7.5

移动消息传递和 *M2M*

注

在使用本资料及其支持的产品之前，请阅读第 165 页的『声明』中的信息。

此版本适用于 IBM® WebSphere MQ V 7 发行版 5 以及所有后续发行版和修订版，直到在新版本中另有声明为止。

当您向 IBM 发送信息时，授予 IBM 以它认为适当的任何方式使用或分发信息的非独占权利，而无需对您承担任何责任。

内容

MQTT 简介

了解有关使用 MQ 遥测传输在移动应用程序之间发送消息的信息 (MQTT)。 该协议旨在用于无线和低带宽网络。 此移动应用程序通过调用 MQTT 库，使用 MQTT 来发送和接收消息。 消息通过 MQTT 消息传递服务器进行交换。 MQTT 客户机和服务器可应对为移动应用程序可靠地传递消息所带来的复杂性，并且将网络管理成本保持在较低水平。

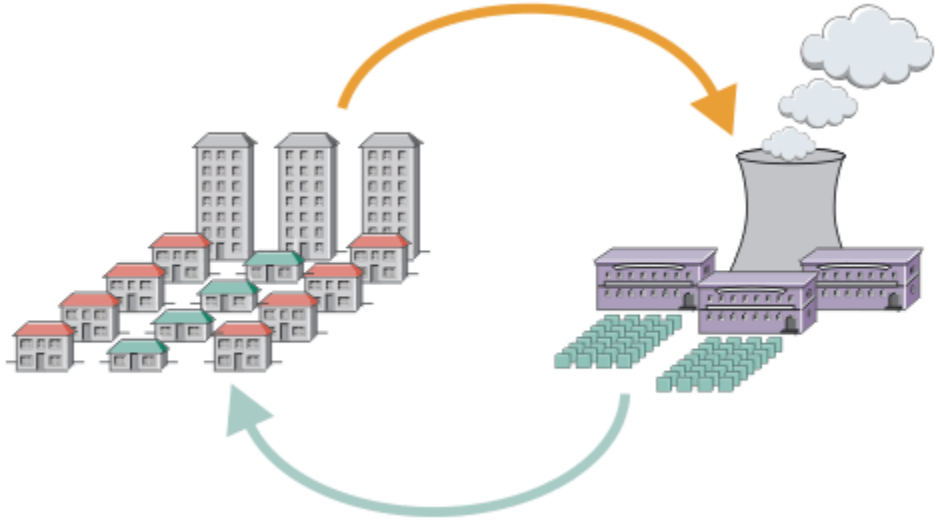
MQTT 应用程序在移动设备上运行，例如智能手机和平板电脑。 MQTT 还用于遥测，以接收来自传感器的数据并对传感器进行远程控制。 对于移动设备和传感器，MQTT 通过可靠的传送提供高度可扩展的发布/预订协议。 要发送和接收 MQTT 消息，请将 MQTT 客户机库添加到应用程序。

MQTT 客户机库很小。 该库充当邮箱功能，通过连接到 MQTT 服务器的其他 MQTT 应用程序发送和接收消息。 通过发送消息，而不是保持连接到等待响应的服务器，MQTT 应用程序延长了电池的使用时间。 该库通过运行 MQTT version 3.1 协议的 MQTT 服务器向其他设备发送消息。 您可以向特定的客户机发送消息，或者使用发布/预订消息传递来连接多个设备。

MQTT 客户机库使用 MQTT 协议将移动设备和传感器的应用程序连接到 MQTT 服务器。

IBM MessageSight 和 IBM WebSphere MQ 是 MQTT 服务器。 它们能连接大量的 MQTT 客户机应用程序，还能将 MQTT 和 IBM WebSphere MQ 网络连到一起。 请参阅 第 122 页的『MQTT 服务器入门』。 IBM WebSphere MQ 和 IBM MessageSight 都可以形成一个网桥，用于连接正在移动设备和传感器上运行的外部 Web 应用程序以及在企业内运行的其他类型的发布/预订和消息传递应用程序。 通过此网桥，可以更轻松地构建包含移动设备和传感器的 “智能解决方案” 。

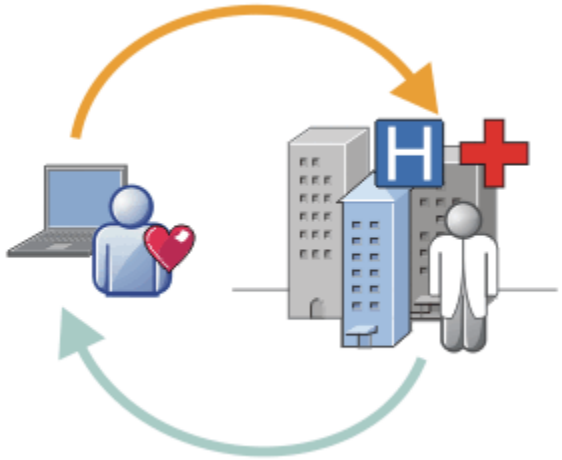
智慧解决方案将来自因特网的信息应用到移动设备和传感设备上运行的应用程序。 基于遥测的两个智慧应用程序示例为智慧电力和智慧健康状况服务。



- 一条 MQTT 消息，其中包含发送给服务供应商的能源使用率数据。
- 遥测应用程序发送基于能源使用率数据分析的控制命令。
- 有关更多信息，请参阅遥测场景：家庭能源监控。

图 *1:* 智慧电力度量

图 *2:* 智慧健康状况监测



- 遥测应用程序向医院和医生发送您的健康数据。
- 可以基于健康状况数据的分析发送 MQTT 消息警报和反馈。
- 有关更多信息，请参阅遥测场景：家庭患者监视。

您可以通过为 MQTT protocol 编写自己的应用程序，将 MQTT 构建到小型设备中。 为了帮助您执行此操作，IBM 提供客户机库，这些库支持通过 MQTT 运行的应用程序。 请参阅 第 9 页的『MQTT 客户机入门』。 IBM 为 iOS 应用程序和 Android 应用程序提供客户机库 V 7.5.0.1，以及为平台不可知的 Web 应用程序提供 JavaScript 浏览器客户机。 V 7.5.0.1 JavaScript 客户机页面通过 WebSockets 通过 MQTT 协议连接到 IBM MessageSight 和 IBM WebSphere MQ 。 IBM 还为 Linux® 和 Windows 上的 C 和 Java 提供了 MQTT 样本应用程序。

C 和 Java 库运行在 iOS、Android、Windows 和大量 UNIX and Linux 平台上。 您可以将 MQTT 客户机库的 C 源代码移植到其他平台。 可以从 Eclipse Paho 项目通过一个开放式源代码许可证来获得 C 和 Java 的 MQTT 客户机库。 请参阅 Eclipse Paho。 MQTT protocol 规范是开放的，并且可从 MQTT.org 获取。

MQTT protocol

客户机较小并且 MQTT protocol 高效地使用网络带宽，在这个意义上，其为轻量级。 MQTT 协议支持可靠的传送和即发即弃的传输。 在此协议中，消息传送与应用程序脱离。 脱离应用程序的程度取决于写入 MQTT 客户机和 MQTT 服务器的方式。 脱离式传送能够将应用程序从任何服务器连接和等待消息中解脱出来。 交互模式与电子邮件相似，但在应用程序编程方面进行了优化。

MQTT V3.1 协议已发布；请参阅 MQTT V3.1 协议规范。 该规范可识别协议的大量鲜明特征。

- 它是一种发布/预订协议。

  除提供一对多消息分发外，发布/预订也脱离了应用程序。 对于具有多个客户机的应用程序来说，这些功能非常有用。
- 它与消息内容没有任何关系。
- 它通过 TCP/IP 运行，TCP/IP 可以提供基本网络连接。
- 它针对消息传送提供三种服务质量：

  “至多一次”
  消息根据底层因特网协议网络尽最大努力进行传递。 可能会丢失消息。

  例如，将此服务质量与通信环境传感器数据一起使用。 对于是否丢失个别读取或是否稍后立即发布新的读取并不重要。

  “至少一次”
  保证消息抵达，但可能会出现重复。

  “刚好一次”
  确保只收到一次消息。

  例如，将此服务质量与记帐系统一起使用。 重复或丢失消息可能会导致不便或收取错误费用。

- 它是一种管理网络中消息流的经济方式。 例如，固定长度的标题仅 2 个字节长度，并且协议交换可最大程度地减少网络流量。
- 它具有 “Last Will and Testament” 功能，用于通知订户客户机与 MQTT 服务器的异常断开连接。 请参阅第 116 页的『“最后的消息”发布』。

MQTT version 3.1 受 IBM IBM WebSphere MQ 和 IBM MessageSight 支持。 MQTT 通过 TCP/IP 实现。 此协议的另一个版本 MQTT-S 可用于非 TCP/IP 网络。 请参阅 MQTT-S version 1.2 规范。

MQTT 社区

对于为 IBM MessageSight 和 IBM WebSphere MQ 编写应用程序的 MQTT 开发者， IBM 正在运行 IBM Developer 消息传递 社区 。

在 MQTT.org，可以很好地了解和讨论 MQTT 协议的实施和扩展。

MQTT 是开放式源代码 Eclipse 项目，基于 Eclipse 技术项目。 Paho 社区正在开发开放式源代码客户机和服务器。 请参阅 Eclipse Paho。

MQTT 简介

了解有关使用 MQ 遥测传输在移动应用程序之间发送消息的信息 (MQTT)。 该协议旨在用于无线和低带宽网络。 此移动应用程序通过调用 MQTT 库，使用 MQTT 来发送和接收消息。 消息通过 MQTT 消息传递服务器进行交换。 MQTT 客户机和服务器可应对为移动应用程序可靠地传递消息所带来的复杂性，并且将网络管理成本保持在较低水平。

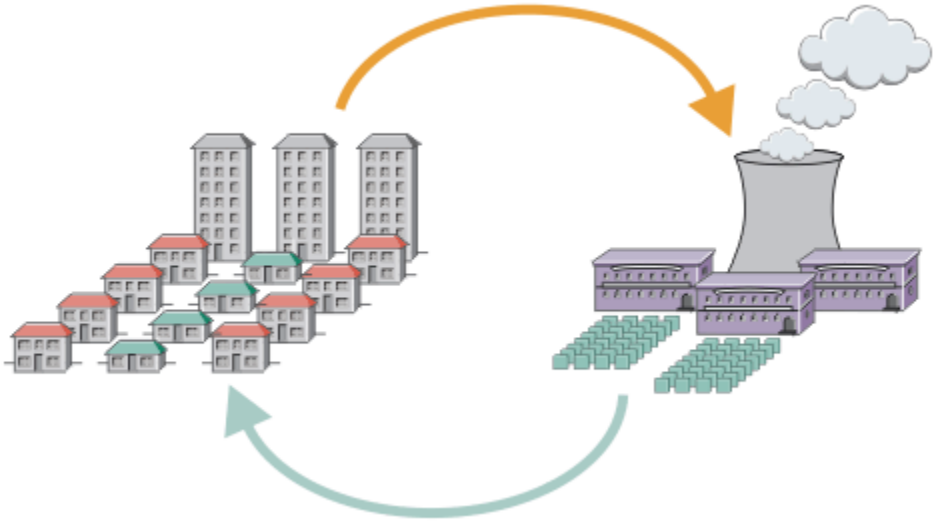
MQTT 应用程序在移动设备上运行，例如智能手机和平板电脑。 MQTT 还用于遥测，以接收来自传感器的数据并对传感器进行远程控制。 对于移动设备和传感器，MQTT 通过可靠的传送提供高度可扩展的发布/预订协议。 要发送和接收 MQTT 消息，请将 MQTT 客户机库添加到应用程序。

MQTT 客户机库很小。 该库充当邮箱功能，通过连接到 MQTT 服务器的其他 MQTT 应用程序发送和接收消息。 通过发送消息，而不是保持连接到等待响应的服务器，MQTT 应用程序延长了电池的使用时间。 该库通过运行 MQTT version 3.1 协议的 MQTT 服务器向其他设备发送消息。 您可以向特定的客户机发送消息，或者使用发布/预订消息传递来连接多个设备。

MQTT 客户机库使用 MQTT 协议将移动设备和传感器的应用程序连接到 MQTT 服务器。

IBM MessageSight 和 IBM WebSphere MQ 是 MQTT 服务器。 它们能连接大量的 MQTT 客户机应用程序，还能将 MQTT 和 IBM WebSphere MQ 网络连到一起。 请参阅 第 122 页的『MQTT 服务器入门』。 IBM WebSphere MQ 和 IBM MessageSight 都可以形成一个网桥，用于连接正在移动设备和传感器上运行的外部 Web 应用程序以及在企业内运行的其他类型的发布/预订和消息传递应用程序。 通过此网桥，可以更轻松地构建包含移动设备和传感器的 “智能解决方案” 。

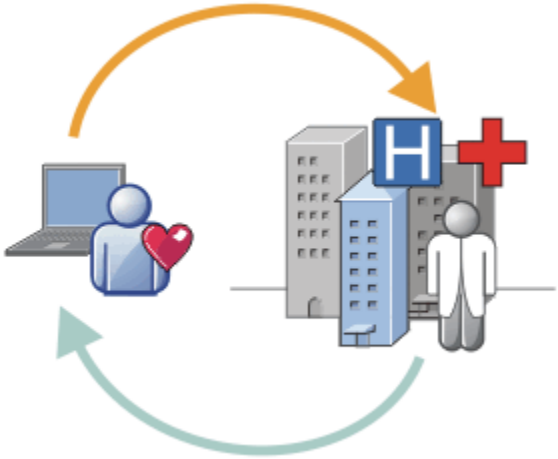
智慧解决方案将来自因特网的信息应用到移动设备和传感设备上运行的应用程序。 基于遥测的两个智慧应用程序示例为智慧电力和智慧健康状况服务。



- 一条 MQTT 消息，其中包含发送给服务供应商的能源使用率数据。
- 遥测应用程序发送基于能源使用率数据分析的控制命令。
- 有关更多信息，请参阅遥测场景：家庭能源监控。

图 *3:* 智慧电力度量

图 *4:* 智慧健康状况监测



- 遥测应用程序向医院和医生发送您的健康数据。
- 可以基于健康状况数据的分析发送 MQTT 消息警报和反馈。
- 有关更多信息，请参阅遥测场景：家庭患者监视。

您可以通过为 MQTT protocol 编写自己的应用程序，将 MQTT 构建到小型设备中。 为了帮助您执行此操作，IBM 提供客户机库，这些库支持通过 MQTT 运行的应用程序。 请参阅 第 9 页的『MQTT 客户机入门』。 IBM 为 iOS 应用程序和 Android 应用程序提供客户机库 V 7.5.0.1，以及为平台不可知的 Web 应用程序提供 JavaScript 浏览器客户机。 V 7.5.0.1 JavaScript 客户机页面通过 WebSockets 通过 MQTT 协议连接到 IBM MessageSight 和 IBM WebSphere MQ 。 IBM 还为 Linux 和 Windows 上的 C 和 Java 提供了 MQTT 样本应用程序。

C 和 Java 库运行在 iOS、Android、Windows 和大量 UNIX and Linux 平台上。 您可以将 MQTT 客户机库的 C 源代码移植到其他平台。 可以从 Eclipse Paho 项目通过一个开放式源代码许可证来获得 C 和 Java 的 MQTT 客户机库。 请参阅 Eclipse Paho。 MQTT protocol 规范是开放的，并且可从 MQTT.org 获取。

MQTT protocol

客户机较小并且 MQTT protocol 高效地使用网络带宽，在这个意义上，其为轻量级。 MQTT 协议支持可靠的传送和即发即弃的传输。 在此协议中，消息传送与应用程序脱离。 脱离应用程序的程度取决于写入 MQTT 客户机和 MQTT 服务器的方式。 脱离式传送能够将应用程序从任何服务器连接和等待消息中解脱出来。 交互模式与电子邮件相似，但在应用程序编程方面进行了优化。

MQTT V3.1 协议已发布；请参阅 MQTT V3.1 协议规范。 该规范可识别协议的大量鲜明特征。

- 它是一种发布/预订协议。

  除提供一对多消息分发外，发布/预订也脱离了应用程序。 对于具有多个客户机的应用程序来说，这些功能非常有用。
- 它与消息内容没有任何关系。
- 它通过 TCP/IP 运行，TCP/IP 可以提供基本网络连接。
- 它针对消息传送提供三种服务质量：

  “至多一次”
  消息根据底层因特网协议网络尽最大努力进行传递。 可能会丢失消息。

  例如，将此服务质量与通信环境传感器数据一起使用。 对于是否丢失个别读取或是否稍后立即发布新的读取并不重要。

  “至少一次”
  保证消息抵达，但可能会出现重复。

  “刚好一次”
  确保只收到一次消息。

  例如，将此服务质量与记帐系统一起使用。 重复或丢失消息可能会导致不便或收取错误费用。
- 它是一种管理网络中消息流的经济方式。 例如，固定长度的标题仅 2 个字节长度，并且协议交换可最大程度地减少网络流量。
- 它具有 “Last Will and Testament” 功能，用于通知订户客户机与 MQTT 服务器的异常断开连接。 请参阅第 116 页的『“最后的消息”发布』。

MQTT version 3.1 受 IBM IBM WebSphere MQ 和 IBM MessageSight 支持。 MQTT 通过 TCP/IP 实现。 此协议的另一个版本 MQTT-S 可用于非 TCP/IP 网络。 请参阅 MQTT-S version 1.2 规范。

MQTT 社区

对于为 IBM MessageSight 和 IBM WebSphere MQ 编写应用程序的 MQTT 开发者， IBM 正在运行 IBM Developer 消息传递 社区 。

在 MQTT.org，可以很好地了解和讨论 MQTT 协议的实施和扩展。

MQTT 是开放式源代码 Eclipse 项目，基于 Eclipse 技术项目。 Paho 社区正在开发开放式源代码客户机和服务器。 请参阅 Eclipse Paho。

MQTT 客户机入门

您可以通过构建和运行使用 MQTT 客户机库的样本 MQTT 客户机应用程序来开始开发移动或 机器到机器 (M2M) 应用程序。 样本应用程序和关联的客户机库在 移动消息传递和 M2M 客户机包 中可从 IBM 获取。 存在使用 Java， JavaScript 和 C 编写的应用程序和客户机库版本。 您可以在大多数平台和设备 (包括 Apple 中的 Android 设备和产品) 上运行这些应用程序。

开始之前

要构建并运行您的应用程序，您需要在为目标设备或平台构建应用程序以及所用编程语言方面具备一些经验。 通常情况下，少许经验便足以在所选设备或平台上启动并运行样本应用程序。

如果您使用企业强度的 MQTT 服务器（如 IBM WebSphere MQ 或 IBM MessageSight），那么可以与现有企业应用程序交换样本应用程序中的信息。

关于此任务

您的目标如下：

1. 选择可以将客户机应用程序连接到的 MQTT 服务器。
2. 下载 移动消息传递和 M2M 客户机包。
3. 为您的目标设备或平台，从客户机包构建样本应用程序。
4. 通过将样本连接到 MQTT 服务器来验证样本的行为是否与预期一致。

由于为您的设备或平台构建并测试了样本应用程序，您创建了可随后用于构建自己的客户机应用程序的工作开发环境。

移动消息传递和 M2M 客户机包包含 MQTT SDK。 此 SDK 为您提供以下资源：

- 以 Java、JavaScript 和 C 编写的样本 MQTT 客户机应用程序。
- MQTT 客户机库，它们支持这些客户机应用程序并使其能够在大多数平台和设备上运行。

该 SDK 还包含 C 的 MQTT 客户机的源代码。 您可以修改此源代码来为其他平台构建 C MQTT 客户机库。 有关执行此操作的帮助，请参阅第 27 页的『构建 C MQTT 客户机库』。 C 的 MQTT 客户机 的源代码也随 Eclipse Paho 的开放式源代码许可证一起提供。

过程

以下文章指导您完成关于在台式计算机或 Android 或 Apple 的移动设备上构建并运行样本 MQTT 应用程序的特定于平台的步骤。

- 第 11 页的『Java MQTT 客户机入门』
- 第 16 页的『Java 的 MQTT 客户机 on Android 入门』
- V 7.5.0.1
  第 21 页的『JavaScript 的 MQTT 消息传递客户机入门』
- 第 23 页的『C MQTT 客户机入门』
- 第 43 页的『iOS 上的 C MQTT 客户机入门』

下一步做什么

要开发新的 MQTT 应用程序，您必须具备或学会以下技能：

- 使用设备或平台所需的语言进行编程。
- 对目标设备或平台进行编程。
- 设计发布/预订应用程序。
- 为 MQTT 编程模型设计程序。
- 设计要在所选移动设备上运行的程序。
- 使用 SSL 和 JAAS 来保护程序。

您无需任何网络编程技能便可以将 MQTT 客户机与其他设备或应用程序连接，因为 MQTT 是消息传递和排队系统。 MQTT 客户机库负责管理您应用程序的网络连接。

要将 MQTT 客户机与现有企业应用程序集成，可以使用以下两个选项。 您可以将 MQTT 发布/预订主题与 IBM WebSphere MQ 或 JMS 应用程序（举例而言）共享，或者也可以编写自己的集成适配器来作为另一个 MQTT 客户机。

目前，要参考的信息源是：

- 为 WebSphere MQ Telemetry 开发应用程序
- MQTT.org
- Eclipse Paho

相关概念

第 122 页的『MQTT 服务器入门』

Java MQTT 客户机入门

使用 IBM MessageSight 或 IBM WebSphere MQ 作为 MQTT 服务器启动并运行 MQTT Client for Java 样本应用程序。 样本应用程序使用来自 IBM 的 MQTT 软件开发工具箱 (SDK) 中的客户机库。 `SampleAsyncCallBack` 样本应用程序是用于为 Android 和其他事件驱动的操作系统编写 MQTT 应用程序的模型。

开始之前

- 您可以在具有 JSE 1.5 或更高版本 (即 “Java 兼容”) 的任何平台上运行 Java 的 MQTT 客户机 应用程序. 请参阅 IBM 移动消息传递和 M2M 客户机包的系统需求。
- 如果客户机与服务器之间存在防火墙，请检查它是否未阻止 MQTT 流量。

关于此任务

该任务的目的是检查您是否可以构建和运行 MQTT Client for Java 样本应用程序，将其作为 MQTT version 3 服务器连接到 IBM WebSphere MQ 或 IBM MessageSight ， 以及交换消息。

按照此任务来从 Eclipse 工作台或从命令行运行样本应用程序。 该示例中的步骤是针对 Windows 的。 通过一些小修改，您可以在任何支持 JSE 1.5 或更高版本的平台上运行样本应用程序。

您可以在与 IBM WebSphere MQ 相同的服务器上运行应用程序，其中为您配置了用于运行连接到 IBM WebSphere MQ 的应用程序的环境。 执行任务 第 125 页的『从命令行配置 MQTT 服务』 以在 Windows 或 Linux 上使用 IBM WebSphere MQ Telemetry 选项来安装和配置 IBM WebSphere MQ 。 安装并配置了环境后，运行样本应用程序 `MQTTV3Sample` 来验证安装。

过程

1. 选择可以将客户机应用程序连接到的 MQTT 服务器。

   服务器必须支持 MQTT version 3.1 协议。 来自 IBM 的所有 MQTT 服务器都执行此操作，包括 IBM WebSphere MQ 和 IBM MessageSight。 请参阅第 122 页的『MQTT 服务器入门』。
2. 可选： 配置 MQTT 服务器。
   - 在 IBM WebSphere MQ 上，您必须完成以下任意一项任务来设置队列管理器并配置其遥测 (MQXR) 服务：
     - 第 125 页的『从命令行配置 MQTT 服务』
     - 第 127 页的『通过 IBM WebSphere MQ Explorer 配置 MQTT 服务』
   - 在其他服务器上，请参考服务器文档。 Really Small Message Broker 不需要执行任何配置步骤。 请参阅 Really Small Message Broker。
3. 下载 移动消息传递和 M2M 客户机包 并安装 MQTT SDK。

   不存在安装程序，您只需展开下载的文件。

   a. 下载 移动消息传递和 M2M 客户机包。

   b. 创建将在其中安装 SDK 的文件夹。

      您可能希望将此文件夹命名为 `MQTT`。 此文件夹的路径在此被称为 *`sdkroot`*。

   c. 将压缩的 移动消息传递和 M2M 客户机包 文件内容展开到 *`sdkroot`* 中。 此展开操作将创建以 *`sdkroot`*`\SDK` 开头的目录树。
4. 安装 Java 开发包 (JDK) V 6 或更高版本。

   由于您正在开发 Java 应用程序 for Android，因此 JDK 必须来自 Oracle。 您可以从 Java SE 下载获取 JDK 。
5. 编译并运行一个或多个 MQTT Client for Java 样本应用程序:

- 第 12 页的『从命令行编译并运行 Paho 样本程序』
- 第 14 页的『从 Eclipse 编译并运行所有 MQTT 客户机样本 Java 应用程序』
- 第 16 页的『Java 的 MQTT 客户机 on Android 入门』

以下 MQTT 客户机样本 Java 应用程序包含在 SDK 中：

MQTTV3Sample
该样本也包含在 IBM WebSphere MQ 中，并且链接到 `com.ibm.micro.client.mqttv3.jar` 程序包。

Sample
`Sample` 位于 Paho 包中，并且链接到 `org.eclipse.paho.client.mqttv3` 包。 它与 `MQTTV3Sample` 类似；它会等到每个 MQTT 操作都完成为止。

SampleAsyncWait
`SampleAsyncWait` 位于 `org.eclipse.paho.client.mqttv3` 程序包中。 它使用异步 MQTT API；它在其他线程上等到操作完成为止。 主要线程可以执行其他工作，直到它在等待 MQTT 操作完成的线程上同步为止。

SampleAsyncCallBack
`SampleAsyncCallBack` 位于 `org.eclipse.paho.client.mqttv3` 程序包中。 它调用异步 MQTT API。 异步 API 没有等待 MQTT 完成对调用的处理；它返回到应用程序。 此应用程序继续执行其他任务，然后等待下一事件到达以供其处理。 MQTT 在完成处理后将事件通知发回此应用程序。 事件驱动的 MQTT 接口适用于 Android 和其他事件驱动的操作系统的服务和活动编程模型。

例如，查看 `mqttExerciser` 样本如何使用服务和活动编程模型将 MQTT 集成到 Android。

mqttExerciser
`mqttExerciser` 是 Android 的样本程序。 因为其构建和运行方式不同，所以对其单独进行描述。 请参阅第 16 页的『Java 的 MQTT 客户机 on Android 入门』。

结果

您编译并运行了 MQTT Java 样本应用程序，这些应用程序已连接到作为 MQTT 服务器的 IBM WebSphere MQ 或 IBM MessageSight。

下一步做什么

研究 Javadoc 参考信息; 请参阅 第 14 页的『从 Eclipse 编译并运行所有 MQTT 客户机样本 Java 应用程序』的步骤 第 14 页的『3』 。 或者，打开位于 移动消息传递和 M2M 客户机包的 `SDK\clients\java\doc\javadoc` 目录中的 Javadoc html 文件。

从命令行编译并运行 Paho 样本程序

从命令行编译并运行 Paho 样本应用程序 `Sample.java` 。 该样本位于 MQTT SDK 中。 此样本是使用 `org.eclipse.paho.client.mqttv3` 包中的 MQTT Paho 客户机库构建的。 它演示 MQTT 发布者和订户。 可以按相同方式构建并运行同一目录中的其他两个 Paho 样本。 它们的不同之处在于异步调用 MQTT 库。

开始之前

执行主要任务中的步骤 第 11 页的『1』 到 第 11 页的『4』 以配置 MQTT 服务器并下载 移动消息传递和 M2M 客户机包。

关于此任务

编译并运行 SDK 客户机样本子目录 `SDK\clients\java\samples` 中的 `Sample.java`。 Java 代码位于 `SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app` 目录中。

过程

1. 在客户机样本目录中创建脚本以在所选平台上编译和运行 `Sample` 。

以下脚本在 Windows 上编译并运行该样本。

```
@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
```

图 *5:* 编译并运行 *Sample.java*

2. 运行脚本。

   结果:

```
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
```

图 *6: MQTTV3Sample* 订户

```
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
```

图 *7: MQTTV3Sample* 发布者

   如果您将订户应用程序保持运行，那么无法重新运行同一订户。 新订户的客户机标识与旧订户相同。 您不能同时运行具有相同客户机标识的两个 MQTT 客户机。 您可以设置 `-i` 选项来设置客户机标识，以便您可以同时运行不同的订户。

   如果您重新运行同一客户机，那么可以利用 `-c` 选项在将 `cleansession` 设置为 false 的情况下启动客户机。 凭借该选项，您可以探索中断的客户机会话的行为。

3. 通过按 enter 键或关闭窗口来结束订户。

下一步做什么

创建脚本来编译并运行 `SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app` 子目录中的其他样本。 复制第 13 页的图 5 中的脚本，并将 `Sample` 替换为 `SampleAsyncWait` 或 `SampleAsyncCallBack`。 其他样本是同步 `Sample` 程序的异步版本。

SampleAsyncWait
`SampleAsyncWait` 位于 `org.eclipse.paho.client.mqttv3` 程序包中。 它使用异步 MQTT API；它在其他线程上等到操作完成为止。 主要线程可以执行其他工作，直到它在等待 MQTT 操作完成的线程上同步为止。

SampleAsyncCallBack
`SampleAsyncCallBack` 位于 `org.eclipse.paho.client.mqttv3` 程序包中。 它调用异步 MQTT API。 异步 API 没有等待 MQTT 完成对调用的处理；它返回到应用程序。 此应用程序继续执行其他任务，然后等待下一事件到达以供其处理。 MQTT 在完成处理后将事件通知发回此应用程序。 事件驱动的 MQTT 接口适用于 Android 和其他事件驱动的操作系统的服务和活动编程模型。

例如，查看 `mqttExerciser` 样本如何使用服务和活动编程模型将 MQTT 集成到 Android。

异步样本演示如何在等待 MQTT 客户机时减少 MQTT 应用程序阻塞的时间量。 重要的是消除主线程中的阻塞调用以在移动环境中提升响应速度和电池寿命。

这些样本演示在 MQTT 客户机中调用异步接口的两种模式。

1. 当 MQTT 正在等待网络交互时， SampleAsyncWait 不会阻塞。
2. SampleAsyncCallBack 没有阻止等待 MQTT 客户机完成任何操作。 后者在 JavaScript 页面从浏览器调用 MQTT 客户机时必要。 JavaScript 页面不得阻塞。 对操作的响应必须发回主浏览器线程，该线程调用您编写来处理通知的 MQTT 事件处理程序。

从 Eclipse 编译并运行所有 MQTT 客户机样本 Java 应用程序

编译并运行 移动消息传递和 M2M 客户机包中的 MQTT 客户机样本 Java 应用程序。 它们演示 MQTT 发布者和订户。

关于此任务

编译并运行 Eclipse 中的 MQTT Java 样本， Sample 和 MQTTV3Sample 。 Sample 位于 *sdkroot*\SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app SDK 客户机子目录中，MQTTV3Sample.java 位于 *sdkroot*\SDK\clients\java\samples 中。

过程

1. 下载 Eclipse IDE for Java Developers。
2. 在 Eclipse 中创建名为 MQTT Samples 的 Java 项目。
   a) **文件** > **新建** > **Java 项目** 并输入 MQTT Samples。 单击**下一步**。

      检查 JRE 是正确版本还是更高版本。JSE 必须为 V 1.5 或更高版本。
   b) 在**"Java 设置"**窗口中，单击**链接其他源文件夹**。
   c) 浏览到您在其中安装 MQTT Java SDK 文件夹的目录。 选择 *sdkroot*\SDK\clients\java\samples 文件夹，然后单击 **确定** > **下一步** > **完成**。
   d) 在**"Java 设置"**窗口中，单击**库** > **添加外部 Jar**
   e) 浏览到您在其中安装 MQTT Java SDK 文件夹的目录。 找到 *sdkroot*\SDK\clients\java 文件夹并选择 org.eclipse.paho.client.mqttv3.jar 和 com.ibm.micro.client.mqttv3.jar 文件; 单击 **打开** > **完成**。

      MQTTV3Sample.java 样本链接到 com.ibm.micro.client.mqttv3.jar，paho 目录树中的样本链接到 org.eclipse.paho.client.mqttv3.jar。 将保留 com.ibm.micro.client.mqttv3.jar ，以便现有 MQTT 应用程序继续构建和运行而不进行更改。

   MQTT Samples 项目已构建，出现一些警告，但未出现错误。

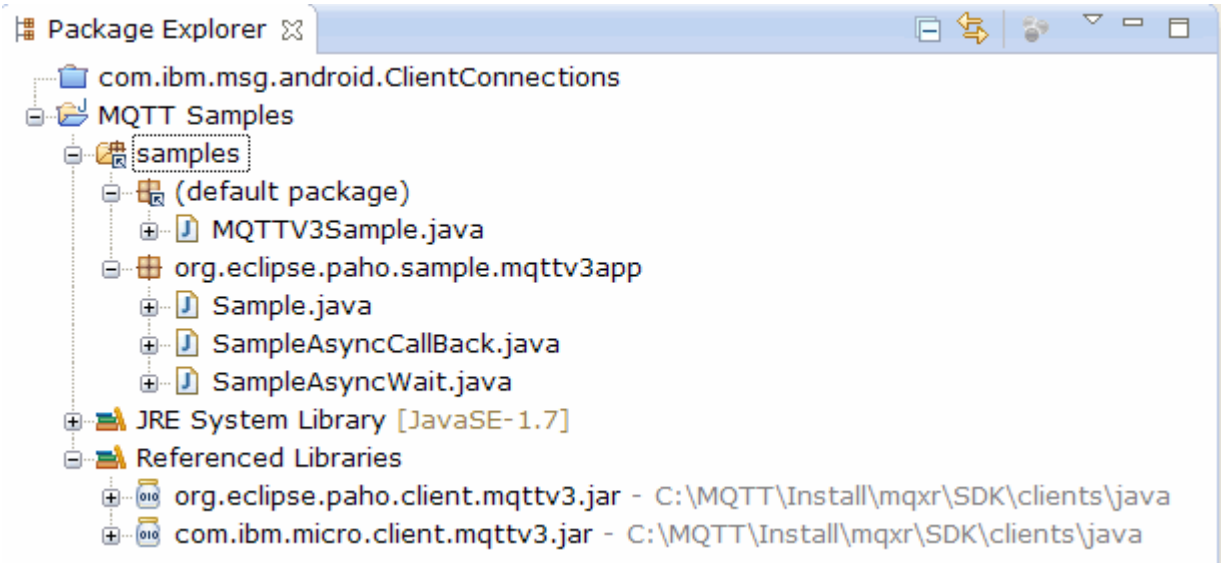


图 *8: Java MQTT* 客户机项目

3. 可选： 安装 MQTT 客户机 Javadoc。

安装了 MQTT 客户机 Javadoc 后， Java 编辑器将在悬浮式帮助中描述 MQTT 类。

a) 在 Java 项目中打开 **包资源管理器** > **引用的库** 。 右键单击 `org.eclipse.paho.client.mqttv3.jar` > **属性**。

b) 在“属性导航器”中，单击 **Javadoc 位置**。

c) 单击 " **Javadoc 位置** " 页面中的 **Javadoc URL** > **浏览** ， 然后找到 `SDK\clients\java\doc\javadoc` 文件夹 > **确定**。

d) 单击**验证** > **确定**

将提示您打开浏览器以查看文档。

e) 为 `com.ibm.micro.client.mqttv3.jar` 文件重复此过程。

4. 创建发布者和订户运行时配置以运行 `mqttv3app.Sample` 应用程序。

a) 右键单击 **样本** 类，然后单击 **运行方式** > **运行配置**。

b) 右键单击 **Java 应用程序** > **新建** ， 然后输入名称 `SampleSubscriber`。

c) 单击“自变量”选项卡，输入程序自变量，然后单击**应用**。

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

d) 重复最后一步以通过省略 `-a subscribe` 参数来创建 `SamplePublisher` 配置。

5. 运行 `mqttv3app.Sample` 订户，然后运行发布者。

a) 单击**运行** > **运行配置**

b) 单击 **SampleSubscriber** > **运行**。

打开“**控制台**”视图。 订户正在等待发布。

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

c) 单击 **SamplePublisher** > **运行**。

打开“**控制台**”视图。 您会看到由发布者创建的发布：

```
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) 将控制台视图切换到订户控制台。

切换控制台图标为  。

订户收到此发布：

