AZURE-IOT-SDK 用户手册

RT-THREAD 文档中心

WWW.RT-THREAD.ORG

Friday 28th September, 2018

版本和修订

Date	Version	Author	Note
2018-07-27	v0.1	SummerGift	初始版本

目录

第 1 章

软件包介绍

Azure 是 RT-Thread 移植的用于连接微软 Azure IoT 中心的软件包，原始 SDK 为：**azure-iot-sdk-c**。通过该软件包，可以让运行 RT-Thread 的设备轻松接入 Azure IoT 中心。

Azure IoT 中心的服务托管在云中运行，充当中央消息中心，用于 IoT 应用程序与其管理的设备之间的双向通信。通过 Azure IoT 中心，可以在数百万 IoT 设备和云托管解决方案后端之间建立可靠又安全的通信，生成 IoT 解决方案。几乎可以将任何设备连接到 IoT 中心。

使用 Azure 软件包连接 IoT 中心可以实现如下功能：

- 轻松连入 Azure IoT 中心，建立与 Azure IoT 的可靠通讯
- 为每个连接的设备设置标识和凭据，并帮助保持云到设备和设备到云消息的保密性
- 管理员可在云端大规模地远程维护、更新和管理 IoT 设备
- 从设备大规模接收遥测数据
- 将数据从设备路由到流事件处理器
- 从设备接收文件上传
- 将云到设备的消息发送到特定设备

可以使用 Azure IoT 中心来实现自己的解决方案后端。此外，IoT 中心还包含标识注册表，可用于预配设备、其安全凭据以及其连接到 IoT 中心的权限。

1.1 软件包目录结构

```
azure
+---azure                                  // Azure 云平台 SDK
+---azure-port                             // 移植文件
+---docs
|     +---figures                          // 文档使用图片
|     |   api.md                           // API 使用说明
|     |   introduction.md                  // 介绍文档
|     |   principle.md                     // 实现原理
|     |   README.md                        // 文档结构说明
|     |   samples.md                       // 软件包示例
|     |   user-guide.md                    // 使用说明
|     +---version.md                       // 版本
+---samples                                // 示例代码
|     |   iothub_ll_telemetry_sample       // 设备上传遥测数据示例
|     +---iothub_ll_c2d_sample             // 设备接收云端数据示例
|   LICENSE                                // 软件包许可证
|   README.md                              // 软件包使用说明
+---SConscript                             // RT-Thread 默认的构建脚本
```

1.2 Azure 简述

微软云在中国区由世纪互联代理提供中国内的服务，国际版的微软云提供全球的服务。世纪互联本土运营的 Microsoft Azure 为不同需求的组织和个人提供了涵盖基础架构、数据库、Web 应用、人工智能、物联网等领域的全面云能力，充分满足了不同行业、不同规模、不同 IT 水平用户的多样化业务需求。

由世纪互联运营的 Microsoft Azure 是第一个在中国正式商用，符合中国政府相关法规要求的国际化公有云服务。该服务由在中国获得公有云业务许可的中国公司世纪互联负责运营，相比其他落地中国的国际公有云服务商，Azure 有一些醒目的优势：稳定高可用，可提供高达 95% 的服务级别协议保障。

由世纪互联运营的 Microsoft Azure 使用位于中国，物理隔离的云服务实例，这些服务采用了微软服务于全球的 Azure 技术，能为客户提供全球一致的服务质量保障。在物联网领域，主要使用微软云的 Azure IoT 中心来完成设备的云连接需求。

1.3 功能介绍

Azure IoT 中心的架构图如下所示：

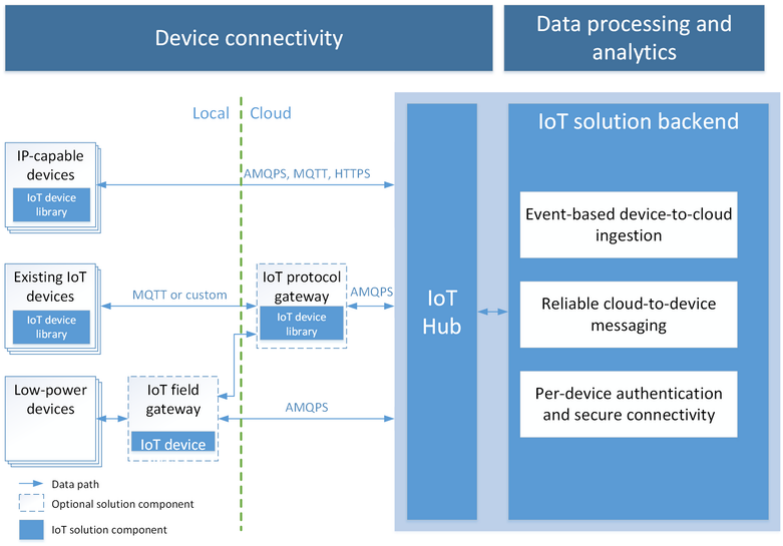

图 1.1: *Azure IoT 中心的架构图*

Azure IoT 中心提供如下功能：

- 与数百万 IoT 设备建立双向通信

依靠 Azure IoT 中心可以轻松安全地连接物联网 (IoT) 资产。在云到设备消息中，可靠地向连接的设备发送命令和通知，并通过确认回执跟踪消息传递。通过持久的方法发送设备消息，以适应间歇性连接的设备。

- 平台与协议

使用适用于多个平台（包括 Linux、Windows 和实时操作系统）的开放源代码设备 SDK 添加新设备并连接现有设备。使用标准协议和自定义协议，包括 **MQTT v3.1.1、HTTPS 1.1 或 AMQP 1.0 协议**。

- 物联网安全性

为每个连接的设备设置标识和凭据，并帮助保持云到设备和设备到云消息的保密性。要保持系统的完整性，请根据需要选择性地撤消特定设备的访问权限。可以为每个设备设置独有的 安全密钥，让它连接到 IoT 中心。 IoT 中心标识注册表 会在解决方案中存储设备标识和密钥。解决方案后端可将单个设备添加到允许或拒绝列表，以便完全掌控设备访问权限。

其他连接安全功能包括：

```
- 设备和 Azure IoT 中心之间，或网关和 Azure IoT 中心之间的通信路径，将配
    合使用 X.509 协议身份验证的 Azure IoT 中心使用行业标准的传输层安全 (
    TLS) 来保护。
```

```
- 为了保护设备以防止来路不明的入站连接，Azure IoT 中心不会打开任何设备的连接。 设备将发起所有连接。

- Azure IoT 中心永久存储设备的消息，并等待连接设备。 这些命令存储两天，使设备能够基于电源或连接因素偶而进行连接来接收这些命令。 Azure IoT 中心维护每个设备的设备队列。
```

- 通过设备管理，大规模管理 IoT 设备

借助 Azure IoT 中心新的设备管理功能，管理员可在云端大规模地远程维护、更新和管理 IoT 设备。设备孪生：可以使用设备孪生存储、同步和查询设备元数据和状态信息。设备孪生是存储设备状态信息（例如元数据、配置和条件）的 JSON 文档。IoT 中心为连接到 IoT 中心的每台设备维护一个设备孪生，设备数量可以达到数百万的规模。

- 通过 Azure IoT 网关 SDK 利用边缘智能

IoT 网关 SDK 提供强大的框架来生成、配置和部署边缘逻辑，从而让你能够使用 Azure IoT 做更多的事。

1.4 平台和硬件兼容性

对接入 Azure IoT 中心的设备平台要求如下：

- 能够建立 IP 连接：只有支持 IP 的设备才能与 Azure IoT 中心直接通信
- 支持 TLS：与 Azure IoT 中心建立安全通信通道所需
- 支持 SHA-256（可选）：生成用于使用服务验证设备的安全令牌所必需的。可以使用不同的身份验证方法，并非全部都需要 SHA-256
- 有一个实时时钟或实现代码连接到 NTP 服务器：建立 TLS 连接和生成安全令牌进行身份验证所必需的
- 拥有至少 64KB 的 RAM：SDK 的内存占用量取决于所使用的 SDK 和协议以及目标平台。针对微控制器实现了最小的空间占用

第 2 章

工作原理

2.1 IoT 客户端框架

Azure IoT 中心为了方便设备连接提供了丰富的连接协议，如 MQTT、HTTP 等，同时 Azure IoT 中心只支持安全连接。与 IoT 中心的连接由设备客户端来完成，每一个连接到 IoT 中心的设备都会创建一个 IoT 中心客户端实例，当连接关闭时，将这个实例释放掉即可。

IoT 中心客户端会向下调用 LL 层来完成工作，LL 层向下对接不同通信协议的传输层，传输层向下对接通信协议实现层。下面两幅图展示了 IoT 中心客户端完成功能时的调用层次关系：

- IoT 客户端框架 HTTP/MQTT 功能调用关系图：

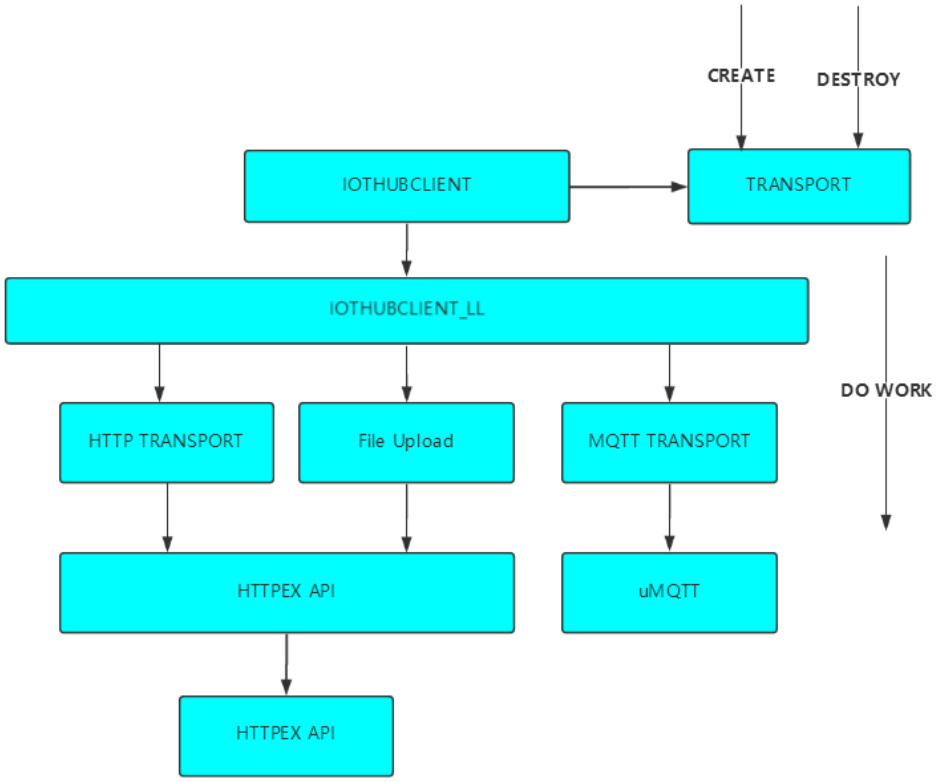

图 2.1: *HTTP/MQTT 功能调用关系图*

- 下图以 HTTP 协议为例展示 API 调用情况：

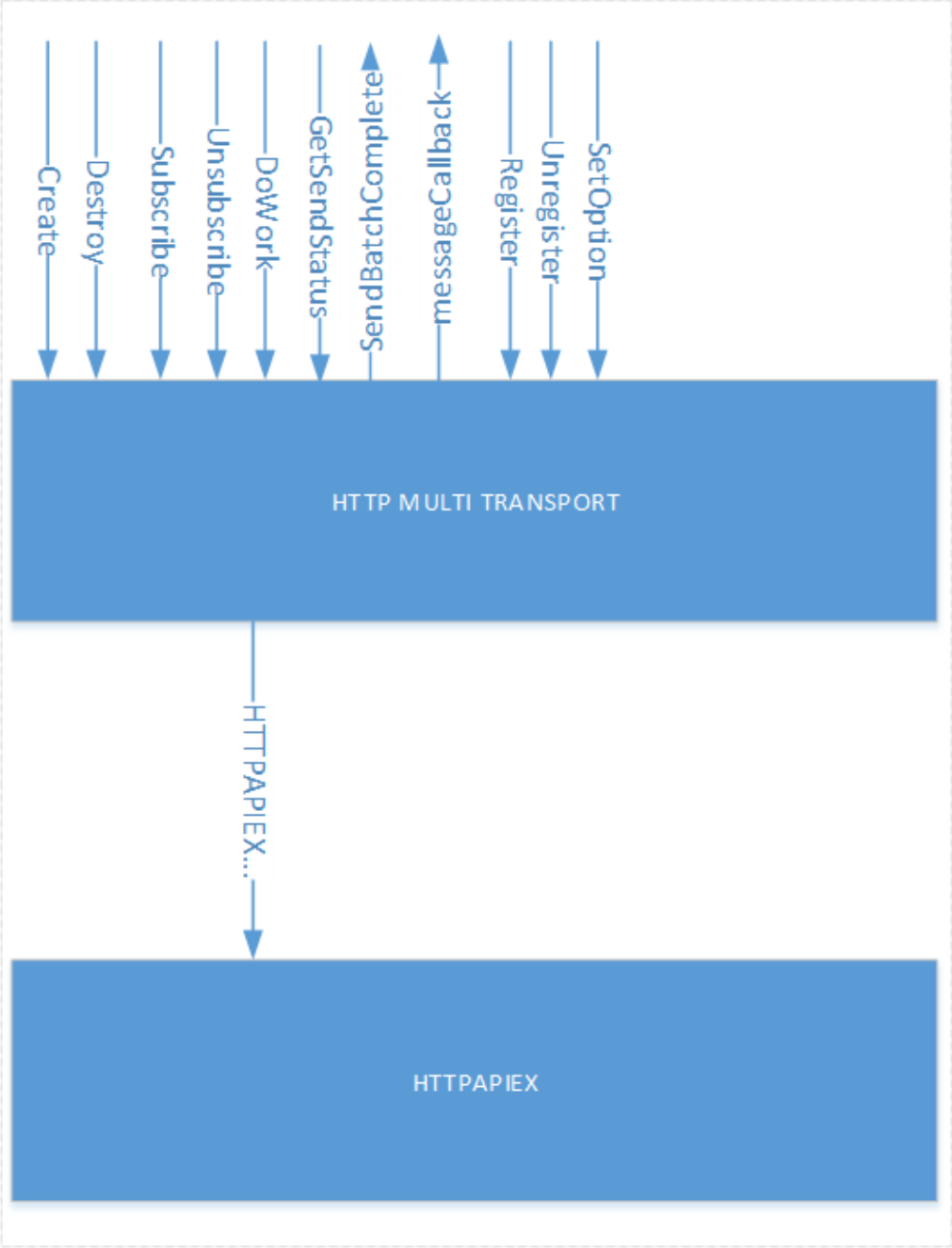

图 2.2: *HTTP 协议 API 调用*

第 3 章

使用指南

Azure IoT 中心是一项完全托管的服务，有助于在数百万台设备和单个解决方案后端之间实现安全可靠的双向通信。Azure IoT 中心提供如下功能：

- 使用每个设备的安全凭据和访问控制来保护通信安全
- 多个设备到云和云到设备的超大规模通信选项
- 各设备状态信息和元数据的可查询存储
- 通过最流行语言和平台的设备库来方便建立设备连接

3.1 设备到云通信

将信息从设备应用发送到解决方案后端时，IoT 中心会公开三个选项：

- 设备到云消息，用于时序遥测和警报
- 设备克隆的报告属性用于报告设备状态信息，例如可用功能、条件或长时间运行的工作流的状态。例如，配置和软件更新
- 文件上传，用于由间歇性连接的设备上传的或为了节省带宽而压缩的媒体文件和大型遥测批文件

下面是各种设备到云通信选项的详细比较：

	设备到云的消息	设备克隆的报告属性	文件上传
方案	遥测时序和警报。 例如，每隔 5 分钟发送的 256KB 传感器数据批。	可用功能和条件。 例如，当前设备连接模式，诸如手机网络或 WiFi。 同步长时间运行的工作流，如配置和软件更新。	媒体文件。 大型（通常为压缩的）遥测批。
存储和检索	通过 IoT 中心临时进行存储，最多存储 7 天。 仅顺序读取。	通过 IoT 中心存储在设备孪生中。 可使用 IoT 中心查询语言检索。	存储在用户提供的 Azure 存储帐户中。
大小	消息大小最大为 256-KB。	报告属性大小最大为 8 KB。	Azure Blob 存储支持的最大文件大小。
频率	高。 有关详细信息，请参阅 IoT 中心限制。	中。 有关详细信息，请参阅 IoT 中心限制。	低。 有关详细信息，请参阅 IoT 中心限制。
协议	在所有协议上可用。	使用 MQTT 或 AMQP 时可用。	在使用任何协议时可用，但设备上必须具备 HTTPS。

图 3.1: *设备到云通信*

应用程序可能需要同时将信息作为遥测时序或警报发送，并且使其在设备孪生中可用。在这种情况下，可以选择以下选项之一：

- 设备应用发送一条设备到云消息并报告属性更改
- 解决方案后端在收到消息时可将信息存储在设备孪生的标记中

由于设备到云消息允许的吞吐量远高于设备孪生更新，因此有时需要避免为每条设备到云消息更新设备孪生。

3.2 云到设备通信

IoT 中心提供三个选项，允许设备应用向后端应用公开功能：

- 直接方法，适用于需要立即确认结果的通信。直接方法通常用于以交互方式控制设备，例如打开风扇
- 孪生的所需属性，适用于旨在将设备置于某个所需状态的长时间运行命令。例如，将遥测发送间隔设置为 30 分钟
- 云到设备消息，适用于向设备应用提供单向通知

下面详细比较了各种从云到设备的通信选项：

	直接方法	克隆的所需属性	云到设备的消息
方案	需要立即确认的命令，例如打开风扇。	旨在将设备置于某个所需状态的长时间运行命令。例如，将遥测发送间隔设置为 30 分钟。	提供给设备应用的单向通知。
数据流	双向。设备应用可以立即响应方法。解决方案后端根据上下文接收请求结果。	单向。设备应用接收更改了属性的通知。	单向。设备应用接收消息
持续性	不联系已断开连接的设备。通知解决方案后端：设备未连接。	设备孪生会保留属性值。设备会在下次重新连接时读取属性值。属性值可通过 IoT 中心查询语言检索。	IoT 中心可保留消息长达 48 小时。
目标	通过 **deviceId**与单个设备通信，或通过 作业与多个设备通信。	通过 **deviceId**与单个设备通信，或通过 作业与多个设备通信。	通过 **deviceId**与单个设备通信。
大小	直接方法有效负载的最大大小为 128 KB。	所需属性大小最大为 8 KB。	最多 64 KB 消息。
频率	高。有关详细信息，请参阅 IoT 中心限制。	中。有关详细信息，请参阅 IoT 中心限制。	低。有关详细信息，请参阅 IoT 中心限制。
协议	使用 MQTT 或 AMQP 时可用。	使用 MQTT 或 AMQP 时可用。	在所有协议上可用。使用 HTTPS 时，设备必须轮询。

图 3.2: *云到设备通信*

在以下教程中学习如何使用直接方法、所需属性以及从云到设备的消息：

- 使用直接方法：针对直接方法
- 使用所需属性配置设备：针对设备孪生的所需属性
- 发送从云到设备的消息：针对从云到设备的消息

3.3 参考资料

Azure 文档中心提供了丰富的开发资料，更多详细资料可以参考如下地址：

- Azure 官网
- IoT 中心文档
- Azure 开发指南

第 4 章

API 说明

在使用 Azure 编写应用的过程中，使用 **IoTHubClient** 库就可以轻松发送和接收消息，本章介绍 **IoTHubClient** 库提供的常用 API。更多详细的 API 请参考 Azure SDK 的 C API 参考。

4.1 编程模型选择

IoTHubClient 库提供两套简单易用的 API，一套 API 的名称中包含 “LL”，另外一套则不包含，名称中包含 “LL” 的 API 级别较低。无论选择带 “LL” 的 API 还是选择不带 “LL” 的 API，都必须前后相一致。如果首先调用 **IoTHubClient_LL_CreateFromConnectionString**，则对于任何后续工作，请务必只使用相应的较低级别的 API：

- IoTHubClient_LL_SendEventAsync
- IoTHubClient_LL_SetMessageCallback
- IoTHubClient_LL_Destroy
- IoTHubClient_LL_DoWork

相反的情况也成立。如果首先调用 **IoTHubClient_CreateFromConnectionString**，请使用非 LL API 进行任何其他处理。

这些函数的 API 名称中包含 “LL”。除此之外，其中每个函数的参数都与其非 LL 的对等项相同。但是，这些函数的行为有一个重要的差异。

当你调用 IoTHubClient_CreateFromConnectionString 时，基础库将创建在后台运行的新线程。此线程将事件发送到 IoT 中心以及从 IoT 中心接收消息。使用 “LL” API 时则不会创建此类线程。创建后台线程是为了给开发人员提供方便。你无需担心如何明确与 IoT 中心相互发送和接收消息，因为此操作会在后台自动进行。相比之下，借助 “LL”API，可根据需要明确控制与 IoT 中心的通信。

4.2 IoTHubClient 常用 API

下面以名称中包含 “LL” 的 API 为例介绍常用 API。

4.2.1 物联网中心客户端初始化

IOTHUB_CLIENT_LL_HANDLE

IoTHubClient_LL_CreateFromConnectionString(

const char* connectionString,

IOTHUB_CLIENT_TRANSPORT_PROVIDER protocol)

使用指定的连接字符串参数创建一个物联网中心客户端来与现有的物联网中心通信。

参数	描述
connectionString	设备连接字符串
protocol	协议实现的函数指针
返回	**描述**
IOTHUB_CLIENT_LL_HANDLE	一个非空的 iothubclientllhandle 值，它在调用物联网中心客户端的其他函数时使用
NULL	失败

4.2.2 物联网中心客户端资源释放

void

IoTHubDeviceClient_LL_Destroy(

IOTHUB_DEVICE_CLIENT_LL_HANDLE iotHubClientHandle)

释放物联网中心客户端分配的资源。这是一个阻塞的调用。

参数	描述
iotHubClientHandle	设备连接字符串
返回	**描述**
无	无

4.2.3 设置发送事件回调函数

IOTHUB_CLIENT_RESULT

IoTHubClient_LL_SendEventAsync(

IOTHUB_CLIENT_LL_HANDLE iotHubClientHandle,

IOTHUB_MESSAGE_HANDLE eventMessageHandle,

IOTHUB_CLIENT_EVENT_CONFIRMATION_CALLBACK eventConfirmation-Callback,

void* userContextCallback)

异步调用发送由 eventMessageHandle 指定的消息。

参数	描述
iotHubClientHandle	已创建的物联网中心客户端句柄
eventMessageHandle	物联网中心消息句柄
eventConfirmation Callback	由设备指定的回调，用于接收物联网中心消息的交付确认。这个回调可以被同一消息调用：当 iothubclientllsendeventasync 函数，试图重试发送一条失败的消息。用户可以在这里指定一个 NULL 值，以表明不需要回调
userContextCallback	用户指定的上下文将被提供给回调，可以为空
返回	**描述**
IOTHUB_CLIENT_OK	设置成功
error code	失败

4.2.4 设置接收消息回调函数

IOTHUB_CLIENT_RESULT

IoTHubDeviceClient_LL_SetMessageCallback(

IOTHUB_DEVICE_CLIENT_LL_HANDLE iotHubClientHandle,

IOTHUB_CLIENT_MESSAGE_CALLBACK_ASYNC messageCallback,

void* userContextCallback)

当物联网中心向设备发送消息时，设置要调用的消息回调。这是一个阻塞的调用。

参数	描述
iotHubClientHandle	已创建的物联网中心客户端句柄
messageCallback	由设备指定的回调，用于接收来自物联网中心的消息
userContextCallback	用户指定的上下文将被提供给回调，可以为空
返回	**描述**
IOTHUB_CLIENT_RESULT	初始化 IoT 客户端成功
NULL	失败

4.2.5 设置物理中心客户端

IOTHUB_CLIENT_RESULT

IoTHubDeviceClient_LL_SetOption

IOTHUB_DEVICE_CLIENT_LL_HANDLE iotHubClientHandle,

const char* optionName, const void* value)

这个 API 设置了一个由参数 `optionName` 标识的运行时选项，选项名和数据类型值指向每个选项都是特定的。

参数	描述
iotHubClientHandle	已创建的物联网中心客户端句柄
optionName	选项名称
value	特定值
返回	**描述**
IOTHUB_CLIENT_OK	设置成功
error code	失败

4.2.6 设置连接状态回调

IOTHUB_CLIENT_RESULT

IoTHubDeviceClient_LL_SetConnectionStatusCallback(

IOTHUB_DEVICE_CLIENT_LL_HANDLE iotHubClientHandle,

IOTHUB_CLIENT_CONNECTION_STATUS_CALLBACK connectionStatusCallback,

void * userContextCallback)

设置要调用的连接状态回调，表示连接到物联网中心的状态。这是一个阻塞的调用。

参数	描述
iotHubClientHandle	已创建的物联网中心客户端句柄
connectionStatusCallback	设备指定的回调，用于接收关于连接到物联网中心的状态的更新
userContextCallback	用户指定的上下文将被提供给回调，可以为空
返回	**描述**
IOTHUB_CLIENT_OK	设置成功
error code	失败

4.2.7 完成（发送/接收）工作

void

IoTHubDeviceClient_LL_DoWork(

IOTHUB_DEVICE_CLIENT_LL_HANDLE iotHubClientHandle)

当工作（发送/接收）可以由 IoTHubClient 完成时，该函数将被用户调用。所有 IoTHubClient 交互（关于网络流量和/或用户级回调）都是调用这个函数的效果，它们在 DoWork 中同步进行。

参数	描述
iotHubClientHandle	已创建的物联网中心客户端句柄
返回	**描述**
无	无

第 5 章

示例程序

5.1 简介

IoT 中心是一项 Azure 服务，用于将大量遥测数据从 IoT 设备引入云中进行存储或处理。本次示例程序将展示设备与 IoT 中心之间进行数据交换的功能。

接下来我们会运行 Azure 软件包提供的两个功能示例，一个示例用于展示设备向云端发送遥测数据功能，另一个示例用于展示接收物联网中心下发数据的功能。运行这两个示例程序之前，需要先创建 IoT 中心并在 IoT 中心上注册设备。

5.2 准备工作

在运行本次示例程序之前需要准备工作如下：

1、注册微软 Azure 账户，如果没有 Azure 订阅，请在开始前创建一个试用帐户。

2、安装 DeviceExplorer 工具，这是一个 windows 平台下测试 Azure 软件包功能必不可少的工具。该工具的安装包为 tools 目录下的 SetupDeviceExplorer.msi，按照提示安装即可，成功运行后的界面如下图。

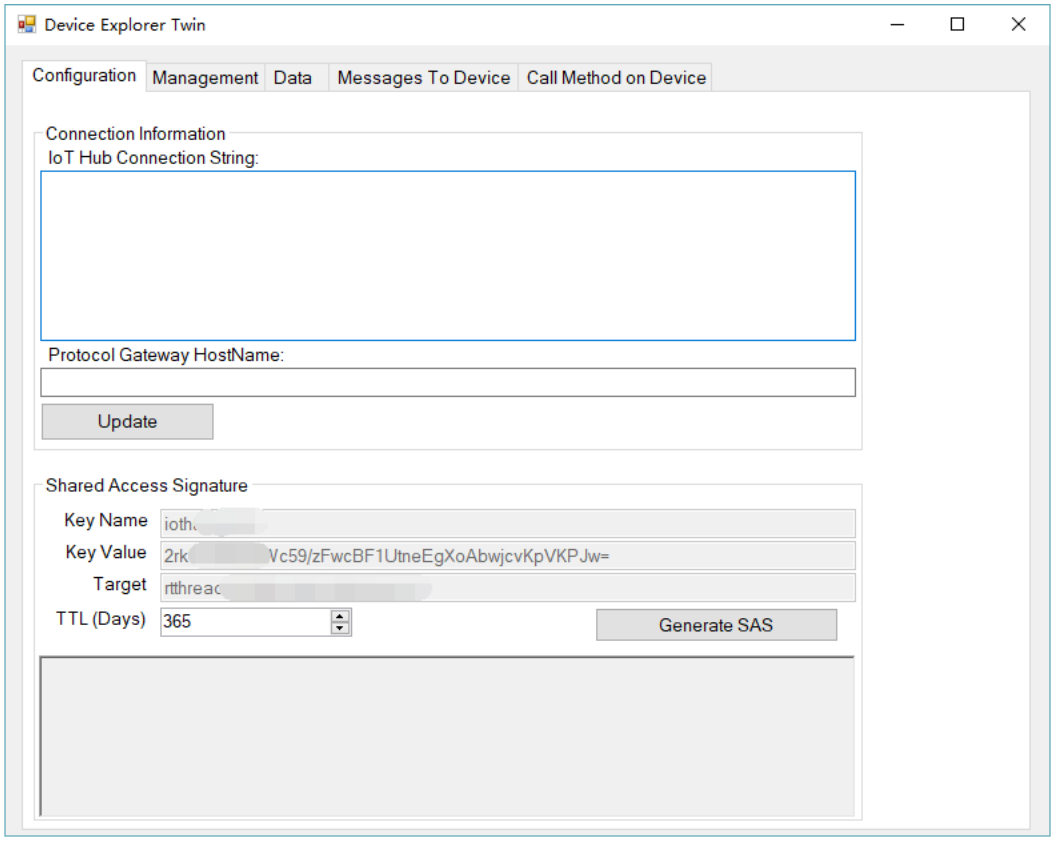

图 5.1: *DeviceExplorer 工具界面*

5.3 获取 Azure 软件包

- 将软件包加入工程，并将系统运行在拥有足够资源的开发板上

打开 RT-Thread 提供的 env 工具，在 bsp 目录下使用 menuconfig 命令打开配置选项，选中 `Azure: Microsoft azure cloud SDK for RT-Thread` 软件包，开启需要使用的功能示例。接下来使用 `pkgs --update` 命令即可将 azure 软件包加入到系统中，env 工具会自动添加该软件包的相关依赖。

```
RT-Thread online packages  --->
    IoT - internet of things  --->
        IoT Cloud  --->
          [*] Azure: Microsoft azure cloud SDK for RT-Thread  --->
          Choose Protocol (Using MQTT Protocol)  --->  #选择通信协议
          [*]   Enable Azure iothub telemetry example  #设备发送遥测数据
              示例
          [*]   Enable Azure iothub cloud to device example #云端下发数据
              示例
```

```
        Version (latest)  --->                    #选择软件包版本
```

5.4 通信协议介绍

目前 RT-Thread Azure 软件包提供的示例代码支持 MQTT 和 HTTP 的通信协议，想要使用哪种协议，只需要在上面选项中选择相应的协议即可。在选择设备端通信协议时，需要注意以下几点：

1、当进行云到设备数据发送时，由于 HTTPS 没有用于实现服务器推送的有效方法。因此，使用 HTTPS 协议时，设备会在 IoT 中心轮询从云到设备的消息。此方法对于设备和 IoT 中心而言是低效的。根据当前 HTTPS 准则，每台设备应每 25 分钟或更长时间轮询一次消息。MQTT 支持在收到云到设备的消息时进行服务器推送。它们会启用从 IoT 中心到设备的直接消息推送。如果传送延迟是考虑因素，最好使用 MQTT 协议。对于很少连接的设备，HTTPS 也适用。

2、使用 HTTPS 时，每台设备应每 25 分钟或更长时间轮询一次从云到设备的消息。但在开发期间，可按低于 25 分钟的更高频率进行轮询。

3、更详细的协议选择文档请参考 Azure 官方文档《选择通信协议》。

5.5 创建 IoT 中心

首先要做的是使用 Azure 门户在订阅中创建 IoT 中心。IoT 中心用于将大量遥测数据从许多设备引入到云中。然后，该中心会允许一个或多个在云中运行的后端服务读取和处理该遥测数据。

1、登录到 Azure 门户。2、选择 “创建资源” > “物联网” > “IoT 中心”。

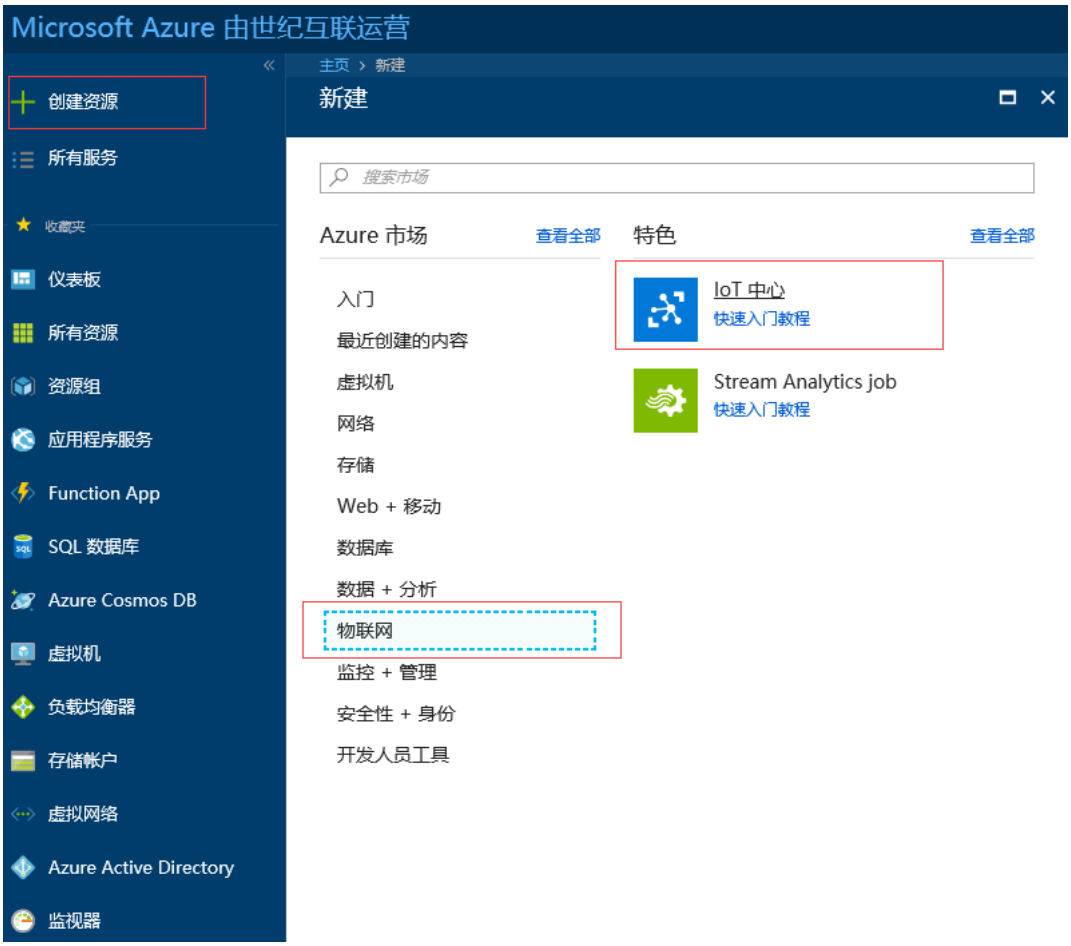

图 5.2: *创建物联网中心*

3、在 “IoT 中心” 窗格中，输入 IoT 中心的以下信息：

- **Subscription（订阅）**：选择需要将其用于创建此 IoT 中心的订阅。
- **Resource Group（资源组）**：创建用于托管 IoT 中心的资源组，或使用现有的资源组，在这个栏目中填入一个合适的名字就可以了。有关详细信息，请参阅使用资源组管理 Azure 资源。
- **Region（区域）**：选择最近的位置。
- **IoT Hub Name（物联网中心名称）**：创建 IoT 中心的名称，这个名称需要是唯一的。如果输入的名称可用，会显示一个绿色复选标记。

图 5.3: *填写 IoT 中心资料*

4、选择“下一步: **Size and scale（大小和规模）**”，以便继续创建 IoT 中心。

5、选择“**Pricing and scale tier（定价和缩放层）**”。就测试用途来说，请选择“**F1:Free tier（F1 - 免费）**”层（前提是此层在订阅上仍然可用）。有关详细信息，请参阅定价和缩放层。

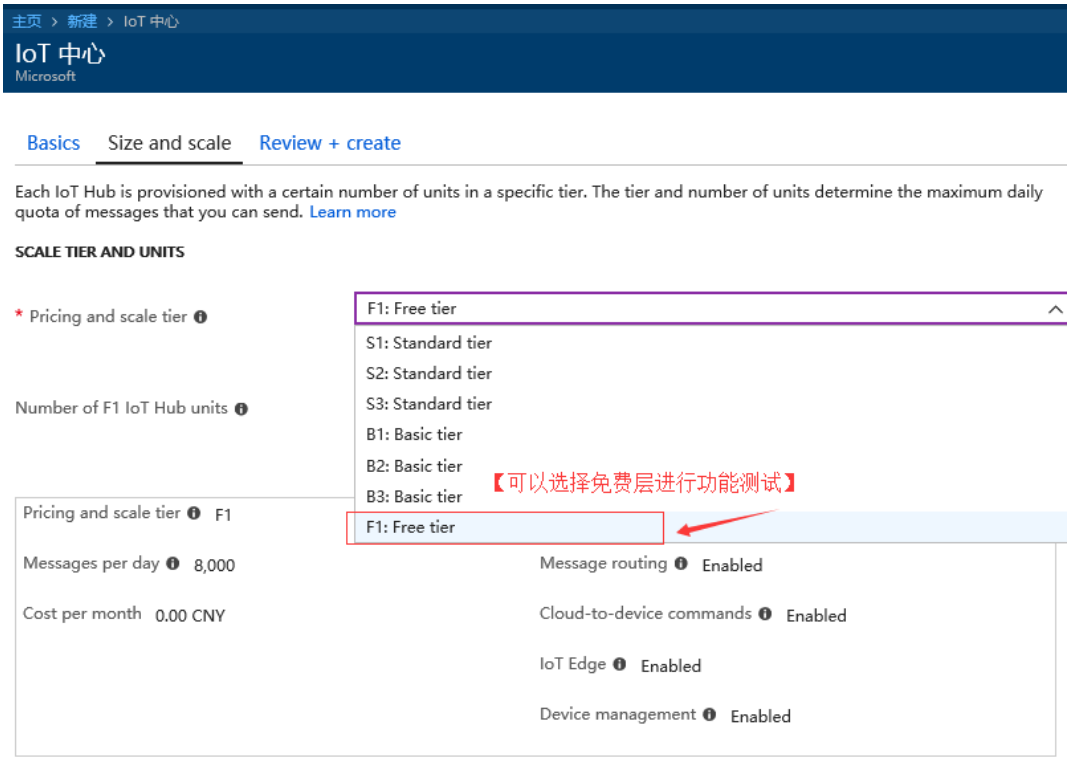

图 5.4: *选择功能层*

6、选择 “**Review + create（查看 + 创建）**”。

7、查看 IoT 中心信息，然后单击 “创建” 即可。创建 IoT 中心可能需要数分钟的时间。可在 “**通知**” 窗格中监视进度，创建成功后就可以进行下一步注册设备的操作了。

8、为了后续方便查找，可以手动将创建成功后的资源添加到**仪表盘**。

5.6 注册设备

要想运行设备端相关的示例，需要先将设备信息注册到 IoT 中心里，然后该设备才能连接到 IoT 中心。在本次示例中，可以使用 DeviceExplorer 工具来注册设备。

- 获得 IoT 中心的**共享访问密钥**（即 IoT 中心连接字符串）

1、IoT 中心创建完毕后，在设置栏目中，点击共享访问策略选项，可以打开 IoT 中心的访问权限设置。打开 iothubowner，在右侧弹出的属性框中获得 IoT 中心的共享访问密钥。

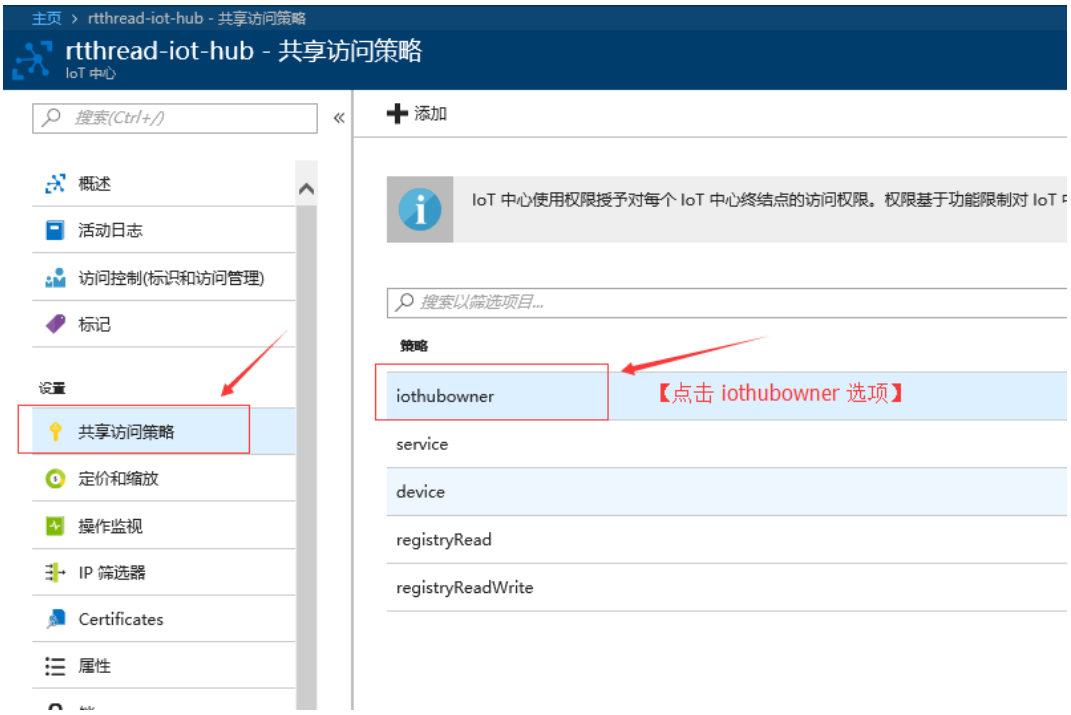

图 5.5: *查看物联网中心共享访问策略*

2、在右侧弹出的属性框中获取 IoT 中心连接字符串：

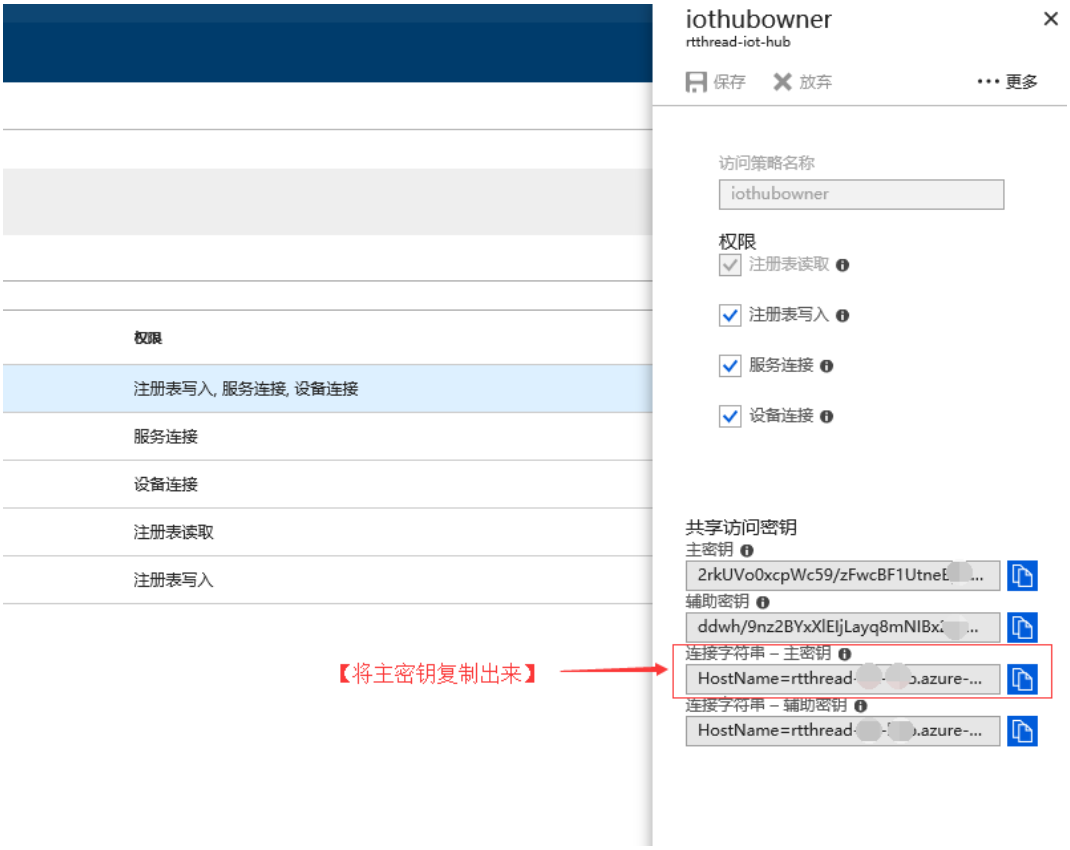

图 5.6: *复制访问主密钥*

- 创建设备

1、有了连接字符串后，我们便可以使用 DeviceExplorer 工具来创建设备，并测试 IoT 中心的功能了。打开测试工具，在配置选项中填入的连接字符串。点击 `update` 按钮更新本地连接 IoT 中心的配置，为下一步创建测试设备做准备。

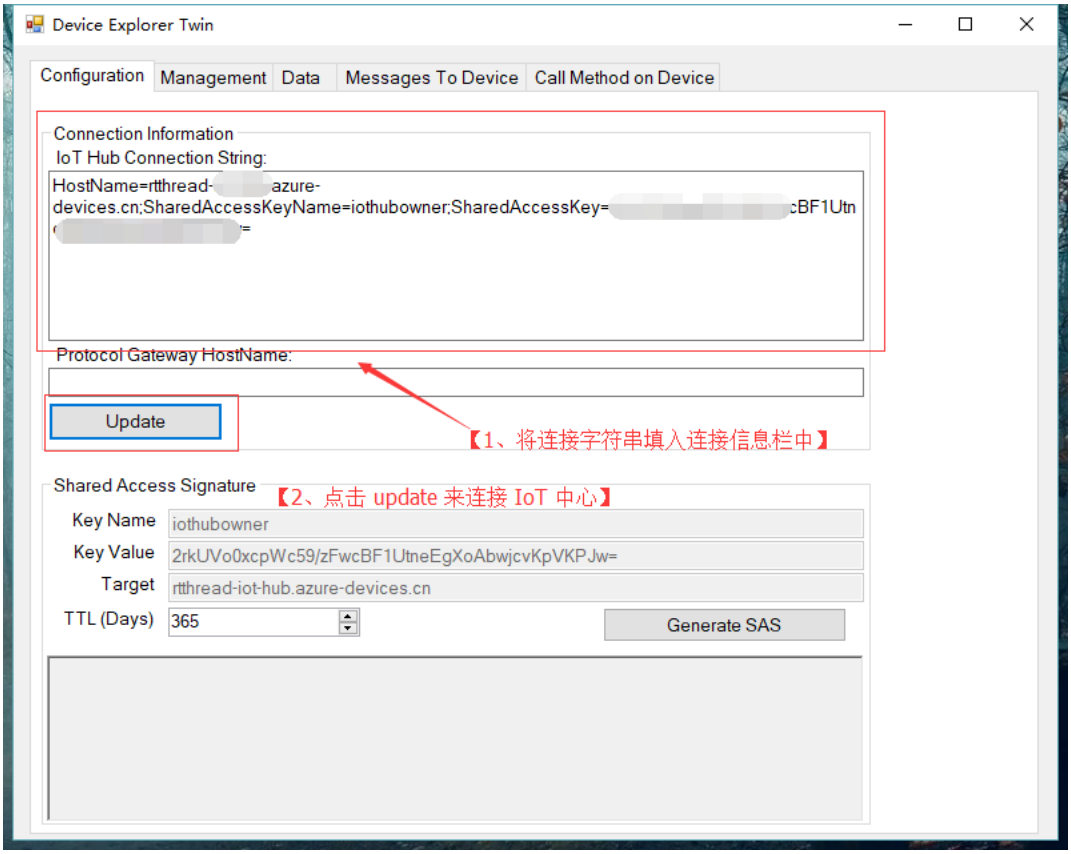

图 5.7: *配置 DeviceExplorer 工具*

2、打开 Management 选项栏，按照下图所示的步骤来创建测试设备。设备创建成功后，就可以运行设备的功能示例了。

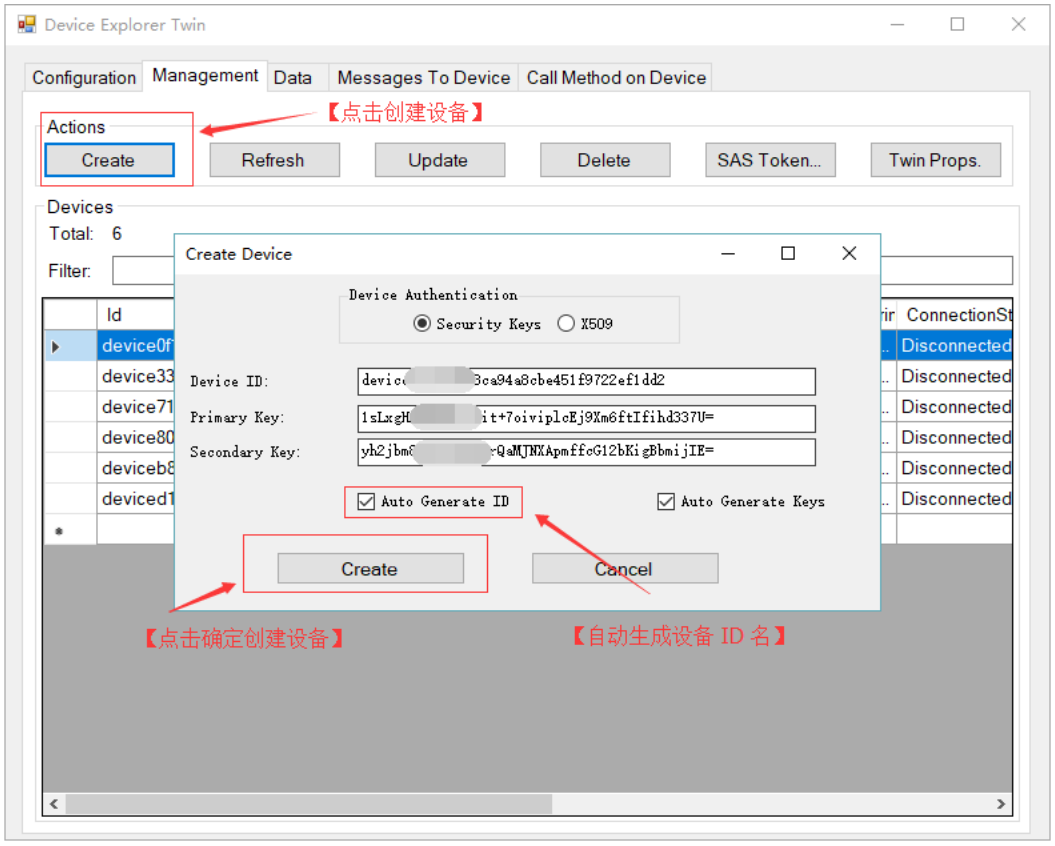
Device Explorer Twin
Configuration
Management
Data
Messages To Device
Call Method on Device
Actions
【点击创建设备】
Create
Refresh
Update
Delete
SAS Token...
Twin Props.
Devices
Total: 6
Filter:
Id
ConnectionSt
device0f
device33
device71
device80
deviceb8
deviced1
Disconnected
Create Device
Device Authentication
Security Keys
X509
Device ID:
Primary Key:
Secondary Key:
Auto Generate ID
Auto Generate Keys
Create
Cancel
【点击确定创建设备】
【自动生成设备 ID 名】

图 5.8: *创建设备*

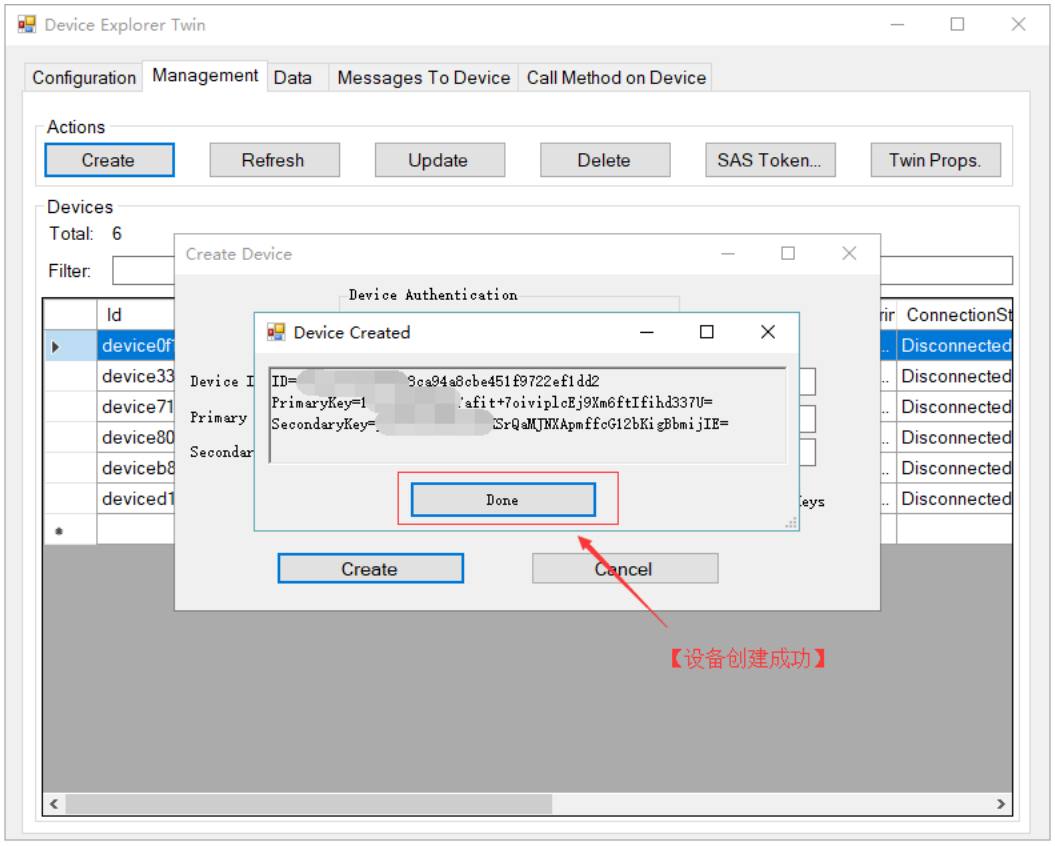

图 5.9: *创建设备成功*

5.7 功能示例一：设备发送遥测数据到物联网中心

5.7.1 示例文件

示例程序路径	说明
samples/iothub_ll_telemetry_sample.c	从设备端发送遥测数据到 Azure IoT 中心

5.7.2 云端监听设备数据

- 打开测试工具的 Data 选项栏，选择需要监听的设备，开始监听：

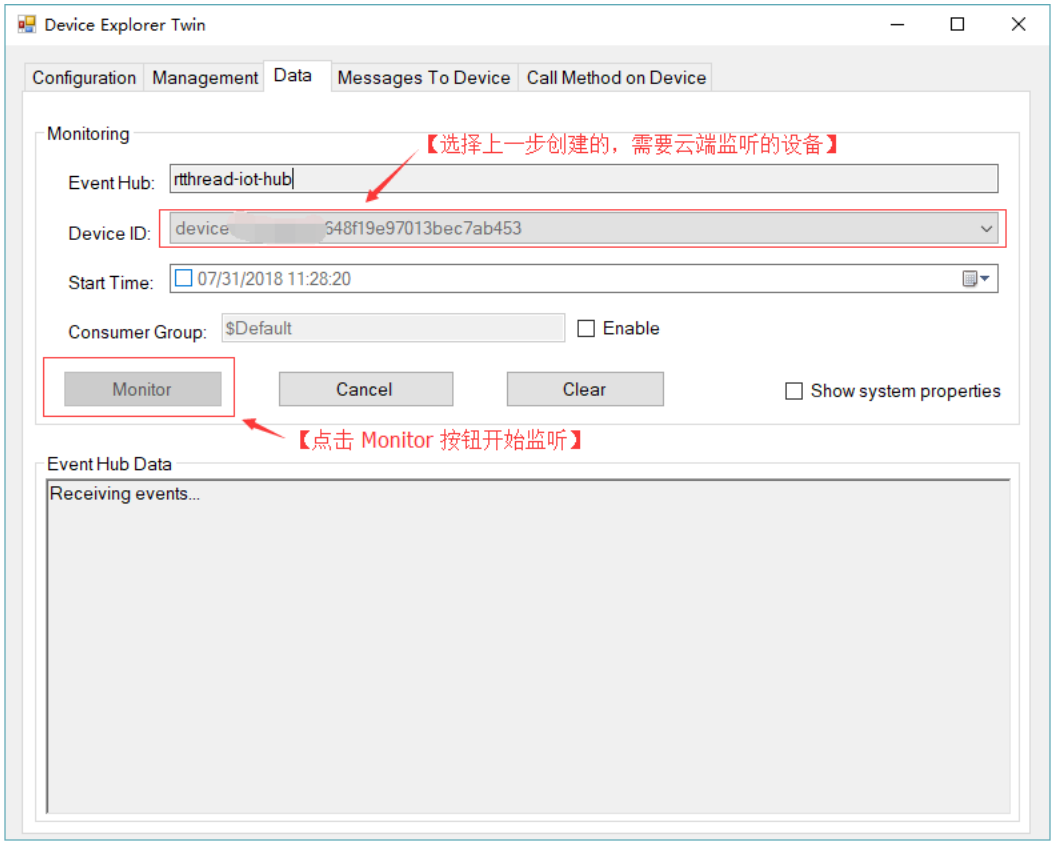

图 5.10: *监听设备遥测数据*

5.7.3 修改示例代码中的设备连接字符串

1、在运行测试示例前需要获取设备的连接字符串。

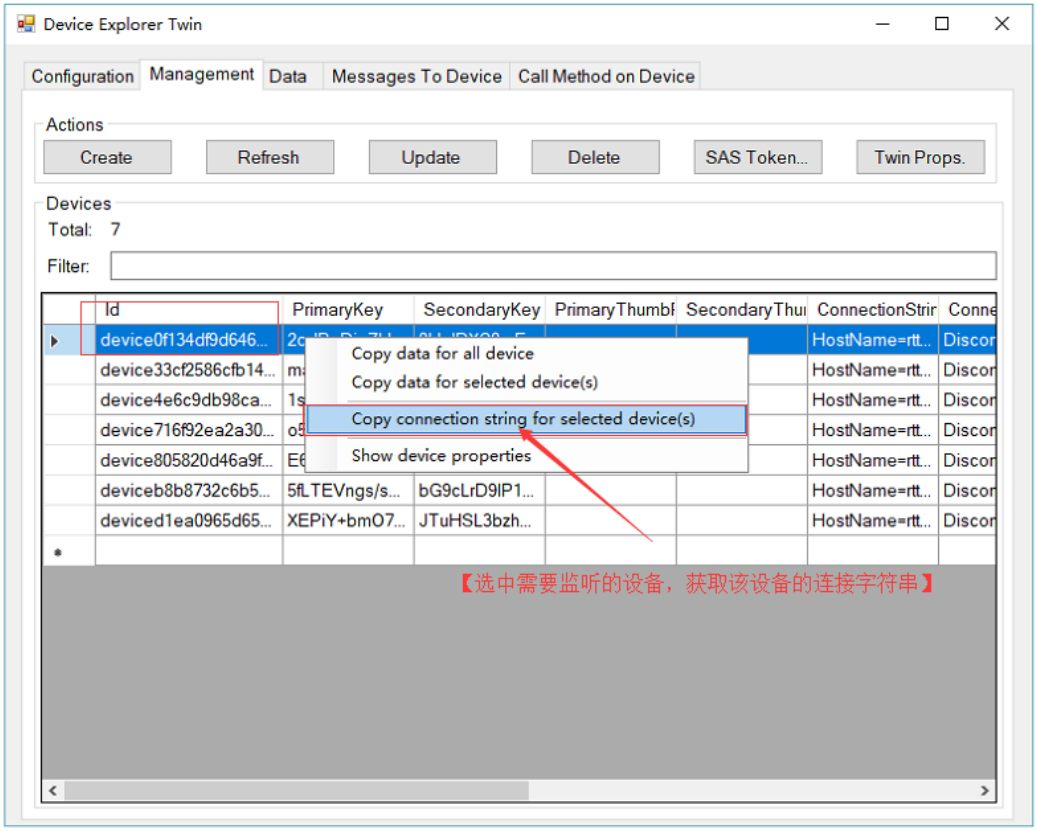

图 5.11: *获取设备连接字符串*

2、将连接字符串填入测试示例中的 connectionString 字符串中，重新编译程序，下载到开发板中。

图 5.12: *填入设备连接字符串*

5.7.4 运行示例程序

1、在 msh 中运行 azure_telemetry_sample 示例程序：

```
msh />azure_telemetry_sample
msh />
ntp init
Creating IoTHub Device handle
Sending message 1 to IoTHub
-> 11:46:58 CONNECT | VER: 4 | KEEPALIVE: 240 | FLAGS: 192 |
USERNAME:
xxxxxxxxxx.azuredevices.cn/devicexxxxxxx9d64648f19e97013bec7ab453
/?api-version=2017-xx-xx-preview&
DeviceClientType=iothubclient%2f1.2.8%20
(native%3b%20xxxxxxxx%3b%20xxxxxx) | PWD: XXXX | CLEAN: 0
<- 11:46:59 CONNACK | SESSION_PRESENT: true | RETURN_CODE: 0x0
The device client is connected to iothub
Sending message 2 to IoTHub
Sending message 3 to IoTHub
-> 11:47:03 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS:
    DELIVER_AT_LEAST_ONCE |
    TOPIC_NAME: devices/device0f134df9d64648f19e97013bec7ab453/messages/
        events
    /hello=RT-Thread | PACKET_ID: 2 | PAYLOAD_LEN: 12
-> 11:47:03 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS:
    DELIVER_AT_LEAST_ONCE |
    TOPIC_NAME: devices/device0f134df9d64648f19e97013bec7ab453/messages/
        events
    /hello=RT-Thread | PACKET_ID: 3 | PAYLOAD_LEN: 12
-> 11:47:03 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS:
    DELIVER_AT_LEAST_ONCE |
    TOPIC_NAME: devices/device0f134df9d64648f19e97013bec7ab453/messages/
        events
    /hello=RT-Thread | PACKET_ID: 4 | PAYLOAD_LEN: 12
<- 11:47:04 PUBACK | PACKET_ID: 2
Confirmation callback received for message 1 with result
    IOTHUB_CLIENT_CONFIRMATION_OK
<- 11:47:04 PUBACK | PACKET_ID: 3
Confirmation callback received for message 2 with result
    IOTHUB_CLIENT_CONFIRMATION_OK
<- 11:47:04 PUBACK | PACKET_ID: 4
Confirmation callback received for message 3 with result
    IOTHUB_CLIENT_CONFIRMATION_OK
Sending message 4 to IoTHub
-> 11:47:06 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS:
```

```
    DELIVER_AT_LEAST_ONCE |
    TOPIC_NAME: devices/device0f134df9d64648f19e97013bec7ab453/messages/
        events
    /hello=RT-Thread | PACKET_ID: 5 | PAYLOAD_LEN: 12
<- 11:47:07 PUBACK | PACKET_ID: 5
Confirmation callback received for message 4 with result
    IOTHUB_CLIENT_CONFIRMATION_OK
Sending message 5 to IoTHub
-> 11:47:09 PUBLISH | IS_DUP: false | RETAIN: 0 | QOS:
    DELIVER_AT_LEAST_ONCE |
    TOPIC_NAME: devices/device0f134df9d64648f19e97013bec7ab453/messages/
        events
    /hello=RT-Thread | PACKET_ID: 6 | PAYLOAD_LEN: 12
<- 11:47:10 PUBACK | PACKET_ID: 6
Confirmation callback received for message 5 with result
    IOTHUB_CLIENT_CONFIRMATION_OK
-> 11:47:14 DISCONNECT
Error: Time:Tue Jul 31 11:47:14 2018 File:packages\azure\azure-port\pal\
    src\
socketio_berkeley.c Func:socketio_send Line:853
Failure: socket state is not opened.
The device client has been disconnected
Azure Sample Exit
```

2、此时可在 DeviceExplorer 工具的 Data 栏查看设备发到云端的遥测数据：

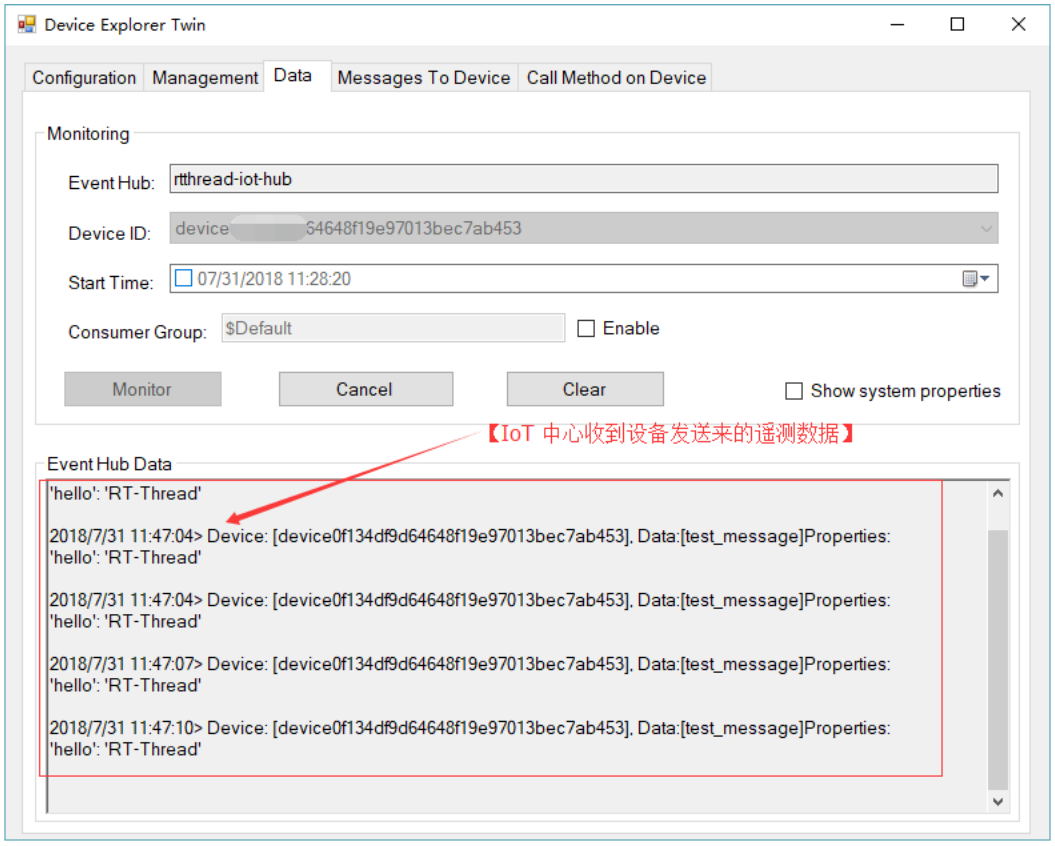

图 5.13: *收到遥测数据*

示例运行成功，在 DeviceExplorer 工具中看到了设备发送到物联网中心的 5 条遥测数据。

5.8 功能示例二：设备监听云端下发的数据

5.8.1 示例文件

示例程序路径	说明
samples/iothub_ll_c2d_sample.c	在设备端监听 Azure IoT 中心下发的数据。

5.8.2 修改示例代码中的设备连接字符串

- 与上面的示例相同，本示例程序也需要填写正确的设备连接字符串，修改完毕后重新编译程序，下载到开发板中即可。修改内容如下所示：

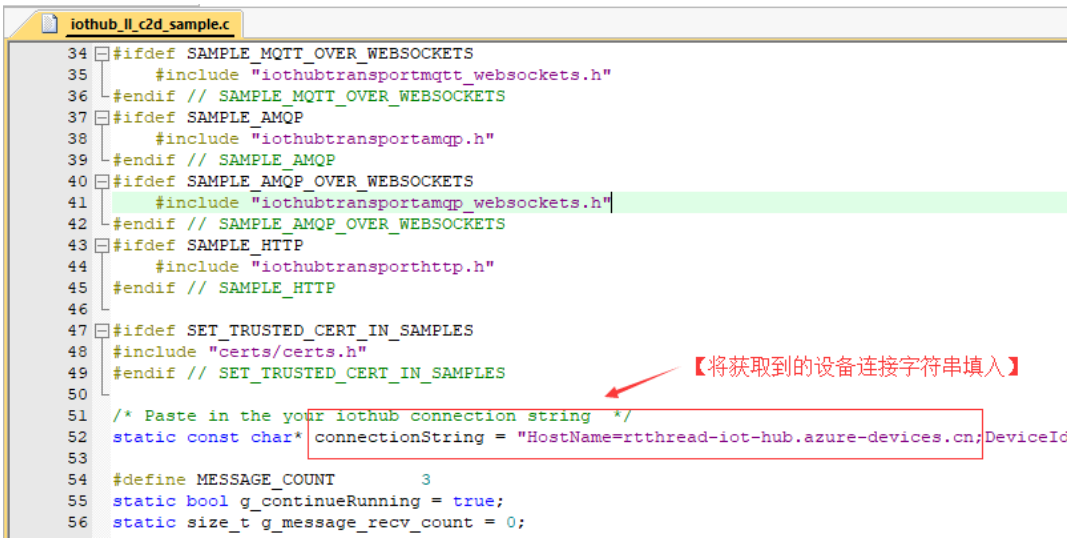

图 5.14: *修改设备连接字符串*

5.8.3 设备端运行示例程序

- 在 msh 中运行 azure_telemetry_sample 示例程序，示例程序运行后设备将会等待并接收云端下发的数据：

```
msh />azure_c2d_sample
msh />
ntp init
Creating IoTHub Device handle          # 等待 IoT 中心的下发数据
Waiting for message to be sent to device (will quit after 3 messages)
```

5.8.4 服务器下发数据给设备

1、打开 DeviceExplorer 工具的 Messages To Device 栏向指定设备发送数据：

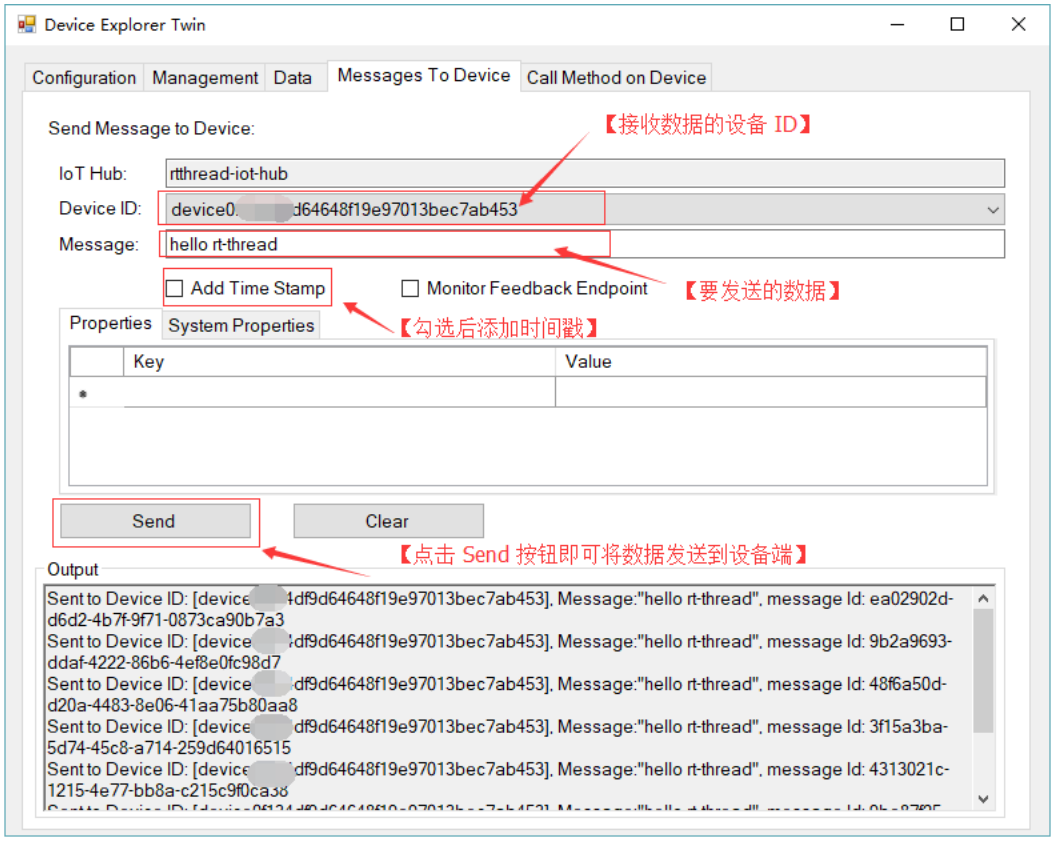

图 5.15: *服务器下发数据给设备*

2、此时在设备端查看从 IoT 中心下发给设备的数据：

```
msh />azure_c2d_sample
msh />
ntp init
Creating IoTHub Device handle
Waiting for message to be sent to device (will quit after 3 messages)
Received Binary message                                  # 收到二进制数据
Message ID: ea02902d-d6d2-4b7f-9f71-0873ca90b7a3
 Correlation ID: <unavailable>
 Data: <<<hello rt-thread>>> & Size=15
Received Binary message
Message ID: 9b2a9693-ddaf-4222-86b6-4ef8e0fc98d7
 Correlation ID: <unavailable>
 Data: <<<hello rt-thread>>> & Size=15
Received Binary message
Message ID: 48f6a50d-d20a-4483-8e06-41aa75b80aa8
 Correlation ID: <unavailable>
 Data: <<<hello rt-thread>>> & Size=15
Received Binary message
```

```
Message ID: 3f15a3ba-5d74-45c8-a714-259d64016515
 Correlation ID: <unavailable>
 Data: <<<hello rt-thread>>> & Size=15
Received Binary message
Message ID: 4313021c-1215-4e77-bb8a-c215c9f0ca38
 Correlation ID: <unavailable>
 Data: <<<hello rt-thread>>> & Size=15
Received Binary message
Message ID: 9be87f25-2a6f-46b5-a413-0fb2f93f85d6
 Correlation ID: <unavailable>
 Data: <<<hello rt-thread>>> & Size=15
Error: Time:Tue Jul 31 13:54:14 2018
File:packages\azure\azure-port\pal\src\socketio_berkeley.c
Func:socketio_send Line:853 Failure: socket state is not opened.
Azure Sample Exit     #收到一定数量的下发数据，功能示例自动退出
```