设计适用于 AWS IoT Core 的 MQTT 主题

2019 年 *5* 月

声明

客户有责任对本文档中的信息进行独立评估。本文档：(a) 仅供参考，(b) 代表 AWS 当前的产品服务和实践，如有变更，恕不另行通知，以及 (c) 不构成 AWS 及其附属公司、供应商或授权商的任何承诺或保证。AWS 产品或服务均“按原样”提供，没有任何明示或暗示的担保、声明或条件。AWS 对其客户的责任和义务由 AWS 协议决定，本文档与 AWS 和客户之间签订的任何协议无关，亦不影响任何此类协议。

目录

摘要

本白皮书重点介绍在 Amazon Web Services (AWS) 物联网 (IoT) 中设计 MQTT 主题的最佳实践。内容包括开发最佳 MQTT 主题架构以及能够从以下几个方面改进 IoT 应用程序的整体架构：更好地了解云到设备通信、提供更多细粒度安全权限，以及增强与其他 AWS IoT Core 服务（例如 AWS IoT Rules Engine、AWS IoT Device Shadow、AWS IoT Device Management 和 AWS IoT Analytics）的集成选项。本白皮书面向技术架构师、IoT 云计算工程师和嵌入式工程师。本白皮书假定读者了解基本 MQTT 概念和术语。

简介

AWS IoT Core 支持 MQTT，后者是一种广泛采用的轻量级消息收发协议，专为受限设备而设计。MQTT 参与者通过 MQTT 主题接受组织的消息。MQTT 主题充当发布者和订阅者之间的匹配机制。从概念上来讲，MQTT 主题类似于临时通知频道。

对于 AWS IoT Core，使用 MQTT 时，首要注意的事项之一是，MQTT 主题的设计策略。MQTT 主题必须在当前设备通信、云端操作和未来的设备功能之间实现平衡。因此，设计具备以下特征的理想 MQTT 主题结构具有挑战性：创建的架构足以执行最小特权通信，但这个结构又不会太严谨以致于难以支持未来的设备部署。

本文档提供了 MQTT 主题设计最佳实践和指导。它概述了一系列可用于解决各种设备消息模式的常用 MQTT 主题结构，然后使用不同 AWS IoT Core 服务应用了几个示例设计模式。

MQTT 通信模式

IoT 应用程序支持多种通信场景，例如设备到设备、设备到云、云到设备、设备到用户和用户到设备。虽然各种模式的范围大不相同，但大多数 MQTT 通信模型源自三种 MQTT 模式：点对点、广播和扇入。

点对点

点对点通信模式是基本构建块之一，与设备在 MQTT 中通常如何发送和接收消息有关。两个事物使用一个 MQTT 主题作为通信频道。接收活动的事物订阅 MQTT 主题。发送消息的事物发布到同一已知 MQTT 主题。此方法常用于最终用户接收有关家中事物更新的智能家居场景。在下面的示例中，机械手臂将消息发布到移动应用程序订阅的主题。

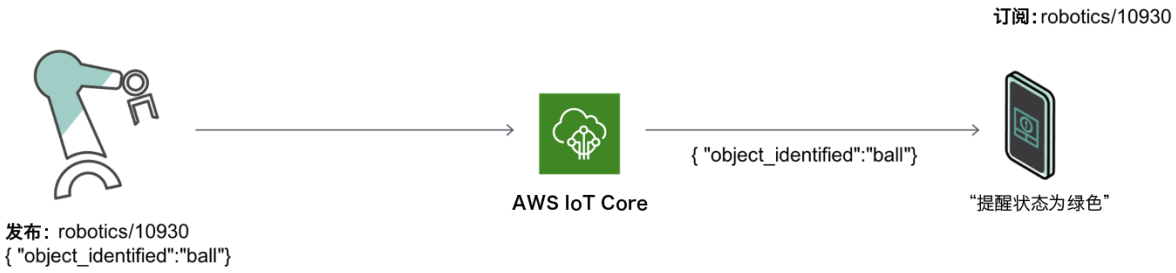


图 *1*：点对点通信中的一对一消息收发

点对点通信不限于设备间的一对一通信。点对点也可用于一对多通信，即一位发布者可以按设备将不同的 MQTT 主题发布到单个设备。此方法常用于管理员将不同的更新发送到特定设备的通知场景。在下面的示例中，修复服务使用一系列点对点通信以编程方式遍历一系列设备并发布消息。

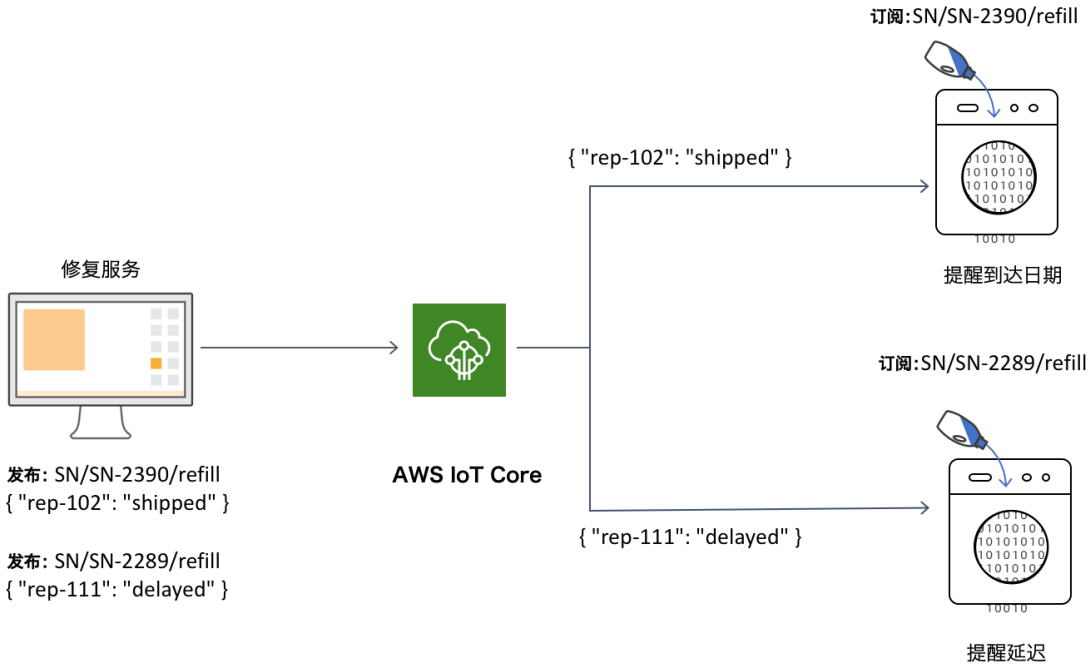


图 *2*：点对点通信中的一对多消息收发

广播

广播模式用于一对多消息收发。广播模式将同一消息发送到多个设备。在广播模式中，多个设备订阅同一 MQTT 主题，发送方将消息发布到这个主题。广播模式的典型使用场景是，根据设备类别或组向设备发送通知。例如，气象站根据设备的当前地理位置（设备按地理位置分组）传送广播消息。

