個用途是 截取發佈。

檢查發佈是否未遞送給任何訂閱者

若要檢查發佈是否尚未遞送至任何訂閱者，請搭配使用 MQPMO_WARN_IF_NO_SUBS_MATCHED put-message 選項與 MQPUT 呼叫。 如果 put 作業傳回完成碼 MQCC_WARNING 及原因碼 MQRC_NO_SUBS_MATCHED ，則不會將發佈遞送至任何訂閱。 如果在 put 作業上指定 MQPMO_RETAIN 選項，則訊息會保留並遞送至任何後續定義的相符訂閱。 在分散式發佈/訂閱系統中，只有在佇列管理程式上未登錄主題的 Proxy 訂閱時，才會傳回 MQRC_NO_SUBS_MATCHED 原因碼。

訂閱選項
有數個選項可用來控制處理訊息訂閱的方式。

訊息持續性

佇列管理程式會依照發佈者所設定，維護它們轉遞給訂閱者之發佈的持續性。 發佈者將持續性設為下列其中一個選項:

0
非持續性

1
持續

2
作為佇列/主題定義的持續性

對於發佈/訂閱，發佈者會將主題物件及 **topicString** 解析為已解析的主題物件。 如果發佈者指定「持續性」作為佇列/主題定義，則會針對發佈設定已解析主題物件中的預設持續性。

保留的發佈

若要控制何時收到保留的發佈，訂閱者可以使用兩個訂閱選項:

僅在要求時發佈 MQSO_PUBLICATIONS_ON_REQUEST

如果您想要訂閱者控制其何時接收發佈，您可以使用 MQSO_PUBLICATIONS_ON_REQUEST 訂閱選項。 然後，訂閱者可以使用 MQSUBRQ 呼叫 (指定從原始 MQSUB 呼叫傳回的 Hsub 控點) 來控制何時接收發佈資訊，以要求將主題的保留發佈資訊傳送給它。 使用 MQSO_PUBLICATIONS_ON_REQUEST 訂閱選項的訂閱者不會收到任何非保留的發佈。

如果您指定 MQSO_PUBLICATIONS_ON_REQUEST ，則必須使用 MQSUBRQ 來擷取任何發佈。 如果您不使用 MQSO_PUBLICATIONS_ON_REQUEST ，則會在發佈訊息時收到訊息。

如果訂閱者使用 MQSUBRQ 呼叫並在訂閱主題中使用萬用字元，則訂閱可能會符合主題樹狀結構上的多個主題或節點，所有具有保留訊息 (如果有的話) 的主題或節點都會傳送至訂閱者。

當與可延續訂閱一起使用時，這個選項特別有用，因為即使該訂閱者應用程式不在執行中，佇列管理程式仍會持續訂閱，繼續將發佈傳送給訂閱者。 這可能導致在訂閱者佇列上建立訊息。 如果訂閱者使用

MQSO_PUBLICATIONS_ON_REQUEST 選項進行登錄，則可以避免此建置。 或者，您可以使用不可延續訂閱 (如果適合您的應用程式的話)，以避免建置不想要的訊息。

如果訂閱可延續且發佈者使用保留的發佈，則在重新啟動之後，訂閱者應用程式可以使用 MQSUBRQ 呼叫來重新整理其狀態資訊。 然後，訂閱者必須使用 MQSUBRQ 呼叫定期重新整理其狀態。

不會因為使用此選項的 MQSUB 呼叫而傳送任何發佈。 如果原始訂閱配置為使用此選項，則在斷線後已回復的可延續訂閱將使用 MQSO_PUBLICATIONS_ON_REQUEST 選項。

僅新發佈 :MQSO_NEW_PUBLICATIONS_ONLY

如果主題上存在保留的發佈，則在發佈之後進行訂閱的任何訂閱者都會收到該發佈的副本。 如果訂閱者不想要接收任何早於所建立訂閱的發佈，則訂閱者可以使用 MQSO_NEW_PUBLICATIONS_ONLY 訂閱選項。

將訂閱分組

如果您已設定佇列來接收發佈資訊，且有許多重疊的訂閱將發佈資訊饋送至相同佇列，請考量將訂閱分組。 此狀況類似於 重疊訂閱中的範例。

當您訂閱主題時，可以透過設定選項 MQSO_GROUP_SUB 來避免接收重複發佈。 結果是當群組中有多個訂閱符合發佈的主題時，只有一個訂閱負責將發佈放置在佇列上。 會忽略符合發佈主題的其他訂閱。

負責將發佈放置在佇列上的訂閱，是在遇到任何萬用字元之前，根據它具有最長相符主題字串來選擇。 可以將它視為最符合的訂閱。 其內容會延伸到發佈，包括它是否具有 MQSO_NOT_OWN_PUBS 內容。 如果如此，則不會將任何發佈遞送至佇列，即使其他相符訂閱可能沒有 MQSO_NOT_OWN_PUBS 內容也一樣。

您無法將所有訂閱放置在單一群組中，以刪除重複的發佈。 分組訂閱必須滿足下列條件:

1. 沒有任何訂閱受管理。
2. 訂閱群組會將發佈遞送至相同的佇列。
3. 每一個訂閱必須處於相同的訂閱層次。
4. 群組中每一個訂閱的發佈訊息具有相同的相關性 ID。

   若要確保每個訂閱都產生具有相同相關性 ID 的發佈訊息，請設定 MQSO_SET_CORREL_ID 以在發佈中建立您自己的相關性 ID，並在每個訂閱的 **SubCorrelId** 欄位中設定相同的值。 請勿將 **SubCorrelId** 設為值 MQCI_NONE。

如需相關資訊，請參閱 ../refdev/q100080_.dita#q100080_/mqso_group_sub 。

查詢及設定物件屬性

屬性是定義 IBM MQ 物件性質的內容。

它們會影響佇列管理程式處理物件的方式。 每一種 IBM MQ 物件類型的屬性在 物件屬性中有詳細說明。

部分屬性會在定義物件時設定，且只能使用 IBM MQ 指令來變更; 這類屬性的範例是放置在佇列中的訊息預設優先順序。 其他屬性會受到佇列管理程式作業的影響，且可能會隨著時間而變更; 例如佇列的現行深度。

您可以使用 MQINQ 呼叫來查詢大部分屬性的現行值。 MQI 也提供 MQSET 呼叫，可讓您變更部分佇列屬性。 您無法使用 MQI 呼叫來變更任何其他類型物件的屬性。 您必須改用下列其中一個資源:

- ALW MQSC 機能，如 MQSC 指令中所說明。
- IBM i CHGMQMx CL 指令 (在 IBM i 的 CL 指令參考手冊或 MQSC 機能中說明)。
- z/OS ALTER 操作員指令，或具有 REPLACE 選項的 DEFINE 指令，如 MQSC 指令中所說明。

註: 物件屬性的名稱會以您搭配 MQINQ 及 MQSET 呼叫使用的格式顯示在本文件中。 當您使用 IBM MQ 指令來定義、變更或顯示屬性時，必須使用主題鏈結中指令說明所顯示的關鍵字來識別屬性。

MQINQ 和 MQSET 呼叫都使用選取元陣列來識別 您想要查詢或設定的那些屬性。 您可以使用的每一個屬性都有一個選取元。 選取元名稱具有由屬性本質所決定的字首:

表 *127:* 選取元名稱的字首	
字首	**說明**
MQCA_	這些選取器參照包含字元資料 (例如，佇列名稱) 的屬性。
MQIA_	這些選取元是指包含數值 (例如 *CurrentQueueDepth*，佇列上的訊息數) 或常數值 (例如 *SyncPoint*，佇列管理程式是否支援同步點) 的屬性。

在使用 MQINQ 或 MQSET 呼叫之前，您的應用程式必須連接至佇列管理程式，且您必須使用 MQOPEN 呼叫來開啟物件，以設定或查詢屬性。 這些作業在 第 616 頁的『連接至佇列管理程式並切斷與佇列管理程式的連線』 和 第 621 頁的『開啟及關閉物件』中有說明。

請使用下列鏈結，以進一步瞭解查詢及設定物件屬性的相關資訊:

- 第 714 頁的『查詢物件的屬性』
- 第 715 頁的『MQINQ 呼叫失敗的部分情況』
- 第 715 頁的『設定佇列屬性』

相關概念

第 604 頁的『訊息佇列介面概觀』
瞭解「訊息佇列介面 (MQI)」元件。

第 616 頁的『連接至佇列管理程式並切斷與佇列管理程式的連線』
若要使用 IBM MQ 程式設計服務，程式必須具有與佇列管理程式的連線。 使用此資訊來瞭解如何連接至佇列管理程式，以及與佇列管理程式中斷連線。

第 621 頁的『開啟及關閉物件』
此資訊提供開啟及關閉 IBM MQ 物件的見解。

第 630 頁的『將訊息放置在佇列上』
使用此資訊來瞭解如何將訊息放置在佇列上。

第 643 頁的『從佇列取得訊息』
使用此資訊來瞭解從佇列取得訊息的相關資訊。

第 716 頁的『確定及取消工作單元』
此資訊說明如何確定及取消工作單元中發生的任何可回復取得及放置作業。

第 725 頁的『使用觸發程式啟動 IBM MQ 應用程式』
瞭解觸發程式以及如何使用觸發程式來啟動 IBM MQ 應用程式。

第 741 頁的『使用 MQI 及叢集』
對於與叢集作業相關的呼叫和回覆碼，有一些特殊選項。

第 745 頁的『在 IBM MQ for z/OS 上使用及撰寫應用程式』
IBM MQ for z/OS 應用程式可以由在許多不同環境中執行的程式組成。 這表示他們可以利用多個環境中可用的設施。

第 57 頁的『IBM MQ for z/OS 上的 IMS 及 IMS 橋接器應用程式』
此資訊可協助您使用 IBM MQ 撰寫 IMS 應用程式。

查詢物件的屬性

使用 MQINQ 呼叫來查詢任何類型 IBM MQ 的屬性。

作為此呼叫的輸入，您必須提供:

- 連線控點。
- 物件控點。
- 選取元數目。

- 屬性選取元的陣列，每一個選取元的格式都是 MQCA_ * 或 MQIA_ *。 每一個選取元代表一個具有您要查詢之值的屬性，且每一個選取元必須對物件控點所代表的物件類型有效。 您可以按任何順序指定選取元。
- 您正在查詢的整數屬性數目。 如果您不查詢整數屬性，請指定零。
- *CharAttrLength* 中字元屬性緩衝區的長度。 這至少必須是保留每一個字元屬性字串所需之長度的總和。 如果您不查詢字元屬性，請指定零。

MQINQ 的輸出為:

- 複製到陣列中的一組整數屬性值。 值數目由 *IntAttrCount* 決定。 如果 *IntAttrCount* 或 *SelectorCount* 是零，則不會使用此參數。
- 在其中傳回字元屬性的緩衝區。 緩衝區的長度由 **CharAttrLength** 參數提供。 如果 *CharAttrLength* 或 *SelectorCount* 是零，則不會使用此參數。
- 完成碼。 如果完成碼發出警告，這表示呼叫只局部完成。 在此情況下，請檢查原因碼。
- 原因碼。 有三種局部完成狀況:
  - 選取元不適用於佇列類型
  - 整數屬性所容許的空間不足
  - 字元屬性沒有足夠的空間

  如果出現其中多個狀況，則會傳回第一個適用的狀況。

如果您開啟佇列以進行輸出或查詢，且它解析為非本端叢集佇列，則只能查詢佇列名稱、佇列類型及共同屬性。 如果使用 MQOO_BIND_ON_OPEN，則共同屬性的值是所選佇列的值。 如果使用 MQOO_BIND_NOT_FIXED 或 MQOO_BIND_ON_GROUP 或使用 MQOO_BIND_AS_Q_DEF 且 **DefBind** 佇列屬性為 MQBND_BIND_NOT_FIXED，則這些值是任意其中一個可能叢集佇列的值。 如需相關資訊，請參閱第 742 頁的『MQOPEN 及叢集』 及 MQOPEN。

註: 呼叫傳回的值是所選取屬性的 Snapshot。 在您的程式對回覆值採取動作之前，可以變更屬性。

MQINQ 中有 MQINQ 呼叫的說明。

MQINQ 呼叫失敗的部分情況

如果您開啟別名來查詢其屬性，則會傳回別名佇列 (用來存取另一個佇列的 IBM MQ 物件) 的屬性，而不是基本佇列的屬性。

不過，佇列管理程式也會開啟別名所解析成的基本佇列定義，如果另一個程式在 MQOPEN 與 MQINQ 呼叫之間的間隔內變更基本佇列的用法，則 MQINQ 呼叫會失敗並傳回 MQRC_OBJECT_CHANGED 原因碼。 如果別名佇列物件的屬性變更，則呼叫也會失敗。

同樣地，當您開啟遠端佇列來查詢其屬性時，只會傳回遠端佇列本端定義的屬性。

如果您指定一或多個不適用於您要查詢之佇列屬性類型的選取器， MQINQ 呼叫會完成並發出警告，並如下設定輸出:

- 對於整數屬性， *IntAttrs* 的對應元素會設為 MQIAV_NOT_APPLICABLE。
- 對於字元屬性， *CharAttrs* 字串的對應部分會設為星號。

如果您指定一或多個對您要查詢的物件屬性類型無效的選取元，則 MQINQ 呼叫會失敗，並傳回 MQRC_SELECTOR_ERROR 原因碼。

您無法呼叫 MQINQ 來查看模型佇列; 請使用 MQSC 機能或平台上可用的指令。

設定佇列屬性

使用此資訊來瞭解如何使用 MQSET 呼叫來設定佇列屬性。

您只能使用 MQSET 呼叫來設定下列佇列屬性:

- *InhibitGet* (但不適用於遠端佇列)
- *DistList* (不在 z/OS 上)
- *InhibitPut*
- *TriggerControl*

- *TriggerType*
- *TriggerDepth*
- *TriggerMsgPriority*
- *TriggerData*

MQSET 呼叫具有與 MQINQ 呼叫相同的參數。 不過，對於 MQSET ，完成碼和原因碼以外的所有參數都是輸入參數。 沒有局部完成狀況。

註: 您無法使用 MQI 來設定非本端定義佇列的 IBM MQ 物件屬性。

如需 MQSET 呼叫的詳細資料，請參閱 MQSET。

確定及取消工作單元

此資訊說明如何確定及取消工作單元中發生的任何可回復取得及放置作業。

本主題中使用下列術語:

- 確定
- 取消
- 同步點協調
- 同步點
- 工作單元
- 一段式確定 (single-phase commit)
- 兩階段確定

如果您熟悉這些交易處理術語，則可以跳至 第 717 頁的『IBM MQ 應用程式中的同步點考量』。

確定並取消
當程式將訊息放入工作單元內的佇列時，只有在程式確定工作單元時，其他程式才會看到該訊息。 若要確定工作單元，必須順利完成所有更新，以保留資料完整性。 如果程式偵測到錯誤並判定放置作業不是永久的，則可以退出工作單元。 當程式執行取消時， IBM MQ 會移除該工作單元放置在佇列上的訊息，以還原佇列。 程式執行確定及取消作業的方式取決於程式執行所在的環境。

同樣地，當程式從工作單元內的佇列取得訊息時，該訊息會保留在佇列上，直到程式確定工作單元為止，但其他程式無法擷取該訊息。 當程式確定工作單元時，會從佇列永久刪除訊息。 如果程式取消工作單元， IBM MQ 會使訊息可供其他程式擷取，以還原佇列。

同步點協調、同步點、工作單元
同步點協調 是透過資料完整性來確定或取消工作單元的程序。

在最簡單的情況下，會在交易結束時做出確定或取消變更的決策。 不過，應用程式在交易內的其他邏輯點上同步化資料變更會更有用。 這些邏輯點稱為 同步點 (或 同步點) 在兩個同步點之間處理一組更新項目的期間稱為 工作單元。 數個 MQGET 呼叫和 MQPUT 呼叫可以是單一工作單元的一部分。

ALTER QMGR 指令的 MAXUMSGS 屬性可以控制工作單元內的訊息數上限。

一段式確定 (single-phase commit)
單一階段確定 處理程序可讓程式將更新項目確定至佇列，而不需與其他資源管理程式協調其變更。

兩階段確定
兩段式確定 程序是指程式對 IBM MQ 佇列所做的更新，可以與其他資源 (例如，受 Db2 控制的資料庫) 的更新協調。 在這類程序下，所有資源的更新會一起確定或取消。

為了協助處理工作單元， IBM MQ 提供 **BackoutCount** 屬性。 每次取消工作單元內的訊息時，即會增加此值。 如果訊息反覆地導致工作單元異常結束，則 *BackoutCount* 最終的值會超出 *BackoutThreshold* 的值。 當定義佇列時，會設定此值。 在此狀況下，應用程式可以從工作單元中移除訊息，並將它放入另一個佇列，如 *BackoutRequeueQName* 中所定義。 當移動訊息時，工作單元可以確定。

使用下列鏈結，以進一步瞭解確定及取消工作單元的相關資訊:

- 第 717 頁的『IBM MQ 應用程式中的同步點考量』

- z/OS 第 718 頁的『IBM MQ for z/OS 應用程式中的同步點』
- IBM i 第 720 頁的『CICS for IBM i 應用程式中的同步點』
- 第 720 頁的『IBM MQ for Multiplatforms 中的同步點』
- IBM i 第 724 頁的『IBM i 外部同步點管理程式的介面』

相關概念

第 604 頁的『訊息佇列介面概觀』
瞭解「訊息佇列介面 (MQI)」元件。

第 616 頁的『連接至佇列管理程式並切斷與佇列管理程式的連線』
若要使用 IBM MQ 程式設計服務，程式必須具有與佇列管理程式的連線。 使用此資訊來瞭解如何連接至佇列管理程式，以及與佇列管理程式中斷連線。

第 621 頁的『開啟及關閉物件』
此資訊提供開啟及關閉 IBM MQ 物件的見解。

第 630 頁的『將訊息放置在佇列上』
使用此資訊來瞭解如何將訊息放置在佇列上。

第 643 頁的『從佇列取得訊息』
使用此資訊來瞭解從佇列取得訊息的相關資訊。

第 713 頁的『查詢及設定物件屬性』
屬性是定義 IBM MQ 物件性質的內容。

第 725 頁的『使用觸發程式啟動 IBM MQ 應用程式』
瞭解觸發程式以及如何使用觸發程式來啟動 IBM MQ 應用程式。

第 741 頁的『使用 MQI 及叢集』
對於與叢集作業相關的呼叫和回覆碼，有一些特殊選項。

第 745 頁的『在 IBM MQ for z/OS 上使用及撰寫應用程式』
IBM MQ for z/OS 應用程式可以由在許多不同環境中執行的程式組成。 這表示他們可以利用多個環境中可用的設施。

第 57 頁的『IBM MQ for z/OS 上的 IMS 及 IMS 橋接器應用程式』
此資訊可協助您使用 IBM MQ 撰寫 IMS 應用程式。

IBM MQ 應用程式中的同步點考量

使用此資訊來瞭解在 IBM MQ 應用程式中使用同步點的相關資訊。

下列環境支援兩階段確定:

- Multi IBM MQ for Multiplatforms
- z/OS CICS Transaction Server for z/OS
- z/OS TXSeries
- z/OS IMS/ESA
- z/OS z/OS 批次 (含 RRS)
- 使用 X/Open XA 介面的其他外部協調程式

下列環境支援單階段確定:

- Multi IBM MQ for Multiplatforms
- z/OS z/OS 批次

如需外部介面的進一步相關資訊，請參閱 第 722 頁的『Multiplatforms 上外部同步點管理程式的介面』及 XA 文件 *CAE Specification Distributed Transaction Processing: The XA Specification*(由 The Open Group 發

佈)。 交易管理程式 (例如 CICS、 IMS、Encina 和 Tuxedo) 可以參與兩段式確定，並與其他可回復資源協調。 這表示 IBM MQ 所提供的佇列作業功能可以放在交易管理程式所管理的工作單元範圍內。

IBM MQ 隨附的範例顯示 IBM MQ 協調符合 XA 標準的資料庫。 如需這些範例的進一步相關資訊，請參閱 第 887 頁的『使用 IBM MQ 範例程序化程式』。

在 IBM MQ 應用程式中，您可以在每個 put 及 get 呼叫上指定是否要讓該呼叫受到同步點控制。 若要讓放置作業在同步點控制下運作，當您呼叫 MQPUT 時，請在 MQPMO 結構的 *Options* 欄位中使用 MQPMO_SYNCPOINT 值。 對於 get 作業，請在 MQGMO 結構的 *Options* 欄位中使用 MQGMO_SYNCPOINT 值。 如果您未明確選擇選項，則預設動作視平台而定:

- Multi 同步點控制預設值為 NO。
- z/OS 同步點控制項預設值為 YES。

使用 MQPMO_SYNCPOINT 發出 MQPUT1 呼叫時，預設行為會變更，以便以非同步方式完成放置作業。 這可能會導致部分應用程式的行為發生變更，這些應用程式依賴所傳回 MQOD 及 MQMD 結構中的特定欄位，但現在包含未定義的值。 應用程式可以指定 MQPMO_SYNC_RESPONSE，以確保同步執行放置作業，並完成所有適當的欄位值。

當您的應用程式收到 MQRC_BACKED_OUT 原因碼以回應同步點下的 MQPUT 或 MQGET 時，應用程式通常應該使用 MQBACK 來取消現行交易，然後在適當時重試整個交易。 如果應用程式在回應 MQCMIT 或 MQDISC 呼叫時收到 MQRC_BACKED_OUT，則不需要呼叫 MQBACK。

每次取消 MQGET 呼叫時，受影響訊息之 MQMD 結構的 *BackoutCount* 欄位都會增加。 高 *BackoutCount* 指出已反覆取消的訊息。 這可能表示此訊息有問題，您應該加以調查。 如需 *BackoutCount* 的詳細資料，請參閱 BackoutCount。

除了具有 RRS 的 z/OS 批次之外，如果程式在有未確定的要求時發出 MQDISC 呼叫，則會發生隱含的同步點。 如果程式異常結束，則會發生隱含的取消。

z/OS 在 z/OS 上，如果程式正常結束而未先呼叫 MQDISC，也會發生隱含同步點。 如果連接至 MQ 的 TCB 正常結束，則會將程式視為正常結束。 在 z/OS UNIX System Services 及 Language Environment (LE) 下執行時，會對異常終止或信號呼叫預設條件處理。 LE 條件處理程式會處理錯誤條件，且 TCB 會正常結束。 在這些狀況下， MQ 會確定工作單元。 如需相關資訊，請參閱 Language Environment Condition Handling 簡介。

z/OS 對於 IBM MQ for z/OS 程式，您可以使用 MQGMO_MARK_SKIP_BACKOUT 選項來指定在發生取消時不得取消訊息 (以避免 *MQGET-error-backout* 迴圈)。 如需使用此選項的相關資訊，請參閱 第 668 頁的『跳過取消』。

工作單元的確定或取消不會影響佇列屬性的變更 (由 MQSET 呼叫或指令所做)。

z/OS *IBM MQ for z/OS* 應用程式中的同步點

本主題說明如何在交易管理程式 (CICS 和 IMS) 中使用同步點 及批次應用程式。

z/OS *CICS Transaction Server for z/OS* 應用程式中的同步點

在 CICS 應用程式中，您可以使用 EXEC CICS SYNCPOINT 指令來建立同步點。

如果要回復前一個同步點的所有變更，您可以使用 EXEC CICS SYNCPOINT ROLLBACK 指令。 如需相關資訊，請參閱 *CICS Application Programming Reference*。

如果工作單元中涉及其他可回復的資源，則佇列管理程式 (與 CICS 同步點管理程式一起使用) 會參與兩階段確定通訊協定; 否則，佇列管理程式會執行單階段確定處理程序。

如果 CICS 應用程式發出 MQDISC 呼叫，則不會採用隱含的同步點。 如果應用程式正常關閉，則會關閉任何開啟的佇列，並進行隱含的確定。 如果應用程式異常關閉，則會關閉任何開啟的佇列，並發生隱含的取消。

z/OS *IMS* 應用程式中的同步點

在 IMS 應用程式中，使用 IMS 呼叫 (例如對 IOPCB 及 CHKP (檢查點) 的 GU (取得唯一)) 來建立同步點。

若要回復自前一個檢查點以來的所有變更，您可以使用 IMS ROLB (回復) 呼叫。 如需相關資訊，請參閱 IMS 文件。

如果工作單元中也包含其他可回復的資源，則佇列管理程式 (與 IMS 同步點管理程式一起使用) 會參與兩段式確定通訊協定。

IMS 配接器會在同步點關閉所有開啟控點 (批次或非訊息驅動 BMP 環境除外)。 這是因為不同的使用者可以起始下一個工作單元，而 IBM MQ 安全檢查是在發出 MQCONN、MQCONNX 及 MQOPEN 呼叫時執行，而不是在發出 MQPUT 或 MQGET 呼叫時執行。

不過，在「等待輸入 (WFI)」或虛擬「等待輸入 (PWFI)」環境中， IMS 不會通知 IBM MQ 關閉控點，直到下一個訊息到達或 QC 狀態碼傳回應用程式為止。 如果應用程式正在 IMS 區域中等待，且任何這些控點都屬於已觸發的佇列，則不會發生觸發，因為佇列已開啟。 基於此原因，在 WFI 或 PWFI 環境中執行的應用程式應該在對 IOPCB 執行 GU 以取得下一則訊息之前，明確地 MQCLOSE 佇列控點。

如果 IMS 應用程式 (BMP 或 MPP) 發出 MQDISC 呼叫，則會關閉開啟佇列，但不會採用隱含的同步點。 如果應用程式正常關閉，則會關閉任何開啟的佇列，並進行隱含的確定。 如果應用程式異常關閉，則會關閉任何開啟的佇列，並發生隱含的取消。

z/OS *z/OS* 批次應用程式中的同步點

對於批次應用程式，您可以使用 IBM MQ 同步點管理呼叫 :MQCMIT 和 MQBACK。 為了與舊版相容，CSQBCMT 和 CSQBBAK 可作為同義字使用。

註: 如果您需要在單一工作單元內，對不同資源管理程式 (例如 IBM MQ 和 Db2) 所管理的資源確定或取消更新，您可以使用 RRS。 如需進一步資訊，請參閱 第 719 頁的『交易管理和可回復的資源管理程式服務』。

使用 MQCMIT 呼叫來確定變更

作為輸入，您必須提供 MQCONN 或 MQCONNX 呼叫所傳回的連線控點 (*Hconn*)。

MQCMIT 的輸出是完成碼和原因碼。 如果同步點已完成，但佇列管理程式在前一個同步點之後取消了放置及取得作業，則呼叫會完成並產生警告。

順利完成 MQCMIT 呼叫會向佇列管理程式指出應用程式已達到同步點，且自前一個同步點以來所進行的所有放置及取得作業都已變成永久。

並非所有失敗回應都表示 MQCMIT 未完成。 例如，應用程式可以接收 MQRC_CONNECTION_BROKEN。

MQCMIT 中有 MQCMIT 呼叫的說明。

使用 MQBACK 呼叫來取消變更

作為輸入，您必須提供連線控點 (*Hconn*)。 使用 MQCONN 或 MQCONNX 呼叫傳回的控點。

MQBACK 的輸出是完成碼和原因碼。

輸出會向佇列管理程式指出應用程式已達到同步點，且自前次同步點以來所做的所有取得及放置都已取消。

MQBACK 中有 MQBACK 呼叫的說明。

交易管理和可回復的資源管理程式服務

交易管理和可回復資源管理程式服務 (RRS) 是一種 z/OS 機能，可在參與的資源管理程式之間提供兩段式同步點支援。

應用程式可以更新各種 z/OS 資源管理程式 (例如 IBM MQ 和 Db2) 所管理的可回復資源，然後以單一工作單元來確定或取消這些更新。 RRS 在正常執行期間提供必要的工作單元狀態記載，協調同步點處理程序，並在子系統重新啟動期間提供適當的工作單元狀態資訊。

IBM MQ for z/OS RRS 參與者支援可讓批次、TSO 和 Db2 儲存程序環境中的 IBM MQ 應用程式同時更新 IBM MQ 和非 IBM MQ 資源 (例如 Db2) 在單一邏輯工作單元內。 如需 RRS 參與者支援的相關資訊，請參閱 *z/OS MVS Programming: Resource Recovery*。

您的 IBM MQ 應用程式可以使用 MQCMIT 和 MQBACK 或對等的 RRS 呼叫， SRRCMIT 和 SRRBACK。 如需相關資訊，請參閱 第 747 頁的『RRS 批次配接卡』。

RRS 可用性

如果 RRS 在 z/OS 系統上非作用中，則從與 RRS Stub (CSQBRSTB 或 CSQBRRSI) 鏈結的程式發出的任何 IBM MQ 呼叫都會傳回 MQRC_ENVIRONMENT_ERROR。

Db2 儲存程序

如果您將 Db2 儲存程序與 RRS 搭配使用，請注意下列事項:

- 使用 RRS 的 Db2 儲存程序必須由工作量管理程式 (WLM 管理) 管理。
- 如果 Db2 管理的儲存程序包含 IBM MQ 呼叫，且它與 RRS Stub (CSQBRSTB 或 CSQBRRSI) 相鏈結，則 MQCONN 或 MQCONNX 呼叫會傳回 MQRC_ENVIRONMENT_ERROR。
- 如果 WLM 管理的儲存程序包含 IBM MQ 呼叫，且與非 RRS Stub 鏈結，則 MQCONN 或 MQCONNX 呼叫會傳回 MQRC_ENVIRONMENT_ERROR ，除非它是自儲存程序位址空間啟動以來所執行的第一個 IBM MQ 呼叫。
- 如果 Db2 儲存程序包含 IBM MQ 呼叫並與非 RRS Stub 鏈結，則在儲存程序位址空間結束之前，或在後續儲存程序執行 MQCMIT (使用 IBM MQ Batch/TSO Stub) 之前，不會確定該儲存程序中更新的 IBM MQ 資源。
- 相同儲存程序的多個副本可以在相同位址空間中同時執行。 如果您想要 Db2 使用儲存程序的單一副本，請確保以重新進入的方式撰寫程式。 否則，您可能會在程式中的任何 IBM MQ 呼叫上收到 MQRC_HCONN_ERROR。
- 請勿在 WLM 管理的 Db2 儲存程序中撰寫 MQCMIT 或 MQBACK 的程式碼。
- 設計要在 Language Environment (LE) 中執行的所有程式。

IBM i *CICS for IBM i 應用程式中的同步點*

IBM MQ for IBM i 會參與 CICS for IBM i 工作單元。 您可以使用 CICS for IBM i 應用程式內的 MQI 在現行工作單元內放置及取得訊息。

您可以使用 EXEC CICS SYNCPOINT 指令來建立包含 IBM MQ for IBM i 作業的同步點。 若要回復直到前一個同步點的所有變更，您可以使用 EXEC CICS SYNCPOINT ROLLBACK 指令。

如果您在 CICS for IBM i 應用程式中使用 MQPUT、 MQPUT1 或 MQGET 與 MQPMO_SYNCPOINT 或 MQGMO_SYNCPOINT 選項集，則在 IBM MQ for IBM i 移除其作為 API 確定資源的登錄之前，您無法登出 CICS for IBM i 。 在切斷佇列管理程式的連線之前，請先確定或取消任何擱置中的放置或取得作業。 這可讓您登出 CICS for IBM i。

Multi *IBM MQ for Multiplatforms 中的同步點*

同步點支援在兩種類型的工作單元上運作: 區域和廣域。

本端 工作單元是其中唯一更新的資源是 IBM MQ 佇列管理程式的資源。 在這裡，由佇列管理程式本身使用一段式確定程序來提供同步點協調。

廣域 工作單元是也會更新屬於其他資源管理程式 (例如資料庫) 之資源的工作單元。 IBM MQ 可以自行協調這類工作單元。 它們也可以由外部確定控制器來協調。 例如:

- 另一個交易管理程式
- IBM i IBM i 確定控制器

如需完整完整性，請使用兩階段確定程序。 符合 XA 標準的交易管理程式和資料庫可以提供兩階段確定。 例如:

- TXSeries
- UDB
- IBM i IBM i 確定控制器

ALW IBM MQ 產品可以使用兩階段確定處理程序來協調廣域工作單元。

IBM i IBM MQ for IBM i 可以作為 WebSphere Application Server 環境內廣域工作單元的資源管理程式，但無法作為交易管理程式。

隱含的同步點

放置持續訊息時，會最佳化 IBM MQ ，以將持續訊息置於同步點之下。 如果多個應用程式使用同步點，則將持續訊息放置在相同佇列中的多個應用程式效能會更好。 這是因為如果使用同步點來放置持續訊息，則佇列的競用會較少。

當應用程式將持續訊息置於同步點之外時， **ImplSyncOpenOutput** 會新增隱含的同步點。 這可改善效能，而不會讓應用程式知道隱含的同步點。

當有多個應用程式放入佇列時，隱含的同步點只會提高效能，因為它會減少對佇列的競用。 因此，**ImplSyncOpenOutput** 指定在新增隱含同步點之前，開啟佇列以供輸出的應用程式數目下限。 預設值為 2。 這表示如果您未指定 **ImplSyncOpenOutput**，則只有在多個應用程式放入佇列時，才會新增隱含的同步點。

如需相關資訊，請參閱 調整參數 。

Multiplatforms 上的本端工作單元
只涉及佇列管理程式的工作單元稱為 本端 工作單元。 同步點協調由佇列管理程式本身 (內部協調) 使用一段式確定處理程序提供。

為了啟動本端工作單元，應用程式會指定適當的同步點選項來發出 MQGET、MQPUT 或 MQPUT1 要求。 工作單元是使用 MQCMIT 來確定或使用 MQBACK 回復。 不過，當應用程式與佇列管理程式之間的連線有意或無意中斷時，工作單元也會結束。

當 IBM MQ 所協調的廣域工作單元仍在作用中時，如果應用程式中斷與佇列管理程式的連線 (MQDISC) ，則會嘗試確定工作單元。 不過，如果應用程式在未中斷連線的情況下終止，則會回復工作單元，因為應用程式會被視為異常終止。

Multiplatforms 上的廣域工作單元
當您也需要包含屬於其他資源管理程式之資源的更新項目時，請使用廣域工作單元。

在這裡，協調可以是佇列管理程式的內部或外部:

內部同步點協調

IBM MQ for IBM i 或 IBM MQ for z/OS 不支援廣域工作單元的佇列管理程式協調。 在 IBM MQ MQI client 環境中不支援。

在這裡， IBM MQ 執行協調。 若要啟動廣域工作單元，應用程式會發出 MQBEGIN 呼叫。

作為 MQBEGIN 呼叫的輸入，您必須提供 MQCONN 或 MQCONNX 呼叫傳回的連線控點 (*Hconn*)。 此控點代表與 IBM MQ 佇列管理程式的連線。

應用程式會指定適當的同步點選項來發出 MQGET、MQPUT 或 MQPUT1 要求。 這表示您可以使用 MQBEGIN 來起始廣域工作單元，以更新本端資源及/或屬於其他資源管理程式的資源。 對屬於其他資源管理程式的資源所做的更新，是使用該資源管理程式的 API 來進行。 不過，您無法使用 MQI 來更新屬於其他佇列管理程式的佇列。 在啟動進一步工作單元 (區域或廣域) 之前，請先發出 MQCMIT 或 MQBACK。

使用 MQCMIT 來確定廣域工作單元; 這會對工作單元中涉及的所有資源管理程式起始兩段式確定。 使用兩階段確定處理程序，首先會要求資源管理程式 (例如，符合 XA 標準的資料庫管理程式，例如 Db2、 Oracle 及 Sybase) 準備確定。 只有在所有已準備好的情況下，才會要求他們確定。 如果有任何資源管理程式發出無法確定的信號，則會要求每一個都取消。 或者，您可以使用 MQBACK 來回復所有資源管理程式的更新項目。

當廣域工作單元仍在作用中時，如果應用程式中斷連接 (MQDISC) ，則會確定工作單元。 不過，如果應用程式在未中斷連線的情況下終止，則會回復工作單元，因為應用程式會被視為異常終止。

MQBEGIN 的輸出是完成碼和原因碼。

當您使用 MQBEGIN 來啟動廣域工作單元時，會包含已配置佇列管理程式的所有外部資源管理程式。 不過，在下列情況下，呼叫會啟動工作單元，但會完成並發出警告:

- 沒有參與的資源管理程式 (亦即，沒有使用佇列管理程式來配置任何資源管理程式)

or

- 一或多個資源管理程式無法使用。

在這些情況下，工作單元必須只包含工作單元啟動時可用的那些資源管理程式的更新項目。

如果其中一個資源管理程式無法確定其更新項目，則會指示所有資源管理程式回復其更新項目，且 MQCMIT 會完成並產生警告。 在不尋常的情況下 (通常是操作員介入)，如果部分資源管理程式確定其更新項目，但其他資源管理程式回復它們，則 MQCMIT 呼叫可能會失敗; 工作被視為已完成，並產生 混合 結果。 在佇列管理程式的錯誤日誌中診斷此類事件，以便可以採取補救動作。

如果所有涉及的資源管理程式都確定其更新項目，則廣域工作單元的 MQCMIT 會成功。

如需 MQBEGIN 呼叫的說明，請參閱 MQBEGIN。

外部同步點協調

當選取 IBM MQ 以外的同步點協調程式時，會發生這種情況; 例如 CICS、Encina 或 Tuxedo。

在此狀況下， IBM MQ for AIX, Linux, and Windows 系統會向同步點協調程式登錄他們對工作單元結果的興趣，以便他們可以根據需要確定或回復任何未確定的取得或放置作業。 外部同步點協調程式決定是否提供一或兩階段確定通訊協定。

當您使用外部協調程式時，無法發出 MQCMIT、MQBACK 及 MQBEGIN。 呼叫這些函數失敗，原因碼為 MQRC_ENVIRONMENT_ERROR。

外部協調工作單元的啟動方式取決於同步點協調程式所提供的程式設計介面。 可能需要明確呼叫。 如果需要明確呼叫，且您在工作單元未啟動時發出指定 MQPMO_SYNCPOINT 選項的 MQPUT 呼叫，則會傳回完成碼 MQRC_SYNCPOINT_NOT_AVAILABLE。

工作單元的範圍由同步點協調程式決定。 應用程式與佇列管理程式之間的連線狀態會影響應用程式所發出 MQI 呼叫的成功或失敗，而不是工作單元的狀態。 例如，應用程式可以在作用中工作單元期間中斷連線並重新連接至佇列管理程式，並在相同工作單元內執行進一步 MQGET 及 MQPUT 作業。 這稱為擱置斷線。

不論您是否選擇使用 CICS 的 XA 功能，您都可以在 CICS 程式中使用 IBM MQ API 呼叫。 如果您不使用 XA，則不會在 CICS 基本工作單元內管理與佇列之間的訊息放置及取得。 選擇此方法的原因之一是工作單元的整體一致性對您而言並不重要。

如果工作單元的完整性對您很重要，則您必須使用 XA。 當您使用 XA 時， CICS 會使用兩段式確定通訊協定來確保工作單元內的所有資源一起更新。

如需設定交易式支援的相關資訊，請參閱 交易式支援實務範例，以及 TXSeries CICS 文件，例如 *TXSeries for Multiplatforms CICS Administration Guide for Open Systems*。

Multi *Multiplatforms* 上的隱含同步點

隱含的同步點支援可讓持續訊息放置在同步點之外。

放置持續訊息時，會最佳化 IBM MQ，以將持續訊息置於同步點之下。 如果多個應用程式使用同步點，則將持續訊息同時放置到相同佇列中的多個應用程式通常會執行得更好。 這是因為如果在放置持續訊息時使用同步點，則 IBM MQ 的鎖定策略會更有效率。

qm.ini 檔中的 **ImplSyncOpenOutput** 參數可控制當應用程式將持續訊息置於同步點之外時，是否可以新增隱含的同步點。 這可以提供效能改善，而不需要應用程式知道隱含的同步點。

當有多個應用程式同時放入佇列時，隱含同步點只會提高效能，因為它會減少鎖定競用。 **ImplSyncOpenOutput** 指定在新增隱含同步點之前，開啟佇列以供輸出的應用程式數目下限。 預設值為 *2*。 這表示如果您未明確指定 **ImplSyncOpenOutput**，則只有在多個應用程式放入佇列時，才會新增隱含的同步點。

如果您新增隱含同步點，統計資料會反映發生的情況，您可能會看到來自 **runmqsc display conn** 的交易輸出。

如果您永不想要新增隱含的同步點，請設定 **ImplSyncOpenOutput**=*OFF* 。

如需相關資訊，請參閱 調整參數 。

Multiplatforms 上外部同步點管理程式的介面

IBM MQ for Multiplatforms 支援由使用 X/Open XA 介面的外部同步點管理程式來協調交易。

部分 XA 交易管理程式 (TXSeries) 需要每一個 XA 資源管理程式提供其名稱。 這是 XA 交換器結構中稱為 name 的字串。

- ALW AIX, Linux, and Windows 上 IBM MQ 的資源管理程式命名為 MQSeries_XA_RMI。
- IBM i 對於 IBM i，資源管理程式名稱是 MQSeries XA RMI。

如需 XA 介面的進一步詳細資料，請參閱 XA 文件 *CAE* 規格分散式交易處理*: XA* 規格，由 The Open Group 發佈。

在 XA 配置中， IBM MQ for Multiplatforms 履行 XA 資源管理程式的角色。 XA 同步點協調程式可以管理一組 XA 資源管理程式，並將兩個資源管理程式中的交易確定或取消同步化。 這是靜態登錄資源管理程式的運作方式:

1. 應用程式會通知同步點協調程式，它想要啟動交易。
2. 同步點協調程式會對它知道的任何資源管理程式發出呼叫，以將現行交易通知給他們。
3. 應用程式會發出呼叫，以更新與現行交易相關聯的資源管理程式所管理的資源。
4. 應用程式要求同步點協調程式確定或回復交易。
5. 同步點協調程式會使用兩階段確定通訊協定，對每一個資源管理程式發出呼叫，以依照要求來完成交易。

XA 規格需要每一個資源管理程式提供稱為 XA 交換器的結構。 這個結構宣告資源管理程式的功能，以及同步點協調程式所要呼叫的功能。

此結構有兩個版本:

表 *128: XA* 交換器版本	
版本	**說明**
MQRMIXASwitch	靜態 XA 資源管理
MQRMIXASwitchDynamic	動態 XA 資源管理

如需包含此結構的程式庫清單，請參閱 IBM MQ XA 交換器結構。

必須用來將它們鏈結至 XA 同步點協調程式的方法是由協調程式所定義; 請參閱該協調程式所提供的文件，以決定如何讓 IBM MQ 與您的 XA 同步點協調程式合作。

同步點協調程式在任何 *xa_open* 呼叫上傳遞的 *xa_info* 結構可以是要管理的佇列管理程式名稱。 此格式與傳遞至 MQCONN 或 MQCONNX 的佇列管理程式名稱相同，如果要使用預設佇列管理程式，則可以空白。 不過，您可以使用兩個額外參數 TPM 及 AXLIB

TPM 可讓您指定 IBM MQ 交易管理程式名稱，例如 CICS。 AXLIB 可讓您在 XA AX 進入點所在的交易管理程式中指定實際的程式庫名稱。

如果您使用這些參數或非預設佇列管理程式，則必須使用 QMNAME 參數指定佇列管理程式名稱。 如需進一步資訊，請參閱 xa_open 字串的 CHANNEL、TRPTYPE、CONNAME 及 QMNAME 參數。

限制

1. 共用 Hconn 不容許廣域工作單元 (如 第 619 頁的『與 MQCONNX 共用 (與執行緒無關) 連線』中所述)。
2. IBM i IBM MQ for IBM i 不支援 XA 資源管理程式的動態登錄。

   唯一支援的交易管理程式是 WebSphere Application Server。
3. Windows 在 Windows 系統上， XA 參數中宣告的所有函數都宣告為 _cdecl 函數。
4. 外部同步點協調程式一次只能管理一個佇列管理程式。 這是因為協調程式具有與每一個佇列管理程式的有效連線，因此遵循一次只容許一個連線的規則。

註: 附註: 在 JEE 伺服器中執行的 JMS 用戶端應用程式 (CLIENT JEE 應用程式) 沒有此限制，因此單一 JEE 伺服器管理的交易可以協調相同交易中的多個佇列管理程式。 不過，以連結模式執行的 JMS 伺服器應用程式仍遵循一次只容許一個連線的規則。

5. 使用同步點協調程式執行的所有應用程式只能連接至協調程式所管理的佇列管理程式，因為它們已有效地連接至該佇列管理程式。 他們必須發出 MQCONN 或 MQCONNX 以取得連線控點，且必須在結束之前發出 MQDISC。 或者，他們也可以對 TXSeries CICS 使用結束程式 UE014015 。

IBM i *IBM i 外部同步點管理程式的介面*

IBM MQ for IBM i 可以使用原生 IBM i 確定控制作為外部同步點協調程式。

確定控制不容許與執行緒無關 (共用) 的連線。 如需 IBM i 確定控制功能的相關資訊，請參閱 *IBM i Programming: Backup and Recovery Guide (SC21-8079)*。

若要啟動 IBM i 確定控制機能，請使用 STRCMTCTL 系統指令。 若要結束確定控制，請使用 ENDCMTCTL 系統指令。

註: 確定定義範圍 的預設值是 *ACTGRP。 對於 IBM i，這必須定義為 IBM MQ 的 *JOB。 例如:

```
STRCMTCTL LCKLVL(*ALL) CMTSCOPE(*JOB)
```

IBM MQ for IBM i 也可以執行只包含 IBM MQ 資源更新項目的本端工作單元。 當應用程式呼叫 MQPUT、MQPUT1 或 MQGET ，並指定 MQPMO_SYNCPOINT 或 MQGMO_SYNCPOINT 或 MQBEGIN 時，會在每一個應用程式中選擇本端工作單元與參與 IBM i 所協調的廣域工作單元。 如果發出第一次此類呼叫時，確定控制不在作用中，則 IBM MQ 會啟動本端工作單元，且此 IBM MQ 連線的所有其他工作單元也會使用本端工作單元，而不論是否啟動確定控制。 若要確定本端工作單元，請使用 MQCMIT。 若要退出本端工作單元，請使用 MQBACK。 IBM i 確定及回復呼叫 (例如 CL 指令 COMMIT) 對 IBM MQ 本端工作單元沒有影響。

如果您要將 IBM MQ for IBM i 與原生 IBM i 確定控制搭配使用作為外部同步點協調程式，請確定任何具有確定控制的工作處於作用中，且您在單一執行緒工作中使用 IBM MQ 。 如果您在已啟動確定控制的多執行緒工作中呼叫 MQPUT、 MQPUT1 或 MQGET (指定 MQPMO_SYNCPOINT 或 MQGMO_SYNCPOINT) ，則呼叫會失敗，原因碼為 MQRC_SYNCPOINT_NOT_AVAILABLE。

可以在多執行緒工作中使用本端工作單元及 MQCMIT 和 MQBACK 呼叫。

如果您呼叫 MQPUT、 MQPUT1 或 MQGET ，並指定 MQPMO_SYNCPOINT 或 MQGMO_SYNCPOINT ，則在啟動確定控制之後， IBM MQ for IBM i 會將本身作為 API 確定資源新增至確定定義。 這通常是工作中的第一個此類呼叫。 雖然在特定確定定義下登錄了任何 API 確定資源，但您無法結束該定義的確定控制。

當您切斷與佇列管理程式的連線時，如果現行工作單元中沒有擱置中的 MQI 作業，則 IBM MQ for IBM i 會移除其作為 API 確定資源的登錄。

如果您在現行工作單元中有擱置中 MQPUT、 MQPUT1 或 MQGET 作業時中斷與佇列管理程式的連線， IBM MQ for IBM i 會保持登錄為 API 確定資源，以便通知它下一次確定或回復。 當達到下一個同步點時， IBM MQ for IBM i 會根據需要確定或回復變更。 在作用中工作單元期間，應用程式可以中斷連線並重新連接至佇列管理程式，並在相同工作單元內執行進一步 MQGET 及 MQPUT 作業 (這是擱置中斷連線)。

如果您嘗試對該確定定義發出 ENDCMTCTL 系統指令，則會發出 CPF8355 訊息，指出擱置中的變更已在作用中。 當工作結束時，此訊息也會出現在工作日誌中。 若要避免此情況，請確定或回復所有擱置中 IBM MQ for IBM i 作業，並與佇列管理程式中斷連線。 因此，在 ENDCMTCTL 之前使用 COMMIT 或 ROLLBACK 指令可順利完成結束確定控制。

當您使用 IBM i 確定控制作為外部同步點協調程式時，無法發出 MQCMIT、MQBACK 及 MQBEGIN 呼叫。 呼叫這些函數失敗，原因碼為 MQRC_ENVIRONMENT_ERROR。

若要確定或回復 (亦即，取消) 您的工作單元，請使用其中一個支援確定控制的程式設計語言。 例如:

- CL 指令 :COMMIT 及 ROLLBACK
- ILE C 程式設計函數: _Rcommit 及 _Rrollback
- ILE RPG: COMMIT 及 ROLBK
- COBOL/400:COMMIT 及 ROLLBACK

當您使用 IBM i 確定控制作為 IBM MQ for IBM i 的外部同步點協調程式時， IBM i 會執行 IBM MQ 參與的兩階段確定通訊協定。 由於每一個工作單元是分兩個階段來確定，在第一個階段投票確定之後，第二個階段可能會無法使用佇列管理程式。 例如，如果佇列管理程式的內部工作已結束，則可能會發生此情況。 在此狀況下，執行確定的工作日誌包含訊息 CPF835F ，指出確定或回轉作業失敗。 前面的訊息指出問題的原因，是否在確定或回復作業期間發生，以及失敗工作單元的邏輯工作單元 ID (LUWID)。

如果問題是由 IBM MQ API 確定資源在備妥工作單元的確定或回復期間失敗所造成，您可以使用 WRKMQMTRN 指令來完成作業並還原交易的完整性。 指令需要您知道要確定及取消之工作單元的 LUWID。

使用觸發程式啟動 IBM MQ 應用程式

瞭解觸發程式以及如何使用觸發程式來啟動 IBM MQ 應用程式。

某些負責處理佇列的 IBM MQ 應用程式會持續執行，因此一律可用來擷取抵達佇列的訊息。 不過，當抵達佇列的訊息數無法預期時，您可能不希望這樣做。 在此情況下，即使沒有要擷取的訊息，應用程式也可能會耗用系統資源。

IBM MQ 提供一種機能，可讓應用程式在有可供擷取的訊息時自動啟動。 此機能稱為 觸發。

如需觸發通道的相關資訊，請參閱 觸發通道。

什麼是觸發?

佇列管理程式會將某些條件定義為構成 觸發事件。

如果啟用佇列的觸發，且發生觸發事件，則佇列管理程式會將 觸發訊息 傳送至稱為 起始佇列的佇列。 起始佇列上的觸發訊息存在指出已發生觸發事件。

佇列管理程式所產生的觸發訊息不會持續存在。 這會減少記載 (導致改善效能) ，並在重新啟動期間將重複項減至最少，以改善重新啟動時間。

處理起始佇列的程式稱為 觸發監視器應用程式，其功能是讀取觸發訊息並根據觸發訊息中包含的資訊採取適當的動作。 通常此動作是啟動某個其他應用程式，以處理產生觸發訊息的佇列。 從佇列管理程式的觀點來看，觸發監視器應用程式並不特殊; 它只是另一個從佇列 (起始佇列) 讀取訊息的應用程式。

如果啟用佇列的觸發，您可以建立與其相關聯的 程序定義物件 。 此物件包含應用程式的相關資訊，該應用程式會處理導致觸發事件的訊息。 如果建立程序定義物件，佇列管理程式會擷取此資訊，並將它放在觸發訊息中，供觸發監視器應用程式使用。 與佇列相關聯的程序定義名稱由 *ProcessName* 本端佇列屬性提供。 每一個佇列都可以指定不同的程序定義，或是數個佇列可以共用相同的程序定義。

如果您想要觸發通道的啟動，則不需要定義程序定義物件。 改用傳輸佇列定義。

在 AIX, Linux, and Windows 上執行的 IBM MQ 用戶端支援觸發。 在用戶端環境中執行的應用程式與在完整 IBM MQ 環境中執行的應用程式相同，但您將它與用戶端程式庫鏈結在一起。 不過，觸發監視器和要啟動的應用程式必須都在相同的環境中。

觸發涉及:

應用程式佇列
應用程式佇列 是本端佇列，當觸發設定為開啟且符合條件時，需要寫入觸發訊息。

程序定義
應用程式佇列可以具有相關聯的 程序定義物件 ，用於保留將從應用程式佇列取得訊息之應用程式的詳細資料。 (如需屬性清單，請參閱 程序定義的屬性 。)

請記住，如果您想要觸發程式啟動通道，則不需要定義程序定義物件。

傳輸佇列
如果您想要觸發程式啟動通道，則需要傳輸佇列。

對於 Linux 以外任何平台上的傳輸佇列，傳輸佇列的 *TriggerData* 屬性可以指定要啟動的通道名稱。 這可以取代觸發通道的程序定義，但僅在未建立程序定義時使用。

觸發事件
觸發事件 是導致佇列管理程式產生觸發訊息的事件。 這通常是到達應用程式佇列的訊息，但也可能在其他時間發生。 例如，請參閱第 731 頁的『觸發事件的條件』。

IBM MQ 有一系列選項可讓您控制導致觸發事件的條件 (請參閱 第 734 頁的『控制觸發程式事件』)。

觸發訊息

當佇列管理程式辨識觸發事件時，會建立 觸發訊息 。 它會複製到要啟動之應用程式的觸發訊息資訊。 此資訊來自應用程式佇列，以及與應用程式佇列相關聯的程序定義物件。

觸發訊息具有固定格式 (請參閱 第 740 頁的『觸發訊息的格式』)。

起始佇列

起始佇列 是佇列管理程式放置觸發訊息的本端佇列。 請注意，起始佇列不能是別名佇列或模型佇列。

佇列管理程式可以擁有多個起始佇列，且每一個佇列都與一或多個應用程式佇列相關聯。

z/OS 共用佇列 (佇列共用群組中的佇列管理程式可存取的本端佇列) 可以是 IBM MQ for z/OS 上的起始佇列。

觸發監視器 (trigger monitor)

觸發監視器 是持續執行的程式，為一或多個起始佇列提供服務。 當觸發訊息抵達起始佇列時，觸發監視器會擷取該訊息。 觸發監視器會使用觸發訊息中的資訊。 它會發出指令來啟動應用程式，以擷取到達應用程式佇列的訊息，並傳遞包含在觸發訊息標頭中的資訊，其中包括應用程式佇列的名稱。

在所有平台上，稱為通道起始程式的特殊觸發監視器負責啟動通道。

z/OS 在 z/OS 上，通道起始程式通常是手動啟動，或者可以在佇列管理程式啟動 JCL 中變更 CSQINP2 來啟動佇列管理程式時自動執行。

Multi 在 多平台上，當佇列管理程式啟動時，會自動啟動通道起始程式，或者可以使用 **runmqchi** 指令手動啟動通道起始程式。

如需相關資訊，請參閱 第 737 頁的『觸發監視器的起始佇列處理』。

若要瞭解觸發如何運作，請考量 第 726 頁的圖 95，這是觸發類型 FIRST (MQTT_FIRST) 的範例。

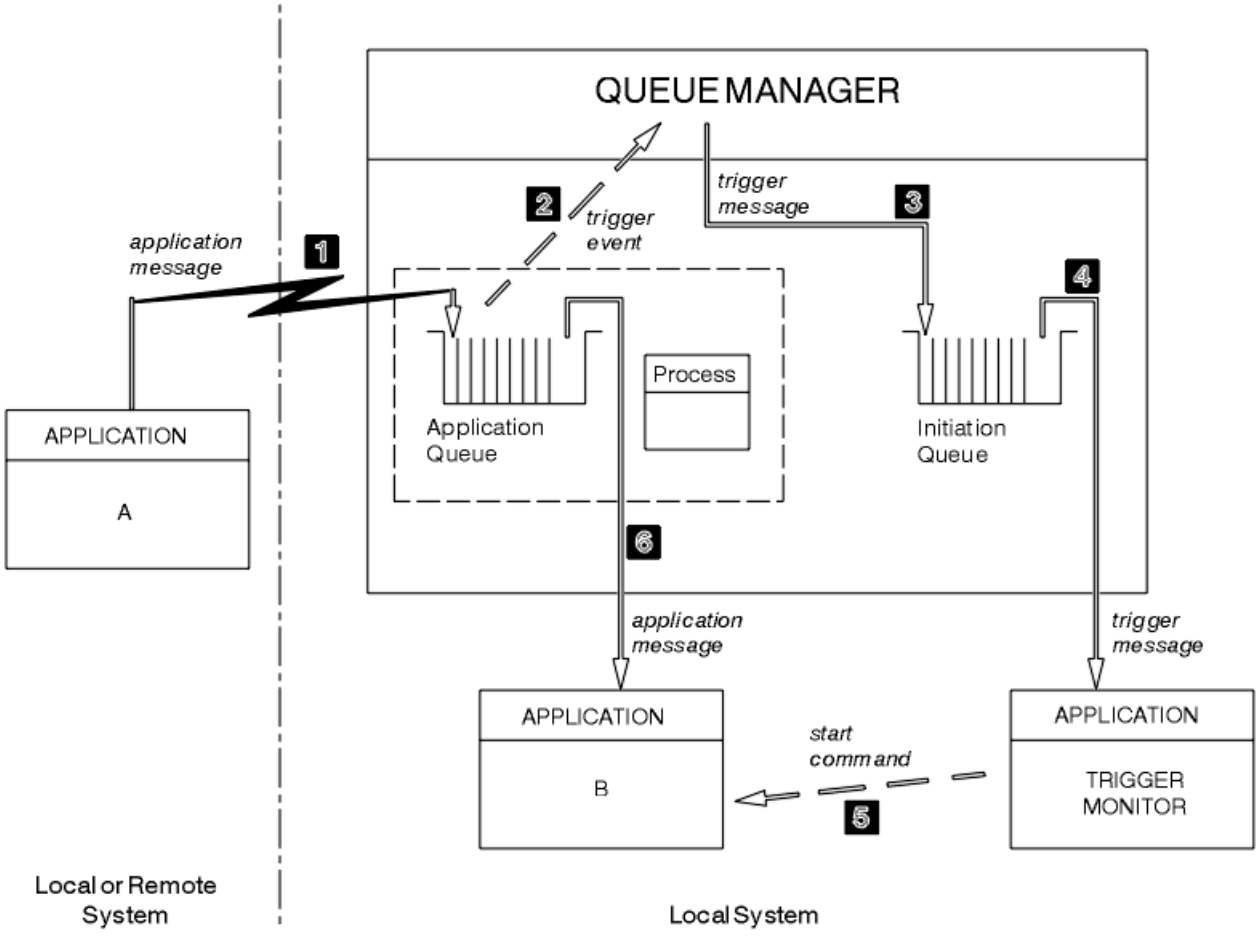


圖 *95:* 應用程式及觸發訊息的流程

在 第 726 頁的圖 95 中，事件順序如下:

1. 應用程式 A (可以是佇列管理程式的本端或遠端) 會將訊息放置在應用程式佇列上。 沒有應用程式開啟此佇列以供輸入。 不過，此事實僅與觸發類型 FIRST 及 DEPTH 相關。
2. 佇列管理程式會檢查是否符合條件，在這些條件下必須產生觸發事件。 它們是，並且會產生觸發事件。 建立觸發訊息時，會使用保留在相關聯程序定義物件內的資訊。
3. 佇列管理程式會建立觸發訊息，並將它放置在與此應用程式佇列相關聯的起始佇列上，但前提是應用程式 (觸發監視器) 已開啟起始佇列以供輸入。
4. 觸發監視器會從起始佇列擷取觸發訊息。
5. 觸發監視器會發出指令來啟動應用程式 B (伺服器應用程式)。
6. 應用程式 B 會開啟應用程式佇列並擷取訊息。

註:

1. 如果應用程式佇列已由任何程式開啟以供輸入，且已針對 FIRST 或 DEPTH 設定觸發，則不會發生觸發事件，因為已提供佇列。
2. 如果未開啟起始佇列以供輸入，則佇列管理程式不會產生任何觸發訊息; 它會等待直到應用程式開啟起始佇列以供輸入為止。
3. 使用通道觸發時，請使用觸發類型 FIRST 或 DEPTH。
4. 觸發應用程式以啟動觸發監視器的使用者、 CICS 使用者或啟動佇列管理程式的使用者的使用者 ID 及群組來執行。

到目前為止，觸發內的佇列之間的關係只是一對一。 請考量 第 728 頁的圖 96。

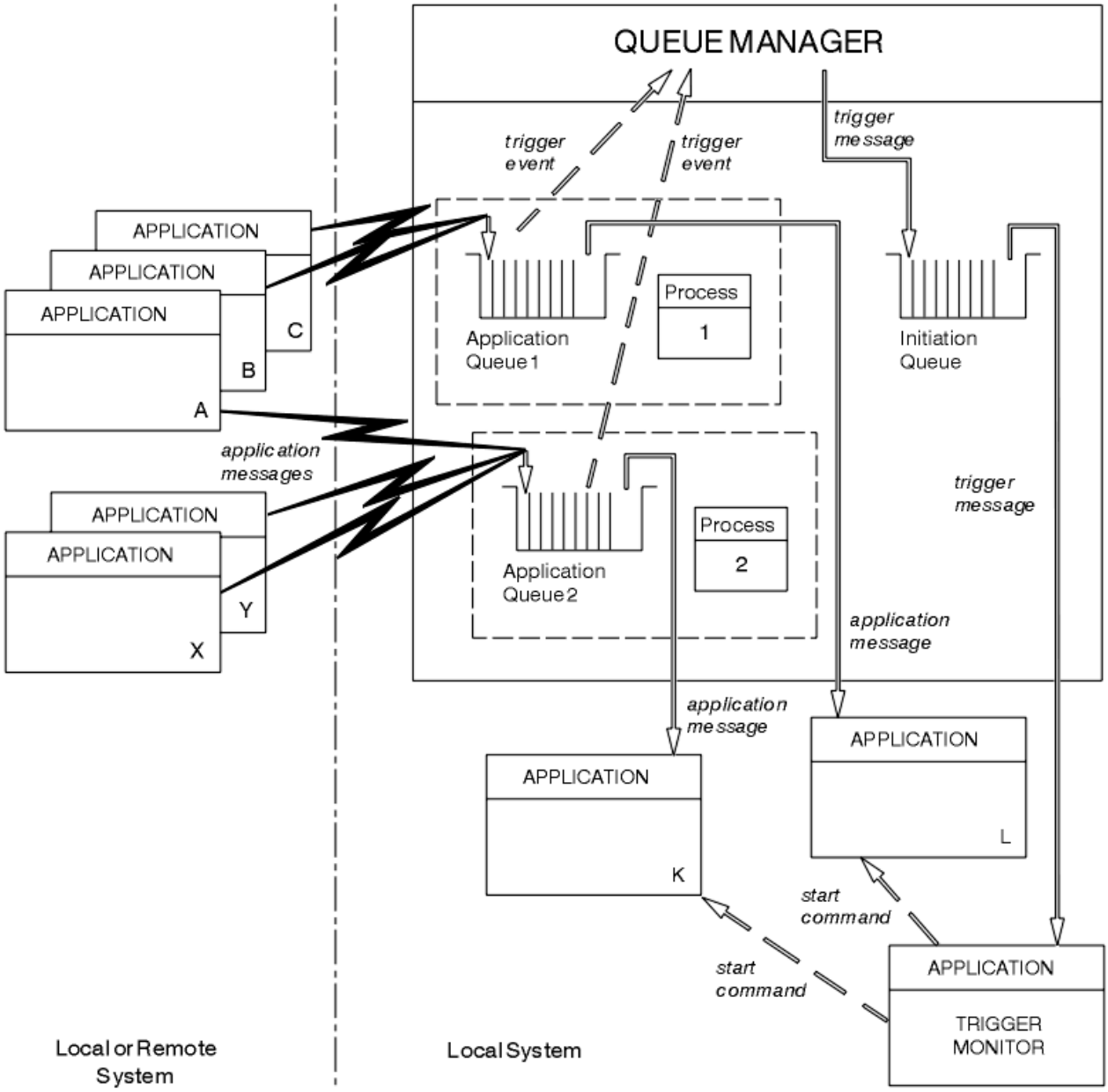


圖 *96:* 觸發內佇列的關係

應用程式佇列具有與其相關聯的程序定義物件，可保留將處理訊息之應用程式的詳細資料。 佇列管理程式會將資訊放在觸發訊息中，因此只需要一個起始佇列。 觸發監視器會從觸發訊息中擷取此資訊，並啟動相關應用程式來處理每一個應用程式佇列上的訊息。

請記住，如果您想要觸發通道的啟動，則不需要定義程序定義物件。 傳輸佇列定義可以決定要觸發的通道。

請使用下列鏈結，以進一步瞭解如何使用觸發程式來啟動 IBM MQ 應用程式:

- 第 729 頁的『觸發的必要條件』
- 第 731 頁的『觸發事件的條件』
- 第 734 頁的『控制觸發程式事件』
- 第 736 頁的『設計使用觸發佇列的應用程式』
- 第 737 頁的『觸發監視器的起始佇列處理』
- 第 739 頁的『觸發訊息的內容』
- 第 741 頁的『當觸發無法運作時』

相關概念

第 604 頁的『訊息佇列介面概觀』
瞭解「訊息佇列介面 (MQI)」元件。

第 616 頁的『連接至佇列管理程式並切斷與佇列管理程式的連線』
若要使用 IBM MQ 程式設計服務，程式必須具有與佇列管理程式的連線。 使用此資訊來瞭解如何連接至佇列管理程式，以及與佇列管理程式中斷連線。

第 621 頁的『開啟及關閉物件』
此資訊提供開啟及關閉 IBM MQ 物件的見解。

第 630 頁的『將訊息放置在佇列上』
使用此資訊來瞭解如何將訊息放置在佇列上。

第 643 頁的『從佇列取得訊息』
使用此資訊來瞭解從佇列取得訊息的相關資訊。

第 713 頁的『查詢及設定物件屬性』
屬性是定義 IBM MQ 物件性質的內容。

第 716 頁的『確定及取消工作單元』
此資訊說明如何確定及取消工作單元中發生的任何可回復取得及放置作業。

第 741 頁的『使用 MQI 及叢集』
對於與叢集作業相關的呼叫和回覆碼，有一些特殊選項。

第 745 頁的『在 IBM MQ for z/OS 上使用及撰寫應用程式』
IBM MQ for z/OS 應用程式可以由在許多不同環境中執行的程式組成。 這表示他們可以利用多個環境中可用的設施。

第 57 頁的『IBM MQ for z/OS 上的 IMS 及 IMS 橋接器應用程式』
此資訊可協助您使用 IBM MQ 撰寫 IMS 應用程式。

觸發的必要條件

使用此資訊來瞭解在使用觸發之前要採取的步驟。

在您的應用程式可以利用觸發之前，請完成下列步驟:

1. 您可以:
   a. 建立應用程式佇列的起始佇列。 例如:

```
DEFINE QLOCAL (initiation.queue) REPLACE      +
       LIKE (SYSTEM.DEFAULT.INITIATION.QUEUE)      +
       DESCR ('initiation queue description')
```

   or

   b. 判斷存在且可供應用程式使用的本端佇列名稱 (通常，此名稱是 SYSTEM.DEFAULT.INITIATION.QUEUE 或 SYSTEM.CHANNEL.INITQ) ， 並在應用程式佇列的 *InitiationQName* 欄位中指定其名稱。

2. 將起始佇列與應用程式佇列相關聯。 佇列管理程式可以擁有多個起始佇列。 您可能想要一些應用程式佇列由不同的程式提供服務，在此情況下，您可以對每一個服務程式使用一個起始佇列 (雖然您不需要這樣做)。 以下是如何建立應用程式佇列的範例:

```
DEFINE QLOCAL (application.queue) REPLACE   +
LIKE (SYSTEM.DEFAULT.LOCAL.QUEUE)           +
DESCR ('appl queue description')    +
INITQ (initiation.queue)          +
PROCESS (process.name)            +
TRIGGER                                     +
TRIGTYPE (FIRST)
```

IBM i 以下是 IBM MQ for IBM i 的 CL 程式摘錄，它會建立起始佇列:

```
/*   Queue used by AMQSINQA                                        */
            CRTMQMQ     QNAME('SYSTEM.SAMPLE.INQ')                  +
                        QTYPE(*LCL)  REPLACE(*YES)                  +
                        MQMNAME                                     +
                        TEXT('queue for AMQSINQA')                  +
                        SHARE(*YES)                 /* Shareable   */+
                        DFTMSGPST(*YES)/* Persistent messages OK  */+
                                                                    +
                        TRGENBL(*YES)  /* Trigger control on      */+
                        TRGTYPE(*FIRST)/* Trigger on first message*/+
                        PRCNAME('SYSTEM.SAMPLE.INQPROCESS')         +
                        INITQNAME('SYSTEM.SAMPLE.TRIGGER')
```

3. 如果您要觸發應用程式，請建立程序定義物件，以包含與要提供應用程式佇列之應用程式相關的資訊。例如，若要觸發-啟動稱為 PAYR 的 CICS 薪資交易:

```
  DEFINE PROCESS (process.name) +
         REPLACE +
         DESCR ('process description') +
         APPLICID ('PAYR') +
         APPLTYPE (CICS) +
         USERDATA ('Payroll data')
```

IBM i 以下是 IBM MQ for IBM i 的 CL 程式摘錄，可建立程序定義物件:

```
/*   Process definition                                            */
            CRTMQMPRC   PRCNAME('SYSTEM.SAMPLE.INQPROCESS')         +
                        REPLACE(*YES)                               +
                        MQMNAME                                     +
                        TEXT('trigger process for AMQSINQA')        +
                        ENVDATA('JOBPTY(3)') /* Submit parameter  */+
                        APPID('AMQSINQA')    /* Program name      */
```

當佇列管理程式建立觸發訊息時，它會將資訊從程序定義物件的屬性複製到觸發訊息中。

平台	建立程序定義物件
AIX, Linux, and Windows 系統	使用 DEFINE PROCESS 或使用 SYSTEM.DEFAULT.PROCESS 並使用 ALTER PROCESS 修改
z/OS z/OS z/OS	使用 DEFINE PROCESS (請參閱步驟 第 730 頁的『3』 中的範例程式碼)，或使用作業和控制台。
IBM i IBM i IBM i	使用包含程式碼的 CL 程式，如步驟 第 730 頁的『3』。

4. 選用項目: 建立傳輸佇列定義，並對 **ProcessName** 屬性使用空白。

   TrigData 屬性可以包含要觸發的通道名稱，也可以保留空白。 除了 IBM MQ for z/OS 之外，如果保留空白，通道起始程式會搜尋通道定義檔，直到找到與指名傳輸佇列相關聯的通道為止。 當佇列管理程式建立觸發訊息時，它會將資訊從傳輸佇列定義的 **TrigData** 屬性複製到觸發訊息中。

5. 如果您已建立程序定義物件來指定要提供應用程式佇列之應用程式的內容，請透過在佇列的 **ProcessName** 屬性中命名程序物件，將該程序物件與應用程式佇列相關聯。

平台	使用指令
AIX, Linux, and Windows 系統	ALTER QLOCAL

平台	使用指令
z/OS z/OS z/OS	ALTER QLOCAL
IBM i IBM i IBM i	CHGMQMQ

6. 啟動觸發監視器 IBM i (或 IBM MQ for IBM i 中的觸發伺服器) 的實例，以處理您已定義的起始佇列。 如需相關資訊，請參閱第 737 頁的『觸發監視器的起始佇列處理』。

如果您想要知道任何未遞送的觸發訊息，請確定您的佇列管理程式已定義無法傳送郵件 (未遞送訊息) 的佇列。 在 *DeadLetterQName* 佇列管理程式欄位中指定佇列的名稱。

然後，您可以使用定義應用程式佇列之佇列物件的屬性，來設定您需要的觸發條件。 如需相關資訊，請參閱第 734 頁的『控制觸發程式事件』。

觸發事件的條件

當滿足本主題中詳細說明的條件時，佇列管理程式會建立觸發訊息。

這個主題中的共用佇列參照表示佇列共用群組中的共用佇列，只適用於 IBM MQ for z/OS。

下列狀況會導致佇列管理程式建立觸發訊息:

1. 訊息會 放置 在佇列上。
2. 訊息的優先順序大於或等於佇列的臨界值觸發優先順序。 此優先順序設定在 **TriggerMsgPriority** 本端佇列屬性中; 如果設為零，則任何訊息都符合條件。
3. 視 *TriggerType* 而定，先前佇列中優先順序大於或等於 *TriggerMsgPriority* 的訊息數:
   - 零 (適用於觸發類型 MQTT_FIRST)
   - 任何數字 (適用於觸發類型 MQTT_EVERY)
   - *TriggerDepth* 減 1 (適用於觸發程式類型 MQTT_DEPTH)

   註:

   a. 對於非共用本端佇列，當它評量觸發事件的條件是否存在時，佇列管理程式會同時計算已確定及未確定的訊息。 因此，因為佇列上的訊息尚未確定，所以沒有訊息可供應用程式擷取時，可能會啟動應用程式。 在此狀況下，請考量將等待選項與適當的 *WaitInterval* 搭配使用，以便應用程式等待其訊息送達。

   b. 對於本端共用佇列，佇列管理程式只會計算已確定的訊息。
4. 對於 FIRST 或 DEPTH 類型的觸發，沒有任何程式開啟應用程式佇列來移除訊息 (亦即，**OpenInputCount** 本端佇列屬性為零)。

   註:

   a. 對於共用佇列，當多個佇列管理程式具有針對佇列執行的觸發監視器時，會套用特殊條件。 在此狀況下，如果一個以上佇列管理程式已開啟佇列以供輸入共用，則其他佇列管理程式上的觸發準則會被視為 *TriggerType* MQTT_FIRST 及 *TriggerMsgPriority* 零。 當所有佇列管理程式都關閉佇列以供輸入時，觸發條件會回復為佇列定義中指定的那些條件。

   受此條件影響的範例實務是多個佇列管理程式 QM1、 QM2 及 QM3 ， 並針對應用程式佇列 A 執行觸發監視器。 訊息到達滿足觸發條件的 A ， 並在起始佇列上產生觸發訊息。 QM1 上的觸發監視器會取得觸發訊息並觸發應用程式。 觸發的應用程式會開啟應用程式佇列以進行共用輸入。 從這個點開始，應用程式佇列 A 的觸發條件會評估為 *TriggerType* MQTT_FIRST ， 並 *TriggerMsgPriority* 在佇列管理程式 QM2 及 QM3 中為零，直到 QM1 關閉應用程式佇列為止。

   b. 對於共用佇列，此條件適用於每一個佇列管理程式。 亦即，佇列管理程式的 *OpenInputCount* 必須為零，該佇列管理程式才會為佇列產生觸發訊息。 不過，如果佇列共用群組中的任何佇列管理程式使用 MQOO_INPUT_EXCLUSIVE 選項開啟佇列，則佇列共用群組中的任何佇列管理程式不會針對該佇列產生觸發訊息。

當觸發應用程式開啟佇列以供輸入時，會變更觸發條件的評估方式。 在只有一個觸發監視器執行的實務範例中，其他應用程式可能具有相同的效果，因為它們會以類似方式開啟應用程式佇列以供輸入。 不論應用程式佇列是由觸發監視器所啟動的應用程式開啟，還是由某個其他應用程式開啟; 事實上，該佇列已開啟以供另一個佇列管理程式上的輸入，而導致觸發準則變更。

5. 在 IBM MQ for z/OS 上，如果應用程式佇列是 **Usage** 屬性為 MQUS_NORMAL 的佇列，則不會禁止其取得要求 (亦即， **InhibitGet** 佇列屬性是 MQQA_GET_ALLOWED)。 此外，如果觸發的應用程式佇列是 **Usage** 屬性為 MQUS_XMITQ 的佇列，則不會禁止取得其要求。
6. 您可以:
   - 佇列的 **ProcessName** 本端佇列屬性不是空白，且已建立該屬性所識別的程序定義物件，或
   - 佇列的 **ProcessName** 本端佇列屬性都是空白，但佇列是傳輸佇列。 由於程序定義是選用的，因此 **TriggerData** 屬性也可能包含要啟動的通道名稱。 在此情況下，觸發訊息包含具有下列值的屬性:
     - **QName**: 佇列名稱
     - **ProcessName**: 空白
     - **TriggerData**: 觸發程式資料
     - **ApplType**:MQAT_UNKNOWN
     - **ApplId**: 空白
     - **EnvData**: 空白
     - **UserData**: 空白
7. 已建立起始佇列，且已在 **InitiationQName** 本端佇列屬性中指定起始佇列。 此外:
   - 起始佇列不禁止取得要求 (亦即， **InhibitGet** 佇列屬性的值為 MQQA_GET_ALLOWED)。
   - 起始佇列不得禁止放置要求 (亦即， **InhibitPut** 佇列屬性值必須為 MQQA_PUT_ALLOWED)。
   - 起始佇列的 **Usage** 屬性值必須是 MQUS_NORMAL。
   - 在支援動態佇列的環境中，起始佇列不能是已標示為邏輯刪除的動態佇列。
8. 觸發監視器目前已開啟起始佇列以移除訊息 (亦即， **OpenInputCount** 本端佇列屬性大於零)。
9. 應用程式佇列的觸發控制 (**TriggerControl** 本端佇列屬性) 設為 MQTC_ON。 若要這樣做，請在定義佇列時設定 **trigger** 屬性，或使用 ALTER QLOCAL 指令。
10. 觸發類型 (**TriggerType** 本端佇列屬性) 不是 MQTT_NONE。

    如果符合所有必要條件，且導致觸發條件的訊息放置為工作單元的一部分，則觸發監視器應用程式無法擷取觸發訊息，直到工作單元完成 (不論工作單元是否已確定，或針對觸發類型 MQTT_FIRST 或 MQTT_DEPTH) 取消為止。
11. 針對 MQTT_FIRST 或 MQTT_DEPTH 的 **TriggerType** 及佇列，會在佇列上放置適當的訊息:
    - 先前不是空的 (MQTT_FIRST)，或
    - 有 **TriggerDepth** 則以上訊息 (MQTT_DEPTH)

    並滿足條件 第 731 頁的『2』 至 第 732 頁的『10』 (排除 第 731 頁的『3』)，如果在 MQTT_FIRST 的情況下，自針對此佇列寫入前次觸發訊息以來已經歷足夠的間隔 (**TriggerInterval** 佇列管理程式屬性)。

    這是容許佇列伺服器在處理佇列上的所有訊息之前結束。 觸發間隔的目的是減少產生重複觸發訊息的數目。

    註: 如果您停止並重新啟動佇列管理程式，則會重設 **TriggerInterval** 計時器。 有一個小視窗，在此期間可以產生兩個觸發訊息。 當訊息到達且佇列先前不是空的 (MQTT_FIRST) 或具有 **TriggerDepth** 或更多訊息 (MQTT_DEPTH) 時，如果佇列的觸發屬性設為啟用，則視窗會存在。
12. 負責處理佇列的唯一應用程式會針對 MQTT_FIRST 或 MQTT_DEPT_DEPTH 的 **TriggerType** 發出 MQCLOSE 呼叫，且至少有:
    - 一個 (MQTT_FIRST)，或
    - **TriggerDepth** (MQTT_DEPTH)

同時滿足佇列上具有足夠優先順序的訊息 (條件 第 731 頁的『2』)，以及 第 732 頁的『6』 到 第 732 頁的『10』 的條件。

這是為了容許佇列伺服器發出 MQGET 呼叫，發現佇列是空的，因而結束; 不過，在 MQGET 與 MQCLOSE 呼叫之間的間隔中，有一或多個訊息到達。

註:

a. 如果負責處理應用程式佇列的程式未擷取所有訊息，這可能會導致封閉迴圈。 每次程式關閉佇列時，佇列管理程式會建立另一個觸發訊息，使觸發監視器重新啟動伺服器程式。

b. 如果負責處理應用程式佇列的程式在關閉佇列之前取消其取得要求 (或程式異常終止)，則會發生相同的情況。 不過，如果程式在取消取得要求之前關閉佇列，且佇列是空的，則不會建立觸發訊息。

c. 若要防止此類迴圈發生，請使用 MQMD 的 *BackoutCount* 欄位來偵測反覆取消的訊息。 如需相關資訊，請參閱 第 39 頁的『已取消的訊息』。

13. 使用 MQSET 或指令滿足下列條件:

a. • **TriggerControl** 變更為 MQTC_ON，或

• **TriggerControl** 已經是 MQTC_ON，且 **TriggerType**、 **TriggerMsgPriority** 或 **TriggerDepth** (如果相關的話) 的值已變更,

而且至少有:

• 一個 (MQTT_FIRST 或 MQTT_EVERY)，或

• **TriggerDepth** (MQTT_DEPTH)

佇列上具有足夠優先順序的訊息 (條件 第 731 頁的『2』)，以及 第 731 頁的『4』 到 第 732 頁的『10』 的條件 (排除 第 732 頁的『8』) 也很滿足。

這是在已滿足觸發條件時，容許應用程式或操作員變更觸發準則。

b. 起始佇列的 **InhibitPut** 佇列屬性值從 MQQA_PUT_INHIBITED 變更為 MQQA_PUT_ALLOWED，且至少有:

• 一個 (MQTT_FIRST 或 MQTT_EVERY)，或

• **TriggerDepth** (MQTT_DEPTH)

優先順序足夠的訊息 (條件 第 731 頁的『2』) 在此起始佇列的任何佇列上，也會滿足 第 731 頁的『4』 透過 第 732 頁的『10』 的條件。 (會針對每一個滿足條件的這類佇列產生一則觸發訊息。)

這是為了容許因為起始佇列上的 MQQA_PUT_INHIBITED 條件而未產生觸發訊息，但現在已變更此條件。

c. 應用程式佇列的 **InhibitGet** 佇列屬性值從 MQQA_GET_INHIBITED 變更為 MQQA_GET_ALLOWED，且至少有:

• 一個 (MQTT_FIRST 或 MQTT_EVERY)，或

• **TriggerDepth** (MQTT_DEPTH)

優先順序足夠的訊息 (條件 第 731 頁的『2』) 在佇列上，也會滿足 第 731 頁的『4』 到 第 732 頁的『10』 (不包括 第 732 頁的『5』) 的條件。

只有在應用程式可以從應用程式佇列擷取訊息時，才會觸發應用程式。

d. 觸發監視器應用程式會針對來自起始佇列的輸入發出 MQOPEN 呼叫，且至少有:

• 一個 (MQTT_FIRST 或 MQTT_EVERY)，或

• **TriggerDepth** (MQTT_DEPTH)

優先順序足夠的訊息 (條件 第 731 頁的『2』) 在此為起始佇列的任何應用程式佇列上，以及 第 731 頁的『4』 透過 第 732 頁的『10』 的條件 (不包括 第 732 頁的『8』) 也滿足，且沒有其他應用程式開啟起始佇列以供輸入 (針對每一個符合條件的此類佇列會產生一個觸發訊息)。

這是為了容許訊息在觸發監視器未執行時到達佇列，以及佇列管理程式重新啟動及觸發訊息 (非持續性) 遺失。

14. 已正確設定 MSGDLVSQ。 如果設定 MSGDLVSQ=FIFO，則會以「先進先出」方式將訊息遞送至佇列。 系統不處理訊息的優先順序，並將佇列的預設優先順序指派給訊息。 如果 **TriggerMsgPriority** 設為高於佇列預設優先順序的值，則不會觸發任何訊息。 如果 **TriggerMsgPriority** 設為等於或低於佇列的預設優先順序，則會針對類型 FIRST、EVERY 及 DEPTH 進行觸發。 如需這些類型的相關資訊，請參閱 第 734 頁的『控制觸發程式事件』下 **TriggerType** 欄位的說明。

    如果您設定 MSGDLVSQ=優先順序，且訊息優先順序等於或大於 *TriggerMsgPriority* 欄位，則訊息只會計入觸發事件。 在此情況下，會發生 FIRST、EVERY 及 DEPTH 類型的觸發。 例如，如果您放置 100 則優先順序低於 **TriggerMsgPriority** 的訊息，則用於觸發目的的有效佇列深度仍為零。 如果您接著將另一則訊息放置在佇列上，但這次優先順序大於或等於 **TriggerMsgPriority**，則有效佇列深度會從零增加到一，且滿足 **TriggerType** FIRST 的條件。

附註:

1. 從步驟 第 732 頁的『12』 (其中觸發訊息是因訊息到達應用程式佇列以外的某個事件而產生)，觸發訊息不會作為工作單元的一部分放置。 此外，如果 **TriggerType** 是 MQTT_EVERY，且應用程式佇列上有一或多個訊息，則只會產生一個觸發訊息。
2. 如果 IBM MQ 在 MQPUT 期間對訊息進行分段，則除非所有區段都已順利放置在佇列上，否則不會處理觸發事件。 不過，一旦訊息區段位於佇列上， IBM MQ 就會將它們視為個別訊息來進行觸發。 例如，分割成三部分的單一邏輯訊息會導致在第一個 MQPUT 及分段時只處理一個觸發事件。 不過，這三個區段中的每一個都會導致在它們透過 IBM MQ 網路移動時處理它們自己的觸發事件。
3. 對於 IBM MQ for z/OS，如果已設定共用佇列來進行觸發，並失去與管理共用佇列之「連結機能」的連線，則可能會產生觸發事件並將訊息放入起始佇列。 即使未將任何訊息放置到原始共用佇列設定來進行觸發，也可能會發生這種情況。 這是由 IXLVECTR 巨集的位元過度指示所造成，如 清單通知向量中所記載。

控制觸發程式事件

您可以使用定義應用程式佇列的部分屬性來控制觸發事件。 此資訊也提供使用觸發類型 :EVERY、FIRST 及 DEPTH 的範例。

您可以啟用及停用觸發，並且可以選取針對觸發事件計數的訊息數目或優先順序。 在 物件屬性中有這些屬性的完整說明。

相關屬性如下:

TriggerControl
使用此屬性可啟用及停用應用程式佇列的觸發。

TriggerMsgPriority
訊息必須具有的優先順序下限，訊息才會計入觸發事件。 如果優先順序小於 *TriggerMsgPriority* 的訊息抵達應用程式佇列，當佇列管理程式決定是否建立觸發訊息時，會忽略該訊息。 如果 *TriggerMsgPriority* 設為零，則所有訊息都會計入觸發事件。

TriggerType
除了觸發類型 NONE (停用觸發，就像將 *TriggerControl* 設為 OFF 一樣) 之外，您還可以使用下列觸發類型來設定佇列觸發事件的靈敏度:

每隔
每次訊息到達應用程式佇列時，即會發生觸發事件。 如果您想要啟動應用程式的多個實例，請使用此類型的觸發程式。

第一個
只有在應用程式佇列上的訊息數從零變更為一時，才會發生觸發事件。 如果您要在第一個訊息到達佇列時啟動服務程式，請使用此類型的觸發程式，繼續直到沒有其他訊息要處理，然後結束。 您必須一律處理佇列，直到它是空的為止。 另請參閱第 735 頁的『觸發類型 FIRST 的特殊案例』。

DEPTH
只有在應用程式佇列上的訊息數達到 **TriggerDepth** 屬性值時，才會發生觸發事件。 此類型觸發的一般用途是在收到一組要求的所有回覆時啟動程式。

依深度觸發: 透過深度觸發，佇列管理程式會在建立觸發訊息之後停用觸發 (使用 *TriggerControl* 屬性)。 發生此情況之後，您的應用程式必須重新啟用觸發本身 (使用 MQSET 呼叫)。

停用觸發的動作不受同步點控制，因此無法藉由取消工作單元來重新啟用觸發。 如果程式取消導致觸發事件的放置要求，或如果程式異常終止，您必須使用 MQSET 呼叫或 ALTER QLOCAL 指令重新啟用觸發。

TriggerDepth
使用依深度觸發時，佇列上導致觸發事件的訊息數。

第 731 頁的『觸發事件的條件』中說明佇列管理程式建立觸發訊息必須滿足的條件。

觸發類型 EVERY 的使用範例

請考量產生汽車保險要求的應用程式。 應用程式可能會將要求訊息傳送給一些保險公司，每次都指定相同的回覆目的地佇列。 它可能會在此回覆目的地佇列上設定 EVERY 類型的觸發程式，因此每次回覆到達時，回覆可能會觸發伺服器實例來處理回覆。

使用觸發類型 FIRST 的範例

假設有一個組織有多個分公司，每一個分公司都會將營業天數的詳細資料傳送給總公司。 他們都在同一時間，在工作日結束時，在總公司有一個處理所有分公司的詳細資料的應用程式。 第一個到達總部的訊息可能導致啟動此應用程式的觸發事件。 此應用程式會繼續處理，直到其佇列中沒有其他訊息為止。

觸發類型 DEPTH 的使用範例

考量旅行社應用程式，它會建立單一要求以確認航班預約、確認旅館房間預約、租車，以及訂購部分旅客檢查。 應用程式可能會將這些項目分隔成四則要求訊息，並將每一則訊息傳送至個別目的地。 它可能會在其回覆目的地佇列上設定類型為 DEPTH 的觸發程式 (深度設為值 4)，因此只有在所有四個回覆都到達時才會重新啟動它。

如果在四個回覆的最後一個之前，有另一個訊息 (可能來自不同的要求) 到達回覆目的地佇列，則會提早觸發要求應用程式。 若要避免此情況，當使用 DEPTH 觸發來收集要求的多個回覆時，請一律針對每一個要求使用新的回覆目的地佇列。

觸發類型 FIRST 的特殊案例

使用觸發類型 FIRST，如果在另一個訊息到達時應用程式佇列上已有訊息，則佇列管理程式通常不會建立另一個觸發訊息。

不過，負責處理佇列的應用程式可能不會實際開啟佇列 (例如，應用程式可能因系統問題而結束)。 如果將不正確的應用程式名稱放入程序定義物件中，負責處理佇列的應用程式將不會挑選任何訊息。 在這些狀況下，如果另一則訊息到達應用程式佇列，則沒有伺服器執行來處理此訊息 (以及佇列上的任何其他訊息)。

為了處理此問題，在下列情況下，佇列管理程式會建立進一步觸發訊息:

- 如果有另一則訊息到達應用程式佇列，但只有在佇列管理程式建立該佇列的最後一則觸發訊息之後，已過了預先定義的時間間隔。 此時間間隔定義在佇列管理程式屬性 *TriggerInterval* 中。 其預設值為 999 999 999 999 毫秒。
- 在 IBM MQ for z/OS 上，會定期掃描名稱為開啟起始佇列的應用程式佇列。 如果自前次傳送觸發訊息以來已經過 *TRIGINT* 毫秒，且佇列滿足觸發事件的條件且 CURDEPTH 大於零，則會產生觸發訊息。 此處理程序稱為 backstop 觸發。

在決定要在應用程式中使用的觸發間隔值時，請考量下列要點:

- 如果您將 *TriggerInterval* 設為低值，且沒有負責處理應用程式佇列的應用程式，則觸發類型 FIRST 的行為可能類似於觸發類型 EVERY。 這取決於將訊息放入應用程式佇列的速率，而應用程式佇列又可能相依於其他系統活動。 這是因為，如果觸發間隔非常小，則每次將訊息放入應用程式佇列時，即使觸發類型是 FIRST (不是 EVERY)，也會產生另一個觸發訊息。 (觸發間隔為零的觸發類型 FIRST 等於觸發類型 EVERY。)
- 在 IBM MQ for z/OS 上，如果您將 *TRIGINT* 設為低值，且沒有應用程式負責處理觸發類型 FIRST 應用程式佇列，則每次定期掃描名稱為開啟起始佇列的應用程式佇列時， backstop 觸發都會產生觸發訊息。
- 如果取消工作單元 (請參閱 觸發訊息及工作單元) 且觸發間隔已設為高值 (或預設值)，當工作單元取消時，會產生一則觸發訊息。 不過，如果您已將觸發間隔設為低值或零 (導致觸發類型 FIRST 行為類似觸發

類型 EVERY)，則可以產生許多觸發訊息。 如果工作單元已取消，則所有觸發訊息仍可使用。 產生的觸發訊息數視觸發間隔而定。 如果觸發間隔設為零，則會產生訊息數目上限。

設計使用觸發佇列的應用程式

您已瞭解如何設定及控制應用程式的觸發。 以下是設計應用程式時要考量的一些提示。

觸發訊息及工作單元

因為觸發事件不是工作單元的一部分而建立的觸發訊息會放在任何工作單元之外的起始佇列中，與任何其他訊息沒有相依關係，且可立即由觸發監視器擷取。

當 UOW 已解決時，不論工作單元是已確定還是已取消，由於觸發事件 (屬於工作單元的一部分) 而建立的觸發訊息可在起始佇列上使用

如果佇列管理程式無法將觸發訊息放置在起始佇列上，則會將它放置在無法傳送郵件 (未遞送訊息) 的佇列上。

註:

1. 當佇列管理程式評量觸發事件的條件是否存在時，會同時計算已確定及未確定的訊息。

   當觸發類型為 FIRST 或 DEPTH 時，即使工作單元已取消，觸發訊息仍可使用，因此在符合必要條件時，觸發訊息一律可用。 例如，針對以觸發類型 FIRST 觸發的佇列，考量工作單元內的放置要求。 這會導致佇列管理程式建立觸發訊息。 如果從另一個工作單元發生另一個放置要求，則不會導致另一個觸發事件，因為應用程式佇列上的訊息數現在已從 1 變更為 2，這不符合觸發事件的條件。 現在，如果第一個工作單元已取消，但第二個工作單元已確定，則仍會建立觸發訊息。

   不過，這表示有時會在未滿足觸發事件的條件時建立觸發訊息。 使用觸發的應用程式必須隨時準備處理此狀況。 建議您將 wait 選項與 MQGET 呼叫搭配使用，並將 *WaitInterval* 設為適當的值。

   不論工作單元是取消或確定，所建立的觸發訊息一律可供使用。
2. 對於本端共用佇列 (亦即，佇列共用群組中的共用佇列)，佇列管理程式只會計算已確定的訊息。

從觸發佇列取得訊息

當您設計使用觸發的應用程式時，請注意在啟動程式的觸發監視器與應用程式佇列上變成可用的其他訊息之間可能會有延遲。 當導致觸發事件的訊息在其他訊息之前已確定時，可能會發生此情況。

若要容許訊息抵達的時間，當您使用 MQGET 呼叫從已設定觸發條件的佇列中移除訊息時，請一律使用等待選項。 *WaitInterval* 必須足夠讓放置訊息與確定該放置呼叫之間的最長合理時間。 如果訊息來自遠端佇列管理程式，則此時間會受到下列影響:

- 在確定之前放置的訊息數
- 通訊鏈結的速度及可用性
- 訊息的大小

如需您應該搭配使用 MQGET 呼叫與等待選項的狀況範例，請考量我們在說明工作單元時所使用的相同範例。 這是以觸發類型 FIRST 觸發之佇列的工作單元內的放置要求。 此事件會導致佇列管理程式建立觸發訊息。 如果發生另一個放置要求，則從另一個工作單元，這不會導致另一個觸發事件，因為應用程式佇列上的訊息數目未從零變更為一。 現在，如果第一個工作單元已取消，但第二個工作單元已確定，則仍會建立觸發訊息。 因此，會在取消第一個工作單元時建立觸發訊息。 如果在確定第二則訊息之前有重大延遲，則觸發的應用程式可能需要等待它。

使用 DEPTH 類型的觸發，即使最終確定所有相關訊息，也可能會發生延遲。 假設 **TriggerDepth** 佇列屬性具有值 2。 當兩個訊息到達佇列時，第二個訊息會導致建立觸發訊息。 不過，如果第二個訊息是第一個要確定的訊息，則此時觸發訊息會變成可用。 觸發監視器會啟動伺服器程式，但在確定第一個訊息之前，程式只能擷取第二個訊息。 因此程式可能需要等待第一個訊息變成可用。

請設計您的應用程式，以便在您的等待間隔到期時，如果沒有訊息可供擷取，則它會終止。 如果稍後有一或多則訊息送達，請依賴正在重新觸發的應用程式來處理它們。 此方法可防止應用程式閒置，以及不必要地使用資源。

觸發監視器的起始佇列處理

對佇列管理程式而言，觸發監視器與處理佇列的任何其他應用程式一樣。 不過，觸發監視器會提供起始佇列。

觸發監視器通常是持續執行的程式。 當觸發訊息到達起始佇列時，觸發監視器會擷取該訊息。 它會使用訊息中的資訊來發出指令，以啟動應用程式來處理應用程式佇列上的訊息。

觸發監視器必須將足夠的資訊傳遞給正在啟動的程式，以便程式可以在正確的應用程式佇列上執行正確的動作。

通道起始程式是訊息通道代理程式之觸發監視器特殊類型的範例。 然而在此狀況下，您必須使用觸發類型 FIRST 或 DEPTH。

ALW *AIX, Linux, and Windows* 系統上的觸發監視器

本主題包含 AIX, Linux, and Windows 系統上提供的觸發監視器相關資訊。

針對伺服器環境提供下列觸發監視器:

amqstrg0
這是一個範例觸發監視器，提供 **runmqtrm** 所提供功能的子集。 如需 amqstrg0 的相關資訊，請參閱 第 888 頁的『在 Multiplatforms 上使用範例程式』 。

runmqtrm
此指令的語法為 **runmqtrm** [*-m QMgrName*] [*-q InitQ*]，其中 QMgrName 是佇列管理程式，而 InitQ 是起始佇列。 預設佇列為 SYSTEM.DEFAULT.INITIATION.QUEUE 。 它會呼叫程式以取得適當的觸發訊息。 此觸發監視器支援預設應用程式類型。

觸發監視器傳遞至作業系統的指令字串建置如下:

1. 來自相關 PROCESS 定義的 *ApplId* (如果已建立)
2. MQTMC2 結構，以雙引號括住
3. 來自相關 PROCESS 定義的 *EnvData* (如果已建立)

其中 *ApplId* 是要執行的程式名稱，因為它將在指令行上輸入。

傳遞的參數是 MQTMC2 字元結構。 所呼叫的指令字串以雙引號括住此字串 (完全如所提供)，以便系統指令接受它作為一個參數。

在完成剛啟動的應用程式之前，觸發監視器不會查看起始佇列上是否有另一則訊息。 如果應用程式有許多處理程序要執行，觸發監視器可能無法跟上到達的觸發訊息數目。 您有兩個選項:

- 有更多觸發監視器在執行中
- 在背景中執行已啟動的應用程式

如果您有多個觸發監視器在執行中，您可以控制任何一次可以執行的應用程式數目上限。 如果您在背景中執行應用程式，則 IBM MQ 對可以執行的應用程式數目沒有任何限制。

Linux AIX To run the started application in the background on AIX and Linux, put an & at the end of the *EnvData* of the PROCESS definition.

若要在 Windows 系統的背景中執行已啟動的應用程式，請在 *ApplId* 欄位內，以 START 指令作為應用程式名稱的字首。 例如:

```
START ?B AMQSECHA
```

註: Windows 如果 Windows 路徑具有空格作為路徑名稱的一部分，則應該以引號 (") 括住這些空格以確保將它當作單一引數來處理。 例如， `"C:\Program Files\Application Directory\Application.exe"`。

以下是 APPLICID 字串的範例，其中檔名包含空格作為路徑的一部分:

```
START "" /B "C:\Program Files\Application Directory\Application.exe"
```

範例中 Windows START 指令的語法包含以雙引號括住的空字串。 START 指定將引號中的第一個引數視為新指令的標題。 若要確保 Windows 不會將應用程式路徑誤以為 'title' 引數，請在指令中的應用程式名稱之前新增以雙引號括住的標題字串。

為 IBM MQ 用戶端提供下列觸發監視器:

runmqtmc
這與 runmqtrm 相同，不同之處在於它與 IBM MQ MQI client 程式庫鏈結。

ALW *CICS* 的觸發監視器

amqltmc0 觸發監視器提供給 CICS。 它的運作方式與標準觸發監視器 runmqtrm 相同，但您以不同的方式執行它，並觸發 CICS 交易。

本主題僅適用於 Windows、 AIX and Linux x86-64 系統。

觸發監視器是以 CICS 程式提供; 請以 4 個字元的交易名稱來定義它。 輸入 4 個字元的名稱以啟動觸發監視器。 它會使用預設佇列管理程式 (如 qm.ini 檔案中所命名，或在 IBM MQ for Windows 上使用登錄) 及 SYSTEM.CICS.INITIATION.QUEUE。

如果您想要使用不同的佇列管理程式或佇列，請建置觸發監視器 MQTMC2 結構: 這需要您使用 EXEC CICS START 呼叫來撰寫程式，因為該結構太長而無法新增為參數。 然後，將 MQTMC2 結構作為資料傳遞至觸發監視器的 START 要求。

當您使用 MQTMC2 結構時，只需要提供 *StrucId*、 *Version*、 *QName* 及 **QMgrName** 參數給觸發監視器，因為它不會參照任何其他欄位。

從起始佇列讀取訊息，並使用 EXEC CICS START (假設觸發訊息中的 APPL_TYPE 是 MQAT_CICS) 來啟動 CICS 交易。 從起始佇列讀取訊息是在 CICS 同步點控制下執行。

當監視器啟動和停止時，以及當發生錯誤時，會產生訊息。 這些訊息會傳送至 CSMT 暫時資料佇列。

表 *129:* 觸發監視器的可用版本.

有兩個直欄的表格。 第一個直欄列出觸發監視器的可用版本，第二個直欄顯示每一個版本用於哪些平台。

版本	使用
amqltmc0	TXSeries 適用於: • AIX AIX • Linux Linux x86-64 系統
amqltmc4	Windows TXSeries for Windows 5.1
amqltmcc	CICS 觸發監視器的用戶端連結版本

如果您需要其他環境的觸發監視器，請撰寫程式來處理佇列管理程式放置在起始佇列上的觸發訊息。 這類程式應該執行下列動作:

1. 使用 MQGET 呼叫來等待訊息到達起始佇列。
2. 請檢查觸發訊息的 MQTM 結構欄位，以尋找要啟動的應用程式名稱，以及它執行所在的環境。
3. 發出環境特定的啟動指令。

   z/OS 例如，在 z/OS 批次上，將工作提交至內部讀取器。
4. 必要的話，將 MQTM 結構轉換為 MQTMC2 結構。
5. 將 MQTMC2 或 MQTM 結構傳遞至已啟動的應用程式。 這可以包含使用者資料。

6. 將要提供該佇列的應用程式與應用程式佇列相關聯。 您可以透過在佇列的 **ProcessName** 屬性中命名程序定義物件 (如果已建立) 來執行此動作。 若要命名程序定義物件，您可以使用 **DEFINE QLOCAL** 或 **ALTER QLOCAL** 指令。

   IBM i 在 IBM i 上，您也可以使用 CRTMQMQ 或 CHGMQMQ。

如需觸發監視器介面的相關資訊，請參閱 MQTMC2。

IBM i *IBM i 上的觸發監視器*

在 IBM i 上，請使用 IBM MQ for IBM i CL 指令 **STRMQMTRM**，而非 **runmqtrm** 控制指令。

使用 STRMQMTRM 指令，如下所示:

```
STRMQMTRM INITQNAME(InitQ) MQMNAME(QMgrName)
```

詳細資料適用於 runmqtrm。

也會提供下列範例程式，您可以使用這些程式作為模型來撰寫您自己的觸發監視器:

AMQSTRG4
此觸發監視器會針對要啟動的處理程序提交 IBM i 工作，但這表示存在與每一個觸發訊息相關聯的其他處理程序。

AMQSERV4
這是觸發程式伺服器。 對於每一個觸發訊息，此伺服器會在其自己的工作中執行處理程序的指令，並且可以呼叫 CICS 交易。

觸發監視器和觸發伺服器都會將 MQTMC2 結構傳遞給它們啟動的程式。 如需此結構的說明，請參閱 MQTMC2。 這兩個範例都以來源及執行檔形式遞送。

因為這些觸發監視器只能呼叫原生 IBM i 程式，所以它們無法直接觸發 Java 程式，因為 Java 類別位於 IFS 中。 不過， Java 程式可以透過觸發 CL 程式來間接觸發，該 CL 程式隨後會呼叫 Java 程式並通過 TMC2 結構。 TMC2 結構的大小下限為 732 個位元組。

以下是範例 CLP 的來源:

```
PGM PARM(&TMC2)
  DCL &TMC2 *CHAR LEN(800)
    ADDENVVAR ENVVAR(TM) VALUE(&TMC2)
    QSH  CMD('java_pgmname $TM')
    RMVENVVAR ENVVAR(TM)
ENDPGM
```

下列觸發監視器程式提供給 IBM MQ MQI client: RUNMQTMC

呼叫 RUNMQTMC ，如下所示:

```
CALL PGM(QMQM/RUNMQTMC) PARM('-m' QMgrName '-q' InitQ)
```

觸發訊息的內容

下列主題說明觸發訊息的部分其他內容。

- 第 739 頁的『觸發訊息的持續性及優先順序』
- 第 740 頁的『佇列管理程式重新啟動及觸發訊息』
- 第 740 頁的『觸發訊息及物件屬性的變更』
- 第 740 頁的『觸發訊息的格式』

觸發訊息的持續性及優先順序

觸發訊息不會持續存在，因為它們不需要持續存在。

不過，產生觸發事件的條件會持續保存，因此只要符合這些條件，就會產生觸發訊息。 如果觸發訊息遺失，應用程式佇列上的應用程式訊息繼續存在，可保證只要符合所有條件，佇列管理程式就會產生觸發訊息。

如果回復工作單元，則一律會遞送它所產生的任何觸發訊息。

觸發訊息採用起始佇列的預設優先順序。

佇列管理程式重新啟動及觸發訊息

在重新啟動佇列管理程式之後，當下次開啟起始佇列以供輸入時，如果與此起始佇列相關聯的應用程式佇列中有訊息，則觸發訊息可以放置在此起始佇列中，並定義用於觸發。

觸發訊息及物件屬性的變更

根據觸發事件發生時生效的觸發屬性值來建立觸發訊息。

如果觸發訊息直到稍後才可供觸發監視器使用 (因為導致產生它的訊息已放在工作單元內)，則在此期間對觸發屬性所做的任何變更都不會影響觸發訊息。 尤其，停用觸發並不會阻止觸發訊息在建立之後變成可用。 此外，在觸發訊息可供使用時，應用程式佇列可能不再存在。

觸發訊息的格式

觸發訊息的格式由 MQTM 結構定義。

這具有下列欄位，當佇列管理程式建立觸發訊息時，會使用應用程式佇列及與該佇列相關聯之處理程序的物件定義中的資訊來填入這些欄位:

StrucId
結構 ID。

Version
結構的版本。

QName
發生觸發事件的應用程式佇列名稱。 當佇列管理程式建立觸發訊息時，它會使用應用程式佇列的 **QName** 屬性填入此欄位。

ProcessName
與應用程式佇列相關聯的程序定義物件名稱。 當佇列管理程式建立觸發訊息時，它會使用應用程式佇列的 **ProcessName** 屬性填入此欄位。

TriggerData
由觸發監視器使用的任意格式欄位。 當佇列管理程式建立觸發訊息時，它會使用應用程式佇列的 **TriggerData** 屬性填入此欄位。 在 IBM MQ for z/OS 以外的任何 IBM MQ 產品上，此欄位可用來指定要觸發的通道名稱。

ApplType
觸發監視器要啟動的應用程式類型。 當佇列管理程式建立觸發訊息時，它會使用 *ProcessName* 中所識別程序定義物件的 **ApplType** 屬性來填寫此欄位。

ApplId
識別觸發監視器要啟動之應用程式的字串。 當佇列管理程式建立觸發訊息時，它會使用 *ProcessName* 中所識別程序定義物件的 **ApplId** 屬性來填寫此欄位。

當您使用 CICS 所提供的觸發監視器 CKTI 時，程序定義物件的 **ApplId** 屬性是 CICS 交易 ID。

當您使用 IBM MQ for z/OS 提供的 CSQQTRMN 時，程序定義物件的 **ApplId** 屬性是 IMS 交易 ID。

EnvData
包含環境相關資料供觸發監視器使用的字元欄位。 當佇列管理程式建立觸發訊息時，它會使用 *ProcessName* 中所識別程序定義物件的 **EnvData** 屬性來填寫此欄位。 CICS 提供的觸發監視器 (CKTI) 或 IBM MQ for z/OS 提供的觸發監視器 (CSQQTRMN) 不使用此欄位，但其他觸發監視器可能會選擇使用它。

UserData
包含使用者資料供觸發監視器使用的字元欄位。 當佇列管理程式建立觸發訊息時，它會使用 *ProcessName* 中所識別程序定義物件的 **UserData** 屬性來填寫此欄位。 此欄位可用來指定要觸發的通道名稱。

MQTM 中有觸發訊息結構的完整說明。

當觸發無法運作時

如果觸發監視器無法啟動程式或佇列管理程式無法遞送觸發訊息，則不會觸發程式。 例如，處理程序物件中的 applid 必須指定要在背景中啟動程式; 否則，觸發監視器無法啟動程式。

如果已建立觸發訊息，但無法放置在起始佇列上 (例如，因為佇列已滿，或觸發訊息的長度大於指定給起始佇列的訊息長度上限) ，則會將觸發訊息放置在無法傳送的郵件 (無法遞送的訊息) 佇列上。

如果無法順利完成對無法傳送郵件之佇列的放置作業，則會捨棄觸發訊息，並將警告訊息 z/OS 傳送至 z/OS 主控台或 給系統操作員，或放置在錯誤日誌中。

將觸發訊息放置在無法傳送郵件的佇列上可能會產生該佇列的觸發訊息。 如果將訊息新增至無法傳送郵件的佇列，則會捨棄此第二個觸發訊息。

如果程式順利觸發，但在接收來自佇列的訊息之前異常終止，請使用追蹤公用程式 (例如， CICS AUXTRACE ，如果程式在 CICS 下執行) 找出失敗原因。

使用 MQI 及叢集

對於與叢集作業相關的呼叫和回覆碼，有一些特殊選項。

請使用下列鏈結，以進一步瞭解與叢集搭配使用的呼叫及回覆碼上可用的選項:

- 第 742 頁的『MQOPEN 及叢集』
- 第 743 頁的『MQPUT、 MQPUT1 及叢集』
- 第 743 頁的『MQINQ 及叢集』
- 第 744 頁的『MQSET 及叢集』
- 第 744 頁的『回覆碼』

相關概念

第 604 頁的『訊息佇列介面概觀』
瞭解「訊息佇列介面 (MQI)」元件。

第 616 頁的『連接至佇列管理程式並切斷與佇列管理程式的連線』
若要使用 IBM MQ 程式設計服務，程式必須具有與佇列管理程式的連線。 使用此資訊來瞭解如何連接至佇列管理程式，以及與佇列管理程式中斷連線。

第 621 頁的『開啟及關閉物件』
此資訊提供開啟及關閉 IBM MQ 物件的見解。

第 630 頁的『將訊息放置在佇列上』
使用此資訊來瞭解如何將訊息放置在佇列上。

第 643 頁的『從佇列取得訊息』
使用此資訊來瞭解從佇列取得訊息的相關資訊。

第 713 頁的『查詢及設定物件屬性』
屬性是定義 IBM MQ 物件性質的內容。

第 716 頁的『確定及取消工作單元』
此資訊說明如何確定及取消工作單元中發生的任何可回復取得及放置作業。

第 725 頁的『使用觸發程式啟動 IBM MQ 應用程式』
瞭解觸發程式以及如何使用觸發程式來啟動 IBM MQ 應用程式。

第 745 頁的『在 IBM MQ for z/OS 上使用及撰寫應用程式』
IBM MQ for z/OS 應用程式可以由在許多不同環境中執行的程式組成。 這表示他們可以利用多個環境中可用的設施。

第 57 頁的『IBM MQ for z/OS 上的 IMS 及 IMS 橋接器應用程式』
此資訊可協助您使用 IBM MQ 撰寫 IMS 應用程式。

MQOPEN 及叢集

開啟叢集佇列時將訊息放入或從中讀取的佇列，取決於 MQOPEN 呼叫。

選取目標佇列

如果您未在物件描述子 MQOD 中提供佇列管理程式名稱，則佇列管理程式會選取要將訊息傳送至其中的佇列管理程式。 如果您在物件描述子中提供佇列管理程式名稱，則訊息一律會傳送至您選取的佇列管理程式。

如果佇列管理程式正在選取目標佇列管理程式，則此選項取決於連結選項 MQOO_BIND_* 以及本端佇列是否存在。 如果有佇列的本端實例，除非 CLWLUSEQ 屬性設為 ANY，否則它一律會優先於遠端實例開啟。 否則，選項取決於連結選項。 當搭配使用 訊息群組 與叢集時，必須指定 MQOO_BIND_ON_OPEN 或 MQOO_BIND_ON_GROUP ，以確保在相同目的地處理群組中的所有訊息。

如果佇列管理程式正在選取目標佇列管理程式，則會使用工作量管理演算法以循環式方式執行此動作; 請參閱 叢集中的工作量平衡。

使用工作量平衡演算法的時間取決於開啟叢集佇列的方式:

- MQOO_BIND_ON_OPEN -在應用程式開啟佇列時使用演算法一次。
- MQOO_BIND_NOT_FIXED -此演算法用於放入佇列的每一則訊息。
- MQOO_BIND_ON_GROUP -在每一個訊息群組開始時使用一次演算法。

MQOO_BIND_ON_OPEN
MQOPEN 呼叫上的 MQOO_BIND_ON_OPEN 選項指定要修正目標佇列管理程式。 如果叢集內有相同佇列的多個實例，請使用 MQOO_BIND_ON_OPEN 選項。 所有放入佇列 (指定從 MQOPEN 呼叫傳回的物件控點) 的訊息，都會導向至相同的佇列管理程式。

- 如果訊息有親緣性，請使用 MQOO_BIND_ON_OPEN 選項。 例如，如果訊息批次全部由相同的佇列管理程式處理，請在開啟佇列時指定 MQOO_BIND_ON_OPEN 。 IBM MQ 會修正佇列管理程式，以及放入該佇列的所有訊息所採用的路徑。
- 如果指定 MQOO_BIND_ON_OPEN 選項，則必須重新開啟佇列，才能選取佇列的新實例。

MQOO_BIND_NOT_FIXED
MQOPEN 呼叫上的 MQOO_BIND_NOT_FIXED 選項指定未修正目標佇列管理程式。 寫入佇列 (指定從 MQOPEN 呼叫傳回的物件控點) 的訊息，會在 MQPUT 以逐訊息為基礎，遞送至佇列管理程式。 如果您不想要強制將所有訊息寫入相同的目的地，請使用 MQOO_BIND_NOT_FIXED 選項。

- 請勿同時指定 MQOO_BIND_NOT_FIXED 和 MQMF_SEGMENTATION_ALLOWED 。 如果您這麼做，則訊息區段可能會遞送至分散在整個叢集中的不同佇列管理程式。

MQOO_BIND_ON_GROUP
容許應用程式要求將訊息群組配置給相同的目的地實例。 此選項僅適用於佇列，且僅影響叢集佇列。 如果指定給非叢集佇列的佇列，則會忽略該選項。

- 只有在 MQPUT 上指定 MQPMO_LOGICAL_ORDER 時，群組才會遞送至單一目的地。 當指定 MQOO_BIND_ON_GROUP ，但訊息不是邏輯群組的一部分時，會改用 BIND_NOT_FIXED 行為。

MQOO_BIND_AS_Q_DEF
如果您未指定 MQOO_BIND_ON_OPEN、 MQOO_BIND_NOT_FIXED 或 MQOO_BIND_ON_GROUP，則預設選項為 MQOO_BIND_AS_Q_DEF。 使用 MQOO_BIND_AS_Q_DEF 會導致從 DefBind 佇列屬性取得用於佇列控點的連結。

MQOPEN 選項的相關性

MQOPEN 選項 MQOO_BROWSE， MQOO_INPUT_*或 MQOO_SET 需要叢集佇列的本端實例， MQOPEN 才能成功。

MQOPEN 選項 MQOO_OUTPUT、 MQOO_BIND_*或 MQOO_INQUIRE 不需要叢集佇列的本端實例即可成功。

解析的佇列管理程式名稱

在 MQOPEN 時間解析佇列管理程式名稱時，會將解析的名稱傳回給應用程式。 如果應用程式在後續的 MQOPEN 呼叫中嘗試使用此名稱，則可能會發現它未獲授權存取該名稱。

MQPUT、 *MQPUT1* 及叢集

如果在 MQOPEN 上指定 MQOO_BIND_NOT_FIXED，則工作量管理常式會選擇 MQPUT 或 MQPUT1 選取哪個目的地。

如果在 MQOPEN 呼叫上指定 MQOO_BIND_NOT_FIXED，則每一個後續的 MQPUT 呼叫都會呼叫工作量管理常式，以決定要將訊息傳送至哪個佇列管理程式。 按訊息逐一選取要採用的目的地及路徑。 如果網路中的條件變更，則在放置訊息之後，目的地及路徑可能會變更。 MQPUT1 呼叫一律會像 MQOO_BIND_NOT_FIXED 一樣運作，也就是說，它一律會呼叫工作量管理常式。

當工作量管理常式已選取佇列管理程式時，本端佇列管理程式會完成放置作業。 訊息可以放置在不同的佇列上:

1. 如果目的地是佇列的本端實例，則訊息會放在本端佇列上。
2. 如果目的地是叢集中的佇列管理程式，則訊息會放置在叢集傳輸佇列上。
3. 如果目的地是叢集外部的佇列管理程式，則會將訊息放置在與目標佇列管理程式同名的傳輸佇列上。

如果在 MQOPEN 呼叫上指定 MQOO_BIND_ON_OPEN，則 MQPUT 呼叫不會呼叫工作量管理常式，因為已選取目的地及路徑。

MQINQ 及叢集

查詢哪個叢集佇列取決於您與 MQOO_INQUIRE 結合的選項。

在查詢佇列之前，請先使用 MQOPEN 呼叫來開啟它，並指定 MQOO_INQUIRE。

若要查詢叢集佇列，請使用 MQOPEN 呼叫並結合其他選項與 MQOO_INQUIRE。 可以查詢的屬性取決於是否有叢集佇列的本端實例，以及開啟佇列的方式:

- 結合 MQOO_BROWSE、 MQOO_INPUT_*或 MQOO_SET 與 MQOO_INQUIRE 需要叢集佇列的本端實例，才能順利開啟。 在此情況下，您可以查詢適用於本端佇列的所有屬性。
- 結合 MQOO_OUTPUT 與 MQOO_INQUIRE，且未指定上述任何選項，所開啟的實例為:
  - 本端佇列管理程式上的實例 (如果有的話)。 在此情況下，您可以查詢適用於本端佇列的所有屬性。
  - 叢集中其他位置的實例 (如果沒有本端佇列管理程式實例的話)。 在此情況下，只能查詢下列屬性。 在此情況下， QType 屬性具有值 MQQT_CLUSTER 。
    - DefBind
    - DefPersistence
    - DefPriority
    - InhibitPut
    - QDesc
    - **完整名稱**
    - QTYPE

若要查詢叢集佇列的 DefBind 屬性，請搭配使用 MQINQ 呼叫與選取元 MQIA_DEF_BIND。 傳回的值為 MQBND_BIND_ON_OPEN 或 MQBND_BIND_NOT_FIXED 或 MQBND_BIND_ON_GROUP。 搭配使用群組與叢集時，必須指定 MQBND_BIND_ON_OPEN 或 MQBND_BIND_ON_GROUP 。

若要查詢佇列本端實例的 CLUSTER 及 CLUSNL 屬性，請搭配使用 MQINQ 呼叫與選取元 MQCA_CLUSTER_NAME 或選取元 MQCA_CLUSTER_NAMELIST。

註: 如果您開啟叢集佇列而未修正 MQOPEN 所連結的佇列，則後續 MQINQ 呼叫可能會查詢叢集佇列的不同實例。

相關概念

第 626 頁的『叢集佇列的 MQOPEN 選項』

用於佇列控點的連結取自 **DefBind** 佇列屬性，其可以採用值 MQBND_BIND_ON_OPEN、MQBND_BIND_NOT_FIXED 或 MQBND_BIND_ON_GROUP。

MQSET 及叢集

MQOPEN option MQOO_SET 選項需要有叢集佇列的本端實例， MQSET 才會成功。

您無法使用 MQSET 呼叫來設定叢集中其他位置的佇列屬性。

您可以開啟以叢集屬性定義的本端別名或遠端佇列，並使用 MQSET 呼叫。 您可以設定本端別名或遠端佇列的屬性。 目標佇列是否為在不同佇列管理程式上定義的叢集佇列並不重要。

回覆碼

叢集特定的回覆碼

MQRC_CLUSTER_EXIT_ERROR (2266 X'8DA')

發出 MQOPEN、 MQPUT 或 MQPUT1 呼叫，以開啟叢集佇列或在其中放置訊息。 佇列管理程式的 ClusterWorkloadExit 屬性所定義的叢集工作量結束程式非預期地失敗或未及時回應。

IBM MQ for z/OS 上的系統日誌中會寫入一則訊息，提供此錯誤的相關資訊。

處理此佇列控點的後續 MQOPEN、 MQPUT 及 MQPUT1 呼叫，就像 ClusterWorkloadExit 屬性為空白一樣。

MQRC_CLUSTER_EXIT_LOAD_ERROR (2267 X'8DB')

在 z/OS 上，無法載入叢集工作量結束程式。

訊息會寫入系統日誌，並繼續執行處理程序，好像 ClusterWorkloadExit 屬性是空白一樣。

Multi 在 多平台上，會發出 MQCONN 或 MQCONNX 呼叫，以連接至佇列管理程式。 呼叫失敗，因為無法載入佇列管理程式的佇列管理程式 ClusterWorkloadExit 屬性所定義的叢集工作量結束程式。

MQRC_CLUSTER_PUT_INHIBITED (2268 X'8DC')

針對叢集佇列發出具有有效 MQOO_OUTPUT 及 MQOO_BIND_ON_OPEN 選項的 MQOPEN 呼叫。 透過將 InhibitPut 屬性設為 MQQA_PUT_INHIBITED，目前禁止放置叢集中佇列的所有實例。 因為沒有佇列實例可用來接收訊息，所以 MQOPEN 呼叫失敗。

只有在下列兩個陳述式都成立時，才會出現此原因碼:

- 沒有佇列的本端實例。 如果有本端實例，則即使本端實例禁止放置， MQOPEN 呼叫也會成功。
- 沒有佇列的叢集工作量結束程式，或有叢集工作量結束程式，但未選擇佇列實例。 (如果叢集工作量結束程式選擇佇列實例，則即使該實例禁止放置， MQOPEN 呼叫仍會成功。)

如果在 MQOPEN 呼叫上指定 MQOO_BIND_NOT_FIXED 選項，則即使叢集中的所有佇列都禁止放置，呼叫也會成功。 不過，如果在該呼叫時仍禁止放置所有佇列，則後續的 MQPUT 呼叫可能會失敗。

MQRC_CLUSTER_RESOLUTION_ERROR (2189 X'88D')

1. 發出 MQOPEN、 MQPUT 或 MQPUT1 呼叫，以開啟叢集佇列或在其中放置訊息。 無法正確解析佇列定義，因為需要來自完整儲存庫佇列管理程式的回應，但沒有可用的回應。
2. 對主題物件發出 MQOPEN、 MQPUT、 MQPUT1 或 MQSUB 呼叫，並指定 PUBSCOPE (ALL) 或 SUBSCOPE (ALL)。 無法正確解析叢集主題定義，因為需要來自完整儲存庫佇列管理程式的回應，但沒有可用的回應。

MQRC_CLUSTER_RESOURCE_ERROR (2269 X'8DD')

針對叢集佇列發出 MQOPEN、 MQPUT 或 MQPUT1 呼叫。 嘗試使用叢集作業所需的資源時發生錯誤。

MQRC_NO_DESTINATIONS_AVAILABLE (2270 X'8DE')

發出 MQPUT 或 MQPUT1 呼叫，以將訊息放置在叢集佇列上。 在呼叫時，叢集中不再有任何佇列實例。 MQPUT 會失敗，且不會傳送訊息。

如果在開啟佇列的 MQOPEN 呼叫上指定 MQOO_BIND_NOT_FIXED，或使用 MQPUT1 來放置訊息，則可能會發生錯誤。

MQRC_STOPPED_BY_CLUSTER_EXIT (2188 X'88C')

發出 MQOPEN、 MQPUT 或 MQPUT1 呼叫，以在叢集佇列上開啟或放置訊息。 叢集工作量結束程式拒絕呼叫。

z/OS 在 IBM MQ for z/OS 上使用及撰寫應用程式

IBM MQ for z/OS 應用程式可以由在許多不同環境中執行的程式組成。 這表示他們可以利用多個環境中可用的設施。

本資訊說明在每一個受支援環境中執行的程式所能使用的 IBM MQ 機能。 另外，

- 如需使用 IBM MQ-CICS bridge 的相關資訊，請參閱 搭配使用 IBM MQ 與 CICS 。
- 如需使用 IMS 及 IMS 橋接器的相關資訊，請參閱 第 57 頁的『IBM MQ for z/OS 上的 IMS 及 IMS 橋接器應用程式』。

請使用下列鏈結，以進一步瞭解在 IBM MQ for z/OS 上使用及撰寫應用程式的相關資訊:

- 第 745 頁的『環境相依 IBM MQ for z/OS 函數』
- 第 746 頁的『除錯機能、同步點支援及回復支援』
- 第 747 頁的『IBM MQ for z/OS 介面與應用程式環境』
- 第 748 頁的『撰寫 z/OS UNIX System Services 應用程式』
- 第 751 頁的『使用共用佇列進行應用程式設計』

相關概念

第 604 頁的『訊息佇列介面概觀』
瞭解「訊息佇列介面 (MQI)」元件。

第 616 頁的『連接至佇列管理程式並切斷與佇列管理程式的連線』
若要使用 IBM MQ 程式設計服務，程式必須具有與佇列管理程式的連線。 使用此資訊來瞭解如何連接至佇列管理程式，以及與佇列管理程式中斷連線。

第 621 頁的『開啟及關閉物件』
此資訊提供開啟及關閉 IBM MQ 物件的見解。

第 630 頁的『將訊息放置在佇列上』
使用此資訊來瞭解如何將訊息放置在佇列上。

第 643 頁的『從佇列取得訊息』
使用此資訊來瞭解從佇列取得訊息的相關資訊。

第 713 頁的『查詢及設定物件屬性』
屬性是定義 IBM MQ 物件性質的內容。

第 716 頁的『確定及取消工作單元』
此資訊說明如何確定及取消工作單元中發生的任何可回復取得及放置作業。

第 725 頁的『使用觸發程式啟動 IBM MQ 應用程式』
瞭解觸發程式以及如何使用觸發程式來啟動 IBM MQ 應用程式。

第 741 頁的『使用 MQI 及叢集』
對於與叢集作業相關的呼叫和回覆碼，有一些特殊選項。

第 57 頁的『IBM MQ for z/OS 上的 IMS 及 IMS 橋接器應用程式』
此資訊可協助您使用 IBM MQ 撰寫 IMS 應用程式。

z/OS 環境相依 *IBM MQ for z/OS* 函數

當考量 IBM MQ for z/OS 函數時，請使用此資訊。

在執行 IBM MQ for z/OS 的環境中， IBM MQ 函數之間要考量的主要差異如下:

- IBM MQ for z/OS 提供下列觸發監視器:

- 在 CICS 環境中使用的 CKTI
- 在 IMS 環境中使用的 CSQQTRMN

您必須撰寫自己的模組，才能在其他環境中啟動應用程式。

- 在 CICS 和 IMS 環境中支援使用兩階段確定來同步。 在 z/OS 批次環境中，也支援使用交易管理及可回復的資源管理程式服務 (RRS)。 IBM MQ 本身在 z/OS 環境中支援單階段確定。
- 對於批次和 IMS 環境， MQI 提供呼叫來連接程式與佇列管理程式，以及切斷程式與佇列管理程式的連線。 程式可以連接至多個佇列管理程式。
- CICS 系統只能連接至一個佇列管理程式。 如果在 CICS 系統啟動工作中定義子系統名稱，則會在起始 CICS 時發生此情況。 在 CICS 環境中可容忍 MQI 連接及中斷連線呼叫，但不會有任何影響。
- API 交互結束程式可讓程式介入所有 MQI 呼叫的處理程序。 此結束程式僅在 CICS 環境中可用。
- 在多處理器系統上的 CICS 中，由於 MQI 呼叫可以在多個 z/OS TCB 下執行，因此會獲得一些效能優勢。

如需相關資訊，請參閱 在 z/OS 上規劃 *IBM MQ for z/OS* 概念與規劃手冊。

這些特性在 第 746 頁的表 130 中彙總。

表 *130: z/OS* 環境特性

	CICS	**IMS**	**批次 /TSO**
提供的觸發監視器	是	是	否
兩階段確定	是	是	是
一段式確定 (single-phase commit)	是	否	是
連接/中斷連接 MQI 呼叫	容忍	是	是
API 交叉結束程式 (API-crossing exit)	是	否	否

註: 在使用 RRS 的 Batch/TSO 環境中支援兩階段確定。

z/OS 除錯機能、同步點支援及回復支援

使用此資訊來瞭解程式除錯機能、同步點支援及回復支援。

程式除錯機能

IBM MQ for z/OS 提供追蹤機能，您可以用來對所有環境中的程式進行除錯。

此外，在 CICS 環境中，您可以使用:

- CICS 執行診斷機能 (CEDF)
- CICS 追蹤控制交易 (CETR)
- IBM MQ for z/OS API 交互結束程式

在 z/OS 平台上，您可以使用您所使用的程式設計語言所支援的任何可用的互動式除錯工具。

同步點支援

在交易處理環境中，必須同步化工作單元的開始和結束，才能安全使用交易處理。

在 CICS 和 IMS 環境中， IBM MQ for z/OS 完全支援這一點。 完整支援是指資源管理程式之間的合作，以便在 CICS 或 IMS 的控制下，一致地確定或取消工作單元。 資源管理程式的範例有 Db2、 CICS 檔案控制、 IMS 和 IBM MQ for z/OS。

z/OS 批次應用程式可以使用 IBM MQ for z/OS 呼叫來提供一段式確定機能。 這表示應用程式定義的一組佇列作業可以確定或取消，而不需要參照其他資源管理程式。

在使用交易管理及可回復資源管理程式服務 (RRS) 的 z/OS 批次環境中，也支援兩階段確定。 如需進一步資訊，請參閱 z/OS 批次應用程式中的同步點。

回復支援

如果在交易期間，佇列管理程式與 CICS 或 IMS 系統之間的連線中斷，則部分工作單元可能無法順利取消。

不過，當重新建立與 CICS 或 IMS 系統的連線時，佇列管理程式 (在同步點管理程式的控制下) 會解決這些工作單元。

z/OS *IBM MQ for z/OS* 介面與應用程式環境

為了容許在不同環境中執行的應用程式透過訊息佇列作業網路來傳送及接收訊息， IBM MQ for z/OS 為其支援的每一個環境提供 配接器 。

這些配接器是應用程式與 IBM MQ for z/OS 子系統之間的介面。 它們容許程式使用 MQI。

z/OS 批次配接器

使用此資訊來瞭解批次配接器及其支援的確定通訊協定。

批次配接器 為在中執行的程式提供對 IBM MQ for z/OS 資源的存取權:

- 作業 (TCB) 模式
- 問題或監督者狀態
- 主要位址空間控制模式

程式不得處於跨記憶體模式。

應用程式與 IBM MQ for z/OS 之間的連線位於作業層次。 配接器提供從應用程式作業控制區塊 (TCB) 到 IBM MQ for z/OS 的單一連線執行緒。

對於對 IBM MQ for z/OS 所擁有的資源所做的變更，配接器支援單階段確定通訊協定; 它不支援多階段確定通訊協定。

z/OS *RRS* 批次配接卡

請利用這項資訊來瞭解 IBM MQ 所提供的 RRS 批次配接卡和兩個 RRS 批次配接卡。

交易管理及可回復資源管理程式服務 (RRS) 配接器:

- 使用 z/OS RRS 進行確定控制。
- 支援從單一作業同時連線至在單一 z/OS 實例上執行的多個 IBM MQ 子系統。
- 針對透過 z/OS RRS 相容的可回復管理程式存取的可回復資源，提供 z/OS 範圍內的協調確定控制 (使用 z/OS RRS):
  - 使用 RRS 批次配接器連接至 IBM MQ 的應用程式。
  - Db2-在 z/OS 上由工作量管理程式 (WLM) 管理的 Db2 儲存程序位址空間中執行的儲存程序。
- 支援在 TCB 之間切換 IBM MQ 批次執行緒的能力。

IBM MQ for z/OS 提供兩個 RRS 批次配接卡:

CSQBRSTB
在 IBM MQ 應用程式中，此配接器需要您將任何 MQCMIT 陳述式變更為 SRRCMIT ， 並將任何 MQBACK 陳述式變更為 SRRBACK。 (如果您在與 CSQBRSTB 鏈結的應用程式中撰寫 MQCMIT 或 MQBACK ， 則會收到 MQRC_ENVIRONMENT_ERROR。)

CSQBRRSI
此配接器容許 IBM MQ 應用程式使用 MQCMIT 和 MQBACK 或 SRRCMIT 和 SRRBACK。

註: CSQBRSTB 和 CSQBRRSI 隨附鏈結屬性 AMODE (31) RMODE (ANY)。 如果您的應用程式載入低於 16 MB 線的 Stub ， 請先以 RMODE (24) 重新鏈結 Stub。

移轉

您可以移轉現有的 Batch/TSO IBM MQ 應用程式，以使用 RRS 協調來進行少量變更或不進行任何變更。

如果您使用 CSQBRRSI 配接器、MQCMIT 和 MQBACK 來鏈結編輯 IBM MQ 應用程式，請跨 IBM MQ 和所有其他啟用 RRS 的資源管理程式，將您的工作單元同步化。 如果您使用 CSQBRSTB 配接器鏈結編輯 IBM MQ

應用程式，請將 MQCMIT 變更為 SRRCMIT ，並將 MQBACK 變更為 SRRBACK。 後者是較好的方法; 它明確指出同步點不限於 IBM MQ 資源。

z/OS *IMS* 配接器

如果您是從 IBM MQ for z/OS 系統使用 IMS 配接卡，請確保 IMS 可以取得足夠的儲存體，以容納最長 100 MB 的訊息。

使用者注意事項

IMS 配接器 可讓您存取下列項目的 IBM MQ for z/OS 資源:

- 線上訊息處理程式 (MPP)
- 互動式捷徑程式 (IFP)
- 批次訊息處理程式 (BMP)

若要使用這些資源，程式必須以作業 (TCB) 模式及問題狀態執行; 它們不得處於跨記憶體模式或存取暫存器模式。

配接器提供從應用程式作業控制區塊 (TCB) 到 IBM MQ 的連線執行緒。 對於對 IBM MQ for z/OS 所擁有的資源所做的變更，配接器支援兩段式確定通訊協定，並以 IMS 作為同步點協調程式。

此配接卡也提供觸發監視器程式，當符合佇列上的特定觸發條件時，可自動啟動程式。 如需相關資訊，請參閱 第 725 頁的『使用觸發程式啟動 IBM MQ 應用程式』。

如果您要撰寫批次 DL/I 程式，請遵循本主題中針對 z/OS 批次程式提供的指引。

z/OS 撰寫 *z/OS UNIX System Services* 應用程式

批次配接器支援來自批次和 TSO 位址空間的佇列管理程式連線。

對於批次位址空間，配接器支援來自該位址空間內多個 TCB 的連線，如下所示:

- 每一個 TCB 都可以使用 MQCONN 或 MQCONNX 呼叫來連接至多個佇列管理程式 (但 TCB 一次只能有一個特定佇列管理程式的連線實例)。
- 多個 TCB 可以連接至相同的佇列管理程式 (但任何 MQCONN 或 MQCONNX 呼叫所傳回的佇列管理程式控點都連結至發出 TCB ，且不能由任何其他 TCB 使用)。

z/OS UNIX System Services 支援兩種類型的 pthread_create 呼叫:

1. 重量級執行緒，針對每一個 TCB 執行一個，即 ATTACHed 和 DETACHed 在執行緒開始和結束時由 z/OS 執行。
2. 中等加權執行緒，針對每一個 TCB 執行一個，但 TCB 可以是長時間執行 TCB 的其中一個儲存區。 應用程式必須執行所有必要的應用程式清理，因為如果它連接至伺服器，則伺服器在作業 (TCB) 終止時可能提供的預設執行緒終止 **不會** 一律由驅動。

不支援輕量型執行緒。 (如果應用程式建立永久執行緒來分派自己的工作要求， **應用程式** 會負責在啟動下一個工作要求之前清除任何資源。)

IBM MQ for z/OS 支援使用「批次配接器」的 z/OS UNIX System Services 執行緒，如下所示:

1. 完全支援重量級執行緒作為批次連線。 每一個執行緒在其自己的 TCB 中執行，而 TCB 在執行緒啟動及結束時連接及分離。 如果執行緒在發出 MQDISC 呼叫之前結束， IBM MQ for z/OS 會執行其標準作業清理，包括在執行緒正常終止時確定任何未完成的工作單元，或在執行緒異常終止時取消它。
2. 完全支援中加權執行緒，但如果另一個執行緒要重複使用 TCB ，則應用程式必須確定在下次啟動執行緒之前發出 MQDISC 呼叫 (前面有 MQCMIT 或 MQBACK)。 這意味著如果應用程式已建立「程式岔斷處理程式」，且應用程式隨後異常終止，則「岔斷處理程式」必須先發出 MQCMIT 及 MQDISC 呼叫，然後再對另一個執行緒重複使用 TCB。

註: 執行緒作業模型 **不** 支援從多個執行緒存取一般 IBM MQ 資源。

z/OS *z/OS* 的 *API* 交互結束程式

本主題包含產品相關程式設計介面資訊。

結束程式是 IBM 提供的程式碼中的一個點，您可以在其中執行自己的程式碼。 IBM MQ for z/OS 提供 跨 *API* 結束程式 ，您可以用來截取對 MQI 的呼叫，以及監視或修改 MQI 呼叫的功能。 本節說明如何使用 API 交互結束程式，並說明 IBM MQ for z/OS 隨附的範例結束程式。

本節僅適用於 CICS TS V3.1 及更早版本的使用者。 CICS TS V3.2 以及更新版本的使用者應該參閱 CICS 產品說明文件中的 CICS 與 IBM MQ 整合一節。

附註

只有 IBM MQ for z/OS 的 CICS 配接器才會呼叫 API 交互結束程式。 結束程式在 CICS 位址空間中執行。

z/OS 撰寫您自己的結束程式

您可以使用隨 IBM MQ for z/OS 提供的範例 API 交互結束程式 (CSQCAPX) 作為您自己程式的架構。

這說明於第 750 頁的『範例 API 交互結束程式 CSQCAPX』。

撰寫結束程式時，若要尋找應用程式發出的 MQI 呼叫名稱，請檢查 MQXP 結構的 *ExitCommand* 欄位。 若要尋找呼叫中的參數數目，請檢查 *ExitParmCount* 欄位。 您可以使用 16 位元組 *ExitUserArea* 欄位來儲存應用程式取得之任何動態儲存體的位址。 此欄位會在呼叫結束程式時保留，且其生命期限與 CICS 作業相同。

如果您使用 CICS Transaction Server V3.2，則必須將跳出程式撰寫成 threadsafe ，並將跳出程式宣告為 threadsafe。 如果您使用舊版 CICS ，也建議您撰寫並宣告結束程式為安全執行緒，以準備移轉至 CICS Transaction Server V3.2。

您的結束程式可以在 *ExitResponse* 欄位中傳回 MQXCC_SUPPRESS_FUNCTION 或 MQXCC_SKIP_FUNCTION ，以暫停執行 MQI 呼叫。 若要容許執行呼叫 (以及在呼叫完成之後重新呼叫結束程式) ，您的結束程式必須傳回 MQXCC_OK。

在 MQI 呼叫之後呼叫時，結束程式可以檢查並修改呼叫所設定的完成碼及原因碼。

使用注意事項

以下是撰寫結束程式時要考量的一些一般要點:

- 基於效能原因，請以組譯語言撰寫您的程式。 如果您以 IBM MQ for z/OS 支援的任何其他語言撰寫它，則必須提供您自己的資料定義檔。
- 以 AMODE (31) 和 RMODE (ANY) 來鏈結編輯您的程式。
- 若要定義程式的結束參數區塊，請使用組譯語言巨集 CMQXPA。
- 當您定義跳出程式及跳出程式所呼叫的任何程式時，請指定 CONCURRENCY (THREADSEAN)。
- 如果您使用 CICS Transaction Server for z/OS 儲存體保護功能，則您的程式必須在 CICS 執行金鑰中執行。 也就是說，您必須指定 EXECKEY (CICS) 定義您的跳出程式及它所傳遞控制的任何程式時。 如需 CICS 結束程式及 CICS 儲存體保護機能的相關資訊，請參閱 *CICS* 自訂作業手冊。
- 您的程式可以使用所有 API (例如， IMS、 Db2 及 CICS) CICS 作業相關使用者結束程式可以使用。 它也可以使用 MQCONN、MQCONNX 及 MQDISC 以外的任何 MQI 呼叫。 不過，結束程式內的任何 MQI 呼叫不會再次呼叫結束程式。
- Your program can issue EXEC CICS SYNCPOINT or EXEC CICS SYNCPOINT ROLLBACK commands. 不過，這些指令會確定或回復 **所有** 作業所完成的更新，直到使用結束程式時為止，因此不建議使用它們。
- 您的程式必須以發出 EXEC CICS RETURN 指令來結束。 不得使用 XCTL 指令來傳送控制。
- 結束程式會撰寫為 IBM MQ for z/OS 程式碼的延伸。 確保您的結束程式不會干擾任何使用 MQI 的 IBM MQ for z/OS 程式或交易。 這些通常以 CSQ 或 CK 為字首。
- 如果 CSQCAPX 定義給 CICS，當 CICS 連接至 IBM MQ for z/OS 時， CICS 系統會嘗試載入結束程式。 如果此嘗試成功，則會將訊息 CSQC301I 傳送至 CKQC 畫面或系統主控台。 如果載入不成功 (例如，如果載入模組不存在於 DFHRPL 連結中的任何檔案庫中) ，則會將訊息 CSQC315 傳送至 CKQC 畫面或系統主控台。
- 因為通訊區中的參數是位址，所以跳出程式必須定義為 CICS 系統的本端 (亦即，不是遠端程式)。

z/OS 範例 *API* 交互結束程式 *CSQCAPX*

範例結束程式提供作為組譯語言程式。 原始檔 (CSQCAPX) 在程式庫 **thlqual**.SCSQASMS 中提供 (其中 **thlqual** 是您安裝所使用的高階限定元)。 此原始檔包括說明程式邏輯的虛擬碼。

範例程式包含起始設定碼及您在撰寫自己的結束程式時可以使用的佈置。

範例顯示如何:

- 設定結束參數區塊
- 處理呼叫及結束參數區塊
- 判斷正在呼叫結束程式的 MQI 呼叫
- 判定是否在處理 MQI 呼叫之前或之後呼叫結束程式
- 將訊息放置在 CICS 暫時儲存體佇列上
- 使用巨集 DFHEIENT 進行動態儲存體取得，以維護重新進入
- 將 DFHEIBLK 用於 CICS 執行程式介面控制區塊
- 設陷錯誤狀況
- 將控制項傳回給呼叫程式

範例結束程式的設計

範例結束程式會將訊息寫入 CICS 暫時儲存體佇列 (CSQ1EXIT)，以顯示結束程式的作業。

這些訊息顯示是否在 MQI 呼叫之前或之後呼叫結束程式。 如果在呼叫之後呼叫結束程式，則訊息會包含呼叫所傳回的完成碼及原因碼。 範例使用 CMQXPA 巨集中的具名常數來檢查項目類型 (亦即，在呼叫之前或之後)。

範例不會執行任何監視功能，只會將具有時間戳記的訊息放入 CICS 佇列中，以指出它正在處理的呼叫類型。 這可指出 MQI 的效能，以及結束程式的正確運作。

註: 範例結束程式會針對程式執行時所進行的每一個 MQI 呼叫，發出六個 EXEC CICS 呼叫。 如果您使用這個結束程式， IBM MQ for z/OS 效能會降低。

z/OS 準備及使用 *API* 交互結束程式

範例結束程式僅以來源格式提供。

若要使用範例結束程式或您已撰寫的結束程式，請建立載入程式庫，如同您對任何其他 CICS 程式所做的一樣，如 第 862 頁的『在 z/OS 中建置 CICS 應用程式』中所述。

- 若為 CICS Transaction Server for z/OS 及 CICS for MVS/ESA，當您更新 CICS 系統定義 (CSD) 資料集時，您需要的定義位於成員 **thlqual**.SCSQPROC (CSQ4B100) 中。

  註: 定義使用 MQ 字尾。 如果您的企業已使用此字尾，則必須在組合階段之前變更此字尾。

如果您使用提供的預設 CICS 程式定義，則結束程式 CSQCAPX 會以 ***disabled*** 狀態安裝。 這是因為使用跳出程式可以大幅降低效能。

如果要暫時啟動 API 交互結束程式，請執行下列動作:

1. 從 CICS 主要終端機發出指令 **CEMT S PROGRAM(CSQCAPX) ENABLED**。
2. 執行 CKQC 交易，並使用「連線」下拉清單中的選項 3，將 API 交互結束程式的狀態變更為 **已啟用**。

如果您想要執行 IBM MQ for z/OS 並永久啟用 API 交互結束程式，且 CICS Transaction Server for z/OS 及 CICS for MVS/ESA，請執行下列其中一項:

- 變更成員 CSQ4B100 中的 CSQCAPX 定義，將 STATUS (DISABLED) 變更為 STATUS (ENABLED)。 您可以使用 CICS 提供的批次程式 DFHCSDUP 來更新 CICS CSD 定義。
- 將狀態從「已停用」變更為「已啟用」，以變更 CSQCAT1 群組中的 CSQCAPX 定義。

在這兩種情況下，您都必須重新安裝群組。 若要這樣做，您可以冷啟動 CICS 系統，或在 CICS 執行時使用 CICS CEDA 交易來重新安裝群組。

註: 如果群組中目前有任何項目在使用中，則使用 CEDA 可能會導致錯誤。

產品相關程式設計介面資訊的結尾。

z/OS 使用共用佇列進行應用程式設計

本主題提供在設計新的應用程式以使用共用佇列時，以及將現有應用程式移轉至共用佇列環境時，您需要考量的部分因素相關資訊。

z/OS 序列化應用程式

某些類型的應用程式可能必須確保從佇列擷取訊息的順序與它們到達佇列的順序完全相同。

例如，如果使用 IBM MQ 將資料庫更新投影到遠端系統上，則必須在說明插入該記錄的訊息之後處理說明記錄更新的訊息。 在本端佇列作業環境中，通常由應用程式使用 MQOO_INPUT_EXCLUSIVE 選項來取得開啟佇列的訊息，從而防止任何其他取得應用程式同時處理佇列。

IBM MQ 可讓應用程式以相同方式專門開啟共用佇列。 不過，如果應用程式是從佇列的分割區運作 (例如，所有資料庫更新項目都在相同佇列上，但表格 A 的相關 ID 為 A，表格 B 的相關 ID 為 B)，且應用程式想要同時取得表格 A 更新項目及表格 B 更新項目的訊息，則無法使用簡單的機制來專門開啟佇列。

如果這種類型的應用程式要利用共用佇列的高可用性，您可以決定當主要取得應用程式或佇列管理程式失敗時，另一個存取相同共用佇列的應用程式實例 (在次要佇列管理程式上執行) 應該接管。

如果主要佇列管理程式失敗，則會發生兩種情況:

- 共用佇列同層級回復可確保來自主要應用程式的任何未完成更新已完成或已取消。
- 次要應用程式接管處理佇列。

次要應用程式可能在處理完所有未完成的工作單元之前啟動，這可能導致次要應用程式擷取失序訊息。 若要解決此類型的問題，應用程式可以選擇成為 序列化應用程式。

序列化應用程式會使用 MQCONNX 呼叫來連接至佇列管理程式，並在連接時指定該應用程式唯一的連線標籤。 應用程式所執行的任何工作單元都會標示連線標籤。 IBM MQ 可確保序列化佇列共用群組內具有相同連線標籤的工作單元 (根據 MQCONNX 呼叫中的序列化選項)。

這表示，如果主要應用程式使用具有連線標籤 `Database shadow retriever` 的 MQCONNX 呼叫，且次要接管應用程式嘗試使用具有相同連線標籤的 MQCONNX 呼叫，則次要應用程式無法連接至第二個 IBM MQ，除非已完成任何未完成的主要工作單元 (在此情況下，是透過同層級回復)。

對於相依於佇列上確切訊息順序的應用程式，請考量使用序列化應用程式技術。 具體如下:

- 在應用程式或佇列管理程式失敗之後，直到完成前次執行應用程式的所有確定及取消作業之前，不得重新啟動的應用程式。

  在此情況下，只有當應用程式在同步點中運作時，序列化應用程式技術才適用。
- 當相同應用程式的另一個實例已在執行中時，不得啟動的應用程式。

  在此情況下，只有在應用程式無法開啟佇列進行專用輸入時，才需要序列化應用程式技術。

註: IBM MQ 只保證在符合特定準則時保留訊息序列。 這些說明在 MQGET 的說明中。

z/OS 不適合與共用佇列搭配使用的應用程式

當您使用共用佇列時，不支援 IBM MQ 的部分特性，因此使用這些特性的應用程式不適用於共用佇列環境。

在設計共用佇列應用程式時，請考量下列要點:

- 共用佇列的佇列索引限制。 如果您要使用訊息 ID 或相關性 ID 來選取您要從佇列中取得的訊息，則應該使用正確值來檢索佇列。 如果您單獨依訊息 ID 選取訊息，則佇列需要 MQIT_MSG_ID 的索引類型 (雖然您也可以使用 MQIT_NONE)。 如果您單獨依相關性 ID 選取訊息，則佇列必須具有 MQIT_CORREL_ID 的索引類型。
- 您無法使用暫時動態佇列作為共用佇列。 不過，您可以使用永久動態佇列。 共用動態佇列的模型具有 SHAREDYN (共用動態) 的 DEFTYPE，雖然它們是以與 PERMDYN (永久動態) 佇列相同的方式建立及毀損。

z/OS 決定是否共用非應用程式佇列

當考量共用非應用程式佇列時，請使用此資訊。

除了您可能要考量共用的應用程式佇列之外，還有其他佇列:

起始佇列

如果您定義共用起始佇列，只要至少有一個觸發監視器在執行中，就不需要在佇列共用群組中的每一個佇列管理程式上執行觸發監視器。 (即使在佇列共用群組中的每一個佇列管理程式上執行觸發監視器，您也可以使用共用起始佇列。)

如果您具有共用應用程式佇列，並使用觸發類型 EVERY (或觸發類型 FIRST，具有小型觸發間隔，其行為類似於觸發類型 EVERY)，則起始佇列必須一律為共用佇列。 如需何時使用共用起始佇列的相關資訊，請參閱 第 752 頁的表 131。

