基于.NET Core 的阿里云物联网平台技术解决方案

摘　要： 我们(指项目小组成员)的项目提供了对接阿里云物联网平台的技术解决方案，解决硬件驱动使用、数据分析处理、数据推送云端、云端数据接收等一系列问题。我们把接入云平台过程中，各类软硬件、数据处理、云端通讯的技术细节封装到类库,开发者可以在我们的解决方案上，开发自己的软件。我们的类库基于微软的跨平台框架 .NET Core 开发，可以支持使用 Linux/Windows 系统的嵌入式物联网设备，支持跨平台、跨 CPU 架构，软件能够兼容绝大部分较新版本的 Linux/Windows 系统。其中，阿里云物联网平台接入类库(下称 SDK)使得区域内的设备数据互联、负载均匀、智能协作、统一调度。SDK 使用了 MQTT 通讯协议，物联网设备通过 SDK 实现无缝对接云计算平台(阿里云物联网平台)，而无需关注底层通讯。

关键字： .NET Core　物联网　跨平台　嵌入式开发　阿里云 IOT

目录

第 1 章　项目背景

1.1　课题概述

1.1.1 5G

步入 2019 年，多个科技大国大张旗鼓地进行 5G 技术及其相关技术的研发和产业布局，基础设施蓬勃建设中。从华为与高通的商业竞争到美国的政治扼制，美国不遗余力的动员多个国家打压中国华为；每当 5G 有所动作时，就会出现媒体铺天盖地的报道；5G 的重要性不言而喻。

5G 技术具有万物互联、泛在网、低功耗、高速度、低时延的特点。5G 的峰值速率、流量密度、端对端时延有了十倍甚至数十倍的进步空间，重要的是 5G 带来的并不是带宽的提升，而是物联网，5G 的到来为智能设备、智能家居、AR、智慧城市等带来更极致的体验。

1.1.2 云计算与物联网平台

云计算已经有数十年的长足发展，依靠超大规模、高质量高可靠的虚拟计算资源，使得 AI、大数据、物联网等产业快速发展。相对于过去的传统的物联网模式，基于云计算式的物联网平台优势极大，得益于云计算厂商的大规模应用经验、超大规模数据中心、虚拟资源易伸缩、应用服务高可扩展性。

基于云计算的物联网平台，向下连接海量设备，支撑设备数据采集和上传，向上承载大规模数据处理和资源调度。云端可以管控设备、下发命令，实现远程控制调度，同时云端控制台具有数据分析、边缘计算、设备模拟等一系列工具。云计算 IOT 平台亦提供了服务市场，可以方便地使用第三方服务，集成商业软件。

1.1.3 .NET Core

在微软官方文档中这样介绍.NET Core ：

.NET is a free, cross-platform, open source developer platform for building many different types of applications. With .NET, you can use multiple languages, editors, and libraries to build for web, mobile, desktop, gaming, and IoT.[1]

.NET 目前有三个框架：.NET Core、Framework、Xamarin/Mono，这些框架有一个统一的 API 标准：.NET Standard。.NET Core 是极其优秀的开源跨平台框架，由微软官方进行支持，使用了宽松的 MIT 和 Apache 2 开源协议，无需担心商业授权。.NET Core 支持 ARM/X86 等 CPU 架构，支持开发 Web、服务器软件、控制台应用等，能够无缝在 Windows、Linux、Mac 系统下切换，因此作为物联网开发是非常合适的。

1.1.3 项目理想

阿里云 IOT 用户量庞大，经发展多年，其有国内最为完整的云计算生态。物联网开发涉及的技术点较多，我们希望能够开发一个 SDK，能够在嵌入式设备中运行，帮助用户接入阿里云 IOT，降低开发技术跨度，使得开发者无需关注底层通讯原理，只需关注于业务逻辑、功能实现和硬件部分。

1.2 使用阿里云 IOT 的原因

阿里云作为全球前三大公共云服务提供商、国内最大的云服务商，国内云计算市场占有率超过第 2-5 名总和。阿里云拥有多年运营经验，有极具竞争力的产品体系，IOT 生态丰富，功能也较多。

以下是阿里云物联网平台的优势对比结果：[2]

优势点	描述
设备接入	提供设备端 SDK，快速连接设备上云，效率高。 同时支持全球设备接入、异构网络设备接入、多环境下设备接入、多协议设备接入。
性能	具有亿级设备的长连接能力、百万级并发的能力，架构支撑水平性扩展。
安全	提供多重防护保障设备云端安全： 设备认证保障设备安全与唯一性 传输加密保障数据不被篡改
稳定	服务可用性高达 99.9%。去中心化，无单点依赖。拥有多数据中心支持。
简单易用	一站式设备管理、实时监控设备场景、无缝连接阿里云产品，物联网复杂应用的搭建灵活简便。

第 2 章　项目底层架构与技术原理

2.1　阿里云 IOT 平台产品架构

阿里云 IoT 提供大量基于云计算架构的服务，多年的云计算经验，建立了完整的 IoT 生态。帮助用户管理数量庞大的设备集群，同时提供大量功能支持和第三方物联网应用。阿里云 IoT Hub 是平台入口，是设备与云端安全通信的数据通道，设备通过 Hub 可连接到物联网平台，物联网平台亦可将设备数据流转到其他阿里云产品中进行存储和处理。这是构建物联网应用的基础。

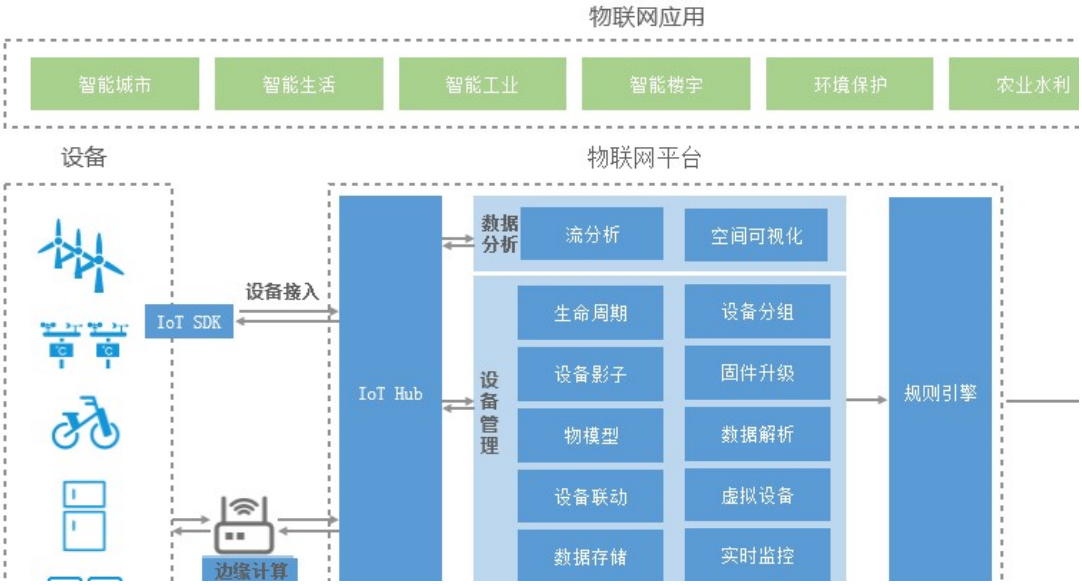


图 1　　阿里云物联网架构　　来源：阿里云文档中心[3]

2.2 项目架构

SDK 中的 AliIOTXFClient 类是设备与云端连接的通道，将各类设备连接起来，与服务器建立通讯。用户无需关注传输的报文格式和数据转换，无需关注数据如何传输，无需担忧网络出错。各类设备硬件能力不同，我们提供硬件驱动直接使用、C/C++间接调用、上层集成、网关等各种技术细节，帮助使用者(可能是个人开发者、厂家、学生等)在设备上集成硬件以及进行数据处理。

针对嵌入式设备中的 Camera、GPIO、RS485/RS232 等硬件接口，我们可以通过直接使用驱动调用硬件、间接使用 C/C++开发的动态链接库操作硬件、通过网关转换协议等各种方式与硬件通讯。

在各个环节我们提供了一些集成类库，减少使用者的开发量；提供完备的技术或说明文档，帮助使用者快速深入；提供 demo 示例，使得使用者迅速掌握开发流程。

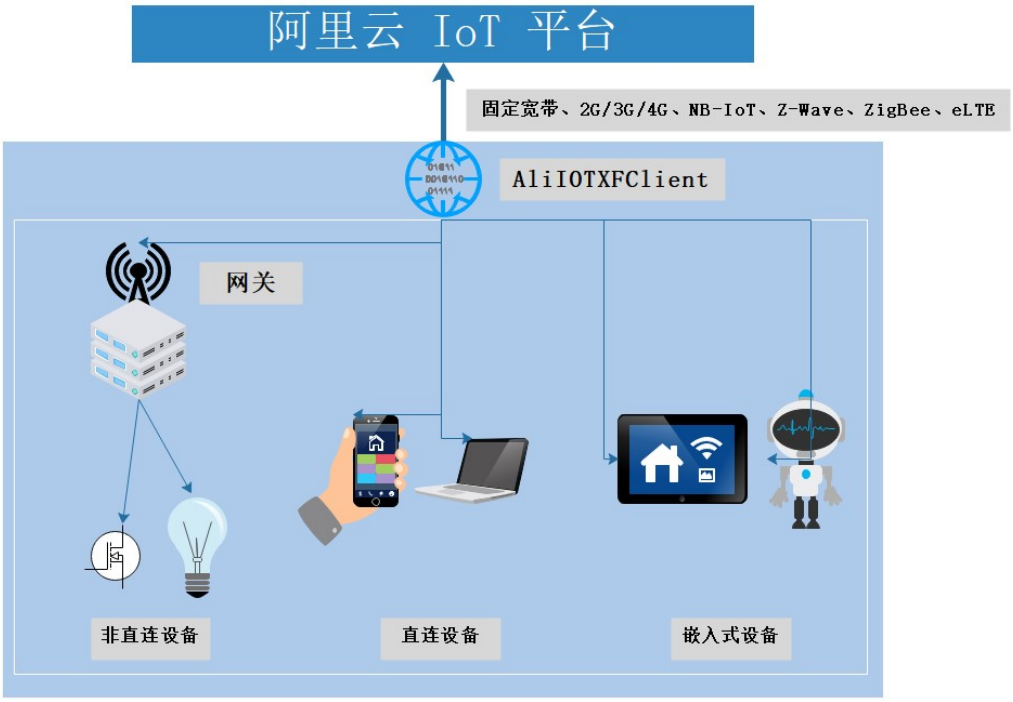


图 2　　物联网系统中 SDK 的作用域和功能

2.3 涉及技术、硬件或环境

2.3.1 MQTT

摘自 IBM 官网：

MQ Telemetry Transport (MQTT) is a lightweight network protocol used for publish/subscribe messaging between devices. The "MQ Integrator SCADA Device Protocol" is an old name for what is now known as MQTT. MQTT is designed to be open, simple and easy to implement, allowing thousands of lightweight clients to be supported by a single server. These characteristics make it ideal for use in constrained environments or low-bandwidth networks with limited processing capabilities, small memory capacities and high latency. The MQTT design minimizes network bandwidth requirements while attempting to ensure reliability of delivery.[4]

2.3.2 阿里云物联网平台

阿里云物联网平台是基于云计算的，因此支持海量设备连接，提供的各类功能和 API，Web 控制台可视化管理。项目实现对阿里云物联网平台的 API 封装，实现数据的上传和接收，实现 Topic 订阅和发布功能，通过项目 SDK 完成设备属性上传和设置、事件上报、设备服务调用等功能。需要大量参考阿里云的官方文档。

2.3.3 .NET/.NET Core

开发环境使用.NET Core SDK，项目框架使用 .NET Standard，实际部署环境使用.NET Core SDK 或 .NET Core Runtime。由于采用.NET Standard，支持框架为.NET Core、.NET Framework、Mono/Xamarin 的项目进行上层开发。.NET 框架如.NET Core 可以支持架构为 ARM/X86 的 CPU 芯片组，能够在 Linux/Windows 系统下运行。

2.3.4 嵌入式硬件和系统

项目使用多个开发板进行测试，对项目的跨平台、多 CPU 架构的兼容性进行了校验。项目使用 ARM 或 X86CPU 架构的设备进行测试，使用嵌入式 Linux(精简 Linux 或、Ubuntu、Debain 等 Linux 发行版)托管运行 SDK 程序。另外，项目接入了 RS485、RS232、Camera、温湿度传感器等物理部件进行测试。

2.3.4 开发环境

Visual Studio 2019 + .NET Core SDK 2.203

第 3 章　项目详细设计

3.1　实现要求

阿里云物联网平台支持 MQTT、CoAP、HTTP/S 等多种协议接入，嵌入式设备往往性能不高、负载能力低，还要保证数据安全可靠地传输，因此我们采用了 MQTT 协议。

MQTT 协议版本为： v3.1.1，规范约束：http://mqtt.org/

我们选择了 M2MQTT 作为 MQTT 通讯类库，https://m2mqtt.wordpress.com/

为了实现跨平台，我们使用.NET Standard 2.0 作为目标框架。

开发环境：Visual Studio 2019，使用.NET Core SDK 2.2.203

硬件测试环境：Raspberry Pi 2/3、Orange Pi 3、米尔科技 MYD-Y6ULX_V04 开发板

要求：

- 封装对接阿里云 IOT 的 API 接口，可靠的 MQTT 通讯
- 可以承载高并发网络 I/O
- 极强的 ARM/X86 CPU 架构兼容性
- 能够支持 Linux 的设备中均可运行

3.2　技术要点

要在各种复杂的网络环境下进行安全可靠的通讯，保证消息能够到达另一端，SDK 采用了 MQTT 通讯协议。MQTT 基于 TCP/IP 协议开发，属于异步通信消息协议，是轻量级、可靠的、低功耗的发布（Subscribe）/订阅（Publish）信息传输协议。MQTT 在时间和空间上，将消息发送者与接受者分离，可以在不可靠的网络环境中进行扩展，适用于设备硬件存储空间有限或网络带宽有限的场景[5]。

开发语言是 C#，设计程序时要避免文件、网络 I/O 流造成的线程阻塞，考虑合理使用异步处理模式，提高底层运行高性能、网络传输高并发。

要在嵌入式设备中运行，必须能够同时支持 X86/ARM 32/64 位 CPU 架构，能够在各类嵌入式 Linux/Windows 系统下运行，使用.NET Core 最合适，.NET Core SDK 版本 2.203，.NET Core Runtime 版本：2.2.4。

由于采用的.NET Standard 2.0 进行开发，所以可以在此基础上可支持使用框架为.NET Core 或.NET Framework 的程序进行上层开发，例如 ASP.NET Core、WPF、Winform。

嵌入式开发往往是硬件底层开发，因此 SDK 也需要提供调用 C/C++程序、对 Linux 硬件的操控的示例。可以通过编写动态链接库文件由 C#程序调用或直接使用 C#调用设备驱动接口。

往往设备数量是以万计算的，还要考虑到设备安全认证。在.NET Core 中，可使用微服务架构，进行设备管理、身份认证和授权，还可以使用 Docker、K8S 进行集群服务。

3.3 SDK 的生命周期

一个类(Object)对应一个客户端(Client)，一个设备可以创建多个连接客户端。对象、程序有其作用域和生命周期，要保证程序能够按照如期地创建、稳健运行、关闭和销毁，必须先构思整体框架。

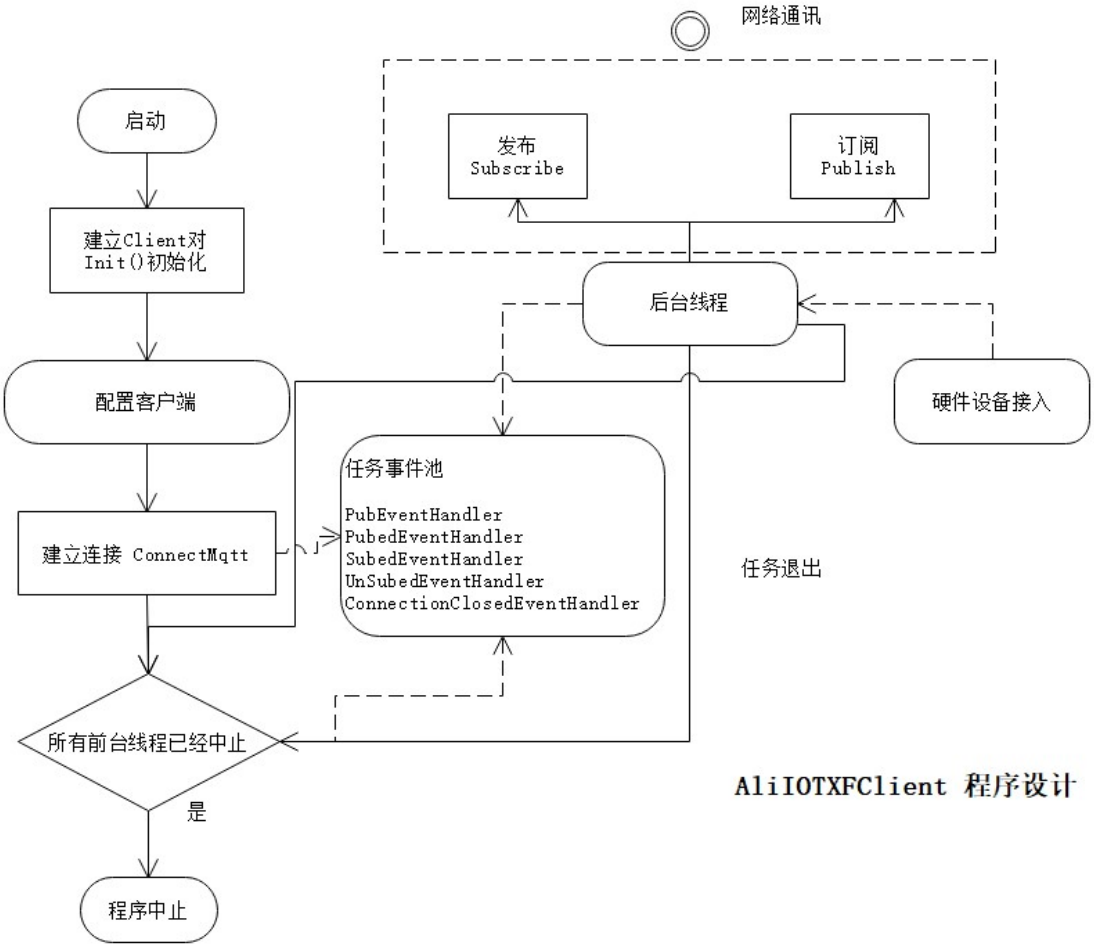


图 3　　项目的架构设计

3.4 SDK 功能模块

项目有模块较多，其中 AliIOTXFClient 是核心部分，有 11 个模块

模块名称	功能作用
字段属性	类内部私有或公开的变量，
创建客户端，建立连接	创建客户端，包括签名、加密、初始化、连接
通用发布	服务器推送 Topic
设备属性上报	推送设备属性，支持透传和 Alink Json
设置设备属性	响应服务器设置设备属性，支持透传和 Alink Json
设备事件上报	设备上报各类预设事件，支持透传和 Alink Json
设备服务调用	服务器向设备下发指令，调用服务，支持透传和 Alink Json
委托处理	添加回调事件、取消回调事件、设置使用默认回调等
设置回调方法	定义各类委托事件，在程序生命周期的不同阶段触发
事件的默认方法	定义相应处理方法，绑定到委托，在合理的条件下会触发
其它函数	用于辅助程序工作的方法

为了同时输出.NET Standard2.0 、.NET Coreapp2.2 两个版本的 dll，修改 AliIOTXFClient.csproj，添加代码：

```
1.    <PropertyGroup>
2.     <TargetFrameworks>netstandard2.0;netcoreapp2.2</TargetFrameworks>
3.     <RootNamespace>MQTTXFClient</RootNamespace>
4.    </PropertyGroup>
```

第 4 章　SDK 代码设计

本章只列举了部分核心核心模块的代码设计场景、描述以及代码定义。

4.1　字段属性

类的内部定义了一些存储消息的字段、属性，较为重要的是 thingModel，用于生成设备属性、服务、事件通讯的 topic，　定义如下：

```
1.    public readonly ThingModel thingModel
```

此字段由于生成设备属性上传和设置、事件上报、服务调用相关服务器通讯地址，包含发送与接收、消息响应，支持透传和 Alink json 两种数据输出方式。详细设计如下：

名称	详细描述(代码中字段名称区分大小写，这里使用小写说明)
UpTopic	用于上传设备属性数据，有以下几个字段 设备上报属性请求 Topic--透传：up_raw 设备上报属性响应 Topic--透传：up_raw_reply 设备上报属性请求 Topic--Alink JSON：post 设备上报属性响应 Topic--Alink JSON：post_reply
SetTopic	设置设备属性，有以下几个字段 下行（透传）响应 Topic：down_raw 下行（透传）响应 Topic：down_raw_reply 下行（Alink JSON）请求 Topic：set 下行（Alink JSON）响应 Topic：set_reply
EventTopic	设备事件上报，有以下几个字段 上行（透传）　请求 Topic：up_raw 上行（透传）响应 Topic：up_raw_reply 上行（Alink JSON）请求 Topic：post 上行（Alink JSON）响应 Topic：post_reply
ServiceTopic	设备服务调用 下行（透传）请求 Topic：down_raw 下行（透传）响应 Topic：down_raw_reply 下行（Alink JSON）请求 Topic：identifier 下行（Alink JSON）Topic：identifier_reply

4.2 创建客户端，建立连接

根据文档，阿里云 IOT 连接域名格式为：

${YourProductKey}.iot-as-mqtt.${YourRegionId}.aliyuncs.com:1883

连接报文需要按照一定格式上传，报文头需要签名加密，同时验证具体的设备 ID。设置函数如下：

初始化连接设置

```
1.  public void Init(string DeviceSecret, string RegionId)
```

生成唯一 clientId

```
1.  public string CreateClientId()
```

创建 MQTT 连接并与服务器通讯，订阅需要的 Topic

```
1.  public void ConnectMqtt(string[] SubTopic, byte[] QOS = null)
```

签名加密

```
1.  public static string Sign(Dictionary<string, string> param, string deviceSecr
et, string signMethod)
```

签名加密的具体实现

```
1.  public static string Sign(Dictionary<string, string> param,
2.  string deviceSecret,
3.  string signMethod)
4.  {
5.    string[] sortedKey = param.Keys.ToArray();
6.    Array.Sort(sortedKey);
7.
8.    StringBuilder builder = new StringBuilder();
9.    foreach (var i in sortedKey)
10.   {
11.     builder.Append(i).Append(param[i]);
12.   }
13.   // deviceSecret
14.   byte[] key = Encoding.UTF8.GetBytes(deviceSecret);
15.   byte[] signContent = Encoding.UTF8.GetBytes(builder.ToString());
16.   //这里根据 signMethod 动态调整，本例子硬编码了： 'hmacmd5' 其它验证方式:
        HmacMD5,HmacSHA1,HmacSHA256 签名算法验证
17.   var hmac = new HMACMD5(key); // deviceSecret
```

```
18.
19.     byte[] hashBytes = hmac.ComputeHash(signContent);
20.
21.     StringBuilder signBuilder = new StringBuilder();
22.     foreach (byte b in hashBytes)
23.       signBuilder.AppendFormat("{0:x2}", b);
24.
25.     return signBuilder.ToString();
26.   }
```

4.3 通用发布

发布(Subscribe)指设备向指定的 Topic 推送消息，数据有透传和 Alink json 两种形式，Topic 有普通 Topic、设备属性、事件上报、服务调用等类型，数据会被转化为 byte 字节流后通过网络传输。对于普通 Topic，阿里云物联网平台不区分透传和 Alink、json，直接对相应的 Topic 上传消息内容。SDK 中提供三种发布方法，适用多种需求。但无论哪种方法，最后都会以 byte 字节流的形式上传。

代码设计：

上传属性或发布 Topic

```
1.     public int Subscribe(string PubTopic, byte[] content)
```

上传属性或发布 Topic

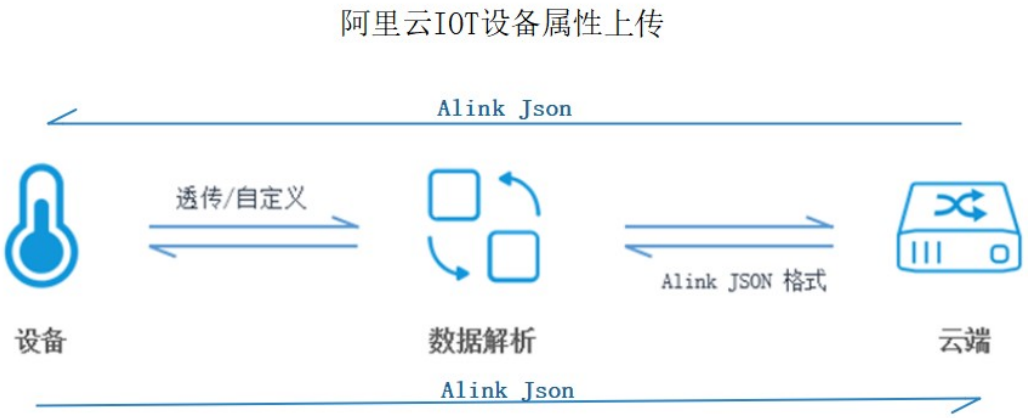
```
1.     public int Subscribe(string PubTopic, string content)
```

上传属性或发布 Topic，会将源数据进行 Base 64 位加密再上传

```
1.     public int SubscribeToBase(string PubTopic, byte[] betys)
```

4.4 设备属性上报

阿里云 IOT 设备属性上传有 ICA 标准数据格式 (Alink json)和透传/自定义，ICA 标准数据格式是阿里云物联网定义规范的的一种 json，需要根据物联网平台定义的标准数据格式编写，本质是 json。透传指上传二进制的原始数据格式，只需要上传数据部分即可，整体报文十分简短，但云端需设置解析脚本对上传的原始数据进行解析。数据上传后，服务器会做出响应，返回一定格式的数据。SDK 同时支持透传和 Alink json，迎合需求，提供多种使用方法。



图：

代码设计：

设备上传属性(透传)，函数会直接上传传入参数而不会对其内容进行任何修改，调用时需设置 byte[]字符的进制。

```
1.  public int Thing_Property_Up_Raw(byte[] bytes)
```

自定义设备上传属性地址、上传属性(透传)，主要用于测试。

```
1.  public int Thing_Property_Up_Raw(string up_rawTopic, byte[] bytes)
```

设备上传属性(透传),会将原始数据转 Base 64 位加密后上传

```
1.  public int Thing_Property_Up_RawToBase64(byte[] bytes)
```

设备上传属性(透传),用于测试，会将原始数据 Base 64 位加密后上传

```
1.  public int Thing_Property_Up_Raw_ToBase64(string up_rawTopic, byte[] bytes)
```

上传设备属性(Alink json)，传入 Json 字符串，可以设置是否转为小写

```
1.  public int Thing_Property_Post(string json,bool isToLwer=true)
```

Json 字符串转为 byte[]格式后上传

```
1.  public int Thing_Property_Post(byte[] json)
```

上传设备属性(Alink Json)，使用模型类，会自动将模型类转为 json 后上传

```
1.  public int Thing_Property_Post<AlinkModel>(AlinkModel model,bool isToLower=
true)
```

4.5 设置设备属性

用于接收服务器向设备设置属性的命令，在接收后返回一定格式的数据作为响应，支持透传和 Alink json 两种数据形式。

代码设计：

收到服务器属性设置命令

```
1.  public int Thing_Property_set(string content,bool isToLower=true)
```

收到服务器属性设置命令，将数据转为 byte[]

```
1.  public int Thing_Property_set(byte[] content)
```

设备属性下发设置，会自动将其转为模型

```
1.  public int Thing_Property_set<SetJson>(SetJson model,bool isToLower=true)
```

4.6 设备事件上报

设备事件上报，用于将设备自身状态信息上传到服务器，根据行业开发通识，设备出现状态变化、数据异常波动、属性超出预设阈值等情况都应上报事件以及记录日志，阿里云物联网平台的设备事件有信息、告警、异常三种类型。SDK 支持透传和 Alink json 两种数据形式。

代码设计：

设备事件上报,以字符串内容上传

```
1.  public int Thing_Event_up_raw(string content)
```

设备事件上报,把原始报文 Base64 加密后上传

```
1.  public int Thing_Event_up_raw_Base64(byte[] content)
```

设备事件上报 Alink JSON

```
1.  public int Thing_Event_Post(string content,bool isToLower=true)
```

设备事件上报 Alink JSON

```
1.  public int Thing_Event_Post<EventJson>(EventJson model,
2.      bool isToLower=true)
```

4.7 设备服务调用

在设备运行的程序中定义多个服务(功能)，一个任务包含一个或多个服务，用于实现特定的工作。在阿里云物联网控制台中设置相应的任务后，云端控制台可以下发命令调用设备程序中的某个任务。SDK 支持透传和 Alink json 两种数据形式。

代码设计：

设备服务调用--透传

```
1.  public int Thing_Service_Down_Reply(byte[] content)
```

设备服务调用

```
1.  public int Thing_Service_Identifier_Reply
2.  (string content,bool isToLower=true)
```

设备服务调用

```
1.  public int Thing_Service_Identifier_Reply<ServiceJsonModel>
2.  (ServiceJsonModel model,bool isToLower=true)
```

4.8 委托处理

SDK 的程序核心部分在整个生命周期中，会开辟多个线程或重用旧线程，处理各种任务。SDK 在连接、发布 Topic、收到订阅 Topic、断开连接等情况下，会触发相应的委托事件。可以预先编写方法绑定到委托事件中，程序会在相应的条件下，自动执行。

代码设计：

订阅回调 - 当收到服务器消息时

```
public uPLibrary.Networking.M2Mqtt.
MqttClient.MqttMsgPublishEventHandler
PubEventHandler;
```

当 QOS=1 或 2 时，收到订阅触发

```
public uPLibrary.Networking.M2Mqtt
.MqttClient.MqttMsgPublishedEventHandler
PubedEventHandler;
```

向服务器发布 Topic 时

```
public uPLibrary.Networking.M2Mqtt
.MqttClient.MqttMsgSubscribedEventHandler
SubedEventHandler;
```

向服务器发布 Topic 失败时

```
public uPLibrary.Networking.M2Mqtt
.MqttClient.MqttMsgUnsubscribedEventHandler
UnSubedEventHandler;
```

断开连接时

```
public uPLibrary.Networking.M2Mqtt
.MqttClient.ConnectionClosedEventHandler
ConnectionClosedEventHandler;
```

4.9 事件的默认方法

SDK 中，预设五个默认的委托方法，对应五个委托事件，传入参数必须与相应的委托定义一致，使用时需绑定到委托事件中。

代码设计：

```
1. public void Default_PubEventHandler(object sender, MqttMsgPublishEventArgs e)
2. public void Default_PubedEventHandler(object sender, MqttMsgPublishedEventArgs e)
3. public void Default_SubedEventHandler(object sender, MqttMsgSubscribedEventArgs e)
4. public void Default_UnSubedEventHandler(object sender, MqttMsgUnsubscribedEventArgs e)
5. public void Default_ConnectionClosedEventHandler(object sender, EventArgs e)
```

第 5 章　硬件设备接入和应用开发问题

在研究过程中,有自身、也有其他开发者使用我们的研究成果时碰到的问题,解决这些问题、完成需求，给我们积累了大量经验。

要把设备接入物联网平台，要先解决各类硬件的数据读取、数据处理、数据分析、数据存储等。一个设备，可能是一台电脑、一个嵌入式开发板；也可能是智能小车、智能机器人；甚至只是一个电灯、一个音响。根据硬件能力，我们提供大量参考代码、文档、和代码示例，帮助使用者快速处理各种设备的接入以及进行软件应用的集成。

树莓派、米尔科技 MYC-Y6ULX 系列开发板等设备具有完备的通讯能力、能够接入互联网、能够使用 Linux/Windows 系统，则可以单独部署应用。一般此种设备往往作为一个控制平台,集中处理一定范围内其它设备的数据,接入传感器、摄像头等硬件。

不具有完整模块和通讯能力的硬件设备，则需“挂载”到其它设备上，受中心设备的控制。中心设备对各种接入到自身的硬件进行协议转换、数据收集、数据分析、与服务器双向通讯等。例如指纹识别器、摄像头、USB 存储器、电灯等。

下面列举几个研究的主要难点、注意点。

5.1　摄像头

摄像头在 Linux 中属于字符设备，采集视频流后或在采集视频时，需要进行一系列的编码解码，才能看到完整的视频，算法处理十分复杂。摄像头设备的接入有 USB、Onvif 和 CSI 等。

V4L2 是 Linux 的 Camera 内核架构，通过 V4L2 提供统一接口，应用程序可以操纵摄像头。以采集视频举例,采集流程有打开设备、查看设备的视频能力(视频编码、技术标准等)、设置采集视频的参数、申请足够的缓冲区存储视频流、通过内存映射将缓冲区的视频流地址映射到用户空间、开始采集视频流、重复采集视频流、采集完毕释放缓冲区和关闭摄像头。所有对摄像头的操作均是通过 V4L2 完成。

查看设备已接入的摄像头及其型号特征(Linux 命令)：v4l2-ctl --list-devices

摄像头的使用有多种方式，通过 C/C++程序调用 V4L2，手动解决各个技术细节；通过开源的 vuc_stream 程序，使用网络的方式远程获取；有人脸识别需求，可以使用 Opencv；还有大量的开源工具，例如 FFmpeg。我们在摄像头的使用上，

提供了示例和文档，供使用者参考，选取合适的开发方式。下面举例部分方式。

5.1.1 UVC Stream

UVC Stream 是一个使用 C 语言编写的软件，采集摄像头后，通过 TCP/HTTP 协议，提供远程视频或图像下载使用， Http Basic Authentication 进行登陆认证。

在 Linux 设备中，后台运行 UVC Stream 服务

```
1. nohup ./uvc_stream -d /dev/video1 -y -P 123456 -r 800x600 &
```

设备本地检测是否正常采集视频

```
1. wget http://uvc_user:123456@127.0.0.1:8080/snapshot.jpeg
```

打开远程 URI 查看实时采集(需先配置网络)

```
1. http://ip:8080/snapshot.jpeg  #实时图像
2. http://ip:8080/stream.mjpeg  #实时视频
```

5.1.2 OpencvSharp

Opencv 是知名的跨平台计算机视觉库，基于 BSD 许可（开源）发行，适合用于计算机视觉领域。可以通过 Opencv 开发人脸识别、以图搜图、智能安防、VR/AR、3D 等应用。Opencv 有众多移植版本，例.NET 的 OpencvSharp。Opencv 使用时，需要注意跨平台的 CPU 架构和系统位数，否则会报错。

因 Opencv 动态库非 ARM 平台版本的报错：

Unable to load shared library 'OpenCvSharpExtern' or one of its dependencies. In order to help diagnose loading problems, consider setting the LD_DEBUG environment variable: libOpenCvSharpExtern: cannot open shared object file: No such file or directory
***** An exception has occurred because of P/Invoke. *****
Please check the following:
(1) OpenCV's DLL files exist in the same directory as the executable file.
(2) Visual C++ Redistributable Package has been installed.
(3) The target platform(x86/x64) of OpenCV's DLL files and OpenCvSharp is the same as your project's.
System.DllNotFoundException: Unable to load shared library 'OpenCvSharpExtern' or one of its dependencies.

Opencv 使用 C 语言/C++开发，OpencvSharp 是在 Opencv 的动态库上做了大量的调用封装。Opencv 可以轻松地在各种 CPU 下编译运行，而 OpencvSharp 使用需要注意很多地方。OpencvSharp 依赖于 Opencv 库，因此要先编译特定平台的动态库，进一步编译 OpencvSharp，即可在.NET 框架中使用。先特定平台或使用交叉编译的方式，使用 gcc 编译 opencv、opencv_contrib，然后在 OpencvSharp 中调用动态库，即可使用 Opencv。

5.2 ICU(International Components for Unicode)

ICU 是用于支持 Unicode 的开发包。计算机字符编码有 GB2312、UTF8、ASCII、Unicode 等，在全球化软件中，各国文字编码标准不一，输出字符时容易乱码，Unicode 统一了地球上各个国家的字符编码。有些 Linux 系统做了精简处理，可能移除了 ICU，这时需要手动安装。

Linux 安装 ICU：

```
# Debain、Ubuntu 系统
apt install libicu60
# 或
Apt install libicu60
# Centos 系统
yum -y install libicu-devel
yum install libunwind
yum install libicu
```

对于只使用 Linux Kernel 的系统，只能通过交叉编译的方式把 ICU 编译到系统镜像中，再把新的系统刷入设备。发布应用(.NET 应用)时，亦可预先在程序里配置，以适应不同环境。

在项目的 csproj 文件中，加上

```
<ItemGroup>
  <RuntimeHostConfigurationOption Include="System.Globalization.Invariant"
 Value="true" />
 </ItemGroup>
```

5.3 数据上传阿里云

设备通过 SDK 连接到阿里云物联网平台，数据格式需遵循阿里云物联网平台的数据规范，目前数据有透传和 Alink json 两种形式，我们的 SDK 均支持。

透传使用二进制数据，可以设置使用 Base64 编码在转为二进制上传，使用透传形式上传数据，云端需要设置解析脚本来处理相应的二进制数据。使用透传形式上传数据，极其容易出错，二进制数据采用八位形式，并且需要包含帧类型、请求 ID、属性数据等，每个字节都必须严格对应并且顺序不能乱。编程语言中，会对数值类型进行隐式转换(十进制)，因此必须在处理数据时，使用强制类型转换、以相应进制存储数据。

云端解析脚本接口样例：

```
// 将上传的二进制数据解析到云端
function protocolToRawData(jsonObj) {
/*自定义解析代码*/
    return rawdata;
}
//将云端数据转为二进制下发到设备
function rawDataToProtocol(rawData) {
/*自定义解析代码*/
    return jsonObj;
}
```

Alink json 是阿里云制定的一种 json 标准，定义了设备上传数据是需设置的数据形式，需预先在云端定义产品的物理模型。C#中，可以定义 class 作为模型存储数据，设备把硬件数据收集后，赋值到模型中，SDK 在将相应的模型转为 Alink json 格式，使用者无需编写数据转换。

使用 SDK 转换模型位 Alink json

```
// 推送内容到特定的 Topic
client.Thing_Property_Post<TModel>(model, true);  //使用泛型
```

5.4 Docker 容器化

云计算时代，互联网架构已经非常成熟，微服务、分布式、容器化、跨平台等各种技术架构层出不穷，而物联网设备可能是成千上万的数量，必须合理使用相应的架构才能支撑大规模的设备集群。我们的 SDK 和部分解决方案使用.NET Core 开发,得益于.NET Core 优良的架构,我们的技术方案可以轻松迁移到Docker 中，享受跨平台、容器化带来的便利服务。SDK Demo(我们编写了相应的代码示例供其他开发者参考)使用华为自研的 EulerOS 作为容器镜像，提供技术文档作为参考。

Docker 构建部分过程

```
# 阿里云安装脚本
curl -fsSL https://get.docker.com | bash -s docker --mirror Aliyu
# docker 官方安装脚本
sudo wget -qO- https://get.docker.com/ | bash
```

导入华为 EulerOS 镜像系统

```
1.  docker import HyperStack_TD_V100R005C00B038-docker.aarch64.tar.xz
```

第 6 章　参考资料

在设计项目过程中，为了支持阿里云阿里云平台、嵌入式系统的硬件设备，参考了大量的技术文档(网页)，论文有部分内容来源于文档的摘取或整理，特此说明。

[1]

内容摘取来自.NET Core 官网

https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet

[2]

资料来源于阿里云物联网开发文档中心

https://dev.iot.aliyun.com/doc?spm=a2c56.193971.iot-bavBar.98.666025c89YYkeY

[3]

图片来源于阿里云物联网开发文档中心

https://dev.iot.aliyun.com/doc?spm=a2c56.193971.iot-bavBar.98.666025c89YYkeY

[4]、[5]

MQTT 协议官网 http://mqtt.org

MQTT 协议规范 http://mqtt.org/documentation

其它参考资料

阿里云 IOT 文档中心　https://dev.iot.aliyun.com/doc

微软.NET 技术文档中心 https://docs.microsoft.com/zh-cn/dotnet/

Linux 官网 https://www.linux.org

Raspberry　https://www.raspberrypi.org

OrangePi　http://www.orangepi.cn

米尔科技　http://www.myir-tech.com