下图描述了一个示例，即广播模式将消息发送到州内所有巴士均订阅的气象主题。根据当前位置，巴士可以忽略或响应消息。

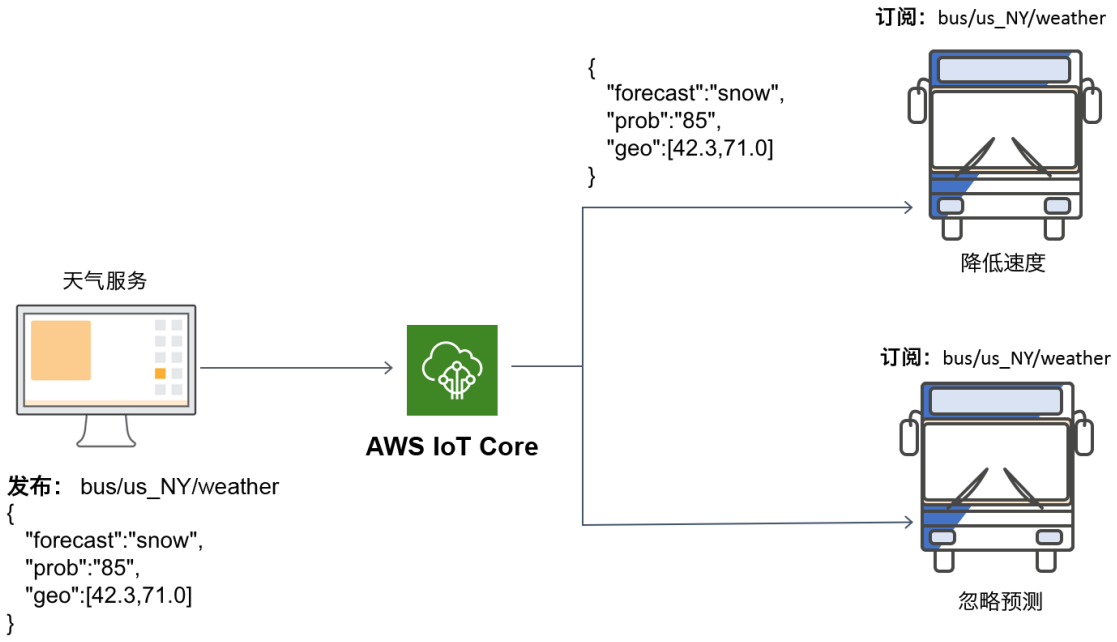


图 *3*：广播通信中的一对多消息收发

扇入

扇入模式是多对一通信模式，与广播模式相反。多个设备发布到只有一个订阅者的同一主题。借助扇入模式，订阅者还能够使用通配符，因为发布者均使用一个类似但唯一的 MQTT 主题。扇入模式通常用于帮助将遥测技术整合到 IoT 应用程序。

在下面的示例中，每个终端设备发布到一个含已知组标识符的 MQTT 主题。AWS IoT Rules Engine 使用通配符订阅接收消息并将它们路由到 Amazon Kinesis 流。具体来说，机械手臂发布到与特定构建（构建 1234）相关联的扇入主题。管理系统使用 MQTT 通配符 (+) 接收构建的所有更新。

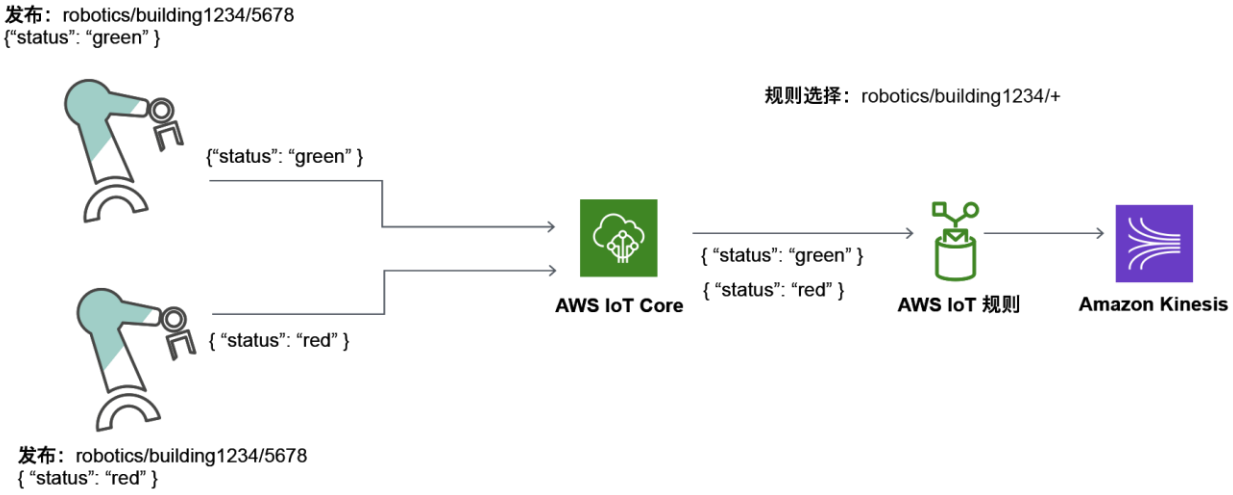


图 4：扇入模式中的多对一通信

设备使用 MQTT 通过云通信时，避免使用扇入模式路由到单个订阅终端设备，因为这种路由可能会达

到单个设备 MQTT 连接的限制（不可调整）。相对应的，使用扇入模式可以通过 AWS IoT Rules Engine 将大量消息路由到 IoT 应用程序。对于大规模扇入场景，结合使用 Rules Engine 和通配符订阅模式以及 Rules Engine 操作，路由到 Amazon Kinesis Data Streams、Amazon Kinesis Data Firehose 或 Amazon Simple Queue Service (Amazon SQS)。