SYSTEM.* 佇列

您可以定義 SYSTEM.ADMIN.* 用來保留事件訊息作為共用佇列的佇列。 這有助於在發生異常狀況時檢查負載平衡。 IBM MQ 所建立的每一則事件訊息都包含一個相關性 ID，指出哪個佇列管理程式產生它。

您必須定義 SYSTEM.QSG.* 用於共用通道及內部群組佇列作業作為共用佇列的佇列。

您也可以變更 SYSTEM.DEFAULT.LOCAL.QUEUE，或定義您自己的預設共用佇列定義。 如需相關資訊，請參閱 定義 IBM MQ for z/OS 的系統物件 。

您無法定義任何其他 SYSTEM.* 佇列作為共用佇列。

z/OS 移轉現有應用程式以使用共用佇列

在共用佇列環境中，原因碼、觸發及 MQINQ API 呼叫可以不同方式運作。

如需將現有佇列移轉至共用佇列的相關資訊，請參閱 將非共用佇列移轉至共用佇列 。

當您移轉現有的應用程式時，請考量下列事項，這可能在共用佇列環境中以不同方式運作:

原因碼

當您移轉現有應用程式以使用共用佇列時，請檢查可發出的新原因碼。

觸發

如果您使用共用應用程式佇列，則觸發只會處理已確定的訊息 (在非共用應用程式佇列上，觸發會處理所有訊息)。

如果您使用觸發來啟動應用程式，則可能想要使用共用起始佇列。 第 752 頁的表 131 說明在決定要使用的起始佇列類型時需要考量的事項。

表 *131:* 何時使用共用起始佇列

	非共用應用程式佇列	**共用應用程式佇列**
非共用起始佇列	對於舊版。	如果您使用觸發類型 FIRST 或 DEPTH，則可以使用具有共用應用程式佇列的非共用起始佇列。 可能會產生額外觸發訊息，但此設定適用於觸發長時間執行的應用程式 (例如 CICS bridge)，並提供高可用性。 對於觸發類型 FIRST 或 DEPTH，觸發訊息會在每個執行觸發監視器且尚未開啟應用程式佇列以供輸入的佇列管理程式上觸發應用程式實例。 每個佇列管理程式會產生一則觸發訊息; 如果有多個觸發監視器針對特定佇列管理程式上的非共用本端起始佇列執行，則它們會競相處理訊息。

表 131: 何時使用共用起始佇列 (繼續)

	非共用應用程式佇列	共用應用程式佇列
共用起始佇列	請勿使用具有非共用應用程式佇列的共用起始佇列。	對於觸發類型 EVERY，當應用程式將訊息放置到共用應用程式佇列時，放置佇列管理程式會判定哪些佇列管理程式對觸發程式有興趣-每個事件，並將通知傳送至其中一個佇列管理程式。 在通知的佇列管理程式上，產生的動作會產生對起始佇列的觸發訊息。 **註:** 如果您具有觸發類型為 EVERY 的共用應用程式佇列，請使用共用起始佇列，或者在特定情況下可能會遺失觸發訊息; 例如，佇列管理程式失敗。 對於觸發類型 FIRST 或 DEPTH，每一個已開啟具名起始佇列以供輸入的佇列管理程式會產生一則觸發訊息。 **註:** 若為觸發類型 FIRST 或 DEPTH，如果一個觸發監視器實例忙碌，則這會讓較不忙碌的觸發監視器處理來自共用起始佇列的多個觸發訊息。 因此，可能會針對給定佇列管理程式啟動伺服器應用程式的多個實例。 請注意，這些多個實例會因處理多個觸發訊息而啟動。 通常，對於觸發類型 FIRST 或 DEPTH，如果應用程式實例已在處理應用程式佇列，則應用程式所連接的佇列管理程式將不會產生另一個觸發訊息。

MQINQ
當您使用 MQINQ 呼叫來顯示共用佇列的相關資訊時，佇列開啟以供輸入及輸出的 MQOPEN 呼叫數值僅與發出該呼叫的佇列管理程式相關。 不會產生佇列共用群組中已開啟佇列之其他佇列管理程式的相關資訊。

z/OS IBM MQ for z/OS 上的 IMS 及 IMS 橋接器應用程式

此資訊可協助您使用 IBM MQ 撰寫 IMS 應用程式。

- 如果要在 IMS 應用程式中使用同步點和 MQI 呼叫，請參閱 第 58 頁的『使用 IBM MQ 撰寫 IMS 應用程式』。
- 若要撰寫使用 IBM MQ - IMS 橋接器的應用程式，請參閱 第 61 頁的『撰寫 IMS 橋接器應用程式』。

使用下列鏈結，以進一步瞭解 IBM MQ for z/OS 上 IMS 及 IMS 橋接器應用程式的相關資訊:

- 第 58 頁的『使用 IBM MQ 撰寫 IMS 應用程式』
- 第 61 頁的『撰寫 IMS 橋接器應用程式』

相關概念
第 604 頁的『訊息佇列介面概觀』
瞭解「訊息佇列介面 (MQI)」元件。

第 616 頁的『連接至佇列管理程式並切斷與佇列管理程式的連線』
若要使用 IBM MQ 程式設計服務，程式必須具有與佇列管理程式的連線。 使用此資訊來瞭解如何連接至佇列管理程式，以及與佇列管理程式中斷連線。

第 621 頁的『開啟及關閉物件』
此資訊提供開啟及關閉 IBM MQ 物件的見解。

第 630 頁的『將訊息放置在佇列上』
使用此資訊來瞭解如何將訊息放置在佇列上。

第 643 頁的『從佇列取得訊息』
使用此資訊來瞭解從佇列取得訊息的相關資訊。

第 713 頁的『查詢及設定物件屬性』
屬性是定義 IBM MQ 物件性質的內容。

第 716 頁的『確定及取消工作單元』
此資訊說明如何確定及取消工作單元中發生的任何可回復取得及放置作業。

第 725 頁的『使用觸發程式啟動 IBM MQ 應用程式』
瞭解觸發程式以及如何使用觸發程式來啟動 IBM MQ 應用程式。

第 741 頁的『使用 MQI 及叢集』
對於與叢集作業相關的呼叫和回覆碼，有一些特殊選項。

第 745 頁的『在 IBM MQ for z/OS 上使用及撰寫應用程式』
IBM MQ for z/OS 應用程式可以由在許多不同環境中執行的程式組成。 這表示他們可以利用多個環境中可用的設施。

z/OS 使用 *IBM MQ* 撰寫 *IMS* 應用程式

在 IMS 應用程式中使用 IBM MQ 時還有進一步的考量。這些包括可以使用哪些 MQ API 呼叫，以及用於同步點的機制。

使用下列鏈結，以進一步瞭解如何在 IBM MQ for z/OS 上撰寫 IMS 應用程式:

- 第 58 頁的『IMS 應用程式中的同步點』
- 第 59 頁的『IMS 應用程式中的 MQI 呼叫』

限制

使用 IMS 配接器的應用程式可以使用哪些 IBM MQ API 呼叫有一些限制。

在使用 IMS 配接器的應用程式內不支援下列 IBM MQ API 呼叫:

- MQCB
- MQCB_XX_ENCODE_CASE_ONE function
- MQCTL

相關概念

第 61 頁的『撰寫 IMS 橋接器應用程式』
本主題包含撰寫應用程式以使用 IBM MQ - IMS 橋接器的相關資訊。

z/OS *IMS* 應用程式中的同步點

在 IMS 應用程式中，您可以使用對 IOPCB 及 CHKP (檢查點) 的 GU (取得唯一) 之類的 IMS 呼叫來建立同步點。

若要回復自前一個檢查點以來的所有變更，您可以使用 IMS ROLB (回復) 呼叫。 如需相關資訊，請參閱 IMS 說明文件中的 ROLB 呼叫 。

佇列管理程式是兩段式確定通訊協定中的參與者; IMS 同步點管理程式是協調程式。

IMS 配接器會在同步點關閉所有開啟控點 (批次或非訊息驅動 BMP 環境除外)。 這是因為不同的使用者可以起始下一個工作單元，而 IBM MQ 安全檢查是在發出 MQCONN、MQCONNX 及 MQOPEN 呼叫時執行，而不是在發出 MQPUT 或 MQGET 呼叫時執行。

不過，在「等待輸入 (WFI)」或虛擬「等待輸入 (PWFI)」環境中， IMS 不會通知 IBM MQ 關閉控點，直到下一個訊息到達或 QC 狀態碼傳回應用程式為止。 如果應用程式正在 IMS 區域中等待，且任何這些控點都屬於已觸發的佇列，則不會發生觸發，因為佇列已開啟。 因此，在 WFI 或 PWFI 環境中執行的應用程式應該在對 IOPCB 執行 GU 以取得下一則訊息之前，明確 MQCLOSE 佇列控點。

如果 IMS 應用程式 (BMP 或 MPP) 發出 MQDISC 呼叫，則會關閉開啟佇列，但不會採用隱含的同步點。 如果應用程式正常結束，則會關閉任何開啟的佇列，並進行隱含的確定。 如果應用程式異常結束，則會關閉任何開啟的佇列，並發生隱含的取消。

z/OS *IMS* 應用程式中的 *MQI* 呼叫

使用此資訊來瞭解在「伺服器」應用程式及「查詢」應用程式上使用 MQI 呼叫的相關資訊。

本節涵蓋在下列類型的 IMS 應用程式中使用 MQI 呼叫:

- 第 755 頁的『伺服器應用程式』

- 第 757 頁的『查詢應用程式』

伺服器應用程式

以下是 MQI 伺服器應用程式模型的大綱:

```
Initialize/Connect
.
Open queue for input shared
.
Get message from IBM MQ queue
.
Do while Get does not fail
.
If expected message received
Process the message
Else
Process unexpected message
End if
.
Commit
.
Get next message from IBM MQ queue
.
End do
.
Close queue/Disconnect
.
END
```

範例程式 CSQ4ICB3 顯示在 C/370 中使用此模型的 BMP 實作。 程式會先建立與 IMS 的通訊，然後再建立與 IBM MQ 的通訊:

```
main()
----
Call InitIMS
If IMS initialization successful
Call InitMQM
If IBM MQ initialization successful
Call ProcessRequests
Call EndMQM
End-if
End-if

Return
```

IMS 起始設定會判定是否以訊息驅動或批次導向 BMP 來呼叫程式，並相應地控制 IBM MQ 佇列管理程式連線及佇列控點:

```
InitIMS
-------
Get the IO, Alternate and Database PCBs
Set MessageOriented to true

Call ctdli to handle status codes rather than abend
If call is successful (status code is zero)
While status code is zero
Call ctdli to get next message from IMS message queue
If message received
Do nothing
Else if no IOPBC
Set MessageOriented to false
Initialize error message
Build 'Started as batch oriented BMP' message
Call ReportCallError to output the message
End-if
Else if response is not 'no message available'
Initialize error message
Build 'GU failed' message
Call ReportCallError to output the message
Set return code to error
End-if
End-if
```

```
End-while
Else
Initialize error message
Build 'INIT failed' message
Call ReportCallError to output the message
Set return code to error
End-if

Return to calling function
```

IBM MQ 起始設定會連接至佇列管理程式，並開啟佇列。 在訊息驅動 BMP 中，這會在取得每一個 IMS 同步點之後呼叫; 在批次導向 BMP 中，這只會在程式啟動期間呼叫:

```
InitMQM
-------
Connect to the queue manager
If connect is successful
Initialize variables for the open call
Open the request queue
If open is not successful
Initialize error message
Build 'open failed' message
Call ReportCallError to output the message
Set return code to error
End-if
Else
Initialize error message
Build 'connect failed' message
Call ReportCallError to output the message
Set return code to error
End-if

Return to calling function
```

MPP 中伺服器模型的實作受到 MPP 每次呼叫處理單一工作單元的影響。 這是因為當採用同步點 (GU) 時，會關閉連線和佇列控點，並遞送下一個 IMS 訊息。 下列其中一項可部分克服此限制:

- **在單一工作單元內處理許多訊息**

  這涉及:

  – 讀取訊息
  – 正在處理必要的更新項目
  – 放置回覆

  迴圈中，直到已處理所有訊息，或直到已處理設定的訊息數目上限為止，此時會採用同步點。

  以這種方式只能處理特定類型的應用程式 (例如，簡式資料庫更新或查詢)。 雖然 MQI 回覆訊息可以放置在所處理 MQI 訊息的發送端權限下，但需要小心處理任何 IMS 資源更新項目的安全含意。

- **每次呼叫 MPP 時處理一則訊息，並確保多重排程 MPP 以處理所有可用的訊息**。

  當 IBM MQ 佇列上有訊息且沒有負責處理 MPP 交易的應用程式時，請使用 IBM MQ IMS 觸發監視器程式 (CSQQTRMN) 來排定 MPP 交易。

  如果觸發監視器啟動 MPP ，則佇列管理程式名稱及佇列名稱會傳遞至程式，如下列 COBOL 程式碼擷取所示:

```
* Data definition extract
01 WS-INPUT-MSG.
05 IN-LL1         PIC S9(3) COMP.
05 IN-ZZ1         PIC S9(3) COMP.
05 WS-STRINGPARM      PIC X(1000).
01 TRIGGER-MESSAGE.
COPY CMQTMC2L.
*
* Code extract
GU-IOPCB SECTION.
MOVE SPACES TO WS-STRINGPARM.
CALL 'CBLTDLI' USING GU,
IOPCB,
WS-INPUT-MSG.
IF IOPCB-STATUS = SPACES
```

```
MOVE WS-STRINGPARM TO MQTMC.
*  ELSE handle error
*
* Now use the queue manager and queue names passed
DISPLAY 'MQTMC-QMGRNAME     ='
MQTMC-QMGRNAME OF MQTMC '='.
DISPLAY 'MQTMC-QNAME        ='
MQTMC-QNAME  OF MQTMC '='.
```

雖然無法使用 CSQQTRMN 觸發 BMP，但在批次處理區域中最好支援伺服器模型 (預期是長時間執行的作業)。

查詢應用程式

一般 IBM MQ 應用程式起始查詢或更新的運作方式如下:

- 從使用者收集資料
- 放置一或多則 IBM MQ 訊息
- 取得回覆訊息 (您可能必須等待它們)
- 提供回應給使用者

因為放置在 IBM MQ 佇列中的訊息在確定之前不會變成可供其他 IBM MQ 應用程式使用，所以必須將它們置於同步點之外，或 IMS 應用程式必須分割成兩個交易。

如果查詢涉及放置單一訊息，您可以使用 無同步點 選項; 不過，如果查詢更複雜，或涉及資源更新，如果發生失敗且未使用同步點，則可能會發生一致性問題。

若要克服此問題，您可以使用 MQI 呼叫 (使用程式對程式訊息切換參數) 來分割 IMS MPP 交易; 如需相關資訊，請參閱 *IMS 跨系統通訊 (ISC)* 。 這容許在 MPP 中實作查詢程式:

```
Initialize first program/Connect
.
Open queue for output
.
Put inquiry to IBM MQ queue
.
Switch to second IBM MQ program, passing necessary data in save
pack area (this commits the put)
.
END
.
.
Initialize second program/Connect
.
Open queue for input shared
.
Get results of inquiry from IBM MQ queue
.
Return results to originator
.
END
```

z/OS 撰寫 *IMS* 橋接器應用程式

本主題包含撰寫應用程式以使用 IBM MQ - IMS 橋接器的相關資訊。

如需 IBM MQ - IMS 橋接器的相關資訊，請參閱 IMS 橋接器。

請使用下列鏈結，以進一步瞭解如何在 IBM MQ for z/OS 上撰寫 IMS 橋接器應用程式:

- 第 62 頁的『IMS 橋接器如何處理訊息』
- 第 763 頁的『透過 IBM MQ 撰寫 IMS 交易程式』

相關概念

第 58 頁的『使用 IBM MQ 撰寫 IMS 應用程式』
在 IMS 應用程式中使用 IBM MQ 時還有進一步的考量。這些包括可以使用哪些 MQ API 呼叫，以及用於同步點的機制。

z/OS *IMS* 橋接器如何處理訊息

當您使用 IBM MQ - IMS 橋接器將訊息傳送至 IMS 應用程式時，需要以特殊格式建構訊息。

您還必須將訊息放置在已定義儲存類別 (指定目標 IMS 系統的 XCF 群組及成員名稱) 的 IBM MQ 佇列上。 這些稱為 MQ-IMS 橋接器佇列，或只是 **橋接器** 佇列。

如果橋接器佇列以 QSGDISP (QMGR) 定義，或以 QSGDISP (SHARED) 搭配 NOSHARE 選項一起定義，則 IBM MQ-IMS 橋接器需要對橋接器佇列的互斥輸入存取 (MQOO_INPUT_EXCLUSIVE)。

在將訊息傳送至 IMS 應用程式之前，使用者不需要登入 IMS 。 MQMD 結構的 *UserIdentifier* 欄位中的使用者 ID 用於安全檢查。 檢查層次是在 IBM MQ 連接至 IMS 時決定，並在 IMS 橋接器的應用程式存取控制中說明。 這可讓您實作虛擬登入。

IBM MQ - IMS 橋接器接受下列類型的訊息:

- 包含 IMS 交易資料及 MQIIH 結構 (在 MQIIH 中說明) 的訊息:

```
MQIIH LLZZ<trancode><data>[LLZZ<data>][LLZZ<data>]
```

註:

1. 方括弧 [] 代表選用的多區段。
2. 將 MQMD 結構的 *Format* 欄位設為 MQFMT_IMS ， 以使用 MQIIH 結構。

- 包含 IMS 交易資料但無 MQIIH 結構的訊息:

```
LLZZ<trancode><data> \
[LLZZ<data>][LLZZ<data>]
```

IBM MQ 會驗證訊息資料，以確保 LL 位元組加上 MQIIH (如果存在的話) 長度的總和等於訊息長度。

當 IBM MQ - IMS 橋接器從橋接器佇列取得訊息時，它會如下處理訊息:

- 如果訊息包含 MQIIH 結構，則橋接器會驗證 MQIIH (請參閱 MQIIH)、建置 OTMA 標頭，並將訊息傳送至 IMS。 交易碼指定在輸入訊息中。 如果這是邏輯終端機， IMS 會以 DFS1288E 訊息回覆。 如果交易碼代表指令，則 IMS 會執行該指令; 否則訊息會排入 IMS 佇列以進行交易。
- 如果訊息包含 IMS 交易資料，但沒有 MQIIH 結構，則 IMS 橋接器會做出下列假設:
  - 交易碼以 5 到 12 個位元組的使用者資料為單位
  - 交易處於非交談模式
  - 交易處於確定模式 0 (commit-then-send)
  - MQMD 中的 *Format* 用作 *MFSMapName* (輸入時)
  - 安全模式是 MQISS_CHECK

  也會建置不含 MQIIH 結構的回覆訊息，從 IMS 輸出的 *MFSMapName* 中取得 MQMD 的 *Format* 。

IBM MQ - IMS 橋接器會針對每一個 IBM MQ 佇列使用一或兩個 Tp 管道:

- 使用「確定」模式 0 (COMMIT_THEN_SEND) (這些在 IMS /DIS TMEMBER 用戶端 TPIPE xxxx 指令的狀態欄位中顯示 SYN) 的所有訊息會使用同步化 Tpipe
- 使用「確定」模式 1 (SEND_THEN_COMMIT) 的所有訊息都使用非同步化 Tpipe

第一次使用 Tp 管道時， IBM MQ 會建立 Tp 管道。 在 IMS 重新啟動之前，會存在未同步的 Tpipe。 已同步的 Tp 管道存在，直到 IMS 冷啟動為止。 您無法自行刪除這些 Tp 管道。

如需 IBM MQ - IMS 橋接器如何處理訊息的相關資訊，請參閱下列主題:

- 第 63 頁的『將 IBM MQ 訊息對映至 IMS 交易類型』
- 第 63 頁的『如果訊息無法放入 IMS 佇列』
- 第 64 頁的『IMS 橋接器回饋碼』
- 第 64 頁的『來自 IMS 橋接器的訊息中的 MQMD 欄位』

- 第 65 頁的『來自 IMS 橋接器的訊息中的 MQIIH 欄位』
- 第 66 頁的『來自 IMS 的回覆訊息』
- 第 66 頁的『在 IMS 交易中使用替代回應 PCB』
- 第 66 頁的『從 IMS 傳送自發的訊息』
- 第 66 頁的『訊息分段』
- 第 67 頁的『與 IMS 橋接器之間的訊息資料轉換』

相關概念

第 763 頁的『透過 IBM MQ 撰寫 IMS 交易程式』

透過 IBM MQ 處理 IMS 交易所需的編碼取決於 IMS 交易所需的訊息格式，以及它可以傳回的回應範圍。 不過，當您的應用程式處理 IMS 畫面格式化資訊時，有幾點需要考量。

z/OS 將 *IBM MQ* 訊息對映至 *IMS* 交易類型

此表格說明 IBM MQ 訊息至 IMS 交易類型的對映。

表 *132: IBM MQ* 訊息如何對映至 *IMS* 交易類型

IBM MQ 訊息類型	**Commit-then-send (模式 0)-使用已同步的 IMS Tp 管道**	**Send-then-commit (模式 1)-使用非同步化 IMS Tp 管道**
持續 IBM MQ 訊息	• 可回復的完整功能交易 • IMS 拒絕無法復原的交易	• 捷徑交易 • 交談式交易 • 完整功能交易
非持續 IBM MQ 訊息	• 無法復原的完整功能交易 • IMS V8 及 APAR PQ61404 以及 IMS 的所有更新版本允許可回復交易	• 捷徑交易 • 交談式交易 • 完整功能交易

註: IMS 指令無法使用具有確定模式 0 的持續 IBM MQ 訊息。 如需相關資訊，請參閱 確定模式 (commitMode) 。

z/OS 如果訊息無法放入 *IMS* 佇列

瞭解無法將訊息放入 IMS 佇列時要採取的動作。

如果訊息無法放入 IMS 佇列， IBM MQ 會採取下列動作:

- 如果因為訊息無效而無法將訊息放置到 IMS ，則會將訊息放置到無法傳送郵件的佇列，並將訊息傳送到系統主控台。
- 如果訊息有效，但遭到 IMS 拒絕，則 IBM MQ 會將錯誤訊息傳送至系統主控台，該訊息包括 IMS 感應碼，且 IBM MQ 訊息會放置在無法傳送郵件的佇列中。 如果 IMS 感應碼為 001A， IMS 會將包含失敗原因的 IBM MQ 訊息傳送至回覆目的地佇列。

  註: 在先前列出的情況下，如果 IBM MQ 因任何原因而無法將訊息放入無法傳送郵件的佇列，則會將訊息傳回原始 IBM MQ 佇列。 錯誤訊息會傳送至系統主控台，且不會從該佇列傳送進一步訊息。

  若要重新傳送訊息，請執行下列 **其中一個** :

  – 停止並重新啟動 IMS 中對應於佇列的 Tp 管道
  – 將佇列變更為 GET (DISABLED) ，並再次變更為 GET (ENABLED)
  – 停止並重新啟動 IMS 或 OTMA
  – 停止並重新啟動 IBM MQ 子系統
- 如果 IMS 拒絕訊息以外的任何其他訊息錯誤，則 IBM MQ 訊息會傳回原始佇列， IBM MQ 會停止處理佇列，並將錯誤訊息傳送至系統主控台。

如果需要異常狀況報告訊息，橋接器會以發送端的權限將它放入回覆目的地佇列。 如果無法將訊息放入佇列，則會以橋接器的權限將報告訊息放入無法傳送郵件的佇列。 如果無法將它放入 DLQ，則會捨棄它。

z/OS *IMS* 橋接器回饋碼

IMS 感應碼通常在 IBM MQ 主控台訊息中以十六進位格式輸出，例如 CSQ2001I (例如，感應碼 0x001F)。 在放入無法傳送郵件之訊息的無法傳送郵件的標頭中看到的 IBM MQ 回饋碼是十進位數。

IMS 橋接器回饋碼的範圍為 301 至 399，或 600 至 855 (若為 NACK 感應碼 0x001A)。 它們是從 IMS-OTMA 感應碼對映，如下所示:

1. IMS-OTMA 感應碼會從十六進位數轉換成十進位數。
2. 300 會新增至 1 中計算所產生的數字，並提供 IBM MQ *Feedback* 代碼。
3. IMS-OTMA 感應碼 0x001A，十進位 26 是特殊情況。 即會產生範圍 600-855 內的 *Feedback* 程式碼。
    a. IMS-OTMA 原因碼會從十六進位數轉換成十進位數。
    b. 600 會新增至 a 中計算所產生的數字，並提供 IBM MQ *Feedback* 代碼。

如需 IMS-OTMA 感應碼的相關資訊，請參閱 NAK 訊息的 OTMA 感應碼。

z/OS 來自 *IMS* 橋接器的訊息中的 *MQMD* 欄位

瞭解來自 IMS 橋接器之訊息中的 MQMD 欄位。

IMS 會在 OTMA 標頭的「使用者資料」區段中攜帶原始訊息的 MQMD。 如果訊息源自 IMS，則由 IMS 目的地解決方案結束程式建置。 從 IMS 接收的訊息 MQMD 建置如下:

StrucID
"MD"

版本
MQMD_VERSION_1

報告
MQRO_NONE

MsgType
MQMT_REPLY

期限
如果 MQIIH 的旗標欄位中設定了 MQIH_PASS_EXPIRATION，則此欄位會包含剩餘到期時間，否則它會設為 MQEI_UNQUALITED

意見
MQFB_NONE

編碼
MQENC.Native (z/OS 系統的編碼)

CodedCharSetId
MQCCSI_Q_MGR (z/OS 系統的 CodedCharSetID)

格式
MQFMT_IMS (如果 MQMD.Format 為 MQFMT_IMS，否則為 IOPCB.MODNAME

優先順序
MQMD.Priority

持續性
視確定模式而定: 如果 CM-1; 持續性符合 IMS 訊息的可回復性 (如果 CM-0)，則輸入訊息的 MQMD.Persistence

MsgId
MQMD.MsgId 如果 MQRO_PASS_MSG_ID，則為 MQRO_PASS_MSG_ID，否則為新的 MsgId (預設值)

CorrelId
MQMD.CorrelId，否則為 MQMD.MsgId (預設值)

BackoutCount
0

ReplyToQ
空白

回覆目的地佇列管理程式
空白 (在 MQPUT 期間由佇列管理程式設為本端佇列管理程式名稱)

UserIdentifier
MQMD.UserIdentifier

AccountingToken
MQMD.AccountingToken

ApplIdentityData
MQMD.ApplIdentityData

PutApplType
如果沒有錯誤，則為 MQAT_XCF ，否則為 MQAT_BRIDGE

PutApplName
<XCFgroupName><XCFmemberName> if no error, otherwise QMGR name

PutDate
放置訊息的日期

PutTime
放置訊息的時間

ApplOriginData
空白

z/OS 來自 *IMS* 橋接器的訊息中的 *MQIIH* 欄位

瞭解來自 IMS 橋接器的訊息中的 MQIIH 欄位。

從 IMS 接收的訊息 MQIIH 建置如下:

StrucId
"IIH"

版本
1

StrucLength
84

編碼
MQENC_NATIVE

CodedCharSetId
MQCCSI_Q_MGR

格式
MQIIH.ReplyToFormat ，如果 MQIIH.ReplyToFormat 不是空白，則為輸入訊息，否則為 IOPCB.MODNAME

旗標
0

LTermOverride
OTMA 標頭中的 LTERM 名稱 (Tpipe)

MFSMapName
OTMA 標頭中的對映名稱

ReplyTo 格式
空白

鑑別程式
輸入訊息的 MQIIH.Authenticator ，如果將回覆訊息放入 MQ-IMS 橋接器佇列，則為空白。

TranInstanceID
來自 OTMA 標頭的交談 ID/伺服器記號 (如果在交談中)。 在 V14 之前的 IMS 版本中，如果不在交談中，則此欄位一律為空值。 從 IMS V14 開始，即使不在交談中， IMS 也可能會設定此欄位。

TranState
如果在交談中，則為 "C" ，否則為空白

CommitMode
OTMA 標頭的確定模式 ("0" 或 "1")

SecurityScope
Blank

已保留
Blank

z/OS 來自 *IMS* 的回覆訊息

當 IMS 交易 ISRT 至其 IOPCB 時，訊息會遞送回原始 LTERM 或 TPIPE。

這些在 IBM MQ 中視為回覆訊息。 來自 IMS 的回覆訊息會放入原始訊息中指定的回覆目的地佇列。 如果無法將訊息放入回覆目的地佇列，則會使用橋接器的權限將訊息放入無法傳送郵件的佇列。 如果無法將訊息放入無法傳送郵件的佇列，則會將負面確認通知傳送至 IMS ，以指出無法接收訊息。 然後，訊息的責任會回到 IMS。 如果您使用確定模式 0 ，則來自該 Tpipe 的訊息不會傳送至橋接器，且會保留在 IMS 佇列上; 亦即，在重新啟動之前不會傳送進一步訊息。 如果您使用確定模式 1 ，則其他工作可以繼續。

如果回覆具有 MQIIH 結構，則其格式類型為 MQFMT_IMS; 否則，其格式類型由插入訊息時使用的 IMS MOD 名稱指定。

z/OS 在 *IMS* 交易中使用替代回應 *PCB*

當 IMS 交易使用替代回應 PCB (ALTPCB 的 ISRT ，或對可修改的 PCB 發出 CHNG 呼叫) 時，會呼叫預先遞送結束程式 (DFSYPRX0) ，以判定是否應該重新遞送訊息。

如果要重新遞送訊息，則會呼叫目的地解析結束程式 (DFSYDRU0) 來確認目的地並準備標頭資訊，如需這些結束程式的相關資訊，請參閱 在 IMS 中使用 OTMA 結束程式 及 預先遞送結束程式 DFSYPRX0 。

除非在結束程式中採取動作，否則從 IBM MQ 佇列管理程式起始之 IMS 交易的所有輸出 (不論是 IOPCB 或 ALTPCB) 都會回到相同的佇列管理程式。

z/OS 從 *IMS* 傳送自發的訊息

若要將訊息從 IMS 傳送至 IBM MQ 佇列，您需要呼叫 ISRT 至 ALTPCB 的 IMS 交易。

您需要撰寫預先遞送及目的地解析結束程式，以從 IMS 遞送自發的訊息並建置 OTMA 使用者資料，以便能夠正確建置訊息的 MQMD。 如需這些結束程式的相關資訊，請參閱 預先遞送結束程式 DFSYPRX0 及 目的地解析使用者結束程式 。

註: IBM MQ - IMS 橋接器不知道它收到的訊息是回覆還是自發訊息。 在每一種情況下，它都會以相同方式處理訊息，並根據隨訊息送達的 OTMA UserData 來建置回覆的 MQMD 及 MQIIH

自發的訊息可以建立新的 Tp 管道。 例如，如果現有 IMS 交易切換至新的邏輯終端機 (例如 PRINT01) ，但實作需要透過 OTMA 遞送輸出，則會建立新的 Tpipe (在此範例中稱為 PRINT01)。 依預設，這是未同步的 Tpipe。 如果實作需要訊息可回復，請設定目的地解析結束程式輸出旗標。 如需相關資訊，請參閱 *IMS* 自訂作業手冊 。

z/OS 訊息分段

您可以將 IMS 交易定義為預期單一或多區段輸入。

原始 IBM MQ 應用程式必須將 MQIIH 結構後面的使用者輸入建構為一或多個 LLZZ 資料區段。 IMS 訊息的所有區段都必須包含在以單一 MQPUT 傳送的單一 IBM MQ 訊息中。

LLZZ 資料區段的長度上限由 IMS/OTMA (32767 位元組) 定義。 IBM MQ 訊息長度總計是 LL 位元組加上 MQIIH 結構長度的總和。

回覆的所有區段都包含在單一 IBM MQ 訊息中。

對於 MQFMT_IMS_VAR_STRING 格式的訊息，有進一步的 32 KB 限制。 當 ASCII 混合 CCSID 訊息中的資料轉換成 EBCDIC 混合 CCSID 訊息時，每次 SBCS 與 DBCS 字元之間有轉移時，都會新增移入位元組或移出位元組。 32 KB 限制適用於訊息的大小上限。 亦即，因為訊息中的 LL 欄位不能超過 32 KB，所以訊息不得超過 32 KB，包括所有移入及移出字元。 建置訊息的應用程式必須容許此情況。

z/OS 與 *IMS* 橋接器之間的訊息資料轉換

資料轉換是由分散式佇列機能 (可能呼叫任何必要的結束程式) 或內部群組佇列代理程式 (不支援使用結束程式) 在將訊息放入目的地佇列時執行，該目的地佇列已針對其儲存類別定義 XCF 資訊。 當發佈/訂閱將訊息遞送至佇列時，不會進行資料轉換。

在 CSQXLIB DD 陳述式所參照的資料集中，任何需要的結束程式都必須可供分散式佇列機能使用。 這表示您可以從任何 IBM MQ 平台使用 IBM MQ - IMS 橋接器，將訊息傳送至 IMS 應用程式。

如果有轉換錯誤，則會將訊息放入未轉換的佇列; 這最終會導致 IBM MQ - IMS 橋接器將其視為錯誤，因為橋接器無法辨識標頭格式。 如果發生轉換錯誤，則會將錯誤訊息傳送至 z/OS 主控台。

如需一般資料轉換的詳細資訊，請參閱 第 823 頁的『寫入資料-轉換結束程式』。

將訊息傳送至 IBM MQ - IMS 橋接器

若要確保正確執行轉換，您必須告知佇列管理程式訊息的格式。

如果訊息具有 MQIIH 結構，則 MQMD 中的 *Format* 必須設為內建格式 MQFMT_IMS，且 MQIIH 中的 *Format* 必須設為說明訊息資料的格式名稱。 如果沒有 MQIIH，請將 MQMD 中的 *Format* 設為您的格式名稱。

如果您的資料 (LLZZ 除外) 是所有字元資料 (MQCHAR)，請使用內建格式 MQFMT_IMS_VAR_STRING 作為您的格式名稱 (在 MQIIH 或 MQMD 中，視情況而定)。 否則，請使用您自己的格式名稱，在此情況下，您也必須提供格式的資料轉換結束程式。 除了資料本身之外，結束程式還必須處理訊息中 LLZZ 的轉換 (但它不需要在訊息開始時處理任何 MQIIH)。

如果您的應用程式使用 *MFSMapName*，則可以改為搭配使用訊息與 MQFMT_IMS，並在 MQIIH 的 MFSMapName 欄位中定義傳遞至 IMS 交易的對映名稱。

從 IBM MQ - IMS 橋接器接收訊息

如果您要傳送至 IMS 的原始訊息上存在 MQIIH 結構，則回覆訊息上也會存在 MQIIH 結構。

若要確保正確轉換您的回覆，請執行下列動作:

- 如果您在原始訊息上具有 MQIIH 結構，請在原始訊息的 MQIIH *ReplytoFormat* 欄位中指定您想要的回覆訊息格式。 此值位於回覆訊息的 MQIIH *Format* 欄位中。 This is particularly useful if all your output data is of the form LLZZ<character data>.
- 如果原始訊息沒有 MQIIH 結構，請在 IMS 應用程式的 ISRT 中將回覆訊息的格式指定為 IOPCB 的 MFS MOD 名稱。

z/OS 透過 *IBM MQ* 撰寫 *IMS* 交易程式

透過 IBM MQ 處理 IMS 交易所需的編碼取決於 IMS 交易所需的訊息格式，以及它可以傳回的回應範圍。 不過，當您的應用程式處理 IMS 畫面格式化資訊時，有幾點需要考量。

從 3270 畫面啟動 IMS 交易時，訊息會透過「IMS 訊息格式服務」傳遞。 這可以從交易看到的資料串流中移除所有終端機相依關係。 透過 OTMA 啟動交易時，不涉及 MFS。 如果在 MFS 中實作應用程式邏輯，則必須在新應用程式中重建此邏輯。

在部分 IMS 交易中，一般使用者應用程式可以修改某些 3270 畫面行為，例如，強調顯示已輸入無效資料的欄位。 此類型資訊的通訊方式是針對每一個需要程式修改的畫面欄位，在 IMS 訊息中新增一個雙位元組屬性欄位。

因此，如果您要撰寫應用程式以模擬 3270，則在建置或接收訊息時需要考量這些欄位。

您可能需要在程式中撰寫程式碼資訊，才能處理:

- 按下的按鍵 (例如， Enter 鍵及 PF1)

- 將訊息傳遞至應用程式時游標所在的位置
- 屬性欄位是否已由 IMS 應用程式設定
  - 高、正常或零明暗度
  - 顏色
  - IMS 是否在下一次按下 Enter 鍵時預期回該欄位
- IMS 應用程式是否在任何欄位中使用空值字元 (X'3F')。

如果 IMS 訊息只包含字元資料 (LLZZ 資料區段除外)，且您使用 MQIIH 結構，請將 MQMD 格式設為 MQFMT_IMS，並將 MQIIH 格式設為 MQFMT_IMS_VAR_STRING。

如果 IMS 訊息只包含字元資料 (LLZZ 資料區段除外)，且您 不 使用 MQIIH 結構，請將 MQMD 格式設為 MQFMT_IMS_VAR_STRING，並確定 IMS 應用程式在回覆時指定 MODname MQFMT_IMS_VAR_STRING。 如果發生問題 (例如，使用者未獲授權使用交易)，且 IMS 傳送錯誤訊息，則其格式為 DFSMOx 的 MODname，其中 x 是 1 到 5 範圍內的數字。 這會放在 MQMD.Format。

如果 IMS 訊息包含二進位、聚集或浮點資料 (LLZZ 資料區段除外)，請撰寫您自己的資料轉換常式的程式碼。 如需 IMS 畫面格式化的相關資訊，請參閱 *IMS/ESA Application Programming: Transaction Manager* 。

撰寫程式碼以透過 IBM MQ 處理 IMS 交易時，請考量下列主題。

- 第 764 頁的『撰寫 IBM MQ 應用程式以呼叫 IMS 交談式交易』
- 第 764 頁的『撰寫包含 IMS 指令的程式』
- 第 765 頁的『觸發』

撰寫 IBM MQ 應用程式以呼叫 IMS 交談式交易

撰寫 IBM MQ 應用程式以呼叫 IMS 交談式交易時，請使用此資訊作為考量的指引。

當您撰寫應用程式來呼叫 IMS 交談時，請考量下列事項:

- 在應用程式訊息中包含 MQIIH 結構。
- 將 MQIIH 中的 *CommitMode* 設為 MQICM_SEND_THEN_COMMIT。
- 若要呼叫新的交談，請將 MQIIH 中的 *TranState* 設為 MQITS_NOT_IN_CONVERSATION。
- 若要呼叫交談的第二個及後續步驟，請將 *TranState* 設為 MQITS_IN_CONVERSATION，並將 *TranInstanceId* 設為在交談的前一個步驟中所傳回欄位的值。
- 如果您遺失從 IMS 傳送的原始訊息，則在 IMS 中很難找到 *TranInstanceId* 的值。
- 應用程式必須檢查來自 IMS 的訊息 *TranState*，以檢查 IMS 交易是否已終止交談。
- 您可以使用 /EXIT 來結束交談。 您也必須以引號括住 *TranInstanceId*，將 *TranState* 設為 MQITS_IN_CONVERSATION，並使用正在其上執行交談的 IBM MQ 佇列。
- 您無法使用 /HOLD 或 /REL 來保留或釋放交談。
- 如果重新啟動 IMS，則會終止透過 IBM MQ - IMS 橋接器呼叫的交談。

撰寫包含 IMS 指令的程式

應用程式可以建置格式為 LLZZ 指令的 IBM MQ 訊息，而不是交易，其中 指令 的格式為 /DIS TRAN PART 或 /DIS POOL ALL。

大部分 IMS 指令可以透過此方式發出; 如需詳細資料，請參閱 *IMS V11* 通訊及連線 。 指令輸出會以文字格式在 IBM MQ 回覆訊息中接收，並傳送至 3270 終端機以供顯示。

OTMA 已實作 IMS 顯示交易指令的特殊形式，它會傳回輸出的架構形式。 確切的格式定義在 *IMS V11* 通訊及連線中。 若要從 IBM MQ 訊息呼叫此表單，請像之前一樣建置訊息資料 (例如 /DIS TRAN PART)，並將 MQIIH 中的 TranState 欄位設為 MQITS_ARCHITECTED。 IMS 會處理指令，並以架構形式傳回回覆。 架構式回應包含以輸出文字形式可找到的所有資訊，以及另一項資訊: 交易定義為可回復或不可回復。

觸發

IBM MQ - IMS 橋接器不支援觸發訊息。

如果您定義的起始佇列使用具有 XCF 參數的儲存類別，則當訊息到達橋接器時，會拒絕放入該佇列的訊息。

撰寫用戶端程序化應用程式

使用程序化語言在 IBM MQ 上撰寫用戶端應用程式所需的資訊。

應用程式可以在 IBM MQ 用戶端環境中建置及執行。 應用程式必須建置並鏈結至所使用的 IBM MQ MQI client 。 應用程式建置及鏈結的方式會根據所使用的平台及程式設計語言而有所不同。 如需如何建置用戶端應用程式的相關資訊，請參閱 第 769 頁的『建置 IBM MQ MQI clients 的應用程式』。

只要符合特定條件，您就可以在完整 IBM MQ 環境及 IBM MQ MQI client 環境中執行 IBM MQ 應用程式，而無需變更程式碼。 如需在 IBM MQ 用戶端環境中執行應用程式的相關資訊，請參閱 第 771 頁的『在 IBM MQ MQI client 環境中執行應用程式』。

如果您使用訊息佇列介面 (MQI) 來撰寫要在 IBM MQ MQI client 環境中執行的應用程式，則在 MQI 呼叫期間，會強制實施一些額外的控制，以確保 IBM MQ 應用程式處理不會中斷。 如需這些控制項的相關資訊，請參閱 第 765 頁的『在用戶端應用程式中使用 MQI』。

如需準備及執行其他應用程式類型作為用戶端應用程式的相關資訊，請參閱下列主題:

- 第 781 頁的『準備及執行 CICS 和 Tuxedo 應用程式』
- 第 41 頁的『準備及執行 Microsoft Transaction Server 應用程式』
- 第 784 頁的『準備及執行 IBM MQ JMS 應用程式』

相關概念

第 6 頁的『應用程式開發概念』
您可以使用選擇的程序化或物件導向語言來撰寫 IBM MQ 應用程式。 在開始設計及撰寫 IBM MQ 應用程式之前，請先熟悉基本 IBM MQ 概念。

第 5 頁的『開發適用於 IBM MQ 的應用程式』
您可以開發應用程式來傳送及接收訊息，以及管理佇列管理程式和相關資源。 IBM MQ 支援以許多不同語言及架構撰寫的應用程式。

第 41 頁的『IBM MQ 應用程式的設計考量』
當您決定應用程式如何利用可供您使用的平台及環境時，您需要決定如何使用 IBM MQ 所提供的特性。

第 603 頁的『撰寫佇列作業的程序化應用程式』
請利用這項資訊來瞭解如何撰寫佇列作業應用程式、連接佇列管理程式、發佈/訂閱，以及開啟和關閉物件。

第 676 頁的『撰寫發佈/訂閱應用程式』
開始撰寫發佈/訂閱 IBM MQ 應用程式。

第 837 頁的『建置程序化應用程式』
您可以使用數種程序化語言之一來撰寫 IBM MQ 應用程式，並在數個不同的平台上執行該應用程式。

第 871 頁的『處理程序化程式錯誤』
此資訊說明與應用程式 MQI 呼叫相關聯的錯誤，當它進行呼叫時，或當它的訊息遞送至其最終目的地時。

相關工作

第 887 頁的『使用 IBM MQ 範例程序化程式』
這些範例程式是以程序化語言撰寫，並示範「訊息佇列介面 (MQI)」的一般用法。 不同平台上的 IBM MQ 程式。

在用戶端應用程式中使用 MQI

此主題集合考量撰寫 IBM MQ 應用程式以在訊息佇列介面 (MQI) 用戶端環境中執行，以及在完整 IBM MQ 佇列管理程式環境中執行之間的差異。

當您設計應用程式時，請考量在 MQI 呼叫期間需要強制的控制項，以確保 IBM MQ 應用程式處理不會中斷。

您必須先建立特定 IBM MQ 物件，才能執行使用 MQI 的應用程式。 如需相關資訊，請參閱 使用 MQI 的應用程式。

限制用戶端應用程式中的訊息大小

佇列管理程式具有訊息長度上限，但通道定義會限制您可以從用戶端應用程式傳輸的訊息大小上限。

佇列管理程式的訊息長度上限 (MaxMsg 長度) 屬性是該佇列管理程式可以處理的訊息長度上限。

Multi 在 多平台上，您可以增加佇列管理程式的訊息長度上限屬性。 如需相關資訊，請參閱 ALTER QMGR。

您可以使用 MQINQ 呼叫來找出佇列管理程式的 MaxMsg 長度值。

如果變更 MaxMsg 長度屬性，則不會檢查是否已有長度大於新值的佇列，甚至訊息。 變更此屬性之後，請重新啟動應用程式及通道，以確保變更生效。 因此無法產生任何超出佇列管理程式或佇列的 MaxMsg 長度的新訊息 (除非容許佇列管理程式分段)。

通道定義中的訊息長度上限會限制您可以沿著用戶端連線傳輸的訊息大小。 如果 IBM MQ 應用程式嘗試使用 MQPUT 呼叫或 MQGET 呼叫，且訊息大於此值，則會傳回錯誤碼給應用程式。 通道定義的訊息大小上限參數不會影響可透過用戶端連線使用 MQCB 來耗用的訊息大小上限。

相關概念

第 769 頁的『使用 MQCONNX』
您可以使用 MQCONNX 呼叫來指定 MQCNO 結構中的通道定義 (MQCD) 結構。

相關參考

訊息長度上限 (MAXMSGL)
ALTER CHANNEL
2010 (07DA) (RC2010): MQRC_DATA_LENGTH_ERROR

選擇用戶端或伺服器 *CCSID*

使用用戶端的區域編碼字集 ID (CCSID)。 佇列管理程式會執行必要的轉換。 您可以使用 **MQCCSID** 環境變數來置換 CCSID。 如果您的應用程式執行多個 PUT，則在完成第一個 PUT 之後，可以改寫 MQMD 的 CCSID 及編碼欄位。

透過訊息佇列介面 (MQI) 從應用程式傳遞至用戶端 Stub 的資料必須使用針對 IBM MQ MQI client 編碼的本端 CCSID。 如果連接的佇列管理程式需要轉換資料，則由佇列管理程式上的用戶端支援程式碼完成轉換。

在 IBM WebSphere MQ 7.0 以及更新版本中，如果佇列管理程式無法執行轉換，則 Java 用戶端可以執行轉換。 請參閱第 313 頁的『IBM MQ classes for Java 用戶端連線』。

用戶端程式碼會假設用戶端中 MQI 的字元資料是在針對該工作站所配置的 CCSID 中。 如果此 CCSID 是不受支援的 CCSID 或不是必要的 CCSID，則可以使用下列其中一個指令，以 **MQCCSID** 環境變數置換它:

- Windows

```
SET MQCCSID=850
```

- Linux AIX

```
export MQCCSID=850
```

- IBM i

```
ADDENVVAR ENVVAR(MQCCSID) VALUE(37)
```

如果在設定檔中設定此參數，則會假設所有 MQI 資料都使用字碼頁 850。

註: 關於字碼頁 850 的假設不適用於訊息中的應用程式資料。

如果您的應用程式正在執行在訊息描述子 (MQMD) 之後包含 IBM MQ 標頭的多個 PUT，請注意，在完成第一個 PUT 之後，會改寫 MQMD 的 CCSID 及編碼欄位。

在第一個 PUT 之後，這些欄位包含連接的佇列管理程式用來轉換 IBM MQ 標頭的值。 請確定您的應用程式將值重設為它需要的值。

在用戶端應用程式中使用 *MQINQ*

用戶端程式碼會修改使用 MQINQ 查詢的部分值。

CCSID
設為用戶端 CCSID，而不是佇列管理程式的 CCSID。

MaxMsgLength
如果受到通道定義的限制，則會減少。 這將是下列項目的較低部分:

- 佇列定義中定義的值，或
- 通道定義中定義的值

如需相關資訊，請參閱 MQINQ。

在用戶端應用程式中使用同步點協調

在基本用戶端上執行的應用程式可以發出 MQCMIT 及 MQBACK，但同步點控制的範圍限制為 MQI 資源。 您可以將外部交易管理程式與延伸交易式用戶端搭配使用。

在 IBM MQ 內，佇列管理程式的其中一個角色是應用程式內的同步點控制。 如果應用程式在 IBM MQ 基本用戶端上執行，則它可以發出 MQCMIT 及 MQBACK，但同步點控制的範圍僅限於 MQI 資源。 IBM MQ 動詞 MQBEGIN 在基本用戶端環境中無效。

在伺服器上完整佇列管理程式環境中執行的應用程式可以透過交易監視器協調多個資源 (例如資料庫)。 在伺服器上，您可以使用 IBM MQ 產品隨附的「交易監視器」，或另一個交易監視器 (例如 CICS)。 您無法將交易監視器與基本用戶端應用程式搭配使用。

您可以將外部交易管理程式與 IBM MQ 延伸交易式用戶端搭配使用。 請參閱 何謂延伸交易式用戶端? 以取得相關詳細資料。

在用戶端應用程式中使用先讀

您可以在用戶端上使用先讀，以容許將非持續訊息傳送至用戶端，而無需用戶端應用程式要求訊息。

當用戶端需要來自伺服器的訊息時，它會將要求傳送至伺服器。 它會針對所耗用的每一個訊息傳送個別要求。 若要透過避免必須傳送這些要求訊息來改善耗用非持續訊息之用戶端的效能，可以將用戶端配置為使用先讀。 先讀容許將訊息傳送至用戶端，而無需應用程式要求它們。

使用先讀可以改善耗用來自用戶端應用程式的非持續訊息時的效能。 此效能改進同時適用於 MQI 及 JMS 應用程式。 使用 MQGET 或非同步使用的用戶端應用程式在使用非持續訊息時，會因效能改善而受益。

當呼叫 MQOPEN 與 MQOO_READ_AHEAD 時，IBM MQ 用戶端只會在符合特定條件時啟用先讀。 這些條件包括:

- 用戶端應用程式必須與執行緒 IBM MQ MQI 用戶端程式庫進行編譯及鏈結。
- 用戶端通道必須使用 TCP/IP 通訊協定。
- 該通道必須在用戶端和伺服器通道定義中都具有非零 SharingConversations (SHARECNV) 設定。

啟用先讀時，訊息會傳送至用戶端上稱為先讀緩衝區的記憶體緩衝區。 對於已開啟且已啟用先讀的每一個佇列，用戶端具有先讀緩衝區。 先讀緩衝區中的訊息不會持續保存。 用戶端會定期以伺服器所耗用資料量的相關資訊來更新伺服器。

並非所有用戶端應用程式設計都適合使用先讀，因為並非所有選項都支援使用。 當啟用先讀時， MQGET 呼叫之間的部分選項必須一致。 如果用戶端在 MQGET 呼叫之間變更其選取準則，則儲存在先讀緩衝區中的訊息仍會停留在用戶端先讀緩衝區中。 如需相關資訊，請參閱 第 658 頁的『增進非持續訊息的效能』

先讀配置由三個屬性 (MaximumSize、 PurgeTime 及 UpdatePercentage) 所控制，這些屬性指定在 IBM MQ 用戶端配置檔的 MessageBuffer 段落中。

在用戶端應用程式中使用非同步放置

使用非同步放置，應用程式可以將訊息放置到佇列中，而無需等待佇列管理程式的回應。 在某些狀況下，您可以使用此項目來改善傳訊效能。

通常，當應用程式使用 MQPUT 或 MQPUT1 將訊息放入佇列時，應用程式必須等待佇列管理程式確認它已處理 MQI 要求。 您可以增進傳訊效能，特別是對於使用用戶端連結的應用程式，以及將大量小型訊息放入佇列的應用程式，方法是選擇以非同步方式放置訊息。 當應用程式非同步放置訊息時，佇列管理程式不會傳回每一個呼叫的成功或失敗，但您可以改為定期檢查錯誤。

若要非同步將訊息放置在佇列上，請在 MQPMO 結構的 *Options* 欄位中使用 MQPMO_ASYNC_RESPONSE 選項。

如果訊息不適用於非同步放置，則會同步放置到佇列中。

當要求 MQPUT 或 MQPUT1 的非同步放置回應時， CompCode 及 MQCC_OK 和 MQRC_NONE 的原因不一定表示訊息已順利放入佇列。 雖然每一個個別 MQPUT 或 MQPUT1 呼叫的成功或失敗可能不會立即傳回，但稍後可以透過對 MQSTAT 的呼叫來判斷非同步呼叫下發生的第一個錯誤。

如需 MQPMO_ASYNC_RESPONSE 的詳細資料，請參閱 MQPMO 選項。

「非同步放置」範例程式示範一些可用的特性。 如需程式的特性和設計以及如何執行程式的詳細資料，請參閱 第 904 頁的『非同步 Put 範例程式』。

在用戶端應用程式中使用共用交談

在允許共用交談的環境中，交談可以共用 MQI 通道實例。

共用交談由兩個欄位（都稱為 SharingConversations）控制，其中一個是通道定義 (MQCD) 結構的一部分，另一個是通道結束程式參數 (MQCXP) 結構的一部分。 MQCD 中的 SharingConversations 欄位是整數值，用於判斷可共用與通道相關聯的通道實例的交談數上限。 MQCXP 中的 SharingConversations 欄位是布林值，用於指出目前是否共用通道實例。

在不允許共用交談的環境中，指定相同 MQCD 的新用戶端連線不會共用通道實例。

滿足下列條件時，新的用戶端應用程式連線會共用通道實例：

- 針對共用交談同時配置通道實例的用戶端連線及伺服器連線端，且通道結束程式未置換這些值。
- 用戶端連線 MQCD 值（用戶端 MQCONNX 呼叫上或用戶端通道定義表 (CCDT) 中提供的）與在最初建立現有通道實例時在用戶端 MQCONNX 呼叫上或 CCDT 中提供的用戶端連線 MQCD 值完全相符。 請注意，原始 MQCD 可能後來已被結束程式或通道協議變更，但在進行這些變更之前，會根據提供給用戶端系統的值尋找相符項。
- 未超出伺服器端的共用交談限制。

如果新的用戶端應用程式連線符合用於將通道實例與其他交談共用的準則，則在對該交談呼叫任何結束程式之前進行此決策。 此類交談上的結束程式無法變更以下事實：它將通道實例與其他交談共用。 如果不存在與新的通道定義相符的現有通道實例，則會連接新的通道實例。

僅通道實例上的第一次交談會進行通道協議；在該階段，通道實例的協議值是固定的，在後續的交談啟動時無法變更。 TLS 鑑別也只在第一次交談時才會發生。

如果在對通道實例的用戶端連線或伺服器連線端的 Socket 上的第一次交談起始設定任何安全、傳送或接收結束程式期間變更 MQCD SharingConversations 值，則在起始設定所有這些結束程式之後它具有的新值將用於判斷通道實例的共用交談值（最低值優先）。

如果共用交談的協議值為零，則永不共用通道實例。 將此欄位設定為零的進一步結束程式將同樣在自己的通道實例上執行。

如果共用交談的協議值大於零，則 MQCXP SharingConversations 設定為 TRUE 以後續呼叫結束程式，指出可以使用此通道實例同時輸入此通道實例上的其他結束程式。

當您撰寫通道結束程式時，請考量它是否在可能涉及共用交談的通道實例上執行。 如果通道實例可能涉及共用交談，請考量對變更的 MQCD 欄位之其他通道結束程式實例的影響；所有 MQCD 欄位具有所有共用交談間的共用值。 在建立通道實例之後，如果結束程式嘗試變更 MQCD 欄位，則可能會發生問題，因為通道實例上執行的結束程式的其他實例可能同時在嘗試變更相同的欄位。 如果結束程式可能發生此狀況，您必須序列化存取結束碼中的 MQCD。

如果您正在使用定義為共用交談的通道，但您不希望在特定通道實例中進行共用，請在通道實例的第一次交談中起始設定通道結束程式時，將 SharingConversations 的 MQCD 值設定為 1 或 0。 如需 SharingConversations 值的相關說明，請參閱 SharingConversations。

範例

已啟用共用交談。

您使用用於指定結束程式的用戶端連線通道定義。

此通道第一次啟動時，結束程式會在起始設定時變更部分 MQCD 參數。 這些參數由通道執行，因此，通道執行使用的定義現在與最初提供的通道不同。 MQCXP SharingConversations 參數設定為 TRUE。

下次應用程式使用此通道連接時，交談會在先前啟動的通道實例上執行，因為它具有相同的原始通道定義。應用程式第二次連接的通道實例與第一次連接的實例相同。 因此，它使用結束程式變更的定義。 當針對第二次交談起始設定結束程式時，雖然它可以變更 MQCD 欄位，但通道不會處理它們。 這些相同的特性會套用至共用通道實例的任何後續交談。

使用 *MQCONNX*

您可以使用 MQCONNX 呼叫來指定 MQCNO 結構中的通道定義 (MQCD) 結構。

這可讓呼叫用戶端應用程式在執行時期指定用戶端連線通道的定義。 如需相關資訊，請參閱 使用 MQCNO 在 IBM MQ MQI client 上建立用戶端連線通道。 當您使用 MQCONNX 時，在伺服器發出的呼叫取決於伺服器層次和接聽器配置。

當您從用戶端使用 MQCONNX 時，會忽略下列選項:

- MQCN_STANDARD_BINDING
- MQCNO_FASTPATH_BINDING

您可以使用的 MQCD 結構視您使用的 MQCD 版本號碼而定。 如需 MQCD 版本 (MQCD_VERSION) 的相關資訊，請參閱 MQCD 版本。 例如，您可以使用 MQCD 結構，將通道結束程式傳遞至伺服器。 如果您是使用 MQCD 第 3 版或更新版本，則可以使用結構將結束程式陣列傳遞至伺服器。 您可以使用此功能，為每一個作業新增一個結束程式，而非修改現有的結束程式，對相同的訊息執行多個作業 (例如加密和壓縮)。 如果您未在 MQCD 結構中指定陣列，則會檢查單一結束程式欄位。 如需通道結束程式的相關資訊，請參閱 第 805 頁的『傳訊通道的通道結束程式』。

MQCONNX 上的共用連線控點

您可以使用共用連線控點，在相同處理程序內的不同執行緒之間共用控點。

當您指定共用連線控點時，從 MQCONNX 呼叫傳回的連線控點可以在處理程序中任何執行緒上的後續 MQI 呼叫中傳遞。

註: 您可以在 IBM MQ MQI client 上使用共用連線控點，來連接不支援共用連線控點的伺服器佇列管理程式。

如需相關資訊，請參閱第 769 頁的『使用 MQCONNX』。

建置 IBM MQ MQI clients 的應用程式

應用程式可以在 IBM MQ MQI client 環境中建置及執行。 應用程式必須建置並鏈結至所使用的 IBM MQ MQI client 。 應用程式建置及鏈結的方式會根據所使用的平台及程式設計語言而有所不同。

如果要在用戶端環境中執行應用程式，您可以使用下表中顯示的語言來撰寫應用程式:

表 *133:* 用戶端環境中支援的程式設計語言

用戶端平台	C	C++	COBOL	pTAL	RPG	Visual Basic
AIX	是	是	是			
IBM i	是		是		是	

表 133: 用戶端環境中支援的程式設計語言 (繼續)						
用戶端平台	**C**	**C++**	**COBOL**	**pTAL**	**RPG**	**Visual Basic**
Linux Linux	是	是	是			
Windows Windows	是	是	是			是

Multi 鏈結 *C* 應用程式與 *IBM MQ MQI client* 程式碼

撰寫您要在 IBM MQ MQI client 上執行的 IBM MQ 應用程式之後，您必須將它鏈結至 IBM MQ MQI client 程式碼。

您可以透過兩種方式將應用程式鏈結至 IBM MQ MQI client 程式碼:

1. 直接將應用程式連接至佇列管理程式，在此情況下，佇列管理程式必須與應用程式位於同一部機器上。
2. 用戶端程式庫檔案，可讓您存取相同或不同機器上的佇列管理程式。

IBM MQ 提供每一個環境的用戶端程式庫檔案:

AIX
libmqic.a 程式庫 (適用於非執行緒應用程式) 或 libmqic_r.a 程式庫 (適用於執行緒應用程式)。

Linux
libmqic.so 程式庫 (適用於非執行緒作業應用程式) 或 libmqic_r.so 程式庫 (適用於執行緒作業應用程式)。

IBM i
連結用戶端應用程式與非執行緒應用程式的 LIBMQIC 用戶端服務程式，或連結執行緒應用程式的 LIBMQIC_R 服務程式。

Windows
MQIC32.LIB.

ALW 鏈結 *C++* 應用程式與 *IBM MQ MQI client* 程式碼

您可以撰寫應用程式，以便在 C++ 中的用戶端上執行。建置方法會因環境而異。

如需如何鏈結 C++ 應用程式的相關資訊，請參閱 建置 IBM MQ C++ 程式。

如需使用 C++ 的所有層面的完整資料，請參閱 使用 C++

Multi 將 *COBOL* 應用程式與 *IBM MQ MQI client* 程式碼鏈結

撰寫您要在 IBM MQ MQI client 上執行的 COBOL 應用程式之後，您必須將它鏈結至適當的程式庫。

IBM MQ 提供每一個環境的用戶端程式庫檔案:

AIX
將無執行緒 COBOL 應用程式與程式庫 libmqicb.a 鏈結在一起，或將 COBOL 應用程式與 libmqicb_r.a 鏈結在一起。

IBM i
將 COBOL 用戶端應用程式與非執行緒應用程式的 AMQCSTUB 服務程式連結，或與執行緒應用程式的 AMQCSTUB_R 服務程式連結。

Windows
將應用程式碼與 32 位元 COBOL 的 MQICCBB 程式庫鏈結。 IBM MQ MQI client for Windows 不支援 16 位元 COBOL。

Windows 鏈結 *Visual Basic* 應用程式與 *IBM MQ MQI client* 程式碼

您可以將 Microsoft Visual Basic 應用程式與 Windows 上的 IBM MQ MQI client 程式碼相鏈結。

Deprecated

從 IBM MQ 9.0 開始， Microsoft Visual Basic 6.0 的支援已淘汰。 .NET 的 IBM MQ 類別是建議的取代技術。 如需相關資訊，請參閱 開發 .NET 應用程式。

使用下列併入檔來鏈結 Visual Basic 應用程式:

CMQB.bas
MQI

CMQBB.bas
MQAI

CMQCFB.bas
PCF 指令

CMQXB.bas
通道

在 Visual Basic 編譯器中為用戶端設定 `mqtype=2` ，以確保正確自動選取用戶端 dll:

MQIC32.dll
Windows 7、 Windows 8、 Windows 2008 及 Windows 2012

相關概念
第 883 頁的『在 Visual Basic 中撰寫程式碼』
在 Microsoft Visual Basic 中編碼 IBM MQ 程式時要考量的資訊。 Visual Basic 僅在 Windows 上受支援。

第 856 頁的『在 Windows 中準備 Visual Basic 程式』
在 Windows 上使用 Microsoft Visual Basic 程式時要考量的資訊。

在 IBM MQ MQI client 環境中執行應用程式

只要符合特定條件，您就可以在完整 IBM MQ 環境及 IBM MQ MQI client 環境中執行 IBM MQ 應用程式，而無需變更程式碼。

這些條件如下:

- 應用程式不需要同時連接多個佇列管理程式。
- 在 `MQCONN` 或 `MQCONNX` 呼叫中，佇列管理程式名稱不會以星號 (*) 作為字首。
- 應用程式不需要使用 哪些應用程式在 IBM MQ MQI client 上執行? 中列出的任何異常狀況。

註: 您在鏈結編輯時使用的程式庫會決定應用程式必須在其中執行的環境。

在 IBM MQ MQI client 環境中工作時，請記住:

- 在 IBM MQ MQI client 環境中執行的每一個應用程式都有自己的伺服器連線。 每次應用程式發出 `MQCONN` 或 `MQCONNX` 呼叫時，都會建立與伺服器的一個連線。
- 應用程式會同步傳送及取得訊息。 這意味著在用戶端發出呼叫的時間與透過網路傳回完成碼和原因碼之間的等待。
- 所有資料轉換都由伺服器完成，但如需置換機器配置的 CCSID 的相關資訊，另請參閱 MQCCSID 。

將 *IBM MQ MQI client* 應用程式連接至佇列管理程式

在 IBM MQ MQI client 環境中執行的應用程式可以各種方式連接至佇列管理程式。 您可以使用環境變數、MQCNO 結構或用戶端定義表。

在 IBM MQ 用戶端環境中執行的應用程式發出 MQCONN 或 MQCONNX 呼叫時，用戶端會識別建立連線的方式。 當應用程式在 IBM MQ 用戶端上發出 MQCONNX 呼叫時， MQI 用戶端程式庫會以下列順序來搜尋用戶端通道資訊:

1. 使用 MQCNO 結構 (如果有提供的話) 的 `ClientConnOffset` 或 `ClientConnPtr` 欄位內容。 這些欄位識別要用作用戶端連線通道定義的通道定義結構 (MQCD)。 可以使用預先連接結束程式來置換連線詳細資料。 如需相關資訊，請參閱 第 831 頁的『使用儲存庫中的預先連接結束程式來參照連線定義』。
2. 如果設定 **MQSERVER** 環境變數，則會使用它所定義的通道。

3. 如果已定義 mqclient.ini 檔，且「通道」段落包含 **ServerConnectionParms** 屬性，則會使用它所定義的通道。 如需相關資訊，請參閱 IBM MQ MQI client 配置檔的 mqclient.ini 及 用戶端配置檔的通道段落。
4. 如果已設定 **MQCHLLIB** 及 **MQCHLTAB** 環境變數，則會使用它們所指向的用戶端通道定義表。 或者，從 IBM MQ 9.0， **MQCCDTURL** 環境變數提供相等的功能來設定 **MQCHLLIB** 與 **MQCHLTAB** 環境變數的組合。 如果設定 **MQCCDTURL** ，則會使用它所指向的用戶端通道定義表。 如需相關資訊，請參閱 URL 存取 CCDT。
5. 如果已定義 mqclient.ini 檔，且「通道」段落包含 **ChannelDefinitionDirectory** 及 **ChannelDefinitionFile** 屬性，則會使用這些屬性來尋找用戶端通道定義表。 如需相關資訊，請參閱 IBM MQ MQI client 配置檔的 mqclient.ini 及 用戶端配置檔的通道段落。
6. 最後，如果未設定環境變數，則用戶端會搜尋用戶端通道定義表，其中包含從 mqs.ini 檔中 AllQueue 段落的 **DefaultPrefix** 屬性建立的路徑及名稱。 如需相關資訊，請參閱 mqs.ini 檔案的 AllQueue 管理程式段落。

   如果搜尋用戶端通道定義表失敗，用戶端會使用下列路徑:

   - Linux AIX 在 AIX and Linux 上: /var/mqm/AMQCLCHL.TAB
   - Windows 在 Windows 上: C:\Program Files\IBM\MQ\amqclchl.tab
   - IBM i 在 IBM i 上: /QIBM/UserData/mqm/@ipcc
   - MQ Appliance 在 IBM MQ Appliance 上: *QMname*_AMQCLCHL.TAB。 它們會出現在 mqbackup:// URI 下。

只有 MQCONNX 呼叫才支援上述清單中說明的第一個選項 (使用 MQCNO 的 ClientConnOffset 或 ClientConnPtr 欄位)。 如果應用程式使用 MQCONN 而非 MQCONNX ，則會以清單中顯示的順序，以其餘五種方式來搜尋通道資訊。 如果用戶端找不到通道資訊，則 MQCONN 或 MQCONNX 呼叫會失敗。

通道名稱 (用於用戶端連線) 必須符合伺服器上定義的伺服器連線通道名稱， MQCONN 或 MQCONNX 呼叫才會成功。

相關概念

用戶端通道定義表的 Web 可定址存取

相關工作

配置伺服器與用戶端之間的連線

相關參考

用戶端通道定義表

MQCNO-連接選項

使用環境變數將用戶端應用程式連接至佇列管理程式

用戶端通道資訊可透過環境變數提供給在用戶端環境中執行的應用程式。

在 IBM MQ MQI client 環境中執行的應用程式可以使用下列環境變數來連接至佇列管理程式:

MQSERVER
: **MQSERVER** 環境變數用來定義最小通道。 **MQSERVER** 指定 IBM MQ 伺服器的位置，以及要使用的通訊方法。

MQCHLLIB
: **MQCHLLIB** 環境變數指定包含用戶端通道定義表 (CCDT) 之檔案的目錄路徑。 該檔案建立在伺服器上，但可以複製到 IBM MQ MQI client 工作站。

MQCHLTAB
: **MQCHLTAB** 環境變數指定包含用戶端通道定義表 (CCDT) 的檔案名稱。

從 IBM MQ 9.0 開始， **MQCCDTURL** 環境變數提供相等的功能來設定 **MQCHLLIB** 與 **MQCHLTAB** 環境變數的組合。 **MQCCDTURL** 可讓您提供檔案、ftp 或 http URL 作為單一值，以從中取得用戶端通道定義表。 如需相關資訊，請參閱 用戶端通道定義表的 Web 可定址存取權。

使用 *MQCNO* 結構將用戶端應用程式連接至佇列管理程式

您可以在通道定義結構 (MQCD) 中指定通道的定義，該結構使用 MQCONNX 呼叫的 MQCNO 結構提供。

如需相關資訊，請參閱 使用 MQCNO 在 IBM MQ MQI client 上建立用戶端連線通道。

使用用戶端通道定義表將用戶端應用程式連接至佇列管理程式
如果您使用 MQSC DEFINE CHANNEL 指令，您提供的詳細資料會放在用戶端通道定義表 (ccdt) 中。 MQCONN 或 MQCONNX 呼叫的 **QMgrName** 參數內容會決定用戶端連接至哪個佇列管理程式。

用戶端會存取此檔案，以決定應用程式將使用的通道。 如果有多個合適的通道定義，通道的選擇會受到用戶端通道加權 (CLNTWGHT) 及連線親緣性 (AFFINITY) 通道屬性的影響。

使用自動用戶端重新連線
您可以透過配置多個元件，讓用戶端應用程式自動重新連接，而無需撰寫任何其他程式碼。

自動用戶端重新連線是行內 作業。 會在用戶端應用程式中的任意時間點自動還原連線，並將用於開啟物件的控點全部還原。

相較之下，手動重新連線需要用戶端應用程式使用 MQCONN 或 MQCONNX 重新建立連線，以及重新開啟物件。 自動用戶端重新連線適合許多（但並非全部）用戶端應用程式。

如需相關資訊，請參閱 自動用戶端重新連線。

用戶端通道定義表的角色
用戶端通道定義表 (CCDT) 包含用戶端連線通道的定義。 如果您的用戶端應用程式可能需要連接至一些替代佇列管理程式，它特別有用。

當您定義佇列管理程式時，會建立用戶端通道定義表。 多個 IBM MQ 用戶端可以使用相同的檔案。

用戶端應用程式使用 CCDT 的方式有許多種。 CCDT 可以複製到用戶端電腦。 您可以將 CCDT 複製到多個用戶端共用的位置。 當 CCDT 仍位於伺服器上時，您可以讓用戶端以共用檔案的形式來存取 CCDT。

從 IBM MQ 9.0 開始，可以在可透過 URI 存取的集中位置管理 CCDT，而不需要個別更新每一個已部署用戶端的 CCDT。

相關概念
用戶端通道定義表的 Web 可定址存取
相關工作
存取用戶端連線通道定義
相關參考
用戶端通道定義表

CCDT 中的佇列管理程式群組
您可以在用戶端通道定義表 (CCDT) 中定義一組連線作為 佇列管理程式群組。 您可以將應用程式連接至屬於佇列管理程式群組的佇列管理程式。 作法是在 MQCONN 或 MQCONNX 呼叫上，以星號作為佇列管理程式名稱的字首。

您可以選擇定義與多部伺服器機器的連線，因為:

- 您想要將用戶端連接至正在執行的一組佇列管理程式中的任何一個，以提高可用性。
- 您想要將用戶端重新連接至其前次順利連接的相同佇列管理程式，但如果連線失敗，請連接至不同的佇列管理程式。
- 如果連線失敗，您希望能夠在用戶端程式中重新發出 MQCONN，以重試與不同佇列管理程式的用戶端連線。
- 如果連線失敗，您想要自動將用戶端連線重新連接至另一個佇列管理程式，而不寫入任何用戶端程式碼。
- 如果待命實例接管，您想要自動將用戶端連線重新連接至多重實例佇列管理程式的不同實例，而不寫入任何用戶端程式碼。
- 您想要在多個佇列管理程式之間平衡用戶端連線，而連接至部分佇列管理程式的用戶端多於其他佇列管理程式的用戶端。
- 您想要將許多用戶端連線的重新連線分散到多個佇列管理程式，並在一段時間內，以防大量連線導致失敗。
- 您想要能夠在不變更任何用戶端應用程式碼的情況下移動佇列管理程式。
- 您想要撰寫不需要知道佇列管理程式名稱的用戶端應用程式。

不一定適合連接至不同的佇列管理程式。 例如， WebSphere Application Server 中的延伸交易式用戶端或 Java 用戶端可能需要連接至可預測的佇列管理程式實例。 IBM MQ classes for Java 不支援自動重新連接用戶端。

佇列管理程式群組是在用戶端通道定義表 (CCDT) 中定義的一組連線。 該集由其成員在其通道定義中具有相同 **QMNAME** 屬性值的成員所定義。

第 774 頁的圖 97 是用戶端連線表格的圖形表示法，其中顯示三個佇列管理程式群組，兩個在 CCDT 中寫入為 **QMNAME**（QM1）及 **QMNAME**（QMGrp1）的具名佇列管理程式群組，以及一個寫入為 **QMNAME**（''）的空白或預設群組。

1. 佇列管理程式群組 QM1 有三個用戶端連線通道，將它連接至佇列管理程式 QM1 及 QM2。 QM1 可能是位於兩個不同伺服器上的多重實例佇列管理程式。
2. 預設佇列管理程式群組有六個用戶端連線通道將它連接至所有佇列管理程式。
3. QMGrp1 具有兩個佇列管理程式的用戶端連線通道: QM4 及 QM5。

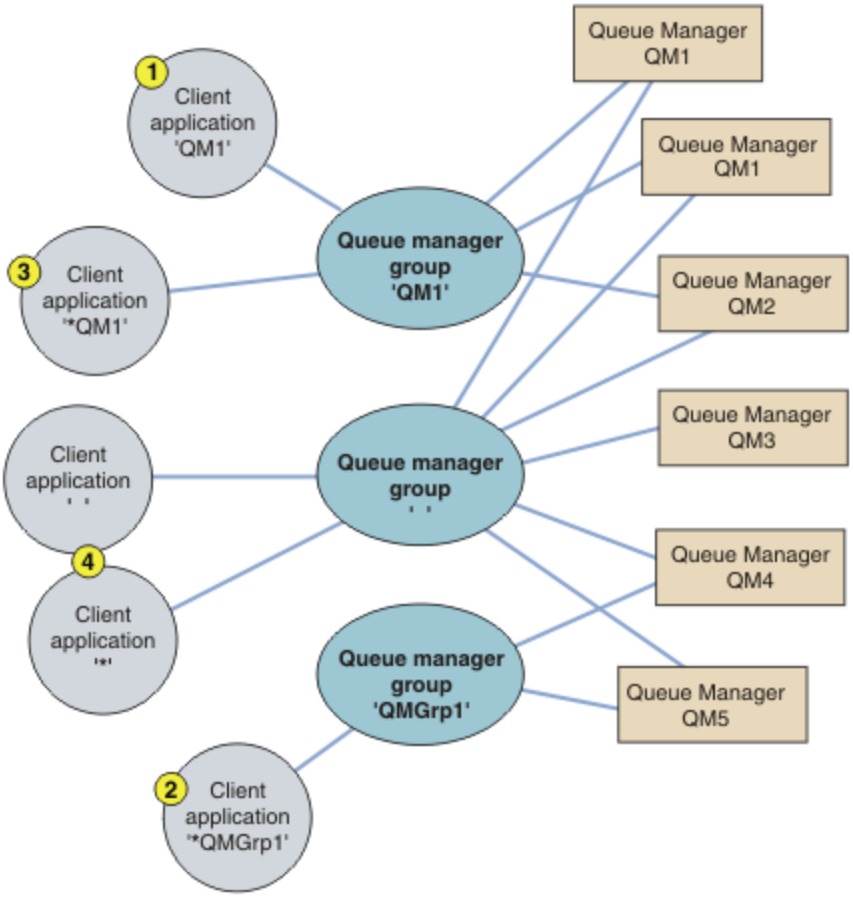


圖 97: 佇列管理程式群組

在 第 774 頁的圖 97 中已編號的用戶端應用程式的協助下，說明了四個使用此用戶端連線表格的範例。

1. 在第一個範例中，用戶端應用程式會將佇列管理程式名稱 QM1 作為 **QmgrName** 參數傳遞至其 MQCONN 或 MQCONNX MQI 呼叫。 IBM MQ 用戶端程式碼會選取相符的佇列管理程式群組 QM1。 該群組包含三個連線通道， IBM MQ MQI client 會嘗試輪流使用其中每一個通道連接至 QM1 ， 直到它找到 IBM MQ 接聽器，以連接至稱為 QM1 的執行中佇列管理程式。

   連線嘗試的順序視用戶端連線 AFFINITY 屬性的值及用戶端通道加權而定。 在這些限制內，連線嘗試的順序會隨機化，同時在三個可能的連線上，以及在一段時間內，以分散建立連線的負載。

   當建立與 QM1 執行中實例的連線時，用戶端應用程式所發出的 MQCONN 或 MQCONNX 呼叫會成功。
2. 在第二個範例中，用戶端應用程式會將字首為星號 *QMGrp1 作為 **QmgrName** 參數的佇列管理程式名稱傳遞至其 MQCONN 或 MQCONNX MQI 呼叫。 IBM MQ 用戶端會選取相符的佇列管理程式群組 QMGrp1。 此

群組包含兩個用戶端連線通道， IBM MQ MQI client 會嘗試依序使用每一個通道連接至 任何 佇列管理程式。 在此範例中， IBM MQ MQI client 需要成功建立連線; 它所連接的佇列管理程式名稱並不重要。

進行連線嘗試的順序規則與之前相同。 唯一的差異是透過在佇列管理程式名稱前面加上星號，用戶端會指出佇列管理程式的名稱不相關。

當建立與 QMGrp1 佇列管理程式群組中通道所連接之任何佇列管理程式的執行中實例的連線時，用戶端應用程式所發出的 MQCONN 或 MQCONNX 呼叫會成功。

3. 第三個範例基本上與第二個範例相同，因為 **QmgrName** 參數以星號 *QM1 作為字首。 此範例說明您無法自行檢查一個通道定義中的 QMNAME 屬性，來判斷用戶端通道連線要連接的佇列管理程式。 通道定義的 **QMNAME** 屬性是 QM1，不足以要求與稱為 QM1 的佇列管理程式建立連線。 如果用戶端應用程式以星號作為其 **QmgrName** 參數的字首，則任何佇列管理程式都是可能的連線目標。

   在此情況下，當建立與 QM1 或 QM2 的執行中實例的連線時，用戶端應用程式發出的 MQCONN 或 MQCONNX 呼叫會成功。

4. 第四個範例說明預設群組的用法。 在此情況下，用戶端應用程式會將星號 '*'或空白 ''作為 **QmgrName** 參數傳遞至其 MQCONN 或 MQCONNX MQI 呼叫。 依用戶端通道定義中的慣例，空白 **QMNAME** 屬性表示預設佇列管理程式群組，且空白或星號 **QmgrName** 參數符合空白 **QMNAME** 屬性。

   在此範例中，預設佇列管理程式群組具有與所有佇列管理程式的用戶端通道連線。 透過選取預設佇列管理程式群組，應用程式可以連接至群組中的任何佇列管理程式。

   當建立與任何佇列管理程式的執行中實例的連線時，用戶端應用程式所發出的 MQCONN 或 MQCONNX 呼叫會成功。

   註: 雖然應用程式使用空白 **QmgrName** 參數來連接至預設佇列管理程式群組或預設佇列管理程式，但預設群組與預設佇列管理程式不同。 預設佇列管理程式群組的概念只與用戶端應用程式相關，而預設佇列管理程式則與伺服器應用程式相關。

僅在一個佇列管理程式上定義用戶端連線通道，包括連接至第二個或第三個佇列管理程式的那些通道。 請勿在兩個佇列管理程式上定義它們，然後嘗試合併兩個用戶端通道定義表。 用戶端只能存取一個用戶端通道定義表。

範例

請在主題開頭再次查看使用佇列管理程式群組的原因 清單 。 使用佇列管理程式群組如何提供這些功能?

連接至一組佇列管理程式中的任何一個。
使用與集合中所有佇列管理程式的連線來定義佇列管理程式群組，並使用字首為星號的 **QmgrName** 參數來連接至群組。

如果前次連接的佇列管理程式無法使用，請重新連接至相同的佇列管理程式，但連接至不同的佇列管理程式。
像之前一樣定義佇列管理程式群組，但在每一個用戶端通道定義上設定屬性 **AFFINITY**（PREFERRED）。

如果連線失敗，請重試與另一個佇列管理程式的連線。
連接至佇列管理程式群組，並在連線中斷或佇列管理程式失敗時重新發出 MQCONN 或 MQCONNX MQI 呼叫。

如果連線失敗，則自動重新連接至另一個佇列管理程式。
使用 MQCONNX **MQCNO** 選項 MQCNO_RECONNECT 來連接佇列管理程式群組。

自動重新連接至多重實例佇列管理程式的不同實例。
請執行與前述範例相同的動作。 在此情況下，如果您想要限制佇列管理程式群組連接至特定多重實例佇列管理程式的實例，請定義僅連接至多重實例佇列管理程式實例的群組。

您也可以要求用戶端應用程式發出其 MQCONN 或 MQCONNX MQI 呼叫，且不會在 **QmgrName** 參數前面加上星號。 這樣用戶端應用程式就只能連接至指名的佇列管理程式。 最後，您可以將 **MQCNO** 選項設為 MQCNO_RECONNECT_Q_MGR。 此選項接受重新連線至先前連接的相同佇列管理程式。 您也可以使用此值來限制重新連線至一般佇列管理程式的相同實例。

平衡佇列管理程式之間的用戶端連線，其中連接至部分佇列管理程式的用戶端比連接至其他佇列管理程式的用戶端更多。
定義佇列管理程式群組，並在每一個用戶端通道定義上設定 **CLNTWGHT** 屬性，以不均勻地配送連線。

在連線或佇列管理程式失敗之後，將用戶端重新連線負載分散不平均，並將它分散在一段時間。
請執行與前述範例相同的動作。 IBM MQ MQI client 會隨機化佇列管理程式之間的重新連線，並在一段時間內分散重新連線。

移動佇列管理程式而不變更任何用戶端程式碼。
CCDT 會將您的用戶端應用程式與佇列管理程式的位置隔離。 CCDT 是可在用戶端定義、從共用位置讀取或從 Web 伺服器提取的資料檔。 如需相關資訊，請參閱 用戶端通道定義表。

撰寫不知道佇列管理程式名稱的用戶端應用程式。
使用佇列管理程式群組名稱，並建立與組織中用戶端應用程式相關的佇列管理程式群組名稱命名慣例，並反映解決方案的架構，而非佇列管理程式的命名。

z/OS 連接至佇列共用群組

您可以將應用程式連接至屬於佇列共用群組的佇列管理程式。 您可以在 MQCONN 或 MQCONNX 呼叫中使用佇列共用群組名稱而非佇列管理程式名稱來完成此作業。

佇列共用群組有一個最多四個字元的名稱。 該名稱在您的網路中必須是唯一的，且必須不同於任何佇列管理程式名稱。

用戶端通道定義應該使用佇列共用群組通用介面來連接至群組中可用的佇列管理程式。 如需相關資訊，請參閱 將用戶端連接至佇列共用群組。 會進行檢查，以確定接聽器所連接的佇列管理程式是佇列共用群組的成員。

如需共用佇列的相關資訊，請參閱 共用佇列及佇列共用群組。

通道加權及親緣性的範例

這些範例說明如何在使用非零 `ClientChannelWeights` 時選取用戶端連線通道。

`ClientChannel` **加權** 及 `ConnectionAffinity` 通道屬性可控制當有多個適合連線的通道可用時，如何選取用戶端連線通道。 這些通道配置為連接至不同的佇列管理程式，以提供更高可用性及/或工作量平衡。 可能導致連線至數個佇列管理程式之一的 MQCONN 呼叫必須以星號作為佇列管理程式名稱的字首，如 範例 MQCONN 呼叫的範例: 範例 1. 佇列管理程式名稱包括星號 (*)。

連線適用的候選通道是 QMNAME 屬性符合 MQCONN 呼叫中指定之佇列管理程式名稱的那些通道。 如果連線的所有適用通道都具有 `ClientChannel` **加權** 零 (預設值)，則會按字母順序選取它們，如下列範例所示: MQCONN 呼叫範例: 範例 1. 佇列管理程式名稱包括星號 (*)。

下列範例說明使用非零 `ClientChannelWeights` 時所發生的情況。 請注意，由於此特性涉及虛擬隨機通道選擇，因此範例會顯示可能發生的一連串動作，而不是絕對會發生的動作。

範例 *1.* 當 *ConnectionAffinity* 設為 *PREFERRED* 時選取通道

此範例說明 IBM MQ MQI client 如何從 CCDT 中選取通道，其中 ConnectionAffinity 設為 PREFERRED。

在此範例中，許多用戶端機器使用佇列管理程式所提供的「用戶端通道定義表 (CCDT)」。 CCDT 包括具有下列屬性的用戶端連線通道 (使用 DEFINE CHANNEL 指令的語法顯示):

```
CHANNEL(A) QMNAME(DEV) CONNAME(devqm.it.company.example)
CHANNEL(B) QMNAME(CORE) CONNAME(core1.ops.company.example) CLNTWGHT(5) +
AFFINITY(PREFERRED)
CHANNEL(C) QMNAME(CORE) CONNAME(core2.ops.company.example) CLNTWGHT(3) +
AFFINITY(PREFERRED)
CHANNEL(D) QMNAME(CORE) CONNAME(core3.ops.company.example) CLNTWGHT(2) +
AFFINITY(PREFERRED)
```

應用程式發出 MQCONN (*CORE)

通道 A 不是此連線的候選項，因為 QMNAME 屬性不相符。 B、C 和 D 頻道被確定為候選者，並根據其權重按優先次序排列。 在這個範例中，訂單可能是 C、B、D。 用戶端會嘗試連接至位於 core2.ops.company.example 的佇列管理程式。 不會檢查該位址的佇列管理程式名稱，因為 MQCONN 呼叫會在佇列管理程式名稱中包含星號。

請務必注意，使用 `AFFINITY(PREFERRED)`，每次此特定用戶端機器連接時，都會以相同的起始喜好設定順序來放置通道。 即使連線來自不同的處理程序或在不同的時間，也適用此情況。

在此範例中，無法呼叫到位於 core.2.ops.company.example 的佇列管理程式。 用戶端會嘗試連接至 core1.ops.company.example，因為通道 B 是依照喜好設定順序的下一個。 此外，通道 C 會降級成最不偏好的通道。

相同應用程式會發出第二個 MQCONN (*CORE) 呼叫。 前一個連線已降級通道 C，因此現在最偏好的通道是 B。 此連線是建立到 core1.ops.company.example。

第二部共用相同「用戶端通道定義表」的機器會以不同的起始喜好設定順序來放置通道。 例如， D、B、C。 在正常情況下，當所有通道都運作時，這部機器上的應用程式會連接至 core3.ops.company.example，而第一部機器上的應用程式則會連接至 core2.ops.company.example。 這容許在多個佇列管理程式之間進行大量用戶端的工作量平衡，同時容許每一個個別用戶端連接至相同的佇列管理程式 (如果可用的話)。

範例 *2.* 當 *ConnectionAffinity* 設為 *NONE* 時選取通道
這個範例說明 IBM MQ MQI client 如何從 CCDT 中選取通道，其中 ConnectionAffinity 設為 NONE。

在此範例中，許多用戶端使用佇列管理程式所提供的「用戶端通道定義表 (CCDT)」。 CCDT 包括具有下列屬性的用戶端連線通道 (使用 DEFINE CHANNEL 指令的語法顯示):

```
CHANNEL(A) QMNAME(DEV) CONNAME(devqm.it.company.example)
CHANNEL(B) QMNAME(CORE) CONNAME(core1.ops.company.example) CLNTWGHT(5) +
AFFINITY(NONE)
CHANNEL(C) QMNAME(CORE) CONNAME(core2.ops.company.example) CLNTWGHT(3) +
AFFINITY(NONE)
CHANNEL(D) QMNAME(CORE) CONNAME(core3.ops.company.example) CLNTWGHT(2) +
AFFINITY(NONE)
```

應用程式發出 MQCONN (*CORE)。 如同前一個範例，不會考量通道 A，因為 QMNAME 不相符。 根據通道 B、C 或 D 的加權來選取通道 B、C 或 D，機率為 50%、30% 或 20%。 在此範例中，可能會選取通道 B。 未建立喜好設定的持續順序。

發出第二個 MQCONN (*CORE) 呼叫。 同樣地，會選取三個適用通道中的一個，且機率相同。 在此範例中，選擇通道 C。 不過， core2.ops.company.example 沒有回應，因此會在其餘候選通道之間進行另一個選擇。 已選取通道 B，且應用程式已連接至 core1.ops.company.example。

使用 AFFINITY (NONE) 時，每一個 MQCONN 呼叫都與任何其他呼叫無關。 因此，當此範例應用程式建立第三個 MQCONN (*CORE) 時，它可能會再次嘗試透過中斷的通道 C 進行連接，然後再選擇 B 或 D 之一。

MQCONN 呼叫範例
使用 MQCONN 連接至特定佇列管理程式或其中一個佇列管理程式群組的範例。

在下列每一個範例中，網路是相同的; 有一個連線定義為來自相同 IBM MQ MQI client 的兩部伺服器。 (在這些範例中，可以使用 MQCONNX 呼叫來取代 MQCONN 呼叫。)

伺服器機器上有兩個佇列管理程式在執行中，一個名為 SALE，另一個名為 SALE_BACKUP。

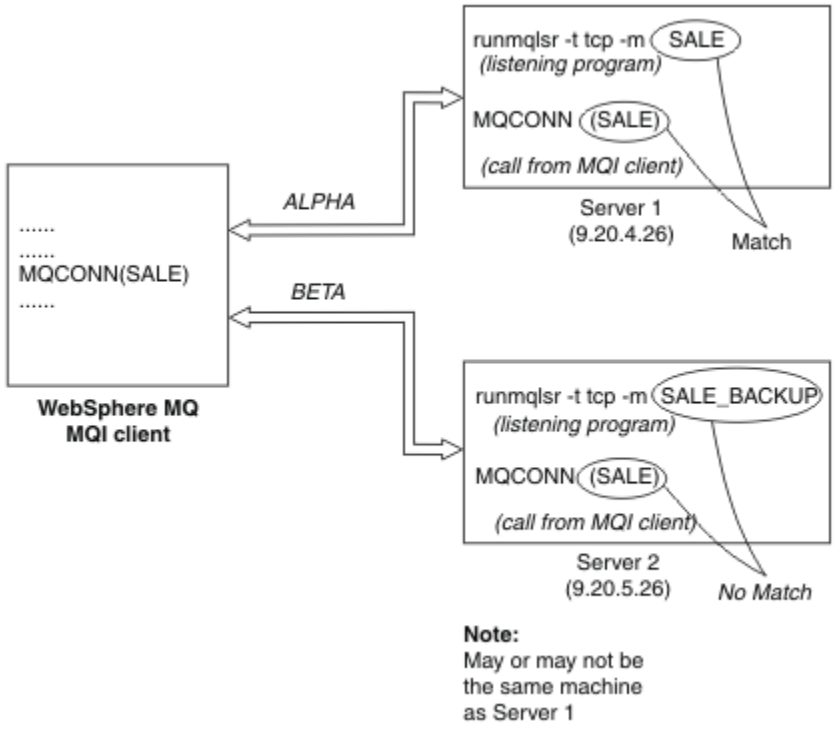


圖 *98: MQCONN* 範例

這些範例中通道的定義如下:

SALE 定義:

```
DEFINE CHANNEL(ALPHA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to IBM MQ MQI client')

DEFINE CHANNEL(ALPHA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.4.26) DESCR('IBM MQ MQI client connection to server 1') +
QMNAME(SALE)

DEFINE CHANNEL(BETA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.5.26) DESCR('IBM MQ MQI client connection to server 2') +
QMNAME(SALE)
```

SALE_BACKUP 定義:

```
DEFINE CHANNEL(BETA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to IBM MQ MQI client')
```

用戶端通道定義可以彙總如下:

名稱	CHLTYPE	TRPTYPE	CONNAME	QMNAME
α	CLNTCONN	TCP	9.20.4.26	出售
測試版	CLNTCONN	TCP	9.20.5.26	出售

MQCONN 範例示範的內容
這些範例示範如何使用多個佇列管理程式作為備份系統。

假設伺服器 1 的通訊鏈結暫時中斷。 示範使用多個佇列管理程式作為備份系統。

每一個範例涵蓋不同的 MQCONN 呼叫，並套用下列規則，以說明在所呈現的特定範例中發生的情況:

1. 用戶端通道定義表 (CCDT) 會依佇列管理程式名稱 (QMNAME 欄位) 的字母通道名稱順序掃描與 MQCONN 呼叫中給定的名稱相對應的佇列管理程式名稱 (QMNAME 欄位)。

2. 如果找到相符項，則會使用通道定義。
3. 嘗試啟動連線名稱 (CONNAME) 所識別之機器的通道。 如果順利完成，應用程式會繼續執行。 它需要:
   - 要在伺服器上執行的接聽器。
   - 要連接至與用戶端要連接之佇列管理程式相同的接聽器 (如果有指定的話)。
4. 如果嘗試啟動通道失敗，且用戶端通道定義表中有多個項目 (在此範例中有兩個項目) ，則會搜尋檔案以找出進一步的相符項。 如果找到相符項，則會在步驟 1 繼續執行處理程序。
5. 如果找不到相符項，或用戶端通道定義表中沒有其他項目，且通道無法啟動，則應用程式無法連接。 MQCONN 呼叫會傳回適當的原因碼和完成碼。 應用程式可以根據傳回的原因及完成碼採取動作。

範例 *1.* 佇列管理程式名稱包括星號 *(*)*
在此範例中，應用程式不關心它所連接的佇列管理程式。 應用程式會對包含星號的佇列管理程式名稱發出 MQCONN 呼叫。 選擇適當的通道。

應用程式問題:

```
MQCONN (*SALE)
```

遵循規則，這是在此實例中發生的情況:

1. 會掃描用戶端通道定義表 (CCDT) ，以找出符合應用程式 MQCONN 呼叫的佇列管理程式名稱 SALE。
2. 找到 ALPHA 和 BETA 的通道定義。
3. 如果一個通道的 CLNTWGHT 值為 0 ，則會選取此通道。 如果兩者都有 CLNTWGHT 值 0 ，則會選取通道 ALPHA ，因為它是按字母順序的第一個通道。 如果兩個通道都有非零 CLNTWGHT 值，則會根據通道的加權隨機選取一個通道。
4. 嘗試啟動通道。
5. 如果已選取通道 BETA ，則嘗試啟動它會成功。
6. 如果已選取通道 ALPHA ，則嘗試啟動它「不會」成功，因為通訊鏈結已中斷。 然後會套用下列步驟:
   a. 佇列管理程式名稱 SALE 的唯一其他通道是 BETA。
   b. 嘗試啟動此通道-已順利完成。
7. 檢查是否有接聽器在執行中，會顯示有一個接聽器在執行中。 它未連接至 SALE 佇列管理程式，但因為 MQI 呼叫參數中包含星號 (*) ，所以不會進行檢查。 應用程式已連接至 SALE_BACKUP 佇列管理程式，並繼續處理。

範例 *2.* 指定的佇列管理程式名稱
在此範例中，應用程式必須連接至特定佇列管理程式。 應用程式對該佇列管理程式名稱發出 MQCONN 呼叫。 選擇適當的通道。

應用程式需要與特定佇列管理程式 (名為 SALE) 的連線，如在 MQI 呼叫中所見:

```
MQCONN (SALE)
```

遵循規則，這是在此實例中發生的情況:

1. 用戶端通道定義表 (CCDT) 會按字母順序掃描佇列管理程式名稱 SALE，以符合應用程式 MQCONN 呼叫。
2. 找到符合的第一個通道定義是 ALPHA。
3. 嘗試啟動通道-因為通訊鏈結已中斷，所以未順利完成。
4. 系統會再次掃描用戶端通道定義表，以找出佇列管理程式名稱 SALE ，並找到通道名稱 BETA 。
5. 嘗試啟動通道-已順利完成。
6. 檢查是否有接聽器在執行中，會顯示有一個接聽器在執行中，但未連接至 SALE 佇列管理程式。
7. 用戶端通道定義表中沒有進一步的項目。 應用程式無法繼續並收到回覆碼 MQRC_Q_MGR_NOT_AVAILABLE。

範例 *3.* 佇列管理程式名稱為空白或星號 *(*)*
在此範例中，應用程式不關心它所連接的佇列管理程式。 應用程式會發出 MQCONN，並指定空白佇列管理程式名稱或星號。 選擇適當的通道。

處理方式與 第 779 頁的『範例 1. 佇列管理程式名稱包括星號 (*)』相同。

註: 如果此應用程式在非 IBM MQ MQI client 的環境中執行，且名稱為空白，則會嘗試連接至預設佇列管理程式。 這不是從用戶端環境執行的情況; 所存取的佇列管理程式是與通道所連接的接聽器相關聯的佇列管理程式。

應用程式問題:

```
MQCONN ("")
```

or

```
MQCONN (*)
```

遵循規則，這是在此實例中發生的情況:

1. 用戶端通道定義表 (CCDT) 會按字母順序掃描，以找出與應用程式 MQCONN 呼叫相符且空白的佇列管理程式名稱。
2. 通道名稱 ALPHA 的項目在 SALE 的定義中具有佇列管理程式名稱。 這不符合 MQCONN 呼叫參數，其需要佇列管理程式名稱為空白。
3. 下一個項目是針對通道名稱 BETA。
4. 定義中的 queue manager name 是 SALE。 這再次不符合 MQCONN 呼叫參數，其需要佇列管理程式名稱為空白。
5. 用戶端通道定義表中沒有進一步的項目。 應用程式無法繼續並收到回覆碼 MQRC_Q_MGR_NOT_AVAILABLE。

在用戶端環境中觸發

在 IBM MQ MQI clients 上執行的 IBM MQ 應用程式所傳送的訊息有助於以與任何其他訊息完全相同的方式進行觸發，而且它們可用來同時在伺服器和用戶端上觸發程式。

觸發會在 觸發通道中詳細說明。

觸發監視器和要啟動的應用程式必須在相同的系統上。

觸發佇列的預設性質與伺服器環境中的性質相同。 特別是，如果在用戶端應用程式中未指定任何 MQPMO 同步點控制選項，將訊息放置到 z/OS 佇列管理程式本端的觸發佇列，則會將訊息放置在工作單元內。 如果隨後符合觸發條件，則觸發訊息會放置在相同工作單元內的起始佇列中，且在工作單元結束之前無法由觸發監視器擷取。 在工作單元結束之前，不會啟動要觸發的處理程序。

程序定義

您必須在伺服器上定義程序定義，因為這與已設定觸發的佇列相關聯。

程序物件定義要觸發的項目。 如果用戶端和伺服器不在相同的平台上執行，則觸發監視器所啟動的任何處理程序都必須定義 *ApplType*，否則伺服器會採用其預設定義 (亦即，通常與伺服器機器相關聯的應用程式類型) 並導致失敗。

例如，如果觸發監視器在 IBM MQ MQI client 上執行，且想要將要求傳送至另一個作業系統上的伺服器，則必須定義 MQAT_WINDOWS_NT，否則其他作業系統會使用其預設定義，且處理程序會失敗。

觸發監視器 *(trigger monitor)*

非 z/OS IBM MQ 產品所提供的觸發監視器在 IBM i IBM i AIX, Linux, and Windows 系統的用戶端環境中執行。

若要執行觸發監視器，請發出下列其中一個指令:

- IBM i 在 IBM i 上:

```
CALL PGM(QMQM/RUNMQTMC) PARM('-m' QmgrName '-q' InitQ)
```

- ALW 在 AIX, Linux, and Windows 平台上:

```
runmqtmc [-m QMgrName] [-q InitQ]
```

預設起始佇列為 SYSTEM.DEFAULT.INITIATION.QUEUE 。 起始佇列是觸發監視器尋找觸發訊息的位置。 然後，它會呼叫程式以取得適當的觸發訊息。 此觸發監視器支援預設應用程式類型，並且與 `runmqtrm` 相同，但它會鏈結用戶端程式庫。

觸發監視器所建置的指令字串如下:

1. 相關程序定義中的 *ApplicId* 。 *ApplicId* 是要執行的程式名稱，因為它會在指令行上輸入。
2. 從起始佇列取得以引號括住的 MQTMC2 結構。 已啟動指令字串，該指令字串完全如所提供，並以引號括住，以便系統指令接受它作為一個參數。
3. 相關程序定義中的 *EnvrData* 。

在啟動觸發監視器的應用程式完成之前，觸發監視器不會查看起始佇列上是否有另一則訊息。 如果應用程式有許多處理程序要執行，觸發監視器可能無法跟上到達的觸發訊息數目。 有兩種方法可以處理此狀況:

1. 有更多觸發監視器在執行中

   如果您選擇讓更多觸發監視器在執行中，您可以控制任何一次可以執行的應用程式數目上限。
2. 在背景中執行已啟動的應用程式

   如果您選擇在背景中執行應用程式， IBM MQ 不會限制可執行的應用程式數目。

To run the started application in the background on AIX and Linux systems, you must put an & (ampersand) at the end of the *EnvrData* of the process definition.

CICS 應用程式 *(*非 *z/OS)*
必須將發出 MQCONN 或 MQCONNX 呼叫的非 z/OS CICS 應用程式定義為「常駐」。 如果重新鏈結 CICS 伺服器應用程式作為用戶端，則可能會失去同步點支援。

必須將發出 MQCONN 或 MQCONNX 呼叫的非 z/OS CICS 應用程式定義為「常駐」。 若要讓常駐碼儘可能小，您可以鏈結至個別程式以發出 MQCONN 或 MQCONNX 呼叫。

如果使用 MQSERVER 環境變數來定義用戶端連線，則必須在 CICSENV.CMD 檔案。

IBM MQ 應用程式可以在 IBM MQ 伺服器環境或 IBM MQ 用戶端上執行，而無需變更程式碼。 However, in an IBM MQ server environment, CICS can act as sync point coordinator, and you use EXEC CICS SYNCPOINT and EXEC CICS SYNCPOINT ROLLBACK rather than **MQCMIT** and **MQBACK**. 如果僅將 CICS 應用程式重新鏈結為用戶端，則會失去同步點支援。 **MQCMIT** 和 **MQBACK** 必須用於在 IBM MQ MQI client 上執行的應用程式。

ALW 準備及執行 CICS 和 Tuxedo 應用程式

若要執行 CICS 及 Tuxedo 應用程式作為用戶端應用程式，您可以使用與伺服器應用程式搭配使用的不同程式庫。 用來執行應用程式的使用者 ID 也不同。

若要準備 CICS 及 Tuxedo 應用程式，以作為 IBM MQ MQI client 應用程式執行，請遵循 配置延伸交易式用戶端中的指示。

不過，請注意，專門處理準備 CICS 和 Tuxedo 應用程式的資訊 (包括 IBM MQ 提供的範例程式) ，假設您準備要在 IBM MQ 伺服器系統上執行的應用程式。 因此，資訊只會參照預期在伺服器系統上使用的 IBM MQ 程式庫。 當您準備用戶端應用程式時，必須執行下列動作:

- 針對您應用程式使用的語言連結，使用適當的用戶端系統程式庫。 例如:

– Linux AIX 對於在 AIX and Linux 上以 C 撰寫的應用程式，請使用程式庫 libmqic 而非 libmqm。
– Windows 在 Windows 系統上，請使用程式庫 mqic.lib 而非 mqm.lib。

- 請使用同等的用戶端系統檔案庫，而不使用 第 782 頁的表 134 和 第 782 頁的表 135 中所顯示的伺服器系統檔案庫。 如果伺服器系統檔案庫未列在這些表格中，請在用戶端系統上使用相同的檔案庫。

表 *134: AIX and Linux* 上的用戶端系統程式庫

IBM MQ 伺服器系統的檔案庫	**在 IBM MQ 用戶端系統上使用的對等程式庫**
libmqmxa	libmqcxa

表 *135: Windows* 系統上的用戶端系統程式庫

IBM MQ 伺服器系統的檔案庫	**在 IBM MQ 用戶端系統上使用的對等程式庫**
mqmxa.lib	mqcxa.lib
mqmtux.lib	mqcxa.lib
mqmenc.lib	mqcxa.lib
mqmcics4.lib	mqccics4.lib

用戶端應用程式使用的使用者 ID

當您在 CICS 下執行 IBM MQ 伺服器應用程式時，它通常會從 CICS 使用者切換至交易的使用者 ID。 不過，當您在 CICS 下執行 IBM MQ MQI client 應用程式時，它會保留 CICS 特許權限。

ALW *CICS* 和 *Tuxedo* 範例程式

在 AIX, Linux, and Windows 系統上使用的 CICS 及 Tuxedo 範例程式。

第 782 頁的表 136 列出提供用於 AIX and Linux 用戶端系統的 CICS 及 Tuxedo 範例程式。 第 782 頁的表 137 列出 Windows 用戶端系統的對等資訊。 這些表格也會列出用於準備及執行程式的檔案。 如需程式範例的說明，請參閱 第 906 頁的『CICS 交易範例』 及 第 945 頁的『在 AIX, Linux, and Windows 上使用 TUXEDO 範例』。

表 *136: AIX and Linux* 用戶端系統的程式範例

說明	**來源**	**執行檔模組**
CICS 程式	amqscic0.ccs	amqscicc
CICS 程式的標頭檔	amqscih0.h	-
Tuxedo 要放置訊息的用戶端程式	amqstxpx.c	-
Tuxedo 用來取得訊息的用戶端程式	amqstxgx.c	-
兩個用戶端程式的 Tuxedo 伺服器程式	amqstxsx.c	-
Tuxedo 程式的 UBBCONFIG 檔案	ubbstxcx.cfg	-
Tuxedo 程式的欄位表格檔案	amqstxvx.flds	-
檢視 Tuxedo 程式的說明檔	amqstxvx.v	-

表 *137: Windows* 用戶端系統的程式範例

說明	**來源**	**執行檔模組**
CICS 交易	amqscic0.ccs	amqscicc
CICS 交易的標頭檔	amqscih0.h	-

表 137: Windows 用戶端系統的程式範例 (繼續)		
說明	**來源**	**執行檔模組**
Tuxedo 要放置訊息的用戶端程式	amqstxpx.c	-
Tuxedo 用來取得訊息的用戶端程式	amqstxgx.c	-
兩個用戶端程式的 Tuxedo 伺服器程式	amqstxsx.c	-
Tuxedo 程式的 UBBCONFIG 檔案	ubbstxcx.cfg	-
Tuxedo 程式的欄位表格檔案	amqstxvx.fld	-
檢視 Tuxedo 程式的說明檔	amqstxvx.v	-
Tuxedo 程式的 make 檔	amqstxmc.mak	-
Tuxedo 程式的 ENVFILE 檔	amqstxen.env	-

ALW 針對 *CICS* 及 *Tuxedo* 應用程式所修改的錯誤訊息 *AMQ5203*

當您執行使用延伸交易式用戶端的 CICS 或 Tuxedo 應用程式時，可能會看到標準診斷訊息。 其中一個已修改，以與延伸交易式用戶端搭配使用

您可能在 IBM MQ 錯誤日誌檔中看到的訊息記載在 診斷訊息: AMQ4000-9999 中。 已修改訊息 AMQ5203 ，以與延伸交易式用戶端搭配使用。 以下是已修改訊息的文字:

AMQ5203: 呼叫 XA 介面時發生錯誤。

說明

The error number is &2 where a value of 1 indicates the supplied flags value of &1 was invalid, 2 indicates that there was an attempt to use threaded and non-threaded libraries in the same process, 3 indicates that there was an error with the supplied queue manager name '&3', 4 indicates that the resource manager ID of &1 was invalid, 5 indicates that an attempt was made to use a second queue manager called '&3' when another queue manager was already connected, 6 indicates that the Transaction Manager has been called when the application isn't connected to a queue manager, 7 indicates that the XA call was made while another call was in progress, 8 indicates that the xa_info string '&4' in the xa_open call contained an invalid parameter value for parameter name '&5', and 9 indicates that the xa_info string '&4' in the xa_open call is missing a required parameter, parameter name '&5'.

使用者回應

請更正錯誤，然後重試作業。

Windows 準備及執行 Microsoft Transaction Server 應用程式

若要準備 MTS 應用程式以作為 IBM MQ MQI client 應用程式執行，請針對您的環境適當遵循下列指示。

如需如何開發存取 IBM MQ 資源之 Microsoft Transaction Server (MTS) 應用程式的一般資訊，請參閱 IBM MQ 說明中心的 MTS 一節。

若要準備 MTS 應用程式以作為 IBM MQ MQI client 應用程式來執行，請針對應用程式的每一個元件執行下列其中一項:

- 如果元件使用 MQI 的 C 語言連結，請遵循 第 853 頁的『在 Windows 中準備 C 程式』 中的指示，但將元件與程式庫 mqicxa.lib 鏈結，而不是 mqic.lib。
- 如果元件使用 IBM MQ C++ 類別，請遵循 第 460 頁的『在 Windows 上建置 C++ 程式』 中的指示，但將元件鏈結至程式庫 imqx23vn.lib ，而不是 imqc23vn.lib。
- 如果元件使用 MQI 的 Visual Basic 語言連結，請遵循 第 856 頁的『在 Windows 中準備 Visual Basic 程式』 中的指示，但當您定義 Visual Basic 專案時，請在 **條件式編譯引數** 欄位中鍵入 `MqType=3` 。

準備及執行 IBM MQ JMS 應用程式

您可以在用戶端模式下執行 IBM MQ JMS 應用程式，並以 WebSphere Application Server 作為交易管理程式。 您可能會看到某些警告訊息。

若要以用戶端模式準備並執行 IBM MQ JMS 應用程式，並以 WebSphere Application Server 作為交易管理程式，請遵循 第 68 頁的『使用 IBM MQ classes for JMS/Jakarta Messaging』中的指示。

當您執行 IBM MQ JMS 用戶端應用程式時，可能會看到下列警告訊息:

MQJE080
授權單元不足-執行 setmqcap

MQJE081
包含授權單元資訊的檔案格式錯誤-請執行 setmqcap

MQJE082
找不到包含授權單元資訊的檔案-請執行 setmqcap

使用者結束程式、API 結束程式及 IBM MQ 可安裝服務

本主題包含使用及開發這些程式之相關資訊的鏈結。

如需如何使用使用者結束程式、API 結束程式及可安裝服務來延伸佇列管理程式機能的簡介，請參閱 延伸佇列管理程式機能。

如需撰寫及編譯結束程式和可安裝服務的相關資訊，請參閱子主題。

相關概念
MQI 通道的通道結束程式
相關參考
API 結束程式參照
可安裝的服務介面參照資訊
IBM i IBM i 上的可安裝服務介面參照資訊

ALW 在 AIX, Linux, and Windows 上撰寫結束程式及可安裝的服務

您可以撰寫及編譯結束程式，而無需鏈結至 AIX, Linux, and Windows 上的任何 IBM MQ 程式庫。

關於這項作業

本主題僅適用於 AIX, Linux, and Windows 系統。 如需為其他平台撰寫結束程式及可安裝服務的詳細資料，請參閱相關平台特定主題。

如果 IBM MQ 安裝在非預設位置，您必須撰寫並編譯結束程式，而不鏈結至任何 IBM MQ 程式庫。

您可以在 AIX, Linux, and Windows 系統上撰寫及編譯結束程式，而不鏈結下列任何 IBM MQ 程式庫:

- mqmzf
- mqm
- mqmvx
- mqmvxd
- mqic
- mqutl

鏈結至這些程式庫的現有結束程式會繼續運作，前提是 AIX and Linux 系統 IBM MQ 已安裝在預設位置。

程序

1. 包括 cmqec.h 標頭檔。
   包括此標頭檔會自動包括 cmqc.h、 cmqxc.h 及 cmqzc.h 標頭檔。

2. 撰寫結束程式，以便透過 MQIEP 結構進行 MQI 和 DCI 呼叫。 如需 MQIEP 結構的相關資訊，請參閱 MQIEP 結構。
   - 可安裝的服務
     - 使用 **Hconfig** 參數來指向 MQZEP 呼叫。
     - 在使用 **Hconfig** 參數之前，您必須檢查 **Hconfig** 的前 4 個位元組是否符合 MQIEP 結構的 **StrucId** 。
     - 如需撰寫可安裝服務元件的相關資訊，請參閱 MQIEP。
   - API 結束程式
     - 使用 **Hconfig** 參數來指向 MQXEP 呼叫。
     - 在使用 **Hconfig** 參數之前，您必須檢查 **Hconfig** 的前 4 個位元組是否符合 MQIEP 結構的 **StrucId** 。
     - 如需撰寫 API 結束程式的相關資訊，請參閱 第 798 頁的『寫入 API 結束程式』。
   - 通道結束程式
     - 使用 MQCXP 結構的 **pEntryPoints** 參數來指向 MQI 和 DCI 呼叫。
     - 在使用 **pEntryPoints** 之前，您必須先檢查 MQCXP 版本號碼是否為第 8 版或更新版本。
     - 如需寫入通道結束程式的相關資訊，請參閱 第 808 頁的『寫入通道結束程式』。
   - 資料轉換結束程式
     - 使用 MQDXP 結構的 **pEntryPoints** 參數來指向 MQI 和 DCI 呼叫。
     - 在使用 **pEntryPoints** 之前，您必須檢查 MQDXP 版本號碼是否為第 2 版或更高版本。
     - 您可以使用 **crtmqcvx** 指令及 amqsvfc0.c 原始檔，來建立使用 **pEntryPoints** 參數的資料轉換碼。 請參閱第 829 頁的『撰寫 IBM MQ for Windows 的資料轉換結束程式』及第 827 頁的『撰寫 IBM MQ for AIX or Linux 系統的資料轉換結束程式』。
     - 如果您具有使用 **crtmqcvx** 指令產生的現有資料轉換結束程式，則必須使用更新的指令重新產生結束程式。
     - 如需寫入資料轉換結束程式的相關資訊，請參閱 第 823 頁的『寫入資料-轉換結束程式』。
   - 預先連接結束程式
     - 使用 MQNXP 結構的 **pEntryPoints** 參數來指向 MQI 和 DCI 呼叫。
     - 在使用 **pEntryPoints** 之前，您必須檢查 MQNXP 版本號碼是否為第 2 版或更高版本。
     - 如需撰寫預先連接結束程式的相關資訊，請參閱 第 831 頁的『使用儲存庫中的預先連接結束程式來參照連線定義』。
   - 發佈結束程式
     - 使用 MQPSXP 結構的 **pEntryPoints** 參數來指向 MQI 和 DCI 呼叫。
     - 在使用 **pEntryPoints** 之前，您必須檢查 MQPSXP 版本號碼是否為第 2 版或更高版本。
     - 如需撰寫發佈結束程式的相關資訊，請參閱 第 833 頁的『撰寫及編譯發佈結束程式』。
   - 叢集工作量結束程式
     - 使用 MQWXP 結構的 **pEntryPoints** 參數來指向 MQXCLWLN 呼叫。
     - 在使用 **pEntryPoints** 之前，您必須先檢查 MQWXP 版本號碼是否為第 4 版或更新版本。
     - 如需寫入叢集工作量結束程式的相關資訊，請參閱 第 834 頁的『撰寫及編譯叢集工作量結束程式』。

   例如，在呼叫 MQPUT 的通道結束程式中:

```
pChannelExitParms -> pEntryPoints -> MQPUT_Call(pChannelExitParms -> Hconn,
                                                Hobj,
                                                &md,
                                                &pmo,
                                                messlen,
                                                buffer,
```

```
                                                &CompCode,
                                                &Reason);
```

如需進一步的範例，請參閱 第 887 頁的『使用 IBM MQ 範例程序化程式』。

3. 編譯結束程式:
   - 請勿鏈結至 IBM MQ 程式庫。
   - 不要在結束程式中包含任何 IBM MQ 程式庫的內嵌 RPath。
   - 如需編譯結束程式的相關資訊，請參閱下列其中一個主題:
     - API 結束程式: 第 800 頁的『編譯 API 結束程式』。
     - 通道結束程式、發佈結束程式、叢集工作量結束程式: 第 822 頁的『在 AIX, Linux, and Windows 系統上編譯通道結束程式』。
     - 資料轉換結束程式: 第 823 頁的『寫入資料-轉換結束程式』。
4. 將結束程式放置在下列其中一個位置:
   - 配置結束程式時您選擇的完全合格路徑
   - 特定安裝目錄中的預設結束程式路徑。 例如，*MQ_DATA_PATH*/exits/installation2。
   - 預設結束程式路徑

     32 位元結束程式的預設結束程式路徑為 *MQ_DATA_PATH*/exits ， 64 位元結束程式的預設結束程式路徑為 *MQ_DATA_PATH*/exits64 。 您可以在 qm.ini 或 mqclient.ini 檔案中變更這些路徑。 如需相關資訊，請參閱 結束路徑。 在 Windows 和 Linux 上，您可以使用 IBM MQ Explorer 來變更路徑:

     a. 用滑鼠右鍵按一下佇列管理程式名稱

     b. 按一下 **內容 ...**

     c. 按一下 **結束程式**

     d. 在結束程式預設路徑欄位中，指定保留結束程式之目錄的路徑名稱。

   如果結束程式同時放置在特定安裝目錄及預設路徑目錄中，則路徑中所指名的 IBM MQ 安裝會使用特定安裝目錄結束程式。 例如，結束程式放置在 /exits/installation2 和 /exits 中，但不在 /exits/installation1 中。 IBM MQ 安裝 installation2 會使用 /exits/installation2 的結束程式。 IBM MQ 安裝 installation1 會使用 /exits 目錄中的結束程式。
5. 必要的話，請配置結束程式:
   - 可安裝的服務: 第 793 頁的『配置服務和元件』。
   - API 結束程式: 第 803 頁的『配置 API 結束程式』。
   - 通道結束程式: 第 823 頁的『配置通道結束程式』。
   - 發佈結束程式: 第 834 頁的『配置發佈結束程式』。
   - 預先連接結束程式: 用戶端配置檔的 PreConnect 段落。

ALW *API 結束程式未與 MQI 程式庫鏈結*

在特定情況下，您應該將無法重新編碼以使用 MQIEP 函數指標的現有 API 結束程式與 IBM MQ API 程式庫相鏈結。

這是必要的，讓系統的執行時期鏈結器可以順利將現有的 API 結束程式載入尚未載入函數指標的程式中。

註: 此資訊僅限於直接發出 MQI 呼叫的現有 API 結束程式。 亦即，那些不使用 MQIEP 的結束程式。 可能的話，您應該計劃重新編寫結束程式的程式碼，以改用 MQIEP 進入點。

從 IBM MQ 8.0 開始， **runmqsc** 是未直接與 MQI 程式庫鏈結的程式範例。

因此，尚未與其必要的 IBM MQ API 程式庫鏈結或重新編碼以使用 MQIEP 的 API 結束程式無法載入至 **runmqsc**。

您會在佇列管理程式錯誤日誌中看到錯誤，例如 AMQ6175: **系統無法動態載入共用程式庫**，以及合格文字，例如 undefined symbol: MQCONN。

及 AMQ7214: **無法載入 API 結束程式 'myexitname' 的模組**。

相關工作

第 784 頁的『在 AIX, Linux, and Windows 上撰寫結束程式及可安裝的服務』
您可以撰寫及編譯結束程式，而無需鏈結至 AIX, Linux, and Windows 上的任何 IBM MQ 程式庫。

ALW AIX, Linux, and Windows 的可安裝服務及元件

本節介紹可安裝的服務及其相關聯的功能和元件。 這些功能的介面已記載，因此您或軟體供應商可以提供元件。

提供 IBM MQ 可安裝服務的主要原因如下:

- 提供您彈性選擇是使用 IBM MQ 產品所提供的元件，還是使用其他產品來取代或擴增它們。
- 若要容許供應商參與，請提供可能使用新技術的元件，而無需對 IBM MQ 產品進行內部變更。
- 允許 IBM MQ 更快且更便宜地利用新技術，並以更低的價格提早提供產品。

可安裝服務 和 服務元件 是 IBM MQ 產品結構的一部分。 此結構的中心是佇列管理程式的一部分，其實作與「訊息佇列介面 (MQI)」相關聯的功能及規則。 此中央組件需要一些服務功能 (稱為 可安裝服務) ，才能執行其工作。 可安裝的服務如下:

- 授權服務
- 名稱服務

每一個可安裝服務都是使用一個以上 服務元件實作的一組相關功能。 使用適當架構且公開可用的介面來呼叫每一個元件。 這可讓獨立軟體供應商及其他協力廠商提供可安裝的元件，以擴增或取代 IBM MQ 產品所提供的元件。 第 787 頁的表 138 彙總可使用的服務和元件。

表 *138:* 可安裝的服務元件摘要

可安裝服務 (installable service)	**提供的元件**	**函數**	**<**
授權服務	Object Authority Manager (OAM)	提供對指令及 MQI 呼叫的授權檢查。 使用者可以撰寫自己的元件來擴增或取代 OAM。 例如，檢查使用者 ID 是否具有開啟佇列的權限。	(假設適當的平台授權機能)
名稱服務	無	提供佇列管理程式的支援，以查閱擁有指定佇列的佇列管理程式名稱。 • 使用者定義	• 協力廠商或使用者撰寫的名稱管理程式

可安裝的服務介面參照資訊中說明可安裝的服務介面。

相關工作

配置可安裝的服務

撰寫服務元件

本節說明服務、元件、進入點及回覆碼之間的關係。

功能和元件

每一個服務都由一組相關功能組成。 例如，名稱服務包含下列項目的函數:

- 查閱佇列名稱並傳回佇列定義所在的佇列管理程式名稱
- 將佇列名稱插入服務的目錄
- 從服務的目錄中刪除佇列名稱

它也包含起始設定和終止功能。

可安裝的服務由一或多個服務元件提供。 每一個元件都可以執行針對該服務定義的部分或所有功能。 例如，在 IBM MQ for AIX 中，所提供的授權服務元件 OAM 會執行所有可用的功能。 如需相關資訊，請參閱第 790 頁的『授權服務介面』。 元件也負責管理實作服務所需的任何基礎資源或軟體 (例如， LDAP 目錄)。 配置檔提供標準方式來載入元件，以及判斷它所提供功能常式的位址。

第 788 頁的圖 99 顯示服務和元件的相關方式:

- 服務是由配置檔中的段落定義給佇列管理程式。
- 佇列管理程式中所提供的程式碼支援每一個服務。 使用者無法變更此程式碼，因此無法建立自己的服務。
- 每一個服務都由一個以上元件實作; 這些元件可以隨產品或使用者撰寫一起提供。 可以呼叫服務的多個元件，每個元件都支援服務內的不同機能。
- 進入點會將服務元件連接至佇列管理程式中的支援程式碼。

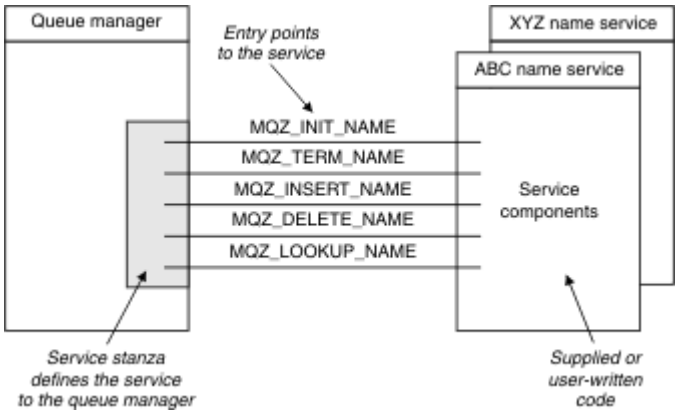


圖 *99:* 瞭解服務、元件及進入點

進入點

每一個服務元件都由支援特定可安裝服務之常式的進入點位址清單代表。 可安裝服務定義每一個常式要執行的功能。

配置服務元件時，服務元件的排序會定義在嘗試滿足服務要求時呼叫進入點的順序。

在提供的標頭檔 `cmqzc.h` 中，提供給每一個服務的進入點都有 `MQZID_` 字首。

如果服務存在，則會以預先定義的順序載入服務。 下列清單顯示服務及其起始設定順序。

1. `NameService`
2. `AuthorizationService`
3. `UserIdentifierService`

`AuthorizationService` 是依預設配置的唯一服務。 如果您想要使用 `NameService` 和 `UserIdentifierService`，請手動配置它們。

服務及服務元件具有一對一或一對多對映。 每一個服務可以定義多個服務元件。 在 AIX and Linux 系統上，`ServiceComponent` 段落的「服務」值必須符合 qm.ini 檔案中「服務」段落的「名稱」值。 在 Windows 上， `ServiceComponent` 的服務登錄機碼值必須符合名稱登錄機碼值，且定義為: `HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere MQ\Installation\`*`MQ_INSTALLATION_NAME`*`\Configuration\QueueManager\`*`qmname`*`\`，其中 *qmname* 是佇列管理程式的名稱。

對於 AIX and Linux 系統，服務元件會以其在 qm.ini 檔案中定義的順序來啟動。 在 Windows 上，因為使用 Windows 登錄，所以 IBM MQ 會發出 **RegEnumKey** 呼叫，以英文字母順序傳回值。 因此，在 Windows 上，會依照登錄中的定義，以英文字母順序來呼叫服務。

`ServiceComponent` 定義的排序非常重要。 此排序指定針對給定服務執行元件的順序。 例如， Windows 上的 `AuthorizationService` 配置了名為 `MQSeries.WindowsNT.auth.service` 的預設 OAM 元件。 可以為此服務定義其他元件，以置換預設 OAM。 除非指定 `MQCACF_SERVICE_COMPONENT`，否則會使用以英文字母順序發現的第一個元件來處理要求，並使用該元件的名稱。

回覆碼

服務元件提供回覆碼給佇列管理程式，以針對各種狀況產生報告。 它們會報告作業成功或失敗，並指出佇列管理程式是否要繼續下一個服務元件。 個別的 接續 參數會包含此指示。

元件資料

單一服務元件可能需要在其各種功能之間共用資料。 可安裝的服務提供要在每次呼叫服務元件時傳遞的選用資料區。 此資料區供服務元件專用。 它由特定函數的所有呼叫共用，即使它們是從不同的位址空間或處理程序產生。 每當呼叫服務元件時，都保證可從服務元件定址。 您必須在 *ServiceComponent* 段落中宣告此區域的大小。

元件的起始設定及終止

使用元件起始設定和終止選項。

當呼叫元件起始設定常式時，它必須針對元件支援的每一個進入點呼叫佇列管理程式 **MQZEP** 函數。 **MQZEP** 定義服務的進入點。 所有未定義的跳出點都假設為空值。

在以任何其他方式呼叫元件之前，一律使用主要起始設定選項來呼叫元件一次。

在特定平台上，可以使用次要起始設定選項來呼叫元件。 例如，可以針對用來存取服務的每一個作業系統處理程序、執行緒或作業呼叫它一次。

如果使用次要起始設定:

- 對於次要起始設定，可以多次呼叫元件。 對於每一個這類呼叫，當不再需要服務時，會發出次要終止的相符呼叫。

  對於命名服務，這是 MQZ_TERM_NAME 呼叫。

  對於授權服務，這是 MQZ_TERM_AUTHORITY 呼叫。
- 每次對主要及次要起始設定呼叫元件時，都必須重新指定進入點 (透過呼叫 MQZEP)。
- 元件只會使用一個元件資料副本; 每一個次要起始設定都沒有不同的副本。
- 在執行次要起始設定之前，不會對服務 (從作業系統處理程序、執行緒或作業，視情況而定) 的任何其他呼叫呼叫呼叫元件。
- 對於主要和次要起始設定，元件必須將 **Version** 參數設為相同的值。

當不再需要元件時，一律會使用主要終止選項來呼叫該元件一次。 不會進一步呼叫此元件。

如果已針對次要起始設定呼叫元件，則會使用次要終止選項來呼叫元件。

Object Authority Manager (OAM)

IBM MQ 產品隨附的授權服務元件稱為「物件權限管理程式 (OAM)」。

依預設， OAM 處於作用中，並使用控制指令 **dspmqaut** (顯示權限)、 **dmpmqaut** (傾出權限) 及 **setmqaut** (設定或重設權限)。

使用控制指令管理 IBM MQ for Multiplatforms 中說明了這些指令的語法以及如何使用它們。

OAM 使用主體或群組的 實體 :

- Linux AIX 在 AIX and Linux 系統上，主體是使用者 ID ，或與代表使用者執行之應用程式相關聯的 ID; 群組是系統定義的主體集合。
- Windows 在 Windows 系統上，主體是 Windows 使用者 ID ，或與代表使用者執行之應用程式相關聯的 ID; 群組是 Windows 群組。

可以在主體或群組層次授與或撤銷授權。

當發出 MQI 要求或指令時， OAM 會檢查與作業相關聯的實體是否有權執行所要求的作業及存取指定的佇列管理程式資源。

授權服務可讓您撰寫您自己的授權服務元件，以擴增或取代提供給佇列管理程式的權限檢查。

名稱服務
名稱服務是可安裝的服務，可支援佇列管理程式查閱擁有指定佇列的佇列管理程式名稱。 無法從名稱服務擷取其他佇列屬性。

名稱服務可讓應用程式開啟輸出的遠端佇列，如同它們是本端佇列一樣。 不會對佇列以外的物件呼叫名稱服務。

註: 遠端佇列必須將其 **Scope** 屬性設為 CELL。

當應用程式開啟佇列時，它會先在佇列管理程式的目錄中尋找佇列名稱。 如果在那裡找不到它，它會查看已配置的名稱服務數目，直到找到可辨識佇列名稱的名稱服務為止。 如果沒有人辨識該名稱，則開啟會失敗。

名稱服務會傳回該佇列的擁有端佇列管理程式。 然後，佇列管理程式會繼續執行 MQOPEN 要求，如同指令已在原始要求中指定佇列及佇列管理程式名稱一樣。

名稱服務介面 (NSI) 是 IBM MQ 架構的一部分。

名稱服務的運作方式

如果佇列定義將 **Scope** 屬性指定為佇列管理程式 (即 MQSC 中的 SCOPE (QMGR)) ，則佇列定義 (以及所有佇列屬性) 只會儲存在佇列管理程式的目錄中。 這無法由可安裝服務取代。

如果佇列定義將 **Scope** 屬性指定為 Cell (即 MQSC 中的 SCOPE (CELL)) ，則佇列定義會再次儲存在佇列管理程式的目錄中，以及所有佇列屬性。 不過，佇列和佇列管理程式名稱也會儲存在名稱服務中。 如果沒有可用來儲存此資訊的服務，則無法定義含有 *Scope* Cell 的佇列。

儲存資訊的目錄可以由服務管理，或者服務可以基於此目的使用基礎服務 (例如 LDAP 目錄)。 在任一情況下，儲存在目錄中的定義必須持續保存，即使在元件及佇列管理程式終止之後，直到明確刪除它們為止。

註:

1. 若要將訊息傳送至命名目錄 Cell 內不同佇列管理程式上遠端主機的本端佇列定義 (CELL 範圍) ，您需要定義通道。
2. 您無法直接從遠端佇列取得訊息，即使它具有 CELL 範圍也一樣。
3. 傳送至 CELL 範圍的佇列時，不需要遠端佇列定義。
4. 命名服務會集中定義目的地佇列，雖然您仍需要目的地佇列管理程式的傳輸佇列及一對通道定義。 此外，本端系統上的傳輸佇列必須與遠端系統上擁有目標佇列 (Cell 範圍) 的佇列管理程式同名。

   例如，如果遠端佇列管理程式具有名稱 QM01，則本端系統上的傳輸佇列也必須具有名稱 QM01。

授權服務介面
授權服務提供佇列管理程式使用的進入點。

進入點如下:

MQZ_AUTHENTICATE_USER
鑑別使用者 ID 和密碼，並且可以設定身分環境定義欄位。

MQZ_CHECK_AUTHORITY
檢查實體是否具有對指定物件執行一個以上作業的權限。

MQZ_CHECK_特許
檢查指定的使用者是否為特許使用者。

MQZ_COPY_ALL_AUTHORITY
將參照物件的所有現行授權複製到另一個物件。

MQZ_DELETE_AUTHORITY
刪除與指定物件相關聯的所有授權。

MQZ_ENUMERATE_AUTHORITY_DATA
擷取符合指定選取準則的所有權限資料。

MQZ_FREE_USER
釋放相關聯的已配置資源。

MQZ_GET_AUTHORITY
取得實體存取指定物件的權限。

MQZ_GET_EXPLICIT_AUTHORITY
取得具名群組必須存取指定物件的權限 (但沒有 **nobody** 群組的其他權限)，或取得具名主體的主要群組必須存取指定物件的權限。

MQZ_INIT_AUTHORITY
起始設定授權服務元件。

MQZ_INQUIRE
查詢授權服務支援的功能。

MQZ_REFRESH_CACHE
重新整理所有授權。

MQZ_SET_AUTHORITY
設定實體對指定物件的權限。

MQZ_TERM_AUTHORITY
終止授權服務元件。

此外，在 IBM MQ for Windows 上，授權服務提供下列進入點供佇列管理程式使用:

- **MQZ_CHECK_AUTHORITY_2**
- **MQZ_GET_AUTHORITY_2**
- **MQZ_GET_EXPLICIT_AUTHORITY_2**
- **MQZ_SET_AUTHORITY_2**

這些進入點支援使用 Windows 安全 ID (NT SID)。

這些名稱在標頭檔 `cmqzc.h` 中定義為 **typedef**，可用來建立元件函數的原型。

起始設定函數 (**MQZ_INIT_AUTHORITY**) 必須是元件的主要進入點。 其他函數會透過起始設定函數已新增至元件進入點向量的進入點位址來呼叫。

名稱服務介面
名稱服務提供佇列管理程式使用的進入點。

提供下列進入點:

MQZ_INIT_NAME
起始設定名稱服務元件。

MQZ_TERM_NAME
終止名稱服務元件。

MQZ_LOOKUP_NAME
查閱指定佇列的佇列管理程式名稱。

MQZ_INSERT_NAME
將包含所指定佇列之擁有佇列管理程式名稱的項目插入服務所使用的目錄中。

MQZ_DELETE_NAME
從服務所使用的目錄中刪除指定佇列的項目。

如果配置了多個名稱服務:

- 為了查閱，會針對清單中的每一個服務呼叫 MQZ_LOOKUP_NAME 函數，直到解析佇列名稱為止 (除非任何元件指出應該停止搜尋)。
- 對於 insert，會針對清單中支援此函數的第一個服務呼叫 MQZ_INSERT_NAME 函數。
- 對於 delete，會針對清單中支援此函數的第一個服務呼叫 MQZ_DELETE_NAME 函數。

不要有多個支援插入和刪除功能的元件。 不過，僅支援查閱的元件是可行的，並且可以使用 (例如，作為清單中的最後一個元件)，將任何其他名稱服務元件不知道的名稱解析為可以定義名稱的佇列管理程式。

在 C 程式設計語言中，使用 typedef 陳述式將名稱定義為函數資料類型。 這些可用來建立服務函數的原型，以確保參數是正確的。

包含可安裝服務的所有特定資料的標頭檔是適用於 C 語言的 cmqzc.h 。

除了起始設定函數 (MQZ_INIT_NAME) (必須是元件的主要進入點) 之外，還會使用 MQZEP 呼叫起始設定函數所新增的進入點位址來呼叫函數。

使用多個服務元件
您可以為一個服務安裝多個元件。 這可讓元件只提供局部服務實作，並依賴其他元件來提供其餘功能。

使用多個元件的範例

假設您建立兩個名稱服務元件，稱為 ABC_name_serv 和 XYZ_name_serv。

ABC_name_serv
此元件支援在服務目錄中插入名稱，或從服務目錄中刪除名稱，但不支援查閱佇列名稱。

XYZ_name_serv
此元件支援查閱佇列名稱，但不支援在服務目錄中插入名稱或刪除名稱。

元件 ABC_name_serv 會保留佇列名稱的資料庫，並使用兩個簡式演算法來插入或刪除服務目錄中的名稱。

元件 XYZ_name_serv 使用簡式演算法，針對用來呼叫它的任何佇列名稱，傳回固定的佇列管理程式名稱。 它不會保留佇列名稱的資料庫，因此不支援插入及刪除功能。

元件安裝在相同的佇列管理程式上。 *ServiceComponent* 段落會依序排列，以便先呼叫元件 ABC_name_serv 。 任何在元件目錄中插入或刪除佇列的呼叫，都由元件 ABC_name_serv ; 它是唯一一個執行這些功能的。 不過，元件 ABC_name_serv 無法解析的查閱呼叫會傳遞至僅限查閱元件 XYZ_name_serv。 此元件從其簡式演算法提供佇列管理程式名稱。

使用多個元件時省略進入點

如果您決定使用多個元件來提供服務，則可以設計未實作特定功能的服務元件。 可安裝的服務架構沒有您可以省略的限制。 不過，對於特定的可安裝服務，省略一或多個功能在邏輯上可能與服務的目的不一致。

與多個元件搭配使用的進入點範例

第 792 頁的表 139 顯示已安裝兩個元件的可安裝名稱服務範例。 每一個都支援與此特定可安裝服務相關聯的不同功能集。 對於 insert 函數，會先呼叫 ABC 元件進入點。 尚未定義給服務的進入點 (使用 **MQZEP**) 假設為 NULL。 表格中提供了起始設定的進入點，但這並非必要，因為起始設定是由元件的主要進入點來執行。

當佇列管理程式必須使用可安裝的服務時，它會使用定義給該服務的進入點 (第 792 頁的表 139 中的直欄)。 依序取得每一個元件，佇列管理程式會決定實作所需函數之常式的位址。 然後，它會呼叫常式 (如果存在的話)。 如果作業成功，佇列管理程式會使用任何結果及狀態資訊。

表 *139:* 可安裝服務的進入點範例

函數號碼	**ABC 名稱服務元件**	**XYZ 名稱服務元件**
MQZID_INIT_NAME (起始設定)	ABC_initialize ()	XYZ_initialize ()
MQZID_TERM_NAME (終止)	ABC_terminate ()	XYZ_terminate ()
MQZID_INSERT_NAME (插入)	ABC_Insert ()	NULL
MQZID_DELETE_NAME (刪除)	ABC_Delete ()	NULL
MQZID_LOOKUP_NAME (查閱)	NULL	XYZ_Lookup ()

如果常式不存在，則佇列管理程式會針對清單中的下一個元件重複此處理程序。 此外，如果常式確實存在，但傳回代碼指出它無法執行作業，則會繼續嘗試下一個可用的元件。 服務元件中的常式可能會傳回程式碼，指出不應進一步嘗試執行作業。

配置服務和元件

您可以使用佇列管理程式配置檔來配置服務元件，但 Windows 系統上除外，其中每一個佇列管理程式在「登錄」中都有自己的段落。

程序

1. 將段落新增至佇列管理程式配置檔 `qm.ini`，以定義佇列管理程式的服務，並指定模組的位置:
   - 所使用的每一個服務都必須有一個 `Service` 段落，用來定義佇列管理程式的服務。 如需相關資訊，請參閱 qm.ini 檔案的服務段落。
   - 對於服務內的每一個元件，必須有一個 `ServiceComponent` 段落。 此段落識別包含該元件之程式碼的模組名稱及路徑。 如需相關資訊，請參閱 qm.ini 檔案的 ServiceComponent 段落。

   授權服務元件 (稱為「物件權限管理程式 (OAM)」) 隨產品一起提供。 當您建立佇列管理程式時，會自動更新佇列管理程式配置檔 (或 Windows 系統上的「登錄」)，以包括授權服務及預設元件 (OAM) 的適當段落。 對於其他元件，您必須手動配置佇列管理程式配置檔。

   當使用動態連結來啟動佇列管理程式時，會將每一個服務元件的程式碼載入至佇列管理程式，其中在平台上受支援。
2. 停止並重新啟動佇列管理程式，以啟動元件。

相關參考

qm.ini 檔案的服務段落

qm.ini 檔案的 ServiceComponent 段落

在變更使用者授權之後重新整理 *OAM*

在 IBM MQ 中，您可以在變更使用者的授權群組成員資格之後立即重新整理 OAM 的授權群組資訊，以反映在作業系統層次所做的變更，而不需要停止並重新啟動佇列管理程式。 若要這樣做，請發出 **REFRESH SECURITY** 指令。

註: 當您使用 `setmqaut` 指令變更授權時， OAM 會立即實作這類變更。

佇列管理程式會將授權資料儲存在稱為 SYSTEM.AUTH.DATA.QUEUE。 此資料由 **amqzfuma.exe** 管理。

相關參考

REFRESH SECURITY

IBM i IBM i 上的可安裝服務及元件

使用此資訊來瞭解可安裝的服務，以及與其相關聯的功能和元件。 這些功能的介面已記載，因此您或軟體供應商可以提供元件。

提供 IBM MQ 可安裝服務的主要原因如下:

- 為您提供彈性，可選擇使用 IBM MQ for IBM i 所提供的元件，還是使用其他元件來取代或擴增它們。
- 若要容許供應商參與，請提供可能使用新技術的元件，而無需對 IBM MQ for IBM i 進行內部變更。
- 允許 IBM MQ 更快且更便宜地利用新技術，並以更低的價格提早提供產品。

可安裝服務 和 服務元件 是 IBM MQ 產品結構的一部分。 此結構的中心是佇列管理程式的一部分，其實作與「訊息佇列介面 (MQI)」相關聯的功能及規則。 此中央組件需要一些服務功能 (稱為 可安裝服務)，才能執行其工作。 IBM MQ for IBM i 中可用的可安裝服務是授權服務。

每一個可安裝服務都是使用一個以上 服務元件實作的一組相關功能。 使用適當架構且公開可用的介面來呼叫每一個元件。 這可讓獨立軟體供應商及其他協力廠商提供可安裝的元件，以擴增或取代 IBM MQ for IBM i 所提供的元件。 第 794 頁的表 140 彙總對授權服務的支援。

表 *140:* 授權服務元件摘要		
提供的元件	**函數**	**<**
Object Authority Manager (OAM)	提供對指令及 MQI 呼叫的授權檢查。 使用者可以撰寫自己的元件來擴增或取代 OAM。	(假設適當的平台授權機能)
DCE 名稱服務元件 **註:** 只有 V6.0 之前的 IBM MQ 版本才支援 DCE。	• 容許佇列管理程式共用佇列，或 • 使用者定義 **註:** 共用佇列必須將其 **Scope** 屬性設為 CELL。	• 所提供的元件需要 DCE ，或 • 協力廠商或使用者撰寫的名稱管理程式

IBM i *IBM i* 上的功能及元件

使用此資訊來瞭解您可以在 IBM MQ for IBM i 中使用的函數及元件、進入點、回覆碼及元件資料。

每一個服務都由一組相關功能組成。 例如，名稱服務包含下列項目的函數:

- 查閱佇列名稱並傳回佇列定義所在的佇列管理程式名稱
- 將佇列名稱插入服務的目錄
- 從服務的目錄中刪除佇列名稱

它也包含起始設定和終止功能。

可安裝的服務由一或多個服務元件提供。 每一個元件都可以執行針對該服務定義的部分或所有功能。 元件也負責管理實作服務所需的任何基礎資源或軟體。 配置檔提供標準方式來載入元件，以及判斷它所提供功能常式的位址。

服務及元件相關如下:

- 服務是由配置檔中的段落定義給佇列管理程式。
- 佇列管理程式中所提供的程式碼支援每一個服務。 使用者無法變更此程式碼，因此無法建立自己的服務。
- 每一個服務都由一個以上元件實作; 這些元件可以隨產品或使用者撰寫一起提供。 可以呼叫服務的多個元件，每個元件都支援服務內的不同機能。
- 進入點會將服務元件連接至佇列管理程式中的支援程式碼。

進入點

每一個服務元件都由支援特定可安裝服務之常式的進入點位址清單代表。 可安裝服務定義每一個常式要執行的功能。 配置服務元件時，服務元件的排序會定義在嘗試滿足服務要求時呼叫進入點的順序。 在提供的標頭檔 `cmqzc.h` 中，提供給每一個服務的進入點都有 `MQZID_` 字首。

回覆碼

服務元件提供回覆碼給佇列管理程式，以報告各種狀況。 它們會報告作業成功或失敗，並指出佇列管理程式是否要繼續下一個服務元件。 個別的 接續 參數會包含此指示。

元件資料

單一服務元件可能需要在其各種功能之間共用資料。 可安裝的服務提供在每次呼叫特定服務元件時傳遞的選用資料區。 此資料區供服務元件專用。 它由給定函數的所有呼叫共用，即使它們是從不同的位址空間或處理程序產生。 每當呼叫服務元件時，都保證可從服務元件定址。 您必須在 *ServiceComponent* 段落中宣告此區域的大小。

IBM i 在 *IBM i* 上起始設定

當呼叫元件起始設定常式時，它必須針對元件支援的每一個進入點呼叫佇列管理程式 `MQZEP` 函數。 `MQZEP` 定義服務的進入點。 所有未定義的跳出點都假設為空值。

主要起始設定
在以任何其他方式呼叫元件之前，一律會使用此選項來呼叫元件一次。

次要起始設定
在特定平台上，可以使用此選項來呼叫元件。 例如，可以針對用來存取服務的每一個作業系統處理程序、執行緒或作業呼叫它一次。

如果使用次要起始設定:

- 對於次要起始設定，可以多次呼叫元件。 對於每一個這類呼叫，當不再需要服務時，會發出次要終止的相符呼叫。

  對於授權服務，這是 MQZ_TERM_AUTHORITY 呼叫。
- 每次對主要及次要起始設定呼叫元件時，都必須重新指定進入點 (透過呼叫 MQZEP)。
- 元件只會使用一個元件資料副本; 每一個次要起始設定都沒有不同的副本。
- 在執行次要起始設定之前，不會對服務 (從作業系統處理程序、執行緒或作業，視情況而定) 的任何其他呼叫呼叫呼叫元件。
- 對於主要和次要起始設定，元件必須將 **Version** 參數設為相同的值。

主要終止
當不再需要元件時，一律使用此選項來啟動該元件一次。 不會進一步呼叫此元件。

次要終止
如果已針對次要起始設定啟動元件，則會使用此選項來啟動元件。

IBM i 在 *IBM i* 上配置服務及元件

您可以使用佇列管理程式配置檔來配置服務元件。

程序

1. 將段落新增至佇列管理程式配置檔 `qm.ini`，以定義佇列管理程式的服務，並指定模組的位置:
   - 所使用的每一個服務都必須有一個 `Service` 段落，用來定義佇列管理程式的服務。 如需相關資訊，請參閱 qm.ini 檔案的服務段落。
   - 對於服務內的每一個元件，必須有一個 `ServiceComponent` 段落。 此段落識別包含該元件之程式碼的模組名稱及路徑。 如需相關資訊，請參閱 qm.ini 檔案的 ServiceComponent 段落。

   授權服務元件 (稱為「物件權限管理程式 (OAM)」) 隨產品一起提供。 當您建立佇列管理程式時，會自動更新佇列管理程式配置檔，以包括授權服務及預設元件 (OAM) 的適當段落。 對於其他元件，您必須手動配置佇列管理程式配置檔。

   當使用動態連結來啟動佇列管理程式時，會將每一個服務元件的程式碼載入至佇列管理程式，其中在平台上受支援。
2.

IBM i 在 *IBM i* 上建立您自己的服務元件

使用此資訊來瞭解如何在 IBM MQ for IBM i 上建立服務元件。

如果要建立您自己的服務元件，請執行下列動作:

- 請確定標頭檔 `cmqzc.h` 包含在您的程式中。
- 透過編譯程式並將其與共用程式庫 `libmqm*` 及 `libmqmzf*`鏈結，來建立共用程式庫。

  註: 因為代理程式可以在執行緒化環境中執行，所以您必須建置 OAM 以在執行緒化環境中執行。 這包括使用 `libmqm` 和 `libmqmzf` 的執行緒版本。
- 將段落新增至佇列管理程式配置檔，以定義佇列管理程式的服務，並指定模組的位置。
- 停止並重新啟動佇列管理程式，以啟動元件。

IBM i *IBM i* 上的授權服務

授權服務是可安裝的服務，可讓佇列管理程式呼叫授權機能，例如，檢查使用者 ID 是否有權開啟佇列。

此服務是 IBM MQ 安全啟用介面 (SEI) 的元件，它是 IBM MQ 架構的一部分。 討論的主題如下:

- 第 796 頁的『Object Authority Manager (OAM)』
- 第 796 頁的『向作業系統定義服務』
- 第 796 頁的『配置授權服務段落』
- 第 797 頁的『IBM i 上的授權服務介面』

Object Authority Manager (OAM)

IBM MQ 產品隨附的授權服務元件稱為物件權限管理程式 (OAM)。 依預設， OAM 處於作用中，並使用下列控制指令:

- **WRKMQMAUT** 使用權限
- **WRKMQMAUTD** 使用權限資料
- **DSPMQMAUT** 顯示物件權限
- **GRTMQMAUT** 授與物件權限
- **RVKMQMAUT** 取消物件權限
- **RFRMQMAUT** 重新整理安全性

CL 指令說明中說明這些指令的語法及使用方式。 OAM 與主體或群組的 實體 一起運作。

當發出 MQI 要求或發出指令時， OAM 會檢查與作業相關聯之實體的授權，以查看它是否可以執行下列動作:

- 執行所要求的作業。
- 存取指定的佇列管理程式資源。

授權服務可讓您撰寫您自己的授權服務元件，以擴增或取代提供給佇列管理程式的權限檢查。

向作業系統定義服務

佇列管理程式配置檔 `qm.ini` 中的授權服務段落定義佇列管理程式的授權服務。 如需段落類型的相關資訊，請參閱 第 795 頁的『在 IBM i 上配置服務及元件』 。

配置授權服務段落

在 IBM MQ for IBM i 上:

主體
是 IBM i 系統使用者設定檔。

群組
是 IBM i 系統群組設定檔。

只能在群組層次授與或撤銷授權。 授與或撤銷使用者權限的要求會更新該使用者的主要群組。

每一個佇列管理程式都有自己的佇列管理程式配置檔。 例如，佇列管理程式 QMNAME 的佇列管理程式配置檔預設路徑及檔名為 `/QIBM/UserData/mqm/qmgrs/QMNAME/qm.ini`。

預設授權元件的 *Service* 段落及 *ServiceComponent* 段落會自動新增至 `qm.ini` ，但可以由 `WRKENVVAR` 置換。 必須手動新增任何其他 *ServiceComponent* 段落。

例如，佇列管理程式配置檔中的下列段落定義兩個授權服務元件:

```
Service:
    Name=AuthorizationService
    EntryPoints=7

ServiceComponent:
    Service=AuthorizationService
    Name=MQ.UNIX.authorization.service
    Module=QMQM/AMQZFU
    ComponentDataSize=0

ServiceComponent:
    Service=AuthorizationService
    Name=user.defined.authorization.service
    Module=LIBRARY/SERVICE PROGRAM NAME
    ComponentDataSize=96
```

圖 *100: IBM i* 上 *qm.ini* 中的授權服務段落

第一個服務元件段落 `MQ.UNIX.authorization.service` 定義預設授權服務元件 OAM。 如果您移除此段落並重新啟動佇列管理程式，則會停用 OAM，且不會進行授權檢查。

IBM i *IBM i* 上的授權服務介面

授權服務介面提供數個進入點供佇列管理程式使用。

MQZ_AUTHENTICATE_USER
鑑別使用者 ID 和密碼，並且可以設定身分環境定義欄位。

MQZ_CHECK_AUTHORITY
檢查實體是否具有對指定物件執行一個以上作業的權限。

MQZ_COPY_ALL_AUTHORITY
將參照物件的所有現行授權複製到另一個物件。

MQZ_DELETE_AUTHORITY
刪除與指定物件相關聯的所有授權。

MQZ_ENUMERATE_AUTHORITY_DATA
擷取符合指定選取準則的所有權限資料。

MQZ_FREE_USER
釋放相關聯的已配置資源。

MQZ_GET_AUTHORITY
取得實體存取指定物件的權限。

MQZ_GET_EXPLICIT_AUTHORITY
取得具名群組必須存取指定物件的權限 (但沒有 **nobody** 群組的其他權限)，或取得具名主體的主要群組必須存取指定物件的權限。

MQZ_INIT_AUTHORITY
起始設定授權服務元件。

MQZ_INQUIRE
查詢授權服務支援的功能。

MQZ_REFRESH_CACHE
重新整理所有授權。

MQZ_SET_AUTHORITY
設定實體對指定物件的權限。

MQZ_TERM_AUTHORITY
終止授權服務元件。

這些進入點支援使用 Windows 安全 ID (NT SID)。

這些名稱在標頭檔 `cmqzc.h` 中定義為 **typedef**，可用來建立元件函數的原型。

起始設定函數 (**MQZ_INIT_AUTHORITY**) 必須是元件的主要進入點。 其他函數會透過起始設定函數已新增至元件進入點向量的進入點位址來呼叫。

如需相關資訊，請參閱第 795 頁的『在 IBM i 上建立您自己的服務元件』。

Multi 在 Multiplatforms 上撰寫及編譯 API 結束程式

API 結束程式可讓您撰寫程式碼來變更 IBM MQ API 呼叫 (例如 MQPUT 和 MQGET) 的行為，然後在那些呼叫之前或之後立即插入該程式碼。

註: z/OS 在 IBM MQ for z/OS 上不受支援。

為何使用 API 結束程式?

