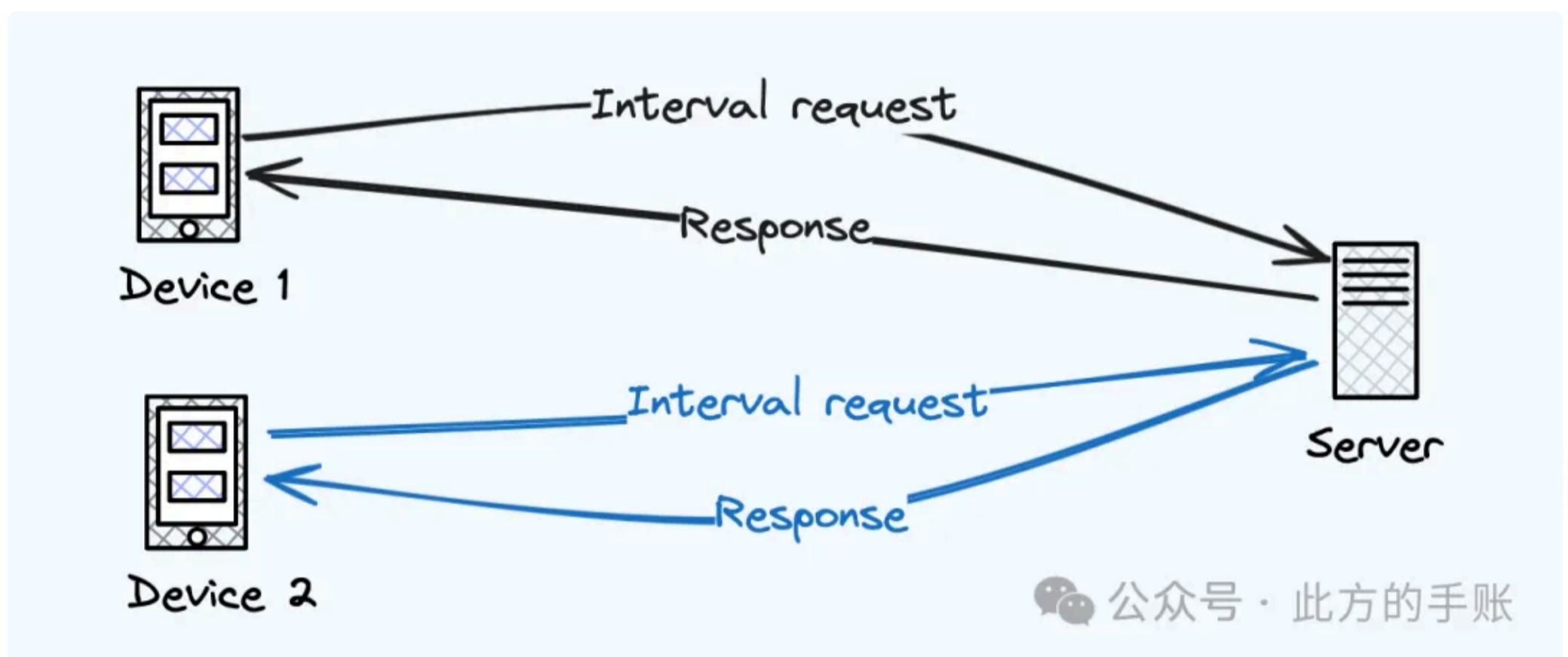


从 HTTP 轮询到 MQTT：我们在 AWS IoT Core 上的架构演进与实战复盘

原创 吃土的小此方 此方的手账 2025年8月19日 11:31 上海

在之前的物联网项目中，由于历史原因，我们的设备通过 HTTP 轮询的方式与服务端交互，这套机制在项目初期看似简单直接，但随着业务量的增长，问题也逐渐暴露出来。



这种模式的痛点非常具体：

1. 电量焦虑

：对于电池供电的设备，频繁的轮询请求就像一个无底洞，快速榨干设备的电量，大大缩短了续航时间。

2. 消息延迟与丢失

：服务端无法主动向设备推送消息。更糟糕的是，如果服务器进行一次版本发布，哪怕只有短短几十秒，也会导致大量的设备消息丢失，这在很多场景下是不可接受的。

3. 扩展性瓶颈

：为了维持状态，服务器被迫设计成“有状态”的，这为水平扩展带来了巨大的麻烦，每次扩容都如履薄冰。

痛定思痛，我们意识到必须寻找一种更现代、更高效的通信方案。团队需要的是一套无状态、易拓展、支持双向通信的架构。经过一番调研和技术选型，MQTT 协议进入了我们的视野。考虑到公司所有基础设施都在 AWS 上，我们自然而然地将目光投向了 AWS IoT Core。

