

7.5

Mobile Messaging and M2M

IBM

附註

使用本資訊及其支援的產品之前，請先閱讀第 167 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® WebSphere MQ 7.5 版及所有後續版次與修訂。

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

Mobile Messaging and M2M.....	5
MQTT 簡介.....	7
開始使用 MQTT 用戶端.....	9
開始使用適用於 Java 的 MQTT 用戶端.....	11
在 Android 上開始使用適用於 Java 的 MQTT 用戶端.....	16
適用於 JavaScript 的 MQTT 傳訊用戶端入門.....	21
開始使用適用於 C 的 MQTT 用戶端.....	23
在 iOS 上開始使用適用於 C 的 MQTT 用戶端.....	43
MQTT 指令行程式範例.....	45
MQTT 安全.....	47
建置及執行安全的 MQTT 用戶端範例 Java 應用程式.....	50
透過 SSL 連接 Android 上的 MQTT 用戶端範例 Java 應用程式.....	58
向 JAAS 鑑別 MQTT 用戶端 Java 應用程式.....	67
透過 SSL 及 WebSockets 連接適用於 JavaScript 的 MQTT 傳訊用戶端.....	71
建置及執行安全的 MQTT 用戶端範例 C 應用程式.....	79
產生金鑰及憑證.....	88
MQTT 用戶端識別、授權和鑑別.....	94
使用 SSL 進行遙測通道鑑別.....	98
遙測通道上的發佈保密.....	100
MQTT 用戶端和遙測通道的 SSL 配置.....	100
遙測通道 JAAS 配置.....	105
程式設計概念.....	106
適用於 JavaScript 的 MQTT 傳訊用戶端及 Web 應用程式.....	106
如何在 JavaScript 中對傳訊應用程式進行程式設計.....	109
MQTT 用戶端應用程式中的回呼及同步處理.....	113
清除階段作業.....	115
用戶端 ID.....	116
遞送記號.....	116
最後留言發佈.....	117
MQTT 用戶端中的訊息持續性.....	117
發佈.....	118
MQTT 用戶端提供的服務品質.....	119
保留的發佈及 MQTT 用戶端.....	120
訂閱.....	121
MQTT 用戶端中的主題字串及主題過濾器.....	121
MQTT 用戶端程式設計參考手冊.....	122
開始使用 MQTT 伺服器.....	122
IBM WebSphere MQ 作為 MQTT 伺服器.....	124
用於裝置的 IBM WebSphere MQ Telemetry 常駐程式概念.....	134
疑難排解 MQTT 用戶端.....	143
Telemetry 日誌、錯誤日誌和配置檔的位置.....	144
MQTT v3 Java 用戶端原因碼.....	146
追蹤遙測 (MQXR) 服務.....	147
追蹤 MQTT 第 3 版 Java 用戶端.....	148
追蹤適用於 C 的 MQTT 用戶端.....	150
追蹤及除錯 MQTT (Paho) Java 用戶端.....	151
追蹤 MQTT JavaScript 用戶端.....	153
對 MQTT 用戶端使用 SHA-2 密碼組合的系統需求.....	154
瀏覽器對於使用 SSL 之行動式傳訊 Web 應用程式的支援限制.....	155
解決問題：MQTT 用戶端未連接.....	158
解決問題：MQTT 用戶端連線中斷.....	160
解決問題：MQTT 應用程式中遺失訊息.....	160

解決問題：遙測 (MQXR) 服務未啟動.....	162
解決問題：Telemetry 服務未呼叫 JAAS 登入模組.....	163
解決問題：啟動或執行常駐程式.....	165
解決問題：MQTT 用戶端未連接至常駐程式.....	166
注意事項.....	167
程式設計介面資訊.....	168
商標.....	168

MQTT 簡介

瞭解如何使用MQ 遙測傳輸 (MQTT) 在行動式應用程式之間傳送訊息。此通訊協定預期在無線及低頻寬網路上使用。使用MQTT 的行動式應用程式透過呼叫MQTT 程式庫傳送及接收訊息。透過MQTT 傳訊伺服器交換訊息。MQTT 用戶端及伺服器可針對行動式應用程式可靠地處理遞送訊息的複雜性，並保持低成本地進行網路管理。

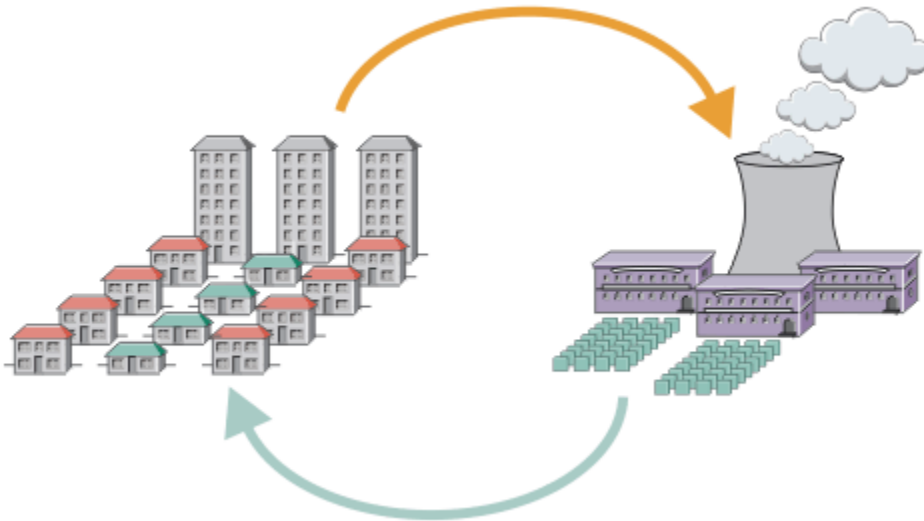
MQTT 應用程式在行動式裝置（例如，智慧型手機及平板電腦）上執行。MQTT 也用於遙測，來從感應器接收資料，及從遠端控制資料。對於行動式裝置及感應器，MQTT 提供了可高度擴充的發佈/訂閱通訊協定，且保證遞送。若要傳送及接收MQTT 訊息，請將MQTT 用戶端程式庫新增至您的應用程式。

MQTT 用戶端程式庫比較小。程式庫如同郵箱，使用連接至MQTT 伺服器的其他MQTT 應用程式傳送及接收訊息。透過傳送訊息（而不是保持連接至等待回應的伺服器），MQTT 應用程式可節約電池壽命。程式庫透過執行MQTT version 3.1 通訊協定的MQTT 伺服器將訊息傳送至其他裝置。您可以將訊息傳送至特定的用戶端，或使用發佈/訂閱傳訊來連接許多裝置。

MQTT 用戶端程式庫會使用MQTT 通訊協定，將用於行動式裝置及感應器的應用程式連接至MQTT 伺服器。

IBM MessageSight 及 IBM WebSphere MQ 是MQTT 伺服器。它們可以連接大量的MQTT 用戶端應用程式，且可以將MQTT 及 IBM WebSphere MQ 網路連接在一起。請參閱第 122 頁的『開始使用MQTT 伺服器』。IBM WebSphere MQ 及 IBM MessageSight 可在執行於行動式裝置與感應器上的外部 Web 應用程式，以及其他在企業內執行的發佈/訂閱與傳訊類型應用程式之間，形成橋接器。此橋接器可讓您更輕鬆地建置 "智慧型解決方案"，以納入行動式裝置及感應器。

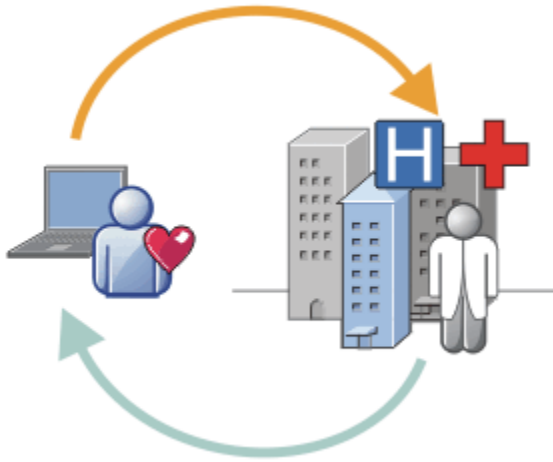
智慧型解決方案可將實際網路上可用的大量資訊解除鎖定至行動式裝置及感應器裝置上執行的應用程式。基於遙測的兩個智慧型應用程式範例是智慧型電力服務及智慧型健康服務。



- MQTT 訊息，包含傳送至服務提供者的電力使用資料。
- 遙測應用程式傳送基於電力使用資料分析的控制指令。
- 如需相關資訊，請參閱遙測實務範例：家庭電力監視及控制。

圖 1: 智慧型電力計量

圖 2: 智慧型健康監視



- 遙測應用程式將您的健康資料傳送至醫院及醫生。
- 可以根據健康資料的分析傳送 MQTT 訊息警示或意見。
- 如需相關資訊，請參閱遙測實務範例：家庭病患監視。

您可以撰寫自己的 MQTT protocol 應用程式，將 MQTT 建置成小型裝置。為協助您執行此動作，IBM 提供了用戶端程式庫，可支援透過 MQTT 執行的應用程式。請參閱第 9 頁的『開始使用 MQTT 用戶端』。

IBM 提供了適用於 iOS 應用程式與 Android 應用程式的用戶端程式庫 **V7.5.0.1**，以及適用於平台不明之 Web 應用程式的 JavaScript 瀏覽器用戶端。 **V7.5.0.1** JavaScript 用戶端頁面透過 WebSockets 使用 MQTT 通訊協定連接至 IBM MessageSight 及 IBM WebSphere MQ。IBM 也在 Linux® 和 Windows 上提供適用於 C 和 Java 的 MQTT 範例應用程式。

C 和 Java 程式庫在 iOS、Android、Windows 及許多 UNIX and Linux 平台上執行。您可以將用於 MQTT 用戶端程式庫的 C 原始碼連接至其他平台。適用於 C 及 Java 的 MQTT 用戶端程式庫，可透過 Eclipse Paho 專案的開放程式碼授權取得。請參閱 [Eclipse Paho](#)。MQTT protocol 是開放式規格，可從 [MQTT.org](#) 取得。

MQTT protocol

輕量型 MQTT protocol 意味著用戶端較小，可有效地使用網路頻寬。MQTT 通訊協定支援保證遞送及「隨發即忘」傳送。在通訊協定中，從應用程式取消連結訊息遞送。應用程式中取消連結的範圍視寫入 MQTT 用戶端及 MQTT 伺服器的方式而定。取消連結的遞送會從任何伺服器連線及從等待訊息釋放應用程式。互動模型如同電子郵件，但已針對應用程式程式設計進行最佳化。

MQTT V3.1 通訊協定已發佈；請參閱 [MQTT V3.1 Protocol Specification](#)。此規格識別有關通訊協定的許多特有的特性：

- 它是發佈/訂閱通訊協定。
 - 除提供一對多訊息配送之外，發佈/訂閱還會取消連結應用程式。在具有許多用戶端的應用程式中，這兩種特性都很有用。
- 無論如何它不依賴於訊息內容。
- 它透過 TCP/IP 執行，TCP/IP 提供基本網路連線功能。
- 它具有三種用於訊息遞送的服務品質：
 - "至多一次"**
根據基礎「網際網路通訊協定」網路的最佳效能遞送訊息。訊息可能會遺失。
例如，將此服務品質與通訊的環境感測器資料搭配使用。如果不久後會發佈下一個閱讀，則個別閱讀是否遺失並不重要。
 - "至少一次"**
保證訊息會送達，但可能會重複。
 - "只一次"**
保證訊息只送達一次。

例如，將此服務品質與帳單系統搭配使用。訊息重複或遺失可能會導致不方便或徵收不正確的費用。

- 在網路上管理訊息流程的方式非常經濟。例如，固定長度標頭的長度僅 2 個位元組，且會最大限度地減少通訊協定交換以減少網路資料流量。
- 它具有 "最後留言" 功能，可將用戶端與 MQTT 伺服器的異常斷線通知訂閱者。請參閱第 117 頁的『最後留言發佈』。

IBM WebSphere MQ 和 IBM MessageSight 支援 MQTT version 3.1。MQTT 透過 TCP/IP 實作。此通訊協定的另一版本 MQTT-S 可用於非 TCP/IP 網路。請參閱 [MQTT-S version 1.2 規格](#)。

MQTT 社群

IBM 正在執行 [IBM Developer 傳訊 社群](#)，適用於為 IBM MessageSight 和 IBM WebSphere MQ 撰寫應用程式的 MQTT 開發人員。

[MQTT.org](#) 是瞭解及討論 MQTT 通訊協定的實作和延伸的好地方。

MQTT 是 Eclipse Technology Project 下的開放程式碼 Eclipse 專案。Paho 社群正在開發開放程式碼用戶端及伺服器。請參閱 [Eclipse Paho](#)。

MQTT 簡介

瞭解如何使用 MQ 遙測傳輸 (MQTT) 在行動式應用程式之間傳送訊息。此通訊協定預期在無線及低頻寬網路上使用。使用 MQTT 的行動式應用程式透過呼叫 MQTT 程式庫傳送及接收訊息。透過 MQTT 傳訊伺服器交換訊息。MQTT 用戶端及伺服器可針對行動式應用程式可靠地處理遞送訊息的複雜性，並保持低成本地進行網路管理。

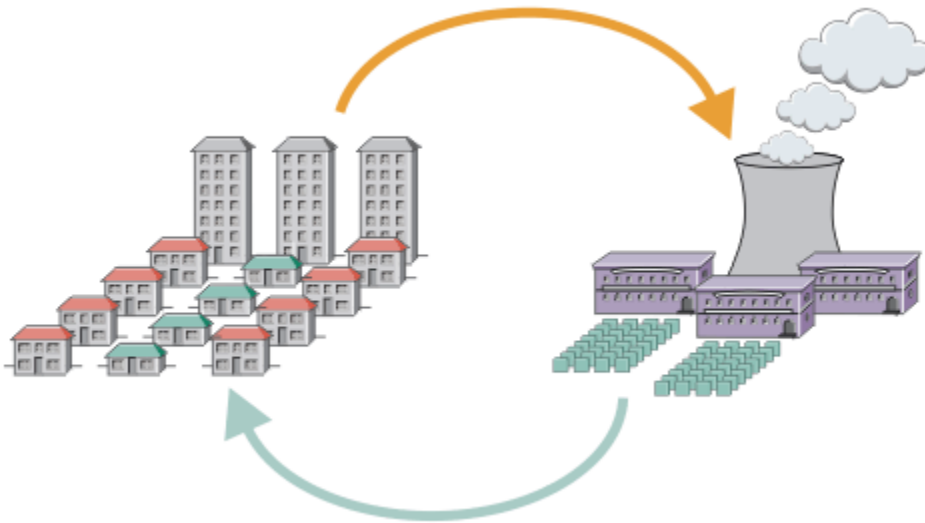
MQTT 應用程式在行動式裝置（例如，智慧型手機及平板電腦）上執行。MQTT 也用於遙測，來從感應器接收資料，及從遠端控制資料。對於行動式裝置及感應器，MQTT 提供了可高度擴充的發佈/訂閱通訊協定，且保證遞送。若要傳送及接收 MQTT 訊息，請將 MQTT 用戶端程式庫新增至您的應用程式。

MQTT 用戶端程式庫比較小。程式庫如同郵箱，使用連接至 MQTT 伺服器的其他 MQTT 應用程式傳送及接收訊息。透過傳送訊息（而不是保持連接至等待回應的伺服器），MQTT 應用程式可節約電池壽命。程式庫透過執行 MQTT version 3.1 通訊協定的 MQTT 伺服器將訊息傳送至其他裝置。您可以將訊息傳送至特定的用戶端，或使用發佈/訂閱傳訊來連接許多裝置。

MQTT 用戶端程式庫會使用 MQTT 通訊協定，將用於行動式裝置及感應器的應用程式連接至 MQTT 伺服器。

IBM MessageSight 及 IBM WebSphere MQ 是 MQTT 伺服器。它們可以連接大量的 MQTT 用戶端應用程式，且可以將 MQTT 及 IBM WebSphere MQ 網路連接在一起。請參閱第 122 頁的『開始使用 MQTT 伺服器』。IBM WebSphere MQ 及 IBM MessageSight 可在執行於行動式裝置與感應器上的外部 Web 應用程式，以及其他在企業內執行的發佈/訂閱與傳訊類型應用程式之間，形成橋接器。此橋接器可讓您更輕鬆地建置 "智慧型解決方案"，以納入行動式裝置及感應器。

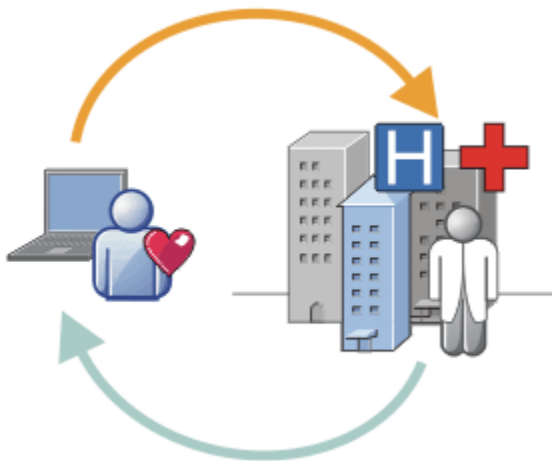
智慧型解決方案可將網際網路上可用的大量資訊解除鎖定至行動式裝置及感應器裝置上執行的應用程式。基於遙測的兩個智慧型應用程式範例是智慧型電力服務及智慧型健康服務。



- MQTT 訊息，包含傳送至服務提供者的電力使用資料。
- 遙測應用程式傳送基於電力使用資料分析的控制指令。
- 如需相關資訊，請參閱遙測實務範例：家庭電力監視及控制。

圖 3: 智慧型電力計量

圖 4: 智慧型健康監視



- 遙測應用程式將您的健康資料傳送至醫院及醫生。
- 可以根據健康資料的分析傳送 MQTT 訊息警示或意見。
- 如需相關資訊，請參閱遙測實務範例：家庭病患監視。

您可以撰寫自己的 MQTT protocol 應用程式，將 MQTT 建置成小型裝置。為協助您執行此動作，IBM 提供了用戶端程式庫，可支援透過 MQTT 執行的應用程式。請參閱第 9 頁的『開始使用 MQTT 用戶端』。

IBM 提供了適用於 iOS 應用程式與 Android 應用程式的用戶端程式庫 **V7.5.0.1**，以及適用於平台不明之 Web 應用程式的 JavaScript 瀏覽器用戶端。 **V7.5.0.1** JavaScript 用戶端頁面透過 WebSockets 使用 MQTT 通訊協定連接至 IBM MessageSight 及 IBM WebSphere MQ。IBM 也在 Linux 和 Windows 上提供適用於 C 和 Java 的 MQTT 範例應用程式。

C 和 Java 程式庫在 iOS、Android、Windows 及許多 UNIX and Linux 平台上執行。您可以將用於 MQTT 用戶端程式庫的 C 原始碼連接至其他平台。適用於 C 及 Java 的 MQTT 用戶端程式庫，可透過 Eclipse Paho 專案的開放程式碼授權取得。請參閱 [Eclipse Paho](#)。MQTT protocol 是開放式規格，可從 [MQTT.org](#) 取得。

MQTT protocol

輕量型 MQTT protocol 意味著用戶端較小，可有效地使用網路頻寬。MQTT 通訊協定支援保證遞送及「隨發即忘」傳送。在通訊協定中，從應用程式取消連結訊息遞送。應用程式中取消連結的範圍視寫入 MQTT 用戶端及 MQTT 伺服器的方式而定。取消連結的遞送會從任何伺服器連線及從等待訊息釋放應用程式。互動模型如同電子郵件，但已針對應用程式程式設計進行最佳化。

MQTT V3.1 通訊協定已發佈；請參閱 [MQTT V3.1 Protocol Specification](#)。此規格識別有關通訊協定的許多特有的特性：

- 它是發佈/訂閱通訊協定。

除提供一對多訊息配送之外，發佈/訂閱還會取消連結應用程式。在具有許多用戶端的應用程式中，這兩種特性都很有用。

- 無論如何它不依賴於訊息內容。
- 它透過 TCP/IP 執行，TCP/IP 提供基本網路連線功能。
- 它具有三種用於訊息遞送的服務品質：

"至多一次"

根據基礎「網際網路通訊協定」網路的最佳效能遞送訊息。訊息可能會遺失。

例如，將此服務品質與通訊的環境感應器資料搭配使用。如果不久後會發佈下一個閱讀，則個別閱讀是否遺失並不重要。

"至少一次"

保證訊息會送達，但可能會重複。

"只一次"

保證訊息只送達一次。

例如，將此服務品質與帳單系統搭配使用。訊息重複或遺失可能會導致不方便或徵收不正確的費用。

- 在網路上管理訊息流程的方式非常經濟。例如，固定長度標頭的長度僅 2 個位元組，且會最大限度地減少通訊協定交換以減少網路資料流量。
- 它具有 "最後留言" 功能，可將用戶端與 MQTT 伺服器的異常斷線通知訂閱者。請參閱第 117 頁的『最後留言發佈』。

IBM WebSphere MQ 和 IBM MessageSight 支援 MQTT version 3.1。MQTT 透過 TCP/IP 實作。此通訊協定的另一版本 MQTT-S 可用於非 TCP/IP 網路。請參閱 [MQTT-S version 1.2 規格](#)。

MQTT 社群

IBM 正在執行 [IBM Developer 傳訊 社群](#)，適用於為 IBM MessageSight 和 IBM WebSphere MQ 撰寫應用程式的 MQTT 開發人員。

[MQTT.org](#) 是瞭解及討論 MQTT 通訊協定的實作和延伸的好地方。

MQTT 是 [Eclipse Technology Project](#) 下的開放程式碼 Eclipse 專案。Paho 社群正在開發開放程式碼用戶端及伺服器。請參閱 [Eclipse Paho](#)。

開始使用 MQTT 用戶端

您可以透過建置並執行使用 MQTT 用戶端程式庫的範例 MQTT 用戶端應用程式，開始開發行動式或 機器對機器 (M2M) 應用程式。範例應用程式及相關聯的用戶端程式庫可在 [Mobile Messaging and M2M 用戶端套件](#) 中從 IBM 取得。有一些版本的應用程式及用戶端程式庫以 Java、JavaScript 及 C 撰寫。您可以在大部分平台及裝置上執行這些應用程式，包括 Apple 中的 Android 裝置及產品。

開始之前

若要建置及執行應用程式，您需要具有針對目標裝置或平台以及所用程式設計語言來建置應用程式的一些經驗。通常，只需一點經驗即可在您選擇的裝置或平台上建立及執行範例應用程式。

如果您使用企業級 MQTT 伺服器（例如 IBM WebSphere MQ 或 IBM MessageSight），則可從範例應用程式與現有企業應用程式交換資訊。

關於這項作業

您的目標如下：

1. 選擇您可以連接用戶端應用程式的 [MQTT 伺服器](#)。
2. 下載 [Mobile Messaging and M2M 用戶端套件](#)。
3. 針對目標裝置或平台，建置用戶端程式集中的範例應用程式。
4. 將範例連接至 MQTT 伺服器，以驗證範例可如預期般運作。

由於要針對您的裝置或平台建置及測試範例應用程式，因此您會建立一個工作開發環境，該環境可供您隨後用來建置自己的用戶端應用程式。

Mobile Messaging and M2M 用戶端套件 包含 MQTT SDK。此 SDK 為您提供下列資源：

- 以 Java、JavaScript 及 C 撰寫的範例 MQTT 用戶端應用程式。
- 支援這些用戶端應用程式並可讓其在大部分平台及裝置上執行的 MQTT 用戶端程式庫。

此 SDK 還包含適用於 C 的 MQTT 用戶端的原始碼。您可以改寫此原始碼，以針對其他平台建置適用於 C 的 MQTT 用戶端程式庫。如需執行此動作的說明，請參閱第 28 頁的『[建置適用於 C 的 MQTT 用戶端程式庫](#)』。適用於 C 的 MQTT 用戶端的原始碼，亦可透過 [Eclipse Paho](#) 的開放程式碼授權取得。

程序

下列文章將指引您完成專屬於平台的步驟，以在桌上型電腦或適用於 Android 或來自於 Apple 的行動式裝置上，建置及執行範例 MQTT 應用程式。

- [第 11 頁的『開始使用適用於 Java 的 MQTT 用戶端』](#)
- [第 16 頁的『在 Android 上開始使用 適用於 Java 的 MQTT 用戶端』](#)
- **V7.5.0.1**
- [第 21 頁的『適用於 JavaScript 的 MQTT 傳訊用戶端入門』](#)
- [第 23 頁的『開始使用適用於 C 的 MQTT 用戶端』](#)
- [第 43 頁的『在 iOS 上開始使用適用於 C 的 MQTT 用戶端』](#)

下一步

若要開發新的 MQTT 應用程式，您必須具有或獲得下列技能：

- 以裝置或平台所需的語言進程式設計。
- 為目標裝置或平台進程式設計。
- 設計發佈/訂閱應用程式。
- 為 MQTT 程式設計模型設計程式。
- 設計要在您選擇的行動式裝置上執行的程式。
- 使用 SSL 及 JAAS 來保護程式的安全。

您無需任何網路程式設計技術，即可將 MQTT 用戶端與其他裝置或應用程式連接，因為 MQTT 是傳訊及佇列系統。MQTT 用戶端程式庫管理應用程式的網路連線。

若要將 MQTT 用戶端與現有的企業應用程式整合，您具有兩個選擇。您可以與 (例如) IBM WebSphere MQ 或 JMS 應用程式共用 MQTT 發佈/訂閱主題，也可以撰寫您自己的整合配接器作為另一個 MQTT 用戶端。

現在參考的資訊來源如下：

- [開發 WebSphere MQ Telemetry 的應用程式](#)
- [MQTT.org](#)
- [Eclipse Paho](#)

相關概念

[第 122 頁的『開始使用 MQTT 伺服器』](#)

開始使用適用於 Java 的 MQTT 用戶端

使用 IBM MessageSight 或 IBM WebSphere MQ 作為 MQTT 伺服器，建立並執行 Java 範例應用程式的 MQTT 用戶端。範例應用程式使用來自 IBM 的 MQTT 軟體開發工具箱 (SDK) 的用戶端程式庫。SampleAsyncCallback 範例應用程式是一個模型，用於撰寫適用於 Android 及其他事件驅動作業系統的 MQTT 應用程式。

開始之前

- 您可以在任何具有 JSE 1.5 或更高版本 (即 "Java 相容") 的平台上執行適用於 Java 的 MQTT 用戶端應用程式。請參閱 [IBM Mobile Messaging and M2M 用戶端套件的系統需求](#)。
- 如果用戶端與伺服器之間有防火牆，請確認它不會封鎖 MQTT 資料流量。

關於這項作業

此作業的目的是檢查您是否可以建置並執行 Java 範例應用程式的 MQTT 用戶端，將它連接至 IBM WebSphere MQ 或 IBM MessageSight 作為 MQTT version 3 伺服器，並交換訊息。

遵循此作業以從 Eclipse 工作台或指令行，執行範例應用程式。範例中的步驟適用於 Windows。經過稍微修改，您即可在支援 JSE 1.5 或更高版本的任何平台上執行範例應用程式。

您可以在與 IBM WebSphere MQ 相同的伺服器上執行應用程式，其中已為您配置用於執行連接至 IBM WebSphere MQ 的應用程式的環境。遵循作業第 126 頁的『從指令行配置 MQTT 服務』，以在 Windows 或 Linux 上使用 IBM WebSphere MQ Telemetry 選項來安裝及配置 IBM WebSphere MQ。安裝並配置環境後，執行範例應用程式 MQTTV3Sample 來驗證安裝。

程序

1. 選擇您可以連接用戶端應用程式的 MQTT 伺服器。

伺服器必須支援 MQTT version 3.1 通訊協定。IBM 中的所有 MQTT 伺服器都會這麼做，包括 IBM WebSphere MQ 和 IBM MessageSight。請參閱第 122 頁的『開始使用 MQTT 伺服器』。

2. 選擇性的: 配置 MQTT 伺服器。

- 在 IBM WebSphere MQ 上，您必須完成下列任一作業，以設定佇列管理程式及配置其遙測 (MQXR) 服務：
 - 第 126 頁的『從指令行配置 MQTT 服務』
 - 第 128 頁的『使用 IBM WebSphere MQ Explorer 配置 MQTT 服務』
- 在其他伺服器上，查閱伺服器文件。Really Small Message Broker 不需要執行任何配置步驟。請參閱 Really Small Message Broker。

3. 下載 Mobile Messaging and M2M 用戶端套件 並安裝 MQTT SDK。

沒有安裝程式，您只是展開了下載的檔案。

- a. 下載 [Mobile Messaging and M2M 用戶端套件](#)。
- b. 建立您要安裝 SDK 的資料夾。

您可能想將資料夾命名為 MQTT。在這裡，此資料夾的路徑稱為 *sdkroot*。

- c. 將壓縮的 Mobile Messaging and M2M 用戶端套件 檔案內容展開到 *sdkroot* 中。展開會建立以 *sdkroot*\SDK 開頭的目錄樹。

4. 安裝 Java 開發套件 (JDK) 第 6 版或更新版本。

因為您正在開發 Java 應用程式 for Android，所以 JDK 必須來自 Oracle。您可以從 [Java SE](#) 下載取得 JDK。

5. 編譯並執行一個以上 Java 範例應用程式的 MQTT 用戶端:

- [第 12 頁的『從指令行編譯並執行 Paho 程式範例』](#)
- [第 14 頁的『從 Eclipse 編譯及執行所有 MQTT 用戶端範例 Java 應用程式』](#)
- [第 16 頁的『在 Android 上開始使用適用於 Java 的 MQTT 用戶端』](#)

SDK 中包含了下列 MQTT 用戶端範例 Java 應用程式：

MQTTV3Sample

此範例亦隨附於 IBM WebSphere MQ，且鏈結至 `com.ibm.micro.client.mqttv3.jar` 套件。

Sample

Sample 位於 Paho 套件中，並鏈結至 `org.eclipse.paho.client.mqttv3` 套件。它類似於 MQTTV3Sample；會等到每一個 MQTT 動作都完成。

SampleAsyncWait

SampleAsyncWait 在 `org.eclipse.paho.client.mqttv3` 套件中。它使用非同步 MQTT API；會等待不同的執行緒，直到動作完成。主要執行緒可以執行其他作業，直到同步處理正在等待 MQTT 動作完成的執行緒。

SampleAsyncCallback

SampleAsyncCallback 在 `org.eclipse.paho.client.mqttv3` 套件中。它會呼叫非同步 MQTT API。非同步 API 不會等待 MQTT 完成處理呼叫；它會回到應用程式。應用程式會執行其他作業，然後等待下一個事件送達，以進行處理。MQTT 在完成處理時，會將事件通知公佈回應用程式。事件驅動 MQTT 介面適用於 Android 和其他事件驅動作業系統的服務和活動程式設計模型。

例如，查看 `mqttExerciser` 範例如何使用服務和活動程式設計模型將 MQTT 整合至 Android。

mqttExerciser

`mqttExerciser` 是 Android 的範例程式。由於其建置及執行方式不同，因此將分別說明。請參閱第 16 頁的『在 Android 上開始使用適用於 Java 的 MQTT 用戶端』。

結果

您已編譯並執行已連接至 [IBM WebSphere MQ](#) 或作為 MQTT 伺服器的 IBM MessageSight 的 MQTT Java 應用程式範例。

下一步

研究 Javadoc 參照資訊；請參閱第 14 頁的『從 Eclipse 編譯及執行所有 MQTT 用戶端範例 Java 應用程式』的步驟第 14 頁的『3』。或者，開啟 Mobile Messaging and M2M 用戶端套件的 `SDK\clients\java\doc\javadoc` 目錄中的 Javadoc html 檔案。

從指令行編譯並執行 Paho 程式範例

從指令行編譯並執行 Paho 範例應用程式 `Sample.java`。範例在 MQTT SDK 中。此範例是使用 `org.eclipse.paho.client.mqttv3` 套件中的 MQTT Paho 用戶端程式庫所建置。它示範了 MQTT 發佈者及訂閱者。您可以使用相同的方式建置及執行相同目錄中的其他兩個 Paho 範例。它們會因非同步地呼叫 MQTT 程式庫而異。

開始之前

執行主要作業中的步驟第 11 頁的『1』至第 11 頁的『4』，以配置 MQTT 伺服器及下載 Mobile Messaging and M2M 用戶端套件。

關於這項作業

從 SDK 用戶端範例子目錄 `SDK\clients\java\samples` 編譯並執行 `Sample.java`。Java 代碼在目錄 `SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app` 中。

程序

1. 在用戶端範例子目錄中建立 Script，以在您選擇的平台上編譯並執行 `Sample`。

下列 Script 會在 Windows 上編譯並執行範例。

```
@echo on
setlocal
set classpath=
set JAVADIR=C:\Program Files\IBM\Java70\bin
"%JAVADIR%\javac"
-cp ..\org.eclipse.paho.client.mqttv3.jar ..\org\ eclipse\paho\sample\mqttv3app\Sample.java
start "Sample Subscriber" "%JAVADIR%\java" -cp .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -a subscribe -b localhost -p 1883
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
"%JAVADIR%\java" -cp .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -b localhost -p 1883
pause
endlocal
```

圖 5: 編譯並執行 *Sample.java*

2. 執行 Script。

結果:

```
Connected to tcp://localhost:1883 with client ID SampleJavaV3_subscribe
Subscribing to topic "Sample/#" qos 2
Press <Enter> to exit
Time: 2012-11-09 17:23:22.718 Topic: Sample/Java/v3 Message: Message
from blocking MQTTv3 Java client sample QoS: 2
```

圖 6: *MQTTV3Sample* 訂閱者

```
Connecting to tcp://localhost:1883 with client ID SampleJavaV3_publish
Connected
Publishing at: 2012-11-09 17:22:07.734 to topic "Sample/Java/v3" qos 2
Disconnected
```

圖 7: *MQTTV3Sample* 發佈者

如果您保持訂閱者應用程式執行，則無法再次執行訂閱者。新訂閱者的用戶端 ID 與舊訂閱者相同。您無法同時使用相同的用戶端 ID 執行兩個 MQTT 用戶端。您可以設定 `-i` 選項來設定用戶端 ID，以便您可以同時執行不同的訂閱者。

如果您再次執行相同的用戶端，您可以利用 `-c` 選項來啟動，及在 `cleansession` 設為 `false` 時啟動用戶端。使用該選項，您可以探索岔斷的用戶端階段作業的行為。

3. 按 Enter 鍵或關閉視窗來結束訂閱者。

下一步

建立 Script 以編譯並執行 `SDK\clients\java\samples\org\ eclipse\paho\sample\mqttv3app` 子目錄中的其他範例。複製第 13 頁的圖 5 中的 Script，並使用 `SampleAsyncWait` 或 `SampleAsyncCallback` 取代 `Sample`。其他範例是同步 `Sample` 程式的非同步版本。

SampleAsyncWait

`SampleAsyncWait` 在 `org.eclipse.paho.client.mqttv3` 套件中。它使用非同步 MQTT API；會等待不同的執行緒，直到動作完成。主要執行緒可以執行其他作業，直到同步處理正在等待 MQTT 動作完成的執行緒。

SampleAsyncCallback

`SampleAsyncCallback` 在 `org.eclipse.paho.client.mqttv3` 套件中。它會呼叫非同步 MQTT API。非同步 API 不會等待 MQTT 完成處理呼叫；它會回到應用程式。應用程式會執行其他作業，然後等待下一個事件送達，以進行處理。MQTT 在完成處理時，會將事件通知公佈回應應用程式。事件驅動 MQTT 介面適用於 Android 和其他事件驅動作業系統的服務和活動程式設計模型。

例如，查看 `mqttExerciser` 範例如何使用服務和活動程式設計模型將 MQTT 整合至 Android。

非同步範例示範如何減少 MQTT 應用程式在等待 MQTT 用戶端時封鎖的時間量。務必刪除主要執行緒中的封鎖呼叫，以增強行動式環境中的回應力及電池壽命。

範例示範在 MQTT 用戶端中呼叫非同步介面的兩種型樣。

1. 當 MQTT 正在等待網路互動時，`SampleAsyncWait` 不會封鎖。

2. `SampleAsyncCallback` 不會封鎖，會等待 MQTT 用戶端完成任何動作。JavaScript 頁面透過瀏覽器呼叫 MQTT 用戶端時，需要後者。JavaScript 頁面不得封鎖。動作回應必須公佈回主要瀏覽器執行緒，從而呼叫您為處理通知撰寫的 MQTT 事件處理程式。

從 Eclipse 編譯及執行所有 MQTT 用戶端範例 Java 應用程式

編譯並執行 Mobile Messaging and M2M 用戶端套件中的 MQTT 用戶端範例 Java 應用程式。它們示範了 MQTT 發佈者及訂閱者。

關於這項作業

編譯並執行 Eclipse 中的 MQTT Java 範例 `Sample` 和 `MQTTV3Sample`。`Sample` 在 `sdkroot\SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app` SDK 用戶端子目錄中，`MQTTV3Sample.java` 在 `sdkroot\SDK\clients\java\samples` 中。

程序

1. 下載 Eclipse IDE for Java Developers。
2. 在 Eclipse 中建立稱為 MQTT Samples 的 Java 專案。
 - a) 檔案 > 新建 > Java 專案 和類型 MQTT Samples。按下一步。

請檢查 JRE 是否為正確或更新版本。JSE 必須是 1.5 版或更新版本。
 - b) 在「Java 設定」視窗中，按一下鏈結其他來源資料夾。
 - c) 瀏覽至在其中安裝 MQTT Java SDK 資料夾的目錄。選取 `sdkroot\SDK\clients\java\samples` 資料夾，然後按 確定 > 下一步 > 完成。
 - d) 在「Java 設定」視窗中，按一下程式庫 > 新增外部 JAR。
 - e) 瀏覽至在其中安裝 MQTT Java SDK 資料夾的目錄。找出 `sdkroot\SDK\clients\java` 資料夾，並選取 `org.eclipse.paho.client.mqttv3.jar` 和 `com.ibm.micro.client.mqttv3.jar` 檔案；按一下 開啟 > 完成。

`MQTTV3Sample.java` 範例會鏈結至 `com.ibm.micro.client.mqttv3.jar`，`paho` 目錄樹中的範例會鏈結至 `org.eclipse.paho.client.mqttv3.jar`。`com.ibm.micro.client.mqttv3.jar` 會保留下來，讓現有的 MQTT 應用程式繼續建置並執行，而不會有任何變更。

MQTT Samples 專案即已建置，會出現一些警告，但無錯誤。

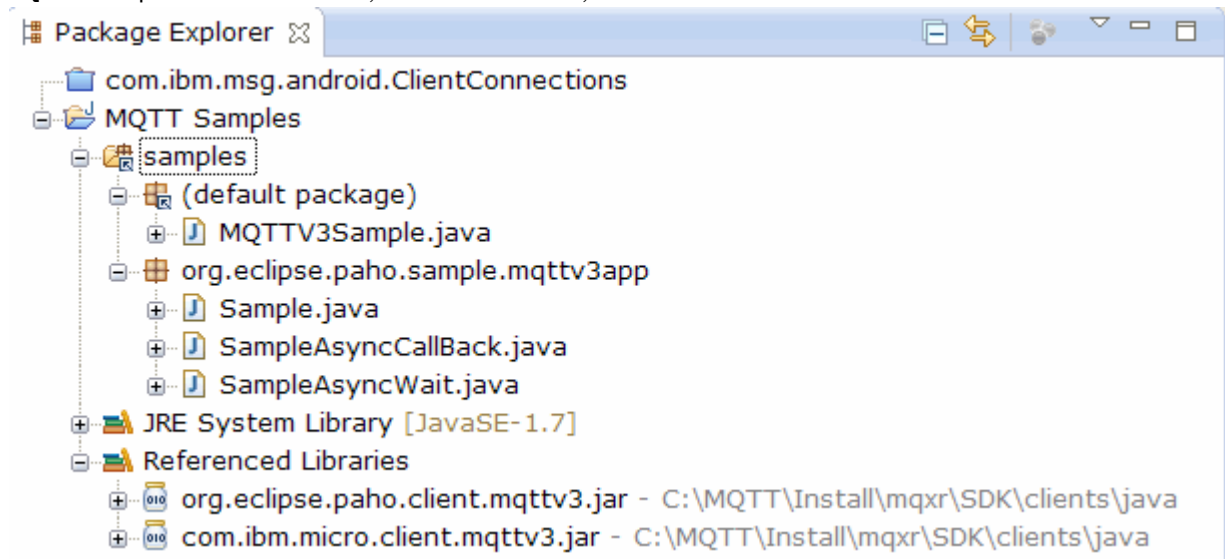


圖 8: 適用於 Java 的 MQTT 用戶端專案

3. 選擇性的: 安裝 MQTT 用戶端 Javadoc。

安裝 MQTT 用戶端 Javadoc 之後，Java 編輯器會說明浮動說明中的 MQTT 類別。

- a) 在 Java 專案中開啟 **套件瀏覽器 > 參照的程式庫**。用滑鼠右鍵按一下 `org.eclipse.paho.client.mqttv3.jar` > **內容**。
 - b) 在「內容」導覽器中，按一下 **Javadoc 位置**。
 - c) 在「**Javadoc 位置**」頁面中按一下 **Javadoc URL > 瀏覽**，並尋找 `SDK\clients\java\doc\javadoc` 資料夾 > **確定**。
 - d) 按一下 **驗證 > 確定**。
系統會提示您開啟瀏覽器以檢視文件。
 - e) 對 `com.ibm.micro.client.mqttv3.jar` 檔案，重複此程序。
4. 建立發佈者及訂閱者執行時期配置，以執行 `mqttv3app.Sample` 應用程式。
- a) 用滑鼠右鍵按一下 **Sample** 類別，然後按一下 **執行身分 > 執行配置**。
 - b) 用滑鼠右鍵按一下 **Java 應用程式 > 新建**，然後輸入名稱 `SampleSubscriber`。
 - c) 按一下引數標籤，然後鍵入程式引數，接著按一下 **套用**。

```
-a subscribe -b localhost -p 1883
```

- d) 重複最後一個步驟以建立 `SamplePublisher` 配置，並省略 `-a subscribe` 參數。
5. 執行 `mqttv3app.Sample` 訂閱者，後接發佈者。
- a) 按一下 **執行 > 執行配置**。
 - b) 按一下 **SampleSubscriber > 執行**。

開啟「主控台」視圖。訂閱者會等待發佈。

```
Connected to tcp://localhost:1883 with client ID SampleJavaV3_subscribe
Subscribing to topic "Sample/#" qos 2
Press <Enter> to exit
```

- c) 按一下 **SamplePublisher > 執行**。
開啟「主控台」視圖。您會看到發佈者建立的發佈：

```
Connecting to tcp://localhost:1883 with client ID SampleJavaV3_publish
Connected
Publishing at: 2012-11-09 14:09:29.859 to topic "Sample/Java/v3" qos 2
Disconnected
```

- d) 將主控台視圖切換至訂閱者主控台。

切換主控台圖示為 。

訂閱者已接收發佈：

```
...
Time: 2012-11-09 14:09:30.593 Topic: Sample/Java/v3 Message: Message from blocking
MQTTV3 Java client sample QoS: 2
```

6. 選擇性的: 為 `MQTTV3Sample` 建立發佈者及訂閱者執行時期配置。
- a) 用滑鼠右鍵按一下 **MQTTV3Sample** 類別，然後按一下 **執行身分 > 執行配置**。
 - b) 用滑鼠右鍵按一下 **Java 應用程式 > 新建**，然後輸入名稱 `MQTTV3SampleSubscriber`。
 - c) 按一下引數標籤，然後鍵入程式引數，接著按一下 **套用**。

```
-a subscribe -b localhost -p 1883
```

- d) 重複最後一個步驟以建立 `MQTTV3SamplePublisher` 配置，並省略 `-a subscribe` 參數。
7. 選擇性的: 執行 `MQTTV3Sample` 訂閱者，後接發佈者。
- a) 按一下 **執行 > 執行配置**。
 - b) 按一下 **MQTTV3SampleSubscriber > 執行**。

開啟「主控台」視圖。訂閱者會等待發佈。


```
Connected to tcp://localhost:1883
Subscribing to topic "MQTTV3Sample/#" qos 2
Press <Enter> to exit
```

c) 按一下 **MQTTV3SamplePublisher** > 執行。

開啟「主控台」視圖。您可以看到發佈者建立的發佈。

```
Connected to tcp://localhost:1883
Publishing to topic "MQTTV3Sample/Java/v3" qos 2
Disconnected
```

d) 將主控台視圖切換至訂閱者主控台。

切換主控台圖示為 。

訂閱者已接收發佈：

```
MQTTV3Sample/Java/v3
Message: Message from MQTTv3 Java client
QoS: 2
```

在 Android 上開始使用 適用於 Java 的 MQTT 用戶端

您可以安裝 Android 的 MQTT 用戶端範例 Java 應用程式，用來與 MQTT 伺服器交換訊息。應用程式使用來自 IBM 的 MQTT SDK 的用戶端程式庫。您可以自己建置應用程式，或下載預先建置的應用程式範例。

開始之前

- 如需瞭解受支援及參照 MQTT 用戶端平台，請參閱 [IBM Mobile Messaging and M2M 用戶端套件的系統需求](#)。
- 如果用戶端與伺服器之間有防火牆，請確認它不會封鎖 MQTT 資料流量。
- MQTT 用戶端應用程式範例適用於 Ice Cream Sandwich (Android 4.0) 及更新版本。此版本的 Android 在平板電腦上也能提供更清晰的顯示解析度。

關於這項作業

Android 的 MQTT 用戶端範例 Java 應用程式稱為 "mqttExerciser"。此應用程式使用來自 MQTT SDK 的用戶端程式庫，並與 MQTT 伺服器交換訊息。

您可以自行建置範例應用程式，然後將它從 Eclipse 匯出為 mqttExerciser.apk，或使用 Mobile Messaging and M2M 用戶端套件的 `sdkroot\SDK\clients\android\samples\apks` 資料夾中作為檔案 mqttExerciser.apk 提供的預先建置範例應用程式。如果您選擇自行建置應用程式，您建置的開發環境會自訂為將行動式傳訊併入 for Android 應用程式。如此應可協助您開始在自己的應用程式中包含行動式傳訊。

程序

1. 選擇您可以連接用戶端應用程式的 MQTT 伺服器。

伺服器必須支援 MQTT version 3.1 通訊協定。IBM 中的所有 MQTT 伺服器都會這麼做，包括 IBM WebSphere MQ 和 IBM MessageSight。請參閱 [第 122 頁的『開始使用 MQTT 伺服器』](#)。

2. 取得正確的工具。

安裝 Java 開發套件 (JDK) 第 6 版或更新版本。因為您正在開發 Java 應用程式 for Android，所以 JDK 必須來自 Oracle。您可以從 [Java SE](#) 下載取得 JDK。

您也需要 Eclipse 開發環境。其必須是 Eclipse 3.6.2 (Helios) 或更新版本。Eclipse 至少必須要有 6 以上的 Java 編譯器層次，才能與您的 JDK 相符。您可以從 [Eclipse Foundation](#) 取得這一切。

最後，您需要 Android SDK。您可以從 [Get the Android SDK](#) 取得此工具。

3. 下載 Mobile Messaging and M2M 用戶端套件 並安裝 MQTT SDK。

沒有安裝程式，您只是展開了下載的檔案。

- a. 下載 [Mobile Messaging and M2M 用戶端套件](#)。
- b. 建立您要安裝 SDK 的資料夾。

您可能想將資料夾命名為 MQTT。在這裡，此資料夾的路徑稱為 *sdkroot*。

- c. 將壓縮的 Mobile Messaging and M2M 用戶端套件 檔案內容展開到 *sdkroot* 中。展開會建立以 *sdkroot*\SDK 開頭的目錄樹。
- ### 4. 選擇性的: 建置 for Android 的 mqttExerciser 應用程式範例。

配置 Eclipse 和 Android 工具，並從 MQTT SDK 匯入並建置 mqttExerciser 專案。

註: 如果您現在不想這麼做，可以使用預先建置的應用程式範例（MQTT SDK 的 *sdkroot*\SDK\clients\android\samples\apks 資料夾中之檔案 mqttExerciser.apk）。

- a) 從 JDK 使用 JRE 啟動 Eclipse 開發環境。

```
eclipse -vm "JRE path"
```

- b) 從 Android SDK 選取並安裝一組套件與平台。

請參閱 [新增平台及套件](#)，以瞭解 Google 建議的平台及套件清單。

註: SDK 平台必須是 Android API 層次 16 或更新層次。使用較早的 API 層次，無法順利地編譯專案。

- c) 將 [Android 開發工具 \(ADT\) 外掛程式](#) 新增至 Eclipse。
- d) 將 mqttExerciser 應用程式專案範例匯入 Eclipse，然後修正錯誤。
 - i) 在路徑 *sdkroot*\SDK\clients\android\samples\mqttExerciser 中，從 MQTT SDK 匯入範例應用程式專案。

「問題」視圖會列出許多建置錯誤。在接下來的幾個步驟中，將會解決建置錯誤。
 - ii) 將 `org.eclipse.paho.client.mqttv3.jar` 程式庫複製到 Android 專案中的 **libs** 資料夾。

Windows 例如，在 Windows 上，這是在 *sdkroot*\SDK\clients\java 資料夾下。這時會顯示「檔案作業」視窗。接受複製檔案選項，然後按一下**確定**。
 - iii) 用滑鼠右鍵按一下專案資料夾 `com.ibm.msg.android`；按一下 **Android 工具 ... > 新增支援程式庫 ...**。閱讀並接受授權條款，然後按一下**安裝**。
 - iv) 用滑鼠右鍵按一下專案資料夾 `com.ibm.msg.android`；按一下 **Android 工具 ... > 修正專案內容**。
 - v) 如果工作區仍有大約 84 個錯誤（參閱置換超類別方法），則編譯器相符性層次可能設為 1.5 或更低。Android SDK 第 16 版預期編譯器相符性層次不會大於 1.5。若要修正剩下的錯誤，請完成下列步驟：
 - a) 檢查您的 Android SDK 與對應的 Eclipse 外掛程式，並升級（必要的話）至 Android SDK 第 17 版。
 - b) 用滑鼠右鍵按一下 **com.ibm.msg.android** 專案資料夾，然後選取 **內容 > Java 編譯器**。檢查編譯器相符性層次，將其設為至少 1.6，然後重建工作區。

專案即已建置，會出現一些警告，但無錯誤。

5. 在 Android 裝置上安裝並啟動 MQTT 用戶端範例 Java 應用程式。

請參閱 [developer.android.com](#) 頁面 [Running your app](#)。

如果您自行將應用程式建置為 Eclipse 專案，可以從 Eclipse 啟動該應用程式。

如果您有應用程式套件 (APK) 檔案 mqttExerciser.apk，可以使用 [Android Debug Bridge \(ADB\)](#) 安裝指令，將其安裝在 Eclipse 之外。此指令會將 APK 檔的位置當作引數。如果您使用預先建置的範例應用程式，則位置為 *sdkroot*\SDK\clients\android\samples\apks\mqttExerciser.apk。

6. 使用 for Android 的 mqttExerciser 應用程式範例來連接、訂閱及發佈至主題。

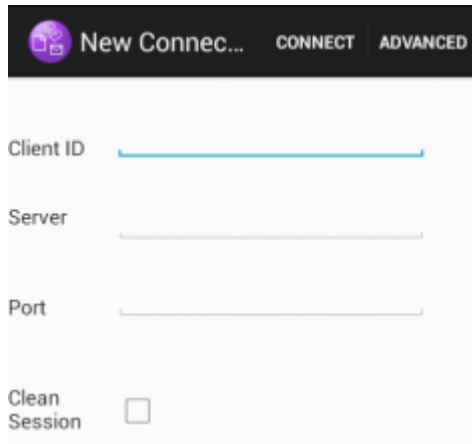
a) 開啟 Android 的 MQTT 用戶端範例 Java 應用程式。

此視窗會在您的 Android 裝置中開啟：



b) 連接至 MQTT 伺服器。

i) 按一下 **+** 號以開啟新的 MQTT 連線。



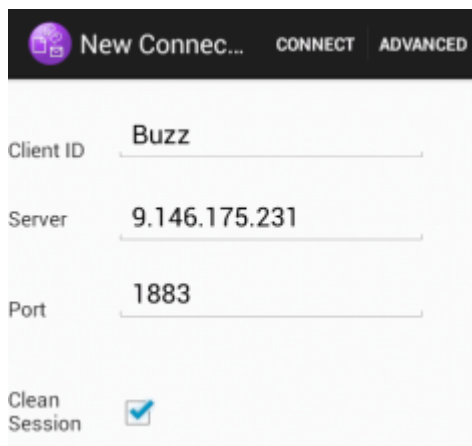
ii) 在**用戶端 ID** 欄位中輸入任何唯一的 ID。請耐心等待，按鍵速度可能緩慢。

iii) 在**伺服器**欄位輸入您 MQTT 伺服器的 IP 位址。

此為您在第一個主要步驟中選擇的伺服器。IP 位址不得為 127.0.0.1

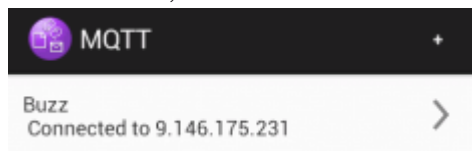
iv) 輸入 MQTT 連線的埠號。

一般 MQTT 連線的預設埠號是 1883。



v) 按一下**連接**。

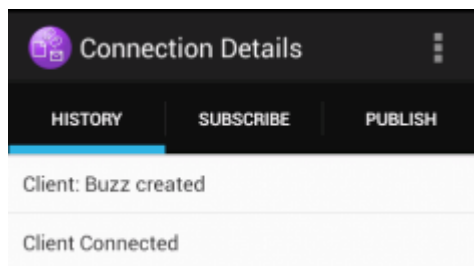
如果連線成功，您會看到 "正在連接" 訊息，後面接著這個視窗：



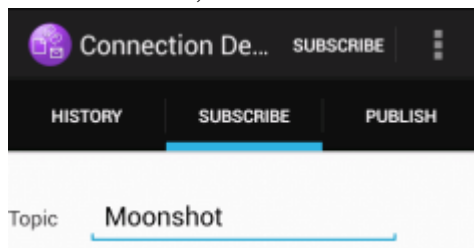
c) 訂閱主題。

i) 按一下**已連接**訊息。

「**連線詳細資料**」視窗會開啟，且會列出歷程：



ii) 按一下訂閱標籤，並輸入主題字串。

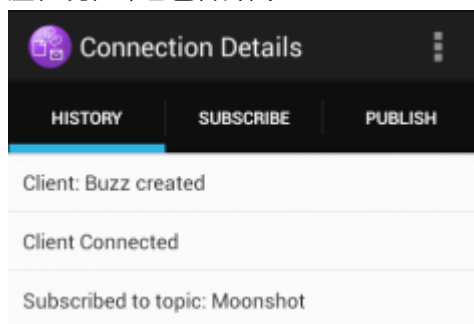


iii) 按一下訂閱動作。

隨即會出現一小段時間的"已訂閱"訊息。

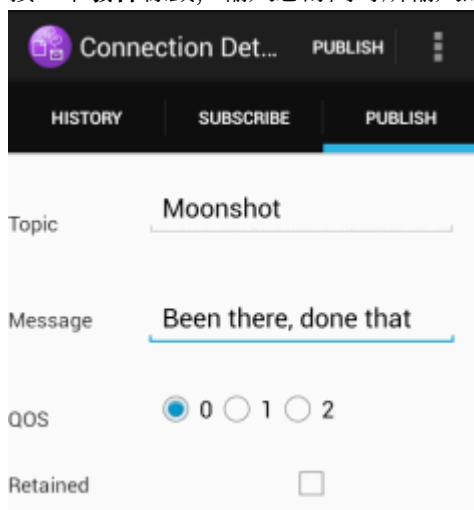
iv) 按一下歷程標籤。

歷程現在即已包含訂閱：



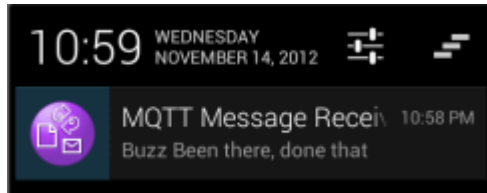
d) 現在發佈至相同的主题。

i) 按一下發佈標籤，輸入您訂閱時所輸入的相同主题字串。輸入訊息。

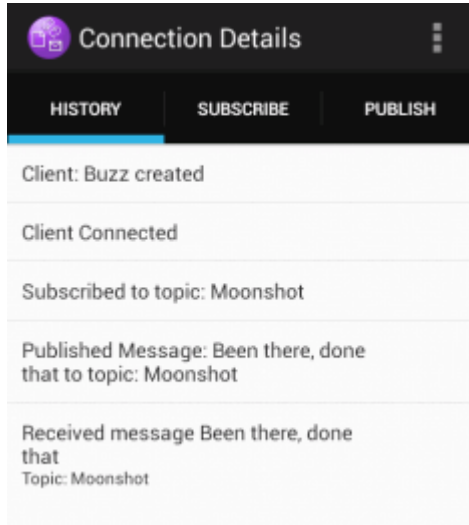


ii) 按一下發佈動作。


隨即會顯示一小段時間的兩個訊息，先是"已發佈"，然後是"已訂閱"。發佈會顯示在狀態區中（將分隔線向下滑即可開啟狀態視窗）。



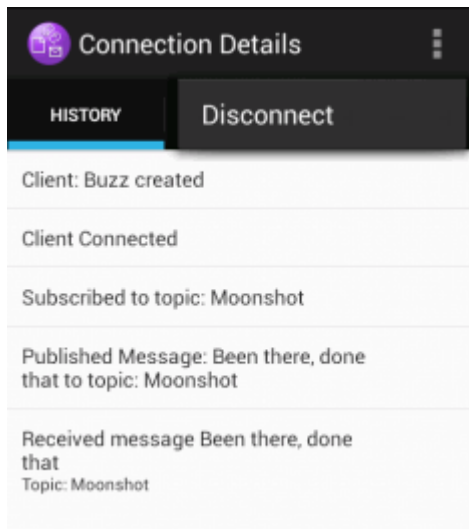
iii) 按一下**歷程**標籤以檢視完整歷程。



e) 中斷用戶端實例的連線。

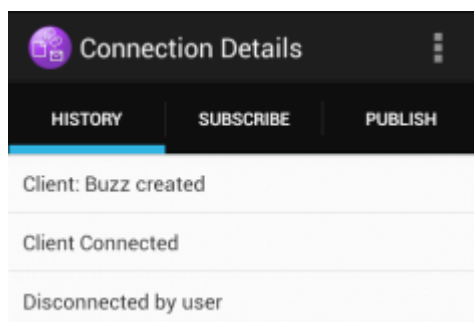
i) 按一下動作列中的功能表圖示。 

Android 的 MQTT 用戶端範例 Java 應用程式會將「**中斷連線**」按鈕新增至 MQTT「**連線詳細資料**」視窗。

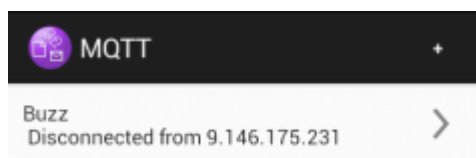


ii) 按一下**中斷連線**。

已連接狀態會變更為已中斷連線：



f) 按上一步回到 MQTT 用戶端範例 Java 應用程式 階段作業的清單。



- 按一下加號以啟動新的 MQTT 用戶端範例 Java 應用程式階段作業。
- 按一下中斷連線的用戶端以重新連接。
- 按上一步回到啟動程式。

g) 按一下作業按鈕以列出正在執行的應用程式。尋找 MQTT 用戶端範例 Java 應用程式。滑動圖示使其脫離畫面，以關閉它。

下一步

如果您自行建置應用程式範例，即已備妥可開始開發您自己的 Android 應用程式，呼叫 MQTT 程式庫來交換訊息。您可以在 `mqttExerciser` 中的類別上建立 Android 應用程式的模型。若要研究範例，請針對 `mqttExerciser` 專案中 `com.ibm.msg.android` 和 `com.ibm.msg.android.service` 中的類別產生 Javadoc。

相關資訊

[使用 ADT 管理 Eclipse 的專案](#)

適用於 JavaScript 的 MQTT 傳訊用戶端入門

透過顯示傳訊用戶端範例首頁並瀏覽它所鏈結的資源，即可開始使用適用於 JavaScript 的 MQTT 傳訊用戶端。若要顯示此首頁，您可以配置 MQTT 伺服器以接受來自 MQTT 傳訊用戶端範例 JavaScript 頁面的連線，然後在 Web 瀏覽器中鍵入您在伺服器上配置的 URL。適用於 JavaScript 的 MQTT 傳訊用戶端會在您的裝置上自動啟動，並且會顯示傳訊用戶端範例首頁。此頁面包含公用程式、程式設計介面文件、指導教學及其他有用資訊的鏈結。

開始之前

對於進階用途或正式作業用途，您將要重新調整或移除傳訊用戶端範例首頁。請注意，我們並不保證從範例程式碼產生的使用者介面，符合所有可存取性標準或可存取性需求。

您需要 MQTT 伺服器，才能支援適用於 JavaScript 的 MQTT 傳訊用戶端。此伺服器必須透過 WebSockets 支援 MQTT V3.1 通訊協定。IBM MessageSight 及 IBM WebSphere MQ Version 7.5.0, Fix Pack 1 以及更新版本，支援 WebSockets 上的 MQTT protocol。請參閱第 122 頁的『開始使用 MQTT 伺服器』。若要安裝 IBM WebSphere MQ 以進行 90 天免費評估，請參閱第 124 頁的『安裝 IBM WebSphere MQ』。

WebSocket protocol 是最近建立的。如果用戶端與伺服器之間存在防火牆，請檢查它未封鎖 WebSockets 資料流量。同樣地，如果您的瀏覽器還不支援 WebSocket protocol¹ 您將無法使用用戶端公用程式或傳訊用戶端範例首頁中提供的指導教學。第 22 頁的表 1 列出了一些瀏覽器，其最新版本經測試證明可與傳訊用戶端配合運作。

¹ 具體而言，如果它不支援 RFC 6455 (WebSocket) 標準。

表 1: 可與適用於 JavaScript 的 MQTT 傳訊用戶端搭配使用的受支援瀏覽器

Android	iOS	Linux	Windows
for Android 的 Firefox 19.0 以及更新版本 for Android 的 Chrome 25.0 以及更新版本	Safari 6.0 以及更新版本 Chrome 14.0 以及更新版本	Firefox 6.0 以及更新版本 Chrome 14.0 以及更新版本	Firefox 6.0 以及更新版本 Chrome 14.0 以及更新版本

關於這項作業

此作業中的大部分步驟將配置 MQTT 伺服器。存取適用於 JavaScript 的傳訊用戶端所需的一切，便是執行支援 WebSocket protocol 的瀏覽器。

在 IBM WebSphere MQ 上，遵循相關步驟以透過建立通道範例來啟用 IBM WebSphere MQ Telemetry。連接至埠 1883 上的範例預設 MQTT WebSockets 通道。傳訊用戶端範例首頁 URL 是 IBM WebSphere MQ 上的 `http://hostname:1883`。

在 IBM MessageSight 上，安裝並設定應用裝置，配置傳訊中心以接受連線，並建立 MQTT WebSockets 端點。

程序

1. 下載 [Mobile Messaging and M2M 用戶端套件](#)，並選擇您可將用戶端應用程式與其連接的 MQTT 伺服器。
請參閱 第 9 頁的『開始使用 MQTT 用戶端』。
2. 配置 MQTT 伺服器，以接受來自適用於 JavaScript 的 MQTT 傳訊用戶端範例 HTML 頁面的連線。
 - 在 IBM WebSphere MQ 上：
 - 如果您已針對 MQTT 配置 IBM WebSphere MQ 佇列管理程式，請變更通道定義中的通訊協定，以同時支援 MQTT 和 HTTP。請參閱 [ALTER CHANNEL](#)。
 - 若要建立 IBM WebSphere MQ 佇列管理程式，並配置範例 MQTT WebSockets 端點，請完成下列一項作業：
 - 第 126 頁的『從指令行配置 MQTT 服務』
 - 第 128 頁的『使用 IBM WebSphere MQ Explorer 配置 MQTT 服務』
3. 在您的裝置上開啟 Web 瀏覽器。
4. 鍵入傳訊用戶端範例首頁的 URL。
 - 在 IBM WebSphere MQ 上，此為 `http://hostname:1883`
 - 在 IBM MessageSight 上，此為 `http://hostname:port`