每一個應用程式都有特定的工作要執行，且其程式碼應該盡可能有效率地執行該作業。 在更高層次上，您可能想要將標準或商業程序套用至使用該佇列管理程式之所有應用程式的特定佇列管理程式。 在個別應用程式層次以上執行此動作會更有效率，因此不必變更每一個受影響應用程式的程式碼。

以下是 API 結束程式可能有用的一些區域建議:

安全
基於安全考量，您可以提供鑑別，檢查應用程式是否已獲授權存取佇列或佇列管理程式。 您也可以對應用程式使用 API、鑑別個別 API 呼叫，甚至他們使用的參數進行鑑別。

靈活彈性
為了靈活彈性，您可以回應商業環境中的快速變更，而無需變更依賴該環境中資料的應用程式。 例如，您可以具有 API 結束程式，以回應製造環境中的利率、貨幣匯率或元件價格的變更。

監視佇列或佇列管理程式的使用
若要監視佇列或佇列管理程式的使用，您可以追蹤應用程式及訊息的流程、在 API 呼叫中記載錯誤、設定審核追蹤以進行結算，或收集使用情形統計資料以進行規劃。

執行 API 結束程式時會發生什麼情況?

在您寫入結束程式並將它識別到 IBM MQ 之後，佇列管理程式會在登錄點自動呼叫您的結束碼。

要執行的 API 結束常式在 Multiplatforms 上的段落中識別。 本主題涵蓋配置檔 `mqs.ini` 和 `qm.ini` 中的段落。

常式的定義可以在三個位置進行:

1. ApiExit 一般，在 mqs.ini 檔案中，識別整個 IBM MQ 的常式，在佇列管理程式啟動時套用。 這些可以由針對個別佇列管理程式定義的常式置換 (請參閱此清單中的項目 第 798 頁的『3』)。
2. ApiExit 範本，在 mqs.ini 檔案中，識別整個 IBM MQ 的常式，當建立新的佇列管理程式時，會複製到 ApiExit 本端集 (請參閱此清單中的項目 第 798 頁的『3』)。
3. ApiExit 本端，在 qm.ini 檔中，識別適用於特定佇列管理程式的常式。

建立新的佇列管理程式時， mqs.ini 中的 ApiExit 範本定義會複製到新佇列管理程式的 qm.ini 中 ApiExit 本端定義。 啟動佇列管理程式時，會同時使用 ApiExitCommon 及 ApiExit 本端定義。 如果「 ApiExit 本端定義」和「 ApiExit 共用定義」都識別相同名稱的常式，則這兩個定義會取代「ApiExit 共用定義」。 Sequence 屬性 (如 第 803 頁的『配置 API 結束程式』 中所述) 決定段落中所定義常式的執行順序。

跨多個 IBM MQ 安裝使用 API 結束程式

請確定針對舊版 IBM MQ 所撰寫的 API 結束程式是用來處理所有版本，因為在 IBM WebSphere MQ 7.1 中對結束程式所做的變更可能無法與舊版一起運作。 如需對結束程式所做變更的相關資訊，請參閱 第 784 頁的『在 AIX, Linux, and Windows 上撰寫結束程式及可安裝的服務』。

針對 API 結束程式 amqsem 及 amqsaxe 提供的範例會反映在撰寫結束程式時所需的變更。 用戶端應用程式必須確保在啟動應用程式之前，與應用程式相關聯的佇列管理程式安裝所對應的正確 IBM MQ 程式庫已鏈結至該應用程式。

Multi 寫入 *API* 結束程式

您可以使用 C 程式設計語言來撰寫每個 API 呼叫的結束程式。

可用的結束程式

每個 API 呼叫都可以使用結束程式，如下所示:

- MQCB，用來重新登錄指定物件控點的回呼，並控制回呼的啟動和變更
- MQCTL，對針對連線開啟的物件控點執行控制動作
- MQCONN/MQCONNX，提供用於後續 API 呼叫的佇列管理程式連線控點
- MQDISC，切斷佇列管理程式的連線
- MQBEGIN，開始廣域工作單元 (UOW)
- MQBACK，取消 UOW
- MQCMIT，確定 UOW
- MQOPEN，開啟 IBM MQ 資源以進行後續存取
- MQCLOSE，關閉先前已開啟供存取的 IBM MQ 資源
- MQGET，從先前開啟以供存取的佇列中擷取訊息
- MQPUT1，將訊息放置在佇列上
- MQPUT，將訊息放置在先前開啟以進行存取的佇列上
- MQINQ，查詢先前開啟以進行存取之 IBM MQ 資源的屬性
- MQSET，設定先前已開啟以進行存取之佇列的屬性
- MQSTAT，擷取狀態資訊
- MQSUB，將應用程式訂閱登錄至特定主題
- MQSUBRQ，用於對訂閱提出要求

MQ_CALLBACK_EXIT 提供要在回呼處理之前及之後執行的結束函數。 如需相關資訊，請參閱 回呼-MQ_CALLBACK_EXIT。

寫入 API 結束程式

在 API 結束程式內，呼叫採用一般格式:

```
MQ_call_EXIT (parameters, context, ApiCallParameters)
```

其中 *call* 是不含 MQ 字首的 MQI 呼叫名稱; 例如 PUT、GET。 *parameters* 控制結束程式的功能，主要提供結束程式與外部控制區塊之間的通訊 MQAXP (API 結束程式參數結構) 及 MQAXC (API 結束程式環境定義結構)。 *context* 說明在其中呼叫 API 結束程式的環境定義，且 *ApiCallParameters* 代表 MQI 呼叫的參數。

為了協助您撰寫 API 結束程式，會提供範例結束程式 amqsaxe0.c; 這個結束程式會對您指定的檔案產生追蹤項目。 您可以使用此範例作為撰寫結束程式時的起點。 如需使用範例結束程式的相關資訊，請參閱 第 902 頁的『API 結束程式範例程式』。

如需 API 結束程式呼叫、外部控制區塊及相關聯主題的相關資訊，請參閱 API 結束程式參照。

如需如何撰寫、編譯及配置結束程式的一般資訊，請參閱 第 784 頁的『在 AIX, Linux, and Windows 上撰寫結束程式及可安裝的服務』。

在 API 結束程式中使用訊息控點

您可以控制 API 結束程式有權存取哪些訊息內容。 內容與 ExitMsg 控點相關聯。 放置結束程式中設定的內容會設定在要放置的訊息上，但在取得結束程式中擷取的內容不會傳回給應用程式。

當您在 **Function** 設為 MQXF_INIT 且 **ExitReason** 設為 MQXR_CONNECTION 時使用 MQXEP MQI 呼叫來登錄 MQ_INIT_EXIT 結束程式函數時，您可以傳入 MQXEPO 結構作為 **ExitOpts** 參數。 MQXEPO 結構包含 ExitProperties 欄位，此欄位指定可供結束程式使用的內容集。 它指定為代表內容字首的字串，對應於 MQRFH2 資料夾名稱。

每一個 API 結束程式都會接收 MQAXP 結構，其中包含 ExitMsgHandle 欄位。 此欄位設為 IBM MQ 所產生的值，且特定於連線。 因此，在相同連線上相同或不同類型的 API 結束程式之間，控點不會變更。

在 MQ_PUT_EXIT 或 MQ_PUT1_EXIT 中， **ExitReason** 為 MQXR_BEFORE (亦即，在放置訊息之前執行的 API 結束程式)，會在放置訊息時設定與 ExitMsg 控點相關聯的任何內容 (訊息描述子內容除外)。 若要防止發生此情況，請將 ExitMsg 控點設為 MQHM_NONE。 您也可以提供不同的訊息控點。

在 MQ_GET_EXIT 及 MQ_CALLBACK_EXIT 中， ExitMsgHandle 會清除內容，並在登錄 MQ_INIT_EXIT 時移入 ExitProperties 欄位中指定的內容，而非訊息描述子內容。 取得應用程式無法使用這些內容。 如果取得應用程式在 MQGMO (取得訊息選項) 欄位中指定訊息控點，則與該控點相關聯的任何內容 (包括訊息描述子內容) 都可供 API 結束程式使用。 若要防止將內容移入 ExitMsg 控點，請將其設為 MQHM_NONE。

註: 對於要在中處理的結束訊息內容:

- 在 MQ_GET_EXIT 函數之後，您必須為結束程式定義一個之前的 MQ_GET_EXIT 函數。
- 在 MQ_CALLBACK_EXIT 函數之前，您必須為結束程式定義一個之前的 MQ_CB_EXIT 函數。

V 9.3.2 從 IBM MQ 9.3.2 開始，先前關於 MQ-GET-EXIT 和 MQ_CALLBACK_EXIT 的陳述式不再適用。

提供範例程式 amqsaem0.c，以說明在 API 結束程式中使用訊息控點的情況。

相關參考
使用者結束程式、API 結束程式及可安裝的服務參照

Multi 編譯 *API* 結束程式

撰寫結束程式之後，您可以編譯並鏈結它，如下所示。

下列範例顯示用於 第 902 頁的『API 結束程式範例程式』中所說明的範例程式的指令。 若為 Windows 系統以外的平台，您可以在 *MQ_INSTALLATION_PATH*/samp 中找到範例 API 結束碼，並在 *MQ_INSTALLATION_PATH*/samp/bin 中找到已編譯及鏈結的共用程式庫。

Windows 若為 Windows 系統，您可以在 *MQ_INSTALLATION_PATH* \Tools\c\Samples 中找到範例 API 結束碼。 *MQ_INSTALLATION_PATH* 代表 IBM MQ 的安裝目錄。

註: 64 位元平台上的編碼標準中列出程式設計 64 位元應用程式的指引。

對於「多重播送」用戶端， API 結束程式及資料轉換結束程式必須能夠在用戶端上執行，因為部分訊息可能不會通過佇列管理程式。 下列程式庫是用戶端套件以及伺服器套件的一部分:

表 *141:* 用戶端和伺服器套件中的程式庫

作業系統	圖書館業
AIX AIX	32 bit & 64 bit: libmqm.a & libmqm_r.a
IBM i IBM i	LIBMQM 與 LIBMQM_R
Linux Linux	32 bit & 64 bit: libmqm.so & libmqm_r.so
Windows Windows	32 bit & 64 bit: mqm.dll & mqm.pdb

Linux AIX 在 *AIX and Linux* 系統上編譯 *API* 結束程式
如何在 AIX and Linux 系統上編譯 API 結束程式的範例。

在所有平台上，模組的進入點是 MQStart。

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

在 AIX 上:

AIX

發出下列其中一個指令來編譯 API 結束程式原始碼:

32 位元應用程式

無執行緒

```
cc -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

執行緒作業

```
xlc_r -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe_r \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

64 位元應用程式

無執行緒

```
cc -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

執行緒作業

```
xlc_r -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe_r \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

在 Linux 上:

Linux

發出下列其中一個指令來編譯 API 結束程式原始碼:

31 位元應用程式

無執行緒

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

執行緒作業

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

32 位元應用程式

無執行緒

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

執行緒作業

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

64 位元應用程式

無執行緒

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

執行緒作業

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Windows 在 *Windows* 系統上編譯 *API* 結束程式

在 Windows 上編譯並鏈結範例 API 結束程式 amqsaxe0.c

資訊清單檔是選用的 XML 文件，包含可內嵌在已編譯應用程式或 DLL 中的版本或任何其他資訊。

如果您沒有這類文件，請在 **mt** 指令中省略 -manifest *manifest.file* 參數。

調整 第 802 頁的圖 101 或 第 802 頁的圖 102 中範例的指令，以編譯並鏈結 Windows 上的 amqsaxe0.c。 這些指令與 Microsoft Visual Studio 2008、2010 或 2012 搭配使用。 這些範例假設 C:\Program Files\IBM\MQ\tools\c\samples 目錄是現行目錄。

32 位元

```
cl /c /nologo /MD /Foamqsaxe0.obj amqsaxe0.c

link /nologo /dll /def:amqsaxe.def



amqsaxe0.obj \
   /manifest /out:amqsaxe.dll

mt -nologo -manifest amqsaxe.dll.manifest \
   -outputresource:amqsaxe.dll;2
```

圖 *101:* 編譯並鏈結 *amqsaxe0.c 32* 位元 *Windows*

64 位元

```
cl /c /nologo /MD /Foamqsaxe0.obj amqsaxe0.c

link /nologo /dll /def:amqsaxe.def \
     /libpath:..\..\lib64 \



amqsaxe0.obj /manifest /out:amqsaxe.dll

mt -nologo -manifest amqsaxe.dll.manifest \
   -outputresource:amqsaxe.dll;2
```

圖 *102:* 在 *64* 位元 *Windows* 上編譯並鏈結 *amqsaxe0.c*

相關概念

第 902 頁的『API 結束程式範例程式』

範例 API 結束程式會對使用者指定的檔案產生 MQI 追蹤，並在 **MQAPI_TRACE_LOGFILE** 環境變數中定義字首。

IBM i *IBM i* 上的複雜 *API* 結束程式

正在 IBM i 上編譯 API 結束程式。

結束程式建立如下 (C 語言範例):

1. 使用 CRTCMOD 建立模組。 請併入 TERASPACE(*YES *TSIFC) 參數來編譯它以使用兆空間。

2. 使用 CRTSRVPGM 從模組建立服務程式。 您必須將它連結至服務程式 QMQM/LIBMQMZF_R ， 以進行多執行緒 API 結束程式。

配置 *API* 結束程式

您可以透過變更配置資訊來配置 IBM MQ 以啟用 API 結束程式。

若要變更配置資訊，您必須變更定義結束常式及其執行順序的段落。 您可以使用下列方式來變更此資訊:

- Windows Linux 在 Windows 和 Linux (x86 和 x86-64 平台) 上使用 IBM MQ Explorer 。
- Windows 在 Windows 上使用 **amqmdain** 指令。
- Multi 直接在 Multiplatforms 上使用 `mqs.ini` 及 `qm.ini` 檔案。

  `mqs.ini` 檔案包含與特定節點上所有佇列管理程式相關的資訊。 您可以在下列位置找到它:

  – Linux AIX 在 AIX and Linux 上的 `/var/mqm` 目錄中。

  – Windows 在 Windows 系統上 `HKLM\SOFTWARE\IBM\WebSphere MQ` 金鑰中指定的 WorkPath 中。

  – IBM i 在 IBM i 上的 `/QIBM/UserData/mqm` 目錄中。

  `qm.ini` 檔案包含與特定佇列管理程式相關的資訊。 每一個佇列管理程式都有一個佇列管理程式配置檔，保留在佇列管理程式所佔用的目錄樹狀結構根目錄中。 例如，佇列管理程式 QMNAME 的配置檔路徑及名稱為:

  IBM i 在 IBM i 系統上:

  ```
  /QIBM/UserData/mqm/qmgrs/QMNAME/qm.ini
  ```

  Linux AIX 在 AIX and Linux 系統上:

  ```
  /var/mqm/qmgrs/QMNAME/qm.ini
  ```

  Windows 在 Windows 系統上:

  ```
  C:\ProgramData\IBM\MQ\qmgrs\QMNAME\qm.ini
  ```

  在編輯配置檔之前，請先備份它，以便在有需要時可以回復成您的副本。

  您可以使用下列一種方式來編輯配置檔:

  – 自動使用變更節點上佇列管理程式配置的指令。

  – 手動，使用標準文字編輯器。

  如果您在配置檔屬性上設定不正確的值，則會忽略該值，並發出操作員訊息以指出問題。 效果與完全遺漏屬性相同。

要配置的段落

必須變更的段落如下:

ApiExitCommon

在 `mqs.ini` 及 IBM MQ 內容頁上的 IBM MQ Explorer 中，於「結束程式」下定義。

當任何佇列管理程式啟動時，會讀取此段落中的屬性，然後由 `qm.ini` 中定義的 API 結束程式置換。

ApiExitTemplate

在 `mqs.ini` 及 IBM MQ 內容頁上的 IBM MQ Explorer 中，於「結束程式」下定義。

當建立任何佇列管理程式時，此段落中的屬性會複製到新建立的 qm.ini 檔案中 ApiExitLocal 段落下。

ApiExitLocal

在 `qm.ini` 及佇列管理程式內容頁面上的 IBM MQ Explorer 中，於「結束程式」下定義。

當佇列管理程式啟動時，這裡定義的 API 結束程式會置換 `mqs.ini` 中定義的預設值。

段落的屬性

- 使用下列屬性來命名 API 結束程式:

  Name=ApiExit_name
  在 MQAXP 結構的 ExitInfo 名稱欄位中傳遞給它的 API 結束程式敘述性名稱。

  此名稱必須是唯一的，長度不得超過 48 個字元，且只包含 IBM MQ 物件名稱的有效字元 (例如，佇列名稱)。

- 使用下列屬性，識別要執行之 API 結束碼的模組及進入點:

  Function=function_name
  包含 API 結束碼之模組的函數進入點名稱。 此進入點是 MQ_INIT_EXIT 函數。

  這個欄位的長度限制為 MQ_EXIT_NAME_LENGTH。

  模組 = 模組名稱
  包含 API 結束碼的模組。

  如果這個欄位含有模組的完整路徑名稱，則會依其現狀使用。

  如果此欄位只包含模組名稱，則會使用 qm.ini 中 `ExitPath` 內的 `ExitsDefaultPath` 屬性來尋找模組。

  在支援個別執行緒程式庫的平台上，您必須提供 API 結束程式模組的非執行緒及執行緒版本。 執行緒版本必須具有 `_r` 字尾。 在載入之前， IBM MQ 應用程式 Stub 的執行緒版本會隱含地將 `_r` 附加至給定的模組名稱。

  此欄位的長度限制為平台支援的路徑長度上限。

- 選擇性地使用下列屬性在結束程式中傳遞資料:

  Data=data_name
  要在 MQAXP 結構的 ExitData 欄位中傳遞至 API 結束程式的資料。

  如果您包括此屬性，則會移除前導及尾端空白，將剩餘字串截斷為 32 個字元，並將結果傳遞至結束程式。 如果您省略此屬性，則會將預設值 32 個空白傳遞給結束程式。

  此欄位的長度上限為 32 個字元。

- 使用下列屬性來識別此結束程式相對於其他結束程式的順序:

  Sequence=sequence_number
  相對於其他 API 結束程式，呼叫此 API 結束程式的順序。 在具有較高序號的結束程式之前，會先呼叫具有較低序號的結束程式。 結束程式的序號不需要連續。 序列 1、2、3 與序列 7、42、1096 具有相同的結果。 如果兩個結束程式具有相同的序號，則佇列管理程式會決定要先呼叫哪個結束程式。 您可以在 MQAXP 中的 ExitChainAreaPtr 所指出的 ExitChain 區域中放置時間或標記，或寫入您自己的日誌檔，以識別事件之後呼叫的項目。

  此屬性是不帶正負號的數值。

範例段落

範例 mqs.ini 檔案包含下列段落:

ApiExitTemplate
此段落定義具有敘述性名稱 `OurPayrollQueueAuditor`、模組名稱 `auditor` 及序號 2 的結束程式。 資料值 123 會傳遞至結束程式。

ApiExitCommon
此段落定義具有敘述性名稱 MQPoliceman、模組名稱 tmqp 及序號 1 的結束程式。 所傳遞的資料是指示 (CheckEverything)。

```
mqs.ini

   ApiExitTemplate:
      Name=OurPayrollQueueAuditor
      Sequence=2
      Function=EntryPoint
      Module=/usr/ABC/auditor
      Data=123
   ApiExitCommon:
      Name=MQPoliceman
      Sequence=1
      Function=EntryPoint
      Module=/usr/MQPolice/tmqp
      Data=CheckEverything
```

下列範例 qm.ini 檔案包含敘述性名稱 ClientApplicationAPIchecker、模組名稱 ClientAppChecker 及序號 3 之結束程式的 ApiExit 本端定義。

```
qm.ini

   ApiExitLocal:
      Name=ClientApplicationAPIchecker
      Sequence=3
      Function=EntryPoint
      Module=/usr/Dev/ClientAppChecker
      Data=9.20.176.20
```

傳訊通道的通道結束程式

這個主題集合包含傳訊通道的 IBM MQ 通道結束程式相關資訊。

訊息通道代理程式 (MCA) 也可以呼叫資料轉換結束程式。 如需寫入資料轉換結束程式的相關資訊，請參閱第 823 頁的『寫入資料-轉換結束程式』。

其中部分資訊也適用於 MQI 通道 (將 IBM MQ MQI clients 連接至佇列管理程式) 上的結束程式。 如需相關資訊，請參閱 MQI 通道的通道結束程式。

在 MCA 程式執行的處理程序中，會在定義的位置呼叫通道跳出程式。

其中部分使用者結束程式以互補配對方式運作。 例如，如果傳送端 MCA 呼叫使用者結束程式來加密要傳輸的訊息，則互補處理程序必須在接收端發揮作用，以反轉處理程序。

第 805 頁的表 142 顯示每一種通道類型可用的通道結束程式類型。

表 *142:* 每一種通道類型可用的通道結束程式

通道類型	訊息結束	訊息重試結束程式	接收結束	安全結束程式	傳送結束程式	自動定義結束程式
傳送端通道	是		是	是	是	
伺服器通道	是		是	是	是	
叢集傳送端通道	是		是	是	是	是
接收端通道	是	是	是	是	是	是
要求端通道	是	是	是	是	是	
叢集接收端通道	是	是	是	是	是	是

表 142: 每一種通道類型可用的通道結束程式 (繼續)						
通道類型	**訊息結束**	**訊息重試結束程式**	**接收結束**	**安全結束程式**	**傳送結束程式**	**自動定義結束程式**
用戶端連線通道			是	是	是	
伺服器連線通道			是	是	是	是

附註: z/OS

1. 在 z/OS 上，自動定義結束程式僅適用於叢集傳送端及叢集接收端通道。

如果您要在用戶端上執行通道結束程式，則無法使用 MQSERVER 環境變數。 相反地，請依照 用戶端通道定義表中的說明來建立及參照用戶端通道定義表 (CCDT)。

處理概觀

MCA 如何使用通道結束程式的概觀。

啟動時， MCA 會交換啟動對話框以同步化處理程序。 然後，它們會切換至包含安全結束程式的資料交換。 這些結束程式必須順利結束，啟動階段才能完成並容許傳送訊息。

安全檢查階段是迴圈，如 第 806 頁的圖 103 所示。

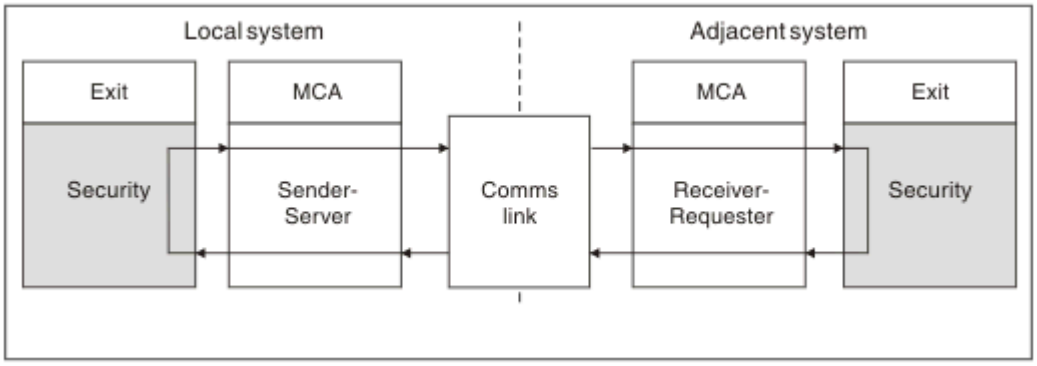


圖 103: 安全結束程式迴圈

在訊息傳送階段期間，傳送端 MCA 會從傳輸佇列取得訊息，呼叫訊息結束程式，呼叫傳送結束程式，然後將訊息傳送至接收端 MCA ，如 第 807 頁的圖 104 所示。

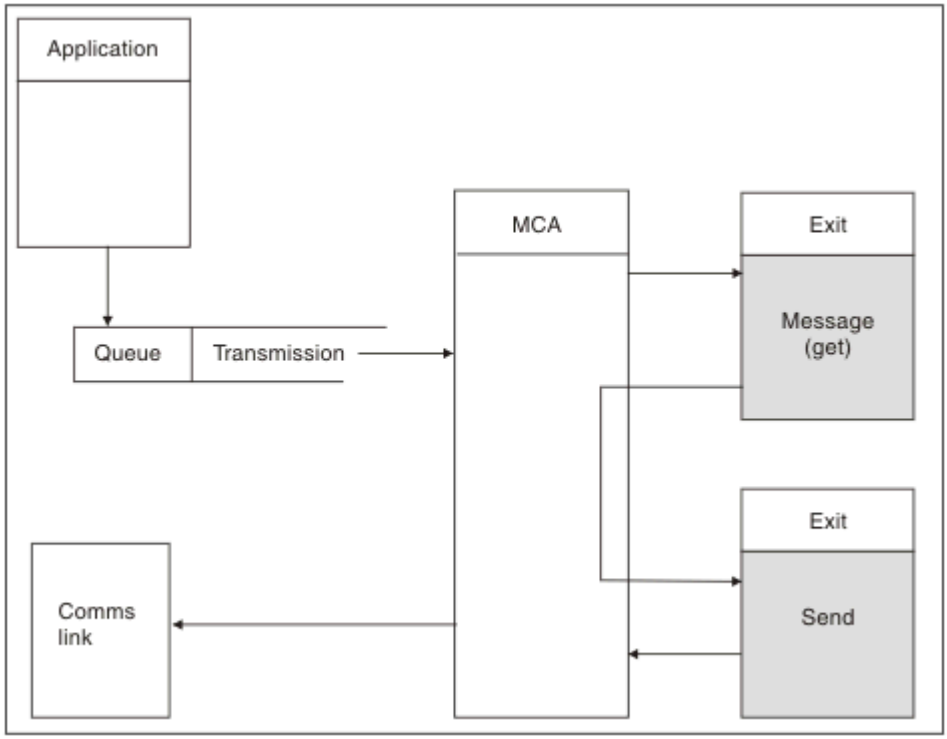


圖 *104:* 訊息通道傳送端的傳送結束程式範例

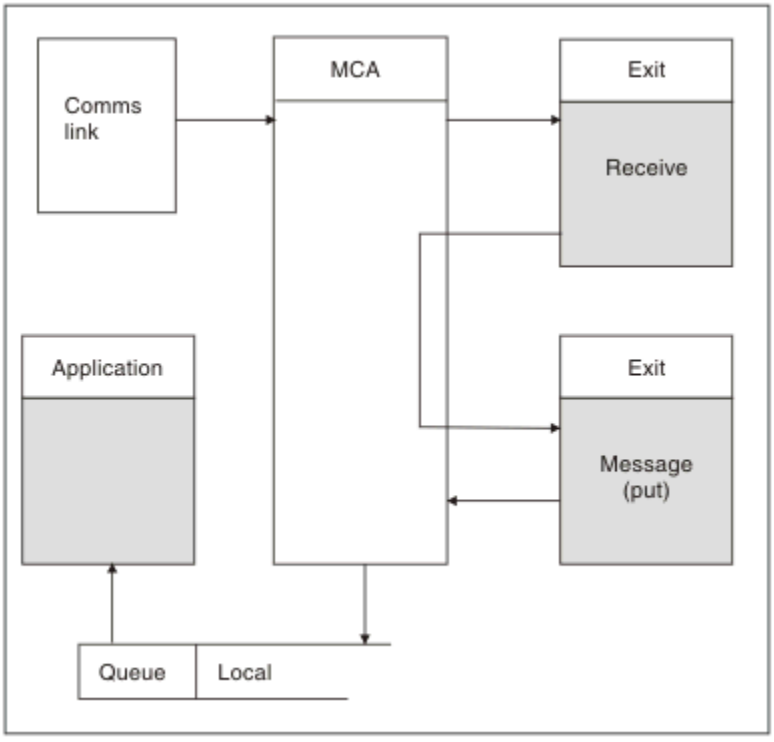


圖 *105:* 訊息通道接收端的接收結束程式範例

接收端 MCA 會從通訊鏈結接收訊息，呼叫接收結束程式，呼叫訊息結束程式，然後將訊息放置在本端佇列上，如 第 807 頁的圖 105 所示。 (在呼叫訊息結束程式之前，可以多次呼叫接收結束程式。)

寫入通道結束程式

您可以使用下列資訊來協助您撰寫通道結束程式。

使用者結束程式及通道結束程式可以使用所有 MQI 呼叫，但下列各節另有說明。 若為 MQ V7 以及更新版本， MQCXP 結構第 7 版及更新版本包含連線控點 hConn，可用來取代發出 MQCONN。 對於舊版，若要取得連線控點，必須發出 MQCONN ，即使因為通道本身連接至佇列管理程式而傳回 MQRC_ALREADY_CONNECTED 警告也一樣。

請注意，通道結束程式必須是安全執行緒。

對於用戶端連線通道上的結束程式，結束程式嘗試連接的佇列管理程式取決於結束程式的鏈結方式。 如果結束程式已與 MQM.LIB (或 IBM i 上的 QMQM/LIBMQM) 且您未在 MQCONN 呼叫中指定佇列管理程式名稱，結束程式會嘗試連接至系統上的預設佇列管理程式。 如果結束程式已與 MQM.LIB (或 IBM i 上的 QMQM/LIBMQM) ，且您指定透過 MQCD 的 QMgrName 欄位傳遞至結束程式的佇列管理程式名稱，結束程式會嘗試連接至該佇列管理程式。 如果結束程式已與 MQIC.LIB 或任何其他檔案庫，不論您是否指定佇列管理程式名稱， MQCONN 呼叫都會失敗。

您應該避免在通道結束程式中變更與所傳遞 hConn 相關聯的交易狀態; 您不得搭配使用 MQCMIT、MQBACK 或 MQDISC 動詞與通道 hConn，且不能使用 MQBEGIN 動詞指定通道 hConn。

如果使用 MQCONNX 指定 MQCNO_HANDLE_SHARE_BLOCK 或 MQCNO_HANDLE_SHARE_NO_BLOCK 來建立新的 IBM MQ 連線，則您必須負責確保連線正確受管理並與佇列管理程式中斷連線。 例如，在每次呼叫時建立與佇列管理程式的新連線而不中斷連線的通道結束程式，會導致建立連線控點，並增加代理程式執行緒數目。

結束程式會在與 MCA 本身相同的執行緒中執行，並使用相同的連線控點。 因此，它在與 MCA 相同的 UOW 內執行，且在同步點下進行的任何呼叫都會在批次結束時由通道確定或取消。

因此，當確定包含原始訊息的批次時，通道訊息結束程式可以傳送僅確定至該佇列的通知訊息。 因此，可以從通道訊息結束程式發出同步點 MQI 呼叫。

通道結束程式可以變更 MQCD 中的欄位。 不過，除非在列出的情況下，否則不會處理這些變更。 如果通道結束程式變更 MQCD 資料結構中的欄位， IBM MQ 通道處理程序會忽略新值。 不過，新值會保留在 MQCD 中，並傳遞至結束鏈中的任何剩餘結束程式，以及傳遞至共用通道實例的任何交談。 如需相關資訊，請參閱 變更通道結束程式中的 MQCD 欄位

此外，對於以 C 撰寫的程式，不可在通道跳出程式中使用非重新進入 C 程式庫函數。

Linux AIX 如果您同時使用多個通道結束程式庫，則當兩個不同結束程式的程式碼包含名稱相同的函數時，部分 UNIX and Linux 平台可能會發生問題。 當載入通道結束程式時，動態載入器會將結束程式庫中的函數名稱解析成載入程式庫的位址。 如果兩個結束程式庫定義的個別函數碰巧具有相同的名稱，則此解析處理程序可能會不正確地解析一個程式庫的函數名稱，以使用另一個程式庫的函數。 如果發生此問題，請對鏈結器指定它只必須匯出必要的 exit 及 MQStart 函數，因為這些函數不受影響。 必須為其他函數提供本端可見性，以便它們不會由其專屬 Exit 程式庫之外的函數使用。 如需相關資訊，請參閱鏈結器的文件。

所有結束程式都會以通道結束程式參數結構 (MQCXP)、通道定義結構 (MQCD)、備妥資料緩衝區、資料長度參數及緩衝區長度參數來呼叫。 不得超出緩衝區長度:

- 對於訊息結束程式，您必須容許透過通道傳送所需的最大訊息，加上 MQXQH 結構的長度。
- 對於傳送及接收結束程式，您必須容許的最大緩衝區如下:

LU 6.2
32 KB

TCP:

IBM i IBM i 16 KB

IBM i 其他 32 KB

註: 最大可用長度可能比此長度小 2 個位元組。 如需詳細資料，請檢查 MaxSegment 長度中傳回的值。 如需 MaxSegment 長度的相關資訊，請參閱 MaxSegment 長度。

NetBIOS:
64KB

SPX:
64KB

註: 傳送端通道上的接收結束程式和接收端通道上的傳送端結束程式使用 2 KB 的 TCP 緩衝區。

- 對於安全結束程式，分散式佇列機能會配置 4000 個位元組的緩衝區。

允許結束程式傳回替代緩衝區以及相關參數。 如需通話詳細資料，請參閱 第 805 頁的『傳訊通道的通道結束程式』 。

z/OS 在 *z/OS* 上撰寫通道結束程式

您可以使用下列資訊來協助您撰寫及編譯 z/OS 的通道結束程式。

結束程式是由下列位置中的 z/OS LINK 所啟動:

- 未獲授權的問題程式狀態
- 主要位址空間控制模式
- 非跨記憶體模式
- 非存取登錄模式
- 31 位元定址模式

鏈結編輯的模組必須放在通道起始程式位址空間程序的 CSQXLIB DD 陳述式所指定的資料集中; 載入模組的名稱會指定為通道定義中的結束程式名稱。

撰寫 z/OS 的通道結束程式時，適用下列規則:

- 結束程式必須以組譯器或 C 撰寫; 如果使用 C ，則必須符合系統結束程式的 C 系統程式設計環境，如 z/OS C/C++ Programming Guide 中所述。
- 從 CSQXLIB DD 陳述式所定義的未獲授權檔案庫載入結束程式。 如果 CSQXLIB 具有 DISP=SHR ，則可以在通道起始程式執行時更新結束程式。 當通道重新啟動時，會使用新版本。
- 結束程式必須重新進入，並且能夠在虛擬儲存體中的任何位置執行。
- 結束程式必須在返回時將環境重設為進入時的環境。
- 結束程式必須釋放任何取得的儲存體，或確定後續的結束程式呼叫已釋放儲存體。

  對於要在呼叫之間持續保存的儲存體，請使用 z/OS STORAGE 服務，或 4kmalc 程式庫函數 for System Programming C。

  如需此功能的相關資訊，請參閱 4kmalc() -- 配置頁面對齊的儲存體。
- 可以使用除 MQCMIT 或 CSQBCMT 及 MQBACK 或 CSQBBAK 以外的所有 IBM MQ MQI 呼叫。 它們必須包含在 MQCONN (具有空白佇列管理程式名稱) 之後。 如果使用這些呼叫，則必須使用 Stub CSQXSTUB 來鏈結編輯結束程式。

  此規則的例外狀況是安全通道結束程式可以發出確定及取消 MQI 呼叫。 如果要發出這類呼叫，請撰寫動詞 CSQXCMT 和 CSQXBAK 來取代 MQCMIT 或 CSQBCMT 和 MQBACK 或 CSQBBAK。
- 從 IBM WebSphere MQ 7.0 或更新版本使用 Stub CSQXSTUB 的所有結束程式都必須在格式為 PDS-E 的 CSQXLIB 載入程式庫中進行鏈結編輯。
- 結束程式不得使用任何導致等待的系統服務，因為使用系統服務會嚴重影響部分或所有其他通道的處理。 許多通道通常在單一 TCB 下執行。 如果您在結束程式中執行導致等待的動作，但未使用 MQXWAIT ，則會導致所有這些通道等待。 造成通道等待不會產生任何功能問題，但可能會對效能產生負面影響。 大部分 SVC 都涉及等待，因此您必須避免它們，但下列 SVC 除外:

  – GETMAIN/FREEMAIN/STORAGE

  – LOAD/DELETE

  因此，一般而言，請避免 SVC、PC 及 I/O。 請改用 MQXWAIT 呼叫。
- 除了附加的任何子作業之外，結束程式不會發出 ESTAE 或 SPIE ，因為其錯誤處理可能會干擾 IBM MQ 所執行的錯誤處理。 這表示 IBM MQ 可能無法從錯誤回復，或您的結束程式可能未收到所有錯誤資訊。
- MQXWAIT 呼叫 (請參閱 MQXWAIT) 提供等待 I/O 及其他事件的等待服務; 如果使用此服務，則結束程式不得使用鏈結堆疊。

對於不提供非封鎖機能或 ECB 等待的 I/O 及其他機能，個別子作業必須是 ATTACHed，且 MQXWAIT 會等待其完成; 由於此技術所產生的處理程序，此機能只能由安全結束程式使用。

- MQDISC MQI 呼叫不會導致隱含確定發生在結束程式內。 只有在通道通訊協定指定時，才會執行通道處理程序的確定。

IBM MQ for z/OS 隨附下列結束程式範例:

CSQ4BAX0
此範例以組譯器撰寫，並說明 MQXWAIT 的用法。

CSQ4BCX1 及 CSQ4BCX2
這些範例以 C 撰寫，並說明如何存取參數。

CSQ4BCX3 和 CSQ4BAX3
這些範例分別以 C 和組譯器撰寫。

CSQ4BCX3 範例 (已前置編譯至 SCSQAUTH LOADLIB 中) 應該會在結束程式本身不需要任何變更的情況下運作。 您可以建立 LOADLIB (例如，稱為 MY.TEST.LOADLIB) 並將 SCSQAUTH (CSQ4BCX3) 成員複製到其中。

若要在用戶端連線上設定安全結束程式，請執行下列程序:

1. 為通道起始程式使用的使用者 ID 建立有效的 OMVS 區段。

   這可讓 IBM MQ for z/OS 通道起始程式搭配使用 TCP/IP 與 z/OS UNIX System Services (z/OS UNIX) Socket 介面，以協助結束程式處理。 請注意，不需要為任何連接用戶端的使用者 ID 定義 OMVS 區段。
2. 請確定結束碼本身只在程式控制的環境中執行。

   這表示載入 CHINIT 位址空間的所有項目都必須從程式控制的程式庫載入 (表示 STEPLIB 中的所有程式庫)，以及 CSQXLIB 和

   ```
   ++hlq++.SCSQANLx
   ++hlq++.SCSQMVR1
   ++hlq++.SCSQAUTH
   ```

   若要將載入程式庫設為受程式控制，請使用與下列範例類似的指令:

   ```
   RALTER PROGRAM * ADDMEM('MY.TEST.LOADLIB'//NOPADCHK)
   ```

   然後，您可以發出下列指令來啟動或重新整理受程式控制的環境:

   ```
   SETROPTS WHEN(PROGRAM) REFRESH
   ```
3. 發出下列指令，將結束程式 LOADLIB 新增至 CSQXLIB DD (在 CHINIT 啟動程序中):

   ```
   ALTER CHANNEL(xxxx) CHLTYPE(SVRCONN)SCYEXIT(CSQ4BCX3)
   ```

   這會啟動具名通道的結束程式。
4. 您的外部安全管理程式 (ESM) 會列出任何要由程式控制的其他程式庫，但請注意， ESM 或 C 程式庫都不需要受到程式控制。

如需使用範例 CSQ4BCX3 設定安全結束程式的進一步資訊，請參閱 IBM MQ for z/OS 伺服器連線通道 。

CSQ4BCX4
此範例以 C 撰寫，並示範使用 MQCXP 中的 **RemoteProduct** 和 **RemoteVersion** 欄位。

相關概念
第 811 頁的『在 IBM i 上撰寫通道結束程式』
您可以使用下列資訊來協助您撰寫及編譯 IBM i 的通道結束程式。

第 811 頁的『在 AIX, Linux, and Windows 上寫入通道結束程式』
您可以使用下列資訊來協助您撰寫 AIX, Linux, and Windows 系統的通道結束程式。

相關參考

IBM MQ for z/OS 伺服器連線通道

IBM i 在 *IBM i* 上撰寫通道結束程式

您可以使用下列資訊來協助您撰寫及編譯 IBM i 的通道結束程式。

跳出程式是以 ILE C、ILE RPG 或 ILE COBOL 語言撰寫的程式物件。 結束程式名稱及其檔案庫在通道定義中命名。

建立及編譯結束程式時，請注意下列條件:

- 程式必須成為安全執行緒，並使用 ILE C、ILE RPG 或 ILE COBOL 編譯器來建立。 若為 ILE RPG ，您必須指定 THREAD (*SERIALIZE) 控制規格，若為 ILE COBOL ，您必須對 PROCESS 陳述式的 THREAD 選項指定 SERIALIZE。 程式也必須連結至含執行緒作業的 IBM MQ 程式庫: 如果是 ILE C 和 ILE RPG ，則為 QMQM/LIBMQM_R; 如果是 ILE COBOL ，則為 AMQ0STUB_R 。 如需使 RPG 或 COBOL 應用程式執行緒安全的相關資訊，請參閱語言的適當 Programmer 's Guide。
- IBM MQ for IBM i 需要啟用跳出程式的兆空間支援。 (Teraspace 是在 OS/400 V4R4 中引進的一種共用記憶體形式。) 對於 ILE RPG 和 COBOL 編譯器，會啟用在 OS/400 V4R4 或更新版本上編譯的任何程式。 對於 C ，必須使用 CRTCMOD 或 CRTBNDC 指令上指定的 TERASPACE (*YES *TSIFC) 選項來編譯程式。
- 將指標傳回其自己的緩衝空間的結束程式必須確定所指向的物件已超出通道結束程式的時間範圍。 指標不能是程式堆疊上變數的位址，也不能是程式資料堆中變數的位址。 相反地，必須從系統取得指標。 例如，在使用者結束程式中建立的使用者空間。 當程式結束時，為了確保通道結束程式所配置的任何資料區仍可供 MCA 使用，通道結束程式必須在呼叫程式或指名啟動群組的啟動群組中執行。 作法是將 CRTPGM 上的 ACTGRP 參數設為使用者定義值或 *CALLER。 如果以這種方式建立程式，通道結束程式可以配置動態記憶體，並將此記憶體的指標傳回 MCA。

相關概念

第 811 頁的『在 AIX, Linux, and Windows 上寫入通道結束程式』
您可以使用下列資訊來協助您撰寫 AIX, Linux, and Windows 系統的通道結束程式。

第 809 頁的『在 z/OS 上撰寫通道結束程式』
您可以使用下列資訊來協助您撰寫及編譯 z/OS 的通道結束程式。

ALW 在 *AIX, Linux, and Windows* 上寫入通道結束程式

您可以使用下列資訊來協助您撰寫 AIX, Linux, and Windows 系統的通道結束程式。

遵循 第 784 頁的『在 AIX, Linux, and Windows 上撰寫結束程式及可安裝的服務』中概述的指示。 適當的話，請使用下列通道結束程式特定資訊:

結束程式必須以 C 撰寫，而且是 Windows 上的 DLL。

在結束程式中定義虛擬 MQStart () 常式，並指定 `MQStart` 作為程式庫中的進入點。 第 811 頁的圖 106 顯示如何設定程式的項目:

```
#include <cmqec.h>

void MQStart() {;} /* dummy entry point - for consistency only */
void MQENTRY ChannelExit ( PMQCXP  pChannelExitParms,
                           PMQCD   pChannelDefinition,
                           PMQLONG pDataLength,
                           PMQLONG pAgentBufferLength,
                           PMQVOID pAgentBuffer,
                           PMQLONG pExitBufferLength,
                           PMQPTR  pExitBufferAddr)
{
... Insert code here
}
```

圖 *106:* 通道結束程式的範例原始碼

使用 Visual C++ 撰寫 Windows 的通道結束程式時，您必須撰寫自己的 DEF 檔案。 如何在 第 812 頁的圖 107 中顯示的範例。 如需寫入通道結束程式的進一步資訊，請參閱 第 808 頁的『寫入通道結束程式』。

```
EXPORTS
ChannelExit
```

圖 *107: Windows* 的範例 *DEF* 檔案

相關概念

第 811 頁的『在 IBM i 上撰寫通道結束程式』
您可以使用下列資訊來協助您撰寫及編譯 IBM i 的通道結束程式。

第 809 頁的『在 z/OS 上撰寫通道結束程式』
您可以使用下列資訊來協助您撰寫及編譯 z/OS 的通道結束程式。

通道安全結束程式
您可以使用安全結束程式來驗證通道另一端的夥伴是否真實。 這稱為鑑別。

若要指定通道必須使用安全結束程式，請在通道定義的 **SCYEXIT** 欄位中指定結束程式名稱。

註: 也可以使用通道鑑別記錄來進行鑑別。 通道鑑別記錄 在防止從特定使用者和通道存取佇列管理程式，以及將遠端使用者對映至 IBM MQ 使用者 ID 時，提供很大的彈性。 IBM MQ 也提供 TLS 支援來鑑別您的使用者，以及為您的資料提供加密和資料完整性檢查。 如需 TLS 的相關資訊，請參閱 IBM MQ 中的 TLS 安全通訊協定。 不過，如果您仍然需要更準確 (或不同) 的安全處理形式，以及其他類型的檢查和安全環境定義建立，請考量撰寫安全結束程式。

「主旨」及「發證者 DN」屬性會出現在下列通道狀態屬性中:

- SSLPEER (PCF 選取元 MQCACH_SSL_SHORT_PEER_NAME)
- SSLCERTI (PCF 選取元 MQCACH_SSL_CERT_ISSUER_NAME)

通道狀態指令會傳回這些值，以及傳遞至所列出通道安全結束程式的資料，如下所示:

- MQCD SSLPeerNamePtr
- MQCXP SSLRemCertIssNamePtr

安全結束程式可以用 C 或 Java 撰寫。

在 MCA 處理週期的下列位置呼叫通道安全結束程式:

- 在 MCA 起始及終止時。
- 在通道啟動時完成起始資料協議之後立即。 通道的接收端或伺服器端可以透過提供要遞送至遠端的安全結束程式的訊息，來起始與遠端的安全訊息交換。 它也可能拒絕這樣做。 重新啟動跳出程式，以處理從遠端系統收到的任何安全訊息。
- 在通道啟動時完成起始資料協議之後立即。 通道的傳送端或要求端會處理從遠端接收的安全訊息，或在遠端無法時起始安全交換。 重新啟動跳出程式，以處理可能收到的所有後續安全訊息。

永遠不會使用 MQXR_INIT_SEC 來呼叫要求端通道。 通道會通知伺服器它有安全結束程式，然後伺服器有機會起始安全結束程式。 如果沒有，則會通知要求端，並將零長度流程傳回給跳出程式。

註: 避免傳送長度為零的安全訊息。

圖 第 813 頁的圖 108 透過 第 815 頁的圖 111 說明安全結束程式交換的資料範例。 這些範例顯示發生的事件順序，包括接收端的安全結束程式，以及傳送端的安全結束程式。 圖中的連續列代表時間的流逝。 在某些情況下，接收端和傳送端的事件不會產生關聯，因此可以同時或在不同時間發生。 在其他情況下，一個結束程式的事件會導致另一個結束程式稍後發生補充事件。 例如，在 第 813 頁的圖 108 中:

1. 接收端和傳送端各以 MQXR_INIT 來呼叫，但這些呼叫並不相關，因此可以同時或不同時間進行。
2. 接下來會使用 MQXR_INIT_SEC 來呼叫接收端，但會傳回 MQXCC_OK (在傳送端結束程式中不需要補充事件)。
3. 接下來會使用 MQXR_INIT_SEC 來呼叫傳送端。 這與 MQXR_INIT_SEC 的接收端呼叫無關。 傳送端會傳回 MQXCC_SEND_SEC_MSG ，這會在接收端結束程式產生補充事件。
4. 然後以 MQXR_SEC_MSG 呼叫接收端，並傳回 MQXCC_SEND_SEC_MSG ，這會在傳送端結束程式產生補充事件。
5. 然後會以 MQXR_SEC_MSG 呼叫傳送端，並傳回 MQXCC_OK ，其在接收端結束程式中不需要補充事件。

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_OK	
	Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_OK
Message transfer begins	

圖 *108:* 傳送端起始與合約的交換

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_OK	
	Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_OK	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_SUPPRESS_FUNCTION
Channel closes	
Invoked with MQXR_TERM Responds with MQXCC_OK	Invoked with MQXR_TERM Responds with MQXCC_OK

圖 *109:* 寄件者起始的交換 *(*無合約*)*

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_OK	
Message transfer begins	
Invoked with MQXR_TERM Responds with MQXCC_OK	Invoked with MQXR_TERM Responds with MQXCC_OK

圖 *110:* 接收端起始的交換 *(*含合約*)*

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_OK
Invoked with MQXR_SEC_MSG Responds with MQXCC_SUPPRESS_FUNCTION	
Channel closes	

圖 *111:* 接收端起始且無合約的交換

通道安全結束程式會傳遞包含安全結束程式所產生之安全資料 (不含任何傳輸標頭) 的代理程式緩衝區。 此資料可以是任何適當的資料，因此通道任一端都可以執行安全驗證。

訊息通道傳送端和接收端的安全結束程式都可以對任何呼叫傳回兩個回應碼之一:

- 安全交換已結束，無錯誤
- 暫停通道並關閉

註:

1. 通道安全結束程式通常成對運作。 當您定義適當的通道時，請確定已為通道兩端命名相容的結束程式。
2. IBM i 在 IBM i 中，已使用 `Use adopted authority` (USEADPAUT = *YES) 編譯的安全跳出程式可以採用 QMQM 或 QMQMADM 權限。 請注意，結束程式不會使用此特性來對系統造成安全風險。
3. 在通道另一端提供憑證的 TLS 通道上，安全結束程式會在 SSLPeerNamePtr 所存取的 MQCD 欄位中接收此憑證主旨的「識別名稱」，並在 SSLRemCertIssNamePtr 所存取的 MQCXP 欄位中接收發證者的「識別名稱」。 可放置此名稱的用途如下:
   - 限制透過 TLS 通道的存取權。
   - 變更 MQCD.MCAUserIdentifier 。

相關概念

傳輸層安全 (TLS) 概念

相關參考

通道鑑別記錄

寫入安全結束程式

您可以使用安全結束程式 Skeleton 程式碼來撰寫安全結束程式。

第 816 頁的圖 112 說明如何撰寫安全結束程式。

```
void MQENTRY MQStart() {;}
void MQENTRY EntryPoint (PMQVOID pChannelExitParms,
                         PMQVOID pChannelDefinition,
                         PMQLONG pDataLength,
                         PMQLONG pAgentBufferLength,
                         PMQVOID pAgentBuffer,
                         PMQLONG pExitBufferLength,
                         PMQPTR  pExitBufferAddr)
{
  PMQCXP pParms = (PMQCXP)pChannelExitParms;
  PMQCD  pChDef = (PMQCD)pChannelDefinition;
  /* TODO: Add Security Exit Code Here */

}
```

圖 *112:* 安全結束程式 *Skeleton* 程式碼

標準 IBM MQ 進入點 MQStart 必須存在，但不需要執行任何功能。 可以變更函數 (在此範例中為 EntryPoint) 的名稱，但在編譯及鏈結程式庫時必須匯出函數。 如前一個範例一樣，指標 pChannelExitParms 必須強制轉型為 PMQCXP ，而 pChannel 定義必須強制轉型為 PMQCD。 如需呼叫通道結束程式及使用參數的一般資訊，請參閱 MQ_CHANNEL_EXIT。 這些參數在安全結束程式中使用，如下所示:

PMQVOID pChannelExitParms

輸入/輸出

MQCXP 結構的指標-強制轉型至 PMQCXP 以存取欄位。 此結構是用來在「結束」與 MCA 之間進行通訊。 MQCXP 中的下列欄位對「安全結束程式」特別感興趣:

ExitReason

告訴「安全結束程式」安全交換中的現行狀態，並在決定要採取的動作時使用。

ExitResponse

對 MCA 的回應，指出安全交換中的下一個階段。

ExitResponse2
額外的控制旗標，用來控管 MCA 如何解譯安全結束程式的回應。

ExitUser 區域
16 個位元組 (上限) 的儲存體，可由「安全結束程式」用來維護呼叫之間的狀態。

ExitData
包含通道定義的 SCYDATA 欄位中指定的資料 (右側以空白填補 32 個位元組)。

PMQVOID pChannel 定義
輸入/輸出

MQCD 結構的指標-強制轉型為 PMQCD 以存取欄位。 此參數包含通道的定義。 MQCD 中的下列欄位對「安全結束程式」特別感興趣:

ChannelName
通道名稱 (右側以空白填補 20 個位元組)。

ChannelType
定義通道類型的程式碼。

MCA 使用者 ID
這三個欄位的群組已起始設定為通道定義中指定的 MCAUSER 欄位值。 「安全結束程式」在這些欄位中指定的任何使用者 ID 都會用於存取控制 (不適用於 SDR、SVR、CLNTCONN 或 CLUSSDR 通道)。

MCAUserIdentifier
以空白填補右側 ID 的前 12 個位元組。

LongMCAUserIdPtr
指標指向包含完整長度 ID (不保證以空值結尾) 的緩衝區，其優先順序高於 MCAUserIdentifier。

LongMCAUserIdLength
如果設定 LongMCAUserIdPtr ，則必須設定 LongMCAUserIdPtr 所指向的字串長度。

遠端使用者 ID
僅適用於 CLNTCONN/SVRCONN 通道配對。 如果未定義 CLNTCONN 安全結束程式，則用戶端 MCA 會起始設定這三個欄位，因此它們可能包含用戶端環境中的使用者 ID ，可供「SVRCONN 安全結束程式」鑑別及指定「MCA 使用者 ID」時使用。 如果已定義「CLNTCONN 安全結束程式」，則這些欄位不會起始設定且可由「CLNTCONN 安全結束程式」設定，或者可以使用安全訊息將使用者 ID 從「用戶端」傳遞至「伺服器」。

RemoteUserID
以空白填補右側 ID 的前 12 個位元組。

LongRemoteUserIdPtr
指向包含完整長度 ID (不保證以空值結尾) 之緩衝區的指標優先於 RemoteUserID。

LongRemoteUserId 長度
如果設定 LongRemoteUserIdPtr ，則必須設定 LongRemoteUserIdPtr 所指向的字串長度。

PMQLONG pData 長度
輸入/輸出

MQLONG 的指標。 包含呼叫「安全結束程式」時 AgentBuffer 中所包含之任何「安全結束程式」的長度。 必須由「安全結束程式」設定為在 AgentBuffer 或 ExitBuffer 中傳送之任何訊息的長度。

PMQLONG pAgentBufferLength
輸入

MQLONG 的指標。 呼叫「安全結束程式」時 AgentBuffer 中包含的資料長度。

PMQVOID pAgent 緩衝區
輸入/輸出

在呼叫「安全結束程式」時，這會指向從夥伴結束程式傳送的任何訊息。 如果 MQCXP 結構中的 ExitResponse2 已設定 MQXR2_USE_AGENT_BUFFER 旗標 (預設值) ，則「安全結束程式」需要將此參數設為指向正在傳送的任何訊息資料。

PMQLONG pExitBufferLength
輸入/輸出

MQLONG 的指標。 在第一次呼叫「安全結束程式」時，此參數會起始設定為 0 ，並在安全交換期間兩次呼叫「安全結束程式」之間保留傳回的值。

PMQPTR pExitBufferAddr
輸入/輸出

在第一次呼叫「安全結束程式」時，此參數會起始設定為空值指標，且在安全交換期間對「安全結束程式」的呼叫之間會維護傳回的值。 如果在 MQCXP 結構的 ExitResponse2 中設定 MQXR2_USE_EXIT_BUFFER 旗標，則「安全結束程式」需要將此參數設為指向正在傳送的任何訊息資料。

在 *CLNTCONN/SVRCONN* 通道配對及其他通道配對上定義的安全結束程式之間的行為差異
可以在所有類型的通道上定義安全結束程式。 不過， CLNTCONN/SVRCONN 通道配對上所定義之安全結束程式的行為，與其他通道配對上所定義的安全結束程式略有不同。

CLNTCONN 通道上的「安全結束程式」可以在通道定義中設定「遠端使用者 ID」，以供夥伴 SVRCONN 結束程式處理; 如果未定義「SVRCONN 安全結束程式」，且未設定 SVRCONN 的 MCAUSER 欄位，則為 OAM 授權。

如果未定義「CLNTCONN 安全結束程式」，則用戶端 MCA 會將通道定義中的「遠端使用者 ID」設為用戶端環境中的使用者 ID (可以空白)。

當「SVRCONN 安全結束程式」傳回 MQXCC_OK 的 ExitResponse 時， CLNTCONN 與 SVRCONN 通道配對上定義的「安全結束程式」之間的安全交換會順利完成。 當起始交換的「安全結束程式」傳回 MQXCC_OK 的 ExitResponse 時，其他通道配對之間的安全交換順利完成。

不過， MQXCC_SEND_AND_REQUEST_SEC_MSG ExitResponse 程式碼可用來強制繼續安全交換: 如果 CLNTCONN 或「SVRCONN 安全結束程式」傳回 MQXCC_SEND_AND_REQUEST_SEC_MSG 的 ExitResponse ，則友機結束程式必須透過傳送安全訊息 (不是 MQXCC_OK 或空值回應) 或通道終止來回應。 對於在其他類型的通道上定義的「安全結束程式」， MQXCC_OK 的 ExitResponse 會從夥伴「安全結束程式」傳回對 MQXCC_SEND_AND_REQUEST_SEC_MSG 的回應，導致安全交換繼續進行，如同已傳回空值回應，而不是通道終止一樣。

SSPI 安全結束程式
IBM MQ for Windows 提供安全結束程式，透過使用「安全服務程式設計介面 (SSPI)」為 IBM MQ 通道提供鑑別。 SSPI 提供 Windows 的整合式安全機能。

這個安全結束程式同時適用於 IBM MQ 用戶端和 IBM MQ 伺服器。

安全套件是從 security.dll 或 secur32.dll 載入。 這些 DLL 隨您的作業系統一起提供。

在 Windows 上使用 NTLM 鑑別服務提供單向鑑別。 在 Windows 2000 上使用 Kerberos 鑑別服務提供雙向鑑別。

以來源及物件格式提供安全跳出程式。 您可以依現狀使用物件程式碼，也可以使用原始碼作為起點來建立您自己的使用者結束程式。 如需使用 SSPI 安全結束程式的物件或原始碼的相關資訊，請參閱 第 954 頁的『在 Windows 上使用 SSPI 安全結束程式』

通道傳送及接收結束程式
您可以使用傳送及接收結束程式來執行資料壓縮及解壓縮等作業。 您可以指定要連續執行的傳送及接收結束程式清單。

在 MCA 處理週期的下列位置會呼叫通道傳送及接收結束程式:

- 會呼叫傳送及接收結束程式，以在 MCA 起始設定時起始設定，以及在 MCA 終止時終止。
- 傳送結束程式會在通道的一端或另一端呼叫，視在透過鏈結傳送傳輸之前，傳送一個訊息傳送的端而定。 附註 4 說明為何即使訊息通道只會以一個方向傳送訊息，也可以雙向使用結束程式。
- 在從鏈結取得傳輸之後，會立即在通道的一端或另一端呼叫接收結束程式，視接收一個訊息傳送的傳輸結束端而定。 附註 4 說明為何即使訊息通道只會以一個方向傳送訊息，也可以雙向使用結束程式。

一個訊息傳送可能有許多傳輸，而且在訊息到達接收端的訊息結束程式之前，傳送和接收結束程式可能會有許多反覆運算。

通道傳送及接收結束程式會傳送代理程式緩衝區，其中包含從通訊鏈結傳送或接收的傳輸資料。 對於傳送跳出程式，緩衝區的前 8 個位元組保留給 MCA 使用，且不得變更。 如果程式傳回不同的緩衝區，則這些前 8 個位元組必須存在於新緩衝區中。 未定義呈現給跳出程式的資料格式。

傳送及接收結束程式必須傳回良好的回應碼。 任何其他回應都會導致 MCA 異常結束 (異常終止)。

註: 請勿從傳送或接收結束程式在同步點內發出 MQGET、MQPUT 或 MQPUT1 呼叫。

註:

1. 傳送和接收結束程式通常成對運作。 例如，傳送結束程式可能會壓縮資料，而接收結束程式會解壓縮它，或者傳送結束程式可能會將資料加密，而接收結束程式會將它解密。 當您定義適當的通道時，請確定已為通道兩端命名相容的結束程式。
2. 如果針對通道開啟壓縮，則會傳遞壓縮資料給結束程式。
3. 對於非應用程式資料 (例如，狀態訊息) 的訊息區段，可能會呼叫通道傳送及接收結束程式。 在啟動對話框期間不會呼叫它們，也不會呼叫安全檢查階段。
4. 雖然訊息通道只會以單一方向傳送訊息，但通道控制資料 (例如活動訊號及批次處理結束)、雙向的流程以及雙向的這些結束程式也可以使用。 不過，部分起始通道啟動資料流程不受任何結束程式處理。
5. 在某些情況下，可能會依序呼叫傳送及接收結束程式; 例如，如果您正在執行一系列結束程式，或如果您也在執行安全結束程式。 然後，當第一次呼叫接收結束程式來處理資料時，它可能會接收尚未通過對應傳送結束程式的資料。 如果接收結束程式剛執行作業 (例如解壓縮) ，而未先檢查是否需要它，則結果會是非預期的。

   您需要撰寫傳送及接收結束程式的程式碼，以便接收結束程式可以檢查其接收的資料是否已由對應的傳送結束程式處理。 這樣做的建議方式是撰寫結束程式的程式碼，以便:

   - 在執行作業之前，傳送結束程式會將資料的第九個位元組值設為 0 ，並將所有資料移至 1 個位元組。 (前 8 個位元組保留給 MCA 使用。)
   - 如果接收結束程式收到位元組 9 中 0 的資料，它會知道資料來自傳送結束程式。 它會移除 0 ，執行補充作業，並將產生的資料移回 1 個位元組。
   - 如果接收結束程式收到的資料不是位元組 9 中的 0 ，則會假設傳送結束程式未執行，並將資料傳回給呼叫程式，而不會變更。

   使用安全結束程式時，如果通道由安全結束程式結束，則可能呼叫傳送結束程式而沒有對應的接收結束程式。 防止此問題的一種方法是撰寫安全結束程式的程式碼，以在 MQCD.SecurityUserData 或 MQCD.SendUserData 中設定旗標，例如，當結束程式決定結束通道時。 然後，傳送結束程式需要檢查此欄位，並僅在未設定旗標時處理資料。 這項檢查可防止傳送結束程式不必要地變更資料，從而防止安全結束程式收到變更資料時可能發生的任何轉換錯誤。

通道傳送結束程式-保留空間
您可以在傳輸之前使用傳送及接收結束程式來轉換資料。 通道傳送結束程式可以透過保留傳輸緩衝區中的空間，來新增自己的轉換相關資料。

此資料由接收跳出程式處理，然後從緩衝區中移除。 例如，您可能想要加密資料並新增用於解密的安全金鑰。

如何保留空間並使用它

當呼叫傳送結束程式以進行起始設定時，請將 MQXCP 的 *ExitSpace* 欄位設為要保留的位元組數。 如需詳細資料，請參閱 MQCXP 。 只有在起始設定期間，即當 *ExitReason* 具有值 MQXR_INIT 時，才能設定 *ExitSpace* 。 當在傳輸之前立即呼叫傳送結束程式時，如果 *ExitReason* 設為 MQXR_XMIT ，則會在傳輸緩衝區中保留 *ExitSpace* 個位元組。 z/OS 不支援 *ExitSpace* 。

傳送結束程式不需要使用所有保留空間。 它可以使用少於 *ExitSpace* 個位元組，或者如果傳輸緩衝區未滿，則結束程式可以使用超過保留的數量。 設定 *ExitSpace* 的值時，您必須至少為傳輸緩衝區中的訊息資料保留 1 KB。 如果將保留空間用於大量資料，則通道效能可能會受到影響。

傳輸緩衝區通常是 32KB 長。 不過，如果通道使用 TLS，則傳輸緩衝區大小會減少至 15,352 個位元組，以符合 RFC 6101 及相關 TLS 標準系列所定義的記錄長度上限。 另外保留 1024 個位元組供 IBM MQ 使用，因此傳送結束程式可用的傳輸緩衝空間上限為 14,328 個位元組。

通道接收端發生的情況

通道接收結束程式必須設定為與對應的傳送結束程式相容。 接收結束程式必須知道保留空間中的位元組數，且必須移除該空間中的資料。

多個傳送結束程式

您可以指定要連續執行的傳送及接收結束程式清單。 IBM MQ 會維護所有傳送結束程式所保留的空間總計。此空間總計必須至少為傳輸緩衝區中的訊息資料保留 1 KB。

下列範例顯示如何為連續呼叫的三個傳送結束程式配置空間:

1. 當呼叫起始設定時:
   - 傳送結束程式 A 保留 1 KB。
   - 傳送結束程式 B 保留 2 KB。
   - 傳送結束程式 C 保留 3 KB。
2. 傳輸大小上限為 32 KB，使用者資料長度為 5 KB。
3. 結束程式 A 稱為 5 KB 資料; 最多可以使用 27 KB，因為 5 KB 保留給結束程式 B 及 C。 結束 A 會新增 1 KB，即它所保留的數量。
4. 結束程式 B 是以 6 KB 的資料來呼叫; 最多可以使用 29 KB，因為 3 KB 保留給結束程式 C。 結束程式 B 會新增 1 KB，小於保留的 2 KB。
5. 結束程式 C 稱為 7 KB 資料; 最多可以使用 32 KB。 Exit C 會新增 10K，超過它保留的 3 KB。 此數量是有效的，因為資料總量 (17 KB) 小於上限 32 KB。

使用 TLS 之通道的傳輸緩衝區大小上限是 15,352 個位元組，而不是 32KB。 這是因為基礎安全 Socket 傳輸區段限制為 16KB，且 TLS 記錄額外需要部分空間。 另外保留 1024 個位元組供 IBM MQ 使用，因此傳送結束程式可用的傳輸緩衝空間上限為 14,328 個位元組。

通道訊息結束程式
您可以使用通道訊息結束程式來執行作業，例如對鏈結進行加密、驗證或替代送入的使用者 ID、訊息資料轉換、日誌登載及參照訊息處理。 您可以指定要連續執行的訊息結束程式清單。

在 MCA 處理週期的下列位置會呼叫通道訊息跳出程式:

- 在 MCA 起始及終止時
- 在傳送端 MCA 發出 MQGET 呼叫之後立即
- 在接收 MCA 發出 MQPUT 呼叫之前

訊息結束程式會傳遞一個代理程式緩衝區，其中包含傳輸佇列標頭 MQXQH，以及從佇列擷取的應用程式訊息文字。 MQXQH 的格式在 MQXQH-Transmission-queue 標頭中提供。

如果您使用參照訊息 (亦即，只包含指向要傳送之其他物件的標頭的訊息)，則訊息結束程式會辨識標頭 MQRMH。 它會識別物件，以適當的方式將它附加至標頭，並將它傳遞至 MCA 以傳輸至接收 MCA。 在接收端 MCA 時，另一個訊息結束程式會辨識此訊息是參照訊息，並擷取物件，然後將標頭傳遞至目的地佇列。如需參照訊息及處理它們的部分範例訊息結束程式的相關資訊，請參閱 第 665 頁的『參照訊息』 及 第 929 頁的『執行參照訊息範例』。

訊息結束程式可能會傳回下列回應:

- 傳送訊息 (GET 結束程式)。 結束程式可能已變更訊息。 (這會傳回 MQXCC_OK。)
- 將訊息放置在佇列上 (PUT 結束程式)。 結束程式可能已變更訊息。 (這會傳回 MQXCC_OK。)
- 不處理訊息。 MCA 會將訊息放置在無法傳送郵件的佇列 (未遞送的訊息佇列) 上。
- 關閉通道。

- 回覆碼不正確，導致 MCA 異常結束。

註:

1. 對於每一個傳送的完整訊息，都會呼叫一次訊息結束程式，即使訊息分割成部分也一樣。
2. Linux AIX 如果您在 AIX 或 Linux 上提供訊息結束程式，則使用者 ID 自動轉換為小寫字元 (說明 這裡) 不會運作。
3. 結束程式會在與 MCA 本身相同的執行緒中執行。 它也在與 MCA 相同的工作單元 (UOW) 內執行，因為它使用相同的連線控點。 因此，在同步點下進行的任何呼叫都會在批次結束時由通道確定或取消。 例如，一個通道訊息跳出程式可以將通知訊息傳送至另一個通道訊息跳出程式，且只有在確定包含原始訊息的批次時，才會將這些訊息確定至佇列。

   因此，您可以從通道訊息結束程式發出同步點 MQI 呼叫。

訊息結束程式之外的訊息轉換
在呼叫訊息結束程式之前，接收 MCA 會對訊息執行部分轉換。 本主題說明用來執行轉換的演算法。

處理哪些標頭

在呼叫訊息結束程式之前，轉換常式會在接收端的 MCA 中執行。 轉換常式在訊息開頭以 MQXQH 標頭開始。 然後轉換常式會透過 MQXQH 之後的鏈結標頭來處理，並在必要時執行轉換。 鏈結的標頭可以延伸超出傳遞至接收端訊息結束程式之 MQCXP 資料的 HeaderLength 參數所包含的偏移。 下列標頭會就地轉換:

- MQXQH (格式名稱 " MQXMIT ")
- MQMD (此標頭是 MQXQH 的一部分，沒有格式名稱)
- MQMDE (格式名稱 " MQHMDE ")
- MQDH (格式名稱 " MQHDIST ")
- MQWIH (格式名稱 " MQHWIH ")

下列標頭不會轉換，但會隨著 MCA 繼續處理鏈結的標頭而逐步執行:

- MQDLH (格式名稱 " MQDEAD ")
- 格式名稱以三個字元 'MQH' 開頭的任何標頭 (例如 " MQHRF ") 未被提及的

如何處理標頭

MCA 會讀取每一個 IBM MQ 標頭的 Format 參數。 「格式」參數在標頭內是 8 個位元組，即包含名稱的 8 個單位元組字元。

然後， MCA 會將每一個標頭後面的資料解譯為具名類型。 如果「格式」是適用於 IBM MQ 資料轉換的標頭類型名稱，則會轉換它。 如果它是另一個名稱，指出非 MQ 資料 (例如 MQFMT_NONE 或 MQFMT_STRING) ，則 MCA 會停止處理標頭。

何謂 MQCXP HeaderLength?

提供給訊息結束程式之 MQCXP 資料中的 HeaderLength 參數是訊息開頭的 MQXQH (包括 MQMD)、MQMDE 及 MQDH 標頭的總長度。 這些標頭會使用「格式」名稱和長度來鏈結。

MQWIH

鏈結的標頭可以延伸超過 HeaderLength 到使用者資料區。 MQWIH 標頭 (如果存在的話) 是出現在 HeaderLength 以外的那些標頭之一。

如果鏈結標頭中有 MQWIH 標頭，則會在呼叫接收端的訊息結束程式之前就地轉換。

通道訊息重試結束程式
嘗試開啟目標佇列失敗時，會呼叫通道訊息重試結束程式。 您可以使用結束程式來決定要重試的情況、重試的次數，以及重試頻率。

在 MCA 起始及終止時，也會在通道接收端呼叫此結束程式。

通道訊息重試結束程式會傳遞一個代理程式緩衝區，其中包含從佇列擷取的傳輸佇列標頭、MQXQH 及應用程式訊息文字。 MQXQH 概觀中提供 MQXQH 的格式。

會針對所有原因碼來呼叫結束程式; 結束程式會決定它要 MCA 重試的原因碼、重試次數及間隔。 (定義通道時所設定的訊息重試次數值會傳遞至 MQCD 中的結束程式，但結束程式可以忽略此值。)

每次呼叫結束程式時， MCA 都會增加 MQCXP 中的 MsgRetry 計數欄位，且結束程式會傳回 MQXCC_OK (具有 MQCXP 的 MsgRetry 間隔欄位中包含的等待時間) 或 MQXCC_SUPPRESS_FUNCTION。 繼續無限期重試，直到結束程式在 MQCXP 的 ExitResponse 欄位中傳回 MQXCC_SUPPRESS_FUNCTION 為止。 如需 MCA 針對這些完成碼所採取之動作的相關資訊，請參閱 MQCXP 。

如果所有重試都不成功，則會將訊息寫入無法傳送郵件的佇列。 如果沒有可用的無法傳送郵件的佇列，通道會停止。

如果您未定義通道的訊息重試結束程式，且發生可能是暫時的失敗 (例如 MQRC_Q_FULL) ，則 MCA 會使用定義通道時所設定的訊息重試次數及訊息重試間隔。 如果失敗更永久，且您尚未定義跳出程式來處理它，則會將訊息寫入無法傳送郵件的佇列。

通道自動定義結束程式
當收到啟動接收端或伺服器連線通道的要求，但沒有該通道的定義 (不適用於 IBM MQ for z/OS) 時，可以使用通道自動定義結束程式。 也可以在叢集傳送端和叢集接收端通道的所有平台上呼叫它，以容許修改通道實例的定義。

當收到啟動接收端或伺服器連線通道的要求，但通道定義不存在時，可以在 z/OS 以外的所有平台上呼叫通道自動定義結束程式。 您可以使用它來修改自動定義的接收端或伺服器連線通道 SYSTEM.AUTO.RECEIVER 或 SYSTEM.AUTO.SVRCON。 如需如何自動建立通道定義的說明，請參閱 準備通道 。

當收到啟動叢集傳送端通道的要求時，也可以呼叫通道自動定義結束程式。 可以針對叢集傳送端和叢集接收端通道呼叫它，以容許修改此通道實例的定義。 在此情況下，結束程式也適用於 IBM MQ for z/OS。 通道自動定義結束程式的一般用途是變更訊息結束程式 (MSGEXIT、RCVEXIT、SCYEXIT 及 SENDEXIT) 的名稱，因為結束程式名稱在不同平台上有不同的格式。 如果未指定通道自動定義結束程式，則 z/OS 上的預設行為是檢查 *[path]/libraryname(function)* 格式的分散式結束程式名稱，並使用最多 8 個字元的函數 (如果有的話) 或 libraryname。 在 z/OS 上，通道自動定義結束程式必須變更 MsgExitPtr、 MsgUserDataPtr、SendExitPtr、 SendUserDataPtr、 ReceiveExitPtr 及 ReceiveUserDataPtr 所定址的欄位，而不是 MsgExit、 MsgUserData、 SendExit、 SendUserData、 ReceiveExit 和 ReceiveUser 資料欄位本身。

如需相關資訊，請參閱 使用自動定義的通道。

與其他通道結束程式一樣，參數清單為:

```
MQ_CHANNEL_AUTO_DEF_EXIT (ChannelExitParms, ChannelDefinition)
```

`ChannelExitParms` 在 MQCXP 中有說明。 `ChannelDefinition` 在 MQCD 中有說明。

MQCD 包含預設通道定義中使用的值 (如果結束程式未變更它們的話)。 結束程式只能修改欄位的子集; 請參閱 MQ_CHANNEL_AUTO_DEF_EXIT。 不過，嘗試變更其他欄位不會導致錯誤。

通道自動定義結束程式會傳回 MQXCC_OK 或 MQXCC_SUPPRESS_FUNCTION 的回應。 如果這兩個回應都未傳回，則 MCA 會繼續處理，好像已傳回 MQXCC_SUPPRESS_FUNCTION 一樣。 也就是說，已放棄自動定義，未建立新的通道定義，且通道無法啟動。

ALW 在 *AIX, Linux, and Windows* 系統上編譯通道結束程式

請使用下列範例來協助您編譯 AIX, Linux, and Windows 系統的通道結束程式。

Windows

Windows

Windows 上通道結束程式的編譯器及鏈結器指令:

```
cl.exe /Ic:\mqm\tools\c\include /nologo /c myexit.c
link.exe /nologo /dll myexit.obj /def:myexit.def /out:myexit.dll
```

AIX and Linux 系統

Linux AIX

在這些範例中， `exit` 是程式庫名稱，而 `ChannelExit` 是函數名稱。 在 AIX 上，匯出檔稱為 `exit.exp`。 通道定義會使用這些名稱，以使用 MQCD-通道定義中說明的格式來參照結束程式。 另請參閱 DEFINE CHANNEL 指令的 MSGEXIT 參數。

AIX AIX 上通道結束程式的編譯器及鏈結器指令範例:

```
$ xlc_r -q64 -e MQStart -bE:exit.exp -bM:SRE -o /var/mqm/exits64/exit
exit.c -I/usr/mqm/inc
```

Linux Linux 上通道結束程式的編譯器及鏈結器指令範例，其中佇列管理程式為 32 位元:

```
$ gcc -shared -fPIC -o /var/mqm/exits/exit exit.c -I/opt/mqm/inc
```

Linux 在佇列管理程式為 64 位元的 Linux 上，通道結束程式的編譯器及鏈結器指令範例:

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/exit exit.c -I/opt/mqm/inc
```

在用戶端上，可以使用 32 位元或 64 位元結束程式。 此結束程式必須鏈結至 mqic_r。

AIX 在 AIX 上，必須匯出 IBM MQ 所呼叫的所有函數。 此 make 檔的範例匯出檔:

```
#
!channelExit
MQStart
```

配置通道結束程式

如果要呼叫通道結束程式，您必須在通道定義中命名它。

通道結束程式必須在通道定義中命名。 您可以在第一次定義通道時執行此命名，也可以稍後使用例如 MQSC 指令 ALTER CHANNEL 來新增資訊。 您也可以提供 MQCD 通道資料結構中的通道結束程式名稱。 結束程式名稱的格式視您的 IBM MQ 平台而定; 如需相關資訊，請參閱 MQCD 或 MQSC 指令 。

如果通道定義不包含使用者結束程式名稱，則不會呼叫使用者結束程式。

通道自動定義結束程式是佇列管理程式的內容，而不是個別通道。 若要呼叫此結束程式，必須在佇列管理程式定義中指定它。 若要變更佇列管理程式定義，請使用 MQSC 指令 ALTER QMGR。

寫入資料-轉換結束程式

此主題集合包含如何撰寫資料轉換結束程式的相關資訊。

註: 在 MQSeries for VSE/ESA 中不受支援。

當您執行 MQPUT 時，您的應用程式會建立訊息的訊息描述子 (MQMD)。 因為不論 MQMD 建立在哪個平台上， IBM MQ 都需要能夠瞭解 MQMD 的內容，所以系統會自動轉換它。

不過，不會自動轉換應用程式資料。 如果在 `CodedCharSetId` 和 `Encoding` 欄位不同的平台之間交換字元資料，例如在 ASCII 和 EBCDIC 之間，應用程式必須安排訊息的轉換。 應用程式資料轉換可以由佇列管理程式本身或使用者結束程式 (稱為 資料轉換結束程式) 執行。 如果應用程式資料採用其中一種內建格式 (例如 MQFMT_STRING) ，則佇列管理程式可以使用其中一個內建轉換常式自行執行資料轉換。 本主題包含當應用程式資料不是內建格式時， IBM MQ 所提供之資料轉換結束程式機能的相關資訊。

在 MQGET 呼叫期間，可以將控制項傳遞至資料轉換結束程式。 這可避免在到達最終目的地之前跨不同平台進行轉換。 不過，如果最終目的地是在 MQGET 上不支援資料轉換的平台，則必須在將資料傳送至其最終目

的地的傳送端通道上指定 CONVERT (YES)。 這可確保 IBM MQ 在傳輸期間轉換資料。 在此情況下，您的資料轉換結束程式必須位於定義傳送端通道的系統上。

MQGET 呼叫是由應用程式直接發出。 將 MQMD 中的 CodedCharSetId 及 Encoding 欄位設為必要的字集及編碼。 如果您的應用程式使用與佇列管理程式相同的字集及編碼，請將 CodedCharSetId 設為 MQCCSI_Q_MGR，並將 Encoding 設為 MQENC_NATIVE。 MQGET 呼叫完成之後，這些欄位具有適用於所傳回訊息資料的值。 如果轉換不成功，則這些值可能與所需的值不同。 您的應用程式應該將這些欄位重設為每一個 MQGET 呼叫之前所需的值。

針對 MQGET 中的 MQGET 呼叫，定義要呼叫資料轉換結束程式所需的條件。

如需傳遞至資料轉換結束程式之參數的說明，以及詳細使用注意事項，請參閱 MQ_DATA_CONV_EXIT 呼叫及 MQDXP 結構的 資料轉換 。

在不同機器編碼與 CCSID 之間轉換應用程式資料的程式必須符合 IBM MQ 資料轉換介面 (DCI)。

對於「多重播送」用戶端， API 結束程式及資料轉換結束程式必須能夠在用戶端上執行，因為部分訊息可能不會通過佇列管理程式。 下列程式庫是用戶端套件以及伺服器套件的一部分:

表 *143:* 用戶端和伺服器套件中的程式庫

作業系統	**圖書館業**
AIX	32 bit & 64 bit: libmqm.a & libmqm_r.a
IBM i	LIBMQM 與 LIBMQM_R
Linux	32 bit & 64 bit: libmqm.so & libmqm_r.so
Windows	32 bit & 64 bit: mqm.dll & mqm.pdb

呼叫資料轉換結束程式

資料轉換結束程式是使用者撰寫的結束程式，在處理 MQGET 呼叫期間接收控制。

如果下列陳述式為真，則會呼叫結束程式:

- MQGET 呼叫中指定 MQGMO_CONVERT 選項。
- 部分或所有訊息資料不在所要求的字集或編碼中。
- 與訊息相關聯的 MQMD 結構中的 *Format* 欄位不是 MQFMT_NONE。
- MQGET 呼叫中指定的 *BufferLength* 不是零。
- 訊息資料長度不是零。
- 訊息包含具有使用者定義格式的資料。 使用者定義格式可以佔用整個訊息，或前面有一或多個內建格式。例如，使用者定義格式之前可能有 MQFMT_DEAD_LETTER_HEADER 格式。 呼叫結束程式以僅轉換使用者定義格式; 佇列管理程式會轉換使用者定義格式之前的任何內建格式。

  也可以呼叫使用者撰寫的結束程式來轉換內建格式，但只有在內建轉換常式無法順利轉換內建格式時，才會發生這種情況。

還有一些其他條件，如 MQ_DATA_CONV_EXIT 中 MQ_DATA_CONV_EXIT 呼叫的使用注意事項。

如需 MQGET 呼叫的詳細資料，請參閱 MQGET 。 資料轉換結束程式無法使用 MQXCNVC 以外的 MQI 呼叫。

自從應用程式連接至佇列管理程式之後，當應用程式嘗試擷取使用該 *Format* 的第一則訊息時，會載入新的結束程式副本。 如果佇列管理程式已捨棄先前載入的副本，則在其他時間也可能會載入新的副本。

資料轉換結束程式在類似發出 MQGET 呼叫之程式的環境中執行。 除了使用者應用程式，程式也可以是 MCA (訊息通道代理程式)，將訊息傳送至不支援訊息轉換的目的地佇列管理程式。 環境包括位址空間及使用者設定檔 (如果適用的話)。 結束程式無法危害佇列管理程式的完整性，因為它不會在佇列管理程式的環境中執行。

z/OS 上的資料轉換

z/OS

在 z/OS 上，請注意下列事項:

- 跳出程式只能以組合語言撰寫。
- 跳出程式必須重新進入，並且能夠在儲存體中的任何位置執行。
- 跳出程式必須將結束時的環境還原至進入時的環境，並且必須釋放取得的任何儲存體。
- 跳出程式不得 WAIT ，或發出 ESTAE 或 SPIE。
- 通常會呼叫結束程式，如同在下列項目中由 z/OS LINK 所呼叫:
  - 未獲授權的問題程式狀態
  - 主要位址空間控制模式
  - 非跨記憶體模式
  - 非存取-登錄模式
  - 31 位元定址模式
  - TCB-PRB 模式
- 當由 CICS 應用程式使用時， EXEC CICS LINK 會呼叫結束程式，且必須符合 CICS 程式設計慣例。 參數由 CICS 通訊區域 (COMMAREA) 中的指標 (位址) 傳遞。

  雖然不建議使用，但使用者結束程式也可以使用 CICS API 呼叫，請注意下列事項:

  - 請勿發出同步點，因為結果可能會影響 MCA 所宣告的工作單元。
  - 請勿更新 IBM MQ for z/OS 以外的資源管理程式所控制的任何資源，包括 CICS Transaction Server 所控制的資源。

  對於 CONVERT = YES 的通道，會從 CSQXLIB DD 陳述式所參照的資料集載入結束程式。 MQ 提供的結束程式 CSQCBDCI 及 IBM MQ CICS Bridge 的 CSQCBDCO 位於 SCSQAUTH 中。

撰寫 *IBM i* 的資料轉換結束程式

撰寫 IBM i 的 MQ 資料轉換結束程式時要考量之步驟的相關資訊。

請遵循下列步驟:

1. 命名您的訊息格式。 名稱必須符合 MQMD 的 *Format* 欄位。 *Format* 名稱不得有前導內含空白，且尾端空白會被忽略。 物件名稱不得超過八個非空白字元，因為 *Format* 的長度只有八個字元。 每次傳送訊息時，請記得使用此名稱 (我們的範例使用名稱「格式」)。
2. 建立結構以代表您的訊息。 如需範例，請參閱 有效語法 。
3. 透過 CVTMQMDTA 指令執行此結構，以建立資料轉換結束程式的程式碼片段。

   CVTMQMDTA 指令所產生的函數會使用 QMQM/H (AMQSVMHA) 檔案中隨附的巨集。 這些巨集是在假設所有結構都已壓縮的情況下撰寫的; 如果不是這種情況，則會對它們進行修正。
4. 取得所提供架構來源檔 QMQMSAMP/QCSRC (AMQSVFC4) 的副本並重新命名。 (我們的範例使用名稱 EXIT_MOD。)
5. 在原始檔中尋找下列註解方框，並依照說明來插入程式碼:

   a. 在原始檔結尾，註解方框的開頭為:

   ```
   /* Insert the functions produced by the data-conversion exit */
   ```

   在這裡，插入在步驟 第 825 頁的『3』 中產生的程式碼片段。

   b. 接近原始檔中間的註解框開頭為:

   ```
   /* Insert calls to the code fragments to convert the format's */
   ```

   後面接著對函數 ConverttagSTRUCT 的註銷呼叫。

將函數名稱變更為您在步驟 第 825 頁的『5.a』中新增的函數名稱。 移除註解字元以啟動函數。 如果有數個函數，請為每一個函數建立呼叫。

c. 在原始檔的開頭附近，註解方框以下列開頭:

```
/* Insert the function prototypes for the functions produced by */
```

在這裡，針對步驟 第 825 頁的『5.a』中新增的函數插入函數原型陳述式。

如果訊息包含字元資料，則產生的程式碼會呼叫 MQXCNVC; 可以透過連結服務程式 QMQM/LIBMQM 來解決此問題。

6. 編譯來源模組 EXIT_MOD ， 如下所示:

```
CRTCMOD MODULE(library/EXIT_MOD) +
SRCFILE(QCSRC) +
TERASPACE(*YES *TSIFC)
```

7. 建立/鏈結程式。

若為非執行緒應用程式，請使用下列:

```
CRTPGM PGM(library/Format)  +
   MODULE(library/EXIT_MOD)  +
   BNDSRVPGM(QMQM/LIBMQM)  +
   ACTGRP(QMQM)  +
   USRPRF(*USER)
```

除了建立基本環境的資料轉換結束程式之外，執行緒環境中還需要另一個。 這個可載入物件後面必須接著 _R。 請使用 LIBMQM_R 程式庫來解析 MQXCNVC 的呼叫。 執行緒環境需要兩個可載入物件。

```
CRTPGM PGM(library/Format_R)  +
   MODULE(library/EXIT_MOD)  +
   BNDSRVPGM(QMQM/LIBMQM_R)  +
   ACTGRP(QMQM)  +
   USRPRF(*USER)
```

8. 將輸出置於 IBM MQ 工作的檔案庫清單中。 對於正式作業，建議將資料轉換跳出程式儲存在 QSYS 中。

註:

1. 如果 CVTMQMDTA 使用壓縮結構，則所有 IBM MQ 應用程式都必須使用 _Packed 限定元。
2. 資料轉換跳出程式必須重新進入。
3. MQXCNVC 是唯一可以從資料轉換結束程式發出的 MQI 呼叫。
4. 在使用者設定檔編譯器選項設為 *USER 的情況下編譯跳出程式，以便跳出程式以使用者的權限執行。
5. 使用 IBM MQ for IBM i 的所有使用者結束程式都需要啟用兆空間記憶體; 在 CRTCMOD 及 CRTBNDC 指令中指定 TERASPACE (*YES *TSIFC)。

撰寫 *IBM MQ for z/OS* 的資料轉換結束程式

撰寫 IBM MQ for z/OS 的資料轉換結束程式時要考量之步驟的相關資訊。

請遵循下列步驟:

1. 採用所提供的來源架構 CSQ4BAX9 (適用於非 CICS 環境) 或 CSQ4CAX9 (適用於 CICS) 作為您的起點。
2. 執行 CSQUCVX 公用程式。
3. 遵循 CSQ4BAX9 或 CSQ4CAX9 的前言中的指示，以您要轉換的訊息中出現結構的順序納入 CSQUCVX 公用程式所產生的常式。
4. 公用程式會假設未壓縮資料結構、允許使用隱含的資料對齊方式，以及結構從全字組界限開始，並根據需要跳過位元組 (如 有效語法 範例中的 ID 與 VERSION 之間)。 如果結構已壓縮，請省略所產生的 CMQXCALA 巨集。 因此，請考慮以這樣的方式來宣告您的結構: 所有欄位都已命名，且未跳過任何位元組; 在 有效語法的範例中，請在 ID 與 VERSION 之間新增欄位 "MQBYTE DUMMY;" 。

5. 如果輸入緩衝區短於要轉換的訊息格式，則提供的結束程式會傳回錯誤。 雖然結束程式會盡可能轉換多個完整欄位，但此錯誤會導致未轉換的訊息傳回至應用程式。 如果您想要儘可能轉換簡短輸入緩衝區 (包括部分欄位)，請將 CSQXCDFA 巨集上的 TRUNC= 值變更為 YES: 不會傳回任何錯誤，因此應用程式會收到轉換的訊息。 應用程式必須處理截斷。
6. 新增您需要的任何其他特殊處理代碼。
7. 將程式重新命名為您的資料格式名稱。
8. 編譯並鏈結-像批次應用程式一樣編輯您的程式 (除非它與 CICS 應用程式搭配使用)。 公用程式所產生之程式碼中的巨集位於程式庫 **thlqual**.SCSQMACS 中。

   如果訊息包含字元資料，則產生的程式碼會呼叫 MQXCNVC。 如果您的結束程式使用此呼叫，請使用結束程式 Stub 程式 CSQASTUB 來鏈結編輯它。 Stub 與語言無關，與環境無關。 或者，您可以使用動態呼叫名稱 CSQXCNVC 來動態載入 Stub。 如需相關資訊，請參閱第 864 頁的『動態呼叫 IBM MQ Stub』。

   將鏈結編輯的模組放在應用程式載入程式庫中，以及放在通道起始程式所啟動之作業程序的 CSQXLIB DD 陳述式所參照的資料集中。
9. 如果結束程式供 CICS 應用程式使用，請像 CICS 應用程式一樣編譯並鏈結編輯它，必要的話，包括 CSQASTUB。 將它放在 CICS 應用程式庫中。 以一般方式將程式定義為 CICS，並指定 EXECKEY (CICS) 定義中。

註: 雖然執行 CSQUCVX 公用程式需要 LE/370 執行時期程式庫 (請參閱步驟 第 826 頁的『2』)，但鏈結編輯或執行資料轉換結束程式本身並不需要它們 (請參閱步驟 第 827 頁的『8』 和 第 827 頁的『9』)。

如需 IBM MQ - IMS 橋接器內資料轉換的相關資訊，請參閱 第 61 頁的『撰寫 IMS 橋接器應用程式』。

Linux AIX 撰寫 *IBM MQ for AIX or Linux* 系統的資料轉換結束程式

撰寫 IBM MQ for AIX or Linux 系統的資料轉換結束程式時要考量之步驟的相關資訊。

請遵循下列步驟:

1. 命名您的訊息格式。 名稱必須符合 MQMD 的 *Format* 欄位，且為大寫，例如 MYFORMAT。 *Format* 名稱不得有前導空白。 系統不處理尾端空白。 物件名稱不得超過八個非空白字元，因為 *Format* 的長度只有八個字元。 每次傳送訊息時請記得使用此名稱。

   如果在執行緒環境中使用資料轉換結束程式，則可載入物件後面必須接著 _r，以指出它是執行緒版本。
2. 建立結構以代表您的訊息。 如需範例，請參閱 有效語法。
3. 透過 `crtmqcvx` 指令執行此結構，以建立資料轉換結束程式的程式碼片段。

   `crtmqcvx` 指令所產生的函數會使用假設所有結構都已壓縮的巨集; 如果不是這種情況，請修正它們。
4. 複製提供的 Skeleton 原始檔，並將它重新命名為您在步驟 第 827 頁的『1』 中設定的訊息格式名稱。 Skeleton 原始檔和副本是唯讀的。

   架構原始檔稱為 amqsvfc0.c。
5. 在 IBM MQ for AIX 上，也會提供稱為 amqsvfc.exp 的架構匯出檔。 複製此檔案，並將它重新命名為 MYFORMAT.EXP。
6. Skeleton 包括 *MQ_INSTALLATION_PATH*/inc 目錄中的範例標頭檔 amqsvmha.h，其中 *MQ_INSTALLATION_PATH* 代表 IBM MQ 安裝所在的高階目錄。 請確定您的併入路徑指向此目錄，以挑選此檔案。

   amqsvmha.h 檔案包含 `crtmqcvx` 指令所產生之程式碼所使用的巨集。 如果要轉換的結構包含字元資料，則這些巨集會呼叫 MQXCNVC。
7. 在原始檔中尋找下列註解方框，並依照說明來插入程式碼:

   a. 在原始檔結尾，註解方框的開頭為:

   ```
   /* Insert the functions produced by the data-conversion exit */
   ```

   在這裡，插入在步驟 第 827 頁的『3』 中產生的程式碼片段。

   b. 接近原始檔中間的註解框開頭為:

```
/* Insert calls to the code fragments to convert the format's */
```

後面接著對函數 ConverttagSTRUCT 的註銷呼叫。

將函數名稱變更為您在步驟 第 827 頁的『7.a』中新增的函數名稱。 移除註解字元以啟動函數。 如果有數個函數，請為每一個函數建立呼叫。

c. 在原始檔的開頭附近，註解方框以下列開頭:

```
/* Insert the function prototypes for the functions produced by */
```

在這裡，針對步驟 第 827 頁的『3』中新增的函數插入函數原型陳述式。

8. 使用 MQStart 作為進入點，將結束程式編譯成共用程式庫。 若要執行此動作，請參閱 第 828 頁的『在 AIX and Linux 系統上編譯資料轉換結束程式』。
9. 將輸出置於結束目錄中。 若為 32 位元系統，預設結束目錄為 /var/mqm/exits ，若為 64 位元系統，則預設結束目錄為 /var/mqm/exits64。 您可以在 qm.ini 或 mqclient.ini 檔案中變更這些目錄。 每一個佇列管理程式都可以設定此路徑，且只會在該路徑中尋找結束程式。

註:

1. 如果 crtmqcvx 使用壓縮結構，則必須以此方式編譯所有 IBM MQ 應用程式。
2. 資料轉換跳出程式必須重新進入。
3. MQXCNVC 是唯一可以從資料轉換結束程式發出的 MQI 呼叫。

Linux AIX 在 *AIX and Linux* 系統上編譯資料轉換結束程式

如何在 AIX and Linux 系統上編譯資料轉換結束程式的範例。

在所有平台上，模組的進入點是 MQStart。

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

AIX

AIX

發出下列其中一個指令來編譯結束程式碼:

32 位元應用程式

無執行緒

```
cc -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT \
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

執行緒作業

```
xlc_r -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT_r \
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

64 位元應用程式

無執行緒

```
cc -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT \
     MYFORMAT.c -I MQ_INSTALLATION_PATH/inc
```

執行緒作業

```
xlc_r -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT_r \
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

Linux

Linux

發出下列其中一個指令來編譯結束程式碼:

31 位元應用程式

無執行緒

```
 gcc -m31 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c \
      -I  MQ_INSTALLATION_PATH/inc
```

執行緒作業

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

32 位元應用程式

無執行緒

```
 gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c
      -I  MQ_INSTALLATION_PATH/inc
```

執行緒作業

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

64 位元應用程式

無執行緒

```
 gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT MYFORMAT.c
      -I  MQ_INSTALLATION_PATH/inc
```

執行緒作業

```
 gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT_r MYFORMAT.c
      -I  MQ_INSTALLATION_PATH/inc
```

Windows 撰寫 *IBM MQ for Windows* 的資料轉換結束程式

撰寫 IBM MQ for Windows 的資料轉換結束程式時要考量之步驟的相關資訊。

請遵循下列步驟:

1. 命名您的訊息格式。 名稱必須符合 MQMD 的 *Format* 欄位。 *Format* 名稱不得有前導空白。 系統不處理尾端空白。 物件名稱不得超過八個非空白字元，因為 *Format* 的長度只有八個字元。

   範例目錄 *MQ_INSTALLATION_PATH*\Tools\C\Samples 中也提供稱為 amqsvfcn.def 的 .DEF 檔案。 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安裝目錄。 取得此檔案的副本並將其重新命名，例如，將其重新命名為 MYFORMAT.DEF。 請確定要建立的 DLL 名稱及 MYFORMAT.DEF 相同。 以新的格式名稱改寫 MYFORMAT.DEF 中的名稱 FORMAT1 。

   每次傳送訊息時請記得使用此名稱。
2. 建立結構以代表您的訊息。 如需範例，請參閱 有效語法 。
3. 透過 crtmqcvx 指令執行此結構，以建立資料轉換結束程式的程式碼片段。

   CRTMQCVX 指令所產生的函數會使用假設所有結構都已壓縮所撰寫的巨集; 如果不是如此，請修正它們。
4. 複製提供的 Skeleton 原始檔 amqsvfc0.c，並將它重新命名為您在步驟 第 830 頁的『1』 中設定的訊息格式名稱。

   amqsvfc0.c 位於 *MQ_INSTALLATION_PATH*\Tools\C\Samples 中，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安裝目錄。 (預設安裝目錄為 C:\Program Files\IBM\MQ。)

   架構包括 *MQ_INSTALLATION_PATH*\Tools\C\include 目錄中的範例標頭檔 amqsvmha.h 。 請確定您的併入路徑指向此目錄，以挑選此檔案。

   amqsvmha.h 檔案包含 CRTMQCVX 指令所產生之程式碼所使用的巨集。 如果要轉換的結構包含字元資料，則這些巨集會呼叫 MQXCNVC。
5. 在原始檔中尋找下列註解方框，並依照說明來插入程式碼:
   a. 在原始檔結尾，註解方框的開頭為:

      ```
      /* Insert the functions produced by the data-conversion exit */
      ```

      在這裡，插入在步驟 第 830 頁的『3』 中產生的程式碼片段。
   b. 接近原始檔中間的註解框開頭為:

      ```
      /* Insert calls to the code fragments to convert the format's */
      ```

      後面接著對函數 ConverttagSTRUCT 的註銷呼叫。

      將函數名稱變更為您在步驟 第 830 頁的『5.a』 中新增的函數名稱。 移除註解字元以啟動函數。 如果有數個函數，請為每一個函數建立呼叫。
   c. 在原始檔的開頭附近，註解方框以下列開頭:

      ```
      /* Insert the function prototypes for the functions produced by */
      ```

      在這裡，針對步驟 第 830 頁的『3』 中新增的函數插入函數原型陳述式。
6. 建立下列指令檔:

   ```
   cl -I  MQ_INSTALLATION_PATH\Tools\C\Include -Tp   \
   MYFORMAT.C


   MYFORMAT.DEF
   ```

   其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安裝目錄。
7. 請發出指令檔，將您的結束程式編譯成 DLL 檔案。
8. 將輸出放在 IBM MQ 資料目錄下的 exit 子目錄中。 在 32 位元系統上安裝結束程式的預設目錄是 *MQ_DATA_PATH*\Exits ，而在 64 位元系統上則是 *MQ_DATA_PATH*\Exits64

   登錄中提供用來尋找資料轉換結束程式的路徑。 登錄資料夾為:

```
HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere
MQ\Installation\MQ_INSTALLATION_NAME\Configuration\ClientExitPath\
```

登錄機碼為: ExitsDefaultPath。 每一個佇列管理程式都可以設定此路徑，且只會在該路徑中尋找結束程式。

註:

1. 如果 CRTMQCVX 使用壓縮結構，則必須以此方式編譯所有 IBM MQ 應用程式。
2. 資料轉換跳出程式必須重新進入。
3. MQXCNVC 是唯一可以從資料轉換結束程式發出的 MQI 呼叫。

Windows 在 *Windows* 作業系統上結束並切換載入檔案

IBM WebSphere MQ for Windows 7.5 佇列管理程式程序是 32 位元。 因此，當使用 64 位元應用程式時，某些類型的結束程式和 XA 交換器載入檔案也需要有 32 位元版本可供佇列管理程式使用。 如果需要結束程式或 XA 交換器載入檔案的 32 位元版本，且無法使用，則相關 API 呼叫或指令會失敗。

在 qm.ini file for *ExitPath* 中支援兩個屬性。 這些是 ExitsDefaultPath= *MQ_INSTALLATION_PATH*\exits 和 ExitsDefaultPath64= *MQ_INSTALLATION_PATH*\exits64。 *MQ_INSTALLATION_PATH* 代表 IBM MQ 安裝所在的高階目錄。 使用這些可確保可以找到適當的程式庫。 如果在 IBM MQ 叢集中使用結束程式，這也可確保在遠端系統上找到適當的程式庫。

下表列出不同類型的「結束」及「切換」載入檔案，並根據使用 32 位元或 64 位元應用程式，指出是否需要 32 位元或 64 位元版本或兩者:

檔案類型	**32 位元應用程式**	**64 位元應用程式**
API 結束程式 (API exit)	32 位元和 64 位元	64 位元
資料轉換結束程式	32 位元	64 位元
伺服器通道結束程式 (所有類型)	64 位元	64 位元
用戶端通道結束程式 (所有類型)	32 位元	64 位元
可安裝的服務結束程式	64 位元	64 位元
叢集 WLM 結束程式	64 位元	64 位元
發佈/訂閱遞送結束程式	64 位元	64 位元
資料庫交換器載入檔案	32 位元和 64 位元	64 位元
外部交易管理程式 AX 程式庫	32 位元	64 位元
預先連接結束程式	32 位元	64 位元

使用儲存庫中的預先連接結束程式來參照連線定義

IBM MQ MQI clients 可以配置成查閱儲存庫，以使用預先連接結束程式庫來取得連線定義。

簡介

用戶端應用程式可以使用用戶端通道定義表 (CCDT) 連接至佇列管理程式。 一般而言， CCDT 檔案位於中央網路檔案伺服器上，且具有參照它的用戶端。 由於很難管理及管理參照 CCDT 檔案的各種用戶端應用程式，因此彈性方法是將用戶端定義儲存在廣域儲存庫中，例如 LDAP 目錄、 WebSphere 登錄及儲存庫或任何其他儲存庫。 將用戶端連線定義儲存在儲存庫中，可讓管理用戶端連線定義更容易，且應用程式可以存取正確且最新的用戶端連線定義。

在 MQCONN/X 呼叫執行期間， IBM MQ MQI client 會載入應用程式指定的預先連接結束程式庫，並呼叫結束函數以擷取連線定義。 然後會使用擷取的連線定義來建立與佇列管理程式的連線。 要呼叫之結束程式庫及函數的詳細資料指定在 mqclient.ini 配置檔中。

語法

void MQ_PRECONNECT_EXIT (*pExitParms*、 *pQMgrName*、 *ppConnectOpts*、 *pCompCode*、 *pReason*);

參數

pExit 參數
類型 :PMQNXP 輸入/輸出

PreConnection 結束程式參數結構。

該結構由結束程式的呼叫者配置及維護。

pQMgr 名稱
類型 :PMQCHAR 輸入/輸出

佇列管理程式的名稱。

在輸入上，此參數是透過 **QMgrName** 參數提供給 MQCONN API 呼叫的過濾器字串。 此欄位可能是空白、明確或包含某些萬用字元。 結束程式會變更欄位。 以 MQXR_TERM 呼叫結束程式時，參數為 NULL。

ppConnect 選項
類型: ppConnectOpts input/output

控制 MQCONNX 動作的選項。

這是 MQCNO 連線選項結構的指標，可控制 MQCONN API 呼叫的動作。 以 MQXR_TERM 呼叫結束程式時，參數為 NULL。 MQI 用戶端一律提供 MQCNO 結構給結束程式，即使它最初不是由應用程式所提供。 如果應用程式提供 MQCNO 結構，用戶端會複製它，將它傳遞至修改它的結束程式。 用戶端保留 MQCNO 的所有權。

透過 MQCNO 參照的 MQCD 優先於透過陣列提供的任何連線定義。 用戶端會使用 MQCNO 結構來連接佇列管理程式，並忽略其他佇列管理程式。

pComp 程式碼
類型 :PMQLONG 輸入/輸出

完成碼。

指向接收結束程式完成碼之 MQLONG 的指標。 它必須是下列其中一個值:

- MQCC_OK -順利完成
- MQCC_WARNING -警告 (局部完成)
- MQCC_FAILED -呼叫失敗

pReason
類型 :PMQLONG 輸入/輸出

符合 pComp 代碼的原因。

指向接收結束原因碼之 MQLONG 的指標。 如果完成碼為 MQCC_OK ，則唯一有效值為:

- MQRC_NONE-(0 ， x '000') 沒有要報告的原因。

如果完成碼是 MQCC_FAILED 或 MQCC_WARNING ，則結束函數可以將原因碼欄位設為任何有效的 MQRC_ * 值。

C 呼叫

```
void MQ_PRECONNECT_EXIT (&ExitParms, &QMgrName, &pConnectOpts, &CompCode, &Reason);
```

Parameter

```
PMQNXP  pExitParms   /*PreConnect exit parameter structure*/
PMQCHAR pQMgrName    /*Name of the queue manager*/
```

```
PPMQCNO ppConnectOpts/*Options controlling the action of MQCONNX*/
PMQLONG pCompCode    /*Completion code*/
PMQLONG pReason      /*Reason qualifying pCompCode*/
```

撰寫及編譯發佈結束程式

您可以在佇列管理程式上配置發佈結束程式，以在訂閱者收到之前變更已發佈訊息的內容。 您也可以變更訊息標頭，或不將訊息遞送至訂閱。

註: z/OS 不支援發佈結束程式。

您可以使用發佈結束程式來檢查及變更遞送至訂閱者的訊息:

- 檢查發佈至每一個訂閱者之訊息的內容
- 修改發佈至每一個訂閱者的訊息內容
- 變更放置訊息的佇列
- 停止將訊息遞送至訂閱者

撰寫發佈結束程式

使用 第 784 頁的『在 AIX, Linux, and Windows 上撰寫結束程式及可安裝的服務』中的步驟，以協助您撰寫並編譯結束程式。

發佈結束程式的提供者定義結束程式執行的動作。 不過，結束程式必須符合 MQPSXP 中定義的規則。

IBM MQ 未提供 MQ_PUBLISH_EXIT 進入點的實作。 它會提供 C 語言類型定義宣告。 請使用 typedef 來正確地將參數宣告給使用者撰寫的結束程式。 下列範例說明如何使用 typedef 宣告:

```
#include "cmqec.h"

MQ_PUBLISH_EXIT MyPublishExit;

void MQENTRY MyPublishExit( PMQPSXP pExitParms,
                            PMQPBC  pPubContext,
                            PMQSBC  pSubContext )
{
/* C language statements to perform the function of the exit */
}
```

由於下列作業，發佈結束程式會在佇列管理程式處理程序內執行:

- 將訊息遞送至一或多個訂閱者的「發佈」作業
- 遞送一或多個保留訊息的「訂閱」作業
- 遞送一或多個保留訊息的「訂閱要求」作業

如果針對連線呼叫發佈結束程式，則第一次呼叫它時，會設定 *ExitReason* 程式碼 MQXR_INIT 。 在使用發佈結束程式之後中斷連線之前，會使用 *ExitReason* 程式碼 MQXR_TERM 來呼叫結束程式。

如果已配置發佈結束程式，但在啟動佇列管理程式時無法載入，則會禁止佇列管理程式的發佈/訂閱訊息作業。 在重新啟用發佈/訂閱傳訊之前，您必須修正問題或重新啟動佇列管理程式。

每一個需要發佈結束程式的 IBM MQ 連線可能無法載入或起始設定結束程式。 如果結束程式無法載入或起始設定，則該連線會停用需要發佈結束程式的發佈/訂閱作業。 作業失敗， IBM MQ 原因碼 MQRC_PUBLISH_EXIT_ERROR。

呼叫發佈結束程式的環境定義是應用程式與佇列管理程式的連線。 對於執行發佈作業的每一個連線，佇列管理程式會維護使用者資料區。 該結束程式可以在每個連線的使用者資料區中保留資訊。

發佈結束程式可以使用部分 MQI 呼叫。 它只能使用那些操作訊息內容的 MQI 呼叫。 呼叫如下:

- MQBUFMH
- MQCRTMH
- MQDLTMH
- MQDLTMP

- MQMHBUF
- MQINQMP
- MQSETMP

如果發佈結束程式變更目的地佇列管理程式或佇列名稱，則不會執行新的權限檢查。

編譯發佈結束程式

發佈結束程式是動態載入的程式庫; 可以將它視為通道結束程式。 如需編譯結束程式的相關資訊，請參閱 第 784 頁的『在 AIX, Linux, and Windows 上撰寫結束程式及可安裝的服務』。

範例發佈結束程式

範例結束程式稱為 amqspse0.c。 它會根據是否呼叫結束程式來起始設定、發佈或終止作業，將不同的訊息寫入日誌檔。 它也示範如何使用結束使用者區域欄位來適當地配置及釋放儲存體。

配置發佈結束程式

您必須定義某些屬性來配置發佈結束程式。

在 Windows 和 Linux 上，您可以使用 IBM MQ 瀏覽器來定義屬性。 這些屬性定義在「發佈/訂閱」下的佇列管理程式內容頁面中。

若要在 AIX and Linux 系統上的 qm.ini 檔案中配置發佈結束程式，請建立稱為 PublishSubscribe 的段落。 PublishSubscribe 段落具有下列屬性:

PublishExit 路徑 = [路徑] |*module_name*
包含發佈結束碼的模組名稱和路徑。 此欄位的長度上限為 MQ_EXIT_NAME_LENGTH。 預設值是無發佈結束程式。

PublishExit 函數 = *function_name*
包含發佈結束碼之模組的函數進入點名稱。 此欄位的長度上限為 MQ_EXIT_NAME_LENGTH。

IBM i 在 IBM i 上，如果使用程式，請省略 PublishExitFunction。

PublishExit 資料 = 字串
如果佇列管理程式正在呼叫發佈結束程式，則會傳遞 MQPSXP 結構作為輸入。 使用 **PublishExitData** 屬性指定的資料在結構的 *ExitData* 欄位中提供。 字串長度最多可以為 MQ_EXIT_DATA_LENGTH 個字元。 預設值是 32 個空白字元。

撰寫及編譯叢集工作量結束程式

撰寫叢集工作量結束程式，以自訂叢集的工作量管理。 在遞送訊息時，您可能會考慮在一天中不同時間使用通道或訊息內容的成本。 這些是標準工作量管理演算法未考量的因素。

在大部分情況下，工作量管理演算法足以滿足您的需求。 不過，為了讓您可以提供自己的使用者結束程式來自訂工作量管理， IBM MQ 包括使用者結束程式，即叢集工作量結束程式。

您可能有一些網路或訊息的特定相關資訊，可用來影響工作量平衡。 您可能知道哪些是高容量通道或廉價網路路徑，或者您可能想要根據其內容來遞送訊息。 您可以決定撰寫叢集工作量跳出程式，或使用協力廠商提供的程式。

存取叢集佇列時，會呼叫叢集工作量結束程式。 它由 MQOPEN、 MQPUT1 和 MQPUT 呼叫。

如果指定 MQOO_BIND_ON_OPEN ，則會修正 MQOPEN 時選取的目標佇列管理程式。 在此情況下，結束程式只會執行一次。

如果在 MQOPEN 時間未修正目標佇列管理程式，則會在 MQPUT 呼叫時選擇目標佇列管理程式。 如果目標佇列管理程式無法使用，或在訊息仍在傳輸佇列上時失敗，則會重新呼叫結束程式。 已選取新的目標佇列管理程式。 如果在傳送訊息時訊息通道失敗，且訊息已取消，則會選取新的目標佇列管理程式。

Multi 在 多平台上，佇列管理程式會在下一次啟動佇列管理程式時載入新的叢集工作量結束程式。

如果佇列管理程式定義不包含叢集工作量結束程式名稱，則不會呼叫叢集工作量結束程式。

在結束程式參數結構 MQWXP 中，會將各種資料傳遞至叢集工作量結束程式:

- 訊息定義結構 MQMD。
- 訊息長度參數。
- 訊息的副本或部分訊息。

在非 z/OS 平台上，如果您使用 CLWLMode=FAST，則每一個作業系統程序都會載入其自己的結束程式副本。 佇列管理程式的不同連線可能會導致呼叫結束程式的不同副本。 如果結束程式以預設安全模式 CLWLMode=SAFE 執行，則結束程式的單一副本會在其自己的個別處理程序中執行。

寫入叢集工作量結束程式

z/OS 如需撰寫 z/OS 的叢集工作量結束程式的相關資訊，請參閱 第 836 頁的『IBM MQ for z/OS 的叢集工作量結束程式程式設計』。

從 IBM MQ 9.1.0 開始，叢集工作量結束程式會在通道起始程式位址空間中執行，而不是在佇列管理程式位址空間中執行。 如果您有叢集工作量結束程式，則應該從佇列管理程式啟動型作業程序中移除 CSQXLIB DD 陳述式，並將包含叢集工作量結束程式的資料集新增至通道起始程式啟動型作業程序上的 CSQXLIB 連結。

Multi 對於 Multiplatforms，叢集工作量結束程式不得使用 MQI 呼叫。 在其他方面，撰寫及編譯叢集工作量結束程式的規則類似於適用於通道結束程式的規則。 遵循 第 784 頁的『在 AIX, Linux, and Windows 上撰寫結束程式及可安裝的服務』中的步驟，並使用範例程式 第 835 頁的『範例叢集工作量結束程式』 來協助撰寫及編譯您的結束程式。

如需通道結束程式的相關資訊，請參閱 第 808 頁的『寫入通道結束程式』。

配置叢集工作量結束程式

您可以在 ALTER QMGR 指令上指定叢集工作量結束程式屬性，來命名佇列管理程式定義中的叢集工作量結束程式。 例如:

```
ALTER QMGR CLWLEXIT(myexit)
```

相關參考

叢集工作量結束程式呼叫及資料結構

範例叢集工作量結束程式

IBM MQ 包括叢集工作量結束程式範例。 您可以複製範例，並使用它作為您自己的程式的基礎。

z/OS **IBM MQ for z/OS**

範例叢集工作量結束程式在 Assembler 及 C 中提供。 組譯器版本稱為 CSQ4BAF1，可在媒體庫 thlqual.SCSQASMS 中找到。 C 版本稱為 CSQ4BCF1，可在媒體庫 thlqual.SCSQC37S 中找到。 thlqual 是安裝中 IBM MQ 資料集的目標程式庫高階限定元。

Multi **IBM MQ for Multiplatforms**

叢集工作量結束程式範例是以 C 提供，稱為 amqswlm0.c。 它可以在下列位置找到:

表 *144: Multiplatforms* 的叢集工作量結束程式位置範例

平台	菲萊帕特
AIX AIX	*MQ_INSTALLATION_PATH*/samp
Windows Windows	*MQ_INSTALLATION_PATH* \Tools\c\Samples
IBM i IBM i	qmqm 程式庫

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

此範例結束程式會將所有訊息遞送至特定佇列管理程式，除非該佇列管理程式變成無法使用。 它會透過將訊息遞送至另一個佇列管理程式，對佇列管理程式失敗做出反應。

指出您要將訊息傳送至哪個佇列管理程式。 在佇列管理程式定義的 CLWLDATA 屬性中提供叢集接收端通道的名稱。 例如：

```
ALTER QMGR CLWLDATA(' my-cluster-name. my-queue-manager ')
```

如果要啟用結束程式，請在 CLLEXIT 屬性中提供其完整路徑和名稱：

Linux AIX 在 AIX and Linux 上：

```
ALTER QMGR CLWLEXIT(' path /amqswlm(clwlFunction)')
```

Windows 在 Windows 上：

```
ALTER QMGR CLWLEXIT(' path \amqswlm(clwlFunction)')
```

z/OS 在 z/OS 上：

```
ALTER QMGR CLWLEXIT(CSQ4BxF1)
```

其中 x 是 'A' 或 'C'，視您所使用版本的程式設計語言而定。

IBM i 在 IBM i 上，使用下列其中一個指令：

- 使用 MQSC 指令：

  ```
  ALTER QMGR CLWLEXIT('AMQSWLM       library       ')
  ```

  程式名稱和檔案庫名稱都會佔用 10 個字元，必要的話，會在右側填補空白。
- 使用 CL 指令：

  ```
  CHGMQM MQMNAME( qmgrname ) CLWLEXIT(' library /AMQSWLM')
  ```

現在， IBM MQ 不使用提供的工作量管理演算法，而是呼叫此結束程式，將所有訊息遞送至您選擇的佇列管理程式。

z/OS *IBM MQ for z/OS* 的叢集工作量結束程式程式設計

叢集工作量結束程式如同由 z/OS **LINK** 指令呼叫一樣。 結束程式受許多嚴格的程式設計規則所規範。 避免使用大部分涉及等待的 SVC 指令，或在工作量結束程式中使用 STAE 或 ESTAE。

呼叫叢集工作量結束程式，如同由下列項目中的 z/OS **LINK** 所呼叫：

- 未獲授權的問題程式狀態
- 主要位址空間控制模式
- 非跨記憶體模式
- 非存取登錄模式
- 31 位元定址模式
- 儲存體金鑰 8
- 程式金鑰遮罩 8

- TCB 鍵 8

將鏈結編輯模組放置在通道起始程式之已啟動作業程序的 CSQXLIB DD 陳述式所指定的資料集中。 載入模組的名稱會指定為佇列管理程式定義中的工作量結束程式名稱。

撰寫 IBM MQ for z/OS 的工作量結束程式時，適用下列規則:

- 您必須在組譯器或 C 中撰寫結束程式。 如果您使用 C，則它必須符合系統結束程式的 C 系統程式設計環境，如 *z/OS C/C++ Programming Guide*(SC09-4765) 中所述。
- 如果使用 MQXCLWLN 呼叫，請使用 *thlqual*.SCSQLOAD 中提供的 CSQMFCLW 進行鏈結編輯。
- 從 CSQXLIB DD 陳述式所定義的未獲授權程式庫載入結束程式。 如果 CSQXLIB 具有 DISP=SHR，則可以在佇列管理程式執行時更新結束程式，並在佇列管理程式啟動的下一個 MQCONN 執行緒中使用新版本。
- 結束程式必須重新進入，並且能夠在虛擬儲存體中的任何位置執行。
- 結束程式必須在返回該項目時重設環境。
- 結束程式必須釋放任何取得的儲存體，或確定後續的結束程式呼叫已釋放儲存體。
- 不容許 MQI 呼叫。
- 結束程式不得使用任何可能導致等待的系統服務，因為等待會嚴重降低佇列管理程式的效能。 因此，一般而言，請避免 SVC、PC 或 I/O。
- 除了附加的任何子作業之外，結束程式不得發出 ESTAE 或 SPIE。

註: 您可以在結束程式中執行的動作沒有絕對限制。 不過，大部分 SVC 都涉及等待，因此請避免等待，但下列指令除外:

- **GETMAIN** / **FREEMAIN**
- **LOAD** / **DELETE**

請勿使用 ESTAE 和 ESPIE，因為它們的錯誤處理可能會干擾 IBM MQ 所執行的錯誤處理。 IBM MQ 可能無法從錯誤回復，或者您的結束程式可能未收到所有錯誤資訊。

系統參數 EXITLIM 會限制結束程式可能執行的時間量。 EXITLIM 的預設值是 30 秒。 如果您看到回覆碼 MQRC_CLUSTER_EXIT_ERROR，則 2266 X'8DA' 您的結束程式可能在迴圈中。 如果您認為結束程式需要超過 30 秒才能完成，請增加 EXITLIM 的值。

建置程序化應用程式

您可以使用數種程序化語言之一來撰寫 IBM MQ 應用程式，並在數個不同的平台上執行該應用程式。

AIX 在 AIX 上建置程序化應用程式

AIX 出版品說明如何從您撰寫的程式建置可執行應用程式。

本主題說明建置要在 AIX 下執行的 IBM MQ for AIX 應用程式時必須執行的其他作業及標準作業的變更。 支援 C、C++ 及 COBOL。 如需準備 C++ 程式的相關資訊，請參閱 使用 C++。

使用 IBM MQ for AIX 來建立可執行應用程式時必須執行的作業，會隨著撰寫原始碼所用的程式設計語言而不同。 除了在原始碼中撰寫 MQI 呼叫的程式碼之外，您還必須新增適當的語言陳述式，以包括您所使用語言的 IBM MQ for AIX 併入檔。 讓您自己熟悉這些檔案的內容。 如需完整說明，請參閱 第 601 頁的『IBM MQ 資料定義檔』 。

當您執行執行緒伺服器或執行緒用戶端應用程式時，請設定環境變數 AIXTHREAD_SCOPE = S。

AIX 在 *AIX* 中準備 *C* 程式

本主題包含在 AIX 上準備 C 程式所需的鏈結程式庫的相關資訊。

在 *MQ_INSTALLATION_PATH*/samp/bin 目錄中提供經過前置編譯的 C 程式。 使用 ANSI 編譯器並執行下列指令。 如需程式設計 64 位元應用程式的相關資訊，請參閱 64 位元平台上的編碼標準。

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

若為 32 位元應用程式:

```
$ xlc_r -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -lmqm
```

其中 amqsput0 是範例程式。

若為 64 位元應用程式:

```
$ xlc_r -q64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64
-lmqm
```

其中 amqsput0 是範例程式。

V 9.3.5 若為使用 XLC 17 編譯器的 32 位元應用程式:

```
$ ibm-clang -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-lmqm
```

其中 amqsput0 是範例程式。

V 9.3.5 對於使用 XLC 17 編譯器的 64 位元應用程式:

```
$ ibm-clang -m64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/
lib64 -lmqm
```

其中 amqsput0 是範例程式。

如果您使用適用於 C++ 程式的 VisualAge C/C++ 編譯器，則必須包括 -q namemangling=v5 選項，以在鏈結程式庫時解析所有 IBM MQ 符號。

如果您想要在僅安裝 IBM MQ MQI client for AIX 的機器上使用程式，請重新編譯程式以改為將它們與用戶端程式庫 (-lmqic) 鏈結。

鏈結檔案庫

您需要下列程式庫:

- 將您的程式鏈結至 IBM MQ 所提供的適當檔案庫。

  在非執行緒環境中，鏈結至下列其中一個檔案庫:

程式庫檔案	程式/結束程式類型
libmqm.a	適用於 C 的伺服器
libmqic.a & libmqm.a	適用於 C 的用戶端

  在執行緒化環境中，鏈結至下列其中一個檔案庫:

程式庫檔案	程式/結束程式類型
libmqm_r.a	適用於 C 的伺服器
libmqic_r.a & libmqm_r.a	適用於 C 的用戶端

  例如，若要從單一編譯單元建置簡式執行緒 IBM MQ 應用程式，請執行下列指令。

  若為 32 位元應用程式:

```
$ xlc_r -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -lmqm_r
```

  其中 amqsput0 是範例程式。

  若為 64 位元應用程式:

```
$ xlc_r -q64 -o amqsputc_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -lmqm_r
```

其中 amqsput0 是範例程式。

V 9.3.5 若為使用 XLC 17 編譯器的 32 位元應用程式:

```
$ ibm-clang_r -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib -lmqm_r
```

其中 amqsput0 是範例程式。

V 9.3.5 對於使用 XLC 17 編譯器的 64 位元應用程式:

```
$ ibm-clang_r -m64 -o amqsput_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib64 -lmqm_r
```

其中 amqsput0 是範例程式。

如果您想要在僅安裝 IBM MQ MQI client for AIX 的機器上使用程式，請重新編譯程式以改為將它們與用戶端程式庫 (-lmqic) 鏈結。

註:

1. 您無法鏈結至多個檔案庫。 亦即，您無法同時鏈結至執行緒及非執行緒檔案庫。
2. 如果您要撰寫可安裝的服務 (如需進一步資訊，請參閱 管理 IBM MQ)，您需要鏈結至非執行緒應用程式中的 libmqmzf.a 程式庫，以及鏈結至執行緒應用程式中的 libmqmzf_r.a 程式庫。
3. 如果您要由符合 XA 標準的交易管理程式 (例如 IBM TXSeries、Encina 或 BEA Tuxedo) 產生外部協調的應用程式，則需要鏈結至 libmqmxa.a (或 libmqmxa64.a，如果您的交易管理程式將 'long' 類型視為 64 位元) 及非執行緒應用程式中的 libmqz.a 程式庫，以及 libmqmxa_r.a (或 libmqmxa64_r.a)。 及 libmqz_r.a 檔案庫。
4. 您需要將授信應用程式鏈結至含執行緒作業的 IBM MQ 程式庫。 不過，一次只能連接 IBM MQ for AIX or Linux 系統上授信應用程式中的一個執行緒。
5. 您必須先鏈結 IBM MQ 程式庫，然後再鏈結任何其他產品程式庫。

AIX 在 *AIX* 中準備 *COBOL* 程式

在 AIX 中使用 IBM COBOL Set 和 Micro Focus COBOL 來準備 COBOL 程式時，請使用此資訊。

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

- 32 位元 COBOL 記錄定義檔安裝在下列目錄中:

  ```
  MQ_INSTALLATION_PATH/inc/cobcpy32
  ```

  和符號鏈結建立於:

  ```
  MQ_INSTALLATION_PATH/inc
  ```

- 64 位元 COBOL 記錄定義檔安裝在下列目錄中:

  ```
  MQ_INSTALLATION_PATH/inc/cobcpy64
  ```

在下列範例中，將 **COBCPY** 環境變數設為:

```
MQ_INSTALLATION_PATH/inc/cobcpy32
```

(適用於 32 位元應用程式)，以及:

```
MQ_INSTALLATION_PATH/inc/cobcpy64
```

適用於 64 位元應用程式。

您需要將程式與下列其中一個檔案庫檔案鏈結:

程式庫檔案	程式/結束程式類型
libmqmcb.a	COBOL 的伺服器 (無執行緒應用程式)
libmqmcb_r.a	COBOL (執行緒應用程式) 的伺服器
libmqicb.a	COBOL (無執行緒應用程式) 的用戶端
libmqicb_r.a	COBOL 用戶端 (含執行緒作業的應用程式)

視程式而定，您可以使用 IBM COBOL Set 編譯器或 Micro Focus COBOL 編譯器:

- 以 amqm 開頭的程式適用於 Micro Focus COBOL 編譯器，以及
- 以 amq0 開頭的程式適用於任一編譯器。

使用 IBM COBOL Set for AIX 來準備 COBOL 程式

隨附於 IBM MQ 的 COBOL 程式範例。 若要編譯這類程式，請從下列清單中輸入適當的指令:

32 位元非執行緒伺服器應用程式

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb -qLIB \
-ICOBCPY_VALUE
```

32 位元非執行緒用戶端應用程式

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb -qLIB \
-ICOBCPY_VALUE
```

32 位元執行緒伺服器應用程式

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -ICOBCPY_VALUE
```

32 位元執行緒用戶端應用程式

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -ICOBCPY_VALUE
```

64 位元非執行緒伺服器應用程式

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqmcb \
-qLIB -ICOBCPY_VALUE
```

64 位元非執行緒用戶端應用程式

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqicb \
-qLIB -ICOBCPY_VALUE
```

64 位元執行緒伺服器應用程式

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -ICOBCPY_VALUE
```

64 位元執行緒用戶端應用程式

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -ICOBCPY_VALUE
```

使用 Micro Focus COBOL 準備 COBOL 程式

在編譯程式之前設定環境變數，如下所示:

```
export COBCPY=COBCPY_VALUE
export LIBPATH=MQ_INSTALLATION_PATH/lib:$LIBPATH
```

若要使用 Micro Focus COBOL 編譯 32 位元 COBOL 程式，請輸入:

- COBOL 的伺服器

```
$ cob32 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb
```

- COBOL 的用戶端

```
$ cob32 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb
```

- COBOL 的執行緒伺服器

```
$ cob32 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r
```

- 適用於 COBOL 的執行緒用戶端

```
$ cob32 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r
```

如果要使用 Micro Focus COBOL 來編譯 64 位元 COBOL 程式，請輸入:

- COBOL 的伺服器

```
$ cob64 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb
```

- COBOL 的用戶端

```
$ cob64 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb
```

- COBOL 的執行緒伺服器

```
$ cob64 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r
```

- 適用於 COBOL 的執行緒用戶端

```
$ cob64 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r
```

其中 amqminqx 是範例程式

如需您需要設定之環境變數的說明，請參閱 Micro Focus COBOL 說明文件。

AIX 在 *AIX* 中準備 *CICS* 應用程式

在 AIX 中準備 CICS 程式時，請使用此資訊。

使用 *XA* 交換器 模組來鏈結 CICS 與 IBM MQ。 如需 XA 交換器結構的相關資訊，請參閱 XA 交換器結構。

提供的範例原始碼檔案可讓您開發其他交易訊息的 XA 參數。 第 842 頁的表 145 中列出所提供的交換器載入模組名稱。

表 *145: AIX* 上 *CICS* 應用程式的基本程式碼 *:XA* 起始設定常式		
說明	**C (來源)**	**C (exec)-新增至您的 XAD.Stanza**
XA 起始設定常式	amqzscix.c	*amqzsc* - CICS for AIX

使用產品隨附的 IBM MQ 交換器載入檔案 *amqzsc* 預先建置版本。

一律將 C 交易與安全執行緒 IBM MQ 程式庫 *libmqm_r.a* 鏈結。， 以及與 COBOL 程式庫 *libmqmcb_r.a.*的 COBOL 交易。

You can find more information about supporting CICS transactions in the 管理 IBM MQ IBM MQ System Administration Guide.