通信工作流程

三个常见通信工作流程是设备到设备、设备到云和云到设备。每个工作流程决定 MQTT 主题的顺序。对于设备到设备，MQTT 主题应包含消息发送方或接收方的标识符。对于设备到云，MQTT 消息应包含有关目标应用程序的信息。目标应用程序负责使用有关设备的内部元数据增强任何 MQTT 消息。最后，对于云到设备通信，MQTT 消息应包含会话信息，用于跟踪任何关键消息的确认。

MQTT 设计最佳实践

通用最佳实践

虽然有许多常见共享的物联网通信模式组合方式，但有一些最佳实践适用于任何消息模式，不受设备发布或接收消息方式的影响。本节介绍了一些整体最佳实践，供您在设计主题结构时查看和应用。

确保 MQTT 主题级别仅使用小写字母、数字和破折号。由于 MQTT 主题区分大小写，设计 MQTT 主题时，一定要使用一组标准的命名约定。因此，客户在创建每个主题级别时，应仅使用小写字母、数字和破折号。客户应避免使用驼峰式大小写和空格等难以调试的字符。

确保 MQTT 主题级别结构遵循从一般到具体的模式。主题架构从左到右，主题级别从一般到具体。例如，HVAC 系统与名为 **hv100** 的 IoT 平台相关联，位于构建 **bld1518** 的**底层**，事物名称为 **hvac719**。主题结构的开头为一般组，在本例中，即 IoT 平台的名称，结尾为最具体的身份，即事物名称。本示例创建以下主题级别结构：

```
hv100/bld1518/basement/hvac719
```

在 MQTT 主题中包含所有相关路由信息。相关路由信息包括但不限于：IoT 应用程序标识符、设备可能属于的任何组（例如安装位置）和 IoT 设备的唯一身份。 延续前面的 HVAC 系统示例，MQTT 主题 hv100/bld1518/basement/hvac719 包含所有相关路由信息。根据此 MQTT 主题，您可以设计具有以下特征的系统：捕获与标识符为 hv100 的整个应用程序相关的所有数据，还可以针对感兴趣的领域（例如构建位置）来订阅消息。

为 MQTT 主题添加前缀，以区分数据主题与命令主题。确保 MQTT 主题在命令和数据消息之间不存在重叠。通过预留第一个主题级别来表示数据和命令主题，您可以更轻松地使用 IoT 策略创建细粒度权限，分开监控命令和命令响应的状态与被动遥测命令。例如，使用 AWS IoT Device Shadow 跟踪报告状态和所需状态，使用单独的数据主题跟踪被动实时遥测数据。

记录 MQTT 主题结构作为运营实践的一部分。记录内容应包括适用于发布、订阅或接收数据的所有主题，以及既定数据产生方和使用方。审查记录内容，确保它遵守所有 AWS IoT 限制、内部安全要求和所有应用程序使用案例。

使用 AWS IoT 事物名称作为 MQTT 客户端 ID，以便作为设备进行 MQTT 连接。连接到 AWS IoT 时，MQTT 要求提供唯一客户端 ID。使用每个设备的唯一事物名称作为 MQTT 客户端 ID。通过将客户端 ID 与事物名称匹配，您可以更轻松地将 IoT 日志等信息关联到相关 AWS IoT 设备。此外，如果客户端 ID 与事物名称匹配，您可以使用其他功能，例如事物策略变量。事物策略变量支持 AWS IoT 策略包含事物名称和事物特征等事物属性，还能帮助减少在所有 IoT 设备上预置所需的策略总数。

在设备用于发布或订阅数据的任何 MQTT 主题中包含设备的事物名称。要跟踪发送到特定设备的消息，请将事物名称包含在由设备发布或发送到特定设备的任何 MQTT 消息中。事物名称应位于 MQTT 主题附近或末尾，任何路由主题信息之后。

查看 AWS IoT Core 默认服务限制。设计通信模式，使其与任何可调整 IoT 服务限制保持一致。AWS IoT 具有一些与使用 AWS IoT 核心服务相关的可调整和不可调整限制。审查主题时，需要查看 AWS IoT 的限制，确保 MQTT 主题和设备通信不与任何不可调整的限制冲突。

在 MQTT 消息的负载中包含有关特定消息的额外上下文信息。上下文信息包括但不限于：会话标识符、请求者标识符、日志记录消息和设备期望从中收到响应的返回主题。虽然 MQTT 3.1.1 规范未要求具体负载属性，但我们建议在 MQTT 负载中包含相关跟踪信息。通过创建包含字段（例如会话标识符和成功或错误代码）的标准结构，您可以更轻松地分析设备行为的趋势。通信架构标准还能增进跨 IoT 团队间使用共享的设备用例术语。

避免导致对单个设备产生大规模扇入场景的 MQTT 通信模式。有些 AWS IoT 限制无法提高，也无法经常关联到每设备操作，例如单个 MQTT 连接的最大发布量。请勿允许单个设备订阅大量其他设备发布到的同一主题。通过避免采用此模式，您更容易避免达到单个连接设备限制，尤其是每秒每连接数限制。

切勿允许设备订阅所有使用 # 的主题，并且仅在 IoT 规则中使用多级别通配符订阅。通过使用多级别通配符，当您无意间将新主题添加到特定设备不适用的层级时，可能会产生意外后果。相反，预留多级别通配符作为 IoT 规则引擎的一部分，并针对设备订阅使用单级别通配符 (+)。

遥测最佳实践

遥测是只读数据，由设备传输并在云中聚合。遥测遵循设备到云模式和通信的扇入模式。遥测不要求从 MQTT 代理返回确认消息，但可以选择性设置更高的服务质量 (QoS) 级别。遥测是被动活动，所以遥测的 MQTT 主题不能与命令等任何主动工作流程的 MQTT 主题重叠。遥测主题支持代表其他设备发布遥测的更复杂的设备，例如边缘网关或带有一个协调器的网状网络。

在 AWS IoT 中，您可以使用不同的 AWS IoT 服务支持遥测通信模式。我们建议您结合使用 AWS IoT Basic Ingest 和标准 MQTT 主题，为遥测使用案例提供支持。

针对遥测使用 AWS IoT Basic Ingest

Basic Ingest 可以将发布/订阅消息代理从提取路径中删除，进而优化大量数据提取工作负载的数据流。这样一来，您能够以更经济实惠的方式将设备数据发送到其他 AWS 服务，同时继续受益于 AWS IoT Core 的所有安全性和数据处理功能。如果设备不需要消息代理的发布和订阅功能，Basic Ingest 使您能够通过 Rules Engine 仅将数据发送到云服务。

