浙江大学
Zhejiang University

# 第七章：物联网云平台

董 玮

浙江大学

# 目录

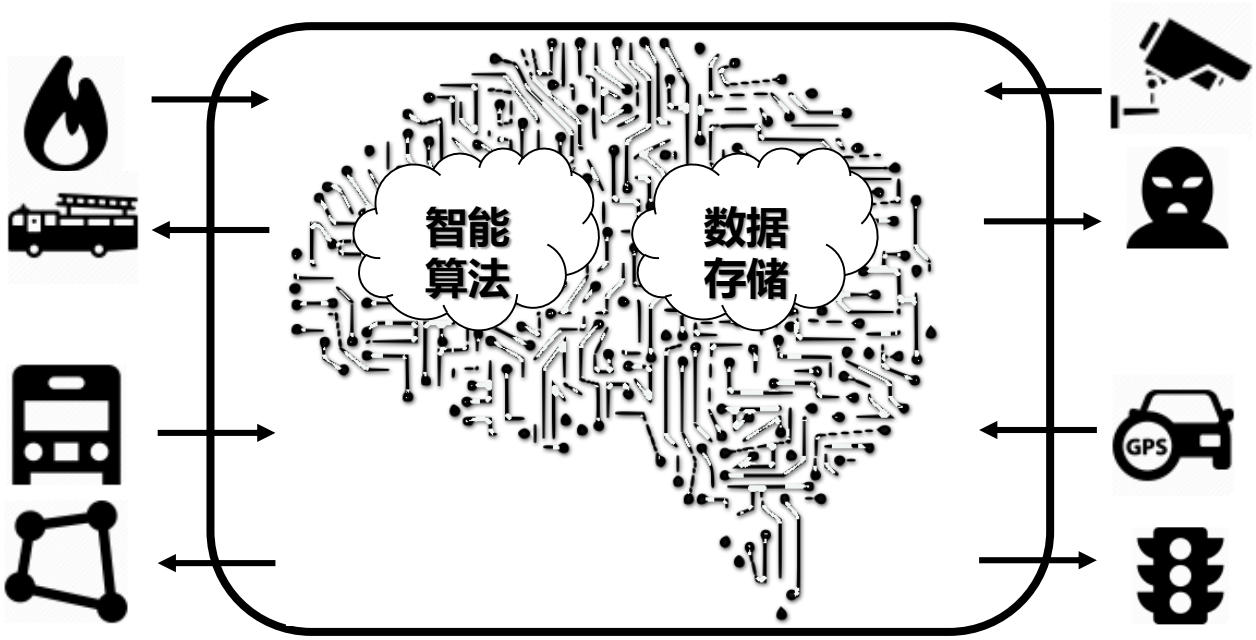

- 概述
- 物联网云平台功能概述
- 典型物联网云平台
- 总结与展望

# 目录


- 概述
- 物联网云平台功能概述
- 典型物联网云平台
- 总结与展望

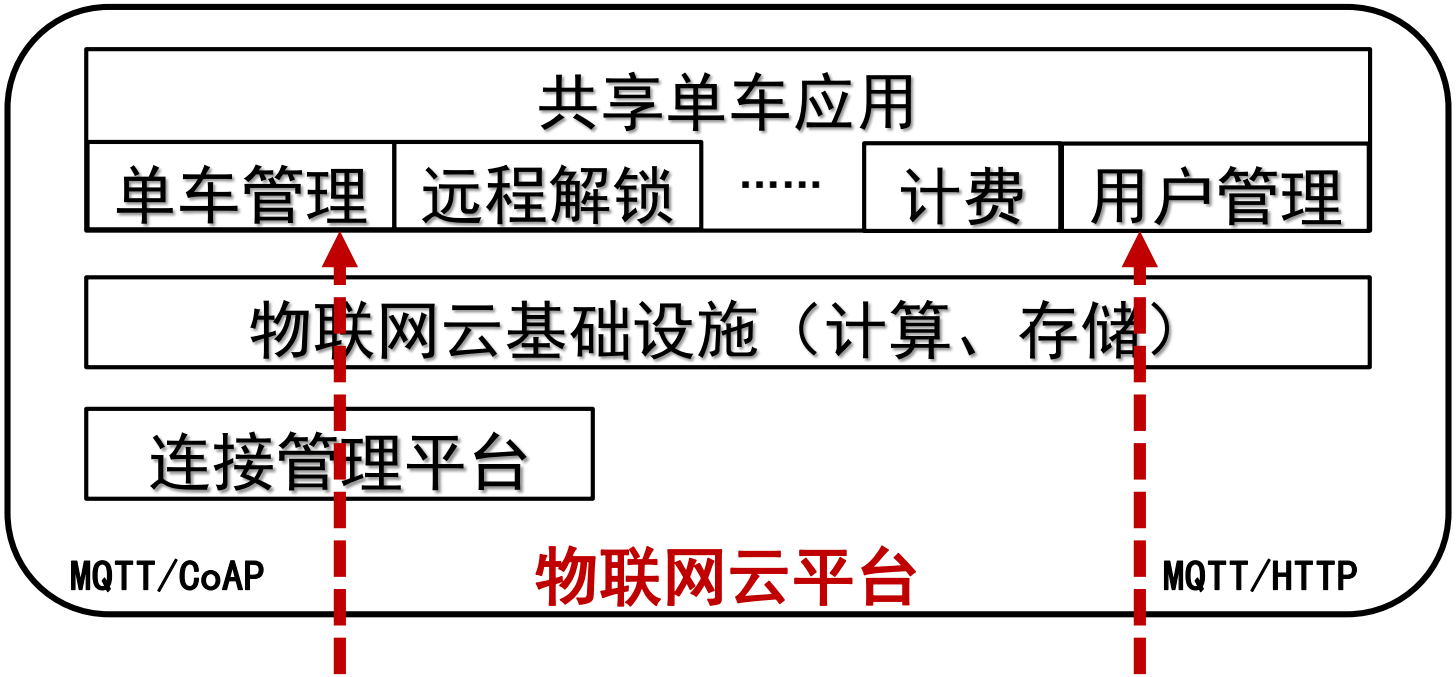
# 云平台是物联网应用的关键一环

- **城市大脑**
  - 云平台通过收集和智能分析从城市各处采集到海量异构数据，来优化城市运行状态(如控制交通信号灯提高通行效率)、预防破坏性事件 (如火情预警、人脸识别追踪罪犯) 和辅助政府的决策。



# 云平台是物联网应用的关键一环

- 共享单车
  - 通过云平台实现车辆管理、计时计费等功能



# 物联网云平台与传统云平台的区别

- **设备的连接**
  - 需要以轻量级的方式，支持多种异构IoT设备连接
- **设备的管理**
  - 需要管理大量、处于各种不同状态的IoT设备
- **数据的存储**
  - 需要存储、管理大量时序数据
- **安全认证**
  - IoT设备容易被攻击，需要更好的安全认证
- **支持高并发的计算模式**
  - IoT设备数量巨大，高并发的计算模式变得更加重要。

# 物联网云平台成为各公司的重点发展方向

- 通信运营商
  - 中国移动OneLink
- 设备提供商
  - 庆科云
  - 海尔优家
- 互联网/云服务厂商
  - Amazon – AWS IoT
  - 阿里巴巴 – 阿里云 IoT
- 初创公司
  - 机智云
  - AbleCloud

# 物联网云平台分类

- 根据功能与业务侧重点，现有的物联网云平台可分为以下4类:

- 连接管理平台CMP(Connectivity Management Platform)
- 设备管理平台DMP(Device Management Platform)
- 应用使能平台AEP(Application Enablement Platform)
- 业务分析平台BAP(Business Analytics Platform)

- 注意: 以上四类并非相互排斥，一个物联网云平台可能会提供多类服务

# 连接管理平台CMP

- **主要功能：**
  - 网络资源管理、SIM卡管控、连接资费管理、套餐管理、网络用量管理、账单管理、故障管理等
- **代表性平台：**
  - 国外：爱立信-DCP、沃达丰-GDSP
  - 国内：中国移动OneLink
- **案例：中国移动OneLink**
  - 如我们刚才提到的共享单车，每个单车是如何接入网络的呢？
  - 目前一般都使用移动提供的物联网卡，而移动的OneLink平台也可以向单车管理公司提供获取卡状态信息(单车是否在线) 、基于网络的定位服务(用来追踪车辆)和流量使用情况(用来计算共享单车的网络成本) 等接口

# 设备管理平台DMP

## ❑ 主要功能：

- ➢ 物联网设备管理，如配置、重启、关闭、恢复出厂、升级/回退等，设备现场产生的数据的查询、报警功能，设备生命周期管理等

## ❑ 代表性平台：

- ➢ IBM – Watson、GE – Predix、三一重工-根云、海尔-U+ 等

## ❑ 案例：海尔家居设备使用海尔U+入网并实现智能化

- ➢ 智能洗衣机：根据放入的衣物的材质、颜色提示水温和设定洗涤模式
- ➢ 智能空气管家：根据室内外温度与空气质量智能选择温度与净化模式
- ➢ 智能热水器：可通过云平台智能调整峰谷用电，在闲时加热水并存储

# 应用使能平台AEP

- **功能：**
  - 为应用开发者提供了成套开发工具、中间件、业务逻辑引擎、API接口、应用服务器等工具，以节省大量开发时间。

- **代表性平台：**
  - 阿里巴巴-阿里云IoT，AWS - IoT，机智云等

- **案例：阿里云IoT的Link Develop简化物联网开发**
  - 设备端、服务、移动端一站式开发
  - 可视化搭建Web和移动前端
  - 可视化流式服务编排，方便用户实现云端逻辑

# 业务分析平台BAP

- **功能：**
  - 将沉淀在平台上结构化和非结构化数据智能分析处理，形成具有预测性、可认知的业务分析逻辑

- **代表性平台： Azure IoT、阿里云+西门子MindSphere**

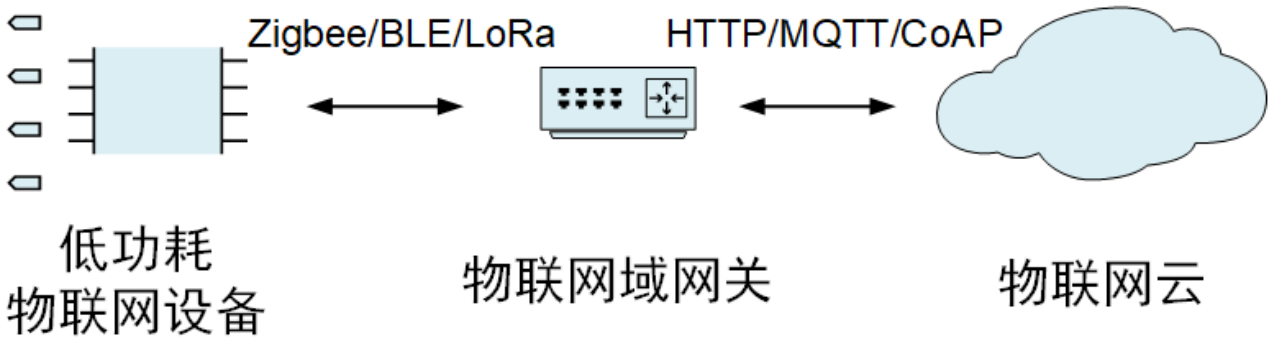
- **<u>目前BAP平台发展仍未成熟</u>**

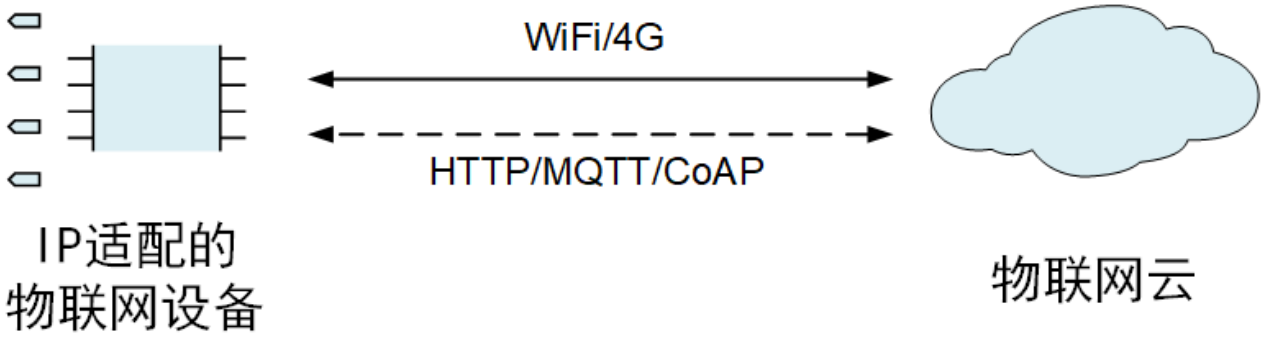
# 目录


- 概述
- 物联网云平台功能概述
  - 设备连接
  - 设备管理
  - 规则引擎
  - 时序数据库/流计算
  - 设备与权限认证
  - 无服务器计算/边缘计算
- 典型物联网云平台
- 总结与展望

# 设备连接

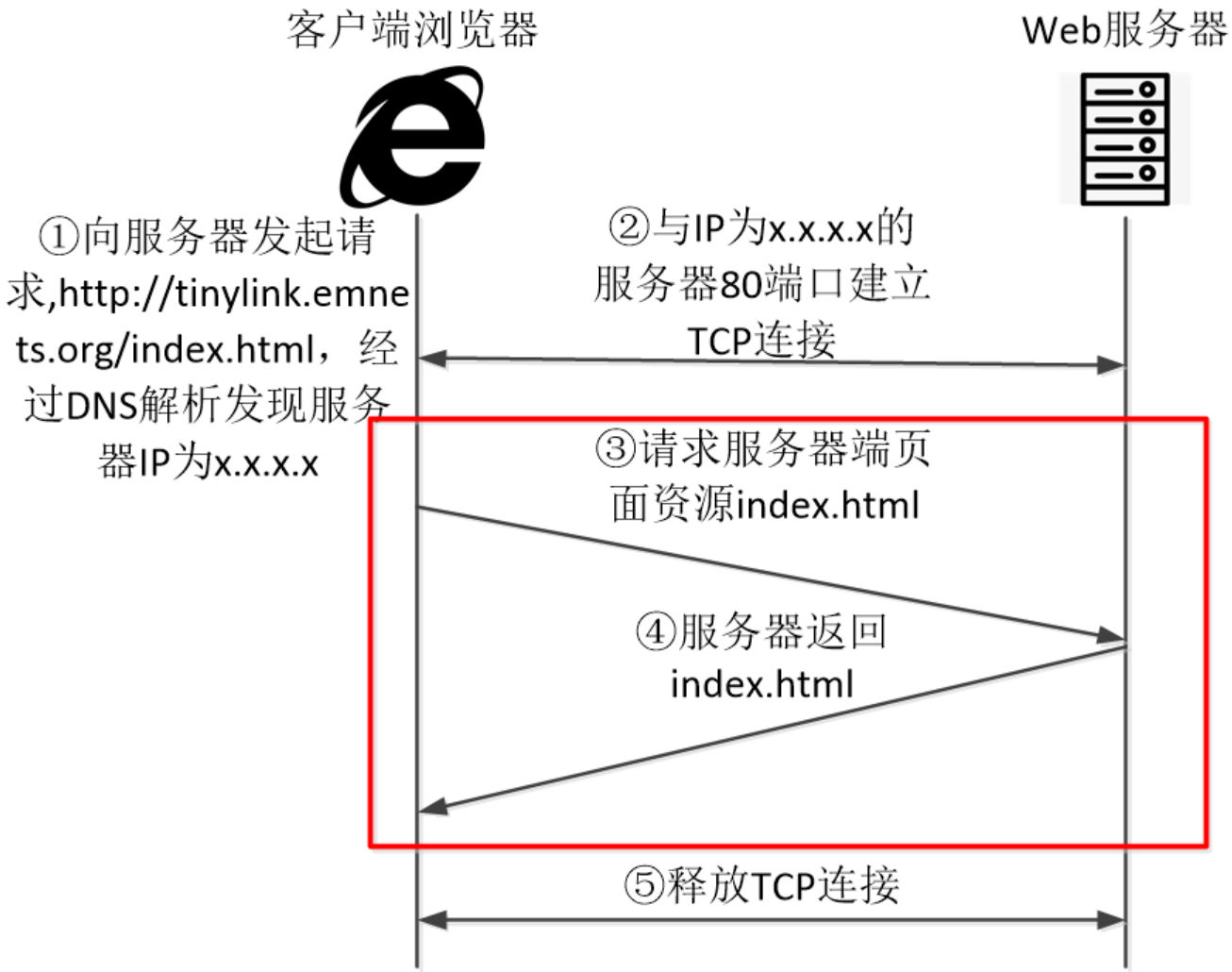
## ❑ 物联网设备连接至IoT云的方式



# 设备连接-HTTP

- HTTP(Hyper Text Transfer Protocol): 用于传输万维网文档



# 设备连接-HTTP

## ❑ 请求报文格式

## ❑ 示例

```
POST http://tinylink.emnets.org/index.php HTTP/1.1 \r\n
Connection:keep-alive \r\n
Host:x.x.x.x \r\n
\r\n
-----------------------------17833758738087276480198730 1
Content-Disposition: form-data; name="upload"; filename="test.cpp"
…
```

# 设备连接-HTTP

## ❑ 应答报文格式

## ❑ 示例

```
HTTP/1.1 200 OK \r\n
Cache-Control:no-store, no-cache \r\n
Connection:Keep-Alive \r\n
\r\n
<html lang="zh-CN" xmlns="http://www.w3.org/1999/xhtml">
…
```

# 设备连接-HTTP

## ❑ 部分常用请求方法

| 请求方法 | 意义 |
| --- | --- |
| GET | 读取一个Web页面 |
| POST | 当提交表单时需要用到POST，需要上传附加数据到服务器 |
| DELETE | 请求服务器删除指定的页面 |

## ❑ 请求返回状态码

| 状态码 | 含义 |
| --- | --- |
| 1xx | 通知消息，如请求已收到或正在处理 |
| 2xx | 成功 |
| 3xx | 重定向，浏览器如要完成请求还需进一步行动 |
| 4xx | 浏览器请求错误，例如页面不存在或请求语法不正确 |
| 5xx | 服务器差错，服务器失效 |

# 设备连接-HTTP

| 报文 | 长度（字节） |
| --- | --- |
| HTTP报文 | 824 |
| TCP控制头 | 20（固定头部）+12（选项） |

## ❑ 示例：上传一个温度数值(25度) 到云平台

请求行

```
POST http:tinylink.emnets.org/receiveData.php HTTP/1.1\r\n
```

首部行

```
Host: 10.214.149.119\r\n
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:47.0) Gecko/20100101 Firefox/47.0\r\n
Accept: application/json, text/javascript, */*; q=0.01\r\n
Accept-Language: en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
X-Requested-With: XMLHttpRequest\r\n
Content-Length: 295\r\n
Content-Type: multipart/form-data; boundary=---------------------------102678585633387710111393502 67\r\n
Connection: keep-alive\r\n
Cache-Control: max-age=0\r\n
\r\n
```

实体

```
----------------------------102678585633387710111393502 67\r\n
Content-Disposition: form-data; name="nodeid"\r\n\r\n
10001
----------------------------102678585633387710111393502 67
Content-Disposition: form-data; name="temperture"\r\n\r\n
25
----------------------------102678585633387710111393502 67--\r\n
```

# 设备连接-HTTP

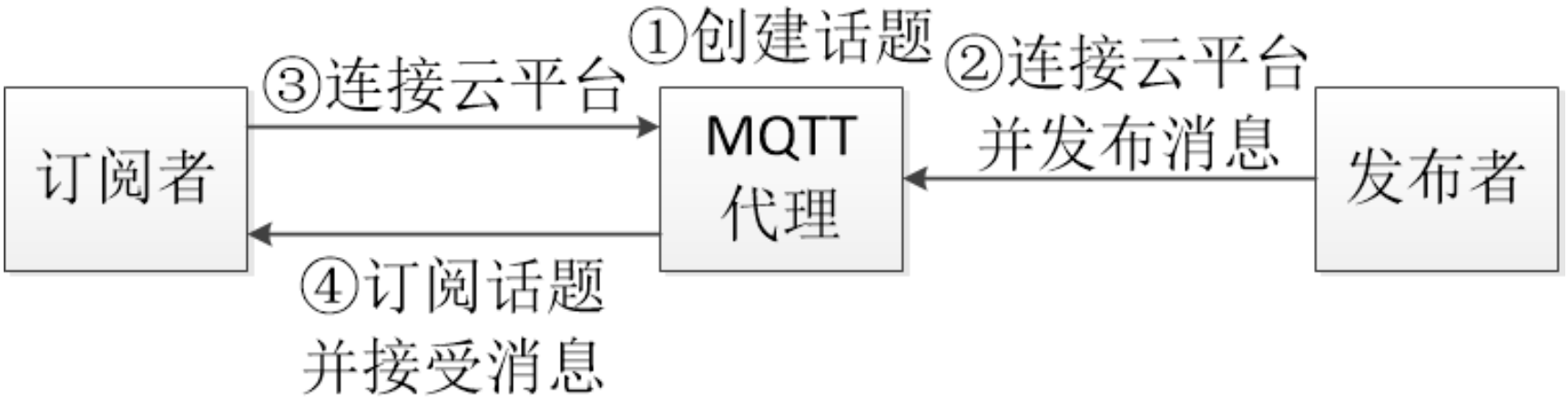
- **无状态**：HTTP请求是无状态的，每次请求都相互独立，请求之前需要建立TCP连接

- **主动请求**：HTTP都是客户端主动发起，服务端被动接收并处理请求，对于设备下行控制场景需要设备端搭建Web Server才能实现

- **协议包头较长**：HTTP请求返回协议携带很多首部行，会增加整个报文长度，对于物联网设备短消息、低频率场景而言是一个重量级数据传输协议

# 设备连接-MQTT

- MQTT（Message Queuing Telemetry Transport）
  - 一种M2M（machine-to-machine）数据传输协议，它被设计为极其轻量的发布/订阅消息传送模型。



# 设备连接-MQTT

## ❑ 消息传输示例（C++）

```
// 订阅者接收消息代码
connect(ip, port, clientID, username, pass);
void callBack(MessageData m){
  // 处理接收数据
}
subscribe("myTopic@rt", callBack);
```

```
// 发布者发布消息代码
connect(ip, port, clientID, username, pass);

char data[] = "{\"Temperature\": 21,
\"Humidity\":67}";
publish("myTopic@rt", data, MQTT::QoS2);
```

# 设备连接-MQTT

## ❑ 报文格式

| | 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 固定头部 | 字节1 | MQTT报文类型 | | | | MQTT报文Flag | | | |
| | 字节2... | 剩余长度（变长头部+数据内容） | | | | | | | |
| 变长头部 | 字节i | 包序号（高位字节） | | | | | | | |
| | 字节i+1 | 包序号（低位字节） | | | | | | | |
| | 字节m... | 附加内容（由报文类型决定） | | | | | | | |
| 数据 | 字节n... | 数据内容（部分报文存在） | | | | | | | |

# 设备连接-MQTT

| | 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 固定头部 | 字节1 | MQTT报文类型 | | | | MQTT报文Flag | | | |
| | 字节2... | 剩余长度（变长头部+数据内容） | | | | | | | |
| 变长头部 | 字节i | 包序号（高位字节） | | | | | | | |
| | 字节i+1 | 包序号（低位字节） | | | | | | | |
| | 字节m... | 附加内容（由报文类型决定） | | | | | | | |
| 数据 | 字节n... | 数据内容（部分报文存在） | | | | | | | |

# 设备连接-MQTT

## ❑ 报文类型

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 字节1 | MQTT报文类型 | | | | MQTT报文Flag | | | |
| 字节2 | 剩余长度（变长头部+数据内容） | | | | | | | |

固定头部

| 报文类型 | 值 | 数据流向 | 描述 |
|---|---|---|---|
| Reserved | 0/15 | 无 | 保留 |
| CONNECT | 1 | Client→Server | Client向Server发起的连接请求 |
| CONNACK | 2 | Server→Client | Server对连接请求返回的ACK |
| PUBLISH | 3 | Client↔Server | 向话题发布的消息 |
| PUBACK | 4 | Client↔Server | 发布消息后返回的ACK |

# 设备连接-MQTT

| | 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 固定头部 | 字节1 | MQTT报文类型 | | | | MQTT报文Flag | | | |
| | 字节2... | 剩余长度（变长头部+数据内容） | | | | | | | |
| 变长头部 | 字节i | 包序号（高位字节） | | | | | | | |
| | 字节i+1 | 包序号（低位字节） | | | | | | | |
| | 字节m... | 附加内容（由报文类型决定） | | | | | | | |
| 数据 | 字节n... | 数据内容（部分报文存在） | | | | | | | |

**报文Flag（仅允许设置PUBLISH类型报文的Flag）**
**Bit3(DUP Flag)：1代表重传，0代表该数据包是第一次发送**
**Bit1~2(QoS Flag)：0~2代表三种QoS，值越大QoS越高**
**Bit0(RETAIN Flag)：1代表MQTT代理保留消息，0代表MQTT不保留消息**

# 设备连接-MQTT

## ❑ Publish数据传输举例

*红色字段表示必须设置的内容

```
struct Message{
    enum QoS qos;
    bool retained;
    bool dup;
    unsigned short id;
    void *payload;
    size_t payloadlen;
};   // 消息结构体

Message myMsg;
myMsg.payload = “hello”;
myMsg.payloadlen = strlen(“hello”)+1;
myMsg.id = 0x1234;

Case 1: MQTT代理要为后续订阅话题的Client
转发保留的消息
myMsg.qos = QoS2;
myMsg.retained = true;
myMsg.dup = false;
```

```
Case 2: 清空MQTT代理某话题下保留的消息
myMsg.qos = QoS0;
myMsg.retained = true;
myMsg.dup = false;

Case 3: 让MQTT至少收到一次该消息，不保留
myMsg.qos = QoS1;
myMsg.retained = false;
myMsg.dup = false;

Case 4: 让MQTT以最低传输代价接收消息
myMsg.qos = QoS0;
myMsg.retained = false;
myMsg.dup = false;

Case 5: 数据包重传
myMsg.qos = QoS1;
myMsg.retained = false;
myMsg.dup = true;
```

# 设备连接-MQTT

| | 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 固定头部 | 字节1 | MQTT报文类型 | | | | MQTT报文Flag | | | |
| | 字节2... | 剩余长度（变长头部+数据内容） | | | | | | | |
| 变长头部 | 字节i | 包序号（高位字节） | | | | | | | |
| | 字节i+1 | 包序号（低位字节） | | | | | | | |
| | 字节m... | 附加内容（由报文类型决定） | | | | | | | |
| 数据 | 字节n... | 数据内容（部分报文存在） | | | | | | | |

**剩余长度：当前报文变长头部和数据内容的字节长度，用1~4个字节表示，不同数值被编码成不同的长度**

# 设备连接-MQTT

| | 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 固定头部 | 字节1 | MQTT报文类型 | | | | MQTT报文Flag | | | |
| 固定头部 | 字节2... | 剩余长度（变长头部+数据内容） | | | | | | | |
| 变长头部 | 字节i | 包序号（高位字节） | | | | | | | |
| 变长头部 | 字节i+1 | 包序号（低位字节） | | | | | | | |
| 变长头部 | 字节m... | 附加内容（由报文类型决定） | | | | | | | |
| 数据 | 字节n... | 数据内容（部分报文存在） | | | | | | | |

# 设备连接-MQTT

| | 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 固定头部 | 字节1 | MQTT报文类型 | | | | MQTT报文Flag | | | |
| | 字节2... | 剩余长度（变长头部+数据内容） | | | | | | | |
| 变长头部 | 字节i | 包序号（高位字节） | | | | | | | |
| | 字节i+1 | 包序号（低位字节） | | | | | | | |
| | 字节m... | 附加内容（由报文类型决定） | | | | | | | |
| 数据 | 字节n... | 数据内容（部分报文存在） | | | | | | | |

**数据内容，拥有数据内容的报文类型：**
**CONNECT，SUBSCRIBE，SUBACK，UNSUBSCRIBE，PUBLISH**

# 设备连接-MQTT

TCP控制头：20固定头部+12选项=32

- **HTTP中同样的示例：上传一个温度数值(25度)到云**

```
struct Message{
    enum QoS qos;
    bool retained;
    bool dup;
    unsigned short id;
    void *payload;
    size_t payloadlen;
};   // 消息结构体

Message myMsg;
myMsg.payload = “25”;
myMsg.payloadlen = strlen(“25”)+1;
myMsg.id = 0x1234;

Case : MQTT代理要为后续订阅话题的Client转发保留的消息
myMsg.qos = QoS2;
myMsg.retained = true;
myMsg.dup = false;
publish(“Temperature”, myMsg);
```

| | 描述 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 固定头部 | | | | | | | | | |
| 字节1 | PUBLISH\|RETAIN | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 字节2 | Len(18) | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 变长头部 | | | | | | | | | |
| 话题名称 | | | | | | | | | |
| 字节3 | Len MSB(0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 字节4 | Len LSB(11) | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 字节5 | 'T'(0x54) | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 字节6 | 'e'(0x65) | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 字节7 | 'm'(0x6D) | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 字节8 | 'p'(0x70) | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 字节9 | 'e'(0x65) | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 字节10 | 'r'(0x72) | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 字节11 | 'a'(0x61) | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 字节12 | 't'(0x74) | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 字节13 | 'u'(0x75) | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 字节14 | 'r'(0x72) | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 字节15 | 'e'(0x65) | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 包序号 | | | | | | | | | |
| 字节16 | Pack ID MSB(0x12) | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 字节17 | Pack ID LSB(0x34) | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 数据内容 | | | | | | | | | |
| 字节18 | '2'(0x32) | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 字节19 | '5'(0x35) | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 字节20 | '\0'(0x00) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# 设备连接-MQTT

TCP控制头：20固定头部+12选项=32

- **HTTP中同样的示例：上传一个温度数值(25度)到云**

```
struct Message{
    enum QoS qos;
    bool retained;
    bool dup;
    unsigned short id;
    void *payload;
    size_t payloadlen;
};   // 消息结构体

Message myMsg;
myMsg.payload = “25”;
myMsg.payloadlen = strlen(“25”)+1;
myMsg.id = 0x1234;

Case : MQTT代理要为后续订阅话题的Client转发保留的消息
myMsg.qos = QoS
myMsg.retained =
myMsg.dup = false;
publish(“Temperature”, myMsg);
```

| | 描述 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 固定头部 | | | | | | | | | |
| 字节1 | PUBLISH\|RETAIN | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 字节2 | Len(18) | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 变长头部 | | | | | | | | | |
| 话题名称 | | | | | | | | | |
| 字节3 | Len MSB(0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 字节4 | Len LSB(11) | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 字节5 | 'T'(0x54) | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 字节6 | 'e'(0x65) | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 字节7 | 'm'(0x6D) | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 字节8 | 'p'(0x70) | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 字节9 | 'e'(0x65) | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 字节10 | 'r'(0x72) | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 字节11 | 'a'(0x61) | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 字节12 | 't'(0x74) | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 字节13 | 'u'(0x75) | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 字节14 | 'r'(0x72) | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 字节15 | 'e'(0x65) | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 包序号 | | | | | | | | | |
| 字节16 | Pack ID MSB(0x12) | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 字节17 | Pack ID LSB(0x34) | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| | | | | | | | | | |
| 字节18 | Z (0x32) | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 字节19 | '5'(0x35) | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 字节20 | '\0'(0x00) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

32+20 B (MQTT) vs. 32+824 B (HTTP) !

# 设备连接-MQTT特点

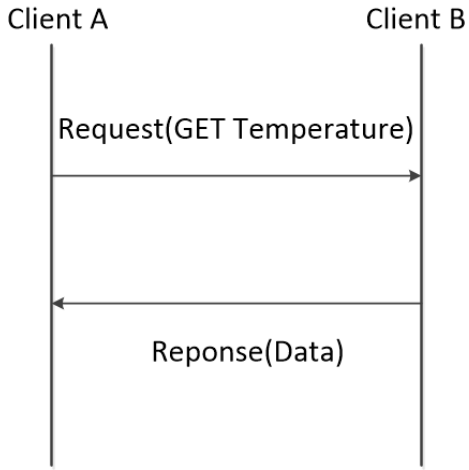
- **轻量级**：数据包头短
- **支持广播**：基于发布-订阅模式，可实现一对多通信
- **支持双向通信**：通过两个话题可建立两个Client之间的双向通信。一个话题中，Client A publish、Client B subscribe，另一个话题中，Client A subscribe、Client B publish.
- **灵活**：可选QoS有三种，可基于不同的应用场景灵活选择

# 设备连接-CoAP

- CoAP（Constrained Application Protocol)
  - 定义： CoAP是一种M2M（machine-to-machine）数据传输协议，主要用于物联网中资源受限的节点和网络应用。

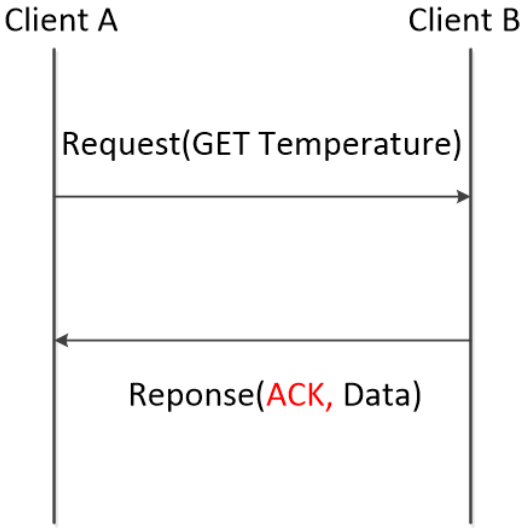
# 设备连接-CoAP

## Non-Confirmable Request/Response消息传输模型




Request和Response只发送一次，没有ACK确认， Request和Response中任何一个丢失都会导致数据传输失败

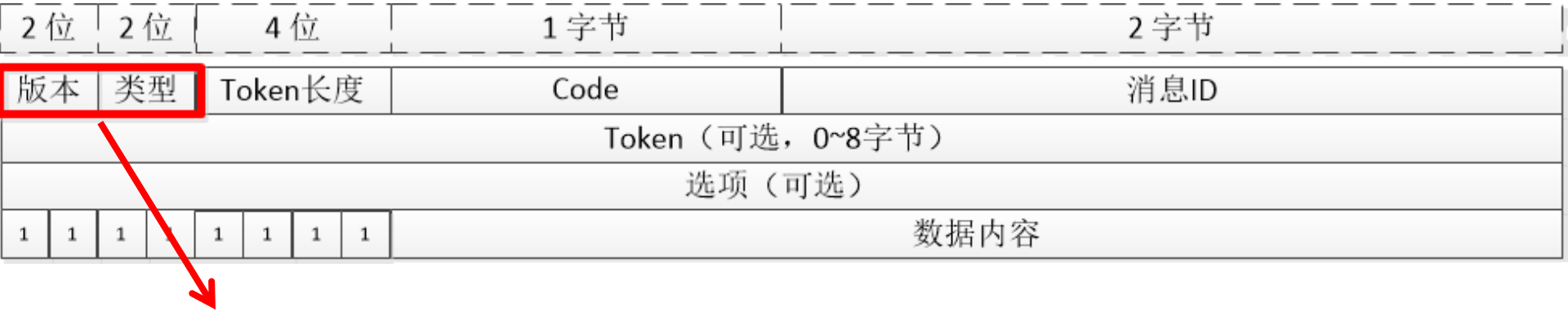
## Confirmable Request/Response消息传输模型




Request和Response任何一个丢失都会导致重传，ACK和数据作为Response同时到达请求端

# 设备连接-CoAP

## ❑ 报文格式




## ❑ 版本：01

## ❑ 类型：

| 值 | 消息类型 | 描述 |
|---|---|---|
| 00 | Confirmable message | 请求消息（需要返回确认），使用停止等待+指数退避实现重传 |
| 01 | Non-confirmable message | 请求消息（不需要返回确认） |
| 10 | ACK（Acknowledgement message） | 确认消息（针对处理正确的请求消息） |
| 11 | RST（Reset Message） | 错误消息（针对处理错误的请求消息） |

# 设备连接-CoAP

## ❑ Code字段

| 2 位 | 2 位 | 4 位 | 1 字节 | 2 字节 |
|---|---|---|---|---|
| 版本 | 类型 | Token长度 | Code | 消息ID |
| Token（可选，0~8字节） | | | | |
| 选项（可选） | | | | |
| 1 1 1 1 1 1 1 1 | | | 数据内容 | |

## ❑ Code字段格式

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 类别 | | | 详细 | | | | |

| 类别 | 详细 | 描述 |
|---|---|---|
| 0 | 00 | 空消息 |
| 0 | 01~31 | 请求 |
| 1 | 00~31 | 保留 |
| 2~5 | 00~31 | 响应 |
| 6~7 | 00~31 | 保留 |

# 设备连接-CoAP

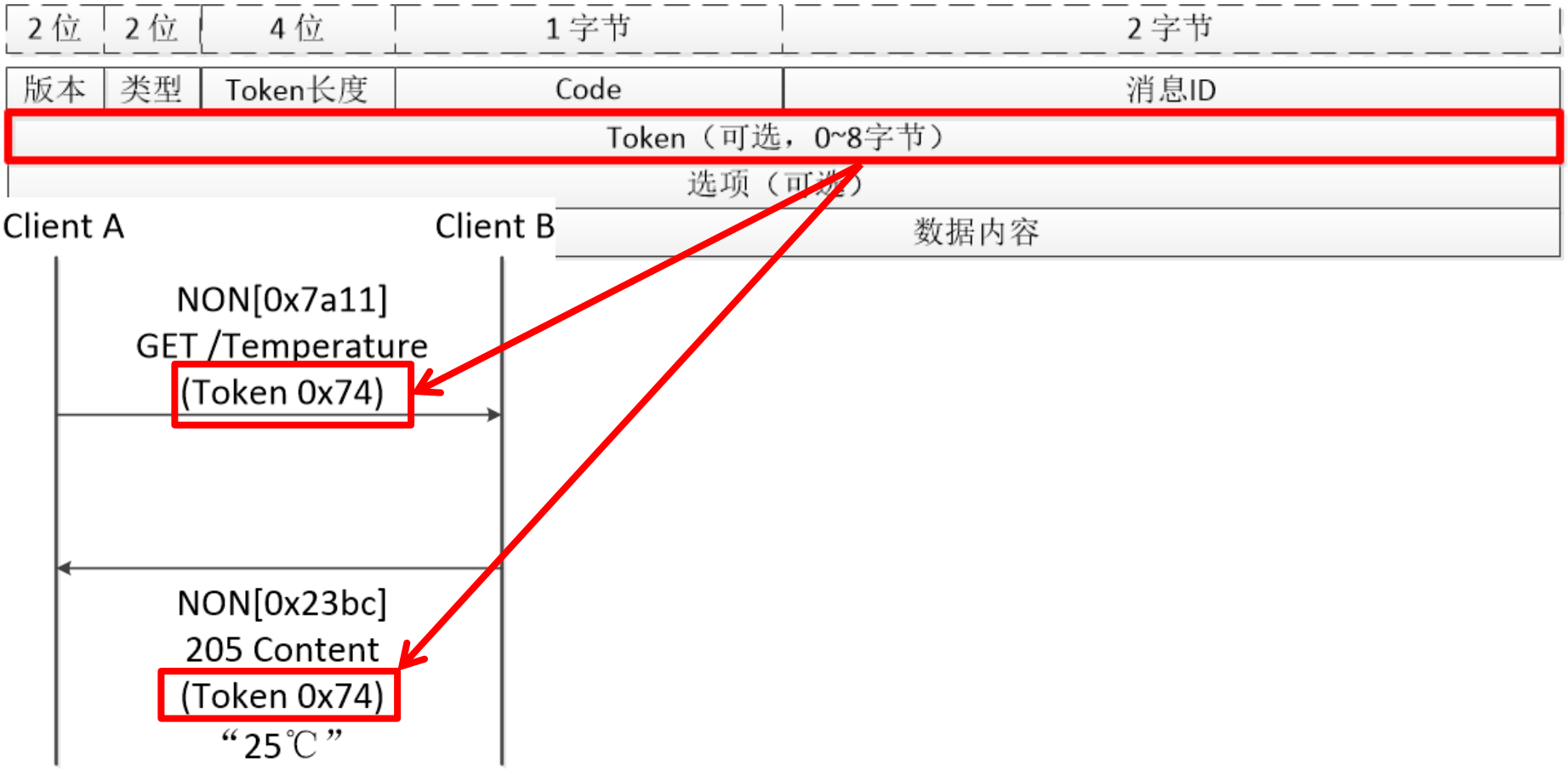
| 2 位 | 2 位 | 4 位 | 1 字节 | 2 字节 |
|---|---|---|---|---|
| 版本 | 类型 | Token长度 | Code | 消息ID |
| Token（可选，0~8字节） | | | | |
| 选项（可选） | | | | |
| 1 1 1 1 1 1 1 1 | | | 数据内容 | |

- **消息ID**
  - 检测重复消息，确保同一份消息在接收端只被处理一次
  - ACK/RST返回确认/拒绝数据包序号

# 设备连接-CoAP

## ❑ Token（0~8字节）

- 将响应报文和请求报文匹配起来
- 由Client生成，每个Token区分一个并发的Request



# 设备连接-CoAP

## ❑ 选项报文

| 2 位 | 2 位 | 4 位 | 1 字节 | 2 字节 |
|---|---|---|---|---|
| 版本 | 类型 | Token长度 | Code | 消息ID |
| Token（可选，0~8字节） | | | | |
| 选项（可选） | | | | |
| 1 1 1 1 1 1 1 1 | | | 数据内容 | |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 选项增量 | | | | 选项长度 | | | |
| 选项增量扩展（0~2字节） | | | | | | | |
| 选项长度扩展（0~2字节） | | | | | | | |
| 选项值（0或多字节） | | | | | | | |

# 设备连接-CoAP

- HTTP中同样的示例：上传一个温度数值(25度)到云

8+32 B (CoAP) vs. 32+824 B (HTTP)！

| 2 位 | 2 位 | 4 位 | 1 字节 | 2 字节 |
|---|---|---|---|---|
| 01 | 00 | 0001 | 0000 0001 | 0x7d34 |
| 0x76 | | | | |
| 1011 | | 1011 | “Temperature”(11 Bytes) | |
| 1101 | | 0011 | 0x0C | “100”(3 Bytes) |

Client A　　Client B

Header: GET(Type=CON, Message ID=0x7d34, Token=0x76)
Uri-Path: “Temperature”
Max-Age: 100

Header: 205 Content(Type=ACK, Message ID=0x7d34,Token=0x76)
Payload: “25 C”

| 2 位 | 2 位 | 4 位 | 1 字节 | 2 字节 |
|---|---|---|---|---|
| 01 | 10 | 0001 | 0100 0101 | 0x7d34 |
| 0x76 | | | | |
| 0xFF | | | “25 C”(4 Bytes) | |

# 设备连接-CoAP特点

- **轻量级**：数据包头部长度短，相比于HTTP消息长度减少很多
- **没有中间代理**：节点之间可以直接通信，无需通过中间服务器进行消息转发
- **类似HTTP**：具有类似HTTP Header的控制字段，可实现GET、POST、DELETE等方法
- **灵活**：消息传输方式（GET、POST，Confirmable、Non-confirmable）多，每种传输方式代价不同，可基于不同的应用场景灵活选择
- **主动请求**：数据从A传输到B，则B必须先主动发送Request请求到A

# 设备连接

## ❑ 对比MQTT，HTTP，CoAP

| | MQTT | CoAP | HTTP |
|---|---|---|---|
| OSI模型所处层次 | 应用层 | 应用层 | 应用层 |
| 传输层协议 | TCP | UDP | TCP |
| 可靠 | √ | 可选 | √ |
| 传输代价 | 小 | 小 | 大 |
| 多播 | √ | √ | × |
| 双向通信（资源受限的设备） | √ | √ | × |
| 中间代理转发 | √ | × | √ |
| 温度上传示例报文长度（单位：字节） | 20（MQTT）<br>+32（TCP） | 32（CoAP）<br>+8（UDP） | 824（HTTP）<br>+32（TCP） |

# 设备连接

## ❑ 各大云平台对不同协议的支持

| | HTTP | HTTPS | MQTT | CoAP |
|---|---|---|---|---|
| IBM Watson | × | × | √ | × |
| AWS IoT | × | √ | √ | × |
| Azure IoT | × | √ | √ | × |
| Link Develop | √ | √ | √ | √ |

# 设备管理——管理内容

- 管理内容：
  - 设备生命周期：注册、删除设备
  - 设备状态：跟踪设备online-offline状态
  - 设备固件版本：跟踪设备当前版本号
  - 固件更新：维护更新地址列表；监视更新过程

| 设备标识属性 | 意义 | 对应管理内容 |
| --- | --- | --- |
| deviceId | 设备ID | |
| status | 设备状态（已启用/已禁用） | 设备生命周期 |
| connectionState | 连接状态（已连接/已断开） | 设备状态 |
| version | 设备当前版本号 | 设备固件版本 |
| updateList | OTA更新包的下载地址及对应的版本信息 | 固件更新 |

# 设备管理——方式与服务

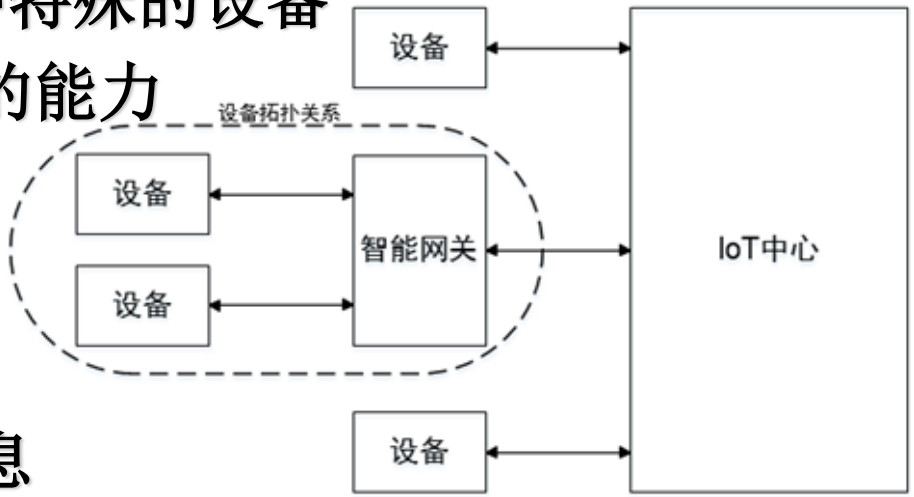
- **对象**
  - 物联网设备
  - 智能网关

- **方式**
  - 直接管理：用户通过云直接向设备发送指令
  - 设备影子：利用设备的持久虚拟版（包含设备的最新状态）管理设备

- **服务**
  - OTA升级：在线检测设备是否需要升级并管理升级过程

# 设备管理——智能网关

- 定义
  - 网关是IoT平台中一种特殊的设备
  - 具有聚合应用和设备的能力
- 特点
  - 管理模式变为多层
- 任务
  - 收集传感器节点的信息
  - 执行数据协议转换（例：Zigbee-Wifi）
  - 将协议转换后的数据快速地发送
  - 管理和控制能力（本地&远程）
    - *直接对单一设备进行控制*



# 设备管理——智能网关

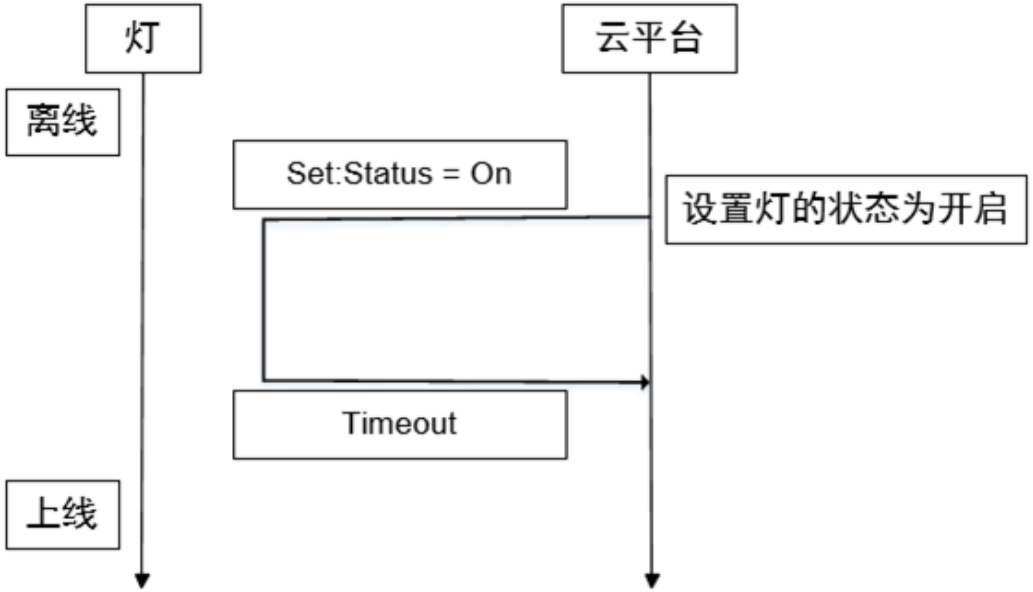
- 例子：Azure IoT中两类网关

- 现场网关
  - 使用 Azure IoT 网关 SDK创建实体网关
  - 该 SDK 可将自定义协议转换为 IoT 中心所理解的协议

- 云网关
  - 是自定义的 Azure IoT虚拟协议网关
  - 是在云中运行的开放源代码组件
  - 方便管理云网关下的一系列IoT设备

# 设备管理——直接管理

- 直接管理是指用户可以通过云直接向设备发送指令
- 特点
  - 遵循请求-响应模式
  - 适用于需要得到立即确认结果的场景
  - 在设备离线状态下，可能造成管理失效

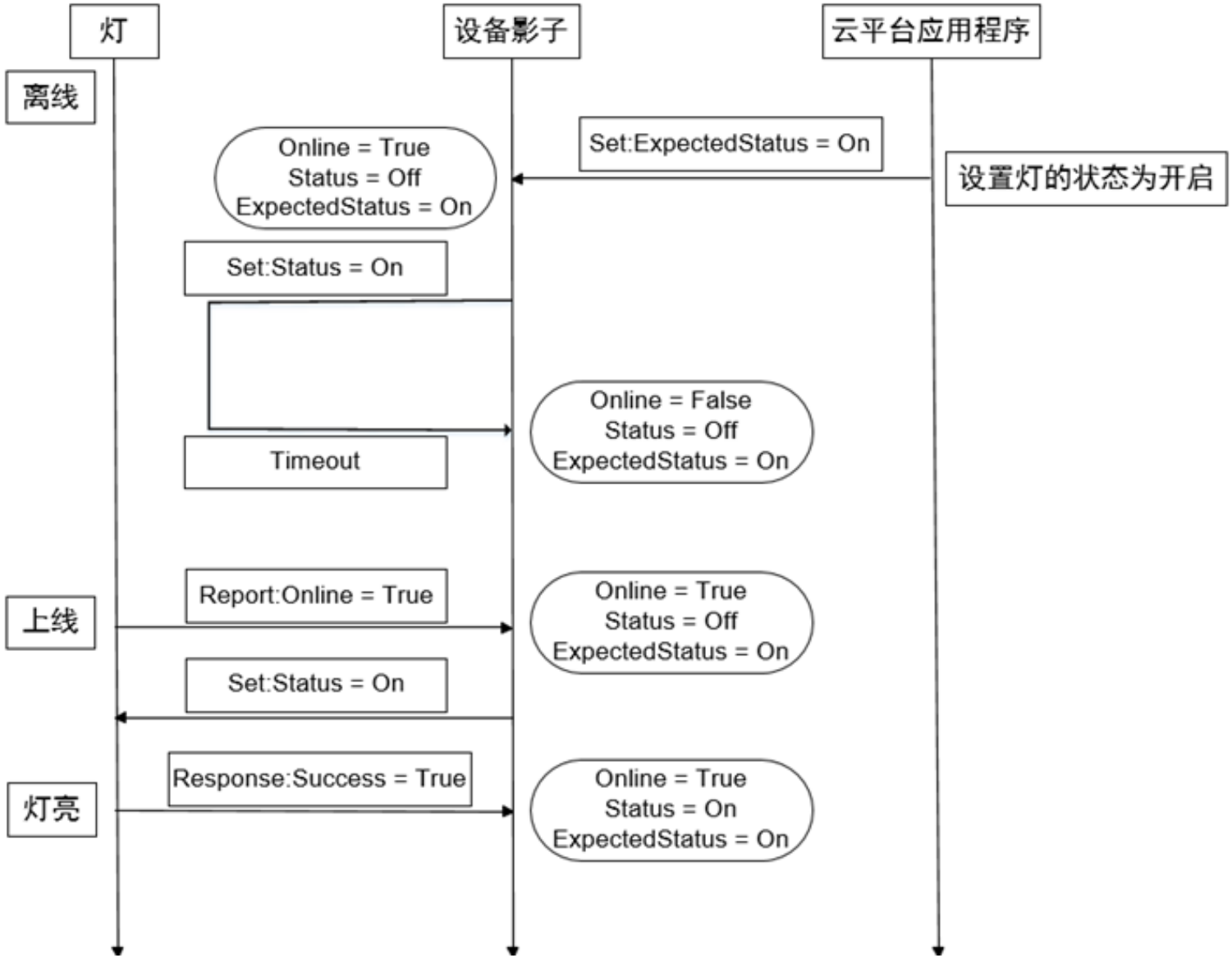


# 设备管理——设备影子

- 通过设备的持久虚拟版（也可称为设备孪生，包含设备的最新状态），间接管理设备。

- 特点
  - IoT云平台将主动比较用户设置的设备期望状态和设备最近报告状态之间的差异并进行相应设置。
  - 即便离线，也可设置设备未来的期望状态，并在设备在线时实时生效。

# 设备管理——设备影子


灯
设备影子
云平台应用程序
离线
Online = True
Status = Off
ExpectedStatus = On
Set:ExpectedStatus = On
设置灯的状态为开启
Set:Status = On
Online = False
Status = Off
ExpectedStatus = On
Timeout
上线
Report:Online = True
Online = True
Status = Off
ExpectedStatus = On
Set:Status = On
灯亮
Response:Success = True
Online = True
Status = On
ExpectedStatus = On

# 设备管理——OTA升级

- **OTA升级：空中(Over the Air)升级技术，在线检测设备是否需要升级并管理升级过程**

- **具体过程**
  - 云端会记录固件当前版本号。设备固件版本号需要在系统启动时上报，不需要周期循环上报
  - 根据固件当前的版本号及最新发布的固件版本号来判断需要升级
  - 固件升级后主动上报升级后的版本号，由此云端可以判断升级是否成功

# OTA升级中的差分计算

- **差分文件格式**
  - 拷贝命令 (cost = β = 5)

    $COPY_1$ $<n>_2$ $<old_addr>_2$
  - 添加命令 (cost = α+n =3+n)

    $ADD_1$ $<n>_2$ $<BYTE1 \ldots BYTEn>_n$

- **差分算法**
  - 问题：要实现增量最小化需要拷贝或添加多少个字节？
  - R3diff: 此算法的最差时间复杂度为$O(n^3)$, 平均时间复杂度为$O(n^2)$. 空间复杂度为$O(n)$.

*Dong W, Chen C, Bu J, et al. Optimizing relocatable code for efficient software update in networked embedded systems[J]. ACM Transactions on Sensor Networks (TOSN), 2015, 11(2): 22.v*

# 设备管理——OTA升级中的安全问题

- **OTA升级中的安全很关键**
  - 百度AI团队OTA安全评估：
    - *95%设备未能保护OTA升级包的数据内容*
    - *90%设备无法抵御通信过程中的中间人攻击*
    - *61%设备可强制将系统/APP版本降级至低版本。*

- **解决方式**
  - 签名校验、双向认证、升级包加密、通讯安全防护

# IFTTT

- IFTTT: If This Then That, 一种简单的基于条件语句的程序逻辑
- Ex1:

  IF soil_moisture <= threshold

  THEN start irrigation

- Ex2:

  IF temp >= threshold1 &&
  humidity <= threshold2

  THEN start alert

- IFTTT特点
  - 可以通过简单的条件语句表达多种应用逻辑
  - 一般在云端执行，可以解耦设备端代码和应用逻辑
  - 支持多设备联动

# IFTTT的问题

- 规则冲突
- 实时性 vs. 能耗
- 公平性问题
- 应用逻辑与执行操作的耦合性

# IFTTT – 规则冲突

**IF** light > 10000lux
**THEN** light.turnOff

**IF** light < 500lux
**THEN** light.turnOn

**有没有冲突?**

---

**IF** light > 10000lux
**THEN** light.turnOff

**IF** light < 20000lux
**THEN** light.turnOn

**有没有冲突?**

# IFTTT – 规则冲突

**IF** light > 10000lux
**THEN** light.turnOff

**IF** light < 20000lux
**THEN** light.turnOff

**有没有冲突?**

---

**IF** light > 10000lux
**THEN** light.turnOff

**IF** time ∈ [19:00, 24:00]
**THEN** light.turnOn

**有没有冲突?**

# IFTTT – 规则冲突

- **规则冲突的条件：**
  - 1. IF条件可能同时成立
  - 2. 控制同一个IoT设备并且控制行为冲突

# IFTTT – 规则冲突检测

- **Ex:** **IF** cond1(sensor1)
  **THEN** light.turnOff

  **IF** cond2(sensor2) || cond3(sensor1)
  **THEN** light.turnOn
- **冲突检测**
  - 引入3个二元指示变量：
    - *x1=1 iff cond1(sensor1)=1*
    - *x2=1 iff cond2(sensor2)=1*
    - *x3=1 iff cond3(sensor1)=1*
  - 求解可满足性问题
    - *是否存在一组(x1,x2,x3)的取值使得x1&&(x2||x3)=1？→ (1,1,1), (1,1,0), (1,0,1)*
  - 对同一sensor1的cond1, cond3依次检查：
    - *(1,1): 如果cond1 ∩ cond3 = Φ, 则不能同时成立，所以第1,3组解肯定不可满足；否则一定会发生冲突*
    - *(1,0): 如果cond1 ∩ !cond3 = Φ, …否则检查sensor2*

# IFTTT– 思考题

- IF (time ∈ [07:00, 17:00]) || (light > 10000lux)
- THEN light.turnOff

- IF (light < 1000lux) || (sound > 55dB)
- THEN light.turnOn

- Conflict: IF light = 999lux at 07:01

# IFTTT – 规则冲突检测SMT

- **布尔可满足性问题 (SAT)**
  - 给定布尔表达式，是否存在一组变量赋值，使结果为真
- **可满足性模理论 (SMT)**
  - SMT是SAT的推广，涉及非布尔变量
    - *例如实数、整数、数组等*
- **Z3**
  - 由微软研究院开发的优秀SMT Solver
  - 能够检查表达式的可满足性


- De Moura, Leonardo, and Nikolaj Bjørner. "Z3: An efficient SMT solver." *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, Berlin, Heidelberg, 2008.

# IFTTT的问题

- 规则冲突
- 实时性 vs. 能耗
- 公平性问题
- 应用逻辑与执行操作的耦合性

# IFTTT – 实时性 vs. 能耗

- If: humidity > 25 and light < 1000lux
- Then: window.close

- 对IoT设备的查询频率高
  - 实时性高
  - 但，能耗过大
- 对IoT设备的查询频率低
  - 能耗小
  - 但，实时性低，e.g. 关窗响应不及时可能导致雨水淋进房间

# IFTTT – 实时性 vs. 能耗

- **解决方法：自适应调整查询间隔**
  - 传感数据可预测接近阈值时，缩小查询间隔；
  - 反之，可以扩大查询间隔

- **增强的IFTTT语法：**

$$
\begin{aligned}
\text{App_rainy} ::= \ & \text{IF } H > 75 \ \&\& \ L < 1{,}000 \\
& \text{THEN Window.close()} \\
& \underline{\text{WITH Time} < 60 \ \&\& \ \text{MissRatio} < 0.01}
\end{aligned}
$$

- *Heo, Seonyeong, et al. " RT-IFTTT: Real-Time IoT Framework with Trigger Condition-aware Flexible Polling Intervals." Real-Time Systems Symposium (RTSS), 2017 IEEE . IEEE, 2017.*

# IFTTT的问题

- 规则冲突
- 实时性 vs. 能耗
- 公平性问题
- 应用逻辑与执行操作的耦合性

# IFTTT – 公平性问题

- **公平性问题：**
  - 某应用设置多条规则或者单条规则执行频率过高，可能会导致其余规则无法及时执行

- **解决方法：**
  - 规则调度

# IFTTT的问题

- 规则冲突
- 实时性 vs. 能耗
- 公平性问题
- 应用逻辑与执行操作的耦合性

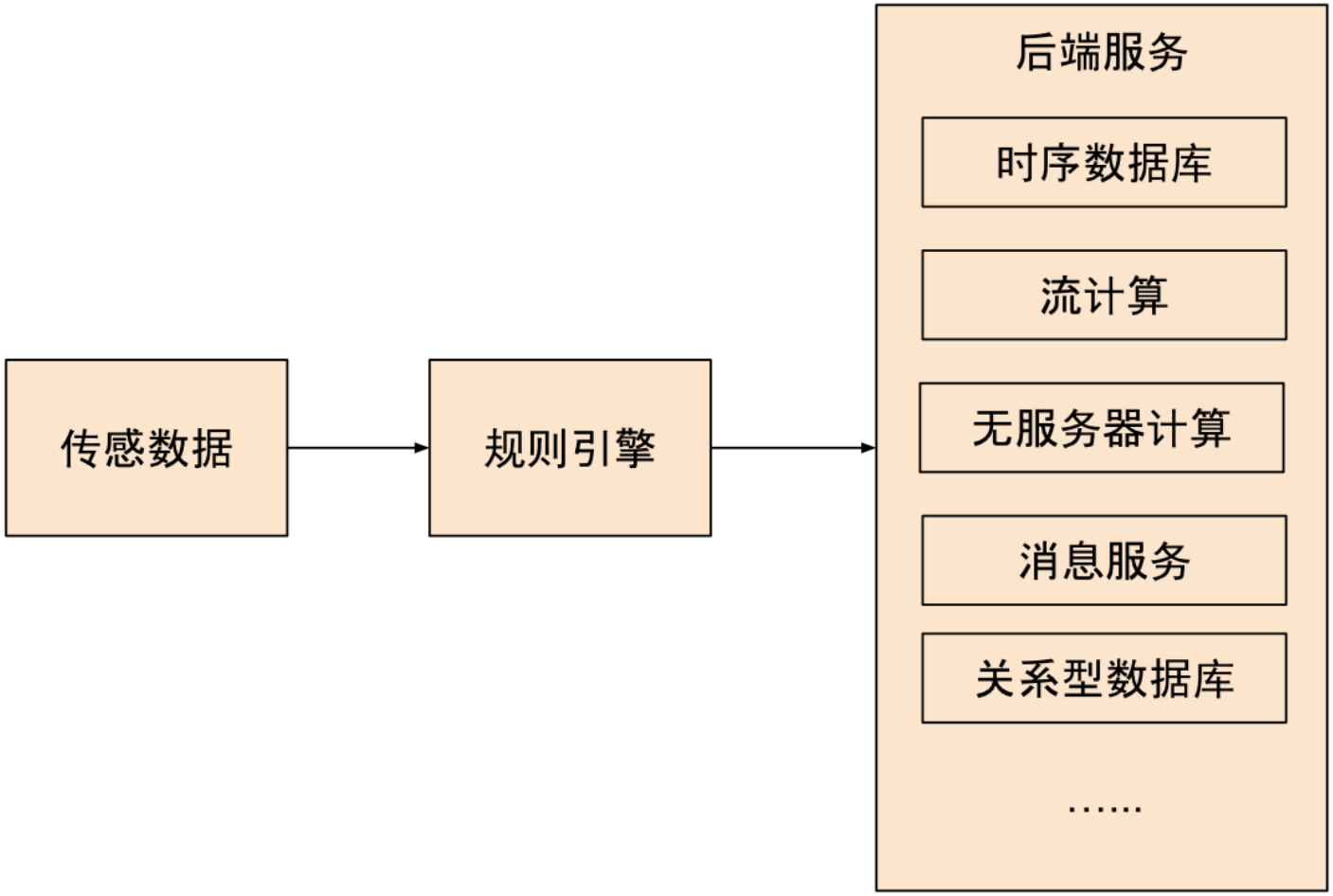
# 规则引擎

- 为了进一步解耦应用逻辑与执行操作，工业界的实际实现（称为规则引擎）不同于上述IFTTT规则的使用方法。以阿里云为例：
  - 可以使用SQL语句提取传感数据

  - 提供消息转发（至其它服务）的功能，例如：
    - *将数据转发至时序数据库*
    - *结合函数计算进行全托管的计算服务*
    - *发送消息推送给所有的相关用户*
    - *将数据发送给云平台的机器学习服务*

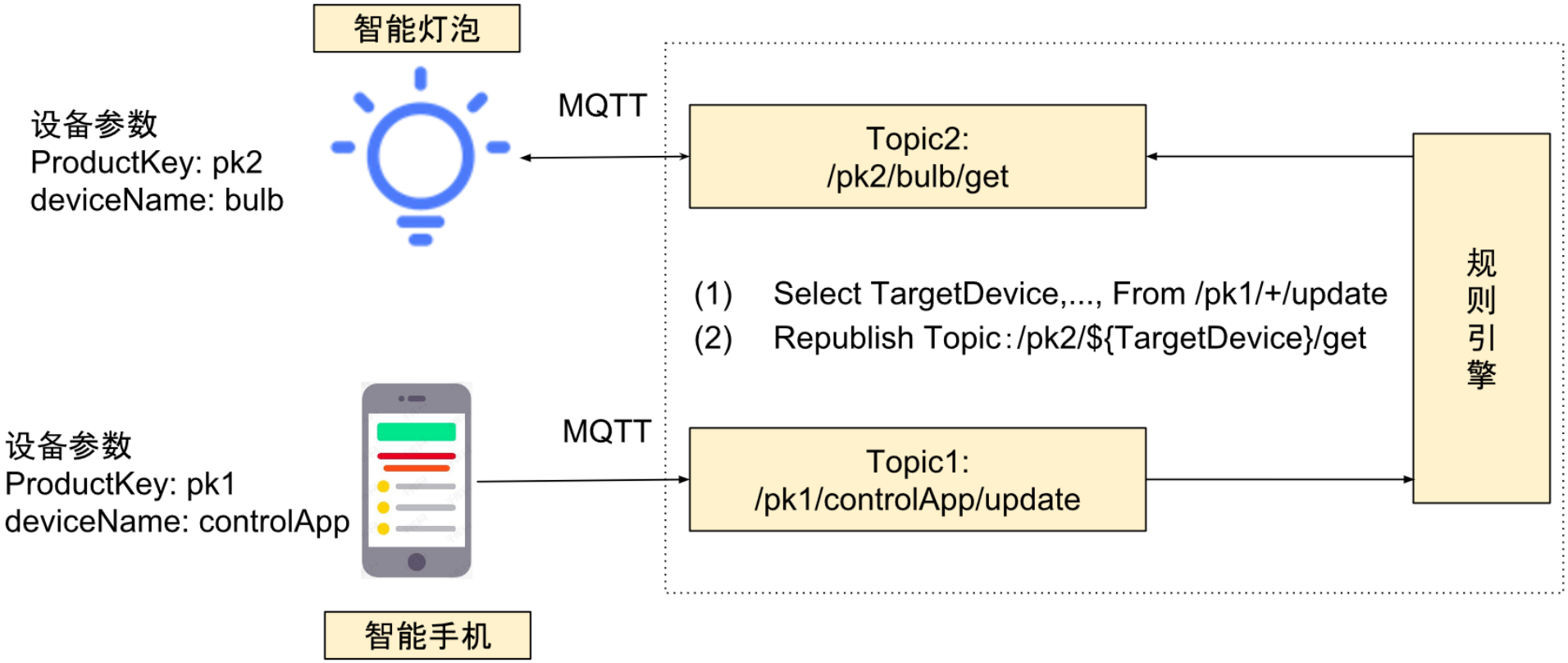
# 规则引擎

## ❑ 规则引擎在物联网云平台中的位置



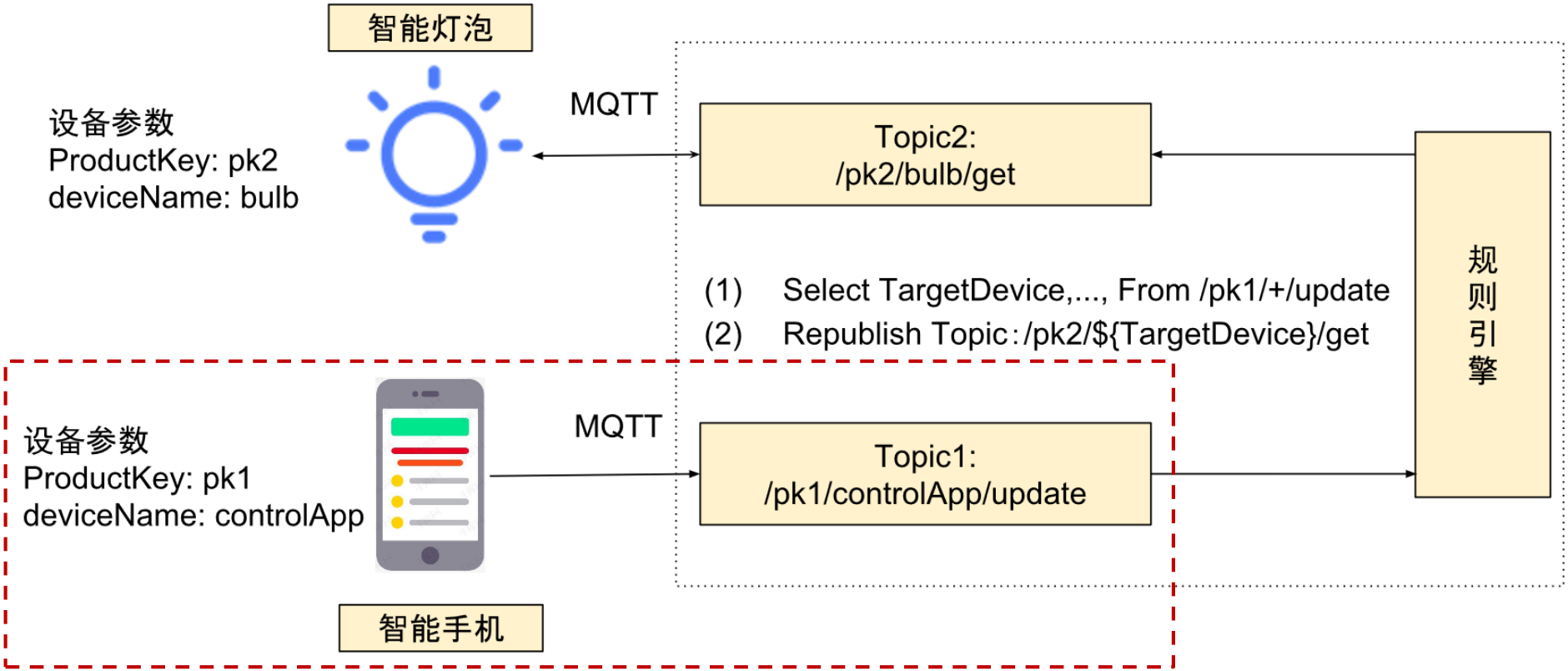
# 规则引擎- 案例

- 案例：使用智能手机控制智能灯泡



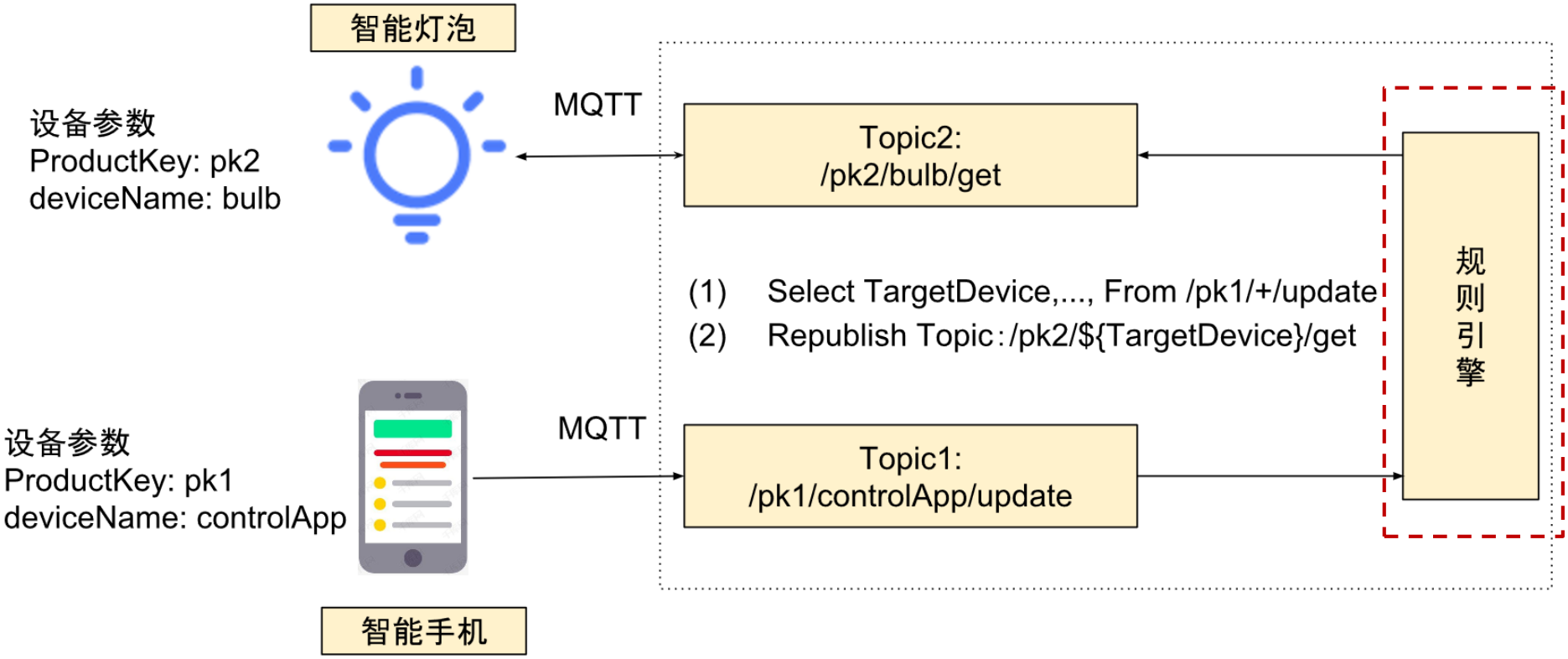
# 规则引擎- 案例

- （1）智能手机将消息发布到Topic1
  - 手机App获取到智能灯泡的deviceName
  - 通过MQTT向物联网平台中的Topic1发布消息



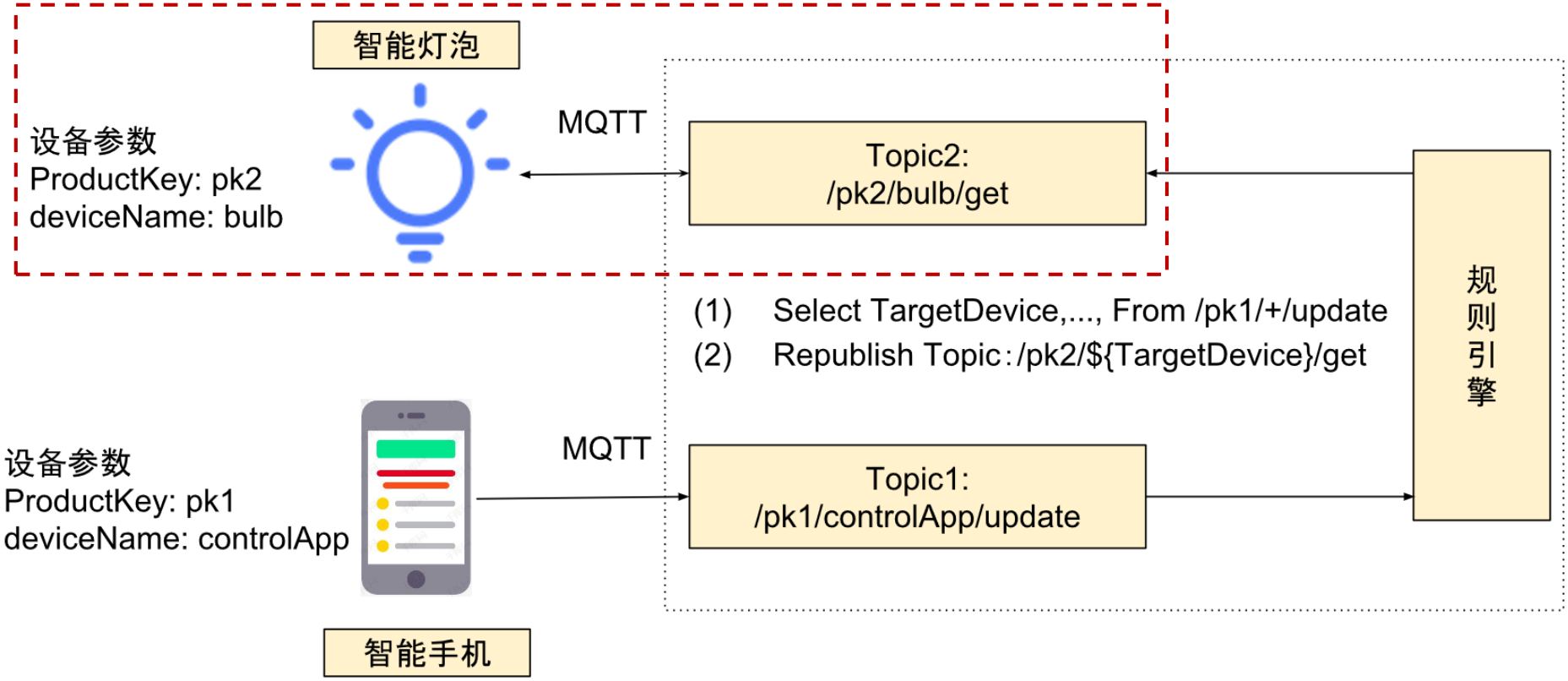
# 规则引擎- 案例

- （2）在规则引擎中配置规则并执行转发
  - 配置规则从Topic1: /pk1/controlApp/update/筛选消息
  - 将消息转发到Topic2: /pk2/bulb/get



# 规则引擎- 案例

- （3）智能灯泡根据接收到的消息修改状态
  - 由于智能灯泡订阅了Topic2，就可以接收消息更改状态



# 规则引擎 - 案例

- 为什么不直接发送至Topic2?

  - 1. 用户无需关心具体的指令格式
  - 2. 更好的安全性

# 时序数据库

- **时序数据（Time Series Data）：带有timestamp的数据**
  - 例：某工厂温度传感器每秒采集的室内温度

- **时序数据库（Time Series Database)**
  - 高效存取时序数据的数据管理系统

- **传统关系型数据库管理时序数据的问题**
  - 写入吞吐低：对频繁插入数据的效率不高
  - 存储成本大：对于时序数据压缩不佳
  - 维护成本高
  - 查询性能差

# 典型时序数据库

- 开源产品
  - InfluxDB
  - KairosDB
  - OpenTSDB
  - Beringei（Facebook）
  - TimeScaleDB

- AWS Timestream
- 阿里云TSDB

# 时序数据库- 特点

- **没有更新操作**
  - 一般只支持插入查询删除操作
- **支持高并发写入**
  - 时序数据库大部分时间都在写入
  - 借助LSM树，先写内存再顺序写入磁盘
- **低查询延时，高查询并发**
  - 分布式查询
  - 数据预处理：预先计算，查询时直接返回少量运算结果
- **存储成本低**
  - 分级存储：内存，固态硬盘，机械硬盘
  - 合理选择编码压缩算法，提高数据压缩比

# 流计算

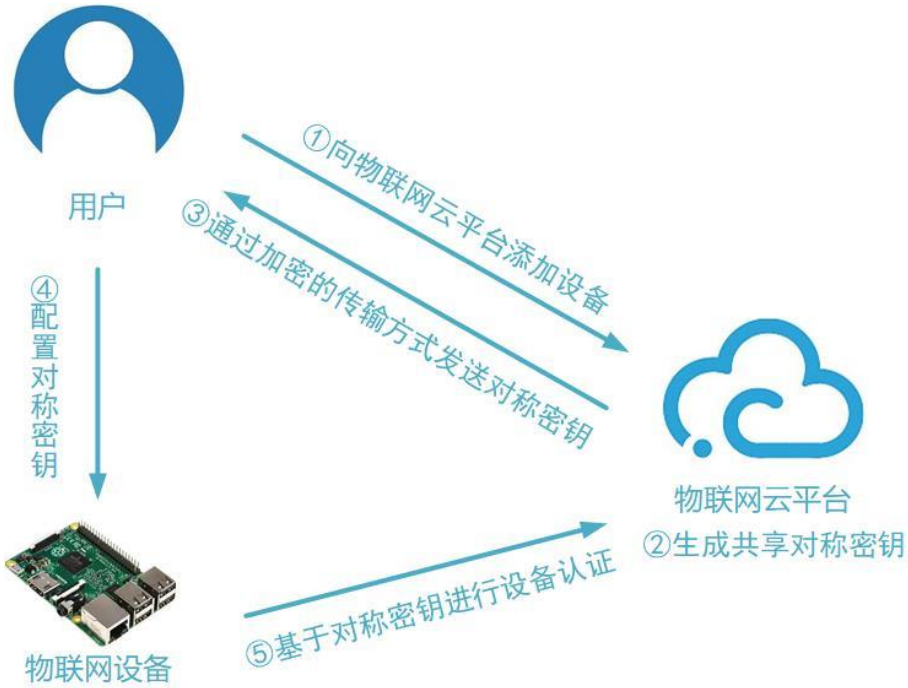
- **流计算**
  - 严格的定义：对数据流进行一次扫描，并对其特征进行计算
  - 一般的定义：流算法只对每个数据在短时间内做数次简单计算，流算法使用的内存空间通常较少
- **流计算的实际案例**
  - 高速路由器上数据流特征统计
  - 金融数据的实时分析
- **是否存在高效的流算法?**
  - 计算数据流的平均数？
  - 计算数据流的中位数？
  - 1 – 100 （无重复）缺了2个数，如何快速找出两个数？

# 典型流计算框架

- Apache Flink
- Apache Storm (Twitter)
- Apache Kafka (LinkedIn)
- Apache Samza
- Blink
- Spark streaming
- Heron

# 设备认证

- **基于对称密钥的设备认证**
  - 云平台和设备共享对称密钥
  - 物联网云平台在添加设备时，生成密钥，用户在编写设备端代码时，将密钥配置在设备上
  - 在实际使用中，设备端通过令牌与云平台连接，令牌由设备密钥对设备标识、随机数或时间戳等加密生成



# 设备认证

- 基于非对称密钥的设备认证
  - 设备使用私钥进行签名，云平台获取公钥用于认证
- 为了管理公钥，使用基于X.509证书的PKI机制

**A possible certificate and its signed hash.**

$P_B \| D_{CA}(SHA(P_B))$

I hereby certify that the public key
19836A8B03030CF83737E3837837FC3s87092827262643FFA82710382828282A
belongs to
Robert John Smith
12345 University Avenue
Berkeley, CA 94702
Birthday: July 4, 1958
Email: bob@superdupernet.com

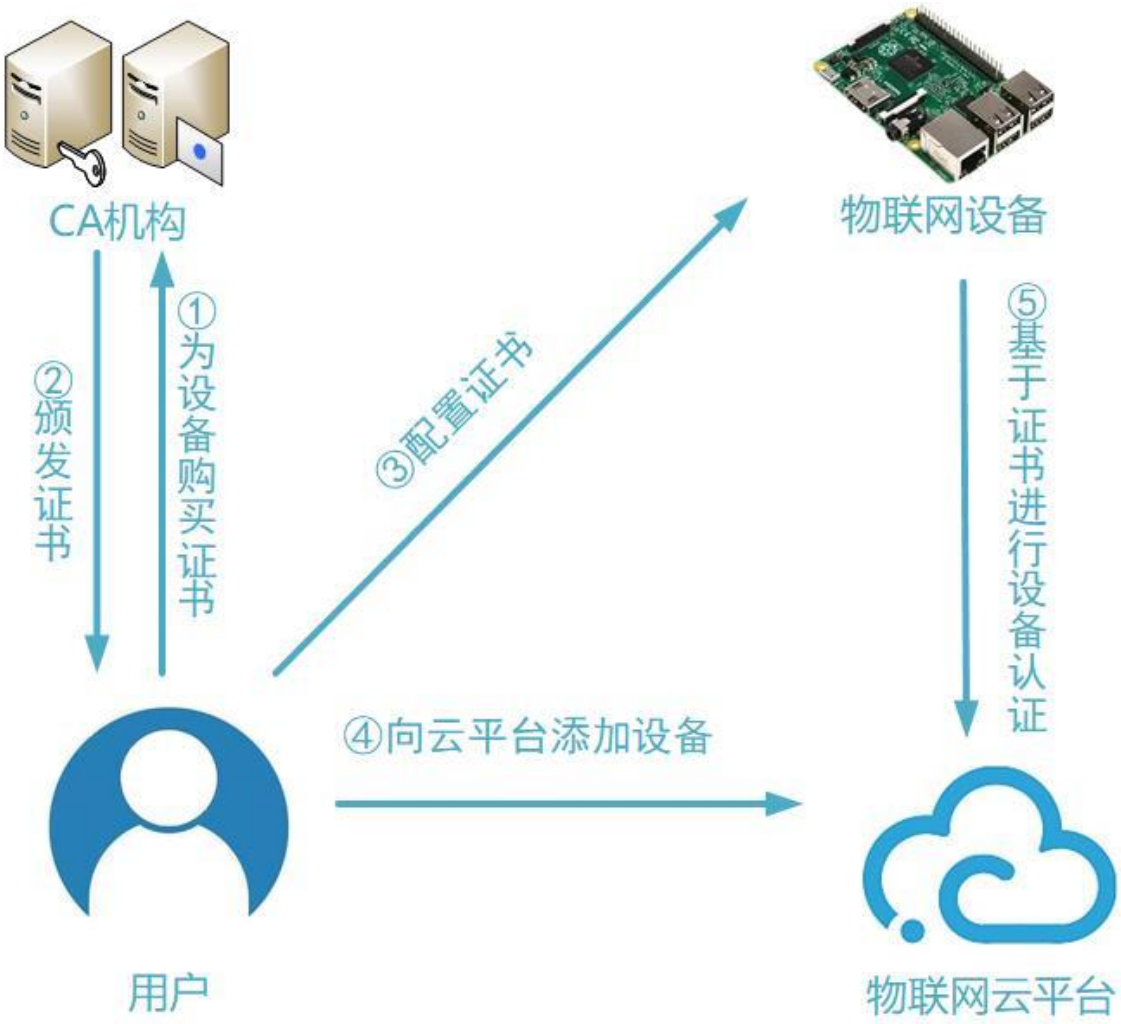
SHA-1 hash of the above certificate signed with the CA's private key

# 设备认证

- X.509是否会被攻击?
- What Trudy can do?
- 1. put $P_T \| D_{CA}(SHA(P_T))$ on fake page. Alice will find since $P_T$ says it is Trudy!
- 2. modify $P_B$ to $P_{B2}$ by replacing Trudy's public key. But Trudy cannot generate the signed block: $P_{B2} \| D_{CA}(SHA(P_B))$. Alice will find mismatch!
  - $SHA(P_{B2}) \mathrel{!=} E_{CA}(D_{CA}(SHA(P_B)))$.

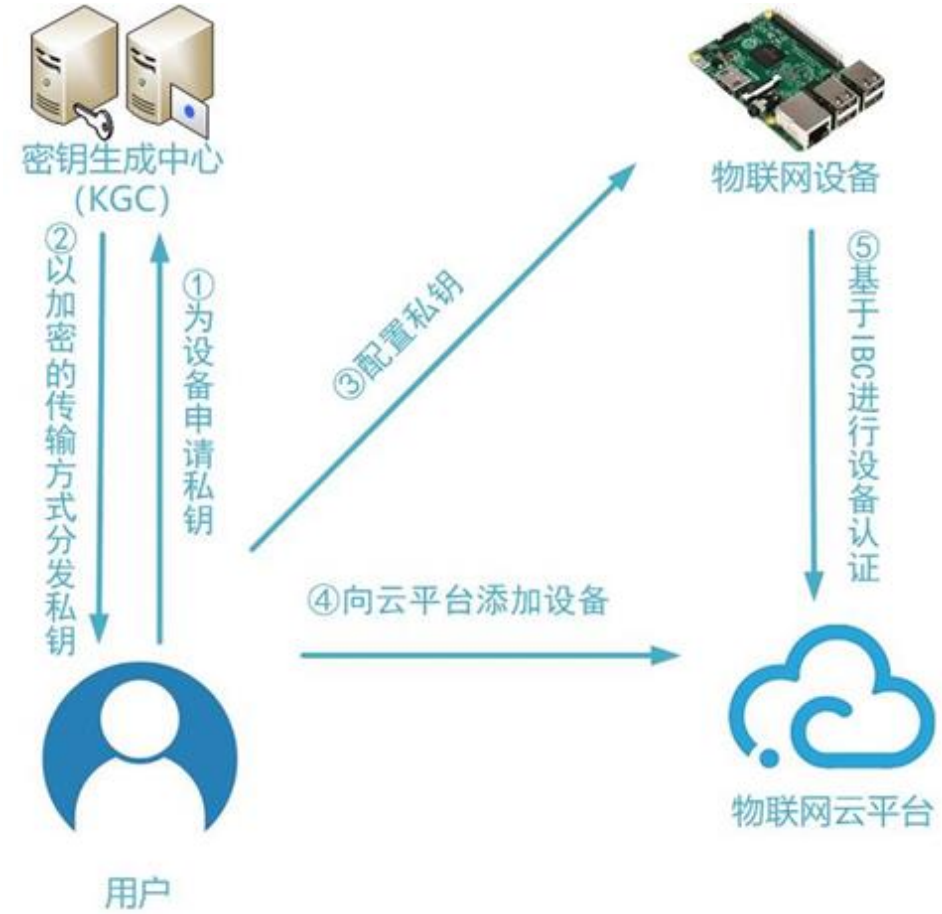
# 设备认证

## ❑ X.509证书的使用



# 设备认证

- IBC（Identity-Based Cryptograph）
  - 将设备标识作为认证过程的公钥
- IBC的使用



# 设备认证

- IBC特点
  - IoT云平台一般知道设备标识
  - 设备标识（公钥）无需PKI管理维护
  - 仅支持IoT设备<->云平台的认证

# 权限认证

- **基于账号和密码的权限认证**
  - 用户或应用程序提供账号和密码与云平台进行连接
  - 云平台赋予用户或应用程序预先分配的权限

- **基于令牌的权限认证**
  - 用户和应用程序通过令牌与云平台进行连接，获取权限
  - 令牌的生成过程融入了权限策略
    - *例如，用户端可生成包含权限密钥签名值字段、权限策略名称字段的令牌用于权限认证*

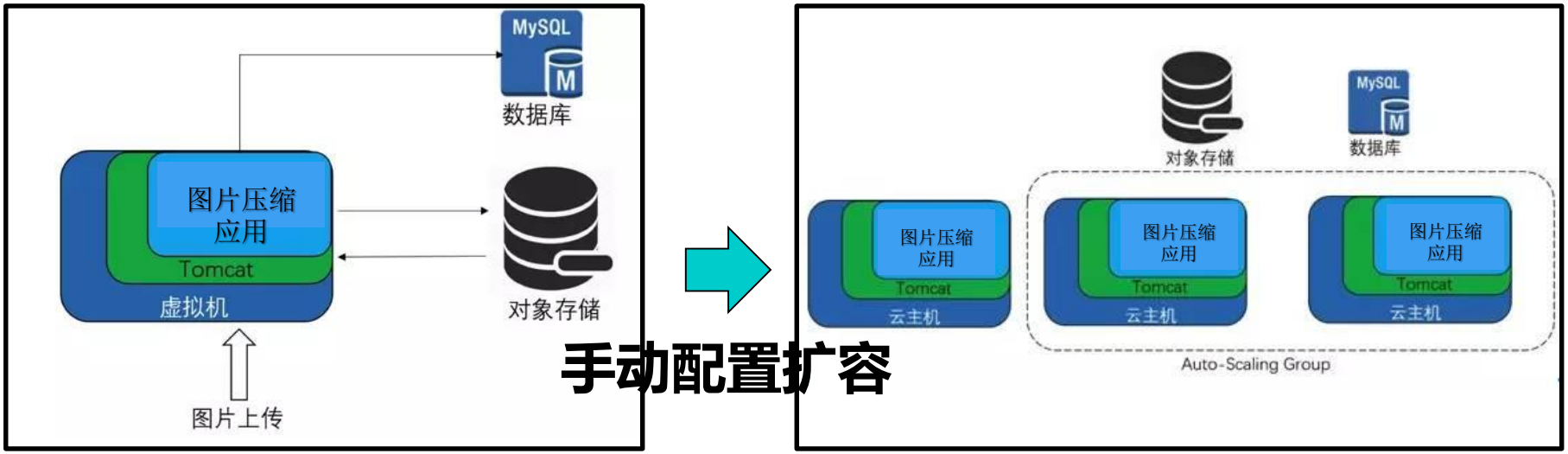
# 无服务器计算

- **什么是无服务器计算?**
  - 无服务器计算(Serverless Computing) 是一种全托管计算服务，免去人工管理、运维，例如管理服务器等基础设施的麻烦，用户唯一需要做的只是编写代码上传。
    - *事件驱动的编程模型*
    - *云服务器自助分配管理计算所需要的资源并保证计算的效率和准确性*
    - *服务提供商提供日志查询、性能监控等功能*
- **它和传统PaaS模式构建应用有何区别?**

  **我们用一个例子来看:**
  - 物联网场景中经常产生许多图像数据上传至云服务器进行分析。但是直接存储原分辨率图片既占用存储资源又降低了智能分析的响应速度。
  - 因此，我们要设计一个图片压缩应用，它接受用户上传的图片，生成其缩略图并存入云数据库。

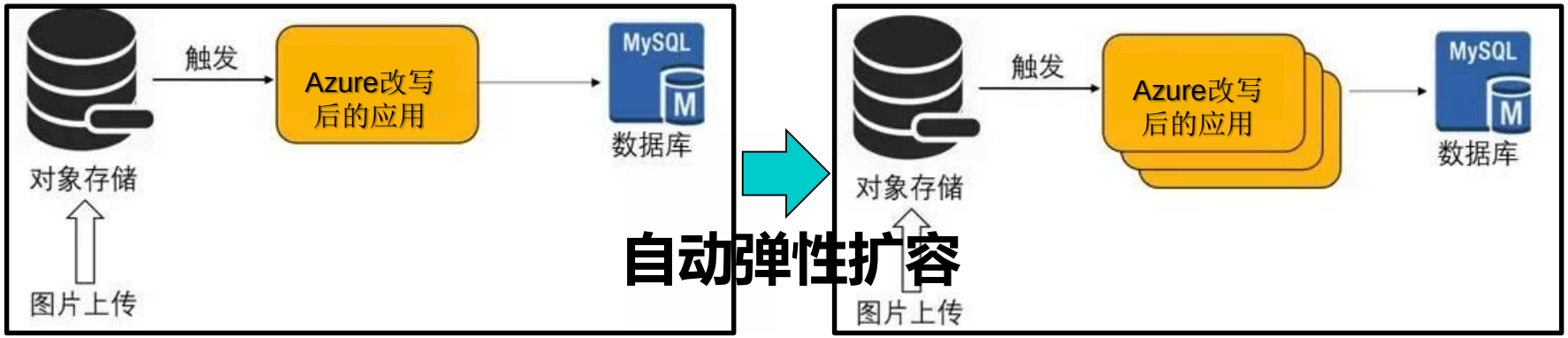
# 无服务器计算

- 无服务器计算模式构建应用 v.s. 传统PaaS模式构建应用
- 传统PaaS模式构建应用流程：
  - 购买/ 管理云主机：计算/网络资源、运维策略、安全补丁等
  - 运行时与依赖的解决
  - 使用云平台的对象存储和数据库服务
  - 编写应用逻辑
  - 手动配置公共云的AutoScaling策略，应对高峰访问量



# 无服务器计算

- 使用无服务器计算构建应用：
  - 使用云平台的对象存储和数据库服务
  - 编写应用逻辑




- 无服务器计算模式构建应用 v.s. 传统PaaS模式构建应用

| | 语言支持 | 运行时环境 | 安装依赖 | 运维 | 计费方式 |
|---|---|---|---|---|---|
| 无服务器计算 | 较少 | 自动搭建 | 受限 | 无需管理 | 按代码运行时间计费 |
| PaaS | 丰富 | 手动搭建 | 可任意手动安装 | 需要管理 | 按实例存在时长计费（代码空闲时可能仍在计费） |

# 无服务器计算

- **无服务器计算优势：**
  - 工程效率提升：
    - *借助函数计算和其他云端服务的集成，用户只需要编写少量代码就可以串联多个服务完成复杂的功能。用户不需要负担服务器配置/升级，资源扩容/缩容等繁琐工作，运维成本大幅降低*
  - 可靠性提升：
    - *用户不需要购买或维护任何的服务器，不需要安装任何的软件，系统自动处理基础设施层各种错误，减少宕机、网络中断等危机情况发生，提升系统可靠性。*
  - 弹性伸缩：
    - *应用程序所需计算资源的伸缩能在毫秒级别由无服务器计算平台自动完成，用户不需要负责增减服务器的数量。*
  - 降低成本：
    - *研究数据表明，70%以上的服务器资源并未得到充分利用，函数计算采用无服务器计算模式，资源使用率达100%，通常能节省10%-90%的成本。*

# 无服务器计算

## 典型无服务器计算服务

- Microsoft Azure Functions
- Amazon AWS Lambda
- 阿里云函数计算
- Google Cloud Function

## 各平台语言支持

| | Node.js | Python | Java | C# | PHP | JavaScript | F# |
|---|---|---|---|---|---|---|---|
| Azure Function | | | √ | √ | | √ | √ |
| AWS Lambda | √ | √ | √ | √ | | | |
| 阿里云函数计算 | √ | √ | √ | | √ | | |
| Google Cloud Func. | √ | | | | | | |

# 无服务器计算

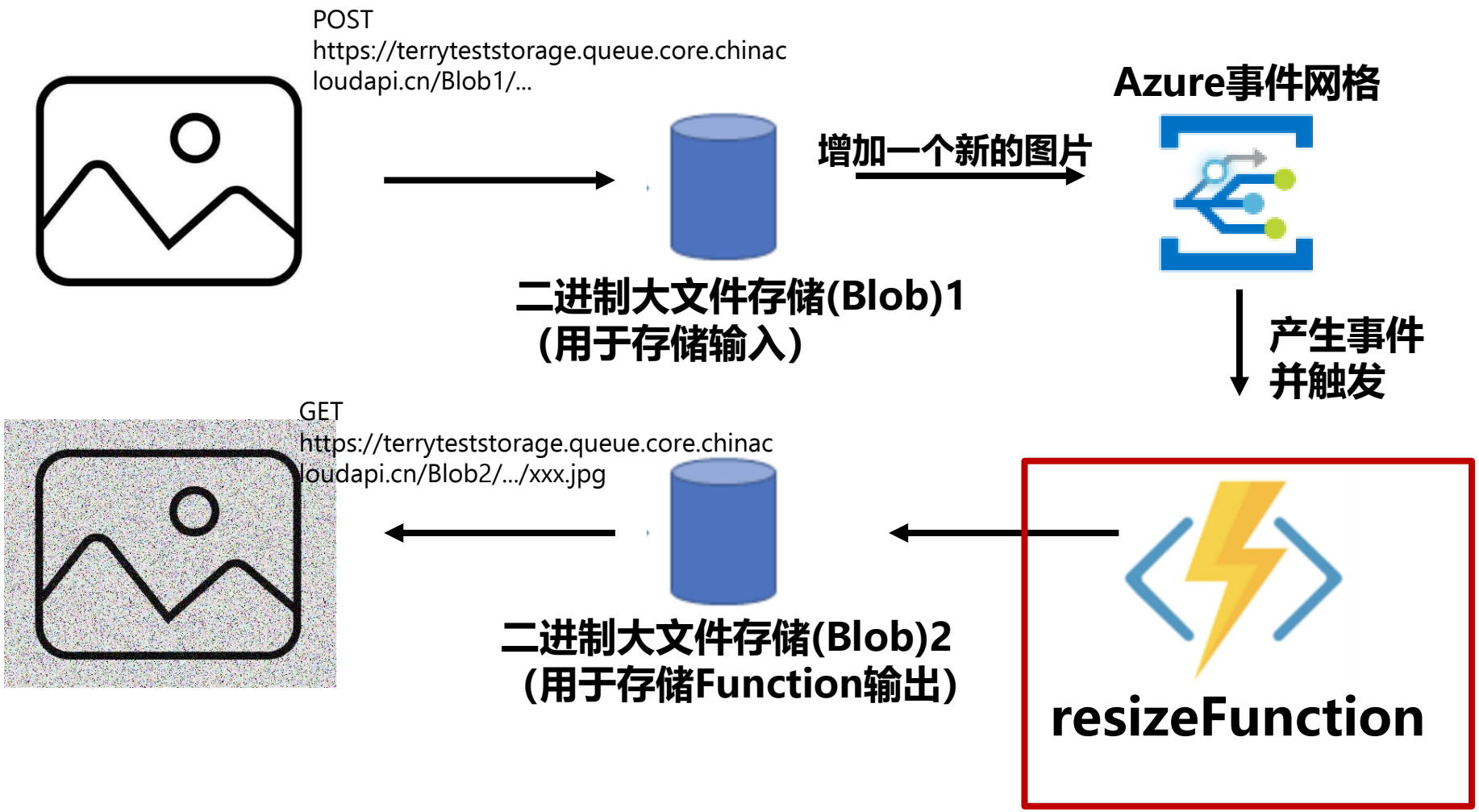
- **常见的几种使用无服务器计算实现的功能**
  - 可以使用HTTP URL形式访问的无服务器云端API
    - ***E.g.** 号码归属地查询、API地址查询、汇率转换、在线分词等*
  - 基于事件触发的服务
    - ***E.g.** 基于对象存储的更新事件触发的图像缩略图生成*
  - 定时触发的服务
    - ***E.g.** 定时刷新微博并对新微博进行情感分析*
  - 无服务器Web应用
    - ***E.g.** 实现一个Web服务器*
      - ✓主要Web服务使用无服务器计算模式搭建
      - ✓静态资源的存储使用对象存储服务

# 无服务器计算– 案例

- 以本节初的图片缩略图应用在Azure Function上的部署为例
- Azure Function
  - 支持语言：C#, F#, Java, JavaScript
- 付费方式：仅为运行Function时的时长和资源收费
  - 费用=资源消耗(GB·s) 费用+ 执行次数(次)费用
  - 每月有400000GB·s的资源消耗和一百万次执行是免费的
  - 例：观察到内存消耗为 512 MB 的函数在本月执行 3,000,000 次，执行持续时间为 1 秒。每月费用为：
    - *资源消耗：3百万次*1s*512/1024GB-400000GB·s=1.1百万GB·s*
    - *执行次数：3百万次-1百万次=2百万次*
    - *1.1百万GB·s*$0.000016+2百万次*$0.0000002=$18*

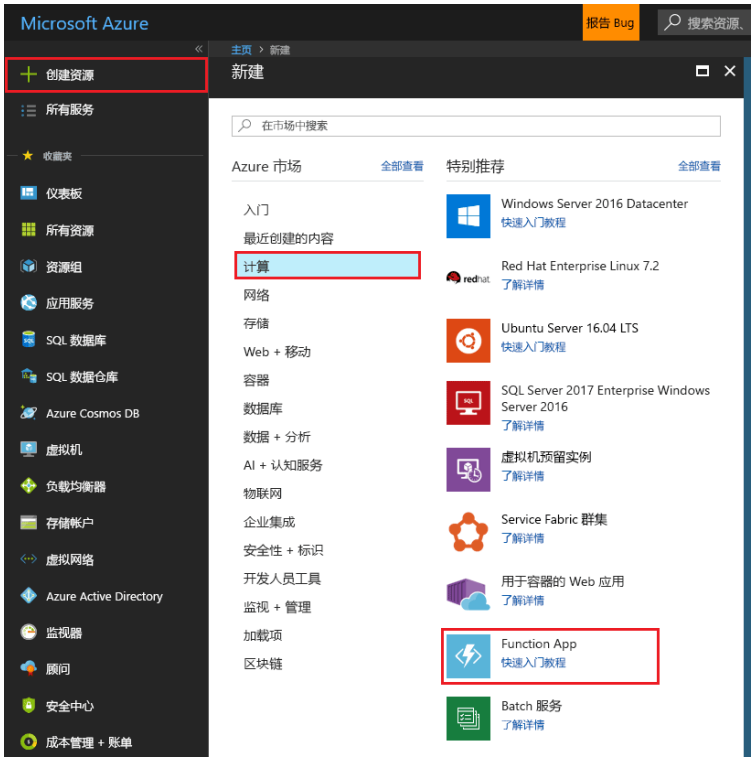
# 无服务器计算– 案例

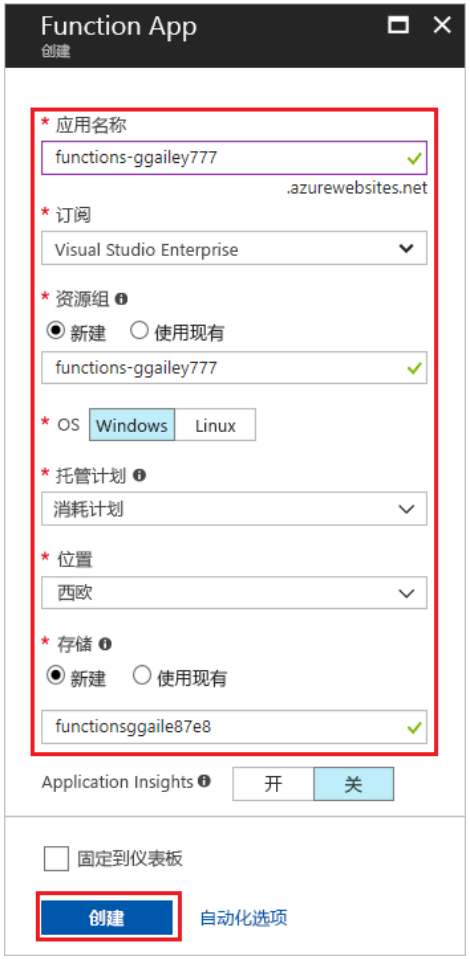
## ❑ 缩略图应用云端逻辑



# 无服务器计算– 案例

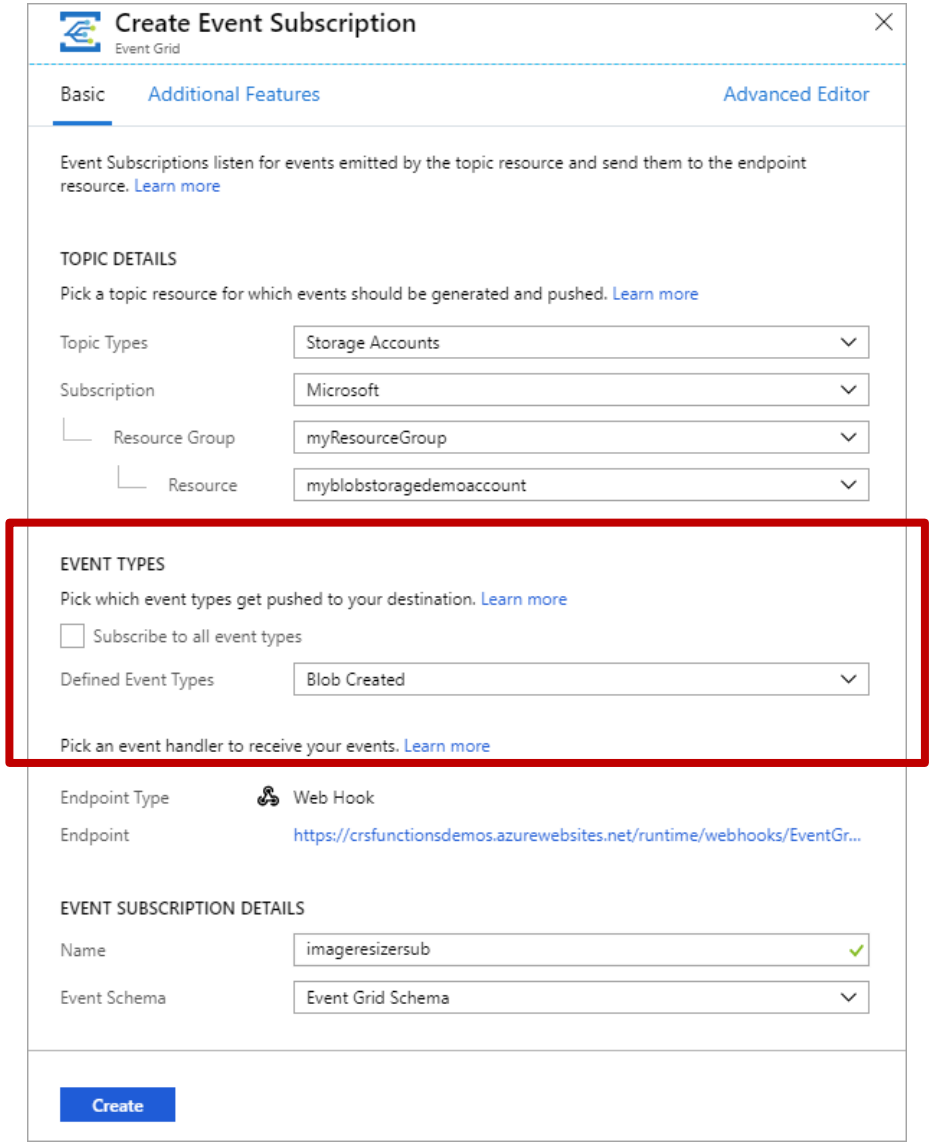
- 创建Azure Function（Azure网站方式）
  - 除Azure网站方式外，还可以使用VS、VS Code、Python、Java/Maven、Linux CLI等方式创建Function



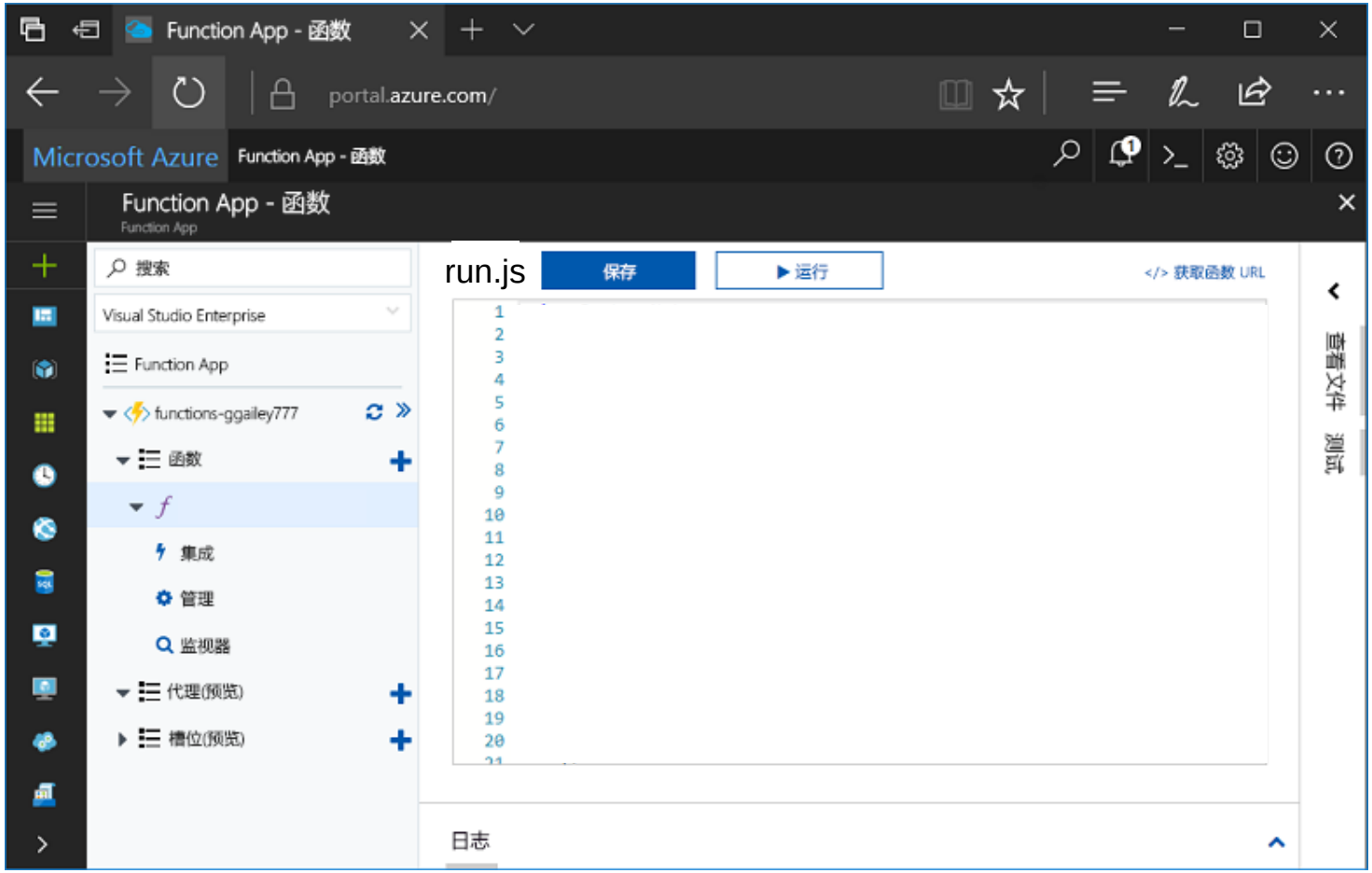
# 无服务器计算– 案例

- 创建事件：当Blob存储更新时触发事件“myEvent”



# 无服务器计算– 案例

## ❑ 编写Function代码



# 无服务器计算– 案例

```
FunctionsProject
| - MyFirstFunction
| | - index.js
| | - function.json
```

- Azure Function主要由两部分组成：
  - 配置文件.json和函数文件.js
  - 本例json为：

```
{
 "disabled": false,
 "bindings": [
  {
   "type": "eventGridTrigger",
   "name": "myEvent",
   "direction": "in"
  },
  {
   "name": "myBlob",
   "type": "blobTrigger",
   "direction": "in",
   "path": "images/{name}",
   "connection": "AzureWebJobsStorage",
   "datatype": "binary"
  }
 ]
}
```

设置该函数为打开

与事件网格中的Event绑定

与Blob存储中的实例绑定
将此实例命名为myBlob
后面JavaScript代码会用到

*红色标注为用户自定义的字段

# 无服务器计算– 案例

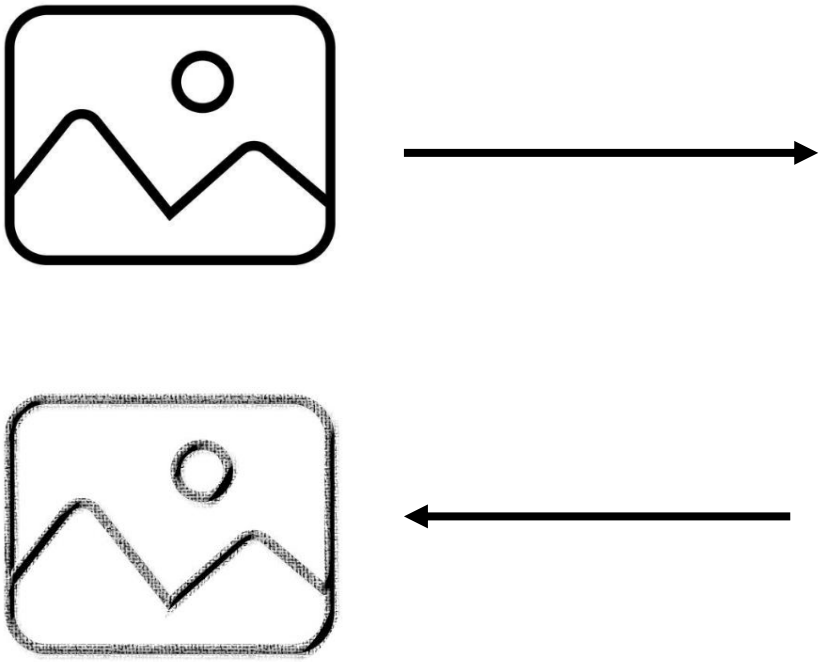
```
FunctionsProject
| - MyFirstFunction
| | - index.js
| | - function.json
```

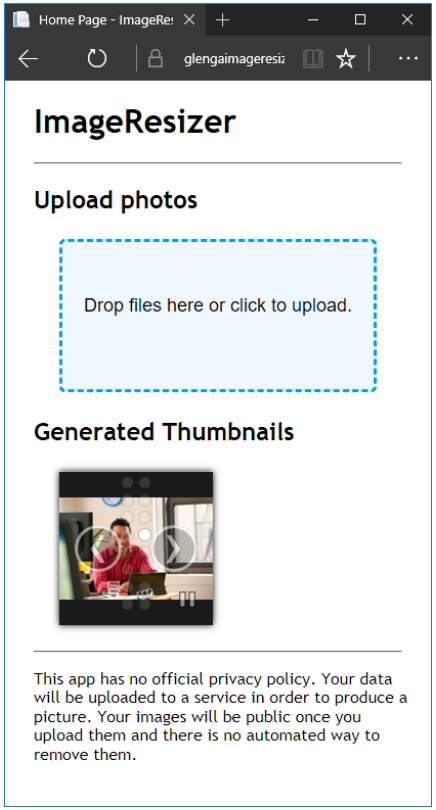
## ❑ 生成图片缩略图Function代码（JavaScript)

```
const stream = require('stream’);
const Jimp = require('jimp’);                                   //JIMP:JavaScript图像处理库
const storage = require('azure-storage’);                        //引入Blob存储
const blobService = storage.createBlobService();
module.exports = (context, myBlob) => {    //入参: context和json中定义的输入名称myBlob
  const widthInPixels = 100;                                     //设定图片压缩参数
  const blobName = context.bindingData.name;                     //绑定Blob存储
  const contentType = context.bindingData.properties.contentType;
  Jimp.read(myBlob).then((thumbnail) => {
        thumbnail.resize(widthInPixels, Jimp.AUTO);              //生成缩略图
        const options = { contentSettings: { contentType: contentType } };
        thumbnail.getBuffer(Jimp.MIME_PNG, (err, buffer) => {
          const readStream = stream.PassThrough();               //存储缩略图
          readStream.end(buffer);
          blobService.createBlockBlobFromStream('thumbnails', blobName, readStream,
buffer.length, options, (err) => {context.done(); });
        });
  }).catch(context.log);
};
```

# 无服务器计算– 案例

- 至此，图片缩略图生成应用的Azure Function已经构建完成。
  - 可以通过直接将图片上传至Blob存储进行测试
  - 或进一步如右例一样编写一个上传网页，调用Blob存储提供的API进行上传测试





# 无服务器计算

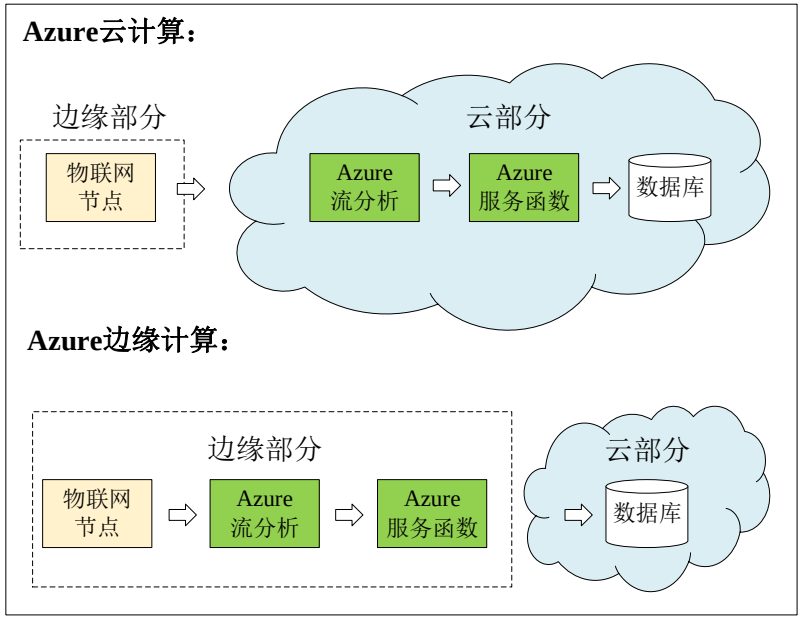
- 问题
  - 云服务供应商锁定
    - *云计算供应商之间支持的语言、提供的服务均各有特色，如果更换运行商则需要很大的迁移成本*
  - 服务的冷启动
    - *在距离上一次调用已有一段时间的情况下，函数调用可能需要很长时间才能做出响应*
  - 工具链的缺少
    - *监控、日志记录、开发和调试工具还不成熟*

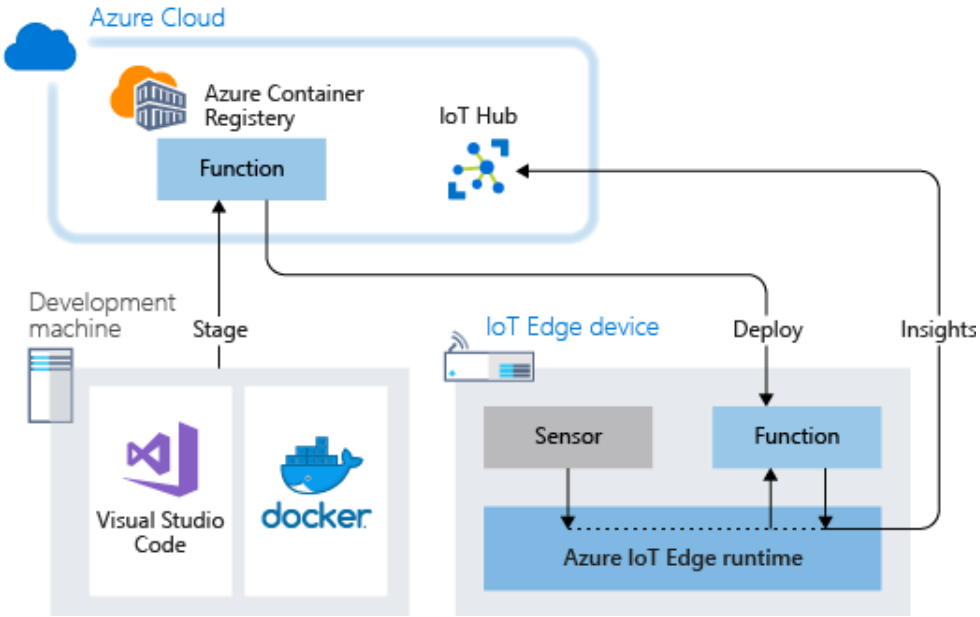
# 无服务器计算的下沉 – 边缘计算

- **边缘计算的定义**
  - 在近用户端为用户提供使用无线接入网络的IT和云计算功能的技术。
- **下沉方法**
  - **RPC**
    - *轻量，但是RPC服务代码需要预置。*
  - **容器**
    - *依赖底层宿主机的内核。*
    - *轻量级。Azure IoT Edge 、AWS Greengrass、阿里云Link Edge均采用此方式*
  - **虚拟机**
    - *相较于容器技术，需要网络分发的数据量更大*
    - *更加灵活，屏蔽宿主机操作系统的异构性*
  - **OTA**
    - *整机的整体软件更新，开销大*

# 边缘计算

- 我们以Azure的边缘计算服务架构为例进行说明
  - 从图中可以看到，传统的物联网云计算模式都是把物联网节点的数据直接上传至云端进行分析与存储
  - 但是在边缘计算模式下，数据的初步分析可以下发到边缘侧的路由器或是设备进行计算。



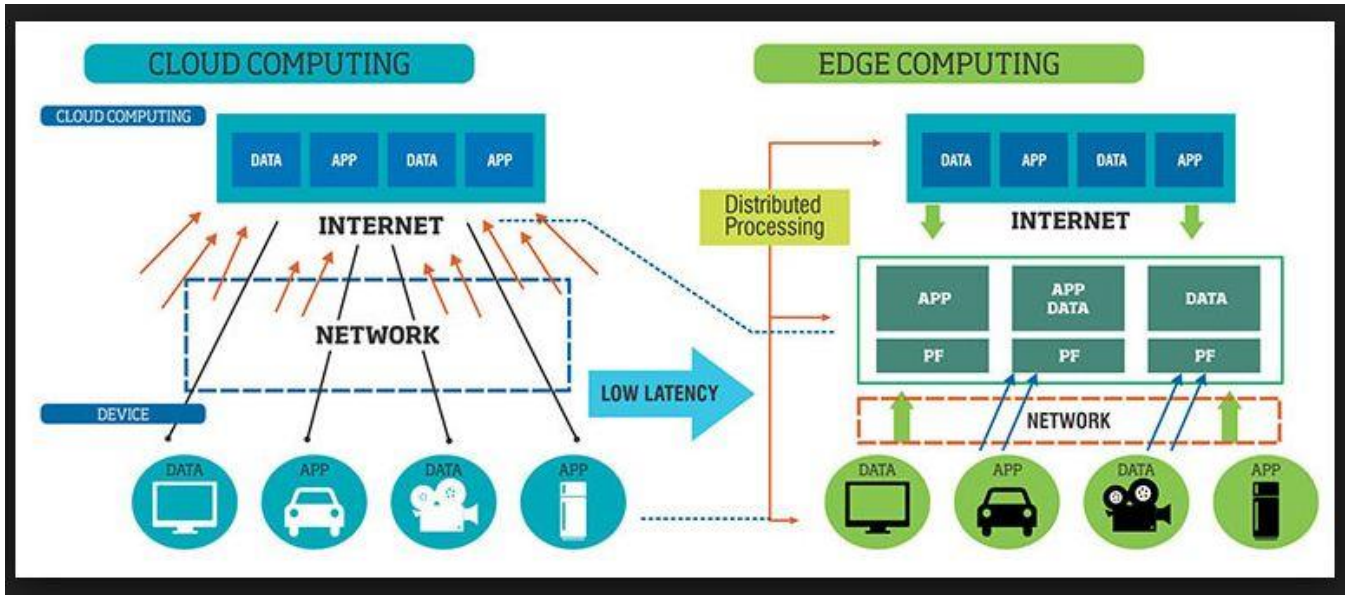



# 边缘计算

- 科技厂商纷纷布局边缘计算
  - Amazon – Greengrass
  - Microsoft – Azure IoT Edge
  - IBM – Watson IoT Platform
  - Aliyun – Link Edge
  - Intel – Intel IoT Platform
  - Linux Foundation – EdgeX Foundry
  - Apache Foundation – Edgent
  - ARM – Mbed Edge
  - ……

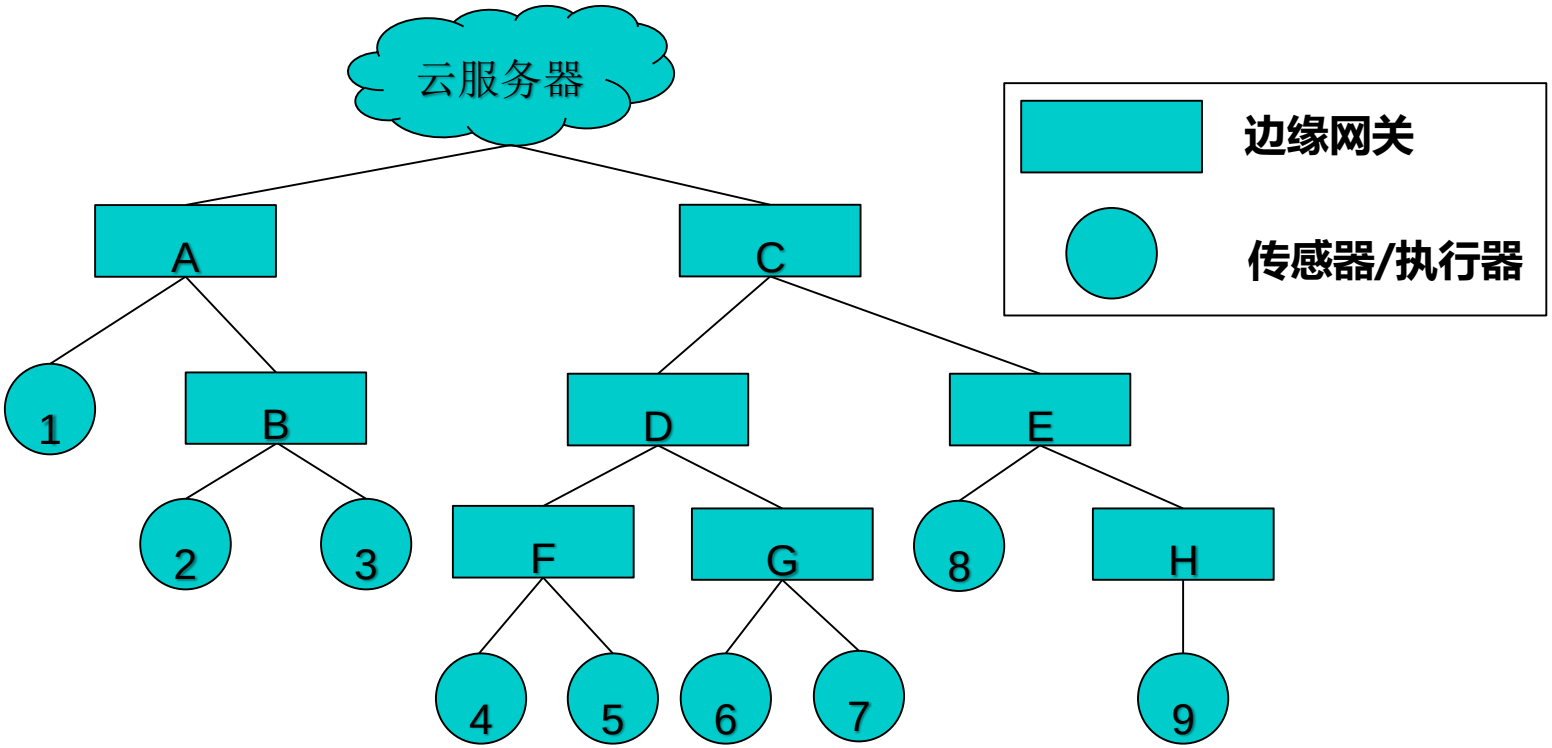
# 边缘计算

- 边缘计算的优势
  - 实时：降低时延，达到高响应的云服务；
  - 节省：将物联网数据在边缘端即时处理，节省骨干网带宽；
  - 安全：数据分析不出户，边缘端处理数据，保证物联网数据的安全与隐私性；
  - 稳定：主干网断网仍可分析，具有较高的可用性。



# 计算下沉

- 计算下沉的锚点
  - 函数规则与8、9的功能相关-> E
  - 函数规则与7、9的功能相关-> C
  - 函数规则与1、9的功能相关-> 无法下沉
- =>寻找最近公共父节点问题



# 边缘计算

## ❑ 厂商对比

| 边缘计算平台 | 公司 | 提出时间 | 安全性 | 文档 | 支持中国区域 | 价格 | 底层技术 | AI |
|---|---|---|---|---|---|---|---|---|
| AWS Greengrass | 亚马逊 | 2016.12 | 提及 | 完善 | 否 | 按设备计费 | 容器 | 支持（ML Interface），可与主流机器学习框架配合使用 |
| Azure IoT Edge | 微软 | 2017.5 | 提及 | 完善 | 是 | 免费 | 容器 | 支持（AI Toolkit） |
| Watson IoT Edge Platform | IBM | 2016.7 | 提及 | 完善 | 是 | 按流量计费 | 容器 | 提及 |
| Link Edge | 阿里巴巴 | 2018.3 | 提及 | 完善 | 是 | 免费（限制设备数与消息数） | 容器 | 暂无 |

# 目录

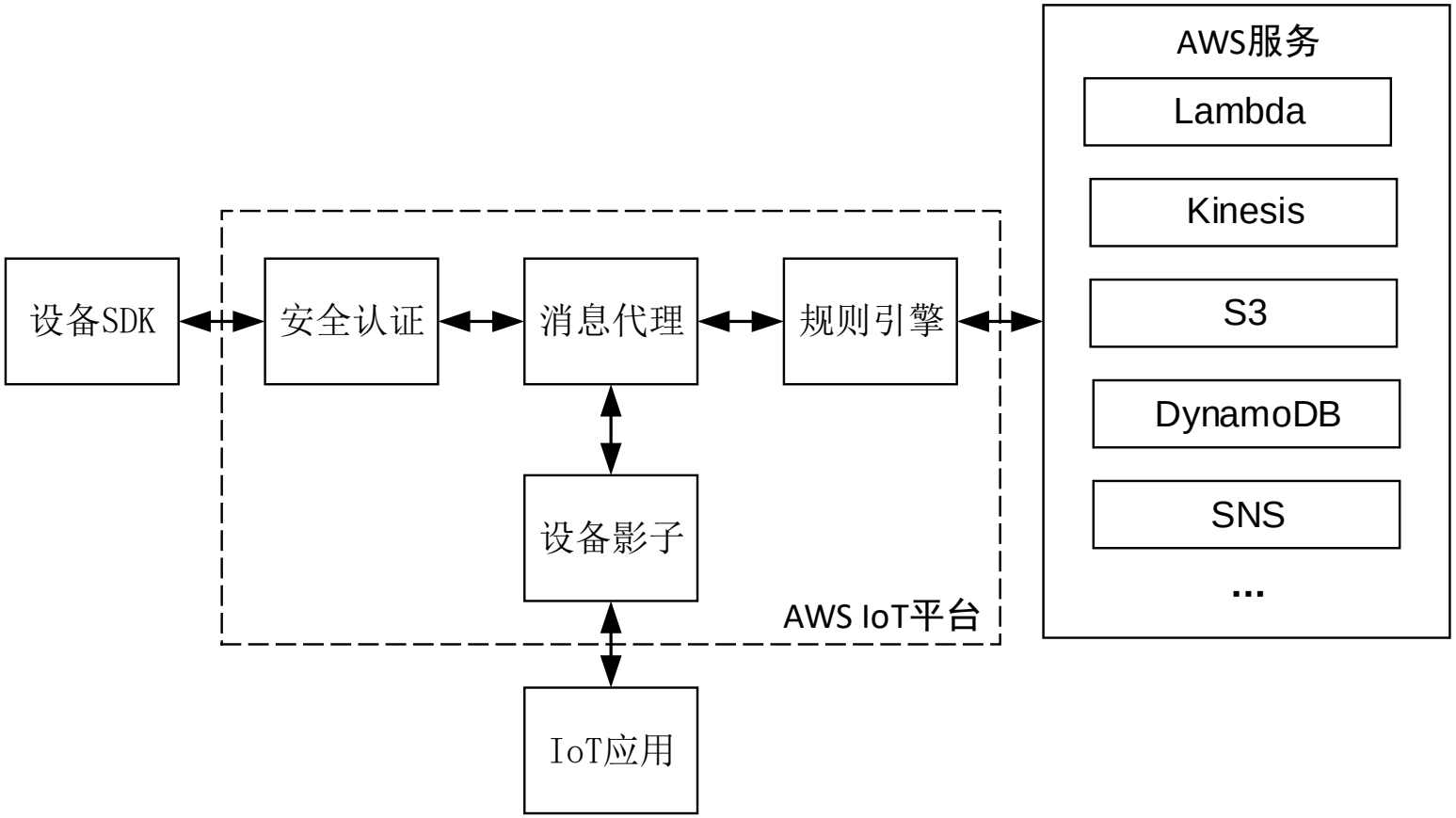
- 物联网云平台概述
- 物联网云平台特色功能与实例
- 主要的物联网云平台
  - Amazon AWS IoT
  - Microsoft Azure IoT
  - 阿里云IoT
- 总结与展望

# Amazon AWS IoT概述

AWS IoT

- 时间点
  - 2002年：AWS推出，先发优势，深耕国际市场
  - 2015年10月：AWS IoT开放测试
  - 2015年12月：AWS IoT正式商用
- 概述
  - AWS IoT是一款托管的云平台，使互联设备可以轻松安全地与云应用程序及其他设备交互
  - AWS IoT可支持数十亿台设备和数万亿条消息，并且可以对这些消息进行处理并将其安全可靠地路由至 AWS 终端节点和其他设备
  - 使用AWS丰富的服务来构建IoT应用程序，能够收集、处理和分析互连设备生成的数据并对其执行操作，且无需管理任何基础设施

# Amazon AWS IoT架构



# AWS IoT功能概述

AWS IoT

- **设备通信**
  - 支持 MQTT、WebSockets 和 HTTP 1.1 协议
  - 支持设备安全高效地与AWS IoT进行通信
- **设备影子**
  - 可以创建每台设备的“设备影子”，包含设备的最新状态
  - 应用程序或其他设备可以读取消息并与此设备进行交互
  - 可以在设备离线时保证与其他设备的通信
- **规则引擎**
  - 规则引擎评估发布到 AWS IoT Core 的入站消息，根据定义的业务规则转换这些消息并将它们传输到另一个设备或云服务

# AWS IoT功能概述

AWS IoT

- **身份验证和授权**
  - AWS IoT 在所有连接点处提供身份验证和加密，在没有验证身份的情况下，数据绝不会在设备和 AWS IoT Core 之间进行交换
  - AWS IoT Core 支持 AWS 的身份验证方法 (称为 “SigV4”)、基于 X.509 证书的身份验证和基于客户创建的令牌的身份验证
- **注册表**
  - 为设备创建一个身份并跟踪元数据，如属性和功能
  - 注册表向格式一致的每台设备分配唯一的标识，而不管设备的类型和连接方式为何
- **可利用包括Lambda在内的AWS各个服务**

# AWS IoT特点：Lambda

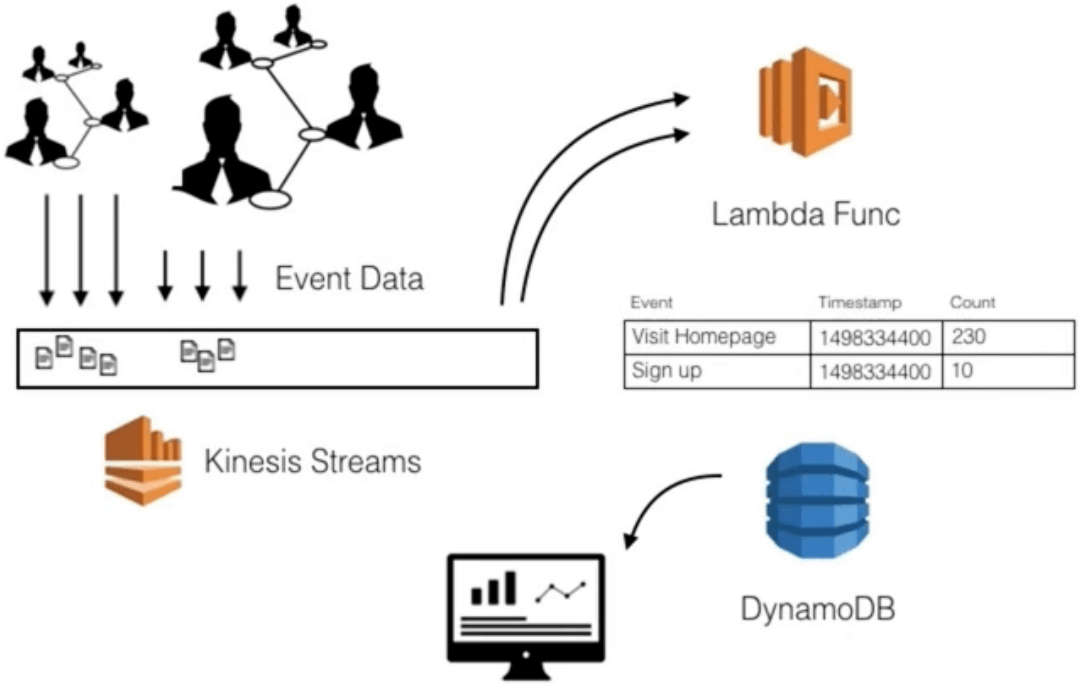
AWS IoT

- **无服务器计算**
  - 无需考虑服务器，即可构建和运行应用程序与服务
  - 应用程序仍在服务器上运行，但所有服务器管理工作均由 AWS 来完成
  - AWS Lambda 是无服务器计算的核心
- **AWS Lambda**
  - 无需配置或管理服务器即可运行代码
  - 持续扩展
    - *AWS Lambda 可通过运行代码自动扩展应用程序*
  - 只需上传代码，Lambda 会处理运行和扩展高可用性代码所需的一切工作

# AWS Lambda案例

AWS IoT

- **案例：图形化展示网站访客信息**
  - 1. 流计算服务Kinesis收集处理原始用户访问数据
  - 2. Lambda 轮询流并在检测到流中的新记录时执行
    - *输入：时间、事件*
    - *输出：将时间、事件、统计的数量存储到DynamoDB中*
  - 3. 根据数据库数据进行可视化展示

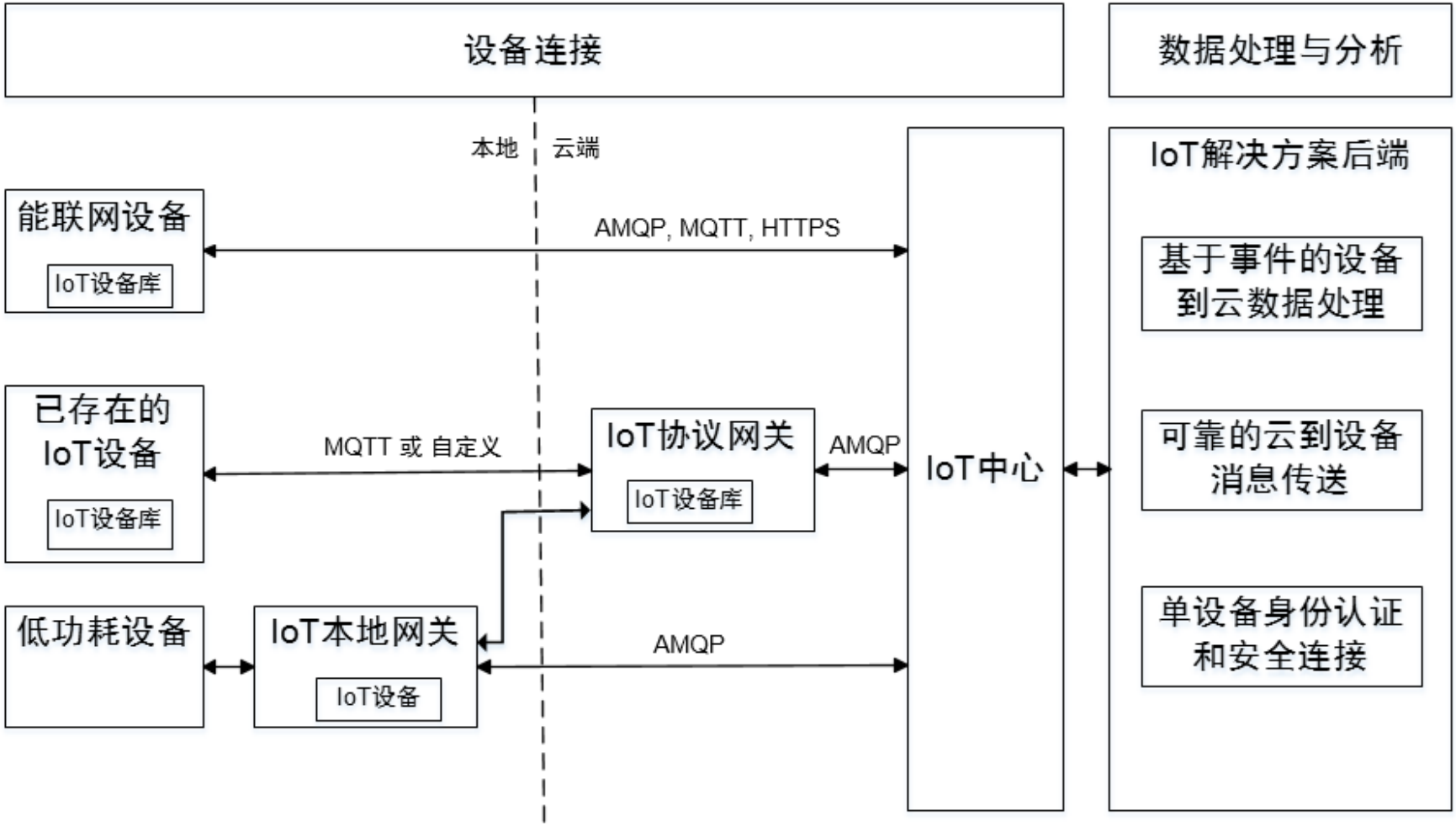


# Azure IoT概述

- Azure IoT是微软提出的物联网云平台

- 平台特点
  - 可用于管理数十亿物联网设备；
  - 提供可靠和安全的云端与设备之间的双向通信支持；
  - 每月可处理数以万亿计消息；
  - 简化了与其他Azure服务之间的集成。

# Azure IoT架构


设备连接
数据处理与分析
本地
云端
能联网设备
IoT设备库
AMQP, MQTT, HTTPS
已存在的
IoT设备
IoT设备库
MQTT 或 自定义
IoT协议网关
IoT设备库
AMQP
IoT中心
IoT解决方案后端
基于事件的设备
到云数据处理
可靠的云到设备
消息传送
单设备身份认证
和安全连接
低功耗设备
IoT本地网关
IoT设备
AMQP

# Azure IoT功能概述

- 设备连接
  - MQTT、HTTPS、AMQP*
- 设备管理
  - 直接管理、设备影子、智能网关
- 数据存储
  - 使用传统数据库
  - Cosmos DB/MySQL/PostgreSQL/MariaDB
- 设备与身份认证
  - 应用层：设备身份认证，客户端/服务器端认证
  - 传输层：HTTPS
  - 网络层：VPN技术（隧道技术和加密技术）

| MQTT<br>消息队列遥测传输 | AMQP<br>高级消息队列协议 |
| --- | --- |
| 轻量级 | 重量级 |
| 不确保安全性 | 使用SASL和TLS确保安全性 |
| 客户端和服务端不对称 | 客户端和服务端对等 |

# Azure IoT功能概述

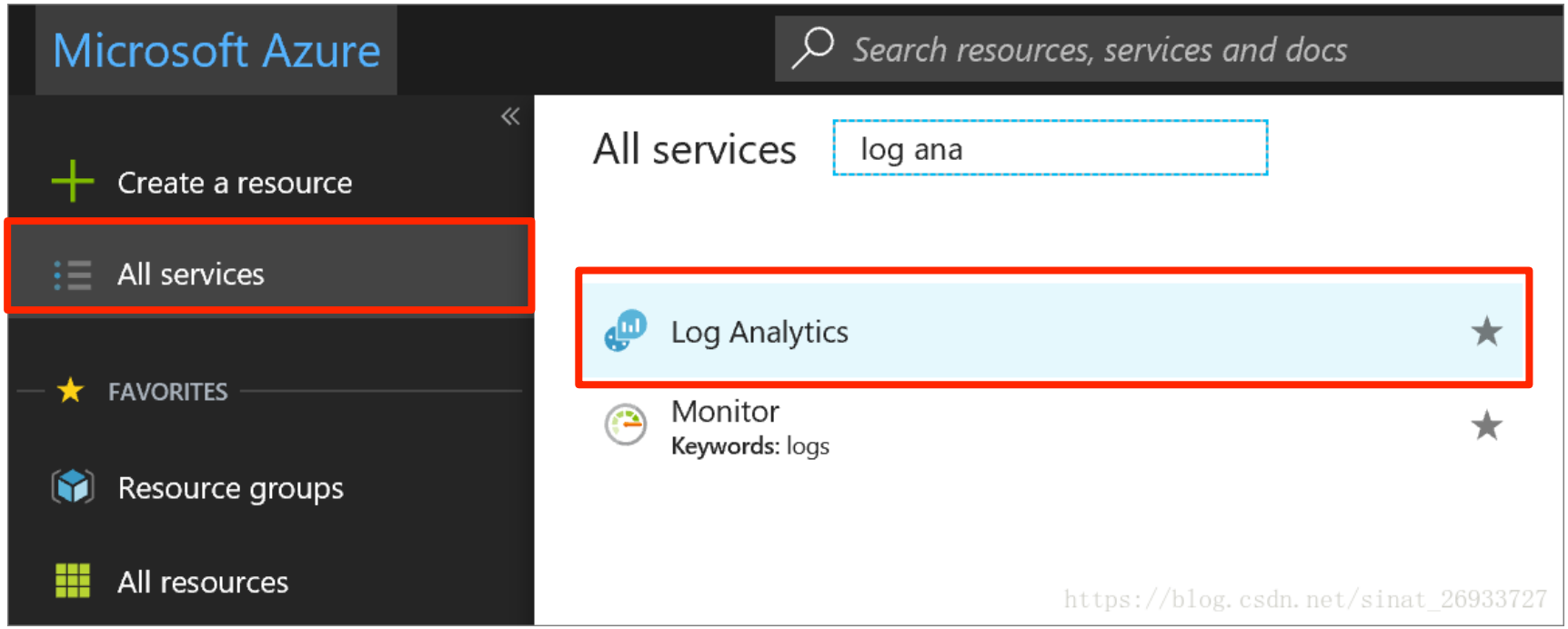
- **无服务器计算**
  - Azure Functions
  - 完全托管的计算平台
  - 提供可根据需要进行缩放的简单无服务器函数
  - 可自行选择编程语言，无需担心服务器或基础结构
  - 轻松生成所需应用
- **边缘计算**
  - 将机器学习、高级分析及AI服务，推向更靠近数据源的网关或IoT设备上
  - Azure IoT中通过无服务器计算下沉实现边缘计算
  - 适应未来需要更多即时分析和处理能力的IoT应用，如自驾车、AR应用等

# Azure IoT特点：机器学习服务

- **机器学习服务**
  - 提供了一个基于云的环境
  - 开发者使用这一环境来开发、培训、测试、部署、管理和跟踪机器学习模型
  - 完全支持开放源代码技术（Python、TensorFlow）
  - 支持丰富的工具，可以交互式地轻松探索数据、转换数据，然后开发和测试模型
  - 自动化模型生成和优化
  - 提供认知服务：是在 机器学习 API 产品组合基础上的拓展，可让开发人员轻松地在其应用程序中添加认知功能
    - 情绪和视频检测
    - 面部、语音与视觉识别
    - 语音与语言理解

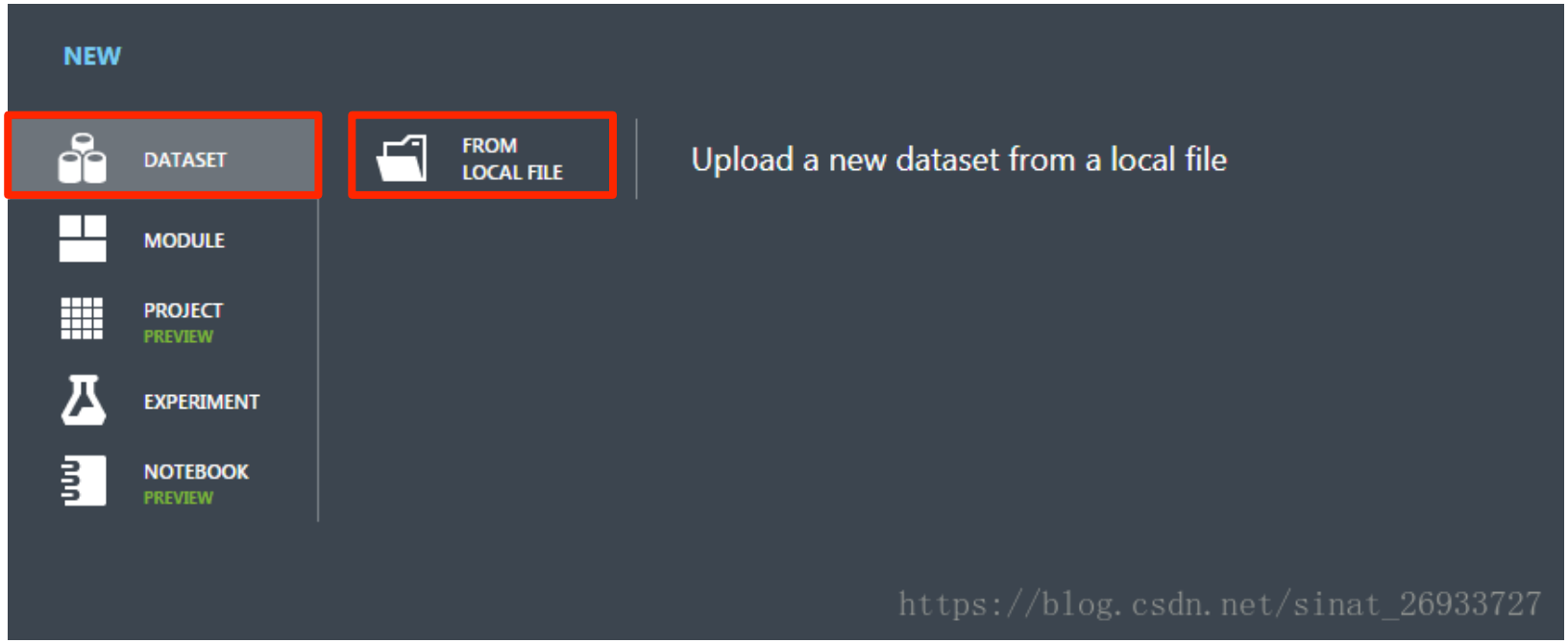
# 机器学习服务案例1：模型选择与训练

- 大致开发流程
  - 1. 创建workspace
    - *选择“All Service”的“Log Analytics”*



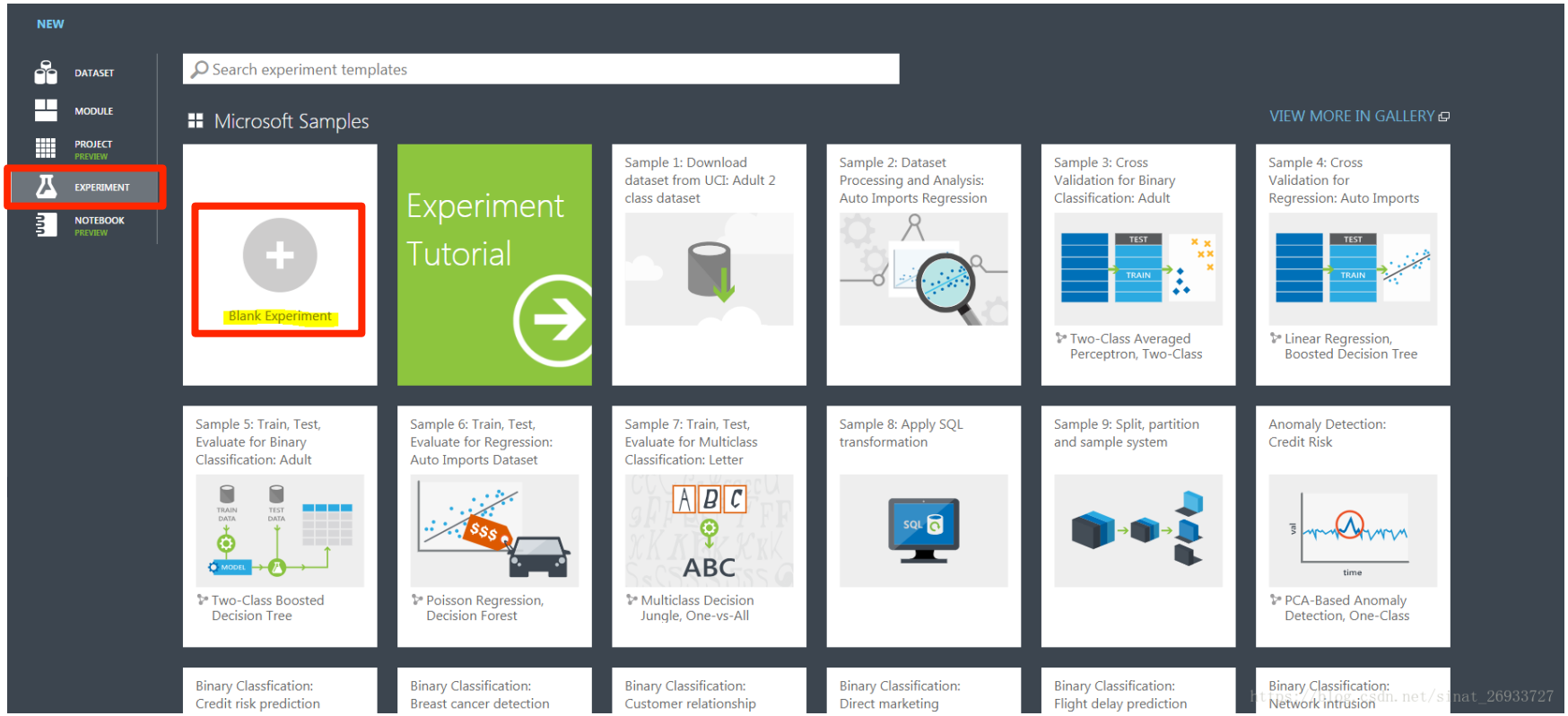
# 机器学习服务案例1：模型选择与训练

- **大致开发流程**
  - 2. 上传数据集
    - *选择“DATASET”的“FROM LOCAL FILE”*



# 机器学习服务案例1：模型选择与训练

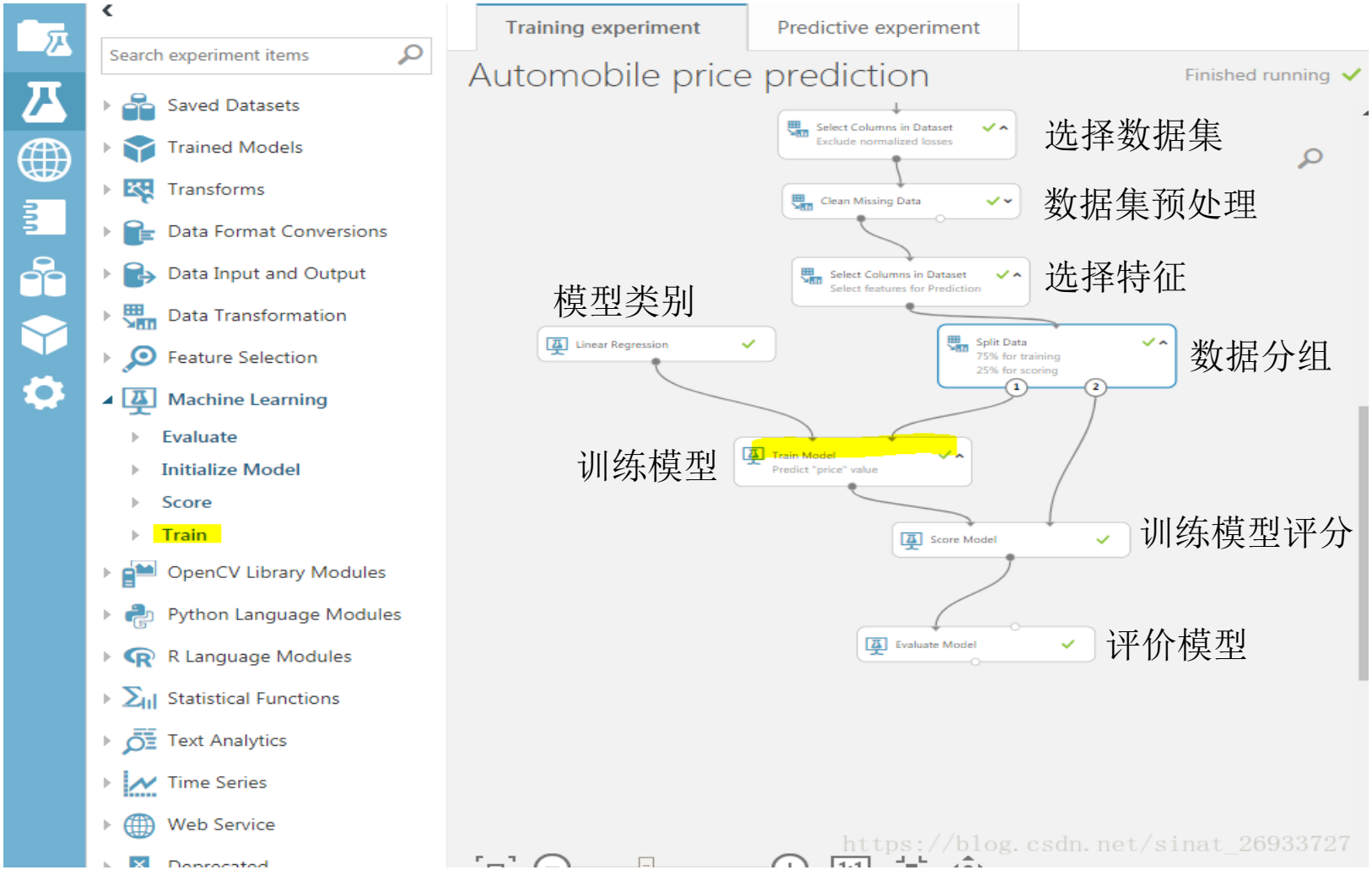
- **大致开发流程**
  - 3. 创建实验
    - *选择“EXPERIENT”的“Blank Experiment”*



# 机器学习服务案例1：模型选择与训练

## ❑ 大致开发流程

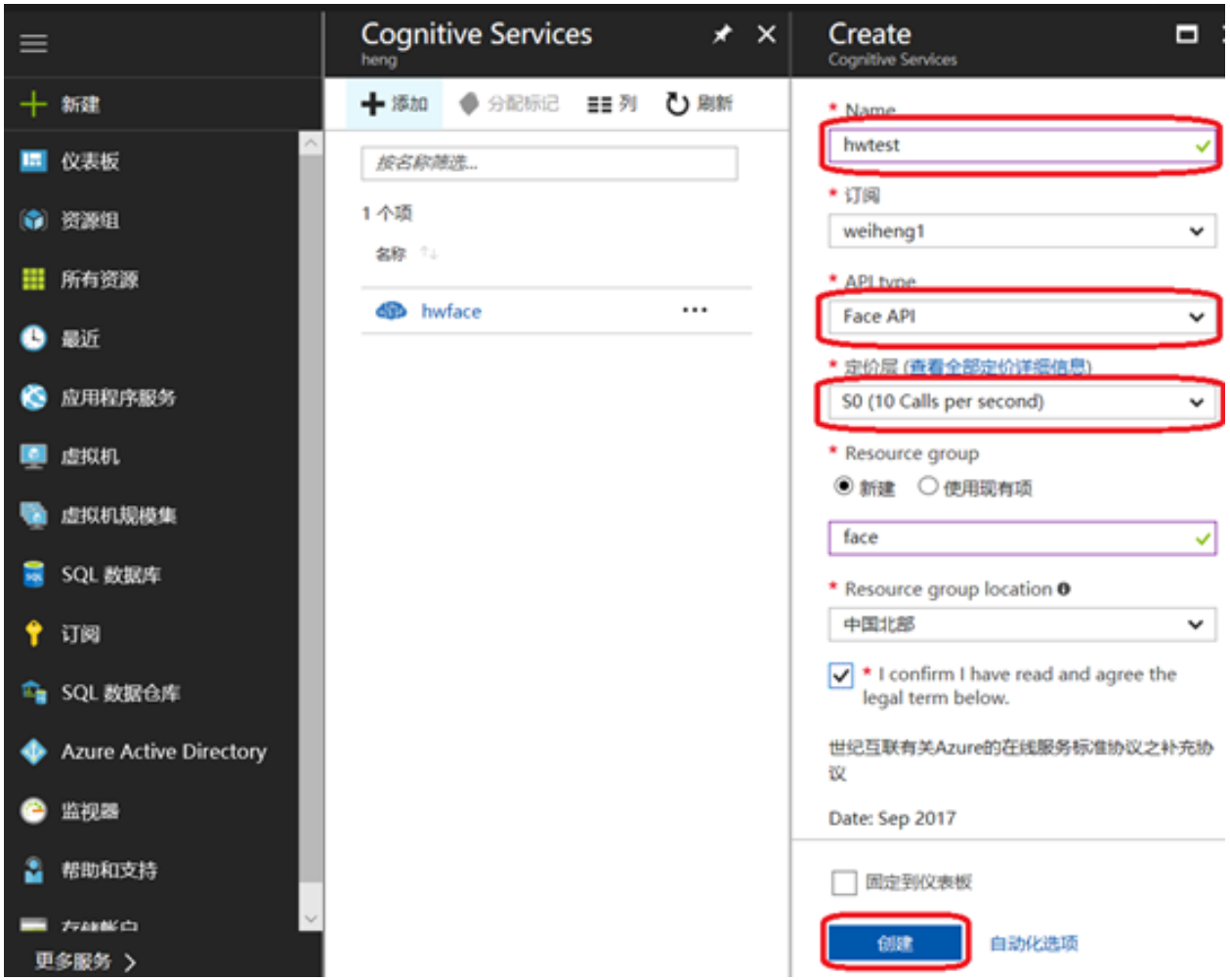
➢ 4. 数据处理、特征选择、模型选择、模型训练等



# 机器学习服务案例2：人脸识别

## ❑ 大致开发流程

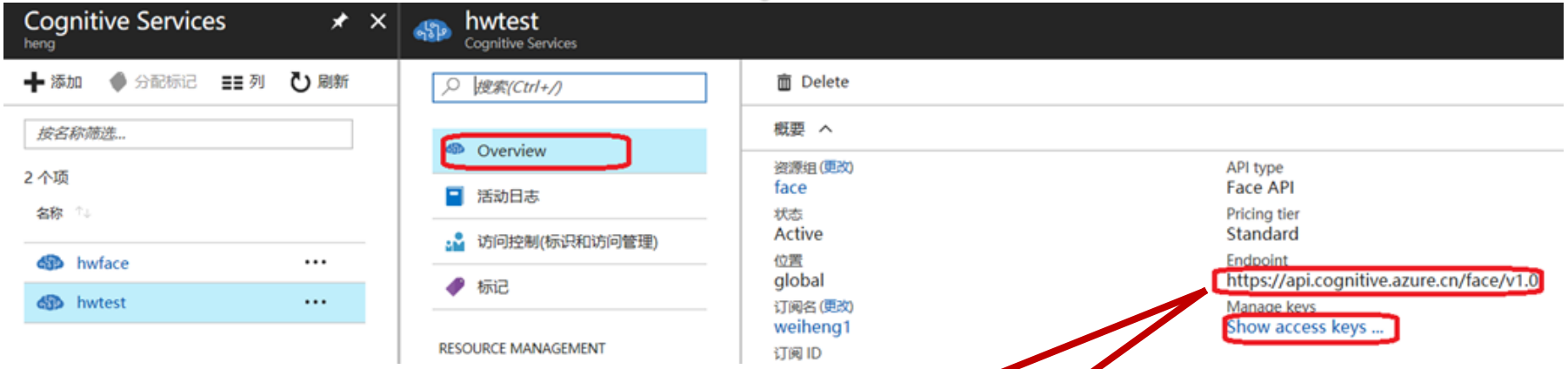
➢ 1. 在Azure上创建Cognitive Service的Face服务



# 机器学习服务案例2：人脸识别

## ❑ 大致开发流程

➢ 2. 获取服务的链接和key




➢ 3. 代码调用API进行识别

```
conn = httplib.HTTPSConnection('api.cognitive.azure.cn')
conn.request("POST", "/face/v1.0/detect?%s" % params, img, headers)
response = conn.getresponse()
data = response.read()
parsed = json.loads(data)
print ("Response:")
print (json.dumps(parsed, sort_keys=True, indent=2))
conn.close()
```

本地图片文件

定义html的header包括Content-type等

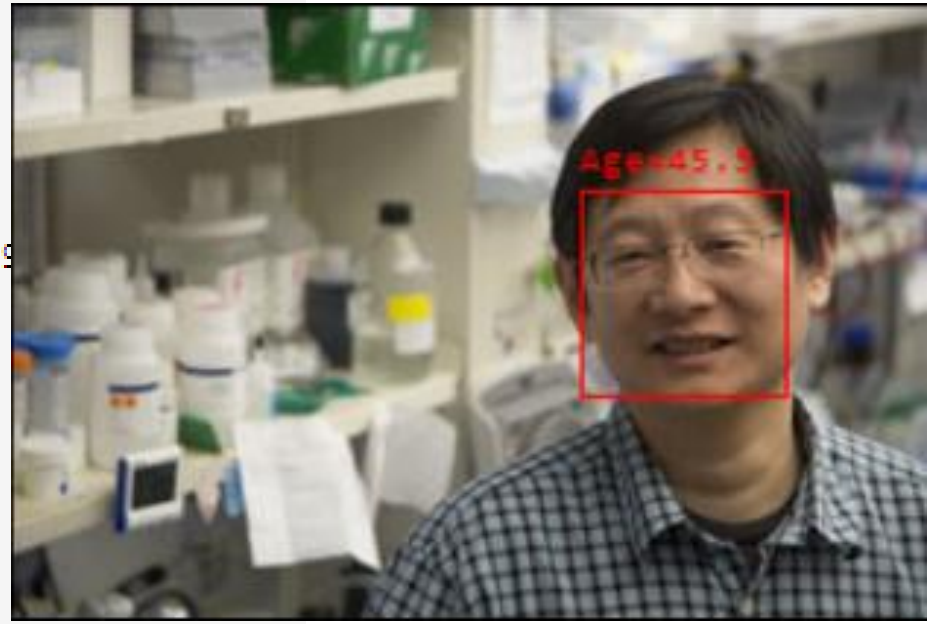
人脸识别对应函数

定义需要返回的内容包括FaceId，年龄、性别等等

# 机器学习服务案例2：人脸识别

- 大致开发流程
  - 4. 得到返回结果


```
[
    {
    "faceAttributes": {
        "age": 45.5,
        ...
        "gender": "male",
        "faceId": "b15284c9-ce1c-40eb-a76b-9
        "faceRectangle": {
            "height": 56,
            "left": 155,
            "top": 50,
            "width": 56
            }
        }
    }
]
```

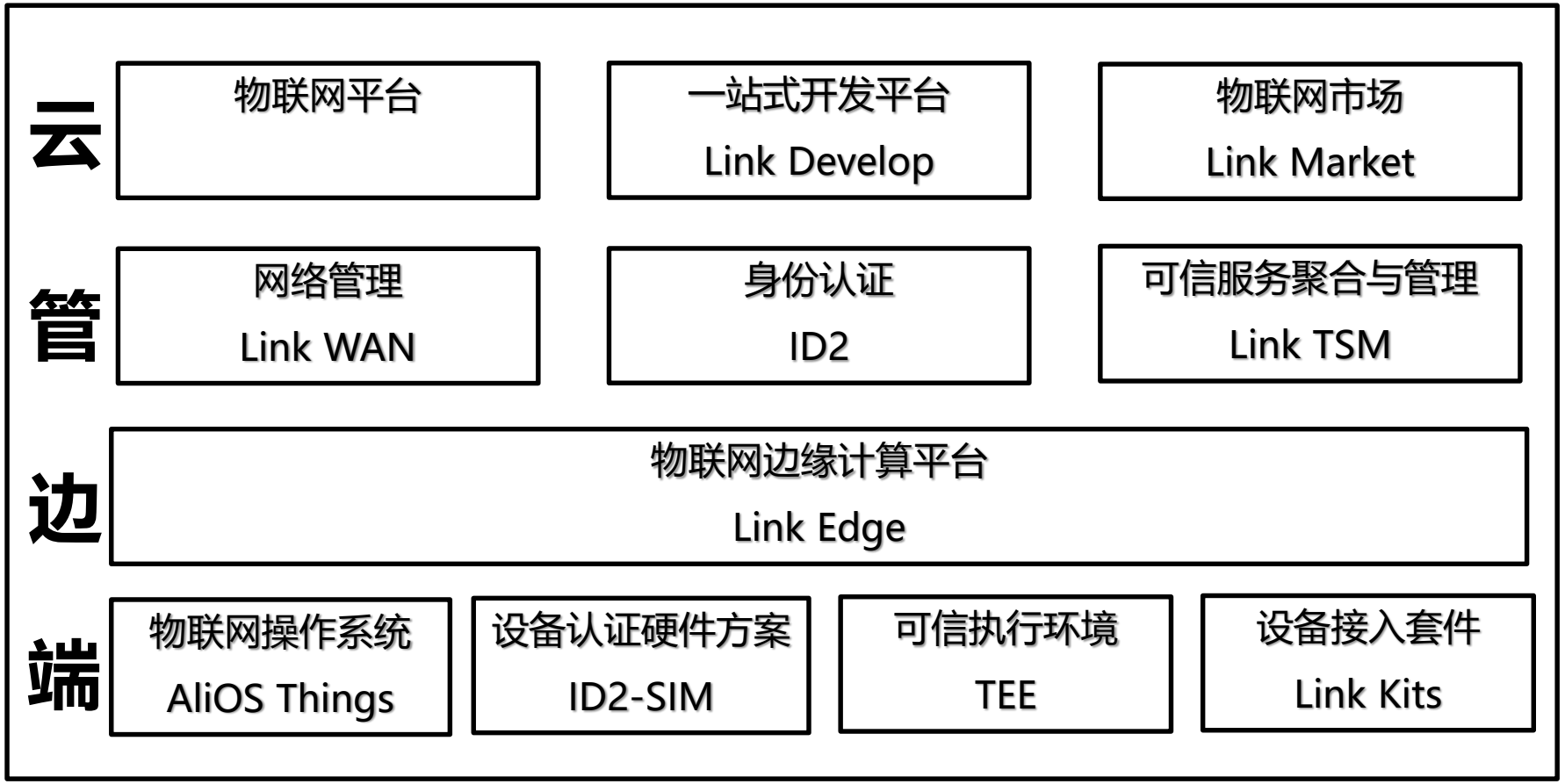
# 阿里云IoT

- 2018年3月28日在云栖大会•深圳峰会上，阿里巴巴宣布“全面进军IoT”,IoT成为阿里“一条新的主赛道”
- 阿里云IoT提供物联网领域内的平台型、基础型和应用服务型产品服务，赋予行业伙伴与开发人员快速搭建物联网的能力。

| 平台产品 | 基础产品 | 应用&服务 |
| --- | --- | --- |
| 阿里云Link生活平台<br>生活物联网平台 | 阿里云Link Edge<br>将计算扩展至边缘 | 阿里云Link测试&认证服务<br>软硬结合的测试&标准化认证服务 |
| 阿里云Link城市平台<br>城市物联网平台 | 阿里云Link WAN<br>广域低功耗物联通讯网络 | 阿里云Link魔笔<br>自然书写笔迹数字化的云端产品 |
| 阿里云Link商业共享平台<br>商业共享平台 | AliOS Things<br>嵌入式物联网实时操作系统 | 阿里云Link语音服务<br>基于物联网的智能&交互语音解决方案 |

# 阿里云IoT

## ❑ 阿里云的“云-管-边-端”产品架构

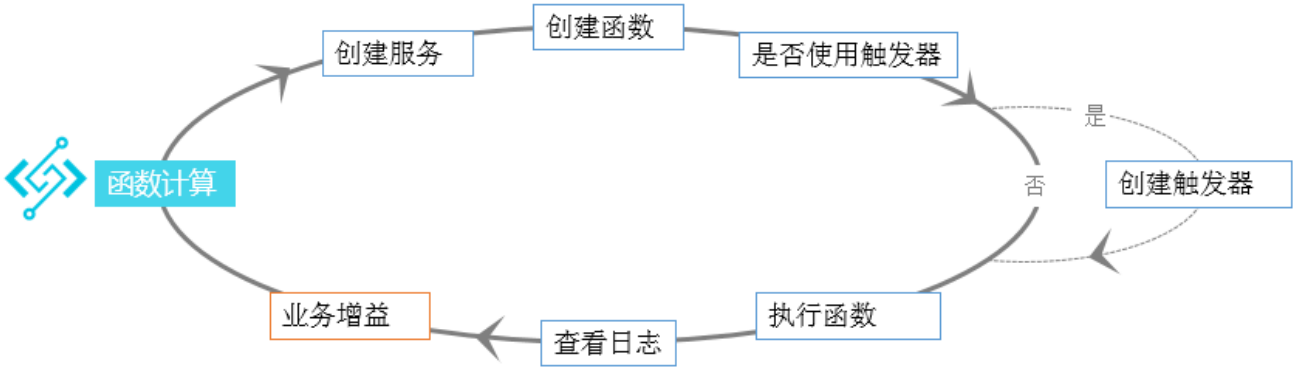


# 阿里云IoT功能概述

- **设备通信**
  - MQTT/CoAP/HTTP/HTTPS
- **设备管理**
  - 支持直接管理与设备影子
  - 支持网关子设备和OTA升级
- **时序存储**
  - 阿里云TSDB(Time Series Database)服务
- **规则引擎**
  - 可以使用规则引擎将处理后的数据：
    - *转发到存储服务（RDS、表格存储、TSDB）进行存储；*
    - *转发到DataHub进行流计算或大规模离线计算；*
    - *转发到函数计算进行事件计算；*
    - *转发到另一个Topic实现M2M通信；*
    - *转发到消息队列MQ或消息服务实现高可靠消费数据。*

# 阿里云IoT功能概述

- **设备与身份认证**
  - ID2设备认证平台
  - ID2-SIM安全技术
  - TEE可信执行环境
- **函数计算**
  - 支持语言：
    - *Node.js*
    - *Python*
    - *PHP*
    - *Java*
  - 计费方式：
    - *调用次数费用+执行时间费用+公网流量费用*



# 阿里云IoT功能概述

## ❑ 边缘计算

- ➢ 功能：
  - *设备接入*
  - *边缘函数计算（将函数计算下沉至边缘端）*
  - *消息路由（将规则引擎下沉至边缘端）*
  - *断网续传（断网后将数据保存在边缘设备上，网络恢复后再同步至云端）*

- ➢ 案例：
  - *在本地对设备数据做单位换算*
  - *在本地对数据进行过滤*
  - *在本地将数据转发至本地存储/应用*
  - *在本地访问其他服务接口*

# 阿里云IoT特色功能

- **设备认证平台ID²**
  - 一站式的软硬结合测试&标准化认证服务

- **物联网市场Link Market**
  - 一个高效的线上线下一体的物联网信息聚合平台，用于物联网设备与解决方案的推广与变现

- **一站式开发平台Link Develop**
  - 以物联网开发者为中心，覆盖设备端、数据上云、数据分析、Web应用、移动应用、服务开发等物联网开发的全部链路，实现一个平台解决全栈开发

# 设备认证平台ID²

- **概述**：阿里云Link ID²（Internet Device ID），是物联网设备的可信身份标识，具备不可篡改、不可伪造、全球唯一的安全属性，是实现万物互联、服务流转的关键基础设施。ID²支持多安全等级载体，合理地平衡物联网在安全、成本、功耗等各方面的诉求，为客户提供用得起、容易用、有保障的安全方案，适应物联网碎片化的市场需求
- **ID²生态中的两种角色**
  - **烧录者**：负责申请ID²并将ID²烧录到安全芯片等载体里，这个角色通常由SE芯片商/卡商、SIM卡商、Secure MCU芯片商、设备商（采用TEE或软沙盒方案时）承担
  - **使用者**：将ID²安全载体集成到设备中，并利用ID²构建业务安全。这个角色通常是设备商（OEM）或者方案商（ODM）承担

# 设备认证平台ID²

## ❑ ID2设备认证流程

➢ “三端-十步”

1. 设备发送认证请求给业务服务端以申请挑战随机数，并由业务服务端转发给ID2认证服务端
2. 业务服务端调用阿里云提供的SDK API，从ID2认证服务端获取到挑战随机数并转发给设备
3. （可选）设备计算额外参数
4. 设备根据本机内置的ID2与额外参数（如有）调用TFS API计算认证码
5. 将认证码发送给业务服务端
6. （可选）业务服务端计算额外参数
7. 业务服务端将认证码调用服务器SDK转发至ID2认证服务端
8. ID2认证服务端返回认证结果
9. 业务服务端根据ID2认证服务端返回的认证结果执行相应的处理
10. 业务服务端将认证业务处理结果返回设备

# 物联网市场 Link Market

- 市场定位：创新型线上线下展会

**解决方案商**

集成商

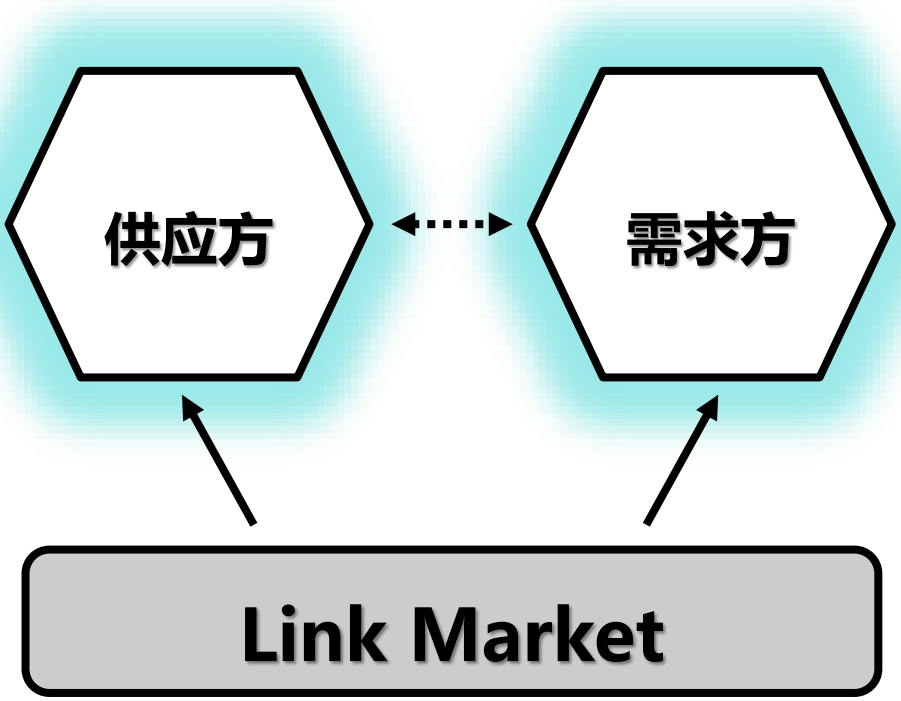
整包商

模组商

芯片商

服务商

设备商




**政府**

地产商

产业带

商家

合作社

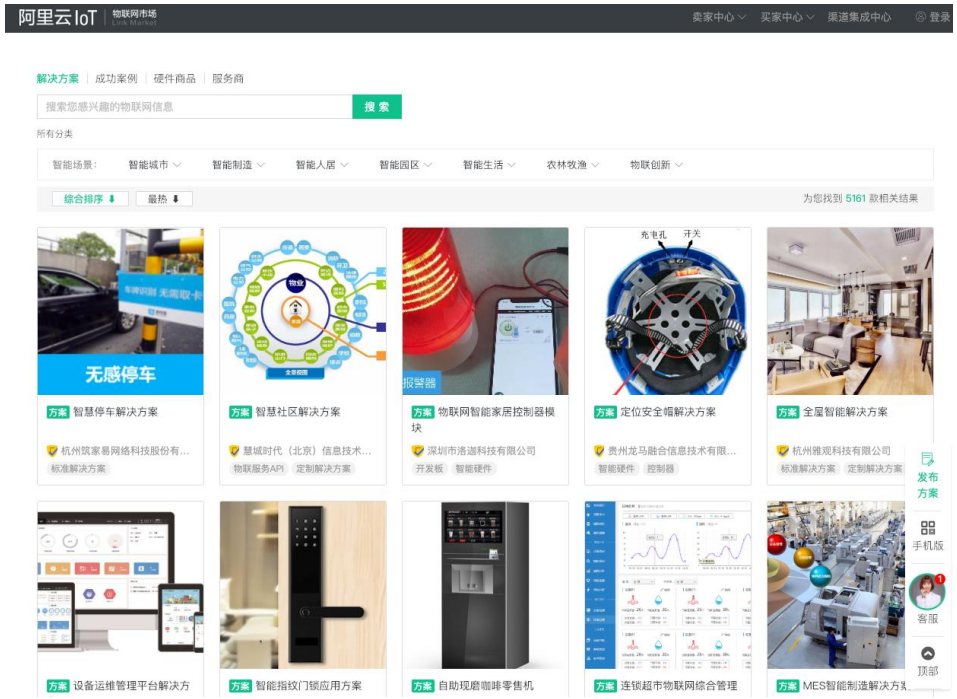
经销商

创业者

开发者

# 物联网市场Link Market

- 线上展会
  - linkmarket.aliyun.com
  - 智能制造馆、 智能城市馆、 智能全屋馆、智能园区馆、 安全产品馆、 广域物联馆、极客馆、 元器件馆
- 线下展会



# 一站式开发平台 Link Develop

- **设备开发**
  - 支持多种开发方式（嵌入式JavaScript开发、嵌入式C开发、基于AliOS-Things的开发、基于SDK的开发）
- **服务开发**
  - 支持流式服务编排，可与应用无缝集成
- **应用开发**
  - 支持可视化应用搭建，通过拖拽的方式即可生成Web应用与移动应用

# 目录


- 概述
- 物联网云平台功能概述
- 典型物联网云平台
- 总结与展望

# 如何选择云平台

| 服务平台 | 设备托管 | | | | | 设备端应用开发 | | | 用户端应用开发 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 存储 | 可视化 | 协议 | | | 统一API | 平台接入SDK | 平台交互SDK | 前端 | | | 后台 | |
| | | | MQTT | CoAP | HTTP | | | | Drag and Drop | 平台接入SDK | 统一API | IFTTT | 统一API |
| AWS IoT | √ | × | √ | × | √ | × | √ | × | × | √ | × | √ | × |
| Azure IoT | √ | √ | √ | × | √ | × | √ | × | × | √ | × | √ | × |
| 百度IoT Hub | √ | × | √ | × | × | × | √ | × | × | √ | × | √ | × |
| 阿里物联网套件 | √ | × | √ | √ | √ | × | √ | × | × | √ | × | √ | × |
| 中移OneNet | √ | √ | √ | × | √ | × | × | × | √ | × | × | √ | × |
| IBM Watson | √ | √ | √ | × | × | × | √ | × | × | √ | × | √ | × |
| Samsung SmartThings | × | × | × | × | × | × | × | × | × | × | √ | × | √ |
| ARM Mbed | × | × | × | √ | × | √ | × | × | × | √ | × | × | × |
| AliOS Things | × | × | × | × | × | √ | × | × | × | × | × | × | × |
| 机智云 | × | × | × | × | × | × | √ | √ | × | √ | × | × | × |
| 阿里Link平台 | √ | √ | √ | √ | √ | × | √ | × | × | × | × | √ | × |
| 京东微联 | × | × | × | × | × | × | √ | √ | × | √ | × | × | × |

# 总结与展望

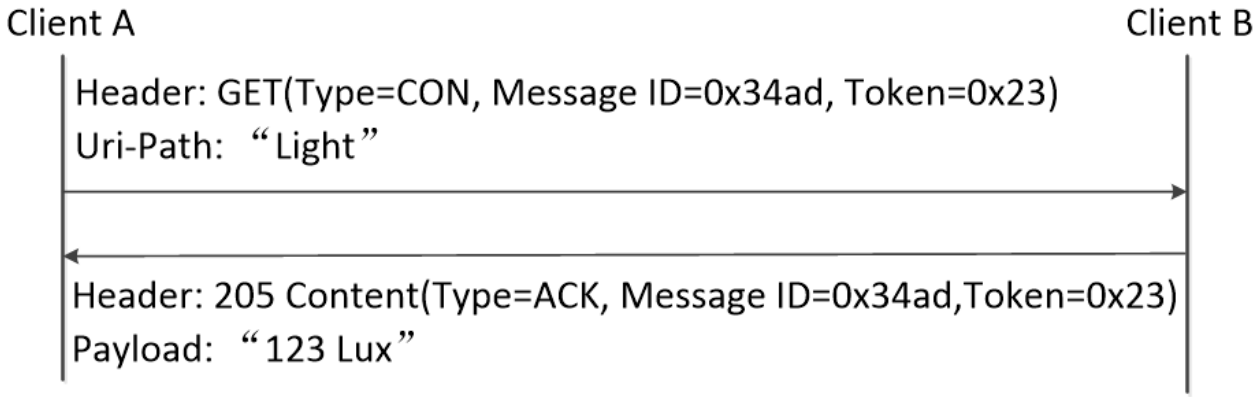
- **云-端一体化开发**
  - 平台无关的代码（或语言）描述物联网应用场景的业务逻辑，云平台会自动生成对应的设备端应用程序、云端服务和移动端应用程序
  - 开发不再受设备异构、网络异构等问题的困扰，开发者只要关注应用的业务逻辑、数据流向、服务调用

# 设备通信-习题

- 对于变长头部和数据内容长度和分别为100,200,30000的MQTT报文，分别写出MQTT固定头部的剩余长度字段。
- 对于以下代码，画出完整的MQTT报文

```
Message myMsg;
myMsg.payload = “175”;
myMsg.payloadlen = strlen(“175”)+1;
myMsg.id = 0x0012;
myMsg.qos = QoS1;
myMsg.retained = false;
myMsg.dup = false;
publish(“Light”, myMsg);
```

- 对于以下CoAP请求，画出完整的CoAP报文



# 设备管理-习题

- 设备管理主要管理哪些内容？请至少举出5个例子。
- 智能网关存在的意义是什么？请举一个例子说明。
- 直接管理和设备影子有什么不同？请举一个具体的例子说明。
- OTA升级的版本号如何保持一致？ （问的有点大）
- 攻击OTA升级主要有哪几种方式？请挑选其中一种攻击方式说明会造成的后果。

# IFTTT/规则引擎/流计算-习题

- 1. 在工业界直接使用IFTTT面临着哪些问题，规则引擎如何避开这些问题?
- 2. 不同于传统的关系型数据库基于B+树实现，时序数据库选择LSM树以实现高性能的写入操作，请调研并回答以下问题：
  - 1. 简述LSM树和B+树最显著的区别。
  - 2. 解释LSM树读取性能不佳的原因并提出一种优化的方法。
- 3. 对于问题“快速找出1 – 100 （无重复）中缺失的2个数”，除了利用流计算思想一次遍历计算得到，还有其它算法可以实现，
  - 请再提出一种算法，并通过实际编程测试，给出两种算法的运行时长的比值。

# 设备认证与权限认证-习题

- 物联网云平台有哪几种设备认证方式？各自有什么特点？
- 物联网云平台有哪几种权限认证方式？各自有什么特点？
- 结合基于对称密钥和非对称密钥的设备认证方法的优缺点，讨论分别在什么样的物联网场景下，适合使用基于对称密钥的设备认证和非对称密钥的设备认证？
- 简述通过CA证书验证设备公钥合法性的过程。

# 无服务器计算-习题

- 若Azure Function 每隔0.05秒被调用一次，每次执行时间为0.1s，每次调用需要0.03秒的冷启动时间，相对于PaaS架构，无服务器计算方式节省了多少CPU时间?

- 为物联网边缘计算场景设计一个寻找计算下沉锚点的算法（使用C语言实现），并求其计算复杂度
  - 边缘节点只知道自身的父节点（也即路由信息）

- 参照缩略图应用，在Azure云平台上编写一个处理Restful API的函数：
  - 调用：GET (Azure API)
  - 返回：“Hello World！”
- 将缩略图应用在Azure云平台使用C#实现

浙江大学
Zhejiang University

# 谢谢！

**Email: dongw@zju.edu.cn**