AIX *TXSeriesCICS* 支援

AIX 上的 IBM MQ 支援使用 XA 介面的 TXSeries CICS 。 請確定 CICS 應用程式已鏈結至 IBM MQ 程式庫的執行緒版本。

您可以使用 IBM COBOL Set for AIX 或 Micro Focus COBOL 來執行 CICS 程式。 下列各節說明在 IBM COBOL Set for AIX 和 Micro Focus COBOL 上執行 CICS 程式之間的差異。

撰寫載入至 C 或 COBOL 中相同 CICS 區域的 IBM MQ 程式。 您無法將 C 和 COBOL MQI 呼叫組合在相同的 CICS 區域中。 使用第二種語言的大部分 MQI 呼叫會失敗，原因碼為 MQRC_HOBJ_ERROR。

使用 IBM COBOL Set for AIX 來準備 CICS COBOL 程式

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

若要使用 IBM COBOL ，請遵循下列步驟:

1. 匯出下列環境變數:

```
export LDFLAGS="-qLIB -bI:/usr/lpp/cics/lib/cicsprIBMCOB.exp \
                -I MQ_INSTALLATION_PATH/inc -I/usr/lpp/cics/include \
                -e _iwz_cobol_main  \
```

   其中 LIB 是編譯器指引。

2. 透過鍵入下列指令來轉換、編譯及鏈結程式:

```
cicstcl -l IBMCOB yourprog.ccp
```

使用 Micro Focus COBOL 來準備 CICS COBOL 程式

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

若要使用 Micro Focus COBOL ，請遵循下列步驟:

1. 使用下列指令，將 IBM MQ COBOL 執行時期程式庫模組新增至執行時期程式庫:

```
cicsmkcobol -L/usr/lib/dce -L MQ_INSTALLATION_PATH/lib \
            MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe_r
```

   註: 使用 cicsmkcobol， IBM MQ 不容許您從 COBOL 應用程式以 C 程式設計語言進行 MQI 呼叫。

如果您現有的應用程式有任何這類呼叫，建議您將這些函數從 COBOL 應用程式移至您自己的程式庫，例如 `myMQ.so`。 移動函數之後，在建置 CICS 的 COBOL 應用程式時，請不要包含 IBM MQ 程式庫 `libmqmcbrt.o` 。

此外，如果您的 COBOL 應用程式未進行任何 COBOL MQI 呼叫，請勿將 `libmqmz_r` 與 `cicsmkcobol` 鏈結。

這會建立 Micro Focus COBOL 語言方法檔案，並讓 CICS 執行時期 COBOL 程式庫呼叫 IBM MQ for AIX or Linux 系統。

註: 僅當您安裝下列其中一個產品時，才執行 `cicsmkcobol` :

- Micro Focus COBOL 的新版本或版次
- CICS for AIX 的新版本或版次
- 任何受支援資料庫產品的新版本或版次 (僅適用於 COBOL 交易)
- IBM MQ 的新版本或版次

2. 匯出下列環境變數:

```
COBCPY= MQ_INSTALLATION_PATH/inc export COBCPY
```

3. 透過鍵入下列指令來轉換、編譯及鏈結程式:

```
cicstcl -l COBOL -e yourprog.ccp
```

準備 CICS C 程式

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

使用標準 CICS 機能來建置 CICS C 程式:

1. 匯出下列環境變數中的 **一個** :
   - LDFLAGS = "-L/ *MQ_INSTALLATION_PATH* lib -lmqm_r" export LDFLAGS
   - USERLIB = "-L *MQ_INSTALLATION_PATH* lib -lmqm_r" export USERLIB
2. 透過鍵入下列指令來轉換、編譯及鏈結程式:

```
cicstcl -l C amqscic0.ccs
```

CICS C 範例交易

AIX IBM MQ 交易的範例 C 來源由 AMQSCIC0.CCS。 交易會從傳輸佇列 SYSTEM.SAMPLE.CICS 讀取訊息。WORKQUEUE 在預設佇列管理程式上，並使用訊息傳輸標頭中包含的佇列名稱將它們放入本端佇列。 任何失敗都會傳送至佇列 SYSTEM.SAMPLE.CICS。DLQ。 使用範例 MQSC Script AMQSCIC0.TST ，用於建立這些佇列及範例輸入佇列。

IBM i 在 IBM i 上建置程序化應用程式

IBM i 出版品說明如何從您撰寫的程式建置可執行應用程式，以在 iSeries 或 System i 系統上與 IBM i 搭配執行。

本主題說明建置要在 IBM i 系統上執行的 IBM MQ for IBM i 程序化應用程式時必須執行的其他作業及標準作業的變更。 支援 COBOL、C、C++、 Java 及 RPG 程式設計語言。 如需準備 C++ 程式的相關資訊，請參閱 使用 C++。 如需準備 Java 程式的相關資訊，請參閱 使用 IBM MQ classes for Java。

您建立可執行的 IBM MQ for IBM i 應用程式時必須執行的作業，取決於撰寫原始碼所用的程式設計語言。 除了在原始碼中撰寫 MQI 呼叫的程式碼之外，您還必須新增適當的語言陳述式，以包括所使用語言的 IBM MQ for IBM i 資料定義檔。 讓您自己熟悉這些檔案的內容。 如需完整說明，請參閱 第 601 頁的『IBM MQ 資料定義檔』 。

IBM i 在 *IBM i* 中準備 *C* 程式

IBM MQ for IBM i 支援大小最多 100 MB 的訊息。 以 ILE C 撰寫的應用程式 (支援大於 16 MB 的 IBM MQ 訊息) 需要使用 Teraspace 編譯器選項為這些訊息配置足夠的記憶體。

如需 C 編譯器選項的相關資訊，請參閱 *WebSphere Development Studio ILE C/C++ Programmer 's Guide*。

若要編譯 C 模組，您可以使用 IBM i 指令 **CRTCMOD**。 編譯時，請確定包含併入檔 (QMQM) 的程式庫在程式庫清單中。

然後，您必須使用 **CRTPGM** 指令，將編譯器的輸出與服務程式連結。

表 *146:* 非執行緒及執行緒環境中 *CRTPGM* 的範例

環境類型	指令	程式/結束程式類型

非執行緒環境	`CRTPGM PGM( pgmname )`
	`MODULE( pgmname )`
	`BNDSRVPGM(QMQM/LIBMQM)`
		適用於 C 的伺服器或用戶端

執行緒環境	`CRTPGM PGM( pgmname )`
	`MODULE( pgmname )`
	`BNDSRVPGM(QMQM/LIBMQM_R)`
		適用於 C 的伺服器或用戶端

其中 *pgmname* 是程式的名稱。

第 844 頁的表 147 列出在非執行緒環境及執行緒環境中的 IBM i 上準備 C 程式時所需的檔案庫。

表 *147:* 非執行緒及執行緒環境所需的檔案庫

環境類型	程式庫檔案	程式/結束程式類型
非執行緒環境	LIBMQM	適用於 C 的伺服器
	LIBMQIC 與 LIBMQM	適用於 C 的用戶端
執行緒環境	LIBMQM_R	適用於 C 的伺服器
	LIBMQIC_R & LIBMQM_R	適用於 C 的用戶端

IBM i 在 *IBM i* 中準備 *COBOL* 程式

瞭解在 IBM i 中準備 COBOL 程式，以及從 COBOL 程式內存取 MQI 的方法。

關於這項作業

若要從 COBOL 程式內存取 MQI， IBM MQ for IBM i 會提供服務程式所提供的連結程序化呼叫介面。 這可讓您存取 IBM MQ for IBM i 中的所有 MQI 功能，並支援執行緒化應用程式。 此介面只能與 ILE COBOL 編譯器一起使用。

標準 COBOL CALL 語法是用來存取 MQI 函數。

包含與 MQI 搭配使用之具名常數及結構定義的 COBOL 副本檔包含在來源實體檔 QMQM/QCBLLESRC 中。

COBOL 複製檔案使用單引號字元 (') 作為字串定界字元。 IBM i COBOL 編譯器會假設定界字元是引號 (")。 若要防止編譯器產生警告訊息，請在指令 **CRTCBLPGM**、 **CRTBNDCBL** 或 **CRTCBLMOD** 上指定 OPTION (*APOST)。

若要讓編譯器接受單引號字元 (') 作為 COBOL 複製檔案中的字串定界字元，請使用編譯器選項 \APOST。

註: IBM MQ 9.0 或更新版本中未提供動態呼叫介面。

若要使用連結程序呼叫介面，請完成下列步驟。

程序

1. 使用指定參數的 **CRTCBLMOD** 編譯器來建立模組:

```
LINKLIT(*PRC)
```

2. 使用 **CRTPGM** 指令來建立程式物件，並指定適當的參數:

   若為非執行緒應用程式:

```
BNDSRVPGM(QMQM/AMQ0STUB)        Server for COBOL for non-threaded applications
BNDSRVPGM(QMQM/AMQCSTUB)        Client for COBOL for non-threaded applications
```

   若為執行緒應用程式:

```
BNDSRVPGM(QMQM/AMQ0STUB_R)        Server for COBOL for threaded applications
BNDSRVPGM(QMQM/AMQCSTUB_R)        Client for COBOL for threaded applications
```

   註: 除了使用 V4R4 ILE COBOL 編譯器建立且在 PROCESS 陳述式中包含 THREAD (SERIALIZE) 選項的程式之外， COBOL 程式不得使用含執行緒作業的 IBM MQ 程式庫。 即使以這種方式使 COBOL 程式成為安全執行緒，在設計應用程式時也要小心，因為 THREAD (SERIALIZE) 會在模組層次強制序列化 COBOL 程序，且可能會影響整體效能。

   如需進一步資訊，請參閱 *WebSphere Development Studio: ILE COBOL Programmer 's Guide* 及 *WebSphere Development Studio: ILE COBOL Reference* 。

   如需編譯 CICS 應用程式的相關資訊，請參閱 *CICS for IBM i Application Programming Guide*(SC41-5454)。

IBM i 在 *IBM i* 中準備 *CICS* 程式

瞭解在 IBM i 中準備 CICS 程式時所需的步驟。

若要建立包括 EXEC CICS 陳述式及 MQI 呼叫的程式，請執行下列步驟:

1. 必要的話，請使用 CRTCICSMAP 指令來準備對映。
2. 將 EXEC CICS 指令轉換成原生語言陳述式。 對 C 程式使用 CRTCICSC 指令。 將 CRTCICSCBL 指令用於 COBOL 程式。

   在 CRTCICSC 或 CRTCICSCBL 指令中併入 CICSOPT(*NOGEN) 。 這會中止處理，讓您能夠併入適當的 CICS 和 IBM MQ 服務程式。 依預設，此指令會將程式碼放入 QTEMP/QACYCICS。
3. 使用 CRTCMOD 指令 (適用於 C 程式) 或 CRTCBLMOD 指令 (適用於 COBOL 程式) 來編譯原始碼。
4. 使用 CRTPGM ，將編譯的程式碼與適當的 CICS 及 IBM MQ 服務程式鏈結。 這會建立可執行程式。

這類程式碼的範例如下 (它編譯隨附的 CICS 範例程式):

```
CRTCICSC  OBJ(QTEMP/AMQSCIC0) SRCFILE(/MQSAMP/QCSRC) +
     SRCMBR(AMQSCIC0) OUTPUT(*PRINT) +
     CICSOPT(*SOURCE *NOGEN)
CRTCMOD   MODULE(MQTEST/AMQSCIC0) +
     SRCFILE(QTEMP/QACYCICS) OUTPUT(*PRINT)
CRTPGM    PGM(MQTEST/AMQSCIC0) MODULE(MQTEST/AMQSCIC0) +
     BNDSRVPGM(QMQM/LIBMQIC QCICS/AEGEIPGM)
```

IBM i 在 *IBM i* 中準備 *RPG* 程式

如果您是使用 IBM MQ for IBM i，則可以在 RPG 中撰寫應用程式。

如需相關資訊，請參閱 第 887 頁的『在 RPG 中撰寫 IBM MQ 程式的程式碼 (僅限 IBM i)』，並參閱 IBM i Application Programming Reference (ILE/RPG)。

IBM i *IBM i 的 SQL 程式設計考量*

瞭解使用 SQL 在 IBM i 上建置應用程式時所需的步驟。

如果您的程式包含 EXEC SQL 陳述式及 MQI 呼叫，請執行下列步驟:

1. 將 EXEC SQL 指令轉換成原生語言陳述式。 對 C 程式使用 CRTSQLCI 指令。 對 COBOL 程式使用 CRTSQLCBLI 指令。

   在 CRTSQLCI 或 CRTSQLCBLI 指令中併入 OPTION(*NOGEN) 。 這會中止處理，讓您能夠併入適當的 IBM MQ 服務程式。 依預設，此指令會將程式碼放入 QTEMP/QSQLTEMP。
2. 使用 CRTCMOD 指令 (適用於 C 程式) 或 CRTCBLMOD 指令 (適用於 COBOL 程式) 來編譯原始碼。
3. 使用 CRTPGM 來鏈結已編譯的程式碼與適當的 IBM MQ 服務程式。 這會建立可執行程式。

這類程式碼的範例如下 (它會編譯檔案庫 SQLUSER 中的程式 SQLTEST):

```
CRTSQLCI  OBJ(MQTEST/SQLTEST) SRCFILE(SQLUSER/QCSRC) +
     SRCMBR(SQLTEST) OUTPUT(*PRINT) OPTION(*NOGEN)
CRTCMOD   MODULE(MQTEST/SQLTEST) +
     SRCFILE(QTEMP/QSQLTEMP) OUTPUT(*PRINT)
CRTPGM    PGM(MQTEST/SQLTEST) +
     BNDSRVPGM(QMQM/LIBMQIC)
```

Linux 在 Linux 上建置程序化應用程式

此資訊說明建置 IBM MQ 時必須執行的其他作業，以及標準作業的變更， Linux 應用程式才能執行。

支援 C 和 C++。 如需準備 C++ 程式的相關資訊，請參閱 使用 C++。

Linux 在 *Linux* 中準備 *C* 程式

前置編譯的 C 程式在 *MQ_INSTALLATION_PATH*/samp/bin 目錄中提供。 若要從原始碼建置範例，請使用 gcc 編譯器。

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

在正常環境中工作。 如需 64 位元應用程式程式設計的進一步相關資訊，請參閱 64 位元平台上的編碼標準。

鏈結檔案庫

下表列出在 Linux 上準備 C 程式時所需的程式庫。

- 您需要將程式與 IBM MQ 提供的適當檔案庫鏈結。

  在非執行緒環境中，只鏈結至下列其中一個檔案庫:

程式庫檔案	程式/結束程式類型
libmqm.so	適用於 C 的伺服器
libmqic.so & libmqm.so	適用於 C 的用戶端

  在執行緒作業環境中，只鏈結至下列其中一個檔案庫:

程式庫檔案	程式/結束程式類型
libmqm_r.so	適用於 C 的伺服器
libmqic_r.so & libmqm_r.so	適用於 C 的用戶端

  註:

  1. 您無法鏈結至多個檔案庫。 亦即，您無法同時鏈結至執行緒及非執行緒檔案庫。

2. 如果您要撰寫可安裝的服務 (如需進一步資訊，請參閱 管理 IBM MQ)，則需要鏈結至 `libmqmzf.so` 程式庫。
3. 如果您要由符合 XA 標準的交易管理程式 (例如 IBM TXSeries Encina 或 BEA Tuxedo) 產生外部協調的應用程式，則需要鏈結至 `libmqmxa.so` (或 `libmqmxa64.so`，如果您的交易管理程式將 'long' 類型視為 64 位元) 及非執行緒應用程式中的 `libmqz.so` 程式庫，以及 `libmqmxa_r.so` (或 `libmqmxa64_r.so`)。 及 `libmqz_r.so` 檔案庫。
4. 您必須先鏈結 IBM MQ 程式庫，然後再鏈結任何其他產品程式庫。

Linux 建置 *31* 位元應用程式

本主題包含在各種環境中用來建置 31 位元程式的指令範例。

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

C 用戶端應用程式， 31 位元，無執行緒

```
gcc -m31 -o famqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic
```

C 用戶端應用程式， 31 位元，執行緒

```
gcc -m31 -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

C 伺服器應用程式， 31 位元，無執行緒

```
gcc -m31 -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

C 伺服器應用程式， 31 位元，執行緒

```
gcc -m31 -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

C++ 用戶端應用程式， 31 位元，非執行緒

```
g++ -m31 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl
-limqb23gl -lmqic
```

C++ 用戶端應用程式， 31 位元，執行緒

```
g++ -m31 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r
-limqb23gl_r -lmqic_r -lpthread
```

C++ 伺服器應用程式， 31 位元，無執行緒

```
g++ -m31 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl
-limqb23gl -lmqm
```

C++ 伺服器應用程式， 31 位元，執行緒

```
g++ -m31 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r
-limqb23gl_r -lmqm_r -lpthread
```

C 用戶端結束程式， 31 位元，無執行緒

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic
```

C 用戶端結束程式， 31 位元，執行緒

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/cliexit_32_r cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

C 伺服器結束程式， 31 位元，無執行緒

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/srvexit_32 srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm
```

C 伺服器結束程式， 31 位元，執行緒

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/srvexit_32_r srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Linux 建置 *32* 位元應用程式

本主題包含在各種環境中用來建置 32 位元程式的指令範例。

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

C 用戶端應用程式， 32 位元，無執行緒

```
gcc -m32 -o amqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic
```

C 用戶端應用程式， 32 位元，執行緒

```
gcc -m32 -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

C 伺服器應用程式， 32 位元，無執行緒

```
gcc -m32 -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

C 伺服器應用程式， 32 位元，執行緒

```
gcc -m32 -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

C++ 用戶端應用程式， 32 位元，無執行緒

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

C++ 用戶端應用程式， 32 位元，執行緒

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

C++ 伺服器應用程式， 32 位元，無執行緒

```
g++ -m32 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl -limqb23gl -lmqm
```

C++ 伺服器應用程式， 32 位元，執行緒

```
g++ -m32 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

C 用戶端結束程式， 32 位元，無執行緒

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic
```

C 用戶端結束程式， 32 位元，執行緒

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/cliexit_32_r cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

C 伺服器結束程式， 32 位元，無執行緒

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/srvexit_32 srvexit.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm
```

C 伺服器結束程式， 32 位元，執行緒

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/srvexit_32_r srvexit.c
I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Linux 建置 *64* 位元應用程式

本主題包含在各種環境中用來建置 64 位元程式的指令範例。

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

C 用戶端應用程式， 64 位元，非執行緒

```
gcc -m64 -o amqsputc_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqic
```

C 用戶端應用程式， 64 位元，執行緒

```
gcc -m64 -o amqsputc_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqic_r
-lpthread
```

C 伺服器應用程式， 64 位元，無執行緒

```
gcc -m64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm
```

C 伺服器應用程式， 64 位元，執行緒

```
gcc -m64 -o amqsput_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm_r
-lpthread
```

C++ 用戶端應用程式， 64 位元，無執行緒

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

C++ 用戶端應用程式， 64 位元，執行緒

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

C++ 伺服器應用程式， 64 位元，無執行緒

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

C++ 伺服器應用程式， 64 位元，執行緒

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

C 用戶端結束程式， 64 位元，無執行緒

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/cliexit_64 cliexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqic
```

C 用戶端結束程式， 64 位元，執行緒

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/cliexit_64_r cliexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqic_r -lpthread
```

C 伺服器結束程式， 64 位元，無執行緒

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/srvexit_64 srvexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqm
```

C 伺服器結束程式， 64 位元，執行緒

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/srvexit_64_r srvexit.c
```

```
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqm_r -lpthread
```

Linux 在 *Linux* 中準備 *COBOL* 程式

瞭解在 Linux 中準備 COBOL 程式，以及使用 IBM COBOL for Linux on x86 和 Micro Focus COBOL 來準備 COBOL 程式。

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

1. 32 位元 COBOL 記錄定義檔安裝在下列目錄中:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   和符號鏈結建立於:

   ```
   MQ_INSTALLATION_PATH/inc
   ```

2. 在 64 位元平台上， 64 位元 COBOL 記錄定義檔安裝在下列目錄中:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

3. 在下列範例中，將 COBCPY 設為:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   適用於 32 位元應用程式，以及:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

   適用於 64 位元應用程式。

您需要將程式與下列其中一項鏈結:

程式庫檔案	程式/結束程式類型
libmqmcb.so	COBOL 的伺服器
libmqicb.so	COBOL 的用戶端
libmqmcb_r.so	COBOL (執行緒應用程式) 的伺服器
libmqicb_r.so	COBOL 用戶端 (含執行緒作業的應用程式)

在 x86 上使用 IBM COBOL for Linux 來準備 COBOL 程式

隨附於 IBM MQ 的 COBOL 程式範例。 若要編譯這類程式，請從下列清單中輸入適當的指令:

32 位元無執行緒伺服器應用程式

```
$ cob2 -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-L MQ_INSTALLATION_PATH/lib -lmqmcb -ICOBCPY_VALUE
```

32 位元非執行緒用戶端應用程式

```
$ cob2 -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-L MQ_INSTALLATION_PATH/lib -lmqicb -ICOBCPY_VALUE
```

32 位元執行緒伺服器應用程式

```
$ cob2_r -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-qTHREAD -L MQ_INSTALLATION_PATH/lib -lmqmcb_r -ICOBCPY_VALUE
```

32 位元執行緒用戶端應用程式

```
$ cob2_r -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-qTHREAD -L MQ_INSTALLATION_PATH/lib -lmqicb_r -ICOBCPY_VALUE
```

使用 Micro Focus COBOL 準備 COBOL 程式

在編譯程式之前設定環境變數，如下所示:

```
export COBCPY=COBCPY_VALUE
export LIB= MQ_INSTALLATION_PATH lib:$LIB
```

如果要使用 Micro Focus COBOL 來編譯支援的 32 位元 COBOL 程式，請輸入:

```
$ cob32 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb Server for COBOL
$ cob32 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb Client for COBOL
$ cob32 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r Threaded Server for COBOL
$ cob32 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r Threaded Client for COBOL
```

如果要使用 Micro Focus COBOL 來編譯 64 位元 COBOL 程式，請輸入:

```
$ cob64 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb Server for COBOL
$ cob64 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb Client for COBOL
$ cob64 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r Threaded Server for COBOL
$ cob64 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r Threaded Client for COBOL
```

其中 amqsput 是範例程式

如需所需環境變數的說明，請參閱 Micro Focus COBOL 說明文件。

Windows 在 Windows 上建置程序化應用程式

Windows 系統出版品說明如何從您撰寫的程式建置可執行應用程式。

本主題說明建置要在 Windows 系統下執行的 IBM MQ for Windows 應用程式時必須執行的其他作業及標準作業的變更。 支援 C、C++、COBOL 及 Visual Basic 程式設計語言。 如需準備 C++ 程式的相關資訊，請參閱 使用 C++。

使用 IBM MQ for Windows 來建立可執行應用程式時必須執行的作業，會隨著撰寫原始碼所用的程式設計語言而不同。 除了在原始碼中撰寫 MQI 呼叫的程式碼之外，您還必須新增適當的語言陳述式，以包括您所使用語言的 IBM MQ for Windows 併入檔。 讓您自己熟悉這些檔案的內容。 如需完整說明，請參閱 第 601 頁的『IBM MQ 資料定義檔』 。

Windows 在 *Windows 64* 位元應用程式

在 IBM MQ for Windows 上支援 32 位元及 64 位元應用程式。 IBM MQ 執行檔及檔案庫檔案以 32 位元及 64 位元格式提供，視您使用的應用程式而定，使用適當的版本。

執行檔和程式庫

下列位置提供 32 位元及 64 位元版本的 IBM MQ 程式庫:

表 *148: IBM MQ* 程式庫的位置

程式庫版本	**包含檔案庫檔案的目錄**
32 位元	*MQ_INSTALLATION_PATH* \Tools\Lib
64 位元	*MQ_INSTALLATION_PATH* \Tools\Lib64

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

移轉之後， 32 位元應用程式會繼續正常運作。 32 位元檔案存在於與舊版產品相同的目錄中。

如果您想要建立 64 位元版本，則必須確保您的環境已配置為使用 *MQ_INSTALLATION_PATH* \Tools\Lib64 中的程式庫檔案。 請確定 LIB 環境變數未設定為查看包含 32 位元檔案庫的資料夾。

Windows 在 *Windows* 中準備 *C* 程式

在一般 Windows 環境中工作; IBM MQ for Windows 不需要任何特殊項目。

如需 64 位元應用程式程式設計的進一步相關資訊，請參閱 64 位元平台上的編碼標準。

- 將您的程式與 IBM MQ 提供的適當程式庫鏈結:

程式庫檔案	程式/結束程式類型
MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib	適用於 32 位元 C 的伺服器
MQ_INSTALLATION_PATH\Tools\Lib\mqic.lib	適用於 32 位元 C 的用戶端
MQ_INSTALLATION_PATH\Tools\Lib\mqicxa.lib	具有交易協調的 32 位元 C 用戶端
MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib	64 位元 C 的伺服器
MQ_INSTALLATION_PATH\Tools\Lib64\mqic.lib	適用於 64 位元 C 的用戶端
MQ_INSTALLATION_PATH\Tools\Lib64\mqicxa.lib	具有交易協調的 64 位元 C 用戶端

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

下列指令提供編譯範例程式 amqsget0 (使用 Microsoft Visual C++ 編譯器) 的範例。

若為 32 位元應用程式:

```
cl -MD amqsget0.c -Feamqsget.exe MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib
```

若為 64 位元應用程式:

```
cl -MD amqsget0.c -Feamqsget.exe MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib
```

註:

- 如果您要撰寫可安裝的服務 (如需進一步資訊，請參閱 管理 IBM MQ)，則需要鏈結至 mqmzf.lib 程式庫。
- 如果您要由符合 XA 標準的交易管理程式 (例如 IBM TXSeries Encina 或 BEA Tuxedo) 產生外部協調應用程式，則需要鏈結至 mqmxa.lib 或 mqmxa.lib 程式庫。
- 如果您要撰寫 CICS 結束程式，請鏈結至 mqmcics4.lib 程式庫。
- 您必須先鏈結 IBM MQ 程式庫，然後再鏈結任何其他產品程式庫。

- DLL 必須在您指定的路徑 (PATH) 中。
- 如果您盡可能使用小寫字元，則可以從 IBM MQ for Windows 移至 IBM MQ for AIX or Linux 系統，其中需要使用小寫字元。

準備 CICS 及 Transaction Server 程式

CICS IBM MQ 交易的範例 C 來源由 AMQSCIC0.CCS。 您可以使用標準 CICS 機能來建置它。 例如，若為 TXSeries for Windows 2000:

1. 設定環境變數 (在一行上輸入下列程式碼):

```
set CICS_IBMC_FLAGS=-I MQ_INSTALLATION_PATH\Tools\C\Include;
%CICS_IBMC_FLAGS%
```

2. 設定 USERLIB 環境變數:

```
set USERLIB=MQM.LIB;%USERLIB%
```

3. 轉換、編譯及鏈結範例程式:

```
cicstcl -l IBMC amqscic0.ccs
```

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

如需相關說明，請參閱 *Transaction Server for Windows NT Application Programming Guide (CICS)。 V4*。

您可以在 管理 IBM MQ 中找到支援 CICS 交易的相關資訊。

Windows 在 *Windows* 中準備 *COBOL* 程式

使用此資訊來學習在 Windows 中準備 COBOL 程式，以及準備 CICS 和交易伺服器程式。

1. 32 位元 COBOL 記錄定義檔安裝在下列目錄中: *MQ_INSTALLATION_PATH* \Tools\cobol\CopyBook。
2. 64 位元 COBOL 記錄定義檔安裝在下列目錄中: *MQ_INSTALLATION_PATH* \Tools\cobol\CopyBook64
3. 在下列範例中，將 CopyBook 設為:

```
CopyBook
```

   適用於 32 位元應用程式，以及:

```
CopyBook64
```

   適用於 64 位元應用程式。

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

若要在 Windows 系統上準備 COBOL 程式，請將您的程式鏈結至 IBM MQ 所提供的下列其中一個程式庫:

程式庫檔案	程式或結束程式類型
MQ_INSTALLATION_PATH \Tools\Lib\mqmcb	Micro Focus COBOL 的 32 位元伺服器
MQ_INSTALLATION_PATH \Tools\Lib\mqiccb	Micro Focus COBOL 的 32 位元用戶端
MQ_INSTALLATION_PATH \Tools\Lib64\mqmcb	Micro Focus COBOL 的 64 位元伺服器
MQ_INSTALLATION_PATH \Tools\Lib64\mqiccb	Micro Focus COBOL 的 64 位元用戶端

當您在 MQI 用戶端環境中執行程式時，請確定 DOSCALLS 程式庫出現在任何 COBOL 或 IBM MQ 程式庫之前。

使用 Micro Focus COBOL 準備 COBOL 程式

使用 mqmcb.lib 或 mqiccb.lib，而非 mqmcbb 和 mqiccbb 程式庫，重新鏈結任何現有的 32 位元 IBM MQ Micro Focus COBOL 程式。

例如，使用 Micro Focus COBOL 來編譯範例程式 amq0put0:

1. 將 COBCPY 環境變數設為指向 IBM MQ COBOL 記錄定義檔 (在一行上輸入下列程式碼):

```
set COBCPY= MQ_INSTALLATION_PATH\
Tools\Cobol\Copybook
```

2. 編譯程式以提供物件檔:

```
cobol amq0put0 LITLINK
```

3. 將物件檔鏈結至執行時期系統。
   - 設定 LIB 環境變數以指向編譯器 COBOL 程式庫。
   - 鏈結物件檔以在 IBM MQ 伺服器上使用:

```
cbllink amq0put0.obj mqmcb.lib
```

   - 或者鏈結物件檔以在 IBM MQ 用戶端上使用:

```
cbllink amq0put0.obj mqiccb.lib
```

準備 CICS 及 Transaction Server 程式

若要使用 IBM VisualAge COBOL 來編譯及鏈結 TXSeries for Windows NT V5.1 程式:

1. 設定環境變數 (在一行上輸入下列程式碼):

```
set CICS_IBMCOB_FLAGS= MQ_INSTALLATION_PATH\
Cobol\Copybook\VAcobol;%CICS_IBMCOB_FLAGS%
```

2. 設定 USERLIB 環境變數:

```
set USERLIB=MQMCBB.LIB
```

3. 轉換、編譯及鏈結您的程式:

```
cicstcl -l IBMCOB myprog.ccp
```

This is described in the *Transaction Server for Windows NT*， *V4 Application Programming* Guide.

如果要使用 Micro Focus COBOL 來編譯及鏈結 CICS for Windows V5 程式，請執行下列動作:

- 設定 INCLUDE 變數:

```
set
INCLUDE=drive:\programname\ibm\websphere\tools\c\include;
        drive:\opt\cics\include;%INCLUDE%
```

- 設定 COBCPY 環境變數:

```
setCOBCPY=drive:\programname\ibm\websphere\tools\cobol\copybook;
          drive:\opt\cics\include
```

- 設定 COBOL 選項:

- set
- COBOPTS=/LITLINK /NOTRUNC

並執行下列程式碼:

```
cicstran cicsmq00.ccp
cobol cicsmq00.cbl /LITLINK /NOTRUNC
cbllink -D -Mcicsmq00 -Ocicsmq00.cbmfnt cicsmq00.obj
%CICSLIB%\cicsprCBMFNT.lib user32.lib msvcrt.lib kernel32.lib mqmcb.lib
```

Windows 在 *Windows* 中準備 *Visual Basic* 程式

在 Windows 上使用 Microsoft Visual Basic 程式時要考量的資訊。

Deprecated 從 IBM MQ 9.0 開始， Microsoft Visual Basic 6.0 的支援已淘汰。 .NET 的 IBM MQ 類別是建議的取代技術。 如需相關資訊，請參閱 開發 .NET 應用程式。

註: 未提供 64 位元版本的 Visual Basic 模組檔案。

若要在 Windows 上準備 Visual Basic 程式，請執行下列動作:

1. 建立新的專案。
2. 新增所提供的模組檔案 CMQB.BAS，至專案。
3. 如果您需要其他提供的模組檔案，請新增它們:
   - CMQBB.BAS: MQAI 支援
   - CMQCFB.BAS: PCF 支援
   - CMQXB.BAS: 通道結束程式支援
   - CMQPSB.BAS: 發佈/訂閱

如需在 Visual Basic 內使用 MQCONNXAny 呼叫的相關資訊，請參閱 第 883 頁的『在 Visual Basic 中撰寫程式碼』 。

在專案程式碼中進行任何 MQI 呼叫之前，請先呼叫 MQ_SETDEFAULTS 程序。 此程序會設定 MQI 呼叫所需的預設結構。

透過設定條件式編譯變數 *MqType*，指定在編譯或執行專案之前是否要建立 IBM MQ 伺服器或用戶端。 將 Visual Basic 專案中的 *MqType* 設為 1 (若為伺服器) 或 2 (若為用戶端) ，如下所示:

1. 選取專案功能表。
2. 選取 *Name* 內容 (其中 *Name* 是現行專案的名稱)。
3. 選取對話框中的「製作」標籤。
4. 在條件式編譯引數欄位中，針對伺服器輸入下列指令:

   ```
   MqType=1
   ```

   或針對用戶端:

   ```
   MqType=2
   ```

相關概念

第 883 頁的『在 Visual Basic 中撰寫程式碼』
在 Microsoft Visual Basic 中編碼 IBM MQ 程式時要考量的資訊。 Visual Basic 僅在 Windows 上受支援。

相關參考

第 770 頁的『鏈結 Visual Basic 應用程式與 IBM MQ MQI client 程式碼』
您可以將 Microsoft Visual Basic 應用程式與 Windows 上的 IBM MQ MQI client 程式碼相鏈結。

Windows *SSPI* 安全結束程式

IBM MQ for Windows 提供 IBM MQ MQI client 和 IBM MQ 伺服器兩者的安全結束程式。 這是通道跳出程式，透過使用「安全服務程式設計介面 (SSPI)」為 IBM MQ 通道提供鑑別。 SSPI 提供 Windows 系統的整合式安全機能。

安全套件是從 security.dll 或 secur32.dll 載入。 這些 DLL 隨您的作業系統一起提供。

使用 NTLM 鑑別服務提供單向鑑別。 使用 Kerberos 鑑別服務提供雙向鑑別。

以來源及物件格式提供安全跳出程式。 您可以依現狀使用物件程式碼，也可以使用原始碼作為起點來建立您自己的使用者結束程式。

另請參閱第 954 頁的『在 Windows 上使用 SSPI 安全結束程式』。

安全結束程式簡介

安全結束程式會在兩個安全結束程式之間形成一個安全連線，其中的一個程式是用於傳送端訊息通道代理程式 (MCA)，另一個則是用於接收端 MCA。

起始安全連線的程式 (即建立 MCA 階段作業之後第一個取得控制權的程式) 稱為 環境定義起始器。 夥伴程式稱為 環境定義接收器。

下表顯示部分通道類型，這些通道類型是環境定義起始程式及其相關聯的環境定義接受程式。

表 *149:* 環境定義起始器及其相關聯的環境定義接收器

環境定義起始器	**環境定義接收器**
MQCHT_CLNTCONN	MQCHT_SVRCONN
MQCHT_RECEIVER	MQCHT_SENDER
MQCHT_CLUSRCVR	MQCHT_CLUSSDR

安全跳出程式有兩個進入點:

- **SCY_NTLM**

  這會使用 NTLM 鑑別服務，提供單向鑑別。 NTLM 可讓伺服器驗證其用戶端的身分。 它不容許用戶端驗證伺服器的身分，或一個伺服器驗證另一個伺服器的身分。 NTLM 鑑別是針對網路環境所設計，其中假設伺服器是真實的。

- **SCY_KERBEROS**

  這會使用 Kerberos 交互鑑別服務。 Kerberos 通訊協定不假設網路環境中的伺服器是真實的。 網路連線兩端的當事人可以驗證另一方的身分。 也就是說，伺服器可以驗證用戶端及其他伺服器的身分，而用戶端可以驗證伺服器的身分。

安全結束程式的作用

本主題說明 SSPI 通道結束程式的用途。

當建立階段作業時，所提供的通道結束程式會提供友機系統的單向或雙向 (交互) 鑑別。 對於特定通道，每一個結束程式都有相關聯的 主體 (類似於使用者 ID，請參閱 第 858 頁的『IBM MQ 存取控制及 Windows 主體』)。 兩個跳出程式之間的連線是兩個主體之間的關聯。

建立基礎階段作業之後，會建立兩個安全結束程式 (一個用於傳送 MCA，另一個用於接收 MCA) 之間的安全連線。 作業順序如下:

1. 每一個程式都與特定主體相關聯，例如，明確登入作業的結果。
2. 環境定義起始器會從安全套件 (若為 Kerberos，指名的夥伴) 要求與夥伴的安全連線，並接收記號 (稱為 token1)。 記號會使用已建立的基礎階段作業傳送至友機程式。

3. 夥伴程式 (環境定義接收器) 會將 token1 傳遞至安全套件，以驗證環境定義起始器是真實的。 對於 NTLM，現在已建立連線。
4. 對於 Kerberos 提供的安全結束程式 (亦即，用於交互鑑別)，安全套件也會產生第二個記號 (稱為 token2)，環境定義接收器會使用基礎階段作業傳回給環境定義起始器。
5. 環境定義起始器使用 token2 來驗證環境定義接收器是真實的。
6. 在此階段，如果兩個應用程式都滿意夥伴記號的確實性，則會建立安全 (已鑑別) 連線。

IBM MQ 存取控制及 Windows 主體

IBM MQ 提供的存取控制基於使用者和群組。 Windows 提供的鑑別基於主體，例如使用者和 servicePrincipal 名稱 (SPN)。 如果是 servicePrincipal 名稱，可能有許多這些名稱與單一使用者相關聯。

SSPI 安全結束程式會使用相關 Windows 主體進行鑑別。 如果 Windows 鑑別成功，結束程式會將與 Windows 主體相關聯的使用者 ID 傳遞至 IBM MQ，以進行存取控制。

與鑑別相關的 Windows 主體會因所用的鑑別類型而異。

- 對於 NTLM 鑑別，「環境定義起始器」的 Windows 主體是與執行中程序相關聯的使用者 ID。 因為此鑑別是單向的，所以與 Context Acceptor 相關聯的主體並不相關。
- 對於 Kerberos 鑑別，在 CLNTCONN 通道上， Windows 主體是與執行中處理程序相關聯的使用者 ID。 否則， Windows 主體是透過將下列字首新增至 QueueManager 名稱所形成的 servicePrincipal 名稱。

```
ibmMQSeries/
```

z/OS 在 z/OS 上建置程序化應用程式

CICS、 IMS 及 z/OS 出版品說明如何建置在這些環境中執行的應用程式。

此主題集合說明建置這些環境的 IBM MQ for z/OS 應用程式時必須執行的其他作業，以及標準作業的變更。 支援 COBOL、C、C++、Assembler 及 PL/I 程式設計語言。 (如需建置 C++ 應用程式的相關資訊，請參閱 使用 C++。)

您必須執行才能建立可執行 IBM MQ for z/OS 應用程式的作業，取決於撰寫程式所用的程式設計語言，以及應用程式將執行的環境。

除了在程式中撰寫 MQI 呼叫的程式碼之外，請新增適當的語言陳述式，以包括您所使用語言的 IBM MQ for z/OS 資料定義檔。 讓您自己熟悉這些檔案的內容。 如需完整說明，請參閱 第 601 頁的『IBM MQ 資料定義檔』。

附註

名稱 **thlqual** 是 z/OS 上安裝程式庫的高階限定元。

z/OS 準備要執行的程式

在為 IBM MQ 應用程式撰寫程式以建立可執行應用程式之後，您必須編譯或組合它，然後將產生的物件程式碼與 IBM MQ for z/OS 為其支援的每一個環境提供的 Stub 程式鏈結編輯。

您如何準備程式取決於應用程式執行所在的環境 (批次、 CICS、 IMS(BMP 或 MPP) Linux 或 z/OS UNIX System Services)，以及 z/OS 安裝上的資料集結構。

第 864 頁的『動態呼叫 IBM MQ Stub』 說明在程式中進行 MQI 呼叫的替代方法，讓您不需要鏈結編輯 IBM MQ Stub。 此方法並非適用於所有語言及環境。

請勿鏈結-編輯比您程式執行所在 IBM MQ for z/OS 版本更高層次的 Stub 程式。 例如，在 MQSeries for OS/390 V5.2 上執行的程式，不得使用 IBM MQ for z/OS V7 所提供的 Stub 程式進行鏈結編輯。

z/OS 建置 *64* 位元 *C* 應用程式

在 z/OS 中，使用 LP64 編譯器及連結程式選項來建置 64 位元 C 應用程式。 IBM MQ for z/OS *cmqc.h* 標頭檔可辨識何時將此選項提供給編譯器，並產生適用於 64 位元作業的 IBM MQ 資料類型及結構。

使用此選項建置的 C 程式碼必須建置成使用適合所需協調語意的動態鏈結程式庫 (DLL)。 若要達到此目的，請將編譯的程式碼與下表中定義的適當副卡片組連結:

表 *150:* 每一個協調語意所需的副卡片組名稱	
對等連接詞	**副卡片組名稱**
單相確定 MQI	CSQBMQ2X
以 RRS 協調進行兩階段確定，使用 RRS 動詞	CSQBRR2X
兩階段使用 RRS 協調來確定，使用 MQI 動詞	CSQBRI2X

註: 對於 31 位元 C 應用程式，您也可以設定呼叫介面 (Language Environment 或 XPLINK) 的編譯器選項，如 第 860 頁的『使用 31 位元 Language Environment 或 XPLINK 來建置 z/OS 批次應用程式』中所述。 對於 64 位元 C 應用程式，您未指定呼叫介面，因為唯一支援的鏈結是 XPLINK。

使用 *z/OS XL C/C++*所提供的 EDCQCB JCL 程序，將單一階段確定 IBM MQ 程式建置成批次工作，如下所示:

```
//PROCS  JCLLIB ORDER=CBC.SCCNPRC
//CLG EXEC EDCQCB,
// INFILE='thlqual.SCSQC37S(CSQ4BCG1)',   < MQ SAMPLES
// CPARM='RENT,SSCOM,DLL,LP64,LIST,NOMAR,NOSEQ',  < COMPILER OPTIONS
// LIBPRFX='CEE',             < PREFIX FOR LIBRARY DSN
// LNGPRFX='CBC',             < PREFIX FOR LANGUAGE DSN
// BPARM='MAP,XREF,RENT,DYNAM=DLL', < LINK EDIT OPTIONS
// OUTFILE='userid.LOAD(CSQ4BCG1),DISP=SHR'
//COMPILE.SYSLIB DD
//         DD
//         DD DISP=SHR,DSN=thlqual.SCSQC370
//BIND.SCSQDEFS DD DISP=SHR,DSN=thlqual.SCSQDEFS
//BIND.SYSIN DD *
INCLUDE SCSQDEFS(CSQBMQ2X)
NAME CSQ4BCG1
```

如果要在 z/OS UNIX System Services 中建置 RRS 協調程式，請依照下列方式來編譯和鏈結:

```
cc -o mqsamp -W c,LP64,DLL -W l,DYNAM=DLL,LP64 -I"//'thlqual.SCSQC370'" "//'thlqual.SCSQDEFS(CSQBRR2X)'" mqsamp.c
```

z/OS 建置 *z/OS* 批次應用程式

瞭解如何建置 z/OS 批次應用程式，以及在這麼做時要考量的步驟。

若要為 IBM MQ for z/OS 建置在 z/OS 批次下執行的應用程式，請建立執行下列作業的工作控制語言 (JCL):

1. 編譯 (或組合) 程式以產生物件程式碼。 編譯的 JCL 必須包含 SYSLIB 陳述式，讓編譯器可以使用產品資料定義檔。 下列 IBM MQ for z/OS 程式庫中提供資料定義:
   - 若為 COBOL: **thlqual**.SCSQCOBC
   - 對於組譯語言: **thlqual**.SCSQMACS
   - 若為 C: **thlqual**.SCSQC370
   - 若為 PL/I， **thlqual**.SCSQPLIC
2. 若為 C 應用程式，請預先鏈結在步驟 第 859 頁的『1』中建立的物件程式碼。
3. 若為 PL/I 應用程式，請使用編譯器選項 EXTRN (SHORT)。
4. 鏈結-編輯在步驟 第 859 頁的『1』 (或 C 應用程式的步驟 第 859 頁的『2』) 中建立的物件程式碼，以產生載入模組。 當您鏈結編輯程式碼時，必須包括其中一個 IBM MQ for z/OS 批次 Stub 程式 (CSQBSTUB 或其中一個 RRS Stub 程式 :CSQBRRSI 或 CSQBRSTB)。

   CSQBSTUB
   IBM MQ for z/OS 提供的單階段確定

   CSQBRRSI
   RRS 使用 MQI 提供的兩段式確定

   CSQBRSTB
   RRS 直接提供的兩段式確定

附註:

a. 如果您使用 CSQBRSTB，也必須使用 SYS1.CSSLIB。 第 860 頁的圖 113 和 第 860 頁的圖 114 顯示要執行此動作的 JCL 片段。 Stub 與語言無關，並在程式庫 **thlqual**.SCSQLOAD 中提供。

b. 如果您的應用程式在「語言環境」下執行，您應該確保改為依照 第 860 頁的『使用 31 位元 Language Environment 或 XPLINK 來建置 z/OS 批次應用程式』中的說明，以「語言環境 DLL」來鏈結編輯。

5. 將載入模組儲存在應用程式載入程式庫中。

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING BATCH STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//*
⋮
//SYSIN     DD *
  INCLUDE CSQSTUB(CSQBSTUB)
⋮
/*
```

圖 *113:* 在批次環境中使用單階段確定來鏈結編輯物件模組的 *JCL* 片段

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING BATCH STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//CSSLIB    DD  DSN=SYS1.CSSLIB,DISP=SHR
//*
⋮
//SYSIN     DD *
INCLUDE CSQSTUB(CSQBRSTB)
INCLUDE CSSLIB(ATRSCSS)
⋮
/*
```

圖 *114:* 在批次環境中使用兩段式確定來鏈結編輯物件模組的 *JCL* 片段

若要執行批次或 RRS 程式，您必須在 STEPLIB 或 JOBLIB 資料集連結中併入程式庫 **thlqual**.SCSQAUTH 和 **thlqual**.SCSQLOAD。

若要執行 TSO 程式，您必須在 TSO 階段作業使用的 STEPLIB 中包含程式庫 **thlqual**.SCSQAUTH 及 **thlqual**.SCSQLOAD。

若要從 z/OS UNIX System Services Shell 執行批次程式，請將程式庫 **thlqual**.SCSQAUTH 及 **thlqual**.SCSQLOAD 新增至 $HOME? .profile 中的 STEPLIB 規格，如下所示:

```
STEPLIB= thlqual.SCSQAUTH: thlqual.SCSQLOAD
export STEPLIB
```

z/OS 使用 *31* 位元 *Language Environment* 或 *XPLINK* 來建置 *z/OS* 批次應用程式

IBM MQ for z/OS 提供鏈結編輯應用程式時必須使用的一組動態鏈結程式庫 (DLL)。

程式庫有兩種變式，可讓應用程式使用下列其中一個呼叫介面:

- 31 位元 Language Environment 呼叫介面。
- 31 位元 XPLINK 呼叫介面。 z/OS XPLINK 是適用於 C 應用程式的高效能呼叫慣例。 請參閱 z/OS 2.2 文件中的 XPLINK | NOXPLINK 。

如果要使用 DLL，應用程式會連結或鏈結至稱為 *sidedecks* 的應用程式，而不是舊版所提供的 Stub。 在 SCSQDEFS 程式庫中找到 sidedecks (而非 SCSQLOAD 程式庫)。

表 *151:* 動態鏈結程式庫的變式			
確定	**31 位元 Language Environment DLL**	**31 位元 XPLINK DLL**	**對等的 Stub 名稱**
1 階段確定 MQI 程式庫	CSQBMQ1	CSQBMQ1X	CSQBSTUB
使用 RRS 交易控制動詞，以 RRS 協調進行 2 階段確定	CSQBRR1	CSQBRR1X	CSQBRSTB
使用 MQI 交易控制動詞，利用 RRS 協調進行 2 階段確定	CSQBRI1	CSQBRI1X	CSQBRRSI

註: 所有側面包含資料轉換進入點 MQXCNVC 的定義，先前已透過包括 CSQASTUB 來解析。

常見問題:

- 如果您的應用程式使用非同步訊息耗用 (MQCB、MQCTL 或 MQSUB 呼叫)，且未使用前一個 DLL 介面，則下列訊息會出現在工作日誌中:

  ```
  CSQB001E 在 z/OS 批次或 z/OS UNIX System Services 中執行的語言環境程式必須使用 DLL 介面 IBM MQ
  ```

  解決方案: 使用 sidedecks 而非先前詳述的 Stub 來重建應用程式。

- 在程式建置時，會出現下列訊息

  ```
  IEW2469E your-code 區段中 MQAPI-NAME 參照的屬性不符合下列項目的屬性:
  目標符號
  ```

  原因: 這表示您已使用 V701 版 (或更新版本) cmqc.h 編譯 XPLINK 程式，但未與 sidedecks 連結。

  解決方案: 變更程式的建置檔，以從 SCSQDEFS 而非 SCSQLOAD 中針對適當的副卡片組進行連結，而不是針對 Stub 進行連結

下列範例 JCL 示範如何編譯及鏈結編輯 C 程式，以使用 31 位元 Language Environment DLL 呼叫介面:

```
//CLG EXEC EDCCB,
//  INFILE=MYPROGS.CPROGS(MYPROGRAM),
//  CPARM='OPTF(DD:OPTF)',
//  BPARM='XREF,MAP,DYNAM=DLL'           < LINKEDIT OPTIONS
//COMPILE.OPTF DD *
RENT,CHECKOUT(ALL),SSCOM,DEFINE(MVS),NOMARGINS,NOSEQ,DLL
SE(DD:SYSLIBV)
//COMPILE.SYSLIB DD
//               DD
//               DD DISP=SHR,DSN=hlq.SCSQC370
//COMPILE.SYSLIBV DD DISP=SHR,DSN=hlq.BASE.H
/*
//BIND.SYSOBJ DD DISP=SHR,DSN=CEE.SCEEOBJ
//            DD DISP=SHR,DSN=hlq.SCSQDEFS
//BIND.SYSLMOD DD DISP=SHR,DSN=hlq.LOAD(MYPROGAM)
//BIND.SYSIN  DD *
 ENTRY CEESTART
 INCLUDE SYSOBJ(CSQBMQ1)
 NAME MYPROGAM(R)
//
```

註: 編譯會使用 **DLL** 選項。 鏈結編輯會使用 **DYNAM=DLL** 選項，並參照 **CSQBMQ1** 程式庫。

下列範例 JCL 示範如何編譯及鏈結編輯 C 程式，以使用 31 位元 XPLINK DLL 呼叫介面:

```
//CLG  EXEC EDCXCB,
//  INFILE=MYPROGS.CPROGS(MYPROGRAM),
//  CPARM='OPTF(DD:OPTF)',
//  BPARM='XREF,MAP,DYNAM=DLL'           < LINKEDIT OPTIONS
//COMPILE.OPTF DD *
RENT,CHECKOUT(ALL),SSCOM,DEFINE(MVS),NOMARGINS,NOSEQ,XPLINK,DLL
SE(DD:SYSLIBV)
//COMPILE.SYSLIB DD
//               DD
```

```
//               DD DISP=SHR,DSN=hlq.SCSQC370
//COMPILE.SYSLIBV DD DISP=SHR,DSN=hlq.BASE.H
/*
//BIND.SYSOBJ DD DISP=SHR,DSN=CEE.SCEEOBJ
//            DD DISP=SHR,DSN=hlq.SCSQDEFS
//BIND.SYSLMOD DD DISP=SHR,DSN=hlq.LOAD(MYPROGAM)
//BIND.SYSIN  DD *
 ENTRY CEESTART
 INCLUDE SYSOBJ(CSQBMQ1X)
 NAME MYPROGAM(R)
//
```

註: 編譯使用 **XPLINK** 和 **DLL** 選項。 鏈結編輯會使用 **DYNAM=DLL** 選項，並參照 **CSQBMQ1X** 程式庫。

請確定您將編譯選項 DLL 新增至模組中的每一個程式。 IEW2456E 9207 SYMBOL CSQ1BAK RESOLVED 之類的訊息指出您需要檢查所有程式是否已使用 DLL 選項進行編譯。

z/OS 在 *z/OS* 中建置 *CICS* 應用程式

在 z/OS 中建置 CICS 應用程式時，請使用此資訊。

若要為在 CICS 下執行的 IBM MQ for z/OS 建置應用程式，您必須:

- 將程式中的 CICS 指令轉換為撰寫程式其餘部分的語言。
- 編譯或組合轉換器的輸出，以產生物件程式碼。
  - 若為 PL/I 程式，請使用編譯器選項 EXTRN (SHORT)。
  - 對於 C 應用程式，如果應用程式未使用 XPLINK，請使用編譯器選項 DEFINE (MQ_OS_LINKAGE=1)。
- 鏈結-編輯物件程式碼以建立載入模組。

CICS 提供針對其支援的每一種程式設計語言依序執行這些步驟的程序。

- 若為 CICS Transaction Server for z/OS， *CICS Transaction Server for z/OS* 系統定義手冊 說明如何使用這些程序， *CICS/ESA* 應用程式設計手冊 會提供轉換程序的相關資訊。

您必須包括:

- 在編譯 (或組合) 階段的 SYSLIB 陳述式中，使產品資料定義檔可供編譯器使用的陳述式。 下列 IBM MQ for z/OS 程式庫中提供資料定義:
  - 若為 COBOL: **thlqual**.SCSQCOBC
  - 對於組譯語言: **thlqual**.SCSQMACS
  - 若為 C: **thlqual**.SCSQC370
  - 若為 PL/I， **thlqual**.SCSQPLIC
- 在鏈結編輯 JCL 中，這是 IBM MQ for z/OS CICS Stub 程式 (CSQCSTUB)。 第 862 頁的圖 115 顯示要執行此動作的 JCL 程式碼片段。 Stub 與語言無關，在程式庫 **thlqual**.SCSQLOAD 中提供。

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING CICS STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//*
⋮
//LKED.SYSIN DD *
  INCLUDE CSQSTUB(CSQCSTUB)
  ⋮
/*
```

圖 *115:* 在 *CICS* 環境中鏈結編輯物件模組的 *JCL* 片段

- 若為 CICS TS 3.2 之後的 CICS 版本，或者如果您想要使用 IBM MQ 訊息內容 API，或 IBM MQ API MQCB、MQCTL、MQSTAT、MQSUB 或 MQSUBR，您必須以 CICS 提供的 Stub DFHMQSTB 而非 IBM MQ 提供的 CSQCSTUB 來鏈結編輯物件碼。 如需為 CICS 建置 IBM MQ 程式的相關資訊，請參閱 CICS 產品說明文件中的 用來存取 IBM MQ MQI 呼叫的 API Stub 程式 。

完成這些步驟之後，請將載入模組儲存在應用程式載入程式庫中，並以一般方式將程式定義為 CICS 。

在執行 CICS 程式之前，系統管理者必須將它定義為 CICS 作為 IBM MQ 程式及交易，然後您可以一般方式執行它。

z/OS 建置 *IMS (BMP 或 MPP)* 應用程式

建置 IMS (BMP 或 MPP) 應用程式時，請使用此資訊。

如果您要建置批次 DL/I 程式，請參閱 第 859 頁的『建置 z/OS 批次應用程式』。 若要建置在 IMS 下執行的其他應用程式 (作為 BMP 或 MPP) ，請建立 JCL 來執行下列作業:

1. 編譯 (或組合) 程式以產生物件程式碼。 編譯的 JCL 必須包含 SYSLIB 陳述式，讓編譯器可以使用產品資料定義檔。 下列 IBM MQ for z/OS 程式庫中提供資料定義:
   - 若為 COBOL: **thlqual**.SCSQCOBC
   - 對於組譯語言: **thlqual**.SCSQMACS
   - 若為 C: **thlqual**.SCSQC370
   - 若為 PL/I: **thlqual**.SCSQPLIC
2. 若為 C 應用程式，請預先鏈結在步驟 第 863 頁的『1』中建立的物件模組。
3. 若為 PL/I 程式，請使用編譯器選項 EXTRN (SHORT)。
4. 對於 C 應用程式，如果應用程式未使用 XPLINK，請使用編譯器選項 DEFINE (MQ_OS_LINKAGE=1)。
5. 鏈結-編輯在步驟 第 863 頁的『1』 (或 C/370 應用程式的步驟 第 863 頁的『2』) 中建立的物件程式碼，以產生載入模組:
   a. 包括 IMS 語言介面模組 (DFSLI000)。
   b. 包括 IBM MQ for z/OS IMS Stub 程式 (CSQQSTUB)。 第 863 頁的圖 116 顯示要執行此動作的 JCL 片段。 Stub 與語言無關，並在程式庫 **thlqual**.SCSQLOAD 中提供。

      註: 如果您使用 COBOL ，請選取 NODYNAM 編譯器選項，讓鏈結編輯器能夠解析 CSQQSTUB 的參照，除非您打算依照 第 864 頁的『動態呼叫 IBM MQ Stub』中的說明來使用動態鏈結。
6. 將載入模組儲存在應用程式載入程式庫中。

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING IMS STUB
//*
//CSQSTUB   DD  DSN=thlqual.SCSQLOAD,DISP=SHR
//*
⋮
//LKED.SYSIN DD *
  INCLUDE CSQSTUB(CSQQSTUB)
  ⋮
/*
```

圖 *116:* 在 *IMS* 環境中鏈結編輯物件模組的 *JCL* 片段

在執行 IMS 程式之前，系統管理者必須將它定義為 IMS 作為 IBM MQ 程式及交易: 然後您可以一般方式執行它。

z/OS 建置 *z/OS UNIX System Services* 應用程式

建置 z/OS UNIX System Services 應用程式時，請使用此資訊。

若要為 IBM MQ for z/OS 建置在 z/OS UNIX System Services 下執行的 C 應用程式，請編譯並鏈結您的應用程式，如下所示:

```
cc -o mqsamp -W c,DLL -I "//' thlqual.SCSQC370'" mqsamp.c "//' thlqual.SCSQDEFS(CSQBMQ1)'"
```

其中 **thlqual** 是安裝所使用的高階限定元。

若要執行 C 程式，您需要將下列內容新增至 .profile 檔案; 這應該位於根目錄中:

```
STEPLIB= thlqual.SCSQANLE:thlqual.SCSQAUTH: STEPLIB
```

請注意，您需要結束 z/OS UNIX System Services，並再次輸入 z/OS UNIX System Services ，才能辨識變更。

如果您要執行多個 Shell ，請在該行的開頭新增單字 export ，亦即:

```
export STEPLIB= thlqual.SCSQANLE:thlqual.SCSQAUTH: STEPLIB
```

順利完成之後，您可以鏈結 CSQBSTUB 並發出 IBM MQ 呼叫。

第 864 頁的『動態呼叫 IBM MQ Stub』 說明在程式中進行 MQI 呼叫的替代方法，讓您不需要鏈結編輯 IBM MQ Stub。 此方法並非適用於所有語言及環境。

請勿鏈結-編輯比您程式執行所在 IBM MQ for z/OS 版本更高層次的 Stub 程式。 例如，在 IBM WebSphere MQ for z/OS 7.1 上執行的程式不得使用 IBM MQ for z/OS 8.0 提供的 Stub 程式進行鏈結編輯。

z/OS 動態呼叫 *IBM MQ Stub*

您可以從程式內動態呼叫 Stub ，而不是使用物件程式碼來鏈結編輯 IBM MQ Stub 程式。

您可以在批次、 IMS 及 CICS 環境中執行此動作。 RRS 環境不支援此機能。 如果您的應用程式使用 RRS 來協調更新項目，請參閱 第 868 頁的『RRS 考量』。

不過，此方法:

- 增加程式的複雜性
- 在執行時增加程式所需的儲存體
- 降低程式的效能
- 表示您無法在其他環境中使用相同的程式

如果您動態呼叫 Stub ，則執行時必須有適當的 Stub 程式及其別名可用。 若要確保如此，請包括 IBM MQ for z/OS 資料集 SCSQLOAD:

- 對於批次和 IMS，在 JCL 的 STEPLIB 連結中。
- 對於 CICS，在 CICS DFHRPL 連結中。

對於 IMS，請確保包含動態 Stub 的程式庫 (如 設定 IMS 配接器 中安裝 IMS 配接器的相關資訊所述建置) 在區域 JCL 的 STEPLIB 連結中，位於資料集 SCSQLOAD 之前。

當您動態呼叫 Stub 時，請使用 第 864 頁的表 152 中顯示的名稱。 在 PL/I 中，只宣告程式中使用的呼叫名稱。

表 *152:* 動態鏈結的呼叫名稱

MQI 呼叫	**批次 (非 RRS) 動態呼叫名稱**	**CICS 動態呼叫名稱**	**IMS 動態呼叫名稱**
MQBACK	CSQBBACK	不支援	不支援
MQBUFMH	CSQBBFMH	CSQCBFMH [1]	MQBUFMH
MQCB	CSQBCB	CSQCCB [1]	不支援
MQCLOSE	CSQBCLOS	CSQCCLOS	MQCLOSE
MQCMIT	CSQBCOMM	不支援	不支援
MQCONN	CSQBCONN	CSQCCONN	MQCONN
MQCONNX	CSQBCONX	CSQCCONX	MQCONNX
MQCRTMH	CSQBCTMH	CSQCCTMH [1]	MQCRTMH

表 *152:* 動態鏈結的呼叫名稱 *(*繼續*)*

MQI 呼叫	**批次 (非 RRS) 動態呼叫名稱**	**CICS 動態呼叫名稱**	**IMS 動態呼叫名稱**
MQCTL	CSQBCTL	CSQCCTL [1]	不支援
MQDISC	CSQBDISC	CSQCDISC	MQDISC
MQDLTMH	CSQBDTMH	CSQCDTMH [1]	MQDLTMH
MQDLTMP	CSQBDTMP	CSQCDTMP [1]	MQDLTMP
MQGet	CSQBGET	CSQCGET	MQGET
MQINQ	CSQBINQ	CSQCINQ	MQINQ
MQINQMP	CSQBIQMP	CSQCIQMP [1]	MQINQMP
MQMHBUF	CSQBMHBF	CSQCMHBF [1]	MQMHBUF
MQOPEN	CSQBOPEN	CSQCOPEN	MQOPEN
MQPUT	CSQBPUT	CSQCPUT	MQPUT
MQPUT1	CSQBPUT1	CSQCPUT1	MQPUT1
MQSET	CSQBSET	CSQCSET	MQSET
MQSETMP	CSQBSTMP	CSQCSTMP [1]	MQSETMP
MQSTAT	CSQBSTAT	CSQCSTAT [1]	MQSTAT
MQSUB	CSQBSUB	CSQCSUB [1]	MQSUB
MQSUBRQ	CSQBSUBR	CSQCSUBR [1]	MQSUBRQ

註: 1. 只有在使用 CICS TS 3.2 或更新版本，且必須使用 CICS 隨附的 CSQCSTUB 時，才能使用這些 API 呼叫。 若為 CICS TS 3.2，必須套用 APAR PK66866 。 若為 CICS TS 4.1，必須套用 APAR PK89844 。

如需如何使用此技術的範例，請參閱下列圖:

- 批次和 COBOL: 請參閱 第 866 頁的圖 117
- CICS 和 COBOL: 請參閱 第 866 頁的圖 118
- IMS 和 COBOL: 請參閱 第 866 頁的圖 119
- 批次和組譯: 請參閱 第 867 頁的圖 120
- CICS 和組譯器: 請參閱 第 867 頁的圖 121
- IMS 和組譯器: 請參閱 第 867 頁的圖 122
- 批次和 C: 第 867 頁的圖 123
- CICS 和 C: 請參閱 第 867 頁的圖 124
- IMS 和 C: 請參閱 第 868 頁的圖 125
- 批次和 PL/I: 請參閱 第 868 頁的圖 126
- IMS 和 PL/I: 請參閱 第 868 頁的圖 127

```
...
       WORKING-STORAGE SECTION.
...
          05 WS-MQOPEN                     PIC X(8) VALUE 'CSQBOPEN'.
...
       PROCEDURE DIVISION.
...
           CALL WS-MQOPEN WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
```

圖 *117:* 在批次環境中使用 *COBOL* 動態鏈結

```
...
       WORKING-STORAGE SECTION.
...
          05 WS-MQOPEN                     PIC X(8) VALUE 'CSQCOPEN'.
...
       PROCEDURE DIVISION.
...
           CALL WS-MQOPEN WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
```

圖 *118:* 在 *CICS* 環境中使用 *COBOL* 的動態鏈結

```
...
       WORKING-STORAGE SECTION.
...
          05 WS-MQOPEN                     PIC X(8) VALUE 'MQOPEN'.
...
       PROCEDURE DIVISION.
...
           CALL WS-MQOPEN WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
      * ------------------------------------------------------------ *
      *
      *    If the compilation option 'DYNAM' is specified
      *    then you may code the MQ calls as follows
      *
      * ------------------------------------------------------------ *
...
           CALL 'MQOPEN'  WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
```

圖 *119:* 在 *IMS* 環境中使用 *COBOL* 的動態鏈結

```
...
         LOAD   EP=CSQBOPEN
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         DELETE EP=CSQBOPEN
...
```

圖 *120:* 在批次環境中使用組合語言的動態鏈結

```
...
         EXEC CICS LOAD PROGRAM('CSQCOPEN') ENTRY(R15)
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         EXEC CICS RELEASE PROGRAM('CSQCOPEN')
...
```

圖 *121:* 在 *CICS* 環境中使用組合語言的動態鏈結

```
...
         LOAD   EP=MQOPEN
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         DELETE EP=MQOPEN
...
```

圖 *122:* 在 *IMS* 環境中使用組合語言的動態鏈結

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * csqbopen;
...
csqbopen = (CALL_ME *) fetch("CSQBOPEN");
(*csqbopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

圖 *123:* 在批次環境中使用 *C* 語言進行動態鏈結

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * csqcopen;
...
  EXEC CICS LOAD PROGRAM("CSQCOPEN") ENTRY(csqcopen);
(*csqcopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

圖 *124:* 在 *CICS* 環境中使用 *C* 語言進行動態鏈結

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * mqopen;
...
mqopen = (CALL_ME *) fetch("MQOPEN");
(*mqopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

圖 *125:* 在 *IMS* 環境中使用 *C* 語言進行動態鏈結

```
...
      DCL CSQBOPEN ENTRY EXT OPTIONS(ASSEMBLER INTER);
...
      FETCH CSQBOPEN;

      CALL CSQBOPEN(HQM,
                    MQOD,
                    OPTIONS,
                    HOBJ,
                    COMPCODE,
                    REASON);

      RELEASE CSQBOPEN;
```

圖 *126:* 在批次環境中使用 *PL/I* 動態鏈結

```
...
      DCL MQOPEN   ENTRY EXT OPTIONS(ASSEMBLER INTER);
...
      FETCH MQOPEN;

      CALL   MQOPEN(HQM,
                    MQOD,
                    OPTIONS,
                    HOBJ,
                    COMPCODE,
                    REASON);

      RELEASE   MQOPEN;
```

圖 *127:* 在 *IMS* 環境中使用 *PL/I* 的動態鏈結

z/OS *RRS 考量*

如果您的應用程式使用 RRS 來協調更新項目，請考量使用此資訊。

IBM MQ 為需要 RRS 協調的批次程式提供兩個不同的 Stub-請參閱 第 747 頁的『RRS 批次配接卡』。 批次配接器會根據 MQCONN 或 MQCONNX API 上 Stub 常式所傳遞的資訊，在 MQCONN 時間決定後續 API 呼叫的行為差異。 這表示需要 RRS 協調的批次程式可以使用動態 API 呼叫，前提是 IBM MQ 的起始連線是使用適當的 Stub 來完成。 下列範例說明這一點:

```
       WORKING-STORAGE SECTION.
           05 WS-MQOPEN       PIC X(8) VALUE 'MQOPEN' .
.
.
.
       PROCEDURE DIVISION.
.
.
.
       *
       * Static call to MQCONN must be resolved by linkage edit to
       * CSQBRSTB or CSQBRRSI for RRS coordination
       *
           CALL 'MQCONN' USING W00-QMGR
                               W03-HCONN
                               W03-COMPCODE
```

```
                                W03-REASON.
.
.
.
       *
           CALL WS-MQOPEN  WS-HCONN
                           MQOD
                           WS-OPTIONS
                           WS-HOBJ
                           WS-COMPCODE
                           WS-REASON.
```

z/OS 程式除錯

使用此資訊來瞭解對 TSO 和 CICS 程式進行除錯的相關資訊，以及對 CICS 追蹤的見解。

除錯 IBM MQ for z/OS 應用程式的主要輔助工具是每一個 API 呼叫所傳回的原因碼。 如需這些 (包括更正動作的構想) 的清單，請參閱:

- IBM MQ for z/OS 訊息、完成及原因碼 for IBM MQ for z/OS
- 訊息及原因碼，適用於所有其他 IBM MQ 平台

本主題也建議在特定環境中使用其他除錯工具。

除錯 TSO 程式

TSO 程式可以使用下列互動式除錯工具:

- TEST 工具
- VS COBOL II 互動式除錯工具
- 適用於 C 和 PL/I 程式的 INSPECT 互動式除錯工具

對 CICS 程式進行除錯

您可以使用「CICS 執行診斷機能 (CEDF)」以互動方式測試 CICS 程式，而無需修改程式或程式準備程序。

如需 EDF 的相關資訊，請參閱 *CICS Transaction Server for z/OS CICS Application Programming Guide*。

CICS 追蹤

您也可能會發現使用 CICS 追蹤控制交易 (CETR) 來控制 CICS 追蹤活動很有用。

如需 CETR 的相關資訊，請參閱 *CICS Transaction Server for z/OS CICS-Supplied Transactions* 手冊。

若要判斷 CICS 追蹤是否作用中，請使用 CKQC 畫面顯示連線狀態。 此畫面也會顯示追蹤號碼。

如果要解譯 CICS 追蹤項目，請參閱 第 869 頁的表 153。

這些值的 CICS 追蹤項目是 AP0 *xxx* (其中 *xxxx* 是啟用 CICS 配接卡時指定的追蹤號碼)。 CSQCTRUE 會發出 CSQCTEST 以外的所有追蹤登錄。 CSQCTEST 由 CSQCRST 和 CSQCDSP 發出。

表 *153: CICS* 配接器追蹤項目

名稱	說明	追蹤順序	追蹤資料
CSQCABNT	異常終止	在對 IBM MQ 發出 END_THREAD ABNORMAL 之前。 這是因為作業已結束，且應用程式可以執行隱含的取消。 在此情況下，END_THREAD 呼叫中包含 ROLLBACK 要求。	工作單元資訊。 在瞭解工作狀態時，您可以使用此資訊。 (例如，可以根據 DISPLAY THREAD 指令或 IBM MQ for z/OS 日誌列印公用程式所產生的輸出來驗證它。)
CSQCBACK	同步點取消	在對 IBM MQ for z/OS 發出 BACKOUT 之前。 這是因為來自應用程式的明確取消要求。	工作單元資訊。

表 153: CICS 配接器追蹤項目 (繼續)			
名稱	**說明**	**追蹤順序**	**追蹤資料**
CSQCCCRC	完成碼和原因碼	從 API 呼叫傳回失敗之後。	完成碼和原因碼。
CSQCCOMM	同步點確定	在對 IBM MQ for z/OS 發出 COMMIT 之前。 這可能是由於一段式確定要求或兩段式確定要求的第二階段。 要求是由於來自應用程式的明確同步點要求。	工作單元資訊。
CSQCEXER	執行解析	對 IBM MQ for z/OS 發出 EXECUTE_RESOLVE 之前。	發出 EXECUTE_RESOLVE 之工作單元的工作單元資訊。 這是重新同步處理程序中最後一個不確定的工作單元。
CSQCGETW	GET 等待	在發出 CICS 等待之前。	要等待的歐洲央行位址。
CSQCGMGD	GET 訊息資料	從 MQGET 順利傳回之後。	最多 40 個位元組的訊息資料。
CSQCGMGH	GET 訊息控點	在對 IBM MQ for z/OS 發出 MQGET 之前。	物件控點。
CSQCGMGI	取得訊息 ID	從 MQGET 順利傳回之後。	訊息的訊息 ID 和相關性 ID。
CSQCINDL	不確定清單	順利從第二個 INQUIRE_INDOUBT 傳回之後。	不確定的工作單元清單。
CSQCINDO	僅限 IBM 使用		
CSQCINDS	不確定清單大小	順利從第一個 INQUIRE_INDOUBT 傳回之後，且不確定清單不是空的。	清單的長度。 除以 64 會提供不確定的工作單元數目。
CSQCINQH	INQ 控點	在對 IBM MQ for z/OS 發出 MQINQ 之前。	物件控點。
CSQCLOSH	CLOSE 控點	在對 IBM MQ for z/OS 發出 MQCLOSE 之前。	物件控點。
CSQCLOST	遺失處置	在重新同步處理程序期間， CICS 會通知配接卡已重新啟動，因此沒有任何關於重新同步化工作單元的處置資訊可用。	要重新同步化之工作單元的 CICS 已知工作單元 ID。
CSQCNIND	處置未不確定	在重新同步化處理程序期間，CICS 會通知配接卡正在重新同步化的工作單元不應是不確定的 (亦即，可能它仍在執行中)。	要重新同步化之工作單元的 CICS 已知工作單元 ID。
CSQCNORT	正常終止	在對 IBM MQ for z/OS 發出 END_THREAD NORMAL 之前。 這是由於作業結束，因此應用程式可能執行隱含的同步點確定。 在此情況下， END_THREAD 呼叫中包含 COMMIT 要求。	工作單元資訊。
CSQCOPNH	OPEN 控點	從 MQOPEN 順利傳回之後。	物件控點。
CSQCOPNO	開啟物件	在對 IBM MQ for z/OS 發出 MQOPEN 之前。	物件名稱。
CSQCPMGD	PUT 訊息資料	在對 IBM MQ for z/OS 發出 MQPUT 之前。	最多 40 個位元組的訊息資料。

表 153: CICS 配接器追蹤項目 (繼續)			
名稱	**說明**	**追蹤順序**	**追蹤資料**
CSQCPMGH	PUT 訊息控點	在對 IBM MQ for z/OS 發出 MQPUT 之前。	物件控點。
CSQCPMGI	PUT 訊息 ID	從 IBM MQ for z/OS 順利完成 MQPUT 之後。	訊息的訊息 ID 和相關性 ID。
CSQCPREP	同步點準備	在兩段式確定處理的第一階段中對 IBM MQ for z/OS 發出 PREPARE 之前。 也可以從分散式佇列元件發出此呼叫作為 API 呼叫。	工作單元資訊。
CSQCP1MD	PUTONE 訊息資料	在向 IBM MQ for z/OS 發出 MQPUT1 之前。	最多 40 個位元組的訊息資料。
CSQCP1MI	PUTONE 訊息 ID	從 MQPUT1 順利傳回之後。	訊息的訊息 ID 和相關性 ID。
CSQCP1ON	PUTONE 物件名稱	在向 IBM MQ for z/OS 發出 MQPUT1 之前。	物件名稱。
CSQCRBAK	已解決取消	在對 IBM MQ for z/OS 發出 RESOLVE_ROLLBACK 之前。	工作單元資訊。
CSQCRCMT	已解決確定	在對 IBM MQ for z/OS 發出 RESOLVE_COMMIT 之前。	工作單元資訊。
CSQCRMIR	RMI 回應	從特定呼叫回到 CICS RMI (資源管理程式介面) 之前。	已架構的 RMI 回應值。 其意義取決於呼叫的類型。 這些值記錄在 *CICS Transaction Server for z/OS* 自訂作業手冊中。 若要判斷呼叫類型，請查看 CICS RMI 元件所產生的先前追蹤項目。
CSQCRSYN	重新同步	在針對作業啟動重新同步處理程序之前。	要重新同步化之工作單元的 CICS 已知工作單元 ID。
CSQCSETH	SET 控點	向 IBM MQ for z/OS 發出 MQSET 之前。	物件控點。
CSQCTASE	僅限 IBM 使用		
CSQCTEST	追蹤測試	在 EXEC CICS ENTER TRACE 呼叫中使用，以驗證使用者提供的追蹤號碼或連線的追蹤狀態。	沒有資料。
CSQCDCFF	僅限 IBM 使用		

處理程序化程式錯誤

此資訊說明與應用程式 MQI 呼叫相關聯的錯誤，當它進行呼叫時，或當它的訊息遞送至其最終目的地時。

可能的話，佇列管理程式會在發出 MQI 呼叫時立即傳回任何錯誤。 這些是 本端判斷的錯誤。

將訊息傳送至遠端佇列時，發出 MQI 呼叫時可能不會明顯發生錯誤。 在此情況下，識別錯誤的佇列管理程式會將另一則訊息傳送至原始程式來報告這些錯誤。 這些是 遠端判定錯誤。

本端判定的錯誤

本端判斷錯誤的相關資訊，包括: MQI 呼叫失敗、系統岔斷，以及包含不正確資料的訊息。

佇列管理程式可以立即報告的三個最常見錯誤原因如下:

- MQI 呼叫失敗; 例如，因為佇列已滿
- 岔斷執行應用程式所相依之系統的某個部分; 例如，佇列管理程式
- 包含無法順利處理之資料的訊息

如果您使用非同步放置機能，則不會立即報告錯誤。 使用 MQSTAT 呼叫來擷取先前非同步放置作業的狀態資訊。

MQI 呼叫失敗

佇列管理程式可以立即報告 MQI 呼叫編碼中的任何錯誤。 它使用一組預先定義的回覆碼來執行此動作。 這些會分成完成碼和原因碼。

為了顯示呼叫是否成功，當呼叫完成時，佇列管理程式會傳回 完成碼 。 有三個完成碼，指出呼叫成功、局部完成及失敗。 佇列管理程式也會傳回 原因碼 ，指出局部完成或呼叫失敗的原因。

每一個呼叫的完成及原因碼都會在 回覆碼中列出，並附有該呼叫的說明。 如需更詳細的資訊，包括更正動作的構想，請參閱:

- z/OS IBM MQ for z/OS 訊息、完成及原因碼 for IBM MQ for z/OS
- 訊息及原因碼 ，適用於所有其他 IBM MQ 平台

設計您的程式，以處理每一個呼叫可能產生的所有回覆碼。

System interruptions

如果應用程式所連接的佇列管理程式必須從系統失敗中回復，則您的應用程式可能不知道任何岔斷。 不過，您必須設計應用程式，以確保在發生這類岔斷時不會遺失您的資料。

您可以用來確保資料保持一致的方法，視執行佇列管理程式的平台而定:

z/OS
在 CICS 和 IMS 環境中，您可以在 CICS 或 IMS 所管理的工作單元內發出 MQPUT 和 MQGET 呼叫。 在批次環境中，您可以使用相同方式來發出 MQPUT 和 MQGET 呼叫，但必須使用下列指令來宣告同步點:

- IBM MQ for z/OS MQCMIT 和 MQBACK 呼叫 (請參閱 第 716 頁的『確定及取消工作單元』)，或
- 提供兩階段同步點支援的「 z/OS 交易管理及可回復的 Resource Manager 服務 (RRS)」。 RRS 可讓您在單一邏輯工作單元內同時更新 IBM MQ 及其他已啟用 RRS 的產品資源，例如 Db2 儲存程序資源。 如需 RRS 同步點支援的相關資訊，請參閱 第 719 頁的『交易管理和可回復的資源管理程式服務』。

IBM i
您可以在 IBM i 確定控制所管理的廣域工作單元內進行 MQPUT 及 MQGET 呼叫。 您可以使用原生 IBM i COMMIT 及 ROLLBACK 指令或特定語言指令來宣告同步點。 本端工作單元由 IBM MQ 使用 MQCMIT 和 MQBACK 呼叫進行管理。

AIX, Linux, and Windows 系統
在這些環境中，您可以使用一般方式進行 MQPUT 和 MQGET 呼叫，但必須使用 MQCMIT 和 MQBACK 呼叫來宣告同步點 (請參閱 第 716 頁的『確定及取消工作單元』)。 在 CICS 環境中， MQCMIT 和 MQBACK 指令已停用，因為您可以在 CICS 所管理的工作單元內發出 MQPUT 和 MQGET 呼叫。

使用持續訊息來承載您無法承受遺失的所有資料。 如果佇列管理程式必須從失敗中回復，則會在佇列上恢復持續訊息。 ALW 使用 IBM MQ on AIX, Linux, and Windows，應用程式內的 MQGET 或 MQPUT 呼叫會在填入所有日誌檔時失敗，並顯示訊息 MQRC_RESOURCE_PROBLEM。 如需 AIX, Linux, and Windows 上日誌檔的相關資訊，請參閱 管理 IBM MQ。 z/OS 若為 z/OS ，請參閱 在 z/OS 上規劃。

如果在應用程式執行時由操作員停止佇列管理程式，則通常會使用靜止選項。 佇列管理程式會進入靜止狀態，在此狀態下應用程式可以繼續執行工作，但它們必須在方便時立即終止。 小型快速應用程式可能可以忽略靜止狀態，並繼續進行，直到它們正常終止為止。 執行較長的應用程式 (或等待訊息送達的應用程式) 應該在使用 MQOPEN、MQPUT、 MQPUT1 及 MQGET 呼叫時使用 如果靜止則失敗 選項。 這些選項表示當佇列管理程式靜止時，呼叫會失敗，但應用程式仍可能有時間透過發出忽略靜止狀態的呼叫來完全終止。 這類應用程式也可以確定或取消他們所做的變更，然後終止。

如果強制停止佇列管理程式 (亦即，在不靜止的情況下停止)，應用程式會在發出 MQI 呼叫時收到 MQRC_CONNECTION_BROKEN 原因碼。 結束應用程式，或者在 IBM i IBM MQ for IBM i、 AIX, Linux, and Windows 系統上，發出 MQDISC 呼叫。

包含不正確資料的訊息

當您在應用程式中使用工作單元時，如果程式無法順利處理它從佇列擷取的訊息，則會取消 MQGET 呼叫。

佇列管理程式會維護發生次數的計數 (在訊息描述子的 *BackoutCount* 欄位中)。 它會在每一個受影響訊息的描述子中維護此計數。 此計數可以提供關於應用程式效率的有價值資訊。 取消計數在一段時間內不斷增加的訊息會被反覆拒絕; 請設計您的應用程式，讓它分析原因並相應地處理這類訊息。

z/OS 在 IBM MQ for z/OS 上，若要讓取消計數在佇列管理程式重新啟動之後仍然存在，請將 **HardenGetBackout** 屬性設為 MQQA_BACKOUT_HARDENED; 否則，如果佇列管理程式必須重新啟動，它不會維護每一則訊息的精確取消計數。 以此方式設定屬性會增加額外處理的懲罰。

在 IBM MQ for IBM i IBM i、 AIX, Linux, and Windows 系統上，取消計數一律在佇列管理程式重新啟動之後仍然存在。

z/OS 此外，在 IBM MQ for z/OS 上，當您從工作單元內的佇列中移除訊息時，您可以標示一則訊息，以便在應用程式取消工作單元時無法再次使用該訊息。 標示的訊息會被視為已在新工作單元下擷取。 您可以使用 MQGMO_MARK_SKIP_BACKOUT 選項來標示要跳過取消的訊息(在 MQGMO 結構中) 當您使用 MQGET 呼叫時。 如需此技術的相關資訊，請參閱 第 668 頁的『跳過取消』。

使用報告訊息來判斷問題

當您發出 MQI 呼叫時，遠端佇列管理程式無法報告錯誤 (例如無法將訊息放置在佇列上)，但它可以傳送報告訊息給您，以指出它如何處理您的訊息。

在應用程式內，您可以建立 (MQPUT) 報告訊息，以及選取接收訊息的選項 (在此情況下，會由另一個應用程式或佇列管理程式傳送訊息)。

建立報告訊息

報告訊息可讓應用程式告知另一個應用程式無法處理已傳送的訊息。

不過，一開始必須分析 *Report* 欄位，以判斷傳送訊息的應用程式是否有興趣收到任何問題的通知。 在判定需要報告訊息之後，您必須決定:

- 您是要包含整個原始訊息、僅包含前 100 個位元組的資料，還是不包含任何原始訊息。
- 如何處理原始訊息。 您可以捨棄它或讓它進入無法傳送郵件的佇列。
- 是否也需要 *MsgId* 和 *CorrelId* 欄位的內容。

使用 *Feedback* 欄位來指出產生報告訊息的原因。 將報告訊息放在應用程式的回覆目的地佇列中。 如需進一步資訊，請參閱 意見。

要求及接收 (MQGET) 報告訊息

當您將訊息傳送至另一個應用程式時，除非您完成 *Report* 欄位以指出您需要的意見，否則不會通知您有任何問題。 如需可用的選項，請參閱 報告欄位的結構。

佇列管理程式一律會將報告訊息放在應用程式的回覆目的地佇列中，建議您自己的應用程式執行相同的動作。 當您使用報告訊息機能時，請在訊息的訊息描述子中指定回覆目的地佇列的名稱; 否則 MQPUT 呼叫會失敗。

您的應用程式必須包含監視回覆目的地佇列的程序，並處理到達它的任何訊息。 請記住，報告訊息可以包含所有原始訊息、原始訊息的前 100 個位元組，或沒有原始訊息。

佇列管理程式會設定報告訊息的 *Feedback* 欄位，以指出錯誤的原因; 例如，目標佇列不存在。 您的程式也應該這麼做。

如需報告訊息的相關資訊，請參閱 第 17 頁的『報告訊息』。

遠端判定的錯誤

當您將訊息傳送至遠端佇列時，即使本端佇列管理程式已處理 MQI 呼叫而未發現錯誤，其他因素也會影響遠端佇列管理程式處理訊息的方式。

例如，您設為目標的佇列可能已滿，甚至可能不存在。 如果您的訊息必須由目標佇列路徑上的其他中間佇列管理程式處理，則其中任何可能發現錯誤。

遞送訊息時發生問題

當 MQPUT 呼叫失敗時，您可以嘗試重新將訊息放置在佇列上，將訊息傳回給傳送端，或將訊息放置在無法傳送郵件的佇列上。

每一個選項都有其優點，但如果 MQPUT 失敗的原因是目的地佇列已滿，您可能不想再次嘗試放置訊息。 在此實例中，將它置於無法傳送郵件的佇列可讓您稍後將它遞送至正確的目的地佇列。

重試訊息遞送

在將訊息放入無法傳送郵件的佇列之前，如果已針對通道設定屬性 *MsgRetryCount* 及 *MsgRetryInterval* ，或如果有重試結束程式可供使用 (在通道屬性 *MsgRetryExitId* 欄位中保留其名稱) ，則遠端佇列管理程式會再次嘗試將訊息放入佇列。

如果 *MsgRetryExitId* 欄位空白，則會使用屬性 *MsgRetryCount* 和 *MsgRetryInterval* 中的值。

如果 *MsgRetryExitId* 欄位不是空白，則會執行此名稱的結束程式。 如需使用您自己的結束程式的相關資訊，請參閱 第 805 頁的『傳訊通道的通道結束程式』。

將訊息傳回給寄件者

您可以要求產生報告訊息以併入所有原始訊息，以將訊息傳回給寄件者。

如需報告訊息選項的詳細資料，請參閱 第 17 頁的『報告訊息』 。

使用無法傳送的郵件 (無法遞送的訊息) 佇列

當佇列管理程式無法遞送訊息時，它會嘗試將訊息放置在其無法傳送郵件的佇列上。 應該在安裝佇列管理程式時定義此佇列。

您的程式可以使用無法傳送郵件的佇列，與佇列管理程式使用它的方式相同。 您可以開啟佇列管理程式物件 (使用 MQOPEN 呼叫) ，並查詢 **DeadLetterQName** 屬性 (使用 MQINQ 呼叫) ，來尋找無法傳送郵件的佇列名稱。

當佇列管理程式將訊息放置在此佇列上時，它會將標頭新增至訊息，其格式由無法傳送郵件的標頭 (MQDLH) 結構說明; 請參閱 MQDLH-無法傳送郵件的標頭。 此標頭包括目標佇列的名稱，以及將訊息放置在無法傳送郵件的佇列上的原因。 在將訊息放入預期的佇列之前，必須先移除它，且必須先解決此問題。 此外，佇列管理程式會變更訊息描述子 (MQMD) 的 *Format* 欄位，以指出訊息包含 MQDLH 結構。

MQDLH 結構

建議您將 MQDLH 結構新增至您放置在無法傳送郵件的佇列上的所有訊息; 不過，如果您想要使用特定 IBM MQ 產品所提供的無法傳送郵件處理程式，則必須將 MQDLH 結構新增至訊息。

將標頭新增至訊息可能會使無法傳送郵件的佇列中的訊息太長，因此請務必確定您的訊息比無法傳送郵件的佇列所容許的大小上限還短，至少要加上 MQ_MSG_HEADER_LENGTH 常數的值。 佇列上容許的訊息大小上限由佇列的 **MaxMsgLength** 屬性值決定。 對於無法傳送郵件的佇列，請確定此屬性設為佇列管理程式所容許的上限。 如果您的應用程式無法遞送訊息，且訊息太長而無法放入無法傳送郵件的佇列，請遵循 MQDLH 結構說明中提供的建議。

請確定已監視無法傳送郵件的佇列，且已處理任何到達它的訊息。 無法傳送郵件的佇列處理程式以批次公用程式執行，可用來對無法傳送郵件的佇列上選取的訊息執行各種動作。 如需詳細資料，請參閱第 875 頁的『無法傳送郵件的佇列處理』。

如果需要資料轉換，當您在 MQGET 呼叫上使用 MQGMO_CONVERT 選項時，佇列管理程式會轉換標頭資訊。 如果放置訊息的處理程序是 MCA ，則標頭後面會接著原始訊息的所有文字。

如果放置在無法傳送郵件的佇列上的訊息對這個佇列來說太長，則可能會被截斷。 此狀況的可能指示是無法傳送郵件的佇列上的訊息長度與佇列的 **MaxMsgLength** 屬性值相同。

無法傳送郵件的佇列處理

此資訊包含使用無法傳送郵件的佇列處理時的一般用途程式設計介面資訊。

無法傳送郵件的佇列處理取決於本端系統需求，但在您起草規格時請考量下列事項:

- 訊息可以識別為具有無法傳送郵件的佇列標頭，因為 MQMD 中的格式欄位值為 MQFMT_DEAD_LETTER_HEADER。
- 在使用 CICS 的 IBM MQ for z/OS 上，如果 MCA 將此訊息放置到無法傳送郵件的佇列，則 *PutApplType* 欄位是 MQAT_CICS ，而 *PutApplName* 欄位是 CICS 系統的 *ApplId* ，後面接著 MCA 的交易名稱。
- 將訊息遞送至無法傳送郵件的佇列的原因包含在無法傳送郵件的佇列標頭的 *Reason* 欄位中。
- 無法傳送郵件的佇列標頭包含目的地佇列名稱及佇列管理程式名稱的詳細資料。
- 無法傳送郵件的佇列標頭包含在將訊息放入目的地佇列之前必須在訊息描述子中恢復的欄位。 它們是:
  1. *Encoding*
  2. *CodedCharSetId*
  3. *Format*
- 訊息描述子與原始應用程式的 PUT 相同，但所顯示的三個欄位 (「編碼」、 CodedCharSetId 及「格式」) 除外。

您的無法傳送郵件的佇列應用程式必須執行下列一或多個動作:

- 檢查 *Reason* 欄位。 由於下列原因， MCA 可能已放置訊息:
  - 訊息長度超過通道的訊息大小上限

    原因是 MQRC_MSG_TOO_BIG_FOR_CHANNEL
  - 無法將訊息放入其目的地佇列

    原因是 MQPUT 作業可以傳回任何 MQRC_ * 原因碼
  - 使用者結束程式已要求此動作

    原因碼是由使用者結束程式或預設 MQRC_SUPPRESSED_BY_EXIT 所提供
- 如果可能的話，請嘗試將訊息轉遞至其預期的目的地。
- 當確定轉移的原因時，在捨棄之前，請先保留訊息一段特定時間，但無法立即更正。
- 請提供指示給管理者，以更正已決定這些問題的問題。
- 捨棄已毀損或無法處理的訊息。

有兩種方法可以處理您從無法傳送郵件的佇列中回復的訊息:

1. 如果訊息是針對本端佇列:
   - 執行擷取應用程式資料所需的任何程式碼轉換
   - 如果這是本端函數，則對該資料執行程式碼轉換
   - 將產生的訊息放置在本端佇列上，並還原訊息描述子的所有詳細資料
2. 如果訊息是針對遠端佇列，請將訊息放置在佇列上。

如需如何在分散式佇列環境中處理未遞送訊息的相關資訊，請參閱 無法遞送訊息時會發生什麼情況?。

多重播送程式設計

使用此資訊來瞭解「 IBM MQ 多重播送」程式設計作業，例如連接至佇列管理程式及異常狀況報告。

IBM MQ Multicast 是設計成盡可能對使用者透通，但仍與現有應用程式相容。 定義 COMMINFO 物件並設定 TOPIC 物件的 **MCAST** 及 **COMMINFO** 參數，表示現有 IBM MQ 應用程式不需要大量重新編寫即可使用多重播送。 不過，可能有一些限制 (如需相關資訊，請參閱 第 876 頁的『多重播送及 MQI』)，以及一些需要考量的安全問題 (如需相關資訊，請參閱 多重播送安全)。

多重播送及 MQI

使用此資訊來瞭解主要「訊息佇列介面 (MQI)」概念，以及它們與 IBM MQ Multicast 的關係。

多重播送訂閱是不可延續的; 因為沒有涉及實體佇列，所以無法儲存可延續訂閱所建立的離線訊息。

在應用程式訂閱多重播送主題之後，會提供它可以使用的物件控點或 MQGET，就像它是佇列的控點一樣。這表示只支援受管理多重播送訂閱 (使用 MQSO_MANAGED 建立的訂閱)，亦即，無法在佇列上建立訂閱及「點」訊息。這表示必須從訂閱呼叫所傳回的物件控點中耗用訊息。在用戶端上，訊息會儲存在訊息緩衝區中，直到用戶端使用為止; 如需相關資訊，請參閱 用戶端配置檔的 MessageBuffer 段落。如果用戶端未跟上發佈速率，則會依需要捨棄訊息，並先捨棄最舊的訊息。

通常是透過設定 TOPIC 物件的 MCAST 屬性來指定應用程式是否使用「多重播送」的管理決策。如果發佈應用程式必須確保未使用多重播送，則可以使用 MQOO_NO_MULTICAST 選項。同樣地，訂閱應用程式可以確保使用 MQSO_NO_MULTICAST 選項訂閱時不會使用多重播送。

IBM MQ Multicast 支援使用訊息選取器。應用程式會使用選取器，只將其興趣登錄在那些具有滿足選取字串所代表之 SQL92 查詢的內容的訊息中。如需訊息選取器的相關資訊，請參閱 第 26 頁的『選取器』。

下表列出所有主要 MQI 概念，以及它們與「多重播送」的關係:

表 *154: MQI* 概念及其與多重播送的關係

MQI 概念	**嘗試使用多重播送時的動作**	**原因碼**
放置零長度訊息	已拒絕	2005 (07D5) (RC2005) :MQRC_BUFFER_LENGTH_ERROR
分組	已拒絕	2046 (07FE) (RC2046) :MQRC_OPTIONS_ERROR
分區段	已拒絕	2443 (098B) (RC2443) :MQRC_SEGMENTATION_NOT_ALLOWED
分送清單	已拒絕	2154 (086A) (RC2154): MQRC_RECS_PRESENT_ERROR
MQINQ	拒絕主題控點: 不支援主題的 MQINQ 和 MQSET。	2038 (07F6) (RC2038) :MQRC_NOT_OPEN_FOR_INQUIRE
MQINQ	已接受受管理控點。只能查詢 **現行深度**。	• 如果值是「現行深度」，則沒有適用的原因碼。 • 如果值不是「現行深度」，則原因碼為 2067 (0813) (RC2067) :MQRC_SELECTOR_ERROR。
MQSET	拒絕所有控點。	2040 (07F8) (RC2040) :MQRC_NOT_OPEN_FOR_SET
交易 (XA 或非 XA)	已拒絕	2072 (0818) (RC2072) :XX_ENCODE_CASE_ONE mqrc_syncpoint_not_available
訊息瀏覽	已拒絕	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE
鎖定訊息	已拒絕	2046 (07FE) (RC2046) :MQRC_OPTIONS_ERROR
以標記瀏覽	已拒絕	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE
傳遞環境定義	已拒絕	2046 (07FE) (RC2046) :MQRC_OPTIONS_ERROR
MQPUT1	已拒絕。嘗試將 MQPUT1 設為「僅限多重播送」主題無效。	2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY

表 *154: MQI* 概念及其與多重播送的關係 *(*繼續*)*		
MQI 概念	**嘗試使用多重播送時的動作**	**原因碼**
可延續訂閱	如果主題標示為「僅限多重播送」，則會拒絕，否則會建立非多重播送訂閱。	2436 (0984) (RC2436) :MQRC_DURABILITY_NOT_ALLOWED
TopicString > 255	已拒絕。 如果主題字串超過 255 個字元，則用戶端會拒絕它。	2425 (0979) (RC2425) :MQRC_TOPIC_STRING_ERROR
已建立非受管理訂閱	如果主題標示為「僅限多重播送」，則會拒絕，否則會建立非多重播送訂閱。	2046 (07FE) (RC2046) :MQRC_OPTIONS_ERROR
MQPMO_NOT_OWN_SUBS	已拒絕	2046 (07FE) (RC2046) :MQRC_OPTIONS_ERROR

下列項目會展開前一個表格中的部分 MQI 概念，並提供不在表格中之部分 MQI 概念的相關資訊:

訊息持續性
對於不可延續的多重播送訂閱者，來自發佈者的持續訊息會以無法復原的方式遞送。

訊息截斷
支援訊息截斷，這表示應用程式可以執行下列動作:

1. 發出 MQGET。
2. 取得 MQRC_TRUNCATED_MSG_FAILED。
3. 配置較大的緩衝區。
4. 重新發出 MQGET 以擷取訊息。

訂閱期限
不支援訂閱期限。 系統不處理任何設定期限的嘗試。

多重播送的高可用性

使用此資訊來瞭解 IBM MQ Multicast 連續對等式作業; 雖然 IBM MQ 連接至 IBM MQ 佇列管理程式，但訊息不會流經該佇列管理程式。

雖然必須建立與佇列管理程式的連線，才能對 MQOPEN 或 MQSUB 進行多重播送主題物件，但訊息本身不會流經佇列管理程式。 因此，在對多重播送主題物件完成 MQOPEN 或 MQSUB 之後，即使佇列管理程式的連線已中斷，仍可以繼續傳輸多重播送訊息。 有兩種作業模式:

建立與佇列管理程式的正常連線
當佇列管理程式的連線存在時，可以進行多重播送通訊。 如果連線失敗，則會套用一般 MQI 規則，例如 :MQPUT 至多重播送物件控點會傳回 2009 (07D9) (RC2009) :MQRC_CONNECTION_毀損。

重新連接用戶端與佇列管理程式的連線
即使在重新連線週期期間，也可以進行多重播送通訊。 這表示即使與佇列管理程式的連線已中斷，放置及耗用多重播送訊息也不會受到影響。 用戶端會嘗試重新連接至佇列管理程式，如果重新連線失敗，連線控點會變成中斷，且所有 MQI 呼叫 (包括多重播送呼叫) 都會失敗。 如需相關資訊，請參閱: 自動用戶端重新連線

如果有任何應用程式明確發出 MQDISC ，則會關閉所有多重播送訂閱及物件控點。

多重播送連續對等作業

用戶端之間對等式通訊的優點之一是訊息不需要流經佇列管理程式; 因此，如果與佇列管理程式的連線中斷，則訊息傳送會繼續。 下列限制適用於此模式的連續訊息需求:

- 必須使用其中一個 MQCNO_RECONNECT_ * 選項來建立連線，以進行連續作業。 此處理程序表示雖然通訊階段作業可能已中斷，但實際連線控點並未中斷，而是處於重新連接狀態。 如果重新連線失敗，現在會岔斷連線控點，這會阻止所有進一步的 MQI 呼叫。
- 此模式只支援 MQPUT、MQGET、MQINQ 及 Async Consume。 任何 MQOPEN、MQCLOSE 或 MQDISC 動詞都需要重新連線至佇列管理程式才能完成。
- 傳送至佇列管理程式的狀態會停止; 因此佇列管理程式中的任何狀態可能已過時或遺漏。 這表示用戶端可能正在傳送及接收訊息，且佇列管理程式上沒有已知的狀態。 如需相關資訊，請參閱: 多重播送應用程式監視

MQI for 多重播送傳訊中的資料轉換

使用此資訊來瞭解 IBM MQ Multicast 傳訊的資料轉換如何運作。

IBM MQ Multicast 是一種共用的無連線通訊協定，因此無法讓每一個用戶端提出特定的要求來進行資料轉換。 每個訂閱相同多重播送串流的用戶端都會接收相同的二進位資料; 因此，如果需要 IBM MQ 資料轉換，則會在每個用戶端本端執行轉換。

在 IBM MQ Multicast 資料流量的用戶端上轉換資料。 如果指定 **MQGMO_CONVERT** 選項，則會依要求執行資料轉換。 使用者定義格式需要在用戶端上安裝資料轉換結束程式; 如需用戶端和伺服器套件中現在有哪些媒體庫的相關資訊，請參閱 第 823 頁的『寫入資料-轉換結束程式』 。

如需管理資料轉換的相關資訊，請參閱 啟用多重播送傳訊的資料轉換。

如需資料轉換的相關資訊，請參閱 資料轉換。

如需資料轉換結束程式及 ClientExitPath 的相關資訊，請參閱 用戶端配置檔的 ClientExit 路徑段落。

多重播送異常狀況報告

使用此資訊來瞭解 IBM MQ 多重播送事件處理程式及報告 IBM MQ 多重播送異常狀況。

IBM MQ Multicast 可呼叫事件處理程式來報告使用標準 IBM MQ 事件處理程式機制所報告的多重播送事件，以協助進行問題判斷。

個別「多重播送」事件可能會導致呼叫多個 IBM MQ 事件，因為可能有多個 MQHCONN 連線控點使用相同的多重播送轉送器或接收端。 不過，每一個多重播送異常狀況都會導致每個 IBM MQ 連線只呼叫一個事件處理程式。

IBM MQ MQCBDO_EVENT_CALL 常數可讓應用程式登錄回呼以僅接收 IBM MQ 事件，而 MQCBDO_MC_EVENT_CALL 可讓應用程式登錄回呼以僅接收多重播送事件。 如果同時使用這兩個常數，則會接收這兩種類型的事件。

要求多重播送事件

IBM MQ 多重播送事件在 cbd.Options 欄位中使用 MQCBDO_MC_EVENT_CALL 常數。 下列範例示範如何要求多重播送事件:

```
cbd.CallbackType     = MQCBT_EVENT_HANDLER;
cbd.Options          = MQCBDO_MC_EVENT_CALL;
cbd.CallbackFunction = EventHandler;
MQCB(Hcon,MQOP_REGISTER,&cbd,MQHO_UNUSABLE_HOBJ,NULL,NULL,&CompCode,&Reason);
```

當針對 cbd.Options 欄位指定 MQCBDO_MC_EVENT_CALL 選項時，事件處理程式只會傳送 IBM MQ 多重播送事件，而不是連線層次事件。 若要要求將這兩種類型的事件傳送至事件處理程式，應用程式必須在 cbd.Options 欄位中指定 MQCBDO_EVENT_CALL 常數，以及 MQCBDO_MC_EVENT_CALL 常數，如下列範例所示:

```
cbd.CallbackType     = MQCBT_EVENT_HANDLER;
```

```
cbd.Options          = MQCBDO_EVENT_CALL | MQCBDO_MC_EVENT_CALL
cbd.CallbackFunction = EventHandler;
MQCB(Hcon,MQOP_REGISTER,&cbd,MQHO_UNUSABLE_HOBJ,NULL,NULL,&CompCode,&Reason);
```

如果這兩個常數都沒有使用，則只會將連線層次事件傳送至事件處理程式。

如需 Options 欄位值的相關資訊，請參閱 選項 (MQLONG)。

多重播送事件格式

IBM MQ 多重播送異常狀況包括回呼函數的 **Buffer** 參數中傳回的部分支援資訊。 **Buffer** 指標指向指標陣列， MQCBC.DataLength 欄位指定陣列的大小 (以位元組為單位)。 陣列的第一個元素一律指向事件的簡短文字說明。 視事件類型而定，可能會提供更多參數。 下表列出異常狀況:

表 *155:* 多重播送事件碼說明

事件碼	**說明**	**其他資料**
MQMCEV_PACKET_LOSS	無法復原的封包流失	遺失封包數
MQMCEV_HEARTBEAT_TIMEOUT	長時間缺少活動訊號控制封包	N/A
MQMCHEV_VERSION_CONFLICT	接收較新的通訊協定版本封包	N/A
MQMCEV_RELIABILITY	發射機和接收機的不同可靠性模式	N/A
MQMCEV_CLOSED_TRANS	由 1 個來源關閉主題傳輸	N/A
MQMCEV_STREAM_ERROR	在串流上偵測到錯誤	N/A
MQMCEV_NEW_SOURCE	新的來源開始對主題進行傳輸	來源結構
MQMCEV_RECEIVE_QUEUE_修整	由於時間或空間到期而從 PacketQ 移除的封包數	修整的封包數
MQMCEV_PACKET_LOSS_NACK_EXPIRE	由於 NACK 到期而無法復原的封包流失	遺失封包數
已超出 MQMCEV_ACK_RETRIES_EXCEEDED	超出 **max_ack_retries** 之後從歷程中移除的封包數	移除的封包數
MQMCEV_STREAM_SUSPEND_NACK	已在此主題接受的串流上暫停 NACK	暫停串流 ID 暫停串流的時間 (毫秒)
MQMCEV_STREAM_RESUME_NACK	在串流中暫停 NACK 之後已回復	串流 ID
MQ 事件 _ 串流 _ 已開除	由於驅逐要求，已拒絕此主題所接受的串流	串流 ID
MQMCEV_FIRST_MESSAGE	來自來源的第一個訊息	訊息號碼
MQMCEV_LATE_JOIN_FAILURE	無法啟動延遲結合階段作業	N/A
MQMCEV_MESSAGE_LOSS	無法復原的訊息遺失	遺失訊息數
MQMCEV_SEND_PACKET_FAILURE	多重播送轉送器無法傳送多重播送封包	N/A
MQMCEV_REPAIR_DELAY	多重播送接收端未收到未執行 NAK 的修復封包	N/A
MQMCEV_MEMORY_ALERT_ON	接收端接收緩衝區已填滿	緩衝池使用率百分比
MQMCEV_MEMORY_ALERT_OFF	接收端接收緩衝區已關閉至正常	緩衝池使用率百分比

表 155: 多重播送事件碼說明 (繼續)		
事件碼	**說明**	**其他資料**
MQMCEV_NACK_ALERT_ON	接收端修復封包要求率達到高臨界值	現行修復要求速率 (以每秒封包數為單位)
MQMCEV_NACK_ALERT_OFF	接收端修復封包要求速率已降低至正常	現行修復要求速率 (以每秒封包數為單位)
MQMCEV_REPAIR_ALERT_ON	轉送器修復封包傳送速率達到高臨界值	N/A
MQMCEV_REPAIR_ALERT_OFF	轉送器修復封包傳送速率已降低至正常	N/A
MQMCEV_SHM_DEST_UNUSABLE	偵測到轉送器主題目的地所使用的「共用記憶體」區域無法使用	N/A
MQMCEV_SHM_PORT_UNUSABLE	偵測到接收端實例使用的「共用記憶體」埠無法使用	N/A
MQMCEV_CCT_GETTIME_FAILED	從「叢集協調時間」取得時間失敗	N/A
MQMCEV_DEST_INTERFACE_FAILURE	轉送器主題目的地使用的網路介面失敗，備份網路介面無法使用	
MQMCEV_DEST_INTERFACE_FAILOVER	轉送器主題目的地所使用的網路介面失敗，已順利完成失效接手至另一個介面	
MQMCEV_PORT_INTERFACE-FAILURE	接收端 rmmPort 使用的網路介面失敗，備份網路介面無法使用 (或也失敗)	RMM 配置
MQMCEV_PORT_INTERFACE_FAILOVER	接收端 rmmPort 使用的網路介面失敗，已順利失效接手至另一個介面	RMM 配置

以 C 撰寫程式碼

在 C 中編碼 IBM MQ 程式時，請注意下列各節中的資訊。

MQI 呼叫的參數

僅輸入 且類型為 MQHCONN、MQHOBJ、MQHMSG 或 MQLONG 的參數會依值傳遞; 對於所有其他參數，參數的 位址 會依值傳遞。

並非每次呼叫函數時都需要指定位址所傳遞的所有參數。 如果不需要特定參數，則可以將空值指標指定為函數呼叫上的參數，以取代參數資料的位址。 可以在呼叫說明中識別的參數。

未傳回任何參數作為函數的值; 在 C 術語中，這表示所有函數都會傳回 void。

函數的屬性由 MQENTRY 巨集變數定義; 此巨集變數的值視環境而定。

具有未定義資料類型的參數

MQGET、MQPUT 及 MQPUT1 函數各有一個具有未定義資料類型的 **Buffer** 參數。 此參數用來傳送及接收應用程式的訊息資料。

此排序的參數在 C 範例中顯示為 MQBYTE 的陣列。 您可以用這種方式來宣告參數，但通常更方便將它們宣告為訊息中說明資料佈置的結構。 函數參數宣告為對 void 的指標，因此任何資料的位址都可以指定為函數呼叫上的參數。

資料類型

所有資料類型都是使用 typedef 陳述式來定義。

對於每一種資料類型，也會定義對應的指標資料類型。 指標資料類型的名稱是基本或結構資料類型的名稱，字首為字母 P 以表示指標。 指標的屬性由 MQPOINTER 巨集變數定義; 此巨集變數的值視環境而定。 下列程式碼說明如何宣告指標資料類型:

```
#define MQPOINTER                   /* depends on environment */
...
typedef MQLONG  MQPOINTER PMQLONG;  /* pointer to MQLONG */
typedef MQMD   MQPOINTER PMQMD;     /* pointer to MQMD */
```

操作二進位字串

二進位資料的字串宣告為其中一個 MQBYTEn 資料類型。

每當您複製、比較或設定此類型的欄位時，請使用 C 函數 memcpy、 memcmp 或 memset:

```
#include <string.h>
#include "cmqc.h"

MQMD MyMsgDesc;

memcpy(MyMsgDesc.MsgId,           /* set "MsgId" field to nulls    */
       MQMI_NONE,                 /* ...using named constant       */
       sizeof(MyMsgDesc.MsgId));

memset(MyMsgDesc.CorrelId,        /* set "CorrelId" field to nulls */
       0x00,                      /* ...using a different method   */
       sizeof(MQBYTE24));
```

請勿使用字串函數 strcpy、strcmp、strncpy 或 strncmp ，因為這些無法正確使用宣告為 MQBYTE24 的資料。

操作字串

當佇列管理程式將字元資料傳回應用程式時，佇列管理程式一律以空白填補字元資料至欄位的定義長度。 佇列管理程式不會傳回以空值結尾的字串，但您可以在輸入中使用它們。 因此，在複製、比較或連結這類字串時，請使用字串函數 strncpy、strncmp 或 strncat。

請勿使用需要以空值 (strcpy、strcmp 及 strcat) 來終止字串的字串函數。 此外，請勿使用函數 strlen 來決定字串的長度; 請改用 sizeof 函數來決定欄位的長度。

結構的起始值

The include file <cmqc.h> defines various macro variables that you can use to provide initial values for the structures when declaring instances of those structures. 這些巨集變數的名稱格式為 MQxxx_DEFAULT , 其中 MQxxx 代表結構的名稱。 請如下所示使用它們:

```
MQMD   MyMsgDesc = {MQMD_DEFAULT};
MQPMO  MyPutOpts = {MQPMO_DEFAULT};
```

對於部分字元欄位， MQI 定義有效的特定值 (例如，適用於 MQMD 中的 *StrucId* 欄位或 *Format* 欄位)。 對於每一個有效值，會提供兩個巨集變數:

- 一個巨集變數將值定義為長度 (不包括隱含空值) 完全符合欄位定義長度的字串。 在下列範例中，符號 ¬ 代表單一空白字元:

  ```
  #define MQMD_STRUC_ID "MD¬¬"
  #define MQFMT_STRING "MQSTR¬¬¬"
  ```

  將此表單與 memcpy 及 memcmp 函數搭配使用。
- 另一個巨集變數將值定義為字元陣列; 此巨集變數的名稱是字串形式的名稱，以 _ARRAY 為字尾。 例如:

  ```
  #define MQMD_STRUC_ID_ARRAY 'M','D','¬','¬'
  #define MQFMT_STRING_ARRAY 'M','Q','S','T','R','¬','¬','¬'
  ```

  使用此表單，以不同於 MQMD_DEFAULT 巨集變數所提供的值來宣告結構實例時，起始設定欄位。

動態結構的起始值

當需要可變數目的結構實例時，通常會在使用 calloc 或 malloc 函數動態取得的主儲存體中建立實例。

若要起始設定此類結構中的欄位，建議使用下列技術:

1. 使用適當的 MQxxx_DEFAULT 巨集變數來宣告結構的實例，以起始設定結構。 此實例會變成其他實例的模型 :

   ```
   MQMD ModelMsgDesc = {MQMD_DEFAULT};
                                    /* declare model instance */
   ```

   在宣告上撰寫靜態或自動關鍵字，以視需要提供模型實例靜態或動態生命期限。
2. 使用 calloc 或 malloc 函數來取得結構動態實例的儲存體:

   ```
   PMQMD InstancePtr;
   InstancePtr = malloc(sizeof(MQMD));
                                    /* get storage for dynamic instance */
   ```
3. 使用 memcpy 函數將模型實例複製到動態實例:

   ```
   memcpy(InstancePtr,&ModelMsgDesc,sizeof(MQMD));
                                    /* initialize dynamic instance */
   ```

從 C++ 使用

對於 C++ 程式設計語言，標頭檔包含下列其他陳述式，只有在使用 C++ 編譯器時才會包含這些陳述式:

```
#ifdef __cplusplus
  extern "C" {
#endif

/* rest of header file */
```

```
#ifdef __cplusplus
  }
#endif
```

Windows 在 Visual Basic 中撰寫程式碼

在 Microsoft Visual Basic 中編碼 IBM MQ 程式時要考量的資訊。 Visual Basic 僅在 Windows 上受支援。

註:

Stabilized 從 IBM WebSphere MQ 7.0 開始，在 .NET 環境之外， Visual Basic (VB) 的支援已穩定在 IBM WebSphere MQ 6.0 層次。 新增至 IBM WebSphere MQ 7.0 或更新版本的大部分新功能無法用於 VB 應用程式。 如果您是在 VB.NET 中進行程式設計，請使用 .NET 的 IBM MQ 類別。 如需相關資訊，請參閱開發 .NET 應用程式。

Deprecated 從 IBM MQ 9.0 開始， Microsoft Visual Basic 6.0 的支援已淘汰。 .NET 的 IBM MQ 類別是建議的取代技術。

若要避免非預期地轉換在 Visual Basic 與 IBM MQ 之間傳遞的二進位資料，請使用 MQBYTE 定義而非 MQSTRING。 CMQB.BAS 定義數個新的 MQBYTE 類型，相當於 C 位元組定義，並在 IBM MQ 結構內使用這些類型。 例如，針對 MQMD (訊息描述子) 結構， MsgId (訊息 ID) 定義為 MQBYTE24。

Visual Basic 沒有指標資料類型，因此對其他 IBM MQ 資料結構的參照是依偏移而非指標。 宣告由兩個元件結構組成的複合結構，並在呼叫時指定複合結構。 Visual Basic 的 IBM MQ 支援提供 MQCONNXAny 呼叫可讓您這樣做，並容許用戶端應用程式在用戶端連線上指定通道內容。 它接受非類型化結構 (MQCNOCD) 來取代一般 MQCNO 結構。

MQCNOCD 結構是由 MQCNO 和 MQCD 組成的複合結構。 此結構在結束程式標頭檔 CMQXB 中宣告。 使用常式 MQCNOCD_DEFAULTS 來起始設定 MQCNOCD 結構。 提供進行 MQCONNX 呼叫的範例 (amqscnxb.vbp)。

MQCONNXAny 具有與 MQCONNX 相同的參數，但 **ConnectOpts** 參數宣告為「任何」資料類型而非 MQCNO 資料類型。 這可讓函數接受 MQCNO 或 MQCNOCD 結構。 此函數在主要標頭檔 CMQB 中宣告。

相關概念

第 856 頁的『在 Windows 中準備 Visual Basic 程式』
在 Windows 上使用 Microsoft Visual Basic 程式時要考量的資訊。

相關參考

第 770 頁的『鏈結 Visual Basic 應用程式與 IBM MQ MQI client 程式碼』
您可以將 Microsoft Visual Basic 應用程式與 Windows 上的 IBM MQ MQI client 程式碼相鏈結。

以 COBOL 撰寫程式碼

在 COBOL 中編碼 IBM MQ 程式時，請注意下列區段中的資訊。

具名常數

會顯示常數名稱，其中包含底線字元 (_) 作為名稱的一部分。 在 COBOL 中，您必須使用連字號字元 (-) 來取代底線。 具有字串值的常數會使用單引號字元 (') 作為字串定界字元。 若要讓編譯器接受此字元，請使用編譯器選項 APOST。

副本檔案 CMQV 包含已命名常數的宣告作為 level-10 項目。 若要使用常數，請明確宣告 level-01 項目，然後使用 COPY 陳述式複製常數的宣告:

```
WORKING-STORAGE SECTION.
01 MQM-CONSTANTS.
COPY CMQV.
```

然而，此方法會導致常數佔用程式中的儲存體，即使未參照它們也一樣。 如果常數包含在相同執行單元內的許多個別程式中，則會存在多個常數副本; 這可能會導致使用大量主儲存體。 若要避免此問題，您可以將 GLOBAL 子句新增至 level-01 宣告:

```
* Declare a global structure to hold the constants
01 MQM-CONSTANTS GLOBAL.
COPY CMQV.
```

這只會配置執行單元內 一組 常數的儲存體; 不過，執行單元內的 任何 程式都可以參照這些常數，而不只是包含 level-01 宣告的程式。

確保結構對齊

請小心確保傳遞以在 MQ 呼叫上啟動的 IBM MQ 結構必須在單字界限上對齊。 單字界限是 32 位元處理程序的 4 個位元組， 64 位元處理程序的 8 個位元組， 128 位元處理程序的 16 個位元組 (IBM i)。

可能的話，將所有 IBM MQ 結構放在一起，使它們全部對齊。

以 System/390 組譯語言撰寫程式碼 (訊息佇列介面)

當以組譯語言撰寫 IBM MQ for z/OS 程式的程式碼時，請注意下列各節中的資訊。

名稱

呼叫說明中的參數名稱，以及結構說明中的欄位名稱會以大小寫混合格式顯示。 在 IBM MQ 提供的組譯語言巨集中，所有名稱都是大寫。

使用 MQI 呼叫

MQI 是呼叫介面，因此組譯語言程式必須遵循 OS 鏈結慣例。

尤其是在發出 MQI 呼叫之前，組譯語言程式必須在至少有 18 個完整單字的儲存區中登錄 R13 。 此儲存區為被呼叫的程式提供儲存體。 它會在內容毀損之前儲存呼叫端的暫存器，並在傳回時還原呼叫端的暫存器內容。

註: 對於使用 DFHEIENT 巨集來設定其動態儲存體，但選擇將 R13 中的預設 DATAREG 置換為其他暫存器的 CICS 組譯語言程式而言，這很重要。 當 CICS Resource Manager 介面從 Stub 接收控制時，它會將暫存器的現行內容儲存在 R13 指向的位址。 無法為此目的保留儲存區會導致無法預期的結果，且可能導致 CICS 異常終止。

宣告常數

大部分常數在巨集 CMQA 中宣告為相等。

不過，下列常數無法定義為相等，當您使用預設選項來呼叫巨集時，並不會包含這些常數:

- MQACT_NONE
- MQCI_NONE

- MQFMT_NONE
- MQFMT_ADMIN
- MQFMT_COMMAND_1
- MQFMT_COMMAND_2
- MQFMT_DEAD_LETTER_HEADER
- MQFMT_EVENT
- MQFMT_IMS
- MQFMT_IMS_VAR_STRING
- MQFMT_PCF
- MQFMT_STRING
- MQFMT_TRIGGER
- MQFMT_XMIT_Q_HEADER
- MQMI_NONE

若要併入它們，請在呼叫巨集時新增關鍵字 EQUONLY=NO。

CMQA 受到多重宣告的保護，因此您可以多次併入它。 不過，關鍵字 EQUONLY 只會在第一次併入巨集時生效。

指定結構的名稱

為了容許宣告結構的多個實例，產生結構的巨集會以使用者指定的字串和底線字元 (_) 作為每一個欄位名稱的字首。

當您呼叫巨集時，請指定字串。 如果您沒有指定字串，巨集會使用結構名稱來建構字首:

```
* Declare two object descriptors
CMQODA         Prefix used="MQOD_" (the default)
MY_MQOD CMQODA         Prefix used="MY_MQOD_"
```

呼叫說明 中的結構宣告會顯示預設字首。

指定結構的形式

巨集可以產生兩種形式之一的結構宣告，由 DSECT 參數控制:

DSECT = YES
使用組譯語言 DSECT 指令來啟動新的資料區段; 結構定義緊跟在 DSECT 陳述式後面。 未配置任何儲存體，因此無法起始設定。 巨集呼叫上的標籤會作為資料區段的名稱; 如果未指定標籤，則會使用結構的名稱。

DSECT = NO
組譯語言 DC 指令是用來定義常式中現行位置的結構。 欄位會以值來起始設定，您可以在巨集呼叫上撰寫相關參數來指定這些值。 在巨集呼叫上未指定任何值的欄位會以預設值起始設定。

如果未指定 DSECT 參數，則假設 DSECT = NO。

控制清單

您可以使用 LIST 參數來控制組譯語言清單中結構宣告的外觀:

LIST = YES
結構宣告會出現在組譯語言清單中。

LIST = NO
結構宣告不會出現在組譯語言清單中。 如果未指定 LIST 參數，則會假設此項。

指定欄位的起始值

您可以將該欄位的名稱 (不含字首) 編碼成巨集呼叫的參數，並伴隨必要的值，以指定要用來起始設定結構中欄位的值。

例如，若要宣告訊息描述子結構，並使用以 MQMT_REQUEST 起始設定的 *MsgType* 欄位，以及以字串 MY_REPLY_TO_QUEUE 起始設定的 *ReplyToQ* 欄位，請使用下列程式碼:

```
MY_MQMD    CMQMDA      MSGTYPE=MQMT_REQUEST,       X
REPLYTOQ=MY_REPLY_TO_QUEUE
```

如果您在巨集呼叫中指定已命名的常數 (或相等) 作為值，請使用 CMQA 巨集來定義已命名的常數。 您不得以單引號 ('') 括住字串值。

撰寫可重新輸入的程式

IBM MQ 將其結構用於輸入及輸出。 如果您想要您的程式保持可重新輸入:

1. 將結構的工作儲存體版本定義為 DSECT ，或在已定義的 DSECT 內行內定義結構。 然後，將 DSECT 複製到使用下列方式取得的儲存體:
   - 若為批次和 TSO 程式， STORAGE 或 GETMAIN z/OS 組譯器巨集
   - 若為 CICS，工作儲存體 DSECT (DFHEISTG) 或 EXEC CICS GETMAIN 指令

   若要正確起始設定這些工作儲存體結構，請將對應結構的常數版本複製到工作儲存體版本。

   註: MQMD 和 MQXQH 結構長度各超過 256 個位元組。 若要將這些結構複製到儲存體，請使用 MVCL 組合器指令。
2. 使用 CALL 巨集的 LIST 形式 (MF=L) 來保留儲存體中的空間。 當您使用 CALL 巨集來發出 MQI 呼叫時，請使用先前保留的儲存體來使用巨集的 EXECUTE 表單 (MF=E) ， 如 第 886 頁的『使用 CEDF』下的範例所示。 如需如何執行此動作的其他範例，請參閱 IBM MQ 隨附的組譯器語言範例程式。

請使用組譯語言 RENT 選項來協助您判斷您的程式是否可重新輸入。

如需撰寫可重新輸入程式的相關資訊，請參閱 *z/OS MVS Application Development Guide: Assembler Language Programs*。

使用 CEDF

如果您想要使用 CICS 提供的交易 CEDF (CICS 執行診斷機能) 來協助您對程式進行除錯，請將 ,VL 關鍵字新增至每一個 CALL 陳述式，例如:

```
CALL MQCONN,(NAME,HCONN,COMPCODE,REASON),MF=(E,PARMAREA),VL
```

前一個範例是可重新輸入的組譯語言碼，其中 PARMAREA 是您指定之工作儲存體中的區域。

使用 MQI 呼叫

MQI 是呼叫介面，因此組譯語言程式必須遵循 OS 鏈結慣例。 尤其是在發出 MQI 呼叫之前，組譯語言程式必須在至少有 18 個完整單字的儲存區中登錄 R13 。 此儲存區為被呼叫的程式提供儲存體。 它會在內容毀損之前儲存呼叫端的暫存器，並在傳回時還原呼叫端的暫存器內容。

註: 對於使用 DFHEIENT 巨集來設定其動態儲存體，但選擇將 R13 中的預設 DATAREG 置換為其他暫存器的 CICS 組譯語言程式而言，這很重要。 當 CICS Resource Manager 介面從 Stub 接收控制時，它會將暫存器的現行內容儲存在 R13 指向的位址。 如果無法為此目的保留適當的儲存區，則會產生無法預期的結果，且可能導致 CICS 異常終止。

IBM i 在 RPG 中撰寫 IBM MQ 程式的程式碼 (僅限 IBM i)

在 IBM MQ 文件中，呼叫的參數、資料類型的名稱、結構的欄位，以及常數的名稱都是使用其完整名稱來說明。 在 RPG 中，這些名稱縮寫為六個或更少的大寫字元。

例如，在 RPG 中，欄位 *MsgType* 會變成 *MDMT* 。 如需相關資訊，請參閱 IBM i Application Programming Reference (ILE/RPG)。

以 PL/I 撰寫程式碼 (僅限 z/OS)

在 PL/I 中編碼 IBM MQ 時的有用資訊。

結構

結構是以 BASED 屬性來宣告，因此除非程式宣告一個以上的結構實例，否則請勿佔用任何儲存體。

可以使用 `like` 屬性來宣告結構的實例，例如:

```
dcl my_mqmd    like MQMD; /* one instance */
dcl my_other_mqmd like MQMD; /* another one */
```

結構欄位以 INITIAL 屬性宣告; 當使用 `like` 屬性來宣告結構的實例時，該實例會繼承為該結構定義的起始值。 您只需要設定所需值不同於起始值的那些欄位。

PL/I 不區分大小寫，因此呼叫、結構欄位及常數的名稱可以小寫、大寫或大小寫混合。

具名常數

已命名的常數會宣告為巨集變數; 因此，程式未參照的已命名常數不會佔用已編譯程序中的任何儲存體。

不過，當編譯程式時，必須指定讓巨集前置處理器處理原始程式的編譯器選項。

所有巨集變數都是字元變數，甚至是代表數值的變數。 雖然這可能違反直覺，但在巨集變數已由巨集處理器替代之後，它不會導致任何資料類型衝突，例如:

```
%dcl MQMD_STRUC_ID char;
%MQMD_STRUC_ID = '''MD ''';

%dcl MQMD_VERSION_1 char;
%MQMD_VERSION_1 = '1';
```

使用 IBM MQ 範例程序化程式

這些範例程式是以程序化語言撰寫，並示範「訊息佇列介面 (MQI)」的一般用法。 不同平台上的 IBM MQ 程式。

關於這項作業

範例有兩組:

- 分散式系統和 IBM i 的程式範例。
- z/OS 的程式範例。

程序

- 請使用下列鏈結，以進一步瞭解範例程式:
  - 第 888 頁的『在 Multiplatforms 上使用範例程式』
  - z/OS 第 975 頁的『使用 z/OS 的程式範例』

相關概念

第 6 頁的『應用程式開發概念』
您可以使用選擇的程序化或物件導向語言來撰寫 IBM MQ 應用程式。 在開始設計及撰寫 IBM MQ 應用程式之前，請先熟悉基本 IBM MQ 概念。

第 5 頁的『開發適用於 IBM MQ 的應用程式』
您可以開發應用程式來傳送及接收訊息，以及管理佇列管理程式和相關資源。 IBM MQ 支援以許多不同語言及架構撰寫的應用程式。

第 41 頁的『IBM MQ 應用程式的設計考量』
當您決定應用程式如何利用可供您使用的平台及環境時，您需要決定如何使用 IBM MQ 所提供的特性。

第 603 頁的『撰寫佇列作業的程序化應用程式』
請利用這項資訊來瞭解如何撰寫佇列作業應用程式、連接佇列管理程式、發佈/訂閱，以及開啟和關閉物件。

第 765 頁的『撰寫用戶端程序化應用程式』
使用程序化語言在 IBM MQ 上撰寫用戶端應用程式所需的資訊。

第 676 頁的『撰寫發佈/訂閱應用程式』
開始撰寫發佈/訂閱 IBM MQ 應用程式。

第 837 頁的『建置程序化應用程式』
您可以使用數種程序化語言之一來撰寫 IBM MQ 應用程式，並在數個不同的平台上執行該應用程式。

第 871 頁的『處理程序化程式錯誤』
此資訊說明與應用程式 MQI 呼叫相關聯的錯誤，當它進行呼叫時，或當它的訊息遞送至其最終目的地時。

Multi 在 Multiplatforms 上使用範例程式

這些範例程序程式隨產品一起提供。 範例以 C 和 COBOL 撰寫，並示範「訊息佇列介面 (MQI)」的一般用法。

關於這項作業

範例不是用來示範一般程式設計技術，因此會省略您可能想要併入正式作業程式中的一些錯誤檢查。

產品隨附所有範例的原始碼; 此來源包括說明程式中示範之訊息佇列作業技術的註解。

IBM i 如需 RPG 程式設計，請參閱 IBM i Application Programming Reference (ILE/RPG)。

範例名稱以字首 amq 開頭。 第四個字元指出程式設計語言，以及必要時的編譯器:

- s: C 語言
- 0: IBM 和 Micro Focus 編譯器上的 COBOL 語言
- i: 僅限 IBM 編譯器上的 COBOL 語言
- m: 僅限 Micro Focus 編譯器上的 COBOL 語言

執行檔的第八個字元指出範例是在本端連結模式還是用戶端模式中執行。 如果沒有第八個字元，則範例會以本端連結模式執行。 如果第 8 個字元是 ' c' ，則範例會以用戶端模式執行。

您必須先建立並配置佇列管理程式，才能執行範例應用程式。 若要設定佇列管理程式以接受用戶端連線，請參閱 第 897 頁的『配置佇列管理程式以接受 Multiplatforms 上的用戶端連線』 。

程序

- 請使用下列鏈結，以進一步瞭解範例程式:
  - 第 889 頁的『Multiplatforms 上範例程式中示範的特性』
  - 第 896 頁的『準備及執行範例程式』
  - 第 902 頁的『API 結束程式範例程式』
  - 第 902 頁的『非同步使用範例程式』
  - 第 904 頁的『非同步 Put 範例程式』

- 第 904 頁的『瀏覽範例程式』
- 第 905 頁的『瀏覽器範例程式』
- 第 906 頁的『CICS 交易範例』
- 第 907 頁的『Connect 範例程式』
- 第 908 頁的『Data-Conversion 範例程式』
- 第 908 頁的『資料庫協調範例』
- 第 914 頁的『無法傳送郵件的佇列處理程式範例』
- 第 915 頁的『「配送清單」範例程式』
- 第 915 頁的『Echo 範例程式』
- 第 916 頁的『Get 範例程式』
- 第 917 頁的『高可用性範例程式』
- 第 921 頁的『Inquire 範例程式』
- 第 922 頁的『訊息控點範例程式的查詢內容』
- 第 922 頁的『發佈/訂閱範例程式』
- 第 926 頁的『發佈結束程式範例程式』
- 第 927 頁的『放置範例程式』
- 第 928 頁的『參照訊息範例程式』
- 第 935 頁的『要求範例程式』
- 第 940 頁的『設定範例程式』
- 第 941 頁的『TLS 範例程式』
- 第 944 頁的『觸發程式範例』
- 第 945 頁的『在 AIX, Linux, and Windows 上使用 TUXEDO 範例』
- 第 954 頁的『在 Windows 上使用 SSPI 安全結束程式』
- 第 955 頁的『使用遠端佇列執行範例』
- 第 955 頁的『「叢集佇列監視」範例程式 (AMQSCLM)』
- 第 962 頁的『連線端點查閱 (CEPL) 的程式範例』

相關概念

第 441 頁的『C++ 範例程式』
提供四個範例程式，以示範取得及放置訊息。

相關工作

第 975 頁的『使用 z/OS 的程式範例』
隨 IBM MQ for z/OS 提供的範例程序化應用程式示範「訊息佇列介面 (MQI)」的一般用法。

Multi *Multiplatforms* 上範例程式中示範的特性

表格的集合，顯示 IBM MQ 範例程式所示範的技術。

所有範例都使用 MQOPEN 和 MQCLOSE 呼叫來開啟和關閉佇列，因此這些技術不會分別列在表格中。 請參閱包含您感興趣之平台的標題。

z/OS 若為 z/OS 平台，請參閱 第 975 頁的『使用 z/OS 的程式範例』。

Linux AIX *AIX and Linux* 系統的範例

IBM MQ for AIX or Linux 的範例程式所示範的技術。

請參閱 第 899 頁的『在 AIX and Linux 上準備及執行範例程式』，以找出 IBM MQ for AIX or Linux 的程式範例儲存在何處。

第 890 頁的表 156 此表格列出所提供的 C 和 COBOL 原始檔，以及是否包含伺服器或用戶端執行檔。

表 156: 示範在 AIX and Linux 上使用 MQI (C 及 COBOL) 的程式範例.

具有四個直欄的表格。 第一個直欄列出範例所示範的技術。 第二個直欄列出 C 範例，而第三個直欄列出示範第一個直欄中所列出之每一個技術的 COBOL 範例。 第四個直欄顯示是否包含伺服器 C 執行檔，第五個直欄顯示是否包含用戶端 C 執行檔。

技術	C (來源) (第 892 頁的『1』)	COBOL (來源) (第 892 頁的『2』)	伺服器 (C 執行檔)	用戶端 (C 執行檔)
使用發佈/訂閱介面	amqspuba amqssuba amqssbxa	無範例	amqspub amqsssub amqssbx	無範例
使用 MQPUT 呼叫來放置訊息	amqsput0	amq0put0	amqsput	amqsputc
使用 MQPUT1 呼叫放置單一訊息	Amqsinqa amqsecha	amqminqx amqmechx amqiinqx amqiechx	amqsinq amqsech	amqsechc
將訊息放入配送清單 (第 892 頁的『3』)	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
回覆要求訊息	阿姆克辛加	amqminqx amqiinqx	阿姆克辛格	無範例
使用瀏覽取得訊息 (不等待)	amqsgbr0	amq0gbr0	amqsgbr	無範例
取得訊息 (等待有時間限制)	amqsget0	amq0get0	amqsget	amqsgetc
取得訊息 (無限制等待)	amqstrg0	無範例	amqstrg	amqstrgc
取得訊息 (含資料轉換)	阿姆克塞查	無範例	阿姆克塞赫	無範例
將參照訊息放入佇列 (第 892 頁的『3』)	阿姆格斯普爾馬	無範例	amqsprm	amqsprmc
從佇列取得參照訊息 (第 892 頁的『3』)	amqsgrma	無範例	amqsgrm	amqsgrmc
參照訊息通道結束程式 (第 892 頁的『3』)	amqsqrma amqsxrma	無範例	amqsxrm	無範例
瀏覽訊息的前 20 個字元	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
瀏覽完整訊息	amqsbcg0	無範例	amqsbcg	amqsbcgc
使用共用輸入佇列	阿姆克辛加	amqminqx amqiinqx	阿姆克辛格	amqsinqc
使用專用輸入佇列	amqstrg0	amq0req0	amqstrg	amqstrgc
使用 MQINQ 呼叫	阿姆克辛加	amqminqx amqiinqx	阿姆克辛格	無範例
使用 MQSET 呼叫	阿姆克塞塔	amqmsetx amqisetx	amqsset	amqssetc
使用回覆目的地佇列	amqsreq0	amq0req0	amqsreq	amqsreqc
要求訊息異常狀況	amqsreq0	amq0req0	amqsreq	無範例
接受截斷的訊息	amqsgbr0	amq0gbr0	amqsgbr	無範例
使用已解析的佇列名稱	amqsgbr0	amq0gbr0	amqsgbr	無範例
觸發程序	amqstrg0	無範例	amqstrg	amqstrgc

表 *156:* 示範在 *AIX and Linux* 上使用 *MQI (C* 及 *COBOL)* 的程式範例.

具有四個直欄的表格。 第一個直欄列出範例所示範的技術。 第二個直欄列出 C 範例，而第三個直欄列出示範第一個直欄中所列出之每一個技術的 COBOL 範例。 第四個直欄顯示是否包含伺服器 C 執行檔，第五個直欄顯示是否包含用戶端 C 執行檔。

*(*繼續*)*

技術	**C (來源) (第 892 頁的『1』)**	**COBOL (來源) (第 892 頁的『2』)**	**伺服器 (C 執行檔)**	**用戶端 (C 執行檔)**
使用資料轉換	(第 892 頁的『4』)	無範例	無範例	無範例
IBM MQ (協調符合 XA 標準的資料庫管理程式) 使用 SQL 存取單一資料庫	amqsxas0.sqc Db2 amqsxas0.ec Informix	amq0xas0.sqb	無範例	無範例
IBM MQ (協調符合 XA 標準的資料庫管理程式) 使用 SQL 存取兩個資料庫	amqsxag0.c amqsxab0.sqc amqsxaf0.sqc	amq0xag0.cbl amq0xab0.sqb amq0xaf0.sqb	無範例	無範例
CICS 交易 (第 892 頁的『5』)	amqscic0.ccs	無範例	amqscic0	無範例
Encina 交易 (第 892 頁的『3』)	amqsxae0	無範例	amqsxae0	無範例
放置訊息的 TUXEDO 交易 第 892 頁的『6』)	amqstxpx	無範例	無範例	無範例
用來取得訊息的 TUXEDO 交易 (第 892 頁的『6』)	amqstxgx	無範例	無範例	無範例
TUXEDO 的伺服器 (第 892 頁的『6』)	amqstxsx	無範例	無範例	無範例
無法傳送郵件的佇列處理程式 (dead-letter queue handler)	目錄 `./tools/c/Samples/dlq` (第 892 頁的『7』)	無範例	amqsdlq	無範例
從 MQI 用戶端，放置訊息	無範例	無範例	無範例	amqsputc
從 MQI 用戶端取得訊息	無範例	無範例	無範例	amqsgetc
使用 MQCONNX 連接至佇列管理程式	amqscnxc	無範例	無範例	amqscnxc
使用 API 結束程式	amqsaxe0	無範例	阿姆克薩謝	無範例
叢集工作量平衡結束程式	amqswlm0	無範例	amqswlm	無範例
非同步放置訊息並使用 MQSTAT 呼叫取得狀態	amqsapt0	無範例	amqsapt	amqsaptc
可重新連接的用戶端	amqsphac amqsghac amqsmhac	無範例	不適用	amqsphac amqsghac amqsmhac
使用訊息消費者以非同步方式耗用來自多個佇列的訊息	amqscbf0	無範例	amqscbf	amqscbfc
在 MQCONNX 上指定 TLS 連線資訊	amqssslc	無範例	不適用	amqssslc

附註:

1. IBM MQ MQI client 範例的執行檔版本與伺服器環境中執行的範例共用相同的來源。
2. 以 Micro Focus COBOL 編譯器開頭 'amqm' 的編譯程式，以 IBM COBOL 編譯器開頭 'amqi' 的編譯程式，以任一個開頭 'amq0' 的編譯程式。
3. AIX 僅在 IBM MQ for AIX 上受支援。
4. AIX 在 IBM MQ for AIX 上，此程式稱為 `amqsvfc0.c`
5. AIX CICS 僅受 IBM MQ for AIX 支援。
6. Linux IBM MQ for Linux on System p 不支援 TUXEDO。
7. 無法傳送郵件的佇列處理程式的來源由數個檔案組成，並在個別目錄中提供。

如需 AIX and Linux 系統支援的詳細資訊，請參閱 IBM MQ 的系統需求。

Windows *IBM MQ for Windows* 的範例

IBM MQ for Windows 的範例程式所示範的技術。

第 892 頁的表 157 列出所提供的 C 和 COBOL 原始檔，以及是否包含伺服器或用戶端執行檔。

表 *157: IBM MQ for Windows* 範例程式示範 *MQI (C* 及 *COBOL)* 的用法

技術	**C (來源)**	**COBOL (程式碼)**	**伺服器 (C 執行檔)**	**用戶端 (C 執行檔)**
使用發佈/訂閱介面	amqspuba amqssuba amqssbxa	無範例	amqspub amqsssub amqssbx	無範例
使用 MQPUT 呼叫來放置訊息	amqsput0	amq0put0	amqsput	amqsputc
使用 MQPUT1 呼叫放置單一訊息	Amqsinqa amqsecha	amqminq2 amqmech2 amqiinq2 amqiech2	amqsinq amqsech	amqsinqc amqsechc
將訊息放入配送清單	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
回覆要求訊息	阿姆克辛加	amqminq2 amqiinq2	阿姆克辛格	amqsinqc
取得訊息 (不等待)	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
取得訊息 (等待有時間限制)	amqsget0	amq0get0	amqsget	amqsgetc
取得訊息 (無限制等待)	amqstrg0	無範例	amqstrg	amqstrgc
取得訊息 (含資料轉換)	阿姆克塞查	無範例	阿姆克塞赫	amqsechc
將參照訊息放入佇列	阿姆格斯普爾馬	無範例	amqsprm	amqsprmc
從佇列取得參照訊息	amqsgrma	無範例	amqsgrm	amqsgrmc
參照訊息通道結束程式	amqsqrma amqsxrma	無範例	amqsxrm	無範例
瀏覽訊息的前 20 個字元	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
瀏覽完整訊息	amqsbcg0	無範例	amqsbcg	amqsbcgc
使用共用輸入佇列	阿姆克辛加	amqminq2 amqiinq2	阿姆克辛格	amqsinqc
使用專用輸入佇列	amqstrg0	amq0req0	amqstrg	amqstrgc

表 157: IBM MQ for Windows 範例程式示範 MQI (C 及 COBOL) 的用法 (繼續)

技術	C (來源)	COBOL (程式碼)	伺服器 (C 執行檔)	用戶端 (C 執行檔)
使用 MQINQ 呼叫	阿姆克辛加	amqminq2 amqiinq2	阿姆克辛格	amqsinqc
使用 MQSET 呼叫	阿姆克塞塔	amqmset2 amqiset2	amqsset	amqssetc
使用 MQINQMP 呼叫	阿姆齊格馬	無範例	無範例	無範例
使用回覆目的地佇列	amqsreq0	amq0req0	amqsreq	amqsreqc
要求訊息異常狀況	amqsreq0	amq0req0	amqsreq	amqsreqc
接受截斷的訊息	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
使用已解析的佇列名稱	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
觸發程序	amqstrg0	無範例	amqstrg	amqstrgc
使用資料轉換	amqsvfc0	無範例	無範例	無範例
IBM MQ (協調符合 XA 標準的資料庫管理程式) 使用 SQL 存取單一資料庫	amqsxas0.sqc Db2 amqsxas0.ec Informix	amq0xas0.sqb	無範例	無範例
IBM MQ (協調符合 XA 標準的資料庫管理程式) 使用 SQL 存取兩個資料庫	amqsxag0.c amqsxab0.sqc Db2 amqsxaf0.sqc Db2	amq0xag0.cbl amq0xab0.sqb amq0xaf0.sqb	無範例	無範例
放置訊息的 TUXEDO 交易	amqstxpx	無範例	無範例	無範例
用來取得訊息的 TUXEDO 交易	amqstxgx	無範例	無範例	無範例
TUXEDO 的伺服器	amqstxsx	無範例	無範例	無範例
無法傳送郵件的佇列處理程式 (dead-letter queue handler)	目錄 `./tools/c/Samples/dlq` (第 894 頁的『1』)	無範例	amqsdlq	無範例
從 IBM MQ MQI client 中，放置訊息	無範例	無範例	無範例	amqsputc
從 IBM MQ MQI client 取得訊息	無範例	無範例	無範例	amqsgetc
使用 MQCONNX 連接至佇列管理程式	amqscnxc	無範例	無範例	amqscnxc
使用 API 結束程式	amqsaxe0	無範例	阿姆克薩謝	無範例
叢集工作量平衡	amqswlm0	無範例	amqswlm	無範例
SSPI 安全常式	阿姆奇斯平	無範例	amqrspin.dll	amqrspin.dll
非同步放置訊息並使用 MQSTAT 呼叫取得狀態	amqsapt0	無範例	amqsapt	amqsaptc

表 157: IBM MQ for Windows 範例程式示範 MQI (C 及 COBOL) 的用法 (繼續)				
技術	**C (來源)**	**COBOL (程式碼)**	**伺服器 (C 執行檔)**	**用戶端 (C 執行檔)**
可重新連接的用戶端	amqsphac amqsghac amqsmhac	無範例	不適用	amqsphac amqsghac amqsmhac
使用訊息消費者以非同步方式耗用來自多個佇列的訊息	amqscbf0	無範例	amqscbf	amqscbfc
在 MQCONNX 上指定 TLS 連線資訊	amqssslc	無範例	不適用	amqssslc

附註:

1. 無法傳送郵件的佇列處理程式的來源由數個檔案組成，並在個別目錄中提供。

Windows *IBM MQ for Windows* 的 *Visual Basic* 範例

Windows 系統上 IBM MQ 的範例程式所示範的技術。

第 894 頁的表 158 顯示 IBM MQ for Windows 範例程式所示範的技術。

專案可以包含數個檔案。 當您在 Visual Basic 內開啟專案時，會自動載入其他檔案。 未提供可執行程式。

所有範例專案 (mqtrivc.vbp 除外) 都設定為使用 IBM MQ 伺服器。 若要瞭解如何變更範例專案以使用 IBM MQ 用戶端，請參閱 第 856 頁的『在 Windows 中準備 Visual Basic 程式』。

表 158: IBM MQ for Windows 範例程式示範 MQI (Visual Basic) 的用法	
技術	**專案檔名稱**
使用 MQPUT 呼叫來放置訊息	amqsputb.vbp
使用 MQGET 呼叫取得訊息	amqsgetb.vbp
使用 MQGET 呼叫瀏覽佇列	amqsbcgb.vbp
簡式 MQGET 和 MQPUT 範例 (用戶端)	mqtrivc.vbp
簡式 MQGET 和 MQPUT 範例 (伺服器)	mqtrivs.vbp
使用 MQPUT 及 MQGET 來放置及取得字串及使用者定義結構	strings.vbp
使用 PCF 結構來啟動及停止通道	pcfsamp.vbp
使用 MQAI 建立佇列	amqsaicq.vbp
使用 MQAI 列出佇列管理程式的佇列	amqsailq.vbp
使用 MQAI 監視事件	amqsaiem.vbp

IBM i *IBM i* 的範例

IBM i 系統上 IBM MQ 的範例程式所示範的技術。

第 894 頁的表 159 顯示 IBM MQ for IBM i 範例程式所示範的技術。 部分技術出現在多個範例程式中，但表格中只會列出一個程式。

表 159: 示範在 IBM i 上使用 MQI (C 和 COBOL) 的範例程式				
技術	**C (來源) (第 896 頁的『1』)**	**COBOL (來源) (第 896 頁的『2』)**	**RPG (來源) (第 896 頁的『3』)**	**用戶端 (C 執行檔) (4)**
使用 MQPUT 呼叫來放置訊息	AMQSPUT0	AMQ0PUT4	AMQ3PUT4	amqsputc

表 159: 示範在 IBM i 上使用 MQI (C 和 COBOL) 的範例程式 (繼續)

技術	C (來源) (第 896 頁的 『1』)	COBOL (來源) (第 896 頁的 『2』)	RPG (來源) (第 896 頁的 『3』)	用戶端 (C 執行檔) (4)
使用 MQPUT 呼叫從資料檔放置訊息	AMQSPUT4	無範例	無範例	無範例
使用 MQPUT1 呼叫放置單一訊息	AMQSINQ4, AMQSECH4	AMQ0INQ4, AMQ0ECH4	AMQ3INQ4, AMQ3ECH4	AMQSINQC、AMQSECHC
將訊息放入配送清單	AMQSPTL4	無範例	無範例	AMQSPTLC
回覆要求訊息	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
取得訊息 (不等待)	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
取得訊息 (等待有時間限制)	AMQSGET4	AMQ0GET4	AMQ3GET4	AMQSGETC
取得訊息 (無限制等待)	AMQSTRG4	無範例	AMQ3TRG4	AMQSTRGC
取得訊息 (含資料轉換)	AMQSECH4	AMQ0ECH4	AMQ3ECH4	AMQSECHC
將參照訊息放入佇列	AMQSPRM4	無範例	無範例	AMQSPRMC
從佇列取得參照訊息	AMQSGRM4	無範例	無範例	AMQSGRMC
參照訊息通道結束程式	AMQSQRM4, AMQSXRM4	無範例	無範例	無範例
訊息結束	AMQSCMX4	無範例	無範例	無範例
瀏覽訊息的前 49 個字元	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
瀏覽完整訊息	AMQSBCG4	無範例	無範例	AMQSBCGC
使用共用輸入佇列	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
使用專用輸入佇列	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
使用 MQINQ 呼叫	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
使用 MQSET 呼叫	AMQSSET4	AMQ0SET4	AMQ3SET4	AMQSSETC
使用回覆目的地佇列	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
要求訊息異常狀況	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
接受截斷的訊息	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
使用已解析的佇列名稱	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
觸發程序	AMQSTRG4	無範例	AMQ3TRG4	AMQSTRGC
觸發程式伺服器	AMQSERV4	無範例	AMQ3SRV4	無範例
使用觸發程式伺服器 (包括 CICS 交易)	AMQSERV4	無範例	AMQ3SRV4	無範例
使用資料轉換	AMQSVFC4	無範例	無範例	無範例
使用 API 結束程式	AMQSAXE0	無範例	無範例	無範例
叢集工作量平衡	AMQSWLM0	無範例	無範例	無範例
非同步放置訊息並使用 MQSTAT 呼叫取得狀態	AMQSAPT0	無範例	無範例	AMQSAPTC
使用發佈/訂閱介面	AMQSPUBA、AMQSSUBA、AMQSSBXA	無範例	無範例	AMQSPUBC、AMQSSUBC、AMQSSBXC

表 159: 示範在 IBM i 上使用 MQI (C 和 COBOL) 的範例程式 (繼續)				
技術	**C (來源) (第 896 頁的『1』)**	**COBOL (來源) (第 896 頁的『2』)**	**RPG (來源) (第 896 頁的『3』)**	**用戶端 (C 執行檔) (4)**
可重新連接的用戶端 (5)	AMQSPHAC、AMQSGHAC、AMQSMHAC	無範例	無範例	無範例
使用訊息消費者非同步使用來自多個佇列的訊息 (5)	AMQSCBFO	無範例	無範例	無範例
在 MQCONNX 上指定 TLS 連線資訊	AMQSSSLC	無範例	無範例	AMQSSSLC
使用 MQCONNX 連接至佇列管理程式	AMQSCNXC	無範例	無範例	AMQSCNXC
使用 MQINQMP 從訊息佇列查詢訊息控點的內容	AMQISQMA	無範例	無範例	AMQISQMC
使用 MQSETMP 設定訊息控點的內容，並將它放置到訊息佇列中	AMQSSQMA	無範例	無範例	AMQSSQMC

附註:

1. C 範例的來源位於檔案 QMQMSAMP/QCSRC。 併入檔在檔案 QMQM/H 中作為成員存在。
2. COBOL 範例的來源位於檔案 QMQMSAMP/QCBLLESRC 中。 成員命名為 AMQ0 xxx 4 ，其中 xxx 指示範例功能。
3. RPG 範例的來源位於 QMQMSAMP/QRPGLESRC。 成員命名為 AMQ3 xxx 4 ，其中 xxx 表示範例功能。副本成員存在於 QMQM/QRPGLESRC 中。 每一個成員名稱都有字尾 G。
4. IBM MQ MQI client 範例的執行檔版本與伺服器環境中執行的範例共用相同的來源。 用戶端環境中的範例來源與伺服器相同。 IBM MQ MQI client 範例與用戶端程式庫 LIBMQIC 相鏈結， IBM MQ 伺服器範例與伺服器程式庫 LIBMQM 相鏈結。
5. 如果必須執行 Reconnectable 用戶端的範例應用程式及非同步消費者應用程式的用戶端執行檔，則必須編譯它並與執行緒檔案庫 LIBMQIC_R 鏈結。 因此，它必須在執行緒環境中執行。 將環境變數 QIBM_MULTI_THREADY 設為 'Y' ，並從 qsh 執行應用程式。

   如需相關資訊，請參閱 使用 Java 和 JMS 設定 IBM MQ 。

如需相關資訊，請參閱 第 898 頁的『在 IBM i 上準備及執行範例程式』 。

除了這些之外， IBM MQ for IBM i 範例選項還包含範例資料檔，您可以使用它作為範例程式、AMQSDATA 及範例 CL 程式的輸入，以示範管理作業。 CL 範例在 管理 IBM i 中有說明。 您可以使用範例 CL 程式 amqsamp4 來建立佇列，以與本主題中說明的範例程式搭配使用。

Multi 準備及執行範例程式

完成部分起始準備之後，您可以執行範例程式。

關於這項作業

在執行範例程式之前，您必須先建立佇列管理程式，並建立您需要的佇列。 您也可能需要執行一些其他準備，例如，如果您想要執行 COBOL 範例。 完成必要的準備之後，您可以執行範例程式。

程序

如需如何準備及執行範例程式的相關資訊，請參閱下列主題:

- 第 897 頁的『配置佇列管理程式以接受 Multiplatforms 上的用戶端連線』
- 第 898 頁的『在 IBM i 上準備及執行範例程式』
- 第 899 頁的『在 AIX and Linux 上準備及執行範例程式』

- 第 900 頁的『在 Windows 上準備及執行範例程式』

Multi 配置佇列管理程式以接受 *Multiplatforms* 上的用戶端連線

您必須先建立佇列管理程式，才能執行範例應用程式。 然後，您可以配置佇列管理程式，以安全地接受來自以用戶端模式執行之應用程式的送入連線要求。

開始之前

請確定佇列管理程式已存在且已啟動。 透過發出 MQSC 指令，判定是否已啟用通道鑑別記錄:

```
DISPLAY QMGR CHLAUTH
```

重要: 這項作業預期會啟用通道鑑別記錄。 如果這是其他使用者及應用程式所使用的佇列管理程式，則變更此設定將會影響所有其他使用者及應用程式。 如果您的佇列管理程式未使用通道鑑別記錄，則步驟 4 可以取代為替代鑑別方法 (例如，安全結束程式) ，它會將 MCAUSER 設定為您將在步驟 第 897 頁的『1』中取得的 *non-privileged-user-id* 。

您必須知道應用程式預期使用的通道名稱，才能允許應用程式使用該通道。 您也必須知道應用程式預期使用哪些物件 (例如佇列或主題) ，以便您的應用程式可以使用它們。

關於這項作業

此作業會建立非特許使用者 ID ，用於連接至佇列管理程式的用戶端應用程式。 用戶端應用程式只會獲授與存取權，以便能夠使用它需要的通道，以及使用這個使用者 ID 所需要的佇列。

程序

1. 在佇列管理程式執行所在的系統上取得使用者 ID。 對於此作業，此使用者 ID 不得是特許管理使用者。此使用者 ID 將是在佇列管理程式上執行用戶端連線所使用的權限。
2. 使用下列指令啟動接聽器程式，其中:

   qmgr-name 是佇列管理程式的名稱
   nnnn 是您選擇的埠號

   a) ALW
   若為 AIX, Linux, and Windows 系統:

   ```
   runmqlsr -t tcp -m qmgr-name -p nnnn
   ```

   b) IBM i
   若為 IBM i:

   ```
   STRMQMLSR MQMNAME(qmgr-name) PORT(nnnn)
   ```

3. 如果您的應用程式使用 SYSTEM.DEF.SVRCONN 則已定義此通道。 如果您的應用程式使用另一個通道，請發出下列 MQSC 指令來建立它:

   ```
   DEFINE CHANNEL(' channel-name ') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
   DESCR('Channel for use by sample programs')
   ```

   其中 *channel-name* 是通道的名稱。
4. 透過發出下列 MQSC 指令，建立通道鑑別規則，只容許用戶端系統的 IP 位址使用通道:

   ```
   SET CHLAUTH(' channel-name ') TYPE(ADDRESSMAP) ADDRESS(' client-machine-IP-address ') +
   MCAUSER(' non-privileged-user-id ')
   ```

   其中

   channel-name 是通道的名稱。

client-machine-IP-address 是用戶端系統的 IP 位址。 如果您的範例用戶端應用程式在與佇列管理程式相同的機器上執行，則如果您的應用程式要使用 'localhost' 來連接，請使用 IP 位址 '127.0.0.1'。 如果要在中連接數個不同的用戶端機器，您可以使用型樣或範圍，而非單一 IP 位址。 如需詳細資料，請參閱 一般 IP 位址 。

non-privileged-user-id 是您在步驟 第 897 頁的『1』 中取得的使用者 ID

5. 如果您的應用程式使用 SYSTEM.DEFAULT.LOCAL.QUEUE 則已定義此佇列。 如果您的應用程式使用另一個佇列，請發出下列 MQSC 指令來建立它:

```
DEFINE QLOCAL(' queue-name ') DESCR('Queue for use by sample programs')
```

其中 *queue-name* 是佇列的名稱。

6. 發出下列 MQSC 指令，授與連接及查詢佇列管理程式的存取權:

```
SET AUTHREC OBJTYPE(QMGR) PRINCIPAL(' non-privileged-user-id ') +
AUTHADD(CONNECT, INQ)
```

其中 *non-privileged-user-id* 是您在步驟 第 897 頁的『1』 中取得的使用者 ID。

7. 如果您的應用程式是點對點應用程式 (亦即，它會使用佇列) ，請發出下列 MQSC 指令來授與存取權，以容許依要使用的使用者 ID 使用佇列來查詢及放置及取得訊息:

```
SET AUTHREC PROFILE(' queue-name ') OBJTYPE(QUEUE) +
PRINCIPAL(' non-privileged-user-id ') AUTHADD(PUT, GET, INQ, BROWSE)
```

其中

queue-name 是佇列的名稱

non-privileged-user-id 是您在步驟 第 897 頁的『1』 中取得的使用者 ID

8. 如果您的應用程式是發佈/訂閱應用程式 (亦即，它會使用主題) ，請發出 MQSC 指令，授與存取權以容許使用您要使用的使用者 ID 所使用的主題進行發佈及訂閱:

```
SET AUTHREC PROFILE('SYSTEM.BASE.TOPIC') OBJTYPE(TOPIC) +
PRINCIPAL(' non-privileged-user-id ') AUTHADD(PUB, SUB)
```

其中

non-privileged-user-id 是您在步驟 第 897 頁的『1』 中取得的使用者 ID

這將授與 非特許使用者 *ID* 對主題樹狀結構中任何主題的存取權，或者，您可以使用 **DEFINE TOPIC** 來定義主題物件，並僅授與存取權給該主題物件所參照的主題樹狀結構部分。 如需詳細資料，請參閱 控制使用者對主題的存取權 。

下一步

您的用戶端應用程式現在可以連接至佇列管理程式，並使用佇列來放置或取得訊息。

相關概念

ALW 授與 AIX, Linux, and Windows 上 IBM MQ 物件的存取權

相關參考

SET CHLAUTH

定義通道

DEFINE QLOCAL

SET AUTHREC

IBM i IBM i 上的 IBM MQ 權限

IBM i 在 *IBM i* 上準備及執行範例程式

在 IBM i 上執行範例程式之前，您必須先建立佇列管理程式，同時建立您需要的佇列。 如果您想要執行 COBOL 範例，則可能需要執行一些其他準備。

關於這項作業

IBM MQ for IBM i 範例程式的來源在程式庫 QMQMSAMP 中以 QCSRC、QCLSRC、QCBLLESRC 及 QRPGLESRC 的成員身分提供。

您可以在執行範例時使用自己的佇列，也可以執行範例程式 AMQSAMP4 來建立一些範例佇列。 AMQSAMP4 程式的來源包括在檔案庫 QMQMSAMP 中的檔案 QCLSRC 中。 您可以使用 CRTCLPGM 指令來編譯它。

若要執行範例，請使用 QMQM 程式庫中提供的 C 執行檔版本，或以類似任何其他 IBM MQ 應用程式的方式編譯它們。

程序

1. 建立佇列管理程式並設定預設定義。

   您必須先執行此動作，才能執行任何範例程式。 如需建立佇列管理程式的相關資訊，請參閱 管理 IBM MQ。 如需配置佇列管理程式以安全地接受來自以用戶端模式執行之應用程式的送入連線要求的相關資訊，請參閱 第 897 頁的『配置佇列管理程式以接受 Multiplatforms 上的用戶端連線』。
2. 若要使用檔案庫 QMQMSAMP 之檔案 AMQSDATA 中的成員 PUT 中的資料來呼叫其中一個範例程式，請使用類似下列的指令:

   ```
   CALL PGM(QMQM/AMQSPUT4) PARM('QMQMSAMP/AMQSDATA(PUT)')
   ```

   註: 若要讓已編譯模組使用 IFS 檔案系統，請在 CRTCMOD 上指定選項 SYSIFCOPT (*IFSIO) ，然後必須以下列格式指定作為參數傳遞的檔名:

   ```
   home/me/myfile
   ```
3. 如果您要使用「查詢」、「設定」及「回應」範例的 COBOL 版本，請先變更程序定義，然後再執行這些範例。

   對於 Inquire、Set 和 Echo 範例，範例定義會觸發這些範例的 C 版本。 如果您想要 COBOL 版本，則必須變更程序定義:

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS
   - SYSTEM.SAMPLE.ECHOPROCESS

   在 IBM i 上，您可以使用 **CHGMQMPRC** 指令 (如需詳細資料，請參閱 變更 MQ 處理程序 (CHGMQMPRC)) ，或編輯並執行具有替代定義的 **AMQSAMP4** 指令。
4. 執行範例程式。

   如需每一個範例所預期參數的相關資訊，請參閱個別範例的說明。

   註: 對於 COBOL 範例程式，當您傳遞佇列名稱作為參數時，必須提供 48 個字元，並在必要時填補空白字元。 48 個字元以外的任何字元都會導致程式失敗，原因碼為 2085。

相關參考

第 894 頁的『IBM i 的範例』
IBM i 系統上 IBM MQ 的範例程式所示範的技術。

Linux AIX 在 *AIX and Linux* 上準備及執行範例程式

在 AIX and Linux 上執行範例程式之前，您必須先建立佇列管理程式，同時建立您需要的佇列。 如果您想要執行 COBOL 範例，則可能需要執行一些其他準備。

關於這項作業

如果在安裝時使用預設值，則 AIX and Linux 系統上的 IBM MQ 範例檔位於 第 900 頁的表 160 中列出的目錄。

表 160: 在 AIX and Linux 系統上尋找 IBM MQ 範例的位置	
內容	**目錄**
來源檔	*MQ_INSTALLATION_PATH*/samp
無法傳送郵件的佇列處理程式原始檔	*MQ_INSTALLATION_PATH*/samp/dlq
執行檔	*MQ_INSTALLATION_PATH*/samp/bin

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

範例需要一組佇列才能使用。 您可以使用自己的佇列或執行範例 MQSC 檔 amqscos0.tst 來建立集合。 若要執行範例，請使用提供的可執行版本，或使用 ANSI 編譯器來編譯來源版本，如同您使用任何其他應用程式一樣。

程序

1. 建立佇列管理程式並設定預設定義。

   您必須先執行此動作，才能執行任何範例程式。 如需建立佇列管理程式的相關資訊，請參閱 管理 IBM MQ。 如需配置佇列管理程式以安全地接受來自以用戶端模式執行之應用程式的送入連線要求的相關資訊，請參閱 第 897 頁的『配置佇列管理程式以接受 Multiplatforms 上的用戶端連線』。
2. 如果您不是使用自己的佇列，請執行範例 MQSC 檔 amqscos0.tst 來建立一組佇列。

   若要在 AIX and Linux 系統上執行此動作，請輸入:

   ```
   runmqsc QManagerName <amqscos0.tst > /tmp/sampobj.out
   ```

   請檢查 `sampobj.out` 檔，以確定沒有錯誤。
3. 如果您要使用「查詢」、「設定」及「回應」範例的 COBOL 版本，請先變更程序定義，然後再執行這些範例。

   對於 Inquire、Set 和 Echo 範例，範例定義會觸發這些範例的 C 版本。 如果您想要 COBOL 版本，則必須變更程序定義:

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS
   - SYSTEM.SAMPLE.ECHOPROCESS

   在 Windows 上，請先編輯 `amqscos0.tst` 檔，並將 C 執行檔名稱變更為 COBOL 執行檔名稱，然後再使用 **runmqsc** 指令來執行這些範例。
4. 執行範例程式。

   若要執行範例，請輸入其名稱後接任何參數，例如:

   ```
   amqsput myqueue qmanagername
   ```

   其中 *myqueue* 是要放置訊息的佇列名稱，而 *qmanagername* 是擁有 *myqueue* 的佇列管理程式。

   如需每一個範例所預期參數的相關資訊，請參閱個別範例的說明。

   註: 對於 COBOL 範例程式，當您傳遞佇列名稱作為參數時，必須提供 48 個字元，並在必要時填補空白字元。 48 個字元以外的任何字元都會導致程式失敗，原因碼為 2085。

相關參考

第 889 頁的『AIX and Linux 系統的範例』
IBM MQ for AIX or Linux 的範例程式所示範的技術。

Windows 在 *Windows* 上準備及執行範例程式

在 Windows 上執行範例程式之前，您必須先建立佇列管理程式，同時建立您需要的佇列。 如果您想要執行 COBOL 範例，則可能需要執行一些其他準備。

關於這項作業

如果在安裝時使用預設值，則 IBM MQ for Windows 範例檔位於 第 901 頁的表 161 中列出的目錄。 The installation drive defaults to <c:>.

表 *161:* 在何處尋找 *IBM MQ for Windows* 的範例

內容	**目錄**
C 原始碼	*MQ_INSTALLATION_PATH*\Tools\C\Samples
無法傳送郵件的處理程式範例的原始碼	*MQ_INSTALLATION_PATH*\Tools\C\Samples\DLQ
COBOL 原始碼	*MQ_INSTALLATION_PATH*\Tools\Cobol \ 範例
C 執行檔 [1]	*MQ_INSTALLATION_PATH*\ 工具 \C\Samples \ Bin (32 位元版本) *MQ_INSTALLATION_PATH*\ Tools\C\Samples\Bin64 (64 位元版本)
MQSC 檔案範例	*MQ_INSTALLATION_PATH*\Tools\MQSC\Samples
Visual Basic 原始碼	*MQ_INSTALLATION_PATH*\Tools\VB\SampVB6
.NET 範例	*MQ_INSTALLATION_PATH*\Tools\dotnet \ Samples

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

註: 部分 C 執行檔範例提供 64 位元版本。

範例需要一組佇列才能使用。 您可以使用自己的佇列，或執行範例 MQSC 檔 amqscos0.tst 來建立一組佇列。 如果要執行範例，請使用所提供的可執行版本，或像任何其他 IBM MQ for Windows 應用程式一樣編譯來源版本。

程序

1. 建立佇列管理程式並設定預設定義。

   您必須先執行此動作，才能執行任何範例程式。 如需建立佇列管理程式的相關資訊，請參閱 管理 IBM MQ。 如需配置佇列管理程式以安全地接受來自以用戶端模式執行之應用程式的送入連線要求的相關資訊，請參閱 第 897 頁的『配置佇列管理程式以接受 Multiplatforms 上的用戶端連線』。
2. 如果您不是使用自己的佇列，請執行範例 MQSC 檔 amqscos0.tst 來建立一組佇列。

   若要在 Windows 系統上執行此動作，請輸入:

   ```
   runmqsc QManagerName < amqscos0.tst > sampobj.out
   ```

   請檢查 `sampobj.out` 檔，以確定沒有錯誤。 此檔案位於現行目錄中。
3. 如果您要使用「查詢」、「設定」及「回應」範例的 COBOL 版本，請先變更程序定義，然後再執行這些範例。

   對於 Inquire、Set 和 Echo 範例，範例定義會觸發這些範例的 C 版本。 如果您想要 COBOL 版本，則必須變更程序定義:

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS
   - SYSTEM.SAMPLE.ECHOPROCESS

   在 Windows 上，請先編輯 `amqscos0.tst` 檔，並將 C 執行檔名稱變更為 COBOL 執行檔名稱，然後再使用 **runmqsc** 指令來執行這些範例。
4. 執行範例程式。

   若要執行範例，請輸入其名稱後接任何參數，例如:

   ```
   amqsput myqueue qmanagername
   ```

   其中 *myqueue* 是要放置訊息的佇列名稱，而 *qmanagername* 是擁有 *myqueue* 的佇列管理程式。

如需每一個範例所預期參數的相關資訊，請參閱個別範例的說明。

註: 對於 COBOL 範例程式，當您傳遞佇列名稱作為參數時，必須提供 48 個字元，並在必要時填補空白字元。 48 個字元以外的任何字元都會導致程式失敗，原因碼為 2085。

相關參考

第 892 頁的『IBM MQ for Windows 的範例』
IBM MQ for Windows 的範例程式所示範的技術。

第 894 頁的『IBM MQ for Windows 的 Visual Basic 範例』
Windows 系統上 IBM MQ 的範例程式所示範的技術。

API 結束程式範例程式

範例 API 結束程式會對使用者指定的檔案產生 MQI 追蹤，並在 **MQAPI_TRACE_LOGFILE** 環境變數中定義字首。

如需 API 結束程式的相關資訊，請參閱 第 798 頁的『在 Multiplatforms 上撰寫及編譯 API 結束程式』。

來源

amqsaxe0.c

二進位

阿姆克薩謝

配置範例結束程式

1. 將下列資訊新增至 qm.ini 檔案的 ApiExitLocal 段落 。

   Windows 以外的平台

   ```
   ApiExitLocal:
   Sequence=100
   Function=EntryPoint
   Module= MQ_INSTALLATION_PATH/samp/bin/amqsaxe
   Name=SampleApiExit
   ```

   其中 *MQ_INSTALLATION_PATH* 代表 IBM MQ 的安裝目錄。

   Windows

   ```
   ApiExitLocal:
   Sequence=100
   Function=EntryPoint
   Module= MQ_INSTALLATION_PATH\Tools\c\Samples\bin\amqsaxe
   Name=SampleApiExit
   ```

   其中 *MQ_INSTALLATION_PATH* 代表 IBM MQ 的安裝目錄。

2. 設定 **MQAPI_TRACE_LOGFILE** 環境變數

   ```
   MQAPI_TRACE_LOGFILE=/tmp/MqiTrace
   ```

3. 執行您的應用程式。

   輸出檔建立在 /tmp 目錄中，名稱如下: MqiTrace.*pid*.*tid*.log。

非同步使用範例程式

amqscbf 範例程式示範如何使用 MQCB 和 MQCTL 以非同步方式耗用來自多個佇列的訊息。

amqscbf 以 C 原始碼提供，以及在 AIX, Linux, and Windows 平台上的二進位用戶端和伺服器執行檔。

程式是從指令行啟動，並採用下列選用參數:

