**目录**

[课程内容 2](#_Toc521852873)

[课程目标 2](#_Toc521852874)

[课程准备 3](#_Toc521852875)

[课程知识点 3](#_Toc521852876)

[脚本解析 3](#_Toc521852877)

[物联网平台侧开发 4](#_Toc521852878)

[新建产品和设备，定义功能 4](#_Toc521852879)

[在线编辑脚本 6](#_Toc521852880)

[模拟运行脚本 10](#_Toc521852881)

[设备端开发 12](#_Toc521852882)

[新建项目 12](#_Toc521852883)

[代码开发 14](#_Toc521852884)

[编译下载 15](#_Toc521852885)

[在线调试 15](#_Toc521852886)

[后续课程 17](#_Toc521852887)

**脚本解析实验**

在上一课中，我们学习了物模型与在线调试两大*设备*管理能力。定义好物模型后，AIoTKIT开发板通过构造*Alink* JSON格式的数据，基于*Alink* *Topic*与云端通信，完成*属性*、*服务*及*事件*消息的上下行。但是在一些业务场景中，*设备*可能配置较低，资源受限或者对网络流量有要求，此时不再适合直接按照定义好的物模型构造*Alink* JSON格式的数据与云端通信，透传是更好的选择。本次课程将为读者详细介绍物联网平台提供的，负责解析透传数据的“脚本解析”功能。



**课程内容**

* 介绍脚本解析的功能和实现流程
* 介绍脚本的在线模拟运行功能
* 实现AIoTKIT开发板与平台间的透传通信

**课程目标**

* 掌握基本的脚本编写能力
* 熟悉脚本的模拟运行功能
* 结合物模型与在线调试功能，完成开发板与平台间的透传实验

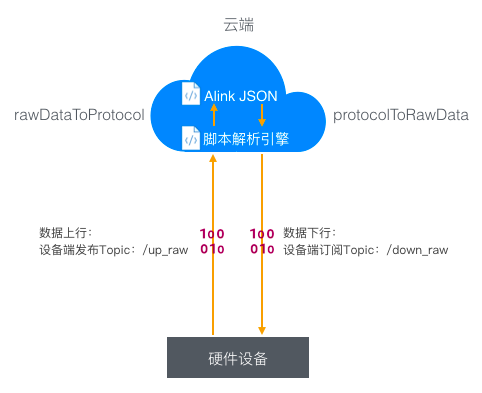
**课程准备**

* 运行Windows系统的 PC机一台
* AIoTKIT开发板一块
* Mk3080 wifi模组
* micro USB连接线
* 安装有AliOS Things Studio插件的VSCode
* AliOS Things 1.3.3版本或更高版本
* ST-Link 驱动程序
* 开通阿里云物联网平台*产品*

**课程知识点**

**脚本解析**

当*设备*将数据透传至云端时，云端可以运行我们在物联网平台上编写的脚本对透传上来的数据进行解析，将其转换为符合*产品*物模型定义的*Alink* JSON格式的数据，数据解析流程如下。



目前，转换脚本使用JavaScript语言开发，仅支持符合ECMAScript5.1标准的JavaScript语法。

物联网平台为开发者提供了在线脚本编辑器，支持保存草稿，删除草稿，或者从草稿中恢复编辑，还支持对脚本的静态语法进行检查。同时，物联网平台还支持对脚本进行模拟数据调试，即用户可以输入上下行数据进行模拟转换，查看脚本的运行结果；调试完毕后用户可以将脚本提交到运行环境，在*设备*上下行时平台会自动进行调用。

要使透传*设备*能够成功接入到物联网平台并进行通信，我们编写的脚本需要支持两个方法：

* *Alink* JSON格式数据转二进制数据方法

此方法用于解析*服务*端向*设备*下发的*Alink* JSON格式数据，将其转换为二进制数据后下发给*设备*，方法中的具体转换代码需要用户自行开发。

|  |
| --- |
| function protocolToRawData(jsonObj){  return rawdata;  } |

* 二进制数据转*Alink* JSON格式数据方法

此方法用于解析*设备*上报到*服务*端的二进制数据，将其转换为*Alink* JSON格式的数据，方法中的具体转换代码需要用户自行开发。

|  |
| --- |
| function rawDataToProtocol(rawData){  return jsonObj;  } |

当包含这两个函数方法的脚本被提交到运行环境中后，*设备*上下行消息时平台便会默认自动调用脚本中的这两个函数，根据用户开发的具体代码内容实现数据解析和转换。

**物联网平台侧开发**

**新建产品和设备，定义功能**

在本实验中，我们仍然希望实现开发板的“温度”和“LED灯闪烁频率”这两个*属性*值的获取和设置。与上一次课程不同的是，在本次实验中，开发板将采用二进制透传的形式发送和接收*属性*值，由云端的脚本负责将其解析为*Alink* JSON格式。

登录物联网平台控制台https://iot.console.aliyun.com，点击“*产品*管理”栏，在“华东2（上海）”区域下新建一个高级版*产品*“透传”，节点类型选择为“*设备*”，数据格式选择为“透传/自定义”, “*设备*类型”选择为“无”。



*产品*创建完成后，在该*产品*下新建“test”*设备*。



在创建好的“透传”*产品*详情页面中，点击“功能定义”栏，与上一次课程中学习的步骤相同，点击右侧的“新增”按钮，新增两个“读写”类型的*属性*：温度和LED灯闪烁频率。



**在线编辑脚本**

在创建好的“透传”*产品*的详情页面中，点击“数据解析”栏，我们可以在下方的编辑框中编写*数据解析脚本*。



脚本Demo参考代码如下：

|  |
| --- |
| var COMMAND\_REPORT = 0x00;//帧类型为*设备*上报  var COMMAND\_SET = 0x01; //帧类型为云端下发  var *ALINK*\_PROP\_REPORT\_METHOD = 'thing.event.property.post';  //标准*Alink* JSON格式*Topic*，*设备*上传*属性*数据到云端  var *ALINK*\_PROP\_SET\_METHOD = 'thing.service.property.set';  //标准*Alink* JSON格式*Topic*，云端下发*属性*控制指令到*设备*端  //解析*设备*端发送的二进制数据，并转换为*Alink* JSON格式数据返回  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];  // command，unit8Array[0]表示0x后的第1个字节，为帧类型字段  if (fHead == COMMAND\_REPORT) {  jsonMap['method'] = *ALINK*\_PROP\_REPORT\_METHOD;  //*Alink* JSON格式 - *属性*上报*Topic*  jsonMap['version'] = '1.0';  //*Alink* JSON格式 - 协议版本号固定字段  jsonMap['id'] = '' + dataView.getInt32(1);  //*Alink* JSON格式 - 标示该次请求id值，从0x后的第2个字节开始读取4个字节，返回一个32位整数id  var params = {};  params['Temperature'] = dataView.getInt16(5);  //对应*产品属性*中 Temperature，从0x后的第6个字节开始读取2个字节，返回一个16位整数Temperature  params['Frequency'] = dataView.getInt16(7);  //对应*产品属性*中 Frequency，从0x后的第8个字节开始读取2个字节，返回一个16位整数Frequency  jsonMap['params'] = params; //*Alink* JSON格式 - params标准字段  }  return jsonMap;  }  //解析*服务*端发送的*Alink*Json格式数据，并转换为二进制数据返回  function protocolToRawData(json) {  var method = json['method'];  var id = json['id'];  var version = json['version'];  var payloadArray = [];  if (method == *ALINK*\_PROP\_SET\_METHOD) // *属性*设置  {  var params = json['params'];  var Temperature = params['Temperature'];  var Frequency = params['Frequency'];  //按照自定义协议格式拼接 rawdata  payloadArray = payloadArray.concat(buffer\_uint8(COMMAND\_SET));  // command字段  payloadArray = payloadArray.concat(buffer\_int32(parseInt(id)));  // *Alink* JSON格式 'id'  payloadArray = payloadArray.concat(buffer\_int16(Temperature));  // *属性*'Temperature'的值  payloadArray = payloadArray.concat(buffer\_int16(Frequency));  // *属性*'Frequency'的值  }  return payloadArray;  }  //以下是部分辅助函数  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);  } |

与脚本解析代码相对应，*设备*发送的第1个字节应为“帧类型”字段，第2个字节开始为int32类型的“id”字段，第6个字节开始为int16类型的“Temperature”*属性*值，第8个字节开始为int16类型的“Frequency”*属性*值。因此，本次实验中*设备*与平台采用二进制格式进行通信的串口协议格式为：

|  |  |  |  |
| --- | --- | --- | --- |
| 帧类型 | ID | Temperature | Frequency |
| 1字节  0-上报；1-下发 | 4字节  请求序号 | 2字节  温度*属性*值 | 2字节  LED灯闪烁频率  *属性*值 |

脚本编辑完成后，点击页面底部的“保存草稿”按钮，系统将保存本次编辑的结果。每次保存草稿会覆盖上一次保存的草稿，草稿不会进入到脚本解析的运行环境中，不影响已经提交的正式脚本。

**模拟运行脚本**

脚本编辑完成后，我们可以通过模拟输入数据，查看脚本的输出结果来验证脚本的运行情况。

* 模拟解析上行数据

在“模拟输入”栏中，选择“模拟类型”为“*设备*上报数据”，模拟*设备*向平台上报数据的场景。输入待透传的二进制数据，点击“运行”，此时系统会调用该脚本模拟进行数据解析，将二进制透传数据按照脚本规则转换为JSON格式数据，并将解析的结果显示在右侧“运行结果”区域中。



若脚本存在错误，页面会提示运行不通过，此时请重新修改脚本代码。

当传入参数为“0x0000223344001e0001”时，解析后的输出结果为：

|  |
| --- |
| {  "method": "thing.event.property.post",  "id": "2241348",  "params": {  "Temperature": 30,  "Frequency": 1  },  "version": "1.0"  } |

* 模拟解析下行数据

选择“模拟类型”为“*设备*接收数据”，模拟平台向*设备*下发“设置”指令的场景。输入JSON格式的数据，点击“运行”，此时系统会调用该脚本模拟进行数据解析，将JSON数据按照脚本规则转换为二进制数据，并将解析的结果显示在右侧“运行结果”区域中。



当传入参数为：

|  |
| --- |
| {  "method": "thing.service.property.set",  "id": "2241348",  "params": {  "Temperature": 33,  "Frequency": 5  },  "version": "1.0"  } |

脚本解析后的输出结果为：“0x010022334400210005”。

脚本运行通过后，我们可以将脚本提交到运行环境中，*设备*上下行时将自动调用脚本，实现数据的解析和转换。



**设备端开发**

在本次课程—脚本解析实验中，我们采用的硬件连接配置和LP应用开发教程4（TSL物模型实验相同）。均是使用AIoTKIT开发板通过mk3080 Wi-Fi模组接入网络。

**新建项目**

在LP应用开发教程2中，我们已经下载了所需要的例程代码。打开VSCode，点击“文件->打开文件夹”，打开已经下载下来的代码，并将当前分支切换到developer分支。点击工程界面左下角，选择此次的例程为passthroughapp例程，开发板选择为AIoTKIT开发板。

**主要代码讲解**

在本节课的实验中，我们每隔3S钟定时采集环境温度信息，之后将环境温度信息和本地的LED闪烁频率按照平台端设置的透传数据格式共同发送到云端。并且我们会订阅透传消息的下行*TOPIC*，用于解析平台端的下发指令，从而进行*设备*端的操作。在本次场景中，为了测试透传的功能，我们将平台下行的数据限制为仅仅可以修改LED灯的闪烁频率*属性*，通过下次的*属性*上报即可查看LED闪烁频率是否修改成功。为了开发方便，在本次实验中，仅仅设置了*属性*的定义，关于*服务*和*事件*的功能并没有涉及。

应用于透传实验的passthroughapp和物模型实验中的tls\_aiotkit\_sample除了订阅和发布的*TOPIC*、上下行消息的格式有区别以外，其余的关于环境温度数据采集，建立MQTT连接等都是相同的，所以在本次代码讲解中，相关重复部分代码我们将仅仅列出其函数声明和功能部分，函数主体内容将不再涉及，详情可以参考LP应用教程4物模型 的实验中。

本节中，passthroughapp例程代码中passthroughapp.c是通过AT联网指令联网并上传与接收数据。我们将详细介绍passthroughapp.c文件，即AIoTKIT开发板和外接mk3080模组进行联网并通过透传格式上传温湿度数据的例程。

1、int application\_start(int argc, char \*argv[])

开发者真正的应用入口函数。在本函数中完成的主要功能为

* AT指令初始化，SAL框架初始化；
* 设置输出的LOG 等级 aos\_set\_log\_level(AOS\_LL\_DEBUG)；
* AliOS Things定义了一系列系统*事件*，程序可以通过aos\_register\_event\_filter()注册*事件*监听函数，进行相应的处理，比如 Wifi*事件*；
* 在配网过程中，netmgr负责定义和注册Wifi回调函数netmgr\_init()。
* 通过调用aos\_loop\_run()进入*事件*循环。

|  |
| --- |
| int application\_start(int argc, char \*argv[])  {  netmgr\_ap\_config\_t apconfig;  #if AOS\_ATCMD //AT指令初始化  at.set\_mode(ASYN);  at.init(AT\_RECV\_PREFIX, AT\_RECV\_SUCCESS\_POSTFIX,  AT\_RECV\_FAIL\_POSTFIX, AT\_SEND\_DELIMITER, 1000);  #endif  #ifdef WITH\_SAL //SAL网络框架初始化  sal\_init();  #endif  printf("== Build on: %s %s ===\n", \_\_DATE\_\_, \_\_TIME\_\_);  aos\_set\_log\_level(AOS\_LL\_DEBUG);  sensor\_all\_open(); //打开外部传感器  aos\_register\_event\_filter(EV\_WIFI, wifi\_service\_event, NULL);  netmgr\_init(); //定义与注册wifi回调函数  netmgr\_start(false);  aos\_cli\_register\_command(&mqttcmd);  //省略部分代码  //每100ms开启定时任务app\_delayed\_action  aos\_post\_delayed\_action(100, app\_delayed\_action, NULL);  aos\_loop\_run();  return 0;  } |

图 15 应用入口函数

2、static void wifi\_service\_event(input\_event\_t \*event, void \*priv\_data)

Wifi*事件*处理函数，当有Wifi*事件*发生时运行该函数。在该函数中完成的主要功能为进行Wifi*事件*的判断，包括*事件*类型的确认等，在确认无误后调用mqtt\_client\_example（）。

3、 int mqtt\_client\_example(void)

mqtt\_client\_example()函数是本次mqtt例程中的鉴权连接函数，该函数所实现的主要功能是：

* 获取*设备*进行鉴权注册时的相关参数。
* 通过Wifi连接IoT平台，进行*设备*注册。

4、static void mqtt\_service\_event(input\_event\_t \*event, void \*priv\_data)

该函数为MQTT例程中*事件*触发后调用的函数，其主要功能为进行*事件*合法性检查，以及调用mqtt\_publish()主函数。

5、static void mqtt\_publish(void \*pclient)

void mqtt\_publish(void \*pclient)函数是本次passthroughapp例程中的mqtt上云发送数据函数，该函数所实现的主要功能是：

* 订阅相关*TOPIC*，接收并解析云端下发的指令。
* 将获取的环境温度数据和LED闪烁频率信息拼接为指定的透传消息格式。
* 定时3S循环发送准备好的透传消息到指定的*TOPIC*。

|  |
| --- |
| static void mqtt\_publish(void \*pclient){  if (is\_subscribed == 0) {  /\*订阅*属性*设置*TOPIC*。平台可以通过该*TOPIC*下发设置LED闪烁频率。并定义其*TOPIC*的回调函数为handle\_raw\_set，回调函数具体内容请参考下文\*/  /\* Subscribe the specific *Topic* \*/  rc = IOT\_MQTT\_Subscribe(pclient, RAW\_*TOPIC*\_PROP\_DOWN,  IOTX\_MQTT\_QOS0, handle\_raw\_set, NULL);  if (rc < 0) {  // IOT\_MQTT\_Destroy(&pclient);  LOG("IOT\_MQTT\_Subscribe() failed, rc = %d", rc);  } is\_subscribed = 1;  } else {  /\* Initialize *Topic* information \*/  memset(&*Topic*\_msg, 0x0, sizeof(iotx\_mqtt\_*Topic*\_info\_t));  *Topic*\_msg.qos = IOTX\_MQTT\_QOS0;  *Topic*\_msg.retain = 0;  *Topic*\_msg.dup = 0;  memset(param, 0, sizeof(param));  memset(msg\_pub, 0, sizeof(msg\_pub));  get\_temp\_data(&temp\_data, &temp\_timestamp); //每隔3S获取温度数据信息  int frequency = get\_led\_fre(); //获取本地的LED闪烁频率  int temp = temp\_data/100;  int frequency\_h = frequency/256;  int frequency\_l = frequency%256;  int temp\_h = temp/256;  int temp\_l = temp%256;  unsigned char p\_buf[10]={0}; //按照指定格式打包数据  p\_buf[0]=0;  p\_buf[1]=cnt/1000;  p\_buf[2]=(cnt/100)%10;  p\_buf[3]=(cnt/10)%10;  p\_buf[4]=cnt%10;  p\_buf[5]=temp\_h;  p\_buf[6]=temp\_l;  p\_buf[7]=frequency\_h;  p\_buf[8]=frequency\_l;  *Topic*\_msg.payload = (void \*)p\_buf;  *Topic*\_msg.payload\_len = 9;  //以透传格式上报数据到/thing/model/up\_raw *TOPIC*  rc = IOT\_MQTT\_Publish(pclient, RAW\_*TOPIC*\_PROP\_UP, &*Topic*\_msg);  if (rc < 0) LOG("error occur when publish");  }  if (++cnt < 20000) {  /\*每3s重新运行mqtt\_publish()函数\*/  aos\_post\_delayed\_action(3000, mqtt\_publish, pclient);  } else {  IOT\_MQTT\_Unsubscribe(pclient, *ALINK*\_*TOPIC*\_PROP\_POSTRSP);  aos\_msleep(200);  IOT\_MQTT\_Destroy(&pclient);  release\_buff();  is\_subscribed = 0;  cnt = 0;  }  } |

图 19 passthroughapp例程数据上云主函数

1. static void handle\_raw\_set (void \*pcontext, void \*pclient, iotx\_mqtt\_event\_msg\_pt msg)

该函数为passthroughapp例程中云端设置*属性*参数的*TOPIC*回调函数，云端可以通过/thing/model/down\_raw *TOPIC*实现LED闪烁频率*属性*的设置。

|  |
| --- |
| /\*  \* MQTT Subscribe handler  \* *Topic*: RAW\_*TOPIC*\_PROP\_DOWN  \*/  static void handle\_raw\_set(void \*pcontext, void \*pclient, iotx\_mqtt\_event\_msg\_pt msg)  {  iotx\_mqtt\_*Topic*\_info\_pt p*Topic*\_info = (iotx\_mqtt\_*Topic*\_info\_pt)msg->msg;  unsigned char receive\_buf[30]={0};  for(int i=0;i<p*Topic*\_info->payload\_len;i++)  { //获取云端下发的消息指令  receive\_buf[i] = ((unsigned char \*)p*Topic*\_info->payload)[i];  }  //按照指定格式解析云端消息  int rec\_temp = ((int)receive\_buf[5])\*10+(int)receive\_buf[6];  int rec\_fre = ((int)receive\_buf[7])\*10+(int)receive\_buf[8];  if(rec\_fre>0)  {  set\_led\_fre(rec\_fre); //设置本地LED闪烁频率  }  } |

图 18 /thing/model/down\_raw *TOPIC*回调函数

**修改设备相关参数**

在上一小节中，我们打开了passthroughapp@AIoTKIT工程。在本小节中，我们需要修改官方passthroughapp项目工程，添加我们在阿里云物联网控制台上新建*设备*的三元组信息。具体流程如下：

demo程序所在路径是/example/passthroughapp,在此次例程当中，我们需要将*设备*的三元组信息修改为新注册的*设备*三元组信息。具体修改文件为：example/ passthroughapp /passthroughapp.c。PRODUCT\_KEY、DEVICE\_NAME、 DEVICE\_SECRET这三个参数是保证*设备*和IOT平台间可靠通信的唯一标识，所以这三个参数必须保证和建立*设备*时的信息相同，*TOPIC*信息也要保证和平台端的*TOPIC*保持一致，因为*设备*在发送与接收消息时都要带有*TOPIC*信息，不一致的话可能会导致数据通信发生错误。具体参数修改如下图所示：



图 42 修改*设备*三元组信息

**编译下载**

工程编译与下载的方式和LP应用开发教程2中相同，详情可参考开发教程2中的*设备*开发编译下载章节，在此处不再赘述。

至此，工程代码烧录进入开发板当中。打开串口助手即可查看程序的运行输出信息，

**在线调试**

通过复位开发板使其运行烧录进去的程序，接入物联网平台。虽然*设备*直接上报和接收的是二进制的透传数据，但是经过脚本的解析，平台侧查看到的仍然是*Alink* JSON格式的数据，与上一次课程中平台侧的数据格式相同。我们仍然可以使用平台提供的“在线调试”功能来查看*设备*上报的*属性*值，或者向*设备*下发命令设置*属性*值。

我们进入*产品*详情页面的“在线调试”栏，选择“调试*设备*”为我们创建的test*设备*，当*设备*上线后，我们可以查看到*设备*主动上报的*属性*值。



Figure 2 云端查看*设备*上报消息

此时查看*设备*端日志，可以发现*设备*上报的是十六进制的数据流，说明我们编写的解析脚本发挥了作用，成功将二进制转换为了JSON格式的数据。



Figure 3 *设备*端上报的消息内容日志

同时，我们还可以对“LED灯闪烁频率”（对应开发板上的LED2）执行“设置”操作。解析脚本会将JSON格式的指令转换为二进制格式的数据下发给*设备*。



Figure 4 *设备*端接收到的消息指令日志

*设备*收到指令后会更改自身*属性*状态并上报最新状态。



Figure 5 云端查看修改后的*设备*上报信息

当我们将LED灯闪烁频率设置为8以后，*设备*主动上报的*属性*内容中Frequency字段也变为了8，同时开发板上的LED灯闪烁频率明显加快。

**课程小结**

本次课程学习了物联网平台*设备*管理组件中的脚本解析功能，并将其与在线调试功能相结合，通过在云端编写简单的*数据解析脚本*，实现透传*设备*到物联网平台的接入与通信。通过本次课程与上一次课程的学习，不管是采用*Alink* JSON格式通信的*设备*，还是采用透传进行通信的*设备*，我们都可以将其顺利地接入物联网平台，进行后续开发了。