本文将完整复盘我们团队是如何从零开始，基于 AWS IoT Core 构建 MQTT 通信体系的全过程，分享其中的架构设计思考、关键决策和一些“踩过才知道”的实践心得。

注：本文基于 AWS 架构，需要您对 AWS 服务有基本的了解。对于 AWS 官方文档已有的详细步骤，本文将不再赘述，而是聚焦于架构设计和经验分享。

一、告别轮询，拥抱 MQTT

1.1 MQTT 架构简介

要理解我们为什么选择 MQTT，首先需要了解它的核心思想。

MQTT (Message Queuing Telemetry Transport) 是一种专为物联网 (IoT) 场景设计的**发布/订阅 (Publish/Subscribe)** 消息协议。它的核心是一个**中央消息代理 (Broker)**，所有消息的收发都通过它来中转。

- **发布者 (Publisher)**

：消息的发送方，例如我们的传感器设备。

- **订阅者 (Subscriber)**

：消息的接收方，例如我们的后端服务。

- **主题 (Topic)**

：消息的分类标签，例如 `sensor/temperature/room1`。

发布者将消息发送到特定的主题，而订阅者则通过订阅这些主题来接收消息。Broker 负责将消息精确地从发布者路由到对应的订阅者。

这种架构的精妙之处在于**解耦**。发布者和订阅者之间无需知道对方的存在，它们只关心消息代理和主题。这带来了几个显而易见的好处：

- **轻量高效**

：MQTT 的消息头部非常小，在网络带宽有限、信号不稳定的环境下表现出色。

- **双向通信**

：服务端和设备端都可以是发布者和订阅者，可以随时互相发送消息。

- **高可扩展性**

：Broker 作为核心，可以独立扩展，而后端服务也可以根据业务需求灵活增减。

1.2 为什么选择 AWS IoT Core?

理解了 MQTT 的核心是 Broker，接下来的问题就是：这个 Broker 谁来提供？是自己动手基于开源方案（如 EMQX、Mosquitto）搭建，还是选择云厂商的托管服务？

考虑到团队的规模和维护成本，我们倾向于使用托管服务，将专业的事情交给专业的人。由于我们的技术栈完全构建在 AWS 之上，AWS IoT Core 自然成为了首选。

AWS IoT Core 不仅仅是一个 MQTT Broker，它是一个功能强大的托管云服务，为物联网设备与云端的交互提供了完整的解决方案。在我们的项目中，主要利用了它的以下几个核心组件：

- **设备网关 (Device Gateway)**

：作为设备连接的入口，它像一个超级网关，能够管理海量的设备连接，并支持 MQTT、WebSocket 和 HTTPS 等多种协议。

- **消息代理 (Message Broker)**

：这正是我们需要的核心功能，一个安全、稳定、高性能的 MQTT Broker，负责收发设备消息。

- **身份验证和授权**

：提供了企业级的安全机制（如 X.509 证书），确保只有合法的设备才能连接和收发数据。

- **设备影子 (Device Shadow)**

：这是一个非常实用的功能。它会在云端缓存设备的“状态副本”。即使设备暂时离线，应用程序依然可以读取和修改这个“影子”，待设备重新上线后，状态会自动同步。这对于网络不稳定的场景简直是“神器”。

虽然 AWS IoT Core 功能繁多，但在我们的核心架构中，我们首先聚焦于它的**消息代理**能力，将其作为我们整个 MQTT 通信体系的基石。

二、架构设计：从 0 到 1 构建 MQTT 通信体系

理论知识准备就绪，接下来就是激动人心的架构设计环节。下面是我们用于 POC (Proof of Concept) 验证的核心架构图，它足够简单，又能清晰地展示消息的完整流转路径。