```
...
Time: 2012-11-09 14:09:30.593  Topic: Sample/Java/v3  Message: Message from blocking
MQTTv3 Java client sample  QoS: 2
```

6. 可选： 为 `MQTTV3Sample` 创建发布者和订户运行时配置。

a) 右键单击 **MQTTV3Sample** 类，然后单击 **运行方式** > **运行配置**。

b) 右键单击 **Java 应用程序** > **新建** ， 然后输入名称 `MQTTV3SampleSubscriber`。

c) 单击“自变量”选项卡，输入程序自变量，然后单击**应用**。

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

d) 重复最后一步以通过省略 `-a subscribe` 参数来创建 `MQTTV3SamplePublisher` 配置。

7. 可选： 运行 `MQTTV3Sample` 订户，然后运行发布者。

a) 单击**运行** > **运行配置**

b) 单击 **MQTTV3SampleSubscriber** > **运行**。

打开**“控制台”**视图。 订户正在等待发布。

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

c) 单击 **MQTTV3SamplePublisher** > **运行**。

打开**“控制台”**视图。 您会看到由发布者创建的发布。

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

d) 将控制台视图切换到订户控制台。

切换控制台图标为  。

订户收到此发布：

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

Java 的 MQTT 客户机 on Android 入门

您可以安装 Android 的 MQTT 客户机样本 Java 应用程序，以便与 MQTT 服务器交换消息。 应用程序使用来自 IBM 的 MQTT SDK 中的客户机库。 您可以自行构建应用程序，或者下载预先构建的样本应用程序。

开始之前

- 要了解有关受支持的 MQTT 客户机平台和参考平台的信息，请参阅 IBM 移动消息传递和 M2M 客户机包的系统需求。
- 如果客户机与服务器之间存在防火墙，请检查它是否未阻止 MQTT 流量。
- MQTT 客户机样本应用程序在 Ice Cream Sandwich (Android 4.0) 和更高版本上能够正常工作。 该 Android 版本还为平板电脑提供了更清晰的屏幕分辨率。

关于此任务

Android 的 MQTT 客户机样本 Java 应用程序 称为 “`mqttExerciser`”。 此应用程序使用 MQTT SDK 中的客户机库，用于和 MQTT 服务器交换消息。

您可以自行构建样本应用程序，然后将其从 Eclipse 导出为 `mqttExerciser.apk`，也可以使用 移动消息传递和 M2M 客户机包的 *`sdkroot`*`\SDK\clients\android\samples\apks` 文件夹中作为文件 `mqttExerciser.apk` 提供的预构建样本应用程序。 如果您选择自行构建应用程序，那么您构建的开发环境需要进行定制，以便将移动消息传递功能包含到 for Android 应用程序中。 当您开始在自己的应用程序中包含移动消息传递功能时，这会对您有所帮助。

过程

1. 选择可以将客户机应用程序连接到的 MQTT 服务器。

   服务器必须支持 MQTT version 3.1 协议。 来自 IBM 的所有 MQTT 服务器都执行此操作，包括 IBM WebSphere MQ 和 IBM MessageSight。 请参阅第 122 页的『MQTT 服务器入门』。
2. 获取合适的工具。

   安装 Java 开发包 (JDK) V 6 或更高版本。 由于您正在开发 Java 应用程序 for Android，因此 JDK 必须来自 Oracle。 您可以从 Java SE 下载获取 JDK 。

   您还需要 Eclipse 开发环境。 开发环境必须为 Eclipse 3.6.2 (Helios) 或更高版本。 Eclipse 必须具有级别至少为 6 的 Java 编译器，以便与 JDK 匹配。 您可以从 Eclipse Foundation 获取所有这些信息。

   最后，您需要 Android SDK。 这可以从 Get the Android SDK 获取。

3. 下载 移动消息传递和 M2M 客户机包 并安装 MQTT SDK。

   不存在安装程序，您只需展开下载的文件。

   a. 下载 移动消息传递和 M2M 客户机包。

   b. 创建将在其中安装 SDK 的文件夹。

      您可能希望将此文件夹命名为 MQTT。 此文件夹的路径在此被称为 *sdkroot*。

   c. 将压缩的 移动消息传递和 M2M 客户机包 文件内容展开到 *sdkroot* 中。 此展开操作将创建以 *sdkroot*\SDK 开头的目录树。

4. 可选： 构建 for Android mqttExerciser 样本应用程序。

   配置 Eclipse 和 Android 工具，并从 MQTT SDK 导入和构建 mqttExerciser 项目。

   注: 如果您不希望立刻进行此操作，可以使用预先构建的样本应用程序，该样本应用程序在 MQTT SDK 的 *sdkroot*\SDK\clients\android\samples\apks 文件夹中作为文件 mqttExerciser.apk 提供。

   a) 使用 JDK 中的 JRE 启动 Eclipse 开发环境。

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

   b) 从 Android SDK 中选择并安装一组包和平台。

      请参阅 Adding Platforms and Packages 以了解 Google 推荐的平台和包的列表。

      注: SDK 平台必须为 Android API 级别 16 或更高版本。 如果使用较早的 API 级别，无法成功编译项目。

   c) 将 Android Development Tools (ADT) 插件添加到 Eclipse。

   d) 将样本 mqttExerciser 应用程序项目导入到 Eclipse，并修复错误。

      i) 从 MQTT SDK 导入样本应用程序项目，位于路径 *sdkroot*\SDK\clients\android\samples\mqttExerciser 中。

      "问题"视图会列出许多构建错误。 在接下来的几个步骤中就会解决这些构建错误。

      ii) 将 org.eclipse.paho.client.mqttv3.jar 库复制到 Android 项目中的 **libs** 文件夹。

      Windows 例如，在 Windows 上，此文件夹位于 *sdkroot*\SDK\clients\java 文件夹下。这时会显示一个**"文件操作"**窗口。 接受**复制文件**选项，然后单击**确定**。

      iii) 右键单击项目文件夹 com.ibm.msg.android; 单击 **Android 工具 ... > 添加支持库 ...**。 阅读并接受许可条款，然后单击**安装**。

      iv) 右键单击项目文件夹 com.ibm.msg.android; 单击 **Android 工具 ... > 修订项目属性**。

      v) 如果工作空间仍然有大约 84 个错误（涉及覆盖超类方法），那么编译器一致性级别可能设置为 1.5 或更低级别。 Android SDK V16 要求编译器一致性级别不高于 1.5。 要修复剩余的错误，请完成以下步骤:

         a) 检查 Android SDK 和相应的 Eclipse 插件 ，如有必要，将它们更新至 Android SDK V17。

         b) 右键单击 **com.ibm.msg.android** 项目文件夹，然后选择 **属性 > Java 编译器**。 检查编译器一致性级别，至少将其设置为 1.6，然后重建工作空间。

   这样会构建该项目，会有一些警告，但不会有错误。

5. 在 Android 设备上安装并启动 MQTT 客户机样本 Java 应用程序 。

   请参阅 developer.android.com 页面 Running your app。

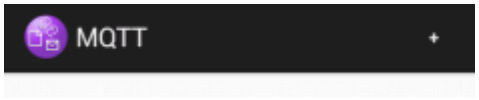
   如果您自己将应用程序构建为 Eclipse 项目，那么可以从 Eclipse 启动该应用程序。

   如果有应用程序包（APK）文件 mqttExerciser.apk，您可以使用 Android Debug Bridge (ADB) 安装命令将其安装到 Eclipse 之外。 该命令采用 APK 文件的位置作为自变量。 如果您要使用预先构建的样本应用程序，那么位置为 *sdkroot*\SDK\clients\android\samples\apks\mqttExerciser.apk。

6. 使用 mqttExerciser 样本应用程序 for Android 来连接主题，以及针对主题进行预订和发布。

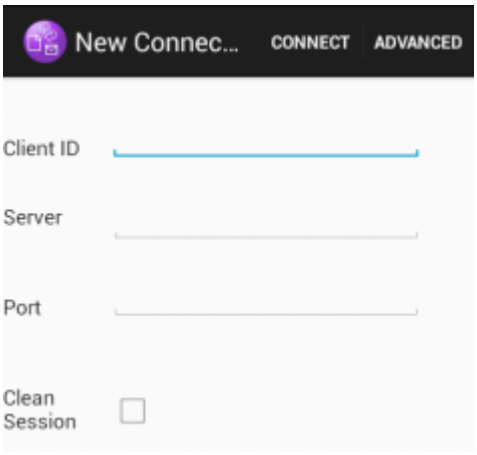
   a) 打开 Android 的 MQTT 客户机样本 Java 应用程序。

此窗口在您的 Android 设备中打开：



b) 连接到 MQTT 服务器。

i) 单击 **+** 符号以打开新的 MQTT 连接。



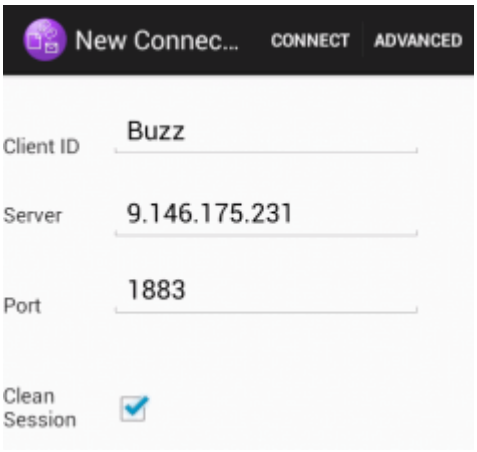
ii) 将任何唯一标识输入到**客户机标识**字段中。 请耐心，击键的速度可能会很慢。

iii) 在**服务器**字段中输入 MQTT 服务器的 IP 地址。

这是在第一个主要步骤中您选择的服务器。 IP 地址不得为 `127.0.0.1`。

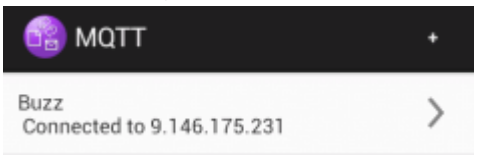
iv) 输入 MQTT 连接的端口号。

正常 MQTT 连接的缺省端口号为 1883。



v) 单击**连接**。

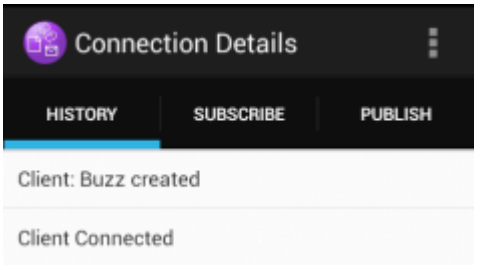
如果连接成功，那么您将看到 “正在连接” 消息，后跟此窗口：



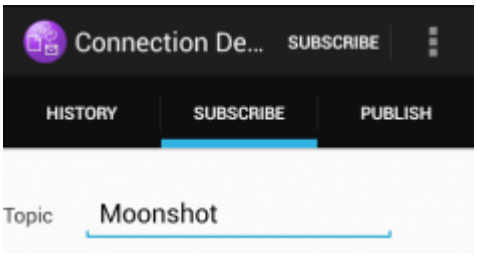
c) 预订主题。

i) 单击**已连接**消息。

这样会打开**“连接详细信息”**窗口，其中列出以下历史记录：



ii) 单击**预订**选项卡，然后输入一个主题字符串。

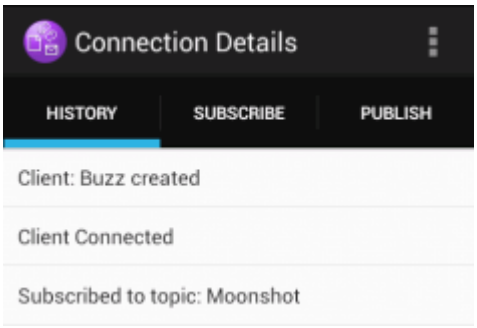


iii) 单击**预订**操作。

“已预订” 消息将在短时间内显示。

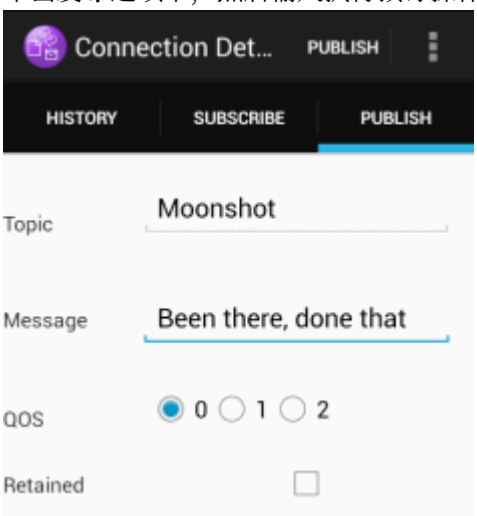
iv) 单击**历史记录**选项卡。

此历史记录现在包含以下预订：



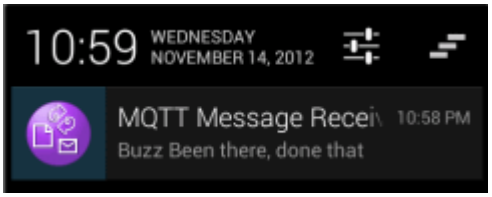
d) 现在发布到同一主题。

i) 单击**发布**选项卡，然后输入执行预订操作时所输入的同一主题字符串。 输入一条消息。

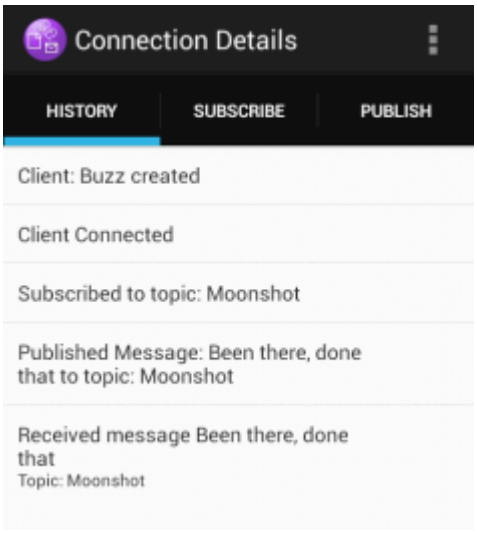


ii) 单击**发布**操作。

将在短时间内显示两条消息: “已发布”，后跟 “已预订”。 发布内容显示在状态区域（向下拉动分隔条可以打开状态窗口）。



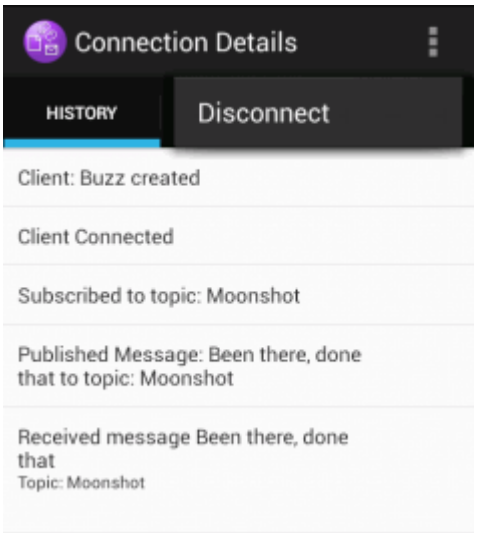
iii) 单击**历史记录**选项卡以查看完整的历史记录。



e) 将客户机实例断开连接。

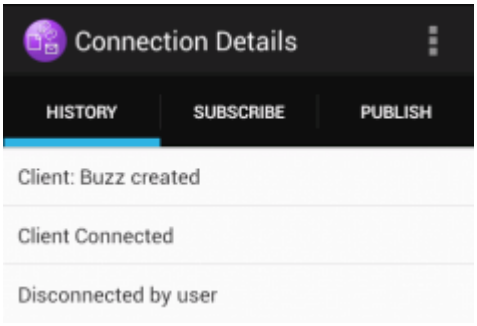
i) 单击操作栏中的菜单图标。 

Android 的 MQTT 客户机样本 Java 应用程序会将**"断开连接"**按钮添加到 MQTT**"连接详细信息"**窗口。

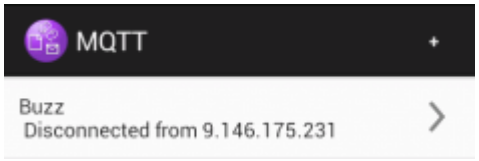


ii) 单击**断开连接**。

"已连接"状态会更改为"已断开连接":



f) 单击**后退**以返回到 MQTT 客户机样本 Java 应用程序会话的列表。



- 单击加号以启动新 MQTT 客户机样本 Java 应用程序会话。
- 单击断开连接的客户机以将其重新连接。
- 单击**后退**以返回到启动板。

g) 单击“任务”按钮以列出正在运行的应用程序。 找到此 MQTT 客户机样本 Java 应用程序。 将该图标滑出屏幕，以将其关闭。

下一步做什么

如果您自行构建了样本应用程序，那么您已准备好着手开发自己的 Android 应用程序，以便调用 MQTT 库来交换消息。 您可以在 `mqttExerciser` 中的类上对 Android 应用程序进行建模。 要研究样本，请为 `mqttExerciser` 项目中的 `com.ibm.msg.android` 和 `com.ibm.msg.android.service` 中的类生成 Javadoc 。

相关信息

通过带 ADT 的 Eclipse 管理项目

JavaScript 的 MQTT 消息传递客户机入门

您可以通过显示消息传递客户机样本主页并浏览其链接到的资源来开始使用 JavaScript 的 MQTT 消息传递客户机。 要显示此主页，请将 MQTT 服务器配置为接受来自 MQTT 消息传递客户机样本 JavaScript 页面的连接，然后将已在服务器上配置的 URL 输入 Web 浏览器中。 JavaScript 的 MQTT 消息传递客户机会在您设备上自动启动，然后消息传递客户机样本主页将显示。 该页面包含指向实用程序、编程接口文档、教程及其他有用信息的链接。

开始之前

要进行高级使用或者在生产中使用，您将需要重塑或移除消息传递客户机样本主页。 请注意并未保证从样本代码生成的用户界面符合任何辅助功能选项标准或辅助功能选项需求。

您需要 MQTT 服务器才能支持 JavaScript 的 MQTT 消息传递客户机。 此服务器必须支持基于 WebSockets 的 MQTT V3.1 协议。 IBM MessageSight 以及 IBM WebSphere MQ Version 7.5.0, Fix Pack 1 和更高版本，支持 MQTT protocol over WebSockets。 请参阅第 122 页的『MQTT 服务器入门』。 要安装 IBM WebSphere MQ 以进行 90 天的免费评估，请参阅第 124 页的『安装 IBM WebSphere MQ』。

最近确立了 WebSocket protocol。 如果客户机与服务器之间存在防火墙，请检查其是否没有阻止 WebSockets 流量。 同样，如果您的浏览器尚不支持 WebSocket protocol[1] 那么您将无法使用消息传递客户机样本主页上提供的客户机实用程序或教程。 第 22 页的表 1 列出了其最新版本已经过测试并显示为可用于消息传递客户机的浏览器。

[1] 具体来说，如果它不支持 RFC 6455 (WebSocket) 标准。

表 1: 用于 *JavaScript* 的 *MQTT* 消息传递客户机的受支持浏览器			
Android	**iOS**	**Linux**	**Windows**
Firefox for Android 19.0 和更高版本 Chrome for Android 25.0 和更高版本	Safari 6.0 和更高版本 Chrome 14.0 和更高版本	Firefox 6.0 和更高版本 Chrome 14.0 和更高版本	Firefox 6.0 和更高版本 Chrome 14.0 和更高版本

关于此任务

此任务中的大多数步骤将用于配置 MQTT 服务器。 访问 JavaScript 的消息传递客户机只需运行支持 WebSocket protocol 的浏览器即可。

在 IBM WebSphere MQ 上，执行通过创建样本通道来启用 IBM WebSphere MQ Telemetry 的步骤。 在端口 1883 上连接到样本缺省 MQTT WebSockets 通道。 消息传递客户机样本主页 URL 是 `http://`*`hostname`*`:1883 on IBM WebSphere MQ`。

在 IBM MessageSight 上，安装并设置设备，将消息传递中心配置为接受连接，然后创建 MQTT WebSockets 端点。

过程

1. 下载 移动消息传递和 M2M 客户机包，并选择一个可以连接客户机应用程序的 MQTT 服务器。

   请参阅第 9 页的『MQTT 客户机入门』。
2. 将 MQTT 服务器配置为接受来自 JavaScript 的 MQTT 消息传递客户机样本 HTML 页面的连接。
   - 在 IBM WebSphere MQ 上:
     - 如果您已有为 MQTT 配置的 IBM WebSphere MQ 队列管理器，请更改通道定义中的协议以同时支持 MQTT 和 HTTP。 请参阅 **ALTER CHANNEL**。
     - 要创建 IBM WebSphere MQ 队列管理器并配置样本 MQTT WebSockets 端点，请完成以下任一任务:
       - 第 125 页的『从命令行配置 MQTT 服务』
       - 第 127 页的『通过 IBM WebSphere MQ Explorer 配置 MQTT 服务』
3. 在您的设备上打开 Web 浏览器。
4. 输入消息传递客户机样本主页的 URL。
   - 在 IBM WebSphere MQ 上，这是 `http://`*`hostname`*`:1883`
   - 在 IBM MessageSight 上，这是 `http://`*`hostname`*`:`*`port`*

     其中 *`hostname`* 是以太网套接字（您在 IBM MessageSight 设备上已将其配置为客户机要连接到的端点）的 DNS 名称或 IP 地址，而 *`port`* 是您已分配给客户机的端点的 TCP/IP 端口号。

   将显示消息传递客户机样本主页。

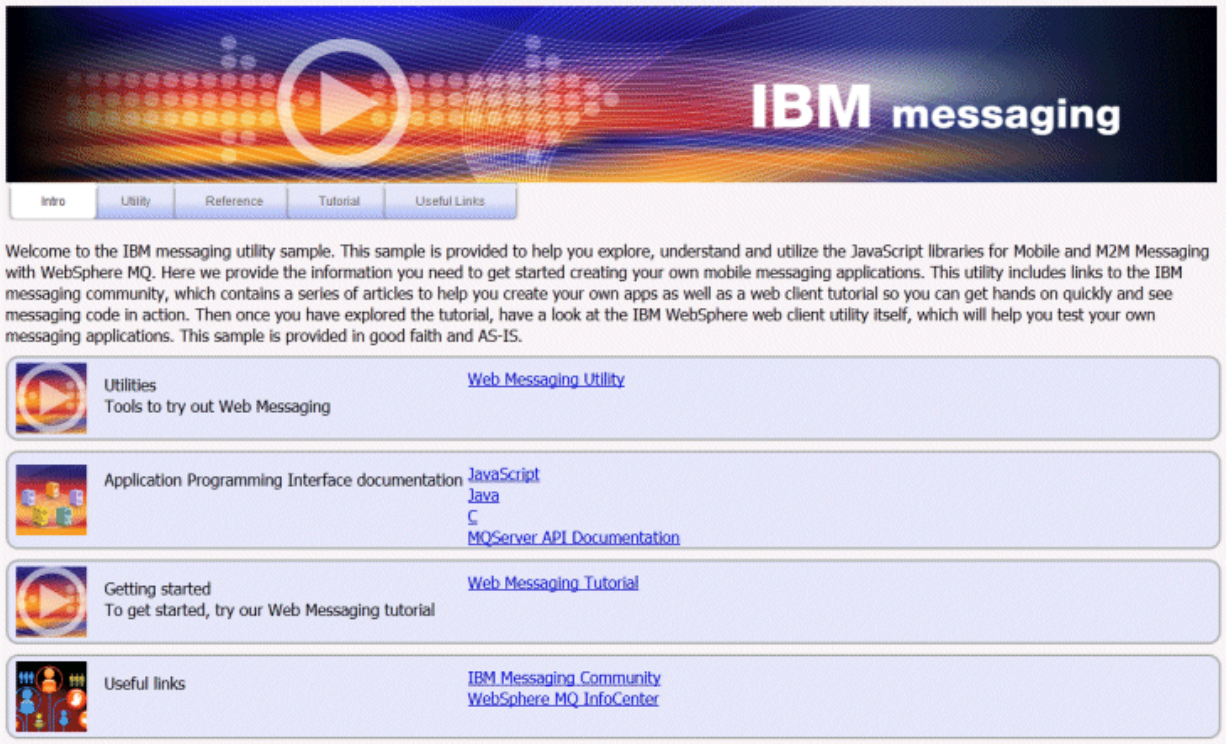


图 *9: JavaScript* 的 *MQTT* 消息传递客户机样本主页

结果

您已配置 WebSockets 的 MQTT 通道。

在 JavaScript 的 MQTT 消息传递客户机的样本主页中，单击 **Web 消息传递实用程序**以尝试使用消息传递客户机 API 中的不同功能。 例如，您可以连接到队列管理器，预订消息，然后发布一些消息。 您还可以单击 **Web 消息传递教程**以了解如何创建用于调用 JavaScript API 的 MQTT 消息传递客户机的网页。

相关概念

第 106 页的『JavaScript 的 MQTT 消息传递客户机和 Web 应用程序』
第 108 页的『如何使用 JavaScript 对消息传递应用程序进行编程』

相关任务

第 70 页的『通过 SSL 连接 JavaScript 的 MQTT 消息传递客户机 和 WebSockets』
通过将 JavaScript 的 MQTT 消息传递客户机 样本 HTML 页面与 SSL 和 WebSocket protocol 配合使用，将 Web 应用程序安全地连接到 IBM WebSphere MQ 。

C MQTT 客户机入门

在任何可编译 C 源代码的平台上启动并运行样本 C MQTT 客户机。 验证是否可以使用 IBM MessageSight 或 IBM WebSphere MQ 作为 MQTT 服务器运行样本 MQTT Client for C。

开始之前

- 如果客户机与服务器之间存在防火墙，请检查它是否未阻止 MQTT 流量。
- 有关受支持的和参考 C MQTT 客户机平台， 请参阅 IBM 移动消息传递和 M2M 客户机包的系统需求。

关于此任务

执行此任务以在 Windows 上从命令行或从 Microsoft Visual Studio 2010 编译并运行样本 C MQTT 客户机。 在命令行示例中，Microsoft Visual Studio 2010 也用于编译客户机。 修改命令行脚本以在其他平台上编译并运行样本。

过程

1. 选择可以将客户机应用程序连接到的 MQTT 服务器。

   服务器必须支持 MQTT version 3.1 协议。 来自 IBM 的所有 MQTT 服务器都执行此操作，包括 IBM WebSphere MQ 和 IBM MessageSight。 请参阅第 122 页的『MQTT 服务器入门』。
2. 将 C 开发环境安装在您要在其上进行构建的平台上。

   本主题中的示例内的 makefile 以下列工具为目标：

   - 

     对于 iOS，在具有 OS X 10.8.2 的 Apple Mac 上，使用来自 Xcode 的 iOS 开发工具。
   - 

     对于 Linux：来自 Red Hat® Enterprise Linux V6.2 的 gcc V4.4.6。

     glibc C 库的最低受支持级别是 2.12，Linux 内核的最低受支持级别是 2.6.32。
   - 

     对于 Microsoft Windows：Visual Studio V10.0。
3. 下载 移动消息传递和 M2M 客户机包 并安装 MQTT SDK。

   不存在安装程序，您只需展开下载的文件。

   a. 下载 移动消息传递和 M2M 客户机包。

   b. 创建将在其中安装 SDK 的文件夹。

      您可能希望将此文件夹命名为 MQTT。 此文件夹的路径在此被称为 *sdkroot*。

   c. 将压缩的 移动消息传递和 M2M 客户机包 文件内容展开到 *sdkroot* 中。 此展开操作将创建以 *sdkroot*\SDK 开头的目录树。
4. 可选： 执行 第 27 页的『构建 C MQTT 客户机库』中的步骤。

   请仅在移动消息传递和 M2M 客户机包不包含目标平台的 C 客户机库的情况下执行此步骤。
5. 编译并运行 MQTT 客户机样本 C 应用程序 MQTTV3Sample.c。

   - 从命令行，执行第 24 页的『从命令行编译并运行 MQTT 客户机样本 C 应用程序』中的步骤。
   - 从 IDE，执行第 25 页的『从 Microsoft Visual Studio 编译并运行 MQTT 客户机样本 C 应用程序』中的步骤。

从命令行编译并运行 MQTT 客户机样本 C 应用程序

从命令行编译并运行 MQTT 客户机样本 C 应用程序。 该样本位于 MQTT SDK 中。 它演示 MQTT 发布者和订户。

开始之前

安装 C 开发环境; 例如，示例中使用的 Microsoft Visual Studio 2010。

关于此任务

编译并运行 SDK 客户机子目录 *sdkroot*\SDK\clients\c\samples 中的 C 样本 MQTTV3Sample。

过程

在客户机样本目录中创建脚本以在所选平台上编译和运行 Sample 。

以下脚本在 Windows 32 位平台上编译并运行使用 Microsoft Visual Studio 2010 构建的样本。 从 *sdkroot*\SDK\clients\c\samples 子目录运行该脚本。

```
@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
```

结果

发布者和订户将输出写入到其命令窗口：

```
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 . . .
```

图 *10:* 来自发布者的输出

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

图 *11:* 来自订户的输出

从 Microsoft Visual Studio 编译并运行 MQTT 客户机样本 C 应用程序

从 Microsoft Visual Studio 编译并运行 MQTT 客户机样本 C 应用程序。 该样本位于移动消息传递和 M2M 客户机包中。 它演示 MQTT 发布者和订户。

开始之前

此示例使用 Microsoft Visual Studio 2010。 您可以在 Windows 和其他平台上使用其他 C 开发环境，例如 Eclipse IDE for C/C++ Developers。

关于此任务

使用 Microsoft Visual Studio 2010 编译并运行 C 样本 MQTTV3Sample 。 MQTTV3Sample.c 位于 SDK 客户机子目录 *sdkroot*\SDK\clients\c\samples 中。

过程

1. 启动 Microsoft Visual Studio。
2. 根据现有代码新建项目。
   a) 单击 **文件** > **新建** > **现有代码中的项目**。
   b) 将 **Visual C++** 选作要创建的项目的类型。
   c) 单击**下一步**。
3. 在**“项目位置和源文件”**窗口中指定参数。
   a) 单击**浏览**并找到 *sdkroot*\SDK\clients\c\samples 目录。
   b) 将该项目命名为 MQTTV3Sample。

c) 单击**下一步**。

4. 在**项目类型**列表中选择**控制台应用程序项目**。 单击**完成**
5. 仅配置“调试”配置。

   缺省情况下，Microsoft Visual Studio 会创建“发布”和“调试”配置。 在本教程中，您将配置“调试”配置。 为防止出现构建错误，请清除“发布”配置的**构建**选项。

   a) 单击 **项目 > MQTTV3Sample 属性 > 配置管理器**。 选择**发布**作为**活动解决方案配置**，并清除**构建**。

   b) 选择 **调试** 作为 **活动解决方案配置 > 关闭**。

   检查是否要在以下所有步骤中修改“调试”配置。
6. 修改“**MQTTV3Sample 属性页面**”中的 **C/C++** 设置。

   a) 在 " **MQTTV3Sample 属性页面** " 窗口中，打开 **配置属性 > C/C++ > 常规**。

   b) 在常规属性列表中，单击**其他 include 目录**，将您的目录路径添加到 *sdkroot*\SDK\clients\c\include，然后单击**应用**。
7. 修改**链接程序**设置

   a) 打开 **配置属性 > 链接程序 > 常规**。

   b) 在常规属性列表中，单击**其他库目录**，然后将您的目录路径添加到 *sdkroot*\SDK\clients\c\windows_ia32。

   c) 在链接程序属性列表中，单击**命令行**。 在**其他选项**数据输入区域中，输入 mqttv3c.lib，然后单击**应用**。
8. 从项目中除去 MQTTV3SSample.c 源文件。

   a) 在 " **解决方案资源管理器** " 窗口中打开 **MQTTV3sample > 源文件** 文件夹。

   b) 右键单击 **MQTTV3SSample.c > 从项目中排除**
9. 构建 MQTTV3Sample 项目。

   a) 在 Solution Explorer 中，右键单击 **MQTTV3sample** 项目，然后单击**构建**。

   该构建已完成且未出现任何错误。
10. 添加两个新项目以将 **MQTTV3Sample** 同时作为订户和发布者运行时实例运行。

    发布者和订户项目将包含用于运行 **MQTTV3Sample** 的命令。 它们尚未构建，且不包含任何代码。

    a) 在 **解决方案资源管理器**中，右键单击 **解决方案 ` MQTTV3Sample ` > 添加 > 新项目**。

    b) 将 Subscriber 输入到**名称**字段中。 保持选中 **Win32 控制台应用程序**。 单击**确定**。

    这样会启动“**Win32 应用程序向导**”。

    c) 在“**Win32 应用程序向导**”中，单击**下一步**。 选中**空项目 > 完成**

    d) 重复这些步骤以添加发布者项目。

    e) 右键单击订户项目，然后单击**设置为 StartUp 项目**

    这样会显示第 26 页的图 12 中的“**Solution Explorer**”窗口。

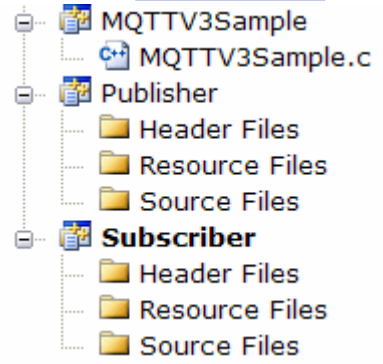


图 *12: MQTTV3Sample* 解决方案

11. 配置订户属性页面。

a) 右键单击解决方案资源管理器中的 **订户 属性** > **配置属性** > **属性** > **调试**

检查该窗口的标题是否为“**订户属性页面**”。

b) 单击**环境**。 输入 path=%path%;*sdkroot*\SDK\clients\c\windows_ia32，然后单击**应用**。

更改 *sdkroot* 以适合您的环境。

c) 单击**命令**，然后将 $(TargetPath) 替换为 MQTTV3Sample 模块的路径

例如，*sdkroot*\SDK\clients\c\samples\debug\MQTTV3Sample

更改 *sdkroot* 以适合您的环境。

d) 单击 **命令参数** 并输入 -a subscribe -b localhost -p 1883，然后单击 **应用**。

12. 配置发布者属性页面。

提示: 您可以通过单击“**Solution Explorer**”窗口中的项目来切换属性页面项目。

a) 针对发布者重复订户步骤。

命令参数为 -b localhost -p 1883

13. 停止用于构建发布者和订户项目的构建过程。

a) 在任何项目的属性页面中，单击**配置管理器**，并清除“发布”和“调试”配置中发布者和订户的**构建**。单击**关闭**。

14. 运行此样本。

a) 单击 **F5** 以启动订户

b) 右键单击 Solution Explorer 中的**发布者**，然后单击**调试** > **启动新实例**

结果

发布者和订户会输出到命令窗口中。 Visual Studio 会关闭发布者窗口。 请查看订户窗口（如下图所示），然后关闭订户窗口。

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

图 *13:* 来自订户的输出

下一步做什么

构建并运行异步发布者和订户。示例包括 *sdkroot*\SDK\clients\c\samples 中的 MQTTV3ASample.c 和 MQTTV3ASSample.c 。

构建 C MQTT 客户机库

执行以下步骤以构建 C MQTT 客户机库。 本主题包含针对许多平台的编译和链接开关，以及在 iOS 和 Windows 上构建库的示例。

开始之前

1. 仅当必要时才构建 C 客户机库。 如果在“软件开发工具箱”(SDK) 的 SDK\clients\c 子目录中，存在与目标平台匹配的预先构建的客户机库，请链接这些库。
2. 配置 MQTT 服务器以通过 MQTT 客户机样本 C 应用程序测试您构建的库。 请参阅第 122 页的『MQTT 服务器入门』。 通过运行 MQTT 客户机样本应用程序之一来验证服务器配置。
3. 如果要构建安全版本的 C 库（支持安全套接字层 (SSL)），那么还必须构建 OpenSSL 库。 请参阅第 41 页的『构建 OpenSSL 包』。

要点: OpenSSL 软件包的下载和重新分发受严格的进出口法规和开源许可条件的约束。 认真阅读限制和警告，然后决定是否下载此程序包。

关于此任务

遵循第 41 页的『构建 OpenSSL 包』中的指示信息以下载并构建 OpenSSL 库。 您必须构建 OpenSSL 才能构建安全版本的 C MQTT 客户机库。 您无需 OpenSSL 就可构建不安全版本的 MQTT 库。 这些步骤包含针对 iOS 和 Windows 构建库的示例。

通过将 C 开发库工具和 MQTT 软件开发工具箱 (SDK) 下载到构建平台上来构建 C MQTT 客户机库。 编写 makefile，结合第 28 页的『针对不同平台的 MQTT 构建选项』中记录的选项，为目标平台构建库。 以下提供了构建并运行 makefile 的特定于平台的步骤：

- iOS 第 30 页的『在 Apple Mac 上构建 C MQTT 客户机库以用于 iOS 设备』
- Windows 第 35 页的『在 Windows 上构建 MQTT 库』

过程

1. 将 C 开发环境安装在您要在其上进行构建的平台上。

   本主题中的示例内的 makefile 以下列工具为目标：

   - iOS 对于 iOS，在具有 OS X 10.8.2 的 Apple Mac 上，使用来自 Xcode 的 iOS 开发工具。
   - Linux 对于 Linux：来自 Red Hat Enterprise Linux V6.2 的 gcc V4.4.6。

     glibc C 库的最低受支持级别是 2.12，Linux 内核的最低受支持级别是 2.6.32。
   - Windows 对于 Microsoft Windows：Visual Studio V10.0。
2. 下载 移动消息传递和 M2M 客户机包 并安装 MQTT SDK。

   不存在安装程序，您只需展开下载的文件。

   a. 下载 移动消息传递和 M2M 客户机包。

   b. 创建将在其中安装 SDK 的文件夹。

      您可能希望将此文件夹命名为 MQTT。 此文件夹的路径在此被称为 *sdkroot*。

   c. 将压缩的 移动消息传递和 M2M 客户机包 文件内容展开到 *sdkroot* 中。 此展开操作将创建以 *sdkroot*\SDK 开头的目录树。
3. 展开 C MQTT 客户机库的源代码。

   源代码压缩文件为 *sdkroot*\SDK\clients\c\source.zip。
4. 可选： 构建 OpenSSL。

   请参阅第 41 页的『构建 OpenSSL 包』。
5. 构建 C MQTT 客户机库。

   在第 28 页的『针对不同平台的 MQTT 构建选项』中列出了用于构建库的命令和选项。

   执行以下示例中的步骤编写 makefile 来为目标平台构建 C MQTT 客户机库。

   - 第 30 页的『在 Apple Mac 上构建 C MQTT 客户机库以用于 iOS 设备』
   - 第 35 页的『在 Windows 上构建 MQTT 库』

针对不同平台的 MQTT 构建选项

下表列出了用于在各种平台上构建 C MQTT 客户机库的编译器和构建选项。

表 2: 针对不同平台的 *MQTT* 构建选项

平台	Compiler	Compiler Options	Linker Options	Extra Options
AIX	gcc	-fPIC -Os -Wall -DREVERSED -I *MQTTCLIENT_DIR*	-Wl,-G	
Linux s390x			-shared -Wl,-soname, libmqttv3c.so	-m64
Linux x86-64		-fPIC -Os -Wall -I *MQTTCLIENT_DIR*		
Linux x86-32				-m32
Linux ARM (glibc)	arm-linux-gcc			
Linux ARM (uclibc)	arm-unknown-linux -uclibcgnueabi-gcc			
Windows 32 位	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)	
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			

表 2: 针对不同平台的 *MQTT* 构建选项 (继续)

平台	Compiler	Compiler Options	Linker Options	Extra Options
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* 上构建 *C MQTT* 客户机库以用于 *iOS* 设备

执行以下步骤编写 makefile 以在 Apple Mac 上构建 C MQTT 客户机库，以便随后用于 iOS 设备。

开始之前

1. 安装构建工具，在具有 OS X 10.8.2 或更高版本的 Apple Mac 上开发并运行 makefile。
2. 安装 Xcode 的命令行工具，其中包括 **make** 程序。 从 Xcode 下载命令行工具。

关于此任务

创建一个 makefile，此文件用于为运行 ARMv7 或 ARMv7s 处理器的 iPhone 或 iPad 以及在 i386 64 位处理器上运行的 iPhone 仿真器构建 C MQTT 客户机库。 请参阅 List of iOS devices。

提示: 第 34 页的『MQTTios.mak makefile 清单』列出了完整的 makefile。

1. 将该清单复制并粘贴到某个文件中。
2. 将目标之后每行的前导字符转换为制表符；请参阅步骤 第 32 页的『8』。
3. 使用该过程的步骤 第 33 页的『9』 中列出的命令运行 makefile。

过程

1. 下载并安装 iOS 开发工具。
   a. 以具有管理特权的用户标识身份登录。
   b. 检查 Apple Mac 是否为 V10.8.2 或更高版本。
   c. 转至 Web 站点 Xcode 以从 Mac 应用商店下载 Xcode 。
   d. 安装 Xcode、命令行环境和仿真器。

   如果 Mac 应用程序商店提供多个版本的仿真器，请选择与适合您应用程序的 iOS 级别兼容的版本。
2. 创建 makefile `MQTTios.mak`

添加序言：

```
# 在当前目录中生成构建输出。
# MQTTCLIENT_DIR 必须指向包含 MQTT 客户机源代码的基本目录。
# 缺省 MQTTCLIENT_DIR 是当前目录
# 缺省 TOOL_DIR 为 /Applications/Xcode.app/Contents/Developer/Platforms
# 缺省 OPENSSL_DIR 是 sdkroot/openssl ，相对于 sdkroot/sdk/clients/c/mqttv3c/src
# OPENSSL_DIR 必须指向包含 OpenSSL 构建的基本目录。
# 示例 :make -f MQTTios.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src all
```

3. 设置 MQTT 源代码的位置。

   在与 MQTT 源文件相同的目录中运行 makefile ，或者设置 MQTTCLIENT_DIR 命令行参数:

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

   将以下行添加到该 makefile：

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

   此示例将 VPATH 设置为 **make** 在其中搜索未显式标识的源文件的目录; 例如，构建中所需的所有头文件。
4. 可选： 设置 OpenSSL 库的位置。

   此步骤是构建 MQTT Client for C 库的 SSL 版本所必需的。

   将 OpenSSL 库的缺省路径设置为与您展开 MQTT SDK 时相同的目录。 否则，将 OPENSSL_DIR 设置为命令行参数。

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

   提示: *OpenSSL* 是包含所有 OpenSSL 子目录的 OpenSSL 目录。 您可能需要从展开位置移走目录树，因为它包含不必要的空父目录。
5. 设置开发工具目录。

   如果您将 Xcode 安装到其他位置中，请在命令行中设置 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. 选择构建每个 MQTT 库所需的所有源文件。 另外，设置要构建的 MQTT 库的名称和位置。

   将以下行添加到 makefile 以列出所有 MQTT 源文件:

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

   所列的源文件取决于您要构建同步库还是异步库，以及该库是否包含 SSL。

   添加以下一行或多行，具体取决于要构建的目标。 在 darwin_x86_64 目录中创建了共享库。

   - 同步，非安全：

     ```
     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
     ```

- 异步，非安全：

```
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
```

- 异步，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. 定义编译器和编译器选项。

   请参阅 不同平台的 MQTT 构建选项中显示的不同平台的选项。

   a) 将 Gnu 项目 C 和 C++ (**gcc**) 设置为编译器。

   选择三个交叉编译器来为不同设备和 iPhone 仿真器构建库：

```
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) 添加编译器选项。

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

   c) 添加 include 路径。

```
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. 定义构建目标。

   提示: 定义目标实现的每个连续行必须以制表符开头。

   a) 定义 **all** 目标。

   “all” 目标将构建所有库。

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

   如果将其列在首位，就会成为缺省目标。

   a) 构建同步非安全库 `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 $@
```

   语句 `rm *.o` 将删除为每个库创建的所有对象文件。 `lipo` 将所有三个库并置到一个文件中。

   b) 构建异步非安全库 `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 $@
```

语句 `rm *.o` 将删除为每个库创建的所有对象文件。 `lipo` 将所有三个库并置到一个文件中。

c) 构建同步安全库 `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 $@
```

语句 `rm *.o` 将删除为每个库创建的所有对象文件。 `lipo` 将所有三个库并置到一个文件中。

d) 构建异步安全库 `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 $@
```

语句 `rm *.o` 将删除为每个库创建的所有对象文件。 `lipo` 将所有三个库并置到一个文件中。

e) 定义 **clean** 目标。

“clean” 目标将除去 makefile 生成的所有文件和目录

```
。假: 干净
clean:
    -rm -f *.obj
    -rm -f -r darwin_x86_64
```

9. 运行 makefile。

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

结果

在 *sdkroot/sdk/clients/c/mqttv3c/src*/darwin_x86_64 目录中创建了以下文件。

```
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 makefile 清单

```
# 在当前目录中生成构建输出。
# MQTTCLIENT_DIR 必须指向包含 MQTT 客户机源代码的基本目录。
# 缺省 MQTTCLIENT_DIR 是当前目录
# 缺省 TOOL_DIR 为 /Applications/Xcode.app/Contents/Developer/Platforms
# 缺省 OPENSSL_DIR 是 sdkroot/openssl ，相对于 sdkroot/sdk/clients/c/mqttv3c/src
# OPENSSL_DIR 必须指向包含 OpenSSL 构建的基本目录。
# 示例 :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 $@

。假: 干净
clean:
    -rm -f *.obj
    -rm -f -r darwin_x86_64
```

Windows 在 *Windows* 上构建 *MQTT* 库

执行以下步骤以编写 makefile，以用于在 Windows 上构建 C MQTT 客户机库。

开始之前

1. 如果需要，请在构建工作站上安装与为 Gnu make 编写的 makefile 兼容的 **Make** 版本; 否则，下载 Gnu make 并进行构建。 请参阅 Gnu Make。 Make for WindowsWeb 站点提供了 **Make** for Windows 的可安装版本。
2. 您还需要适用于 Windows 的 Linux 命令以使用 makefile 示例中的 clean 目标。 您可以从 Web 站点 (例如 Cygwin) 获取 Windows 的 Linux 命令。

关于此任务

创建用于为 Windows 32 位构建 C MQTT 客户机库的 makefile。

提示: 第 39 页的『MQTTwin.mak makefile 清单』列出了完整的 makefile。

1. 将该清单复制并粘贴到某个文件中。
2. 将目标之后每行的前导字符转换为制表符；请参阅步骤 第 38 页的『7』。
3. 使用该过程的步骤 第 39 页的『9』 中列出的命令运行 makefile。

过程

1. 创建 makefile MQTTwin.mak

   添加序言:

   ```
   # 在当前目录中生成构建输出。
   # MQTTCLIENT_DIR 必须指向包含 MQTT 客户机源代码的基本目录。
   # 缺省 MQTTCLIENT_DIR 是当前目录
   # 缺省 OPENSSL_DIR 是 sdkroot\openSSL，相对于 sdkroot\sdk\clients\c\mqttv3c\src
   # OPENSSL_DIR 必须指向包含 OpenSSL 构建的基本目录。
   # 示例 :make -f MQTTwin.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src
   # 设置构建环境，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. 设置 MQTT 源代码的位置。

   在与 MQTT 源文件相同的目录中运行 makefile，或者设置 MQTTCLIENT_DIR 命令行参数:

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

   将以下行添加到该 makefile:

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

   此示例将 VPATH 设置为 **make** 在其中搜索未显式标识的源文件的目录; 例如，构建中所需的所有头文件。

3. 可选： 设置 OpenSSL 库的位置。

   此步骤是构建 MQTT Client for C 库的 SSL 版本所必需的。

   将 OpenSSL 库的缺省路径设置为与您展开 MQTT SDK 时相同的目录。 否则，将 OPENSSL_DIR 设置为命令行参数。

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

   提示: *OpenSSL* 是包含所有 OpenSSL 子目录的 OpenSSL 目录。 您可能需要从展开位置移走目录树，因为它包含不必要的空父目录。

4. 选择构建每个 MQTT 库所需的所有源文件。 另外，设置要构建的 MQTT 库的名称和位置。

   将以下行添加到 makefile 以列出所有 MQTT 源文件:

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

   所列的源文件取决于您要构建同步库还是异步库，以及该库是否包含 SSL。

   添加以下一行或多行，具体取决于要构建的目标。 在 windows_ia32 目录中创建了共享库和清单。

   - 同步，非安全:

     ```
     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
```

- 异步，非安全：

```
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
```

- 异步，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. 定义编译器和编译器选项。

   请参阅 不同平台的 MQTT 构建选项中显示的不同平台的选项。

   a) 将 Microsoft Visual C++ 设置为编译器。

```
CC = cl
```

   b) 添加预处理器选项。

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

   c) 添加编译器选项。

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

   d) 添加 include 路径。

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

   e) 可选： 如果要构建安全库，请添加预处理器选项。

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

   f) 可选： 如果您要构建安全库，请添加 OpenSSL 头文件。

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

   提示: 头文件位于 `${OPENSSL_DIR}/inc32/openssl` 中，但 `ssl.h` 文件包含在 “`openssl/ssl.h`”中。

6. 设置链接程序和链接程序选项。

   a) 将 Microsoft Visual C++ 设置为链接程序。

```
LD = link
```

   b) 添加链接程序选项。

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

   c) 添加库路径。

```
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) 添加中间输出文件。

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

e) 可选： 如果您要构建安全库，请添加 OpenSSL 库。

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

f) 可选： 如果您要构建安全库，请添加 OpenSSL 库路径。

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

7. 定义四个构建目标。

a) 定义 **all** 目标。

提示: 定义目标实现的每个连续行必须以制表符开头。

“all” 目标将构建所有库。

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

b) 构建同步非安全库 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}
```

-rm ${CURDIR}/MQTTAsync.obj 语句删除为较早目标创建的任何 MQTTAsync.obj。MQTTAsync.obj 和 MQTTClient.obj 是互斥的。

c) 构建异步非安全库 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}
```

-rm ${CURDIR}/MQTTClient.obj 语句删除为较早目标创建的任何 MQTTClient.obj。MQTTAsync.obj 和 MQTTClient.obj 是互斥的。

d) 构建同步安全库 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}
```

-rm ${CURDIR}/MQTTAsync.obj 语句删除为较早目标创建的任何 MQTTAsync.obj。MQTTAsync.obj 和 MQTTClient.obj 是互斥的。

e) 构建异步安全库 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}
```

`-rm $(CURDIR}/MQTTClient.obj` 语句删除为较早目标创建的任何 `MQTTClient.obj`。`MQTTAsync.obj` 和 `MQTTClient.obj` 是互斥的。

f) 定义 **clean** 目标。

“clean” 目标将除去 makefile 生成的所有文件和目录

```
。假: 干净
clean:
    -rm -f *.obj
    -rm -f -r windows_ia32
```

8. 设置用于运行 makefile 的 Windows 路径。

   设置以斜体表示的部分，以便与自己的安装匹配。

   a) 设置 Microsoft Visual Studio 环境。

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

   b) 设置 `Path` 变量以包含 make 程序和 Linux 命令环境。

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

9. 运行 makefile。

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

   提示: 文件分隔符必须是正斜杠，而不是反斜杠。

结果

在 *sdkroot**SDK**clients**c**mqttv3c**src*\windows_ia32 目录中创建了以下文件。

```
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
mqttv3c.lib
mqttv3c.map
mqttv3cs.dll
mqttv3cs.dll.manifest
mqttv3cs.exp
mqttv3cs.lib
mqttv3cs.map
```

MQTTwin.mak makefile 清单

```
# 在当前目录中生成构建输出。
# MQTTCLIENT_DIR 必须指向包含 MQTT 客户机源代码的基本目录。
# 缺省 MQTTCLIENT_DIR 是当前目录
# 缺省 OPENSSL_DIR 是 sdkroot\openSSL，相对于 sdkroot\sdk\clients\c\mqttv3c\src
# OPENSSL_DIR 必须指向包含 OpenSSL 构建的基本目录。
# 示例 :make -f MQTTwin.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src
# 设置构建环境，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 = 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: ${MQTTDLL}\; 2
MQTTLIB_S = mqttv3cs
MQTTDLL_S = windows_ia32/${MQTTLIB_S} .dll
SOURCE_FILES_S = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c , ${ALL_SOURCE_FILES}}
MANIFEST_S = mt -manifest ${MQTTDLL_S}.manifest -outputresource: ${MQTTDLL_S}\; 2
MQTTLIB_A = mqttv3a
MQTTDLL_A = windows_ia32/${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: ${MQTTDLL_S}\; 2
MQTTLIB_AS = mqttv3as
MQTTDLL_AS = windows_ia32/${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

# 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}

。假: 干净
clean:
```

```
    -rm -f *.obj
    -rm -f -r windows_ia32
```

构建 *OpenSSL* 包

在为 C， `mqttv3cs` 和 `mqttv3as` 构建安全 MQTT 客户机库之前，请先构建 OpenSSL 软件包。 该构建将创建要构建安全版本的 C MQTT 客户机库所必需的库以及 OpenSSL 证书管理工具。

开始之前

1. iOS iOS makefile 定制针对运行 iOS6 的目标设备。 对于不同版本的 iOS，定制可能会有所不同。
2. Windows Windows makefile 定制适用于 32 位 Windows。

关于此任务

下载并安装 OpenSSL 包以及所有必备软件。 定制 OpenSSL makefile，然后为目标平台构建 OpenSSL 库。 在 Windows 和 Linux 上，make 还会构建 OpenSSL 密钥创建和管理工具。

过程

1. 安装 OpenSSL 包。
   a) 从 OpenSSL 下载 OpenSSL 包。

   要点: OpenSSL 软件包的下载和重新分发受严格的进出口法规和开源许可条件的约束。 认真阅读限制和警告，然后决定是否下载此程序包。

   b) 将压缩文件内容解压缩到 *`sdkroot`* 中。

   在 OpenSSL 站点上的 **News** 选项卡下查找最新包的下载位置。 此包压缩为扩展名为 `tar.gz` 的 tar 文件。 解压缩时，此包会创建顶级文件夹 `openssl`*`version`*，例如，*`openssl-1.0.1c`*。 这些示例将文件夹的路径称为 %openssl% (在 Windows 上) 和 $openssl (在 iOS 上); 例如，在 Windows 上，%openssl% 是 *`sdkroot`*`\openssl-1.0.1c`。

   提示: 检查通过解压缩 OpenSSL 包所创建的目录路径。 某些包具有重复级别的 `openssl`*`version`* 文件夹。

2. Windows
   可选： 在 Windows 上，下载并安装 perl。 请参阅 perl.org。

   对于本示例，已从 ActivePerl 下载 下载 perl。

3. iOS
   可选： 在 iOS 上，再创建三个目录。

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

   对于 iOS，您必须为三个不同的硬件平台构建 OpenSSL 包。

4. 生成 OpenSSL makefile，以便为您的硬件和操作系统构建 OpenSSL 包。
   a) 在 %openssl% 或 $openssl 目录中打开命令窗口。
   b) 以适当的参数运行 **Configure** perl 命令。

   - 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，为不同的 Apple 设备定制生成的 OpenSSL makefile。

a) 为生成的 makefile $openssl/Makefile 创建三个副本

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

b) 更改每个 makefile 中的 “CC=gcc” 语句。

CC=gcc 语句位于第 62 行或其附近。 将其更改为以下命令:

$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
```

c) 更改每个 makefile 中的 “CFLAG=...” 语句。

该语句位于第 63 行或其附近（我们将其分为三行以便于阅读）：

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

所显示的 SDK 的位置取决于您的 Xcode 安装选择。 这些 SDK 的版本取决于要为其构建 makefile 的操作系统的级别。

iPhone 仿真器

iPhone 仿真器 makefile 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 makefile 是 $openssl/Makefile_arm7 和 $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 -DDSO_DLFCN -DHAVE_DLFCN_H -DTERMIOS
-O3 -fomit-frame-pointer -Wall
```

6. 运行生成的 makefile。

- 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 $sslarm7s/libcrypto.a
mv $openssl/libssl.a $sslarm7s/libssl.a
make clean
make -f $openssl/Makefile_i386
mv $openssl/libcrypto.a $ssli386/libcrypto.a
mv $openssl/libssl.a $ssli386/libssl.a
```

结果

此构建将生成共享库、库和头文件（这些是构建安全版本的 C MQTT 客户机库所必需的项）。

iOS 上的 C MQTT 客户机入门

了解如何使 iOS 应用程序与 MQTT 服务器交换消息。 为了在 iOS 设备（即 iPhone 和 iPad）上予以使用，您必须从作为 MQTT 软件开发包的一部分提供的源代码构建 C MQTT 客户机库。

开始之前

1. 链接到 iOS Dev Center，并了解如何为 iOS 开发应用程序。
2. 获取具有 OS X 10.8.2 或更高版本的 Apple Mac 以运行 Xcode 集成开发环境 (IDE)。
3. （可选）在 Windows 或 Linux 上配置 C 开发环境。 先在 Windows 或 Linux 上构建并运行 MQTT 客户机样本 C 应用程序对接下来开发 MQTT iOS 应用程序很有用。 或者，研究样本源代码，而不构建样本。
4. 有关受支持的和参考 C MQTT 客户机平台，请参阅 IBM 移动消息传递和 M2M 客户机包的系统需求。
5. 如果客户机与服务器之间存在防火墙，请检查它是否未阻止 MQTT 流量。

关于此任务

该过程指导您完成以下步骤：

1. 通过研究、构建和运行 MQTT 客户机样本应用程序以及 C MQTT 客户机库来了解 MQTT 编程。
2. 在 Apple Mac 上安装针对 iOS 的 Xcode 开发环境。
3. 执行任务 第 27 页的『构建 C MQTT 客户机库』 为 iOS 设备构建 MQTT Client for C 库。

过程

1. 选择可以将客户机应用程序连接到的 MQTT 服务器。

   服务器必须支持 MQTT version 3.1 协议。 来自 IBM 的所有 MQTT 服务器都执行此操作，包括 IBM WebSphere MQ 和 IBM MessageSight。 请参阅第 122 页的『MQTT 服务器入门』。
2. 下载 移动消息传递和 M2M 客户机包 并安装 MQTT SDK。

   不存在安装程序，您只需展开下载的文件。

   a. 下载 移动消息传递和 M2M 客户机包。

   b. 创建将在其中安装 SDK 的文件夹。

      您可能希望将此文件夹命名为 MQTT。 此文件夹的路径在此被称为 *sdkroot*。

   c. 将压缩的 移动消息传递和 M2M 客户机包 文件内容展开到 *sdkroot* 中。 此展开操作将创建以 *sdkroot*\SDK 开头的目录树。
3. 可选： 通过研究 MQTT 客户机样本 C 应用程序来熟悉 MQTT API。

   a) 为 Windows 或 Linux 构建同步 MQTT 客户机样本 C 应用程序 `MQTTV3sample.c`。 请参阅第 23 页的『C MQTT 客户机入门』。

b) 连接到 MQTT version 3 服务器，然后在服务器上发布和预订主题。

c) 研究源代码和 MQTT API 文档。 有关 MQTT 客户机库的客户机 API 文档的链接，请参阅 MQTT 客户机编程参考。

通过研究同步样本，以了解如何创建和恢复 MQTT 客户机以及如何发布和预订 MQTT 主题。 同步样本比异步样本简单一些。 如果您以前没有从事过 MQTT 编程，可编写一个同步 MQTT 程序以熟悉 MQTT 编程模型和 API。

d) 在 Windows 或 Linux 上为 C 构建异步 MQTT 客户机库。 请参阅第 27 页的『构建 C MQTT 客户机库』。

e) 构建并运行异步 MQTT 客户机样本发布和预订 C 应用程序。

f) 研究 MQTT 客户机样本异步 C 应用程序源代码和 MQTT 参考文档。

您必须使用异步接口为移动设备编写 MQTT 应用程序。 与针对同步接口编写的应用程序相比，调用异步接口的应用程序如果编写得当，响应速度可以更快，而电池寿命可以更长。

异步接口具有两种异步等级：

i) 第一种等级是在 MQTT 客户机库等待来自服务器的发布期间将应用程序取消阻塞。

ii) 第二种等级是在客户机库连接到服务器、创建预订并公布发布期间将应用程序取消阻塞。

4. 下载并安装 iOS 开发工具。
   a. 以具有管理特权的用户标识身份登录。
   b. 检查 Apple Mac 是否为 V10.8.2 或更高版本。
   c. 转至 Web 站点 Xcode 以从 Mac 应用商店下载 Xcode 。
   d. 安装 Xcode、命令行环境和仿真器。

   如果 Mac 应用程序商店提供多个版本的仿真器，请选择与适合您应用程序的 iOS 级别兼容的版本。
5. 在 iOS 上构建 C MQTT 客户机库。 请参阅第 27 页的『构建 C MQTT 客户机库』。

下一步做什么

1. 验证您构建的 C MQTT 客户机库：
   a. 使用 Xcode 开发环境来编译异步 MQTT 客户机样本 C 应用程序，并链接到不安全的异步 C MQTT 客户机库。
   b. 使用 Xcode 开发环境在 iOS 设备上运行异步 MQTT 客户机样本 C 应用程序。 将样本连接到所配置的 MQTT version 3 服务器；请参阅第 125 页的『从命令行配置 MQTT 服务』。
2. 为 iOS 创建 MQTT 客户机 C 应用程序。 异步 C 应用程序的编码示例可能会很有帮助。 示例包括 *sdkroot*\SDK\clients\c\samples 中的 MQTTV3ASample.c 和 MQTTV3ASSample.c 。作为练习，请从实施 MQTT 发布/预订样本开始。

   提示: 要了解应用程序可能具有的外观和功能，请查看 MQTT 客户机样本 C 应用程序的截屏。 请参阅第 16 页的『Java 的 MQTT 客户机 on Android 入门』。

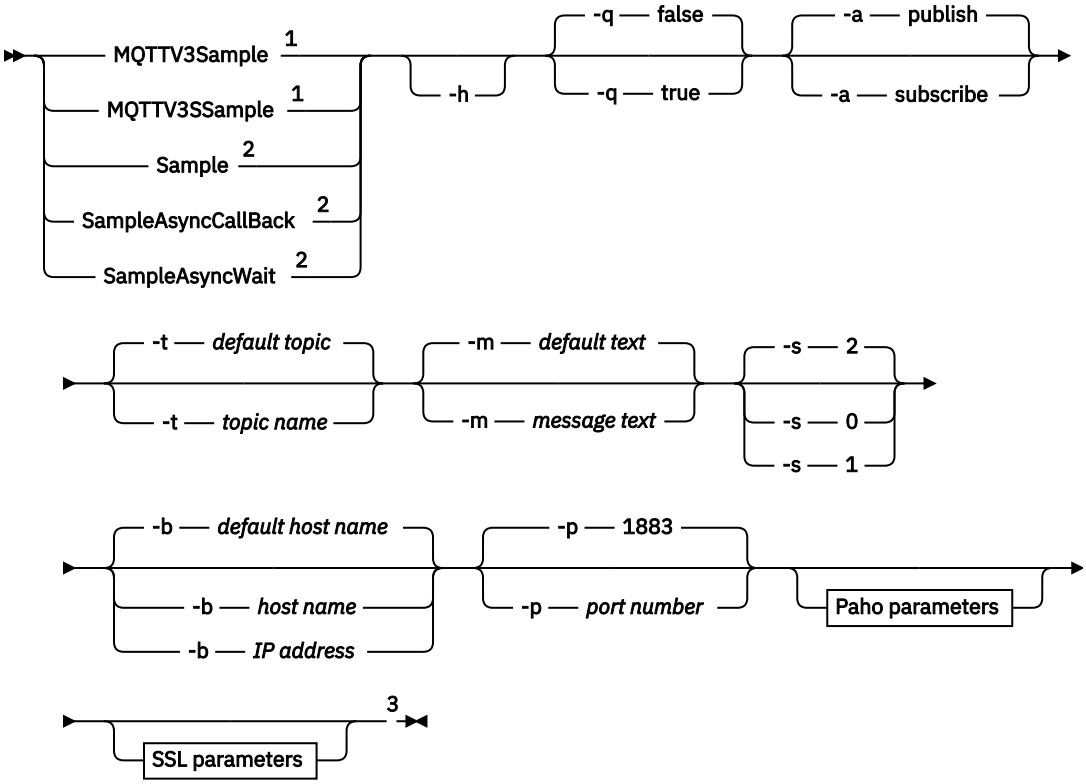
MQTT 命令行样本程序

MQTT 命令行样本程序的语法和参数。

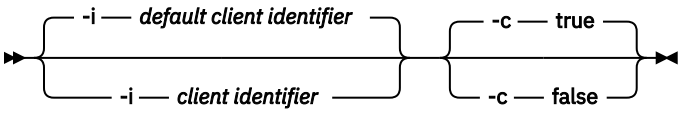
用途

发布和预订主题。

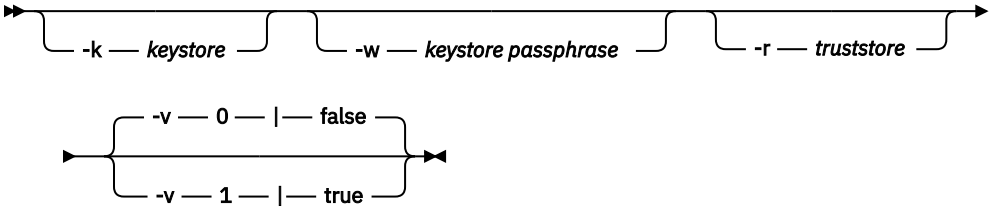
Syntax



Paho parameters



SSL parameters



注:

[1] IBM WebSphere MQ sample
[2] Paho sample
[3] *Not* `MQTTV3Sample`.

参数

-h
打印此帮助文本并退出

-q
设置静默方式，而不是使用缺省方式 false。

-a publish|subscribe
将操作设置为 `publish` 或 `subscribe`，而不是执行缺省操作（发布）。

-t ***topic name***
发布或预订 *topic name*，而不是发布或预订缺省主题。 缺省主题如下：

Paho 样本

发布
Sample/Java/v3

预订
Sample/#

IBM WebSphere MQ 样本

发布
MQTTV3Sample/Java/v3 或 MQTTV3Sample/C/v3

预订
MQTTV3Sample/#

-m ***message text***
发布 *message text*，而不是发送缺省文本。 缺省文本为 “Message from MQTTv3 C client”或 “Message from MQTTv3 Java client”

-s 0|1|2
设置服务质量 (QoS)，而不是使用缺省 QoS (2)。

-b ***host name***
连接到 *host name* 或 IP 地址，而不是连接到缺省主机名。 Paho 样本的缺省主机名是 m2m.eclipse.org。 对于 IBM WebSphere MQ 样本，它是 localhost。

-p ***port number***
使用端口 *port number*，而不是使用缺省端口 1883。

Paho 参数

-i ***client identifier***
将客户机标识设置为 *client identifier*. 缺省客户机标识为 SampleJavaV3_"+action，其中 action 是 publish 或 subscribe。

-c true|false
设置 clean 会话标志。 缺省值是 true：预订不是持久预订。

SSL 参数

-k ***keystore***
设置包含专用密钥（向 *keystore* 标识客户机）的密钥库的路径。对于 C 程序样本，该库是保密性增强邮件 (PEM) 文件。 对于 Java 样本，它是 Java 密钥库 (JKS)。

-w ***keystore passphrase***
将用于授权客户机访问密钥库的口令设置为 *keystore passphrase*。

-r ***truststore***
将包含客户机信任的 MQTT 服务器公用密钥的密钥库的路径设置为 *truststore*。 密钥库是隐私增强邮件 (PEM) 文件。 对于 C 程序样本，该库是保密性增强邮件 (PEM) 文件。 对于 Java 样本，它是 Java 密钥库 (JKS)。

-v 0|false|1|true
将验证选项设置为 1|true 以要求服务器证书。 缺省值为 0|false: 未检查服务器证书。 始终会加密 SSL 通道。

将选项设置为 0|1 (对于 C 程序) 和 true|false (对于 Java 程序)。

相关任务
第 11 页的『Java MQTT 客户机入门』
使用 IBM MessageSight 或 IBM WebSphere MQ 作为 MQTT 服务器启动并运行 MQTT Client for Java 样本应用程序。 样本应用程序使用来自 IBM 的 MQTT 软件开发工具箱 (SDK) 中的客户机库。
SampleAsyncCallBack 样本应用程序是用于为 Android 和其他事件驱动的操作系统编写 MQTT 应用程序的模型。

第 23 页的『C MQTT 客户机入门』
在任何可编译 C 源代码的平台上启动并运行样本 C MQTT 客户机。 验证是否可以使用 IBM MessageSight 或 IBM WebSphere MQ 作为 MQTT 服务器运行样本 MQTT Client for C。

第 27 页的『构建 C MQTT 客户机库』
执行以下步骤以构建 C MQTT 客户机库。 本主题包含针对许多平台的编译和链接开关，以及在 iOS 和 Windows 上构建库的示例。

MQTT 安全性

MQTT 安全性具有三个基本概念：身份、认证和权限。 身份是对要为其授权或赋予权限的客户机命名。 认证是提供客户机的身份；而权限是管理赋予客户机的权利。

试验安全样本

- 第 49 页的『构建和运行安全 MQTT 客户机样本 Java 应用程序』
- 第 57 页的『通过 SSL 在 Android 上连接 MQTT 客户机样本 Java 应用程序』
- 第 66 页的『向 JAAS 认证 MQTT 客户机 Java 应用程序』
- V 7.5.0.1 第 70 页的『通过 SSL 连接 JavaScript 的 MQTT 消息传递客户机 和 WebSockets』
- 第 78 页的『构建和运行安全 MQTT 客户机样本 C 应用程序』

身份

通过客户机标识、用户标识或公用数字证书来识别 MQTT 客户机。 这两个属性中有一个用于定义客户机身份。 MQTT 服务器使用 SSL 协议对客户机发送的证书进行认证，或使用客户机设置的密码对客户机身份进行认证。 该服务器根据客户机的身份来控制客户机可以访问的资源。

MQTT 服务器使用其 IP 地址和其数据证书向客户机标识自己。 MQTT 客户机使用 SSL 协议对服务器发送的证书进行认证。 某些情况下，它使用服务器的 DNS 名称来验证向其发送证书的服务器是否注册为证书持有者。

通过以下某种方式设置客户机的身份：

客户机标识
`MqttClient` 类（C 中的 `MQTTClient_create` 或 `MQTTAsync_create`）设置客户机标识。 调用类构造函数以将客户机标识作为参数进行设置，或者返回随机生成的客户机标识。客户机标识必须在连接到服务器的所有客户机中是唯一的，且不能与服务器上的队列管理器名称相同。 所有客户机都必须具有客户机标识，即使它未用于身份检查也是如此。 请参阅第 115 页的『客户机标识』。

用户标识
`MqttClient` 类（C 中的 `MQTTClient_create` 或 `MQTTAsync_create`）将客户机用户标识设置为 `MqttConnectOptions`（C 中的 `MqttClient_ConnectOptions`）的属性。 对客户机来说，用户标识不必唯一。

客户机数字证书
客户机数字证书存储在客户机密钥库中。 密钥库位置取决于客户机：

- **Java**

  通过调用 `MqttConnectOptions` 的 `setSSLProperties` 方法并传递密钥库属性来设置客户机密钥库的位置和属性。 请参阅对 Example.java 的 SSL 修改。 **keytool** 工具管理 Java 密钥和密钥库。

- **C**

  `MQTTClient_create` 或 `MQTTAsync_create` 将密钥库属性设置为 `MQTTClient_SSLOptions` `ssl_opts` 的属性。 **openSSL** 工具创建并管理 MQTT Client for C 访问的密钥和密钥库。

- **Android**

  通过**设置** > **安全性**菜单管理 Android 设备密钥库。 通过 SD 卡装入新的证书。

通过在服务器密钥库中存储专用密钥来设置服务器的身份：

IBM WebSphere MQ

MQTT 服务器密钥库是客户机连接到的遥测通道的属性。

使用 IBM WebSphere MQ Explorer 或 **DEFINE CHANNEL** 命令设置密钥库位置和属性; 请参阅 DEFINE CHANNEL (MQTT)。 多个通道可以共享一个密钥库。

认证

MQTT 客户机可以对其连接到的 MQTT 服务器进行认证，服务器可以对连接到它的客户机进行认证。

客户机通过 SSL 协议对服务器进行认证。 MQTT 服务器通过 SSL 协议和/或密码对客户机进行认证。

如果客户机对服务器进行认证，但服务器不对客户机进行认证，那么通常会将客户机视为匿名客户机。 通常通过 SSL 建立匿名客户机连接，然后使用通过 SSL 会话加密的密码对客户机进行认证。 由于证书分发和管理问题，通常使用密码而不是客户机证书对客户机进行认证。 在高价值设备（如 ATM 和“芯片密码”机）和定制设备（如智能电表）中，您会发现其中使用了客户机证书。

由客户机执行的服务器认证

MQTT 客户机通过使用 SSL 协议对服务器证书进行认证来验证其是否连接到了正确的服务器。 当您通过 HTTPS 协议浏览 Web 站点时，会经常使用这种形式的验证。

服务器会向客户机发送由认证中心签名的公用证书。 客户机使用认证中心的公用密钥来验证服务器证书上的认证中心签名。 它还会检查该证书是否是最新的。 这些检查能够确定证书是否有效。

认证中心证书（通常被称为根证书）存储在客户机信任库中:

- **Java**

  调用 `MqttConnectOptions` 的 `setSSLProperties` 方法并传递信任库属性以设置客户机信任库的位置和属性。 请参阅对 Example.java 的 SSL 修改。 使用 **keytool** 工具管理证书和信任库。

- **C**

  `MQTTClient_create` 或 `MQTTAsync_create` 将信任库属性设置为 `MQTTClient_SSLOptions` `ssl_opts` 的属性。 使用 **openSSL** 工具管理证书和信任库。

- **Android**

  通过**设置** > **安全性**菜单管理 Android 设备信任库。 通过 SD 卡装入新的根证书。

服务器执行的客户机认证

MQTT 服务器通过使用 SSL 协议对客户机证书进行认证，或通过使用密码对客户机身份进行认证，来验证其是否连接到了正确的客户机。

它通过客户机在 MQTT protocol 头中发送给服务器的密码来对客户机进行认证。 服务器可能会选择使用密码对客户机标识、用户标识或证书进行认证。 这取决于服务器。 通常，服务器会对用户标识进行认证。 通过已通过验证服务器来确保安全的 SSL 连接来验证密码，避免以明码形式发送密码。

- **IBM WebSphere MQ**

  IBM WebSphere MQ 使用 SSL 协议对客户机证书进行认证。 将根证书存储在 IBM WebSphere MQ Telemetry 密钥库中。 您只能将客户机证书作为相互 SSL 认证的一部分进行认证。 换而言之，您必须向客户机提供服务器公用证书，而向服务器提供客户机公用证书。

  IBM WebSphere MQ Telemetry 针对自己的专用证书和公用证书以及其他公用证书（如认证中心提供的根证书）使用相同的库。

  使用 IBM WebSphere MQ Explorer 或 **DEFINE CHANNEL** 命令设置密钥库位置和属性; 请参阅 DEFINE CHANNEL (MQTT)。 多个通道可以共享一个密钥库。

  IBM WebSphere MQ 通过调用 Java 认证和授权服务 (JAAS) 对客户机用户标识或客户机标识进行认证。

  在存储在 `jaas.config` 文件中的 `MQXRConfig` 配置节中配置 JAAS。 该文件存储在 IBM WebSphere MQ 数据路径中的 `qmgrs\`*`QmgrName`*`\mqxr` 目录中。

  通过编写 `JAASLoginModule` 的 `login` 方法来检查客户机的真实性。 请参阅第 104 页的『遥测通道 JAAS 配置』。

IBM WebSphere MQ Telemetry 向 JAASLoginModule.login 方法传递以下参数:

- 用户标识
- 密码
- 客户机标识
- 网络标识
- 通道名称
- ValidPrompts

Authorization

权限不是 MQTT protocol 的一部分。 它由 MQTT 服务器提供。 授予那些权限取决于服务器执行什么操作。 MQTT 服务器是发布/预订代理程序，而有用的 MQTT 授权规则控制可以连接到服务器的客户机，以及客户机可以发布或预订的主题。 如果 MQTT 客户机可以管理服务器，那么更多的授权规则将控制可以管理服务器不同方面的客户机。

可能的客户机数量非常巨大，因此为每个客户机分别授权是不可行的。 MQTT 服务器具有按照概要文件或组对客户机进行分组的方式。

来自访问和授权的视图点的客户机身份对 MQTT 客户机不是唯一的。 不同于具有该客户机标识的客户机的身份。 它们也可能相同，但通常不相同。 例如，您可能有一个在多个服务之间通用的用户名，其中一些服务在 “单点登录”中进行了协作。 企业范围的 MQTT 服务器很可能调用为不同应用程序提供通用标识和权限的授权服务。

IBM WebSphere MQ

IBM WebSphere MQ 具有可插入式授权服务。 在 Windows 和 Linux 上提供的缺省授权服务是对象权限管理器 (OAM)。 请参阅 在 UNIX， Linux 和 Windows 系统上使用 OAM 控制对对象的访问。 它将操作系统用户标识和组关联到针对 IBM WebSphere MQ 对象（如主题和队列）的操作。

您可以配置遥测通道来使用固定用户标识访问 IBM WebSphere MQ。 这是设置样本通道的方法。 或者，您可以使用 MQTT 客户机设置的用户标识来访问 IBM WebSphere MQ 。 授权 MQTT 客户机访问 WebSphere MQ 对象描述了设置 IBM WebSphere MQ Telemetry 以实现粗、中和细致的客户机访问控制的方法。

相关任务

第 78 页的『构建和运行安全 MQTT 客户机样本 C 应用程序』
您可以根据 Windows 示例在您可以为其编译 C 源代码的任何操作系统上启动并运行安全样本 C 应用程序。 验证您是否可以在 IBM MessageSight 或 IBM WebSphere MQ 作为 MQTT 服务器上运行样本 C 应用程序。

第 49 页的『构建和运行安全 MQTT 客户机样本 Java 应用程序』
根据 Windows 示例，您可以在 IBM MessageSight 或 IBM WebSphere MQ 作为 MQTT 服务器上启动并运行安全样本 Java 应用程序。 您可以在具有 JSE 1.5 或更高版本 (即 “Java 兼容”) 的任何平台上运行 Java 的 MQTT 客户机 应用程序

第 70 页的『通过 SSL 连接 JavaScript 的 MQTT 消息传递客户机 和 WebSockets』
通过将 JavaScript 的 MQTT 消息传递客户机 样本 HTML 页面与 SSL 和 WebSocket protocol 配合使用，将 Web 应用程序安全地连接到 IBM WebSphere MQ 。

第 57 页的『通过 SSL 在 Android 上连接 MQTT 客户机样本 Java 应用程序』
启动并运行通过 SSL 连接到 IBM WebSphere MQ 的样本 Android MQTT 客户机。

第 66 页的『向 JAAS 认证 MQTT 客户机 Java 应用程序』
了解如何使用 JAAS 来认证客户机。 完成此任务中的步骤以修改样本程序 JAASLoginModule.java ， 并配置 IBM WebSphere MQ 以向 JAAS 认证 MQTT 客户机 Java 应用程序。

构建和运行安全 MQTT 客户机样本 Java 应用程序

根据 Windows 示例，您可以在 IBM MessageSight 或 IBM WebSphere MQ 作为 MQTT 服务器上启动并运行安全样本 Java 应用程序。 您可以在具有 JSE 1.5 或更高版本 (即 “Java 兼容”) 的任何平台上运行 Java 的 MQTT 客户机 应用程序

开始之前

1. 您必须有权访问支持 MQTT protocol over SSL 的 MQTT version 3.1 服务器。
2. 如果客户机与服务器之间存在防火墙，请检查它是否未阻止 MQTT 流量。
3. 您可以在具有 JSE 1.5 或更高版本 (即 “Java 兼容”) 的任何平台上运行 Java 的 MQTT 客户机 应用程序. 请参阅 IBM 移动消息传递和 M2M 客户机包的系统需求。
4. SSL 通道必须已启动。

关于此任务

作为示例，本文说明如何从命令行编译和运行 Windows 上的安全 MQTT 客户机样本 Java 应用程序 。

使用认证中心签名的密钥或自签名的密钥保护 SSL 通道。

过程

1. 选择可以将客户机应用程序连接到的 MQTT 服务器。

   服务器必须支持基于 SSL 的 MQTT version 3.1 协议。 来自 IBM 的所有 MQTT 服务器都执行此操作，包括 IBM WebSphere MQ 和 IBM MessageSight。 请参阅第 122 页的『MQTT 服务器入门』。
2. 可选： 安装 V 7 或更高版本的 Java 开发包 (JDK)。

   V 7 才能运行 **keytool** 命令来认证证书。 如果将不对证书进行认证，那么就不需要 JDK V7。
3. 下载 移动消息传递和 M2M 客户机包 并安装 MQTT SDK。

   不存在安装程序，您只需展开下载的文件。

   a. 下载 移动消息传递和 M2M 客户机包。

   b. 创建将在其中安装 SDK 的文件夹。

      您可能希望将此文件夹命名为 MQTT。 此文件夹的路径在此被称为 *sdkroot*。

   c. 将压缩的 移动消息传递和 M2M 客户机包 文件内容展开到 *sdkroot* 中。 此展开操作将创建以 *sdkroot*\SDK 开头的目录树。
4. 创建并运行脚本以生成密钥对和证书，并将 IBM WebSphere MQ 配置为 MQTT 服务器。

   遵循 第 87 页的『生成密钥和证书』 中的步骤来创建和运行脚本。在第 52 页的『用于为 Windows 配置 SSL 证书的示例脚本』中也列出了这些脚本。
5. 检查 SSL 通道是否正在运行，以及是否为期望的设置。

   在 IBM WebSphere MQ 上，在命令窗口中输入以下命令:

   - 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. 创建用于构建和运行安全 MQTT 客户机样本 Java 应用程序的脚本。

   a) 创建并运行 `ssjavaclient.bat`，以测试受自签名证书保护的 SSL 通道。

   b) 创建并运行 `cajavaclient.bat`，以测试受认证中心签名证书保护的 SSL 通道。

用于运行 MQTT 安全 Java 客户机的脚本

在运行这些脚本之前先运行第 52 页的『用于为 Windows 配置 SSL 证书的示例脚本』中的脚本。

具有自签名证书的 MQTT 安全 Java 客户机。
使用通过运行 `sscerts.bat` 脚本所创建的自签名证书来运行此脚本。

```
@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
```

图 *14: ssjavaclient.bat*

运行具有认证中心签名的证书的 MQTT 安全 Java 客户机。

使用通过运行 cacerts.bat 脚本所创建的认证中心签名证书来运行此脚本。

```
@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
```

图 *15: cajavaclient.bat*

相关概念

第 47 页的『MQTT 安全性』

MQTT 安全性具有三个基本概念：身份、认证和权限。 身份是对要为其授权或赋予权限的客户机命名。 认证是提供客户机的身份；而权限是管理赋予客户机的权利。

相关任务

第 87 页的『生成密钥和证书』

按照本过程来为 Java 和 C 客户机（包括 Android 和 iOS 应用程序）以及 IBM WebSphere MQ 和 IBM MessageSight 服务器生成密钥和证书。

第 57 页的『通过 SSL 在 Android 上连接 MQTT 客户机样本 Java 应用程序』
启动并运行通过 SSL 连接到 IBM WebSphere MQ 的样本 Android MQTT 客户机。

第 66 页的『向 JAAS 认证 MQTT 客户机 Java 应用程序』
了解如何使用 JAAS 来认证客户机。 完成此任务中的步骤以修改样本程序 `JAASLoginModule.java`，并配置 IBM WebSphere MQ 以向 JAAS 认证 MQTT 客户机 Java 应用程序。

用于为 Windows 配置 SSL 证书的示例脚本

** **

此示例命令文件根据任务中各个步骤的描述来创建证书和证书库。 此外，本示例还将 MQTT 客户机队列管理器设置为使用服务器证书库。 此示例通过调用 IBM WebSphere MQ 随附的 `SampleMQM.bat` 脚本来删除并重新创建队列管理器。

initcert.bat

`initcert.bat` 设置证书的名称和路径以及 **keytool** 和 **openSSL** 命令所需的其他参数。 脚本中的注释描述了这些设置。

```
@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
```

```
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 脚本中的命令会删除 MQTT 客户机队列管理器，以确保服务器证书库未锁定，然后会删除样本安全脚本创建的所有密钥库和证书。

```
@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 %srvcertreq%
erase %srvcertcasigned%
erase %srvcertselfsigned%
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 脚本中的命令将创建用于专用认证中心、服务器和客户机的密钥对。

```
@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 脚本中的命令会将客户机和服务器自签名证书从其密钥库中导出，然后将服务器证书导入客户机信任库中，并将客户机证书导入到服务器密钥库中。 服务器没有信任库。 该命令将从客户机 JKS 信任库创建 PEM 格式的客户机信任库。

```
@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
@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

该脚本会将认证中心根证书导入专用密钥库。 需要使用 CA 根证书在根证书和签名证书之间创建密钥链。 cacerts.bat 脚本会将客户机和服务器证书请求从其密钥库中导出。 此脚本使用 cajkskeystore.jks 密钥库中专用认证中心的密钥来签署证书请求，然后将签署的证书导入到发出请求的同一密钥库。 该导入使用 CA 根证书创建证书链。 该脚本将从客户机 JKS 信任库创建 PEM 格式的客户机信任库。

```
@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:%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

该脚本将在证书目录中列出密钥库和证书。 然后，它将创建 MQTT 样本队列管理器并配置安全遥测通道。

```
@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%
V 7.5.0.1
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
```

通过 SSL 在 Android 上连接 MQTT 客户机样本 Java 应用程序

启动并运行通过 SSL 连接到 IBM WebSphere MQ 的样本 Android MQTT 客户机。

开始之前

本文章假定您至少在运行 Android API 级别 14 (ICS 4.0)。 更低级别已具有密钥库，但是只有系统应用程序才可以访问此密钥库。

1. 您必须有权访问支持 MQTT protocol over SSL 的 MQTT version 3.1 服务器。
2. 如果客户机与服务器之间存在防火墙，请检查它是否未阻止 MQTT 流量。
3. 如果您在早期 Android 设备上测试连接，那么可能需要 SD 卡才能将证书传输到该设备。
4. 如果您在虚拟 Android 设备上测试连接，请为此虚拟设备配置 SD 卡。
5. SSL 通道必须已启动。

关于此任务

完成本任务以通过 SSL 运行 Android 的 MQTT 客户机样本 Java 应用程序。 成功的 SSL 连接会在 Android 设备与 MQTT 服务器之间建立安全的加密通道。 将认证服务器的身份。

对于 Android，您可以使用 SSL 来认证服务器。 您还可以认证设备，尽管样本应用程序不支持此操作。 要认证设备，请使用 KeyChain API 或使用 JAAS 来认证 MQTT Android 应用程序所提供的客户机标识、客户机 IP 地址或用户名和密码。

您安装到 Android 信任库中的任何 X.509 证书都必须由认证中心签署。 在本示例中，您创建认证中心，而该中心签署您在 Android 设备中安装的证书。 许多根证书已预安装到 Android 设备中。

您必须在安装可信证书之前对 Android 设备创建锁。 该锁用于阻止某人在您不知情的情况下在设备上安装证书。

过程

1. 选择可以将客户机应用程序连接到的 MQTT 服务器。

   服务器必须支持基于 SSL 的 MQTT version 3.1 协议。 来自 IBM 的所有 MQTT 服务器都执行此操作，包括 IBM WebSphere MQ 和 IBM MessageSight。 请参阅第 122 页的『MQTT 服务器入门』。
2. 在不受保护的 MQTT 通道上运行 MQTT 客户机样本应用程序 for Android “MQTTExerciser”。 请参阅第 16 页的『Java 的 MQTT 客户机 on Android 入门』。

   您再次使用此应用程序以测试安全通道。

   如果您启动了 Android 虚拟设备，请使其保持运行。
3. 可选： 安装 V 7 或更高版本的 Java 开发包 (JDK)。

   V 7 才能运行 **keytool** 命令来认证证书。 如果将不对证书进行认证，那么就不需要 JDK V7。
4. 创建并运行脚本以生成密钥对和证书，并将 IBM WebSphere MQ 配置为 MQTT 服务器。

   遵循第 87 页的『生成密钥和证书』中的步骤来创建和运行脚本。在第 61 页的『用于为 Windows 配置 SSL 证书的示例脚本』中也列出了这些脚本。

   您需要认证中心公用证书和服务器密钥库。 您无需客户机证书或者任何 .pem 或 .p12 格式的证书。
5. 检查 SSL 通道是否正在运行，以及是否为期望的设置。

   在 IBM WebSphere MQ 上，在命令窗口中输入以下命令：

   - 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. 将认证中心证书安装到 Android 信任库中。

   本示例中的认证中心文件是 cacert.cer。

   a) 将该证书重命名为 cacert.crt。

   b) 将该证书复制到根内部存储器，或者复制到 SD 卡上。

      对于运行中的虚拟设备，请打开 Eclipse 或运行 Android 调试桥 (ADB) 以将该证书复制到此虚拟设备：

      Eclipse

      i) 运行 Eclipse，然后打开 DDMS 透视图。

      ii) 在主视图中，打开 **File Explorer** 窗口。

      iii) 打开 mnt/sdcard 目录。

      iv) 将 cacert.crt 文件拖到 mnt/sdcard 目录。

      ADB

      i) 打开命令窗口，并将其当前目录设置为 android 安装目录中的 android-sdk\platform-tools；例如， C:\Program Files\Android\android-sdk\platform-tools。

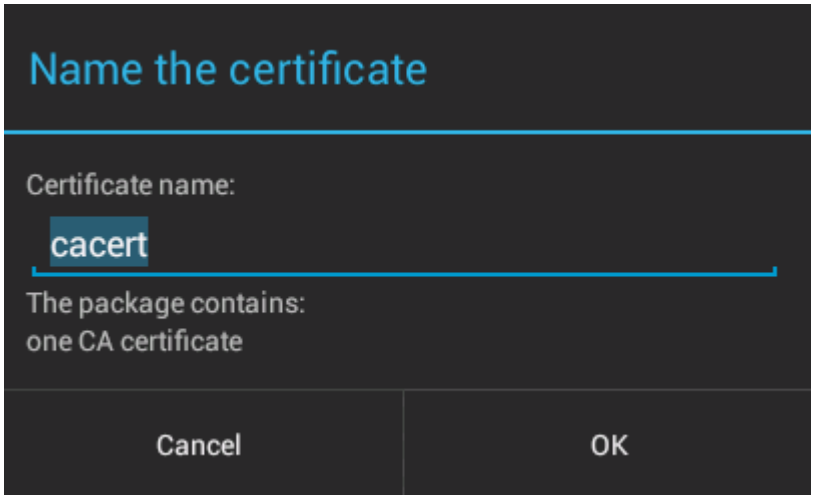
      ii) 将该证书复制到 mnt/sdcard 目录：

      ```
      adb push %cacert% /mnt/sdcard/cacert.crt
      ```
7. 将该证书安装到 Android 设备上的证书信任库中。

该证书必须具有值为 `Subject Type=CA` 的基本约束子句。

a) 将设备解锁，然后单击**窗口小部件**按钮。

b) 单击 **设置 > 安全性 > 凭证存储器 > 从 SD 卡安装**。

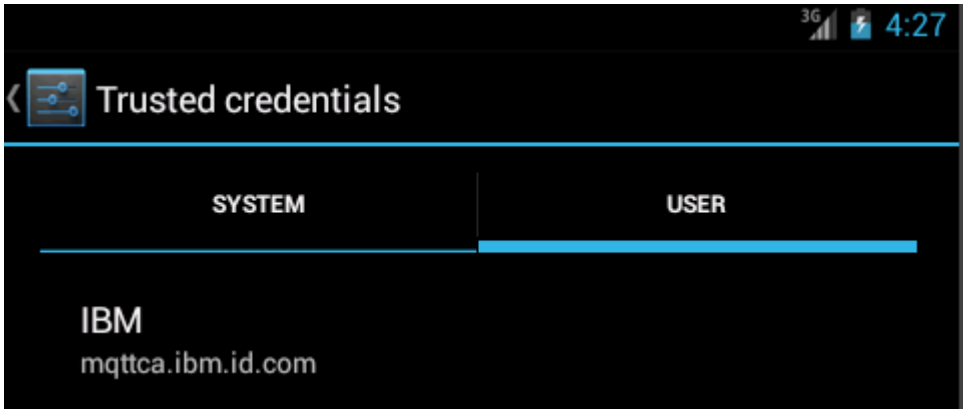


c) 确认证书文件名正确，然后单击**确定**。

注: 如果您尚未定义设备锁，那么现在 Android 会提示您设置锁定。

8. 确认该证书已安装在设备上。

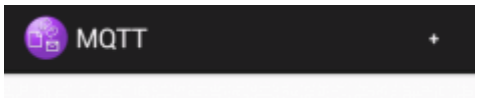
a) 单击 **可信凭证 > 用户**，然后等待几分钟左右，以便您的证书显示在用户证书列表中。



9. 重新运行 `MQTTExerciser` 应用程序，然后连接到您已为匿名 SSL 客户机配置的 MQTT 通道。

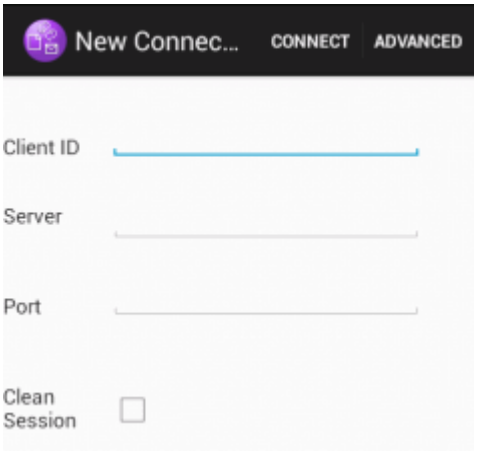
a) 打开 Android 的 MQTT 客户机样本 Java 应用程序。

此窗口在您的 Android 设备中打开:



b) 连接到 MQTT 服务器。

i) 单击 **+** 符号以打开新的 MQTT 连接。



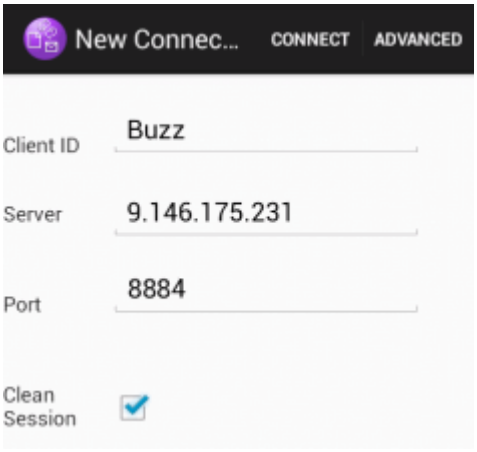
ii) 将任何唯一标识输入到**客户机标识**字段中。 请耐心，击键的速度可能会很慢。

iii) 在**服务器**字段中输入 MQTT 服务器的 IP 地址。

这是在第一个主要步骤中您选择的服务器。 IP 地址不得为 `127.0.0.1`。

iv) 输入 MQTT 连接的端口号。

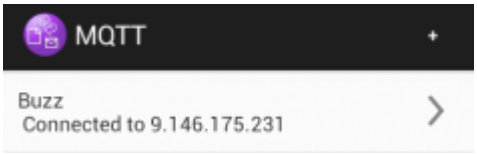
将端口号设置为 8884，这由示例脚本中的变量 `%sslportopt%` 所设置。 此端口号是您已通过在步骤 第 58 页的『4』 中运行样本脚本来为匿名 SSL 客户机配置的 MQTT 通道的端口号。



v) 单击 **Advanced** 选项卡并选择 **SSL** 选项。 单击**保存**。

vi) 单击**连接**。

如果连接成功，那么您将看到 “正在连接” 消息，后跟此窗口:



结果

`MQTTExerciser` 应用程序会耗费更长一点的时间来连接和交换消息，但是在其他方面与在非安全连接上进行连接没有行为差异。

相关概念

第 47 页的『MQTT 安全性』
MQTT 安全性具有三个基本概念：身份、认证和权限。 身份是对要为其授权或赋予权限的客户机命名。 认证是提供客户机的身份；而权限是管理赋予客户机的权利。

相关任务

第 87 页的『生成密钥和证书』
按照本过程来为 Java 和 C 客户机（包括 Android 和 iOS 应用程序）以及 IBM WebSphere MQ 和 IBM MessageSight 服务器生成密钥和证书。

第 49 页的『构建和运行安全 MQTT 客户机样本 Java 应用程序』
根据 Windows 示例，您可以在 IBM MessageSight 或 IBM WebSphere MQ 作为 MQTT 服务器上启动并运行安全样本 Java 应用程序。 您可以在具有 JSE 1.5 或更高版本 (即 “Java 兼容”) 的任何平台上运行 Java 的 MQTT 客户机 应用程序

第 66 页的『向 JAAS 认证 MQTT 客户机 Java 应用程序』
了解如何使用 JAAS 来认证客户机。 完成此任务中的步骤以修改样本程序 `JAASLoginModule.java` ，并配置 IBM WebSphere MQ 以向 JAAS 认证 MQTT 客户机 Java 应用程序。

用于为 Windows 配置 SSL 证书的示例脚本

** **

此示例命令文件根据任务中各个步骤的描述来创建证书和证书库。 此外，本示例还将 MQTT 客户机队列管理器设置为使用服务器证书库。 此示例通过调用 IBM WebSphere MQ 随附的 `SampleMQM.bat` 脚本来删除并重新创建队列管理器。

initcert.bat

`initcert.bat` 设置证书的名称和路径以及 **keytool** 和 **openSSL** 命令所需的其他参数。 脚本中的注释描述了这些设置。

```
@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
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 脚本中的命令会删除 MQTT 客户机队列管理器，以确保服务器证书库未锁定，然后会删除样本安全脚本创建的所有密钥库和证书。

```
@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 %srvcertreq%
erase %srvcertcasigned%
erase %srvcertselfsigned%
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 脚本中的命令将创建用于专用认证中心、服务器和客户机的密钥对。

```
@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 脚本中的命令会将客户机和服务器自签名证书从其密钥库中导出，然后将服务器证书导入客户机信任库中，并将客户机证书导入到服务器密钥库中。 服务器没有信任库。 该命令将从客户机 JKS 信任库创建 PEM 格式的客户机信任库。

```
@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
@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

该脚本会将认证中心根证书导入专用密钥库。 需要使用 CA 根证书在根证书和签名证书之间创建密钥链。 cacerts.bat 脚本会将客户机和服务器证书请求从其密钥库中导出。 此脚本使用 cajkskeystore.jks 密钥库中专用认证中心的密钥来签署证书请求，然后将签署的证书导入到发出请求的同一密钥库。 该导入使用 CA 根证书创建证书链。 该脚本将从客户机 JKS 信任库创建 PEM 格式的客户机信任库。

```
@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:%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
该脚本将在证书目录中列出密钥库和证书。 然后，它将创建 MQTT 样本队列管理器并配置安全遥测通道。

```
@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%
V 7.5.0.1
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
```

向 JAAS 认证 MQTT 客户机 Java 应用程序

了解如何使用 JAAS 来认证客户机。 完成此任务中的步骤以修改样本程序 JAASLoginModule.java，并配置 IBM WebSphere MQ 以向 JAAS 认证 MQTT 客户机 Java 应用程序。

开始之前

1. 您可以在具有 JSE 1.5 或更高版本 (即 “Java 兼容”) 的任何平台上运行 Java 的 MQTT 客户机 应用程序. 请参阅 IBM 移动消息传递和 M2M 客户机包的系统需求。
2. 如果客户机与服务器之间存在防火墙，请检查它是否未阻止 MQTT 流量。
3. 您必须有权访问 IBM WebSphere MQ 安装中的 MQXR JAASLoginModule 和 JAASPrincipal Java 样本。 样本位于路径 %MQ_FILE_PATH%\mqxr\samples. 中
4. 在 Windows 或 Linux 上完成这些步骤；示例取自 Windows。
5. 要完成步骤 第 66 页的『1』，您必须有权在 IBM WebSphere MQ 上创建 MQXR_SAMPLE_QM 队列管理器。

关于此任务

在该任务中，从 JAASLoginModule 版本输出 MQTT Sample 客户机标识参数。 写出客户机参数需要修改样本 JAASLoginModule 程序并将 IBM WebSphere MQ 配置为装入您的 JAASLoginModule 版本。

过程

1. 完成第 14 页的『从 Eclipse 编译并运行所有 MQTT 客户机样本 Java 应用程序』中的步骤以运行 Paho MQTT Sample 客户机。

您的目标是准备开发环境以开发和测试 JAAS 认证。 您需要 Java 开发环境以定制 JAAS 认证模块。 在本示例中，您运行样本 Java Paho 客户机以测试 JAAS 配置。 为了简单起见，请使用同一开发环境来修改样本客户机和样本 JAAS 登录模块。 或者，您也可以使用 C MQTT 客户机或任何其他 MQTT 客户机来测试 JAAS 登录模块。

2. 可选： 将用户名和密码参数添加到 MQTT Paho 样本。

   注: 如果 Java Paho 客户机包含用户名和密码参数，那么不必执行此步骤。 查看下载站点以查找更新。请参阅 IBM 消息传递社区下载，否则更改您的 `Sample.java` 副本。

   a) 在 Paho 样本项目中的 `org.eclipse.paho.sample.mqttv3app` 软件包内打开 Package Explorer。

   b) 右键单击 `Sample.java`，然后选择**复制** > **粘贴**。 在“**名称冲突**”窗口中，输入名称 `SampleForJAAS`。

   c) 将以下代码行添加到 `main` 方法。

      i) 在 “`boolean ssl = false;`”行之后，声明 `userName` 和 `password` 变量:

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

      为了与更旧的 MQTT 服务器兼容，缺省情况下，请勿设置密码和用户名参数。

      ii) 在 “`case 'v': ssl = Boolean.valueOf(args[++i]).booleanValue(); break;`”行之后，解析两个新的输入参数:

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

      iii) 在行 “`if (action.equals("publish")) {`”之前，将 `userName` 和 `password` 添加到 `Sample` 构造函数自变量:

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

   d) 将 `userName` 和 `password` 添加到 `Sample` 构造方法。

      将:

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

      到:

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

   e) 将以下代码行添加到 `Sample` 构造方法。

      在 “`conOpt.setCleanSession(clean);`”行之后，在 `Sample` 方法的 `conOpt` 对象中设置 `userName` 和 `password` 变量:

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

   f) 在 `publish` 和 `subscribe` 方法中，修改以下代码行:

      将行 “`client.connect();`” 更改为

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

3. 为 JAAS 示例创建 Java 项目 `JAASSample`。

   a) 在 Eclipse 工作空间中，打开“**新建 Java 项目**”向导：单击**文件** > **新建** > **Java 项目**。

b) 在**项目名称**字段中，输入 JAASSample。

c) 在 JRE 选项中，选择 J2SE-1.5 作为执行环境 JRE，然后单击**下一步**。

此 JRE 必须与 IBM WebSphere MQ 服务器运行的 JRE 匹配。 IBM WebSphere MQ Version 7.5 运行 J2SE-1.5。

d) 在“**Java 设置**”窗口中，单击**库**选项卡，然后单击**添加外部 JARS...**。 浏览至%MQ_FILE_PATH%\mqxr\lib 目录并选择 **MQXR.jar**; 单击 **完成**。

4. 导入 JAAS 样本 JAASLoginModule 和 JAASPrincipal 类。

a) 右键单击 "包资源管理器" 中的 JAASSample 项目 **导入 ...** > **常规** > **文件系统** ，然后单击 **下一步**。

b) 浏览到 %MQ_FILE_PATH%\mqxr\samples 并选中 JAASLoginModule.java 和 JAASPrincipal.java；单击**完成**。

c) 选择并右键单击 "包资源管理器" 中的两个 Java 文件 **重构 ...** > **移动**。

d) 在“**移动**”窗口中，确认已将 JAASSample 选作这两个元素的目标，然后单击**创建包...**。

e) 在“**新建 Java 包**”向导的**名称**字段内输入 samples；单击**完成** > **确定**。

Eclipse 会构建所导入的 Java 类，但带有若干条关于未使用的值的警告。

5. 重命名 JAASLoginModule 类。

重命名类，以便更容易将其与 IBM WebSphere MQ 随附的样本 JAASLoginModule 类区分开来。

a) 右键单击包资源管理器中的 JAASloginModule.java **重构 ...** > **重命名**。

b) 在“**重命名编译单元**”窗口中，将**新名称**字段从 JAASloginModule 更改为 MyJAASloginModule；单击**完成**。

6. 修改 MyJAASloginModule 类以输出回调字段的内容。

a) 将以下代码行添加到 MyJAASloginModule.java。

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

将行放在语句前面:“if (true) loggedIn = true;”

b) 按 CTRL+Shift+O 以重新组织导入，然后保存该文件。

7. 将 JAASPrincipal 重命名为 MyJAASPrincipal。

将该类重命名可避免与样本 JAASPrincipal 类混淆。 在本示例中，将 MyJAASPrincipal 类的内容保持不变。

8. 将对 JAAS 类的 read 和 execute 许可权授予要运行队列管理器进程的用户标识。

a) 在 Windows Explorer 中，打开 Eclipse 工作空间目录。 在此示例中， Eclipse 工作空间位置由 Eclipse 变量 *workspace_loc* 表示。

b) 浏览到包含 MyJAASLoginModule 和 MyJAASPrincipal 类的目录。

目录路径为 *workspace_loc*\JAASSample\bin\samples。

c) 选择然后右键单击这两个类，并单击**属性**；单击“**属性**”窗口中的**安全性**选项卡。

d) 单击 **添加 ...**， 输入对象名 mqm，然后单击 **检查名称** 以进行验证; 单击 **确定**。

e) 在“**组或用户名**”的列表中选择 **mqm**，然后在 mqm 的许可权列表内选中**读取和执行**以及**读取**；单击“确定”。

9. 将 IBM WebSphere MQ 配置为运行您的 MyJAASLoginModule 类。

a) 将 service.env 文件添加到 IBM WebSphere MQ 配置以定义用于装入您的 MyJAASLoginModule 类的类路径。

使用以下类路径语句在 *WMQ_DATA_PATH* 目录中创建 service.env 文件:

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

其中， *user.dir* 是在 Eclipse 工作空间中编译的类文件的目录根目录。 *WMQ_DATA_PATH* 目录包含 qmgrs 目录。 请参阅其他环境变量。

提示: CLASSPATH=*user.dir*\JAASSample\bin 可能是 service.env 文件中的唯一语句。

b) 将 MyJAASStanza 节添加到 jaas.config 文件以相对于 service.env 文件中的类路径来标识 MyJAASLoginModule 类。

jaas.config 在队列管理器 mqxr 目录 *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr 中。

该节为:

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

c) 以 JAAS 配置节的名称来配置 IBM WebSphere MQ Telemetry 通道。

从命令窗口运行以下命令:

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

该命令将 MyJAAS 通道与 jaas.config 文件中的 MyJAASStanza 关联。 通过在通道定义中不指定 MCAUSER 选项或 USECLTID 选项，通道授权使用 MQTT 客户机程序所提供的用户名来访问队列管理器资源。 在此示例中，客户机提供的用户名设置为 “Guest”。 本示例中还使用了由 SampleMQM 命令文件设置的现有 Guest 权限。

10. 重新启动 IBM WebSphere MQ Telemetry 服务以读取新配置数据。
要重新启动 IBM WebSphere MQ Telemetry 服务，请从 IBM WebSphere MQ Explorer 启动队列管理器或此服务，或者为样本配置运行以下命令:

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

11. 运行 Sample 程序。

要配置 SampleForJAAS 的运行配置，请遵循步骤 第 66 页的『1』 中的相同过程，同时对其做出以下修改:

a) 将端口号设置为 1890 以匹配 MQTT 通道配置。

b) 对于您已为 Sample 程序创建的订户和发布者配置，均将参数 -u Guest -z password 添加到 “**(x)= 自变量**”选项卡上的密码中。

样本程序会运行，而且输出中没有任何变化，但端口号现在是 1890，而非 1883。

在 *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM 目录中，打开 mqxr.stdout 文件。 来自 MyJAASLoginModule 的输出会写到 mqxr.stdout:

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

下一步做什么

如果您的示例无法正常工作，请阅读针对 JAAS 的故障诊断主题第 161 页的『解决问题：遥测服务未调用 JAAS 登录模块』，然后尝试以下调试提示。

1. 将 -verbose 添加到 *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr\java.properties 中的参数。 在此日志中，您可以了解是否成功装入了类。

输出会写到 *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr.stderr。

2. 在 *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\errors\mqxr.log 中查找 MyJAASLoginModule 中所抛出的异常。 例如，如果您尝试输出空的 password（这是字符数组而不是字符串），那么将抛出异常。
3. 查看 *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\errors\AMQERR01.log。 如果 userName 中的用户名未获访问队列管理器资源的授权，并且在未指定 MCAUSER 或在指定 USECLTID 选项的情况下配置通道，那么此处会报告所有错误。
4. 检查 *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr\jaas.config 中节的名称是否与为 Sample 客户机尝试连接的端口配置的 MQTT 通道中的名称相同。
5. 确保节中的路径与 Eclipse 中 MyJAASLoginModule 类的路径匹配；例如：

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

6. 要消除 *WMQ_DATA_PATH* 中的 service.env 文件中的类路径是否存在故障，请将 %MQ_FILE_PATH%\mqxr\bin\controlMQXR.BAT 中的 “set CLASSPATH=%MQXRCLASSPATH%;%CLASSPATH%” 行更改为包含您的类路径。 您还可以将此类路径回传。 然而，此类路径不包含 service.env 中所设置的类路径，所以这仅在您修改 controlMQXR.BAT 文件的情况下才有效。

相关概念

第 47 页的『MQTT 安全性』
MQTT 安全性具有三个基本概念：身份、认证和权限。 身份是对要为其授权或赋予权限的客户机命名。 认证是提供客户机的身份；而权限是管理赋予客户机的权利。

第 104 页的『遥测通道 JAAS 配置』
配置 JAAS 以认证由客户机发送的**用户名**。

相关任务

第 161 页的『解决问题：遥测服务未调用 JAAS 登录模块』
了解 JAAS 登录模块是否未由遥测 (MQXR) 服务调用，并配置 JAAS 以更正该问题。

第 49 页的『构建和运行安全 MQTT 客户机样本 Java 应用程序』
根据 Windows 示例，您可以在 IBM MessageSight 或 IBM WebSphere MQ 作为 MQTT 服务器上启动并运行安全样本 Java 应用程序。 您可以在具有 JSE 1.5 或更高版本 (即 “Java 兼容”) 的任何平台上运行 Java 的 MQTT 客户机 应用程序

第 57 页的『通过 SSL 在 Android 上连接 MQTT 客户机样本 Java 应用程序』
启动并运行通过 SSL 连接到 IBM WebSphere MQ 的样本 Android MQTT 客户机。

相关信息

其他环境变量

通过 SSL 连接 JavaScript 的 MQTT 消息传递客户机 和 WebSockets

通过将 JavaScript 的 MQTT 消息传递客户机 样本 HTML 页面与 SSL 和 WebSocket protocol 配合使用，将 Web 应用程序安全地连接到 IBM WebSphere MQ 。

开始之前

1. 您必须能够访问可支持 WebSockets 上的 MQTT protocol 的 MQTT version 3 服务器。
2. 浏览器必须支持 SSL 和 WebSocket protocol。 请参阅第 153 页的『基于 SSL 的移动消息传递 Web 应用程序的浏览器支持限制』。
3. SSL 通道必须已启动。

关于此任务

完成本任务以通过 SSL 运行 JavaScript MQTT 消息传递客户机样本页面。 该任务会将您引导至第 87 页的『生成密钥和证书』，以创建证书并配置 IBM WebSphere MQ。

使用认证中心签名的密钥或自签名的密钥保护 SSL 通道。

过程

1. 选择可以将客户机应用程序连接到的 MQTT 服务器。

   服务器必须支持安全 WebSockets 上的 MQTT protocol。

   - IBM MessageSight 以及 IBM WebSphere MQ V7.5.0.1 和更高版本的发行版执行此操作。
2. 可选： 安装 V7 或更高版本的 Java 开发包 (JDK)。

   如果您在设置测试系统，并且希望使用自签名证书，那么需要使用 JDK V7 **keytool** 命令来认证您的证书。 如果您在设置生产系统，并且将证书签署请求 (CSR) 发送到外部认证中心，那么不需要 JDK V7。
3. 创建并运行脚本以生成密钥对和证书，并将 IBM WebSphere MQ 配置为 MQTT 服务器。

   遵循 第 87 页的『生成密钥和证书』 中的步骤来创建和运行脚本。在第 72 页的『用于为 Windows 配置 SSL 证书的示例脚本』中也列出了这些脚本。

   服务器证书的通用名称必须与服务器通道的 DNS 名称匹配。 某些浏览器接受包含一列通用名称的证书；例如：

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

   其他浏览器只接受一个通用名称。 例如，Firefox（最高到 V18）只接受一个通用名称。 之后的版本可能会有所不同。
4. 检查 SSL 通道是否正在运行，以及是否为期望的设置。