```
Usage: [Options] Queue Name {queue_name}
  where Options are:
    -m Queue Manager Name
```

```
    -o Open options
    -r Reconnect Type
       d Reconnect Disabled
       r Reconnect
       m Reconnect Queue Manager
```

提供多個佇列名稱以從多個佇列讀取訊息 (範例最多支援 10 個佇列)。

註: Reconnect type 僅適用於用戶端程式。

範例

此範例顯示 amqscbf 以伺服器程式身分執行，從 QL1 讀取一則訊息，然後停止。

使用「IBM MQ 檔案總管」，將測試訊息放置在 QL1 上。 按 Enter 鍵停止程式。

```
C:\>amqscbf QL1
Sample AMQSCBF0 start

Press enter to end
Message Call (9 Bytes) :
Message 1

Sample AMQSCBF0 end
```

amqscbf 示範的內容

此範例顯示如何依訊息到達的順序從多個佇列讀取訊息。 這將需要更多使用同步 MQGET 的程式碼。 在非同步使用的情況下，不需要輪詢，且 IBM MQ 會執行執行緒和儲存體管理。 「真實世界」範例需要處理錯誤; 在範例中，錯誤會寫出至主控台。

範例程式碼具有下列步驟:

1. 定義單一訊息使用回呼函數,

   ```
   void MessageConsumer(MQHCONN      hConn,
                        MQMD    * pMsgDesc,
                        MQGMO   * pGetMsgOpts,
                        MQBYTE  * Buffer,
                        MQCBC   * pContext)
   { ... }
   ```

2. 連接至佇列管理程式,

   ```
   MQCONNX(QMName,&cno,&Hcon,&CompCode,&CReason);
   ```

3. 開啟輸入佇列，並將每一個佇列與 MessageConsumer 回呼函數相關聯。

   ```
   MQOPEN(Hcon,&od,O_options,&Hobj,&OpenCode,&Reason);
   cbd.CallbackFunction = MessageConsumer;
   MQCB(Hcon,MQOP_REGISTER,&cbd,Hobj,&md,&gmo,&CompCode,&Reason);
   ```

   cbd.CallbackFunction 不需要針對每一個佇列設定; 它是僅供輸入的欄位。 但您可以將不同的回呼函數與每一個佇列相關聯。

4. 開始使用訊息,

   ```
   MQCTL(Hcon,MQOP_START,&ctlo,&CompCode,&Reason);
   ```

5. 等到使用者按下 Enter 鍵，然後停止使用訊息,

   ```
   MQCTL(Hcon,MQOP_STOP,&ctlo,&CompCode,&Reason);
   ```

6. 最後切斷與佇列管理程式的連線,

```
MQDISC(&Hcon,&CompCode,&Reason);
```

非同步 *Put* 範例程式

瞭解執行 amqsapt 範例及「非同步放置」範例程式的設計。

非同步放置範例程式會使用非同步 MQPUT 呼叫將訊息放置在佇列上，然後使用 MQSTAT 呼叫擷取狀態資訊。 如需此程式在不同平台上的名稱，請參閱 第 889 頁的『Multiplatforms 上範例程式中示範的特性』。

執行 amqsapt 範例

此程式最多使用 6 個參數:

1. 目標佇列的名稱 (必要)
2. 佇列管理程式的名稱 (選用)
3. 開啟選項 (選用)
4. 關閉選項 (選用)
5. 目標佇列管理程式的名稱 (選用)
6. 動態佇列的名稱 (選用)

如果未指定佇列管理程式，則 amqsapt 會連接至預設佇列管理程式。

非同步 Put 範例程式的設計

程式使用 MQOPEN 呼叫搭配提供的輸出選項，或搭配 MQOO_OUTPUT 及 MQOO_FAIL_IF_QUIESCING 選項，來開啟目標佇列以放置訊息。

如果無法開啟佇列，程式會輸出一則錯誤訊息，其中包含 MQOPEN 呼叫所傳回的原因碼。 為了讓程式保持簡單，在此及後續的 MQI 呼叫上，程式會對許多選項使用預設值。

對於每一行輸入，程式會將文字讀取到緩衝區中，並使用 MQPUT 呼叫搭配 MQPMO_ASYNC_RESPONSE 來建立資料包訊息，其中包含該行的文字，並非同步地將它放置到目標佇列中。 程式會繼續執行，直到到達輸入結尾或 MQPUT 呼叫失敗為止。 如果程式到達輸入結尾，則會使用 MQCLOSE 呼叫來關閉佇列。

然後，程式會發出 MQSTAT 呼叫，並傳回 MQSTS 結構，並顯示訊息，其中包含順利放置的訊息數、放置警告的訊息數，以及失敗次數。

瀏覽範例程式

「瀏覽」範例程式會使用 MQGET 呼叫來瀏覽佇列上的訊息。

如需這些程式的名稱，請參閱 第 889 頁的『Multiplatforms 上範例程式中示範的特性』。

瀏覽範例程式的設計

程式會使用 MQOPEN 呼叫及 MQOO_BROWSE 選項來開啟目標佇列。 如果無法開啟佇列，程式會輸出一則錯誤訊息，其中包含 MQOPEN 呼叫所傳回的原因碼。

對於佇列上的每一個訊息，程式會使用 MQGET 呼叫來複製佇列中的訊息，然後顯示訊息中包含的資料。 MQGET 呼叫使用下列選項:

MQGMO_BROWSE_NEXT
在 MQOPEN 呼叫之後，瀏覽游標在邏輯上位於佇列中第一個訊息之前，因此這個選項會在第一次呼叫時傳回 **第一個** 訊息。

MQGMO_NO_WAIT
如果佇列中沒有訊息，則程式不會等待。

MQGMO_ACCEPT_TRUNCATED_MSG
MQGET 呼叫指定固定大小的緩衝區。 如果訊息比這個緩衝區長，程式會顯示截斷的訊息，以及訊息已截斷的警告。

此程式示範如何必須在每一個 MQGET 呼叫之後清除 MQMD 結構的 *MsgId* 和 *CorrelId* 欄位，因為呼叫會將這些欄位設為它所擷取訊息中包含的值。 清除這些欄位表示後續的 MQGET 呼叫會依照訊息保留在佇列中的順序來擷取訊息。

程式會繼續到佇列結尾 ;MQGET 呼叫會傳回 MQRC_NO_MSG_AVAILABLE 原因碼，且程式會顯示警告訊息。 如果 MQGET 呼叫失敗，程式會顯示包含原因碼的錯誤訊息。

然後，程式會使用 MQCLOSE 呼叫來關閉佇列。

AIX, Linux, and Windows 的「瀏覽」程式範例
當瞭解 AIX, Linux, and Windows 上的「瀏覽」範例程式時，請考量使用本主題。

程式的 C 版本採用 2 個參數

1. 來源佇列的名稱 (必要)
2. 佇列管理程式的名稱 (選用)

如果未指定佇列管理程式，則會連接至預設佇列管理程式。 例如，輸入下列其中一項:

- `amqsgbr myqueue qmanagername`
- `amqsgbrc myqueue qmanagername`
- `amq0gbr0 myqueue`

其中 `myqueue` 是將從中檢視訊息的佇列名稱，而 `qmanagername` 是擁有 `myqueue` 的佇列管理程式。

如果您省略 `qmanagername`，則在執行 C 範例時，會假設預設佇列管理程式擁有佇列。

COBOL 版本沒有任何參數。 它會連接至預設佇列管理程式，當您執行它時，系統會提示您:

```
Please enter the name of the target queue
```

在這種情況下，只會顯示每則訊息的前 50 個字元，後面接著 `- - - truncated`。

IBM i 上的瀏覽範例程式
每一個程式會擷取您在呼叫程式時所指定佇列上所有訊息的副本; 這些訊息會保留在佇列上。

您可以使用所提供的佇列 SYSTEM.SAMPLE.LOCAL; 先執行「放置」範例程式，將部分訊息放置在佇列上。 您可以使用佇列 SYSTEM.SAMPLE.ALIAS，這是相同本端佇列的別名。 程式會繼續執行，直到到達佇列結尾或 MQI 呼叫失敗為止。

C 範例可讓您以類似於 Windows 系統範例的方式指定佇列管理程式名稱 (通常是第二個參數)。 例如:

```
CALL PGM(QMQM/AMQSTRG4) PARM('SYSTEM.SAMPLE.TRIGGER' 'QM01')
```

如果未指定佇列管理程式，則會連接至預設佇列管理程式。 這也與 RPG 範例相關。 不過，使用 RPG 範例時，您必須提供佇列管理程式名稱，而不是讓它成為預設值。

ALW 瀏覽器範例程式

瀏覽器範例程式會讀取及寫入佇列上所有訊息的訊息描述子及訊息內容欄位。

範例程式撰寫為公用程式，不只是為了示範技術。 如需這些程式的名稱，請參閱 第 889 頁的『Multiplatforms 上範例程式中示範的特性』。

此程式採用下列位置參數:

1. 來源佇列的名稱 (必要)
2. 佇列管理程式的名稱 (必要)
3. 內容的選用參數 (選用)

請使用下列環境變數來提供用來向佇列管理程式進行鑑別的認證:

MQSAMP_USER_ID
如果您想要使用使用者 ID 和密碼向佇列管理程式進行鑑別，請設為用於連線鑑別的使用者 ID。 程式會提示輸入密碼以隨附使用者 ID。

Linux AIX V 9.3.4 **MQSAMP_TOKEN**
如果您要提供鑑別記號以向佇列管理程式進行鑑別，請設為非空白值。 程式會提示輸入鑑別記號。 鑑別記號只能由使用用戶端連結的 **amqsbcgc** 範例使用。

若要執行這些程式，請輸入下列其中一個指令:

- `amqsbcg myqueue qmanagername`
- `amqsbcgc myqueue qmanagername`

其中 *myqueue* 是要瀏覽訊息的佇列名稱， *qmanagername* 是擁有 *myqueue* 的佇列管理程式。

它會從佇列中讀取每一個訊息，並將下列內容寫入 stdout:

- 格式化訊息描述子欄位
- 訊息資料 (以十六進位及字元格式 (可能的話) 傾出)

表 *162:* 內容參數允許的值

值	行為
0	預設行為。 遞送至應用程式的內容取決於從中擷取訊息的 **PropertyControl** 佇列屬性。

1	會建立訊息控點，並與 MQGET 搭配使用。 訊息的內容 (訊息描述子或延伸中包含的內容除外) 會以與訊息描述子類似的方式顯示。 例如:
	`****Message properties****`
	`  property name: property value`
	或者如果沒有可用的內容:
	`****Message properties****`
	`  None`
	使用 `printf` 顯示數值，字串值以單引號括住，位元組字串以 X 及單引號括住，就像訊息描述子一樣。

2	已指定 MQGMO_NO_PROPERTIES，因此只會傳回訊息描述子內容。
3	已指定 MQGMO_PROPERTIES_FORCE_MQRFH2，因此會在訊息資料中傳回所有內容。
4	已指定 MQGMO_PROPERTIES_COMPATIBILITY，因此可以根據是否包含 IBM MQ 內容來傳回所有內容，否則會捨棄這些內容。

程式限制只能列印訊息的前 65535 個字元，如果讀取較長的訊息，則失敗原因為 `truncated msg`。

如需此公用程式的輸出範例，請參閱 瀏覽佇列。

CICS 交易範例
提供範例 CICS 交易程式，名稱為 amqscic0.ccs (適用於原始碼) 及 amqscic0 (適用於執行檔版本)。 您可以使用標準 CICS 機能來建置交易。

如需平台所需指令的詳細資料，請參閱 第 837 頁的『建置程序化應用程式』。

交易會從傳輸佇列 SYSTEM.SAMPLE.CICS 讀取訊息。WORKQUEUE 在預設佇列管理程式上，並將它們放置在本端佇列上，其名稱包含在訊息的傳輸標頭中。 任何失敗都會傳送至佇列 SYSTEM.SAMPLE.CICS。DLQ。

註: 您可以使用範例 MQSC Script amqscic0.tst 來建立這些佇列及範例輸入佇列。

Connect 範例程式

「連接」範例程式可讓您從用戶端探索 MQCONNX 呼叫及其選項。 此範例會使用 MQCONNX 呼叫連接至佇列管理程式，並使用 MQINQ 呼叫查詢佇列管理程式的名稱，然後顯示它。 此外，請瞭解如何執行 amqscnxc 範例。

註: Connect 範例程式是用戶端範例。 您可以在伺服器上編譯並執行它，但該函數只在用戶端上才有意義，且只會提供用戶端執行檔。

執行 amqscnxc 範例

Connect 範例程式的指令行語法如下:

```
amqscnxc [-x ConnName [-c SvrconnChannelName]] [-u User] [QMgrName]
```

參數是選用的，其順序不重要，但 QMgrName 除外，如果指定的話，則必須是最後一個。 參數如下:

ConnName
伺服器佇列管理程式的 TCP/IP 連線名稱

如果未指定 TCP/IP 連線名稱，則會發出 MQCONNX 並將 *ClientConnPtr* 設為 NULL。

SvrconnChannel 名稱
伺服器連線通道的名稱

如果您指定 TCP/IP 連線名稱，但未指定伺服器連線通道 (不容許反向) ，則範例會使用名稱 SYSTEM.DEF.SVRCONN。

使用者
用於連線鑑別的使用者名稱

如果您指定此選項，則程式會提示輸入密碼以隨附該使用者 ID。

QMgrName
目標佇列管理程式的名稱

如果您未指定目標佇列管理程式，則範例會連接至以給定 TCP/IP 連線名稱接聽的任何佇列管理程式。

註: 如果您輸入問號作為唯一參數，或輸入不正確的參數，則會收到一則訊息，說明如何使用程式。

如果您在沒有指令行選項的情況下執行範例，則會使用 MQSERVER 環境變數的內容來判定連線資訊。 (在此範例中， MQSERVER 設為 SYSTEM.DEF.SVRCONN/TCP/machine.site.company.com。) 您會看到如下的輸出:

```
Sample AMQSCNXC start
Connecting to the default queue manager
with no client connection information specified.
Connection established to queue manager machine

Sample AMQSCNXC end
```

如果您執行範例並提供 TCP/IP 連線名稱及伺服器連線通道名稱，但沒有目標佇列管理程式名稱，如下所示:

```
amqscnxc -x machine.site.company.com -c SYSTEM.ADMIN.SVRCONN
```

會使用預設佇列管理程式名稱，且您會看到如下所示的輸出:

```
Sample AMQSCNXC start
Connecting to the default queue manager
using the server connection channel SYSTEM.ADMIN.SVRCONN
on connection name machine.site.company.com.
Connection established to queue manager MACHINE

Sample AMQSCNXC end
```

如果您執行範例並提供 TCP/IP 連線名稱及目標佇列管理程式名稱，如下所示:

```
amqscnxc -x machine.site.company.com MACHINE
```

您會看到如下所示的輸出:

```
Sample AMQSCNXC start
Connecting to queue manager MACHINE
using the server connection channel SYSTEM.DEF.SVRCONN
on connection name machine.site.company.com.
Connection established to queue manager MACHINE

Sample AMQSCNXC end
```

Data-Conversion 範例程式

資料轉換範例程式是資料轉換結束常式的架構。 瞭解資料轉換範例的設計。

如需這些程式的名稱，請參閱 第 889 頁的『Multiplatforms 上範例程式中示範的特性』 。

資料轉換範例的設計

每一個資料轉換結束常式都會轉換單一具名訊息格式。 這個 Skeleton 是用來作為資料轉換結束程式產生公用程式所產生之程式碼片段的封套。

公用程式會針對每一個資料結構產生一個程式碼片段; 數個這類結構組成一個格式，因此會將數個程式碼片段新增至此架構，以產生一個常式來執行整個格式的資料轉換。

然後，程式會檢查轉換是成功還是失敗，並將所需的值傳回給呼叫者。

資料庫協調範例

提供兩個範例，示範 IBM MQ 如何在相同的工作單元內協調 IBM MQ 更新項目及資料庫更新項目。

這些範例如下:

1. AMQSXAS0 (C) 或 AMQ0XAS0 (COBOL) ，其會更新 IBM MQ 工作單元內的單一資料庫。
2. AMQSXAG0 (在 C 中) 或 AMQ0XAG0 (在 COBOL 中)、 AMQSXAB0 (在 C 中) 或 AMQ0XAB0 (在 COBOL 中) ， 以及 AMQSXAF0 (在 C 中) 或 AMQ0XAF0 (在 COBOL 中) ，它們一起更新 IBM MQ 工作單元內的兩個資料庫，以顯示如何存取多個資料庫。 提供這些範例以顯示 MQBEGIN 呼叫、混合 SQL 及 IBM MQ 呼叫的用法，以及連接至資料庫的位置及時間。

第 909 頁的圖 128 顯示如何使用所提供的範例來更新資料庫:

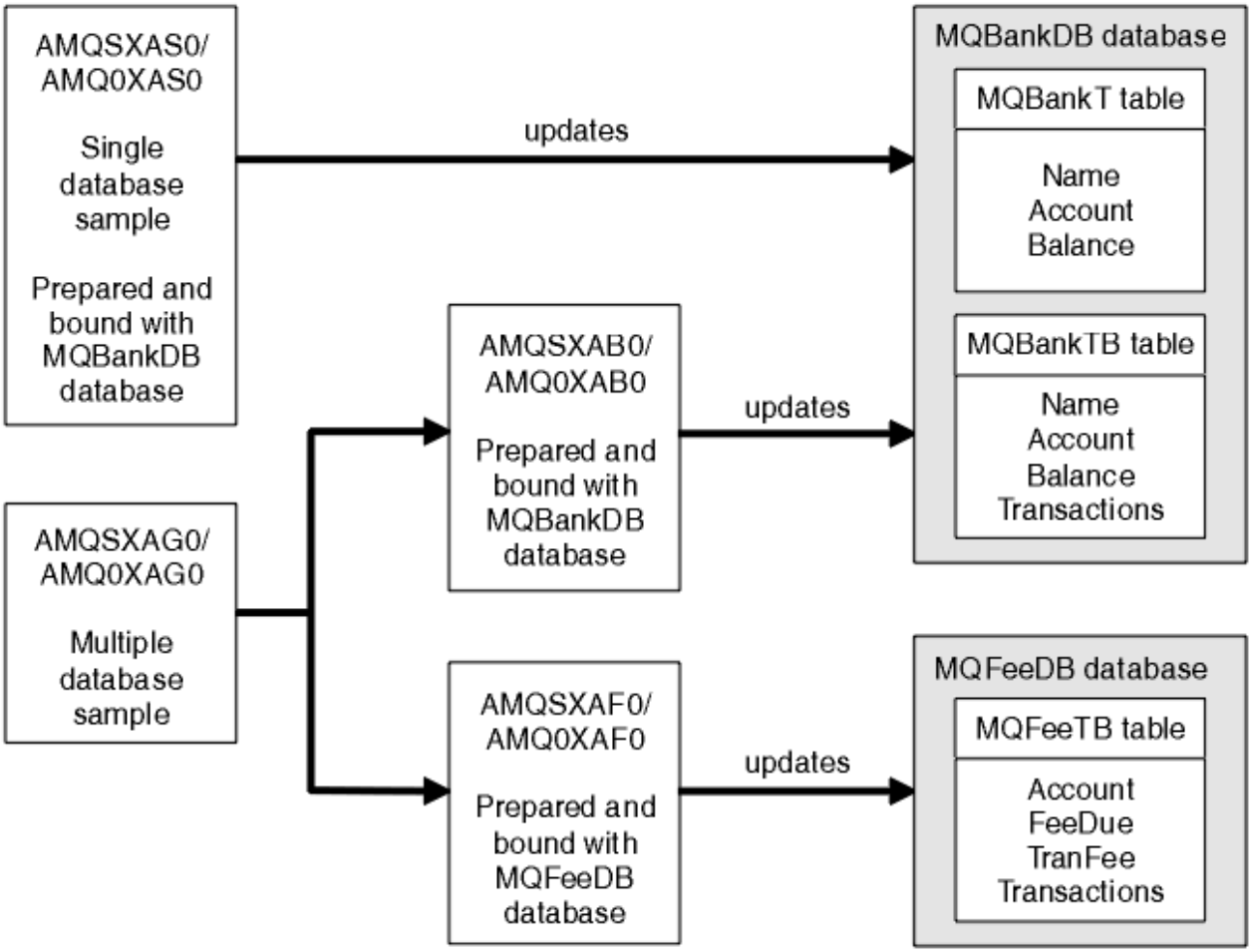


圖 *128:* 資料庫協調範例

程式會從佇列讀取訊息 (在同步點下)，然後使用訊息中的資訊，從資料庫取得相關資訊並更新它。 然後會列印資料庫的新狀態。

程式邏輯如下:

1. 使用來自程式引數的輸入佇列名稱
2. 使用 MQCONN 連接至預設佇列管理程式 (或選擇性地連接至 C 中提供的名稱)
3. 在沒有失敗的情況下開啟佇列 (使用 MQOPEN) 進行輸入
4. 使用 MQBEGIN 啟動工作單元
5. 從同步點下的佇列取得下一則訊息 (使用 MQGET)
6. 從資料庫取得資訊
7. 從資料庫更新資訊
8. 使用 MQCMIT 確定變更
9. 列印更新的資訊 (沒有可用的訊息計數為失敗，迴圈結束)
10. 使用 MQCLOSE 關閉佇列
11. 使用 MQDISC 切斷佇列的連線

在範例中使用 SQL 游標，以便在處理訊息時鎖定資料庫中的讀取 (即多個實例)，讓這些程式的多個實例同步執行。 游標會明確開啟，但 MQCMIT 呼叫會隱含地關閉。

單一資料庫範例 (AMQSXAS0 或 AMQ0XAS0) 沒有 SQL CONNECT 陳述式，且 IBM MQ 搭配 MQBEGIN 呼叫來隱含地建立資料庫連線。 多個資料庫範例 (AMQSXAG0 或 AMQ0XAG0、 AMQSXAB0 或 AMQ0XAB0 及 AMQSXAF0 或 AMQ0XAF0) 具有 SQL CONNECT 陳述式，因為部分資料庫產品僅容許一個作用中連線。 如

果不是資料庫產品的情況，或如果您存取多個資料庫產品中的單一資料庫，則可以移除 SQL CONNECT 陳述式。

這些範例是使用 IBM Db2 資料庫產品準備的，因此您可能需要修改它們，以使用其他資料庫產品。

SQL 錯誤檢查會使用 Db2 所提供的 UTIL.C 及 CHECKERR.CBL 中的常式。 在編譯和鏈結之前，必須先編譯或取代這些。

註: 如果您使用 Micro Focus COBOL 原始碼 CHECKERR.MFC 用於 SQL 錯誤檢查，您必須將程式 ID 變更為大寫 (即 CHECKERR) ， AMQ0XAS0 才能正確鏈結。

建立資料庫和表格

在編譯範例之前，請先建立資料庫和表格。

若要建立資料庫，請使用資料庫產品的一般方法，例如:

```
DB2 CREATE DB MQBankDB
DB2 CREATE DB MQFeeDB
```

使用 SQL 陳述式來建立表格，如下所示:

在 C 中:

```
EXEC SQL CREATE TABLE MQBankT(Name         VARCHAR(40) NOT NULL,
                              Account      INTEGER     NOT NULL,
                              Balance      INTEGER     NOT NULL,
                              PRIMARY KEY (Account));

EXEC SQL CREATE TABLE MQBankTB(Name         VARCHAR(40) NOT NULL,
                               Account      INTEGER     NOT NULL,
                               Balance      INTEGER     NOT NULL,
                               Transactions INTEGER,
                               PRIMARY KEY (Account));

EXEC SQL CREATE TABLE MQFeeTB(Account      INTEGER     NOT NULL,
                              FeeDue       INTEGER     NOT NULL,
                              TranFee      INTEGER     NOT NULL,
                              Transactions INTEGER,
                              PRIMARY KEY (Account));
```

在 COBOL 中:

```
EXEC SQL CREATE TABLE
  MQBankT(Name         VARCHAR(40) NOT NULL,
          Account      INTEGER     NOT NULL,
          Balance      INTEGER     NOT NULL,
          PRIMARY KEY (Account))
  END-EXEC.

EXEC SQL CREATE TABLE
  MQBankTB(Name         VARCHAR(40) NOT NULL,
           Account      INTEGER     NOT NULL,
           Balance      INTEGER     NOT NULL,
           Transactions INTEGER,
           PRIMARY KEY (Account))
  END-EXEC.

EXEC SQL CREATE TABLE
  MQFeeTB(Account      INTEGER   NOT NULL,
          FeeDue       INTEGER   NOT NULL,
          TranFee      INTEGER   NOT NULL,
          Transactions INTEGER,
          PRIMARY KEY (Account))
  END-EXEC.
```

使用 SQL 陳述式在表格中輸入資料，如下所示:

```
EXEC SQL INSERT INTO MQBankT VALUES ('Mr Fred Bloggs',1,0);
EXEC SQL INSERT INTO MQBankT VALUES ('Mrs S Smith',2,0);
EXEC SQL INSERT INTO MQBankT VALUES ('Ms Mary Brown',3,0);
⋮
```

```
EXEC SQL INSERT INTO MQBankTB VALUES ('Mr Fred Bloggs',1,0,0);
EXEC SQL INSERT INTO MQBankTB VALUES ('Mrs S Smith',2,0,0);
EXEC SQL INSERT INTO MQBankTB VALUES ('Ms Mary Brown',3,0,0);
⋮
EXEC SQL INSERT INTO MQFeeTB VALUES (1,0,50,0);
EXEC SQL INSERT INTO MQFeeTB VALUES (2,0,50,0);
EXEC SQL INSERT INTO MQFeeTB VALUES (3,0,50,0);
⋮
```

註: 若為 COBOL，請使用相同的 SQL 陳述式，但在每一行的結尾新增 END_EXEC 。

前置編譯、編譯及鏈結範例
瞭解 C 和 COBOL 中的前置編譯、編譯及鏈結範例。

前置編譯 .SQC 檔案 (在 C 中) 和 .SQB 檔案 (在 COBOL 中)，並根據適當的資料庫來連結它們，以產生 .C 或 .CBL 檔案。 若要這麼做，請使用資料庫產品的一般方法。

在 C 中前置編譯

```
db2 connect to MQBankDB
db2 prep AMQSXAS0.SQC
db2 connect reset

db2 connect to MQBankDB
db2 prep AMQSXAB0.SQC
db2 connect reset

db2 connect to MQFeeDB
db2 prep AMQSXAF0.SQC
db2 connect reset
```

在 COBOL 中前置編譯

```
db2 connect to MQBankDB
db2 prep AMQ0XAS0.SQB bindfile target ibmcob
db2 bind AMQ0XAS0.BND
db2 connect reset

db2 connect to MQBankDB
db2 prep AMQ0XAB0.SQB bindfile target ibmcob
db2 bind AMQ0XAB0.BND
db2 connect reset

db2 connect to MQFeeDB
db2 prep AMQ0XAF0.SQB bindfile target ibmcob
db2 bind AMQ0XAF0.BND
db2 connect reset
```

編譯及鏈結

下列範例指令使用符號 *DB2TOP* 及 *MQ_INSTALLATION_PATH*。 *DB2TOP* 代表 Db2 產品的安裝目錄。 *MQ_INSTALLATION_PATH* 代表 IBM MQ 安裝所在的高階目錄。

- AIX 在 AIX 上，目錄路徑為:

  ```
  /usr/lpp/db2_05_00
  ```

- Windows 在 Windows 系統上，目錄路徑取決於安裝產品時選擇的路徑。 如果您選擇預設值，則路徑如下:

  ```
  c:\sqllib
  ```

註: 在 Windows 系統上發出鏈結指令之前，請確定 LIB 環境變數包含 Db2 和 IBM MQ 程式庫的路徑。

將下列檔案複製到暫存目錄:

- IBM MQ 安裝架構中的 amqsxag0.c 檔

  註: 此檔案可在下列目錄中找到:

  – Linux AIX 在 AIX and Linux 系統上:

  ```
  MQ_INSTALLATION_PATH/samp/xatm
  ```

  – Windows 在 Windows 系統上:

  ```
  MQ_INSTALLATION_PATH\tools\c\samples\xatm
  ```

- 您透過前置編譯 .sqc 原始檔、 amqsxas0.sqc、 amqsxaf0.sqc 及 amqsxab0.sqc 所取得的 .c 檔案。
- Db2 安裝架構中的檔案 util.c 和 util.h 。

  註: 您可以在下列目錄中找到這些檔案:

  ```
  DB2TOP/samples/c
  ```

針對您正在使用的平台，使用下列編譯器指令來建置每一個 .c 檔案的物件檔:

- AIX AIX

  ```
  xlc_r -I MQ_INSTALLATION_PATH/inc -I DB2TOP/include -c -o
  FILENAME.o FILENAME.c
  ```

- Windows Windows 系統

  ```
  cl /c /I MQ_INSTALLATION_PATH\tools\c\include /I DB2TOP\include
  FILENAME.c
  ```

針對您所使用的平台，使用下列鏈結指令來建置 amqsxag0 執行檔:

- AIX AIX

  ```
  xlc_r -H512 -T512 -L DB2TOP/lib -ldb2 -L MQ_INSTALLATION_PATH/lib
  -lmqm util.o amqsxaf0.o amqsxab0.o amqsxag0.o -o amqsxag0
  ```

- Windows Windows 系統

  ```
  link util.obj amqsxaf0.obj amqsxab0.obj amqsxag0.obj mqm.lib db2api.lib
  /out:amqsxag0.exe
  ```

針對您正在使用的平台，使用下列編譯及鏈結指令來建置 amqsxas0 執行檔:

- AIX AIX

  ```
  xlc_r -H512 -T512 -L DB2TOP/lib -ldb2
  -L MQ_INSTALLATION_PATH/lib -lmqm util.o amqsxas0.o -o amqsxas0
  ```

- Windows Windows 系統

  ```
  link util.obj amqsxas0.obj mqm.lib db2api.lib /out:amqsxas0.exe
  ```

其他資訊

AIX 如果您正在使用 AIX，並且想要存取 Oracle，請使用 xlc_r 編譯器並鏈結至 libmqm_r.a。

執行範例
使用此資訊來瞭解如何在 C 和 COBOL 上執行資料庫協調範例之前配置佇列管理程式。

在執行範例之前，請使用您正在使用的資料庫產品來配置佇列管理程式。 如需如何執行此動作的相關資訊，請參閱 實務範例 1: 佇列管理程式執行協調。

下列標題提供如何在 C 和 COBOL 中執行範例的相關資訊:

- 第 913 頁的『C 範例』
- 第 914 頁的『COBOL 範例』

C 範例

訊息必須採用下列格式才能從佇列讀取:

```
UPDATE Balance change=nnn WHERE Account=nnn
```

AMQSPUT 可用來將訊息放置在佇列上。

資料庫協調範例採用兩個參數:

1. 佇列名稱 (必要)
2. 佇列管理程式名稱 (選用)

假設您已針對稱為 singDBQM 的單一資料庫範例建立並配置佇列管理程式，並使用稱為 singDBQ 的佇列，則會將 Fred Bloggs 的帳戶增量 50，如下所示:

```
AMQSPUT singDBQ singDBQM
```

然後，下列訊息中的索引鍵:

```
UPDATE Balance change=50 WHERE Account=1
```

您可以在佇列上放置多則訊息。

```
AMQSXAS0 singDBQ singDBQM
```

然後會列印 Fred Bloggs 先生帳戶的更新狀態。

假設您已使用稱為 multDBQ 的佇列建立並配置多資料庫範例 multDBQM 的佇列管理程式，您會將 Mary Brown 女士的帳戶減少 75，如下所示:

```
AMQSPUT multDBQ multDBQM
```

然後，下列訊息中的索引鍵:

```
UPDATE Balance change=-75 WHERE Account=3
```

您可以在佇列上放置多則訊息。

```
AMQSXAG0 multDBQ multDBQM
```

然後會列印 Mary Brown 女士帳戶的更新狀態。

COBOL 範例

訊息必須採用下列格式才能從佇列讀取:

```
UPDATE Balance change=snnnnnnnn WHERE Account=nnnnnnnn
```

為了簡單起見， `Balance change` 必須是帶正負號的 8 個字元的數字，而 `Account` 必須是 8 個字元的數字。

範例 AMQSPUT 可用來將訊息放置在佇列上。

範例不採用任何參數，並使用預設佇列管理程式。 它可以配置為隨時只執行其中一個範例。 假設您已使用稱為 singDBQ 的佇列來配置單一資料庫範例的預設佇列管理程式，則會將 Fred Bloggs 的帳戶增加 50 ，如下所示:

```
AMQSPUT singDBQ
```

然後，下列訊息中的索引鍵:

```
UPDATE Balance change=+00000050 WHERE Account=00000001
```

您可以在佇列上放置多則訊息:

```
AMQ0XAS0
```

輸入佇列的名稱:

```
singDBQ
```

然後會列印 Fred Bloggs 先生帳戶的更新狀態。

假設您已使用稱為 multDBQ 的佇列配置多個資料庫範例的預設佇列管理程式，您會將 Mary Brown 女士的帳戶減少 75 ，如下所示:

```
AMQSPUT multDBQ
```

然後，下列訊息中的索引鍵:

```
UPDATE Balance change=-00000075 WHERE Account=00000003
```

您可以在佇列上放置多則訊息:

```
AMQ0XAG0
```

輸入佇列的名稱:

```
multDBQ
```

然後會列印 Mary Brown 女士帳戶的更新狀態。

無法傳送郵件的佇列處理程式範例

提供範例無法傳送郵件的佇列處理程式，執行檔版本的名稱為 `amqsdlq`。 如果您想要無法傳送郵件的佇列處理程式不同於 **RUNMQDLQ**，則可以使用範例來源作為基本程式。

範例與產品內提供的無法傳送郵件的處理程式類似，但追蹤與錯誤報告不同。 有兩個環境變數可供您使用:

ODQ_TRACE
設為 YES 或 `yes` 以開啟追蹤。

ODQ_MSG
設為包含錯誤及參考訊息的檔案名稱。 提供的檔案稱為 amqsdlq.msg。

視您的平台而定，您需要使用 **export** 或 **set** 指令，讓環境知道這些變數; 使用 **unset** 指令會關閉追蹤。

您可以修改錯誤訊息檔案 amqsdlq.msg，以符合您自己的需求。 此範例會將訊息放置在 stdout 中， **不** 放置在 IBM MQ 錯誤日誌檔中。

如需無法傳送郵件的處理程式如何運作以及如何執行它的相關資訊，請參閱適用於您平台的 處理 IBM MQ 無法傳送郵件的佇列上的訊息 或 系統管理手冊 。

「配送清單」範例程式

「配送清單」範例 amqsptl0 提供在數個訊息佇列上放置訊息的範例。 它是以 MQPUT 範例 amqsput0 為基礎。

執行「配送清單」範例 amqsptl0

「配送清單」樣本的執行方式與「放置」樣本類似。

它採用下列參數:

- 佇列的名稱
- 佇列管理程式的名稱

這些值會成對輸入。 例如:

```
amqsptl0 queue1 qmanagername1 queue2 qmanagername2
```

使用 MQOPEN 開啟佇列，並使用 MQPUT 將訊息放置到佇列。 如果無法辨識任何佇列或佇列管理程式名稱，則會傳回原因碼。

請記得定義佇列管理程式之間的通道，以便訊息可以在它們之間流動。 範例程式不會為您這麼做。

配送清單範例的設計

「放置訊息記錄 (MQPMR)」指定每一個目的地的訊息屬性。 此範例提供 *MsgId* 和 *CorrelId* 的值，這些會置換 MQMD 結構中指定的值。

MQPMO 結構中的 *PutMsgRecFields* 欄位指出 MQPMR 中存在哪些欄位:

```
MQLONG PutMsgRecFields=MQPMRF_MSG_ID + MQPMRF_CORREL_ID;
```

接下來，範例會配置回應記錄和物件記錄。 物件記錄 (MQOR) 至少需要一對名稱及偶數名稱，即 *ObjectName* 及 *ObjectQMgrName*。

下一個階段涉及使用 MQCONN 來連接至佇列管理程式。 範例會嘗試連接至與 MQOR 中第一個佇列相關聯的佇列管理程式; 如果這樣做失敗，它會依序通過物件記錄。 系統會通知您是否無法連接任何佇列管理程式及程式結束程式。

使用 MQOPEN 開啟目標佇列，並使用 MQPUT 將訊息放置到這些佇列。 任何問題及失敗都會在回應記錄 (MQRR) 中報告。

最後，會使用 MQCLOSE 來關閉目標佇列，且程式會使用 MQDISC 來切斷與佇列管理程式的連線。 對於每一個陳述 *CompCode* 和 *Reason* 的呼叫，會使用相同的回應記錄。

Echo 範例程式

「回應」範例程式會將訊息從訊息佇列回應至回覆佇列。

如需這些程式的名稱，請參閱 第 889 頁的『Multiplatforms 上範例程式中示範的特性』 。

這些程式預期以觸發程式的形式執行。

在 IBM i AIX, Linux, and Windows 系統上，其唯一輸入是包含目標佇列及佇列管理程式名稱的 MQTMC2 (觸發訊息) 結構。 COBOL 版本使用預設佇列管理程式。

IBM i 在 IBM i 上，若要讓觸發程序運作，請確定您要使用的「回應」範例程式是由到達佇列 SYSTEM.SAMPLE.ECHO。 若要執行此動作，請在程序定義 SYSTEM.SAMPLE.ECHOPROCESS 的 *ApplId* 欄位中指定您要使用的 Echo 範例程式名稱。 (因此，您可以使用 CHGMQMPRC 指令; 如需詳細資料，請參閱變更 MQ 處理程序 (CHGMQMPRC)。) 範例佇列具有觸發類型 FIRST ，因此，如果在您執行「要求」範例之前佇列上已有訊息，則您傳送的訊息不會觸發「回應」範例。

當您正確設定定義時，請先在一個工作中啟動 AMQSERV4 ，然後在另一個工作中啟動 AMQSREQ4 。 您可以使用 AMQSTRG4 而非 AMQSERV4，但潛在的工作提交延遲可能會讓您更不容易追蹤發生的情況。

使用要求範例程式，將訊息傳送至佇列 SYSTEM.SAMPLE.ECHO。 「回應範例」程式會將包含要求訊息中資料的回覆訊息傳送至要求訊息中指定的回覆目的地佇列。

Echo 範例程式的設計

程式會開啟在啟動時所傳遞的觸發訊息結構中指名的佇列。 (為了清楚說明，這稱為要求佇列。) 程式使用 MQOPEN 呼叫來開啟此佇列以供共用輸入。

程式會使用 MQGET 呼叫來移除此佇列中的訊息。 此呼叫使用 MQGMO_ACCEPT_TRUNCATED_MSG、MQGMO_CONVERT 及 MQGMO_WAIT 選項，等待間隔為 5 秒。 程式會測試每一個訊息的描述子，以查看它是否為要求訊息; 如果不是，則程式會捨棄訊息並顯示警告訊息。

然後，針對每一行輸入，程式會將文字讀取到緩衝區中，並使用 MQPUT1 呼叫，將包含該行文字的要求訊息放入回覆目的地佇列中。

如果 MQGET 呼叫失敗，程式會將報告訊息放在回覆目的地佇列中，並將訊息描述子的 *Feedback* 欄位設為 MQGET 所傳回的原因碼。

當要求佇列中沒有剩餘訊息時，程式會關閉該佇列，並切斷與佇列管理程式的連線。

IBM i 在 IBM i 上，程式也可以回應從非 IBM MQ for IBM i 平台傳送至佇列的訊息，但不會針對此狀況提供任何範例。 若要讓 ECHO 程式運作，請執行下列動作:

- 撰寫程式，並正確地指定 **Format**、 **Encoding** 及 **CCSID** 參數，以傳送文字要求訊息。

  ECHO 程式會要求佇列管理程式執行訊息資料轉換 (如果需要的話)。
- 如果您撰寫的程式未提供類似的回覆轉換，請在 IBM MQ for IBM i 傳送通道上指定 CONVERT (*YES)。

Get 範例程式

Get 範例程式會使用 MQGET 呼叫從佇列取得訊息。

如需這些程式的名稱，請參閱 第 889 頁的『Multiplatforms 上範例程式中示範的特性』 。

Get 範例程式的設計

程式會使用 MQOPEN 呼叫搭配 MQOO_INPUT_AS_Q_DEF 選項來開啟目標佇列。 如果無法開啟佇列，程式會顯示一則錯誤訊息，其中包含 MQOPEN 呼叫所傳回的原因碼。

對於佇列上的每一個訊息，程式會使用 MQGET 呼叫來移除佇列中的訊息，然後顯示訊息中包含的資料。 MQGET 呼叫會使用 MQGMO_WAIT 選項，並指定 *WaitInterval* 為 15 秒，因此如果佇列上沒有訊息，程式會等待此期間。 如果在此間隔到期之前沒有任何訊息到達，則呼叫會失敗並傳回 MQRC_NO_MSG_AVAILABLE 原因碼。

該程式示範如何在每一個 MQGET 呼叫之後清除 MQMD 結構的 *MsgId* 和 *CorrelId* 欄位，因為呼叫會將這些欄位設為它所擷取訊息中包含的值。 清除這些欄位表示後續的 MQGET 呼叫會依照訊息保留在佇列中的順序來擷取訊息。

MQGET 呼叫指定固定大小的緩衝區。 如果訊息長於此緩衝區，則呼叫會失敗且程式會停止。

程式會繼續執行，直到 MQGET 呼叫傳回 MQRC_NO_MSG_AVAILABLE 原因碼或 MQGET 呼叫失敗為止。 如果呼叫失敗，程式會顯示包含原因碼的錯誤訊息。

然後，程式會使用 MQCLOSE 呼叫來關閉佇列。

執行 *amqsget* 和 *amqsgetc* 範例

這些程式各採用下列位置參數:

1. 來源佇列的名稱 (必要)
2. 佇列管理程式的名稱 (選用)

   如果未指定佇列管理程式，則「**amqsget**」會連接至預設佇列管理程式，而「**amqsgetc**」會連接至 *MQSERVER* 環境變數或用戶端通道定義檔所識別的佇列管理程式。
3. 開啟選項 (選用)

   如果未指定開啟選項，則範例會使用值 8193 ，這是這兩個選項的組合:

   - MQOO_INPUT_AS_Q_DEF
   - MQOO_FAIL_IF_QUIESCING
4. 關閉選項 (選用)

   如果未指定 close 選項，則範例會使用值 0 (即 MQCO_NONE)。

請使用下列環境變數來提供用來向佇列管理程式進行鑑別的認證:

MQSAMP_USER_ID
如果您想要使用使用者 ID 和密碼向佇列管理程式進行鑑別，請設為用於連線鑑別的使用者 ID。 程式會提示輸入密碼以隨附使用者 ID。

Linux AIX V 9.3.4 **MQSAMP_TOKEN**
如果您要提供鑑別記號以向佇列管理程式進行鑑別，請設為非空白值。 程式會提示輸入鑑別記號。 鑑別記號只能由使用用戶端連結的 **amqsgetc** 範例使用。

若要執行這些程式，請輸入下列其中一項:

- `amqsget myqueue qmanagername`
- `amqsgetc myqueue qmanagername`

其中 *myqueue* 是程式從中取得訊息的佇列名稱， *qmanagername* 是擁有 *myqueue* 的佇列管理程式。

使用 amqsget 和 amqsgetc

請注意， **amqsget** 會執行與佇列管理程式的本端連線，並使用共用記憶體來連接至佇列管理程式，因此只能在佇列管理程式所在的系統上執行，而 **amqsgetc** 會執行用戶端樣式連線 (即使連接至相同系統上的佇列管理程式也一樣)。

使用 **amqsgetc** 時，您需要根據佇列管理程式主機或 IP 位址及佇列管理程式接聽器埠，提供如何實際抵達佇列管理程式的應用程式詳細資料。

通常可以使用 **MQSERVER** 環境變數或使用用戶端通道定義表來定義連線詳細資料，也可以使用環境變數提供給 **amqsgetc** ; 例如，請參閱 MQCCDTURL。

使用「**MQSERVER**」在本端連接至佇列管理程式的範例，其具有在埠 1414 上執行且使用預設伺服器連線通道的接聽器:

```
export MQSERVER="SYSTEM.DEF.SVRCONN/TCP/ localhost(1414)"
```

高可用性範例程式

amqsghac、 **amqsphac** 和 **amqsmhac** 高可用性範例程式會使用自動化用戶端重新連線，來示範佇列管理程式失敗之後的回復。 **amqsfhac** 會檢查使用網路儲存體的佇列管理程式是否在失敗之後維護資料完整性。

amqsghac、 **amqsphac** 及 **amqsmhac** 程式是從指令行啟動，可結合使用，以示範多重實例佇列管理程式的一個實例失敗之後的重新連線。

或者，您也可以使用 **amqsghac**、 **amqsphac** 及 **amqsmhac** 範例來示範用戶端重新連線至單一實例佇列管理程式 (通常配置到佇列管理程式群組中)。

為了讓範例保持簡單，以便易於配置，您會看到範例程式重新連接至已啟動、已停止然後再次重新啟動的單一實例佇列管理程式; 請參閱 第 919 頁的『設定並控制佇列管理程式』。

將 **amqsfhac** 與 **amqmfsck** 平行使用，以檢查檔案系統完整性。 如需相關資訊，請參閱 **amqmfsck** (檔案系統檢查) 及 驗證共用檔案系統行為 。

amqsphac *queueName* [*qMgrName*]

- **amqsphac** 是 IBM MQ MQI client 應用程式。 它會將一連串訊息放入佇列中，每則訊息之間會延遲兩秒，並顯示傳送至其事件處理程式的事件。
- 沒有同步點用來將訊息放入佇列。
- 可以重新連線至相同佇列管理程式群組中的任何佇列管理程式。

amqsghac *queueName* [*qMgrName*]

- **amqsghac** 是 IBM MQ MQI client 應用程式。 它會從佇列取得訊息，並顯示傳送至其事件處理程式的事件。
- 沒有同步點可用來從佇列取得訊息。
- 可以重新連線至相同佇列管理程式群組中的任何佇列管理程式。

amqsmhac -s *sourceQueueName* -t *targetQueueName* [-m *qMgrName*] [-w *waitInterval*]

- **amqsmhac** 是 IBM MQ MQI client 應用程式。 它會在程式完成之前收到最後一則訊息之後 15 分鐘的預設等待間隔，將訊息從一個佇列複製到另一個佇列。
- 在同步點內複製訊息。
- 只能重新連線至相同的佇列管理程式。

amqsfhac *QueueManagerName QueueName SideQueueName InTransactionCount RepeatCount* (0 | 1 | 2)

- **amqsfhac** 是 IBM MQ MQI client 應用程式。 它會檢查使用網路儲存體 (例如 NAS 或叢集檔案系統) 的 IBM MQ 多重實例佇列管理程式是否維護資料完整性。 遵循 驗證共用檔案系統行為中的 **amqsfhac** 執行步驟。
- 當連接至 *QueueManager* 名稱時，它會使用 MQCNO_RECONNECT_Q_MGR 選項。 當佇列管理程式失效接手時，它會自動重新連接。
- 它會將 *InTransactionCount*RepeatCount* 持續訊息放置到 *QueueName* ，在此期間您會導致佇列管理程式失效接手任意次數。 「 **amqsfhac** 」每次都會重新連接至佇列管理程式，並繼續進行。 測試是確保不會遺失任何訊息。
- *InTransaction* 計數 訊息會放置在每一個交易內。 交易會重複 *RepeatCount* 次。 如果交易內發生失敗，當 **amqsfhac** 重新連接至佇列管理程式時， **amqsfhac** 會回復並重新提交交易。
- 它也會將訊息放入 *SideQueue* 名稱。 它使用 *SideQueueName* 來檢查是否已順利從 *QueueName* 確定或回復所有訊息。 如果它偵測到不一致，則會寫出錯誤訊息。
- 將最後一個參數設為 (0|1|2)，以改變 **amqsfhac** 的輸出追蹤量。

  0
  最少輸出。

  1
  輸出中。

  2
  大部分輸出。

配置用戶端連線

您需要配置用戶端和伺服器連線通道，才能執行範例。 用戶端驗證程序說明如何設定用戶端測試環境。

或者，使用下列範例中提供的配置。

使用 amqsghac、 amqsphac 和 amqsmhac 的範例

此範例示範使用單一實例佇列管理程式可重新連接的用戶端。

訊息由 **amqsphac** 放置在佇列 SOURCE 上，由 **amqsmhac** 傳送至 TARGET ， 並由 **amqsghac** 從 TARGET 擷取; 請參閱 第 919 頁的圖 129。

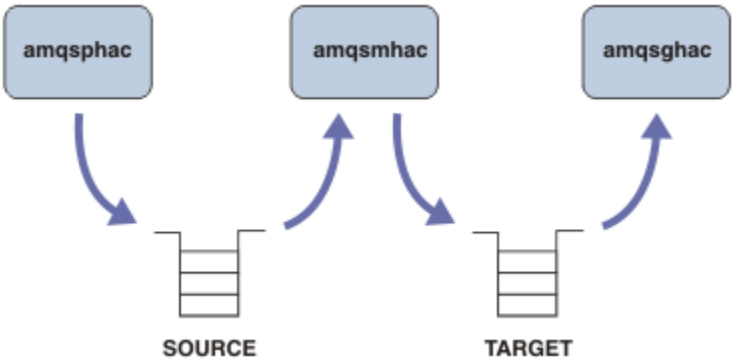


圖 *129:* 可重新連接的用戶端範例

請遵循下列步驟來執行範例。

1. 建立包含下列指令的檔案 hasamples.tst :

```
DEFINE QLOCAL(SOURCE) REPLACE
DEFINE QLOCAL(TARGET) REPLACE
DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN)  TRPTYPE(TCP) +
       MCAUSER(MUSR_MQADMIN) REPLACE
DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
       CONNAME('LOCALHOST(2345)') QMNAME(QM1) REPLACE
ALTER  LISTENER(SYSTEM.DEFAULT.LISTENER.TCP) TRPTYPE(TCP) +
       PORT(2345)
START  LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)
START  CHANNEL(CHANNEL1)
```

2. 在命令提示字元鍵入下列指令:
   a. crtmqm QM1
   b. strmqm QM1
   c. runmqsc QM1 < hasamples.tst
3. 將環境變數 **MQCHLLIB** 設為 AMQCLCHL.TAB 用戶端通道定義檔的路徑; 例如 SET MQCHLLIB=C:\IBM\MQ\MQ7\Data\qmgrs\QM1\@ipcc。
4. 在已設定 **MQCHLLIB** 的情況下開啟三個新視窗; 例如在 Windows 上，在前一個命令提示字元中鍵入 **start** 三次，以啟動其中一個視窗中的每一個程式。 請參閱 第 919 頁的『設定並控制佇列管理程式』中的步驟 第 920 頁的『5』 。)
5. 鍵入指令 endmqm -r -p QM1 以停止佇列管理程式，然後容許用戶端重新連接。
6. 鍵入指令 strmqm QM1 ， 以重新啟動佇列管理程式。

下列範例顯示在 Windows 上執行 **amqsghac**、 **amqsphac** 及 **amqsmhac** 範例的結果。

設定並控制佇列管理程式

1. 建立佇列管理程式。

```
C:\> crtmqm QM1
IBM MQ queue manager created.
Directory 'C:\IBM\MQ\MQ7\Data\qmgrs\QM1' created.
Creating or replacing default objects for QM1.
Default objects statistics : 67 created. 0 replaced. 0 failed.
Completing setup.
Setup completed.
```

請記住稍後要設定 **MQCHLLIB** 變數的資料目錄。

2. 啟動佇列管理程式。

```
C:\> strmqm QM1

IBM MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
IBM MQ queue manager 'QM1' started.
```

3. 建立佇列和通道，修改接聽器埠，並啟動接聽器和通道。

```
C:\> runmqsc QM1 < hasamples.tst

5724-H72 (C) Copyright IBM Corp. 1994, 2024.  ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM1.

     1 : DEFINE QLOCAL(SOURCE) REPLACE
AMQ8006: IBM MQ queue created.
     2 : DEFINE QLOCAL(TARGET) REPLACE
AMQ8006: IBM MQ queue created.
     3 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN)   TRPTYPE(TCP) MCAUSER(MUSR_MQADMIN)
REPLACE
AMQ8014: IBM MQ channel created.
     4 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN)  TRPTYPE(TCP) CONNAME('LOCALHOST(2345)')
QMNAME(QM1) REPLACE
AMQ8014: IBM MQ channel created.
     5 : ALTER LISTENER(SYSTEM.DEFAULT.LISTENER.TCP) TRPTYPE(TCP) PORT(2345)
AMQ8623: IBM MQ listener changed.
     6 : START LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)
AMQ8021: Request to start IBM MQ Listener accepted.
     7 : START CHANNEL(CHANNEL1)
AMQ8018: Start IBM MQ channel accepted.
7 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

4. 讓用戶端知道用戶端通道表格。

使用步驟 第 919 頁的『1』 中從 **crtmqm** 指令傳回的資料目錄，並在其中新增目錄 `@ipcc` 以設定 **MQCHLLIB** 變數。

```
C:\> SET MQCHLLIB=C:\IBM\MQ\MQ7\Data\qmgrs\QM1\@ipcc
```

5. 在其他視窗中啟動範例程式

```
C:\> start amqsphac SOURCE QM1
C:\> start amqsmhac -s SOURCE -t TARGET -m QM1
C:\> start amqsghac TARGET QM1
```

6. 請結束佇列管理程式，然後重新啟動它。

```
C:\> endmqm -r -p QM1

Waiting for queue manager 'QM1' to end.
IBM MQ queue manager 'QM1' ending.
IBM MQ queue manager 'QM1' ended.


C:\> strmqm QM1

IBM MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
IBM MQ queue manager 'QM1' started.
```

amqsphac

```
Sample AMQSPHAC start
target queue is SOURCE
message Message 1
message Message 2
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnectedmessage
Message 3
message Message 4
message Message 5
```

amqsmhac

```
Sample AMQSMHA0 start
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnected
No more messages.
Sample AMQSMHA0 end
C:\>
```

amqsghac

```
Sample AMQSGHAC start
message Message 1
message Message 2
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnected
message Message 3
message Message 4
message Message 5
```

相關工作

驗證共用檔案系統行為

相關參考

amqmfsck (檔案系統檢查)

Inquire 範例程式

Inquire 範例程式會使用 MQINQ 呼叫來查詢佇列的部分屬性。

如需這些程式的名稱，請參閱 第 889 頁的『Multiplatforms 上範例程式中示範的特性』 。

這些程式預期作為觸發程式執行，因此它們的唯一輸入是 IBM MQ for Multiplatforms 的 MQTMC2 (觸發訊息) 結構。 此結構包含具有要查詢之屬性的目標佇列名稱。 C 版本也會使用佇列管理程式名稱。 COBOL 版本使用預設佇列管理程式。

若要讓觸發處理程序運作，請確定您要使用的「查詢範例」程式是由到達佇列 SYSTEM.SAMPLE.INQ。 若要這麼做，請在程序定義 SYSTEM.SAMPLE.INQPROCESS 的 *ApplicId* 欄位中指定您要使用的 Inquire sample 程式名稱。 IBM i 若為 IBM i，您可以對此使用 CHGMQMPRC 指令; 如需詳細資料，請參閱變更 MQ 處理程序 (CHGMQMPRC)。 範例佇列的觸發類型為 FIRST; 如果在執行要求範例之前佇列上已有訊息，則您傳送的訊息不會觸發查詢範例。

當您已正確設定定義時:

- ALW 若為 AIX, Linux, and Windows，請在一個階段作業中啟動 **runmqtrm** 程式，然後在另一個階段作業中啟動 `amqsreq` 程式。

- IBM i 若為 IBM i，請在一個階段作業中啟動 AMQSERV4 程式，然後在另一個階段作業中啟動 AMQSREQ4 程式。 您可以使用 AMQSTRG4 而非 AMQSERV4，但潛在的工作提交延遲可能會讓您更不容易追蹤發生的情況。

使用「要求範例」程式，將要求訊息 (每一個都只包含佇列名稱) 傳送至佇列 SYSTEM.SAMPLE.INQ。 對於每一個要求訊息，「查詢範例」程式會傳送回覆訊息，其中包含要求訊息中所指定佇列的相關資訊。 回覆會傳送至要求訊息中指定的回覆目的地佇列。

IBM i 在 IBM i 上，如果範例輸入檔成員 QMQMSAMP.AMQSDATA(INQ)，且最後一個佇列不存在，因此範例會傳回失敗的報告訊息及原因碼。

Inquire sample 程式的設計

程式會開啟在啟動時所傳遞的觸發訊息結構中指名的佇列。 (為了明確起見，我們將此稱為 要求佇列。) 程式使用 MQOPEN 呼叫來開啟此佇列以供共用輸入。

程式會使用 MQGET 呼叫來移除此佇列中的訊息。 此呼叫使用 MQGMO_ACCEPT_TRUNCATED_MSG 和 MQGMO_WAIT 選項，等待間隔為 5 秒。 程式會測試每一個訊息的描述子，以查看它是否為要求訊息; 如果不是，則程式會捨棄訊息並顯示警告訊息。

對於從要求佇列中移除的每一個要求訊息，程式會讀取佇列的名稱 (我們將呼叫 目標佇列) 包含在資料中，並使用 MQOO_INQ 選項搭配 MQOPEN 呼叫來開啟該佇列。 然後，程式會使用 MQINQ 呼叫來查詢目標佇列的 *InhibitGet*、 **CurrentQDepth** 及 **OpenInputCount** 屬性值。

如果 MQINQ 呼叫成功，則程式會使用 MQPUT1 呼叫，將回覆訊息放置在回覆目的地佇列上。 此訊息包含三個屬性的值。

如果 MQOPEN 或 MQINQ 呼叫不成功，則程式會使用 MQPUT1 呼叫，將報告訊息放置在回覆目的地佇列上。 在此報告訊息之訊息描述子的 *Feedback* 欄位中，是 MQOPEN 或 MQINQ 呼叫所傳回的原因碼，視失敗的原因碼而定。

在 MQINQ 呼叫之後，程式會使用 MQCLOSE 呼叫來關閉目標佇列。

當要求佇列中沒有剩餘訊息時，程式會關閉該佇列，並切斷與佇列管理程式的連線。

訊息控點範例程式的查詢內容

AMQSIQMA 是一個範例 C 程式，可從訊息佇列查詢訊息控點的內容，並且是使用 MQINQMP API 呼叫的範例。

此範例會建立訊息控點，並將它放入 MQGMO 結構的 MsgHandle 欄位中。 然後，範例會取得一則訊息，並查詢並列印已移入訊息控點的所有內容。

```
C:\Program Files\IBM\MQ\tools\c\Samples\Bin >amqsiqm Q QM1
Sample AMQSIQMA start
property name MyProp value MyValue
message text Hello world!
Sample AMQSIQMA end
```

發佈/訂閱範例程式

發佈/訂閱範例程式示範如何使用 IBM MQ 中的發佈和訂閱特性。

有三個 C 語言範例程式說明如何對 IBM MQ 發佈/訂閱介面進行程式設計。 有些 C 範例使用較舊的介面，而有些 Java 範例。 Java 範例使用 com.ibm.mq.jar 中的 IBM MQ 發佈/訂閱介面，以及 com.ibm.mqjms 中的 JMS 發佈/訂閱介面。 本主題未涵蓋 JMS 範例。

C

在 C samples 資料夾中尋找發佈者範例 amqspub。 請以您喜歡的任何主題名稱作為第一個參數來執行它，後面接著選用的佇列管理程式名稱。 例如， amqspub mytopic QM3。 還有一個稱為 amqspubc 的用戶端版本。 如果您選擇執行用戶端版本，請先參閱 第 897 頁的『配置佇列管理程式以接受 Multiplatforms 上的用戶端連線』 以取得詳細資料。

發佈者連接至預設佇列管理程式，並以輸出 `target topic is mytopic` 回應。 從現在開始，您進入此視窗的每一行都會發佈至 `mytopic` 。

在相同目錄中開啟另一個指令視窗，並執行訂閱者程式 `amqssub`，為它提供相同的主題名稱及選用的佇列管理程式名稱。 例如， `amqssub mytopic QM3` 。

訂閱者會以輸出 `Calling MQGET : 30 seconds wait time` 回應。 從現在開始，您輸入發佈者的線路會出現在訂閱者的輸出中。

在另一個指令視窗中啟動另一個訂閱者，並監看這兩個訂閱者接收發佈。

如需參數的完整文件 (包括設定選項) ，請參閱範例原始碼。 下列主題說明訂閱者選項欄位的值: 選項 (MQLONG)。

有另一個訂閱者範例 `amqssbx`，它提供其他訂閱選項作為指令行切換。

鍵入 `amqssbx -d mysub -t mytopic -k` ，以使用訂閱者終止之後所保留的可延續訂閱來呼叫訂閱者。

透過使用發佈者發佈另一個項目來測試訂閱。 等待 30 秒，讓訂閱者終止。 在相同主題下發佈更多項目。 重新啟動訂閱者。 訂閱者未執行時所發佈的最後一個項目會立即重新啟動訂閱者。

C 舊式

還有一組額外的 C 範例示範已排入佇列的指令。 其中部分範例最初隨附於 MQ0C Supportpac。 基於相容性原因，完全支援範例所示範的功能。

我們不建議您使用已排入佇列的指令介面。 它比發佈/訂閱 API 更複雜，而且沒有令人信服的功能理由對複雜的佇列指令進行程式設計。 不過，您可能會發現排入佇列的方法更適合，可能是因為您已使用介面，或因為您的程式設計環境可讓您更容易建置複式訊息並呼叫一般 MQPUT ，而不是建構對 MQSUB 的不同呼叫。

其他範例位於 `samples` 資料夾的 `pubsub` 子目錄中。

第 923 頁的表 163 中列出六種類型的範例。

表 *163:* 舊式發佈/訂閱範例 *C* 程式的種類

種類	程式	註解
RFH1	`amqssr1a.c` `amqspr1a.c`	使用 RFH1 格式訊息建置的簡式發佈/訂閱範例。
RFH2	`amqssr2a.c` `amqspr2a.c`	使用 RFH2 格式訊息建置的簡式發佈/訂閱範例。
MQAI 範例	`amqsppca.c` `amqsspca.c`	使用 PCF 指令及 MQAI 指令介面建置的簡式發佈/訂閱範例。
MA0C 使用 RFH1 的結果服務	`amqsgama.c` `amqsresa.c`	使用 RFH1 標頭建置的結果服務 1. 需要在 `amqsgama.tst` 和 `amqsresa.tst` 中定義的佇列 2. `amqsresa` 必須在 `amqsgama` 之前啟動
MA0C 使用 RFH2 的結果服務	`amqsgr2a.c` `amqsrr2a.c`	使用 RFH2 標頭建置的結果服務 1. 需要在 `amqsgama.tst` 和 `amqsresa.tst` 中定義的佇列 2. `amqsresa` 必須在 `amqsgama` 之前啟動
遞送結束程式發佈/訂閱範例	`amqspsra.c`	示範如何在遞送結束程式中變更發佈/訂閱訊息的佇列或佇列管理程式目的地。

Java 的範例程式

Java 範例 MQPubSubApiSample.java 會在單一程式中結合發佈者和訂閱者。 它的原始檔和已編譯的類別檔位於 wmqjava 範例資料夾中。

如果您選擇以用戶端模式執行，請先參閱 第 897 頁的『配置佇列管理程式以接受 Multiplatforms 上的用戶端連線』 以取得詳細資料。

如果您已配置 Java 環境，請使用 Java 指令從指令行執行範例。 您也可以從已設定 Java 程式設計工作台的 IBM MQ Explorer Eclipse 工作區執行範例。

您可能需要變更部分範例程式的內容，才能執行它。 作法是提供參數給 JVM ，或編輯來源。

第 924 頁的『執行 MQPubSubApiSample Java 範例』 中的指示顯示如何從 Eclipse 工作區執行範例。

執行 *MQPubSubApiSample Java* 範例
如何使用 Eclipse 平台中的 Java 開發工具來執行 MQPubSubApiSample 。

開始之前

開啟 Eclipse 工作台。 建立新的工作區目錄並選取它。 關閉歡迎使用視窗。

在作為用戶端執行之前，請遵循 第 897 頁的『配置佇列管理程式以接受 Multiplatforms 上的用戶端連線』 中的步驟。

關於這項作業

Java 發佈/訂閱範例程式是 IBM MQ MQI client Java 程式。 範例會使用在埠 1414 上接聽的預設佇列管理程式來執行，而不進行修改。 此作業說明此簡式案例，並以一般術語指出如何提供參數及修改範例，以符合不同的 IBM MQ 配置。 範例說明在 Windows 上執行。 在其他平台上，檔案路徑會有所不同。

程序

1. 匯入 Java 範例程式
   a) 在工作台中，按一下 **視窗** > **開啟視景** > **其他** > **Java** ，然後按一下 **確定**。
   b) 切換至 **套件瀏覽器** 視圖。
   c) 在 **套件瀏覽器** 視圖中，用滑鼠右鍵按一下空格。 按一下 **新建** > **Java 專案**。
   d) 在 **Project name** 欄位中，輸入 MQ Java Samples。 按 **下一步**。
   e) 在 **Java Settings** 畫面中，切換至 **程式庫** 標籤。
   f) 按一下 **新增外部 JAR**。
   g) 瀏覽至 *MQ_INSTALLATION_PATH* \java\lib ，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 安裝資料夾，並選取 com.ibm.mq.jar 和 com.ibm.mq.jmqi.jar
   h) 按一下 **開啟** > **完成**。
   i) 在「 **套件瀏覽器** 」視圖中，用滑鼠右鍵按一下 src 。
   j) 選取 **匯入 ...** > **一般** > **檔案系統** > **下一步** > **瀏覽**... 並瀏覽至路徑 *MQ_INSTALLATION_PATH* \tools\wmqjava\samples ，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 安裝目錄。
   k) 在「 **匯入** 」畫面 第 925 頁的圖 130 上，按一下 samples (不要選取勾選框)。
   l) 選取 MQPubSubApiSample.java。 **Into folder** 欄位應該包含 MQ Java Samples/src。 按一下**完成**。

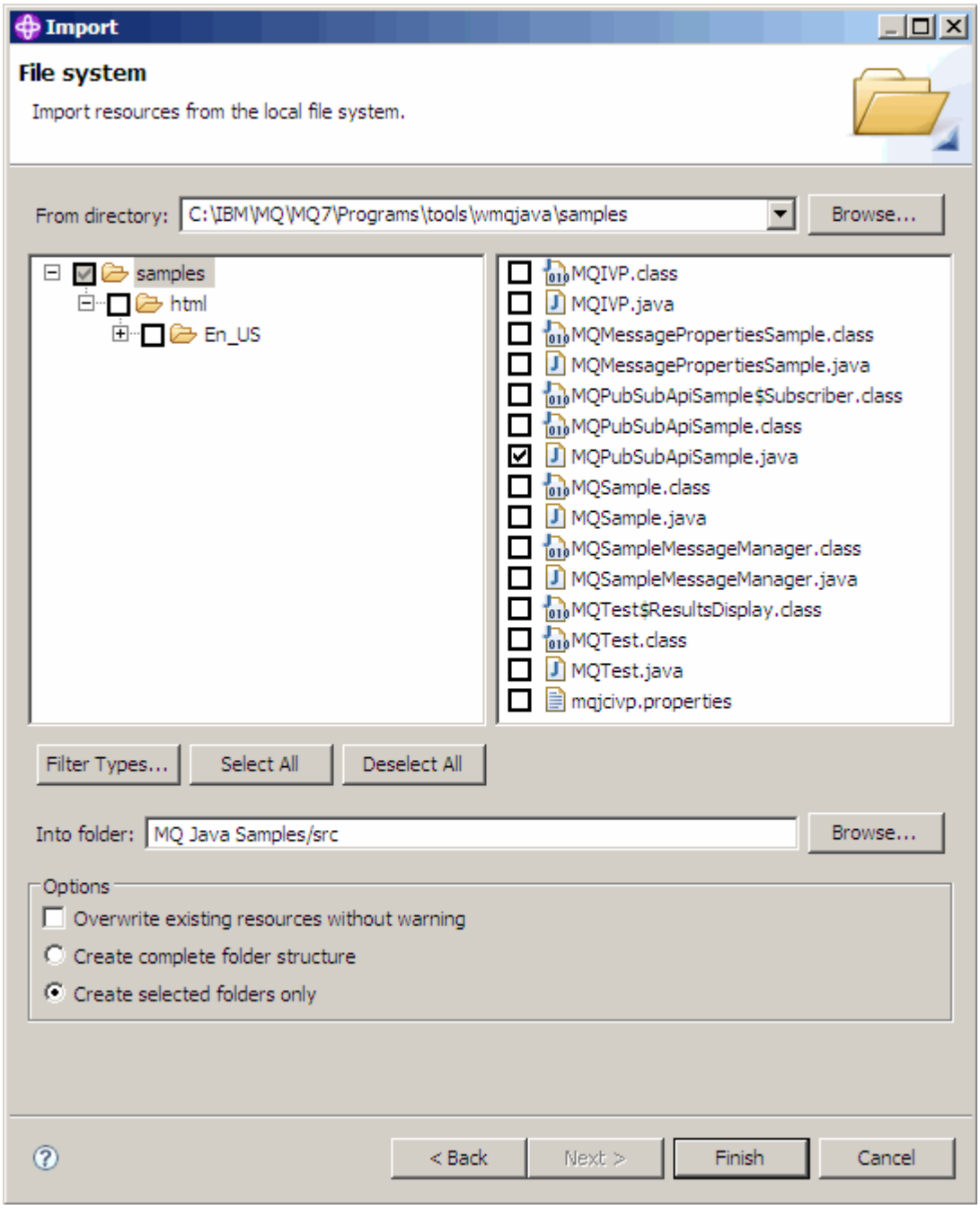


圖 *130:* 檔案系統匯入

2. 執行發佈/訂閱範例程式。
   視您是否需要變更預設參數而定，有兩種方式可執行程式。
   - 第一個選項會執行程式，而不進行任何變更:
     - 在工作區主功能表中，展開 `src` 資料夾。 用滑鼠右鍵按一下 **MQPubSubApiSample.java 執行身分 > 1. Java 應用程式**
   - 第二個選項會針對您的環境，使用參數或修改的原始碼來執行程式:
     - 開啟 `MQPubSubApiSample.java`，並研究 `MQPubSubApiSample` 建構子。
     - 修改程式的屬性。
       這些屬性可使用 -D JVM 參數來修改，或透過編輯原始碼來提供 System property 的預設值。

- topicObject
- QueueManagerName
- subscriberCount

只有在建構子中編輯原始碼，才能變更這些屬性。

- hostname
- port
- channel

若要設定 System pRopertes ， 請在存取元中撰寫預設值的程式碼， 例如:

```
queueManagerName = System.getProperty("com.ibm.mq.pubSubSample.queueManagerName",
"QM3");
```

或者使用 -D 選項提供參數給 JVM ， 如下列步驟所示:

a. 複製您要設定之 System.Property 的完整名稱， 例如: `com.ibm.mq.pubSubSample.queueManagerName`。
b. 在工作區中， 用滑鼠右鍵按一下 **執行 > 開啟執行對話框**。 按兩下 Java **建立、管理及執行應用程式** 中的應用程式， 然後按一下 **(x) = 引數** 標籤。
c. 在 **VM 引數:** 窗格中， 輸入 -D ， 並貼上 System.property 名稱 `com.ibm.mq.pubSubSample.queueManagerName`， 後面接著 =QM3。 按一下 **套用 > 執行**。
d. 將進一步引數新增為逗點區隔清單， 或新增為窗格中沒有逗點分隔字元的其他行。

例如： `-Dcom.ibm.mq.pubSubSample.queueManagerName=QM3, -Dcom.ibm.mq.pubSubSample.subscriberCount=6`。

發佈結束程式範例程式

AMQSPSE0 是結束程式的範例 C 程式， 用來在將發佈遞送至訂閱者之前截取它。 例如， 結束程式可以變更訊息標頭、有效負載或目的地， 或阻止將訊息發佈至訂閱者。

若要執行範例， 請執行下列作業:

1. 配置佇列管理程式:
   - Linux AIX 在 AIX and Linux 系統上， 將如下的段落新增至 `qm.ini` 檔案:

     ```
     PublishSubscribe:
     PublishExitPath=Module
     PublishExitFunction=EntryPoint
     ```

     其中模組是 *MQ_INSTALLATION_PATH*`/samp/bin/amqspse`。 *MQ_INSTALLATION_PATH* 代表 IBM MQ 安裝所在的高階目錄。
   - Windows 在 Windows 上， 設定登錄中的對等屬性。
2. 請確定 IBM MQ 可以存取模組。
3. 重新啟動佇列管理程式以挑選配置。
4. 在要追蹤的應用程式程序中， 說明應寫入追蹤檔的位置。 例如:
   - Linux AIX 在 AIX and Linux 系統上， 請確定 `/var/mqm/trace` 目錄存在， 並匯出 **MQPSE_TRACE_LOGFILE** 環境變數:

     ```
     export MQPSE_TRACE_LOGFILE=/var/mqm/trace/PubTrace
     ```

- Windows 在 Windows 上，請確定 C:\temp 目錄存在，並設定 **MQPSE_TRACE_LOGFILE** 環境變數:

```
set MQPSE_TRACE_LOGFILE=C:\temp\PubTrace
```

放置範例程式

「放置範例」程式會使用 MQPUT 呼叫將訊息放置在佇列上。

如需這些程式的名稱，請參閱 第 889 頁的『Multiplatforms 上範例程式中示範的特性』 。

Put 範例程式的設計

程式使用 MQOPEN 呼叫搭配 MQOO_OUTPUT 選項，來開啟目標佇列以放置訊息。

如果無法開啟佇列，程式會輸出一則錯誤訊息，其中包含 MQOPEN 呼叫所傳回的原因碼。 為了讓程式保持簡單，在此及後續的 MQI 呼叫上，程式會對許多選項使用預設值。

對於每一行輸入，程式會將文字讀取到緩衝區中，並使用 MQPUT 呼叫來建立包含該行文字的資料包訊息。 程式會繼續執行，直到到達輸入結尾或 MQPUT 呼叫失敗為止。 如果程式到達輸入結尾，則會使用 MQCLOSE 呼叫來關閉佇列。

執行 *Put* 範例程式

執行 amqsput 和 amqsputc 範例

ALW

amqqsput 範例是使用本端連結來放置訊息的程式， **amqsputc** 範例是使用用戶端連結來放置訊息的程式。 這些程式各採用下列位置參數:

1. 目標佇列的名稱 (必要)
2. 佇列管理程式的名稱 (選用)

   如果未指定佇列管理程式，則「 **amqsput** 」會連接至預設佇列管理程式，而「 **amqsputc** 」會連接至 *MQSERVER* 環境變數或用戶端通道定義檔所識別的佇列管理程式。
3. 開啟選項 (選用)

   如果未指定開啟選項，則範例會使用值 8208 ，這是下列兩個選項的組合:

   - MQOO_OUTPUT
   - MQOO_FAIL_IF_QUIESCING
4. 關閉選項 (選用)

   如果未指定 close 選項，則範例會使用值 0 (即 MQCO_NONE)。
5. 目標佇列管理程式的名稱 (選用)

   如果未指定目標佇列管理程式，則 MQOD 中的 ObjectQMgrName 欄位會保留空白。
6. 動態佇列的名稱 (選用)

   如果未指定動態佇列名稱，則 MQOD 中的 DynamicQName 欄位會保留空白。

請使用下列環境變數來提供用來向佇列管理程式進行鑑別的認證:

MQSAMP_USER_ID
如果您想要使用使用者 ID 和密碼向佇列管理程式進行鑑別，請設為用於連線鑑別的使用者 ID。 程式會提示輸入密碼以隨附使用者 ID。

Linux AIX V 9.3.4 **MQSAMP_TOKEN**
如果您要提供鑑別記號以向佇列管理程式進行鑑別，請設為非空白值。 程式會提示輸入鑑別記號。 鑑別記號只能由使用用戶端連結的 **amqsputc** 範例使用。

若要執行這些程式，請輸入下列其中一個指令:

- amqsput *myqueue qmanagername*
- amqsputc *myqueue qmanagername*

其中 *myqueue* 是要放置訊息的佇列名稱， *qmanagername* 是擁有 *myqueue* 的佇列管理程式。

執行 amq0put 範例

ALW

COBOL 版本沒有任何參數。 它會連接至預設佇列管理程式，當您執行它時，系統會提示您:

```
Please enter the name of the target queue
```

它接受來自 StdIn 的輸入，並將每一行輸入新增至目標佇列。 空白行表示沒有其他資料。

執行 AMQSPUT4 C 範例 (IBM i)

IBM i

C 程式 AMQSPUT4(僅適用於 IBM i 平台) 會透過從原始檔成員中讀取資料來建立訊息。

當您啟動程式時，必須將檔案名稱指定為參數。 檔案的結構必須是:

```
queue name
text of message 1
text of message 2
:
text of message n
blank line
```

程式庫 QMQMSAMP 檔案 AMQSDATA 成員 PUT 中提供放置範例的輸入範例。

註: 請記住佇列名稱區分大小寫。 範例檔案建立程式 AMQSAMP4 所建立的所有佇列都具有以大寫字元建立的名稱。

C 程式會將訊息放置在檔案第一行所指名的佇列上; 您可以使用提供的佇列 SYSTEM.SAMPLE.LOCAL。 程式會將檔案下列每一行的文字放入個別的資料包訊息中，並在讀取檔案結尾的空白行時停止。

使用範例資料檔時，指令為:

```
CALL PGM(QMQM/AMQSPUT4) PARM('QMQMSAMP/AMQSDATA(PUT)')
```

執行 AMQ0PUT4 COBOL 範例 (IBM i)

IBM i

COBOL 程式 AMQ0PUT4(僅適用於 IBM i 平台) 透過接受鍵盤中的資料來建立訊息。

若要啟動程式，請呼叫程式，並提供目標佇列的名稱作為程式參數。 程式接受從鍵盤輸入至緩衝區的輸入，並為每一行文字建立資料封包訊息。 當您在鍵盤上輸入空白行時，程式會停止。

參照訊息範例程式

「參照訊息」範例容許將大型物件從一個節點傳送至另一個節點 (通常位於不同系統上) ， 而不需要將物件儲存在來源節點或目的地節點的 IBM MQ 佇列上。

提供一組範例程式，以示範如何將「參照訊息」放入佇列、由訊息結束程式接收，以及從佇列中取得。 範例程式會使用「參照訊息」來移動檔案。 如果您要移動其他物件 (例如資料庫) ，或要執行安全檢查，請根據範例 amqsxrm 定義您自己的結束程式。

要使用的「參照訊息結束程式」範例程式版本取決於通道執行所在的平台:

- 在所有平台上，請在傳送端使用 amqsxrma。
- 如果接收端在 IBM i 以外的任何平台下執行，請在接收端使用 amqsxrma。

- IBM i 如果接收端在 IBM i 下執行，請使用 amqsxrm4。

.

IBM i *IBM i* 使用者注意事項

若要使用範例訊息結束程式接收「參照訊息」，請在 IFS 或任何子目錄的根檔案系統中指定檔案，以便可以建立串流檔。

IBM i 上的範例訊息結束程式會建立檔案，將資料轉換成 EBCDIC，並將字碼頁設為您的系統字碼頁。 然後，您可以將此檔案複製到 QSYS.LIB 檔案系統使用 CPYFRMSTMF 指令。 例如：

```
CPYFRMSTMF FROMSTMF('JANEP/TEST.TXT')
TOMBR('qsys.lib.janep.lib/test.fie/test.mbr') MBROPT(*REPLACE)
CVTDTA(*NONE)
```

CPYFRMSTMF 指令不會建立檔案。 您必須先建立它，然後再執行此指令。

如果您從 QSYS.LIB，不需要對範例進行任何變更。 對於任何其他檔案系統，請確保 MQRMH 結構中 CodedCharSetId 欄位中指定的 CCSID 符合您正在傳送的大量資料。

使用整合檔案系統時，請使用 SYSIFCOPT (*IFSIO) 選項集來建立程式模組。 如果您要移動資料庫或固定長度記錄檔，請根據所提供的範例 AMQSXRM4 來定義您自己的結束程式。

傳送資料庫檔案的建議方法是使用 CPYTOSTMF 指令將它轉換為 IFS 結構，然後傳送附加 IFS 檔案的「參照訊息」。 如果您選擇透過從 IFS 內參照資料庫檔案來傳送資料庫檔案，但不將其轉換為 IFS 結構，則必須指定成員名稱。 如果您選擇此方法，則不保證資料完整性。

Multi 執行參照訊息範例

使用此範例來瞭解如何在 AIX, Linux, and Windows 上執行「參照訊息」範例應用程式 AMQSPRM，或在 IBM i 上執行 AMQSPRMA。 此範例顯示如何將「參照訊息」放入佇列、由訊息結束程式接收，以及從佇列中取得。

「參照訊息」範例如下所示執行:

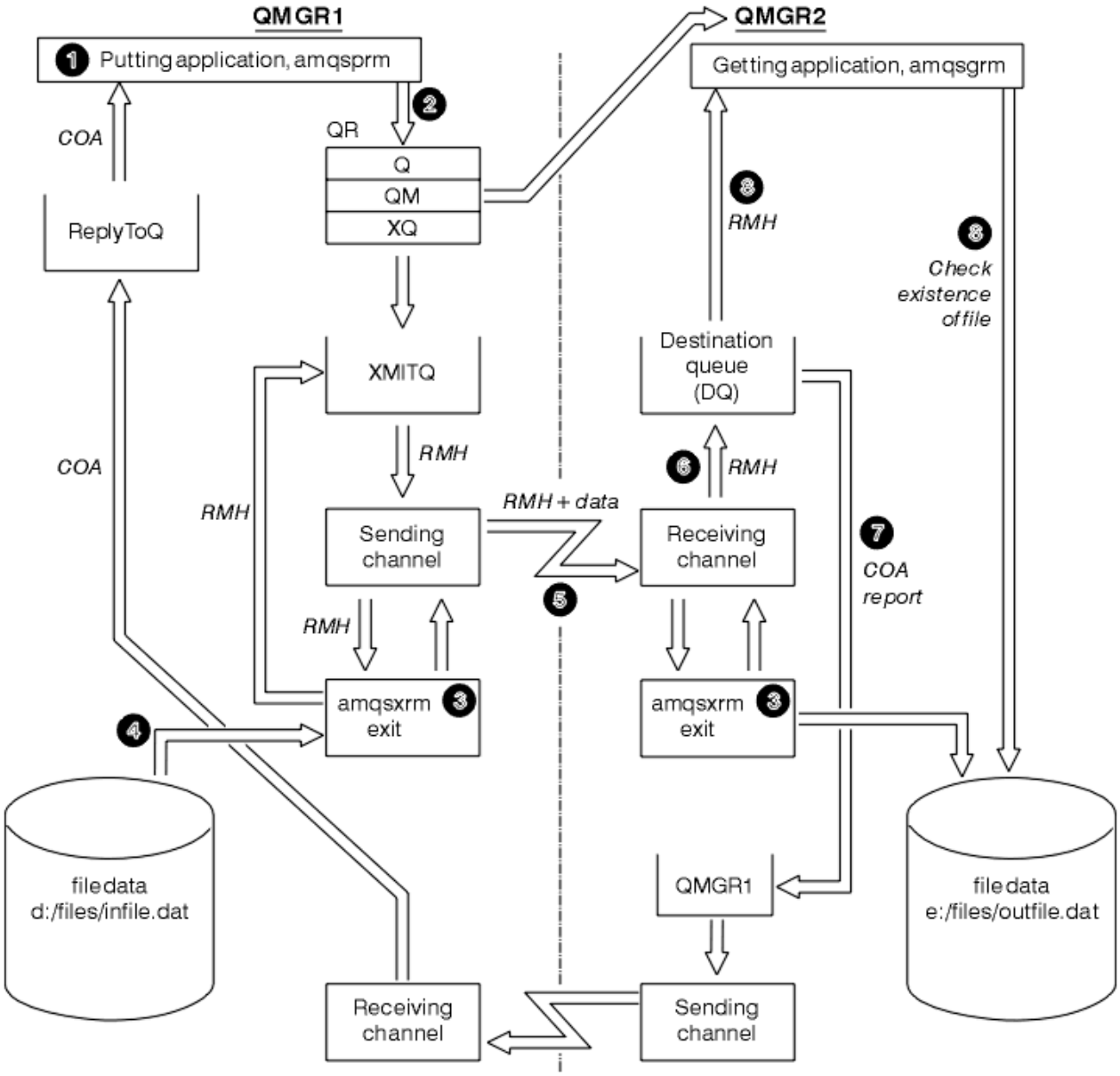


圖 *131:* 執行參照訊息範例

1. 設定環境以啟動接聽器、通道及觸發監視器，並定義通道及佇列。

   為了說明如何設定「參照訊息」，這個範例將傳送端機器稱為 MACHINE1，其佇列管理程式稱為 QMGR1，而接收端機器稱為 MACHINE2，其佇列管理程式稱為 QMGR2。

   註: 下列定義容許建置「參照訊息」，以將物件類型為 FLATFILE 的檔案從佇列管理程式 QMGR1 傳送至 QMGR2，並重建在對 AMQSPRM (或 IBM i 上的 AMQSPRMA)的呼叫中定義的檔案。「參照訊息」(包括檔案資料) 會使用通道 CHL1 及傳輸佇列 XMITQ 來傳送，並置於佇列 DQ 上。異常狀況及 COA 報告會使用通道 REPORT 及傳輸佇列 QMGR1 傳回 QMGR1。

   使用起始佇列 INITQ 並處理 PROC 來觸發接收「參照訊息」(AMQSGRM 或 IBM i 上的 AMQSGRMA) 的應用程式。請確定 CONNAME 欄位設定正確，且 MSGEXIT 欄位會反映您的目錄結構，視機型及安裝 IBM MQ 產品的位置而定。

   IBM i MQSC 定義已使用 AIX 樣式來定義結束程式，因此如果您在 IBM i 上使用 MQSC，則需要相應地修改這些樣式。請務必注意，訊息資料 FLATFILE 區分大小寫，除非是大寫，否則範例將無法運作。

   在機器 MACHINE1 上，佇列管理程式 QMGR1

MQSC 語法

```
define chl(chl1) chltype(sdr) trptype(tcp) conname('machine2') xmitq(xmitq)
msgdata(FLATFILE) msgexit('/usr/lpp/mqm/samp/bin/amqsxrm(MsgExit)
')

define ql(xmitq) usage(xmitq)

define chl(report) chltype(rcvr) trptype(tcp) replace

define qr(qr) rname(dq) rqmname(qmgr2) xmitq(xmitq) replace
```

IBM i 指令語法

註: 如果您未指定佇列管理程式名稱，系統會使用預設佇列管理程式。

```
CRTMQMCHL CHLNAME(CHL1) CHLTYPE(*SDR) MQMNAME(QMGR1) +
            REPLACE(*YES) TRPTYPE(*TCP) +
            CONNAME('MACHINE2(60501)') TMQNAME(XMITQ) +
            MSGEXIT(QMQM/AMQSXRM4) MSGUSRDATA(FLATFILE)

CRTMQMQ   QNAME(XMITQ) QTYPE(*LCL) MQMNAME(QMGR1) +
            REPLACE(*YES) USAGE(*TMQ)

CRTMQMCHL CHLNAME(REPORT) CHLTYPE(*RCVR) +
            MQMNAME(QMGR1) REPLACE(*YES) TRPTYPE(*TCP)

CRTMQMQ   QNAME(QR) QTYPE(*RMT) MQMNAME(QMGR1) +
            REPLACE(*YES) RMTQNAME(DQ) +
            RMTMQMNAME(QMGR2) TMQNAME(XMITQ)
```

在機器 MACHINE2 上，佇列管理程式 QMGR2

MQSC 語法

```
define chl(chl1) chltype(rcvr) trptype(tcp)
msgexit('/usr/lpp/mqm/samp/bin/amqsxrm(MsgExit)')
        msgdata(flatfile)

define chl(report) chltype(sdr) trptype(tcp) conname('MACHINE1')
        xmitq(qmgr1)

define ql(initq)

define ql(qmgr1) usage(xmitq)

define pro(proc) applicid('/usr/lpp/mqm/samp/bin/amqsgrm')

define ql(dq) initq(initq) process(proc) trigger trigtype(first)
```

IBM i 指令語法

註: 在 IBM i 上，如果您未指定佇列管理程式名稱，系統會使用預設佇列管理程式。

```
CRTMQMCHL CHLNAME(CHL1) CHLTYPE(*RCVR) MQMNAME(QMGR2) +
REPLACE(*YES) TRPTYPE(*TCP) +
MSGEXIT(QMQM/AMQSXRM4) MSGUSRDATA(FLATFILE)

CRTMQMCHL CHLNAME(REPORT) CHLTYPE(*SDR) MQMNAME(QMGR2) +
REPLACE(*YES) TRPTYPE(*TCP) +
CONNAME('MACHINE1(60500)') TMQNAME(QMGR1)

CRTMQMQ  QNAME(INITQ) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) USAGE(*NORMAL)

CRTMQMQ  QNAME(QMGR1) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) USAGE(*TMQ)

CRTMQMPRC PRCNAME(PROC) MQMNAME(QMGR2) REPLACE(*YES) +
APPID('QMQM/AMQSGRM4')

CRTMQMQ  QNAME(DQ) QTYPE(*LCL) MQMNAME(QMGR2) +
```

```
REPLACE(*YES) PRCNAME(PROC) TRGENBL(*YES) +
INITQNAME(INITQ)
```

2. 建立 IBM MQ 物件之後:
   a. 如果適用於平台，請啟動傳送及接收佇列管理程式的接聽器
   b. 啟動通道 CHL1 和 REPORT
   c. 在接收端佇列管理程式上，啟動起始佇列 INITQ 的觸發監視器
3. 使用下列參數，從指令行呼叫放置參照訊息範例程式 AMQSPRM (AMQSPRMA on IBM i) :

   -m
   本端佇列管理程式的名稱; 這會預設為預設佇列管理程式

   -i
   原始檔的名稱和位置

   -o
   目的地檔案的名稱及位置

   -q
   佇列的名稱

   -g
   定義在 -q 參數中的佇列所在的佇列管理程式名稱。 這會預設為 -m 參數中指定的佇列管理程式。

   -t
   物件類型

   -w
   等待間隔，亦即來自接收端佇列管理程式的異常狀況及 COA 報告的等待時間

   例如，若要搭配使用範例與先前定義的物件，您將使用下列參數:

```
-mQMGR1 -iInput File -oOutput File -qQR -tFLATFILE -w120
```

   增加等待時間容許在程式放置訊息逾時之前，透過網路傳送大型檔案的時間。

```
amqsprm -q QR -m QMGR1 -i d:\x\file.in -o d:\y\file.out -t FLATFILE
```

   IBM i 使用者: IBM i 在 IBM i 上，完成下列步驟:

   a. 使用下列指令:

```
CALL    PGM(QMQM/AMQSPRM4) PARM('-mQMGR1' +
'-i/refmsgs/rmsg1' +
'-o/refmsgs/rmsgx' '-qQR' +
'-gQMGR1' '-tFLATFILE' '-w15')
```

      這會假設原始檔案 `rmsg1` 位於 IFS 目錄 `/refmsgs` 中，且您希望目的地檔案在目標系統上的 `rmsgx` IFS 目錄中 `/refmsgs` 。
   b. 使用 CRTDIR 指令而非使用根目錄來建立您自己的目錄。
   c. 當您呼叫放置資料的程式時，請記住輸出檔名稱需要反映 IFS 命名慣例; 例如 /TEST/FILENAME 會在 TEST 目錄中建立一個稱為 FILENAME 的檔案。

   註:

   IBM i 在 IBM i 上，您可以在指定參數時使用正斜線 (/) 或橫線 (-)。 例如:

```
amqsprm /i d:\files\infile.dat /o e:\files\outfile.dat /q QR
/m QMGR1 /w 30 /t FLATFILE
```

Linux AIX 對於 AIX and Linux 平台，您必須使用兩條反斜線 (\\) 而非一條反斜線來表示目的地檔案目錄。 因此， **amqsprm** 指令看起來如下:

```
amqsprm -i /files/infile.dat -o e:\\files\\outfile.dat -q QR
-m QMGR1 -w 30 -t FLATFILE
```

執行「放置參照訊息」程式會執行下列動作:

- 「參照訊息」會放入佇列管理程式 QMGR1 上的佇列 QR 中。
- 來源檔和路徑是 `d:\files\infile.dat`，且存在於發出範例指令的系統上。
- 如果佇列 QR 是遠端佇列，則「參照訊息」會傳送至另一個系統上的另一個佇列管理程式，其中建立的檔案具有名稱及路徑 `e:\files\outfile.dat`。 此檔案的內容與原始檔相同。
- amqsprm 會等待來自目的地佇列管理程式的 COA 報告 30 秒。
- 物件類型為 `flatfile`，因此用來從佇列 QR 移動訊息的通道必須在 *MsgData* 欄位中指定此類型。

4. 當您定義通道時，請選取傳送端和接收端的訊息結束程式，以作為 amqsxrm。

Windows 這在 Windows 上定義如下:

```
msgexit(' pathname\amqsxrm.dll(MsgExit)')
```

Linux AIX 這在 AIX and Linux 上定義如下:

```
msgexit(' pathname/amqsxrm(MsgExit)')
```

如果您指定路徑名稱，請指定完整名稱。 如果您省略路徑名稱，則會假設程式位於 `qm.ini` 檔案中指定的路徑 (或在 IBM MQ for Windows 上，則為登錄中指定的路徑)。

5. 通道結束程式會讀取「參照訊息」標頭，並尋找它所參照的檔案。
6. 然後，通道結束程式可以將檔案分段，然後再將它與標頭一起傳送至通道。

Linux AIX 在 AIX and Linux 上，將目標目錄的群組擁有者變更為 'mqm'，以便範例訊息結束程式可以在該目錄中建立檔案。 此外，請變更目標目錄的許可權，以容許 mqm 群組成員寫入其中。 檔案資料不會儲存在 IBM MQ 佇列上。

7. 當接收訊息結束程式處理檔案的最後一個區段時，「參照訊息」會放入 amqsprm 指定的目的地佇列中。如果觸發此佇列 (亦即，定義指定 **Trigger**、 **InitQ** 及 **Process** 佇列屬性)，則會觸發目的地佇列的 PROC 參數所指定的程式。 要觸發的程式必須定義在 **Process** 屬性的 `ApplId` 欄位中。
8. 當「參照訊息」到達目的地佇列 (DQ) 時， COA 報告會傳回放置應用程式 (amqsprm)。
9. 「取得參照訊息」範例 amqsgrm 會從輸入觸發訊息中指定的佇列取得訊息，並檢查檔案是否存在。

「放置參照訊息」範例的設計 *(amqsprma.c、 AMQSPRM4)*

本主題提供「放置參照訊息」範例的詳細說明。

此範例會建立參照檔案的「參照訊息」，並將它放置在指定的佇列上:

1. 範例使用 MQCONN 連接至本端佇列管理程式。
2. 然後，它會開啟 (MQOPEN) 用來接收報告訊息的模型佇列。
3. 範例會建置「參照訊息」，其中包含移動檔案所需的值，例如來源及目的地檔案名稱及物件類型。 例如， IBM MQ 隨附的範例會建置「參照訊息」，以將檔案 `d:\x\file.in` 從 `QMGR1` 傳送至 `QMGR2`，並使用下列參數將檔案重建為 `d:\y\file.out`:

```
amqsprm -q QR -m QMGR1 -i d:\x\file.in -o d:\y\file.out -t FLATFILE
```

其中 `QR` 是參照 `QMGR2` 上目標佇列的遠端佇列定義。

註: 若為 AIX and Linux 平台，請使用兩條反斜線 (\\) 而非一條反斜線來表示目的地檔案目錄。 因此，**amqsprm** 指令看起來如下:

```
amqsprm -q QR -m QMGR1 -i /x/file.in -o d:\\y\\file.out -t FLATFILE
```

4. 「參照訊息」會放入 (不含任何檔案資料) /q 參數指定的佇列中。 如果這是遠端佇列，則會將訊息放入對應的傳輸佇列。
5. 針對 COA 報告，樣本會在 /w 參數 (預設值為 15 秒) 中指定的持續時間內等待，這些報告與異常狀況報告一起傳回本端佇列管理程式上建立的動態佇列 (QMGR1)。

「參照訊息結束程式」範例的設計 *(amqsxrma.c、 AMQSXRM4)*
此範例可辨識「參照訊息」，其物件類型符合通道定義的訊息結束程式使用者資料欄位中的物件類型。

對於這些訊息，會發生下列情況:

- 在傳送端或伺服器通道上，指定的資料長度會從指定檔案的指定偏移複製到參照訊息之後代理程式緩衝區中剩餘的空間。 如果未到達檔案結尾，則在更新 *DataLogicalOffset* 欄位之後，會將「參照訊息」放回傳輸佇列。
- 在要求端或接收端通道上，如果 *DataLogicalOffset* 欄位為零，且指定的檔案不存在，則會建立該檔案。 「參照訊息」之後的資料會新增至指定檔案的結尾。 如果「參照訊息」不是所指定檔案的最後一個訊息，則會捨棄它。 否則，它會回到通道結束程式 (不含附加的資料) ，以放置在目標佇列上。

對於傳送端及伺服器通道，如果輸入「參照訊息」中的 *DataLogicalLength* 欄位為零，則檔案的其餘部分 (從 *DataLogicalOffset* 到檔案結尾) 將沿著通道傳送。 如果它不是零，則只會傳送指定的長度。

如果發生錯誤 (例如，如果範例無法開啟檔案) ，則為 MQCXP。 *ExitResponse* 設定為 MQXCC_SUPPRESS_FUNCTION ，以便將正在處理的訊息放入無法傳送郵件的佇列，而不是繼續傳送至目的地佇列。 MQCXP 會傳回回饋碼。 *Feedback* ，並傳回給將訊息放置在報告訊息之訊息描述子的 *Feedback* 欄位中的應用程式。 這是因為放置應用程式透過在 MQMD 的 *Report* 欄位中設定 MQRO_EXCEPTION 來要求異常狀況報告。

如果「參照訊息」的編碼或 *CodedCharacterSetId* (CCSID) 與佇列管理程式的編碼或 CCSID 不同，則「參照訊息」會轉換為本端編碼及 CCSID。 在我們的範例 amqsprm 中，物件的格式為 MQFMT_STRING ，因此在將資料寫入檔案之前， amqsxrm 會將物件資料轉換成接收端的本端 CCSID。

如果檔案包含多位元組字元 (例如 DBCS 或 Unicode) ，請不要指定要以 MQFMT_STRING 傳送的檔案格式。 這是因為當檔案在傳送端分段時，可以分割多位元組字元。 若要傳送及轉換此類檔案，請將格式指定為 MQFMT_STRING 以外的格式，以便「參照訊息」結束程式不會轉換它，並在傳送完成時於接收端轉換檔案。

編譯參照訊息結束程式範例
若要編譯「參照訊息結束程式」範例，請使用適用於安裝 IBM MQ 之平台的指令。

MQ_INSTALLATION_PATH 代表 IBM MQ 安裝所在的高階目錄。

若要編譯 amqsxrma ，請使用下列指令:

在 AIX 上:

AIX

```
xlc_r -q64 -e MsgExit -bE:amqsxrm.exp -bM:SRE -o amqsxrm_64_r
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -lmqm_r amqsqrma.c
```

在 IBM i 上:

IBM i

```
CRTCMOD MODULE(MYLIB/AMQSXRMA) SRCFILE(QMQMSAMP/QCSRC)
TERASPACE(*YES *TSIFC)
```

註:

1. 如果要建立模組以便它使用 IFS 檔案系統，請新增選項 SYSIFCOPT(*IFSIO)
2. 若要建立程式以與非執行緒通道搭配使用，請使用下列指令: CRTPGM PGM(MYLIB/AMQSXRMA) BNDSRVPGM(QMQM/LIBMQM)
3. 若要建立與執行緒通道搭配使用的程式，請使用下列指令: CRTPGM PGM(MYLIB/AMQSXRMA) BNDSRVPGM(QMQM/LIBMQM_R)

在 Linux 上：

Linux

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsxrma amqsqrma.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-lmqm_r
```

在 Windows 上：

Windows IBM MQ 現在為 mqm 程式庫提供用戶端套件及伺服器套件，因此下列範例使用 mqm.lib 而非 mqmvx.lib:

```
cl amqsqrma.c /link /out:amqsxrm.dll /dll mqm.lib mqm.lib /def:amqsxrm.def
```

相關概念

第 808 頁的『寫入通道結束程式』
您可以使用下列資訊來協助您撰寫通道結束程式。

「取得參照訊息」範例的設計 *(amqsgrma.c、 AMQSGRM4)*
本主題說明「取得參照訊息」範例的設計。

程式邏輯如下:

1. 即會觸發範例，並從輸入觸發訊息中擷取佇列及佇列管理程式名稱。
2. 然後，它會使用 MQCONN 連接至指定的佇列管理程式，並使用 MQOPEN 開啟指定的佇列。
3. 此範例會發出 MQGET，並在迴圈中以 15 秒的等待間隔從佇列中取得訊息。
4. 如果訊息是「參照訊息」，範例會檢查已傳送的檔案是否存在。
5. 然後，它會關閉佇列並與佇列管理程式中斷連線。

要求範例程式

「要求程式範例」示範主從式處理。 範例是將要求訊息放置在伺服器程式所處理之目標伺服器佇列上的用戶端。 它們會等待伺服器程式將回覆訊息放置在回覆目的地佇列上。

「要求範例」會使用 MQPUT 呼叫，將一系列要求訊息放置在目標伺服器佇列上。 這些訊息指定本端佇列 SYSTEM.SAMPLE.REPLY 作為回覆目的地佇列，它可以是本端或遠端佇列。 程式會等待回覆訊息，然後顯示它們。 只有在伺服器應用程式正在處理目標伺服器佇列時，或是為了該目的而觸發應用程式時，才會傳送回覆 (「查詢」、「設定」及「回應」範例程式設計為要觸發)。 C 範例會等待 1 分鐘 (COBOL 範例會等待 5 分鐘)，讓第一個回覆到達 (以容許觸發伺服器應用程式的時間)，以及 15 秒後續回覆，但這兩個範例可以在沒有任何回覆的情況下結束。 如需要求範例程式的名稱，請參閱 第 889 頁的『Multiplatforms 上範例程式中示範的特性』。

執行要求程式範例

執行 amqsreq0.c、amqsreq 及 amqsreqc 範例

程式的 C 版本採用三個參數:

1. 目標伺服器佇列的名稱 (必要)
2. 佇列管理程式的名稱 (選用)

3. 回覆佇列 (選用)

例如，輸入下列其中一項:

- `amqsreq myqueue qmanagername replyqueue`
- `amqsreqc myqueue qmanagername`
- `amq0req0 myqueue`

其中 `myqueue` 是目標伺服器佇列的名稱， `qmanagername` 是擁有 `myqueue` 的佇列管理程式名稱，`replyqueue` 是回覆佇列的名稱。

如果您省略佇列管理程式的名稱，則會假設預設佇列管理程式擁有該佇列。 如果您省略回覆佇列的名稱，則會提供預設回覆佇列。

執行 amq0req0.cbl 範例

COBOL 版本沒有任何參數。 它會連接至預設佇列管理程式，當您執行它時，系統會提示您:

```
Please enter the name of the target server queue
```

程式會從 StdIn 取得其輸入，並將每一行新增至目標伺服器佇列，以每一行文字作為要求訊息的內容。 當讀取空行時，程式結束。

執行 AMQSREQ4 範例

C 程式會從 stdin (鍵盤) 取得含有空白時間終止輸入的資料，以建立訊息。 程式最多接受三個參數: 目標佇列的名稱 (必要)、佇列管理程式名稱 (選用) 及回覆目的地佇列名稱 (選用)。 如果未指定佇列管理程式名稱，則會使用預設佇列管理程式。 如果未指定回覆目的地佇列，則為 SYSTEM.SAMPLE.REPLY 佇列。

以下是如何呼叫 C 範例程式 (指定回覆目的地佇列，但讓佇列管理程式預設) 的範例:

```
CALL PGM(QMQM/AMQSREQ4) PARM('SYSTEM.SAMPLE.LOCAL' '' 'SYSTEM.SAMPLE.REPLY')
```

註: 請記住佇列名稱區分大小寫。 範例檔案建立程式 AMQSAMP4 所建立的所有佇列都具有以大寫字元建立的名稱。

執行 AMQ0REQ4 範例

COBOL 程式會從鍵盤接受資料來建立訊息。 若要啟動程式，請呼叫程式並指定目標佇列的名稱作為參數。 程式接受從鍵盤到緩衝區的輸入，並為每一行文字建立要求訊息。 當您在鍵盤上輸入空白行時，程式會停止。

使用觸發來執行要求範例

如果範例與觸發及其中一個「查詢」、「設定」或「回應」範例程式一起使用，則輸入行必須是您要被觸發程式存取之佇列的佇列名稱。

ALW 在 *AIX, Linux, and Windows* 上使用觸發來執行要求範例

在 AIX, Linux, and Windows 上，在一個階段作業中啟動觸發監視器程式 RUNMQTRM ，然後在另一個階段作業中啟動 amqsreq 程式。

如果要使用觸發來執行範例，請執行下列動作:

1. 在一個階段作業 (起始佇列 SYSTEM.SAMPLE.TRIGGER 可供您使用)。
2. 在另一個階段作業中啟動 amqsreq 程式。
3. 請確定您已定義目標伺服器佇列。

   可供您用來作為要求範例放置訊息之目標伺服器佇列的範例佇列如下:

   - SYSTEM.SAMPLE.INQ -適用於 Inquire sample 程式

- SYSTEM.SAMPLE.SET -適用於 Set 範例程式
- SYSTEM.SAMPLE.ECHO -適用於「回應」範例程式

這些佇列具有 FIRST 觸發類型，因此如果在執行「要求」範例之前佇列上已有訊息，則您傳送的訊息不會觸發伺服器應用程式。

4. 請確定您已定義佇列供「查詢」、「設定」或「回應」範例程式使用。

這表示當要求範例傳送訊息時，觸發監視器已備妥。

註: 使用 RUNMQSC 及 amqscos0.tst 檔案建立的範例程序定義會觸發 C 範例。 變更 amqscos0.tst 中的程序定義，並搭配使用 RUNMQSC 與此更新檔案，以使用 COBOL 版本。

第 937 頁的圖 132 示範如何一起使用「要求」和「查詢」範例。

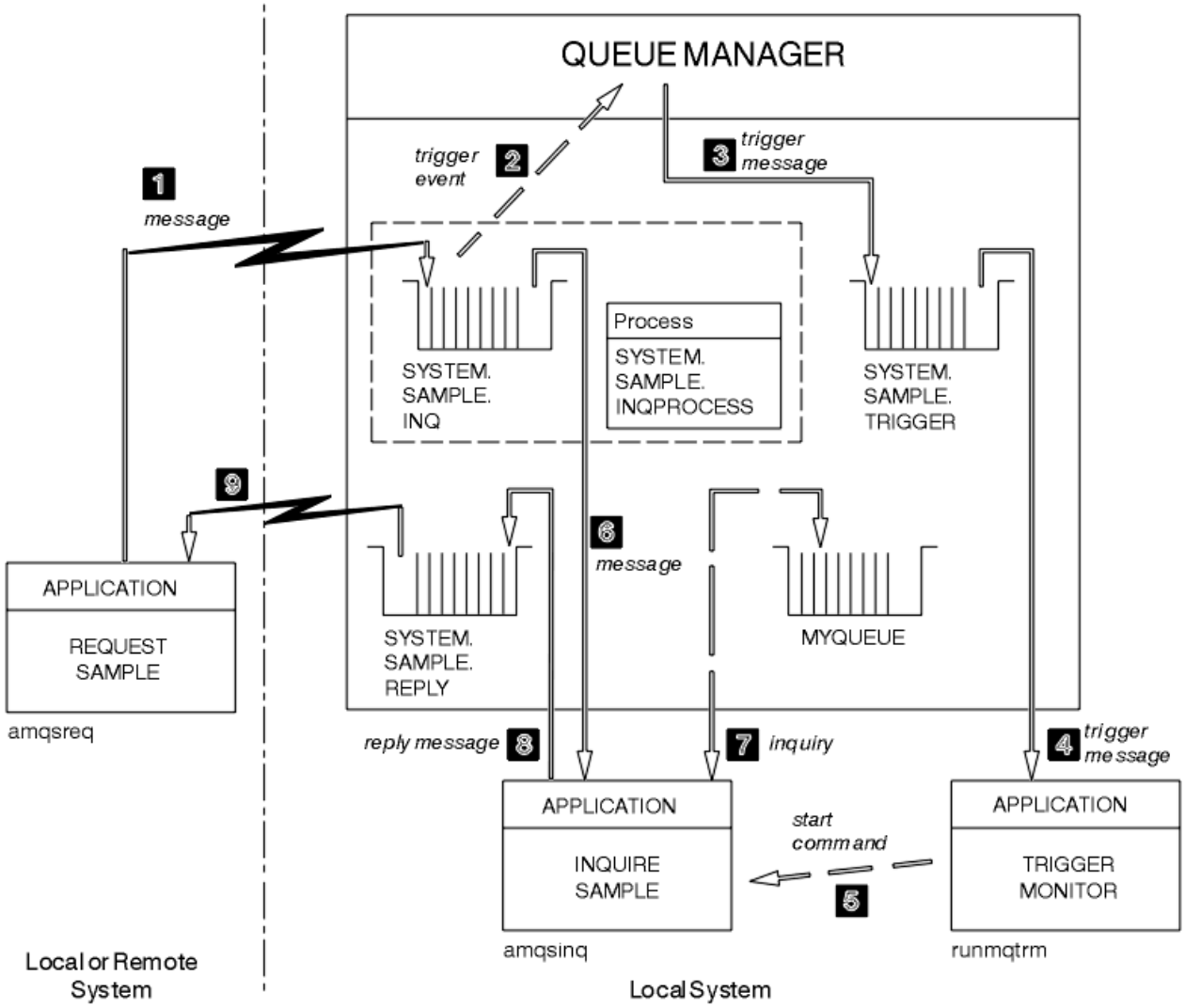


圖 *132:* 使用觸發的要求及查詢範例

在 第 937 頁的圖 132 中，「要求」範例會將訊息放入目標伺服器佇列 SYSTEM.SAMPLE.INQ 及「查詢」範例會查詢佇列 MYQUEUE。 或者，您可以使用執行 amqscos0.tst 時所定義的其中一個範例佇列，或您已定義的任何其他佇列，來進行「查詢」範例。

註: 第 937 頁的圖 132 中的數字顯示事件的順序。

若要使用觸發來執行「要求」及「查詢」範例，請執行下列動作:

1. 請檢查是否已定義您要使用的佇列。 執行 amqscos0.tst，以定義範例佇列，並定義佇列 MYQUEUE。
2. 執行觸發監視器指令 RUNMQTRM:

```
RUNMQTRM -m qmanagername -q SYSTEM.SAMPLE.TRIGGER
```

3. 執行要求範例

```
amqsreq SYSTEM.SAMPLE.INQ
```

註: 程序物件定義要觸發的項目。 如果用戶端和伺服器不在相同的平台上執行，則觸發監視器所啟動的任何處理程序都必須定義 *ApplType*，否則伺服器會採用其預設定義 (亦即，通常與伺服器機器相關聯的應用程式類型) 並導致失敗。

如需應用程式類型的清單，請參閱 ApplType。

4. 輸入您要「查詢」範例使用的佇列名稱:

```
MYQUEUE
```

5. 輸入空白行 (以結束「要求」程式)。
6. 然後要求範例會顯示一則訊息，其中包含「查詢」程式從 MYQUEUE 取得的資料。

您可以使用多個佇列; 在此情況下，請在步驟 第 938 頁的『4』中輸入其他佇列的名稱。

如需觸發的相關資訊，請參閱 第 725 頁的『使用觸發程式啟動 IBM MQ 應用程式』。

IBM i **在 *IBM i* 上使用觸發來執行要求範例**

在 IBM i 上，在一個工作中啟動範例觸發程式伺服器 AMQSERV4，然後在另一個工作中啟動 AMQSREQ4 。這表示當要求範例程式傳送訊息時，觸發程式伺服器已備妥。

註:

1. AMQSAMP4 所建立的範例定義會觸發範例的 C 版本。 如果您想要觸發 COBOL 版本，請變更程序定義 SYSTEM.SAMPLE.ECHOPROCESS， SYSTEM.SAMPLE.INQPROCESS。 您可以使用 CHGMQMPRC 指令 (如需詳細資料，請參閱 變更 MQ 處理程序 (CHGMQMPRC)) 執行此動作，或編輯並執行您自己的 AMQSAMP4 版本。
2. AMQSERV4 的原始碼僅針對 C 語言提供。 不過，程式庫 QMQM 中提供了已編譯的版本 (您可以與 COBOL 範例搭配使用)。

您可以將要求訊息放置在下列範例伺服器佇列上:

- SYSTEM.SAMPLE.ECHO (適用於回應範例程式)
- SYSTEM.SAMPLE.INQ (適用於「查詢」範例程式)
- SYSTEM.SAMPLE.SET (適用於 Set 範例程式)

SYSTEM.SAMPLE.ECHO 程式顯示在 第 940 頁的圖 133 中。 使用範例資料檔，向此伺服器發出 C 程式要求的指令為:

```
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(ECHO)')
```

註: 此範例佇列具有觸發類型 FIRST ，因此如果在執行「要求」範例之前佇列上已有訊息，則您傳送的訊息不會觸發伺服器應用程式。

如果您想要嘗試進一步的範例，您可以嘗試下列變異:

- 使用 AMQSTRG4 (或其指令行對等 STRMQMTRM ，如需詳細資料，請參閱 啟動 MQ 觸發監視器 (STRMQMTRM)) 而不是 AMQSERV4 來提交工作，但潛在的工作提交延遲可能會讓您更不容易追蹤發生的情況。
- 執行 SYSTEM.SAMPLE.INQUIRE 和 SYSTEM.SAMPLE.SET 範例程式。 使用範例資料檔，向這些伺服器發出 C 程式要求的指令分別為:

```
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(INQ)')
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(SET)')
```

這些範例佇列也具有 FIRST 觸發類型。

要求範例程式的設計

程式會開啟目標伺服器佇列，以便它可以放置訊息。 它會搭配使用 MQOPEN 呼叫與 MQOO_OUTPUT 選項。 如果無法開啟佇列，程式會顯示一則錯誤訊息，其中包含 MQOPEN 呼叫所傳回的原因碼。

然後，程式會開啟名為 SYSTEM.SAMPLE.REPLY 以便它可以取得回覆訊息。 為此，程式會使用 MQOPEN 呼叫並指定 MQOO_INPUT_EXCLUSIVE 選項。 如果無法開啟佇列，程式會顯示一則錯誤訊息，其中包含 MQOPEN 呼叫所傳回的原因碼。

然後，針對每一行輸入，程式會將文字讀取到緩衝區中，並使用 MQPUT 呼叫來建立包含該行文字的要求訊息。 在此呼叫上，程式會使用 MQRO_EXCEPTION_WITH_DATA 報告選項，要求所傳送關於要求訊息的任何報告訊息將包括訊息資料的前 100 個位元組。 程式會繼續執行，直到到達輸入結尾或 MQPUT 呼叫失敗為止。

然後，程式會使用 MQGET 呼叫來移除佇列中的回覆訊息，並顯示回覆中包含的資料。 MQGET 呼叫使用 MQGMO_WAIT、MQGMO_CONVERT 及 MQGMO_ACCEPT_TRUNCATED 選項。 *WaitInterval* 在 COBOL 版本中是 5 分鐘，在 C 版本中是 1 分鐘，用於第一次回覆 (容許觸發伺服器應用程式的時間) ，在後續回覆中是 15 秒。 如果佇列上沒有訊息，則程式會等待這些期間。 如果在此間隔到期之前沒有任何訊息到達，則呼叫會失敗並傳回 MQRC_NO_MSG_AVAILABLE 原因碼。 此呼叫也會使用 MQGMO_ACCEPT_TRUNCATED_MSG 選項，因此會截斷長於所宣告緩衝區大小的訊息。

此程式示範如何在每一個 MQGET 呼叫之後清除 MQMD 結構的 *MsgId* 及 *CorrelId* 欄位，因為呼叫會將這些欄位設為其所擷取訊息中包含的值。 清除這些欄位表示後續的 MQGET 呼叫會依照訊息保留在佇列中的順序來擷取訊息。

程式會繼續執行，直到 MQGET 呼叫傳回 MQRC_NO_MSG_AVAILABLE 原因碼或 MQGET 呼叫失敗為止。 如果呼叫失敗，程式會顯示包含原因碼的錯誤訊息。

然後，程式會使用 MQCLOSE 呼叫來關閉目標伺服器佇列及回覆目的地佇列。

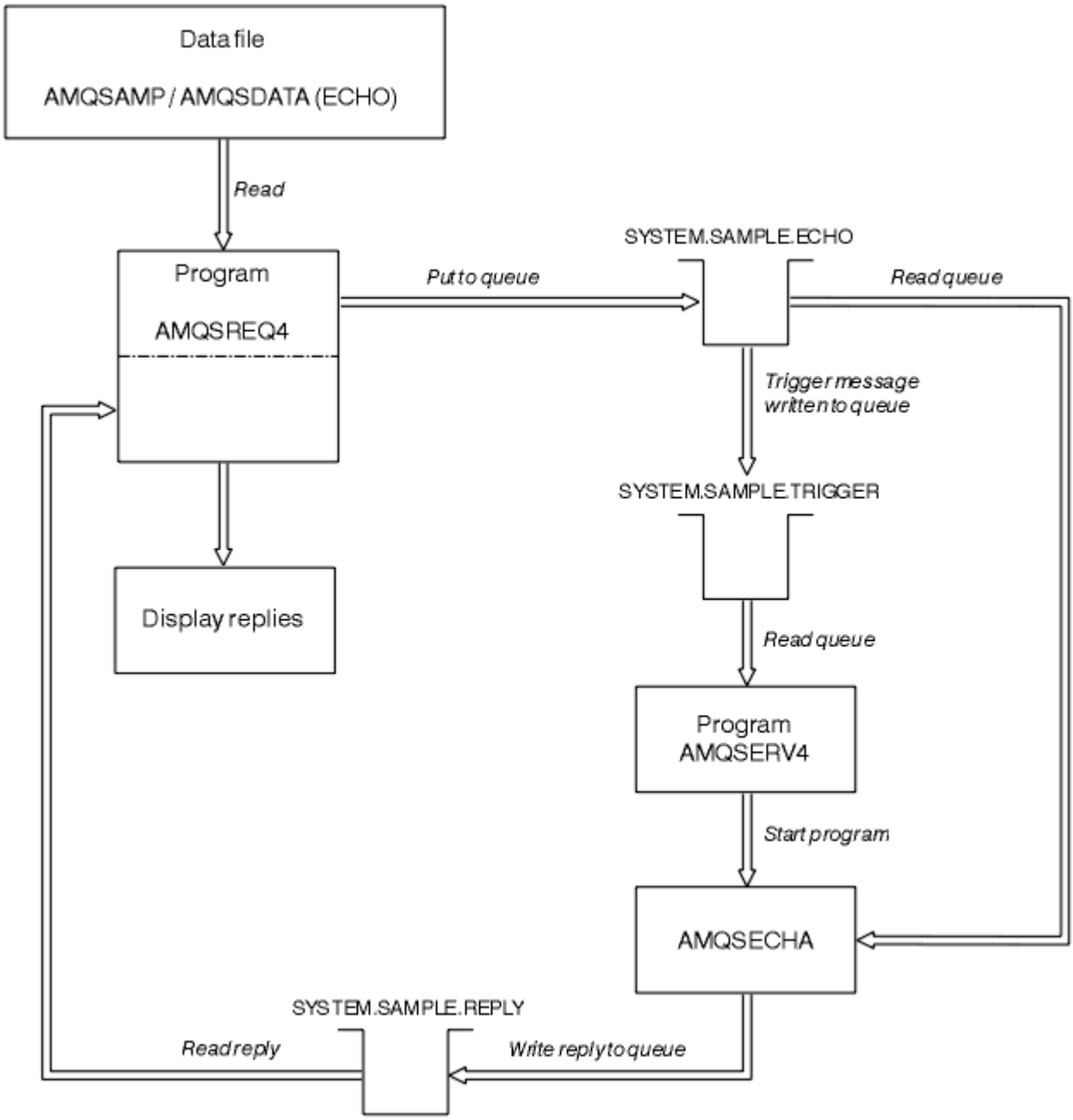


圖 *133:* 範例 *IBM i Client/Server (Echo)* 程式流程圖

設定範例程式

「設定範例」程式會使用 MQSET 呼叫來變更佇列的 **InhibitPut** 屬性，以禁止佇列上的放置作業。 此外，請瞭解 Set 範例程式的設計。

如需這些程式的名稱，請參閱 第 889 頁的『Multiplatforms 上範例程式中示範的特性』 。

這些程式預期作為觸發程式執行，因此它們的唯一輸入是 MQTMC2 (觸發訊息) 結構，其中包含目標佇列的名稱，以及要查詢的屬性。 C 版本也會使用佇列管理程式名稱。 COBOL 版本使用預設佇列管理程式。

若要讓觸發程序運作，請確定您要使用的 Set 範例程式是由到達佇列 SYSTEM.SAMPLE.SET 的訊息所觸發。 若要這樣做，請在程序定義 SYSTEM.SAMPLE.SETPROCESS 的 *ApplicId* 欄位中指定您要使用的 Set 範例程式名稱。 範例佇列的觸發類型為 FIRST; 如果在執行「要求」範例之前佇列上已有訊息，則您傳送的訊息不會觸發「設定」範例。

當您已正確設定定義時:

- ALW 若為 AIX, Linux, and Windows 系統，請在一個階段作業中啟動 **runmqtrm** 程式，然後在另一個階段作業中啟動 amqsreq 程式。
- IBM i 若為 IBM i，請在一個階段作業中啟動 AMQSERV4 程式，然後在另一個階段作業中啟動 AMQSREQ4 程式。 您可以使用 AMQSTRG4 而非 AMQSERV4，但潛在的工作提交延遲可能會讓您更不容易追蹤發生的情況。

使用「要求範例」程式，將要求訊息 (每則只包含佇列名稱) 傳送至佇列 SYSTEM.SAMPLE.SET。 對於每一個要求訊息，「設定範例」程式會傳送回覆訊息，其中包含已在指定佇列上禁止放置作業的確認。 回覆會傳送至要求訊息中指定的回覆目的地佇列。

Set 範例程式的設計

程式會開啟在啟動時所傳遞的觸發訊息結構中指名的佇列。 (為了明確起見，我們將此稱為 要求佇列。) 程式使用 MQOPEN 呼叫來開啟此佇列以供共用輸入。

程式會使用 MQGET 呼叫來移除此佇列中的訊息。 此呼叫使用 MQGMO_ACCEPT_TRUNCATED_MSG 和 MQGMO_WAIT 選項，等待間隔為 5 秒。 程式會測試每一個訊息的描述子，以查看它是否為要求訊息; 如果不是，則程式會捨棄訊息，並顯示警告訊息。

對於從要求佇列中移除的每一個要求訊息，程式會讀取佇列的名稱 (我們將呼叫 目標佇列) 包含在資料中，並使用 MQOO_SET 選項搭配 MQOPEN 呼叫來開啟該佇列。 然後，程式會使用 MQSET 呼叫，將目標佇列的 **InhibitPut** 屬性值設為 MQQA_PUT_INHIBITED。

如果 MQSET 呼叫成功，則程式會使用 MQPUT1 呼叫，將回覆訊息放置在回覆目的地佇列上。 此訊息包含字串 PUT inhibited。

如果 MQOPEN 或 MQSET 呼叫不成功，則程式會使用 MQPUT1 呼叫，將 report 訊息放置在回覆目的地佇列上。 在此報告訊息之訊息描述子的 *Feedback* 欄位中，是 MQOPEN 或 MQSET 呼叫所傳回的原因碼，視失敗的原因碼而定。

在 MQSET 呼叫之後，程式會使用 MQCLOSE 呼叫來關閉目標佇列。

當要求佇列中沒有剩餘訊息時，程式會關閉該佇列，並切斷與佇列管理程式的連線。

TLS 範例程式

AMQSSSLC 是一個範例 C 程式，示範如何使用 MQCNO 及 MQSCO 結構來提供 MQCONNX 呼叫的 TLS 用戶端連線資訊。 這可讓用戶端 MQI 應用程式在執行時期提供其用戶端連線通道及 TLS 設定的定義，而不需要用戶端通道定義表 (CCDT)。

如果提供連線名稱，則程式會在 MQCD 結構中建構用戶端連線通道定義。

如果提供金鑰儲存庫檔的系統名稱，則程式會建構 MQSCO 結構; 如果也提供 OCSP 回應端 URL，則程式會建構鑑別資訊記錄 MQAIR 結構。

然後，程式會使用 MQCONNX 連接至佇列管理程式。 它會查詢並列印出它所連接的佇列管理程式名稱。

此程式預期鏈結為 MQI 用戶端應用程式。 不過，它可以鏈結成一般 MQI 應用程式，在這種情況下，它只會連接至本端佇列管理程式，並忽略用戶端連線資訊。

V 9.3.0 V 9.3.0 如果用來存取金鑰儲存庫的通行詞組未隱藏在檔案中，您必須在應用程式執行時提供通行詞組給 **amqssslc** 。 您可以透過下列方式提供通行詞組:

- 要求 **amqssslc** 提示輸入通行詞組，或
- 使用 *MQKEYRPWD* 環境變數，或
- 在用戶端配置檔中使用 **SSLKeyRepositoryPassword** 屬性

如需將金鑰儲存庫密碼提供給 IBM MQ MQI client 應用程式的相關資訊，請參閱 在 AIX, Linux, and Windows 上提供 IBM MQ MQI client 的金鑰儲存庫密碼 。

amqssslc 接受下列參數，所有這些都是選用參數:

-m QmgrName
要連接的佇列管理程式名稱

-c ChannelName
要使用的通道名稱

-x ConnName
伺服器連線名稱

TLS 參數:

V 9.3.0 V 9.3.0 **-k KeyReposFileName**
金鑰儲存庫檔案的名稱。 如果未提供副檔名，則會假設它是 .kdb。 例如:

```
/home/user/client.kdb
C:\User\client.p12
```

-s CipherSpec
TLS 通道 CipherSpec 字串，對應於佇列管理程式上 SVRCONN 通道定義的 **SSLCIPH** 。

-f
指定只必須使用 FIPS 140-2 認證的演算法。

-b VALUE1[, VALUE2...]
指定只必須使用套組 B 相容演算法。 此參數是下列一或多個值的逗點區隔清單: NONE,128_BIT,192_BIT。 這些值的意義與 **MQSUITEB** 環境變數的意義相同，且在用戶端配置檔 SSL 段落中具有對等的 **EncryptionPolicySuiteB** 設定。

-p 原則
指定要使用的憑證驗證原則。 這可以是下列其中一個值:

ANY
套用 Secure Socket Library 所支援的每一個憑證驗證原則，並接受憑證鏈 (如果有任何原則認為憑證鏈有效的話)。 此設定可用於與不符合現代憑證標準的舊數位憑證的最大舊版相容性。

RFC5280
僅套用 RFC 5280 相容憑證驗證原則。 此設定提供比 ANY 設定更嚴格的驗證，但拒絕部分較舊的數位憑證。

預設值為任何 (any)。

-l CertLabel
用於安全連線的憑證標籤。

註: 您必須使用小寫字元來指定值。

V 9.3.0 V 9.3.0 **-w**
指定 **amqssslc** 提示輸入要提供的金鑰儲存庫通行詞組。

V 9.3.0 V 9.3.0 **-i**
指定 **amqssslc** 提示輸入用來加密要提供之金鑰儲存庫通行詞組的起始金鑰。

如果在使用 **runmqicred** 公用程式加密金鑰儲存庫通行詞組時指定起始金鑰檔，請指定此選項。

OCSP 憑證撤銷參數:

-o URL
OCSP 回應端 URL

您也可以設定下列其中一個環境變數，以提供用來向佇列管理程式進行鑑別的認證:

MQSAMP_USER_ID
如果您想要使用使用者 ID 和密碼向佇列管理程式進行鑑別，請設為用於連線鑑別的使用者 ID。 程式會提示輸入密碼以隨附使用者 ID。

Linux AIX V 9.3.4 **MQSAMP_TOKEN**
如果您要提供鑑別記號以向佇列管理程式進行鑑別，請設為非空白值。 程式會提示輸入鑑別記號。

執行 *TLS* 範例程式
若要執行 TLS 範例程式，您必須先設定 TLS 環境。 然後從指令行執行範例，並提供一些參數。

關於這項作業

下列指示使用個人憑證來執行範例程式。 例如，透過改變指令，您可以使用 CA 憑證，並使用 OCSP 回應端來檢查其狀態。 請參閱範例內的指示。

程序

1. 建立名稱為 QM1 的佇列管理程式。 如需相關資訊，請參閱 crtmqm。
2. 建立佇列管理程式的金鑰儲存庫。 如需相關資訊，請參閱 在 AIX, Linux, and Windows 上設定金鑰儲存庫。
3. 建立用戶端的金鑰儲存庫。 請將它稱為 *clientkey.kdb*。

   建立金鑰儲存庫時，將金鑰儲存庫密碼隱藏在檔案中。
4. 建立佇列管理程式的個人憑證。 如需相關資訊，請參閱 在 AIX, Linux, and Windows 上建立自簽個人憑證。
5. 建立用戶端的個人憑證。
6. 從伺服器金鑰儲存庫擷取個人憑證，並將它新增至用戶端儲存庫。 如需相關資訊，請參閱 從 AIX, Linux, and Windows 上的金鑰儲存庫擷取自簽憑證的公用部分，以及 在 AIX, Linux, and Windows 系統上將 CA 憑證 (或自簽憑證的公用部分) 新增至金鑰儲存庫。
7. 從用戶端金鑰儲存庫擷取個人憑證，並將它新增至伺服器金鑰儲存庫。
8. 使用 MQSC 指令建立伺服器連線通道:

   ```
   DEFINE CHANNEL(QM1SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
   SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA256)
   ```

   如需相關資訊，請參閱 伺服器連線通道
9. 在佇列管理程式上定義並啟動通道接聽器。 如需相關資訊，請參閱 DEFINE LISTENER 和 START LISTENER。
10. 使用下列指令執行範例程式:

    V 9.3.0 V 9.3.0

    ```
    AMQSSSLC -m QM1 -c QM1SVRCONN -x localhost
    -k "C:\Program Files\IBM\MQ\clientkey.kdb" -s TLS_RSA_WITH_AES_128_CBC_SHA256
    -o http://dummy.OCSP.responder
    ```

結果

範例程式會執行下列動作:

1. 使用任何指定的選項，連接至任何指定的佇列管理程式或預設佇列管理程式。
2. 開啟佇列管理程式並查詢其名稱。
3. 關閉佇列管理程式。
4. 切斷與佇列管理程式的連線。

如果範例程式順利執行，它會顯示類似下列範例的輸出:

```
Sample AMQSSSLC start
Connecting to queue manager QM1
Using the server connection channel QM1SVRCONN
on connection name localhost.
Using TLS CipherSpec TLS_RSA_WITH_AES_128_CBC_SHA256
Using TLS key repository stem C:\Program Files\IBM\MQ\clientkey
Using OCSP responder URL http://dummy.OCSP.responder
Connection established to queue manager QM1

Sample AMQSSSLC end
```

如果範例程式遇到問題，則會顯示適當的錯誤訊息，例如，如果您指定無效的 OCSP 回應端 URL，則會收到下列訊息:

```
MQCONNX ended with reason code 2553
```

如需原因碼清單，請參閱 API 完成及原因碼。

觸發程式範例

觸發範例中提供的功能是 **runmqtrm** 程式中觸發監視器中提供的功能子集。

如需這些程式的名稱，請參閱 第 889 頁的『Multiplatforms 上範例程式中示範的特性』 。

觸發範例的設計

觸發範例程式會使用 MQOPEN 呼叫搭配 MQOO_INPUT_AS_Q_DEF 選項來開啟起始佇列。 它會使用 MQGET 呼叫並指定 MQGMO_ACCEPT_TRUNCATED_MSG 及 MQGMO_WAIT 選項，從起始佇列取得訊息，並指定無限制等待間隔。 程式會在每一個 MQGET 呼叫之前清除 *MsgId* 和 *CorrelId* 欄位，以依序取得訊息。

當它從起始佇列擷取訊息時，程式會檢查訊息大小來測試訊息，以確定其大小與 MQTM 結構相同。 如果此測試失敗，程式會顯示警告。

對於有效的觸發訊息，觸發範例會從下列欄位複製資料: *ApplicId*、 *EnvrData*、 *Version* 及 *ApplType*。 這些欄位的最後兩個是數值，因此程式會建立字元取代，以在 IBM i AIX, Linux, and Windows 系統的 MQTMC2 結構中使用。

觸發範例會向觸發訊息的 *ApplicId* 欄位中指定的應用程式發出啟動指令，並傳遞 MQTMC2 或 MQTMC (觸發訊息的字元版本) 結構。

- ALW 在 AIX, Linux, and Windows 系統中， *EnvData* 欄位用來作為呼叫指令字串的延伸。
- IBM i 在 IBM i 中，它用作工作提交參數，例如，工作優先順序或工作說明。

最後，程式會關閉起始佇列。

在 IBM i 上結束觸發範例程式

IBM i

觸發監視器程式可以由 sysrequest 選項 2 (ENDRQS) 或禁止從觸發佇列取得來結束。

如果使用範例觸發佇列，則指令為:

```
CHGMQMQ QNAME('SYSTEM.SAMPLE.TRIGGER') MQMNAME GETENBL(*NO)
```

重要: 在此佇列上重新開始觸發之前，您必須輸入下列指令:

```
CHGMQMQ QNAME('SYSTEM.SAMPLE.TRIGGER') GETENBL(*YES)
```

執行觸發程式範例

本主題包含執行 Triggering 範例程式的相關資訊。

執行 amqstrg0.c、amqstrg 及 amqstrgc 範例

程式採用 2 個參數:

1. 起始佇列的名稱 (必要)
2. 佇列管理程式的名稱 (選用)

如果未指定佇列管理程式，則會連接至預設佇列管理程式。 當您執行 amqscos0.tst; 時，會定義範例起始佇列; 該佇列的名稱是 SYSTEM.SAMPLE.TRIGGER，您可以在執行此程式時使用它。

註: 此範例中的函數是 runmqtrm 程式中提供之完整觸發函數的子集。

執行 AMQSTRG4 範例

IBM i

這是 IBM i 環境的觸發監視器。 它會針對每一個要啟動的應用程式提交一個 IBM i 工作。 這表示每一個觸發訊息都有相關聯的額外處理。

AMQSTRG4 (在 QCSRC 中) 採用兩個參數: 它要提供的起始佇列名稱，以及佇列管理程式的名稱 (選用)。 AMQSAMP4 (在 QCLSRC 中) 定義範例起始佇列 SYSTEM.SAMPLE.TRIGGER，可在您嘗試範例程式時使用。

使用範例觸發佇列，要發出的指令為:

```
CALL PGM(QMQM/AMQSTRG4) PARM('SYSTEM.SAMPLE.TRIGGER')
```

或者，您可以使用 CL 對等的 STRMQMTRM; 如需詳細資料，請參閱 啟動 MQ 觸發監視器 (STRMQMTRM)。

執行 AMQSERV4 範例

IBM i

這是 IBM i 環境的觸發程式伺服器。 對於每一個觸發訊息，此伺服器會在其自己的工作中執行啟動指令，以啟動指定的應用程式。 觸發程式伺服器可以呼叫 CICS 交易。

AMQSERV4 採用兩個參數: 它要提供的起始佇列名稱，以及佇列管理程式的名稱 (選用)。 AMQSAMP4 定義範例起始佇列 SYSTEM.SAMPLE.TRIGGER，可在您嘗試範例程式時使用。

使用範例觸發佇列時，要發出的指令為:

```
CALL PGM(QMQM/AMQSERV4) PARM('SYSTEM.SAMPLE.TRIGGER')
```

觸發程式伺服器的設計
觸發程式伺服器的設計類似於觸發監視器的設計，但有幾個例外

觸發伺服器的設計與觸發監視器的設計類似，除了觸發伺服器:

- 容許 MQAT_CICS 及 MQAT_OS400 應用程式。
- IBM i 在自己的工作中呼叫 IBM i 應用程式 (或使用 STRCICSUSR 來啟動 CICS 應用程式)，而不是提交 IBM i 工作。
- 對於 CICS 應用程式，例如，替換來自 STRCICSUSR 指令中觸發訊息的 *EnvData*，以指定 CICS 區域。
- 開啟共用輸入的起始佇列，讓許多觸發伺服器可以同時執行。

註: AMQSERV4 啟動的程式不得使用 MQDISC 呼叫，因為這會停止觸發伺服器。 如果由 AMQSERV4 啟動的程式使用 MQCONN 呼叫，則它們會收到 MQRC_ALREADY_CONNECTED 原因碼。

ALW 在 *AIX, Linux, and Windows* 上使用 *TUXEDO* 範例

瞭解 TUXEDO 的 Put and Get 範例程式，並在 TUXEDO 中建置伺服器環境。

開始之前

在執行這些範例之前，您必須先建置伺服器環境。

關於這項作業

註: 在整個區段中，反斜線 (\) 字元用來將長指令分割成多行。 請勿輸入此字元。 以單行輸入每一個指令。

ALW 建置伺服器環境
針對不同平台建置 IBM MQ 的伺服器環境的相關資訊。

開始之前

假設您具有工作中的 TUXEDO 環境。

AIX 建置 *AIX (32 位元)* 的伺服器環境
如何為 IBM MQ for AIX (32 位元) 建置伺服器環境。

程序

1. 建立在其中建置伺服器環境的目錄 (例如， APPDIR) ， 並執行此目錄中的所有指令。
2. 匯出下列環境變數，其中 TUXDIR 是 TUXEDO 的根目錄，而 *MQ_INSTALLATION_PATH* 代表 IBM MQ 安裝所在的高階目錄:

```
$ export CFLAGS="-I MQ_INSTALLATION_PATH/inc -I /APPDIR -L MQ_INSTALLATION_PATH/lib"
$ export LDOPTS="-lmqm"
$ export FIELDTBLS= MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=/APPDIR/amqstxvx.V
$ export LIBPATH=$TUXDIR/lib: MQ_INSTALLATION_PATH/lib:/lib
```

3. 將下列行新增至 TUXEDO 檔案 udataobj/RM:

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: -lmqmxa -lmqm
```

4. 執行下列指令:

```
$ mkfldhdr    MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc       MQ_INSTALLATION_PATH/samp/amqstxvx.v
$ buildtms    -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
$ buildserver -o MQSERV2 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT2:MPUT
       -s MGET2:MGET \
       -v -bshm
$ buildclient -o doputs -f MQ_INSTALLATION_PATH/samp/amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a
$ buildclient -o dogets -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a
```

5. 編輯 ubbstxcx.cfg ， 並視需要新增機器名稱、工作目錄及佇列管理程式的詳細資料:

```
$ tmloadcf  -y MQ_INSTALLATION_PATH/samp/ubbstxcx.cfg
```

6. 建立 TLOGDEVICE:

```
$tmadmin -c
```

即會顯示提示。 在此提示中，輸入:

```
> crdl -z /APPDIR/TLOG1
```

7. 啟動佇列管理程式:

```
$ strmqm
```

8. 啟動 Tuxedo:

```
$ tmboot -y
```

下一步

現在您可以使用 doputs 及 dogets 程式，將訊息放入佇列並從佇列中擷取訊息。

AIX 建置 *AIX (64 位元)* 的伺服器環境
如何為 IBM MQ for AIX (64 位元) 建置伺服器環境。

程序

1. 建立在其中建置伺服器環境的目錄 (例如， APPDIR) ，並執行此目錄中的所有指令。
2. 匯出下列環境變數，其中 TUXDIR 代表 TUXEDO 的根目錄， *MQ_INSTALLATION_PATH* 代表 IBM MQ 安裝 installed.:

```
$ export CFLAGS="-I MQ_INSTALLATION_PATH/inc -I /APPDIR -L MQ_INSTALLATION_PATH/lib64"
$ export LDOPTS="-lmqm"
$ export FIELDTBLS= MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=/APPDIR/amqstxvx.V
$ export LIBPATH=$TUXDIR/lib64: MQ_INSTALLATION_PATH/lib64:/lib64
```

3. 將下列行新增至 TUXEDO 檔案 udataobj/RM:

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: -lmqmxa64 -lmqm
```

4. 執行下列指令:

```
$ mkfldhdr    MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc       MQ_INSTALLATION_PATH/samp/amqstxvx.v
$ buildtms    -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
$ buildserver -o MQSERV2 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT2:MPUT
       -s MGET2:MGET \
       -v -bshm
$ buildclient -o doputs -f MQ_INSTALLATION_PATH/samp/amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a
$ buildclient -o dogets -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a
```

5. 編輯 ubbstxcx.cfg ，並視需要新增機器名稱、工作目錄及佇列管理程式的詳細資料:

```
$ tmloadcf  -y MQ_INSTALLATION_PATH/samp/ubbstxcx.cfg
```

6. 建立 TLOGDEVICE:

```
$tmadmin -c
```

即會顯示提示。 在此提示中，輸入:

```
> crdl -z /APPDIR/TLOG1
```

7. 啟動佇列管理程式:

```
$ strmqm
```

8. 啟動 Tuxedo:

```
$ tmboot -y
```

下一步

現在您可以使用 doputs 及 dogets 程式，將訊息放入佇列並從佇列中擷取訊息。

Windows 建置 *Windows (32 位元)* 的伺服器環境
建置 IBM MQ for Windows (32 位元) 的伺服器環境。

關於這項作業

註: 將下列中識別為 *VARIABLES* 的欄位變更為目錄路徑:

表 *164:* 切換至目錄路徑的欄位

欄位	目錄路徑
MQMDIR	安裝 IBM MQ 時指定的目錄路徑，例如 `g:\Program Files\IBM\MQ`。
TUXDIR	安裝 TUXEDO 時指定的目錄路徑，例如 `f:\tuxedo`。
APPDIR	用於範例應用程式的目錄路徑，例如 `f:\tuxedo\apps\mqapp`。

```
*RESOURCES
IPCKEY       99999
UID          0
GID          0
MAXACCESSERS 20
MAXSERVERS   20
MAXSERVICES  50
MASTER       SITE1
MODEL        SHM
LDBAL        N

*MACHINES
MachineName LMID=SITE1
            TUXDIR="f:\tuxedo"
            APPDIR="f:\tuxedo\apps\mqapp;g:\Program Files\IBM\WebSphere MQ\bin"
            ENVFILE="f:\tuxedo\apps\mqapp\amqstxen.env"
            TUXCONFIG="f:\tuxedo\apps\mqapp\tuxconfig"
            ULOGPFX="f:\tuxedo\apps\mqapp\ULOG"
            TLOGDEVICE="f:\tuxedo\apps\mqapp\TLOG"
            TLOGNAME=TLOG
            TYPE="i386NT"
            UID=0
            GID=0

*GROUPS
GROUP1
        LMID=SITE1 GRPNO=1
        TMSNAME=MQXA
        OPENINFO="MQSERIES_XA_RMI:MYQUEUEMANAGER"

*SERVERS
DEFAULT: CLOPT="-A -- -m MYQUEUEMANAGER"

MQSERV1   SRVGRP=GROUP1 SRVID=1
MQSERV2   SRVGRP=GROUP1 SRVID=2

*SERVICES
MPUT1
MGET1
MPUT2
MGET2
```

圖 *134: IBM MQ for Windows* 的 *ubbstxcn.cfg* 檔案範例

註: 變更機器名稱 *MachineName* 及目錄路徑，以符合您的安裝。 同時將佇列管理程式名稱 *MYQUEUEMANAGER* 變更為您要連接的佇列管理程式名稱。

IBM MQ for Windows 的範例 `ubbconfig` 檔列在 第 948 頁的圖 134 中。 它在 IBM MQ 範例目錄中以 `ubbstxcn.cfg` 提供。

提供給 IBM MQ for Windows 的範例 make 檔 (請參閱 第 949 頁的圖 135) 稱為 ubbstxmn.mak，並保留在 IBM MQ 範例目錄中。

```
TUXDIR  = f:\tuxedo
MQMDIR  = g:\Program Files\IBM\WebSphere MQ
APPDIR  = f:\tuxedo\apps\mqapp
MQMLIB  = $(MQMDIR)\tools\lib
MQMINC  = $(MQMDIR)\tools\c\include
MQMSAMP = $(MQMDIR)\tools\c\samples
INC = -f "-I$(MQMINC) -I$(APPDIR)"
DBG = -f "/Zi"

amqstx.exe:
 $(TUXDIR)\bin\mkfldhdr    -d$(APPDIR) $(MQMSAMP)\amqstxvx.fld
 $(TUXDIR)\bin\viewc       -d$(APPDIR) $(MQMSAMP)\amqstxvx.v
 $(TUXDIR)\bin\buildtms    -o MQXA -r MQSERIES_XA_RMI
 $(TUXDIR)\bin\buildserver -o MQSERV1 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT1:MPUT -s MGET1:MGET
 $(TUXDIR)\bin\buildserver -o MQSERV2 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT2:MPUT -s MGET2:MGET
 $(TUXDIR)\bin\buildclient -o doputs -f $(MQMSAMP)\amqstxpx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG)
$(TUXDIR)\bin\buildclient  -o dogets -f $(MQMSAMP)\amqstxgx.c \
                           -f $(MQMLIB)\mqm.lib $(INC) -v $(DBG)
 $(TUXDIR)\bin\tmloadcf    -y $(APPDIR)\ubbstxcn.cfg
```

圖 *135: IBM MQ for Windows* 的 *TUXEDO make* 檔範例

如果要建置伺服器環境和範例，請完成下列步驟。

程序

1. 建立要在其中建置範例應用程式的應用程式目錄，例如:

   ```
   f:\tuxedo\apps\mqapp
   ```

2. 將下列範例檔從 IBM MQ 範例目錄複製到應用程式目錄:
   - amqstxmn.mak
   - amqstxen.env
   - ubbstxcn.cfg
3. 編輯每一個檔案，以設定安裝上使用的目錄名稱及目錄路徑。
4. 編輯 ubbstxcn.cfg (請參閱 第 948 頁的圖 134)，以新增您要連接之機器名稱及佇列管理程式的詳細資料。
5. 將下列行新增至 TUXEDO 檔案 *TUXDIR*udataobj\rm:

   ```
   MQSERIES_XA_RMI;MQRMIXASwitchDynamic;MQMDIR\tools\lib\mqmxa.lib MQMDIR\tools\lib\mqm.lib
   ```

   在檔案中，新項目必須是一行。
6. 設定下列環境變數:

   ```
   TUXDIR=TUXDIR
   TUXCONFIG=APPDIR\tuxconfig
   FIELDTBLS=MQMDIR\tools\c\samples\amqstxvx.fld
   LANG=C
   ```

7. 建立 TUXEDO 的 TLOG 裝置。

若要這麼做，請呼叫 tmadmin -c，並輸入下列指令:

```
crdl -z APPDIR\TLOG
```

8. 將現行目錄設為 *APPDIR*，並呼叫範例 make 檔 amqstxmn.mak 作為外部專案 make 檔。 例如，使用 Microsoft Visual C++ ，發出下列指令:

```
msvc amqstxmn.mak
```

選取 **建置** ，以建置所有範例程式。

Windows 建置 *Windows (64 位元)* 的伺服器環境
如何為 IBM MQ for Windows (64 位元) 建置伺服器環境。

關於這項作業

註: 將下列中識別為 *VARIABLES* 的欄位變更為目錄路徑:

表 *165:* 切換至目錄路徑的欄位

欄位	**目錄路徑**
MQMDIR	安裝 IBM MQ 時指定的目錄路徑，例如 g:\Program Files\IBM\MQ。
TUXDIR	安裝 TUXEDO 時指定的目錄路徑，例如 f:\tuxedo。
APPDIR	用於範例應用程式的目錄路徑，例如 f:\tuxedo\apps\mqapp。

```
*RESOURCES
IPCKEY       99999
UID          0
GID          0
MAXACCESSERS 20
MAXSERVERS   20
MAXSERVICES  50
MASTER       SITE1
MODEL        SHM
LDBAL        N

*MACHINES
MachineName LMID=SITE1
            TUXDIR="f:\tuxedo"
            APPDIR="f:\tuxedo\apps\mqapp;g:\Programï¿½Files\IBM\WebSphere MQ\bin"
            ENVFILE="f:\tuxedo\apps\mqapp\amqstxen.env"
            TUXCONFIG="f:\tuxedo\apps\mqapp\tuxconfig"
            ULOGPFX="f:\tuxedo\apps\mqapp\ULOG"
            TLOGDEVICE="f:\tuxedo\apps\mqapp\TLOG"
            TLOGNAME=TLOG
            TYPE="i386NT"
            UID=0
            GID=0

*GROUPS
GROUP1
        LMID=SITE1 GRPNO=1
        TMSNAME=MQXA
        OPENINFO="MQSERIES_XA_RMI:MYQUEUEMANAGER"

*SERVERS
DEFAULT: CLOPT="-A -- -m MYQUEUEMANAGER"

MQSERV1   SRVGRP=GROUP1 SRVID=1
MQSERV2   SRVGRP=GROUP1 SRVID=2

*SERVICES
MPUT1
MGET1
MPUT2
MGET2
```

圖 *136: IBM MQ for Windows* 的 *ubbstxcn.cfg* 檔案範例

註: 變更機器名稱 *MachineName* 及目錄路徑，以符合您的安裝。 同時將佇列管理程式名稱 *MYQUEUEMANAGER* 變更為您要連接的佇列管理程式名稱。

第 951 頁的圖 136 中列出 IBM MQ for Windows 的範例 ubbconfig 檔案。 它在 IBM MQ 範例目錄中以 `ubbstxcn.cfg` 提供。

範例 make 檔 (請參閱 第 952 頁的圖 137) 針對 IBM MQ for Windows 提供的稱為 `ubbstxmn.mak`，並保留在 IBM MQ 範例目錄中。

```
TUXDIR  = f:\tuxedo
MQMDIR  = g:\Program Files\IBM\WebSphere MQ
APPDIR  = f:\tuxedo\apps\mqapp
MQMLIB  = $(MQMDIR)\tools\lib64
MQMINC  = $(MQMDIR)\tools\c\include
MQMSAMP = $(MQMDIR)\tools\c\samples
INC = -f "-I$(MQMINC) -I$(APPDIR)"
DBG = -f "/Zi"

amqstx.exe:
 $(TUXDIR)\bin\mkfldhdr    -d$(APPDIR) $(MQMSAMP)\amqstxvx.fld
 $(TUXDIR)\bin\viewc       -d$(APPDIR) $(MQMSAMP)\amqstxvx.v
 $(TUXDIR)\bin\buildtms    -o MQXA -r MQSERIES_XA_RMI
 $(TUXDIR)\bin\buildserver -o MQSERV1 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT1:MPUT -s MGET1:MGET
 $(TUXDIR)\bin\buildserver -o MQSERV2 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT2:MPUT -s MGET2:MGET
 $(TUXDIR)\bin\buildclient -o doputs -f $(MQMSAMP)\amqstxpx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG)
 $(TUXDIR)\bin\buildclient -o dogets -f $(MQMSAMP)\amqstxgx.c \
                           -f $(MQMLIB)\mqm.lib $(INC) -v $(DBG)
 $(TUXDIR)\bin\tmloadcf    -y $(APPDIR)\ubbstxcn.cfg
```

圖 *137: IBM MQ for Windows* 的 *TUXEDO make* 檔範例

如果要建置伺服器環境和範例，請完成下列步驟。

程序

1. 建立要在其中建置範例應用程式的應用程式目錄，例如:

   ```
   f:\tuxedo\apps\mqapp
   ```

2. 將下列範例檔從 IBM MQ 範例目錄複製到應用程式目錄:
   - amqstxmn.mak
   - amqstxen.env
   - ubbstxcn.cfg
3. 編輯每一個檔案，以設定安裝上使用的目錄名稱及目錄路徑。
4. 編輯 ubbstxcn.cfg (請參閱 第 951 頁的圖 136) 以新增您要連接之機器名稱及佇列管理程式的詳細資料。
5. 將下列行新增至 TUXEDO 檔案 *TUXDIR*udataobj\rm

   ```
   MQSERIES_XA_RMI;MQRMIXASwitchDynamic;MQMDIR\tools\lib64\mqmxa64.lib
   MQMDIR\tools\lib64\mqm.lib
   ```

   在檔案中，新項目必須是一行。
6. 設定下列環境變數:

   ```
   TUXDIR=TUXDIR
   TUXCONFIG=APPDIR\tuxconfig
   FIELDTBLS=MQMDIR\tools\c\samples\amqstxvx.fld
   LANG=C
   ```

7. 建立 TUXEDO 的 TLOG 裝置。 如果要這麼做，請呼叫 tmadmin -c，並輸入指令:

   ```
   crdl -z APPDIR\TLOG
   ```

8. 將現行目錄設為 *APPDIR*，並呼叫範例 make 檔 amqstxmn.mak 作為外部專案 make 檔。 例如，使用 Microsoft Visual C++ ，發出下列指令:

```
msvc amqstxmn.mak
```

選取 **建置**，以建置所有範例程式。

ALW *TUXEDO* 的範例伺服器程式

範例伺服器程式 (amqstxsx) 設計成與 Put (amqstxpx.c) 及 Get (amqstxgx.c) 範例程式一起執行。 當 TUXEDO 啟動時，範例伺服器程式會自動執行。

註: 您必須先啟動佇列管理程式，然後再啟動 TUXEDO。

範例伺服器提供兩個 TUXEDO 服務: MPUT1 及 MGET1:

- MPUT1 服務由 PUT 範例驅動，並在同步點中使用 MQPUT1，將訊息放置在 TUXEDO 所控制的工作單元中。 它會採用 PUT 範例所提供的「完整名稱」和「訊息文字」參數。
- 每次 MGET1 服務取得訊息時，即會開啟並關閉佇列。 它採用 GET 範例所提供的「完整名稱」和「訊息文字」參數。

任何錯誤訊息、原因碼及狀態訊息都會寫入 TUXEDO 日誌檔。

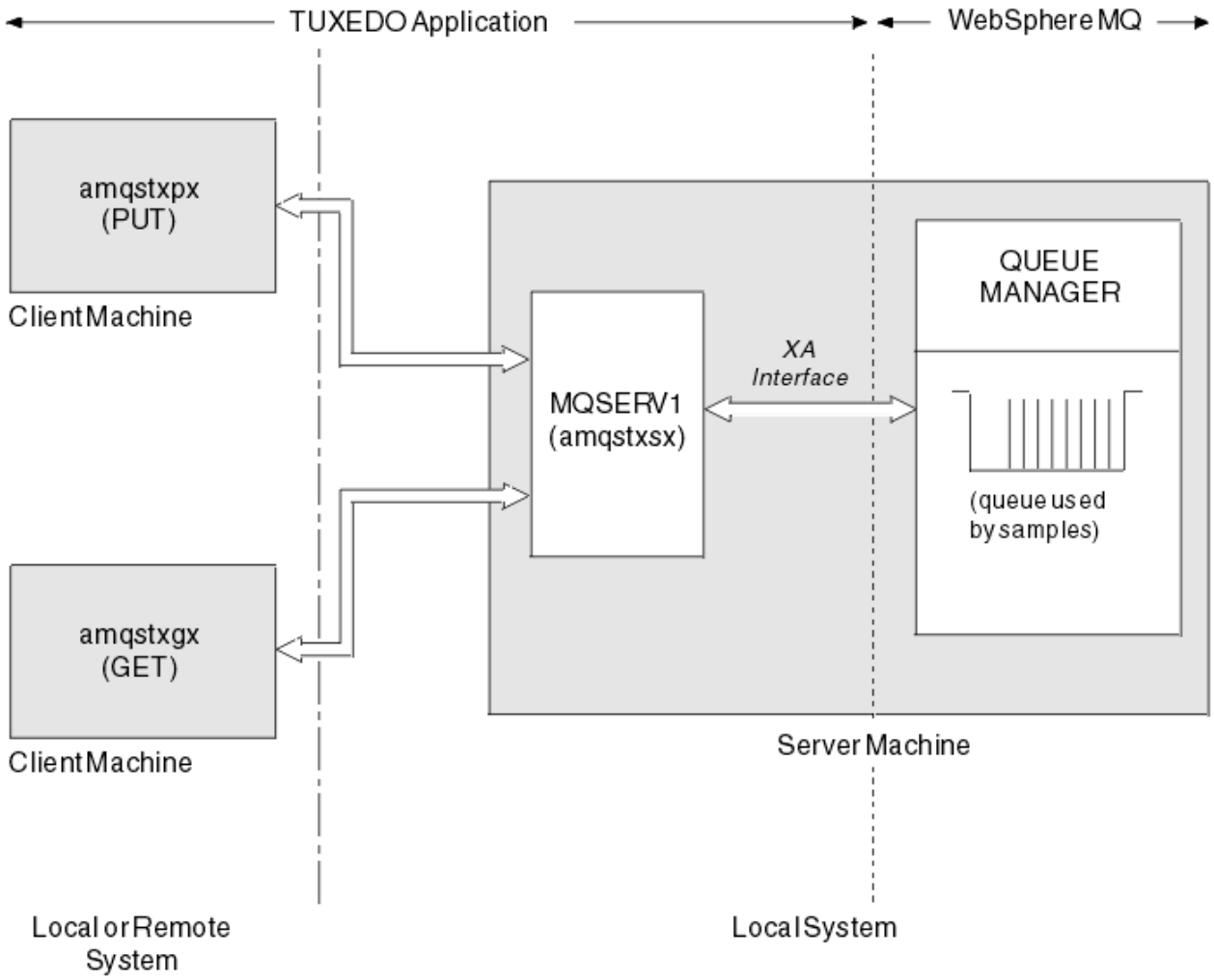


圖 *138: TUXEDO* 範例如何一起運作

ALW 放置 *TUXEDO* 的範例程式

此範例可讓您將訊息以批次方式多次放置在佇列上，並使用 TUXEDO 作為資源管理程式來示範同步點。

範例伺服器程式 `amqstxsx` 必須在執行中，放置範例才能成功; 伺服器範例程式會連接至佇列管理程式，並使用 XA 介面。 若要執行範例，請輸入:

- `doputs -n queuename -b batchsize -c trancount -t message`

例如:

- `doputs -n myqueue -b 5 -c 6 -t "Hello World"`

這會以六個批次方式，將 30 則訊息放入名為 myqueue 的佇列中，每個批次中有五則訊息。 如果有任何問題，它會備份一批訊息，否則它會確定它們。

任何錯誤訊息都會寫入 TUXEDO 日誌檔及 stderr。 任何原因碼都會寫入 stderr。

ALW 取得 *TUXEDO* 的範例

此範例可讓您以批次方式從佇列取得訊息。

範例伺服器程式 amqstxsx 必須在執行中，「取得範例」才能成功; 範例伺服器程式會連接至佇列管理程式，並使用 XA 介面。 若要執行範例，請輸入下列指令:

- `dogets -n queuename -b batchsize -c trancount`

例如:

- `dogets -n myqueue -b 6 -c 4`

這會以六個批次從名為 myqueue 的佇列中移除 24 則訊息，每個批次中有四則訊息。 如果您在放置範例之後執行此動作，在 myqueue 上放置 30 則訊息，則在 myqueue 上只會有 6 則訊息。 在放置訊息和取得訊息之間，批次數目和批次大小可能有所不同。

任何錯誤訊息都會寫入 TUXEDO 日誌檔及 stderr。 任何原因碼都會寫入 stderr。

Windows 在 *Windows* 上使用 *SSPI* 安全結束程式

本主題說明如何在 Windows 系統上使用 SSPI 通道結束程式。 提供的結束碼有兩種格式: 物件和來源。

物件碼

物件程式檔稱為 amqrspin.dll。 對於用戶端和伺服器，它會作為 IBM MQ for Windows 的標準組件安裝在 *MQ_INSTALLATION_PATH*\exits*INSTALLATION_NAME* 資料夾中。 例如， C:\Program Files\IBM\MQ\exits\installation2。 它會載入為標準使用者結束程式。 您可以執行提供的安全通道結束程式，並在通道定義中使用鑑別服務。

若要執行此動作，請指定下列其中一項:

```
SCYEXIT('amqrspin(SCY_KERBEROS)')
SCYEXIT('amqrspin(SCY_NTLM)')
```

若要提供受限通道的支援，請在 SVRCONN 通道上指定下列:

```
SCYDATA('remote_principal_name')
```

其中 *remote_principal_name* 的格式是 *DOMAIN**user*。 只有在遠端主體名稱符合 *remote_principal_name* 時，才會建立安全通道。

如果要在 Kerberos 安全網域內運作的系統之間使用所提供的通道結束程式，請為佇列管理程式建立 **servicePrincipalName** 。

原始碼

結束程式原始碼檔案稱為 amqsspin.c。 它位於 C:\Program Files\IBM\MQ\Tools\c\Samples 中。

如果您修改原始碼，則必須重新編譯已修改的原始碼。

編譯及鏈結的方式與相關平台的任何其他通道結束程式相同，但需要在編譯時期存取 SSPI 標頭，且需要在鏈結時存取 SSPI 安全程式庫以及任何建議的關聯程式庫。

在執行下列指令之前，請確定您的路徑中有 cl.exe、Visual C++ 程式庫及 include 資料夾。 例如：

```
cl /VERBOSE /LD /MT /Ipath_to_Microsoft_platform_SDK\include
/Ipath_to_IBM_MQ\tools\c\include amqsspin.c /DSECURITY_WIN32
-link /DLL /EXPORT:SCY_KERBEROS /EXPORT:SCY_NTLM STACK:8192
```

註: 原始碼不包含任何用於追蹤或錯誤處理的佈建。 如果您修改並使用原始碼，請新增您自己的追蹤及錯誤處理常式。

使用遠端佇列執行範例

您可以在連接的佇列管理程式上執行範例，以示範遠端佇列作業。

程式 amqscos0.tst 提供遠端佇列 (SYSTEM.SAMPLE.REMOTE) 使用名為 OTHER 的遠端佇列管理程式。 若要使用此範例定義，請將 OTHER 變更為您要使用的第二個佇列管理程式的名稱。 您也必須設定兩個佇列管理程式之間的訊息通道; 如需如何執行此動作的相關資訊，請參閱 定義通道。

要求範例程式會將自己的本端佇列管理程式名稱放置在它們所傳送訊息的 *ReplyToQMgr* 欄位中。 「查詢及設定」範例會將回覆訊息傳送至它們所處理之要求訊息的 *ReplyToQ* 及 *ReplyToQMgr* 欄位中指定的佇列及訊息佇列管理程式。

「叢集佇列監視」範例程式 *(AMQSCLM)*

此範例使用內建 IBM MQ 叢集工作量平衡特性，將訊息導向已連接消費端應用程式的佇列實例。 此自動方向可防止在未連接消費端應用程式的叢集佇列實例上建立訊息。

概觀

您可以針對不同佇列管理程式上的相同佇列，設定具有多個定義的叢集。 此配置提供增加可用性及工作量平衡的好處。 不過， IBM MQ 中沒有任何功能可根據所連接應用程式的狀態來動態修改跨叢集的訊息配送。 因此，消費端應用程式必須一律連接至佇列的每一個實例，以確保處理訊息。

叢集佇列監視範例程式會監視所連接應用程式的狀態。 程式會動態調整內建工作量平衡配置，以將訊息導向已連接消費端應用程式的叢集佇列實例。 在某些狀況下，此程式可用來放鬆消費端應用程式一律連接至佇列的每一個實例的需求。 它也會重新傳送在佇列實例上已排入佇列且未連接任何消費端應用程式的訊息。 重新傳送訊息可讓訊息在暫時關閉的消費端應用程式周圍遞送。

此程式設計成在消費端應用程式是長時間執行的應用程式時使用，而不是經常連接及分離應用程式。

叢集佇列監視範例程式是 C 範例檔 amqsclma.c 的已編譯可執行程式。

如需叢集及工作量的進一步相關資訊，請參閱 使用叢集進行工作量管理

AMQSCLM: 使用範例的設計與規劃

叢集佇列監視範例程式如何運作的相關資訊，在設定系統以執行範例程式時要考量的點，以及可以對範例原始碼進行的修改。

設計

叢集佇列監視範例程式會監視已連接消費端應用程式的本端叢集佇列。 程式會監視使用者指定的佇列。 佇列名稱可能是特定 (例如 APP.TEST01) 或一般。 同屬名稱必須採用符合 PCF (可程式化指令格式) 的格式。 同屬名稱的範例有 APP.TEST*或 APP*。

叢集中擁有要監視之本端佇列實例的每一個佇列管理程式，都需要連接叢集佇列監視範例程式的實例。

動態訊息遞送

叢集佇列監視範例程式會使用佇列的 **IPPROCS** (開啟以取得輸入處理程序計數) 值，來判斷該佇列是否有任何消費者。 大於 0 的值表示佇列已連接至少一個消費端應用程式。 這類佇列處於作用中。 值 0 表示佇列沒有附加的耗用程式。 這類佇列處於非作用中狀態。

對於叢集中具有多個實例的叢集佇列， IBM MQ 會使用每一個佇列實例的叢集工作量優先順序內容 **CLWLPRTY** 來決定要將訊息傳送至哪些實例。 IBM MQ 會將訊息傳送至具有最高 **CLWLPRTY** 值的佇列可用實例。

叢集佇列監視範例程式會將本端 **CLWLPRTY** 值設為 1，以啟動叢集佇列。 程式會將其 **CLWLPRTY** 值設為 0，以取消啟動叢集佇列。

IBM MQ 叢集技術會將叢集佇列的已更新 **CLWLPRTY** 內容傳播至叢集中的所有相關佇列管理程式。 例如，

- 具有已連接應用程式的佇列管理程式，可將訊息放入佇列。
- 在相同叢集中擁有同名本端佇列的佇列管理程式。

傳送是使用叢集的完整儲存庫佇列管理程式來完成。 叢集佇列的新訊息會導向至叢集內具有最高 **CLWLPRTY** 值的實例。

已排入佇列的訊息傳送

動態修改 **CLWLPRTY** 的值會影響新訊息的遞送。 此動態修改不會影響已在佇列實例上排入佇列且沒有連接消費者的訊息，或在已修改的 **CLWLPRTY** 值在整個叢集中傳送之前已透過工作量平衡機制的訊息。 因此，訊息會保留在任何非作用中佇列上，且不會由消費端應用程式處理。 為了解決此問題，叢集佇列監視範例程式能夠從本端佇列取得沒有消費者的訊息，並將這些訊息傳送至連接消費者之相同佇列的遠端實例。

叢集佇列監視範例程式透過取得訊息，將訊息從非作用中本端佇列傳送至一或多個作用中遠端佇列 (使用 **MQGET**) 及放置訊息 (使用 **MQPUT**) 至相同的叢集佇列。 此傳送會導致 IBM MQ 叢集工作量管理根據高於本端佇列實例的 CLWLPRTY 值來選取不同的目標實例。 在訊息傳送期間會保留訊息持續性和環境定義。 不會保留訊息順序及任何連結選項。

規劃

當耗用應用程式的連線功能發生變更時，叢集佇列監視範例程式會修改叢集配置。 修改會從叢集佇列監視範例程式正在監視佇列的佇列管理程式，傳輸至叢集中的完整儲存庫佇列管理程式。 完整儲存庫佇列管理程式會處理配置更新項目，並將它們重新傳送至叢集中所有相關的佇列管理程式。 相關佇列管理程式包括擁有同名叢集佇列 (叢集佇列監視範例程式實例執行所在的位置) 的那些佇列管理程式，以及應用程式在過去 30 天內開啟叢集佇列以向其放置訊息的任何佇列管理程式。

跨叢集非同步處理變更。 因此，在每次變更之後，叢集中的不同佇列管理程式在一段時間內可能會有不同的配置視圖。

叢集佇列監視範例程式僅適用於耗用應用程式不常連接或分離的系統; 例如，長時間執行耗用應用程式。 當用來監視只在短時間內連接消費端應用程式的系統時，在配送配置更新項目時所產生的延遲可能會導致叢集中的佇列管理程式對連接消費者的佇列具有不正確的視圖。 此延遲可能會導致不正確遞送訊息。

監視許多佇列時，所有佇列中所連接消費者的變更率相對較低，可能會增加叢集中的叢集配置資料流量。 增加叢集配置資料流量可能會導致下列一或多個佇列管理程式的負載過多。

- 叢集佇列監視範例程式執行所在的佇列管理程式
- 完整儲存庫佇列管理程式
- 佇列管理程式，具有將訊息放入佇列的已連接應用程式
- 在相同叢集中擁有同名本端佇列的佇列管理程式

必須評量完整儲存庫佇列管理程式上的處理器用量。 在完整儲存庫佇列 SYSTEM.CLUSTER.COMMAND.QUEUE。 如果訊息建置在該佇列上，則指出完整儲存庫佇列管理程式無法跟上系統中叢集配置變更的速率。

當叢集佇列監視範例程式監視許多佇列時，範例程式及佇列管理程式會執行大量工作。 即使附加的消費者沒有任何變更，也會執行這項工作。 可以修改 **-i** 引數，以降低監視週期的頻率，以減少本端系統上範例程式的處理器用量。

為了協助偵測過多活動，叢集佇列監視範例程式會報告每個輪詢間隔的平均處理時間、經歷處理時間及配置變更數。 報告會以參考訊息 (**CLM0045I**)、每 30 分鐘或每 600 個輪詢間隔 (以較早的時間為準) 來遞送。

叢集佇列監視使用情形需求

叢集佇列監視範例程式具有需求及限制。 您可以修改所提供的範例原始碼，以變更其使用方式中的部分限制。 本節中列出的範例詳細說明可以進行的修改。

- 叢集佇列監視範例程式設計用來監視已連接或未連接消費端應用程式的佇列。 如果系統具有經常連接及分離的耗用應用程式，則範例程式可能會在整個叢集中產生過多叢集配置活動。 這可能會影響叢集中佇列管理程式的效能。
- 叢集佇列監視範例程式取決於基礎 IBM MQ 系統及叢集技術。 受監視佇列數目、監視頻率及每一個佇列狀態的變更頻率會影響整體系統上的負載。 選取要監視的佇列及監視的輪詢間隔時，必須考量這些因素。
- 叢集佇列監視範例程式的實例必須連接至叢集中擁有要監視之佇列實例的每個佇列管理程式。 不需要將範例程式連接至叢集中未擁有佇列的佇列管理程式。
- 必須以適當的授權來執行叢集佇列監視範例程式，才能存取所有必要的 IBM MQ 資源。 例如，
  - 要連接的佇列管理程式
  - SYSTEM.ADMIN.COMMAND.QUEUE
  - 執行訊息傳送時要監視的所有佇列
- 對於已連接叢集佇列監視範例程式的每一個佇列管理程式，指令伺服器必須在執行中。
- 叢集佇列監視範例程式的每一個實例都需要專用它所連接之佇列管理程式上的本端 (非叢集) 佇列。 此本端佇列用來控制範例程式，並從對佇列管理程式的指令伺服器所進行的查詢接收回覆訊息。
- 要由叢集佇列監視範例程式的單一實例監視的所有佇列必須位於相同的叢集中。 如果佇列管理程式在多個叢集中具有需要監視的佇列，則需要範例程式的多個實例。 每一個實例都需要控制及回覆訊息的本端佇列。
- 所有要監視的佇列必須位於單一叢集中。 不會監視配置為使用叢集名單的佇列。
- 啟用從非作用中佇列傳送訊息是選用的。 它適用於叢集佇列監視範例程式實例所監視的所有佇列。 如果只有受監視佇列的子集需要啟用訊息傳送，則需要兩個叢集佇列監視範例程式實例。 一個範例程式已啟用訊息傳送，另一個範例程式已停用訊息傳送。 範例程式的每一個實例都需要控制及回覆訊息的本端佇列。
- 依預設， IBM MQ 叢集工作量平衡會將訊息傳送至位於放置應用程式所連接之相同佇列管理程式上的叢集佇列實例。 在下列情況下，當本端佇列處於非作用中狀態時，必須停用此選項:
  - 將應用程式連接至擁有受監視非作用中佇列實例的佇列管理程式
  - 正在將佇列訊息從非作用中佇列傳送至作用中佇列。

  透過將 CLWLUSEQ 值設為 ANY，可以靜態停用佇列上的本端工作量平衡喜好設定。 在此配置中，即使有本端耗用應用程式，也會將放置在本端佇列上的訊息配送至本端及遠端佇列實例，以平衡工作量。 或者，可以配置叢集佇列監視範例程式，以在佇列沒有附加消費者時，將 **CLWLUSEQ** 值暫時設為 ANY ，這會導致在該佇列處於作用中狀態時，只將本端訊息移至佇列的本端實例。
- IBM MQ 系統及應用程式不得針對要監視的佇列或正在使用的通道使用 **CLWLPRTY** 。 否則，叢集佇列監視範例程式在 **CLWLPRTY** 佇列屬性上的動作可能會有不想要的效果。
- 叢集佇列監視範例程式會將執行時期資訊記載至一組報告檔。 需要儲存這些報告的目錄，且叢集佇列監視範例程式必須具有寫入它的權限。

AMQSCLM: 準備並執行範例

叢集佇列監視範例可以在本端連接至佇列管理程式，或作為透過通道連接的用戶端來執行。 每當佇列管理程式執行時，範例就應該執行，當在本端執行時，它可以配置成佇列管理程式服務，以自動啟動和停止含有佇列管理程式的範例。

開始之前

在執行叢集佇列監視範例之前，必須先完成下列步驟。

1. 在每一個佇列管理程式上建立工作佇列，供範例內部使用。

   範例的每一個實例都需要本端非叢集佇列以供專用內部使用。 您可以選擇佇列的名稱。 範例使用名稱 AMQSCLM.CONTROL.QUEUE。 例如，在 Windows 上，您可以使用下列 **MQSC** 指令來建立此佇列:

   ```
   DEFINE QLOCAL(AMQSCLM.CONTROL.QUEUE)
   ```

   您可以保留 **MAXDEPTH** 和 **MAXMSGL** 的值作為預設值。
2. 建立錯誤及參考訊息日誌的目錄。

此範例會將診斷訊息寫入報告檔。 您必須選擇儲存檔案的目錄。 例如，在 Windows 上，您可以使用下列指令來建立目錄:

```
mkdir C:\AMQSCLM\rpts
```

範例所建立的報告檔具有下列命名慣例:

```
 QmgrName.ClusterName.RPT0n.LOG
```

3. (選用) 將叢集佇列監視範例定義為 IBM MQ 服務。

   若要監視佇列，範例必須一律在執行中。 若要確保叢集佇列監視範例一律在執行中，您可以將範例定義為佇列管理程式服務。 將範例定義為服務表示在佇列管理程式啟動時啟動 AMQSCLM。 您可以使用下列範例，將叢集佇列監視範例定義為 IBM MQ 服務。

```
define service(AMQSCLM) +
      descr('Active Cluster Queue Message Distribution Monitor - AMQSCLM') +
      control(qmgr) +
      servtype(server) +
      startcmd('MQ_INSTALLATION_PATH\tools\c\samples\Bin\AMQSCLM.exe') +
      startarg('-m +QMNAME+ -c CLUSTER1 -q ABC* -r AMQSCLM.CONTROL.QUEUE -l
c:\AMQSCLM\rpts') +
      stdout('C:\AMQSCLM\rpts\+QMNAME+.TSTCLUS.stdout.log') +
      stderr('C:\AMQSCLM\rpts\+QMNAME+.TSTCLUS.stderr.log')
```

確立	說明
service	指定服務名稱。 您可以選擇服務名稱。
descr	指定服務的文字說明。
control	指出服務與佇列管理程式同時啟動和停止。
servtype	指出這個佇列管理程式一次只能執行一個實例的伺服器服務物件。
startcmd	指定程式的位置和名稱。
startarg	指定範例的引數。 請注意使用 *+ QMNAME +*。 佇列管理程式的名稱會自動替換。
stdout	標準輸出重新導向至的完整檔名。 範例只會將確認範例已終止的訊息寫入此檔案。 範例會這樣做，因為標準錯誤檔案已在範例終止程序的較早階段中關閉。
stderr	將標準錯誤輸出重新導向至其中的完整檔名。 範例會在終止範例之前，將任何錯誤訊息寫入標準錯誤檔案。

關於這項作業

此作業可讓您以不同方式啟動及停止叢集佇列監視範例。 它還可讓您以產生報告檔案的模式執行範例，報告檔案包含所監視佇列的相關統計資訊。

可以使用下列指令來執行範例程式。

```
AMQSCLM -m QMgrName -c ClusterName (-q QNameMask| -f QListFile) -r MonitorQName
[-l ReportDir] [-t] [-u ActiveVal] [-i Interval] [-d] [-s] [-v]
```

此表格列出可與叢集佇列監視範例搭配使用的引數，以及每一個引數的其他相關資訊。

引數	變數	進一步資訊
-m	**QMgrName**	要監視的佇列管理程式。
-c	**ClusterName**	包含要監視之佇列的叢集。
-q	**QNameMask**	要監視的一或多個佇列。 尾端 * 會監視名稱符合零或多個尾端字元的所有佇列。
-f	**QListFile**	包含要監視之佇列名稱或佇列名稱遮罩清單的檔案完整路徑及檔名。 檔案必須每行包含一個佇列名稱/遮罩。 您可以指定 **-q** 或 **-f**，但不能同時指定兩者。
-r	**MonitorQName**	樣本專用的本端佇列。
-l	**ReportDir**	將記載的參考訊息儲存在一組折返中的目錄路徑 [9] 報告檔案。
-t		(選用) 啟用將佇列訊息從非作用中本端佇列傳送至作用中佇列。 如果未啟用，則只有進入叢集的新訊息會動態遞送至佇列的作用中實例。
-u	**ActiveVal**	(選用) 當受監視佇列實例處於非作用中狀態時，自動將其 **CLWLUSEQ** 內容切換至 ANY，當處於作用中狀態時，自動切換至 **ActiveVal** 的值。 **ActiveVal** 可以是 LOCAL 或 QMGR。 如果在放置應用程式連接至相同佇列管理程式或已啟用訊息傳送的系統中未設定此引數，則受監視佇列必須具有 **CLWLUSEQ** 值 ANY 或 QMGR，且佇列管理程式具有值 ANY。
-i	**Interval**	(選用) 監視器檢查佇列的時間間隔 (以秒為單位)。 預設值為 300 秒 (5 分鐘)。
-d		(選用) 啟用其他診斷輸出。 起始配置系統或使用範例程式碼時，除錯輸出可能很有用。
-s		(選用) 啟用每個間隔的最小統計輸出。
-v		(選用) 除了報告檔案之外，還將報告資訊記載至 `standard out`。

引數清單範例:

```
-m QMGR1 -c CLUS1   -f c:\QList.txt -r CLMQ -l c:\amqsclm\rpts -s
-m QMGR2 -c CLUS1   -q ABC* -r CLMQ -l c:\amqsclm\rpts -i 600
-m QMGR1 -c CLUSDEV -q QUEUE.* -r CLMQ -l c:\amqsclm\rpts -t -u QMGR -d
```

佇列清單檔範例:

```
Q1
QUEUE.*
ABC
ABD
```

程序

1. 啟動叢集佇列監視範例。 您可以使用下列其中一種方式來啟動範例:
   - 使用具有適當使用者授權的命令提示字元。
   - 如果範例配置為 IBM MQ 服務，請使用 MQSC **START SERVICE** 指令。

   在這兩種情況下，引數清單都相同。

   在起始設定程式之後 10 秒，範例不會開始監視佇列。 此延遲可讓耗用應用程式先連接至受監視佇列，以防止對佇列作用中狀態進行不必要的變更。
2. 停止叢集佇列監視範例。 當佇列管理程式停止、停止、靜止或與佇列管理程式的連線中斷時，範例會自動停止。 有一些方法可以在不結束佇列管理程式的情況下停止範例:

[9] 對於每一個佇列管理程式及佇列組合，會產生一個固定大小的日誌檔，當已滿時，會改寫該日誌檔。 日誌程式一律寫入至相同的檔案，並保留檔案的兩個舊版。

- 配置範例專用的本端佇列，以停用 Get 功能。
- 將 **CorrelId** 為 "STOP CLUSTER MONITOR\0\0\0\0"的訊息傳送至範例專用的本端佇列。
- 終止取樣程序。 這可能會導致遺失傳送至作用中佇列的非持續訊息。 它也可能導致樣本所使用的本端佇列在終止之後保持開啟數秒。 此狀況會防止叢集佇列監視範例的新實例立即啟動。

如果範例已作為 IBM MQ 服務啟動，則 **STOP SERVICE** 沒有作用。 您可以在佇列管理程式中使用其中一個終止方法，說明為已配置的 **STOP SERVICE** 機制。

下一步

請檢查範例的狀態。

如果已啟用報告，您可以檢閱報告檔的狀態。 使用下列指令來檢閱最新的報告檔案:

```
QMgrName.ClusterName.RPT01.LOG
```

若要檢閱較舊的報告檔案，請使用下列指令:

```
QMgrName.ClusterName.RPT02.LOG
QMgrName.ClusterName.RPT03.LOG
```

報告檔會成長至大約 1 MB 的大小上限。 當 RPT01 檔案填滿時，會建立新的 RPT01 檔案。 舊的 RPT01 檔案會重新命名為 RPT02。 RPT02 已重新命名為 RPT03。 會捨棄舊的 RPT03 。

此範例會在下列狀況中建立參考訊息:

- 啟動時
- 終止時
- 當它標示佇列 **ACTIVE** 或 **INACTIVE** 時
- 當它將訊息從非作用中佇列重新排入作用中實例時

此範例會建立錯誤訊息 *CLMnnnnE* ，以報告需要注意的問題。

每 30 分鐘，樣本會報告每個輪詢間隔的平均處理時間，以及經歷處理時間。 此資訊保留在訊息 CLM0045I 中。

當啟用 **-s** 統計訊息時，範例會報告每一個佇列檢查的下列統計資訊:

- 處理佇列所花費的時間 (毫秒)
- 已檢查的佇列數
- 已進行的作用中/非作用中變更數
- 傳送的訊息數

此資訊在訊息 CLM0048I 中報告。

報告檔可能會在除錯模式中快速成長，並快速覆蓋。 在此狀況下，可能會超出個別檔案的 1 MB 大小限制。

AMQSCLM: 疑難排解
下列各節包含使用範例時可能遇到的實務範例相關資訊。 提供實務範例潛在說明的相關資訊，以及如何解決它的選項。

實務範例 :AMQSCLM 未啟動

可能的說明: 語法不正確。

動作: 請檢查標準錯誤輸出，以取得正確語法

潛在說明: 無法使用佇列管理程式。

動作: 請檢查報告檔中的訊息 ID CLM0010E。

潛在說明: 無法開啟或建立一或多個報告檔。

動作: 請在起始設定期間檢查標準錯誤輸出中的錯誤訊息。

實務範例 :AMQSCLM 未將佇列變更為 ACTIVE 或 INACTIVE

潛在說明: 佇列不在要監視的佇列清單中

動作: 請檢查 **-q** 和 **-f** 參數值。

潛在說明: 佇列不是正確叢集中的本端佇列。

動作: 請檢查佇列是否在本端且在正確的叢集中。

潛在說明: AMQSCLM 未針對此佇列管理程式及叢集執行。

動作: 啟動相關佇列管理程式及叢集的 AMQSCLM。

潛在說明: 佇列保留非作用中， **CLWLPRTY** = 0 ，因為它沒有消費者。 或者，它保留為 ACTIVE **CLWLPRTY** >=1 ，因為它至少有 1 個消費者。

動作: 檢查消費端應用程式是否連接至佇列。

潛在說明: 佇列管理程式的指令伺服器不在執行中。

動作: 請檢查報告檔是否有錯誤。

實務範例: 未在 INACTIVE 佇列周圍遞送訊息

潛在說明: 訊息會直接放置到擁有非作用中佇列的佇列管理程式，且佇列的 **CLWLUSEQ** 值不是 ANY，且不會將 **-u** 引數用於 AMQSCLM。

動作: 請檢查相關佇列管理程式的 **CLWLUSEQ** 值，或確定 **-u** 引數用於 AMQSCLM。

潛在說明: 任何佇列管理程式都沒有作用中的佇列。 訊息會在所有非作用中佇列之間平均平衡工作量，直到佇列變成作用中為止。

動作: 請檢查所有佇列管理程式上的佇列狀態。

潛在說明: 將訊息放置在叢集中與擁有非作用中佇列的不同佇列管理程式之間，且更新的 **CLWLPRTY** 值 0 不會延伸到放置應用程式的佇列管理程式。

動作: 請檢查受監視佇列管理程式與完整儲存庫佇列管理程式之間的叢集通道是否在執行中。 請檢查放置佇列管理程式與完整儲存庫佇列管理程式之間的通道是否在執行中。 請檢查受監視、放置及完整儲存庫佇列管理程式的錯誤日誌。

潛在說明: 遠端佇列實例在作用中 (CLWLPRTY=1) ，但由於來自本端佇列管理程式的叢集傳送端通道不在執行中，因此無法將訊息遞送至這些佇列實例。

動作: 使用佇列的作用中實例，檢查從本端佇列管理程式至遠端佇列管理程式的叢集傳送端通道狀態。

實務範例 :AMQSCLM 未從非作用中佇列傳送訊息

潛在說明: 未啟用訊息傳送 (**-t**)。

動作: 請確定已啟用訊息傳送 (**-t**)。

潛在說明: 佇列不在要監視的佇列清單中。

動作: 請檢查 **-q** 和 **-f** 參數值。

潛在說明: 對於擁有相同佇列實例的這個或叢集中其他佇列管理程式， AMQSCLM 未執行。

動作: 啟動 AMQSCLM。

潛在說明: 佇列具有 **CLWLUSEQ** = LOCAL 或 **CLWLUSEQ** = QMGR，且未設定 **-u** 引數。

動作: 請設定 **-u** 參數，或將佇列或佇列管理程式配置變更為 ANY。

潛在說明: 叢集中沒有佇列的作用中實例。

動作: 請檢查 **CLWLPRTY** 值為 1 或以上的佇列實例。

潛在說明: 遠端佇列實例具有消費者 (**IPPROCS** > = 1) ，但在那些佇列管理程式 (**CLWLPRTY** = 0) 上處於非作用中狀態，因為 AMQSCLM 未在監視那些遠端實例。

動作: 請檢查 **-q** 和 **-f** 參數值，確定 AMQSCLM 正在那些佇列管理程式上執行，且/或佇列位於要監視的佇列清單中。

潛在說明: 遠端佇列實例在作用中 (**CLWLPRTY** = 1) ，但在本端佇列管理程式上被視為非作用中 (**CLWLPRTY** = 0)。 此狀況是因為未將更新的 **CLWLPRTY** 值傳送至此佇列管理程式。

動作: 請確定遠端佇列管理程式至少連接至叢集中的其中一個完整儲存庫佇列管理程式。 請確定完整儲存庫佇列管理程式正常運作。 請檢查完整儲存庫佇列管理程式與受監視佇列管理程式之間的通道是否在執行中。

潛在說明: 訊息未確定，因此無法擷取。

動作: 請檢查傳送端應用程式是否正常運作。

潛在說明: AMQSCLM 無法存取將訊息排入佇列的本端佇列。

動作: 檢查 AMQSCLM 是否以具有足夠存取佇列權限的使用者身分執行。

潛在說明: 佇列管理程式的指令伺服器不在執行中。

動作: 啟動佇列管理程式的指令伺服器。

潛在說明: AMQSCLM 發現錯誤。

動作: 請檢查報告檔是否有錯誤。

潛在說明: 遠端佇列實例處於作用中 (CLWLPRTY=1) ，但由於來自本端佇列管理程式的叢集傳送端通道不在執行中，因此無法將訊息傳送至那些佇列實例。 這通常伴隨著 amqsclm 報告日誌中的 CLM0030W 警告。

動作: 使用佇列的作用中實例，檢查從本端佇列管理程式至遠端佇列管理程式的叢集傳送端通道狀態。

ALW 連線端點查閱 *(CEPL)* 的程式範例

IBM MQ 「連線端點查閱」範例提供簡單但功能強大的結束程式模組，可讓 IBM MQ 使用者從 LDAP 儲存庫 (例如 Tivoli Directory Server) 擷取連線定義。

Tivoli Directory Server v6.3 必須安裝用戶端才能使用 CEPL。

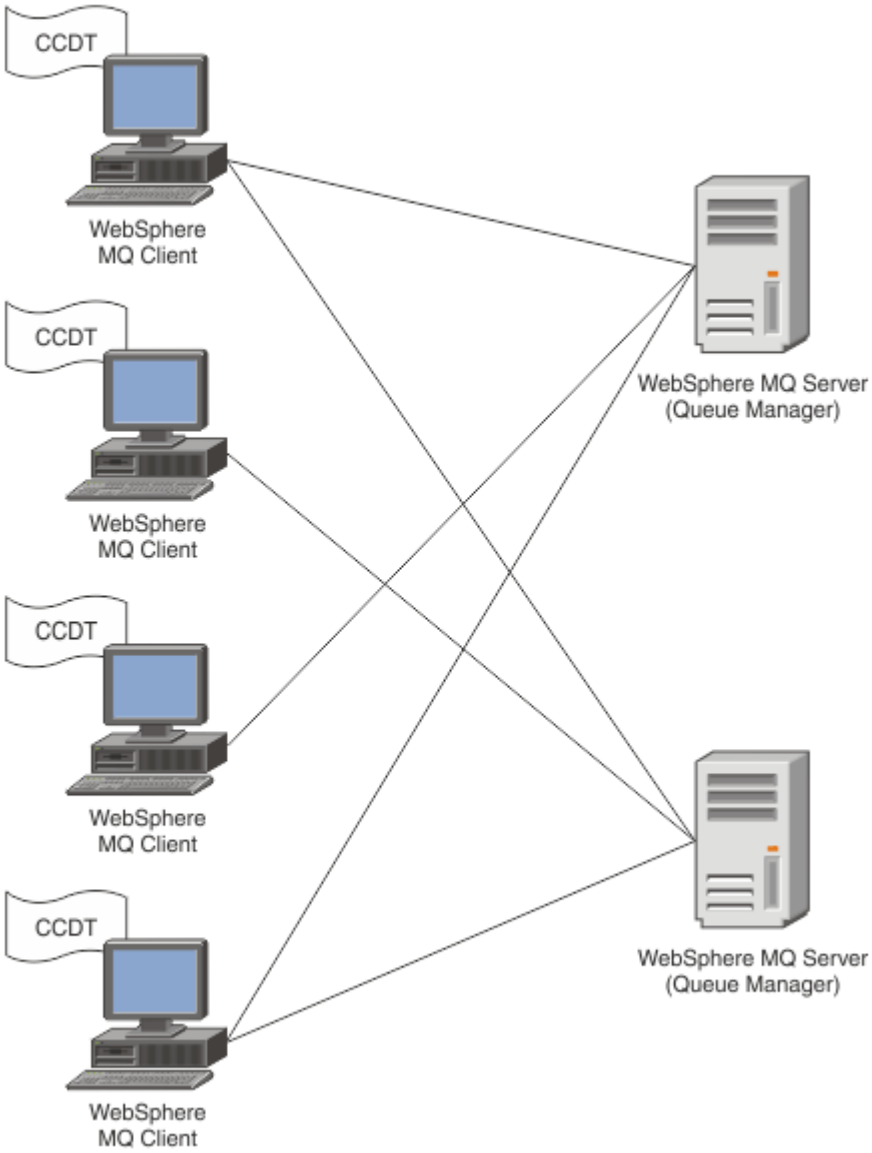
需要具備受支援平台上的 IBM MQ 管理運作知識，才能使用此範例。

Windows Linux AIX 簡介

配置廣域儲存庫 (例如 LDAP (輕量型目錄存取通訊協定) 目錄) ，以儲存用戶端連線定義來協助維護及管理。

使用 IBM MQ 用戶端應用程式，透過「用戶端連線定義表 (CCDT)」建立與「佇列管理程式」的連線。

CCDT 是透過標準「 IBM MQ MQSC 管理」介面來建立。 使用者必須連接至「佇列管理程式」才能建立用戶端連線定義，即使定義內包含的資料不限於「佇列管理程式」。 產生的 CCDT 檔案必須在用戶端機器和應用

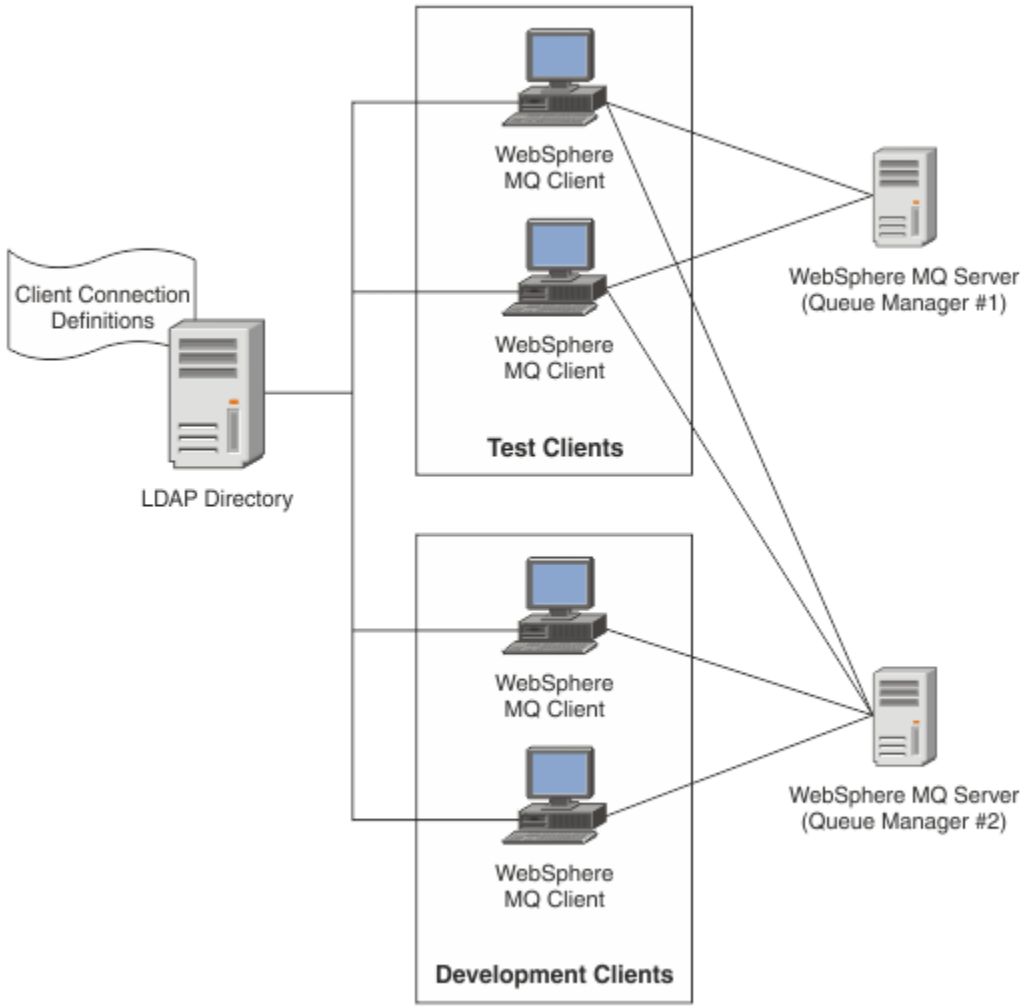


程式之間手動配送。

CCDT 檔案必須配送至每一個 IBM MQ 用戶端。 當數以千計的客戶可以在當地或全球各地存在時，就會很快變得難以維護和管理。 需要更靈活的方法來協助確保每一個用戶端都有正確的用戶端定義可供它們使用。

其中一種方法是將用戶端連線定義儲存在廣域儲存庫中，例如 LDAP (輕量型目錄存取通訊協定) 目錄。 LDAP 目錄也可以提供額外的安全、檢索及搜尋機能，因此每一個用戶端只能存取與它們相關的那些連線定義。

LDAP 目錄可以配置成只有特定的定義可供特定使用者群組使用。 例如，「測試用戶端」可以同時存取「佇列管理程式」 #1 和 #2，而「開發用戶端」只能存取「佇列管理程式」 #2 。



結束程式模組可以查閱 LDAP 儲存庫，例如 IBM Tivoli Directory Server，以擷取通道定義。 使用那些連線定義， IBM MQ 用戶端應用程式可以建立與佇列管理程式的連線。

結束程式模組是預先連接結束程式模組，可讓您在從 LDAP 儲存庫進行 MQCONN/MQCONNX 呼叫期間取得通道定義。

結束程式模組和綱目可能由下列實作:

- 已使用現有 CCDT 檔案型技術建置技能基礎且想要降低管理及配送成本的客戶。
- 已採用自己專屬技術來配送用戶端連線定義的現有客戶。
- 目前未採用任何類型用戶端連線解決方案，且想要使用 IBM MQ 所提供特性的新客戶或現有客戶。
- 想要直接使用或調整其傳訊模型與任何現行 LDAP 商業架構行內的新客戶或現有客戶。

ALW 支援的環境

在執行「連線端點查閱」範例之前，請驗證您具有受支援的作業系統及相關軟體。

IBM MQ Connection Endpoint Lookup 的範例程式需要下列軟體:

- IBM WebSphere MQ 7.0 或更新版本
- Tivoli Directory Server V6.3 用戶端或更新版本

支援的作業系統:

1. Windows Windows (7/8/2008/2012)

2. AIX
3. Linux
   - System p 上的 RHEL v4 和 v5
   - System p 上的 SUSE v9 及 v10
   - RHEL v4 及 v5 x86-64 32 位元及 64 位元
   - SUSE v9 及 v10 x86-64 32 位元及 64 位元

註: 此範例不適用於下列平台:

- z/OS
- IBM i

ALW 安裝與配置

安裝及配置結束模組和連線端點綱目。

安裝結束程式模組

在安裝 IBM MQ 期間，結束程式模組安裝在 `tools/samples/c/preconnexit/bin` 下。 若為 32 位元平台，必須將結束程式模組複製到 `exit/installation_name/` ，才能使用它。 若為 64 位元平台，必須將結束程式模組複製到 exit64/*installation_name*/才能使用它。

正在安裝連線端點綱目

結束程式會使用「連線端點」綱目 `ibm-amq.schema`。 綱目檔必須先匯入至任何 LDAP 伺服器，才能使用結束程式。 匯入綱目之後，必須新增屬性的值。

以下是匯入「連線端點」綱目的範例。 此範例假設正在使用 IBM Tivoli Directory Server (ITDS)。

- 確定 IBM Tivoli Directory Server 正在執行中，然後將 `ibm-amq.schema` 檔案複製或 FTP 至 ITDS 伺服器。
- 在 ITDS 伺服器上，輸入下列指令以將綱目安裝至 ITDS 儲存庫，其中 *LDAP ID* 和 *LDAP* 密碼 是 LDAP 伺服器的根 DN 和密碼:

  `ldapadd -D "LDAP ID" -w "LDAP password" -f ibm-amq.schema`
- 在指令視窗中，輸入下列指令或使用協力廠商工具來瀏覽綱目以進行驗證:

```
ldapsearch objectclass=ibm-amqClientConnection
```

如需匯入綱目檔的進一步詳細資料，請參閱 LDAP 伺服器說明文件。

配置

必須將稱為 PreConnect 的新區段新增至用戶端配置檔，例如 `mqclient.ini`。 PreConnect 區段包含下列關鍵字:

模組
包含 API 結束碼的模組名稱。 如果此欄位包含模組的完整路徑，則會依現狀使用它。 否則會搜尋 IBM MQ 安裝架構中的 `exit` 或 `exit64` 資料夾。

函數
進入包含 `LdapPreConnect` 結束碼之媒體庫的功能進入點名稱。 函數定義遵循您企業的函數原型。

小心: 當您指定實際的結束進入點時，應該移除函數陳述式中的引號。

資料
包含通道定義之 LDAP 儲存庫的 URI。

下列 Snippet 是 mqclient.ini 檔中所需的變更範例。

```
PreConnect:
Module=amqlcelp
Function="LdapPreconnectExit"
Data=ldap:dap://myLDAPServer.com:389/cn=wmq,ou=ibm,ou=com
Sequence=1
```

ALW 結束程式和綱目概觀

語法，以及用來建立佇列管理程式連線的參數。

IBM MQ 9.3 定義結束模組中進入點的下列語法。

```
void MQENTRY MQ_PRECONNECT_EXIT ( PMQNXP pExitParms
                                , PMQCHAR pQMgrName
                                , PPMQCNO ppConnectOpts
                                , PMQLONG pCompCode
                                , PMQLONG pReason)
```

在 MQCONN/X 呼叫執行期間， IBM MQ C Client 會載入包含函數語法實作的結束程式模組。 然後，它會呼叫結束函數來擷取通道定義。 然後會使用擷取的通道定義來建立與佇列管理程式的連線。

參數

pExit 參數

類型 :PMQNXP 輸入/輸出

PreConnection 結束程式參數結構。 該結構由結束程式的呼叫者配置及維護。

```
struct tagMQNXP
{
  MQCHAR4    StrucId;             /* Structure identifier */
  MQLONG     Version;             /* Structure version number */
  MQLONG     ExitId;              /* Type of exit */
  MQLONG     ExitReason;          /* Reason for invoking exit */
  MQLONG     ExitResponse;        /* Response from exit */
  MQLONG     ExitResponse2;       /* Secondary response from exit */
  MQLONG     Feedback;            /* Feedback code (reserved) */
  MQLONG     ExitDataLength;      /* Exit data length */
  PMQCHAR    pExitDataPtr;        /* Exit data */
  MQPTR      pExitUserAreaPtr;    /* Exit user area */
  PMQCD *    ppMQCDArrayPtr;      /* Array of pointers to MQCDs */
  MQLONG     MQCDArrayCount;      /* Number of entries found */
  MQLONG     MaxMQCDVersion;      /* Maximum MQCD version */
};
```

pQMgr 名稱

類型 :PMQCHAR 輸入/輸出

佇列管理程式的名稱。 在輸入上，此參數是透過 **QMgrName** 參數提供給 MQCONN API 呼叫的過濾器字串。 此欄位可能是空白、明確或包含某些萬用字元。 結束程式會變更欄位。 使用 MQXR_TERM 呼叫結束程式時，此參數為 NULL 。

ppConnect 選項

類型: ppConnectOpts input/output

控制 MQCONNX 動作的選項。 這是 MQCNO 連線選項結構的指標，可控制 MQCONN API 呼叫的動作。 使用 MQXR_TERM 呼叫結束程式時，此參數為 NULL 。 MQI 用戶端一律提供 MQCNO 結構給結束程式，即使它最初不是由應用程式所提供。 如果應用程式提供 MQCNO 結構，用戶端會複製它，將它傳遞至修改它的結束程式。 用戶端保留 MQCNO 的所有權。 透過 MQCNO 參照的 MQCD 優先於透過陣列提供的任何連線定義。 用戶端會使用 MQCNO 結構來連接佇列管理程式，並忽略其他佇列管理程式。

pComp 程式碼

類型 :PMQLONG 輸入/輸出

完成碼。 指向接收結束程式完成碼之 MQLONG 的指標。 它必須是下列其中一個值:

- MQCC_OK -順利完成

- MQCC_WARNING -警告 (局部完成)
- MQCC_FAILED -呼叫失敗

pReason
類型 :PMQLONG 輸入/輸出

符合 pComp 代碼的原因。 指向接收結束原因碼之 MQLONG 的指標。 如果完成碼為 MQCC_OK ， 則唯一有效值為: MQRC_NONE -(0 ， x '000') 無報告原因。

如果完成碼是 MQCC_FAILED 或 MQCC_WARNING ， 則結束函數可以將原因碼欄位設為任何有效的 MQRC_ * 值。

ALW *MQ LDAP* 環境定義資訊
結束程式使用下列資料結構來取得環境定義資訊。

MQNLDAPCTX
MQNLDAPCTX 結構具有下列 C 原型。

```
typedef struct tagMQNLDAPCTX MQNLDAPCTX;
typedef MQNLDAPCTX MQPOINTER PMQNLDAPCTX;

struct tagMQNLDAPCTX
{
  MQCHAR4      StrucId;          /* Structure identifier */
  MQLONG       Version;          /* Structure version number */
  LDAP *       objectDirectory   /* LDAP Instance */
  MQLONG       ldapVersion;      /* Which LDAP version to use? */
  MQLONG       port;             /* Port number for LDAP server*/
  MQLONG       sizeLimit;        /* Size limit */
  MQBOOL       ssl;              /* SSL enabled? */
  MQCHAR *     host;             /* Hostname of LDAP server */
  MQCHAR *     password;         /* Password of LDAP server */
  MQCHAR *     searchFilter;     /* LDAP search filter */
  MQCHAR *     baseDN;           /* Base Distinguished Name */
  MQCHAR *     charSet;          /* Character set */
};
```

Windows Linux AIX 建置連線端點查閱結束程式的範例程式碼
您可以使用範例程式碼 Snippet ， 在 AIX、 Linux 或 Windows 上編譯原始檔。

正在編譯原始檔

您可以使用任何 LDAP 用戶端程式庫來編譯來源，例如 IBM Tivoli Directory Server V6.3 用戶端程式庫。 本文件假設您使用 Tivoli Directory Server V6.3 用戶端程式庫。

註: 下列 LDAP 伺服器支援預先連接結束程式庫:

- IBM Tivoli Directory Server V6.3
- Novell eDirectory V8.2

下列程式碼 Snippet 說明如何編譯結束程式:

Windows **在 Windows 平台上編譯結束程式**
您可以使用下列 Snippet 來編譯結束程式檔:

```
CC=cl.exe
LL=link.exe
CCARGS=/c /I. /DWIN32 /W3 /DNDEBUG /EHsc /D_CRT_SECURE_NO_DEPRECATE /Zl

# The libraries to include
LDLIBS=Ws2_32.lib Advapi32.lib libibmldapstatic.lib libibmldapdbgstatic.lib \
kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib \
shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib msvcrt.lib

OBJS=amqlcel0.obj

all: amqlcelp.dll
```

```
amqlcelp.dll: $(OBJS)
        $(LL) /OUT:amqlcelp.dll /INCREMENTAL /NOLOGO /DLL /SUBSYSTEM:WINDOWS /MACHINE: X86 \
              /DEF:amqlcelp.def $(OBJS) $(LDLIBS) /NODEFAULTLIB:msvcrt.lib

# The exit source
amqlcel0.obj: amqlcel0.c
$(CC) $(CCARGS) $*.c
```

註: 如果您使用使用 Microsoft Visual Studio 2003 編譯器所編譯的 IBM Tivoli Directory Server V6.3 用戶端程式庫，當您使用 Microsoft Visual Studio 2012 或更新版本的編譯器來編譯 IBM Tivoli Directory Server V6.3 用戶端程式庫時，可能會收到警告。

Linux AIX **在 AIX、 Linux 上編譯結束程式**

下列程式碼 Snippet 用於在 Linux 上編譯結束程式檔。 部分編譯器選項在 AIX 上可能不同。

```
#Make file to build exit
CC=gcc

MQML=/opt/mqm/lib
MQMI=/opt/mqm/inc
TDSI=/opt/ibm/ldap/V6.3/include
XFLAG=-m32

TDSL=/opt/ibm/ldap/V6.3/lib
```

IBM Tivoli Directory Server 隨附靜態和動態鏈結程式庫，但您只能使用一種類型的程式庫。 此 Script 假設您使用靜態程式庫。

```
#Use static libraries.
LDLIBS=-L$(TDSL) -libibmldapstatic

CFLAGS=-I. -I$(MQMI) -I$(TDSI)

all:amqlcepl

amqlcepl: amqlcel0.c
$(CC) -o cepl amqlcel0.c -shared -fPIC $(XFLAG) $(CFLAGS) $(LDLIBS)
```

ALW 呼叫 *PreConnect* 結束程式模組

可以使用三個不同的原因碼來呼叫 PreConnect 結束程式模組: 用於起始設定及建立 LDAP 伺服器連線的 MQXR_INIT 原因碼、用於從 LDAP 伺服器擷取通道定義的 MQXR_PRECONNECT 原因碼，或要清除結束程式時的 MQXR_TERM 原因碼。

MQXR_INIT

使用 MQXR_INIT 原因碼來呼叫結束程式，以起始設定及建立與 LDAP 伺服器的連線。

在 MQXR_INIT 呼叫之前， MQNXP 結構的 pExitDataPtr 欄位會移入 `mqclient.ini` 檔內 PreConnect 段落中的 Data 屬性 (即 LDAP)。

LDAP URL 至少包含用於搜尋的通訊協定、主機名稱、埠號及基本 DN。 結束程式會剖析 pExitDataPtr 欄位內包含的 LDAP URL，配置 MQNLDAPCTX LDAP 查閱環境定義結構並相應地移入它。 此結構的位址儲存在 pExitUserAreaPtr 欄位中。 無法正確剖析 LDAP URL 會導致錯誤 MQCC_FAILED。

此時，結束程式會使用 **MQNLDAPCTX** 參數連接並連結至 LDAP 伺服器。 產生的 LDAP API 控點也會儲存在此結構內。

MQXR_PRECONNECT

使用 MQXR_PRECONNECT 原因碼來呼叫結束模組，以從 LDAP 伺服器擷取通道定義。

結束程式會在 LDAP 伺服器中搜尋符合給定過濾器的通道定義。 如果 **QMgrNameparameter** 包含特定的佇列管理程式名稱，則搜尋會傳回 **ibm-amqQueueManagerName** LDAP 屬性值符合給定佇列管理程式名稱的所有通道定義。

如果 **QMgrName** 參數是 '*' 或 ' ' (空白)，則搜尋會傳回 **ibm-amqIsClientDefault Connection** 端點屬性設為 TRUE 的所有通道定義。

成功搜尋之後，結束程式會準備一個或一個 MQCD 定義陣列，並傳回給呼叫程式。

MQXR_TERM
當要清除結束程式時，會以這個原因碼來呼叫結束程式。 在此清除期間，結束程式會中斷與 LDAP 伺服器的連線，並釋放結束程式所配置及維護的所有記憶體，包括 MQNLDAPCTX 結構、指標陣列及其參照的每一個 MQCD。 任何其他欄位都會設為預設值。 在具有 MQXR_TERM 原因碼的結束程式期間，**pQMgrName** 及 **ppConnectOpts** 結束程式參數未使用，且可能是 NULL。

相關參考
用戶端配置檔的 PreConnect 段落

ALW *LDAP* 綱目
用戶端連線資料儲存在稱為 LDAP (輕量型目錄存取通訊協定) 目錄的廣域儲存庫中。 IBM MQ 用戶端使用 LDAP 目錄來取得連線定義。 LDAP 目錄中 IBM MQ 用戶端連線定義的結構稱為 LDAP 綱目。 LDAP 綱目是屬性類型定義、物件類別定義及其他資訊的集合，供伺服器用來判斷過濾器或屬性值主張是否符合項目的屬性，以及是否允許、新增及修改作業。

將資料儲存在 LDAP 目錄中

用戶端連線定義位於目錄樹狀結構內稱為連線點的特定分支下。 如同 LDAP 目錄中的所有其他節點，連線點也有相關聯的「識別名稱 (DN)」。 您可以使用此節點作為對目錄進行任何查詢的起點。 在查詢 LDAP 目錄以傳回用戶端連線定義子集時使用過濾。 您可以根據目錄樹狀結構其他部分中所授與的許可權 (例如，使用者、部門或群組) 來限制子樹狀結構的存取權。

定義您自己的屬性和類別
修改 LDAP 綱目來儲存用戶端通道定義。 所有 LDAP 資料定義都需要物件及屬性。 物件及屬性由物件 ID (OID) 號碼識別，該號碼可唯一識別物件或屬性。 LDAP 綱目內的所有類別直接或間接從頂端物件繼承。 用戶端通道定義物件包含最上層物件的屬性。 所有 LDAP 資料定義都需要物件及屬性:

- 物件定義是 LDAP 屬性的集合。
- 屬性是 LDAP 資料類型。

LDAP 屬性中說明每一個屬性的說明，以及它們如何對映至一般 IBM MQ 內容。

ALW *LDAP* 屬性
定義的 LDAP 屬性是 IBM MQ 特有的，並直接對映至用戶端連線內容。

IBM MQ 用戶端通道目錄字串屬性
下表列出字串屬性及其與 IBM MQ 內容的對映。 這些屬性可以保留 directoryString (UTF-8 編碼 Unicode，亦即包含 IA5/ASCII 作為子集的可變位元組編碼系統) 語法的值。 語法由其物件識別碼 (OID) 指定。

表 *166: IBM MQ* 用戶端通道目錄字串屬性

LDAP 屬性	**說明**	**IBM MQ 內容**
CN	由通道名稱和定義佇列管理程式名稱組成的一般名稱。	
ibm-amqChannel 名稱	通道定義的名稱。	CHANNEL
ibm-amqConnection 名稱	通訊連線 ID。	CONNAME
ibm-amqDescription	通道說明。	DESCR
ibm-amqLocal 位址	通道的本端通訊位址。	LOCLADDR
ibm-amqMode 名稱	LU 6.2 模式名稱。	MODENAME
ibm-amqPassword	可以使用的密碼。	PASSWORD
ibm-amqQueueManagerName	IBM MQ 用戶端應用程式可以要求連線的佇列管理程式或佇列管理程式群組名稱。	QMNAME
ibm-amqSecurityExitUser 資料	傳遞至安全結束程式的使用者資料。	SCYDATA

表 166: IBM MQ 用戶端通道目錄字串屬性 (繼續)		
LDAP 屬性	**說明**	**IBM MQ 內容**
ibm-amqSecurityExitName	通道安全結束程式要執行的結束程式名稱。	SCYEXIT
ibm-amqSslCipherSpec	TLS 連線的單一 CipherSpec 。	SSLCIPH
ibm-amqSslPeerName	檢查來自 IBM MQ 通道另一端同層級佇列管理程式或用戶端的憑證識別名稱 (DN)。	SSLPEER
ibm-amqTransactionProgramName	交易程式名稱。	TPNAME
ibm-amqUserID	嘗試使用遠端 MCA 來起始安全 SNA 階段作業時，MCA 要使用的使用者 ID。	USERID

IBM MQ 用戶端連線整數屬性

具有預定值的屬性 (例如，列舉類型) 會儲存為標準整數。 這些值會以整數值儲存在 LDAP 目錄中，而不是使用相關聯的常數名稱。

表 167: IBM MQ 用戶端通道目錄整數屬性		
LDAP 屬性	**說明**	**IBM MQ 內容**
ibm-amqConnection 親緣性	決定透過相同佇列管理程式名稱多次連接的用戶端應用程式是否使用相同的用戶端通道。	AFFINITY
ibm-amqClientChannelWeight	影響使用哪個用戶端連線通道定義的加權。	CLNTWGHT
ibm-amqHeartBeatInterval	傳輸佇列中沒有訊息時，要從傳送端 MCA 傳遞之活動訊號流間的大約時間。	HBINT
ibm-amqKeepAliveInterval	通道的逾時值。	KAINT
ibm-amqMaximumMessageLength	可以在通道上傳輸的訊息長度上限。	MAXMSGL
ibm-amqSharing 交談	共用每一個 TCP/IP 通道實例的交談數上限。	SHARECNV
ibm-amqTransport 類型	要使用的傳輸類型。	TRPTYPE

IBM MQ 用戶端通道布林屬性

此布林屬性未對映至任何 IBM MQ 內容。 此屬性的語法指出布林值。

表 168: IBM MQ 用戶端通道布林屬性	
LDAP 屬性	**說明**
ibm-amqIsClientDefault	定義這個布林屬性是為了解決搜尋尚未定義 ibm-amqQueueManagerName 屬性的項目問題。

IBM MQ 用戶端通道清單屬性

IBM MQ 內容在 LDAP 目錄內儲存為單一值、以逗點區隔的清單屬性。 屬性的定義方式與其他目錄字串屬性相同。 下表說明清單屬性及其與 IBM MQ 內容的對映。

表 169: IBM MQ 用戶端通道清單屬性		
LDAP 屬性	**說明**	**IBM MQ 內容**
ibm-amqHeader 壓縮	通道支援的標頭資料壓縮技術清單。	COMPHDR
ibm-amqMessage 壓縮	通道支援的訊息資料壓縮技術清單。	COMPMSG
ibm-amqSendExitUser 資料	傳遞至傳送結束程式的使用者資料。	SENDDATA

表 169: IBM MQ 用戶端通道清單屬性 (繼續)		
LDAP 屬性	**說明**	**IBM MQ 內容**
ibm-amqSendExitUser 名稱	通道傳送結束程式要執行的結束程式名稱。	SENDEXIT
ibm-amqReceiveExitUser 資料	傳遞至接收結束程式的使用者資料。	RCVDATA
ibm-amqReceiveExitName	通道接收使用者結束程式要執行的使用者結束程式名稱。	RCVEXIT

ALW 通用名稱
通用名稱 (CN) 由通道名稱及定義佇列管理程式名稱組成。

它是預先存在的屬性。

CN 的格式為:

```
CN=CHANNEL_NAME(DEFINING_Q_MGR_NAME)
```

例如:

```
CN=TC1(QM_T1)
```

您只能對此屬性指定一個值。

此屬性是字串屬性，值不區分大小寫。 會忽略子字串比對。 子字串比對是子綱目中使用的比對規則，它使用子字串 (例如 CN=jim * ，其中 CN 是屬性) 來指定搜尋過濾器中屬性的行為，並包含一個以上萬用字元。

ALW *ibm-amqChannel* 名稱
此屬性指定通道定義的名稱。

此屬性具有單一字串值，最多 20 個不區分大小寫的字元。 它不是預先存在的屬性。

會忽略子字串比對。 子字串比對是在子綱目中使用的比對規則，它使用子字串並包含一個以上萬用字元來指定搜尋過濾器中屬性的行為。

ALW *ibm-amqDescription*
此 LDAP 屬性提供通道說明。

此屬性具有最多 64 個位元組的單一字串值，不區分大小寫。 它不是預先存在的屬性。

會忽略子字串比對。 子字串比對是子綱目中使用的比對規則，指定搜尋過濾器中屬性的行為。

ALW *ibm-amqConnection* 名稱
此 LDAP 屬性是通訊連線 ID。 它指定此通道要使用的特定通訊鏈結。

此屬性具有單一字串值，最多 264 個字元，不區分大小寫。 它不是預先存在的屬性。

會忽略子字串比對。 子字串比對是子綱目中使用的比對規則，指定搜尋過濾器中屬性的行為。

ALW *ibm-amqLocal* 位址
此屬性指定通道的本端通訊位址。

此屬性具有最多 48 個字元的單一字串值，不區分大小寫。 它不是預先存在的屬性。

會忽略子字串比對。 子字串比對是子綱目中使用的比對規則，指定搜尋過濾器中屬性的行為。

ALW *ibm-amqMode* 名稱
此屬性用於 LU 6.2 連線。 在執行通訊階段作業配置時，提供額外的連線階段作業性質定義。

此屬性具有正好 8 個字元的單一字串值，不區分大小寫。 它不是預先存在的屬性。

會忽略子字串比對。 子字串比對是子綱目中使用的比對規則，指定搜尋過濾器中屬性的行為。

ALW *ibm-amqPassword*
此 LDAP 屬性指定在嘗試起始與遠端 MCA 的安全 LU 6.2 階段作業時， MCA 可以使用的密碼。
此屬性具有單一整數值，最多 12 位數。 它不是預先存在的屬性。

ALW *ibm-amqQueueManagerName*
此屬性指定 IBM MQ 用戶端應用程式可以要求連線的佇列管理程式或佇列管理程式群組名稱。
此屬性具有最多 48 個字元的單一字串值，不區分大小寫。 它不是預先存在的屬性。
會忽略子字串比對。 子字串比對是子綱目中使用的比對規則，指定搜尋過濾器中屬性的行為。

相關參考
第 973 頁的『ibm-amqIsClientDefault』
此布林屬性可解決搜尋尚未定義 ibm-amqQueueManagerName 屬性的項目問題。

ALW *ibm-amqSecurityExitUser* 資料
此 LDAP 屬性指定傳遞至安全結束程式的使用者資料。
此屬性具有最多 999 個字元的單一字串值，不區分大小寫。 它不是預先存在的屬性。
會忽略子字串比對。 子字串比對是子綱目中使用的比對規則，指定搜尋過濾器中屬性的行為。

ALW *ibm-amqSecurityExitName*
這個 LDAP 屬性指定通道安全結束程式要執行的結束程式名稱。
如果沒有有效的通道安全結束程式，請保留空白。
此屬性具有最多 999 個字元的單一字串值，不區分大小寫。 此屬性不是預先結束的屬性。
會忽略子字串比對。 子字串比對是子綱目中使用的比對規則，指定搜尋過濾器中屬性的行為。

ALW *ibm-amqSslCipherSpec*
此 LDAP 屬性指定 TLS 連線的單一 CipherSpec 。
此屬性具有最多 32 個字元的單一字串值，不區分大小寫。 它不是預先存在的屬性。
會忽略子字串比對。 子字串比對是子綱目中使用的比對規則，指定搜尋過濾器中屬性的行為。

ALW *ibm-amqSslPeerName*
這個 LDAP 屬性用來檢查來自 IBM MQ 通道另一端同層級佇列管理程式或用戶端的憑證識別名稱 (DN)。
此 LDAP 屬性具有單一字串值，最多 1024 個位元組，不區分大小寫。 它不是預先存在的。
會忽略子字串比對。 子字串比對是子綱目中使用的比對規則，指定搜尋過濾器中屬性的行為。

ALW *ibm-amqTransactionProgramName*
此 LDAP 屬性指定交易程式名稱。 它與 LU 6.2 連線搭配使用。
此屬性具有最多 64 個字元的單一字串值，不區分大小寫。 它不是預先存在的。
會忽略子字串比對。 子字串比對是子綱目中使用的比對規則，指定搜尋過濾器中屬性的行為。

ALW *ibm-amqUserID*
此 LDAP 屬性指定當嘗試起始與遠端 MCA 的安全 SNA 階段作業時， MCA 要使用的使用者 ID。
此屬性具有正好 12 個字元的單一字串值 (不區分大小寫)。 它不是預先存在的屬性。
會忽略子字串比對。 子字串比對是子綱目中使用的比對規則，指定搜尋過濾器中屬性的行為。

ALW *ibm-amqConnection* 親緣性
這個 LDAP 屬性指定使用相同佇列管理程式名稱多次連接的用戶端應用程式是否使用相同的用戶端通道。

此屬性具有單一整數值。 它不是預先存在的屬性。

ALW *ibm-amqClientChannelWeight*
此 LDAP 屬性指定加權，會影響使用的用戶端連線通道定義。

當有多個適合的定義可用時，使用用戶端通道加權屬性來偏誤用戶端通道定義的選擇。

此屬性具有單一整數值。 它不是預先存在的屬性。

ALW *ibm-amqHeartBeatInterval*
此 LDAP 屬性指定當傳輸佇列上沒有訊息時，要從傳送端 MCA 傳遞的活動訊號流之間的大約時間。

此屬性具有單一整數值。 它不是預先存在的屬性。 預設值為 1。 預設值是在現行 MQSERVER 環境變數作業中設定。

ALW *ibm-amqKeepAliveInterval*
這個 LDAP 屬性用來指定通道的逾時值。

此屬性的值會傳遞至通訊堆疊，並指定通道的保持作用中計時。 您可以使用此選項，為每一個通道指定不同的保留作用中值。

此屬性具有單一整數值。 它不是預先存在的屬性。

ALW *ibm-amqMaximumMessageLength*
此 LDAP 屬性指定可在通道上傳輸的訊息長度上限。

根據現行 MQSERVER 環境變數作業，此屬性的預設值為 104857600。 此屬性具有單一整數值，且不是預先存在的屬性。

ALW *ibm-amqSharing* 交談
此 LDAP 屬性指定共用每一個 TCP/IP 通道實例的交談數上限。

此屬性具有單一整數值。 此屬性不是預先存在的屬性。

ALW *ibm-amqTransport* 類型
這個 LDAP 屬性指定要使用的傳輸類型。

此屬性具有單一整數值。 它不是預先存在的屬性。

ALW *ibm-amqIsClientDefault*
此布林屬性可解決搜尋尚未定義 ibm-amqQueueManagerName 屬性的項目問題。

預先連接結束程式模組通常會以 ibm-amqQueueManagerName 屬性值作為搜尋準則來搜尋 LDAP 伺服器。 這類查詢會傳回 ibm-amqQueueManagerName 屬性值符合 MQCONN/X 呼叫所指定佇列管理程式名稱的所有項目。 不過，當使用用戶端通道定義表 (CCDT) 時，您可以將 MQCONN/X 呼叫上的佇列管理程式名稱設為空白，或在名稱前面加上星號 (*)。 如果佇列管理程式的名稱為空白，則用戶端會連接至預設佇列管理程式。 如果佇列管理程式的名稱字首為星號 (*) ，則用戶端會連接任何佇列管理程式。

同樣地，項目中的 ibm-amqQueueManagerName 屬性可以維持未定義。 在此情況下，預期使用此端點資訊的用戶端可以連接至任何佇列管理程式。 例如，項目包含下列行:

```
ibm-amqChannelName = "CHANNEL1"
ibm-amqConnectionName = myhost(1414)
```

在此範例中，用戶端會嘗試連接至在 `myhost` 上執行的指定佇列管理程式。

不過，在「LDAP 伺服器」中，不會對尚未定義的屬性值進行搜尋。 例如，如果項目包含 ibm-amqQueueManagerName 以外的連線資訊，則搜尋結果不會包含此項目。 若要克服此問題，您可以設定 ibm-amqIsClientDefault。 這是布林屬性，如果未定義，則假設其值為 FALSE。

對於尚未定義 ibm-amqQueueManagerName 且預期成為搜尋的一部分的項目，請將 ibm-amqIsClientDefault 設為 TRUE。 在 MQCONN/X 呼叫中指定空白或星號 (*) 作為佇列管理程式名稱時，預先連接結束程式會在 LDAP 伺服器中搜尋 ibm-amqIsClientDefault 屬性值設為 TRUE 的所有項目。

註: 如果 ibm-amqIsClientDefault 設為 TRUE ，則請勿設定或定義 ibm-amqQueueManagerName 屬性。

相關參考

第 972 頁的『ibm-amqQueueManagerName』
此屬性指定 IBM MQ 用戶端應用程式可以要求連線的佇列管理程式或佇列管理程式群組名稱。

ALW *ibm-amqHeader* 壓縮
此 LDAP 屬性是通道支援的標頭資料壓縮技術清單。

此屬性的大小上限為 48 個字元。 它不是預先存在的屬性。

您只能對此屬性指定一個值。

此清單屬性是以逗點區隔格式指定為目錄字串。 例如，指定給 **ibm-amqHeaderCompression** 的值是 `0` ，對映至 `NONE`。 用戶端會忽略任何超出容許上限的值。 例如， ibm-amqHeader 壓縮在清單中最多包含 2 個整數。

ALW *ibm-amqMessage* 壓縮
此 LDAP 屬性是通道支援的訊息資料壓縮技術清單。

此屬性的大小上限為 48 個字元。 它不是預先存在的屬性。

此屬性不支援多個值。

此清單屬性是以逗點區隔格式指定為目錄字串。 例如，指定給這個屬性的值是 1,2,4 ，它對映至基礎壓縮順序 RLE、ZLIBFAST 和 ZLIBHIGH。

用戶端會忽略任何超出容許上限的值。 例如， ibm-amqMessage 壓縮在清單中最多包含 16 個整數。

ALW *ibm-amqSendExitUser* 資料
此 LDAP 屬性指定傳遞至傳送結束程式的使用者資料。

此 LDAP 屬性具有單一字串值，最多 999 個字元，不區分大小寫。 它不是預先存在的屬性。

會忽略子字串比對。 子字串比對是子綱目中使用的比對規則，指定搜尋過濾器中屬性的行為。

註: ibm-amqSendExitName 和 **ibm-amqSendExitUserData** 需要成對同步化。 使用者資料應該與結束程式名稱同步。 因此，如果指定一個，則另一個也必須對稱指定，即使它不包含任何資料。

ALW *ibm-amqSendExitName*
這個 LDAP 屬性指定通道傳送結束程式要執行的結束程式名稱。

此屬性具有最多 999 個字元的單一字串值，不區分大小寫。 它不是預先存在的屬性。

會忽略子字串比對。 子字串比對是子綱目中使用的比對規則，指定搜尋過濾器中屬性的行為。

註: ibm-amqSendExitName 和 **ibm-amqSendExitUserData** 必須成對同步。 使用者資料必須與結束程式名稱同步。 因此，如果指定其中一個，則即使它不包含任何資料，也必須對稱指定另一個。

ALW *ibm-amqReceiveExitUser* 資料
此 LDAP 屬性指定傳遞至接收結束程式的使用者資料。

您可以執行一連串接收結束程式。 一系列結束程式的使用者資料字串以逗點、空格或兩者區隔。

此屬性具有最多 999 個字元的單一字串值，不區分大小寫。 它不是預先存在的屬性。

會忽略子字串比對。 子字串比對是子綱目中使用的比對規則，指定搜尋過濾器中屬性的行為。

註: ibm-amqReceiveExitName 和 **ibm-amqReceiveExitUserData** 必須成對同步。 使用者資料必須與結束程式名稱同步。 因此，如果指定其中一個，則即使它不包含任何資料，也必須對稱指定另一個。

ALW *ibm-amqReceiveExitName*
這個 LDAP 屬性指定通道接收使用者結束程式所要執行的使用者結束程式名稱。

此屬性是要連續執行的程式名稱清單。 如果沒有有效的通道接收使用者結束程式，請保留空白。

此屬性具有最多 999 個字元的單一字串值，不區分大小寫。 它不是預先存在的屬性。

會忽略子字串比對。 子字串比對是子綱目中使用的比對規則，指定搜尋過濾器中屬性的行為。

註: ibm-amqReceiveExitName 和 **ibm-amqReceiveExitUserData** 必須成對同步。 使用者資料必須與結束程式名稱同步。 因此，如果指定一個，則另一個也必須對稱指定，即使它不包含任何資料也一樣。

z/OS 使用 z/OS 的程式範例

隨 IBM MQ for z/OS 提供的範例程序化應用程式示範「訊息佇列介面 (MQI)」的一般用法。

關於這項作業

IBM MQ for z/OS 也提供範例資料轉換結束程式，如 第 823 頁的『寫入資料-轉換結束程式』中所述。

所有範例應用程式都以來源形式提供; 數個範例應用程式也以可執行形式提供。 來源模組包括說明程式邏輯的虛擬碼。

註: 雖然部分範例應用程式具有基本畫面驅動介面，但它們並不旨在示範如何設計應用程式的外觀與操作方式。 如需如何為不可程式化終端機設計面板驅動介面的相關資訊，請參閱 *SAA Common User Access: Basic Interface Design Guide* (SC26-4583) 及其附錄 (GG22-9508)。 這些提供準則來協助您設計在應用程式內及其他應用程式之間一致的應用程式。

程序

- 請使用下列鏈結，以進一步瞭解範例程式:
  - 第 976 頁的『z/OS 的範例應用程式中示範的特性』
  - 第 982 頁的『在 z/OS 上準備及執行批次環境的範例應用程式』
  - 第 984 頁的『在 z/OS 上為 TSO 環境準備範例應用程式』
  - 第 985 頁的『在 z/OS 上為 CICS 環境準備範例應用程式』
  - 第 988 頁的『在 z/OS 上為 IMS 環境準備範例應用程式』
  - 第 989 頁的『z/OS 上的「放置」範例』
  - 第 991 頁的『z/OS 上的 Get 範例』
  - 第 993 頁的『z/OS 上的瀏覽範例』
  - 第 995 頁的『z/OS 上的「列印訊息」範例』
  - 第 997 頁的『z/OS 上的佇列屬性範例』
  - 第 999 頁的『z/OS 上的 Mail Manager 範例』
  - 第 1005 頁的『z/OS 上的信用檢查範例』
  - 第 1014 頁的『z/OS 上的「訊息處理程式」範例』
  - 第 1017 頁的『z/OS 上的「非同步放置」範例』
  - 第 1018 頁的『z/OS 上的「批次非同步耗用」範例』
  - 第 1020 頁的『z/OS 上的 CICS 非同步耗用和發佈/訂閱範例』
  - 第 1022 頁的『z/OS 上的發佈/訂閱範例』
  - 第 1024 頁的『z/OS 上的 Set and Inquire message 內容範例』

相關工作
第 888 頁的『在 Multiplatforms 上使用範例程式』

這些範例程序程式隨產品一起提供。 範例以 C 和 COBOL 撰寫，並示範「訊息佇列介面 (MQI)」的一般用法。

z/OS *z/OS* 的範例應用程式中示範的特性

本節彙總每一個範例應用程式中示範的 MQI 特性，顯示撰寫每一個範例的程式設計語言，以及每一個範例執行的環境。

z/OS 在 *z/OS* 上放置範例

「放置」範例示範如何使用 MQPUT 呼叫將訊息放置在佇列上。

應用程式會使用下列 MQI 呼叫:

- MQCONN
- MQOPEN
- MQPUT
- MQCLOSE
- MQDISC

程式以 COBOL 和 C 交付，並在批次和 CICS 環境中執行。 請參閱 第 982 頁的表 172，以取得批次應用程式，並參閱 第 986 頁的表 179，以取得 CICS 應用程式。

z/OS 在 *z/OS* 上取得範例

Get 範例示範如何使用 MQGET 呼叫從佇列取得訊息。

應用程式會使用下列 MQI 呼叫:

- MQCONN
- MQOPEN
- MQGET
- MQCLOSE
- MQDISC

程式以 COBOL 和 C 交付，並在批次和 CICS 環境中執行。 請參閱 第 982 頁的表 172，以取得批次應用程式，並參閱 第 986 頁的表 179，以取得 CICS 應用程式。

z/OS 在 *z/OS* 上瀏覽範例

「瀏覽」範例示範如何使用「瀏覽」選項來尋找訊息、列印訊息，然後逐步執行佇列上的訊息。

應用程式會使用下列 MQI 呼叫:

- MQCONN
- MQOPEN
- 用於瀏覽訊息的 MQGET
- MQCLOSE
- MQDISC

程式以 COBOL、組譯器、PL/I 和 C 語言交付。 應用程式在批次環境中執行。 如需批次應用程式，請參閱 第 983 頁的表 173。

z/OS 在 *z/OS* 上列印訊息範例

「列印訊息」範例示範如何從佇列中移除訊息，並列印訊息中的資料及其訊息描述子的所有欄位。 它可以選擇性地顯示與每一個訊息相關聯的所有訊息內容。

透過從原始模組中的兩行移除註解字元，您可以變更程式，讓它瀏覽 (而非移除) 佇列上的訊息。 此程式可用來診斷將訊息放入佇列之應用程式的問題。

應用程式會使用下列 MQI 呼叫:

- MQCONN
- MQOPEN
- MQGET , 用於從佇列中移除訊息 (具有瀏覽選項)
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP

程式以 C 語言交付。 應用程式在批次環境中執行。 如需批次應用程式，請參閱 第 983 頁的表 174 。

z/OS *z/OS* 上的佇列屬性範例
「佇列屬性」範例示範如何查詢及設定 IBM MQ for z/OS 物件屬性的值。

應用程式會使用下列 MQI 呼叫:

- MQOPEN
- MQINQ
- MQSET
- MQCLOSE

程式以 COBOL、組譯器及 C 語言交付。 應用程式在 CICS 環境中執行。 如需 CICS 應用程式，請參閱 第 986 頁的表 180 。

z/OS *z/OS* 上的郵件管理程式範例
使用 Mail Manager 範例時要注意的考量。

「郵件管理程式」範例示範下列技術:

- 使用別名佇列
- 使用模型佇列來建立暫時動態佇列
- 使用回覆目的地佇列
- 在 CICS 和批次環境中使用同步點
- 將指令傳送至系統指令輸入佇列
- 測試回覆碼
- 使用遠端佇列的本端定義，以及將訊息直接放置在遠端佇列管理程式的具名佇列中，將訊息傳送至遠端佇列管理程式

應用程式會使用下列 MQI 呼叫:

- MQCONN
- MQOPEN
- MQPUT1
- MQGET
- MQINQ
- MQCMIT
- MQCLOSE
- MQDISC

提供了三個版本的應用程式:

- 以 COBOL 撰寫的 CICS 應用程式

- 以 COBOL 撰寫的 TSO 應用程式
- 以 C 撰寫的 TSO 應用程式

TSO 應用程式使用 IBM MQ for z/OS 批次配接器，並包含一些 ISPF 畫面。

請參閱 第 984 頁的表 177 (若為 TSO 應用程式)，以及 第 987 頁的表 181 (若為 CICS 應用程式)。

z/OS *z/OS* 上的信用檢查範例

此資訊包含使用「信用檢查」範例時要考量的點。

「信用檢查」範例是一套示範下列技術的程式:

- 開發在多個環境中執行的應用程式
- 使用模型佇列來建立暫時動態佇列
- 使用相關性 ID
- 設定及傳遞環境定義資訊
- 使用訊息優先順序和持續性
- 使用觸發來啟動程式
- 使用回覆目的地佇列
- 使用別名佇列
- 使用無法傳送郵件的佇列
- 使用名單
- 測試回覆碼

應用程式會使用下列 MQI 呼叫:

- MQOPEN
- MQPUT
- MQPUT1
- MQGET，用於瀏覽及取得訊息，使用等待及信號選項，以及用於取得特定訊息
- MQINQ
- MQSET
- MQCLOSE

範例可以作為獨立式 CICS 應用程式執行。 不過，為了示範如何設計訊息佇列作業應用程式，以使用 CICS 及 IMS 環境所提供的機能，也會提供一個模組作為 IMS 批次訊息處理程式。

CICS 程式以 C 和 COBOL 交付。 單一 IMS 程式以 C 提供。

請參閱 第 987 頁的表 182 (若為 CICS 應用程式) 及 第 989 頁的表 184 (若為 IMS 應用程式)。

z/OS *z/OS* 上的「訊息處理程式」範例

「訊息處理程式」範例可讓您瀏覽、轉遞及刪除佇列上的訊息。

應用程式會使用下列 MQI 呼叫:

- MQCONN
- MQOPEN
- MQINQ
- MQPUT1
- MQCMIT
- MQBACK
- MQGET
- MQCLOSE

- MQDISC

程式以 C 和 COBOL 程式設計語言交付。 應用程式在 TSO 下執行。 如需 TSO 應用程式，請參閱第 985 頁的表 178。

z/OS z/OS 上的分散式佇列結束程式範例

分散式佇列作業結束程式範例的來源程式表格。

下表列出分散式佇列結束程式範例的來源程式名稱:

表 170: 分散式佇列結束程式範例的來源			
成員名稱	**針對語言**	**說明**	**在檔案庫中提供**
CSQ4BAX0	Assembler	來源程式	SCSQASMS
CSQ4BCX1	C	來源程式	SCSQC37S
CSQ4BCX2	C	來源程式	SCSQC37S
CSQ4BCX4	C	來源程式	SCSQC37S

註: 使用 CSQXSTUB 來鏈結編輯來源程式。

z/OS z/OS 上的資料轉換結束程式範例

提供了用於資料轉換結束常式的架構，並隨附說明 MQXCNVC 呼叫的範例 IBM MQ 。

下表列出資料轉換結束程式範例的來源程式名稱:

表 171: 資料轉換結束程式範例的來源 (僅限組譯語言)		
成員名稱	**說明**	**在檔案庫中提供**
CSQ4BAX8	來源程式	SCSQASMS
CSQ4BAX9	來源程式	SCSQASMS
CSQ4CAX9	來源程式	SCSQASMS

註: 來源程式是使用 CSQASTUB 進行鏈結編輯。

如需相關資訊，請參閱 第 823 頁的『寫入資料-轉換結束程式』。

z/OS z/OS 上的發佈/訂閱範例

「發佈/訂閱」範例程式示範如何使用 IBM MQ 中的發佈和訂閱特性。

有四個 C 和兩個 COBOL 程式設計語言範例程式示範如何對「 IBM MQ 發佈/訂閱」介面進行程式設計。

應用程式使用下列 MQI 呼叫:

- MQCONN
- MQOPEN
- MQPUT
- MQSUB
- MQGET
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP

「公用/訂閱」範例程式以 C 和 COBOL 程式設計語言交付。 範例應用程式在批次環境中執行。 請參閱批次應用程式的 發佈/訂閱範例 。

z/OS 配置佇列管理程式以接受 *z/OS* 上的用戶端連線

您必須先建立佇列管理程式，才能執行範例應用程式。 然後，您可以配置佇列管理程式，以安全地接受來自以用戶端模式執行之應用程式的送入連線要求。

開始之前

請確定佇列管理程式已存在且已啟動。 透過發出 MQSC 指令，判定是否已啟用通道鑑別記錄:

```
DISPLAY QMGR CHLAUTH
```

重要: 這項作業預期會啟用通道鑑別記錄。 如果這是其他使用者及應用程式所使用的佇列管理程式，則變更此設定將會影響所有其他使用者及應用程式。 如果您的佇列管理程式未使用通道鑑別記錄，則步驟 4 可以取代為替代鑑別方法 (例如安全結束程式) ，它會將 MCAUSER 設定為您將在步驟 第 980 頁的『1』 中取得的 *non-privileged-user-id* 。

您必須知道應用程式預期使用的通道名稱，才能允許應用程式使用該通道。 您也必須知道應用程式預期使用哪些物件 (例如佇列或主題) ，以便您的應用程式可以使用它們。

關於這項作業

此作業會建立非特許使用者 ID ，用於連接至佇列管理程式的用戶端應用程式。 用戶端應用程式只會獲授與存取權，以便能夠使用它需要的通道，以及使用這個使用者 ID 所需要的佇列。

程序

1. 在佇列管理程式執行所在的系統上取得使用者 ID。

   對於此作業，此使用者 ID 不得是特許管理使用者。 此使用者 ID 是將在佇列管理程式上執行用戶端連線的權限。
2. 啟動接聽器程式。

   a) 請確定通道起始程式已啟動。 如果沒有，請發出 **START CHINIT** 指令來啟動它。

   b) 發出下列指令來啟動接聽器程式:

   ```
   START LISTENER TRPTYPE(TCP) PORT(nnnn)
   ```

   其中 *nnnn* 是您選擇的埠號。
3. 如果您的應用程式使用 SYSTEM.DEF.SVRCONN 則已定義此通道。 如果您的應用程式使用另一個通道，請發出 MQSC 指令來建立它:

   ```
   DEFINE CHANNEL(' channel-name ') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
   DESCR('Channel for use by sample programs')
   ```

   channel-name 是通道的名稱。
4. 透過發出 MQSC 指令，建立通道鑑別規則，只容許用戶端系統的 IP 位址使用通道:

   ```
   SET CHLAUTH(' channel-name ') TYPE(ADDRESSMAP) ADDRESS(' client-machine-IP-address ') +
   MCAUSER(' non-privileged-user-id ')
   ```

   其中

   channel-name 是通道的名稱。

   client-machine-IP-address 是用戶端系統的 IP 位址。 如果您的範例用戶端應用程式在與佇列管理程式相同的機器上執行，則如果您的應用程式要使用 'localhost' 來連接，請使用 IP 位址 '127.0.0.1'。 如果要在中連接數個不同的用戶端機器，您可以使用型樣或範圍，而非單一 IP 位址。 如需詳細資料，請參閱 一般 IP 位址 。

non-privileged-user-id 是您在步驟 第 980 頁的『1』 中取得的使用者 ID

5. 如果您的應用程式使用 SYSTEM.DEFAULT.LOCAL.QUEUE，則已定義此佇列。 如果您的應用程式使用另一個佇列，請發出 MQSC 指令來建立它:

```
DEFINE QLOCAL(' queue-name ') DESCR('Queue for use by sample programs')
```

其中 *queue-name* 是佇列的名稱。

6. 授與存取權以連接及查詢佇列管理程式:

a) 請確定通道起始程式已啟動。 如果沒有，請發出 START CHINIT 指令來啟動通道起始程式。

b) 啟動 TCP 接聽器，例如發出下列指令:

```
START LISTENER TRPTYPE(TCP) PORT(nnnn)
```

其中 *nnnn* 是您選擇的埠號。

7. 如果您的應用程式是點對點應用程式 (亦即，它會使用佇列)，請發出 MQSC 指令來授與存取權，以容許依要使用的使用者 ID 使用佇列來查詢及放置及取得訊息:

發出 RACF 指令:

```
RDEFINE MQQUEUE qmgr-name.QUEUE. queue-name UACC(NONE)

PERMIT qmgr-name.QUEUE. queue-name CLASS(MQQUEUE) ID(non-privileged-user-id) ACCESS(UPDATE)
```

其中

qmgr-name 是佇列管理程式的名稱
queue-name 是佇列的名稱。
non-privileged-user-id 是您在步驟 第 980 頁的『1』 中取得的使用者 ID

8. 如果您的應用程式是發佈/訂閱應用程式 (亦即，它會使用主題)，請發出下列 RACF 指令，授與存取權以容許使用使用者 ID 所要使用的主題進行發佈及訂閱:

```
RDEFINE MQTOPIC qmgr-name.PUBLISH.SYSTEM.BASE.TOPIC UACC(NONE)

PERMIT qmgr-name.PUBLISH.SYSTEM.BASE.TOPIC CLASS(MQTOPIC) ID(non-privileged-user-id)
ACCESS(UPDATE)

RDEFINE MQTOPIC qmgr-name.SUBSCRIBE.SYSTEM.BASE.TOPIC UACC(NONE)

PERMIT qmgr-name.SUBSCRIBE.SYSTEM.BASE.TOPIC CLASS(MQTOPIC) ID(non-privileged-user-id)
ACCESS(UPDATE)
```

其中

qmgr-name 是佇列管理程式的名稱
non-privileged-user-id 是您在步驟 第 980 頁的『1』 中取得的使用者 ID
這將授與 非特許使用者 *ID* 對主題樹狀結構中任何主題的存取權，或者，您可以使用 **DEFINE TOPIC** 來定義主題物件，並僅授與存取權給該主題物件所參照的主題樹狀結構部分。 如需相關資訊，請參閱 控制使用者對主題的存取權。

下一步

您的用戶端應用程式現在可以連接至佇列管理程式，並使用佇列來放置或取得訊息。

相關概念

z/OS 在 z/OS 上使用 IBM MQ 物件的權限

相關參考

SET CHLAUTH
定義通道
DEFINE QLOCAL
SET AUTHREC

z/OS 在 *z/OS* 上準備及執行批次環境的範例應用程式

若要準備在批次環境中執行的範例應用程式，請執行與建置任何批次 IBM MQ for z/OS 應用程式時相同的步驟。

這些步驟在 第 859 頁的『建置 z/OS 批次應用程式』 中列出。

或者，在我們提供可執行形式的範例時，您可以從 thlqual.SCSQLOAD 載入程式庫執行它。

註: 「瀏覽」範例的組譯語言版本使用資料控制區塊 (DCB)，因此您必須使用 RMODE(24) 來鏈結編輯它。

要使用的程式庫成員列在 第 982 頁的表 172、 第 983 頁的表 173、 第 983 頁的表 174 和 第 983 頁的表 175 中。

您必須編輯針對您要使用的範例所提供的執行 JCL (請參閱 第 982 頁的表 172、 第 983 頁的表 173、 第 983 頁的表 174 及 第 983 頁的表 175)。

所提供 JCL 中的 PARM 陳述式包含一些您需要修改的參數。 若要執行 C 範例程式，請以空格區隔參數; 若要執行組譯器、COBOL 及 PL/I 範例程式，請以逗點區隔它們。 例如，如果佇列管理程式的名稱是 CSQ1，且您想要在 COBOL、PL/I 及組譯語言 JCL 中使用名為 LOCALQ1 的佇列來執行應用程式，則 PARM 陳述式應該如下所示:

```
PARM=(CSQ1,LOCALQ1)
```

在 C 語言 JCL 中， PARM 陳述式應該如下所示:

```
PARM=('CSQ1 LOCALQ1')
```

您現在已準備好提交工作。

z/OS *z/OS* 上的範例批次應用程式名稱

提供給範例批次應用程式的程式摘要。

下表彙總批次應用程式:

- 第 982 頁的表 172 放置及取得範例
- 第 983 頁的表 173 瀏覽範例
- 第 983 頁的表 174 列印訊息範例
- 第 983 頁的表 175 發佈/訂閱範例
- 第 984 頁的表 176 其他範例

表 *172:* 批次放置和取得範例

成員名稱	針對語言	說明	檔案庫中提供的原始檔	程式庫中提供的執行檔
CSQ4BCJ1	C	取得來源程式	SCSQC37S	SCSQLOAD
CSQ4BCK1	C	放置來源程式	SCSQC37S	SCSQLOAD
CSQ4BCJR	C	CSQ4BCJ1 和 CSQBCK1 的執行 JCL 範例	SCSQPROC	無
CSQ4BVJ1	COBOL	取得來源程式	SCSQCOBS	SCSQLOAD
CSQ4BVK1	COBOL	放置來源程式	SCSQCOBS	SCSQLOAD
CSQ4BVJR	COBOL	CSQBVJ1 和 CSQBVK1 的執行 JCL 範例	SCSQPROC	無

表 *173:* 批次瀏覽範例

成員名稱	針對語言	說明	檔案庫中提供的原始檔	程式庫中提供的執行檔
CSQ4BVA1	COBOL	來源程式	SCSQCOBS	SCSQLOAD
CSQ4BVAR	COBOL	CSQ4BVA1 的執行 JCL 範例	SCSQPROC	無
CSQ4BAA1	Assembler	來源程式	SCSQASMS	SCSQLOAD
CSQ4BAAR	Assembler	CSQ4BAA1 的執行 JCL 範例	SCSQPROC	無
CSQ4BCA1	C	來源程式	SCSQC37S	SCSQLOAD
CSQ4BCAR	C	CSQ4BCA1 的執行 JCL 範例	SCSQPROC	無
CSQ4BPA1	PL/I	來源程式	SCSQPLIS	SCSQLOAD
CSQ4BPAR	PL/I	CSQ4BPA1 的執行 JCL 範例	SCSQPROC	無

表 *174:* 批次列印訊息範例 *(*僅限 *C* 語言*)*

成員名稱	說明	檔案庫中提供的原始檔	程式庫中提供的執行檔
CSQ4BCG1	來源程式	SCSQC37S	SCSQLOAD
CSQ4BCGR	CSQ4BCG1 的執行 JCL 範例	SCSQPROC	無
CSQ4BCL1	瀏覽來源程式	SCSQC37S	SCSQLOAD
CSQ4BCLR	CSQ4BCL1 的執行 JCL 範例	SCSQPROC	無

表 *175:* 發佈/訂閱範例

成員名稱	針對語言	說明	檔案庫中提供的原始檔	SCSQPROC 中的 JCL	程式庫中提供的執行檔
CSQ4BCP1	C	發佈至主題來源程式	SCSQC37S	CSQ4BCPP	SCSQLOAD
CSQ4BCP2	C	訂閱主題並取得訊息來源程式	SCSQC37S	CSQ4BCPS	SCSQLOAD
CSQ4BCP3	C	使用使用者提供的目的地訂閱主題並取得訊息來源程式	SCSQC37S	CSQ4BCPD	SCSQLOAD
CSQ4BCP4	C	使用延伸選項訂閱主題並取得訊息來源程式	SCSQC37S	CSQ4BCPE	SCSQLOAD
CSQ4BVP1	COBOL	發佈至主題來源程式	SCSQCOBS	CSQ4BVPP	SCSQLOAD
CSQ4BVP2	COBOL	訂閱主題並取得訊息來源程式	SCSQCOBS	CSQ4BVPS	SCSQLOAD

表 176: 其他範例					
成員名稱	**針對語言**	**說明**	**檔案庫中提供的原始檔**	**SCSQPROC 中的 JCL**	**程式庫中提供的執行檔**
CSQ4BCS1	C	非同步使用來源程式	SCSQC37S	CSQ4BCSC	SCSQLOAD
CSQ4BCS2	C	非同步放置，並檢查狀態來源程式	SCSQC37S	CSQ4BCSP	SCSQLOAD
CSQ4BCM1	C	查詢訊息內容來源程式	SCSQC37S	CSQ4BCMP	SCSQLOAD
CSQ4BCM2	C	設定訊息內容來源程式	SCSQC37S	CSQ4BCMP	SCSQLOAD

z/OS 在 *z/OS* 上為 *TSO* 環境準備範例應用程式

若要準備在 TSO 環境中執行的範例應用程式，請執行與建置任何批次 IBM MQ for z/OS 應用程式時相同的步驟。

這些步驟在 第 859 頁的『建置 z/OS 批次應用程式』中列出。 要使用的檔案庫成員列在 第 984 頁的表 177 中。

或者，在我們提供可執行形式的範例時，您可以從 thlqual.SCSQLOAD 載入程式庫執行它。

對於「郵件管理程式」範例應用程式，請確保其使用的佇列在您的系統上可用。 它們定義在成員 **thlqual**.SCSQPROC (CSQ4CVD) 中。 若要確保這些佇列一律可用，您可以將這些成員新增至 CSQINP2 起始設定輸入資料集，或使用 CSQUTIL 程式來載入這些佇列定義。

z/OS *z/OS* 上的範例 *TSO* 應用程式名稱

提供給每一個範例 TSO 應用程式的程式名稱，以及原始檔、JCL 及執行檔 (僅適用於「訊息處理程式」範例) 所在之程式庫的相關資訊。

下表彙總 TSO 應用程式:

- 第 984 頁的表 177 郵件管理程式範例
- 第 985 頁的表 178 訊息處理程式範例

這些範例使用 ISPF 畫面。 因此，當您鏈結編輯程式時，必須包含 ISPF Stub ISPLINK。

表 177: *TSO Mail Manager* 範例			
成員名稱	**針對語言**	**說明**	**檔案庫中提供的原始檔**
CSQ4CVD	independent	IBM MQ for z/OS 物件定義	SCSQPROC
CSQ40	independent	ISPF 訊息	SCSQMSGE
CSQ4RVD1	COBOL	用於起始 CSQ4TVD1 的 CLIST	SCSQCLST
CSQ4TVD1	COBOL	功能表程式的來源程式	SCSQCOBS
CSQ4TVD2	COBOL	「取得郵件」程式的來源程式	SCSQCOBS
CSQ4TVD4	COBOL	傳送郵件程式的來源程式	SCSQCOBS
CSQ4TVD5	COBOL	暱稱程式的來源程式	SCSQCOBS
CSQ4VDP1-6	COBOL	畫面定義	SCSQPNLA
CSQ4VD0	COBOL	資料定義	SCSQCOBC
CSQ4VD1	COBOL	資料定義	SCSQCOBC

表 177: TSO Mail Manager 範例 (繼續)			
成員名稱	**針對語言**	**說明**	**檔案庫中提供的原始檔**
CSQ4VD2	COBOL	資料定義	SCSQCOBC
CSQ4VD4	COBOL	資料定義	SCSQCOBC
CSQ4RCD1	C	用於起始 CSQ4TCD1 的 CLIST	SCSQCLST
CSQ4TCD1	C	功能表程式的來源程式	SCSQC37S
CSQ4TCD2	C	「取得郵件」程式的來源程式	SCSQC37S
CSQ4TCD4	C	傳送郵件程式的來源程式	SCSQC37S
CSQ4TCD5	C	暱稱程式的來源程式	SCSQC37S
CSQ4CDP1-6	C	畫面定義	SCSQPNLA
CSQ4TC0	C	併入檔	SCSQC370

表 178: TSO 訊息處理程式範例				
成員名稱	**針對語言**	**說明**	**檔案庫中提供的原始檔**	**程式庫中提供的執行檔**
CSQ4TCH0	C	資料定義	SCSQC370	無
CSQ4TCH1	C	來源程式	SCSQC37S	SCSQLOAD
CSQ4TCH2	C	來源程式	SCSQC37S	SCSQLOAD
CSQ4TCH3	C	來源程式	SCSQC37S	SCSQLOAD
CSQ4RCH1	C 和 COBOL	CLIST 以起始 CSQ4TCH1 或 CSQ4TVH1	SCSQCLST	無
CSQ4CHP1	C 和 COBOL	畫面定義	SCSQPNLA	無
CSQ4CHP2	C 和 COBOL	畫面定義	SCSQPNLA	無
CSQ4CHP3	C 和 COBOL	畫面定義	SCSQPNLA	無
CSQ4CHP9	C 和 COBOL	畫面定義	SCSQPNLA	無
CSQ4TVH0	COBOL	資料定義	SCSQCOBC	無
CSQ4TVH1	COBOL	來源程式	SCSQCOBS	SCSQLOAD
CSQ4TVH2	COBOL	來源程式	SCSQCOBS	SCSQLOAD
CSQ4TVH3	COBOL	來源程式	SCSQCOBS	SCSQLOAD

z/OS 在 *z/OS* 上為 *CICS* 環境準備範例應用程式

在執行 CICS 範例程式之前，請使用 LOGMODE 32702 來登入 CICS 。 這是因為已寫入範例程式來使用 3270 模式 2 畫面。

若要準備在 CICS 環境中執行的範例應用程式，請執行下列步驟:

1. 組合 BMS 畫面定義來源 (在程式庫 **thlqual**.SCSQMAPS 中提供，其中 **thlqual** 是安裝所使用的高階限定元) ，以建立範例的符號說明對映和實體畫面對映。 當您命名對映時，請使用 BMS 畫面定義來源的名稱 (不適用於「放置」及「取得」範例程式) ，但省略該名稱的最後一個字元。

2. 執行與建置任何 CICS IBM MQ for z/OS 應用程式時相同的步驟。 這些步驟在 第 862 頁的『在 z/OS 中建置 CICS 應用程式』中列出。 要使用的程式庫成員列在 第 986 頁的表 179、 第 986 頁的表 180、 第 987 頁的表 181 和 第 987 頁的表 182 中。

   或者，在我們提供可執行形式的範例時，您可以從 thlqual.SCSQCICS 載入程式庫執行它。

3. 透過更新 CICS 系統定義 (CSD) 資料集，識別 CICS 的對映集、程式及交易。 您需要的定義位於成員 **thlqual**.SCSQPROC (CSQ4S100) 中。 如需如何執行此動作的指引，請參閱 CICS Transaction Server for z/OS 4.1 產品說明文件中的 *CICS-IBM MQ* 配接器 一節，網址為: CICS Transaction Server for z/OS 4.1, CICS-IBM MQ 配接器。

   註: 對於 Credit Check 範例應用程式，如果您尚未建立範例使用的 VSAM 資料集，則會在此階段收到錯誤訊息。

4. 對於 Credit Check 和 Mail Manager 範例應用程式，請確定它們使用的佇列在您的系統上可用。 對於 Credit Check 範例，它們定義在成員 **thlqual**.SCSQPROC (CSQ4CVB) for COBOL 中，以及 **thlqual**.SCSQPROC (CSQ4CCB) for C 中。 對於「郵件管理程式」範例，它們定義在成員 **thlqual**.SCSQPROC (CSQ4CVD) 中。 若要確保這些佇列一律可用，您可以將這些成員新增至 CSQINP2 起始設定輸入資料集，或使用 CSQUTIL 程式來載入這些佇列定義。

   對於「佇列屬性」範例應用程式，您可以使用提供給其他範例應用程式的一或多個佇列。 或者，您可以使用自己的佇列。 不過，在提供的格式中，此範例僅適用於其名稱的前八個位元組中具有字元 CSQ4SAMP 的佇列。

z/OS *z/OS 上範例 CICS 應用程式的名稱*

本主題提供提供給範例 CICS 應用程式的程式摘要。

下表彙總了 CICS 應用程式:

- 第 986 頁的表 179 放置及取得範例
- 第 986 頁的表 180 佇列屬性範例
- 第 987 頁的表 181 Mail Manager 範例 (僅限 COBOL)
- 第 987 頁的表 182 信用檢查範例
- 第 988 頁的表 183 非同步耗用和發佈/訂閱範例

表 *179: CICS* 放置及取得範例

成員名稱	**針對語言**	**說明**	**檔案庫中提供的原始檔**	**程式庫中提供的執行檔**
CSQ4CCK1	C	放置來源程式	SCSQC37S	SCSQCICS
CSQ4CCJ1	C	取得來源程式	SCSQC37S	SCSQCICS
CSQ4CVJ1	COBOL	取得來源程式	SCSQCOBS	SCSQCICS
CSQ4CVK1	COBOL	放置來源程式	SCSQCOBS	SCSQCICS
CSQ4S100	independent	CICS 系統定義資料集	SCSQPROC	無

表 *180: CICS* 佇列屬性範例

成員名稱	**針對語言**	**說明**	**檔案庫中提供的原始檔**	**程式庫中提供的執行檔**
CSQ4CVC1	COBOL	來源程式	SCSQCOBS	SCSQCICS
CSQ4VMSG	COBOL	訊息定義	SCSQCOBC	無
CSQ4VCMS	COBOL	BMS 畫面定義	SCSQMAPS	SCSQCICS (名為 CSQ4ACM)
CSQ4CAC1	Assembler	來源程式	SCSQASMS	SCSQCICS

表 *180: CICS* 佇列屬性範例 *(*繼續*)*

成員名稱	針對語言	說明	檔案庫中提供的原始檔	程式庫中提供的執行檔
CSQ4AMSG	Assembler	訊息定義	SCSQMACS	無
CSQ4ACMS	Assembler	BMS 畫面定義	SCSQMAPS	SCSQCICS (名為 CSQ4ACM)
CSQ4CCC1	C	來源程式	SCSQC37S	SCSQCICS
CSQ4CMSG	C	訊息定義	SCSQC370	無
CSQ4CCMS	C	BMS 畫面定義	SCSQMAPS	SCSQCICS (名為 CSQ4ACM)
CSQ4S100	independent	CICS 系統定義資料集	SCSQPROC	無

表 *181: CICS Mail Manager* 範例 *(*僅限 *COBOL)*

成員名稱	說明	檔案庫中提供的原始檔
CSQ4CVD	IBM MQ for z/OS 物件定義	SCSQPROC
CSQ4CVD1	功能表程式的來源	SCSQCOBS
CSQ4CVD2	「取得郵件」程式的來源	SCSQCOBS
CSQ4CVD3	顯示訊息程式的來源	SCSQCOBS
CSQ4CVD4	傳送郵件程式的來源	SCSQCOBS
CSQ4CVD5	暱稱程式的來源	SCSQCOBS
CSQ4VDMS	BMS 畫面定義來源	SCSQMAPS
CSQ4S100	CICS 系統定義資料集	SCSQPROC
CSQ4VD0	資料定義	SCSQCOBC
CSQ4VD3	資料定義	SCSQCOBC
CSQ4VD4	資料定義	SCSQCOBC

表 *182: CICS* 信用檢查範例

成員名稱	針對語言	說明	檔案庫中提供的原始檔
CSQ4CVB	independent	IBM MQ 物件定義	SCSQPROC
CSQ4CCB	independent	IBM MQ 物件定義	SCSQPROC
CSQ4CVB1	COBOL	使用者介面程式的來源	SCSQCOBS
CSQ4CVB2	COBOL	信用申請經理的來源	SCSQCOBS
CSQ4CVB3	COBOL	核對帳戶程式的來源	SCSQCOBS
CSQ4CVB4	COBOL	配送程式的來源	SCSQCOBS
CSQ4CVB5	COBOL	代理查詢程式的來源	SCSQCOBS
CSQ4CCB1	C	使用者介面程式的來源	SCSQC37S
CSQ4CCB2	C	信用申請經理的來源	SCSQC37S

表 182: CICS 信用檢查範例 (繼續)			
成員名稱	**針對語言**	**說明**	**檔案庫中提供的原始檔**
CSQ4CCB3	C	核對帳戶程式的來源	SCSQC37S
CSQ4CCB4	C	配送程式的來源	SCSQC37S
CSQ4CCB5	C	代理查詢程式的來源	SCSQC37S
CSQ4CB0	C	併入檔	SCSQC370
CSQ4CBMS	C	BMS 畫面定義來源	SCSQMAPS
CSQ4VBMS	COBOL	BMS 畫面定義來源	SCSQMAPS
CSQ4VB0	COBOL	資料定義	SCSQCOBC
CSQ4VB1	COBOL	資料定義	SCSQCOBC
CSQ4VB2	COBOL	資料定義	SCSQCOBC
CSQ4VB3	COBOL	資料定義	SCSQCOBC
CSQ4VB4	COBOL	資料定義	SCSQCOBC
CSQ4VB5	COBOL	資料定義	SCSQCOBC
CSQ4VB6	COBOL	資料定義	SCSQCOBC
CSQ4VB7	COBOL	資料定義	SCSQCOBC
CSQ4VB8	COBOL	資料定義	SCSQCOBC
CSQ4BAQ	independent	VSAM 資料集的來源	SCSQPROC
CSQ4FILE	independent	用來建置 CSQ4CVB3 所使用 VSAM 資料集的 JCL	SCSQPROC
CSQ4S100	independent	CICS 系統定義資料集	SCSQPROC

表 183: CICS 非同步耗用和發佈/訂閱範例		
成員名稱	**說明**	**檔案庫中提供的原始檔**
CSQ4CVCN	簡式訊息耗用程式的來源	SCSQCOBS
CSQ4CVCT	控制訊息耗用程式的來源	SCSQCOBS
CSQ4CVEV	事件處理程式的來源	SCSQCOBS
CSQ4CVPT	訊息放置用戶端程式的來源	SCSQCOBS
CSQ4CVRG	登錄用戶端程式的來源	SCSQCOBS
CSQ4S100	CICS 系統定義資料集	SCSQPROC

z/OS 在 z/OS 上為 IMS 環境準備範例應用程式

部分 Credit Check 範例應用程式可以在 IMS 環境中執行。

若要準備應用程式的這個部分以搭配 CICS 範例執行，請先執行 第 985 頁的『在 z/OS 上為 CICS 環境準備範例應用程式』中說明的步驟。

然後，執行下列步驟：

1. 執行與建置任何 IMS IBM MQ for z/OS 應用程式時相同的步驟。 這些步驟在 第 863 頁的『建置 IMS (BMP 或 MPP) 應用程式』中列出。 要使用的檔案庫成員列在 第 989 頁的表 184 中。

2. 向 IMS 識別應用程式及資料庫。 PSBGEN、DBDGEN、ACB 定義、 IMSGEN 及 IMSDALOC 陳述式會提供範例，以啟用此功能。
3. 透過修改並執行為此目的提供的範例 JCL (CSQ4ILDB)，載入資料庫 CSQ4CA 。 此 JCL 會使用檔案 CSQ4BAQ 中的資料來載入資料庫。 使用資料庫 CSQ4CA 的 DD 陳述式來更新 IMS 控制區域。
4. 透過修改並執行為此目的提供的範例 JCL，以批次訊息處理 (BMP) 程式形式啟動核對帳戶程式。 此 JCL 會啟動批次導向 BMP 程式。 若要以訊息導向 BMP 程式執行程式，請從 JCL 中包含 IN= 陳述式的行移除註解字元。

z/OS *z/OS 上範例 IMS 應用程式的名稱*

此資訊提供一個表格，其中包含針對 Credit Check 範例 IMS 應用程式提供的來源及 JCL 的清單。

表 *184: Credit Check IMS* 範例的來源及 *JCL (*僅限 *C)*

成員名稱	說明	在檔案庫中提供
CSQ4CVB	IBM MQ 物件定義	SCSQPROC
CSQ4ICB3	核對帳戶程式的來源	SCSQC37S
CSQ4ICBL	用於載入核對帳戶資料庫的來源	SCSQC37S
CSQ4CBI	資料定義	SCSQC370
CSQ4PSBL	資料庫載入程式的 PSBGEN JCL	SCSQPROC
CSQ4PSB3	用於檢查帳戶程式的 PSBGEN JCL	SCSQPROC
CSQ4DBDS	資料庫 CSQ4CA 的 DBDGEN JCL	SCSQPROC
CSQ4GIMS	IMS CSQ4IVB3 及 CSQ4CA	SCSQPROC
CSQ4ACBG	CSQ4IVB3 的應用程式控制區塊 (ACB) 定義	SCSQPROC
CSQ4BAQ	資料庫的來源	SCSQPROC
CSQ4ILDB	資料庫載入工作的範例執行 JCL	SCSQPROC
CSQ4ICBR	用於檢查帳戶程式的範例執行 JCL	SCSQPROC
CSQ4DYNA	資料庫的 IMSDALOC 巨集定義	SCSQPROC

z/OS *z/OS* 上的「放置」範例

「放置範例」程式會使用 MQPUT 呼叫將訊息放置在佇列上。

來源程式在批次和 CICS 環境中的 C 和 COBOL 中提供 (請參閱 第 982 頁的表 172 和 第 986 頁的表 179)。

Put 範例的設計

程式邏輯的流程如下:

1. 使用 MQCONN 呼叫連接至佇列管理程式。 如果此呼叫失敗，請列印完成及原因碼，並停止處理。

   註: 如果您在 CICS 環境中執行範例，則不需要發出 MQCONN 呼叫; 如果這樣做，則會傳回 DEF_HCONN。 您可以對後續的 MQI 呼叫使用連線控點 MQHC_DEF_HCONN。
2. 使用 MQOO_OUTPUT 選項搭配 MQOPEN 呼叫來開啟佇列。 在此呼叫的輸入上，程式會使用步驟 第 991 頁的『1』中所傳回的連線控點。 對於物件描述子結構 (MQOD)，它會使用所有欄位的預設值，但佇列名稱欄位除外，該欄位會作為參數傳遞至程式。 如果 MQOPEN 呼叫失敗，請列印完成碼和原因碼，並停止處理。
3. 在發出 MQPUT 呼叫的程式內建立迴圈，直到將必要的訊息數放入佇列為止。 如果 MQPUT 呼叫失敗，則會提早放棄迴圈，不會嘗試進一步 MQPUT 呼叫，且會傳回完成碼和原因碼。

4. 搭配使用 MQCLOSE 呼叫與步驟 第 991 頁的『2』 中傳回的物件控點來關閉佇列。 如果此呼叫失敗，請列印完成及原因碼。
5. 使用 MQDISC 呼叫與步驟 第 991 頁的『1』 中傳回的連線控點中斷佇列管理程式的連線。 如果此呼叫失敗，請列印完成及原因碼。

   註: 如果您是在 CICS 環境中執行範例，則不需要發出 MQDISC 呼叫。

z/OS *z/OS* 上批次環境的「放置」範例

當考量批次環境的「放置」範例時，請使用本主題。

若要執行範例，請編輯並執行範例 JCL ，如 第 982 頁的『在 z/OS 上準備及執行批次環境的範例應用程式』 中所述。

程式在 EXEC PARM 中採用下列參數，以 C 中的空格及 COBOL 中的逗點區隔:

1. 佇列管理程式的名稱 (4 個字元)
2. 目標佇列的名稱 (48 個字元)
3. 訊息數目 (最多 4 位數)
4. 要在訊息中寫入的填補字元 (1 個字元)
5. 要在訊息中寫入的字元數 (最多 4 位數)
6. 訊息的持續性 (1 個字元 :P 代表持續性， N 代表非持續性)

如果您錯誤地輸入任何這些參數，則會收到適當的錯誤訊息。

來自範例的任何訊息都會寫入 SYSPRINT 資料集。

使用注意事項

- 為了讓範例保持簡單，語言版本之間有一些次要功能差異。 不過，如果您使用範例執行 JCL、 CSQ4BCJR 及 CSQ4BVJR 中所顯示參數的佈置，則這些差異會最小化。 沒有任何差異與 MQI 相關。
- CSQ4BCK1 可讓您輸入超過四位數的已傳送訊息數及訊息長度。
- 對於兩個數值欄位，請輸入 1 到 9999 範圍內的任何數字。 您輸入的值應該是正數。 例如，若要放置單一訊息，您可以輸入 1、01、001 或 0001 作為值。 如果您輸入非數值或負值，可能會收到錯誤。 例如，如果您輸入 -1 ， COBOL 程式會傳送 1 個位元組的訊息，但 C 程式會收到錯誤。
- 對於這兩個程式 (CSQ4BCK1 和 CSQ4BVK1) ，如果您想要訊息持續存在，則必須在持續性參數 + + PER + + 中輸入 P。 如果您無法這麼做，則訊息將是非持續性。

z/OS *z/OS* 上 *CICS* 環境的「放置」範例

當考量 CICS 環境的「放置」範例時，請使用本主題。

交易採用下列以逗點區隔的參數:

1. 訊息數目 (最多 4 位數)
2. 要在訊息中寫入的填補字元 (1 個字元)
3. 要在訊息中寫入的字元數 (最多 4 位數)
4. 訊息的持續性 (1 個字元 :P 代表持續性， N 代表非持續性)
5. 目標佇列的名稱 (48 個字元)

如果您錯誤地輸入其中任何參數，則會收到適當的錯誤訊息。

對於 COBOL 範例，透過輸入下列指令，在 CICS 環境中呼叫 Put 範例:

```
MVPT,9999,*,9999,P,QUEUE.NAME
```

對於 C 範例，輸入下列指令，以在 CICS 環境中呼叫「放置」範例:

```
MCPT,9999,*,9999,P,QUEUE.NAME
```

來自範例的任何訊息都會顯示在畫面上。

使用注意事項

- 為了讓範例保持簡單，語言版本之間有一些次要功能差異。 沒有任何差異與 MQI 相關。
- 如果您輸入的佇列名稱長度超過 48 個字元，則其長度會截斷至最多 48 個字元，但不會傳回任何錯誤訊息。
- 在輸入交易之前，請按 CLEAR 鍵。
- 對於兩個數值欄位，請輸入 1 到 9999 範圍內的任何數字。 您輸入的值應該是正數。 例如，若要放置單一訊息，您可以輸入值 1、01、001 或 0001。 如果您輸入非數值或負值，可能會收到錯誤。 例如，如果您輸入 -1 ，則 COBOL 程式會傳送 1 個位元組的訊息，且 C 程式會異常終止 malloc () 中的錯誤。
- 對於這兩個程式 (CSQ4CCK1 和 CSQ4CVK1) ，如果您想要持續保存訊息，請在持續性參數中輸入 P。 對於非持續訊息，請在持續性參數中輸入 N。 如果您輸入任何其他值，則會收到錯誤訊息。
- 訊息會放入同步點，因為除了程式呼叫期間所設定的參數之外，所有參數都使用預設值。

z/OS *z/OS* 上的 *Get* 範例

Get 範例程式會使用 MQGET 呼叫從佇列取得訊息。

來源程式在批次和 CICS 環境中的 C 和 COBOL 中提供 (請參閱 第 982 頁的表 172 和 第 986 頁的表 179)。

z/OS *z/OS* 上 *Get* 範例的設計

瞭解 Get 範例的設計，以及一些要考量的使用注意事項。

程式邏輯的流程如下:

1. 使用 MQCONN 呼叫連接至佇列管理程式。 如果此呼叫失敗，請列印完成及原因碼，並停止處理。

   註: 如果您在 CICS 環境中執行範例，則不需要發出 MQCONN 呼叫; 如果這樣做，則會傳回 DEF_HCONN。 您可以對後續的 MQI 呼叫使用連線控點 MQHC_DEF_HCONN。
2. 使用 MQOPEN 呼叫搭配 MQOO_INPUT_SHARED 和 MQOO_BROWSE 選項來開啟佇列。 在此呼叫的輸入上，程式會使用步驟 第 991 頁的『1』中所傳回的連線控點。 對於物件描述子結構 (MQOD) ，它會使用所有欄位的預設值，但佇列名稱欄位除外，該欄位會作為參數傳遞至程式。 如果 MQOPEN 呼叫失敗，請列印完成碼和原因碼，並停止處理。
3. 在發出 MQGET 呼叫的程式內建立迴圈，直到從佇列中擷取所需訊息數為止。 如果 MQGET 呼叫失敗，則會提早放棄迴圈，不會嘗試進一步 MQGET 呼叫，且會傳回完成及原因碼。 下列是在 MQGET 呼叫上指定的選項:

   - MQGMO_NO_WAIT
   - MQGMO_ACCEPT_TRUNCATED_MESSAGE
   - MQGMO_SYNCPOINT 或 MQGMO_NO_SYNCPOINT
   - MQGMO_BROWSE_FIRST 及 MQGMO_BROWSE_NEXT

   如需這些選項的說明，請參閱 MQGET。 對於每一個訊息，會列印訊息號碼，後面接著訊息及訊息資料的長度。
4. 搭配使用 MQCLOSE 呼叫與步驟 第 991 頁的『2』中傳回的物件控點來關閉佇列。 如果此呼叫失敗，請列印完成及原因碼。
5. 使用 MQDISC 呼叫與步驟 第 991 頁的『1』中傳回的連線控點中斷佇列管理程式的連線。 如果此呼叫失敗，請列印完成及原因碼。

   註: 如果您是在 CICS 環境中執行範例，則不需要發出 MQDISC 呼叫。

使用注意事項

- 為了讓範例保持簡單，語言版本之間有一些次要功能差異。 不過，如果您使用範例執行 JCL、 CSQ4BCJR 及 CSQ4BVJR 中所顯示參數的佈置，則這些差異會最小化。 沒有任何差異與 MQI 相關。
- CSQ4BCJ1 可讓您輸入超過四位數的擷取訊息數。

- 超過 64 KB 的訊息會被截斷。
- CSQ4BCJ1 只能正確顯示字元訊息，因為它只會在顯示第一個 NULL (\0) 字元之前才會顯示。
- 對於數字訊息數欄位，請輸入 1 到 9999 範圍內的任何數字。 您輸入的值應該是正數。 例如，若要取得單一訊息，您可以輸入 1、01、001 或 0001 作為值。 如果您輸入非數值或負值，可能會收到錯誤。 例如，如果您輸入 -1， COBOL 程式會擷取一則訊息，但 C 程式不會擷取任何訊息。
- 對於這兩個程式 (CSQ4BCJ1 和 CSQ4BVJ1) ，如果您想要瀏覽訊息，請在 get 參數 + + GET + + 中輸入 B。
- 對於這兩個程式 CSQ4BCJ1 及 CSQ4BVJ1，請在同步點參數 + + SYNC + + 中輸入 S ， 以在同步點擷取訊息。

z/OS *z/OS 上批次環境的 Get 範例*

若要執行範例，請編輯並執行範例 JCL ，如 第 982 頁的『在 z/OS 上準備及執行批次環境的範例應用程式』中所述。

程式在 EXEC PARM 中採用下列參數，以 C 中的空格及 COBOL 中的逗點區隔:

1. 佇列管理程式的名稱 (4 個字元)
2. 目標佇列的名稱 (48 個字元)
3. 要取得的訊息數 (最多 4 位數)
4. 瀏覽/取得訊息選項 (1 個字元: B 表示瀏覽， D 表示破壞性取得訊息)
5. 同步點控制 (1 個字元: S 代表同步點， N 代表無同步點)

如果您不正確地輸入其中任何參數，則會收到適當的錯誤訊息。

範例的輸出會寫入 SYSPRINT 資料集:

```
====================================
PARAMETERS PASSED :
QMGR    - VC9
QNAME    - A.Q
NUMMSGS   - 000000002
GET     - D
SYNCPOINT  - N
====================================
MQCONN SUCCESSFUL
MQOPEN SUCCESSFUL
000000000 : 000000010 : **********
000000001 : 000000010 : **********
000000002 MESSAGES GOT FROM QUEUE
MQCLOSE SUCCESSFUL
MQDISC SUCCESSFUL
```

z/OS *z/OS 上 CICS 環境的 Get 範例*

CICS 環境的 Get 範例特殊考量。

交易會採用 EXEC PARM 中的下列參數，並以逗點區隔:

1. 要取得的訊息數 (最多四位數)
2. 瀏覽/取得訊息選項 (一個字元 :B 表示瀏覽， D 表示破壞性取得訊息)
3. 同步點控制 (一個字元: S 代表同步點， N 代表無同步點)
4. 目標佇列的名稱 (48 個字元)

如果您不正確地輸入其中任何參數，則會收到適當的錯誤訊息。

對於 COBOL 範例，在 CICS 環境中輸入下列指令來呼叫 Get 範例:

```
MVGT,9999,B,S,QUEUE.NAME
```

對於 C 範例，在 CICS 環境中輸入下列指令來呼叫 Get 範例:

```
MCGT,9999,B,S,QUEUE.NAME
```

從佇列中擷取訊息時，會將訊息放置在與 CICS 交易同名的 CICS 暫時儲存體佇列中 (例如， C 範例的 MCGT)。

以下是 Get 範例的範例輸出:

```
*************************** TOP OF QUEUE ************************
000000000 : 000000010: **********
000000001 : 000000010 :**********
*************************** BOTTOM OF QUEUE *********************
```

使用注意事項

- 為了讓範例保持簡單，語言版本之間有一些次要功能差異。 沒有任何差異與 MQI 相關。
- 如果您輸入的佇列名稱長度超過 48 個字元，則其長度會截斷至最多 48 個字元，但不會傳回任何錯誤訊息。
- 在輸入交易之前，請按 CLEAR 鍵。
- CSQ4CCJ1 只能正確顯示字元訊息，因為它只會在顯示第一個 NULL (\0) 字元之前才會顯示。
- 對於數值欄位，請輸入 1 到 9999 範圍內的任何數字。 您輸入的值應該是正數。 例如，若要取得單一訊息，您可以輸入值 1、01、001 或 0001。 如果您輸入非數值或負值，可能會收到錯誤。
- 在 C 中超過 24 526 個位元組的訊息及在 COBOL 中超過 9 950 個位元組的訊息會被截斷。 這是因為使用 CICS 暫時儲存體佇列的方式。
- 對於這兩個程式 (CSQ4CCK1 和 CSQ4CVK1) ，如果您想要瀏覽訊息，請在 get 參數中輸入 B ，否則輸入 D。 這會執行破壞性 MQGET 呼叫。 如果您輸入任何其他值，則會收到錯誤訊息。
- 對於這兩個程式 (CSQ4CCJ1 和 CSQ4CVJ1) ，在同步點參數中輸入 S 以擷取同步點中的訊息。 如果您在同步點參數中輸入 N ，則會從同步點發出 MQGET 呼叫。 如果您輸入任何其他值，則會收到錯誤訊息。

z/OS z/OS 上的瀏覽範例

「瀏覽」範例是一個批次應用程式，示範如何使用 MQGET 呼叫來瀏覽佇列上的訊息。

應用程式會逐步執行佇列中的所有訊息，並列印每一個訊息的前 80 個位元組。 您可以使用此應用程式來查看佇列上的訊息，而不變更它們。

以 COBOL、組譯器、PL/I 和 C 語言提供來源程式和範例執行 JCL (請參閱 第 983 頁的表 173)。

若要啟動應用程式，請編輯並執行範例執行 JCL ，如 第 982 頁的『在 z/OS 上準備及執行批次環境的範例應用程式』中所述。 您可以在執行 JCL 中指定佇列名稱，以查看您自己的佇列之一上的訊息。

當您執行應用程式 (且佇列上有一些訊息) 時，輸出資料集會看起來如下:

```
07/12/1998            SAMPLE QUEUE REPORT     PAGE  1
QUEUE MANAGER NAME : VC4
QUEUE NAME : CSQ4SAMP.DEAD.QUEUE
RELATIVE
MESSAGE  MESSAGE
NUMBER  LENGTH ------------------- MESSAGE DATA -------------

1    740 HELLO. PLEASE CALL ME WHEN YOU GET BACK.
2    429 CSQ4BQRM
3    429 CSQ4BQRM
4    429 CSQ4BQRM
5    22 THIS IS A TEST MESSAGE
6     8 CSQ4TEST
7    36 CSQ4MSG - ANOTHER TEST MESSAGE.....
!8     9 CSQ4STOP
********** END OF REPORT **********
```

如果佇列中沒有訊息，則資料集只會包含標題及 **報告結尾** 訊息。 如果任何 MQI 呼叫發生錯誤，則會將完成碼及原因碼新增至輸出資料集。

z/OS *z/OS* 上「瀏覽」範例的設計

「瀏覽」範例應用程式使用單一程式模組; 每一種支援的程式設計語言都提供一個程式模組。

程式邏輯的流程如下:

1. 開啟列印資料集，並列印報告的標題行。 請檢查是否已從執行 JCL 傳遞佇列管理程式及佇列的名稱。 如果已傳遞兩個名稱，則列印包含名稱的報告行。 如果沒有，請列印錯誤訊息，關閉列印資料集，並停止處理。

   程式測試從 JCL 傳遞參數的方式取決於撰寫程式所用的語言; 如需相關資訊，請參閱 第 994 頁的『z/OS 上與語言相關的設計考量』。
2. 使用 MQCONN 呼叫連接至佇列管理程式。 如果此呼叫不成功，請列印完成及原因碼，關閉列印資料集，並停止處理。
3. 使用 MQOPEN 呼叫搭配 MQOO_BROWSE 選項來開啟佇列。 在此呼叫的輸入時，程式會使用步驟 第 994 頁的『2』中傳回的連線控點。 對於物件描述子結構 (MQOD)，除了佇列名稱 (在步驟 第 994 頁的『1』中傳遞) 之外，它會使用所有欄位的預設值。 如果此呼叫不成功，請列印完成及原因碼，關閉列印資料集，並停止處理。
4. 使用 MQGET 呼叫來瀏覽佇列上的第一個訊息。 在此呼叫的輸入上，程式會指定:
   - 來自步驟 第 994 頁的『2』 及 第 994 頁的『3』 的連線及佇列控點
   - 所有欄位都設為其起始值的 MQMD 結構
   - 兩個選項:
     - MQGMO_BROWSE_FIRST
     - MQGMO_ACCEPT_TRUNCATED_MSG
   - 大小為 80 位元組的緩衝區，用來保留從訊息複製的資料

   MQGMO_ACCEPT_TRUNCATED_MSG 選項可讓呼叫完成，即使訊息長於呼叫中指定的 80 位元組緩衝區也一樣。 如果訊息長於緩衝區，則會截斷訊息以適合緩衝區，並將完成及原因碼設為顯示此訊息。 此範例是設計成將訊息截斷為 80 個字元，讓報告易於閱讀。 緩衝區大小由 DEFINE 陳述式設定，因此您可以在想要時輕鬆變更它。
5. 執行下列迴圈，直到 MQGET 呼叫失敗為止:
   a. 列印報告的一行，顯示:
      - 訊息的序號 (這是瀏覽作業的計數)。
      - 訊息的真實長度 (不是截斷的長度)。 此值會在 MQGET 呼叫的 DataLength 欄位中傳回。
      - 訊息資料的前 80 個位元組。

   b. 將 MQMD 結構的 MsqId 和 CorrelId 欄位重設為空值

   c. 搭配使用 MQGET 呼叫與下列兩個選項，以瀏覽下一個訊息:
      - MQGMO_BROWSE_NEXT
      - MQGMO_ACCEPT_TRUNCATED_MSG
6. 如果 MQGET 呼叫失敗，請測試原因碼，以查看呼叫是否因為瀏覽游標已進入佇列結尾而失敗。 在此情況下，請列印 **報告結尾** 訊息，並跳至步驟 第 994 頁的『7』; 否則，列印完成及原因碼，關閉列印資料集，並停止處理。
7. 搭配使用 MQCLOSE 呼叫與步驟 第 994 頁的『3』中傳回的物件控點來關閉佇列。
8. 使用 MQDISC 呼叫與步驟 第 994 頁的『2』中傳回的連線控點中斷佇列管理程式的連線。
9. 關閉列印資料集並停止處理。

z/OS *z/OS* 上與語言相關的設計考量

以四種程式設計語言提供「瀏覽」範例的來源模組。

來源模組之間有兩個主要差異:

- 在測試從執行 JCL 傳遞的參數時， COBOL、PL/I 和組譯語言模組會搜尋逗點字元 (，)。 如果 JCL 通過 PARM=(,LOCALQ1)，則應用程式會嘗試在預設佇列管理程式上開啟佇列 LOCALQ1 。 如果逗點之後沒有名稱 (或沒有逗點)，應用程式會傳回錯誤。 C 模組不會搜尋逗點字元。 如果 JCL 傳遞單一參數 (例如, PARM=('LOCALQ1'))，則 C 模組會使用此名稱作為預設佇列管理程式上的佇列名稱。
- 為了讓組譯語言模組保持簡單，它會在建立列印報告時使用日期格式 *yy/ddd* (例如 05/116)。 其他模組使用 *mm/dd/yy* 格式的行事曆日期。

z/OS *z/OS* 上的「列印訊息」範例

「列印訊息」範例是一個批次應用程式，示範如何使用 MQGET 呼叫從佇列中移除所有訊息。

「列印訊息」範例使用三個參數:

1. 佇列管理程式的名稱
2. 來源佇列的名稱
3. 內容的選用參數

它也會針對每一個訊息，列印訊息描述子的欄位，後面接著訊息資料。 程式會以十六進位及字元來列印資料 (如果它們是可列印的)。 如果無法列印字元，程式會以句點字元 (.) 取代它。 您可以在診斷將訊息放入佇列之應用程式的問題時使用程式。

內容參數允許的值如下:

表 *185:* 內容參數允許的值

值	行為
0	預設行為。 遞送至應用程式的內容取決於從中擷取訊息的 **PropertyControl** 佇列屬性。

1	會建立訊息控點，並與 MQGET 搭配使用。 訊息的內容 (訊息描述子或延伸中包含的內容除外) 會以與訊息描述子類似的方式顯示。 例如:
	`****Message properties****`
	`  property name: property value`
	或者如果沒有可用的內容:
	`****Message properties****`
	`  None`
	使用 `printf` 顯示數值，字串值以單引號括住，位元組字串以 X 及單引號括住，就像訊息描述子一樣。

2	已指定 MQGMO_NO_PROPERTIES ，因此只會傳回訊息描述子內容。
3	已指定 MQGMO_PROPERTIES_FORCE_MQRFH2 ，因此會在訊息資料中傳回所有內容。
4	已指定 MQGMO_PROPERTIES_COMPATIBILITY ，因此可以根據是否包含 IBM MQ 內容來傳回所有內容，否則會捨棄這些內容。

您可以變更應用程式，讓它瀏覽訊息，而不是從佇列中移除它們。 若要執行此動作，請使用 -DBROWSE 選項進行編譯，以定義 BROWSE 巨集，如 第 996 頁的『z/OS 上「列印訊息」範例的設計』中所示。 在 SCSQLOAD 程式庫中為您提供執行碼。 模組 CSQ4BCG0 使用 -DBROWSE; 模組 CSQ4BCG1 破壞性讀取佇列。

應用程式具有以 C 語言撰寫的單一來源程式。 也會提供執行 JCL 程式碼範例 (請參閱 第 983 頁的表 174)。

若要啟動應用程式，請編輯並執行範例執行 JCL ，如 第 982 頁的『在 z/OS 上準備及執行批次環境的範例應用程式』中所述。 當您執行應用程式 (且佇列上有一些訊息) 時，輸出資料集與 第 996 頁的圖 139 中的輸出資料集類似。

```
CSQ4BCG1 - starts here
**********************

MQCONN to MQ1E
MQOPEN - 'TEST.QUEUE'
MQCRTMH


MQGET of message number 1
****Message descriptor****
StrucId : 'MD ' Version : 2
Report  : 0 MsgType : 8
Expiry  : -1 Feedback : 0
Encoding : 785 CodedCharSetId : 500
Format : 'MQSTR   '
Priority : 0 Persistence : 0
MsgId : X'C3E2D840D4D8F1C540404040404040C1EA537F03167D88'
CorrelId : X'C3E2D840D4D8F1C540404040404040C1EA537F0317A928'
BackoutCount : 0
ReplyToQ    : '                          '
ReplyToQMgr  : ''
** Identity Context
UserIdentifier : 'FRED    '
AccountingToken :
X'0000000000000000000000000000000000000000000000000000000000000000'
ApplIdentityData : '                 '
** Origin Context
PutApplType  : '2'
PutApplName  : 'FRED6            '
PutDate : '20080207'  PutTime : '17373745'
ApplOriginData : '   '
GroupId : X'000000000000000000000000000000000000000000000000'
MsgSeqNumber  : '1'
Offset     : '0'
MsgFlags    : '0'
OriginalLength : '-1'

****Message properties****
None

****  Message   ****

length - 30 bytes

00000000: E388 89A2 4089 A240 8140 A289 9497 9385 'This is a simple'
00000010: 40A3 85A2 A340 9485 A2A2 8187 855A   ' test message! '



No more messages
MQDLTMH
MQCLOSE
MQDISC
```

圖 *139:* 來自「列印訊息」範例應用程式的報告範例

z/OS *z/OS* 上「列印訊息」範例的設計

「列印訊息」範例應用程式使用以 C 語言撰寫的單一程式。

程式邏輯的流程如下:

1. 請檢查是否已從執行 JCL 傳遞佇列管理程式及佇列的名稱。 如果沒有，則列印錯誤訊息並停止處理。
2. 使用 MQCONN 呼叫連接至佇列管理程式。 如果此呼叫失敗，請列印完成碼及原因碼，並停止處理; 否則，請列印佇列管理程式的名稱。
3. 使用 MQOPEN 呼叫搭配 MQOO_INPUT_SHARED 選項來開啟佇列。

   註: 如果您想要應用程式瀏覽訊息而不是從佇列中移除訊息，請使用 -DBROWSE 編譯範例，或在來源開頭新增 `#define BROWSE` 。 當您執行此動作時，巨集前置處理器會在程式中新增一行，以在編譯中選取 MQOO_BROWSE 選項。

   在此呼叫的輸入時，程式會使用步驟 第 996 頁的『2』 中傳回的連線控點。 對於物件描述子結構 (MQOD)，除了佇列名稱 (在步驟 第 996 頁的『1』 中傳遞) 之外，它會使用所有欄位的預設值。 如果此呼叫失敗，請列印完成及原因碼，並停止處理; 否則，請列印佇列名稱。

4. 如果您使用訊息控點來取得訊息內容，請使用 MQCRTMH 來建立這類控點，以與後續的 MQGET 呼叫搭配使用。 如果此呼叫不成功，請列印完成及原因碼，並停止處理。
5. 設定取得訊息選項，以反映任何訊息內容的要求動作。
6. 執行下列迴圈，直到 MQGET 呼叫失敗為止:
   a. 將緩衝區起始設定為空白，以便訊息資料不會因緩衝區中已有的任何資料而毀損。
   b. 將 MQMD 結構的 `MsgId` 和 `CorrelId` 欄位設為空值，以便 MQGET 呼叫從佇列中選取第一個訊息。
   c. 使用 MQGET 呼叫從佇列取得訊息。 在此呼叫的輸入上，程式會指定:
      - 步驟 第 996 頁的『2』 和 第 996 頁的『3』中的連線和物件控點。
      - MQMD 結構，所有欄位都設為其起始值。 (對於每一個 MQGET 呼叫，`MsgId` 和 `CorrelId` 會重設為空值。)
      - 選項 MQGMO_NO_WAIT。

        註: 如果您想要應用程式瀏覽訊息而不是從佇列中移除訊息，請使用 -DBROWSE 編譯範例，或在來源開頭新增 `#define BROWSE`。 當您這麼做時，巨集前置處理器會將程式中選取 MQGMO_BROWSE_NEXT 選項的那一行新增至編譯。 當在針對佇列的呼叫中使用此選項時，如果先前未使用任何瀏覽游標與現行物件控點一起使用，則瀏覽游標會在邏輯上位於第一則訊息之前。
      - 大小為 64KB 的緩衝區，用來保留從訊息複製的資料。
   d. 呼叫 printMD 子常式。 這會列印訊息描述子中每一個欄位的名稱，後面接著其內容。
   e. 如果您在步驟 第 997 頁的『4』 中建立訊息控點，請呼叫 printProperties 子常式來顯示任何訊息內容。
   f. 列印訊息的長度，後面接著訊息資料。 每一行訊息資料都採用下列格式:
      - 此部分資料的相對位置 (以十六進位表示)
      - 16 位元組十六進位資料
      - 字元格式的相同 16 個位元組資料，如果它是可列印的 (不可列印的字元會被句點取代)
7. 如果 MQGET 呼叫失敗，請測試原因碼，以查看該呼叫是否因為佇列上沒有其他訊息而失敗。 在此情況下，請列印訊息: 沒有其他訊息; 否則，請列印完成碼和原因碼。 在這兩種情況下，請跳至步驟 第 997 頁的『9』。

   註: 如果 MQGET 呼叫找到超過 64KB 資料的訊息，則它會失敗。 若要變更程式以處理較大的訊息，您可以執行下列其中一項:
   - 將 MQGMO_ACCEPT_TRUNCATED_MSG 選項新增至 MQGET 呼叫，以便呼叫取得第一個 64KB 資料，並捨棄剩餘的資料
   - 讓程式在找到具有此資料數量的訊息時，將該訊息保留在佇列上
   - 增加緩衝區大小
8. 如果您在步驟 第 997 頁的『4』 中建立訊息控點，請呼叫 MQDLTMH 來刪除它。
9. 搭配使用 MQCLOSE 呼叫與步驟 第 996 頁的『3』中傳回的物件控點來關閉佇列。
10. 使用 MQDISC 呼叫與步驟 第 996 頁的『2』中傳回的連線控點中斷佇列管理程式的連線。

z/OS z/OS 上的佇列屬性範例

「佇列屬性」範例是一種交談模式 CICS 應用程式，示範如何使用 MQINQ 及 MQSET 呼叫。

它顯示如何查詢佇列的 **InhibitPut** 及 **InhibitGet** 屬性值，以及如何變更它們，讓程式無法將訊息放置在佇列上，或從佇列取得訊息。 當您測試程式時，您可能想要以此方式 鎖定 佇列。

為了防止意外干擾您自己的佇列，此範例僅適用於名稱前八個位元組具有字元 CSQ4SAMP 的佇列物件。 不過，原始碼包含註解，告訴您如何移除這項限制。

原始程式以 COBOL、組譯器及 C 語言提供 (請參閱 第 986 頁的表 180)。

範例的組譯語言版本使用可重新輸入的程式碼。 若要這樣做，您將注意到該範例版本中每一個 MQI 呼叫的程式碼都包括 MF 關鍵字; 例如:

