ALIBABA CLOUD

阿里云

物联网边缘计算
最佳实践

文档版本：20201010

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
`Courier字体`	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.使用Modbus TCP连接Modbus从设备

Modbus通信协议遵循主设备和从设备的通信步骤，边缘网关中Modbus驱动使用Master角色（主设备），采取主动询问方式，发送Query Message给Modbus从设备，然后由Modbus从设备依据接到的Query Message内容准备Response Message回传给网关。

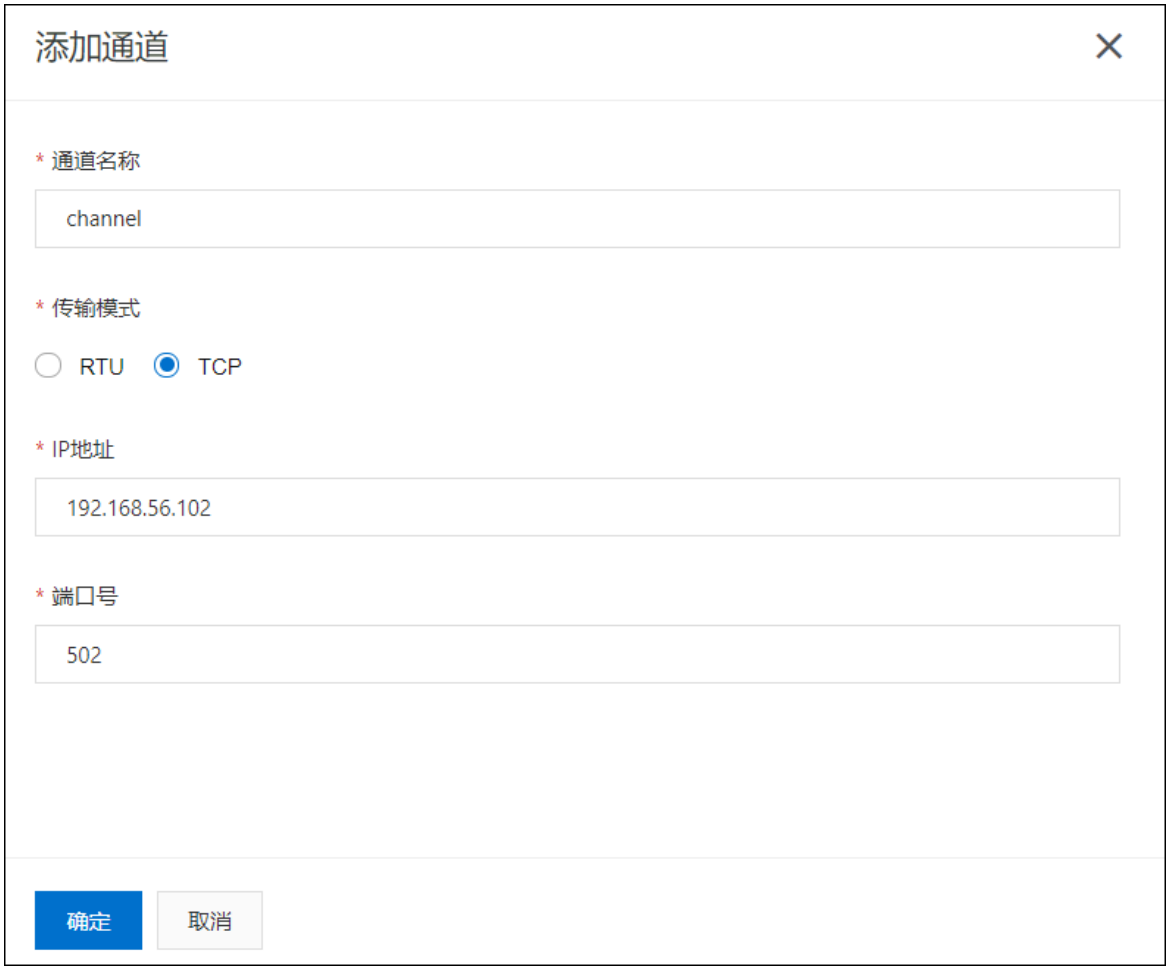
准备工作

1. 准备一个Ubuntu 16.04 x86_64系统，用于运行网关。
2. 准备一个Windows系统的机器，用于运行模拟的Modbus从设备。
3. 从Modbus工具下载地址中获取Modbus从设备模拟软件，下载并安装到Windows机器上。
4. 检查Windows机器的防火墙，确保能正常访问Modbus从设备502端口，若不能访问，请关闭防火墙或者防火墙中允许访问端口。
5. 创建边缘实例并上线网关，具体操作请参见环境搭建。

一、分配Modbus驱动到边缘实例

1. 登录边缘计算控制台。
2. 在左侧导航栏，选择**边缘实例**，单击目标边缘实例右侧的**查看**。
3. 根据网关CPU，分配合适的官方Modbus驱动到边缘实例，具体操作请参见Modbus驱动中的分配驱动步骤。

4. 单击已分配的Modbus驱动，在**设备列表**右侧单击**驱动配置**。
5. 在弹出页面中单击**添加通道**，为Modbus驱动添加通道。参数说明请参见Modbus驱动-驱动配置。



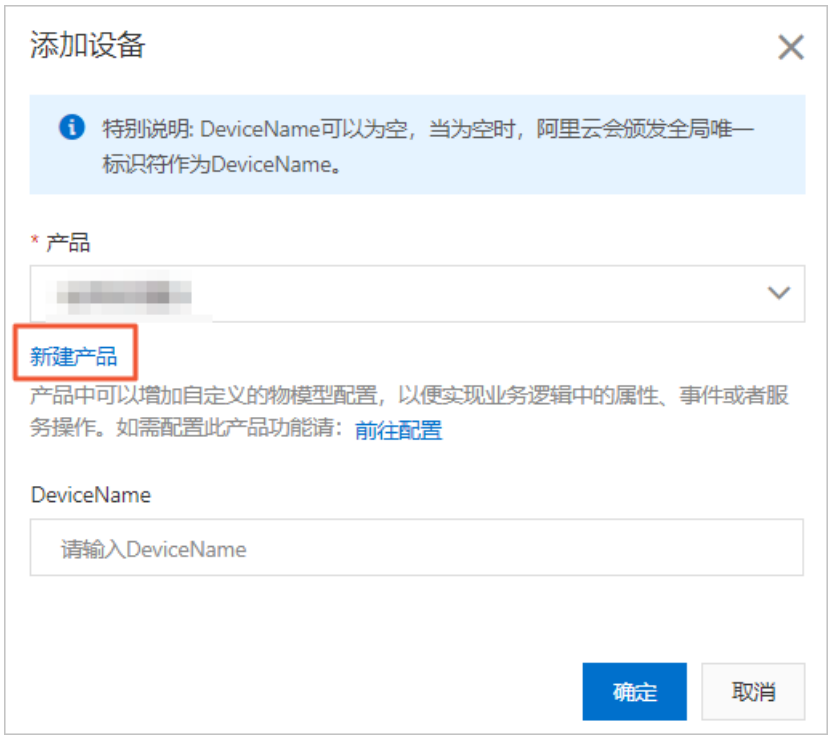
二、Modbus驱动关联子设备

1. 单击**设备列表**区域框下的**分配子设备**，在Modbus驱动下为边缘实例分配设备。您可以分配已有的Modbus设备，也可以根据下面的步骤，新建Modbus设备。

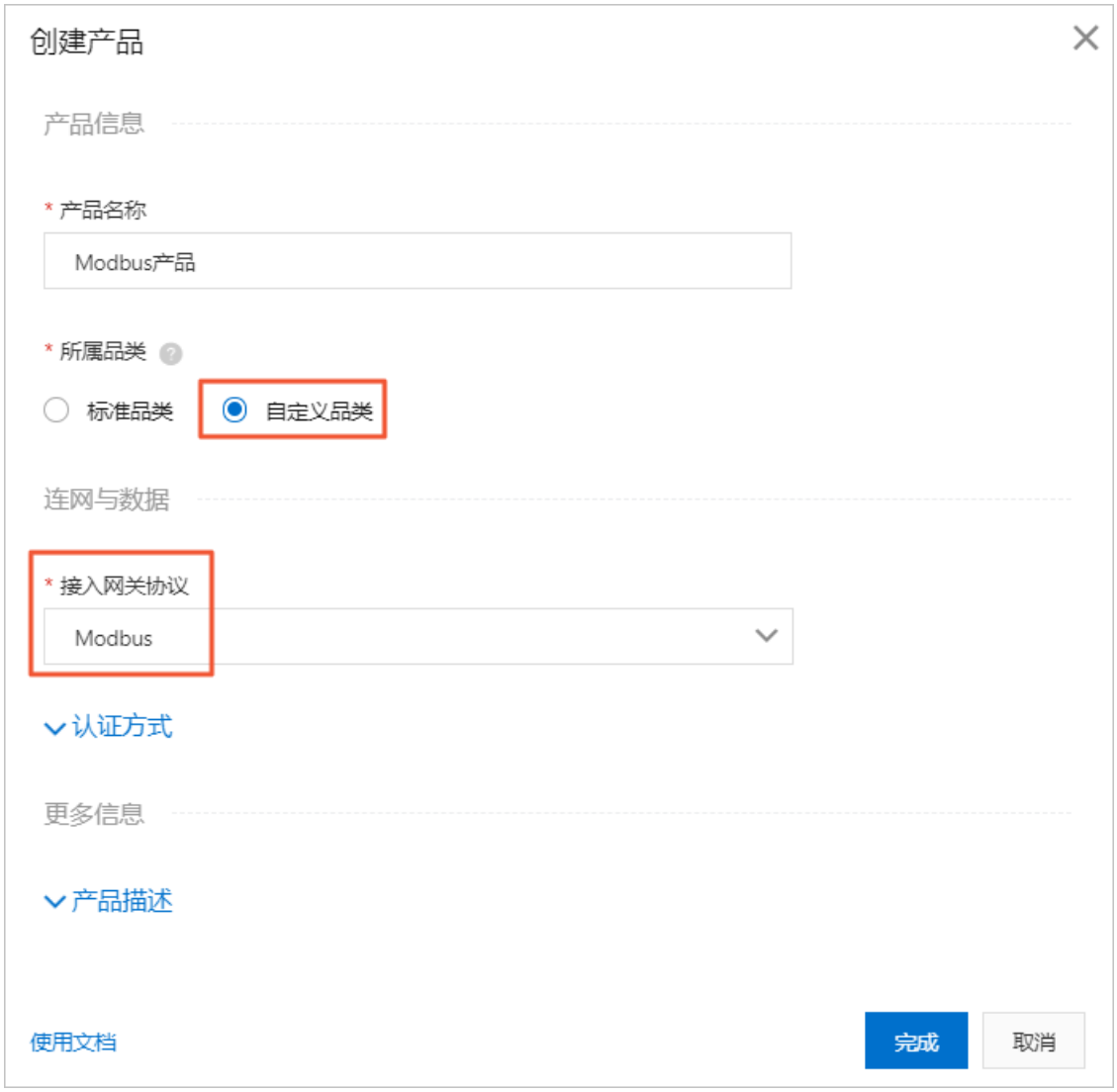
   > **说明** 分配已有的Modbus设备时，该设备所属产品必须接入网关，且接入网关协议为Modbus。详细说明请参见创建产品。

2. 在右侧弹出的**分配子设备**面板中，单击**添加子设备**。

3. 在**添加设备**对话框，单击**新建产品**，创建Modbus设备所属产品。



4. 在**创建产品**对话框设置参数后，单击**完成**。

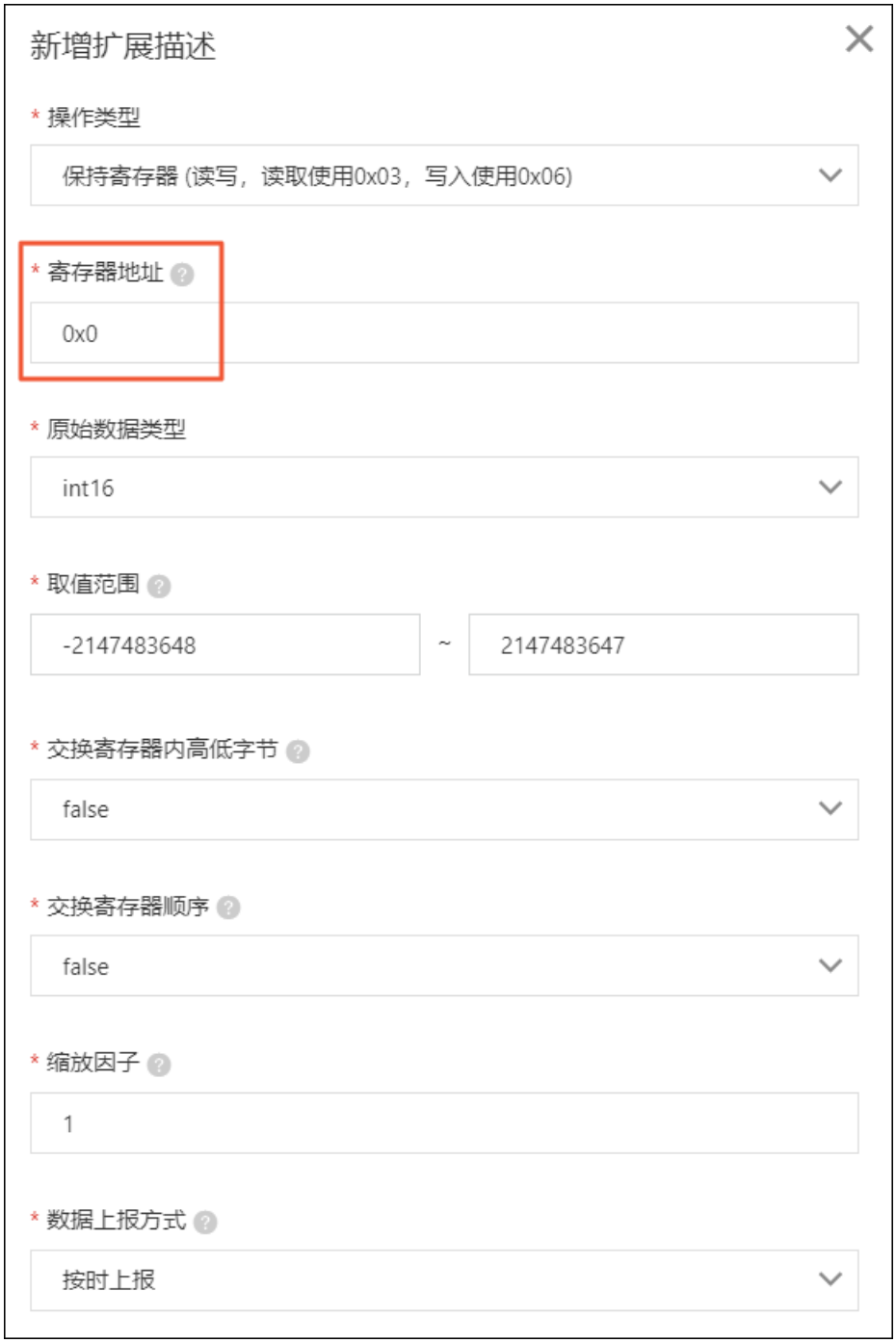


参数说明

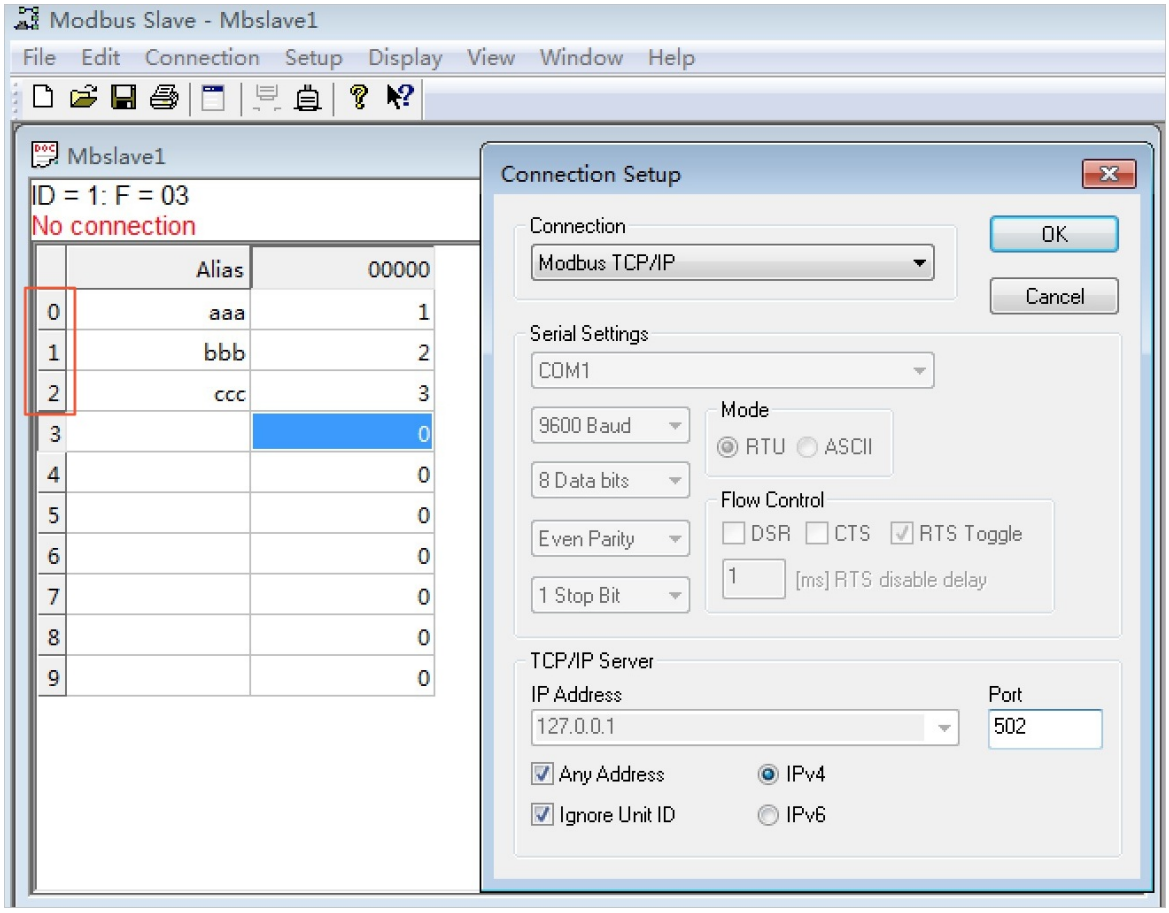
参数	描述
产品名称	设置产品名称，产品名称在账号内具有唯一性。支持中文、英文字母、数字、下划线（_）、短划线（-）、at符号（@）和英文圆括号，长度限制4~30个字符，一个中文汉字算2个字符。
所属品类	选择品类，为该产品定义物模型。此处选择自定义品类。
接入网关协议	此处必须选择Modbus。
认证方式	选择适合您设备的认证方式。详情请参见设备安全认证。
产品描述	添加对该产品的描述。可以为空。

5. 在**添加设备**对话框，产品自动分配已创建的产品，单击产品下的**前往配置**，为产品添加自定义功能，具体操作请参见单个添加物模型。

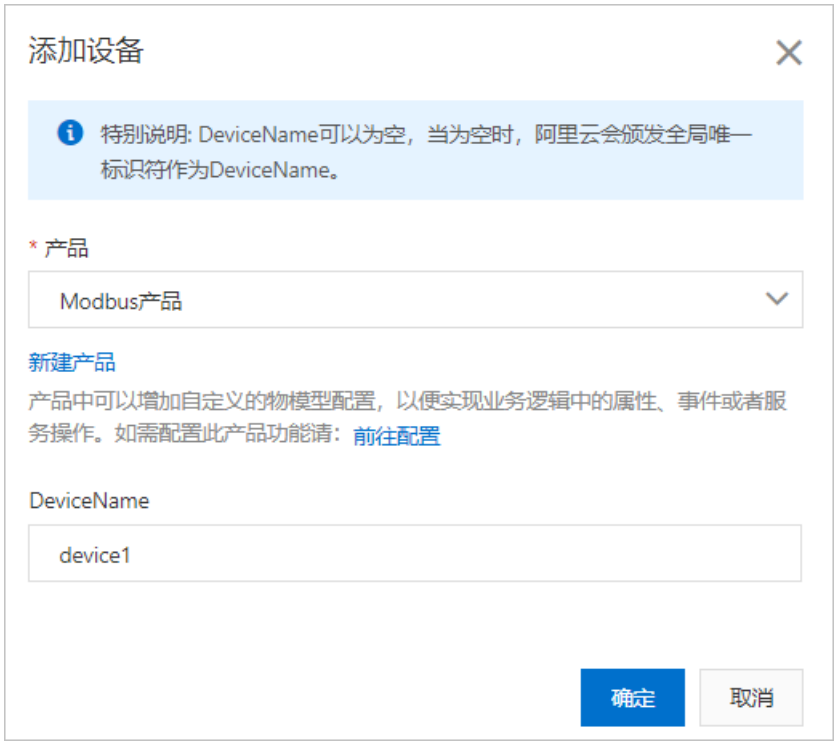
6. **在添加自定义功能对话框，设置属性参数后单击新增扩展描述，设置如下扩展描述。通过新增扩展描述，对Modbus设备的点位进行描述。**



其中，设置Modbus设备的寄存器地址时，需要参考模拟的Modbus从设备中的点位设置，如下图所示。本文示例产品中创建了3个属性点aaa、bbb、ccc分别对应Modbus从设备中点位地址0、1、2三个地址。

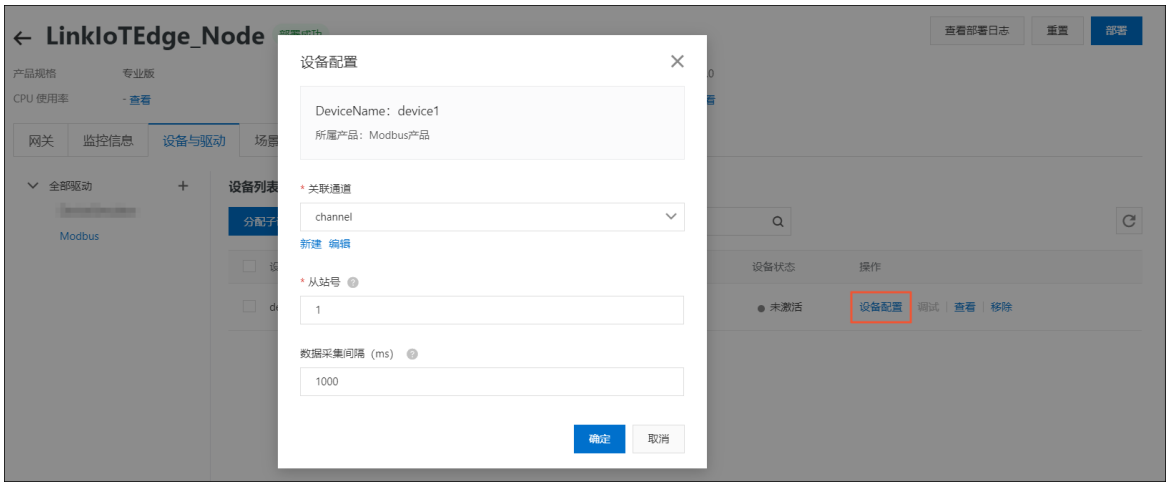


7. 返回边缘计算控制台实例详情页面添加设备对话框，添加Modbus设备。



三、配置并部署边缘实例

1. 将新建的Modbus设备分配到边缘实例。
2. 分配设备到边缘实例后，单击设备名称对应操作栏中的**设备配置**，通过关联通道，关联设备与Modbus驱动。



参数说明

参数	描述
关联通道	选择本文上方一、分配Modbus驱动到边缘实例中已创建的通道。
从站号	从站号请参见本文上方二、Modbus驱动关联子设备中第6步Modbus从设备的ID值，即本示例中从站号为1。

3. 在**实例详情**页面，单击右上角**部署**，部署边缘实例。
4. 登录物联网平台控制台，在左侧导航栏单击**设备管理 > 设备**，找到目标Modbus产品，单击**查看**。在**设备详情**页面单击**物模型数据 > 运行状态**页签，查看设备显示的属性。

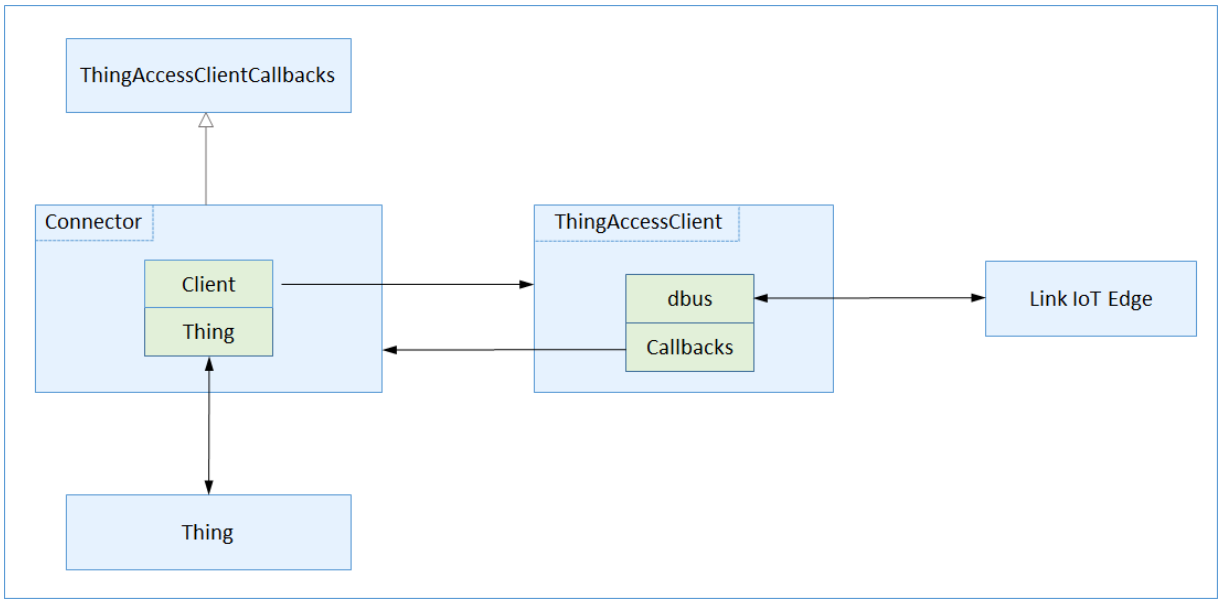
2.使用Connector架构完成设备接入

本文档介绍驱动（设备接入模块）的Connector架构模式。Connector是一种结构清晰又灵活的模式，方便您快速构建驱动。我们推荐您使用Connector架构模式构建驱动程序。

Connector架构模式目前只适用于Node.js和Python的设备接入SDK。

概述

Connector模式架构图如下：



在Connector架构模式中，驱动程序由4个部分组成：

- ThingAccessClient

  此类由设备接入SDK提供，提供多个方法与Link IoT Edge交互，包括数据上行和数据下行。同时接受外部传入ThingAccessClientCallbacks类型回调函数，在收到Link IoT Edge的下行数据时调用回调接口。Connector架构中ThingAccessClientCallbacks的实现类是Connector类。

- Connector

  Connector架构核心组件。对外，Connector组件提供connect和disconnect接口，并接受外部注入Thing接口。对内，Connector组件实现ThingAccessClientCallbacks接口，并在构建ThingAccessClient对象时传入，以建立与Link IoT Edge的连接，并在收到回调指令时转发指令到设备。

- Thing

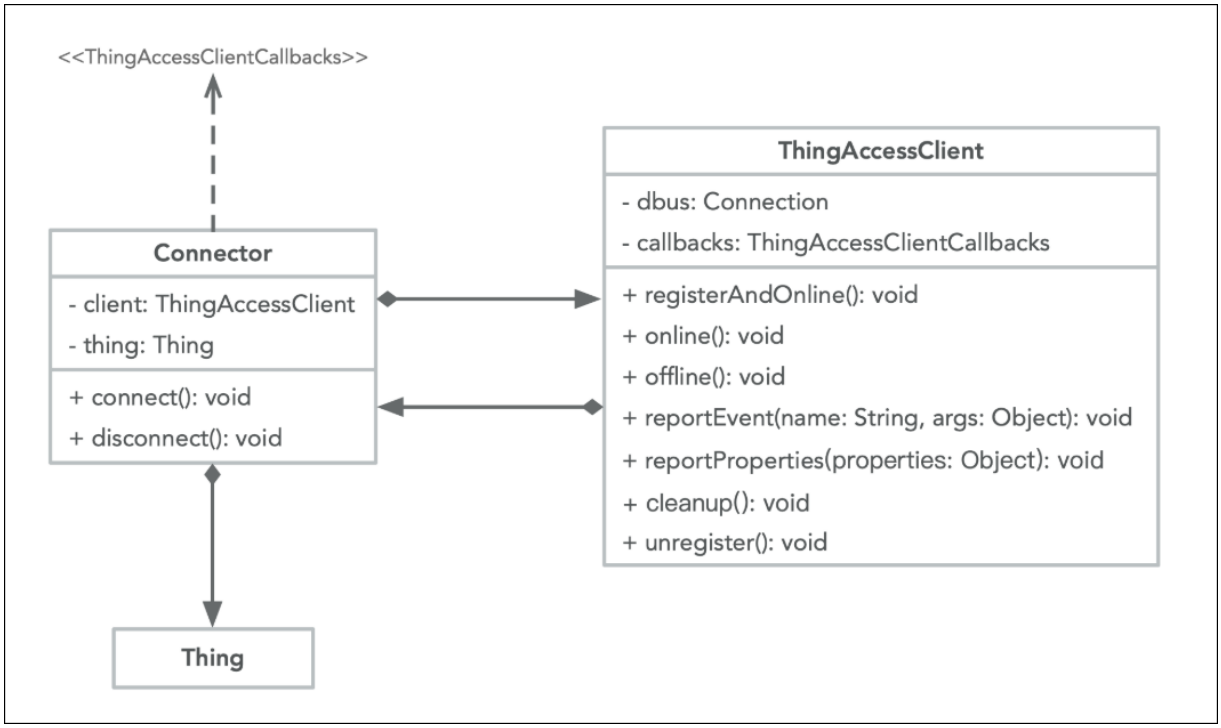
  对物理设备接口提供封装，负责与设备交互，方便Connector组件调用，对外提供面向对象的API。Thing在这里只是一个统称，接入具体设备时为具体设备抽象类，如Light（表示灯设备）。

- Entry

  驱动程序主入口，将会获取驱动配置，初始Thing组件和Connector组件，最终调用Connector组件的connect方法连接设备和Link IoT Edge。也可调用disconnect方法断开设备与Link IoT Edge的连接。

Connector组件是Connector架构中最重要的组件，它通过组合的方式将设备抽象接口（Thing）和Link IoT Edge抽象接口（ThingAccessClient）关联起来，因此而得名Connector。

UML类图如下所示：



操作步骤

下文示例使用Node.js版本设备接入SDK实现Connector架构模式。关于Python版本示例及详细信息可参考Python版本。

Light

本示例演示一个模拟灯的驱动程序设计。

1. 抽象模拟灯类。通过设置isOn属性的*true*和*false*来标识模拟灯的开和关。

   示例代码如下：

```
/**
 * A virtual light which can be turned on or off by updating its
 * <code>isOn</code> property.
 */
class Light {
 constructor() {
  this._isOn = true;
 }

 get isOn() {
  return this._isOn;
 }

 set isOn(value) {
  return this._isOn = value;
 }
```

2. 实现Connector。代码主要包含如下功能：
   - 构造函数接收设备的配置参数和设备抽象对象，内部构造ThingAccessClient以便与Link IoT Edge交互。
   - 实现ThingAccessClientCallbacks的3个回调方法，并在回调方法中调用设备对象接口与设备交互。
   - 提供connect方法和disconnect方法。其中在connect方法里连接Link IoT Edge，在disconnect方法里断开设备与Link IoT Edge的连接。

   示例代码如下：

```
/**
 * The class combines ThingAccessClient and the thing that connects
 * to Link IoT Edge.
 */
class Connector {
 constructor(config, light) {
  this.config = config;
  this.light = light;
  this._client = new ThingAccessClient(config, {
   setProperties: this._setProperties.bind(this),
   getProperties: this._getProperties.bind(this),
   callService: this._callService.bind(this),
  });
 }

 /**
  * Connects to Link IoT Edge and publishes properties to it
```

```
 * Connects to Link IoT Edge and publishes properties to it.
 */
connect() {
  registerAndOnlineWithBackOffRetry(this._client, 1)
    .then(() => {
      return new Promise(() => {
        // Publish properties to Link IoT Edge platform.
        const properties = { 'LightSwitch': this.light.isOn ? 1 : 0 };
        this._client.reportProperties(properties);
      });
    })
    .catch(err => {
      console.log(err);
      return this._client.cleanup();
    })
    .catch(err => {
      console.log(err);
    });
}

/**
 * Disconnects from Link IoT Edge and stops publishing properties to it.
 */
disconnect() {
  this._client.cleanup()
    .catch(err => {
      console.log(err);
    });
}

_setProperties(properties) {
  console.log('Set properties %s to thing %s-%s', JSON.stringify(properties),
    this.config.productKey, this.config.deviceName);
  if ('LightSwitch' in properties) {
    var value = properties['LightSwitch'];
    var isOn = value === 1;
    if (this.light.isOn !== isOn) {
      // Report new property to Link IoT Edge if it changed.
      this.light.isOn = isOn;
      if (this._client) {
        properties = {'LightSwitch': value};
```

```
      console.log(`Report properties: ${JSON.stringify(properties)}`);
      this._client.reportProperties(properties);
     }
    }
    return {
     code: RESULT_SUCCESS,
     message: 'success',
    };
   }
   return {
    code: RESULT_FAILURE,
    message: 'The requested properties does not exist.',
   };
  }

  _getProperties(keys) {
   console.log('Get properties %s from thing %s-%s', JSON.stringify(keys),
    this.config.productKey, this.config.deviceName);
   if (keys.includes('LightSwitch')) {
    return {
     code: RESULT_SUCCESS,
     message: 'success',
     params: {
      'LightSwitch': this.light.isOn ? 1 : 0,
     }
    };
   }
   return {
    code: RESULT_FAILURE,
    message: 'The requested properties does not exist.',
   }
  }

  _callService(name, args) {
   console.log('Call service %s with %s on thing %s-%s', JSON.stringify(name),
    JSON.stringify(args), this.config.productKey, this.config.deviceName);
   return {
    code: RESULT_FAILURE,
    message: 'The requested service does not exist.',
   };
  }
```

```
}
```

3. 获取配置信息，并初始化Connector架构组件。
   - 调用getConfig获取驱动配置。
   - 调用getThingInfos获取设备信息及配置。
   - 初始化Connector组件。
   - 调用connect连接Link IoT Edge。

   示例代码如下：

```
// Get the config which is auto-generated when devices are bound to this driver.
getConfig()
 .then((config) => {
  // Get the device information from config, which contains product key, device
  // name, etc. of the device.
  const thingInfos = config.getThingInfos();
  thingInfos.forEach((thingInfo) => {
   const light = new Light();
   // The ThingInfo format is just right for connector config, pass it directly.
   const connector = new Connector(thingInfo, light);
   connector.connect();
  });
 });
```

LightSensor

本示例演示一个模拟光照度传感器的驱动程序设计。

1. 抽象模拟光照度传感器类。此处模拟光照度传感器有外部监听时会自动运行，在重置外部监听后会停止运行。

   示例代码如下：

```
/**
 * A virtual light sensor which starts to publish illuminance between 100
 * and 600 with 100 delta changes once someone listen to it.
 */
class LightSensor {
 constructor() {
  this._illuminance = 200;
  this._delta = 100;
 }

 get illuminance() {
  return this._illuminance;
 }
```

```
}

// Start to work.
start() {
  if (this._clearInterval) {
    this._clearInterval();
  }
  console.log('Starting light sensor...');
  const timeout = setInterval(() => {
    // Update illuminance and delta.
    let delta = this._delta;
    let illuminance = this._illuminance;
    if (illuminance >= 600 || illuminance <= 100) {
      delta = -delta;
    }
    illuminance += delta;
    this._delta = delta;
    this._illuminance = illuminance;

    if (this._listener) {
      this._listener({
        properties: {
          illuminance,
        }
      });
    }
  }, 2000);
  this._clearInterval = () => {
    clearInterval(timeout);
    this._clearInterval = undefined;
  };
  return this._clearInterval;
}

stop() {
  console.log('Stopping light sensor ...');
  if (this._clearInterval) {
    this._clearInterval();
  }
}
```

```
  listen(callback) {
    if (callback) {
      this._listener = callback;
      // Start to work when some one listen to this.
      this.start();
    } else {
      this._listener = undefined;
      this.stop();
    }
  }
}
```

2. 实现Connector。
   - 构造函数接收设备的配置参数和设备抽象对象，内部构造ThingAccessClient以便与Link IoT Edge交互。
   - 实现ThingAccessClientCallbacks的3个回调方法，并在回调方法中调用设备对象接口与设备交互。
   - 提供connect方法和disconnect方法。其中在connect方法里连接Link IoT Edge，在disconnect方法里断开设备与Link IoT Edge的连接。

   示例代码如下：

```
/**
 * The class combines ThingAccessClient and the thing that connects to Link IoT Edge.
 */
class Connector {
  constructor(config, lightSensor) {
    this.config = config;
    this.lightSensor = lightSensor;
    this._client = new ThingAccessClient(config, {
      setProperties: this._setProperties.bind(this),
      getProperties: this._getProperties.bind(this),
      callService: this._callService.bind(this),
    });
  }

  /**
   * Connects to Link IoT Edge and publishes properties to it.
   */
  connect() {
    registerAndOnlineWithBackOffRetry(this._client, 1)
      .then(() => {
        return new Promise(() => {
          // Running..., listen to sensor, and report to Link IoT Edge.
```

```
      this.lightSensor.listen((data) => {
        const properties = {'MeasuredIlluminance': data.properties.illuminance};
        console.log(`Report properties: ${JSON.stringify(properties)}`);
        this._client.reportProperties(properties);
      });
    });
  })
  .catch(err => {
    console.log(err);
    return this._client.cleanup();
  })
  .catch(err => {
    console.log(err);
  });
}

/**
 * Disconnects from Link IoT Edge.
 */
disconnect() {
 // Clean the listener.
 this.lightSensor.listen(undefined);
 this._client.cleanup()
  .catch(err => {
    console.log(err);
  });
}

_setProperties(properties) {
 console.log('Set properties %s to thing %s-%s', JSON.stringify(properties),
   this.config.productKey, this.config.deviceName);
 return {
   code: RESULT_FAILURE,
   message: 'The property is read-only.',
 };
}

_getProperties(keys) {
 console.log('Get properties %s from thing %s-%s', JSON.stringify(keys),
   this.config.productKey, this.config.deviceName);
 if (keys.includes('MeasuredIlluminance')) {
```

```
    return {
      code: RESULT_SUCCESS,
      message: 'success',
      params: {
        'MeasuredIlluminance': this.lightSensor.illuminance,
      }
    };
  }
  return {
    code: RESULT_FAILURE,
    message: 'The requested properties does not exist.',
  }
 }

 _callService(name, args) {
  console.log('Call service %s with %s on thing %s-%s', JSON.stringify(name),
    JSON.stringify(args), this.config.productKey, this.config.deviceName);
  return {
    code: RESULT_FAILURE,
    message: 'The requested service does not exist.',
  };
 }
}
```

3. 获取配置信息，并初始化Connector架构组件。
   - 调用getConfig获取驱动配置。
   - 调用getThingInfos获取设备信息及配置。
   - 初始化Connector组件。
   - 调用connect连接Link IoT Edge。

   示例代码如下：

```
// Get the config which is auto-generated when devices are bound to this driver.
getConfig()
 .then((config) => {
  // Get the device information from config, which contains product key, device
  // name, etc. of the device.
  const thingInfos = config.getThingInfos();
  thingInfos.forEach((thingInfo) => {
   const lightSensor = new LightSensor();
   // The ThingInfo format is just right for connector config, pass it directly.
   const connector = new Connector(thingInfo, lightSensor);
   connector.connect();
  });
 });
```

3.OPC UA设备接入实践

本文介绍基于OPC UA协议的设备（以下统称设备）接入网关，并与物联网平台交互的方法。

前提条件

- 仅支持使用Link IoT Edge专业版（LE Pro），实现OPC UA设备接入。
- 根据您的实际环境，参考专业版环境搭建完成边缘实例的创建，上线网关。

一、搭建OPC UA Server

OPC UA Server的环境依赖如下表格所示：

依赖组件	版本要求	安装命令
python	≥ 3.5.2	无
pip	≥ 9.0.1	无
opcua	≥ 0.98.3	pip install opcua==0.98.3

根据以下步骤，完成OPC UA Server的搭建。该OPC UA Server模拟一个LED灯设备，该设备具有温度（temperature）属性，高温报警（high_temperature）事件。

1. 下载OPC UA Server。

```
wget http://iotedge-web.oss-cn-shanghai.aliyuncs.com/public/driverSample/opcua_simulation_server.tar.gz
```

2. 启动OPC UA Server。

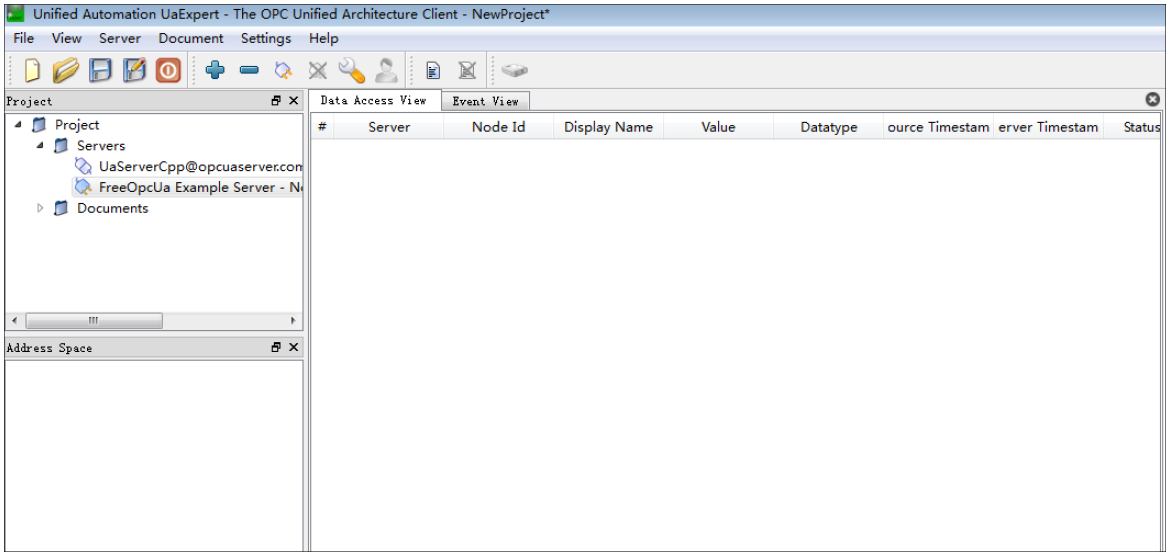
```
tar -zxvf opcua_simulation_server.tar.gz
cd opcua_simulation_server && ./opcua_simulation_server.sh
```

二、安装OPC UA客户端

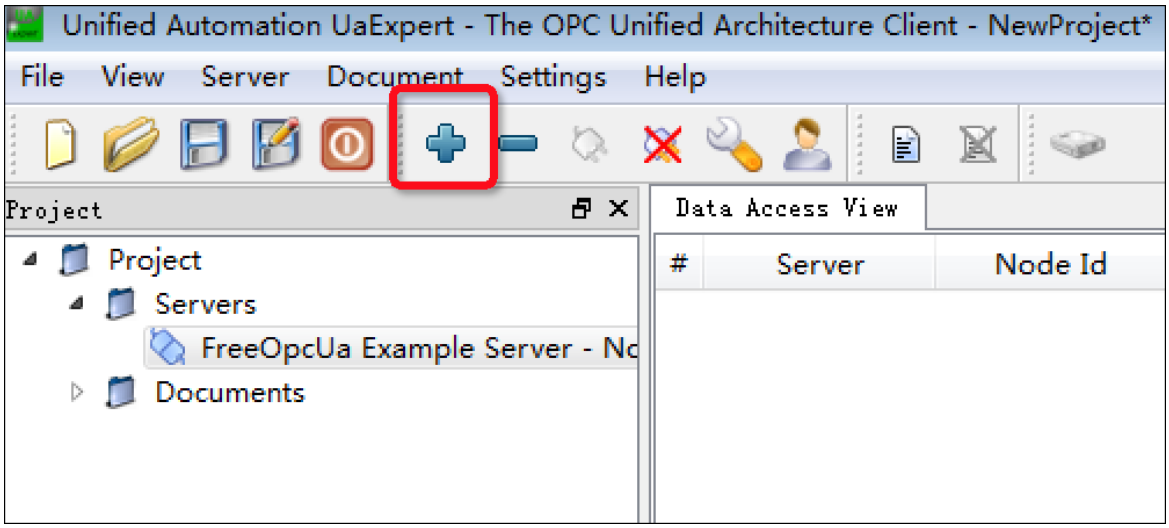
使用OPC UA驱动接入OPC UA设备时需要完成设备配置操作，该操作需要借助OPC UA客户端作为辅助工具，获取OPC UA Server模拟设备信息，用于在控制台创建产品和配置驱动时使用。

本示例使用OPC UA客户端UaExpert工具。

1. 下载并安装OPC UA客户端UaExpert工具。
2. 安装完成后打开UaExpert工具。

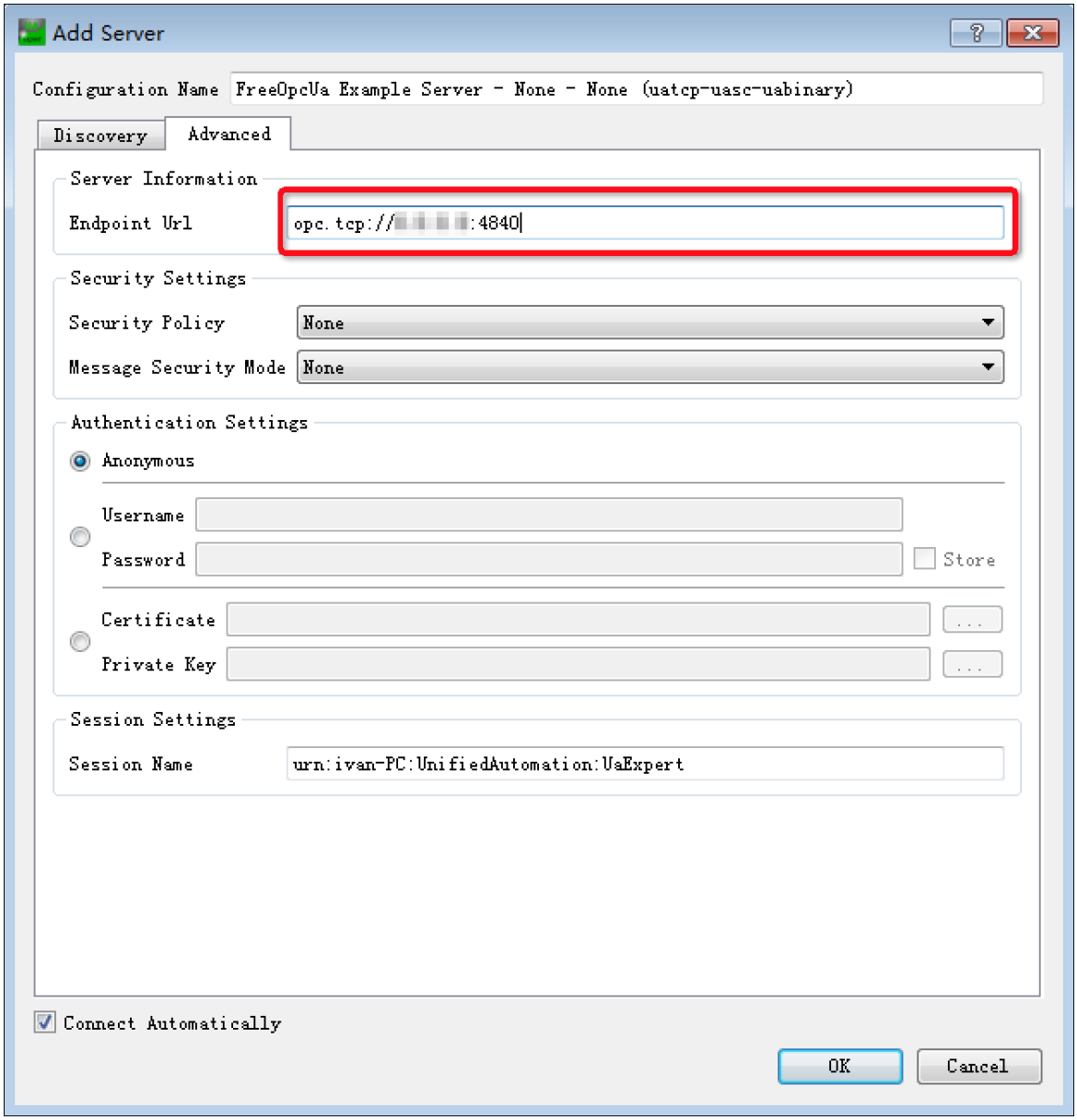


3. 在工具栏中单击“ + ”图标，新增OPC UA Server。

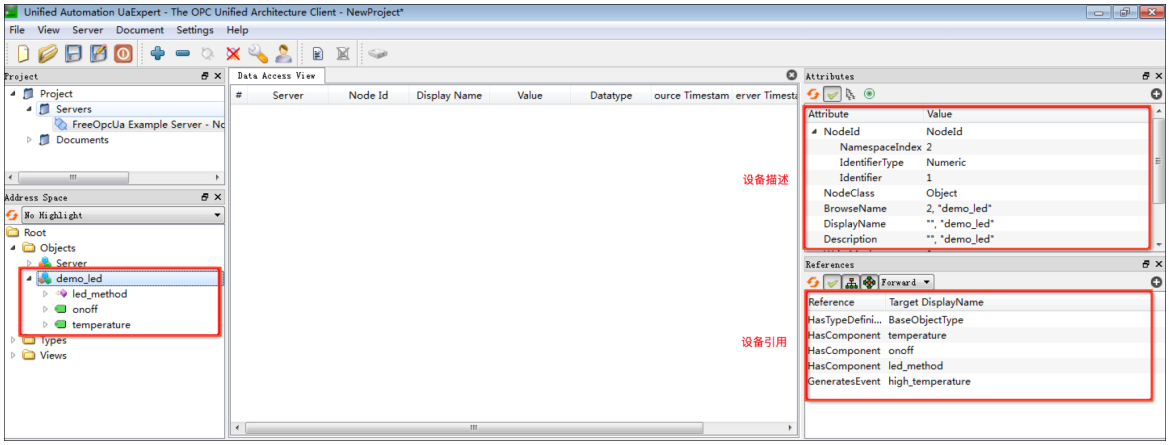


4. 填写OPC UA Server的URL地址，建立与OPC UA Server的连接。URL为 OPC UA Server所在主机的IP地址:端口号 。

> 说明 OPC UA Server示例中默认监听端口为 4840 ，因此OPC UA Server的URL地址格式示例如下：
>
> opc.tcp://192.168.1.1:4840

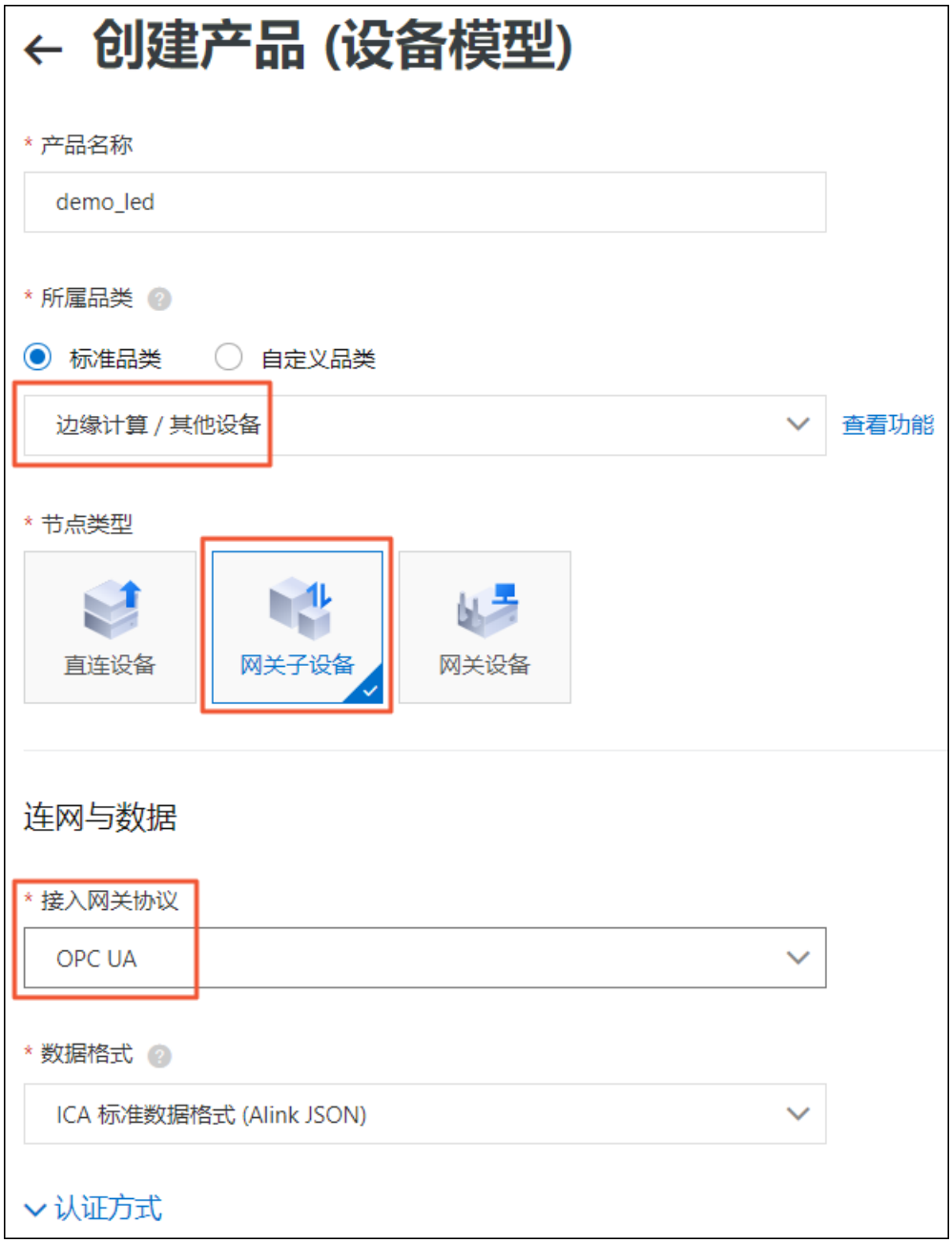


5. 配置完成URL地址后单击OK，显示设备信息。



三、创建基于OPC UA协议的设备

1. 参考创建产品，创建OPC UA产品。



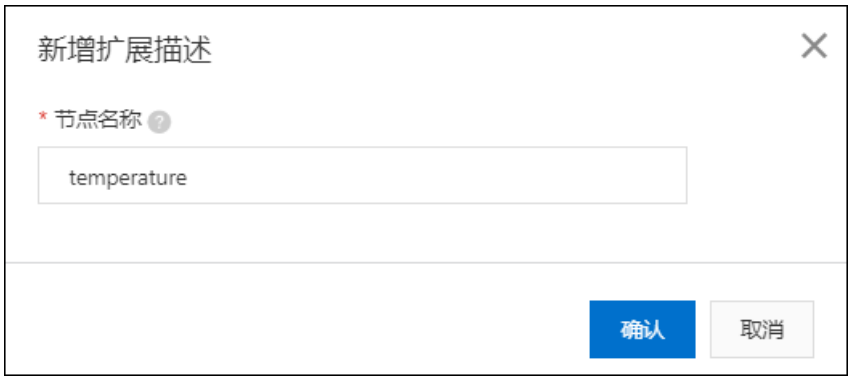
其中，部分参数设置如下：

参数	描述
所属品类	选择标准品类下的边缘计算 > 其他设备。
节点类型	选择网关子设备。
接入网关协议	选择OPC UA。

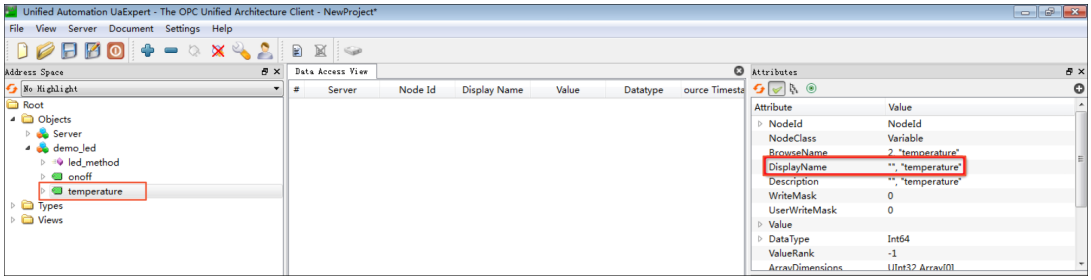
2. 参考单个添加物模型，在产品详情页为OPC UA产品添加如下自定义功能，然后发布上线自定义功能。
   - *添加属性*

a. 根据下图所示，设置属性参数。

b. 设置参数完成后，单击**新增扩展描述**，配置节点名称。



节点名称：设备在OPC UA Server中的变量节点DisplayName的值。

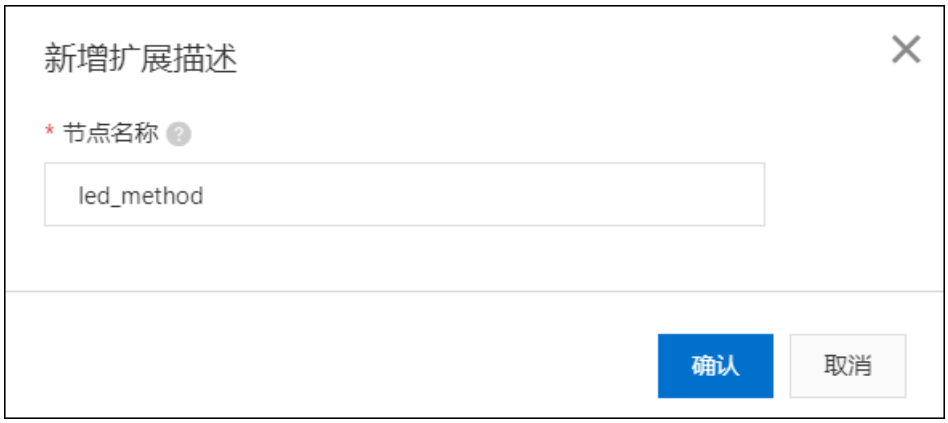


- *添加服务*

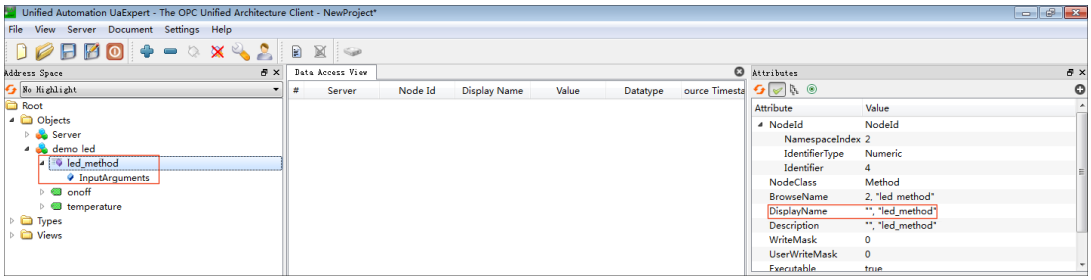
a. 根据下图所示，设置服务参数。

b. 单击输入参数下的增加参数，为产品服务新增参数。

c. 设置参数完成后，单击**新增扩展描述**，配置节点名称。



节点名称：设备method在OPC UA Server中的变量节点DisplayName的值。



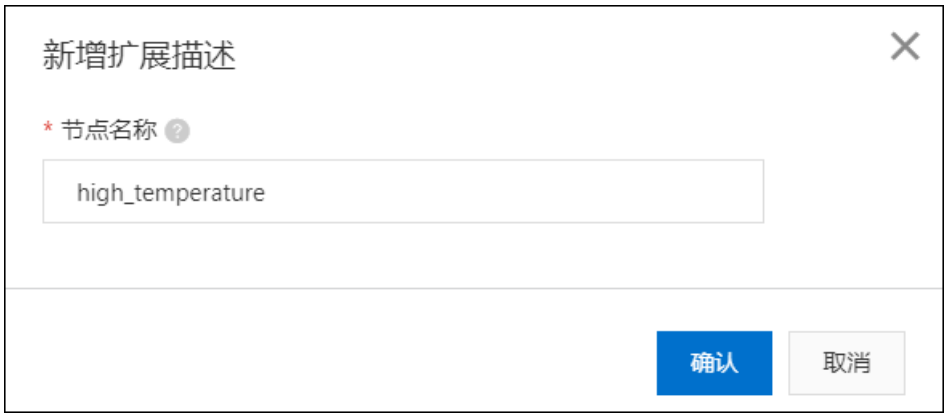
- *添加事件*

a. 根据下图所示，设置事件参数。

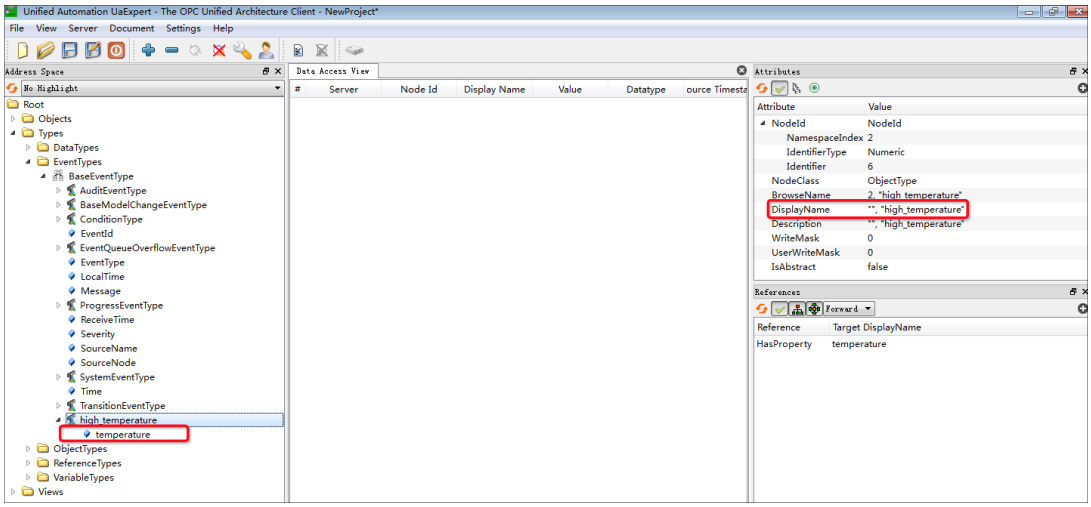
b. 单击输出参数下的**增加参数**，为产品事件新增参数。


新增参数
* 参数名称
temperature
* 标识符
temperature
* 数据类型
int32 (整数型)
* 取值范围
-20
~
100
* 步长
1
单位
摄氏度 / °C
扩展描述
* 参数索引：
0
确认
取消

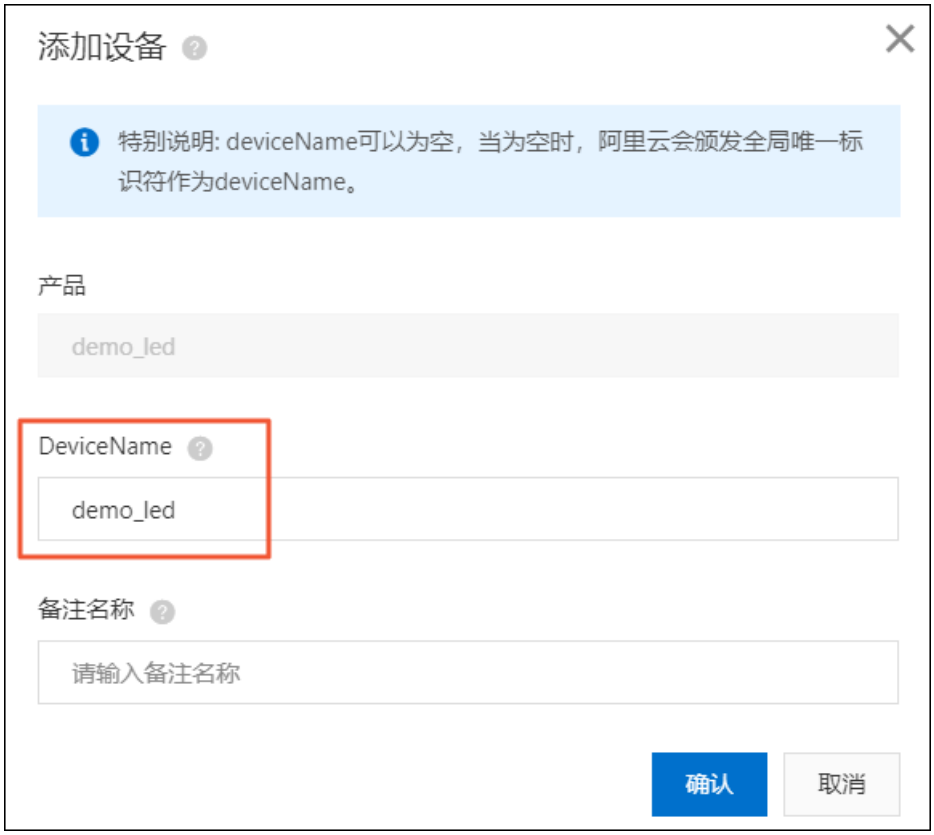
c. 设置参数完成后，单击**新增扩展描述**，配置节点名称。



节点名称：设备事件在OPC UA Server中的变量节点DisplayName的值。



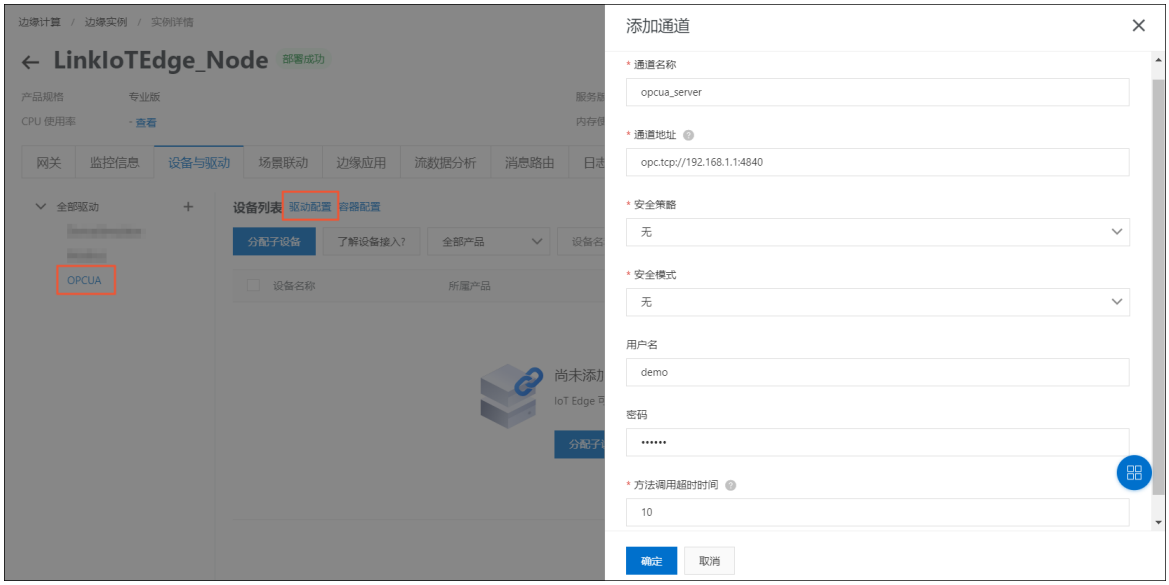
3. 参考单个创建设备，添加设备。



四、配置边缘实例

1. 登录边缘计算控制台，左侧导航栏单击**边缘实例**。
2. 在**边缘实例**页面找到**前提条件**中已创建的边缘实例，单击实例名称后的**查看**。
3. 分配OPC UA驱动到边缘实例中。

4. 选择**OPCUA**驱动，单击**设备列表**后的**驱动配置**，在弹出对话框中单击**添加通道**，设置通道参数。

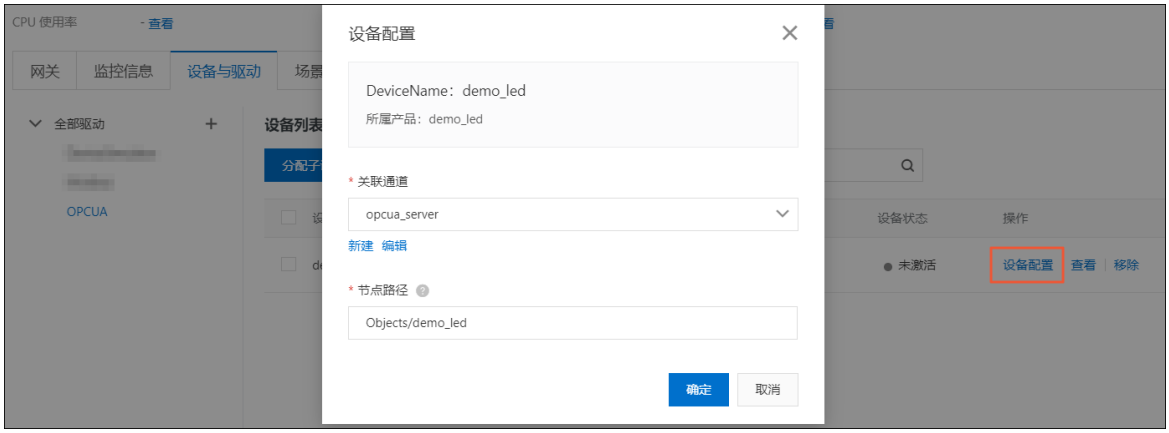


参数说明

参数	描述	配置举例
通道名称	OPC UA通道名称	*opcua_server*
通道地址	OPC UA Server的URL地址	*opc.tcp://192.168.1.1:4840*
用户名	OPC UA Server连接用户名	*demo*
密码	OPC UA Server连接密码	*abc123*
方法调用超时时间	请求调用OPC UA Server调用超时时间	*10*

5. 单击**分配子设备**，在**OPCUA**驱动下为边缘实例分配子设备。

6. 分配子设备成功后，单击设备名称右侧的**设备配置**。



参数说明

参数	描述
关联通道	选择已添加的通道。
节点路径	设备在OPC UA Server中，从Objects开始到设备节点的绝对路径。 例如demo_led设备在OPC UA Server中的路径为*Objects/demo_led*。

7. 在**实例详情**页面右上角单击**部署**，部署边缘实例。
8. 在**实例详情**页面**设备驱动配置**页签中，选择**OPCUA**驱动，查看设备是否在线。


全部驱动
OPCUA
设备列表
驱动配置
容器配置
分配子设备
了解设备接入?
全部产品
设备名称
设备名称
所属产品
关联通道
设备状态
操作
demo_led
demo_led
opcua_server
在线
设备配置
查看
移除