其中，*hostname* 是乙太網路 Socket（您在 IBM MessageSight 應用裝置上配置為用戶端將要連接的端點）的 DNS 名稱或 IP 位址，*port* 是您為用戶端指派給該端點的 TCP/IP 埠號。

即會顯示傳訊用戶端範例首頁。

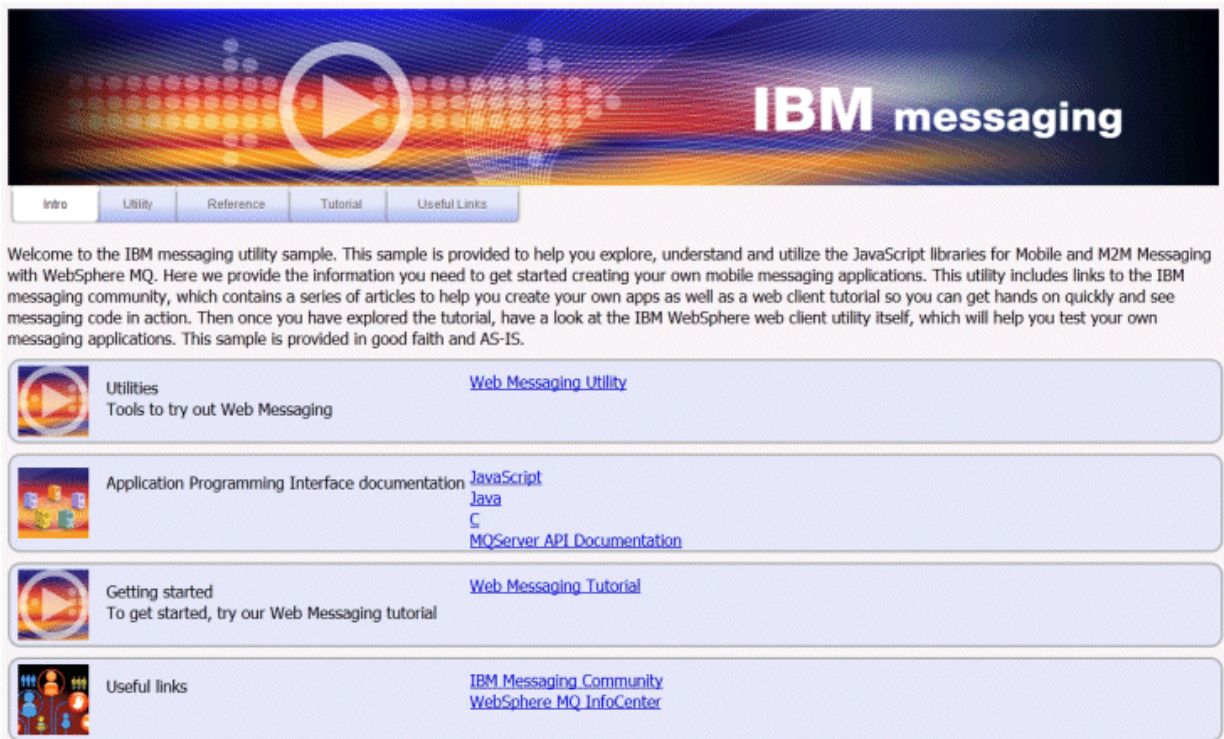


圖 9: 適用於 JavaScript 的 MQTT 傳訊用戶端範例首頁

結果

您已針對 WebSockets 配置 MQTT 通道。

在適用於 JavaScript 的 MQTT 傳訊用戶端的範例首頁中，按一下 **Web 傳訊公用程式** 試用傳訊用戶端 API 中的各種不同功能。例如，您可以連接至佇列管理程式、訂閱訊息，然後發佈部分訊息。您也可以按一下 **Web 傳訊指導教學**，以學習如何建立網頁來呼叫適用於 JavaScript 的 MQTT 傳訊用戶端 API。

相關概念

[第 106 頁的『適用於 JavaScript 的 MQTT 傳訊用戶端及 Web 應用程式』](#)

[第 109 頁的『如何在 JavaScript 中對傳訊應用程式進行程式設計』](#)

相關工作

[第 71 頁的『透過 SSL 及 WebSockets 連接 適用於 JavaScript 的 MQTT 傳訊用戶端』](#)

使用適用於 JavaScript 的 MQTT 傳訊用戶端範例 HTML 頁面搭配 SSL 及 WebSocket protocol，將 Web 應用程式安全地連接至 IBM WebSphere MQ。

開始使用適用於 C 的 MQTT 用戶端

您可以在其中編譯 C 原始檔的任何平台上，建立並執行適用於 C 的 MQTT 用戶端範例。驗證您可以使用 IBM MessageSight 或 IBM WebSphere MQ 作為 MQTT 伺服器來執行適用於 C 的範例 MQTT 用戶端。

開始之前

- 如果用戶端與伺服器之間有防火牆，請確認它不會封鎖 MQTT 資料流量。
- 如需瞭解適用於 C 的受支援及參照 MQTT 用戶端平台，請參閱 [IBM Mobile Messaging and M2M 用戶端套件的系統需求](#)。

關於這項作業

執行此作業可從指令行或從 Microsoft Visual Studio 2010，在 Windows 上編譯及執行適用於 C 的 MQTT 用戶端範例。Microsoft Visual Studio 2010 也用於在指令行範例中編譯用戶端。修改指令行 Script 以在其他平台上編譯並執行範例。

程序

1. 選擇您可以連接用戶端應用程式的 MQTT 伺服器。

伺服器必須支援 MQTT version 3.1 通訊協定。IBM 中的所有 MQTT 伺服器都會這麼做，包括 IBM WebSphere MQ 和 IBM MessageSight。請參閱 [第 122 頁的『開始使用 MQTT 伺服器』](#)。

2. 在您要建置的平台上安裝 C 開發環境。

本主題的範例中的 make 檔將下列工具設為目標：

-  若為 iOS，在具有 OS X 10.8.2 的 Apple Mac 上，使用 [Xcode](#) 中的 iOS 開發工具。
-  對於 Linux，為 Red Hat® Enterprise Linux 6.2 版中的 gcc 4.4.6 版。
glibc C 程式庫的最低支援層次為 2.12，Linux 核心的最低支援層次為 2.6.32。
-  對於 Microsoft Windows，為 Visual Studio 10.0 版。

3. 下載 Mobile Messaging and M2M 用戶端套件 並安裝 MQTT SDK。

沒有安裝程式，您只是展開了下載的檔案。

- a. 下載 [Mobile Messaging and M2M 用戶端套件](#)。
- b. 建立您要安裝 SDK 的資料夾。

您可能想將資料夾命名為 MQTT。在這裡，此資料夾的路徑稱為 *sdkroot*。

- c. 將壓縮的 Mobile Messaging and M2M 用戶端套件 檔案內容展開到 *sdkroot* 中。展開會建立以 *sdkroot*\SDK 開頭的目錄樹。

4. 選擇性的: 遵循 [第 28 頁的『建置適用於 C 的 MQTT 用戶端程式庫』](#) 中的步驟。

請僅在 Mobile Messaging and M2M 用戶端套件 不包含適用於目標平台的 C 用戶端程式庫時，才執行此步驟。

5. 編譯並執行 MQTT 用戶端範例 C 應用程式 MQTTV3Sample.c。

- 從指令行，遵循 [第 24 頁的『從指令行編譯及執行 MQTT 用戶端範例 C 應用程式』](#) 中的步驟進行。
- 從 IDE，遵循 [第 25 頁的『從 Microsoft Visual Studio 編譯及執行 MQTT 用戶端範例 C 應用程式』](#) 中的步驟進行。

從指令行編譯及執行 MQTT 用戶端範例 C 應用程式

從指令行編譯及執行 MQTT 用戶端範例 C 應用程式。範例在 MQTT SDK 中。它示範了 MQTT 發佈者及訂閱者。

開始之前

安裝 C 開發環境; 例如範例中使用的 Microsoft Visual Studio 2010。

關於這項作業

編譯並執行 SDK 用戶端子目錄 *sdkroot*\SDK\clients\c\samples 中的 C 範例 MQTTV3Sample。

程序

在用戶端範例目錄中建立 Script，以在您選擇的平台上編譯並執行 Sample。

下列 Script 會在 Windows 32 位元平台（已建置 Microsoft Visual Studio 2010）上編譯並執行範例。從 `sdkroot\SDK\clients\c\samples` 子目錄執行 Script。

```
@echo off
setlocal
call "C:\Program Files\Microsoft Visual Studio 10.0\VC\vcvarsall.bat" x86
cl /nologo /D "WIN32" /I "..\include" "MQTTV3Sample.c" /link /
nologo ..\windows_ia32\mqttv3c.lib
set path=%path%;..\windows_ia32;
start "MQTT Subscriber" MQTTV3Sample -a subscribe -b localhost -p 1883
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
MQTTV3Sample -b localhost -p 1883
pause
endlocal
```

結果

發佈者及訂閱者會將輸出寫入其指令視窗：

```
Setting environment for using Microsoft Visual Studio 2008 x86 tools.
MQTTV3Sample.c
Connected to tcp://localhost:1883
Publishing to topic "MQTTV3Sample/C/v3" qos 2
Disconnected
Press any key to continue . . .
```

圖 10: 來自發佈者的輸出

```
Connected to tcp://localhost:1883
Subscribing to topic "MQTTV3Sample/#" qos 2
Topic:      MQTTV3Sample/C/v3
Message:    Message from MQTTv3 C client
QoS:       2
```

圖 11: 來自訂閱者的輸出

從 Microsoft Visual Studio 編譯及執行 MQTT 用戶端範例 C 應用程式

從 Microsoft Visual Studio 編譯及執行 MQTT 用戶端範例 C 應用程式。範例在 Mobile Messaging and M2M 用戶端套件中。它示範了 MQTT 發佈者及訂閱者。

開始之前

範例使用 Microsoft Visual Studio 2010。您可以在 Windows 及其他平台上使用其他 C 開發環境；例如，[Eclipse IDE for C/C++ Developers](#)。

關於這項作業

使用 Microsoft Visual Studio 2010 編譯並執行 C 範例 MQTTV3Sample。MQTTV3Sample.c 在 SDK 用戶端子目錄 `sdkroot\SDK\clients\c\samples` 中。

程序

1. 啟動 Microsoft Visual Studio。
2. 從現有程式碼建立新的專案。
 - a) 按一下 **檔案 > 新建 > 現有程式碼中的專案**。
 - b) 選取 **Visual C++** 作為要建立的專案類型。
 - c) 按下一步。
3. 在「專案位置和原始程式檔」視窗中指定參數。
 - a) 按一下 **瀏覽**，並尋找 `sdkroot\SDK\clients\c\samples` 目錄。
 - b) 將專案命名為 MQTTV3Sample。

- c) 按下一步。
 - 4. 在專案類型清單中選取主控台應用程式專案。按一下完成
 - 5. 僅配置「除錯」配置。
依預設，Microsoft Visual Studio 會建立發行及除錯配置。在指導教學中，會配置除錯配置。若要隱藏建置錯誤，請清除發行配置的建置選項。
 - a) 按一下 專案 > **MQTTV3Sample** 內容 > 配置管理程式。選取發行作為使用中的方案組態，並清除建置。
 - b) 選取 除錯 作為 作用中解決方案配置 > 關閉。
請檢查您是否要在下列所有步驟中修改「除錯」配置。
 - 6. 修改「MQTTV3Sample 內容頁」中的 C/C++ 設定。
 - a) 在 **MQTTV3Sample** 內容頁面 視窗中，開啟 配置內容 > C/C++ > 一般。
 - b) 在一般內容的清單中，按一下其他 **Include** 目錄，然後將您的目錄路徑新增至 `sdkroot\SDK\clients\c\include` 並按一下套用。
 - 7. 修改連結器設定
 - a) 開啟 配置內容 > 鏈結器 > 一般。
 - b) 在一般內容清單中，按一下其他程式庫目錄，並將您的目錄路徑新增至 `sdkroot\SDK\clients\c\windows_ia32`
 - c) 在「連結器」內容的清單中，按一下命令列。在其他選項資料輸入區中，鍵入 `mqttv3c.lib`，然後按一下套用。
 - 8. 從專案移除 MQTTV3SSample.c 原始檔。
 - a) 在「解決方案瀏覽器」視窗中開啟 **MQTTV3sample** > 原始檔 資料夾。
 - b) 用滑鼠右鍵按一下 **MQTTV3SSample.c** > 從專案排除
 - 9. 建置 MQTTV3Sample 專案。
 - a) 在「方案總管」中用滑鼠右鍵按一下 **MQTTV3sample** 專案，然後按一下建置。
建置會完成，且無任何錯誤。
 - 10. 新增兩個新專案，以執行 **MQTTV3Sample** 作為「訂閱者」及「發佈者」執行時期實例。
「發佈者」及「訂閱者」專案將包含執行 **MQTTV3Sample** 的指令。它們未建置，且未包含任何程式碼。
 - a) 在「解決方案瀏覽器」中，用滑鼠右鍵按一下 解決方案 `MQTTV3Sample` > 新增 > 新專案。
 - b) 在名稱欄位中，鍵入 Subscriber。保留選取 **Win32** 主控台應用程式。按一下確定。
「Win32 應用程式精靈」即會啟動。
 - c) 在「Win32 應用程式精靈」中，按一下下一步。勾選空專案 > 完成
 - d) 重複這些步驟以新增「發佈者」專案。
 - e) 用滑鼠右鍵按一下「訂閱者」專案，然後按一下設為啟始專案
- 第 27 頁的圖 12 顯示了「方案總管」視窗。

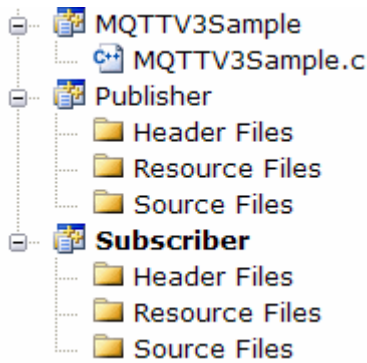


圖 12: MQTTV3Sample 解決方案

11. 配置「訂閱者」內容頁。

- a) 在「解決方案瀏覽器」中，用滑鼠右鍵按一下 **訂閱者 內容 > 配置內容 > 內容 > 除錯** 檢查視窗標題是否為「訂閱者內容頁」。
- b) 按一下**環境**。鍵入 `path=%path%;sdkroot\SDK\clients\c\windows_ia32`，然後按一下**套用**。
變更 `sdkroot` 以符合您的環境。
- c) 按一下**命令**，並用 MQTTV3Sample 模組的路徑取代 `$(TargetPath)`
例如，`sdkroot\SDK\clients\c\samples\debug\MQTTV3Sample`
變更 `sdkroot` 以符合您的環境。
- d) 按一下 **指令引數**，並輸入 `-a subscribe -b localhost -p 1883`，然後按一下**套用**。

12. 配置「發佈者」內容頁。

提示: 您可以按一下「方案總管」視窗中的專案來切換內容頁專案。

- a) 針對「發佈者」重複「訂閱者」步驟。
指令引數為 `-b localhost -p 1883`

13. 停止建置「發佈者」及「訂閱者」專案的建置程序。

- a) 在任何專案的內容頁中，按一下**配置管理程式**，並在「發行」及「除錯」配置中針對「發佈者」和「訂閱者」清除**建置**。按一下**關閉**。

14. 執行範例。

- a) 按一下 **F5** 以啟動「訂閱者」
- b) 在「方案總管」中用滑鼠右鍵按一下**發佈者**，然後按一下**除錯 > 啟動新的實例**。

結果

發佈者及訂閱者會輸出至指令視窗。Visual Studio 會關閉發佈者視窗。請查看下圖中所示的訂閱者視窗，然後予以關閉。

```
Connected to tcp://localhost:1883
Subscribing to topic "MQTTV3Sample/#" qos 2
Topic:          MQTTV3Sample/C/v3
Message:        Message from MQTTv3 C client
QoS:           2
```

圖 13: 來自訂閱者的輸出

下一步

建置並執行非同步的發佈者及訂閱者。範例是 `sdkroot\SDK\clients\c\samples` 中的 `MQTTV3ASample.c` 和 `MQTTV3ASSample.c`。

建置適用於 C 的 MQTT 用戶端程式庫

遵循下列步驟，以建置適用於 C 的 MQTT 用戶端程式庫。本主題包括用於許多平台的編譯及鏈結開關，以及在 iOS 和 Windows 上建置程式庫的範例。

開始之前

1. 僅在必要時建置 C 用戶端程式庫。鏈結 SDK\clients\c 子目錄中 SDK（軟體開發套件）中的預先建置的用戶端程式庫（如果用戶端與您的目標平台相符）。
2. 配置 MQTT 伺服器，以測試您使用 MQTT 用戶端範例 C 應用程式建置的程式庫。請參閱第 122 頁的『開始使用 MQTT 伺服器』。透過執行其中一個 MQTT 用戶端範例應用程式，來驗證伺服器配置。
3. 如果建置的是安全的 C 程式庫版本（支援 SSL (Secure Sockets Layer)），您還必須建置 OpenSSL 程式庫。請參閱第 41 頁的『建置 OpenSSL 套件』。

重要：下載及重新配送 OpenSSL 套件必須遵守嚴格的匯入及匯出法規，以及開放程式碼授權條件。在決定是否下載套件之前，請仔細注意限制及警告。

關於這項作業

遵循第 41 頁的『建置 OpenSSL 套件』中的指示，以下載及建置 OpenSSL 程式庫。您必須建置 OpenSSL，才能建置安全的適用於 C 的 MQTT 用戶端程式庫版本。建置不受保護的 MQTT 程式庫版本不需要 OpenSSL。這些步驟包括建置用於 iOS 及 Windows 的程式庫的範例。

透過將 C 開發程式庫工具及 MQTT 軟體開發工具箱 (SDK) 下載到建置平台上，來建置適用於 C 的 MQTT 用戶端程式庫。撰寫 make 檔以建置用於您的目標平台的程式庫，從而併入第 29 頁的『MQTT 不同平台的建置選項』中記載的選項。這裡給出建置及執行 make 檔的平台專用步驟：

- **iOS** 第 30 頁的『在 Apple Mac 上建置適用於 C 的 MQTT 用戶端程式庫，以便與 iOS 裝置搭配使用』
- **Windows** 第 36 頁的『在 Windows 上建置 MQTT 程式庫』

程序

1. 在您要建置的平台上安裝 C 開發環境。

本主題的範例中的 make 檔將下列工具設為目標：

- **iOS** 若為 iOS，在具有 OS X 10.8.2 的 Apple Mac 上，使用 [Xcode](#) 中的 iOS 開發工具。
- **Linux** 對於 Linux，為 Red Hat Enterprise Linux 6.2 版中的 gcc 4.4.6 版。
glibc C 程式庫的最低支援層次為 2.12，Linux 核心的最低支援層次為 2.6.32。
- **Windows** 對於 Microsoft Windows，為 Visual Studio 10.0 版。

2. 下載 Mobile Messaging and M2M 用戶端套件 並安裝 MQTT SDK。

沒有安裝程式，您只是展開了下載的檔案。

- a. 下載 [Mobile Messaging and M2M 用戶端套件](#)。
 - b. 建立您要安裝 SDK 的資料夾。
您可能想將資料夾命名為 MQTT。在這裡，此資料夾的路徑稱為 *sdkroot*。
 - c. 將壓縮的 Mobile Messaging and M2M 用戶端套件 檔案內容展開到 *sdkroot* 中。展開會建立以 *sdkroot*\SDK 開頭的目錄樹。
3. 展開適用於 C 的 MQTT 用戶端程式庫的原始碼。
原始碼壓縮檔為 *sdkroot*\SDK\clients\c\source.zip。
 4. 選擇性的: 建置 OpenSSL。
請參閱第 41 頁的『建置 OpenSSL 套件』。

5. 建置適用於 C 的 MQTT 用戶端程式庫。

第 29 頁的『MQTT 不同平台的建置選項』中列出了建置程式庫的指令及選項。

遵循下列範例中的步驟撰寫 make 檔，以針對您的目標平台建置適用於 C 的 MQTT 用戶端程式庫。

- 第 30 頁的『在 Apple Mac 上建置適用於 C 的 MQTT 用戶端程式庫，以便與 iOS 裝置搭配使用』
- 第 36 頁的『在 Windows 上建置 MQTT 程式庫』

MQTT 不同平台的建置選項

下表列出可以在各種平台上建置適用於 C 之 MQTT 用戶端程式庫的編譯器及建置選項。

平台	Compiler	Compiler Options	Linker Options	Extra Options
AIX	gcc	-fPIC -Os -Wall -DREVERSED -I MQTTCLIENT_DIR	-Wl,-G	
Linux s390x				-m64
Linux x86-64				-m32
Linux x86-32				
Linux ARM (glibc)	arm-linux-gcc	-fPIC -Os -Wall -I MQTTCLIENT_DIR	-shared -Wl,-soname,libmqttv3c.so	
Linux ARM (uclibc)	arm-unknown-linux-uclibcgnueabi-gcc			
Windows 32 位元	cl	/D "WIN32" /D "_UNICODE" /D "UNICODE" /D "_CRT_SECURE_NO_WARNINGS" / nologo /c /O2 /W3 / Fd /MD /TC	/nologo /machine:x86 / manifest kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib ws2_32.lib / implib:mqttv3c.lib) (/pdb:mqttv3c.pdb) / map:mqttv3c.map)	

表 2: MQTT 不同平台的建置選項 (繼續)

平台	Compiler	Compiler Options	Linker Options	Extra Options
iOS ARMv7	gcc -arch armv7	-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE -DOPENSSL -Os -Wall -fomit-frame-pointer -isysroot /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS6.0.sdk	-L/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS6.0.sdk/usr/lib/system	
iOS ARMv7s	gcc -arch armv7s	-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE -DOPENSSL -Os -Wall -fomit-frame-pointer -isysroot /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS6.0.sdk	-L/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS6.0.sdk/usr/lib/system	
iOS ARMi386	gcc -arch i386	-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE -DOPENSSL -Os -Wall -fomit-frame-pointer -isysroot /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator6.0.sdk	-L/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator6.0.sdk/usr/lib/system	

iOS 在 Apple Mac 上建置適用於 C 的 MQTT 用戶端程式庫，以便與 iOS 裝置搭配使用

遵循下列步驟撰寫 make 檔，以在 Apple Mac 上建置適用於 C 的 MQTT 用戶端程式庫，以便後續與 iOS 裝置搭配使用。

開始之前

1. 在 Apple Mac OS X 10.8.2 或更新版本上安裝建置工具，開發及執行 make 檔。
2. 安裝 Xcode 的指令行工具，其中包括 **make** 程式。從 [Xcode](#) 下載指令行工具。

關於這項作業

建立 make 檔，以針對執行 ARMv7 或 ARMv7s 處理器的 iPhone 或 iPad，及在 i386-64 位元處理器上執行的 iPhone 模擬器，建置適用於 C 的 MQTT 用戶端程式庫。請參閱 [iOS 裝置清單](#)。

提示: 第 34 頁的『MQTTios.mak make 檔清單』列出了完整的 make 檔。

1. 複製清單並貼至檔案中。
2. 將追蹤目標的每行的前導字元轉換為標籤；請參閱步驟 第 32 頁的『8』。
3. 使用此程序的步驟 第 34 頁的『9』 中列出的指令加以執行。

程序

1. 下載並安裝 iOS 開發工具。
 - a. 使用具有管理專用權的使用者 ID 登入。
 - b. 檢查您的 Apple Mac 是否為 10.8.2 版或更新版本。
 - c. 移至網站 [Xcode](#)，以從 Mac App Store 下載 Xcode。
 - d. 安裝 Xcode、指令行環境及模擬器。

如果 Mac 應用程式商店提供了多種模擬器版本，請選擇與您為應用程式所設定目標的 iOS 層次相容的版本。

2. 建立 make 檔 MQTTios.mak

新增前言：

```
# 在現行目錄中產生建置輸出。
# MQTTCLIENT_DIR 必須指向包含 MQTT 用戶端原始碼的基本目錄。
# 預設 MQTTCLIENT_DIR 是現行目錄
# 預設 TOOL_DIR 是 /Applications/Xcode.app/Contents/Developer/Platforms
# 預設 OPENSLL_DIR 是 sdkroot/openssl，相對於 sdkroot/sdk/clients/c/mqttv3c/src
# OPENSLL_DIR 必須指向包含 OpenSSL 建置的基本目錄。
# 範例 :make -f MQTTios.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src all
```

3. 設定 MQTT 原始碼的位置。

在與 MQTT 原始檔相同的目錄中執行 make 檔，或設定 MQTTCLIENT_DIR 指令行參數：

```
make -f make 檔 MQTTCLIENT_DIR=sdkroot/SDK/clients/c/mqttv3c/src
```

將下列行新增至 make 檔：

```
ifndef MQTTCLIENT_DIR
    MQTTCLIENT_DIR = ${CURDIR}
endif
VPATH = ${MQTTCLIENT_DIR}
```

範例會將 VPATH 設為目錄，其中 **make** 會搜尋未明確識別的原始檔；例如，建置中所需的所有標頭檔。

4. 選擇性的：設定 OpenSSL 程式庫的位置。

建置適用於 C 的 MQTT 用戶端程式庫的 SSL 版本需要此步驟。

將 OpenSSL 程式庫的預設路徑設為已展開 MQTT SDK 的相同目錄。否則，請將 OPENSLL_DIR 設為指令行參數。

```
ifndef OPENSLL_DIR
    OPENSLL_DIR = ${MQTTCLIENT_DIR}/../../../../../../../../openssl-1.0.1c
endif
```

提示：OpenSSL 是包含所有 OpenSSL 子目錄的 OpenSSL 目錄。您可能需要將目錄樹移離您展開它的位置，因為它包含不必要的空的上層目錄。

5. 設定開發工具目錄。

如果您已將 Xcode 安裝在其他位置，請在指令行中設定 TOOL_DIR。

```
ifndef TOOL_DIR
    TOOL_DIR = /Applications/Xcode.app/Contents/Developer/Platforms
endif
IPHONE_SDK = iPhoneOS.platform/Developer/SDKs/iPhoneOS6.0.sdk
IPHONESIM_SDK = iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator6.0.sdk
SDK_ARM = ${TOOL_DIR}/${IPHONE_SDK}
SDK_i386 = ${TOOL_DIR}/${IPHONESIM_SDK}
```

6. 選取建置每一個 MQTT 程式庫所需的所有原始檔。另外，設定要建置的 MQTT 程式庫的名稱及位置。

將下列行新增至 make 檔，以列出所有 MQTT 原始檔：

```
ALL_SOURCE_FILES = ${wildcard ${MQTTCLIENT_DIR}/*.c}
```

原始檔視您要建置同步還是非同步程式庫，及程式庫是否包含 SSL 而定。

新增其中一行以上，視要建置的目標而定。在 darwin_x86_64 目錄中建立共用程式庫。

- 同步、未受保護：

```
MQTTLIB = mqttv3c
SOURCE_FILES = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c ${MQTTCLIENT_DIR}/SSLSocket.c ,
${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN = darwin_x86_64/lib${MQTTLIB} .a
```

- 同步、安全：

```
MQTTLIB_S = mqttv3cs
SOURCE_FILES_S = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c , ${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_S = darwin_x86_64/lib${MQTTLIB_S} .a
```

- 非同步、未受保護：

```
MQTTLIB_A = mqttv3a
SOURCE_FILES_A = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c ${MQTTCLIENT_DIR}/
SSLSocket.c , ${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_A = darwin_x86_64/lib${MQTTLIB_A} .a
```

- 非同步、安全：

```
MQTTLIB_AS = mqttv3as
SOURCE_FILES_AS = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c , ${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_AS = darwin_x86_64/lib${MQTTLIB_AS} .a
```

7. 定義編譯器及編譯器選項。

請參閱 不同平台的 MQTT 建置選項中顯示的不同平台的選項。

- a) 將 Gnu 專案 C 及 C++ (**gcc**) 設為編譯器。

選取三個跨編譯器，為不同的裝置及 iPhone 模擬器建置程式庫：

```
CC = iPhoneOS.platform/Developer/usr/bin/gcc
CC_armv7 = ${TOOL_DIR}/${CC} -arch armv7
CC_armv7s = ${TOOL_DIR}/${CC} -arch armv7s
CC_i386 = ${TOOL_DIR}/${CC} -arch i386
```

- b) 新增編譯器選項。

```
CCFLAGS = -Os -Wall -fomit-frame-pointer
```

- c) 新增併入路徑。

```
CCFLAGS_SO_ARM = ${CCFLAGS} -isysroot ${SDK_ARM} -I${OPENSSL_DIR}/include -L$
${SDK_ARM}/usr/lib/system
CCFLAGS_SO_i386 = ${CCFLAGS} -isysroot ${SDK_i386} -I${OPENSSL_DIR}/include -L$
${SDK_i386}/usr/lib/system
```

8. 定義建置目標。

提示：定義目標實作的每一連續行必須以定位字元開頭。

- a) 定義 **all** 目標。

"all" 目標會建置所有程式庫。

```
all: ${MQTTLIB_DARWIN} ${MQTTLIB_DARWIN_A} ${MQTTLIB_DARWIN_AS} ${MQTTLIB_DARWIN_S}
```

先列出的目標是預設目標。

- a) 建置同步、未受保護的程式庫 libmqttv3c.a。


```

${MQTTLIB_DARWIN}: ${SOURCE_FILES}
-mkdir darwin_x86_64
${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES} -DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
libtool -static -syslibroot ${SDK_ARM} -o ${@.armv7 *.o}
rm *.o
${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES} -DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
libtool -static -syslibroot ${SDK_ARM} -o ${@.armv7s *.o}
rm *.o
${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES} -DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
libtool -static -syslibroot ${SDK_i386} -o ${@.i386 *.o}
rm *.o
lipo -create ${@.armv7} ${@.armv7s} ${@.i386} -output ${@}

```

陳述式 `rm *.o` 會刪除針對每一個檔案庫建立的所有物件檔。 `lipo` 將所有三個程式庫連結成一個檔案。

b) 建置非同步、未受保護的程式庫 `libmqttv3a.a`。

```

${MQTTLIB_DARWIN_A}: ${SOURCE_FILES_A}
-mkdir darwin_x86_64
${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_A} -DMQTT_ASYNC
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
libtool -static -syslibroot ${SDK_ARM} -o ${@.armv7 *.o}
rm *.o
${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_A} -DMQTT_ASYNC
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
libtool -static -syslibroot ${SDK_ARM} -o ${@.armv7s *.o}
rm *.o
${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES_A} -DMQTT_ASYNC
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
libtool -static -syslibroot ${SDK_i386} -o ${@.i386 *.o}
rm *.o
lipo -create ${@.armv7} ${@.armv7s} ${@.i386} -output ${@}

```

陳述式 `rm *.o` 會刪除針對每一個檔案庫建立的所有物件檔。 `lipo` 將所有三個程式庫連結成一個檔案。

c) 建置同步、安全的程式庫 `libmqttv3cs.a`。

```

${MQTTLIB_DARWIN_S}: ${SOURCE_FILES_S}
-mkdir darwin_x86_64
${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_S} -DOPENSSL -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7 -lssl -lcrypto -o
${@.armv7 *.o}
rm *.o
${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_S} -DOPENSSL -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7s -lssl -lcrypto -o
${@.armv7s *.o}
rm *.o
${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES_S} -DOPENSSL -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/i386 -lssl -lcrypto -o
${@.i386 *.o}
rm *.o
lipo -create ${@.armv7} ${@.armv7s} ${@.i386} -output ${@}

```

陳述式 `rm *.o` 會刪除針對每一個檔案庫建立的所有物件檔。 `lipo` 將所有三個程式庫連結成一個檔案。

d) 建置非同步、安全的程式庫 `libmqttv3as.a`。

```

${MQTTLIB_DARWIN_AS}: ${SOURCE_FILES_AS}
-mkdir darwin_x86_64
${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_AS} -DMQTT_ASYNC -DOPENSSL
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7 -lssl -lcrypto -o
${@.armv7 *.o}
rm *.o
${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_AS} -DMQTT_ASYNC -DOPENSSL
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7s -lssl -lcrypto -o
${@.armv7s *.o}
rm *.o
${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES_AS} -DMQTT_ASYNC -DOPENSSL
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE

```

```
libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/i386 -lssl -lcrypto -o
${@.i386 *.o
rm *.o
lipo -create ${@.armv7} ${@.armv7s} ${@.i386} -output $@
```

陳述式 `rm *.o` 會刪除針對每一個檔案庫建立的所有物件檔。 `lipo` 將所有三個程式庫連結成一個檔案。

e) 定義 **clean** 目標。

"clean" 目標會移除 `make` 檔產生的所有檔案及目錄

```
。系統：清除
clean:
    -rm -f *.obj
    -rm -f -r darwin_x86_64
```

9. 執行 `make` 檔。

```
make -f MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src
```

結果

`sdkroot/sdk/clients/c/mqttv3c/src/darwin_x86_64` 目錄中會建立下列檔案。

```
libmqttv3c.a.armv7
libmqttv3c.a.armv7s
libmqttv3c.a.i386
libmqttv3c.a
libmqttv3a.a.armv7
libmqttv3a.a.armv7s
libmqttv3a.a.i386
libmqttv3a.a
libmqttv3cs.a.armv7
libmqttv3cs.a.armv7s
libmqttv3cs.a.i386
libmqttv3cs.a
libmqttv3as.a.armv7
libmqttv3as.a.armv7s
libmqttv3as.a.i386
libmqttv3as.a
```

MQTTios.mak make 檔清單

```
# 在現行目錄中產生建置輸出。
# MQTTCLIENT_DIR 必須指向包含 MQTT 用戶端原始碼的基本目錄。
# 預設 MQTTCLIENT_DIR 是現行目錄
# 預設 TOOL_DIR 是 /Applications/Xcode.app/Contents/Developer/Platforms
# 預設 OPENSSL_DIR 是 sdkroot/openssl，相對於 sdkroot/sdk/clients/c/mqttv3c/src
# OPENSSL_DIR 必須指向包含 OpenSSL 建置的基本目錄。
# 範例 :make -f MQTTios.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src all

ifndef MQTTCLIENT_DIR
    MQTTCLIENT_DIR = ${CURDIR}
endif
VPATH = ${MQTTCLIENT_DIR}
ifndef OPENSSL_DIR
    OPENSSL_DIR = ${MQTTCLIENT_DIR}/../../../../../../../../openssl-1.0.1c
endif
ALL_SOURCE_FILES = ${wildcard ${MQTTCLIENT_DIR}/*.c}
ifndef TOOL_DIR
    TOOL_DIR = /Applications/Xcode.app/Contents/Developer/Platforms endif
IPHONE_SDK = iPhoneOS.platform/Developer/SDKs/iPhoneOS6.0.sdk
IPHONESIM_SDK = iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator6.0.sdk
SDK_ARM = ${TOOL_DIR}/${IPHONE_SDK}
SDK_i386 = ${TOOL_DIR}/${IPHONESIM_SDK}

MQTTLIB = mqttv3c
SOURCE_FILES = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c ${MQTTCLIENT_DIR}/SSLSocket.c, $
{ALL_SOURCE_FILES}
MQTTLIB_DARWIN = darwin_x86_64/lib${MQTTLIB} .a
MQTTLIB_S = mqttv3cs
SOURCE_FILES_S = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c, ${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_S = darwin_x86_64/lib${MQTTLIB_S} .a
MQTTLIB_A = mqttv3a
```

```

SOURCE_FILES_A = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c ${MQTTCLIENT_DIR}/SSLSocket.c, ${
ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_A = darwin_x86_64/lib${MQTTLIB_A} .a
MQTTLIB_AS = mqttv3as
SOURCE_FILES_AS = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c, ${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_AS = darwin_x86_64/lib${MQTTLIB_AS} .a

# compiler
CC = iPhoneOS.platform/Developer/usr/bin/gcc
CC_armv7 = ${TOOL_DIR}/${CC} -arch armv7
CC_armv7s = ${TOOL_DIR}/${CC} -arch armv7s
CC_i386 = ${TOOL_DIR}/${CC} -arch i386
CCFLAGS = -Os -Wall -fomit-frame-pointer
CCFLAGS_SO_ARM = ${CCFLAGS} -isysroot ${SDK_ARM} -I${OPENSSL_DIR}/include -L${SDK_ARM}/usr/lib/
system
CCFLAGS_SO_i386 = ${CCFLAGS} -isysroot ${SDK_i386} -I${OPENSSL_DIR}/include -L$
${SDK_i386}/usr/lib/system

# targets
all: ${MQTTLIB_DARWIN} ${MQTTLIB_DARWIN_A} ${MQTTLIB_DARWIN_AS} ${MQTTLIB_DARWIN_S}

${MQTTLIB_DARWIN}: ${SOURCE_FILES}
    -mkdir darwin_x86_64
    ${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES} -DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -o ${@.armv7} *.o
    rm *.o
    ${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES} -DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -o ${@.armv7s} *.o
    rm *.o
    ${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES} -DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_i386} -o ${@.i386} *.o
    rm *.o
    lipo -create ${@.armv7} ${@.armv7s} ${@.i386} -output ${@}

${MQTTLIB_DARWIN_A}: ${SOURCE_FILES_A}
    -mkdir darwin_x86_64
    ${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_A} -DMQTT_ASYNC -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -o ${@.armv7} *.o
    rm *.o
    ${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_A} -DMQTT_ASYNC -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -o ${@.armv7s} *.o
    rm *.o
    ${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES_A} -DMQTT_ASYNC -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
    libtool -static -syslibroot ${SDK_i386} -o ${@.i386} *.o
    rm *.o
    lipo -create ${@.armv7} ${@.armv7s} ${@.i386} -output ${@}

${MQTTLIB_DARWIN_S}: ${SOURCE_FILES_S}
    -mkdir darwin_x86_64
    ${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_S} -DOPENSSL -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7 -lssl -lcrypto -o ${@.armv7} *.o
    rm *.o
    ${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_S} -DOPENSSL -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7s -lssl -lcrypto -o ${@.armv7s}
*.o
    rm *.o
    ${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES_S} -DOPENSSL -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/i386 -lssl -lcrypto -o ${@.i386} *.o
    rm *.o
    lipo -create ${@.armv7} ${@.armv7s} ${@.i386} -output ${@}

${MQTTLIB_DARWIN_AS}: ${SOURCE_FILES_AS}
    -mkdir darwin_x86_64
    ${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_AS} -DMQTT_ASYNC -DOPENSSL
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7 -lssl -lcrypto -o ${@.armv7} *.o
    rm *.o
    ${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_AS} -DMQTT_ASYNC -DOPENSSL
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7s -lssl -lcrypto -o ${@.armv7s}
*.o
    rm *.o
    ${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES_AS} -DMQTT_ASYNC -DOPENSSL
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/i386 -lssl -lcrypto -o ${@.i386} *.o
    rm *.o

```

```
lipo -create $@.armv7 $@.armv7s $@.i386 -output $@
```

```
。系統：清除  
clean:  
-rm -f *.obj  
-rm -f -r darwin_x86_64
```

Windows 在 Windows 上建置 MQTT 程式庫

遵循下列步驟撰寫 make 檔，以在 Windows 上建置適用於 C 的 MQTT 用戶端程式庫。

開始之前

1. 必要的話，請在建置工作站上安裝與針對 Gnu make 所撰寫的 make 檔相容的 **Make** 版本；否則請下載 Gnu make 並建置它。請參閱 [Gnu Make](#)。網站 [Make for Windows](#) 提供 **Make for Windows** 的可安裝版本。
2. 您還需要用於 Windows 的 Linux 指令，以在 make 檔範例中使用 clean 目標。您可以從 [Cygwin](#) 之類的網站取得 Windows 的 Linux 指令。

關於這項作業

建立 make 檔，以為 Windows 32 位元系統建置適用於 C 的 MQTT 用戶端程式庫。

提示：第 40 頁的『MQTTwin.mak make 檔清單』列出了完整的 make 檔。

1. 複製清單並貼至檔案中。
2. 將追蹤目標的每行的前導字元轉換為標籤；請參閱步驟第 38 頁的『7』。
3. 使用此程序的步驟第 39 頁的『9』中列出的指令加以執行。

程序

1. 建立 make 檔 MQTTwin.mak

新增前言：

```
# 在現行目錄中產生建置輸出。  
# MQTTCLIENT_DIR 必須指向包含 MQTT 用戶端原始碼的基本目錄。  
# 預設 MQTTCLIENT_DIR 是現行目錄  
# 預設 OPENSSL_DIR 是 sdkroot\openssl，相對於 sdkroot\sdk\clients\c\mqttv3c/src  
# OPENSSL_DIR 必須指向包含 OpenSSL 建置的基本目錄。  
# 範例 :make -f MQTTwin.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src  
# 設定建置環境，例如：  
# %comspec% /k "" C:\Program Files\Microsoft Visual Studio 9.0\VC\vcvarsall.bat" x86  
# set path= %path%; C:\Program Files\GnuWin32\bin;C:\cygwin\bin
```

2. 設定 MQTT 原始碼的位置。

在與 MQTT 原始檔相同的目錄中執行 make 檔，或設定 MQTTCLIENT_DIR 指令行參數：

```
make -f make 檔 MQTTCLIENT_DIR=sdkroot/SDK/clients/c/mqttv3c/src
```

將下列行新增至 make 檔：

```
ifndef MQTTCLIENT_DIR  
    MQTTCLIENT_DIR = ${CURDIR}  
endif  
VPATH = ${MQTTCLIENT_DIR}
```

範例會將 VPATH 設為目錄，其中 **make** 會搜尋未明確識別的原始檔；例如，建置中所需的所有標頭檔。

3. 選擇性的：設定 OpenSSL 程式庫的位置。

建置適用於 C 的 MQTT 用戶端程式庫的 SSL 版本需要此步驟。

將 OpenSSL 程式庫的預設路徑設為已展開 MQTT SDK 的相同目錄。否則，請將 OPENSSL_DIR 設為指令行參數。

```
ifndef OPENSSL_DIR
    OPENSSL_DIR = ${MQTTCLIENT_DIR}/../../../../../../../../openssl-1.0.1c
endif
```

提示: *OpenSSL* 是包含所有 OpenSSL 子目錄的 OpenSSL 目錄。您可能需要將目錄樹移離您展開它的位置，因為它包含不必要的空的上層目錄。

4. 選取建置每一個 MQTT 程式庫所需的所有原始檔。另外，設定要建置的 MQTT 程式庫的名稱及位置。

將下列行新增至 make 檔，以列出所有 MQTT 原始檔：

```
ALL_SOURCE_FILES = ${wildcard ${MQTTCLIENT_DIR}/*.c}
```

原始檔視您要建置同步還是非同步程式庫，及程式庫是否包含 SSL 而定。

新增其中一行以上，視要建置的目標而定。windows_ia32 目錄中會建立共用程式庫及資訊清單。

- 同步、未受保護：

```
MQTTLIB = mqttv3c
MQTTDLL = windows_ia32/${MQTTLIB}.dll
SOURCE_FILES = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c ${MQTTCLIENT_DIR}/SSLSocket.c,
${ALL_SOURCE_FILES}}
MANIFEST = mt -manifest ${MQTTDLL}.manifest -outputresource: ${MQTTDLL}\; 2
```

- 同步、安全：

```
MQTTLIB_S = mqttv3cs
MQTTDLL_S = windows_ia32/${MQTTLIB_S}.dll
SOURCE_FILES_S = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c, ${ALL_SOURCE_FILES}}
MANIFEST_S = mt -manifest ${MQTTDLL_S}.manifest -outputresource: ${MQTTDLL_S}\; 2
```

- 非同步、未受保護：

```
MQTTLIB_A = mqttv3a
MQTTDLL_A = windows_ia32/${MQTTLIB_A}.dll
SOURCE_FILES_A = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c ${MQTTCLIENT_DIR}/
SSLSocket.c, ${ALL_SOURCE_FILES}}
MANIFEST_S = mt -manifest ${MQTTDLL_S}.manifest -outputresource: ${MQTTDLL_S}\; 2
```

- 非同步、安全：

```
MQTTLIB_AS = mqttv3as
MQTTDLL_AS = windows_ia32/${MQTTLIB_AS}.dll
SOURCE_FILES_AS = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c, ${ALL_SOURCE_FILES}}
MANIFEST_S = mt -manifest ${MQTTDLL_S}.manifest -outputresource: ${MQTTDLL_S}\; 2
```

5. 定義編譯器及編譯器選項。

請參閱 [不同平台的 MQTT 建置選項](#) 中顯示的不同平台的選項。

- a) 將 Microsoft Visual C++ 設為編譯器。

```
CC = cl
```

- b) 新增前置處理器選項。

```
CPPFLAGS = /D "WIN32" /D "_UNICODE" /D "UNICODE" /D "_CRT_SECURE_NO_WARNINGS"
```

- c) 新增編譯器選項。

```
CFLAGS = /nologo /c /O2 /W3 /Fd /MD /TC
```

- d) 新增併入路徑。

```
INC = /I ${MQTTCLIENT_DIR} /I ${MQTTCLIENT_DIR}/..
```

- e) 選擇性的：如果您建置的是安全程式庫，請新增前置處理器選項。

```
CPPFLAGS_S = ${CPPFLAGS} /D "OPENSSL"
```

- f) 選擇性的: 如果您建置的是安全程式庫, 請新增 OpenSSL 標頭檔。

```
INC_S = ${INC} /I ${OPENSSL_DIR}/inc32/
```

提示: 標頭檔在 `${OPENSSL_DIR}/inc32/openssl` 中, 但 `ssl.h` 檔案隨附於 "openssl/ssl.h"。

6. 設定鏈結器及鏈結器選項。

- a) 將 Microsoft Visual C++ 設為鏈結器。

```
LD = link
```

- b) 新增鏈結器選項。

```
LINKFLAGS = /nologo /machine:x86 /manifest /dll
```

- c) 新增程式庫路徑。

```
WINLIBS = kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib\  
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib\  
odbc32.lib odbccp32.lib ws2_32.lib
```

- d) 新增中間輸出檔。

```
IMP = /implib: ${@:.dll=.lib}  
LIBPDB = /pdb: ${@:.dll=.pdb}  
LIBMAP = /map: ${@:.dll=.map}
```

- e) 選擇性的: 如果您建置的是安全程式庫, 請新增 OpenSSL 程式庫。

```
WINLIBS_S = ${WINLIBS} crypt32.lib ssleay32.lib libeay32.lib
```

- f) 選擇性的: 如果您建置的是安全程式庫, 請新增 OpenSSL 程式庫路徑。

```
LIBPATH_S = /LIBPATH:${OPENSSL_DIR}/lib
```

7. 定義四個建置目標。

- a) 定義 **all** 目標。

提示: 定義目標實作的每一連續行必須以定位字元開頭。

"all" 目標會建置所有程式庫。

```
all: ${MQTTDLL} ${MQTTDLL_A} ${MQTTDLL_AS} ${MQTTDLL_S}
```

- b) 建置同步、未受保護的程式庫 `mqttv3c.dll`。

```
${MQTTDLL}: ${SOURCE_FILES}  
-mkdir windows_ia32  
-rm ${CURDIR}/MQTTAsync.obj  
${CC} ${CPPFLAGS} ${CFLAGS} ${INC} /Fo ${SOURCE_FILES}  
${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS} *.obj /out: ${MQTTDLL}  
${MANIFEST}
```

陳述式 `-rm ${CURDIR}/MQTTAsync.obj` 會刪除為較早目標建立的任何 `MQTTAsync.obj`。
`MQTTAsync.obj` 和 `MQTTClient.obj` 互斥。

- c) 建置非同步、未受保護的程式庫 `mqttv3a.dll`。

```
${MQTTDLL_A}: ${SOURCE_FILES_A}  
-mkdir windows_ia32  
-rm ${CURDIR}/MQTTClient.obj  
${CC} ${CPPFLAGS} ${CFLAGS} ${INC} /Fo ${SOURCE_FILES_A}  
${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS} *.obj /out: ${MQTTDLL_A}  
${MANIFEST_A}
```

陳述式 `-rm ${CURDIR}/MQTTClient.obj` 會刪除為較早目標建立的任何 `MQTTClient.obj`。
`MQTTAsync.obj` 和 `MQTTClient.obj` 互斥。

- d) 建置同步、安全的程式庫 `mqttv3cs.dll`。

```
MQTTDLL_S}: ${SOURCE_FILES_S}
    -mkdir windows_ia32
    -rm ${CURDIR}/MQTTAsync.obj
    ${CC} ${CPPFLAGS_S} ${CFLAGS} ${INC_S} /Fo ${SOURCE_FILES}
    ${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS_S} ${LIBPATH_S} *.obj /out: $
    MQTTDLL_S}
    ${MANIFEST_S}
```

陳述式 `-rm ${CURDIR}/MQTTAsync.obj` 會刪除為較早目標建立的任何 `MQTTAsync.obj`。
`MQTTAsync.obj` 和 `MQTTClient.obj` 互斥。

- e) 建置非同步、安全的程式庫 `mqttv3as.dll`。

```
MQTTDLL_AS}: ${SOURCE_FILES_AS}
    -rm ${CURDIR}/MQTTClient.obj
    ${CC} ${CPPFLAGS_S} ${CFLAGS} ${INC_S} /Fo ${SOURCE_FILES_AS}
    ${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS_S} ${LIBPATH_S} *.obj /out: $
    (MQTTDLL_AS}
    $(MANIFEST_as}
```

陳述式 `-rm $(CURDIR)/MQTTClient.obj` 會刪除為較早目標建立的任何 `MQTTClient.obj`。
`MQTTAsync.obj` 和 `MQTTClient.obj` 互斥。

- f) 定義 **clean** 目標。

"clean" 目標會移除 make 檔產生的所有檔案及目錄

```
。系統：清除
clean:
    -rm -f *.obj
    -rm -f -r windows_ia32
```

8. 設定執行 make 檔的 Windows 路徑。

設定斜體部分，以匹配您的安裝。

- a) 設定 Microsoft Visual Studio 環境。

```
%comspec% /k "%C:\Program Files\Microsoft Visual Studio 9.0\VC\vcvarsall.bat" x86
```

- b) 設定 Path 變數以包含 make 程式及 Linux 指令環境。

```
set path=%path%;C:\Program Files\GnuWin32\bin;C:\cygwin\bin
```

9. 執行 make 檔。

```
make -f MQTTw.in.mak MQTTCLIENT_DIR=sdkroot/SDK/clients/c/mqttv3c/src
```

提示：檔案分隔字元必須是正斜線，而不是反斜線。

結果

`sdkroot\SDK\clients\c\mqttv3c\src\windows_ia32` 目錄中會建立下列檔案。

```
mqttv3a.dll
mqttv3a.dll.manifest
mqttv3a.exp
mqttv3a.lib
mqttv3a.map
mqttv3as.dll
mqttv3as.dll.manifest
mqttv3as.exp
mqttv3as.lib
mqttv3as.map
mqttv3c.dll
mqttv3c.dll.manifest
mqttv3c.exp
```

```
mqt3c.lib
mqt3c.map
mqt3cs.dll
mqt3cs.dll.manifest
mqt3cs.exp
mqt3cs.lib
mqt3cs.map
```

MQTTwin.mak make 檔清單

```
# 在現行目錄中產生建置輸出。
# MQTTCLIENT_DIR 必須指向包含 MQTT 用戶端原始碼的基本目錄。
# 預設 MQTTCLIENT_DIR 是現行目錄
# 預設 OPENSSL_DIR 是 sdkroot\openssl, 相對於 sdkroot\sdk\clients\c\mqt3c\src
# OPENSSL_DIR 必須指向包含 OpenSSL 建置的基本目錄。
# 範例 :make -f MQTTwin.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqt3c/src
# 設定建置環境, 例如:
# %comspec% /k "" C:\Program Files\Microsoft Visual Studio 9.0\VC\vcvarsall.bat" x86
# set path= %path%; C:\Program Files\GnuWin32\bin;C:\cygwin\bin
ifndef MQTTCLIENT_DIR
    MQTTCLIENT_DIR = ${CURDIR}
endif
VPATH = ${MQTTCLIENT_DIR}
ifndef OPENSSL_DIR
    OPENSSL_DIR = ${MQTTCLIENT_DIR}/../../../../../openssl-1.0.1c
endif

ALL_SOURCE_FILES = ${wildcard ${MQTTCLIENT_DIR}/*.c}

MQTTLIB = mqt3c
MQTTDLL = windows_ia32/${MQTTLIB}.dll
SOURCE_FILES = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c ${MQTTCLIENT_DIR}/SSLSocket.c, ${ALL_SOURCE_FILES}}
MANIFEST = mt -manifest ${MQTTDLL}.manifest -outputresource: ${MQTTDLL}\; 2
MQTTLIB_S = mqt3cs
MQTTDLL_S = windows_ia32/${MQTTLIB_S}.dll
SOURCE_FILES_S = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c, ${ALL_SOURCE_FILES}}
MANIFEST_S = mt -manifest ${MQTTDLL_S}.manifest -outputresource: ${MQTTDLL_S}\; 2
MQTTLIB_A = mqt3a
MQTTDLL_A = windows_ia32/${MQTTLIB_A}.dll
SOURCE_FILES_A = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c ${MQTTCLIENT_DIR}/SSLSocket.c, ${ALL_SOURCE_FILES}}
MANIFEST_A = mt -manifest ${MQTTDLL_A}.manifest -outputresource: ${MQTTDLL_A}\; 2
MQTTLIB_AS = mqt3as
MQTTDLL_AS = windows_ia32/${MQTTLIB_AS}.dll
SOURCE_FILES_AS = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c, ${ALL_SOURCE_FILES}}
MANIFEST_AS = mt -manifest ${MQTTDLL_AS}.manifest -outputresource: ${MQTTDLL_AS}\; 2

# compiler
CC = cl
CPPFLAGS = /D "WIN32" /D "UNICODE" /D "UNICODE" /D "_CRT_SECURE_NO_WARNINGS"
CFLAGS = /nologo /c /O2 /W3 /Fd /MD /TC
INC = /I ${MQTTCLIENT_DIR} /I ${MQTTCLIENT_DIR}/..
CPPFLAGS_S = ${CPPFLAGS} /D "OPENSSL"
INC_S = ${INC} /I ${OPENSSL_DIR}/inc32/

# linker
LD = link
LINKFLAGS = /nologo /machine:x86 /manifest /dll
WINLIBS = kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib\
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib\
odbc32.lib odbccp32.lib ws2_32.lib
IMP = /implib: ${@:.dll=.lib}
LIBPDB = /pdb: ${@:.dll=.pdb}
LIBMAP = /map: ${@:.dll=.map}
WINLIBS_S = ${WINLIBS} crypt32.lib ssleay32.lib libeay32.lib
LIBPATH_S = /LIBPATH:${OPENSSL_DIR}/lib

# targets
all: ${MQTTDLL} ${MQTTDLL_A} ${MQTTDLL_AS} ${MQTTDLL_S}

${MQTTDLL}: ${SOURCE_FILES}
    -mkdir windows_ia32
    -rm ${CURDIR}/MQTTAsync.obj
    ${CC} ${CPPFLAGS} ${CFLAGS} ${INC} /Fo ${SOURCE_FILES}
    ${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS} *.obj /out: ${MQTTDLL}
    ${MANIFEST}

${MQTTDLL_A}: ${SOURCE_FILES_A}
```



```

-mkdir windows_ia32
-rm ${CURDIR}/MQTTClient.obj
${CC} ${CPPFLAGS} ${CFLAGS} ${INC} /Fo ${SOURCE_FILES_A}
${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS} *.obj /out: ${MQTTDLL_A}
${MANIFEST_A}

${MQTTDLL_S}: ${SOURCE_FILES_S}
-mkdir windows_ia32
-rm ${CURDIR}/MQTTAsync.obj
${CC} ${CPPFLAGS_S} ${CFLAGS} ${INC_S} /Fo ${SOURCE_FILES}
${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS_S} ${LIBPATH_S} *.obj /out: $
${MQTTDLL_S}
${MANIFEST_S}

${MQTTDLL_AS}: ${SOURCE_FILES_AS}
-rm ${CURDIR}/MQTTClient.obj
${CC} ${CPPFLAGS_S} ${CFLAGS} ${INC_S} /Fo ${SOURCE_FILES_AS}
${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS_S} ${LIBPATH_S} *.obj /out: $
(MQTTDLL_AS}
$(MANIFEST_as}



。系統：清除
clean:
-rm -f *.obj
-rm -f -r windows_ia32

```

建置 OpenSSL 套件

建置適用於 C、mqttv3cs 及 mqttv3as 的安全 MQTT 用戶端程式庫之前，請先建置 OpenSSL 套件。建置會建立建置安全的適用於 C 的 MQTT 用戶端程式庫版本及 OpenSSL 憑證管理工具所需的程式庫。

開始之前

1.  iOS make 檔自訂作業適用於執行 iOS6 的目標裝置。iOS 的更舊或更新版本的自訂作業可能不同。
2.  Windows make 檔自訂作業適用於 32 位元視窗。

關於這項作業


下載並安裝 OpenSSL 套件及任何必備軟體。自訂 OpenSSL make 檔，及建置用於您的目標平台的 OpenSSL 程式庫。在 Windows and Linux 上，make 也會建置 OpenSSL 金鑰建立及管理工具。

程序


1. 安裝 OpenSSL 套件。
 - a) 從 [OpenSSL](#) 下載 OpenSSL 套件

重要: 下載及重新配送 OpenSSL 套件必須遵守嚴格的匯入及匯出法規，以及開放程式碼授權條件。在決定是否下載套件之前，請仔細注意限制及警告。
 - b) 將壓縮檔內容展開到 *sdkroot*。

在 OpenSSL 網站的 **News** 標籤下查看，以尋找最新套件的下載位置。套件壓縮為 tar 檔案，副檔名為 tar.gz。展開時，套件會建立頂層資料夾 *opensslversion*；例如 *openssl-1.0.1c*。範例將資料夾的路徑稱為 %openssl% (在 Windows 上) 和 \$openssl (在 iOS 上)；例如，在 Windows 上，%openssl% 是 *sdkroot\openssl-1.0.1c*。

提示: 檢查透過解壓縮 OpenSSL 套件建立的目錄路徑。部分套件具有重複的 *opensslversion* 資料夾層次。
2.  Windows

選擇性的: 在 Windows 上，下載並安裝 Perl。請參閱 [perl.org](#)。

對於此範例，從 [ActivePerl Downloads](#) 下載 Perl。
3.  iOS

選擇性的: 在 iOS 上，再建立三個目錄。

```
$ssarm7 = $openssl/arm7
$sslarm7s = $openssl/arm7s
$ssli386 = $openssl/i386
```

對於 iOS，您必須為三個不同的硬體平台建置 OpenSSL 套件。

4. 產生 OpenSSL make 檔，以為您的硬體及作業系統建置 OpenSSL 套件。

- 在 %openssl% 或 \$openssl 目錄中開啟指令視窗。
- 以適當的參數執行 **Configure** Perl 指令。

Windows

```
perl Configure VC-WIN32 enable-capieng no-asm no-idea no-mdc2 no-rc5 --prefix=%openssl%
ms\do_ms.bat
```

iOS

```
./Configure BSD-generic32 no-idea no-mdc2 no-rc5 --prefix=$openssl
```

5. iOS

在 iOS 上，為不同的 Apple 裝置自訂產生的 OpenSSL make 檔。

- 複製三份所產生 make 檔 \$openssl/Makefile 的副本

```
$openssl/Makefile_armv7
$openssl/Makefile_armv7s
$openssl/Makefile_i386
```

- 變更每一個 make 檔中的 "CC=gcc" 陳述式。

CC=gcc 陳述式在第 62 行左右。請將它變更為下列指令：

\$openssl/Makefile_armv7

```
CC=/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/
Developer/usr/bin/gcc -arch armv7
```

\$openssl/Makefile_armv7s

```
CC=/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/
Developer/usr/bin/gcc -arch armv7s
```

\$openssl/Makefile_i386

```
CC=/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/
Developer/usr/bin/gcc -arch i386
```

- 變更每一個 make 檔中的 "CFLAG=..." 陳述式。

陳述式在第 63 行左右（分為三行以便於閱讀）：

```
CFLAG= -DOPENSSL_THREADS -pthread -D_THREAD_SAFE
-D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -DTERMIOS
-O3 -fomit-frame-pointer -Wall
```

所顯示的 SDK 位置視您的 Xcode 安裝選項而定。SDK 的版本視您要為其建置 make 檔的作業系統層次而定。

iPhone 模擬器

iPhone 模擬器 make 檔是 \$openssl/Makefile_i386。

```
CFLAG= -isysroot /Applications/Xcode.app/Contents/Developer/Platforms/
iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator6.0.sdk
-DOPENSSL_THREADS -pthread -D_THREAD_SAFE
-D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -DTERMIOS
-O3 -fomit-frame-pointer -Wall
```

iOS

iOS make 檔是 \$openssl/Makefile_arm7 及 \$openssl/Makefile_arm7s。

```
CFLAG= -isysroot /Applications/Xcode.app/Contents/Developer/Platforms/  
iPhoneOS.platform/Developer/SDKs/iPhoneOS6.0.sdk  
-DOPENSSL_THREADS -pthread -D_THREAD_SAFE  
-D_REENTRANT -D_DSO_DLFCN -DHAVE_DLFCN_H -DTERMIOS  
-O3 -fomit-frame-pointer -Wall
```

6. 執行產生的 make 檔。

Windows

```
nmake -clean  
nmake -f ms\nt.mak  
nmake -f ms\nt.mak install
```

iOS 在 iOS 上:

```
make clean  
make -f $openssl/Makefile_arm7  
mv $openssl/libcrypto.a $ssarm7/libcrypto.a  
mv $openssl/libssl.a $ssarm7/libssl.a  
make clean  
make -f $openssl/Makefile_arm7s  
mv $openssl/libcrypto.a $ssarm7s/libcrypto.a  
mv $openssl/libssl.a $ssarm7s/libssl.a  
make clean  
make -f $openssl/Makefile_i386  
mv $openssl/libcrypto.a $ssli386/libcrypto.a  
mv $openssl/libssl.a $ssli386/libssl.a
```

結果

建置會產生建置安全的適用於 C 的 MQTT 用戶端程式庫版本所需的共用程式庫、程式庫及標頭檔。

在 iOS 上開始使用適用於 C 的 MQTT 用戶端

學習如何取得 iOS 應用程式，以與 MQTT 伺服器交換訊息。若要在 iOS 裝置（即 iPhone 與 iPad）上使用，您必須從作為 MQTT 軟體開發套件一部分提供的原始碼，建置適用於 C 的 MQTT 用戶端程式庫。

開始之前

1. 鏈結至 [iOS Dev Center](#)，並知道如何開發 iOS 的應用程式。
2. 取得 Apple Mac OS X 10.8.2 或更新版本，以執行 Xcode 整合開發環境 (IDE)。
3. (選用) 在 Windows 或 Linux 上配置 C 開發環境。開發 MQTT iOS 應用程式之前，在 Windows 或 Linux 上建置及執行 MQTT 用戶端範例 C 應用程式將會很有用。此外，也可以研究範例原始碼，而不建置範例。
4. 如需瞭解適用於 C 的受支援及參照 MQTT 用戶端平台，請參閱 [IBM Mobile Messaging and M2M 用戶端套件的系統需求](#)。
5. 如果用戶端與伺服器之間有防火牆，請確認它不會封鎖 MQTT 資料流量。

關於這項作業

指引您執行下列步驟的程序：

1. 透過研究、建置及執行 MQTT 用戶端範例應用程式以及適用於 C 的 MQTT 用戶端程式庫，來瞭解 MQTT 的程式設計。
2. 在 Apple Mac 上安裝用於 iOS 的 Xcode 開發環境。
3. 執行作業 [第 28 頁的『建置適用於 C 的 MQTT 用戶端程式庫』](#)，以建置適用於 iOS 裝置的 MQTT 用戶端 C 程式庫。

程序

1. 選擇您可以連接用戶端應用程式的 MQTT 伺服器。

伺服器必須支援 MQTT version 3.1 通訊協定。IBM 中的所有 MQTT 伺服器都會這麼做，包括 IBM WebSphere MQ 和 IBM MessageSight。請參閱 [第 122 頁的『開始使用 MQTT 伺服器』](#)。

2. 下載 Mobile Messaging and M2M 用戶端套件 並安裝 MQTT SDK。

沒有安裝程式，您只是展開了下載的檔案。

- a. 下載 [Mobile Messaging and M2M 用戶端套件](#)。
- b. 建立您要安裝 SDK 的資料夾。

您可能想將資料夾命名為 MQTT。在這裡，此資料夾的路徑稱為 *sdkroot*。

- c. 將壓縮的 Mobile Messaging and M2M 用戶端套件 檔案內容展開到 *sdkroot* 中。展開會建立以 *sdkroot*\SDK 開頭的目錄樹。

3. 選擇性的: 透過研究 MQTT 用戶端範例 C 應用程式來熟悉 MQTT API。

- a) 針對 Windows 或 Linux 建置同步 MQTT 用戶端範例 C 應用程式 MQTTV3sample.c。請參閱 [第 23 頁的『開始使用適用於 C 的 MQTT 用戶端』](#)。

- b) 連接至 MQTT version 3 伺服器，然後發佈和訂閱伺服器上的主題。

- c) 研究原始碼和 MQTT API 文件。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](#)。

透過研究同步的範例，來瞭解如何建立及回復 MQTT 用戶端，及發佈和訂閱 MQTT 主題。同步範例比非同步範例更簡單。如果您之前未對 MQTT 進程式設計，請撰寫同步的 MQTT 程式以熟悉 MQTT 程式設計模型及 API。

- d) 在 Windows 或 Linux 上建置適用於 C 的非同步 MQTT 用戶端程式庫。請參閱 [第 28 頁的『建置適用於 C 的 MQTT 用戶端程式庫』](#)。

- e) 建置及執行非同步 MQTT 用戶端範例發佈及訂閱 C 應用程式。

- f) 研究 MQTT 用戶端範例非同步 C 應用程式原始碼及 MQTT 參考文件。

您必須使用非同步介面，來撰寫用於行動式裝置的 MQTT 應用程式。與寫入同步介面的應用程式相比，可呼叫非同步介面的撰寫良好的應用程式更具回應力，且可進一步延長電池壽命。

非同步介面具有兩種非同步程度：

- i) 第一種程度是在 MQTT 用戶端程式庫等待來自伺服器的發佈時解除封鎖應用程式。
- ii) 第二種程度是在用戶端程式庫連接至伺服器、建立訂閱及公佈發佈時解除封鎖應用程式。

4. 下載並安裝 iOS 開發工具。

- a. 使用具有管理專用權的使用者 ID 登入。
- b. 檢查您的 Apple Mac 是否為 10.8.2 版或更新版本。
- c. 移至網站 [Xcode](#)，以從 Mac App Store 下載 Xcode。
- d. 安裝 Xcode、指令行環境及模擬器。

如果 Mac 應用程式商店提供了多種模擬器版本，請選擇與您為應用程式所設定目標的 iOS 層次相容的版本。

5. 在 iOS 上，建置適用於 C 的 MQTT 用戶端程式庫。請參閱 [第 28 頁的『建置適用於 C 的 MQTT 用戶端程式庫』](#)。

下一步

1. 驗證您建置的適用於 C 的 MQTT 用戶端程式庫：

- a. 使用 Xcode 開發環境以編譯非同步 MQTT 用戶端範例 C 應用程式，並鏈結至未受保護的適用於 C 的非同步 MQTT 用戶端程式庫。
- b. 使用 Xcode 開發環境，在 iOS 裝置上執行非同步 MQTT 用戶端範例 C 應用程式。將範例連接至您配置的 MQTT version 3 伺服器；請參閱 [第 126 頁的『從指令行配置 MQTT 服務』](#)。

2. 針對 iOS 建立 MQTT 用戶端 C 應用程式。非同步 C 應用程式的程式碼範例可能很有幫助。範例是 `sdkroot\SDK\clients\c\samples` 中的 `MQTTV3ASample.c` 和 `MQTTV3ASSample.c`。作為練習，請先開始實作 MQTT 發佈/訂閱範例。

提示: 如需瞭解應用程式的外觀及其用途，請查看 MQTT 用戶端範例 C 應用程式的畫面擷取。請參閱 [第 16 頁的『在 Android 上開始使用適用於 Java 的 MQTT 用戶端』](#)。

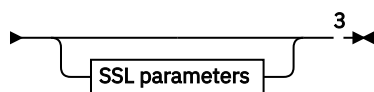
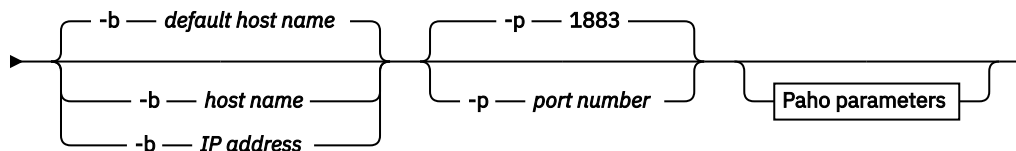
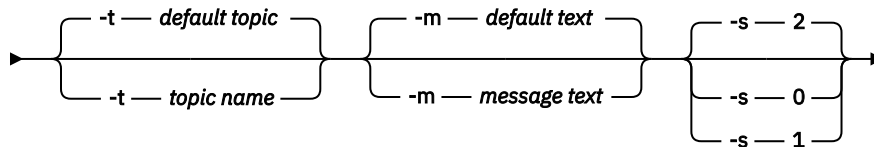
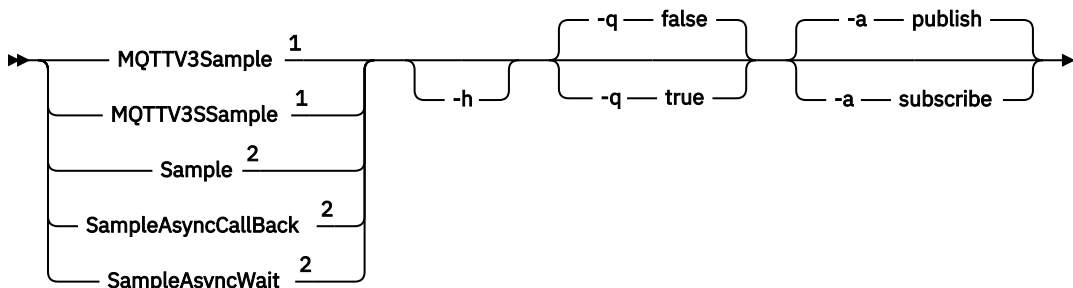
MQTT 指令行程式範例

MQTT 指令行程式範例的語法及參數。

用途

發佈和訂閱主題。

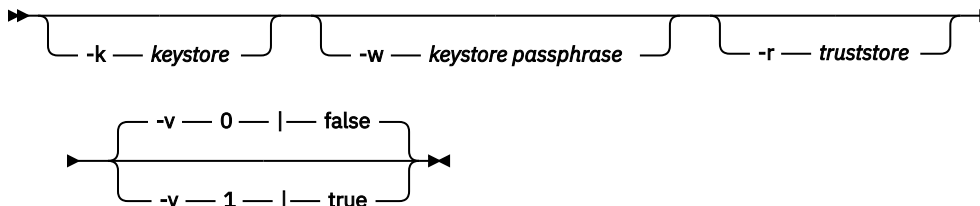
Syntax



Paho parameters



SSL parameters



註:

¹ IBM WebSphere MQ sample

² Paho sample

³ Not MQTTV3Sample.

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 `publish|subscribe`

將動作設為 `publish` 或 `subscribe`，而不是假設預設發佈動作。

-t `topic name`

發佈或訂閱 `topic name`，而不是發佈或訂閱預設主題。預設主題如下所示：

Paho 範例

發佈

Sample/Java/v3

訂閱

Sample/#

IBM WebSphere MQ 範例

發佈

MQTTV3Sample/Java/v3 或 MQTTV3Sample/C/v3

訂閱

MQTTV3Sample/#

-m 訊息文字

發佈 `message text`，而不是傳送預設文字。預設文字為 "Message from MQTTv3 C client" 或 "Message from MQTTv3 Java client"

-s `0|1|2`

設定服務品質 (QoS)，而不是使用預設 QoS 2。

-b 主機名稱

連接至 `host name` 或 IP 位址，而不是連接至預設主機名稱。Paho 範例的預設主機名稱為 `m2m.eclipse.org`。對於 IBM WebSphere MQ 範例，它是 `localhost`。

-p `port number`

使用埠 `port number`，而不是使用預設埠 1883。

Paho 參數

-i `client identifier`

將用戶端 ID 設為 `client identifier`。預設用戶端 ID 為 `SampleJavaV3_`+`action`，其中 `action` 是 `publish` 或 `subscribe`。

-c `true|false`

設定清除階段作業旗標。預設值為 `true`：訂閱不可延續。

SSL 參數

-k `keystore`

將金鑰儲存庫 (包含可識別用戶端的私密金鑰) 的路徑設為 `keystore`。對於 C 範例，儲存庫為「保密加強型郵件」(PEM) 檔案。對於 Java 範例，其為 Java 金鑰儲存庫 (JKS)。

-w `keystore passphrase`

將授權用戶端存取金鑰儲存庫的通行詞組設為 `keystore passphrase`。

-r truststore

將包含用戶端信任之 MQTT 伺服器公開金鑰的金鑰儲存庫路徑設為 `truststore`。金鑰儲存庫是保密加強型郵件 (PEM) 檔案。對於 C 範例，儲存庫為「保密加強型郵件」(PEM) 檔案。對於 Java 範例，其為 Java 金鑰儲存庫 (JKS)。

-v 0|false|1>true

將驗證選項設為 `1|true` 以需要伺服器憑證。預設值為 `0|false`：不檢查伺服器憑證。SSL 通道一律已加密。

將選項設為 `0|1` (若為 C 程式) 及 `true|false` (若為 Java 程式)。

相關工作

[第 11 頁的『開始使用適用於 Java 的 MQTT 用戶端』](#)

使用 IBM MessageSight 或 IBM WebSphere MQ 作為 MQTT 伺服器，建立並執行 Java 範例應用程式的 MQTT 用戶端。範例應用程式使用來自 IBM 的 MQTT 軟體開發工具箱 (SDK) 的用戶端程式庫。SampleAsyncCallback 範例應用程式是一個模型，用於撰寫適用於 Android 及其他事件驅動作業系統的 MQTT 應用程式。

[第 23 頁的『開始使用適用於 C 的 MQTT 用戶端』](#)

您可以在其中編譯 C 原始檔的任何平台上，建立並執行適用於 C 的 MQTT 用戶端範例。驗證您可以使用 IBM MessageSight 或 IBM WebSphere MQ 作為 MQTT 伺服器來執行適用於 C 的範例 MQTT 用戶端。

[第 28 頁的『建置適用於 C 的 MQTT 用戶端程式庫』](#)

遵循下列步驟，以建置適用於 C 的 MQTT 用戶端程式庫。本主題包括用於許多平台的編譯及鏈結開關，以及在 iOS 和 Windows 上建置程式庫的範例。

MQTT 安全

以下是 MQTT 安全的三個基本概念：身分、鑑別及授權。身分可為要授權及提供權限的用戶端命名。鑑別可提供用戶端的身分，而授權可管理為用戶端提供的權限。

嘗試安全範例

- [第 50 頁的『建置及執行安全的 MQTT 用戶端範例 Java 應用程式』](#)
- [第 58 頁的『透過 SSL 連接 Android 上的 MQTT 用戶端範例 Java 應用程式』](#)
- [第 67 頁的『向 JAAS 鑑別 MQTT 用戶端 Java 應用程式』](#)
- **V7.5.0.1** [第 71 頁的『透過 SSL 及 WebSockets 連接 適用於 JavaScript 的 MQTT 傳訊用戶端』](#)
- [第 79 頁的『建置及執行安全的 MQTT 用戶端範例 C 應用程式』](#)

身分

透過用戶端 ID、使用者 ID 或公用數位憑證來識別 MQTT 用戶端。這些屬性中有一項定義了用戶端身分。MQTT 伺服器使用 SSL 通訊協定鑑別用戶端傳送的憑證，或使用用戶端設定的密碼鑑別用戶端身分。伺服器根據用戶端身分控制用戶端可以存取的資源。

MQTT 伺服器使用 IP 位址及數位憑證自行識別用戶端。MQTT 用戶端使用 SSL 通訊協定鑑別伺服器傳送的憑證。在某些情況下，它使用伺服器的 DNS 名稱來驗證為其傳送憑證的伺服器登錄為憑證持有者。

請按下列其中一種方式設定用戶端的身分：

用戶端 ID

MqttClient 類別 (C 中的 MqttClient_create 或 MqttAsync_create) 可設定用戶端 ID。呼叫類別建構子，將用戶端 ID 設為參數，或傳回隨機產生的用戶端 ID。用戶端 ID 在所有連接至伺服器的用戶端中必須是唯一的，而且不得與伺服器上的佇列管理程式同名。所有用戶端都必須具有用戶端 ID，即使它未用於身分檢查。請參閱 [第 116 頁的『用戶端 ID』](#)。

使用者 ID

MqttClient 類別 (C 中的 MqttClient_create 或 MqttAsync_create) 可將用戶端使用者 ID 設為 MqttConnectOptions 的屬性 (C 中的 MqttClient_ConnectOptions)。使用者 ID 不需要是用戶端唯一的。

用戶端數位憑證

用戶端數位憑證儲存在用戶端金鑰儲存庫中。金鑰儲存庫位置視用戶端而定：

- **Java**

透過呼叫 `MqttConnectOptions` 的 `setSSLProperties` 方法並傳遞金鑰儲存庫內容，來設定用戶端金鑰儲存庫的位置及內容。請參閱對 `Example.java` 的 [SSL 修改](#)。**keytool** 工具可管理 Java 金鑰和金鑰儲存庫。

- **C**

`MQTTClient_create` 或 `MQTTAsync_create` 會將金鑰儲存庫內容設為 `MQTTClient_SSLOptions ssl_opts` 的屬性。**openSSL** 工具會建立並管理 MQTT Client for C 所存取的金鑰和金鑰儲存庫。

- **Android**

從設定 > 安全功能表，管理 Android 裝置金鑰儲存庫。從 SD 卡載入新憑證。

將伺服器的私密金鑰儲存在伺服器金鑰儲存庫中，以設定伺服器的身分：

IBM WebSphere MQ

MQTT 伺服器金鑰儲存庫是用戶端所連接遙測通道的屬性。

使用 IBM WebSphere MQ Explorer 或 **DEFINE CHANNEL** 指令來設定金鑰儲存庫位置及屬性；請參閱 [DEFINE CHANNEL \(MQTT\)](#)。多個通道可以共用金鑰儲存庫。

鑑別

MQTT 用戶端可以鑑別其所連接之 MQTT 伺服器，且該伺服器可以鑑別它所連接的用戶端。

用戶端使用 SSL 通訊協定鑑別伺服器。MQTT 伺服器使用 SSL 通訊協定及/或密碼鑑別用戶端。

如果用戶端鑑別伺服器，但伺服器未鑑別用戶端，用戶端通常稱為匿名用戶端。通常透過 SSL 建立匿名用戶端連線，然後使用 SSL 階段作業加密的密碼鑑別用戶端。由於憑證公佈及管理問題，使用密碼（而非用戶端憑證）鑑別用戶端更為普遍。您可能在高價值裝置（如 ATM 及晶片銀行卡機）及在自訂裝置（智慧型電表）中尋找用戶端憑證。

透過用戶端鑑別伺服器

MQTT 用戶端透過使用 SSL 通訊協定鑑別伺服器憑證，來驗證連接至正確的伺服器。當您透過 HTTPS 通訊協定瀏覽網站時，此驗證表單對您來說很熟悉。

伺服器可將公用憑證（由憑證管理中心簽署）傳送至用戶端。用戶端使用憑證管理中心的公開金鑰，來驗證伺服器憑證上的憑證管理中心的簽章。它也會檢查憑證是否是最新的。這些檢查會確定憑證是否有效。

憑證管理中心憑證（通常稱為主要憑證）儲存在用戶端信任儲存庫中：

- **Java**

呼叫 `MqttConnectOptions` 的 `setSSLProperties` 方法並傳遞信任儲存庫內容，以設定用戶端信任儲存庫的位置及內容。請參閱對 `Example.java` 的 [SSL 修改](#)。使用 **keytool** 工具管理憑證及信任儲存庫。

- **C**

`MQTTClient_create` 或 `MQTTAsync_create` 將信任儲存庫內容設為 `MQTTClient_SSLOptions ssl_opts` 的屬性。使用 **openSSL** 工具管理憑證及信任儲存庫。

- **Android**

從設定 > 安全功能表，管理 Android 裝置信任儲存庫。從 SD 卡載入新的主要憑證。

透過伺服器鑑別用戶端

MQTT 伺服器透過使用 SSL 通訊協定鑑別用戶端憑證，或使用密碼鑑別用戶端身分，來驗證連接至正確的用戶端。

它使用 MQTT protocol 標頭中用戶端傳送至伺服器的密碼，來鑑別用戶端。伺服器可以選擇使用密碼鑑別用戶端 ID、使用者 ID 或憑證。它視伺服器而定。通常，伺服器鑑別使用者 ID。透過 SSL 連線（驗證伺服器來保護安全）驗證密碼，以避免用明碼傳送密碼。

• IBM WebSphere MQ

IBM WebSphere MQ 使用 SSL 通訊協定鑑別用戶端憑證。將主要憑證儲存在 IBM WebSphere MQ Telemetry 金鑰儲存庫中。您只能透過相互 SSL 鑑別來鑑別用戶端憑證。亦即，您必須為用戶端提供伺服器公用憑證，以及為伺服器提供用戶端公用憑證。

對於它自己的專用和公用憑證，及其他公用憑證（例如，憑證管理中心提供的主要憑證），IBM WebSphere MQ Telemetry 使用相同的儲存庫。

使用 IBM WebSphere MQ Explorer 或 **DEFINE CHANNEL** 指令來設定金鑰儲存庫位置及屬性；請參閱 [DEFINE CHANNEL \(MQTT\)](#)。多個通道可以共用金鑰儲存庫。

IBM WebSphere MQ 透過呼叫 Java 鑑別及授權服務 (JAAS)，來鑑別用戶端使用者 ID 或用戶端 ID。

在儲存於 `jaas.config` 檔案的 `MQXRConfig` 配置段落中配置 JAAS。該檔案儲存在 IBM WebSphere MQ 資料路徑的 `qmgrs\QmgrName\mqxr` 目錄中。

透過為 `JAASLoginModule` 撰寫 `login` 方法來檢查用戶端的確實性。請參閱第 105 頁的『[遙測通道 JAAS 配置](#)』。

IBM WebSphere MQ Telemetry 會傳遞下列參數給 `JAASLoginModule.login` 方法：

- 使用者 ID
- 密碼
- 用戶端 ID
- 網路 ID
- 通道名稱
- ValidPrompts

授權

授權不是 MQTT protocol 的一部分。它由 MQTT 伺服器提供。授權的內容視伺服器執行的作業而定。MQTT 伺服器是發佈/訂閱分配管理系統，且有用的 MQTT 授權規則可控制哪些用戶端可以連接至伺服器，及用戶端可以發佈或訂閱哪些主題。如果 MQTT 用戶端可以管理伺服器，則更多授權規則控制哪些用戶端可以管理伺服器的不同方面。

可用的用戶端非常多，因此分別授權每一個用戶端是不可行的。MQTT 伺服器將具有按設定檔或群組將用戶端分組的方法。

從存取及權限的觀點來看，用戶端的身分在某些方面對於 MQTT 用戶端不是唯一的。用戶端的身分不等於用戶端 ID。它們可能相同，但通常是不同的。例如，您可能有一個在許多服務之間通用的使用者名稱，其中有些服務會在 "單一登入" 中合作。企業級 MQTT 伺服器可能會呼叫授權服務，從而為不同的應用程式提供共用身分及權限。

IBM WebSphere MQ

IBM WebSphere MQ 具有外掛授權服務。Windows 及 Linux 上提供的預設授權服務是物件權限管理程式 (OAM)。請參閱在 UNIX、Linux 及 Windows 系統上使用 OAM 來控制對物件的存取權。它會將作業系統使用者 ID 及群組與 IBM WebSphere MQ 物件上的作業（例如，主題和佇列）相關聯。

您可以配置遙測通道，以使用固定的使用者 ID 來存取 IBM WebSphere MQ。這是設定範例通道的方式。或者，您可以使用 MQTT 用戶端所設定的使用者 ID 來存取 IBM WebSphere MQ。授權 MQTT 用戶端存取 WebSphere MQ 物件中說明了如何設定 IBM WebSphere MQ Telemetry，以實現粗略、中等及精細用戶端存取控制的方法。

相關工作

第 79 頁的『[建置及執行安全的 MQTT 用戶端範例 C 應用程式](#)』

您可以根據 Windows 範例，在可為其編譯 C 原始檔的任何作業系統上，建立及執行安全的範例 C 應用程式。請驗證您可以在 IBM MessageSight 或 IBM WebSphere MQ 作為 MQTT 伺服器上執行範例 C 應用程式。

[第 50 頁的『建置及執行安全的 MQTT 用戶端範例 Java 應用程式』](#)

根據 Windows 範例，您可以在 IBM MessageSight 或 IBM WebSphere MQ 作為 MQTT 伺服器上啟動並執行安全範例 Java 應用程式。您可以在任何具有 JSE 1.5 或更高版本 (即 "Java 相容") 的平台上執行適用於 Java 的 MQTT 用戶端應用程式

[第 71 頁的『透過 SSL 及 WebSockets 連接 適用於 JavaScript 的 MQTT 傳訊用戶端』](#)

使用適用於 JavaScript 的 MQTT 傳訊用戶端 範例 HTML 頁面搭配 SSL 及 WebSocket protocol，將 Web 應用程式安全地連接至 IBM WebSphere MQ。

[第 58 頁的『透過 SSL 連接 Android 上的 MQTT 用戶端範例 Java 應用程式』](#)

開始進行透過 SSL 連接至 IBM WebSphere MQ 的範例 Android MQTT 用戶端。

[第 67 頁的『向 JAAS 鑑別 MQTT 用戶端 Java 應用程式』](#)

學習如何使用 JAAS 鑑別用戶端。完成此作業中的步驟，以修改範例程式 JAASLoginModule.java，並配置 IBM WebSphere MQ 以向 JAAS 鑑別 MQTT 用戶端 Java 應用程式。

建置及執行安全的 MQTT 用戶端範例 Java 應用程式

根據 Windows 範例，您可以在 IBM MessageSight 或 IBM WebSphere MQ 作為 MQTT 伺服器上啟動並執行安全範例 Java 應用程式。您可以在任何具有 JSE 1.5 或更高版本 (即 "Java 相容") 的平台上執行適用於 Java 的 MQTT 用戶端應用程式

開始之前

1. 您必須能夠存取支援 MQTT protocol over SSL 的 MQTT version 3.1 伺服器。
2. 如果用戶端與伺服器之間有防火牆，請確認它不會封鎖 MQTT 資料流量。
3. 您可以在任何具有 JSE 1.5 或更高版本 (即 "Java 相容") 的平台上執行適用於 Java 的 MQTT 用戶端應用程式。請參閱 [IBM Mobile Messaging and M2M 用戶端套件的系統需求](#)。
4. SSL 通道必須已啟動。

關於這項作業

作為圖解，本文顯示如何從指令行編譯及執行 Windows 上的安全 MQTT 用戶端範例 Java 應用程式。使用憑證管理中心簽署的金鑰或自簽金鑰保護 SSL 通道。

程序

1. 選擇您可以連接用戶端應用程式的 MQTT 伺服器。

伺服器必須支援透過 SSL 的 MQTT version 3.1 通訊協定。IBM 中的所有 MQTT 伺服器都會這麼做，包括 IBM WebSphere MQ 和 IBM MessageSight。請參閱 [第 122 頁的『開始使用 MQTT 伺服器』](#)。
2. 選擇性的: 安裝第 7 版或更新版本的 Java 開發套件 (JDK)。

第 7 版，才能執行 **keytool** 指令來認證憑證。如果您不打算對憑證進行認證，則不需要第 7 版 JDK。
3. 下載 Mobile Messaging and M2M 用戶端套件 並安裝 MQTT SDK。

沒有安裝程式，您只是展開了下載的檔案。

 - a. 下載 [Mobile Messaging and M2M 用戶端套件](#)。
 - b. 建立您要安裝 SDK 的資料夾。

您可能想將資料夾命名為 MQTT。在這裡，此資料夾的路徑稱為 *sdkroot*。
 - c. 將壓縮的 Mobile Messaging and M2M 用戶端套件 檔案內容展開到 *sdkroot* 中。展開會建立以 *sdkroot\SDK* 開頭的目錄樹。
4. 建立並執行 Script 以產生金鑰組和憑證，並將 IBM WebSphere MQ 配置為 MQTT 伺服器。

遵循第 88 頁的『產生金鑰及憑證』中的步驟來建立並執行 Script。第 52 頁的『為 Windows 配置 SSL 憑證的 Script 範例』中也列有這些 Script。

5. 檢查 SSL 通道是否正在執行，且如您的預期設定。

在 IBM WebSphere MQ 上，於指令視窗中鍵入下列指令：

• **Linux**

```
echo 'DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
echo 'DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
```

• **Windows**

```
echo DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM
echo DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM
```

6. 建立 Script 以建置及執行安全的 MQTT 用戶端範例 Java 應用程式。
 - a) 建立及執行 `ssjavaclient.bat`，以測試受到自簽憑證保護的 SSL 通道。
 - b) 建立及執行 `cajavaclient.bat`，以測試受到憑證管理中心簽署憑證保護的 SSL 通道。

執行 MQTT 安全 Java 用戶端的 Script

先執行第 52 頁的『為 Windows 配置 SSL 憑證的 Script 範例』中的 Script，然後再執行這些 Script。

具有自簽憑證的 MQTT 安全 Java 用戶端。

使用您利用執行 `sscerts.bat` Script 所建立的自簽憑證來執行此 Script。

```
@echo off
setlocal
cd %jsamppath%
set classpath=
set JAVADIR=C:\Program Files\IBM\Java70\bin
cd %
"%JAVADIR%\javac"
-cp ..\org.eclipse.paho.client.mqttv3.jar .\org\ eclipse\paho\sample\mqttv3app\Sample.java
ping -n 2 127.0.0.1 > NUL 2>&1
start "Sample Subscriber" "%JAVADIR%\java" -cp .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -a subscribe -b %host% -p %sslportopt% -k
%cltjkskeystore% -w %cltjkskeystorepass% -r %cltsrvjkstruststore% -v true
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
"%JAVADIR%\java" -cp .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -b %host% -p %sslportopt% -k %cltjkskeystore% -w
%cltjkskeystorepass% -r %cltsrvjkstruststore% -v true
pause
ping -n 2 127.0.0.1 > NUL 2>&1
start "Sample Subscriber" "%JAVADIR%\java" -cp .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -a subscribe -b %host% -p %sslportreq% -k
%cltjkskeystore% -w %cltjkskeystorepass% -r %cltsrvjkstruststore% -v true
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
"%JAVADIR%\java" -cp .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -b %host% -p %sslportreq% -k %cltjkskeystore% -w
%cltjkskeystorepass% -r %cltsrvjkstruststore% -v true
pause
endlocal
```

圖 14: `ssjavaclient.bat`

利用憑證管理中心簽署的憑證，執行 MQTT 安全 Java 用戶端。

使用您利用執行 `cacerts.bat` Script 所建立之憑證管理中心簽署的憑證來執行此 Script。

```

@echo off
setlocal
cd %jsamppath%
set classpath=
set JAVADIR=C:\Program Files\IBM\Java70\bin
cd %
"%JAVADIR%\javac"
-cp ..\org.eclipse.paho.client.mqttv3.jar .\org\ eclipse\paho\sample\mqttv3app\Sample.java
ping -n 2 127.0.0.1 > NUL 2>&1
start "Sample Subscriber" "%JAVADIR%\java" -cp .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -a subscribe -b %host% -p %sslportopt% -k
%cltjkskeystore% -w %cltjkskeystorepass% -r %cltcajkstruststore% -v true
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
"%JAVADIR%\java" -cp .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -b %host% -p %sslportopt% -k %cltjkskeystore% -w
%cltjkskeystorepass% -r %cltcajkstruststore% -v true
pause
ping -n 2 127.0.0.1 > NUL 2>&1
start "Sample Subscriber" "%JAVADIR%\java" -cp .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -a subscribe -b %host% -p %sslportreq% -k
%cltjkskeystore% -w %cltjkskeystorepass% -r %cltcajkstruststore% -v true
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
"%JAVADIR%\java" -cp .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -b %host% -p %sslportreq% -k %cltjkskeystore% -w
%cltjkskeystorepass% -r %cltcajkstruststore% -v true
pause
endlocal

```

圖 15: *cajavaclient.bat*

相關概念

第 47 頁的『MQTT 安全』

以下是 MQTT 安全的三個基本概念：身分、鑑別及授權。身分可為要授權及提供權限的用戶端命名。鑑別可提供用戶端的身分，而授權可管理為用戶端提供的權限。

相關工作

第 88 頁的『產生金鑰及憑證』

遵循下列程序，為 Java 和 C 用戶端（包括 Android 及 iOS 應用程式）以及 IBM WebSphere MQ 和 IBM MessageSight 伺服器產生金鑰及憑證。

第 58 頁的『透過 SSL 連接 Android 上的 MQTT 用戶端範例 Java 應用程式』

開始進行透過 SSL 連接至 IBM WebSphere MQ 的範例 Android MQTT 用戶端。

第 67 頁的『向 JAAS 鑑別 MQTT 用戶端 Java 應用程式』

學習如何使用 JAAS 鑑別用戶端。完成此作業中的步驟，以修改範例程式 JAASLoginModule.java，並配置 IBM WebSphere MQ 以向 JAAS 鑑別 MQTT 用戶端 Java 應用程式。

為 Windows 配置 SSL 憑證的 Script 範例

指令檔範例會建立憑證及憑證儲存庫，如作業中的步驟所述。此外，範例還會將 MQTT 用戶端佇列管理程式設為使用伺服器憑證儲存庫。範例會呼叫 IBM WebSphere MQ 隨附的 SampleMQM.bat Script，來刪除並重建佇列管理程式。

initcert.bat

initcert.bat 會設定 **keytool** 及 **openssl** 指令所需之憑證及其他參數的名稱和路徑。Script 中的註解說明了這些設定。

```

@echo off
@rem Set the path where you installed the MQTT SDK
@rem and short cuts to the samples directories.
set SDKRoot=C:\MQTT
set jsamppath=%SDKRoot%\sdk\clients\java\samples
set csamppath=%SDKRoot%\sdk\clients\c\samples

```

```

@rem Set the paths to Version 7 of the JDK
@rem and to the directory where you built the openssl package.

```

```
@rem Set short cuts to the tools.
set javapath=C:\Program Files\IBM\Java70
set keytool="%javapath%\jre\bin\keytool.exe"
set ikeyman="%javapath%\jre\bin\ikeyman.exe"
set openssl=%SDKRoot%\openssl
set runopenssl="%openssl%\bin\openssl"
```

```
@rem Set the path to where certificates are to be stored,
@rem and set global security parameters.
@rem Omit set password, and the security tools prompt you for passwords.
@rem Validity is the expiry time of the certificates in days.
set certpath=%SDKRoot%\Certificates
set password=password
set validity=5000
set algorithm=RSA
```

```
@rem Set the certificate authority (CA) jks keystore and certificate parameters.
@rem Omit this step, unless you are defining your own certificate authority.
@rem The CA keystore contains the key-pair for your own certificate authority.
@rem You must protect the CA keystore.
@rem The CA certificate is the self-signed certificate authority public certificate.
@rem It is commonly known as the CA root certificate.
set caalias=caalias
set cadname="CN=mqttca.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cakeypass=%password%
@rem ca key store
set cajkskeystore=%certpath%\cakeystore.jks
set cajkskeystorepass=%password%
@rem ca certificate (root certificate)
set cacert=%certpath%\cacert.cer
```

```
@rem Set the server jks keystore and certificate parameters.
@rem The server keystore contains the key-pair for the server.
@rem You must protect the server keystore.
@rem If you then export the server certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the server keystore for the server key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the server certificate to the CA.
@rem When you now export the server certificate,
@rem the exported certificate includes the certificate chain.
set srvalias=srvalias
set srvdname="CN=mqttserver.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set srvkeypass=%password%
@rem server key stores
set srvjkskeystore=%certpath%\srvkeystore.jks
set srvjkskeystorepass=%password%
@rem server certificates
set srvcertreq=%certpath%\srvcertreq.csr
set srvcertcasigned=%certpath%\srvcertcasigned.cer
set srvcertselfsigned=%certpath%\srvcertselfsigned.cer
```

```
@rem Set the client jks keystore and certificate parameters
@rem Omit this step, unless you are authenticating clients.
@rem The client keystore contains the key-pair for the client.
@rem You must protect the client keystore.
@rem If you then export the client certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the client keystore for the client key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the client certificate to the CA.
@rem When you now export the client certificate,
@rem the exported certificate includes the certificate chain.
set cltalias=cltalias
set cltdname="CN=mqttclient.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cltkeypass=%password%
@rem client key stores
set cltjkskeystore=%certpath%\cltkeystore.jks
set cltjkskeystorepass=%password%
set cltcertreq=%certpath%\cltcertreq.csr
set cltcertcasigned=%certpath%\cltcertcasigned.cer
set cltcertselfsigned=%certpath%\cltcertselfsigned.cer
```

```
@rem Set the paths to the client truststores signed by CA and signed by server key.
@rem You only need to define one of the trust stores.
```

```

@rem A trust store holds certificates that you trust,
@rem which are used to authenticate untrusted certificates.
@rem In this example, when the client authenticates the MQTT server it connects to,
@rem it authenticates the certificate it is sent by the server
@rem with the certificates in its trust store.
@rem For example, the MQTT server sends its server certificate,
@rem and the client authenticates it with either the same server certificate
@rem that you have stored in the cltsrvtruststore.jks trust store,
@rem or against the CA certificate, if the server certificate is signed by the CA.
set cltcajkstruststore=%certpath%\cltcatruststore.jks
set cltcajkstruststorepass=%password%
set cltsrvjkstruststore=%certpath%\cltsrvtruststore.jks
set cltsrvjkstruststorepass=%password%

```

```

@rem Set the paths to the client PKCS12 and PEM key and trust stores.
@rem Omit this step, unless you are configuring a C or iOS client.
@rem You only need to define either one of the trust stores for storing CA
@rem or server signed server certificates.
set cltp12keystore=%certpath%\cltkeystore.p12
set cltp12keystorepass=%password%
set cltpemkeystore=%certpath%\cltkeystore.pem
set cltpemkeystorepass=%password%
set cltcap12truststore=%certpath%\cltcatruststore.p12
set cltcap12truststorepass=%password%
set cltcapemtruststore=%certpath%\cltcatruststore.pem
set cltcapemtruststorepass=%password%
set cltsrvp12truststore=%certpath%\cltsrvtruststore.p12
set cltsrvp12truststorepass=%password%
set cltsrvpemtruststore=%certpath%\cltsrvtruststore.pem
set cltsrvpemtruststorepass=%password%

```

```

@rem set WMQ Variables
set authopt=NEVER
set authreq=REQUIRED
set qm=MQXR_SAMPLE_QM
set host=localhost
set mcauser='Guest'
set portsslopt=8884
set chlopt=SSLOPT
set portsslreq=8885
set chlreq=SSLREQ
V7.5.0.1 set portws=1886
set chlws=PLAINWS
set chlssloptws=SSLOPTWS
set portssloptws=8886
set chlsslreqws=SSLREQWS
set portsslreqws=8887
set mqlog=%certpath%\wmq.log

```

cleancert.bat

cleancert.bat Script 中的指令會刪除 MQTT 用戶端佇列管理程式以確保伺服器憑證儲存庫未鎖定，然後會刪除範例安全 Script 建立的所有金鑰儲存庫及憑證。

```

@rem Delete the MQTT sample queue manager, MQXR_SAMPLE_QM
call "%MQ_FILE_PATH%\bin\setmqenv" -s
endmqm -i %qm%
dltmqm %qm%

```

```

@rem Erase all the certificates and key stores created by the sample scripts.
erase %cajkskeystore%
erase %cacert%
erase %srvjkskeystore%
erase %svrcertreq%
erase %svrcertcasigned%
erase %svrcertselfsigned%
erase %cltjkskeystore%
erase %cltp12keystore%
erase %cltpemkeystore%
erase %cltcertreq%
erase %cltcertcasigned%
erase %cltcertselfsigned%
erase %cltcajkstruststore%
erase %cltcap12truststore%
erase %cltcapemtruststore%
erase %cltsrvjkstruststore%
erase %cltsrvp12truststore%

```

```

erase %cltsrvpemtruststore%
erase %mqlog%
@echo Cleared all certificates
dir %certpath%\*.* /b

```

genkeys.bat

genkeys.bat Script 中的指令會為您的專用憑證管理中心、伺服器及用戶端建立金鑰組。

```

@rem
@echo -----
@echo Generate %caalias%, %srvalias%, and %cltalias% key-pairs in %cajkskeystore%,
%srvjkskeystore%, and %cltjkskeystore%
@rem
@rem -- Generate a client certificate and a private key pair
@rem Omit this step, unless you are authenticating clients.
%keytool% -genkeypair -noprompt -alias %cltalias% -dname %cltdname% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass% -keypass %cltkeypass% -keyalg %algorithm%
-validity %validity%

```

```

@rem -- Generate a server certificate and private key pair
%keytool% -genkeypair -noprompt -alias %srvalias% -dname %srvdname% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass% -keypass %srvkeypass% -keyalg %algorithm%
-validity %validity%

```

```

@rem Create CA, client and server key-pairs
@rem -- Generate a CA certificate and private key pair - The extension asserts this is a
certificate authority certificate, which is required to import into firefox
%keytool% -genkeypair -noprompt -ext bc=ca:true -alias %caalias% -dname %cadname%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass% -keyalg
%algorithm% -validity %validity%

```

sscerts.bat

sscerts.bat Script 中的指令會從其金鑰儲存庫匯出用戶端及伺服器自簽憑證，然後將伺服器憑證匯入用戶端信任儲存庫，並將用戶端憑證匯入伺服器金鑰儲存庫。伺服器沒有信任儲存庫。這些指令會從用戶端 JKS 信任儲存庫，以 PEM 的格式建立用戶端信任儲存庫。

```

@rem
@echo -----
@echo Export self-signed certificates: %svrcertselfsigned% and %cltcertselfsigned%
@rem Export Server public certificate
%keytool% -exportcert -noprompt -rfc -alias %srvalias% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass% -file %svrcertselfsigned%
@rem Export Client public certificate
@rem Omit this step, unless you are authenticating clients.
%keytool% -exportcert -noprompt -rfc -alias %cltalias% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass% -file %cltcertselfsigned%

```

```

@rem
@echo -----
@echo Add selfsigned server certificate %svrcertselfsigned% to client trust store:
%cltsrvjkstruststore%
@rem Import the server certificate into the client-server trust store (for server self-
signed authentication)
%keytool% -import -noprompt -alias %srvalias% -file %svrcertselfsigned% -keystore
%cltsrvjkstruststore% -storepass %cltsrvjkstruststorepass%

```

```

@rem
@echo -----
@echo Add selfsigned client certificate %cltcertselfsigned% to server trust store:
%srvjkskeystore%
@rem Import the client certificate into the server trust store (for client self-signed
authentication)
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %cltalias% -file %cltcertselfsigned% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%

```

```

@rem
@echo -----
@echo Create a pem client-server trust store from the jks client-server trust store:
%cltsrvpemtruststore%
@rem

```

```

%keytool% -importkeystore -noprompt -srckeystore %cltsrvjkstruststore% -destkeystore
%cltsrvp12truststore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltsrvjkstruststorepass% -deststorepass %cltsrvp12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltsrvp12truststore% -out %cltsrvpemtruststore% -passin
pass:%cltsrvp12truststorepass% -passout pass:%cltsrvpemtruststorepass%@rem
@rem
@echo -----
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore% -destkeystore
%cltp12keystore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass% -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore% -out %cltpemkeystore% -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%

```

cacerts.bat

此 Script 會將憑證管理中心主要憑證匯入專用金鑰儲存庫。需要 CA 主要憑證，才可在主要憑證與已簽章的憑證之間建立金鑰鏈。cacerts.bat Script 會從其金鑰儲存庫匯出用戶端及伺服器憑證申請。Script 會使用 cajkskeystore.jks 金鑰儲存庫中專用憑證管理中心的金鑰，來簽署憑證申請，然後將已簽章的憑證匯回提出申請的同一個金鑰儲存庫。匯入會使用 CA 主要憑證建立憑證鏈。Script 會從用戶端 JKS 信任儲存庫，以 PEM 的格式建立用戶端信任儲存庫。

```

@rem
@echo -----
@echo Export self-signed certificates: %cacert%
@rem
@rem Export CA public certificate
%keytool% -exportcert -noprompt -rfc -alias %caalias% -keystore %cajkskeystore% -storepass
%cajkskeystorepass% -file %cacert%

```

```

@rem
@echo -----
@echo Add CA to server key and client key and trust stores: %srvjkskeystore%,
%cltjkskeystore%, %cltcajkstruststore%,
@rem The CA certificate is necessary to create key chains in the client and server key
stores,
@rem and to certify key chains in the server key store and the client trust store
@rem
@rem Import the CA root certificate into the server key store
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass%
@rem Import the CA root certificate into the client key store
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass%
@rem Import the CA root certificate into the client ca-trust store (for ca chained
authentication)
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltcajkstruststore%
-storepass %cltcajkstruststorepass%

```

```

@rem
@echo -----
@echo Create certificate signing requests: %srvcertreq% and %cltcertreq%
@rem
@rem Create a certificate signing request (CSR) for the server key
%keytool% -certreq -alias %srvalias% -file %srvcertreq% -keypass %srvkeypass% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Create a certificate signing request (CSR) for the client key
%keytool% -certreq -alias %cltalias% -file %cltcertreq% -keypass %cltkeypass% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass%

```

```

@rem
@echo -----
@echo Sign certificate requests: %srvcertcasigned% and %cltcertcasigned%
@rem The requests are signed with the ca key in the cajkskeystore.jks keystore
@rem
@rem Sign server certificate request
%keytool% -gencert -infile %srvcertreq% -outfile %srvcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
@rem Sign client certificate request
@rem Omit this step, unless you are authenticating clients.
%keytool% -gencert -infile %cltcertreq% -outfile %cltcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%

```



```

@rem
@echo -----
@echo Import the signed certificates back into the key stores to create the key chain:
%srvjkskeystore% and %cltjkskeystore%
@rem
@rem Import the signed server certificate
%keytool% -import -noprompt -alias %srvalias% -file %svrcertcasigned% -keypass %srvkeypass%
-keystore %srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Import the signed client certificate and key chain back into the client keystore
%keytool% -import -noprompt -alias %cltalias% -file %cltcertcasigned% -keypass %cltkeypass%
-keystore %cltjkskeystore% -storepass %cltjkskeystorepass%
@rem
@rem The CA certificate is needed in the server key store, and the client trust store
@rem to verify the key chain sent from the client or server
@echo Delete the CA certificate from %cltjkskeystore%: it causes a problem in converting
keystore to pem
@rem Omit this step, unless you are authenticating clients.
%keytool% -delete -alias %caalias% -keystore %cltjkskeystore% -storepass
%cltjkskeystorepass%

```

```

@rem
@echo -----
@echo Create a pem client-ca trust store from the jks client-ca trust store:
%cltcapemtruststore%
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltcajkstruststore% -destkeystore
%cltcap12truststore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltcajkstruststorepass% -deststorepass %cltcap12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltcap12truststore% -out %cltcapemtruststore% -passin
pass:%cltcap12truststorepass% -passout pass:%cltpeptruststorepass%

```

```

@rem
@echo -----
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore% -destkeystore
%cltp12keystore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass% -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore% -out %cltpemkeystore% -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%

```

mqcerts.bat

Script 會列出憑證目錄中的金鑰儲存庫及憑證。然後，它會建立 MQTT 範例佇列管理程式，並配置安全的遙測通道。

```

@echo -----
@echo List keystores and certificates
dir %certpath%\*.* /b

```

```

@rem
@echo Create queue manager and define mqtt channels and certificate stores
call "%MQ_FILE_PATH%\mqxr\Samples\SampleMQM" >> %mqlog%
echo DEFINE CHANNEL(%chlreq%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreq%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlopt%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslopt%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
V7.5.0.1
echo DEFINE CHANNEL(%chlsslreqws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreqws%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlssloptws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portssloptws%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portws%)
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
@echo MQ logs saved in %mqlog%echo

```

透過 SSL 連接 Android 上的 MQTT 用戶端範例 Java 應用程式

開始進行透過 SSL 連接至 IBM WebSphere MQ 的範例 Android MQTT 用戶端。

開始之前

本文章假設您至少執行 Android API 層次 14 (ICS 4.0)。較早的層次具有金鑰儲存庫，但僅系統應用程式可以進行存取。

1. 您必須能夠存取支援 MQTT protocol over SSL 的 MQTT version 3.1 伺服器。
2. 如果用戶端與伺服器之間有防火牆，請確認它不會封鎖 MQTT 資料流量。
3. 如果您要測試早期 Android 裝置上的連線，您可能需要 SD 卡才能將憑證傳送至裝置。
4. 如果您要測試虛擬 Android 裝置上的連線，則請針對虛擬裝置配置 SD 卡。
5. SSL 通道必須已啟動。

關於這項作業

完成此作業，以透過 SSL 執行 Android 的 MQTT 用戶端範例 Java 應用程式。順利完成 SSL 連線可在您的 Android 裝置與 MQTT 伺服器之間建立安全的加密通道。伺服器的身分已經過鑑別。

使用 Android 時，您可以透過 SSL 來鑑別伺服器。您也可以鑑別裝置，雖然範例應用程式並不支援這樣做。若要鑑別裝置，請使用 [KeyChain API](#) 或使用 JAAS，來鑑別 MQTT Android 應用程式所提供的用戶端 ID、用戶端 IP 位址或使用名稱及密碼。

安裝到 Android 信任儲存庫的任何 X.509 憑證都必須由憑證管理中心簽署。在範例中，您會建立憑證管理中心來簽署您在 Android 裝置中所安裝的憑證。Android 裝置中已預先安裝許多主要憑證。

您必須先在 Android 裝置上建立鎖定，然後再安裝授信憑證。鎖定可防止他人在您不知情的狀況下於裝置上安裝憑證。

程序

1. 選擇您可以連接用戶端應用程式的 MQTT 伺服器。

伺服器必須支援透過 SSL 的 MQTT version 3.1 通訊協定。IBM 中的所有 MQTT 伺服器都會這麼做，包括 IBM WebSphere MQ 和 IBM MessageSight。請參閱 [第 122 頁的『開始使用 MQTT 伺服器』](#)。
2. 在未受保護的 MQTT 通道上，執行 for Android 的 MQTT 用戶端範例應用程式 "MQTTExciser"。請參閱 [第 16 頁的『在 Android 上開始使用適用於 Java 的 MQTT 用戶端』](#)。

您會再次使用該應用程式來測試安全通道。

如果您已啟動 Android 虛擬裝置，請讓其保持執行中狀態。
3. 選擇性的: 安裝第 7 版或更新版本的 Java 開發套件 (JDK)。

第 7 版，才能執行 **keytool** 指令來認證憑證。如果您不打算對憑證進行認證，則不需要第 7 版 JDK。
4. 建立並執行 Script 以產生金鑰組和憑證，並將 IBM WebSphere MQ 配置為 MQTT 伺服器。

遵循 [第 88 頁的『產生金鑰及憑證』](#) 中的步驟來建立並執行 Script。[第 61 頁的『為 Windows 配置 SSL 憑證的 Script 範例』](#) 中也列有這些 Script。

您需要憑證管理中心公用憑證以及伺服器金鑰儲存庫。您不需要用戶端憑證，或任何採用 .pem 或 .p12 格式的憑證。
5. 檢查 SSL 通道是否正在執行，且如您的預期設定。

在 IBM WebSphere MQ 上，於指令視窗中鍵入下列指令：

•  Linux

```
echo 'DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
echo 'DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
```

Windows

```
echo DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM  
echo DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM
```

- 將憑證管理中心憑證安裝在 Android 信任儲存庫中。

範例中的憑證管理中心檔案為 `cacert.cer`。

- 將該憑證重新命名為 `cacert.crt`
- 將該憑證複製到內部儲存體或 SD 卡的根目錄。

對於執行中的虛擬裝置，請開啟 Eclipse 或執行 Android Debug Bridge (ADB)，以將該憑證複製到虛擬裝置：

Eclipse

- 執行 Eclipse，然後開啟 DDMS 視景。
- 在主視圖中，開啟「檔案瀏覽器」視窗。
- 開啟 `mnt/sdcard` 目錄。
- 將 `cacert.crt` 檔案拖曳到 `mnt/sdcard` 目錄。

ADB

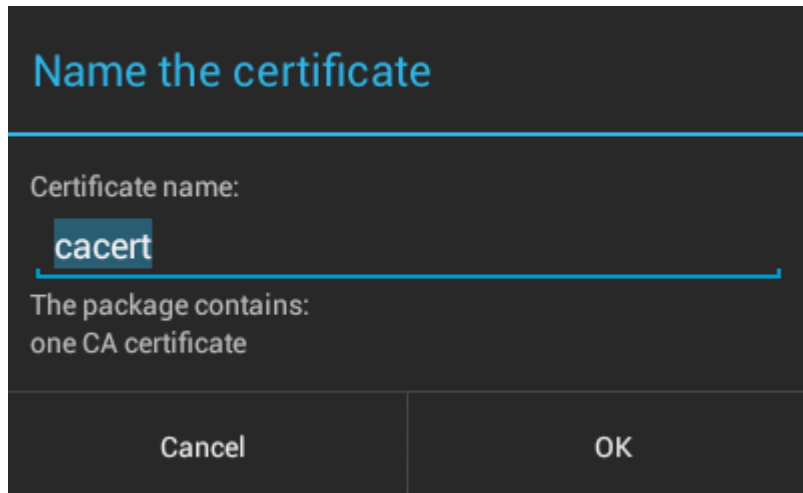
- 開啟指令視窗，並在 android 安裝目錄中將其現行目錄設為 `android-sdk\platform-tools`；例如 `C:\Program Files\Android\android-sdk\platform-tools`。
- 將該憑證複製到 `mnt/sdcard` 目錄：

```
adb push %cacert% /mnt/sdcard/cacert.crt
```

- 將憑證安裝在 Android 裝置上的憑證信任儲存庫中。

憑證必須具有 Basic Constraints 子句，且值為 Subject Type=CA。

- 解除鎖定裝置並按一下 **小組件** 按鈕。
- 按一下 **設定 > 安全 > 認證儲存體 > 從 SD 卡安裝**。

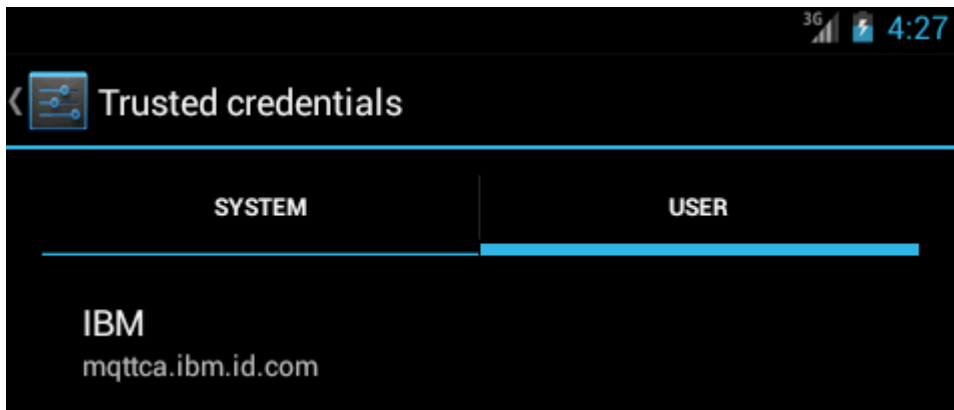


- 確認憑證檔名正確無誤，然後按一下 **確定**。

註：如果您尚未為裝置定義鎖定，此時 Android 會提示您設定鎖定。

- 確認已將憑證安裝在裝置上。

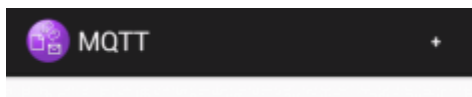
- 按一下 **授信認證 > 使用者**，並等待幾分鐘左右，讓您的憑證顯示在使用者憑證清單中。



9. 重新執行 MQTTExciser 應用程式，並連接至您為匿名 SSL 用戶端配置的 MQTT 通道。

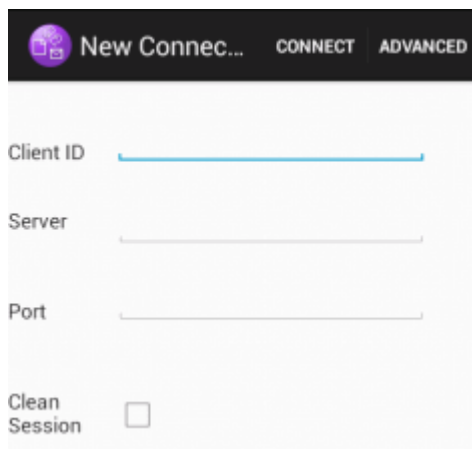
a) 開啟 Android 的 MQTT 用戶端範例 Java 應用程式。

此視窗會在您的 Android 裝置中開啟：



b) 連接至 MQTT 伺服器。

i) 按一下 + 號以開啟新的 MQTT 連線。



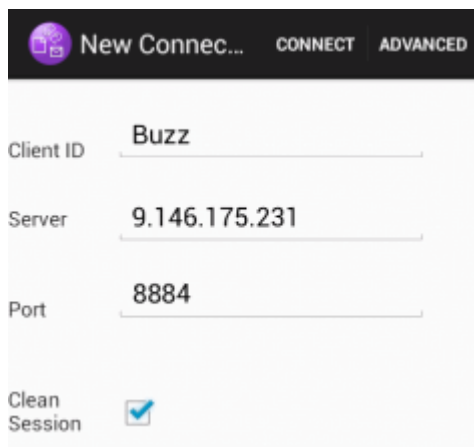
ii) 在**用戶端 ID** 欄位中輸入任何唯一的 ID。請耐心等待，按鍵速度可能緩慢。

iii) 在**伺服器**欄位輸入您 MQTT 伺服器的 IP 位址。

此為您在第一個主要步驟中選擇的伺服器。IP 位址不得為 127.0.0.1

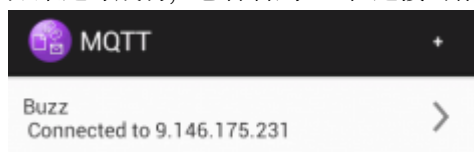
iv) 輸入 MQTT 連線的埠號。

將埠號設為 8884，其在範例 Script 中由變數 %sslportopt% 設定。此埠號是您透過在步驟 [第 58 頁的『4』](#) 中執行範例 Script，為匿名 SSL 用戶端所配置的 MQTT 通道的埠號。



- v) 按一下**進階**標籤，並選取 **SSL** 選項。按一下**儲存**。
- vi) 按一下**連接**。

如果連線成功，您會看到 "正在連接" 訊息，後面接著這個視窗：



結果

MQTTExciser 應用程式需要略長的時間才會連接並交換訊息，但在其他方面的行為與透過非安全連線進行連接時沒有任何差異。

相關概念

[第 47 頁的『MQTT 安全』](#)

以下是 MQTT 安全的三個基本概念：身分、鑑別及授權。身分可為要授權及提供權限的用戶端命名。鑑別可提供用戶端的身分，而授權可管理為用戶端提供的權限。

相關工作

[第 88 頁的『產生金鑰及憑證』](#)

遵循下列程序，為 Java 和 C 用戶端（包括 Android 及 iOS 應用程式）以及 IBM WebSphere MQ 和 IBM MessageSight 伺服器產生金鑰及憑證。

[第 50 頁的『建置及執行安全的 MQTT 用戶端範例 Java 應用程式』](#)

根據 Windows 範例，您可以在 IBM MessageSight 或 IBM WebSphere MQ 作為 MQTT 伺服器上啟動並執行安全範例 Java 應用程式。您可以在任何具有 JSE 1.5 或更高版本 (即 "Java 相容") 的平台上執行適用於 Java 的 MQTT 用戶端 應用程式

[第 67 頁的『向 JAAS 鑑別 MQTT 用戶端 Java 應用程式』](#)

學習如何使用 JAAS 鑑別用戶端。完成此作業中的步驟，以修改範例程式 JAASLoginModule.java，並配置 IBM WebSphere MQ 以向 JAAS 鑑別 MQTT 用戶端 Java 應用程式。

為 Windows 配置 SSL 憑證的 Script 範例

指令檔範例會建立憑證及憑證儲存庫，如作業中的步驟所述。此外，範例還會將 MQTT 用戶端佇列管理程式設為使用伺服器憑證儲存庫。範例會呼叫 IBM WebSphere MQ 隨附的 SampleMQM.bat Script，來刪除並重建佇列管理程式。

initcert.bat

initcert.bat 會設定 **keytool** 及 **openssl** 指令所需之憑證及其他參數的名稱和路徑。Script 中的註解說明了這些設定。

```
@echo off
@rem Set the path where you installed the MQTT SDK
@rem and short cuts to the samples directories.
set SDKRoot=C:\MQTT
set jsamppath=%SDKRoot%\sdk\clients\java\samples
set csamppath=%SDKRoot%\sdk\clients\c\samples
```

```
@rem Set the paths to Version 7 of the JDK
@rem and to the directory where you built the openssl package.
@rem Set short cuts to the tools.
set javapath=C:\Program Files\IBM\Java70
set keytool="%javapath%\jre\bin\keytool.exe"
set ikeyman="%javapath%\jre\bin\ikeyman.exe"
set openssl=%SDKRoot%\openssl
set runopenssl="%openssl%\bin\openssl"
```

```
@rem Set the path to where certificates are to be stored,
@rem and set global security parameters.
@rem Omit set password, and the security tools prompt you for passwords.
@rem Validity is the expiry time of the certificates in days.
set certpath=%SDKRoot%\Certificates
set password=password
set validity=5000
set algorithm=RSA
```

```
@rem Set the certificate authority (CA) jks keystore and certificate parameters.
@rem Omit this step, unless you are defining your own certificate authority.
@rem The CA keystore contains the key-pair for your own certificate authority.
@rem You must protect the CA keystore.
@rem The CA certificate is the self-signed certificate authority public certificate.
@rem It is commonly known as the CA root certificate.
set caalias=caalias
set cadname="CN=mqttca.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cakeypass=%password%
@rem ca key store
set cajkskeystore=%certpath%\cakeystore.jks
set cajkskeystorepass=%password%
@rem ca certificate (root certificate)
set cacert=%certpath%\cacert.cer
```

```
@rem Set the server jks keystore and certificate parameters.
@rem The server keystore contains the key-pair for the server.
@rem You must protect the server keystore.
@rem If you then export the server certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the server keystore for the server key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the server certificate to the CA.
@rem When you now export the server certificate,
@rem the exported certificate includes the certificate chain.
set srvalias=srvalias
set srvdname="CN=mqttserver.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set srvkeypass=%password%
@rem server key stores
set srvjkskeystore=%certpath%\srvkeystore.jks
set srvjkskeystorepass=%password%
@rem server certificates
set srvcertreq=%certpath%\srvcertreq.csr
set srvcertsigned=%certpath%\srvcertsigned.cer
set srvcertselfsigned=%certpath%\srvcertselfsigned.cer
```

```
@rem Set the client jks keystore and certificate parameters
@rem Omit this step, unless you are authenticating clients.
@rem The client keystore contains the key-pair for the client.
@rem You must protect the client keystore.
@rem If you then export the client certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the client keystore for the client key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
```

```

@rem The certificate chain links the client certificate to the CA.
@rem When you now export the client certificate,
@rem the exported certificate includes the certificate chain.
set cltalias=cltalias
set cltdname="CN=mqttclient.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cltkeypass=%password%
@rem client key stores
set cltjkskeystore=%certpath%\cltkeystore.jks
set cltjkskeystorepass=%password%
set cltcertreq=%certpath%\cltcertreq.csr
set cltcertassigned=%certpath%\cltcacertsassigned.cer
set cltcertselfsigned=%certpath%\cltcertselfsigned.cer

```

```

@rem Set the paths to the client truststores signed by CA and signed by server key.
@rem You only need to define one of the trust stores.
@rem A trust store holds certificates that you trust,
@rem which are used to authenticate untrusted certificates.
@rem In this example, when the client authenticates the MQTT server it connects to,
@rem it authenticates the certificate it is sent by the server
@rem with the certificates in its trust store.
@rem For example, the MQTT server sends its server certificate,
@rem and the client authenticates it with either the same server certificate
@rem that you have stored in the cltsrvtruststore.jks trust store,
@rem or against the CA certificate, if the server certificate is signed by the CA.
set cltcajkstruststore=%certpath%\cltcatruststore.jks
set cltcajkstruststorepass=%password%
set cltsrvjkstruststore=%certpath%\cltsrvtruststore.jks
set cltsrvjkstruststorepass=%password%

```

```

@rem Set the paths to the client PKCS12 and PEM key and trust stores.
@rem Omit this step, unless you are configuring a C or iOS client.
@rem You only need to define either one of the trust stores for storing CA
@rem or server signed server certificates.
set cltp12keystore=%certpath%\cltkeystore.p12
set cltp12keystorepass=%password%
set cltpemkeystore=%certpath%\cltkeystore.pem
set cltpemkeystorepass=%password%
set cltcap12truststore=%certpath%\cltcatruststore.p12
set cltcap12truststorepass=%password%
set cltcapemtruststore=%certpath%\cltcatruststore.pem
set cltcapemtruststorepass=%password%
set cltsrvp12truststore=%certpath%\cltsrvtruststore.p12
set cltsrvp12truststorepass=%password%
set cltsrvpemtruststore=%certpath%\cltsrvtruststore.pem
set cltsrvpemtruststorepass=%password%

```

```

@rem set WMQ Variables
set authopt=NEVER
set authreq=REQUIRED
set qm=MQXR_SAMPLE_QM
set host=localhost
set mcauser='Guest'
set portsslopt=8884
set chllopt=SSLOPT
set portsslreq=8885
set chlreq=SSLREQ
V 7.5.0.1 set portws=1886
set chlws=PLAINWS
set chlssloptws=SSLOPTWS
set portssloptws=8886
set chlsslreqws=SSLREQWS
set portsslreqws=8887
set mqlog=%certpath%\wmq.log

```

cleancert.bat

cleancert.bat Script 中的指令會刪除 MQTT 用戶端佇列管理程式以確保伺服器憑證儲存庫未鎖定，然後會刪除範例安全 Script 建立的所有金鑰儲存庫及憑證。

```

@rem Delete the MQTT sample queue manager, MQXR_SAMPLE_QM
call "%MQ_FILE_PATH%\bin\setmqenv" -s
endmqm -i %qm%
dlmqm %qm%

```

```

@rem Erase all the certificates and key stores created by the sample scripts.
erase %cajkskeystore%

```

```

erase %cacert%
erase %srvjkskeystore%
erase %svrcertreq%
erase %svrcertcasigned%
erase %svrcertselfsigned%
erase %cltjkskeystore%
erase %cltp12keystore%
erase %cltpemkeystore%
erase %cltcertreq%
erase %cltcertcasigned%
erase %cltcertselfsigned%
erase %cltcajkstruststore%
erase %cltcap12truststore%
erase %cltcapemtruststore%
erase %cltsrvjkstruststore%
erase %cltsrvp12truststore%
erase %cltsrvpemtruststore%
erase %mqlog%
@echo Cleared all certificates
dir %certpath%\*.* /b

```

genkeys.bat

genkeys.bat Script 中的指令會為您的專用憑證管理中心、伺服器及用戶端建立金鑰組。

```

@rem
@echo -----
@echo Generate %caalias%, %srvalias%, and %cltalias% key-pairs in %cajkskeystore%,
%srvjkskeystore%, and %cltjkskeystore%
@rem
@rem -- Generate a client certificate and a private key pair
@rem Omit this step, unless you are authenticating clients.
%keytool% -genkeypair -noprompt -alias %cltalias% -dname %cltdname% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass% -keypass %cltkeypass% -keyalg %algorithm%
-validity %validity%

```

```

@rem -- Generate a server certificate and private key pair
%keytool% -genkeypair -noprompt -alias %srvalias% -dname %srvdname% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass% -keypass %srvkeypass% -keyalg %algorithm%
-validity %validity%

```

```

@rem Create CA, client and server key-pairs
@rem -- Generate a CA certificate and private key pair - The extension asserts this is a
certificate authority certificate, which is required to import into firefox
%keytool% -genkeypair -noprompt -ext bc=ca:true -alias %caalias% -dname %cadname%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass% -keyalg
%algorithm% -validity %validity%

```

sscerts.bat

sscerts.bat Script 中的指令會從其金鑰儲存庫匯出用戶端及伺服器自簽憑證，然後將伺服器憑證匯入用戶端信任儲存庫，並將用戶端憑證匯入伺服器金鑰儲存庫。伺服器沒有信任儲存庫。這些指令會從用戶端 JKS 信任儲存庫，以 PEM 的格式建立用戶端信任儲存庫。

```

@rem
@echo -----
@echo Export self-signed certificates: %svrcertselfsigned% and %cltcertselfsigned%
@rem Export Server public certificate
%keytool% -exportcert -noprompt -rfc -alias %srvalias% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass% -file %svrcertselfsigned%
@rem Export Client public certificate
@rem Omit this step, unless you are authenticating clients.
%keytool% -exportcert -noprompt -rfc -alias %cltalias% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass% -file %cltcertselfsigned%

```

```

@rem
@echo -----
@echo Add selfsigned server certificate %svrcertselfsigned% to client trust store:
%cltsrvjkstruststore%
@rem Import the server certificate into the client-server trust store (for server self-
signed authentication)
%keytool% -import -noprompt -alias %srvalias% -file %svrcertselfsigned% -keystore
%cltsrvjkstruststore% -storepass %cltsrvjkstruststorepass%

```



```

@rem
@echo -----
@echo Add selfsigned client certificate %cltcertselfsigned% to server trust store:
%srvjkskeystore%
@rem Import the client certificate into the server trust store (for client self-signed
authentication)
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %cltalias% -file %cltcertselfsigned% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%

```

```

@rem
@echo -----
@echo Create a pem client-server trust store from the jks client-server trust store:
%cltsrvpemtruststore%
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltsrvjkskeystore% -destkeystore
%cltsrvp12truststore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltsrvjkskeystorepass% -deststorepass %cltsrvp12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltsrvp12truststore% -out %cltsrvpemtruststore% -passin
pass:%cltsrvp12truststorepass% -passout pass:%cltsrvpemtruststorepass%@rem
@rem
@echo -----
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore% -destkeystore
%cltp12keystore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass% -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore% -out %cltpemkeystore% -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%

```

cacerts.bat

此 Script 會將憑證管理中心主要憑證匯入專用金鑰儲存庫。需要 CA 主要憑證，才可在主要憑證與已簽章的憑證之間建立金鑰鏈。cacerts.bat Script 會從其金鑰儲存庫匯出用戶端及伺服器憑證申請。Script 會使用 cajkskeystore.jks 金鑰儲存庫中專用憑證管理中心的金鑰，來簽署憑證申請，然後將已簽章的憑證匯回提出申請的同一個金鑰儲存庫。匯入會使用 CA 主要憑證建立憑證鏈。Script 會從用戶端 JKS 信任儲存庫，以 PEM 的格式建立用戶端信任儲存庫。

```

@rem
@echo -----
@echo Export self-signed certificates: %cacert%
@rem
@rem Export CA public certificate
%keytool% -exportcert -noprompt -rfc -alias %caalias% -keystore %cajkskeystore% -storepass
%cajkskeystorepass% -file %cacert%

```

```

@rem
@echo -----
@echo Add CA to server key and client key and trust stores: %srvjkskeystore%,
%cltjkskeystore%, %cltcajkstruststore%,
@rem The CA certificate is necessary to create key chains in the client and server key
stores,
@rem and to certify key chains in the server key store and the client trust store
@rem
@rem Import the CA root certificate into the server key store
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass%
@rem Import the CA root certificate into the client key store
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass%
@rem Import the CA root certificate into the client ca-trust store (for ca chained
authentication)
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltcajkstruststore%
-storepass %cltcajkstruststorepass%

```

```

@rem
@echo -----
@echo Create certificate signing requests: %srvcertreq% and %cltcertreq%
@rem
@rem Create a certificate signing request (CSR) for the server key
%keytool% -certreq -alias %srvalias% -file %srvcertreq% -keypass %srvkeypass% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Create a certificate signing request (CSR) for the client key

```

```
%keytool% -certreq -alias %cltalias% -file %cltcertreq% -keypass %cltkeypass% -keystore %cltjkskeystore% -storepass %cltjkskeystorepass%
```

```
@rem
@echo -----
@echo Sign certificate requests: %srvcertassigned% and %cltcertassigned%
@rem The requests are signed with the ca key in the cajkskeystore.jks keystore
@rem
@rem Sign server certificate request
%keytool% -gencert -infile %srvcertreq% -outfile %srvcertassigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
@rem Sign client certificate request
@rem Omit this step, unless you are authenticating clients.
%keytool% -gencert -infile %cltcertreq% -outfile %cltcertassigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
```

```
@rem
@echo -----
@echo Import the signed certificates back into the key stores to create the key chain:
%srvjkskeystore% and %cltjkskeystore%
@rem
@rem Import the signed server certificate
%keytool% -import -noprompt -alias %srvalias% -file %srvcertassigned% -keypass %srvkeypass%
-keystore %srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Import the signed client certificate and key chain back into the client keystore
%keytool% -import -noprompt -alias %cltalias% -file %cltcertassigned% -keypass %cltkeypass%
-keystore %cltjkskeystore% -storepass %cltjkskeystorepass%
@rem
@rem The CA certificate is needed in the server key store, and the client trust store
@rem to verify the key chain sent from the client or server
@echo Delete the CA certificate from %cltjkskeystore%: it causes a problem in converting
keystore to pem
@rem Omit this step, unless you are authenticating clients.
%keytool% -delete -alias %caalias% -keystore %cltjkskeystore% -storepass
%cltjkskeystorepass%
```

```
@rem
@echo -----
@echo Create a pem client-ca trust store from the jks client-ca trust store:
%cltcapemtruststore%
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltcajkskeystore% -destkeystore
%cltcap12truststore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltcajkskeystorepass% -deststorepass %cltcap12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltcap12truststore% -out %cltcapemtruststore% -passin
pass:%cltcap12truststorepass% -passout pass:%cltpeptruststorepass%
```

```
@rem
@echo -----
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore% -destkeystore
%clt12keystore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass% -deststorepass %clt12keystorepass%
%openssl%\bin\openssl pkcs12 -in %clt12keystore% -out %cltpeakeystore% -passin
pass:%clt12keystorepass% -passout pass:%cltpeakeystorepass%
```

mqcerts.bat

Script 會列出憑證目錄中的金鑰儲存庫及憑證。然後，它會建立 MQTT 範例佇列管理程式，並配置安全的遙測通道。

```
@echo -----
@echo List keystores and certificates
dir %certpath%\*.* /b
```

```
@rem
@echo Create queue manager and define mqtt channels and certificate stores
call "%MQ_FILE_PATH%\mqxr\Samples\SampleMQM" >> %mqlog%
echo DEFINE CHANNEL(%chlreq%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreq%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chllopt%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portssllopt%)
SSLCAUTH(%authlopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
```

```
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
V7.5.0.1
echo DEFINE CHANNEL(%chlsslreqws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreqws%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlsslptws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslptws%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portws%)
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
@echo MQ logs saved in %mqlog%echo
```

向 JAAS 鑑別 MQTT 用戶端 Java 應用程式

學習如何使用 JAAS 鑑別用戶端。完成此作業中的步驟，以修改範例程式 JAASLoginModule.java，並配置 IBM WebSphere MQ 以向 JAAS 鑑別 MQTT 用戶端 Java 應用程式。

開始之前

1. 您可以在任何具有 JSE 1.5 或更高版本 (即 "Java 相容") 的平台上執行適用於 Java 的 MQTT 用戶端應用程式。請參閱 [IBM Mobile Messaging and M2M 用戶端套件的系統需求](#)。
2. 如果用戶端與伺服器之間有防火牆，請確認它不會封鎖 MQTT 資料流量。
3. 您必須有權存取 IBM WebSphere MQ 安裝中的 MQXR JAASLoginModule 和 JAASPrincipal Java 範例。範例位於路徑 %MQ_FILE_PATH%\mqxr\samples。
4. 在 Windows 或 Linux 上完成所需的步驟；範例取自 Windows。
5. 若要完成步驟 [第 67 頁的『1』](#)，您必須具有在 IBM WebSphere MQ 上建立 MQXR_SAMPLE_QM 佇列管理程式的權限。

關於這項作業

在作業中，您將從 JAASLoginModule 版本輸出 MQTT Sample 用戶端識別參數。寫出用戶端參數需要修改範例 JAASLoginModule 程式並配置 IBM WebSphere MQ，以載入您的 JAASLoginModule 版本。

程序

1. 完成第 14 頁的『從 Eclipse 編譯及執行所有 MQTT 用戶端範例 Java 應用程式』中的步驟，以執行 Paho MQTT Sample 用戶端。

您的目標是準備開發環境，以開發及測試 JAAS 鑑別。您需要 Java 開發環境來自訂 JAAS 鑑別模組。在範例中，您會執行適用於 Java 的範例 Paho 用戶端，以測試您的 JAAS 配置。為了簡單起見，請使用相同的開發環境來修改範例用戶端及範例 JAAS 登入模組。另外還可以使用適用於 C 的 MQTT 用戶端，或者其他 MQTT 用戶端來測試 JAAS 登入模組。

2. 選擇性的：將使用者名稱及密碼參數新增至 MQTT Paho 範例。

註：如果適用於 Java 的 Paho 用戶端包含使用者名稱及密碼參數，則不需要執行此步驟。檢查下載網站是否有更新。請參閱 [IBM 傳訊社群下載](#)，否則請變更您的 Sample.java 副本。

- a) 在 Paho 範例專案的 org.eclipse.paho.sample.mqttv3app 套件中，開啟套件瀏覽器。
- b) 用滑鼠右鍵按一下 Sample.java 複製 > 貼上。在「名稱衝突」視窗中，鍵入名稱 SampleForJAAS。
- c) 將下列程式碼行新增至 main 方法。

- i) 在 "boolean ssl = false;" 行後面，宣告 userName 及 password 變數：

```
String password = null;
String userName = null;
```

為了與較舊的 MQTT 伺服器相容，依預設不會設定密碼及使用者名稱參數。

- ii) 在 "case 'v': ssl = Boolean.valueOf(args[++i]).booleanValue(); break;" 行後面，剖析這兩個新輸入參數：

```
case 'u': userName = args[++i]; break;
case 'z': password = args[++i]; break;
```

- iii) 在 "if (action.equals("publish")) {" 行前面，將 userName 及 password 新增至 Sample 建構子引數：

```
Sample sampleClient = new Sample(url, clientId, cleanSession, quietMode, userName,
password);
```

- d) 將 userName 及 password 新增至 Sample 的建構子。

變更：

```
public Sample(String brokerUrl, String clientId, boolean cleanSession,
boolean quietMode) throws MqttException {
```

至：

```
public Sample(String brokerUrl, String clientId, boolean cleanSession,
boolean quietMode, String userName, char[] password) throws MqttException {
```

- e) 將下列程式碼行新增至 Sample 建構子。

在 "conOpt.setCleanSession(clean);" 行後面，於 Sample 方法的 conOpt 物件中設定 userName 及 password 變數：

```
if(password != null ) {
    conOpt.setPassword(this.password.toCharArray());
}
if(userName != null) {
    conOpt.setUserName(this.userName);
}
```

- f) 在 publish 及 subscribe 方法中，修改下列程式碼行：

將 "client.connect();" 行變更為

```
client.connect(conOpt);
```

3. 為您的 JAAS 範例建立 Java 專案 JAASSample。

- 在 Eclipse 工作區中，開啟「新建 Java 專案」精靈：按一下檔案 > 新建 > Java 專案。
- 在專案名稱欄位中，鍵入 JAASSample。
- 在 JRE 選項中，選取 J2SE-1.5 作為執行環境 JRE，然後按下一步。

該 JRE 必須符合 IBM WebSphere MQ 伺服器所執行的 JRE。IBM WebSphere MQ Version 7.5 執行 J2SE-1.5。

- 在「Java 設定」視窗中，按一下程式庫標籤，然後按一下新增外部 JARS...。瀏覽至 %MQ_FILE_PATH%\mqxr\lib 目錄，然後選取 MQXR.jar；按一下完成。

4. 匯入 JAAS 範例 JAASLoginModule 和 JAASPrincipal 類別。

- 在套件瀏覽器中，用滑鼠右鍵按一下 JAASSample 專案 匯入 ... > 一般 > 檔案系統，然後按下一步。
- 瀏覽至 %MQ_FILE_PATH%\mqxr\samples 並勾選 JAASLoginModule.java 及 JAASPrincipal.java；然後按一下完成。
- 在「套件瀏覽器」中，選取並用滑鼠右鍵按一下這兩個 Java 檔案，重構 ... > 移動。
- 在「移動」視窗中，驗證已選取 JAASSample 作為這兩個元素的目的地，然後按一下建立套件...
- 在「新建 Java 套件」精靈的名稱欄位中鍵入 samples；然後按一下完成 > 確定

Eclipse 即會建置已匯入的 Java 類別，並顯示一些有關未用的值之警告。

5. 重新命名 JAASLoginModule 類別

重新命名類別，以便更容易將它與 IBM WebSphere MQ 隨附的範例 JAASLoginModule 類別區分。

- a) 在套件瀏覽器中，用滑鼠右鍵按一下 JAASloginModule.java 重構 ... > 重新命名。
 - b) 在「重新命名編譯單元」視窗中，將新名稱欄位從 JAASloginModule 變更為 MyJAASloginModule；然後按一下完成。
6. 修改 MyJAASloginModule 類別，以輸出回呼欄位的內容。
- a) 將下列程式碼行新增至 MyJAASloginModule.java。

```
System.out.println("Username=" + username
    + "\nPassword=" + new String(password)
    + "\nClientId=" + clientId
    + "\nNetwork address=" + networkAddress);
```

將這些行置於陳述式 "if (true) loggedIn = true;" 的正前面

- b) 按 CTRL+Shift+O 以重組匯入項目，然後儲存檔案。
7. 將 JAASPrincipal 重新命名為 MyJAASPrincipal。

重新命名該類別，以免與範例 JAASPrincipal 類別混淆。在範例中，保持 MyJAASPrincipal 類別的內容不變。

8. 為執行佇列管理程式程序的使用者 ID，提供對 JAAS 類別的 read 及 execute 許可權。
- a) 在「Windows 檔案總管」中，開啟您的 Eclipse 工作區目錄。在此範例中，Eclipse 工作區位置由 Eclipse 變數 *workspace_loc* 代表。
 - b) 瀏覽至包含 MyJAASLoginModule 及 MyJAASPrincipal 類別的目錄。
目錄路徑為 *workspace_loc\JAASSample\bin\samples*
 - c) 選取並用滑鼠右鍵按一下這兩個類別，然後按一下內容；按一下「內容」視窗中的安全標籤。
 - d) 按一下新增 ...，鍵入物件名稱 *mqm*，然後按一下檢查名稱以驗證它；按一下確定。
 - e) 選取「群組或使用者名稱」清單中的 *mqm*，並為 *mqm* 勾選許可權清單中的讀取及執行及讀取；然後按一下「確定」。
9. 配置 IBM WebSphere MQ，以執行 MyJAASLoginModule 類別。

- a) 將 *service.env* 檔案新增至 IBM WebSphere MQ 配置，以定義類別路徑來載入 MyJAASLoginModule 類別。

在 *WMQ_DATA_PATH* 目錄中，使用下列類別路徑陳述式建立 *service.env* 檔案：

```
CLASSPATH=user.dir\JAASSample\bin
```

其中 *user.dir* 是在 Eclipse 工作區中編譯之類別檔的根目錄。 *WMQ_DATA_PATH* 目錄包含 *qmgrs* 目錄。請參閱其他環境變數。

提示: CLASSPATH=*user.dir*\JAASSample\bin 可能是 *service.env* 檔案中的唯一陳述式

- b) 將段落 MyJAASStanza 新增至 *jaas.config* 檔案，以識別您的 MyJAASLoginModule 類別（相對於 *service.env* 檔案中的類別路徑）。

jaas.config 位於佇列管理程式 *mqxr* 目錄 *WMQ_DATA_PATH\Qmgrs\MQXR_SAMPLE_QM\mqxr* 中。

該段落為：

```
MyJAASStanza {
    samples.MyJAASLoginModule required debug=true;
};
```

- c) 使用 JAAS 配置段落名稱，來配置 IBM WebSphere MQ Telemetry 通道。

從指令視窗中執行下列指令：

```
echo DEFINE CHANNEL('MyJAAS') CHLTYPE(MQTT) TRPTYPE(TCP) PORT(1890)
JAASCFG('MyJAASStanza') | runmqsc MQXR_SAMPLE_QM
```

該指令會將 MyJAAS 通道連結至 `jaas.config` 檔案中的 MyJAASStanza。如果在通道定義中不指定 MCAUSER 選項，或指定了 USECLTID 選項，則該通道會授權使用 MQTT 用戶端程式提供的使用者名稱來存取佇列管理程式資源。在範例中，用戶端提供的使用者名稱設為 "Guest"。此範例中還會使用由 SampleMQM 指令檔所設定 Guest 的現有授權。

10. 重新啟動 IBM WebSphere MQ Telemetry 服務，以讀取新的配置資料。
若要重新啟動 IBM WebSphere MQ Telemetry 服務，請從「IBM WebSphere MQ Explorer」啟動佇列管理程式或服務，或者針對配置範例執行下列指令：

```
echo stop service(SYSTEM.MQXR.SERVICE) | runmqsc MQXR_SAMPLE_QM
echo start service(SYSTEM.MQXR.SERVICE) | runmqsc MQXR_SAMPLE_QM
```

11. 執行 Sample 程式。

若要配置 SampleForJAAS 的執行配置，請遵循步驟 [第 67 頁的『1』](#) 中的相同程序，並進行下列修改：

- a) 將埠號設定為 1890 以符合 MQTT 通道配置。
- b) 在您為 Sample 程式建立的「訂閱者」及「發佈者」配置的「(x)= 引數」標籤上，將參數 `-u Guest -z password` 新增至密碼

除埠號現在為 1890 而非 1883 之外，執行範例程式之後的輸出沒有任何變更。

在 `WMQ_DATA_PATH\Qmgrs\MQXR_SAMPLE_QM` 目錄中，開啟 `mqxr.stdout` 檔案。來自 MyJAASLoginModule 的輸出會寫入 `mqxr.stdout`：

```
Username=Guest
Password=password
ClientId=SampleJavaV3_subscribe
Network address=/127.0.0.1
```

下一步

如果您的範例無法運作，請閱讀 JAAS 的疑難排解主題 [第 163 頁的『解決問題：Telemetry 服務未呼叫 JAAS 登入模組』](#)，然後嘗試下列除錯要訣。

1. 將 `-verbose` 新增至 `WMQ_DATA_PATH\Qmgrs\MQXR_SAMPLE_QM\mqxr\java.properties` 中的參數。從此日誌中，可以查看您的類別是否順利載入。
輸出會寫入 `WMQ_DATA_PATH\Qmgrs\MQXR_SAMPLE_QM\mqxr.stderr`。
2. 查看 `WMQ_DATA_PATH\Qmgrs\MQXR_SAMPLE_QM\errors\mqxr.log` 中是否有 MyJAASLoginModule 中擲出的異常狀況。比方說，如果您嘗試輸出空值 `password`（此為字元陣列而非字串），則會擲出異常狀況。
3. 查看 `WMQ_DATA_PATH\Qmgrs\MQXR_SAMPLE_QM\errors\AMQERR01.log`。如果 `userName` 中的使用者名稱未獲授權來存取佇列管理程式資源，且未使用 MCAUSER 或 USECLTID 選項來配置通道，則會在此處報告所有錯誤。
4. 檢查 `WMQ_DATA_PATH\Qmgrs\MQXR_SAMPLE_QM\mqxr\jaas.config` 中段落的名稱是否與針對 Sample 用戶端嘗試連接之埠配置的 MQTT 通道中的名稱相同。
5. 檢查段落中的路徑是否符合 Eclipse 中 MyJAASLoginModule 類別的路徑；例如：

```
MyJAASStanza {
    samples.MyJAASLoginModule required debug=true;
};
```

6. 為排除錯誤是否在於未正確挑選 `WMQ_DATA_PATH` 內 `service.env` 檔案中的類別路徑，請變更 `%MQ_FILE_PATH%\mqxr\bin\controlMQXR.BAT` 中的 "set CLASSPATH=%MQXRCLASSPATH%;%CLASSPATH%" 行，以包含您的類別路徑。您也可以回應類別路徑。但是，由於該類別路徑不包含 `service.env` 中所設定的類別路徑，因此這僅在您修改 `controlMQXR.BAT` 檔案時才起作用。

相關概念

[第 47 頁的『MQTT 安全』](#)

以下是 MQTT 安全的三個基本概念：身分、鑑別及授權。身分可為要授權及提供權限的用戶端命名。鑑別可提供用戶端的身分，而授權可管理為用戶端提供的權限。

[第 105 頁的『遙測通道 JAAS 配置』](#)

配置 JAAS 以鑑別用戶端傳送的 Username。

相關工作

[第 163 頁的『解決問題：Telemetry 服務未呼叫 JAAS 登入模組』](#)

瞭解遙測 (MQXR) 服務是否未呼叫您的 JAAS 登入模組，並配置 JAAS 以更正問題。

[第 50 頁的『建置及執行安全的 MQTT 用戶端範例 Java 應用程式』](#)

根據 Windows 範例，您可以在 IBM MessageSight 或 IBM WebSphere MQ 作為 MQTT 伺服器上啟動並執行安全範例 Java 應用程式。您可以在任何具有 JSE 1.5 或更高版本 (即 "Java 相容") 的平台上執行適用於 Java 的 MQTT 用戶端 應用程式

[第 58 頁的『透過 SSL 連接 Android 上的 MQTT 用戶端範例 Java 應用程式』](#)

開始進行透過 SSL 連接至 IBM WebSphere MQ 的範例 Android MQTT 用戶端。

相關資訊

[其他環境變數](#)

透過 SSL 及 WebSockets 連接 適用於 JavaScript 的 MQTT 傳訊用戶端

使用 適用於 JavaScript 的 MQTT 傳訊用戶端 範例 HTML 頁面搭配 SSL 及 WebSocket protocol，將 Web 應用程式安全地連接至 IBM WebSphere MQ。

開始之前

1. 您必須對支援 MQTT protocol (透過 WebSockets) 的 MQTT version 3 伺服器具有存取權。
2. 瀏覽器必須支援 SSL 及 WebSocket protocol。請參閱 [第 155 頁的『瀏覽器對於使用 SSL 之行動式傳訊 Web 應用程式的支援限制』](#)。
3. SSL 通道必須已啟動。

關於這項作業

完成此作業，以透過 SSL 執行適用於 JavaScript 的 MQTT 傳訊用戶端範例頁面。此作業會將您指引至 [第 88 頁的『產生金鑰及憑證』](#)，以建立憑證及配置 IBM WebSphere MQ。

使用憑證管理中心簽署的金鑰或自簽金鑰保護 SSL 通道。

程序

1. 選擇您可以連接用戶端應用程式的 MQTT 伺服器。
伺服器必須支援 MQTT protocol (透過安全的 WebSockets)。
 - IBM MessageSight、IBM WebSphere MQ 7.5.0.1 版以及更新版本皆支援該通訊協定。
2. 選擇性的: 安裝 Java 開發套件 (JDK) 第 7 版或更新版本。
如果您要設定測試系統，並且想要使用自簽憑證，則需要使用 JDK 第 7 版 **keytool** 指令以對憑證進行認證。如果您要設定正式作業系統，並要將憑證簽署要求 (CSR) 傳送至外部憑證管理中心，則不需要第 7 版 JDK。
3. 建立並執行 Script 以產生金鑰組和憑證，並將 IBM WebSphere MQ 配置為 MQTT 伺服器。
遵循 [第 88 頁的『產生金鑰及憑證』](#) 中的步驟來建立並執行 Script。[第 73 頁的『為 Windows 配置 SSL 憑證的 Script 範例』](#) 中也列有這些 Script。

伺服器憑證的通用名稱必須與伺服器通道的 DNS 名稱相符。部分瀏覽器接受包含通用名稱清單的憑證；例如：

```
"CN=localhost, CN=*.example.com"
```

其他瀏覽器則僅接受一個通用名稱。例如，Firefox（高達第 18 版）僅接受一個通用名稱。更新版本可能會不同。

4. 檢查 SSL 通道是否正在執行，且如您的預期設定。

在 IBM WebSphere MQ 上，於指令視窗中鍵入下列指令：

- **Linux**

```
echo 'DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM  
echo 'DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
```

- **Windows**

```
echo DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM  
echo DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM
```

5. 將憑證安裝在瀏覽器憑證儲存庫。

對於此範例，請選擇下列其中一個伺服器憑證：

- a. 對於自簽伺服器憑證，憑證為 `srvcertselfsigned.cer`。
- b. 對於專用憑證管理中心簽署的伺服器憑證，憑證為 `cacert.cer`。
- c. 對於外部憑證管理中心簽署的伺服器憑證，請檢查憑證管理中心的主要憑證是否已安裝在憑證儲存庫中。

根據您瀏覽器所提供的支援，將 `cacert.cer` 安裝到「受信任的根憑證授權單位」清單中。請參閱 [第 155 頁的『瀏覽器對於使用 SSL 之行動式傳訊 Web 應用程式的支援限制』](#)。

6. 選擇性的：鑑別用戶端。

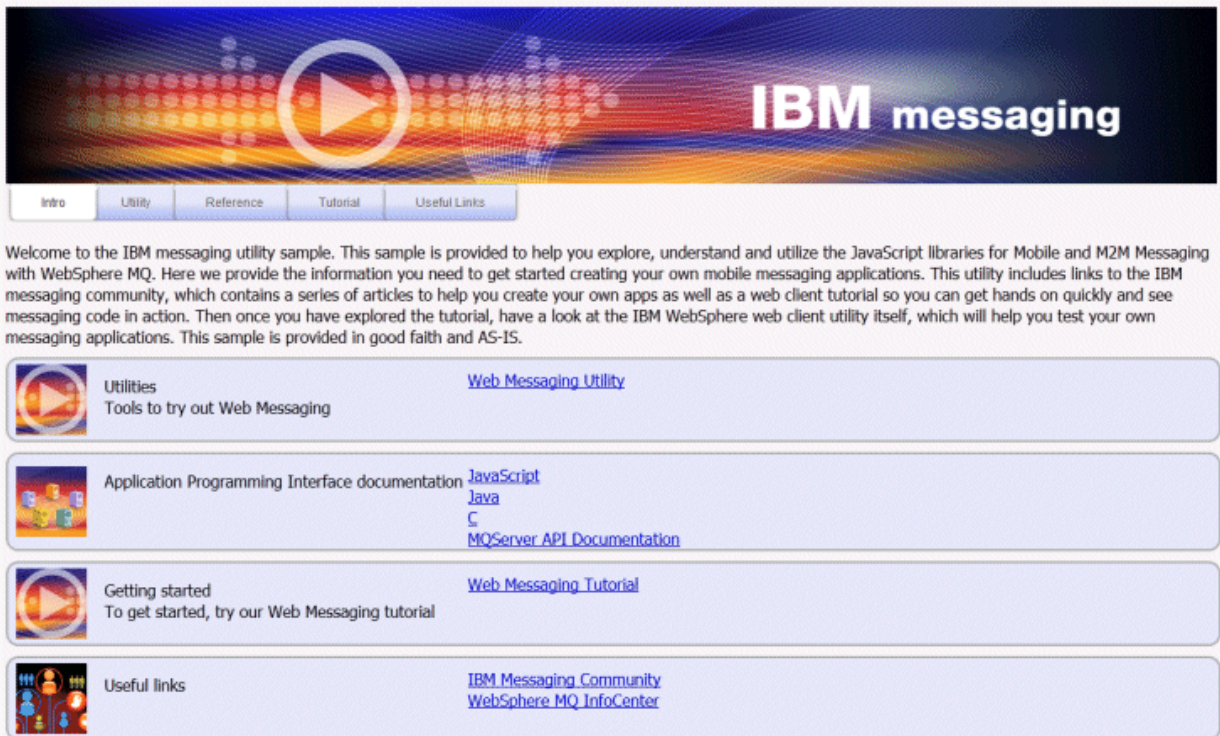
在此範例中，您安裝 `cacert.cer` 以鑑別伺服器及加密通道，而不是鑑別用戶端。若要鑑別用戶端，您必須將用戶端金鑰儲存庫 `cltkeystore.p12` 安裝在瀏覽器中。並非所有瀏覽器都支援鑑別用戶端。請參閱 [第 155 頁的『瀏覽器對於使用 SSL 之行動式傳訊 Web 應用程式的支援限制』](#)。

7. 連接至安全的 WebSockets 通道。

開啟瀏覽器，並在位址列中鍵入 WebSockets 通道的 URL；在範例中：

```
https://localhost:8886
```

IBM WebSphere MQ 會透過顯示 MQTT 傳訊用戶端範例 JavaScript 頁面的首頁進行回應。



Welcome to the IBM messaging utility sample. This sample is provided to help you explore, understand and utilize the JavaScript libraries for Mobile and M2M Messaging with WebSphere MQ. Here we provide the information you need to get started creating your own mobile messaging applications. This utility includes links to the IBM messaging community, which contains a series of articles to help you create your own apps as well as a web client tutorial so you can get hands on quickly and see messaging code in action. Then once you have explored the tutorial, have a look at the IBM WebSphere web client utility itself, which will help you test your own messaging applications. This sample is provided in good faith and AS-IS.

- Utilities: [Web Messaging Utility](#)
Tools to try out Web Messaging
- Application Programming Interface documentation: [JavaScript](#), [Java](#), [C](#), [MQServer API Documentation](#)
- Getting started: [Web Messaging Tutorial](#)
To get started, try our Web Messaging tutorial
- Useful links: [IBM Messaging Community](#), [WebSphere MQ InfoCenter](#)

如果連線失敗，且您執行了範例 Script 來設定範例 MQTT 佇列管理程式，請嘗試連接至埠 1886 上的一般 WebSockets 通道。埠 1886 上的成功會隔離 SSL 連線的失敗。

```
https://localhost:1886
```

相關概念

[第 106 頁的『適用於 JavaScript 的 MQTT 傳訊用戶端及 Web 應用程式』](#)

[第 109 頁的『如何在 JavaScript 中對傳訊應用程式進行程式設計』](#)

相關工作

[第 88 頁的『產生金鑰及憑證』](#)

遵循下列程序，為 Java 和 C 用戶端（包括 Android 及 iOS 應用程式）以及 IBM WebSphere MQ 和 IBM MessageSight 伺服器產生金鑰及憑證。

[第 21 頁的『適用於 JavaScript 的 MQTT 傳訊用戶端入門』](#)

透過顯示傳訊用戶端範例首頁並瀏覽它所鏈結的資源，即可開始使用適用於 JavaScript 的 MQTT 傳訊用戶端。若要顯示此首頁，您可以配置 MQTT 伺服器以接受來自 MQTT 傳訊用戶端範例 JavaScript 頁面的連線，然後在 Web 瀏覽器中鍵入您在伺服器上配置的 URL。適用於 JavaScript 的 MQTT 傳訊用戶端會在您的裝置上自動啟動，並且會顯示傳訊用戶端範例首頁。此頁面包含公用程式、程式設計介面文件、指導教學及其他有用資訊的鏈結。

相關參考

[第 155 頁的『瀏覽器對於使用 SSL 之行動式傳訊 Web 應用程式的支援限制』](#)

這在不同瀏覽器與平台組合上會有所差異。瞭解這些差異對於您配置應用程式、憑證管理中心 (CA)，以及使用適用於 JavaScript 的 MQTT 傳訊用戶端經由 SSL 及 WebSockets 連線時所需的用戶端憑證會很有幫助。

為 Windows 配置 SSL 憑證的 Script 範例

指令檔範例會建立憑證及憑證儲存庫，如作業中的步驟所述。此外，範例還會將 MQTT 用戶端佇列管理程式設為使用伺服器憑證儲存庫。範例會呼叫 IBM WebSphere MQ 隨附的 SampleMQM.bat Script，來刪除並重建佇列管理程式。

initcert.bat

initcert.bat 會設定 **keytool** 及 **openSSL** 指令所需之憑證及其他參數的名稱和路徑。Script 中的註解說明了這些設定。

```
@echo off
@rem Set the path where you installed the MQTT SDK
@rem and short cuts to the samples directories.
set SDKRoot=C:\MQTT
set jsamppath=%SDKRoot%\sdk\clients\java\samples
set csamppath=%SDKRoot%\sdk\clients\c\samples
```

```
@rem Set the paths to Version 7 of the JDK
@rem and to the directory where you built the openSSL package.
@rem Set short cuts to the tools.
set javapath=C:\Program Files\IBM\Java70
set keytool="%javapath%\jre\bin\keytool.exe"
set ikeyman="%javapath%\jre\bin\ikeyman.exe"
set openssl=%SDKRoot%\openSSL
set runopenssl="%openssl%\bin\openssl"
```

```
@rem Set the path to where certificates are to be stored,
@rem and set global security parameters.
@rem Omit set password, and the security tools prompt you for passwords.
@rem Validity is the expiry time of the certificates in days.
set certpath=%SDKRoot%\Certificates
set password=password
set validity=5000
set algorithm=RSA
```

```
@rem Set the certificate authority (CA) jks keystore and certificate parameters.
@rem Omit this step, unless you are defining your own certificate authority.
@rem The CA keystore contains the key-pair for your own certificate authority.
@rem You must protect the CA keystore.
@rem The CA certificate is the self-signed certificate authority public certificate.
@rem It is commonly known as the CA root certificate.
set caalias=caalias
set cadname="CN=mqttca.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cakeypass=%password%
@rem ca key store
set cajkskeystore=%certpath%\cakeystore.jks
set cajkskeystorepass=%password%
@rem ca certificate (root certificate)
set cacert=%certpath%\cacert.cer
```

```
@rem Set the server jks keystore and certificate parameters.
@rem The server keystore contains the key-pair for the server.
@rem You must protect the server keystore.
@rem If you then export the server certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the server keystore for the server key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the server certificate to the CA.
@rem When you now export the server certificate,
@rem the exported certificate includes the certificate chain.
set srvalias=srvalias
set srvdname="CN=mqttserver.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set srvkeypass=%password%
@rem server key stores
set srvjkskeystore=%certpath%\srvkeystore.jks
set srvjkskeystorepass=%password%
@rem server certificates
set srvcertreq=%certpath%\srvcertreq.csr
set srvcertassigned=%certpath%\srvcertassigned.cer
set srvcertselfsigned=%certpath%\srvcertselfsigned.cer
```

```
@rem Set the client jks keystore and certificate parameters
@rem Omit this step, unless you are authenticating clients.
@rem The client keystore contains the key-pair for the client.
@rem You must protect the client keystore.
@rem If you then export the client certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the client keystore for the client key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
```

```

@rem The certificate chain links the client certificate to the CA.
@rem When you now export the client certificate,
@rem the exported certificate includes the certificate chain.
set cltalias=cltalias
set cltdname="CN=mqttclient.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cltkeypass=%password%
@rem client key stores
set cltjkskeystore=%certpath%\cltkeystore.jks
set cltjkskeystorepass=%password%
set cltcertreq=%certpath%\cltcertreq.csr
set cltcertassigned=%certpath%\cltcertsigned.cer
set cltcertselfsigned=%certpath%\cltcertselfsigned.cer

```

```

@rem Set the paths to the client truststores signed by CA and signed by server key.
@rem You only need to define one of the trust stores.
@rem A trust store holds certificates that you trust,
@rem which are used to authenticate untrusted certificates.
@rem In this example, when the client authenticates the MQTT server it connects to,
@rem it authenticates the certificate it is sent by the server
@rem with the certificates in its trust store.
@rem For example, the MQTT server sends its server certificate,
@rem and the client authenticates it with either the same server certificate
@rem that you have stored in the cltsrvtruststore.jks trust store,
@rem or against the CA certificate, if the server certificate is signed by the CA.
set cltcajkstruststore=%certpath%\cltcatruststore.jks
set cltcajkstruststorepass=%password%
set cltsrvjkstruststore=%certpath%\cltsrvtruststore.jks
set cltsrvjkstruststorepass=%password%

```

```

@rem Set the paths to the client PKCS12 and PEM key and trust stores.
@rem Omit this step, unless you are configuring a C or iOS client.
@rem You only need to define either one of the trust stores for storing CA
@rem or server signed server certificates.
set cltp12keystore=%certpath%\cltkeystore.p12
set cltp12keystorepass=%password%
set cltpemkeystore=%certpath%\cltkeystore.pem
set cltpemkeystorepass=%password%
set cltcap12truststore=%certpath%\cltcatruststore.p12
set cltcap12truststorepass=%password%
set cltcapemtruststore=%certpath%\cltcatruststore.pem
set cltcapemtruststorepass=%password%
set cltsrvp12truststore=%certpath%\cltsrvtruststore.p12
set cltsrvp12truststorepass=%password%
set cltsrvpemtruststore=%certpath%\cltsrvtruststore.pem
set cltsrvpemtruststorepass=%password%

```

```

@rem set WMQ Variables
set authopt=NEVER
set authreq=REQUIRED
set qm=MQXR_SAMPLE_QM
set host=localhost
set mcauser='Guest'
set portsslopt=8884
set chllopt=SSLOPT
set portsslreq=8885
set chlreq=SSLREQ
V 7.5.0.1 set portws=1886
set chlws=PLAINWS
set chlssloptws=SSLOPTWS
set portssloptws=8886
set chlsslreqws=SSLREQWS
set portsslreqws=8887
set mqlog=%certpath%\wmq.log

```

cleancert.bat

cleancert.bat Script 中的指令會刪除 MQTT 用戶端佇列管理程式以確保伺服器憑證儲存庫未鎖定，然後會刪除範例安全 Script 建立的所有金鑰儲存庫及憑證。

```

@rem Delete the MQTT sample queue manager, MQXR_SAMPLE_QM
call "%MQ_FILE_PATH%\bin\setmqenv" -s
endmqm -i %qm%
dlmqm %qm%

```

```

@rem Erase all the certificates and key stores created by the sample scripts.
erase %cajkskeystore%

```

```

erase %cacert%
erase %srvjkskeystore%
erase %svrcertreq%
erase %svrcertcasigned%
erase %svrcertselfsigned%
erase %cltjkskeystore%
erase %cltp12keystore%
erase %cltpemkeystore%
erase %cltcertreq%
erase %cltcertcasigned%
erase %cltcertselfsigned%
erase %cltcajkstruststore%
erase %cltcap12truststore%
erase %cltcapemtruststore%
erase %cltsrvjkstruststore%
erase %cltsrvp12truststore%
erase %cltsrvpemtruststore%
erase %mqlog%
@echo Cleared all certificates
dir %certpath%\*.* /b

```

genkeys.bat

genkeys.bat Script 中的指令會為您的專用憑證管理中心、伺服器及用戶端建立金鑰組。

```

@rem
@echo -----
@echo Generate %caalias%, %srvalias%, and %cltalias% key-pairs in %cajkskeystore%,
%srvjkskeystore%, and %cltjkskeystore%
@rem
@rem -- Generate a client certificate and a private key pair
@rem Omit this step, unless you are authenticating clients.
%keytool% -genkeypair -noprompt -alias %cltalias% -dname %cltdname% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass% -keypass %cltkeypass% -keyalg %algorithm%
-validity %validity%

```

```

@rem -- Generate a server certificate and private key pair
%keytool% -genkeypair -noprompt -alias %srvalias% -dname %srvdname% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass% -keypass %srvkeypass% -keyalg %algorithm%
-validity %validity%

```

```

@rem Create CA, client and server key-pairs
@rem -- Generate a CA certificate and private key pair - The extension asserts this is a
certificate authority certificate, which is required to import into firefox
%keytool% -genkeypair -noprompt -ext bc=ca:true -alias %caalias% -dname %cadname%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass% -keyalg
%algorithm% -validity %validity%

```

sscerts.bat

sscerts.bat Script 中的指令會從其金鑰儲存庫匯出用戶端及伺服器自簽憑證，然後將伺服器憑證匯入用戶端信任儲存庫，並將用戶端憑證匯入伺服器金鑰儲存庫。伺服器沒有信任儲存庫。這些指令會從用戶端 JKS 信任儲存庫，以 PEM 的格式建立用戶端信任儲存庫。

```

@rem
@echo -----
@echo Export self-signed certificates: %svrcertselfsigned% and %cltcertselfsigned%
@rem Export Server public certificate
%keytool% -exportcert -noprompt -rfc -alias %srvalias% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass% -file %svrcertselfsigned%
@rem Export Client public certificate
@rem Omit this step, unless you are authenticating clients.
%keytool% -exportcert -noprompt -rfc -alias %cltalias% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass% -file %cltcertselfsigned%

```

```

@rem
@echo -----
@echo Add selfsigned server certificate %svrcertselfsigned% to client trust store:
%cltsrvjkstruststore%
@rem Import the server certificate into the client-server trust store (for server self-
signed authentication)
%keytool% -import -noprompt -alias %srvalias% -file %svrcertselfsigned% -keystore
%cltsrvjkstruststore% -storepass %cltsrvjkstruststorepass%

```

```

@rem
@echo -----
@echo Add selfsigned client certificate %cltcertselfsigned% to server trust store:
%srvjkskeystore%
@rem Import the client certificate into the server trust store (for client self-signed
authentication)
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %cltalias% -file %cltcertselfsigned% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%

```

```

@rem
@echo -----
@echo Create a pem client-server trust store from the jks client-server trust store:
%cltsrvpemtruststore%
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltsrvjkstruststore% -destkeystore
%cltsrvp12truststore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltsrvjkstruststorepass% -deststorepass %cltsrvp12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltsrvp12truststore% -out %cltsrvpemtruststore% -passin
pass:%cltsrvp12truststorepass% -passout pass:%cltsrvpemtruststorepass%@rem
@rem
@echo -----
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore% -destkeystore
%cltp12keystore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass% -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore% -out %cltpemkeystore% -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%

```

cacerts.bat

此 Script 會將憑證管理中心主要憑證匯入專用金鑰儲存庫。需要 CA 主要憑證，才可在主要憑證與已簽章的憑證之間建立金鑰鏈。cacerts.bat Script 會從其金鑰儲存庫匯出用戶端及伺服器憑證申請。Script 會使用 cajkskeystore.jks 金鑰儲存庫中專用憑證管理中心的金鑰，來簽署憑證申請，然後將已簽章的憑證匯回提出申請的同一個金鑰儲存庫。匯入會使用 CA 主要憑證建立憑證鏈。Script 會從用戶端 JKS 信任儲存庫，以 PEM 的格式建立用戶端信任儲存庫。

```

@rem
@echo -----
@echo Export self-signed certificates: %cacert%
@rem
@rem Export CA public certificate
%keytool% -exportcert -noprompt -rfc -alias %caalias% -keystore %cajkskeystore% -storepass
%cajkskeystorepass% -file %cacert%

```

```

@rem
@echo -----
@echo Add CA to server key and client key and trust stores: %srvjkskeystore%,
%cltjkskeystore%, %cltcajkstruststore%,
@rem The CA certificate is necessary to create key chains in the client and server key
stores,
@rem and to certify key chains in the server key store and the client trust store
@rem
@rem Import the CA root certificate into the server key store
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass%
@rem Import the CA root certificate into the client key store
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass%
@rem Import the CA root certificate into the client ca-trust store (for ca chained
authentication)
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltcajkstruststore%
-storepass %cltcajkstruststorepass%

```

```

@rem
@echo -----
@echo Create certificate signing requests: %srvcertreq% and %cltcertreq%
@rem
@rem Create a certificate signing request (CSR) for the server key
%keytool% -certreq -alias %srvalias% -file %srvcertreq% -keypass %srvkeypass% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Create a certificate signing request (CSR) for the client key

```

```
%keytool% -certreq -alias %cltalias% -file %cltcertreq% -keypass %cltkeypass% -keystore %cltjkskeystore% -storepass %cltjkskeystorepass%
```

```
@rem
@echo -----
@echo Sign certificate requests: %srvcertassigned% and %cltcertassigned%
@rem The requests are signed with the ca key in the cajkskeystore.jks keystore
@rem
@rem Sign server certificate request
%keytool% -gencert -infile %srvcertreq% -outfile %srvcertassigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
@rem Sign client certificate request
@rem Omit this step, unless you are authenticating clients.
%keytool% -gencert -infile %cltcertreq% -outfile %cltcertassigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
```

```
@rem
@echo -----
@echo Import the signed certificates back into the key stores to create the key chain:
%srvjkskeystore% and %cltjkskeystore%
@rem
@rem Import the signed server certificate
%keytool% -import -noprompt -alias %srvalias% -file %srvcertassigned% -keypass %srvkeypass%
-keystore %srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Import the signed client certificate and key chain back into the client keystore
%keytool% -import -noprompt -alias %cltalias% -file %cltcertassigned% -keypass %cltkeypass%
-keystore %cltjkskeystore% -storepass %cltjkskeystorepass%
@rem
@rem The CA certificate is needed in the server key store, and the client trust store
@rem to verify the key chain sent from the client or server
@echo Delete the CA certificate from %cltjkskeystore%: it causes a problem in converting
keystore to pem
@rem Omit this step, unless you are authenticating clients.
%keytool% -delete -alias %caalias% -keystore %cltjkskeystore% -storepass
%cltjkskeystorepass%
```

```
@rem
@echo -----
@echo Create a pem client-ca trust store from the jks client-ca trust store:
%cltcapemtruststore%
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltcajkskeystore% -destkeystore
%cltcap12truststore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltcajkskeystorepass% -deststorepass %cltcap12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltcap12truststore% -out %cltcapemtruststore% -passin
pass:%cltcap12truststorepass% -passout pass:%cltpeptruststorepass%
```

```
@rem
@echo -----
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore% -destkeystore
%clt12keystore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass% -deststorepass %clt12keystorepass%
%openssl%\bin\openssl pkcs12 -in %clt12keystore% -out %cltpeakeystore% -passin
pass:%clt12keystorepass% -passout pass:%cltpeakeystorepass%
```

mqcerts.bat

Script 會列出憑證目錄中的金鑰儲存庫及憑證。然後，它會建立 MQTT 範例佇列管理程式，並配置安全的遙測通道。

```
@echo -----
@echo List keystores and certificates
dir %certpath%\*.* /b
```

```
@rem
@echo Create queue manager and define mqtt channels and certificate stores
call "%MQ_FILE_PATH%\mqxr\Samples\SampleMQM" >> %mqlog%
echo DEFINE CHANNEL(%chlreq%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreq%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlopt%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslopt%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
```

```
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
V7.5.0.1
echo DEFINE CHANNEL(%chlsslreqws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreqws%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlsslptws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslptws%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portws%)
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
@echo MQ logs saved in %mqlog%echo
```

建置及執行安全的 MQTT 用戶端範例 C 應用程式

您可以根據 Windows 範例，在可為其編譯 C 原始檔的任何作業系統上，建立及執行安全的範例 C 應用程式。請驗證您可以在 IBM MessageSight 或 IBM WebSphere MQ 作為 MQTT 伺服器上執行範例 C 應用程式。

開始之前

1. 您必須能夠存取支援 MQTT protocol over SSL 的 MQTT version 3.1 伺服器。
2. 如果用戶端與伺服器之間有防火牆，請確認它不會封鎖 MQTT 資料流量。
3. 為許多作業系統提供適用於 C 的用戶端程式庫的二進位版本。對於其中部分作業系統，未提供安全的用戶端版本作為二進位檔。對於這些作業系統，您必須遵循第 28 頁的『建置適用於 C 的 MQTT 用戶端程式庫』中的指示。
4. 為了解決問題，IBM 支援中心可能會要求您在參照平台上，執行適用於 C 的 MQTT 用戶端。
5. SSL 通道必須已啟動。

如需受支援及參照平台的概觀，請參閱 [IBM Mobile Messaging and M2M 用戶端套件的系統需求](#)。如需 C 用戶端所支援項目的詳細資料，請參閱 [System Requirements for WebSphere MQ V7.5 Telemetry](#) 中的相關小節。

關於這項作業

作為圖解，本文顯示如何從指令行編譯及執行 Windows 上的安全 MQTT 用戶端範例 C 應用程式。在圖解中，將使用 Microsoft Visual Studio 2010 來編譯用戶端。您可以修改指令行 Script，以在其他作業系統（例如 Linux 及 iOS）上執行範例應用程式。

註：

本文中所提供的 Windows Script，假設您從原始檔建置整個 OpenSSL 套件。如果您選擇使用 IBM 提供的經過前置編譯的程式庫，則您還可能希望取得 OpenSSL 的經過前置編譯的二進位版本。iOS 無法使用經過前置編譯的程式庫。

使用憑證管理中心簽署的金鑰或自簽金鑰保護 SSL 通道。

程序

1. 選擇您可以連接用戶端應用程式的 MQTT 伺服器。

伺服器必須支援透過 SSL 的 MQTT version 3.1 通訊協定。IBM 中的所有 MQTT 伺服器都會這麼做，包括 IBM WebSphere MQ 和 IBM MessageSight。請參閱第 122 頁的『開始使用 MQTT 伺服器』。

2. 選擇性的：安裝第 7 版或更新版本的 Java 開發套件 (JDK)。

第 7 版，才能執行 **keytool** 指令來認證憑證。如果您不打算對憑證進行認證，則不需要第 7 版 JDK。

3. 在您要建置的平台上安裝 C 開發環境。

本主題的範例中的 make 檔將下列工具設為目標：

- **iOS** 若為 iOS，在具有 OS X 10.8.2 的 Apple Mac 上，使用 [Xcode](#) 中的 iOS 開發工具。
- **Linux** 對於 Linux，為 Red Hat Enterprise Linux 6.2 版中的 gcc 4.4.6 版。

glibc C 程式庫的最低支援層次為 2.12，Linux 核心的最低支援層次為 2.6.32。

- **Windows** 對於 Microsoft Windows，為 Visual Studio 10.0 版。
4. 下載 Mobile Messaging and M2M 用戶端套件 並安裝 MQTT SDK。
沒有安裝程式，您只是展開了下載的檔案。
 - a. 下載 Mobile Messaging and M2M 用戶端套件。
 - b. 建立您要安裝 SDK 的資料夾。
您可能想將資料夾命名為 MQTT。在這裡，此資料夾的路徑稱為 *sdkroot*。
 - c. 將壓縮的 Mobile Messaging and M2M 用戶端套件 檔案內容展開到 *sdkroot* 中。展開會建立以 *sdkroot*\SDK 開頭的目錄樹。
 5. 選擇性的: 遵循 第 28 頁的『建置適用於 C 的 MQTT 用戶端程式庫』中的步驟。
請僅在 MQTT SDK 不包含適用於目標作業系統的安全 C 用戶端程式庫時，才執行此步驟。
 - **Windows** 程式庫為 *mqttv3cs.lib* (用於編譯) 及 *mqttv3cs.dll* (用於執行)。
 - **Linux** 程式庫為 *libmqttv3cs.so*
 - **iOS** 程式庫為 *libmqttv3cs.a*
 6. 建立並執行 Script 以產生金鑰組和憑證，並將 IBM WebSphere MQ 配置為 MQTT 伺服器。
遵循 第 88 頁的『產生金鑰及憑證』中的步驟來建立並執行 Script。第 82 頁的『為 Windows 配置 SSL 憑證的 Script 範例』中也列有這些 Script。
 7. 檢查 SSL 通道是否正在執行，且如您的預期設定。
在 IBM WebSphere MQ 上，於指令視窗中鍵入下列指令：
 - **Linux**

```
echo 'DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
echo 'DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
```
 - **Windows**

```
echo DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM
echo DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM
```
 8. 建立 Script 以建置及執行安全的 MQTT 用戶端範例 C 應用程式。
 - a) 建立及執行 sscclient.bat，以測試受到自簽憑證保護的 SSL 通道。
 - b) 建立及執行 cacclient.bat，以測試受到自簽憑證保護的 SSL 通道。

結果

結果類似於執行未受保護的用戶端。

```
Connected to ssl://localhost:8884
Subscribing to topic "MQTTV3SSample/#" qos 2
Topic:           MQTTV3SSample/C/v3
Message:         Message from MQTTv3 SSL C client
QoS:             2
Connected to ssl://localhost:8885
Subscribing to topic "MQTTV3SSample/#" qos 2
Topic:           MQTTV3SSample/C/v3
Message:         Message from MQTTv3 SSL C client
QoS:             2
```

圖 16: 安全的訂閱者


```

Setting environment for using Microsoft Visual Studio 2010 x86 tools.
MQTTV3SSample.c
Connected to ssl://localhost:8884
Publishing to topic "MQTTV3SSample/C/v3" qos 2
Disconnected
Press any key to continue . . .
Connected to ssl://localhost:8885
Publishing to topic "MQTTV3SSample/C/v3" qos 2
Disconnected
Press any key to continue . . .

```

圖 17: 安全的發佈者

用於執行安全 MQTT 用戶端範例 C 應用程式的 Script

先執行第 82 頁的『為 Windows 配置 SSL 憑證的 Script 範例』中的 Script，然後再執行這些 Script。

利用自簽憑證保護 MQTT 用戶端範例 C 應用程式的安全。

使用您利用執行 `sscerts.bat` Script 所建立的自簽憑證來執行此 Script。

```

@echo off
setlocal
cd %csamppath%
erase MQTTV3SSample.obj
erase MQTTV3SSample.exe
call "C:\Program Files\Microsoft Visual Studio 10.0\VC\vcvarsall.bat" x86
cl /nologo /D "WIN32" /I ".\include" "MQTTV3SSample.c" /link /
nologo ..\windows_ia32\mqttv3cs.lib
set path=%path%;%csamppath%\..\windows_ia32
ping -n 2 127.0.0.1 > NUL 2>&1
@echo start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportopt% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportopt% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
@echo MQTTV3SSample -b %host% -p %sslportopt% -k %cltpemkeystore% -w %cltpemkeystorepass%
-r %cltsrvpemtruststore% -v 1
MQTTV3SSample -b %host% -p %sslportopt% -k %cltpemkeystore% -w %cltpemkeystorepass% -r
%cltsrvpemtruststore% -v 1
pause
ping -n 2 127.0.0.1 > NUL 2>&1
@echo start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportreq% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportreq% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
@echo MQTTV3SSample -b %host% -p %sslportreq% -k %cltpemkeystore% -w %cltpemkeystorepass%
-r %cltsrvpemtruststore% -v 1
MQTTV3SSample -b %host% -p %sslportreq% -k %cltpemkeystore% -w %cltpemkeystorepass% -r
%cltsrvpemtruststore% -v 1
pause
ping -n 2 127.0.0.1 > NUL 2>&1
endlocal

```

圖 18: `ssclient.bat`

利用憑證管理中心簽署的憑證，執行 MQTT 安全用戶端範例 C 應用程式。

使用您利用執行 `cacerts.bat` Script 所建立之憑證管理中心簽署的憑證來執行此 Script。

```

@echo off
setlocal
cd %csamppath%
erase MQTTV3SSample.obj
erase MQTTV3SSample.exe
call "C:\Program Files\Microsoft Visual Studio 10.0\VC\vcvarsall.bat" x86
cl /nologo /D "WIN32" /I "..\include" "MQTTV3SSample.c" /link /
nologo ..\windows_ia32\mqttv3cs.lib
set path=%path%;%csamppath%\..\windows_ia32
ping -n 2 127.0.0.1 > NUL 2>&1
@echo start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportopt% -k
%cltpekeystore% -w %cltpekeystorepass% -r %cltsrvpemtruststore% -v 1
start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportopt% -k
%cltpekeystore% -w %cltpekeystorepass% -r %cltsrvpemtruststore% -v 1
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
@echo MQTTV3SSample -b %host% -p %sslportopt% -k %cltpekeystore% -w %cltpekeystorepass%
-r %cltsrvpemtruststore% -v 1
MQTTV3SSample -b %host% -p %sslportopt% -k %cltpekeystore% -w %cltpekeystorepass% -r
%cltsrvpemtruststore% -v 1
pause
ping -n 2 127.0.0.1 > NUL 2>&1
@echo start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportreq% -k
%cltpekeystore% -w %cltpekeystorepass% -r %cltsrvpemtruststore% -v 1
start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportreq% -k
%cltpekeystore% -w %cltpekeystorepass% -r %cltsrvpemtruststore% -v 1
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
@echo MQTTV3SSample -b %host% -p %sslportreq% -k %cltpekeystore% -w %cltpekeystorepass%
-r %cltsrvpemtruststore% -v 1
MQTTV3SSample -b %host% -p %sslportreq% -k %cltpekeystore% -w %cltpekeystorepass% -r
%cltsrvpemtruststore% -v 1
pause
ping -n 2 127.0.0.1 > NUL 2>&1
endlocal

```

圖 19: cacclient.bat

相關概念

第 47 頁的『MQTT 安全』

以下是 MQTT 安全的三個基本概念：身分、鑑別及授權。身分可為要授權及提供權限的用戶端命名。鑑別可提供用戶端的身分，而授權可管理為用戶端提供的權限。

相關工作

第 88 頁的『產生金鑰及憑證』

遵循下列程序，為 Java 和 C 用戶端（包括 Android 及 iOS 應用程式）以及 IBM WebSphere MQ 和 IBM MessageSight 伺服器產生金鑰及憑證。

為 Windows 配置 SSL 憑證的 Script 範例

範例

指令檔範例會建立憑證及憑證儲存庫，如作業中的步驟所述。此外，範例還會將 MQTT 用戶端佇列管理程式設為使用伺服器憑證儲存庫。範例會呼叫 IBM WebSphere MQ 隨附的 SampleMQM.bat Script，來刪除並重建佇列管理程式。

initcert.bat

initcert.bat 會設定 **keytool** 及 **openSSL** 指令所需之憑證及其他參數的名稱和路徑。Script 中的註解說明了這些設定。

```

@echo off
@rem Set the path where you installed the MQTT SDK
@rem and short cuts to the samples directories.
set SDKRoot=C:\MQTT
set jsamppath=%SDKRoot%\sdk\clients\java\samples
set csamppath=%SDKRoot%\sdk\clients\c\samples

@rem Set the paths to Version 7 of the JDK
@rem and to the directory where you built the openSSL package.
@rem Set short cuts to the tools.
set javapath=C:\Program Files\IBM\Java70

```

```
set keytool="%javapath%\jre\bin\keytool.exe"
set ikeyman="%javapath%\jre\bin\ikeyman.exe"
set openssl=%SDKRoot%\openssl
set runopenssl="%openssl%\bin\openssl"
```

```
@rem Set the path to where certificates are to be stored,
@rem and set global security parameters.
@rem Omit set password, and the security tools prompt you for passwords.
@rem Validity is the expiry time of the certificates in days.
set certpath=%SDKRoot%\Certificates
set password=password
set validity=5000
set algorithm=RSA
```

```
@rem Set the certificate authority (CA) jks keystore and certificate parameters.
@rem Omit this step, unless you are defining your own certificate authority.
@rem The CA keystore contains the key-pair for your own certificate authority.
@rem You must protect the CA keystore.
@rem The CA certificate is the self-signed certificate authority public certificate.
@rem It is commonly known as the CA root certificate.
set caalias=caalias
set cadname="CN=mqttca.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cakeypass=%password%
@rem ca key store
set cajkskeystore=%certpath%\cakeystore.jks
set cajkskeystorepass=%password%
@rem ca certificate (root certificate)
set cacert=%certpath%\cacert.cer
```

```
@rem Set the server jks keystore and certificate parameters.
@rem The server keystore contains the key-pair for the server.
@rem You must protect the server keystore.
@rem If you then export the server certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the server keystore for the server key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the server certificate to the CA.
@rem When you now export the server certificate,
@rem the exported certificate includes the certificate chain.
set srvalias=srvalias
set srvdname="CN=mqttserver.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set srvkeypass=%password%
@rem server key stores
set srvjkskeystore=%certpath%\srvkeystore.jks
set srvjkskeystorepass=%password%
@rem server certificates
set srvcertreq=%certpath%\srvcertreq.csr
set srvcertcasigned=%certpath%\srvcertcasigned.cer
set srvcertselfsigned=%certpath%\srvcertselfsigned.cer
```

```
@rem Set the client jks keystore and certificate parameters
@rem Omit this step, unless you are authenticating clients.
@rem The client keystore contains the key-pair for the client.
@rem You must protect the client keystore.
@rem If you then export the client certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the client keystore for the client key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the client certificate to the CA.
@rem When you now export the client certificate,
@rem the exported certificate includes the certificate chain.
set cltalias=cltalias
set cltdname="CN=mqttclient.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cltkeypass=%password%
@rem client key stores
set cltjkskeystore=%certpath%\cltkeystore.jks
set cltjkskeystorepass=%password%
set cltcertreq=%certpath%\cltcertreq.csr
set cltcertcasigned=%certpath%\cltcacertsigned.cer
set cltcertselfsigned=%certpath%\cltcertselfsigned.cer
```

```
@rem Set the paths to the client truststores signed by CA and signed by server key.
@rem You only need to define one of the trust stores.
@rem A trust store holds certificates that you trust,
@rem which are used to authenticate untrusted certificates.
```

```

@rem In this example, when the client authenticates the MQTT server it connects to,
@rem it authenticates the certificate it is sent by the server
@rem with the certificates in its trust store.
@rem For example, the MQTT server sends its server certificate,
@rem and the client authenticates it with either the same server certificate
@rem that you have stored in the cltsrvtruststore.jks trust store,
@rem or against the CA certificate, if the server certificate is signed by the CA.
set cltcajkstruststore=%certpath%\cltcatruststore.jks
set cltcajkstruststorepass=%password%
set cltsrvjkstruststore=%certpath%\cltsrvtruststore.jks
set cltsrvjkstruststorepass=%password%

```

```

@rem Set the paths to the client PKCS12 and PEM key and trust stores.
@rem Omit this step, unless you are configuring a C or iOS client.
@rem You only need to define either one of the trust stores for storing CA
@rem or server signed server certificates.
set cltp12keystore=%certpath%\cltkeystore.p12
set cltp12keystorepass=%password%
set cltpemkeystore=%certpath%\cltkeystore.pem
set cltpemkeystorepass=%password%
set cltcap12truststore=%certpath%\cltcatruststore.p12
set cltcap12truststorepass=%password%
set cltcapemtruststore=%certpath%\cltcatruststore.pem
set cltcapemtruststorepass=%password%
set cltsrvp12truststore=%certpath%\cltsrvtruststore.p12
set cltsrvp12truststorepass=%password%
set cltsrvpemtruststore=%certpath%\cltsrvtruststore.pem
set cltsrvpemtruststorepass=%password%

```

```

@rem set WMQ Variables
set authopt=NEVER
set authreq=REQUIRED
set qm=MQXR_SAMPLE_QM
set host=localhost
set mcauser='Guest'
set portsslopt=8884
set chlopt=SSLOPT
set portsslreq=8885
set chlreq=SSLREQ
V7.5.0.1 set portws=1886
set chlws=PLAINWS
set chlsslloptws=SSLOPTWS
set portsslloptws=8886
set chlsslreqws=SSLREQWS
set portsslreqws=8887
set mqlog=%certpath%\wmq.log

```

cleancert.bat

cleancert.bat Script 中的指令會刪除 MQTT 用戶端佇列管理程式以確保伺服器憑證儲存庫未鎖定，然後會刪除範例安全 Script 建立的所有金鑰儲存庫及憑證。

```

@rem Delete the MQTT sample queue manager, MQXR_SAMPLE_QM
call "%MQ_FILE_PATH%\bin\setmqenv" -s
endmqm -i %qm%
dltmqm %qm%

```

```

@rem Erase all the certificates and key stores created by the sample scripts.
erase %cajkskeystore%
erase %cacert%
erase %srvjkskeystore%
erase %svrcertreq%
erase %svrcertcasigned%
erase %svrcertselfsigned%
erase %cltjkskeystore%
erase %cltp12keystore%
erase %cltpemkeystore%
erase %cltcertreq%
erase %cltcertcasigned%
erase %cltcertselfsigned%
erase %cltcajkstruststore%
erase %cltcap12truststore%
erase %cltcapemtruststore%
erase %cltsrvjkstruststore%
erase %cltsrvp12truststore%
erase %cltsrvpemtruststore%
erase %mqlog%

```

```
@echo Cleared all certificates
dir %certpath%\*.* /b
```

genkeys.bat

genkeys.bat Script 中的指令會為您的專用憑證管理中心、伺服器及用戶端建立金鑰組。

```
@rem
@echo -----
@echo Generate %caalias%, %srvalias%, and %cltalias% key-pairs in %cajkskeystore%,
%srvjkskeystore%, and %cltjkskeystore%
@rem
@rem -- Generate a client certificate and a private key pair
@rem Omit this step, unless you are authenticating clients.
%keytool% -genkeypair -noprompt -alias %cltalias% -dname %cltdname% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass% -keypass %cltkeypass% -keyalg %algorithm%
-validity %validity%
```

```
@rem -- Generate a server certificate and private key pair
%keytool% -genkeypair -noprompt -alias %srvalias% -dname %srvdname% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass% -keypass %srvkeypass% -keyalg %algorithm%
-validity %validity%
```

```
@rem Create CA, client and server key-pairs
@rem -- Generate a CA certificate and private key pair - The extension asserts this is a
certificate authority certificate, which is required to import into firefox
%keytool% -genkeypair -noprompt -ext bc=ca:true -alias %caalias% -dname %cadname%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass% -keyalg
%algorithm% -validity %validity%
```

sscerts.bat

sscerts.bat Script 中的指令會從其金鑰儲存庫匯出用戶端及伺服器自簽憑證，然後將伺服器憑證匯入用戶端信任儲存庫，並將用戶端憑證匯入伺服器金鑰儲存庫。伺服器沒有信任儲存庫。這些指令會從用戶端 JKS 信任儲存庫，以 PEM 的格式建立用戶端信任儲存庫。

```
@rem
@echo -----
@echo Export self-signed certificates: %svrcertselfsigned% and %cltcertselfsigned%
@rem Export Server public certificate
%keytool% -exportcert -noprompt -rfc -alias %srvalias% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass% -file %svrcertselfsigned%
@rem Export Client public certificate
@rem Omit this step, unless you are authenticating clients.
%keytool% -exportcert -noprompt -rfc -alias %cltalias% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass% -file %cltcertselfsigned%
```

```
@rem
@echo -----
@echo Add selfsigned server certificate %svrcertselfsigned% to client trust store:
%cltsrvjkstruststore%
@rem Import the server certificate into the client-server trust store (for server self-
signed authentication)
%keytool% -import -noprompt -alias %srvalias% -file %svrcertselfsigned% -keystore
%cltsrvjkstruststore% -storepass %cltsrvjkstruststorepass%
```

```
@rem
@echo -----
@echo Add selfsigned client certificate %cltcertselfsigned% to server trust store:
%srvjkskeystore%
@rem Import the client certificate into the server trust store (for client self-signed
authentication)
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %cltalias% -file %cltcertselfsigned% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
```

```
@rem
@echo -----
@echo Create a pem client-server trust store from the jks client-server trust store:
%cltsrvpemtruststore%
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltsrvjkstruststore% -destkeystore
%cltsrvp12truststore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
```

```

%cltsrvjkstruststorepass% -deststorepass %cltsrvp12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltsrvp12truststore% -out %cltsrvpemtruststore% -passin
pass:%cltsrvp12truststorepass% -passout pass:%cltsrvpemtruststorepass%@rem
@rem
@echo -----
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore% -destkeystore
%cltp12keystore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass% -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore% -out %cltpemkeystore% -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%

```

cacerts.bat

此 Script 會將憑證管理中心主要憑證匯入專用金鑰儲存庫。需要 CA 主要憑證，才可在主要憑證與已簽章的憑證之間建立金鑰鏈。cacerts.bat Script 會從其金鑰儲存庫匯出用戶端及伺服器憑證申請。Script 會使用 cajkskeystore.jks 金鑰儲存庫中專用憑證管理中心的金鑰，來簽署憑證申請，然後將已簽章的憑證匯回提出申請的同一個金鑰儲存庫。匯入會使用 CA 主要憑證建立憑證鏈。Script 會從用戶端 JKS 信任儲存庫，以 PEM 的格式建立用戶端信任儲存庫。

```

@rem
@echo -----
@echo Export self-signed certificates: %cacert%
@rem
@rem Export CA public certificate
%keytool% -exportcert -noprompt -rfc -alias %caalias% -keystore %cajkskeystore% -storepass
%cajkskeystorepass% -file %cacert%

```

```

@rem
@echo -----
@echo Add CA to server key and client key and trust stores: %srvjkskeystore%,
%cltjkskeystore%, %cltcajkstruststore%,
@rem The CA certificate is necessary to create key chains in the client and server key
stores,
@rem and to certify key chains in the server key store and the client trust store
@rem
@rem Import the CA root certificate into the server key store
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass%
@rem Import the CA root certificate into the client key store
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass%
@rem Import the CA root certificate into the client ca-trust store (for ca chained
authentication)
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltcajkstruststore%
-storepass %cltcajkstruststorepass%

```

```

@rem
@echo -----
@echo Create certificate signing requests: %srvcertreq% and %cltcertreq%
@rem
@rem Create a certificate signing request (CSR) for the server key
%keytool% -certreq -alias %srvalias% -file %srvcertreq% -keypass %srvkeypass% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Create a certificate signing request (CSR) for the client key
%keytool% -certreq -alias %cltalias% -file %cltcertreq% -keypass %cltkeypass% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass%

```

```

@rem
@echo -----
@echo Sign certificate requests: %srvcertcasigned% and %cltcertcasigned%
@rem The requests are signed with the ca key in the cajkskeystore.jks keystore
@rem
@rem Sign server certificate request
%keytool% -gencert -infile %srvcertreq% -outfile %srvcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
@rem Sign client certificate request
@rem Omit this step, unless you are authenticating clients.
%keytool% -gencert -infile %cltcertreq% -outfile %cltcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%

```

```

@rem
@echo -----
@echo Import the signed certificates back into the key stores to create the key chain:
%srvjkskeystore% and %cltjkskeystore%
@rem
@rem Import the signed server certificate
%keytool% -import -noprompt -alias %srvalias% -file %svrcertcasigned% -keypass %srvkeypass%
-keystore %srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Import the signed client certificate and key chain back into the client keystore
%keytool% -import -noprompt -alias %cltalias% -file %cltcertcasigned% -keypass %cltkeypass%
-keystore %cltjkskeystore% -storepass %cltjkskeystorepass%
@rem
@rem The CA certificate is needed in the server key store, and the client trust store
@rem to verify the key chain sent from the client or server
@echo Delete the CA certificate from %cltjkskeystore%: it causes a problem in converting
keystore to pem
@rem Omit this step, unless you are authenticating clients.
%keytool% -delete -alias %caalias% -keystore %cltjkskeystore% -storepass
%cltjkskeystorepass%

```

```

@rem
@echo -----
@echo Create a pem client-ca trust store from the jks client-ca trust store:
%cltcapemtruststore%
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltcajkstruststore% -destkeystore
%cltcap12truststore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltcajkstruststorepass% -deststorepass %cltcap12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltcap12truststore% -out %cltcapemtruststore% -passin
pass:%cltcap12truststorepass% -passout pass:%cltpemtruststorepass%

```

```

@rem
@echo -----
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore% -destkeystore
%cltp12keystore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass% -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore% -out %cltpemkeystore% -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%

```

mqcerts.bat

Script 會列出憑證目錄中的金鑰儲存庫及憑證。然後，它會建立 MQTT 範例佇列管理程式，並配置安全的遙測通道。

```

@echo -----
@echo List keystores and certificates
dir %certpath%\*.* /b

```

```

@rem
@echo Create queue manager and define mqtt channels and certificate stores
call "%MQ_FILE_PATH%\mqxr\Samples\SampleMQM" >> %mqlog%
echo DEFINE CHANNEL(%chlreq%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreq%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlopt%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslopt%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
V7.5.0.1
echo DEFINE CHANNEL(%chlsslreqws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreqws%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlssloptws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portssloptws%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portws%)
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
@echo MQ logs saved in %mqlog%echo

```

產生金鑰及憑證

遵循下列程序，為 Java 和 C 用戶端（包括 Android 及 iOS 應用程式）以及 IBM WebSphere MQ 和 IBM MessageSight 伺服器產生金鑰及憑證。

開始之前

1. 您必須具有 **keytool** 指令的副本。並非所有 **keytool** 版本都支援將金鑰儲存庫從 Java 金鑰儲存庫 (JKS) 轉換為「公開金鑰加密系統 (PKCS)」，或簽署憑證申請。此範例使用 JDK 7.0 版中的 **keytool** 指令（支援上述兩個功能）。
2. 如果要為適用於 C 的用戶端產生金鑰及憑證（「保密加強型郵件 (PEM)」格式），您必須具有 **openssl** 指令的副本。請遵循第 28 頁的『建置適用於 C 的 MQTT 用戶端程式庫』中的步驟來建置 **openssl** 套件。
3. 依照您的需求變更 **initcert.bat** Script 中的參數值。尤其是，您可以選擇省略密碼參數來避免記下密碼。**keytool** 指令會提示您遺漏密碼。

關於這項作業

您需要金鑰及憑證，以在 MQTT 用戶端與伺服器之間建立安全的 SSL 連線。此作業會為您顯示兩種不同的方法，來建立您需要的金鑰及憑證：自簽的及您自己的憑證管理中心簽署的。您遵循的方法取決於您計劃管理金鑰儲存庫及憑證的方式。

若要使用外部憑證管理中心簽署的憑證，請透過將憑證申請傳送至外部憑證管理中心，來取代 **cacerts.bat** 中的簽署步驟。除已簽章的憑證之外，憑證管理中心可能還會傳回中間憑證及主要憑證。請遵循安裝傳回的憑證所在的外部 CA 提供的指引。

IBM WebSphere MQ 伺服器僅在您於遙測通道配置參數中指定的憑證儲存庫中搜尋憑證。它不會另外在 JSE **cacerts** 儲存庫中搜尋。Java 用戶端在您指定的信任儲存庫中搜尋憑證。如果您未指定信任儲存庫，它會在 JSE **jre\lib\security** 目錄的 **cacerts** 金鑰儲存庫中搜尋。Android 用戶端在 Android 裝置上預先定義的憑證儲存庫中搜尋憑證。C 用戶端應用程式及 iOS 應用程式，僅會在應用程式指定的憑證儲存庫中進行搜尋。

Android 及 Java 用戶端在預先配置的信心儲存庫中搜尋授信憑證。CA 主要憑證儲存在 Android 授信憑證儲存庫及 JSE **jre\lib\security\cacerts** 儲存庫中。如果已在預先配置的信心儲存庫中安裝通過伺服器憑證認證的 CA 的主要憑證，請勿定義用戶端信心儲存庫。所需的唯一配置是為安全的 MQTT 伺服器通道設定 TCP/IP 埠。

建立金鑰及憑證及管理所有不同的格式的工具並不簡單易用。它們具有許多要管理的參數，**openssl** 需要配置檔、**openssl.cnf** 及指令行參數。沒有一個工具可提供管理 C 及 Java 上執行的應用程式的金鑰和憑證所需的所有功能。IBM WebSphere MQ 中的遙測通道需要 JKS 金鑰儲存庫，因此範例主要使用 Java 憑證工具，即 **keyman** 及 **keytool**。但是，Java 工具不支援 C 用戶端應用程式需要的 PEM 格式。若要以 PEM 格式建立金鑰儲存庫，請執行 **openssl** 工具。**openssl** 工具可將金鑰儲存庫從 PKCS12 格式轉換為 PEM 格式，**keytool** 可在 JKS 格式與 PKCS12 格式之間轉換金鑰儲存庫。金鑰儲存庫轉換不需要任何 **openssl.cnf** 檔案。只有在您計劃建置 C 用戶端應用程式或 iOS 應用程式時，才需要 **openssl**。如果您喜歡使用 **openssl**，可以使用它來簽署憑證，而不是使用 **keytool** 來簽署憑證。

程序

1. 開啟指令視窗以執行下列 Script。
2. 建立並執行 **initcert.bat** Script，以設定執行 MQTT 安全範例用戶端所需的參數。
3. 建立及執行 **cleancert.bat** Script，以清除環境來準備建立新的金鑰儲存庫及憑證。
4. 建立及執行 **genkeys.bat** Script，以產生所需要的金鑰組。
5. 執行下列其中一個選項：
 - 建立及執行 **sscerts.bat** Script，以建立自簽憑證。
 - 建立及執行 **cacerts.bat** Script，以建立憑證管理中心簽署的憑證鏈。
6. 建立及執行 **mqcerts.bat** Script，以建立 MQXR_SAMPLE_QM 佇列管理程式及配置其遙測通道。

相關工作

第 79 頁的『建置及執行安全的 MQTT 用戶端範例 C 應用程式』

您可以根據 Windows 範例，在可為其編譯 C 原始檔的任何作業系統上，建立及執行安全的範例 C 應用程式。請驗證您可以在 IBM MessageSight 或 IBM WebSphere MQ 作為 MQTT 伺服器上執行範例 C 應用程式。

第 50 頁的『建置及執行安全的 MQTT 用戶端範例 Java 應用程式』

根據 Windows 範例，您可以在 IBM MessageSight 或 IBM WebSphere MQ 作為 MQTT 伺服器上啟動並執行安全範例 Java 應用程式。您可以在任何具有 JSE 1.5 或更高版本 (即 "Java 相容") 的平台上執行適用於 Java 的 MQTT 用戶端應用程式

為 Windows 配置 SSL 憑證的 Script 範例

範例

指令檔範例會建立憑證及憑證儲存庫，如作業中的步驟所述。此外，範例還會將 MQTT 用戶端佇列管理程式設為使用伺服器憑證儲存庫。範例會呼叫 IBM WebSphere MQ 隨附的 SampleMQM.bat Script，來刪除並重建佇列管理程式。

initcert.bat

initcert.bat 會設定 **keytool** 及 **openssl** 指令所需之憑證及其他參數的名稱和路徑。Script 中的註解說明了這些設定。

```
@echo off
@rem Set the path where you installed the MQTT SDK
@rem and short cuts to the samples directories.
set SDKRoot=C:\MQTT
set jsamppath=%SDKRoot%\sdk\clients\java\samples
set csamppath=%SDKRoot%\sdk\clients\c\samples
```

```
@rem Set the paths to Version 7 of the JDK
@rem and to the directory where you built the openssl package.
@rem Set short cuts to the tools.
set javapath=C:\Program Files\IBM\Java70
set keytool="%javapath%\jre\bin\keytool.exe"
set ikeyman="%javapath%\jre\bin\ikeyman.exe"
set openssl=%SDKRoot%\openssl
set runopenssl="%openssl%\bin\openssl"
```

```
@rem Set the path to where certificates are to be stored,
@rem and set global security parameters.
@rem Omit set password, and the security tools prompt you for passwords.
@rem Validity is the expiry time of the certificates in days.
set certpath=%SDKRoot%\Certificates
set password=password
set validity=5000
set algorithm=RSA
```

```
@rem Set the certificate authority (CA) jks keystore and certificate parameters.
@rem Omit this step, unless you are defining your own certificate authority.
@rem The CA keystore contains the key-pair for your own certificate authority.
@rem You must protect the CA keystore.
@rem The CA certificate is the self-signed certificate authority public certificate.
@rem It is commonly known as the CA root certificate.
set caalias=caalias
set cadname="CN=mqttca.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cakeypass=%password%
@rem ca key store
set cajkskeystore=%certpath%\cakeystore.jks
set cajkskeystorepass=%password%
@rem ca certificate (root certificate)
set cacert=%certpath%\cacert.cer
```

```
@rem Set the server jks keystore and certificate parameters.
@rem The server keystore contains the key-pair for the server.
@rem You must protect the server keystore.
@rem If you then export the server certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the server keystore for the server key,
```

```

@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the server certificate to the CA.
@rem When you now export the server certificate,
@rem the exported certificate includes the certificate chain.
set svalias=srvalias
set srvidname="CN=mqttserver.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set srvkeypass=%password%
@rem server key stores
set srvjkskeystore=%certpath%\srvkeystore.jks
set srvjkskeystorepass=%password%
@rem server certificates
set srvcertreq=%certpath%\srvcertreq.csr
set srvcertcasigned=%certpath%\srvcertcasigned.cer
set srvcertselfsigned=%certpath%\srvcertselfsigned.cer

```

```

@rem Set the client jks keystore and certificate parameters
@rem Omit this step, unless you are authenticating clients.
@rem The client keystore contains the key-pair for the client.
@rem You must protect the client keystore.
@rem If you then export the client certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the client keystore for the client key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the client certificate to the CA.
@rem When you now export the client certificate,
@rem the exported certificate includes the certificate chain.
set cltalias=cltalias
set cltidname="CN=mqttclient.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cltkeypass=%password%
@rem client key stores
set cltjkskeystore=%certpath%\cltkeystore.jks
set cltjkskeystorepass=%password%
set cltcertreq=%certpath%\cltcertreq.csr
set cltcertcasigned=%certpath%\cltcacertsigned.cer
set cltcertselfsigned=%certpath%\cltcertselfsigned.cer

```

```

@rem Set the paths to the client truststores signed by CA and signed by server key.
@rem You only need to define one of the trust stores.
@rem A trust store holds certificates that you trust,
@rem which are used to authenticate untrusted certificates.
@rem In this example, when the client authenticates the MQTT server it connects to,
@rem it authenticates the certificate it is sent by the server
@rem with the certificates in its trust store.
@rem For example, the MQTT server sends its server certificate,
@rem and the client authenticates it with either the same server certificate
@rem that you have stored in the cltsrvtruststore.jks trust store,
@rem or against the CA certificate, if the server certificate is signed by the CA.
set cltcajkstruststore=%certpath%\cltcatruststore.jks
set cltcajkstruststorepass=%password%
set cltsrvjkstruststore=%certpath%\cltsrvtruststore.jks
set cltsrvjkstruststorepass=%password%

```

```

@rem Set the paths to the client PKCS12 and PEM key and trust stores.
@rem Omit this step, unless you are configuring a C or iOS client.
@rem You only need to define either one of the trust stores for storing CA
@rem or server signed server certificates.
set cltp12keystore=%certpath%\cltkeystore.p12
set cltp12keystorepass=%password%
set cltpemkeystore=%certpath%\cltkeystore.pem
set cltpemkeystorepass=%password%
set cltcap12truststore=%certpath%\cltcatruststore.p12
set cltcap12truststorepass=%password%
set cltcapemtruststore=%certpath%\cltcatruststore.pem
set cltcapemtruststorepass=%password%
set cltsrvp12truststore=%certpath%\cltsrvtruststore.p12
set cltsrvp12truststorepass=%password%
set cltsrvpemtruststore=%certpath%\cltsrvtruststore.pem
set cltsrvpemtruststorepass=%password%

```

```

@rem set WMQ Variables
set authopt=NEVER
set authreq=REQUIRED
set qm=MQXR_SAMPLE_QM
set host=localhost
set mcauser='Guest'
set portsslopt=8884

```

```

set chlopt=SSLOPT
set portsslreq=8885
set chlreq=SSLREQ
V 7.5.0.1 set portws=1886
set chlws=PLAINWS
set chlssloptws=SSLOPTWS
set portsslloptws=8886
set chlsslreqws=SSLREQWS
set portsslreqws=8887
set mqlog=%certpath%\wmq.log

```

cleancert.bat

cleancert.bat Script 中的指令會刪除 MQTT 用戶端佇列管理程式以確保伺服器憑證儲存庫未鎖定，然後會刪除範例安全 Script 建立的所有金鑰儲存庫及憑證。

```

@rem Delete the MQTT sample queue manager, MQXR_SAMPLE_QM
call "%MQ_FILE_PATH%\bin\setmqenv" -s
endmqm -i %qm%
dlmqm %qm%

```

```

@rem Erase all the certificates and key stores created by the sample scripts.
erase %cajkskeystore%
erase %cacert%
erase %srvjkskeystore%
erase %svrcertreq%
erase %svrcertcasigned%
erase %svrcertselfsigned%
erase %cltjkskeystore%
erase %cltp12keystore%
erase %cltpemkeystore%
erase %cltcertreq%
erase %cltcertcasigned%
erase %cltcertselfsigned%
erase %cltcajkstruststore%
erase %cltcap12truststore%
erase %cltcapemtruststore%
erase %cltsrvjkstruststore%
erase %cltsrvp12truststore%
erase %cltsrvpemtruststore%
erase %mqlog%
@echo Cleared all certificates
dir %certpath%\*.* /b

```

genkeys.bat

genkeys.bat Script 中的指令會為您的專用憑證管理中心、伺服器及用戶端建立金鑰組。

```

@rem
@echo -----
@echo Generate %caalias%, %srvalias%, and %cltalias% key-pairs in %cajkskeystore%,
%srvjkskeystore%, and %cltjkskeystore%
@rem
@rem -- Generate a client certificate and a private key pair
@rem Omit this step, unless you are authenticating clients.
%keytool% -genkeypair -noprompt -alias %cltalias% -dname %cltdname% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass% -keypass %cltkeypass% -keyalg %algorithm%
-validity %validity%

```

```

@rem -- Generate a server certificate and private key pair
%keytool% -genkeypair -noprompt -alias %srvalias% -dname %srvdname% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass% -keypass %srvkeypass% -keyalg %algorithm%
-validity %validity%

```

```

@rem Create CA, client and server key-pairs
@rem -- Generate a CA certificate and private key pair - The extension asserts this is a
certificate authority certificate, which is required to import into firefox
%keytool% -genkeypair -noprompt -ext bc=ca:true -alias %caalias% -dname %cadname%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass% -keyalg
%algorithm% -validity %validity%

```

sscerts.bat

sscerts.bat Script 中的指令會從其金鑰儲存庫匯出用戶端及伺服器自簽憑證，然後將伺服器憑證匯入用戶端信任儲存庫，並將用戶端憑證匯入伺服器金鑰儲存庫。伺服器沒有信任儲存庫。這些指令會從用戶端 JKS 信任儲存庫，以 PEM 的格式建立用戶端信任儲存庫。

```
@rem
@echo -----
@echo Export self-signed certificates: %srvcertselfsigned% and %cltcertselfsigned%
@rem Export Server public certificate
%keytool% -exportcert -noprompt -rfc -alias %srvalias% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass% -file %srvcertselfsigned%
@rem Export Client public certificate
@rem Omit this step, unless you are authenticating clients.
%keytool% -exportcert -noprompt -rfc -alias %cltalias% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass% -file %cltcertselfsigned%
```

```
@rem
@echo -----
@echo Add selfsigned server certificate %srvcertselfsigned% to client trust store:
%cltsrvjkstruststore%
@rem Import the server certificate into the client-server trust store (for server self-
signed authentication)
%keytool% -import -noprompt -alias %srvalias% -file %srvcertselfsigned% -keystore
%cltsrvjkstruststore% -storepass %cltsrvjkstruststorepass%
```

```
@rem
@echo -----
@echo Add selfsigned client certificate %cltcertselfsigned% to server trust store:
%srvjkskeystore%
@rem Import the client certificate into the server trust store (for client self-signed
authentication)
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %cltalias% -file %cltcertselfsigned% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
```

```
@rem
@echo -----
@echo Create a pem client-server trust store from the jks client-server trust store:
%cltsrvpemtruststore%
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltsrvjkstruststore% -destkeystore
%cltsrvp12truststore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltsrvjkstruststorepass% -deststorepass %cltsrvp12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltsrvp12truststore% -out %cltsrvpemtruststore% -passin
pass:%cltsrvp12truststorepass% -passout pass:%cltsrvpemtruststorepass%@rem
@echo -----
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore% -destkeystore
%cltp12keystore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass% -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore% -out %cltpemkeystore% -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

cacerts.bat

此 Script 會將憑證管理中心主要憑證匯入專用金鑰儲存庫。需要 CA 主要憑證，才可在主要憑證與已簽章的憑證之間建立金鑰鏈。cacerts.bat Script 會從其金鑰儲存庫匯出用戶端及伺服器憑證申請。Script 會使用 cajkskeystore.jks 金鑰儲存庫中專用憑證管理中心的金鑰，來簽署憑證申請，然後將已簽章的憑證匯回提出申請的同一個金鑰儲存庫。匯入會使用 CA 主要憑證建立憑證鏈。Script 會從用戶端 JKS 信任儲存庫，以 PEM 的格式建立用戶端信任儲存庫。

```
@rem
@echo -----
@echo Export self-signed certificates: %cacert%
@rem
@rem Export CA public certificate
%keytool% -exportcert -noprompt -rfc -alias %caalias% -keystore %cajkskeystore% -storepass
%cajkskeystorepass% -file %cacert%
```

```

@rem
@echo -----
@echo Add CA to server key and client key and trust stores: %srvjkskeystore%,
%cltjkskeystore%, %cltcajkstruststore%,
@rem The CA certificate is necessary to create key chains in the client and server key
stores,
@rem and to certify key chains in the server key store and the client trust store
@rem
@rem Import the CA root certificate into the server key store
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass%
@rem Import the CA root certificate into the client key store
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass%
@rem Import the CA root certificate into the client ca-trust store (for ca chained
authentication)
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltcajkstruststore%
-storepass %cltcajkstruststorepass%

```

```

@rem
@echo -----
@echo Create certificate signing requests: %srvcertreq% and %cltcertreq%
@rem
@rem Create a certificate signing request (CSR) for the server key
%keytool% -certreq -alias %srvalias% -file %srvcertreq% -keypass %srvkeypass% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Create a certificate signing request (CSR) for the client key
%keytool% -certreq -alias %cltalias% -file %cltcertreq% -keypass %cltkeypass% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass%

```

```

@rem
@echo -----
@echo Sign certificate requests: %srvcertcasigned% and %cltcertcasigned%
@rem The requests are signed with the ca key in the cajkskeystore.jks keystore
@rem
@rem Sign server certificate request
%keytool% -gencert -infile %srvcertreq% -outfile %srvcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
@rem Sign client certificate request
@rem Omit this step, unless you are authenticating clients.
%keytool% -gencert -infile %cltcertreq% -outfile %cltcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%

```

```

@rem
@echo -----
@echo Import the signed certificates back into the key stores to create the key chain:
%srvjkskeystore% and %cltjkskeystore%
@rem
@rem Import the signed server certificate
%keytool% -import -noprompt -alias %srvalias% -file %srvcertcasigned% -keypass %srvkeypass%
-keystore %srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Import the signed client certificate and key chain back into the client keystore
%keytool% -import -noprompt -alias %cltalias% -file %cltcertcasigned% -keypass %cltkeypass%
-keystore %cltjkskeystore% -storepass %cltjkskeystorepass%
@rem
@rem The CA certificate is needed in the server key store, and the client trust store
@rem to verify the key chain sent from the client or server
@echo Delete the CA certificate from %cltjkskeystore%: it causes a problem in converting
keystore to pem
@rem Omit this step, unless you are authenticating clients.
%keytool% -delete -alias %caalias% -keystore %cltjkskeystore% -storepass
%cltjkskeystorepass%

```

```

@rem
@echo -----
@echo Create a pem client-ca trust store from the jks client-ca trust store:
%cltcapemtruststore%
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltcajkstruststore% -destkeystore
%cltcap12truststore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltcajkstruststorepass% -deststorepass %cltcap12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltcap12truststore% -out %cltcapemtruststore% -passin
pass:%cltcap12truststorepass% -passout pass:%cltcapemtruststorepass%

```

```

@rem
@echo
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore% -destkeystore
%cltp12keystore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass% -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore% -out %cltpemkeystore% -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%

```

mqcerts.bat

Script 會列出憑證目錄中的金鑰儲存庫及憑證。然後，它會建立 MQTT 範例佇列管理程式，並配置安全的遙測通道。

```

@echo
@echo List keystores and certificates
dir %certpath%\*.* /b

```

```

@rem
@echo Create queue manager and define mqtt channels and certificate stores
call "%MQ_FILE_PATH%\mqxr\Samples\SampleMQM" >> %mqlog%
echo DEFINE CHANNEL(%chlreq%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreq%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlopt%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslopt%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
V7.5.0.1
echo DEFINE CHANNEL(%chlsslreqws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreqws%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlssloptws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portssloptws%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portws%)
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
@echo MQ logs saved in %mqlog%echo

```

MQTT 用戶端識別、授權和鑑別

遙測 (MQXR) 服務使用 MQTT 通道，代表 MQTT 用戶端發佈至 WebSphere MQ 主題，或訂閱該主題。WebSphere MQ 管理者會配置用於 WebSphere MQ 授權的 MQTT 通道身分。管理者可以定義通道的一般身分，或者使用已連接至通道的用戶端的 Username 或 ClientIdentifier。

遙測 (MQXR) 服務可以使用用戶端提供的 Username 或者使用用戶端憑證來鑑別用戶端。則使用用戶端提供的密碼來鑑別 Username。

彙總：用戶端識別是用戶端身分的選項。根據環境定義，用戶端由 ClientIdentifier、Username、管理者建立的一般用戶端身分或用戶端憑證識別。用於確實性檢查的用戶端 ID 不一定是用於授權的 ID。

MQTT 用戶端程式會設定使用 MQTT 通道傳送至伺服器的 Username 和 Password。它們還可以設定加密和鑑別連線所需的 SSL 內容。管理者決定是否以及如何鑑別 MQTT 通道。

若要授權 MQTT 用戶端存取 IBM WebSphere MQ 物件，請授權用戶端的 ClientIdentifier 或 Username，或者授權一般用戶端身分。若要允許用戶端連接至 IBM WebSphere MQ，請鑑別 Username 或使用用戶端憑證。配置 JAAS 以鑑別 Username，或者配置 SSL 以鑑別用戶端憑證。

如果您在用戶端設定 Password，請使用 VPN 加密連線，或者配置 MQTT 通道以使用 SSL，來讓密碼保持私密。

管理用戶端憑證非常困難。因此，如果與密碼鑑別相關聯的風險可以接受，則一般會使用密碼鑑別來鑑別用戶端。

如果可以安全地管理和儲存用戶端憑證，則可以依賴於憑證鑑別。不過，在使用遙測的環境類型中，極少能夠安全地管理憑證。而是使用用戶端憑證鑑別裝置，補充在伺服器鑑別用戶端密碼。因為其他複雜性，用戶端憑證的使用限制為高度機密的應用。使用兩種形式的鑑別稱為兩因素鑑別。您必須知道其中一個因素（例如密碼），並具有另一個因素（例如憑證）。

在高度機密的應用（例如 chip-and-pin 裝置）中，在製造期間會鎖定裝置，以防止竄改內部軟硬體。將受時間限制的授信用戶端憑證複製到裝置。將裝置部署至要使用它的位置。每次使用裝置時，都會使用密碼或智慧卡的另一個憑證，來執行進一步鑑別。

MQTT 用戶端身分和授權

使用 `ClientIdentifier`、`Username` 或用於授權的一般用戶端身分來存取 WebSphere MQ 物件。

IBM WebSphere MQ 管理者有三個選取 MQTT 通道身分的選項。管理者在定義或修改用戶端使用的 MQTT 通道時進行選擇。身分用於授與 IBM WebSphere MQ 主題的存取權。您可以選取：

1. 用戶端 ID。
2. 管理者提供給通道的身分。
3. 從 MQTT 用戶端傳遞的 `Username`。

`Username` 是 `MqttConnectOptions` 類別的屬性。必須在用戶端連接至服務之前將其設定。其預設值是空值。

使用 IBM WebSphere MQ `setmqaut` 指令，以選取要授權給與 MQTT 通道相關聯的身分使用的物件和動作。例如，若要授權佇列管理程式 QM1 管理者所提供的通道身分 `MQTTClient`：

```
setmqaut -m QM1 -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p MQTTClient -all +put
setmqaut -m QM1 -t topic -n SYSTEM.BASE.TOPIC -p MQTTClient -all +pub +sub
```

相關資訊

[授權 MQTT 用戶端存取 WebSphere MQ 物件](#)

使用密碼進行 MQTT 用戶端鑑別

使用用戶端密碼鑑別 `Username`。用於鑑別用戶端的身分，可以不同於用於授權用戶端發佈至和訂閱主題的身分。

遙測 (MQXR) 服務會使用 JAAS 來鑑別用戶端 `Username`。JAAS 會使用 MQTT 用戶端提供的 `Password`。

IBM WebSphere MQ 管理者透過配置用戶端連接至的 MQTT 通道，決定是鑑別 `Username`，還是或者根本不執行鑑別。可以將用戶端指派給不同通道，而且可以配置每個通道以使用不同的方法鑑別其用戶端。如果使用 JAAS，則您可以配置哪些方法必須鑑別用戶端，哪些方法可以選擇性地鑑別用戶端。

選擇用於鑑別的身分不會影響選擇用於授權的身分。為了方便管理，您可能會設定用於授權的一般身分，但是鑑別要使用該身分的每個使用者。下列程序概述的步驟，用於鑑別要使用一般身分的個別使用者：

1. IBM WebSphere MQ 管理者使用 IBM WebSphere MQ Explorer 將 MQTT 通道身分設為任何名稱，例如 `MQTTClientUser`。
2. IBM WebSphere MQ 管理者授權 `MQTTClient` 發佈和訂閱任何主題：

```
setmqaut -m QM1 -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p MQTTClient -all +put
setmqaut -m QM1 -t topic -n SYSTEM.BASE.TOPIC -p MQTTClient -all +pub +sub
```

3. MQTT 用戶端應用程式開發者建立 `MqttConnectOptions` 物件，並在連接至伺服器之前，設定 `Username` 及 `Password`。
4. 安全開發者建立 JAAS `LoginModule`，以利用 `Password` 鑑別 `Username`，並將它併入 JAAS 配置檔。
5. IBM WebSphere MQ 管理者配置 MQTT 通道以使用 JAAS 來鑑別用戶端的 `UserName`。

使用 SSL 進行 MQTT 用戶端鑑別

MQTT 用戶端與佇列管理程式之間的連線一律由 MQTT 用戶端起始。MQTT 用戶端一律是 SSL 用戶端。伺服器的用戶端鑑別和 MQTT 用戶端的伺服器鑑別皆為選用項目。

透過向用戶端提供專用簽署數位憑證，您可以向 IBM WebSphere MQ 鑑別 MQTT 用戶端。IBM WebSphere MQ 管理者可以強制 MQTT 用戶端使用 SSL 向佇列管理程式鑑別本身。您只能透過交互鑑別來要求用戶端鑑別。

作為使用 SSL 的替代方案，一些類型的「虛擬私密網路 (VPN)」(例如 IPsec)，會鑑別 TCP/IP 連線的端點。VPN 會加密流經網路的每個 IP 封包。一旦建立這類 VPN 連線，即建立可信的網路。您可以透過 VPN 網路使用 TCP/IP 將 MQTT 用戶端連接至遙測通道。

使用 SSL 進行的用戶端鑑別依賴於具有密碼的用戶端。對於自簽憑證，密碼是用戶端的私密金鑰，否則是憑證管理中心提供的金鑰。金鑰用於簽章用戶端的數位憑證。擁有對應公開金鑰的任何人都可以驗證該數位憑證。可以信任憑證，如果這些憑證已鏈結，則可以透過憑證鏈回溯追蹤至授信主要憑證。用戶端驗證會將用戶端所提供憑證鏈中的所有憑證傳送至伺服器。伺服器會檢查憑證鏈，直到它找到信任的憑證為止。授信憑證是從自簽憑證產生的公用憑證，或者是一般由憑證管理中心發出的主要憑證。在最後一個選用步驟中，可以將信任的憑證與「現用」的憑證撤銷清冊相互比較。

授信憑證可能由憑證管理中心發出，並且已經包含在 JRE 憑證儲存庫中。它可以是自簽憑證，也可以是已作為授信憑證新增至遙測通道金鑰儲存庫的任何憑證。

註：遙測通道具有結合的金鑰儲存庫/信任儲存庫，其中保留一個以上遙測通道的私密金鑰，以及鑑別用戶端所需的任何公用憑證。因為 SSL 通道必須具有金鑰儲存庫，所以它與通道信任儲存庫是同一個檔案，永不參照 JRE 憑證儲存庫。言下之意，如果用戶端鑑別需要 CA 主要憑證，您必須將主要憑證放置於通道的金鑰儲存庫中，即使 JRE 憑證儲存庫中已經存在 CA 主要憑證也是如此。永不參照 JRE 憑證儲存庫。

考慮用戶端鑑別打算應對的威脅，以及用戶端和伺服器在應對威脅時所扮演的角色。單獨鑑別用戶端憑證不足以防止未獲授權存取系統。如果其他使用者已在使用該用戶端裝置，則該用戶端裝置不需使用憑證持有者的權限就可以運作。切勿依賴單一防禦措施來防範意外攻擊。至少使用雙重因數的鑑別方法，以及補充關於擁有憑證的私密資訊知識。例如，使用 JAAS，並且使用伺服器發出的密碼鑑別用戶端。

用戶端憑證的主要威脅是落入不適合的人手中。憑證保存在用戶端上受密碼保護的金鑰儲存庫中。系統是如何將憑證放入金鑰儲存庫中？MQTT 用戶端如何取得金鑰儲存庫的密碼？密碼保護的安全程度如何？遙測裝置通常易於移除，然後可被私下入侵。裝置硬體是否必須具有防竄改功能？配送和保護用戶端憑證非常困難，這稱為金鑰管理問題。

次要威脅是無意中誤用裝置來存取伺服器。比方說，如果 MQTT 應用程式被竄改，則它可能會使用已鑑別的用戶端身分，利用伺服器配置中的缺點。

若要使用 SSL 來鑑別 MQTT 用戶端，請配置遙測通道及用戶端。

-
-

使用 SSL 進行用戶端鑑別的 MQTT 用戶端配置

若要使用 SSL 來鑑別 MQTT 用戶端，MQTT 用戶端需使用 SSL 連接至遙測通道。它必須指定一個 TCP 埠，該埠對應於配置為鑑別 SSL 用戶端的遙測通道。

例如，在用戶端上：

```
MQTTClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```

用戶端 JVM 必須使用 JSSE 提供的標準 Socket Factory。如果您使用 Java ME，則必須確定已載入 JSSE 套件。如果您使用 Java SE，從 Java 1.4.1 版開始，JSSE 已包含在 JRE 中。

如果使用 SSL 連線，則需要在連接之前，設定許多 SSL 內容。您可以透過使用 -D 參數將內容傳遞至 JVM 來設定內容，也可以使用 `MqttConnectionOptions.setSSLProperties` 方法來設定內容。

如果您透過呼叫方法 `MqttConnectOptions.setSocketFactory(javax.net.SocketFactory)` 來載入非標準 Socket Factory，則 SSL 設定傳遞至網路 Socket 的方式已定義應用程式。

將用戶端的數位憑證(透過使用用戶端的私密金鑰簽署，或由 CA 簽署)，新增至用戶端上受密碼保護的金鑰儲存庫。如果憑證具有金鑰鏈，則您可以將憑證從金鑰鏈新增至儲存庫。伺服器驗證用戶端憑證時，它會使用用戶端傳送的憑證與其金鑰儲存庫中的憑證比對。它會利用本身具有的憑證，在金鑰鏈中尋找第一個符合項目。會忽略金鑰鏈的其餘內容。

MQTT 用戶端將其金鑰儲存庫中的所有憑證傳送至伺服器。如果伺服器鑑別用戶端傳送的任何金鑰鏈，便可鑑別用戶端。

您也可以使用 SSL 密碼組合來進行用戶端鑑別。以下是目前支援的 SSL 密碼組合清單(依英文字母排序)：

- SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA
- SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
- SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
- SSL_DH_anon_WITH_AES_128_CBC_SHA
- SSL_DH_anon_WITH_DES_CBC_SHA
- SSL_DH_anon_WITH_RC4_128_MD5
- SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_DSS_WITH_AES_128_CBC_SHA
- SSL_DHE_DSS_WITH_DES_CBC_SHA
- SSL_DHE_DSS_WITH_RC4_128_SHA
- SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_RSA_WITH_AES_128_CBC_SHA
- SSL_DHE_RSA_WITH_DES_CBC_SHA
- SSL_KRB5_EXPORT_WITH_DES_CBC_40_MD5
- SSL_KRB5_EXPORT_WITH_DES_CBC_40_SHA
- SSL_KRB5_EXPORT_WITH_RC4_40_MD5
- SSL_KRB5_EXPORT_WITH_RC4_40_SHA
- SSL_KRB5_WITH_3DES_EDE_CBC_MD5
- SSL_KRB5_WITH_3DES_EDE_CBC_SHA
- SSL_KRB5_WITH_DES_CBC_MD5
- SSL_KRB5_WITH_DES_CBC_SHA
- SSL_KRB5_WITH_RC4_128_MD5
- SSL_KRB5_WITH_RC4_128_SHA
- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA
- **V7.5.0.2** SSL_RSA_FIPS_WITH_AES_128_CBC_SHA256
- **V7.5.0.2** SSL_RSA_FIPS_WITH_AES_256_CBC_SHA256
- SSL_RSA_FIPS_WITH_DES_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_AES_128_CBC_SHA
- **V7.5.0.2** SSL_RSA_WITH_AES_128_CBC_SHA256
- **V7.5.0.2** SSL_RSA_WITH_AES_256_CBC_SHA256
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- **V7.5.0.2** SSL_RSA_WITH_NULL_SHA256
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC4_128_SHA

V7.5.0.2 如果您計劃使用 SHA-2 密碼組合，請參閱第 154 頁的『對 MQTT 用戶端使用 SHA-2 密碼組合的系統需求』。

相關概念

第 98 頁的『使用 SSL 進行通道鑑別的 MQTT 用戶端配置』

若要使用 SSL 鑑別遙測通道，用戶端必須使用 SSL 連接至遙測通道。它必須指定一個埠，該埠對應於配置為使用 SSL 的遙測通道。配置必須包括受通行詞組保護的金鑰儲存庫，其中包含私密簽章的伺服器數位憑證。

使用 SSL 進行遙測通道鑑別

MQTT 用戶端與佇列管理程式之間的連線一律由 MQTT 用戶端起始。MQTT 用戶端一律是 SSL 用戶端。伺服器的用戶端鑑別和 MQTT 用戶端的伺服器鑑別皆為選用項目。

除非將用戶端配置為使用支援匿名連線的 CipherSpec，否則它一律會嘗試鑑別伺服器。如果鑑別失敗，則不會建立連線。

作為使用 SSL 的替代方案，一些類型的「虛擬私密網路 (VPN)」(例如 IPsec)，會鑑別 TCP/IP 連線的端點。VPN 會加密流經網路的每個 IP 封包。一旦建立這類 VPN 連線，即建立可信的網路。您可以透過 VPN 網路使用 TCP/IP 將 MQTT 用戶端連接至遙測通道。

使用 SSL 的伺服器鑑別會鑑別作為要傳送機密資訊目標的伺服器。用戶端會針對放置在其信任儲存庫或 JRE cacerts 儲存庫中的憑證，執行符合從伺服器傳送之憑證的檢查。

JRE 憑證儲存庫是 JKS 檔案 cacerts。它位於 JRE InstallPath\lib\security\。安裝後，它的預設密碼為 changeit。您可以將信任的憑證儲存在 JRE 憑證儲存庫或用戶端信任儲存庫中。不能同時使用這兩個儲存庫。如果您想要將用戶端信任的公用憑證與其他 Java 應用程式使用的憑證分開，請使用用戶端信任儲存庫。如果您想要對用戶端上執行的所有 Java 應用程式使用一般憑證儲存庫，請使用 JRE 憑證儲存庫。如果決定使用 JRE 憑證儲存庫，請檢閱它所包含的憑證，以確保您信任這些憑證。

透過提供不同的信任提供者，您可以修改 JSSE 配置。您可以自訂信任提供者，以對憑證執行不同的檢查。在某些使用 MQTT 用戶端的 OGSi 環境中，環境會提供不同的信任提供者。

若要使用 SSL 來鑑別遙測通道，請配置伺服器及用戶端。

•

相關概念

第 98 頁的『使用 SSL 進行通道鑑別的 MQTT 用戶端配置』

若要使用 SSL 鑑別遙測通道，用戶端必須使用 SSL 連接至遙測通道。它必須指定一個埠，該埠對應於配置為使用 SSL 的遙測通道。配置必須包括受通行詞組保護的金鑰儲存庫，其中包含私密簽章的伺服器數位憑證。

使用 SSL 進行通道鑑別的 MQTT 用戶端配置

若要使用 SSL 鑑別遙測通道，用戶端必須使用 SSL 連接至遙測通道。它必須指定一個埠，該埠對應於配置為使用 SSL 的遙測通道。配置必須包括受通行詞組保護的金鑰儲存庫，其中包含私密簽章的伺服器數位憑證。

例如，在用戶端上：

```
MQTTClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```

用戶端 JVM 必須使用 JSSE 提供的標準 Socket Factory。如果您使用 Java ME，則必須確定已載入 JSSE 套件。如果您使用 Java SE，從 Java 1.4.1 版開始，JSSE 已包含在 JRE 中。

如果使用 SSL 連線，則需要在連接之前，設定許多 SSL 內容。您可以透過使用 -D 參數將內容傳遞至 JVM 來設定內容，也可以使用 MqttConnectionOptions.setSSLProperties 方法來設定內容。

如果您透過呼叫方法 MqttConnectOptions.setSocketFactory(javax.net.SocketFactory) 來載入非標準 Socket Factory，則 SSL 設定傳遞至網路 Socket 的方式已定義應用程式。

將用戶端編寫為使用 SSL 連接至遙測通道，並使用以下三種方法之一，將用戶端配置為信任伺服器憑證：

使用 cacerts 儲存庫中由已知憑證管理中心簽章的伺服器憑證。

如果伺服器傳送憑證鏈中的所有中間金鑰，就不需要進行額外配置。建議您檢查用戶端 JRE 的 cacerts 儲存庫中的憑證，並變更 cacerts 儲存庫的密碼

其他憑證

將您信任的憑證儲存在用戶端上的信任儲存庫中。您至少必須將憑證鏈中的其中一個憑證，儲存在信任儲存庫中，而且在 `MqttConnectionOptions.SSLProperty` 中設定信任儲存庫參數。

- `com.ibm.ssl.trustStore`
- `com.ibm.ssl.trustStorePassword`

使用自訂信任管理程式

實作信任提供者並向其傳遞所用演算法的名稱。設定提供者類別和演算法的名稱，以在 `MqttConnectionOptions.SSLProperty` 中使用。

- `com.ibm.ssl.trustStoreProvider`
- `com.ibm.ssl.trustStoreManager`

您也可以使用 SSL 密碼組合來進行通道鑑別。以下是目前支援的 SSL 密碼組合清單（依英文字母排序）：

- `SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DH_anon_EXPORT_WITH_RC4_40_MD5`
- `SSL_DH_anon_WITH_3DES_EDE_CBC_SHA`
- `SSL_DH_anon_WITH_AES_128_CBC_SHA`
- `SSL_DH_anon_WITH_DES_CBC_SHA`
- `SSL_DH_anon_WITH_RC4_128_MD5`
- `SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA`
- `SSL_DHE_DSS_WITH_AES_128_CBC_SHA`
- `SSL_DHE_DSS_WITH_DES_CBC_SHA`
- `SSL_DHE_DSS_WITH_RC4_128_SHA`
- `SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA`
- `SSL_DHE_RSA_WITH_AES_128_CBC_SHA`
- `SSL_DHE_RSA_WITH_DES_CBC_SHA`
- `SSL_KRB5_EXPORT_WITH_DES_CBC_40_MD5`
- `SSL_KRB5_EXPORT_WITH_DES_CBC_40_SHA`
- `SSL_KRB5_EXPORT_WITH_RC4_40_MD5`
- `SSL_KRB5_EXPORT_WITH_RC4_40_SHA`
- `SSL_KRB5_WITH_3DES_EDE_CBC_MD5`
- `SSL_KRB5_WITH_3DES_EDE_CBC_SHA`
- `SSL_KRB5_WITH_DES_CBC_MD5`
- `SSL_KRB5_WITH_DES_CBC_SHA`
- `SSL_KRB5_WITH_RC4_128_MD5`
- `SSL_KRB5_WITH_RC4_128_SHA`
- `SSL_RSA_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_RSA_EXPORT_WITH_RC4_40_MD5`
- `SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA`
- **V7.5.0.2** `SSL_RSA_FIPS_WITH_AES_128_CBC_SHA256`

- **V7.5.0.2** SSL_RSA_FIPS_WITH_AES_256_CBC_SHA256
- SSL_RSA_FIPS_WITH_DES_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_AES_128_CBC_SHA
- **V7.5.0.2** SSL_RSA_WITH_AES_128_CBC_SHA256
- **V7.5.0.2** SSL_RSA_WITH_AES_256_CBC_SHA256
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- **V7.5.0.2** SSL_RSA_WITH_NULL_SHA256
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC4_128_SHA

V7.5.0.2 如果您計劃使用 SHA-2 密碼組合，請參閱第 154 頁的『對 MQTT 用戶端使用 SHA-2 密碼組合的系統需求』。

相關概念

第 96 頁的『使用 SSL 進行用戶端鑑別的 MQTT 用戶端配置』

若要使用 SSL 來鑑別 MQTT 用戶端，MQTT 用戶端需使用 SSL 連接至遙測通道。它必須指定一個 TCP 埠，該埠對應於配置為鑑別 SSL 用戶端的遙測通道。

遙測通道上的發佈保密

使用 SSL 加密透過連線的傳輸，可保護在遙測通道之間以任一方向所傳送 MQTT 發佈保密的安全。

連接至遙測通道的 MQTT 用戶端，會使用 SSL 透過對稱金鑰加密法來維護在通道上所傳輸發佈保密的安全。因為不會鑑別端點，所以您不能信任單獨使用的通道加密。將保密安全與伺服器或交互鑑別結合使用。

作為使用 SSL 的替代方案，一些類型的「虛擬私密網路 (VPN)」(例如 IPsec)，會鑑別 TCP/IP 連線的端點。VPN 會加密流經網路的每個 IP 封包。一旦建立這類 VPN 連線，即建立可信的網路。您可以透過 VPN 網路使用 TCP/IP 將 MQTT 用戶端連接至遙測通道。

對於會加密通道和鑑別伺服器的一般配置，請參閱第 98 頁的『使用 SSL 進行遙測通道鑑別』。

如果加密 SSL 連線而不鑑別伺服器，則會使連線受到中間人攻擊。雖然您交換的資訊可以防竊聽，但是您不知道與您交換資訊的對象是誰。除非您控制網路，否則您就會面臨別人截取您的 IP 傳輸，以及假冒端點的問題。

您可以建立加密的 SSL 連線，而不鑑別伺服器，方法是使用支援匿名 SSL 的 Diffie-Hellman 金鑰交換 CipherSpec。在用戶端和伺服器之間共用且用於加密 SSL 傳輸的主要機密，在建立時不必交換私密簽章的伺服器憑證。

因為匿名連線不安全，所以大部分 SSL 實作不會預設為使用匿名 CipherSpec。如果遙測通道接受 SSL 連線的用戶端要求，則該通道必須具有受通行詞組保護的金鑰儲存庫。依預設，因為 SSL 實作不會使用匿名 CipherSpec，所以金鑰儲存庫必須包含用戶端可以鑑別的私密簽章憑證。

如果您使用匿名 CipherSpec，則伺服器金鑰儲存庫必須存在，但是它不必包含任何私密簽章的憑證。

建立加密連線的另一種方法，是利用您自己的實作，取代用戶端上的信任提供者。您的信任提供者不會鑑別伺服器憑證，但是會加密連線。

MQTT 用戶端和遙測通道的 SSL 配置

MQTT 用戶端及 WebSphere MQ Telemetry (MQXR) 服務使用 Java Secure Socket Extension (JSSE)，以使用 SSL 來連接遙測通道。MQTT C 用戶端和用於裝置的 WebSphere MQ Telemetry 常駐程式不支援 SSL。

配置 SSL 以鑑別遙測通道及 MQTT 用戶端，並加密用戶端與遙測通道之間的訊息傳送作業。

作為使用 SSL 的替代方案，一些類型的「虛擬私密網路 (VPN)」(例如 IPsec)，會鑑別 TCP/IP 連線的端點。VPN 會加密流經網路的每個 IP 封包。一旦建立這類 VPN 連線，即建立可信的網路。您可以透過 VPN 網路使用 TCP/IP 將 MQTT 用戶端連接至遙測通道。

您可以配置 Java MQTT 用戶端與遙測通道之間的連線，以使用透過 TCP/IP 的 SSL 通訊協定。維護安全的內容視配置 SSL 以使用 JSSE 的方式而定。您可以配置三個不同的安全等級，從最安全的配置開始：

1. 僅允許信任的 MQTT 用戶端連接。僅將 MQTT 用戶端連接至信任的遙測通道。加密用戶端和佇列管理程式之間的訊息；請參閱第 95 頁的『[使用 SSL 進行 MQTT 用戶端鑑別](#)』。
2. 僅將 MQTT 用戶端連接至信任的遙測通道。加密用戶端和佇列管理程式之間的訊息；請參閱第 98 頁的『[使用 SSL 進行遙測通道鑑別](#)』。
3. 加密用戶端和佇列管理程式之間的訊息；請參閱第 100 頁的『[遙測通道上的發佈保密](#)』。

JSSE 配置參數

修改 JSSE 參數，以變更配置 SSL 連線的方式。JSSE 配置參數組織成三個集：

1. [IBM WebSphere MQ 遙測通道](#)
2. [MQTT Java 用戶端](#)
3. [JRE](#)

使用「IBM WebSphere MQ 探險家」來配置遙測通道參數。在 `MqttConnectionOptions.SSLProperties` 屬性中設定 MQTT Java 用戶端參數。在用戶端和伺服器上，透過編輯 JRE 安全目錄中的檔案，修改 JRE 安全參數。

IBM WebSphere MQ 遙測通道

使用「WebSphere MQ 探險家」設定所有遙測通道 SSL 參數。

ChannelName

在所有通道上，ChannelName 都是必要的參數。

通道名稱識別與特定埠號相關聯的通道。對通道命名，以協助您管理 MQTT 用戶端集。

PortNumber

在所有通道上，PortNumber 都是選用參數。對於 TCP 通道，預設值為 1883，對於 SSL 通道，預設值為 8883。

與此通道相關聯的 TCP/IP 埠號。透過指定定義給通道的埠，將 MQTT 用戶端連接至通道。如果通道具有 SSL 內容，則用戶端必須使用 SSL 通訊協定進行連接；例如：

```
MQTTClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```

KeyFileName

KeyFileName 是 SSL 通道的必要參數。對於 TCP 通道，必須省略。

KeyFile 名稱是 Java 金鑰儲存庫的路徑，其中包含您提供的數位憑證。在伺服器上，使用 JKS、JCEKS 或 PKCS12 作為金鑰儲存庫類型。

使用下列其中一個副檔名來識別金鑰儲存庫類型：

```
.jks
.jceks
.p12
.pkcs12
```

使用任何其他副檔名的金鑰儲存庫將被視為 JKS 金鑰儲存庫。

您可以將伺服器上的其中一種金鑰儲存庫類型和用戶端上的其他金鑰儲存庫類型結合使用。

將伺服器的專用憑證放置在金鑰儲存庫中。該憑證稱為伺服器憑證。憑證可以是自簽憑證，也可以是簽章管理中心所簽章的憑證鏈之一部分。

如果您使用憑證鏈，請將相關憑證置於伺服器的金鑰儲存庫中。

會將伺服器憑證及其憑證鏈中的所有憑證，傳送至用戶端以鑑別伺服器身分。

如果您已將 `ClientAuth` 設定為 `Required`，則金鑰儲存庫必須包含鑑別用戶端所需的所有憑證。用戶端會傳送自簽憑證或憑證鏈，同時會對照金鑰儲存庫內的憑證，以此資料的第一個驗證來鑑別用戶端。透過使用憑證鏈，即使多個用戶端由不同的用戶端憑證發出，都可利用單一憑證加以驗證。

PassPhrase

`PassPhrase` 是 SSL 通道的必要參數。對於 TCP 通道，必須省略。

通行詞組用於保護金鑰儲存庫。

ClientAuth

`ClientAuth` 是一個選用 SSL 參數。它預設為不執行用戶端鑑別。對於 TCP 通道，必須省略。

如果要遙測 (MQXR) 服務先鑑別用戶端，再允許用戶端連接至遙測通道，請設定 `ClientAuth`。

如果設定 `ClientAuth`，則用戶端必須使用 SSL 連接至伺服器，並鑑別伺服器。作為設定 `ClientAuth` 的結果，用戶端會將其數位憑證及其金鑰儲存庫中的所有憑證傳送至伺服器。其數位憑證稱為用戶端憑證。會針對通道金鑰儲存庫和 JRE cacerts 儲存庫中保有的憑證，鑑別這些憑證。

CipherSuite

`CipherSuite` 是一個選用 SSL 參數。它預設為嘗試所有已啟用的 `CipherSpecs`。對於 TCP 通道，必須省略。

如果要使用特定 `CipherSpec`，請將 `CipherSuite` 設定為必須用於建立 SSL 連線的 `CipherSpec` 名稱。

Telemetry 服務和 MQTT 用戶端根據在每端上啟用的所有 `CipherSpec`，協議一般 `CipherSpecs`。如果在連線的任一端或全部兩端指定特定 `CipherSpec`，則它必須與另一端的 `CipherSpec` 相符。

透過將其他提供者新增至 JSSE，安裝其他密碼。

Federal Information Processing Standards (FIPS)

FIPS 是一項選用設定。依預設不會對其設定。

在佇列管理程式的「內容」畫面中，或者使用 `runmqsc`，可以設定 `SSLFIPS`。`SSLFIPS` 指定是否僅使用通過 FIPS 認證的演算法。

名單

撤銷名單是一項選用設定。依預設不會對其設定。

在佇列管理程式的「內容」畫面中，或者使用 `runmqsc`，可以設定 `SSLCRLNL`。`SSLCRLNL` 指定用於提供憑證撤銷位置的鑑別資訊物件名單。

不會使用其他設定 SSL 內容的佇列管理程式參數。

MQTT Java 用戶端

在 `MqttConnectionOptions.SSLProperties` 中設定 Java 用戶端的 SSL 內容; 例如:

```
java.util.Properties sslClientProperties = new Properties();
sslClientProperties.setProperty("com.ibm.ssl.keyStoreType", "JKS");
com.ibm.micro.client.mqttv3.MqttConnectOptions conOptions = new MqttConnectOptions();
conOptions.setSSLProperties(sslClientProperties);
```

特定內容的名稱與值在 `MqttConnectOptions` 的 API 文件中會有所說明。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](#)。

Protocol

`Protocol` 是選用項目。

通訊協定是在與遙測伺服器協議後選取。如果您需要特定通訊協定，則您可以選取它。如果遙測伺服器不支援該通訊協定，則連線失敗。

ContextProvider

ContextProvider 是選用項目。

KeyStore

KeyStore 是選用項目。如果在伺服器端設定 ClientAuth 以強制鑑別用戶端，請予以配置。

將使用用戶端的私密金鑰簽章的用戶端數位憑證，放入金鑰儲存庫。指定金鑰儲存庫路徑和密碼。類型和提供者是選用項目。JKS 是預設類型，IBMJCE 是預設提供者。

指定不同的金鑰儲存庫提供者，以參照新增金鑰儲存庫提供者的類別。透過設定金鑰管理者名稱，傳遞金鑰儲存庫提供者所用演算法的名稱，以實例化 KeyManagerFactory。

TrustStore

TrustStore 是選用項目。您可以將您信任的所有憑證都放在 cacerts 儲存庫中。

如果想要擁有不同的用戶端信任儲存庫，請配置信任儲存庫。如果伺服器是使用常用 CA（已將其主要憑證儲存在 cacerts 中）簽章的憑證，則無法配置信任儲存庫。

將伺服器的公共簽章的憑證或主要憑證新增至信任儲存庫，並指定信任儲存庫路徑和密碼。JKS 是預設類型，IBMJCE 是預設提供者。

指定不同的信任儲存庫提供者，以參照新增信任儲存庫提供者的類別。透過設定信任管理者名稱，傳遞信任儲存庫提供者所用演算法的名稱，以實例化 TrustManagerFactory。

JRE

可影響用戶端及伺服器上 SSL 行為的 Java 安全的其他方面，在 JRE 中配置。Windows 上的配置檔位於 *Java Installation Directory*\jre\lib\security 中。如果您使用的是 IBM WebSphere MQ 隨附的 JRE，則路徑如下表中所示：

表 3: JRE SSL 配置檔的檔案路徑 (依平台)	
平台	菲萊帕特
Windows	<i>WMQ Installation Directory</i> \java\jre\lib\security
Linux for System x (32 位元)	<i>WMQ Installation Directory</i> /java/jre/lib/security
其他 UNIX and Linux 平台	<i>WMQ Installation Directory</i> /java/jre64/jre/lib/security

常用憑證管理中心

cacerts 檔案包含常用憑證管理中心的主要憑證。除非您指定信任儲存庫，否則依預設會使用 cacerts。如果您使用 cacerts 信任儲存庫或未提供信任儲存庫，則必須檢閱並編輯 cacerts 中的簽章者清單，以滿足您的安全需求。

您可以使用執行 IBM Key Management 公用程式的 WebSphere MQ 指令 `strmqikm` 來開啟 cacerts。使用密碼 `changeit`，將 cacerts 作為 JKS 檔案開啟。修改密碼以維護檔案的安全。

配置安全類別

使用 `java.security` 檔案以登錄其他安全提供者和其他預設安全內容。

許可權

使用 `java.policy` 檔案來修改授與資源的權限。`javaws.policy` 會授與 `javaws.jar` 的權限

加密強度

一些 JRE 提供強度減弱的加密。如果您無法將金鑰匯入至金鑰儲存庫，則原因可能是加密的強度減弱。請嘗試使用 `strmqikm` 指令啟動 `ikeyman`，或者從 [IBM 開發者套件安全資訊](#) 下載嚴密但適用範圍受限制的檔案。

重要: 您所在的國家/地區可能會對加密軟體的進口、佔有、使用或轉口至其他國家/地區施加限制。在下載或使用未限定政策檔案之前，您必須檢查您所在國家/地區的法律。請檢查其進口、佔有、使用和轉口加密軟體的相關法規和政策，判斷是否允許該軟體。