```
CALL MQCONN,(NAME,HCONN,COMPCODE,REASON),MF=(E,PARMAREA),VL
```

(VL 關鍵字表示您可以使用 CICS Execution Diagnostic Facility (CEDF) 提供的交易來除錯程式。) 如需撰寫可重新輸入程式的相關資訊，請參閱 以 System/390 組譯語言撰寫程式碼。

若要啟動應用程式，請啟動 CICS 系統並使用下列 CICS 交易:

- 若為 COBOL: MVC1
- 對於組譯語言: MAC1
- 若為 C: MCC1

您可以變更步驟 3 中提及的 CSD 資料集，以變更任何這些交易的名稱。

樣本的設計

當您啟動範例時，它會顯示具有下列欄位的畫面對映:

- 佇列的名稱
- 使用者要求 (有效動作為 :inquire、allow 或 inquire)
- 佇列的放置作業現行狀態
- 佇列取得作業的現行狀態

前兩個欄位用於使用者輸入。 應用程式會填入最後兩個欄位: 它們會顯示單字 INHIBITED 或單字 ALLOWED。

應用程式會驗證您在前兩個欄位中輸入的值。 它會檢查佇列名稱是否以字元 CSQ4SAMP 開頭，並檢查您是否在「動作」欄位中輸入三個有效要求的其中一個。 應用程式會將所有輸入轉換為大寫，因此您無法使用名稱包含小寫字元的任何佇列。

如果您在 **動作** 欄位中輸入 `inquire`，則通過程式邏輯的流程如下:

1. 使用 MQOPEN 呼叫搭配 MQOO_INQUIRE 選項來開啟佇列
2. 使用選取元 MQIA_INHIBIT_GET 和 MQIA_INHIBIT_PUT 來呼叫 MQINQ
3. 使用 MQCLOSE 呼叫來關閉佇列
4. 分析在 MQINQ 呼叫的 **IntAttrs** 參數中傳回的屬性，並視情況將單字 INHIBITED 或 ALLOWED 移至相關畫面欄位

如果您在 **動作** 欄位中輸入 `inhibit`，則通過程式邏輯的流程如下:

1. 使用 MQOO_SET 選項搭配 MQOPEN 呼叫來開啟佇列
2. 使用 **IntAttrs** 參數中的選取元 MQIA_INHIT_GET 和 MQIA_INHIT_PUTH， 以及值 MQQA_GET_INHIBITED 和 MQQA_PUT_INHIBITED 來呼叫 MQSET
3. 使用 MQCLOSE 呼叫來關閉佇列
4. 將單字 INHIBITED 移至相關畫面欄位

如果您在 **動作** 欄位中輸入 `allow`，則應用程式會執行與禁止要求類似的處理程序。 唯一的差異是屬性的設定以及畫面上顯示的單字。

當應用程式開啟佇列時，它會使用佇列管理程式的預設連線控點。 (當您啟動 CICS 系統時， CICS 會建立與佇列管理程式的連線。) 在此階段，應用程式可以設陷下列錯誤:

- 應用程式未連接至佇列管理程式
- 佇列不存在
- 使用者未獲授權存取佇列
- 應用程式未獲授權開啟佇列

對於其他 MQI 錯誤，應用程式會顯示完成碼及原因碼。

z/OS *z/OS 上的 Mail Manager 範例*

「郵件管理程式」範例應用程式是一套程式，可示範在單一環境內及跨不同環境傳送及接收訊息。 應用程式是簡式電子郵件系統，可讓使用者交換訊息，即使他們使用不同的佇列管理程式也一樣。

應用程式示範如何使用 MQOPEN 呼叫及將 IBM MQ for z/OS 指令放置在系統指令輸入佇列上，來建立佇列。

提供了三個版本的應用程式:

- 以 COBOL 撰寫的 CICS 應用程式
- 以 COBOL 撰寫的 TSO 應用程式
- 以 C 撰寫的 TSO 應用程式

z/OS 在 *z/OS* 上準備郵件管理程式範例

郵件管理程式在兩個環境中執行的版本中提供。 在執行應用程式之前必須執行的準備，取決於您要使用的環境。

使用者可以同時從 TSO 和 CICS 存取郵件佇列和暱稱佇列，只要其登入使用者 ID 在每一個系統上都相同即可。

您必須先設定該佇列管理程式的訊息通道，才能將訊息傳送至另一個佇列管理程式。 若要這麼做，請使用 IBM MQ 的通道控制功能，如 通道控制功能中所述。

準備 TSO 環境的範例

請遵循下列步驟:

1. 準備範例，如 第 984 頁的『在 z/OS 上為 TSO 環境準備範例應用程式』中所述。
2. 自訂針對範例所提供的 CLIST ， 以定義:
   - 畫面的位置
   - 訊息檔案的位置
   - 載入模組的位置
   - 您要與應用程式搭配使用的佇列管理程式名稱

   針對範例的每一個語言版本提供個別 CLIST:
   - 若為 COBOL 版本: CSQ4RVD1
   - 若為 C 版本: CSQ4RCD1
3. 請確定應用程式所使用的佇列可在佇列管理程式上使用。 (佇列定義在 CSQ4CVD 中。)

註: VS COBOL II 不支援使用 ISPF 進行多工。 這表示您無法在分割畫面的兩端使用「郵件管理程式」範例應用程式。 如果您這麼做，結果將無法預期。

z/OS 在 *z/OS* 上執行郵件管理程式範例

若要在 CICS Transaction Server for z/OS 環境中啟動範例，請執行交易 MAIL。 如果您尚未登入 CICS，應用程式會提示您輸入可以傳送郵件的使用者 ID。

當您啟動應用程式時，它會開啟您的郵件佇列。 如果此佇列不存在，則應用程式會為您建立一個佇列。 郵件佇列的名稱格式為 CSQ4SAMP.MAILMGR。 *userid*，其中 *userid* 取決於環境:

在 TSO 中
使用者的 TSO ID

輸入 CICS
當「郵件管理程式」啟動時出現提示時，使用者輸入的使用者 CICS 登入或使用者輸入的使用者 ID

「郵件管理程式」使用的佇列名稱所有部分都必須是大寫。

然後，應用程式會呈現具有下列選項的功能表畫面:

- 讀取來信

- 傳送郵件
- 建立暱稱

功能表畫面也會顯示在郵件佇列上等待的訊息數。 每一個功能表選項都會顯示另一個畫面:

讀取來信
「郵件管理程式」會顯示郵件佇列上的訊息清單。 (只會顯示佇列上的前 99 則訊息。) 如需此畫面的範例，請參閱 第 1003 頁的圖 142。 當您從此清單中選取訊息時，會顯示訊息的內容 (請參閱 第 1004 頁的圖 143)。

傳送郵件
畫面會提示您輸入:

- 您要向其傳送訊息的使用者名稱
- 擁有其郵件佇列的佇列管理程式名稱
- 您的訊息文字

在使用者名稱欄位中，您可以輸入使用者 ID 或您使用「郵件管理程式」所建立的暱稱。 如果使用者的郵件佇列是由您正在使用的相同佇列管理程式所擁有，您可以將佇列管理程式名稱欄位保留空白，如果您在使用者名稱欄位中輸入暱稱，則必須將它保留空白:

- 如果您只指定使用者名稱，則程式會先假設該名稱是暱稱，並將訊息傳送至該名稱所定義的物件。 如果沒有這樣的暱稱，程式會嘗試將訊息傳送至該名稱的本端佇列。
- 如果您同時指定使用者名稱及佇列管理程式名稱，則程式會將訊息傳送至由這兩個名稱定義的郵件佇列。

例如，如果您要將訊息傳送至遠端佇列管理程式 QM12 上的使用者 JSONESM，則可以使用下列兩種方式之一來傳送訊息:

- 請使用這兩個欄位來指定佇列管理程式 QM12 中的使用者 JSONESM。
- 定義該使用者的暱稱 (例如， MARY)，並將訊息傳送給他們，方法是在使用者名稱欄位中放置 MARY，而在佇列管理程式名稱欄位中沒有任何訊息。

建立暱稱
您可以定義容易記住的名稱，當您將訊息傳送給您經常聯絡的另一個使用者時，可以使用該名稱。 系統會提示您輸入其他使用者的使用者 ID，以及擁有其郵件佇列的佇列管理程式名稱。

暱稱是具有 CSQ4SAMP.MAILMGR。 *userid.nickname*，其中 *userid* 是您自己的使用者 ID，而 *nickname* 是您要使用的暱稱。 使用這種結構的名稱，使用者可以各有自己的暱稱組合。

程式建立的佇列類型取決於您如何完成「建立暱稱」畫面的欄位:

- 如果您只指定使用者名稱，或佇列管理程式名稱與「郵件管理程式」所連接的佇列管理程式名稱相同，則程式會建立別名佇列。
- 如果您同時指定使用者名稱及佇列管理程式名稱 (且佇列管理程式不是「郵件管理程式」所連接的佇列管理程式)，則程式會建立遠端佇列的本端定義。 程式不會檢查此定義所解析的佇列是否存在，甚至不會檢查遠端佇列管理程式是否存在。

例如，如果您自己的使用者 ID 是 SMITHK，且您為使用者 JSONESM (使用遠端佇列管理程式 QM12) 建立名為 MARY 的暱稱，則暱稱程式會建立名為 CSQ4SAMP.MAILMGR.SMITHK.MARY。 此定義解析為 Mary 的郵件佇列，即 CSQ4SAMP.MAILMGR.JONESM 在佇列管理程式中 QM12。 如果您自己使用佇列管理程式 QM12，則程式會改為建立同名 (CSQ4SAMP.MAILMGR.SMITHK.MARY)。

TSO 應用程式的 C 版本比 COBOL 版本更能使用 ISPF 的訊息處理功能。 您可能會注意到 C 和 COBOL 版本顯示不同的錯誤訊息。

z/OS *z/OS 上 Mail Manager 範例的設計*

下列各節說明組成「郵件管理程式」範例應用程式的每一個程式。

對於 TSO 版本，程式與應用程式使用的畫面之間的關係顯示在 第 1001 頁的圖 140 中，對於 CICS Transaction Server for z/OS 版本，則顯示在 第 1002 頁的圖 141 中。

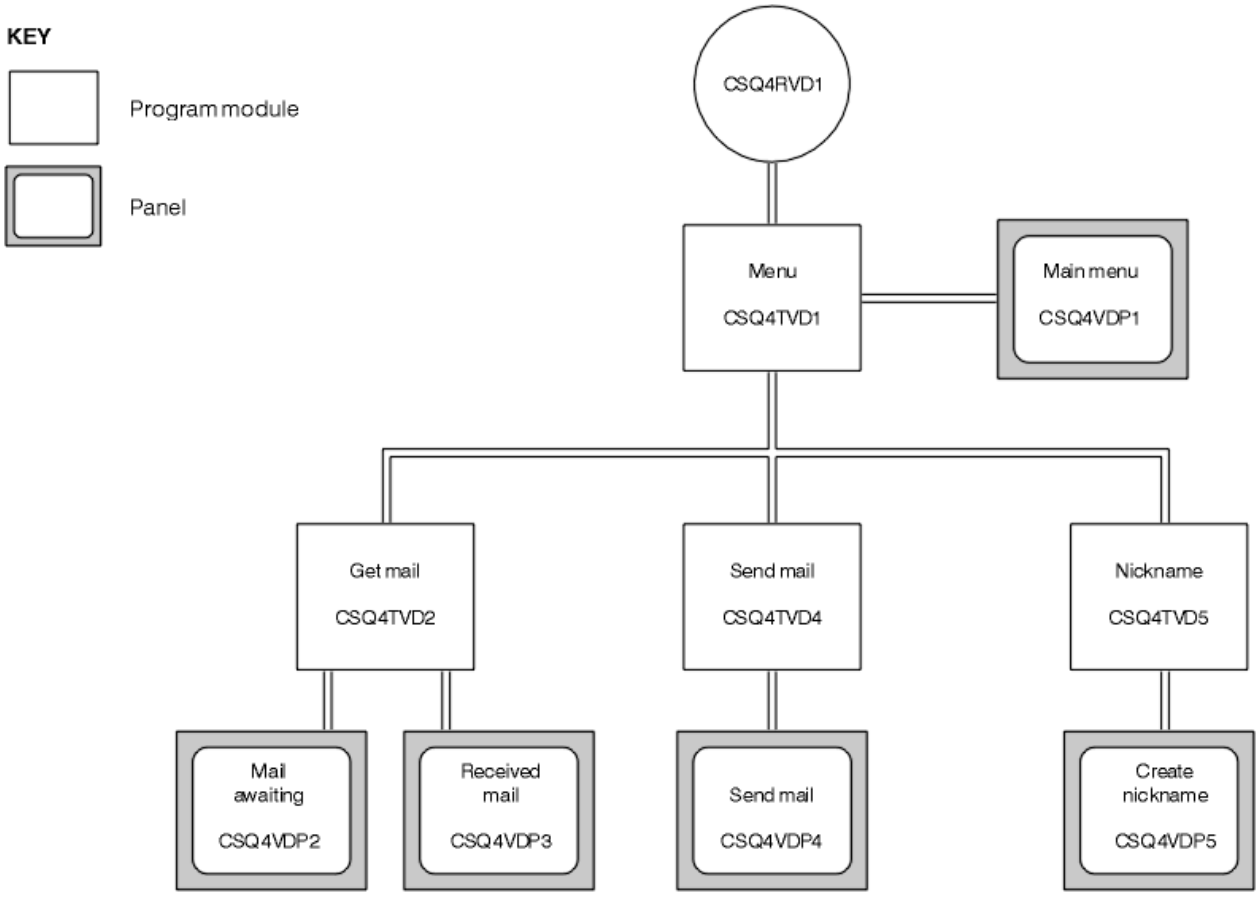


圖 *140:* 「郵件管理程式」*TSO* 版本的程式及畫面

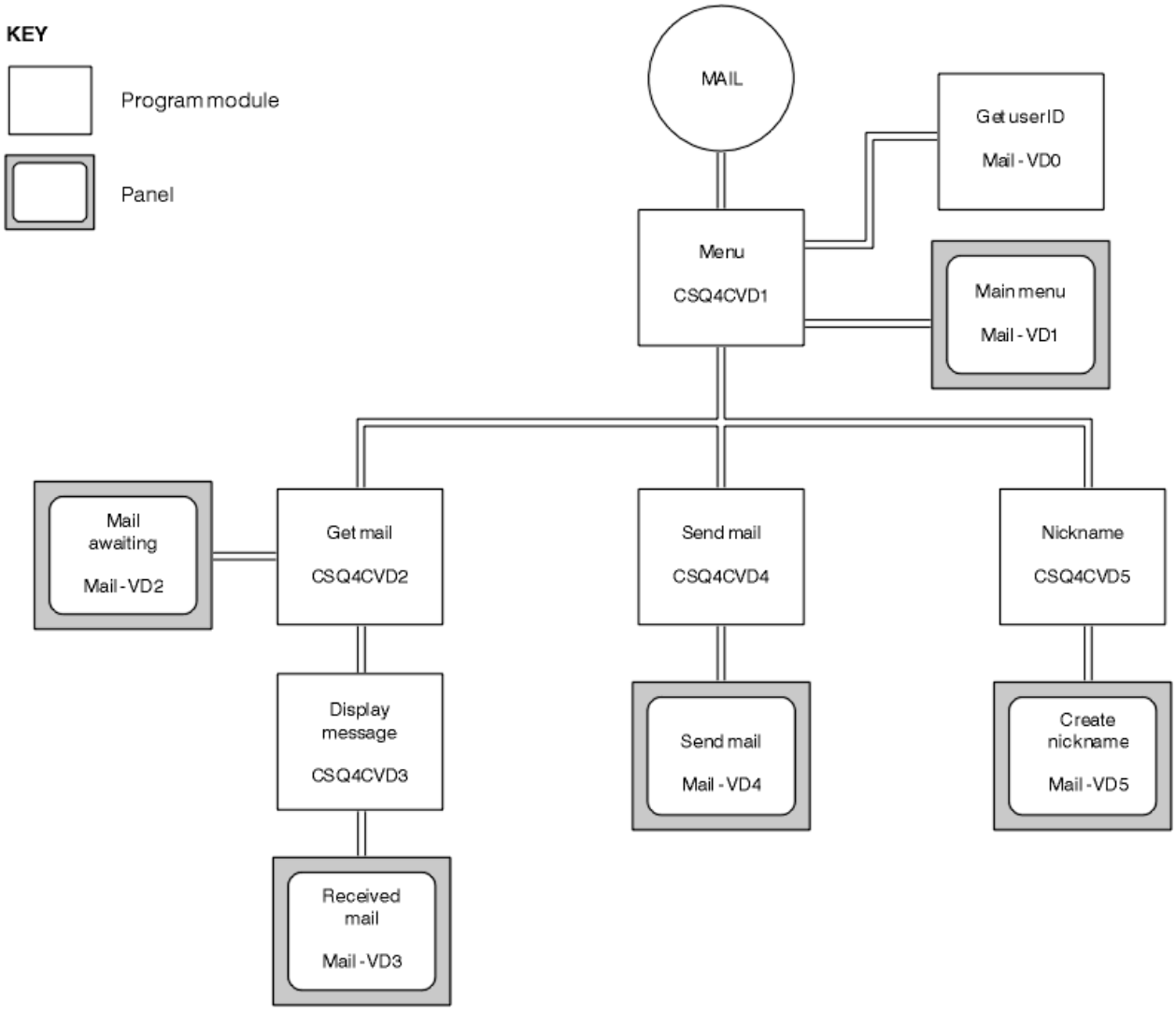


圖 *141:* 郵件管理程式 *CICS* 版本的程式及畫面

z/OS *z/OS* 上的功能表程式

在 TSO 環境中， CLIST 會呼叫功能表程式。 在 CICS 環境中，程式由交易 MAIL 呼叫。

功能表程式 (CSQ4TVD1 for TSO， CSQ4CVD1 for CICS) 是套組中的起始程式。 它會顯示功能表 (若為 TSO，則為 CSQ4VDP1，若為 CICS，則為 VD1)，並在從功能表中選取其他程式時呼叫它們。

程式會先取得使用者的 ID:

- 在程式的 CICS 版本中，如果使用者已登入 CICS，則會使用 CICS 指令 ASSIGN USERID 來取得使用者 ID。 如果使用者尚未登入，則程式會顯示登入畫面 (CSQ4VD0)，以提示使用者輸入使用者 ID。 此程式內沒有安全處理; 使用者可以提供任何使用者 ID。
- 在 TSO 版本中，使用者 ID 是從 CLIST 中的 TSO 取得。 它會以 ISPF 共用儲存區中的變數形式傳遞給功能表程式。

程式取得使用者 ID 之後，它會檢查以確定使用者具有郵件佇列 (CSQ4SAMP.MAILMGR。 *userid*)。 如果郵件佇列不存在，程式會透過將訊息放入系統指令輸入佇列來建立郵件佇列。 此訊息包含 IBM MQ for z/OS 指令 DEFINE QLOCAL。 此指令使用的物件定義會將佇列深度上限設為 9999 個訊息。

程式也會建立暫時動態佇列，以處理來自系統指令輸入佇列的回覆。 若要執行此動作，程式會使用 MQOPEN 呼叫並指定 SYSTEM.DEFAULT.MODEL.QUEUE 作為動態佇列的範本。 佇列管理程式會以字首為 CSQ4SAMP; 的名稱來建立暫時動態佇列; 其餘名稱由佇列管理程式產生。

然後，程式會開啟使用者的郵件佇列，並透過查詢佇列的現行深度來尋找佇列上的訊息數。 若要這樣做，程式會使用 MQINQ 呼叫，並指定 MQIA_CURRENT_Q_DEPTH 選取器。

然後，程式會執行迴圈，以顯示功能表並處理使用者所做的選擇。 當使用者按下 PF3 鍵時，會停止迴圈。 當進行有效的選擇時，會啟動適當的程式; 否則會顯示錯誤訊息。

z/OS *z/OS 上的 get-mail 及 display-message 程式*

在應用程式的 TSO 版本中， get-mail 及 display-message 功能是由相同的程式執行 (CSQ4TVD2)。 在 CICS 應用程式版本中，這些功能由個別程式 (CSQ4CVD2 和 CSQ4CVD3) 執行。

「郵件等待」畫面 (CSQ4VDP2 for TSO， VD2 for CICS ; 如需範例，請參閱 第 1003 頁的圖 142) 顯示使用者郵件佇列上的所有訊息。 若要建立此清單，程式會使用 MQGET 呼叫來瀏覽佇列上的所有訊息，並儲存每一個訊息的相關資訊。 除了顯示的資訊之外，程式還會記錄每一個訊息的 `MsgId` 和 `CorrelId` 。

```
---------------------- IBM MQ for z/OS Sample Programs ------- ROW 16 OF 29
COMMAND ==>                          Scroll  ===> PAGE
USERID - NTSFV02
Mail Manager System      QMGR  - VC4
Mail Awaiting

Msg   Mail    Date    Time
No    From    Sent    Sent
16
16   Deleted
17   JOHNJ   01/06/1993 12:52:02
18   JOHNJ   01/06/1993 12:52:02
19   JOHNJ   01/06/1993 12:52:03
20   JOHNJ   01/06/1993 12:52:03
21   JOHNJ   01/06/1993 12:52:03
22   JOHNJ   01/06/1993 12:52:04
23   JOHNJ   01/06/1993 12:52:04
24   JOHNJ   01/06/1993 12:52:04
25   JOHNJ   01/06/1993 12:52:05
26   JOHNJ   01/06/1993 12:52:05
27   JOHNJ   01/06/1993 12:52:05
28   JOHNJ   01/06/1993 12:52:06
29   JOHNJ   01/06/1993 12:52:06
```

圖 *142:* 顯示等待訊息清單的畫面範例

從「郵件等待」畫面中，使用者可以選取一則訊息，並顯示訊息的內容 (如需範例，請參閱 第 1004 頁的圖 143)。 程式會使用 MQGET 呼叫從佇列中移除此訊息，並使用該程式在瀏覽所有訊息時所記下的 `MsgId` 及 `CorrelId` 。 此 MQGET 呼叫是使用 MQGMO_SYNCPOINT 選項來執行。 程式會顯示訊息內容，然後宣告同步點: 這會確定 MQGET 呼叫，因此訊息現在不再存在。

```
--------------------- IBM MQ for z/OS Sample Programs ---------------------
COMMAND ==>
USERID - NTSFV02
Mail Manager System      QMGR  - VC4
Received Mail

Mail sent from JOHNJ  at VC4

Sent on the 01/06/1993 at 12:52:02
----------------------------------- Message ------------------------------
| HELLO FROM JOHNJ                                |
|                                     |
|                                     |
|                                     |
|                                     |
|                                     |
|                                     |
|                                     |
|                                     |
'-------------------------------------------------------------------------'
```

圖 *143:* 顯示訊息內容的畫面範例

「郵件管理程式」所提供之功能的明顯延伸，是讓使用者在檢視其內容之後，可以選擇將訊息留在佇列上。若要這樣做，在顯示訊息之後，您必須取消從佇列中移除訊息的 MQGET 呼叫。

z/OS *z/OS* 上的傳送郵件程式

當使用者完成「傳送郵件」畫面 (CSQ4VDP4 for TSO， VD4 for CICS) 時，傳送郵件程式 (CSQ4TVD4 for TSO， CSQ4CVD4 for CICS) 會將訊息放置在接收端的郵件佇列中。

為此，程式會使用 MQPUT1 呼叫。 訊息的目的地視使用者如何填寫「傳送郵件」畫面中的欄位而定:

- 如果使用者僅指定使用者名稱，則程式會先假設該名稱是暱稱，並將訊息傳送至該名稱所定義的物件。 如果沒有這樣的暱稱，程式會嘗試將訊息傳送至該名稱的本端佇列。
- 如果使用者同時指定使用者名稱及佇列管理程式名稱，則程式會將訊息傳送至由這兩個名稱定義的郵件佇列。

程式不接受空白訊息，且會從訊息文字的每一行移除前導空白。

如果 MQPUT1 呼叫成功，則程式會顯示一則訊息，指出放置訊息的使用者名稱及佇列管理程式名稱。 如果呼叫不成功，程式會特別檢查指出佇列或佇列管理程式不存在的原因碼; 這些是 MQRC_UNKNOWN_OBJECT_NAME 和 MQRC_UNKNOWN_OBJECT_Q_MGR。 對於每一個錯誤，程式會顯示自己的錯誤訊息; 對於其他錯誤，程式會顯示呼叫所傳回的完成及原因碼。

z/OS *z/OS* 上的暱稱程式

當使用者定義暱稱時，暱稱程式 (CSQ4TVD5 代表 TSO， CSQ4CVD5 代表 CICS) 會建立以暱稱作為其名稱一部分的佇列。

程式會在系統指令輸入佇列上放置訊息來執行此動作。 訊息包含 IBM MQ for z/OS 指令 DEFINE QALIAS 或 DEFINE QREMOTE。 程式建立的佇列類型取決於使用者如何填寫「建立暱稱」畫面 (CSQ4VDP5 for TSO， VD5 for CICS) 的欄位:

- 如果使用者只指定使用者名稱，或佇列管理程式名稱與郵件管理程式所連接的佇列管理程式名稱相同，則程式會建立別名佇列。
- 如果使用者同時指定使用者名稱及佇列管理程式名稱 (且佇列管理程式不是郵件管理程式所連接的佇列管理程式)，則程式會建立遠端佇列的本端定義。 程式不會檢查此定義所解析的佇列是否存在，甚至不會檢查遠端佇列管理程式是否存在。

程式也會建立暫時動態佇列，以處理來自系統指令輸入佇列的回覆。

如果佇列管理程式因程式預期的原因 (例如，佇列已存在) 而無法建立暱稱佇列，則程式會顯示自己的錯誤訊息。 如果佇列管理程式因程式無法預期的原因而無法建立佇列，則程式會顯示最多兩個由指令伺服器傳回給程式的錯誤訊息。

註: 對於每一個暱稱，暱稱程式只會建立遠端佇列的別名佇列或本端定義。 只有在使用暱稱所包含的使用者 ID 來啟動「郵件管理程式」應用程式時，才會建立這些佇列名稱所解析的本端佇列。

z/OS *z/OS* 上的信用檢查範例

「信用檢查」範例應用程式是一套程式，示範如何使用 IBM MQ for z/OS 所提供的許多特性。 它顯示應用程式有多少元件程式可以使用訊息佇列技術彼此傳遞訊息。

範例可以作為獨立式 CICS 應用程式執行。 不過，為了示範如何設計訊息佇列作業應用程式，以使用 CICS 及 IMS 環境所提供的機能，也會提供一個模組作為 IMS 批次訊息處理程式。 第 1013 頁的『z/OS 上 Credit Check 範例的 IMS 延伸』中說明此範例延伸。

您也可以在多個佇列管理程式上執行範例，並在每一個應用程式實例之間傳送訊息。 如果要執行這個動作，請參閱第 1013 頁的『在 z/OS 上具有多個佇列管理程式的「信用檢查」範例』。

CICS 程式以 C 和 COBOL 交付。 單一 IMS 程式僅在 C 中提供。 所提供的資料集顯示在 第 987 頁的表 182 和 第 989 頁的表 184 中。

此應用程式示範在銀行客戶要求貸款時評量風險的方法。 應用程式顯示銀行如何以兩種方式處理貸款申請:

- 直接與客戶交易時，銀行職員想要立即存取帳戶及信用風險資訊。
- 在處理書面申請時，銀行職員可以提交一系列的帳戶和信用風險資訊要求，稍後再處理回覆。

應用程式中的財務和安全詳細資料已保持簡單，因此訊息佇列技術是明確的。

z/OS 在 *z/OS* 上準備並執行「信用檢查」範例

若要準備並執行「信用檢查」範例，請執行下列步驟:

1. 建立 VSAM 資料集，以保留部分範例帳戶的相關資訊。 透過編輯並執行資料集 CSQ4FILE 中提供的 JCL 來執行此動作。
2. 執行第 985 頁的『在 z/OS 上為 CICS 環境準備範例應用程式』中的步驟。 (如果您想要對範例使用 IMS 延伸，則必須執行的其他步驟在 第 1013 頁的『z/OS 上 Credit Check 範例的 IMS 延伸』中說明。)
3. 啟動 CKTI 觸發監視器 (隨 IBM MQ for z/OS 提供) 針對佇列 CSQ4SAMP.INITIATION.QUEUE，使用 CICS 交易 CKQC。
4. 若要啟動應用程式，請啟動 CICS 系統並使用交易 MVB1。
5. 從第一個畫面選取 **立即** 或 **批次** 查詢。

   立即和批次查詢畫面類似; 第 1006 頁的圖 144 顯示「立即查詢」畫面。

```
CSQ4VB2       IBM MQ for z/OS Sample Programs

Credit Check - Immediate Inquiry

Specify details of the request, then press Enter.
Name . . . . . _____________________
Social security number ___ __ ____
Bank account name . . _______________________________
Account number . . . . __________
Amount requested . . . 012345
Response from CHECKING ACCOUNT for name : _____________________
Account information not found
Credit worthiness index - NOT KNOWN
..
..
..
..
..
..
..
..
..
MESSAGE LINE
F1=Help F3=Exit F5=Make another inquiry
```

圖 *144:*「信用檢查」範例應用程式的「立即查詢」畫面

6. 在適當的欄位中輸入帳號和貸款金額。 如需在這些欄位中輸入哪些資訊的指引，請參閱 第 1006 頁的『在查詢畫面中輸入資訊』。

在查詢畫面中輸入資訊

「信用檢查」範例應用程式會檢查您在查詢畫面的 **要求的金額** 欄位中輸入的資料是否為整數格式。

如果您輸入下列其中一個帳號，則應用程式會在 VSAM 資料集 CSQ4BAQ: 中尋找適當的帳戶名稱、平均帳戶餘額及信用額度索引:

- 2222222222
- 3111234329
- 3256478962
- 3333333333
- 3501676212
- 3696879656
- 4444444444
- 5555555555
- 6666666666
- 7777777777

您可以在其他欄位中輸入任何或否資訊。 應用程式會保留您輸入的任何資訊，並在它所產生的報告中傳回相同的資訊。

z/OS *z/OS 上 Credit Check 範例的設計*

本節說明構成「信用檢查」範例應用程式之每一個程式的設計。

如需在設計應用程式期間考量的部分技術的相關資訊，請參閱 第 1011 頁的『z/OS 上「信用檢查」範例的設計考量』。

第 1007 頁的圖 145 顯示組成應用程式的程式，以及這些程式所提供的佇列。 在此圖中，已從所有佇列名稱中省略字首 CSQ4SAMP，讓圖例更容易瞭解。

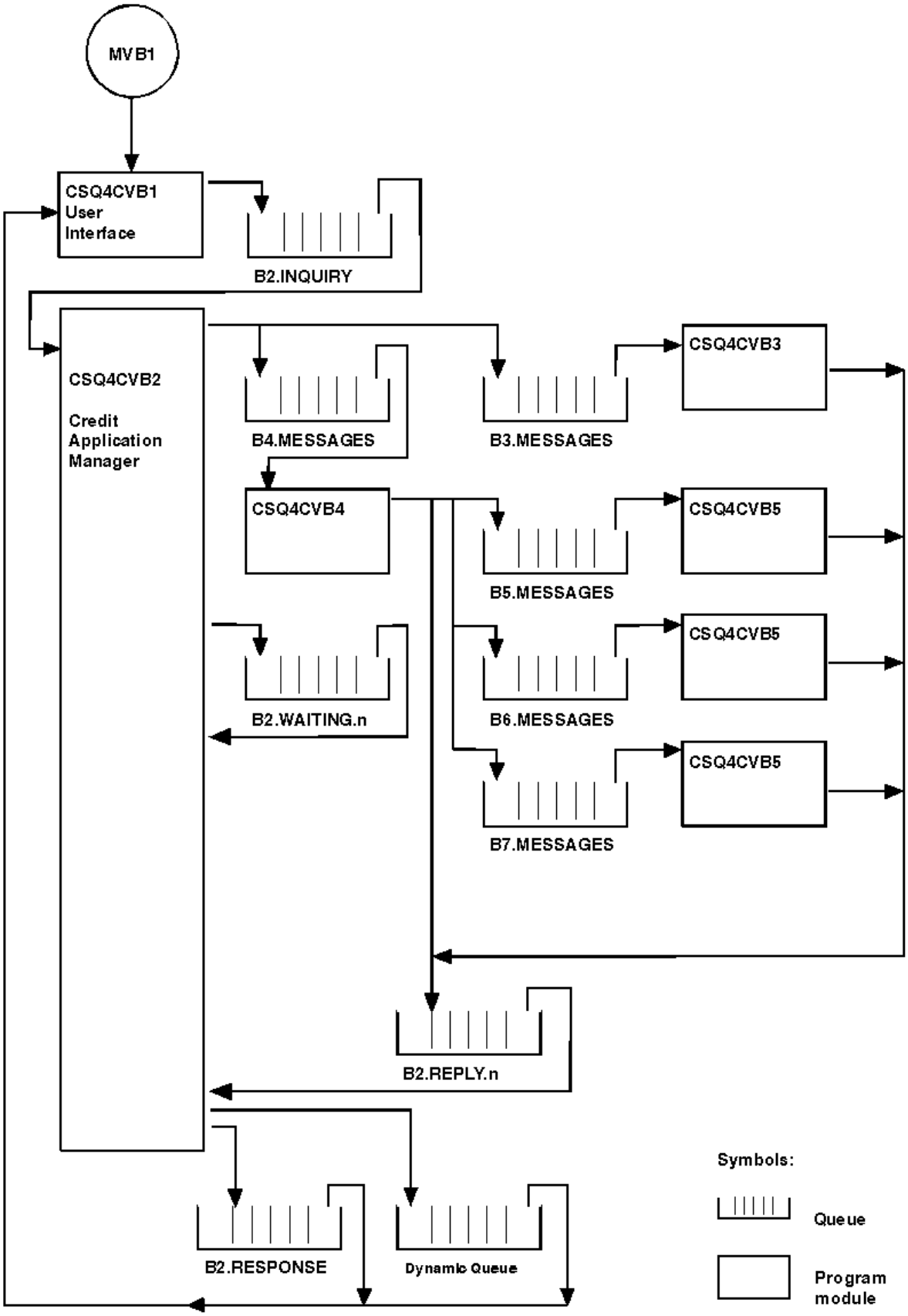


圖 *145: Credit Check* 範例應用程式的程式和佇列 *(*僅限 *COBOL* 程式*)*

z/OS *z/OS 上的使用者介面程式 (CSQ4CVB1)*

當您啟動交談模式 CICS 交易 MVB1 時，這會啟動應用程式的使用者介面程式。

此程式會將查詢訊息放入佇列 CSQ4SAMP.B2.INQUIRY ，並從其指定何時進行查詢的回覆目的地佇列中取得對那些查詢的回覆。 從使用者介面中，您可以提交立即或批次查詢:

- 若為立即查詢，程式會建立一個暫時動態佇列，以作為回覆目的地佇列。 這表示每一個查詢都有自己的回覆目的地佇列。
- 若為批次查詢，使用者介面程式會從佇列 CSQ4SAMP.B2.RESPONSE。 為了簡單起見，程式會從這個回覆目的地佇列取得其所有查詢的回覆。 很容易看出銀行可能想要對 MVB1 的每一個使用者使用個別的回覆目的地佇列，因此他們每一個都只能看到他們所起始之查詢的回覆。

當處於批次和立即模式時，應用程式中所使用訊息的內容之間的重要差異如下:

- 對於批次工作，訊息具有低優先順序，因此會在以立即模式輸入的任何貸款要求之後處理它們。 此外，訊息是持續的，因此如果應用程式或佇列管理程式必須重新啟動，則會回復這些訊息。
- 為了立即運作，訊息具有高優先順序，因此在以批次模式輸入的任何貸款要求之前會先處理它們。 此外，訊息不是持續的，因此如果應用程式或佇列管理程式必須重新啟動，則會捨棄它們。

不過，在所有情況下，貸款申請訊息的內容會延伸到整個應用程式。 因此，例如，從高優先順序要求產生的所有訊息也將具有高優先順序。

z/OS *z/OS 上的額度應用程式管理程式 (CSQ4CVB2)*

Credit Application Manager (CAM) 程式會執行「信用檢查」應用程式的大部分處理。

當佇列 CSQ4SAMP.B2.INQUIRY 或佇列 CSQ4SAMP.B2.REPLY 上發生觸發事件時， CAM 會由 CKTI 觸發監視器 (隨 IBM MQ for z/OS 提供) 啟動。 *n*，其中 *n* 是識別一組回覆佇列之一的整數。 觸發訊息包含的資料包括發生觸發事件的佇列名稱。

CAM 使用名稱格式為 CSQ4SAMP.B2.WAITING.n ，以儲存其正在處理之查詢的相關資訊。 佇列會命名，以便每一個佇列都與回覆目的地佇列配對; 例如，佇列 CSQ4SAMP.B2.WAITING.3 包含特定查詢的輸入資料，以及佇列 CSQ4SAMP.B2.REPLY.3 包含一組與該相同查詢相關的回覆訊息 (來自查詢資料庫的程式)。 若要瞭解此設計背後的原因，請參閱 第 1012 頁的『CAM 中的個別查詢和回覆佇列』。

啟動邏輯

如果觸發事件發生在佇列 CSQ4SAMP.B2.INQUIRY， CAM 會開啟佇列以進行共用存取。 然後，它會嘗試開啟每一個回覆佇列，直到找到可用的回覆佇列為止。 如果找不到免費回覆佇列， CAM 會記載事實並正常終止。

如果觸發事件發生在佇列 CSQ4SAMP.B2.REPLY.n， CAM 會開啟佇列以進行互斥存取。 如果回覆碼報告物件已在使用中， CAM 會正常終止。 如果發生任何其他錯誤， CAM 會記載錯誤並終止。 CAM 會開啟對應的等待佇列和查詢佇列，然後開始取得和處理訊息。 從等待佇列中， CAM 會回復局部完成之查詢的詳細資料。

為了簡單起見，在此範例中，使用的佇列名稱會保留在程式中。 在商業環境中，佇列名稱可能保留在程式所存取的檔案中。

從查詢佇列取得訊息

CAM 會先嘗試使用 MQGET 呼叫搭配 MQGMO_SET_SIGNAL 選項，從查詢佇列取得訊息。 如果訊息立即可用，則會處理訊息; 如果沒有可用的訊息，則會設定信號。

然後， CAM 會再次嘗試使用 MQGET 呼叫搭配相同的選項，從回覆佇列取得訊息。 如果訊息立即可用，則會處理訊息; 否則會設定信號。

當兩個信號都已設定時，程式會等待直到其中一個信號被張貼。 如果張貼信號以指出訊息可用，則會擷取並處理訊息。 如果信號到期或佇列管理程式正在終止，則程式會終止。

處理 CAM 所擷取的訊息

CAM 所擷取的訊息可以是四種類型之一:

- 查詢訊息
- 回覆訊息
- 傳送訊息
- 非預期或不想要的訊息

CAM 會依照 第 1009 頁的『在 z/OS 上處理 CAM 所擷取的訊息』中的說明來處理這些訊息。

傳送回答

當 CAM 收到它預期進行查詢的所有回覆時，它會處理回覆並建立單一回應訊息。 它會將所有回覆訊息中具有相同 `CorrelId` 的所有資料合併成一則訊息。 此回應會放在原始貸款申請中指定的回覆目的地佇列上。 回應訊息放置在包含擷取最終回覆訊息的相同工作單元內。 這是為了簡化回復，方法是確保佇列 CSQ4SAMP.B2.WAITING.n。

恢復部分完成的查詢

CAM 會複製到佇列 CSQ4SAMP.B2.WAITING.n 它接收的所有訊息。 它會設定訊息描述子的欄位，如下所示:

- *`Priority`* 取決於訊息類型:
  - 若為要求訊息，優先順序 = 3
  - 若為資料包，優先順序 = 2
  - 若為回覆訊息，優先順序 = 1
- *`CorrelId`* 設為貸款申請訊息的 *`MsgId`*
- 其他 MQMD 欄位是從已接收訊息的那些 MQMD 欄位複製而來

當查詢已完成時，在回答處理期間，會從等待佇列中移除特定查詢的訊息。 因此，在任何時間，等待佇列都會包含與進行中查詢相關的所有訊息。 如果程式必須重新啟動，則會使用這些訊息來回復進行中查詢的詳細資料。 會設定不同的優先順序，以便在傳送或回覆訊息之前回復查詢訊息。

z/OS 在 *z/OS* 上處理 *CAM* 所擷取的訊息

Credit Application Manager (CAM) 所擷取的訊息可以是四種類型之一。 CAM 處理訊息的方式取決於其類型。

CAM 所擷取的訊息可以是四種類型之一:

- 查詢訊息
- 回覆訊息
- 傳送訊息
- 非預期或不想要的訊息

CAM 會依照下列方式來處理這些訊息:

查詢訊息

查詢訊息來自使用者介面程式。 它會針對每一個貸款申請建立查詢訊息。

對於所有貸款申請， CAM 會要求客戶支票帳戶的平均餘額。 作法是在別名佇列 CSQ4SAMP.B2.OUTPUT.ALIAS。 此佇列名稱解析為佇列 CSQ4SAMP.B3.MESSAGES，由 checking-account 程式 CSQ4CVB3 處理。 當 CAM 將訊息放入這個別名佇列時，它會指定適當的 CSQ4SAMP.B2.REPLY.n 佇列。 這裡使用別名佇列，以便程式 CSQ4CVB3 可以輕易被另一個處理不同名稱基本佇列的程式所取代。 若要這樣做，您可以重新定義別名佇列，使其名稱解析為新佇列。 此外，您可以對別名佇列及基本佇列指派不同的存取權。

如果使用者要求大於 10000 個單位的貸款， CAM 也會對其他資料庫起始檢查。 作法是將要求訊息放在佇列 CSQ4SAMP.B4.MESSAGES，由配送程式 CSQ4CVB4 處理。 負責處理此佇列的程序會將訊息傳播至程式所提供的佇列，這些程式可存取其他記錄，例如信用卡歷程、儲蓄帳戶及抵押貸款付款。 這些程式的資料會傳回至放置作業中指定的回覆目的地佇列。 此外，此程式會將傳送訊息傳送至回覆目的地佇列，以指定已傳送多少傳送訊息。

在商業環境中，配送程式可能會重新格式化所提供的資料，以符合每一種其他類型的銀行帳戶所需要的格式。

所參照的任何佇列都可以在遠端系統上。

對於每一個查詢訊息， CAM 會在記憶體常駐「查詢記錄表 (IRT)」中起始一個項目。 此記錄包含:

- 查詢訊息的 `MsgId`
- 在 ReplyExp 欄位中，預期的回應數 (等於傳送的訊息數)
- 在 ReplyRec 欄位中，已收到的回覆數 (在此階段為零)
- 在 PropsOut 欄位中，指出是否預期傳送訊息

CAM 會使用下列指令，將查詢訊息複製到等待佇列:

- `Priority` 設為 3
- `CorrelId` 設為查詢訊息的 `MsgId`
- 其他訊息描述子欄位設為查詢訊息的那些訊息描述子欄位

傳播訊息

傳送訊息包含配送程式已轉遞查詢的佇列數。 訊息處理方式如下:

1. 將傳送的訊息數新增至 IRT 中適當記錄的 ReplyExp 欄位。 此資訊位於訊息中。
2. 以 IRT 中記錄的 ReplyRec 欄位 1 為增量。
3. 減少 1 IRT 中記錄的 PropsOut 欄位。
4. 將訊息複製到等待佇列。 CAM 會將 `Priority` 設為 2 ，並將訊息描述子的其他欄位設為傳播訊息的那些欄位。

回覆訊息 (reply message)

回覆訊息包含對支票帳戶程式或其中一個代理查詢程式的其中一個要求的回應。 回覆訊息的處理方式如下:

1. 以 IRT 中記錄的 ReplyRec 欄位 1 為增量。
2. 將訊息複製到等待佇列中，並將 `Priority` 設為 1 ，並將訊息描述子的其他欄位設為回覆訊息的欄位。
3. 如果 ReplyRec = ReplyExp，且 PropsOut = 0 ，請設定 MsgComplete 旗標。

其他訊息

應用程式不預期其他訊息。 不過，應用程式可能會接收系統播送的訊息，或以不明 `CorrelIds` 來回覆訊息。

CAM 會將這些訊息放入佇列 CSQ4SAMP.DEAD.QUEUE，可在其中檢查它們。 如果此放置作業失敗，則訊息會遺失，且程式會繼續執行。 如需此部分程式設計的相關資訊，請參閱 第 1012 頁的『範例如何處理非預期的訊息』。

z/OS *z/OS 上的檢查帳戶程式 (CSQ4CVB3)*

佇列 CSQ4SAMP.B3.MESSAGES。 開啟佇列之後，此程式會使用 MQGET 呼叫搭配等待選項，並將等待間隔設為 30 秒，從佇列中取得訊息。

程式會在 VSAM 資料集 CSQ4BAQ 中搜尋貸款申請訊息中的帳號。 它會擷取對應的帳戶名稱、平均餘額及信用額度索引，或帳號不在資料集中的附註。

然後，程式會將回覆訊息 (使用 MQPUT1 呼叫) 放置在貸款要求訊息中指名的回覆目的地佇列上。 對於此回覆訊息，程式:

- 複製貸款申請訊息的 `CorrelId`
- 使用 MQPMO_PASS_IDENTITY_CONTEXT 選項

程式會繼續從佇列取得訊息，直到等待間隔到期為止。

z/OS *z/OS 上的配送程式 (CSQ4CVB4)*

佇列 CSQ4SAMP.B4.MESSAGES。

為了模擬將貸款申請配送至可存取記錄 (例如信用卡歷程、儲蓄帳戶及抵押貸款付款) 的其他機構，程式會在名單 CSQ4SAMP.B4.NAMELIST。 這些佇列有三個，名稱格式為 CSQ4SAMP.B *n*。MESSAGES，其中 *n* 是 5、6 或 7。 在商業應用程式中，代理機構可能位於不同的位置，因此這些佇列可能是遠端佇列。 如果您要修改範例應用程式以顯示此項目，請參閱 第 1013 頁的『在 z/OS 上具有多個佇列管理程式的「信用檢查」範例』。

配送程式會執行下列步驟:

1. 從名單中，取得程式要使用的佇列名稱。 程式會使用 MQINQ 呼叫來查詢名稱清單物件的屬性，以執行此動作。
2. 開啟這些佇列以及 CSQ4SAMP.B4.MESSAGES。
3. 執行下列迴圈，直到佇列 CSQ4SAMP.B4.MESSAGES:
   a. 搭配使用 MQGET 呼叫與等待選項，並將等待間隔設為 30 秒，以取得訊息。
   b. 將訊息放置在名單中列出的每一個佇列上，並指定適當 CSQ4SAMP.B2.REPLY.n 佇列。 程式會將貸款要求訊息的 *CorrelId* 複製到這些複製訊息，並在 MQPUT 呼叫上使用 MQPMO_PASS_IDENTITY_CONTEXT 選項。
   c. 將資料封包訊息傳送至佇列 CSQ4SAMP.B2.REPLY.n，顯示它已順利放置的訊息數。
   d. 宣告同步點。

z/OS *z/OS 上的機構查詢程式 (CSQ4CVB5/CSQ4CCB5)*

提供代理查詢程式作為 COBOL 程式及 C 程式。 兩個程式都有相同的設計。 這顯示不同類型的程式可以輕易地同時存在於 IBM MQ 應用程式內，且組成這類應用程式的程式模組可以輕易地更換。

程式的實例由下列任何佇列上的觸發事件啟動:

- 若為 COBOL 程式 (CSQ4CVB5):
  - CSQ4SAMP.B5.MESSAGES
  - CSQ4SAMP.B6.MESSAGES
  - CSQ4SAMP.B7.MESSAGES
- 對於 C 程式 (CSQ4CCB5)，佇列為 CSQ4SAMP.B8.MESSAGES

註: 如果您要使用 C 程式，則必須變更名單 CSQ4SAMP.B4.NAMELIST 的定義，以將佇列 CSQ4SAMP.B7.MESSAGES 取代為 CSQ4SAMP.B8.MESSAGES。 若要執行此動作，您可以使用下列任何一項:

- IBM MQ for z/OS 作業及控制面板
- ALTER NAMELIST 指令
- CSQUTIL 公用程式

在開啟適當的佇列之後，此程式會使用 MQGET 呼叫與等待選項，並將等待間隔設為 30 秒，從佇列中取得訊息。

此程式會透過在 VSAM 資料集 CSQ4BAQ 中搜尋在貸款要求訊息中傳遞的帳號，來模擬搜尋機構的資料庫。 然後，它會建置一個回覆，其中包含它所提供的佇列名稱及信譽索引。 為了簡化處理程序，會隨機選取信譽指數。

放置回覆訊息時，程式會使用 MQPUT1 呼叫，並執行下列動作:

- 複製貸款申請訊息的 `CorrelId`
- 使用 MQPMO_PASS_IDENTITY_CONTEXT 選項

程式會將回覆訊息傳送至貸款要求訊息中指名的回覆目的地佇列。 (貸款要求訊息中也指定擁有回覆目的地佇列的佇列管理程式名稱。)

z/OS *z/OS 上「信用檢查」範例的設計考量*

「信用檢查」範例的設計考量。

本主題包含下列相關資訊:

CAM 中的個別查詢和回覆佇列

應用程式可以使用單一佇列來進行查詢及回覆，但由於下列原因而設計使用個別佇列:

- 當程式處理查詢數目上限時，可以將進一步查詢留在佇列中。 如果正在使用單一佇列，則必須從佇列中取出並儲存在其他位置。
- 如果訊息資料流量足夠高，可自動啟動 CAM 的其他實例，以處理相同的查詢佇列。 但程式必須追蹤進行中的查詢，若要執行此動作，它必須將已起始之查詢的所有回覆傳回。 如果只使用一個佇列，則程式必須瀏覽這些訊息，以查看它們是否適用於此程式或另一個程式。 這將大大降低作業的效率。

  應用程式可以支援多個 CAMS ，並且可以使用成對的回覆和等待佇列來有效地回復進行中的查詢。
- 程式可以使用信號在多個佇列上有效地等待。

範例如何處理錯誤

使用者介面程式會直接向使用者報告錯誤，以處理錯誤。

其他程式沒有使用者介面，因此它們必須以其他方式處理錯誤。 此外，在許多情況下 (例如，如果 MQGET 呼叫失敗) ，這些其他程式並不知道應用程式使用者的身分。

其他程式會將錯誤訊息放置在稱為 CSQ4SAMP 的 CICS 暫時儲存體佇列上。 您可以使用 CICS 提供的交易 CEBR 來瀏覽此佇列。 程式也會將錯誤訊息寫入 CICS CSML 日誌。

範例如何處理非預期的訊息

當您設計訊息佇列作業應用程式時，必須決定如何非預期地處理抵達佇列的訊息。

兩個基本選擇是:

- 除非應用程式已處理非預期的訊息，否則它不會再運作。 這可能表示應用程式會通知操作員，自行終止，並確保不會自動重新啟動它 (它可以透過將觸發設為關閉來執行此動作)。 此選項表示應用程式的所有處理都可以由單一非預期的訊息中止，且需要操作員介入才能重新啟動應用程式。
- 應用程式會從它所提供的佇列中移除訊息，將訊息放置在另一個位置，並繼續處理。 放置此訊息的最佳位置是在系統無法傳送郵件的佇列上。

如果您選擇第二個選項:

- 操作員或另一個程式應該檢查放置在無法傳送郵件的佇列上的訊息，以找出訊息來自何處。
- 如果無法將非預期的訊息放置在無法傳送郵件的佇列上，則會遺失非預期的訊息。
- 如果長非預期的訊息超過無法傳送郵件之佇列上的訊息限制，或超過程式中的緩衝區大小，則會截斷長非預期的訊息。

為了確保應用程式順利處理來自外部活動的所有查詢， Credit Check 範例應用程式會使用第二個選項。 為了讓範例與使用相同佇列管理程式的其他應用程式分開，「信用檢查」範例不會使用系統無法傳送郵件的佇列; 而是使用自己的無法傳送郵件的佇列。 此佇列名為 CSQ4SAMP.DEAD.QUEUE。 範例會截斷任何長於提供給範例程式之緩衝區的訊息。 您可以使用「瀏覽」範例應用程式來瀏覽此佇列上的訊息，或使用「列印訊息」範例應用程式來列印訊息及其訊息描述子。

不過，如果您延伸範例以跨多個佇列管理程式執行，則佇列管理程式可能會將非預期的訊息或無法遞送的訊息放置在系統無法傳送郵件的佇列上。

範例如何使用同步點

「信用檢查」範例應用程式中的程式會宣告同步點，以確保:

- 僅傳送一個回覆訊息以回應每一個預期訊息
- 永不將非預期訊息的多個副本放置在範例的無法傳送郵件的佇列上
- CAM 可以從其等待佇列取得持續訊息，以回復所有局部完成之查詢的狀態

為了達到此目的，會使用單一工作單元來涵蓋取得訊息、處理該訊息，以及任何後續放置作業。

範例如何使用訊息環境定義資訊

當使用者介面程式 (CSQ4CVB1) 傳送訊息時，它會使用 MQPMO_DEFAULT_CONTEXT 選項。 這表示佇列管理程式會同時產生身分及原始環境定義資訊。 佇列管理程式會從啟動程式的交易 (MVB1) 及啟動交易的使用者 ID 取得此資訊。

當 CAM 傳送查詢訊息時，它會使用 MQPMO_PASS_IDENTITY_CONTEXT 選項。 這表示將從原始查詢訊息的身分環境定義複製所放置訊息的身分環境定義資訊。 使用此選項，佇列管理程式會產生原始環境定義資訊。

當 CAM 傳送回覆訊息時，它會使用 MQPMO_ALTERNATE_USER_AUTHORITY 選項。 當 CAM 開啟回覆目的地佇列時，這會導致佇列管理程式使用替代使用者 ID 進行安全檢查。 CAM 會使用原始查詢訊息之提交者的使用者 ID。 這表示只容許使用者查看其起始之查詢的回覆。 從原始查詢訊息的訊息描述子中的身分環境定義資訊取得替代使用者 ID。

當查詢程式 (CSQ4CVB3/4/5) 傳送回覆訊息時，它們會使用 MQPMO_PASS_IDENTITY_CONTEXT 選項。 這表示將從原始查詢訊息的身分環境定義複製所放置訊息的身分環境定義資訊。 使用此選項，佇列管理程式會產生原始環境定義資訊。

註: 與 MVB3/4/5 交易相關聯的使用者 ID 需要存取 B2.REPLY.n 佇列。 這些使用者 ID 可能與正在處理之要求相關聯的使用者 ID 不同。 為了避免這種可能的安全漏洞，查詢程式在放置其回覆時可以使用 MQPMO_ALTERNATE_USER_AUTHORITY 選項。 這表示 MVB1 的每一個個別使用者都需要權限才能開啟 B2.REPLY.n 佇列。

在 CAM 中使用訊息和相關性 ID

應用程式必須隨時監視它正在處理的所有即時查詢的進度。 為了執行此動作，它會使用每一個貸款申請訊息的唯一訊息 ID，來關聯它所擁有的每一個查詢的所有相關資訊。

CAM 會將查詢訊息的 `MsgId` 複製到針對該查詢所傳送之所有要求訊息的 `CorrelId`。 範例 (CSQ4CVB3-5) 中的其他程式會將它們收到的每一則訊息的 `CorrelId` 複製到其回覆訊息的 `CorrelId` 中。

z/OS 在 *z/OS* 上具有多個佇列管理程式的「信用檢查」範例

您可以使用「信用檢查」範例應用程式，透過在兩個佇列管理程式及 CICS 系統 (每一個佇列管理程式都連接至不同的 CICS 系統) 上安裝範例，來示範分散式佇列作業。

當安裝範例程式，且觸發監視器 (CKTI) 在每一個系統上執行時，您需要:

1. 設定兩個佇列管理程式之間的通訊鏈結。 如需如何執行此動作的相關資訊，請參閱 配置分散式佇列作業。
2. 在一個佇列管理程式上，為您要使用的每一個遠端佇列 (在另一個佇列管理程式上) 建立本端定義。 這些佇列可以是任何 CSQ4SAMP.B *n*。MESSAGES，其中 *n* 是 3、5、6 或 7。 (這些是檢查帳戶程式及代理查詢程式所提供的佇列。) 如需如何執行此動作的相關資訊，請參閱 DEFINE QREMOTE 及 DEFINE 佇列。
3. 變更名單 (CSQ4SAMP.B4.NAMELIST)，讓它包含您要使用的遠端佇列名稱。 如需如何執行此動作的相關資訊，請參閱 DEFINE NAMELIST。

z/OS *z/OS* 上 *Credit Check* 範例的 *IMS* 延伸

以 IMS 批次訊息處理 (BMP) 程式來提供核對帳戶程式的版本。 它以 C 語言撰寫。

程式執行與 CICS 版本相同的功能，但為了取得帳戶資訊，程式會讀取 IMS 資料庫而非 VSAM 檔案。 如果您將核對帳戶程式的 CICS 版本取代為 IMS 版本，則在使用應用程式的方法中看不到任何差異。

若要準備並執行 IMS 版本，您必須:

1. 遵循第 1005 頁的『在 z/OS 上準備並執行「信用檢查」範例』中的步驟進行。
2. 遵循第 988 頁的『在 z/OS 上為 IMS 環境準備範例應用程式』中的步驟進行。
3. 變更別名佇列 CSQ4SAMP.B2.OUTPUT.ALIAS 可解析為佇列 CSQ4SAMP.B3.IMS。MESSAGES (而非 CSQ4SAMP.B3.MESSAGES)。 若要這樣做，您可以使用下列其中一項:
   - IBM MQ for z/OS 作業及控制面板
   - ALTER QALIAS 指令。

使用 IMS 檢查帳戶程式的另一種方式是讓它為從配送程式接收訊息的其中一個佇列提供服務。 在「信用檢查」範例應用程式的遞送表單中，有三個佇列 (B5/6/7.MESSAGES) ，全部由機構查詢程式提供。 此程式會搜尋 VSAM 資料集。 若要比較 VSAM 資料集與 IMS 資料庫的使用，您可以改為讓 IMS 檢查帳戶程式提供其中一個佇列。 若要這樣做，您必須變更名單 CSQ4SAMP.B4.NAMELIST 用來取代其中一個 CSQ4SAMP.B *n*。具有 CSQ4SAMP.B3.IMS 的 MESSAGES 佇列。MESSAGES 佇列。 您可以使用下列其中一項:

- IBM MQ for z/OS 作業及控制面板
- ALTER NAMELIST 指令。

然後，您可以從 CICS 交易 MVB1 執行範例。 使用者在作業或回應中看不到任何差異。 IMS BMP 會在收到停止訊息或處於非作用中狀態五分鐘之後停止。

IMS 檢查帳戶程式的設計 (CSQ4ICB3)

此程式以 BMP 方式執行。 在將任何 IBM MQ 訊息傳送至程式之前，請先使用其 JCL 來啟動該程式。

程式會在 IMS 資料庫中搜尋貸款申請訊息中的帳號。 它會擷取對應的帳戶名稱、平均餘額及信用狀況指數。

程式會將資料庫搜尋的結果傳送至正在處理的 IBM MQ 訊息中所指名的回覆目的地佇列。 傳回的訊息會將帳戶類型及搜尋結果附加至接收的訊息，以便建置回應的交易可以確認正在處理正確的查詢。 訊息採用三個 79 個字元群組的形式，如下所示:

```
'Response from CHECKING ACCOUNT for name : JONES J B'
'   Opened 870530, 3-month average balance = 000012.57'
'   Credit worthiness index - BBB'
```

以訊息導向 BMP 執行時，程式會排除 IMS 訊息佇列，然後從 IBM MQ for z/OS 佇列讀取訊息並處理它們。 未從 IMS 訊息佇列收到任何資訊。 程式會在每一個檢查點之後重新連接至佇列管理程式，因為控點已關閉。

在批次導向 BMP 中執行時，程式會在每一個檢查點之後繼續連接至佇列管理程式，因為控點未關閉。

z/OS *z/OS* 上的「訊息處理程式」範例

「訊息處理程式」範例 TSO 應用程式可讓您瀏覽、轉遞及刪除佇列上的訊息。 範例在 C 和 COBOL 中提供。

準備並執行範例

請遵循下列步驟:

1. 準備範例，如第 984 頁的『在 z/OS 上為 TSO 環境準備範例應用程式』中所述。
2. 自訂為範例提供的 CLIST (CSQ4RCH1) ，以定義畫面的位置、訊息檔案的位置及載入模組的位置。

您可以使用 CLIST CSQ4RCH1 來執行範例的 C 及 COBOL 版本。 所提供的 CSQ4RCH1 版本會執行 C 版本，並包含 COBOL 版本自訂所需的指示。

註:

1. 範例沒有隨附任何範例佇列定義。

2. VS COBOL II 不支援具有 ISPF 的多工作業，因此請勿在分割畫面的兩端使用「訊息處理程式」範例應用程式。 如果您這麼做，結果將無法預期。

z/OS 在 *z/OS* 上使用「訊息處理程式」範例

安裝範例並從自訂的 CLIST CSQ4RCH1 呼叫它之後，即會顯示 第 1015 頁的圖 146 中顯示的畫面。

```
----------------------- IBM MQ for z/OS -- Samples ------------------------
COMMAND ===>
User Id : JOHNJ



Enter information. Press ENTER :


Queue Manager Name  : ________________________________________________ :

Queue Name      : ________________________________________________ :









F1=HELP   F2=SPLIT   F3=END    F4=RETURN  F5=RFIND   F6=RCHANGE
F7=UP     F8=DOWN    F9=SWAP   F10=LEFT   F11=RIGHT  F12=RETRIEVE
```

圖 *146:*「訊息處理程式」範例的起始畫面

輸入要檢視的佇列管理程式及佇列名稱 (區分大小寫)，即會顯示訊息清單畫面 (請參閱 第 1015 頁的圖 147)。

```
----------------------- IBM MQ for z/OS -- Samples ------- Row 1 to 4 of 4
COMMAND ==>

Queue Manager  : VM03                        :
Queue      : MQEI.IMS.BRIDGE.QUEUE              :

Message number  01 of 04

Msg  Put Date  Put Time Format   User   Put Application
No  MM/DD/YYYY HH:MM:SS Name  Identifier  Type   Name
01  10/16/1998 13:51:19 MQIMS  NTSFV02   00000002 NTSFV02A
02  10/16/1998 13:55:45 MQIMS  JOHNJ    00000011 EDIT\CLASSES\BIN\PROGTS
03  10/16/1998 13:54:01 MQIMS  NTSFV02   00000002 NTSFV02B
04  10/16/1998 13:57:22 MQIMS  johnj    00000011 EDIT\CLASSES\BIN\PROGTS
****************************** Bottom of data ********************************
```

圖 *147:*「訊息處理程式」範例的訊息清單畫面

此畫面會顯示佇列上的前 99 則訊息，並針對每則訊息顯示下列欄位:

訊息號碼
訊息號碼

放置日期 MM/DD/YYYY
將訊息放入佇列的日期 (GMT)

放置時間 HH:MM: SS
將訊息放入佇列的時間 (GMT)

格式名稱
MQMD.Format 欄位

使用者 ID
MQMD.UserIdentifier 欄位

放置應用程式類型
MQMD.PutApplType 欄位

放置應用程式名稱
MQMD.PutApplName 欄位

也會顯示佇列上的訊息總數。

從這個畫面中，可以選擇訊息，依號碼而非游標位置，然後顯示。 如需範例，請參閱第 1016 頁的圖 148。

```
---------------------- IBM MQ for z/OS -- Samples ----- Row 1 to 35 of 35
COMMAND ==>

Queue Manager  : VM03                        :
Queue      : MQEI.IMS.BRIDGE.QUEUE               :
Forward to Q Mgr : VM03                       :
Forward to Queue : QL.TEST.ISCRES1                 :

Action : _ :  (D)elete (F)orward

Message Content :
------------------------------------------------------------------------------
Message Descriptor
StrucId     : `MD `
Version     : 000000001
Report      : 000000000
MsgType     : 000000001
Expiry      : -00000001
Feedback     : 000000000
Encoding     : 000000785
CodedCharSetId  : 000000500
Format      : `MQIMS  `
Priority     : 000000000
Persistence    : 000000001
MsgId      : `C3E2D840E5D4F0F34040404040404040AF6B30F0A89B7605`X
CorrelId     : `000000000000000000000000000000000000000000000000`X
BackoutCount   : 000000000
ReplyToQ     : `QL.TEST.ISCRES1                `
ReplyToQMgr   : `VM03                        `
UserIdentifier  : `NTSFV02   `
AccountingToken :
`06F2F5F5F3F0F1000000000000000000000000000000000000000000000000`X
ApplIdentityData : `                    `
PutApplType   : 000000002
PutApplName   : `NTSFV02A          `
PutDate     : `19971016`
PutTime     : `13511903`
ApplOriginData  : `   `

Message Buffer :  108 byte(s)
00000000 : C9C9 C840 0000 0001 0000 0054 0000 0311 `IIH .....T......`
00000010 : 0000 0000 4040 4040 4040 4040 0000 0000 `....    ....`
00000020 : 4040 4040 4040 4040 4040 4040 4040 4040 `                `
00000030 : 4040 4040 4040 4040 4040 4040 4040 4040 `                `
00000040 : 0000 0000 0000 0000 0000 0000 0000 0000 `................`
00000050 : 40F1 C300 0018 0000 C9C1 D7D4 C4C9 F2F8 ` 1C.....IAPMDI28`
00000060 : 40C8 C5D3 D3D6 40E6 D6D9 D3C4      ` HELLO WORLD  `
****************************** Bottom of data ********************************
```

圖 *148:* 顯示選擇的訊息

一旦顯示訊息，就可以將它刪除、留在佇列中，或轉遞至另一個佇列。 `Forward to Q Mgr` 和 `Forward to Queue` 欄位會以 MQMD 的值來起始設定，在轉遞訊息之前可以變更這些值。

範例設計只容許選取及顯示具有唯一 `MsgId / CorrelId` 組合的訊息，因為會以 `MsgId` 和 `CorrelId` 作為索引鍵來擷取訊息。 如果索引鍵不是唯一的，則範例無法確實擷取所選擇的訊息。

註: 當您使用 SCSQCLST (CSQ4RCH1) 範例來瀏覽訊息時，每次呼叫都會導致訊息的取消計數增加。 如果您想要變更此範例的行為，請複製範例並視需要修改內容。 您應該注意，依賴此取消計數的其他應用程式可能會受到此遞增計數的影響。

z/OS *z/OS* 上範例「訊息處理程式」範例的設計

本主題說明組成「訊息處理程式」範例應用程式的每一個程式的設計。

物件驗證程式

這會要求有效的佇列及佇列管理程式名稱。

如果您未指定佇列管理程式名稱，則會使用預設佇列管理程式 (如果有的話)。 只能使用本端佇列; 會發出 MQINQ 來檢查佇列類型，如果佇列不是本端佇列，則會報告錯誤。 如果未順利開啟佇列，或佇列上禁止 MQGET 呼叫，則會傳回錯誤訊息，指出 CompCode 及「原因」回覆碼。

訊息清單程式

這會顯示佇列上的訊息清單及其相關資訊，例如 putdate、puttime 及訊息格式。

清單中儲存的訊息數目上限為 99。 如果佇列上的訊息數超過此值，則也會顯示現行佇列深度。 若要選擇要顯示的訊息，請在輸入欄位中鍵入訊息號碼 (預設值為 01)。 如果您的輸入無效，您會收到適當的錯誤訊息。

訊息內容程式

這會顯示訊息內容。

內容會格式化並分割成兩個部分:

1. 訊息描述子
2. 訊息緩衝區

訊息描述子會在個別行上顯示每一個欄位的內容。

訊息緩衝區會根據其內容來格式化。 如果緩衝區保留無法傳送的郵件標頭 (MQDLH) 或傳輸佇列標頭 (MQXQH) ，這些會格式化並顯示在緩衝區本身之前。

在格式化緩衝區資料之前，標題行會顯示訊息的緩衝區長度 (以位元組為單位)。 緩衝區大小上限是 32768 個位元組，超過此上限的任何訊息都會被截斷。 緩衝區的完整大小會隨訊息一起顯示，指出只會顯示訊息的前 32768 個位元組。

緩衝區資料有兩種格式化方式:

1. 列印到緩衝區的偏移之後，緩衝區資料會以十六進位顯示。
2. 然後緩衝區資料會再次顯示為 EBCDIC 值。 如果無法列印任何 EBCDIC 值，則會改為列印句點 (.)。

您可以在動作欄位中輸入 D (代表刪除) 或 F (代表轉遞)。 如果您選擇轉遞訊息，則必須正確設定 `forward-to queue` 和 `queue manager name` 。 這些欄位的預設值是從訊息描述子 ReplyToQ 及 ReplyToQMgr 欄位讀取。

如果您轉遞訊息，則會除去儲存在緩衝區中的任何標頭區塊。 如果順利轉遞訊息，則會從原始佇列中移除該訊息。 如果您輸入無效動作，則會顯示錯誤訊息。

也提供稱為 CSQ4CHP9 的說明畫面範例。

z/OS *z/OS* 上的「非同步放置」範例

「非同步放置」範例程式會使用非同步 MQPUT 呼叫將訊息放置在佇列上。 此範例也會使用 MQSTAT 呼叫來擷取狀態資訊。

「非同步放置」應用程式使用下列 MQI 呼叫:

- MQCONN
- MQOPEN
- MQPUT
- MQSTAT

- MQCLOSE
- MQDISC

範例程式以 C 程式設計語言提供。

「非同步放置」應用程式在批次環境中執行。 請參閱批次應用程式的 其他範例 。

本主題也提供「非同步耗用程式」設計及執行 CSQ4BCS2 範例的相關資訊。

- 第 1018 頁的『執行 CSQ4BCS2 範例』
- 第 1018 頁的『非同步 Put 範例程式的設計』

執行 CSQ4BCS2 範例

此範例程式最多佔用六個參數:

1. 目標佇列的名稱 (必要)。
2. 佇列管理程式的名稱 (選用)。
3. 開啟選項 (選用)。
4. 關閉選項 (選用)。
5. 目標佇列管理程式的名稱 (選用)。
6. 動態佇列的名稱 (選用)。

如果未指定佇列管理程式，則 CSQ4BCS2 會連接至預設佇列管理程式。 訊息內容是透過標準輸入 (**SYSIN DD**) 提供。

有一個範例 JCL 可用來執行程式，它位於 CSQ4BCSP 中。

非同步 Put 範例程式的設計

程式使用 MQOPEN 呼叫搭配提供的輸出選項，或搭配 MQOO_OUTPUT 及 MQOO_FAIL_IF_QUIESCING 選項，來開啟目標佇列以放置訊息。

如果程式無法開啟佇列，則程式會輸出包含 MQOPEN 呼叫所傳回原因碼的錯誤訊息。 為了讓程式在此 MQI 呼叫及後續的 MQI 呼叫中保持簡單，許多選項會使用預設值。

對於每一行輸入，程式會將文字讀取到緩衝區中，並使用 MQPUT 呼叫搭配 MQPMO_ASYNC_RESPONSE 來建立資料包訊息，其中包含該行的文字，並以非同步方式將訊息放置在目標佇列上。 程式會繼續執行，直到到達輸入結尾，或直到 MQPUT 呼叫失敗為止。 如果程式到達輸入結尾，則會使用 MQCLOSE 呼叫來關閉佇列。

然後，程式會發出 MQSTAT 呼叫，該呼叫會傳回 MQSTS 結構，並顯示訊息，其中包含順利放置的訊息數、放置有警告的訊息數，以及失敗數。

註: 若要觀察 MQSTAT 呼叫偵測到 MQPUT 錯誤時所發生的情況，請將目標佇列上的 MAXDEPTH 設為低值。

z/OS *z/OS* 上的「批次非同步耗用」範例

CSQ4BCS1 範例程式以 C 遞送，它示範如何使用 MQCB 和 MQCTL 以非同步方式耗用來自多個佇列的訊息。

「非同步耗用」範例在批次環境中執行。 請參閱批次應用程式的 其他範例 。

還有一個在 CICS 環境中執行的 COBOL 範例，請參閱 第 1020 頁的『z/OS 上的 CICS 非同步耗用和發佈/訂閱範例』。

應用程式使用下列 MQI 呼叫:

- MQCONN
- MQOPEN
- MQCLOSE
- MQDISC
- MQCB

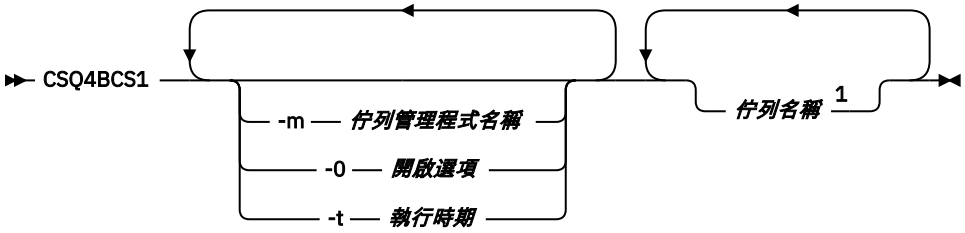
- MQCTL

這個主題也提供下列標題的相關資訊:

- 第 1019 頁的『執行 CSQ4BCS1 範例』
- 第 1019 頁的『批次非同步耗用範例程式的設計』

執行 CSQ4BCS1 範例

此範例程式遵循下列語法:



註:

[1] 此範例最多支援 10 個佇列。 請提供多個佇列名稱，以便從多個佇列讀取訊息。

有一個範例 JCL 可執行此程式，它位於 CSQ4BCSC 中。

批次非同步耗用範例程式的設計

此範例顯示如何依訊息到達的順序從多個佇列讀取訊息。 這將需要更多使用同步 MQGET 的程式碼。 使用非同步使用時，不需要輪詢，且 IBM MQ 會執行執行緒及儲存體管理。 在範例程式中，錯誤會寫入主控台。

範例程式碼具有下列步驟:

1. 定義單一訊息使用回呼函數。

```
void MessageConsumer(MQHCONN   hConn,
MQMD  * pMsgDesc,
MQGMO  * pGetMsgOpts,
MQBYTE * Buffer,
MQCBC  * pContext)
{ ... }
```

2. 連接至佇列管理程式。

```
MQCONN(QMName,&Hcon,&CompCode,&CReason);
```

3. 開啟輸入佇列，並將每一個佇列與 MessageConsumer 回呼函數相關聯。

```
MQOPEN(Hcon,&od,O_options,&Hobj,&OpenCode,&Reason);
cbd.CallbackFunction = MessageConsumer;
MQCB(Hcon,MQOP_REGISTER,&cbd,Hobj,&md,&gmo,&CompCode,&Reason);
```

cbd.CallbackFunction 不需要針對每一個佇列設定; 它是僅供輸入的欄位。 您可以將不同的回呼函數與每一個佇列相關聯。

4. 開始使用訊息。

```
MQCTL(Hcon,MQOP_START,&ctlo,&CompCode,&Reason);
```

5. 等待使用者按 Enter 鍵，然後停止使用訊息。

```
MQCTL(Hcon,MQOP_STOP,&ctlo,&CompCode,&Reason);
```

6. 最後，中斷與佇列管理程式的連線。

```
MQDISC(&Hcon,&CompCode,&Reason);
```

z/OS z/OS 上的 CICS 非同步耗用和發佈/訂閱範例

「非同步消費與發佈/訂閱」範例程式示範如何使用非同步消費，以及在 CICS 內發佈和訂閱特性。

登錄用戶端 程式會登錄三個回呼處理程式 (一個事件處理程式，以及兩個訊息消費者)，並啟動「非同步耗用」。 傳訊用戶端 程式會將訊息放入佇列，或從 CICS 主控台發佈適當的訊息，以供兩個「訊息消費者」(CSQ4CVCN 和 CSQ4CVCT) 使用。

若要提供對範例行為的執行時期控制，可以指示其中一個訊息消費者使用它所接收的訊息，來 SUSPEND、RESUME 或 DEREGISTER 任何回呼處理程式。 它也可以用來發出 MQCTL STOP，以在控制下結束「非同步耗用」。 另一個訊息消費者已登錄來訂閱主題。

每一個程式都會在適當的點發出 COBOL DISPLAY 陳述式，以顯示範例的行為。

應用程式使用下列 MQI 呼叫:

- MQOPEN
- MQPUT
- MQSUB
- MQGET
- MQCLOSE
- MQCB
- MQCTL

程式以 COBOL 語言交付。 請參閱 CICS 應用程式的 CICS 非同步耗用和發佈/訂閱範例 。

本主題也提供下列資訊:

- 第 1020 頁的『設定』
- 第 1020 頁的『登錄用戶端 CSQ4CVRG』
- 第 1021 頁的『事件處理程式 CSQ4CVEV』
- 第 1021 頁的『簡式訊息消費者 CSQ4CVCN』
- 第 1021 頁的『控制訊息消費者 CSQ4CVCT』
- 第 1021 頁的『傳訊用戶端 CSQ4CVPT』

設定

「訊息消費者」所使用的「佇列」及「主題」名稱會寫在「登錄及傳訊用戶端」程式中。

佇列 **SAMPLE.CONTROL.QUEUE** 應該定義給與 CICS 區域相關聯的「佇列管理程式」。 必要的話，可以定義主題 **News/Media/Movies**，如果預設管理物件不存在，則會在執行時期在預設管理物件下建立該主題。

可以透過安裝群組來安裝 CICS 程式及交易定義: CSQ4SAMP。

登錄用戶端 CSQ4CVRG

「註冊用戶端」程式必須在 CICS 交易 MVRG 下啟動。 它不需要任何輸入。

啟動時，「登錄用戶端」會使用 MQCB 來登錄下列回呼處理程式:

- CSQ4CVEV 作為事件處理程式。
- CSQ4CVCN，作為主題 **新聞/媒體/電影**的訊息消費者。

- CSQ4CVCT，作為佇列上的訊息消費者 **SAMPLE.CONTROL.QUEUE**。

「登錄用戶端」會將包含所有三個已登錄回呼處理程式名稱的資料結構傳遞至 CSQ4CVCT，以及與兩個訊息消費者相關聯的物件控點。

登錄回呼處理程式之後，「登錄用戶端」會發出 MQCTL START_WAIT 以啟動「非同步耗用」，並暫停直到傳回控制為止 (例如，由其中一個回呼處理程式發出 MQCTL STOP)。

事件處理程式 CSQ4CVEV

驅動時，「事件處理程式」會顯示一則訊息，指出呼叫類型 (例如， START)。 當驅動 IBM MQ 原因碼 CONNECTION_QUIESCING 時，「事件處理程式」會發出 MQCTL STOP 以結束「非同步耗用」，並將控制權傳回給「登錄用戶端」。

簡式訊息消費者 CSQ4CVCN

驅動時，此「訊息消費者」會顯示一則訊息，指出呼叫類型 (例如， REGISTER)。 針對 MSG_REMOVED 呼叫類型驅動時，「訊息消費者」會擷取入埠訊息並將其輸出至 CICS 工作日誌。

控制訊息消費者 CSQ4CVCT

驅動時，此「訊息消費者」會顯示一則訊息，指出呼叫類型 (例如， START)。 當驅動 MSG_REMOVED 呼叫類型時，「訊息消費者」會擷取入埠訊息及「登錄用戶端」所傳遞的資料結構。 根據訊息內容，它會對下列其中一項發出適當的 MQCB 或 MQCTL 指令:

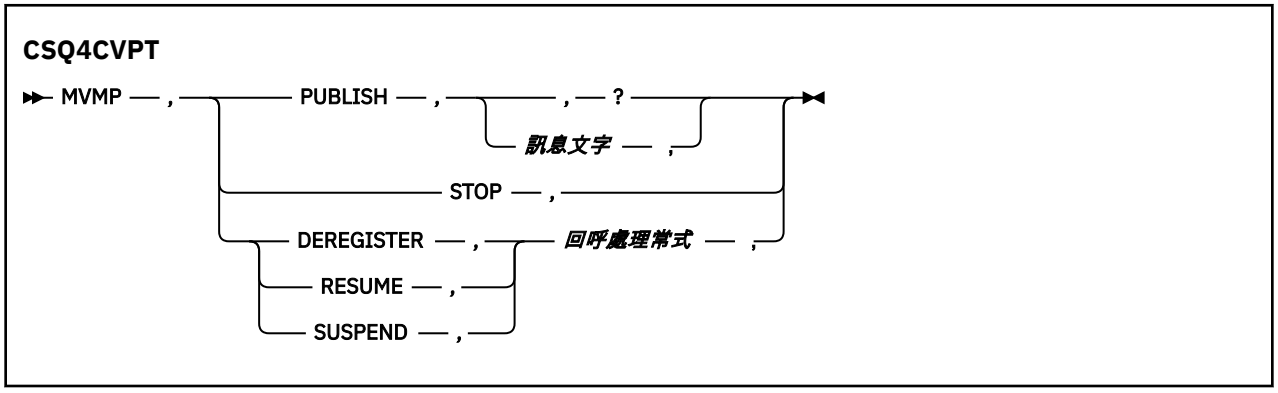
- STOP 非同步耗用 (將控制權傳回給登錄用戶端)。
- SUSPEND、RESUME 或 DEREGISTER 指定的回呼處理常式 (包括本身)。

傳訊用戶端 CSQ4CVPT

傳訊用戶端有兩個功能:

- 它會將訊息發佈至主題，以供訊息消費者 CSQ4CVCN 使用。
- 它會將控制訊息放入佇列以供「控制訊息消費者」 CSQ4CVCT 使用，導致範例行為可能發生變更。

「傳訊用戶端」程式必須從 CICS 交易下的 CICS 主控台啟動，並採用具有下列語法的指令行輸入:



PUBLISH
將「訊息文字」(或預設訊息) 發佈為「保留訊息」，以供「簡式訊息消費者」使用。

停止
停止非同步耗用。

取消登錄
取消登錄指定的回呼處理程式。

回復
回復指定的回呼處理程式。

SUSPEND
暫停指定的回呼處理程式。

輸入欄位是位置欄位，並以逗點區隔。 關鍵字和回呼處理程式名稱不區分大小寫。

範例:

表 *186:* 輸入範例	
範例	**說明**
MVMP， PUBLISH， ，	發佈預設訊息
MVMP,publish， A short message,	發佈給定的文字
MVMP 停止	停止非同步耗用
MVMP,DEREGISTER,CSQ4CVEV,	取消登錄事件處理程式
MVMP,resume,csq4cvcn,	回復簡式訊息消費者
MVMP,SUSPEND,CSQ4CVEV,	暫停事件處理程式

其中 MVMP 是與傳訊用戶端程式 CSQ4CVPT 相關聯的 CICS 交易。

註:

- 暫停或取消登錄所有回呼處理程式會終止「登錄用戶端」發出的 START_WAIT，將控制權交還給它，並結束作業。
- 故意不阻止暫停或取消登錄「控制回呼處理程式」，但它會移除進一步控制範例行為的能力。

z/OS *z/OS* 上的發佈/訂閱範例

「發佈/訂閱」範例程式示範如何使用 IBM MQ 中的發佈和訂閱特性。

有四個 C 和兩個 COBOL 程式設計語言範例程式示範如何對「 IBM MQ 發佈/訂閱」介面進行程式設計。 程式以 C 和 COBOL 語言交付。 應用程式在批次環境中執行; 請參閱批次應用程式的 發佈/訂閱範例 。

也有在 CICS 環境中執行的 COBOL 範例; 請參閱 第 1020 頁的『z/OS 上的 CICS 非同步耗用和發佈/訂閱範例』。

本主題也提供如何執行「發佈/訂閱」範例程式的相關資訊。 這些範例程式包括:

- 第 1022 頁的『執行 CSQ4BCP1 範例』
- 第 1023 頁的『執行 CSQ4BCP2 範例』
- 第 1023 頁的『執行 CSQ4BCP3 範例』
- 第 1023 頁的『執行 CSQ4BCP4 範例』
- 第 1024 頁的『執行 CSQ4BVP1 範例』
- 第 1024 頁的『執行 CSQ4BVP2 範例』

執行 CSQ4BCP1 範例

此程式以 C 撰寫; 它會將訊息發佈至主題。 在執行此程式之前，請先啟動其中一個訂閱者範例。

此程式最多使用四個參數:

1. 目標主題字串的名稱 (必要)。
2. 佇列管理程式的名稱 (選用)。
3. 開啟選項 (選用)。
4. 關閉選項 (選用)。

如果未指定佇列管理程式，則 CSQ4BCP1 會連接至預設佇列管理程式。 有一個範例 JCL 可執行程式，它位於 CSQ4BCPP 中。

訊息內容是透過標準輸入 (**SYSIN DD**) 提供。

執行 CSQ4BCP2 範例

此程式以 C 撰寫; 它會訂閱主題並列印收到的訊息。

此程式最多需要三個參數:

1. 目標主題字串的名稱 (必要)。
2. 佇列管理程式的名稱 (選用)。
3. MQSD 訂閱選項 (選用)。

如果未指定佇列管理程式，則 CSQ4BCP2 會連接至預設佇列管理程式。 有一個範例 JCL 可執行程式，它位於 CSQ4BCPS 中。

執行 CSQ4BCP3 範例

此程式以 C 撰寫; 它會使用使用者指定的目的地佇列來訂閱主題，並列印收到的訊息。

此程式最多使用四個參數:

1. 目標主題字串的名稱 (必要)。
2. 目的地的名稱 (必要)。
3. 佇列管理程式的名稱 (選用)。
4. MQSD 訂閱選項 (選用)。

如果未指定佇列管理程式，則 CSQ4BCP3 會連接至預設佇列管理程式。 有一個範例 JCL 可執行程式，它位於 CSQ4BCPD 中。

執行 CSQ4BCP4 範例

此程式以 C 撰寫; 它會從容許在 MQSUB 呼叫上使用延伸選項的主題中訂閱及取得訊息，並延伸在較簡單 MQSUB 範例上可用的選項: CSQ4BCP2。 除了訊息有效負載之外，還會接收並顯示每一個訊息的訊息內容。

此程式會採用一組變數參數:

- **-t** *Topic string*.
- **-o** *Topic object name*.

  重要: 需要 **-t** 或 **-o** 之一或兩者
- **-m** *Queue manager name* (選用)。
- **-b** *Connection binding type* (選用)，其中 *type* 可以具有下列任何值:
  - *standard* :MQCNO_STANDARD_BINDING，這是預設值
  - 共用:MQCNO_SHARED_BINDING
  - *fastpath*:XX_ENCODE_CASE_ONE mqcno_fastpath_binding
  - *isolated*: MQCNO_ISOLATED_BINDING
- **-q** *Destination queue name* (選用)。
- **-w** *Wait interval on MQGET in seconds* (選用)，其中 *seconds* 可以具有下列任何值:
  - unlimited:MQWI_UNLIMITED
  - none: 不等待
  - *n*: 等待間隔 (以秒為單位)
  - 未指定任何值: 未指定任何值時，預設值為 30 秒
- **-d** *Subscription name* (選用)。 建立或回復指名的可延續訂閱。
- **-k** (選用)。 保留 MQCLOSE 上的可延續訂閱。

如果未指定佇列管理程式，則 CSQ4BCP4 會連接至預設佇列管理程式。 有一個範例 JCL 可執行程式，它位於 CSQ4BCPE 中。

執行 CSQ4BVP1 範例

此程式以 COBOL 撰寫，它會將訊息發佈至主題。 在執行此程式之前，請先啟動其中一個訂閱者範例。

此程式不採用任何參數。 **SYSIN DD** 提供輸入主題名稱、佇列管理程式名稱及訊息內容。

如果未指定佇列管理程式，則 CSQ4BVP1 會連接至預設佇列管理程式。 有一個範例 JCL 可執行程式，它位於 CSQ4BVPP 中。

執行 CSQ4BVP2 範例

這個程式是以 COBOL 撰寫，它會訂閱主題並列印收到的訊息。

此程式不採用任何參數。 **SYSIN DD** 提供主題名稱及佇列管理程式名稱的輸入。

如果未指定佇列管理程式，則 CSQ4BVP1 會連接至預設佇列管理程式。 有一個範例 JCL 可執行程式，它位於 CSQ4BVPP 中。

z/OS *z/OS* 上的 *Set and Inquire message* 內容範例

訊息內容範例程式示範將使用者定義內容新增至訊息控點，以及查詢與該訊息相關聯的內容。

應用程式使用下列 MQI 呼叫:

- MQCONN
- MQOPEN
- MQPUT
- MQGET
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP
- MQSETMP

程式以 C 語言交付。 應用程式在批次環境中執行。 請參閱批次應用程式的 其他範例 。

CSQ4BCM1 程式用來從訊息佇列查詢訊息控點的內容，它是使用 MQINQMP API 呼叫的範例。 範例會從佇列中取得一則訊息，然後列印所有訊息控點內容。

CSQ4BCM2 程式用來設定訊息佇列上訊息控點的內容，它是使用 MQSETMP API 呼叫的範例。 此範例會建立訊息控點，並將它放入 MQGMO 結構的 `MsgHandle` 欄位中。 然後，它會將訊息放入佇列。

CSQ4BCG1 及 CSQ4BCP4 範例程式中包含查詢及列印訊息內容的其他範例。

本主題也提供在下列標題下執行「設定」及「查詢」訊息內容範例的相關資訊:

- 第 1024 頁的『執行 CSQ4BCM1 範例』
- 第 1025 頁的『執行 CSQ4BCM2 範例』

執行 CSQ4BCM1 範例

此程式最多使用四個參數:

1. 目標佇列的名稱 (必要)。
2. 佇列管理程式的名稱 (選用)。
3. 開啟選項 (選用)。

4. 關閉選項 (選用)。

執行 CSQ4BCM2 範例

此程式最多佔用六個參數:

1. 目標佇列的名稱 (必要)。
2. 佇列管理程式的名稱 (選用)。
3. 開啟選項 (選用)。
4. 關閉選項 (選用)。
5. 目標佇列管理程式的名稱 (選用)。
6. 動態佇列的名稱 (選用)。

內容名稱、值及訊息內容是透過標準輸入 (**SYSIN DD**) 提供。 有一個範例 JCL 可執行程式，它位於 CSQ4BCMP 中。

開發適用於 Managed File Transfer 的應用程式

指定要與 Managed File Transfer 搭配執行的程式、搭配使用 Apache Ant 與 Managed File Transfer、搭配使用自訂 Managed File Transfer 與使用者結束程式，以及透過將訊息放置在代理程式指令佇列上來控制 Managed File Transfer 。

指定要使用 MFT 執行的程式

您可以在執行 Managed File Transfer Agent 的系統上執行程式。 在檔案傳送要求過程中，您可以指定程式在傳送開始前或結束後執行。 此外，您也可以提交受管理的呼叫要求，以啟動不在檔案傳送要求過程中的程式。

關於這項作業

您可以指定讓程式在下列五種情況下執行:

- 作為傳送要求的一部分，在傳送開始之前，在來源代理程式上。
- 作為傳送要求的一部分，在傳送開始之前，在目的地代理程式中。
- 作為傳送要求的一部分，在傳送完成之後，在來源代理程式上。
- 作為傳送要求的一部分，在傳送完成之後，在目的地代理程式上。
- 不在傳送要求過程中。 您可以向代理程式提交執行程式的要求。 這種情況有時稱為受管理的呼叫。

使用者結束程式和程式呼叫的呼叫順序如下:

```
- SourceTransferStartExit(onSourceTransferStart).
- PRE_SOURCE Command.
- DestinationTransferStartExits(onDestinationTransferStart).
- PRE_DESTINATION Command.
- The Transfer request is performed.
- DestinationTransferEndExits(onDestinationTransferENd).
- POST_DESTINATION Command.
- SourceTransferEndExits(onSourceTransferEnd.
- POST_SOURCE Command.
```

附註:

1. 只有在傳送順利完成或局部順利完成時，才會執行 **DestinationTransferEndExits** 。
2. 只有在傳送順利完成或局部順利完成時，才會執行 **postDestinationCall** 。
3. **SourceTransferEndExits** 是針對成功、局部成功或失敗的傳送而執行。
4. 只有在下列情況下，才會呼叫 **postSourceCall** :
   - 傳送未取消。

- 有成功或部分成功的結果。
- 已順利執行任何後置目的地傳送程式。

程序

- 使用下列其中一個選項來指定您要執行的程式:

  使用 Apache Ant 作業

  使用其中一個 `fte:filecopy`、 `fte:filemove` 及 `fte:call` Ant 作業來啟動程式。 使用 Ant 作業，您可以使用 `fte:presrc`、 `fte:predst`、 `fte:postdst`、 `fte:postsrc` 及 `fte:command` 巢狀元素，在五個實務範例中的任何一個實務範例中指定程式。 如需相關資訊，請參閱 程式呼叫巢狀元素。

  編輯檔案傳送要求訊息

  您可以編輯傳送要求所產生的 XML。 使用此方法可讓您將 **preSourceCall**、**postSourceCall**、**preDestinationCall**、**postDestinationCall** 及 **managedCall** 元素新增至 XML 檔，在上述五種的任何一種情況中執行程式。 然後，這個修改過的 XML 檔即可作為新檔案傳送要求（例如使用 **fteCreateTransfer -td** 參數）的傳送定義。 如需相關資訊，請參閱 MFT 代理程式呼叫要求訊息範例。

  使用 `fteCreateTransfer` 指令

  您可以使用 **fteCreateTransfer** 指令來指定要啟動的程式。 您可以使用此指令來指定在前四種情況（在傳送要求過程中）下執行程式，但無法啟動受管理的呼叫。 如需使用之參數的相關資訊，請參閱 **fteCreateTransfer**: 啟動新的檔案傳送。 如需使用此指令的範例，請參閱 使用 fteCreateTransfer 來啟動程式的範例。

相關參考

commandPath MFT 內容

受管理呼叫

Managed File Transfer (MFT) 代理程式通常用來傳送檔案或訊息。 這些稱為 受管理傳送。 代理程式也可以用來執行指令、Script 或 JCL ， 而不需要傳送檔案或訊息。 此功能稱為 受管理呼叫。

受管理呼叫要求可以透過數種方式提交給代理程式:

- 使用 fte: call Ant 作業。
- 使用執行指令或 Script 的作業 XML 來配置資源監視器。 如需相關資訊，請參閱 配置監視器作業以啟動指令及 Script 。
- 將 XML 訊息直接放入代理程式的指令佇列中。 如需受管理呼叫 XML 綱目的詳細資料，請參閱 檔案傳送要求訊息格式 。

對於受管理呼叫，必須在代理程式內容 **commandPath** 中指定包含正在執行之指令或 Script 的目錄。

受管理呼叫無法執行位於未在代理程式的 **commandPath** 中指定之目錄中的指令或 Script。 這是為了確保代理程式不會執行任何惡意程式碼。

重要: 為了確保如此，當您指定 **commandPath** 時，依預設會執行下列動作:

- 代理程式會在啟動時配置任何現有的代理程式沙盤推演，以便將所有 **commandPath** 目錄自動新增至拒絕存取傳送的目錄清單。
- 當代理程式啟動時，會更新任何現有的使用者沙盤推演，以便將所有 **commandPath** 目錄 (及其子目錄) 作為 `<exclude>` 元素新增至 `<read>` 和 `<write>` 元素。
- 如果代理程式未配置為使用代理程式沙盤推演或使用者沙盤推演，則當代理程式啟動時，會建立新的代理程式沙盤推演，並將 **commandPath** 目錄指定為拒絕目錄。

此外，您也可以在代理程式上啟用權限檢查，以確保只容許授權使用者提交受管理呼叫要求。 如需此作業的相關資訊，請參閱 限制 MFT 代理程式動作的使用者權限。

在受管理呼叫中呼叫的指令、Script 或 JCL，會以代理程式所監視的外部程序來執行。 當處理程序結束時，受管理呼叫會完成，且處理程序的回覆碼可供呼叫 **fte:call** Ant 作業的代理程式或 Ant Script 使用。

如果受管理呼叫由 **fte:call** Ant 作業啟動，則 Ant Script 可以檢查回覆碼的值，以判定受管理呼叫是否成功。

對於所有其他類型的受管理呼叫，您可以指定應使用哪些回覆碼值來指出受管理呼叫已順利完成。 當外部處理程序完成時，代理程式會將來自處理程序的回覆碼與這些回覆碼進行比較。

註: 因為受管理呼叫以外部處理程序執行，所以一旦啟動就無法取消它們。

受管理呼叫及來源傳送插槽

代理程式包含許多來源傳送插槽，如代理程式內容 **maxSourceTransfers** 所指定，如 進階代理程式內容: 傳送限制中所述。

每當執行受管理呼叫或受管理傳送時，它們會佔用來源傳送插槽。 當受管理呼叫或受管理轉接完成時，會釋放插槽。

當代理程式收到新的受管理呼叫或受管理傳送要求時，如果所有來源傳送時段都在使用中，則代理程式會將要求排入佇列，直到時段變成可用為止。

如果受管理呼叫啟動受管理傳送 (例如，如果受管理呼叫執行 Ant Script，且該 Ant Script 使用 fte: filecopy 或 fte: filemove 作業來傳送檔案)，則需要兩個來源傳送插槽:

- 一個用於受管理傳送
- 一個用於受管理呼叫

在此狀況下，請務必注意，如果受管理傳送需要較長時間才能完成，或進入回復，則兩個來源傳送插槽會被佔用，直到受管理傳送完成、取消或因 **transferRecoveryTimeout** 而逾時為止。 如需 **transferRecoveryTimeout** 的詳細資料，請參閱 傳送回復逾時概念。 這可能會限制代理程式可以處理的其他受管理傳送或受管理呼叫數目。

因此，您應該考量受管理呼叫的設計，以確保它不會長時間佔用來源傳送時段。

搭配使用 REST API 與受管理呼叫

V 9.3.0 V 9.3.0

HTTP GET 和 HTTP POST 動詞支援啟用受管理呼叫，且只能在 REST API 第 3 版上運作。

不支援其他動詞，例如 HTTP DELETE 和 HTTP UPDATE，如果您嘗試使用它們，則會傳回 HTTP 405 錯誤碼。

小心: 提交之後，無法使用 REST API 來取消受管理呼叫。

將 Apache Ant 與 MFT 搭配使用

Managed File Transfer 提供可用來將檔案傳送功能整合至 Apache Ant 工具的作業。

您可以使用 **fteAnt** 指令，在您已配置的 Managed File Transfer 環境中執行 Ant 作業。 您可以使用 Ant Script 中的檔案傳送 Ant 作業，從解譯 Scripting 語言協調複式檔案傳送作業。

如需 Apache Ant 的相關資訊，請參閱 Apache Ant 專案網頁: https://ant.apache.org/

相關概念

第 1028 頁的『開始使用 Ant Script 與 MFT 搭配』

搭配使用 Ant Script 與 Managed File Transfer 可讓您從解譯 Scripting 語言協調複式檔案傳送作業。

fteAnt: 在 MFT 中執行 Ant 作業

相關參考

第 1029 頁的『MFT 的 Ant 作業範例』

Managed File Transfer 的安裝提供了一些範例 Ant Script。 這些範例位於 *MQ_INSTALLATION_PATH*/mqft/samples/fteant 目錄中。 每一個範例 Script 包含一個 init 目標，請編輯 init 目標中的內容集，以搭配您的配置來執行這些 Script。

開始使用 Ant Script 與 MFT 搭配

搭配使用 Ant Script 與 Managed File Transfer 可讓您從解譯 Scripting 語言協調複式檔案傳送作業。

Ant Script

Ant Script（或建置檔）是可定義一個以上目標的 XML 文件。 這些目標包含要執行的作業元素。 Managed File Transfer 提供可用來將檔案傳送功能整合至 Apache Ant 的作業。 如果要瞭解 Ant Script ，請參閱 Apache Ant 專案網頁: https://ant.apache.org/

產品安裝架構的 *MQ_INSTALLATION_PATH*/mqft/samples/fteant 目錄中提供了使用 Managed File Transfer 作業的 Ant Script 範例。

在通訊協定橋接器代理程式上， Ant Script 會在通訊協定橋接器代理程式系統上執行。 這些 Ant Script 無法直接存取 FTP 或 SFTP 伺服器上的檔案。

名稱空間

名稱空間用來區分檔案傳送 Ant 作業與可能共用相同名稱的其他 Ant 作業。 您可以在 Ant Script 的專案標籤中定義名稱空間。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns:fte="antlib:com.ibm.wmqfte.ant.taskdefs" default="do_ping">

  <target name="do_ping">
     <fte:ping cmdqm="qm@localhost@1414@SYSTEM.DEF.SVRCONN" agent="agent1@qm1"
      rcproperty="ping.rc" timeout="15"/>
  </target>

</project>
```

屬性 xmlns:fte="antlib:com.ibm.wmqfte.ant.taskdefs" 告知 Ant 在媒體庫 com.ibm.wmqfte.ant.taskdefs 中尋找以 fte 為字首之作業的定義。

您不需要使用 fte 作為名稱空間字首；您可以使用任何值。 名稱空間字首 fte 用於所有範例及範例 Ant Script 中。

執行 Ant Script

若要執行包含檔案傳送 Ant 作業的 Ant Script ，請使用 **fteAnt** 指令。 例如：

```
fteAnt -file ant_script_location/ant_script_name
```

如需相關資訊，請參閱 **fteAnt**: 在 MFT 中執行 Ant 作業。

回覆碼

檔案傳送 Ant 作業會傳回與 Managed File Transfer 指令相同的回覆碼。 如需相關資訊，請參閱 MFT 的回覆碼。

相關參考

fteAnt: 在 MFT 中執行 Ant 作業

第 1029 頁的『MFT 的 Ant 作業範例』

Managed File Transfer 的安裝提供了一些範例 Ant Script。 這些範例位於 *MQ_INSTALLATION_PATH*/mqft/samples/fteant 目錄中。 每一個範例 Script 包含一個 init 目標，請編輯 init 目標中的內容集，以搭配您的配置來執行這些 Script。

MFT 的 Ant 作業範例

Managed File Transfer 的安裝提供了一些範例 Ant Script。 這些範例位於 *MQ_INSTALLATION_PATH*/mqft/samples/fteant 目錄中。 每一個範例 Script 包含一個 init 目標，請編輯 init 目標中的內容集，以搭配您的配置來執行這些 Script。

email

email 範例示範如何使用 Ant 作業來傳送檔案，並在傳送失敗時將電子郵件傳送至指定的電子郵件位址。 Script 會使用 Managed File Transfer ping 作業來確認來源及目的地代理程式為作用中，且能夠處理傳送。 如果兩個代理程式都在作用中，則 Script 會使用 Managed File Transfer fte: filecopy 作業在來源與目的地代理程式之間傳送檔案，而不刪除原始檔案。 如果傳送失敗， Script 會使用標準 Ant 電子郵件 作業來傳送包含失敗相關資訊的電子郵件。

hub

hub 範例由兩個 Script 組成: hubcopy.xml 和 hubprocess.xml. hubcopy.xml Script 顯示如何使用 Ant Scripting 來建置「hub 和輪輻」樣式拓蹼。 在此範例中，兩個檔案從輻射機器上執行的代理程式傳送至主軸機器上執行的代理程式。 這兩個檔案會同時傳送，當傳送完成時，會在中心機器上執行 hubprocess.xml Ant Script 來處理檔案。 如果這兩個檔案都正確傳送， Ant Script 會連結檔案的內容。 如果檔案未正確傳送，則 Ant Script 會透過刪除已傳送的任何檔案資料來清除。 您必須將 hubprocess.xml Script 放置在主軸代理程式的指令路徑，此範例才能正確運作。 如需設定代理程式指令路徑的相關資訊，請參閱 commandPath MFT 內容。

librarytransfer（僅適用於 IBM i 平台）

IBM i

IBM i librarytransfer 範例示範如何使用 Ant 作業，將 IBM i 系統上的 IBM i 程式庫傳送至第二個 IBM i 系統。

IBM i librarytransfer 範例使用 IBM i 上的原生儲存檔支援，以及 Managed File Transfer 中可用的預先定義 Ant 作業，在兩個 IBM i 系統之間傳送原生程式庫物件。 The sample uses a <presrc> nested element in a Managed File Transfer filecopy task to invoke an executable script librarysave.sh that saves the requested library on the source agent system into a temporary save file. The save file is moved by the filecopy ant task to the destination agent system where a <postdst> nested element is used to invoke the executable script libraryrestore.sh to restore the library saved in the save file to the destination system.

IBM i 執行此範例之前，您需要完成 librarytransfer.xml 檔中說明的一些配置。 您也必須在兩部 IBM i 機器上具有工作中的 Managed File Transfer 環境。 設定必須包含第一部 IBM i 機器上執行的來源代理程式及第二部 IBM i 機器上執行的目的地代理程式。 兩個代理程式必須能夠彼此通訊。

IBM i librarytransfer 範例由下列三個檔案組成:

- librarytransfer.xml
- librarysave.sh (<presrc> executable script)
- libraryrestore.sh (<postdst> executable script)

範例檔案位於下列目錄: /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/librarytransfer

IBM i 若要執行此範例，使用者必須完成下列步驟:

1. 啟動 QShell 階段作業。 在 IBM i 指令視窗，鍵入: STRQSH
2. 切換至 bin 目錄，如下所示:

```
cd  /QIBM/ProdData/WMQFTE/V7/bin
```

3. 完成必要配置之後，使用下列指令執行範例:

```
fteant -f /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/librarytransfer/librarytransfer.xml
```

physicalfiletransfer（僅適用於 IBM i 平台）

IBM i physicalfiletransfer 範例示範如何使用 Ant 作業，將來源實體或資料庫檔案從一個 IBM i 系統上的檔案庫傳送至第二個 IBM i 系統上的檔案庫。

IBM i physicalfiletransfer 範例使用 IBM i 上的原生儲存檔支援，並搭配 Managed File Transfer 中提供的預先定義 Ant 作業，在兩個 IBM i 系統之間傳送完整的「來源實體」及「資料庫」檔案。 The sample uses a <presrc> nested element within a Managed File Transfer filecopy task to invoke an executable script `physicalfilesave.sh` to save the requested Source Physical or Database file from a library on the source agent system into a temporary save file. The save file is moved by the filecopy ant task to the destination agent system where a <postdst> nested element is used to invoke the executable script `physicalfilerestore.sh` then restores the file object inside the save file into a specified library on the destination system.

IBM i 執行此範例之前，您必須完成 `physicalfiletransfer.xml` 檔中說明的一些配置。 您還必須在兩個 IBM i 系統上具有工作中 Managed File Transfer 環境。 設定必須包含第一個 IBM i 系統上執行的來源代理程式及第二個 IBM i 系統上執行的目的地代理程式。 兩個代理程式必須能夠彼此通訊。

IBM i physicalfiletransfer 範例由下列三個檔案組成：

- `physicalfiletransfer.xml`
- `physicalfilesave.sh` (<presrc> executable script)
- `physicalfilerestore.sh` (<postdst> executable script)

範例檔案位於下列目錄：`/QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/physicalfiletransfer`

IBM i 若要執行此範例，使用者必須完成下列步驟：

1. 啟動 QShell 階段作業。 在 IBM i 指令視窗，鍵入：STRQSH
2. 切換至 `bin` 目錄，如下所示：

```
cd  /QIBM/ProdData/WMQFTE/V7/bin
```

3. 完成必要配置之後，使用下列指令執行範例：

```
fteant -f /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/physicalfiletransfer/
physicalfiletransfer.xml
```

timeout

timeout 範例示範如何使用 Ant 作業來嘗試檔案傳送，以及在所費時間超過指定的逾時值時取消傳送。 Script 會使用 Managed File Transfer fte: filecopy 作業來起始檔案傳送。 此傳送的結果會延遲。 Script 會使用 Managed File Transfer fte: awaitoutcome Ant 作業 來等待給定的秒數，讓傳送完成。 如果傳送未在給定時間內完成，則會使用 Managed File Transfer fte: cancel Ant 作業 來取消檔案傳送。

vsamtransfer

z/OS

z/OS vsamtransfer 範例示範如何使用 Ant 作業，透過使用 Managed File Transfer 從 VSAM 資料集傳送至另一個 VSAM 資料集。 Managed File Transfer 目前不支援傳送 VSAM 資料集。 範例 Script 會使用 `presrc` 程式呼叫巢狀元素 來呼叫執行檔 `datasetcopy.sh`，將 VSAM 資料記錄卸載至循序資料集。 該 Script 使用 Managed File Transfer fte: filemove 作業，將循序資料集從來源代理程式傳送至目的地代理程

式。 然後該 Script 會使用 postdst 程式呼叫巢狀元素 來呼叫 loadvsam.jcl Script。 此 JCL Script 會將傳送的資料集記錄載入目的地 VSAM 資料集。 此範例使用 JCL 進行目的地呼叫，以示範此語言選項。 使用第二個 Shell Script 也能獲得相同結果。

z/OS 此範例不要求來源及目的地資料集必須為 VSAM。 如果來源及目的地資料集是相同類型，則此範例適用於任何資料集。

z/OS 您必須將 datasetcopy.sh Script 放置在來源代理程式的指令路徑，將 loadvsam.jcl Script 放置在目的地代理程式的指令路徑，此範例才能正確運作。 如需設定代理程式指令路徑的相關資訊，請參閱 commandPath MFT 內容。

zip

zip 範例由兩個 Script 組成： zip.xml 及 zipfiles.xml。 此範例示範如何在執行檔案傳送移動作業之前，使用 Managed File Transfer fte: filemove 作業內的 presrc 巢狀元素 來執行 Ant Script。 zip.xml Script 中的 presrc 巢狀元素所呼叫的 zipfiles.xml Script 會壓縮目錄的內容。 zip.xml Script 會傳送壓縮檔。 此範例要求 zipfiles.xml Ant Script 存在於來源代理程式的指令路徑上。 這是因為 zipfiles.xml Ant Script 包含用來壓縮來源代理程式上目錄內容的目標。 如需設定代理程式指令路徑的相關資訊，請參閱 commandPath MFT 內容。

相關概念

第 1028 頁的『開始使用 Ant Script 與 MFT 搭配』

搭配使用 Ant Script 與 Managed File Transfer 可讓您從解譯 Scripting 語言協調複式檔案傳送作業。

相關參考

fteAnt: 在 MFT 中執行 Ant 作業

利用使用者結束程式自訂 MFT

您可以使用專屬程式（即所謂的使用者結束常式）來自訂 Managed File Transfer 的特性。

重要: IBM 不支援使用者結束程式內的任何程式碼，您的企業或提供該結束程式的供應商必須先調查該程式碼的任何問題。

Managed File Transfer 在程式碼提供位置點，讓 Managed File Transfer 能夠將控制權傳遞給您所撰寫的程式（使用者結束程式）。 這些點即所謂的使用者結束點。 當您的程式完成其工作時，Managed File Transfer 可以回復控制權。 您未必要使用任何使用者結束程式，但如果您想要延伸及自訂 Managed File Transfer 系統的功能來符合特定需求，則使用者結束程式會很有幫助。

在檔案傳送處理期間，有兩個位置點可供您在來源系統呼叫使用者結束程式，在檔案傳送處理期間，還有兩個位置點可供您在目的地系統呼叫使用者結束程式。 下表彙總這些使用者結束點，以及您要使用結束點時必須實作的 Java 介面。

表 *187:* 來源端和目的地端結束點及 *Java* 介面的摘要

結束點	**要實作的 Java 介面**
來源端結束點:	
在整個檔案傳送開始之前	SourceTransferStartExit.java 介面
在整個檔案傳送完成後	SourceTransferEndExit.java 介面
目的地端結束點:	
在整個檔案傳送開始之前	DestinationTransferStartExit.java 介面
在整個檔案傳送完成後	DestinationTransferEndExit.java 介面

依下列順序呼叫使用者結束程式:

1. SourceTransferStartExit
2. DestinationTransferStartExit

3. DestinationTransferEndExit
4. SourceTransferEndExit

SourceTransferStartExit 及 DestinationTransferStartExit 結束程式所做的變更會延伸到後續的結束程式成為輸入。 比方說，例如 SourceTransferStartExit 結束程式修改傳送 meta 資料，此變更會反映在其他結束程式的輸入傳送 meta 資料中。

使用者結束程式和程式呼叫的呼叫順序如下:

```
- SourceTransferStartExit(onSourceTransferStart).
- PRE_SOURCE Command.
- DestinationTransferStartExits(onDestinationTransferStart).
- PRE_DESTINATION Command.
- The Transfer request is performed.
- DestinationTransferEndExits(onDestinationTransferENd).
- POST_DESTINATION Command.
- SourceTransferEndExits(onSourceTransferEnd.
- POST_SOURCE Command.
```

附註:

1. 只有在傳送順利完成或局部順利完成時，才會執行 **DestinationTransferEndExits** 。
2. 只有在傳送順利完成或局部順利完成時，才會執行 **postDestinationCall** 。
3. **SourceTransferEndExits** 是針對成功、局部成功或失敗的傳送而執行。
4. 只有在下列情況下，才會呼叫 **postSourceCall** :
   - 傳送未取消。
   - 有成功或部分成功的結果。
   - 已順利執行任何後置目的地傳送程式。

建置使用者結束程式

建置使用者結束程式的介面包含在 *MQ_INSTALL_DIRECTORY*/mqft/lib/com.ibm.wmqfte.exitroutines.api.jar 中。 在建置結束程式時，您必須將這個 .jar 檔併入類別路徑中。 若要執行結束程式，請將結束程式解壓縮成 .jar 檔，然後將這個 .jar 檔放到下一節所說明的目錄中。

使用者結束程式位置

您可以在兩個可能的位置儲存使用者結束常式:

- exits 目錄。 每一個代理程式目錄下都有一個 exits 目錄。 例如: var\mqm\mqft\config\QM_JUPITER\agents\AGENT1\exits
- 您可以設定 exitClassPath 內容來指定替代位置。 如果 exits 目錄及 exitClassPath 所設定的類別路徑中都有結束程式類別，則以 exits 目錄中的類別為優先，這表示如果這兩個位置有相同名稱的類別，以 exits 目錄中的類別為優先。

配置代理程式使用使用者結束程式

可設定四項代理程式內容來指定代理程式呼叫的使用者結束程式。 這些代理程式內容是 sourceTransferStartExitClasses、sourceTransferEndExitClasses、destinationTransferStartExitClasses 及 destinationTransferEndExitClasses。 如需如何使用這些內容的相關資訊，請參閱 MFT 使用者結束程式的代理程式內容。

在通訊協定橋接器代理程式上執行使用者結束程式

當來源代理程式呼叫結束程式時，它會將一份要傳送的來源項目清單傳遞給結束程式。 對於一般代理程式，這是一份完整檔名清單。 因為檔案應該是本端檔案（或可透過裝載來存取），所以結束程式能夠存取它並對它進行加密。

但是，對於通訊協定橋接器代理程式，清單中的項目為下列格式:

```
"<file server identifier>:<fully-qualified file name of the file on the remote file server>"
```

對於清單中的每一個項目，結束程式都必須先連接至檔案伺服器（使用 FTP、 FTPS 或 SFTP 通訊協定），下載檔案，在本端加密檔案，然後將已加密的檔案上傳回檔案伺服器。

在 Connect:Direct 橋接器代理程式上執行使用者結束程式

您無法在 Connect:Direct 橋接器代理程式上執行使用者結束程式。

相關概念

第 1033 頁的『MFT 來源及目的地使用者結束程式』

MFT 使用者結束程式的 meta 資料

MFT 使用者結束程式的 Java 介面

相關參考

第 1037 頁的『對 MFT 使用者結束程式啟用遠端除錯』

在開發使用者結束程式時，您想要使用除錯器來協助找出程式碼的問題。

第 1038 頁的『MFT 來源傳送使用者結束程式範例』

第 1039 頁的『通訊協定橋接器認證使用者結束程式範例』

MFT 資源監視器使用者結束程式

使用者結束程式的 MFT 代理程式內容

MFT 來源及目的地使用者結束程式

目錄分隔字元

無論您在 **fteCreateTransfer** 指令或「IBM MQ Explorer」中指定的目錄分隔字元為何，原始檔規格中的目錄分隔字元一律使用正斜線 (/) 字元來表示。 在撰寫結束程式時，您必須考慮到這一點。 比方說，如果想要檢查下列原始檔是否存在：`c:\a\b.txt`，並且已使用 **fteCreateTransfer** 指令或 IBM MQ Explorer 指定此原始檔，請注意，該檔名實際上是儲存為 `c:/a/b.txt`。因此，如果以 `c:\a\b.txt` 的原始字串進行搜尋，將找不到相符項。

來源端結束點

在整個檔案傳送開始之前

當傳送要求為擱置傳送清單中的下一個項目，並且傳送即將開始時，來源代理程式會呼叫此結束程式。

此結束點的使用範例包括：使用外部指令分階段將檔案傳送至代理程式具有讀/寫存取權限的目錄，或者重新命名目的地系統上的檔案。

將下列引數傳遞至此結束程式：

- 來源代理程式名稱
- 目的地代理程式名稱
- 環境 meta 資料
- 傳送 meta 資料
- 檔案規格（包括檔案 meta 資料）

從此結束程式傳回的資料如下所示：

- 更新的傳送 meta 資料。 可以新增、修改及刪除項目。
- 更新的檔案規格清單，由來源檔案名稱及目的地檔名稱配對組成。 可以新增、修改及刪除項目
- 指定是否繼續進行傳送的指示器
- 要插入到「傳送日誌」的字串。

實作 SourceTransferStartExit.java 介面 來呼叫此結束點的使用者結束程式碼。

在整個檔案傳送完成後

在整個檔案傳送完成後，來源代理程式會呼叫此結束程式。

此結束點的使用範例包括執行一些完成作業，例如傳送電子郵件或 IBM MQ 訊息，以標示傳送已完成。

將下列引數傳遞至此結束程式：

- 傳送結束程式結果
- 來源代理程式名稱
- 目的地代理程式名稱
- 環境 meta 資料
- 傳送 meta 資料
- 檔案結果

從此結束程式傳回的資料如下所示：

- 要插入到「傳送日誌」的已更新字串。

實作 SourceTransferEndExit.java 介面，以在此結束點呼叫使用者結束程式碼。

目的地端結束點

在整個檔案傳送開始之前

此結束點的一個使用範例是驗證目的地上的權限。

將下列引數傳遞至此結束程式：

- 來源代理程式名稱
- 目的地代理程式名稱
- 環境 meta 資料
- 傳送 meta 資料
- 檔案規格

從此結束程式傳回的資料如下所示：

- 更新的目的地檔案名稱集。 可以修改，但不能新增或刪除項目。
- 指定是否繼續進行傳送的指示器
- 要插入到「傳送日誌」的字串。

實作 DestinationTransferStartExit.java 介面，以在此結束點呼叫使用者結束程式碼。

在整個檔案傳送完成後

此使用者結束程式的使用範例是啟動使用已傳送檔案的批次程序，或者在傳送失敗時傳送電子郵件。

將下列引數傳遞至此結束程式：

- 傳送結束程式結果
- 來源代理程式名稱
- 目的地代理程式名稱
- 環境 meta 資料
- 傳送 meta 資料
- 檔案結果

從此結束程式傳回的資料如下所示：

- 要插入到「傳送日誌」的已更新字串。

實作 DestinationTransferEndExit.java 介面，以在此結束點呼叫使用者結束程式碼。

相關概念

MFT 使用者結束程式的 Java 介面

相關參考

第 1037 頁的『對 MFT 使用者結束程式啟用遠端除錯』
在開發使用者結束程式時，您想要使用除錯器來協助找出程式碼的問題。

第 1038 頁的『MFT 來源傳送使用者結束程式範例』

MFT 資源監視器使用者結束程式

使用 MFT 傳送 I/O 使用者結束程式

您可以使用 Managed File Transfer 傳送 I/O 使用者結束程式，配置自訂程式碼來執行 Managed File Transfer 傳送的基礎檔案系統 I/O 工作。

通常針對 MFT 傳送，代理程式會選取其中一個內建 I/O 提供者，來與適當的檔案系統互動以進行傳送。 內建 I/O 提供者支援下列類型的檔案系統：

- 一般 UNIX 型及 Windows 型檔案系統
- z/OS z/OS 循序及分割的資料集（僅限在 z/OS 上）
- IBM i IBM i 原生儲存檔（僅限在 IBM i 上）
- IBM MQ 佇列
- 遠端 FTP 及 SFTP 通訊協定伺服器（僅限通訊協定橋接器代理程式）
- 遠端 Connect:Direct 節點（僅適用於 Connect:Direct 橋接器代理程式）

對於不支援的檔案系統，或是需要自訂 I/O 行為的地方，您可以撰寫傳送 I/O 使用者結束程式。

傳送 I/O 使用者結束程式使用使用者結束程式的現有基礎架構。 不過，這些傳送 I/O 使用者結束程式與其他使用者結束程式不同，因為在傳送每一個檔案時會多次存取其功能。

使用代理程式內容 IOExitClasses（在 `agent.properties` 檔中）來指定要載入的 I/O 結束程式類別。 請以逗點區隔每一個結束程式類別，例如：

```
IOExitClasses=testExits.TestExit1,testExits.testExit2
```

用於傳送 I/O 使用者結束程式的 Java 介面如下所示：

IOExit
用來決定是否使用 I/O 結束程式的主要進入點。 這個實例負責建立 IOExitPath 實例。

您只需要對代理程式內容 IOExitClasses 指定 IOExit I/O 結束程式介面。

IOExitPath
代表抽象介面；例如，資料儲存器或代表一組資料儲存器的萬用字元。 您不能建立實作此介面的類別實例。 此介面可以檢查路徑及列出衍生的路徑。 IOExitResourcePath 及 IOExitWildcardPath 介面延伸 IOExitPath。

IOExitChannel
能夠在 IOExitPath 資源中讀取或寫入資料。

IOExitRecordChannel
對於記錄導向的 IOExitPath 資源延伸 IOExitChannel 介面，能夠以多筆記錄的形式在 IOExitPath 資源中讀取或寫入資料。

IOExitLock
代表對 IOExitPath 資源的共用或互斥存取的鎖定。

z/OS **IOExitRecordResourcePath**
延伸 IOExitResourcePath 介面，以代表記錄導向檔案的資料儲存器；例如，z/OS 資料集。 您可以使用此介面來尋找資料，以及建立 IOExitRecordChannel 實例來進行讀取或寫入作業。

IOExitResourcePath
延伸 IOExitPath 介面來代表資料儲存器；例如，檔案或目錄。 您可以使用此介面來尋找資料。 如果此介面代表目錄，您可以使用 listPaths 方法來傳回路徑清單。

IOExitWildcardPath
延伸 IOExitPath 介面來代表一個表示萬用字元的路徑。 您可以使用此介面來比對多個 IOExitResourcePaths。

IOExitProperties
指定內容，以決定 Managed File Transfer 如何針對 I/O 的特定層面處理 IOExitPath 。 例如，如果重新啟動傳送，則是使用中間檔還是從頭開始重新讀取資源。

相關概念
第 1031 頁的『利用使用者結束程式自訂 MFT』
您可以使用專屬程式（即所謂的使用者結束常式）來自訂 Managed File Transfer 的特性。

相關參考
IOExit.java 介面
IOExitChannel.java 介面
IOExitLock.java 介面
IOExitPath.java 介面
IOExitProperties.java 介面
IOExitRecordChannel.java 介面
z/OS IOExitRecordResourcePath.java 介面
IOExitResourcePath.java 介面
IOExitWildcardPath.java 介面
MFT `agent.properties` 檔案

IBM i IBM i 使用者結束程式上的範例 MFT

Managed File Transfer 為您的安裝提供了 IBM i 特定的範例使用者結束程式。 這些範例位在 *MQMFT_install_dir*`/samples/ioexit-IBMi` 和 *MQMFT_install_dir*`/samples/userexit-IBMi` 目錄中。

com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit
com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit 範例使用者結束程式會傳送 IBM i 上 QDLS 檔案系統中的檔案。 安裝結束程式之後，對以 /QDLS 開頭的檔案的任何傳送都會自動使用結束程式。

若要安裝此結束程式，請完成下列步驟：

1. 將 `com.ibm.wmqfte.samples.ibmi.ioexits.jar` 檔案從 *WMQFTE_install_dir*`/samples/ioexit-IBMi` 目錄複製到代理程式的 `exits` 目錄。
2. 將 com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit 新增至 IOExitClasses 內容。
3. 請重新啟動代理程式。

com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit
com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit 範例使用者結束程式行為如同 MFT 檔案監視器，且會自動從 IBM i 程式庫傳送實體檔案成員。

若要執行此結束程式，請為 "library.qsys.monitor" meta 資料欄位指定值（例如，使用 **-md** 參數）。 此參數會擷取檔案成員的 IFS 樣式的路徑，且可以包含檔案及成員萬用字元。 例如，/QSYS.LIB/FOO.LIB/BAR.FILE/*.MBR、/QSYS.LIB/FOO.LIB/*.FILE/BAR.MBR 及 /QSYS.LIB/FOO.LIB/*.FILE/*.MBR。

此範例結束程式還具有選用的 meta 資料欄位 "naming.scheme.qsys.monitor"，您可以使用它來判斷傳送期間使用的命名方法。 依預設，此欄位設定為 "unix"，這會導致目的地檔案稱為 `F00.MBR`。 您也可以指定值 "ibmi" ，以使用 IBM i FTP FILE.MEMBER 架構，例如 /QSYS.LIB/FOO.LIB/BAR.FILE/BAZ.MBR 以 BAR.BAZ。

若要安裝此結束程式，請完成下列步驟：

1. 將 `com.ibm.wmqfte.samples.ibmi.userexits.jar` 檔案從 *`WMQFTE_install_dir`*`/samples/userexit-IBMi` 目錄複製到代理程式的 `exits` 目錄。
2. 將 com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit 新增至 `agent.properties` 檔案中的 sourceTransferStartExitClasses 內容。
3. 請重新啟動代理程式。

com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit

如果在傳送時刪除來源檔案成員，com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit 範例使用者結束程式會刪除空檔案物件。 因為 IBM i 檔案物件可能會保留許多成員，所以 MFT 會將檔案物件視為目錄。 因此，您無法使用 MFT 對檔案物件執行移動作業；僅在成員層次上支援移動作業。 從而，在您對成員執行移動作業時，目前的空檔案會保留。 在傳送要求時，如果您要刪除這些空檔案，請使用此範例結束程式。

如果您為 "empty.file.delete" meta 資料指定 "true"，且傳送 FTEFileMember，則範例結束程式會刪除空的上層檔案。

若要安裝此結束程式，請完成下列步驟：

1. 將 com.ibm.wmqfte.samples.ibmi.userexits.jar 檔案從 *`WMQFTE_install_dir`*`/samples/userexit-IBMi` 複製到代理程式的 `exits` 目錄。
2. 將 com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit 新增至 `agent.properties` 檔案中的 sourceTransferStartExitClasses 內容。
3. 請重新啟動代理程式。

相關參考

第 1035 頁的『使用 MFT 傳送 I/O 使用者結束程式』
您可以使用 Managed File Transfer 傳送 I/O 使用者結束程式，配置自訂程式碼來執行 Managed File Transfer 傳送的基礎檔案系統 I/O 工作。

使用者結束程式的 MFT 代理程式內容

對 MFT 使用者結束程式啟用遠端除錯

在開發使用者結束程式時，您想要使用除錯器來協助找出程式碼的問題。

因為結束程式是在執行代理程式的 Java 虛擬機器內執行，所以您無法使用整合開發環境通常會包括的直接除錯支援。 不過，您可以啟用 JVM 的遠端除錯，然後連接適當的遠端除錯器。

若要啟用遠端除錯，請使用標準 JVM 參數 **-Xdebug** 及 **-Xrunjdwp**。 這些內容會由 **BFG_JVM_PROPERTIES** 環境變數傳遞至執行代理程式的 JVM。 例如，在 AIX and Linux 上，下列指令會啟動代理程式，並使 JVM 在 TCP 埠 8765 接聽除錯器連線。

```
export BFG_JVM_PROPERTIES="-Xdebug -Xrunjdwp:transport=dt_socket,server=y,address=8765"
fteStartAgent -F TEST_AGENT
```

代理程式會等到除錯器連接之後才啟動。 在 Windows 上使用 **set** 指令，而非 **export** 指令。

您也可以在除錯器與 JVM 之間使用其他通訊方法。 例如，JVM 可以開啟除錯器的連線，反之則不然，或者，您可以使用共用記憶體代替 TCP。 如需進一步詳細資料，請參閱 Java 平台除錯器架構文件。

當您在遠端除錯模式中啟動代理程式時，必須使用 **-F**（前景）參數。

使用 Eclipse 除錯器

下列步驟適用於 Eclipse 開發環境中的遠端除錯功能。 您也可以使用 JPDA 相容的其他遠端除錯器。

1. 按一下**執行 > 開啟除錯對話框**（或**執行 > 除錯配置**或**執行 > 除錯對話框**，視您的 Eclipse 版本而定）。
2. 在配置類型清單中按兩下**遠端 Java 應用程式**，以建立除錯配置。
3. 完成配置欄位，並儲存除錯配置。 如果您已在除錯模式中啟動代理程式 JVM，可立即連接至 JVM。

MFT 來源傳送使用者結束程式範例

```
/*
 * A Sample Source Transfer End Exit that prints information about a transfer to standard
output.
 * If the agent is run in the background the output will be sent to the agent's event log file.
If
 * the agent is started in the foreground by specifying the -F parameter on the fteStartAgent
 * command the output will be sent to the console.
 *
 * To run the exit execute the following steps:
 *
 * Compile and build the exit into a jar file. You need the following in the class path:
 * {MQ_INSTALLATION_PATH}\mqft\lib\com.ibm.wmqfte.exitroutines.api.jar
 *
 * Put the jar in your agent's exits directory:
 * {MQ_DATA_PATH}\config\coordQmgrName\agents\agentName\exits\
 *
 * Update the agent's properties file:
 * {MQ_DATA_PATH}\config\coordQmgrName\agents\agentName\agent.properties
 * to include the following property:
 * sourceTransferEndExitClasses=[packageName.]SampleEndExit
 *
 * Restart agent to pick up the exit
 *
 * Send the agent a transfer request:
 * For example: fteCreateTransfer -sa myAgent -da YourAgent -df output.txt input.txt
 */

import java.util.List;
import java.util.Map;
import java.util.Iterator;

import com.ibm.wmqfte.exitroutine.api.SourceTransferEndExit;
import com.ibm.wmqfte.exitroutine.api.TransferExitResult;
import com.ibm.wmqfte.exitroutine.api.FileTransferResult;

public class SampleEndExit implements SourceTransferEndExit {

    public String onSourceTransferEnd(TransferExitResult transferExitResult,
            String sourceAgentName,
            String destinationAgentName,
            Map<String, String>environmentMetaData,
            Map<String, String>transferMetaData,
            List<FileTransferResult>fileResults) {

        System.out.println("Environment Meta Data: " + environmentMetaData);
        System.out.println("Transfer Meta Data:    " + transferMetaData);

        System.out.println("Source agent:          " +
                sourceAgentName);
        System.out.println("Destination agent:     " +
                destinationAgentName);

        if (fileResults.isEmpty()) {
            System.out.println("No files in the list");
            return "No files";
        }
        else {

            System.out.println( "File list: ");

            final Iterator<FileTransferResult> iterator = fileResults.iterator();

            while (iterator.hasNext()){
                final FileTransferResult thisFileSpec = iterator.next();
                System.out.println("Source file spec: "  +
                        thisFileSpec.getSourceFileSpecification() +
                        ", Destination file spec: " +
                        thisFileSpec.getDestinationFileSpecification());
            }
        }
        return "Done";
    }
}
```

通訊協定橋接器認證使用者結束程式範例

如需如何使用此範例使用者結束程式的相關資訊，請參閱 使用結束程式類別對映檔案伺服器的認證。

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.Map;
import java.util.Properties;
import java.util.StringTokenizer;

import com.ibm.wmqfte.exitroutine.api.CredentialExitResult;
import com.ibm.wmqfte.exitroutine.api.CredentialExitResultCode;
import com.ibm.wmqfte.exitroutine.api.CredentialPassword;
import com.ibm.wmqfte.exitroutine.api.CredentialUserId;
import com.ibm.wmqfte.exitroutine.api.Credentials;
import com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit;

/**
 * A sample protocol bridge credential exit
 *
 * This exit reads a properties file that maps mq user ids to server user ids
 * and server passwords. The format of each entry in the properties file is:
 *
 * mqUserId=serverUserId,serverPassword
 *
 * The location of the properties file is taken from the protocol bridge agent
 * property protocolBridgeCredentialConfiguration.
 *
 * To install the sample exit compile the class and export to a jar file.
 * Place the jar file in the exits subdirectory of the agent data directory
 * of the protocol bridge agent on which the exit is to be installed.
 * In the agent.properties file of the protocol bridge agent set the
 * protocolBridgeCredentialExitClasses to SampleCredentialExit
 * Create a properties file that contains the mqUserId to serverUserId and
 * serverPassword mappings applicable to the agent. In the agent.properties
 * file of the protocol bridge agent set the protocolBridgeCredentialConfiguration
 * property to the absolute path name of this properties file.
 * To activate the changes stop and restart the protocol bridge agent.
 *
 * For further information on protocol bridge credential exits refer to
 * the WebSphere MQ Managed File Transfer documentation online at:
 * https://www.ibm.com/docs/SSEP7X_7.0.4/welcome/WelcomePagev7r0.html
 */
public class SampleCredentialExit implements ProtocolBridgeCredentialExit {

    // The map that holds mq user ID to serverUserId and serverPassword mappings
    final private Map<String,Credentials> credentialsMap = new HashMap<String, Credentials>();

    /* (non-Javadoc)
      * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#initialize(java.util.Map)
      */
     public synchronized boolean initialize(Map<String, String> bridgeProperties) {

        // Flag to indicate whether the exit has been successfully initialized or not
        boolean initialisationResult = true;

        // Get the path of the mq user ID mapping properties file
         final String propertiesFilePath = bridgeProperties.get("protocolBridgeCredentialConfiguration");

         if (propertiesFilePath == null || propertiesFilePath.length() == 0) {
             // The properties file path has not been specified. Output an error and return false
             System.err.println("Error initializing SampleCredentialExit.");
              System.err.println("The location of the mqUserID mapping properties file has not been
specified in the
       protocolBridgeCredentialConfiguration property");
             initialisationResult = false;
         }

        if (initialisationResult) {

            // The Properties object that holds mq user ID to serverUserId and serverPassword
             // mappings from the properties file
             final Properties mappingProperties = new Properties();
```

```
            // Open and load the properties from the properties file
            final File propertiesFile = new File (propertiesFilePath);
            FileInputStream inputStream = null;
            try {
                // Create a file input stream to the file
                inputStream = new FileInputStream(propertiesFile);

                // Load the properties from the file
                mappingProperties.load(inputStream);
            }
            catch (FileNotFoundException ex) {
                System.err.println("Error initializing SampleCredentialExit.");
                 System.err.println("Unable to find the mqUserId mapping properties file: " +
propertiesFilePath);
                initialisationResult = false;
            }
            catch (IOException ex) {
                System.err.println("Error initializing SampleCredentialExit.");
                 System.err.println("Error loading the properties from the mqUserId mapping properties
file: " + propertiesFilePath);
                initialisationResult = false;
            }
            finally {
                // Close the inputStream
                if (inputStream != null) {
                    try {
                        inputStream.close();
                    }
                    catch (IOException ex) {
                        System.err.println("Error initializing SampleCredentialExit.");
                         System.err.println("Error closing the mqUserId mapping properties file: " +
propertiesFilePath);
                        initialisationResult = false;
                    }
                }
            }

            if (initialisationResult) {
                // Populate the map of mqUserId to server credentials from the properties
                final Enumeration<?> propertyNames = mappingProperties.propertyNames();
                 while ( propertyNames.hasMoreElements()) {
                    final Object name = propertyNames.nextElement();
                    if (name instanceof String ) {
                        final String mqUserId = ((String)name).trim();
                         // Get the value and split into serverUserId and serverPassword
                        final String value = mappingProperties.getProperty(mqUserId);
                        final StringTokenizer valueTokenizer = new StringTokenizer(value, ",");
                         String serverUserId = "";
                         String serverPassword = "";
                         if (valueTokenizer.hasMoreTokens()) {
                             serverUserId = valueTokenizer.nextToken().trim();
                         }
                         if (valueTokenizer.hasMoreTokens()) {
                             serverPassword = valueTokenizer.nextToken().trim();
                         }
                         // Create a Credential object from the serverUserId and serverPassword
final Credentials credentials = new Credentials(new CredentialUserId(serverUserId), new
CredentialPassword(serverPassword));
                         // Insert the credentials into the map
                         credentialsMap.put(mqUserId, credentials);
                    }
                }

           }
        }

        return initialisationResult;
    }
     /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#mapMQUserId(java.lang.String)
     */
    public synchronized CredentialExitResult mapMQUserId(String mqUserId) {
        CredentialExitResult result = null;
        // Attempt to get the server credentials for the given mq user id
        final Credentials credentials = credentialsMap.get(mqUserId.trim());
        if ( credentials == null) {
            // No entry has been found so return no mapping found with no credentials
            result = new CredentialExitResult(CredentialExitResultCode.NO_MAPPING_FOUND, null);
        }
        else {
            // Some credentials have been found so return success to the user along with the credentials
```

```
            result = new CredentialExitResult(CredentialExitResultCode.USER_SUCCESSFULLY_MAPPED,
credentials);
        }
        return result;
    }
     /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#shutdown(java.util.Map)
     */
    public void shutdown(Map<String, String> bridgeProperties) {
        // Nothing to do in this method because there are no resources that need to be released
    }
}
```

通訊協定橋接器內容使用者結束程式範例

如需如何使用此範例使用者結束程式的相關資訊，請參閱 ProtocolBridgePropertiesExit2: 查閱通訊協定檔案伺服器內容

SamplePropertiesExit2.java

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Properties;

import com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit2;
import com.ibm.wmqfte.exitroutine.api.ProtocolServerPropertyConstants;

/**
 * A sample protocol bridge properties exit. This exit reads a properties file
 * that contains properties for protocol servers.
 * <p>
 * The format of each entry in the properties file is:
 * {@literal serverName=type://host:port}
 * Ensure there is a default entry such as
 * {@literal default=type://host:port}
 * otherwise the agent will fail to start with a BFGBR0168 as it must have a
 * default server.
 * <p>
 * The location of the properties file is taken from the protocol bridge agent
 * property {@code protocolBridgePropertiesConfiguration}.
 * <p>
 * The methods {@code getCredentialLocation} returns the location of the associated
 * ProtocolBridgeCredentials.xml, this sample it is defined to be stored in a directory
 * defined by the environment variable CREDENTIALSHOME
 * <p>
 * To install the sample exit:
 * <ol>
 * <li>Compile the class and export to a jar file.
 * <li>Place the jar file in the {@code exits} subdirectory of the agent data directory
 * of the protocol bridge agent on which the exit is to be installed.
 * <li>In the {@code agent.properties} file of the protocol bridge agent
 * set the {@code protocolBridgePropertiesExitClasses} to
 * {@code SamplePropertiesExit2}.
 * <li>Create a properties file that contains the appropriate properties to specify the
 * required servers.
 * <li>In the {@code agent.properties} file of the protocol bridge agent
 * set the <code>protocolBridgePropertiesConfiguration</code> property to the
 * absolute path name of this properties file.
 * <li>To activate the changes stop and restart the protocol bridge agent.
 * </ol>
 * <p>
 * For further information on protocol bridge properties exits refer to the
 * WebSphere MQ Managed File Transfer documentation online at:
 * <p>
 * {@link https://www.ibm.com/docs/SSEP7X_7.0.4/welcome/WelcomePagev7r0.html}
 */
public class SamplePropertiesExit2 implements ProtocolBridgePropertiesExit2 {

    /**
     * Helper class to encapsulate protocol server information.
     */
```

```
    private static class ServerInformation {
        private final String type;
        private final String host;
        private final int port;

        public ServerInformation(String url) {
            int index = url.indexOf("://");
            if (index == -1) throw new IllegalArgumentException("Invalid server URL: "+url);
            type = url.substring(0, index);

            int portIndex = url.indexOf(":", index+3);
            if (portIndex == -1) {
                host = url.substring(index+3);
                port = -1;
            } else {
                host = url.substring(index+3,portIndex);
                port = Integer.parseInt(url.substring(portIndex+1));
            }
        }

        public String getType() {
            return type;
        }

        public String getHost() {
            return host;
        }

        public int getPort() {
            return port;
        }
    }

    /** A {@code Map} that holds information for each configured protocol server */
    final private Map<String, ServerInformation> servers = new HashMap<String, ServerInformation>();


    /* (non-Javadoc)
     * @see
com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#getProtocolServerProperties(java.lang.String)
     */
    public Properties getProtocolServerProperties(String protocolServerName) {
        // Attempt to get the protocol server information for the given protocol server name
        // If no name has been supplied then this implies the default.
        final ServerInformation info;
        if (protocolServerName == null || protocolServerName.length() == 0) {
            protocolServerName = "default";
        }
        info = servers.get(protocolServerName);

        // Build the return set of properties from the collected protocol server information, when
available.
        // The properties set here is the minimal set of properties to be a valid set.
        final Properties result;
        if (info != null) {
            result = new Properties();
            result.setProperty(ProtocolServerPropertyConstants.SERVER_NAME, protocolServerName);
            result.setProperty(ProtocolServerPropertyConstants.SERVER_TYPE, info.getType());
            result.setProperty(ProtocolServerPropertyConstants.SERVER_HOST_NAME, info.getHost());
            if (info.getPort() != -1)
result.setProperty(ProtocolServerPropertyConstants.SERVER_PORT_VALUE, ""+info.getPort());
            result.setProperty(ProtocolServerPropertyConstants.SERVER_PLATFORM, "UNIX");
            if (info.getType().toUpperCase().startsWith("FTP")) {        // FTP & FTPS
                result.setProperty(ProtocolServerPropertyConstants.SERVER_TIMEZONE, "Europe/London");
                result.setProperty(ProtocolServerPropertyConstants.SERVER_LOCALE, "en-GB");
            }
            result.setProperty(ProtocolServerPropertyConstants.SERVER_FILE_ENCODING, "UTF-8");
        } else {
            System.err.println("Error no default protocol file server entry has been supplied");
            result = null;
        }

        return result;
    }

    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#initialize(java.util.Map)
     */
    public boolean initialize(Map<String, String> bridgeProperties) {
        // Flag to indicate whether the exit has been successfully initialized or not
        boolean initialisationResult = true;
```

```
        // Get the path of the properties file
        final String propertiesFilePath = bridgeProperties.get("protocolBridgePropertiesConfiguration");
        if (propertiesFilePath == null || propertiesFilePath.length() == 0) {
            // The protocol server properties file path has not been specified. Output an error and
return false
            System.err.println("Error initializing SamplePropertiesExit.");
            System.err.println("The location of the protocol server properties file has not been
specified in the
      protocolBridgePropertiesConfiguration property");
            initialisationResult = false;
        }

        if (initialisationResult) {
            // The Properties object that holds protocol server information
            final Properties mappingProperties = new Properties();

            // Open and load the properties from the properties file
            final File propertiesFile = new File (propertiesFilePath);
            FileInputStream inputStream = null;
            try {
                // Create a file input stream to the file
                inputStream = new FileInputStream(propertiesFile);

                // Load the properties from the file
                mappingProperties.load(inputStream);
            } catch (final FileNotFoundException ex) {
                System.err.println("Error initializing SamplePropertiesExit.");
                System.err.println("Unable to find the protocol server properties file: " +
propertiesFilePath);
                initialisationResult = false;
            } catch (final IOException ex) {
                System.err.println("Error initializing SamplePropertiesExit.");
                System.err.println("Error loading the properties from the protocol server properties
file: " + propertiesFilePath);
                initialisationResult = false;
            } finally {
                // Close the inputStream
                if (inputStream != null) {
                    try {
                        inputStream.close();
                    } catch (final IOException ex) {
                        System.err.println("Error initializing SamplePropertiesExit.");
                        System.err.println("Error closing the protocol server properties file: " +
propertiesFilePath);
                        initialisationResult = false;
                    }
                }
            }

            if (initialisationResult) {
                // Populate the map of protocol servers from the properties
                for (Entry<Object, Object> entry : mappingProperties.entrySet()) {
                    final String serverName = (String)entry.getKey();
                    final ServerInformation info = new ServerInformation((String)entry.getValue());
                    servers.put(serverName, info);
                }
            }
        }

        return initialisationResult;
    }

    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#shutdown(java.util.Map)
     */
    public void shutdown(Map<String, String> bridgeProperties) {
        // Nothing to do in this method because there are no resources that need to be released
    }

    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit2#getCredentialLocation()
     */
    public String getCredentialLocation() {
        String envLocationPath;
        if (System.getProperty("os.name").toLowerCase().contains("win")) {
            // Windows style
            envLocationPath = "%CREDENTIALSHOME%\\ProtocolBridgeCredentials.xml";
        }
        else {
            // Unix style
            envLocationPath = "$CREDENTIALSHOME/ProtocolBridgeCredentials.xml";
        }
```

```
        return envLocationPath;
    }

}
```

將訊息放置在代理程式指令佇列上以控制 MFT

您可以撰寫應用程式，以透過將訊息放置在代理程式指令佇列上來對 Managed File Transfer 進行控制。

您可以將訊息放置在代理程式的指令佇列上，以要求代理程式執行下列其中一個動作：

- 建立檔案傳送
- 建立排定的檔案傳送
- 取消檔案傳送
- 取消排定的檔案傳送
- 呼叫指令
- 建立監視器
- 刪除監視器
- 傳回連線測試，以指出代理程式處於作用中狀態

若要要求代理程式執行上述其中一個動作，訊息必須使用符合下列其中一個綱目的 XML 格式：

FileTransfer.xsd
使用此格式的訊息可用來建立檔案傳送或排定的檔案傳送、呼叫指令，或取消檔案傳送或排定的檔案傳送。 如需相關資訊，請參閱 檔案傳送要求訊息格式。

Monitor.xsd
使用此格式的訊息可用來建立或刪除資源監視器。 如需相關資訊，請參閱 MFT 監視器要求訊息格式。

PingAgent.xsd
使用此格式的訊息可用來連線測試代理程式，以確認它處於作用中狀態。 如需相關資訊，請參閱 Ping MFT 代理程式要求訊息格式。

代理程式會對要求訊息傳回回覆。 回覆訊息會放置在要求訊息中所定義的回覆佇列上。 回覆訊息會使用下列綱目所定義的 XML 格式：

Reply.xsd
如需相關資訊，請參閱 MFT 代理程式回覆訊息格式。

開發適用於 MQ Telemetry 的應用程式

遙測應用程式會整合感應及控制裝置與網際網路及企業中可用的其他資訊來源。

使用設計型樣、工作範例、程式範例、程式設計概念及參照資訊來開發 MQ Telemetry 的應用程式。

相關概念
MQ Telemetry
遙測使用案例

相關工作
正在安裝 MQ Telemetry
管理 MQ Telemetry
MQ Telemetry 問題疑難排解

相關參考
MQ Telemetry 參照

IBM MQ Telemetry Transport 範例程式

所提供的範例 Script 會使用範例 IBM MQ Telemetry Transport v3 用戶端應用程式 (`mqttv3app.jar`)。 對於 IBM MQ 8.0.0 以及更新版本，範例用戶端應用程式不再包含在 MQ Telemetry 中。 它是 IBM Messaging Telemetry Clients SupportPac 的一部分 (不再可用)。 類似的範例應用程式仍可從 Eclipse Paho 和 MQTT.org 免費取得。

如需最新資訊及下載項目，請參閱下列資源：

- Eclipse Paho 專案和 MQTT.org 具有最新遙測用戶端的免費下載，以及一系列程式設計語言的範例。 使用這些網站來協助您開發程式範例以發佈和訂閱 IBM MQ Telemetry Transport 及新增安全特性。
- IBM Messaging Telemetry Clients SupportPac 已無法再下載。 如果有先前下載的副本，則它具有下列內容：
  - IBM Messaging Telemetry Clients SupportPac 的 MA9B 版本包括完整的範例應用程式 (`mqttv3app.jar`) 及關聯用戶端程式庫 (`mqttv3.jar`)。在一下目錄提供：
    - `ma9b/SDK/clients/java/org.eclipse.paho.sample.mqttv3app.jar`
    - `ma9b/SDK/clients/java/org.eclipse.paho.client.mqttv3.jar`
  - 在此 SupportPac 的 MA9C 版本中，已移除 `/SDK/` 目錄及內容：
    - 只提供了範例應用程式（`mqttv3app.jar`）的來源。 它位於下列目錄中：

      ```
      ma9c/clients/java/samples/org/eclipse/paho/sample/mqttv3app/*.java
      ```

    - 仍提供已編譯的用戶端程式庫。 它位於下列目錄中：

      ```
      ma9c/clients/java/org.eclipse.paho.client.mqttv3-1.0.2.jar
      ```

如果您仍有 IBM Messaging Telemetry Clients SupportPac 的副本 (不再可用)，則 使用指令行驗證 MQ Telemetry 的安裝中會提供安裝及執行範例應用程式的相關資訊。

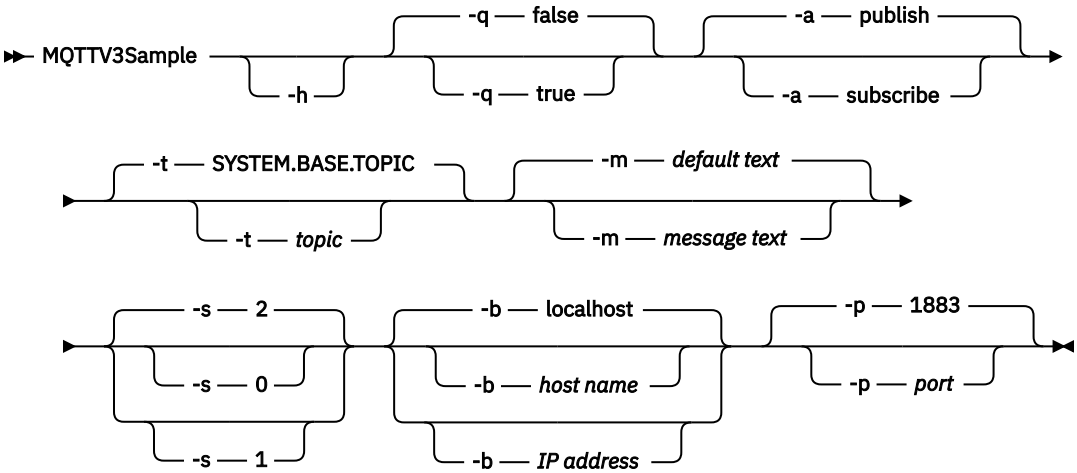
MQTTV3Sample 程式

`MQTTV3Sample` 程式的範例語法及參數的相關參考資訊。

用途

`MQTTV3Sample` 程式可用來發佈訊息及訂閱主題。 如需如何取得此範例程式的相關資訊，請參閱 第 1044 頁的『IBM MQ Telemetry Transport 範例程式』。

MQTTV3Sample syntax



參數

-h
列印此說明文字，然後退出

-q
設定無聲模式，而不是使用預設的 false 模式。

-a
設定發佈或訂閱，而不是採用預設動作發佈。

-t
發佈或訂閱主題，而不是發佈或訂閱預設主題

-m
發佈訊息文字，而不是傳送預設發佈文字 "Hello from an MQTT v3 application"。

-s
設定 QoS ，而不是使用預設 QoS， 2.

-b
請連接至這個主機名稱或 IP 位址，而不是連接至預設主機名稱 localhost。

-p
請使用此埠，而不是使用預設值 1883。

執行 MQTTV3Sample 程式

若要訂閱 Windows 上的主題，請使用下列指令:

```
run MQTTV3Sample -a subscribe
```

若要在 Windows 上發佈訊息，請使用下列指令:

```
run MQTTV3Sample
```

MQTT 用戶端程式設計概念

本節中說明的概念可協助您瞭解 MQTT protocol 的用戶端程式庫。 這些概念補充用戶端程式庫隨附的 API 文件。

如需最新資訊及下載項目，請參閱下列資源:

- Eclipse Paho 專案和 MQTT.org 具有最新遙測用戶端的免費下載，以及一系列程式設計語言的範例。 使用這些網站來協助您開發程式範例以發佈和訂閱 IBM MQ Telemetry Transport 及新增安全特性。
- IBM Messaging Telemetry Clients SupportPac 已無法再下載。 如果有先前下載的副本，則它具有下列內容:
  - IBM Messaging Telemetry Clients SupportPac 的 MA9B 版本包括完整的範例應用程式 (`mqttv3app.jar`) 及關聯用戶端程式庫 (`mqttv3.jar`)。在一下目錄提供:
    - `ma9b/SDK/clients/java/org.eclipse.paho.sample.mqttv3app.jar`
    - `ma9b/SDK/clients/java/org.eclipse.paho.client.mqttv3.jar`
  - 在此 SupportPac 的 MA9C 版本中，已移除 `/SDK/` 目錄及內容:
    - 只提供了範例應用程式（`mqttv3app.jar`）的來源。 它位於下列目錄中:

      ```
      ma9c/clients/java/samples/org/eclipse/paho/sample/mqttv3app/*.java
      ```

    - 仍提供已編譯的用戶端程式庫。 它位於下列目錄中:

      ```
      ma9c/clients/java/org.eclipse.paho.client.mqttv3-1.0.2.jar
      ```

若要開發及執行 MQTT 用戶端，您需要在用戶端裝置上複製或安裝這些資源。 您不需要安裝單獨的用戶端執行時期。

用戶端的授權狀況，與用戶端要連接的伺服器相關聯。

MQTT 用戶端程式庫是 MQTT protocol 的參照實作。 您可以利用適合不同裝置平台的不同語言，來撰寫自己適用的用戶端。 請參閱 IBM MQ Telemetry Transport 格式及通訊協定。

API 文件不會對用戶端所連接的 MQTT 伺服器做出任何假設。 當連接至不同的伺服器時，用戶端的行為可能會略有不同。 下列說明說明用戶端連接至 IBM MQ 遙測服務時的行為。

MQTT 用戶端應用程式中的回呼及同步化

MQTT 用戶端程式設計模型廣泛使用執行緒。 執行緒會盡可能取消 MQTT 用戶端應用程式與伺服器之間傳輸訊息的延遲之間的連結。 發佈、遞送記號及連線遺失事件會遞送至實作 MqttCallback 的回呼類別中的方法。

回呼

註: 如需 MqttCallback 的最新變更，請參閱 Eclipse Paho 網站。 例如， MqttCallback 在用戶端的 Paho 版本中定義為「介面」，且 Paho MqttAsyncClient 類別提供非同步方法。

MqttCallback 介面有三個回呼方法:

connectionLost(java.lang.Throwable cause)

當通訊錯誤導致連線中斷時，會呼叫 connectionLost。 如果在建立連線之後，由於伺服器上的錯誤導致伺服器中斷連線，也會呼叫此函數。 會將伺服器錯誤記載到佇列管理程式錯誤日誌。 伺服器中斷與用戶端的連線，而且用戶端呼叫 MqttCallback.connectionLost。

在與用戶端應用程式相同的執行緒上，唯一作為異常狀況擲出的遠端錯誤是來自 MqttClient.connect 的異常狀況。 在建立連線之後伺服器偵測到的錯誤，會向 MqttCallback.connectionLost 回呼方法回報為 throwables。

導致 connectionLost 的一般伺服器錯誤是權限錯誤。 例如，遙測伺服器嘗試代表未獲授權來在某個主題上發佈的用戶端，在該主題上進行發佈。 導致將 MQCC_FAIL 狀況碼傳回給遙測伺服器的任何內容，都會導致中斷連線。

deliveryComplete(IMqttDeliveryToken token)

deliveryComplete 由 MQTT 用戶端呼叫，以將遞送記號傳遞回用戶端應用程式; 請參閱 第 1052 頁的『遞送記號』。 透過使用遞送記號，回呼可以利用方法 token.getMessage 來存取已發佈的訊息。

當應用程式回呼在 deliveryComplete 方法呼叫之後將控制權交還給 MQTT 用戶端時，遞送即會完成。 在遞送完成之前，持續性類別會保留具有 QoS 1 或 2 的訊息。

對 deliveryComplete 的呼叫是應用程式與持續性類別之間的一個同步點。 不會對相同的訊息呼叫 deliveryComplete 方法兩次。

當應用程式回呼從 deliveryComplete 回到 MQTT 用戶端時，用戶端會針對具有 QoS 1 或 2 的訊息呼叫 MqttClientPersistence.remove 。 MqttClientPersistence.remove 會刪除已發佈訊息的本端儲存副本。

從交易處理的角度看，呼叫 deliveryComplete 是確認遞送的單一階段交易。 如果在回呼期間處理失敗，則在重新啟動用戶端時，會再次呼叫 MqttClientPersistence.remove，以刪除已發佈訊息的本端副本。 不會再次呼叫回呼。 如果您使用回呼來儲存已遞送訊息的日誌，則無法將日誌與 MQTT 用戶端同步化。 如果要可靠地儲存日誌，請在 MqttClientPersistence 類別中更新日誌。

遞送記號及訊息由主要應用程式執行緒和 MQTT 用戶端參照。 當遞送完成時， MQTT 用戶端會取消參照 MqttMessage 物件，當用戶端中斷連線時，會取消參照遞送記號物件。 如果用戶端應用程式取消參照 MqttMessage 物件，則可以在遞送完成之後對其進行記憶體回收。 在階段作業中斷連線後，可以對遞送記號進行記憶體回收。

在發佈訊息之後，您可以取得 IMqttDeliveryToken 及 MqttMessage 屬性。 如果您嘗試在發佈訊息之後設定任何 MqttMessage 屬性，則結果未定義。

如果 MQTT 用戶端使用相同的 ClientIdentifier 重新連接至前一個階段作業，則該用戶端會繼續處理遞送確認通知; 請參閱 第 1049 頁的『清除階段作業』。 對於前一個階段作業， MQTT 用戶端應用程式必須將 MqttClient.CleanSession 設為 false ，並在新階段作業中將它設為 false 。 MQTT 用戶端會在新階段作業中為擱置的遞送建立新的遞送記號及訊息物件。 它會使用 MqttClientPersistence 類別回復物件。 如果應用程式用戶端仍然具有對舊遞送記號和訊息的參照，請將它們解除參照。 對於在前一個階段作業起始、在此階段作業中完成的所有遞送，會呼叫應用程式回呼。

擱置的遞送完成後，會在應用程式用戶端連接後呼叫應用程式回呼。 在應用程式用戶端連接之前，它可以使用 MqttClient.getPendingDeliveryTokens 方法擷取擱置的遞送。

請注意，用戶端應用程式最初建立已發佈的訊息物件及其有效負載位元組陣列。 MQTT 用戶端會參照這些物件。 透過遞送記號在方法 token.getMessage 中傳回的訊息物件，不一定是用戶端建立的相同訊息物件。 如果新的 MQTT 用戶端實例重建遞送記號，MqttClientPersistence 類別便會重

建 MqttMessage 物件。 為了一致起見，如果 token.isCompleted 是 true，則 token.getMessage 傳回 null，不論訊息物件是由應用程式用戶端還是由 MqttClientPersistence 類別建立的。

messageArrived(String topic, MqttMessage message)

當發佈到達符合訂閱主題的用戶端時，會呼叫 messageArrived 。 topic 是發佈主題，而不是訂閱過濾器。 如果過濾器包含萬用字元，則兩者可能不同。

如果主題符合用戶端建立的多個訂閱，則用戶端會收到發佈的多個副本。 如果用戶端發佈至它訂閱的主題，則它會收到它自己的發佈的副本。

如果以 QoS 1 或 2 來傳送訊息，則在 MQTT 用戶端呼叫 messageArrived 之前，MqttClientPersistence 類別會儲存該訊息。 messageArrived 行為類似 deliveryComplete: 對於發佈只會呼叫一次，當 messageArrived 回到 MQTT 用戶端時，MqttClientPersistence.remove 會移除發佈的本端副本。 當 messageArrived 回到 MQTT 用戶端時， MQTT 用戶端會捨棄其對主題和訊息的參照。 如果應用程式用戶端未保留對物件的參照，則會對主題和訊息物件進行記憶體回收。

回呼、執行緒作業及用戶端應用程式同步化

MQTT 用戶端會在主要應用程式執行緒的個別執行緒上呼叫回呼方法。 用戶端應用程式不會建立回呼的執行緒，而是由 MQTT 用戶端建立。

MQTT 用戶端會同步化回呼方法。 一次只能執行回呼方法的一個實例。 同步處理可讓您輕鬆更新計算已遞送發佈的物件。 一次只能執行 MqttCallback.deliveryComplete 的一個實例，因此，可以安全地更新計算，無須進一步的同步處理。 一次也只有一個發佈到達。 messageArrived 方法中的程式碼可以更新物件，而不對它進行同步處理。 如果您要參閱另一個執行緒中的計算或者正在更新的物件，請對計算或物件進行同步處理。

遞送記號提供主要應用程式執行緒與發佈遞送之間的同步處理機制。 方法 token.waitForCompletion 會等待到特定發佈遞送完成，或者等待到選用逾時過期。 您可以用下列方式來使用 token.waitForCompletion , 一次處理一個發佈。

與 MqttCallback.deliveryComplete 方法同步。 只有在 MqttCallback.deliveryComplete 返回到 MQTT 用戶端後，token.waitForCompletion 才會回復。 您可以使用此機制，在主要應用程式執行緒中執行程式碼之前，對 MqttCallback.deliveryComplete 中正在執行的程式碼進行同步處理。

如果您想發佈而不等待遞送每個發佈，而是希望在所有發佈皆已遞送後確認，便會如何？ 如果您在單一執行緒上發佈，則會最後遞送最後一個要傳送的發佈。

傳送至伺服器的要求同步處理

第 1048 頁的表 188 說明 MQTT Java 用戶端中傳送要求至伺服器的方法。 除非應用程式用戶端設定無限期逾時，否則用戶端絕不會無限期等待伺服器。 如果用戶端當掉，可能是應用程式設計問題，或是 MQTT 用戶端中的問題。

表 *188:* 導致伺服器要求之方法的同步處理行為

方法	同步處理	逾時間隔
MqttClient.Connect	等待與伺服器建立連線。	預設為 30 秒，或由參數設定，則會擲出異常狀況。
MqttClient.Disconnect	等待 MQTT 用戶端完成它必須執行的任何工作，以及等待 TCP/IP 階段作業切斷連線。	
MqttClient.Subscribe	等待「訂閱」或 UnSubscribe 方法完成。	
MqttClient.UnSubscribe		
MqttClient.Publish	將要求傳遞至 MQTT 用戶端之後，立即返回應用程式執行緒。	無。

表 *188:* 導致伺服器要求之方法的同步處理行為 *(*繼續*)*		
方法	**同步處理**	**逾時間隔**
`IMqttDeliveryToken.waitForCompletion`	等待傳回遞送記號。	無限期或設為參數。

相關概念

清除階段作業

MQTT 用戶端及遙測 (MQXR) 服務會維護階段作業狀態資訊。 狀態資訊用來確保 "至少一次" 和 "正好一次" 遞送，以及 "正好一次" 接收發佈。 階段作業狀態也包括 MQTT 用戶端所建立的訂閱。 您可以選擇在階段作業之間執行具有或不具有維護狀態資訊的 MQTT 用戶端。 在連接之前設定 `MqttConnectOptions.cleanSession`，以變更清除階段作業模式。

用戶端 ID

用戶端 ID 是一個 23 個位元組的字串，用來識別 MQTT 用戶端。 對於一次僅一個已連接的用戶端來說，每一個 ID 都必須是唯一的。 ID 只能包含在佇列管理程式名稱中有效的字元。 在這些限制的範圍內，您可以使用任何識別字串。 具有配置用戶端 ID 的程序，以及透過用戶端的選定 ID 來配置用戶端的方法，是非常重要的。

遞送記號

最後留言發佈

如果 MQTT 用戶端連線非預期地結束，您可以配置 MQ Telemetry 以傳送「最後留言」發佈。 預先定義發佈的內容，以及要將發佈傳送至的主題。 「最後留言」是連線內容。 它應在連接用戶端之前建立。

MQTT 用戶端中的訊息持續性

如果以 "至少一次"或 "正好一次"的服務品質來傳送發佈訊息，則會持續保存發佈訊息。 您可以在用戶端上實作您自己的持續性機制，或者使用用戶端提供的預設持續性機制。 持續性對於兩個方向起作用：傳送至用戶端的發佈或用戶端傳送的發佈。

發佈

出版品是與主題字串相關聯的 `MqttMessage` 實例。 MQTT 用戶端可以建立發佈資訊以傳送至 IBM MQ，並訂閱 IBM MQ 上的主題以接收發佈資訊。

MQTT 用戶端提供的服務品質

MQTT 用戶端提供三種服務品質，可將發佈遞送至 IBM MQ 及 MQTT 用戶端:「最多一次」、「至少一次」及「正好一次」。 MQTT 用戶端將要求傳送至 IBM MQ 以建立訂閱時，會以「至少一次」服務品質傳送要求。

保留的發佈及 MQTT 用戶端

主題可以有一個 (且只能有一個) 保留的發佈資訊。 如果您建立具有保留發佈資訊之主題的訂閱，則會立即將發佈資訊轉遞給您。

訂閱

建立訂閱，以使用主題過濾器註冊所需的發佈主題。 用戶端可以建立多個訂閱，或者建立包含使用萬用字元的主題過濾器的訂閱，以註冊所需的多個主題。 會將主題上符合過濾器的發佈傳送至用戶端。 用戶端中斷連線時，訂閱可以保持處於作用中狀態。 用戶端重新連接後，會將發佈傳送至用戶端。

MQTT 用戶端中的主題字串及主題過濾器

主題字串和主題過濾器用於發佈和訂閱。 MQTT 用戶端中主題字串和過濾器的語法，與 IBM MQ 中的主題字串語法大致相同。

清除階段作業

MQTT 用戶端及遙測 (MQXR) 服務會維護階段作業狀態資訊。 狀態資訊用來確保 "至少一次" 和 "正好一次" 遞送，以及 "正好一次" 接收發佈。 階段作業狀態也包括 MQTT 用戶端所建立的訂閱。 您可以選擇在階段作業之間執行具有或不具有維護狀態資訊的 MQTT 用戶端。 在連接之前設定 `MqttConnectOptions.cleanSession`，以變更清除階段作業模式。

當您使用 `MqttClient.connect` 方法連接 MQTT 用戶端應用程式時，用戶端會使用用戶端 ID 及伺服器位址來識別連線。 伺服器會檢查是否已將階段作業資訊從前一個連線儲存到伺服器。 如果前一個階段作業仍然存在，而且 `cleanSession=true`，則會在用戶端和伺服器上清除前一個階段作業資訊。 如果 `cleanSession=false`，則會回復前一個階段作業。 如果沒有前一個階段作業，則會開始新的階段作業。

註:「IBM MQ 管理者」可以強制關閉開啟的階段作業，並刪除所有階段作業資訊。 如果在 `cleanSession=false` 的情況下重新開啟階段作業，則會開始新的階段作業。

發佈

如果您使用預設 `MqttConnectOptions`，或者在連接用戶端之前將 `MqttConnectOptions.cleanSession` 設為 `true`，則在用戶端連接時，會移除用戶端的所有擱置中發佈遞送。

清除階段作業設定不會影響利用 `QoS=0` 傳送的發佈。 對於 `QoS=1` 和 `QoS=2`，使用 `cleanSession=true` 可能會導致遺失發佈。

訂閱

在連接用戶端之前，如果您使用預設 `MqttConnectOptions`，或將 `MqttConnectOptions.cleanSession` 設為 `true` ，則在用戶端連接時，會移除用戶端的任何舊訂閱。 用戶端在階段作業期間建立的所有新訂閱，皆會在連線中斷後移除。

如果您在連接之前將 `MqttConnectOptions.cleanSession` 設為 `false` ，則用戶端建立的任何訂閱都會新增至用戶端連接之前存在的所有訂閱。 用戶端中斷連線時，所有訂閱都保持為作用中狀態。

另一種瞭解 `cleanSession` 屬性如何影響訂閱的方式，是將它視為限制模式的屬性。 在其預設模式 `cleanSession=true` 中，用戶端僅在階段作業的範圍內建立訂閱並接收發佈。 在替代模式 `cleanSession=false` 中，訂閱是可延續的。 用戶端可以連接和中斷連線，其訂閱會保持為作用中狀態。 用戶端重新連接時，它會接收所有未遞送的發佈。 它連接時，可以自行修改作用中的訂閱集。

您必須在連接之前設定 `cleanSession` 模式; 此模式會持續整個階段作業。 若要變更其設定，您必須中斷連線，然後重新連接用戶端。 如果您將模式從使用 `cleanSession=false` 變更為 `cleanSession=true`，則會捨棄用戶端的所有先前訂閱，以及尚未收到的任何發佈。

相關概念

MQTT 用戶端應用程式中的回呼及同步化
MQTT 用戶端程式設計模型廣泛使用執行緒。 執行緒會盡可能取消 MQTT 用戶端應用程式與伺服器之間傳輸訊息的延遲之間的連結。 發佈、遞送記號及連線遺失事件會遞送至實作 `MqttCallback` 的回呼類別中的方法。

用戶端 ID
用戶端 ID 是一個 23 個位元組的字串，用來識別 MQTT 用戶端。 對於一次僅一個已連接的用戶端來說，每一個 ID 都必須是唯一的。 ID 只能包含在佇列管理程式名稱中有效的字元。 在這些限制的範圍內，您可以使用任何識別字串。 具有配置用戶端 ID 的程序，以及透過用戶端的選定 ID 來配置用戶端的方法，是非常重要的。

遞送記號

最後留言發佈
如果 MQTT 用戶端連線非預期地結束，您可以配置 MQ Telemetry 以傳送「最後留言」發佈。 預先定義發佈的內容，以及要將發佈傳送至的主題。 「最後留言」是連線內容。 它應在連接用戶端之前建立。

MQTT 用戶端中的訊息持續性
如果以 "至少一次"或 "正好一次"的服務品質來傳送發佈訊息，則會持續保存發佈訊息。 您可以在用戶端上實作您自己的持續性機制，或者使用用戶端提供的預設持續性機制。 持續性對於兩個方向起作用：傳送至用戶端的發佈或用戶端傳送的發佈。

發佈
出版品是與主題字串相關聯的 `MqttMessage` 實例。 MQTT 用戶端可以建立發佈資訊以傳送至 IBM MQ，並訂閱 IBM MQ 上的主題以接收發佈資訊。

MQTT 用戶端提供的服務品質
MQTT 用戶端提供三種服務品質，可將發佈遞送至 IBM MQ 及 MQTT 用戶端:「最多一次」、「至少一次」及「正好一次」。 MQTT 用戶端將要求傳送至 IBM MQ 以建立訂閱時，會以「至少一次」服務品質傳送要求。

保留的發佈及 MQTT 用戶端

主題可以有一個 (且只能有一個) 保留的發佈資訊。 如果您建立具有保留發佈資訊之主題的訂閱，則會立即將發佈資訊轉遞給您。

訂閱
建立訂閱，以使用主題過濾器註冊所需的發佈主題。 用戶端可以建立多個訂閱，或者建立包含使用萬用字元的主題過濾器的訂閱，以註冊所需的多個主題。 會將主題上符合過濾器的發佈傳送至用戶端。 用戶端中斷連線時，訂閱可以保持處於作用中狀態。 用戶端重新連接後，會將發佈傳送至用戶端。

MQTT 用戶端中的主題字串及主題過濾器
主題字串和主題過濾器用於發佈和訂閱。 MQTT 用戶端中主題字串和過濾器的語法，與 IBM MQ 中的主題字串語法大致相同。

用戶端 ID

用戶端 ID 是一個 23 個位元組的字串，用來識別 MQTT 用戶端。 對於一次僅一個已連接的用戶端來說，每一個 ID 都必須是唯一的。 ID 只能包含在佇列管理程式名稱中有效的字元。 在這些限制的範圍內，您可以使用任何識別字串。 具有配置用戶端 ID 的程序，以及透過用戶端的選定 ID 來配置用戶端的方法，是非常重要的。

用戶端 ID 用於管理 MQTT 系統。 由於可能需要管理幾十萬個用戶端，您需要能夠迅速識別特定用戶端。 例如，假設裝置發生故障，且您收到通知，可能是客戶打電話給服務台。 客戶必須能夠識別裝置，且您必須能夠將該識別與通常連接至用戶端的伺服器產生關聯。

當您瀏覽 MQTT 用戶端連線時，會以用戶端 ID 標示每一個連線。 若要協助決定如何最好地將此 ID 對映至裝置及伺服器，請詢問您自己下列問題:

- 維護及使用將每一個裝置對映至用戶端 ID 及伺服器的資料庫是否方便?
- 裝置名稱是否可以識別它所連接的伺服器?
- 您需要將用戶端 ID 對映至實體裝置的查閱表格嗎?
- 用戶端 ID 是否識別在用戶端上執行的特定裝置、使用者或應用程式?
- 如果客戶將故障裝置取代為新裝置，則新裝置是否具有與舊裝置相同的 ID ，或者您是否配置新 ID? (如果您變更實體裝置並保留相同的 ID ，則未完成的發佈及作用中訂閱會自動傳送至新裝置。)

您還需要系統來確保用戶端 ID 是唯一的，並且您必須具有在用戶端上設定 ID 的可靠處理程序。 如果用戶端裝置是「黑盒」，且沒有使用者介面，則您可以使用用戶端 ID 來製造裝置，或者您可以具有軟體安裝及配置處理程序，以在裝置啟動之前對其進行配置。

若要保持 ID 簡短且唯一，您可以從 48 位元裝置 MAC 位址建立用戶端 ID。 如果傳輸大小不是嚴重問題，您可以使用剩餘的 17 個位元組，讓位址更容易管理。

相關概念
MQTT 用戶端應用程式中的回呼及同步化
MQTT 用戶端程式設計模型廣泛使用執行緒。 執行緒會盡可能取消 MQTT 用戶端應用程式與伺服器之間傳輸訊息的延遲之間的連結。 發佈、遞送記號及連線遺失事件會遞送至實作 `MqttCallback` 的回呼類別中的方法。

清除階段作業
MQTT 用戶端及遙測 (MQXR) 服務會維護階段作業狀態資訊。 狀態資訊用來確保 "至少一次" 和 "正好一次" 遞送，以及 "正好一次" 接收發佈。 階段作業狀態也包括 MQTT 用戶端所建立的訂閱。 您可以選擇在階段作業之間執行具有或不具有維護狀態資訊的 MQTT 用戶端。 在連接之前設定 `MqttConnectOptions.cleanSession` ，以變更清除階段作業模式。

遞送記號

最後留言發佈
如果 MQTT 用戶端連線非預期地結束，您可以配置 MQ Telemetry 以傳送「最後留言」發佈。 預先定義發佈的內容，以及要將發佈傳送至的主題。 「最後留言」是連線內容。 它應在連接用戶端之前建立。

MQTT 用戶端中的訊息持續性
如果以 "至少一次"或 "正好一次"的服務品質來傳送發佈訊息，則會持續保存發佈訊息。 您可以在用戶端上實作您自己的持續性機制，或者使用用戶端提供的預設持續性機制。 持續性對於兩個方向起作用：傳送至用戶端的發佈或用戶端傳送的發佈。

發佈

出版品是與主題字串相關聯的 MqttMessage 實例。 MQTT 用戶端可以建立發佈資訊以傳送至 IBM MQ，並訂閱 IBM MQ 上的主題以接收發佈資訊。

MQTT 用戶端提供的服務品質
MQTT 用戶端提供三種服務品質，可將發佈遞送至 IBM MQ 及 MQTT 用戶端:「最多一次」、「至少一次」及「正好一次」。 MQTT 用戶端將要求傳送至 IBM MQ 以建立訂閱時，會以「至少一次」服務品質傳送要求。

保留的發佈及 MQTT 用戶端
主題可以有一個 (且只能有一個) 保留的發佈資訊。 如果您建立具有保留發佈資訊之主題的訂閱，則會立即將發佈資訊轉遞給您。

訂閱
建立訂閱，以使用主題過濾器註冊所需的發佈主題。 用戶端可以建立多個訂閱，或者建立包含使用萬用字元的主題過濾器的訂閱，以註冊所需的多個主題。 會將主題上符合過濾器的發佈傳送至用戶端。 用戶端中斷連線時，訂閱可以保持處於作用中狀態。 用戶端重新連接後，會將發佈傳送至用戶端。

MQTT 用戶端中的主題字串及主題過濾器
主題字串和主題過濾器用於發佈和訂閱。 MQTT 用戶端中主題字串和過濾器的語法，與 IBM MQ 中的主題字串語法大致相同。

遞送記號

用戶端在主題上發佈時，便會建立新的遞送記號。 請使用遞送記號來監視發佈資訊的遞送，或封鎖用戶端應用程式直到遞送完成為止。

記號是 MqttDeliveryToken 物件。 記號透過呼叫 MqttTopic.publish() 方法建立並由 MQTT 用戶端保留，直到用戶端階段作業斷開連線並且遞送完成為止。

記號的一般用途是檢查遞送是否完成。 使用傳回的記號來呼叫 token.waitForCompletion，以封鎖用戶端應用程式，直到遞送完成為止。 此外，提供了 MqttCallBack 處理程式。 MQTT 用戶端收到它預期的遞送發佈的所有確認通知後，它便會呼叫 MqttCallBack.deliveryComplete 並傳遞遞送記號作為參數。

在遞送完成之前，您可以使用呼叫 token.getMessage 傳回的遞送記號檢查發佈。

已完成遞送

完成遞送為非同步作業，並且取決於與發佈相關聯的服務品質。

至多一次

QoS=0
從 MqttTopic.publish 返回後，遞送立即完成。 會立即呼叫 MqttCallback.deliveryComplete。

至少一次

QoS=1
收到佇列管理程式的發佈確認通知後，遞送即告完成。 會在收到確認通知時呼叫 MqttCallback.deliveryComplete。 如果通訊速度慢或者不可靠，則可能會在呼叫 MqttCallback.deliveryComplete 之前多次遞送訊息。

只一次

QoS=2
用戶端收到發佈已發佈至訂閱者的完成訊息時，遞送即告完成。 一旦收到發佈訊息，便會呼叫 MqttCallback.deliveryComplete。 系統不會等待完成訊息。

在極少數情況下，您的用戶端應用程式可能不會正常從 MqttCallback.deliveryComplete 回到 MQTT 用戶端。 您得以知道遞送已完成，是因為已呼叫 MqttCallback.deliveryComplete。 如果用戶端重新啟動相同的階段作業，則不會再次呼叫 MqttCallback.deliveryComplete。

未完成遞送

如果在用戶端階段作業中斷連線之後，遞送尚未完成，則您可以再次連接用戶端並完成遞送。 只有在將 `MqttConnectionOptions` 屬性設為 `false` 的情況下在階段作業中發佈訊息時，才能完成訊息遞送。

使用相同的用戶端 ID 和伺服器位址建立用戶端，然後連接，並再次將 `cleanSession` `MqttConnectionOptions` 屬性設為 `false`。 如果將 `cleanSession` 設為 `true`，則會捨棄擱置的遞送記號。

透過呼叫 `MqttClient.getPendingDeliveryTokens`，您可以檢查是否有擱置的遞送。 可以在連接用戶端之前呼叫 `MqttClient.getPendingDeliveryTokens`。

相關概念

MQTT 用戶端應用程式中的回呼及同步化
MQTT 用戶端程式設計模型廣泛使用執行緒。 執行緒會盡可能取消 MQTT 用戶端應用程式與伺服器之間傳輸訊息的延遲之間的連結。 發佈、遞送記號及連線遺失事件會遞送至實作 `MqttCallback` 的回呼類別中的方法。

清除階段作業
MQTT 用戶端及遙測 (MQXR) 服務會維護階段作業狀態資訊。 狀態資訊用來確保 "至少一次" 和 "正好一次" 遞送，以及 "正好一次" 接收發佈。 階段作業狀態也包括 MQTT 用戶端所建立的訂閱。 您可以選擇在階段作業之間執行具有或不具有維護狀態資訊的 MQTT 用戶端。 在連接之前設定 `MqttConnectOptions.cleanSession` ，以變更清除階段作業模式。

用戶端 ID
用戶端 ID 是一個 23 個位元組的字串，用來識別 MQTT 用戶端。 對於一次僅一個已連接的用戶端來說，每一個 ID 都必須是唯一的。 ID 只能包含在佇列管理程式名稱中有效的字元。 在這些限制的範圍內，您可以使用任何識別字串。 具有配置用戶端 ID 的程序，以及透過用戶端的選定 ID 來配置用戶端的方法，是非常重要的。

最後留言發佈
如果 MQTT 用戶端連線非預期地結束，您可以配置 MQ Telemetry 以傳送「最後留言」發佈。 預先定義發佈的內容，以及要將發佈傳送至的主題。 「最後留言」是連線內容。 它應在連接用戶端之前建立。

MQTT 用戶端中的訊息持續性
如果以 "至少一次"或 "正好一次"的服務品質來傳送發佈訊息，則會持續保存發佈訊息。 您可以在用戶端上實作您自己的持續性機制，或者使用用戶端提供的預設持續性機制。 持續性對於兩個方向起作用：傳送至用戶端的發佈或用戶端傳送的發佈。

發佈
出版品是與主題字串相關聯的 `MqttMessage` 實例。 MQTT 用戶端可以建立發佈資訊以傳送至 IBM MQ，並訂閱 IBM MQ 上的主題以接收發佈資訊。

MQTT 用戶端提供的服務品質
MQTT 用戶端提供三種服務品質，可將發佈遞送至 IBM MQ 及 MQTT 用戶端:「最多一次」、「至少一次」及「正好一次」。 MQTT 用戶端將要求傳送至 IBM MQ 以建立訂閱時，會以「至少一次」服務品質傳送要求。

保留的發佈及 MQTT 用戶端
主題可以有一個 (且只能有一個) 保留的發佈資訊。 如果您建立具有保留發佈資訊之主題的訂閱，則會立即將發佈資訊轉遞給您。

訂閱
建立訂閱，以使用主題過濾器註冊所需的發佈主題。 用戶端可以建立多個訂閱，或者建立包含使用萬用字元的主題過濾器的訂閱，以註冊所需的多個主題。 會將主題上符合過濾器的發佈傳送至用戶端。 用戶端中斷連線時，訂閱可以保持處於作用中狀態。 用戶端重新連接後，會將發佈傳送至用戶端。

MQTT 用戶端中的主題字串及主題過濾器
主題字串和主題過濾器用於發佈和訂閱。 MQTT 用戶端中主題字串和過濾器的語法，與 IBM MQ 中的主題字串語法大致相同。

最後留言發佈

如果 MQTT 用戶端連線非預期地結束，您可以配置 MQ Telemetry 以傳送「最後留言」發佈。 預先定義發佈的內容，以及要將發佈傳送至的主題。 「最後留言」是連線內容。 它應在連接用戶端之前建立。

為最後留言建立主題。 您可以建立主題，例如 MQTTManagement/Connections/*server URI*/*client identifer*/Lost。

使用 MqttConnectionOptions.setWill(MqttTopic lastWillTopic, byte [] lastWillPayload, int lastWillQos, boolean lastWillRetained) 方法來設定「最後留言」。

考量在 lastWillPayload 訊息中建立時間戳記。 包括可協助識別用戶端和連線情況的其他用戶端資訊。 將 MqttConnectionOptions 物件傳遞至 MqttClient 建構子。

將 lastWillQos 設為 1 或 2，使訊息在 IBM MQ 中持續保存，並保證遞送。 若要保留前次中斷的連線資訊，請將 lastWillRetained 設為 true。

如果連線非預期地結束，便會將「最後留言」發佈傳送至訂閱者。 如果連線不是因為用戶端呼叫 MqttClient.disconnect 方法而結束，便會傳送最後留言。

若要監視連線，請利用其他發佈補充「最後留言」發佈，以記錄連線和程式化的斷線。

相關概念

MQTT 用戶端應用程式中的回呼及同步化
MQTT 用戶端程式設計模型廣泛使用執行緒。 執行緒會盡可能取消 MQTT 用戶端應用程式與伺服器之間傳輸訊息的延遲之間的連結。 發佈、遞送記號及連線遺失事件會遞送至實作 MqttCallback 的回呼類別中的方法。

清除階段作業
MQTT 用戶端及遙測 (MQXR) 服務會維護階段作業狀態資訊。 狀態資訊用來確保 "至少一次" 和 "正好一次" 遞送，以及 "正好一次" 接收發佈。 階段作業狀態也包括 MQTT 用戶端所建立的訂閱。 您可以選擇在階段作業之間執行具有或不具有維護狀態資訊的 MQTT 用戶端。 在連接之前設定 MqttConnectOptions.cleanSession，以變更清除階段作業模式。

用戶端 ID
用戶端 ID 是一個 23 個位元組的字串，用來識別 MQTT 用戶端。 對於一次僅一個已連接的用戶端來說，每一個 ID 都必須是唯一的。 ID 只能包含在佇列管理程式名稱中有效的字元。 在這些限制的範圍內，您可以使用任何識別字串。 具有配置用戶端 ID 的程序，以及透過用戶端的選定 ID 來配置用戶端的方法，是非常重要的。

遞送記號

MQTT 用戶端中的訊息持續性
如果以 "至少一次"或 "正好一次"的服務品質來傳送發佈訊息，則會持續保存發佈訊息。 您可以在用戶端上實作您自己的持續性機制，或者使用用戶端提供的預設持續性機制。 持續性對於兩個方向起作用：傳送至用戶端的發佈或用戶端傳送的發佈。

發佈
出版品是與主題字串相關聯的 MqttMessage 實例。 MQTT 用戶端可以建立發佈資訊以傳送至 IBM MQ，並訂閱 IBM MQ 上的主題以接收發佈資訊。

MQTT 用戶端提供的服務品質
MQTT 用戶端提供三種服務品質，可將發佈遞送至 IBM MQ 及 MQTT 用戶端:「最多一次」、「至少一次」及「正好一次」。 MQTT 用戶端將要求傳送至 IBM MQ 以建立訂閱時，會以「至少一次」服務品質傳送要求。

保留的發佈及 MQTT 用戶端
主題可以有一個 (且只能有一個) 保留的發佈資訊。 如果您建立具有保留發佈資訊之主題的訂閱，則會立即將發佈資訊轉遞給您。

訂閱
建立訂閱，以使用主題過濾器註冊所需的發佈主題。 用戶端可以建立多個訂閱，或者建立包含使用萬用字元的主題過濾器的訂閱，以註冊所需的多個主題。 會將主題上符合過濾器的發佈傳送至用戶端。 用戶端中斷連線時，訂閱可以保持處於作用中狀態。 用戶端重新連接後，會將發佈傳送至用戶端。

MQTT 用戶端中的主題字串及主題過濾器
主題字串和主題過濾器用於發佈和訂閱。 MQTT 用戶端中主題字串和過濾器的語法，與 IBM MQ 中的主題字串語法大致相同。

MQTT 用戶端中的訊息持續性

如果以 "至少一次"或 "正好一次"的服務品質來傳送發佈訊息，則會持續保存發佈訊息。 您可以在用戶端上實作您自己的持續性機制，或者使用用戶端提供的預設持續性機制。 持續性對於兩個方向起作用：傳送至用戶端的發佈或用戶端傳送的發佈。

在 MQTT 中，訊息持續性有兩個層面: 如何傳送訊息，以及它是否在 IBM MQ 中作為持續訊息排入佇列。

1. MQTT 用戶端會連結訊息持續性與服務品質。 將會根據您為訊息選擇的服務品質，持續保存訊息。 訊息持續性是實作必要的服務品質所必需的。

   如果您指定 "最多一次"(QoS=0) ，則用戶端會在訊息發佈後立即捨棄它。 如果在訊息的上游處理過程中發生故障，不會再次傳送該訊息。 即使用戶端保持作用中狀態，也不會再次傳送該訊息。 QoS=0 訊息的行為與 IBM MQ 快速非持續訊息相同。

   如果用戶端利用 QoS 1 或 2 發佈訊息，則會使該訊息持續。 訊息會儲存在本端，且只有在不再需要保證 "至少一次"、 QoS=1 或 "正好一次"、 QoS=2 遞送時，才會從用戶端捨棄。
2. 如果訊息標示為 QoS 1 或 2 ，則會在 IBM MQ 中作為持續訊息排入佇列。 如果它標示為 QoS=0，則會在 IBM MQ 中作為非持續訊息排入佇列。 在 IBM MQ 中，除非訊息通道將 NPMSPEED 屬性設為 FAST，否則在佇列管理程式之間傳送非持續訊息 "只一次"。

持續發佈會儲存在用戶端上，直到用戶端應用程式收到為止。 對於 QoS=2，應用程式回呼交還控制權後，便會從用戶端捨棄發佈。 對於 QoS=1，如果發生故障，應用程式可以再次接收發佈。 對於 QoS=0，回呼會最多接收發佈一次。 如果發生故障，或者如果發佈時用戶端中斷連線，則它可能無法收到發佈。

訂閱主題時，您可以降低訂閱者接收訊息所用的 QoS，以與其持續性功能相配。 以更高 QoS 建立的發佈會以訂閱者所要求的最高 QoS 來傳送。

儲存訊息

在小裝置上，資料儲存體的實作大相逕庭。 在 MQTT 用戶端所管理的儲存體中暫時儲存持續訊息的模型可能太慢，或需要太多儲存體。 在行動式裝置中，行動式作業系統可能提供適用於 MQTT 訊息的儲存體服務。

為了提供滿足小型裝置限制的彈性， MQTT 用戶端有兩個持續性介面。 這些介面定義儲存持續訊息所涉及的作業。 這些介面在 MQTT client for Java 的 API 文件會有所說明。 如需 MQTT 用戶端程式庫的用戶端 API 文件鏈結，請參閱 MQTT 用戶端程式設計參考手冊。您可以實作這些介面來滿足裝置的需求。 在 Java SE 上執行的 MQTT 用戶端具有在檔案系統中儲存持續訊息之介面的預設實作。 它使用 `java.io` 套件。

持續性類別

MqttClientPersistence
將 MqttClientPersistence 實作的實例傳遞至 MQTT 用戶端作為 MqttClient 建構子的參數。 如果您在 MqttClient 建構子中省略 MqttClientPersistence 參數， MQTT 用戶端會使用 MqttDefaultFilePersistence 類別來儲存持續訊息。

MqttPersistable
MqttClientPersistence 會使用儲存體金鑰取得和放置 MqttPersistable 物件。 如果您不是使用 MqttDefaultFilePersistence，則必須提供 MqttPersistable 的實作以及 MqttClientPersistence 的實作。

MqttDefaultFilePersistence

MQTT 用戶端提供了 MqttDefaultFilePersistence 類別。 如果您在用戶端應用程式中實例化 MqttDefaultFilePersistence ，則可以提供目錄以將持續訊息儲存為 MqttDefaultFilePersistence 建構子的參數。

或者， MQTT 用戶端可以實例化 MqttDefaultFilePersistence ，並將檔案放置在下列預設目錄中:

client identifier -tcp *hostname portnumber*

會從目錄名稱字串中移除下列字元:

"\", "\\", "/", ":" 及 " "

目錄的路徑是系統內容 rcp.data 的值; 如果未設定 rcp.data，則路徑是系統內容 usr.data 的值，其中

- rcp.data 是與 OSGi 或 Eclipse Rich Client Platform (RCP) 安裝相關聯的內容。
- usr.data 是啟動應用程式的 Java 指令所在的目錄。

相關概念

MQTT 用戶端應用程式中的回呼及同步化

MQTT 用戶端程式設計模型廣泛使用執行緒。 執行緒會盡可能取消 MQTT 用戶端應用程式與伺服器之間傳輸訊息的延遲之間的連結。 發佈、遞送記號及連線遺失事件會遞送至實作 MqttCallback 的回呼類別中的方法。

清除階段作業

MQTT 用戶端及遙測 (MQXR) 服務會維護階段作業狀態資訊。 狀態資訊用來確保 "至少一次" 和 "正好一次" 遞送，以及 "正好一次" 接收發佈。 階段作業狀態也包括 MQTT 用戶端所建立的訂閱。 您可以選擇在階段作業之間執行具有或不具有維護狀態資訊的 MQTT 用戶端。 在連接之前設定 MqttConnectOptions.cleanSession，以變更清除階段作業模式。

用戶端 ID

用戶端 ID 是一個 23 個位元組的字串，用來識別 MQTT 用戶端。 對於一次僅一個已連接的用戶端來說，每一個 ID 都必須是唯一的。 ID 只能包含在佇列管理程式名稱中有效的字元。 在這些限制的範圍內，您可以使用任何識別字串。 具有配置用戶端 ID 的程序，以及透過用戶端的選定 ID 來配置用戶端的方法，是非常重要的。

遞送記號

最後留言發佈

如果 MQTT 用戶端連線非預期地結束，您可以配置 MQ Telemetry 以傳送「最後留言」發佈。 預先定義發佈的內容，以及要將發佈傳送至的主題。 「最後留言」是連線內容。 它應在連接用戶端之前建立。

發佈

出版品是與主題字串相關聯的 MqttMessage 實例。 MQTT 用戶端可以建立發佈資訊以傳送至 IBM MQ，並訂閱 IBM MQ 上的主題以接收發佈資訊。

MQTT 用戶端提供的服務品質

MQTT 用戶端提供三種服務品質，可將發佈遞送至 IBM MQ 及 MQTT 用戶端:「最多一次」、「至少一次」及「正好一次」。 MQTT 用戶端將要求傳送至 IBM MQ 以建立訂閱時，會以「至少一次」服務品質傳送要求。

保留的發佈及 MQTT 用戶端

主題可以有一個 (且只能有一個) 保留的發佈資訊。 如果您建立具有保留發佈資訊之主題的訂閱，則會立即將發佈資訊轉遞給您。

訂閱

建立訂閱，以使用主題過濾器註冊所需的發佈主題。 用戶端可以建立多個訂閱，或者建立包含使用萬用字元的主題過濾器的訂閱，以註冊所需的多個主題。 會將主題上符合過濾器的發佈傳送至用戶端。 用戶端中斷連線時，訂閱可以保持處於作用中狀態。 用戶端重新連接後，會將發佈傳送至用戶端。

MQTT 用戶端中的主題字串及主題過濾器

主題字串和主題過濾器用於發佈和訂閱。 MQTT 用戶端中主題字串和過濾器的語法，與 IBM MQ 中的主題字串語法大致相同。

發佈

出版品是與主題字串相關聯的 MqttMessage 實例。 MQTT 用戶端可以建立發佈資訊以傳送至 IBM MQ，並訂閱 IBM MQ 上的主題以接收發佈資訊。

MqttMessage 使用位元組陣列作為其內容。 法還會將用於發佈訊息的主題字串傳送至訂閱者。 MQTT protocol 允許的訊息長度上限為 250 MB。

MQTT 用戶端程式一般使用 java.lang.String 或 java.lang.StringBuffer 來操作訊息內容。 為了方便起見，MqttMessage 類別使用 toString 方法來將其內容轉換為字串。 若要從 java.lang.String 或 java.lang.StringBuffer 建立位元組陣列內容，請使用 getBytes 方法。

getBytes 方法會將字串轉換為平台的預設字集。 預設字集通常是 UTF-8。 只包含文字的 MQTT 發佈通常使用 UTF-8 編碼。 使用方法 getBytes("UTF8") 來置換預設字集。

在 IBM MQ 中，MQTT 發佈以 jms-bytes 訊息形式接收。 此訊息包括包含 <mqtt>的 MQRFH2 資料夾，以及 <mqps> 資料夾。 <mqtt> 資料夾包含 clientId、 msgId 及 qos，但此內容未來可能會變更。

MqttMessage 具有三個附加屬性：服務品質、是否保留它以及它是否可重複。 只有在服務品質是「至少一次」或「只一次」時，才會設定重複旗標。 如果先前已傳送訊息，並且 MQTT 用戶端確認不夠迅速，則會再次傳送該訊息，並將重複屬性設定為 true。

發佈

若要在 MQTT 用戶端應用程式中建立發佈，請建立 MqttMessage。 設定其內容、服務品質以及是否保留它，然後呼叫 MqttTopic.publish(MqttMessage message) 方法；會傳回 MqttDeliveryToken 並且完成發佈是非同步作業。

或者， MQTT 用戶端可以在建立發佈時，從 MqttTopic.publish(byte [] payload, int qos, boolean retained) 方法上的參數為您建立暫時訊息物件。

如果發佈的服務品質是「至少一次」或「只一次」（即 QoS=1 或 QoS=2）， MQTT 用戶端會呼叫 MqttClientPersistence 介面。 它會先呼叫 MqttClientPersistence 以儲存訊息，再將遞送記號傳回至應用程式。

應用程式可以選擇使用 MqttDeliveryToken.waitForCompletion 方法進行封鎖，直到將訊息遞送至伺服器為止。 此外，應用程式也可以繼續執行而不封鎖。 如果您想要檢查是否已遞送發佈，而不封鎖，請向 MQTT 用戶端登錄實作 MqttCallback 的回呼類別實例。 一旦發佈遞送完成， MQTT 用戶端便會呼叫 MqttCallback.deliveryComplete 方法。 根據服務品質，對於 QoS=0，遞送幾乎是立即進行，而對於 QoS=2，遞送可能需要延遲一些時間。

使用 MqttDeliveryToken.isComplete 方法來輪詢遞送是否已完成。 MqttDeliveryToken.isComplete 的值是 false 時，您可以呼叫 MqttDeliveryToken.getMessage 以取得訊息內容。 如果呼叫 MqttDeliveryToken.isComplete 的結果是 true，則已捨棄訊息，而且呼叫 MqttDeliveryToken.getMessage 會擲出空值指標異常狀況。 MqttDeliveryToken.getMessage 與 MqttDeliveryToken.isComplete 之間沒有內建同步處理作業。

如果用戶端在收到所有擱置的遞送記號之前便已中斷連線，則該用戶端的新實例可以在連接之前，查詢擱置的遞送記號。 用戶端連接之前，不會完成新的遞送，因此可以安全地呼叫 MqttDeliveryToken.getMessage。 使用 MqttDeliveryToken.getMessage 方法可以瞭解尚未遞送哪些發佈。 如果在將 MqttConnectOptions.cleanSession 設為其預設值 true 的情況下連接，則會捨棄擱置的遞送記號。

訂閱

佇列管理程式負責建立要傳送至 MQTT 訂閱者的發佈。 佇列管理程式會檢查 MQTT 用戶端所建立訂閱中的主題過濾器，是否符合發佈中的主題字串。 符合可以是完全相符，符合也可以包括萬用字元。 在佇列管理程式將發佈轉遞至訂閱者之前，佇列管理程式會檢查與發佈相關聯的主題屬性。 它會遵循使用包含萬用字元的主題字串來訂閱中說明的搜尋程序，來識別管理主題物件是否授與使用者訂閱權限。

MQTT 用戶端收到服務品質為「至少一次」的發佈時，便會呼叫 MqttCallback.messageArrived 方法以處理發佈。 如果發佈的服務品質是「只一次」(QoS=2)，則 MQTT 用戶端會呼叫 MqttClientPersistence 介面以儲存它收到的訊息。 然後，它會呼叫 MqttCallback.messageArrived。

相關概念

MQTT 用戶端應用程式中的回呼及同步化
MQTT 用戶端程式設計模型廣泛使用執行緒。 執行緒會盡可能取消 MQTT 用戶端應用程式與伺服器之間傳輸訊息的延遲之間的連結。 發佈、遞送記號及連線遺失事件會遞送至實作 MqttCallback 的回呼類別中的方法。

清除階段作業
MQTT 用戶端及遙測 (MQXR) 服務會維護階段作業狀態資訊。 狀態資訊用來確保 "至少一次" 和 "正好一次" 遞送，以及 "正好一次" 接收發佈。 階段作業狀態也包括 MQTT 用戶端所建立的訂閱。 您可以選擇在階段作

業之間執行具有或不具有維護狀態資訊的 MQTT 用戶端。 在連接之前設定 MqttConnectOptions.cleanSession，以變更清除階段作業模式。

用戶端 ID
用戶端 ID 是一個 23 個位元組的字串，用來識別 MQTT 用戶端。 對於一次僅一個已連接的用戶端來說，每一個 ID 都必須是唯一的。 ID 只能包含在佇列管理程式名稱中有效的字元。 在這些限制的範圍內，您可以使用任何識別字串。 具有配置用戶端 ID 的程序，以及透過用戶端的選定 ID 來配置用戶端的方法，是非常重要的。

遞送記號

最後留言發佈
如果 MQTT 用戶端連線非預期地結束，您可以配置 MQ Telemetry 以傳送「最後留言」發佈。 預先定義發佈的內容，以及要將發佈傳送至的主題。 「最後留言」是連線內容。 它應在連接用戶端之前建立。

MQTT 用戶端中的訊息持續性
如果以 "至少一次"或 "正好一次"的服務品質來傳送發佈訊息，則會持續保存發佈訊息。 您可以在用戶端上實作您自己的持續性機制，或者使用用戶端提供的預設持續性機制。 持續性對於兩個方向起作用：傳送至用戶端的發佈或用戶端傳送的發佈。

MQTT 用戶端提供的服務品質
MQTT 用戶端提供三種服務品質，可將發佈遞送至 IBM MQ 及 MQTT 用戶端:「最多一次」、「至少一次」及「正好一次」。 MQTT 用戶端將要求傳送至 IBM MQ 以建立訂閱時，會以「至少一次」服務品質傳送要求。

保留的發佈及 MQTT 用戶端
主題可以有一個 (且只能有一個) 保留的發佈資訊。 如果您建立具有保留發佈資訊之主題的訂閱，則會立即將發佈資訊轉遞給您。

訂閱
建立訂閱，以使用主題過濾器註冊所需的發佈主題。 用戶端可以建立多個訂閱，或者建立包含使用萬用字元的主題過濾器的訂閱，以註冊所需的多個主題。 會將主題上符合過濾器的發佈傳送至用戶端。 用戶端中斷連線時，訂閱可以保持處於作用中狀態。 用戶端重新連接後，會將發佈傳送至用戶端。

MQTT 用戶端中的主題字串及主題過濾器
主題字串和主題過濾器用於發佈和訂閱。 MQTT 用戶端中主題字串和過濾器的語法，與 IBM MQ 中的主題字串語法大致相同。

MQTT 用戶端提供的服務品質

MQTT 用戶端提供三種服務品質，可將發佈遞送至 IBM MQ 及 MQTT 用戶端:「最多一次」、「至少一次」及「正好一次」。 MQTT 用戶端將要求傳送至 IBM MQ 以建立訂閱時，會以「至少一次」服務品質傳送要求。

發佈的服務品質是 MqttMessage 的屬性。 它透過方法 MqttMessage.setQos 設定。

方法 MqttClient.subscribe 可以降低套用至在某個主題上傳送至用戶端之發佈的服務品質。 轉遞至訂閱者的發佈服務品質可能與發佈服務品質不同。 會使用兩者之中的較低值來轉遞發佈。

至多一次
QoS=0

訊息最多遞送一次，或者根本不遞送。 將不會確認它透過網路進行的遞送。
不會儲存訊息。 如果用戶端已斷線，或伺服器失敗，則會遺失該訊息。
QoS=0 是最快的傳輸模式。 有時將它稱為「隨發即忘」。
MQTT protocol 不需要伺服器將位於 QoS=0 的出版品轉遞至用戶端。 如果在伺服器接收發佈時，用戶端已中斷連線，則可能會捨棄發佈，視伺服器而定。 遙測 (MQXR) 服務不會捨棄以 QoS=0 傳送的訊息。 這些訊息會儲存為非持續訊息，並且僅在佇列管理程式停止時才會捨棄。

至少一次
QoS=1

QoS=1 是預設傳輸模式。
一律會遞送訊息至少一次。 如果傳送端未接收到確認通知，則會再次傳送訊息並設定 DUP 旗標，直到接收到確認通知為止。 因此，接收端可以多次傳送相同的訊息，且可能會多次處理。

必須在訊息本端儲存在傳送端和接收端，直到其得到處理為止。

接收端處理訊息之後，會從接收端中刪除訊息。 如果接收端是分配管理系統，則會將訊息發佈至其訂閱者。 如果接收端是用戶端，則會將訊息遞送至其訂閱者應用程式。 刪除訊息之後，接收端會傳送確認通知至傳送端。

在傳送端收到接收端的確認通知之後，會從傳送端中刪除訊息。

只一次

QoS=2

訊息一律只遞送一次。

必須在訊息本端儲存在傳送端和接收端，直到其得到處理為止。

QoS=2 是最安全，但是最慢的傳送模式。 從傳送端中刪除訊息之前，會在傳送端和接收端之間進行至少兩組傳輸。 在第一組傳輸之後，可在接收端上處理訊息。

在第一組傳輸中，傳送端會傳送訊息，並取得來自接收端的已儲存訊息確認通知。 如果傳送端未接收到確認通知，則會再次傳送訊息並設定 DUP 旗標，直到接收到確認通知為止。

在第二組傳輸中，傳送端通知接收端，它已完成處理訊息 ("PUBREL")。 如果傳送端未收到 "PUBREL" 訊息的確認通知，則會再次傳送 "PUBREL" 訊息，直到收到確認通知為止。 在收到 "PUBREL" 訊息的確認之後，傳送端會刪除它已儲存的訊息

接收端可以在第一個或第二個階段處理訊息，只要它不會重新處理訊息即可。 如果接收端是分配管理系統，則它會將訊息發佈至訂閱者。 如果接收端是用戶端，則它會將訊息遞送至訂閱者應用程式。 接收端會將完成訊息傳送回傳送端，確認它已完成處理訊息。

相關概念

MQTT 用戶端應用程式中的回呼及同步化

MQTT 用戶端程式設計模型廣泛使用執行緒。 執行緒會盡可能取消 MQTT 用戶端應用程式與伺服器之間傳輸訊息的延遲之間的連結。 發佈、遞送記號及連線遺失事件會遞送至實作 `MqttCallback` 的回呼類別中的方法。

清除階段作業

MQTT 用戶端及遙測 (MQXR) 服務會維護階段作業狀態資訊。 狀態資訊用來確保 "至少一次" 和 "正好一次" 遞送，以及 "正好一次" 接收發佈。 階段作業狀態也包括 MQTT 用戶端所建立的訂閱。 您可以選擇在階段作業之間執行具有或不具有維護狀態資訊的 MQTT 用戶端。 在連接之前設定 `MqttConnectOptions.cleanSession`，以變更清除階段作業模式。

用戶端 ID

用戶端 ID 是一個 23 個位元組的字串，用來識別 MQTT 用戶端。 對於一次僅一個已連接的用戶端來說，每一個 ID 都必須是唯一的。 ID 只能包含在佇列管理程式名稱中有效的字元。 在這些限制的範圍內，您可以使用任何識別字串。 具有配置用戶端 ID 的程序，以及透過用戶端的選定 ID 來配置用戶端的方法，是非常重要的。

遞送記號

最後留言發佈

如果 MQTT 用戶端連線非預期地結束，您可以配置 MQ Telemetry 以傳送「最後留言」發佈。 預先定義發佈的內容，以及要將發佈傳送至的主題。 「最後留言」是連線內容。 它應在連接用戶端之前建立。

MQTT 用戶端中的訊息持續性

如果以 "至少一次"或 "正好一次"的服務品質來傳送發佈訊息，則會持續保存發佈訊息。 您可以在用戶端上實作您自己的持續性機制，或者使用用戶端提供的預設持續性機制。 持續性對於兩個方向起作用：傳送至用戶端的發佈或用戶端傳送的發佈。

發佈

出版品是與主題字串相關聯的 `MqttMessage` 實例。 MQTT 用戶端可以建立發佈資訊以傳送至 IBM MQ，並訂閱 IBM MQ 上的主題以接收發佈資訊。

保留的發佈及 MQTT 用戶端

主題可以有一個 (且只能有一個) 保留的發佈資訊。 如果您建立具有保留發佈資訊之主題的訂閱，則會立即將發佈資訊轉遞給您。

訂閱

建立訂閱，以使用主題過濾器註冊所需的發佈主題。 用戶端可以建立多個訂閱，或者建立包含使用萬用字元的主題過濾器的訂閱，以註冊所需的多個主題。 會將主題上符合過濾器的發佈傳送至用戶端。 用戶端中斷連線時，訂閱可以保持處於作用中狀態。 用戶端重新連接後，會將發佈傳送至用戶端。

MQTT 用戶端中的主題字串及主題過濾器
主題字串和主題過濾器用於發佈和訂閱。 MQTT 用戶端中主題字串和過濾器的語法，與 IBM MQ 中的主題字串語法大致相同。

保留的發佈及 MQTT 用戶端

主題可以有一個 (且只能有一個) 保留的發佈資訊。 如果您建立具有保留發佈資訊之主題的訂閱，則會立即將發佈資訊轉遞給您。

使用 `MqttMessage.setRetained` 方法來指定是否保留主題的發佈資訊。

當您建立或更新保留的發佈時，請使用 QoS 1 或 2 來傳送發佈。 如果您以 QoS 0 來傳送它，則 IBM MQ 會建立非持續性保留發佈。 如果佇列管理程式已停止，則不會保留發佈。

如果將非保留的發佈，發佈至具有保留發佈的主題，不會影響保留的發佈。 現行訂閱者會接收新發佈。 新訂閱者會先接收保留的發佈，然後接收任何新發佈。

您可以使用保留的發佈資訊來記錄測量的最新值。 主題的新訂閱者會立即收到測量的最新值。 如果自從訂閱者前次訂閱發佈主題以來，沒有執行新的測量，則在訂閱者再次訂閱時，訂閱者會再次收到主題上最新的保留發佈。

若要刪除保留的發佈資訊，您有兩個選項:

- 執行 **CLEAR TOPICSTR** MQSC 指令。
- 建立長度為零的保留發佈資訊。 如 MQTT 3.1.1 規格中所指定，如果將長度為零的保留訊息發佈至主題，則會清除該主題的任何保留訊息。

相關概念

MQTT 用戶端應用程式中的回呼及同步化
MQTT 用戶端程式設計模型廣泛使用執行緒。 執行緒會盡可能取消 MQTT 用戶端應用程式與伺服器之間傳輸訊息的延遲之間的連結。 發佈、遞送記號及連線遺失事件會遞送至實作 `MqttCallback` 的回呼類別中的方法。

清除階段作業
MQTT 用戶端及遙測 (MQXR) 服務會維護階段作業狀態資訊。 狀態資訊用來確保 "至少一次" 和 "正好一次" 遞送，以及 "正好一次" 接收發佈。 階段作業狀態也包括 MQTT 用戶端所建立的訂閱。 您可以選擇在階段作業之間執行具有或不具有維護狀態資訊的 MQTT 用戶端。 在連接之前設定 `MqttConnectOptions.cleanSession`，以變更清除階段作業模式。

用戶端 ID
用戶端 ID 是一個 23 個位元組的字串，用來識別 MQTT 用戶端。 對於一次僅一個已連接的用戶端來說，每一個 ID 都必須是唯一的。 ID 只能包含在佇列管理程式名稱中有效的字元。 在這些限制的範圍內，您可以使用任何識別字串。 具有配置用戶端 ID 的程序，以及透過用戶端的選定 ID 來配置用戶端的方法，是非常重要的。

遞送記號

最後留言發佈
如果 MQTT 用戶端連線非預期地結束，您可以配置 MQ Telemetry 以傳送「最後留言」發佈。 預先定義發佈的內容，以及要將發佈傳送至的主題。 「最後留言」是連線內容。 它應在連接用戶端之前建立。

MQTT 用戶端中的訊息持續性
如果以 "至少一次"或 "正好一次"的服務品質來傳送發佈訊息，則會持續保存發佈訊息。 您可以在用戶端上實作您自己的持續性機制，或者使用用戶端提供的預設持續性機制。 持續性對於兩個方向起作用：傳送至用戶端的發佈或用戶端傳送的發佈。

發佈
出版品是與主題字串相關聯的 `MqttMessage` 實例。 MQTT 用戶端可以建立發佈資訊以傳送至 IBM MQ，並訂閱 IBM MQ 上的主題以接收發佈資訊。

MQTT 用戶端提供的服務品質
MQTT 用戶端提供三種服務品質，可將發佈遞送至 IBM MQ 及 MQTT 用戶端:「最多一次」、「至少一次」及「正好一次」。 MQTT 用戶端將要求傳送至 IBM MQ 以建立訂閱時，會以「至少一次」服務品質傳送要求。

訂閱

建立訂閱，以使用主題過濾器註冊所需的發佈主題。 用戶端可以建立多個訂閱，或者建立包含使用萬用字元的主題過濾器的訂閱，以註冊所需的多個主題。 會將主題上符合過濾器的發佈傳送至用戶端。 用戶端中斷連線時，訂閱可以保持處於作用中狀態。 用戶端重新連接後，會將發佈傳送至用戶端。

MQTT 用戶端中的主題字串及主題過濾器
主題字串和主題過濾器用於發佈和訂閱。 MQTT 用戶端中主題字串和過濾器的語法，與 IBM MQ 中的主題字串語法大致相同。

訂閱

建立訂閱，以使用主題過濾器註冊所需的發佈主題。 用戶端可以建立多個訂閱，或者建立包含使用萬用字元的主題過濾器的訂閱，以註冊所需的多個主題。 會將主題上符合過濾器的發佈傳送至用戶端。 用戶端中斷連線時，訂閱可以保持處於作用中狀態。 用戶端重新連接後，會將發佈傳送至用戶端。

可以使用 `MqttClient.subscribe` 方法，並傳遞一個以上的主題過濾器和服務品質參數，來建立訂閱。 服務品質參數設定訂閱者準備用於接收訊息的最高服務品質。 不能使用更高的服務品質，來遞送傳送至此用戶端的訊息。 會將服務品質設定為發佈訊息時的原始值與為訂閱指定的層次中的較低值。 用於接收訊息的預設服務品質是 `QoS=1`（至少一次）。

會利用 `QoS=1` 傳送訂閱要求本身。

MQTT 用戶端呼叫 `MqttCallback.messageArrived` 方法時，訂閱者便可收到發佈。 `messageArrived` 方法還會將用於發佈訊息的主題字串傳送至訂閱者。

可以使用 `MqttClient.unsubscribe` 方法來移除一個訂閱或一組訂閱。

IBM MQ 指令可以移除訂閱。 使用 IBM MQ Explorer 或使用 **runmqsc** 或 PCF 指令來列出訂閱。 所有 MQTT 用戶端訂閱皆已命名。 他們會取得下列格式的名稱: *ClientIdentifier*:*Topic name*

在連接用戶端之前，如果您使用預設 `MqttConnectOptions`，或將 `MqttConnectOptions.cleanSession` 設為 `true` ，則在用戶端連接時，會移除用戶端的任何舊訂閱。 用戶端在階段作業期間建立的所有新訂閱，皆會在連線中斷後移除。

如果您在連接之前將 `MqttConnectOptions.cleanSession` 設為 `false` ，則用戶端建立的任何訂閱都會新增至用戶端連接之前存在的所有訂閱。 用戶端中斷連線時，所有訂閱都保持為作用中狀態。

另一種瞭解 `cleanSession` 屬性如何影響訂閱的方式，是將它視為限制模式的屬性。 在其預設模式 `cleanSession=true` 中，用戶端僅在階段作業的範圍內建立訂閱並接收發佈。 在替代模式 `cleanSession=false` 中，訂閱是可延續的。 用戶端可以連接和中斷連線，其訂閱會保持為作用中狀態。 用戶端重新連接時，它會接收所有未遞送的發佈。 它連接時，可以自行修改作用中的訂閱集。

您必須在連接之前設定 `cleanSession` 模式; 此模式會持續整個階段作業。 若要變更其設定，您必須中斷連線，然後重新連接用戶端。 如果您將模式從使用 `cleanSession=false` 變更為 `cleanSession=true`，則會捨棄用戶端的所有先前訂閱，以及尚未收到的任何發佈。

符合作用中訂閱的發佈一旦發佈，便會立即傳送至用戶端。 如果用戶端已中斷連線，則當用戶端利用相同的用戶端 ID 重新連接至相同的伺服器，而且 `MqttConnectOptions.cleanSession` 設為 `false` 時，會將這些發佈傳送至該用戶端。

特定用戶端的訂閱是透過用戶端 ID 識別。 您可以將用戶端從不同的用戶端裝置重新連接至相同的伺服器，並繼續相同的訂閱和接收未遞送的發佈。

相關概念

MQTT 用戶端應用程式中的回呼及同步化
MQTT 用戶端程式設計模型廣泛使用執行緒。 執行緒會盡可能取消 MQTT 用戶端應用程式與伺服器之間傳輸訊息的延遲之間的連結。 發佈、遞送記號及連線遺失事件會遞送至實作 `MqttCallback` 的回呼類別中的方法。

清除階段作業
MQTT 用戶端及遙測 (MQXR) 服務會維護階段作業狀態資訊。 狀態資訊用來確保 "至少一次" 和 "正好一次" 遞送，以及 "正好一次" 接收發佈。 階段作業狀態也包括 MQTT 用戶端所建立的訂閱。 您可以選擇在階段作業之間執行具有或不具有維護狀態資訊的 MQTT 用戶端。 在連接之前設定 `MqttConnectOptions.cleanSession` ，以變更清除階段作業模式。

用戶端 ID

用戶端 ID 是一個 23 個位元組的字串，用來識別 MQTT 用戶端。 對於一次僅一個已連接的用戶端來說，每一個 ID 都必須是唯一的。 ID 只能包含在佇列管理程式名稱中有效的字元。 在這些限制的範圍內，您可以使用任何識別字串。 具有配置用戶端 ID 的程序，以及透過用戶端的選定 ID 來配置用戶端的方法，是非常重要的。

遞送記號

最後留言發佈
如果 MQTT 用戶端連線非預期地結束，您可以配置 MQ Telemetry 以傳送「最後留言」發佈。 預先定義發佈的內容，以及要將發佈傳送至的主題。 「最後留言」是連線內容。 它應在連接用戶端之前建立。

MQTT 用戶端中的訊息持續性
如果以 "至少一次"或 "正好一次"的服務品質來傳送發佈訊息，則會持續保存發佈訊息。 您可以在用戶端上實作您自己的持續性機制，或者使用用戶端提供的預設持續性機制。 持續性對於兩個方向起作用：傳送至用戶端的發佈或用戶端傳送的發佈。

發佈
出版品是與主題字串相關聯的 `MqttMessage` 實例。 MQTT 用戶端可以建立發佈資訊以傳送至 IBM MQ，並訂閱 IBM MQ 上的主題以接收發佈資訊。

MQTT 用戶端提供的服務品質
MQTT 用戶端提供三種服務品質，可將發佈遞送至 IBM MQ 及 MQTT 用戶端:「最多一次」、「至少一次」及「正好一次」。 MQTT 用戶端將要求傳送至 IBM MQ 以建立訂閱時，會以「至少一次」服務品質傳送要求。

保留的發佈及 MQTT 用戶端
主題可以有一個 (且只能有一個) 保留的發佈資訊。 如果您建立具有保留發佈資訊之主題的訂閱，則會立即將發佈資訊轉遞給您。

MQTT 用戶端中的主題字串及主題過濾器
主題字串和主題過濾器用於發佈和訂閱。 MQTT 用戶端中主題字串和過濾器的語法，與 IBM MQ 中的主題字串語法大致相同。

MQTT 用戶端中的主題字串及主題過濾器

主題字串和主題過濾器用於發佈和訂閱。 MQTT 用戶端中主題字串和過濾器的語法，與 IBM MQ 中的主題字串語法大致相同。

主題字串用於將發佈傳送至訂閱者。 使用方法 `MqttClient.getTopic(java.lang.String topicString)`來建立主題字串。

主題過濾器用於訂閱主題以及接收發佈。 主題過濾器可以包含萬用字元。 利用萬用字元，您可以訂閱多個主題。 使用訂閱方法來建立主題過濾器; 例如， `MqttClient.subscribe(java.lang.String topicFilter)`。

主題字串

主題字串中說明 IBM MQ 主題字串的語法。 MQTT client for Java 的 API 文件中的 `MqttClient` 類別說明 MQTT 主題字串的語法。 如需 MQTT 用戶端程式庫的用戶端 API 文件鏈結，請參閱 MQTT 用戶端程式設計參考手冊。

每種主題字串類型的語法幾乎相同。 只有四個微小的差異：

1. MQTT 用戶端傳送至 IBM MQ 的主題字串必須遵循佇列管理程式名稱的慣例。
2. 長度上限不同。 IBM MQ 主題字串限制為 10,240 個字元。 MQTT 用戶端可以建立最多 65535 個位元組的主題字串。
3. MQTT 用戶端建立的主題字串不能包含空值字元。
4. 在 IBM Integration Bus 中，空值主題層次 `'...//...'` 無效。 IBM MQ 支援空值主題層次。

與 IBM MQ 發佈/訂閱不同，`mqttv3` 通訊協定沒有管理主題物件的概念。 您無法從主題物件和主題字串建構主題字串。 不過，主題字串會對映至 IBM MQ 中的管理主題。 與管理主題相關聯的存取控制，決定發佈是發佈至主題還是捨棄。 將發佈轉遞至訂閱者時套用至發佈的屬性，受管理主題的屬性影響。

主題過濾器

主題型萬用字元架構中說明 IBM MQ 主題過濾器的語法。 您可以使用 MQTT 用戶端建構之主題過濾器的語法，在 MQTT client for Java 的 API 文件中的 `MqttClient` 類別中有說明。 如需 MQTT 用戶端程式庫的用戶端 API 文件鏈結，請參閱 MQTT 用戶端程式設計參考手冊。

相關概念

MQTT 用戶端應用程式中的回呼及同步化
MQTT 用戶端程式設計模型廣泛使用執行緒。 執行緒會盡可能取消 MQTT 用戶端應用程式與伺服器之間傳輸訊息的延遲之間的連結。 發佈、遞送記號及連線遺失事件會遞送至實作 `MqttCallback` 的回呼類別中的方法。

清除階段作業
MQTT 用戶端及遙測 (MQXR) 服務會維護階段作業狀態資訊。 狀態資訊用來確保 "至少一次" 和 "正好一次" 遞送，以及 "正好一次" 接收發佈。 階段作業狀態也包括 MQTT 用戶端所建立的訂閱。 您可以選擇在階段作業之間執行具有或不具有維護狀態資訊的 MQTT 用戶端。 在連接之前設定 `MqttConnectOptions.cleanSession`，以變更清除階段作業模式。

用戶端 ID
用戶端 ID 是一個 23 個位元組的字串，用來識別 MQTT 用戶端。 對於一次僅一個已連接的用戶端來說，每一個 ID 都必須是唯一的。 ID 只能包含在佇列管理程式名稱中有效的字元。 在這些限制的範圍內，您可以使用任何識別字串。 具有配置用戶端 ID 的程序，以及透過用戶端的選定 ID 來配置用戶端的方法，是非常重要的。

遞送記號

最後留言發佈
如果 MQTT 用戶端連線非預期地結束，您可以配置 MQ Telemetry 以傳送「最後留言」發佈。 預先定義發佈的內容，以及要將發佈傳送至的主題。 「最後留言」是連線內容。 它應在連接用戶端之前建立。

MQTT 用戶端中的訊息持續性
如果以 "至少一次"或 "正好一次"的服務品質來傳送發佈訊息，則會持續保存發佈訊息。 您可以在用戶端上實作您自己的持續性機制，或者使用用戶端提供的預設持續性機制。 持續性對於兩個方向起作用：傳送至用戶端的發佈或用戶端傳送的發佈。

發佈
出版品是與主題字串相關聯的 `MqttMessage` 實例。 MQTT 用戶端可以建立發佈資訊以傳送至 IBM MQ，並訂閱 IBM MQ 上的主題以接收發佈資訊。

MQTT 用戶端提供的服務品質
MQTT 用戶端提供三種服務品質，可將發佈遞送至 IBM MQ 及 MQTT 用戶端:「最多一次」、「至少一次」及「正好一次」。 MQTT 用戶端將要求傳送至 IBM MQ 以建立訂閱時，會以「至少一次」服務品質傳送要求。

保留的發佈及 MQTT 用戶端
主題可以有一個 (且只能有一個) 保留的發佈資訊。 如果您建立具有保留發佈資訊之主題的訂閱，則會立即將發佈資訊轉遞給您。

訂閱
建立訂閱，以使用主題過濾器註冊所需的發佈主題。 用戶端可以建立多個訂閱，或者建立包含使用萬用字元的主題過濾器的訂閱，以註冊所需的多個主題。 會將主題上符合過濾器的發佈傳送至用戶端。 用戶端中斷連線時，訂閱可以保持處於作用中狀態。 用戶端重新連接後，會將發佈傳送至用戶端。

使用 IBM MQ 開發 Microsoft Windows Communication Foundation 應用程式

IBM MQ 的 Microsoft Windows Communication Foundation (WCF) 自訂通道會在 WCF 用戶端與服務之間傳送及接收訊息。

相關概念

第 1064 頁的『IBM MQ custom channel for WCF with .NET 簡介』
IBM MQ 的自訂通道是使用 Microsoft Windows Communication Foundation (WCF) 統一程式設計模型的傳輸通道。

第 1068 頁的『使用 WCF 的 IBM MQ 自訂通道』
使用 Windows Communication Foundation (WCF) 的 IBM MQ 自訂通道可供程式設計師使用的資訊概觀。

第 1083 頁的『使用 WCF 範例』
Windows Communication Foundation (WCF) 範例提供一些簡單範例，說明如何使用 IBM MQ 自訂通道。

FFST: WCF XMS 首次失敗支援技術

相關工作

追蹤 IBM MQ 的 WCF 自訂通道

IBM MQ 問題的 WCF 自訂通道疑難排解

IBM MQ custom channel for WCF with .NET 簡介

IBM MQ 的自訂通道是使用 Microsoft Windows Communication Foundation (WCF) 統一程式設計模型的傳輸通道。

Microsoft.NET 3 中引進的 Microsoft Windows Communication Foundation 架構可讓您獨立開發 .NET 應用程式及服務，與用來連接它們的傳輸及通訊協定無關，並可讓您根據服務或應用程式部署所在的環境來使用替代傳輸或配置。

在執行時期由 WCF 透過建置包含下列必要組合的通道堆疊來管理連線:

- 通訊協定元素: 選用元素集，其中可以新增 none、one 或 more 來支援 WS-* 標準之類的通訊協定。
- 訊息編碼器: 堆疊中的必要元素，用於控制將訊息序列化為其發訊格式。
- 傳輸通道: 堆疊中的必要元素，負責將序列化訊息傳輸至其端點。

IBM MQ 的自訂通道是傳輸通道，因此必須使用 WCF 自訂連結與應用程式所需的訊息編碼器及選用通訊協定配對。 以此方式，已開發使用 WCF 的應用程式可以使用 IBM MQ 的自訂通道來傳送及接收資料，其方式與使用 Microsoft 所提供的內建傳輸相同，可與 IBM MQ 的非同步、可調式及可靠傳訊功能進行簡單整合。 如需受支援函數的完整清單，請參閱: 第 1068 頁的『WCF 自訂通道特性及功能』。

何時及為何要對 WCF 使用 IBM MQ 自訂通道?

您可以使用 IBM MQ 自訂通道在 WCF 用戶端與服務之間傳送及接收訊息，其方式與 Microsoft 所提供的內建傳輸相同，可讓應用程式存取 WCF 統一程式設計模型內 IBM MQ 的特性。

WCF 的 IBM MQ 自訂通道的一般使用型樣實務範例是作為非 SOAP 介面來傳輸原生 IBM MQ 訊息。

使用非 SOAP/Non-JMS 訊息 (純 MQMessage) 格式傳送的訊息

當您使用 WCF 的 IBM MQ 自訂通道作為非 SOAP 介面來傳輸原生 IBM MQ 訊息時，會使用 IBM MQ 的非 SOAP/Non-JMS 訊息 (純 MQMessage) 格式來攜帶訊息。

WCF 使用者能夠啟動服務，換句話說，服務使用者能夠使用 MQMessages 將訊息傳送至 IBM MQ 佇列。 應用程式可以取得並設定 MQMD 欄位及有效負載。 當訊息在 IBM MQ 佇列中可用時，任何 WCF 服務或非 WCF 應用程式 (例如在 AIX、 Linux、 Windows 或 z/OS 上執行的 C 或 Java 應用程式) 都可以處理此訊息。

WCF 的 IBM MQ 自訂通道的軟體需求

本主題概述 WCF 的 IBM MQ 自訂通道的軟體需求。 WCF 的 IBM MQ 自訂通道只能連接至 IBM WebSphere MQ 7.0 或更高版本的佇列管理程式。

執行時期環境需求

- Microsoft.NET Framework v4.7.2 或更高版本必須安裝在主機上。
- 依預設， *Java* 和 *.NET* 傳訊及 *Web* 服務 會隨 IBM MQ 安裝程式一起安裝。 此元件會將自訂通道所需的 .NET 組件安裝至「廣域組件快取」。

註: 如果在安裝 IBM MQ 之前未安裝 Microsoft .NET Framework V4.7.2 或更高版本，則 IBM MQ 產品安裝會繼續進行，且不會發生錯誤，但 IBM MQ classes for .NET 無法使用。 如果在安裝 IBM MQ 之後安裝.NET Framework ，則必須透過執行 *WMQInstallDir*\bin\amqiRegisterdotNet.cmd Script 來登錄 IBM MQ.NET 組件，其中 *WMQInstallDir* 是 IBM MQ 的安裝目錄。 此 Script 會在「廣域組件快取 (GAC)」中安裝

必要的組件。 會在 %TEMP% 目錄中建立一組記錄所採取動作的 amqi*.log 檔案。 如果 .NET 從舊版 (例如，從 .NET V3.5) 升級至 V4.7.2 或更高版本，則不需要重新執行 amqiRegisterdotNet.cmd Script。

開發環境需求

- Microsoft Visual Studio 2015 或 Windows Software Development Kit for .NET 4.7.2 或更新版本。
- Microsoft.NET Framework V4.7.2 或更高版本必須安裝在主機上，才能建置範例解決方案檔案。

WCF 的 IBM MQ 自訂通道: 已安裝哪些?

IBM MQ 的自訂通道是使用 Microsoft Windows Communication Foundation (WCF) 統一程式設計模型的傳輸通道。 依預設，在安裝期間會安裝自訂通道。

WCF 的 IBM MQ 自訂通道

自訂通道及其相依關係包含在依預設安裝的 Java and .NET Messaging and Web Services 元件內。 從早於 IBM MQ 8.0 的版本升級 IBM MQ 時，如果先前已在較早的安裝中安裝 Java and .NET Messaging and Web Services 元件，則依預設，更新會安裝適用於 WCF 的 IBM MQ 自訂通道。

.NET Messaging and Web Services 元件包含 IBM.XMS.WCF.dll 檔案及 IBM.WMQ.WCF.dll 檔案，這些檔案是主要自訂通道組合，其中包含 WCF 介面類別。 這些檔案安裝在「廣域組件快取 (GAC)」中，也可以在下列目錄中找到: *MQ_INSTALLATION_PATH* \bin，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安裝目錄。

下表彙總使用自訂通道所需的索引鍵類別。

表 *189:* 使用自訂通道所需的索引鍵類別

	SOAP/JMS 介面 (現有)	**非 SOAP/Non-JMS 介面 (來自 IBM MQ 8.0)**
自訂通道組件	IBM.XMS.WCF.dll	IBM.WMQ.WCF.dll
傳輸連結名稱	IBM.XMS.WCF.SoapJmsIbmTransportBindingElement	IBM.WMQ.WCF.WmqIbmTransportBindingElement
傳輸連結匯入器	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementImporter	IBM.WMQ.WCF.WmqIbmTransportBindingElementImporter
傳輸連結配置	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementConfig	IBM.WMQ.WCF.WmqIbmTransportBindingElementConfig
範例 (單向)	SimpleOneWay_Client , SimpleOneWay_Service	MQMessaging_OneWay_Client、MQMessaging_OneWay_Service
範例 (RequestReply)	SimpleRequestReply_Client、SimpleRequestReply_Service	MQMessaging_RequestReply_Client、MQMessaging_RequestReply_Service

IBM.WMQ.WCF.dll 同時支援 SOAP/JMS 及非 SOAP/Non-JMS 介面。 建議使用 IBM.WMQ.WCF 組件，因為它支援兩個介面。

傳送 MQSTR 格式化訊息

如果要求訊息是 MQSTR 類型，您可以選取以 MQSTR 格式傳送回覆訊息。

您必須使用其他 URI 參數 **replyMessageFormat** 來變更回覆訊息的格式。 支援的值如下:

""

"" 是預設值。

回覆訊息採用位元組 (MQMFT_NONE) 格式。 例如:

```
"jms:/queue?
```

```
destination=SampleQ@QM1&connectionFactory=binding(server)connectQueueManager(QM1)
&initialContextFactory=com.ibm.mq.jms.Nojndi&replyDestination=SampleReplyQ&replyMessageForma
t= "
```

MQSTR
回覆訊息採用 MQSTR (MQMFT_STRING) 格式。 例如：

```
"jms:/queue?
destination=SampleQ@QM1&connectionFactory=binding(server)connectQueueManager(QM1)
&initialContextFactory=com.ibm.mq.jms.Nojndi&replyDestination=SampleReplyQ&replyMessageForma
t=MQSTR"
```

附註：

1. **replyMessageFormat** 的值不區分大小寫。
2. 使用 "" 或 *MQSTR* 以外的任何值會導致無效參數值異常狀況。

IBM MQ 自訂通道範例

這些範例提供一些簡單範例，說明如何使用 WCF 的 IBM MQ 自訂通道。 範例及其關聯檔案位於 *MQ_INSTALLATION_PATH* \tools\dotnet\samples\cs\wcf 目錄中，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安裝目錄。 如需 IBM MQ 自訂通道範例的相關資訊，請參閱 第 1083 頁的『使用 WCF 範例』。

svcutil.exe.config

svcutil.exe.config 是啟用 Microsoft WCF svcutil 用戶端 Proxy 產生工具來辨識自訂通道所需的配置設定範例。 svcutil.exe.config 檔案位於 *MQ_INSTALLATION_PATH* \tools\wcf\docs\examples\ 目錄中，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安裝目錄。 如需使用 svcutil.exe.config 的相關資訊，請參閱 第 1081 頁的『使用 svcutil 工具搭配執行中服務的 meta 資料來產生 WCF 用戶端 Proxy 及應用程式配置檔』。

WCF 架構

WCF 的 IBM MQ 自訂通道整合在 IBM Message Service Client for .NET (XMS .NET) API 之上。

SOAP/JMS 介面

WCF 架構如下圖所示：

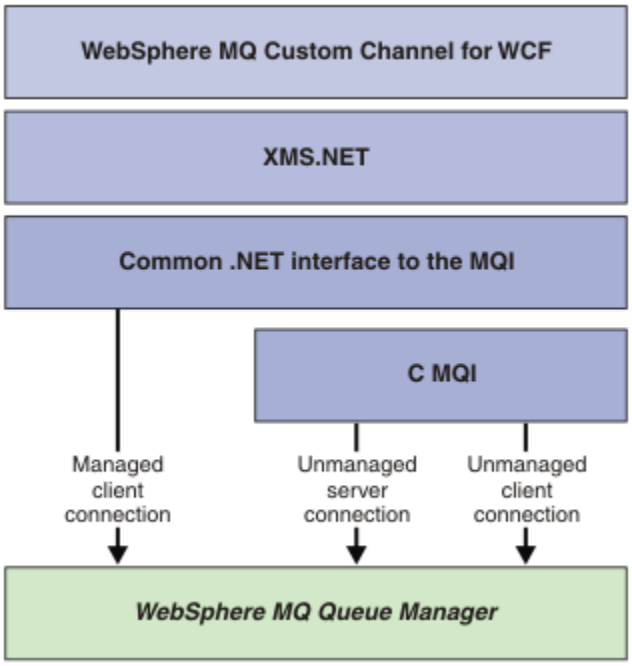


圖 *149: SOAP/JMS* 介面的 *WCF* 架構

依預設，所有必要元件都會隨產品安裝一起安裝。

這三個連線為:

- 受管理用戶端連線
- 未受管理的伺服器連線
- 未受管理的用戶端連線

如需這些連線的相關資訊，請參閱 第 1073 頁的『WCF 連線選項』。

非 SOAP/Non-JMS 介面

WCF 的 IBM MQ 自訂通道同時支援 SOAP/JMS 介面 (可從 IBM WebSphere MQ 7.0.1 取得) 及非 SOAP/Non-JMS 介面。

WCF 架構如下圖所示:

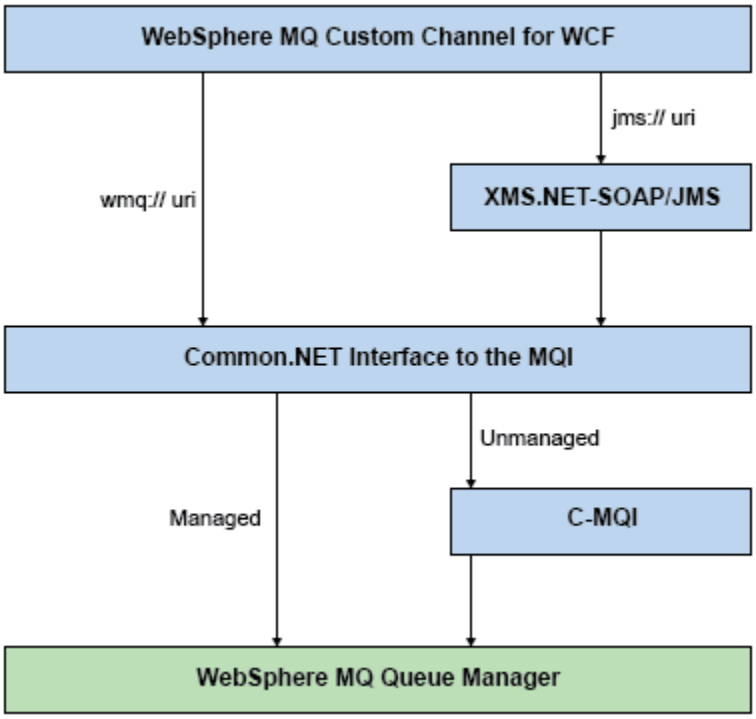


圖 *150:* 非 *SOAP/Non-JMS* 介面的 *WCF* 架構

使用 WCF 的 IBM MQ 自訂通道

使用 Windows Communication Foundation (WCF) 的 IBM MQ 自訂通道可供程式設計師使用的資訊概觀。

Microsoft Windows Communication Foundation 在 Microsoft.NET Framework 3 中建立 Web 服務及傳訊支援。 IBM MQ 可以用作 .NET Framework 3 中 WCF 的自訂通道，其方式與 Microsoft 提供的內建通道相同。

透過自訂通道傳輸的訊息會根據 IBM MQ 的 SOAP over JMS 實作來格式化。 然後，應用程式可以與 WCF 或 WebSphere SOAP over JMS 服務基礎架構所管理的服務進行通訊。

WCF 自訂通道特性及功能

如需 WCF 自訂通道特性及功能的相關資訊，請使用下列主題。

WCF 自訂通道形狀

IBM MQ 可以在 Microsoft Windows Communication Foundation (WCF) 自訂通道內使用的自訂通道形狀概觀。

WCF 的 IBM MQ 自訂通道支援兩種通道形狀:

- 單向
- 要求 - 回覆

WCF 會根據所管理的服務合約自動選取通道形狀。

包含僅使用 **IsOneWay** 參數之方法的合約由單向通道形狀處理，例如:

```
[OperationContract(IsOneWay = true)]
void printString(String text);
```

包含單向及 request-reply 方法或所有 request-reply 方法的合約，會由 request-reply 通道形狀來處理。 例如:

```
[OperationContract]
int subtract(int a, int b);

[OperationContract(IsOneWay = true)]
void printString(string text);
```

註: 在相同合約中混合單向及要求/回覆方法時，您必須確保行為符合預期，尤其是在混合環境中工作時，因為單向方法會等到它們收到來自服務的空值回覆。

單向通道

例如，使用 WCF 的 IBM MQ 單向自訂通道，以使用單向通道形狀從 WCF 用戶端傳送訊息。 通道只能單向傳送訊息，例如，從用戶端佇列管理程式傳送至 WCF 服務上的佇列。

要求/回覆通道

例如，使用 WCF 的 IBM MQ 要求/回覆自訂通道來非同步地以兩個方向傳送訊息; 相同的用戶端實例必須用於非同步傳訊。 通道可以單向傳送訊息，例如，從用戶端佇列管理程式傳送至 WCF 服務上的佇列，然後從 WCF 傳送回覆訊息至用戶端佇列管理程式上的佇列。

WCF URI 參數名稱和值

SOAP/JMS 介面及非 SOAP/Non JMS 介面的 URI 參數名稱及值。

SOAP/JMS 介面

connectionFactory
需要 connectionFactory 參數。

InitialContextFactory
initialContextFactory 參數是必要的，且必須設為 "com.ibm.mq.jms.Nojndi" ， 以與 WebSphere Application Server 及其他產品相容。

非 SOAP/Non JMS 介面

URI 格式適用於 MA93 規格。 如需 IBM MQ IRI 規格的進一步詳細資料，請參閱 SupportPac - MA93 。

IBM MQ URI 語法

```
wmq-iri = "wmq:" [ "//" connection-name ] "/" wmq-dest ["?" parm *("&" parm)]
connection-name = tcp-connection-name / other-connection-name
tcp-connection-name = ihost [ ":" port ]
other-connection-name = 1*(iunreserved / pct-encoded)
wmq-dest = queue-dest / topic-dest
queue-dest = "msg/queue/" wmq-queue ["@" wmq-qmgr]
wmq-queue = wmq-name
wmq-qmgr = wmq-name
wmq-name = 1*48( wmq-char )
topic-dest = "msg/topic/" wmq-topic
wmq-topic = segment *( "/" segment )
```

IBM MQ IRI 範例
下列範例 IRI 告訴服務要求端，它可以使用 IBM MQ TCP 用戶端連結連線至埠 1414 上稱為 example.com 的機器，並將持續要求訊息放置到佇列管理程式 QM1 上稱為 SampleQ 的佇列。 IRI 指定服務提供者將回覆放入稱為 SampleReplyQ 的佇列。

```
1)wmq://example.com:1414/msg/queue/SampleQ@QM1?
ReplyTo=SampleReplyQ&persistence=MQPER_NOT_PERSISTENT
2)wmq://localhost:1414/msg/queue/Q1?
connectQueueManager=QM1&replyTo=Q2&connectionmode=managed
```

對於已啟用 TLS 的連線
若要使用「WCF 用戶端/服務」建立「安全 (TLS)」連線，請在 URI 中使用適當的值設定下列內容。 字首為 "*" 的所有內容都是建立安全連線所必要的。

- **sslKeyRepository**: *SYSTEM 或 *USER
- * **sslCipherSpec**: 有效的 CipherSpec，例如 TLS_RSA_WITH_AES_128_CBC_SHA256。
- **sslCertRevocationCheck**:true 或 false。
- **sslKeyResetCount**: 大於 32kb 的值。
- **sslPeerName:** 伺服器憑證的識別名稱

例如:

```
"wmq://localhost:1414/msg/queue/SampleQ?
connectQueueManager=QM1&sslkeyrepository=*SYSTEM&sslcipherspec=
TLS_RSA_WITH_AES_128_CBC_SHA&sslcertrevocationcheck=true&"sslpe
ername=" + "" + "CN=ibmwebspheremqqmm&sslkeyresetcount=45000"
```

WCF 自訂通道確保遞送

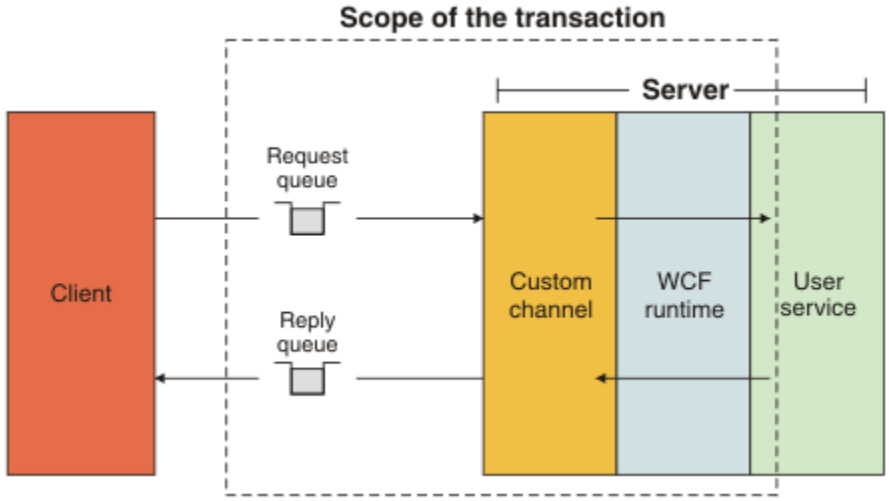
「保證遞送」保證服務要求或回覆已採取行動，不會遺失。

會接收要求訊息，並在區域交易同步點下傳送任何回覆訊息，可在執行時期失敗時回復。 這些失敗的範例包括: 服務擲出無法處理的異常狀況、無法將訊息分派給服務，或無法遞送回覆訊息。

AssuredDelivery 是可指定在服務合約上的保證遞送屬性，以保證在執行時期失敗時不會遺失服務所收到的任何要求訊息，以及從服務傳送的任何回覆訊息。

為了確保在系統故障或電源中斷時也會保留訊息，必須以持續傳送訊息。 若要使用持續訊息，用戶端應用程式必須在其端點 URI 上指定此選項。

不支援分散式交易，且交易的範圍不會延伸到 IBM MQ 所執行的要求及回覆訊息處理之外。 在服務內執行的任何工作可能會因為失敗而重新執行，這會導致重新接收訊息。 下圖顯示交易的範圍:



透過將 AssuredDelivery 屬性套用至服務類別來啟用保證遞送，如下列範例所示:

```
[AssuredDelivery]
class TestCalculatorService : IWMQSampleCalculatorContract
{
    public int add(int a, int b)
    {
        int ans = a + b;
        return ans;
    }
}
```

使用 AssuredDelivery 屬性時，您必須注意下列要點:

- 當通道判定如果回復並重新接收訊息，可能會再次發生失敗時，該訊息會被視為有毒訊息，且不會傳回要求佇列以重新處理。 例如: 如果收到的訊息格式不正確，或無法分派給服務。 一律會重新傳送從服務作業

擲出的未處理異常狀況，直到重新遞送訊息達到要求佇列的取消臨界值內容所指定的次數上限為止。 如需相關資訊，請參閱: 第 1071 頁的『WCF 自訂通道有害訊息』

- 通道會使用單一執行執行緒來執行讀取、處理及回覆每一個要求訊息作為基本作業，以施行交易式完整性。 為了讓服務作業能夠同時執行，通道可讓 WCF 建立通道的多個實例。 可用於處理要求的通道實例數由連結內容 `MaxConcurrentCalls` 控制。 如需相關資訊，請參閱: 第 1078 頁的『WCF 連結配置選項』
- 保證交付功能同時使用 IOperationInvoker 和 IErrorHandler WCF 延伸點。 如果應用程式在外部使用這些延伸點，應用程式必須確定會呼叫任何先前登錄的延伸點。 若未針對 IErrorHandler 執行此動作，可能會導致錯誤未報告。 若無法對 IOperationInvoker 執行此動作，可能會導致 WCF 停止回應。

WCF 自訂通道安全

WCF 的 IBM MQ 自訂通道僅支援對佇列管理程式的未受管理用戶端連線使用 TLS。

使用用戶端通道定義表 (CCDT) 中的項目來指定 TLS。 如需 CCDT 的相關資訊，請參閱 用戶端通道定義表。

WCF 用戶端通道定義表 *(CCDT)*

WCF 的 IBM MQ 自訂通道支援使用用戶端通道定義表 (CCDT) 來配置用戶端連線的連線資訊。

CCDT 是透過下列兩個環境變數來控制:

- *MQCHLLIB* 指定表格所在的目錄。
- *MQCHLTAB* 指定表格的檔名。

如果已定義這些環境變數，則它們會優先於 URI 中指定的任何用戶端連線詳細資料。

如需用戶端通道定義表的相關資訊，請參閱: 用戶端通道定義表。

WCF 自訂通道有害訊息

當服務無法處理要求訊息，或無法將回覆訊息遞送至回覆佇列時，該訊息會被視為有毒訊息。

有害要求訊息

如果無法處理要求訊息，則會將它視為有害訊息。 此動作會防止服務再次接收相同的無法處理的訊息。 若要將無法處理的要求訊息視為有害訊息，下列其中一種狀況必須為 true:

- 訊息取消計數已超出要求佇列上指定的取消臨界值，只有在為服務指定保證遞送時才會發生此情況。 如需保證遞送的相關資訊，請參閱: 第 1070 頁的『WCF 自訂通道確保遞送』
- 訊息未正確格式化，無法解譯為 SOAP over JMS 訊息。

有害回覆訊息

如果服務無法將回覆訊息遞送至回覆佇列，則會將回覆訊息視為有害訊息。 對於回覆訊息，此動作可讓您稍後擷取回覆訊息，以協助判斷問題。

有害訊息處理

對有害訊息採取的動作視佇列管理程式配置及訊息報告選項中設定的值而定。 對於 SOAP over JMS，依預設會在要求訊息上設定下列報告選項，且無法配置:

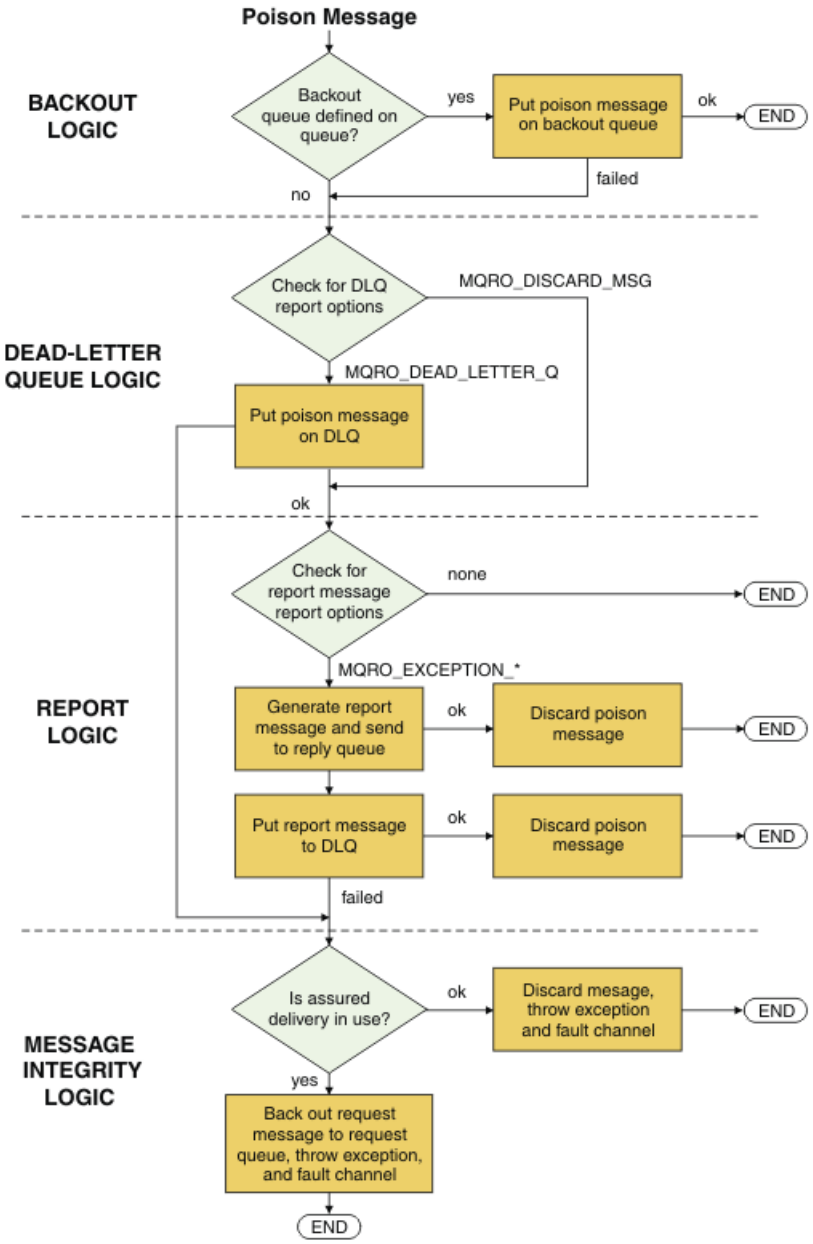
- MQRO_EXCEPTION_WITH_FULL_DATA
- MQRO_EXPIRATION_WITH_FULL_DATA
- MQRO_DISCARD_MSG

若為 SOAP over JMS，依預設會在回覆訊息上設定下列報告選項，且無法配置:

- MQRO_DEAD_LETTER_Q

如果訊息來自非 WCF 來源，請參閱該來源的文件。

下圖顯示有毒訊息處理失敗時可能採取的動作和步驟:



WCF 應用程式的 *IBM MQ* 訊息功能

非 SOAP/Non-JMS (即 IBM MQ) WCF 應用程式的訊息功能。

對於非 SOAP/Non-JMS 介面， WCF 應用程式的 IBM MQ 訊息功能如下所示:

- WCF 應用程式可以傳送及接收基本 IBM MQ 訊息，這些訊息可以由任何 IBM MQ 應用程式處理。
- WCF 應用程式具有完整控制項，可更新 MQMD 及有效負載。
- WCF 用戶端可以傳送任何 IBM MQ 用戶端 (例如 C、 Java、 JMS 及 .NET 用戶端) 可以使用的 IBM MQ 訊息。

WCF for Non-SOAP/Non-JMS 介面必須使用下列類別來設定訊息的訊息有效負載及 MQMD:

- WmqString 類型為「字串」之有效負載的訊息
- WmqBytes 類型位元組的有效負載訊息
- WmqXmlXML 類型有效負載的訊息

若要設定訊息的有效負載，請視有效負載類型而定，使用 WmqString 訊息、 WmqBytes 訊息或 WmqXml 訊息類別的 **Data** 內容。 例如，使用下列程式碼來設定「字串」類型的有效負載:

```
WmqStringMessage strMsg = new WmqStringMessage();
//Setting the Message PayLoad
strMsg.Data = "Hello World";
//MQMD property
strMsg.Format = WmqMessageFormat.MQFMT_STRING;
```

WCF 連線選項

將 WCF 的 IBM MQ 自訂通道連接至佇列管理程式有三種模式。 請考量最適合您需求的連線類型。

如需連線選項的相關資訊，請參閱: 第 484 頁的『連線差異』

如需 WCF 架構的相關資訊，請參閱: 第 1066 頁的『WCF 架構』

未受管理的用戶端連線

在此模式下建立的連線會以 IBM MQ 用戶端身分連接至在本端機器或遠端機器上執行的 IBM MQ 伺服器。

若要使用 WCF 的 IBM MQ 自訂通道作為 IBM MQ 用戶端，您可以使用 IBM MQ MQI client 在 IBM MQ 伺服器上或在個別機器上安裝它。

未受管理的伺服器連線

在伺服器連結模式中使用時， WCF 的 IBM MQ 自訂通道會使用佇列管理程式 API ，而不是透過網路進行通訊。 使用連結連線會為 IBM MQ 應用程式提供比使用網路連線更好的效能。

如果要使用連結連線，您必須在 IBM MQ 伺服器上安裝 WCF 的 IBM MQ 自訂通道。

受管理用戶端連線

在此模式下建立的連線會以 IBM MQ 用戶端身分連接至在本端機器或遠端機器上執行的 IBM MQ 伺服器。

在此模式下連接的 .NET 3 的 IBM MQ 自訂通道類別會保留在 .NET 受管理程式碼中，且不會呼叫原生服務。 如需受管理程式碼的相關資訊，請參閱 Microsoft 文件。

使用受管理用戶端有一些限制。 如需這些限制的相關資訊，請參閱 第 484 頁的『受管理用戶端連線』。

建立及配置 WCF 的 IBM MQ 自訂通道

WCF 工作的 IBM MQ 自訂通道，與 Microsoft 提供的傳輸 WCF 通道相同。 WCF 的 IBM MQ 自訂通道可以用兩種方式之一來建立。

關於這項作業

IBM MQ 自訂通道與 WCF 整合作為 WCF 傳輸通道，因此必須與訊息編碼器及選用通訊協定通道配對，以便它可以建立可供應用程式使用的完整通道堆疊。 若要順利建立完整通道堆疊，需要兩個元素:

1. 連結定義: 指定建置應用程式通道堆疊所需的元素，包括傳輸通道、訊息編碼器及任何通訊協定，以及任何一般配置設定。 對於自訂通道，必須以 WCF 自訂連結形式建立連結定義。
2. 端點定義: 鏈結服務合約與連結定義，並提供實際連線 URI 來說明應用程式可以連接的位置。 對於自訂通道， URI 採用 SOAP over JMS URI 形式。

這些定義可以用兩種不同方式之一來建立:

- 在管理上; 定義是透過在應用程式配置檔中提供詳細資料來建立 (例如: `app.config`)。
- 以程式化方式; 直接從應用程式碼建立定義。

要使用哪個方法來建立定義的決策必須根據應用程式的需求，如下所示:

- 配置的管理方法提供彈性來變更服務和用戶端後置部署的詳細資料，而不需重建應用程式。
- 配置的程式化方法提供更大的保護，可避免發生配置錯誤，並且能夠在執行時期動態產生配置。

在應用程式配置檔中提供連結和端點資訊，以管理方式建立 *WCF* 自訂通道

WCF 的 IBM MQ 自訂通道是傳輸層次 WCF 通道。 必須定義端點和連結才能使用自訂通道，您可以在應用程式配置檔中提供連結和端點資訊來完成這些定義。

若要配置及使用 WCF 的 IBM MQ 自訂通道 (即傳輸層次 WCF 通道)，必須定義連結及端點定義。 連結會保留通道的配置資訊，端點定義會保留連線詳細資料。 有兩種方式可以建立這些定義:

- 以程式化方式直接從應用程式碼，如這裡所述: 第 1075 頁的『以程式化方式提供連結和端點資訊來建立 WCF 自訂通道』
- 以管理方式，在應用程式配置檔中提供詳細資料，如下列程序所述。

用戶端或服務應用程式配置檔通常命名為 *yourappname*`.exe.config`，其中 *yourappname* 是應用程式的名稱。 使用稱為 `SvcConfigEditor.exe` 的 Microsoft 服務配置編輯器工具，最容易修改應用程式配置檔，方法如下:

- 啟動 `SvcConfigEditor.exe` 配置編輯器工具。 該工具的預設安裝位置為: *Drive:*`\Program Files\Microsoft SDKs\Windows\v6.0\Bin\SvcConfigEditor.exe`，其中 *磁碟機:* 是安裝磁碟機的名稱。

步驟 1: 新增連結元素延伸，以啟用 WCF 來尋找自訂通道

1. 用滑鼠右鍵按一下 **進階** > **延伸** > **連結元素** 以開啟功能表，然後選取 **新建**
2. 完成此表格中顯示的欄位:

表 *190:* 新建連結元素欄位	
欄位	**值**
名稱	`IBM.XMS.WCF.SoapJmsIbmTransportChannel`
類型	導覽至「廣域組件快取 (GAC)」中的 `IBM.XMS.WCF.dll`，然後選取 `IBM.XMS.WCFSoapJmsIbmTransportBindingElementConfig`

步驟 2: 建立自訂連結定義，以將自訂通道與 WCF 訊息編碼器配對

1. 用滑鼠右鍵按一下 **連結** 以開啟功能表，然後選取 **新建連結配置**
2. 完成此表格中顯示的欄位:

表 *191:* 新建連結配置欄位	
欄位	**值**
名稱	`CustomBinding_WMQ`
BindingElement 1	`textMessageEncoding (MessageVersion: Soap11)`
BindingElement 2	`IBM.XMS.WCF.SoapJmsIbmTransportChannel`

步驟 3: 指定連結內容

1. 選取 *IBM.XMS.WCF.SoapJmsIbmTransportChannel* 傳輸連結來自您在 第 1074 頁的『步驟 2: 建立自訂連結定義，以將自訂通道與 WCF 訊息編碼器配對』 中建立的連結
2. 對內容的預設值進行任何必要的變更，如 第 1078 頁的『WCF 連結配置選項』 所述。

步驟 4: 建立端點定義

建立端點定義，以參照您在 第 1074 頁的『步驟 2: 建立自訂連結定義，以將自訂通道與 WCF 訊息編碼器配對』 中建立的自訂連結，並提供服務的連線詳細資料。 指定此資訊的方式取決於定義是用於用戶端應用程式還是服務應用程式。

若為用戶端應用程式，請將端點定義新增至用戶端區段，如下所示:

1. 用滑鼠右鍵按一下 **用戶端** > **端點** 以開啟功能表，然後選取 **新建用戶端端點**
2. 完成此表格中顯示的欄位:

表 *192:* 新建用戶端端點欄位	
欄位	**值**
名稱	`Endpoint_WMQ`
位址	*SOAP/JMS URI* ，說明存取服務所需的 *WMQ* 連線詳細資料。 如需進一步詳細資料，請參閱: 第 1077 頁的『WCF 端點 URI 位址格式的 IBM MQ 自訂通道』
連結中	`customBinding`
BindingConfiguration	`CustomBinding_WMQ`
合約	服務合約介面的名稱

若為服務應用程式，請將服務定義新增至服務區段，如下所示:

1. 用滑鼠右鍵按一下 **服務** 以開啟功能表，並選取 **新建服務**，然後選取要管理的服務類別。
2. 將端點定義新增至新服務的 **端點** 區段，並完成下表中所示的欄位:

表 *193:* 新建服務端點欄位	
欄位	**值**
名稱	`Endpoint_WMQ`
位址	*SOAP/JMS URI* ，說明存取服務所需的 *WMQ* 連線詳細資料。 如需進一步詳細資料，請參閱: 第 1077 頁的『WCF 端點 URI 位址格式的 IBM MQ 自訂通道』
連結中	`customBinding`
BindingConfiguration	`CustomBinding_WMQ`
合約	服務實作類別的名稱

以程式化方式提供連結和端點資訊來建立 *WCF* 自訂通道

WCF 的 IBM MQ 自訂通道是傳輸層次 WCF 通道。 必須定義端點和連結才能使用自訂通道，且這些定義可以直接從應用程式碼以程式化方式完成。

若要配置及使用 WCF 的 IBM MQ 自訂通道 (即傳輸層次 WCF 通道) ，必須定義連結及端點定義。 連結會保留通道的配置資訊，端點定義會保留連線詳細資料。 如需相關資訊，請參閱 第 1083 頁的『使用 WCF 範例』。

有兩種方式可以建立這些定義:

- 以管理方式，在應用程式配置檔中提供詳細資料，如 第 1074 頁的『在應用程式配置檔中提供連結和端點資訊，以管理方式建立 WCF 自訂通道』中所述。
- 以程式化方式直接從應用程式碼，如下列子主題中所述。

以程式化方式定義連結和端點資訊*: SOAP/JMS* 介面

對於 SOAP/JMS 介面，您可以定義端點，並以程式化方式直接從應用程式碼進行連結。

關於這項作業

如果要以程式化方式提供連結和端點資訊，請完成下列步驟，將必要的程式碼新增至應用程式。

程序

1. 將下列程式碼新增至應用程式，以建立通道的傳輸連結元素實例:

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
```

2. 設定任何必要的連結內容，例如，將下列程式碼新增至應用程式以設定 ClientConnectionMode:

```
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.AS_URI;
```

3. 將下列程式碼新增至應用程式，以建立自訂連結，將傳輸通道與訊息編碼器配對:

```
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
```

4. 建立 SOAP/JMS URI。
   必須提供說明存取服務所需之 IBM MQ 連線詳細資料的 SOAP/JMS URI 作為端點位址。 您指定的位址視通道是用於服務應用程式還是用戶端應用程式而定。
   - 對於用戶端應用程式， SOAP/JMS URI 必須建立成 EndpointAddress ，如下所示:

     ```
     EndpointAddress address = new EndpointAddress("jms:/queue?
     destination=SampleQ@QM1&connectionFactory
     =connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
     ```

   - 對於服務應用程式， SOAP/JMS URI 必須建立為 URI ，如下所示:

     ```
     Uri address = new Uri("jms:/queue?destination=SampleQ@QM1&connectionFactory=
     connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
     ```

     如需端點位址的相關資訊，請參閱 第 1077 頁的『WCF 端點 URI 位址格式的 IBM MQ 自訂通道』。

以程式化方式定義連結和端點資訊*:* 非 *SOAP/Non-JMS* 介面
對於非 SOAP/Non-JMS 介面，您可以直接從應用程式碼定義端點並以程式化方式進行連結。

關於這項作業

如果要以程式化方式提供連結和端點資訊，請完成下列步驟，將必要的程式碼新增至應用程式。

程序

1. 透過將下列程式碼新增至應用程式，建立 WmqBinding :

```
WmqBinding binding = new WmqBinding();
```

   此程式碼會建立連結，以配對 WmqMsgEncodingElement 與 WmqIbmTransportBinding 非 SOAP/Non-JMS 介面所需的元素。

2. 提供 wmq: // URI ， 以說明存取服務所需的 IBM MQ 連線詳細資料。
   您提供 wmq: // URI 的方式取決於通道是用於服務應用程式還是用戶端應用程式。
   - 對於用戶端應用程式， wmq: // URI 必須建立為 EndpointAddress ，如下所示:

     ```
     EndpointAddress address = new EndpointAddress
     ("wmq://localhost:1414/msg/queue/Q1?connectQueueManager=QM1&replyTo=Q2");
     ```

   - 對於服務應用程式， wmq: // URI 必須建立為 URI ，如下所示:

```
Uri sampleAddress = new Uri(
"wmq://localhost:1414/msg/queue/Q1?connectQueueManager=QM1&replyTo=Q2");
```

WCF 端點 *URI* 位址格式的 *IBM MQ* 自訂通道

使用提供位置和連線詳細資料的「通用資源 ID (URI)」來指定 Web 服務。 URI 格式視您是使用 SOAP/JMS 介面還是非 SOAP/Non-JMS 介面而定。

SOAP/JMS 介面

SOAP 的 IBM MQ 傳輸中支援的 URI 格式允許在存取目標服務時，對 SOAP/ IBM MQ 特定參數和選項進行綜合性的控制。 此格式與 WebSphere Application Server 及 CICS 相容，有助於 IBM MQ 與這兩個產品整合。

URI 語法如下:

```
jms:/queue? name=value&name=value...
```

where 名 is a parameter name and 價值 is an appropriate value, and the 名 = 價值 element can be repeated any number of times with the second and subsequent occurrences being preceded by an ampersand (&).

參數名稱會區分大小寫， IBM MQ 物件的名稱也會區分大小寫。 如果多次指定任何參數，則參數的最終出現會生效，這表示用戶端應用程式可以透過附加至 URI 來置換參數值。 如果包含任何其他無法辨識的參數，則會忽略它們。

If you store a URI in an XML string, you must represent the ampersand character as "&". 同樣地，如果在 Script 中撰寫 URI ，請小心跳出字元 (例如 **&**)，否則 shell 將會解譯這些字元。

這是 Axis 服務的簡式 URI 範例:

```
jms:/queue?destination=myQ&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

以下是 .NET 服務的簡式 URI 範例:

```
jms:/queue?destination=myQ&connectionFactory=()&targetService=MyService.asmx
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

只會提供必要的參數 (只有 .NET 服務才需要 `targetService`)，且 `connectionFactory` 沒有任何選項。

在此 Axis 範例中， `connectionFactory` 包含許多選項:

```
jms:/queue?destination=myQ@myRQM&connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

在此 Axis 範例中，也已指定 `connectionFactory` 的 `sslPeer` **名稱** 選項。 sslPeerName 本身的值包含名稱/值配對及有效的內嵌空白:

```
jms:/queue?destination=myQ@myRQM&connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
sslPeerName(CN=MQ Test 1,O=IBM,S=Hampshire,C=GB)
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

非 SOAP/Non-JMS 介面

存取目標服務時， NON-SOAP/Non-JMS 介面的 URI 格式允許對 IBM MQ 特定參數及選項進行綜合性程度的控制。

URI 語法如下:

```
wmq://example.com:1415/msg/queue/INS.QUOTE.REQUEST@MOTOR.INS ?ReplyTo=msg/queue/
INS.QUOTE.REPLY@BRANCH452&persistence=MQPER_NOT_PERSISTENT
```

這個 IRI 告訴服務要求程式，它可以在埠 1415 上使用 IBM MQ TCP 用戶端連結連線至稱為 example.com 的機器，並將持續要求訊息放入稱為 INS.QUOTE.REQUEST on queue manager MOTOR.INS。 IRI 指定服務提供者將回覆放入稱為 INS.QUOTE.REPLY 佇列管理程式 BRANCH452 的佇列。 URI 格式是指定給 SupportPac MA93。 如需 IBM MQ IRI 規格的詳細資料，請參閱 SupportPac MA93: IBM MQ -服務定義 。

WCF 連結配置選項

有兩種方法可將配置選項套用至自訂通道連結資訊。 您可以透過管理方式來設定內容，或以程式化方式來設定它們。

連結配置選項可以用下列兩種不同方式之一來設定:

1. 管理: 必須在應用程式配置檔中自訂連結定義的傳輸區段中指定連結內容設定，例如: `app.config`。
2. 以程式化方式: 在起始設定自訂連結期間，必須修改應用程式碼來指定內容。

以管理方式設定連結內容

連結內容設定可以在應用程式配置檔中指定，例如: `app.config`。 配置檔由 **svcutil** 產生，如下列範例所示。

SOAP/JMS 介面

```
<customBinding>
...
      <IBM.XMS.WCF.SoapJmsIbmTransportChannel maxBufferPoolSize="524288"
      maxMessageSize="4000000" clientConnectionMode="0" maxConcurrentCalls="16"/>
...
</customBinding>
```

非 SOAP/Non-JMS 介面

```
<customBinding>
      <IBM.WMQ.WCF.WmqMsgEncodingElement/>
      <IBM.WMQ.WCF.WmqIbmTransportChannel maxBufferPoolSize="524288"
      maxMessageSize="65536" clientConnectionMode="managedclient"/>
</customBinding>
```

以程式化方式設定連結內容

若要新增 WCF 連結內容以指定用戶端連線模式，您必須修改服務程式碼，以在起始設定自訂連結期間指定該內容。

使用下列範例來指定未受管理的用戶端連線模式:

```
SoapJmsIbmTransportBindingElement
transportBindingElement = new SoapJmsIbmTransportBindingElement();
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.CLIENT_UNMANAGED;

Binding sampleBinding = new CustomBinding(new TextMessageEncodingBindingElement(),
                                                    transportBindingElement);
```

WCF 連結內容

表 *194:* 以管理方式或程式化方式設定時，連結內容的值

內容名稱	用戶端或服務應用程式	管理值	程式化值	說明
maxBufferPoolSize	兩者	0 到 64 位元帶正負號的整數	0 到 64 位元帶正負號的整數	指定可用來儲存通道實例之 WCF 訊息緩衝區的記憶體大小上限。
maxMessage 大小	兩者	1 到 32 位元帶正負號的整數	1 到 32 位元帶正負號的整數	指定可用於個別 WCF 訊息的記憶體上限。
clientConnection 模式	兩者	0 (預設值) 1	AS_URI (預設值) CLIENT_UNMANAGED	指定傳輸通道的用戶端連線模式。 0 表示用戶端連線模式如 URI 中所指定。 僅在使用用戶端連線時使用。 指定用戶端連線模式與 URI 中指定的相同。 如果未設定用戶端連線模式，則 0 是預設值。 1 表示用戶端連線模式是未受管理的用戶端。 僅在使用用戶端連線時使用。
MaxConcurrent 呼叫數	用戶端	範圍是 0-2 147 483 647 16 是預設值	範圍是 0-2 147 483 647 16 是預設值	此內容定義在任何一次可以在個別用戶端 Proxy 上執行的並行作業數上限。 如果啟動更多作業，則會將它們排入佇列，直到進行中作業完成或逾時為止。 此設定可用來控制個別 Proxy 可以耗用的執行緒及資源數目上限。 0 會移除此限制，使所有作業能夠同時嘗試。
MaxConcurrent 呼叫數	服務	範圍為 1-2 147 483 647 16 是預設值	範圍為 1-2 147 483 647 16 是預設值	只有在已啟用保證遞送特性時，才會使用此內容 (如需保證遞送的相關資訊，請參閱 第 1070 頁的『WCF 自訂通道確保遞送』)。 它指定給定端點可同時進行的並行作業數上限。 變更此設定時需要小心。 每一個並行作業都需要其他資源，特別是來自執行緒儲存區的自訂通道及相關聯執行緒的新實例，以處理要求。 過度配置可能會產生反效果，並嚴重影響效能。 必須進行執行緒儲存區的適當配置，才能支援此內容。

建置及管理 WCF 的服務

說明如何建立及配置 WCF 服務的 Microsoft Windows Communication Foundation (WCF) 服務概觀。

WCF 的 IBM MQ 自訂通道及使用它的 WCF 服務可以透過下列方法進行管理:

- 自我管理
- Windows 服務

無法在「Windows 處理程序啟動服務」中管理 WCF 的 IBM MQ 自訂通道。

下列主題提供一些簡單的自我管理範例，以示範所涉及的步驟。 Microsoft WCF 線上文件 (包含進一步資訊及最新詳細資料) 可以在 Microsoft MSDN 網站上找到，網址為 https://msdn.microsoft.com。

使用方法 *1* 建置 *WCF* 服務應用程式*:* 使用應用程式配置檔以管理方式自行管理

建立應用程式配置檔之後，開啟服務的實例，並將指定的程式碼新增至應用程式。

開始之前

建立或編輯服務的應用程式配置檔，如 第 1074 頁的『在應用程式配置檔中提供連結和端點資訊，以管理方式建立 WCF 自訂通道』 所述

關於這項作業

1. 在服務主機中實例化並開啟服務實例。 服務類型必須與服務配置檔中指定的服務類型相同。
2. 將下列程式碼新增至應用程式:

```
ServiceHost service = new ServiceHost(typeof(MyService));
service.Open();
...
service.Close();
```

使用方法 *2* 建置 *WCF* 服務應用程式*:* 直接從應用程式以程式化方式自行管理

新增連結內容，使用所需服務類別的實例來建立服務主機，然後開啟服務。

開始之前

1. 將自訂通道 `IBM.XMS.WCF.dll` 檔的參照新增至專案。 `IBM.XMS.WCF.dll` 位於 *WMQInstallDir*`\bin` 中，其中 *WMQInstallDir* 是 IBM MQ 的安裝目錄。
2. 將 使用 陳述式新增至 `IBM.XMS.WCF` 名稱空間，例如: `using IBM.XMS.WCF`
3. 建立通道連結元素和端點的實例，如 第 1075 頁的『以程式化方式提供連結和端點資訊來建立 WCF 自訂通道』 所述。

關於這項作業

如果需要變更通道的連結內容，請完成下列步驟:

1. 將連結內容新增至 `transportBindingElement` ，如下列範例所示:

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
Uri address = new Uri("jms:/queue?destination=SampleQ@QM1&connectionFactory=

connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

2. 使用所需服務類別的實例來建立服務主機:

```
ServiceHost service = new ServiceHost(typeof(MyService));
```

3. 開啟服務:

```
service.AddServiceEndpoint(typeof(IMyServiceContract), binding, address);
service.Open();
...
service.Close();
```

使用 *HTTP* 端點公開 *meta* 資料
公開配置為使用 WCF 的 IBM MQ 自訂通道之服務 meta 資料的指示。

關於這項作業
如果必須公開服務 meta 資料 (例如，使 `svcutil` 之類的工具可以直接從執行中服務而非離線 WSDL 檔來存取它)，則必須透過使用 HTTP 端點公開服務 meta 資料來完成。 下列步驟可用來新增此額外端點。

1. 新增必須向 ServiceHost 公開 meta 資料的基本位址，例如:

```
ServiceHost service = new ServiceHost(typeof(TestService),
                      new Uri("http://localhost:8000/MyService"));
```

2. 在開啟服務之前，請將下列程式碼新增至 ServiceHost :

```
ServiceMetadataBehavior metadataBehavior = new ServiceMetadataBehavior();
  metadataBehavior.HttpGetEnabled = true;
  service.Description.Behaviors.Add(metadataBehavior);
  service.AddServiceEndpoint(typeof(IMetadataExchange),
     MetadataExchangeBindings.CreateMexHttpBinding(), "mex");
```

結果
meta 資料現在位於下列位址: `http://localhost:8000/MyService`

建置 WCF 的用戶端應用程式

產生及建置 Microsoft Windows Communication Foundation (WCF) 用戶端應用程式的概觀。

可以為 WCF 服務建立用戶端應用程式; 通常會使用 Microsoft ServiceModel Metadata Utility Tool (Svcutil.exe) 來產生用戶端應用程式，以建立可供應用程式直接使用的必要配置及 Proxy 檔案。

使用 *svcutil* 工具搭配執行中服務的 *meta* 資料來產生 *WCF* 用戶端 *Proxy* 及應用程式配置檔
使用 Microsoft svcutil.exe 工具為服務產生用戶端的指示，該服務配置為使用 WCF 的 IBM MQ 自訂通道。

開始之前
使用 svcutil 工具來建立應用程式可直接使用的必要配置及 Proxy 檔案，有三個必要條件:

- 在啟動 svcutil 工具之前， WCF 服務必須在執行中。
- 除了 IBM MQ 自訂通道端點參照之外， WCF 服務還必須使用 HTTP 埠來公開其 meta 資料，以直接從執行中服務產生用戶端。
- 自訂通道必須登錄在 svcutil 的配置資料中。

關於這項作業
下列步驟說明如何為配置為使用 IBM MQ 自訂通道的服務產生用戶端，但也會在執行時期透過個別 HTTP 埠公開其 meta 資料:

1. 啟動 WCF 服務 (在啟動 svcutil 工具之前，服務必須在執行中)。
2. 將 `svcutil.exe` 配置檔中的詳細資料從安裝根目錄新增至作用中 svcutil 配置檔，通常是 `C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\svcutil.exe.config`，因此 svcutil 可辨識 IBM MQ 自訂通道。
3. 從命令提示字元執行 svcutil，例如:

```
svcutil /language:C# /r: installlocation\bin\IBM.XMS.WCF.dll
/config:app.config http://localhost:8000/IBM.XMS.WCF/samples
```

4. 將產生的 `app.config` 和 `YourService.cs` 檔案複製到 Microsoft Visual Studio 用戶端專案。

下一步

如果無法直接擷取服務 meta 資料，則可以改用 svcutil 從 wsdl 產生用戶端檔案。 如需相關資訊，請參閱: 第 1082 頁的『使用 svcutil 工具搭配 WSDL 來產生 WCF 用戶端 Proxy 及應用程式配置檔』

使用 *svcutil* 工具搭配 *WSDL* 來產生 *WCF* 用戶端 *Proxy* 及應用程式配置檔

如果服務的 meta 資料無法使用，從 WSDL 產生 WCF 用戶端的指示。

如果無法直接從執行中服務的 meta 資料擷取服務的 meta 資料來產生用戶端，則可以改用 svcutil 來從 WSDL 產生用戶端檔案。 必須對 WSDL 進行下列修改，以指定要使用 IBM MQ 自訂通道:

1. 新增下列名稱空間定義及原則資訊:

```
<wsdl:definitions
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd">

        <wsp:Policy wsu:Id="CustomBinding_IWMQSampleContract_policy">
            <wsp:ExtactlyOne>
                <wsp:All>
                    <xms:xms xmlns:xms="http://sample.schemas.ibm.com/policy/xms" />
                </wsp:All>
            </wsp:ExactlyOne>
        </wsp:Policy>

...

</wsdl:definitions>
```

2. 修改連結區段以參照新的原則區段，並從基礎連結元素中移除任何 `transport` 定義:

```
<wsdl:definitions ...>

        <wsdl:binding ...>
                <wsp:PolicyReference URI="#CustomerBinding_IWMQSampleContract_policy" />
                <[soap]:binding ... transport="" />
                ...
        </wsdl:binding>
</wsdl:definitions>
```

3. 從命令提示字元執行 svcutil ， 例如:

```
svcutil /language:C# /r:  MQ_INSTALLATION_PATH\bin\IBM.XMS.WCF.dll
/config:app.config  MQ_INSTALLATION_PATH\src\samples\WMQAxis\default\service
\soap.server.stockQuoteAxis_Wmq.wsdl
```

使用用戶端 *Proxy* 搭配應用程式配置檔來建置 *WCF* 用戶端應用程式

開始之前

建立或編輯用戶端的應用程式配置檔，如 第 1074 頁的『在應用程式配置檔中提供連結和端點資訊，以管理方式建立 WCF 自訂通道』 所述

關於這項作業

實例化並開啟用戶端 Proxy 的實例。 傳遞至所產生 Proxy 的參數必須與用戶端配置檔中指定的端點名稱相同，例如 `Endpoint_WMQ`:

```
MyClientProxy myClient = new MyClientProxy("Endpoint_WMQ");
            try {
                   myClient.myMethod("HelloWorld!");
                   myClient.Close();
            }
            catch (TimeoutException e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
            catch (CommunicationException e) {
```

```
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
            catch (Exception e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
```

使用用戶端 *Proxy* 搭配程式化配置來建置 *WCF* 用戶端應用程式

開始之前

1. 將自訂通道 IBM.XMS.WCF.dll 檔的參照新增至專案。 IBM.XMS.WCF.dll 位於 *WMQInstallDir*\bin 目錄中，其中 *WMQInstallDir* 是 IBM MQ 的安裝目錄。
2. 將 使用 陳述式新增至 IBM.XMS.WCF 名稱空間，例如: `using IBM.XMS.WCF`
3. 依照 第 1075 頁的『以程式化方式提供連結和端點資訊來建立 WCF 自訂通道』 中的說明，建立通道的第個連結元素和端點的實例。

關於這項作業

如果需要變更通道的連結內容，請完成下列步驟。

1. 將連結內容新增至 `transportBindingElement` ，如下圖所示:

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
EndpointAddress address =
  new EndpointAddress("jms:/queue?destination=SampleQ@QM1&connectionFactory=
     connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

2. 建立用戶端 Proxy ，如下圖所示，其中 *binding* 和 *endpoint address* 是在步驟 1 中配置並傳入的連結和端點位址:

```
MyClientProxy myClient = new MyClientProxy(binding, endpoint address);
            try {
                   myClient.myMethod("HelloWorld!");
                   myClient.Close();
            }
            catch (TimeoutException e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
            catch (CommunicationException e) {
                Console.Out.WriteLine(e);
                                myClient.Abort();
            }
            catch (Exception e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
```

使用 WCF 範例

Windows Communication Foundation (WCF) 範例提供一些簡單範例，說明如何使用 IBM MQ 自訂通道。

若要建置範例專案，需要 Microsoft.NET 3.5 SDK 或 Microsoft Visual Studio 2008。

簡式單向用戶端及伺服器 WCF 範例

此範例示範使用單向通道形狀從 WCF 用戶端啟動 Windows 通訊基礎 (WCF) 服務時所使用的 IBM MQ 自訂通道。

關於這項作業

服務會實作單一方法，將字串輸出至主控台。 已使用 `svcutil` 工具產生用戶端，以從個別公開的 HTTP 端點擷取服務 meta 資料，如 第 1081 頁的『使用 svcutil 工具搭配執行中服務的 meta 資料來產生 WCF 用戶端 Proxy 及應用程式配置檔』 中所述

已使用下列程序中說明的特定資源名稱來配置範例。 如果您必須變更資源名稱，則還必須在 *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\client\app.config` 檔案中變更用戶端應用程式上的對應值，以及在 *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\TestServices.cs` 檔案中變更服務應用程式上的對應值，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安裝目錄。 如需格式化 JMS 端點 URI 的相關資訊，請參閱 IBM MQ 產品說明文件中的 *IBM MQ SOAP* 的傳輸 。 如果您需要修改範例解決方案和來源，則需要 IDE ，例如 Microsoft Visual Studio 8 或更高版本。

程序

1. 建立稱為 *QM1* 的佇列管理程式
2. 建立稱為 *SampleQ* 的佇列目的地
3. 啟動服務，讓接聽器等待訊息: 執行 *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\bin\Release\TestService.exe` 檔案，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安裝目錄。
4. 執行用戶端一次: 執行 *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\client\bin\Release\TestClient.exe` 檔案，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安裝目錄。

   用戶端應用程式五次迴圈將五則訊息傳送至 *SampleQ*

結果

服務應用程式會從 *SampleQ* 取得訊息，並在畫面上顯示 `Hello World` 五次。

下一步

簡式要求-回覆用戶端及伺服器 WCF 範例

此範例示範使用要求/回覆通道形狀，從 WCF 用戶端啟動「Windows 通訊基礎 (WCF)」服務時所使用的 IBM MQ 自訂通道。

關於這項作業

此服務提供一些簡單的計算機方法來加減兩個數字，然後傳回結果。 已使用 `svcutil` 工具產生用戶端，以從個別公開的 HTTP 端點擷取服務 meta 資料，如 第 1081 頁的『使用 svcutil 工具搭配執行中服務的 meta 資料來產生 WCF 用戶端 Proxy 及應用程式配置檔』 中所述

範例已配置特定的資源名稱，如下列程序所述。 如果您需要變更資源名稱，則還需要在 *MQ_INSTALLATION_PATH* `\Tools\wcf\samples\WCF\requestreply\client\app.config` 檔案中變更用戶端應用程式上的對應值，以及在 *MQ_INSTALLATION_PATH* `\Tools\wcf\samples\WCF\requestreply\service\RequestReplyService.cs` 檔案中變更服務應用程式上的對應值，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安裝目錄。 如需格式化 JMS 端點 URI 的相關資訊，請參閱 IBM MQ 產品說明文件中的 *IBM MQ SOAP* 的傳輸 。 如果您需要修改範例解決方案和來源，則需要 IDE ，例如 Microsoft Visual Studio 8 或更高版本。

程序

1. 建立稱為 *QM1* 的佇列管理程式
2. 建立稱為 *SampleQ* 的佇列目的地
3. 建立稱為 *SampleReplyQ* 的佇列目的地

4. 啟動服務，讓接聽器等待訊息: 執行 *MQ_INSTALLATION_PATH* \Tools\wcf\samples\WCF\requestreply\service\bin\Release\SimpleRequestReply_Service.exe 檔案，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安裝目錄。
5. 執行用戶端一次: 執行 *MQ_INSTALLATION_PATH* \Tools\wcf\samples\WCF\requestreply\client\bin\Release\SimpleRequestReply_Client.exe 檔案，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安裝目錄。

結果

當用戶端已執行時，下列處理程序會啟動並重複四次，因此每一種方式總共會傳送五則訊息:

1. 用戶端會在 *SampleQ* 上放置要求訊息，並等待回應。
2. 服務會從 *SampleQ* 取得要求訊息。
3. 服務會使用訊息內容來新增及扣除部分值。
4. 然後，服務會將結果放入 *SampleReplyQ* 上的訊息，並等待用戶端放置新訊息。
5. 用戶端會從 *SampleReplyQ* 取得訊息，並在畫面上顯示結果。

下一步

WCF 用戶端至 IBM MQ 範例所管理的 .NET 服務

同時提供 .NET 和 Java 的範例用戶端應用程式和範例服務 Proxy 應用程式。 這些範例基於「股票報價」服務，該服務接受對股票報價的要求，然後提供股票報價。

開始之前

此範例要求 .NET SOAP over JMS 服務管理環境正確安裝並配置在 IBM MQ 中，並且可從本端佇列管理程式存取。

當 .NET SOAP over JMS 服務管理環境正確安裝並配置在 IBM MQ 中，且可從本端佇列管理程式存取時，必須完成其他配置步驟。

1. 將 **WMQSOAP_HOME** 環境變數設為 IBM MQ 安裝目錄，例如: `C:\Program Files\IBM\MQ`
2. 請確定 Java 編譯器 `javac` 可用且在 PATH 上。
3. 將檔案 `axis.jar` 從安裝映像檔的 `prereqs/axis` 目錄複製到 IBM MQ 正式作業目錄，例如: `C:\Program Files\IBM\MQ\java\lib\soap`
4. 新增至 PATH: *MQ_INSTALLATION_PATH*`\Java\lib`，其中 *MQ_INSTALLATION_PATH* 代表 IBM MQ 的安裝目錄，例如: `C:\Program Files\IBM\MQ`
5. 確保在 *MQ_INSTALLATION_PATH*`\bin\amqwcallWSDL.cmd` 中正確指定 .NET 的位置，其中 *MQ_INSTALLATION_PATH* 代表 IBM MQ 的安裝目錄，例如: `C:\Program Files\IBM\MQ`。 可以指定 .NET 的位置，例如: `set msfwdir=%ProgramFiles%\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin`

當上述步驟完成時，測試並執行服務:

1. 導覽至 SOAP over JMS 工作目錄。
2. 輸入下列其中一個指令，以執行驗證測試，並讓服務接聽器維持執行:
   - 若為 .NET: *MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt dotnet hold`，其中 *MQ_INSTALLATION_PATH* 代表 IBM MQ 的安裝目錄。
   - 對於 AX: *MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt Dotnet2AxisClient hold`，其中 *MQ_INSTALLATION_PATH* 代表 IBM MQ 的安裝目錄。

在測試完成之後， `hold` 引數會讓接聽器保持執行中。

如果在此配置期間報告錯誤，您可以移除所有變更，以便以下列方式重新啟動程序:

1. 刪除產生的 SOAP over JMS 目錄。
2. 刪除佇列管理程式。

關於這項作業

此範例示範從 WCF 用戶端到 IBM MQ 中提供的 .NET SOAP over JMS 範例服務 (使用單向通道形狀) 的連線。 服務會實作簡式 StockQuote 範例，它會將字串輸出至主控台。

已使用 WSDL 來產生用戶端，以產生用戶端檔案，如 第 1082 頁的『使用 svcutil 工具搭配 WSDL 來產生 WCF 用戶端 Proxy 及應用程式配置檔』 中所述

已使用下列程序中說明的特定資源名稱來配置範例。 如果您需要變更資源名稱，則還必須在 *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WMQNET\default\client\app.config` 檔案中變更用戶端應用程式上的對應值，以及在 *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WMQNET\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl` 檔案中變更服務應用程式上的對應值，其中 *MQ_INSTALLATION_PATH* 代表 IBM MQ 的安裝目錄。 如需格式化 JMS 端點 URI 的相關資訊，請參閱 IBM MQ 產品說明文件中的 *IBM MQ SOAP* 的傳輸 。

程序

執行用戶端一次: 執行 *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WMQNET\default\client\bin\Release\TestClient.exe` 檔案，其中 *MQ_INSTALLATION_PATH* 代表 IBM MQ 的安裝目錄。

用戶端應用程式會迴圈五次，將五則訊息傳送至範例佇列。

結果

服務應用程式會從範例佇列取得訊息，並在畫面上顯示 `Hello World` 五次。

WCF 用戶端至 IBM MQ 範例所管理的 Axis Java 服務

同時提供 Java 和 .NET 的範例用戶端應用程式和範例服務 Proxy 應用程式。 這些範例基於「股票報價」服務，該服務接受對股票報價的要求，然後提供股票報價。

開始之前

此範例要求在 IBM MQ 中正確安裝並配置 .NET SOAP over JMS 服務管理環境，並且可從本端佇列管理程式存取。

當 .NET SOAP over JMS 服務管理環境正確安裝並配置在 IBM MQ 中，且可從本端佇列管理程式存取時，必須完成其他配置步驟。

1. 將 **WMQSOAP_HOME** 環境變數設為 IBM MQ 安裝目錄，例如: `C:\Program Files\IBM\MQ`
2. 請確定 Java 編譯器 `javac` 可用且在 PATH 上。
3. 將檔案 `axis.jar` 從安裝映像檔的 `prereqs/axis` 目錄複製到 IBM MQ 安裝目錄。
4. 新增至 PATH: *MQ_INSTALLATION_PATH*`\Java\lib` ，其中 *MQ_INSTALLATION_PATH* 代表 IBM MQ 的安裝目錄，例如: `C:\Program Files\IBM\MQ`
5. 確保在 *MQ_INSTALLATION_PATH*`\bin\amqwcallWSDL.cmd` 中正確指定 .NET 的位置，其中 *MQ_INSTALLATION_PATH* 代表 IBM MQ 的安裝目錄，例如: `C:\Program Files\IBM\MQ`。 可以指定 .NET 的位置，例如: `set msfwdir=%ProgramFiles%\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin`

當上述步驟完成時，測試並執行服務:

1. 導覽至 SOAP over JMS 工作目錄。
2. 輸入下列其中一個指令，以執行驗證測試，並讓服務接聽器維持執行:
   - 若為 .NET: *MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt dotnet hold` ，其中 *MQ_INSTALLATION_PATH* 代表 IBM MQ 的安裝目錄。
   - 對於 AX: *MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt Dotnet2AxisClient hold` ，其中 *MQ_INSTALLATION_PATH* 代表 IBM MQ 的安裝目錄。

在測試完成之後， `hold` 引數會讓接聽器保持執行中。

如果在此配置期間報告錯誤，您可以移除所有變更，以便以下列方式重新啟動程序:

1. 刪除產生的 SOAP over JMS 目錄。
2. 刪除佇列管理程式。

關於這項作業

此範例示範使用單向通道形狀從 WCF 用戶端到 IBM MQ 中提供的 Axis Java SOAP over JMS 範例服務的連線。 服務會實作簡式 StockQuote 範例，它會將字串輸出至儲存在現行目錄中的檔案。

已使用 WSDL 來產生用戶端，以產生用戶端檔案，如 第 1082 頁的『使用 svcutil 工具搭配 WSDL 來產生 WCF 用戶端 Proxy 及應用程式配置檔』 中所述

已依照本段落的說明，以特定的資源名稱來配置範例。 如果您需要變更資源名稱，則還必須在 *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WMQAxis\default\client\app.config` 檔案中變更用戶端應用程式上的對應值，以及在 *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WMQAxis\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl` 檔案中變更服務應用程式上的對應值，其中 *MQ_INSTALLATION_PATH* 代表 IBM MQ 的安裝目錄。

程序

執行用戶端一次: 執行 *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WMQAxis\default\client\bin\Release\TestClient.exe` 檔案，其中 *MQ_INSTALLATION_PATH* 代表 IBM MQ 的安裝目錄。

用戶端應用程式會迴圈五次，將五則訊息傳送至範例佇列。

結果

服務應用程式會從範例佇列取得訊息，並將 `Hello World` 新增至現行目錄中的檔案五次。

WCF 用戶端至 WebSphere Application Server 範例所管理的 Java 服務

針對 WebSphere Application Server 6 提供了範例用戶端應用程式及範例服務 Proxy 應用程式。 也會提供要求/回應服務。

開始之前

此範例需要使用下列 IBM MQ 配置:

表 *195: IBM MQ* 必要配置

物件	**必要名稱**
佇列管理程式	QM1
本端佇列	HelloWorld
本端佇列	HelloWorld 回覆

此範例還要求正確安裝並配置 WebSphere Application Server 6 管理環境。 依預設， WebSphere Application Server 6 會使用連結模式連線來連接至 IBM MQ 。 因此， WebSphere Application Server 6 必須安裝在與佇列管理程式相同的機器上。

在配置 WAS 環境之後，必須完成下列其他配置步驟:

1. 在 WebSphere Application Server JNDI 儲存庫中建立下列 JNDI 物件:
   a. 稱為 `HelloWorld` 的 JMS 佇列目的地
      - 將 JNDI 名稱設為 `jms/HelloWorld`
      - 將佇列名稱設為 `HelloWorld`
   b. 稱為 `HelloWorldQCF` 的 JMS Queue Connection Factory
      - 將 JNDI 名稱設為 `jms/HelloWorldQCF`
      - 將佇列管理程式名稱設為 `QM1`

c. 稱為 WebServicesReplyQCF 的 JMS Queue Connection Factory
   - 將 JNDI 名稱設為 jms/WebServicesReplyQCF
   - 將佇列管理程式名稱設為 QM1

2. 使用下列配置，在 WebSphere Application Server 中建立稱為 HelloWorldPort 的「訊息接聽器埠」：
   - 將 Connection Factory JNDI 名稱設為 jms/HelloWorldQCF
   - 將目的地 JNDI 名稱設為 jms/HelloWorld
3. 將 Web 服務 HelloWorldEJBEAR.ear 應用程式安裝至 WebSphere Application Server，如下所示：
   a. 按一下 **應用程式 > 新建應用程式 > 新建企業應用程式**。
   b. 導覽至 *MQ_INSTALLATION_PATH*\tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安裝目錄。
   c. 請勿變更精靈中的任何預設選項，並在安裝應用程式之後重新啟動應用程式伺服器。

WAS 配置完成時，透過執行服務一次來測試服務：

1. 導覽至 JMS 工作目錄上的 Soap。
2. 輸入下列指令以執行範例：*MQ_INSTALLATION_PATH* \tools\wcf\samples\WAS\TestClient.exe，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安裝目錄。

關於這項作業

此範例使用要求/回應通道形狀，示範從 WCF 用戶端到 IBM MQ 所含 WCF 範例中所提供之 WebSphere Application Server SOAP over JMS 範例服務的連線。 使用 IBM MQ 佇列在 WCF 與 WebSphere Application Server 之間傳送訊息。 服務會實作 HelloWorld(...) 方法，它會採用字串並傳回問候語給用戶端。

已使用 svcutil 工具來產生用戶端，以從個別公開的 HTTP 端點擷取服務 meta 資料，如 第 1081 頁的『使用 svcutil 工具搭配執行中服務的 meta 資料來產生 WCF 用戶端 Proxy 及應用程式配置檔』 中所述。

已使用下列程序中說明的特定資源名稱來配置範例。 如果您需要變更資源名稱，則還必須在 *MQ_INSTALLATION_PATH* \tools\wcf\samples\WAS\default\client\app.config 檔案中變更用戶端應用程式上的對應值，以及在 *MQ_INSTALLATION_PATH* \tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear 中變更服務應用程式上的對應值，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安裝目錄。

服務及用戶端是以 IBM Developer 文章 使用 *SOAP over JMS* 和 *WebSphere Studio* 建置 *JMS Web* 服務中所概述的服務及用戶端為基礎。 如需開發與 IBM MQ WCF 自訂通道相容之 SOAP over JMS Web 服務的相關資訊，請參閱 https://www.ibm.com/developerworks/websphere/library/techarticles/0402_du/0402_du.html。

程序

執行用戶端一次：執行 *MQ_INSTALLATION_PATH* \tools\wcf\samples\WAS\default\client\bin\Release\TestClient.exe 檔案，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安裝目錄。

用戶端應用程式會同時啟動這兩種服務方法，並將兩則訊息傳送至範例佇列。

結果

服務應用程式會從範例佇列取得訊息，並對用戶端應用程式輸出至主控台的 HelloWorld(...) 方法呼叫提供回應。

注意事項

本資訊係針對 IBM 在美國所提供之產品與服務所開發。

在其他國家或地區中，IBM 可能未提供本文件所提及的各項產品、服務或功能。 請洽當地 IBM 業務代表，以取得當地目前提供的產品和服務之相關資訊。 本文件在提及 IBM 產品、程式或服務時，不表示或暗示只能使用 IBM 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，任何非 IBM 之產品、程式或服務，使用者必須自行負責作業之評估和驗證責任。

本文件所說明之主題內容，IBM 可能擁有其專利或專利申請案。 提供本文件不代表提供這些專利的授權。 您可以書面提出授權查詢，來函請寄到：

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

如果是有關雙位元組 (DBCS) 資訊的授權查詢，請洽詢所在國的 IBM 智慧財產部門，或書面提出授權查詢，來函請寄到：

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

下列段落不適用於英國，若與任何其他國家之法律條款抵觸，亦不適用於該國： International Business Machines Corporation 只依 "現況" 提供本出版品，不提供任何明示或默示之保證，其中包括且不限於不侵權、可商用性或特定目的之適用性的隱含保證。 有些地區在特定交易上，不允許排除明示或暗示的保證，因此，這項聲明不一定適合您。

本資訊中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 得隨時改進並（或）變動本書中所提及的產品及（或）程式。

本資訊中任何對非 IBM 網站的敘述僅供參考，IBM 對該網站並不提供任何保證。 這些網站所提供的資料不是 IBM 本產品的資料內容，如果要使用這些網站的資料，您必須自行承擔風險。

IBM 得以各種適當的方式使用或散布由您提供的任何資訊，無需對您負責。

如果本程式的獲授權人為了 (i) 在個別建立的程式和其他程式（包括本程式）之間交換資訊，以及 (ii) 相互使用所交換的資訊，因而需要相關的資訊，請洽詢：

IBM Corporation
軟體交互作業能力協調程式，部門 49XA
3605 公路 52 N
Rochester， MN 55901
U.S.A.

在適當條款與條件之下，包括某些情況下（支付費用），或可使用此類資訊。

IBM 基於雙方之 IBM 客戶合約、IBM 國際程式授權合約或任何同等合約之條款，提供本資訊所提及的授權程式與其所有適用的授權資料。

本文件中所含的任何效能資料都是在受管制的環境下判定。 因此，在其他作業環境下取得的結果可能大不相同。 有些測定已在開發階段系統上做過，不過這並不保證在一般系統上會出現相同結果。 甚至有部分的測量，是利用插補法而得的估計值， 實際結果可能有所不同。 本書的使用者應依自己的特定環境，查證適用的資料。

© Copyright IBM Corp. 2007, 2024

本文件所提及之非 IBM 產品資訊，取自產品的供應商，或其發佈的聲明或其他公開管道。 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。 有關非 IBM 產品的性能問題應直接洽詢該產品供應商。

有關 IBM 未來動向的任何陳述，僅代表 IBM 的目標而已，並可能於未事先聲明的情況下有所變動或撤回。

這份資訊含有日常商業運作所用的資料和報告範例。 為了要使它們儘可能完整，範例包括個人、公司、 品牌和產品的名稱。 所有這些名稱都是虛構的，如有任何類似實際企業所用的名稱及地址之處，純屬巧合。

著作權授權:

本資訊含有原始語言之範例應用程式，用以說明各作業平台中之程式設計技術。 您可以基於研發、使用、銷售或散布符合作業平台（撰寫範例程式的作業平台）之應用程式介面的應用程式等目的，以任何形式複製、修改及散布這些範例程式，而不必向 IBM 付費。 這些範例並未在所 有情況下完整測試。 因此，IBM 不保證或暗示這些程式的可靠性、服務性或功能。

若 貴客戶正在閱讀本項資訊的電子檔，可能不會有照片和彩色說明。

程式設計介面資訊

程式設計介面資訊 (如果有提供的話) 旨在協助您建立與此程式搭配使用的應用軟體。

本書包含預期程式設計介面的相關資訊，可讓客戶撰寫程式以取得 WebSphere MQ 的服務。

不過，本資訊也可能包含診斷、修正和調整資訊。 提供診斷、修正和調整資訊，是要協助您進行應用軟體的除錯。

重要: 請勿使用此診斷、修改及調整資訊作為程式設計介面，因為它可能會變更。

商標

IBM、 IBM 標誌 ibm.com 是 IBM Corporation 在全球許多適用範圍的商標。 IBM 商標的最新清單可在 Web 的 "Copyright and trademark information"www.ibm.com/legal/copytrade.shtml 中找到。 其他產品及服務名稱可能是 IBM 或其他公司的商標。

Microsoft 及 Windows 是 Microsoft Corporation 在美國及/或其他國家或地區的商標。

UNIX 是 The Open Group 在美國及/或其他國家/地區的註冊商標。

Linux 是 Linus Torvalds 在美國及/或其他國家或地區的註冊商標。

本產品包含 Eclipse Project (https://www.eclipse.org/) 所開發的軟體。

Java 和所有以 Java 為基礎的商標及標誌是 Oracle 及/或其子公司的商標或註冊商標。

產品編號:

(1P) P/N: