浙江大学
Zhejiang University

第八章：物联网云-端一体开发

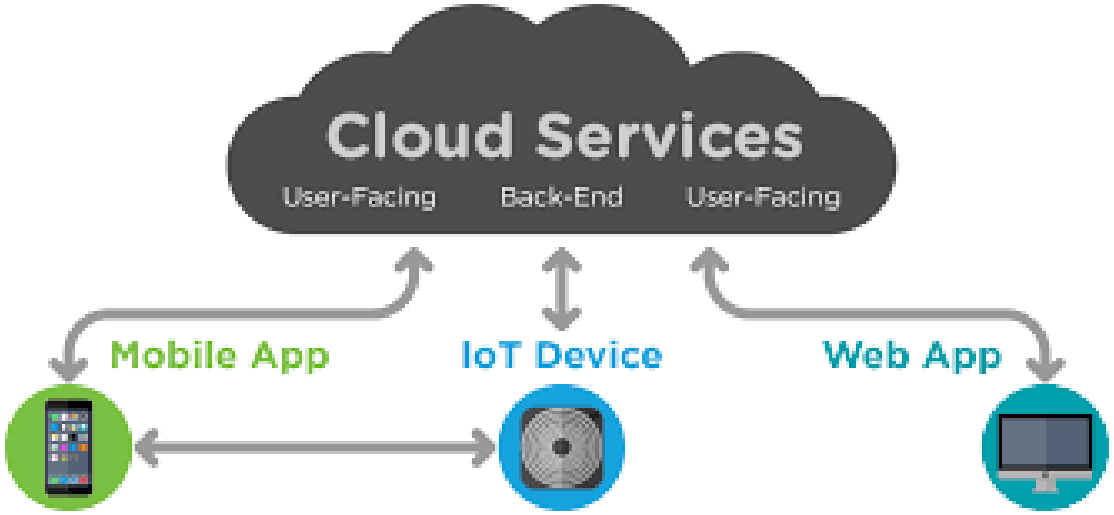
董 玮

浙江大学

目录

- 概述
- 云-端一体开发平台主要功能
- 典型物联网云-端一体开发平台
  - TinyLink 2.0
  - 三星 SmartThings
  - Link Develop
- 总结

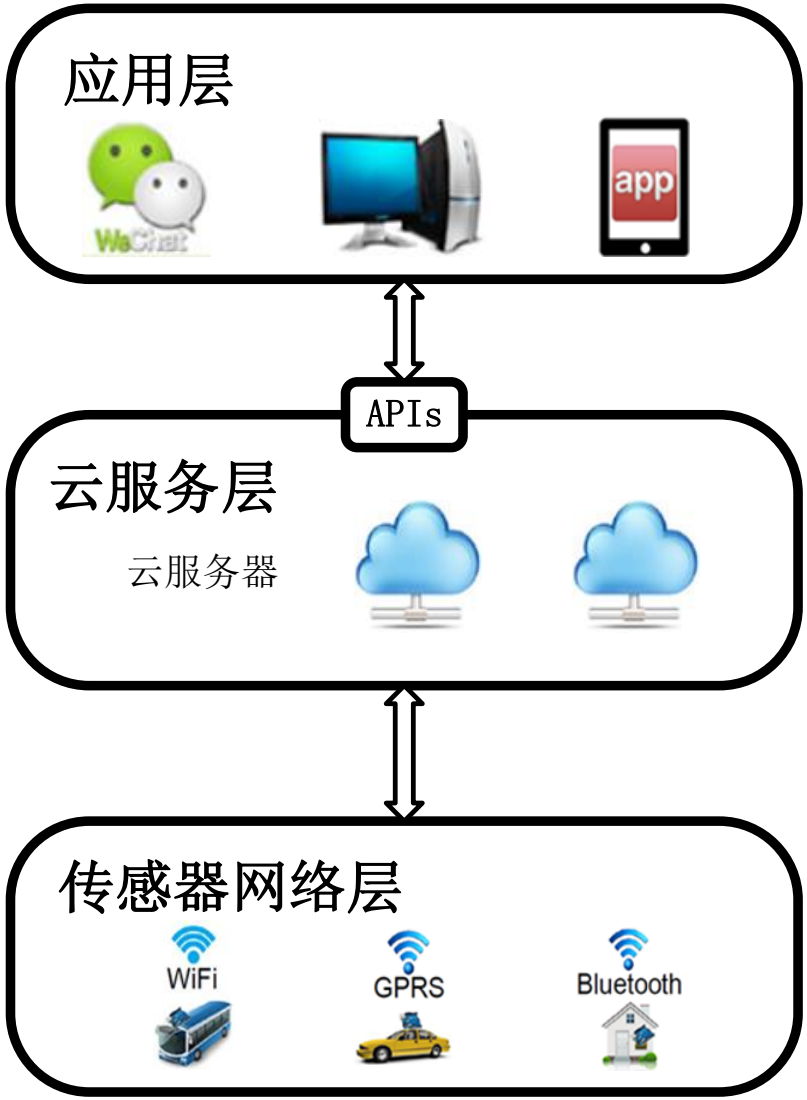
概述



物联网技术栈复杂，一个完整的物联网应用涉及嵌入式开发、网络协议、云平台开发、Web开发、移动开发等多个领域

Mosaic

- 空气质量监测应用
- 涉及嵌入式开发，WiFi，蓝牙，云端服务开发，Web应用开发，移动应用开发等多种技术

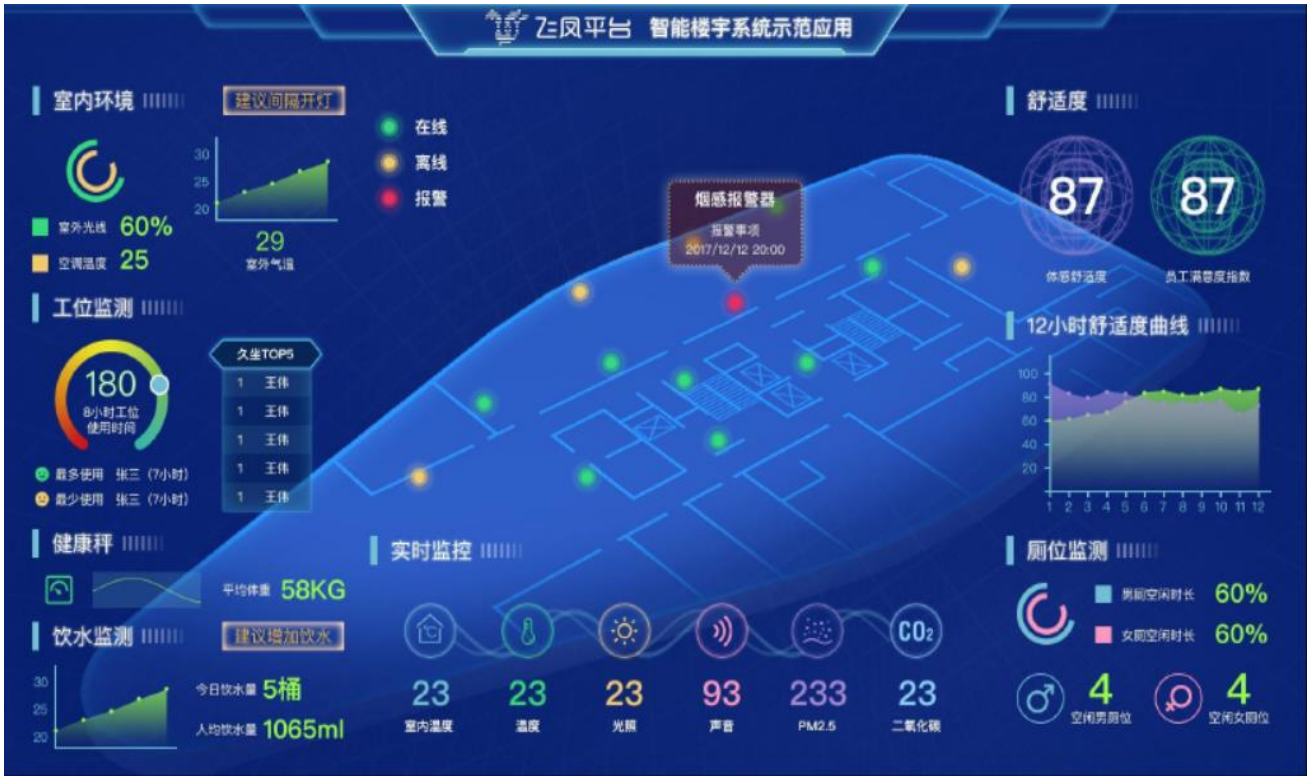


共享单车

- 设备端：GPS、蓝牙、NB-IoT……
- 服务端：微软Azure（摩拜）……
- 移动应用：APP、小程序……

智能楼宇

- 设备端：多种异构的嵌入式设备，SDS018、DHT11……
- Web应用：复杂的数据展示页面，折线图、文本框、仪表盘……



概述

- 为解决上述问题，各大厂商推出云-端一体开发平台，如三星SmartThings、阿里云Link Develop等，旨在通过云-端一体开发平台涵盖物联网应用的整个开发流程，解决物联网开发领域存在的应用开发碎片化、标准不统一、技术栈复杂等问题

SmartThings

Link Develop

机智云 Gizwits

OneNET

目录

- 概述
- 云-端一体开发平台主要功能
- 典型物联网云-端一体开发平台
  - TinyLink 2.0
  - 三星 SmartThings
  - LinkDevelop
- 总结与展望

物联网云-端一体开发主要功能

我们认为，理想的物联网云-端一体开发应该包括如下功能：

- 支持设备端开发
- 支持云端服务开发
- 支持Web应用/移动应用开发
- 简化三端集成开发流程，优化整体应用的性能

设备端开发-现状与问题

- 物联网设备种类繁多，不同种类的设备需要不同的开发语言、开发环境
  - 开发语言：C/C++、Python、JavaScript……
  - IDE：Keil，IAR，ARDUINO……

IAR Embedded Workbench

云-端一体的设备端开发

- 基于本地编译的设备端开发
  - 基于设备模型，通过SDK等方式，提供统一的设备操作与平台接入接口
  - 需要开发者在本地配置开发环境
- 基于云端编译的设备端开发
  - 代码编译过程放在云端，本地无需配置开发环境
- 基于解释型语言的设备开发
  - 使用JavaScript等解释型语言，代码无需编译
  - 设备端需要安装解释器

基于本地编译的设备端开发

- 基于设备模型的设备端开发
  - 设备模型定义设备的功能，是设备与云平台交互的桥梁
  - 设备端开发者需要实现设备模型所定义的设备功能
- 基于SDK的设备端开发
  - 在设备模型的基础上，基于SDK提供的设备开发API，进一步简化设备开发
- 基于物联网操作系统的设备端开发
  - 在设备模型的基础上，使用物联网操作系统，提供统一的API供开发者调用，屏蔽硬件差异

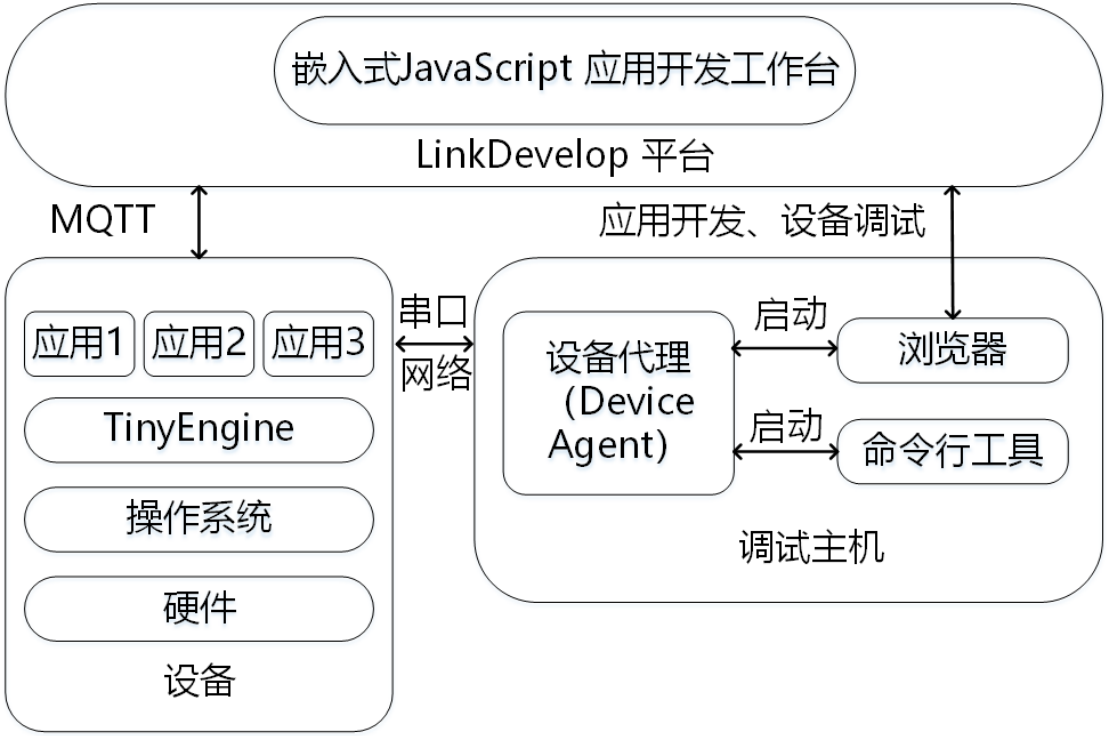
基于云端编译的设备端开发

- 代码可在本地或云端编写，在云端实现代码的编译

- 开发者无需在本地搭建编译环境，简化了开发流程

基于解释型语言的设备端开发

- 与基于编译型语言的设备端开发的不同
  - 跨平台容易，代码可移植性强
  - 需要在设备端运行解释器
- Link Develop 在线JavaScript开发工作台



平台对比

一体开发平台	设备端开发		
	本地编译	云端编译	解释型语言
TinyLink 2.0	×	云端编译+自顶向下的开发方式	×
三星 SmartThings	基于设备模型的开发	×	×
Link Develop	提供C语言SDK，以及基于AliOS Things的开发方式	√	支持基于JavaScript语言的开发

云端服务开发

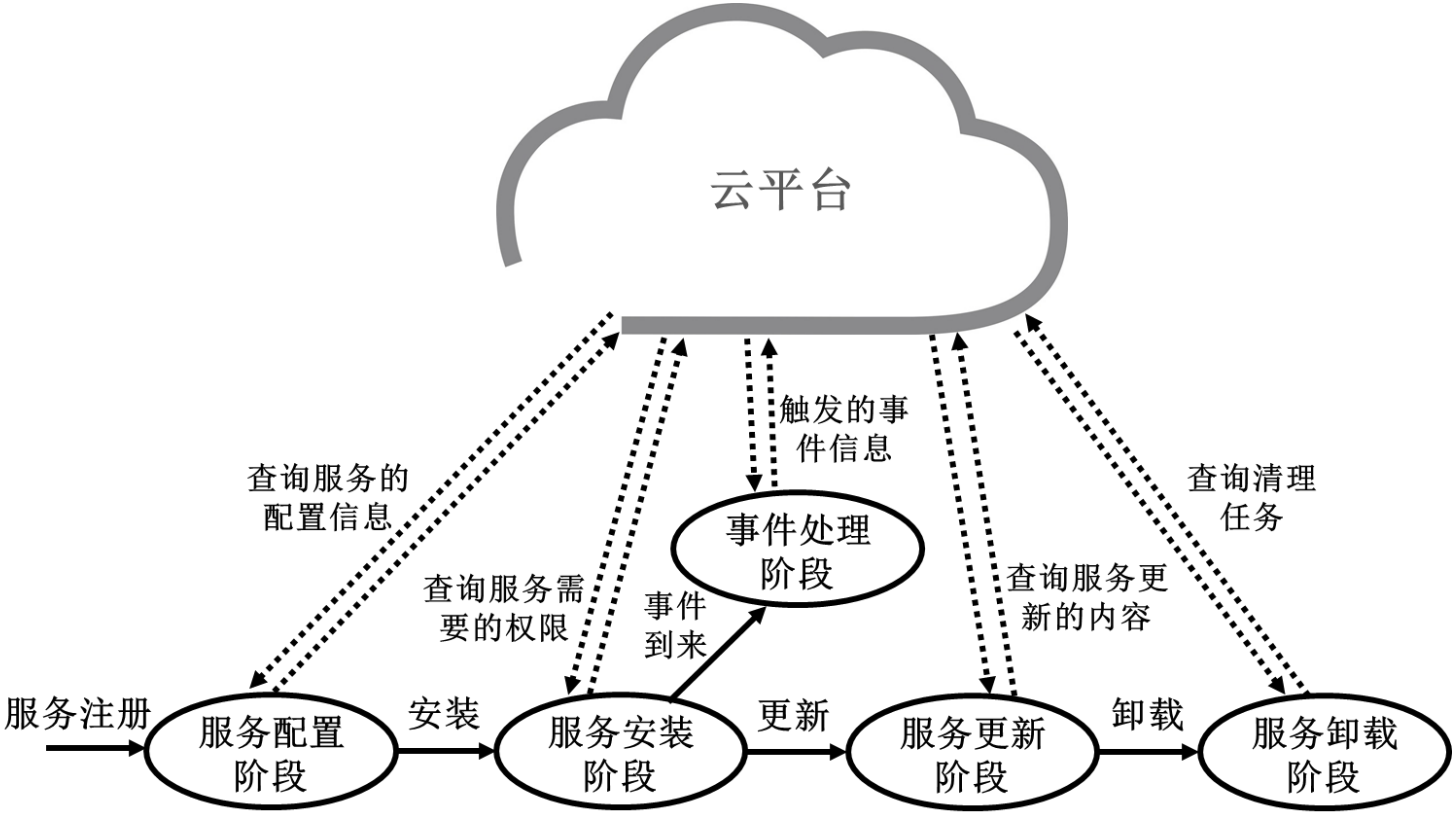
- 物联网云端服务的特点
  - 云端服务处理逻辑包含大量由传感数据触发的规则
  - 服务与设备数据直接交互，设备的异构性导致服务可重用性不高
  - 数据操作相对较为简单，包含大量重复的操作，如数据存储、数据读取等
- 云端服务开发方式
  - 基于IFTTT规则的云端服务开发
  - 基于生命周期的云端服务开发
  - 基于服务编排的云端服务开发

基于IFTTT规则的云端服务开发

- 通过开发工作台，设置IFTTT规则，可实现简单的云端服务
- 示例为使用中国移动OneNet平台，通过IFTTT规则配置，开发一个温度告警服务
  - **接收信息方式：**
    - *邮箱：用于把信息推送给用户*
    - *URL：用于数据的进一步处理*

基于生命周期的云端服务开发

- 服务实现平台规定的完整生命周期。每个生命周期阶段分工明确，每个事件或用户行为会使服务进入不同的生命周期阶段
- 在每个生命周期阶段，平台会向服务发送请求，服务要求能够处理并响应这些请求

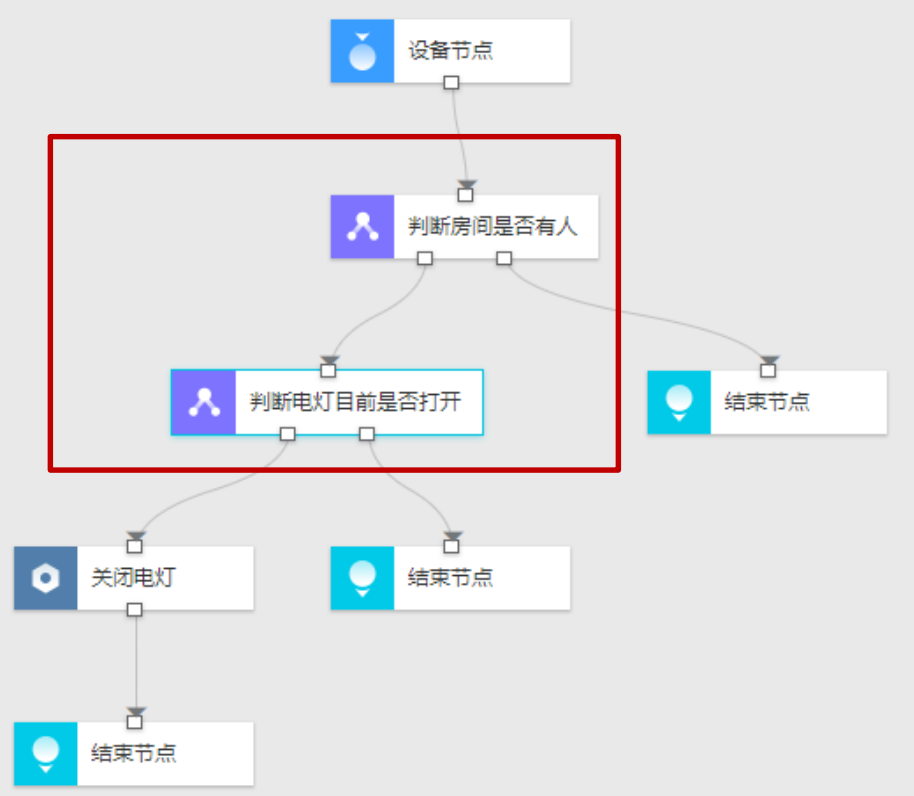


基于服务编排的云端服务开发

- 服务节点：每一个服务节点代表一个基础的API，如数据查询、数据存储等
- 服务编排：组合不同的服务节点，并对节点进行配置，从而实现特定的功能

基于服务编排的云端服务开发

- 智能家居灯：当房间没有人并且电灯打开时，关闭电灯（房间有没有人通过摄像头判断，处理时间较长；电灯是否打开通过当前光照强度判断，处理时间较短）



服务托管

- 传统的服务开发需要开发者自行购买服务器，并将服务部署在自己的服务器上
- 使用无服务计算技术(详细介绍见第七章)，云端服务可直接由云平台托管，开发者无需自己购买、配置、管理云服务器
- 典型的无服务计算技术如AWS Lambda、阿里云函数计算等

平台对比

一体开发平台	云端服务开发			
	基于IFTTT规则	基于生命周期	基于服务编排	服务托管
TinyLink 2.0	√	×	×	√
三星 SmartThings	×	√	×	√
Link Develop	×	×	√	√

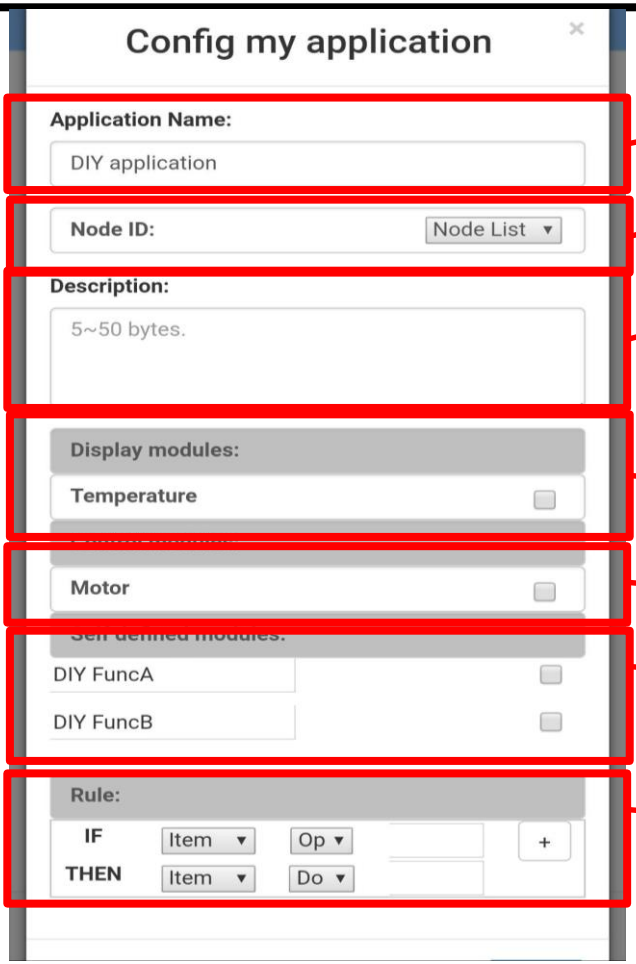
Web应用/移动应用开发

- 物联网Web应用/移动应用特点
  - 应用UI与设备端所提供的功能密切相关
  - 应用组件的可重用性高（开关、文本框、折线图、仪表盘等)
- Web应用/移动应用开发方式
  - 基于设备端代码的应用框架自动生成
  - 基于可视化拖拽的Web应用/移动应用开发
  - 基于插件的Web应用/移动应用开发

基于设备端代码的应用框架自动生成

- 基于设备端应用代码，自动生成Web应用/移动应用框架，用户只需要简单的配置即可完成Web应用/移动应用开发。支持这种开发方式的有TinyLink 2.0、中移OneNet等

```
void mesgArrived(MQTT::MessageData& md){
   MQTT::Message &message = md.message;
 /*根据MQTT下行信息响应*/
   switch(message)
     case 'MOTOR_TURNON':
        TL_Motor.turnOn();
     ......
}//主循环周期读取环境温度数据上传
void loop() {
   TL_Temperature.read();
   String data = "{";
   data += ",\"Temperature\":";
   data += TL_Temperature.data();
   data += "}";
   char buf[100];
   data.toCharArray( buf,100 );
   ......}
```



应用名称

关联节点ID

应用描述

数据展示模块

控制模块

自定义模块

IFTTT规则配置

基于可视化拖拽的Web/移动应用开发

- 采用可视化组件，开发者只需拖拽相关组件并进行简单配置，即可完成开发
- 示例为使用Link Develop平台开发一个简单的温度数据展示应用

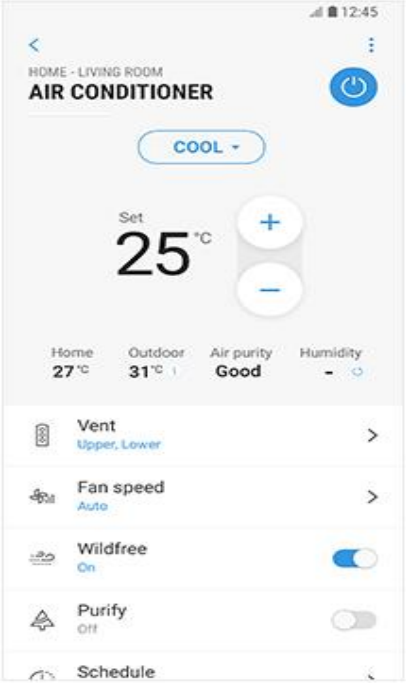
基于插件的Web/移动应用开发

- 插件：使用HTML、CSS、JavaScript和JSON来定义设备用户界面和行为的网页应用程序
- 开发者可自行开发插件并上传到平台，然后在官方app上按需加载插件
- 示例为SmartThings app设备插件界面和其中一个空调插件界面

点击ADD DEVICE可以添加新的设备（插件）

App中嵌入的插件

用户在官方app中下载的插件



平台对比

一体开发平台	Web应用/移动应用开发			
	基于设备端代码的应用框架自动生成	基于可视化拖拽	基于插件	传统的Web应用/移动应用开发
TinyLink 2.0	√	×	×	×
三星 SmartThings	×	×	√	√
Link Develop	×	√	×	√

三端应用集成简化

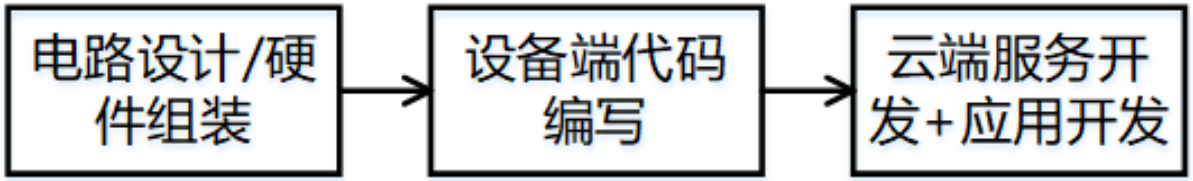
- 现状与问题
  - 很多平台可以提供对三端应用开发的支持
  - 三端应用的集成还较为复杂，如设备端与云端服务交互的数据格式不统一，导致设备端数据无法被云端服务直接解析
- 三端应用集成简化
  - 设备模型
  - 命名机制

三端应用集成简化

- 设备模型
  - 设备的抽象描述，规定设备与云端数据交互规范
  - 解耦设备开发与云端服务开发
- 命名机制
  - 物理设备与虚拟ID的映射

设备模型

- 传统的物联网开发方式
  - 应用开发依赖硬件设备，开发流程无法同步进行

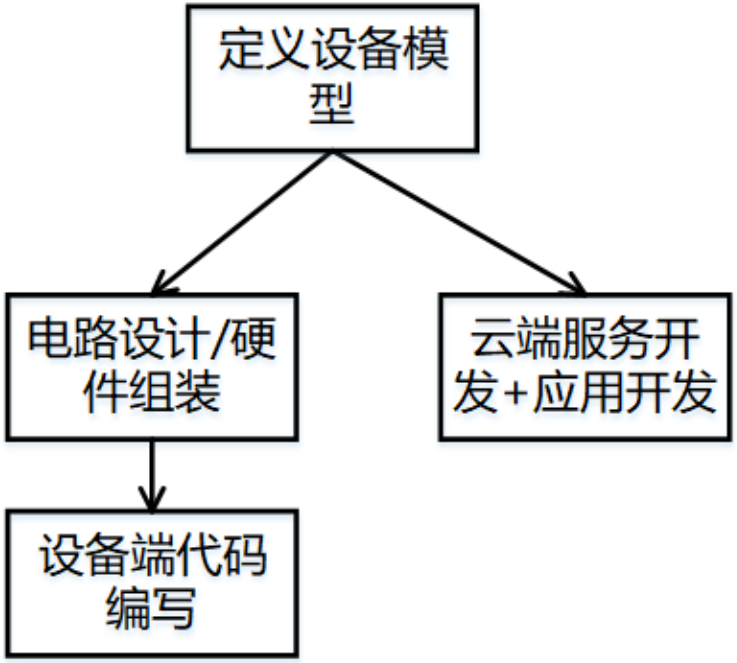


- 设备模型
  - 设备所能提供功能的描述
  - 设备开发和应用开发的桥梁，基于设备模型，应用开发与设备开发可完全分离

基于设备模型的开发流程

示例：智能灯泡

- **步骤1：设备模型定义**
  - *包括开、关两个功能，可通过云平台进行控制*
  - *包含光照、声音强度两个属性*
  - *包含一个属性上报事件*
- **步骤2：应用开发**
  - *硬件开发人员设计电路，编写设备端代码，实现开灯、关灯两个功能；软件开发人员基于设备模型定义，使用虚拟设备模拟真实设备，开发Web应用/移动应用*



设备模型: SmartThings VS Link Develop

- 设备档案（三星 SmartThings）
  - 使用JSON语言描述
  - 设备>组件>功能>（属性，命令）
  - 设备所包含的功能必须是平台已有的
- 物模型（Link Develop）
  - 使用JSON语言描述
  - 将设备功能划分为“属性”、“事件”、“服务”三类
  - 设备功能可使用平台提供的标准功能，也可由开发者自行定义

设备模型: SmartThings VS Link Develop

智能灯泡

- 包含光照、声音强度两个属性
- 包括开、关两个功能，可通过云平台进行控制
- 包含一个属性上报事件，可把两个属性数据上报至云平台

设备模型: SmartThings VS Link Develop

❑ 智能灯泡设备模型定义-三星 SmartThings

```
{
  "name": “smart-bulb",
  “components”: [ { // 该灯泡包含一个组件
      "id": "main",
      // 该灯泡包含的二个功能
      “capabilities”: [// 每个id对应具体的功能定义
        { "id": “switch"},
        { "id": “soundSensor"}
      ]
    }
  ],
  "metadata": {
    "dthName": “Smart Bulb",
    "dthNamespace": "smartthings"
  }
}
```

```
id: switch, //ID由官方提供，共73种，与功能对应
name: Switch
attributes: //定义属性
  switch:
    schema:
      type: object
      properties:
        type: string
        enum:          //属性数据类型：数字，字符串
          - 'off’      // 枚举（ENUM ）等
          - ‘on’       //开关有”on”和”off”两种状态
      required: ["value"]
commands: // 定义命令
  ‘off’:                // off 命令
    arguments:[]        // 命令参数
  ‘on’:                 // on 命令
    arguments:[]        // 命令参数
```

注：平台无Light相关功能定义

设备模型: SmartThings VS Link Develop

❑ 智能灯泡设备模型定义-Link Develop

```
{
  “services”: [   {        // 服务定义
      “outputData”: [],          // 输出参数
      “identifier”: “turnOn”,    // 标识符
       “inputData”: [],           // 输入参数
       “method”: “thing.service.turnOn”,    //
对应方法名
       “name”: “开灯”,            // 名称
       “required”: false,         // 是否必须
       “callType”: “async”    },  // 调用方式(同
步/异步 )
     {
        “identifier”: ”turnOff”
        "method": "thing.service.turnOff",
         ……
      }],
}
```

```
{
   “properties”: [   {      // 属性定义
      “identifier”: “SoundValue“  // 标识符
       dataType“: {        // 数据类型
          “specs”: {        // 数据参数
              “unit”: “dB”,    // 单位
              “min”: “0.0”,     // 最小值
              “unitName”: “分贝”,    // 单位名称
              “max”: “100.0”,     // 最大值
              “step”: “0.1”    // 步长
           },
         “type”: “float”      }, // 类型
        “name”: “声音强度”,       // 属性名称
        “accessMode”: “r”,  // 是否可读写
        “required”: false      // 是否必须
   },
   {    "identifier”: “LightLuxValue“
          ……
   } ]
}
```

设备模型: SmartThings VS Link Develop

❑ 智能灯泡设备模型定义-Link Develop

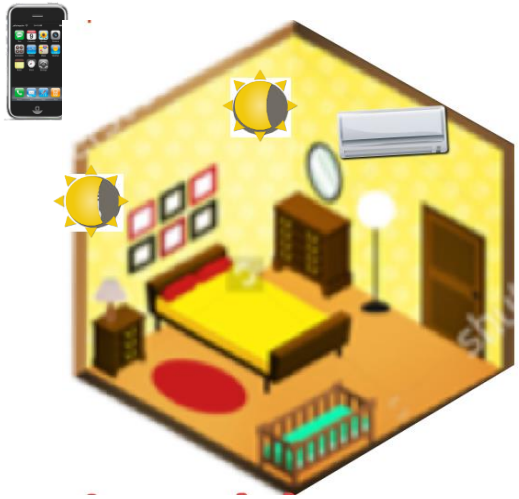
```
"events": [ {              // 事件定义
    "outputData": [    // 输出参数，包含声音强度和光照数据
      {
        "identifier": "SoundValue",  // 声音强度数据
        "dataType": {   // 参数类型
          "specs": {  …….},
          "type": "float"  // 数据类型
        },
        "name": "声音强度"  // 参数名称
      },
      {
        "identifier": "LightLuxValue",   // 光照数据
      }
    ],
    "identifier": "post",        // 事件标识符
    "method": "thing.event.property.post",  // 方法
    "name": "post",              // 事件名称
    "type": "info",              // 事件类型
    "required": true,            // 是否必须
    "desc": "属性上报"           // 事件描述
  }
]
```

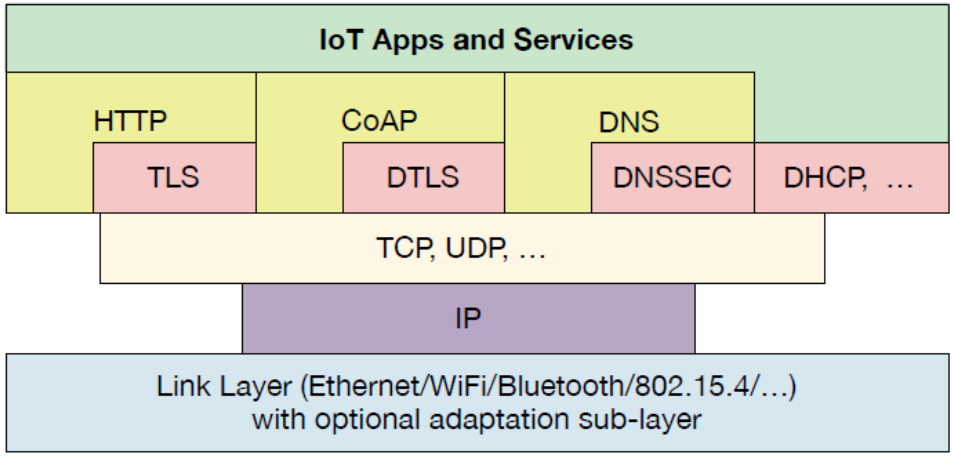
命名机制

- 物联网场景中，内容检索和命令推送是两个基本功能
  - 例如，“查询现在的室内光线强度”，“打开客厅的灯”
- 命名机制是支持这两种操作的基础
  - 需要通过名字找到对应的数据或设备
- 传统基于TCP/IP的命名机制
  - 为设备分配IP地址，建立从设备到IP，再到应用层名称等多层映射关系
  - 问题：关注于建立设备之间的连接，需要实现复杂的协议栈，通过多层映射建立设备之间的连接

命名机制：基于TCP/IP的命名机制

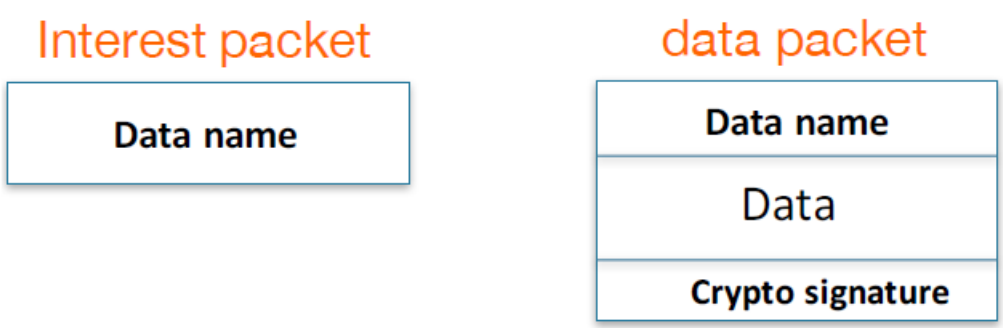
- 一个简单的例子
  - APP: “查询现在的卧室光强” “打开卧室的台灯”
- 操作过程
  - 查询映射：“卧室光强”->光强计 URI
  - 连接到AlexHome.com服务上
  - DNS查询AlexHome.com服务IP地址
  - DHCP为所有设备分配IP地址
  - 建立手机到光强计设备的连接





命名机制：基于NDN的命名机制

- NDN: Name Data Networking
  - 以数据为中心的网络
  - 与位置无关，直接通过名字获取数据
  - 基于两种数据包通信，“兴趣包（Interest）”和“数据包（Data）”，分别用于请求和传送内容
    - 兴趣包：请求的数据名称，或希望执行的操作（作为名称的一部分）
    - 数据包：回复的数据内容，或对操作执行的ACK



命名机制：基于NDN的命名机制

- 分层命名机制
  - 按照约定好的命名规则对数据对象命名
  - 常见的命名规则：/location/fixture-namespace/command/timestamp
    - 例如，/MyHome/Bedroom/Lighting/sensing/Intense/201812311920
- 数据高度可聚合
  - 收集所有光强读数：/MyHome/Bedroom/Lighting/sensing/Intense/

命名机制：基于NDN的命名机制

- 新设备的加入
  - 新设备可以通过兴趣包来告诉配置管理器其设备标识和功能
    - 例如，“/ local / discovery / lighting / serial = 123456”
  - 配置管理器进一步配置其属性，包括ID，NDN名称前缀，功能集合等
    - ID: “/ MyHome/Bedroom / _fixtures / 123456”
    - 关联事物：“/ MyHome/Bedroom / Lighting / TableLamp”
    - 功能集合：“.../action/ON”，“.../action/OFF”

命名机制：基于NDN的命名机制

❑ 一个简单的例子

“打开卧室的台灯”

INTREST(/MyHome/Bedroom/Lighting/TableLamp/action/ON)

DATA(name|ACK|signature)

“查询现在的室内光强”

INTREST(/MyHome/Bedroom/Lighting/Intense)

DATA(name|data|signature)

三端应用集成简化

一体开发平台	三端应用集成简化	
	设备模型	命名机制
TinyLink 2.0	×	基于TCP/IP
三星 SmartThings	设备档案	基于TCP/IP
Link Develop	物模型	基于TCP/IP

市场主流云-端一体开发平台概览

一体化平台	设备端开发			服务开发			应用开发			三端应用集成简化	
	本地编译	云端编译	解释型语言	IFTTT规则配置	服务编排	基于生命周期	可视化拖拽	应用框架自动生成	基于插件	设备模型	命名机制
ARM Mbed	√	√	×	×	×	×	×	×	×	×	√
中移 OneNet	×	×	×	√	×	×	√	×	×	×	√
TinyLink 2.0	×	√	×	√	×	×	×	√	×	×	√
三星 SmartThings	√	×	×	×	×	√	×	×	√	√	√
机智云	√	×	×	√	×	×	×	√	×	√	√
LinkDevelop	√	√	√	×	√	×	√	×	×	√	√

目录

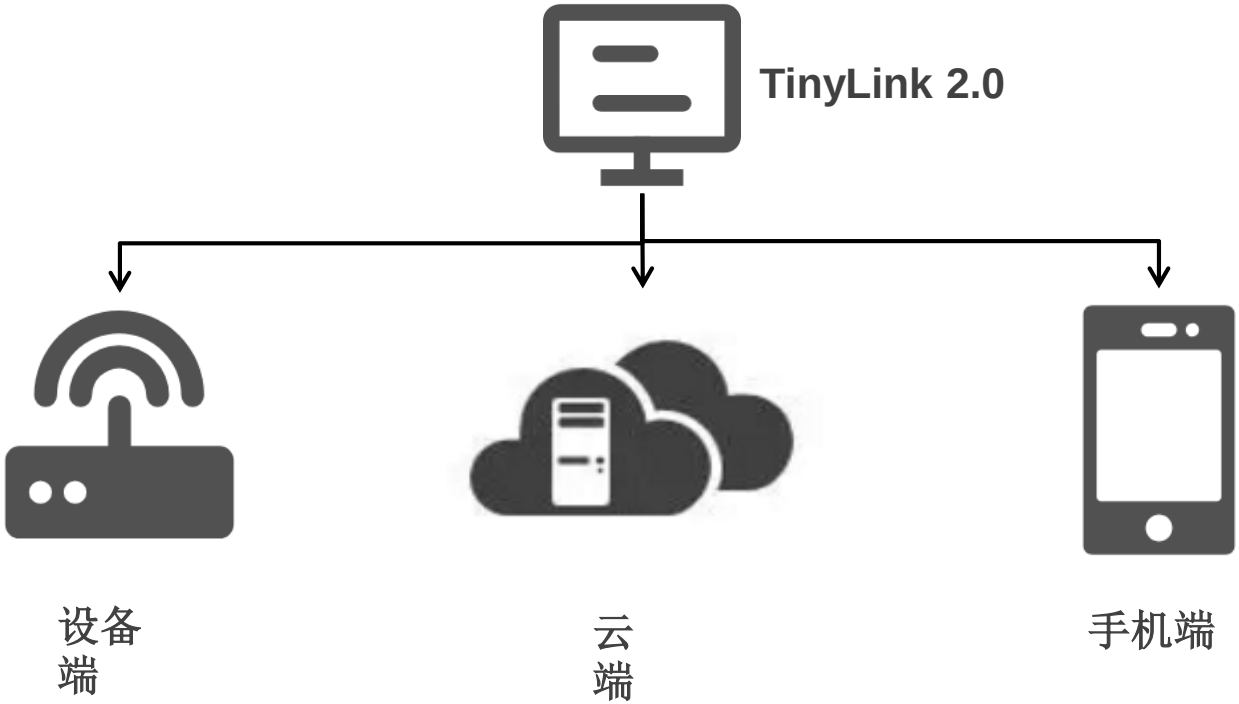
- 概述
- 云-端一体开发平台主要功能
- 典型物联网云-端一体开发平台
  - TinyLink 2.0
  - 三星 SmartThings
  - Link Develop
- 总结

TinyLink 2.0 平台

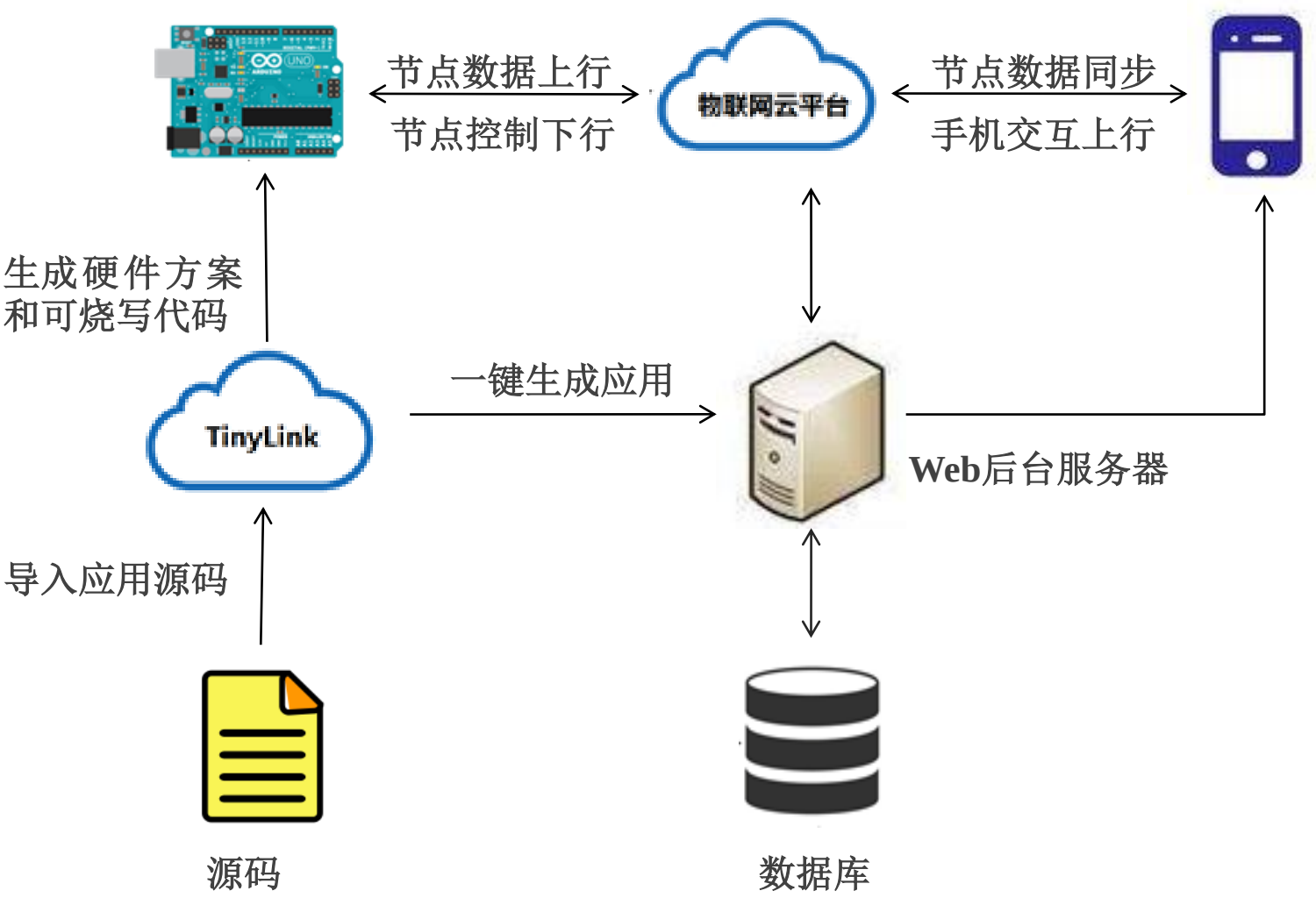
- 概述
- 特点
  - 基于设备端代码的应用框架自动生成
- 开发案例
  - 智能教室座位使用统计系统

TinyLink 2.0 平台概述

- 2017年3月浙大EmNets实验室推出Tinylink 1.0 开发平台（详见第四章相关介绍）
- 2017年12月浙大EmNets实验室推出Tinylink 2.0开发平台，在1.0平台设备端开发的基础上，**集成Web端应用和手机端应用开发**，构成了物联网一体化开发平台。



TinyLink 2.0 平台架构



TinyLink2.0 关键特性

- 设备开发
  - 采用云端编译的开发方式，使用统一的TinyLink语言屏蔽硬件差异
- 云端服务开发
  - 使用基于IFTTT规则的云端服务开发方式，开发简单，适合初学者
- Web应用/移动应用开发
  - 基于设备端代码，自动生成Web应用/移动应用框架
  - 开发者仅需对应用框架进行简单配置即可完成应用开发

TinyLink 2.0 平台

- 概述
- 特点
  - 基于设备端代码的应用框架自动生成
- 开发案例
  - 智能教室座位使用统计系统

TL_Function API 抽象实例

```
#include "TL_Config.h"
#include "MPU9250_RPI.h"
/*设备Grove_IMU9封装的加速度功能*/
class Grove_IMU9_Accelerometer_RPI{
        private:
                MPU9250 accelgyro;
                int status;
                int16_t ax, ay, az;
        public:
                Grove_IMU9_Accelerometer_RPI();
                void init();
                int read();             //读取一次设备寄存器即时数据
                double data_x();        //返回加速度X轴传感器数据
                double data_y();        //返回加速度Y轴传感器数据
                double data_z();        //返回加速度Z轴传感器数据
        };

#if TINYLINK_ACCELEROMETER == GROVE_IMU9
        extern Grove_IMU9_Accelerometer_RPI    TL_Accelerometer;
#endif
#endif
```

实际物理设备（可变）　提供的功能（统一）　开发板（可变）　抽象成加速度API

设备端API的分类

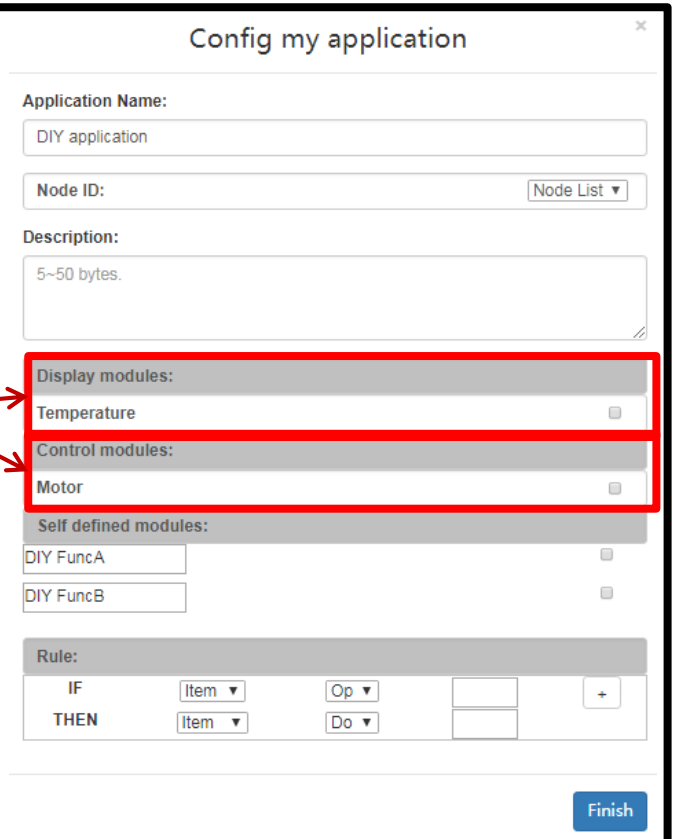
- 基于物理硬件功能的抽象
- 使用C++类封装设备功能
- API归为四类

API类型	功能说明	举例
系统功能（System）	开发板提供的系统功能抽象	存储，定时器，串口，时间
通信（Communication）	通信类硬件功能的抽象	HTTP，MQTT，BLE
感知（Sensing）	感知类硬件数据的读取的抽象	心跳，PM2.5，甲醛等传感器数据的读取
控制（Control ）	可用下行信号控制的硬件的抽象	控制电机的开关甚至转速

利用TL_Function 生成应用框架

- 控制类API
  - 例如TL_Motor生成在Control模块可选
- 感知类API
  - 例如TL_Temperature生成在Display模块可选

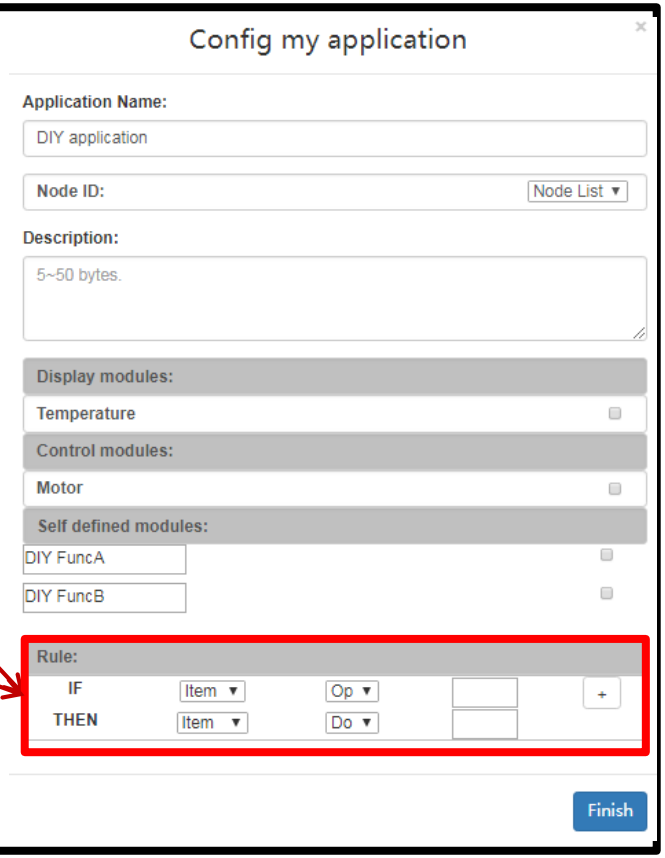
```
void messageArrived(MQTT::MessageData& md){
    MQTT::Message &message = md.message;
 /*根据MQTT下行信息响应*/
    switch(message)
      case 'MOTOR_TURNON': TL_Motor.turnOn();
      ......
}//主循环周期读取环境温度数据上传
void loop() {
    TL_Temperature.read();
    String data = "{";
    data += ",\"Temperature\":";
    data += TL_Temperature.data();
    data += "}";
    char buf[100];
    data.toCharArray( buf,100 );
    ......}
```



利用TL_Function 生成应用框架

- 应用IFTTT配置框架
  - 感知类API可设置为IFTTT条件语句内容
  - 控制类API可设置为IFTTT响应语句内容

```
void messageArrived(MQTT::MessageData& md){
    MQTT::Message &message = md.message;
 /*根据MQTT下行信息响应*/
    switch(message)
      case 'MOTOR_TURNON' TL_Motor.turnOn();
      ......
}//主循环周期读取环境温度数据上传
void loop() {
    TL_Temperature.read();
    String data = "{";
    data += ",\"Temperature\":";
    data += TL_Temperature.data();
    data += "}";
    char buf[100];
    data.toCharArray( buf,100 );
    ......}
```



TinyLink 2.0 平台

- 概述
- 特点
  - 基于设备端代码的应用框架自动生成
- 开发案例
  - 智能教室座位使用统计系统

TinyLink2.0 开发案例

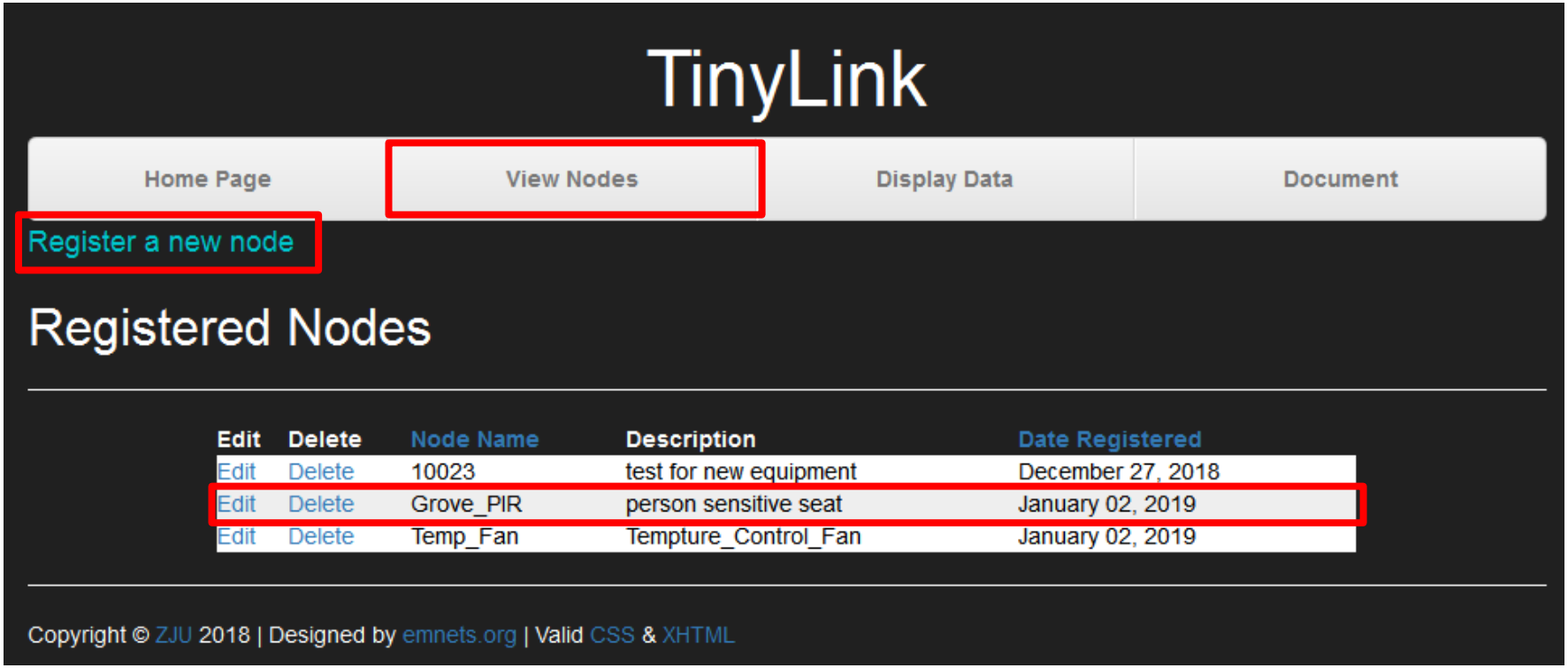
- **智能教室座位使用统计系统：利用红外传感器感知教室座位上是否有人，如果监测到座位上有人，点亮设备端LED灯，并在手机APP显示**

Grove-PIR Motion Sensor是一款利用热电红外传感器来侦测面前120度扇形，距离3米范围内的移动物体，如人或宠物等。

当任何人在其检测范围内移动时，传感器将在其SIG引脚上输出高电平

TinyLink2.0 开发案例

- 步骤1：注册云平台的数据节点。在TinyLink平台View Node界面注册一个新的Grove_PIR节点



TInyLink2.0节点注册

TinyLink2.0 开发案例

- 步骤2：设备端代码设计与构建。监听云平台下行消息，控制LED灯的亮灭

```
REQUIRE("Raspberry pi");
/*以树莓派为开发板*/
TL_MQTT mqtt;        //声明MQTT客户端
/*MQTT Broker参数*/
int port = 1883;
char serverName[] = "10.214.147.171";
char clientName[] = "username";
/*云平台节点参数*/
char pubTopic[] = "Grove_PIR@wt";
char subTopic[] = "Grove_PIR@rt";
char userName[] = "username";
char password[] = "password";
/*统计变量声明与初始化*/
int Time = 0;
int Count = 0;
/*MQTT消息缓存*/
char buf[100];
```

```
/*MQTT下行回调函数*/
void messageArrived(MQTT::MessageData& md){
/*得到MQTT服务器下行消息*/
    MQTT::Message &message = md.message;
    char res[20];
    strncpy(res, (char*)message.payload, 19);
/*根据消息内容点亮或者熄灭LED*/
    if(strncmp(res, "LED_TURNON",
strlen("LED_TURNON")) == 0){
        TL_LED.turnOn();
        delay(500);
            }
    else if(strncmp(res, "LED_TURNOFF",
strlen("LED_TURNOFF")) == 0){
        TL_LED.turnOff();
        delay(500);
            }
}
```

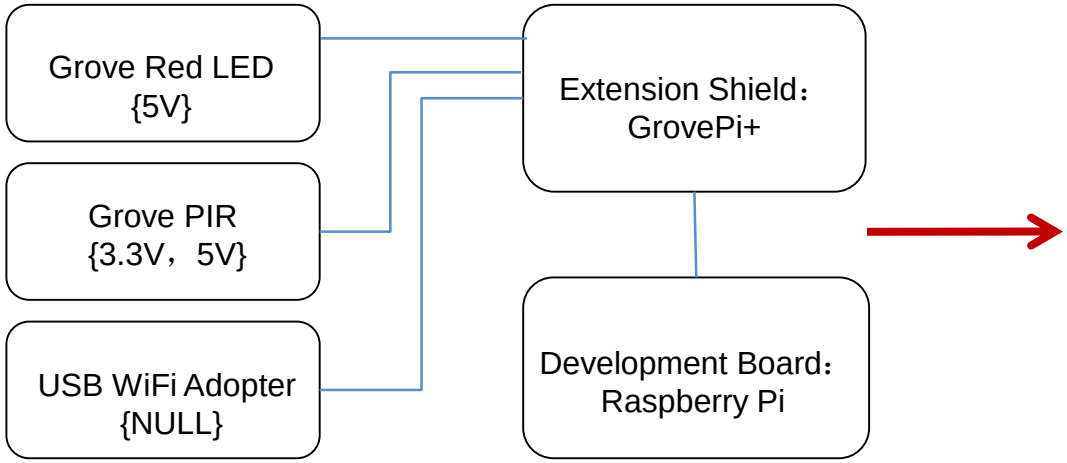
TinyLink2.0 开发案例

- 步骤2：设备端代码设计与构建。设备端使用MQTT协议与云平台进行通信

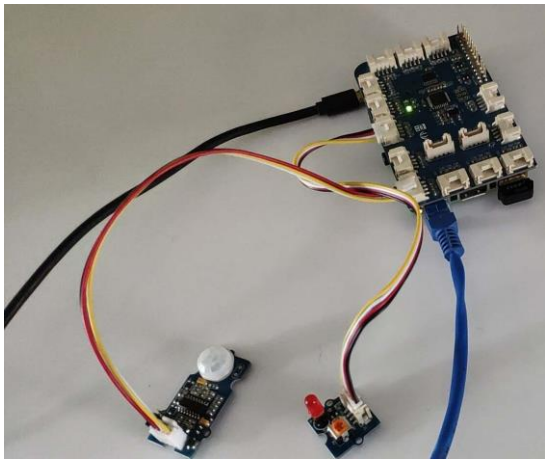
```
void setup(){
/*初始化WiFi并建立连接*/
    TL_WiFi.init();
    TL_WiFi.join("SSID","Password");
/*从WiFi连接实例化MQTT客户端*/
    mqtt = TL_WiFi.fetchMQTT();
/* MQTT客户端连接MQTT Broker */
    mqtt.connect(serverName, port, clientName,
userName,password);
/* 注册回调函数 */
    mqtt.subscribe(subTopic, messageArrived, 0);
}

void loop(){
    if(TL_PIR.read())//红外PIR检测
        Count++;//检测到周围有人移动，计数加1
    Time++;      //计时
/*计时30秒内计数达到10次则座位上有人*/
    if(Time==30){
        if(Count>=10){
            String data = "{\"pir_motion\":1}";
            data.toCharArray( buf,100 );
        }else{
            String data = "{\"pir_motion\":0}";
            data.toCharArray( buf,100 );
        }
/* MQTT客户端向Topic发布消息 */
    mqtt.publish(pubTopic, buf, strlen(buf));
    Time=0;
    Count=0;
    }
/*等待1秒，接受下行数据*/
    mqtt.yield(1000);
}
```

TinyLink2.0 开发案例

- 步骤3：基于TinyLink 平台，上传设备端代码，并在云端进行编译，得到编译后的文件以及硬件连接关系图，根据连接关系图，组装设备并烧写设备端代码



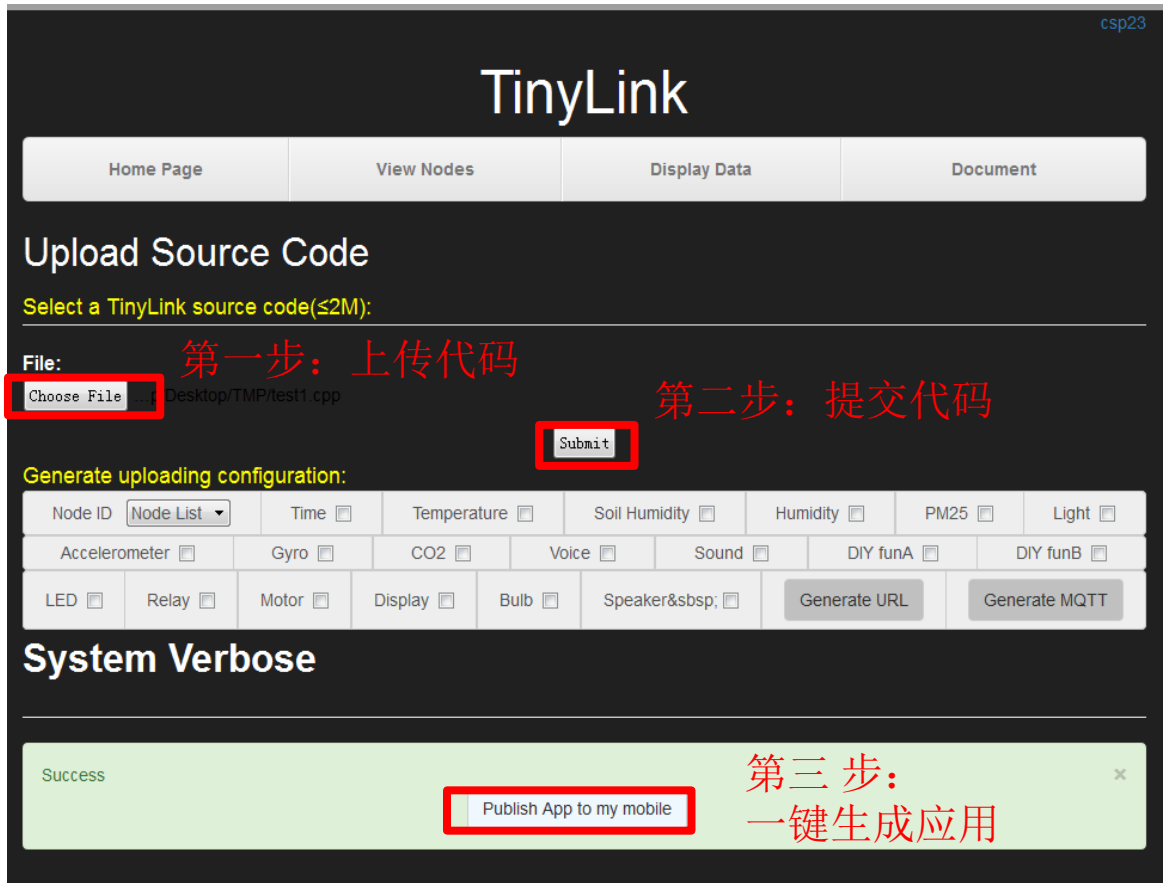
硬件方案图



硬件连接实物图

TinyLink2.0 开发案例

- 步骤4：基于设备端代码，生成移动应用框架，在移动端登录即可看到生成的移动应用

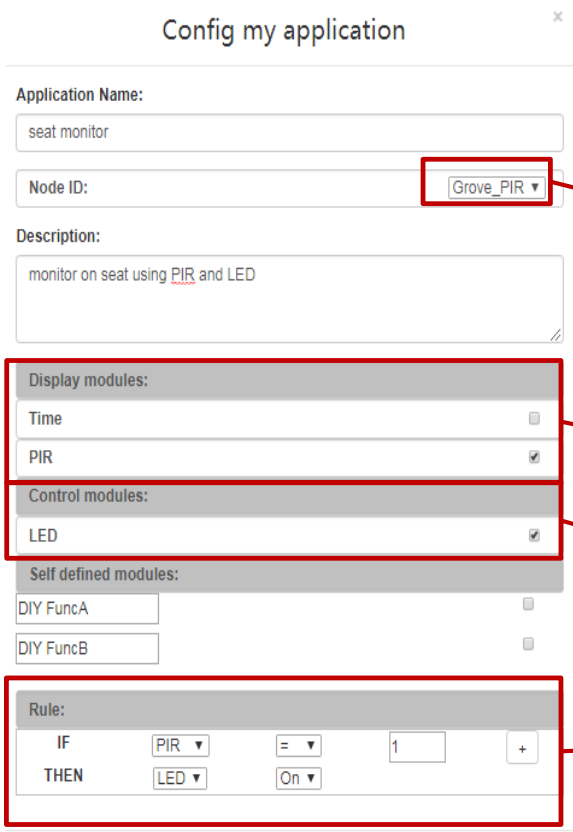


TInyLink2.0设备端

TInyLink2.0应用端

TinyLink2.0 开发案例

- 步骤5：基于自动生成的移动应用框架，配置节点、显示的数据等信息，并配置IFTTT规则，完成服务开发与移动应用开发

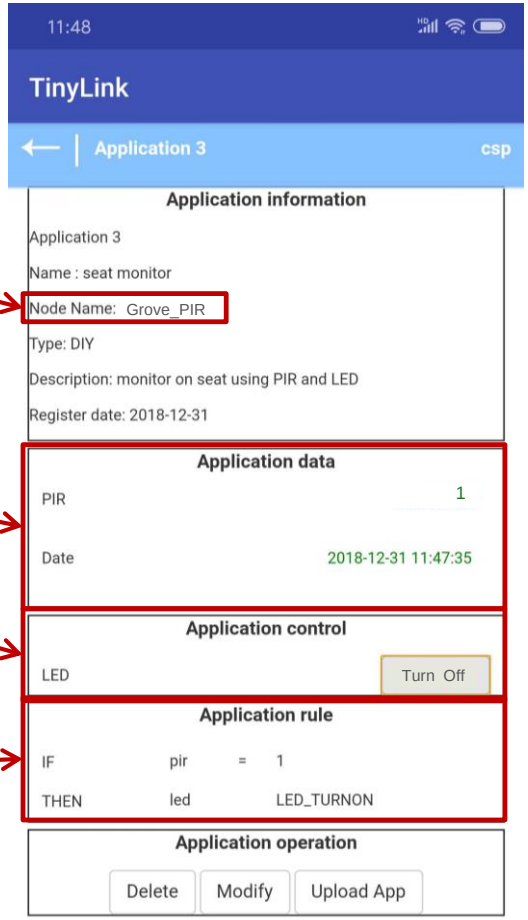


选择数据来源节点

感知类API生成data面板数据

控制类API生成control面板的控件

配置API之间的IFTTT规则



TInyLink2.0服务配置

生成的应用界面

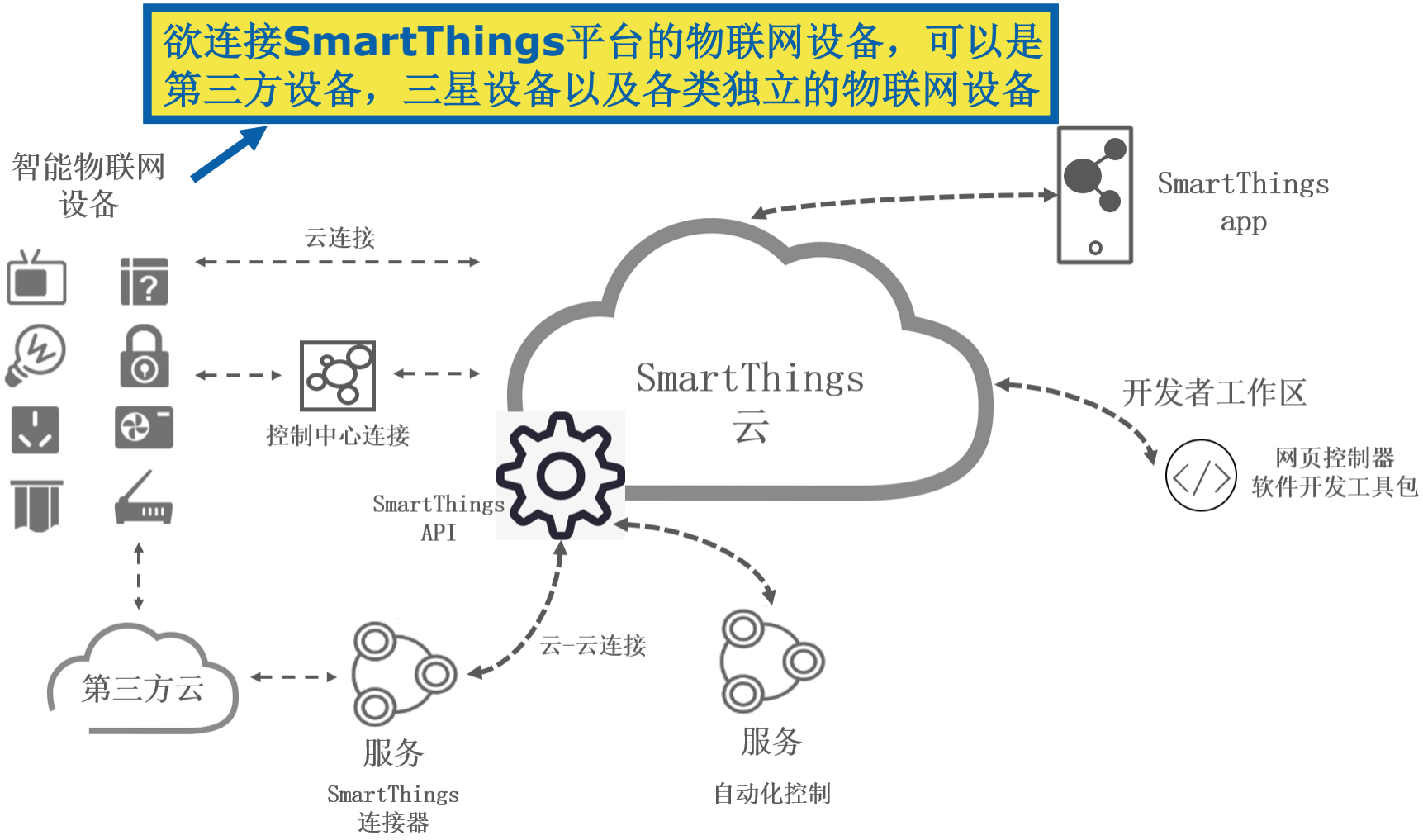
三星 SmartThings 平台

- 概述
- 特点
  - 支持多种设备集成方式
  - 支持基于生命周期的服务开发
  - 支持基于插件的移动应用开发
  - 丰富的安全机制
- 开发案例
  - 天气感知灯

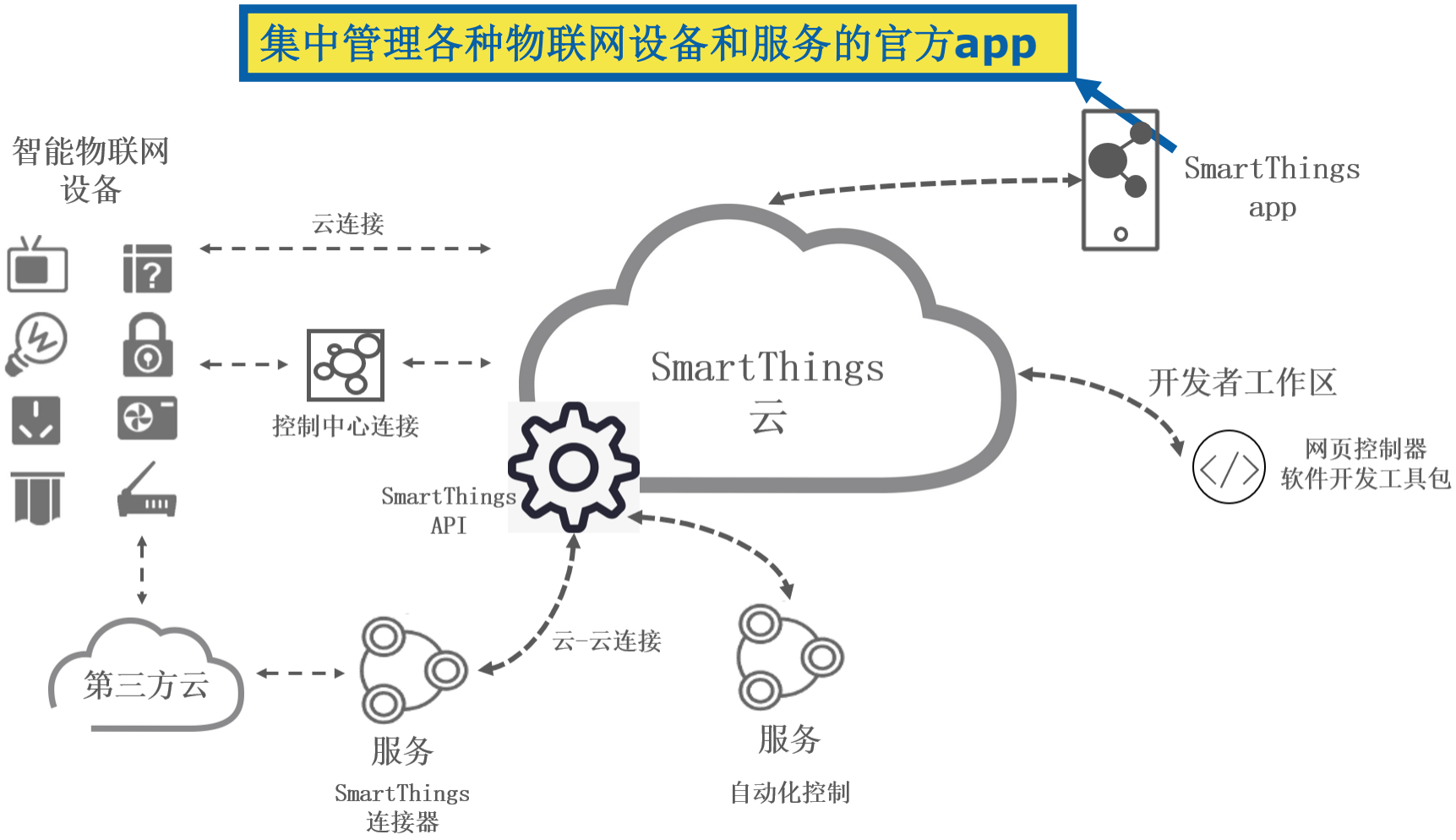
三星SmartThings概述

- SmartThings 成立于 2012 年，于2014年被三星收购
- 综合性一站式的智能家居平台
- SmartThings通过统一的控制中心和官方app集中管理各种各样品牌和不同平台的物联网设备
- 为开发者提供了一个应用开发的开源社区

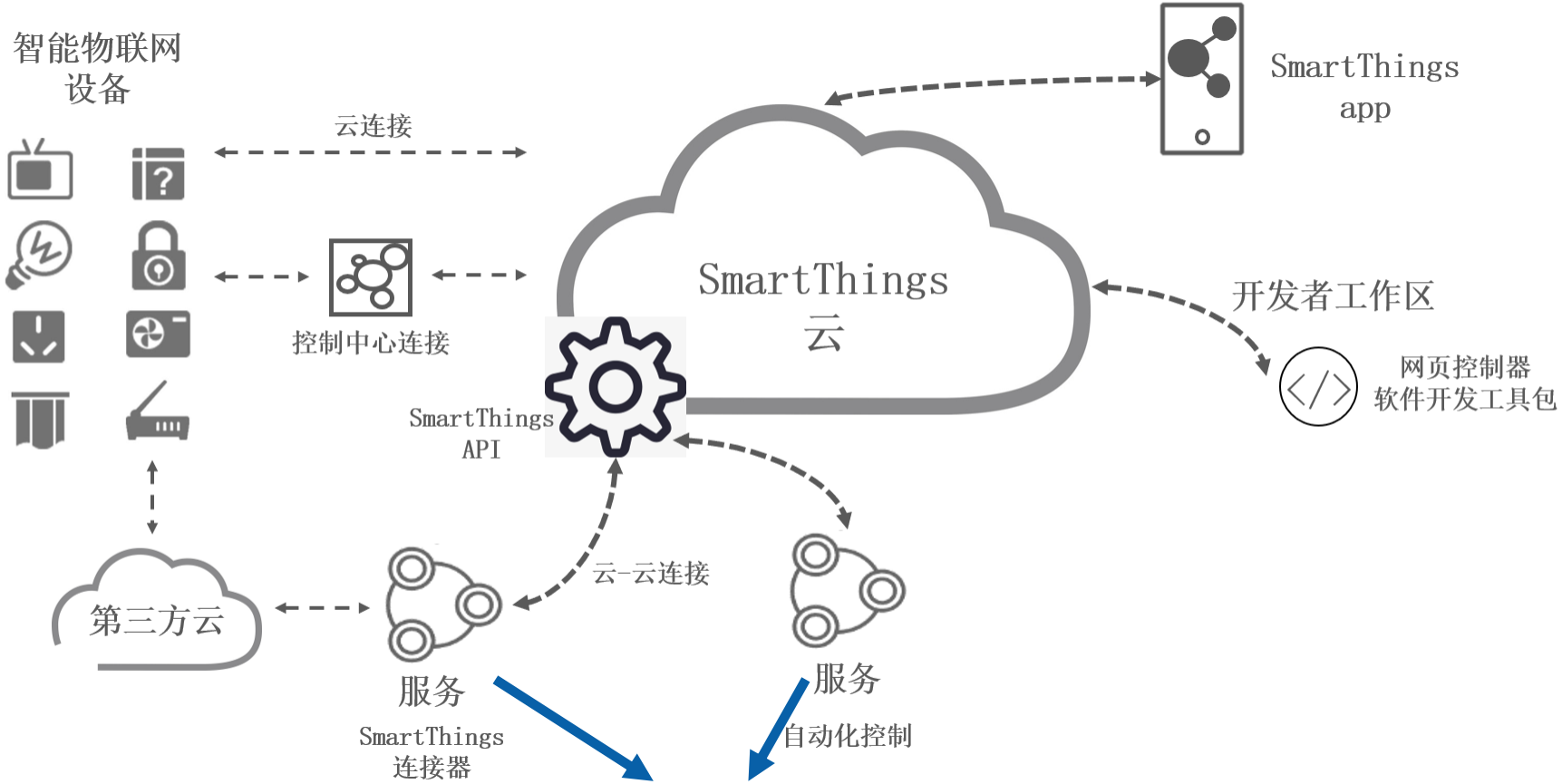
SmartThings平台架构



SmartThings平台架构

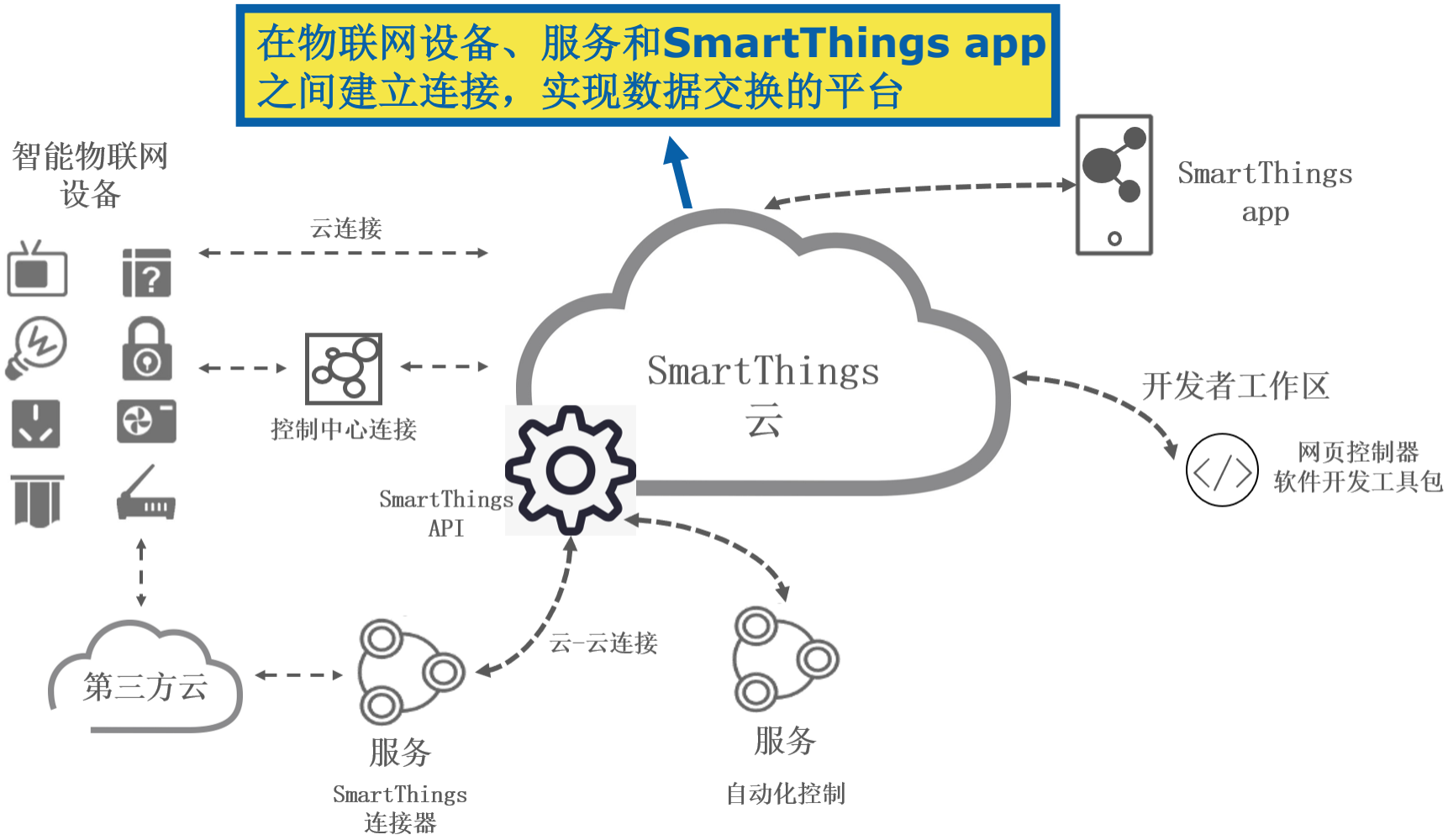


SmartThings平台架构

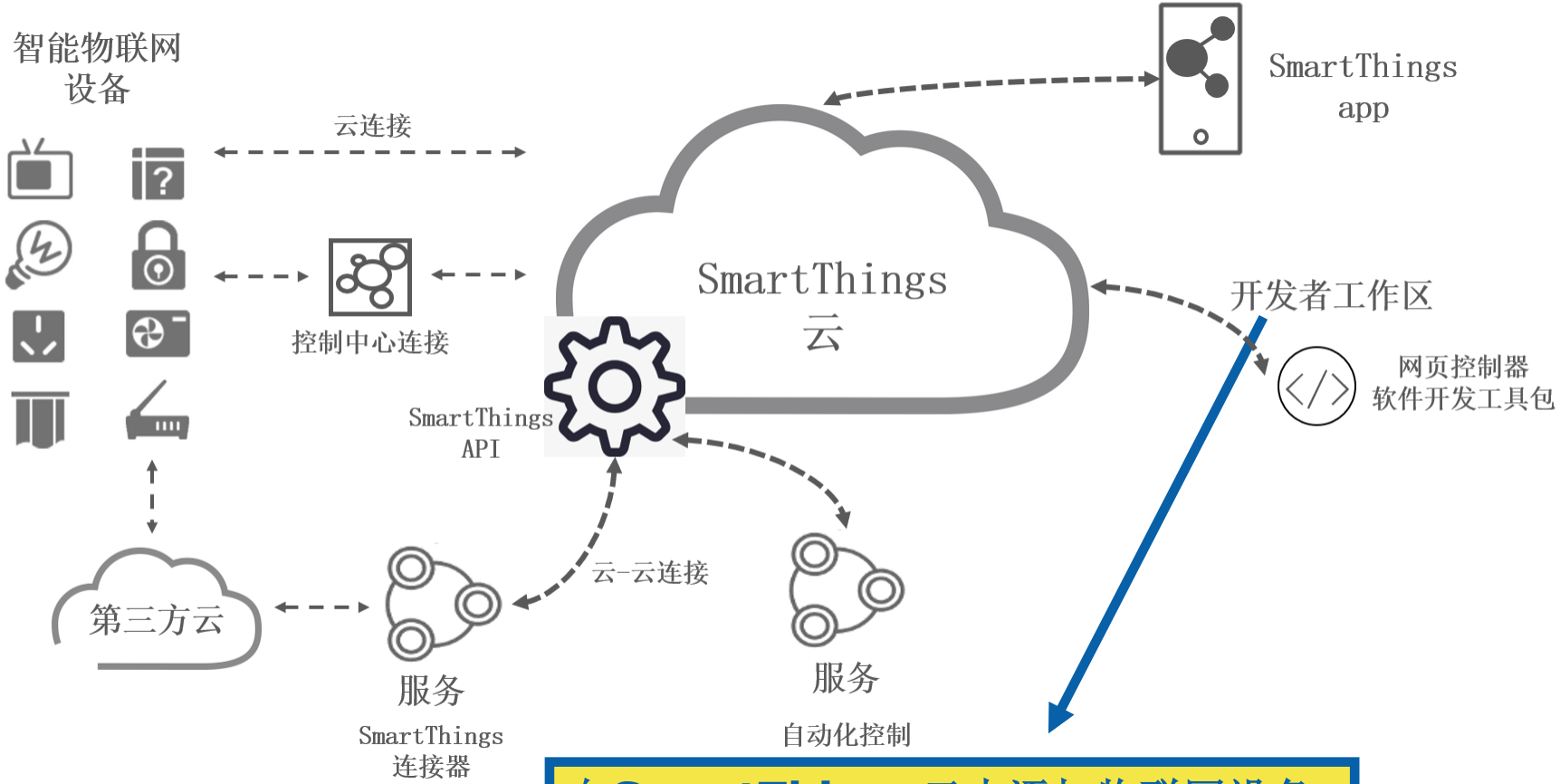


在SmartThings平台和设备间建立连接
得以控制和获取物联网设备状态的各类服务

SmartThings平台架构

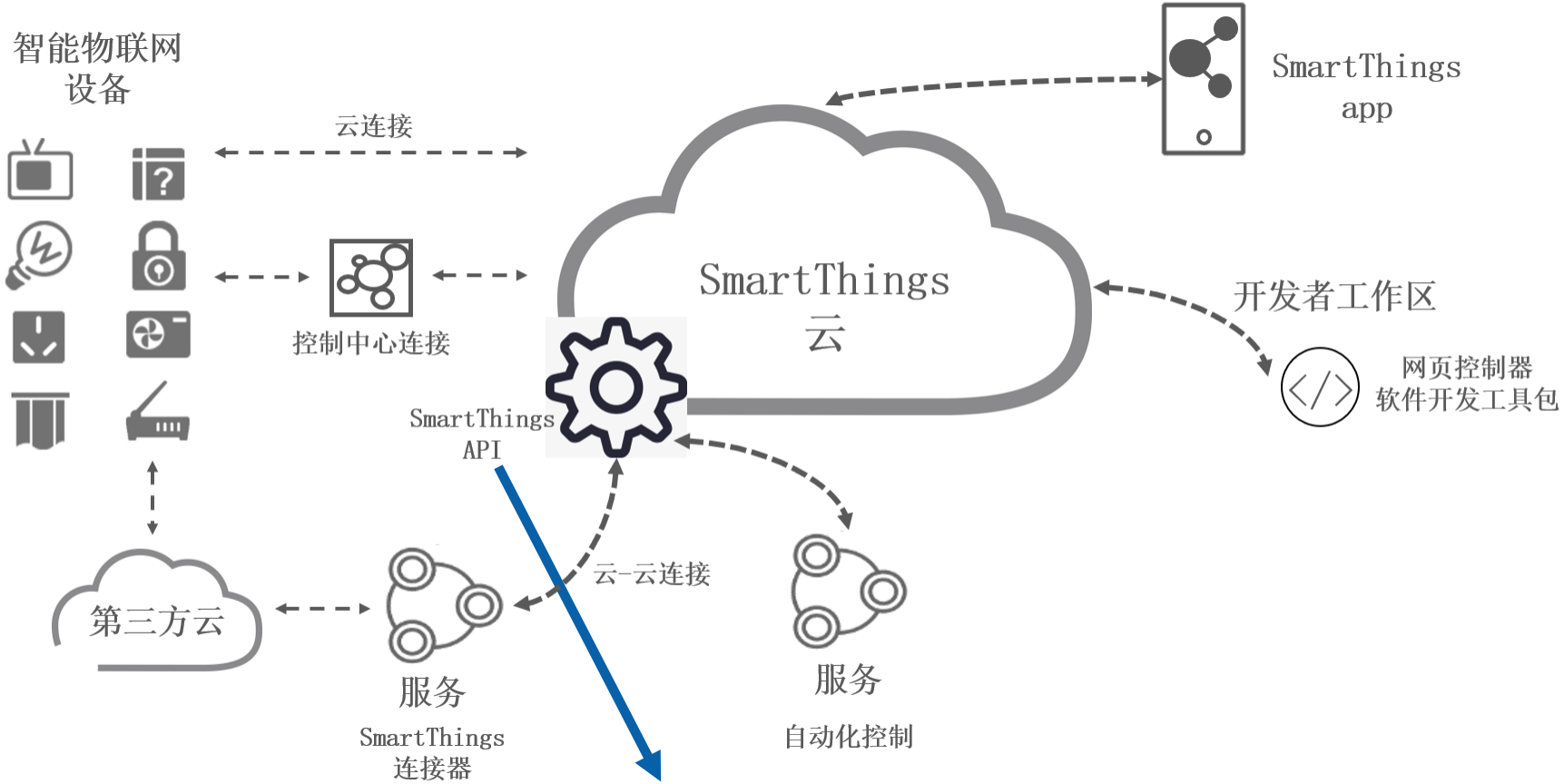


SmartThings平台架构



向SmartThings云中添加物联网设备和自定义服务的一套工具

SmartThings平台架构

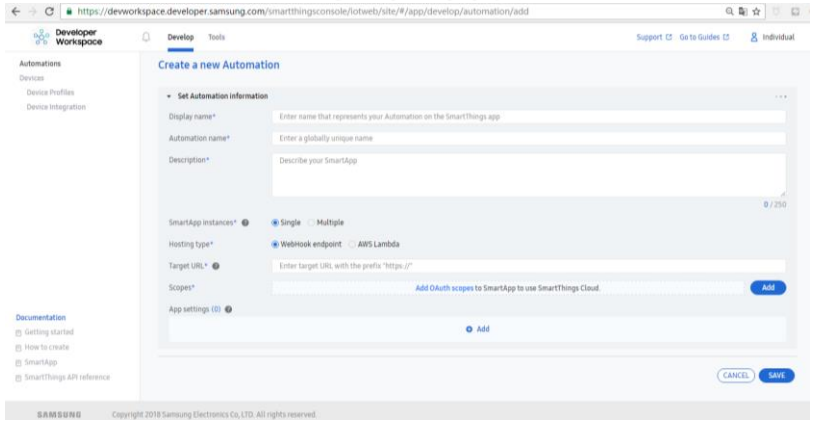


SmartThings平台提供了用于集成、控制、管理物联网设备和自定义服务的接口

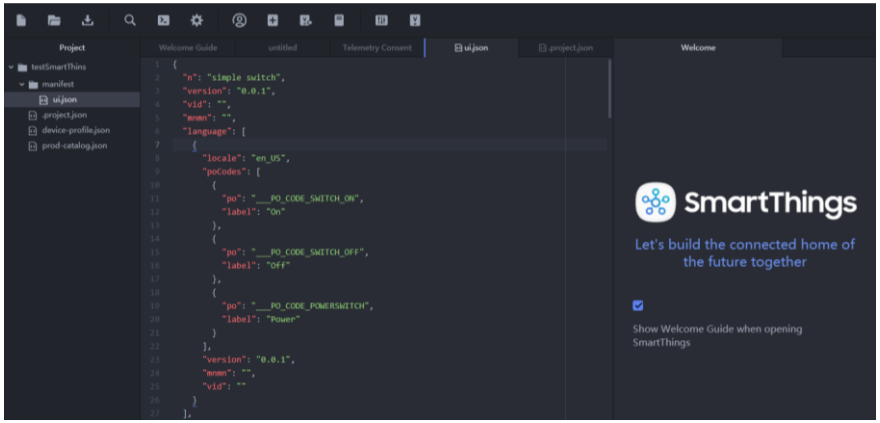
SmartThings开发者工作区

- 网页控制台
  - 支持设备、服务的图形化集成
  - 设备插件模拟器和虚拟设备
  - 消息日志
- SmartThings SDK
  - 支持设备、服务集成
  - 依附于Atom编辑器的IDE，并提供命令行接口
  - 设备插件模拟器和虚拟设备

网页控制台



SDK



SmartThings支持的物联网设备

- 目前支持的48个厂商的智能物联网设备共304款
- 设备分类

类别	数量
控制中心	9
集成平台	8
语音助手	8
灯&开关	105
插座	39
传感器	43
摄像头&门铃	11
门锁	40
温度调节装置	9
音响	14
其他	18

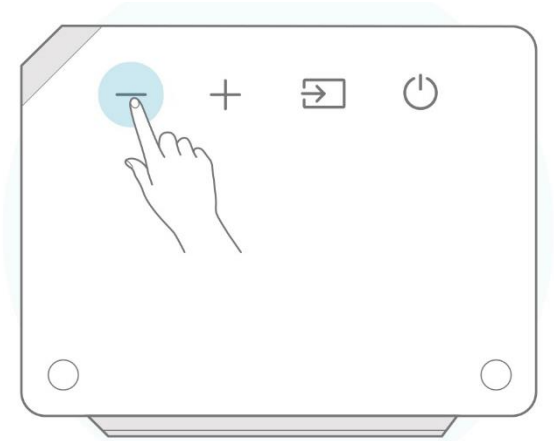
SmartThings关键特性

- 设备开发
  - 基于设备模型的设备端开发，提供三种设备集成方式
- 服务开发
  - 基于生命周期的服务开发
- 移动应用开发
  - 支持插件式移动应用开发
- 三端应用集成简化
  - 基于设备档案的集成方案

设备：集成方式

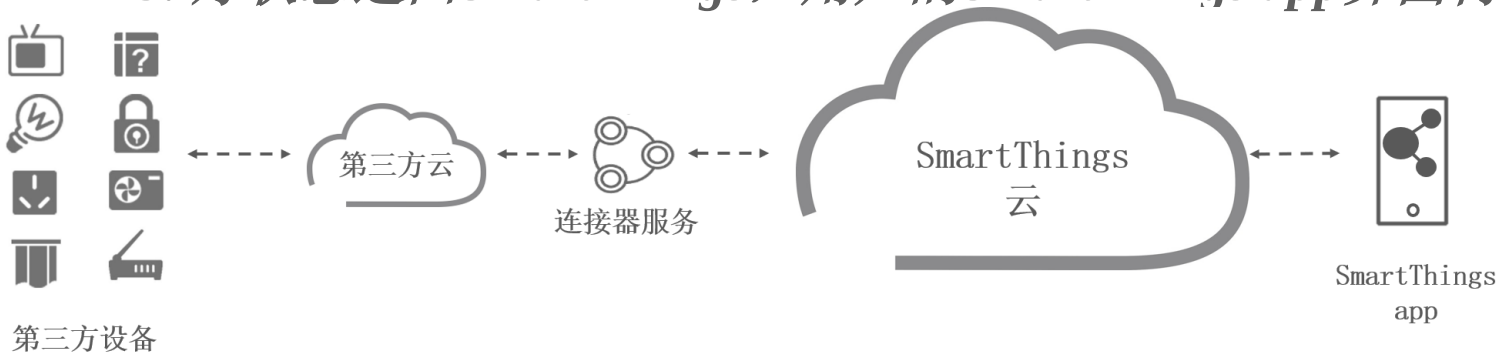
❑ 设备- SmartThings云连接集成

- 设备开发者预先申请安全认证并将其存放在设备中
  - *X.509 (证书)认证、ED25519 (公钥)认证*
- 设备开发者自主开发SmartThings要求的设备接口
- 需要向SmartThings云提供一个URL来接收和发送数据和命令，并通过CoAP协议与SmartThings云通信
- 需要提供与SmartThings app配对的方案
  - *通过SmartThings云通信配对*
  - *遥控按钮配对、设备按钮配对（基于共同连接的室内WiFi）*



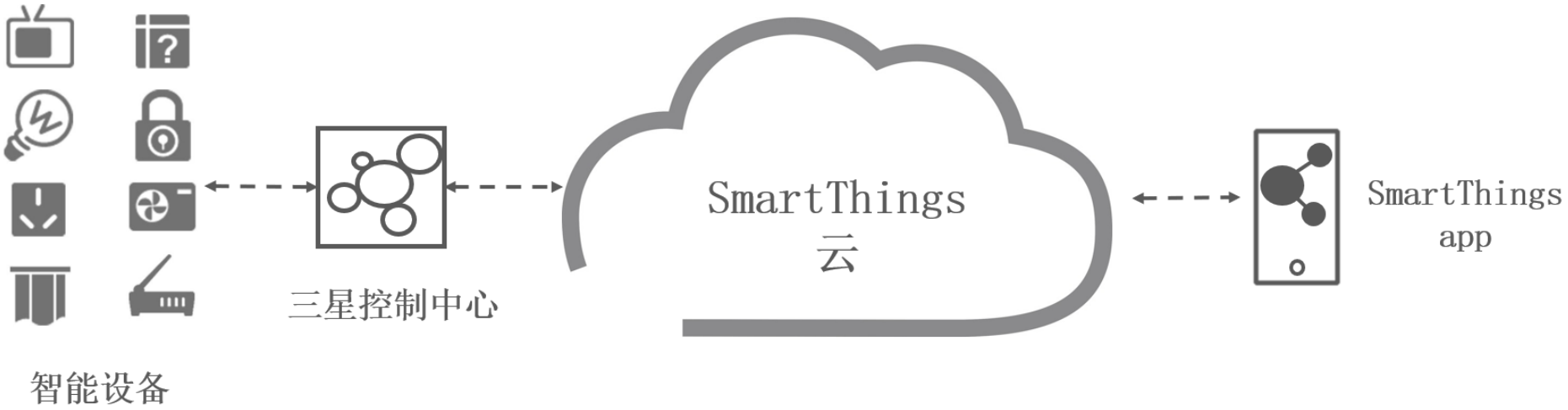
设备：集成方式

- 设备-第三方云- SmartThings云连接集成
  - 第三方设备可以通过第三方云和自定义的连接器服务间接地与SmartThings云通信
    - *此连接器要求实现云-云之间的通信和命令的解析*
  - 工作流程（以开启第三方产品：LIFX智能灯为例）
    - *1. 用户在app中点击打开LIFX智能灯的按钮*
    - *2. SmartThings云接收此携带第三方访问令牌的命令发送到连接器服务*
    - *3. 服务解析此命令并将其传输到第三方的LIFX云*
    - *4. LIFX云发送“on”命令到LIFX智能灯将其开启*
    - *5. 灯状态返回SmartThings，用户的SmartThings app界面得到更新*



设备：集成方式

- 设备-控制中心- SmartThings云连接集成
  - 支持ZigBee或Z-Wave物联网设备可以通过连接官方的控制中心与SmartThings云通信
  - 开发者需要自主开发SmartThings要求的设备接口，以及与控制中心配对的方案
  - 支持的控制中心
    - *Samsung Connect Home hub*
    - *SmartThings hub*

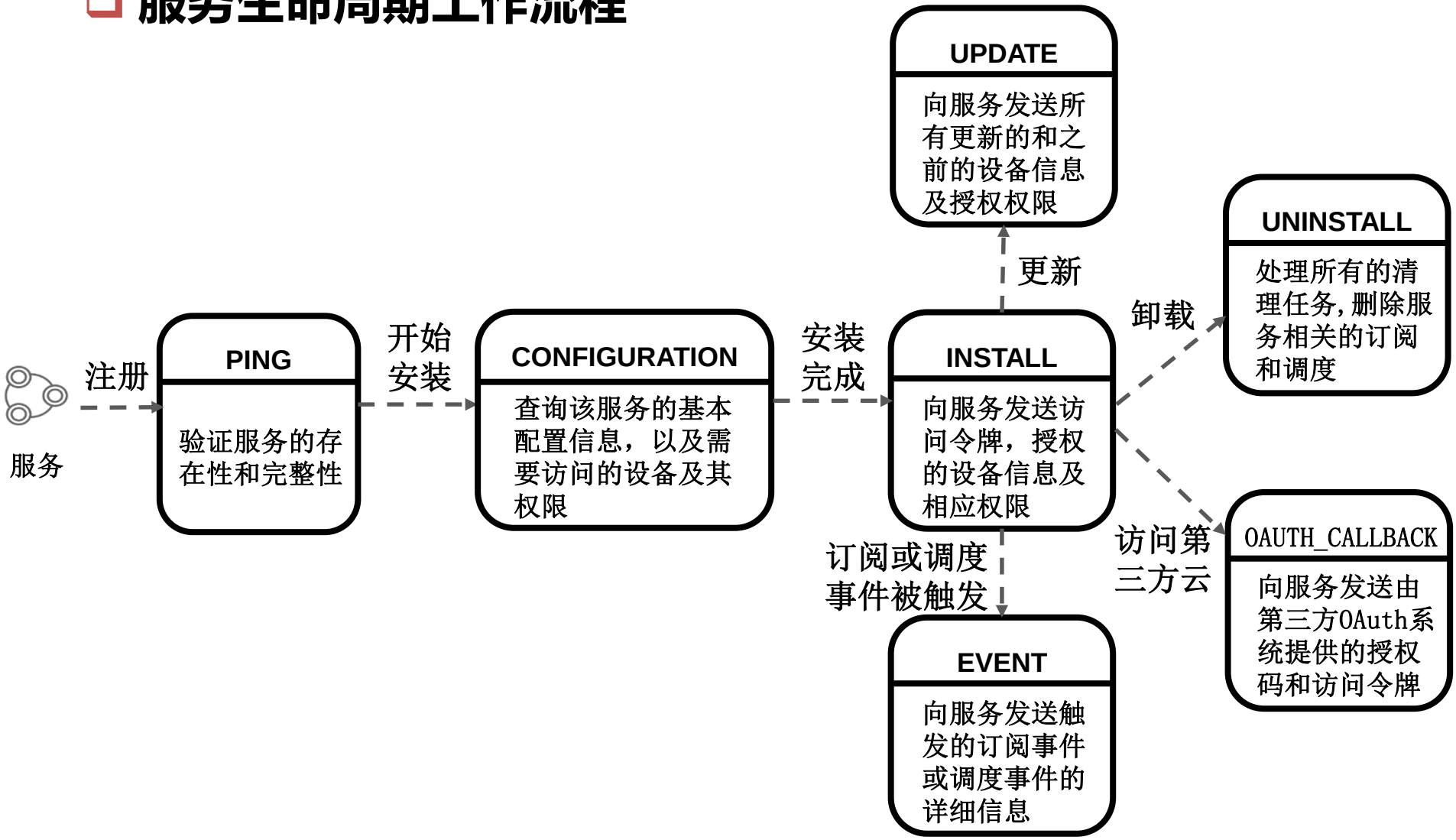


服务：生命周期

- 开发者需要在服务中实现各类**生命周期请求**的处理程序
  - 生命周期阶段持续时间：从服务注册到服务卸载
  - 7个生命周期阶段
    - *PING（服务存在访问请求）*
    - *CONFIGURATION（服务相关配置信息请求）*
    - *INSTALL（服务安装请求）*
    - *UPDATE（服务更新请求）*
    - *EVENT（服务响应触发事件请求）*
    - *OAUTH_CALLBACK（服务访问第三方云请求）*
    - *UNINSTALL（服务卸载请求）*
  - SmartThings平台在服务的每个生命周期阶段向其发送一个POST请求，服务要求能够解析并响应此请求

服务：生命周期

❑ 服务生命周期工作流程



服务：事件订阅与调度

- **订阅的事件被触发时，服务将收到一个来自平台的POST请求，其中会包含触发事件的相关信息，服务需要据此在EVENT阶段作出响应**
  - 两种订阅方式
    - *设备订阅（声明订阅类型为DEVICE）*
      - 如：订阅某一个灯的开关状态变化事件
    - *功能订阅（声明订阅类型为CAPABILITY）*
      - 如：订阅用户控制的所有灯的开关状态变化事件
- **调度指的是为物联网设备创建工作计划，在指定时间或周期性地执行指定事件**
  - 两种调度方式
    - *一次调度：在指定的日期和时间执行一次*
      - 如：调度灯在明天早上八点开启
    - *周期性调度：使用Cron表达式周期性地执行事件*
      - 如：调度灯在每天早上八点开启

Cron表达式

字段名	可取值	允许特殊字符
秒（在SmartThings中省略）	0-59	, - * /
分	0-59	, - * /
时	0-23	, - * /
日	1-31	, - * ? / L W C
月	1-12 or JAN-DEC	, - * /
周	1-7 or SUN-SAT	, - * ? / L C #
年（可选字段）	empty, 1970-2099	, - * /

- “*”字符：表示匹配该域的任意值
- “?”字符：表示不确定的值，用于日和周字段是因为这两字段可能会相互冲突。例如想在每月的20日触发调度，那么需要在周字段设置问号
- “,”字符：同时指定多个值
- “-”字符：指定一个值的范围
- “/”字符：指定一个值的增加幅度。n/m表示从n开始，每次增加m
- “L”字符：用在日表示一个月中的最后一天，用在周表示该月最后一个星期X: 6L表示最后的周五
- “W”字符：指定离给定日期最近的工作日(周一到周五)
- “C”字符：指定日期的后一天。 5C在日字段表示当月第六天，在周字段表示周五
- “#”字符：表示该月第几个周X。6#3表示该月第3个周五

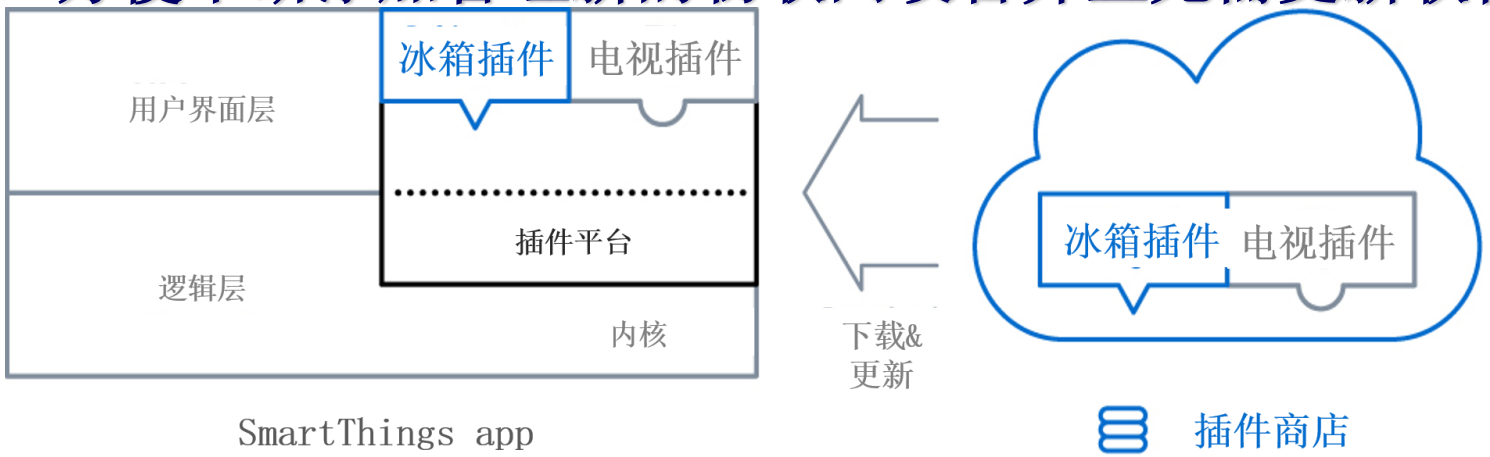
- 0 2 1 * ? * 表示每月的1日的凌晨2点
- 0 10,14,16 * * ? 每天上午10点，下午2点，4点
- 0/30 9-17 * * ? 朝九晚五工作时间内每30分钟
- 15 10 ? 6L 2016-2018 表示16-18年的每个月的最后一个星期五上午10:15
- 15 10 C * ? 第二天上午10:15
- 15 10 ? * 6#3 每月的第三个星期五上午10:15
- 15 10 15W * ? 离每月15日最近的工作日的上午10:15
- 15 10 LW * ? 每月最后一个工作日的上午10:15
- 习题：1.在每年三月的周三至周五的下午2:10和2:44触发
- 2. 在离每月15日最近的工作日下午2点到2:55期间和下午6点到6:55期间的每5分钟触发

服务：开发形式

- **WebHook形式**
  - WebHook对应一个公网可以访问的URL
  - 在此URL上保存服务代码，即事件触发后对应的自定义回调函数，能接收并处理来自SmartThings平台的HTTP POST请求
  - 支持任何语言和工具开发服务代码
- **AWS Lambda函数形式**
  - 亚马逊提供的一项计算服务，可使用户无需预配置或管理服务器即可运行自定义回调函数
  - 在AWS平台和SmartThings平台之间配置相互认证
  - 当设备事件发生时，Lambda服务就会自动使用指定的AWS资源去执行预先关联的回调函数
  - 支持多种开发语言：Node.js、Java、C#、Go 和 Python

移动应用：插件式工作

- 在官方的SmartThings app中根据匹配的物联网设备下载、添加其对应的插件
- 插件是一个使用HTML、CSS、JavaScript和JSON设计的网页应用程序，并被打包成一个PPK（Plugin Package）文件嵌入SmartThings app中
  - 定义了设备的移动应用界面及一些交互行为
  - 对于某一设备，用户既可以使用SmartThings平台提供的默认插件，也可以根据需求自己设计插件
  - 方便不断添加管理新的物联网设备并且无需更新软件



移动应用：插件式工作流程

安全机制：设备和应用的认证与发布

- 设备及其应用在开发完成之后还需要通过SmartThings工作认证项目（WWST，Works with SmartThings）的功能鉴定和安全认证才能在平台上正式发布
  - 功能审核
    - *SmartThings认证团队 + 全球认证中心审核*
  - 安全审核
    - *SmartThinas安全经理+ 认证团队审核*

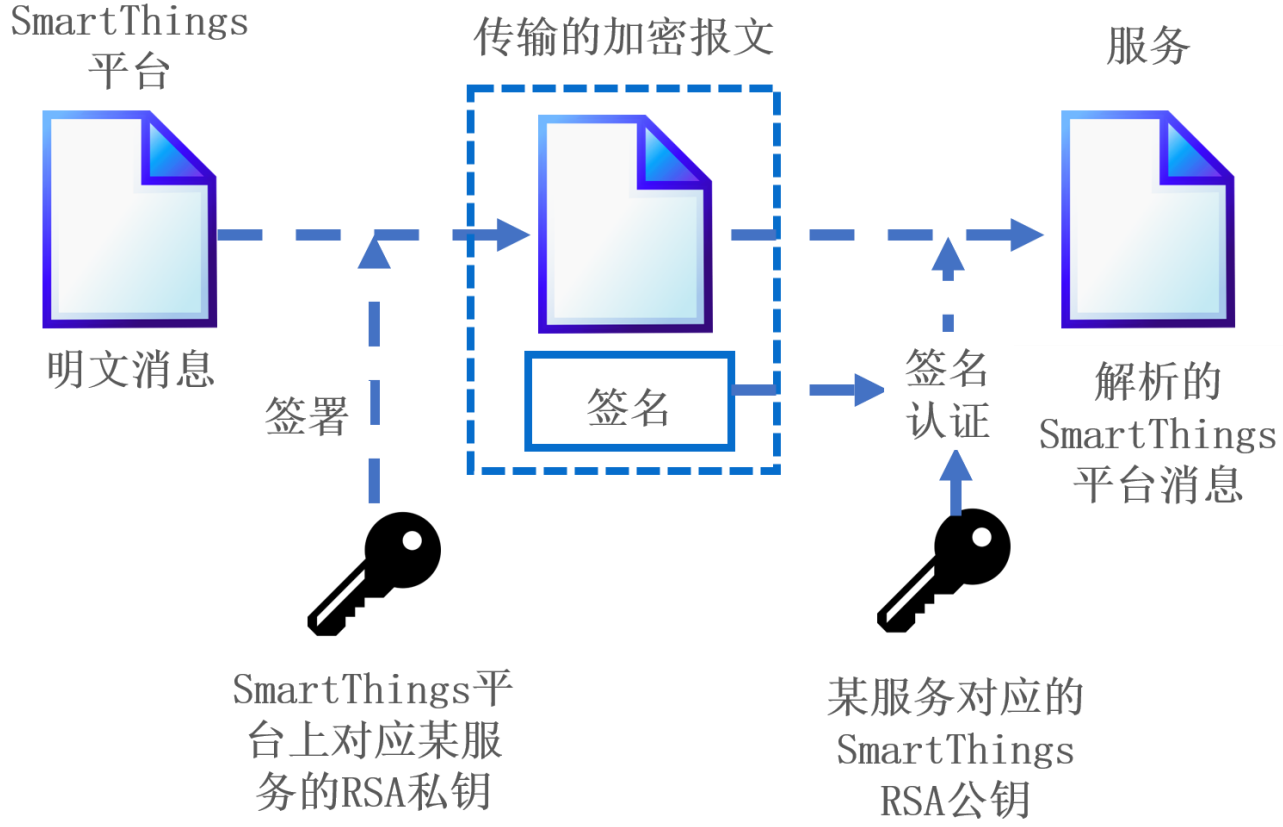
安全机制：平台与服务的通信安全

- 给服务或第三方云发放OAuth 2.0的无记名令牌，使其可以调用SmartThings平台的资源（包括API和设备资源）
  - 创建令牌时指定允许访问的平台API和设备权限

- 两种令牌形式
  - 服务令牌
    - *包含一个认证令牌和一个更新令牌（用于更新此服务令牌）*
    - *用于已认证服务或第三方云与SmartThings平台API之间的通信*
  - 个人访问令牌
    - *包含一个认证令牌*
    - *用于个人脚本或测试与SmartThings平台API之间的通信*

安全机制：平台与服务的通信安全

- 在服务中添加SmartThings平台的RSA公钥，确保消息来源于SmartThings平台
  - 在将服务集成到SmartThings平台时，SmartThings会生成对应该服务的RSA公钥



开发案例-天气感知灯

- 设备：需要一款与SmartThings平台兼容的彩色灯泡，如西尼亚的智能RGBW或LIFX智能灯或飞利浦的Hue
- 效果：根据用户的邮编地址的天气自动设置灯的颜色
  - 紫色（如果在预报中有降水），橙色（如果天气预报温度高于80华氏度），蓝色（如果天气预报温度低于50华氏度），或者白色（如果没有降水并且温度在50到80华氏度之间）
- 需要软件：
  - Node.js，npm：本例使用Node.js语言开发
  - ngrok：在本地创建一个全球可用的URL存放WebHook服务
  - 安装Docker运行Redis，或者手动安装和运行Redis：高效的开源数据库，支持事件处理

天气感知灯

工程目录

- weather-color-light
  - *config*
    - forecast-test.json //日志文件
    - smartthings_rsa.pub //SmartThings公钥文件
  - *constants*
    - colors.js //灯光颜色设置文件
  - *lib* //一些使用到的API库
    - commands.js //向设备发送命令的API库
    - config.js //应用页面的配置信息
    - scheduling.js //调度相关的库
    - weather.js //获得邮编所在地天气的库
  - *package.json* //定义应用基本信息和安装依赖项文件
  - *server.js* //服务的主要处理文件

天气感知灯

1.设置不同灯光颜色的文件colors.js

```
module.exports = {
  PURPLE:  {hue: 75, saturation: 100},
  BLUE: {hue: 70, saturation: 100},
  WHITE: {hue: 79, saturation: 7},
  RED: {hue: 10, saturation: 100}
};
```

2.开发需要的库文件

- lib
  - *commands.js* *//将设备控制命令封装为POST请求*
  - *config.js* *//包含服务的配置信息；调度事件的配置信息；服务包含的基本信息*
  - *scheduling.js* *//将向SmartThings发送和删除cron调度的命令封装为POST请求*
  - *weather.js* *//集成第三方map应用，获得邮编所在地实时的天气，并可以输出对应的颜色*

天气感知灯

❑ 3.在package.json中定义应用基本信息和依赖项

```
{
  "name": "weather-color-light-SmartApp",
  "version": "1.0.0",
  "description": "sample SmartThings webhook SmartApp",
  "repository": "https://github.com/SmartThingsCommunity/weather-color-light-smartapp-nodejs",
  “main”: “server.js”, //主要处理文件
  …
  "author": "SmartThings",
  "license": "Apache-2.0",
  "dependencies": {        //在生产环境中需要用到的依赖
    "ava": "^0.23.0",
    …  },
  "devDependencies": {        //在开发、测试环境中用到的依赖
    "ava": "^0.23.0"
  }
}
```

应用基本信息

天气感知灯

❑ 4.创建主要处理文件server.js

```
'use strict';
const express = require('express');
const bodyParser = require('body-parser’);
…
app.post('/', function (req, response) {
  if (req.body && req.body.lifecycle === "PING" || signatureIsVerified(req)) {
 //签名认证，确认消息来自SmartThings
     handleRequest(req, response); //处理来自SmartThings的请求
  } else {
    response.status(401).send("Forbidden");
  }
});

function signatureIsVerified(req) {
  const publicKey = fs.readFileSync('./config/smartthings_rsa.pub', 'utf8’);
//使用存放在smartthings_rsa.pub文件中的公钥认证消息
…}
```

天气感知灯

❑ 4.创建主要处理文件server.js

```
function handleRequest(req, response) {        //各生命周期的SmartThings
                                                 请求处理程序
  let evt = req.body;
  let lifecycle = evt.lifecycle; //读取当前生命周期阶段
…
  switch(lifecycle) {
    case 'PING': {              //设置PING阶段响应
      let chal = evt.pingData.challenge; //返回服务号
      response.json({statusCode: 200, pingData: {challenge: chal}});
      //服务的PING响应JSON
      break;    }
    case 'CONFIGURATION': {     //设置CONFIGURATION阶段响应
      res = stConfig.handle(evt.configurationData); //返回服务配置信息
      …
      response.json({statusCode: 200, configurationData: res});
      break;}
```

天气感知灯

4.创建主要处理文件server.js

```
case 'INSTALL': {            //设置INSTALL阶段响应
  let token = evt.installData.authToken;      //服务的授权令牌，确保能够控制设备
  setBulbColor(evt.installData.installedApp, token); //实现灯能根据天气设置颜色
  createSchedule(evt.installData.installedApp, token); //创建调度
  response.json({statusCode: 200, installData: {}});
  break;    }
case 'UPDATE': {             //设置UPDATE阶段响应，内容与INSTALL近似
  …
  response.json({statusCode: 200, updateData: {}});
  break;    }
case 'UNINSTALL': {          //设置UNINSTALL阶段响应
  response.json({statusCode: 200, uninstallData: {}});
  break;    }
case 'EVENT': {              //设置EVENT阶段响应
  handleEvent(evt.eventData); //处理调度事件
  response.json({statusCode: 200, eventData: {}});
  break;    }
default: {
  console.log(`Lifecycle ${lifecycle} not supported`);    } } }
```

天气感知灯

❑ 4.创建主要处理文件server.js

```
function setBulbColor(installedApp, token) {//设置灯能根据天气设置颜色的函数
  const zipCode = installedApp.config.zipCode[0].stringConfig.value; //获取邮编
  …
  weather.getForecast(zipCode)        //根据天气设置灯色的代码段
    .then(function(forecast) {
    const color = weather.getColorForForecast(forecast, chunks); //获取天气对
                                                                  应的颜色
    commands.actuate(deviceId, token, [     //向设备发送携带令牌的命令
      { command: 'on',   capability: 'switch',   component: 'main',   arguments: []   },
      //打开灯
      { command: 'setLevel',    capability: 'switchLevel',    component: 'main',
        arguments: [20]   },
      //设置灯的亮度
      { command: 'setColor',    capability: 'colorControl',    component: 'main',
        arguments: [color]   }
      //设置灯的颜色
    ])
    … //命令出错日志代码
}
```

天气感知灯

4.创建主要处理文件server.js

```
function createSchedule(installedApp, token) {//创建调度的函数
  const scheduleSetting
=installedApp.config.scheduleInterval[0].stringConfig.value;
  const scheduleInterval =
stConfig.getScheduleInterval(scheduleSetting); //设置调度周期

  scheduling.deleteSchedules(installedApp.installedAppId, stApi,
token)
  .then(function(resp) {          //默认先删除原有调度再创建新的调度
      scheduling.createCron(`0/${scheduleInterval} * * * ? *`, “UTC”,
installedApp.installedAppId, stApi, token) //每隔scheduleInterval的
时间调度事件被触发一次，会在EVENT周期中调用handleEvent函数
 … //创建和删除调度出错日志代码
}
```

天气感知灯

❑ 4.创建主要处理文件server.js

```
function handleEvent(eventData) {  //创建在EVENT阶段的事件处理的函数
  const eventType = eventData.events[0].eventType;
  const token = eventData.authToken;
  if (“TIMER_EVENT” === eventType) { //调度事件触发
    let timerEvent = eventData.events[0].timerEvent;
    console.log(`Received timer event for schedule ${timerEvent.name} at
${timerEvent.time}`);
    setBulbColor(eventData.installedApp, token); //重新根据天气设置灯色
  } else {
    console.error(`This app only expects TIMER_EVENTs. Got
${eventType}`)   //事件出错日志代码
  }
}
let server = app.listen(3005);      //在3005端口监听
module.exports = server;
console.log('Open: http://127.0.0.1:3005');
```

天气感知灯

5.在网页控制台上完成天气感知灯应用的安装

- 步骤一：导入第三方天气地图信息
  - *在Open Weather Map网站上创建一个API密钥，并将其存储在服务的名为WEATHER_API_KEY的环境变量中*

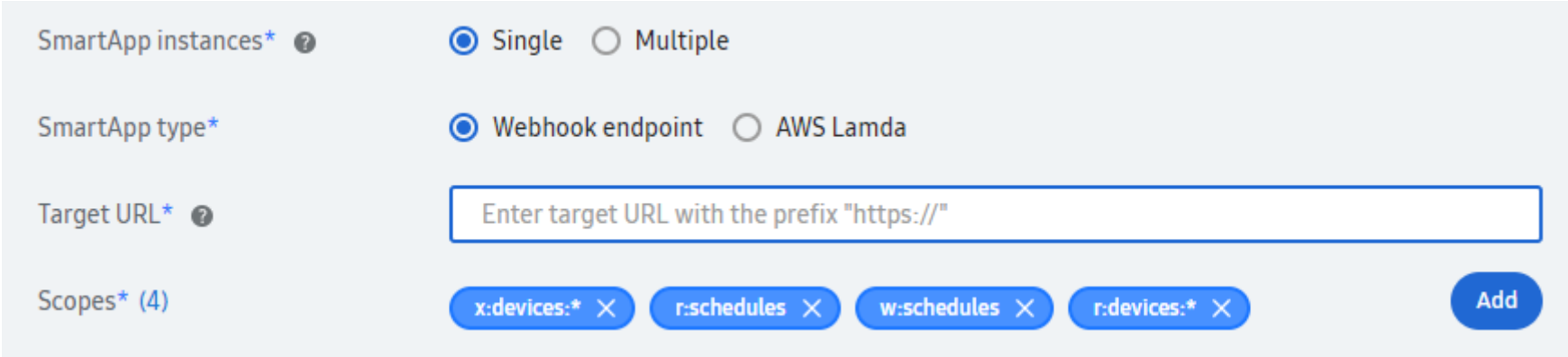
- 步骤二：启动WebHook服务
  - *安装此应用的所有依赖项：npm install*
  - *启动服务器：npm start*
  - *启动ngrok（在另一个终端窗口/选项卡）：ngrok http 3005，此时能得到一个自建服务器的https URL*

天气感知灯

5.在网页控制台上完成天气感知灯应用的安装

步骤三：集成服务到SmartThings平台中

- *点击开发者工作区中的Automation选项卡*
- *填写服务信息：在SmartApp type选项中选择WebHook endpoint，输入自建服务器的URL，并在Scope选项中配置需要的权限*



- *配置完成后可以获得一个SmartThings平台公钥，将此公钥复制到服务的config/smartthings_rsa.pub文件中*
- *点击SELF-PUBLISH注册此服务*

天气感知灯

5.在网页控制台上完成天气感知灯应用的安装

- **步骤四：重启WebHook**
  - *停止npm服务器：CTRL-C*
  - *再次启动服务器：npm start（确保公钥可以正常认证SmartThings平台的请求）*
- **步骤五：测试此天气感知灯应用**
  - *在手机上启用开发者模式并打开SmartThings app（目前只支持安卓手机）*
  - *转到app的Automations选项卡并点击ADD AUTOMATION。此时能看到此天气感知灯应用，点击安装*
  - *在配置屏幕上补全所有必需的输入（如：输入用户所在地的邮编）*
  - *一旦此应用安装完毕，配置的灯泡就会开启，并根据当前天气自动设置灯光颜色*

Link Develop 平台

- 概述
- 特点
  - 物模型
  - 流式服务编排
- 开发案例
  - 智能教室座位使用统计系统

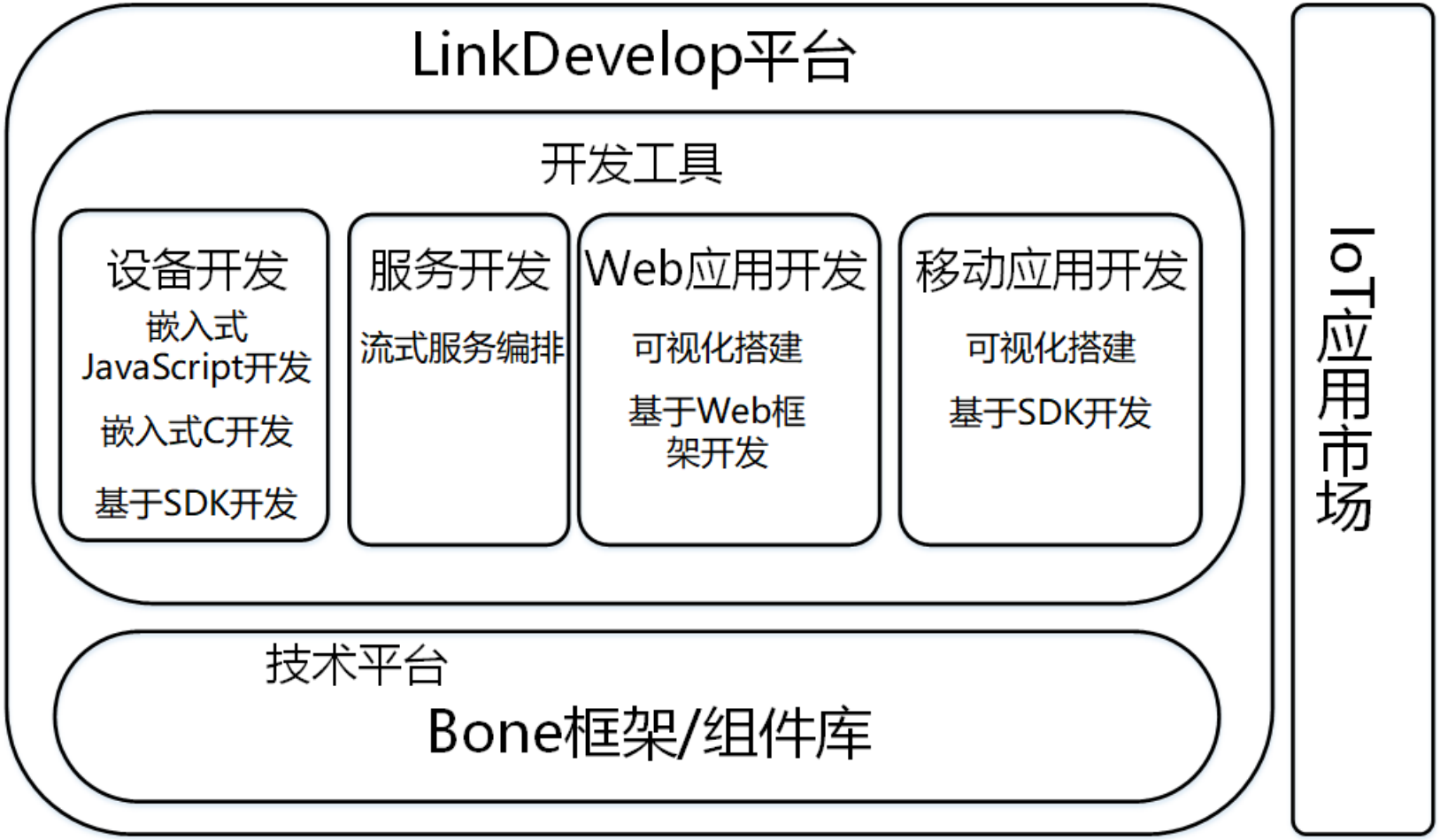
Link Develop 平台概述

- 2018年3月，阿里巴巴宣布进军物联网领域，推出Link Develop 1.0 版本，涵盖物联网应用全链路开发流程；

Link Develop 平台概述

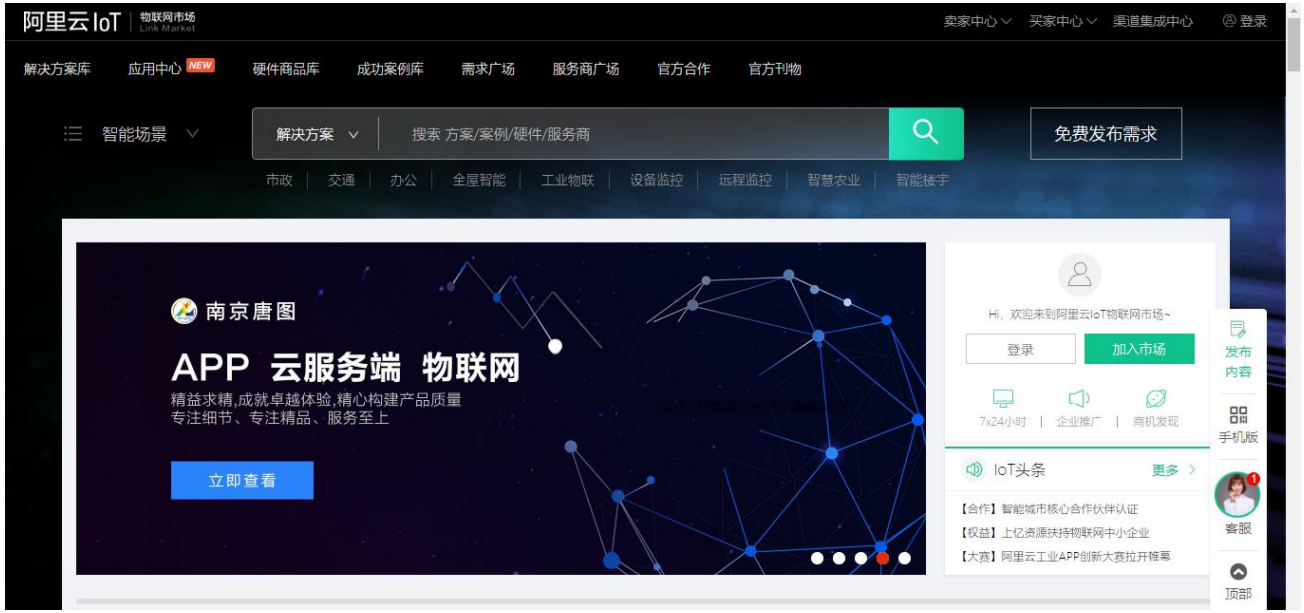
- 2018年6月，阿里巴巴推出Link Develop 2.0 版本，在1.0版本的基础上，集成四大开发工作台：**设备开发工作台、服务开发工作台、Web应用开发工作台、移动应用开发工作台**

Link Develop 系统架构



Link Develop 系统架构

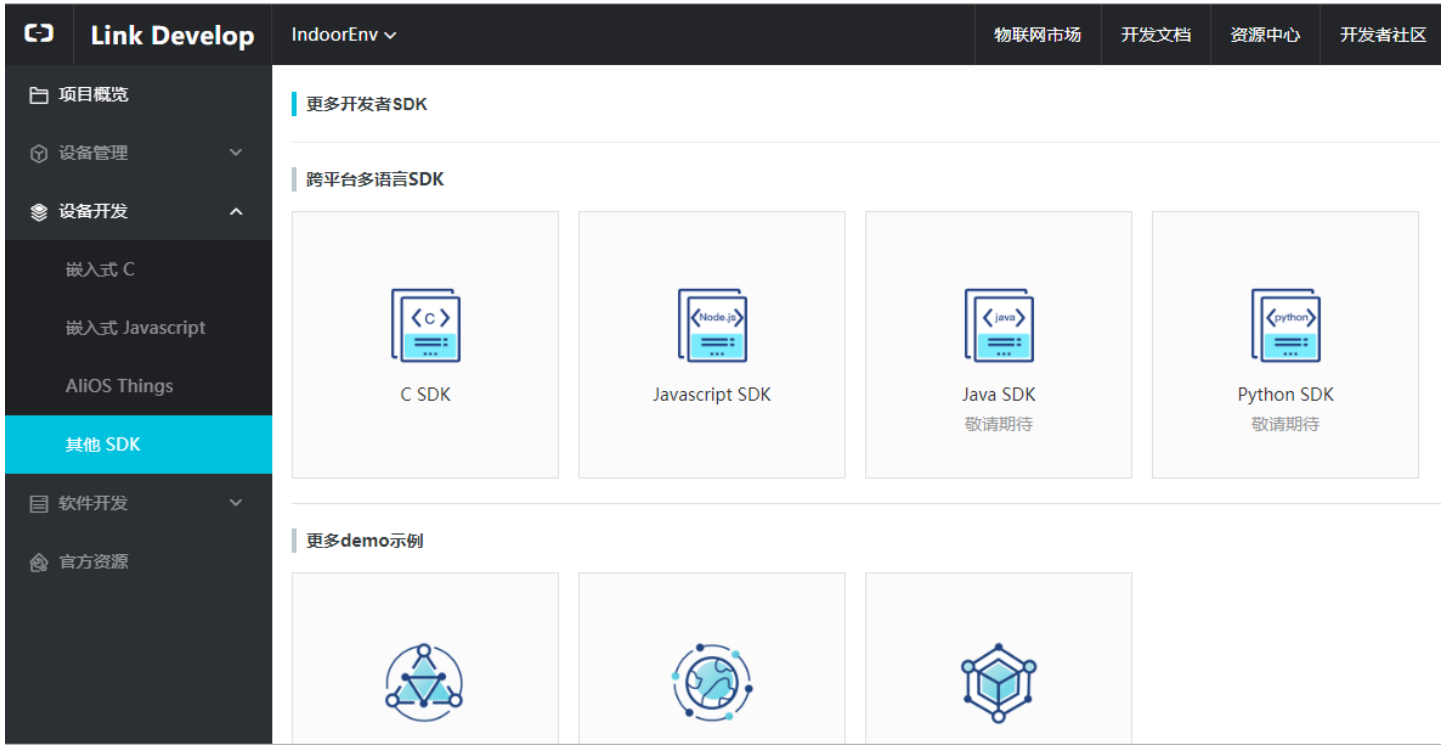
- Bone框架
  - 阿里巴巴推出的Web应用/移动应用开发框架
  - 集成常用的物联网应用UI组件
- IoT应用市场
  - 包含从硬件到解决方案一系列产品，开发者可将自己开发的产品放到IoT应用市场，实现在线商业化



Link Develop 系统架构

❑ 设备开发工作台

- 用于设备端应用的代码编写、应用调试
- 提供嵌入式C开发、嵌入式JavaScript开发、基于AliOS Things的开发以及基于SDK的开发四种开发方式
- 同时支持本地编译、云端编译及基于解释型语言的开发



Link Develop 系统架构

❑ 服务开发工作台

- 用于云端服务的开发
- 支持可视化的服务编排

Link Develop 系统架构

❑ Web应用开发工作台

- ➢ 用于Web应用开发
- ➢ 基于可视化拖拽的Web应用开发
- ➢ 基于Bone框架的Web应用开发

Link Develop 系统架构

- 移动应用开发工作台
  - 用于移动应用开发
  - 基于可视化拖拽的移动应用开发
  - 基于Bone框架的移动应用开发

Link Develop 关键特性

- 设备开发
  - 支持基于SDK的开发、基于物联网操作系统的开发、基于JavaScript语言的开发等多种开发方式
- 云端服务开发
  - 集成服务开发工作台，支持服务编排
- Web应用/移动应用开发
  - 支持基于可视化搭建的Web应用/移动应用开发
- 三端应用集成简化
  - 通过物模型对设备功能进行抽象，简化应用集成

Link Develop 平台

- 概述
- 特点
  - 物模型
  - 流式服务编排
- 开发案例
  - 智能教室座位使用统计系统

Link Develop 平台特点

- 设备开发
  - 基于**物模型**的设备抽象
- 服务开发
  - 支持**流式服务编排**，可与应用无缝集成

Link Develop 平台特点

❑ 物模型

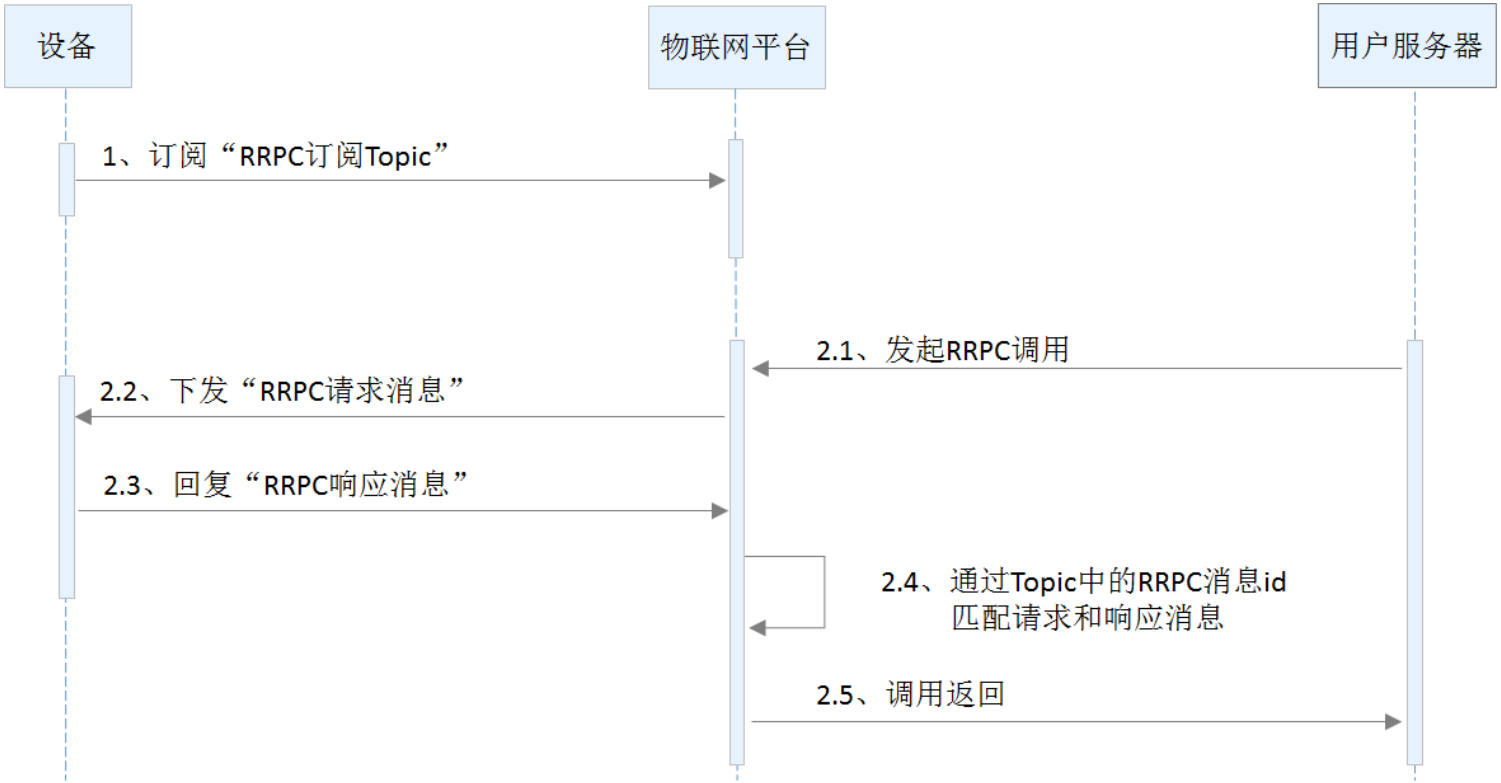
- 由设备端开发者和服务开发者共同定义
- 使用JSON语言，描述一类设备所具有的功能
- 将设备所具有的功能划分为三类

功能类型	功能说明	举例
属性（Property）	设备运行时的状态	环境监测设备测量的温度、湿度等环境数据
服务（Service）	设备可被外部调用的方法	设备向外部提供的打开风扇方法，分为同步调用服务和异步调用服务
事件（Event）	设备运行时的事件，包含需要被外部感知和处理的通知信息	任务完成的信息、设备发生故障的信息

物模型服务同步调用

❑ 同步调用

- MQTT协议提供基于PUB/SUB的异步通信，不适用于服务端同步控制设备端返回结果的场景。
- RRPC：基于MQTT协议定制的请求和响应的同步机制



物模型定义

```
{
    “schema”:“物的TSL描述schema”, 由云平台自动生成
    “link”:“云端系统级uri,用来调用服务/订阅事件”, 由云平台自动生成
    "profile":{
        "productKey":"产品key"
    },
    “properties”:[],          // 属性定义
    “events”:[],   // 事件定义
     services“:[]  // 服务定义
}
```

物模型定义-属性

```
"properties":[ {
        "identifier":"属性唯一标识符(产品下唯一)",
        "name":"属性名称",
        "accessMode":"属性读写类型，只读(r)，读写(rw)",
        "required":"是否是标准功能的必选属性"
        "dataType":{
            "type":"属性类型: int，float，double, text，bool，array(数组类型，支持
int/double/float/text)",
            "specs":{
                "min":"参数最小值(int, float, double类型特有)",
                "max":"参数最大值(int, float, double类型特有)",
                "unit":"属性单位",
                "unitName":"单位名称",
                "size":"数组大小，默认最大128(数组特有)",
                "item":{  "type":"数组元素的类型"  }  }  }
    } ]
```

物模型定义-事件

```
"events":[ {
        "identifier":"事件唯一标识符(产品下唯一)",
        "name":"事件名称",
        "desc":"事件描述",
        "type":"事件类型(info,alert,error)",
        "required":"是否是标准功能的必选事件",
        "outputData":[ {
            "identifier":"参数唯一标识符",
            "name":"参数名称",
            "dataType":{
              "type":"属性类型: int，float，double, text，bool，array(数组类型，支持int/double/float/text)",
                "min":"参数最小值(int, float, double类型特有)",
                "max":"参数最大值(int, float, double类型特有)",
                "unit":"属性单位",
                "unitName":"单位名称",
                "size":"数组大小，默认最大128(数组特有)",
                "item":{ "type":"数组元素的类型“}}}}],
        "method":"事件对应的方法名称(根据identifier生成)"
    }]
```

物模型定义-服务

```
"services":[ {
        "identifier":"服务唯一标识符(产品下唯一)",
        "name":"服务名称",
        "desc":"服务描述",
        "required":"是否是标准功能的必选服务",
        “callType”: “服务调用方式，sync:同步调用；async:异步调用",
        "inputData":[ {
            ......
          } ],
        "outputData":[ {
            ......
          } ],
        "method":"服务对应的方法名称(根据identifier生成)"
      }
   ]
}
```

物模型示例

- 设备描述：智能空调
- 设备功能：
  - 属性：
    - *温度（temperature），float类型，最小值为-40，最大值为50，单位为°C*
  - 事件：
    - *温度告警（temp_alert, 输出值为枚举类型，0代表正常温度，1代表高温报警（温度 > 30 °C））*
  - 服务：
    - *打开空调（open，返回值为bool 类型，0代表打开空调失败，1代表打开空调成功，服务调用方式为同步调用）*

物模型示例

设备属性定义

```
"properties": [ {
    “identifier”: “temp”,    // 属性标识符
    "dataType": {
      “specs”: {             // 属性相关特性定义
        "unit": "°C",
        "min": "-40",
        "max": "50",
        "step": "0.1"
      },
      “type”: “float“        // 数据类型
    },
    “name”: “温度”,          // 属性名称
    “accessMode”: “r”,       // 属性读写类型
    “required”: false        // 是否是标准功能的必选属性
  } ]
```

物模型示例

设备事件定义

```
"events": [ {
    “outputData”: [ {                    // 事件返回参数
      “identifier”: “AlarmType”,         // 参数标识符
      "dataType": {
        "specs": { "0": "恢复正常", "1": "高温报警“},
        “type”: “enum“                   // 属性类型：枚举
      },
      “name”: “报警类型“                 // 参数名称
    } ],
    “identifier”: “TemperatureAlarm”, // 事件标识符
    “method”: “thing.event.TemperatureAlarm.post”,   //事件对应的方法名称
    “name”: “温度报警”,                  // 事件名称
    “type”: “alert”,                     // 事件类型
    “required”: false                    // 是否是标准功能的必选事件
  } ]
```

物模型示例

❑ 设备服务定义

```
"services": [ {
    “outputData”: [ {                          // 服务返回参数
      “identifier”: “successOrNot”,            // 参数标识符
      “dataType”: {                            // 参数类型
        "specs": { "0": "失败", "1": "成功" },
        "type": "bool"
      },
      “name”: “打开空调是否成功“                // 参数名称
    } ],
    “identifier”: “open”,                      // 服务标识符
    “inputData”: [],                           // 入参（此例无入参）
    “method”: “thing.service.open”,            // 服务对应的方法名称
    “name”: “打开空调”,                        // 服务名称
    “required”: false,                         // 是否是标准功能的必选服务
    “callType”: “sync”} ]                      // 服务调用方式，此处为同步
```

课堂练习

- **现在我们要定义一个空气盒子的物模型，模型具体功能信息如下**
  - **属性：**
    - *温度（Temp）： float类型，最小值为-40，最大值为50，单位为°C，步长0.1，可读*
    - *湿度（Humi）： float类型，最小值为0%，最大值为100%，单位为%，步长0.1， 可读*
  - **事件：**
    - *属性数据上报事件（PropPost）：输出参数包含上述两个属性的值，事件对应的方法名称为“thing.event.prop.post”，其余参数自定义即可*

物模型的使用-以智能空调为例

- 设备属性上报
  - Topic: /sys/{ProductKey}/{deviceName}/thing/event/property/post
  - 数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": {
        “temp”:30,
  },
  "method": "thing.event.property.post"
}
```

物模型的使用-以智能空调为例

- 设备事件上报
  - Topic: /sys/{ProductKey}/{deviceName}/thing/event/{dsl.event.identifer}/post
  - 数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    “AlarmType”: 0,
  },
  "method": "thing.event.TemperatureAlarm.post"
}
```

物模型的使用-以智能空调为例

- 设备服务调用
  - Topic: /sys/{ProductKey}/{deviceName}/thing/service/{dsl.service.identifer}
  - 数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": {
  },
  "method": "thing.service.open”
}
```

Link Develop 平台特点

❑ 服务编排

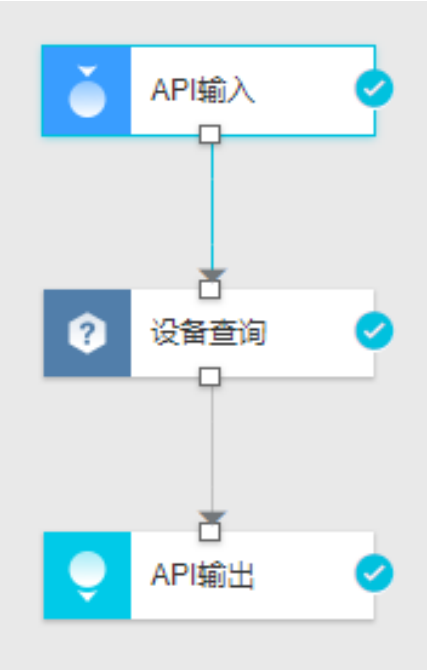
- Link Develop 提供可视化的服务编排服务，通过将不同类型、不同功能的服务节点进行组合，得到实现特定功能的API
- 两种类型的服务
  - *API服务：**可被Web应用/移动应用调用的API***
  - *设备触发服务：**设备云端场景的联动，表现形式为可被设备属性/事件变更触发的云端规则。***

服务编排示例

- 案例一描述：使用Link Develop服务编排功能，开发API服务，读取指定设备最新上传的多类数据
  - 步骤一：根据节点功能，选择需要的节点。本案例选择3个节点，分别是API输入节点、API输出节点以及设备查询节点

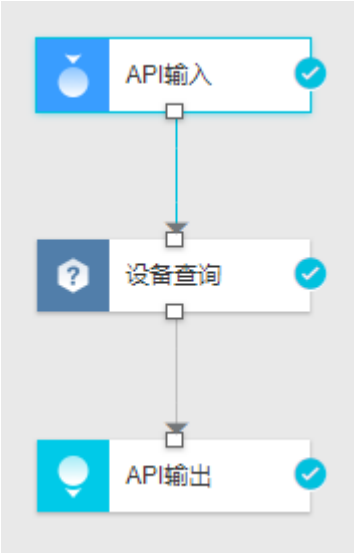
服务编排示例

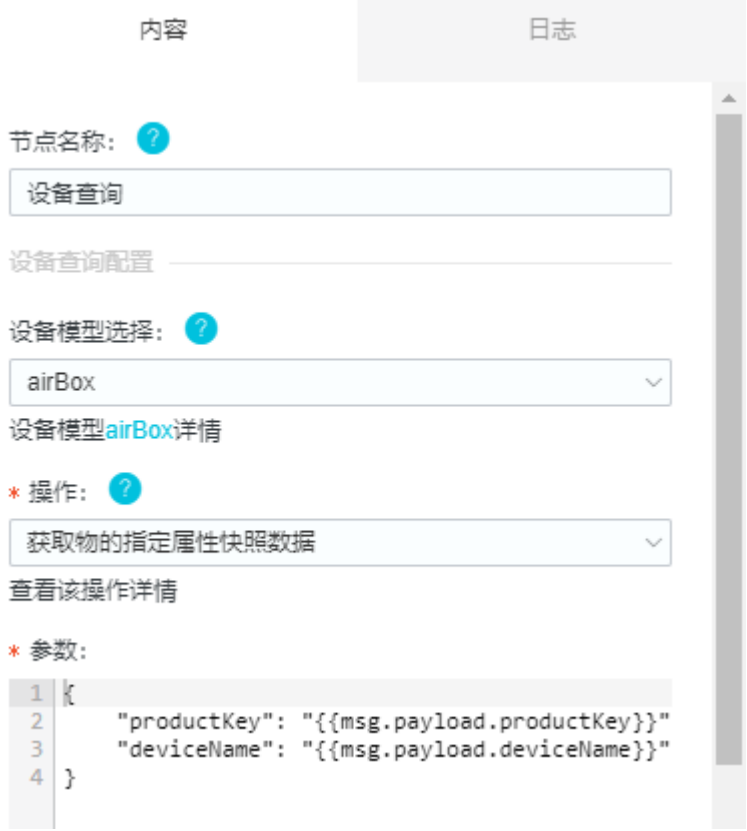
- 步骤二：将节点按照应用逻辑进行连接，并对节点进行配置
  - *API输入节点：增加两个参数，分别是deviceName 和 produckKey，用于指定查询的设备*



服务编排示例

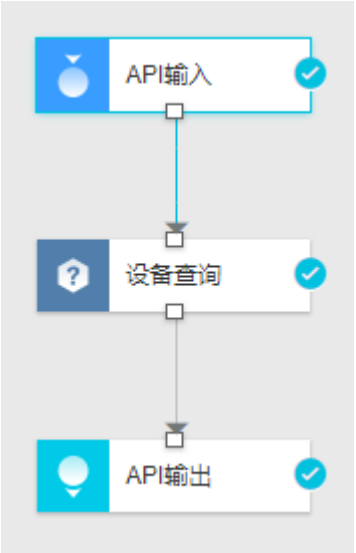
- 步骤二：将节点按照应用逻辑进行连接，并对节点进行配置
  - *设备查询节点：选择设备查询节点的操作为“获取物的指定属性快照数据”，并获得API输入节点的两个参数*





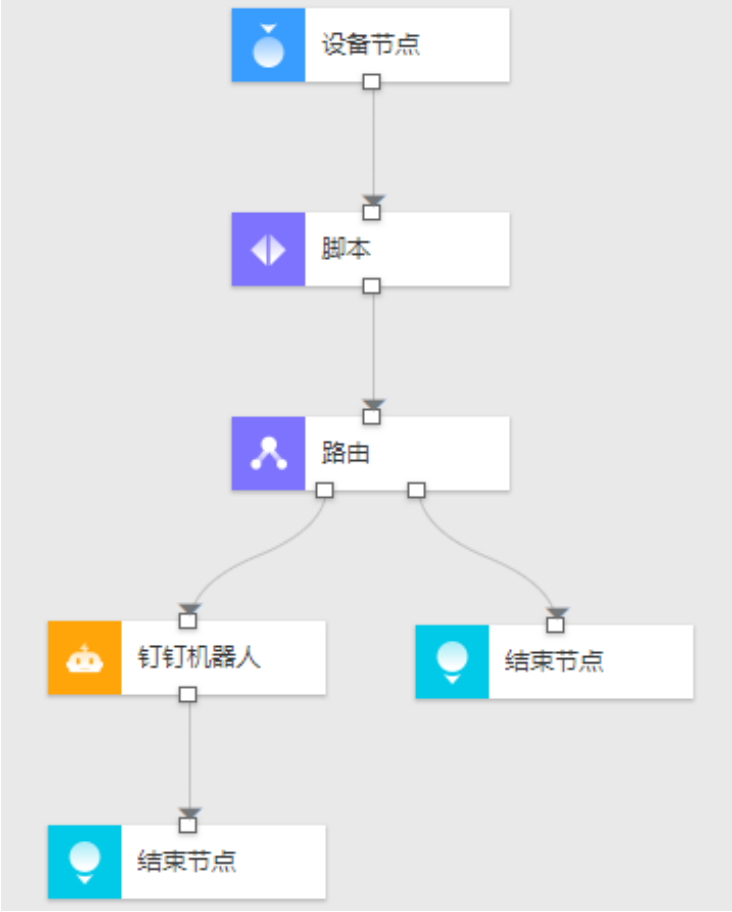
服务编排示例

- 步骤二：将节点按照应用逻辑进行连接，并对节点进行配置
  - *API输出节点：节点配置采用默认即可，将上一节点输入的数据全部输出*



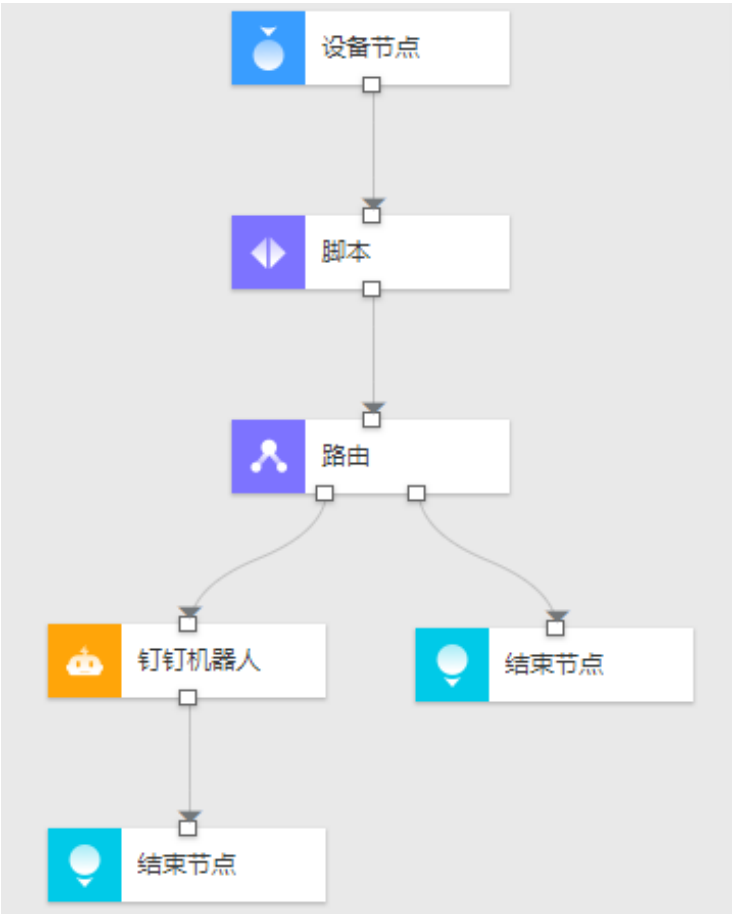
服务编排示例

- 案例二描述：使用Link Develop服务编排功能，开发设备触发服务，当温度大于30℃，将信息通过钉钉机器人发送到钉钉群
  - 步骤一：根据节点功能，选择需要的节点。本案例选择5种节点，分别是设备节点、脚本节点、路由节点、钉钉机器人节点以及结束节点



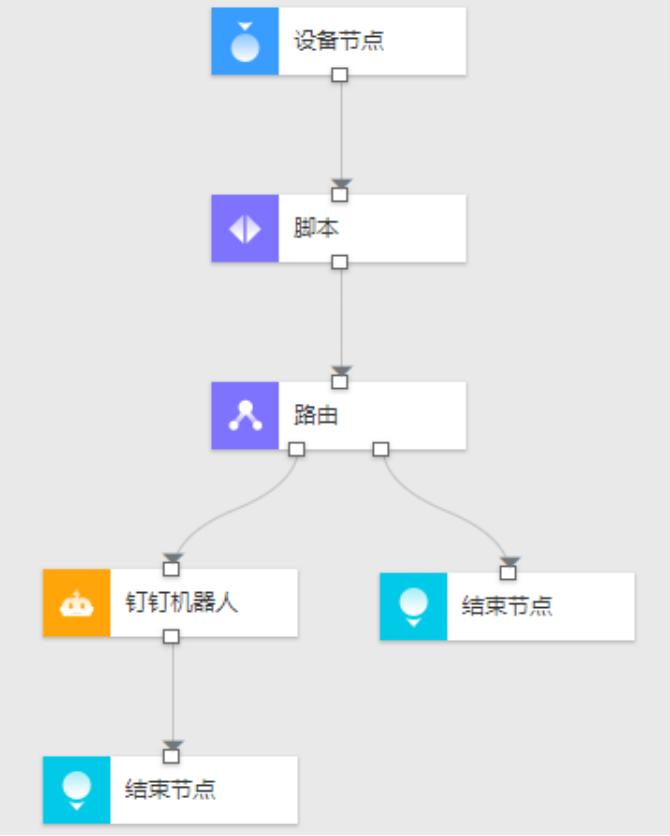
服务编排示例

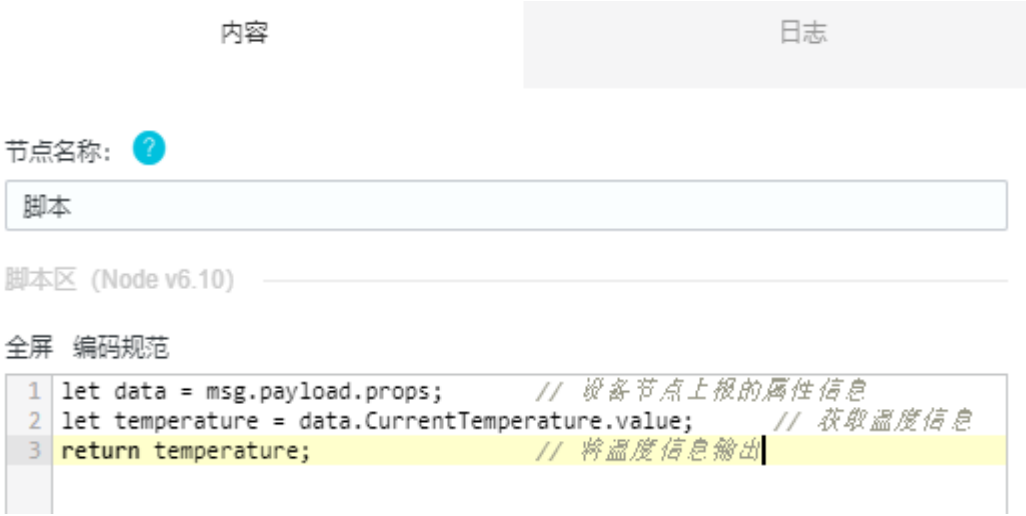
- 步骤二：将节点按照应用逻辑进行连接，并对节点进行配置
  - *设备节点：指定设备模型、设备、以及上报类型。*



服务编排示例

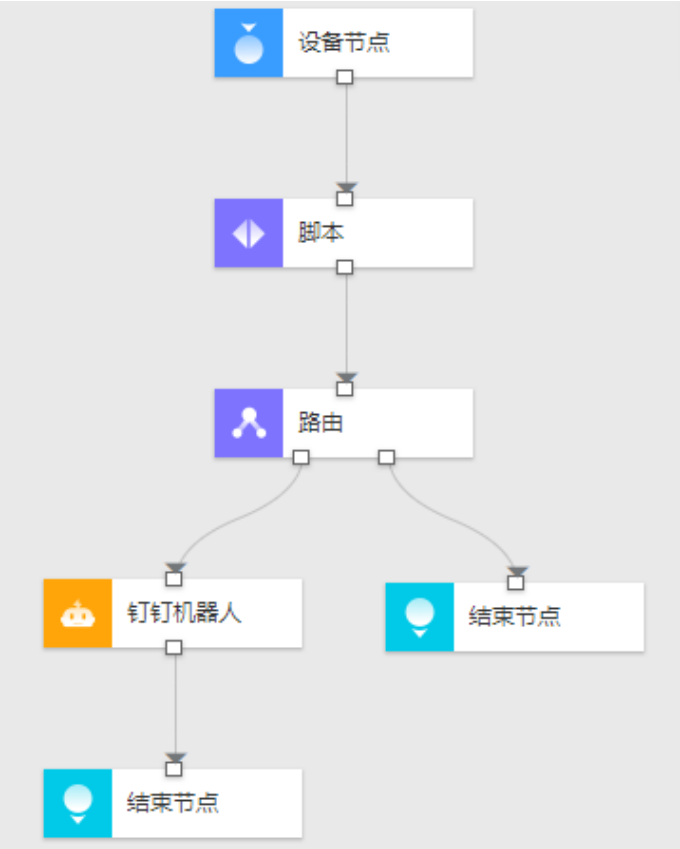
- 步骤二：将节点按照应用逻辑进行连接，并对节点进行配置
  - *脚本节点：编写NodeJS 代码指定节点逻辑，获取设备上报的属性中的温度信息。*





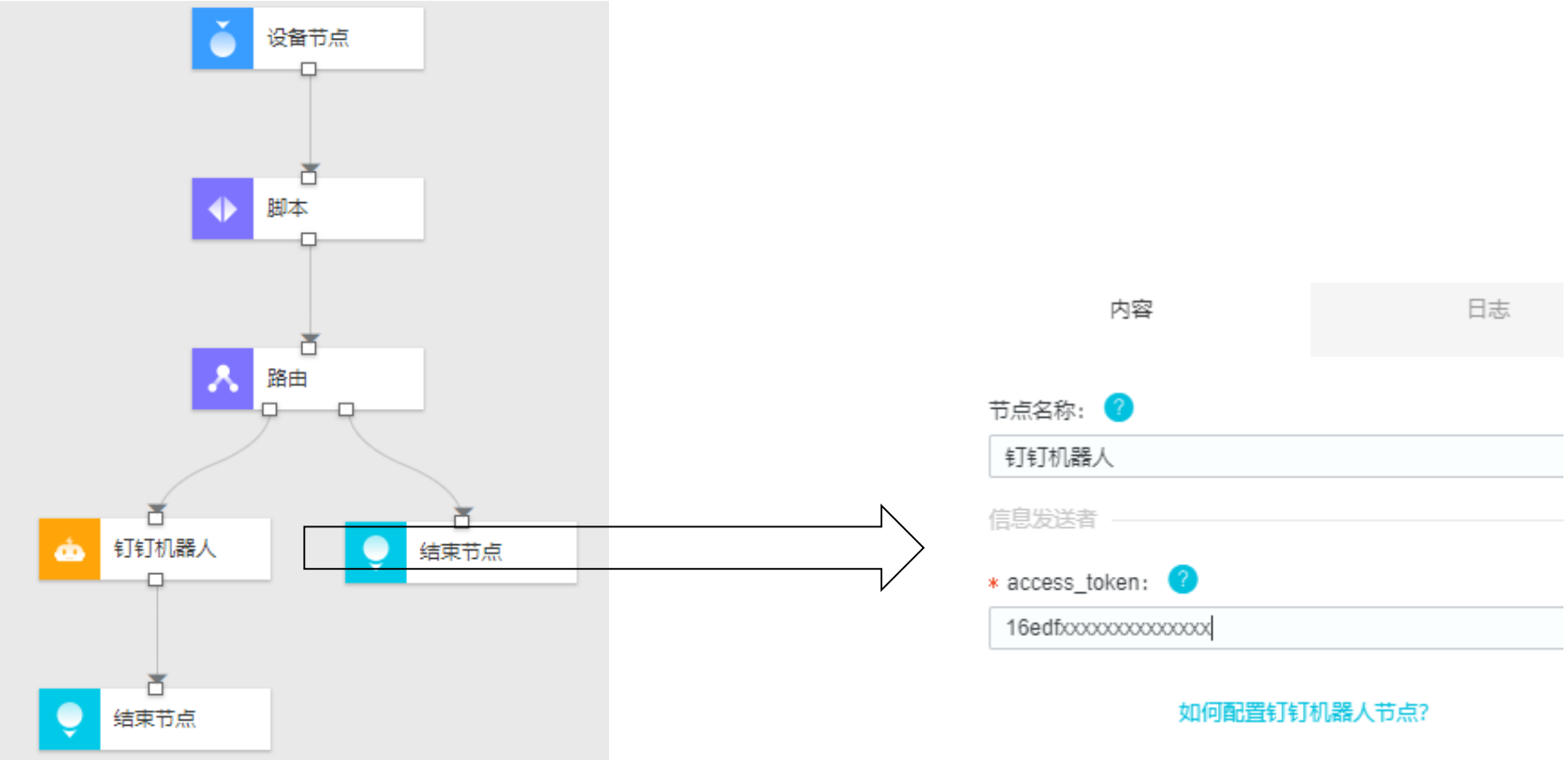
服务编排示例

- 步骤二：将节点按照应用逻辑进行连接，并对节点进行配置
  - *路由节点：配置路由节点，根据温度是否大于等于30°C，若温度大于等于30 °C，输出到钉钉机器人节点，否则，结束*



服务编排示例

- 步骤二：将节点按照应用逻辑进行连接，并对节点进行配置
  - *钉钉机器人节点：配置钉钉机器人节点的access_token, access_token的获取参考钉钉文档。*



服务编排示例

➢步骤三：部署应用，服务逻辑是否正确

Link Develop 提供的服务节点

服务节点名称	服务节点描述
API输入节点	API流的起始节点，用于API入参定义
API输出节点	用于API流的结果返回
设备节点	设备触发流的起始节点，用于设备触发流输入信息定义
结束节点	标志设备触发流的结束
路由节点	分支节点，一个输入，多个输出，实现if-else的应用逻辑
脚本节点	自定义数据处理逻辑，可编写自定义nodeJS代码
RDS节点	阿里云关系型数据库操作节点，支持数据库增删改查操作
OTS节点	阿里云结构化数据库操作节点，支持数据库增删改查操作
设备操作节点	阿里云物的管理服务的封装，提供物的上报数据查询、获取物的详情等功能
设备查询节点	阿里云设备查询服务的封装，提供查询设备列表、查询设备上报数据等功能
钉钉机器人节点	用于消息推送至钉钉群

Link Develop 平台

- 概述
- 特点
  - 物模型
  - 流式服务编排
- 开发案例
  - 智能教室座位使用统计系统

Link Develop 开发案例

- 智能教室座位使用统计系统：设备部署在教室，可自动监测座位上是否有人，并在Web应用和移动应用上实时显示。
- 需要的设备：
  - 树莓派3B+
  - 红外热释电运动传感器

Link Develop 开发案例

- 步骤1：进入Link Develop工作台（https://linkdevelop.aliyun.com/createProject），创建空白项目

Link Develop 开发案例

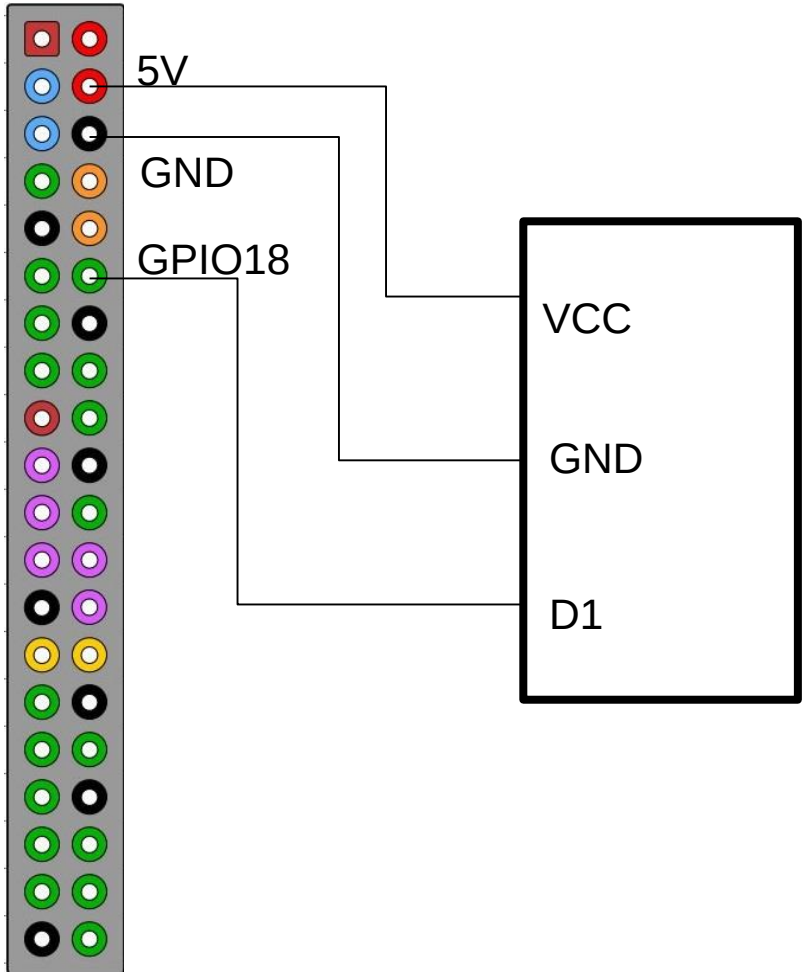
- 步骤2：创建设备模型和设备。本例中，我们为设备模型添加两个属性，分别是设备编号（deviceID）和红外检测状态（motionState）

自定义功能

功能类型	功能名称	标识符	数据类型	数据定义
属性	红外检测状态	motionState 译	bool	布尔值：无人 - 0；有人 - 1；
属性	设备编号	deviceID	int32	取值范围：0 ~ 1000

Link Develop 开发案例

- 步骤3：硬件连接。将树莓派和红外热释电运动传感器进行连接，引脚对应关系如下。



Link Develop 开发案例

- 步骤4：SDK安装。在项目目录下运行如下命令，安装所需模块。

```
npm install --save aliyun-iot-device-sdk pigpio
```

  - aliyun-iot-device-sdk: 云平台接入SDK
  - Pigpio：树莓派GPIO相关接口

Link Develop 开发案例

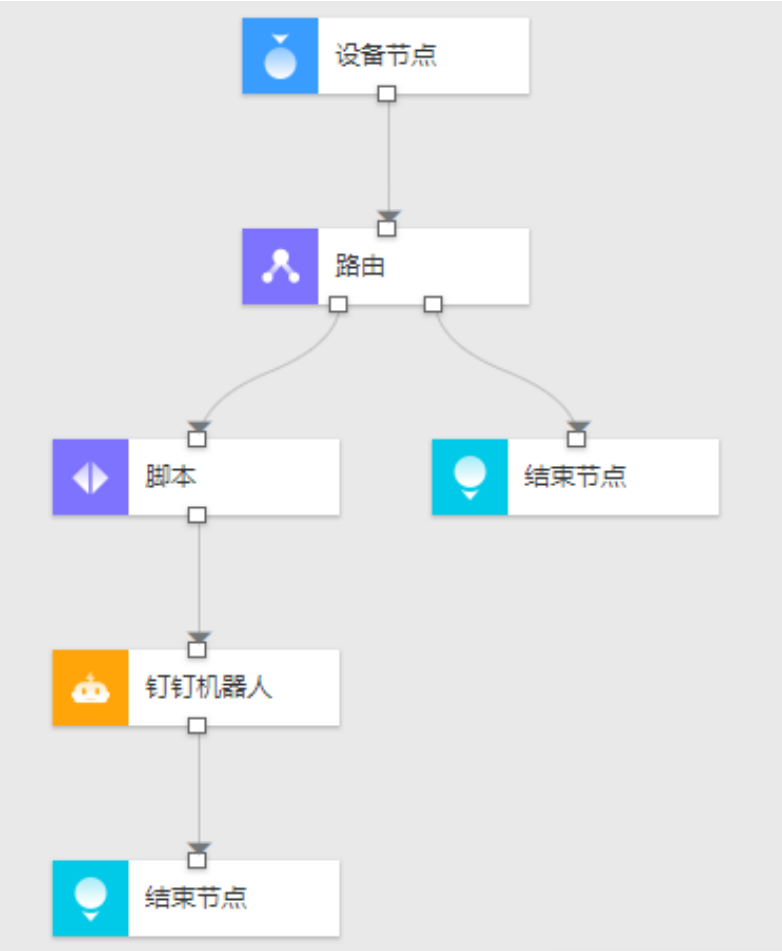
❑ 步骤5：设备端代码编写，代码如下。

```
const aliyunIot = require('aliyun-iot-device-sdk');
const Gpio = require('pigpio').Gpio;
const device_id = 0;
const gpio_pir = new Gpio(18, {
    mode: Gpio.INPUT,
});

// 创建设备实例
const device = aliyunIot.device({
  // 激活凭证 这里替换成你自己上一步申请到的激活凭证
  productKey: ‘xxx',
  deviceName: ‘xxx',
  deviceSecret: ‘xxx'
});
// 读取传感器数值
function getMotionState = function() {
  return gpio_pir.digitalRead();
}
```

```
// 上传数据
function postProps() {
    device.postProps({
        deviceID: device_id,
        motionState: getMotionState()
    });
    console.log('post');
}
// 定义定时器，周期上传数据
var myInterval = setInterval(postProps, 10000);

device.on('connect', () => {
    console.log('connect successfully');
});
```

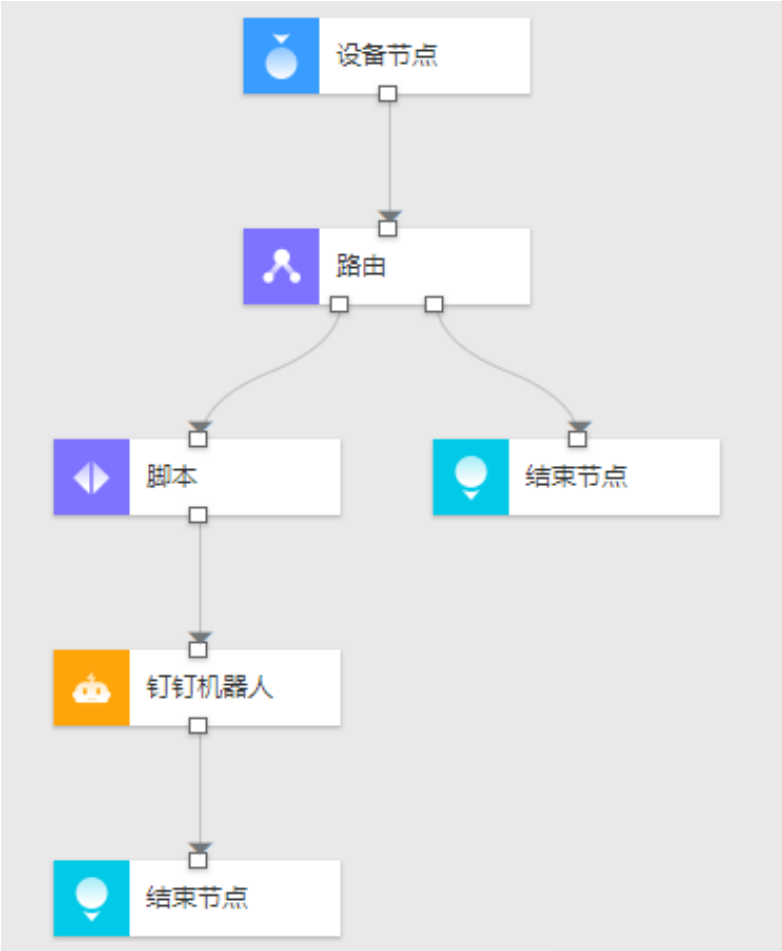
Link Develop 开发案例

- 步骤6：进入 Link Develop 项目，新建自定义设备触发服务流，编排节点如下所示。



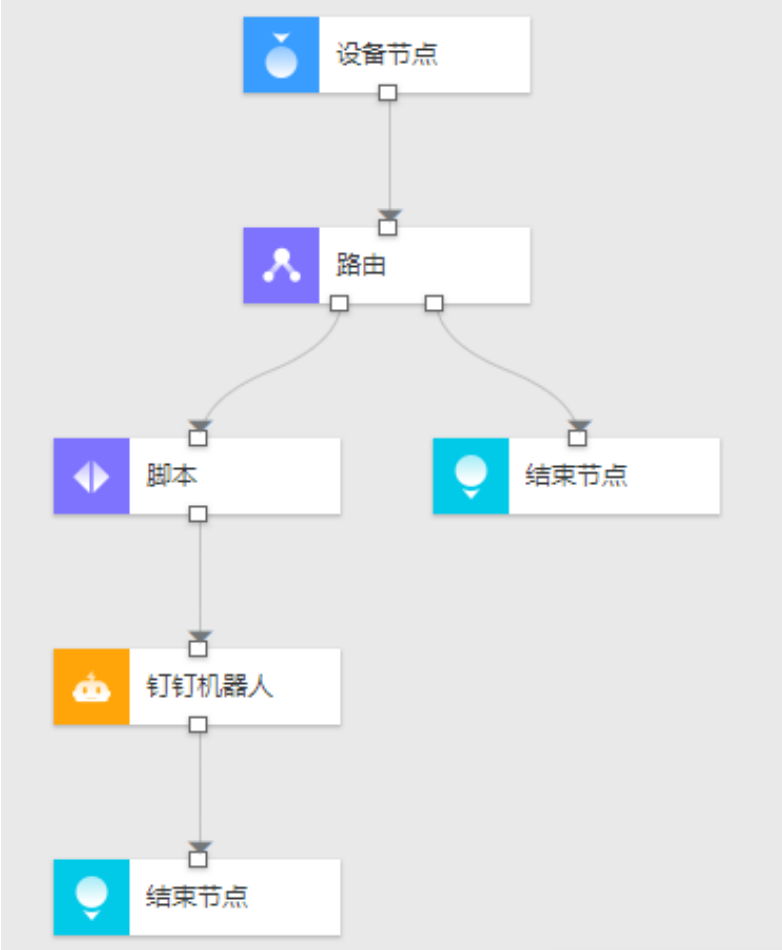
Link Develop 开发案例

- 步骤7：对节点进行配置。设置设备节点的设备模型以及上报类型



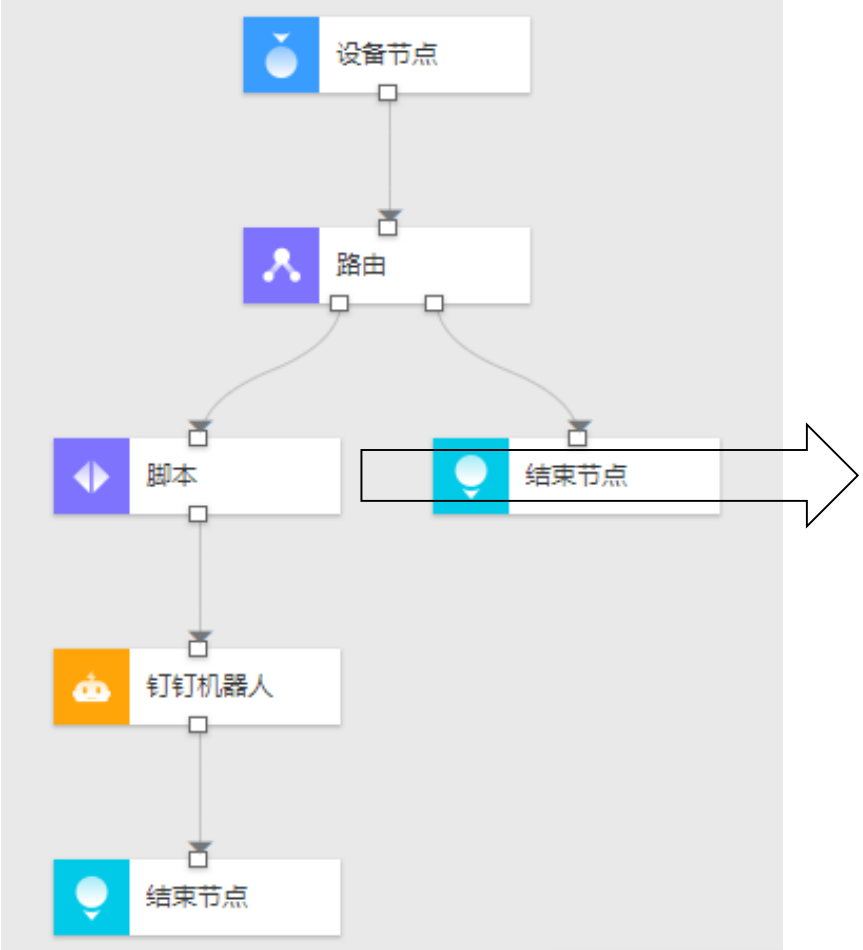
Link Develop 开发案例

- 步骤7：对节点进行配置。路由节点判断设备是否检测到有人，若有人则将数据传到钉钉，否则，服务结束



Link Develop 开发案例

- 步骤7：对节点进行配置。设置脚本节点的数据处理逻辑，提取设备上传的数据，得到设备编号



内容　　日志

节点名称：

脚本

脚本区（Node v6.10）

全屏　编码规范

```
// 获取设备编号
let device_id = msg.payload.props.deviceID.value;

// 获取传感器信息
let motion_state = msg.payload.props.motionState.value;

// 将信息组合
let result = 'device ID: ' + device_id + ' online';

return result;
```

Link Develop 开发案例

- 步骤8：进入 Link Develop 项目，新建基于可视化搭建的Web应用

Link Develop 开发案例

- 步骤9：使用指示灯、文本框、文字时钟等组件搭建Web页面

Link Develop 开发案例

❑ 步骤10：配置指示灯组件的数据源

Link Develop 开发案例

- 步骤11：在Web应用开发控制台，点击页面预览，查看Web页面效果

Link Develop 设备端开发的不足

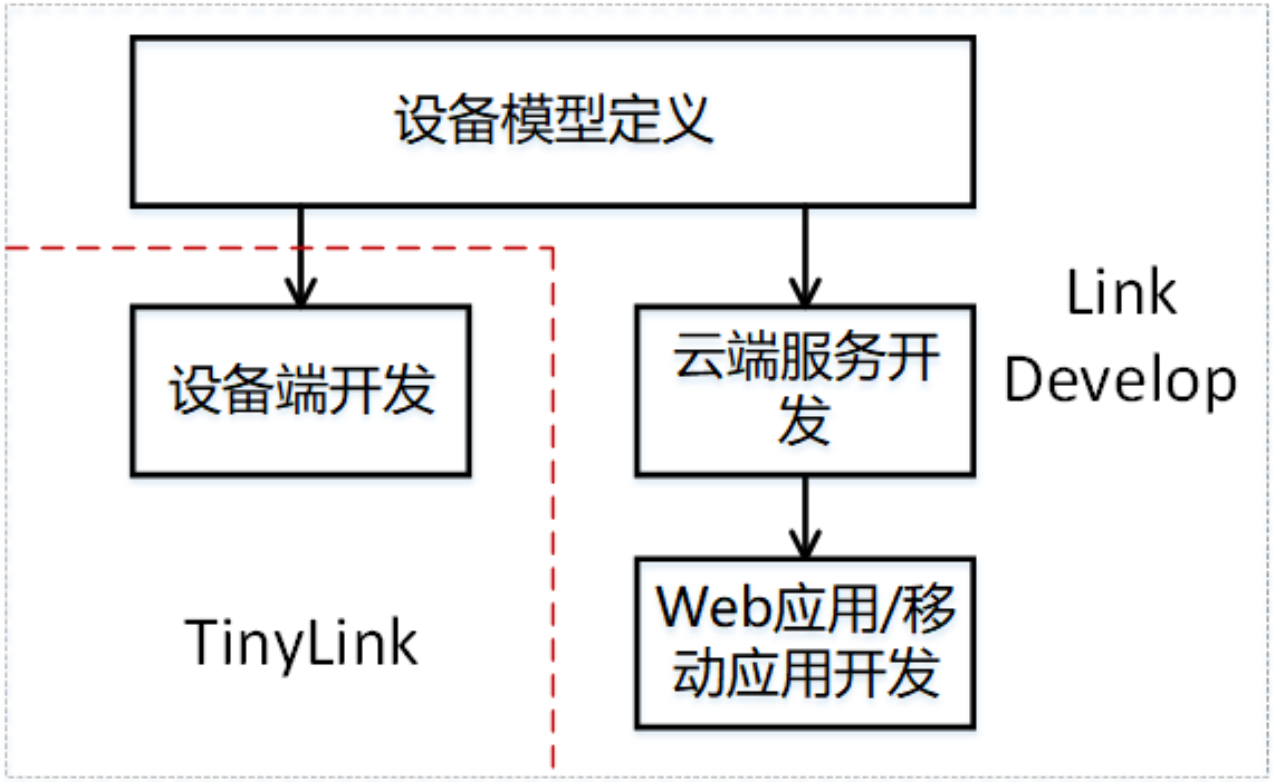
- 针对设备端开发，目前Link Develop支持的硬件如下（以JavaScript开发为例）

芯片型号	厂商	操作系统
ESP32	乐鑫	AliOS-Things, FreeRTOS, UCOSII
STM32F/L4XX	意法半导体	AliOS-Things, FreeRTOS, UCOSII
EMW3060/EMW3080	庆科	AliOS-Things
STM32F103X	意法半导体	UCOSII
STM32F7XX	意法半导体	AliOS-Things, FreeRTOS, UCOSII

- Link Develop在设备端开发上的不足
  - 支持的硬件种类较少
  - 仍采用传统的自底向上的开发方式

Link Develop + TinyLink 开发流程

- 使用TinyLink实现设备端开发，云端服务开发和Web应用/移动应用开发由Link Develop完成



- 关键点：使用TinyLink实现基于物模型的设备接入

Link Develop+TinyLink 设备接入

❑使用MQTT协议连接阿里云，连接参数如下

参数名称	参数值
serverName	${productKey}.iot-as-mqtt.cn-shanghai.aliyuncs.com
port	1883
clientName	clientId+“|securemode=x,signmethod=xxxxxx,timestamp=xxxxxx|”，其中，clientId由用户自定义，64字符以内；timestamp表示当前时间；signmethod代表签名算法类型，支持hmacSha1，hmacSha256，hmacMd5；securemode代表目前安全模式，可取2（TLS直连模式），3（TCP直连模式）
userName	deviceName+"&"+productKey
passWord	sign_hmac(deviceSecret, content)，其中，content =（productKey, deviceName, timestamp, clientId）

Link Develop+TinyLink设备接入示例

❑假设设备相关属性如下

clientId	deviceName	productKey	tinestamp	signmethod	deviceSecret
12345	device	pk	789	hmacsha1	secret

❑设备使用MQTT连接Link Develop参数

参数名称	参数值
serverName	pk.iot-as-mqtt.cn-shanghai.aliyuncs.com
port	1883
clientName	12345|securemode=3,signmethod=hmacsha1,timestamp=789|
userName	device&pk
passWord	hmacsha1("secret","clientId12345deviceNamedeviceproductKeypktimestamp789").toHexString() = FAFD82A3D602B37FB0FA8B7892F24A477F851A14

Link Develop+TinyLink设备开发示例

- 示例描述：基于TinyLink系统，开发智能教室座位使用统计系统设备端，接入Link Develop

```
TL_MQTT mqtt;
// 设置MQTT参数
int port = 1883;
char serverName[]= “pk.iot-as-mqtt.cn-shanghai.aliyuncs.com";
char clientName[] = "12345|securemode=3,signmethod=hmacsha1,timestamp=789|";
char userName[] = "device&pk";
char password[] = "FAFD82A3D602B37FB0FA8B7892F24A477F851A14";
char propertyPostTopic[] = "/sys/pk/device/thing/event/property/post";

unsigned int messageID = 0;    // 消息ID， 自增
unsigned int deviceID = 0;     // deviceID，默认为0

void setup() {
    TL_Serial.begin(9600);
    TL_WiFi.init();                    // 连接WiFi
    bool wifi_conn = TL_WiFi.join("EmNets-301", "eagle402");
    mqtt = TL_WiFi.fetchMQTT();        // 连接MQTT
    int mqtt_conn = mqtt.connect(serverName, port, clientName, userName, password);
}
```

Link Develop+TinyLink设备开发示例

```
void loop() {
    int motionState = TL_PIR.read();        // 读取传感器数据
    char buf[150];
    String payload = “{\”id\“:\”“;          // 根据物模型，构造数据包
    payload += messageID;
    payload += "\"";
    payload += ",\"version\":\"1.0\"";
    payload += ",\"params\":{\"deviceID\":";
    payload += deviceID;
    payload += ",\"motionState\":";
    payload += motionState;
    payload += "},\"method\":\"thing.event.property.post\"}";
    int payloadlength = payload.length();
    payload.toCharArray( buf, 150 );
    mqtt.publish(propertyPostTopic, buf, payloadlength);  // 数据包发送到指定
Topic
    messageID++;
    TL_Serial.println(buf);
    mqtt.yield(4000);
}
```

目录

- 概述
- 云-端一体开发平台主要功能
- 典型物联网云-端一体开发平台
  - TinyLink 2.0
  - 三星 SmartThings
  - Link Develop
- 总结

总结

- 物联网云-端一体开发平台涵盖了设备开发、云服务开发、Web应用/移动应用开发等物联网应用开发的全链路流程
- 本章涉及三个主流的物联网云平台，TinyLink 2.0，三星Smart Things，Link Develop，三个平台的比较如下

平台名称	设备开发	服务开发	Web应用/移动应用开发	特点
TinyLink 2.0	采用硬件无关的TinyLink 语言，支持云端编译	支持基于IFTTT规则的服务开发	基于设备端代码，自动生成Web应用/移动应用框架，开发难度最低	开发简单，适合物联网技术的初学者
三星SmartThings	传统的嵌入式开发	使用基于生命周期的服务开发方式	基于插件的Web应用/移动应用开发，在降低开发难度的同时保持灵活性	设备端开发支持较少，专注智能家居应用
Link Develop	提供基于SDK、基于AliOS-Things、基于解释型语言多种设备端开发方式	提供基于可视化服务编排的服务开发方式	提供可视化搭建的Web应用/移动应用开发方式，适合初学者	支持多种开发方式，开发更为灵活

浙江大学
Zhejiang University

谢谢！

Email: dongw@zju.edu.cn