修改信任提供者以允許用戶端連接至所有伺服器

範例說明如何新增信任提供者，以及如何從 MQTT 用戶端程式碼中參照它。範例不會執行用戶端或伺服器鑑別。產生的 SSL 連線已加密但未經鑑別。

第 104 頁的圖 20 中的程式碼 Snippet 會為 MQTT 用戶端設定 `AcceptAllProviders` 信任提供者和信任管理程式。

```
java.security.Security.addProvider(new AcceptAllProvider());
java.util.Properties sslClientProperties = new Properties();
sslClientProperties.setProperty("com.ibm.ssl.trustManager", "TrustAllCertificates");
sslClientProperties.setProperty("com.ibm.ssl.trustStoreProvider", "AcceptAllProvider");
conOptions.setSSLProperties(sslClientProperties);
```

圖 20: MQTT 用戶端程式碼 Snippet

```
package com.ibm.mq.id;
public class AcceptAllProvider extends java.security.Provider {
    private static final long serialVersionUID = 1L;
    public AcceptAllProvider() {
        super("AcceptAllProvider", 1.0, "Trust all X509 certificates");
        put("TrustManagerFactory.TrustAllCertificates",
            AcceptAllTrustManagerFactory.class.getName());
    }
}
```

圖 21: `AcceptAllProvider.java`

```
protected static class AcceptAllTrustManagerFactory extends
    javax.net.ssl.TrustManagerFactorySpi {
    public AcceptAllTrustManagerFactory() {}
    protected void engineInit(java.security.KeyStore keystore) {}
    protected void engineInit(
        javax.net.ssl.ManagerFactoryParameters parameters) {}
    protected javax.net.ssl.TrustManager[] engineGetTrustManagers() {
        return new javax.net.ssl.TrustManager[] { new AcceptAllX509TrustManager() };
    }
}
```

圖 22: `AcceptAllTrustManagerFactory.java`

```
protected static class AcceptAllX509TrustManager implements
    javax.net.ssl.X509TrustManager {
    public void checkClientTrusted(
        java.security.cert.X509Certificate[] certificateChain,
        String authType) throws java.security.cert.CertificateException {
        report("Client authtype=" + authType);
        for (java.security.cert.X509Certificate certificate : certificateChain) {
            report("Accepting:" + certificate);
        }
    }
    public void checkServerTrusted(
        java.security.cert.X509Certificate[] certificateChain,
        String authType) throws java.security.cert.CertificateException {
        report("Server authtype=" + authType);
        for (java.security.cert.X509Certificate certificate : certificateChain) {
            report("Accepting:" + certificate);
        }
    }
    public java.security.cert.X509Certificate[] getAcceptedIssuers() {
        return new java.security.cert.X509Certificate[0];
    }
    private static void report(String string) {
        System.out.println(string);
    }
}
```

圖 23: `AcceptAllX509TrustManager.java`

遙測通道 JAAS 配置

配置 JAAS 以鑑別用戶端傳送的 Username。

WebSphere MQ 管理者會配置哪些 MQTT 通道需要使用 JAAS 進行用戶端鑑別。指定要執行 JAAS 鑑別的每個通道的 JAAS 配置名稱。通道可以全部使用相同的 JAAS 配置，也可以使用不同的 JAAS 配置。配置定義在 `WMQData directory\qmgrs\qMgrName\mqxr\jaas.config` 中。

`jaas.config` 檔案依 JAAS 配置名稱組織。在每個配置名稱下面，是「登入」配置清單；請參閱第 105 頁的圖 24。

JAAS 提供四個標準「登入」模組。標準 NT 和 UNIX「登入」模組具有受限制的值。

JndiLoginModule

針對在 JNDI (Java 命名和目錄介面) 下配置的目錄服務進行鑑別。

Krb5LoginModule

使用 Kerberos 通訊協定進行鑑別。

NTLoginModule

使用現行使用者的 NT 安全資訊進行鑑別。

UnixLoginModule

使用現行使用者的 UNIX 安全資訊進行鑑別。

使用 `NTLoginModule` 或 `UnixLoginModule` 的問題是遙測 (MQXR) 服務以 `mqm` 身分執行，而不是以 MQTT 通道身分執行。`mqm` 是傳遞至 `NTLoginModule` 或 `UnixLoginModule` 以進行鑑別的身分，而不是用戶端的身分。

若要解決此問題，請撰寫您自己的「登入」模組，或者使用其他標準「登入」模組。WebSphere MQ Telemetry 隨附了範例 `JAASLoginModule.java`。它是 `javax.security.auth.spi.LoginModule` 介面的實作。可以使用它來開發您自己的鑑別方法。

您提供的所有新 `LoginModule` 類別都必須在遙測 (MQXR) 服務的類別路徑上。請勿將類別放置在類別路徑中的 WebSphere MQ 目錄上。建立您自己的目錄，並定義遙測 (MQXR) 服務的完整類別路徑。

透過在 `service.env` 檔案中設定類別路徑，可以擴增遙測 (MQXR) 服務使用的類別路徑。`CLASSPATH` 必須大寫，並且類別路徑陳述式只能包含文字。不能在 `CLASSPATH` 中使用變數；例如 `CLASSPATH=%CLASSPATH%` 就是不正確的。遙測 (MQXR) 服務會設定其專屬類別路徑。會將定義在 `service.env` 中的 `CLASSPATH` 新增至該類別路徑。

遙測 (MQXR) 服務提供兩個回呼，它們會傳回連接至 MQTT 通道的用戶端的 Username 及 Password。使用者名稱和密碼設定在 `MqttConnectOptions` 物件中。請參閱第 106 頁的圖 25，以取得如何存取 Username 和 Password 的範例。

範例

具有一個具名配置 `MQXRConfig` 的 JAAS 配置檔範例。

```
MQXRConfig {
    samples.JAASLoginModule required debug=true;
    //com.ibm.security.auth.module.NTLoginModule required;
    //com.ibm.security.auth.module.Krb5LoginModule required
    //    principal=principal@your_realm
    //    useDefaultCcache=TRUE
    //    renewTGT=true;
    //com.sun.security.auth.module.NTLoginModule required;
    //com.sun.security.auth.module.UnixLoginModule required;
    //com.sun.security.auth.module.Krb5LoginModule required
    //    useTicketCache="true"
    //    ticketCache="${user.home}/${}tickets";
};
```

圖 24: 範例 `jaas.config` 檔案

「JAAS 登入」模組的範例，編寫目的是接收 MQTT 用戶端提供的 Username 和 Password。

```

public boolean login()
    throws javax.security.auth.login.LoginException {
    javax.security.auth.callback.Callback[] callbacks =
        new javax.security.auth.callback.Callback[2];
    callbacks[0] = new javax.security.auth.callback.NameCallback("NameCallback");
    callbacks[1] = new javax.security.auth.callback.PasswordCallback(
        "PasswordCallback", false);
    try {
        callbackHandler.handle(callbacks);
        String username = ((javax.security.auth.callback.NameCallback) callbacks[0])
            .getName();
        char[] password = ((javax.security.auth.callback.PasswordCallback) callbacks[1])
            .getPassword();
        // Accept everything.
        if (true) {
            loggedIn = true;
        } else
            throw new javax.security.auth.login.FailedLoginException("Login failed");

        principal= new JAASPrincipal(username);

    } catch (java.io.IOException exception) {
        throw new javax.security.auth.login.LoginException(exception.toString());
    } catch (javax.security.auth.callback.UnsupportedCallbackException exception) {
        throw new javax.security.auth.login.LoginException(exception.toString());
    }

    return loggedIn;
}

```

圖 25: 範例 `JAASLoginModule.Login()` 方法

用戶端程式設計概念

本節中所述的概念，可協助您瞭解用於 MQTT protocol 3.1 版的 Java 用戶端。這些概念是套件 `com.ibm.micro.client.mqttv3` 隨附之 API 文件的補充。

`com.ibm.micro.client.mqttv3` 包含為 MQTT 3.1 版通訊協定的 Java 實作提供公用方法的類別。安裝 IBM WebSphere MQ Telemetry 時隨附 `com.ibm.micro.client.mqttv3` 套件，以及實作 Java SE 及 ME 的通訊協定的隨附套件。

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套件。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。

Java 用戶端是 MQTT protocol 3.1 版的參照實作。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 [Telemetry Transport format and protocol](#)，以取得詳細資料。

套件 `com.ibm.micro.client.mqttv3` 的用戶端 API 文件未假設用戶端所連接的伺服器。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](#)。用戶端在連接到不同伺服器時，其行為可能會略有不同。下面的說明描述用戶端在連接至 IBM WebSphere MQ 遙測 (MQXR) 服務時的行為。

適用於 JavaScript 的 MQTT 傳訊用戶端及 Web 應用程式

直到最近，對 Web 應用程式進程式設計以及建立傳訊應用程式，一直都還遵循著各自不同的規則。無論您先前有多少經驗，將 JavaScript 與傳訊搭配使用，都會讓您受益匪淺。當您將傳訊應用程式撰寫為 Web 應用程式時，即可將其拉入任何最新的瀏覽器並在其中執行它。如果您變更應用程式，則只要重新整理瀏覽器，就會拉入最新版本。瀏覽器還會監視安全及可靠的訊息傳輸。

如何使用 Web 應用程式簡化應用程式部署

如果您有在（例如）IBM WebSphere MQ 上開發及部署傳統傳訊應用程式的經驗，可能會熟悉下列部署程序：

1. 系統管理者安裝及內嵌用戶端程式庫。
2. 系統管理者安排要配送給一般使用者並在其本端系統上安裝的傳訊應用程式。
3. 當程式碼變更時，系統管理者會重複先前的步驟（因此變更管理很複雜）。

如果您將傳訊應用程式撰寫為 Web 應用程式，則部署程序如下：

1. 系統管理者在 URL 上提供 Web 應用程式及用戶端程式庫。
2. 一般使用者的瀏覽器同時拉入 Web 應用程式及用戶端程式庫。
3. 當程式碼變更且重新整理瀏覽器時，會選擇更新的版本（因此變更管理很簡單）。

可能想要從 Web 應用程式中之瀏覽器直接使用傳訊的原因

如果您有在 JavaScript 中對應用程式進行程式設計的經驗，您可能有興趣知道傳訊系統（例如 IBM WebSphere MQ）提供的好處：

- 如果您透過傳訊系統傳送及接收訊息，此系統會負責確保遞送訊息。
- 因為傳訊系統會監視遞送，所以 Web 應用程式可以「隨發即忘」。這樣會大大簡化程式設計邏輯。如果為您遞送訊息，則您的程式碼不需要檢查這些訊息是否送達。應用程式不再需要處理傳送確認，或儲存未遞送的訊息及稍後重試它們。
- 傳訊系統會提供事件驅動的傳訊。您的用戶端應用程式不再需要傳送要求，然後繼續輪詢回應。相反，在發生有興趣的事件時，傳訊伺服器會將訊息傳送給您的用戶端應用程式。這也意味著，在事件發生時會警示您的用戶端應用程式，而不是等到應用程式下次輪詢伺服器。
- 事件驅動的傳訊還會大大地減少管理用戶端應用程式之裝置的負載、瀏覽器與傳訊伺服器之間的網路資料流量，以及傳訊伺服器的負載。隨著越來越多的系統在行動式裝置上執行，並且跨越無線網路進行連接，這也就顯示越來越重要。

各個部分如何配合運作

適用於 JavaScript 的 MQTT 傳訊用戶端包括用戶端程式庫，以及使用程式庫的範例 Web 應用程式。您可以撰寫自己的使用程式庫的 Web 應用程式。然後，使 Web 應用程式及用戶端程式庫在您選擇的 URL 上可用，例如，由 MQ 佇列管理程式（如下圖所示）或由應用程式伺服器。瀏覽器會拉入 Web 應用程式及用戶端程式庫，然後 Web 應用程式即會使用瀏覽器連接至 MQTT 伺服器（如 IBM WebSphere MQ Telemetry 或 IBM MessageSight），並與其交換訊息。

流程如下：

1. 每個瀏覽器實例均會重新整理與 Web 應用程式可用的 URL 之連線，並將最新版本的 Web 應用程式及用戶端程式庫載入瀏覽器。
2. Web 應用程式會使用 MQTT 透過 WebSocket protocol 連接至佇列管理程式，並訂閱感興趣的主題。
3. 佇列管理程式使用相同的連線，傳送符合 Web 應用程式所傳回訂閱的訊息。

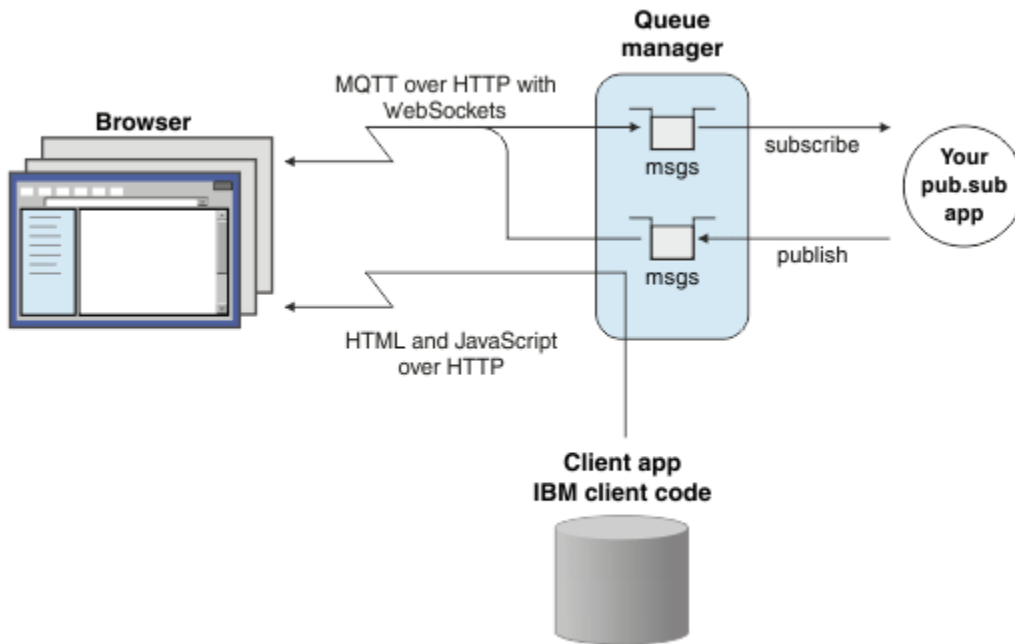


圖 26: 使用具有發佈和訂閱傳訊的適用於 JavaScript 的 MQTT 傳訊用戶端

Web 應用程式包含應用程式邏輯，及 MQTT 伺服器的 URL。在瀏覽器中開啟之後，應用程式會連接至 MQTT 伺服器，建立其需要的訂閱，然後等待接收事件驅動的警示並處理它們。

Web 應用程式將 MQTT 用作透過 WebSockets 執行的傳輸通訊協定進行連接。最先進的瀏覽器可以進行 WebSockets 連線。透過使用 WebSockets，Web 應用程式可以透過接受 HTTP 及 WebSocket protocol 的防火牆傳遞訊息，並且可以傳送資料封包（稱為「訊框」），就像使用 TCP over IP 一樣。

當 Web 應用程式傳送的訊息到達 MQTT 伺服器時，伺服器端應用程式會僅將其視為訊息。但不知道訊息來自瀏覽器。

管理及控制 MQTT 伺服器

MQTT 伺服器會處理伺服器端的傳訊複雜性。它可確保遞送從 Web 應用程式接收的訊息，並管理回應 Web 應用程式的發佈和訂閱應用程式。對於任何 MQTT 伺服器，您需要完成下列步驟：

- 建立伺服器。
- 選擇埠。
- 定義新的 MQTT 通道。
- 配置用戶端 Web 應用程式以透過新的 MQTT 通道連接至選擇的埠。

您還需要向瀏覽器提供 Web 應用程式執行檔 JavaScript。如果您使用 IBM WebSphere MQ Telemetry，依預設，MQTT 伺服器會使用 Web 應用程式用於連接至 MQTT 伺服器的相同 MQTT 通道為您執行此作業。如果您在嘗試 MQTT，這會協助您快速開始及執行。對於正式作業，特別在高傳輸量環境中，您可能更喜歡使用專用的應用程式伺服器（例如，WebSphere Application Server），在個別通道上提供 Web 應用程式執行檔 JavaScript。

註：因為它為高傳輸量環境而設計，所以 IBM MessageSight 預期您執行此作業。

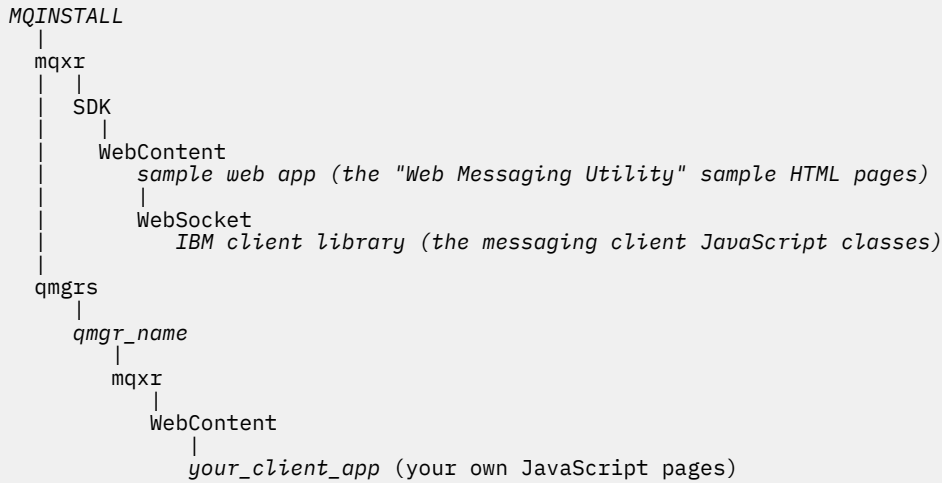
例如，如果您使用的是 IBM WebSphere MQ Telemetry，則可使用「IBM WebSphere MQ Explorer」的**新建遙測通道精靈**完成下列步驟：

1. 建立伺服器。
2. 選擇埠（依預設為 1883）。
3. 定義新的 MQTT 通道。

4. 配置用戶端 Web 應用程式以透過新的 MQTT 通道連接至選擇的埠。

Web 應用程式執行檔 JavaScript 也可以（選用）透過相同通道中的佇列管理程式提供。若要執行此作業，佇列管理程式必須支援 MQTT 及 HTTP。如果您已有為 MQTT 配置的佇列管理程式，可以使用 MQSC 指令行工具在通道定義中變更通訊協定，以支援 MQTT 及 HTTP。請參閱 [ALTER CHANNEL](#)。

Web 應用程式及適用於 JavaScript 的 MQTT 傳訊用戶端用戶端程式庫在磁碟上，儲存在應用程式伺服器或佇列管理程式定義的結構中。如果您使用 IBM WebSphere MQ Telemetry，Web 應用程式及用戶端程式庫會儲存在下列目錄結構中：



範例 Web 應用程式及用戶端程式庫儲存在 `MQINSTALL/mqxr/SDK/WebContent` 目錄中。此目錄中的資料由所有佇列管理程式提供。如果您不想讓使用者看到及使用所有這些資料，就應該建立自己的應用程式版本。若要使此應用程式或您自己的替代應用程式在特定的佇列管理程式中可用，您可以將應用程式置於 `MQINSTALL/qmgrs/qmgr_name/mqxr/WebContent` 目錄中。若要選取應用程式及關聯的 JavaScript 類別以在 URL 中提供服務，佇列管理程式會先在其自己的 `WebContent` 目錄中尋找，然後在廣域 `WebContent` 目錄中尋找。在先前的範例目錄樹中，佇列管理程式提供 `your_client_app` 及 JavaScript 類別的廣域複製。

若要讓佇列管理程式停止提供 Web 應用程式執行檔，或修改佇列管理程式尋找執行檔的位置，您可以配置 `webcontentpath` 內容，並將其新增至 `mqxr.properties` 檔。請參閱 [MQXR](#) 內容。

相關概念

[第 109 頁的『如何在 JavaScript 中對傳訊應用程式進行程式設計』](#)

相關工作

[第 71 頁的『透過 SSL 及 WebSockets 連接 適用於 JavaScript 的 MQTT 傳訊用戶端』](#)

使用 適用於 JavaScript 的 MQTT 傳訊用戶端 範例 HTML 頁面搭配 SSL 及 WebSocket protocol，將 Web 應用程式安全地連接至 IBM WebSphere MQ。

[第 21 頁的『適用於 JavaScript 的 MQTT 傳訊用戶端入門』](#)

透過顯示傳訊用戶端範例首頁並瀏覽它所鏈結的資源，即可開始使用適用於 JavaScript 的 MQTT 傳訊用戶端。若要顯示此首頁，您可以配置 MQTT 伺服器以接受來自 MQTT 傳訊用戶端範例 JavaScript 頁面的連線，然後在 Web 瀏覽器中鍵入您在伺服器上配置的 URL。適用於 JavaScript 的 MQTT 傳訊用戶端會在您的裝置上自動啟動，並且會顯示傳訊用戶端範例首頁。此頁面包含公程式、程式設計介面文件、指導教學及其他有用資訊的鏈結。

如何在 JavaScript 中對傳訊應用程式進行程式設計

適用於 JavaScript 的 MQTT 傳訊用戶端包含指導教學，其中會示範如何建立簡式發佈及訂閱 Web 應用程式。透過探索 "First steps, Hello world" 應用程式碼，您可以基本瞭解對於傳訊的 Web 應用程式進行程式設計的機制。

如果您迄今的經驗主要是開發及部署傳統的傳訊應用程式，您可能也會發現 [第 110 頁的『JavaScript 編碼提示』](#) 小節很有用。如果您是對傳訊不熟悉的經驗豐富的 JavaScript 開發者，將會在 [第 112 頁的『傳訊基礎』](#) 小節中發現對主要傳訊概念的簡介。

IBM messaging

Intro Utility Reference Tutorial Useful Links

o Tutorials

- o First steps, Hello World application
- o Making a complete web page
- o More about messages
- o More about subscribing
- o Diagnosing problems

First steps, the hello world application.

The example below is a simple javascript application that shows how to subscribe to a topic called "World" and publish a message containing the string "Hello" to it.

Example

```
// Make connection to the server.
client = new Messaging.Client(location.hostname, Number(location.port), "clientId");

// Set up a callBacks used when the connection is completed,
// when a message arrives for this client and when the connection is lost.
client.onConnectionLost = onConnectionLost;
client.onMessageArrived = onMessageArrived;
client.connect({onSuccess:onConnect});

function onConnect() {
    // Once a connection has been made, make a subscription and send a message.
    console.log("onConnect");
    client.subscribe("/World");
    message = new Messaging.Message("Hello");
    message.destinationName = "/World";
    client.send(message);
};

function onConnectionLost(responseObject) {
    if (responseObject.errorCode !== 0)
        console.log("onConnectionLost:"+responseObject.errorMessage);
};

function onMessageArrived(message) {
    console.log("onMessageArrived:"+message.payloadString);
    client.disconnect();
};
```

Click me to try. The Console output is shown below.

JavaScript 編碼提示

如果您習慣開發傳訊應用程式，而對 Web 應用程式不熟悉，可能會發現下列有用提示：

對 onSuccess 回呼中的每個事件覆蓋程式碼

當您撰寫傳訊應用程式時，您可以按下列順序撰寫下列事件之程式碼：

1. 連接
2. 訂閱
3. 發佈
4. 接收訊息

適用於 JavaScript 的 MQTT 傳訊用戶端 API 完全非同步，這意味著您的應用程式執行緒在等待諸如連接或訂閱的呼叫生效時不會封鎖。相反，這些呼叫會透過呼叫 onSuccess 或 onFailure 回呼用信號來告知完成。為了確保每個事件在觸發下一個事件之前已完成，您需要對 onSuccess 回呼中的每個事件覆蓋程式碼。例如，JavaScript 應用程式可能會在建立連線之前從進行連接呼叫中返回。為了確保在您訂閱之前已發生連線，您需要在 onSuccess 回呼中放入用於連接的訂閱程式碼。

"First steps, Hello world" 應用程式碼使用此方法。

在 HTML 標記中內嵌應用程式碼

以下是範例 JavaScript 頁面：

Example Web Messaging web page.

Connect
Make a connection to the server, and set up a call back used if a message arrives for this client.

Subscribe
Make a subscription to topic "/World".

Send
Create a Message object containing the word "Hello" and then publish it at the server.

Receive
A copy of the published Message is received in the callback we created earlier.

Disconnect
Now disconnect this client from the server.

以下是前一頁的原始檔，以顯示如何在 HTML 標記中內嵌應用程式碼：

```
<!DOCTYPE html>

<head>
  <script type="text/javascript" src="../WebSocket/mqttws31.js"></script>
  <script type="text/javascript">

var client;
var form = document.getElementById("tutorial");

function doConnect() {
  client = new Messaging.Client("whistler1.hursley.ibm.com", 1883, "ClientId");
  client.onConnect = onConnect;
  client.onMessageArrived = onMessageArrived;
  client.onConnectionLost = onConnectionLost;
  client.connect({onSuccess: onConnect});
}

function doSubscribe() {
  client.subscribe("/World");
}

function doSend() {
  message = new Messaging.Message("Hello");
  message.destinationName = "/World";
  client.send(message);
}

function doDisconnect() {
  client.disconnect();
}

// Web Messaging API callbacks

function onConnect() {
  var form = document.getElementById("example");
  form.connected.checked= true;
}

function onConnectionLost(responseObject) {
  var form = document.getElementById("example");
  form.connected.checked= false;
  if (responseObject.errorCode !== 0)
    alert(client.clientId+"\n"+responseObject.errorCode);
}

function onMessageArrived(message) {
  var form = document.getElementById("example");
  form.receiveMsg.value = message.payloadString;
}


```

```

</script>
</head>

<body>
<h1>Example Web Messaging web page.</h1>
<form id="example">
<fieldset>
<legend id="Connect" > Connect </legend>
  Make a connection to the server, and set up a call back used if a
  message arrives for this client.
  <br>
  <input type="button" value="Connect" onClick="doConnect(this.form)" name="Connect"/>
  <input type="checkbox" name="connected" disabled="disabled"/>
</fieldset>

<fieldset>
<legend id="Subscribe" > Subscribe </legend>
  Make a subscription to topic "/World".
  <br> <input type="button" value="Subscribe" onClick="doSubscribe(this.form)"/>
</fieldset>

<fieldset>
<legend id="Send" > Send </legend>
  Create a Message object containing the word "Hello" and then publish it at
  the server.
  <br>
  <input type="button" value="Send" onClick="doSend(this.form)"/>
</fieldset>

<fieldset>
<legend id="Receive" > Receive </legend>
  A copy of the published Message is received in the callback we created earlier.
  <textarea name="receiveMsg" rows="1" cols="40" disabled="disabled"></textarea>
</fieldset>

<fieldset>
<legend id="Disconnect" > Disconnect </legend>
  Now disconnect this client from the server.
  <br> <input type="button" value="Disconnect" onClick="doDisconnect()"/>
</fieldset>
</form>
</body>
</html>

```

傳訊基礎

以下是為不熟悉傳訊的 Web 應用程式開發者提供的部分背景傳訊資訊：

非同步及「隨發即忘」傳訊

MQTT 通訊協定支援保證遞送及「隨發即忘」傳送。在通訊協定中，訊息遞送為非同步：應用程式會將訊息傳遞給用戶端 API，並且不會採取任何進一步動作來確保訊息遞送。此方法稱為隨發即忘。當回應可用時，它會自動傳送至應用程式。

非同步遞送會從任何伺服器連線及從等待訊息釋放應用程式。互動模型如同電子郵件，但已針對應用程式程式設計進行最佳化。

另請參閱第 5 頁的『MQTT 簡介』的 "MQTT 通訊協定" 一節

發佈和訂閱傳訊概觀

資訊的提供者稱為發佈者。發佈者提供主題的相關資訊，無需瞭解有關對此資訊感興趣的應用程式之任何資訊。發佈者會選擇一個主題，這是有關特定主題的訊息儲存器。然後，發佈者會為此主題產生每則資訊作為訊息，稱為發佈，並將其張貼至關聯主題。

資訊的消費者稱為訂閱者。訂閱者會建立對所感興趣主題的訂閱。當新訊息張貼到某個主題時，會將訊息轉遞給此主題的所有訂閱者。訂閱者可以執行多個訂閱，並且可以從許多不同的發佈者接收資訊。

另請參閱 [IBM WebSphere MQ 發佈/訂閱傳訊簡介](#)

訂閱如何與主題相符。

如果您將 IBM WebSphere MQ 用作 MQTT 伺服器，就需要瞭解 IBM WebSphere MQ 如何指定主題。在 IBM WebSphere MQ 中，發佈者會建立訊息，並將其與最適合發佈資訊主旨的主題字串一併發佈。為了接收發佈資訊，訂閱者會建立訂閱，此訂閱具有用來選取發佈主題的型樣相符主題字串。佇列管理程式會將發佈資訊，遞送至其訂閱與發佈主題相符且獲授權來接收發佈資訊的訂閱者。

通常，主旨會按照階層架構的方式組織成主題樹狀結構，並使用 '/' 字元在主題字串中建立子主題。主題是主題樹狀結構中的節點。主題可以是沒有任何子主題的葉節點，或者是具有子主題的中間節點。訂閱者可以使用萬用字元一次訂閱多個主題。例如，訂閱 /sport/tennis 僅會取得張貼到網球子主題的訊息，而訂閱 /sport/# 則可取得張貼到 /sport 之任何子主題的訊息。

另請參閱[主題](#)、[主題樹狀結構](#)及[萬用架構](#)。

相關概念

第 106 頁的『[適用於 JavaScript 的 MQTT 傳訊用戶端及 Web 應用程式](#)』

相關工作

第 71 頁的『[透過 SSL 及 WebSockets 連接 適用於 JavaScript 的 MQTT 傳訊用戶端](#)』

使用 [適用於 JavaScript 的 MQTT 傳訊用戶端 範例 HTML 頁面](#) 搭配 SSL 及 WebSocket protocol，將 Web 應用程式安全地連接至 IBM WebSphere MQ。

第 21 頁的『[適用於 JavaScript 的 MQTT 傳訊用戶端入門](#)』

透過顯示傳訊用戶端範例首頁並瀏覽它所鏈結的資源，即可開始使用適用於 JavaScript 的 MQTT 傳訊用戶端。若要顯示此首頁，您可以配置 MQTT 伺服器以接受來自 MQTT 傳訊用戶端範例 JavaScript 頁面的連線，然後在 Web 瀏覽器中鍵入您在伺服器上配置的 URL。適用於 JavaScript 的 MQTT 傳訊用戶端會在您的裝置上自動啟動，並且會顯示傳訊用戶端範例首頁。此頁面包含公用程式、程式設計介面文件、指導教學及其他有用資訊的鏈結。

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會大量使用執行緒。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

`MqttCallback` 介面具有三個回呼方法；請參閱 [Callback.java](#) 中的範例實作。

`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`。

在用戶端應用程式所在的相同執行緒上，僅有的作為異常狀況拋出的遠端錯誤，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(MqttDeliveryToken token)`

`deliveryComplete` 由 MQTT 用戶端呼叫，以將遞送記號傳回給用戶端應用程式；請參閱第 116 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。在遞送完成後，如果

用戶端應用程式將 `MqttMessage` 物件解除參照，則可以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，您可以設定 `MqttDeliveryToken` 和 `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 用戶端利用相同的 `ClientIdentifier` 重新連接至前一個階段作業，該用戶端會繼續處理遞送確認通知；請參閱第 115 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，用戶端應用程式原先建立已發佈的訊息物件及其內容位元組陣列。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，則 `MqttClientPersistence` 類別會重建 `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(MqttTopic topic, MqttMessage message)

當發佈到達符合訂閱主題的用戶端時，便會呼叫 `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。

如果使用 QoS 1 或 2 傳送訊息，則在 MQTT 用戶端呼叫 `messageArrived` 之前，`MqttClientPersistence` 類別會儲存訊息。`messageArrived` 的行為方式與 `deliveryComplete` 相同：僅對發佈呼叫一次，`messageArrived` 返回到 MQTT 用戶端後，由 `MqttClientPersistence.remove` 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、執行緒作業和用戶端應用程式同步處理

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為回呼建立執行緒，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。您可以透過多個方法使用 `token.waitForCompletion` 來一次處理一個發佈：

1. 暫停應用程式用戶端，直到發佈遞送完成為止；請參閱 [PubSync.java](#)。
2. 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

表 4: 導致伺服器要求之方法的同步處理行為。

此表格列出 MQTT Java 用戶端中傳送要求至伺服器的方法。表格針對每種方法說明該方法在何種情況會等待或傳回，以及該方法所等待的時間。

方法	同步化	逾時間隔
<code>MqttClient.Connect</code>	等待與伺服器建立連線。	預設為 30 秒，或透過參數設定。
<code>MqttClient.Disconnect</code>	等待 MQTT 用戶端完成它必須執行的任何工作，以及等待 TCP/IP 階段作業中斷連線。	預設為 30 秒，或透過參數設定。
<code>MqttClient.Subscribe</code>	等待訂閱要求完成。	預設為 30 秒，或透過參數設定。
<code>MqttClient.UnSubscribe</code>	等待取消訂閱要求完成。	預設為 30 秒，或透過參數設定。
<code>MqttClient.Publish</code>	將要求傳遞至 MQTT 用戶端之後，立即返回應用程式執行緒。	無。
<code>MqttDeliveryToken.waitForCompletion</code>	等待傳回遞送記號。	預設為無限期等待，或透過參數設定。

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。狀態資訊用於確保「至少一次」和「只一次」遞送，以及「只一次」接收發佈。階段作業狀態還包括 MQTT 用戶端建立的訂閱。您可以選擇在執行 MQTT 用戶端時，在階段作業之間保留或不保留狀態資訊。在連接之前設定 `MqttConnectOptions.cleanSession`，以變更清除階段作業模式。

使用 `MqttClient.connect` 方法連接 MQTT 用戶端應用程式時，用戶端會利用用戶端 ID 及伺服器位址識別連線。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: WebSphere MQ 管理者可以強制關閉和開啟階段作業，以及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`，則會將用戶端建立的任何訂閱新增至用戶端連接之前已存在的所有訂閱。用戶端中斷連線時，所有訂閱都保持為作用中狀態。

另一個瞭解 `cleanSession` 屬性如何影響訂閱的方法是將它視為限制模式屬性。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；該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。

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。用戶端 ID 在所有連接至伺服器的用戶端中必須是唯一的，而且不得與伺服器上的佇列管理程式同名。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，假設某個裝置發生故障，而您可能透過客戶呼叫服務台得到通知。客戶如何識別裝置，而您如何使該識別與通常連接至用戶端的伺服器相關聯呢？您需要查詢將每個裝置對映至用戶端 ID 以及伺服器的資料庫嗎？裝置的名稱是否識別它連接至的伺服器？當您瀏覽 MQTT 用戶端連線時，會以用戶端 ID 標示每一個連線。您需要查閱某個表格，以將用戶端 ID 對映至實體裝置嗎？

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？如果客戶用新裝置更換發生故障的裝置，新裝置的 ID 與舊裝置是否相同？您來配置新 ID 嗎？如果您變更實體裝置，但保留相同的 ID，則會自動將尚未完成的發佈和作用中的訂閱傳送至新的裝置。

如何確保用戶端 ID 唯一？而且對於產生唯一 ID 的系統，您必須有可靠的程序來在用戶端上設定 ID。用戶端裝置可能是一個「黑箱」，沒有使用者介面。您是否利用用戶端 ID（例如使用其 MAC 位址）來製造該裝置？或者，您是否具有軟體安裝和配置程序，會在啟動裝置之前對其進行配置？

您可能會利用 48 位元的裝置 MAC 位址建立用戶端 ID，以讓 ID 保持簡短且唯一。如果傳輸大小並非重要問題，則您可能會使用剩餘的 17 個位元組來簡化位址管理。

遞送記號

用戶端在主題上發佈時，便會建立新的遞送記號。使用遞送記號可監視發佈的遞送，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。可以使用傳回的記號呼叫 `token.waitForCompletion`，來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`。

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 WebSphere MQ Telemetry 來傳送「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 identifier/Lost`。

透過使用 `MqttConnectionOptions.setWill(MqttTopic lastWillTopic, byte [] lastWillPayload, int lastWillQos, boolean lastWillRetained)` 方法建立最後留言。

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，以將訊息持續保存在 IBM WebSphere MQ 中，以及確保遞送。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。

MQTT 用戶端中的訊息持續性

如果利用服務品質「至少一次」或「只一次」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，訊息持續性具有兩個方面：訊息的傳送方式，以及它是否作為持續訊息排入 MQTT 伺服器的佇列中。

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「至多一次」(QoS=0)，則一旦發佈訊息，用戶端便會將其捨棄。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 WebSphere 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。該訊息會儲存在本端，而且只有在不再需要它之後，才會從用戶端捨棄，以確保「至少一次」(QoS=1) 或「只一次」(QoS=2) 遞送。

2. 如果將訊息標示為 QoS 1 或 2，則會將其作為持續訊息排入佇列中。如果將訊息標示為 QoS=0，則會將訊息作為非持續訊息排入佇列中。在 IBM WebSphere MQ 中，除非訊息通道具有設為 FAST 的 NPMSPPEED 屬性，否則會在佇列管理程式之間「正好一次」傳送非持續訊息。

持續發佈會儲存在用戶端上，直到用戶端應用程式收到該發佈為止。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 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會提供適用於 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。這些介面在適用於 Java 的 MQTT 用戶端的 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。用戶端也有 Java ME 的預設實作 `MqttDefaultMIDPPersistence`。

持續性類別

MqttClientPersistence

將 `MqttClientPersistence` 實作的實例，作為 `MqttClient` 建構子的參數傳遞至 MQTT 用戶端。如果您在 `MqttClient` 建構子中省略 `MqttClientPersistence` 參數，MQTT 用戶端會使用類別 `MqttDefaultFilePersistence` 或 `MqttDefaultMIDPPersistence` 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。如果您不使用 `MqttDefaultFilePersistence` 或 `MqttDefaultMIDPPersistence`，則必須提供 `MqttPersistable` 的實作和 `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`，則可以將用於儲存持續訊息的目錄，作為 `MqttDefaultFilePersistence` 建構子的參數提供。

此外，MQTT 用戶端還可以實例化 `MqttDefaultFilePersistence`，並將檔案放置在預設目錄中。目錄的名稱是 `client identifier-tcp hostname portnumber`。"`\`"、"`\\`"、"`/`"、"`:`" 和 "`"`" 會從目錄名稱字串中移除。

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。

`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。

MqttDefaultMIDPPersistence

`MqttDefaultMIDPPersistence` 具有預設建構子，沒有參數。它使用 `javax.microedition.rms.RecordStore` 套件來儲存訊息。

發佈

出版品是與主題字串相關聯的 `MqttMessage` 實例。MQTT 用戶端可以建立要傳送至 IBM WebSphere MQ 的發佈，以及訂閱 IBM WebSphere MQ 上的主題以接收發佈。

`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 通訊協定允許的訊息長度上限是 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。通常會在 UTF-8 中編碼只包含文字的 MQTT 出版品。使用方法 `getBytes("UTF8")` 來置換預設字集。

在 IBM WebSphere 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 `<mqs>` 資料夾。`<mqtt>` 資料夾包含 `clientId` 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，但 MQTT 用戶端確認的速度不夠快，則會重新傳送訊息，並將重複屬性設為 `true`。

正在發佈

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。

如果出版品具有「至少一次」或「正好一次」服務品質 `QoS=1` 或 `QoS=2`，則 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

佇列管理程式或 IBM MessageSight 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。

當 MQTT 用戶端收到具有「至少一次」服務品質的發佈時，它會呼叫 `MqttCallback.messageArrived` 方法來處理發佈。如果發佈的服務品質是「正好一次」`QoS=2`，當收到訊息時，MQTT 用戶端會呼叫 `MqttClientPersistence` 介面來儲存訊息。然後，它會呼叫 `MqttCallback.messageArrived`。

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 WebSphere 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 WebSphere 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 通訊協定不需要伺服器便可利用 `QoS=0`，將發佈轉遞至用戶端。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，可將相同的訊息多次傳送至接收端，並且可能會多次處理訊息。

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。

保留的發佈及 MQTT 用戶端

如果您訂閱的主題具有保留的發佈，則會將該主題上最新的保留的發佈立即轉遞給您。

可使用 `MqttMessage.setRetained` 方法來指定是否保留主題上的發佈。

若要刪除 IBM WebSphere MQ 中保留的發佈，請執行 `CLEAR TOPICSTR CLEAR TOPICSTR MQSC` 指令。

如果您以空值內容建立發佈，則會將空發佈轉遞給訂閱者。其他 MQTT 分配管理系統可能不會將空發佈轉遞給訂閱者。

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。

當您建立或更新保留的發佈時，請使用 `QoS` 或 1 或 2 來傳送發佈。如果您以 `QoS 0` 來傳送它，則 IBM WebSphere 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。

使用保留的發佈來記錄最新的測量值。所保留主題的新訂閱者會立即收到最新的測量值。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。

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。

當 MQTT 用戶端呼叫 `MqttCallback.messageArrived` 方法時，訂閱者會收到發佈。`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。

WebSphere MQ 指令可以移除訂閱。使用「WebSphere MQ 探險家」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`，則會將用戶端建立的任何訂閱新增至用戶端連接之前已存在的所有訂閱。用戶端中斷連線時，所有訂閱都保持為作用中狀態。

另一個瞭解 `cleanSession` 屬性如何影響訂閱的方法是將它視為限制模式屬性。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；該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。

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 WebSphere 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。主題字串透過使用方法 `MqttClient.getTopic(java.lang.String topicString)` 來建立。

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。主題過濾器透過使用訂閱方法來建立；例如，`MqttClient.subscribe(java.lang.String topicFilter)`。

主題字串

主題字串中說明 IBM WebSphere MQ 主題字串的語法。MQTT 主題字串的語法在適用於 Java 的 MQTT 用戶端。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](#)。的 API 文件中的 `MqttClient` 類別中有說明。

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 WebSphere MQ 的主題字串必須遵循佇列管理程式名稱的慣例。具體地說，主題字串不得包含連字號。
2. 長度上限不同。IBM WebSphere 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 WebSphere Message Broker 中，空值主題層次 '...//...' 無效。IBM WebSphere MQ 支援空值主題層次。

與 IBM WebSphere 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，會將主題字串對映至 WebSphere 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 WebSphere 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，在適用於 Java 的 MQTT 用戶端的 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。

每種主題過濾器類型的語法幾乎相同。唯一的差異是不同 MQTT 分配管理系統解譯主題過濾器的方式。在 WebSphere Message Broker 第 6 版中，多層次萬用字元只能在主題過濾器結尾使用。在 WebSphere MQ 中，多層次萬用字元可以在主題樹狀結構的任何層次中使用；例如 `USA/#/Dutchess County`。

MQTT 用戶端程式設計參考手冊

此處為 Mobile Messaging and M2M 用戶端套件 及相關用戶端 API 文件的鏈結。

在 Mobile Messaging and M2M 用戶端套件 中，MQTT 用戶端程式庫隨附於其產生的 API 文件。您可以從 IBM 傳訊社群下載中下載該用戶端程式集。

若要查看最新 API 文件的線上副本，則可以遵循下列指向 [Eclipse Paho](#) 專案的鏈結：

- [適用於 Java 的 MQTT 用戶端類別](#)
- [適用於 C 的 MQTT 用戶端程式庫](#)
- [適用於 C 的非同步 MQTT 用戶端程式庫](#)

註：

1. 將 MQTT Java 應用程式鏈結至 `org.eclipse.paho.client.mqttv3` 套件，而不是 `com.ibm.micro.client.mqttv3`。套件。提供 `com.ibm.micro.client.mqttv3` 套件以支援現有的 MQTT Java 應用程式。
2. **V7.5.0.1** 將適用於 C 的 MQTT 用戶端應用程式鏈結至 `MQTTAsync` 程式庫，而不是 `MQTTClient` 程式庫。提供 `MQTTClient` 以支援 C 的現有 MQTT 應用程式。
3. 適用於 JavaScript 的 MQTT 傳訊用戶端 需要支援 WebSockets 的 MQTT 伺服器。例如，IBM WebSphere MQ Version 7.5 及更新版本即具備此功能。

開始使用 MQTT 伺服器

支援 MQTT 傳輸通訊協定的傳訊伺服器，可從 IBM 及其他公司取得。最基本的 MQTT 伺服器，可讓 MQTT 用戶端程式庫所支援的行動式應用程式與裝置，能交換訊息。IBM WebSphere MQ 和 IBM MessageSight 是來自 IBM 的 MQTT 伺服器。除了可用作基本 MQTT 伺服器外，它們還可以在 MQTT 用戶端應用程式與企業應用程式之間交換訊息。IBM 提供的所有 MQTT 伺服器，皆支援 MQTT version 3.1 通訊協定以及 MQTT (透過 WebSocket protocol)。

IBM 目前提供的 MQTT 伺服器

IBM WebSphere MQ

- IBM WebSphere MQ 提供企業級傳訊。遙測元件使得 IBM WebSphere MQ 也能夠用作 MQTT 伺服器。

- 此伺服器支援行動式、機器對機器 (M2M) 及裝置型應用程式，並且還容許它們與企業傳訊應用程式（例如 IBM WebSphere MQ 及 JMS 應用程式）交換訊息。
- IBM WebSphere MQ 安裝包括來自 IBM 的 MQTT SDK 副本。此 SDK 提供範例 MQTT 用戶端應用程式，以及支援這些應用程式的 MQTT 用戶端程式庫。

註：若要取得此 SDK 的最新版本，請下載 [Mobile Messaging and M2M 用戶端套件](#)。如需相關資訊，請參閱第 9 頁的『開始使用 MQTT 用戶端』。

- MQTT 支援最初包含於 IBM WebSphere MQ Version 7.0.1。如需每個 IBM WebSphere MQ 版本的完整資訊，請參閱下列產品說明文件：
 - [WebSphere MQ Telemetry 7.5 版](#)
 - [WebSphere MQ Telemetry 7.1 版](#)

如需 IBM WebSphere MQ 的簡介及開始使用 IBM WebSphere MQ Telemetry 元件的步驟，請參閱第 124 頁的『IBM WebSphere MQ 作為 MQTT 伺服器』。

IBM MessageSight

- IBM MessageSight 是一種應用裝置型 MQTT 伺服器，可同時連接大量 MQTT 用戶端，並可提供所需的效能及可調整性，以適應不斷增長的行動式裝置及感應器需求。它支援 MQTT version 3.1 通訊協定，以及透過 WebSocket protocol 的 MQTT。



- 作為 MQTT 伺服器的 IBM MessageSight 的主要特性及好處如下：
 - 高效能、可靠性及可調式傳訊。
 - 專為機器對機器 (M2M) 及「物聯網」實務範例而設計，可針對同時連接的端點支援大量社群。
 - 易於安裝及使用。可在 30 分鐘內安裝完畢並執行。
 - 支援包含 Android 及 iOS 的原生行動式應用程式。
 - 與 IBM WebSphere MQ 整合為發佈/訂閱分配管理系統。
- 如需 IBM MessageSight 的快速簡介，請參閱 [YouTube 上的 MessageSight 簡介](#) 及 [MessageSight 公告](#)。如需詳細技術資訊，請參閱 [MessageSight 產品說明文件](#)。

IBM WebSphere MQ Telemetry daemon for devices

- 這也稱為 IBM WebSphere MQ Telemetry advanced client for C。它是一種小型覆蓋區的 MQTT 伺服器，通常執行於靠近網路邊緣的衛星位置或裝置中；例如像是執行於機上盒、遠端遙測裝置或銷售點終端機中。
- 它的一般用途是集中許多 MQTT 用戶端連線，然後以單一 MQTT 連線透過網際網路連接至 IBM WebSphere MQ。例如，您可能會在大廈中安裝大量感應器，並將其連接至 IBM WebSphere MQ Telemetry daemon for devices，然後將此常駐程式連接至 IBM WebSphere MQ。
- IBM WebSphere MQ Telemetry daemon for devices 隨附於 IBM WebSphere MQ。需要個別授權才能將它連接至 IBM WebSphere MQ。請參閱 [IBM United States Software Announcement 212-091](#)。

Really Small Message Broker

- Really Small Message Broker (RSMB) 是 IBM WebSphere MQ Telemetry daemon for devices 的其中一個版本。二者的主要差異在於用途。RSMB 是一種小型測試伺服器，可從 IBM alphaWorks 取得，且旨在用於評估或試用 MQTT 型解決方案。RSMB 在許多 Linux 平台、Windows XP、Apple Mac OS X Leopard 及 Unslung (Linksys NSLU12) 上皆支援 MQTT

IBM 提供的舊版 MQTT 伺服器

WebSphere Message Broker (現在稱為 IBM Integration Bus)

- WebSphere Message Broker 第 6 版提供了它自己的 MQTT 伺服器。在 WebSphere Message Broker 第 7 版中，此支援由 IBM WebSphere MQ 的遙測元件取代。

其他 MQTT 伺服器

[MQTT.org](#) 的軟體頁面維護 MQTT 伺服器及分配管理系統的清單，包括開放程式碼伺服器。

相關工作

第 9 頁的『[開始使用 MQTT 用戶端](#)』

您可以透過建置並執行使用 MQTT 用戶端程式庫的範例 MQTT 用戶端應用程式，開始開發行動式或 機器對機器 (M2M) 應用程式。範例應用程式及相關聯的用戶端程式庫可在 [Mobile Messaging and M2M 用戶端套件](#) 中從 IBM 取得。有一些版本的應用程式及用戶端程式庫以 Java、JavaScript 及 C 撰寫。您可以在大部分平台及裝置上執行這些應用程式，包括 Apple 中的 Android 裝置及產品。

IBM WebSphere MQ 作為 MQTT 伺服器

使用 IBM WebSphere MQ 中包含的 MQTT 伺服器簡介。

若要開始使用，請遵循下列文章中的步驟：

- [第 124 頁的『安裝 IBM WebSphere MQ』](#)
- [第 126 頁的『從指令行配置 MQTT 服務』](#)
- [第 128 頁的『使用 IBM WebSphere MQ Explorer 配置 MQTT 服務』](#)

註：您可以透過使用指令行介面範例快速開始使用。但是，如果您的配置與範例有很大差異，您需要更多知識和技術才能有效使用指令行介面。使用「IBM WebSphere MQ Explorer」介面可更輕鬆地開始使用及執行標準配置作業。

如需 IBM WebSphere MQ Telemetry 元件的主要概念資訊，請參閱 IBM WebSphere MQ 產品說明文件中的下列文章：

- [將遙測裝置連接至佇列管理程式](#)
- [遙測 \(MQXR\) 服務](#)
- [遙測通道](#)

相關資訊

[在 Linux 及 AIX 上配置遙測的佇列管理程式](#)

[在 Windows 上為遙測配置佇列管理程式](#)

[配置分散式佇列以將訊息傳送至 MQTT 用戶端管理 WebSphere MQ Telemetry](#)

安裝 IBM WebSphere MQ

遵循下列指示，以取得並安裝 IBM WebSphere MQ，並在 Windows 或 Linux 上配置 IBM WebSphere MQ Telemetry。

開始之前

如需 IBM WebSphere MQ 上執行的 MQTT 服務所支援的作業系統，請參閱 [IBM WebSphere MQ Telemetry 系統需求](#)。

按下列其中一種方式取得 IBM WebSphere MQ 安裝資料副本及授權：

1. 向 IBM WebSphere MQ 管理者取得安裝資料，並確認您可以接受授權合約。
2. 取得 IBM WebSphere MQ 的 90 天評估版。請參閱 [評估: IBM WebSphere MQ](#)。
3. 購買 IBM WebSphere MQ。請參閱 [IBM WebSphere MQ 產品頁面](#)。

關於這項作業

在 Linux 上以 root 身分及在 Windows 上以管理者身分安裝 IBM WebSphere MQ。安裝時，請選取其他選項遙測服務及遙測用戶端，以安裝 IBM WebSphere MQ Telemetry 元件。建立使用者 ID 以管理 IBM WebSphere MQ，及檢查是否已定義來賓使用者 ID。在 MQTT 服務配置範例中使用來賓使用者 ID，以將 MQTT 存取權授與 IBM WebSphere MQ。

安裝 IBM WebSphere MQ 之後，請執行第 126 頁的『從指令行配置 MQTT 服務』或第 128 頁的『使用 IBM WebSphere MQ Explorer 配置 MQTT 服務』中的步驟來啟動 MQTT 服務。

程序

1. 在 Linux 上以 root 身分登入，或在 Windows 上以管理者身分登入。
2. 安裝 IBM WebSphere MQ。

請遵循在 Linux 上安裝 WebSphere MQ 伺服器或在 Windows 上安裝 WebSphere MQ 伺服器中的指示。選取遙測服務及遙測用戶端，以安裝 IBM WebSphere MQ Telemetry 元件。

在 Linux 上，請記下 "下一步" 一節中的指示，將您的安裝設為主要安裝。即使此安裝是您工作站上的唯一 IBM WebSphere MQ 安裝，亦請將其設定為主要安裝。請參閱配置為主要安裝的 [WebSphere MQ 7.1 版或更新版本的單一安裝](#)。

若要完全遵循配置範例指示，您必須將安裝設定為主要安裝。

多個安裝: 如果您要使用非主要安裝，請執行 `setmqenv` 指令。它會在您工作站上的指令視窗中設定 IBM WebSphere MQ 環境。請參閱 [多個安裝](#)。

假設您已接受安裝程式提供的預設安裝位置，則 IBM WebSphere MQ 安裝在下列目錄中：

Linux 64 位元

```
/opt/mqm
```

Windows 32 位元

```
C:\Program Files\IBM\WebSphere MQ
```

Windows 64 位元

```
C:\Program Files (x86)\IBM\WebSphere MQ
```

安裝目錄顯示為 `MQ_INSTALLATION_PATH`

3. 選擇性的: 將您要使用其管理 IBM WebSphere MQ 的使用者新增至此工作站上的 mqm 群組。

在 Windows 上，此步驟是選用的，因為您可以 Windows 管理者身分來管理 IBM WebSphere MQ。請參閱在 UNIX 及 Windows 系統上管理 WebSphere MQ 的權限。

如果您的 Windows 工作站是網域的成員，請參閱具有非預設安全權限的 [Windows 2000 網域或具有預設安全權限的 Windows 2003 及 Windows Server 2008 網域](#)。

在 Linux 上，安裝程式會建立使用者 mqm，作為群組 mqm 的成員。為此使用者提供密碼，或建立其他使用者 (mqm 作為其主要群組)。

4. 選擇性的: 以您設定為 mqm 群組成員的使用者身分登入。

在 Windows 上，此步驟是選用的，因為您可以 Windows 管理者身分來管理 IBM WebSphere MQ。

5. 檢查您的工作站上是否已定義來賓使用者 ID。

訪客使用者 ID 在 Windows 上為 "guest"，在 Linux 上為 "nobody"。來賓使用者 ID 不需要任何作業系統權限。

結果

您已在工作站上安裝 IBM WebSphere MQ 作為主要 IBM WebSphere MQ 安裝，並已建立群組 mqm。安裝會提供管理 IBM WebSphere MQ 的權限給 mqm 群組的成員。Windows 上的管理者群組成員也具有管理 IBM WebSphere MQ 的權限。

下一步

1. 從指令行或 IBM WebSphere MQ Explorer 配置 MQTT 服務; 請參閱 [第 128 頁的『使用 IBM WebSphere MQ Explorer 配置 MQTT 服務』](#) 或 [第 126 頁的『從指令行配置 MQTT 服務』](#)。
2. 測試您的 Android、iOS、WebSockets、Java 及 "C" MQTT 用戶端。
3. 完成測試時，請在 Windows 上執行 `MQ_INSTALLATION_PATH\mqxr\samples\CleanupMQM.bat` 指令，並在 Linux 上執行 `MQ_INSTALLATION_PATH/mqxr/samples/CleanupMQM.sh` 指令，以移除佇列管理程式及 MQTT 服務。

相關資訊

[安裝 WebSphere MQ Telemetry](#)

[在 Linux 上安裝 WebSphere MQ 伺服器](#)

[在 Windows 上安裝 WebSphere MQ 伺服器](#)

從指令行配置 MQTT 服務

遵循下列指示，以使用指令行配置 IBM WebSphere MQ 來執行 IBM WebSphere MQ Telemetry 應用程式範例。這些步驟顯示如何執行 Script，以在稱為 MQXR_SAMPLE_QM 的新佇列管理程式上建立 MQTT 服務。

開始之前

您必須具有 IBM WebSphere MQ 佇列管理程式的管理存取權，才能設定 MQTT 服務。您具有多種方法來存取佇列管理程式：

1. 取得 IBM WebSphere MQ 的副本，並在您自己的 Linux 或 Windows 工作站上建立佇列管理程式。請遵循 [第 124 頁的『安裝 IBM WebSphere MQ』](#) 中的指示取得並安裝 IBM WebSphere MQ。請注意，安裝時還必須選取遙測服務及遙測用戶端。您也可以修改現有安裝，以新增這些選項。
2. 聯絡 IBM WebSphere MQ 管理者，要求已安裝 IBM WebSphere MQ Telemetry（可選）的伺服器上的佇列管理程式管理存取權。 **V7.5.0.1** 除佇列管理程式名稱之外，您至少還需要兩個 TCP/IP 埠（用於 MQTT 及透過 WebSockets 用於 MQTT）。如果您計劃連接安全用戶端，您至少還需要兩個埠。

若要完全按所述執行作業中的步驟，您必須能建立一個稱為 MQXR_SAMPLE_QM 的佇列管理程式，且 TCP/IP 埠 1883 必須是未使用的。

關於這項作業

在此作業中，您執行 Script 來建立佇列管理程式，然後配置 MQTT 服務以在埠 1883 上接聽 MQTT V3.1 用戶端連線。此配置提供每個人發佈及訂閱任何主題的許可權。安全及存取控制的配置為最小，預期僅用於存取權受限的安全網路上的佇列管理程式。若要在不安全的環境中執行 IBM WebSphere MQ 及 MQTT，您必須配置安全。若要配置 IBM WebSphere MQ 及 MQTT 的安全，請參閱此作業結尾的相關鏈結。

程序

1. 以具有 IBM WebSphere MQ 管理權限的使用者 ID 登入。
若要定義具有 IBM WebSphere MQ 管理權限的使用者 ID，請參閱 [第 124 頁的『安裝 IBM WebSphere MQ』](#) 中的步驟 3。
2. 開啟指令視窗並執行指令 Script 範例，以建立並啟動稱為 MQXR_SAMPLE_QM 及 MQTT 服務的佇列管理程式範例。

範例 Script 的路徑是 %MQ_FILE_PATH%\mqxr\samples\SampleMQM.bat (在 Windows 上) 和 MQ_INSTALLATION_PATH/mqxr/samples/SampleMQM.sh (在 Linux 上)。

鍵入下列指令以建立並配置佇列管理程式：

- Windows

```
"%MQ_FILE_PATH%\mqxr\samples\SampleMQM.bat"
```

- Linux

```
MQ_INSTALLATION_PATH/mqxr/samples/SampleMQM.sh
```

結果

此範例會使用 Windows 上的下列內容來建立稱為 PlainText 的 MQTT 通道：

```
com.ibm.mq.MQXR.channel/PlainText: \  
com.ibm.mq.MQXR.Protocol=MQTT;\br/>com.ibm.mq.MQXR.Port=1883;\br/>com.ibm.mq.MQXR.Backlog=4096;\br/>com.ibm.mq.MQXR.UserName=Guest;\br/>com.ibm.mq.MQXR.StartWithMQXRService=true
```

除 `com.ibm.mq.MQXR.UserName=nobody` 之外，Linux 上的通道內容與 Windows 相同。

使用變數 `com.ibm.mq.MQXR.UserName` 中設定的使用者 ID 連接至埠 1883 存取權 IBM WebSphere MQ 的 MQTT V3.1 用戶端。範例 Script 使用下列 IBM WebSphere MQ 指令授與使用者 ID：

```
setmqaut -m MQXR_SAMPLE_QM -t topic -n SYSTEM.BASE.TOPIC -p com.ibm.mq.MQXR.UserName -all +pub  
+sub  
setmqaut -m MQXR_SAMPLE_QM -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p com.ibm.mq.MQXR.UserName -all  
+put
```

第一個指令為使用者提供發佈和訂閱從基本主題繼承權限的主題的權限。第二個指令為使用者提供將訊息放置在 SYSTEM.MQTT.TRANSMIT.QUEUE 傳輸佇列上的權限。MQTT 服務會將 SYSTEM.MQTT.TRANSMIT.QUEUE 上的訊息作為發佈傳送至 MQTT 訂閱者。

Script 會在佇列管理程式上啟動 MQTT 服務，以接聽埠 1883 上的連線。

下一步

遵循下列步驟，以透過執行 MQTT V3.1 Java 應用程式範例來測試連線。

範例 Java 應用程式的來源位於 MQTTV3Sample.java 檔案中。

執行此範例需要兩個指令視窗。在一個視窗中以訂閱者身分執行範例，在另一個視窗中以發佈者身分執行範例。

- Windows 若要啟動訂閱者，請執行下列指令

```
"%MQ_FILE_PATH%\mqxr\samples\RunMQTTV3Sample.bat" -a subscriber
```

若要發佈，請執行指令：

```
"%MQ_FILE_PATH%\mqxr\samples\RunMQTTV3Sample.bat"
```

- Linux 若要啟動訂閱者，請執行下列指令

```
MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh -a subscriber
```

若要發佈，請執行指令：

```
MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh
```

發佈者及訂閱者會將輸出寫入其指令視窗：

```
Connected to tcp://localhost:1883
Publishing to topic "MQTTV3Sample/Java/v3" qos 2
Disconnected
Press any key to continue . . .
```

圖 27: 來自發佈者的輸出

```
Connected to tcp://localhost:1883
Subscribing to topic "MQTTV3Sample/#" qos 2
Press <Enter> to exit
Topic:      MQTTV3Sample/Java/v3
Message:    Message from MQTTv3 Java client
QoS:       2
```

圖 28: 來自訂閱者的輸出

伺服器現在已備妥，可讓您測試 MQTT V3.1 應用程式。

相關工作

[使用 WebSphere MQ 探險家配置 MQTT 服務](#)

遵循下列指示，使用 IBM WebSphere MQ Explorer 來配置 IBM WebSphere MQ，以執行範例 IBM WebSphere MQ Telemetry 用戶端。這些步驟顯示如何執行 Define sample 配置精靈來建立 MQTT 服務。

相關資訊

[WebSphere MQ Telemetry](#)

[開發 WebSphere MQ Telemetry 應用程式](#)

[管理 WebSphere MQ Telemetry](#)

[WebSphere MQ Telemetry 安全](#)

使用 IBM WebSphere MQ Explorer 配置 MQTT 服務

遵循下列指示，使用 IBM WebSphere MQ Explorer 來配置 IBM WebSphere MQ，以執行範例 IBM WebSphere MQ Telemetry 用戶端。這些步驟顯示如何執行 Define sample 配置精靈來建立 MQTT 服務。

開始之前

您必須具有 IBM WebSphere MQ 佇列管理程式的管理存取權，才能設定 MQTT 服務。您具有多種方法來存取佇列管理程式：

1. 取得 IBM WebSphere MQ 的副本，並在您自己的 Linux 或 Windows 工作站上建立佇列管理程式。請遵循第 124 頁的『[安裝 IBM WebSphere MQ](#)』中的指示取得並安裝 IBM WebSphere MQ。請注意，安裝時還必須選取遙測服務及遙測用戶端。您也可以修改現有安裝，以新增這些選項。
2. 聯絡 IBM WebSphere MQ 管理者，要求已安裝 IBM WebSphere MQ Telemetry（可選）的伺服器上的佇列管理程式管理存取權。 **V7.5.0.1** 除佇列管理程式名稱之外，您至少還需要兩個 TCP/IP 埠（用於 MQTT 及透過 WebSockets 用於 MQTT）。如果您計劃連接安全用戶端，您至少還需要兩個埠。