这张图的核心是两条消息路径：

- **上行路径（蓝色）**

：设备 -> AWS IoT Core -> 后端服务

- **下行路径（橙色）**

：后端服务 -> AWS IoT Core -> 设备

接下来，我们详细拆解一下其中的关键节点和注意事项。

| 2.1 身份认证：谁可以连接？

安全是物联网的生命线。AWS IoT Core 提供了非常严格的访问控制，不是任何设备或服务想连就能连的。

- **对于设备端**

：我们采用 **X.509 证书** 进行身份验证。每个设备都会烧录一个在 AWS IoT Core 中注册过的唯一证书。当设备尝试连接时，AWS 会校验该证书的合法性。这种方式远比简单的用户名密码要安全。

- **证书从哪来？**

：你可以在 AWS Console 的 **IoT Core > 管理 > 安全 > 证书** 页面手动创建，然后下载并烧录到设备中。对于大规模生产，可以通过 API 批量注册。

- **证书有何权限？**

：证书的权限由**策略（Policy）** 定义。一个策略就是一个 JSON 文档，详细规定了该证书被允许执行哪些操作（如连接、发布、订阅）以及可以操作哪些 Topic。**POC 阶段为了方便，可以暂时授予 * (所有) 权限，但生产环境严禁如此！**

- **对于服务端**

：我们的后端服务运行在 AWS 的服务器上，因此采用 **IAM Role** (配合 Access Key/Secret Key) 的方式进行认证，这是 AWS 生态内服务间调用的标准实践。

2.2 消息上行：从设备到云端

这条路径是设备上报数据的生命线。

1. 设备使用加载了证书的 MQTT 客户端，连接到 AWS IoT Core 提供的唯一域名。
2. 连接成功后，设备向预先定义好的 Topic (例如 `/dt/sensor/12345/temperature`) 发布消息。
3. 消息到达 AWS IoT Core 的 Message Broker。

2.3 消息下行：从云端到设备

这条路径负责向设备下发指令。

1. 后端服务通过 AWS SDK，使用 IAM Role 授予的权限，向目标设备的 Topic (例如 `/cmd/sensor/12345/reboot`) 发布消息。
2. Message Broker 收到消息后，将其转发给订阅了该 Topic 的设备。
3. 设备端的 MQTT 客户端收到消息，执行相应操作。

上面提到的如创建证书、配置 Policy 等操作，AWS 官方文档有非常详尽的步骤，这里不再赘述，我们更聚焦于“为什么”和“如何设计”。

如果说 Topic 是 MQTT 世界的“街道地址”，那么 Rule 就是这些街道上的“智能交通枢纽”。它们是 AWS IoT Core 的一大特色，也是我们架构设计中的点睛之笔。

3.1 Rule：连接万物的“胶水”

在标准的 MQTT 协议中，如果后端服务要接收消息，它自己也必须是一个 MQTT 客户端。但 AWS IoT Rule 打破了这个限制。

Rule 的核心作用是：监听一个或多个 Topic，当有消息到达时，触发一个预设的动作。

这个“动作”非常强大，几乎可以连接任何你需要的地方：

- 转发到其他 AWS 服务

：将消息无缝写入 DynamoDB、触发 Lambda 函数进行实时处理、推送到 SQS 消息队列等。

- **转发到另一个 Topic**

：实现消息的二次路由和处理。

- **转发到外部 HTTP 端点**

：这是我们项目中采用的方式。Rule 将 MQTT 消息直接转换为 HTTP 请求，发送给我们现有的后端服务。这极大地降低了改造和维护成本，让我们的后端服务无需关心 MQTT 协议本身，专注于业务逻辑即可。

3.2 Topic 设计：不只是命名，更是架构

从架构图中不难发现，Topic 是整个消息路由的关键。一个设计良好的 Topic 结构，不仅能让消息路由清晰，还能在权限控制、问题排查甚至成本优化方面带来巨大帮助。

Topic 分为两类：一类是 AWS 保留的系统 Topic（以 \$ 开头，如 `$aws/events/presence/connected/{clientId}` 用于监控设备连接状态），我们只能订阅不能发布；另一类就是与我们业务紧密相关的自定义 Topic。