   在 IBM WebSphere MQ 上，在命令窗口中输入以下命令：

   - 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. 将证书安装到浏览器证书库。

   例如，选择以下某个服务器证书：

   a. 对于自签名服务器证书，该证书为 `srvcertselfsigned.cer`。

   b. 对于由专用认证中心签名的服务器证书，该证书为 `cacert.cer`。

   c. 对于由外部认证中心签名的服务器证书，请查看是否已在证书库中安装了认证中心的根证书。

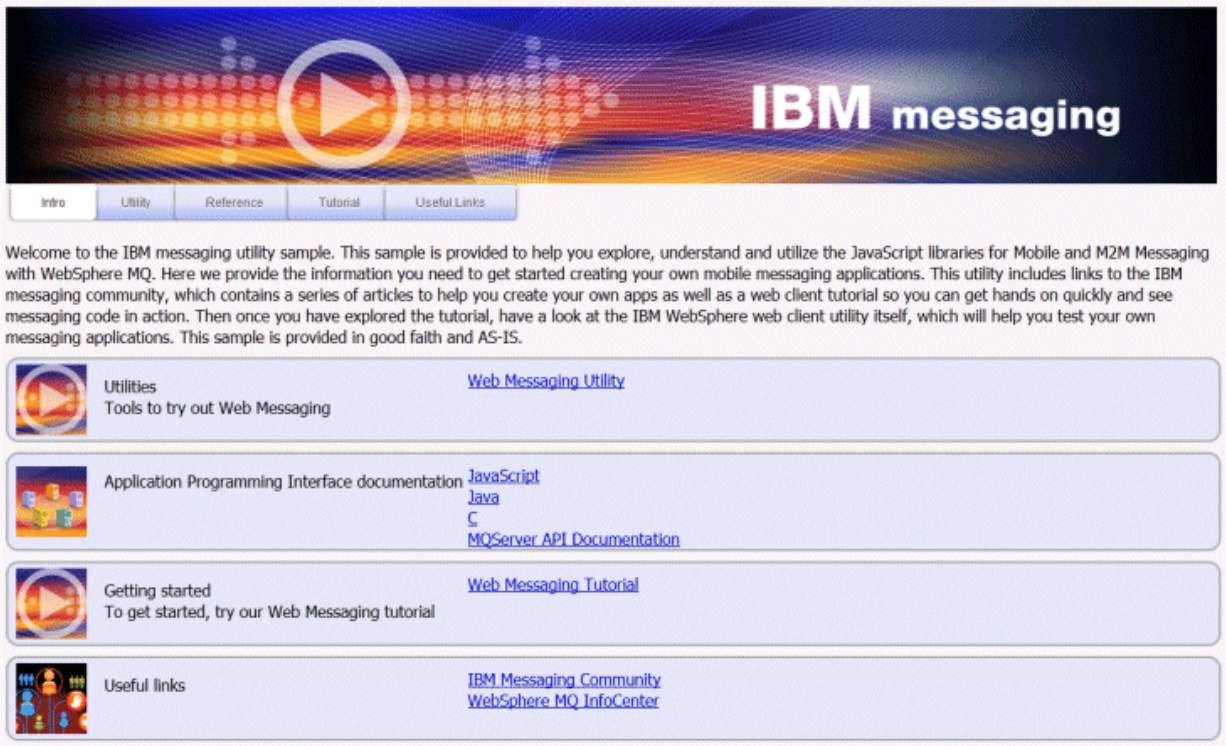
   根据浏览器中的支持情况，将 `cacert.cer` 安装到可信根认证中心的列表中。 请参阅第 153 页的『基于 SSL 的移动消息传递 Web 应用程序的浏览器支持限制』。
6. 可选： 对客户机进行认证。

   在本示例中，您安装了 `cacert.cer` 来对服务器进行认证并对通道进行加密，但不对客户机进行认证。要对客户机进行认证，必须在浏览器中安装客户机密钥库 `cltkeystore.p12`。 并非所有浏览器都支持对客户机进行认证。 请参阅第 153 页的『基于 SSL 的移动消息传递 Web 应用程序的浏览器支持限制』。
7. 连接到安全 WebSockets 通道。

   打开浏览器并在地址栏中输入 WebSockets 通道的 URL；在本示例中为：

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

   IBM WebSphere MQ 通过 MQTT 消息传递客户机样本 JavaScript 页面的第一页进行响应。



如果连接失败，而您已运行示例脚本来设置样本 MQTT 队列管理器，请尝试连接到端口 1886 上的正常 WebSockets 通道。 在端口 1886 上连接成功会隔离 SSL 连接故障。

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

相关概念

第 106 页的『JavaScript 的 MQTT 消息传递客户机和 Web 应用程序』
第 108 页的『如何使用 JavaScript 对消息传递应用程序进行编程』

相关任务

第 87 页的『生成密钥和证书』
按照本过程来为 Java 和 C 客户机（包括 Android 和 iOS 应用程序）以及 IBM WebSphere MQ 和 IBM MessageSight 服务器生成密钥和证书。

第 21 页的『JavaScript 的 MQTT 消息传递客户机入门』
您可以通过显示消息传递客户机样本主页并浏览其链接到的资源来开始使用 JavaScript 的 MQTT 消息传递客户机。 要显示此主页，请将 MQTT 服务器配置为接受来自 MQTT 消息传递客户机样本 JavaScript 页面的连接，然后将已在服务器上配置的 URL 输入 Web 浏览器中。 JavaScript 的 MQTT 消息传递客户机会在您设备上自动启动，然后消息传递客户机样本主页将显示。 该页面包含指向实用程序、编程接口文档、教程及其他有用信息的链接。

相关参考

第 153 页的『基于 SSL 的移动消息传递 Web 应用程序的浏览器支持限制』
不同平台上的不同浏览器在功能方面也各有差异。 了解这些差异将帮助您配置应用程序、认证中心 (CA) 和客户机证书，以通过基于 SSL 的 JavaScript 的 MQTT 消息传递客户机 和 WebSockets 进行连接。

用于为 Windows 配置 SSL 证书的示例脚本

** **

此示例命令文件根据任务中各个步骤的描述来创建证书和证书库。 此外，本示例还将 MQTT 客户机队列管理器设置为使用服务器证书库。 此示例通过调用 IBM WebSphere MQ 随附的 `SampleMQM.bat` 脚本来删除并重新创建队列管理器。

initcert.bat

initcert.bat 设置证书的名称和路径以及 **keytool** 和 **openSSL** 命令所需的其他参数。 脚本中的注释描述了这些设置。

```
@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 chlssloptws=SSLOPTWS
set portssloptws=8886
set chlsslreqws=SSLREQWS
set portsslreqws=8887
set mqlog=%certpath%\wmq.log
```

cleancert.bat

cleancert.bat 脚本中的命令会删除 MQTT 客户机队列管理器，以确保服务器证书库未锁定，然后会删除样本安全脚本创建的所有密钥库和证书。

```
@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 %srvcertreq%
erase %srvcertcasigned%
erase %srvcertselfsigned%
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 脚本中的命令将创建用于专用认证中心、服务器和客户机的密钥对。

```
@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 脚本中的命令会将客户机和服务器自签名证书从其密钥库中导出，然后将服务器证书导入客户机信任库中，并将客户机证书导入到服务器密钥库中。 服务器没有信任库。 该命令将从客户机 JKS 信任库创建 PEM 格式的客户机信任库。

```
@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
@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

该脚本会将认证中心根证书导入专用密钥库。 需要使用 CA 根证书在根证书和签名证书之间创建密钥链。 cacerts.bat 脚本会将客户机和服务器证书请求从其密钥库中导出。 此脚本使用 cajkskeystore.jks 密钥库中专用认证中心的密钥来签署证书请求，然后将签署的证书导入到发出请求的同一密钥库。 该导入使用 CA 根证书创建证书链。 该脚本将从客户机 JKS 信任库创建 PEM 格式的客户机信任库。

```
@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:%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

该脚本将在证书目录中列出密钥库和证书。 然后，它将创建 MQTT 样本队列管理器并配置安全遥测通道。

```
@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%
```
V 7.5.0.1
```
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
```

构建和运行安全 MQTT 客户机样本 C 应用程序

您可以根据 Windows 示例在您可以为其编译 C 源代码的任何操作系统上启动并运行安全样本 C 应用程序。验证您是否可以在 IBM MessageSight 或 IBM WebSphere MQ 作为 MQTT 服务器上运行样本 C 应用程序。

开始之前

1. 您必须有权访问支持 MQTT protocol over SSL 的 MQTT version 3.1 服务器。
2. 如果客户机与服务器之间存在防火墙，请检查它是否未阻止 MQTT 流量。
3. 为若干种操作系统提供了 C 客户机库的二进制版本。 对于部分操作系统，未以二进制文件形式提供客户机的安全版本。 对于这些操作系统，必须遵循第 27 页的『构建 C MQTT 客户机库』中的指示信息。
4. 为了解决问题，IBM 支持人员可能会要求您在参考平台上运行 C MQTT 客户机。
5. SSL 通道必须已启动。

有关受支持平台和参考平台的概述，请参阅 IBM 移动消息传递和 M2M 客户机包的系统需求。 有关 C 客户机所支持内容的详细信息，请参阅 System Requirements for WebSphere MQ V7.5 Telemetry 的相关部分。

关于此任务

作为示例，本文说明如何从命令行编译和运行 Windows 上的安全 MQTT 客户机样本 C 应用程序 。 在说明中，Microsoft Visual Studio 2010 用于编译客户机。 您可以修改命令行脚本以在其他操作系统（如 Linux 和 iOS）上编译和运行样本应用程序。

注:

本文章中提供的 Windows 脚本假定您从源构建整个 OpenSSL 包。 如果您选择使用 IBM 提供的预编译的库，那么还可能会首选获取 OpenSSL 的预编译二进制发行版。 预编译库不可用于 iOS。

使用认证中心签名的密钥或自签名的密钥保护 SSL 通道。

过程

1. 选择可以将客户机应用程序连接到的 MQTT 服务器。

   服务器必须支持基于 SSL 的 MQTT version 3.1 协议。 来自 IBM 的所有 MQTT 服务器都执行此操作，包括 IBM WebSphere MQ 和 IBM MessageSight。 请参阅第 122 页的『MQTT 服务器入门』。
2. 可选： 安装 V 7 或更高版本的 Java 开发包 (JDK)。

   V 7 才能运行 **keytool** 命令来认证证书。 如果将不对证书进行认证，那么就不需要 JDK V7。
3. 将 C 开发环境安装在您要在其上进行构建的平台上。

   本主题中的示例内的 makefile 以下列工具为目标：

   - iOS 对于 iOS，在具有 OS X 10.8.2 的 Apple Mac 上，使用来自 Xcode 的 iOS 开发工具。
   - Linux 对于 Linux：来自 Red Hat Enterprise Linux V6.2 的 gcc V4.4.6。

     glibc C 库的最低受支持级别是 2.12，Linux 内核的最低受支持级别是 2.6.32。
   - Windows 对于 Microsoft Windows：Visual Studio V10.0。

4. 下载 移动消息传递和 M2M 客户机包 并安装 MQTT SDK。

   不存在安装程序，您只需展开下载的文件。

   a. 下载 移动消息传递和 M2M 客户机包。

   b. 创建将在其中安装 SDK 的文件夹。

      您可能希望将此文件夹命名为 MQTT。 此文件夹的路径在此被称为 *sdkroot*。

   c. 将压缩的 移动消息传递和 M2M 客户机包 文件内容展开到 *sdkroot* 中。 此展开操作将创建以 *sdkroot*\SDK 开头的目录树。

5. 可选： 执行 第 27 页的『构建 C MQTT 客户机库』中的步骤。

   请仅在 MQTT SDK 不包含目标操作系统的安全 C 客户机库的情况下执行此步骤。

   - Windows 库是用于编译的 `mqttv3cs.lib` 和用于运行的 `mqttv3cs.dll`。
   - Linux 库是 `libmqttv3cs.so`。
   - iOS 库是 `libmqttv3cs.a`。

6. 创建并运行脚本以生成密钥对和证书，并将 IBM WebSphere MQ 配置为 MQTT 服务器。

   遵循 第 87 页的『生成密钥和证书』中的步骤来创建和运行脚本。在第 81 页的『用于为 Windows 配置 SSL 证书的示例脚本』中也列出了这些脚本。

7. 检查 SSL 通道是否正在运行，以及是否为期望的设置。

   在 IBM WebSphere MQ 上，在命令窗口中输入以下命令：

   - 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. 创建用于构建和运行安全 MQTT 客户机样本 C 应用程序的脚本。

   a) 创建并运行 `sscclient.bat`，以测试受自签名证书保护的 SSL 通道。

   b) 创建并运行 `cacclient.bat`，以测试受认证中心签名证书保护的 SSL 通道。

结果

结果与运行不受保护的客户机结果类似。

```
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
```

图 *16:* 安全订户

```
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 . . .
```

图 *17:* 安全发布者

用于运行安全 MQTT 客户机样本 C 应用程序的脚本

在运行这些脚本之前先运行第 81 页的『用于为 Windows 配置 SSL 证书的示例脚本』中的脚本。

使用自签名证书保护 MQTT 客户机样本 C 应用程序 。

使用通过运行 `sscerts.bat` 脚本所创建的自签名证书来运行此脚本。

```
@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
```

图 *18: sscclient.bat*

使用认证中心签署的证书来运行 MQTT 安全客户机样本 C 应用程序。

使用通过运行 `cacerts.bat` 脚本所创建的认证中心签名证书来运行此脚本。

```
@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
```

图 *19: cacclient.bat*

相关概念

第 47 页的『MQTT 安全性』

MQTT 安全性具有三个基本概念：身份、认证和权限。 身份是对要为其授权或赋予权限的客户机命名。 认证是提供客户机的身份；而权限是管理赋予客户机的权利。

相关任务

第 87 页的『生成密钥和证书』

按照本过程来为 Java 和 C 客户机（包括 Android 和 iOS 应用程序）以及 IBM WebSphere MQ 和 IBM MessageSight 服务器生成密钥和证书。

用于为 Windows 配置 SSL 证书的示例脚本

示例

此示例命令文件根据任务中各个步骤的描述来创建证书和证书库。 此外，本示例还将 MQTT 客户机队列管理器设置为使用服务器证书库。 此示例通过调用 IBM WebSphere MQ 随附的 `SampleMQM.bat` 脚本来删除并重新创建队列管理器。

`initcert.bat`

`initcert.bat` 设置证书的名称和路径以及 **keytool** 和 **openSSL** 命令所需的其他参数。 脚本中的注释描述了这些设置。

```
@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 chlssloptws=SSLOPTWS
set portssloptws=8886
set chlsslreqws=SSLREQWS
set portsslreqws=8887
set mqlog=%certpath%\wmq.log
```

cleancert.bat

cleancert.bat 脚本中的命令会删除 MQTT 客户机队列管理器，以确保服务器证书库未锁定，然后会删除样本安全脚本创建的所有密钥库和证书。

```
@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 %srvcertreq%
erase %srvcertcasigned%
erase %srvcertselfsigned%
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 脚本中的命令将创建用于专用认证中心、服务器和客户机的密钥对。

```
@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 脚本中的命令会将客户机和服务器自签名证书从其密钥库中导出，然后将服务器证书导入客户机信任库中，并将客户机证书导入到服务器密钥库中。 服务器没有信任库。 该命令将从客户机 JKS 信任库创建 PEM 格式的客户机信任库。

```
@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
@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

该脚本会将认证中心根证书导入专用密钥库。 需要使用 CA 根证书在根证书和签名证书之间创建密钥链。 cacerts.bat 脚本会将客户机和服务器证书请求从其密钥库中导出。 此脚本使用 cajkskeystore.jks 密钥库中专用认证中心的密钥来签署证书请求，然后将签署的证书导入到发出请求的同一密钥库。 该导入使用 CA 根证书创建证书链。 该脚本将从客户机 JKS 信任库创建 PEM 格式的客户机信任库。

```
@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:%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

该脚本将在证书目录中列出密钥库和证书。 然后，它将创建 MQTT 样本队列管理器并配置安全遥测通道。

```
@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%
```

V 7.5.0.1

```
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
```

生成密钥和证书

按照本过程来为 Java 和 C 客户机（包括 Android 和 iOS 应用程序）以及 IBM WebSphere MQ 和 IBM MessageSight 服务器生成密钥和证书。

开始之前

1. 您必须具有 **keytool** 命令的副本。 并非所有版本的 **keytool** 都支持将密钥库从 Java 密钥库 (JKS) 转换为公用密钥加密系统 (PKCS) 或对证书请求进行签名。 本示例使用 JDK V7.0（支持上述两种功能）中的 **keytool** 命令。
2. 如果您打算为 C 客户机生成密钥和证书（采用保密性增强邮件 (PEM) 格式），那么必须拥有 **openSSL** 命令的副本。 遵照第 27 页的『构建 C MQTT 客户机库』中的步骤构建 openSSL 包。
3. 更改 `initcert.bat` 脚本中的参数值以满足您自己的需求。 特别说明，您可以选择省略密码参数以避免记录密码。 **keytool** 会提示您缺少密码。

关于此任务

在 MQTT 客户机和服务器之间创建安全 SSL 连接时需要密钥和证书。 本任务展示用于创建您所需密钥和证书的两种不同方式：自签名方式和您拥有的认证中心签名方式。 采用的方式取决于计划管理密钥库和证书的方式。

要使用由外部认证中心签署的证书，请将 `cacerts.bat` 中的签署步骤替换为向外部认证中心发送认证请求。 除签名证书外，认证中心还可能返回中间和根证书。 遵循要安装返回的证书的外部 CA 提供的指南。

IBM WebSphere MQ 服务器仅在您在遥测通道配置参数中指定的证书库中搜索证书。 它不会在 JSE `cacerts` 库中进行额外的搜索。 Java 客户机将在您指定的信任库中搜索证书。 如果未指定信任库，那么它将在 JSE `jre\lib\security` 目录中的 `cacerts` 密钥库中搜索。 Android 将在 Android 设备上预定义的证书库中搜索证书。 C 客户机应用程序和 iOS 应用程序仅在应用程序所指定的证书库中进行搜索。

Android 和 Java 客户机在预配置的信任库搜索可信证书。 CA 根证书存储在 Android 可信证书库和 JSE `jre\lib\security\cacerts` 库中。 如果在预配置信任库中已安装了已认证服务器证书的 CA 的根证书，那么不要在定义客户机信任库。 唯一需要的配置是为受保护的 MQTT 服务器通道设置 TCP/IP 端口。

用于创建密钥和证书以及管理所有不同格式的工具使用起来并不简单。 它们具有各种需要管理的参数，并且 **openSSL** 需要一个配置文件 `openssl.cnf` 和多个命令行参数。 没有一种工具能够提供管理 C 和 Java 上运行的应用程序的密钥和证书所需的所有功能。 IBM WebSphere MQ 中的遥测通道需要 JKS 密钥库，因此示例主要使用 Java 证书工具 **ikeyman** 和 **keytool**。 然而，Java 工具不支持 PEM 格式（C 客户机应用程序所必需）。 要创建 PEM 格式的密钥库，请运行 **openSSL** 工具。 **openSSL** 工具可以将密钥库从 PKCS12 格式转换为 PEM 格式，而 **keytool** 可以在 JKS 格式和 PKCS12 格式之间转换密钥库。 密钥库转换不需要 `openssl.cnf` 文件。 仅当您计划构建 C 客户机应用程序或 iOS 应用程序时才需要 **openSSL**。 如果更喜欢使用 **openSSL**，您可以使用它签署证书，而不是使用 **keytool** 签署证书。

过程

1. 打开命令窗口，运行以下脚本。
2. 创建并运行 `initcert.bat` 脚本以设置运行 MQTT 安全样本客户机所需的参数。
3. 创建并运行 `cleancert.bat` 脚本以清理已准备好创建新密钥库和证书的环境。
4. 创建并运行 `genkeys.bat` 脚本，以生成您所需的密钥对。
5. 执行以下任一选项：
   - 创建并运行 `sscerts.bat` 脚本以创建自签名证书。
   - 创建并运行 `cacerts.bat` 脚本以创建认证中心签署的证书链。
6. 创建并运行 `mqcerts.bat` 脚本，以创建 `MQXR_SAMPLE_QM` 队列管理器并配置其遥测通道。

相关任务

第 78 页的『构建和运行安全 MQTT 客户机样本 C 应用程序』
您可以根据 Windows 示例在您可以为其编译 C 源代码的任何操作系统上启动并运行安全样本 C 应用程序。 验证您是否可以在 IBM MessageSight 或 IBM WebSphere MQ 作为 MQTT 服务器上运行样本 C 应用程序。

第 49 页的『构建和运行安全 MQTT 客户机样本 Java 应用程序』
根据 Windows 示例，您可以在 IBM MessageSight 或 IBM WebSphere MQ 作为 MQTT 服务器上启动并运行安全样本 Java 应用程序。 您可以在具有 JSE 1.5 或更高版本 (即 “Java 兼容”) 的任何平台上运行 Java 的 MQTT 客户机 应用程序

用于为 Windows 配置 SSL 证书的示例脚本

示例

此示例命令文件根据任务中各个步骤的描述来创建证书和证书库。 此外，本示例还将 MQTT 客户机队列管理器设置为使用服务器证书库。 此示例通过调用 IBM WebSphere MQ 随附的 `SampleMQM.bat` 脚本来删除并重新创建队列管理器。

initcert.bat

`initcert.bat` 设置证书的名称和路径以及 **keytool** 和 **openSSL** 命令所需的其他参数。 脚本中的注释描述了这些设置。

```
@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 chlssloptws=SSLOPTWS
set portssloptws=8886
```

```
set chlsslreqws=SSLREQWS
set portsslreqws=8887
set mqlog=%certpath%\wmq.log
```

cleancert.bat

cleancert.bat 脚本中的命令会删除 MQTT 客户机队列管理器，以确保服务器证书库未锁定，然后会删除样本安全脚本创建的所有密钥库和证书。

```
@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 %srvcertreq%
erase %srvcertcasigned%
erase %srvcertselfsigned%
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 脚本中的命令将创建用于专用认证中心、服务器和客户机的密钥对。

```
@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 脚本中的命令会将客户机和服务器自签名证书从其密钥库中导出，然后将服务器证书导入客户机信任库中，并将客户机证书导入到服务器密钥库中。 服务器没有信任库。 该命令将从客户机 JKS 信任库创建 PEM 格式的客户机信任库。

```
@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
@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

该脚本会将认证中心根证书导入专用密钥库。 需要使用 CA 根证书在根证书和签名证书之间创建密钥链。 cacerts.bat 脚本会将客户机和服务器证书请求从其密钥库中导出。 此脚本使用 cajkskeystore.jks 密钥库中专用认证中心的密钥来签署证书请求，然后将签署的证书导入到发出请求的同一密钥库。 该导入使用 CA 根证书创建证书链。 该脚本将从客户机 JKS 信任库创建 PEM 格式的客户机信任库。

```
@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:%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

该脚本将在证书目录中列出密钥库和证书。 然后，它将创建 MQTT 样本队列管理器并配置安全遥测通道。

```
@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%
V 7.5.0.1
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 客户机标识、授权和认证

遥测 (MQXR) 服务使用 MQTT 通道来代表 MQTT 客户机发布或预订 WebSphere MQ 主题。 WebSphere MQ 管理员配置用于 WebSphere MQ 授权的 MQTT 通道身份。 管理员可以为通道定义一个公共身份，也可以使用与此通道相连的客户机的**用户名**或**客户机标识**。

遥测 (MQXR) 服务可以使用客户机所提供的 Username 或者使用客户机证书来认证客户机。 使用客户机所提供的密码来认证**用户名**。

总结：客户机标识就是选择的客户机身份。 根据上下文不同，可通过**客户机标识**、**用户名**、由管理员创建的公共客户机身份或者客户机证书来标识客户机。 用于检查真实性的客户机标识不必是用于授权的同一标识。

MQTT 客户机程序设置通过使用 MQTT 通道发送至服务器的**用户名**和**密码**。 它们还可以设置一些 SSL 属性，对连接进行加密和认证时就需要这些 SSL 属性。 管理员将决定是否对 MQTT 通道进行认证以及如何进行认证。

要授权 MQTT 客户机访问 IBM WebSphere MQ 对象，请对客户机的**客户机标识**或**用户名**授权，或者对公共客户机身份授权。 要允许客户机连接到 IBM WebSphere MQ，请对**用户名**进行认证或使用客户机证书。 配置 JAAS 以对**用户名**进行认证，并配置 SSL 以对客户机证书进行认证。

如果您在客户机中设置**密码**，那么使用 VPN 对连接进行加密或者配置 MQTT 通道以使用 SSL，从而保持密码的私密性。

难以管理客户机证书。 正因为如此，如果可以接受进行密码认证存在的风险，那么通常使用密码认证对客户机进行认证。

如果有一种安全的方法来管理和存储客户机证书，那么可以依赖于证书认证来实现。 但是，在使用 Telemetry 的各种类型的环境中，很少能够安全地管理证书。 然而，通过在服务器中对客户机密码进行认证，可以作为“使用客户机证书对客户机进行认证”的补充。 由于还存在其他复杂性，因此，仅限于对高度敏感的应用程序使用客户机证书。 使用两种形式的认证被称为双因素认证。 您必须知道其中一个因素（例如，密码），并且具有另一个因素（例如，证书）。

在高度敏感的应用程序中（例如，chip-and-pin 设备），在制造期间将锁定此设备，以防止篡改内部硬件和软件。 会将一个可信的、具有时间限制的客户机证书复制到此设备中。 将此设备部署到要使用它的位置。 每当使用此设备时，都会使用密码或者来自智能卡的另一个证书对它执行进一步认证。

MQTT 客户机身份和权限

使用 ClientIdentifier、Username 或者用于授权的常用客户机身份来访问 WebSphere MQ 对象。

IBM WebSphere MQ 管理员在选择 MQTT 通道的身份时有三个选项可供选择。 当定义或修改客户机所使用的 MQTT 通道时，管理员可以作出选择。 身份用来授予对于 IBM WebSphere MQ 主题的访问权。 这些选项包括：

1. 客户机标识。
2. 管理员为通道提供的身份。
3. 从 MQTT 客户机传递的**用户名**。

用户名是 MqttConnectOptions 类的一种属性。 它必须在客户机连接至服务之前进行设置。 它的缺省值为 NULL。

使用 IBM WebSphere MQ **setmqaut** 命令来选择授权与 MQTT 通道相关联的身份使用哪些对象和哪些操作。 例如，要授予由队列管理器 QM1 的管理员提供的通道身份 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
```

相关信息

授权 MQTT 客户机访问 WebSphere MQ 对象

使用密码对 MQTT 客户机进行认证

使用客户机密码来认证**用户名**。 使用与用来授权客户机发布和预订主题的身份不同的身份对客户机进行认证。

遥测 (MQXR) 服务使用 JAAS 来认证客户机 Username。 JAAS 使用 MQTT 客户机所提供的**密码**。

IBM WebSphere MQ 管理员决定是通过配置客户机连接至的 MQTT 通道来认证**用户名**，还是根本不认证。 可以将客户机分配给不同的通道，并且可以配置每个通道以采用不同方式对它的客户机进行认证。 通过使用 JAAS，可以配置哪些方法必须对客户机进行认证，哪些方法可以有选择地对客户机进行认证。

为认证选择的身份并不会影响为授权选择的身份。 为了便于管理，您可能想为授权设置一个公共身份，但是对每个用户进行认证以使用该身份。 以下过程概述了对各个用户进行认证以使用公共身份时要执行的步骤：

1. IBM WebSphere MQ 管理员使用 IBM WebSphere MQ Explorer 将 MQTT 通道身份设置为任何名称，例如 MQTTClientUser。
2. IBM WebSphere MQ 管理员授权 MQTTClient 发布和预订任何主题：

   ```
   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. 在连接至服务器之前，MQTT 客户机应用程序开发人员创建 MqttConnectOptions 对象并设置 Username 和 Password。
4. 安全性开发者创建一个 JAAS LoginModule，以使用**密码**来认证**用户名**，并将此模块包括在 JAAS 配置文件中。
5. IBM WebSphere MQ 管理员将 MQTT 通道配置为使用 JAAS 认证客户机的**用户名**。

使用 SSL 进行 MQTT 客户机认证

MQTT 客户机与队列管理器之间的连接始终由 MQTT 客户机发起。 MQTT 客户机始终是 SSL 客户机。 服务器的客户机认证和 MQTT 客户机的服务器认证都是可选的。

通过向客户机提供专用签名数字证书，可以向 IBM WebSphere MQ 认证 MQTT 客户机。 IBM WebSphere MQ 管理员可以强制 MQTT 客户机使用 SSL 向队列管理器认证这些客户机自身。 只能在相互认证的过程中请求客户机认证。

除了使用 SSL 以外，某些种类的虚拟专用网 (VPN)（例如，IPSec）将对 TCP/IP 连接的端点进行认证。 VPN 将对流经网络的每个 IP 包进行加密。 一旦建立了这样一个 VPN 连接，您就已经建立了一个可信网络。 您可以使用 VPN 网络上的 TCP/IP 将 MQTT 客户机连接至遥测通道。

使用 SSL 的客户机认证依赖于具有私钥的客户机。 私钥是客户机的专用密钥（对于自签名证书来说），或者是由认证中心提供的密钥。 密钥用来为客户机的数字证书签名。 拥有相应的公用密钥的任何人都可以验证数字证书。 证书可能是可信的，如果它们是链式证书，那么追溯整个证书链以找到可信根证书。 客户机验

证将客户机所提供的证书链中的所有证书发送至服务器。 服务器将检查证书链，直到找到它信任的证书为止。 可信证书是根据自签名证书生成的公用证书，或者是通常由认证中心发放的根证书。 作为最后的可选步骤，可以将可信证书与“实时”的证书撤销列表进行比较。

可信证书可能已由认证中心颁发并已经包含在 JRE 证书库中。 它可以是自签名证书，也可以是已经作为可信证书添加至遥测通道密钥库的任何证书。

注: 遥测通道具有组合密钥库/信任密钥库，该库中包含一个或多个遥测通道的专用密钥以及对客户机进行认证所需的所有公用证书。 由于 SSL 通道必须具有密钥库且它是与通道信任库相同的文件，因此绝不会引用 JRE 证书库。 其含义为，如果客户机认证需要认证中心根证书，那么即使该证书已存在于 JRE 证书库中，也必须将其放入该通道的密钥库中。 绝不会引用 JRE 证书库。

请考虑进行客户机认证是为了对付威胁，以及客户机在服务器对付威胁时所起的作用。 只是对客户机证书进行认证还不足以防止对系统进行未经授权的访问。 如果其他人控制了客户机设备，那么该客户机设备不一定采用证书拥有者的权限来运作。 决不能依赖单一防护措施来对付不希望出现的攻击。 至少应使用双重认证方法，并使用私有信息的知识来补充证书的内容。 例如，使用 JAAS，以及使用由服务器发放的密码来认证客户机。

客户机证书面对的主要威胁是被不适当的人拥有。 证书保存在客户机上的一个受密码保护的密钥库中。 它是如何保存在密钥库中的？ MQTT 客户机如何将密码保存到密钥库中？ 密码保护的安全程度如何？ 遥测设备通常容易被移除，然后可以私下修改。 设备硬件必须防篡改吗？ 分发和保护客户机端证书被认为是一项艰巨的任务；它被称为密钥管理问题。

次要的威胁是，无意识地误用了设备来访问服务器。 例如，如果篡改了 MQTT 应用程序，那么有可能在使用已认证的客户机身份的服务器配置中使用薄弱环节。

要使用 SSL 对 MQTT 客户机进行认证，请配置遥测通道和此客户机。

-
-

使用 *SSL* 进行客户机认证的 *MQTT* 客户机配置

要使用 SSL 来认证 MQTT 客户机，客户机将使用 SSL 来连接至遥测通道。 它必须指定与配置为认证 SSL 客户机的遥测通道相对应的 TCP 端口。

例如，在客户机中:

```
MQTTClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```

客户机 JVM 必须使用 JSSE 提供的标准套接字工厂。 如果您正在使用 Java ME，那么必须确保装入 JSSE 包。 如果您正在使用 Java SE，那么自 Java 版本 1.4.1 以来， JSSE 已包含在 JRE 中。

在连接之前，SSL 连接需要设置许多 SSL 属性。 您可以通过使用 `-D` 开关将属性传递到 JVM 来设置属性，也可以使用 `MqttConnectionOptions.setSSLProperties` 方法来设置属性。

如果通过调用方法 `MqttConnectOptions.setSocketFactory(javax.net.SocketFactory)`来装入非标准套接字工厂，那么将 SSL 设置传递到网络套接字的方式由应用程序定义。

添加客户机数字证书，使用客户机专用密钥或由认证中心签署客户机上受密码保护的密钥库。 如果证书具有密钥链，那么可以将证书从密钥链添加至密钥库。 当服务器验证客户机证书时，它会比较客户机发送的证书和密钥库中的证书。 它将在密钥链中查找与它拥有的证书相匹配的第一个证书。 将忽略密钥链中其余的证书。

MQTT 客户机将其密钥库中的所有证书发送至服务器。 如果服务器对客户机发送的任何密钥链进行认证，那么也就对客户机进行了认证。

您还可以将 SSL 密码套件用于客户机认证。以下是当前受支持的 SSL 密码套件的字母顺序列表:

- `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 如果您计划使用 SHA-2 密码套件，请参阅第 153 页的『关于将 SHA-2 密码套件用于 MQTT 客户机的系统需求』。

相关概念

第 97 页的『使用 SSL 进行通道认证的 MQTT 客户机配置』

要使用 SSL 对遥测通道进行认证，客户机必须使用 SSL 来连接至遥测通道。 它必须指定与为 SSL 配置的遥测通道相对应的端口。 配置中必须包括一个受口令保护的密钥库，而此密钥库中包含服务器的私下签名的数字证书。

使用 SSL 进行遥测通道认证

MQTT 客户机与队列管理器之间的连接始终由 MQTT 客户机发起。 MQTT 客户机始终是 SSL 客户机。 服务器的客户机认证和 MQTT 客户机的服务器认证都是可选的。

除非客户机被配置为使用支持匿名连接的 CipherSpec，否则客户机始终会尝试对服务器进行认证。 如果认证失败，那么不会建立连接。

除了使用 SSL 以外，某些种类的虚拟专用网 (VPN)（例如，IPSec）将对 TCP/IP 连接的端点进行认证。 VPN 将对流经网络的每个 IP 包进行加密。 一旦建立了这样一个 VPN 连接，您就已经建立了一个可信网络。 您可以使用 VPN 网络上的 TCP/IP 将 MQTT 客户机连接至遥测通道。

使用 SSL 的服务器认证将对您要将保密信息发送至的服务器进行认证。 客户机根据其信任库或 JRE `cacerts` 库中的证书执行与从服务器发送的证书相匹配的检查。

JRE 证书库是 JKS 文件 `cacerts`。 它位于 `JRE InstallPath\lib\security\` 中。 它是使用缺省密码 `changeit` 进行安装的。 您可以将信任的证书存储在 JRE 证书库或客户机信任库中。 不能同时使用这两个库。 如果要使客户机信任的公用证书独立于其他 Java 应用程序使用的证书，请使用客户机信任库。 如果要针对客户机上正在运行的所有 Java 应用程序使用公共证书库，请使用 JRE 证书库。 如果确定要使用 JRE 证书库，请查看其所含证书以确保您信任这些证书。

可以通过提供另外的信任提供程序来修改 JSSE 配置。 可以定制信任提供程序以对证书执行另外的检查。 在一些使用了 MQTT 客户机的 OGSi 环境中， 该环境提供了不同的信任提供程序。

要使用 SSL 对遥测通道进行认证，请配置服务器和客户机。

-

相关概念

第 97 页的『使用 SSL 进行通道认证的 MQTT 客户机配置』
要使用 SSL 对遥测通道进行认证，客户机必须使用 SSL 来连接至遥测通道。 它必须指定与为 SSL 配置的遥测通道相对应的端口。 配置中必须包括一个受口令保护的密钥库，而此密钥库中包含服务器的私下签名的数字证书。

使用 SSL 进行通道认证的 MQTT 客户机配置

要使用 SSL 对遥测通道进行认证，客户机必须使用 SSL 来连接至遥测通道。 它必须指定与为 SSL 配置的遥测通道相对应的端口。 配置中必须包括一个受口令保护的密钥库，而此密钥库中包含服务器的私下签名的数字证书。

例如，在客户机中：

```
MQTTClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```

客户机 JVM 必须使用 JSSE 提供的标准套接字工厂。 如果您正在使用 Java ME，那么必须确保装入 JSSE 包。 如果您正在使用 Java SE，那么自 Java 版本 1.4.1 以来， JSSE 已包含在 JRE 中。

在连接之前，SSL 连接需要设置许多 SSL 属性。 您可以通过使用 `-D` 开关将属性传递到 JVM 来设置属性，也可以使用 `MqttConnectionOptions.setSSLProperties` 方法来设置属性。

如果通过调用方法 `MqttConnectOptions.setSocketFactory(javax.net.SocketFactory)` 来装入非标准套接字工厂，那么将 SSL 设置传递到网络套接字的方式由应用程序定义。

为客户机编写代码以使用 SSL 连接至遥测通道，并配置客户机以使用以下三种方式之一来信任服务器证书：

使用 `cacerts` 库中由众所周知的认证中心签名的服务器证书。

如果服务器发送证书链中的所有中间密钥，那么不进行其他配置。 建议您复查客户机 JRE 的 `cacerts` 库中的证书，并更改 `cacerts` 库的密码

其他证书

将您信任的证书存储在客户机上的信任库中。 必须将证书链中的至少一个证书存储在信任库中，并在 MqttConnectionOptions.SSLProperty 中设置信任库参数。

- com.ibm.ssl.trustStore
- com.ibm.ssl.trustStorePassword

使用定制信任管理器

实现信任提供程序，并为它传递所使用算法的名称。 设置提供程序类的名称以及要在 MqttConnectionOptions.SSLProperty 中使用的算法。

- com.ibm.ssl.trustStoreProvider
- com.ibm.ssl.trustStoreManager

也可使用 SSL 密码套件进行通道认证。以下是当前受支持的 SSL 密码套件的字母顺序列表:

- 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 如果您计划使用 SHA-2 密码套件，请参阅第 153 页的『关于将 SHA-2 密码套件用于 MQTT 客户机的系统需求』。

相关概念

第 95 页的『使用 SSL 进行客户机认证的 MQTT 客户机配置』
要使用 SSL 来认证 MQTT 客户机，客户机将使用 SSL 来连接至遥测通道。 它必须指定与配置为认证 SSL 客户机的遥测通道相对应的 TCP 端口。

发布在遥测通道上的隐私

通过使用 SSL 对借助连接实现的传输进行加密，即可保护在遥测通道中按任一方向发送的 MQTT 发布的隐私。

连接至遥测通道的 MQTT 客户机使用 SSL 来保护在通道上使用对称密钥密码来传输的发布的隐私。 由于未对端点进行认证，因此您不能只信任通道加密。 应将保护隐私与服务器认证或相互认证结合起来。

除了使用 SSL 以外，某些种类的虚拟专用网 (VPN)（例如，IPSec）将对 TCP/IP 连接的端点进行认证。 VPN 将对流经网络的每个 IP 包进行加密。 一旦建立了这样一个 VPN 连接，您就已经建立了一个可信网络。 您可以使用 VPN 网络上的 TCP/IP 将 MQTT 客户机连接至遥测通道。

对于典型配置（它会对通道进行加密和对服务器进行认证），请查阅第 97 页的『使用 SSL 进行遥测通道认证』。

只是对 SSL 连接进行加密，而不对服务器进行认证，会使连接受到中间人攻击。 尽管可以保护您交换的信息不被窃听，但是您并不知道是与谁在交换信息。 除非您控制整个网络，否则其他人就有可能拦截您的 IP 传输并且伪装成端点。

通过使用支持匿名 SSL 的 Diffie-Hellman 密钥交换 CipherSpec，即可创建已加密的 SSL 连接，而不对服务器进行认证。 建立了在客户机与服务器之间共享的主密钥，用来对 SSL 传输进行加密，而不交换私下签名的服务器证书。

因为匿名连接不安全，所以大多数 SSL 实现并不缺省设置为使用匿名 CipherSpec。 如果遥测通道接受了客户机请求以建立 SSL 连接，那么此通道必须具有受口令保护的密钥库。 缺省情况下，由于 SSL 实现不使用匿名 CipherSpec，因此密钥库中必须包含私下签名的证书，客户机可以对此证书进行认证。

如果您使用匿名 CipherSpec，那么服务器密钥库必须存在，但是它不需要包含任何私下签名的证书。

另一种建立加密连接的方法是，将客户机中的信任提供程序替换为您自己的实现。 您的信任提供程序将不对服务器证书进行认证，但是连接将加密。

MQTT 客户机和遥测通道的 SSL 配置

MQTT 客户机和 WebSphere MQ Telemetry (MQXR) 服务使用 Java 安全套接字扩展 (JSSE) 通过 SSL 连接遥测通道。 MQTT C 客户机以及设备的 WebSphere MQ Telemetry 守护程序不支持 SSL。

配置 SSL 以认证遥测通道和 MQTT 客户机，并对在客户机与遥测通道之间传递的消息进行加密。

除了使用 SSL 以外，某些种类的虚拟专用网 (VPN)（例如，IPSec）将对 TCP/IP 连接的端点进行认证。 VPN 将对流经网络的每个 IP 包进行加密。 一旦建立了这样一个 VPN 连接，您就已经建立了一个可信网络。 您可以使用 VPN 网络上的 TCP/IP 将 MQTT 客户机连接至遥测通道。

您可以配置 Java MQTT 客户机与遥测通道之间的连接，以使用基于 TCP/IP 的 SSL 协议。 所保护的对象取决于您如何配置 SSL 以使用 JSSE。 从最安全的配置开始，您可以配置三种不同级别的安全性：

1. 只允许可信的 MQTT 客户机进行连接。 将 MQTT 客户机仅连接至可信的遥测通道。 对客户机与队列管理器之间传递的消息进行加密；请参阅第 94 页的『使用 SSL 进行 MQTT 客户机认证』。
2. 将 MQTT 客户机仅连接至可信的遥测通道。 对客户机与队列管理器之间传递的消息进行加密；请参阅第 97 页的『使用 SSL 进行遥测通道认证』。
3. 对客户机与队列管理器之间传递的消息进行加密；请参阅第 99 页的『发布在遥测通道上的隐私』。

JSSE 配置参数

修改 JSSE 参数，以改变 SSL 连接的配置方式。 JSSE 配置参数分为三个集合：

1. IBM WebSphere MQ 遥测通道
2. MQTT Java 客户机
3. JRE

使用 IBM WebSphere MQ Explorer 配置遥测通道参数。 在 `MqttConnectionOptions.SSLProperties` 属性中设置 MQTT Java 客户机参数。 通过编辑客户机和服务器中的 JRE 安全性目录中的文件来修改 JRE 安全性参数。

IBM WebSphere MQ 遥测通道

使用 WebSphere MQ Explorer 来设置所有遥测通道 SSL 参数。

ChannelName

`ChannelName` 是所有通道的一个必需参数。

通道名称标识与特定端口号相关联的通道。 为通道命名，以帮助您管理多组 MQTT 客户机。

PortNumber

`PortNumber` 是所有通道的一个可选参数。 对于 TCP 通道，此参数的缺省值为 1883；对于 SSL 通道，此参数的缺省值为 8883。

TCP/IP 端口号与此通道相关联。 通过指定为通道定义的端口来将 MQTT 客户机连接至该通道。 如果该通道具有 SSL 属性，那么客户机必须使用 SSL 协议进行连接；例如：

```
MQTTClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```

KeyFileName

`KeyFileName` 是 SSL 通道的一个必需参数。 对于 TCP 通道，必须省略此参数。

`KeyFileName` 是包含您提供的数字证书的 Java 密钥库的路径。 使用 JKS、JCEKS 或 PKCS12 作为服务器上的密钥库类型。

可使用以下某个文件扩展名来确定密钥库类型：

```
.jks
.jceks
.p12
.pkcs12
```

具有其他任何文件扩展名的密钥库均视为 JKS 密钥库。

您可以将服务器上的一类密钥库与客户机上其他类型的密钥库相合并。

将服务器的私有证书放在密钥库中。 证书被称为服务器证书。 证书可以是自签名证书，也可以是签署机构所签署的证书链的一部分。

如果您要使用证书链，请将关联的证书放在服务器密钥库中。

服务器证书及其证书链中的任何证书被发送至客户机以认证服务器的身份。

如果您已将 ClientAuth 设置为 Required，那么密钥库中必须包含对客户机进行认证时所需要的任何证书。 客户机发送一个自签名证书，或一个证书链，这样会对照密钥库中的证书，通过首次验证此材料，对客户机进行认证。 通过使用证书链，一个证书可以验证许多客户机，即使向它们颁发了不同的客户机证书，也是如此。

PassPhrase

PassPhrase 是 SSL 通道的一个必需参数。 对于 TCP 通道，必须省略此参数。

口令用来保护密钥库。

ClientAuth

ClientAuth 是一个可选的 SSL 参数。 它缺省设置为不进行客户机认证。 对于 TCP 通道，必须省略此参数。

如果您想要遥测 (MQXR) 服务在允许客户机连接至遥测通道之前认证客户机，那么请设置 ClientAuth。

如果您设置 ClientAuth，那么客户机必须使用 SSL 连接至服务器并且对服务器进行认证。 作为对设置 ClientAuth 的响应，客户机将它的数字证书及其密钥库中的任何其他证书发送至服务器。 它的数字证书被称为客户机证书。 将针对保存在通道密钥库以及 JRE cacerts 库中的证书来认证这些证书。

CipherSuite

CipherSuite 是一个可选的 SSL 参数。 它缺省设置为尝试所有已启用的 CipherSpec。 对于 TCP 通道，必须省略此参数。

如果您想使用特定 CipherSpec，那么将 CipherSuite 设置为必须用来建立 SSL 连接的 CipherSpec 的名称。

遥测服务和 MQTT 客户机协商在每一端启用的所有 CipherSpec 中的一个公用 CipherSpec。 如果在连接的其中一端或者两端指定了一个特定 CipherSpec，那么此 CipherSpec 必须与另一端的 CipherSpec 相匹配。

通过将其他提供程序添加至 JSSE 来安装其他密码。

联邦信息处理标准 (FIPS)

FIPS 是一个可选设置。 缺省情况下，不会设置此项。

在队列管理器的“属性”面板中设置 SSLFIPS，或者使用 **runmqsc** 来设置。 SSLFIPS 指定是否仅使用经过 FIPS 证明的算法。

撤销名称列表

“撤销名称列表”是一个可选设置。 缺省情况下，不会设置此项。

在队列管理器的“属性”面板中设置 SSLCRLNL，或者使用 **runmqsc** 来设置。 SSLCRLNL 指定用来提供证书撤销位置的认证信息对象的名称列表。

不会使用其他用于设置 SSL 属性的队列管理器参数。

MQTT Java 客户机

在 MqttConnectionOptions.SSLProperties 中设置 Java 客户机的 SSL 属性; 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);
```

MqttConnectOptions 的 API 文档中描述了特定属性的名称和值。有关 MQTT 客户机库的客户机 API 文档的链接，请参阅 MQTT 客户机编程参考。

协议

Protocol 是可选的。

协议是在与遥测服务器协商时选择的。 如果您需要特定协议，那么可以选择一种协议。 如果遥测服务器不支持此协议，那么连接将失败。

ContextProvider

ContextProvider 是可选的。

KeyStore

KeyStore 是可选的。 如果在服务器中设置了 ClientAuth 以强制对客户机进行认证，那么请配置此项。

将使用客户机的专用密钥签名的数字证书放入密钥库中。 指定密钥库的路径和密码。 类型和提供程序是可选的。 JKS 是缺省类型，IBMJCE 是缺省提供程序。

指定另外的密钥库提供程序，以引用用于添加新的密钥库提供程序的类。 传递密钥库提供程序所使用的算法的名称，以通过设置密钥管理器名称来实例化 KeyManagerFactory。

TrustStore

TrustStore 是可选的。 您可以将您信任的所有证书都放在 JRE cacerts 存储库中。

如果要为客户机设置不同的信任库，请配置信任库。 如果服务器正在使用已在 cacerts 中存储其根证书的众所周知的认证中心发放的证书，那么可能无法配置信任库。

将服务器的公开签名的证书或者根证书添加至信任库，并指定信任库的路径和密码。 JKS 是缺省类型，IBMJCE 是缺省提供程序。

指定另外的信任库提供程序，以引用用于添加新的信任库提供程序的类。 传递信任库提供程序所使用的算法的名称，以通过设置信任管理器名称来实例化 TrustManagerFactory。

JRE

在 JRE 中配置了 Java 安全性的其他方面（这些方面影响客户机和服务器上的 SSL 行为）。 Windows 上的配置文件位于 *Java Installation Directory*\jre\lib\security 中。 如果使用 IBM WebSphere MQ 随附的 JRE，那么该路径如下表中所示：

表 *3: JRE SSL* 配置文件所在的不同平台的文件路径

平台	**菲尔帕特**
Windows	*WMQ Installation Directory*\java\jre\lib\security
Linux for System x 32 位	*WMQ Installation Directory*/java/jre/lib/security
其他 UNIX and Linux 平台	*WMQ Installation Directory*/java/jre64/jre/lib/security

众所周知的认证中心

cacerts 文件中包含众所周知的认证中心的根证书。 除非您指定信任库，否则缺省情况下将使用 cacerts。 如果使用 cacerts 存储库或未提供信任库，那么必须查看和编辑 cacerts 中的签署者列表以满足安全性需求。

您可以使用运行 IBM Key Management 实用程序的 WebSphere MQ 命令 strmqikm 来打开 cacerts 。 使用密码 changeit 将 cacerts 作为 JKS 文件打开。 修改密码以保护此文件的安全。

配置安全性类

使用 java.security 文件来注册其他安全性提供程序和其他缺省安全性属性。

许可权

使用 java.policy 文件来修改为资源授予的许可权。 javaws.policy 为 javaws.jar 授予许可权

加密强度

某些 JRE 提供了强度降低的加密。 如果您无法将密钥导入密钥库中，可能是由于加密强度降低引起的。 尝试使用 **strmqikm** 命令来启动 **ikeyman**，或者从 IBM Developer Kit 安全信息中下载强度较高、但是管辖区域受到限制的文件。

要点: 您的产地国对加密软件的进口、拥有、使用或再次出口到其他国家可能有一些限制。 在下载或使用不受限制的策略文件之前，必须针对您所在国家或地区的法律进行检查。 检查相应的规章以及对加密软件进行进口、拥有、使用和再次出口的相关政策，从而确定是否允许下载或使用这些文件。

修改信任提供程序以允许客户机连接至任何服务器

该示例说明了如何添加信任提供程序以及从 MQTT 客户机代码中引用此信任提供程序。 该示例对客户机或服务器不执行认证。 最终获得的 SSL 连接已加密，但是未进行认证。

第 103 页的图 20 中的代码片段将为 MQTT 客户机设置 AcceptAllProviders 信任提供程序和信任管理器。

```
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);
```

图 *20: MQTT* 客户机代码段

```
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());
    }
```

图 *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() };
        }
```

图 *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);
  }
}
```

图 *23: AcceptAllX509TrustManager.java*

遥测通道 JAAS 配置

配置 JAAS 以认证由客户机发送的**用户名**。

WebSphere MQ 管理员负责配置哪些 MQTT 通道使用 JAAS 对客户机进行认证。 指定每个要执行 JAAS 认证的通道的 JAAS 配置名称。 这些通道可以全部使用同一个 JAAS 配置，也可以使用不同的 JAAS 配置。 配置在 *WMQData directory*\qmgrs*qMgrName*\mqxr\jaas.config 中定义。

jaas.config 文件按 JAAS 配置名称进行组织。 在每个配置名称下是一个登录配置列表；请参阅 第 105 页的图 24。

JAAS 提供了四个标准登录模块。 标准 NT 和 UNIX 登录模块都是有限值。

JndiLoginModule
针对在 JNDI（Java 命名和目录接口）下配置的目录服务进行认证。

Krb5LoginModule
使用 Kerberos 协议进行认证。

NTLoginModule
使用当前用户的 NT 安全性信息来进行认证。

UnixLoginModule
使用当前用户的 UNIX 安全性信息来进行认证。

使用 NTLoginModule 或 UnixLoginModule 时存在的问题是：遥测 (MQXR) 服务使用 mqm 身份而不是 MQTT 通道的身份来运行。 mqm 是传递给 NTLoginModule 或 UnixLoginModule 以进行认证的身份，而不是客户机的身份。

要解决此问题，可编写您自己的登录模块，或者使用其他标准登录模块。 随 WebSphere MQ Telemetry 一起提供了一个样本 JAASLoginModule.java。 它是 javax.security.auth.spi.LoginModule 接口的实现。 使用它来开发您自己的认证方法。

您提供的任何新 LoginModule 类都必须在遥测 (MQXR) 服务的类路径上。 请不要将您的类放在类路径中的 WebSphere MQ 目录下。 创建您自己的目录，并定义遥测 (MQXR) 服务的整个类路径。

您可以通过在 service.env 文件中设置类路径来扩充遥测 (MQXR) 服务所使用的类路径。 CLASSPATH 必须使用大写字母，class path 语句只能包含文字。 不能在 CLASSPATH 中使用变量；例如，CLASSPATH=%CLASSPATH% 就不正确。 遥测 (MQXR) 服务设置其自己的类路径。 将 service.env 中所定义的 CLASSPATH 添加至此类路径。

遥测 (MQXR) 服务提供两个回调，这两个回调将返回与 MQTT 通道连接的客户机的 Username 和 Password。 **用户名** 和 **密码** 在 MqttConnectOptions 对象中设置。 请参阅第 105 页的图 25 以了解如何访问**用户名**和**密码**的示例。

示例

具有一个已命名的配置 MQXRConfig 的 JAAS 配置文件示例。

```
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";
};
```

图 *24:* 样本 *jaas.config* 文件

已编写的 JAAS 登录模块示例，用于接收 MQTT 客户机所提供的**用户名**和**密码**。

```
  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;
  }
```

图 *25:* 样本 *JAASLoginModule.Login()* 方法

客户机编程概念

本部分中所描述的概念可帮助您了解 MQTT protocol V3.1 的 Java 客户机。 这些概念是对随 com.ibm.micro.client.mqttv3 软件包一起提供的 API 文档的补充。

com.ibm.micro.client.mqttv3 包含为 MQTT V 3.1 协议的 Java 实现提供公用方法的类。 com.ibm.micro.client.mqttv3 软件包以及用于实现 Java SE 和 ME 协议的随附软件包随 IBM WebSphere MQ Telemetry 的安装一起提供。

要开发和运行 MQTT 客户机，需要在客户机设备上复制或安装这些包。 您无需安装独立的客户机运行时。

客户机的许可条件与其连接的服务器相关。

Java 客户机是 MQTT protocol V3.1 的参考实施。 可以使用不同的语言来实现您自己的适合于不同设备平台的客户机。 有关详细信息，请参阅 MQ 遥测传输格式与协议。

软件包 `com.ibm.micro.client.mqttv3` 的客户机 API 文档不对客户机连接到哪个服务器做任何假定。 有关 MQTT 客户机库的客户机 API 文档的链接，请参阅 MQTT 客户机编程参考。 当连接不同的服务器时，客户机的行为可能略有不同。 随后的描述介绍了客户机在连接到 IBM WebSphere MQ 遥测 (MQXR) 服务时的行为。

JavaScript 的 MQTT 消息传递客户机和 Web 应用程序

直到最近，对 Web 应用程序进行编程和创建消息传递应用程序都具有独立的规程。 无论您之前有无经验，同时使用 JavaScript 和消息传递功能都有着明显的优势。 您在将消息传递应用程序编码为 Web 应用程序时，可以将其拉入任何最新浏览器并在此浏览器上予以运行。 如果您更改此应用程序，那么只要刷新该浏览器，就会拉入最新版本。 该浏览器还会照管安全性以及消息的可靠传输。

使用 Web 应用程序如何方便应用程序部署

如果您拥有在 IBM WebSphere MQ（举例而言）上开发和部署传统消息传递应用程序的经验，那么您可能熟悉以下部署流程：

1. 系统管理员安装或嵌入客户机库。
2. 系统管理员安排将消息传递应用程序分发到最终用户并安装在其本地系统上。
3. 代码变更时，系统管理员重复先前的步骤（所以变更管理比较复杂）。

如果您将消息传递应用程序编码为 Web 应用程序，那么这是部署流程：

1. 系统管理员在某个 URL 提供 Web 应用程序和客户机库。
2. 最终用户的浏览器将 Web 应用程序和客户机库一起拉入。
3. 代码变更时，将在刷新浏览器时获取已更新的版本（所以变更管理比较简单）。

您为何可能希望直接从 Web 应用程序中的浏览器使用消息传递

如果您拥有使用 JavaScript 对应用程序编程方面的经验，那么您可能有兴趣知道消息传递系统（如 IBM WebSphere MQ）所提供的益处：

- 如果您通过消息传递系统发送和接收消息，那么该系统负责确保这些消息已传递。
- 因为消息传递系统照管传递，所以 Web 应用程序能够“即发即弃”(fire and forget)。 这极大地简化了编程逻辑。 如果为您传递消息，那么您的代码无需检查这些消息是否到达目标位置。 您的应用程序不再需要处理接收确认，或者保存未传递的消息并稍后予以重试。
- 消息传递系统提供事件驱动的消息传递。 您的客户机应用程序不再需要发送请求，然后连续地轮询以获得响应。 取而代之的是，消息传递服务器在发生引起兴趣的事件时向您的客户机应用程序发送消息。 这也意味着，一旦发生此事件就会提醒您的客户机应用程序，而不是等到应用程序下次轮询服务器时才这么做。
- 事件驱动的消息传递还会极大地减少托管了客户机应用程序的设备上的负载、浏览器与消息传递服务器之间的网络流量以及消息传递服务器上的负载。 这越来越重要，因为越来越多的系统在移动设备上运行并且跨无线网络进行连接。

各部分如何融合在一起

JavaScript 的 MQTT 消息传递客户机包括客户机库以及使用该库的示例 Web 应用程序。 您对自己的使用该库的 Web 应用程序进行编码。 然后，由 MQ 队列管理器（如下图中所示）或由应用程序服务器（举例而言）在所选的 URL 提供 Web 应用程序和客户机库。 浏览器拉入 Web 应用程序和客户机库，Web 应用程序然后使用浏览器来连接到诸如 IBM WebSphere MQ Telemetry 或 IBM MessageSight 的 MQTT 服务器并与此服务器交换消息。

以下是流程：

1. 浏览器的各实例都刷新其与提供 Web 应用程序的 URL 的连接，然后 Web 应用程序和客户机库的最新版本将装入到浏览器中。

2. Web 应用程序使用 WebSocket protocol 上的 MQTT 来连接到队列管理器，然后预订感兴趣的主题。
3. 队列管理器使用同一连接将匹配此预订的消息发送回 Web 应用程序。

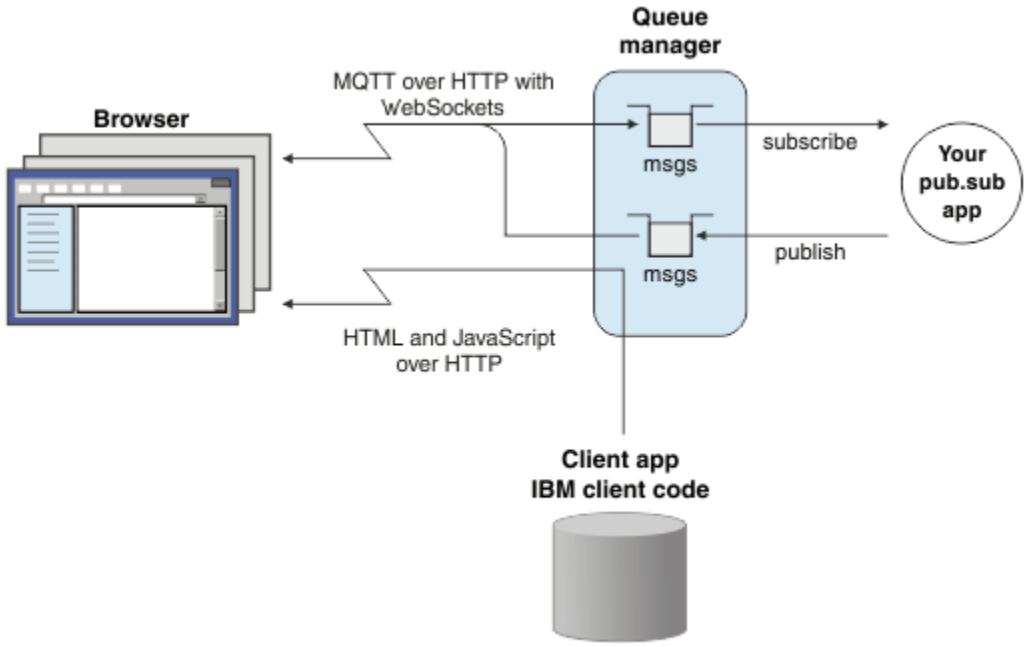


图 *26:* 将 *JavaScript* 的 *MQTT* 消息传递客户机用于发布和预订消息传递

Web 应用程序包含应用程序逻辑以及 MQTT 服务器的 URL。 在浏览器中打开的情况下，该应用程序连接到 MQTT 服务器，创建其需要的预订，然后等待接收事件驱动的警报并对其采取行动。

Web 应用程序将 MQTT 用作传输协议（在 WebSockets 上运行）来进行连接。 大多数现代浏览器都可以建立 WebSockets 连接。 通过使用 WebSockets， Web 应用程序可以通过接受 HTTP 和 WebSocket protocol 的防火墙传递消息，并且可以像使用基于 IP 的 TCP 一样发送数据包 (称为 “帧”)。

Web 应用程序发送的消息到达 MQTT 服务器中时，服务器端应用程序只是将其视为消息。 它不知道此消息来自浏览器。

管理和控制 MQTT 服务器

MQTT 服务器处理消息传递的服务器端复杂性。 它确保对其从 Web 应用程序收到的消息的传递，并且托管对 Web 应用程序进行响应的发布和预订应用程序。 对于任何 MQTT 服务器，您都需要完成以下步骤:

- 创建服务器。
- 选取端口。
- 定义新 MQTT 通道。
- 将客户机 Web 应用程序配置为跨新 MQTT 通道连接到所选端口。

您还需要向浏览器提供 Web 应用程序可执行 JavaScript。 如果您要使用 IBM WebSphere MQ Telemetry，缺省情况下 MQTT 服务器会使用 Web 应用程序用于连接到 MQTT 服务器的同一 MQTT 通道为您执行此操作。 如果您要尝试 MQTT，那么这可以帮助您进行快速启动和运行。 对于生产用途，尤其是在高吞吐量环境中，您可能会首选使用专用应用程序服务器（如 WebSphere Application Server）在另一个通道上提供 Web 应用程序可执行 JavaScript。

注: 因为这是为高吞吐量环境所设计，所以 IBM MessageSight 预期您会执行此操作。

例如，如果您要使用 IBM WebSphere MQ Telemetry，那么请使用 IBM WebSphere MQ Explorer **新建遥测通道**向导来完成以下步骤:

1. 创建服务器。

2. 选取端口（缺省情况下为 1883）。
3. 定义新 MQTT 通道。
4. 将客户机 Web 应用程序配置为跨新 MQTT 通道连接到所选端口。

Web 应用程序可执行 JavaScript 还（可选）通过队列管理器在同一通道上提供。 为此，队列管理器必须同时支持 MQTT 和 HTTP。 如果您已经拥有为 MQTT 配置的队列管理器，那么可以使用 MQSC 命令行工具变更通道定义中的协议以同时支持 MQTT 和 HTTP。 请参阅 ALTER CHANNEL。

Web 应用程序和 JavaScript 的 MQTT 消息传递客户机客户机库存储在磁盘上的由应用程序服务器或队列管理器所定义的结构中。 如果您要使用 IBM WebSphere MQ Telemetry，那么 Web 应用程序和客户机库会存储在以下目录结构中：

```
MQINSTALL
  |
  mqxr
  |  |
  |  SDK
  |    |
  |    WebContent
  |       sample web app (the “Web Messaging Utility” sample HTML pages)
  |       |
  |       WebSocket
  |          IBM client library (the messaging client JavaScript classes)
  |
  qmgrs
     |
     qmgr_name
        |
        mqxr
           |
           WebContent
              |
              your_client_app (your own JavaScript pages)
```

样本 Web 应用程序和客户机库存储在 *MQINSTALL*/mqxr/SDK/WebContent 目录中。 此目录中的资料由所有队列管理器提供。 如果您不希望您的用户查看和使用所有这些资料，那么应该创建自己的应用程序版本。 为了使此应用程序或您自己的替换应用程序在特定队列管理器上可用，请将应用程序置于 *MQINSTALL*/qmgrs/*qmgr_name*/mqxr/WebContent 目录中。 要选择要在 URL 处提供的应用程序和关联 JavaScript 类，队列管理器首先在其自己的 WebContent 目录中查找，然后在全局 WebContent 目录中查找。 在先前的示例目录树中，队列管理器提供 *your_client_app* 以及 JavaScript 类的全局副本。

要使队列管理器停止提供 Web 应用程序可执行文件，或者修改队列管理器查找可执行文件的位置，请配置 **webcontentpath** 属性并将其添加到 mqxr.properties 文件。 请参阅 MQXR 属性。

相关概念

第 108 页的『如何使用 JavaScript 对消息传递应用程序进行编程』

相关任务

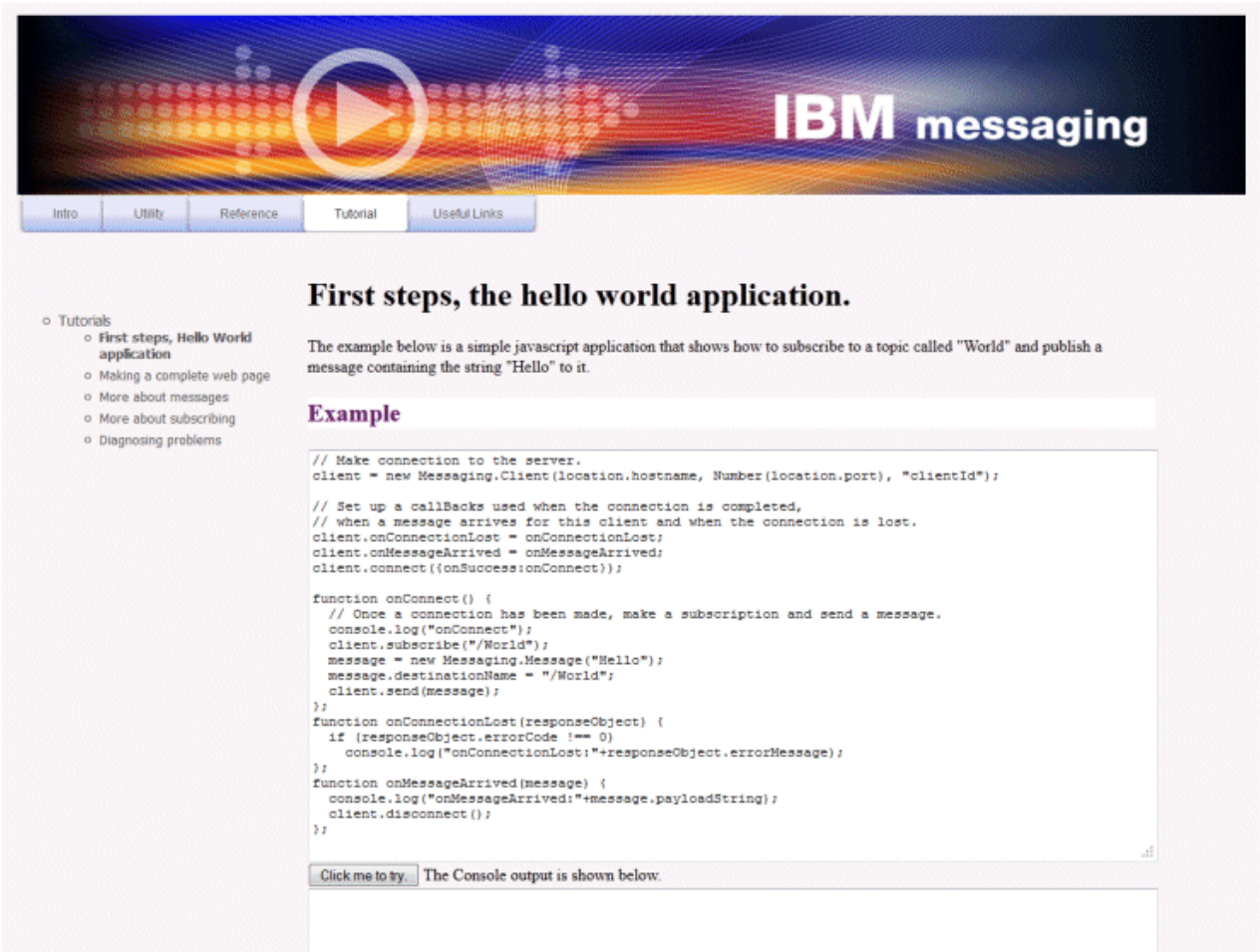
第 70 页的『通过 SSL 连接 JavaScript 的 MQTT 消息传递客户机 和 WebSockets』
通过将 JavaScript 的 MQTT 消息传递客户机 样本 HTML 页面与 SSL 和 WebSocket protocol 配合使用，将 Web 应用程序安全地连接到 IBM WebSphere MQ 。

第 21 页的『JavaScript 的 MQTT 消息传递客户机入门』
您可以通过显示消息传递客户机样本主页并浏览其链接到的资源来开始使用 JavaScript 的 MQTT 消息传递客户机。 要显示此主页，请将 MQTT 服务器配置为接受来自 MQTT 消息传递客户机样本 JavaScript 页面的连接，然后将已在服务器上配置的 URL 输入 Web 浏览器中。 JavaScript 的 MQTT 消息传递客户机会在您设备上自动启动，然后消息传递客户机样本主页将显示。 该页面包含指向实用程序、编程接口文档、教程及其他有用信息的链接。

如何使用 JavaScript 对消息传递应用程序进行编程

JavaScript 的 MQTT 消息传递客户机包含一个教程，该教程演示如何创建简单的发布和预订 Web 应用程序。 通过考察“First steps, Hello world”应用程序代码，您可以对用于消息传递的 Web 应用程序编程机制有个基本的了解。

如果您迄今为止的经验主要在开发和部署传统消息传递应用程序方面，那么还可能会觉得 第 109 页的『JavaScript 编码提示』部分很有帮助。 如果您是有经验的 JavaScript 开发人员但是对于消息传递是新手，那么将在第 111 页的『消息传递基本信息』部分中找到对关键消息传递概念的简介。



JavaScript 编码提示

如果您习惯于开发消息传递应用程序，但是对于 Web 应用程序是新手，那么可能会觉得以下提示很有帮助：

将每个事件的代码都封装在 onSuccess 回调中

您对消息传递应用程序进行编码时，请按以下顺序对以下事件进行编码：

1. 连接
2. 预订
3. 发布
4. 接收消息

JavaScript 的 MQTT 消息传递客户机 API 完全异步，这意味着您的应用程序线程在等待诸如连接或预订之类的调用生效时不会阻塞。 取而代之的是，这些调用通过调用 onSuccess 或 onFailure 回调来指示其完成。 为了确保每个事件均已在触发下一个事件之前完成，您需要将每个事件的代码均封装在 onSuccess 回调中。 例如，JavaScript 应用程序可能会在创建了连接之前就从执行连接调用这一操作中返回。 为了确保在您预订之前已进行连接，您需要将预订代码放置在用于连接的 onSuccess 回调中。

“First steps, Hello world”应用程序代码使用此方法。

将应用程序代码嵌入 HTML 标记内

以下是示例 JavaScript 页面：

以下是上一页面的源代码，以显示应用程序代码如何嵌入 HTML 标记内：

```
<!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>
```

消息传递基本信息

以下是针对新近涉足消息传递的 Web 应用程序开发人员的一些背景消息传递信息：

异步和即发即弃消息传递。

MQTT 协议支持可靠的传送和即发即弃的传输。 在此协议中，消息传递是异步的：应用程序将消息传递到客户机 API，然后并不会执行进一步操作来确保消息已传递。 此方法称为即发即弃。 响应可用时，会自动将其发送到应用程序。

异步传递会从任何服务器连接中并从对消息的等待中释放应用程序。 交互模式与电子邮件相似，但在应用程序编程方面进行了优化。

另请参阅 第 5 页的『MQTT 简介』 的 “MQTT 协议” 部分

发布和预订消息传递的概述。

信息的提供者称为发布者。 发布者提供关于主题的信息，而无需知道关于对该信息感兴趣的应用程序的任何信息。 发布者选择主题，这是关于特定主题的消息的容器。 然后，发布者将该主题的每个信息都生成为一条消息（称为发布内容），然后将其发布到关联的主题。

信息的使用者称为订户。 订户创建对其感兴趣的主题的预订。 新消息发布到主题时，该消息会转发给该主题的所有订户。 订户可以创建多个预订，并可以接收来自许多不同发布者的信息。

另请参阅 IBM WebSphere MQ 发布/预订消息传递简介

预订与主题如何匹配。

如果您要将 IBM WebSphere MQ 用作 MQTT 服务器，那么需要了解 IBM WebSphere MQ 如何指定主题。 在 IBM WebSphere MQ 中，发布者创建消息，并使用与发布内容的主题最相符的主题字符串来发布该消息。 要接收发布内容，订户创建带有模式匹配主题字符串的预订，该字符串用于选择发布内容主题。 队列管理器将发布内容传递给其预订与发布内容主题匹配并有权接收发布内容的订户。

通常情况下，主题以分层方式组织成主题树，同时使用“/”字符在主题字符串中创建子主题。 主题是主题树中的节点。 主题可以是没有进一步子主题的叶节点，也可以是具有子主题的中间节点。 订户可以使用通配符来一次预订多个主题。 例如，对 /sport/tennis 的预订只会获取发布到网球子主题的消息，而对 /sport/# 的预订会获取发布到 /sport 的任何子主题的消息。

另请参阅主题、主题树和通配符方案。

相关概念

第 106 页的『JavaScript 的 MQTT 消息传递客户机和 Web 应用程序』

相关任务

第 70 页的『通过 SSL 连接 JavaScript 的 MQTT 消息传递客户机 和 WebSockets』
通过将 JavaScript 的 MQTT 消息传递客户机 样本 HTML 页面与 SSL 和 WebSocket protocol 配合使用，将 Web 应用程序安全地连接到 IBM WebSphere MQ 。

第 21 页的『JavaScript 的 MQTT 消息传递客户机入门』
您可以通过显示消息传递客户机样本主页并浏览其链接到的资源来开始使用 JavaScript 的 MQTT 消息传递客户机。 要显示此主页，请将 MQTT 服务器配置为接受来自 MQTT 消息传递客户机样本 JavaScript 页面的连接，然后将已在服务器上配置的 URL 输入 Web 浏览器中。 JavaScript 的 MQTT 消息传递客户机会在您设备上自动启动，然后消息传递客户机样本主页将显示。 该页面包含指向实用程序、编程接口文档、教程及其他有用信息的链接。

MQTT 客户机应用程序中的回调和同步

MQTT 客户机编程模型广泛地使用线程。 这些线程尽可能使 MQTT 客户机应用程序与在服务器之间传输消息的延迟脱钩。 发布，传递令牌和连接丢失事件将传递到实现 MqttCallback 的回调类中的方法。

回调

MqttCallback 接口有三种回调方法；请参阅 Callback.java 中的示例实现。

connectionLost(java.lang.Throwable cause)

当通信错误导致连接断开时，就会调用 connectionLost。 在已经建立连接之后，如果由于服务器上发生错误而导致服务器断开连接，那么也会调用此方法。 服务器错误被记录到队列管理器错误日志中。 服务器将断开与客户机的连接，并且客户机将调用 MqttCallback.connectionLost。

在客户机应用程序所在的同一线程上作为异常抛出的唯一远程错误就是 MqttClient.connect 产生的异常。 建立连接后服务器检测到的错误将作为 throwables 报告回 MqttCallback.connectionLost 回调方法。

导致 connectionLost 的典型服务器错误是权限错误。 例如，遥测服务器试图代表一个未被授权发布主题的客户机来发布主题。 导致将 MQCC_FAIL 条件代码返回到遥测服务器的任何情况都会导致断开连接。

deliveryComplete(MqttDeliveryToken token)

MQTT 客户机调用 deliveryComplete 以将传递令牌传递回客户机应用程序; 请参阅 第 115 页的『传递令牌』。 通过使用传递令牌，回调可以使用 token.getMessage 方法来访问已发布的消息。

当应用程序回调在被 deliveryComplete 方法调用后将控制权返回给 MQTT 客户机时，将完成交付。 在完成传递之前，持久类将保留服务质量为 QoS 1 或 QoS 2 的消息。

调用 deliveryComplete 是应用程序与持久性类之间的同步点。 决不会对同一条消息两次调用 deliveryComplete 方法。

当应用程序回调从 deliveryComplete 返回到 MQTT 客户机时，客户机会调用 MqttClientPersistence.remove 以获取 QoS 为 1 或 2 的消息。 MqttClientPersistence.remove 删除已发布消息的本地存储副本。

从事务处理角度来说，调用 deliveryComplete 是用于落实传递的单阶段事务。 如果在回调期间处理失败，那么在重新启动客户机时，将再次调用 MqttClientPersistence.remove，以删除已发布的消息的本地副本。 不会再次调用此回调。 如果要使用回调来存储已传递的消息的日志，那么将无法使该日志与 MQTT 客户机同步。 如果您想可靠地存储日志，请更新 MqttClientPersistence 类中的日志。

传递令牌和消息由主应用程序线程和 MQTT 客户机引用。 完成传递后，MQTT 客户机将取消引用 MqttMessage 对象；客户机断开连接时，它将取消引用传递令牌对象。 在完成传递之后，如果客户

机应用程序取消对 MqttMessage 对象的引用，那么可以对此对象进行垃圾回收。 在会话断开连接之后，可以对传递令牌进行垃圾回收。

在已经发布消息之后，可以获取 MqttDeliveryToken 和 MqttMessage 属性。 如果您在已经发布消息之后尝试设置任何 MqttMessage 属性，那么结果是不确定的。

如果客户机重新连接到具有相同 ClientIdentifier 的先前会话，那么 MQTT 客户机将继续处理传递应答; 请参阅 第 114 页的『清除会话』。 对于先前会话， MQTT 客户机应用程序必须将 MqttClient.CleanSession 设置为 false ，并在新会话中将其设置为 false 。 MQTT 客户机可在新会话中创建新的传递令牌和消息对象，以供暂挂的传递使用。 它将使用 MqttClientPersistence 类来恢复对象。 如果应用程序客户机仍然引用了旧的传递令牌和消息，请取消对它们的引用。 对于在先前会话中启动的以及在此会话中完成的任何传递，会在这些传递的新会话中调用应用程序回调。

在应用程序客户机连接之后，当完成暂挂的传递时，就会调用应用程序回调。 在应用程序客户机连接之前，它可以使用 MqttClient.getPendingDeliveryTokens 方法来检索暂挂的传递。

请注意，客户机应用程序最初创建了已发布的消息对象及其有效内容字节数组。 MQTT 客户机引用这些对象。 由 token.getMessage 方法中的传递令牌返回的消息对象不一定是客户机所创建的同一消息对象。 如果新的 MQTT 客户机实例重新创建了传递令牌，那么 MqttClientPersistence 类将重新创建 MqttMessage 对象。 为了保持一致性，无论消息对象是由应用程序客户机还是由 MqttClientPersistence 类创建的，如果 token.isCompleted 为 true，那么 token.getMessage 将返回 null。

messageArrived(MqttTopic topic, MqttMessage message)

当客户机的与预订主题相匹配的预订到达时，就会调用 messageArrived。 topic 是发布主题，而不是预订过滤器。 如果过滤器中包含通配符，那么这两者可能不同。

如果此主题与客户机所创建的多个预订相匹配，那么客户机将收到此发布的多个副本。 如果客户机发布至它也预订了的某个主题，那么它将收到它自己的发布的副本。

如果使用值为 1 或 2 的 QoS 发送消息，那么该消息将由 MqttClientPersistence 类存储，之后 MQTT 客户机将调用 messageArrived。 messageArrived 的运行方式类似 deliveryComplete：它只能针对发布调用一次，当 messageArrived 返回到 MQTT 客户机时，此发布的本地副本将由 MqttClientPersistence.remove 除去。 当 messageArrived 返回到 MQTT 客户机时， MQTT 客户机会删除其对主题和消息的引用。 如果应用程序客户机尚不具备对对象的引用，那么会对主题和消息对象进行垃圾回收。

回调、线程技术和客户机应用程序同步

MQTT 客户机会将独立线程上的回调方法调用到主应用程序线程。 客户机应用程序不会为回调创建线程，它由 MQTT 客户机创建。

MQTT 客户机将同步回调方法。 一次只有回调方法的一个实例运行。 通过同步，很容易更新一个用于清点已经传递了哪些发布的对象。 一次只运行 MqttCallback.deliveryComplete 的一个实例，因此，更新记录而不进一步进行同步是安全的。 一次只有一个发布到达也是这种情况。 messageArrived 方法中的代码可以更新对象而不使此对象同步。 如果您正在另一个线程中引用记录或者要更新的对象，那么使此记录或对象同步。

传递令牌提供了主应用程序线程与发布的传递之间的同步机制。 token.waitForCompletion 方法将一直等到完成了传递特定发布，或者等到可选超时已到期。 可以使用 token.waitForCompletion 并通过两种简单的方法来一次处理一个发布：

1. 暂停应用程序客户机，直到完成了发布的传递；请参阅 PubSync.java。
2. 与 MqttCallback.deliveryComplete 方法同步。 只有当 MqttCallback.deliveryComplete 返回到 MQTT 客户机时，token.waitForCompletion 才会恢复。 通过使用此机制，在代码运行于主应用程序线程中之前，可以使 MqttCallback.deliveryComplete 中正在运行的代码同步。

如果您想进行发布而不等待传递每个发布，但您想确认何时已经传递了所有发布，情况会怎样呢？ 如果您在单个线程上发布，那么要发送的最后一个发布也是要传递的最后一个发布。

使发送至服务器的请求同步

表 *4:* 导致将请求发送至服务器的方法的同步行为.

此表列出了 MQTT Java 客户机中向服务器发送请求的方法。 对于每一种方法，该表描述了一些条件，在这些条件下，方法或者等待或者返回，并描述了方法的等待时间。

方法	同步	超时时间间隔
`MqttClient.Connect`	等待与服务器建立连接。	缺省设置为 30 秒，或者由参数进行设置。
`MqttClient.Disconnect`	等待 MQTT 客户机完成所有必需的工作，并等待 TCP/IP 会话断开连接。	缺省设置为 30 秒，或者由参数进行设置。
`MqttClient.Subscribe`	等待订阅请求完成。	缺省设置为 30 秒，或者由参数进行设置。
`MqttClient.UnSubscribe`	等待取消预订请求完成。	缺省设置为 30 秒，或者由参数进行设置。
`MqttClient.Publish`	将请求传递到 MQTT 客户机后，立即返回到应用程序线程。	无。
`MqttDeliveryToken.waitForCompletion`	等待返回传递令牌。	缺省设置无限期，或者由参数进行设置。

清除会话

MQTT 客户机和遥测 (MQXR) 服务都将维护会话状态信息。 状态信息用来确保进行“至少一次”和“刚好一次”传递，以及“刚好一次”接收发布。 会话状态还包括由 MQTT 客户机创建的预订。 运行 MQTT 客户机时，您可以选择维护会话之间的状态信息，也可以选择不维护。 通过在连接之前设置 `MqttConnectOptions.cleanSession` 来更改清除会话方式。

当您使用 `MqttClient.connect` 方法来连接 MQTT 客户机应用程序时，客户机使用客户机标识和服务器地址来标识连接。 服务器将检查是否保存了先前与服务器建立连接时使用的会话信息。 如果先前的会话仍然存在，并且 `cleanSession=true`，那么将清除客户机和服务器中先前的会话信息。 如果 `cleanSession=false`，那么先前的会话将继续。 如果不存在先前的会话，那么将启动新的会话。

注: WebSphere MQ 管理员可以强制关闭打开的会话，并删除所有会话信息。 如果客户机重新打开会话并且 `cleanSession=false`，那么将启动新的会话。

出版物

如果您使用缺省 `MqttConnectOptions`，或者在连接客户机之前将 `MqttConnectOptions.cleanSession` 设置为 `true`，那么在客户机建立连接时，将除去为客户机传递的所有暂挂的发布。

“清除会话”设置对于使用 `QoS=0` 发送的发布不起作用。 对于 `QoS=1` 和 `QoS=2`，使用 `cleanSession=true` 可能会导致丢失发布。

预订

如果在连接客户机之前使用缺省 `MqttConnectOptions` 或将 `MqttConnectOptions.cleanSession` 设置为 `true` ，那么在客户机连接时将除去客户机的任何旧预订。 当客户机断开连接时，将移除客户机在会话期间创建的所有新预订。

如果在连接之前将 `MqttConnectOptions.cleanSession` 设置为 `false` ，那么客户机创建的任何预订都将添加到客户机在连接之前已存在的所有预订。 当客户机断开连接时，所有预订仍保持活动状态。

要了解 cleanSession 属性影响预订的方式，另一种方法就是将它视作模态属性。 在其缺省方式 cleanSession=true 下，客户机仅在会话的作用域内创建预订和接收发布。 在另一种方式 cleanSession=false 下，预订是持久预订。 客户机可以连接和断开连接，而其预订保持活动状态。 当客户机重新连接时，它将接收任何未传递的发布。 在它连接之后，它可以自己修改处于活动状态的预订集。

必须先设置 cleanSession 方式，然后再进行连接; 该方式将持续整个会话。 要更改此属性的设置，必须将客户机断开连接，然后再重新连接客户机。 如果将方式从使用 cleanSession=false 更改为 cleanSession=true，那么将废弃客户机的所有先前预订以及尚未接收的任何发布。

客户机标识

客户机标识是标识 MQTT 客户机的 23 字节字符串。 客户机标识必须在连接到服务器的所有客户机中是唯一的，且不能与服务器上的队列管理器名称相同。 在满足这些约束的条件下，您可以使用任何标识字符串。 必须具备分配客户机标识的过程以及使用所选标识配置客户机的方法，这一点非常重要。

客户机标识在 MQTT 系统管理过程中使用。 由于可能要管理许多客户机，因此您需要能够快速标识特定客户机。 例如，假定某个设备发生了故障，也许客户通过呼叫服务台来通知您。 客户如何标识此设备？ 您如何使该标识与通常跟客户机相连的服务器相关联？ 您必须查阅将每台设备映射至客户机标识和服务器的数据库吗？ 设备的名称会标识它连接至哪个服务器吗？ 当您浏览 MQTT 客户机连接时，每个连接都会使用客户机标识进行标记。 您需要查找一个表以将客户机标识映射至实际设备吗？

客户机标识将标识特定设备、用户或者在客户机中运行的应用程序吗？ 如果客户将故障设备更换为一个新设备，此新设备与旧设备具有相同标识吗？ 您将分配新的标识吗？ 如果您更改实际设备，但是保持使用相同标识，那么会将未完成的发布和活动预订自动传递至新设备。

您如何确保客户机标识是唯一的？ 除了用于生成唯一标识的系统以外，您还必须通过执行可靠过程在客户机上设置标识。 也许客户机设备是一个“黑匣”，没有用户界面。 您在制造这种设备时会使用客户机标识（例如，使用它的 MAC 地址）吗？ 或者，在激活设备之前，您是否会执行软件安装以及用于配置此设备的配置过程？

您可以使用 48 位设备 MAC 地址来创建客户机标识，以使此标识较短并且是唯一的。 如果传输大小并不是关键问题，那么可以使用其余 17 个字节以使地址更容易管理。

传递令牌

当客户机在主题上发布内容时，将创建新的传递令牌。 使用传递令牌来监控发布的传递，或者阻止客户机应用程序，直到完成传递为止。

令牌是一个 MqttDeliveryToken 对象。 它通过调用 MqttTopic.publish() 方法来创建并由 MQTT 客户机保留，直到客户机会话断开连接且完成传递为止。

令牌的常规用法是检查是否已完成传递。 通过使用所返回的令牌来调用 token.waitForCompletion，从而阻塞客户机应用程序直到已完成传递为止。 或者，提供 MqttCallBack 处理程序。 当 MQTT 客户机接收到其期望作为传递发布的一部分的所有应答后，它会调用 MqttCallBack.deliveryComplete，将传递令牌作为参数进行传递。

在完成传递之前，可以通过调用 token.getMessage 从而使用所返回的传递令牌来检查发布。

已完成传递

完成传递的过程为异步，取决于与发布关联的服务质量。

至多一次

QoS=0
从 MqttTopic.publish 返回时，立即完成传递。 会立即调用 MqttCallback.deliveryComplete。

至少一次

QoS=1
从队列管理器接收到有关发布的确认信息时就完成了传递。 接收到确认信息时就会调用 MqttCallback.deliveryComplete。 如果通信速度很慢或者不可靠，那么在调用 MqttCallback.deliveryComplete 之前可能会多次传递消息。

刚好一次

QoS=2
当客户机接收到有关已完成将发布消息发布至订户的消息时就完成了传递。 一旦接收到发布消息，就会调用 `MqttCallback.deliveryComplete`。 它并不会等待完成消息。

在极少数情况下，客户机应用程序可能不会正常从 `MqttCallback.deliveryComplete` 返回到 MQTT 客户机。 但是，您知道已完成传递，因为已经调用了 `MqttCallback.deliveryComplete`。 如果客户机重新启动同一会话，那么不会再次调用 `MqttCallback.deliveryComplete`。

未完成的传递

如果在客户机会话断开连接之后未完成传递，那么您可以再次连接客户机并完成传递。 仅当通过 `MqttConnectionOptions` 属性设置为 `false` 的会话发布了消息时，才能完成传递此消息。

使用同一客户机标识和服务器地址来创建客户机，然后在将 `cleanSession` `MqttConnectionOptions` 属性设置为 `false` 的情况下再次连接。 如果您将 `cleanSession` 设置为 `true`，那么会抛弃暂挂的传递令牌。

可以通过调用 `MqttClient.getPendingDeliveryTokens` 来检查是否有任何暂挂的传递。 在连接客户机之前，可以调用 `MqttClient.getPendingDeliveryTokens`。

“最后的消息”发布

如果 MQTT 客户机连接意外结束，那么可以配置 WebSphere MQ Telemetry 以发送 "last will and testament" 出版物。 预定义发布的内容以及要将它发送至的主题。 “最后的消息”是一种连接属性。 请在连接客户机之前创建此消息。

为最后的消息创建主题。 您可以创建主题，例如 `MQTTManagement/Connections/`*`server URI`*`/`*`client identifer`*`/Lost`。

使用 `MqttConnectionOptions.setWill(MqttTopic lastWillTopic, byte [] lastWillPayload, int lastWillQos, boolean lastWillRetained)` 方法设置 "last will and testament"。

考虑在 `lastWillPayload` 消息中创建一个时间戳记。 请包括用于帮助标识客户机以及连接环境的其他客户机信息。 将 `MqttConnectionOptions` 对象传递至 `MqttClient` 构造函数。

将 `lastWillQos` 设置为 1 或 2，以使消息持久保存在 IBM WebSphere MQ 中并且确保传递。 要保留最后的断开连接信息，请将 `lastWillRetained` 设置为 `true`。

如果连接意外结束，那么会将“最后的消息”发布发送至订户。 如果连接结束，而客户机未调用 `MqttClient.disconnect` 方法，就会发送此发布。

要监控连接，请使用其他发布来补充“最后的消息”发布，以记录连接和程序化断开连接。

MQTT 客户机中的消息持久性

如果发布消息是使用“至少一次”或者“刚好一次”服务质量来发送的，那么这些消息具有持久性。 您可以在客户机上实施自己的持久性机制，也可以使用随客户机一起提供的缺省持久性机制。 将发布发送至客户机或者从客户机发送发布时都存在持久性。

在 MQTT 中，消息持久性具有两个方面：如何传输消息，以及消息是否作为持久消息在 MQTT 服务器中排队。

1. MQTT 客户机将消息持久性与服务质量进行了耦合。 根据您为消息选择的服务质量，使消息成为持久消息。 要实现必需的服务质量，必须具备消息持久性。

   如果指定“至多一次”(QoS=0)，那么客户机将在发布消息的同时废弃此消息。 如果在消息的上游处理过程中发生了任何故障，将不会再次发送此消息。 即使客户机保持活动状态，也不会再次发送此消息。 QoS=0 消息的行为与 IBM WebSphere MQ 快速非持久消息相同。
   如果客户机采用 QoS 1 或者 QoS 2 来发布消息，那么此消息为持久消息。 此消息存储在本地，并且仅在不再需要它来确保“至少一次”QoS=1 或“刚好一次”QoS=2 传递时才从客户机中除去。

2. 如果消息被标记为 QoS 1 或 2，那么它将作为持久消息来排队。 如果它被标记为 QoS=0，那么将作为非持久消息来排队。 在 IBM WebSphere MQ 中，除非消息通道具有设置为 FAST 的 NPMSPEED 属性，否则非持久消息将在队列管理器之间 "正好传输一次"。

持久发布内容存储在客户机上，直到其被客户机应用程序接收为止。 对于 QoS=2，当应用程序回调返回控制时，将从客户机上废弃发布。 对于 QoS=1，如果发生了故障，那么应用程序可能会再次接收到发布。 对于 QoS=0，回调接收到发布不会超过一次。 如果发生了故障，或者在发布时客户机断开连接，那么回调可能不会接收到发布。

当您预订主题时，可以降低订户用来接收消息的 QoS，使它与其持久性功能相匹配。 将使用订户请求的最高的 QoS 发送在更高的 QoS 的情况下创建的发布。

存储消息

在不同的小型设备上，数据存储器的实现的变化很大。 受 MQTT 客户机管理的存储器中临时保存的持久消息模型可能太慢，或需要太大的存储空间。 在移动设备中，移动操作系统可提供充分适用于 MQTT 消息的存储服务。

要在符合小型设备限制的同时提供灵活性，MQTT 客户机有两个持久性接口。 该接口可定义存储持久消息的过程中涉及的操作。 在 Java 的 MQTT 客户机 的 API 文档中描述了这些接口。 有关 MQTT 客户机库的客户机 API 文档的链接，请参阅 MQTT 客户机编程参考。 可以根据设备需求来实现这些接口。 在 Java SE 上运行的 MQTT 客户机具有在文件系统中存储持久消息的接口的缺省实现。 它使用 `java.io` 包。 客户机还具有 Java ME `MqttDefaultMIDPPersistence` 的缺省实现。

持久性类

MqttClientPersistence
将 `MqttClientPersistence` 实现实例作为 `MqttClient` 构造函数的参数传递到 MQTT 客户机。 如果在 `MqttClient` 构造函数中省略了 `MqttClientPersistence` 参数，那么 MQTT 客户机将使用类 `MqttDefaultFilePersistence` 或 `MqttDefaultMIDPPersistence` 存储持久消息。

MqttPersistable
`MqttClientPersistence` 使用存储关键字来获取和放置 `MqttPersistable` 对象。 如果您未使用 `MqttDefaultFilePersistence` 或 `MqttDefaultMIDPPersistence`，那么必须提供 `MqttPersistable` 的实现以及 `MqttClientPersistence` 的实现。

MqttDefaultFilePersistence

MQTT 客户机提供了 `MqttDefaultFilePersistence` 类。 如果您在客户机应用程序中将 `MqttDefaultFilePersistence` 实例化，那么可以提供相应目录来将持久消息作为 `MqttDefaultFilePersistence` 构造函数的一个参数进行存储。

或者，MQTT 客户机可实例化 `MqttDefaultFilePersistence`，并将文件放在缺省目录中。 目录的名称为 *`client identifier`*`-tcp` *`hostname portnumber`*。 将从目录名称字符串中除去 `"\", "\\", "/", ":"` 和 `" "`。

目录的路径是系统属性 `rcp.data` 的值。 如果未设置 `rcp.data`，那么路径是系统属性 `usr.data` 的值。

`rcp.data` 是与 OSGi 或 Eclipse 富客户机平台 (RCP) 安装相关联的属性。

`usr.data` 是启动应用程序的 Java 命令在其中启动的目录。

MqttDefaultMIDPPersistence
`MqttDefaultMIDPPersistence` 有一个缺省构造函数，但没有参数。 它使用 `javax.microedition.rms.RecordStore` 包来存储消息。

出版物

出版物是与主题字符串关联的 `MqttMessage` 实例。 MQTT 客户机可创建发布内容以发送至 IBM WebSphere MQ，并预订 IBM WebSphere MQ MQ 上的主题以接收发布内容。

`MqttMessage` 将字节数组作为其有效内容。 其目的在于使消息尽可能小。 MQTT 协议允许的最大消息长度为 250 MB。

通常，MQTT 客户机程序使用 java.lang.String 或 java.lang.StringBuffer 处理消息内容。 为了方便起见，MqttMessage 类使用 toString 方法将其有效内容转换为字符串。 要从 java.lang.String 或 java.lang.StringBuffer 来创建字节数组有效内容，可使用 getBytes 方法。

getBytes 方法将字符串转换为平台的缺省字符集。 缺省字符集通常为 UTF-8。 仅包含文本的 MQTT 发布通常使用 UTF-8 编码。 使用 getBytes("UTF8") 方法来覆盖缺省字符集。

在 IBM WebSphere MQ 中，会接收 MQTT 发布作为 jms-bytes 消息。 该消息包含一个 MQRFH2 文件夹（包含 <mqtt>）和一个 <mqps> 文件夹。 <mqtt> 文件夹中包含 clientId 和 qos，但是将来可能会更改此内容。

MqttMessage 还有另外三个属性：服务质量、是否保留了此消息以及此消息是否重复。 仅当服务质量为“至少一次”或者“刚好一次”时才设置重复标志。 如果先前已发送消息且 MQTT 客户机未作足够快速的应答，将会把重复属性设置为 true 来重新发送消息。

正在发布

要在 MQTT 客户机应用程序中创建发布，请创建 MqttMessage。 设置其有效内容、服务质量以及是否保留，并调用 MqttTopic.publish(MqttMessage message) 方法；将返回 MqttDeliveryToken 且完成发布的过程为异步。

或者， MQTT 客户机可以在创建发布时从 MqttTopic.publish(byte [] payload, int qos, boolean retained) 方法上的参数为您创建临时消息对象。

如果发布具有“至少一次”或“刚好一次”服务质量（QoS=1 或 QoS=2），那么 MQTT 客户机将调用 MqttClientPersistence 接口。 在将传递令牌返回给应用程序之前，它将调用 MqttClientPersistence 以存储消息。

应用程序可以选择使用 MqttDeliveryToken.waitForCompletion 方法来一直阻塞到将消息传递至服务器为止。 或者，应用程序可以继续运行而不进行阻塞。 如果要查看是否已传递发布并且未受到阻止，请向 MQTT 客户机注册实施 MqttCallback 的回调类实例。 成功发布传递之后，MQTT 客户机会调用 MqttCallback.deliveryComplete 方法。 根据服务质量不同，对于 QoS=0，几乎可以立即进行传递；而对于 QoS=2，可能要花一些时间。

使用 MqttDeliveryToken.isComplete 方法来轮询是否完成了传递。 当 MqttDeliveryToken.isComplete 的值为 false 时，可以调用 MqttDeliveryToken.getMessage 以获取消息内容。 如果调用 MqttDeliveryToken.isComplete 获得的结果为 true，说明已废弃此消息，调用 MqttDeliveryToken.getMessage 将抛出空指针异常。 MqttDeliveryToken.getMessage 与 MqttDeliveryToken.isComplete 之间没有内置同步。

如果客户机在接收所有暂挂的传递令牌之前断开连接，那么客户机的新实例在连接之前可以查询暂挂的传递令牌。 在客户机连接之前，没有完成新的传递，并且调用 MqttDeliveryToken.getMessage 很安全。 使用 MqttDeliveryToken.getMessage 方法来弄清楚尚未传递哪些发布。 如果您在 MqttConnectOptions.cleanSession 设置为缺省值 true 的情况下进行连接，那么会废弃暂挂的传递令牌。

预订

队列管理器或 IBM MessageSight 负责创建要发送给 MQTT 订户的发布内容。 队列管理器可检查 MQTT 客户机在预订中创建的主题过滤器是否与发布中的主题字符串匹配。 匹配可以是精确匹配，也可以包括通配符。 在由队列管理器将发布转发至订户之前，队列管理器将检查与此发布相关联的主题属性。 它按照使用包含了通配符的主题字符串来进行预订中所描述的搜索过程来确定管理主题对象是否授权用户进行预订。

当 MQTT 客户机接收到具有“至少一次”服务质量的发布时，它将调用 MqttCallback.messageArrived 方法来处理此发布。 如果发布的服务质量是“刚好一次”，QoS=2，那么 MQTT 客户机将调用 MqttClientPersistence 接口以存储接收到的消息。 然后，它将调用 MqttCallback.messageArrived。

MQTT 客户机提供的服务质量

MQTT 客户机提供了三种服务质量，用于将出版物交付到 WebSphere MQ 和 MQTT 客户机: "最多一次" , "至少一次" 和 "正好一次"。 当 MQTT 客户机将请求发送至 IBM WebSphere MQ 以创建预订时，将使用“至少一次”服务质量来发送请求。

发布的服务质量是 MqttMessage 的一种属性。 它是由 MqttMessage.setQos 方法设置的。

MqttClient.subscribe 方法可以降低应用于向客户机发送的、针对某个主题的发布的服务质量。 转发至订户的发布的服务质量可能不同于该发布的服务质量。 转发发布时，将使用这两个值当中的较小者。

至多一次
QoS=0

至多传递一次消息，或者根本就不传递消息。 不会确认通过网络来传递此消息。
不存储此消息。 如果客户机断开连接或者服务器失败，那么消息可能会丢失。
QoS=0 是最快传递方式。 有时候将它称为“发出消息之后无需等待应答”。
MQTT 协议不要求服务器将 QoS=0 时的发布转发至客户机。 如果在服务器接收发布时客户机已断开连接，那么根据服务器，可能会废弃此发布。 遥测 (MQXR) 服务不会废弃使用 QoS=0 发送的消息。 它们存储为非持久消息，且只有当队列管理器停止运行时才被废弃。

至少一次
QoS=1

QoS=1 是缺省传递方式。
始终会至少传递一次消息。 如果发送方未接收到确认信息，那么会在设置了 DUP 标志的情况下再次发送此消息，直到接收到确认信息为止。 因此，接收方可以多次发送同一消息，并且可以多次处理此消息。
必须将消息存储在发送方和接收方本地，直到已处理此消息为止。
在接收方已处理消息之后，就会从接收方删除此消息。 如果接收方是一个代理，那么会将消息发布至它的订户。 如果接收方是客户机，那么会将消息传递至订户应用程序。 删除消息之后，接收方会向发送方发送确认信息。
在发送方接收到来自接收方的确认信息之后，就会从发送方删除此消息。

刚好一次
QoS=2

始终刚好传递一次消息。
必须将消息存储在发送方和接收方本地，直到已处理此消息为止。
QoS=2 是最安全、但是最慢的传递方式。 从发送方删除消息之前，此方式在发送方与接收方之间至少采用两对传输。 在第一次传输之后，可以在接收方处理消息。
在第一对传输中，发送方将传输消息，并从它已将消息存储于的接收方获取确认信息。 如果发送方未接收到确认信息，那么会在设置了 DUP 标志的情况下再次发送此消息，直到接收到确认信息为止。
在第二对传输中，发送方将告知接收方 - 它可以完成处理 "PUBREL" 消息。 如果发送方未接收到 "PUBREL" 消息的确认信息，那么会再次发送 "PUBREL" 消息，直到接收到确认信息为止。 当发送方接收到 "PUBREL" 消息的确认信息时，它就会删除它保存的消息。
如果接收方不重新处理消息，那么接收方可以在第一阶段或者第二阶段处理此消息。 如果接收方是一个代理，那么它会将消息发布至订户。 如果接收方是客户机，那么它会将消息传递至订户应用程序。 接收方会将一条完成消息发送回发送方，指出它已经完成了处理此消息。

保留的发布和 MQTT 客户机

如果您针对具有保留发布的主题创建预订，那么将立即转发给您有关该主题的最新保留发布。

使用 MqttMessage.setRetained 方法来指定是否保留了有关某个主题的发布。

要在 IBM WebSphere MQ 中删除保留的发布，请运行 CLEAR TOPICSTR**CLEAR TOPICSTR** MQSC 命令。

如果创建了有效内容为空的发布，那么会将这个空白发布转发给订户。 其他 MQTT 代理程序可能不会将空白发布转发给订户。

如果您针对具有保留发布的主题发布非保留发布，那么保留的发布不会受到影响。 当前订户将接收到新的发布。 新订户将首先接收到保留的发布，然后接收所有新的发布。

创建或更新保留发布时，请使用 QoS， 1 或 2 发送该发布。 如果发送 QoS 为 0 的发布，那么 IBM WebSphere MQ 将创建非持久性保留发布。 如果队列管理器停止，就不会保留此发布。

使用保留的发布来记录测量结果的最新值。 保留的主题的新订户将立即接收到测量结果的最新值。 如果自订户上一次预订发布主题后未进行任何新的测量，且如果订户重新预订，那么该订户将再次接收到有关该主题的最新的保留发布。

预订

使用主题过滤器创建一些预订以注册对发布主题的兴趣。 客户机可以创建多个预订，或者创建一个包含使用通配符的主题过滤器的预订，以注册对于多个主题的兴趣。 关于主题的、与过滤器相匹配的发布被发送至客户机。 当客户机断开连接时，预订可以保持活动状态。 当客户机重新连接时，就会将这些发布发送至客户机。

使用 MqttClient.subscribe 方法并传递一个或多个主题过滤器和服务质量参数来创建预订。 服务质量参数设置订户准备用于接收消息的最高服务质量。 无法使用更高的服务质量来传递发送至此客户机的消息。 服务质量被设置为发布消息时的原始值与为预订指定的级别这两者之间的较小者。 接收消息的缺省服务质量为 QoS=1（至少一次）。

预订请求本身是使用 QoS=1 发送的。

当 MQTT 客户机调用 MqttCallback.messageArrived 方法时，订户会接收发布。 messageArrived 方法还会传递主题字符串，消息与此主题字符串一起被发布至订户。

可以使用 MqttClient.unsubscribe 方法来除去某个预订或者一组预订。

WebSphere MQ 命令可以除去预订。使用 WebSphere MQ Explorer 或使用 **runmqsc** 或 PCF 命令列出预订。 已命名全部 MQTT 客户机预订。 将为其提供以下格式的名称: *ClientIdentifier*:*Topic name*

如果在连接客户机之前使用缺省 MqttConnectOptions 或将 MqttConnectOptions.cleanSession 设置为 true，那么在客户机连接时将除去客户机的任何旧预订。 当客户机断开连接时，将移除客户机在会话期间创建的所有新预订。

如果在连接之前将 MqttConnectOptions.cleanSession 设置为 false，那么客户机创建的任何预订都将添加到客户机在连接之前已存在的所有预订。 当客户机断开连接时，所有预订仍保持活动状态。

要了解 cleanSession 属性影响预订的方式，另一种方法就是将它视作模态属性。 在其缺省方式 cleanSession=true 下，客户机仅在会话的作用域内创建预订和接收发布。 在另一种方式 cleanSession=false 下，预订是持久预订。 客户机可以连接和断开连接，而其预订保持活动状态。 当客户机重新连接时，它将接收任何未传递的发布。 在它连接之后，它可以自己修改处于活动状态的预订集。

必须先设置 cleanSession 方式，然后再进行连接; 该方式将持续整个会话。 要更改此属性的设置，必须将客户机断开连接，然后再重新连接客户机。 如果将方式从使用 cleanSession=false 更改为 cleanSession=true，那么将废弃客户机的所有先前预订以及尚未接收的任何发布。

一旦发布与活动预订相匹配的发布，就会将它们发送至客户机。 如果客户机已断开连接，那么如果它使用同一客户机标识来重新连接至同一服务器，并且将 MqttConnectOptions.cleanSession 设置为 false，就会将它们发送至客户机。

由客户机标识来标识特定客户机的预订。 可以将客户机从不同的客户机设备重新连接至同一服务器，并继续处理相同的预订和接收未传递的发布。

MQTT 客户机中的主题字符串和主题过滤器

主题字符串和主题过滤器用于发布和预订。 MQTT 客户机中的主题字符串和过滤器的语法大部分与 IBM WebSphere MQ 中的主题字符串相同。

主题字符串用来将发布发送至订户。 使用方法 MqttClient.getTopic(java.lang.String topicString)创建主题字符串。

主题过滤器用来预订主题和接收发布。 主题过滤器中可以包含通配符。 借助通配符，可以预订多个主题。 使用预订方法创建主题过滤器; 例如， `MqttClient.subscribe(java.lang.String topicFilter)`。

主题字符串

主题字符串中描述了 IBM WebSphere MQ 主题字符串的语法。 MQTT 主题字符串的语法在 Java 的 MQTT 客户机. 有关 MQTT 客户机库的客户机 API 文档的链接，请参阅 MQTT 客户机编程参考。的 API 文档中的 `MqttClient` 类中进行了描述。

每种类型的主题字符串的语法几乎完全相同。 它们之间有四项次要的差别：

1. MQTT 客户机发送到 IBM WebSphere MQ 的主题字符串必须遵循队列管理器名称的约定。 尤其是，非主题字符串不能包含连字符。
2. 最大长度不同。 IBM WebSphere MQ 主题字符串不能超过 10,240 个字符。 MQTT 客户机最多可创建 65535 字节的主题字符串。
3. MQTT 客户机创建的主题字符串不能包含空字符。
4. 在 WebSphere Message Broker 中，空的主题级别 `'...//...'` 无效。 然而，IBM WebSphere MQ 支持空的主题级。

与 IBM WebSphere MQ 发布/预订不同，`mqttv3` 协议没有“管理主题对象”概念。 不能根据主题对象和主题字符串来构造主题字符串。 但是，在 WebSphere MQ 中，主题字符串将映射至管理主题。 与管理主题相关联的访问控制会确定是废弃发布，还是将它发布至主题。 将发布转发至订户时，应用于此发布的属性会受到管理主题的属性的影响。

主题过滤器

基于主题的通配符方案中描述了 IBM WebSphere MQ 主题过滤器的语法。 可以使用 MQTT 客户机构造的主题过滤器的语法在 Java 的 MQTT 客户机的 API 文档中的 `MqttClient` 类中进行了描述。 有关 MQTT 客户机库的客户机 API 文档的链接，请参阅 MQTT 客户机编程参考。

每种类型的主题过滤器的语法几乎完全相同。 唯一的差别是不同的 MQTT 代理程序解释主题过滤器的方式有所不同。 在 WebSphere Message Broker V6 中，只能在主题过滤器末尾使用多点传送通配符。 然而，在 WebSphere MQ 中，可以在主题树的任何层使用多点传送通配符；例如，`USA/#/Dutchess County`。

MQTT 客户机编程参考

此处是指向 移动消息传递和 M2M 客户机包 和相关客户机 API 文档的链接。

在移动消息传递和 M2M 客户机包中，MQTT 客户机库与其生成的 API 文档捆绑在一起。 您可以从 IBM 消息传递社区下载来下载此客户机包。

您可以通过访问以下 Eclipse Paho 项目链接来查看最新 API 文档的联机副本：

- MQTT Java 客户机类
- MQTT C 客户机库
- 异步 MQTT C 客户机库

注:

1. 将 MQTT Java 应用程序链接到 `org.eclipse.paho.client.mqttv3` 软件包而不是 `com.ibm.micro.client.mqttv3`。 新程序包。 提供 `com.ibm.micro.client.mqttv3` 软件包是为了支持现有 MQTT Java 应用程序。
2. V7.5.0.1 将 C 的 MQTT 客户机应用程序链接到 `MQTTAsync` 库，而不是 `MQTTClient` 库。 提供 `MQTTClient` 是为了支持 C 的现有 MQTT 应用程序。
3. JavaScript 的 MQTT 消息传递客户机 需要支持 WebSockets 的 MQTT 服务器。 例如，IBM WebSphere MQ Version 7.5 和更高版本即满足这一要求。

MQTT 服务器入门

IBM 及其他供应商提供可支持 MQTT 传输协议的消息传递服务器。 最基本的 MQTT 服务器启用由 MQTT 客户机库支持的移动应用程序和设备，以便交换消息。 IBM WebSphere MQ 和 IBM MessageSight 是来自 IBM 的 MQTT 服务器。 除了充当基本 MQTT 服务器，它们还在 MQTT 客户机应用程序与企业应用程序之间交换消息。 IBM 提供的所有 MQTT 服务器都支持 MQTT version 3.1 协议以及 WebSocket protocol 上的 MQTT。

IBM 提供的当前 MQTT 服务器

IBM WebSphere MQ

- IBM WebSphere MQ 提供企业级消息传递。 启用遥测组件的 IBM WebSphere MQ 还可充当 MQTT 服务器。
- 这支持移动、机器到机器 (M2M) 和基于设备的应用程序，并且还允许它们与诸如 IBM WebSphere MQ 和 JMS 应用程序的企业消息传递应用程序交换消息。
- IBM WebSphere MQ 安装包含来自 IBM 的 MQTT SDK 的副本。 此 SDK 提供样本 MQTT 客户机应用程序以及支持这些应用程序的 MQTT 客户机库。

  注: 要获取此 SDK 的最新版本，请下载移动消息传递和 M2M 客户机包。 有关更多信息，请参阅 第 9 页的『MQTT 客户机入门』。
- MQTT 支持最先包含在 IBM WebSphere MQ Version 7.0.1 中。 有关 IBM WebSphere MQ 的每个发行版的完整信息，请参阅以下产品文档:
  - WebSphere MQ Telemetry V7.5
  - WebSphere MQ Telemetry V7.1

  有关 IBM WebSphere MQ 简介以及 IBM WebSphere MQ Telemetry 组件入门步骤，请参阅 第 123 页的『IBM WebSphere MQ 作为 MQTT 服务器』。

IBM MessageSight

- IBM MessageSight 是基于设备的 MQTT 服务器，它可以同时连接大量 MQTT 客户机，并且可以提供所需的性能和可扩展性来适应不断增多的移动设备和传感器。 它支持 MQTT version 3.1 协议以及 WebSocket protocol 上的 MQTT

。
- 作为 MQTT 服务器的 IBM MessageSight 的主要特性和益处如下:
  - 高性能、可靠性以及可扩展的消息传递。
  - 通过支持大规模社区的并发连接端点，专门设计为用于机器到机器 (M2M) 和事项互联网场景。
  - 易于安装和使用。 它可以在 30 分钟内启动并运行。
  - 支持包括 Android 和 iOS 的本机移动应用程序。
  - 作为发布/预订代理程序与 IBM WebSphere MQ 集成。

- 有关 IBM MessageSight 快速简介，请参阅 YouTube 上的 MessageSight 简介以及 MessageSight 声明。有关详细技术信息，请参阅 MessageSight 产品文档。

IBM WebSphere MQ Telemetry daemon for devices

- 这也称为 IBM WebSphere MQ Telemetry advanced client for C。 这是占用空间小的 MQTT 服务器，它一般在卫星位置或朝向网络边缘的设备中运行；例如，在机顶盒、远程遥测单元或者销售点终端内。
- 它的典型用途是集中大量 MQTT 客户机连接，然后在单个 MQTT 连接中通过因特网连接到 IBM WebSphere MQ 。 例如，您可在建筑物内安装大量传感器，将它们连接到 IBM WebSphere MQ Telemetry daemon for devices，然后再将此守护程序连接到 IBM WebSphere MQ。
- IBM WebSphere MQ Telemetry daemon for devices 包含在 IBM WebSphere MQ 中。 需要一个单独的许可证才能将其连接到 IBM WebSphere MQ。 请参阅 IBM 美国软件发布 212-091。

Really Small Message Broker

- Really Small Message Broker (RSMB) 是 IBM WebSphere MQ Telemetry daemon for devices 的一种版本。 主要差别是在用途上。 RSMB 是 IBM alphaWorks 提供的小型测试服务器，旨在评估或试验基于 MQTT 的解决方案时使用。 RSMB 在若干 Linux 平台、Windows XP、Apple Mac OS X Leopard 和 Unslung (Linksys NSLU12) 上支持 MQTT。

IBM 提供的先前 MQTT 服务器

WebSphere Message Broker *(现在称为 IBM Integration Bus)*

- WebSphere Message Broker V6 提供了其自己的 MQTT 服务器。 该支持在 WebSphere Message Broker V7 中已替换为 IBM WebSphere MQ 的遥测组件。

其他 MQTT 服务器

MQTT.org 在其“软件”页面维护 MQTT 服务器和代理程序的列表，包括开放式源代码服务器。

相关任务

第 9 页的『MQTT 客户机入门』
您可以通过构建和运行使用 MQTT 客户机库的样本 MQTT 客户机应用程序来开始开发移动或 机器到机器 (M2M) 应用程序。 样本应用程序和关联的客户机库在 移动消息传递和 M2M 客户机包 中可从 IBM 获取。 存在使用 Java， JavaScript 和 C 编写的应用程序和客户机库版本。 您可以在大多数平台和设备 (包括 Apple 中的 Android 设备和产品) 上运行这些应用程序。

IBM WebSphere MQ 作为 MQTT 服务器

有关使用 IBM WebSphere MQ 中包含的 MQTT 服务器的简介。

要入门，请执行以下文章中的步骤：

- 第 124 页的『安装 IBM WebSphere MQ』
- 第 125 页的『从命令行配置 MQTT 服务』
- 第 127 页的『通过 IBM WebSphere MQ Explorer 配置 MQTT 服务』

注: 您可以通过使用命令行界面示例来快速入门。 但是，如果您的配置与示例显著不同，那么您需要更多知识和技能才能有效地使用命令行界面。 使用 IBM WebSphere MQ Explorer 界面可轻松地入门并执行标准配置任务。

有关 IBM WebSphere MQ Telemetry 组件的关键概念信息，请参阅 IBM WebSphere MQ 产品文档内的以下文章：

- 将遥测设备连接至队列管理器
- 遥测 (MQXR) 服务
- 遥测通道

相关信息

在 Linux 和 AIX 上配置队列管理器以进行遥测

在 Windows 上为 Telemetry 配置队列管理器
配置分布式队列以将消息发送至 MQTT 客户机
管理 WebSphere MQ Telemetry

安装 IBM WebSphere MQ

遵循以下指示信息以获取和安装 IBM WebSphere MQ，并在 Windows 或 Linux 上配置 IBM WebSphere MQ Telemetry。

开始之前

对于在 IBM WebSphere MQ 上运行的 MQTT 服务所支持的操作系统，请参阅 IBM WebSphere MQ Telemetry 系统需求。

通过以下方法之一获取 IBM WebSphere MQ 安装材料的副本和一个许可证：

1. 咨询您的 IBM WebSphere MQ 管理员以获取安装材料，并确认您可以接受许可协议。
2. 获取 IBM WebSphere MQ 的 90 天评估副本。 请参阅 评估: IBM WebSphere MQ。
3. 购买 IBM WebSphere MQ。 请参阅 IBM WebSphere MQ 产品页面。

关于此任务

以 `root` 身份 (在 Linux 上) 和管理员身份 (在 Windows 上) 安装 IBM WebSphere MQ。 安装时，选择额外的选项**遥测服务**和**遥测客户机**，安装 IBM WebSphere MQ Telemetry 组件。 创建用户标识以管理 IBM WebSphere MQ，然后检查是否定义了访客用户标识。 访客用户标识在样本 MQTT 服务配置中用于授权 MQTT 访问 IBM WebSphere MQ。

安装 IBM WebSphere MQ 后，通过执行 第 125 页的『从命令行配置 MQTT 服务』 或 第 127 页的『通过 IBM WebSphere MQ Explorer 配置 MQTT 服务』 中的步骤来启动 MQTT 服务。

过程

1. 以 `root` 身份 (在 Linux 上) 或以管理员身份 (在 Windows 上) 登录。
2. 安装 IBM WebSphere MQ。

   遵循 在 Linux 上安装 WebSphere MQ 服务器或 在 Windows 上安装 WebSphere MQ 服务器中的指示信息。 选择**遥测服务**和**遥测客户机**，安装 IBM WebSphere MQ Telemetry 组件。

   在 Linux 上，记下 “后续操作” 部分中的指示信息，以使安装成为主安装。 即使该安装是工作站上的唯一 IBM WebSphere MQ 安装，也请使其成为主安装。 请参阅单个安装 WebSphere MQ V7.1 或更高版本，配置为主安装。

   要完全遵照样本配置指示信息，必须将安装确定为主安装。

   多个安装: 如果您希望使用非主安装，请运行 setmqenv 命令。 它会在工作站上的命令窗口中设置 IBM WebSphere MQ 环境。 请参阅多个安装。

   假定您接受了安装程序提供的缺省安装位置，那么 IBM WebSphere MQ 将安装在以下目录：

   Linux 64 位

   ```
   /opt/mqm
   ```

   Windows 32 位

   ```
   C:\Program Files\IBM\WebSphere MQ
   ```

   Windows 64 位

   ```
   C:\Program Files (x86)\IBM\WebSphere MQ
   ```

   安装目录显示为 *MQ_INSTALLATION_PATH*

3. 可选： 将您将管理其 IBM WebSphere MQ 的用户添加到工作站上的 mqm 组。

   此步骤在 Windows 上是可选的，因为您可以作为 Windows 管理员来管理 IBM WebSphere MQ 。 请参阅在 UNIX 和 Windows 系统上管理 WebSphere MQ 的权限。

   如果您的 Windows 工作站是某个域的成员，请参阅使用非缺省安全许可权的 Windows 2000 域或使用缺省安全许可权的 Windows 2003 和 Windows Server 2008 域。

   在 Linux 上，安装程序将创建用户 mqm 作为组 mqm 的成员。 为此用户提供密码，或创建将 mqm 作为主组的另一个用户。
4. 可选： 使用您指定为 mqm 组成员的用户登录。

   此步骤在 Windows 上是可选的，因为您可以作为 Windows 管理员来管理 IBM WebSphere MQ 。
5. 检查在工作站上是否定义了访客用户标识。

   访客用户标识为 “guest” (在 Windows 上) 和 “nobody” (在 Linux 上)。 访客用户标识不需要任何操作系统许可权或权利。

结果

您在工作站以主 IBM WebSphere MQ 安装完成了 IBM WebSphere MQ 安装并创建了组 mqm。 安装会将管理 IBM WebSphere MQ 的许可权授予 mqm 组的成员。 Windows 上的管理员组成员还具有管理 IBM WebSphere MQ 的权限。

下一步做什么

1. 从命令行或 IBM WebSphere MQ Explorer 配置 MQTT 服务; 请参阅 第 127 页的『通过 IBM WebSphere MQ Explorer 配置 MQTT 服务』 或 第 125 页的『从命令行配置 MQTT 服务』。
2. 测试 Android， iOS， WebSockets， Java 和 “C” MQTT 客户机。
3. 完成测试后，通过在 Windows 上运行命令 *MQ_INSTALLATION_PATH*\mqxr\samples\CleanupMQM.bat ，在 Linux 上运行命令 *MQ_INSTALLATION_PATH*/mqxr/samples/CleanupMQM.sh 来除去队列管理器和 MQTT 服务。

相关信息

安装 WebSphere MQ Telemetry
在 Linux 上安装 WebSphere MQ 服务器
在 Windows 系统上安装 WebSphere MQ 服务器

从命令行配置 MQTT 服务

按照下列指示信息使用命令行将 IBM WebSphere MQ 配置为运行样本 IBM WebSphere MQ Telemetry 应用程序。 这些步骤说明如何运行脚本以在名为 MQXR_SAMPLE_QM 的新队列管理器上创建 MQTT 服务。

开始之前

您必须具有对 IBM WebSphere MQ 队列管理器的管理访问权才能够设置 MQTT 服务。 您可以通过多种方式获得队列管理器的访问权：

1. 获得 IBM WebSphere MQ 的副本并在您自己的 Linux 或 Windows 工作站上的创建队列管理器。 遵照第 124 页的『安装 IBM WebSphere MQ』 中的指示信息以获取和安装 IBM WebSphere MQ。 请注意，安装时您还需要选择**遥测服务**和**遥测客户机**。 您也可以修改现有安装以添加这些选项。
2. 联系 IBM WebSphere MQ 管理员，获取对于将 IBM WebSphere MQ Telemetry 作为选件安装的服务器上的队列管理器的管理访问权。 V 7.5.0.1 除队列管理器名称外，针对 MQTT 和基于 WebSockets 的 MQTT 至少需要两个 TCP/IP 端口。 如果计划连接到安全客户机，那么另外还需要至少两个端口。

要完全按照对任务中步骤的描述来执行这些步骤，您必须能够创建称为 MQXR_SAMPLE_QM 的队列管理器，并且 TCP/IP 端口 1883 必须处于未使用状态。

关于此任务

在此任务中，您运行用于创建队列管理器的脚本，然后配置 MQTT 服务以侦听端口 1883 上的 MQTT V3.1 客户机连接。此配置授予每个人发布和预订任何主题的许可权。 安全和访问控制的配置是最低限度的，只用于访问权受限的安全网络上的队列管理器。 要在不安全的环境中运行 IBM WebSphere MQ 和 MQTT，必须配置安全性。 要配置 IBM WebSphere MQ 和 MQTT 的安全性，请参阅本任务末尾的相关链接。

过程

1. 使用对 IBM WebSphere MQ 具有管理权限的用户标识登录。

   要定义对 IBM WebSphere MQ 具有管理权限的用户标识，请参阅 第 124 页的『安装 IBM WebSphere MQ』 中的步骤 3。
2. 打开命令窗口并运行样本命令脚本，以创建并启动称为 MQXR_SAMPLE_QM 的样本队列管理器和 MQTT 服务。

   样本脚本的路径为 %MQ_FILE_PATH%\mqxr\samples\SampleMQM.bat on Windows 和 *MQ_INSTALLATION_PATH*/mqxr/samples/SampleMQM.sh on Linux。

   输入以下命令，以创建和配置队列管理器：

   - 


     ```
     "%MQ_FILE_PATH%\mqxr\samples\SampleMQM.bat"
     ```
   - 


     ```
     MQ_INSTALLATION_PATH/mqxr/samples/SampleMQM.sh
     ```

结果

此样本使用 Windows 上的以下属性创建名为 PlainText 的 MQTT 通道：

```
com.ibm.mq.MQXR.channel/PlainText: \
com.ibm.mq.MQXR.Protocol=MQTT;\
com.ibm.mq.MQXR.Port=1883;\
com.ibm.mq.MQXR.Backlog=4096;\
com.ibm.mq.MQXR.UserName=Guest;\
com.ibm.mq.MQXR.StartWithMQXRService=true
```

除 com.ibm.mq.MQXR.UserName=nobody 外，Linux 上的通道属性与 Windows 上的相同。

MQTT V3.1 使用变量 *com.ibm.mq.MQXR.UserName* 中设置的用户标识连接到端口 1883 访问 IBM WebSphere MQ 的客户机。 样本脚本通过以下 IBM WebSphere MQ 命令为用户标识授权：

```
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
```

第一个命令为用户提供在从基本主题继承其许可权的主题上发布和预订的权限。 第二个命令为用户提供将消息放在 SYSTEM.MQTT.TRANSMIT.QUEUE 传输队列上的权限。 MQTT 服务将 SYSTEM.MQTT.TRANSMIT.QUEUE 上的消息作为出版物发送给 MQTT 订户。

脚本启动队列管理上的 MQTT 服务来侦听端口 1883 上的连接。

下一步做什么

按照以下步骤通过运行样本 MQTT V3.1 Java 应用程序来测试连接。

样本 Java 应用程序的源位于 MQTTV3Sample.java 文件中。

运行此样本需要两个命令窗口。 在一个窗口中以订户身份运行样本，在另一个窗口中以发布者身份运行样本。

- Windows 要启动订户，请运行以下命令

```
"%MQ_FILE_PATH%\mqxr\samples\RunMQTTV3Sample.bat" -a subscriber
```

要发布，请运行命令：

```
"%MQ_FILE_PATH%\mqxr\samples\RunMQTTV3Sample.bat"
```

- Linux 要启动订户，请运行以下命令

```
MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh -a subscriber
```

要发布，请运行命令：

```
MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh
```

发布者和订户将输出写入到其命令窗口：

```
Connected to tcp://localhost:1883
Publishing to topic "MQTTV3Sample/Java/v3" qos 2
Disconnected
Press any key to continue . . .
```

图 *27:* 来自发布者的输出

```
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
```

图 *28:* 来自订户的输出

服务器现在已准备就绪，可供您测试 MQTT V3.1 应用程序。

相关任务

使用 WebSphere MQ Explorer 配置 MQTT 服务

遵循以下指示信息以配置 IBM WebSphere MQ，使用 IBM WebSphere MQ Explorer 来运行样本 IBM WebSphere MQ Telemetry 客户机。 这些步骤为您展示如何通过运行 `Define sample` 配置向导来创建 MQTT 服务。

相关信息

WebSphere MQ Telemetry

为 WebSphere MQ Telemetry 开发应用程序

管理 WebSphere MQ Telemetry

WebSphere MQ Telemetry 安全性

通过 IBM WebSphere MQ Explorer 配置 MQTT 服务

遵循以下指示信息以配置 IBM WebSphere MQ，使用 IBM WebSphere MQ Explorer 来运行样本 IBM WebSphere MQ Telemetry 客户机。 这些步骤为您展示如何通过运行 `Define sample` 配置向导来创建 MQTT 服务。

开始之前

您必须具有对 IBM WebSphere MQ 队列管理器的管理访问权才能够设置 MQTT 服务。 您可以通过多种方式获得队列管理器的访问权：

1. 获得 IBM WebSphere MQ 的副本并在您自己的 Linux 或 Windows 工作站上的创建队列管理器。 遵照第 124 页的『安装 IBM WebSphere MQ』 中的指示信息以获取和安装 IBM WebSphere MQ。 请注意，安装时您还需要选择**遥测服务**和**遥测客户机**。 您也可以修改现有安装以添加这些选项。

2. 联系 IBM WebSphere MQ 管理员，获取对于将 IBM WebSphere MQ Telemetry 作为选件安装的服务器上的队列管理器的管理访问权。 V 7.5.0.1 除队列管理器名称外，针对 MQTT 和基于 WebSockets 的 MQTT 至少需要两个 TCP/IP 端口。 如果计划连接到安全客户机，那么另外还需要至少两个端口。

要完全按照对任务中步骤的描述来执行这些步骤，您必须能够创建称为 MQXR_SAMPLE_QM 的队列管理器，并且 TCP/IP 端口 1883 必须处于未使用状态。

关于此任务

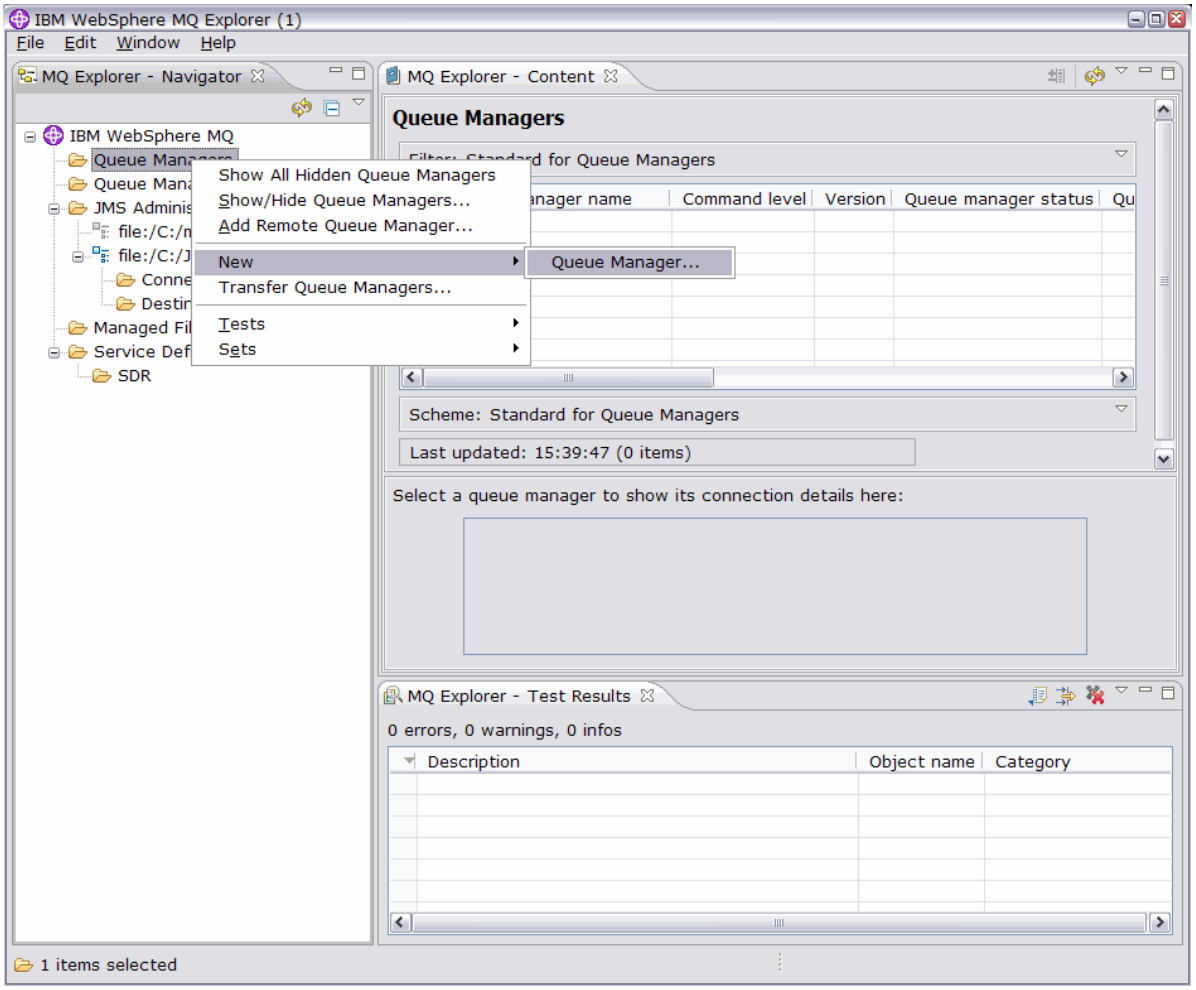
在此任务中，您将运行 IBM WebSphere MQ Explorer `Define sample` 配置向导来创建 MQTT 服务，以在端口 1883 上侦听 MQTT V3.1 客户机连接。此配置授予每个人发布和预订任何主题的许可权。 安全和访问控制的配置是最低限度的，只用于访问权受限的安全网络上的队列管理器。 要在不安全的环境中运行 IBM WebSphere MQ 和 MQTT，必须配置安全性。 要配置 IBM WebSphere MQ 和 MQTT 的安全性，请参阅本任务末尾的相关链接。

过程

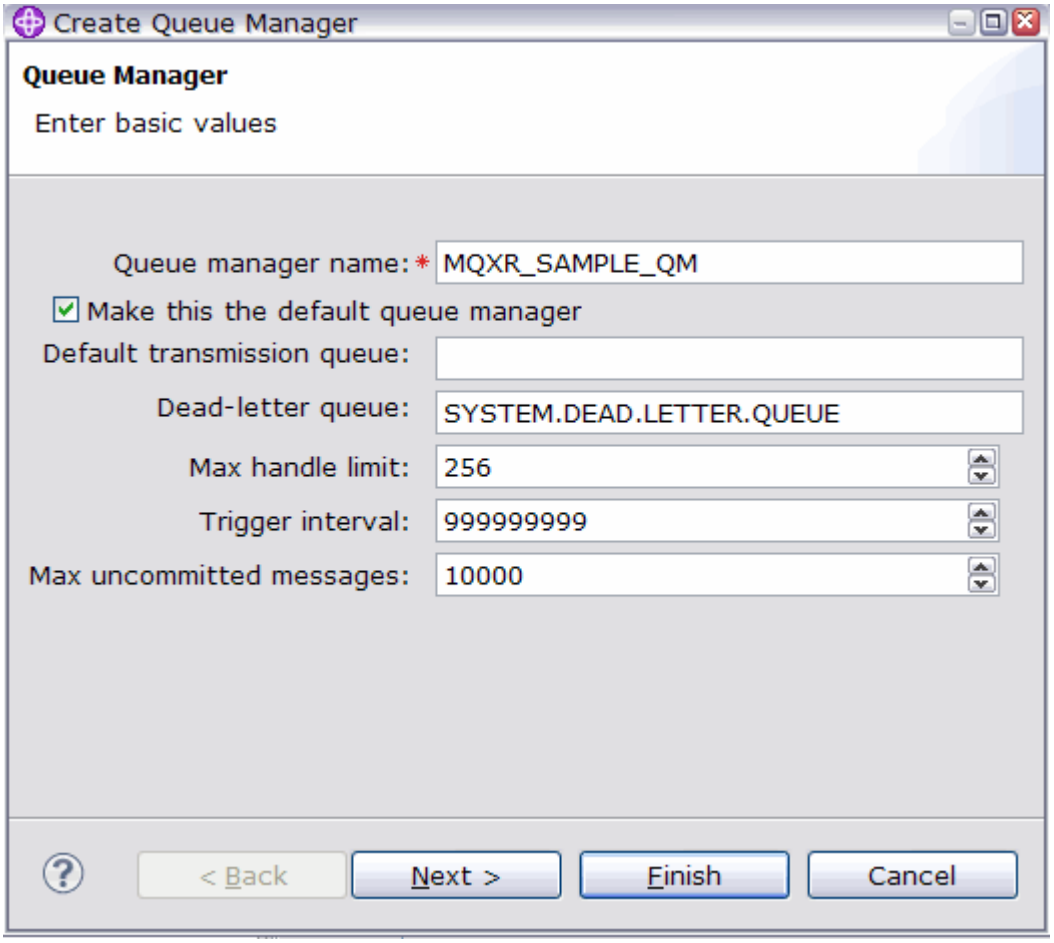
1. 使用对 IBM WebSphere MQ 具有管理权限的用户标识登录。

   要定义对 IBM WebSphere MQ 具有管理权限的用户标识，请参阅 第 124 页的『安装 IBM WebSphere MQ』 中的步骤 3。
2. 打开命令窗口，然后运行 IBM WebSphere MQ Explorer 命令 **strmqcfg** 以启动 IBM WebSphere MQ Explorer。
3. 创建队列管理器

   a) 启动**新建队列管理器**向导

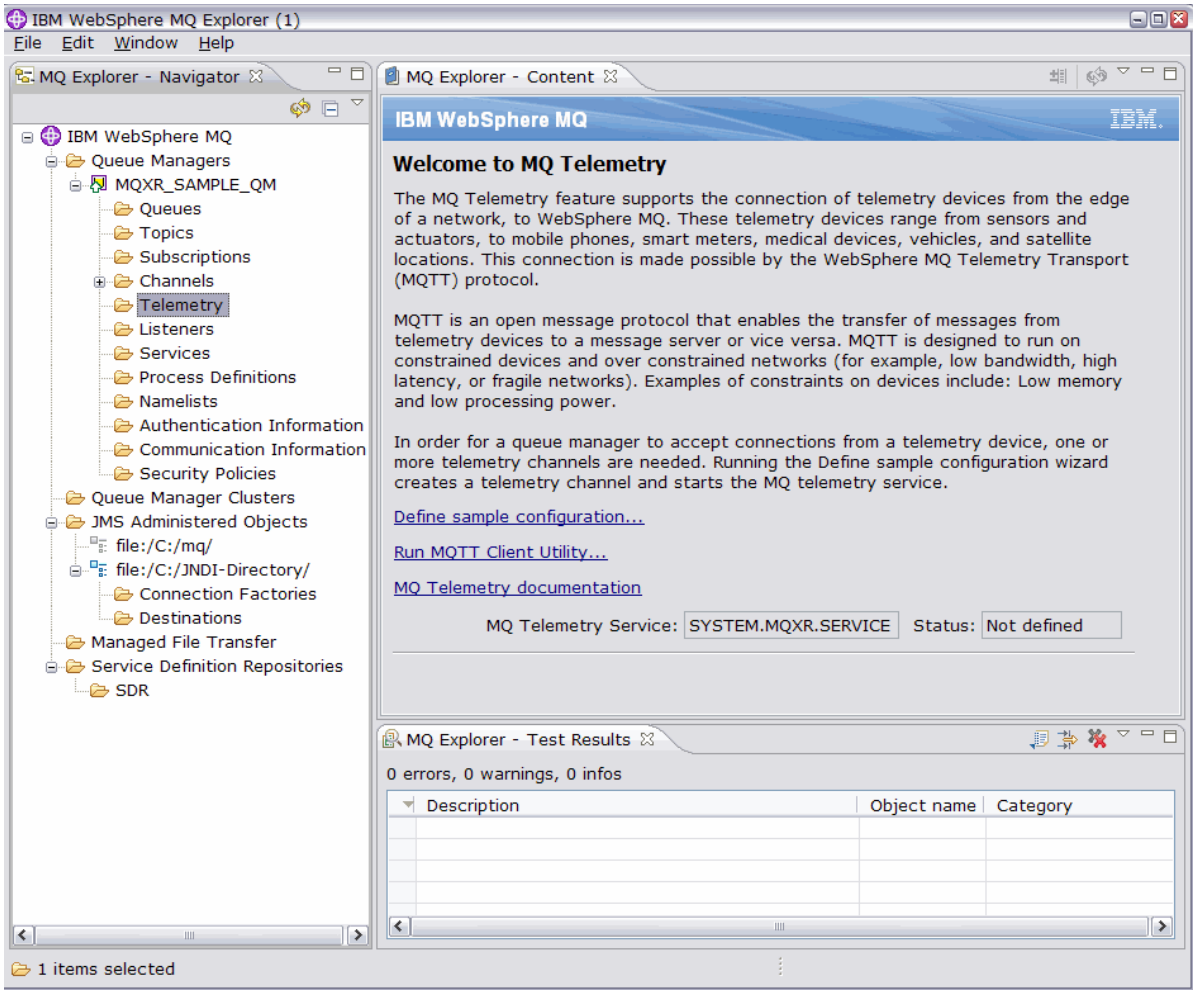


   b) 输入**队列管理器名称**和**死信队列**的名称。 为了方便起见，请将其作为缺省队列管理器。 单击**完成**。



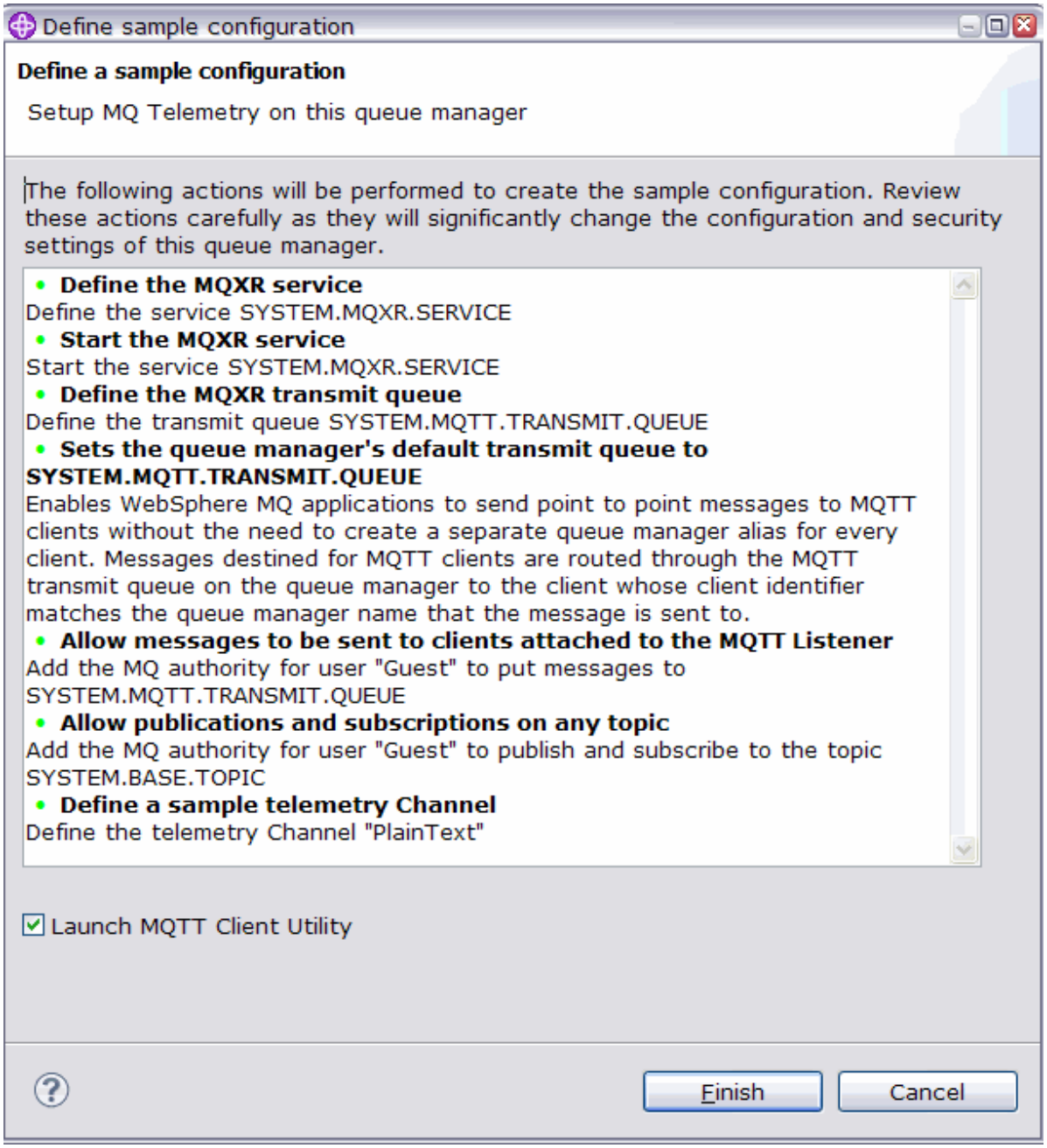
IBM WebSphere MQ Explorer 将创建并启动队列管理器。

4. 运行 Telemetry 的**定义样本配置**向导。
   a) 打开队列管理器的“遥测”文件夹。



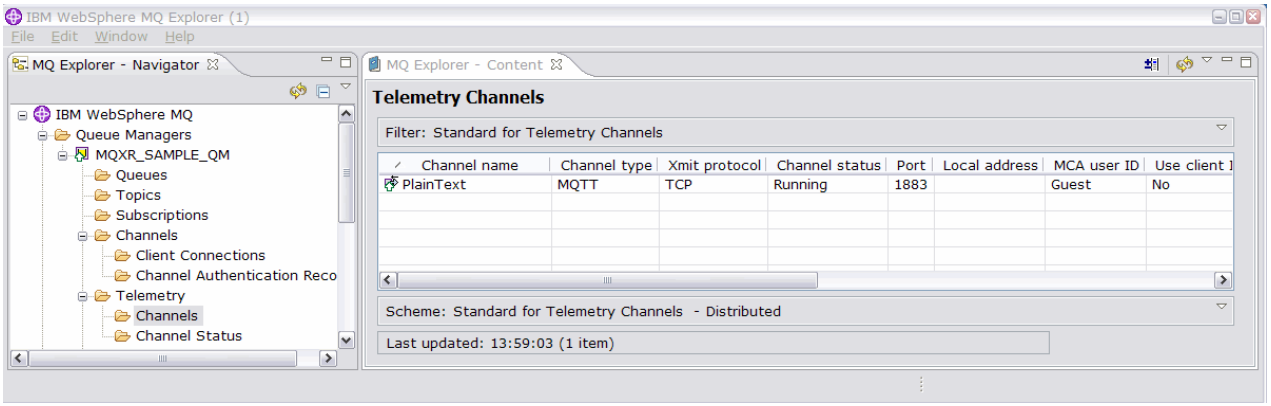
b) 单击**定义样本配置**启动向导。

c) 单击**完成**以创建遥测服务，并运行 MQTT 客户机实用程序。



结果

打开“遥测通道”文件夹，以列出样本通道。

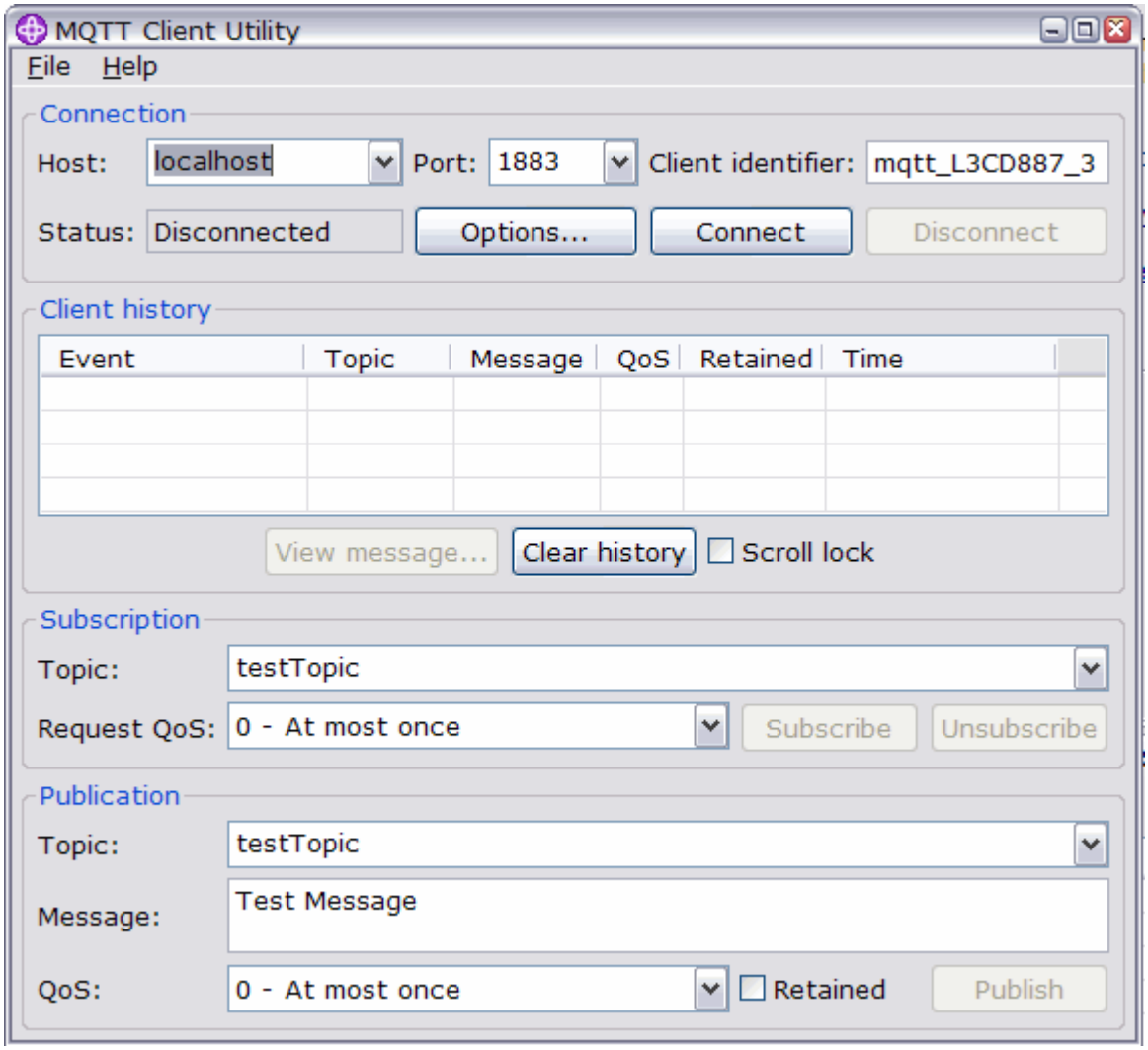


您可以修改此通道的属性，并可以在此窗口中添加和删除通道。

下一步做什么

通过运行 MQTT 客户机实用程序来测试连接。

1. 要启动客户机实用程序，请打开**遥测**文件夹并将单击**运行 MQTT 客户机实用程序**两次。
   这将打开两个**“MQTT 客户机实用程序”**窗口，内容相同但具有不同的客户机标识。



2. 单击两个窗口中的**连接**。
3. 单击两个窗口中的**预订**。
4. 单击任一窗口中的**发布**。 结果如第 133 页的图 29 中所示

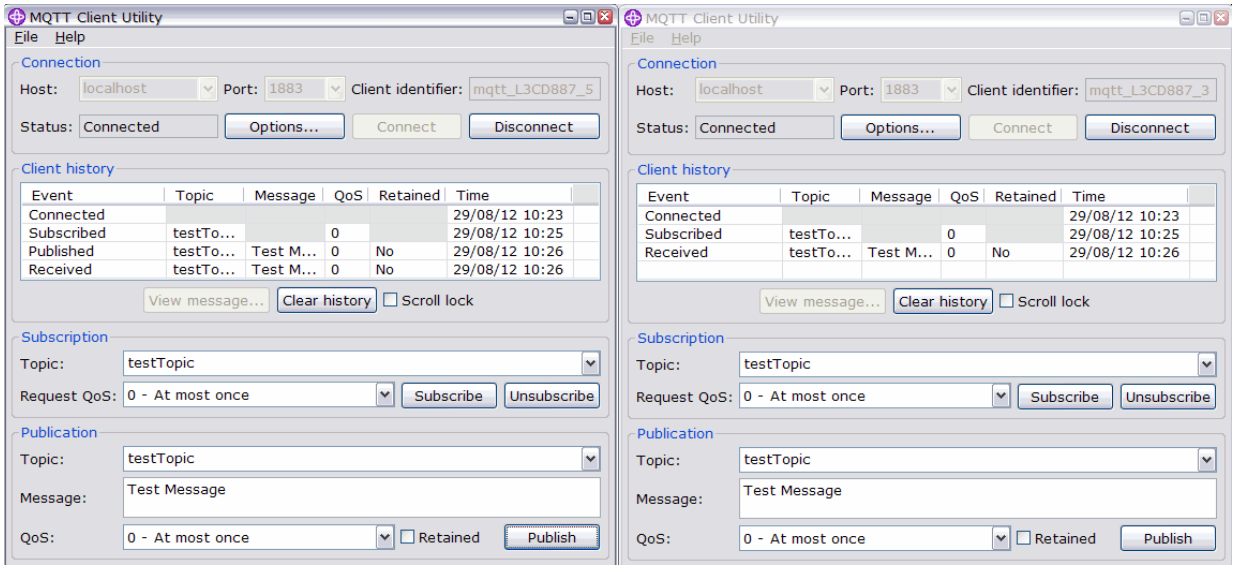


图 *29:* 结果

5. 单击两个窗口中的**断开连接**。

服务器现在已准备就绪，可供您测试 MQTT V3.1 应用程序。

相关任务

使用命令行配置 MQTT 服务

按照下列指示信息使用命令行将 IBM WebSphere MQ 配置为运行样本 IBM WebSphere MQ Telemetry 应用程序。 这些步骤说明如何运行脚本以在名为 MQXR_SAMPLE_QM 的新队列管理器上创建 MQTT 服务。

管理 WebSphere MQ Telemetry

相关信息

WebSphere MQ Telemetry

通过 WebSphere MQ Explorer 管理 WebSphere MQ Telemetry

为 WebSphere MQ Telemetry 开发应用程序

安全性

WebSphere MQ Telemetry 安全性

设备的 IBM WebSphere MQ Telemetry 守护程序概念

设备的 IBM WebSphere MQ Telemetry 守护程序是一个高级 MQTT V3 客户机应用程序。 使用它来存储和转发来自其他 MQTT 客户机的消息。 它像 MQTT 客户机一样连接至 IBM WebSphere MQ，但是您还可以将它连接至其他 MQTT 客户机。

守护程序是发布/预订代理程序。 MQTT V3 客户机与其连接，以使用要发布的主题字符串和要预订的主题过滤器来发布和预订主题。 主题字符串采用分层形式，通过由 / 隔开的主题级别实现。 主题过滤器是主题字符串，可包含单一级别 + 通配符和多级别 # 通配符作为主题字符串的最后部分。

注: 守护程序中的通配符遵循 WebSphere Message Broker v6 限制性更强的规则。 IBM WebSphere MQ 与之不同。 它支持多个多级别通配符，通配符可代替主题字符串中任意位置、任意数量的层次结构级别。

多个 MQTT v3 客户机使用侦听器端口连接到守护程序。 缺省侦听器端口可修改。 您可以定义多个侦听器端口并向其分配不同的名称空间，请参阅第 139 页的『设备侦听器端口的 WebSphere MQ Telemetry 守护程序』。 守护程序本身就是 MQTT v3 客户机。 配置守护程序网桥连接，将该守护程序连接到另一守护程序的侦听器端口或 WebSphere MQ Telemetry (MQXR) 服务。

您可以为设备的 WebSphere MQ Telemetry 守护程序配置多个网桥。 使用网桥将可以交换发布的守护程序网连接起来。

每个网桥都能发布和预订其本地守护程序的主题。 此外，也能发布和预订所连接的其他守护程序、WebSphere MQ 发布/预订代理程序或其他任何 MQTT v3 代理程序的主题。 利用主题过滤器，您可以选择要从一个代理程序传播到另一个代理程序的发布。 您可以向任一方向传播发布。 您可以将发布从本地守护程

序传播到其连接的每个远程代理程序，或从所连接的任一代理程序传播到本地守护程序；请参阅第 134 页的『设备网桥的 IBM WebSphere MQ Telemetry 守护程序』。

设备网桥的 IBM WebSphere MQ Telemetry 守护程序

设备网桥的 IBM WebSphere MQ Telemetry 守护程序使用 MQTT v3 协议连接两个发布/预订代理程序。 该网桥可沿任一方向将发布从一个代理程序传播到另一代理程序。 设备网桥连接的一端是 WebSphere MQ Telemetry 守护程序，另一端可能是队列管理器或另一个守护程序。 队列管理器使用遥测通道连接到网桥连接。 守护程序使用守护程序侦听器连接到网桥连接。

设备的 IBM WebSphere MQ Telemetry 守护程序支持同时连接到一个或多个其他代理程序。 来自守护程序的连接称为网桥，通过守护程序配置文件中的连接条目进行定义。 使用 IBM WebSphere MQ 遥测通道建立到 IBM WebSphere MQ 的连接，如下图所示：

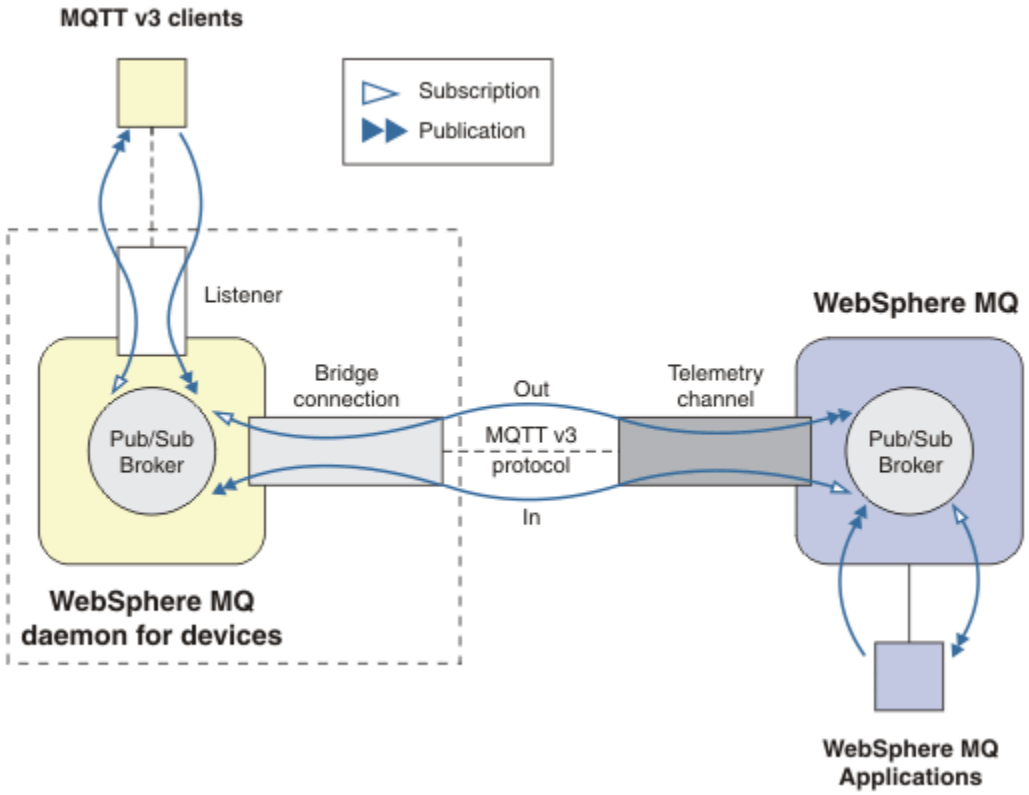


图 *30:* 将 *IBM WebSphere MQ Telemetry daemon for devices* 连接至 *IBM WebSphere MQ*

网桥将守护程序连接到另一个充当 MQTT v3 客户机的代理程序。 网桥参数将镜像 MQTT v3 客户机的属性。

网桥不仅仅是一个连接。 它还充当两个发布/预订代理程序之间的发布和预订代理。 本地代理程序是设备的 IBM WebSphere MQ Telemetry 守护程序，远程代理程序是支持 MQTT V3 协议的任何发布/预订代理程序。 通常，远程代理程序是另一个守护程序或 IBM WebSphere MQ。

网桥的工作是在两个代理程序之间传播发布内容。 网桥是双向的。 它采用任一方向传播发布内容。 第 134 页的图 30 说明了网桥将设备的 IBM WebSphere MQ Telemetry 守护程序连接至 IBM WebSphere MQ 的方式。 第 135 页的『网桥的主题设置示例』使用示例阐明如何使用主题参数配置网桥。

第 134 页的图 30 中的 In 和 Out 指示网桥的双向性。 在箭头的一端创建预订。 在箭头的另一端，将与预订匹配的发布内容发布到代理程序。 根据发布内容的流向对箭头进行标记。 发布内容将流入 (In) 守护程序，然后从守护程序流出 (Out)。 标签的重要性是在命令语法中使用。 请记住，In 和 Out 指发布内容的流向，而非预订发送方向。

可以将其他客户机、应用程序或代理程序连接到 IBM WebSphere MQ 或设备的 WebSphere MQ Telemetry 守护程序。 它们在所连接到的代理程序发布和预订主题。 如果代理程序是 IBM WebSphere MQ，那么可以集群和分发主题，而不在本地队列管理器显式定义。

使用网桥

使用网桥连接和侦听器将守护程序连接在一起。 使用网桥连接和遥测通道将守护程序和队列管理器连接在一起。 将多个代理程序连接在一起时，可以创建环路。 请注意：发布内容可能会围绕未检测到的代理程序环路无止境地循环。

使用桥接到 IBM WebSphere MQ 的守护程序的某些原因如下所示：

减少到 WebSphere MQ 的 MQTT 客户机连接数
通过使用守护程序层次结构，您可以将许多客户机连接到 WebSphere MQ；可以一次连接超过单个队列管理器可以连接数量的客户机。

在 MQTT 客户机与 WebSphere MQ 之间存储转发消息
您可以使用存储和转发操作，避免在客户机与 IBM WebSphere MQ 之间保持持续连接（如果客户机没有自己的存储器）。 您可以在 MQTT 客户机与 WebSphere MQ 之间使用多种类型的连接；请参阅用于监视与控制的遥测概念和方案。

对 MQTT 客户机与 WebSphere MQ 之间交换的发布内容进行过滤
通常，会将发布内容分为在本地处理的消息和涉及其他应用程序的消息。 本地发布内容可能包含传感器和传动结构之间的控制流，远程发布内容包含读数、状态和配置命令的请求。

更改发布内容的主题空间
避免来自与不同侦听器端口连接的客户机的主题字符串彼此冲突。 本示例使用守护程序对来自不同建筑物的计量表读数进行标记；请参阅分隔不同客户机组的主题空间。

网桥的主题设置示例

将所有内容发布至远程代理程序 - 使用缺省值

缺省方向称为 out，网桥将主题发布至远程代理程序。 topic 参数控制使用主题过滤器传播哪些主题。

网桥使用第 135 页的图 31 中的 topic 参数预订通过 MQTT 客户机或其他代理程序发布到本地守护程序的所有内容。 网桥将主题发布至通过网桥连接的远程代理程序。

```
connection Daemon1
topic #
```

图 *31:* 将所有内容发布至远程代理程序

将所有内容发布至远程代理程序 - 显式

以下代码段中的 topic 设置提供了与使用缺省值相同的结果。 唯一的差别是，**direction** 参数是显式的。 使用 `out` 方向预订本地代理程序、守护程序并发布到远程代理程序。 在远程代理程序发布网桥预订的本地守护程序上创建的发布内容。

```
connection Daemon1
topic # out
```

图 *32:* 将所有内容发布至远程代理程序 - 显式

将所有内容发布至本地代理程序

代替使用方向 `out`，您可以设置相反方向 `in`。 以下代码段配置网桥以预订该网桥连接的远程代理程序发布的所有内容。 网桥将主题发布至本地代理程序、守护程序。

```
connection Daemon1
topic # in
```

图 *33:* 将所有内容发布至本地代理程序

将所有内容从本地代理程序的导出主题发布至远程代理程序的导入主题

使用两个额外的主题参数 **local_prefix** 和 **remote_prefix** 来修改主题过滤器，前面几个示例中的 #。 一个参数用于修改要在预订中使用的主题过滤器，另一个参数用于修改要将发布内容发布至的主题。 效果是将一个代理程序中使用的主题字符串的开头替换为另一个代理程序上的其他主题字符串。

根据主题命令的方向，**local_prefix** 和 **remote_prefix** 的含义会相反。 如果方向是 out，缺省情况下，**local_prefix** 将用作主题预订的一部分，**remote_prefix** 将替换远程发布内容中主题字符串的 **local_prefix** 部分。 如果方向是 in，**remote_prefix** 将变成远程预订的一部分，**local_prefix** 将替换主题字符串的 **remote_prefix** 部分。

主题字符串的第一部分通常被认为是定义主题空间。 使用额外的参数来更改主题将发布至的主题空间。 您可以执行此操作以避免传播的主题与目标代理程序上的另一个主题产生冲突，或者用于除去安装点主题字符串。

例如，在以下代码段中，到守护程序的主题字符串 export/# 的所有发布内容将重新发布至远程代理程序的 import/#。

```
topic # out export/ import/
```

图 *34:* 将所有内容从本地代理程序的导出主题发布至远程代理程序的导入主题

将所有内容从远程代理程序的导出主题发布至本地代理程序的导入主题

以下代码段显示了反向配置；网桥预订通过远程代理程序的 export/# 主题字符串发布的所有内容，并将其发布至本地代理程序的 import/#。

```
connection Daemon1
topic # in import/ export/
```

图 *35:* 将所有内容从远程代理程序的导出主题发布至本地代理程序的导入主题

将所有内容从 1884/ 安装点发布至具有原始主题字符串的远程代理程序

在以下代码段中，网桥预订通过与本地守护程序的安装点 1884/ 连接的客户机发布的所有内容。 网桥将发布到该安装点的所有内容发布至远程代理程序。 将从发布至远程代理程序的主题中除去安装点字符串 1884/。 *local_prefix* 与安装点字符串 1884/ 相同，*remote_prefix* 是空字符串。

```
listener 1884
mount_point 1884/
connection Daemon1
topic # out 1884/ ""
```

图 *36:* 将所有内容从 *1884/* 安装点发布至具有原始主题字符串的远程代理程序。

分隔与不同守护程序连接的不同客户机的主题空间

假定为电能表编写应用程序以发布建筑物的计量表读数。 使用 MQTT 客户机将读数发布至托管在同一个建筑物中的守护程序。 为发布内容选择的主题是 power。 将同一个应用程序部署到复合体中的许多建筑物。 对于站点监视和数据存储，将使用网桥连接聚集来自所有建筑物的读数。 这些连接会将建筑物守护程序链接到中央位置的 WebSphere MQ。

所有建筑物都使用一个完全相同的客户机应用程序。 此应用程序发布到主题 power。 但是必须按大楼区分数据。 通过每一个大楼的守护程序进行此项操作，添加建筑物数量作为主题名称的前缀。 复合体中第一个建筑物中的网桥使用前缀 meters/building01/，第二个建筑物中的网桥使用前缀 meters/building02/。 其他建筑物中的读数遵循相同的模式。 因此， WebSphere MQ 接收具有 meters/building01/power 等主题的读数。

每个守护程序的配置文件具有遵循以下代码段中模式的主题语句：

```
connection Daemon1
topic power out "" meters/building01/
```

图 *37:* 分隔与不同守护程序连接的客户机的主题空间

在以前的代码片段中，空字符串是未使用的 local_prefix 参数的占位符。

注: 该示例有点不切实际，在此只作为一种例证。 实际上，应用程序发布至的主题空间很可能是可配置的。

分隔与相同守护程序连接的客户机的主题空间

假设单个守护程序用于连接所有电表。 假设在应用程序中可以配置为连接到不同的端口，您可以将不同建筑物中的电表连接到不同的侦听器端口来区分建筑物，如以下代码段所示。 本示例同样是人为的；它阐明了可以如何使用安装点。

```
listener 1884
mount_point meters/building01/
listener 1885
mount_point meters/building02/
connection Daemon1
topic meters/+/power out
```

图 *38:* 分隔与相同守护程序连接的客户机的主题空间

重新映射双向流动的发布内容的不同主题

在以下代码片段中的配置中，网桥在远程代理上预订单个主题 b ，并将有关 b 的发布转发到本地守护程序，将主题更改为 a。 网桥还会在本地代理上预订单个主题 x ，并将有关 x 的出版物转发到远程代理，从而将主题更改为 y。

```
connection Daemon1
topic "" in a b
topic "" out x y
```

图 *39:* 重新映射双向流动的发布内容的不同主题

关于本示例的重要一点是，在两个代理程序处预订和发布不同的主题。 两个代理程序的主题空间互不相交。

重新映射双向流动的发布内容的相同主题（循环）

与上一个示例不同，第 138 页的图 40 中的配置通常会产生一个环路。 在主题语句 topic "" in a b 中，网桥远程预订 b 并在本地发布至 a。 在另一个主题语句中，网桥在本地预订 a 并远程发布至 b。 可以如第 138 页的图 41 中所示编写同一配置。

一般结果是，如果客户机远程发布到 b ，那么该出版物将作为主题 a 上的出版物传输到本地守护程序。 但是，当网桥发布到主题 a 上的本地守护程序时，该发布与网桥对本地主题 a 进行的预订相匹配。 预订为 topic "" out a b。 因此，该出版物将作为主题 b 上的出版物传输回远程代理。 现在，网桥已预订远程主题 b，并且循环将再次开始。

一些代理程序实施环路检测以防止发生环路。 但当不同类型的代理程序桥接在一起时，环路检测机制必须工作。 如果将 WebSphere MQ 桥接到设备的 WebSphere MQ Telemetry 守护程序，环路检测将不工作。 如果将设备的两个 IBM WebSphere MQ Telemetry 守护程序桥接在一起，环路检测将起作用。 缺省情况下将开启环路检测；请参阅 try_private。

```
connection Daemon1
topic "" in a b
topic "" out a b
```

图 *40: !*重新映射双向流动的出版物的相同主题

```
connection Daemon1
topic "" both a b
```

图 *41: !*使用 *both* 重新映射双向流动的出版物的相同主题。

第 137 页的图 39 中的配置与第 138 页的图 40 的相同。

IBM WebSphere MQ Telemetry daemon for devices 网桥连接的可用性

配置多个 IBM WebSphere MQ Telemetry daemon for devices 网桥连接地址以连接至第一个可用的远程代理程序。 如果此代理程序是一个多实例队列管理器，请提供该队列管理器的两个 TCP/IP 地址。 配置主连接以连接或重新连接到主服务器（可用时）。

连接网桥参数 addresses 是 TCP/IP 套接字地址列表。 网桥尝试轮流连接到每个地址，直到成功连接为止。 round_robin 和 start_type 连接参数可控制成功连接后如何使用地址。

如果 start_type 是 auto、manual 或 lazy，那么当连接失败时，网桥会尝试重新进行连接。 它会轮流使用每个地址，每次尝试连接间约有 20 秒钟延时。 如果 `start_type` 是 once，那么当连接失败时，网桥不会尝试自动进行重新连接。

如果 round_robin 是 true，那么网桥连接会尝试在列表的第一个地址处启动，并轮流尝试列表中的每个地址。 当遍历列表后，它会重新在第一个地址处启动。 如果列表中仅有一个地址，它将每 20 秒重新尝试一次。

如果 round_robin 是 false，那么将优先考虑列表中的第一个地址（称为“主服务器”）。 如果第一次尝试连接到主服务器失败，那么网桥会继续尝试在后台重新连接主服务器。 同时，网桥也会使用列表中的其他地址尝试进行连接。 如果后台尝试连接主服务器获得成功，网桥将从当前连接断开并切换到主服务器连接。

如果连接自动断开（如通过发出 **`connection_stop`** 命令），那么当重新启动连接时，它将重新尝试使用同一地址。 如果由于连接故障或远程代理程序断开连接导致连接断开，网桥将等待 20 秒钟的时间。 然后，它会尝试连接到列表中的下一个地址或相同的地址（如列表中仅有一个地址）。

连接到多实例队列管理器

在多实例队列管理器配置中，队列管理器会在具有不同的 IP 地址的两个不同的服务器上运行。 通常，不使用特定的 IP 地址配置遥测通道。 仅使用端口号进行配置。 启动遥测通道时，缺省情况下它会选择本地服务器上第一个可用的网络地址。

利用队列管理器所使用的两个 IP 地址来配置网桥连接的 addresses 参数。 将 round_robin 设置为 true。

如果活动队列管理器实例发生故障，那么队列管理器会切换到备用实例。 守护程序会检测与活动实例的连接是否中断，并尝试重新连接到备用实例。 它会使用为网桥连接配置的地址列表中的其他 IP 地址。

网桥连接到的队列管理器仍是同一队列管理器。 该队列管理器可恢复自身状态。 如果将 cleansession 设置为 false，那么网桥连接会话将恢复到故障转移前的同一状态。 连接会在一段延迟后恢复。 具有 “至少一次” 或 “最多一次” 服务质量的消息不会丢失，并且预订将继续工作。

重新连接时间取决于备用实例启动时重新启动的通道和客户机数以及使用的消息数。 重新建立连接前，网桥连接可能会多次尝试重新连接两个 IP 地址。

请勿使用特定的 IP 地址配置多实例队列管理器遥测通道。 IP 地址仅在一台服务器上有效。

如果您正在使用可管理 IP 地址的其他高可用性解决方案，那么可使用特定的 IP 地址配置遥测通道。

cleansession

网桥连接是 MQTT v3 客户机会话。 您可以控制连接是否启动新会话或恢复现有会话。 如果恢复现有会话，那么网桥连接会保留上一次会话的预订和保留的发布。

如果 addresses 列出多个 IP 地址，且此 IP 地址连接到由不同的队列管理器托管的遥测通道或不同的遥测守护程序，请不要将 cleansession 设置为 false。 会话状态不会在队列管理器或守护程序间进行传输。 尝试在不同的队列管理器或正在启动的新的会话中的守护程序结果中重新启动现有会话。 不确定消息已丢失，并且预订可能未按预期运行。

notifications

应用程序可跟踪是否使用通知来运行网桥连接。 通知是值为 1（已连接）或 0（断开连接）的发布。 它将发布到 notification_topic 参数定义的 *topicString* 。 *topicString* 的缺省值为 $SYS/broker/connection/ *clientIdentifier*/state。 缺省 *topicString* 包含前缀 $SYS。 通过定义以 $SYS 开头的主题过滤器来预订以 $SYS 开头的主题。 主题过滤器 # 将预订全部主题，但不会预订守护程序上以 $SYS 开头的主题。 在定义与应用程序主题空间不同的特殊系统主题空间时，请考虑 $SYS 事项。

通知 允许 IBM WebSphere MQ Telemetry daemon for devices 在网桥连接或断开连接时通知 MQTT 客户机。

keepalive_interval

keepalive_interval 网桥连接参数可设置网桥发送 TCP/IP ping 到远程服务器的时间间隔。 缺省时间间隔为 60 秒。 ping 可防止检测连接静止期的远程服务器或防火墙关闭 TCP/IP 会话。

clientid

网桥连接是 MQTT v3 客户机会话，并具有由网桥连接参数 clientid 设置的 clientIdentifier。 如果要将 cleansession 参数设置为 false，重新进行连接以恢复先前的会话，那么每个会话中使用的 clientIdentifier 必须相同。 clientid 的缺省值 *hostname.connectionName*，与之前相同。

安装、验证、配置和控制设备的 WebSphere MQ Telemetry 守护程序

安装、配置和控制该守护程序均基于文件完成。

将软件开发包复制到要运行守护程序的设备，即可安装该守护程序。

例如，运行 MQTT 客户机实用程序并连接到作为发布/预订代理程序的设备的 WebSphere MQ Telemetry 守护程序；请参阅使用设备的 WebSphere MQ Telemetry 守护程序作为发布/预订代理程序。

通过创建配置文件来配置守护程序；请参阅设备 WebSphere MQ Telemetry 守护程序的配置文件。

通过在文件 amqtdd.upd 中创建命令来控制正在运行的守护程序。 守护程序会每 5 秒钟阅读一次文件、运行一次命令并删除一次文件；请参阅设备 WebSphere MQ Telemetry 守护程序的命令文件。

设备侦听器端口的 WebSphere MQ Telemetry 守护程序

使用侦听器端口将 MQTT V3 客户机连接到设备的 WebSphere MQ Telemetry 守护程序。 您可以通过安装点和最大连接数来限制侦听器端口。

侦听器端口必须与连接到该端口的客户机的 MQTT 客户机 connect(serverURI) 方法上指定的端口号相对应。 缺省情况下，在客户机和守护程序上为 1883。

您可以通过在守护程序配置文件中设置全局定义**端口**来更改守护程序缺省端口。 您可以通过将**侦听器**定义添加到守护程序配置文件来设置特定的端口。

对于缺省端口以外的每个侦听器端口，您都可以指定一个安装点来隔离客户机。 通过安装点连接到端口的客户机与其他客户机相互隔离；请参阅第 140 页的『设备安装点的 WebSphere MQ Telemetry 守护程序』。

您可以限制可连接到任一端口的客户机数。 设置全局定义 max_connections 以与缺省端口的连接，或通过 max_connections 限制每个侦听器端口。

示例

配置文件示例，此配置文件将缺省端口从 1883 更改为 1880，并将端口 1880 的连接数限制为 10000。 端口 1884 的连接数限制为 1000。 连接到端口 1884 的客户机与连接到其他端口的客户机相互隔离。

```
port 1880
max_connections 10000
listener 1884
mount_point 1884/
max_connections 1000
```

设备安装点的 WebSphere MQ Telemetry 守护程序

您可以将安装点与 MQTT 客户机所使用的侦听器端口相关联，以连接到设备的 WebSphere MQ Telemetry 守护程序。 安装点使用连接到另一侦听器端口的 MQTT 客户机的一个侦听器端口来隔离由 MQTT 客户机交换的发布和预订。

通过安装点连接到侦听器端口的客户机永远不能与连接到其他任何侦听器端口的客户机直接交换主题。 不通过安装点连接到侦听器端口的客户机可发布或预订任一客户机的主题。 客户机不了解是否通过安装点进行连接，它不会识别客户机创建的主题字符串。

安装点是发布和预订的主题字符串前面的文本字符串。 它位于通过安装点连接到侦听器端口的客户机创建的所有主题字符串之前。 文本字符串会从发送到连接侦听器端口的客户机的所有主题字符串中除去。

如果侦听器端口没有安装点，那么连接到该端口的客户机所创建和接收的发布和预订的主题字符串不会改变。

创建带 / 结尾的安装点字符串。 这样，该安装点将成为安装点主题树的父主题。

示例

配置文件包含以下侦听器端口：

```
listener 1883
mount_point 1883/
listener 1884 127.0.0.1
mount_point 1884/
listener 1885
```

连接到端口 1883 的客户机会创建对 MyTopic 的预订。 守护程序会将该预订注册为 1883/MyTopic。 连接到端口 1883 的其他客户机会发布有关 MyTopic 主题的消息。 守护程序会将主题字符串更改为 1883/MyTopic，并搜索匹配的预订。 端口 1883 上的订户通过原始主题字符串 MyTopic 接收发布。 守护程序从主题字符串中除去安装点前缀。

连接到端口 1884 的其他客户机同样发布有关 MyTopic 主题的消息。 这一次，守护程序会将主题注册为 1884/MyTopic。 端口 1883 上的订户不会接收发布，因为不同的安装点会生成具有不同的主题字符串的预订。

连接到端口 1885 的客户机将发布有关 1883/MyTopic 主题的消息。 守护程序不会更改主题字符串。 端口 1883 上的订户将接收对 MyTopic 的发布。

设备服务质量、持久预订和保留发布的 WebSphere MQ Telemetry 守护程序

服务质量设置仅适用于正在运行的守护程序。 如果某个守护程序停止运行，无论是以受控方式还是由于发生故障，所使用的消息的状态都将丢失。 如果该守护程序停止运行，那么将无法保证以“至少一次”或“至多一次”的方式提供消息。 设备的 WebSphere MQ Telemetry 守护程序支持有限的持久性。 设置 **retained_persistence** 配置参数，以便在关闭守护程序时保存保留的发布和预订。

与 WebSphere MQ 不同， 设备的 WebSphere MQ Telemetry 守护程序不会记录持久数据。 会话状态、消息状态及保留的发布不会得到事务性保存。 缺省情况下，守护程序在停止运行时会废弃所有数据。 您可以设

置选项以定期检查预订和保留的发布。 守护程序停止运行时，总是丢失消息状态。 所有非保留的发布都将丢失。

将守护程序配置选项 Retained_persistence 设置为 true，以定期将保留的发布保存至文件。 守护程序重新启动时，将恢复上次自动保存的保留发布。 缺省情况下，当守护程序重新启动时，不会恢复客户机所创建的保留消息。

将守护程序配置选项 Retained_persistence 设置为 true，以定期将持久会话中创建的预订保存至文件。 如果 Retained_persistence 设置为 true，那么将复原客户机在将 CleanSession 设置为 false(“持久会话”) 的会话中创建的预订。 守护程序会在重新启动时恢复预订，从而开始接收发布。 如果将 CleanSession 设置为 false，客户机会在重新启动时接收发布。 缺省情况下，当守护程序停止运行时，不会保存客户机会话状态，因此，即使客户机将 CleanSession 设置为 false，也不会恢复预订。

Retained_persistence 是自动保存机制。 它可能不会保存最新保留的发布或预订。 您可以更改保存保留发布和预订的频率。 使用配置选项 autosave_on_changes 和 autosave_interval 设置每次保存的时间间隔或更改次数。

设置持久性示例配置

```
# 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
```

设备安全性的 WebSphere MQ Telemetry 守护程序

设备的 WebSphere MQ Telemetry 守护程序可对其连接的客户机进行认证，并使用凭证连接到其他代理程序以及控制对主题的访问。 通过使用不提供 SSL 支持的 WebSphere MQ Telemetry C 客户机来限制该守护程序提供的安全性。 因此，与该守护程序建立的连接不会加密，且无法使用证书进行认证。

缺省情况下，不开启任何安全性。

客户机的认证

MQTT 客户机可使用方法 MqttConnectOptions.setUserName 和 MqttConnectOptions.setPassword 设置用户名和密码。

根据密码文件中的条目检查客户机提供的用户名和密码，对连接到守护程序的客户机进行认证。 要启用认证，请创建密码文件并在守护程序配置文件中设置 password_file 参数；请参阅 password_file。

在守护程序配置文件中设置 allow_anonymous 参数，使未通过用户名或密码连接的客户机连接到检查认证情况的守护程序；请参阅 allow_anonymous。 设置 password_file 参数之后，如果某客户机未提供用户名或密码，那么将始终比照密码文件检查用户名和密码。

在守护程序配置文件中设置 clientid_prefixes 参数以限制与特定客户机的连接。 客户机必须具有以 clientid_prefixes 参数中列出的某个前缀开头的 clientIdentifiers；请参阅 clientid_prefixes。

网桥连接安全性

用于设备网桥连接的每个 WebSphere MQ Telemetry 守护程序都是一个 MQTT V3 客户机。 您可以为每个网桥连接设置用户名和密码，以作为守护程序配置文件中的网桥连接参数；请参阅用户名和密码。 网桥可针对代理程序进行自认证。

主题访问控制

如果客户机正在进行认证，那么守护程序还可为每个用户提供针对主题的控制访问功能。 根据客户机正在发布或预订的主题与访问控制文件中的访问主题字符串的匹配情况，守护程序可授予访问控制功能；请参阅 acl_file。

访问控制表包括两部分。 第一部分可控制对所有客户机的访问，包括匿名客户机。 第二部分含密码文件中针对所有用户的内容部分。 它列出了每个用户的特定访问控制功能。

示例

以下示例显示了安全性参数。

```
acl_file c:\WMQTDaemon\config\acl.txt
password_file c:\WMQTDaemon\config\passwords.txt
allow_anonymous true
connection Daemon1
username daemon1
password deamonpassword
```

图 *42:* 守护程序配置文件

```
Fred:Fredpassword
Barney:Barneypassword
```

图 *43:* 密码文件，*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/#
```

图 *44:* 访问控制文件，*acl.txt*

对 MQTT 客户机进行故障诊断

查找故障诊断任务以帮助您解决有关运行 MQTT 客户机的问题。

相关任务

第 150 页的『跟踪和调试 MQTT (Paho) Java 客户机』
缺省记录器使用称为 `java.util.logging` (JSR47) 的标准 Java 日志记录工具。 您可以使用配置文件或以编程方式对其进行配置。

第 152 页的『跟踪 MQTT JavaScript 客户机』
您可以通过将客户机 Web 应用程序变更为对连接的客户机对象调用方法，以使用 JavaScript 客户机来收集跟踪。

第 146 页的『跟踪遥测 (MQXR) 服务』
遵循下列指示信息来开始跟踪遥测服务、设置用于控制跟踪的参数以及查找跟踪输出。

第 147 页的『跟踪 MQTT v3 Java 客户机』
遵循下列指示信息来创建 MQTT Java 客户机跟踪和控制它的输出。

第 149 页的『跟踪 MQTT C 客户机』
设置环境变量 `MQTT_C_CLIENT_TRACE` 以跟踪 MQTT 客户机 C 应用程序

第 156 页的『解决问题：MQTT 客户机未连接』
解决 MQTT 客户机程序未能连接至遥测 (MQXR) 服务的问题。

第 158 页的『解决问题：MQTT 客户机连接已断开』
了解在成功建立连接并且运行较短或较长一段时间之后导致客户机抛出意外的 `ConnectionLost` 异常的原因。

第 158 页的『解决问题：MQTT 应用程序中丢失消息』
解决“丢失消息”这一问题。 消息是非持久消息、发送至错误的位置还是从未发送？ 错误编写的客户机程序可能会丢失消息。

第 160 页的『解决问题：Telemetry (MQXR) 服务未启动』
解决 Telemetry (MQXR) 服务未能启动的问题。 请检查 WebSphere MQ Telemetry 安装情况，以及是否缺少文件或者文件已移动或其权限有误。 请检查遥测 (MQXR) 服务所用路径是否找到遥测 (MQXR) 服务程序。

第 161 页的『解决问题：遥测服务未调用 JAAS 登录模块』
了解 JAAS 登录模块是否未由遥测 (MQXR) 服务调用，并配置 JAAS 以更正该问题。

第 163 页的『解决问题：启动或运行守护程序』
请参阅设备的 IBM WebSphere MQ Telemetry 守护程序控制台日志，开启跟踪，或使用本主题中的症状表对守护程序方面的问题进行故障诊断。

第 164 页的『解决问题：MQTT 客户机未连接到守护程序』
客户机未连接到守护程序，或者守护程序未连接到其他守护程序或 WebSphere MQ 遥测通道。

相关参考
第 143 页的『遥测日志、错误日志和配置文件的位置』
查找 IBM WebSphere MQ Telemetry 所使用的日志、错误日志和配置文件。

第 145 页的『MQTT v3 Java 客户机原因码』
在 MQTT v3 Java 客户机异常或可抛出异常中查找原因码的原因。

第 153 页的『关于将 SHA-2 密码套件用于 MQTT 客户机的系统需求』
对于从 IBM SR13 开始的 Java 6 ，您可以使用 SHA-2 密码套件来保护 MQTT 通道和客户机应用程序。 但是，缺省情况下，直到从 IBM SR4 开始的 Java 7 之后，才会启用 SHA-2 密码套件，因此在较早版本中，必须指定所需的套件。 如果您使用自己的 JRE 运行 MQTT 客户机，那么需要确保它支持 SHA-2 密码套件。 要使客户机应用程序使用 SHA-2 密码套件，客户机还必须将 SSL 上下文设置为支持传输层安全性 (TLS) V1.2 的值。

第 153 页的『基于 SSL 的移动消息传递 Web 应用程序的浏览器支持限制』
不同平台上的不同浏览器在功能方面也各有差异。 了解这些差异将帮助您配置应用程序、认证中心 (CA) 和客户机证书，以通过基于 SSL 的 JavaScript 的 MQTT 消息传递客户机 和 WebSockets 进行连接。

遥测日志、错误日志和配置文件的位置

查找 IBM WebSphere MQ Telemetry 所使用的日志、错误日志和配置文件。

注: 这些示例是针对 Windows 进行编写的。 更改语法以在 Linux 上运行示例

服务器端日志

IBM WebSphere MQ Telemetry 的安装向导将消息写入它的安装日志：

```
WMQ program directory\mqxr
```

遥测 (MQXR) 服务会将消息写入 WebSphere MQ 队列管理器错误日志中，将 FDC 文件写入 IBM WebSphere MQ 错误目录中：

```
WMQ data directory\Qmgrs\qMgrName\errors\AMQERR01.LOG
WMQ data directory\errors\AMQnnn.n.FDC
```

它还会编写遥测 (MQXR) 服务日志。 该日志显示服务启动时具有的属性，以及它在充当 MQTT 客户机的代理时所发现的错误。 例如，从客户机未创建的预订中取消预订。 日志路径为：

```
WMQ data directory\Qmgrs\qMgrName\errors\mqxr.log
```

IBM WebSphere MQ Explorer 所创建的 IBM WebSphere MQ Telemetry 样本配置使用命令 **runMQXRService** 启动遥测服务。 **runMQXRService** 位于 *WMQ Telemetry install directory*\bin 中。 它将写入：

```
WMQ data directory\Qmgrs\qMgrName\mqxr.stdout
WMQ data directory\Qmgrs\qMgrName\mqxr.stderr
```

修改 **runMQXRService** 以显示针对遥测 (MQXR) 服务配置的路径，或在启动遥测 (MQXR) 服务前回送初始化信息。

服务器端的配置文件

遥测通道和遥测 (MQXR) 服务

限制: 在将来的发行版中，可能会更改遥测通道配置文件的格式、位置、内容和解释。 您必须使用 IBM WebSphere MQ Explorer 来配置遥测通道。

IBM WebSphere MQ Explorer 将遥测配置保存在 Windows 上的 mqxr_win.properties 文件和 Linux 上的 mqxr_unix.properties 文件中。 属性文件保存在遥测配置目录中：

```
WMQ data directory\Qmgrs\qMgrName\mqxr
```

图 *45: Windows* 上的遥测配置目录

```
/var/mqm/qmgrs/qMgrName/mqxr
```

图 *46: Linux* 上的遥测配置目录

JVM

设置作为自变量传递到文件 java.properties 中的遥测 (MQXR) 服务的 Java 属性。 该文件中的属性会直接传递到运行遥测 (MQXR) 服务的 JVM。 它们在 Java 命令行上作为其他 JVM 属性传递。 命令行上设置的属性优先于从 java.properties 文件添加到命令行的属性。

在遥测配置所在的文件夹中查找 java.properties 文件，请参阅 第 144 页的图 45 和 第 144 页的图 46。

通过将每个属性指定为单独的一行来修改 java.properties。 完全按照您的想法安排每个属性的布局，以便将属性作为自变量传递至 JVM，例如：

```
-Xmx1024m
-Xms1024m
```

JAAS

遥测通道 JAAS 配置 中描述了 JAAS 配置文件，其中包括 IBM WebSphere MQ Telemetry 随附的样本 JAAS 配置文件 JAAS.config。

如果您配置 JAAS，那么您几乎必然要编写一个类以认证用户，从而替换标准 JAAS 认证过程。

要在遥测 (MQXR) 服务类路径使用的类路径中包含 Login 类，请提供 WebSphere MQ service.env 配置文件。

在 service.env 中设置 JAAS LoginModule 的类路径。 不能在 service.env 中使用 %classpath% 变量。 service.env 中的类路径添加至遥测 (MQXR) 服务定义中已设置的类路径。

通过将 echo set classpath 添加至 runMQXRService.bat， 显示遥测 (MQXR) 服务使用的类路径。 输出被发送至 mqxr.stdout。

service.env 文件的缺省位置为：

```
WMQ data directory\service.env
```

使用以下目录中每个队列管理器的 service.env 文件来覆盖这些设置：

```
WMQ data directory\Qmgrs\qMgrName\service.env
```

第 145 页的图 47 显示了要使用样本 LoginModule.class 的样本 service.env 文件。

```
CLASSPATH=WMQ Install Directory\mqxr\samples
```

注: service.env 不得包含任何变量。 替换 *WMQ Install Directory* 的实际值。

图 *47:* 样本 *service.env*（针对 *Windows*）

跟踪

IBM 维护工程师可能会要求您配置跟踪；请参阅第 146 页的『跟踪遥测 (MQXR) 服务』。 用于配置跟踪的参数存储在两个文件中：

```
WMQ data directory\Qmgrs\qMgrName\mqxr\trace.config
WMQ data directory\Qmgrs\qMgrName\mqxr\mqxrtrace.properties
```

客户端日志文件

IBM WebSphere MQ Telemetry 提供的 Java SE MQTT 客户机中的缺省文件持久性类将创建具有以下名称的文件夹: *clientIdentifier*-tcp*hostNameport* 或客户机工作目录中的 *clientIdentifier*-ssl*hostNameport* 。 此文件夹名称告诉您在连接尝试中所使用的 hostName 和 port。。 该文件夹包含由持久性类存储的消息。 在已经成功地传递消息之后，就会删除这些消息。

当具有清除会话的客户机结束时，就会删除此文件夹。

如果已打开客户机跟踪，那么缺省情况下会将未格式化的日志存储在客户机工作目录中。 跟踪文件称为 mqtt-*n*.trc

客户端的配置文件

使用 Java 属性文件为 MQTT Java 客户机设置跟踪和 SSL 属性，或者以编程方式设置属性。 使用 JVM -D 开关将属性传递到 MQTT Java 客户机: 例如，

```
Java -Dcom.ibm.micro.client.mqttv3.trace=c:\\MqttTrace.properties
     -Dcom.ibm.ssl.keyStore=C:\\MyKeyStore.jks
```

请参阅第 147 页的『跟踪 MQTT v3 Java 客户机』。有关 MQTT 客户机库的客户机 API 文档的链接，请参阅 MQTT 客户机编程参考。

MQTT v3 Java 客户机原因码

在 MQTT v3 Java 客户机异常或可抛出异常中查找原因码的原因。

表 *5: MQTT v3 Java* 客户机原因码

原因码	值	原因
REASON_CODE_BROKER_UNAVAILABLE	3	
REASON_CODE_CLIENT_ALREADY_CONNECTED	32100	客户机已连接。
REASON_CODE_CLIENT_ALREADY_DISCONNECTED	32101	客户机已断开连接。
REASON_CODE_CLIENT_DISCONNECT_PROHIBITED	32107	当已经从 MqttCallback 上的方法尝试调用 MqttClient.disconnect 时，就会抛出此异常。
REASON_CODE_CLIENT_DISCONNECTING	32102	客户机当前正在断开连接，无法接受任何新工作。
REASON_CODE_CLIENT_EXCEPTION	0	客户机遇到了异常。

表 5: *MQTT v3 Java* 客户机原因码 (继续)		
原因码	**值**	**原因**
REASON_CODE_CLIENT_NOT_CONNECTED	32104	客户机未连接至服务器。
REASON_CODE_CLIENT_TIMEOUT	32000	客户机在等待来自服务器的响应时超时。
REASON_CODE_FAILED_AUTHENTICATION	4	由于用户名或密码错误，因此向服务器进行认证时已失败。
REASON_CODE_INVALID_CLIENT_ID	2	服务器已拒绝所提供的客户机标识。
REASON_CODE_INVALID_PROTOCOL_VERSION	1	服务器不支持所请求的协议版本。
REASON_CODE_NO_MESSAGE_IDS_AVAILABLE	32001	这是由于没有新的消息标识可用而导致的内部错误。
REASON_CODE_NOT_AUTHORIZED	5	未授权执行所请求的操作。
REASON_CODE_SERVER_CONNECT_ERROR	32103	无法连接至服务器。
REASON_CODE_SOCKET_FACTORY_MISMATCH	32105	服务器 URI 与所提供的 SocketFactory 不匹配。
REASON_CODE_SSL_CONFIG_ERROR	32106	SSL 配置错误。
REASON_CODE_UNEXPECTED_ERROR	6	发生了意外错误。

跟踪遥测 (MQXR) 服务

遵循下列指示信息来开始跟踪遥测服务、设置用于控制跟踪的参数以及查找跟踪输出。

开始之前

跟踪是一项支持功能。 如果 IBM 维护工程师要求您跟踪遥测 (MQXR) 服务，请遵循下列指示信息。 产品文档未说明跟踪文件的格式或者如何使用它来调试客户机。

关于此任务

您可以使用 IBM WebSphere MQ **strmqtrc** 和 **endmqtrc** 命令来启动和停止 IBM WebSphere MQ 跟踪。 **strmqtrc** 可捕获遥测 (MQXR) 服务跟踪情况。 使用 **strmqtrc** 时，在启动遥测服务跟踪前会有长达几秒的延迟。 要获取有关 IBM WebSphere MQ 跟踪的更多信息，请参阅使用跟踪。 或者，可使用以下过程来跟踪遥测 (MQXR) 服务：

过程

1. 设置跟踪选项来控制详细信息量以及跟踪大小。 这些选项适用于通过 **strmqtrc** 或 **controlMQXRChannel** 命令启动的跟踪。

   设置以下文件中的跟踪选项：

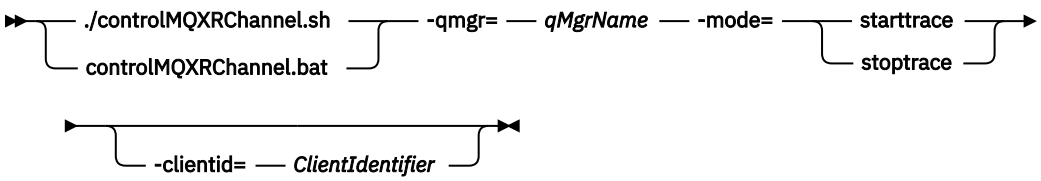
   ```
   mqxrtrace.properties
   trace.config
   ```

   这些文件位于以下目录中：

   - 在 Windows 上， *WebSphere MQ data directory*\qmgrs*qMgrName* \mqxr。

- 在 Linux 上， `var/mqm/qmgrs/ qMgrName/mqxr`。

2. 在以下目录中打开命令窗口：
   - 在 Windows 系统上， *WebSphere MQ installation directory*\mqxr\bin 。
   - 在 Linux 系统 /opt/mqm/mqxr/bin 上。
3. 运行以下命令以启动 SYSTEM.MQXR.SERVICE 跟踪：



必需参数

qmgr=*qmgrName*
将 *qmgrName* 设置为队列管理器名称

mode=starttrace| stoptrace
设置 starttrace 以开始跟踪；或设置 stoptrace 以结束跟踪

可选参数

clientid=*ClientIdentifier*
将 *ClientIdentifier* 设置为客户机的 `ClientIdentifier`。 clientid 将跟踪过滤到一台客户机。 多次运行跟踪命令以跟踪多台客户机。

例如：

```
/opt/mqm/mqxr/bin/controlMQXRChannel.sh -qmgr=QM1 -mode=starttrace -clientid=
problemclient
```

结果

要查看跟踪输出，请切换至以下目录：

- 在 Windows 上， *WebSphere MQ data directory*\trace。
- 在 Linux 上，`/var/mqm/trace`。

跟踪文件名为 `mqxr_PPPPP.trc`，其中 PPPPP 是进程标识。

相关参考

strmqtrc

跟踪 MQTT v3 Java 客户机

遵循下列指示信息来创建 MQTT Java 客户机跟踪和控制它的输出。

开始之前

本主题仅适用于 IBM WebSphere MQ V7.5.0.0。 有关描述在更高版本中跟踪 Java 客户机的信息，请参阅第 150 页的『跟踪和调试 MQTT (Paho) Java 客户机』。

跟踪是一项支持功能。 如果 IBM 维护工程师要求您跟踪 MQTT Java 客户机，请遵循下列指示信息。 产品文档未说明跟踪文件的格式或者如何使用它来调试客户机。

仅对 WebSphere MQ Telemetry Java 客户机进行跟踪。

关于此任务

注: 这些示例是针对 Windows 进行编写的。 更改语法以在 Linux 上运行示例 [2].

[2] Java 使用正确的路径定界符。 您可以将属性文件中的定界符编码为 '/' 或 '\\'; '\' 是转义字符

过程

1. 创建一个包含跟踪配置的 Java 属性文件。

   在此属性文件中指定下列可选属性。 如果一个属性关键字指定了多次，那么最后一个属性关键字实例将设置此属性。

   a) `com.ibm.micro.client.mqttv3.trace.outputName`

   要将跟踪文件写入的目录。 它缺省设置为客户机的工作目录。 跟踪文件称为 `mqtt-n.trc`。

   ```
   java com.ibm.micro.client.mqttv3.trace.outputName=c:\\MQTT_Trace
   ```

   b) `com.ibm.micro.client.mqttv3.trace.count`

   要写入的跟踪文件的数目。 缺省值是大小不受限制的一个文件。

   ```
   java com.ibm.micro.client.mqttv3.trace.count=5
   ```

   c) `com.ibm.micro.client.mqttv3.trace.limit`

   要写入的文件的最大大小，缺省值为 500000。 仅当请求了多个跟踪文件时，才会应用此限制。

   ```
   java com.ibm.micro.client.mqttv3.trace.limit=100000
   ```

   d) `com.ibm.micro.client.mqttv3.trace.client.clientIdentifier.status`

   根据客户机来打开或关闭跟踪。 如果 *clientIdentifier*=*，那么将对所有客户机打开或关闭跟踪。 缺省情况下，将对所有客户机关闭跟踪。

   ```
   java com.ibm.micro.client.mqttv3.trace.client.*.status=on
   ```

   ```
   java com.ibm.micro.client.mqttv3.trace.client.Client10.status=on
   ```

2. 使用系统属性将跟踪属性文件传递至 JVM。

   ```
   java -Dcom.ibm.micro.client.mqttv3.trace=c:\\MqttTrace.properties
   ```

3. 运行客户机。
4. 将跟踪文件从二进制编码转换为文本或 `.html`。 使用以下命令：

   ```
   java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter [-i traceFile] [-o
   outputFile] [-h] [-d
   time]
   ```

   变量位置在：

   -?
   显示帮助

   -i traceFile
   必需。 在输入文件（例如，`mqtt-0.trc`）中传递。

   -o outputFile
   必需。 定义输出文件（例如，`mqtt-0.trc.html` 或 `mqtt-0.trc.txt`）。

   -h
   输出为 HTML。 输出文件扩展名必须是 `.html`。 如果未指定，那么输出为纯文本。

   -d time
   如果毫秒级时差大于或等于 (>=) time，请使用 * 使行缩进。 不适用于 HTML 输出。

   下例将输出 HTML 格式的跟踪文件

   ```
   java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
   mqtt-0.trc.html -h
   ```

第二个示例将跟踪文件输出为纯文本，其中毫秒差为 50 或更高的任何连续时间戳记均使用星号 (*) 缩进。

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.txt -d 50
```

最后一个示例将跟踪文件输出为纯文本：

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.txt
```

跟踪 MQTT C 客户机

设置环境变量 MQTT_C_CLIENT_TRACE 以跟踪 MQTT 客户机 C 应用程序

开始之前

MQTT C 客户机跟踪可用于预构建的 Windows 和 Linux MQTT C 客户机库以及您亲自构建的 iOS 库。

关于此任务

将环境变量 MQTT_C_CLIENT_TRACE 设置为包含跟踪输出的文件的路径。 跟踪输出将写入该文件中。

过程

请先设置 MQTT_C_CLIENT_TRACE=mqttcclient.log，然后运行 MQTT 客户机 C 应用程序。

a) 例如，修改 第 23 页的『C MQTT 客户机入门』中的样本脚本：

```
@echo off
setlocal
set MQTT_C_CLIENT_TRACE=mqttcclient.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) 从 %sdkroot%/sdk/client/c/samples 目录运行该脚本。

结果

跟踪输出文件以如下行开头：

```
============================================================
                  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
```

相关任务

第 23 页的『C MQTT 客户机入门』
在任何可编译 C 源代码的平台上启动并运行样本 C MQTT 客户机。 验证是否可以使用 IBM MessageSight 或 IBM WebSphere MQ 作为 MQTT 服务器运行样本 MQTT Client for C。

跟踪和调试 MQTT (Paho) Java 客户机

缺省记录器使用称为 `java.util.logging` (JSR47) 的标准 Java 日志记录工具。 您可以使用配置文件或以编程方式对其进行配置。

关于此任务

注: Paho Java 客户机仅适用于 IBM WebSphere MQ 版本 7.5.0.1 和更高版本。 有关描述在 IBM WebSphere MQ V 7.5.0.0 中跟踪 Java 客户机的信息，请参阅 第 147 页的『跟踪 MQTT v3 Java 客户机』。

注: 跟踪是一项支持功能。 如果 IBM 服务工程师要求您跟踪 MQTT Java 客户机，请遵循以下指示信息。 产品文档未说明跟踪文件的格式或者如何使用它来调试客户机。 跟踪仅适用于 IBM WebSphere MQ Telemetry Java 客户机。

使用配置文件的最简单方法是在 `java.util.logging.config.file` 属性中指定其名称。

在 `org.eclipse.paho.client.mqttv3.logging` 包中提供了有效的属性文件 `jsr47min.properties`。

可以采用一些方式使用 JSR47 日志记录工具:

- 从所选的一组包中收集消息
- 在日志级别或低于日志级别收集消息
- 为日志消息选择多个目标
- 通过提供内置日志记录器，以写入文件并控制所使用文件的大小和数量
- 通过提供内置日志记录器，以写入内存并支持基于触发器写出内存中的消息
- 如果使用 MQTT 客户机库的应用程序也使用 JSR47 进行检测，那么将混合应用程序和客户机库中的消息。

将提供实用程序类以帮助收集调试信息。 此类包含先前描述的日志和跟踪消息，但可以从 Paho 客户机内部收集诸如 Java 系统属性和变量值之类的信息。

在公共类 Debug 中提供了调试工具，该类是 `org.eclipse.paho.client.mqttv3.util` 包的一部分。 可以对异步和同步 MQTT 客户机对象使用方法 getDebug() 来获取 Debug 的实例。

例如:

```
MqttClient cl = new MqttClient();
Debug d = cl.getDebug();
```

方法 dumpClientDebug() 将转储最大的调试信息量。 必须启用日志工具才能捕获写入其中的完整调试信息。 要捕获完整的调试信息，请在已知发生问题时（例如，在发生特定异常后）调用转储方法。

过程

1. 创建配置文件或使用提供的 jsr47min.properties。

   如果您使用提供的属性文件，请检查推送触发器是否已设置为正确的错误级别。 缺省情况下，会设置为 Severe 级别错误，可能需要将跟踪持续写入文件，而不是将其存储在内存中直至发生错误。 为此，请将

   ```
   java.util.logging.MemoryHandler.push=SEVERE
   ```

   到

   ```
   java.util.logging.MemoryHandler.push=ALL
   ```

2. 使用系统属性将跟踪配置文件传递至 JVM。

   如果您使用 jsr4min.properties，那么为：

   ```
   java -Djava.util.logging.config.file=C:\temp\jsr47min.properties
   ```

3. 运行客户机。

结果

当发生异常或问题时，Paho Debug 类会将内存中跟踪写入配置的文件目标。

当生成该文件时，不会自动将跟踪写入该文件，只有当命中推送触发器或调试类导致写入跟踪时，才会写入。 后者可能需要更改应用程序代码。

在创建文件处理程序时将每行写入文件处理程序。 您可以通过配置 FileHandler，来控制写出消息的格式。 随 Paho 提供的定制文件处理程序比 SimpleHandler 写出的多，比随 JRE 提供的 XMLHandler 写出的少。 使用 Paho 日志格式化程序的跟踪记录采用以下格式：

```
Level   Data and Time   Class   Method   Thread   clientID   Message
```

示例

将提供有效的属性文件 `jsr47min.properties`。 该文件包含用于收集跟踪的建议配置，有助于解决与 Paho MQTT 客户机相关的问题。 它配置在内存中持续收集跟踪，而最大限度地减少对性能的影响。 当推送触发器触发或者发出特定的推送请求时，会将内存中的跟踪推送到配置的目标处理程序。 缺省推送触发器是 Severe 级别消息，即一个中断连接。 缺省情况下，此时会将收集在内存中的跟踪写入指定的文件。 缺省情况下，该文件是标准 `java.util.logging.FileHandler`。 您可以使用 Paho Debug 类将内存跟踪推送到其目标。

可以在 java.util.logging 包的 Javadoc 中找到 JSR47 的完整详细信息。

```
# 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.SimpleLogFormatte
r
```

下一步做什么

为了以编程方式收集跟踪，提供了实用程序类来帮助收集调试信息。 此类包含先前描述的日志和跟踪消息，但可以从 Paho 客户机内部收集诸如 Java 系统属性和变量值之类的信息。

在公共类 Debug 中提供了调试工具，该类是 `org.eclipse.paho.client.mqttv3.util` 包的一部分。可以对异步和同步 MQTT 客户机对象使用方法 getDebug() 来获取 Debug 的实例。

例如：

```
MqttClient cl = new MqttClient();
Debug d = cl.getDebug();
```

方法 dumpClientDebug() 将转储最大的调试信息量。 必须启用日志工具才能捕获写入其中的完整调试信息。 要捕获完整的调试信息，请在已知发生问题时（例如，在发生特定异常后）调用转储方法。

跟踪 MQTT JavaScript 客户机

您可以通过将客户机 Web 应用程序变更为对连接的客户机对象调用方法，以使用 JavaScript 客户机来收集跟踪。

关于此任务

要收集跟踪，可以使用以下方法：

- `client.startTrace()` 对客户机启动跟踪。
- `client.stopTrace()` 对客户机停止跟踪。
- `client.getTraceLog()` 返回当前的跟踪缓冲区。

您可以输出跟踪缓冲区以发送至 IBM 软件支持。 可采用一些方式执行此操作。 以下示例显示启动跟踪，然后输出发送至控制台和指定的电子邮件地址，最后停止跟踪。

```
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("ConnectionLost:"+responseObject.errorMessage);
    console.log(client.getTraceLog());
    window.location="mailto:helpdesk@"+location.hostname+
        "?Subject=Web%20Messaging%20Utility%20Trace&body="+
        client.getTraceLog().join("%0A");
```

```
    client.stopTrace();
};
```

样本输出：

```
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,
```

V 7.5.0.2 关于将 SHA-2 密码套件用于 MQTT 客户机的系统需求

对于从 IBM SR13 开始的 Java 6，您可以使用 SHA-2 密码套件来保护 MQTT 通道和客户机应用程序。但是，缺省情况下，直到从 IBM SR4 开始的 Java 7 之后，才会启用 SHA-2 密码套件，因此在较早版本中，必须指定所需的套件。如果您使用自己的 JRE 运行 MQTT 客户机，那么需要确保它支持 SHA-2 密码套件。要使客户机应用程序使用 SHA-2 密码套件，客户机还必须将 SSL 上下文设置为支持传输层安全性 (TLS) V1.2 的值。

对于从 IBM SR4 开始的 Java 7，缺省情况下会启用 SHA-2 密码套件。对于来自 IBM， SR13 和更高发行版的 Java 6，如果在未指定密码套件的情况下定义 MQTT 通道，那么该通道将不接受来自使用 SHA-2 密码套件的客户机的连接。要使用 SHA-2 密码套件，您必须在通道定义中指定所需的套件。这 使 MQTT 服务器在建立连接前启用该套件。这还意味着只有使用指定套件的客户机应用程序才可以连接到此通道。

对于 Java 的 MQTT 客户机，存在类似的限制。如果客户机代码正在来自 IBM 的 Java 1.6 JRE 上运行，那么必须显式启用所需的 SHA-2 密码套件。为了使用这些套件，客户机还必须将 SSL 上下文设置为支持传输层安全性 (TLS) 协议 V1.2 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 JavaScript 的 MQTT 消息传递客户机 且还支持 TLS 1.2 协议的浏览器，因此如果要与 JavaScript 客户机建立 SHA-2 连接，它是唯一可以使用的浏览器。

有关当前受支持的密码套件的列表，请参阅相关链接。

相关概念

第 95 页的『使用 SSL 进行客户机认证的 MQTT 客户机配置』
要使用 SSL 来认证 MQTT 客户机，客户机将使用 SSL 来连接至遥测通道。它必须指定与配置为认证 SSL 客户机的遥测通道相对应的 TCP 端口。

第 97 页的『使用 SSL 进行通道认证的 MQTT 客户机配置』
要使用 SSL 对遥测通道进行认证，客户机必须使用 SSL 来连接至遥测通道。它必须指定与为 SSL 配置的遥测通道相对应的端口。配置中必须包括一个受口令保护的密钥库，而此密钥库中包含服务器的私下签名的数字证书。

V 7.5.0.1 基于 SSL 的移动消息传递 Web 应用程序的浏览器支持限制

不同平台上的不同浏览器在功能方面也各有差异。了解这些差异将帮助您配置应用程序、认证中心 (CA) 和客户机证书，以通过基于 SSL 的 JavaScript 的 MQTT 消息传递客户机 和 WebSockets 进行连接。

使用基于 SSL 的 JavaScript 的移动消息传递是一项全新的技术，因此不同的浏览器和平台组合在实施该功能的方式和程度方面会略有差异。下表概述了当前适用于和不适用于每个浏览器（Firefox、Chrome、Internet Explorer 和 Safari）和平台（Windows、Linux、Mac、iOS 和 Android）组合的内容。

表 *6:* 按平台和浏览器组合划分的 *SSL* 支持. 对于每个浏览器和平台组合，该表指定了是否支持 SSL 匿名和非匿名连接，以及浏览器与所有认证中心 (CA) 和客户机证书的兼容度。

浏览器	SSL 支持 (Y/N)	SSL 与任何 CA (Y/N) 兼容	详细信息
Firefox 桌面。	SSL 匿名 - 是 SSL 非匿名 - 是	SSL 匿名 - 是 SSL 非匿名 - 是	将 CA 和客户机证书添加至浏览器。 Firefox 使用自己的证书库。 要导入 CA 证书，请单击 **工具 > 选项 > 高级 > 加密 > 查看证书 > 权限 > 导入** 要导入客户机证书，请单击 **工具 > 选项 > 高级 > 加密 > 查看证书 > 您的证书 > 导入** 要启用安全连接，请在 URL 中指定 `https://`。Firefox 提供了用于自动选择证书或每次询问的选项。 Firefox 还提供了使用 SSL 3.0 或 TLS 1.0 的选项；请确保同时选中这两个选项。
Chrome 桌面。	SSL 匿名 - 是 SSL 非匿名 - 是	SSL 匿名 - 是 SSL 非匿名 - 是	使用浏览器将 CA 和客户机证书添加到操作系统证书库中，将与其他软件共享该证书库。 要导入 CA 证书，请单击 **设置 > 显示高级设置 > 管理证书 > 可信根认证中心 > 导入** 要导入客户机证书，请单击 **设置 > 显示高级设置 > 管理证书 > 个人 > 导入** 要启用安全连接，请在 URL 中指定 `https://`。Chrome 将提示您在多个选项中进行选择；请根据配置的是匿名连接还是非匿名连接，选择正确的选项。

表 *6:* 按平台和浏览器组合划分的 *SSL* 支持. 对于每个浏览器和平台组合，该表指定了是否支持 SSL 匿名和非匿名连接，以及浏览器与所有认证中心 (CA) 和客户机证书的兼容度。 *(*继续*)*

浏览器	SSL 支持 (Y/N)	SSL 与任何 CA (Y/N) 兼容	详细信息
Internet Explorer.	SSL 匿名 - 是 SSL 非匿名 - 是	SSL 匿名 - 是 SSL 非匿名 - 是	建立非匿名 SSL 连接时，将提示您选择正确的客户机证书。 Internet Explorer 使用 Windows 证书库，将与其他软件共享该证书库。 要导入 CA 证书，请单击 **工具 > Internet 选项 > 内容 > 证书 > 可信根认证中心 > 导入** 要导入客户机证书，请单击 **工具 > Internet 选项 > 内容 > 证书 > 个人 > 导入**
Safari 桌面。	SSL 匿名 - 是 SSL 非匿名 - 是	SSL 匿名 - 是 SSL 非匿名 - 是	使用浏览器将 CA 和客户机证书添加到操作系统证书库中，将与其他软件共享该证书库。
Firefox on Android	SSL 匿名 - 是 SSL 非匿名 - 是	SSL 匿名 - 是 SSL 非匿名 - 否	非匿名: 客户机证书未生效，因为您没有满足将 CA 添加至 Firefox 的列表中的需求。 要导入客户机证书，请单击 **设置 > 安全性 > 凭证存储器**。 如果您的证书由列表中可信的 CA 签署，那么您可以建立安全连接。
Chrome on Android	SSL 匿名 - 是 SSL 非匿名 - 是	SSL 匿名 - 是 SSL 非匿名 - 否	非匿名: 客户机证书未生效，因为您没有满足将 CA 添加至 Chrome 的列表中的需求。 **注:** Google 计划在 Chrome V27 中支持此内容。 这是自 V18 起就存在的公认缺陷。 要导入客户机证书，请单击 **设置 > 安全性 > 凭证存储器**。 如果您的证书由列表中可信的 CA 签署，那么您可以建立安全连接。

表 6: 按平台和浏览器组合划分的 *SSL* 支持. 对于每个浏览器和平台组合，该表指定了是否支持 SSL 匿名和非匿名连接，以及浏览器与所有认证中心 (CA) 和客户机证书的兼容度。 *(继续)*

浏览器	SSL 支持 (Y/N)	SSL 与任何 CA (Y/N) 兼容	详细信息
Safari on iOS	SSL 匿名 - 是 SSL 非匿名 - 是	SSL 匿名 - 是 SSL 非匿名 - 否	非匿名: 该设备不信任客户机证书，即使同时安装了 CA 证书也如此。 Safari 使用设备证书库。要导入到此商店中，请单击 **设置 > 常规 > 概要文件**，并从 Web 页面提供 CA 或客户机证书，或者通过电子邮件将其发送给您自己。
Chrome on iOS	SSL 匿名 - 是 SSL 非匿名 - 否	SSL 匿名 - 否 SSL 非匿名 - 否	匿名: 仅 Apple 应用程序可访问 iOS 系统根库。 因此，Chrome 必须使用自己的 CA 列表，而您无法进行添加。 非匿名: 客户机证书未生效，因为您没有满足将 CA 添加至列表的需求。

相关任务

第 70 页的『通过 SSL 连接 JavaScript 的 MQTT 消息传递客户机 和 WebSockets』
通过将 JavaScript 的 MQTT 消息传递客户机 样本 HTML 页面与 SSL 和 WebSocket protocol 配合使用，将 Web 应用程序安全地连接到 IBM WebSphere MQ 。

相关信息

Mozilla：(SSL) Firefox 使用 Android CA 存储器还是自己的 CA 存储器?
Chromium：发布 134418 - 实施客户机证书支持
无法打开 https 站点，因为在 ie10 上无可信证书

解决问题：MQTT 客户机未连接

解决 MQTT 客户机程序未能连接至遥测 (MQXR) 服务的问题。

开始之前

此问题发生于服务器、客户机还是连接? 您是否已编写自己的 MQTT v3 协议处理客户机或使用 C 或 Java WebSphere MQTT 客户机的 MQTT 客户机应用程序?

在服务器上运行与 WebSphere MQ Telemetry 一起提供的验证应用程序，并检查遥测通道和遥测 (MQXR) 服务运行是否正常。 然后，将验证应用程序传递至客户机，并在客户机中运行此验证应用程序。

关于此任务

有许多原因会造成 MQTT 客户机可能未连接至遥测服务器，或者使您可以断定 MQTT 客户机尚未连接至遥测服务器。

过程

1. 请考虑可以从遥测 (MQXR) 服务返回至 `MqttClient.Connect` 的原因码中得到哪些推论。 它是哪种类型的连接故障?

选项	描述
REASON_CODE_INVALID_PROTOCOL_VERSION	请确保套接字地址对应于遥测通道，并且您未对另一个代理使用同一个套接字地址。
REASON_CODE_INVALID_CLIENT_ID	检查客户机标识是否不超过 23 个字节，并且仅包含范围内的字符: `A-Z, a-z, 0-9, './_%`
REASON_CODE_INVALID_DESTINATION	请检查客户机标识是否与队列管理器名称不同。
REASON_CODE_SERVER_CONNECT_ERROR	请检查遥测 (MQXR) 服务和队列管理器是否正常运行。 使用 **netstat** 来检查是否未将套接字地址分配给另一个应用程序。

   如果您已经编写了 MQTT 客户机库而不使用 IBM WebSphere MQ Telemetry 所提供的其中一个库，请查看 CONNACK 返回码。

   从这三个错误中您可以推断客户机已连接至遥测 (MQXR) 服务，但该服务已发现错误。

2. 请考虑当遥测 (MQXR) 服务未作响应时，可以从客户机生成的原因码中得到哪些推论:

选项	描述
REASON_CODE_CLIENT_EXCEPTION **REASON_CODE_CLIENT_TIMEOUT**	在服务器中查找 FDC 文件；请参阅第 143 页的『服务器端日志』。 当遥测 (MQXR) 服务检测到客户机发生超时时，它会编写一个首次故障数据捕获 (FDC) 文件。 每当连接意外断开时，它就会编写一个 FDC 文件。

   遥测 (MQXR) 服务可能不会对客户机以及客户机到期时的超时情况做出响应。 仅当应用程序设置了无限期超时时， WebSphere MQ Telemetry Java 客户机才会挂起。 在由于未经诊断的连接问题而导致为 `MqttClient.Connect` 设置的超时到期之后，客户机就会抛出这些异常之一。

   除非您找到了与连接故障相关联的 FDC 文件，否则您无法推断出客户机已尝试连接至服务器:

   a) 确认客户机是否已发送连接请求。

      使用一个工具（例如 **tcpmon**，可从 https://java.net/projects/tcpmon 获得此工具）来检查 TCPIP 请求

   b) 客户机所使用的远程套接字地址与为遥测通道定义的套接字地址相匹配吗?

      IBM WebSphere MQ Telemetry 提供的 Java SE MQTT 客户机中的缺省文件持久性类将创建具有以下名称的文件夹: *clientIdentifier*-tcp*hostNameport* 或客户机工作目录中的 *clientIdentifier*-ssl*hostNameport* 。 此文件夹名称告诉您在连接尝试中所使用的 `hostName` 和 `port`。；请参阅第 145 页的『客户端日志文件』。

   c) 您可以对远程服务器地址执行 ping 操作吗?

   d) 服务器上执行的 **netstat** 命令指出遥测通道正在客户机要连接至的端口上运行吗?

3. 请检查遥测 (MQXR) 服务是否发现客户机请求问题。

   遥测 (MQXR) 服务会将其检测到的错误写入 `mqxr.log` 中，队列管理器会将相关错误写入 `AMQERR01.LOG` 中；请参阅

4. 尝试通过运行另一个客户机来找出问题。

   - 使用同一遥测通道来运行 MQTT 样本应用程序。
   - 运行 **wmqttSample** GUI 客户机来验证连接。 通过下载 SupportPac IA92 获取 **wmqttSample** 。

     注: 较低版本的 IA92 不包含 MQTT v3 Java 客户机库。

在服务器平台上运行样本程序，以消除网络连接的不确定性，然后在客户机平台上运行样本。

5. 要检查的其他事项：

   a) 有大量的 MQTT 客户机试图同时连接吗？

   遥测通道有一个队列，用来缓存储备的入局连接。 一秒钟处理超过 10,000 个连接。 可通过使用 IBM WebSphere MQ Explorer 中的“遥测通道”向导来配置储备缓冲区的大小。 它的缺省大小是 4096。 请检查是否尚未将储备配置为一个较小值。

   b) 遥测 (MQXR) 服务和队列管理器是否仍在运行？

   c) 客户机已连接至一个已切换其 TCPIP 地址的高可用性队列管理器吗？

   d) 防火墙将有选择地过滤出站或者返回数据包吗？

解决问题：MQTT 客户机连接已断开

了解在成功建立连接并且运行较短或较长一段时间之后导致客户机抛出意外的 `ConnectionLost` 异常的原因。

开始之前

MQTT 客户机已成功连接。 客户机可能已启动较长一段时间。 如果不同客户机之间的启动时间间隔较短，那么从成功连接到连接断开之间的时间可能较短。

区分“已断开的连接”与“已成功建立连接、但是稍后又断开的连接”并不难。 已断开的连接是 MQTT 客户机通过调用 `MqttCallback.ConnectionLost` 方法来定义的。 仅在已成功建立连接之后才调用此方法。 症状不同于 `MqttClient.Connect` 在接收到否定确认信息或者超时之后抛出异常。

如果 MQTT 客户机应用程序未使用 IBM WebSphere MQ 提供的 MQTT 客户机库，那么此症状取决于客户机。 在 MQTT V3 协议中，症状是缺少对向服务器发出的请求的及时响应，或者是 TCP/IP 连接发生故障。

关于此任务

MQTT 客户机调用 `MqttCallback.ConnectionLost` 并产生可抛出异常，以作为在接收肯定的连接确认信息之后遇到了任何服务器端问题的响应。 当 MQTT 客户机从 `MqttTopic.publish` 和 `MqttClient.subscribe` 返回时，会将请求传输至负责发送和接收消息的 MQTT 客户机线程。 通过将可抛出异常传递至 `ConnectionLost` 回调方法，从而以异步方式报告服务器端错误。

如果遥测 (MQXR) 服务断开连接，那么它始终会编写一个首次故障数据捕获文件。

过程

1. 是否已启动使用了同一 `ClientIdentifier` 的另一客户机？

   如果使用同一 `ClientIdentifier` 启动另一客户机，或重新启动同一客户机，那么将断开与第一个客户机的首次连接。

2. 是否客户机访问了一个未授权它发布或预订的主题？

   如果遥测服务代表客户机执行的任何操作返回 `MQCC_FAIL`，那么将导致此服务断开客户机连接。

   原因码未返回给客户机。

   - 在 `mqxr.log` 和 `AMQERR01.LOG` 文件中查找与客户机相连的队列管理器的日志消息；请参阅第 143 页的『服务器端日志』。

3. TCP/IP 连接是否已断开？

   防火墙可能具有较低的超时设置，以将 TCPIP 连接标记为不活动状态并且断开连接。

   - 使用 `MqttConnectOptions.setKeepAliveInterval` 来缩短不活动 TCPIP 连接时间。

解决问题：MQTT 应用程序中丢失消息

解决“丢失消息”这一问题。 消息是非持久消息、发送至错误的位置还是从未发送？ 错误编写的客户机程序可能会丢失消息。

开始之前

您有多肯定您所发送的消息已丢失？ 您可以推断是因为未收到消息而丢失了此消息吗？ 如果消息是发布，而此消息丢失，那么该消息由发布者发送，还是发送到订户？ 或者是因为预订丢失了，而代理未将该预订的发布发送至订户？

如果解决方案涉及到使用集群或者发布/预订层次结构的分布式发布/预订，那么有许多配置问题可能会导致丢失消息。

如果您发送了一条消息，其服务质量为“至少一次”或者“至多一次”，那么对于您认为已丢失的消息，实际上有可能是未按您期望的方式来传递此消息。 不太可能已经从系统中错误地删除了此消息。 有可能未能创建您期望的发布或预订。

在确定“丢失消息”这一问题时，您执行的最重要的步骤是确认此消息是否确实已丢失。 重现此场景，丢失更多消息。 使用“至少一次”或者“至多一次”服务质量来消除系统废弃消息的所有情况。

关于此任务

可以通过四条途径来诊断“丢失消息”这一问题。

1. “发出消息之后无需等待应答”消息按设计那样工作。 系统有时候会废弃“发出消息之后无需等待应答”消息。
2. 配置：在分布式环境中使用正确的权限来设置发布/预订并不简单。
3. 客户机编程错误：消息传送的责任不仅仅是 IBM 所编写代码的责任。
4. 如果您已经排除了所有这些可能性，就可以确定“丢失消息”这一问题与 IBM 服务有关。

过程

1. 如果丢失的消息的服务质量为“发出消息之后无需等待应答”，那么请设置“至少一次”或者“至多一次”服务质量。 尝试再次丢失此消息。
   - 在许多情况下，IBM WebSphere MQ 会抛弃使用“发出消息之后无需等待应答”服务质量发送的消息：
     - 通信中断，并且通道已停止。
     - 队列管理器已关闭。
     - 消息数过多。
   - 传递“发出消息之后无需等待应答”消息依赖于 TCP/IP 的可靠性。 TCP/IP 将继续反复发送数据包，直到传递的数据包获得确认为止。 如果 TCP/IP 会话已中断，那么服务质量为“发出消息之后无需等待应答”的消息就会丢失。 当客户机或服务器停机、发生通信问题或者防火墙使会话断开连接时，会话就可能会中断。
2. 检查客户机是否正在重新启动先前的会话，以便使用“至少一次”或者“至多一次”服务质量来再次发送未传递的消息。
   a) 如果客户机应用程序正在使用 Java SE MQTT 客户机，请检查它是否将 `MqttClient.CleanSession` 设置为 `false`
   b) 如果您正在使用不同的客户机库，请检查是否正在正确地重新启动会话。
3. 检查客户机应用程序是否正在重新启动同一会话，且未错误启动另一会话。

   要再次启动同一个会话，`cleanSession = false`、`Mqttclient.clientIdentifier` 和 `MqttClient.serverURI` 必须与前一个会话相同。
4. 如果会话过早关闭，请检查消息在客户机的持久库中是否可用以便再次发送。
   a) 如果客户机应用程序正在使用 Java SE MQTT 客户机，请检查是否将消息保存在持久性文件夹中; 请参阅 第 145 页的『客户端日志文件』
   b) 如果您正在使用不同的客户机库，或者您已经实现了自己的持久性机制，请检查它是否在正常工作。
5. 请检查在传递消息之前是否没有人删除此消息。

   正在等待传递到 MQTT 客户机、但是尚未未传递的消息存储在 `SYSTEM.MQTT.TRANSMIT.QUEUE` 中。等待传送到遥测服务器的消息由客户机持久机制存储；请参阅 MQTT 客户机中的消息持久性。
6. 请检查客户机是否具有它期望接收的发布的预订。

使用 WebSphere MQ Explorer 或使用 **runmqsc** 或 PCF 命令列出预订。 已命名全部 MQTT 客户机预订。 将为其提供以下格式的名称: *ClientIdentifier*:*Topic name*

7. 请检查发布程序是否有权限发布以及订户是否有权限预订发布主题。

```
dspmqaut -m qMgr -n topicName -t topic -p user ID
```

在集群发布/预订系统中，必须授权订户预订与此订户相连的队列管理器上的主题。 不需要授权订户预订已发布此发布的队列管理器上的主题。 必须正确地授权队列管理器之间的通道传递代理预订和转发发布。

使用 IBM WebSphere MQ Explorer 创建同一预订并向其进行发布。 使用客户机实用程序来模拟应用程序客户机进行发布和预订。 从 IBM WebSphere MQ Explorer 启动实用程序，并更改其用户标识以便与客户机应用程序所采用的标识匹配。

8. 请检查订户是否具有许可权将发布放在 SYSTEM.MQTT.TRANSMIT.QUEUE 上。

```
dspmqaut -m qMgr -n queueName -t queue -p user ID
```

9. 检查 IBM WebSphere MQ 点到点应用程序是否有权将其消息放在 SYSTEM.MQTT.TRANSMIT.QUEUE 上。

```
dspmqaut -m qMgr -n queueName -t queue -p user ID
```

请参阅 配置分布式排队以将消息发送到 MQTT 客户机中的 “直接将消息发送到客户机” 。

解决问题：Telemetry (MQXR) 服务未启动

解决 Telemetry (MQXR) 服务未能启动的问题。 请检查 WebSphere MQ Telemetry 安装情况，以及是否缺少文件或者文件已移动或其权限有误。 请检查遥测 (MQXR) 服务所用路径是否找到遥测 (MQXR) 服务程序。

开始之前

已安装 WebSphere MQ Telemetry 功能部件。 IBM WebSphere MQ Explorer 在 **IBM WebSphere MQ > 队列管理器> *qMgrName* > Telemetry** 中具有 Telemetry 文件夹。 如果此文件夹不存在，那么表明安装失败。

必须创建 Telemetry (MQXR) 服务才能将其启动。 如果尚未创建遥测 (MQXR) 服务，请运行 **定义样本配置 ...** Telemetry 文件夹中的向导。

如果之前已启动遥测 (MQXR) 服务，那么会在 Telemetry 文件夹下创建其他**通道**和**通道状态**文件夹。 遥测服务 SYSTEM.MQXR.SERVICE 位于**服务**文件夹中。 如果单击了用于显示系统对象的“资源管理器”单选按钮，那么此遥测服务可视。

右键单击 SYSTEM.MQXR.SERVICE 以启动和停止服务，显示其状态，并显示您的用户标识是否有权启动服务。

关于此任务

SYSTEM.MQXR.SERVICE Telemetry (MQXR) 服务未能启动。 启动清单本身时，会出现两种不同形式的故障:

1. 启动命令立即失败。
2. 启动命令成功，但该服务紧接着就停止了。

过程

1. 启动服务

结果

该服务立即停止。 此时会出现一个窗口，其中显示错误消息，例如:

```
WebSphere MQ cannot process the request because the
executable specified cannot be started. (AMQ4160)
```

原因

安装过程中缺少文件，或者错误地设置了对已安装文件的许可权。

仅在其中一对高可用队列管理器上安装 IBM WebSphere MQ Telemetry 功能部件。 如果此队列管理器实例切换到备用实例，那么将尝试启动 SYSTEM.MQXR.SERVICE。 用于启动此服务的命令将失败，因为备用实例上未安装 Telemetry (MQXR) 服务。

调查

查看错误日志；请参阅第 143 页的『服务器端日志』。

操作

安装 WebSphere MQ Telemetry 功能部件，或者将它卸载之后再重新安装。

2. 启动服务；等待 30 秒钟；刷新资源管理器并检查服务状态。

结果

该服务启动，然后又停止。

原因

SYSTEM.MQXR.SERVICE 启动了 **runMQXRService** 命令，但该命令失败。

调查

查看错误日志；请参阅第 143 页的『服务器端日志』。

查看唯一定义的样本通道是否出现问题。 备份并清除 *WMQ data directory*\Qmgrs*qMgrName*\mqxr\ 目录的内容。 运行“样本配置”向导，然后尝试启动该服务。

操作

查找许可权和路径问题。

解决问题：遥测服务未调用 JAAS 登录模块

了解 JAAS 登录模块是否未由遥测 (MQXR) 服务调用，并配置 JAAS 以更正该问题。

开始之前

您已修改 *WMQ installation directory*\mqxr\samples\LoginModule.java 以创建自己的认证类 *WMQ installation directory*\mqxr\samples\samples\LoginModule.class。 或者，您已经编写了自己的 JAAS 认证类并且已将它们放入您选择的目录中。 在利用遥测 (MQXR) 服务进行某些初始测试后，您会认为认证类未由遥测 (MQXR) 服务调用。

注: 防止出现以下可能：认证类由应用于 WebSphere MQ 的维护所覆盖。 使用您自己的认证类路径，而不使用 WebSphere MQ 目录树中的路径。

关于此任务

此任务使用一种方案来说明如何解决此问题。 在此方案中，一个称为 security.jaas 的包中包含一个称为 JAASLogin.class 的 JAAS 认证类。 此认证类存储在 C:\WMQTelemetryApps\security\jaas 路径下。 请参阅遥测通道 JAAS 配置，以获取为 IBM WebSphere MQ Telemetry 配置 JAAS 方面的相关帮助。 第 162 页的『示例 JAAS 配置』这一示例是一个样本配置。

过程

1. 请在 mqxr.log 中查找 javax.security.auth.login.LoginException 抛出的异常。

   请参阅第 143 页的『服务器端日志』以获取 mqxr.log 路径，并参阅第 163 页的图 54 以获取该日志中所列异常的示例。
2. 通过将您的 JAAS 配置与第 162 页的『示例 JAAS 配置』中已使用的示例进行比较，从而更正您的 JAAS 配置。
3. 将您的登录类重构到认证包之后，将它替换为样本 JAASLoginModule，然后使用同一路径部署此登录类。 将 loggedIn 的值在 true 与 false 之间进行切换。

如果当 loggedIn 为 true 时并不存在此问题，而 loggedIn 为 false 时又出现此问题，那么说明登录类存在问题。

4. 请检查此问题是否与授权有关，而不是与认证有关。
   a) 更改遥测通道定义，以使用固定的用户标识执行授权检查。 选择一个是 mqm 组的成员的用户标识。
   b) 重新运行客户机应用程序。

      如果不再存在此问题，那么就要从为授权而传递的用户标识来考虑解决方案。 所传递的用户名是什么？ 从您的登录模块打印至文件。 使用 IBM WebSphere MQ Explorer 或 **dspmqauth** 检查其访问许可权。

示例 JAAS 配置

使用 WebSphere MQ Explorer 中的**新建遥测通道**向导来配置遥测通道。 客户机在端口 1884 上进行连接，并连接至 JAASMCAUser 遥测通道。 第 162 页的图 48 显示遥测向导创建的遥测属性文件的示例。 请勿直接编辑此文件。 通道使用 JAAS 进行认证，并使用称为 JAASConfig 的配置。 一旦客户机进行认证之后，它就会使用用户标识 Admin 来授予其对于 IBM WebSphere MQ 对象的访问权。

```
com.ibm.mq.MQXR.channel/JAASMCAUser: \
com.ibm.mq.MQXR.Port=1884;\
com.ibm.mq.MQXR.JAASConfig=JAASConfig;\
com.ibm.mq.MQXR.UserName=Admin;\
com.ibm.mq.MQXR.StartWithMQXRService=true
```

图 *48: WMQ Installation directory\data\qmgrs\qMgrName\mqxr\mqxr_win.properties*

JAAS 配置文件具有一个名为 JAASConfig 的节，该节将 Java 类命名为 security.jaas.JAASLogin，JAAS 用于认证客户机。

```
JAASConfig {
  security.jaas.JAASLogin required debug=true;
};
```

图 *49: WMQ Installation directory\data\qmgrs\qMgrName\mqxr\jaas.config*

当 SYSTEM.MQTT.SERVICE 启动时，它会将第 162 页的图 50 中的路径添加至其类路径。

```
CLASSPATH=C:\WMQTelemtryApps;
```

图 *50: WMQ Installation directory\data\qmgrs\qMgrName\service.env*

第 162 页的图 51 显示了第 162 页的图 50 中添加到为遥测 (MQXR) 服务设置的类路径的其他路径。

```
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:\WMQTelemtryApps;
```

图 *51: runMQXRService.bat* 的类路径输出

第 163 页的图 52 中的输出显示遥测 (MQXR) 服务已使用第 162 页的图 48 中所示的通道定义启动。

```
21/05/2010 15:32:12 [main] com.ibm.mq.MQXRService.MQXRPropertiesFile
AMQXR2011I: Property com.ibm.mq.MQXR.channel/JAASMCAUser 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
```

图 *52: WMQ Installation directory\data\qmgrs\qMgrName\errors\mqxr.log*

当客户机应用程序连接到 JAAS 通道时，如果 `com.ibm.mq.MQXR.JAASConfig=JAASWrongConfig` 与 `jaas.config` 文件中的 JAAS 节名称不匹配，那么此连接将失败，并且客户机会抛出异常（返回码为 `0`）；请参阅第 163 页的图 53。 由于客户机尝试在未连接时断开连接，因此抛出了第二个异常 `Client is not connected (32104)`。

```
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.ja
va: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)
```

图 *53:* 连接 *com.ibm.mq.id.PubAsyncRestartable* 时抛出异常

`mqxr.log` 包含第 163 页的图 53 中所显示的附加输出。

JAAS 检测到该错误，它将抛出 `javax.security.auth.login.LoginException`（原因为 `No LoginModules configured for JAAS`）。 如第 163 页的图 54 中所示，这可能由于配置名称错误所致。 它也可能是在装入 JAAS 配置时 JAAS 遇到的其他问题导致的。

如果 JAAS 没有报告异常，那么 JAAS 已成功地装入在 `JAASConfig` 一节中命名的 `security.jaas.JAASLogin` 类。

```
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
```

图 *54: mqxr.log* - 装入 *JAAS* 配置时出错

解决问题：启动或运行守护程序

请参阅设备的 IBM WebSphere MQ Telemetry 守护程序控制台日志，开启跟踪，或使用本主题中的症状表对守护程序方面的问题进行故障诊断。

过程

1. 检查控制台日志。

   如果守护程序在前台运行，那么控制台消息将写入终端窗口中。 如果已在后台启动守护程序，那么表明您已将 stdout 重定向至控制台。
2. 请重新启动守护程序。

   在重新启动守护程序之前，未激活对配置文件所作的更改。
3. 请参阅第 164 页的表 7：

表 7: 症状表

问题	建议的解决方案
在 Windows 上，当您启动守护程序时将显示以下消息： **系统无法执行所指定的程序** 或 **应用程序未能启动** **因为其并排配置不正确。**	请安装 Microsoft Visual C++ 2008 Redistributable Package。
两个或更多守护程序或支持 MQTT 的服务器通过一个或多个网桥进行互连，同时处理器显示超载。	可能存在消息循环，会反复将一条或多条消息从一个服务器传递至另一个服务器。 请检查配置文件中的 topic 参数。 请尽可能使用更具体的主题。 在两个方向上都广泛使用通配符是造成连接循环的最常见原因。
网桥无法连接至支持 MQTT 的远程服务器，而其他 MQTT 客户机可以连接至该远程服务器。	远程服务器可能与确定该服务器是否也是设备的 WebSphere MQ Telemetry 守护程序的尝试操作不兼容。 请尝试将 **try_private** 设置为 off，以禁止进行特殊处理以消除消息循环。
配置网桥时显示了以下消息： **警告：连接的不是套接字 1888 上的第一个包，获得的是 CONNACK。**	您可能配置了一个网桥用于将消息回送到本地守护程序。 不支持回送。

解决问题：MQTT 客户机未连接到守护程序

客户机未连接到守护程序，或者守护程序未连接到其他守护程序或 WebSphere MQ 遥测通道。

关于此任务

跟踪守护程序发送和接收的每个 MQTT 包。

过程

在守护程序配置文件中将 **trace_output** 参数设置为 protocol，或使用 amqtdd.upd 文件将命令发送至守护程序。

请参阅在设备的 IBM WebSphere MQ Telemetry 守护程序与 IBM WebSphere MQ 之间传输消息，以获取使用 amqtdd.upd 文件的示例。

使用协议设置，守护程序可将描述其发送和接收的每个 MQTT 包的消息打印至控制台。

声明

本信息是为在美国提供的产品和服务编写的。

IBM 可能在其他国家或地区不提供本文档中讨论的产品、服务或功能。 有关您当前所在区域的产品和服务的信息，请向您当地的 IBM 代表咨询。 任何对 IBM 产品、程序或服务的引用并非意在明示或默示只能使用 IBM 的产品、程序或服务。 只要不侵犯 IBM 的知识产权，任何同等功能的产品、程序或服务都可以代替 IBM 产品、程序或服务。 但是， 评估和验证任何非 IBM 产品、程序或服务的操作，由用户自行负责。

IBM 公司可能已拥有或正在申请与本文档内容有关的各项专利。 提供本文档并未授予用户使用这些专利的任何许可。 您可以以书面形式将许可查询寄往：

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

有关双字节（DBCS）信息的许可查询，请与您所在国家或地区的 IBM 知识产权部门联系，或用书面方式将查询寄往：

知识产权许可
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 063-8506 Japan

本条款不适用英国或任何这样的条款与当地法律不一致的国家或地区: International Business Machines Corporation “按现状” 提供本出版物，不附有任何种类的 (无论是明示的还是暗含的) 保证，包括但不限于暗含的有关非侵权，适销和适用于某种特定用途的保证。 某些国家或地区在某些交易中不允许免除明示或暗含的保证。 因此本条款可能不适用于您。

本信息中可能包含技术方面不够准确的地方或印刷错误。 此处的信息将定期更改；这些更改将编入本资料的新版本中。 IBM 可以随时对本出版物中描述的产品和/或程序进行改进和/或更改，而不另行通知。

本信息中对非 IBM Web 站点的任何引用都只是为了方便起见才提供的，不以任何方式 充当对那些 Web 站点的保证。 那些 Web 站点中的资料不是 IBM 产品资料的一部分，使用那些 Web 站点带来的风险将由您自行承担。

IBM 可以按它认为适当的任何方式使用或分发您所提供的任何信息而无须对您承担任何责任。

本程序的被许可方如果要了解有关程序的信息以达到如下目的：(i) 允许在独立创建的程序和其他程序（包括本程序）之间进行信息交换，以及 (ii) 允许对已经交换的信息进行相互使用，请与下列地址联系：

IBM Corporation
软件互操作性协调员，部门 49XA
北纬 3605 号公路
罗切斯特，明尼苏达州 55901
U.S.A.

只要遵守适当的条件和条款，包括某些情形下的一定数量的付费，都可获得这方面的信息。

本资料中描述的许可程序及其所有可用的许可资料均由 IBM 依据 IBM 客户协议、IBM 国际软件许可协议或任何同等协议中的条款提供。

此处包含的任何性能数据都是在受控环境中测得的。 因此，在其他操作环境中获得的数据可能会有明显的不同。 有些测量可能是在开发级的系统上进行的，因此不保证与一般可用系统上进行的测量结果相同。 此外，有些测量是通过推算而估计的， 实际结果可能会有差异。 本文档的用户应当验证其特定环境的适用数据。

涉及非 IBM 产品的信息可从这些产品的供应商、其出版说明或其他可公开获得的资料中 获取。 IBM 没有对这些产品进行测试，也无法确认其性能的精确性、兼容性或任何其他关于非 IBM 产品的声明。 有关非 IBM 产品性能的问题应当向这些产品的供应商提出。

所有关于 IBM 未来方向或意向的声明都可随时更改或收回，而不另行通知，它们仅仅表示了目标和意愿而已。

本信息包含日常商业运作所使用的数据和报表的示例。 为了尽可能全面地说明这些数据和报表，这些示例包括个人、公司、品牌和产品的名称。 所有这些名称都是虚构的，如与实际商业企业所使用的名称和地址有任何雷同，纯属巧合。

版权许可：

本信息包含源语言形式的样本应用程序，用以阐明在不同操作平台上的编程技术。 如果是为按照在编写样本程序的操作平台上的应用程序编程接口（API）进行应用程序的开发、使用、经销或分发为目的，您可以任何形式对这些样本程序进行复制、修改、分发，而无须向 IBM 付费。 这些示例并未在所有条件下作全面测试。 因此，IBM 不能担保或默示这些程序的可靠性、可维护性或功能。

如果您正在查看本信息的软拷贝，图片和彩色图例可能无法显示。

编程接口信息

编程接口信息 (如果提供) 旨在帮助您创建用于此程序的应用软件。

本书包含有关允许客户编写程序以获取 IBM WebSphere MQ 服务的预期编程接口的信息。

但是，该信息还可能包含诊断、修改和调优信息。 提供诊断、修改和调优信息是为了帮助您调试您的应用程序软件。

要点: 请勿将此诊断，修改和调整信息用作编程接口，因为它可能会发生更改。

商标

IBM IBM 徽标 ibm.com 是 IBM Corporation 在全球许多管辖区域的商标。 当前的 IBM 商标列表可从 Web 上的 “Copyright and trademark information”www.ibm.com/legal/copytrade.shtml 获取。 其他产品和服务名称可能是 IBM 或其他公司的商标。

Microsoft 和 Windows 是 Microsoft Corporation 在美国和/或其他国家或地区的商标。

UNIX 是 Open Group 在美国和其他国家或地区的注册商标。

Linux 是 Linus Torvalds 在美国和/或其他国家或地区的商标。

此产品包含由 Eclipse 项目 (http://www.eclipse.org/) 开发的软件。

Java 和所有基于 Java 的商标和徽标是 Oracle 和/或其附属公司的商标或注册商标。

部件号:

(1P) P/N: