Alibaba Cloud
物#网平台

ユーザーガイド

Document Version20200413

目次

1 TSL

1.1 TSL (Thing Specification Language) の説明

TSL (Thing Specification Language) は、物理エンティティをデジタル化し、クラウド内にエンティティを構築するデータモデルです。 IoT Platform では、TSLモデルはプロダクト機能のセットを指します。 プロダクトに機能が定義されると、プロダクトの TSL モデルが自動生成されます。 TSLモデルは、プロダクトが何であるか、何ができるか、およびどんなサービスを提供できるかを記述します。

TSL モデルは JSON 形式のファイルです。 TSL ファイルは、センサー、車載デバイス、建物、工場などの物理エンティティをデジタル化し表現したものです。 TSL ファイルは、エンティティをプロパティ (エンティティが何であるか)、サービス (エンティティができること)、およびイベント (エンティティが報告するイベント情報) の 3 つの側面から記述します。 これら 3 つの側面を定義することで、プロダクトの機能を定義します。

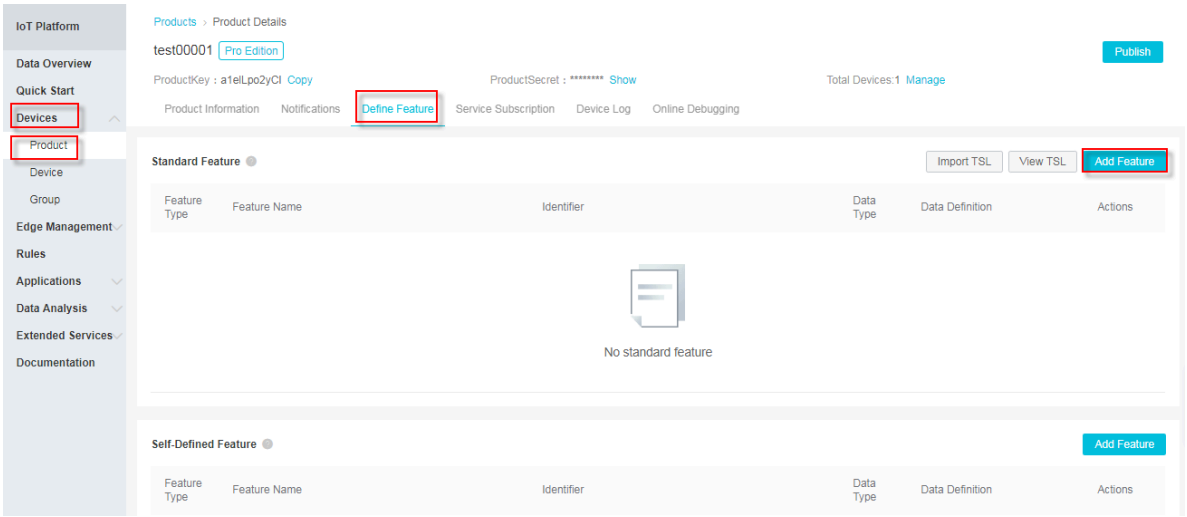
プロダクトの機能タイプは、プロパティ、サービス、およびイベントです。 この 3 種類の機能は、コンソールで定義できます。

機能タイプ	説明
プロパティ	デバイスの動作状態を記述します (例: 環境モニタリング装置によって読み取られた現在の温度)。 プロパティは、GET および SET のリクエストメソッドをサポートします。 アプリケーションシステムは、プロパティを取得および設定するリクエストを送信できます。
サービス	公開され、外部からリクエストで呼び出せる、デバイスの機能またはメソッド。 入力パラメーターと出力パラメーターを設定できます。 プロパティと比較すると、サービスは、特定のタスクなど、より複雑なビジネスロジックの実行を指示できます。
イベント	操作中に生成されるイベントです。 イベントには通常、対処または注意を要求する通知が含まれ、複数の出力パラメーターが含まれる場合があります。 たとえば、イベントには、タスクの完了、システム障害、または温度警告に関する通知が考えられます。 イベントのサブスクライブまたはプッシュも可能です。

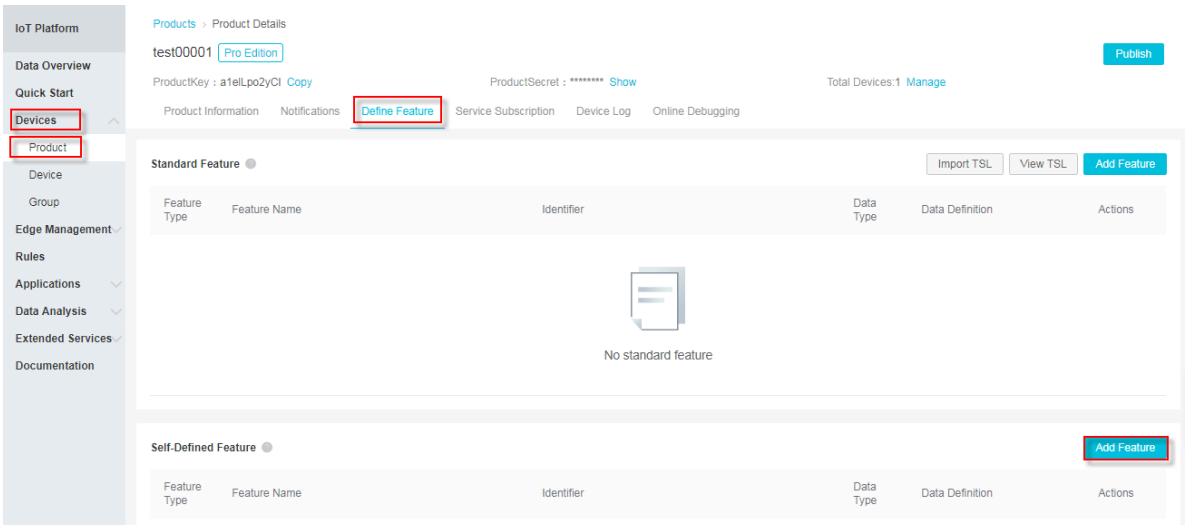
1.2 機能の定義

このトピックでは、IoT Platform コンソールで機能を定義する方法を説明します。

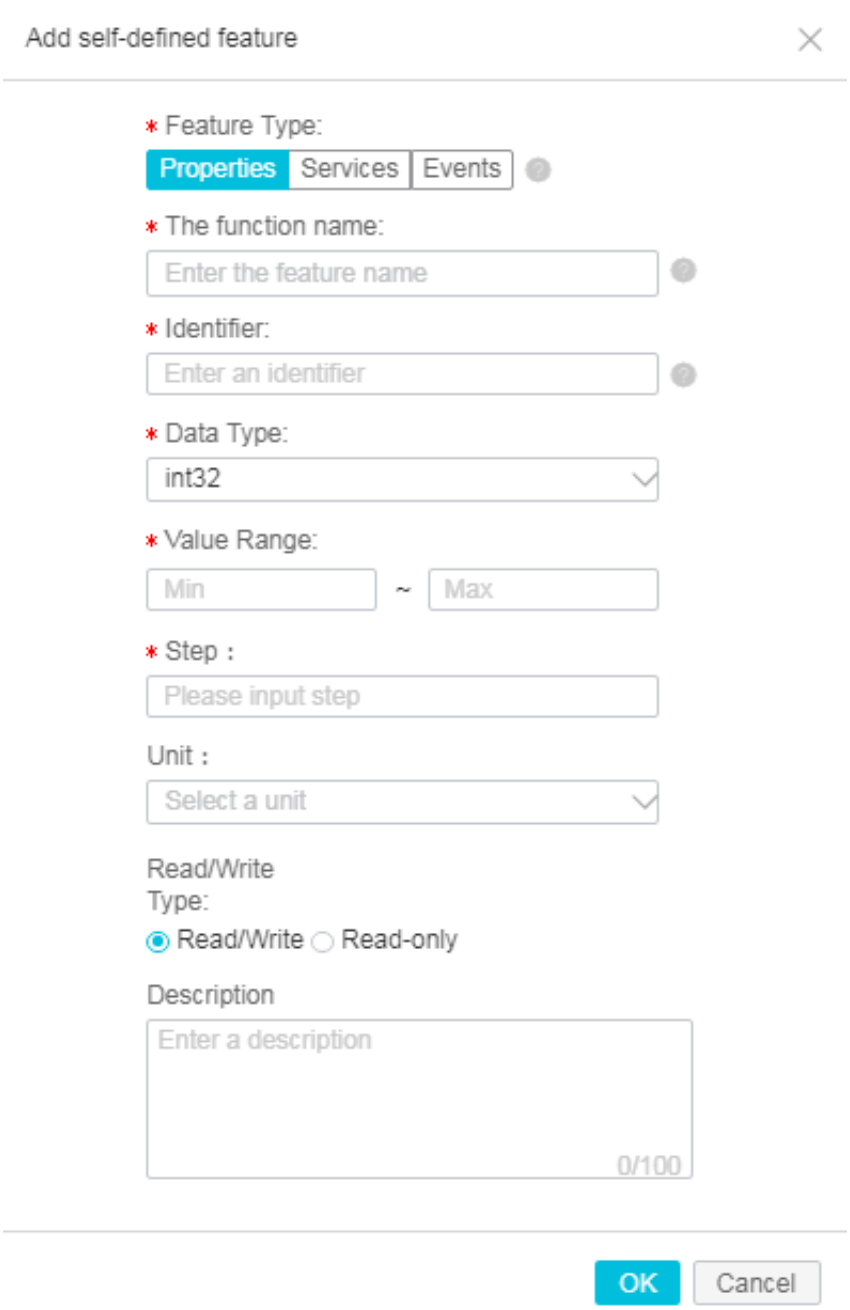
1. IoT Platform コンソールにログインします。
2. 左側のナビゲーションウィンドウで、**[デバイス]** > **[プロダクト]** をクリックします。
3. **[プロダクト]** ページで、機能を定義するプロダクトを検索し、**[表示]** をクリックします。
4. プロダクト詳細ページで、**[機能の定義]** をクリックします。
5. 標準機能を追加します。 **[標準機能]** に対応する **[機能の追加]** ボタンをクリックし、ダイアログボックスで、プロダクトに使用できる事前定義機能を選択します。



6. 自己定義機能を追加します。 **[自己定義機能]** に対応する **[機能の追加]** ボタンをクリックし、プロダクトのカスタマイズ機能を定義します。 プロダクトに対し、プロパティ、サービス、イベントを定義できます。



- プロパティを定義します。 **[自己定義機能の追加]** ダイアログボックスで、機能タイプとして **[プロパティ]** を選択します。



プロパティのパラメーターを下記の一覧表に示します。

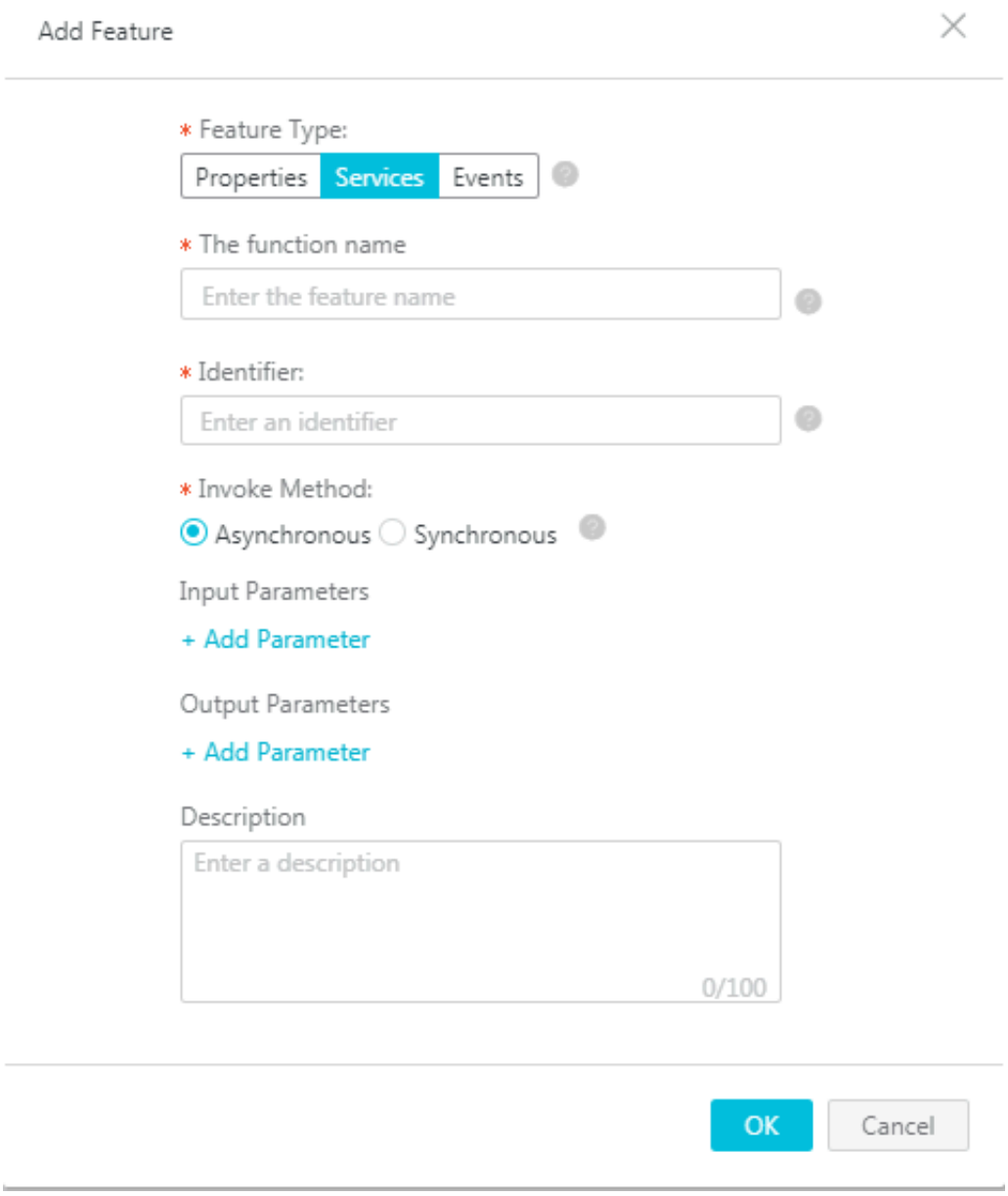
パラメーター	説明
機能名	プロパティ名 (例: 消費電力) 機能名はプロダクト内で一意である必要があります。 機能名は、漢字、英数字で始まる必要があり、漢字、英数字、ダッシュ (-)、アンダースコア (_) を含めることができます。長さは 30 文字を超えることはできません。 プロダクト作成中に機能テンプレートを含むカテゴリを選択していた場合、標準機能ライブラリから選択可能な標準プロパティが表示されます。 注： ゲートウェイ接続プロトコルが Modbus の場合、標準プロパティはサポートされません。 代わりに、プロパティを手動で定義する必要があります。
識別子	プロパティを識別します。 プロダクト内で一意である必要があります。 Alink JSON の識別子パラメーターで、デバイスがこのプロパティのデータを報告したときキーとして使用されます。 特に、IoT Platform はこのパラメーターを使用して、データを受信するかどうかを確認および決定します。 識別子は英数字とアンダースコア (_) を含めることができ、長さは 50 文字を超えることはできません。 (例: PowerConsumption) 注： set、get、post、time、value は、システムパラメーター名のため、識別子として使用することはできません。

パラメーター	説明
データ型	- int32: 32 ビット整数。 "int32" を選択した場合、値の範囲、ステップ、および単位を定義する必要があります。 - float: フロート。 "float" を選択した場合、値の範囲、ステップ、および単位を定義する必要があります。 - double: ダブルフロート。 "double" を選択した場合、値の範囲、ステップ、および単位を定義する必要があります。 - enum: 列挙型。 列挙項目を値と説明で指定する必要があります。 (例: 1: 加熱モード、2: 冷却モード) - bool: ブール型。 ブール値を指定する必要があります。 値は 0 と 1 です。 たとえば、無効を 0、有効を 1 で示すことができます。 - text: テキスト文字列。 データ長を特定する必要があります。 最大値は 2048 バイトです。 - date: タイムスタンプ。 ミリ秒単位の UTC タイムスタンプ文字列。 - struct: JSON 構造。 JSON 構造を定義し、新しいパラメーターを追加します。 たとえば、ランプの色を赤、緑、青の 3 つのパラメーターで構成される構造として定義できます。 構造の入れ子はサポートされていません。 - array: 配列。 配列内の要素には、int32、float、double、text、および struct からデータ型を選択する必要があります。 1 つの配列内で、要素のデータ型は同じにし、配列の長さは 128 要素を超えないようにします。  注 : ゲートウェイ接続プロトコルが Modbus の場合は、このパラメーターを設定しません。
ステップ	プロパティ、イベント、およびサービスの入出力パラメーター値の変更における最小粒度。 データ型が int32、float、または double の場合、ステップが必要です。
単位	[なし] または適切な単位を選択できます。
読み取り/書き込みタイプ	- 読み取り/書き込み: 読み取りまたは書き込みのリクエストに対して、GET メソッドと SET メソッドがサポートされています。 - 読み取り専用: 読み取り専用リクエストに対しては、GET メソッドのみがサポートされています。  注 : ゲートウェイ接続プロトコルが Modbus の場合は、このパラメーターを設定する必要はありません。

パラメーター	説明
説明	プロパティに関する説明や備考を入力します。 100 文字まで入力できます。
詳細情報  注：ゲートウェイ接続プロトコルは Modbus です。	- 操作タイプ: 入力ステータス (読み取り専用)、コイルステータス (読み書き)、保持レジスタ (読み書き)、入力レジスタ (読み取り専用) の中から操作タイプを選択できます。 - レジスタアドレス: 0x で始まる 16 進数のアドレスを入力します (例: 0xFE)。 範囲は、0x0 ～ 0xFFFF です。 - 元のデータ型: int16、uint16、int32、uint32、int64、uint64、float、double、string、bool、およびカスタマイズされたデータ (RAW データ) など、複数のデータ型がサポートされています。 - レジスタ数: 操作タイプが入力ステータス (読み取り専用) およびコイルステータス (読み書き) に設定されている場合、レジスタの数値範囲は 1 ～ 2000 です。 操作タイプが保持レジスタ (読み書き) および入力レジスタ (読み取り専用) の場合、レジスタの数値範囲は 1 ～ 125 です。 - レジスタの上位バイトと下位バイトの切り替え: レジスタ内の 16 ビットデータの最初の 8 ビットと最後の 8 ビットを入れ替えます。 - レジスタビットシーケンスの切り替え: 元の 32 ビットデータのビットを交換します。 - ズーム係数: ズーム係数はデフォルトで 1 に設定されています。 負の数に設定することができますが、0 に設定することはできません。 - 収集間隔: データ収集の時間間隔。 ミリ秒単位で、値を 10 より小さくすることはできません。 - データレポート: データレポートのトリガーです。 [特定の時点] または [レポートの変更] のいずれかになります。

パラメーター	説明
詳細情報 注：ゲートウェイ接続プロトコルは OPC UA です。	ノード名です。プロパティ内で一意である必要があります。

- サービスを定義します。**[自己定義機能の追加]** ダイアログボックスで、機能タイプとして **[サービス]** を選択します。



サービスのパラメーターは次のとおりです。

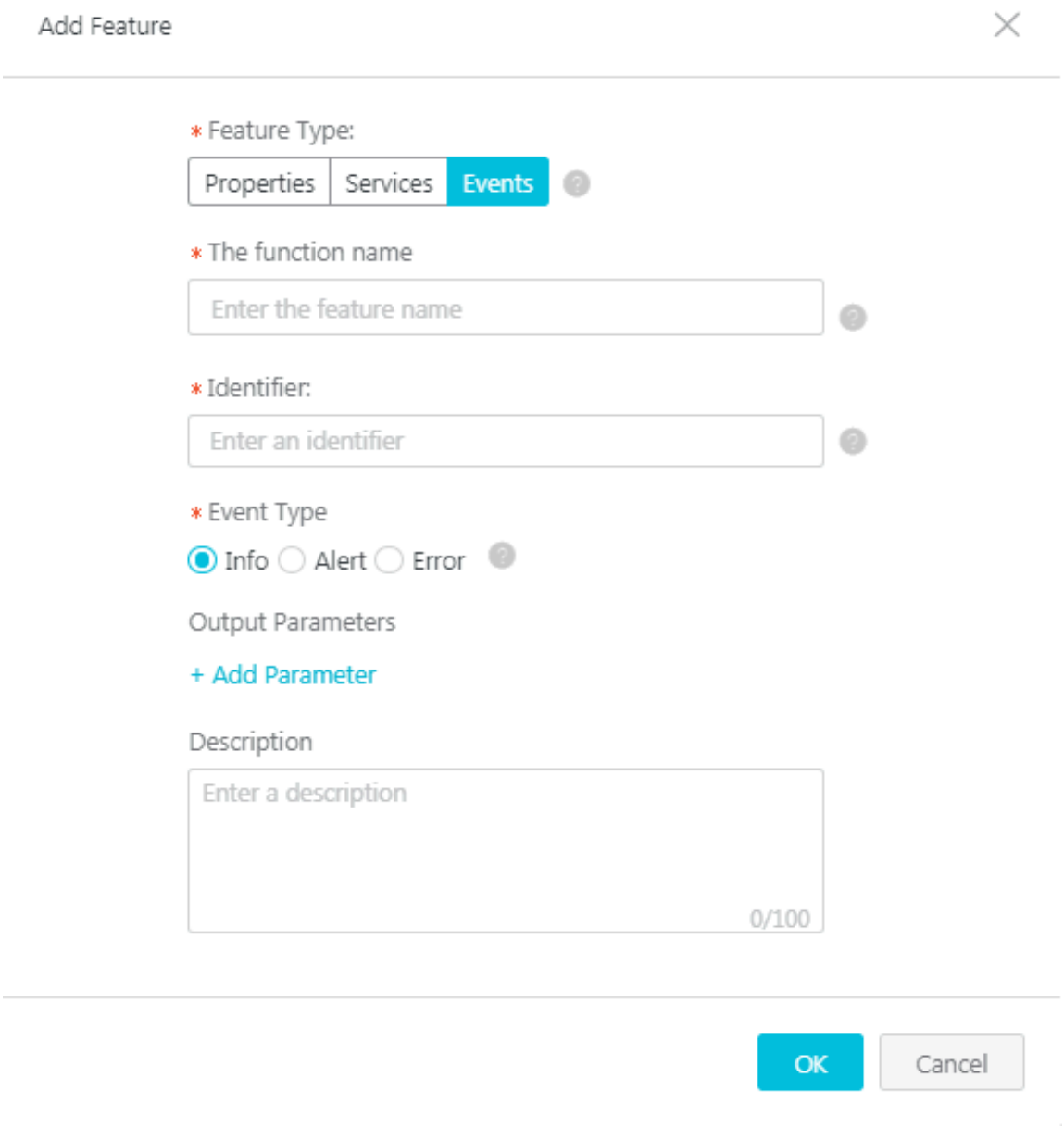
パラメーター	説明
機能名	サービス名です。 機能名は、英数字または漢字で始める必要があります。 英数字、漢字、ダッシュ (-)、アンダースコア (_) を含めることができ、長さは 30 文字を超えることはできません。 プロダクト作成中に機能テンプレートを含むカテゴリを選択していた場合、システムは標準機能ライブラリから選択可能な標準プロパティを表示します。  注： ゲートウェイ接続プロトコルが Modbus の場合、プロダクトにカスタマイズサービスを定義することはできません。
識別子	サービスを識別します。 プロダクト内で一意である必要があります。 Alink JSON TSL 内の [識別子] パラメーターです。 このサービスが呼び出されると、キーとして使用されます。 識別子は、英字、数字、アンダースコア (_) を含めることができ、長さは 30 文字を超えることはできません。  注： set、get、post、time、value は、システムパラメーター名のため、識別子として使用することはできません。
呼び出し方式	- 非同期: 非同期呼び出しの場合、IoT Platform はリクエストが送信された直後に結果を返します。デバイスからの応答は待ちません。 - 同期: 同期呼び出しの場合、クラウドはデバイスからの応答を待ちます。 応答が受信されない場合、呼び出しはタイムアウトします。

パラメーター	説明
入力パラメーター	サービスに入力パラメーターを設定します (省略可)。 **[**パラメーターの追加**]** をクリックし、表示されるダイアログボックスで入力パラメーターを追加します。 ゲートウェイ接続プロトコルが "OPC UA" の場合、パラメーターの順序をマークするために使用されるパラメーターインデックスを設定する必要があります。 注 **:** - プロパティを入力パラメーターとして選択することも、入力パラメーターを定義することもできます。 たとえば、**Sprinkling_Interval** プロパティと **Sprinkling_Amount** プロパティを **Automatic_Sprinkler** サービス機能の入力パラメーターとして指定します。 その場合、Automatic_Sprinkle が呼び出されると、スプリンクラーは散水間隔 (Sprinkling_Interval) と散水量 (Sprinkling_Amount) に従って自動的に散水を開始します。 - set、get、post、time、value は、システムパラメーター名のため、入力パラメーターの識別子として使用することはできません。 - 1 つのサービスに対して最大 20 個の入力パラメーターを追加します。

パラメーター	説明
出力パラメーター	サービスに出力パラメーターを設定します (省略可)。 **[**パラメーターの追加**]** をクリックし、表示されるダイアログボックスで出力パラメーターを追加します。 ゲートウェイ接続プロトコルが OPC UA の場合、パラメーターの順序をマークするために使用されるパラメーターインデックスを設定する必要があります。  注 **:** - プロパティを出力パラメーターとして選択することも、出力パラメーターを定義することもできます。 たとえば、出力パラメーターとして **SoilHumidity** を指定します。 その場合、Automatic Sprinkler サービスが呼び出されると、IoT Platform は土壌湿度に関するデータを返します。 - set、get、post、time、value は、システムパラメーター名のため、入力パラメーターの識別子として使用することはできません。 - 1 つのサービスに対して最大 20 個の出力パラメーターを追加できます。
詳細情報  注 **:** ゲートウェイ接続プロトコルは OPC UA です。	ノード名です。サービス内で一意である必要があります。

パラメーター	説明
説明	サービスについての説明や備考を入力します。 100 文字まで入力できます。

- イベントを定義します。 **[自己定義機能の追加]** ダイアログボックスで、機能タイプとして **[イベント]** をクリックします。



イベントのパラメーターは次のとおりです。

パラメーター	説明
機能名	イベント名です。 機能名は、漢字、英字、または数字で始める必要があります。漢字、英字、数字、ダッシュ (-)、および下線 (_) を含めることができ、長さは 30 文字を超えることはできません。  注： ゲートウェイ接続プロトコルが Modbus の場合は、イベントを定義できません。
識別子	イベントを識別します。 プロダクト内で一意である必要があります。 Alink JSON TSL 内の【識別子】パラメーターで、デバイスがこのイベントのデータを報告するときにキーとして使用されます (例: ErrorCode)。  注： set、get、post、time、value は、システムパラメーター名のため、入力パラメーターの識別子として使用することはできません。
イベントタイプ	- Info: デバイスによって報告される一般的な通知 (特定のタスクの完了など) を表します。 - Alert: 予期しないまたは異常なイベントが発生したときにデバイスによって報告される警告を示します。 優先度は高めです。 イベントタイプに応じて、ロジックの処理や分析を実行できます。 - Error: 予期しないまたは異常なイベントが発生したときにデバイスによって報告されるエラーを示します。 優先度は高めです。 イベントタイプに応じて、ロジックの処理や分析を実行できます。

パラメーター	説明
出力パラメーター	イベントの出力パラメーターです。 **[パラメーターの追加]** をクリックし、表示されるダイアログボックスで出力パラメーターを追加します。プロパティを出力パラメーターとして選択することも、出力パラメーターを定義することもできます。 たとえば、出力パラメーターとして **Voltage** を指定します。 この場合、デバイスは、故障診断のために現在の電圧値とともにエラーを報告します。 ゲートウェイ接続プロトコルが OPC UA の場合、パラメーターの順序をマークするために使用されるパラメーターインデックスを設定する必要があります。 注： - set、get、post、time、value は、システムパラメーター名のため、入力パラメーターの識別子として使用することはできません。 - 1 つのイベントに対して最大 20 個の出力パラメーターを追加できます。
詳細情報 注：ゲートウェイ接続プロトコルは OPC UA です。	ノード名です。イベント内で一意である必要があります。
説明	イベントについての説明または備考を入力します。 100 文字まで入力できます。

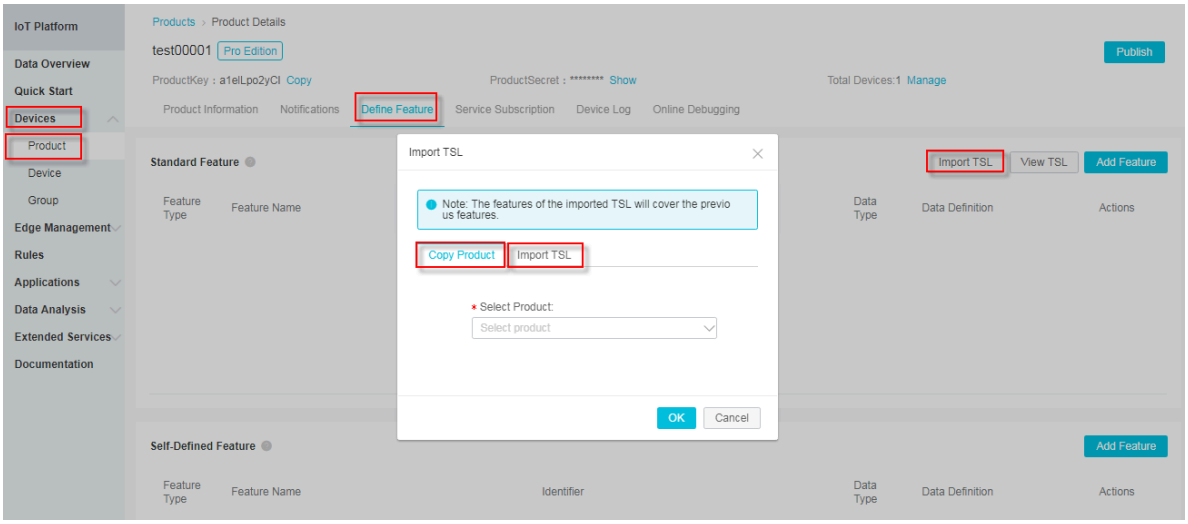
1.3 TSL (Thing Specification Language) のインポート

ここでは、プロダクトに既存の TSL をインポートする方法を紹介します。

1. IoT Platform コンソールにログインします。
2. 左側のナビゲーションウィンドウで、**[デバイス]** > **[プロダクト]** をクリックします。
3. **[プロダクト]** ページで、TSL をインポートするプロダクトを検索し、**[表示]** をクリックします。

4. **[機能の定義] > [TSL のインポート]** をクリックします。

> 注：
>
> - プロダクトに新しい TSL をインポートすると、以前プロダクトに定義した機能は上書きされます。 したがって、この機能は慎重に使用する必要があります。
> - ゲートウェイ接続プロトコルが Modbus として定義されているプロダクトに TSL をインポートすることはできません。



TSL をインポートする方法は 2 つあります。

- プロダクトのコピー: 別のプロダクトから TSL をコピーします。 既存のプロダクトをクリックし、**[OK]** をクリックして、選択されたプロダクトの TSL を対象プロダクトにインポートします。

  一部の機能を変更する場合は、[機能の定義] タブページ上の、機能に対応する **[編集]** をクリックします。
- **TSL** のインポート: 自己定義 TSL スクリプトを編集ボックスに貼り付けて、**[OK]** をクリックします。

2 データ解析

2.1 データ解析の概要

IoT Platform で定義されている標準データ形式は Alink JSON です。 リソースの限られたローエンドデバイスや高いネットワークスループットを必要とするデバイスの場合、JSON データを使用して IoT Platform と通信するのは適していません。 IoT Platform には、生データを直接送信することができます。 IoT Platform のデータ解析機能により、ユーザーから送信されたスクリプトに基づいて、デバイス固有の形式と JSON 形式の間でデータが変換されます。

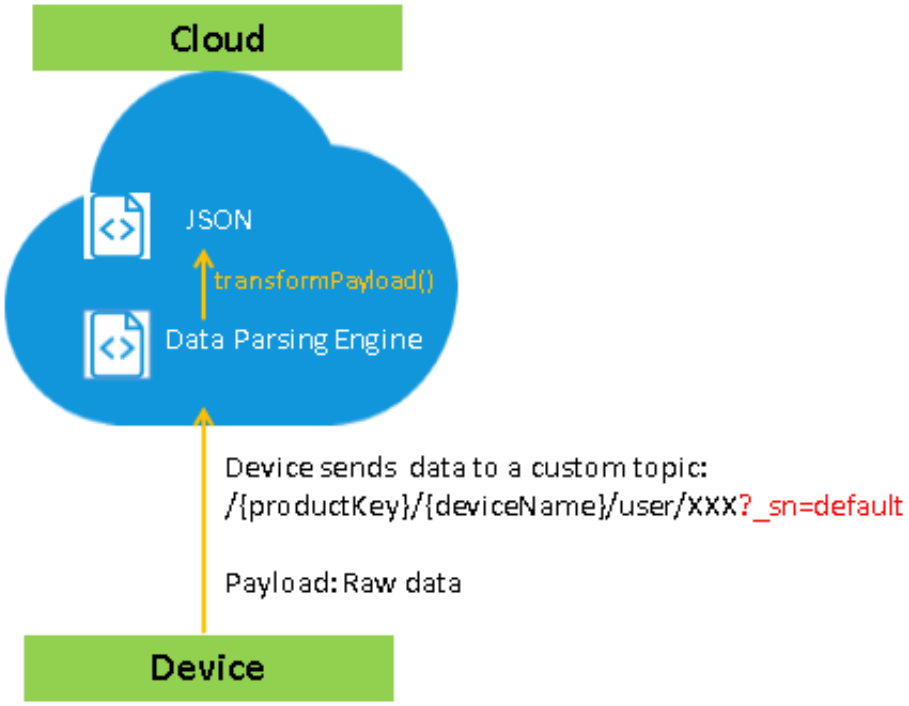
次の 2 種類のデータを解析できます。

- カスタム Topic 用に定義されたカスタムデータ。 デバイスがカスタム Topic を介してデータをレポートする場合、ペイロードは JSON 形式に解析されます。
- アップストリームとダウンストリームの TSL データ。 IoT Platform は、デバイスからレポートされたカスタム TSL データを Alink JSON データに解析し、Alink JSON データをデバイスに送信する前にカスタム形式に解析します。

カスタム Topic データの解析

デバイスがカスタム Topic を介してデータをレポートする場合、その Topic には解析フラグ (?_sn=default) が含まれている必要があります。 IoT Platform はデータを受信した後、コンソールから送信されたデータ解析スクリプトを呼び出して、カスタム形式のデータを JSON データに変換します。 次に、JSON データを後続のプロセスに転送します。

以下は、データ解析のフローチャートです。



カスタム Topic データを解析するスクリプトの記述方法については、以下をご参照ください。

概要

JavaScript 例

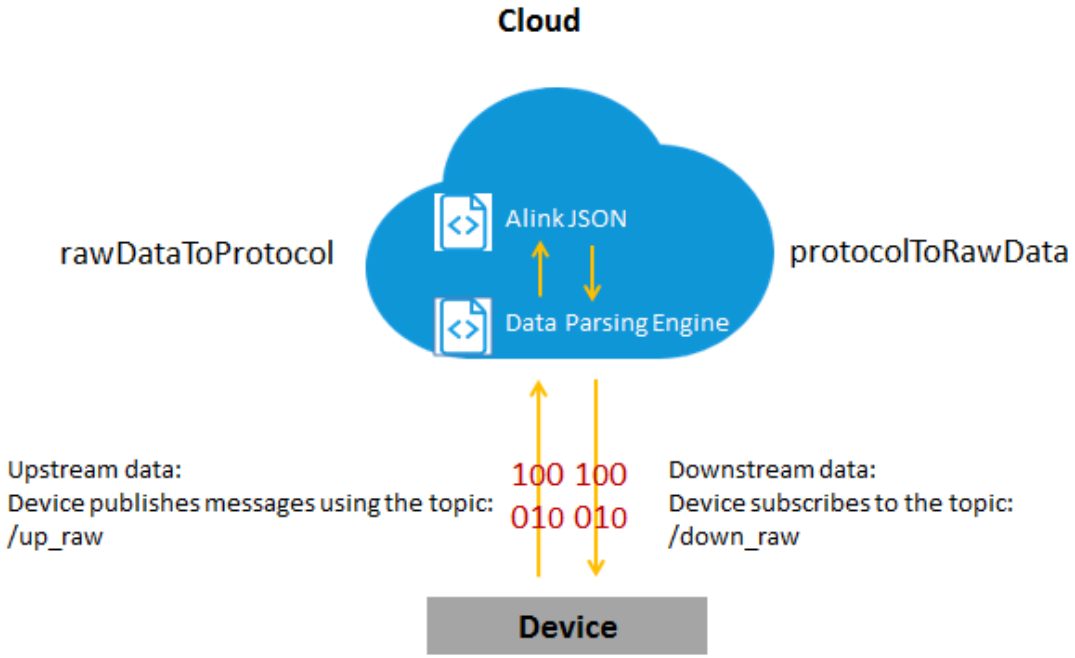
Python 例

Thing Specification Language (TSL) データの解析

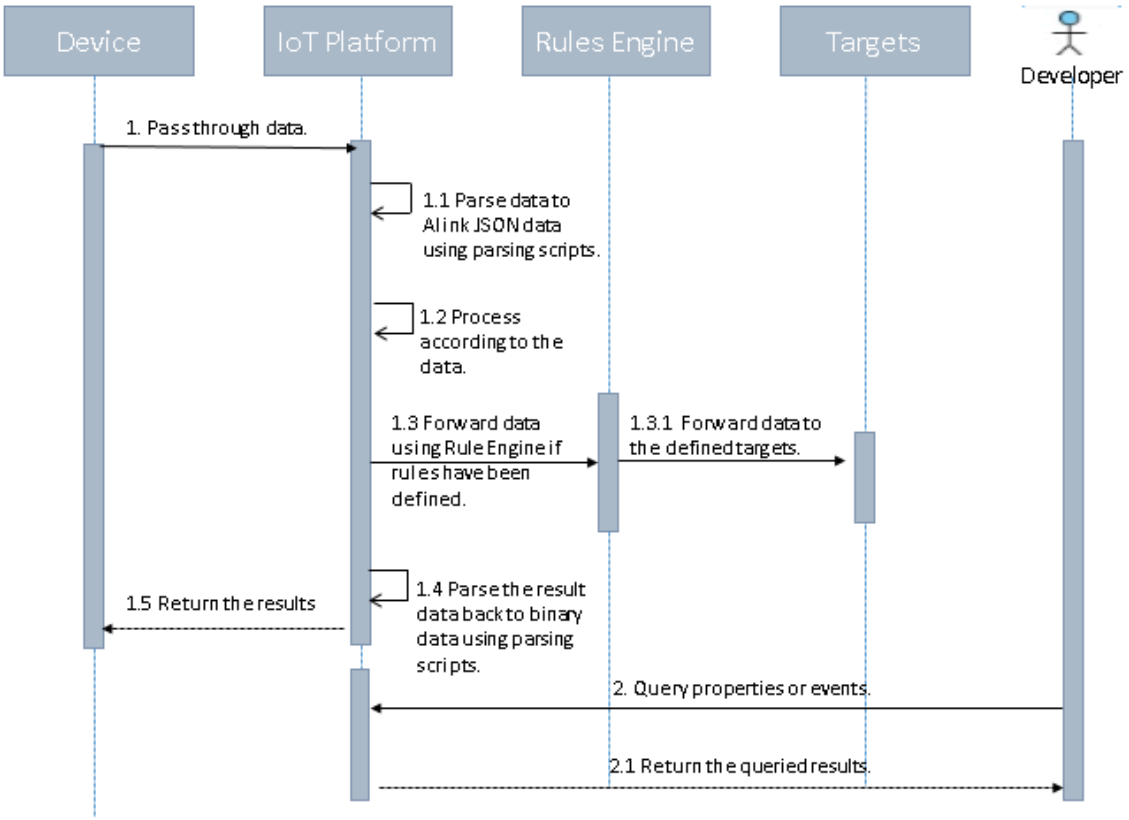
データ形式がカスタムのプロダクトのデバイスが IoT Platform と通信する場合、IoT Platform は送信されたデータ解析スクリプトを呼び出します。 次に、このスクリプトでアップストリームデータは標準の Alink JSON 形式に解析され、ダウンストリームデータはデバイスのカスタムデータ形式に解析されます。

IoT Platform はデバイスからデータを受信すると、解析スクリプトを実行し、後続のプロセスのためにパススルーデータを Alink JSON データに変換します。 データは、IoT Platform がデバイスに送信する前に、スクリプトに基づいてデバイスのカスタム形式に変換されます。

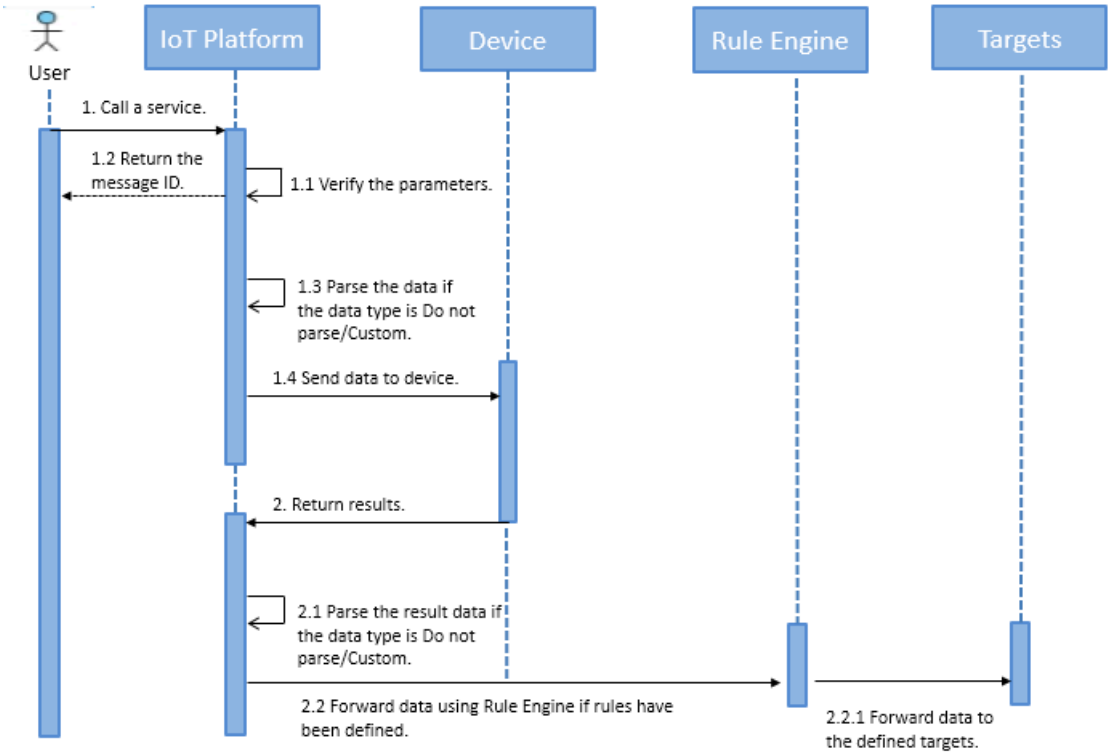
以下は、データ解析のフローチャートです。



以下は、デバイスがパススルー形式でプロパティやイベント (アップストリームデータ) をレポートするフローチャートです。



以下は、デバイスサービスを呼び出したりデバイスプロパティ (ダウンストリームデータ) を設定したりするコマンドを送信するフローチャートです。



関連トピック：

サンプルスクリプトについては、「TSL データの解析例」、「JavaScript 例」、「Python 例」をご参照ください。

2.2 カスタム Topic データの解析

2.2.1 概要

デバイスは、解析フラグ (? _sn=default) を含むカスタム Topic を介して IoT Platform にデータを送信できます。 IoT Platform はデータを受信した後、コンソールから送信されたデータ解析スクリプトを呼び出して、カスタムデータを JSON データに変換します。 次に、IoT Platform は後続のプロセスにデータを転送します。

説明

- カスタム Topic データの解析をサポートしているのは、中国 (上海) リージョンのみです。

- デバイスを設定する際、解析フラグ (?_sn=default) をカスタム Topic の後ろに追加する必要があります。 IoT Platform は、タグ付きの Topic を使用してデバイスから送信されたデータのみを解析します。

  たとえば、デバイスから Topic /${productKey}/${deviceName}/user/update にメッセージを送信するとします。 IoT Platform でデータを JSON 形式に解析する場合、 デバイス SDK の設定時、Topic を /${productKey}/${deviceName}/user/update?_sn=default と定義する必要があります。

  > 注：
  > IoT Platform コンソールでカスタム Topic を作成すると、解析フラグが追加されず、通常の Topic として定義されます。
- IoT Platform は、デバイスからクラウドにレポートされたデータのみを解析し、クラウドから送信されたダウンストリームデータは解析しません。
- IoT Platform は、クラウドにレポートされたペイロードのみを解析し、解析したペイロードを返します。
- データの Topic は、解析の前後で変化しません。 たとえば、デバイスから /${productKey}/${deviceName}/user/update にデータが送信されると、解析されたデータもこの Topic に送信されます。

データ解析用スクリプトの送信

1. IoT Platform コンソールにログインします。
2. 左側のナビゲーションウィンドウで、**[デバイス]** > **[プロダクト]** を選択します。
3. **[プロダクト]** ページでターゲットプロダクトを見つけ、**[表示]** をクリックします。
4. **[プロダクト詳細]** ページで、**[データ解析]** タブを選択します。
5. スクリプト言語を選択します。 **[スクリプトの編集]** テキストボックスにスクリプトを入力します。

   JavaScript (ECMAScript 5) と Python 2.7 がサポートされています。

   呼び出される関数をスクリプトに定義します。

   - JavaScript (ECMAScript 5)：**transformPayload()**
   - Python 2.7：**Transform_payload ()**

   完全なサンプルコードについては、「JavaScript 例」と「Python 例」をご参照ください。

   > 注：

プロダクトのデータ形式がカスタムの場合、TSL データを解析するスクリプトを記述する必要があります。 TSL データの解析の詳細については、「TSL データの解析例」をご参照ください。

6. スクリプトをテストします。

   a) **[**アナログ入力**]** タブで、**[**シミュレーションタイプ**]** を **[**カスタム**]** に設定し、デバイスと Topic を選択します。

   b) シミュレートするデバイスレポートデータを入力し、**[**実行**]** をクリックします。

7. スクリプトでデータが正しく解析されることを確認したら、 **[**送信**]** をクリックして、IoT Platform のバックエンドシステムにスクリプトを送信します。

2.2.2 JavaScript 例

このトピックでは、カスタム Topic データを解析する JavaScript のサンプルスクリプトを示します。

説明

- カスタム Topic データの解析をサポートしているのは、中国 (上海) リージョンのみです。
- デバイスを設定する際、解析フラグ (?_sn=default) をカスタム Topic の後ろに追加する必要があります。 IoT Platform は、タグ付きの Topic を使用してデバイスから送信されたデータのみを解析します。

  たとえば、デバイスから Topic `/${productKey}/${deviceName}/user/update` にメッセージを送信するとします。 IoT Platform でデータを JSON 形式に解析する場合、 デバイス SDK の設定時、Topic を `/${productKey}/${deviceName}/user/update?_sn=default` と定義する必要があります。

   注 :
  IoT Platform コンソールでカスタム Topic を作成すると、解析フラグが追加されず、通常の Topic として定義されます。
- IoT Platform は、デバイスからクラウドにレポートされたデータのみを解析し、クラウドから送信されたダウンストリームデータは解析しません。
- IoT Platform は、クラウドにレポートされたペイロードのみを解析し、解析したペイロードを返します。
- データの Topic は、解析の前後で変化しません。 たとえば、デバイスから `/${productKey}/${deviceName}/user/update` にデータが送信されると、解析されたデータもこの Topic に送信されます。

スクリプトテンプレート

```
var SELF_DEFINE_TOPIC_UPDATE_FLAG = '/user/update'  // Custom topic: /user/update
var SELF_DEFINE_TOPIC_ERROR_FLAG = '/user/update/error'  // Custom topic: /user/
update/error
/**
 * Converts data from devices to JSON data. The function is called when devices report
data through a custom topic to IoT Platform.
 * Input: A topic string. It is the topic for devices reporting messages.
 * Input: A rawData byte [] array. It cannot be empty.
 * Output: A jsonObj dictionary. It cannot be empty.
 */
function transformPayload(topic, rawData) {
var jsonObj = {};
return jsonObj;
}
```

サンプルスクリプト

注：

プロダクトのデータ形式がカスタムの場合、TSL データを解析するスクリプトを記述する必要があります。 TSL データの解析の詳細については、「TSL データの解析例」をご参照ください。

```
var SELF_DEFINE_TOPIC_UPDATE_FLAG = '/user/update'  // Custom topic: /user/update
var SELF_DEFINE_TOPIC_ERROR_FLAG = '/user/update/error'  // Custom topic: /user/
update/error
/*
  サンプルデータ
  Custom Topic:/user/update for reporting data
Input parameters: topic:/{productKey}/{deviceName}/user/update, and bytes:
0x000000000100320100000000
  Output parameters:
  {
    "prop_float": 0,
    "prop_int16": 50,
    "prop_bool": 1,
    "topic": "/{productKey}/{deviceName}/user/update"
  }
 */
function transformPayload(topic, bytes) {
    var uint8Array = new Uint8Array(bytes.length);
    for (int i = 0; i < bytes.length; i++) {
        uint8Array[i] = bytes[i] & 0xff;
    }
    var dataView = new DataView(uint8Array.buffer, 0);
    var jsonMap = {};

    if(topic.includes(SELF_DEFINE_TOPIC_ERROR_FLAG)) {
        jsonMap['topic'] = topic;
        jsonMap['errorCode'] = dataView.getInt8(0)
    } else if (topic.includes(SELF_DEFINE_TOPIC_UPDATE_FLAG)) {
        jsonMap['topic'] = topic;
        jsonMap['prop_int16'] = dataView.getInt16(5);
        jsonMap['prop_bool'] = uint8Array[7];
        jsonMap['prop_float'] = dataView.getFloat32(8);
    }

    return jsonMap;
```

```
}
```

2.2.3 Python 例

このトピックでは、カスタム Topic データを解析する Python のサンプルスクリプトを示します。

説明

- カスタム Topic データの解析をサポートしているのは、中国 (上海) リージョンのみです。
- デバイスを設定する際、解析フラグ (? _sn=default) をカスタム Topic の後ろに追加する必要があります。 IoT Platform は、タグ付きの Topic を使用してデバイスから送信されたデータのみを解析します。

  たとえば、デバイスから Topic /${productKey}/${deviceName}/user/update にメッセージを送信するとします。 IoT Platform でデータを JSON 形式に解析する場合、 デバイス SDK の設定時、Topic を /${productKey}/${deviceName}/user/update? _sn=default と定義する必要があります。

  > 注 :
  > IoT Platform コンソールでカスタム Topic を作成すると、解析フラグが追加されず、通常の Topic として定義されます。
- IoT Platform は、デバイスからクラウドにレポートされたデータのみを解析し、クラウドから送信されたダウンストリームデータは解析しません。
- IoT Platform は、クラウドにレポートされたペイロードのみを解析し、解析したペイロードを返します。
- データの Topic は、解析の前後で変化しません。 たとえば、デバイスから /${productKey}/${deviceName}/user/update にデータが送信されると、解析されたデータもこの Topic に送信されます。

スクリプトテンプレート

```
SELF_DEFINE_TOPIC_UPDATE_FLAG = '/user/update'  #Custom topic: /user/update
SELF_DEFINE_TOPIC_ERROR_FLAG = '/user/update/error'  #Custom topic: /user/update/
error

# Convert data from devices to JSON data.The function is called when devices report data
 through a custom topic to IoT Platform.
# Input: A topic string. It is the topic for devices reporting messages.
# Input: A rawData list. The list elements must be of the Int type and cannot be empty.
# Output: A jsonObj dictionary. It cannot be empty.
def transform_payload(topic, rawData):
  jsonObj = {}
```

```
    return jsonObj
```

例

注：

以下のスクリプトは、カスタム Topic データの解析にのみ使用します。 また、プロダクトのデータ形式がカスタムの場合、TSL データを解析するスクリプトを記述する必要があります。TSL データの解析の詳細については、「TSL データの解析例」をご参照ください。

```
# coding=utf-8
SELF_DEFINE_TOPIC_UPDATE_FLAG = '/user/update'  #Custom topic: /user/update
SELF_DEFINE_TOPIC_ERROR_FLAG = '/user/update/error'  #Custom topic: /user/update/
error

# Sample data
# Custom topic: /user/update reports data
# Input parameters: topic: /{productKey}/{deviceName}/user/update and bytes:
0x000000000100320100000000
# Output parameters:
# {
#    "prop_float": 0,
#    "prop_int16": 50,
#    "prop_bool": 1,
#    "topic": "/{productKey}/{deviceName}/user/update"
#  }
def transform_payload(topic, bytes):
    uint8Array = []
    for byteValue in bytes:
        uint8Array.append(byteValue & 0xff)

    jsonMap = {}
    if SELF_DEFINE_TOPIC_ERROR_FLAG in topic:
        jsonMap['topic'] = topic
        jsonMap['errorCode'] = bytes_to_int(uint8Array[0:1])

    elif SELF_DEFINE_TOPIC_UPDATE_FLAG in topic:
        jsonMap['topic'] = topic
        jsonMap['prop_int16'] = bytes_to_int(uint8Array[5:7])
        jsonMap['prop_bool'] = bytes_to_int(uint8Array[7: 8])
        jsonMap['prop_float'] = bytes_to_int(uint8Array[8:])

    return jsonMap

# Converts a byte array to an integer type
def bytes_to_int(bytes):
    data = ['%02X' % i for i in bytes]
```

```
return int(''.join(data), 16)
```

2.3 TSL データ解析

2.3.1 TSL データの解析例

このトピックでは、データ形式がカスタムのプロダクトの Thing Specification Language (TSL) データを解析するスクリプトを記述する方法について説明します。 アップストリームとダウンストリームのプロパティデータを解析するサンプルスクリプトを示します。

手順 1：スクリプトの送信

1. IoT Platform コンソールでプロダクトを作成します。

   プロダクトの作成方法の詳細については、「#unique_15」をご参照ください。

   > 注：
   > **[データ形式]** を **[カスタム]** に設定します。

2. プロダクトの TSL モデルを定義します。 カスタムプロパティの詳細については、「機能の定義」をご参照ください。

   この例では、次の 3 つのプロパティが定義されています。

識別子	データ型	有効値	読み取り/書き込みタイプ
prop_float	float	-100 から 100	読み取り/書き込み
prop_int16	int32	-100 から 100	読み取り/書き込み
prop_bool	bool	0：有効、1：無効	読み取り/書き込み

   アップストリームとダウンストリームの TSL データを解析するスクリプトは、ここで定義されている TSL に基づいて定義されます。

3. 通信プロトコルを次のように定義します。

   表 2-1：デバイスからレポートされたデータのリクエスト

フィールド	バイト数
フレームタイプ	1
リクエスト ID	4
プロパティ prop_int16	2

フィールド	バイト数
プロパティ prop_bool	1
プロパティ prop_float	4

表 2-2：デバイスからレポートされたデータに対するレスポンス

フィールド	バイト数
フレームタイプ	1
リクエスト ID	4
リターンコード	1

表 2-3：プロパティ設定のリクエスト

フィールド	バイト数
フレームタイプ	1
リクエスト ID	4
プロパティ prop_int16	2
プロパティ prop_bool	1
プロパティ prop_float	4

表 2-4：プロパティ設定に対するレスポンス

フィールド	バイト数
フレームタイプ	1
リクエスト ID	4
リターンコード	1

4. スクリプトを記述します。

   a) **[プロダクト]** ページでターゲットプロダクトを見つけ、**[表示]** をクリックします。

   b) **[プロダクト詳細]** ページで、**[データ解析]** タブを選択します。

   c) スクリプト言語を選択します。 **[スクリプトの編集]** テキストボックスにスクリプトを入力します。

   JavaScript (ECMAScript 5) と Python 2.7 がサポートされています。

   アップストリームとダウンストリームの TSL データを解析するには、スクリプトで次の関数を呼び出す必要があります。

   - Alink JSON データをカスタムデータに変換する関数
     - JavaScript (ECMAScript 5)：**protocolToRawData**
     - Python 2.7：**protocol_to_raw_data**
   - カスタムデータを Alink JSON データに変換する関数
     - JavaScript (ECMAScript 5)：**rawDataToProtocol**
     - Python 2.7：**raw_data_to_protocol**

   完全なサンプルスクリプトについては、「JavaScript 例」と「Python 例」をご参照ください。

   > 注：
   > また、解析したデータをカスタム Topic にアップロードするスクリプトを記述する必要があります。 スクリプトの記述方法の詳細については、「概要」をご参照ください。
   >
   > カスタム Topic データと TSL データのデータ解析を含む完全なサンプルスクリプトについては、「#unique_16」と「#unique_17」をご参照ください。

手順 2：オンラインでのスクリプトのテスト

スクリプトの編集後、**[アナログ入力]** タブで **[シミュレーションタイプ]** を選択し 、テストスクリプトとシミュレーションデータを入力してスクリプトをテストします。

- デバイスからレポートされたプロパティデータを解析します。

  シミュレーションタイプを **[**デバイス送信データ**]** に設定し、次のシミュレーションデータを入力して **[**実行**]** をクリックします。

  ```
  0x00002233441232013fa00000
  ```

  データ解析エンジンは、スクリプトで定義されたルールに従って、パススルーデータを JSON データに変換します。

  [解析結果**]** をクリックして、解析されたデータを表示します。

  ```
  {
    "method": "thing.event.property.post",
    "id": "2241348",
    "params": {
      "prop_float": 1.25,
      "prop_int16": 4658,
      "prop_bool": 1
    },
    "version": "1.0"
  }
  ```

- IoT Platform から返されたレスポンスを解析します。

  シミュレーションタイプを **[**デバイス受信データ**]** に設定し、次の JSON データを入力して **[**実行**]** をクリックします。

  ```
  {
   "id": "12345",
   "version": "1.0",
   "code": 200,
   "method": "thing.event.property.post",
   "data": {}
  }
  ```

  データ解析エンジンは、JSON データを次のデータに変換します。

  ```
  0x0200003039c8
  ```

- IoT Platform から送信されたプロパティ設定用のデータを解析します。

  シミュレーションタイプを **[**デバイス受信データ**]** に設定し、次の JSON データを入力して **[**実行**]** をクリックします。

  ```
  {
    "method": "thing.service.property.set",
    "id": "12345",
    "version": "1.0",
    "params": {
      "prop_float": 123.452,
      "prop_int16": 333,
      "prop_bool": 1
    }
  ```

```
}
```

データ解析エンジンは、JSON データを次のデータに変換します。

```
0x0100003039014d0142f6e76d
```

- プロパティを設定した後、デバイスから返された結果データを解析します。

  シミュレーションタイプを **[デバイス送信データ]** に設定し、次のシミュレーションデータを入力して **[実行]** をクリックします。

  ```
  0x0300223344c8
  ```

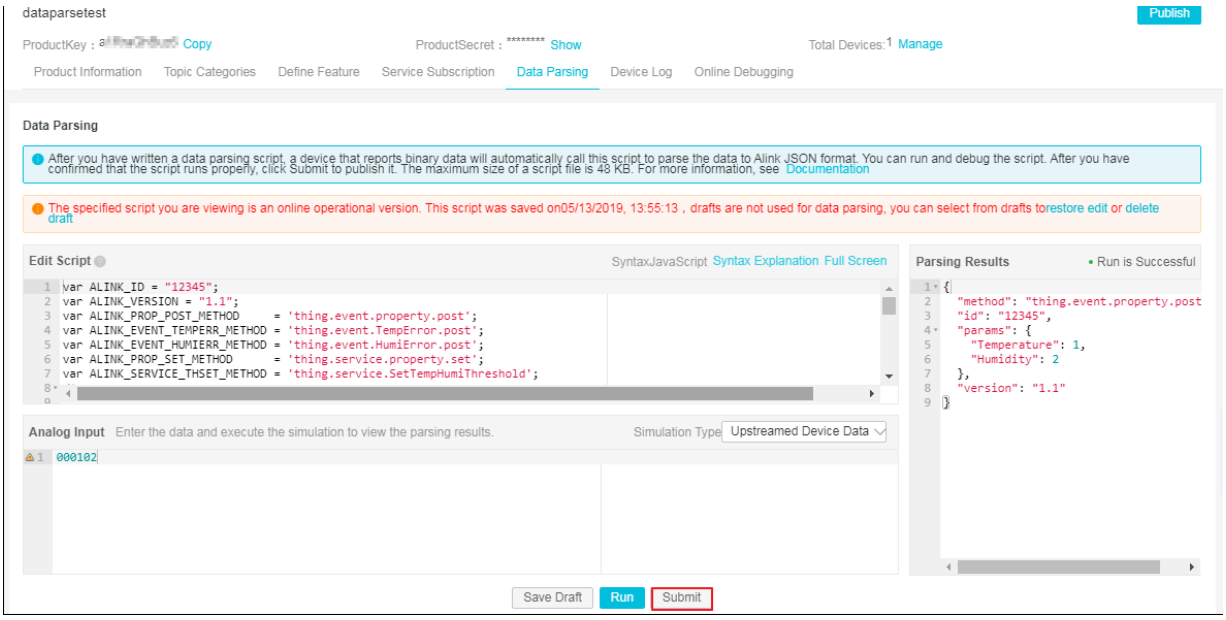
  データ解析エンジンは、パススルーデータを次の JSON データに変換します。

  ```
  {
    "code": "200",
    "data": {},
    "id": "2241348",
    "version": "1.0"
  }
  ```

手順 3：スクリプトの送信

スクリプトでデータが正しく解析されることを確認したら、**[送信]** をクリックして、IoT Platform のバックエンドシステムにスクリプトを送信します。

> 注：
> IoT Platform は、送信されたスクリプトのみを呼び出すことができます。 ステータスが下書きのスクリプトは呼び出すことができません。



手順 4：物理デバイスでのデバッグ

スクリプトを使用する前に、実デバイスを使用して IoT Platform と通信します。 IoT Platform がスクリプトを呼び出し、アップストリームデータとダウンストリームデータを解析できることを確認してください。

- アップストリームプロパティデータをテストします。
  1. デバイスからレポートされたプロパティデータ (0x00022334412320 13fa00000 など) を使用します。
  2. IoT Platform コンソールで、**[デバイス]** > **[デバイス]** を選択します。
  3. ターゲットデバイスを見つけ、**[表示]** をクリックします。 **[デバイス情報]** ページの **[ステータス]** タブで、プロパティデータが存在するかどうかを確認します。
- ダウンストリームプロパティデータをテストします。
  1. IoT Platform コンソールで、**[監視と運用]** > **[オンラインデバッグ]** を選択します。
  2. デバッグするプロダクトとデバイスを選択します。 **[実デバイスのデバッグ]** をクリックします。 デバッグ機能をプロパティの識別子 (prop_int16 など) に設定します。 メソッドを **[Set]** に設定します。 次のデータを入力し、**[コマンドの送信]** をクリックします。

     ```
     {
       "method": "thing.service.property.set",
       "id": "12345",
       "version": "1.0",
       "params": {
         "prop_float": 123.452,
         "prop_int16": 333,
         "prop_bool": 1
       }
     }
     ```
  3. デバイスがプロパティ設定用のデータを受信しているかどうかを確認します。
  4. このデバイスの **[デバイス情報]** ページの **[ステータス]** タブで、プロパティデータをレポートしているかどうかを確認します。

関連トピック

- JavaScript (ECMAScript 5) のスクリプトテンプレートと例の詳細については、「JavaScript 例」をご参照ください。
- Python 2.7 のスクリプトテンプレートと例の詳細については、「Python 例」をご参照ください。
- データ解析プロセスの詳細については、「データ解析の概要」をご参照ください。
- カスタム Topic データの解析方法の詳細については、「概要」をご参照ください。

2.3.2 JavaScript 例

このトピックでは、Thing Specification Language (TSL) データを解析する JavaScript のスクリプトテンプレートとサンプルスクリプトを示します。

スクリプトテンプレート

次の JavaScript スクリプトテンプレートに基づいて、データ解析スクリプトを記述できます。

注：

このテンプレートは、データ形式がカスタムのプロダクトにのみ適用できます。

```
/**
 * Converts Alink data to a data format that can be recognized by devices before data is sent to devices from IoT Platform.
 * Input: A jsonObj object. It cannot be empty
 * Output: A rawData byte[] array. It cannot be empty
 *
 */
function protocolToRawData(jsonObj) {
    return rawdata;
}

/**
 * Converts custom data from devices to Alink protocol data. This function is called when devices report data to IoT Platform.
 * Input: A rawData byte [] array. It cannot be empty
 * Output: A jsonObj dictionary. It cannot be empty.
 */
function rawDataToProtocol(rawData) {
    return jsonObj;
}
```

追加の考慮事項

- グローバル変数を使用しないでください。 グローバル変数を使用すると、実行結果に不整合が生じる可能性があります。
- スクリプトでは、2 つの補数演算を使用してデータを処理します。 範囲 [-128,127] の値の補数は [0,255] です。 たとえば、-1 の補数は 10 進数の 255 です。
- rawDataToProtocol 関数の入力は整数配列です。 ビット単位の AND 演算 には、`0xFF` を使用して補数を取得します。
- protocolToRawData 関数は、IoT Platform から送信されたデータを解析し、結果を配列で返します。 配列要素は 0～255 の整数です。

例

次のスクリプトは、「TSL データの解析例」で定義されているプロパティとプロトコルに基づいています。

```
var COMMAND_REPORT = 0x00; // Property data reported by devices
var COMMAND_SET = 0x01; // Property setting command data from the cloud
var COMMAND_REPORT_REPLY = 0x02;  // Response data for devices reporting properties
var COMMAND_SET_REPLY = 0x03;  // Response data for setting device properties
var COMMAD_UNKOWN = 0xff;  // Unknown commands
var ALINK_PROP_REPORT_METHOD = 'thing.event.property.post'; // The topic for devices to report current property data to the cloud
var ALINK_PROP_SET_METHOD = 'thing.service.property.set';  // The topic for IoT platform to send setting property commands to control devices
var ALINK_PROP_SET_REPLY_METHOD = 'thing.service.property.set';  // The topic for devices to report property setting results to IoT platform
/*
Sample data:
The device reports property data
Input parameters->
   0x0000000001003201000000000
Output result->
   {"method":"thing.event.property.post","id":"1","params":{"prop_float":0,"prop_int16":50,"prop_bool":1},"version":"1.0"}

The device responds after setting properties
Input parameters->
   0x0300223344c8
Output result->
   {"code":"200","data":{},"id":"2241348","version":"1.0"}
*/
function rawDataToProtocol(bytes) {
   var uint8Array = new Uint8Array(bytes.length);
   for (var i = 0; i < bytes.length; i++) {
      uint8Array[i] = bytes[i] & 0xff;
   }
   var dataView = new DataView(uint8Array.buffer, 0);
   var jsonMap = new Object();
   var fHead = uint8Array[0]; // The command prefix
   if (fHead == COMMAND_REPORT) {
      jsonMap['method'] = ALINK_PROP_REPORT_METHOD; // The topic for reporting properties in the ALink JSON
      jsonMap['version'] = '1.0'; // The fixed protocol version in the ALink JSON
      jsonMap['id'] = ''+ dataView.getInt32(1); // The request ID in the ALink JSON
      var params = {};
      params['prop_int16'] = dataView.getInt16(5); // The value of the prop_int16 property
      params['prop_bool'] = uint8Array[7]; // The value of the prop_bool property
      params['prop_float'] = dataView.getFloat32(8); // The value of the prop_float property
      jsonMap['params'] = params; // The params field in the ALink JSON
   } else if(fHead == COMMAND_SET_REPLY) {
      jsonMap['version'] = '1.0'; // The fixed protocol version in the ALink JSON
      jsonMap['id'] = ''+ dataView.getInt32(1); // The request ID in the ALink JSON
      jsonMap['code'] = ''+ dataView.getUint8(5);
      jsonMap['data'] = {};
   }

   return jsonMap;
}
/*
Sample data:
IoT Platform pushes a command for setting properties to the device
```

```
Input parameters->
   {"method":"thing.service.property.set","id":"12345","version":"1.0","params":{"
prop_float":123.452, "prop_int16":333, "prop_bool":1}}
Output result->
   0x0100003039014d0142f6e76d
IoT Platform responds after the device reports properties
Input data->
   {"method":"thing.event.property.post","id":"12345","version":"1.0","code":200,"data":{}}
Output result->
   0x0200003039c8
*/
function protocolToRawData(json) {
   var method = json['method'];
   var id = json['id'];
   var version = json['version'];
   var payloadArray = [];
   if (method == ALINK_PROP_SET_METHOD) // Sets properties
   {
      var params = json['params'];
      var prop_float = params['prop_float'];
      var prop_int16 = params['prop_int16'];
      var prop_bool = params['prop_bool'];
      // Raw data is concatenated based on the custom protocol format.
      payloadArray = payloadArray.concat(buffer_uint8(COMMAND_SET)); // The
command field
      payloadArray = payloadArray.concat(buffer_int32(parseInt(id))); // The id in the
ALink JSON
      payloadArray = payloadArray.concat(buffer_int16(prop_int16)); // The value of the
prop_int16 property
      payloadArray = payloadArray.concat(buffer_uint8(prop_bool)); // The value of the
prop_bool property
      payloadArray = payloadArray.concat(buffer_float32(prop_float)); // The value of the
 prop_float property
   } else if (method == ALINK_PROP_REPORT_METHOD) {// The response of the reported
data
      var code = json['code'];
      payloadArray = payloadArray.concat(buffer_uint8(COMMAND_SET)); // The
command field
      payloadArray = payloadArray.concat(buffer_int32(parseInt(id))); // The id in the
ALink JSON
      payloadArray = payloadArray.concat(buffer_uint8(code));
   } else {// Unknown commands. Some commands are not processed.
      var code = json['code'];
      payloadArray = payloadArray.concat(buffer_uint8(COMMAND_UNKOWN)); // The
command field
      payloadArray = payloadArray.concat(buffer_int32(parseInt(id))); // The id in the
ALink JSON
      payloadArray = payloadArray.concat(buffer_uint8(code));
   }
   return payloadArray;
}
// The following are some helper functions.
function buffer_uint8(value) {
   var uint8Array = new Uint8Array(1);
   var dv = new DataView(uint8Array.buffer, 0);
   dv.setUint8(0, value);
   return [].slice.call(uint8Array);
}
function buffer_int16(value) {
   var uint8Array = new Uint8Array(2);
   var dv = new DataView(uint8Array.buffer, 0);
   dv.setInt16(0, value);
   return [].slice.call(uint8Array);
}
```

```
function buffer_int32(value) {
    var uint8Array = new Uint8Array(4);
    var dv = new DataView(uint8Array.buffer, 0);
    dv.setInt32(0, value);
    return [].slice.call(uint8Array);
}
function buffer_float32(value) {
    var uint8Array = new Uint8Array(4);
    var dv = new DataView(uint8Array.buffer, 0);
    dv.setFloat32(0, value);
    return [].slice.call(uint8Array);
}
```

2.3.3 Python 例

このトピックでは、Thing Specification Language (TSL) データを解析する Python のスクリプトテンプレートとサンプルスクリプトを示します。

スクリプトテンプレート

次の Python スクリプトテンプレートに基づいて、データ解析スクリプトを記述できます。

 注：

このテンプレートは、データ形式がカスタムのプロダクトにのみ適用できます。

```
# Converts the custom data of devices to Alink protocol data. This function is called when
 devices report data to IoT Platform.
# Input: A rawData list. The list elements must be of the Int type and cannot be empty.
# Output: A jsonObj dictionary. It cannot be empty.
def raw_data_to_protocol(rawData):
    jsonObj = {}
    return jsonObj

# Converts Alink data to data formats that can be recognized by devices before data is
sent to devices from IoT Platform.
# Input: A jsonData dictionary. It cannot be empty.
# Output: A rawData list. The list elements must be integers from 0 to 255. The list cannot
 be empty.
def protocol_to_raw_data(jsonData):
    rawData = []
    return rawData
```

追加の考慮事項

- グローバル変数を使用しないでください。 グローバル変数を使用すると、実行結果に不整合が生じる可能性があります。
- スクリプトでは、2 つの補数演算を使用してデータを処理します。 範囲 [-128,127] の値の補数は [0,255] です。 たとえば、-1 の補数は 10 進数の 255 です。
- raw_data_to_protocol 関数の入力は整数配列です。 ビット単位の AND 演算 には、`0xFF` を使用して補数を取得します。

- protocol_to_raw_data 関数は、IoT Platform から送信されたデータを解析し、結果を配列で返します。 配列要素は 0～255 の整数です。

例

次のスクリプトは、「TSL データの解析例」で定義されているプロパティとプロトコルに基づいています 。

```
# coding=UTF-8
import struct

COMMAND_REPORT = 0x00 # Property data reported by devices
COMMAND_SET = 0x01 # Property setting command data from the cloud
COMMAND_REPORT_REPLY = 0x02 # // Response data for devices reporting properties
COMMAND_SET_REPLY = 0x03 # // Response data for setting device property command
COMMAD_UNKOWN = 0xff  # // Unknown commands
ALINK_PROP_REPORT_METHOD = 'thing.event.property.post' # The topic for devices to upload property data to the cloud
ALINK_PROP_SET_METHOD = 'thing.service.property.set' # // The topic for IoT platform to send property setting commands to control devices
ALINK_PROP_SET_REPLY_METHOD = 'thing.service.property.set' # // The topic for devices to report property setting results to IoT platform


# Example data:
# The device reports property data
# Input parameters->
#    0x0000000001003201000000000
# Output result->
#    {"method":"thing.event.property.post","id":"1","params":{"prop_float":0,"prop_int16":50,"prop_bool":1},"version":"1.0"}
# The device responds after setting properties
# Input parameters->
#    0x0300223344c8
# Output result->
#    {"code":"200","data":{},"id":"2241348","version":"1.0"}

def raw_data_to_protocol(bytes):
    uint8Array = []
    for byteValue in bytes:
        uint8Array.append(byteValue & 0xff)

    fHead = uint8Array[0]
    jsonMap = {}
    if fHead == COMMAND_REPORT:
        jsonMap['method'] = ALINK_PROP_REPORT_METHOD
        jsonMap['version'] = '1.0'
        jsonMap['id'] = str(bytes_to_int(uint8Array[1:5]))
        params = {}
        params['prop_int16'] = bytes_to_int(uint8Array[5:7])
        params['prop_bool'] = bytes_to_int(uint8Array[7: 8])
        params['prop_float'] = bytes_to_int(uint8Array[8:])
        jsonMap['params'] = params
    elif fHead == COMMAND_SET_REPLY:
        jsonMap['version'] = '1.0'
        jsonMap['id'] = str(bytes_to_int(uint8Array[1:5]))
        jsonMap['code'] = str(bytes_to_int(uint8Array[5:]))
        jsonMap['data'] = {}

    return jsonMap
```

```
# Example data:
# IoT Platform pushes a command for setting properties to the device
# Input parameters->
#    {"method":"thing.service.property.set","id":"12345","version":"1.0",
#    "params":{"prop_float":123.452, "prop_int16":333, "prop_bool":1}}
# Output result->
#    0x0100003039014d0142f6e76d
# IoT Platform responds after the device reports properties
# Input data->
#    {"method":"thing.event.property.post","id":"12345","version":"1.0","code":200,"data":
{}}
# Output result->
#    0x0200003039c8
def protocol_to_raw_data(json):
    method = json.get('method', None)
    id = json.get('id', None)
    version = json.get('version', None)
    payload_array = []

    if method == ALINK_PROP_SET_METHOD:
        params = json.get('params')
        prop_float = params.get('prop_float', None)
        prop_int16 = params.get('prop_int16', None)
        prop_bool = params.get('prop_bool', None)

        payload_array = payload_array + int_8_to_byte(COMMAND_SET)
        payload_array = payload_array + int_32_to_byte(int(id))
        payload_array = payload_array + int_16_to_byte(prop_int16)
        payload_array = payload_array + int_8_to_byte(prop_bool)
        payload_array = payload_array + float_to_byte(prop_float)

    elif method == ALINK_PROP_REPORT_METHOD:
        code = json.get('code', None)
        payload_array = payload_array + int_8_to_byte(COMMAND_REPORT_REPLY)
        payload_array = payload_array + int_32_to_byte(int(id))
        payload_array = payload_array + int_8_to_byte(code)
    else:
        code = json.get('code')
        payload_array = payload_array + int_8_to_byte(COMMAD_UNKOWN)
        payload_array = payload_array + int_32_to_byte(int(id))
        payload_array = payload_array + int_8_to_byte(code)

    return payload_array


# Converts a byte array to an integer type
def bytes_to_int(bytes):
    data = ['%02X' % i for i in bytes]
    return int(''.join(data), 16)


# Converts a byte array to a floating point without precision
def bytes_to_float(bytes):
    data = []
    for i in bytes:
        t_value = '%02X' % i
        if t_value % 2 ! = 0:
            t_value += 0
        data.append(t_value)
    hex_str = ''.join(data)
    return struct.unpack('! f', hex_str.decode('hex'))[0]
```

```
# Converts an 8-bit integer to a byte array
def int_8_to_byte(value):
    t_value = '%02X' % value
    if len(t_value) % 2 ! = 0:
        t_value += '0'

    return hex_string_to_byte_array(t_value)


# Converts a 32-bit integer to a byte array
def int_32_to_byte(value):
    t_value = '%08X' % value
    if len(t_value) % 2 ! = 0:
        t_value += '0'

    return hex_string_to_byte_array(t_value)


# Converts a 16-bit integer to a byte array
def int_16_to_byte(value):
    t_value = '%04X' % value
    if len(t_value) % 2 ! = 0:
        t_value += '0'

    return hex_string_to_byte_array(t_value)


# Converts a float into an integer array
def float_to_byte(param):
    return hex_string_to_byte_array(struct.pack(">f", param).encode('hex'))


# Converts a hexadecimal string to a byte array
def hex_string_to_byte_array(str_value):
    if len(str_value) % 2 ! = 0:
        return None

    cycle = len(str_value) / 2

    pos = 0
    result = []
    for i in range(0, cycle, 1):
        temp_str_value = str_value[pos:pos + 2]
        temp_int_value = int(temp_str_value, base=16)

        result.append(temp_int_value)
        pos += 2
    return result
```

3 タグ

タグは、プロダクト、デバイス、またはデバイスグループに対して設定するカスタマイズ識別子です。 タグを使用することで、プロダクトやデバイス、グループを柔軟に管理できます。

IoT では、多くの場合、多数のプロダクトやデバイスを管理する必要があります。 課題となるのは、さまざまなプロダクトやデバイスをどう区別するか、そして、どのように集中管理を達成するかです。 Alibaba Cloud IoT Platform は、これらの問題に対処するためにタグを提供します。 タグを使用すれば、さまざまなプロダクトやデバイスを集中管理できます。

そのため、プロダクトやデバイス、デバイスグループに対してタグを作成することを推奨します。 タグの構造は `key: value` です。

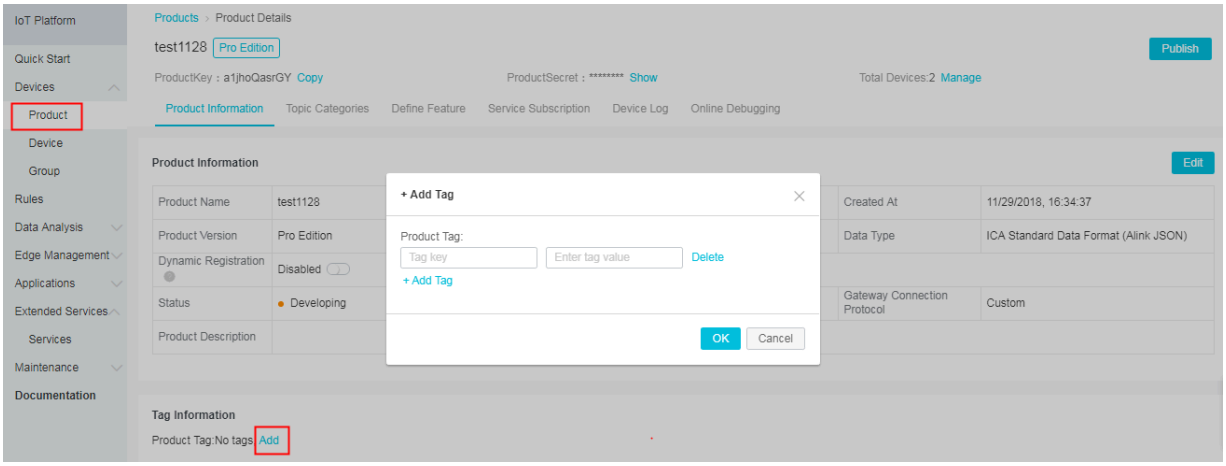
このトピックでは、プロダクトタグ、デバイスタグ、グループタグを作成する方法を説明します。

プロダクトタグ

プロダクトタグは、通常、プロダクトのすべてのデバイスに共通する情報を示します。 たとえば、タグは特定のメーカー、組織、物理的なサイズ、およびオペレーティングシステムを示すことができます。 プロダクト作成後、そのプロダクトにタグを追加できます。

プロダクトタグを追加するには、次の手順を実行します。

1. IoT Platform コンソールにログインします。
2. 左側のナビゲーションウィンドウで、**[デバイス]** > **[プロダクト]** をクリックします。
3. **[プロダクト詳細]** ページで、タグを追加するプロダクトを検索し、**[表示]** をクリックします。
4. **[タグ情報]** の下の **[追加]** をクリックします。
5. ダイアログボックスで、**[タグキー]** と **[タグ値]** の値を入力し、**[OK]** をクリックします。

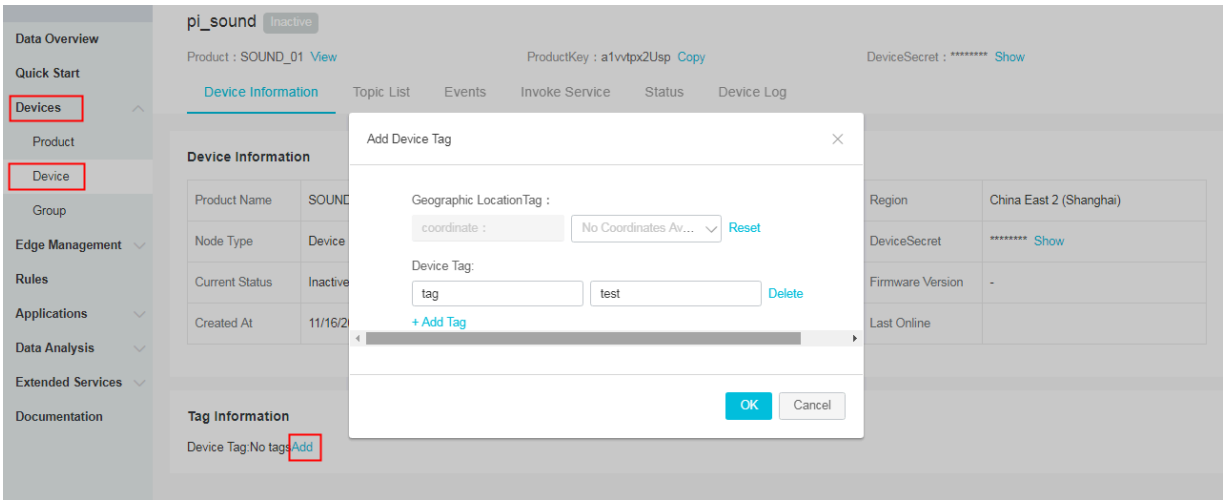


デバイスタグ

デバイスに固有のタグを追加することでデバイス管理を容易にすることができます。 たとえば、201 号室の電気メーターに対して PowerMeter:room201 とするように、デバイス機能情報をタグとして使用できます。 デバイスタグは、コンソールまたは API を使用して管理できます。

コンソールでデバイスタグを追加するには、次の手順を実行します。

1. IoT Platform コンソールにログインします。
2. 左側のナビゲーションウィンドウで、**[デバイス]** > **[デバイス]** をクリックします。
3. **[デバイス]** ページで、タグを追加するデバイスを検索し、**[表示]** をクリックして **[デバイスの詳細]** ページに移動します。
4. **[タグ情報]** の下の **[追加]** をクリックします。
5. ダイアログボックスで、**[タグキー]** と **[タグ値]** の値を入力し、**[OK]** をクリックします。



デバイスタグは常にデバイスの後に続き、ルールエンジンを使用することにより、他の Alibaba Cloud サービスに転送できます。

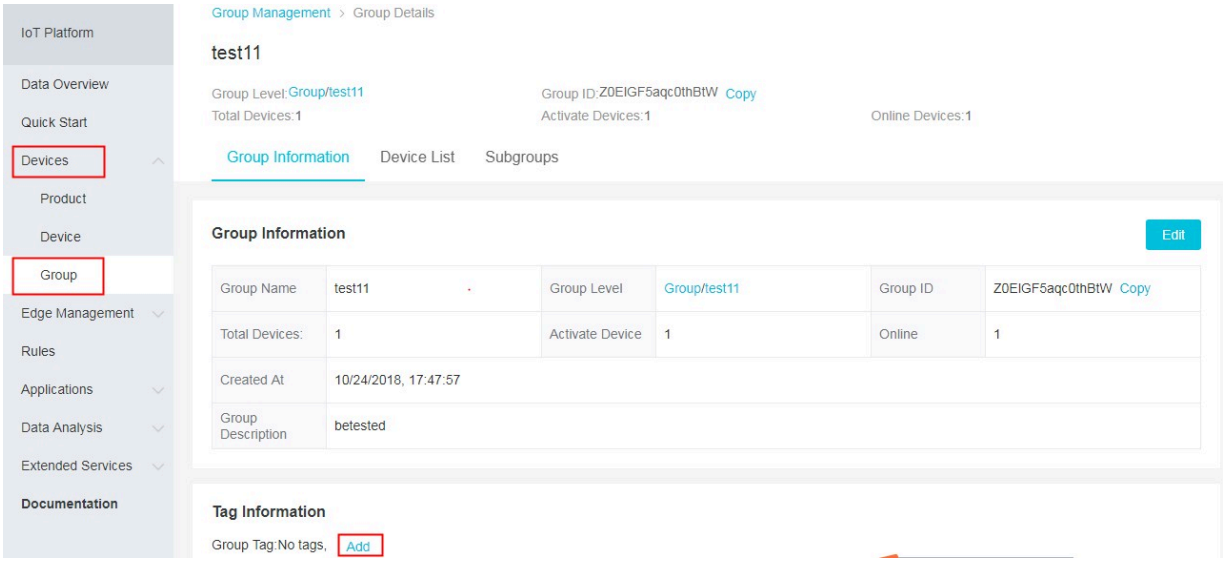
グループタグ

デバイスをグループ化することで、プロダクト間でデバイスを管理できます。 グループタグは通常、グループやサブグループ内のデバイスの一般的な情報を示します。 たとえば、リージョン情報をグループタグとして使用できます。 グループ作成後、そのグループにタグを追加できます。

プロダクトタグを追加するには、次の手順を実行します。

1. IoT Platform コンソールにログインします。
2. 左側のナビゲーションウィンドウで、**[デバイス]** > **[グループ]** をクリックします。
3. **[グループ管理]** ページで、タグを追加するグループを検索し、**[表示]** をクリックします。
4. **[タグ情報]** の下の **[追加]** をクリックします。

5. ダイアログボックスで、**[タグキー]** と **[タグ値]** の値を入力し、**[OK]** をクリックします。

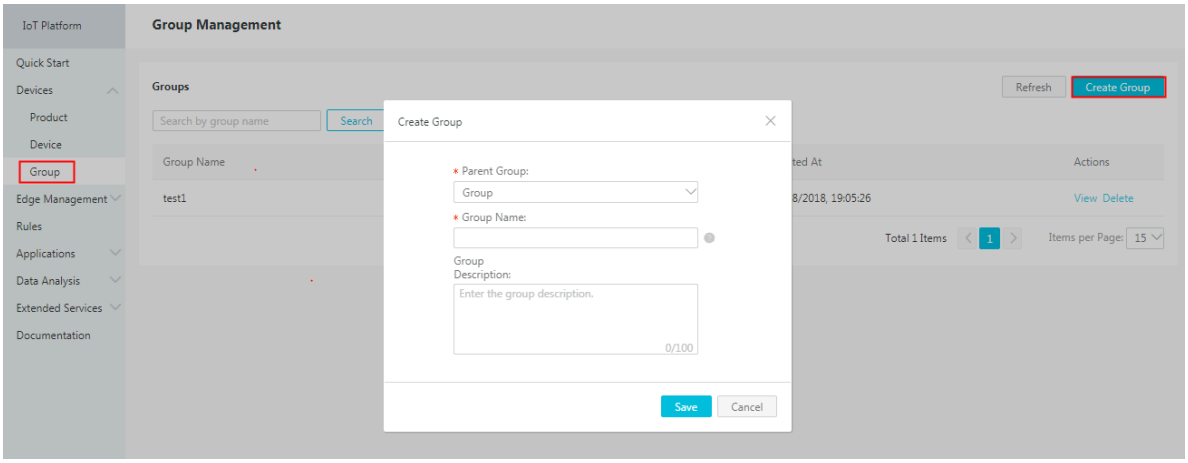


4 デバイスグループ

IoT Platform はデバイスグループをサポートします。 異なるプロダクトのデバイスを同じグループに割り当てることができます。 ここでは、IoT Platform コンソールで、デバイスグループを作成し管理する方法を紹介します。

1. IoT Platform コンソールにログインします。
2. **[デバイス] > [グループ]** をクリックします。
3. [グループ管理] ページで、**[グループの作成]** をクリックし、グループ情報を入力して **[保存]** をクリックします。

> 注 :
> 最大 1,000 グループ (親グループとサブグループを含む) を作成できます。



パラメーターは次のとおりです。

- 親グループ: グループタイプを選択します。
  - グループ: 作成するグループが親グループであることを示します。
  - 既存のグループを選択: グループを親グループとして指定し、その親グループ用のサブグループを作成します。
- グループ名: グループの名前を入力します。 グループ名の長さは 4 ～ 30 文字で、漢字、英数字、およびアンダースコア (_) を含めることができます。 グループ名はアカウントのグループ間で一意である必要があり、一度グループが作成されたら変更できません。
- グループの説明: グループの説明です。 空欄のままにできます。

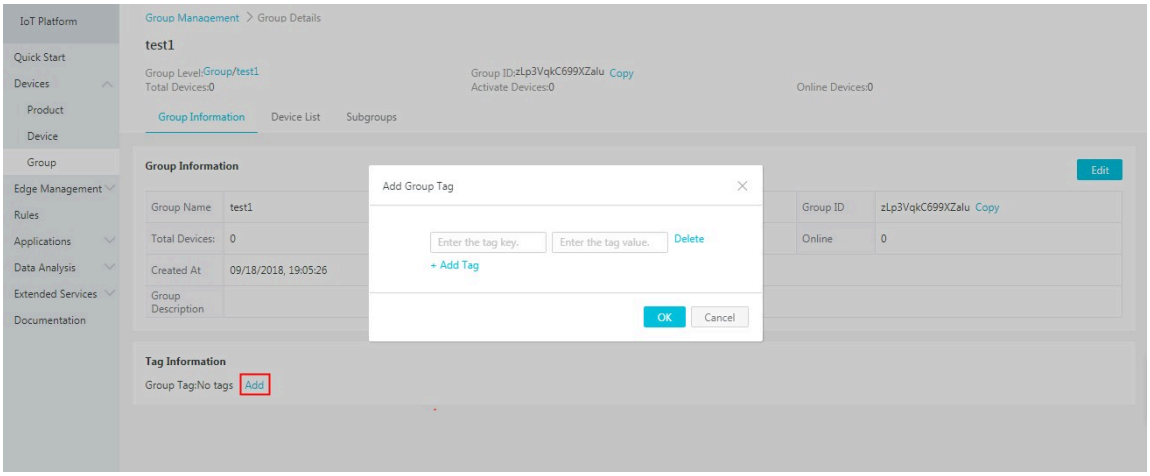
4. **[グループ管理]** ページで **[表示]** をクリックし、対応するグループの **[グループの詳細]** ページを表示します。

5. グループにタグを追加します (省略可)。 タグは、グループを管理するとき、グループ識別子として使用されます。

   a) タグ情報で **[追加]** をクリックし、タグのキーと値を入力します。

   b) **[OK]** をクリックして、入力したすべてのタグを作成します。

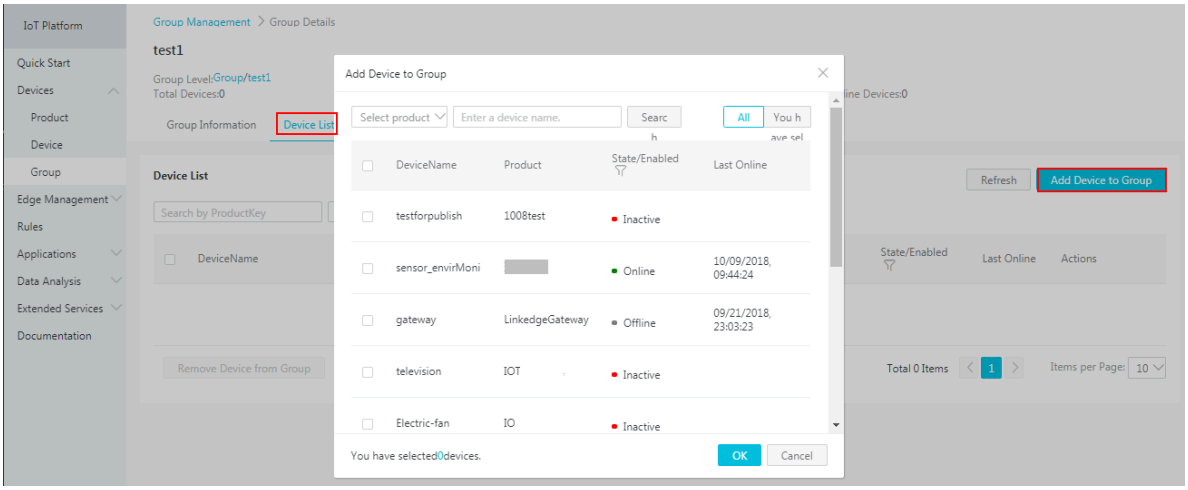
> 注：
> 1 つのグループに最大 100 個のタグを追加できます。



6. **[デバイス一覧] > [グループにデバイスを追加]** をクリックします。 グループに追加するデバイスを選択します。

> 注：
> - デバイスは、一度に 200 個まで追加できます。 1 つのグループに合計で最大 20,000 個のデバイスを追加できます。

- 1 つのデバイスを、最大で 10 のグループに含めることができます。



[グループにデバイスを追加] ページの右上隅には、**[すべて]** と **[選択済み]** の 2 つのボタンがあります。

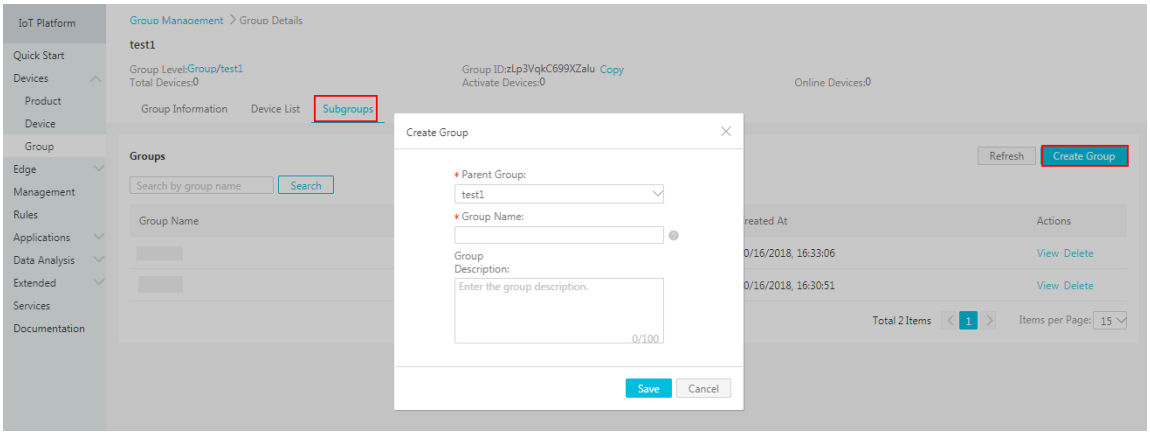
- **[すべて]** をクリックすると、すべてのデバイスが表示されます。
- **[選択済み]** をクリックすると、選択したデバイスが表示されます。

7. **[サブグループ]** > **[グループの作成]** をクリックし、グループにサブグループを追加します (省略可)。

   サブグループは、より具体的な方法でデバイスを管理するために使用されます。 たとえば、親グループ "SmartHome" に対して、"SmartKitchen" や "SmartBedroom" のようなサブグ

ループを作成します。その場合、キッチンのデバイスとバスルームのベッドルームのデバイスを別々に管理することができます。 手順は次のとおりです。

a) 親グループを選択し、グループ名と説明を入力し、**[保存]** をクリックします。



b) 親グループの **[サブグループ]** ページで、**[表示]** をクリックし、対応する **[グループ詳細]** ページを表示します。

c) **[デバイス一覧]** > **[グループにデバイスを追加]** をクリックし、サブグループにデバイスを追加します。

サブグループを作成し、そこにデバイスを追加したら、管理が可能になります。 サブグループ内にサブサブグループを作成することもできます。

注：

- 1 つのグループには最大 100 個のサブグループを含めることができます。
- グループは 3 層 のみサポートされています (親グループ、サブグループ、サブサブグループ)。
- グループは、1 つの親グループのみのサブグループになることができます。
- 一度作成した親グループとサブグループ間の関係を変更することはできません。 関係を変更する場合、既存のサブグループを削除し、新しいものを作成します。
- サブグループを持つグループを削除することはできません。 親グループを削除する前に、その親グループに所持しているサブグループをすべて削除する必要があります。

5 デバイスシャドウ

5.1 デバイスシャドウ

デバイスシャドウは、デバイスから報告されたステータスと、デバイスに対して希望するステータスを保存するための JSON ファイルです。

- 1 つのデバイスにデバイスシャドウは 1 つのみです。 デバイスは、MQTT (Message Queuing Telemetry Transport) を使用してデバイスシャドウを取得および設定します。 したがって、デバイスシャドウのステータスとデバイスのステータスを同期させることができます。
- アプリケーションは、IoT Platform の SDK を使用して、デバイスシャドウを取得および設定します。 アプリケーションはデバイスシャドウを使用して、対象デバイスの最新のステータスを取得したり、希望するステータスを対象デバイスに送信したりできます。

シナリオ 1

デバイスと IoT Platform との間で、切断と再接続が頻繁に繰り返されています。 これは、ネットワークの状態が不安定なためです。 アプリケーションは、オフラインのデバイスにステータスをリクエストしても取得できず、デバイスが再接続したときに別のデバイスステータスリクエストを送信しても失敗します。

デバイスシャドウは、デバイスと同期されているので、最新のデバイスステータスに更新され、保存されています。 したがって、デバイスがオフラインかオンラインかにかかわらず、アプリケーションはデバイスシャドウから最新のデバイスステータスを取得できます。

シナリオ 2

ネットワークが安定している状態で、デバイスは複数のアプリケーションからデバイスステータスリクエストを受信し、それぞれのステータスリクエストにレスポンスする必要があります。 レスポンスの内容が同じであっても、これらのリクエストを処理するとき、デバイスが過負荷になることがあります。

IoT Platform では、デバイスはデバイスシャドウだけに同期します。 アプリケーションは、対象デバイス自体ではなく、デバイスシャドウに対して最新のデバイスステータスをリクエストできます。 したがって、アプリケーションはデバイスから切り離されます。

シナリオ3

- デバイスと IoT Platform との間で、切断と再接続が頻繁に繰り返されています。 これは、ネットワークの状態が不安定なためです。 オフライン状態のデバイスは、アプリケーションコマンドを受信できません。
  - この問題は、QoS 1 または 2 (Quality of Service 1 または 2) で解決される可能性があります。 ただし、この方法を使用することは推奨しません。 この方法は、サービスの作業負荷を増加させるからです。
  - IoT Platform では、デバイスシャドウに制御コマンドを保存していて、このコマンドにはアプリケーションが送信したときのタイムスタンプが含まれています。 デバイスが IoT Platform に再接続したとき、これらのコマンドを取得してタイムスタンプをチェックし、コマンドを実行するかどうかを判断します。
- オフライン状態のデバイスは、アプリケーションからコマンドを受信できません。 接続が回復したとき、デバイスはデバイスシャドウに記録されたコマンドのタイムスタンプをチェックし、有効なコマンドだけを実行します。

5.2 デバイスシャドウ JSON 形式

デバイスシャドウ JSON ファイルの形式

形式は次のとおりです。

```
{
"state": {
"desired": {
"attribute1": integer2,
"attribute2": "string2",
...
"attributeN": boolean2
},
"reported": {
"attribute1": integer1,
"attribute2": "string1",
...
"attributeN": boolean1
}
},
"metadata": {
"desired": {
"attributes": {
"timestamp": timestamp
},
"attributes": {
"timestamp": timestamp
},
...
"attributes": {
"timestamp": timestamp
```

```
}
},
"reported": {
"attributes": {
"timestamp": timestamp
},
"attributes": {
"timestamp": timestamp
},
...
"attributes": {
"timestamp": timestamp
}
}
},
"timestamp": timestamp,
"version": version
}
```

JSON のプロパティについては、表 5-1 : JSON のプロパティ で説明します。

表 5-1 : JSON のプロパティ

プロパティ	説明
desired	デバイスに対して希望するステータス。 アプリケーションは、デバイスにアクセスせずに、デバイスの desired プロパティを書き込みます。
reported	デバイスが報告したステータス。 デバイスは、reported プロパティにデータを書き込むことで、最新のステータスを報告します。 アプリケーションは、このプロパティを読み取ることによってデバイスのステータスを取得します。
metadata	デバイスシャドウサービスは、デバイスシャドウ JSON ファイルの更新に従って、メタデータ (metadata) を自動的に更新します。 デバイスシャドウ JSON ファイルの State メタデータには、各プロパティのタイムスタンプ (timestamp) が含まれています。 タイムスタンプは、正確な更新時刻を取得するためのもので、エポック時刻 (1970-01-01 00:00:00 UTC からの経過秒数) で表されます。
timestamp	デバイスシャドウ JSON ファイルの最終更新時刻。

プロパティ	説明
version	デバイスシャドウのバージョン更新をリクエストすると、デバイスシャドウは、リクエストされたバージョンが現在のバージョンよりも新しいかどうかを確認します。 リクエストされたバージョンが現在のものよりも新しい場合、デバイスシャドウは、リクエストされたバージョンに更新します。 そうでない場合、デバイスシャドウはリクエストを却下します。 バージョン番号は、バージョンの更新に従って増加し、最新のデバイスシャドウ JSON ファイルのバージョンを維持します。

デバイスシャドウ JSON ファイルの例:

```
{
"state" : {
"desired" : {
"color" : "RED",
"sequence" : [ "RED", "GREEN", "BLUE" ]
},
"reported" : {
"color" : "GREEN"
}
},
"metadata" : {
"desired" : {
"color" : {
"timestamp" : 1469564492
},
"sequence" : {
"timestamp" : 1469564492
}
},
"reported" : {
"color" : {
"timestamp" : 1469564492
}
}
},
"timestamp" : 1469564492,
"version" : 1
}
```

空のプロパティ

- デバイスシャドウ JSON ファイルには、desired ステータスが指定された場合にのみ、desired プロパティが含まれます。 次のデバイスシャドウ JSON ファイルには desired プロパティが含まれていませんが、有効です。

```
{
"state" : {
```

```
"reported" : {
"color" : "red",
}
},
"metadata" : {
"reported" : {
"color" : {
"timestamp" : 1469564492
}
}
},
"timestamp" : 1469564492,
"version" : 1
}
```

- 次のデバイスシャドウ JSON ファイルには reported プロパティが含まれていませんが、有効です。

```
{
"state" : {
"desired" : {
"color" : "red",
}
},
"metadata" : {
"desired" : {
"color" : {
"timestamp" : 1469564492
}
}
},
"timestamp" : 1469564492,
"version" : 1
}
```

配列

デバイスシャドウ JSON ファイルは、配列を使用することができます。更新が必要なときには、配列全体を更新する必要があります。

- 初期状態:

```
{
"reported" : { "colors" : ["RED", "GREEN", "BLUE" ] }
}
```

- 更新:

```
{
"reported" : { "colors" : ["RED"] }
}
```

- 最終状態:

```
{
```

```
"reported" : { "colors" : ["RED"] }
}
```

5.3 デバイスシャドウのデータストリーム

IoT Platform は、データ伝送を可能にするために、各デバイスに対して 2 つのトピックをあらかじめ定義しています。 定義済みトピックの形式は固定です。

- トピック: /shadow/update/${YourProductKey}/${YourDeviceName}

  デバイスとアプリケーションは、このトピックにメッセージをパブリッシュします。 IoT Platform がこのトピックからメッセージを受信すると、メッセージ内のステータス情報を抽出し、デバイスシャドウのステータスを更新します。
- トピック: /shadow/get/${YourProductKey}/${YourDeviceName}

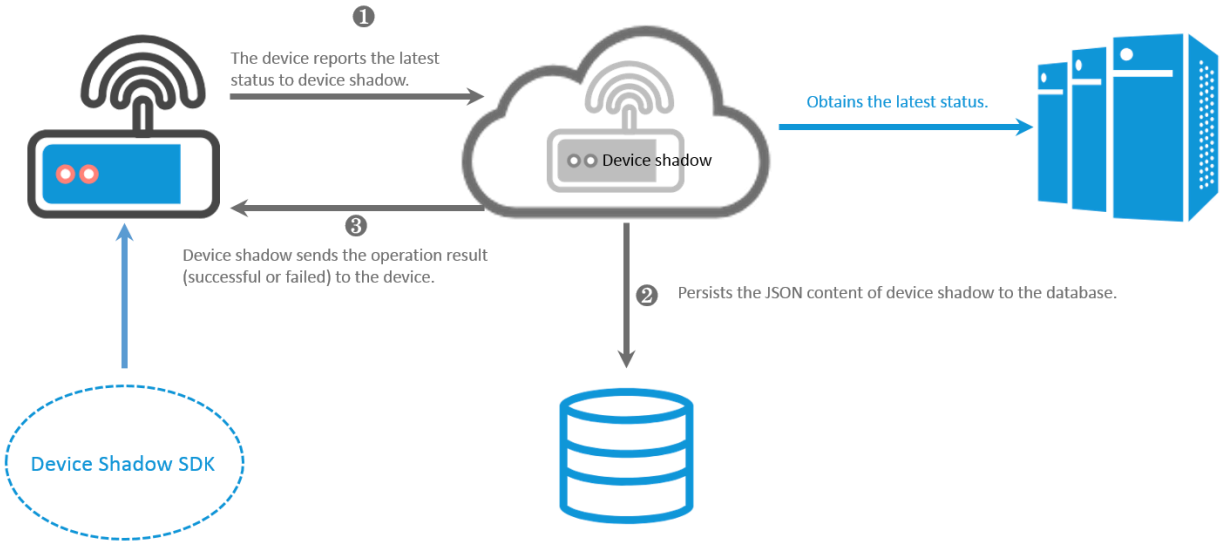
  デバイスシャドウはこのトピックのステータスを更新し、デバイスはこのトピックからのメッセージをサブスクライブします。

例として、プロダクト bulb_1 の lightbulb デバイスを取り上げて、デバイス、デバイスシャドウ、およびアプリケーション間の通信を説明します。 次の例では、ProductKey は 10000、DeviceName は lightbulb です。 デバイスは、QoS 1 の方法を使用して、メッセージをパブリッシュし、2 つのカスタムトピックのメッセージをサブスクライブします。

デバイスによるステータスの自動報告

フローチャートを「図 5-1：デバイスによるステータスの自動報告」に示します。

図 5-1：デバイスによるステータスの自動報告



1. lightbulb がオンラインのとき、デバイスはトピック /shadow/update/10000/lightbulb を使用して、最新のステータスをデバイスシャドウに報告します。

   JSON メッセージの形式:

```
{
"method": "update",
"state": {
"reported": {
"color": "red"
}
},
"version": 1
}
```

   JSON パラメーターを「表 5-2：パラメーターの説明」で説明します。

   表 5-2：パラメーターの説明

パラメーター	説明
method	デバイスまたはアプリケーションがデバイスシャドウをリクエストしたときの操作タイプ。 ステータスを更新する場合、このパラメーター **method** は必須で、update に設定する必要があります。
state	デバイスがデバイスシャドウに送信するステータス情報。 **reported** フィールドは必須です。 ステータス情報は、デバイスシャドウの reported フィールドと同期しています。
version	リクエストに含まれているバージョン情報。 新バージョンが現在のバージョンより新しい場合、デバイスシャドウはリクエストを受け入れて、指定されたバージョンに更新します。

2. デバイスシャドウがデバイス lightbulb によって報告されたステータスを受け入れると、デバイスシャドウの JSON ファイルは正常に更新されます。

```
{
"state" : {
"reported" : {
"color" : "red"
}
},
"metadata" : {
"reported" : {
"color" : {
"timestamp" : 1469564492
}
```

```
}
},
"timestamp" : 1469564492
"version" : 1
}
```

3. デバイスシャドウが更新されると、トピック /shadow/get/10000/lightbulb にメッセージを送信して、結果をデバイス (lightbulb) に返します。

   - 更新が成功すると、メッセージは次のようになります。

     ```
     {
     "method":"reply",
     "payload": {
     "status":"success",
     "version": 1
     },
     "timestamp": 1469564576
     }
     ```

   - 更新中にエラーが発生した場合、メッセージは次のようになります。

     ```
     {
     "method":"reply",
     "payload": {
     "status":"error",
     "content": {
     "errorcode": "${errorcode}",
     "errormessage": "${errormessage}"
     }
     },
     "timestamp": 1469564576
     }
     ```

   エラーコードを「表 5-3 : エラーコード」で説明します。

表 5-3 : エラーコード

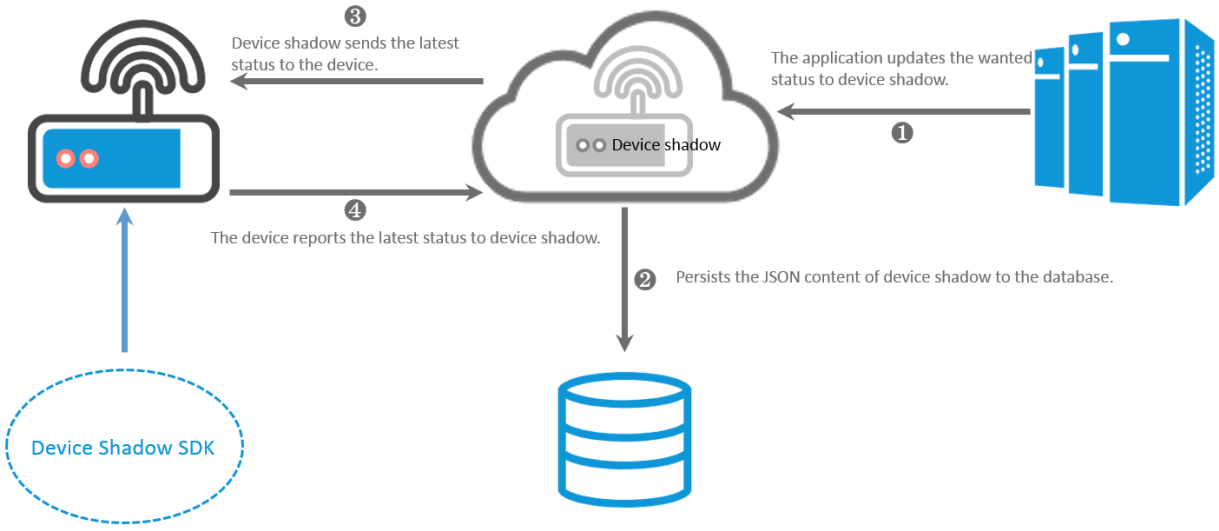
errorCode	errorMessage
400	JSON ファイルが正しくありません。
401	method フィールドが見つかりません。
402	state フィールドが見つかりません。
403	無効な version フィールドです。
404	reported フィールドが見つかりません。
405	reported フィールドは空です。
406	method フィールドが無効です。
407	JSON ファイルが空です。

errorCode	errorMessage
408	reported フィールドには、128 を超える属性が含まれています。
409	バージョンが競合しています。
500	サーバー例外です。

アプリケーションによるデバイスステータスの変更

フローチャートを「図 5-2：アプリケーションによるデバイスステータスの変更」に示します。

図 5-2：アプリケーションによるデバイスステータスの変更



1. アプリケーションは lightbulb のステータスを変更するために、デバイスシャドウにコマンドを送信します。

   アプリケーションはトピック `/shadow/update/10000/lightbulb/` にメッセージを送信します。 メッセージは次のとおりです:

```
{
"method": "update",
"state": {
"desired": {
"color": "green"
}
},
"version": 2
}
```

2. アプリケーションは、デバイスシャドウ JSON ファイルを更新するための更新リクエストを送信します。 デバイスシャドウ JSON ファイルは次のように変更されます。

```
{
"state" : {
```

```
"reported" : {
"color" : "red"
},
"desired" : {
"color" : "green"
}
},
"metadata" : {
"reported" : {
"color" : {
"timestamp" : 1469564492
}
},
"desired" : {
"color" : {
"timestamp" : 1469564576
}
}
},
"timestamp" : 1469564576,
"version" : 2
}
```

3. 更新後、デバイスシャドウはトピック /shadow/get/10000/lightbulb にメッセージを送信し、更新の結果をデバイスに返します。 結果メッセージは、デバイスシャドウによって作成されます。

```
{
"method":"control",
"payload": {
"status":"success",
"state": {
"reported": {
"color": "red"
},
"desired": {
"color": "green"
}
},
"metadata": {
"reported": {
"color": {
"timestamp": 1469564492
}
},
"desired" : {
"color" : {
"timestamp" : 1469564576
}
}
}
},
"version": 2,
"timestamp": 1469564576
}
```

4. デバイス lightbulb がオンラインで、トピック /shadow/get/10000/lightbulb をサブスクライブしている場合、デバイスはメッセージを受信し、リクエストファイルの **desired** フィール

ドに従ってその色を緑色に変更します。 デバイスはステータスを更新した後、クラウドに最新のステータスを報告します。

```
{
method": "update",
"state": {
"reported": {
"color": "green"
}
},
"version": 3
}
```

コマンドが期限切れであることをタイムスタンプが示している場合は、更新しません。

5. 最新のステータスが正常に報告された後、デバイスクライアントはトピック `/shadow/update/10000/lightbulb` にメッセージを送信して、desired フィールドのプロパティを空にします。 メッセージは次のとおりです。

```
{
"method": "update",
"state": {
"desired":"null"
},
"version": 4
}
```

6. ステータスが報告された後、デバイスシャドウは同期して更新されます。 デバイスシャドウ JSON ファイルは次のとおりです:

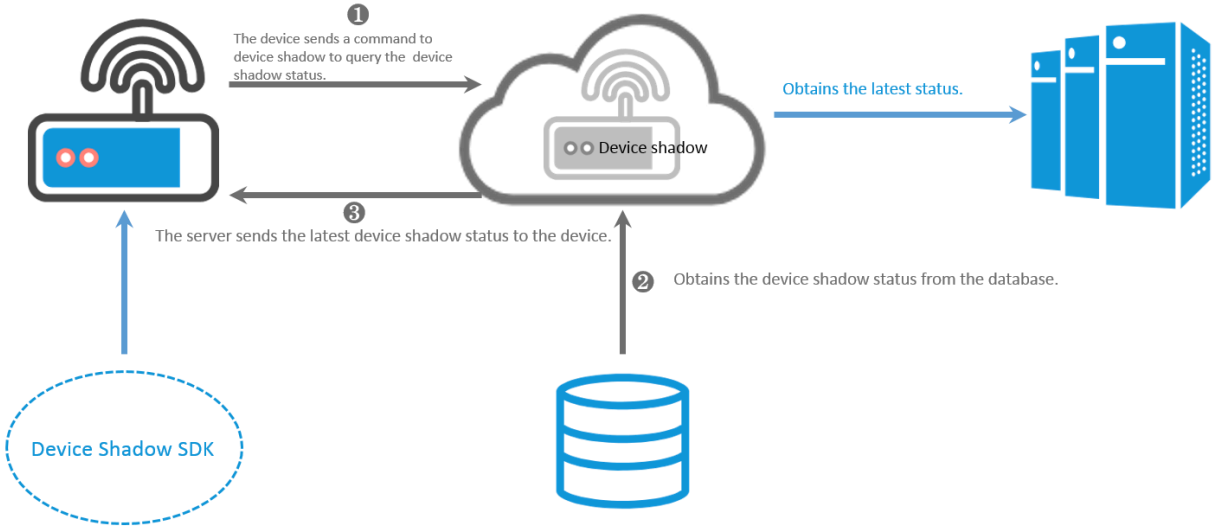
```
{
"state" : {
"reported" : {
"color" : "green"
}
},
"metadata" : {
"reported" : {
"color" : {
"timestamp" : 1469564577
}
},
"desired" : {
"timestamp" : 1469564576
}
},
"version" : 4
```

```
}
```

デバイスからデバイスシャドウに対するリクエスト

フローチャートを「図 5-3：デバイスからデバイスシャドウに対するリクエスト」に示します。

図 5-3：デバイスからデバイスシャドウに対するリクエスト



1. デバイス lightbulb は、トピック /shadow/update/10000/lightbulb にメッセージを送信して、デバイスシャドウに保存されている最新のステータスを取得します。 メッセージは次のとおりです。

```
{
"method": "get"
}
```

2. デバイスシャドウが上記のメッセージを受信すると、トピック/shadow/get/10000/lightbulb にメッセージを送信します。 メッセージは次のとおりです:

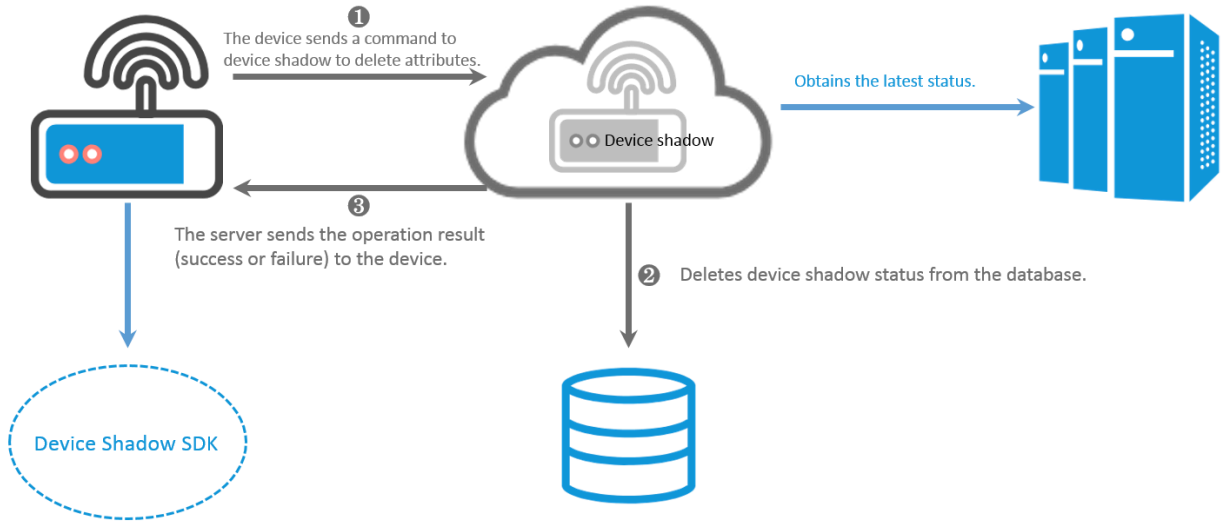
```
{
"method":"reply",
"payload": {
"status":"success",
"state": {
"reported": {
"color": "red"
},
"desired": {
"color": "green"
}
},
"metadata": {
"reported": {
"color": {
"timestamp": 1469564492
}
```

```
},
"desired": {
"color": {
"timestamp": 1469564492
}
}
}
},
"version": 2,
"timestamp": 1469564576
}
```

デバイスによるデバイスシャドウ属性の削除

フローチャートを「図 5-4：デバイスシャドウ属性の削除」に示します。

図 5-4：デバイスシャドウ属性の削除



デバイス lightbulb は、デバイスシャドウに保存されている、指定された属性を削除します。 デバイスは JSON メッセージをトピック /shadow/update/10000/lightbulb に送信します。 次の例のメッセージをご参照ください。

属性を削除するには、**method** の値を delete に設定し、属性の値を null に設定します。

- 属性を 1 つ削除します。

```
{
"method": "delete",
"state": {
"reported": {
"color": "null",
"temperature":"null"
}
},
"version": 1
```

```
}
```

- 属性をすべて削除します。

```
{
"method": "delete",
"state": {
"reported":"null"
},
"version": 1
}
```

6 ファイルの管理

IoT Platform を使用すると、デバイスは HTTP/2 チャネル経由でファイルを Alibaba Cloud IoT Platform サーバーにアップロードしてストレージに保存できます。 ファイルをアップロードした後、IoT Platform コンソールでファイルをダウンロードして削除できます。

- デバイスが IoT プラットフォームに接続されていること。

  デバイス SDK 開発の詳細については、「Link Kit SDK ドキュメント」をご参照ください。
- HTTP/2 ファイルアップロード機能は、デバイス上にコンパイルされ設定されます。

1. IoT Platform コンソールにログインします。
2. 左側のナビゲーションウィンドウで **[デバイス]** > **[デバイス]** を選択し、対応するデバイスの横にある **[表示]** をクリックします。
3. **[デバイス情報]** ページで、 **[ファイルの管理]** タブをクリックします。

   [ファイルの管理] タブページで、HTTP/2 チャネル経由でデバイスによってアップロードされたファイルを表示できます。

   > 注 :
   > 各 Alibaba Cloud アカウントの IoT Platform サーバーに保存できる最大ファイルサイズは 1 GB です。 各デバイスに保存できるファイルの最大数は 1,000 です。

   アップロードされたファイルに対して次の操作を実行できます。

操作	説明
ダウンロード	ファイルをローカルデバイスにダウンロードします。
削除	ファイルを削除します。

   コンソールでのファイル管理に加えて、次のクラウド API 操作を呼び出してファイルを照会または削除することもできます。#unique_25、#unique_26、および #unique_27

7 ゲートウェイとサブデバイス

7.1 ゲートウェイとサブデバイス

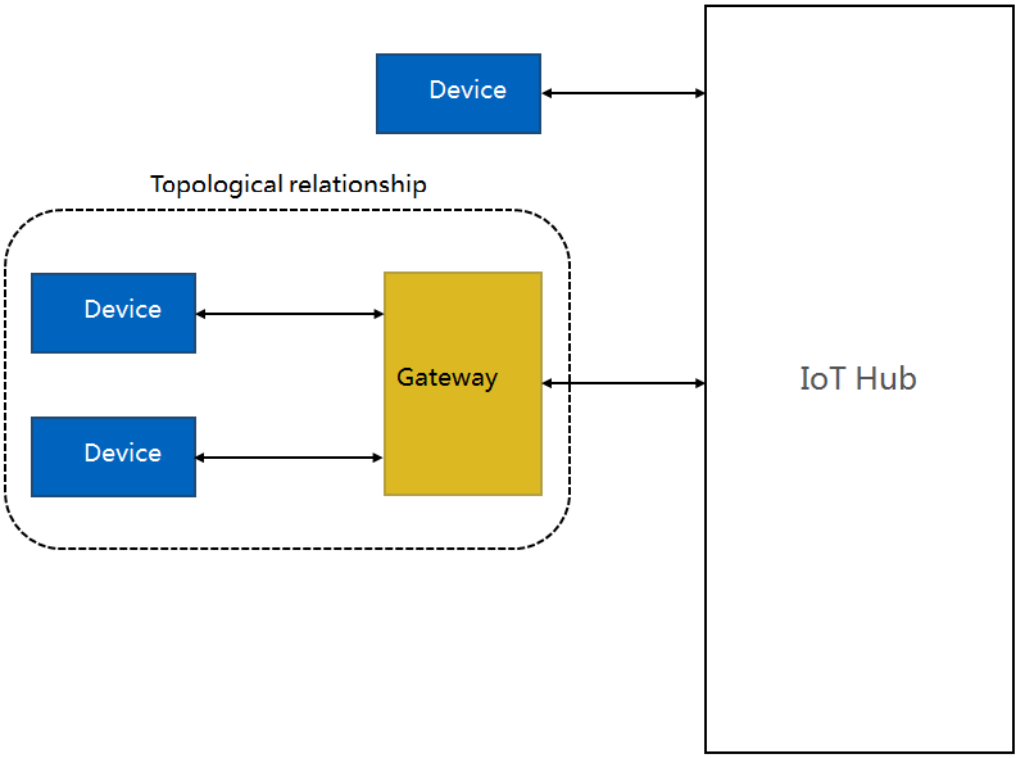
IoT Platform では、デバイスを直接接続することも、IoT Platform に接続するゲートウェイにサブデバイスとしてマウントすることもできます。

ゲートウェイとデバイス

プロダクトを作成するときは、プロダクトのデバイス用にノードタイプを選択する必要があります。 現在、IoT Platform は、デバイスとゲートウェイの 2 つのノードタイプをサポートしています。

- デバイス: このノードタイプのデバイスはサブデバイスとともにマウントすることはできませんが、IoT Platform に直接接続するか、サブデバイスとしてゲートウェイにマウントすることはできます。
- ゲートウェイ: このタイプのデバイスは IoT Platform に直接接続することができ、サブデバイスとともにマウントできます。 ゲートウェイは、サブデバイスを管理し、サブデバイスとのトポロジ関係を維持し、これらのトポロジ関係を IoT Platform に同期させることができます。

次の図に、ゲートウェイとそのサブデバイスとの間のトポロジ関係を示します。



ゲートウェイとサブデバイスから IoT Platform への接続

ゲートウェイが IoT Platform に接続されると、ゲートウェイはサブデバイスとのトポロジ関係を IoT Platform に同期させます。 ゲートウェイは、すべてのサブデバイスに対して、デバイス認証、メッセージ送信、命令受信など、IoT Platform との通信をサポートします。 つまり、サブデバイスは対応するゲートウェイによって管理されます。

1. ゲートウェイから IoT Platform へ接続する方法の詳細については、『リンクキット SDK』をご参照ください。
2. 次の 2 つの方法のいずれかを使用して、サブデバイスを IoT Platform に接続することができます。
   - デバイス固有証明書認証 方式 この方式では、デバイス証明書 (ProductKey、DeviceName、および DeviceSecret) を物理サブデバイスにインストールしてから、サブデバイスを IoT Platform に接続する必要があります。
   - プロダクト固有証明書認証 方式 この方式では、プロダクト詳細ページの [動的登録] を有効にして、IoT Platform コンソール内でデバイスを登録する必要があります。 物理的なサブデバイスが接続されているとき、ゲートウェイはサブデバイスのために IoT Platform への接続要求を開始します。 その後、IoT Platform はサブデバイス情報を検証します。 検証に合格すると、IoT Platform はサブデバイスにデバイスシークレットを割り当てます。 サブデバイスは、IoT Platform に正常に接続するために必要なすべての情報 (ProductKey、DeviceName、および DeviceSecret) を受信します。

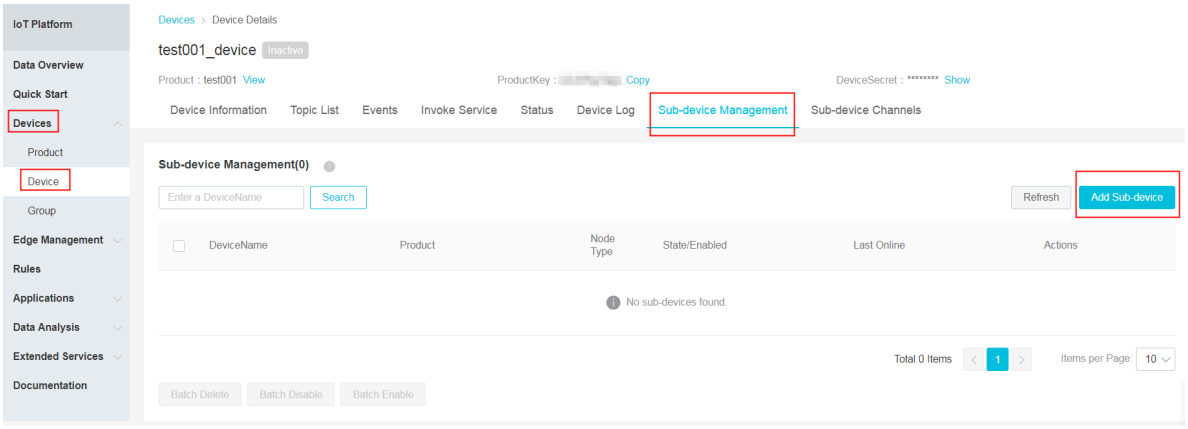
7.2 サブデバイス管理

サブデバイスをゲートウェイデバイスに追加し、TSL および サブデバイスの拡張サービス情報をゲートウェイに送信することができます。

Procedure

1. 左側のナビゲーションペインで、**[デバイス管理] > [デバイス]** をクリックします。
2. **[デバイス管理]** ページで、サブデバイスを追加するゲートウェイデバイスを見つけ、**[表示]** をクリックします。**[デバイス情報]** ページに移動します。

3. **[サブデバイス管理] > [サブデバイスの追加]** をクリックします。



4. ダイアログボックスにサブデバイスの情報を入力します。

パラメーター	説明
プロダクト	サブデバイスが属するプロダクトの名前を選択します。
デバイス名	サブデバイスとして追加するデバイスの名前を選択します。

What's next

ゲートウェイとサブデバイス間のトポロジー関係が作成されました。サブデバイスの詳細ページで、ゲートウェイデバイス情報を確認できます。

8 拡張サービス

9 Alink プロトコルに基づくデバイス開発

9.1 Alink プロトコル

ここでは、Alink プロトコルデータをカプセル化し、Alink プロトコルを使用して、デバイスから IoT Platform への接続を確立する方法について説明します。

Alink プロトコルは、IoT 開発用のデータ交換規格で、デバイスと IoT Platform 間の通信が可能になります。 プロトコルは Alink JSON 形式のデータを交換します。

以降のセクションでは、Alink プロトコル使用時のデバイス接続手順およびデータパススループロセス (アップストリームとダウンストリーム) について説明します。

デバイスの IoT Platform への接続

次の図に示すように、直接接続されたデバイス、またはサブデバイスとして、デバイスを IoT Platform に接続できます。 接続プロセスには、デバイスの登録、接続の確立、データの報告などの重要な手順が含まれます。

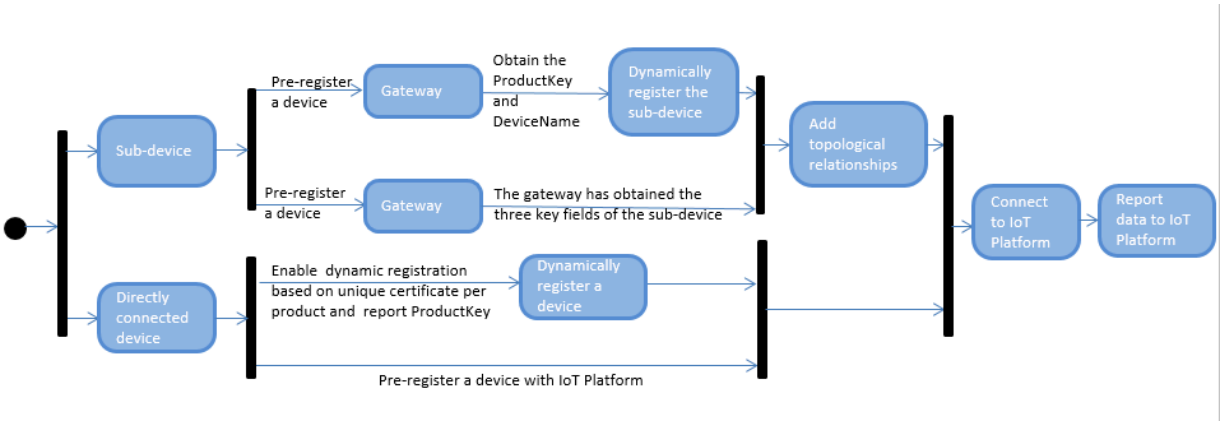
直接接続されたデバイスは、次の方法で IoT Platform に接続できます。

- #unique_30 が有効になっている場合は、認証用の 3 つのキーフィールド (ProductKey、DeviceName、DeviceSecret) を物理デバイスにインストールし、デバイスを IoT Platform に接続し、IoT Platform にデータを報告します。
- #unique_31 に基づく動的登録が有効になっている場合、認証用のプロダクト証明書 (ProductKey と ProductSecret) を物理デバイスにインストールし、デバイスを IoT Platform に接続し、IoT Platform にデータを報告します。

ゲートウェイは、サブデバイスの接続プロセスを開始します。 サブデバイスは、次の方法で IoT Platform に接続できます。

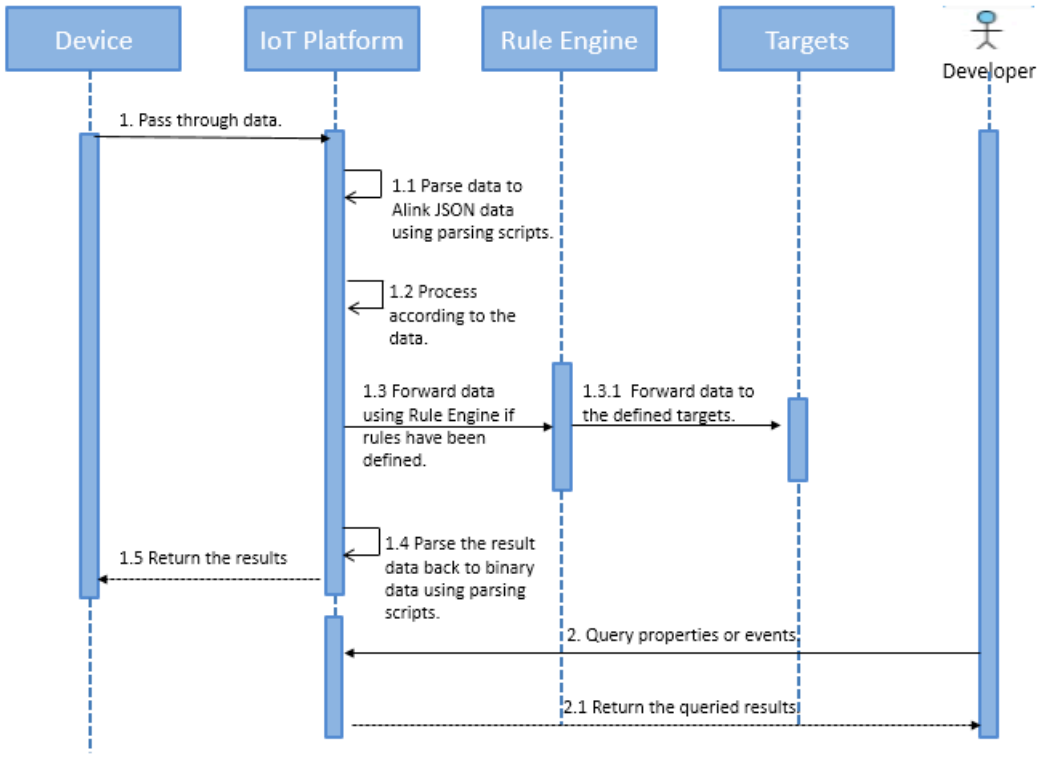
- #unique_30 が有効になっている場合、認証用の ProductKey、DeviceName、DeviceSecret を物理サブデバイスにインストールします。 サブデバイスは、これら 3 つのキーフィールドをゲートウェイに送信します。ゲートウェイはトポロジー関係を追加し、ゲートウェイ接続チャネルを介してサブデバイスのデータを送信します。
- 動的登録が有効になっている場合、認証用の ProductKey を事前に物理サブデバイスにインストールします。 サブデバイスは ProductKey と DeviceName をゲートウェイに送信します。ゲートウェイは ProductKey と DeviceName を IoT Platform に転送します。 IoT Platform は受け取った DeviceName を検証し、DeviceSecret をサブデバイスに送信します。 サブデバイ

スは取得した ProductKey、DeviceName、DeviceSecret をゲートウェイに送信します。ゲートウェイはトポロジー関係を追加し、ゲートウェイ接続チャネルを介して IoT Platform にデータを送信します。



デバイスからプロパティまたはイベントを報告

- パススルーデータ (データ型は、"パススルー/カスタム")



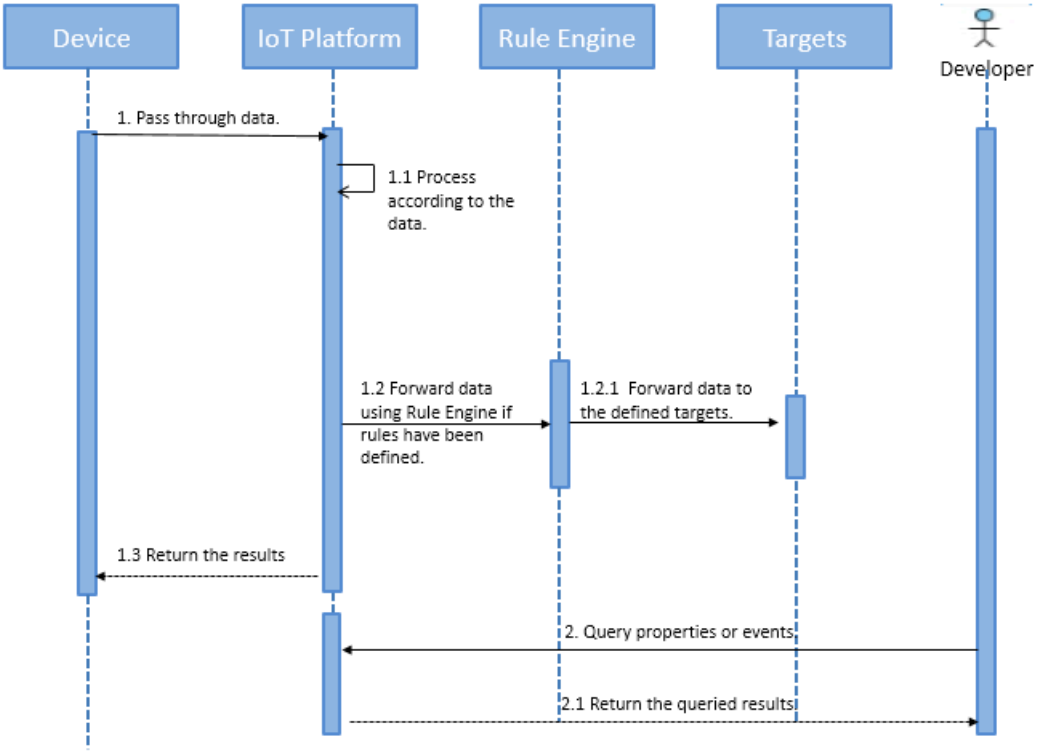
1. デバイスはパススルーデータの Topic を使用して、IoT Platform にバイナリデータを報告します。
2. IoT Platform は、IoT Platform コンソール上で送信されたデータ解析スクリプトを使用して、受信したデータを解析します。 スクリプト内の rawDataToProtocol メソッドを呼び出して、デバイスから報告されたバイナリデータを、処理に使用される Alink JSON データに
3. 変換します。

   データ転送のルールを設定している場合、Alink JSON データはこのルールに基づいてターゲットに転送されます。
4. IoT Platform は、IoT Platform コンソール上で送信されたデータ解析スクリプトを使用して、返されたデータをバイナリデータに解析します。
5. IoT Platform は、変換されたバイナリデータをデバイスにプッシュします。

> 注：
>
> - ルールエンジンによって転送されるデータは、データ解析スクリプトで解析されたデータです。

- ルールエンジンを使用してデータ転送を設定する場合、デバイスのプロパティを取得するには Topic: /sys/{productKey}/{deviceName}/thing/event/property/post を使用します。
- ルールエンジンを使用してデータ転送を設定する場合、デバイスのイベントを取得するには Topic: /sys/{productKey}/{deviceName}/thing/event/{tsl.event.identifier}/post を使用します。

- パススルー (Alink JSON) 以外のデータ



1. デバイスは、パススルー以外のデータ用の Topic を使用して、Alink JSON データを IoT Platform に報告します。
2. IoT Platform は受信データを処理します。

   データ転送のルールを設定している場合、データはルールに従ってターゲットに転送されます。
3. IoT Platform は結果をデバイスに返します。

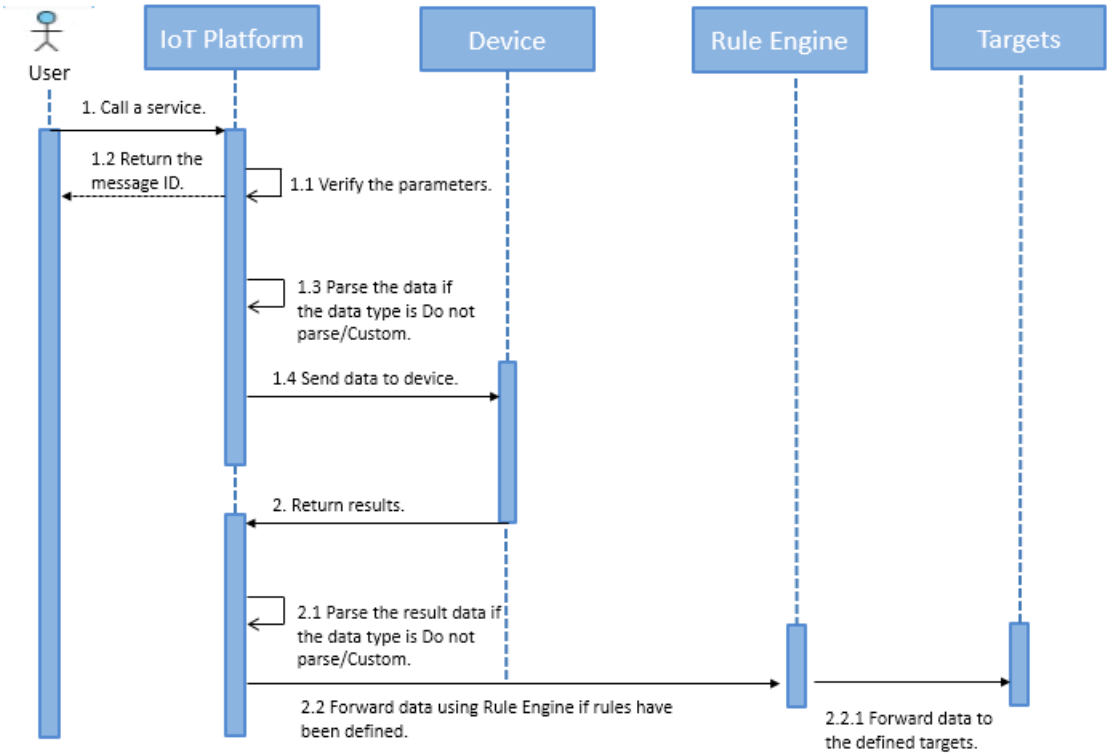
注：

- ルールエンジンを使用してデータ転送を設定する場合、デバイスのプロパティを取得するには Topic: /sys/{productKey}/{deviceName}/thing/event/property/post を使用します。

- ルールエンジンを使用してデータ転送を設定する場合、デバイスのイベントを取得するには Topic: /sys/{productKey}/{deviceName}/thing/event/{tsl.event.identifier}/post を使用します。

デバイスサービスの呼び出し、またはデバイスプロパティの設定

- デバイスサービスの呼び出し、またはデバイスプロパティの設定 (非同期)



1. ユーザーは非同期呼び出しメソッドを使用して、デバイスプロパティを設定するか、デバイスサービスを呼び出します。
2. IoT Platform はパラメーターを検証します。
3. IoT Platform は、非同期呼び出しメソッドを使用して、リクエストを処理し結果を返します。 呼び出しが成功した場合、デバイスに送信されるメッセージ ID がレスポンスに含まれます。

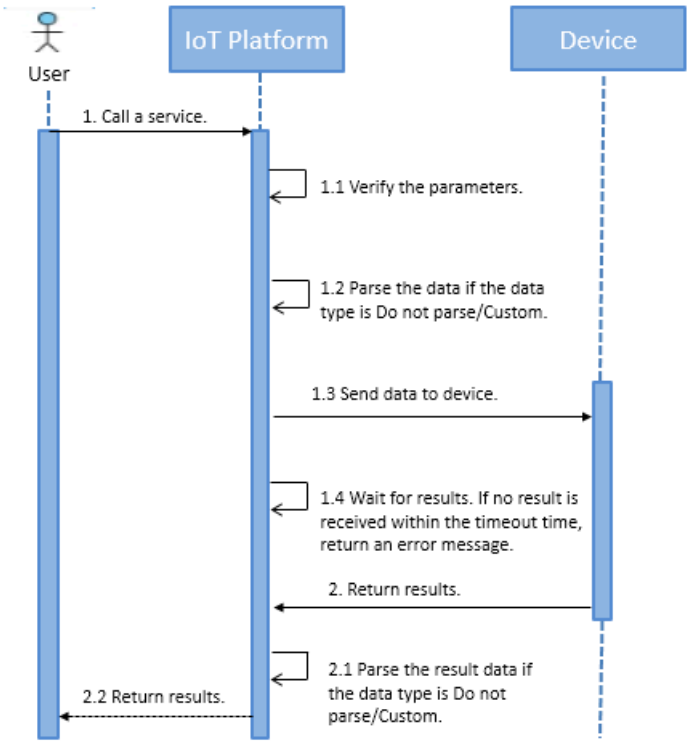
   データ型がパススルー (パススルー/カスタム) の場合、IoT Platform はデータ解析スクリプト内の protocolToRawData メソッドを呼び出し、データを変換してからデバイスに送信します。
4. IoT Platform はデータをデバイスに送信し、デバイスはリクエストを非同期に処理します。
   - データがパススルー (パススルー/カスタム) データの場合、パススルーデータ用の Topic を使用します。
   - データがパススルー以外 (Alink JSON) のデータの場合、パススルー以外のデータ用の Topic を使用します。
5. デバイスがリクエスト操作を完了した後、結果を返します。

- データ型がパススルー (パススルー / カスタム) の場合、IoT Platform はデータ解析スクリプト内の rawDataToProtocol メソッドを呼び出し、デバイスから返されるデータを変換します。
- データ転送ルールを設定している場合、データはルールに従ってターゲットに転送されます。

 注：

- ルールエンジンを使用してデータ転送を設定する場合、呼び出し結果を取得するには Topic: /sys/{productKey}/{deviceName}/thing/downlink/reply/message を使用します。
- データ型がパススルー (パススルー/カスタム) の場合、ルールエンジンによって転送されたデータは、データ解析スクリプトによって解析されたデータです。

- デバイスサービスの呼び出しと、デバイスプロパティの設定 (同期)



1. ユーザーは同期呼び出しメソッドを使用して、デバイスサービスを呼び出します。
2. IoT Platform はパラメーターを検証します。

   データ型がパススルー (パススルー/カスタム) の場合、IoT Platform はデータ解析スクリプト内の protocolToRawData メソッドを呼び出し、データを変換してからデバイスに送信します。
3. 同期呼び出しメソッドでは、IoT Platform が RRPC Topic を呼び出してリクエストデータをデバイスに送信し、デバイスが結果を返すのを待ちます。
4. デバイスは、リクエスト操作の完了後に結果を返します。 結果をタイムアウト期間内に受信しない場合、タイムアウトエラーが返されます。

   データ型がパススルー (パススルー/カスタム) の場合、IoT Platform はデータ解析スクリプト内の rawDataToProtocol メソッドを呼び出し、デバイスが返すデータを変換します。
5. IoT Platform は結果をユーザーに返します。

9.2 デバイス ID 登録

デバイスを IoT Platform に接続する前に、デバイス ID を IoT Platform に登録する必要があります。

ID 登録には以下の方法があります。

- デバイス固有証明書: IoT Platform 上でデバイスの ProductKey、DeviceName、DeviceSecret を取得し、これらを一意の識別子として使用します。 デバイスのファームウェアにこれら 3 つのキーフィールドをインストールします。 デバイスが IoT Platform に接続された後、IoT Platform との通信を開始します。
- 動的登録: 直接接続されたデバイスの場合、プロダクト固有証明書による認証に基づく動的登録を実行します。サブデバイスの場合、動的登録を実行します。
  - プロダクト固有証明書による認証に基づき、直接接続されたデバイスを動的に登録するには、以下の手順に従います。
    1. IoT Platform コンソールで、デバイスを事前登録し、ProductKey と ProductSecret を取得します。 デバイスを事前登録する際、デバイスの MAC アドレスやシリアル番号など、DeviceName としてデバイスから直接読み取ることができるデバイス情報を使用します。
    2. コンソールで動的登録を有効にします。
    3. デバイスファームウェア上にプロダクト証明書をインストールします。
    4. IoT Platform でデバイスが認証されます。 デバイスが認証されると、DeviceSecret が割り当てられます。
    5. デバイスは、ProductKey、DeviceName、DeviceSecret を使用して、IoT Platform への接続を確立します。
  - サブデバイスを動的に登録するには、以下の手順に従います。
    1. IoT Platform コンソールで、サブデバイスを事前登録し、ProductKey を取得します。 サブデバイスを事前登録する際、MAC アドレスや SN など、DeviceName としてサブデバイスから直接読み取ることができるデバイス情報を使用します。
    2. コンソールで動的登録を有効にします。
    3. サブデバイスのファームウェア上、またはゲートウェイ上に ProductKey をインストールします。
    4. ゲートウェイはサブデバイスの代わりに IoT Platform で認証されます。

サブデバイスの動的登録

アップストリーム

- Topic: /sys/{productKey}/{deviceName}/thing/sub/register
- Reply topic: /sys/{productKey}/{deviceName}/thing/sub/register_reply

リクエストメッセージ

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554"
    }
  ],
  "method": "thing.sub.register"
}
```

レスポンスメッセージ

```
{
  "id": "123",
  "code": 200,
  "data": [
    {
      "iotId": "12344",
      "productKey": "1234556554",
      "deviceName": "deviceName1234",
      "deviceSecret": "xxxxxx"
    }
  ]
}
```

パラメーターの説明

パラメーター	型	説明
id	String	メッセージ ID。 将来の使用のために予約されたパラメーターの値
version	String	プロトコルのバージョン。 現在、値は 1.0 のみ。
params	List	動的登録に使用されるパラメーター。
deviceName	String	サブデバイス名。
productKey	String	サブデバイスが属するプロダクトの ID。
iotId	String	サブデバイスの一意識別子。
deviceSecret	String	DeviceSecret キー。

パラメーター	型	説明
method	String	リクエスト方法。
code	Integer	結果コード。

エラーメッセージ

エラーコード	エラーメッセージ	説明
460	request parameter error	リクエストパラメーターが不正。
6402	topo relation cannot add by self	デバイスをサブデバイスとして追加できません。
401	request auth error	署名の検証に失敗しました。

プロダクト固有証明書の認証に基づき直接接続されたデバイスの動的登録

直接接続されたデバイスは、HTTP リクエストを送信して動的登録を実行します。 プロダクト固有証明書の認証に基づく動的登録を有効にしていることをコンソールで確認します。

- URL テンプレート: https://iot-auth.cn-shanghai.aliyuncs.com/auth/register/device
- HTTP メソッド: POST

リクエストメッセージ

```
POST /auth/register/device  HTTP/1.1
Host: iot-auth.cn-shanghai.aliyuncs.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 123
productKey=1234556554&deviceName=deviceName1234&random=567345&sign=
adfv123hdfdh&signMethod=HmacMD5
```

レスポンスメッセージ

```
{
  "code": 200,
  "data": {
    "productKey": "1234556554",
    "deviceName": "deviceName1234",
    "deviceSecret": "adsfweafdsf"
  },
  "message": "success"
}
```

パラメーターの説明

パラメーター	型	説明
productKey	String	デバイスが属するプロダクトの ID。

パラメーター	型	説明
deviceName	String	デバイス名。
random	String	ランダムな番号。
sign	String	署名。
signMethod	String	署名方法。 サポートされる方法は、hmacmd5、hmacsha1、hmacsha256 です。
code	Integer	結果コード。
deviceSecret	String	DeviceSecret キー。

パラメーターの署名

IoT Platform に報告されるパラメーターは、**sign** と **signMethod** を除いてすべて署名されます。署名パラメーターをアルファベット順に並べ替え、連結記号なしでパラメーターと値を連結します。

その後、**signMethod** で指定されたアルゴリズムを使用してパラメーターに署名します。

例:

```
sign = hmac_sha1(productSecret, deviceNamedeviceName1234productKey1234556554random123)
```

9.3 トポロジー関係の追加

サブデバイスが IoT Platform に登録されてから IoT Platform に接続するまでの間に、ゲートウェイはゲートウェイとサブデバイスのトポロジー関係を IoT Platform に報告します。

IoT Platform は接続の間に ID とトポロジー関係を検証します。 検証に成功すると、IoT Platform はサブデバイスとの論理接続を確立し、ゲートウェイの物理接続と論理接続を関連付けます。 サブデバイスは、直接接続されたデバイスと同じプロトコルを使用して、データのアップロードとダウンロードを行います。 ゲートウェイ情報をプロトコルに含める必要はありません。

IoT Platform からサブデバイスのトポロジー関係を削除した後、サブデバイスはゲートウェイ経由で IoT Platform に接続することはできません。 トポロジー関係が存在しないため、IoT Platform が認証に失敗します。

サブデバイスのトポロジー関係の追加

アップストリーム

- Topic: /sys/{productKey}/{deviceName}/thing/topo/add

- Reply topic: sys/{productKey}/{deviceName}/thing/topo/add_reply

Alink プロトコル使用時のリクエストデータ形式

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554",
      "sign": "xxxxxx",
      "signmethod": "hmacSha1",
      "timestamp": "1524448722000",
      "clientId": "xxxxxx"
    }
  ],
  "method": "thing.topo.add"
}
```

Alink プロトコル使用時のレスポンスデータ形式

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

パラメーターの説明

パラメーター	型	説明
id	String	メッセージ ID。 将来の使用のために予約されたパラメーター値
version	String	プロトコルのバージョン。 現在、値は 1.0 のみ。
params	List	リクエストの入力パラメーター。
deviceName	String	デバイス名。 値はサブデバイス名です。
productKey	String	プロダクト ID。 値はサブデバイスが属するプロダクトの ID です。

パラメーター	型	説明
sign	String	署名。 署名アルゴリズム。 サーバーに送信されるすべてのパラメーター (**sign** と **signMethod** を除く) を辞書順に並べ、パラメーターと値を順番に連結します (連結記号なし)。 署名方法で指定されたアルゴリズムを使用して、署名パラメーターに署名します。 たとえば、以下のリクエストでは **params** のパラメーターをアルファベット順に並べ、パラメーターに署名します。 `sign= hmac_md5(deviceSecret, clientId123deviceNametestproductKey123timestamp1524448722000)`
signmethod	String	署名方法。 サポートされる方法は、hmacSha1、hmacSha256,、hmacMd5、Sha256 です。
timestamp	String	タイムスタンプ。
clientId	String	サブデバイスの識別子。 このパラメーターは省略可能です。ProductKey または DeviceName と同じ値の可能性があります。
code	Integer	結果コード。 200 という値は、リクエストが成功したことを示します。

エラーメッセージ

エラーコード	エラーメッセージ	説明
460	request parameter error	リクエストパラメーターが不正です。
6402	topo relation cannot add by self	デバイスをサブデバイスとして追加できません。
401	request auth error	署名検証が失敗しました。

サブデバイスのトポロジー関係の削除

ゲートウェイはこの Topic に対しメッセージをパブリッシュし、IoT Platform がゲートウェイとサブデバイス間のトポロジー関係を削除するようにリクエストできます。

アップストリーム

- Topic: /sys/{productKey}/{deviceName}/thing/topo/delete
- Reply topic: /sys/{productKey}/{deviceName}/thing/topo/delete_reply

Alink プロトコル使用時のリクエストデータ形式

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554"
    }
  ],
  "method": "thing.topo.delete"
}
```

Alink プロトコル使用時のレスポンスデータ形式

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

パラメーターの説明

パラメーター	型	説明
id	String	メッセージ ID。 将来の使用のために予約されたパラメーター値。
version	String	プロトコルのバージョン。 現在、値は 1.0 のみ。
params	List	リクエストパラメーター。
deviceName	String	デバイス名。 値はサブデバイス名です。
productKey	String	プロダクト ID。 値はサブデバイスが属するプロダクトの ID。
method	String	リクエストメソッド。
code	Integer	結果コード。 200 という値は、リクエストが成功したことを示します。

エラーメッセージ

エラーコード	エラーメッセージ	説明
460	request parameter error	リクエストパラメーターが不正です。
6100	device not found	デバイスが存在しません。

サブデバイスのトポロジー関係の取得

アップストリーム

- Topic: /sys/{productKey}/{deviceName}/thing/topo/get
- Reply topic: /sys/{productKey}/{deviceName}/thing/topo/get_reply

ゲートウェイは Topic に対してメッセージをパブリッシュし、ゲートウェイと、接続されたサブデバイス間のトポロジー関係を取得します。

Alink プロトコル使用時のリクエストデータ形式

```
{
  "id": "123",
  "version": "1.0",
  "params": {},
  "method": "thing.topo.get"
}
```

Alink プロトコル使用時のレスポンスデータ形式

```
{
  "id": "123",
  "code": 200,
  "data": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554"
    }
  ]
}
```

パラメーターの説明

パラメーター	型	説明
id	String	メッセージ ID。 将来の使用のために予約されたパラメーター値。
version	String	プロトコルのバージョン。 現在、値 1.0 のみ。
params	Object	リクエストパラメーター。 空白のままにできます。

パラメーター	型	説明
method	String	リクエストメソッド。
deviceName	String	サブデバイス名。
productKey	String	サブデバイスのプロダクト ID。
code	Integer	結果コード。 00 という値は、リクエストが成功したことを示します。

エラーメッセージ

エラーコード	エラーメッセージ	説明
460	request parameter error	リクエストパラメーターが不正です。

新たなサブデバイスの報告

アップストリーム

- Topic: /sys/{productKey}/{deviceName}/thing/list/found
- Reply topic: /sys/{productKey}/{deviceName}/thing/list/found_reply

一部のシナリオでは、ゲートウェイが新しいサブデバイスを検出します。 ゲートウェイは新しいサブデバイス情報を IoT Platform へ報告します。 IoT Platform はサブデバイス情報をサードパーティーのアプリケーションに転送し、サードパーティーアプリケーションはゲートウェイに接続するサブデバイスを選択します。

Alink プロトコル使用時のリクエストデータ形式

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554"
    }
  ],
  "method": "thing.list.found"
}
```

Alink プロトコル使用時のレスポンスデータ形式

```
{
  "id": "123",
  "code": 200,
  "data":{}
```

```
}
```

パラメーターの説明

パラメーター	型	説明
id	String	メッセージ ID。 将来の使用のために予約されたパラメーター
version	String	プロトコルのバージョン。 現在、値は 1.0 のみ。
params	Object	リクエストパラメーター。 空白のままにできます。
method	String	リクエストメソッド。
deviceName	String	サブデバイス名。
productKey	String	サブデバイスのプロダクト ID。
code	Integer	結果コード。 200 という値は、リクエストが成功したことを示します。

エラーメッセージ

エラーコード	エラーメッセージ	説明
460	request parameter error	リクエストパラメーターが不正です。
6250	product not found	サブデバイスが属する、指定されたプロダクトが存在しません。
6280	devicename not meet specs	サブデバイス名が無効。 デバイス名は 4〜32 文字の長さで、文字、数字、ハイフン (-)、アンダースコア (_)、アットマーク (@)、ピリオド (.)、およびコロン (:) を含めることができます。

接続されたサブデバイスのトポロジー関係を追加するようゲートウェイに通知

ダウンストリーム

- Topic: /sys/{productKey}/{deviceName}/thing/topo/add/notify
- Reply topic: /sys/{productKey}/{deviceName}/thing/topo/add/notify_reply

IoT Platform はこの Topic にメッセージをパブリッシュして、接続されたサブデバイスのトポロジー関係を追加するようにゲートウェイに通知します。 IoT Platform に新しいサブデバイスを

報告する Topic とともに、このトピックを使用できます。 IoT Platform は、データ交換 Topic をサブスクライブし、ゲートウェイからレスポンスを受け取ります。 データ交換 Topic は /{productKey}/{deviceName}/thing/downlink/reply/message です。

Alink プロトコル使用時のリクエストデータ形式

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554"
    }
  ],
  "method": "thing.topo.add.notify"
}
```

Alink プロトコル使用時のレスポンスデータ形式

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

パラメーターの説明

パラメーター	型	説明
id	String	メッセージ ID。 将来の使用のために予約されたパラメーター値。
version	String	プロトコルバージョン。 現在、値は 1.0 のみ。
params	Object	リクエストパラメーター。 空白のままにできます。
method	String	リクエストメソッド。
deviceName	String	サブデバイス名。
productKey	String	サブデバイスのプロダクト ID。
code	Integer	結果コード。 200 という値は、リクエストが成功したことを示します。

9.4 サブデバイスから IoT Platform への接続

デバイスを IoT Platform に登録し、そのデバイスをサブデバイスとしてゲートウェイデバイスに関連付けてから、これらのサブデバイスを IoT Platform に接続します。

> 注：
> ゲートウェイデバイスは最大 1,500 のサブデバイスを持つことができます。

IoT Platform に接続する前に、サブデバイスが IoT Platform に登録されていることを確認します。 また、ゲートウェイとのトポロジー関係がゲートウェイに追加されていることも確認する必要があります。 IoT Platform は、トポロジー関係に従ってサブデバイスの ID を検証して、サブデバイスがゲートウェイ接続を使用できるかどうか識別します。

サブデバイスから IoT Platform への接続

アップストリーム

- トピック：/ext/session/{productKey}/{deviceName}/combine/login
- 返信トピック：/ext/session/{productKey}/{deviceName}/combine/login_reply

リクエストメッセージ

```
{
  "id": "123",
  "params": {
    "productKey": "123",
    "deviceName": "test",
    "clientId": "123",
    "timestamp": "123",
    "signMethod": "hmacmd5",
    "sign": "xxxxxx",
    "cleanSession": "true"
  }
}
```

レスポンスメッセージ

```
{
  "id":"123",
  "code":200,
  "message":"success"
  "data":""
}
```

パラメーターの説明

パラメーター	データタイプ	説明
id	文字列	メッセージ ID です。 今後使用するものとしてこのパラメーターの値を予約します。

パラメーター	データタイプ	説明
params	オブジェクト	パラメーターをリクエストします。
deviceName	文字列	サブデバイスの名前
productKey	文字列	サブデバイスが属するプロダクトのキー
sign	文字列	サブデバイスの署名です。 サブデバイスはゲートウェイと同じ署名ルールを使用します。 署名アルゴリズム： すべてのパラメーター (**sign** および **signmethod** 以外) をソートしアルファベット順にサーバーにサブミットしてから、パラメーターと値を順番に (接続記号なしで) 接続します。 その後、**signMethod** で指定されたアルゴリズムを使用してパラメーターに署名します。 例： `sign= hmac_md5(deviceSecret, cleanSessiontrueclientId123deviceNametestproductKey123timestamp123)`
signmethod	文字列	署名のメソッドです。 サポートされているメソッドは、hmacSha1、hmacSha256、hmacMd5、および Sha256 です。
timestamp	文字列	タイムスタンプ
clientId	文字列	デバイスクライアントの識別子です。 このパラメーターは、ProductKey パラメーターまたは DeviceName パラメーターと同じ値にすることができます。
cleanSession	文字列	値が true の場合、デバイスがオフラインのときに、QoS=1 方式に基づいて送信されたメッセージが消去されます。
code	整数	結果コードです。 値 200 は、呼び出しが成功したことを示します。
message	文字列	結果メッセージ
data	文字列	レスポンス内の追加情報 (JSON 形式) です。

：
ゲートウェイは最大 1,500 の同時オンラインサブデバイスを収容できます。 最大数に達すると、ゲートウェイは接続リクエストをすべて拒否します。

エラーメッセージ

エラーコード	エラーメッセージ	説明
460	request parameter error	リクエストパラメーターが正しくありません。
429	rate limit, too many subDeviceOnline msg in one minute	デバイスから IoT Platform への認証リクエストが頻繁過ぎるため、デバイスからの認証リクエストにはスロットル制限がかかります。
428	too many subdevices under gateway	同時に多数のサブデバイスがゲートウェイに接続しています。
6401	topo relation not exist	ゲートウェイとサブデバイスとの間にトポロジー関係が存在しません。
6100	device not found	デバイスが存在しません。
521	device deleted	デバイスが削除されました。
522	device forbidden	サブデバイスが無効にされました。
6287	invalid sign	サブデバイスのパスワードまたは署名が正しくありません。

IoT Platform からサブデバイスを切断します。

アップストリーム

- トピック：/ext/session/{productKey}/{deviceName}/combine/logout
- 返信トピック：/ext/session/{productKey}/{deviceName}/combine/logout_reply

リクエストメッセージ

```
{
  "id": 123,
  "params": {
    "productKey": "xxxxx",
    "deviceName": "xxxxx"
  }
}
```

レスポンスメッセージ

```
{
```

```
  "id": "123",
  "code": 200,
  "message": "success",
  "data": ""
}
```

パラメーターの説明

パラメーター	データタイプ	説明
id	文字列	メッセージ ID です。 今後使用するものとしてパラメーターを予約します。
params	オブジェクト	パラメーターをリクエストします。
deviceName	文字列	サブデバイスの名前
productKey	文字列	サブデバイスが属するプロダクトの ID
code	整数	結果コードです。 値 200 は、呼び出しが成功したことを示します。
message	文字列	結果コード
data	文字列	レスポンス内の追加情報 (JSON 形式)

エラーメッセージ

エラーコード	エラーメッセージ	説明
460	request parameter error	リクエストパラメーターが正しくありません。
520	device no session	サブデバイスセッションが存在しません。

サブデバイス接続の詳細は、 #unique_39をご参照ください。 エラーコードの詳細は、#unique_40をご参照ください。

9.5 デバイスのプロパティ、イベント、およびサービス

プロダクトに TSL を定義している場合、このプロダクトのデバイスは定義したプロパティ、イベント、およびサービスに関するデータを個別に報告できます。 このトピックでは、TSL に基づいてデータがどのように報告されるか説明します。 TSL のデータ形式の詳細は、「TSL 形式 (The TSL format)」をご参照ください。

プロダクトを作成するときは、そのプロダクトのデバイスのデータタイプを選択する必要があります。 IoT Platform は、ICA 標準データ形式 (Alink JSON) と解析しない/カスタマイズの 2 つのデータタイプをサポートします。

- ICA 標準データ形式 (Alink JSON)：デバイスは IoT Platform で定義されている標準形式でデータを生成し、そのデータをIoT Platform に報告します。 このデータタイプの選択を推奨します。 このトピックの以下の例では Alink JSON 形式を使用していることにご注目ください。
- 解析しない/カスタマイズ：デバイスはバイナリデータなどの生データをIoT Platform に報告し、その後 IoT Platform はコンソールでサブミットした解析スクリプトを使用して生データを標準データに解析します。 データ解析スクリプトの編集方法の詳細は、 「データ解析 (Data parsing)」をご参照ください。

デバイスレポートのプロパティ

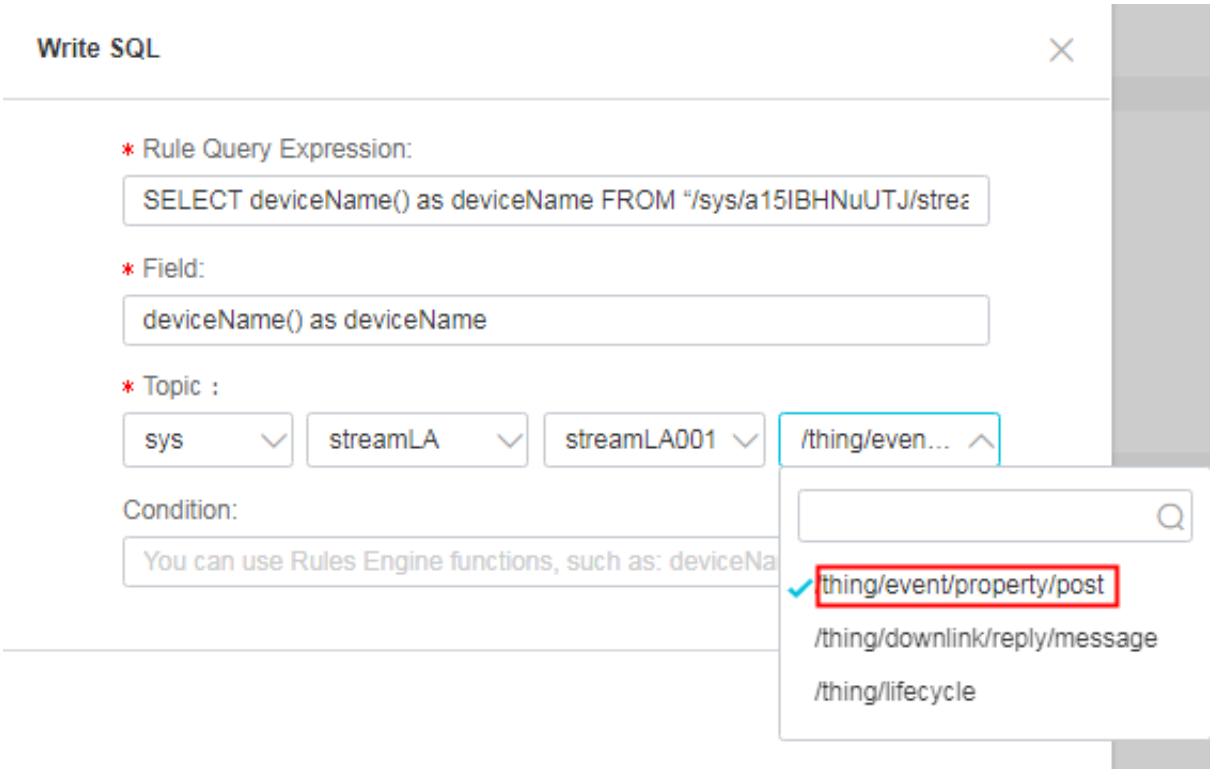
レポートデータ (解析しない/カスタマイズ)

- リクエストトピック：/sys/{productKey}/{deviceName}/thing/model/up_raw
- レスポンストピック：/sys/{productKey}/{deviceName}/thing/model/up_raw_reply

レポートデータ (Alink JSON)

- リクエストトピック：/sys/{productKey}/{deviceName}/thing/event/property/post
- レスポンストピック：/sys/{productKey}/{deviceName}/thing/event/property/post_reply

ルールエンジンを設定して受信したプロパティデータを別の Alibaba Cloud プロダクトインスタンスに転送できます。 ルールアクションの設定例を以下の図に示します。



リクエストメッセージ

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "Power": {
      "value": "on",
      "time": 1524448722000
    },
    "WF": {
      "value": 23.6,
      "time": 1524448722000
    }
  },
  "method": "thing.event.property.post"
}
```

レスポンスデータ

- プロダクトのデータタイプが [解析しない/カスタマイズ] の場合、レスポンスデータと類似したデータとなり、デバイスに送信される前にデータ解析スクリプトを使用して IoT Platform によって解析されます。 データ処理手順の詳細は、「デバイスレポートのプロパティとイベント」をご参照ください。

  ```
  {
  "id":"123",
  "code":200,
  "method":"thing.event.property.post"
  "data":{}
  }
  ```

- Alink レスポンスメッセージ

  ```
  {
    "id": "123",
    "code": 200,
    "data": {}
  }
  ```

表 9-1：リクエストパラメーター

パラメーター	データタイプ	説明
id	文字列	メッセージ ID
version	文字列	プロトコルのバージョンです。 現在、値は 1.0 です。

パラメーター	データタイプ	説明
params	オブジェクト	リクエストパラメーターです。 上記のリクエスト例では、デバイスは Power とWF の 2 つのプロパティを報告します。 プロパティ情報には、時間 (プロパティが報告された時間) と値 (プロパティの値) が含まれます。
time	Long	プロパティが報告された時刻
value	オブジェクト	プロパティの値
method	文字列	リクエストメソッド

表 9-2：レスポンスパラメーター

パラメーター	データタイプ	説明
id	文字列	メッセージ ID
code	整数	結果コードです。「デバイスの共通コード」をご参照ください。
data	文字列	リクエストが成功したときに返されるデータ

エラーメッセージ

エラーコード	エラーメッセージ	説明
460	request parameter error	リクエストパラメーターが正しくありません。
6106	map size must less than 200	報告されたプロパティの数が上限を超えています。 一回で報告できるプロパティの最大数は 200 です。

エラーコード	エラーメッセージ	説明
6313	tsl service not available	TSL 検証サービスは利用できません。 IoT Platform は、プロダクトの TSL に従って受信したすべてのプロパティを検証します。 システム例外が発生するとこのエラーが報告されます。 TSL の定義については、「TSL (Thing Specification Language) の説明」をご参照ください。 注： TSL 検証サービスが利用可能であっても報告されたいくつかのプロパティが TSL で定義されたどのプロパティとも一致しない場合、IoT Platform は無効なプロパティを無視します。 報告されたすべてのプロパティが TSL で定義されたどのプロパティとも一致しない場合、IoT Platform はすべてプロパティを無視します。 この場合、レスポンスは依然として検証が成功したことを示しています。

デバイスプロパティを設定

データをデバイスにプッシュ (解析しない/カスタマイズ)

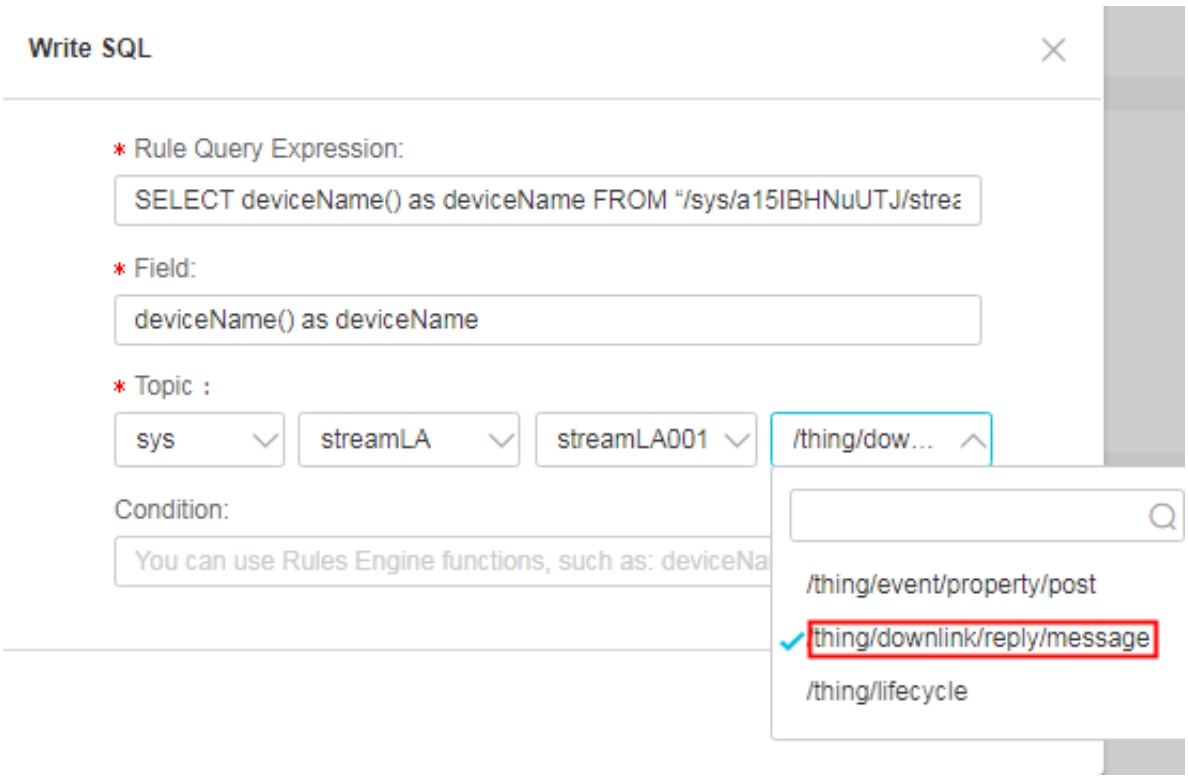
- リクエストトピック：/sys/{productKey}/{deviceName}/thing/model/down_raw
- レスポンストピック：/sys/{productKey}/{deviceName}/thing/model/down_raw_reply

データをデバイスにプッシュ (Alink JSON)

- リクエストトピック：/sys/{productKey}/{deviceName}/thing/service/property/set
- レスポンストピック：/sys/{productKey}/{deviceName}/thing/service/property/set_reply

プロパティ設定の結果は、以下のとおりデータ交換のトピックから取得できます。`/sys/{productKey}/{deviceName}/thing/downlink/reply/message`ルールエンジンを設定して受信し

たプロパティ設定の結果を別の Alibaba Cloud プロダクトインスタンスに転送できます。 ルールアクションの設定例を以下の図に示します。



リクエストメッセージ

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "temperature": "30.5"
  },
  "method": "thing.service.property.set"
}
```

レスポンスメッセージ

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

表 9-3：リクエストパラメーター

パラメーター	データタイプ	説明
id	文字列	メッセージ ID
version	文字列	プロトコルのバージョンです。 現在、値は 1.0 です。

パラメーター	データタイプ	説明
params	オブジェクト	プロパティパラメーターです。 上記のリクエスト例では、設定されるプロパティは以下のとおりです。 `{ "temperature": "30.5" }` .
method	文字列	リクエストメソッド

表 9-4：レスポンスパラメーター

パラメーター	データ	説明
id	文字列	メッセージ ID
code	整数	結果コードです。「デバイスの共通コード」をご参照ください。
data	文字列	リクエストが成功したときに返されるデータ

デバイスレポートイベント

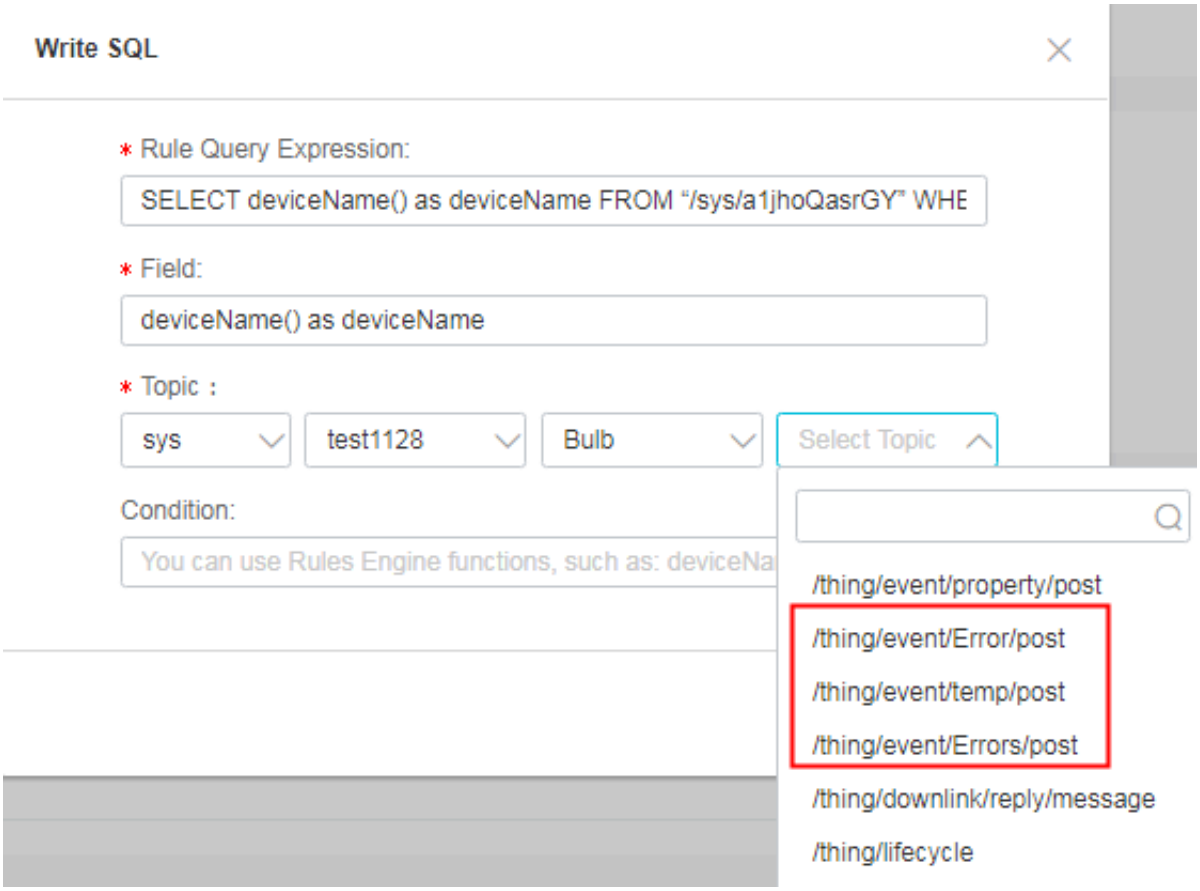
レポートデータ (解析しない/カスタマイズ)

- リクエストトピック：/sys/{productKey}/{deviceName}/thing/model/up_raw
- レスポンストピック：/sys/{productKey}/{deviceName}/thing/model/up_raw_reply

レポートデータ (Alink JSON)

- リクエストトピック：/sys/{productKey}/{deviceName}/thing/event/{tsl.event.identifier}/post
- レスポンストピック：/sys/{productKey}/{deviceName}/thing/event/{tsl.event.identifier}/post_reply

ルールエンジンを設定して受信したイベントデータを別の Alibaba Cloud プロダクトインスタンスに転送できます。 ルールアクションの設定例を以下の図に示します。



Request message

```
{
 "id": "123",
 "version": "1.0",
 "params": {
  "value": {
   "Power": "on",
   "WF": "2"
  },
  "time": 1524448722000
 },
 "method": "thing.event.{tsl.event.identifier}.post"
}
```

レスポンスメッセージ

- プロダクトのデータ型が [解析しない/カスタマイズ] の場合、レスポンスデータは以下の出力と類似したデータとなり、デバイスに送信される前にデータ解析スクリプトを使用して IoT Platform によって解析されます。 データ処理手順の詳細は、「デバイスレポートのプロパティとイベント (Devices report properties and events)」をご参照ください。

  ```
  {
  "id":"123",
  "code":200,
  "method":"thing.event.{tsl.event.identifier}.post"
  "data":{}
  ```

```
}
```

- Alink レスポンスメッセージ

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

表 9-5：リクエストパラメーター

パラメーター	データタイプ	説明
id	文字列	メッセージ ID
version	文字列	プロトコルのバージョンです。 現在、値は 1.0 です。
params	オブジェクト	報告されたイベントのパラメーター
value	オブジェクト	イベント情報です。 上記のリクエスト例では、イベントは以下のとおりです。 `{ "Power": "on", "WF": "2" }`
time	Long	イベントが発生したときの UTC タイムスタンプ
method	文字列	リクエストメソッド

表 9-6：レスポンスパラメーター

パラメーター	データタイプ	説明
id	文字列	メッセージ ID
code	整数	結果コードです。 「デバイスの共通コード」をご参照ください。
data	文字列	リクエストが成功したときに返されるデータ

例

たとえば、イベントアラームはプロダクトのTSLで定義されています。

```
{
  "schema": "https://iot-tsl.oss-cn-shanghai.aliyuncs.com/schema.json",
  "link": "/sys/${productKey}/airCondition/thing/",
  "profile": {
    "productKey": "al123456789",
    "deviceName": "airCondition"
  },
  "events": [
    {
      "identifier": "alarm",
      "name": "alarm",
      "desc": "Fan alarm",
      "type": "alert",
      "required": true,
      "outputData": [
        {
          "identifier": "errorCode",
          "name": "ErrorCode",
          "dataType": {
            "type": "text",
            "specs": {
              "length": "255"
            }
          }
        }
      ],
      "method": "thing.event.alarm.post"
    }
  ]
}
```

リクエストメッセージ

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "value": {
      "errorCode": "error"
    },
    "time": 1524448722000
  },
  "method": "thing.event.alarm.post"
}
```

注：

- tsl.event.identifier は TSL 内のイベント ID を示します。 TSL テンプレートの詳細は、 TSL (Thing Specification Language) の説明をご参照ください。
- IoT Platform は、デバイスが報告したすべてのイベントをプロダクトの TSL に従って検証します。 報告されたイベントが TSL で定義されているどのイベントとも一致しない場合は、エラーコードが返されます。

デバイスサービスの呼び出し

- データをデバイスにプッシュ (解析しない/カスタマイズ)
  - リクエストトピック：/sys/{productKey}/{deviceName}/thing/model/down_raw
  - レスポンストピック：/sys/{productKey}/{deviceName}/thing/model/down_raw_reply
- データをデバイスにプッシュ (Alink JSON)
  - リクエストトピック：/sys/{productKey}/{deviceName}/thing/service/{tsl.service.identifier}
  - レスポンストピック：/sys/{productKey}/{deviceName}/thing/service/{tsl.service.identifier}_reply

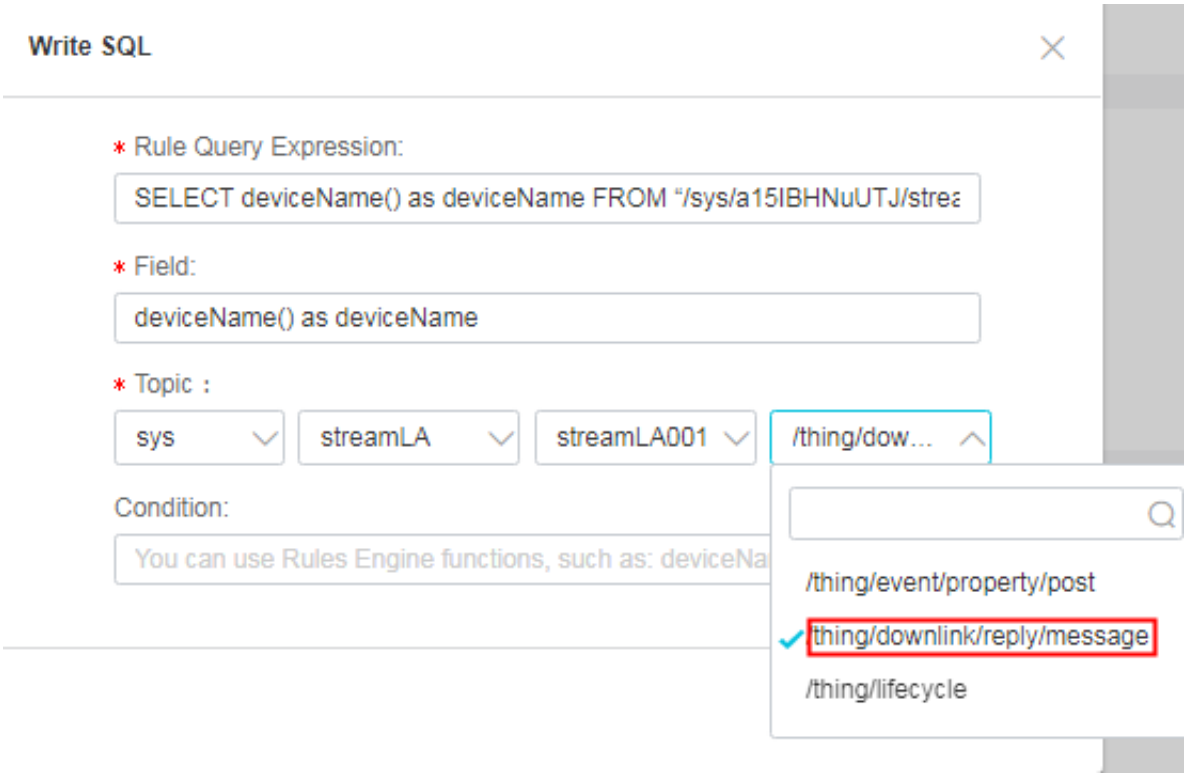
サービスを定義するときは、サービスの方式を選択する必要があります。 サービスの方式は、同期方式または非同期方式のどちらでもかまいません。

- 同期方式：IoT Platform は RRPC 方式を使用してリクエストをデバイスにプッシュします。RRPC 方式の詳細は、「RRPC について (What is RRPC)」をご参照ください。

- 非同期方式：IoT Platform は非同期的にリクエストをデバイスにプッシュし、デバイスは非同期的にオペレーション結果を返します。

  IoT Platform がレスポンストピックをサブスクライブするのはサービスに対して非同期方式が選択されている場合だけです。 オペレーション結果は、以下のとおりデータ交換のトピックから取得できます。`/sys/{productKey}/{deviceName}/thing/downlink/reply/message`.

  ルールエンジンを設定してデバイスから返されるサービス呼び出しの結果を別の Alibaba Cloud プロダクトインスタンスに転送できます。 ルールアクションの設定例を以下の図に示します。



リクエストメッセージ

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "Power": "on",
    "WF": "2"
  },
  "method": "thing.service.{tsl.service.identifier}"
}
```

レスポンスメッセージ

```
{
  "id": "123",
  "code": 200,
  "data": {}
```

```
}
```

表 9-7：リクエストパラメーター

パラメーター	データタイプ	説明
id	文字列	メッセージ ID
version	文字列	プロトコルのバージョンです。 現在、値は 1.0 です。
params	オブジェクト	サービスの呼び出しに使用されるパラメーターで、サービスの識別子と値が含まれます。 例： `{ "Power": "on", "WF": "2" }`
method	文字列	リクエスト方式

表 9-8：レスポンスパラメーター

パラメーター	データタイプ	説明
id	文字列	メッセージ ID
code	整数	結果コードです。 「デバイスの共通コード」をご参照ください。
data	文字列	リクエストが成功したときに返されるデータです。 データの値はプロダクトの TSL によって決まります。 デバイスがサービスに関する情報を返さない場合、data の値は空です。 デバイスがサービス情報を返す場合、返されるデータ値は TSL 内のサービスの定義に厳密に準拠します。

例

たとえば、サービス SetWeight の プロダクトの TSL 内の定義は以下のとおりです。

```
{
  "schema": "https://iotx-tsl.oss-ap-southeast-1.aliyuncs.com/schema.json",
  "profile": {
    "productKey": "testProduct01"
  },
  "services": [
    {
      "outputData": [
        {
          "identifier": "OldWeight",
          "dataType": {
            "specs": {
              "unit": "kg",
              "min": "0",
              "max": "200",
              "step": "1"
            },
            "type": "double"
          },
          "name": "OldWeight"
        },
        {
          "identifier": "CollectTime",
          "dataType": {
            "specs": {
              "length": "2048"
            },
            "type": "text"
          },
          "name": "CollectTime"
        }
      ],
      "identifier": "SetWeight",
      "inputData": [
        {
          "identifier": "NewWeight",
          "dataType": {
            "specs": {
              "unit": "kg",
              "min": "0",
              "Max": 200,
              "step": "1"
            },
            &amp;quot;type&amp;quot; :  &amp;quot;double&amp;quot;
          },
          "name": "NewWeight"
        }
      ],
      "method": "thing.service.SetWeight",
      "name": "SetWeight",
      "required": false,
      "callType": "async"
    }
  ]
}
```

サービス呼び出しのためのリクエストメッセージ:

```
{
  "method": "thing.service.SetWeight",
```

```
  "id": "105917531",
  "param": {
    "NewWeight": 100.8
  },
  &amp;quot;version&amp;quot;: &amp;quot;1.0.0&amp;quot;、
}
```

レスポンスメッセージ:

```
{
  "id": "105917531",
  "code": "200",
  "data": {
    "CollectTime": "1536228947682",
    "OldWeight": 100.101
  }
}
```

注:

tsl.service.identifier は、TSL でのサービスの識別子を示しています。TSL の定義方法について詳しくは、「TSL (Thing Specification Language) の説明」をご参照ください。

ゲートウェイデバイスレポートデータ

ゲートウェイデバイスは、そのサブデバイスも含めてプロパティとイベントを IoT Platform にレポートすることができます。

注:

- 200 までプロパティをレポートできます。
- ゲートウェイは一度に 20 までイベントをレポートできます。
- ゲートウェイは 20 までのサブデバイスのプロパティとイベントをレポートできます。
- IoT Platform は、デバイス、位相関係、TSL でのプロパティとイベントの定義を検証します。いずれか 1 つの検証が失敗すると、データも失敗とレポートします。

レポートデータ (解析しない/カスタマイズ)

- リクエストトピック: Topic：/sys/{productKey}/{deviceName}/thing/model/up_raw
- レスポンストピック: /sys/{gw_productKey}/{gw_deviceName}/thing/topo/add_reply

レスポンスデータ (Alink JSON)

- リクエストトピック: /sys/{productKey}/{deviceName}/thing/event/property/pack/post
- レスポンストピック: /sys/{productKey}/{deviceName}/thing/event/property/pack/post_reply

リクエストメッセージ

```
{
 "id": "123",
 &amp;quot;version&amp;quot;： &amp;quot;1.0&amp;quot;、
 "param": {
  "properties": {
   "Power": {
    "value": "on",
    "time": 1524448722000
   },
   "WF": {
    "value" : {
    "time": 1524448722000
   }
  },
  "events": {
   "alarmEvent1": {
    "value" : {
     "param1": "on",
     "param2": "2"
    },
    "time": 1524448722000
   },
   "alertEvent2": {
    "value" : {
     "param1": "on",
     "param2": "2"
    },
    "time": 1524448722000
   }
  },
  "subDevices": [
   {
    "identity": {
     "productKey": "",
     "deviceName": ""
    },
    "properties": {
     "Power": {
      "value": "on",
      "time": 1524448722000
     },
     "WF": {
      "value": {},
      "time": 1524448722000
     }
    },
    "events": {
     "alarmEvent1": {
      "value": {
       "param1": "on",
       "param2": "2"
      },
      "time": 1524448722000
     },
     "alertEvent2": {
      "value": {
       "param1": "on",
       "param2": "2"
      },
      "time": 1524448722000
     }
```

```
      }
    }
   ]
  },
  "method": "thing.event.property.pack.post"
}
```

レスポンスメッセージ

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

リクエストパラメーター

パラメーター	データタイプ	説明
id	文字列	メッセージ ID
version	文字列	プロトコルのバージョンです。 現在、値は 1.0 です。
params	オブジェクト	リクエストパラメーターです。
properties	オブジェクト	プロパティ名、値、プロパティが生成された時刻など、プロパティに関する情報
events	オブジェクト	プロパティ名、値、イベントが生成された時刻など、イベントに関する情報
subDevices	オブジェクト	サブデバイス情報
productKey	文字列	サブデバイスの ProductKey
deviceName	文字列	サブデバイスの名前
method	文字列	リクエスト方式

レスポンスパラメーター

パラメーター	データタイプ	説明
id	文字列	メッセージ ID
code	整数	結果コードです。 200 はリクエストが成功したことを示します。
data	オブジェクト	リクエストが成功したときに返されるデータ

9.6 ゲートウェイデバイスに設定データを送信

クラウド上で設定した TSL モデルおよびサブデバイス接続チャネル設定の拡張設定情報をゲートウェイデバイスに送信します。

- トピック：/sys/{productKey}/{deviceName}/thing/model/config/push

リクエストメッセージ

```
{
  "id": 123,
  "version": "1.0",
  "method": "thing.model.config.push",
  "data": {
    "digest":"",
    "digestMethod":"",
    "url": ""
  }
}
```

パラメーターの説明

パラメーター	データタイプ	説明
id	文字列	メッセージ ID
version	文字列	プロトコルのバージョン番号 デフォルト値 : 1.0
method	文字列	メソッドは thing.model.config.push です。
data	オブジェクト	データ
digest	文字列	url から取得したデータの整合性を検証するために使用される署名
digestMethod	文字列	署名メソッドです。 デフォルトのメソッドは sha256 です。
url	文字列	OSS から取得するデータの URL

レスポンスメッセージ

```
{
  "id":123,
  "code":200,
  "message":"success",
  "data":{
    "digest":"",
    "digestMethod":"",
    "url":""
```

```
  }
}
```

url データ

```
{
  "modelList": [
    {
      "profile": {
        "productKey": "test01"
      },
      "services": [
        {
          "outputData": "",
          "identifier": "AngleSelfAdaption",
          "inputData": [
            {
              "identifier": "test01",
              "index": 0
            }
          ],
          "displayName": "test01"
        }
      ],
      "properties": [
        {
          "identifier": "identifier",
          "displayName": "test02"
        },
        {
          "identifier": "identifier_01",
          "displayName": "identifier_01"
        }
      ],
      "events": [
        {
          "outputData": [
            {
              "identifier": "test01",
              "index": 0
            }
          ],
          "identifier": "event1",
          "displayName": "abc"
        }
      ]
    },
    {
      "profile": {
        "productKey": "test02"
      },
      "properties": [
        {
          "originalDataType": {
            "specs": {
              "registerCount": 1,
              "reverseRegister": 0,
              "swap16": 0
            },
            "type": "bool"
          },
          "identifier": "test01",
          "registerAddress": "0x03",
```

```
        "scaling": 1,
        "operateType": "inputStatus",
        "pollingTime": 1000,
        "trigger": 1
       },
       {
        "originalDataType": {
         "specs": {
          "registerCount": 1,
          "reverseRegister": 0,
          "swap16": 0
         },
         "type": "bool"
        },
        "identifier": "test02",
        "registerAddress": "0x05",
        "scaling": 1,
        "operateType": "coilStatus",
        "pollingTime": 1000,
        "trigger": 2
       }
      ]
     }
    ],
    "serverList": [
     {
      "baudRate": 1200,
      "protocol": "RTU",
      "byteSize": 8,
      "stopBits": 2,
      "parity": 1,
      "name": "modbus01",
      "serialPort": "0",
      "serverId": "D73251B4277742"
     },
     {
      "protocol": "TCP",
      "port": 8000,
      "ip": "192.168.0.1",
      "name": "modbus02",
      "serverId": "586CB066D6A34"
     },
     {
      "password": "XIJTginONohPEUAyZxLB7Q==",
      "secPolicy": "Basic128Rsa15",
      "name": "server_01",
      "secMode": "Sign",
      "userName": "123",
      "serverId": "55A9D276A7ED470",
      "url": "tcp:00",
      "timeout": 10
     },
     {
      "password": "hAaX5s13gwX2JwyvUkOAfQ==",
      "name": "service_09",
      "secMode": "None",
      "userName": "1234",
      "serverId": "44895C63E3FF401",
      "url": "tcp:00",
      "timeout": 10
     }
    ],
    "deviceList": [
     {
```

```
    "deviceConfig": {
     "displayNamePath": "123",
     "serverId": "44895C63E3FF4013924CEF31519ABE7B"
    },
    "productKey": "test01",
    "deviceName": "test_02"
   },
   {
    "deviceConfig": {
     "displayNamePath": "1",
     "serverId": "55A9D276A7ED47"
    },
    "productKey": "test01",
    "deviceName": "test_03"
   },
   {
    "deviceConfig": {
     "slaveId": 1,
     "serverId": "D73251B4277742D"
    },
    "productKey": "test02",
    "deviceName": "test01"
   },
   {
    "deviceConfig": {
     "slaveId": 2,
     "serverId": "586CB066D6A34E"
    },
    "productKey": "test02",
    "deviceName": "test02"
   }
  ],
  "tslList": [
   {
    "schema": "https://iotx-tsl.oss-ap-southeast-1.aliyuncs.com/schema.json",
    "profile": {
     "productKey": "test02"
    },
    "services": [
     {
      "outputData": [],
      "identifier": "set",
      "inputData": [
       {
        "identifier": "test02",
        "dataType": {
         "specs": {
          "unit": "mm",
          "min": "0",
          "max": "1"
         },
         "type": "int"
        },
        "name": "FeatureTest02"
       }
      ],
      "method": "thing.service.property.set",
      "name": "set",
      "required": true,
      "callType": "async",
      "desc": "Set properties"
     },
     {
      "outputData": [
```

```
        {
          "identifier": "test01",
          "dataType": {
            "specs": {
              "unit": "m",
              "min": "0",
              "max": "1"
            },
            "type": "int"
          },
          "name": "FeatureTest01"
        },
        {
          "identifier": "test02",
          "dataType": {
            "specs": {
              "unit": "mm",
              "min": "0",
              "max": "1"
            },
            "type": "int"
          },
          "name": "FeatureTest02"
        }
      ],
      "identifier": "get",
      "inputData": [
        "test01",
        "test02"
      ],
      "method": "thing.service.property.get",
      "name": "get",
      "required": true,
      "callType": "async",
      "desc": "Get properties"
    }
  ],
  "properties": [
    {
      "identifier": "test01",
      "dataType": {
        "specs": {
          "unit": "m",
          "min": "0",
          "max": "1"
        },
        "type": "int"
      },
      "name": "FeatureTest01",
      "accessMode": "r",
      "required": false
    },
    {
      "identifier": "test02",
      "dataType": {
        "specs": {
          "unit": "mm",
          "min": "0",
          "max": "1"
        },
        "type": "int"
      },
      "name": "FeatureTest02",
      "accessMode": "rw",
```

```
      "required": false
     }
    ],
    "events": [
     {
      "outputData": [
       {
        "identifier": "test01",
        "dataType": {
         "specs": {
          "unit": "m",
          "min": "0",
          "max": "1"
         },
         "type": "int"
        },
        "name": "FeatureTest01"
       },
       {
        "identifier": "test02",
        "dataType": {
         "specs": {
          "unit": "mm",
          "min": "0",
          "max": "1"
         },
         "type": "int"
        },
        "name": "FeatureTest02"
       }
      ],
      "identifier": "post",
      "method": "thing.event.property.post",
      "name": "post",
      "type": "info",
      "required": true,
      "desc": "Report properties"
     }
    ]
   },
   {
    "schema": "https://iotx-tsl.oss-ap-southeast-1.aliyuncs.com/schema.json",
    "profile": {
     "productKey": "test01"
    },
    "services": [
     {
      "outputData": [],
      "identifier": "set",
      "inputData": [
       {
        "identifier": "identifier",
        "dataType": {
         "specs": {
          "length": "2048"
         },
         "type": "text"
        },
        "name": "7614"
       },
       {
        "identifier": "identifier_01",
        "dataType": {
         "specs": {
```

```
            "length": "2048"
           },
           "type": "text"
          },
          "name": "FeatureTest01"
         }
        ],
        "method": "thing.service.property.set",
        "name": "set",
        "required": true,
        "callType": "async",
        "desc": "Set properties",
       },
       {
        "outputData": [
         {
          "identifier": "identifier",
          "dataType": {
           "specs": {
            "length": "2048"
           },
           "type": "text"
          },
          "name": "7614"
         },
         {
          "identifier": "identifier_01",
          "dataType": {
           "specs": {
            "length": "2048"
           },
           "type": "text"
          },
          "name": "FeatureTest01"
         }
        ],
        "identifier": "get",
        "inputData": [
         "identifier",
         "identifier_01"
        ],
        "method": "thing.service.property.get",
        "name": "get",
        "required": true,
        "callType": "async",
        "desc": "Get properties",
       },
       {
        "outputData": [],
        "identifier": "AngleSelfAdaption",
        "inputData": [
         {
          "identifier": "test01",
          "dataType": {
           "specs": {
            "min": "1",
            "max": "10",
            "step": "1"
           },
           "type": "int"
          },
          "name": "Parameter1",
         }
        ],
```

```
      "method": "thing.service.AngleSelfAdaption",
      "name": "adaptive angle calibration",
      "required": false,
      "callType": "async"
     }
    ],
    "properties": [
     {
      "identifier": "identifier",
      "dataType": {
       "specs": {
        "length": "2048"
       },
       "type": "text"
      },
      "name": "7614",
      "accessMode": "rw",
      "required": true
     },
     {
      "identifier": "identifier_01",
      "dataType": {
       "specs": {
        "length": "2048"
       },
       "type": "text"
      },
      "name": "FeatureTest01",
      "accessMode": "rw",
      "required": false
     }
    ],
    "events": [
     {
      "outputData": [
       {
        "identifier": "identifier",
        "dataType": {
         "specs": {
          "length": "2048"
         },
         "type": "text"
        },
        "name": "7614"
       },
       {
        "identifier": "identifier_01",
        "dataType": {
         "specs": {
          "length": "2048"
         },
         "type": "text"
        },
        "name": "FeatureTest01"
       }
      ],
      "identifier": "post",
      "method": "thing.event.property.post",
      "name": "post",
      "type": "info",
      "required": true,
      "desc": "Report properties."
     },
     {
```

```
        "outputData": [
          {
            "identifier": "test01",
            "dataType": {
              "specs": {
                "min": "1",
                "max": "20",
                "step": "1"
              },
              "type": "int"
            },
            "name": "ParameterTest1"
          }
        ],
        "identifier": "event1",
        "method": "thing.event.event1.post",
        "name": "event1",
        "type": "info",
        "required": false
      }
    ]
  }
 ]
}
```

パラメーターの説明

パラメーター	データタイプ	説明
modelList	オブジェクト	ゲートウェイにマウントされているすべてのサブデバイスの拡張プロダクト情報
serverList	オブジェクト	ゲートウェイのサブデバイスチャネル
deviceList	オブジェクト	ゲートウェイにマウントされているすべてのサブデバイスの接続設定
tslList	オブジェクト	ゲートウェイにマウントされているすべてのサブデバイスの TSL

modelList description

現在、通信プロトコル Modbus と OPC UA がサポートされていますが、2 つのプロトコルの拡張情報は異なります。

- Modbus

```
{
 "profile": {
  "productKey": "test02"
 },
 "properties": [
  {
   "originalDataType": {
    "specs": {
     "registerCount": 1,
     "reverseRegister": 0,
     "swap16": 0
```

```
      },
      "type": "bool"
    },
    "identifier": "test01",
    "registerAddress": "0x03",
    "scaling": 1,
    "operateType": "inputStatus",
    "pollingTime": 1000,
    "trigger": 1
   },
   {
    "originalDataType": {
     "specs": {
      "registerCount": 1,
      "reverseRegister": 0,
      "swap16": 0
     },
     "type": "bool"
    },
    "identifier": "test02",
    "registerAddress": "0x05",
    "scaling": 1,
    "operateType": "coilStatus",
    "pollingTime": 1000,
    "trigger": 2
   }
  ]
}
```

パラメーターの説明

パラメーター	データタイプ	説明
identifier	文字列	プロパティ、イベント、またはサービスの識別子
operateType	文字列	操作タイプです。 サポートされている値は以下のとおりです。 - coilStatus - inputStatus - holdingRegister - inputRegister
registerAddress	文字列	レジスタアドレス
originalDataType	オブジェクト	元のデータ型
type	文字列	サポートされている値は以下のとおりです。 int16、uint16、int32、uint32、int64、uint64、float、double、string、およびカスタマイズされたデータ
specs	オブジェクト	説明

パラメーター	データタイプ	説明
registerCount	整数	レジスタ内のデータ数
swap16	整数	レジスタ内の 16 ビットデータの最初の 8 ビットと最後の 8 ビットを入れ替えます。 0 : false; 1: true.
reverseRegister	整数	元の 32 ビットデータのビットを入れ替えます。 0: false; 1: true.
scaling	整数	ズーム倍率
pollingTime	整数	収集間隔
trigger	整数	データ報告メソッドです。 1: 特定の時期に報告 2: 変更検出時に報告

- OPC UA

```
{
 "profile": {
  "productKey": "test01"
 },
 "services": [
  {
   "outputData": "",
   "identifier": "AngleSelfAdaption",
   "inputData": [
    {
     "identifier": "test01",
     "index": 0
    }
   ],
   "displayName": "test01"
  }
 ],
 "properties": [
  {
   "identifier": "identifier",
   "displayName": "test02"
  },
  {
   "identifier": "identifier_01",
   "displayName": "identifier_01"
  }
 ],
 "events": [
  {
   "outputData": [
    {
     "identifier": "test01",
     "index": 0
    }
   ],
   "identifier": "event1",
   "displayName": "abc"
  }
 ]
```

```
}
```

パラメーターの説明

パラメーター	データタイプ	説明
サービス	オブジェクト	サービス
properties	オブジェクト	プロパティ
イベント	オブジェクト	イベント
outputData	オブジェクト	イベント報告データおよびサービス呼び出しに対して戻される結果など出力パラメーター
identifier	文字列	識別子
inputData	オブジェクト	入力パラメーター
index	整数	インデックス情報
displayName	文字列	表示されている名前

serverList description

2 つのプロトコル (Modbus と OPC UA) がチャネルに対してサポートされています。

- Modbus protocol

```
[
 {
  "baudRate": 1200,
  "protocol": "RTU",
  "byteSize": 8,
  "stopBits": 2,
  "parity": 1,
  "name": "modbus01",
  "serialPort": "0",
  "serverId": "D73251B4277742"
 },
 {
  "protocol": "TCP",
  "port": 8000,
  "ip": "192.168.0.1",
  "name": "modbus02",
  "serverId": "586CB066D6A34"
 }
]
```

パラメーター	データタイプ	説明
protocol	文字列	プロトコルの種類 TCP でも RTU でもかまいません。
port	整数	ポート番号
ip	文字列	IP アドレス

パラメーター	データタイプ	説明
name	文字列	チャネル名
serverId	文字列	チャネル ID
baudRate	整数	ボーレート
byteSize	整数	バイト数
stopBits	整数	ストップビット
parity	整数	パリティビットです。 サポートされている値は以下のとおりです。 - E: 偶数パリティチェック - O: 奇数パリティチェック - N: パリティチェックなし
serialPort	文字列	サーバーのポート番号

- OPC UA プロトコル

```
{
  "password": "XIJTginONohPEUAyZxLB7Q==",
  "secPolicy": "Basic128Rsa15",
  "name": "server_01",
  "secMode": "Sign",
  "userName": "123",
  "serverId": "55A9D276A7ED470",
  "url": "tcp:00",
  "timeout": 10
}
```

パラメーターの説明

パラメーター	データタイプ	説明
password	文字列	AES 暗号化アルゴリズムによって暗号化されたパスワードです。 OPC UA のパスワード暗号化の詳細は、この表の最後にある情報をご参照ください。
secPolicy	文字列	暗号化ポリシーです。 サポートされているオプションには、None、Basic128Rsa15、および Basic256 があります。
secMode	文字列	暗号化モード サポートされているオプションには、None、Sign、およびSignAndEncrypt があります。
name	文字列	サーバー名
userName	文字列	ユーザー名

パラメーター	データタイプ	説明
serverId	文字列	サーバー ID
url	文字列	サーバー接続アドレス
timeout	整数	タイムアウトの値

OPC UA のパスワード暗号化方式

AES 暗号化アルゴリズムと 128 ビット (16 バイト) のグループ化を使用します。 デフォルトのモードは CBC、デフォルトのパディングは PKCS5Padding です。 秘密としてデバイスの deviceSecret を使用します。 暗号化された結果は Base64 でエンコードされています。

コード例 :

```
private static String instance = "AES/CBC/PKCS5Padding";

    private static String algorithm = "AES";

    private static String charsetName = "utf-8";
    /**
     * Encryption algorithm
     *
     * @param data (Data to be encrypted)
     * @param deviceSecret (The deviceSecret of the device)
     * @return
     */
    public static String aesEncrypt(String data, String deviceSecret) {
        try {
            Cipher cipher = Cipher.getInstance(instance);
            byte[] raw = deviceSecret.getBytes();
            SecretKeySpec key = new SecretKeySpec(raw, algorithm);
            IvParameterSpec ivParameter = new IvParameterSpec(deviceSecret.substring(0,
16).getBytes());
            cipher.init(Cipher.ENCRYPT_MODE, key, ivParameter);
            byte[] encrypted = cipher.doFinal(data.getBytes(charsetName));

            return new BASE64Encoder().encode(encrypted);
        } catch (Exception e) {
            e.printStackTrace();
        }

        return null;
    }

    public static String aesDecrypt(String data, String deviceSecret) {
        try {
            byte[] raw = deviceSecret.getBytes(charsetName);
            byte[] encrypted1 = new BASE64Decoder().decodeBuffer(data);
            SecretKeySpec key = new SecretKeySpec(raw, algorithm);
            Cipher cipher = Cipher.getInstance(instance);
            IvParameterSpec ivParameter = new IvParameterSpec(deviceSecret.substring(0,
16).getBytes());
            cipher.init(Cipher.DECRYPT_MODE, key, ivParameter);
            byte[] originalBytes = cipher.doFinal(encrypted1);
            String originalString = new String(originalBytes, charsetName);
            return originalString;
        } catch (Exception ex) {
```

```
            ex.printStackTrace();
        }

        return null;
    }

    public static void main(String[] args) throws Exception {
        String text = "test123";
        String secret = "testTNmjyWHQzniA8wEkTNmjyWHQtest";
        String data = null;
        data = aesEncrypt(text, secret);
        System.out.println(data);
        System.out.println(aesDecrypt(data, secret));
    }
```

deviceList description

- Modbus protocol

```
{
 "deviceConfig": {
  "slaveId": 1,
  "serverId": "D73251B4277742D"
 },
 "productKey": "test02",
 "deviceName": "test01"
}
```

パラメーターの説明

パラメーター	データタイプ	説明
deviceConfig	オブジェクト	デバイス情報
slaveId	整数	スレーブ局 ID
serverId	文字列	チャネル ID
productKey	文字列	プロダクト ID
deviceName	文字列	デバイスの名前

- OPC UA プロトコル

```
{
 "deviceConfig": {
  "displayNamePath": "123",
  "serverId": "44895C63E3FF4013924CEF31519ABE7B"
 },
 "productKey": "test01",
 "deviceName": "test_02"
}
```

パラメーターの説明

パラメーター	データタイプ	説明
deviceConfig	オブジェクト	デバイス接続設定情報

パラメーター	データタイプ	説明
productKey	文字列	プロダクト ID
deviceName	文字列	デバイスの名前
displayNamePath	文字列	表示されている名前
serverId	文字列	関連付けられているチャネル ID

9.7 デバイスを無効にして削除する

ゲートウェイはサブデバイスを無効にしたり削除したりすることができます。

デバイスを無効にする

ダウンストリーム

- トピック：/sys/{productKey}/{deviceName}/thing/disable
- レスポンストピック：/sys/{productKey}/{deviceName}/thing/disable_reply

このトピックでは、デバイス接続を無効にします。 IoT Platform はこのトピックへのメッセージを非同期的に公開し、デバイスはこのトピックをサブスクライブします。 ゲートウェイはこのトピックをサブスクライブして、対応するサブデバイスを無効にすることができます。

リクエストメッセージ

```
{
  "id": "123",
  "version": "1.0",
  "params": {},
  "method": "thing.disable"
}
```

レスポンスメッセージ

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

パラメーターの説明

パラメーター	データタイプ	説明
id	文字列	メッセージ ID
version	文字列	プロトコルバージョンです。現在、値は 1.0 です。

パラメーター	データタイプ	説明
params	オブジェクト	パラメーターをリクエストします。 空のままにします。
method	文字列	リクエストのメソッド
code	整数	結果情報です。 詳細は、以下の資料をご参照ください。デバイスの共通コード

デバイスを有効にする

ダウンストリーム

- トピック：/sys/{productKey}/{deviceName}/thing/enable
- レスポンストピック：/sys/{productKey}/{deviceName}/thing/enable_reply

このトピックでは、デバイス接続を有効にします。 IoT Platform はこのトピックへのメッセージを非同期的に公開し、デバイスはこのトピックをサブスクライブします。 ゲートウェイはこのトピックを購読して、対応するサブデバイスを有効にすることができます。

リクエストメッセージ

```
{
  "id": "123",
  "version": "1.0",
  "params": {},
  "method": "thing.enable"
}
```

レスポンスメッセージ

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

パラメーターの説明

パラメーター	データタイプ	説明
id	文字列	メッセージ ID
version	文字列	プロトコルバージョンです。現在、値は 1.0 です。
params	オブジェクト	パラメーターを要求します。空のままにします。
method	文字列	リクエストのメソッド

パラメーター	データタイプ	説明
code	整数	結果コードです。 詳細は、共通コードをご参照ください。

デバイスを削除

ダウンストリーム

- トピック：/sys/{productKey}/{deviceName}/thing/delete
- レスポンストピック：/sys/{productKey}/{deviceName}/thing/delete_reply

このトピックでは、デバイス接続を削除します。 IoT Platform はこのトピックへのメッセージを非同期的に公開し、デバイスはこのトピックをサブスクライブします。 ゲートウェイはこのトピックをサブスクライブして、対応するサブデバイスを削除できます。

リクエストメッセージ

```
{
  "id": "123",
  "version": "1.0",
  "params": {},
  "method": "thing.delete"
}
```

レスポンスメッセージ

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

パラメーターの説明

パラメーター	データタイプ	説明
id	文字列	メッセージ ID
version	文字列	プロトコルバージョンです。 現在、値は 1.0 です。
params	オブジェクト	パラメーターをリクエストします。 空のままにします。
method	文字列	リクエストのメソッド
code	文字列	結果コードです。 詳細は、共通コードをご参照ください。

9.8 デバイスタグ

ベンダーモデルやデバイスモデルなどの一部の静的な拡張デバイス情報は、デバイスタグとして保存できます。

レポートタグ

アップストリーム

- トピック：/sys/{productKey}/{deviceName}/thing/deviceinfo/update
- 返信トピック：/sys/{productKey}/{deviceName}/thing/deviceinfo/update_reply

リクエストメッセージ

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "attrKey": "Temperature",
      "attrValue": "36.8"
    }
  ],
  "method": "thing.deviceinfo.update"
}
```

レスポンスメッセージ

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

パラメーターの説明

パラメーター	データタイプ	説明
id	文字列	メッセージ ID です。 今後使用するものとしてパラメーターの値を予約します。
version	文字列	プロトコルバージョンです。現時点では、値は 1.0 だけです。

パラメーター	データタイプ	説明
params	オブジェクト	パラメーターをリクエストします。 このパラメーターには最大 200 個の項目を含めることができます。
method	文字列	リクエストの方法
attrKey	文字列	タグ名です。 • 長さ：最大 100 バイト • 有効な文字は、小文字の a ～ z、大文字の A ～ Z、数字 0 ～ 9、およびアンダースコア (_) です。 • タグ名の先頭文字は、英字またはアンダースコア (_) にする必要があります。
attrValue	文字列	タグ値
code	整数	結果コードです。 値が 200 の場合は、リクエストが成功したことを示します。

エラーコード

エラーコード	エラーメッセージ	説明
460	request parameter error	リクエストパラメーターが正しくありません。
6100	device not found	デバイスが存在しません。

タグを削除

アップストリーム

- トピック：/sys/{productKey}/{deviceName}/thing/deviceinfo/delete
- 返信トピック：/sys/{productKey}/{deviceName}/thing/deviceinfo/delete_reply

リクエストメッセージ

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "attrKey": "Temperature"
    }
  ],
  "method": "thing.deviceinfo.delete"
}
```

レスポンスメッセージ

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

パラメーターの説明

パラメーター	データタイプ	説明
id	文字列	メッセージ ID です。 今後使用するものとしてパラメーターの値を予約します。
version	文字列	プロトコルバージョンです。現時点では、値は 1.0 だけです。
params	オブジェクト	パラメーターをリクエストします。
method	文字列	リクエストの方法
attrKey	文字列	タグ名です。 • 長さ：最大 100 バイト。 • 有効な文字は、小文字の a ～ z、大文字の A ～ Z、数字 0 ～ 9、およびアンダースコア (_) です。 • タグ名の先頭文字は、英字またはアンダースコア (_) にする必要があります。
attrValue	文字列	タグ値

パラメーター	データタイプ	説明
code	整数	結果コードです。 値が 200 の場合は、リクエストが成功したことを示します。

エラーメッセージ

エラーコード	エラーメッセージ	説明
460	request parameter error	リクエストパラメーターが正しくありません。
6100	device not found	デバイスが存在しません。

9.9 TSLモデル

デバイスはこのトピックに対するリクエストを発行して IoT Platform からデバイス TSL モデルを取得します。

- トピック：/sys/{productKey}/{deviceName}/thing/dsltemplate/get
- 返信トピック：/sys/{productKey}/{deviceName}/thing/dsltemplate/get_reply

リクエストの Allink データ形式

```
{
  "id": "123",
  "version": "1.0",
  "params": {},
  "method": "thing.dsltemplate.get"
}
```

レスポンスの Allink データ形式

```
{
  "id": "123",
  "code": 200,
  "data": {
    "schema": "https://iot-tsl.oss-cn-shanghai.aliyuncs.com/schema.json",
    "link": "/sys/1234556554/airCondition/thing/",
    "profile": {
      "productKey": "1234556554",
      "deviceName": "airCondition"
    },
    "properties": [
      {
        "identifier": "fan_array_property",
        "name": "Fan array property",
        "accessMode": "r",
        "required": true,
        "dataType": {
          "type": "array",
          "specs": {
```

```
        "size": "128",
        "item": {
          "type": "int"
        }
      }
    }
  }
 ],
 "events": [
  {
   "identifier": "alarm",
   "name": "alarm",
   "desc": "Fan alert",
   "type": "alert",
   "required": true,
   "outputData": [
    {
     "identifier": "errorCode",
     "name": "Error code",
     "dataType": {
      "type": "text",
      "specs": {
       "length": "255"
      }
     }
    }
   ],
   "method": "thing.event.alarm.post"
  }
 ],
 "services": [
  {
   "identifier": "timeReset",
   "name": "timeReset",
   "desc": "Time calibration",
   "inputData": [
    {
     "identifier": "timeZone",
     "name": "Time zone",
     "dataType": {
      "type": "text",
      "specs": {
       "length": "512"
      }
     }
    }
   ],
   "outputData": [
    {
     "identifier": "curTime",
     "name": "Current time",
     "dataType": {
      "type": "date",
      "specs": {}
     }
    }
   ],
   "method": "thing.service.timeReset"
  },
  {
   "identifier": "set",
   "name": "set",
   "required": true,
   "desc": "Set properties",
```

```
      "method": "thing.service.property.set",
      "inputData": [
       {
        "identifier": "fan_int_property",
        "name": "Integer property of the fan",
        "accessMode": "rw",
        "required": true,
        "dataType": {
         "type": "int",
         "specs": {
          "min": "0",
          "max": "100",
          "unit": "g/ml",
          "unitName": "Millilitter"
         }
        }
       }
      ],
      "outputData": []
     },
     {
      "identifier": "get",
      "name": "get",
      "required": true,
      "desc": "Get properties",
      "method": "thing.service.property.get",
      "inputData": [
       "array_property",
       "fan_int_property",
       "batch_enum_attr_id",
       "fan_float_property",
        "fan_double_property",
       "fan_text_property",
       "Maid ",
       "batch_boolean_attr_id",
       "fan_struct_property"
      ],
      "outputData": [
       {
        "identifier": "fan_array_property",
        "name": "Fan array property",
        "accessMode": "r",
        "required": true,
        "dataType": {
         "type": "array",
         "specs": {
          "size": "128",
          "item": {
           "type": "int"
          }
         }
        }
       }
      ]
     }
    ]
   }
}
```


パラメーターの説明 :

パラメーター	データタイプ	説明
id	文字列	メッセージ ID です。 今後使用するものとしてパラメーター値を予約します。
version	文字列	プロトコルバージョンです。 現在、値は 1.0 です。
params	オブジェクト	このパラメーターは空のままにします。
method	文字列	リクエストのメソッド
productKey	文字列	ProductKey. この例では、ProductKey は 1234556554 です。
deviceName	文字列	デバイス名です。 この例では、デバイス名は airCondition です。
data	オブジェクト	デバイスの TSL モデルです。 詳細 :TSL (Thing Specification Language) の説明

エラーコード

エラーコード	エラーメッセージ	説明
460	request parameter error	リクエストパラメーターが正しくありません。
6321	tsl: device not exist in product	デバイスが存在しません。

9.10 ファームウェアの更新

ファームウェアの更新の詳細は、 「#unique_51」 と「#unique_52」をご参照ください。

ファームウェアのバージョンを報告

アップストリーム

- トピック: /ota/device/inform/{productKey}/{deviceName}

  デバイスは、このトピックに対するメッセージを発行して、現在のファームウェアのバージョンを IoT Platform に報告します。

リクエストメッセージ

```
{
  "id": 1,
  "params": {
    "version": "1.0.1"
  }
}
```

パラメーターの説明

パラメーター	データタイプ	説明
id	文字列	メッセージ ID
version	文字列	ファームウェアのバージョン情報

ファームウェア情報をプッシュ

ダウンストリーム

- トピック: /ota/device/upgrade/{productKey}/{deviceName}

  IoT Platform はこのトピックに対するメッセージを発行し、ファームウェア情報をプッシュします。 デバイスは、このトピックをサブスクライブしてファームウェア情報を取得します。

リクエストメッセージ

```
{
  "code": "1000",
  "data": {
    "size": 432945,
    "version": "2.0.0",
    "url": "https://iotx-ota-pre.oss-cn-shanghai.aliyuncs.com/nopoll_0.4.4.tar.gz?Expires
=1502955804&OSSAccessKeyId=XXXXXXXXXXXXXXXXXXXX&Signature=XfgJu7P6DW
WejstKJgXJEH0qAKU%3D&security-token=CAISuQJ1q6Ft5B2yfSjIpK6MGsyN1Jx5jo6mVnfB
glIPTvlvt5D50Tz2IHtIf3NpAusdsv03nWxT7v4flqFyTINVAEvYZJOPKGrGR0DzDbDasumZsJbo4f
%2FMQBqEaXPS2MvVfJ%2BzLrf0ceusbFbpjzJ6xaCAGxypQ12iN%2B%2Fr6%2F5gdc9FcQ
SkL0B8ZrFsKxBltdUROFbIKP%2BpKWSKuGfLC1dysQcO1wEP4K%2BkkMqH8Uic3h%2Boy
%2BgJt8H2PpHhd9NhXuV2WMzn2%2FdtJOiTknxR7ARasaBqhelc4zqA%2FPPlWgAKv
kXba7aIoo01fV4jN5JXQfAU8KLO8tRjofHWmojNzBJAAPpYSSy3Rvr7m5efQrrybY1lLO6iZy
%2BVio2VSZDxshI5Z3McKARWct06MWV9ABA2TTXXOi40BOxuq%2B3JGoABXC54TOlo7
%2F1wTLTsCUqzzeIiXVOK8CfNOkfTucMGHkeYeCdFkm%2FkADhXAnrnGf5a4FbmK
MQph2cKsr8y8UfWLC6IzvJsClXTnbJBMeuWIqo5zIynS1pm7gf%2F9N3hVc6%2BEeIk0xfl
2tycsUpbL2FoaGk6BAF8hWSWYUXsv59d5Uk%3D",
    "md5": "93230c3bde425a9d7984a594ac55ea1e",
    "sign": "93230c3bde425a9d7984a594********",
    "signMethod": "Md5"
  },
  "id": 1507707025,
  "message": "success"
}
```

パラメーターの説明

パラメーター	データタイプ	説明
id	文字列	メッセージ ID
message	文字列	結果情報
version	文字列	ファームウェアのバージョン情報
size	Long	ファームウェアのサイズ (バイト単位)
url	文字列	ファームウェアの OSS アドレス
sign	文字列	ファームウェアの署名
signMethod	文字列	署名のメソッドです。 現在サポートされているメソッドは MD5 と sha256 です。
md5	文字列	予備のパラメーターです。 このパラメーターは前のデバイス情報と互換を取る場合にに使用されます。 署名メソッドが MD5 の場合、IoT Platform は sign パラメーターと md5 パラメーターの両方に値を割り当てます。

更新の進捗状況を報告

アップストリーム

- トピック: /ota/device/progress/{productKey}/{deviceName}

  デバイスはこのトピックをサブスクライブし、ファームウェア更新の進行状況を報告します。

リクエストメッセージ

```
{
  "id": 1,
  "params": {
    "step": "-1",
    "desc": "Firmware update has failed. No firmware information is available."
  }
}
```

パラメーターの説明

パラメーター	データタイプ	説明
id	文字列	メッセージ ID
step	文字列	ファームウェア更新の進捗情報です。 値の範囲： • 1 ～ 100 の値は進捗率を示します。 • 値 -1 は、ファームウェアの更新が失敗したことを示します。 • 値 -2 は、ファームウェアのダウンロードが失敗したことを示します。 • 値 -3 は、ファームウェアの検証が失敗したことを示します。 • 値 -4 は、ファームウェアのインストールが失敗したことを示します。
desc	文字列	現在のステップの説明です。 例外が発生した場合、このパラメーターがエラーメッセージを表示します。

IoT Platform からファームウェア情報をリクエスト

- トピック：/ota/device/request/{productKey}/{deviceName}

リクエストメッセージ

```
{
  "id": 1,
  "params": {
    "version": "1.0.1"
  }
}
```

パラメーターの説明

パラメーター	データタイプ	説明
id	文字列	メッセージ ID
version	文字列	ファームウェアのバージョン情報

9.11 リモート設定

ここでは、Topic と Alink の JSON 形式のリクエストとリモート設定のレスポンスについて紹介します。 リモート設定の使用方法については、『ユーザーガイド (User Guide)』の「#unique_54」 をご参照ください。

デバイスが IoT Platform に設定情報を要求

アップストリーム

- Topic: /sys/{productKey}/{deviceName}/thing/config/get
- Reply topic: /sys/{productKey}/{deviceName}/thing/config/get_reply

リクエストメッセージ

```
{
  "id": 123,
  "version": "1.0",
  "params": {
    "configScope": "product",
    "getType": "file"
  },
  "method": "thing.config.get"
}
```

レスポンスメッセージ

```
{
  "id": "123",
  "version": "1.0",
  "code": 200,
  "data": {
    "configId": "123dagdah",
    "configSize": 1234565,
    "sign": "123214adfadgadg",
    "signMethod": "Sha256",
    "url": "https://iotx-config.oss-cn-shanghai.aliyuncs.com/nopoll_0.4.4.tar.gz?Expires
=1502955804&OSSAccessKeyId=XXXXXXXXXXXXXXXXXXX&Signature=XfgJu7P6DW
WejstKJgXJEH0qAKU%3D&security-token=CAISuQJ1q6Ft5B2yfSjIpK6MGsyN1Jx5jo6mVnfB
glIPTvlvt5D50Tz2IHtIf3NpAusdsv03nWxT7v4flqFyTINVAEvYZJOPKGrGR0DzDbDasumZsJbo4f
%2FMQBqEaXPS2MvVfJ%2BzLrf0ceusbFbpjzJ6xaCAGxypQ12iN%2B%2Fr6%2F5gdc9FcQ
SkL0B8ZrFsKxBltdUROFbIKP%2BpKWSKuGfLC1dysQcO1wEP4K%2BkkMqH8Uic3h%2Boy
%2BgJt8H2PpHhd9NhXuV2WMzn2%2FdtJOiTknxR7ARasaBqhelc4zqA%2FPPlWgAKv
kXba7aIoo01fV4jN5JXQfAU8KLO8tRjofHWmojNzBJAAPpYSSy3Rvr7m5efQrrybY1lLO6iZy
%2BVio2VSZDxshI5Z3McKARWct06MWV9ABA2TTXXOi40BOxuq%2B3JGoABXC54TOlo7
%2F1wTLTsCUqzzeIiXVOK8CfNOkfTucMGHkeYeCdFkm%2FkADhXAnrnGf5a4FbmK
MQph2cKsr8y8UfWLC6IzvJsClXTnbJBMeuWIqo5zIynS1pm7gf%2F9N3hVc6%2BEeIk0xfl
2tycsUpbL2FoaGk6BAF8hWSWYUXsv59d5Uk%3D",
    "getType": "file"
  }
}
```

パラメーター説明

パラメーター	データタイプ	説明
id	文字列	メッセージ ID
version	文字列	プロトコルバージョンです。 現在の値は 1.0 です。
configScope	文字列	設定スコープです。 現在、IoT Platform がサポートしているのはプロダクトのディメンションの設定だけです。 値 : プロダクト
getType	文字列	設定の目的のファイルタイプです。 現在サポートされているタイプは file です。 値を file に設定します。
configId	文字列	設定の ID
configSize	Long	設定ファイルのサイズ (バイト単位)
sign	文字列	署名値
signMethod	文字列	署名メソッドです。 サポートされている署名メソッドは Sha256 です。
url	文字列	設定ファイルが格納されている OSS アドレス
code	整数	結果コードです。 値 200 は操作が成功したことを示し、その他の状況コードは操作が失敗したことを示します。

エラーコード

エラーコード	エラーメッセージ	説明
6713	thing config function is not available	プロダクトのリモート設定機能が無効になっています。 IoT Platform コンソールの [リモート設定] ページで、プロダクトのリモート設定を有効にします。
6710	no data	設定データが見つかりません。

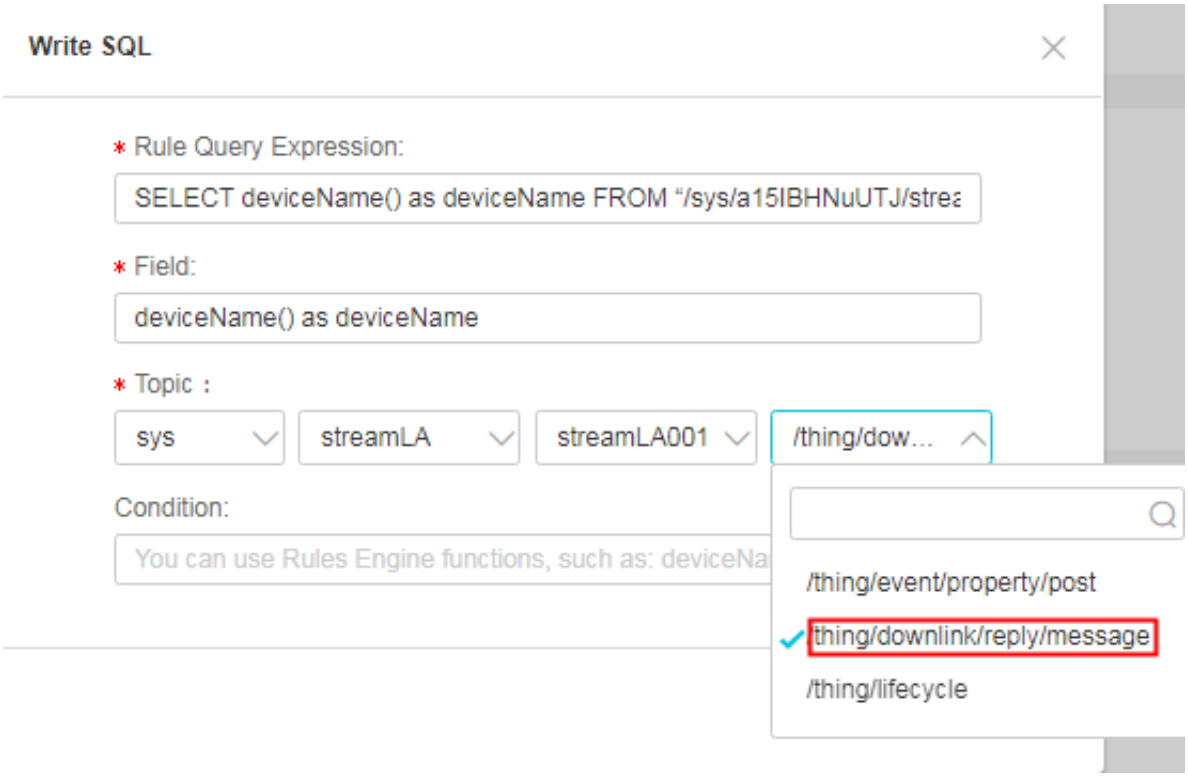
IoT Platform コンソールの設定をデバイスにプッシュする

ダウンストリーム

- Topic: /sys/{productKey}/{deviceName}/thing/config/push
- Reply topic: /sys/{productKey}/{deviceName}/thing/config/push_reply

デバイスは、IoT Platform によってプッシュされる設定のトピックをサブスクライブします。 IoT Platform コンソールで設定ファイルを編集して送信した後、IoT Platform は非同期方式で設定をデバイスにプッシュします。 IoT Platform は、非同期呼び出しの結果のデータ交換トピックをサブスクライブします。 データ交換のトピックは `/{productKey}/{deviceName}/thing/downlink/reply/message` にあります。

[ルールエンジン] を使用してデバイスが返す結果を別の Alibaba Cloud プロダクトに転送できます。 次の図に、ルールアクションの設定例を示します。



リクエストメッセージ：

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "configId": "123dagdah",
    "configSize": 1234565,
    "sign": "123214adfadgadg",
    "signMethod": "Sha256",
    "url": "https://iotx-config.oss-cn-shanghai.aliyuncs.com/nopoll_0.4.4.tar.gz?Expires
=1502955804&OSSAccessKeyId=XXXXXXXXXXXXXXXXXXXX&Signature=XfgJu7P6DW
WejstKJgXJEH0qAKU%3D&security-token=CAISuQJ1q6Ft5B2yfSjIpK6MGsyN1Jx5jo6mVnfB
```

```
glIPTvlvt5D50Tz2IHtIf3NpAusdsv03nWxT7v4flqFyTINVAEvYZJOPKGrGR0DzDbDasumZsJbo4f
%2FMQBqEaXPS2MvVfJ%2BzLrf0ceusbFbpjzJ6xaCAGxypQ12iN%2B%2Fr6%2F5gdc9FcQ
SkL0B8ZrFsKxBltdUROFbIKP%2BpKWSKuGfLC1dysQcO1wEP4K%2BkkMqH8Uic3h%2Boy
%2BgJt8H2PpHhd9NhXuV2WMzn2%2FdtJOiTknxR7ARasaBqhelc4zqA%2FPPlWgAKv
kXba7aIoo01fV4jN5JXQfAU8KLO8tRjofHWmojNzBJAAPpYSSy3Rvr7m5efQrrybY1lLO6iZy
%2BVio2VSZDxshI5Z3McKARWct06MWV9ABA2TTXXOi40BOxuq%2B3JGoABXC54TOlo7
%2F1wTLTsCUqzzeIiXVOK8CfNOkfTucMGHkeYeCdFkm%2FkADhXAnrnGf5a4FbmK
MQph2cKsr8y8UfWLC6IzvJsClXTnbJBMeuWIqo5zIynS1pm7gf%2F9N3hVc6%2BEeIk0xfl
2tycsUpbL2FoaGk6BAF8hWSWYUXsv59d5Uk%3D",
    "getType": "file"
  },
  "method": "thing.config.push"
}
```

レスポンスメッセージ

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

パラメーター説明

パラメーター	データタイプ	説明
id	文字列	メッセージ ID
version	文字列	プロトコルバージョンです。現在の値は 1.0 です。
configScope	文字列	設定スコープです。 現在、IoT Platform がサポートしているのはプロダクトのディメンションの設定だけです。 値：プロダクト
getType	文字列	設定の目的のファイルタイプです。 現在サポートされているタイプはfileです。 値を file に設定します。
configId	文字列	設定の ID
configSize	Long	設定ファイルのサイズ (バイト単位)
sign	文字列	署名値
signMethod	文字列	署名メソッドです。 サポートされている署名メソッドは Sha256 です。
url	文字列	設定ファイルが格納されている OSS アドレス

パラメーター	データタイプ	説明
code	整数	結果コードです。 詳細は、「デバイスの共通コード (Common codes on devices)」をご参照ください。

9.12 デバイスの共通コード

デバイスの共通コードは、IoT Platform からのリクエストへのレスポンスとして IoT Plaltform に返される結果を示します。

結果コード	メッセージ	説明
200	success	リクエストは成功しました。
400	request error	内部サービスエラーです。
460	request parameter error	リクエストパラメーターが無効です。 デバイスは入力パラメーターの検証に失敗しました。
429	too many requests	システムはビジーです。 このコードは、デバイスがリクエストを処理するにはビジー状態のときに使用できます。
100000-110000	デバイス固有のエラーメッセージ	デバイスは、デバイス固有のエラーメッセージを示すため 100000 ～ 110000 の番号を使用します。