如果唯一对 IoT 消息感兴趣的订阅者是您的后端 IoT 应用程序，则 Basic Ingest 是理想使用案例。Basic Ingest 使用与特定 AWS IoT 规则相关联的预留 MQTT 主题结构。设备可以发布到与特定 AWS IoT 规则相关联的预留主题，而 Basic Ingest 则会针对匹配的规则名称触发任何相关联的 IoT 规则操作。Basic Ingest 的 MQTT 主题结构遵循与下面类似的语法：

```
$aws/rules/<rule-name>/<optional-customer-defined-segments>
```

其中，rule-name 字段与应触发的 AWS IoT 规则的名称匹配，optional-customer-defined-segments 包括客户可能用于路由或登录等 IoT 规则操作的任何其他主题级别。

使用 AWS IoT Basic Ingest 的最佳实践

包含位于 Basic Ingest MQTT 主题中规则名称之后的任何其他路由信息作为最佳实践，AWS 建议您使用可以位于 MQTT 主题中规则名称之后的可选字段，以包含其他相关信息，这些信息可由 AWS IoT 规则用于多种功能，例如替代模板、IoT 规则 SQL 函数和 Where 子句。与 MQTT 主题的整体最佳实践类似，任何可用于路由或 IoT 规则评估的字段（例如应用程序标识符或设备标识符）均应附加到 Basic Ingest 主题的结尾处。

为 Basic Ingest 选择简短的描述性规则名称当 AWS IoT 规则由设备通过 Basic Ingest 直接使用时，AWS 建议您确保规则名称遵循 MQTT 主题最佳实践，以保持一致。规则会直接链接到预留 MQTT 主题，因此请确保规则名称简短描述规则的底层使用案例，且仅含小写字母和破折号，因为 MQTT 区分大小写。

针对遥测使用 MQTT 主题

除了使用 Basic Ingest 之外，您还可以使用传统 MQTT 主题。这些类型的 MQTT 消息是其他设备现在或未来可能订阅的被动 IoT 数据。例如，发送当前状态的设备可能希望其数据不仅路由到内部应用程序，还希望路由到需要设备当前状态的用户。要实现这种程度的灵活性，您可以使用标准 MQTT 主题发送和接收遥测。

MQTT 遥测主题语法

下面的示例和章节描述了遥测的 MQTT 主题结构。

```
dt/<application>/<context>/<thing-name>/<dt-type>
```

dt：设置表示消息类型的前缀。

对于遥测主题，我们使用 dt 表示数据。所有遥测主题针对应用程序使用此顶级前缀。通过重复使用相同的遥测值，您可以参考初始前缀值，从而确定消息的意图。在本例中，所有 dt 主题均是遥测主题。

application：识别与设备关联的整体 IoT 应用程序。

常用应用程序属性包括设备硬件版本或云应用程序的内部标识符（充当消息的主提取点）。IoT 应用程序与整体 IoT 产品的内部名称相关联，或者与设备的硬件类型具体相关联。应用程序主题部分与一组设备消息相关联，并且不可改变，所以 MQTT 遥测主题的应用程序前缀部分位于 dt 消息类型之后。

context：关于设备正在发布的消息的单个或多个级别的附加上下文数据。

上下文信息与设备预置期间设置的信息相关。例如，出厂设置中的上下文信息可能包括设备在设施中的当前物理位置。上下文信息的另一个示例是 MQTT 主题中的 group-id。group-id 表示多个设备具有基于特定属性的内在关系，例如购买一套智能灯泡来控制室内照明。group-id 使大量设备能够作为一个单元协调活动。

thing-name：识别正在发送遥测消息的设备。

dt-type（可选）：将消息与设备的特定子组件相关联，或者用于任何下游设备的边缘网关。

复杂的设备通常具有多个处理特定任务的子组件，例如传感器、执行器或独立系统级芯片 (SOC)。dt-type 使您能够将特定设备的每个子组件关联到单个 MQTT 主题。例如，检测车辆地理位置和方向的子组件。该子组件的 dt-type 值为 geo，可以将其地理位置消息与汽车的其他组件（例如加速计）的区分开。

命令最佳实践

在 IoT 应用程序中，命令主题用于远程控制设备和确认命令成功执行。与遥测不同，命令主题不是只读的。命令是往返工作流程，可以发生在两个设备之间，也可以发生在云和设备之间。命令是可执行的消息，所以请将命令消息的 MQTT 主题与遥测主题隔离。

您可以使用一些服务在 AWS IoT 上执行命令和控制操作。AWS IoT Shadow 能够在云中存储所需状态和报告状态，是执行单个设备命令的首选 AWS IoT 服务。AWS IoT Device Jobs 应用于队列范围的操作，因为它可以提供额外的好处，例如用于跟踪作业的 Amazon CloudWatch 指标，以及能够针对单个设备跟踪多个传输中作业。您可以结合使用 AWS IoT Shadow、AWS IoT Job 文档和标准 MQTT 主题，为命令使用案例提供支持。

针对命令使用 AWS IoT Shadow

Device Shadow 服务充当状态媒介，允许设备和应用程序检索和更新设备的影子状态。您可以使用影子通过 MQTT 或 HTTP 获取和设置设备的状态。影子包括以下支持命令和控制的各个状态属性：

所需状态。具备向设备发送命令权限的应用程序可以将请求的状态更改写入到影子文档的所需部分。通过更新所需状态，AWS IoT Shadow 服务将所需状态更改存储在 AWS 云中，然后使用预留的影子主题向设备发送 MQTT 消息。当设备接收到影子请求后，它便会从所需状态执行所需更改。

报告状态。事物的报告状态存储设备最近发布的当前属性。事物写入文档的这一部分来记录它们的新状态，而应用程序读取影子的这一部分来确定特定设备的状态。对于执行命令主题，建议使用 AWS

IoT Shadow 存储、检索和更改设备属性。
如果命令留待日后使用，则可以使用 AWS IoT Shadow，即使设备当前处于离线状态。例如，如果系统通过影子的所需状态向 GPS 系统发送了新的目的地，但是不能立即到达这个目的地，则新的坐标将保留在 GPS IoT Shadow 中。一旦 GPS 系统恢复连接性，它便会主动请求最新的影子状态并检索新的坐标。影子还适用于存储设备属性的最新报告状态。

使用 AWS IoT Shadow 的最佳实践

AWS IoT Shadow 是一种出色的命令和控制机制，还可以存储特定设备的报告状态。下面的列表提供了通过影子最大限度提高命令效率的最佳实践。
IoT 设备不应共享影子。要隔离每个设备的命令，请确保每个设备都具备对自己影子的权限，并且设备不共享一个影子。对于复杂的场景，比如边缘网关或具有多个子组件的大型设备资产，主资产应使用多个与下游设备单独关联的 IoT 事物和影子。
对于每秒事务处理量 (TPS) 处于中低水平的状态或命令，请使用影子。影子适用于在数分钟、数小时或数天内发生的不频繁更新。影子发布其他主题来确认操作是否成功，所以请使用标准 MQTT 命令主题来实现高吞吐量遥测，例如每秒发送多次更新的请求。
使用影子存储设备的状态指标。存储有关设备当前运行状况的信息数据，包括但不限于：连接性、设备传感器和控制单元的状态以及关于这些子组件的任何错误信息。如果您知道设备的当前状态，可以在命令请求期间做出可执行的决策。
使用设备影子为当前固件版本编写目录。影子是设备报告硬件上已安装固件版本的理想选择。固件应当是一个简单属性，例如突出显示服务 major.minor.patch 版本的字段。
使用具有设备影子更新的可选 clientToken 字段跟踪影子消息的发送方。clientToken 是影子中的一个字段，使订阅者能够将响应与 MQTT 应用程序中的请求关联。如果设备在影子更新请求期间设置 clientToken，则 AWS IoT Shadow 服务将在关联的影子输出事件中包含相同的 clientToken。

针对命令使用 AWS IoT Jobs

AWS IoT Jobs 是一项服务，可让您定义一组远程操作，系统会将这些操作发送到与 AWS IoT 连接的一个或多个事物并在其上执行这些操作。对于命令使用案例，Jobs 允许应用程序运行多步骤任务。AWS IoT Job 包含事物必须运行才能完成其事务的指令。对于队列范围操作任务（例如软件更新），建议使用 AWS IoT Jobs，这些任务只能由整个 IoT 应用程序的可信管理员执行。

使用 AWS IoT Jobs 的最佳实践

使用事物组为 AWS IoT Jobs 组织设备。创建按共同设备属性组织的多个事物组，例如当前固件版本、硬件版本或部署环境（例如，暂存或生产）。事物组还应具备共同的层次结构，例如业务单元或位置。在部署期间，您可以使用事物组作为特定 IoT Job 的部署目标。
使用 Device Jobs 通过已暂存发布部署命令。Device Jobs 是向设备交付队列范围操作的理想解决方案。首先为队列的子集创建多个小型部署，让设备应用更改，然后逐渐将命令扩展到更多设备。通过支持在数

周或数月内更改进度，您可以更加放心，突发问题将会减少，并且如果在发布早期出现问题，您可以更快地做出反应。

使用 MQTT 命令主题

MQTT 命令主题语法

在一些场景中，您需要使用标准 MQTT 发布和订阅模型设计命令通信。如果设备必须执行临时命令（即，只能在当前类型下处理，而且如果设备不可用，则会失败）或同时在多个设备上运行一个命令，则可能会发生这种情况。在棕地环境中，设备可能无法使用更高级别的 AWS IoT 服务，这时也可能会发生这种情况。您还可能需要灵活地选择自己的一组 MQTT 主题，以定义发送给设备的命令和来自设备的响应。

如果您使用的是一组单独的命令主题，请遵循与遥测部分描述类似的 MQTT 命令主题最佳实践。命令主题应具备适用于发布或向其他设备传递命令的复杂设备的灵活性，还应支持查看基本属性。MQTT 命令主题的设计应能够回答与 MQTT 主题和负载相关的操作问题：

- 谁是命令的发起者？
- 谁是命令的目标接收方？
- 命令是否已成功处理？
- 命令的当前状态是什么？
- 如果命令未成功处理，是什么地方出错了？

除了这些问题之外，您可能还需要确定命令的请求时间、设备的响应时间，以及监控云中队列内任何一个请求的状态。

当您为命令请求设计 MQTT 主题时，请遵循以下结构：

```
cmd/<application>/<context>/<destination-id>/<req-type>
```

命令是双向通信模式，所以请设计一个类似的 MQTT 主题结构来响应命令，如下所示：

```
cmd/<application>/<context>/<destination-id>/<res-type>
```

遥测主题设计与命令主题设计类似，所以本节仅提供不相同的命令请求和响应的 IoT 主题部分。

cmd：表示消息类型的前缀。

命令主题使用 cmd 表示命令。通过为所有命令加上前缀 cmd 和为所有遥测加上前缀 dt，系统会将 MQTT 遥测主题和命令主题隔离。

req-type：对命令进行分类。

对于简单的请求和响应模式，req-type 属性应为一个命令请求的静态值，例如 req。如果命令类型有限，则 MQTT 消息将在负载中包含附加数据。

在更复杂的系统中，设备会编排多个设备、执行器或子组件，req-type 属性与每个可用来接收命令的子组件相关联。例如，如果设备是移动设备，您可能需要远程控制该设备或接收有关该设备周围环境的导航信息。这种子组件的 req-type 为 nav，系统将发送命令控制一架或多架飞机。

destination-id：识别此消息的目标设备或应用程序。
通过包含 destination-id，目标设备可以订阅自己的一组命令主题，并接收所有命令请求。
res-type：表示命令响应，识别与先前发送的命令相关的响应。
res-type 支持单个设备对所有传入的命令确认消息使用一个单级别通配符订阅。如果设备的命令有限，则响应主题可以使用静态字段，例如 res。

MQTT 命令负载语法

除了为命令创建一个清晰的 MQTT 主题结构之外，还要确保为消息负载生成一个架构。接收设备或 IoT 应用程序解析 MQTT 负载信息，以了解完成操作可能需要的任何附加逻辑。对于 MQTT 命令，在命令消息负载中包含以下字段：
session-id：识别唯一的会话。
请求者为命令生成 session-id，并将其包含在请求负载中。响应主题在命令完成后使用 session-id。通过使用 session-id，AWS IoT Rules Engine 可以存储和跟踪命令的状态，并确定请求是仍在传输、成功还是出现了错误。与多个设备通信时，设备还可以跟踪传输中请求。
response-topic：命令中包含一个要发生的操作请求和一个表明命令状态（成功或出现错误）的响应。为了避免硬编码响应主题，我们建议，对于任何 MQTT 命令，命令请求负载包含一个具有响应主题的字段。设备使用响应主题发布响应负载。例如，请考虑以下命令主题：

```
cmd/security/device-1/cert-rotation
```

在此请求的负载中，IoT 应用程序包含一个表明设备 (device-1) 应将响应发送到何处的字段和一个用于跟踪的会话标识符。有关此命令的负载结构，请参阅以下示例：

```
{
   "session-id":"session-820923084792",
   "res-topic":"cmd/security/app1/res"
}
```

AWS 上的应用程序

下面几节提供了使用 AWS IoT 实施 MQTT 主题最佳实践的使用案例。

AWS IoT Shadow 示例

在本例中，管理员向风力涡轮机发送一个命令，以更改涡轮机叶片的当前角度，从而适应即将到来的风向变化。为此，管理员首先使用 AWS IoT Shadow 作为向设备发出命令的主机制，并将其作为与设备相关的重要设备读取数据的存储位置。
此场景假设以下详细信息：

- 风力涡轮机的名称为 turbine8000
- 管理员使用有权通过仅限于 turbine8000 的临时 IAM 权限使用的内部应用程序。

- 管理员的会话标识符为 session-admin100

向风力涡轮机设备影子发送的命令请求

管理员使用为 turbine8000 预留的 MQTT Shadow 主题向 IoT Shadow 发布消息。命令负载包括设置影子的所需状态。出于跟踪目的，管理员还将会话标识符以 clientToken 的形式嵌入到 IoT Shadow 请求中。

发布：$aws/things/turbine8000/shadow/update
负载：

```
{
   "state": {
              "desired": { "bladeAngle":3 }
    },
   "clientToken": "admin1234"
}
```

图 *5*：向风力涡轮机设备影子发送的命令请求

风力涡轮机上的命令处理

使用影子时，终端设备负责订阅一些不同的 MQTT 主题，以接收来自 AWS IoT Shadow 服务的相应通知。至少，确保设备订阅 update/delta、update/rejected、get/accepted 和 get/rejected 预留影子主题，以接收成功和失败的影子请求。

增量负载：

```
{
   "version": 2,
   "timestamp": 1536755492,
   "state": { "bladeAngle": "3" },
   "metadata": { "welcome": { "timestamp": 1536755492 } },
   "clientToken": "admin1234"
}
```

图 *6*：风力涡轮机上的命令处理

在本例中，风力涡轮机收到一条关于 update/delta 的消息，这条消息表明报告状态与所需状态不同。处理命令之前，风力涡轮机检查 clientToken 字段来确定请求的发送方。然后，它检查元数据中的时间戳字段。（对于某些使用案例，命令应仅在指定的时间段内有效。）确认后，风力涡轮机调整叶片角度，并使用 update 主题发布带有新报告状态的响应。

将命令响应发送给管理员

为了让管理员收到关于报告状态更改的更新，管理 UI 订阅 update/accepted 和 update/rejected 主题。在涡轮机将其更新后的状态发布到 IoT Shadow 后，管理员将收到关于 update/accepted 主题的更新消息的确认消息。

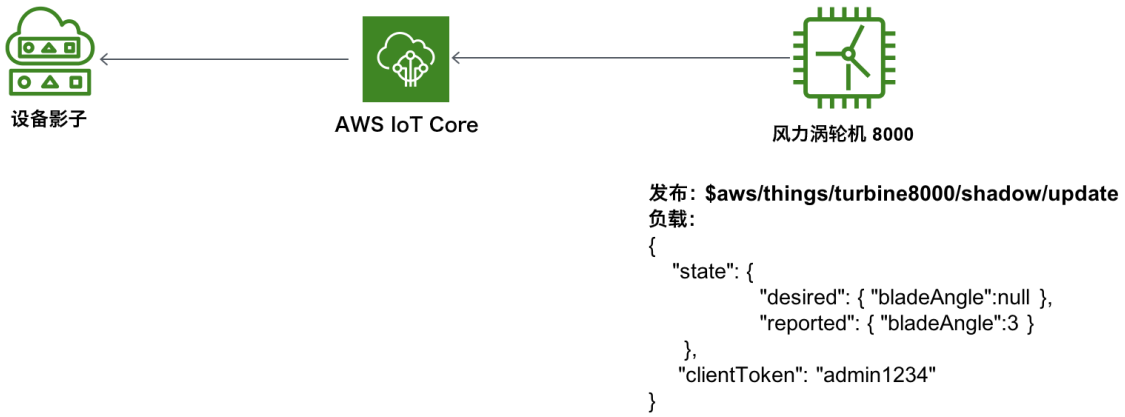


图 *7*：风力涡轮机将更新发布到设备影子

MQTT 命令主题示例

对于智能门锁应用程序，用户必须能够向门锁提交命令，以触发要向获准访问者发放的临时密钥。临时密钥包含 TTL、代码和授权用户的相关信息。临时密钥创建功能可以让其他人在指定的时间内打开门锁。此使用案例适用于以下场景，例如在主屋主工作期间拜访到家的家人。

此场景假设以下详细信息：

- 屋主的移动设备 ID 为 mobile-1
- 获准拜访者的移动设备 ID 为 mobile-2
- 智能锁与家里安装的其他锁配套。这套锁的上下文为 groupId，其中，groupId 等于 group-3
- 前门智能锁的 lockId 为 lock-1
- 智能锁硬件的产品序列号为 series100。该系列是此产品版本的唯一标识符。

生成智能锁代码的命令请求

主屋主首先向锁发布命令，请求为获准拜访者创建的临时访问代码。命令负载包括屋主移动设备的标识、用于跟踪当前请求随机生成的会话标识符、包含命令类型的操作字段和主题字段。智能锁使用主

题字段中的主题向屋主发布回响应。

发布：cmd/series100/group-3/lock-1/credentials
负载：

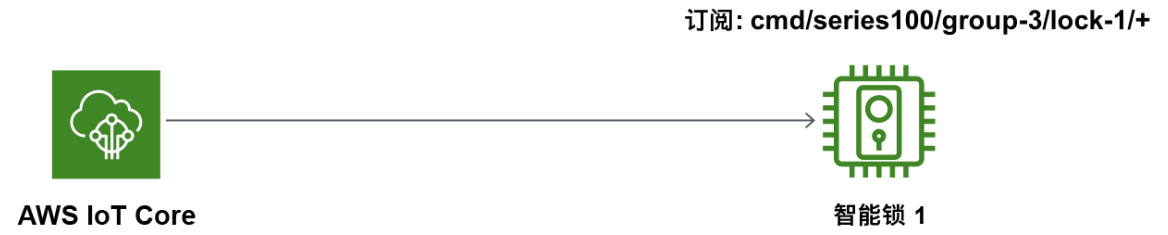
```
{
   "clientId":"mobile-1",
   "sessionId":"0193-0428",
   "topic":"cmd/series100/mobile-1/res",
   "action": {
                "type":"generate-passcode",
                "uid":"visitor-1"
    }
}
```

图 *8*：移动用户请求前门的临时凭证

使用请求、响应和遥测的任意内部标准增强 MQTT 消息。例如，通过在负载中添加命令请求者，应用程序可以根据使用案例指定不同的响应主题。如果屋主请求临时锁定，但需要家中的所有门锁响应，则可以将主题字段更改为发送到一组设备。

智能锁上的命令处理

智能锁接收来自 MQTT 主题的命令消息。通过对此应用程序中的命令使用一致性命名结构，智能锁可以确保它只接收其特定命令主题上的命令。本主题设计还可以通过在锁标识符后使用单级 MQTT 通配符实现对锁的订阅。IoT 应用程序添加新的命令类型后，单级通配符命令会向后兼容。



负载:

```
{
   "clientId":"mobile-1",
   "sessionId":"0193-0428",
   "topic":"cmd/series100/mobile-1/res",
   "action": {
            "type":"generate-passcode",
            "uid":"visitor-1"
    }
}
```

图 *9*：从 *AWS IoT* 设备网关发送到智能锁的命令

收到 MQTT 负载后，智能锁将分析命令请求以确定它运行的操作类型。在这种情况下，该命令将与凭据相关联。设备还会从命令负载中提取客户端 ID、会话 ID 和响应主题。因为一个住宅可以有多个授权屋主，所以客户 ID 要确定是哪个屋主请求了此更改。在此示例中，“action”字段包括凭据请求类型“generate-password”以及与临时密钥关联的用户。最后，设备获取 MQTT 消息中的响应主题字段，并使用此信息发布其响应。

发布: cmd/series100/mobile-1/res
负载:

```
{
   "clientId":"lock-1",
   "sessionId":"0193-0428",
   "passcode":"728e84c6cd8c40ecd47beb5f9cd7f",
   "ttl":"3600",
   "res": {
            "code":"200"
    }
}
```

图 *10*：发布到 *AWS IoT Core* 的智能锁响应

发送到移动客户端的命令响应

屋主的移动客户端订阅组中任何智能锁的命令响应。每当移动客户端收到成功的命令响应时，临时代码以及任何附加授权权限都会在设备上得到处理，并同时存储在 AWS 云中。稍后，授权访问者便可使用此临时代码以及附加安全凭据（例如交换 OAuth 凭据、证明已在门前等），以对智能锁应用临时代码。

在此工作流程中，应用程序对单个设备使用命令主题。但可使用类似的工作流程为组中多个设备请求命令，例如为家中的所有门上锁。

订阅: cmd/series100/mobile-1/+

负载:

```
{
   "clientId":"lock-1",
   "sessionId":"0193-0428",
   "passcode":"728e84c6cd8c40ecd47beb5f9cd7f",
   "ttl":"3600",
   "res": {
              "code":"200"
    }
}
```

图 *11*：从 *AWS IoT Core* 发送到移动客户端的命令响应

MQTT 遥测主题示例

本节讲的是有关聚合来自一组占用传感器的遥测数据示例，这些传感器放置在大楼的各个位置，以监视房间使用情况。占用传感器与运行 AWS IoT Greengrass 的本地网关进行通信。然后，AWS IoT Greengrass 会向 AWS IoT Core 提供有关 MQTT 主题的所有传感器指标。由于此使用案例的重点是遥测，因此 Greengrass 和 AWS IoT Core 之间不需要响应主题，无论是本地还是上游。

此场景假设以下详细信息：

- 占用传感器的 ID 为 occupancy-1 和 occupancy-2
- 此大楼名为 building-fresco
- 每个传感器都放置在 building-fresco 中的特定楼层和房间名称上。
- Greengrass 本地网关的唯一标识符为 gateway-1
- 当前的大楼自动化系统与名为 acme 的内部项目相关联

从占用传感器到 Greengrass 的本地遥测

每个占用传感器每分钟以及每当有人进入或离开房间时，便会发布一次占用读数。由于占用传感器不是与设备的状态精确关联，而是引用房间的状态，因此传感器会在遥测主题中发布房间状态。负载包括时间戳、占用计数以及用于在房间空置时关灯的任何效率倒计时。占用传感器使用的 MQTT 主题包含有关传感器在大楼及其关联项目中位置的上下文信息。Greengrass Core 在本地接收所有占用传感器数据。

发布：dt/acme/building-fresco/room4/occupancy-1

负载：

```
{
   "occupancy":true,
   "delaytiming":664,
   "recordedTime":1532233524
}
```

图 *12*：本地占用传感器将传感器读数发布到 *AWS IoT Greengrass*

从边缘到云的 Greengrass 遥测

在此示例中，Greengrass 的主要作用是聚合来自多个占用传感器的数据，然后将数据发送到 AWS IoT Core。由于 Greengrass 是通往云的本地桥梁，因此 Greengrass 会为每个传感器读数添加元数据。Greengrass 向每条消息添加大楼信息，以 5 分钟为增量显示大楼的总体使用情况。Greengrass 还通过包含适当的应用程序标识符 acme 来补充 MQTT 主题。

发布：dt/acme/building-fresco/gateway-1

负载：

```
[ {
    "occupancy":true,
    "delaytiming":664,
    "recordedTime":1532233524,
    "room":4,
    "sensorId":"occupancy-1"

  },
  …
]
```

图 *13*：*Greengrass* 聚合和补充遥测数据然后将消息转发到 *AWS IoT Core*

在 AWS IoT 规则引擎中使用 MQTT 主题的最佳实践

借助 AWS IoT 规则引擎，您可以定义发送到 AWS IoT Core 的消息与 AWS 服务的交互方式。AWS IoT 规则由 SQL SELECT 语句、主题筛选器和规则操作组成。SQL SELECT 语句可以从传入的 MQTT 消息中提取数据。AWS IoT 规则的主题筛选器将指定哪些 MQTT 主题会触发 IoT 规则操作。

规则引擎在智能地将消息定向到其他 AWS 服务或重新发布到设备方面发挥着关键作用。IoT 规则支持运营指标收集、数据扩充、设备遥测的数据聚合等使用案例，以进行分析以及错误排查。

规则引擎与遥测主题集成

我们建议使用主题结构进行遥测，如下所示：

```
dt/<application-prefix>/<context>/<thing-name>/<dt-type>
```

MQTT 主题遥测模式中定义为 application-prefix 的第二个字段表示队列中设备之间不可变的自然分支。应用程序的常见属性是设备硬件版本或 IoT 应用程序的名称。使用遥测 MQTT 结构，您可以创建 IoT 规则来捕获与特定应用程序版本关联的所有遥测数据：

```
{
 "sql":"SELECT *, topic(2) as applicationVersion, topic(3) as contextIdentifier FROM 'dt/#'",
 "awsIoTSqlVersion":"2016-03-23",
 "ruleDisabled":false,
 "actions":[{
   ...
 }]
}
```

由于 MQTT 主题结构反映了层次结构，因此此规则可以选择 MQTT 主题层次结构的不同部分并将其注入到新的负载中。在消息得到处理并存储在其他 AWS 服务中时，这些属性会提供进一步的上下文。

规则引擎与命令主题集成

无论是使用 AWS IoT Device Shadow、IoT 作业还是使用 MQTT 命令主题发送命令，规则引擎都是一项用于辨别命令成功与否的理想服务。

跟踪命令的成功情况

AWS IoT 规则引擎可用于跟踪各个命令的成功率。IoT 规则可推断负载信息，例如会话标识符；在规则 SELECT 语句中生成其他元数据，例如创建有效时间；并临时将新消息负载存储到数据存储中，例如 Amazon DynamoDB。接下来的规则为 AWS 客户呈现了此使用案例的常见实现。IoT 规则将每个会话存储为单个 DynamoDB 记录，并且由于 WHERE 子句将其标识为传入命令，因此该规则将添加名为 status 的文本值，该值将命令标记为 In progress。

```
{
 "sql":"SELECT session ID AS token,timestamp()/1000 as ttl, topic AS responseTopic, client ID AS
requestorId, action.type AS commandType, 'In Progress' AS status FROM 'cmd/series100/# WHERE
topic(5) == 'credentials' ",
 "awsIoTSqlVersion":"2016-03-23",
 "ruleDisabled":false,
 "actions":[{
   "dynamoDBv2": {
      "roleArn":"arn:aws:sns:us-east-1:12345678:sns_role",
      "putItem":{
         "tableName": "command_sessions_table"
      }
   }
 }]
}
```

当 IoT 生态系统接收命令请求时，上述规则会保留所有传输中命令的记录。

通过遵循 MQTT 主题最佳实践，响应主题包括作为命令本身重叠的信息（例如，原始会话 ID 和智能锁使用的响应主题）。作为附加功能，云应用程序可能具有第二个 IoT 规则，该规则使用会话 ID 通过响应元数据中的信息来更新特定命令的状态。

请参阅以下示例：

```
{
 "sql":"SELECT session ID AS token, timestamp()/1000 as ttl, topic() AS responseTopic, client ID AS requestorId, res.code AS responseCode, 'Complete' AS status FROM 'cmd/series100/# WHERE topic(5) == 'res' ",
 "awsIoTSqlVersion":"2016-03-23",
 "ruleDisabled":false,
 "actions":[{
   "dynamoDBv2": {
     "roleArn":"arn:aws:sns:us-east-1:12345678:sns_role",
     "putItem":{
       "tableName": "command_sessions_table"
     }
   }
 }]
}
```

使规则引擎功能与 MQTT 主题相匹配

在定义 IoT 规则的使用方式时，请查看与 MQTT 主题和 AWS IoT 规则引擎相关的以下建议：

- 使用 topic(Decimal) 规则函数通过 MQTT 主题中包含的上下文信息来补充 MQTT 消息。
- 使用 timestamp() 规则函数包括与消息到达 AWS IoT Core 的时间关联的时间戳。
- 如果您的命令采用的是 JSON 格式，请在规则引擎的 SELECT 和 WHERE 子句中引用所有上下文负载元数据，如会话 ID。其他负载信息可用于确定是否应触发规则以及何时触发规则。
- 在 AWS IoT 操作中使用替换模板将变量表示为触发的 AWS IoT 规则操作部分。借助替换表达式，可以很轻松地缩放和动态路由到下游 IoT 规则操作。
- 要跟踪命令或请求的完成情况，请使用 AWS IoT 规则引擎将数据和状态存储在 DynamoDB 等服务中。处理消息时，可以使用 TTL 字段从 DynamoDB 自动存档数据。如果以高吞吐率发送命令，则可以将 IoT 规则引擎与 Amazon Kinesis 搭配使用在 DynamoDB 存储之前缓冲数据。
- 使用 AWS IoT 规则 WHERE 子句筛选不适用于 AWS IoT 操作的消息。WHERE 子句可与 JSON 负载或规则引擎函数一起使用，例如 get thing shadow(thingName, roleARN) 或 aws lambda(functionArn, inputJson)。
- AWS IoT Core 收到消息后，使用 Amazon Kinesis 或 Amazon SQS 等 AWS 服务来缓冲消息负载以及消息发布到的 MQTT 主题。缓冲消息后，您可以在 AWS Lambda 或 Amazon Elastic Compute Cloud (Amazon EC2) 上运行自己的逻辑，以映射负载或主题中的字段，并使用与各个设备、设备类型或设备组相关的其他元数据来丰富负载。

总结

MQTT 是一种简单、安全、灵活且强大的 IoT 协议。它允许您定义设备和云之间的通信网络，这些网

络可以定制以适应越来越多的客户使用案例。为了就在 AWS IoT 上使用 MQTT 的初始步骤提供支持，本白皮书介绍了在审查如何扩展 IoT 设备通信时可以使用的若干最佳实践、指南和注意事项。
AWS IoT 支持定义与不同使用案例相关的点对点、广播、扇入等多个 MQTT 通信模式。此外，AWS IoT 服务还为您提供其他托管服务，包括但不限于 AWS IoT 作业、AWS IoT Shadow 和 AWS IoT 规则引擎。扩展 IoT 解决方案时，请使用定义遥测和命令的最佳实践，同时确保整个 MQTT 设计不会与本文中概述的 MQTT 总体最佳实践相冲突。最后，考虑 IoT 主题结构如何影响云中的运营和业务可见性。要实现强大的业务价值，您必须规划、开发、设计和使用 AWS IoT 规则，以充分利用 IoT 应用程序的 MQTT 主题架构。

贡献者

以下是对此文档做出贡献的个人和组织：

- Olawale Oladehin、Solutions Architect、AWS IoT

延伸阅读

有关更多信息，请参阅以下内容：

- AWS 白皮书
- AWS IoT Core 文档

文档修订

日期	描述
2019 年 5 月	已更新基本提取功能
2018 年 10 月	首次发布