



物联网边缘计算 最佳实践

文档版本: 20201010



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。

- 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。
- 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。
- 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。	
▲ 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。	會告 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。	大) 注意 权重设置为0,该服务器不会再接受新 请求。
⑦ 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。	⑦ 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 <i>,</i> 单击确定。
Courier字体	命令或代码。	执行 cd /d C:/window 命令 <i>,</i> 进入 Windows系统文件夹。
斜体	表示参数、变量。	bae log listinstanceid Instance_ID
[] 或者 [alb]	表示可选项,至多选择一个。	ipconfig [-all -t]
{} 或者 {alb}	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	05
2.使用Connector架构完成设备接入	13
3.OPC UA设备接入实践	24

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。

边缘计算 / 边缘实例 / 实例注荷	分配驱动	×
← LinkloTEdge_Node ™₩₩₩	南方認动 × Modbus	Q Ø
产品現俗 を3000 CPU 使用率 2.83% <u>業</u> 書	驱动络称 驱动语言	通信协议类型 CPU架构 操作
网关 监控信息 设备与驱动 场景联动 边缘应用 流数据分析 满意路由 日	Modbus 富方 C	modbus x86-64 分配
* 金額服助 +	Modbus 當方 C	modbus aarch64 不可分配
新无服动分配	Modbus 當方 C	modbus armv7-hf 不可分配
	Modbus 首方 C	modbus armv7 不可分配
	Modbus 當方 Python 3.5	modbus - 分配
i		
	关闭	

- 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。

添加通道	×
* 通道名称	
channel	
* 传输模式 RTU ③ TCP	
* IP地址	
192.168.56.102	
* 端口号	
502	
确定 取消	

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。

② 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。

分配子设备			×
· · · ·	设备名称	Q	全部未分配
设备名称	所属产品	状态	操作
	1010100	- 181	-
分配 关闭		有新	设备? 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。

添加设备
1 特别说明: DeviceName可以为空,当为空时,阿里云会颁发全局唯一 标识符作为DeviceName。
* 产品
· · · · · · · · · · · · · · · · · · ·
新建产品 产品中可以增加自定义的物模型配置,以便实现业务逻辑中的属性、事件或者服 务操作。如需配置此产品功能请:前往配置 DeviceName
请输入DeviceName
确定 取消

4. 在创建产品对话框设置参数后,单击完成。

创建产品	×
产品信息	
* 产品名称	
Modbus产品	
* 所属品类 💿	
连网与数据	
* 接入网关协议	
Modbus	
◇认证方式	
更多信息	
✔产品描述	
使用文档	完成取消

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 <mark>设备安全认证</mark> 。
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。

添加自定义功能	\times
* 功能类型 💿	
属性 服务 事件	
* 功能名称 🕐	
aaa	
* 标识符 🕐	
aaa	
单位	
请选择单位	\sim
描述	
请输入描述	
	0/100
* 扩展描述 ②	
+新增扩展描述	
确认	取消

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。

新增扩展描述		×
*操作类型		
保持寄存器(读写,读取使用0x03,	写入使用0x06)	\sim
* 寄存器地址 ② 0x0		
* 原始数据类型		
int16		\sim
* 取值范围 ②	~ 2147492647	
-214/483048	~ 214/483647	
* 交换寄存器内高低字节 💿		
false		\sim
* 交换寄存器顺序 😰		
false		\sim
* 缩放因子 ②		
1		
* 数据上报方式 💿		
按时上报		\sim

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。

📲 Modbus Slave - Mbslave1	
File Edit Connection Setup Display Vi	iew Window Help
□ 🗳 🖬 🎒 🛅 🗏 🚊 🤶 🕅	
Mbslave1 ID = 1: F = 03 No connection	Connection Setup
Alias 00000	Modbus TCP/IP
0 aaa 1 1 bbb 2 2 ccc 3 3 0 4 0 5 0 6 0 7 0	Serial Settings COM1 9600 Baud ØRTU ASCII 8 Data bits Flow Control Even Parity DSR CTS T Stop Bit
8 0 9 0	TCP/IP Server IP Address Port 127.0.0.1 ▼ 502 ✓ Any Address ◎ IPv4 ✓ Ignore Unit ID ◎ IPv6

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。

添加设备
特别说明: DeviceName可以为空,当为空时,阿里云会颁发全局唯一 标识符作为DeviceName。
* 产品
Modbus产品 V
新建产品 产品中可以增加自定义的物模型配置,以便实现业务逻辑中的属性、事件或者服 务操作。如需配置此产品功能请:前往配置 DeviceName
device1
确定 取消

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。

← LinkloTEdge_Node	27.27.4974-		查看部署日志 重置	部署
产品规格 专业版	设备配置 X	.0		
CPU 使用率 - 查看 网关 监控信息 设备与驱动 场景	DeviceName: device1 所屬产品: Modbus产品	•		
◇ 全部驱动 + 设备列表 Modbus	* 关联通道 channel 新建 編編 * 从35号 @ 1	Q 设备状态 操作 ● 未激活 设备配置	洞50 直看 移除	C
	教知道決美制回帰(ms) ③ 1000 報知会 取び時			

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 <mark>二、Modbus驱动关联子设备</mark> 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。

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。

ааа	查看数据	bbb	查看数据	ссс	查看数据
1 •		2 •		3 •	
2019/12/26 16:47:54		2019/12/26 16:48:06		2019/12/26 16:48:14	

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),
 });
}
 /**
        ante to Link Int Falme and mublishes meanantis
```

```
" Connects to Link IOI Eage 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:

 $\ensuremath{\textit{//}}\xspace$ 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。

 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 loT Edge,在disconnect方法里断开设备与Link loT 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 (kevs.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:

 $\ensuremath{{\prime\prime}}$ Get the config which is auto-generated when devices are bound to this driver.

getConfig()

.then((config) => {

 $\ensuremath{//}$ 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
рір	≥ 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_s erver.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。

Unified Automation UaExpert - The OPC U	Unified Architecture Client - NewProject*
File View Server Document Settings	Help
🜔 🥟 🕞 🗭 🧿 🔶 🗢 🗞	
Project 🗗 🛪	C Data Access View Event View
Project Project Servers UaServerCpp@opcuaserver.cor FreeOpcUa Example Server - N Documents	# Server Node Id Display Name Value Datatype ource Timestam erver Timestam Status N N N N N N N N N N N
∢ III → Address Space 67 ×	

3. 在工具栏中单击" + "图标,新增OPC UA Server。

Unified Automation UaExpert - The OPC Unified Architecture Client - NewProject*					
File View Server Document Settings Help					
🗋 🥟 🕞 🗭 🧿 🔶 🗖	🗙 🔦 🙎 🖹	8			
Project 🗗 🗸	Data Access View				
4 🃁 Project	# Server	Node Id			
 Servers FreeOpcUa Example Server - No Documents 					

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 。

⑦ 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

Madd Server
Configuration Name FreeOpcUa Example Server - None - None (uatcp-uasc-uabinary)
Discovery Advanced
Server Information
Endpoint Url opc. tcp:// :4840
Security Settings
Security Policy None
Message Security Mode None
Authentication Settings
Anonymous
Username
Password Store
Certificate
Private Key
Session Settings
Session Name urn:ivan-PC:UnifiedAutomation:UaExpert
Connect Automatically
OK Cancel

5. 配置完成URL地址后单击OK,显示设备信息。

Unified Automation UaExpension	ert - The OPC Ur	nified Architectur	e Client - NewProjec	ť							
File View Server Docum	ent Settings	Help									
🗋 💋 🕞 🖉 🧿	4 = 🔅	🗙 🔦 🙎	8 🛛 🧇								
Project	8×	Data Access V	iew					0	Attributes		đ×
4 🗊 Project		# Serve	r Node Id	Display Name	Value	Datatype	ource Timestam erver 1	Timesta	😏 🗹 🦜		0
Servers									Attribute	Value	^
FreeOpcUa Exan	mple Server - No								A Nodeld	NodeId	
Documents									NamespaceInd	ex 2	
									IdentifierType	Numeric	E
< III	÷.						设备	描述	Identifier	1	
Address Space	ē ×	il							NodeClass	Object	
No Highlight	-	i							BrowseName	2, "demo_led"	-
Post		<							DisplayName	"", "demo_led"	
4 Directs									Description	"", "demo_led"	
Server									References		ē X
🔺 🚕 demo_led									🗛 💷 🏟 Forwar	4 💌	-
Ied_method		11								• Disels Mana	ĭ
onoff									Reference larg	et Displayiname	
🖻 💷 temperature									HasTypeDefini Base	ObjectType	
Types							设备	引用	HasComponent temp	erature c	
Views									HasComponent lod	anthod	
									GeneratesEvent high	temperature	
									Concreteserent high,	temperature	
		•						F			

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。

← 创建产品 (设备模型)	
* 产品名称	
demo_led	
* 所属品类 ②	
边缘计算 / 其他设备 🛛 🗸 🗸 🗸	查看功能
* 节点类型 直连设备	
连网与数据	
* 接入网关协议	_
OPC UA 🗸	
* 数据格式 @	
ICA 标准数据格式 (Alink JSON) V	
◇认证方式	

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。

添加自定义功能	×
* 功能类型 💿	
属性服务事件	
* 功能名称 ②	
temperature	
* 标识符 💿	
temperature	
* 数据类型	
int32 (整数型)	\sim
* 取值范围	
-20 ~ 100	
* 步长	
1	
单位	
摄氏度 / ℃	\sim
* 读写类型	
描述	
请输入描述	
	0/100
* 扩展描述 @ +新增扩展描述	

b. 设置参数完成后,单击新增扩展描述,配置节点名称。

新增扩展描述	×
* 节点名称 ??	
temperature	
	确认 取消



Unified Automation UaExpert - The OPC Unified Architectur	e Client - NewProject*								
File View Server Document Settings Help									
🗋 🥔 🕞 🗭 🙆 💠 🗕 🗞 🗙 🔧 🔔	2 🛛 🥯								
Address Space 🗗 🗙	Data Access View					0	Attributes		đ×
😏 No Highlight 🗸 🗸	# Server	Node Id	Display Name	Value	Datatype	ource Timesta	😏 👽 🥾 💿		0
C Root							Attribute	Value	A
A Dijects							Nodeld	NodeId	
Server							NodeClass	Variable	
4 💑 demo_led							BrowseName	2. "temperature"	
Ied_method							DisplayName	"", "temperature"	
onoff							Description	"", "temperature"	
temperature							WriteMask	0	
Types							UserWriteMask	0	
Views							▷ Value		
							DataType	Int64	
							ValueRank	-1	
							ArrayDimensions	UInt32 ArravI01	*

○ 添加服务

a. 根据下图所示,设置服务参数。

添加自定义功能	×
* 功能类型 💿	
属性 服务 事件	
* 功能名称 🕐	
led_method	
*标识符 ②	
led_method	
* 调用方式 💿 ● 异步 〇 同步	
输入参数	
+ 省加麥效	
输出参数 + 増加参数	
描述	
请输入描述	
	0/100
* 扩展描述 👔	
ANCEN INSTART	

b. 单击输入参数下的增加参数,为产品服务新增参数。

新增参数			×
* 参数名称 🕐			
temperature			
* 标识符 💿			
temperature			
* 数据类型			
int32 (整数型)			\sim
* 取值范围			
-20	~ 100		
* 步长			
1			
单位			
摄氏度 / ℃			\sim
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。

新增扩展描述		×
* 节点名称 💿		
led_method		
	确认	取消



Unified Automation UaExpert - The OPC Unified Architectur	e Client - NewProject*	•						- 6 ×
File View Server Document Settings Help								
🗋 💋 🕞 🙆 🧿 🜩 😑 🖄 🔌 🔔	8 🛛 🥯							
Address Space 🗗 🗙	Data Access View				0	Attributes		đ×
😏 No Highlight 👻	# Server	Node Id Display Nam	e Value	Datatype	ource Timesta	😏 👽 💺 💿		0
C Root						Attribute	Value	^
A Dijects						A Nodeld	NodeId	
🖻 💑 Server						NamespaceInde	ex 2	
4 💑 demo led						IdentifierType	Numeric	
▲ [○ led_method						Identifier	4	=
InputArguments						NodeClass	Method	
Image: Second						BrowseName	2, "led method"	
Itemperature						DisplayName	"", "led_method"	
Types						Description	"", "led_method"	
Views						WriteMask	0	
						UserWriteMask	0	
						Executable	true	*

○ 添加事件

a. 根据下图所示,设置事件参数。

* 功能类型 🕜	
属性 服务 事件	
* 功能名称 ?	
high_temperature	
* 标识符 💿	
high_temperature	
* 事件类型 👔 信息 告警 故障 	
输出参数	
+増加参数	
描述	
请输入描述	
0/100	
* 扩展描述	
→ / F3田42 100	

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 ②			
temperature			
* 标识符 💿			
temperature			
* 数据类型			
int32 (整数型)			\sim
* 取值范围			
-20	~ 100		
* 步长			
1			
单位			
摄氏度 / ℃			~
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。

新增扩展描述		\times
* 节点名称 ②		
high_temperature		
	确认	取消





3. 参考单个创建设备,添加设备。

添加设备 ②	×
针别说明: deviceName可以为空,当为空时,阿里云会颁发全局唯一标 识符作为deviceName。	
产品	
demo_led	
DeviceName 💿	
备注名称 ②	
请输入备注名称	
确认 取消	

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。

边缘计算 / 边缘实例 / 实例详情	分配驱动	×
← LinkloTEdge_Node ^{部語成功}	(第方驱动) ∨ OPCUA Q ◎	C
7 BOWINE マムIAX CPU 使用率 2.56 % 查看	驱动名称 驱动语言 通信协议类型 CPU架构 操作	
网关 监控信息 设备与驱动 场景联动 边缘应用 流数据分	OPCUA 省方 Python 3.5 opc-ua - 分配	
✓ 全部構造 新元憲前分配	关闭	8

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。

边缘计算 / 边缘实例 / 实例详情	添加通道
← LinkloTEdge_Node ﷺ mass for the second se	* 通道名称
产品规格 专业版 酸	opcua_server
CPU 使用率 - 查看 内?	字() * 通道地址 @
网关 监控信息 设备与驱动 场景联动 边缘应用 流数据分析 消息路由 日	Jn≓ opc.tcp://192.168.1.1:4840
	* 安全策略
→ → → → → → → → → → → → → → → → → → →	- 无
	* 安全標式
Urcua 设备名称 所属产品	无 ~
	田白女
した しんしょう しんしょ しんしょ	demo
IoT Edg	2 D
	* 75/2*/4/#####1997/01 @
	确定 取消

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。

分配子设备	×
demo_led	Q 全部 未分配
设备名称 所属产品	状态操作
demo_led demo_led	● 未激活 分配
分配 关闭	有新设备? 添加子设备

6. 分配子设备成功后,单击设备名称右侧的**设备配**置。

CPU 使用率 - 查看	i i i	设备配置 X	5
网关 监控信息 V 全部驱动	设备与驱动 场景 + 设备列表	DeviceName: demo_led 所属产品: demo_led	
Carried Section 1	分配子	* 关联通道	Q
OPCUA	ii ii	opcua_server 🗸	设备状态 操作
	de	新建编辑	● 未激活 设备配置 查看 移除
		Objects/demo_led	
		職定 取消	

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 <i>Objects/demo_led</i> 。

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 +	设备列表 驱动配置 容器配	20 20			
No. of Congenities	分配子设备 了解设	醫接入? 全部产品 >	设备名称	Q	
OPCUA	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