设计自定义 Topic 是一门艺术。下面结合一个温度传感器的例子，分享一些我们团队总结的最佳实践。

最佳实践一：动静分离，上行下行要分明

一个常见的误区是设备上报和指令下发使用同一个 Topic。更好的做法是使用不同的前缀或路径来区分数据流向。

- **上行数据（Data）**

：设备 -> 云。建议使用 /dt (data) 或 /up (upload) 作为前缀。

- **例如**

： `/dt/sensor/temp-001/report`

- **下行指令 (Command)**

: 云 -> 设备。建议使用 `/cmd` (command) 或 `/down` (download) 作为前缀。

- 例如

: `/cmd/sensor/temp-001/reboot`

这样做好处是权限控制可以做得非常精细。例如，你可以配置策略，只允许设备向 `/dt/` 开头的 Topic 发布消息，而只能从 `/cmd/` 开头的 Topic 订阅消息，最小化安全风险。

最佳实践二：层级清晰，包含关键实体信息

Topic 的层级结构应该像一个有意义的 URL。将关键的实体信息（如设备类型、设备 ID）放入 Topic 中，而不是全部塞进消息体（Payload）。

- **推荐结构**

: `{direction}/{deviceType}/{deviceId}/{action}`

- **好的例子**

: `/dt/sensor/temp-001/report`

- **不好的例子**

: `/report` (消息体里是 `{"deviceType": "sensor", "deviceId": "temp-001"}`)

将关键信息放在 Topic 里有两大优势：

1. **路由更高效**

: 你可以使用通配符 (+ 和 #) 对一类设备进行订阅，而无需解析消息体。例如，订阅 `/dt/sensor/#` 可以获取所有传感器的上报数据。

2. **Rule 处理更便捷**

: 在 IoT Rule 的 SQL 中，你可以用 `topic(n)` 函数直接从 Topic 路径中提取信息。例如，对于 `/dt/sensor/temp-001/report`，`topic(3)` 的结果就是 `temp-`

001。这可以让你在将消息转发到后端服务时，轻松地将设备 ID 等信息带上，避免了复杂的 JSON 解析。

最佳实践三：预留空间，兼顾当下与未来

AWS IoT Core 对 Topic 的层级有数量限制（最多 7 层，总长度不超过 256 字节）。在设计时，既要包含当前业务所需的所有信息，也要为未来的功能扩展预留空间。

{direction}/{deviceType}/{deviceId}/{action} 这样的四层结构通常能满足大部分场景，并且还留有扩展的余地（例如，可以在后面增加版本号或子模块）。

最佳实践四：巧用通配符，实现灵活订阅

MQTT 的通配符是其强大功能之一，要善加利用。

- +

(加号)：单层通配符，可以匹配一个层级。例如 dt/sensor/+/report 可以匹配 dt/sensor/temp-001/report 和 dt/sensor/humidity-002/report，但不能匹配 dt/sensor/temp-001/extra/report。

- #

(井号)：多层通配符，必须放在 Topic 的末尾，可以匹配任意数量的后续层级。例如 dt/sensor/# 可以匹配上面两个例子以及 dt/sensor/temp-001/extra/report。

通过通配符，运维人员或后端服务可以非常灵活地订阅他们关心的消息流，而无需为每个设备单独配置。

想了解更多？

强烈推荐阅读 AWS 官方白皮书：MQTT design best practices，里面有更详尽的案例和深度思考。

四、总结与展望

从最初面对 HTTP 轮询的种种弊端，到最终选择并实践 AWS IoT Core，我们团队完成了一次重要的技术升级。本文我们从“为什么选择 MQTT”出发，介绍了其核心架构和 AWS IoT Core 的优势，并详细拆解了我们从 0 到 1 构建通信体系的架构设计，最后，也是最重要的，分享了我们在 Topic 和 Rule 设计上总结出的“四项基本原则”。

然而，这仅仅是万里长征的第一步。真实的生产环境远比这复杂，我们会面临设备批量预配、海量消息的成本优化、更复杂的 Rule 引擎设计、以及如何与现有的业务系统更深度集成等一系列挑战。

在下一篇文章中，我将基于这套核心架构进行扩展，分享我们在生产环境中如何应对这些挑战，以及成本优化方面的实践。

参考资料

1. MQTT - The Standard for IoT Messaging
2. MQTT Essentials: Your 2025 Learning Hub for IoT & IIoT Data Streaming
3. What is AWS IoT? - AWS IoT Core
4. How AWS IoT works - AWS IoT Core
5. AWS IoT Core Features - Amazon Web Services
6. MQTT design best practices

上一篇

MCP 探索笔记：查找与管理你需要的 MCP

下一篇

AWS IoT Core 成本优化实战：从 PoC 到生产的省钱之旅

作者提示: 个人观点，仅供参考