若要完全按所述執行作業中的步驟，您必須能建立一個稱為 MQXR_SAMPLE_QM 的佇列管理程式，且 TCP/IP 埠 1883 必須是未使用的。

關於這項作業

在這項作業中，您將執行 IBM WebSphere MQ Explorer Define sample 配置精靈來建立 MQTT 服務，以接聽埠 1883 上的 MQTT V3.1 用戶端連線。此配置提供每個人發佈及訂閱任何主題的許可權。安全及存取控制的配置為最小，預期僅用於存取權受限的安全網路上的佇列管理程式。若要在不安全的環境中執行 IBM WebSphere MQ 及 MQTT，您必須配置安全。若要配置 IBM WebSphere MQ 及 MQTT 的安全，請參閱此作業結尾的相關鏈結。

程序

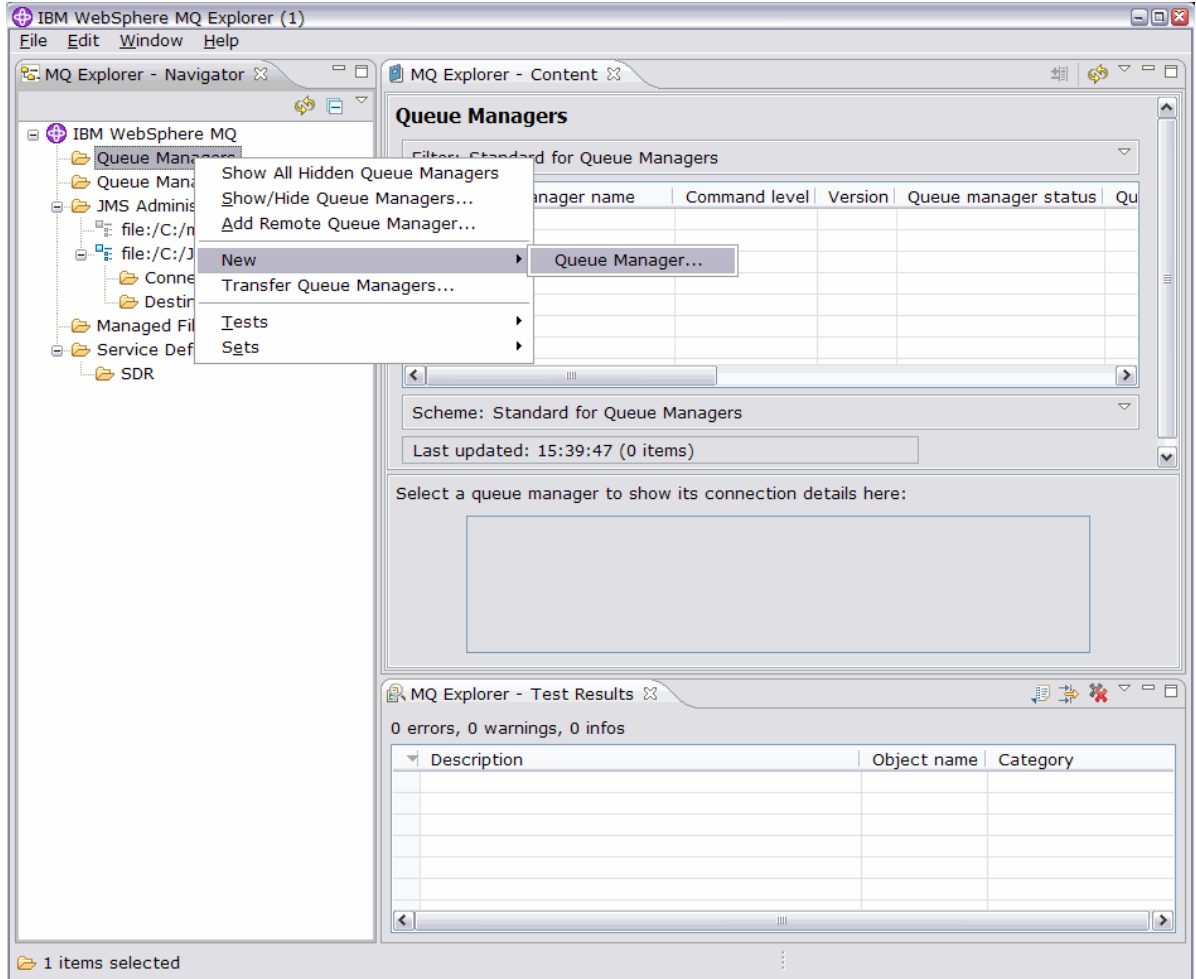
1. 以具有 IBM WebSphere MQ 管理權限的使用者 ID 登入。

若要定義具有 IBM WebSphere MQ 管理權限的使用者 ID，請參閱第 124 頁的『安裝 IBM WebSphere MQ』中的步驟 3。

2. 開啟指令視窗，並執行 IBM WebSphere MQ Explorer 指令 **strmqcfc** 以啟動 IBM WebSphere MQ Explorer。

3. 建立佇列管理程式

- a) 啟動新建佇列管理程式精靈



- b) 鍵入佇列管理程式名稱，及無法傳送郵件的佇列的名稱。為方便起見，請將它設為預設佇列管理程式。按一下完成。

Create Queue Manager

Queue Manager
Enter basic values

Queue manager name: * MQXR_SAMPLE_QM

Make this the default queue manager

Default transmission queue:

Dead-letter queue: SYSTEM.DEAD.LETTER.QUEUE

Max handle limit: 256

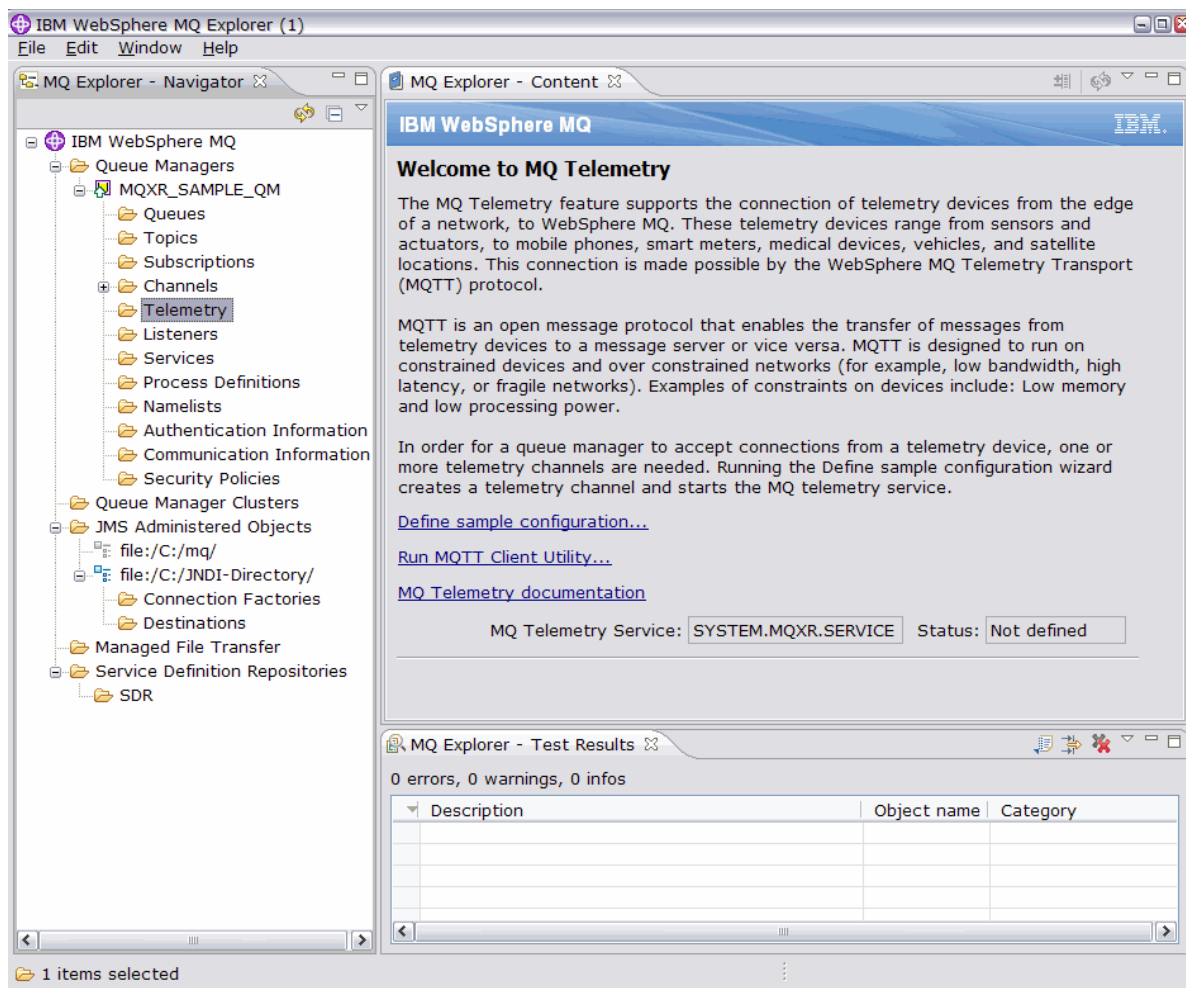
Trigger interval: 999999999

Max uncommitted messages: 10000

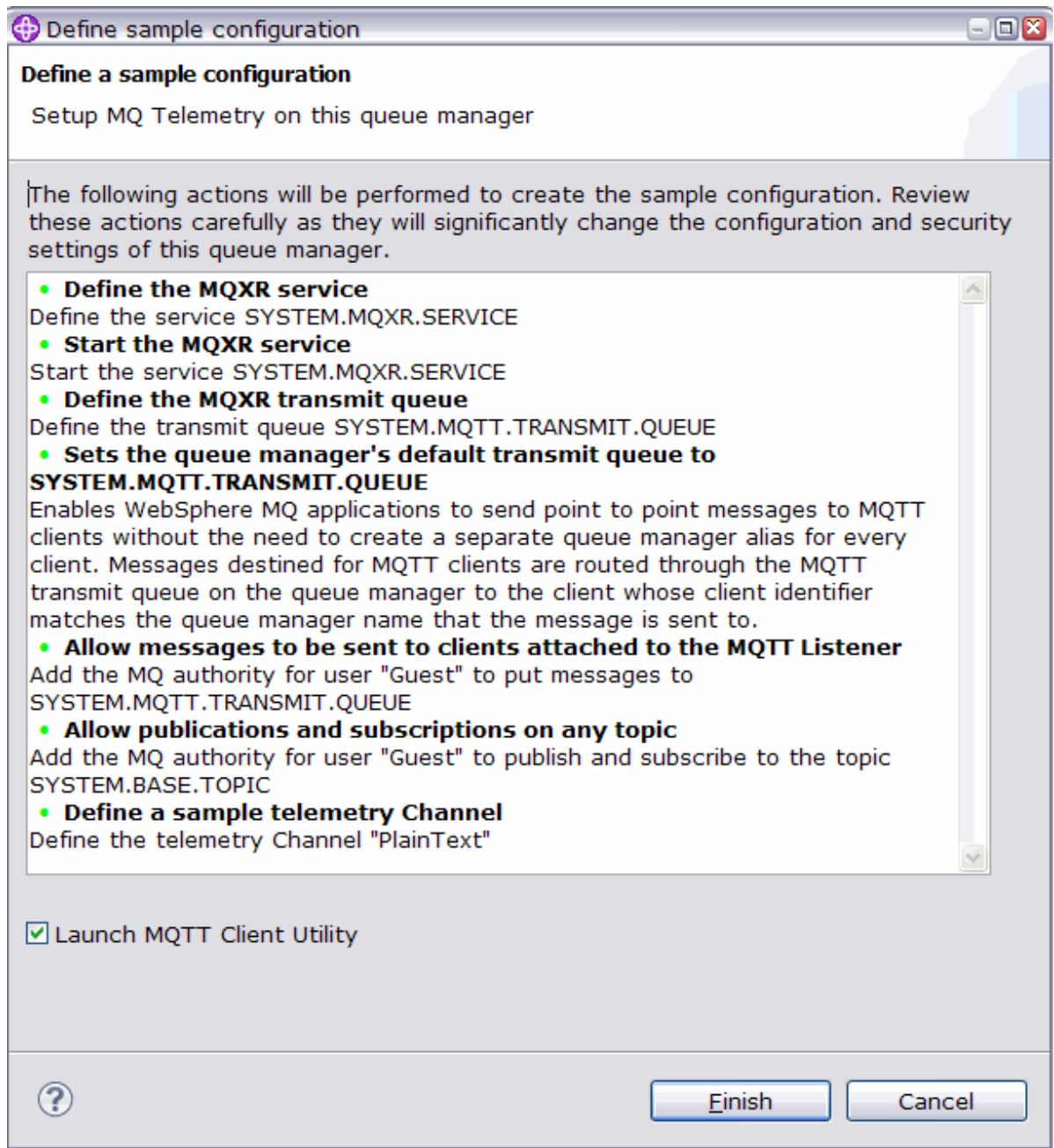
? < Back Next > Finish Cancel

IBM WebSphere MQ Explorer 會建立佇列管理程式並啟動它。

4. 執行「遙測」定義配置範例精靈。
 - a) 開啟佇列管理程式的 Telemetry 資料夾。

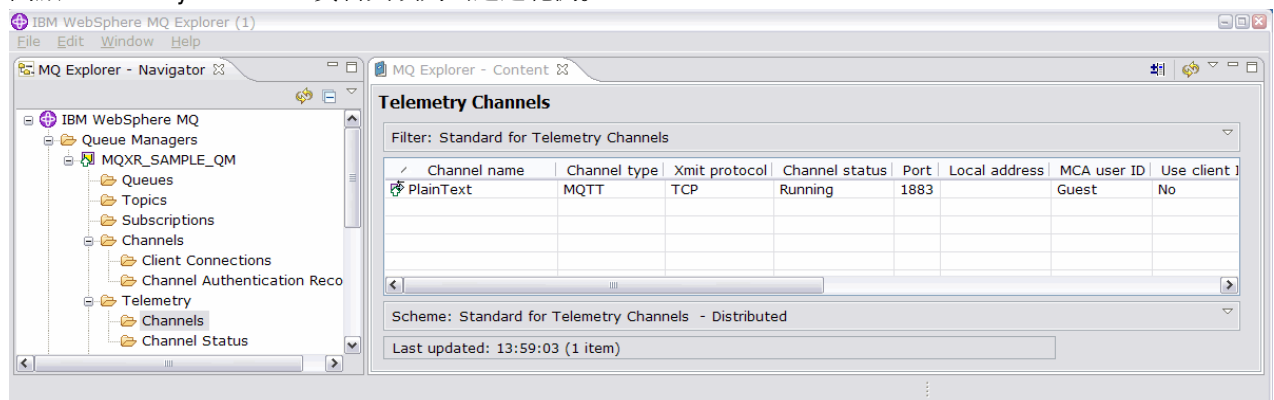


- b) 按一下**定義配置範例**以啟動精靈。
- c) 按一下**完成**以建立遙測服務，然後執行 MQTT 用戶端公用程式



結果

開啟 Telemetry Channels 資料夾以列出通道範例。

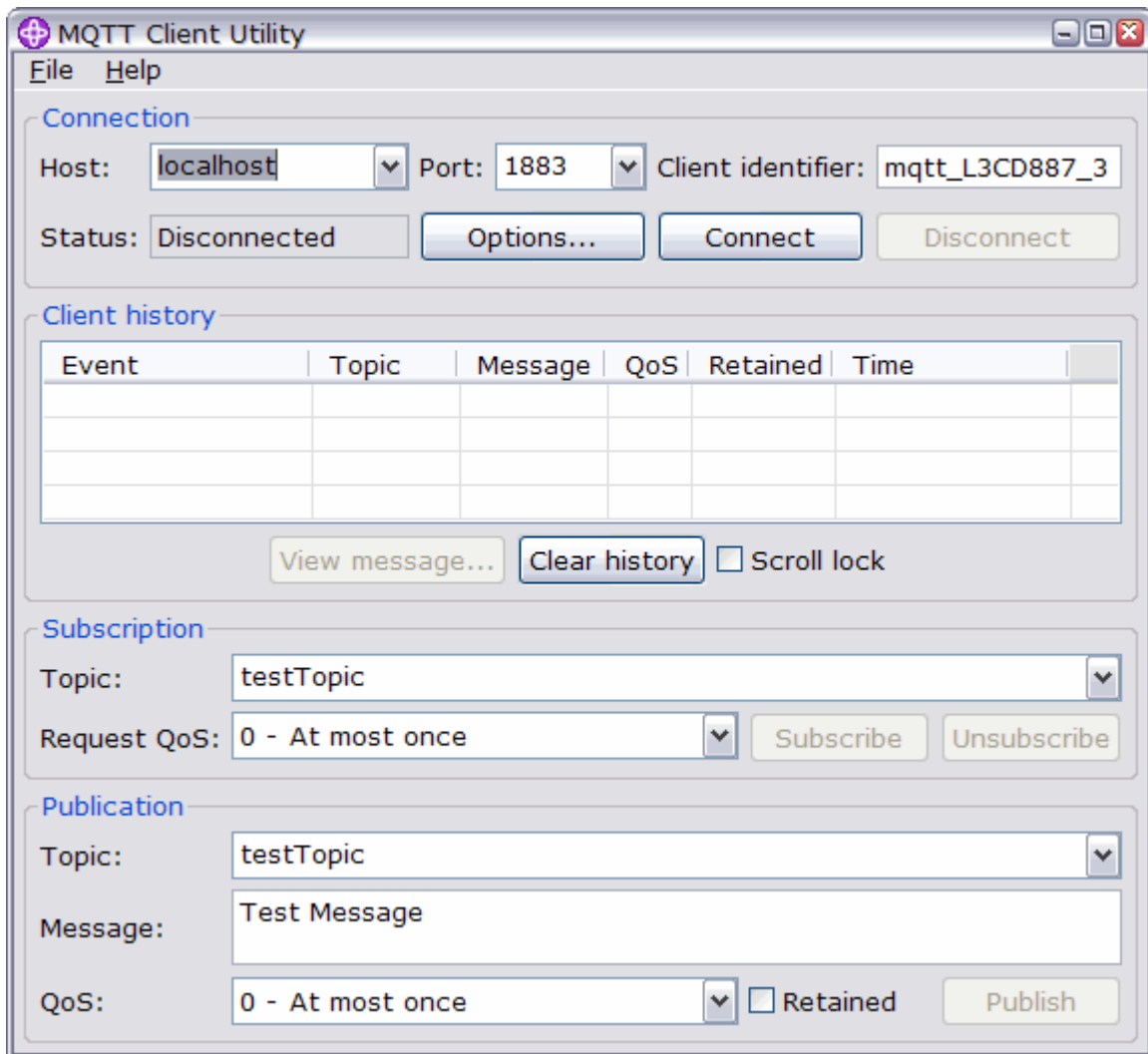


您可以修改此通道的內容，並在此視窗中新增及刪除通道。

下一步

透過執行 MQTT 用戶端公用程式來測試連線。

1. 若要啟動用戶端公用程式，請開啟 **Telemetry** 資料夾，然後按兩下執行 **MQTT 用戶端公用程式**。兩個「MQTT 用戶端公用程式」視窗即會開啟，這兩個視窗相同，但用戶端 ID 不同。



2. 在兩個視窗中按一下**連接**。
3. 在兩個視窗中按一下**訂閱**。
4. 在任一視窗中按一下**發佈**。結果會顯示在[第 134 頁的圖 29](#)中。

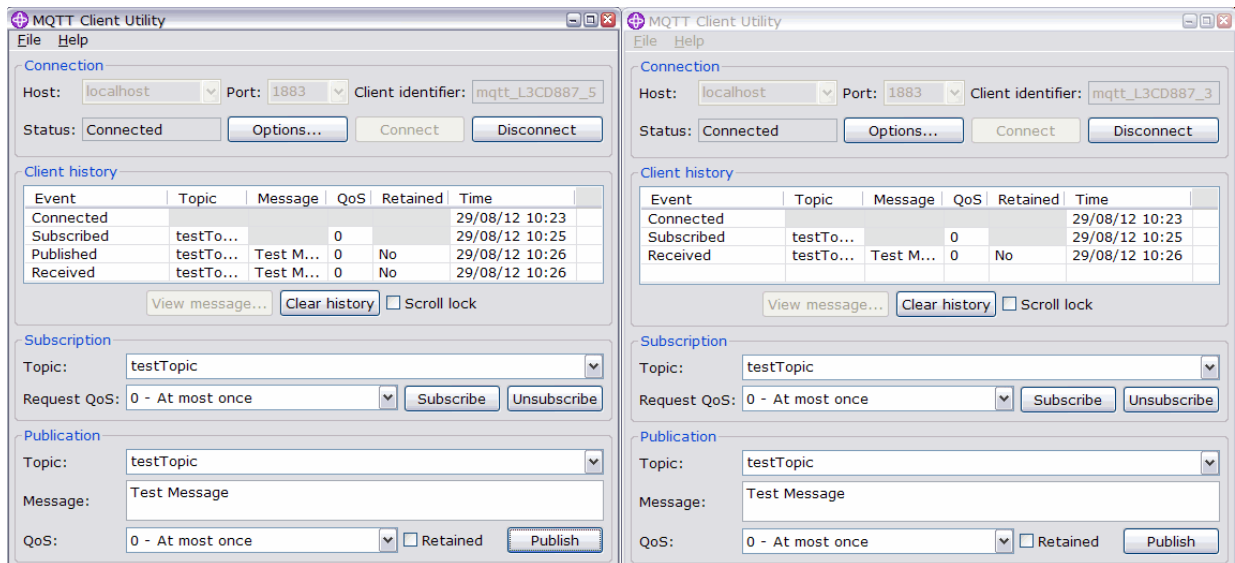


圖 29: 結果

5. 在兩個視窗中按一下**斷線**。

伺服器現在已備妥，可讓您測試 MQTT V3.1 應用程式。

相關工作

[從指令行配置 MQTT 服務](#)

遵循下列指示，以使用指令行配置 IBM WebSphere MQ 來執行 IBM WebSphere MQ Telemetry 應用程式範例。這些步驟顯示如何執行 Script，以在稱為 MQXR_SAMPLE_QM 的新佇列管理程式上建立 MQTT 服務。

[管理 WebSphere MQ Telemetry](#)

相關資訊

[WebSphere MQ Telemetry](#)

[使用 WebSphere MQ 探險家管理 WebSphere MQ Telemetry](#)

[開發 WebSphere MQ Telemetry 應用程式](#)

[安全](#)

[WebSphere MQ Telemetry 安全](#)

用於裝置的 IBM WebSphere MQ Telemetry 常駐程式概念

適用於裝置的 IBM WebSphere MQ Telemetry 常駐程式是一個進階 MQTT 第 3 版用戶端應用程式。可以利用它來儲存及轉遞來自其他 MQTT 用戶端的訊息。它可以如 MQTT 用戶端一樣連接至 IBM WebSphere MQ，但您也可以將它與其他 MQTT 用戶端連接。

常駐程式是一種發佈/訂閱分配管理系統。MQTT 第 3 版用戶端連接至該常駐程式來發佈至及訂閱主題，使用主題字串進行發佈，使用主題過濾器進行訂閱。主題字串是階層式的，主題層次以 / 分割。主題過濾器是可以包含單一層次 + 萬用字元的主題字串，並包含多層次 # 萬用字元作為主題字串的最後一部分。

註: 常駐程式中的萬用字元遵循 WebSphere Message Broker 第 6 版中較為嚴格的規則。IBM WebSphere MQ 則不同。它支援多個多層次萬用字元；萬用字元可以代表任何數目的階層層次，並且可以位於主題字串的任何位置。

多個 MQTT 第 3 版用戶端使用接聽器埠連接至常駐程式。可以修改預設接聽器埠。您可以定義多個接聽器埠並為其配置不同的名稱空間，請參閱第 140 頁的『[用於裝置的 WebSphere MQ Telemetry 常駐程式接聽器埠](#)』。常駐程式自身是一個 MQTT 第 3 版用戶端。配置常駐程式橋接器連線，可將常駐程式連接至另一個常駐程式的接聽器埠，或連接至 WebSphere MQ Telemetry (MQXR) 服務。

您可以為適用於裝置的 WebSphere MQ Telemetry 常駐程式配置多個橋接器。使用橋接器可將能夠交換發佈的常駐程式網路連接在一起。

每個橋接器都可以發佈至及訂閱其本端常駐程式上的主題。它還可以發佈至及訂閱其他常駐程式、WebSphere MQ 發佈/訂閱分配管理系統或其連接的任何其他 MQTT 第 3 版分配管理系統上的主題。透過使用主題過濾器，您可以選取要從一個分配管理系統傳送至另一個分配管理系統的發佈。您可以任一方向傳送

發佈。您可以將發佈從本端常駐程式傳送至其連接的每個遠端分配管理系統，或從連接的任一分配管理系統傳送至本端常駐程式；請參閱第 135 頁的『用於裝置的 IBM WebSphere MQ Telemetry 常駐程式橋接器』。

用於裝置的 IBM WebSphere MQ Telemetry 常駐程式橋接器

用於裝置的 IBM WebSphere MQ Telemetry 常駐程式橋接器，可使用 MQTT 第 3 版通訊協定連接兩個發佈/訂閱分配管理系統。該橋接器可以任一方向將發佈從一個分配管理系統傳送至另一個分配管理系統。一端為用於裝置的 WebSphere MQ Telemetry 常駐程式橋接器連線，另一端可以是佇列管理程式或另一個常駐程式。使用遙測通道可將佇列管理程式連接至橋接器連線。使用常駐程式接聽器可將常駐程式連接至橋接器連線。

裝置的 IBM WebSphere MQ Telemetry 常駐程式同時間可只有一條對其他分配管理系統的連線，也可有多條連線。來自常駐程式的連線稱為橋接器，並由常駐配置檔中的連線項目定義。使用 IBM WebSphere MQ 遙測通道來建立 IBM WebSphere MQ 連線，如下圖所示：

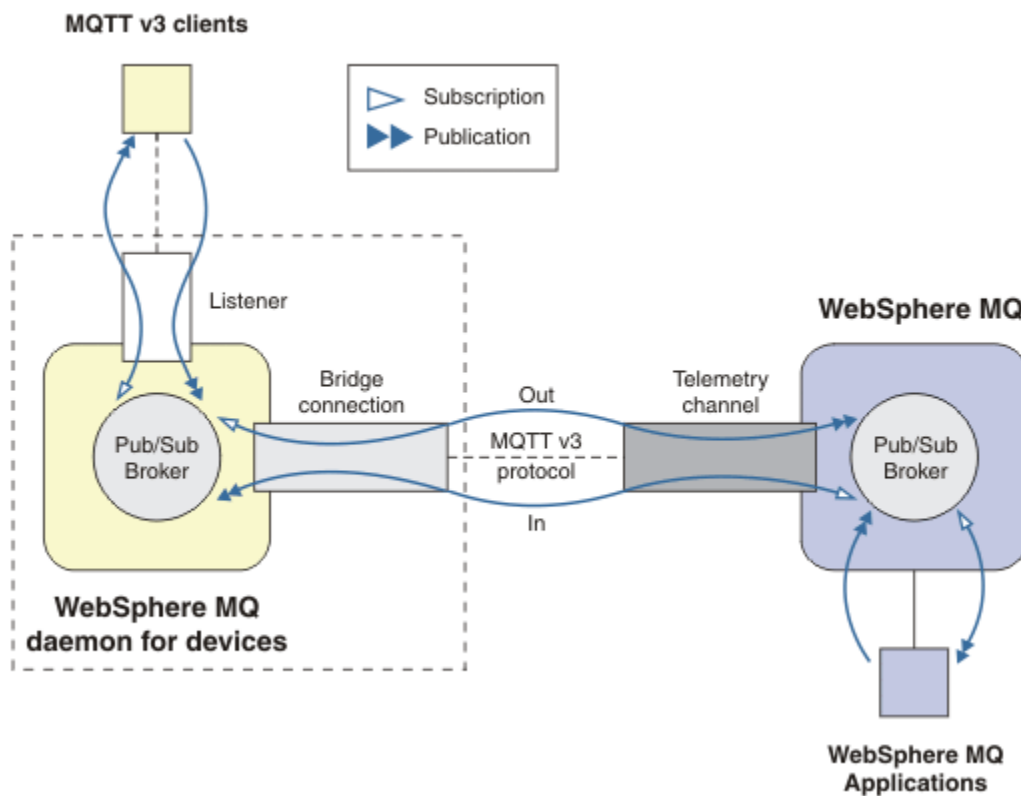


圖 30: 將 IBM WebSphere MQ Telemetry daemon for devices 連接至 IBM WebSphere MQ

橋接器可將常駐程式以 MQTT v3 用戶端形式連接至另一個分配管理系統。橋接器參數可鏡映 MQTT 第 3 版用戶端的屬性。

橋接器不只是一條連線。它還可作為兩個發佈/訂閱分配管理系統之間的發佈與訂閱代理程式。本端分配管理系統為裝置的 IBM WebSphere MQ Telemetry 常駐程式，遠端分配管理系統則是支援 MQTT 第 3 版通訊協定的任何發佈/訂閱分配管理系統。遠端分配管理系統一般是另一個常駐程式或 IBM WebSphere MQ。

橋接器的工作是在兩個分配管理系統之間傳送發佈。橋接器是雙向的。它可以任一方向傳送發佈。第 135 頁的圖 30 說明橋接器如何將裝置的 IBM WebSphere MQ Telemetry 常駐程式連接至 IBM WebSphere MQ。第 136 頁的『橋接器的主題設定範例』利用範例說明如何使用主題參數來配置橋接器。

第 135 頁的圖 30 中的 In 及 Out 箭頭表示橋接器具有雙向性。在箭頭的其中一端建立訂閱。符合訂閱的發佈會發佈至箭頭另一端的分配管理系統。箭頭是根據發佈流向標示的。發佈流向 In 表示流入常駐程式，

Out 則表示流出常駐程式。標籤的重要性是它們會在指令語法中使用。請記住，In and Out 參照發佈流向，而非訂閱的傳送方向。

其他用戶端、應用程式或分配管理系統可以連接至 IBM WebSphere MQ 或適用於裝置的 WebSphere MQ Telemetry 常駐程式。它們可發佈至及訂閱其所連接的分配管理系統上的主題。如果分配管理系統是 IBM WebSphere MQ，主題可能為叢集或分散式，並且未在本端佇列管理程式上明確定義。

橋接器的使用

使用橋接器連線及接聽器將常駐程式連接在一起。使用橋接器連線及遙測通道，將常駐程式及佇列管理程式連接在一起。將多個分配管理系統連接在一起時，可以建立迴圈。請注意，發佈可能會在分配管理系統迴圈中不斷地循環且無法偵測到。

使用常駐程式橋接至 IBM WebSphere MQ 的部分原因如下所示：

減少與 WebSphere MQ 的 MQTT 用戶端連線數目

透過使用常駐程式階層，您可以將多個用戶端連接至 WebSphere MQ；一次可以連接的用戶端數目多於單一佇列管理程式。

在 MQTT 用戶端與 WebSphere MQ 之間儲存及轉遞訊息

如果用戶端沒有自己的儲存體，您可以使用儲存及轉遞來避免維持用戶端與 IBM WebSphere MQ 之間的連續連線。您可以在 MQTT 用戶端與 WebSphere MQ 之間使用多種連線類型；請參閱[遙測概念及監視與控制實務範例](#)。

過濾 MQTT 用戶端與 WebSphere MQ 之間交換的發佈

發佈通常分為在本端處理的訊息，以及涉及其他應用程式的訊息。本端發佈可能包括控制感應器與掣動器之間的流向，而遠端發佈則包括讀數、狀態及配置指令要求。

變更發佈的主題空間

避免來自連接至不同接聽器埠之用戶端的主題字串，彼此發生衝突。範例使用常駐程式來標示來自不同大廈的計量讀數；請參閱[分隔不同用戶端群組的主題字串](#)。

橋接器的主題設定範例

將所有內容發佈至遠端分配管理系統 - 使用預設值

預設方向稱為 out，橋接器會將主題發佈至遠端分配管理系統。topic 參數使用主題過濾器來控制傳送的主題。

橋接器使用第 136 頁的圖 31 中的 topic 參數，來訂閱由 MQTT 用戶端或其他分配管理系統發佈至本端常駐程式的所有內容。橋接器會將主題發佈至橋接器所連接的遠端分配管理系統。

```
connection Daemon1
topic #
```

圖 31: 將所有內容發佈至遠端分配管理系統

將所有內容發佈至遠端分配管理系統 - 明確

下列程式碼片段中的 topic 設定提供與使用預設值相同的結果。唯一的差異是 **direction** 參數是明確的。使用 out 方向可訂閱本端分配管理系統、常駐程式，並發佈至遠端分配管理系統。橋接器訂閱的於本常駐程式上建立的發佈，會在遠端分配管理系統上進行發佈。

```
connection Daemon1
topic # out
```

圖 32: 將所有內容發佈至遠端分配管理系統 - 明確

將所有內容發佈至本端分配管理系統

您可以設定相反的方向 `in`，來取代使用方向 `out`。下列程式碼片段可將橋接器配置為訂閱橋接器所連接遠端分配管理系統上發佈的所有內容。橋接器會將主題發佈至本端分配管理系統，即常駐程式。

```
connection Daemon1
topic # in
```

圖 33: 將所有內容發佈至本端分配管理系統

將所有內容從本端分配管理系統上的 `export` 主題，發佈至遠端分配管理系統上的 `import` 主題

使用另外兩個參數 `local_prefix` 及 `remote_prefix`，來修改前一個範例中的主題過濾器 `#`。一個參數用於修改訂閱中所使用的主題過濾器，另一個參數用於修改將發佈發佈至其中的主題。如此一來，其中一個分配管理系統中使用的主題字串開頭，會由另一個分配管理系統上的另一個主題字串所取代。

根據主題指令的方向，`local_prefix` 及 `remote_prefix` 的意義是相反的。如果方向為 `out`（預設值），則 `local_prefix` 用作主題訂閱的一部分，`remote_prefix` 會取代遠端訂閱中主題字串的 `local_prefix` 部分。如果方向為 `in`，`remote_prefix` 會變成遠端訂閱的一部分，`local_prefix` 則會取代主題字串的 `remote_prefix` 部分。

在定義主題空間時，通常需要考量主題字串的第一部分。可使用其他參數，來變更將主題發佈至其中的主題空間。您可以透過此舉來避免要傳送的主題與目標分配管理系統上的另一個主題發生衝突，或透過此舉移除裝載點主題字串。

例如，在下列程式碼片段中，會將對常駐程式上主題字串 `export/#` 的所有發佈，重新發佈至遠端分配管理系統上的 `import/#`。

```
topic # out export/ import/
```

圖 34: 將所有內容從本端分配管理系統上的 `export` 主題，發佈至遠端分配管理系統上的 `import` 主題

將所有內容從遠端分配管理系統上的 `export` 主題，發佈至本端分配管理系統上的 `import` 主題

下列程式碼片段顯示相反的配置，橋接器會訂閱以遠端分配管理系統上的 `export/#` 主題字串發佈的所有內容，並將該內容發佈至本端分配管理系統上的 `import/#`。

```
connection Daemon1
topic # in import/ export/
```

圖 35: 將所有內容從遠端分配管理系統上的 `export` 主題，發佈至本端分配管理系統上的 `import` 主題

將來自 `1884/` 裝載點的所有內容，以原始主題字串發佈至遠端分配管理系統

在下列程式碼片段中，橋接器會訂閱連接至本端常駐程式裝載點 `1884/` 的用戶端發佈的所有內容。橋接器會將發佈至該裝載點的所有內容，發佈至遠端分配管理系統。發佈至遠端分配管理系統的主題中會移除裝載點字串 `1884/`。`local_prefix` 與裝載點字串 `1884/` 相同，而 `remote_prefix` 為空白字串。

```
listener 1884
mount_point 1884/
connection Daemon1
topic # out 1884/ ""
```

圖 36: 將來自 `1884/` 裝載點的所有內容，以原始主題字串發佈至遠端分配管理系統。

分隔不同用戶端（連接至不同常駐程式）的主題空間

假設要為電源電錶撰寫應用程式，以用於發佈建築物的電錶讀數。使用 MQTT 用戶端將讀數發佈至位於同一棟建築物的常駐程式。選取的發佈主題為 `power`。將相同的應用程式部署至綜合大樓中的多棟建築物。為了實現網站監視及資料儲存，會使用橋接器連線聚集來自所有建築物的讀數。這些連線會將大廈常駐程式鏈結至位於集中位置的 WebSphere MQ。

所有建築物都使用相同的用戶端應用程式。此應用程式會發佈至主題 `power`。但是資料必須依建築物加以區分。常駐程式會為每一棟建築物完成此步驟，而將建築物號碼新增為主題名稱的字首。來自綜合大樓第一棟建築物的橋接器使用字首 `meters/building01/`，來自第二棟建築物的字首為 `meters/building02/`。來自其他建築物的讀數遵循相同的模式。WebSphere MQ 因此會收到含有 `meters/building01/power` 之類主題的讀數。

每個常駐程式的配置檔都具有遵循下列程式碼片段中型樣的主題陳述式：

```
connection Daemon1
topic power out "" meters/building01/
```

圖 37: 分隔用戶端（連接至不同常駐程式）的主題空間

在上述程式碼片段中，空字串是未用 `local_prefix` 參數的位置保留元。

註：此範例經刻意設計，僅供示範之用。在實際情況下，可能可以配置應用程式發佈目的地的主題空間。

分隔用戶端（連接至相同常駐程式）的主題空間

假設使用單一常駐程式來連接所有電源計量器。假設在應用程式中可以配置為連接至不同埠，您可以透過將來自不同建築物的計量器連接至不同接聽器埠來識別建築物；如下列程式碼片段中所示。再次聲明，範例是刻意設計的；它說明可以如何使用裝載點。

```
listener 1884
mount_point meters/building01/
listener 1885
mount_point meters/building02/
connection Daemon1
topic meters/+/power out
```

圖 38: 分隔用戶端（連接至相同常駐程式）的主題空間

重新對映在兩個方向流動的發佈的不同主題

在下列程式碼片段的配置中，橋接器會訂閱遠端分配管理系統上的單一主題 `b`，並將 `b` 的相關發佈轉遞至本端常駐程式，將主題變更為 `a`。橋接器也會訂閱本端分配管理系統上的單一主題 `x`，並將 `x` 的相關出版品轉遞至遠端分配管理系統，將主題變更為 `y`。

```
connection Daemon1
topic "" in a b
topic "" out x y
```

圖 39: 重新對映在兩個方向流動的發佈的不同主題

此範例的重點是訂閱及發佈至兩個分配管理系統上的不同主題。兩個分配管理系統上的主題空間未連接。

重新對映在兩個方向流動的發佈的相同主題（迴圈）

與前一個範例不同，第 139 頁的圖 40 中的配置通常會導致迴圈。在主題陳述式 `topic "" in a b` 中，橋接器會從遠端訂閱 `b`，並在本端發佈至 `a`。在另一個主題陳述式中，橋接器會在本端訂閱 `a`，並從遠端發佈至 `b`。可以按第 139 頁的圖 41 所示撰寫相同的配置。

一般結果是如果用戶端從遠端發佈至 **b**，則發佈會以主題 **a** 的發佈方式傳送至本端常駐程式。不過，當橋接器發佈至主題 **a** 上的本端常駐程式時，發佈會比對橋接器對本端主題 **a** 所進行的訂閱。訂閱為 `topic "" out a b`。因此，發佈資訊會作為主題 **b** 的發佈資訊傳回遠端分配管理系統。橋接器現在已訂閱遠端主題 **b**，且循環重新開始。

部分分配管理系統會實作迴圈偵測，以防止發生迴圈。但是，將不同類型的分配管理系統橋接在一起時，迴圈偵測機制必須能正常運作。如果將 WebSphere MQ 橋接至用於裝置的 WebSphere MQ Telemetry 常駐程式，則迴圈偵測無法運作。橋接的兩端必須是裝置的 IBM WebSphere MQ Telemetry 常駐程式，此機制才能運作。依預設會開啟迴圈偵測；請參閱 [try_private](#)。

```
connection Daemon1
topic "" in a b
topic "" out a b
```

圖 40: 針對雙向流動的發佈重新對映相同的主題

```
connection Daemon1
topic "" both a b
```

圖 41: 使用 `both`，針對雙向流動的發佈重新對映相同的主題。

第 138 頁的圖 39 中的配置與第 139 頁的圖 40 相同。

IBM WebSphere MQ Telemetry daemon for devices 橋接器連線的可用性

配置多個 IBM WebSphere MQ Telemetry daemon for devices 橋接器連線位址，以連接至第一個可用的遠端分配管理系統。如果分配管理系統是多重實例佇列管理程式，請提供其兩個 TCP/IP 位址。將主要連線配置為在主要伺服器可用時，連接或重新連接至該伺服器。

連線橋接器參數 `addresses`，是 TCP/IP 通訊端位址清單。橋接器會依次嘗試連接至每個位址，直到其成功建立連線為止。`round_robin` 及 `start_type` 連線參數，用於控制成功建立連線後位址的使用方式。

如果 `start_type` 為 `auto`、`manual` 或 `lazy`，則如果連線失敗，橋接器便會嘗試進行重新連接。它會依次使用每個位址，每個連線嘗試之間會有大約 20 秒的延遲。如果 `start_type` 為 `once`，則如果連線失敗，橋接器不會自動嘗試進行重新連接。

如果 `round_robin` 為 `true`，則橋接器連線會從清單中的第一個位址開始嘗試，並依次嘗試清單中的每一個位址。當清單用盡時，它會重新從第一個位址開始嘗試。如果清單中只有一個位址，它便會每隔 20 秒重新嘗試一次。

如果 `round_robin` 為 `false`，則清單中的第一個位址（該位址稱為主要伺服器）具有優先權。如果第一次嘗試連接至主要伺服器失敗，橋接器會在背景繼續嘗試重新連接至主要伺服器。同時，橋接器會嘗試使用清單中的其他位址進行連接。如果背景嘗試連接至主要伺服器成功，橋接器便會中斷現行連線，並切換至主要伺服器連線。

如果連線主動中斷（例如透過發出 `connection_stop` 指令），則重新啟動連線時，它會嘗試仍使用相同的位址。如果連線因連接失敗或遠端分配管理系統捨棄連線而中斷，橋接器會等待 20 秒。之後，它便會嘗試連接至清單中的下一個位址，如果清單中只有一個位址，它會嘗試連接至相同位址。

連接至多重實例佇列管理程式

在多重實例佇列管理程式配置中，佇列管理程式會在兩個具有不同 IP 位址的不同伺服器上執行。一般來說，不會以特定 IP 位址來配置遙測通道。它們僅以埠號進行配置。遙測通道啟動後，依預設它會選取本端伺服器上第一個可用的網址。

以佇列管理程式使用的兩個 IP 位址，來配置橋接器連線的 `addresses` 參數。將 `round_robin` 設定為 `true`。

如果作用中佇列管理程式實例失敗，佇列管理程式便會切換至待命實例。常駐程式可偵測到作用中實例的連線已中斷，並嘗試重新連接至待命實例。它會使用為橋接器連線配置的位址清單中的其他 IP 位址。

橋接器連接的佇列管理程式仍為同一個佇列管理程式。佇列管理程式會回復其自己的狀態。如果將 `cleansession` 設定為 `false`，橋接器連線會恢復為失效接手之前的相同狀態。連線會在延遲之後回復。具有 "至少一次" 或 "最多一次" 服務品質的訊息不會遺失，且訂閱會繼續運作。

重新連線時間取決於待命實例啟動時重新啟動的通道及用戶端數，以及進行中的訊息數。在重新建立連線之前，橋接器連線可能會嘗試重新連接至這兩個 IP 位址次數。

請勿以特定的 IP 位址配置多重實例佇列管理程式遙測通道。IP 位址僅在一部伺服器上有效。

如果您是使用替代的高可用性解決方案來管理 IP 位址，則它可能更正為以特定的 IP 位址配置遙測通道。

cleansession

橋接器連線為 MQTT v3 用戶端階段作業。您可以控制連線是否啟動新階段作業，或者它是否還原現有階段作業。如果它還原現有階段作業，則橋接器連線會保留前一個階段作業的訂閱及保留的發佈。

如果 `addresses` 列出多個 IP 位址，並且這些 IP 位址連接至由不同佇列管理程式管理的遙測通道，或連接至不同遙測通道，則請勿將 `cleansession` 設定為 `false`。不會在佇列管理程式或常駐程式之間傳送階段作業狀態。如果嘗試在不同佇列管理程式或常駐程式上重新啟動現有階段作業，會導致啟動新的階段作業。不確定的訊息會遺失，並且訂閱可能會發生未預期的行為。

notifications

透過使用通知，應用程式可以追蹤橋接器連線是否在執行中。通知是具有值 1（已連接），或 0（已中斷連線）的發佈。它會發佈至 `notification_topic` 參數所定義的 `topicString`。`topicString` 的預設值是 `$/SYS/broker/connection/clientIdentifier/state`。預設 `topicString` 包含字首 `$/SYS`。透過定義開頭為 `$/SYS` 的主題過濾器，來訂閱開頭為 `$/SYS` 的主題。主題過濾器 `#` 可訂閱所有主題，但不會訂閱常駐程式上開頭為 `$/SYS` 的主題。在定義與應用程式主題空間截然不同的特殊系統主題空間時，請考慮使用 `$/SYS`。

通知可讓 IBM WebSphere MQ Telemetry daemon for devices 在橋接器已連接或中斷連線時通知 MQTT 用戶端。

keepalive_interval

`keepalive_interval` 橋接器連線參數，可設定橋接器將 TCP/IP 連線測試傳送至遠端伺服器的間隔。預設間隔為 60 秒。連線測試可防止遠端伺服器或防火牆（會偵測連線的閒置時間）關閉 TCP/IP 階段作業。

clientid

橋接器連線為 MQTT 第 3 版用戶端階段作業，並且具有橋接器連線參數 `clientid` 所設定的 `clientIdentifier`。如果您想要透過將 `cleansession` 參數設定為 `false` 進行重新連線以回復前一個階段作業，則每個階段作業中使用的 `clientIdentifier` 必須相同。`clientid` 的預設值保持不變仍為 `hostname.connectionName`。

安裝、驗證、配置及控制用於裝置的 WebSphere MQ Telemetry 常駐程式

安裝、配置及控制常駐程式以檔案為基礎。

透過將「軟體開發套件」複製到要在其中執行常駐程式的裝置，來安裝常駐程式。

例如，執行 MQTT 用戶端公用程式，並以發佈/訂閱分配管理系統形式連接至用於裝置的 WebSphere MQ Telemetry 常駐程式；請參閱使用適用於裝置的 WebSphere MQ Telemetry 常駐程式作為發佈/訂閱分配管理系統。

透過建立配置檔配置常駐程式；請參閱用於裝置的 WebSphere MQ Telemetry 常駐程式配置檔。

透過在檔案 `amqtdl.upd` 中建立指令來控制執行中的常駐程式。每隔 5 秒常駐程式就會讀取該檔案、執行指令並刪除檔案；請參閱用於裝置的 WebSphere MQ Telemetry 常駐程式指令檔。

用於裝置的 WebSphere MQ Telemetry 常駐程式接聽器埠

透過使用接聽器埠，將 MQTT 第 3 版用戶端連接至用於裝置的 WebSphere MQ Telemetry 常駐程式。您可以使用裝載點及連線數目上限來限定接聽器埠。

接聽器埠必須對應於連接至此埠之用戶端的 MQTT 用戶端 `connect(serverURI)` 方法上指定的埠號。它在用戶端及常駐程式上均使用預設值 1883。

透過設定常駐配置檔中的廣域定義 `port`，您可以變更常駐程式的預設埠。透過在常駐配置檔中新增 `listener` 定義，您可以設定特定的埠。

對於預設埠以外的每個接聽器埠，您都可以指定裝載點以隔離用戶端。連接至具有裝載點之埠的用戶端會與其他用戶端隔離；請參閱第 141 頁的『用於裝置的 WebSphere MQ Telemetry 常駐程式裝載點』。

您可以限制可以連接至任一埠的用戶端數目。請設定廣域定義 `max_connections` 以限制預設埠的連線，或使用 `max_connections` 限定每個接聽器埠。

範例

以下配置檔範例可將預設埠由 1883 變更為 1880，並將埠 1880 的連線數限制為 10000。埠 1884 的連線數限制為 1000。連接至埠 1884 的用戶端與連接至其他埠的用戶端隔離。

```
port 1880
max_connections 10000
listener 1884
mount_point 1884/
max_connections 1000
```

用於裝置的 WebSphere MQ Telemetry 常駐程式裝載點

您可以將裝載點與 MQTT 用戶端所使用的接聽器埠關聯，以連接至用於裝置的 WebSphere MQ Telemetry 常駐程式。裝載點會將使用一個接聽器埠的 MQTT 用戶端所交換的發佈及訂閱，與連接至其他接聽器埠的 MQTT 用戶端隔離。

連接至具有裝載點之接聽器埠的用戶端，無法與連接至任何其他接聽器埠的用戶端直接交換主題。連接至沒有裝載點之接聽器埠的用戶端，可以發佈至或訂閱任何用戶端的主題。用戶端無法知曉它們是否透過裝載點連接；這對用戶端所建立的主題字串來說沒有差別。

裝載點是加在發佈及訂閱主題字串前面的文字字串。它會加在用戶端（連接至具有裝載點的接聽器埠）建立的所有主題字串的前面。傳送至連接接聽器埠之用戶端的所有主題字串中會移除該文字字串。

如果接聽器埠沒有裝載點，則不會變更連接至該埠之用戶端所建立及接收的發佈及訂閱的主題字串。

建立尾端為 / 的裝載點字串。如此一來，裝載點便成為裝載點主題樹狀結構的上層主題。

範例

配置檔包含下列接聽器埠：

```
listener 1883
mount_point 1883/
listener 1884 127.0.0.1
mount_point 1884/
listener 1885
```

連接至埠 1883 的用戶端會建立對 MyTopic 的訂閱。常駐程式會將該訂閱登錄為 1883/MyTopic。連接至埠 1883 的另一個用戶端會在主題 MyTopic 上發佈訊息。常駐程式會將主題字串變更為 1883/MyTopic，並搜尋相符的訂閱。埠 1883 上的訂閱者會接收含有原始主題字串 MyTopic 的發佈。常駐程式已從主題字串中移除裝載點字首。

連接至埠 1884 的另一個用戶端也會在主題 MyTopic 上進行發佈。此時，常駐程式會將主題登錄為 1884/MyTopic。埠 1883 上的訂閱者不會收到發佈，因為不同的裝載點會導致含有不同主題字串的訂閱。

連接至埠 1885 的用戶端會在主題 1883/MyTopic 上進行發佈。常駐程式不會變更主題字串。埠 1883 上的訂閱者會接收 MyTopic 的發佈。

用於裝置的 WebSphere MQ Telemetry 常駐程式服務品質、可延續訂閱及保留的發佈

服務品質設定僅適用於執行中的常駐程式。如果常駐程式停止（透過採用受控方式或因發生故障），進行中訊息的狀態會遺失。如果常駐程式停止，則無法保證遞送訊息至少一次，或至多一次。用於裝置的 WebSphere MQ Telemetry 常駐程式支援有限的持續性。設定 **retained_persistence** 配置參數，可在常駐程式關閉時儲存保留的發佈及訂閱。

與 WebSphere MQ 不同，適用於裝置的 WebSphere MQ Telemetry 常駐程式不會日誌登載持續資料。不會以交易方式儲存階段作業狀態、訊息狀態及保留的發佈。依預設，常駐程式在其停止時會捨棄所有資料。您可以設定一個選項，以定期儲存訂閱及保留的發佈之檢查點。常駐程式停止時，訊息狀態會一律遺失。所有非保留的發佈都會遺失。

將常駐程式配置選項 `Retained_persistence` 設為 `true`，可定期將保留的發佈儲存至檔案。常駐程式重新啟動時，即會恢復前次自動儲存的保留的發佈。依預設，常駐程式重新啟動時不會恢復由用戶端建立的保留訊息。

將常駐程式配置選項 `Retained_persistence` 設為 `true`，可定期將持續性階段作業中建立的訂閱儲存至檔案。如果 `Retained_persistence` 設為 `true`，則會還原用戶端在 `CleanSession` 設為 `false` ("持續性階段作業") 的階段作業中建立的訂閱。常駐程式會在其重新啟動後開始接收發佈時還原訂閱。在 `CleanSession` 設為 `false` 的情況下，用戶端會在其重新啟動時接收發佈。依預設，在常駐程式停止時不會儲存用戶端階段作業狀態，因此不會還原訂閱，即使用戶端將 `CleanSession` 設為 `false` 也是如此。

`Retained_persistence` 是一種自動儲存機制。它可能無法儲存最新保留的發佈或訂閱。您可以變更儲存保留的發佈及訂閱的頻率。使用配置選項 `autosave_on_changes` 及 `autosave_interval`，可設定儲存間隔或儲存之間的變更次數。

設定持續性的範例配置

```
# Sample configuration
# Daemon listens on port 1882 with persistence in /tmp
# Autosave every minute
port 1882
persistence_location /tmp/
retained_persistence true
autosave_on_changes false
autosave_interval 60
```

用於裝置的 WebSphere MQ Telemetry 常駐程式安全

用於裝置的 WebSphere MQ Telemetry 常駐程式可以鑑別它所連接的用戶端、使用認證連接至其他分配管理系統，以及控制主題的存取。該常駐程式提供的安全有所限制，因為它是使用不提供 SSL 支援的 WebSphere MQ Telemetry C 用戶端進行建置。因此，進出常駐程式的連線不會加密，並且無法使用憑證進行鑑別。

依預設，不會開啟任何安全。

用戶端鑑別

MQTT 用戶端可以使用 `MqttConnectOptions.setUsername` 及 `MqttConnectOptions.setPassword` 方法，來設定使用者名稱及密碼。

透過針對密碼檔中的項目檢查用戶端所提供的使用者名稱及密碼，來鑑別連接至常駐程式的用戶端。若要啟用鑑別，請建立密碼檔並設定常駐配置檔中的 `password_file` 參數；請參閱 [password_file](#)。

設定常駐配置檔中的 `allow_anonymous` 參數，可容許沒有使用者名稱或密碼的用戶端進行連接，以連接至檢查鑑別的常駐程式；請參閱 `allow_anonymous`。如果已設定 `password_file` 參數，則在用戶端未提供使用者名稱或密碼時會一律針對密碼檔對其進行檢查。

設定常駐配置檔中的 `clientid_prefixes` 參數，可限制對特定用戶端的連線。用戶端必須具有以 `clientid_prefixes` 參數中所列其中一個字首為開頭的 `clientIdentifiers`；請參閱 `clientid_prefixes`。

橋接器連線安全

每個用於裝置的 WebSphere MQ Telemetry 常駐程式橋接器連線自身都是一個 MQTT 第 3 版用戶端。您可以在常駐配置檔中，針對每個橋接器連線設定 `username` 及 `password` 作為橋接器連線參數；請參閱 `username` 及 `password`。之後，橋接器即可向分配管理系統鑑別自身。

主題的存取控制

如果正在鑑別用戶端，該常駐程式還可以針對每位使用者提供對主題的存取控制。常駐程式會根據用戶端發佈至或訂閱的主題是否與存取控制檔中的存取主題字串相符，來授與存取控制權；請參閱 `acl_file`。

存取控制清單包含兩部分。第一部分用於控制對所有用戶端（包括匿名用戶端）的存取。第二部分包含針對密碼檔中任何使用者的區段。它會列出每位使用者的特定存取控制。

範例

下列範例顯示了安全參數。

```
acl_file c:\WMQTDaemon\config\acl.txt
password_file c:\WMQTDaemon\config\passwords.txt
allow_anonymous true
connection Daemon1
username daemon1
password deamonpassword
```

圖 42: 常駐配置檔

```
Fred:Fredpassword
Barney:Barneypassword
```

圖 43: 密碼檔，`passwords.txt`

```
topic home/public/#
topic read meters/#
user Fred
topic write meters/fred
topic home/fred/#
user Barney
topic write meters/barney
topic home/barney/#
```

圖 44: 存取控制檔，`acl.txt`

疑難排解 MQTT 用戶端

尋找疑難排解作業，以協助您解決執行 MQTT 用戶端時出現的問題。

相關工作

第 151 頁的『[追蹤及除錯 MQTT \(Paho\) Java 用戶端](#)』

預設日誌程式使用標準 Java 記載機能 (稱為 `java.util.logging` (JSR47))。您可以透過使用配置檔或是以程式化方式來配置它。

第 153 頁的『[追蹤 MQTT JavaScript 用戶端](#)』

透過變更用於對所連接用戶端物件呼叫方法的用戶端 Web 應用程式，即可使用 JavaScript 用戶端來收集追蹤。

第 147 頁的『[追蹤遙測 \(MQXR\) 服務](#)』

遵循下列指示，以啟動遙測服務追蹤、設定用於控制追蹤的參數及尋找追蹤輸出。

第 148 頁的『[追蹤 MQTT 第 3 版 Java 用戶端](#)』

遵循下列指示，以建立 MQTT Java 用戶端追蹤並控制其輸出。

[第 150 頁的『追蹤適用於 C 的 MQTT 用戶端』](#)

設定環境變數 `MQTT_C_CLIENT_TRACE`，以追蹤 MQTT 用戶端 C 應用程式

[第 158 頁的『解決問題：MQTT 用戶端未連接』](#)

解決 MQTT 用戶端程式無法連接至遙測 (MQXR) 服務的問題。

[第 160 頁的『解決問題：MQTT 用戶端連線中斷』](#)

找出導致用戶端在順利連接並執行或長或短的一段時間之後，擲出非預期的 `ConnectionLost` 異常狀況的原因。

[第 160 頁的『解決問題：MQTT 應用程式中遺失訊息』](#)

解決遺失訊息問題。訊息是非持續性的、傳送至錯誤的位置還是從未傳送？編寫不正確的用戶端程式可能會遺失訊息。

[第 162 頁的『解決問題：遙測 \(MQXR\) 服務未啟動』](#)

解決遙測 (MQXR) 服務無法啟動的問題。檢查 WebSphere MQ Telemetry 安裝，確定沒有任何檔案遺漏、被移動或具有錯誤的許可權。檢查遙測 (MQXR) 服務所使用的路徑是否可以找到遙測 (MQXR) 服務程式。

[第 163 頁的『解決問題：Telemetry 服務未呼叫 JAAS 登入模組』](#)

瞭解遙測 (MQXR) 服務是否未呼叫您的 JAAS 登入模組，並配置 JAAS 以更正問題。

[第 165 頁的『解決問題：啟動或執行常駐程式』](#)

查閱適用於裝置的 IBM WebSphere MQ Telemetry 常駐程式主控台日誌、開啟追蹤或使用此主題中的症狀表格，來疑難排解常駐程式問題。

[第 166 頁的『解決問題：MQTT 用戶端未連接至常駐程式』](#)

用戶端未連接至常駐程式，或是常駐程式未連接至其他常駐程式或 WebSphere MQ Telemetry 通道。

相關參考

[第 144 頁的『Telemetry 日誌、錯誤日誌和配置檔的位置』](#)

尋找 IBM WebSphere MQ Telemetry 使用的日誌、錯誤日誌及配置檔。

[第 146 頁的『MQTT v3 Java 用戶端原因碼』](#)

在 MQTT v3 Java 用戶端異常狀況或可擲出異常狀況中查閱原因碼的原因。

[第 154 頁的『對 MQTT 用戶端使用 SHA-2 密碼組合的系統需求』](#)

對於從 IBM SR13 開始的 Java 6，您可以使用 SHA-2 密碼組合來保護 MQTT 通道及用戶端應用程式的安全。不過，在從 IBM SR4 開始的 Java 7 之前，依預設不會啟用 SHA-2 密碼組合，因此在舊版中，您必須指定所需的組合。若您是搭配自己的 JRE 執行 MQTT 用戶端，必須確認其支援 SHA-2 密碼組合。若要讓用戶端應用程式使用 SHA-2 密碼組合，用戶端還必須將 SSL 環境定義設定為支援「傳輸層安全 (TLS)」1.2 版的值。

[第 155 頁的『瀏覽器對於使用 SSL 之行動式傳訊 Web 應用程式的支援限制』](#)

這在不同瀏覽器與平台組合上會有所差異。瞭解這些差異對於您配置應用程式、憑證管理中心 (CA)，以及使用適用於 JavaScript 的 MQTT 傳訊用戶端經由 SSL 及 WebSockets 連線時所需的用戶端憑證會很有幫助。

Telemetry 日誌、錯誤日誌和配置檔的位置

尋找 IBM WebSphere MQ Telemetry 使用的日誌、錯誤日誌及配置檔。

註：這些範例的程式碼是針對 Windows 所撰寫。變更語法以在 Linux 上執行範例

伺服器端日誌

IBM WebSphere MQ Telemetry 的安裝精靈會將訊息寫入其安裝日誌：

```
WMQ program directory\mqxr
```


遙測 (MQXR) 服務會將訊息寫入 WebSphere MQ 佇列管理程式錯誤日誌，以及 IBM WebSphere MQ 錯誤目錄中的 FDC 檔案：

```
WMQ data directory\Qmgrs\qMgrName\errors\AMQERR01.LOG
WMQ data directory\errors\AMQnnn.n.FDC
```

它還會寫入遙測 (MQXR) 服務的日誌。日誌會顯示用於啟動服務的內容，以及在它用作 MQTT 用戶端的代理時發現的錯誤。例如，取消訂閱用戶端未建立的訂閱。日誌路徑是：

```
WMQ data directory\Qmgrs\qMgrName\errors\mqxr.log
```

由「IBM WebSphere MQ 探險家」所建立的 IBM WebSphere MQ Telemetry 配置範例，可使用指令 **runMQXRService** 啟動遙測服務。**runMQXRService** 位於 *WMQ Telemetry install directory\bin* 中。它會寫入：

```
WMQ data directory\Qmgrs\qMgrName\mqxr.stdout
WMQ data directory\Qmgrs\qMgrName\mqxr.stderr
```

在啟動遙測 (MQXR) 服務之前，請先修改 **runMQXRService** 以顯示為遙測 (MQXR) 服務配置的路徑，或回應起始設定。

伺服器端配置檔

遙測通道及遙測 (MQXR) 服務

限制: 遙測通道配置檔的格式、位置、內容和解譯在未來的版本中可能會發生變化。您必須使用「IBM WebSphere MQ 探險家」來配置遙測通道。

「IBM WebSphere MQ 檔案總管」會將遙測配置儲存在 Windows 上的 `mqxr_win.properties` 檔案中，以及 Linux 上的 `mqxr_unix.properties` 檔案中。這些內容檔會儲存在遙測配置目錄中：

```
WMQ data directory\Qmgrs\qMgrName\mqxr
```

圖 45: Windows 上的遙測配置目錄

```
/var/mqm/qmgrs/qMgrName/mqxr
```

圖 46: Linux 上的遙測配置目錄

JVM

在檔案 `java.properties` 中設定作為引數傳遞至遙測 (MQXR) 服務的 Java 內容。該檔案中的內容會直接傳遞至執行遙測 (MQXR) 服務的 JVM。它們在 Java 指令行上作為其他 JVM 內容傳遞。指令行上所設定的內容，優先於從 `java.properties` 檔案新增至指令行的內容。

在遙測配置的相同資料夾中尋找 `java.properties` 檔案，請參閱 [第 145 頁的圖 45](#) 和 [第 145 頁的圖 46](#)。

以分隔線形式指定每個內容，修改 `java.properties`。請確實按照將內容作為引數傳遞至 JVM 的格式格式化每個內容，例如：

```
-Xmx1024m
-Xms1024m
```

JAAS

JAAS 配置檔在 [遙測通道 JAAS 配置](#) 中說明，其中包含 IBM WebSphere MQ Telemetry 隨附的範例 JAAS 配置檔 `JAAS.config`。

如果您配置 JAAS，則您必定會要撰寫類別來鑑別使用者，以取代標準的 JAAS 鑑別程序。

若要在遙測 (MQXR) 服務類別路徑所使用的類別路徑中包含您的 Login 類別，請提供 WebSphere MQ `service.env` 配置檔。

在 `service.env` 中設定您 JAAS LoginModule 的類別路徑。不能在 `service.env` 中使用變數 `%classpath%`。`service.env` 中的類別路徑會新增至遙測 (MQXR) 服務定義中已設定的類別路徑。

透過將 `echo set classpath` 新增至 `runMQXRService.bat`，可顯示遙測 (MQXR) 服務所使用的類別路徑。會將輸出傳送至 `mqxr.stdout`。

`service.env` 檔的預設位置是：

```
WMQ data directory\service.env
```

利用下列位置中每個佇列管理程式的 `service.env` 檔置換這些設定：

```
WMQ data directory\Qmgrs\qMgrName\service.env
```

第 146 頁的圖 47 顯示範例 `service.env` 檔案，以使用範例 `LoginModule.class`。

```
CLASSPATH=WMQ Install Directory\mqxr\samples
```

註：`service.env` 不得包含任何變數。請替換 `WMQ Install Directory` 的實際值。

圖 47: `service.env` for Windows 範例

追蹤

IBM 服務中心工程師可能會要求您配置追蹤；請參閱第 147 頁的『[追蹤遙測 \(MQXR\) 服務](#)』。所配置追蹤的參數儲存在兩個檔案中：

```
WMQ data directory\Qmgrs\qMgrName\mqxr\trace.config  
WMQ data directory\Qmgrs\qMgrName\mqxr\mqxrtrace.properties
```

用戶端日誌檔

IBM WebSphere MQ Telemetry 隨附的 Java SE MQTT 用戶端中的預設檔案持續性類別會建立名稱如下的資料夾：`clientIdentifier-tcphostName` 埠 或用戶端工作目錄中的 `clientIdentifier-sslhostname` 埠。資料夾名稱告知您在連線嘗試中使用的 `hostName` 和埠。資料夾包含持續性類別所儲存的訊息。在這些訊息遞送成功後，便會將其刪除。

具有未變動過的階段作業用戶端結束時，便會刪除資料夾。

如果用戶端追蹤已開啟，則依預設會將未格式化的日誌儲存在用戶端工作目錄中。追蹤檔案名為 `mqtt-n.trc`

用戶端配置檔

使用 Java 內容檔來設定 MQTT Java 用戶端的追蹤及 SSL 內容，或以程式化方式設定內容。使用 `JVM -D` 參數將內容傳遞至 MQTT Java 用戶端：例如，

```
Java -Dcom.ibm.micro.client.mqttv3.trace=c:\\MqttTrace.properties  
-Dcom.ibm.ssl.keyStore=C:\\MyKeyStore.jks
```

請參閱第 148 頁的『[追蹤 MQTT 第 3 版 Java 用戶端](#)』。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](#)。

MQTT v3 Java 用戶端原因碼

在 MQTT v3 Java 用戶端異常狀況或可擲出異常狀況中查閱原因碼的原因。

原因碼	值	原因
REASON_CODE_BROKER_UNAVAILABLE	3	
REASON_CODE_CLIENT_ALREADY_CONNECTED	32100	用戶端已連接。

表 5: MQTT v3 Java 用戶端原因碼 (繼續)		
原因碼	值	原因
REASON_CODE_CLIENT_ALREADY_DISCONNECTED	32101	用戶端已中斷連線。
REASON_CODE_CLIENT_DISCONNECT_PROHIBITED	32107	嘗試從 MqttCallback 上的方法中呼叫 MqttClient.disconnect 時擲出。
REASON_CODE_CLIENT_DISCONNECTING	32102	用戶端目前正在中斷連線，無法接受任何新工作。
REASON_CODE_CLIENT_EXCEPTION	0	用戶端發生異常狀況。
REASON_CODE_CLIENT_NOT_CONNECTED	32104	用戶端未連接至伺服器。
REASON_CODE_CLIENT_TIMEOUT	32000	用戶端在等待伺服器的回應時發生逾時。
REASON_CODE_FAILED_AUTHENTICATION	4	由於使用者名稱或密碼不正確，向伺服器鑑別失敗。
REASON_CODE_INVALID_CLIENT_ID	2	伺服器已拒絕所提供的用戶端 ID。
REASON_CODE_INVALID_PROTOCOL_VERSION	1	伺服器不支援所要求的通訊協定版本。
REASON_CODE_NO_MESSAGE_IDS_AVAILABLE	32001	內部錯誤，因為沒有新訊息 ID 可用。
REASON_CODE_NOT_AUTHORIZED	5	未獲授權執行所要求的作業。
REASON_CODE_SERVER_CONNECT_ERROR	32103	無法連接至伺服器，
REASON_CODE_SOCKET_FACTORY_MISMATCH	32105	伺服器 URI 與所提供的 SocketFactory 不符。
REASON_CODE_SSL_CONFIG_ERROR	32106	SSL 配置錯誤。
REASON_CODE_UNEXPECTED_ERROR	6	發生非預期的錯誤。

追蹤遙測 (MQXR) 服務

遵循下列指示，以啟動遙測服務追蹤、設定用於控制追蹤的參數及尋找追蹤輸出。

開始之前

追蹤是一個支援功能。如果 IBM 服務中心工程師要求您追蹤遙測 (MQXR) 服務，請遵循下列指示。產品說明文件不會說明追蹤檔案的格式，或者如何使用它來除錯用戶端。

關於這項作業

您可以使用 IBM WebSphere MQ **strmqtrc** 和 **endmqtrc** 指令來啟動和停止 IBM WebSphere MQ 追蹤。**strmqtrc** 可擷取遙測 (MQXR) 服務的追蹤。使用 **strmqtrc** 時，啟動遙測服務追蹤之前會出現幾秒的延遲。如需 IBM WebSphere MQ 追蹤的進一步資訊，請參閱[使用追蹤](#)。或者，您可以使用下列程序來追蹤遙測 (MQXR) 服務：

程序

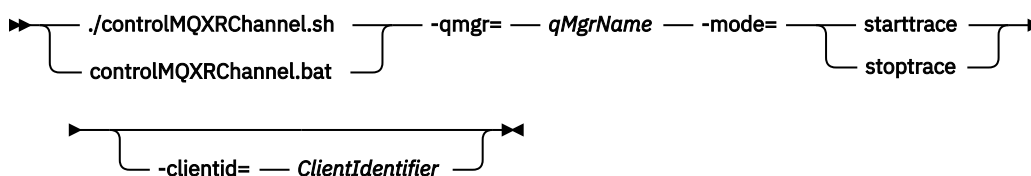
1. 設定追蹤選項，以控制詳細資料量及追蹤大小。這些選項適用於使用 **strmqtrc** 或 **controlMQXRChannel** 指令啟動的追蹤。

在下列檔案中設定追蹤選項：

```
mqxrtrace.properties  
trace.config
```

這些檔案位於下列目錄：

- 在 Windows 上: *WebSphere MQ data directory\qmgrs\qMgrName \mqxr*。
 - 在 Linux 上: *var/mqm/qmgrs/ qMgrName/mqxr*。
2. 在下列目錄中開啟指令視窗：
 - 在 Windows 系統上: *WebSphere MQ installation directory\mqxr\bin*。
 - 在 Linux 系統 */opt/mqm/mqxr/bin* 上。
 3. 執行下列指令，以啟動 SYSTEM.MQXR.SERVICE 追蹤：



必要的參數

qmgr=qmgrName

將 *qmgrName* 設為佇列管理程式名稱

mode=starttrace| stoptrace

設定 *starttrace* 以開始追蹤，或設定 *stoptrace* 以結束追蹤

選用參數

clientid=ClientIdentifier

將 *ClientIdentifier* 設為用戶端的 *ClientIdentifier*。 *clientid* 可將追蹤過濾為單一用戶端。執行追蹤指令多次可追蹤多個用戶端。

例如：

```
/opt/mqm/mqxr/bin/controlMQXRChannel.sh -qmgr=QM1 -mode=starttrace -clientid=  
problemclient
```

結果

若要檢視追蹤輸出，請跳至下列目錄：

- 在 Windows 上: *WebSphere MQ data directory\trace*。
- 在 Linux 上為 */var/mqm/trace*。

追蹤檔案名稱為 *mqxr_PPPPP.trc*，其中 *PPPPP* 為程序 ID。

相關參考

[strmqtrc](#)

追蹤 MQTT 第 3 版 Java 用戶端

遵循下列指示，以建立 MQTT Java 用戶端追蹤並控制其輸出。

開始之前

本主題僅適用於 IBM WebSphere MQ 7.5.0.0 版。如需說明追蹤更新版本的 Java 用戶端的資訊，請參閱第 151 頁的『[追蹤及除錯 MQTT \(Paho\) Java 用戶端](#)』。

追蹤是一個支援功能。如果 IBM 服務中心工程師要求您追蹤 MQTT Java 用戶端，請遵循下列指示。產品說明文件不會說明追蹤檔案的格式，或者如何使用它來除錯用戶端。

追蹤只能針對 WebSphere MQ Telemetry Java 用戶端進行。

關於這項作業

註：範例是針對 Windows 編寫的。變更語法以在 Linux 上執行範例²。

程序

1. 建立包含追蹤配置的 Java 內容檔。

在內容檔中，指定下列選用內容。如果多次指定某個內容索引鍵，則最後一個內容索引鍵會設定內容。

- a) `com.ibm.micro.client.mqttv3.trace.outputName`

要寫入追蹤檔案的目錄。預設為用戶端工作目錄。追蹤檔案名為 `mqtt-n.trc`。

```
java com.ibm.micro.client.mqttv3.trace.outputName=c:\\MQTT_Trace
```

- b) `com.ibm.micro.client.mqttv3.trace.count`

要寫入的追蹤檔案數目。預設值是一個檔案，無大小限制。

```
java com.ibm.micro.client.mqttv3.trace.count=5
```

- c) `com.ibm.micro.client.mqttv3.trace.limit`

要寫入的檔案大小上限，預設值是 500000。只有在要求多個追蹤檔案時，才會套用限制。

```
java com.ibm.micro.client.mqttv3.trace.limit=100000
```

- d) `com.ibm.micro.client.mqttv3.trace.client.clientIdentifier.status`

逐個用戶端開啟或關閉追蹤。如果指定 `clientIdentifier=*`，則會針對所有用戶端開啟或關閉追蹤。依預設，會針對所有用戶端關閉追蹤。

```
java com.ibm.micro.client.mqttv3.trace.client.*.status=on
```

```
java com.ibm.micro.client.mqttv3.trace.client.Client10.status=on
```

2. 使用系統內容將追蹤內容檔傳遞至 JVM。

```
java -Dcom.ibm.micro.client.mqttv3.trace=c:\\MqttTrace.properties
```

3. 執行用戶端。

4. 將追蹤檔案從二進位編碼轉換為文字或 .html。使用下列指令：

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter [-i traceFile] [-o  
outputFile] [-h] [-d  
time]
```

此處的引數為：

-?
顯示說明

² Java 使用正確的路徑定界字元。您可以將內容檔中的定界字元編碼為 '/' 或 '\\'; '\\' 是跳出字元

- i traceFile**
必要項目。傳入輸入檔 (例如, mqtt-0.trc)。
- o outputFile**
必要項目。定義輸出檔 (例如, mqtt-0.trc.html 或 mqtt-0.trc.txt)。
- h**
輸入為 HTML。輸出檔副檔名必須為 .html。如果未指定, 則輸出為純文字。
- d time**
如果時間差異 (毫秒) 大於或等於 (>=) 時間, 則以 * 縮排行。不適用於 HTML 輸出。

下列範例將以 HTML 格式輸出追蹤檔案

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.html -h
```

第二個範例將以純文字形式輸出追蹤檔案, 任何時間差異 (毫秒) 等於或超過 50 的連線時間戳記會使用星號 (*) 進行縮排。

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.txt -d 50
```

最後一個範例將以純文字形式輸出追蹤檔案:

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.txt
```

追蹤適用於 C 的 MQTT 用戶端

設定環境變數 MQTT_C_CLIENT_TRACE, 以追蹤 MQTT 用戶端 C 應用程式

開始之前

適用於 C 的 MQTT 用戶端追蹤可供以下項目使用: 預先建置的 Windows 和 Linux MQTT 用戶端程式庫 (適用於 C), 以及您自行建置的 iOS 程式庫。

關於這項作業

將環境變數 MQTT_C_CLIENT_TRACE 設為包含追蹤輸出之檔案的路徑。追蹤輸出會寫入該檔案。

程序

執行 MQTT 用戶端 C 應用程式之前, 請先設定 MQTT_C_CLIENT_TRACE=mqtccclient.log。

- a) 例如, 修改第 23 頁的『開始使用適用於 C 的 MQTT 用戶端』中的範例 Script:

```
@echo off
setlocal
set MQTT_C_CLIENT_TRACE=mqtccclient.log
call "C:\Program Files\Microsoft Visual Studio 10.0\VC\vcvarsall.bat" x86
cl /nologo /D "WIN32" /I "..\include" "MQTTV3Sample.c" /link /
nologo ..\windows_ia32\mqttv3c.lib
set path=%path%;..\windows_ia32;
start "MQTT Subscriber" MQTTV3Sample -a subscribe -b localhost -p 1883
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
MQTTV3Sample -b localhost -p 1883
pause
endlocal
```

- b) 從 %sdkroot%/sdk/client/c/samples 目錄執行 Script。

結果

追蹤輸出檔會以下列行開頭:

Trace Output

```
19700101 000000.000 (8084) (1)> Socket_outInitialize:113
19700101 000000.000 (8084) (2)> SocketBuffer_initialize:81
19700101 000000.000 (8084) (2)< SocketBuffer_initialize:85
19700101 000000.000 (8084) (1)< Socket_outInitialize:129
19700101 000000.000 (8084) (1)> Thread_create_sem:189
19700101 000000.000 (8084) (1)< Thread_create_sem:222 (0)
19700101 000000.000 (8084) (1)> Thread_create_sem:189
19700101 000000.000 (8084) (1)< Thread_create_sem:222 (0)
19700101 000000.000 (8084) (1)> Thread_create_sem:189
19700101 000000.000 (8084) (1)< Thread_create_sem:222 (0)
19700101 000000.000 (8084) (1)> Thread_create_sem:189
19700101 000000.000 (8084) (1)< Thread_create_sem:222 (0)
19700101 000000.000 (8084) (1)> MQTTPersistence_create:43
19700101 000000.000 (8084) (1)< MQTTPersistence_create:89 (0)
19700101 000000.000 (8084) (1)> MQTTPersistence_initialize:104
19700101 000000.000 (8084) (1)< MQTTPersistence_initialize:112 (0)
19700101 000000.000 (8084) (0)< MQTTClient_create:267 (0)
19700101 000000.000 (8084) (0)> MQTTClient_connect:701
19700101 000000.000 Connecting to serverURI localhost:1883
20130201 125912.234 (8084) (1)> MQTTProtocol_connect:93
20130201 125912.234 (8084) (2)> MQTTProtocol_addressPort:43
20130201 125912.234 (8084) (2)< MQTTProtocol_addressPort:68
20130201 125912.234 (8084) (2)> Socket_new:594
20130201 125912.234 New socket 1860 for localhost, port 1883
```

相關工作

第 23 頁的『開始使用適用於 C 的 MQTT 用戶端』

您可以在其中編譯 C 原始檔的任何平台上，建立並執行適用於 C 的 MQTT 用戶端範例。驗證您可以使用 IBM MessageSight 或 IBM WebSphere MQ 作為 MQTT 伺服器來執行適用於 C 的範例 MQTT 用戶端。

追蹤及除錯 MQTT (Paho) Java 用戶端

預設日誌程式使用標準 Java 記載機能 (稱為 `java.util.logging` (JSR47))。您可以透過使用配置檔或是以程式化方式來配置它。

關於這項作業

註: Paho Java 用戶端僅適用於 IBM WebSphere MQ 7.5.0.1 版以及更新版本。如需在 IBM WebSphere MQ 7.5.0.0 版中說明追蹤 Java 用戶端的資訊，請參閱 [第 148 頁的『追蹤 MQTT 第 3 版 Java 用戶端』](#)。

註: 追蹤是一個支援功能。如果 IBM 服務工程師要求您追蹤 MQTT Java 用戶端，請遵循下列指示。產品說明文件不會說明追蹤檔案的格式，或者如何使用它來除錯用戶端。追蹤僅適用於 IBM WebSphere MQ Telemetry Java 用戶端。

使用配置檔的最簡單方法是在 `java.util.logging.config.file` 內容中指定其名稱。

`org.eclipse.paho.client.mqttv3.logging` 套件中提供 `jsr47min.properties` 工作內容檔

您可以採用許多方式來使用 JSR47 記載機能:

- 從選取的一組套件中收集訊息
- 從記載層次及以下層次收集訊息
- 為日誌訊息選擇多個目的地
- 透過提供可寫入檔案並控制所用檔案大小及數目的內建日誌程式
- 透過提供可寫入記憶體並允許根據觸發程式寫出記憶體內部訊息的內建日誌程式

- 如果使用 MQTT 用戶端程式庫的應用程式也是透過使用 JSR47 進行監控，則會混合來自應用程式及用戶端程式庫的訊息

為了協助收集除錯資訊，專門提供了一個公用程式類別。此類別包括先前說明的日誌及追蹤訊息，但可以從 Paho 用戶端內部收集 Java 系統內容及變數值等資訊。

在屬於 `org.eclipse.paho.client.mqttv3.util` 套件一部分的 `Debug` 公用類別中，提供了除錯機能。對非同步及同步 MQTT 用戶端物件使用 `getDebug()` 方法，即可取得 `Debug` 實例。

例如：

```
MqttClient cl = new MqttClient();
Debug d = cl.getDebug();
```

`dumpClientDebug()` 方法會傾出最大數量的除錯資訊。必須啟用記載機能才能擷取寫入其中的完整除錯資訊。若要擷取完整除錯資訊，請在已知發生問題時（如在發生特定異常狀況之後）呼叫傾出方法。

程序

1. 建立配置檔或使用提供的 `jsr47min.properties`。

如果您是使用提供的內容檔，請檢查是否將推送觸發程式設定為正確的錯誤層次。依預設會將此錯誤層次設定為 `Severe` 層次的錯誤，但可能會需要將追蹤持續寫入檔案而非保留在記憶體中，直到發生錯誤為止。若要這樣做，請將：

```
java.util.logging.MemoryHandler.push=SEVERE
```

至

```
java.util.logging.MemoryHandler.push=ALL
```

2. 使用系統內容將追蹤配置檔傳遞至 JVM。

如果您是使用 `jsr4min.properties`，則此內容如下：

```
java -Djava.util.logging.config.file=C:\temp\jsr47min.properties
```

3. 執行用戶端。

結果

發生異常狀況或問題時，`Paho Debug` 類別會將記憶體內部追蹤寫入已配置的檔案目標。

產生追蹤時並不會將追蹤自動寫入檔案，只有在觸發推送觸發程式或除錯類型導致寫入追蹤時，才會發生此情況。後者可能需要變更應用程式碼。

每行在建立時皆會寫入檔案處理程式。您可以透過配置 `FileHandler` 來控制用於寫出訊息的格式。`Paho` 隨附的自訂檔案處理程式所寫出的內容，將會多於 `SimpleHandler` 但少於 `JRE` 隨附的 `XMLHandler`。使用 `Paho` 日誌格式製作程式的追蹤記錄具有下列格式：

```
Level    Data and Time    Class    Method    Thread    clientID    Message
```

範例

已提供 `jsr47min.properties` 工作內容檔。此檔案包含用於收集追蹤（可協助解決與 `Paho MQTT` 用戶端相關的問題）的建議配置。它會將追蹤配置為在記憶體內持續進行收集，並對效能產生最小影響。當推送觸發程式觸發時或發出特定推送要求時，即會將記憶體內部追蹤推送至已配置的目標處理程式。預設推送觸發程式是 `Severe` 層次訊息（已中斷的連線）。依預設，此時會將在記憶體內收集的追蹤寫入指定檔案。依預設，此檔案為標準 `java.util.logging.FileHandler`。您可以使用 `Paho Debug` 類別，將記憶體追蹤推送至其目標。

在 `java.util.logging` 套件的 Javadoc 中，可以找到 JSR47 的完整詳細資料。

```
# Loggers
# -----
```



```

# A memory handler is attached to the Paho packages
# and the level specified to collect all trace related
# to Paho packages. This will override any root/global
# level handlers if set.
org.eclipse.paho.client.mqttv3.handlers=java.util.logging.MemoryHandler
org.eclipse.paho.client.mqttv3.level=ALL
# It is possible to set more granular trace on a per class basis e.g.
#org.eclipse.paho.client.mqttv3.internal.ClientComms.level=ALL

# Handlers
# -----
# Note: the target handler that is associated with the Memory Handler is not a root handler
# and hence not returned when getting the handlers from root. It appears accessing
# target handler programmatically is not possible as target is a private variable in
# class MemoryHandler
java.util.logging.MemoryHandler.level=FINEST
java.util.logging.MemoryHandler.size=10000
java.util.logging.MemoryHandler.push=SEVERE
java.util.logging.MemoryHandler.target=java.util.logging.FileHandler
#java.util.logging.MemoryHandler.target=java.util.logging.ConsoleHandler

# --- FileHandler ---
# Override of global logging level
java.util.logging.FileHandler.level=ALL

# Naming style for the output file:
# (The output file is placed in the directory
# defined by the "user.home" System property.)
# See java.util.logging for more options
java.util.logging.FileHandler.pattern=%h/paho%u.log

# Limiting size of output file in bytes:
java.util.logging.FileHandler.limit=200000

# Number of output files to cycle through, by appending an
# integer to the base file name:
java.util.logging.FileHandler.count=3

# Style of output (Simple or XML):
java.util.logging.FileHandler.formatter=org.eclipse.paho.client.mqttv3.logging.SimpleLogFormate
r

```

下一步

為了以程式化方式收集追蹤，專門提供了一個公用程式類別來協助收集除錯資訊。此類別包括先前說明的日誌及追蹤訊息，但可以從 Paho 用戶端內部收集 Java 系統內容及變數值等資訊。

在屬於 `org.eclipse.paho.client.mqttv3.util` 套件一部分的 `Debug` 公用類別中，提供了除錯機能。對非同步及同步 MQTT 用戶端物件使用 `getDebug()` 方法，即可取得 `Debug` 實例。

例如：

```

MqttClient cl = new MqttClient();
Debug d = cl.getDebug();

```

`dumpClientDebug()` 方法會傾出最大數量的除錯資訊。必須啟用記載機能才能擷取寫入其中的完整除錯資訊。若要擷取完整除錯資訊，請在已知發生問題時（如在發生特定異常狀況之後）呼叫傾出方法。

追蹤 MQTT JavaScript 用戶端

透過變更用於對所連接用戶端物件呼叫方法的用戶端 Web 應用程式，即可使用 JavaScript 用戶端來收集追蹤。

關於這項作業

若要收集追蹤，您可以使用下列方法：

- `client.startTrace()` 會啟動用戶端追蹤。
- `client.stopTrace()` 會停止用戶端追蹤。
- `client.getTraceLog()` 會傳回現行追蹤緩衝區。

您可以輸出追蹤緩衝區，以便傳送至「IBM 軟體支援中心」。有許多方式可以執行此作業。下列範例先顯示要啟動的追蹤，再顯示傳送至主控台及指定電子郵件位址的輸出，最後顯示要停止的追蹤。

```
client = new Messaging.Client(location.hostname, Number(location.port), "clientId");
// Start the client tracing, the trace records capture the method calls and network
//flows from now on.
client.startTrace();

client.onConnectionLost = onConnectionLost;
client.connect({onSuccess:onConnect});

function onConnect() {
    console.log("onConnect, will now disconnect then email Trace");
    client.disconnect();
};
function onConnectionLost(responseObject) {
    if (responseObject.errorCode !== 0)
        console.log("onConnectionLost:"+responseObject.errorMessage);
    console.log(client.getTraceLog());
    window.location="mailto:helpdesk@"+location.hostname+
        "?Subject=Web%20Messaging%20Utility%20Trace&body="+
        client.getTraceLog().join("%0A");
    client.stopTrace();
};
```

範例輸出：

```
Client.startTrace, "2013-10-03T10:58:10.531Z", "0.0.0.0",
Client.connect, {"keepAliveInterval":60,"cleanSession":true},, false,
Client._socket_send, {"type":1,"keepAliveInterval":60,"cleanSession":true,
    "clientId":"clientId"},
Client._on_socket_message, {},
Client._on_socket_message, {"type":2,"topicNameCompressionResponse":0,"returnCode":0},
Client.disconnect,Client._socket_send, {"type":14},
Client.getTraceLog, "2013-10-03T10:58:10.548Z",
Client.getTraceLog in flight messages,
```

V7.5.0.2 對 MQTT 用戶端使用 SHA-2 密碼組合的系統需求

對於從 IBM SR13 開始的 Java 6，您可以使用 SHA-2 密碼組合來保護 MQTT 通道及用戶端應用程式的安全。不過，在從 IBM SR4 開始的 Java 7 之前，依預設不會啟用 SHA-2 密碼組合，因此在舊版中，您必須指定所需的組合。若您是搭配自己的 JRE 執行 MQTT 用戶端，必須確認其支援 SHA-2 密碼組合。若要讓用戶端應用程式使用 SHA-2 密碼組合，用戶端還必須將 SSL 環境定義設定為支援「傳輸層安全 (TLS)」1.2 版的值。

對於從 IBM SR4 開始的 Java 7，依預設會啟用 SHA-2 密碼組合。對於 IBM 中的 Java 6，SR13 以及更新版本的服務版本，如果您定義 MQTT 通道而未指定密碼組合，則通道將不會接受來自使用 SHA-2 密碼組合的用戶端連線。若要使用 SHA-2 密碼組合，您必須在通道定義中指定所需的組合。這會使 MQTT 伺服器在建立連線之前先啟用該組合。它還表示只有使用指定組合的用戶端應用程式才能連接至此通道。

適用於 Java 的 MQTT 用戶端存在類似限制。如果用戶端程式碼是在 IBM 中的 Java 1.6 JRE 上執行，則必須明確啟用必要的 SHA-2 密碼組合。若要使用這些組合，用戶端還必須將 SSL 環境定義設為支援 1.2 版「傳輸層安全 (TLS)」通訊協定的值。例如：

```
MqttConnectOptions mqttConnectOptions = new MqttConnectOptions();
java.util.Properties sslClientProps = new java.util.Properties();
sslClientProps.setProperty("com.ibm.ssl.keyStore", sslKeys.clientKeyStore);
sslClientProps.setProperty("com.ibm.ssl.keyStorePassword", sslKeys.clientStorePassword);
sslClientProps.setProperty("com.ibm.ssl.trustStore", sslKeys.clientKeyStore);
sslClientProps.setProperty("com.ibm.ssl.trustStorePassword", sslKeys.clientStorePassword);
sslClientProps.setProperty("com.ibm.ssl.protocol", "TLSv1.2");
sslClientProps.setProperty("com.ibm.ssl.enabledCipherSuites",
    "SSL_RSA_WITH_AES_256_CBC_SHA256" );
mqttConnectOptions.setSSLProperties(sslClientProps);
```

截至 2013 年 6 月，Internet Explorer 10 是唯一使用適用於 JavaScript 的 MQTT 傳訊用戶端且同時支援 TLS 1.2 通訊協定的瀏覽器，因此如果您想要與 JavaScript 用戶端建立 SHA-2 連線，則它是唯一可以使用的瀏覽器。

如需目前支援的密碼組合清單，請參閱相關鏈結。

相關概念

第 96 頁的『使用 SSL 進行用戶端鑑別的 MQTT 用戶端配置』

若要使用 SSL 來鑑別 MQTT 用戶端，MQTT 用戶端需使用 SSL 連接至遙測通道。它必須指定一個 TCP 埠，該埠對應於配置為鑑別 SSL 用戶端的遙測通道。

第 98 頁的『使用 SSL 進行通道鑑別的 MQTT 用戶端配置』

若要使用 SSL 鑑別遙測通道，用戶端必須使用 SSL 連接至遙測通道。它必須指定一個埠，該埠對應於配置為使用 SSL 的遙測通道。配置必須包括受通行詞組保護的金鑰儲存庫，其中包含私密簽章的伺服器數位憑證。

V7.5.0.1 瀏覽器對於使用 SSL 之行動式傳訊 Web 應用程式的支援限制

這在不同瀏覽器與平台組合上會有所差異。瞭解這些差異對於您配置應用程式、憑證管理中心 (CA)，以及使用適用於 JavaScript 的 MQTT 傳訊用戶端經由 SSL 及 WebSockets 連線時所需的用戶端憑證會很有幫助。

使用 JavaScript 經由 SSL 進行行動式傳訊是很新的功能，因此在不同的瀏覽器與平台組合上，必須以些微不同的方式實作此功能，而且支援的範圍也不同並不奇怪。下表是目前各種瀏覽器 (Firefox、Chrome、Internet Explorer 及 Safari) 與平台 (Windows、Linux、Mac、iOS 及 Android) 組合能否運作的概觀。

瀏覽器	SSL 支援 (是/否)	SSL 可與任何 CA 搭配運作 (是/否)	相關資訊
Firefox 桌面。	SSL 匿名 - 是 SSL 非匿名 - 是	SSL 匿名 - 是 SSL 非匿名 - 是	新增 CA 及用戶端憑證至瀏覽器。 Firefox 會使用自有的憑證儲存庫。 若要匯入 CA 憑證，請按一下 工具 > 選項 > 進階 > 加密 > 檢視憑證 > 憑證管理中心 > 匯入 若要匯入用戶端憑證，請按一下 工具 > 選項 > 進階 > 加密 > 檢視憑證 > 您的憑證 > 匯入 若要啟用安全連線，請在 URL 中指定 https:// 。Firefox 會提供選項，讓您選擇自動選取憑證或要每次詢問。Firefox 也會提供選項，讓您選擇使用 SSL 3.0 或 TLS 1.0。請務必同時確認這兩個選項。

表 6: 平台與瀏覽器的 SSL 支援. 此表格會指出每種瀏覽器與平台組合是否支援 SSL 匿名及非匿名連線, 以及瀏覽器與所有憑證管理中心 (CA) 及用戶端憑證搭配運作時所能支援的範圍。(繼續)

瀏覽器	SSL 支援 (是/否)	SSL 可與任可 CA 搭配運作 (是/否)	相關資訊
Chrome 桌面。	SSL 匿名 - 是 SSL 非匿名 - 是	SSL 匿名 - 是 SSL 非匿名 - 是	<p>使用瀏覽器新增 CA 及用戶端憑證至作業系統憑證儲存庫, 該儲存庫會與其他軟體共用。</p> <p>若要匯入 CA 憑證, 請按一下 設定 > 顯示進階設定 > 管理憑證 > 授信主要憑證管理中心 > 匯入</p> <p>若要匯入用戶端憑證, 請按一下 設定 > 顯示進階設定 > 管理憑證 > 個人 > 匯入</p> <p>若要啟用安全連線, 請在 URL 中指定 https://。Chrome 會提供數個選項; 請根據您要配置匿名或非匿名連線來選取正確的選項。</p>
Internet Explorer.	SSL 匿名 - 是 SSL 非匿名 - 是	SSL 匿名 - 是 SSL 非匿名 - 是	<p>當您進行非匿名 SSL 連線時, 會提示您選擇正確的用戶端憑證。</p> <p>Internet Explorer 會使用和其他軟體共用的 Windows 憑證儲存庫。</p> <p>若要匯入 CA 憑證, 請按一下 工具 > 網際網路選項 > 內容 > 憑證 > 授信主要憑證管理中心 > 匯入</p> <p>若要匯入用戶端憑證, 請按一下 工具 > 網際網路選項 > 內容 > 憑證 > 個人 > 匯入</p>
Safari 桌面。	SSL 匿名 - 是 SSL 非匿名 - 是	SSL 匿名 - 是 SSL 非匿名 - 是	<p>使用瀏覽器新增 CA 及用戶端憑證至作業系統憑證儲存庫, 該儲存庫會與其他軟體共用。</p>

表 6: 平台與瀏覽器的 SSL 支援. 此表格會指出每種瀏覽器與平台組合是否支援 SSL 匿名及非匿名連線, 以及瀏覽器與所有憑證管理中心 (CA) 及用戶端憑證搭配運作時所能支援的範圍。(繼續)

瀏覽器	SSL 支援 (是/否)	SSL 可與任何 CA 搭配運作 (是/否)	相關資訊
Firefox on Android	SSL 匿名 - 是 SSL 非匿名 - 是	SSL 匿名 - 是 SSL 非匿名 - 否	非匿名: 用戶端憑證無效, 因為您不符合將 CA 新增至 Firefox 中之清單的需求。 若要匯入用戶端憑證, 請按一下 設定 > 安全 > 認證儲存體 。您的憑證若是由清單中信任的 CA 簽署, 便可執行安全連線。
Chrome on Android	SSL 匿名 - 是 SSL 非匿名 - 是	SSL 匿名 - 是 SSL 非匿名 - 否	非匿名: 用戶端憑證無效, 因為您不符合將 CA 新增至 Chrome 中之清單的需求。 註: Google 預計會在第 27 版的 Chrome 中支援此功能。該問題自第 18 版開始即一直未獲解決。 若要匯入用戶端憑證, 請按一下 設定 > 安全 > 認證儲存體 。您的憑證若是由清單中信任的 CA 簽署, 便可執行安全連線。
Safari on iOS	SSL 匿名 - 是 SSL 非匿名 - 是	SSL 匿名 - 是 SSL 非匿名 - 否	非匿名: 即使是同時安裝此 CA 憑證, 裝置也不會信任此用戶端憑證。 Safari 會使用裝置憑證儲存庫。若要匯入至此儲存庫, 請按一下 設定 > 一般 > 設定檔 , 然後從網頁提供 CA 或用戶端憑證, 或透過電子郵件將它傳送給您自己。

表 6: 平台與瀏覽器的 SSL 支援. 此表格會指出每種瀏覽器與平台組合是否支援 SSL 匿名及非匿名連線, 以及瀏覽器與所有憑證管理中心 (CA) 及用戶端憑證搭配運作時所能支援的範圍。(繼續)

瀏覽器	SSL 支援 (是/否)	SSL 可與任何 CA 搭配運作 (是/否)	相關資訊
Chrome on iOS	SSL 匿名 - 是 SSL 非匿名 - 否	SSL 匿名 - 否 SSL 非匿名 - 否	匿名: 只有 Apple 應用程式才可存取 iOS 系統根儲存庫。因此, Chrome 必須使用自有的 CA 清單, 而您無法新增項目至該清單。 非匿名: 用戶端憑證無效, 因為您不符合將 CA 新增至清單的需求。

相關工作

第 71 頁的『[透過 SSL 及 WebSockets 連接 適用於 JavaScript 的 MQTT 傳訊用戶端](#)』

使用適用於 JavaScript 的 MQTT 傳訊用戶端 範例 HTML 頁面搭配 SSL 及 WebSocket protocol, 將 Web 應用程式安全地連接至 IBM WebSphere MQ。

相關資訊

Mozilla: (SSL) Firefox 使用 Android CA 儲存庫或其自有的儲存庫?

Chromium: [問題 134418 - 實作用戶端憑證支援](#)

[無法在 ie10 上開啟包含未受信任之憑證的 https 網站](#)

解決問題：MQTT 用戶端未連接

解決 MQTT 用戶端程式無法連接至遙測 (MQXR) 服務的問題。

開始之前

問題是出在伺服器上、用戶端還是連線? 您是否已使用 C 或 Java WebSphere MQTT 用戶端撰寫自己的 MQTT v3 通訊協定處理用戶端或 MQTT 用戶端應用程式?

在伺服器上執行 WebSphere MQ Telemetry 隨附的驗證應用程式, 檢查遙測通道及遙測 (MQXR) 服務是否正確執行。隨後將驗證應用程式傳送至用戶端, 並在用戶端上執行驗證應用程式。

關於這項作業

有許多原因會使 MQTT 用戶端未連接至 Telemetry 伺服器, 或者使您判斷它未連接至 Telemetry 伺服器。

程序

1. 考量可從遙測 (MQXR) 服務傳回至 MqttClient.Connect 的原因碼推出的結論。連線失敗的類型為何?

選項	敘述
REASON_CODE_INVALID_PROTOCOL_VERSION	請確保 Socket 位址對應於遙測通道, 而且您未將相同的 Socket 位址用於另一個分配管理系統。
REASON_CODE_INVALID_CLIENT_ID	請檢查用戶端 ID 不超過 23 個位元組, 且只包含下列範圍內的字元: A-Z, a-z, 0-9, '._/%
REASON_CODE_INVALID_DESTINATION	檢查用戶端 ID 與佇列管理程式名稱是否不同。

選項	敘述
REASON_CODE_SERVER_CONNECT_ERROR	檢查遙測 (MQXR) 服務及佇列管理程式是否正常執行。使用 netstat 來檢查 Socket 位址是否未配置給另一個應用程式。

如果您已撰寫 MQTT 用戶端程式庫，而不是使用 IBM WebSphere MQ Telemetry 提供的其中一個程式庫，請查看 CONNACK 回覆碼。

從這三個錯誤中，您可以推斷用戶端已連接至遙測 (MQXR) 服務，但該服務已發現錯誤。

2. 考量可從遙測 (MQXR) 服務未回應時用戶端所產生的原因碼推出的結論：

選項	敘述
REASON_CODE_CLIENT_EXCEPTION REASON_CODE_CLIENT_TIMEOUT	在伺服器上尋找 FDC 檔；請參閱第 144 頁的『伺服器端日誌』。當遙測 (MQXR) 服務偵測到用戶端逾時時，它會寫入首次失敗資料擷取 (FDC) 檔案。無論連線何時非預期地岔斷，它都會寫入 FDC 檔。

遙測 (MQXR) 服務可能未回應用戶端，並且用戶端逾時到期。只有在應用程式設定無限期逾時時時，WebSphere MQ Telemetry Java 用戶端才會當掉。在為 `MqttClient.Connect` 設定的逾時到期之後，如果發生無法診斷的問題，則用戶端會擲出這三個異常狀況之一。

除非您找到與連線失敗關聯的 FDC 檔，否則您無法推斷出該用戶端嘗試連接至伺服器：

- a) 確認用戶端已傳送連線要求。

利用 **tcpmon** (可從 <https://java.net/projects/tcpmon> 取得) 等工具，檢查 TCP/IP 要求。

- b) 用戶端使用的遠端 Socket 位址與定義給遙測通道的 Socket 位址是否相符？

IBM WebSphere MQ Telemetry 隨附的 Java SE MQTT 用戶端中的預設檔案持續性類別會建立名稱如下的資料夾：`clientIdentifier-tcphostName` 埠 或用戶端工作目錄中的 `clientIdentifier-sslhostname` 埠。資料夾名稱告知您在連線嘗試中使用的 `hostname` 和埠；請參閱第 146 頁的『用戶端日誌檔』。

- c) 您可以 ping 遠端伺服器位址嗎？

- d) 在伺服器上執行 **netstat**，是否顯示遙測通道正在用戶端所連接的埠上執行？

3. 檢查遙測 (MQXR) 服務是否在用戶端要求中發現問題。

遙測 (MQXR) 服務會將其偵測到的錯誤寫入 `mqxr.log`，而佇列管理程式會將錯誤寫入 `AMQERR01.LOG`；請參閱

4. 嘗試透過執行另一個用戶端隔離問題。

- 使用相同的遙測通道執行 MQTT 範例應用程式。
- 執行 **wmqttsample** GUI 用戶端，以驗證連線。下載 [SupportPac IA92](#) 以取得 **wmqttsample**。

註：舊版 IA92 不包含 MQTT v3 Java 用戶端程式庫。

在伺服器平台上執行程式範例以排除網路連線相關的不確定問題，然後在用戶端平台上執行範例。

5. 要檢查的其他內容：

- a) 是否有成千上萬的 MQTT 用戶端同時嘗試連接？

遙測通道具有佇列，可以緩存送入連線待辦事項。每秒處理的連線數超過 10,000。待辦事項緩衝區的大小，可以使用「IBM WebSphere MQ 探險家」中的遙測通道精靈進行配置。其預設大小為 4096。檢查待辦事項是否配置為較低的值。

- b) 遙測 (MQXR) 服務及佇列管理程式仍在執行中嗎？

- c) 用戶端是否已連接至已切換其 TCP/IP 位址的高可用性佇列管理程式？

- d) 防火牆是否選擇性地過濾掉出埠或傳回資料封包？

解決問題：MQTT 用戶端連線中斷

找出導致用戶端在順利連接並執行或長或短的一段時間之後，擲出非預期的 `ConnectionLost` 異常狀況的原因。

開始之前

MQTT 用戶端已順利連接。用戶端可能會執行一長段時間。如果用戶端啟動間隔很短，則順利連接與連接中斷之間的時間可能很短。

區分中斷的連線與成功建立、隨後中斷的連線並不困難。中斷的連線由 MQTT 用戶端呼叫 `MqttCallback.ConnectionLost` 方法來定義。該方法僅在順利建立連線之後呼叫。該症狀與 `MqttClient.Connect` 在收到負面確認通知或逾時之後擲出異常狀況不同。

如果 MQTT 用戶端應用程式不是使用由 IBM WebSphere MQ 提供的 MQTT 用戶端程式庫，則症狀取決於用戶端。在 MQTT v3 通訊協定中，症狀是伺服器要求的回應不及時，或者 TCP/IP 連線失敗。

關於這項作業

MQTT 用戶端利用可擲出異常狀況呼叫 `MqttCallback.ConnectionLost`，以回應在收到正面連線確認通知之後發生的伺服器端問題。MQTT 用戶端從 `MqttTopic.publish` 和 `MqttClient.subscribe` 返回時，會將該要求傳送至負責傳送和接收訊息的 MQTT 用戶端執行緒。會透過將可擲出異常狀況傳遞至 `ConnectionLost` 回呼方法，來以非同步方式報告伺服器端錯誤。

遙測 (MQXR) 服務一律寫入首次失敗資料擷取檔案（如果它中斷連線的話）。

程序

1. 是否已啟動使用相同 `ClientIdentifier` 的另一個用戶端？

如果已啟動第二個用戶端或已重新啟動相同的用戶端（使用相同的 `ClientIdentifier`），則會中斷與第一個用戶端的第一個連線。

2. 用戶端是否存取了未獲授權來發佈至或訂閱的主題？

Telemetry 服務代表用戶端執行的所有動作，它們傳回 `MQCC_FAIL` 導致服務中斷用戶端連線連線。不會將原因碼傳回至用戶端。

- 在用戶端所連接佇列管理程式的 `mqxr.log` 及 `AMQERR01.LOG` 檔案中尋找日誌訊息；請參閱 [第 144 頁的『伺服器端日誌』](#)。

3. TCP/IP 連線是否已中斷？

防火牆可能設定較低的逾時以將 TCP/IP 連線標示為非作用中，從而中斷連線。

- 使用 `MqttConnectOptions.setKeepAliveInterval` 縮短非作用中的 TCP/IP 連線時間。

解決問題：MQTT 應用程式中遺失訊息

解決遺失訊息問題。訊息是非持續性的、傳送至錯誤的位置還是從未傳送？編寫不正確的用戶端程式可能會遺失訊息。

開始之前

您是否確定所傳送訊息已遺失？您可以因未收到訊息而推斷它已遺失嗎？如果訊息是個發佈，遺失哪一種訊息：發佈者傳送的訊息還是傳送至訂閱者的訊息？或者是訂閱遺失，還是分配管理系統未將該訂閱的發佈傳送至訂閱者？

如果解決方案涉及使用叢集或發佈/訂閱階層的分散式發佈/訂閱，則有許多配置問題可能會導致訊息遺失的現象。

如果利用「至少一次」或「至多一次」服務品質傳送訊息，則您認為已遺失的訊息可能未以您預期的方式遞送。不太可能已從系統誤刪該訊息。系統可能無法建立您預期的發佈或訂閱。

在進行遺失訊息問題判斷過程中，您執行的最重要步驟是確認訊息已遺失。重建該狀況，遺失更多訊息。使用「至少一次」或「至少一次」服務品質以排除系統捨棄訊息的所有情況。

關於這項作業

診斷遺失的訊息有四個要素。

1. 「隨發即忘」訊息依設計運作。系統有時會捨棄「隨發即忘」訊息。
2. 配置：並非以直接明確的方式，在分散式環境中使用正確的權限來設定發佈/訂閱。
3. 用戶端程式設計錯誤：負責訊息遞送並不僅僅是 IBM 所撰寫的程式碼責任。
4. 如果已排除了所有這些可能性，則您可能會決定諮詢 IBM 服務中心。

程序

1. 如果遺失訊息的服務品質為「隨發即忘」，請設定「至少一次」或「至多一次」服務品質。重試遺失該訊息。
 - 在多種情況下，IBM WebSphere MQ 會捨棄利用「隨發即忘」服務品質傳送的訊息：
 - 通訊中斷和通道已停止。
 - 佇列管理程式關閉。
 - 訊息數過多。
 - 「隨發即忘」訊息的遞送依賴於 TCP/IP 的可靠性。TCP/IP 會不斷重新傳送資料封包，直到其遞送得到確認為止。如果 TCP/IP 階段作業被岔斷，服務品質為「隨發即忘」的訊息便會遺失。階段作業被岔斷的可能原因有：用戶端或伺服器關閉、發生通訊問題或防火牆使階段作業中斷連線。
2. 檢查用戶端是否正在重新啟動前一個階段作業，以便利用「至少一次」或「至多一次」服務品質再次傳送未遞送的訊息。
 - a) 如果用戶端應用程式使用 Java SE MQTT 用戶端，請檢查它是否將 `MqttClient.CleanSession` 設為 `false`
 - b) 如果您使用的是不同的用戶端程式庫，請檢查是否正確地重新啟動階段作業。
3. 檢查用戶端應用程式是否重新啟動同一個階段作業，而非不小心啟動其他階段作業。

若要再次啟動相同的階段作業，請設定 `cleanSession = false`，而且 `Mqttclient.clientIdentifier` 和 `MqttClient.serverURI` 必須與前一個階段作業相同。
4. 如果階段作業已永久關閉，請檢查用戶端持續性儲存庫中可用的訊息，以再次傳送。
 - a) 如果用戶端應用程式使用 Java SE MQTT 用戶端，請檢查訊息是否儲存在持續性資料夾中；請參閱 [第 146 頁的『用戶端日誌檔』](#)
 - b) 如果您使用的是不同的用戶端程式庫，或者您已經實作自己的持續性機制，請檢查它是否正確地運作。
5. 檢查訊息在遞送之前沒有被刪除。

等待遞送至 MQTT 用戶端的未遞送訊息，儲存於 `SYSTEM.MQTT.TRANSMIT.QUEUE`。等待遞送至遙測伺服器的訊息採用用戶端持續性機制進行儲存；請參閱 [MQTT 用戶端中的訊息持續性](#)。
6. 檢查用戶端是否有訂閱預期要接收的發佈。

使用「WebSphere MQ 探險家」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`
7. 檢查發佈者是否具有發佈的權限，以及訂閱者是否具有訂閱發佈主題的權限。

```
dspmqaut -m qMgr -n topicName -t topic -p user ID
```

在叢集化發佈/訂閱系統中，必須針對訂閱者所連接佇列管理程式上的主題，為訂閱者授與權限。不必針對訂閱發佈所在佇列管理程式上的主題，為訂閱者授與權限。必須正確授與佇列管理程式之間通道的權限，才能傳遞 Proxy 訂閱和轉遞發佈。

使用「IBM WebSphere MQ 探險家」建立相同的訂閱並發佈至其中。使用用戶端公用程式，模擬應用程式用戶端的發佈和訂閱功能。從「IBM WebSphere MQ 探險家」啟動該公用程式，並變更其使用者 ID 以符合您的用戶端應用程式所採用的 ID。

8. 檢查訂閱者是否具有可將發佈放置在 `SYSTEM.MQTT.TRANSMIT.QUEUE` 上的權限。

```
dspmqaout -m qMgr -n queueName -t queue -p user ID
```

9. 檢查 IBM WebSphere MQ 點對點應用程式是否有權將其訊息放置在 `SYSTEM.MQTT.TRANSMIT.QUEUE` 上。

```
dspmqaout -m qMgr -n queueName -t queue -p user ID
```

請參閱 [配置分散式佇列以將訊息傳送至 MQTT 用戶端中的 "直接傳送訊息至用戶端"](#)。

解決問題：遙測 (MQXR) 服務未啟動

解決遙測 (MQXR) 服務無法啟動的問題。檢查 WebSphere MQ Telemetry 安裝，確定沒有任何檔案遺漏、被移動或具有錯誤的許可權。檢查遙測 (MQXR) 服務所使用的路徑是否可以找到遙測 (MQXR) 服務程式。

開始之前

已安裝 WebSphere MQ Telemetry 特性。「IBM WebSphere MQ 探險家」在 **IBM WebSphere MQ > 佇列管理程式 > qMgr 名稱 > 遙測** 中具有 Telemetry 資料夾。如果該資料夾不存在，則安裝失敗。

遙測 (MQXR) 服務必須已建立，其才能啟動。如果尚未建立遙測 (MQXR) 服務，請執行 [定義配置範例 ... Telemetry](#) 資料夾中的精靈。

如果先前曾啟動過遙測 (MQXR) 服務，則會在 Telemetry 資料夾下另外建立 **通道及通道狀態** 資料夾。Telemetry 服務 `SYSTEM.MQXR.SERVICE` 位於 **Services** 資料夾。如果已勾選用於顯示「系統物件」的「探險家」圓鈕，則可以看到它。

用滑鼠右鍵按一下 `SYSTEM.MQXR.SERVICE`，以啟動及停止服務，顯示其狀態，並顯示您的使用者 ID 是否有權啟動服務。

關於這項作業

`SYSTEM.MQXR.SERVICE` 遙測 (MQXR) 服務無法啟動。無法以兩種不同的方法啟動資訊清單本身：

1. `start` 指令立即失敗。
2. `start` 指令成功，但是服務隨即停止。

程序

1. 啟動服務

結果

服務立即停止。視窗顯示錯誤訊息；例如：

```
WebSphere MQ cannot process the request because the executable specified cannot be started. (AMQ4160)
```

原因

安裝中遺漏檔案，或者已安裝檔案的權限設定錯誤。

IBM WebSphere MQ Telemetry 功能僅安裝在一對高可用性佇列管理程式中的一個上。如果佇列管理程式實例切換至待用伺服器，則它會嘗試啟動 `SYSTEM.MQXR.SERVICE`。啟動服務的指令會因待用伺服器上未安裝遙測 (MQXR) 服務而失敗。

調查

查看錯誤日誌；請參閱第 144 頁的『[伺服器端日誌](#)』。

動作

安裝（或解除安裝然後重新安裝）WebSphere MQ Telemetry 特性。

2. 啟動服務；等待 30 秒；重新整理「探險家」並檢查服務狀態。

結果

服務啟動，然後停止。

原因

SYSTEM.MQXR.SERVICE 已啟動 `runMQXRService` 指令，但指令失敗。

調查

查看錯誤日誌；請參閱第 144 頁的『伺服器端日誌』。
瞭解是否僅定義的範例通道發生問題。備份並清除 `WMQ data directory\Qmgrs\qMgrName\mqxr\` 目錄的內容。執行配置範例精靈，嘗試啟動該服務。

動作

尋找權限和路徑問題。

解決問題：Telemetry 服務未呼叫 JAAS 登入模組

瞭解遙測 (MQXR) 服務是否未呼叫您的 JAAS 登入模組，並配置 JAAS 以更正問題。

開始之前

您已修改 `WMQ installation directory\mqxr\samples>LoginModule.java`，以建立您自己的鑑別類別 `WMQ installation directory\mqxr\samples\samples>LoginModule.class`。此外，您已撰寫自己適用的 JAAS 鑑別類別，並將它們放置在您選擇的目錄中。對遙測 (MQXR) 服務進行一些初步測試後，您懷疑遙測 (MQXR) 服務未呼叫您的鑑別類別。

註：請避免您的鑑別類別被 WebSphere MQ 所套用的維護改寫的可能性。請使用您自己的鑑別類別路徑，而非 WebSphere MQ 目錄樹中的路徑。

關於這項作業

作業使用實務範例來說明如何解決問題。在該實務範例中，名為 `security.jaas` 的套件包含名為 `JAASLogin.class` 的鑑別類別。該類別儲存在路徑 `C:\WMQTelemetryApps\security\jaas` 中。請參閱遙測通道 JAAS 配置，以取得有關針對 IBM WebSphere MQ Telemetry 配置 JAAS 的說明。範例第 163 頁的『範例 JAAS 配置』是一個配置範例。

程序

1. 查看 `mqxr.log`，以取得 `javax.security.auth.login.LoginException` 所擲出的異常狀況。
請參閱第 144 頁的『伺服器端日誌』以取得 `mqxr.log` 的路徑，並參閱第 165 頁的圖 54 取得日誌中列出的異常狀況範例。
2. 透過將 JAAS 配置與第 163 頁的『範例 JAAS 配置』中已運作的範例進行比較，更正 JAAS 配置。
3. 將範例 `JAASLoginModule` 重構到鑑別套件並使用相同的路徑進行部署之後，用其取代登入類別。將 `loggedIn` 的值在 `true` 和 `false` 之間切換。
如果 `loggedIn` 是 `true` 時問題消失，而 `loggedIn` 是 `false` 時問題出現，則登入類別有問題。
4. 檢查問題是否與授權相關，而不是與鑑別相關。
 - a) 變更遙測通道定義，以使用固定的使用者 ID 執行授權檢查。選取屬於 `mqm` 群組的使用者 ID。
 - b) 重新執行用戶端應用程式。

如果問題消失，則解決方案在於要傳遞以授權的使用者 ID。要傳遞的使用者名稱為何？請將其列印到登入模組中的檔案。使用「IBM WebSphere MQ 探險家」或 `dspmqaauth` 來檢查其存取權。

範例 JAAS 配置

使用「WebSphere MQ 探險家」中的新建遙測通道精靈，來配置遙測通道。用戶端在埠 1884 上進行連接，並連接至 `JAASMCUser` 遙測通道。第 164 頁的圖 48 顯示遙測精靈所建立的遙測內容檔範例。請勿直接

編輯這個檔案。通道會使用 JAAS 及名為 JAASConfig 的配置進行鑑別。一旦用戶端通過鑑別，它便會將使用者 ID Admin 用於授與其對 IBM WebSphere MQ 物件的存取權。

```
com.ibm.mq.MQXR.channel/JAASMCUser: \  
com.ibm.mq.MQXR.Port=1884;\  
com.ibm.mq.MQXR.JAASConfig=JAASConfig;\  
com.ibm.mq.MQXR.UserName=Admin;\  
com.ibm.mq.MQXR.StartWithMQXRService=true
```

圖 48: *WMQ Installation directory\data\mqgrs\qMgrName\mqxr\mqxr_win.properties*

JAAS 配置檔有一個名為 JAASConfig 的段落，名稱是 JAAS 用來鑑別用戶端的 Java 類別 security.jaas.JAASLogin。

```
JAASConfig {  
    security.jaas.JAASLogin required debug=true;  
};
```

圖 49: *WMQ Installation directory\data\mqgrs\qMgrName\mqxr\jaas.config*

SYSTEM.MQTT.SERVICE 啟動時，它會將第 164 頁的圖 50 中的路徑新增至其類別路徑。

```
CLASSPATH=C:\WMQTelemetryApps;
```

圖 50: *WMQ Installation directory\data\mqgrs\qMgrName\service.env*

第 164 頁的圖 51 顯示新增至為遙測 (MQXR) 服務所設定類別路徑的第 164 頁的圖 50 中的附加路徑。

```
CLASSPATH=;C:\IBM\MQ\Program\mqxr\bin\..\lib\MQXRListener.jar;  
C:\IBM\MQ\Program\mqxr\bin\..\lib\WMQCommonServices.jar;  
C:\IBM\MQ\Program\mqxr\bin\..\lib\objectManager.utils.jar;  
C:\IBM\MQ\Program\mqxr\bin\..\lib\com.ibm.micro.xr.jar;  
C:\IBM\MQ\Program\mqxr\bin\..\..\java\lib\com.ibm.mq.jmqi.jar;  
C:\IBM\MQ\Program\mqxr\bin\..\..\java\lib\com.ibm.mqjms.jar;  
C:\IBM\MQ\Program\mqxr\bin\..\..\java\lib\com.ibm.mq.jar;  
C:\WMQTelemetryApps;
```

圖 51: *runMQXRService.bat* 的類別路徑輸出

第 164 頁的圖 52 中的輸出顯示已使用第 164 頁的圖 48 中所示通道定義啟動的遙測 (MQXR) 服務。

```
21/05/2010 15:32:12 [main] com.ibm.mq.MQXRService.MQXRPropertiesFile  
AMQXR2011I: Property com.ibm.mq.MQXR.channel/JAASMCUser value  
com.ibm.mq.MQXR.Port=1884;  
com.ibm.mq.MQXR.JAASConfig=JAASConfig;  
com.ibm.mq.MQXR.UserName=Admin;  
com.ibm.mq.MQXR.StartWithMQXRService=true
```

圖 52: *WMQ Installation directory\data\mqgrs\qMgrName\errors\mqxr.log*

用戶端應用程式連接至 JAAS 通道時，如果 com.ibm.mq.MQXR.JAASConfig=JAASWrongConfig 與 jaas.config 檔案中的 JAAS 段落名稱不符，則連線失敗，並且用戶端會擲出回覆碼為 0 的異常狀況；請參閱第 165 頁的圖 53。擲出第二個異常狀況 Client is not connected (32104) 的原因，是用戶端在未連接的情況下，嘗試中斷連線。

```

C:\WMQTelemetryApps>java com.ibm.mq.id.PubAsyncRestartable
Starting a clean session for instance "Admin_PubAsyncRestartab"
Publishing "Hello World Fri May 21 17:23:23 BST 2010" on topic "MQTT Example"
for client instance: "Admin_PubAsyncRestartab" using QoS=1 on address tcp://localhost:1884"
Userid: "Admin", Password: "Password"
Delivery token "528752516" has been received: false
Connection lost on instance "Admin_PubAsyncRestartab" with cause "MqttException"
MqttException (0) - java.io.EOFException
    at com.ibm.micro.client.mqttv3.internal.CommsReceiver.run(CommsReceiver.java:118)
    at java.lang.Thread.run(Thread.java:801)
Caused by: java.io.EOFException
    at java.io.DataInputStream.readByte(DataInputStream.java:269)
    at
com.ibm.micro.client.mqttv3.internal.wire.MqttInputStream.readMqttWireMessage(MqttInputStream.java:56)
    at com.ibm.micro.client.mqttv3.internal.CommsReceiver.run(CommsReceiver.java:90)
    ... 1 more
Client is not connected (32104)
    at
com.ibm.micro.client.mqttv3.internal.ExceptionHelper.createMqttException(ExceptionHelper.java:33)
    at com.ibm.micro.client.mqttv3.internal.ClientComms.internalSend(ClientComms.java:100)
    at com.ibm.micro.client.mqttv3.internal.ClientComms.sendNowait(ClientComms.java:117)
    at com.ibm.micro.client.mqttv3.internal.ClientComms.disconnect(ClientComms.java:229)
    at com.ibm.micro.client.mqttv3.MqttClient.disconnect(MqttClient.java:385)
    at com.ibm.mq.id.PubAsyncRestartable.main(PubAsyncRestartable.java:49)

```

圖 53: 連接 `com.ibm.mq.id.PubAsyncRestartable` 時擲出異常狀況

`mqxr.log` 包含第 165 頁的圖 53 中所示的其他輸出。

錯誤由擲出 `javax.security.auth.login.LoginException` (原因為 `No LoginModules configured for JAAS`) 的 JAAS 偵測到。原因可能是配置名稱不正確，如第 165 頁的圖 54 所示。它也可能是在載入 JAAS 配置期間，JAAS 發生的其他問題引起的。

如果 JAAS 未報告異常狀況，則 JAAS 已順利載入在 `JAASConfig` 段落中指名的 `security.jaas.JAASLogin` 類別。

```

21/05/2010 12:06:12 [ServerWorker0] com.ibm.mq.MQXRService.MQTTCommunications
AMQXR2050E: Unable to load JAAS config: JAASWrongConfig.
The following exception occurred javax.security.auth.login.LoginException:
No LoginModules configured for JAAS

```

圖 54: `mqxr.log` - 載入 JAAS 配置時發生錯誤

解決問題：啟動或執行常駐程式

查閱適用於裝置的 IBM WebSphere MQ Telemetry 常駐程式主控台日誌、開啟追蹤或使用此主題中的症狀表格，來疑難排解常駐程式問題。

程序

1. 檢查主控台記錄。

如果常駐程式正在前景執行，則會將主控台訊息寫入終端機視窗。如果常駐程式已在背景啟動，則您會將 `stdout` 重新導向至主控台。

2. 重新啟動常駐程式。

重新啟動常駐程式之後，對配置檔的變更才會啟動。

3. 請參閱第 166 頁的表 7:

表 7: 症狀表格	
問題	建議的解決方案
在 Windows 上啟動常駐程式時顯示下列訊息： 系統無法執行指定的程式 or 因為應用程式的並列組態錯誤， 所以無法啟動。	安裝 Microsoft Visual C++ 2008 可轉散發套件。
兩個以上常駐程式或具有 MQTT 能力的伺服器透過一或多個橋接器處理器，處理器顯示負載過重。	可能有訊息迴圈，反覆地將一則以上的訊息從一個伺服器傳遞至另一個伺服器。請檢查配置檔中的主題參數。應儘量使用更特定的主題。兩個方向上的常用萬用字元是連線迴圈的最常見原因。
橋接器無法連接至其他 MQTT 用戶端可以連接至的具有 MQTT 功能的遠端伺服器。	遠端伺服器可能不支援嘗試判定遠端伺服器是否同時為適用於裝置的 WebSphere MQ Telemetry 常駐程式。嘗試將 try_private 設為 off 以停用特殊處理，以消除訊息迴圈。
配置橋接器時，會列印下列訊息： 警告：連接不是位於第一個封包，Socket 為 1888，已取得 CONNACK。	您可能已配置橋接器以迴圈至本端常駐程式。不支援迴圈。

解決問題：MQTT 用戶端未連接至常駐程式

用戶端未連接至常駐程式，或是常駐程式未連接至其他常駐程式或 WebSphere MQ Telemetry 通道。

關於這項作業

追蹤常駐程式傳送及接收的每一個封包。

程序

在常駐配置檔中，將 **trace_output** 參數設定為 `protocol`，或使用 `amqtdc.upd` 檔案將指令傳送至常駐程式。

如需使用 `amqtdc.upd` 檔案的範例，請參閱 [在用於裝置的 IBM WebSphere MQ Telemetry 常駐程式與 IBM WebSphere MQ 之間傳送訊息](#)。

透過使用 `protocol` 設定，常駐程式可將訊息列印至主控台，其中會說明該常駐程式傳送及接收的每個 MQTT 封包。

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。

在其他國家中，IBM 可能不會提供本書中所提的各項產品、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。這份文件在提及 IBM 的產品、程式或服務時，不表示或暗示只能使用 IBM 的產品、程式或服務。只要未侵犯 IBM 的智慧財產權，任何功能相當的產品、程式或服務都可以取代 IBM 的產品、程式或服務。不過，任何非 IBM 的產品、程式或服務，使用者必須自行負責作業的評估和驗證責任。

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：

智慧財產權授權
法務部與智慧財產權法律
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。

這項資訊中可能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。IBM 隨時會改進及/或變更本出版品所提及的產品及/或程式，不另行通知。

本資訊中任何對非 IBM 網站的敘述僅供參考，IBM 對該網站並不提供任何保證。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。因此不同作業環境之下所得的結果，可能會有很大的差異。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。本文件的使用者應驗證其特定環境適用的資料。

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 未來方針或目的之所有聲明，僅代表 IBM 的目標與主旨，隨時可能變更或撤銷，不必另行通知。

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。這些名稱全屬虛構，如與實際公司的名稱和住址雷同，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、有用性或功能。

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 IBM 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](http://www.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。其他產品和服務名稱，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 (<http://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: