
IOT 文档

2018-12-26



百度云
cloud.baidu.com

目录

| | |
|-------------------------|---|
| 1 产品介绍 | 1 |
| 1.1 产品概述 | 1 |
| 1.1.1 设备型项目 | 1 |
| 适用场景 | 1 |
| 特性概述 | 1 |
| 1.1.2 数据型项目 | 1 |
| 适用场景 | 1 |
| 特性描述 | 1 |
| 1.2 核心概念 | 2 |
| 1.2.1 项目 | 2 |
| 1.2.2 物影子 | 2 |
| 1.2.3 物模型 | 2 |
| 1.2.4 MQTT | 2 |
| 1.2.5 用户 (thing) | 2 |
| 1.2.6 身份 (principal) | 2 |
| 1.2.7 策略 (policy) | 3 |
| 1.2.8 权限 (permission) | 3 |
| 1.2.9 主题 (topic) | 3 |
| 1.2.10 操作权限 (operation) | 3 |
| 1.3 产品架构 | 3 |
| 1.4 产品优势 | 4 |

| | | |
|-------|------------------|---|
| 1.4.1 | 开放高效 | 4 |
| 1.4.2 | 安全可靠 | 4 |
| 1.4.3 | 快速开发 | 4 |
| 1.4.4 | 端云一体 | 4 |
| 1.4.5 | 多场景支持 | 5 |
| 1.5 | 系统限制 | 5 |
| 1.5.1 | 项目数 | 5 |
| 1.5.2 | 消息长度限制 | 5 |
| 1.5.3 | 项目限制 | 5 |
| | 新建连接数 | 5 |
| | 设备型项目限制 | 5 |
| | 数据型项目限制 | 5 |
| 2 | 产品定价 | 6 |
| 2.1 | 计费策略 | 6 |
| 2.1.1 | 升级和续费 | 7 |
| | 升级 | 7 |
| | 续费 | 7 |
| 2.2.1 | 计费消息 | 7 |
| 2.2.2 | 免费消息 | 8 |
| 2.2 | 计费项目 | 7 |
| 2.3 | 到期提醒和处理 | 8 |
| 2.3.1 | 用量提醒 | 8 |
| 2.3.2 | 到期提醒 | 8 |
| 2.3.3 | 到期后处理 | 8 |
| 3 | 快速入门 | 9 |
| 3.1 | 概述 | 9 |
| 3.2 | 步骤一：注册并登录IoT Hub | 9 |

| | |
|--------------------------------------|----|
| 3.3 步骤二：创建计费套餐 | 10 |
| 3.4 步骤三：创建项目 | 11 |
| 3.5 步骤四：获取连接信息 | 12 |
| 3.5.1 设备型项目 | 12 |
| 3.5.2 数据型项目 | 15 |
| 3.6 步骤五：通过 MQTT.fx 客户端测试连接 | 16 |
| 3.6.1 前提条件 | 16 |
| 3.6.2 操作步骤 | 16 |
| 3.7 多种方式使用物接入服务 | 19 |
| 3.7.1 Python | 19 |
| 3.7.2 NodeJS | 20 |
| 3.7.3 Java | 21 |
| 3.7.4 Arduino D1 & NodeMCU | 22 |
| 3.7.5 Arduino Wido | 23 |
| 3.7.6 Arduino Yun | 24 |
| 3.7.7 MQTT.fx | 25 |
| | |
| 4 操作指南 | 31 |
| 4.1 通用功能 | 31 |
| 4.1.1 连接建立与断开事件订阅 | 31 |
| 获取连接建立事件 | 31 |
| 获取连接断开事件 | 31 |
| 4.1.2 持久化会话和消息缓存 | 32 |
| 4.1.3 保留 (Retain) 消息 | 34 |
| 4.1.4 HTTP 方式更新主题 | 34 |
| 4.2 设备型项目 | 36 |
| 4.2.1 创建物影子 | 36 |
| 物模型 | 36 |

| | |
|--------------------------------|----|
| 创建物模型 | 36 |
| 管理物模型 | 37 |
| 物影子 | 38 |
| 创建物影子 | 38 |
| 查看影子详情 | 39 |
| 获取连接配置 | 43 |
| 4.2.2 物影子操作 | 43 |
| 使用Baidu IoT Edge SDK | 44 |
| 使用开源MQTT客户端SDK | 44 |
| 更新设备状态到设备影子 | 44 |
| 从设备影子获取设备状态 | 46 |
| 通过设备影子控制设备状态 | 47 |
| 清空设备影子 | 48 |
| 订阅设备影子的变化 | 49 |
| 订阅设备快照 | 49 |
| Device Profile | 50 |
| 4.2.3 权限管理 | 51 |
| 介绍 | 51 |
| 创建权限 | 51 |
| 创建超级权限 | 53 |
| 4.2.4 设备在线状态 | 54 |
| 4.2.5 设备间相互通信 | 55 |
| 4.2.6 OTA 服务 | 55 |
| 介绍 | 55 |
| 云端OTA操作 | 55 |
| 1. 账号体系 | 55 |
| 2. 使用前准备 | 56 |
| 3. 上传升级包 | 56 |

| | |
|-----------------------------------|----|
| 4. 创建升级任务 | 58 |
| 5. 追踪升级任务进度 | 59 |
| 设备端使用MQTT OTA | 59 |
| MQTT OTA的功能 | 59 |
| 通过MQTT客户端使用MQTT OTA | 60 |
| 跨平台OTA SDK (HTTPS) 接入 | 65 |
| 术语介绍 | 65 |
| OTA SDK接入流程 | 66 |
| OTA SDK接口说明 | 70 |
| OTA SDK跨平台支持 | 77 |
| OTA SDK示例程序 | 77 |
| 制作升级包工具 | 79 |
| 4.2.7 数据存储 TSDB | 80 |
| 开启存储配置 | 80 |
| 存储TSDB格式 | 81 |
| 4.2.8 物影子数据可视化 | 82 |
| 4.3 数据型项目 | 85 |
| 4.3.1 操作准备 | 85 |
| 4.3.2 数据型项目配置 | 86 |
| 创建物接入项目 | 86 |
| 创建物接入用户 | 86 |
| 创建物接入身份 | 88 |
| 创建物接入策略 | 90 |
| 关于通配符的使用方法 | 91 |
| 避免用广播方式向特定用户发送消息 | 92 |
| 4.3.3 连接测试 | 94 |
| 4.3.4 基于证书的双向认证 | 96 |

| | | |
|-------|------------------------|-----|
| 4.3.5 | 数据型消息存储配置 | 101 |
| 4.4 | 其他 | 101 |
| 4.4.1 | 多用户协作 | 101 |
| 5 | 最佳实践 | 104 |
| 5.1 | 基于物影子快速搭建物联网应用 | 104 |
| 5.1.1 | 简介 | 104 |
| 5.1.2 | 云端创建设备影子 | 104 |
| 5.1.3 | 设备端发消息更新云端的物影子 | 105 |
| 5.1.4 | 设备端与云端通信（非更新物影子） | 105 |
| 5.1.5 | 应用服务消费设备的数据 | 106 |
| 5.2 | ST物联网开发套件连接例程 | 106 |
| 5.2.1 | 操作准备 | 106 |
| 5.2.2 | SW4STM32使用说明 | 108 |
| 5.2.3 | 创建物接入项目 | 112 |
| 5.2.4 | 修改源码内Baidu IoT Hub连接信息 | 112 |
| 5.2.5 | 编译下载程序 | 113 |
| 5.2.6 | Tera Term使用说明 | 116 |
| 5.2.7 | 运行程序 | 118 |
| 5.2.8 | 参考文档 | 120 |
| 6 | API参考(数据型) | 121 |
| 6.1 | 目录 | 121 |
| 6.2 | 介绍 | 122 |
| 6.2.1 | 简介 | 122 |
| 6.2.2 | 调用方式 | 122 |
| | 概述 | 123 |
| | 通用约定 | 123 |
| | 公共请求头 | 123 |

| | |
|--|-----|
| 公共响应头 | 123 |
| 响应状态码 | 123 |
| 通用错误返回格式 | 124 |
| 公共错误码 | 125 |
| 签名认证 | 125 |
| 签名生成算法 | 125 |
| 6.2.3 多区域选择 | 125 |
| 6.3 认证 | 126 |
| 6.3.1 认证 | 126 |
| 6.3.2 鉴权 | 126 |
| 6.4 动作 | 127 |
| 6.4.1 给一个Thing添加一个Principal | 127 |
| 6.4.2 从一个Thing移除一个Principal | 128 |
| 6.4.3 给一个Principal添加一个Policy | 129 |
| 6.4.4 从一个Principal移除一个Policy | 130 |
| 6.5 Endpoint | 131 |
| 6.5.1 获取endpoint列表 | 131 |
| 6.5.2 获取指定的endpoint信息 | 133 |
| 6.5.3 创建endpoint | 134 |
| 6.5.4 删除endpoint | 136 |
| 6.6 Thing | 136 |
| 6.6.1 获取thing列表 | 136 |
| 6.6.2 获取指定的thing信息 | 138 |
| 6.6.3 创建thing | 139 |
| 6.6.4 删除thing | 140 |
| 6.7 Principal | 141 |
| 6.7.1 获取principal列表 | 141 |
| 6.7.2 获取指定的principal信息 | 143 |

| | | |
|--------|--------------------|-----|
| 6.7.3 | 创建principal | 144 |
| 6.7.4 | 重新生成密钥 | 145 |
| 6.7.5 | 删除principal | 146 |
| 6.8 | Policy | 147 |
| 6.8.1 | 获取policy列表 | 147 |
| 6.8.2 | 获取指定的policy信息 | 148 |
| 6.8.3 | 创建policy | 149 |
| 6.8.4 | 删除policy | 150 |
| 6.9 | Permission | 151 |
| 6.9.1 | 获取policy下所有topic信息 | 151 |
| 6.9.2 | 获取指定topic的信息 | 153 |
| 6.9.3 | 在policy下设置topic | 154 |
| 6.9.4 | 更新已有的topic设置 | 155 |
| 6.9.5 | 删除已有的topic | 157 |
| 6.10 | Client | 158 |
| 6.10.1 | 获取指定MQTT客户端在线状态 | 158 |
| 6.10.2 | 获取所有MQTT客户端在线状态 | 158 |
| 6.10.3 | Publish Message | 159 |
| 6.11 | 使用量 | 161 |
| 6.11.1 | 获取当前账单月内使用量 | 161 |
| 6.11.2 | 获取当前账单月内特定实例的使用量 | 161 |
| 6.11.3 | 查询特定实例某个时间段内的使用量 | 162 |
| 7 | API参考(设备型) | 165 |
| 7.1 | 目录 | 165 |
| 7.2 | 介绍 | 166 |
| 7.2.1 | 简介 | 166 |
| 7.2.2 | 签名认证 | 167 |

| | | |
|--------|--------------------------|-----|
| 7.2.3 | 多区域选择 | 167 |
| 7.3 | 设备列表管理 | 168 |
| 7.3.1 | 创建单个设备 | 168 |
| 7.3.2 | 删除设备 | 169 |
| 7.3.3 | 获取设备Profile | 170 |
| 7.3.4 | 获取设备View | 171 |
| 7.3.5 | 获取及查询影子列表 | 172 |
| 7.3.6 | 获取设备接入详情 | 174 |
| 7.3.7 | 更新密钥 | 175 |
| 7.3.8 | 更新设备属性 | 176 |
| 7.3.9 | 更新单个设备View信息 | 178 |
| 7.3.10 | 更新单个设备注册表信息 | 179 |
| 7.3.11 | 重置设备影子 | 181 |
| 7.4 | 设备模板管理 | 182 |
| 7.4.1 | 创建模板 | 182 |
| 7.4.2 | 删除模板 | 183 |
| 7.4.3 | 获取模板列表 | 184 |
| 7.4.4 | 获取模板 | 186 |
| 7.4.5 | 更新模板 | 187 |
| 7.5 | 物管理数据写入TSDB | 188 |
| 7.5.1 | 创建规则 | 188 |
| 7.5.2 | 获取规则详情 | 191 |
| 7.5.3 | 修改规则 | 193 |
| 7.5.4 | 删除规则 | 197 |
| 7.5.5 | 禁用一条规则 | 197 |
| 7.5.6 | 启用一条规则 | 198 |
| 7.5.7 | 创建带TSDB格式的规则 | 198 |
| 7.5.8 | 获取带TSDB格式的规则详情 | 201 |

| | | |
|--------|---------------------------------------|-----|
| 7.5.9 | 修改带TSDB格式的规则 | 203 |
| 7.5.10 | 参数定义 | 207 |
| | DeviceFormatRuleRequest | 208 |
| | DeviceRuleRequest | 208 |
| | DeviceRuleSource | 209 |
| | DeviceRuleDestination | 210 |
| | DeviceRuleFormat | 211 |
| | DeviceRuleResponse | 211 |
| | DeviceFormatRuleResponse | 212 |
| | DeviceRuleSourceDetail | 213 |
| | DeviceRuleDestinationDetail | 214 |
| 7.6 | OTA升级服务 | 214 |
| 7.6.1 | 上传固件包文件 | 214 |
| 7.6.2 | 创建固件包 | 215 |
| 7.6.3 | 获取固件包详情 | 217 |
| 7.6.4 | 获取固件包列表 | 218 |
| 7.6.5 | 查询设备使用固件包版本 | 220 |
| 7.6.6 | 创建升级任务 | 222 |
| 7.6.7 | 上传升级设备文件 | 223 |
| 7.6.8 | 查询升级任务 | 224 |
| 7.6.9 | 查询升级任务列表 | 226 |
| 7.6.10 | 查询升级任务结果 | 227 |
| 7.6.11 | 查询升级任务各设备升级结果 | 229 |
| 7.6.12 | 参数定义 | 230 |
| | FirmwarePackage | 231 |
| | FirmwarePackageDetail | 231 |
| | SimplifiedJob | 231 |
| | OtaJob | 232 |

| | |
|--------------------------------------|-----|
| JobStatistics | 232 |
| DeviceJobResult | 232 |
| ResourceID | 232 |
| VersionList | 233 |
| 7.7 权限管理 | 233 |
| 7.7.1 简介 | 233 |
| 7.7.2 设备权限组管理 | 233 |
| 创建权限组 | 233 |
| 删除权限组 | 235 |
| 获取权限组列表 | 235 |
| 获取权限组详情 | 237 |
| 权限组中更改设备 | 238 |
| 更新权限组注册信息 | 239 |
| 获取权限组接入信息 | 240 |
| 更新权限组密钥 | 241 |
| 获取及查询设备列表 | 242 |
| 7.7.3 参数定义 | 244 |
| 7.8 参数定义 | 245 |
| 7.8.1 DeviceListRequest参数列表 | 245 |
| 7.8.2 DeviceListResponse参数列表 | 245 |
| 7.8.3 DeviceProfile参数列表 | 245 |
| 7.8.4 DeviceProfileResponse参数列表 | 245 |
| 7.8.5 DeviceBasicInfo参数列表 | 246 |
| 7.8.6 DeviceAttributes参数列表 | 246 |
| 7.8.7 DeviceViewResponse参数列表 | 247 |
| 7.8.8 DeviceViewAttributes参数列表 | 247 |
| 7.8.9 DeviceAccessDetailResponse参数列表 | 247 |
| SchemaProperty参数列表 | 248 |

| | |
|---------------------------------------|-----|
| Schema参数列表 | 249 |
| 7.9 更新历史 | 249 |
| 8 Java SDK文档(数据型) | 250 |
| 8.1 目录 | 250 |
| 8.2 概述 | 251 |
| 8.3 安装SDK工具包 | 251 |
| 8.4 快速入门 | 252 |
| 8.5 创建IotHubClient | 252 |
| 8.6 endpoint操作 | 254 |
| 8.6.1 创建endpoint | 254 |
| 8.6.2 获取endpoint列表 | 254 |
| 8.6.3 查看指定的endpoint信息 | 254 |
| 8.6.4 删除endpoint | 254 |
| 8.7 thing操作 | 254 |
| 8.7.1 创建thing | 254 |
| 8.7.2 查看thing | 255 |
| 8.7.3 删除thing | 255 |
| 8.8 principal操作 | 255 |
| 8.8.1 创建principal | 255 |
| 8.8.2 查看principal | 255 |
| 8.8.3 绑定指定的thing和principal | 256 |
| 8.8.4 重新获得principal的密码 | 256 |
| 8.8.5 删除principal | 256 |
| 8.9 policy操作 | 256 |
| 8.9.1 创建policy | 256 |
| 8.9.2 查看policy | 256 |
| 8.9.3 绑定指定的principal和policy | 257 |

| | | |
|--------|---------------------------------|-----|
| 8.9.4 | 删除policy | 257 |
| 8.10 | permission操作 | 257 |
| 8.10.1 | 创建permission | 257 |
| 8.10.2 | 更新permission | 258 |
| 8.10.3 | 查看permission | 258 |
| 8.11 | Certificate操作 | 258 |
| 8.11.1 | 1.创建Certificate | 258 |
| 8.11.2 | 2.重置Certificate | 259 |
| 8.11.3 | 3.使用MqttSDK实现双向认证示例代码 | 259 |
| 8.12 | 版本说明 | 262 |
| 8.12.1 | v0.10.13 | 262 |
| 9 | Java SDK文档(设备型) | 263 |
| 9.1 | 概述 | 263 |
| 9.1.1 | 安装SDK工具包 | 263 |
| 9.2 | 创建IotDmV3Client | 264 |
| 9.3 | 设备管理 | 266 |
| 9.3.1 | 创建设备 | 266 |
| 9.3.2 | 删除设备 | 266 |
| 9.3.3 | 获取设备Profile | 266 |
| 9.3.4 | 获取设备View | 267 |
| 9.3.5 | 获取及查询影子列表 | 267 |
| 9.3.6 | 获取设备接入详情 | 268 |
| 9.3.7 | 更新密钥 | 268 |
| 9.3.8 | 更新设备属性 | 269 |
| 9.3.9 | 更新设备View | 269 |
| 9.3.10 | 更新设备注册表信息 | 270 |

| | | |
|--------|--------------|-----|
| 9.3.11 | 重置设备影子 | 270 |
| 9.4 | 模板管理 | 271 |
| 9.4.1 | 创建模板 | 271 |
| 9.4.2 | 删除模板 | 271 |
| 9.4.3 | 更新模板 | 272 |
| 9.4.4 | 获取模板详情 | 272 |
| 9.4.5 | 获取及查询模板列表 | 273 |
| 9.5 | 权限组管理 | 273 |
| 9.5.1 | 创建权限组 | 273 |
| 9.5.2 | 删除权限组 | 274 |
| 9.5.3 | 获取权限组列表 | 274 |
| 9.5.4 | 获取权限组详情 | 274 |
| 9.5.5 | 权限组中更改设备 | 274 |
| 9.5.6 | 更新权限组注册信息 | 275 |
| 9.5.7 | 获取权限组接入信息 | 275 |
| 9.5.8 | 更新权限组密钥 | 275 |
| 9.5.9 | 获取及查询设备列表 | 276 |
| 9.6 | 规则引擎 | 276 |
| 9.6.1 | 创建规则 | 276 |
| 9.6.2 | 修改规则 | 277 |
| 9.6.3 | 删除规则 | 277 |
| 9.6.4 | 禁用规则 | 278 |
| 9.6.5 | 启动规则 | 278 |
| 9.7 | 物管理转储TSDB | 278 |
| 9.7.1 | 创建带TSDB格式的规则 | 278 |
| 9.7.2 | 修改带TSDB格式的规则 | 279 |
| 9.8 | 版本说明 | 280 |
| 9.8.1 | v0.10.34 | 280 |

| | |
|---|-----|
| 9.8.2 v0.10.21 | 280 |
| 10 IoT Edge SDK | 281 |
| 10.1 IoT Edge SDK | 281 |
| 10.2 移植百度天工SDK到其他平台 | 281 |
| 10.2.1 介绍 | 281 |
| 10.3 参考 | 281 |
| 10.3.1 概括 | 282 |
| 10.3.2 tickcounter适配器 | 284 |
| 10.3.3 agenttime适配器 | 284 |
| 10.3.4 sleep适配器 | 284 |
| 10.3.5 platform适配器 | 284 |
| 10.3.6 threadapi和lock适配器 | 284 |
| 10.3.7 tlsio适配器介绍 | 285 |
| 10.3.8 socketio适配简介 | 286 |
| 10.3.9 tlsio适配器实现 | 286 |
| 10.3.10 支持设备支持仓库 | 287 |
| 11 物接入IoT Hub服务等级协议 (SLA) | 288 |
| 11.1 IoT Hub SLA | 288 |
| 12 常见问题 | 292 |
| 12.1 产品配置操作问题 | 292 |
| 12.1.1 物接入的设备型和数据型如何选择? | 292 |
| 12.1.2 物接入项目列表中为什么出现了不是我自己创建的项目 ? | 292 |
| 12.1.3 物接入中是否能够批量创建设备 ? | 292 |
| 12.1.4 物接入设备型提供了哪些 topic? | 292 |
| 12.1.5 可以使用MQTT的SDK连接物影子吗? | 293 |
| 12.1.6 物影子如何获得设备在线状态? | 293 |
| 12.1.7 物影子有什么作用? | 293 |

| | |
|--|-----|
| 12.1.8 编辑设备影子中，reported字段和desired字段代表什么意思？ | 293 |
| 12.1.9 编辑设备影子中，profileVersion是什么意思？ | 293 |
| 12.1.10 设备上传到云端的字段会覆盖设备影子吗？ | 293 |
| 12.1.11 影子中“lastActiveTime”字段是什么意思？ | 294 |
| 12.1.12 物接入的数据能存储到哪里？ | 294 |
| 12.2 产品规格及使用限制 | 294 |
| 12.2.1 我设备数量较多，会超过上限，如何提升？ | 294 |
| 12.2.2 物接入上传的消息大小有限制吗？ | 294 |
| 12.2.3 每个百度云账号可以创建多少个项目？ | 294 |
| 12.3 客户端及MQTT SDK相关问题 | 294 |
| 12.3.1 请问是否有基于FreeRTOS或者RTX操作系统的MQTT SDK源码？ | 294 |
| 12.3.2 为什么publish的QOS等级为2后，则会断开服务？ | 295 |
| 12.3.3 物接入是否支持消息缓存？ | 295 |
| 12.3.4 与物接入服务连接成功，往一个主题发送消息，就直接断开。 | 295 |
| 12.3.5 连接物接入服务时，出现连接协议错误。 | 295 |
| 12.3.6 遗嘱消息的触发条件有哪些？ | 295 |
| 12.3.7 MQTT客户端网络连接异常，但在keepalive时间内恢复，客户端是否需要重新建立MQTT连接？ | 295 |
| 12.4 API使用问题 | 296 |
| 12.4.1 使用API连接物接入时，返回connection timeout错误。 | 296 |
| 12.4.2 使用API连接物接入时，返回invalid ClientID | 296 |
| 13 其他参考 | 297 |
| 13.1 MQTT协议介绍 | 297 |
| 13.1.1 MQTT协议是什么？ | 297 |
| 13.1.2 MQTT协议如何工作？ | 297 |
| 13.1.3 如何将消息正确送达？ | 298 |
| 13.1.4 如何使用通配符订阅多个主题？ | 298 |
| 13.1.5 如何确保消息已被送达？ | 299 |

| | |
|------------------------------------|-----|
| 13.1.6 什么是临终遗嘱？ | 299 |
| 13.2 MQTT客户端使用指南 | 300 |
| 13.2.1 Websockets Client | 300 |
| 13.2.2 MQTT.fx | 300 |
| 13.2.3 连接IoT Hub服务 | 300 |
| 13.2.4 订阅消息 | 301 |
| 13.2.5 发布消息 | 301 |
| 13.3 MQTT客户端代码示例 | 302 |
| 13.3.1 C代码示例 | 302 |
| 下载TLS认证文件 | 302 |
| 下载并执行示例代码 | 302 |
| 13.3.2 C#代码示例 | 305 |
| 13.3.3 python代码示例 | 306 |
| 下载TLS认证文件 | 307 |
| 13.3.4 Java代码示例 | 311 |
| 13.4 MQTT Client SDK | 311 |
| 13.5 相关参考 | 312 |
| 14 下载专区 | 313 |
| 14.1 下载TLS认证文件 | 313 |
| 14.2 下载IoT Edge SDK | 313 |
| 14.3 下载示例开发版教程 | 313 |
| 14.4 下载MQTT相关 | 314 |
| 14.4.1 下载MQTT客户端代码示例 | 314 |
| 14.4.2 下载MQTT.fx客户端 | 314 |
| 14.4.3 下载MQTT Client SDK | 314 |
| 14.5 多种语言试用物接入 | 315 |
| 14.5.1 概述 | 315 |

| | | |
|--------|----------------------------|-----|
| 14.5.2 | Python | 315 |
| 14.5.3 | NodeJS | 315 |
| 14.5.4 | Java | 315 |
| 14.5.5 | Arduino D1 & NodeMCU | 315 |
| 14.5.6 | Arduino Wido | 315 |
| 14.5.7 | Arduino Yun | 316 |

第1章 产品介绍

1.1 产品概述

物接入 (IoT Hub) 是面向物联网领域开发者的全托管云服务，通过主流的物联网协议（如 MQTT）通讯，可以在智能设备与云端之间建立安全的双向连接，快速实现物联网项目。

物接入分为设备型（原物管理）和数据型两种项目类型。设备型适用于基于设备的物联网场景；数据型适用于基于数据流的物联网场景。

您可以利用物接入来作为搭建您物联网应用的第一步，支持亿级并发连接和消息数，支持海量设备与云端安全可靠的双向连接，无缝对接天工平台和百度云的各项产品和服务。

1.1.1 设备型项目

适用场景 适用基于设备的物联网场景，以物影子作为设备在云端的映像，帮助开发者聚焦业务。

特性概述

- 提供设备模型构建工具，快速建立以设备为核心的物联网应用
- 无需关心协议细节，无缝对接时序数据库TSDB、物可视等产品
- 支持设备在线状态、权限、反控及OTA远程升级等丰富特性

1.1.2 数据型项目

适用场景 无设备概念或深度依赖数据流的场景，需使用者有较强的软硬件开发能力

特性描述

- 支持自定义 Topic，需对协议有较好了解
- 需开发者搭配规则引擎或自行处理数据流转及存储

1.2 核心概念

1.2.1 项目

物接入IoT Hub的项目，每一个项目代表一个完整的物接入 endpoint。项目有设备型和数据型两种类型。

1.2.2 物影子

物影子反映物理世界中的一个物（设备），是物在云端的『影子』或『数字双胞胎』。运行时，物将监控值上报给物影子，物影子会用一个 json 文档存储设备的最后一次上报的状态，您可以直接通过MQTT或HTTP访问。同时，物影子也提供反控功能。

1.2.3 物模型

物模型由一个或多个属性构成，您可以用他来表示一类（或同一型号的一批）设备。基于物模型可以创建物影子。

1.2.4 MQTT

MQTT (Message Queuing Telemetry Transport) 是一个基于二进制消息的客户端服务端架构的发布/订阅 (Publish/Subscribe) 模式的消息传输协议，最早由IBM提出的，如今已经业界通行规范，更符合机器与机器的通信 (M2M) 以及物联网环境 (IoT) 。

如您在使用数据型项目，还需了解以下内容。

1.2.5 用户 (thing)

表示物接入IoT Hub 的用户，用户可以在每个endpoint项目中创建一个或多个用户thing。

1.2.6 身份 (principal)

principal即身份，是一个抽象概念，表示连接用户(thing)的身份，基于身份可以对用户进行权限管理。每个用户thing可以绑定一个身份principal，每个身份principal拥有一个策略policy。

1.2.7 策略 (policy)

策略，表示每个身份对于对应用户所具备的权限，可以为身份principal设置对应的策略policy，一个principal对应一个policy。

1.2.8 权限 (permission)

权限，表示对策略policy、主题topic所拥有的能力。为每一个policy设置一组权限permission，其中包括主题topic，和对该主题的操作权限operation。

1.2.9 主题 (topic)

每一个策略policy都需要指定一个主题topic，在进行使用物接入服务之前，需要先为我们即将开展的订阅发布信息创建一个主题名称，该主题应用于MQTT客户端。topic规则允许字符串可以带一个通配符“#”，例如“temperature/#”就是匹配前缀是temperature的所有topic；单独的“#”表示匹配所有topic。

1.2.10 操作权限 (operation)

对topic的操作权限。目前基于MQTT协议，IoT Hub 支持创建发布 Publish 和订阅 Subscribe 两种权限。

1.3 产品架构

物接入（设备型）产品使用架构如图：

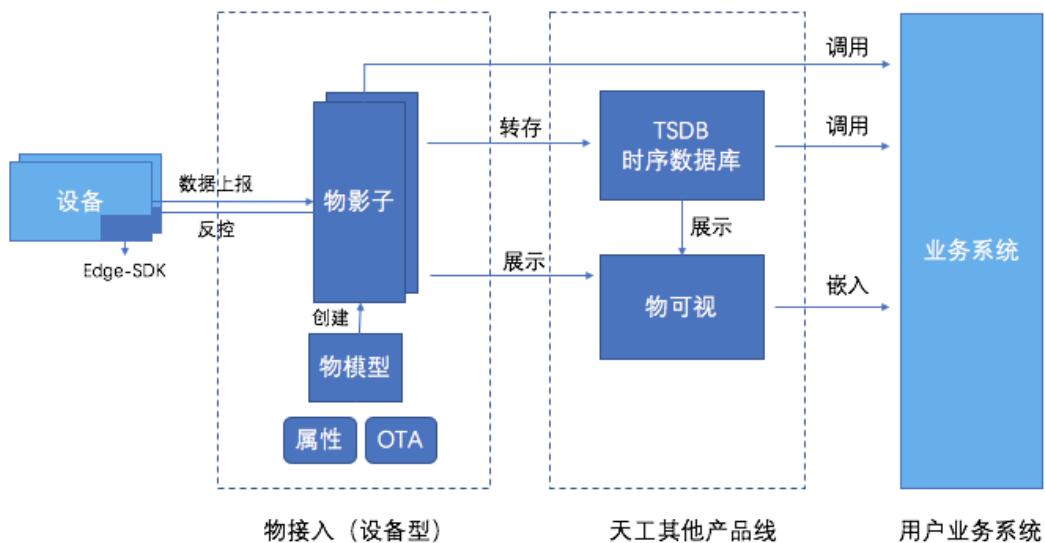


图 1.1：图

设备端可以通过集成 Edge-SDK 或开源的 MQTT client，与其对应物影子连接，进行消息收发来实现设备数据上报及反控。

同时，可与天工其他产品协作，完成历史数据存储化展示等需求。

1.4 产品优势

1.4.1 开放高效

原生支持 MQTT 协议，基于可支撑亿级设备连接及双向大规模消息传输的高可用架构

1.4.2 安全可靠

支持设备安全认证，提供权限管理能力，并通过 SSL 双向认证保证设备数据安全传输

1.4.3 快速开发

IoT Edge SDK 适配主流操作系统，支持多种语言，多种平台开发，兼容主流设备平台

1.4.4 端云一体

以物影子作为设备的云端映射，支持以影子为核心的设备全生命周期、一站式设备管理服务，包括设备监测、反控、固件升级（OTA）等多项能力

1.4.5 多场景支持

与规则引擎、告警服务、物可视等产品无缝对接，在云端映射设备之间的真实关系，轻松构建基于物的物联网应用

1.5 系统限制

1.5.1 项目数

每个百度云账户目前最多可创建1个设备型项目和100个数据型项目，如有更高配额需求，可提交[工单](#)申请。

1.5.2 消息长度限制

用户上传的单条消息大小限制是32KB，超过32KB的消息会被丢弃。

特别地，在计费上，每条消息的最大长度为512Bytes，超出部分将被算作多条新的消息，也就是“实际消息长度/512Bytes”的计算结果向上取整。

1.5.3 项目限制

目前，每个物接入项目并发连接数默认限制为10000。用户可通过提交[工单](#)申请更高配额。

新建连接数 每个物接入项目，每秒新建连接数默认限额为200，用户可以通过提交[工单](#)申请更高配额。

设备型项目限制 每个设备型项目默认可创建10000台。用户可以通过提交[工单](#)申请更高配额。

数据型项目限制 每个数据型项目，最多可以默认10000个用户、10000个身份及10000个策略。如有更高配额需求，可提交[工单](#)申请。

第2章 产品定价

2.1 计费策略

IoT Hub为预付费服务，实行阶梯定价，根据每月可收发的消息额度及购买时长计费。用户在购买服务前对每月发布/订阅的消息条数进行预估，选择购买相应的消息额度。

注意：

- 以下计价单位中的条数代表发布消息（PUB）和订阅消息（SUB）之和。例如：有5台设备订阅了同一个主题（topic），第6台设备向该主题发布1条消息，则总条数的计算方法为：1（PUB）+5（SUB）=6。
- 在计费上，每条消息的最大长度为512Bytes，超出部分将被算作是一条或多条新的消息，也就是“实际消息长度/512Bytes”的计算结果向上取整。在实际使用中，用户上传的单条消息大小限制是32KB，超过32KB的消息会被丢弃。
- 当月消息用量超过所购买的消息额度后，新发送的消息将不会被接收处理。
- 当月剩余消息额度不累计至次月。

您可以通过[物接入价格计算器](#)快速获取价格。

The screenshot shows the 'Purchase Configuration' section of the price calculator. It includes fields for 'Resource Type' (selected 'IoT Hub'), 'Current Region' (selected 'North China - Beijing'), and 'Purchase Specification' (set to 2 million messages per month). Below this, the 'Purchase Information' section shows 'Purchase Duration' (selected '1 month') and a note: 'For the first month, up to 1 million messages are free. If it exceeds the free quota, we will charge accordingly.'

图 2.1: 器

计算公式

以下通过一个实例介绍IoT Hub的费用计算方法。

以用户选择购买每月收发1亿（即100百万）条消息额度为例，用户需要支付的月费用计算公式为：

$$0*1+1*4+0.9*5+0.8*40+0.7*50=75.5 \text{元/月}$$

2.1.1 升级和续费

升级和续费操作仅针对计费套餐，每个用户只能创建一个计费套餐，所有项目将共享该套餐的额度。

注意：不支持降级操作，如果用户需要降低套餐额度，可在当前套餐过期后重新创建套餐。

升级 如果套餐额度不足，可选择对套餐配置进行升级。用户可在计费套餐详情中获取本月账期内剩余额度。当发布和订阅的消息总数达到已购额度的75%和90%时，系统将分别发送用量提醒消息，当月消息用量超过所购买的消息额度后，新发送的消息将不会被接收处理。为避免因达到额度上限导致业务中断，请及时进行升级配置操作。

1. 选择“产品服务>物联网服务>物接入IoT Hub”，进入项目列表。
2. 在页面上方的“用量与计费”选择“计费套餐详情”，点击“配置升级”，在配置升级页面中重新提交购买规格，点击“下一步”进入在线支付页面进行支付。

升级时仅需支付差价，即：[当月剩余天数 / 30 * 阶梯价格](#)。支付成功后，用户当前剩余额度为：新套餐额度 - 当前已用额度。

续费 IoT Hub服务到期前7天，系统会通过邮件及短信给您发送即将到期提醒通知。为了避免服务中断，请在服务到期前进行续费。服务到期后立即停止，系统会通过邮件及短信发送欠费停服通知。数据为您保留30天，期间不收取费用，30天内未充值则释放，释放前1天和释放时系统都会发送释放通知。

1. 选择“产品服务>物联网服务>物接入IoT Hub”，进入项目列表。
2. 在页面上方的“用量与计费”选择“计费套餐详情”，点击“续费”，进入计费套餐详情页面。
3. 在续费页面中选择续费时长，点击“下一步”进入在线支付页面进行支付。

2.2 计费项目

2.2.1 计费消息

- 发布(Pub)消息

- 订阅 (Sub) 消息

注意：含被动触发的 Pub 消息，如更新物影子状态时触发发布的 documents、snapshot，及返回状态所用的 accepted 或 rejected 等主题消息

2.2.2 免费消息

MQTT 与云端服务进行建立连接、心跳保持等操作时，不对操作本身的请求进行计费，列表如下：

- Connect
- Disconnect
- Ping
- PubAck
- SubAck
- Subscribe
- Unsubscribe

2.3 到期提醒和处理

2.3.1 用量提醒

当发布和订阅的消息总数达到已购额度的 75% 和 90% 时，系统将分别发送用量提醒消息，当月消息用量超过所购买的消息额度后，新发送的消息将不会被接收处理。为避免因达到额度上限导致业务中断，请及时进行升级续费操作。

2.3.2 到期提醒

IoT Hub 服务到期前 7 天，系统会通过邮件及短信给您发送即将到期提醒通知。

2.3.3 到期后处理

服务到期后立即停止，系统会通过邮件及短信发送欠费停服通知。数据为您保留 30 天，期间不收取费用，30 天内未充值则释放，释放前 1 天和释放时系统都会发送释放通知。

第3章 快速入门

3.1 概述

说明：如果您还不了解MQTT协议，推荐您首先查看[MQTT协议介绍](#)，了解MQTT的工作原理。

本文档用于帮助用户试用物接入服务或快速完成物接入服务的部署，如果您想要了解更多功能，请参看[操作指南](#)。

3.2 步骤一：注册并登录IoT Hub

在使用物接入服务前，您需要创建一个百度云账号，请按照下述步骤进行注册和登录。

1. 注册并登录百度云平台，请参考[注册和登录](#)。
2. 如果未进行实名认证，请参考[实名认证操作方法](#)完成认证。
3. 登录成功后，导航栏选择“产品服务 > 物联网服务 > 物接入IoT Hub”，即可开始使用物接入服务。



3.3 步骤二：创建计费套餐

在[创建项目](#)之前应先创建计费套餐并设定每个月收发消息的额度，系统将根据额度自动计算每个月的服务费用。每个用户只能创建一个计费套餐，所有项目将共享该套餐的额度。

1. 登录[百度云官网](#)，点击右上角的“管理控制台”，快速进入控制台界面。
2. 选择“产品服务 > 物联网服务 > 物接入IoT Hub”，进入服务页面。
3. 点击“项目列表”，选择一种计费方式，关于产品的定价和费用计算方法，请参看[产品定价](#)。

选择所需套餐后，点击“下一步”进入在线支付页面进行支付。支付成功后，用户可进入“项目列表”，创建物接入项目。



3.4 步骤三：创建项目

连接物接入服务需要先创建一个项目，每个项目会为您对应一个接入点（endpoint）。一个项目表示一个完整的物接入服务。

登录物接入控制台页面，点击“创建项目”，填写需要创建 IoT Hub 服务的项目名称、选择项目类型设备型（推荐）或数据型，并提交即可。通过项目可以将不同项目的设备进行隔离和管理。

The screenshot shows the 'Create Project' page. It includes fields for 'Current Region' (set to 华南 - 广州), 'Project Name' (输入框), 'Description' (输入框), and 'Project Type' (radio buttons: 设备型, 推荐). A note at the bottom says: '温馨提示：请谨慎选择项目类型，选择后暂不支持修改。如何选择>' (Tip: Please choose the project type carefully, as it cannot be modified after selection. How to choose?). On the right, there's a 'Selected Configuration' panel with 'Region: 华南 - 广州' and 'Type: 设备型'. A large blue 'Submit' button is at the bottom right.

创建项目后，在项目列表页可以看到物接入默认提供的三类地址。选择不同的地址，意味着您可以通过不同的方式连接到百度云物接入。

- `tcp://yourendpoint.mqtt.iot.gz.baiduce.com:1883`, 端口 1883, 不支持传输数据加密，可以通过MQTT.fx客户端连接。

- <ssl://yourendpoint.mqtt.iot.gz.baiduce.com:1884>, 端口 1884, 支持SSL/TLS加密传输, MQTT.fx客户端连接, 参考[配置MQTT客户端](#)。
- <wss://yourendpoint.mqtt.iot.gz.baidubce.com:8884>, 端口 8884, 支持 Websockets 方式连接, 同样包含 SSL 加密, 参考[Websockets Client](#)。

说明：当前每个账户能创建 100 个项目，含 1 个设备型项目和 99 个数据型项目。

3.5 步骤四：获取连接信息

物接入提供设备型和数据型两种项目类型。此步骤中，将分别予以描述，读者根据自己期望使用的项目类型，选其一进行了解即可。

3.5.1 设备型项目

成功创建物接入项目后，点击项目名称，进入配置物接入设备型项目页面，创建影子并获取连接信息。具体操作步骤如下：

1. 创建物模型：点击项目名称进入后，选择「物模型」，进入物模型列表页面，点击「新建物模型」。填写名称、属性等信息后，点击创建即可。

| 属性名称 | 显示名称 | 类型 | 默认值 | 单位 | 操作 |
|-------------|-------|--------|-----|-----|--------|
| temperature | 温度 | number | | °C | Delete |
| humidity | 湿度 | number | | %RH | Delete |
| lightColor | 指示灯颜色 | string | | | Delete |

说明：物模型用来表示一类（或同一型号的一批）设备。可为设备定义一套属性模板，在创建物影子时可以引用该模板，实现业务的快速部署。

2. 创建物影子：左侧选择「物影子」，进入物影子列表页面，点击「新建物影子」。输入名称，并选择需要的物模型（这里我们选择了刚才创建的 `monitor`），点击创建即可完成，此时会弹出连接信息。

创建物影子

返回物影子列表

名称：

描述：
0-128个字符

选择物模型： 模型详情

存储配置： OFF

创建 取消

说明：物影子与真实设备一一对应，是设备在云端体现。通过一组 json，反映设备最后一次上报的状态信息。

3. 获取连接信息：物影子创建完成时，会弹出连接信息，建议下载保存备用。



若连接信息未及时保存，或连接密码丢失，亦可点击相应影子的卡片，进入物影子详情页面，随后通过以下路径查看连接信息及重新生成密钥。



说明：出于安全考虑，物影子连接密钥无法找回，若已丢失，需重新生成。生成后原密码将失效，使用原密码的连接也将被断开。

4. 查看主题 (Topic) 列表：MQTT 消息的收发是基于主题的。设备型项目的主题是由系统定义的，可点击相应影子的卡片，通过以下页面查看，可根据实际需求选择主题。每个消息均需为 json 格式。

The screenshot shows the Baidu IoT Cloud Platform's configuration interface. On the left, there's a sidebar with options like 'Object Model', 'Thing Shadow' (which is highlighted in blue), 'Permission Groups', and 'OTA Remote Upgrade'. The main content area has a header 'Return to Thing Shadow List' and 'myMonitor' with a 'Offline' status. Below this, there are tabs for 'Thing Shadow Details', 'Object Details', and 'Interaction' (which is highlighted with a red box). Under 'Interaction', there are sections for 'HTTP/HTTPS' (with a note about Restful API), 'IoT Edge SDK' (with a note about the platform providing IoT Edge SDK), and 'Universal MQTT SDK' (with a note about the system providing default topics). Three code snippets are shown: 1. \$baidu/iot/shadow/myMonitor/update (with a 'Copy' button), 2. \$baidu/iot/shadow/myMonitor/get (with a 'Copy' button), and 3. \$baidu/int/charlesw/mymonitor/delta (with a 'Copy' button).

至此，即可通过获取到的连接信息，建立设备与影子之间的连接，进行数据的上报与反控下发。

3.5.2 数据型项目

成功创建物接入数据型项目后，点击数据型项目对应的项目名称，进入配置数据型项目页面，创建设备、身份和策略。具体操作步骤如下：

1. **创建用户：**选择用户列表，点击“创建用户”，输入需要连接物接入数据型服务的用户名。为了便于灵活连接物接入数据型服务，每一个用户都需要绑定相应的身份。
2. **创建身份：**在身份输入框内点击“创建”，快速创建身份。输入身份名称，以及为每个身份授权的策略。
3. **创建策略：**在策略输入框内点击“创建”，快速创建策略。输入策略名称，主题，选择该身份拥有的权限：发布消息(publish)、订阅消息(subscribe)。

说明：每个策略可以创建多个主题，在创建策略弹框右侧，点击“+”可以绑定更多的主题。

4. 成功创建身份后，返回密码，在配置客户端时会用到，需要您复制保存。
5. 至此，成功配置了物接入数据型项目中的相关参数。

3.6 步骤五：通过 MQTT.fx 客户端测试连接

配置 MQTT 的应用客户端，可以快速验证是否可以实现与物接入服务交流发送或者接收消息。

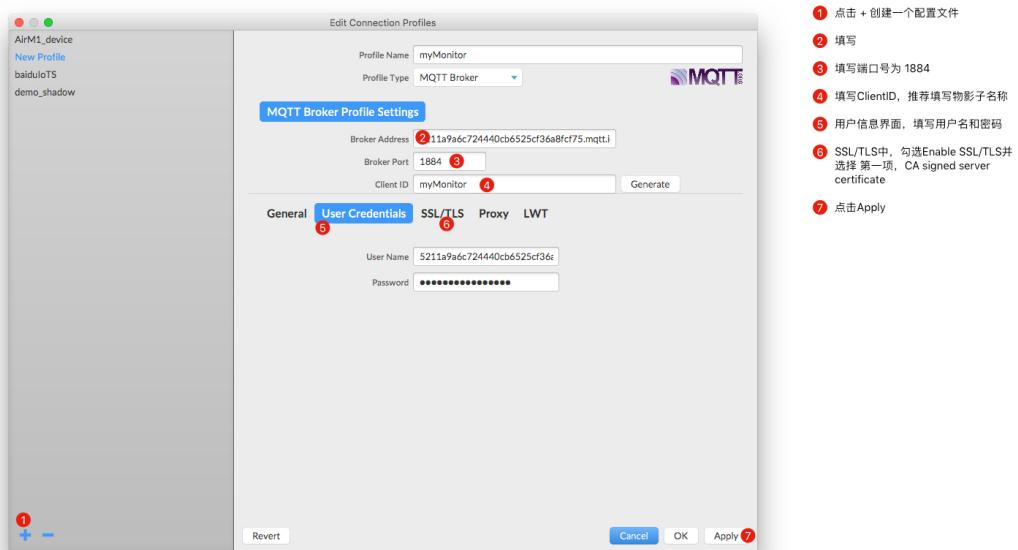
3.6.1 前提条件

登录[MQTT.fx官网](#)，找到适合的版本下载并安装MQTT.fx客户端。

注意：MQTT.fx 1.7.0版本对带有 \$ 的主题（Topic）处理存在 bug，请避免使用此版本进行测试。点击查看[MQTT.fx 官方 issue](#)

3.6.2 操作步骤

1. 打开MQTT客户端的设置页面，点击“+”按键，创建一个新的配置文件。



- 填写Connection profile相关信息：

| 参数名称 | 说明 |
|----------------|------------------|
| profile name | 配置文件名称，可随意填写 |
| Broker Address | 创建项目后返回的hostname |
| Broker Port | 1884 |

| 参数名称 | 说明 |
|-----------|--|
| Client ID | 客 户 端 ID， 支 持 “a-z”， “0-9”， “_”， “-” 字 符， 且 不 能 大 于 128bytes， UTF8编码 |

- 选择User Credential，输入创建 IoT Hub 服务返回的 username/password，参考[配置项目](#)。
- 配置SSL/TLS安全认证，勾选 `Enable SSL/TLS`，选择[CA signed server certificate](#)认证。

点击“Apply”按键，完成客户端配置。

2. 返回MQTT客户端界面，选择新创建的配置文件，点击“connect”按键连接服务。

连接成功时，控制台物影子页面的在线状态指示灯也会亮起。物影子在线状态，需 clientID 与物影子名称一致，且使用物影子指定的用户名、密码进行连接。

3. 成功连接后，即可开始订阅消息。

打开Subscribe标签，填写主题topic，例如[\\$baidu/iot/shadow/myMonitor/update/accepted](#) 及 [\\$baidu/iot/shadow/myMonitor/update/rejected](#)，分别被用作 update 被接受和拒绝时返回信息，选择默认的QoS 0，点击“Subscribe”进行订阅操作。

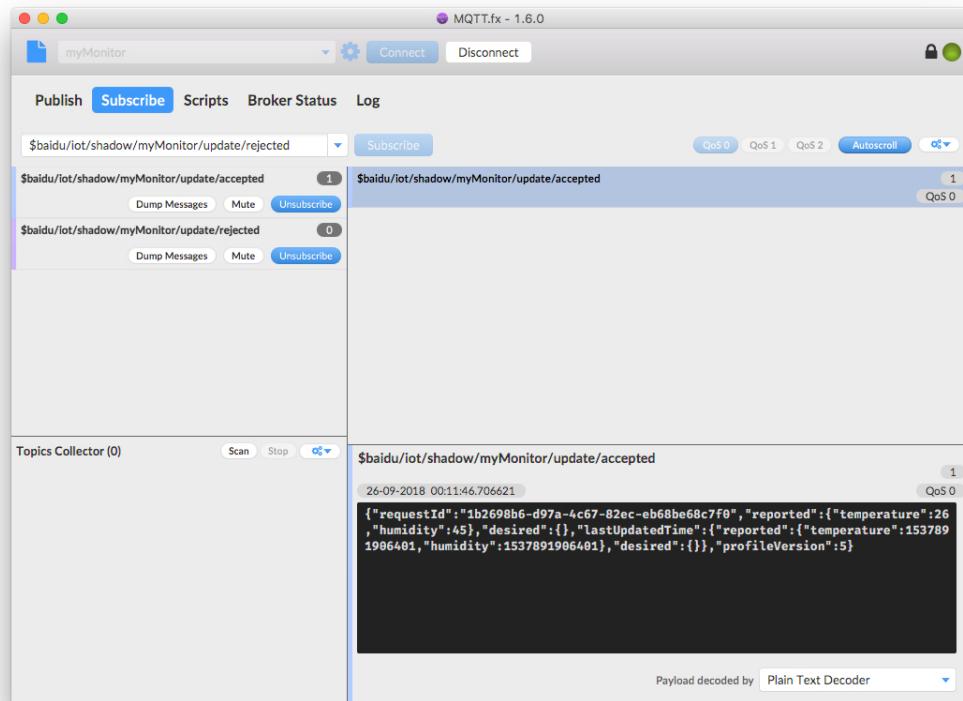
4. 发布消息。

打开Publish标签，填写主题topic，例如[\\$baidu/iot/shadow/myMonitor/update](#)，选择默认的QoS 0，输入框中填写以下信息

```
{
  "reported": {
    "temperature": 26,
    "humidity": 45
  }
}
```

其中[temperature](#)和[humidity](#)及其值，均可自行替换为创建物模型时所指定的属性值，点击“Publish”进行发布操作。

5. 返回Subscribe界面，即可看到已接收的订阅消息，我们上报的状态已经被影子接受了，参见下图。



6. 物影子的反控

打开Subscribe标签，填写主题topic，这次我们在控制台中，物影子「交互」页面找到反控信息下发时会触发的主题 [\\$baidu/iot/shadow/myMonitor/delta](#)。

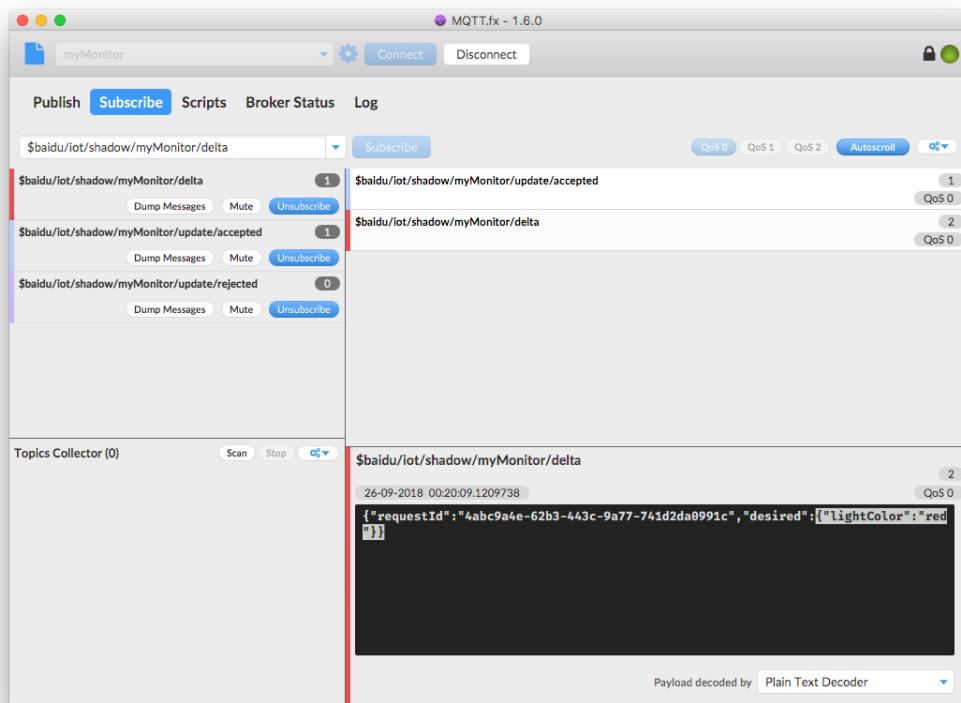
接下来尝试在物影子详情页，点击编辑，输入期望值，点击保存。

The screenshot shows the 'myMonitor' shadow detail page. The 'Edit' button (marked with ①) and the 'Desired' value input field for the 'lightColor' property (marked with ②) are highlighted.

| 属性名称 | 显示名称 | 类型 | 默认值 | 单位 | 当前值 | 修改时间 | 期望值 | 发送时间 |
|-------------|-------|--------|-----|-----|-------|---------------------|----------------------------------|------|
| temperature | 温度 | number | | °C | 26 | 2018-09-26 00:11:46 | <input type="text"/> | N/A |
| humidity | 湿度 | number | | %RH | 45 | 2018-09-26 00:11:46 | <input type="text"/> | N/A |
| lightColor | 指示灯颜色 | string | | | green | 2018-09-25 23:51:41 | <input type="text" value="red"/> | N/A |

Buttons at the bottom: '保存' (Save) and '取消' (Cancel).

保存后，切换至 MQTT.fx 的 Subscribe标签，可以看到收到了一条新的信息，正是我们刚才修改的期望值。



3.7 多种方式使用物接入服务

本例中，采用的是物接入数据型项目示例，设备型项目类型，但需注意主题和消息格式的选择及限制（详见步骤四-设备型项目-4. 查看主题（Topic）列表）。

3.7.1 Python

通过此 demo 试用物接入服务时，用户无需登录控制台，物接入服务的云端配置已经由百度云工程师预置好，可直接使用以下代码测试消息收发。有关控制台的配置方法，请参看[操作指南](#)。

注意事项：

- 工程中所使用的物接入项目隶属于百度天工研发团队。请勿在真实产品中使用。
- MQTT协议规定两个连接的client id不能相同。否则后连的会踢走先连的。如果两个设备都带重连的话，相同的client id会导致互踢死循环。因此工程中的client id做了随机化，避免连接间互踢的情况发生。实际使用时，请选择一种 clientID 生成规则，如影子名称、设备网卡 MAC 地址等。
- 本工程的用户名和密码只能访问“demoTopic”这个主题。访问其它主题会导致连接断开。

百度云工程师已在云端配置了物接入项目“iotfreetest”，并为其创建了设备“thing01”，

给主题demoTopic设置了发布和订阅权限。您可以使用[MQTT客户端](#)或调用[MQTT SDK](#)进行连接测试。连接的用户名为“iotfreetest/thing01”，密码为“YU7Tov8zFW+WuaLx9s9I3MKyclie9SGDuuNkl6o9LXo”。也可以参考以下步骤，运行工程代码测试消息收发。

1. 预安装环境

本工程需要使用Python 3以上版本。本工程在Windows、Linux和Mac上均可运行。

2. 下载项目文件

从<http://iot-demo.cdn.bcebos.com/SampleCode/TestMQTTPython.zip>处下载工程文件，并解压至磁盘。

3. 编译并运行

打开命令行工具，进入目录TestMQTTPython（该目录里有一个requirements.txt文件和一个server.py文件）。然后运行“pip install -r requirements.txt && python server.py”。这时在命令行就能看到工程向百度天工发送了消息，并且自己接收了该消息。（按“Ctrl + C”可以退出运行。）

4. 测试工程

打开源代码文件，我们能看到连接IoT Hub所使用的用户名和密码。打开MQTT.fx（[MQTT调试工具](#)），使用相同的用户名和密码连接百度天工。连接完毕之后，向topic “demoTopic”发送字符串。发送的字符串会显示在“步骤3”运行测试项目的命令行中。最后，发送“exit”字符串，工程会结束运行。

3.7.2 NodeJS

试用物接入服务时，用户无需登录控制台，物接入服务的云端配置已经由百度云工程师预置好，可直接使用以下代码测试消息收发。有关控制台的配置方法，请参看[操作指南](#)。

注意事项:

- 工程中所使用的项目隶属于百度天工研发团队。请勿在真实产品中使用。
- 工程使用SSL/TLS来连接百度天工。如果不使用SSL/TLS，把endpoint URL换成“tcp://iotfreetest.mqtt.iot.gz.baidubce.com:1883”
- MQTT协议规定两个连接的client id不能相同。否则后连的会踢走先连的。如果两个设备都带重连的话，相同的client id会导致互踢死循环。因此工程中的client id做了随机化，避免连接间互踢的情况发生。实际使用时，请选择一种clientID生成规则，如影子名称、设备网卡 MAC 地址等。
- 本工程的用户名和密码只能访问“demoTopic”这个主题。访问其它主题会导致连接断开。

百度云工程师已在云端配置了物接入项目“iotfreetest”，并为其创建了设备“thing01”，给主题demoTopic设置了发布和订阅权限。您可以使用[MQTT客户端](#)或调用[MQTT SDK](#)进行连接测试。连接的用户名为“iotfreetest/thing01”，密码为“YU7Tov8zFW+WuaLx9s9I3MKyclie9SGDuuNkl6o9LXo”。也可以参考以下步骤，运行工程代码测试消息收发。

1. 预安装环境

本工程需要使用Node.js 4以上版本。本工程在Windows、Linux和Mac上均可运行。

2. 下载项目文件

从<http://iot-demo.cdn.bcebos.com/SampleCode/TestMQTTNode.zip>处下载工程文件，并解压至磁盘。

3. 编译并运行

打开命令行工具，进入目录TestMQTTNode（该目录里有一个package.json文件和一个server.js文件）。然后运行“npm install && npm start”。这时在命令行就能看到工程向百度天工发送了消息，并且自己接收了该消息。（按“Ctrl + C”可以退出运行。）

4. 测试工程

打开源代码文件，我们能看到连接IoT Hub所使用的用户名和密码。打开MQTT.fx（[MQTT调试工具](#)），使用相同的用户名和密码连接百度天工。连接完毕之后，向topic “demoTopic”发送字符串。发送的字符串会显示在“步骤3”运行测试项目的命令行中。最后，发送“exit”字符串，工程会结束运行。

3.7.3 Java

试用物接入服务时，用户无需登录控制台，物接入服务的云端配置已经由百度云工程师预置好，可直接使用以下代码测试消息收发。有关控制台的配置方法，请参看[操作指南](#)。

注意事项：

- 工程中所使用的项目隶属于百度天工研发团队。请勿在真实产品中使用。
- 工程使用SSL/TLS来连接百度天工。如果不使用SSL/TLS，把endpoint URL换成“tcp://iotfreetest.mqtt.iot.gz.baidubce.com:1883”
- MQTT协议规定两个连接的client id不能相同。否则后连的会踢走先连的。如果两个设备都带重连的话，相同的client id会导致互踢死循环。因此工程中的client id做了随机化，避免连接间互踢的情况发生。实际使用时，请选择一种 clientID 生成规则，如影子名称、设备网卡 MAC 地址等。
- 本工程的用户名和密码只能访问“demoTopic”这个主题。访问其它主题会导致连接断开。

百度云工程师已在云端配置了物接入项目“iotfreetest”，并为其创建了设备“thing01”，给主题demoTopic设置了发布和订阅权限。您可以使用[MQTT客户端](#)或调用[MQTT SDK](#)进行连接测试。连接的用户名为“iotfreetest/thing01”，密码为“YU7Tov8zFW+WuaLx9s9I3MKyclie9SGDuuNkl6o9lXo”。也可以参考以下步骤，运行工程代码测试消息收发。

1. 预安装环境

本工程需要使用JDK 1.8和Gradle 3.3。本工程在Windows、Linux和Mac上均可运行。

2. 下载项目文件

从<http://iot-demo.cdn.bcebos.com/SampleCode/TestMQTTJava.zip>处下载工程文件，并解压至磁盘。

您也可以通过导入Maven工程的方式测试消息收发，下载地址是<http://iot-demo.cdn.bcebos.com/SampleCode/TestMQTTJavaMaven.zip>。

3. 编译并运行

1. 命令行运行

打开命令行工具，进入目录TestMQTTJava（该目录里有一个gradle.build文件和一个src文件夹）。然后运行“gradle build && gradle run”。这时在命令行就能看到工程向百度天工发送了消息，并且自己接收了该消息。（按“Ctrl + C”可以退出运行。）

2. 使用Eclipse运行

打开Eclipse。单击菜单“File”->“Import…”。然后在Import对话框中选择“Gradle”->“Gradle Project”，然后单击两次“下一步”。再选择“TestMQTTJava”目录为工程的根目录，单击“Finish”即可。最后，单击“Run”->“Run”运行工程。如果在Run的时候弹出对话框选择运行类型，请选择“Application”。

4. 测试工程

打开源代码文件，我们能看到连接IoT Hub所使用的用户名和密码。打开MQTT.fx（[MQTT调试工具](#)），使用相同的用户名和密码连接百度天工。连接完毕之后，向topic“demoTopic”发送字符串。发送的字符串会显示在“步骤3”运行测试项目的命令行中。最后，发送“exit”字符串，工程会结束运行。

3.7.4 Arduino D1 & NodeMCU

Arduino平台是非常方便的开源硬件平台。本文介绍如何使用Arduino D1和NodeMCU开发板来连接百度天工物接入服务。Arduino D1和NodeMCU都使用了一块价廉物美的Wifi芯片，ESP8266。使用ESP8266，可以让开发板非常方便的通过Wifi来连接百度天工物接入服务。

注意事项：

- 工程中所使用的项目隶属于百度天工研发团队。请勿在真实产品中使用。
- 工程使用SSL/TLS来连接百度天工。如果不使用SSL/TLS，可以把WiFiClientSecure换成WiFiClient，且把端口1884换成1883。
- MQTT协议规定两个连接的client id不能相同。否则后连的会踢走先连的。如果两个设备都带重连的话，相同的client id会导致互踢死循环。因此工程中的client id做了随机化，避免连接间互踢的情况发生。实际使用时，请选择一种clientID生成规则，如影子名称、设备网卡 MAC 地址等。
- 本工程的用户名和密码只能访问“demoTopic”这个主题。访问其它主题会导致连接断开。

1. 准备环境

1. 下载Arduino IDE并且安装对Arduino D1和NodeMCU的支持。
 2. 使用USB线把开发板接入PC，并且在IDE中选择开发板的型号。
 3. 在Arduino IDE中，点击“Sketch”->“Include Library”->“Manage Libraries...”，安装“ESP8266Wifi”，“PubSubClient”和“TaskScheduler”三个库。
2. 下载代码至开发板
1. 从<http://iot-demo.cdn.bcebos.com/SampleCode/TestESP8266.zip>处下载代码，并且解压至本地磁盘。
 2. 使用Arduino IDE打开“TestESP8266.ino”文件，修改“WIFI_SSID”和“WIFI_PASSWORD”两个变量的值，把这两个变量设置成开发板能访问的Wifi的名称和密码。
 3. 点击“Upload”按钮，编译并上传代码至开发板。
3. 运行并查看结果
打开Arduino IDE，单击“Tools”->“Serial Monitor”。观察输出的字符，可以看到开发板连接上了Wifi，然后连上了天工物接入服务，然后开始打印收到的字符串。

3.7.5 Arduino Wido

Arduino平台是非常方便的开源硬件平台。本文介绍如何使用Arduino Wido开发板来连接百度天工物接入服务。Arduino Wido使用了CC3000这款Wifi芯片。使用CC3000，可以让开发板非常方便的通过Wifi来连接百度天工物接入服务。

注意事项

- 工程中所使用的项目隶属于百度天工研发团队。请勿在真实产品中使用。
- 由于CC3000不支持SSL/TLS，因此只能使用TCP来连接天工物接入服务。
- MQTT协议规定两个连接的client id不能相同。否则后连的会踢走先连的。如果两个设备都带重连的话，相同的client id会导致互踢死循环。因此工程中的client id做了随机化，避免连接间互踢的情况发生。实际使用时，请选择一种 clientID 生成规则，如影子名称、设备网卡 MAC 地址等。
- 本工程的用户名和密码只能访问“demoTopic”这个主题。访问其它主题会导致连接断开。

1. 准备环境

1. 下载Arduino IDE。
2. 使用USB线把开发板接入PC，并且在IDE中选择开发板的型号“Arduino Leonardo”。
3. 在Arduino IDE中，点击“Sketch”->“Include Library”->“Manage Libraries...”，安装“Adafruit_CC3000”，“PubSubClient”和“TaskScheduler”三个库。

2. 下载代码至开发板

1. 从<http://iot-demo.cdn.bcebos.com/SampleCode/TestWido.zip>处下载代码，并且解压至本地磁盘。

2. 使用Arduino IDE打开“TestWido.ino”文件，修改“WIFI_SSID”和“WIFI_PASSWORD”两个变量的值，把这两个变量设置成开发板能访问的Wifi的名称和密码。
 3. 点击“Upload”按钮，编译并上传代码至开发板。
3. 运行并查看结果
打开Arduino IDE，单击“Tools”->“Serial Monitor”。观察输出的字符，可以看到开发板连接上了Wifi，然后连上了天工物接入服务，然后开始打印收到的字符串。

3.7.6 Arduino Yun

Arduino平台是非常方便的开源硬件平台。本文介绍如何使用Arduino Yun开发板来连接百度天工物接入服务。Arduino Yun自带Wifi和Ethernet模块。本文介绍在Arduino Yun上通过Wifi来连接百度天工物接入服务。

注意事项

- 工程中所使用的项目隶属于百度天工研发团队。请勿在真实产品中使用。
- 由于Arduino Yun不支持SSL/TLS，因此只能使用TCP来连接天工物接入服务。
- MQTT协议规定两个连接的client id不能相同。否则后连的会踢走先连的。如果两个设备都带重连的话，相同的client id会导致互踢死循环。因此工程中的client id做了随机化，避免连接间互踢的情况发生。实际使用时，请选择一种 clientID 生成规则，如影子名称、设备网卡 MAC 地址等。
- 本工程的用户名和密码只能访问“demoTopic”这个主题。访问其它主题会导致连接断开。

1. 准备环境

1. 下载Arduino IDE。
2. 使用USB线把开发板接入PC，并且在IDE中选择开发板的型号“Arduino Yun”。
3. 在Arduino IDE中，点击“Sketch”->“Include Library”->“Manage Libraries...”，安装“PubSubClient”和“TaskScheduler”两个库。
4. 复位Arduino Yun的Wifi，并设置其连接上本地的Wifi连接。

2. 下载代码至开发板

1. 从<http://iot-demo.cdn.bcebos.com/SampleCode/TestYun.zip>处下载代码，并且解压至本地磁盘。
2. 使用Arduino IDE打开“TestYun.ino”文件，点击“Upload”按钮，编译并上传代码至开发板。

3. 运行并查看结果

打开Arduino IDE，单击“Tools”->“Serial Monitor”。观察输出的字符，可以看到开发板连接上了Wifi，然后连上了天工物接入服务，然后开始打印收到的字符串。

3.7.7 MQTT.fx

MQTT.fx简介

MQTT.fx是一款免费的MQTT测试工具，我们可以使用它来连接百度天工的物接入服务并测试消息收发。

注意：MQTT.fx 1.7.0 版本对带有 \$ 的主题（Topic）处理存在 bug，请避免使用此版本进行测试。点击查看[MQTT.fx 官方 issue](#)

下载MQTT.fx

- 源站：“<http://www.jensd.de/apps/mqttfx/>”
- 国内：“<http://mqttfx.bceapp.com/>”

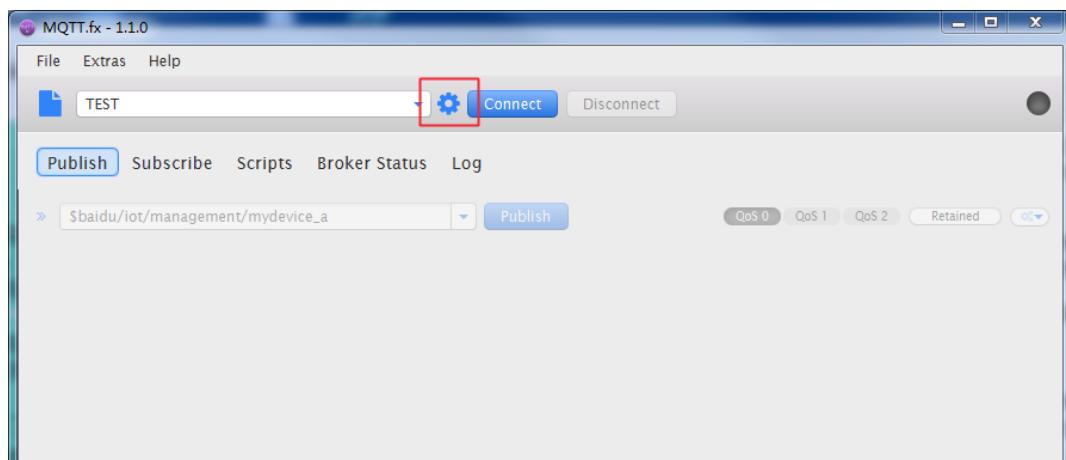
连接百度天工平台的物接入服务

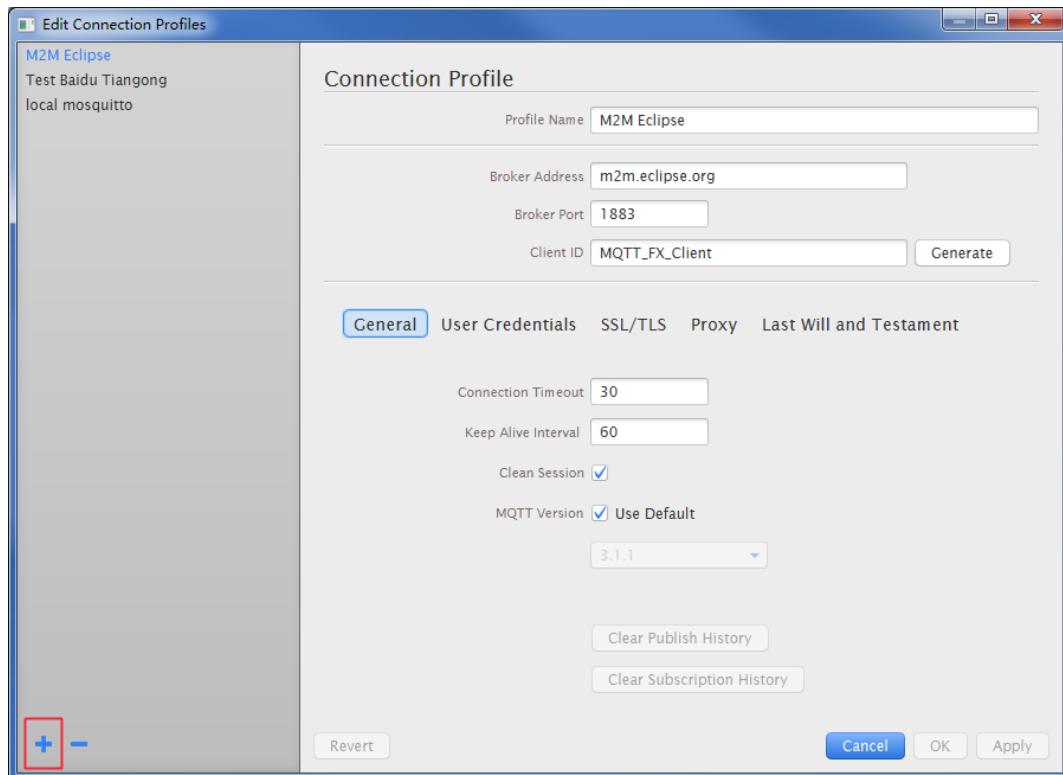
注意事项：

- 针对同一个设备的连接，Client Id 不可以重复。
- 本文使用SSL/TLS连接物接入。如果不使用SSL/TLS，请把端口改成1883并且取消对“Enable SSL/TLS”的勾选。
- 本文的用户名和密码只有访问“demoTopic”这一个主题的权限。访问其它主题会导致连接断开。
- 本文的用户名和密码所有权属于百度天工研发团队，仅供冒烟测试使用。请不要在真实产品中使用。

百度云工程师已在云端配置了物接入项目“iotfreetest”，并为其创建了设备“thing01”，给主题demoTopic设置了发布和订阅权限。连接的用户名为“iotfreetest/thing01”，密码为“YU7Tov8zFW+WuaLx9s9I3MKyclie9SGDuuNkl6o9LXo=”。

1. 打开MQTT.fx，单击“设置”图标。然后单击弹出的对话框的左下角的“添加”图标。

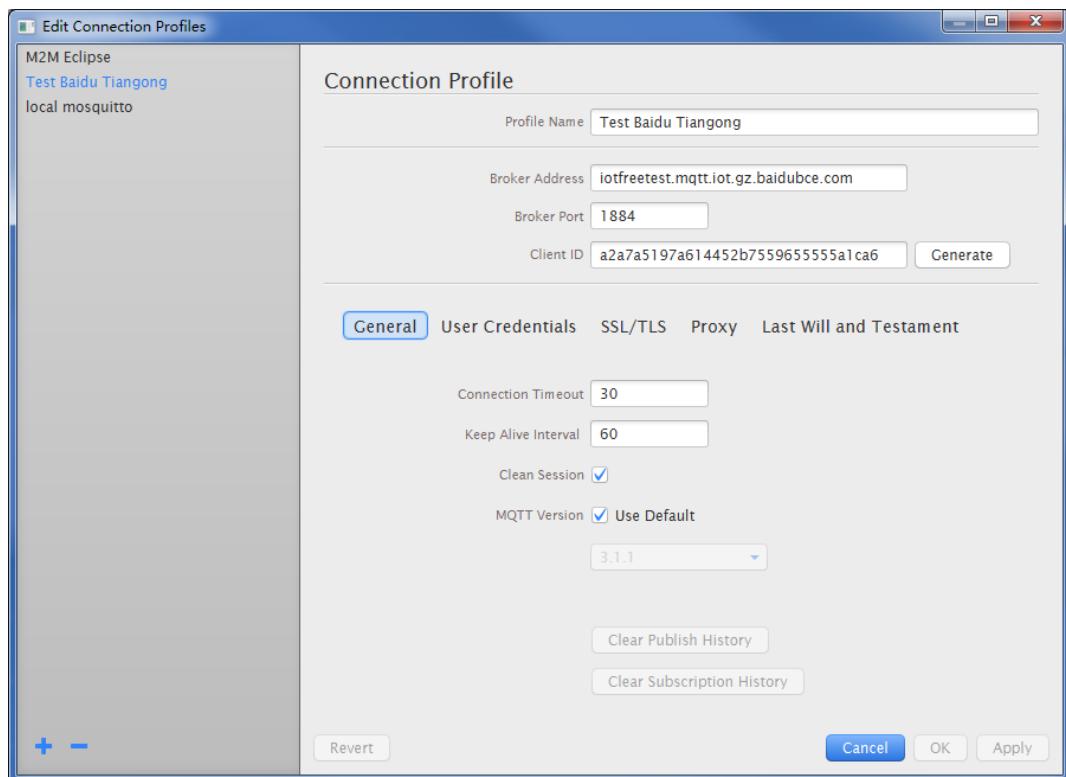




2. 按照下面的设置填写相应字段：

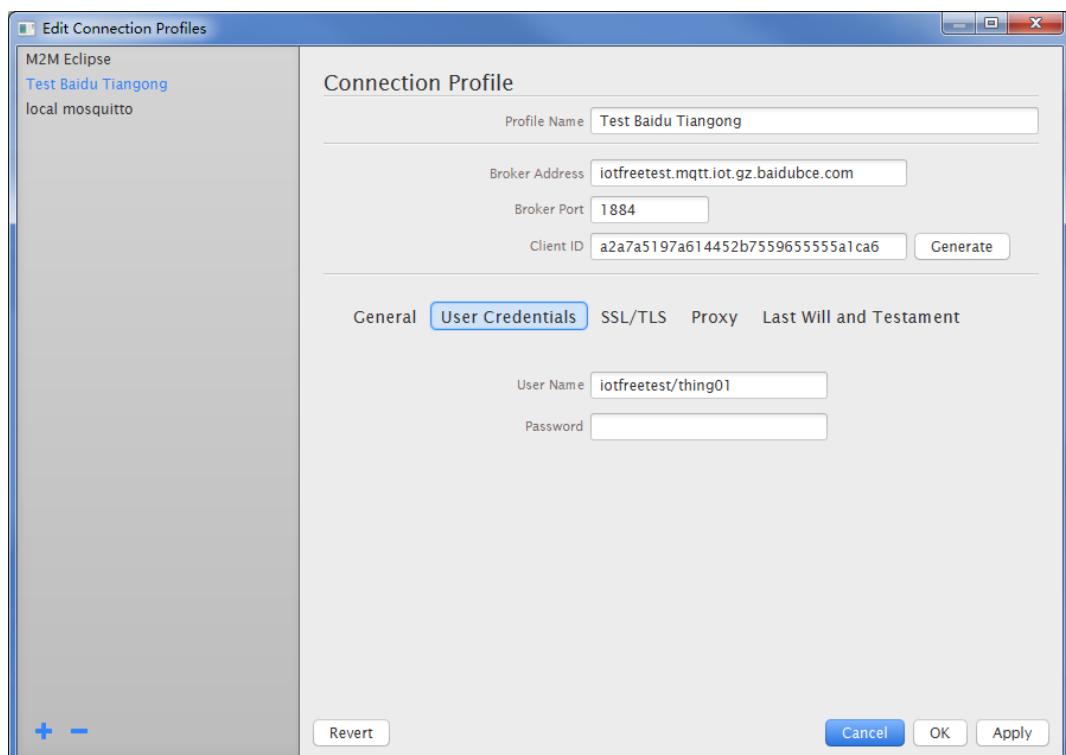
- Profile Name: Test Baidu Tiangong
- Broker Address: iotfreetest.mqtt.iot.gz.baidubce.com
- Broker Port: 1884
- Client ID: MQTT\FX\Client_81923749

注意，由于同一个Endpoint的Client Id不允许重复，因此上面的Client Id请确保足够随机。

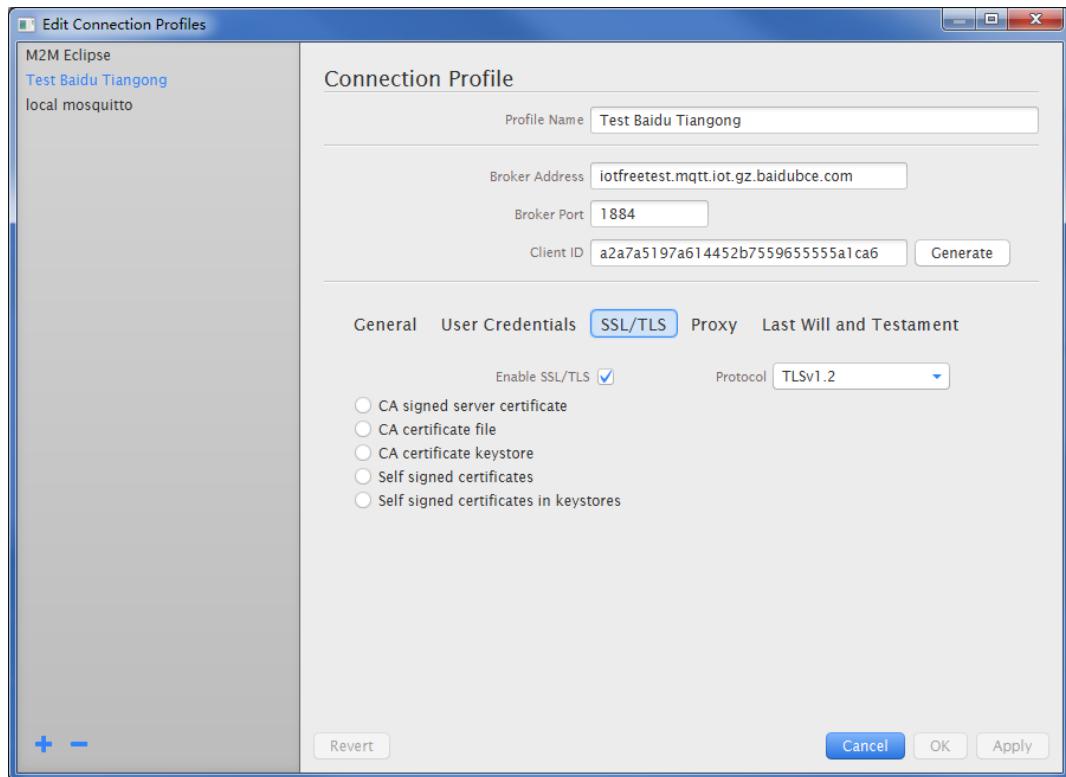


3. 选择“User Credentials”选项卡，并且按以下字段填写：

- User Name: iotfreetest/thing01
- Password: YU7Tov8zFW+WuaLx9s9I3MKyclie9SGDuuNkl6o9LXo=

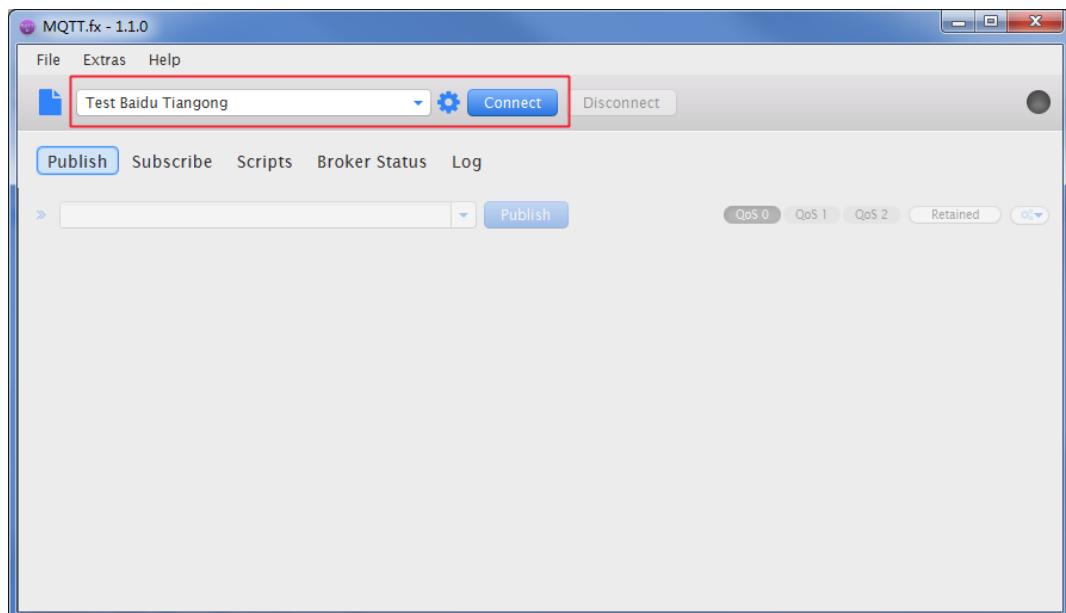


4. 选择“SSL/TLS”选项卡，勾上“Enable SSL/TLS”，并选择“CA signed server certificate”。

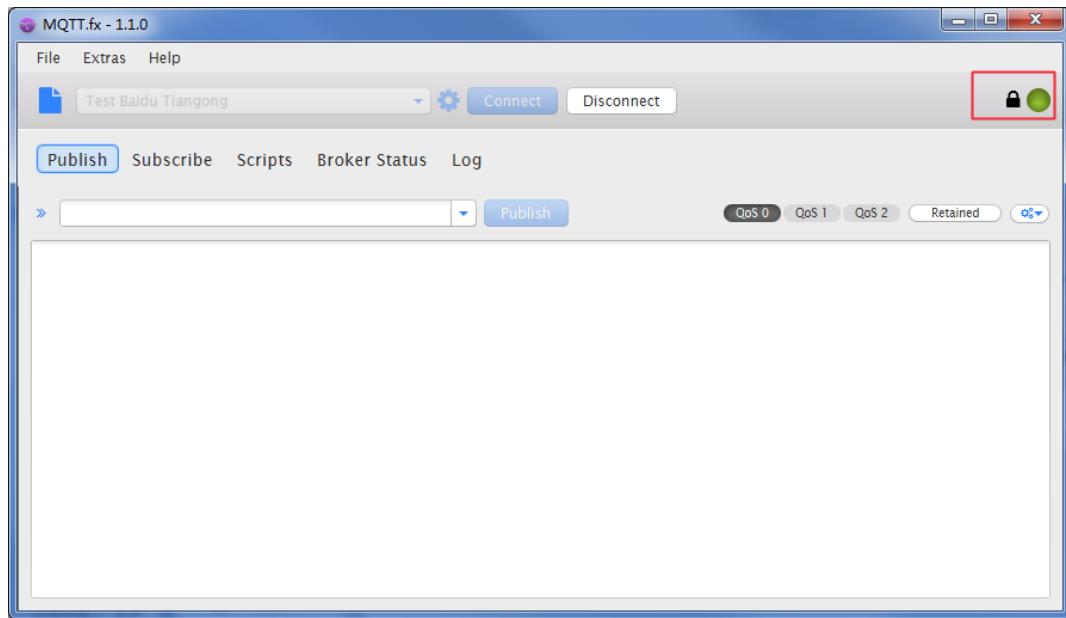


单击“OK”按钮保存设置。

5. 回到主对话框，确保选择了“Test Baidu Tiangong”，然后单击“Connect”按钮。

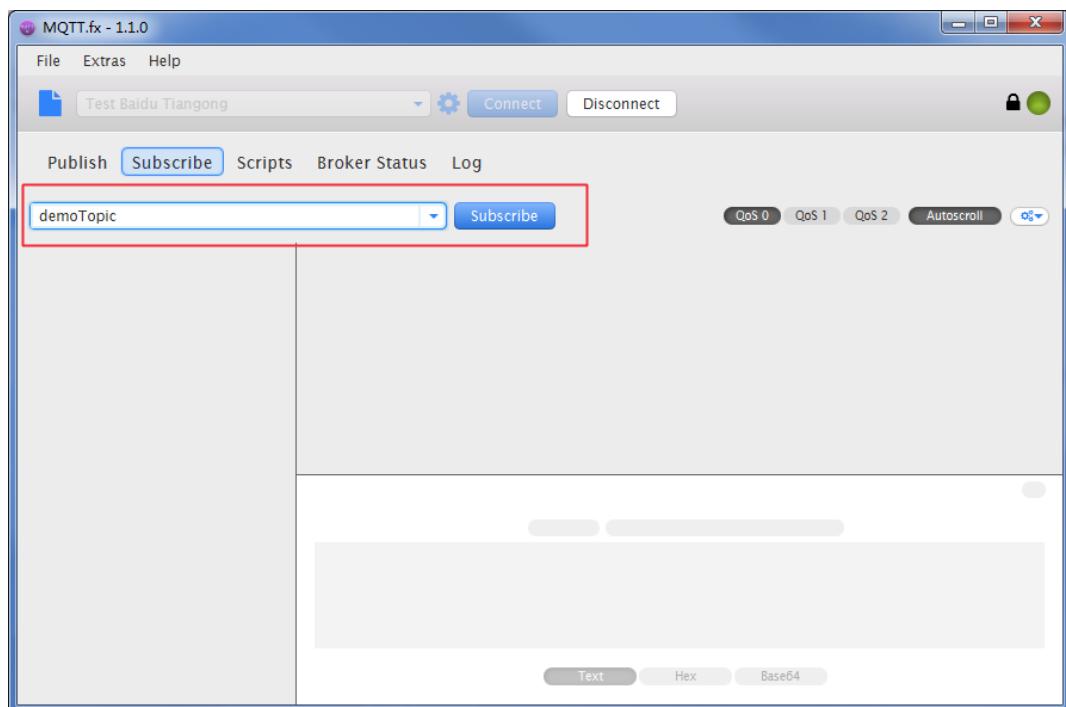


连接状态变绿，说明连接上了百度天工物联网平台的物接入服务。

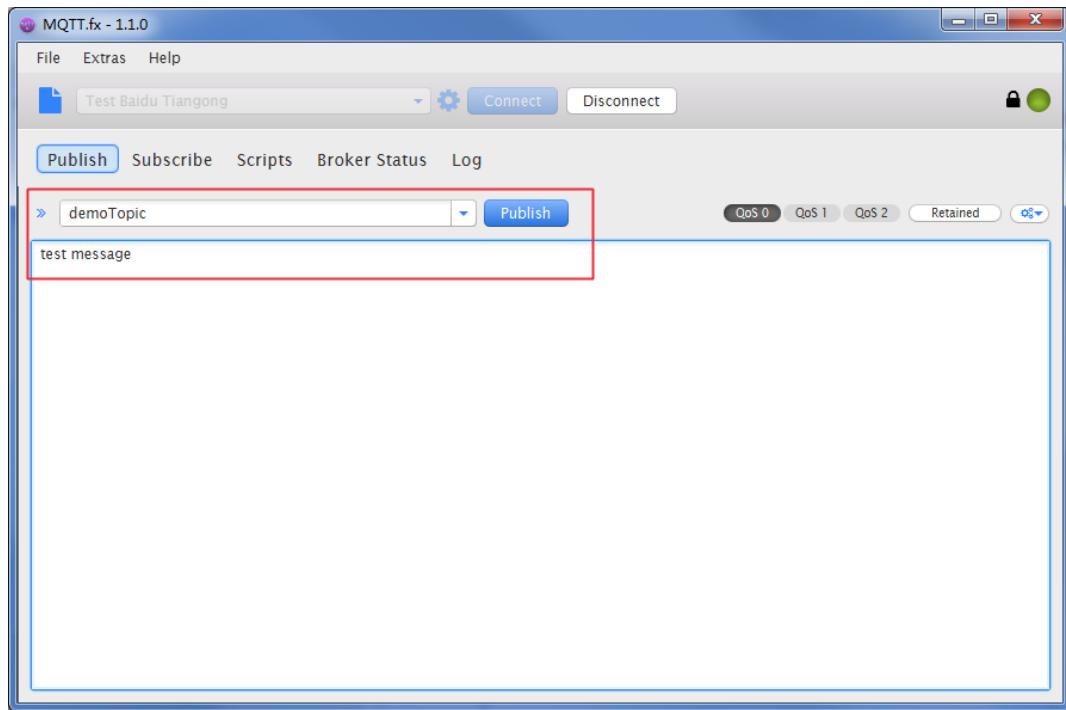


4. 测试消息收发

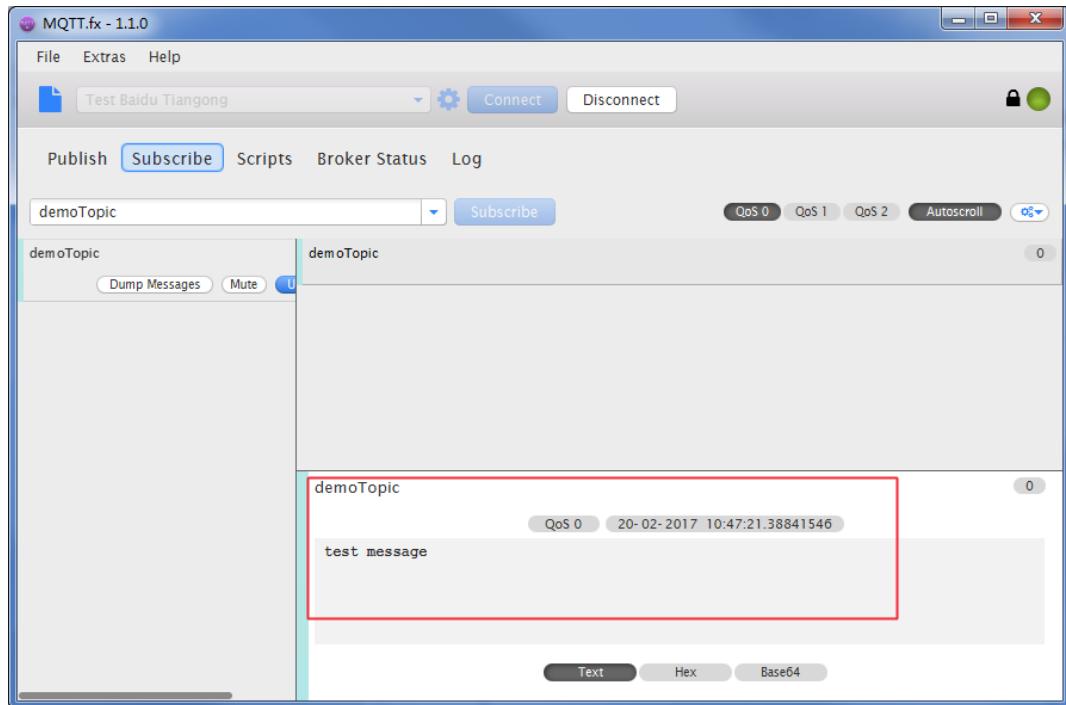
单击“Subscribe”选项卡，然后在编辑框中输入demoTopic，最后单击“Subscribe”按钮。“Subscribe”按钮变灰表示监听“demoTopic”这个主题成功。



单击“Publish”选项卡，向demoTopic发送消息。



单击“Subscribe”选项卡，查看是否收到了消息。



右下方的列表里出现了内容，则表示收到了消息。

第4章 操作指南

4.1 通用功能

4.1.1 连接建立与断开事件订阅

当一个 MQTT 连接建立或者断开时，会触发一条系统主题的通知。订阅此通知可获取连接建立及断开事件。

设备型项目可通过超级权限订阅的连接此类别相关主题。

对应主题及消息详情如下：

获取连接建立事件 订阅 [\\$baidu/sys/event/connect](#)

消息格式：

```
{  
    "deviceName": "a",  
    "connectionType": "CONNECT",  
    "version": 79974,  
    "timestamp": 1534237636338,  
    "clientId": "client-283216bb-26b6-4779-9506-4d10f6a49c5f"  
}
```

获取连接断开事件 订阅 [\\$baidu/sys/event/disconnect](#)

消息格式：

```
{  
    "deviceName": "a",  
    "connectionType": "DISCONNECT",  
    "version": 79974,  
    "timestamp": 1534237636338,  
    "clientId": "client-283216bb-26b6-4779-9506-4d10f6a49c5f"  
}
```

4.1.2 持久化会话和消息缓存

物接入支持 MQTT 标准中的会话持久化及消息缓存机制。

如果客户端在连接的时候指定Clean Session=false，那这个会话将成为一个持久化会话。如果客户端异常离线，物接入将缓存“QoS=1”的消息，待客户端重新连接后将消息发送至客户端。需要注意以下情况：

- 消息只能缓存24小时。
- 如果缓存消息的数量较大，推荐使用百度Kafka服务。
- 客户端重新连接时Clean Session需置为False (flag=0)。

如果Clean Session为True(flag=1)，物接入会丢弃已经缓存任何会话状态信息，创建一个新的会话连接。

如下例所示：

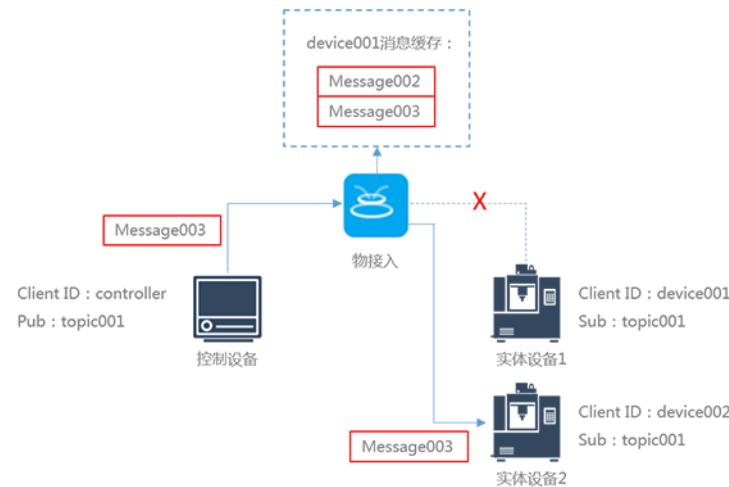
1. 控制用户和实体设备1正常接入物接入，控制用户发布“QoS=1”的消息Message001，实体设备1可以正常接收到。



2. 如果实体设备1异常离线，此时物接入将缓存所有发送至该用户的消息。

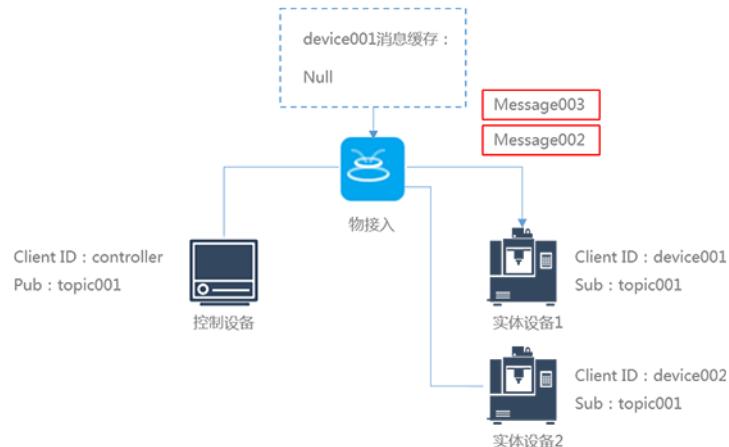


3. 此时如果有其它订阅了相同主题的设备连接至物接入，可以正常接收到控制用户新发送的消息，但无法接收到之前被缓存的消息。

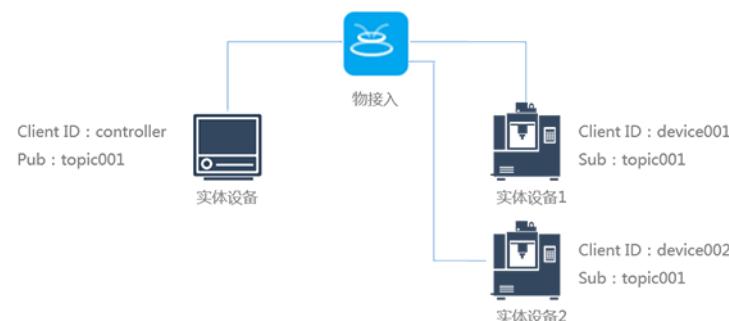


4. 实体设备1重新接人物接入，根据状态的不同，物接入有以下两种处理方式：

- 如果Clean Session置为False，此时物接入将所有缓存的消息转发至实体设备1。



* 如果Clean Session为True，物接入会丢弃已经缓存任何会话状态信息，为实体设备1创建一个新的会话连接。



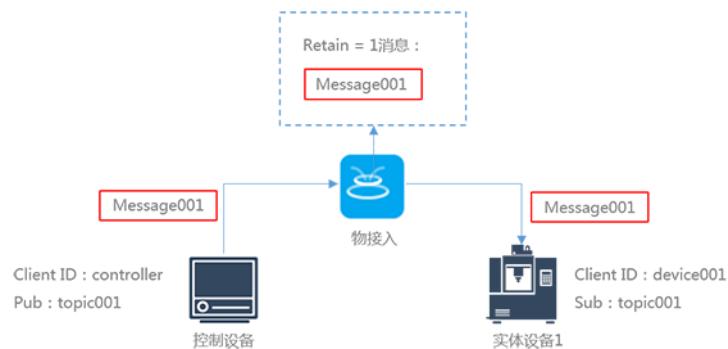
4.1.3 保留 (Retain) 消息

物接入支持 MQTT 标准中的保留消息机制。

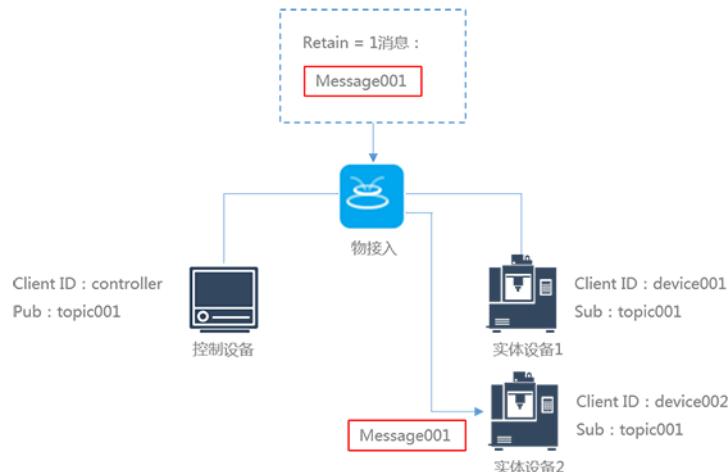
如果 Publish 消息固定头部 Retain 标记为1，物接入会持久保存此消息，直到该消息被新的 Publish 消息 (Retain=1) 覆盖或用户主动清除该消息。

对于“Retain=1”的消息，物接入不但会将该消息发送给所有当前的订阅者，同时新的接入用户也会收到该消息，如下图所示：

1. 控制用户和实体设备1正常接入物接入，控制用户发布“Retain=1”的消息Message001，实体设备1可以正常接收到并且物接入持久保存该消息。



2. 新实体设备2连接后，物接入持久保存的消息发布给新接入用户。



4.1.4 HTTP方式更新主题

除标准 MQTT 方式以外，您也可以通过 HTTP 方式更新主题。针对不同地域，入口域名不同，对应域名如下：

- 北京: api.mqtt.iot.bj.baidubce.com

- 广州: api.mqtt.iot.gz.baidubce.com

出于安全考虑，建议使用HTTPS访问。

| URI | HTTP 方式 | Content Type | Return Type |
|--|---------|--------------------------|---------------------------------|
| <code>https://api.mqtt.iot.gz.baidubce.com/v1/proxy? qos=0&topic=yourTopic&retain=false</code> | POST | application/octet-stream | application/json; charset=UTF-8 |

请求参数

| 名称 | 是否必选 | 含义 |
|--------|------|---------------------------------------|
| qos | Y | 该消息的QoS取值，可选0或1 |
| topic | Y | topic名称，客户端将向指定的topic发布消息 |
| retain | N | retain标记，详细介绍请参看 保留消息 |

请求头参数

| 名称 | 是否必选 | 含义 |
|---------------|------|-----------------|
| auth.username | Y | 用户名，即：影子名称/用户名 |
| auth.password | Y | 密码，即：创建身份时获得的密钥 |

发布的具体消息内容放在HTTP Content中，可以是二进制消息（向设备型项目主题Publish消息，需符合指定的json格式）。消息长度不大于32K。

请求示例

```
POST /v1/proxy?qos=0&topic=\$baidu/iot/shadow/test/update&retain=false HTTP/1.1
host: api.mqtt.iot.gz.baidubce.com
content-type: application/octet-stream

auth.username: jka2njk/test
auth.password: j92NcN8Czs1w3ifY
```

Content:

```
{
  "reported": {
    "key": value
  }
}
```

成功: 201, 失败: 4XX or 5XX

4.2 设备型项目

4.2.1 创建物影子

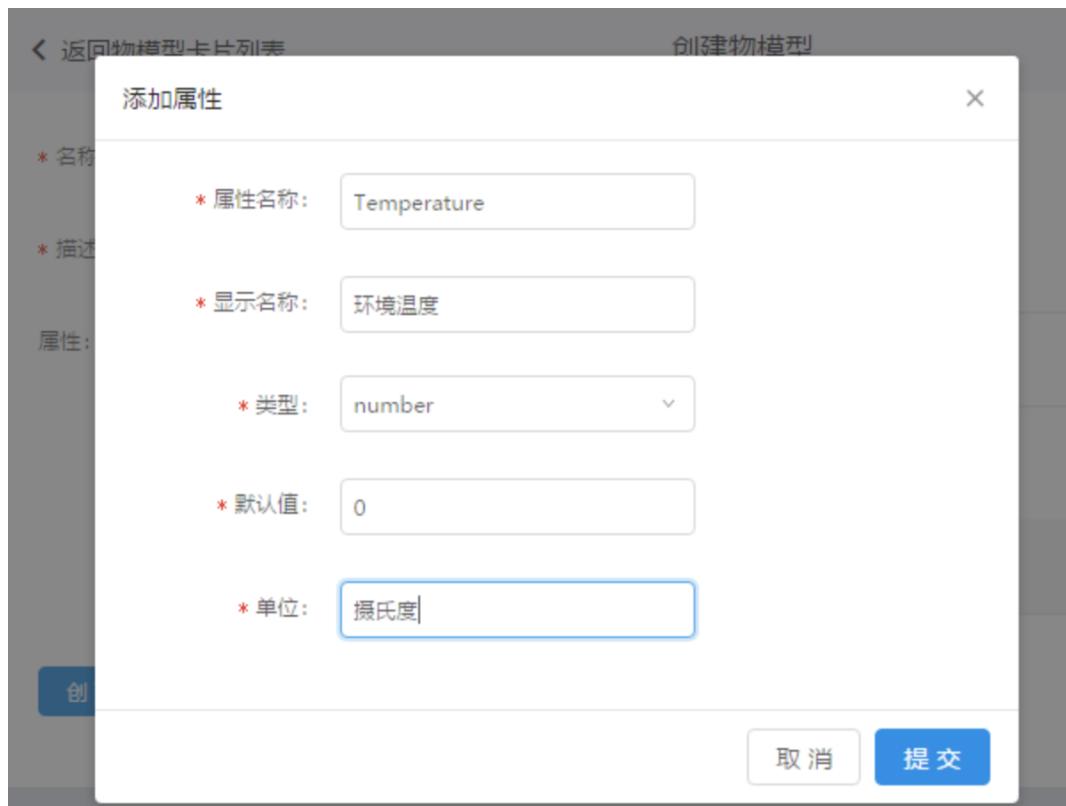
物模型

创建物模型 通过物模型可以为设备定义一套属性模板，在[创建物影子](#)时可以引用该模板，实现业务的快速部署。具体操作方法如下：

1. 登录百度云官网，点击右上角的“管理控制台”，快速进入控制台界面，选择“产品服务 > 物联网服务 > 物接入 IoT Hub”，选择“进入项目列表”。
2. 选择“创建项目”，输入项目名称、选择项目类型为「设备型」。
3. 返回项目列表，点击刚才创建的项目名称，接入项目详情页。选择“物模型”，进入物模型卡片列表；点击“新建物模型”进入物模型配置界面。



4. 填写物模型配置，包括名称、描述和属性。创建成功后物模型名称和属性名称无法修改，同一物模型下属性名称必须唯一。
点击“添加属性”为物模型新增一条属性。



完成属性配置后，点击“创建”，完成物模型创建。

The screenshot shows a list of object models. A specific card for 'floor-5' is highlighted with a red border. The card displays the object model name and a note: '说明：楼层环境监控'.

管理物模型 创建物模型以后，可以对以下信息进行编辑修改，包括：

- 物模型描述信息
- 删除或新增属性
- 属性描述、默认值、单位

具体修改方法如下：

1. 选择“产品服务>物联网服务>物接入 IoT Hub”，进入“天工物联网”控制台。
2. 点击设备型项目的项目名称，选择“物模型”，进入物模型卡片列表。

3. 选择需要修改的物模型卡片，以物模型示例为例，点击卡片进入详情页面。



4. 点击“编辑”进入编辑模式，此时可对物模型进行属性编辑等操作。完成操作后点击“保存”使配置生效。



物影子

创建物影子 在创建物影子之前，必须先[创建物模型](#)。

1. 选择“产品服务>物联网服务>物接入 IoT Hub”，进入“天工物联网”控制台。
2. 点击设备型项目的项目名称，选择“物影子”，进入物影子卡片列表；点击“新建物影子”进入物模型配置界面。



3. 填写物模型配置，包括名称、描述、选择物模型和是否开启存储配置。[数据存储TSDB](#)

创建物影子

* 名称:

描述:

* 选择物模型:

存储配置: OFF

取消

点击“创建”完成物影子创建。

[查看影子详情](#) [查看设备当前状态](#)

在影子详情界面，用户可以看到以下信息

- 模型数据，通过图表展示物模型定义的所有属性，如下图所示：

物影子的列表页展示模型数据，在关联模型中存在的字段才展示在该页。如果用户 reported了不在该物详情中的属性，通过“原始数据”查看。

The screenshot shows the Baidu Cloud Device Model Management interface. On the left, there's a sidebar with options like '返回项目列表' (Back to Project List), '物模型' (Thing Model), '权限组' (Permission Groups), and 'OTA远程升级' (OTA Remote Upgrade). The main area is titled '返回物影子列表' (Return to Device Shadow List) and shows a device shadow named '_baidu_sample_pump_instance'. It includes fields for '名称' (Name), '描述' (Description), '创建时间' (Creation Time), '最后活跃时间' (Last Active Time), '物影子版本号' (Device Shadow Version Number), and '影子状态' (Shadow Status). Below this is a table titled '属性名称' (Attribute Name) with columns for 属性名称 (Attribute Name), 显示名称 (Display Name), 类型 (Type), 默认值 (Default Value), 单位 (Unit), 当前值 (Current Value), 修改时间 (Modification Time), 期望值 (Desired Value), and 发送时间 (Send Time). The table lists several properties: FrequencyIn (输入频率, number, 0, Hz, 20, 2018-07-25 16:48:22, N/A, N/A), FrequencyOut (输出频率, number, 0, Hz, N/A, N/A, N/A, N/A), Current (电流, number, 0, A, 111, 2018-07-25 16:48:22, N/A, N/A), Speed (速度, number, 0, rpm, 1033, 2018-07-25 16:48:22, N/A, N/A), Torque (输出转矩, number, 0, %, 41.5, 2018-07-25 16:48:22, N/A, N/A), Power (输出功率, number, 0, KW, 31.9, 2018-07-25 16:48:22, N/A, N/A), and DC_bus_voltag (直流侧电压, number, 0, V, 543, 2018-07-25 16:48:22, N/A, N/A). The 'Model Data' tab is highlighted with a red circle.

| 属性名称 | 显示名称 | 类型 | 默认值 | 单位 | 当前值 | 修改时间 | 期望值 | 发送时间 |
|---------------|-------|--------|-----|-----|------|---------------------|-----|------|
| FrequencyIn | 输入频率 | number | 0 | Hz | 20 | 2018-07-25 16:48:22 | N/A | N/A |
| FrequencyOut | 输出频率 | number | 0 | Hz | N/A | N/A | N/A | N/A |
| Current | 电流 | number | 0 | A | 111 | 2018-07-25 16:48:22 | N/A | N/A |
| Speed | 速度 | number | 0 | rpm | 1033 | 2018-07-25 16:48:22 | N/A | N/A |
| Torque | 输出转矩 | number | 0 | % | 41.5 | 2018-07-25 16:48:22 | N/A | N/A |
| Power | 输出功率 | number | 0 | KW | 31.9 | 2018-07-25 16:48:22 | N/A | N/A |
| DC_bus_voltag | 直流侧电压 | number | 0 | V | 543 | 2018-07-25 16:48:22 | N/A | N/A |

通过查看“当前值”字段，可以获取设备各属性的实时数据信息。相关字段解释如下：

- **当前值**，指设备最后一次上报的该属性的值。如果没有上报则为空。
- **修改时间**，指设备最后一次更新该属性的值。
- **期望值**，指通过控制台或者应用程序性修改的desired值。如果没有。期望值则为空。
- **发送时间**，指该属性最后一次有desired值修改的时间
- **原始数据**，即设备影子的原始数据，用户可以通过原始数据查看设备状态或远程控制设备，如下图所示：

影子状态：

模型数据

原始数据

Shadow State:

```
1  {
2    "desired": {},
3    "reported": {
4      "FrequencyIn": 20,
5      "Current": 111,
6      "Speed": 1033,
7      "Torque": 41.5,
8      "Power": 31.9,
9      "DC_bus_voltage": 543,
10     "Output_voltage": 440,
11     "Drive_temp": 40
12   }
13 }
```

Metadata:

```
1  {
2    "reported": {
3      "FrequencyIn": 1532508502732,
4      "Current": 1532508502732,
5      "Speed": 1532508502732,
6      "Torque": 1532508502732
7    }
8 }
```

设备影子介绍

用户可通过设备影子查看实体设备的属性和状态等信息，如设备id、设备名称等。

设备影子是一个json文档，如下例所示：

```
"device": {

  "reported": {

    "light": "green"

  },

  "desired": {

    "light": "red"

  },

  "profileVersion": 10,

  "lastUpdatedTime": {
```

```

"reported": {

    "light": 1530016776000


},


"desired": {


    "light": 1530516776000


}

}

```

“device” 中的属性需要同步到设备实体。其中，“reported” 表示设备端通过MQTT连接汇报到影子的设备状态。“desired” 表示控制端希望控制设备变换到的目标状态。

远程控制属性

1. 点击“编辑物影子”，编辑属性的期望值。
2. 点击“提交”后，物影子将期望值下发至设备端。

| 属性名称 | 显示名称 | 类型 | 默认值 | 单位 | 当前值 | 修改时间 | 期望值 | 发送时间 |
|----------------|-------|--------|-----|-----|-----------------------------------|---------------------|---------------------------------|------|
| FrequencyIn | 输入频率 | number | 0 | Hz | <input type="text" value="20"/> | 2018-07-25 16:48:22 | <input type="text" value="10"/> | N/A |
| FrequencyOut | 输出频率 | number | 0 | Hz | <input type="text" value="输入"/> | N/A | <input type="text" value="输入"/> | N/A |
| Current | 电流 | number | 0 | A | <input type="text" value="111"/> | 2018-07-25 16:48:22 | <input type="text" value="输入"/> | N/A |
| Speed | 速度 | number | 0 | rpm | <input type="text" value="1033"/> | 2018-07-25 16:48:22 | <input type="text" value="输入"/> | N/A |
| Torque | 输出转矩 | number | 0 | % | <input type="text" value="41.5"/> | 2018-07-25 16:48:22 | <input type="text" value="输入"/> | N/A |
| Power | 输出功率 | number | 0 | KW | <input type="text" value="31.9"/> | 2018-07-25 16:48:22 | <input type="text" value="输入"/> | N/A |
| DC_bus_voltage | 直流侧电压 | number | 0 | V | <input type="text" value="543"/> | 2018-07-25 16:48:22 | <input type="text" value="输入"/> | N/A |
| Output_voltage | 输出电压 | number | 0 | V | <input type="text" value="440"/> | 2018-07-25 16:48:22 | <input type="text" value="输入"/> | N/A |
| Drive_temp | 变频器温度 | number | 0 | C | <input type="text" value="40"/> | 2018-07-25 16:48:22 | <input type="text" value="输入"/> | N/A |

用户也可以通过编辑原始数据desired中的属性实现设备的远程控制。

获取连接配置 创建物影子后，系统将自动生成与该物影子对应的连接配置，包括endpoint项目地址、设备名称和密钥信息。该信息将可用于将实体设备连接至物影子。具体操作如下：

1. 进入物详情界面。



The screenshot shows the 'Thing Shadow Details' page. At the top, there are two tabs: '物影子详情' and '物详情'. The '物详情' tab is selected and highlighted with a red box. Below the tabs, there are several configuration fields: '名称' (Name) set to 'floor5', '描述' (Description) set to '环境监测5', '来自模型' (From Model) set to 'floor5', and a '存储配置' (Storage Configuration) switch set to 'ON'. To the right of these fields are two buttons: '编辑' (Edit) and '连接配置' (Connection Configuration), with '连接配置' also highlighted with a red box. Below these fields is a table showing a single attribute: Temperature (环境温度) with type number, default value 0, unit Celsius (摄氏度), storage configuration '上报且满足条件存储 > 20', and last stored time 'N/A'. At the bottom left, it says '数据存储到: viztest'.

2. 点击“连接配置”。

注意:点击“更新密钥”后，原有密钥将失效，会导致已经接入的设备断开连接。



The screenshot shows a modal dialog box titled '连接配置' (Connection Configuration). Inside the dialog, there is an orange warning message: '① 更新密钥后会导致原有的连接断开，需要用新的密钥连接，请谨慎操作！' (Warning: Updating the key will cause the existing connection to disconnect, and new connections will be established using the new key. Please operate with caution!). Below the message, there are five connection parameters listed: 'TCP Address: tcp://Or8999a.mqtt.iot.bj.baidubce.com:1883', 'SSL Address: ssl://Or8999a.mqtt.iot.bj.baidubce.com:1884', 'WSS Address: wss://Or8999a.mqtt.iot.bj.baidubce.com:8884', 'name: Or8999a/_baidu_sample_pump_instance', and 'key: xxxxxxxxx'. To the right of the key field is a blue button labeled '更新密钥' (Update Key), which is highlighted with a red box. At the bottom right of the dialog is a blue '下载' (Download) button.

4.2.2 物影子操作

您可通过多种方式建立与物影子的连接。包括使用 Baidu IoT Edge SDK、标准 MQTT 客户端（如 Paho 项目）、HTTP Post（仅支持Pub）等。

说明

推荐用户优先使用百度云提供的SDK。

百度云提供的SDK封装了MQTT客户端SDK，屏蔽了MQTT客户端SDK的细节和topic信息，可以帮助用户实现业务的快速部署。

使用Baidu IoT Edge SDK 用户可以在设备端安装百度云提供的SDK，并配置[连接信息](#)，实现设备与百度云物接入的快速对接。

有关IoT Edge SDK的下载和安装，请查看：<https://github.com/baidu/iot-edge-sdk-for-iot-parser/releases/tag/v1.0.1>

使用开源MQTT客户端SDK 如果设备端已经调用了[Paho \(即MQTT Client SDK\)](#)，可以通过与特定的topic通信实现与百度云的对接。在物接入中定义了系统topic用于物接入服务和设备端基于物接入服务以及MQTT协议进行通信。

其中，clientID填写物影子名称。

更新设备状态到设备影子 将信息推送到主题' \$baidu/iot/shadow/{deviceName}/update'，可实现将设备状态更新到设备影子。

示例：

```
pub \$baidu/iot/shadow/myDeviceName/update
{
    "requestId": "{requestId}",
    "reported": {
        "memoryFree": "32MB",
        "light": "green"
    },
    "desired": {
        "rotate": 100
    },
    "profileVersion": 5,
    "lastUpdatedTime": {
        "reported": {
            "light": 1494904250
        },
        "desired": {
            "rotate": 1494904250
        }
    }
}
```

- “requestId” 为请求的唯一标识符，每一个请求的requestId是唯一的，可随机生成。
- reported为可选字段，代表物影子中设备上报的最新状态。服务端能通过MQTT或HTTP从reported字段中拿到物影子的最新状态。
- desired为可选字段，代表控制端期望设备变换到的目标状态。设备端通过MQTT从desired字段中拿到某个属性的期望值（如“light”：“red”），就收到了控制端期望执行的操作，硬件即可执行相关操作。硬件执行相关操作后，应该把对应的值上报到reported字段上。用户可以通过判断repoeted与desired的差别来判断是否反控成功。
- “profileVersion” 为可选字段，当未指定profileVersion时，物接入接收设备影子更新请求后，会将profileVersion自动加1；若指定profileVersion，物接入会检查请求中的profileVersion是否大于当前的profileVersion。只有在大于的情况下，物接入才会接受设备端的请求，更新设备影子，并将profileVersion更新到相应的版本。
- “lastUpdatedTime” 为可选字段，“lastUpdatedTime.reported” 和 “lastUpdatedTime.desired” 中的时间表示属性（“reported” 和 “desired”）的更新时间，如果没有相应字段，则更新时间由系统时间决定。注意只有当相应位置的属性键值对存在于本次请求中，且请求更新时间为非负整数（毫秒为单位），相应的时间更新有效，无效的更新时间会被替换为系统时间。此外，若本次请求中属性的更新时间为早于系统中该属性已存储的更新时间，则该属性的本次更新时间判断为过时，不予更新。

更新设备影子适用于两种应用场景：

1. 设备同步状态到物接入服务。设备在状态发生变化时，将实时的状态同步到物接入服务，包括状态的自动变化以及设备反控后状态的变化。更新设备状态，通常更新“reported”字段中的相关属性。对于反控后更新状态，设备可以用实时状态同时更新该属性的“reported”和“desired”中的值。
2. 通过MQTT协议反控设备状态。如果需要通过MQTT协议反控设备属性，可以通过更新“desired”字段实现。当物影子接收到“desired”相关属性的更新后，会diff设备影子中“reported”和“desired”相关字段，将diff后的结果发送到delta主题。设备端通过订阅delta主题，可将设备状态同步到“desired”的状态。状态反控后，更新设备影子，使“reported”和“desired”的值一致。物影子对设备的反控请参考通过设备影子控制设备状态。

订阅主题获取设备影子更新成功后的结果：

```
sub \$baidu/iot/shadow/myDeviceName/update/accepted
{
    "requestId": "{requestId}",
    "reported": {
        "firewareVersion": "1.0.0",
        "light": "green"
    },
}
```

```

  "desired": {
    "light": "red"
  },
  "lastUpdatedTime": {
    "reported": {
      "firewareVersion": 1494904250,
      "light": 1494904250
    },
    "desired": {
      "light": 1494904250
    }
  },
  "profileVersion": 10
}

```

订阅主题获取设备影子更新失败后的结果：

```

sub \$baidu/iot/shadow/myDeviceName/update/rejected
{
  "requestId": "{requestId}",
  "code": "{errorCode}",
  "message": "{errorMessage}"
}

```

从设备影子获取设备状态 发送请求到主题' \\$baidu/iot/shadow/{deviceName}/get'，可以获取该设备在设备影子中的所有状态信息。

示例：

```

pub \$baidu/iot/shadow/myDeviceName/get
{
  "requestId": "{requestId}"
}

```

订阅主题获取设备影子：

```

sub \$baidu/iot/shadow/myDeviceName/get/accepted
{
  "requestId": "{requestId}",
  "reported": {
    "firewareVersion": "1.0.0",
    ...
  }
}

```

```

    "light": "green"
},
"desired": {
    "light": "red"
},
"lastUpdatedTime": {
    "reported": {
        "firewareVersion": 1494904250,
        "light": 1494904250
    },
    "desired": {
        "light": 1494904250
    }
},
"profileVersion": 10
}

```

同时，可以订阅获取设备影子失败的相关消息：

```

sub \$baidu/iot/shadow/myDeviceName/get/rejected
{
    "requestId": "{requestId}",
    "code": "{errorCode}",
    "message": "{errorMessage}"
}

```

通过设备影子控制设备状态 控制端可以通过MQTT协议更新设备影子中的‘desired’字段，达到反控设备的目的。物影子在接受到‘desired’字段更新后，会比较‘reported’和‘desired’之间的差异，并将diff结果发送到主题‘\$baidu/iot/shadow/{deviceName}/delta’。

例如，当前‘reported’中的‘light’字段为green，控制端将‘desired’中的‘light’字段更新为‘red’，此时物影子会通过delta主题反控设备：

```

sub \$baidu/iot/shadow/myDeviceName/delta
{
    "requestId": "{requestId}",
    "desired": {
        "light": "red"
    }
}

```

若设备更新状态失败，可将相关错误信息发送到物影子：

```
pub \$baidu/iot/shadow/myDeviceName/delta/rejected
```

```
{  
    "requestId": "{requestId}",  
    "code": "{errorCode}",  
    "message": "{errorMessage}"  
}
```

清空设备影子 支持通过MQTT主题 ‘\$baidu/iot/shadow/{deviceName}/delete’ 清空设备影子。

示例：

```
pub \$baidu/iot/shadow/myDeviceName/delete  
{  
    "requestId": "{requestId}"  
}
```

通过订阅 ‘\$baidu/iot/shadow/{deviceName}/delete/accepted’ 可以获取设备影子清空成功后的response。

```
sub \$baidu/iot/shadow/myDeviceName/delete/accepted  
{  
    "requestId": "{requestId}",  
    "reported": {  
        "firewareVersion": "1.0.0",  
        "light": "green"  
    },  
    "desired": {  
        "light": "red"  
    },  
    "lastUpdatedTime": {  
        "reported": {  
            "firewareVersion": 1494904250,  
            "light": 1494904250  
        },  
        "desired": {  
            "light": 1494904250  
        }  
    },  
    "profileVersion": 10  
}
```

主题’\$baidu/iot/shadow/{deviceName}/delete/rejected’ 推送清空设备影子失败后的相关信息

```
sub \$baidu/iot/shadow/myDeviceName/delete/rejected
{
    "requestId": "{requestId}",
    "code": "{errorCode}",
    "message": "{errorMessage}"
}
```

订阅设备影子的变化 可以通过主题“\$baidu/iot/shadow/{deviceName}/update/documents” 订阅设备影子中reported字段内容的变化。物接入在收到设备影子的update请求、并成功更新后，如果reported字段内容有变（变化条件包括：增加属性、减少属性、属性值有变化），会把reported字段中更新属性的当前值和更新前的值发送到“documents”主题。

示例：

```
sub \$baidu/iot/shadow/{deviceName}/update/documents
{
    "requestId": "{requestId}",
    "profileVersion": 10,
    "current": {
        "light": "green"
    },
    "previous": {
        "light": "red"
    }
}
```

订阅设备快照 documents topic只反映了reported字段内容中发生变化的属性状态，如在reported字段内容变化时，需要订阅设备的全量的属性状态，可以通过主题“\$baidu/iot/shadow/{deviceName}/update/snapshot” 获取。该主题内容会包括shadow的reported字段的全部属性，此外还包含相应的lastUpdatedTime、profileVersion字段内容。

示例：

```
sub \$baidu/iot/shadow/{deviceName}/update/snapshot
{
    "requestId": "{requestId}",
    "profileVersion": 10,
```

```
"reported": {  
    "light": "green"  
},  
"lastUpdatedTime": {  
    "reported": {  
        "light": 1494904250  
    }  
}  
}
```

Device Profile Device Profile由Device Registry和Device Shadow两部分组成。

```
{  
    "name": "test", //设备名称  
    "id": "098f6bcd4621d373cade4e832627b4f6", //设备ID  
    "description": "测试设备", //设备描述  
    "state": "online", //设备状态, online/offline/unknown  
    "templateId": "123456", //设备模板ID  
    "templatedName": "TestTemplate", //设备模板名称  
    "createTime": 1494904250, //创建时间  
    "lastActiveTime": 149490300, //最后一次设备影子(reported)更新时间  
    "attributes": {  
        "region": "Shanghai" //设备Tag  
    },  
    "device": { //设备影子  
        "reported": {  
            "firewareVersion": "1.0.0",  
            "light": "green"  
        },  
        "desired": {  
            "light": "red"  
        },  
        "lastUpdatedTime": {  
            "reported": {  
                "firewareVersion": 1494904250,  
                "light": 1494904250  
            },  
            "desired": {  
                "light": 1494904250  
            }  
        },  
        "profileVersion": 10  
    }  
}
```

4.2.3 权限管理

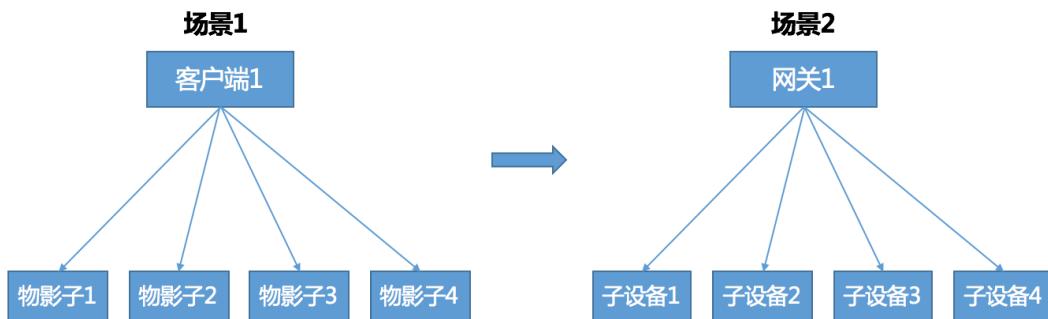
介绍 在物接入中设置权限管理，可以使得一个客户端的连接配置具有访问多个设备的能力。

物接入设备型权限分为普通权限和超级权限。

- 普通权限:由用户自行选择添加物影子，上限100个。
- 超级权限:默认包含用户名下所有的物影子。

如图所示：

1. 给客户端1新建权限1，权限1内包括物影子1、2、3、4；
2. 权限1的用户名与密钥可以向物影子1、2、3、4的topic发送消息；
3. 根据权限1起一个连接就可以向物影子1、2、3、4对应的设备通信。



注意：

1. 用户通过服务端连接物影子，订阅所有/部分设备的物影子变化状态，可以用一个mqtt客户端订阅，不需要启用n个客户端。
2. 网关设备连接n个子设备，需要有一个MQTT客户端可以对网关和子设备都有访问权限，如更新shadow/获取delta等。设置权限管理后，则可避免一个网关启用n个连接来更新n个子设备的物影子。
3. 一个设备最多允许属于10个权限组。

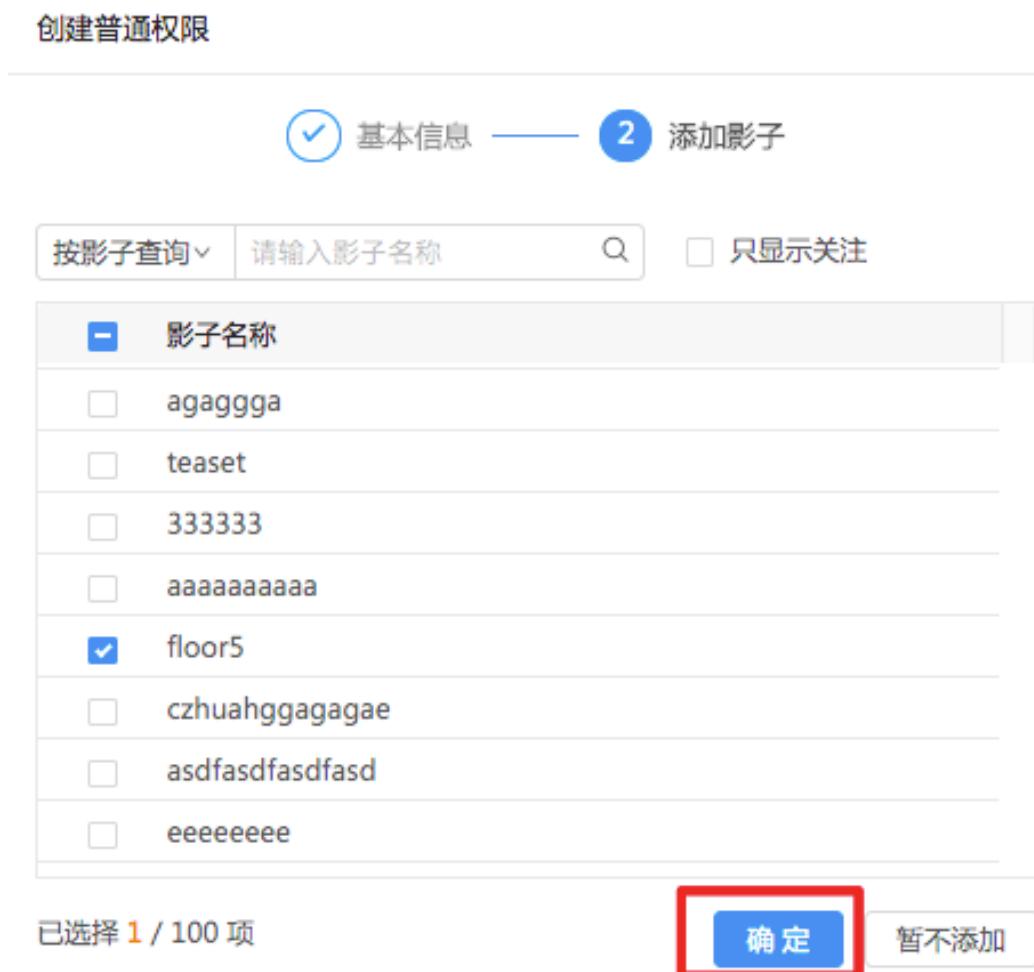
创建权限

1. 选择“产品服务>物联网服务>物接入 IoT Hub”，进入“天工物联网”控制台。
2. 点击设备型项目的项目名称进入详情页，选择“权限组”，接下来进行权限管理设置。

3. 点击“创建权限”，填写基本信息，分别为名称和描述。



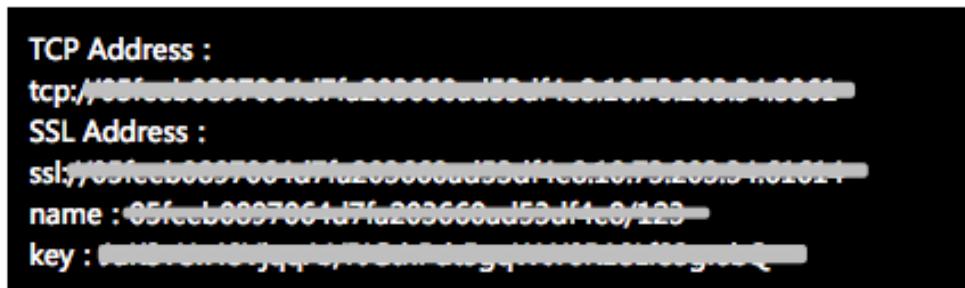
4. 选择物影子，支持多选，目前上限100个。



5. 点击“确定”，完成权限设置，并生成配置信息。

提示

✓ 创建成功！请将连接信息配置到SDK中，实现设备与云端连接。 [如何连接](#)
 为了安全考虑，请合理保管以上密钥，密钥丢失无法找回，只能在物详情中重新生成。

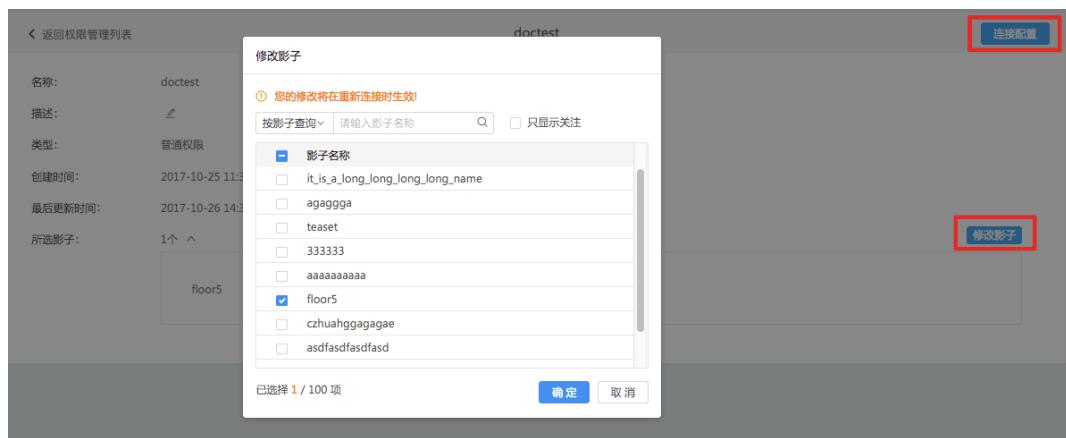


[关闭](#)

[下载](#)

6. 点击权限名称，进入权限管理列表。

- 可以编辑权限描述；
- 点击“连接配置”，可以查看配置信息。
- 点击“修改影子”，可以重新添加物影子。



创建超级权限

1. 选择“产品服务>物联网服务>物接入 IoT Hub”，进入“天工物联网”控制台。
2. 点击设备型项目的项目名称进入详情页，选择“权限组”，接下来进行权限管理设置。
3. 点击“创建超级权限”。



4. 超级权限具有访问所有设备的权限，请妥善保管好用户名和秘钥，谨慎使用。

- 超级权限不支持权限内物影子的增加和减少。
- 在创建超级权限后，新创建的影子会被默认添加到超级权限内。

| 权限名称 | 描述 | 类型 | 影子总数 | 操作 |
|----------|----|------|------|--------------------|
| doctest1 | | 超级权限 | 42 | 删除 |
| doctest | | 普通权限 | 1 | 删除 |

5. 点击权限名称，进入权限管理列表，可以编辑权限描述；点击“连接配置”，可以查看配置信息。



4.2.4 设备在线状态

注意

使用IoT Edge SDK,当使用SDK向云端发送过一次数据后，物影子即标志为『在线』。如果设备由于网络原因或自身关机等原因断开连接，物影子即显示离线，可以通过API获取。

使用MQTT开源SDK，如果使用MQTT开源SDK，请将MQTT客户端的clientid设置为物影子名称，物接入则可以监测在线和离线状态。

1. 在物影子列表中，我们可以看到物影子设备初始默认处于离线状态

The screenshot shows a list of device shadows in the Baidu Cloud Device Shadow List. There are three entries:

- Shadow_test_1**: 离线 (Offline). Description: 说明: 属性总计: 2个 memoryFree : 32... light : green
- baidu_sample_pu...**: 离线 (Offline). Description: 说明: 示例: 一个实体水泵, 请在物详情页面获取... 属性总计: 11个 FrequencyIn : 400 Current : 111 Speed : 1033
- f5real**: 离线 (Offline). Description: 说明: 环境监测5 属性总计: 0个

2. 使用IoT Edge SDK, 当使用SDK向云端发送过一次数据后, 物影子即标志为『在线』。如果设备由于网络原因或自身关机等原因断开连接, 物影子即显示离线, 可以通过API获取。
3. 使用MQTT开源SDK, 如果使用MQTT开源SDK, 请将MQTT客户端的clientid设置为物影子名称, 物接入则可以监测在线和离线状态。

4.2.5 设备间相互通信

设备可以通过更新物影子来上传当前状态, 在有些场景下, 设备还需要相互通信。物接入默认给每个设备绑定有主题\$baidu/iot/general/#的订阅和发布权限。

例如, 设备A发布消息到主题\$baidu/iot/general/a, 设备B订阅主题\$baidu/iot/general/a, 则设备B就能收到设备A的消息。

4.2.6 OTA 服务

介绍 百度云天工OTA服务平台是致力为物联网智能设备系统安全保驾护航的固件升级服务。我们为广大设备厂商提供稳定、快速、易用、高效的固件下发服务, 以保证在洞察到设备系统安全隐患后能迅速响应, 降低潜在损失。

除常规的下发服务外, 物接入OTA服务将逐步整合百度自主研发, 业内领先的智能设备硬件安全检测与修复能力, 为所覆盖的智能设备群打造安全护城河, 是设备厂商最佳的智能设备系统升级和漏洞安全修复解决方案。

云端OTA操作

1. 账号体系 物接入OTA作为百度云天工物接入产品的功能, 沿用百度账号体系, 如何注册和登录详见物接入产品文档。

2. 使用前准备 在创建物模型时需要选择是否开启OTA远程升级，只有开启服务后，该物模型下面的设备才可以进行固件升级。在创建时未开启服务的物模型，也可以在物模型编辑页面开启服务。



开启OTA远程升级服务后，用户需要选择下发协议。

- HTTPS协议：目前只有Android和Linux操作系统的设备可以选择https。
- 对于Android和Linux操作系统的设备的升级任务，在设备端我们提供刷写固件、重启等功能。选择了https协议之后，您还需要填写设备的操作系统和芯片类型。

注意: 设备的操作系统和芯片类型,这两个字段在填写后将无法修改，请慎重填写。

- 对于Linux操作系统的设备，您可以提工单把交叉编译的环境和参数给我们，我们将针对您的设备编译相应的SDK，详情请查看[跨平台OTA SDK接入文档](#)。
- MQTT协议：对所有操作系统的设备都适用。设备可直接集成SDK，具体的SDK配置方法请阅读[设备SDK集成指南](#)。

3. 上传升级包 OTA升级服务的入口在物设备型项目详情页中，左侧选择 OTA，主界面如下图所示：



具体操作方法如下：

- 登录[百度云官网](#)，点击右上角的“管理控制台”，快速进入控制台界面。
- 选择“产品服务>物联网服务>物接入 IoT Hub”，进入“天工物联网”控制台。

- c. 点击设备型项目的项目名称，选择选择“OTA远程升级”，进入OTA页面；点击“添加升级包”进入添加升级包页面。

The screenshot shows a form titled 'Add Upgrade Package'. It includes fields for 'Device Model' (a dropdown menu), 'Select Upgrade Package' (a file input field with a 'Upload File' button and a note about size limit), 'Version Number' (a text input field with a note about naming conventions), and 'Version Description' (a text input field with a character count limit). At the bottom are 'Confirm' and 'Cancel' buttons.

按照要求填写升级包创建信息，各字段的填写规则如下表：

| 字段 | 填写方式 | 说明 | 备注 |
|-------|------|--------------------------------|----|
| 物模型 | 下拉框 | / | 必填 |
| 选择升级包 | 选择文件 | 文件大小不超过50M | 必选 |
| 版本号 | 填空 | 为了便于管理，版本号有严格的形式，只能以x.x.x的形式命名 | 必填 |
| 版本描述 | 下拉框 | 可填写128字以内的版本备注 | 选填 |

完成填写后，点击“确定”即可上传升级包。

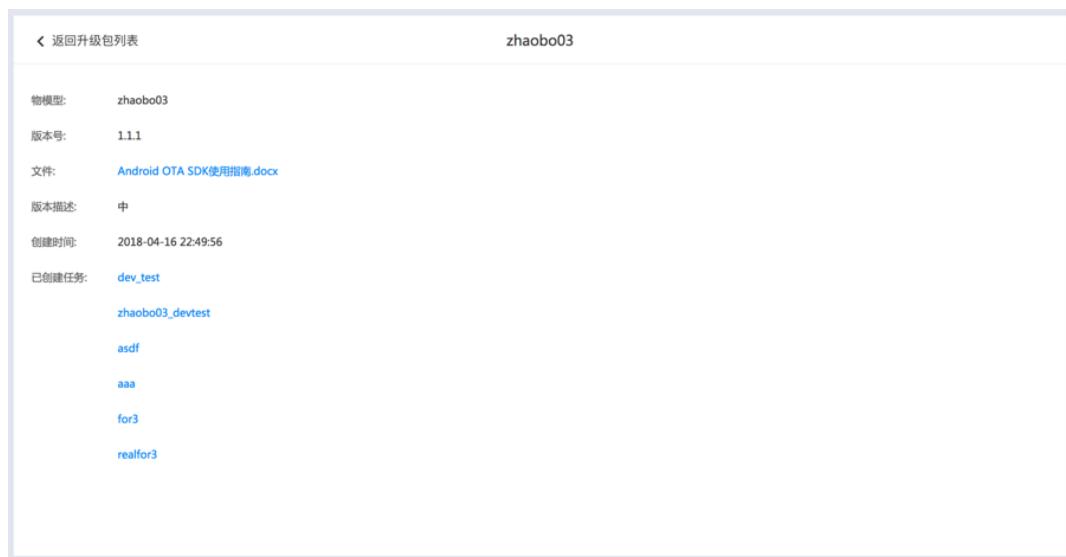
The screenshot shows a list of upgrade packages. The table columns are: 物模型名称, 版本号, 版本说明, 添加时间, and 操作. The data rows are:

| 物模型名称 | 版本号 | 版本说明 | 添加时间 | 操作 |
|----------|-------|------|---------------------|----|
| zhaobo03 | 3.0.0 | | 2018-04-17 15:50:23 | 升级 |
| zhaobo03 | 1.0.2 | ddd | 2018-04-17 15:29:02 | 升级 |
| zhaobo03 | 1.1.1 | 中 | 2018-04-16 22:49:56 | 升级 |

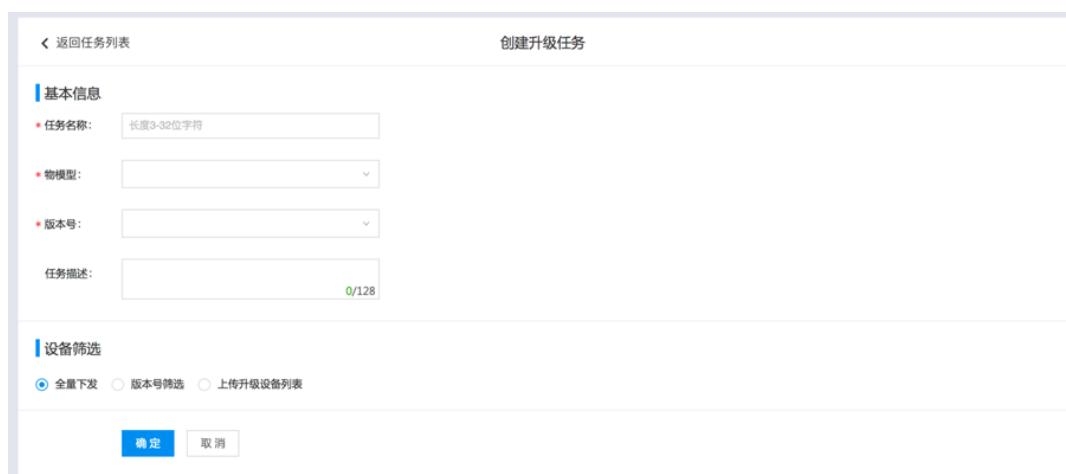
上传成功后能在升级包列表页看到新创建的升级包。点击版本号可以查看升级包详情。点击物模型会跳到物模型详情页。点击“升级”可以用该升级包创建升级任务。

在升级包的详情页可以看到该升级包上传的时间、已用该升级包创建的升级任务等信息。点

点击具体的任务可以跳转到任务详情页。



4. 创建升级任务 在升级包列表页或者升级任务页均可以创建升级任务，进入任务信息填写：



| 字段 | 填写方式 | 说明 | 备注 |
|------|------|------------------|----|
| 任务名称 | 填空 | 3-32位字符，支持中英文和数字 | 必填 |
| 物模型 | 下拉框 | 选择升级哪个物模型下的设备 | 必选 |
| 版本号 | 下拉框 | 选择升级包的版本号 | 必填 |
| 任务描述 | 填空 | 128个字符以内 | 可选 |

| 字段 | 填写方式 | 说明 | 备注 |
|------|------|--|----|
| 设备筛选 | 单选框 | 全量下发、版本号筛选和上传升级设备列表三种。全量下发：升级该物模型下的所有设备；版本号筛选：可以选择特定版本号的设备进行升级，支持多选；上传升级设备列表：用户可上传一个.txt文件，文件中罗列需要升级的设备的ID | 必选 |

完成填写后，点击“确定”完成创建。

5. 追踪升级任务进度 在升级任务详情页可以追踪任务进度，可以看到升级耗时、升级设备总数、升级成功数和升级失败数。在设备信息栏有每台设备的任务进度。

| 物影子： | 物影子 | 升级开始时间 | 升级结束时间 | 升级前版本号 | 结果 |
|------|-------------------|---------------------|---------------------|--------|-----|
| | zhaobo03_devtest2 | 2018-04-17 15:39:41 | N/A | 0.1.0 | 升级中 |
| | zhaobo03_devtest | 2018-04-17 15:38:35 | 2018-04-17 15:38:35 | 0.1.0 | 成功 |
| | zhaobo03_devtest3 | 2018-04-17 15:39:45 | 2018-04-17 15:45:15 | 0.1.0 | 失败 |

设备端使用MQTT OTA

MQTT OTA的功能

- 云端向设备推送固件升级信息，包括固件包的URL

- 设备向云端上报固件升级的结果

用户需要适配自己的设备，实现固件刷写、重启等功能。

说明

推荐用户优先使用百度云提供的SDK。

百度云提供的SDK封装了MQTT客户端SDK，屏蔽了MQTT客户端SDK的细节和topic信息，可以帮助用户实现业务的快速部署。

用户可以在设备端安装百度云提供的SDK，并配置连接信息，实现设备与百度云物接入的快速对接。

有关IoT Edge SDK的下载和安装，请查看：<https://github.com/baidu/iot-edge-c-sdk>

通过MQTT客户端使用MQTT OTA 有些情况下，设备<—>云的交互需要立刻返回某种结果，或者立刻给予确认。比如设备向云查询最新的固件版本、云请求设备端开始固件升级。

除了用shadow做交互，物接入还使用了一种直接的Method调用。Method调用代表着一种请求-响应式的交互，与HTTP调用很类似。它支持设备->云和云->设备这两种请求方向。

Method的MQTT主题

设备请求物接入 (cloud method)

```
{\color{emcolor}\textbf{\$baidu/iot/shadow/{deviceName}/method/cloud/req}} // 设备向该主题发布请求，物接入订阅
```

```
{\color{emcolor}\textbf{\$baidu/iot/shadow/{deviceName}/method/cloud/resp}} // 物接入向该主题发布响应，设备订阅
```

物接入请求设备 (device method)

```
{\color{emcolor}\textbf{\$baidu/iot/shadow/{deviceName}/method/device/req}} // 物接入向该主题发布请求，设备订阅
```

```
{\color{emcolor}\textbf{\$baidu/iot/shadow/{deviceName}/method/device/resp}} // 设备向该主题发布响应，物接入订阅
```

作为设备端，在建立MQTT连接后，应当即刻订阅上面物接入向该主题发布响应，设备订阅和物接入向该主题发布请求，设备订阅这2个主题。

METHOD 的MQTT载荷

载荷是以Json序列化的字符串。请求和响应的格式分别如下：

请求参数

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|------------|--------|------|-----------|
| methodName | String | 必选 | Method名字 |
| requestId | String | 必选 | 请求ID |
| payload | Object | 可选 | Method的参数 |

请求示例

```
{
  "methodName": "updateFirmware",
  "requestId": "993ff7e9-018b-4246-a7ba-5ddac970054",
  "payload": {
    "someField": "someOutput"
  }
}
```

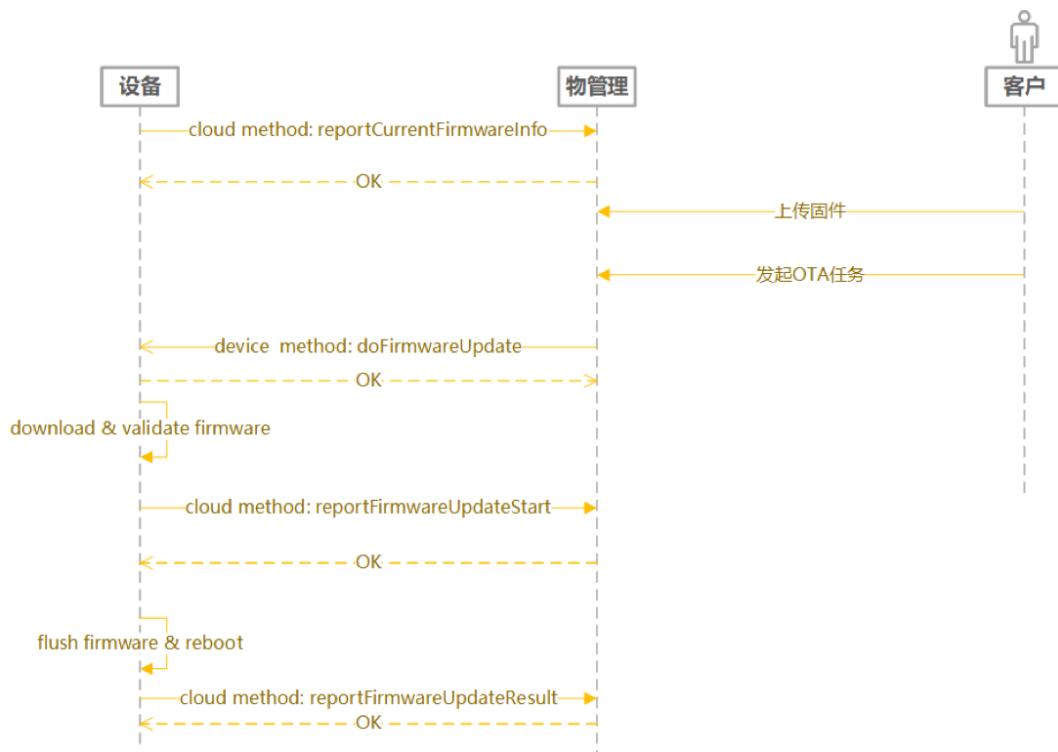
响应参数

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|------------|--------|------|------------------|
| methodName | String | 必选 | Method名字 |
| requestId | String | 必选 | 请求ID |
| status | int | 必选 | 状态码, 200-299表示正常 |
| payload | Object | 可选 | Method的结果 |

响应示例

```
{
  "methodName": "updateFirmware",
  "requestId": "993ff7e9-018b-4246-a7ba-5ddac970054",
  "status": 200,
  "payload": {
    "someField": "someOutput"
  }
}
```

设备与物接入交互过程简介



上报设备版本 (REPORTCURRENTFIRMWAREINFO)

通过物接入OTA进行过固件升级的设备，物接入知晓它的固件版本。而对于那些从未升级过的设备，应当至少使用它一次，以让物接入知晓它的版本。

请求示例

```

pub \$baidu/iot/shadow/{deviceName}/method/cloud/req

{
    "requestId": "12345",
    "methodName": "reportCurrentFirmwareInfo",
    "payload": {
        "firmwareVersion": "1.1.1"
    }
}
  
```

返回如下结果：

```

rcv \$baidu/iot/shadow/{deviceName}/method/cloud/resp
{
    "requestId": "12345",
    "methodName": "reportCurrentFirmwareInfo",
  
```

```
"status": 204,  
}
```

请求参数

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|-----------------|--------|------|------|
| firmwareVersion | String | 必选 | 当前版本 |

返回参数

无

上报设备OTA开始事件 (REPORTFIRMWAREUPDATESTART)

在真正开始固件刷写之前，设备应当向物接入汇报固件升级开始，作为整个固件升级流程跟踪的一部分。

请求示例

```
pub \$baidu/iot/shadow/{deviceName}/method/cloud/req  
{  
    "requestId": "12345",  
    "methodName": "reportFirmwareUpdateStart",  
    "payload": {  
        "jobId": "9476da26-8cf3-497d-9959-c2017a248901"  
    }  
}
```

返回示例

```
recv \$baidu/iot/shadow/{deviceName}/method/cloud/resp  
{  
    "requestId": "12345",  
    "methodName": "reportFirmwareUpdateStart",  
    "status": 204  
}
```

请求参数

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|-------|--------|------|------|
| jobId | String | 必选 | 任务ID |

上报设备OTA结果 (REPORTFIRMWAREUPDATERESULT)

在固件升级完成之后，设备向物接入汇报升级结果。Result字段可以是“success”表示成功，或者“failure”表示失败。

请求示例

```
pub \$baidu/iot/shadow/{deviceName}/method/cloud/req
{
    "requestId": "12345",
    "methodName": "reportFirmwareUpdateResult",
    "payload": {
        "result": "success",
        "jobId": "9476da26-8cf3-497d-9959-c2017a248901"
    }
}
```

返回示例

```
recv \$baidu/iot/shadow/{deviceName}/method/cloud/resp
{
    "requestId": "12345",
    "methodName": "reportFirmwareUpdateResult",
    "status": 204
}
```

请求参数

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|-----------------|--------|------|------|
| jobId | String | 必选 | 任务ID |
| firmwareUrl | String | 必选 | 名称 |
| firmwareVersion | String | 必选 | 描述 |

跨平台OTA SDK (HTTPS) 接入 跨平台 OTA 特指控制台中 HTTPS 方式所支持的 Android、Linux 系统 OTA。

术语介绍 OTA完整包：OTA包中包含了目标版本中所有文件的完整内容

OTA差分包：OTA包使用差分算法生成，其中对于目标版本和源版本中有变化的同一文件，差分包中只包含该文件的差分patch文件 (.p文件)

说明：

- 1、特殊情况下差分包中不存在.p文件，此时表示目标版本和源版本中没有相同文件（同名、同路径）
- 2、完整包中只包含目标版本信息，差分包中包含源版本和目标版本信息，只有原版本与当前系统版本相同时才能执行升级

文件分段：将单个大文件按照一定的大小切分成多个小文件的过程，切分成的小文件称为分段文件

说明：

- 1、当待升级设备上剩余空间很小，不允许存放过大的完整升级文件时，可对大文件分段，然后依次升级每个分段文件
- 2、升级包制作时，对文件分段后，生成的每个分段文件对于整个升级包来说是一个独立文件，执行分片操作后，每个分段文件会单独生成一个分片OTA升级包

OTA升级包分片：使用ota生成工具的div命令，将完整OTA升级包切分成若干个分片OTA升级包的过程

完整的OTA升级包：OTA升级包中包含了本次升级中的所有升级文件及元信息

说明

- 1、完整的OTA升级包是以.bdota为后缀的升级文件
- 2、完整的OTA升级包可以是“OTA完整包”也可以是“OTA差分包”
- 3、完整的OTA升级包中的升级文件可以分段，也可以不分段

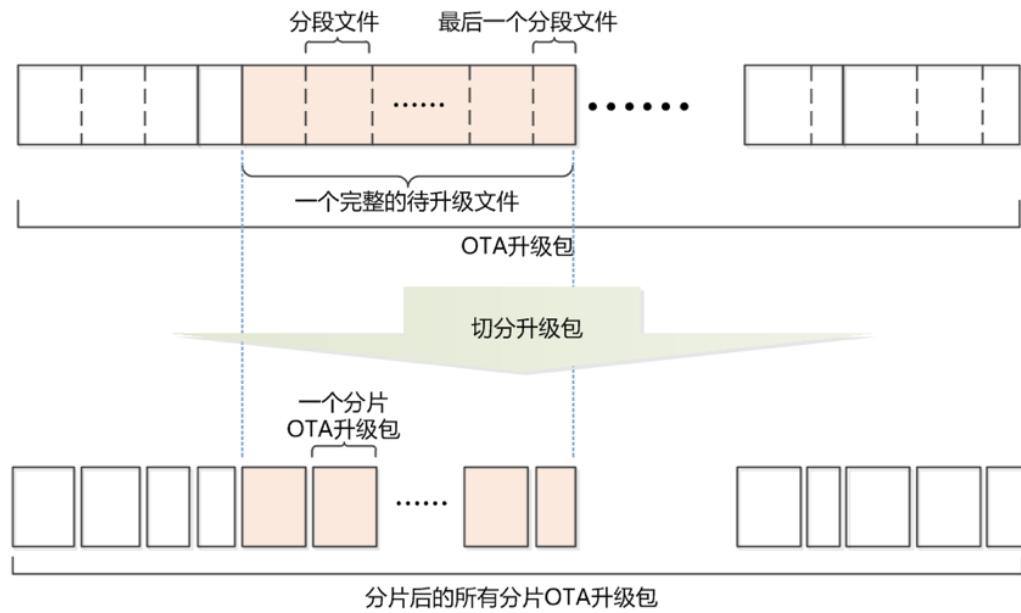
分片OTA升级包：完整OTA包经过OTA升级包分片操作后生成，一个分片OTA升级包中只包含一个文件

说明

分片OTA升级包中包含的文件可以是单个完整文件，也可以是单个差分文件 (.p文件)，也可以是一个分段文件

注意：以下OTA包结构图均为示意图，其中不包括文件的元信息和整个OTA包的头部信息。

OTA升级包分片操作过程，及其前后的OTA升级包结构示意图如下：



OTA SDK接入流程

1. 获取SDK的方式

对于Android和Linux操作系统的设备的升级任务，在设备端我们提供刷写固件、重启等功能。相关SDK目前还未开源。您需要[提工单](#)把交叉编译的环境和参数给我们，我们将针对您的设备编译相应的SDK。

2. 在物接入中创建物模型

如下图所示，在物模型的创建页面选择下发协议为https协议，并填写好操作系统、芯片类型和相关的属性，点击“创建”即可创建物模型。

The screenshot shows the 'Create Device Model' interface. It includes fields for:

- 名称:** test000518
- 描述:** 0-128个字符
- OTA远程升级:** ON
- 下发协议:** HTTPS MQTT
- 对于Android和Linux操作系统的设备的升级任务，在设备端我们提供刷写固件、重启等功能。这类升级任务会通过https下发升级包。详情请参考 [OTA操作文档](#)**
- 操作系统:** Linux Android
- 芯片类型:** testchip

A note at the bottom says: **请慎重选择，操作系统和芯片类型填写后无法修改！**

图 4.1: 片

在物模型的详情页可以得到“OTA ID”和“OTA Secret”，如下图所示：

The screenshot shows the 'Device Model Details' page for device model 'test000518'. It displays the following information:

- 名称:** test000518
- 描述:**
- 修改时间:** 2018-05-18 20:49:35
- 属性:**

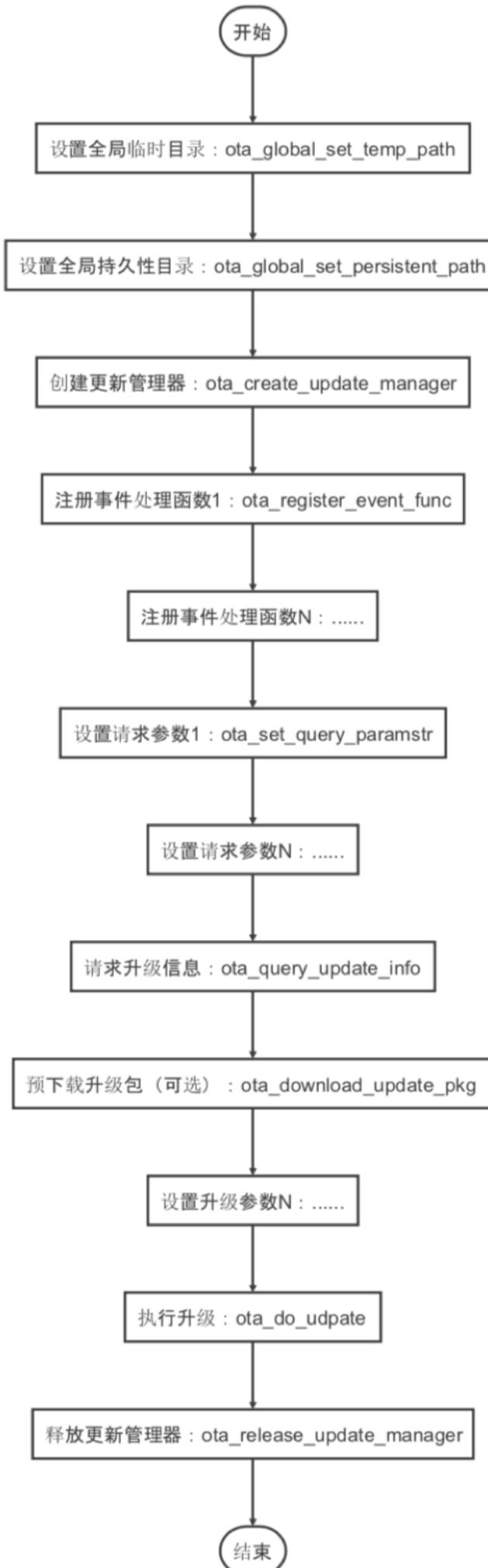
| | 属性名称 | 显示名称 | 类型 | 默认值 | 单位 |
|--|------|-------------|--------|-----|----|
| | temp | temperature | number | | |
- OTA远程升级:** 已开通服务
- 下发协议:** HTTPS
- 操作系统:** LINUX
- 芯片类型:** testchip
- OTA ID:** 1128
- OTA Secret:** SUyeX[REDACTED]QLCQxXFAA==

图 4.2: 片

OTA ID是当前项目的唯一标识，在`ota_set_query_paramstr` API中对应`QUERY_PARAM_PID`字段 OTA Secret是用于SDK对数据加密的密钥，在`ota_set_query_paramstr` API中对应`QUERY_PARAM_SECRET`字段

3. 代码中接入SDK API

根据实际需求，依此调用相应API，网络升级调用流程如下：



OTA SDK接口说明 `include/bdota` 目录下的头文件包含的接口均可使用，其中，与升级任务相关的SDK接口定义于 `include/bdota/bdota.h` 头文件，其中主要接口介绍如下：

`ota\global\set\temp\path`

- 函数定义

```
int ota_global_set_temp_path(const char* path);
```

- 功能

设置临时目录，该目录内容重启后可能丢失，被用作存储会被删除的临时文件。

`ota\global\set\persistent\path`

- 函数定义

```
int ota_global_set_persistent_path(const char* path);
```

- 功能

设置持久性目录，改目录内容重启后不会丢失，被用作存储预下载升级包、上报缓存文件等。

`**ota\create\update_manager**`

- 函数定义

```
pota_update_manager ota_create_update_manager();
```

- 功能

创建更新管理器对象指针 `pota_update_manager`，该指针用于后续系列函数的调用。

`**ota\register\event_func**`

- 函数定义

```
int ota_register_event_func (
    pota_update_manager manger,
    const char* event_name, // 事件名称
    func_event_callback event_func, // 事件发生时的回调函数
    void* user_data // 透传给event_func的参数
);
```

- 功能

为更新管理器设置回调函数

- 参数说明

manger : 更新管理器对象；

event_name : 事件名称，详细说明请参见[func\event\callback回调事件](#)部分；

event_func : 注册的事件被触发时，该回调函数会被调用；

user\data : 用户自定义数据，该参数会作为参数透传给event\func回调函数；

- 返回值说明

返回0表示成功，非0表示失败。

[func\event\callback](#)

- 函数定义

```
typedef int(*func_event_callback)(  
    void* user_data, // 透传参数，注册回调时指定的参数  
    pbdstring* params, // 事件的参数列表指针  
    size_t size, // 事件参数的个数  
    pbdstring* out); // 传出参数  
>);
```

- 功能

[ota_register_event_func](#)函数中的回调函数接口定义

- 参数说明

user\data : 用户自定义数据，即ota\register\event\func函数调用时的user_data参数，会通过该参数透传回来；

params : 事件的参数内容，不同事件类型对应不同类型的参数内容；

size : 事件的参数内容的数量；

out : 传出参数，预留位，传null即可；

- 返回值说明

返回OTAR_OK升级继续，其它将终止升级

- 回调事件

支持的事件名称使用宏定义，已定义的事件宏包括：

```
#define OTA_UPDATE_EVENT_START "start"
#define OTA_UPDATE_EVENT_END "end"
#define OTA_UPDATE_EVENT_PROCESS_FILE "process_file"
#define OTA_UPDATE_EVENT_DELETE_FILE "delete_file"
#define OTA_UPDATE_EVENT_GET_LOCAL_FILE_PATH "get_local_file_path"
#define OTA_UPDATE_EVENT_SHOW_PROGRESS "show_progress"
```

不同回调事件，对应的回调函数参数含义不同，具体各事件对应的参数含义说明如下：

| 事件名称 | 事件含义 | 触发时机 | params参数含义 | out参数含义 |
|--|------|---|--|---------|
| OTA \UPDATE \EVENT \START 升级开始 SDK 解析升级包，第一次遇到全局头时将调用该函数，如果没有注册，则不调用 无 无 OTA \UPDATE \EVENT\END | 升级结束 | 升级包的全部文件都已经处理完毕后调用，业务放可以在该回调中修改写入设备的启动信息。例如对于AB系统，则要在该回调中完成切换启动主分区的操作 | 参数一：“0”：使用的是分段升级，当前段处理完成，但还有后续段需要处理。“1”：升级包中的所有文件已经处理完成。 | 无 |

| 事件名称 | 事件含义 | 触发时机 | params参数含义 | out参数含义 |
|--|-------------------------|--|--|--------------|
| OTA \UPDATE \EVENT \PROCESS \FILE | 处理文件 | 解析升级包时，当一个文件已经在本地准备好时（该文件可能来之网络、本地升级包、本地分片升级包）调用 | 参数一：升级包中指定的当前文件的路径。参数二：偏移，当前文件内容在原始文件中的偏移。参数三：文件内容在本地硬盘上的路径，如果启用了分段功能，该内容可能为原始文件的一部分。参数四：当前文件内容大小。参数五：总文件大小。 | 无 |
| OTA \UPDATE \EVENT \DELETE\FILE | 删除文件或文件夹 | 当需要删除文件或文件夹时，调用该函数 | 参数一：升级包中指定的待删除的文件路径。参数二：路径标识的是文件还是目录，文件用字符串0表示，目录用字符串1表示。 | 无 |
| OTA \UPDATE \EVENT \GET \LOCAL \FILE \PATH | 返回本地文件系统中对应与升级包中的文件的全路径 | 升级过程中任何需要获取本地路径完整路径的时候均可回调次函数 | 参数一：升级包中的文件路径 | 本地系统中的完整文件路径 |
| OTA \UPDATE \EVENT \SHOW \PROGRESS | 显示进度 | 当前版本暂未触发该事件，为后续版本保留 | 无 | 无 |

ota\set\query_paramstr

- 函数定义

```
int ota_set_query_paramstr(pota_update_manager manager, const char* key, const char* value);
```

- 功能

设置升级请求参数

- 参数说明

manager：更新管理器对象；

key：升级请求字段名称；

value：升级请求字段内容。

- 返回值说明

返回0表示成功，非0表示失败。

- 已支持字段

```
#define QUERY_PARAM_PID "pid" // 产品ID  
#define QUERY_PARAM_SECRET "acckey" // 产品密钥  
#define QUERY_PARAM_DID "did" // 设备唯一标识符  
#define QUERY_PARAM_VER "ver" // 系统版本号，格式必须为a.b.c.d  
#define QUERY_PARAM_MANUAL_CHECK "mchk" // 请求类型，value设置字符串1表示手动升级，其它为自动升级
```

ota\query\update_info

- 函数定义

```
int ota_query_update_info(pota_update_manager manager);
```

- 功能

执行升级请求

- 参数说明

manager：更新管理器对象；

- 返回值说明

返回0表示成功，非0表示失败。

****ota\download\update_pkg****

- 函数定义

```
int ota_download_update_pkg(pota_update_manager manager);
```

- 功能

执行预下载，此函数需要在ota_query_update_info函数调用以后被调用

- 参数说明

manger：更新管理器对象；

- 返回值说明

返回0表示成功，非0表示失败。

****ota\set\update_param****

- 函数定义

```
int ota_set_update_param(pota_update_manager manager, const char* key, const char* value);
```

- 功能

设置更新程序参数

- 参数说明

manger：更新管理器对象；

key：参数字段名称；

value：参数字段内容。

- 返回值说明

返回0表示成功，非0表示失败。

- 已支持字段

```

// 设置了该key，将执行本地升级，该key的value指明了本地包的文件路径
#define OTA_UPDATE_PARAM_KEY_LOCAL_FILE "local_file"
// 指定本地分段升级时当前处理的分片文件路径
#define OTA_UPDATE_PARAM_KEY_SEG_FILE "seg_file"
// 设置用于校验升级包的公钥路径
#define OTA_UPDATE_PARAM_KEY_PUBLIC_KEY_PATH "pk"
// 指定本地升级信息存取的文件，如果指定从本地查询升级，则读取该文件
// 如果指定从网络查询升级，则存在升级时保存升级信息到该文件
#define OTA_UPDATE_PARAM_KEY_LOCAL_UPDATE_INFO_PATH "localupdate_info_path"
// 设置服务地址，一般不用使用
#define OTA_UPDATE_PARAM_KEY_SERVICE_URL "service_url"
// 指示升级请求是从服务器问询或者是从本地文件读取
#define OTA_UPDATE_PARAM_KEY_SERVER_TYPE "server_type"
// 升级请求从文件中读取，文件路径通过ota_set_update_param设置
#define OTA_UPDATE_PARAM_KEY_SERVER_TYPE_LOCAL "local"
// 升级请求从网络读取，服务器地址通过ota_set_update_param设置
#define OTA_UPDATE_PARAM_KEY_SERVER_TYPE_NET "net"

```

[ota\do\udpdate](#)

- 函数定义

```
int ota\_do\_update\(pota\_update\_manager manager, uint32\_t flag\);
```

- 功能

执行实际升级功能，调用此函数前需要调用ota_set_update_param函数设置更新参数

- 参数说明

manager：更新管理器对象；

- 返回值说明

返回0表示成功，非0表示失败。

[ota_release_update_manager](#)

- 函数定义

```
int ota\_release\_update\_manager\(pota\_update\_manager manager\);
```

- 功能

释放更新管理器对象

- 参数说明

manger : 更新管理器对象；

- 返回值说明

返回0表示成功，非0表示失败。

OTA SDK跨平台支持 OTA SDK v1.1

v1.1版本SDK支持Windows、Linux、Mac操作系统，其中依赖的第三方开源库包括：[C标准库](#)、[libcurl](#) 和 [openssl](#)。如果目标平台无法使用上述开源库，则暂不支持使用v1.1版本OTA SDK

OTA SDK v2.x

v2.x版本SDK在v1.1版本基础上，为了能够适配只有[C标准库](#)的IOT设备，将使用[libcurl](#)和[openssl](#)库的部分封装成的单独的静态库，并提供了默认的实现[platform_adapter_default.a](#)，默认实现中使用了[libcurl](#)和[openssl](#)开源库。

- SDK接入方法

- 针对可以使用libcurl和openssl的设备：直接链接[otaskd.a](#)、[libcurl.a](#)、[libssl.a](#)、[libcrypto.a](#)文件即可使用。
- 针对无法使用libcurl和openssl的设备：需要接入方根据[include\bdota\adapter](#)目录中的头文件[ota_cipher.h](#)和[ota_http_stream.h](#)文件定义的接口实现一个静态库，然后与[otaskd.a](#)一起静态链接即可。

OTA SDK示例程序 程序名称

[otaupdate_explain](#)

功能

本示例程序用于测试和体验SDK，其中使用一系列命令行参数，实现对OTA SDK中主要接口的调用。

参数介绍

用法：[otaupdate_explain](#) COMMAND [args]

COMMAND:

`net-update` 通过网络进行升级
`write` 通过本地升级文件升级

`net-update`命令:

- pid 指定product ID
- key 用于指定验证升级包的公钥文件路径
- pidpass base64编码的密钥，用于连接升级服务器
- ver [可选] 指定当前系统版本号，默认为0.0.0.0
- did [可选] 指定设备唯一id，默认为“bdtestdev”
- temp [可选] 设置临时目录
- persistent [可选] 设置持久性目录
- predownload [可选] 指定使用预下载功能
- url [可选] 设置升级服务器url
- reporturl [可选] 设置上报服务器url

`write`命令:

用法: `otaupdate_explain write -key firstsegfile [segfile]`

参数介绍:

| | |
|-----------------|--|
| -key <key_path> | 用于指定验证升级包的公钥文件路径 |
| firstsegfile | 如果使用本地分片OTA升级包升级:此参数为第一个分片升级包路径 如果使用本地完整包升级，此参数为完整包路径 |
| segfile | 如果使用本地分片OTA升级包升级:此参数为需要升级的分片升级包路径 如果使用本地完整包升级，不传入此参数 |

常用功能示例

- 网络正式环境升级 `otaupdate_explain net-update -pid -key -pidpass`
- 网络测试环境升级 `otaupdate_explain net-update -pid -key -pidpass -url -reporturl`
- 使用预下载功能升级 `otaupdate_explain net-update -pid -key -pidpass -persistent -predownload`

示例程序中使用预下载文件升级时，会在查询完成后预下载完整升级包到指定的持久性目录下`predownload.bdota`文件，然后使用该文件进行升级，若不指定`persistent`参数，则下载到当前目录

- 使用本地完整包升级 `otaupdate_explain write -key`
- 使用本地分片升级包升级

升级包分片后所有分片OTA升级包路径表示为: `seg\1 ~ seg\N` , i 从2至N, 循环调用以下命令: `otaupdate_explain write -key seg_1 seg_i`

制作升级包工具 命令行参数说明

```
ota_gen <COMMAND> <PARAM...> [OPTION...]
```

COMMAND (选择一个):

```
gen (*default) // 根据目录生成ota包  
div // 根据ota升级包，切分成片段升级包
```

gen命令支持参数:

```
-t, --target <path>  
对于完整包: 表示准备生成ota包的目标目录  
对于差分包: 表示新版本所在目录  
-o, --output <path>  
设置输出ota包的文件路径  
-d, --desc <path> [可选]  
设置信息描述文件路径  
-i, --incremental_from <path>  
指定进行差分升级, path表示进行差分升级时旧版本所在目录  
-s, --segment <size> 设置分段升级中要求的最大文件大小  
支持k/K/m/M为单位的参数  
-k, --key <path> [可选]  
设置数字签名私钥路径
```

div命令支持参数:

```
-t, --target <path>  
设置ota包路径  
-o, --output <path>  
设置存储分片OTA包目录
```

其他参数:

```
-c, --compress  
对ota包中数据进行压缩  
-h, --help  
显示帮助信息
```

主要功能

本命令行工具包括两个主要功能：生成OTA升级包和OTA升级包分片

- 生成OTA升级包：可以生成完整升级包和差分升级包两类OTA升级包。生成OTA包时，

可通过参数控制以下特性：文件分段、数据压缩。

- OTA升级包分片：将完整升级包按照其中包含的文件（单个完整文件、分段后的分段文件、差分文件），切分成若干个分片OTA升级包。

常用功能示例

- 生成完整OTA升级包

```
ota_pkg_gen gen -t -o -d -k
```

- 生成完整OTA升级包，对数据压缩，并对文件按照1M大小分段

```
ota_pkg_gen gen -t -o -d -k -c -s 1M
```

- 生成差分包

```
ota_pkg_gen gen -t -o -d -k -i
```

- OTA包切分成分片升级包

```
ota_pkg_gen div -t -o
```

4.2.7 数据存储 TSDB

开启存储配置 物接入设备型，支持将数据存储到时序数据库TSDB。可以在[创建物影子](#)时，开启存储配置。同时，也可以在物详情[获取连接配置](#)中，开启与关闭存储配置。

开启存储配置：

- 可修改存储配置，选择不存储，上报即存储和上报满足条件存储三种方式。
- 可将数据存储到时序数据库中（TSDB），即需要选择相应的TSDB实例。
- 配置生效后，物影子会在收到更新时，将更新的数据按配置条件写入时序数据库，这里需要保证时序数据库无欠费等异常情况。



存储TSDB格式 TSDB中表示每个数据点的格式如下：

| TSDB实例 | metric | timestamp | value | tag1 (显示名称) | tag2 (物影子名称) | tag3 (物模型名称) |
|--------|------------------|------------------------|----------|----------------|------------------------------|--------------------------------|
| 需提前创建 | 属性名如 temperature | 属性的更新时间如 1499071119722 | 该属性本次更新值 | “desc” = “温 度” | “deviceName” = “schepanName” | “template_name” = “template_pu |

物影子数据存储TSDB方式有两种：多域存储、单域存储。

- **多域存储：**将物影子的所有属性存入同一个度量 (Metric) 的多个域 (Field) 中，一个属性作为一个域，需要填写度量名称。
- **单域存储：**将物影子的每个属性单独存入一个度量中，作为一个域，属性名称即度量名称。

详细可参考[TSDB产品介绍](#)



4.2.8 物影子数据可视化

天工物可视作为可视化开发工具，无缝对接物管理数据，可以通过物可视将物管理设备数据可视化。物可视具体介绍可参考[物可视产品介绍](#)。

进入物可视服务，选择新建仪表盘

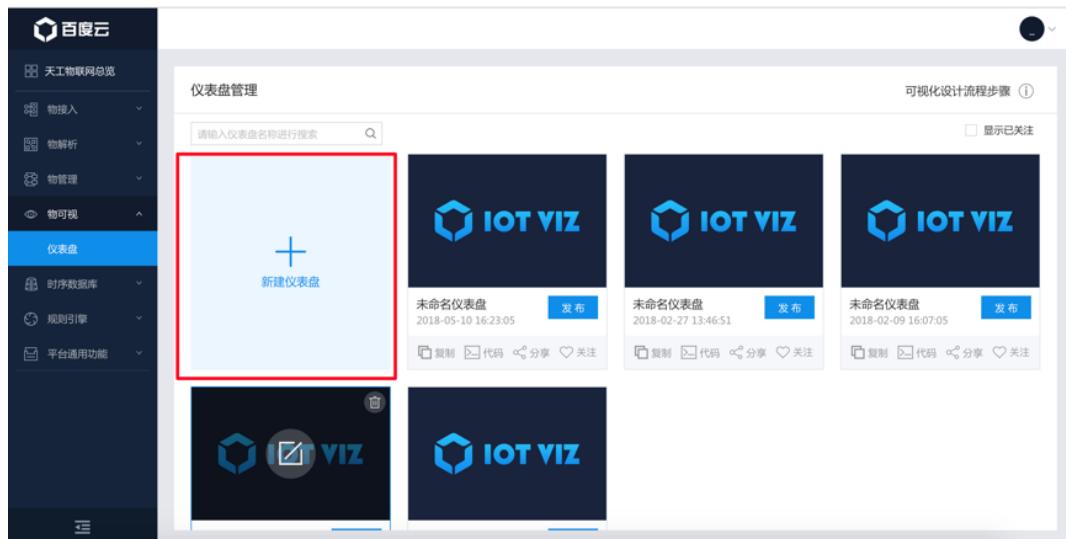


图 4.4: 片

进入仪表盘后，选择数据表

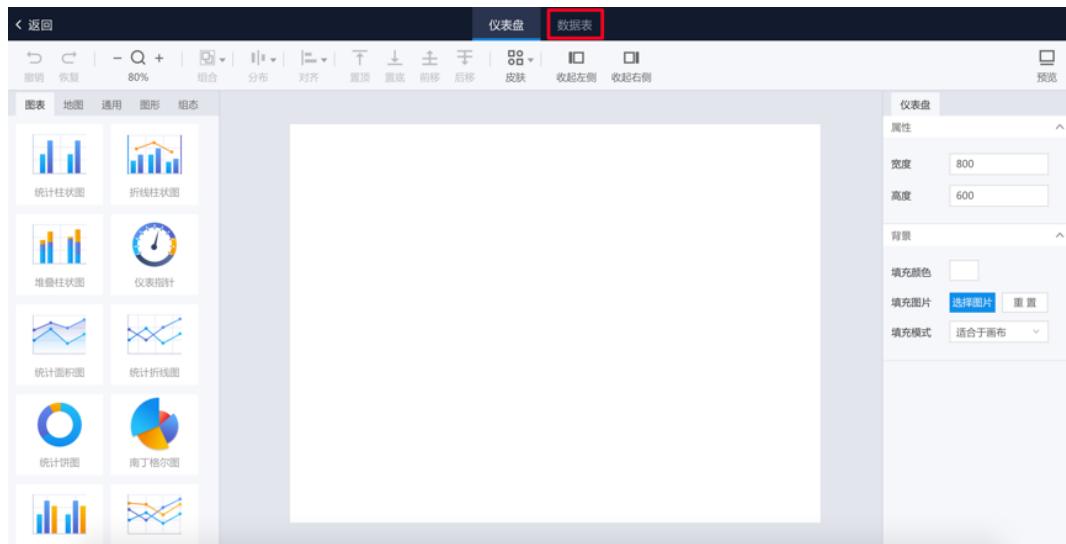


图 4.5：片

新建数据表

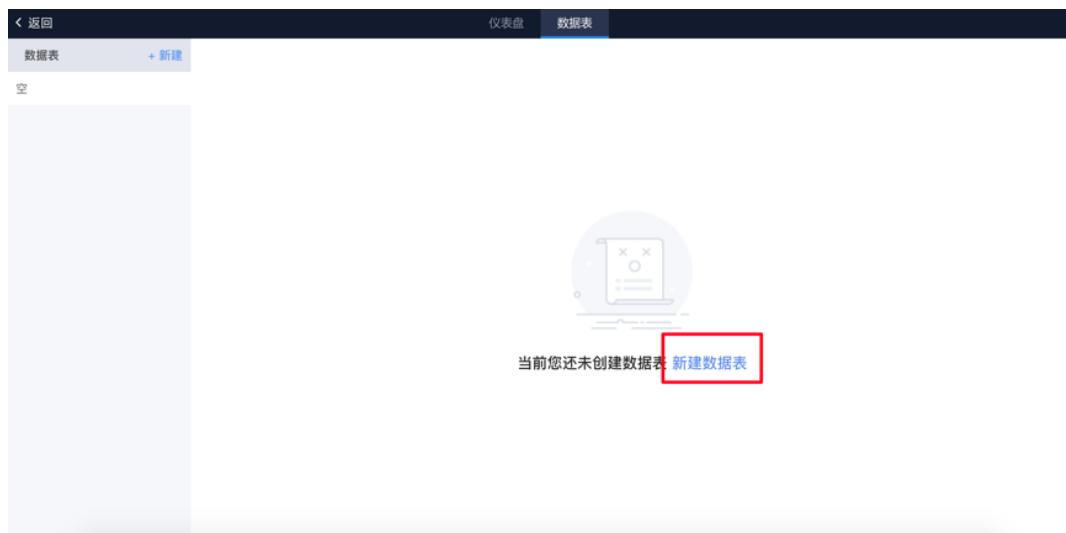


图 4.6：片

选择物管理作为数据源

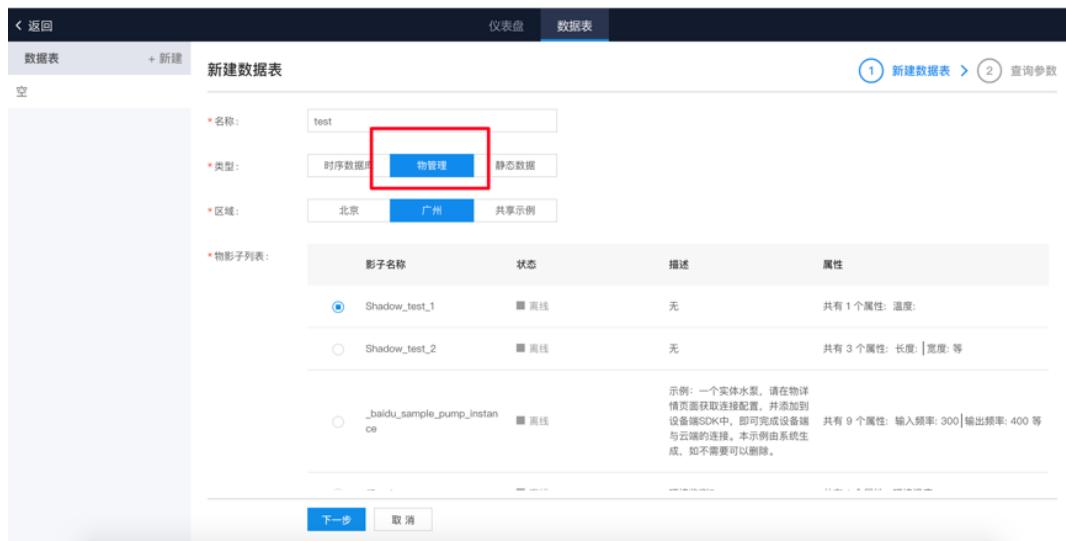


图 4.7：片

选择想要展示的属性，点击确认，就会为物可视仪表盘生成物管理数据源

| test | | timestamp | device | temp |
|---------------------|--|---------------|--------|------|
| 2018-05-22 02:26:21 | | Shadow_test_1 | 0.00 | |

图 4.8：片

回到设计器页面，选取需要展示的组件，将数据配置改为之前增加的物管理数据表，即可完成物管理设备数据可视化，更多详情可参考[物可视介绍](#)。

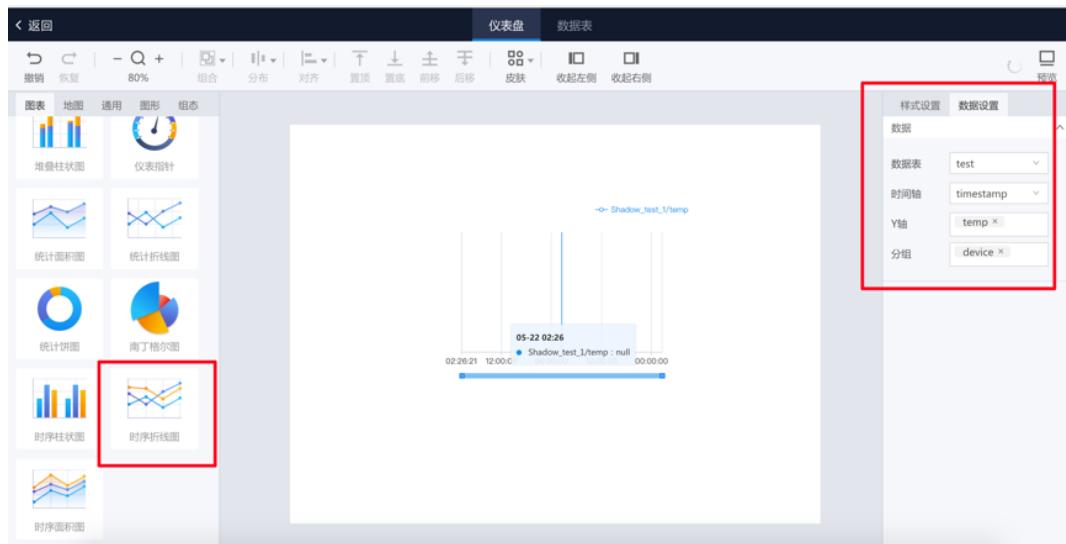


图 4.9: 片

4.3 数据型项目

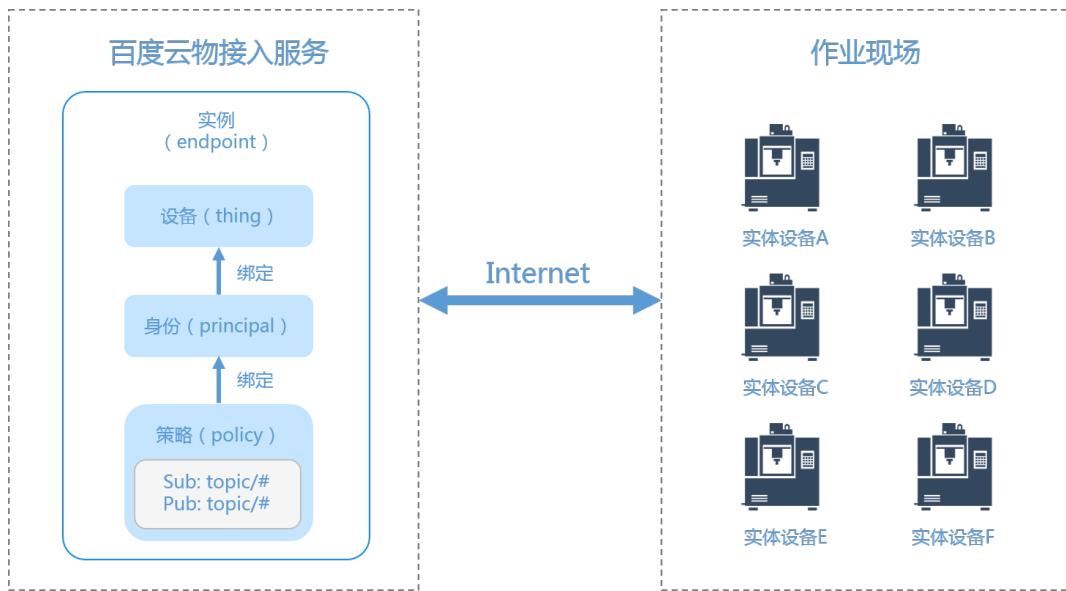
4.3.1 操作准备

说明

如果您还不了解MQTT协议，推荐您首先查看[MQTT协议介绍](#)，了解MQTT的工作原理。

- 物接入项目 (endpoint)：简称“项目”，代表一个完整的物接入服务。一个百度云账号可以创建100个项目。
- 物接入用户 (thing)：简称“用户”，在项目中创建的虚拟用户，每个项目下最多可以创建10000个用户。
- 物接入身份 (principal)：简称“身份”，在项目中创建的虚拟设备的身份，每个设备可以绑定一个身份。每个项目下最多可以创建10000个身份。
- 物接入策略 (policy)：简称“策略”，策略中定义了关于特定主题的收发权限，每个身份可以绑定一个策略。每个项目下最多可以创建10000个策略。
- MQTT主题 (topic)：简称“主题”，每个策略都需要指定主题及主题对应的权限。该主题应用于MQTT客户端。物接入允许主题中带一个通配符“#”，例如“temperature/#”就是匹配前缀是temperature/的所有topic；单独的“#”表示匹配所有topic。
- 实体设备：通过物接入连接的实体设备。

有关项目实例、用户、身份、策略和实体设备的关系，请参看下图：

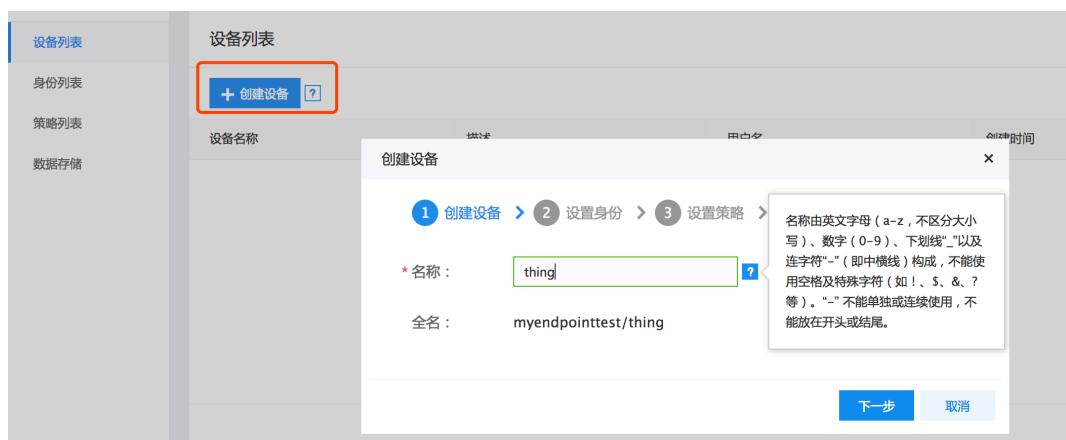


4.3.2 数据型项目配置

创建物接入项目 详见[快速入门-步骤三：创建项目](#)

创建物接入用户 创建物接入设备前，应先完成[创建物接入项目操作](#)。

1. 选择“产品服务>物接入IoT Hub”，进入项目列表。
 2. 点击项目名称，进入物接入项目页面。
 3. 点击“创建用户”，在弹出窗口中输入用户名，并点击“下一步”。
- “项目Endpoint名称/用户名”将作为实体设备连接云端的用户名。



4. 为用户绑定身份。此处可以直接从下拉菜单中选择身份名称，如果没有可用身份，也可以选择“+创建”，创建新身份。



5. 输入身份名称，并点击“下一步”。
6. 为身份绑定策略。此处可以直接从下拉菜单中选择策略名称，如果没有可用策略，也可以选择“+创建”，创建新策略。
7. 输入策略名称、主题及主题的权限，点击“下一步”完成用户创建。

说明:每个策略下只能配置100个主题。



每个策略可以创建多个主题，在创建策略弹框右侧，点击“+”可以绑定更多的主题。添加主题时，可以使用“#”或“+”作为通配符，关于通配符的介绍，请参看[关于通配符的使用方法] (<https://cloud.baidu.com/doc/IOT/GettingStarted.html#.E5.85.B3.E4.BA.8E.E9.80.9A.E9.85.8D.E7.AC.A6.E7.9A.84.E4.BD.BF.E7.94.A8.E6.96.B9.E6.B3.95>)。

创建新用户后，系统将自动初始化该身份对应的密钥。该密钥将用于后续实体设备登录，请妥善保管。



您也可以使用IoT Hub CLI命令创建各项参数，详细过程请参考[IoT Hub CLI](#)。

视频操作指导-创建物接入用户（无语音）

创建物接入身份 身份需要与用户绑定后才有意义。用户可以在创建用户的过程中创建身份（具体操作请参考[创建物接入用户](#)），也可以单独创建。

创建物接入用户前，应先完成[创建物接入项目](#)操作。

1. 选择“产品服务>物接入IoT Hub”，进入项目列表。
2. 点击项目名称，进入物接入项目页面。
3. 点击左侧导航中的“身份列表”，进入身份列表。

| 身份名称 | 创建时间 | 操作 |
|-----------|---------------------|----------------|
| principal | 2017-12-11 20:10:27 | 编辑 删除 重置密钥 |

4. 点击“创建身份”，在弹出窗口中输入身份名称，并点击“下一步”。
5. 为身份绑定策略。此处可以直接从下拉菜单中选择策略名称，如果没有可用策略，也可以选择“+创建”，创建新策略。



6. 输入策略名称、主题及主题的权限，点击“下一步”完成身份创建。

说明：每个策略下只能配置100个主题。

每个策略可以创建多个主题，在创建策略弹框右侧，点击“+”可以绑定更多的主题。添加主题时，可以使用“#”或“+”作为通配符，关于通配符的介绍，请参看[关于通配符的使用方法] (<https://cloud.baidu.com/doc/IOT/GettingStarted.html#.E5.85.B3.E4.BA.8E.E9.80.9A.E9.85.8D.E7.AC.A6.E7.9A.84.E4.BD.BF.E7.94.A8.E6.96.B9.E6.B3.95>)。

创建新身份后，系统将自动初始化改身份对应的密钥。该密钥将用于后续实体设备登录，请妥善保管。



说明：重新生成密钥后，已经连接的用户会断开连接，请谨慎使用。

每个身份对应一个密钥，该密钥用于实体设备连接物接入服务。创建新身份后，系统将自动初始化改身份对应的密钥。如果用户忘记该密钥，可重新生成密钥。具体操作如下：

1. 选择“产品服务>物接入IoT Hub”，进入项目列表。
2. 点击项目名称，进入物接入项目页面。
3. 点击左侧导航中的“身份列表”，进入身份列表。

4. 找到指定的身份，点击“重新生成密钥”。此时系统将在弹出对话框中给出重新初始化后的密钥，请妥善保管。

| 身份名称 | 创建时间 | 操作 |
|-------------|---------------------|--|
| principal01 | 2017-12-11 20:23:43 | 编辑 删除 重置密钥 |
| principal | 2017-12-11 20:10:27 | 编辑 删除 重置密钥 |

创建物接入策略 策略需要与身份和用户绑定后才有意义。用户可以在创建用户的过程中创建策略（具体操作请参考[创建物接入用户](#)），也可以单独创建。

创建物接入策略前，应先完成[创建物接入项目](#)操作。

1. 选择“产品服务>物接入IoT Hub”，进入项目列表。
2. 点击项目名称，进入物接入项目页面。
3. 点击左侧导航中的“策略列表”，进入策略列表。

| 策略名称 | 主题 | 创建时间 | 操作 |
|--------|-----------------|---------------------|---------------------------------------|
| policy | topic01,topic01 | 2017-12-11 20:10:27 | 编辑 删除 |

4. 点击“创建策略”，在弹出窗口中输入策略名称、主题及主题的权限，点击“确定”完成策略创建。

说明：重新生成密钥后，已经连接的用户会断开连接，请谨慎使用。

每个策略可以创建多个主题，在创建策略弹框右侧，点击“+”可以绑定更多的主题。添加主题时，可以使用“#”或“+”作为通配符，关于通配符的介绍，请参看[关于通配符的使用方法](#)。

* 名称：

* 主题： ?

* 权限： 发布(PUB) 订阅(SUB)

注：点击右侧加(+)号为本策略绑定更多的主题。

关于通配符的使用方法 MQTT通过“主题”实现将消息从发布者客户端送达至接收者客户端。“主题”是附加在应用消息上的一个标签，发布者客户端将“主题”和“消息”发送至代理服务器，代理服务器将该消息转发至每一个订阅了该“主题”的订阅者客户端。

一个主题名可以由多个主题层级组成，每一层通过“/”斜杠分隔开，例如：“baidu/F1”，“baidu/F2”。如果用户需要一次订阅多个具有类似结构的主题，可以在主题过滤器中包含通配符。通配符只可用在主题过滤器中，在发布应用消息时的主题名不允许包含通配符，主题通配符有两种：

- \#: 表示匹配 ≥ 0 个层次，比如a/#就匹配a, a/, a/b, a/b/c。单独的一个#表示匹配所有，不允许a#或a#/c等形式。
- \+: 表示匹配一个层次，例如a/+匹配a/b, a/c，不匹配a/b/c。单独的一个+是允许的，但a+为非法形式。

通配符可以应用在物接入策略中和实体设备的订阅主题中，通过以下示例我们可以进一步了解通配符的作用。

我们使用下表的配置在云端创建三个物接入用户。

| 项目 Endpoint 名称 | 用户名 | 身份名称 | 策略名称 | 主题 | 权限 |
|----------------|---------|-------------|----------|--------------------|-------|
| endpoint01 | thing01 | principal01 | policy01 | baidu/# | 发布/订阅 |
| endpoint01 | thing02 | principal02 | policy02 | baidu/+/area1 | 发布/订阅 |
| endpoint01 | thing03 | principal03 | policy03 | baidu/floor1 | 发布/订阅 |
| endpoint01 | thing04 | principal04 | policy04 | baidu/floor1/area1 | 发布/订阅 |

通过MQTT.fx客户端模拟4台实体设备（Device01 ~ Device04），分别使用thing01、thing02、thing03和thing04的用户名和密码连接物接入服务。

互通情况测试结果可参看下表：

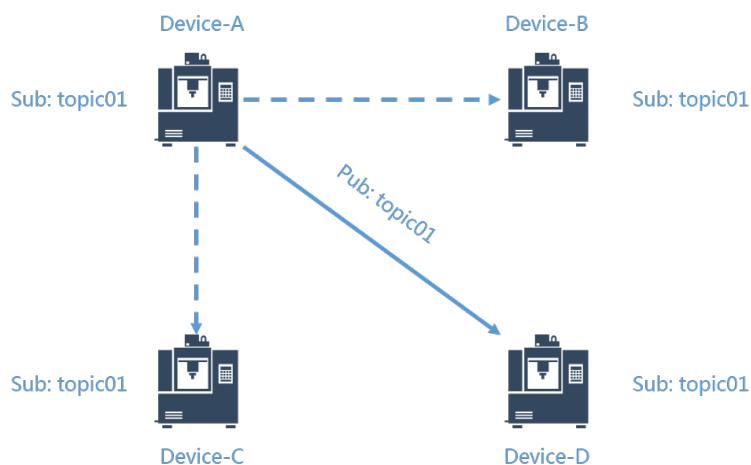
| 发布用户 | 发布主题 | 订阅用户 | 订阅主题 | 是否可以互通 |
|----------|----------------------|----------|----------------------|--------|
| Device01 | baidu/device02/area1 | Device02 | baidu/device02/area1 | 是 |
| Device01 | baidu/floor1 | Device03 | baidu/floor1 | 是 |
| Device01 | baidu/floor1/area1 | Device04 | baidu/floor1/area1 | 是 |

| 发布用户 | 发布主题 | 订阅用户 | 订阅主题 | 是否可以互通 |
|----------|----------------------|----------|----------------------|--------|
| Device02 | baidu/device01/area1 | Device01 | baidu/device01/area1 | 是 |
| Device02 | - | Device03 | - | 否 |
| Device02 | baidu/floor1/area1 | Device04 | baidu/floor1/area1 | 是 |
| Device03 | baidu/floor1 | Device01 | baidu/floor1 | 是 |
| Device03 | - | Device02 | - | 否 |
| Device03 | - | Device04 | - | 否 |
| Device04 | baidu/floor1/area1 | Device01 | baidu/floor1/area1 | 是 |
| Device04 | baidu/floor1/area1 | Device02 | baidu/floor1/area1 | 是 |
| Device04 | - | Device03 | - | 否 |

避免用广播方式向特定用户发送消息 在百度云创建一个物接入用户，具体配置如下：

| 项目 Endpoint 名称 | 用户名 | 身份名称 | 策略名称 | 主题 | 权限 |
|----------------|---------|-------------|----------|---------|-------|
| endpoint01 | thing01 | principal01 | policy01 | topic01 | 发布/订阅 |

如下图所示，在该场景中所有实体设备都通过相同的物接入用户接入，拥有相同的用户名、密码和权限。由于所有用户都订阅了相同的话题，当Device-A向Device-D发布消息时Device-B和Device-C也会收到。这种部署方式会对实体设备造成额外的处理负担，同时也会对当月的物接入消息配额造成不必要的损失。



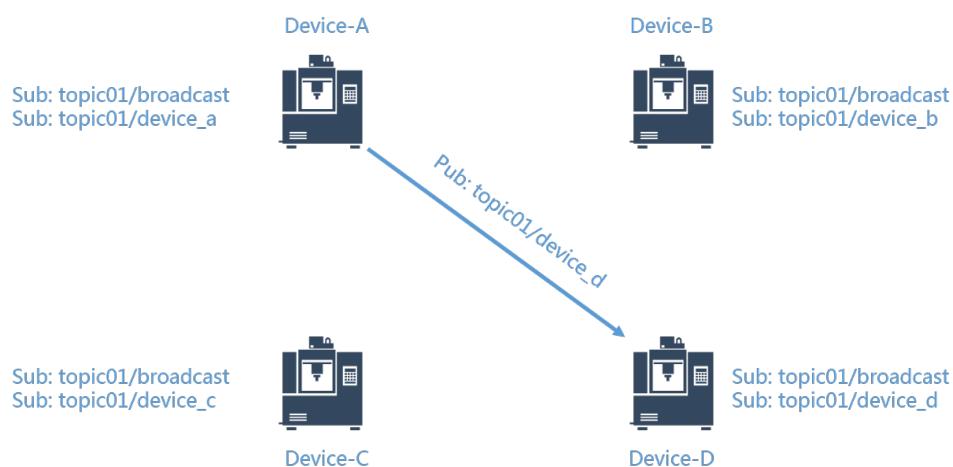
在不增加额外配置的情况下，可以采取以下部署方式解决上述问题。在百度云创建一个物接入用户，具体配置如下：

| 项目 Endpoint 名称 | 用户名 | 身份名称 | 策略名称 | 主题 | 权限 |
|----------------|---------|-------------|----------|-----------|-------|
| endpoint01 | thing01 | principal01 | policy01 | topic01/# | 发布/订阅 |

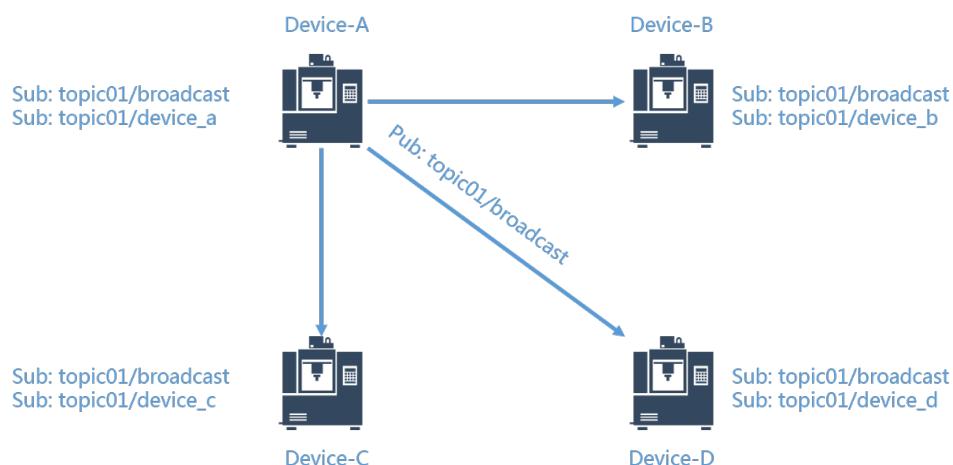
由于在主题中使用了“#”通配符，因此每个用户可以订阅两个主题：

- topic01/broadcast：用来订阅广播消息。
- topic01/DeviceName：DeviceName需要替换为用户的实际名称并全局唯一，用来订阅单播消息。

在单播场景下，Device-A向Device-D发送消息，Device-B和Device-C都不会接收到。



在广播场景下，Device-A发布的消息可以同时被Device-B、Device-C和Device-D接收到。



4.3.3 连接测试

创建设备后，可以在控制台通过“连接测试”功能模拟MQTT客户端，验证物接入的连接情况。用户也可以通过[MQTT.fx](#)验证连接情况。

在执行连接测试前，必须先[创建物接入设备](#)。

1. 选择“产品服务>物接入IoT Hub”，进入实例列表。
2. 点击实例名称，进入物接入实例页面。

3. 点击“连接测试”，进入测试页面，填写“身份密钥”，点击“connect”，此时可看到设备状态由“未连接”变成“运行中”。

有关其它参数的介绍，如下表所示：

| 参数名称 | 描述 | 必要性 |
|------|---|-----|
| 主机名称 | 实例地址, yourend-point.mqt.t.iot.gz.baidubce.com | 必填 |
| 端口 | wss端口8884 | 必填 |

| 参数名称 | 描述 | 必要性 |
|---|--|-----|
| 用户ID | 客户端ID，用户自定义。在同一个实例下，每个实体设备需要有一个唯一的ID，不同实体设备使用同一个client id建立连接会导致其它连接下线。用户ID只支持英文大小写字母，数字0-9，中划线和下划线，不支持其它字符。 | 必填 |
| 用户名 | 成功创建thing后生成的用户名， your endpoint / 设备名称 | 必填 |
| 密码 | 与thing绑定的principal的密码。成功创建身份principal后系统会自动生成的密码 | 必填 |
| Keep Alive 连接保持时长，单位为秒 必填 SSL SSL 安全验证 必填 Clean Session | 清理会话 | 选填 |
| Last-Will Topic | 遗嘱消息主题 | 选填 |
| Last-Will QoS | 遗嘱QoS，发布遗嘱消息时使用的服务等级 | 选填 |
| Last-Will Retain | 遗嘱保留，如果勾选遗嘱保留，遗嘱消息发布时将会保留且发送给新的订阅消息 | 选填 |
| Last-Will Message | 遗嘱消息，在网络连接关闭后，IoT Hub将会自动发布本条遗嘱消息 | 选填 |

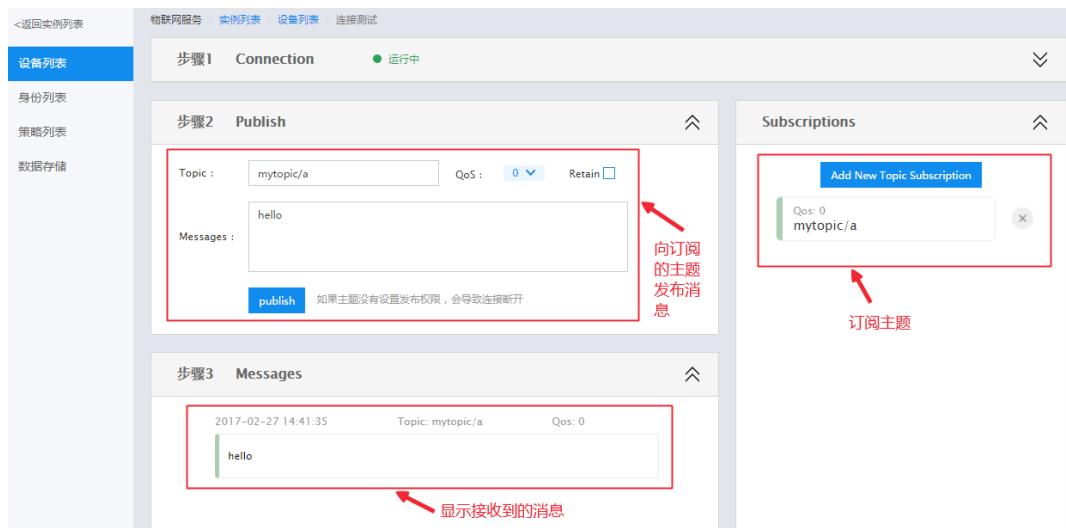
***Keep Alive：**MQTT协议是一个客户端和服务器端长连接的过程。Keep Alive timer以秒为单位，定义的是从客户端相邻两次接收消息的最大间隔时间，也可以说是一次长连接的保持时间。因此，客户端每隔一段时间就需要向服务器发送数据来保持连接（相当于心跳报文的功能），服务器接收到连接信息后，会反馈一个响应ACK。当服务器端在Keep Alive timer的1.5倍时间内都没有收到来自客户端的任何消息，就会默认为客户端断开连接。

***Clean Session：**如果该位被设置为false，则该连接被认为是持久连接，其具体表现为：当该客户断开后，任何订阅的主题和QoS被设置为1或2的信息都会保存，直到该客户端再次连接上server端，物接入服务支持将该消息保留24小时。若“clean session”被设置为true，当该客户断开后，所有的订阅主题都会被移除。

Last-Will当一个客户端断开连接的时候，它希望客户端可以发送它指定的消息。该消息

和普通消息的结构相同。通过设置Last-Will Topic和Last-Will Message实现。（[遗嘱消息的触发条件有哪些？](#)）

连接成功后，可以设置消息订阅和发布，通过自发自收的方式测试连接，如下图所示：



4.3.4 基于证书的双向认证

天工物接入支持双向认证，在创建身份的时候可以选择“用户名密码方式”连接，或者PKI证书连接，设备嵌入证书即可和云端连接。此时PKI证书不再额外收费。

用户可以使用PKI证书实现和云端的TLS双向链接，保证链路层通道层的加密。

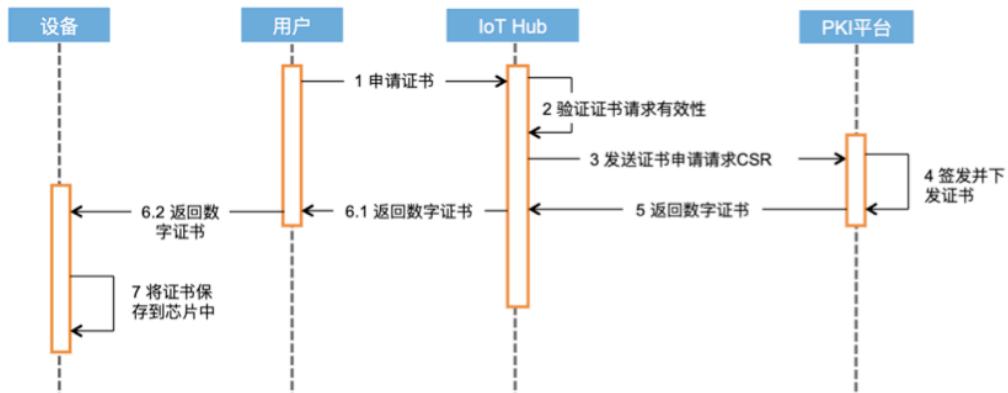


图 4.10： 片

百度云IoT的PKI证书服务还支持客户为自己的设备创建CA、证书管理等服务，更多使用百度PKI证书进行设备和云端双向认证的方式，请参考[PKI产品文档](#)。

物接入双向认证操作流程如下：

1. 创建身份：创建项目实例后，进入项目，选择身份列表，创建身份，选择证书认证。点击下一步



图 4.11：片

2. 选择绑定策略/创建策略。点击下一步。



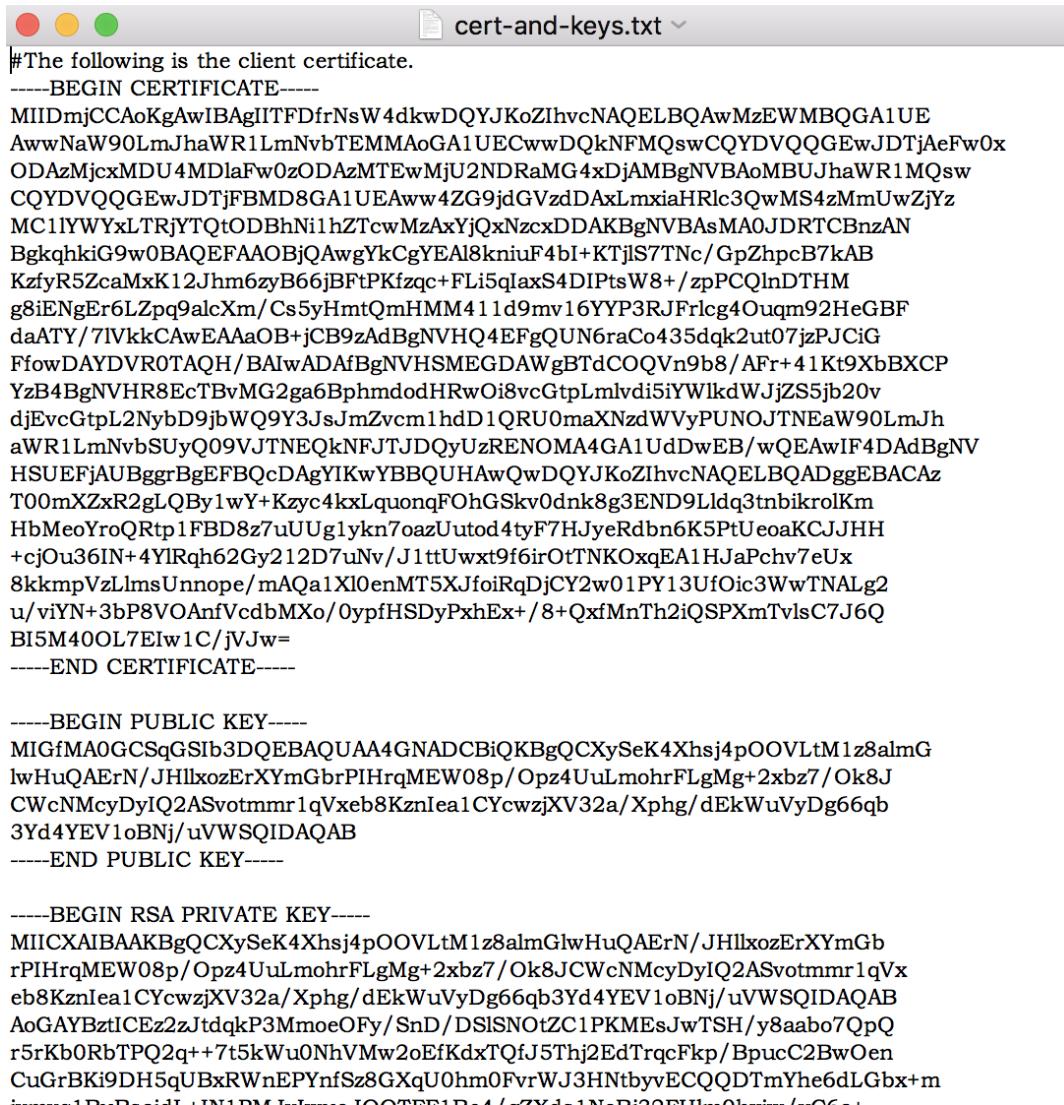
图 4.12：片

3. 配置确认后，生成对应证书，点击下载。



图 4.13：片

会生成如截图的证书：



```
#The following is the client certificate.
-----BEGIN CERTIFICATE-----
MIIDmjCCAoKgAwIBAgIITFDfrNsW4dkwDQYJKoZIhvcNAQELBQAwMzEWMBQGA1UE
AwwNaW90LmJhaWR1LmNvbTEMMAoGA1UECwwDQkNFMQswCQYDVQQGEwJDTjAeFw0x
ODAzMjcxMDU4MDlaFw0zODAzMTEwMjU2NDRaMG4xDjAMBgNVBAoMBUJhaWR1MQsw
CQYDVQQGEwJDTjFBMD8GA1UEAww4ZG9jdGVzdDAxLmxiaHRlc3QwMS4zMmUwZjYz
MC1IYWYxLTRjYTQtODBhNi1hZTcwMzAxYjQxNzcxDDAKBgNVBAsMA0JDRTCnzAN
BgkqhkiG9w0BAQEFAAOBJQAwgYkCgYEAl8kniuF4bI+KTjIS7TNC/GpZhpcB7kAB
KzfyR5ZcaMxK12Jhm6zyB66jBFtPKfzqc+FLi5qIaxS4DIptsW8+/zpPCQlnDTHM
g8iENgEr6LZpq9alcXm/Cs5yHmtQmHMM411d9mv16YYP3RJFrlcg4Ouqm92HeGBF
daATY/7lVkkCAwEAAoOB+jCB9zAdBgNVHQ4EFgQUN6raCo435dqk2ut07jzPJCiG
FflowDAYDVR0TAQH/BAIwADAfBgNVHSMEGDAwgbTdCOQVn9b8/AFr+41Kt9XbBXCP
YzB4BgNVHR8EcTBvMG2ga6BphmdodHRwOi8vcGtpLmlvdi5iYWlkdWJjZS5jb20v
djEvcGtpL2NybD9jbWQ9Y3JsJmZvcm1hdD1QU0maXNzdWVvPUN0JTNEaW90LmJh
aWR1LmNvbSUyQ09VJTNEQkNFJTJDQyUzRENOMA4GA1UdDwEB/wQEAvIF4DAdBgNV
HSUEFjAUBgrBgfEFBQcDAgYIKwYBBQUHAwQwDQYJKoZIhvcNAQELBQADggEBACAz
T00mXZxR2gLQBy1wY+Kzyc4kxLquonqFOhGSkv0dnk8g3END9Lldq3tnbikrolKm
HbMeoYroQRtp1FBD8z7uUuglykn7oazUutod4tyF7HJyeRdbn6K5PtUeoakCJJHH
+cjOu36IN+4YIRqh62Gy212D7uNv/J1ttUwxt9f6irOfTNKOxqEA1HJaPchv7eUx
8kkmpVzLlmsUnnope/mAQa1XI0enMT5XJfoiRqDjCY2w01PY13UfOic3WwTNALg2
u/viYN+3bP8VOAnfVcdMXo/0yfHSDyPxhEx+/8+QxfMnTh2iQSPXmTvlsC7J6Q
BI5M40OL7EIw1C/jVJw=
-----END CERTIFICATE-----

-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCXYSeK4Xhsj4pOOVLtM1z8almG
lwHuQAErN/JHllxozErXYmGbrPIHrqMEW08p/Opz4UuLmohrFLgMg+2xbz7/Ok8J
CWcNMcyDyIQ2ASvtmmr1qVxeb8KznIea1CYcwzjXV32a/Xphg/dEkWuVyDg66qb
3Yd4YEV1oBNj/uVWSQIDAQAB
-----END PUBLIC KEY-----

-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQCXYSeK4Xhsj4pOOVLtM1z8almGlwHuQAErN/JHllxozErXYmGb
rPIHrqMEW08p/Opz4UuLmohrFLgMg+2xbz7/Ok8JWCeNMcyDyIQ2ASvtmmr1qVx
eb8KznIea1CYcwzjXV32a/Xphg/dEkWuVyDg66qb3Yd4YEV1oBNj/uVWSQIDAQAB
AoGAYBztICEz2zJtdqkP3MmoeOFy/SnD/DSISNOTZC1PKMEsJwTSH/y8aab07QpQ
r5rKb0RbTPQ2q++7t5kWu0NhVMw2oEfKdxTQfj5Thj2EdTrqcFkp/BpucC2BwOen
CuGrBKi9DH5qUBxRWnEPYnfSz8GXqU0hm0FvrWJ3HNtbyvECQQDTmYhe6dLGbx+m
-----END RSA PRIVATE KEY-----
```

图 4.14：片

4. 设备嵌入证书，在此以MQTT.fx客户端为例。在Connection Profile中选择SSL/TLS。

> 注意：密码换为证书，用户名仍然需要填写

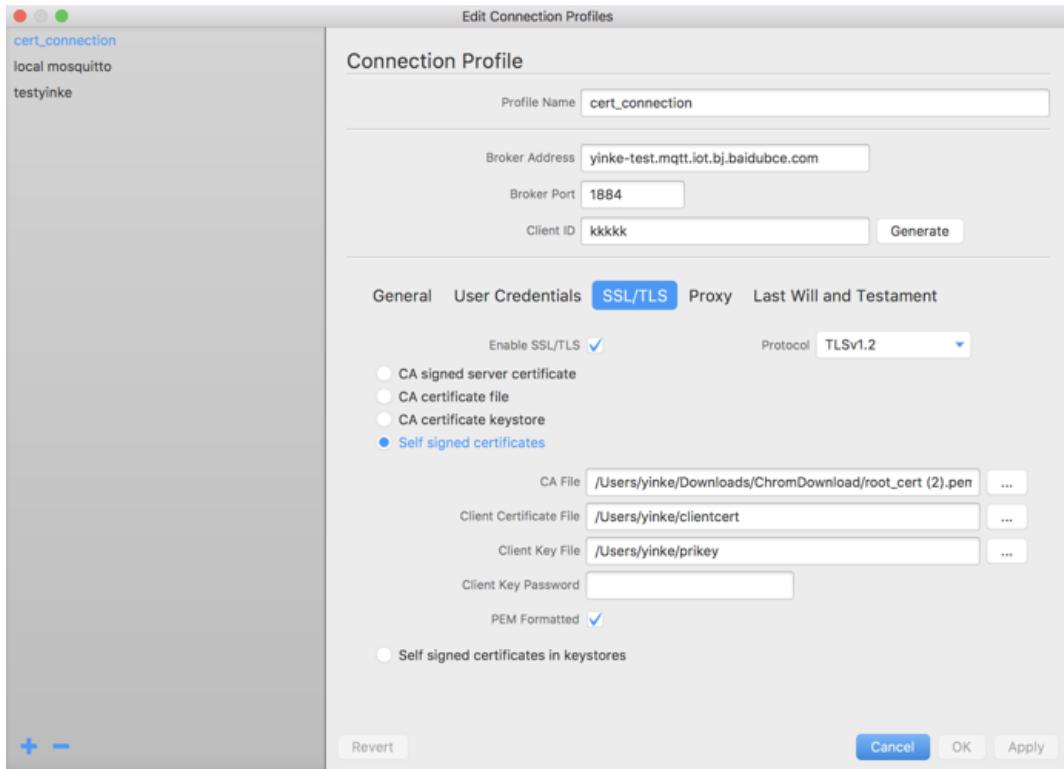


图 4.15：片

- CA File: TLS认证文件下载，点此链接下载 https://sdk.bce.baidu.com/console-sdk/root_cert.pem?responseContentDisposition=attachment
- Client Certificate File: 将CERTIFICATE这段复制到文本中另存为，导入路径

```
#The following is the client certificate.
-----BEGIN CERTIFICATE-----
MIIEGjCCAkwKgIBAgIIWAB+o6jT93EwDQYJKoZIhvvcNAQELBQAwMzEWMBQGA1UE
AwwNaW90LmJhaWR1LmNvbTEMMAoGA1UECwxDQkNFMQswCQYDVQQGEwJDTjAeFw0x
ODA0Mjgw0DQ3NDFaFw0z0DAzMTExMjU2NDRaMGoxDjAMBgNVBAoMBUJhaWR1MQsw
CQYDVQQGEwJDTjE9MDsGA1UEAww0dGV0ZXRLLnRlc3QxMjMzMm3MWYyYWFlLThl
NzctNGIzMS04MzI3LTBk0TdhNjIyMTYwMTEMMAoGA1UECwxDQkNFMIIBIjANBgkq
hkiGw0BAQEFAAOCAQ8AMIIIBcgKCAQEAAo6cx0Pv+mTRa/070oHKSX5UsS4+Z+10S
3doZHPuE54B43pQC/ekQwjL3pGtlaMn6hC569ecVgZEzl3+tsa+UngC8gjB8WmSQ
bSA2C1zMNgBQ9L61CSijpkavofkhv3dqZwSqwK5DgWX0B9M47097hYM09J9oSRh
U1icUfDSULW01GjYGn6sDZ69BTnGCUhv/jd20MUzD96F+dX1JS4ivg4atdLlqBf
DmHf850ATCVAMXSWQhk1BHQceXSeGdb/iEoe3B7ZClrRQL0Ady0J7NZsFkx5qzQ
1InjdA11teHMUeICrlW11ukSoiLysVbnksF1aG5aq20nDfoalPdbXQIDAQAB04H6
MIH3MB0GA1UdDgQWBFAQ+BI1Vi03N0BJnYU8C5tqY0boDAMbgNVHRMBAf8EAjAA
MB8GA1UdIwQYMBaAFBQ0I5BWf1vz8AWv7jUq31dsFcI9jMHgGA1UdHwRxMG8wbaBr
oGmGZ2h0dHA6Ly9wa2kuaW92LnJhaWR1YmNlLmNb92MS9wa2kvY3JsP2NTZD1j
cmwmZm9ybWF0PVBFTSZpc3N1ZXI9Q04lM0Rpb3QuYmFpZHUuY29tJTJDT1U1LM0RC
Q0U1MkNDJTNEQ04wDgYDVR0PAQH/BAQDAgP4MB0GA1UdJQQWMBQGCCsGAQUFBwMC
BggRFBQcDBDANBgkqhkiG9w0BAQsFAAOCAQEAVfkODj3IRZRYHl7NBMo9UD
Ug0Vz+q3D4bo0WLuz3y9b2bpBELfqQmGMB0ZPna7tK8R8gHQWlPPm2DzP0uhDN2
bCUMg9T2s9NgGdkcoyfRIjHHJS89ewoDePTw+7DcyuTUD/RT1xbijErxFZLQCH
mbuGcbCceWp0B+a3yqfbawcPcl6Ew4vpHHRsj31P3ec12r/ikIQszJ+SQLFpn9s5
rKhT2qm+wxyVKQYFRDUNdJe7e0wgoF675u/ai0T/rAwFYYYY4jE7JYMKL+2H8WI7B
aa+Dw3u0V8sQlqznaWh2eGZo5+FjyVdSVFeQCeTE72P1MYF8QV0919mHUB7tAw==
-----END CERTIFICATE-----

-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIIBcgKCAQEAAo6cx0Pv+mTRa/070oHKS
X5UsS4+Z+10S3doZHPuE54B43pQC/ekQwjL3pGtlaMn6hC569ecVgZEzl3+tsa+U
ngC8gjB8WmSQbSA2C1zMNgBQ9L61CSijpkavofkhv3dqZwSqwK5DgWX0B9M47097

```

- Client Key File: 将PRIVATE KEY这段复制到文本中另存为，导入路径



```

cert-and-keys.txt

-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCXySeK4Xhsj4pOOVLtM1z8almG
lwHuQAErN/JHllxozErXYmGbrPIHrqMEW08p/Opz4UuLmohrFLgMg+2xbz7/Ok8J
CWcNMcyDyIQ2ASvtmmr1qVxeb8KznIea1CYcwzjXV32a/Xphg/dEkWuVyDg66qb
3Yd4YEV1oBNj/uVWSQIDAQAB
-----END PUBLIC KEY-----

-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQCXySeK4Xhsj4pOOVLtM1z8almGlwHuQAErN/JHllxozErXYmGb
rPIHrqMEW08p/Opz4UuLmohrFLgMg+2xbz7/Ok8JCWcNMcyDyIQ2ASvtmmr1qVx
eb8KznIea1CYcwzjXV32a/Xphg/dEkWuVyDg66qb3Yd4YEV1oBNj/uVWSQIDAQAB
AoGAYBztICEz2zJtdqkP3MmoeOFy/SnD/DSISNOtZC1PKMEsJwTSH/y8aab07QpQ
r5rKb0RbTPQ2q++7t5kWu0NhVMw2oEfKdxTQfJ5Thj2EdTrqcFkp/BpuC2BwOen
CuGrBK19DH5qUBxRWnEPYnfSz8GXqU0hm0FvrWJ3HNtbyvECQQDTmYhe6dLGbx+m
ivmvqlBuBsaidL+IN1PMJvlwuoJQQTFF1Be4/gZXds1NcBj32FHIml0xiw/yC6a+
8ZYTYMOVAlkEAt6KavTA9+I2nQb+Q4vJQABb5V2MxAZ8KpwIAdveOG5ECd5BzW9Ql
7+hOyw9KITktHaQFFDwqn/cntBP79g/5ZQJAclIK/hzAvGcH2g1h8IRMnZgmGTtvJ
DivVC8Vdm9lf9wpjB60ZnAy+FH2f9l/3drI3+m1R3MXDicD2Pas5UD51bQJASihu
WOx8ej2qR9D2z8+PIxokA3hZOBBkGuGiEmkTVy6OUUn6RN4pK78Fe4H4CL7yJTakJ
LAzIIeWLsRtl6yu0SQJBAsq2PbyY1i5z9rnnCNA3dzpaGr+g7SA5/SSm+c1U5ka
WPwAS/iuTc1Lw6eqy0rDuzdNOBHWkfr+dXFNRROBdrQ=
-----END RSA PRIVATE KEY-----

#The following is the certificate of CA.
-----BEGIN CERTIFICATE-----
MIIECzCCAvOgAwIBAgIIY2iFs2sfeg4AwDOYJKoZIhvcNAOELBOAwMzEWMBQGA1UE

```

图 4.16：片

鉴于不同客户端嵌入证书的方式不同，我们将CERTIFICATE、PUBLIC KEY、PRIVATE KEY、都集成在同一个文本中。同时在JAVA SDK也支持创建证书、重置证书。详情请参考[Java SDK](#)

4.3.5 数据型消息存储配置

用户可以通过[规则引擎](#)将物接入服务接收到的消息转存至BOS或者百度Kafka进行存储。

规则引擎为免费服务，有关规则引擎的详细操作，请参看[规则引擎-操作指南](#)。

4.4 其他

4.4.1 多用户协作

多用户访问控制功能实现了多用户协同开发，项目管理者可以基于项目为其他开发测试人员开放查看、配置等权限。项目管理者通过IAM子账号方式进行授权，即在主账号下添加子用户的账号并对子用户进行策略管理，子用户可以通过“IAM用户登录链接”访问主用户的资源。

物接入子账户权限范围说明如下：

| 权限资源粒度 | 权限 | 说明 |
|--------------|----|---|
| 项目(endpoint) | 只读 | 被授权用户可以查看“项目列表”（只能看到有权限的项目）、“用户列表”、“身份列表”、“策略列表”、“用量统计”页面，只能查看，不能做任何修改操作； |
| 项目(endpoint) | 管理 | 被授权用户针对该项目享有与主账号权限一致的权限（不包括删除、创建endpoint，不包括物接入服务额度升级和续费） |

配置方法

1. 主账号用户登录后在控制台左下角选择“多用户访问控制”进入用户中心/IAM用户页面。
2. 在“子用户管理列表”页面点击新建用户，填写“用户名”，用户名为子用户的昵称，说明主要用于标识用户类型或其它说明。
创建成功后，系统会自动为该用户生成AK/SK，请妥善记录保管。
3. 点击“去绑定”，将该用户与子用户的百度账号绑定。

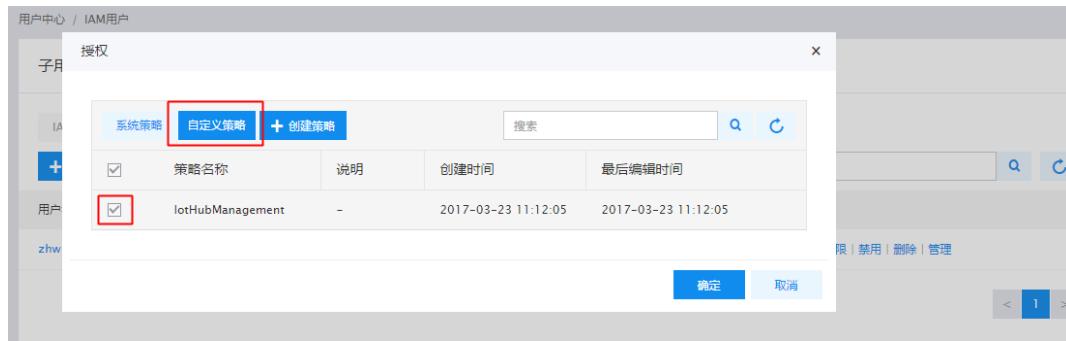
子用户管理列表

| 用户名 | 百度账号 | 说明 | 状态 | 创建时间 | 操作 |
|------------|-------------------------|----|------|---------------------|---|
| zhw | 未绑定 去绑定 | - | ● 活跃 | 2017-03-23 10:50:25 | 添加权限 禁用 删除 管理 |
| [REDACTED] | [REDACTED] | - | ● 活跃 | 2016-05-31 17:49:13 | 添加权限 禁用 删除 管理 |
| [REDACTED] | [REDACTED] | - | ● 活跃 | 2016-05-31 17:46:37 | 添加权限 禁用 删除 管理 |
| [REDACTED] | [REDACTED] | - | ● 活跃 | 2016-04-05 14:52:54 | 添加权限 禁用 删除 管理 |

4. 子用户添加完成后需要对子用户进行策略管理，点击“策略管理” > “创建策略”，自定义物接入子账户权限。



5. 返回子用户管理列表，点击“添加权限”，在弹出对话框中点击“自定义策略”，选择步骤4中创建的策略。



6. 子用户的账户和权限添加完成后，用户可以直接通过子用户管理列表的“IAM用户登录链接”登录主账号的物接入资源，登录成功后用户根据设定的权限享有对主账户资源的操作和查看权限。

第5章 最佳实践

5.1 基于物影子快速搭建物联网应用

5.1.1 简介

本文档介绍了基于物接入快速搭建一个物联网应用的方式，帮助用户理解和使用物接入。阅读本示例前请先熟悉快速入门文档。

应用场景

一个智能设备（如智能家居设备）生产厂商，智能设备部署到各地，需要按照一定的频率回传状态数据。

应用端需要收到这些数据进行下一步消费。同时这些数据也需要用一个手机APP端实时查看。

5.1.2 云端创建设备影子

通过控制台或API创建物影子，每一个物影子就是一个连接到云端的实体设备。创建物影子时，每一个物影子都会生成一个连接用户名和密码，将该用户名密码的配置到设备端的连接程序中。

- 如果使用MQTT开源SDK，则每个物影子有独立的MQTT连接，该用户名密码即作为MQTT连接的用户名密码，影子名称作为clientID。
- 如果使用天工Edge SDK（下载地址：<https://github.com/baidu/iot-edge-c-sdk>），可以在如下路径中找到物接入的sample

[iot-edge-c-sdk](#) / [iothub_client](#) / [samples](#) / [iotdm_client_sample](#) / [iotdm_client_sample.c](#)

并将如下位置，替换成要每个物影子生成的相对应的用户名、密码。

```

// 2. "ssl://xxxxxx.mqtt.iot.xx.baidubce.com:1884"
#define ADDRESS "tcp://xxxxxx.mqtt.iot.xx.baidubce.com:1883"

// The device name you created in device management service.
#define DEVICE "xxxxxx"

// The username you can find on the device connection configuration web,
// and the format is like "xxxxxx/xxxxx"
#define USERNAME "xxxxxx/xxxxxx"

// The key (password) you can find on the device connection configuration web.
#define PASSWORD "xxxxxx"

```

5.1.3 设备端发消息更新云端的物影子

如果使用物接入SDK，物接入设备型SDK已经封装好了更新物影子的函数 `iotdm_client_update_shadow_with`。更新成功后，云端的物影子即更新了相应的值。对于物影子的操作，物接入不仅提供更新状态到物影子，还可以

- 从物影子获取最新状态。
- 通过物影子反控设备。
- 获取本次物影子与上一次的变化。
- 获取物影子的实时快照。

物接入设备型SDK都提供相应操作的封装函数，可以直接调用。物影子操作的相关介绍请参考：[物影子操作](#)

如果使用MQTT开源SDK，设备端按照生成的用户名密码连接云端后，可以向主题 `$baidu/iot/shadow/{thingName}/update` 发布消息。需要符合物影子的schema，具体详情请见：[操作指南](#)

其他功能，如从物影子获取最新状态、获取本次物影子与上一次的变化、获取物影子实时快照等操作也有相对应的mqtt主题，详情请参见[操作指南](#)

注意： 物影子名称是主题名称的一部分，如 `$baidu/iot/shadow/{thingName}/update`，需要更新到设备端的代码中。物影子生成的用户名密码，只能对该物影子进行操作，即只对该物影子的一系列主题有发布或订阅的权限。

5.1.4 设备端与云端通信（非更新物影子）

如果设备端除了更新物影子以外，还有其他的数据信息需要上传到云端。物接入提供自定义的主题，供用户使用。

- define类主题，形如 `$baidu/iot/define/{thingName}/#`，用户可以在 `#{}` 处自定义字段，如 `$baidu/iot/define/{thingName}/abc/sys` 等。该主题的特点

是，主题名中含有thingName（物影子名称）。物影子生成的用户名密码只能向该物影子发布和订阅消息，不能对其他物影子的define类主题发布和订阅消息。

- general类主题，形如 \$baidu/iot/general/#，用户可以在『#』处自定义字段，如\$baidu/iot/general/alarm等。该主题的特点是，任意一个物影子的连接或者权限组的连接，都对该主题拥有订阅或发布权限，可以用于不同设备之间的通信。

5.1.5 应用服务消费设备的数据

用户的应用服务程序可以通过物接入的API接口，通过HTTP来获取设备的数据，如获取物影子实时数据等。

如果用户的应用服务程序需要实时订阅到物影子的变化，则可以通过MQTT的连接获取。

- 物影子生成的用户名密码，同样可以在应用程序中连接到物接入，订阅与该物相关的主题，如\$baidu/iot/shadow/{thingName}/update/accepted或\$baidu/iot/define/{thingName}/abc等，就可以实时订阅到设备端发到云端的消息。
- 应用服务程序使用MQTT连接时需要注意：clientID不能是物影子名称。因为设备端的MQTT连接中的clientID是物影子名称，并以此来作为设备是否在线的判断标准，详情请参见：[设备在线状态](#)
- 对于一个应用需要同时订阅多个设备的数据的情况，可以利用物接入设备型项目的权限管理来完成。物接入会为每一个权限组分配一个用户名密码，权限组可以添加多个设备，则这个权限组即拥有这多个设备的权限，包括物影子操作（shadow类主题）和物的自定义主题（define类主题）。一个业务应用用此权限组的用户名密码建立MQTT连接，即可以实时订阅该权限组里的设备的信息。

5.2 ST物联网开发套件连接例程

5.2.1 操作准备

[教程概述](#)

基于ST的B-L475E-IOT01物联网开发套件与百度云天工连接的例程。

- ST和百度云天工基于ST推出的一款物联网开发套件，提供了连接百度云天工的软件扩展包，内含连接到百度云天工物接入（IOT Hub），进行消息订阅和发布的应用程序。
- 用户基于ST的B-L475E-IOT01A与本教程可以快速上手搭建自己的应用程序。

[硬件概述](#)

B-L475E-IOT01A探索套件提供了：

1. 采用ARM Cortex内核的STM32L4系列MCU

2. 存储

- 1MB闪存和128KBSRAM
- 64Mbit Quad-SPI闪存

3. 模块

- SPBTLE-RF蓝牙模块，兼容BLE 4.1
- SPSGRF-915次吉赫兹RF模块(另外还有一款同型号的支持的是868MHz)
- ISM43362-M3G-L44 WiFi模块，支持802.11 b/g/n通信
- M24SR NFC模块
- 2路数字microphone(MP34DT01)
- 电容式数字传感器HTS221，用于测量相对湿度和温度
- LIS3MDL，高性能三轴磁力计
- LSM6DSL三轴陀螺仪
- LPS22HB压力计
- VL53L0X姿态传感器

4. 接口

- USB OTG接口
- PMOD接口，Arduino UNO V3接口
- 灵活的供电接口设计

配合这个套件，ST和百度共同推出了连接百度天工的软件扩展包。提供了连接到百度IoT hub, 进行消息订阅和发布的应用例程，客户可以基于这个例程快速的上手搭建自己的应用程序。

从[ST的官网](#)可以下载到关于该开发套件的详细介绍文档和硬件设计资料。

硬件环境

B-L475E-IOT01A板子上已经集成了WIFI模块和ST-LINK，所以不需要额外的模块和调试工具，只需要一根micro的USB线给板子提供电源就可以开始使用了。当然，如果需要更新程序的话，还需要一台带USB接口的电脑。

硬件准备：

- 一块B-L475E-IOT01A开发板（集成了WIFI模块和ST-LINK）。
- 一根microUSB接口的线（给板子供电，提供调试和程序下载接口）。
- 一个WIFI热点（WIFI路由器或者手机热点）。
- 一台电脑（编译和烧录程序，配置WIFI密码）。

软件环境

需要的软件包括：

- 免费的SW4STM32 IDE (Windows, Linux, macOS)。
- 串口调试工具Tera Term。
- 百度云天工物接入资源。
- 基于B-L475E-IOT01A板子的源代码。下载地址：<https://github.com/baidu/iot-edge-sdk-samples>

通过免费的SW4STM32，可以修改，编译，下载和调试程序。下面是SW4STM32的下载页面链接，不过需要先注册登录后才能下载：<http://www.openstm32.org/Downloading%2Bthe%2BSYSTEM%2BWorkbench%2Bfor%2BSTM32%2Binstaller>

还有更多的文档资料可以在这里找到：<http://www.openstm32.org/Documentation>

通过Tera Term可以向板子配置WIFI用户名和密码，并且显示程序运行时的信息。

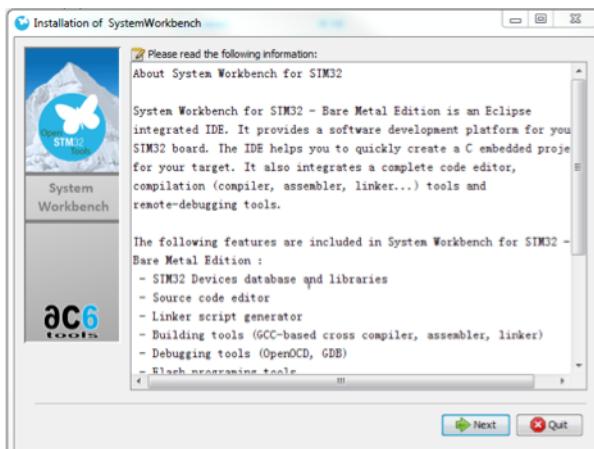
下面将具体介绍，如何使用SW4STM32来编译，下载程序到开发板。以及Tera Term的使用方法。

5.2.2 SW4STM32使用说明

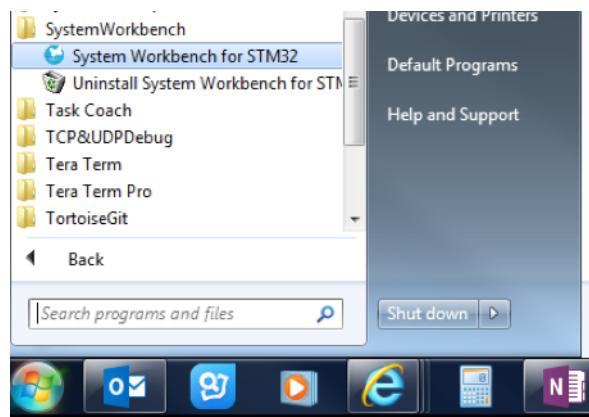
SW4STM32下载安装

从SW4STM32官网下载安装程序，官网上提供了Windows, Mac OS/X和Linux的版本。选择合适的版本下载安装。本文档以Windows版本为例进行介绍。SW4STM32包含一系列的Eclipse插件，可以安装在现有的Eclipse上，也可以通过下载独立的安装程序来安装。

双击下载的安装程序，按照提示进行安装。详细的安装说明见官网：<http://www.openstm32.org/Installing%2BSYSTEM%2BWorkbench%2Bfor%2BSTM32>

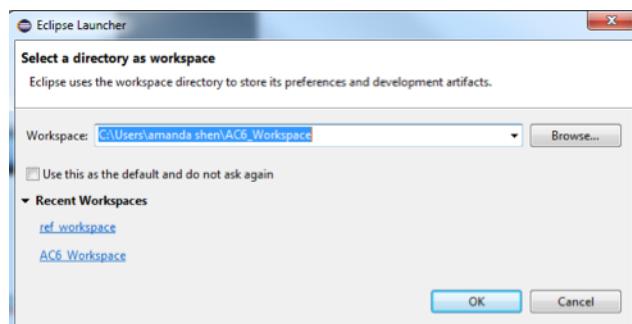


安装完成后，在开始菜单里找到SystemWorkbench for STM32，打开。

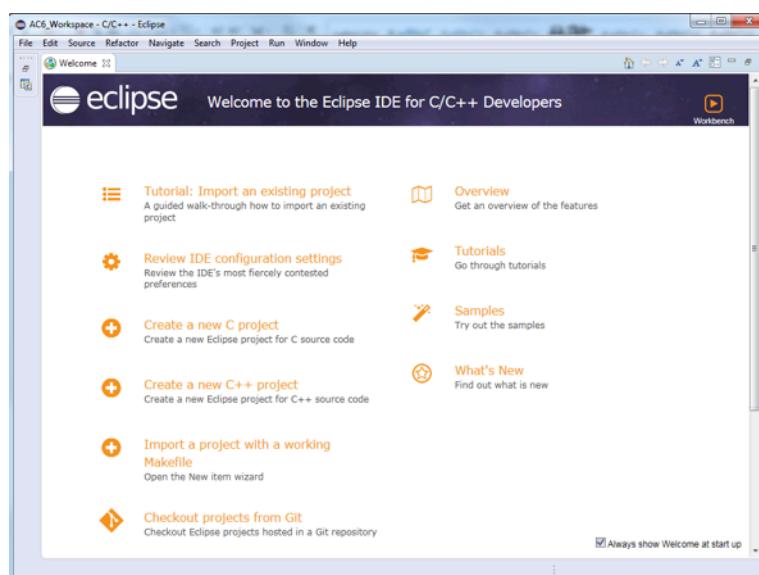


导入工程

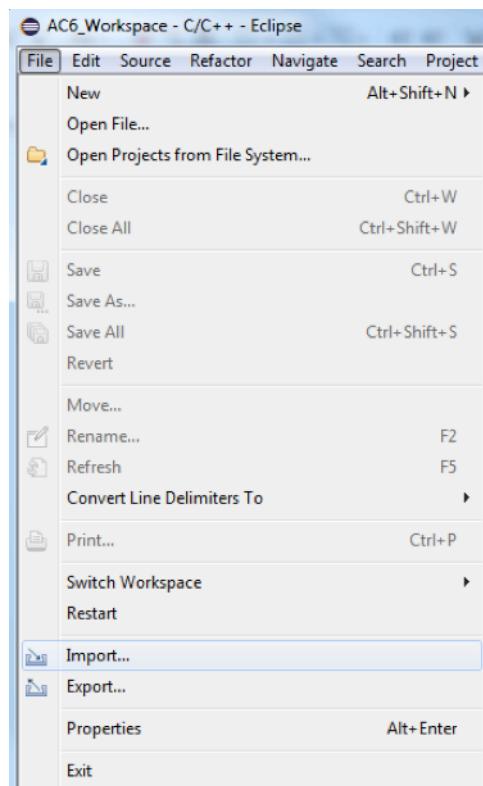
打开SW4STM32后，首先会弹出下面窗口，让你选择一个Workspace。如果是第一次使用，可以通过“Browse”按钮设置workspace的位置。如果之前已经设置过，就可以在下拉菜单中选择一个。



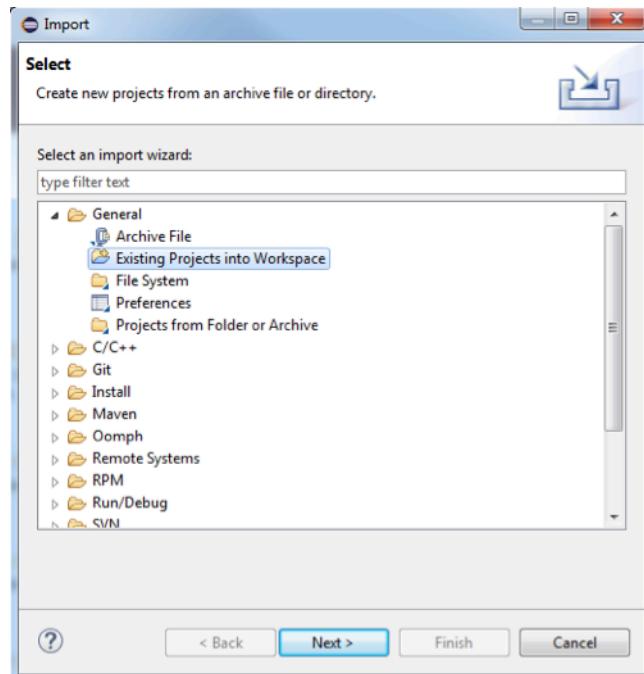
点击“OK”，就可以看到SW4STM32的欢迎界面了。



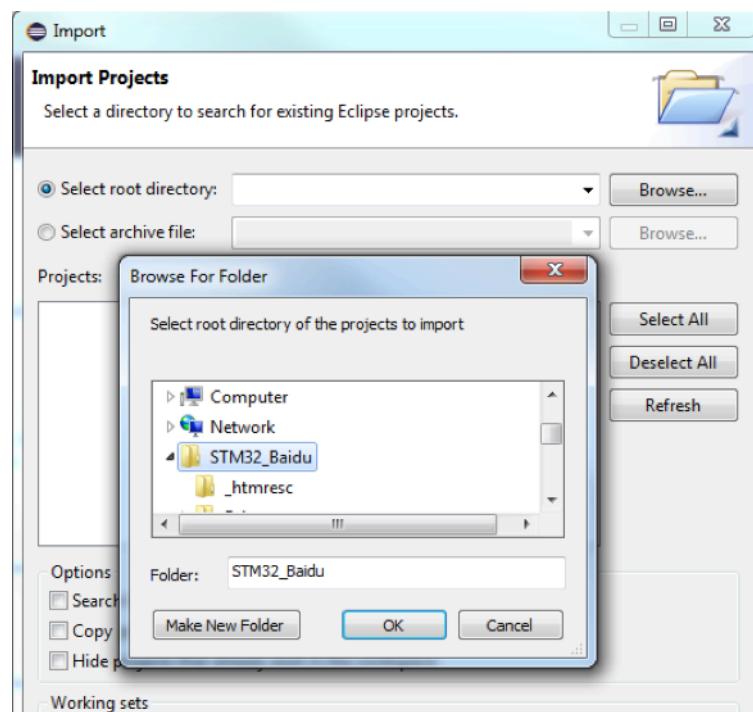
下面通过File->Import菜单导入已有的工程。



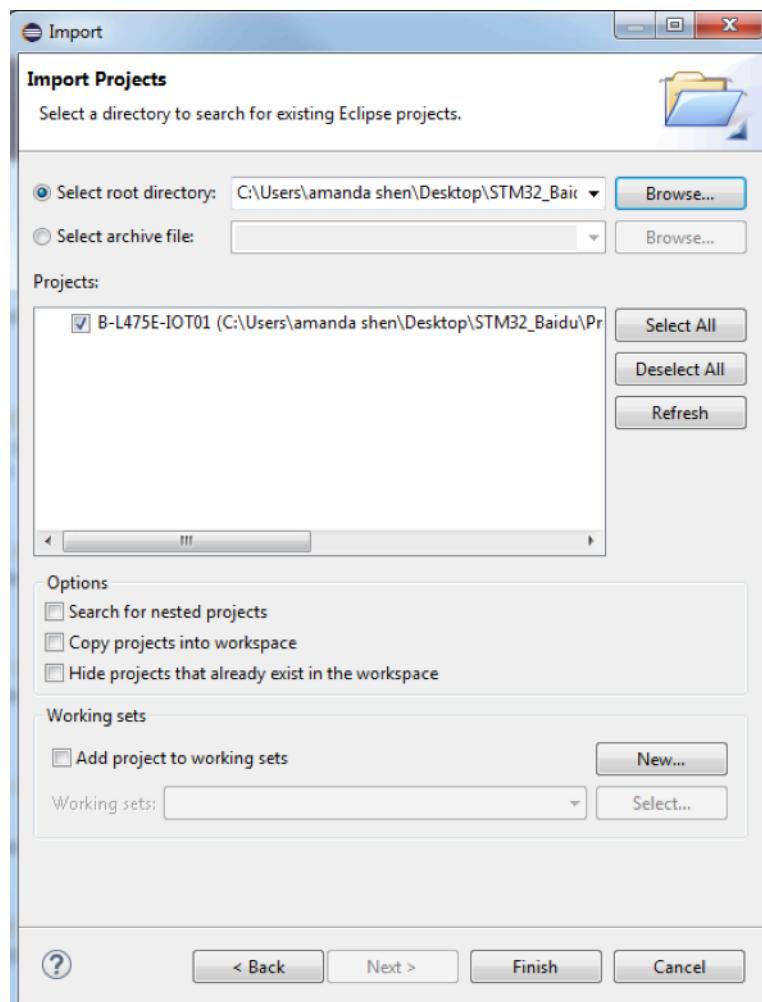
在弹出的窗口里，选择“Existing Projects into Workspace”，然后点击“Next”。



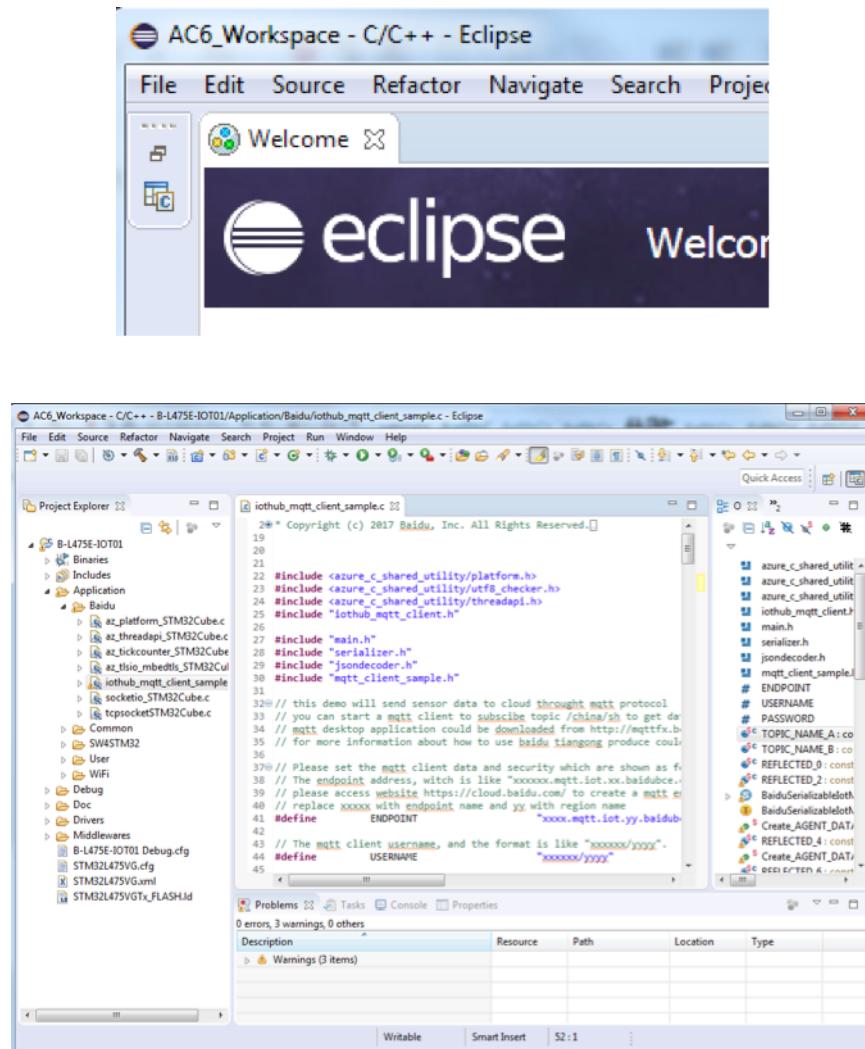
接下来，选择源文件所在的位置。



在Projects的窗口会显示已存在的工程，选择该工程。点击“Finish”按钮。



工程就已经被导入进来了，关闭Welcome欢迎窗口或者点击左上角的“Restore”按钮，就可以看到整个工程的界面了。



5.2.3 创建物接入项目

请参考[快速入门](#)。设备型和数据型项目均适用本例。

5.2.4 修改源码内Baidu IoT Hub连接信息

创建好百度云IOT Hub的服务，之前将示例工程导入到SW4STM32后，现在可以进行程序的修改，编译和下载。

例程中提供的MQTT连接例程主要在iothub_mqtt_client_sample.c这个文件中实现。如果您使用的设备型项目，可基于iotdm_client_sample.c例程更好地使用。

现在，将其中关于MQTT服务器地址，实例，用户名，密码还有主题（endpoint、username、password、topic）修改成之前的步骤中在云端创建的相关内容。而后进行编译。

将获得的MQTT服务器地址，用户名，密码还有主题填写到下面代码对应的位置。主题的属性，可以在云端设置为可以发布和订阅消息。



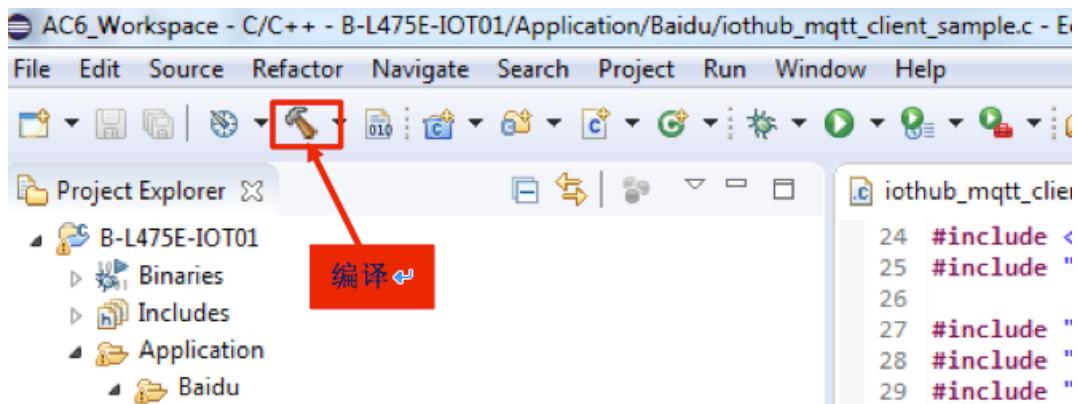
```

24 #include <azure_c_shared_utility/threadapi.h>
25 #include "iothub_mqtt_client.h"
26
27 #include "main.h"
28 #include "serializer.h"
29 #include "jsondecoder.h"
30 #include "mqtt_client_sample.h"
31
32 // this demo will send sensor data to cloud through mqtt protocol
33 // you can start a mqtt client to subscribe topic /china/sh to get data send by board.
34 // mqtt desktop application could be downloaded from http://mqtfx.bceapp.com/
35 // for more information about how to use baidu tiangong produce could be found at https://cloud.baidu.com/doc/IOT/Qu
36
37 // Please set the mqtt client data and security which are shown as follow.
38 // The endpoint address, which is like "xxxxxx.mqtt.iot.xx.baidubce.com".
39 // please access website https://cloud.baidu.com/ to create a mqtt endpoint
40 // replace XXXXXX with endpoint name and yy with region name
41 #define ENDPOINT "xxxx.mqtt.iot.yy.baidubce.com"
42
43 // The mqtt client username, and the format is like "xxxxxx/yyyy".
44 #define USERNAME "xxxxxx/yyyy"
45
46 // The key (password) of mqtt client.
47 #define PASSWORD "XXXXXXXXXXXXXX"
48
49 static const char* TOPIC_NAME_A = "topicNameA";
50 static const char* TOPIC_NAME_B = "topicNameB";
51
--
```

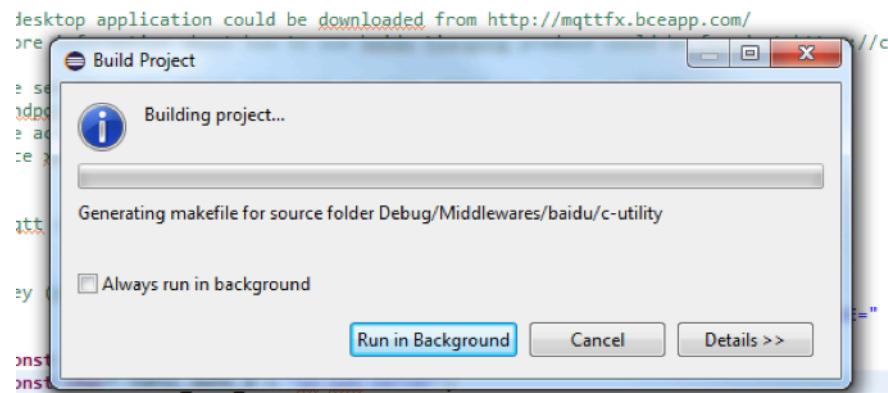
5.2.5 编译下载程序

接下来，就可以点击对应的按钮进行编译和下载了。本例程已经做好配置，直接点击对应的按钮就可以执行。关于编译和调试的配置说明，可以参考[官网的文档](#)

编译



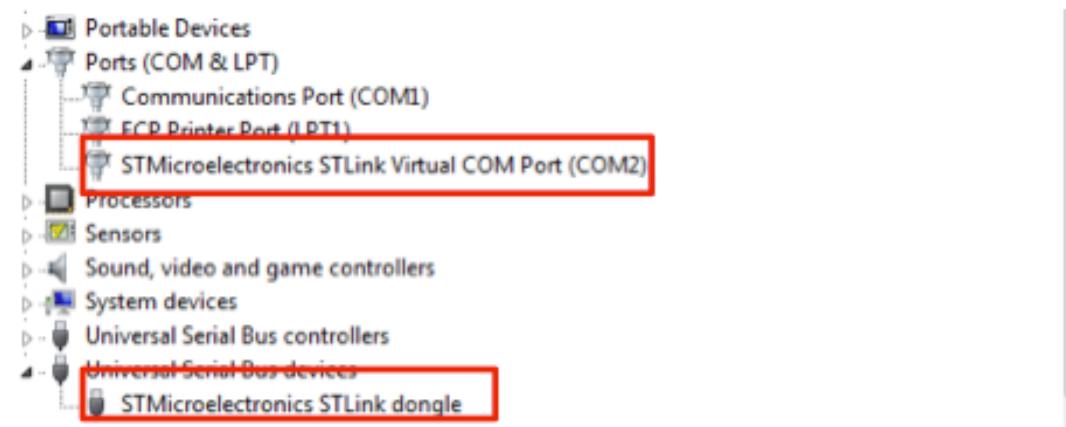
弹出窗口会显示编译进度。



下载

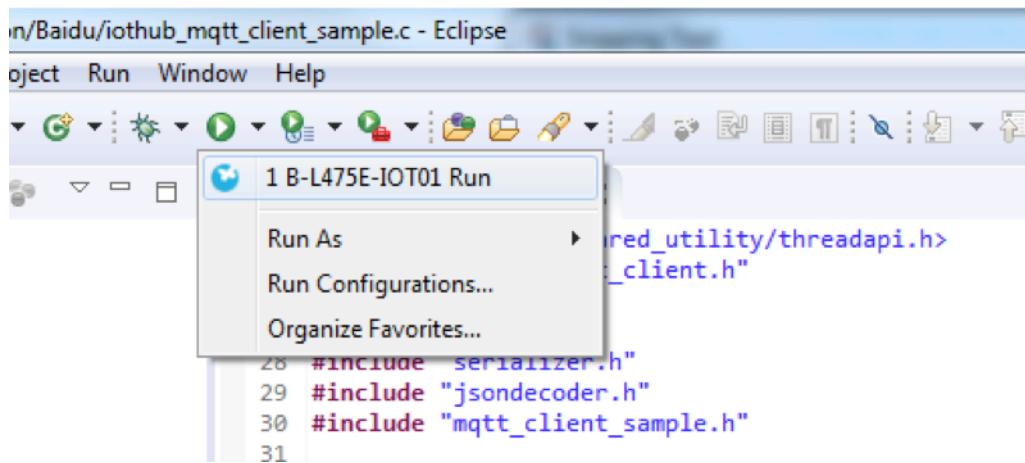
下载程序之前，请确保已经用USB线将B-L475E-IOT01A与电脑相连。

- 如果下载出现问题，请确认ST-LINK的驱动已经正确安装。
- 如果ST-LINK的驱动正确安装，可以在电脑的设备管理器中看到如下设备：



一般情况下当插入板子时，会自动安装驱动。如果没有看到这两个设备，请重新安装驱动。驱动下载地址：http://www.st.com/content/st_com/en/products/development-tools/software-development-tools/stm32-software-development-tools/stm32-utilities/stsw-link009.html

通过“Run”按钮开始下载程序，操作见下图

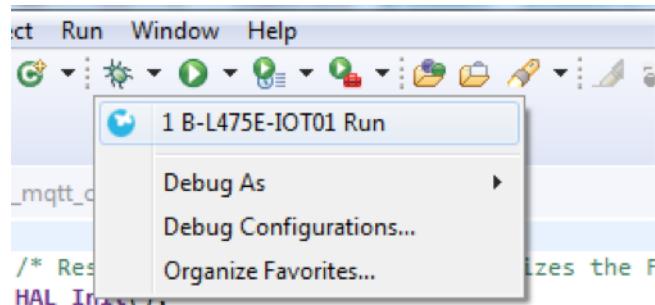


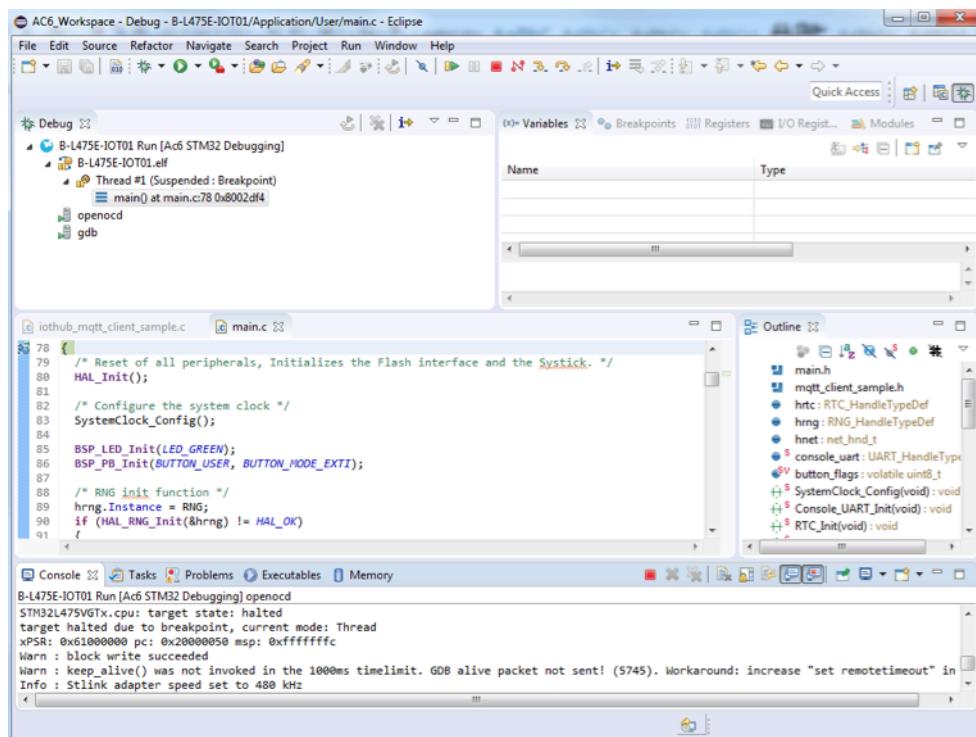
下方的Console窗口可以看到下载过程信息提示。

```
<terminated> B-L475E-IOT01 Run [Ac6 STM32 Debugging] openocd
target halted due to breakpoint, current mode: Thread
xPSR: 0x61000000 pc: 0x2000002e msp: 0x20018000
STM32L475VGTx.cpu: target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x61000000 pc: 0x2000002e msp: 0x20018000
verified 281792 bytes in 9.115912s (30.188 KiB/s)
** Verified OK **
** Resetting Target ***
Info : Stlink adapter speed set to 480 kHz
adapter speed: 480 kHz
shutdown command invoked
```

调试程序

通过Debug窗口启动调试。见下图：



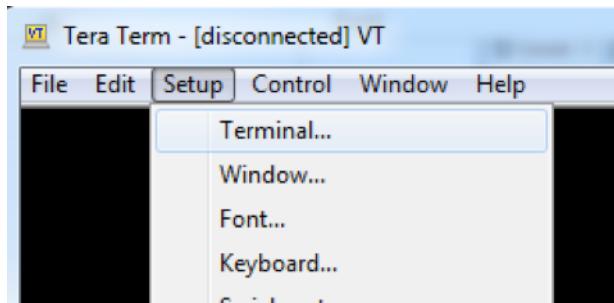


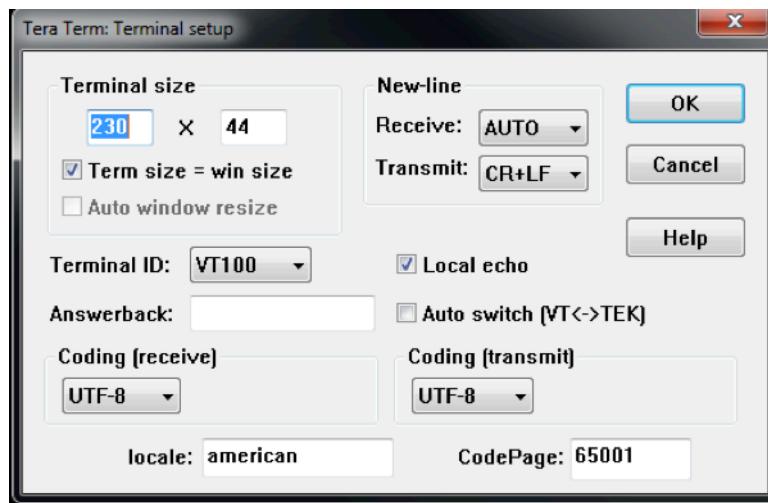
5.2.6 Tera Term使用说明

程序下载到开发板后，复位开始运行。通过TeraTerm可以看到程序执行过程中的信息。并且WIFI的用户名和密码也是通过Tera Term进行配置的。

第一次使用Tera Term时，请参考下面的步骤进行配置。

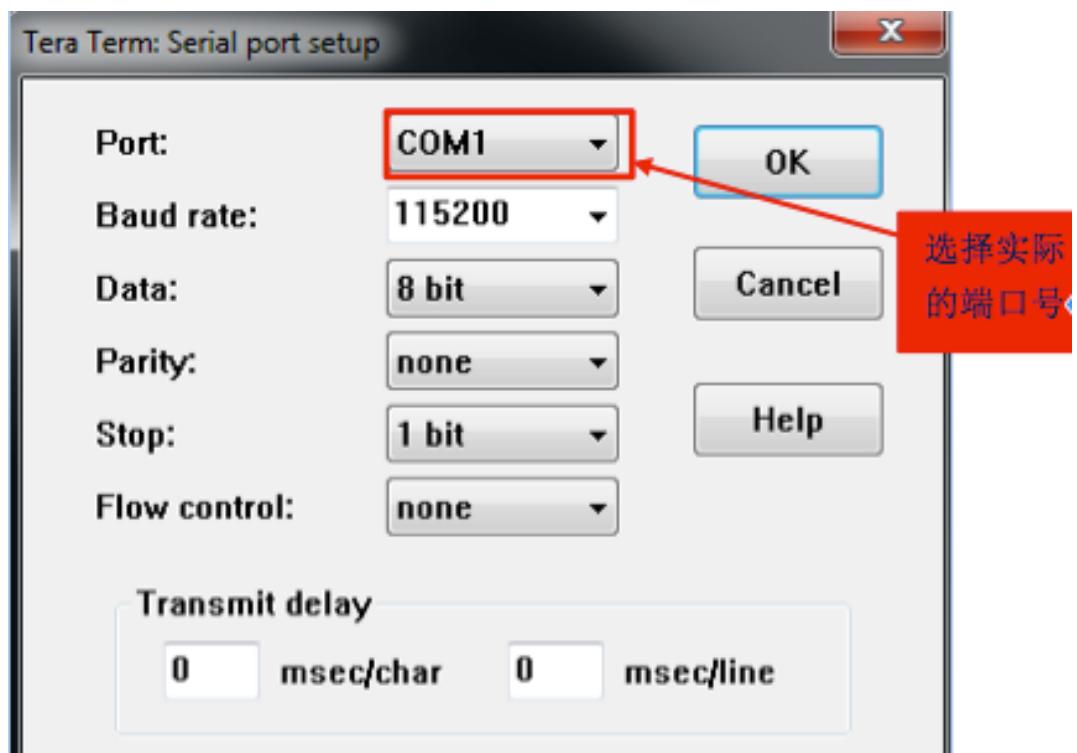
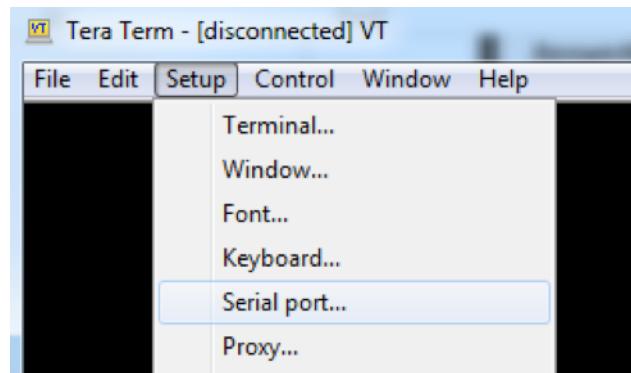
终端设置





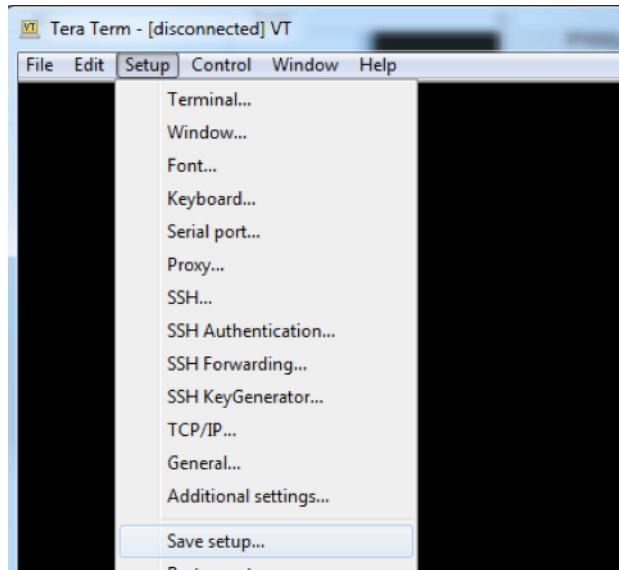
串口通信设置

串口配置为：115200波特率，8位数据位，无奇偶校验，1位停止位。



保存设置

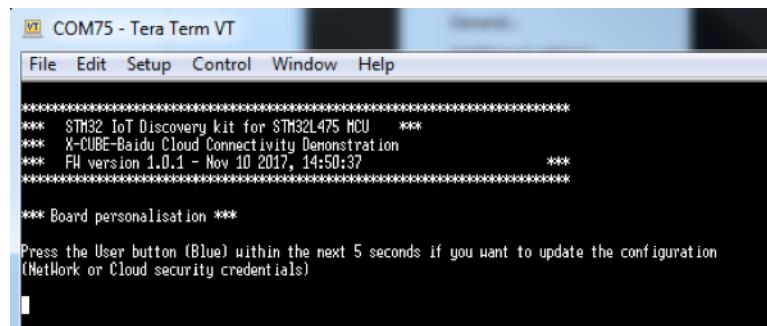
不要忘记将前面的设置保存。



5.2.7 运行程序

将B-L475E-IOT01A开发板通过USB插到电脑，打开Tera Term（选择对应的COM口）。按黑色的按键复位开发板。

在看到提示“Press the User button (Blue) within the next 5 seconds if you want to update the configuration”后5秒内按下板子上蓝色的按键。（如果之前没有输入过wifi信息，会直接提示输入SSID等内容）



提示是否要更新WIFI的设置。输入“y”，然后依照提示输入wifi的用户名，加密方式和密码。

```

*** STH32 IoT Discovery kit for STM32L475 MCU ***
*** X-CUBE-Baidu Cloud Connectivity Demonstration ***
*** FW version 1.0.1 - Nov 10 2017, 14:50:37 ***
*** Board personalisation ***

Press the User button (Blue) within the next 5 seconds if you want to update the configuration
(NetWork or Cloud security credentials)

Do you want to configure the Wifi credentials ? (y/n)
y
Enter SSID: [REDACTED]

```

```

*** STH32 IoT Discovery kit for STM32L475 MCU ***
*** X-CUBE-Baidu Cloud Connectivity Demonstration ***
*** FW version 1.0.1 - Nov 10 2017, 14:50:37 ***
*** Board personalisation ***

Press the User button (Blue) within the next 5 seconds if you want to update the configuration
(NetWork or Cloud security credentials)

Do you want to configure the Wifi credentials ? (y/n)
y
Enter SSID: iPhonex
You have entered iPhonex as the ssid.

Enter Security Mode (0 - Open, 1 - HEP, 2 - HPA, 3 - HPA2):3
You have entered 3 as the security mode.

Enter password: [REDACTED]

```

接下来，板子会连接到WIFI路由器，然后自动连接到之前在程序中设定好的MQTT服务器，向设定好的主题订阅消息/发布消息。发布的消息就是板子上集成的传感器数据。可以通过百度云端的客户端或者MQTT.fx订阅同样的主题来查看消息，或者发送消息给B-L475E-IOT01A开发板。

```

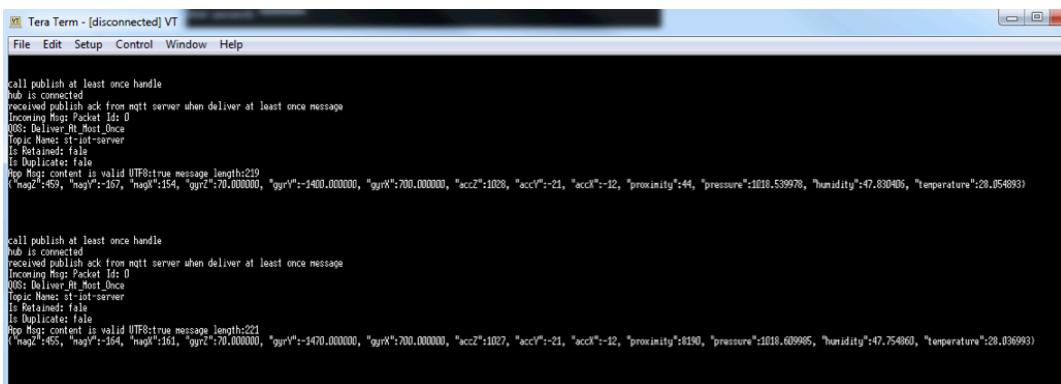
Tera Term - [disconnected] VT
File Edit Setup Control Window Help
You have entered 3 as the security mode.
Enter password: 88888888

*** WIFI connection ***
Initializing the WiFi module
Module initialized successfully: Inventek eS-WIFI 10M43362-M36-L44-SFI (3.5.2.3.BET9
Retrieving the WiFi module MAC address: c4:7f:51:03:12:9d

Connecting to AP: iPhonex Attempt 1/3 ...
Connected to AP: iPhonex
Mac address: c4:7f:51:03:12:9d
Retrieving the IP address,
IP address: 172.20.10.2
Setting the RTC from the network time.
Error: TimeSet Jan 1 00:33:53 2000 File:C:/Users/andrea.shen/Desktop/STM32_Baidu/Middlewares/Third_Party/baidu/iothub_client/src/iothub_mqtt_client.c Func:CreateRetryLogic Line:121 Not implemented choosing default

Incoming Msg: Packet Id: 31498
008: Deliver_ Exactly_Once
Topic Name: st-iot-server
Is Publish: false
Is Duplicate: false
Pop Hop content is valid UTF8:true message length:261
state": {
  "reported": {
    "temperature": 23.00,
    "humidity": 50.20,
    "pressure": 1039.50,
    "proximity": 0,
    "acc_x": -14, "acc_y": -13, "acc_z": 1033,
    "gyr_x": 980, "gyr_y": -6280, "gyr_z": 70,
    "mag_x": 334, "mag_y": 26, "mag_z": 123
  }
}

```



The screenshot shows the Tera Term terminal window with the title "Tera Term - [disconnected] VT". The menu bar includes File, Edit, Setup, Control, Window, and Help. The terminal output displays two MQTT publish messages. The first message is for topic "st/iot-server" with QoS 0 and delivery count 1. It contains fields: "accX":1028, "accY":-21, "accZ":-12, "proximity":44, "pressure":1018.539979, "humidity":47.830406, "temperature":28.054893. The second message is similar but has a different timestamp and slightly different values for proximity and pressure.

```
call publish at least once handle
hub is connected
received publish ack from mqtt server when deliver at least once message
Message topic: Packet Id: 0
QOS: Deliver_R_Most_Once
Topic Name: st/iot-server
Is Retained: false
Is Duplicate: false
App Msg content is valid UTF8:true message length:219
{"accX":1028, "accY":-21, "accZ":-12, "proximity":44, "pressure":1018.539979, "humidity":47.830406, "temperature":28.054893}

call publish at least once handle
hub is connected
received publish ack from mqtt server when deliver at least once message
Message topic: Packet Id: 0
QOS: Deliver_R_Most_Once
Topic Name: st/iot-server
Is Retained: false
Is Duplicate: false
App Msg content is valid UTF8:true message length:221
{"accX":1028, "accY":-21, "accZ":-12, "proximity":44, "pressure":1018.609985, "humidity":47.75406, "temperature":28.036093}
```

5.2.8 参考文档

1. UM2153-Discovery kit for IoT node, multi-channel communication with STM32L4
2. [物接入IoT Hub产品文档](<https://cloud.baidu.com/doc/IOT/ProductDescription.html>)

第6章 API参考(数据型)

6.1 目录

- 介绍
 - 简介
 - 调用方式
 - * 概述
 - * 通用约定
 - * 公共请求头
 - * 公共响应头
 - * 响应状态码
 - * 通用错误返回格式
 - * 公共错误码
 - * 签名认证
 - * 签名生成算法
 - 多区域选择
- 认证
 - 认证
 - 鉴权
- 动作
 - 给一个Thing添加一个Principal
 - 从一个Thing移除一个Principal
 - 给一个Principal添加一个Policy
 - 从一个Principal移除一个Policy
- Endpoint
 - 获取endpoint列表
 - 获取指定的endpoint信息
 - 创建endpoint
 - 删除endpoint
- Thing
 - 获取thing列表
 - 获取指定的thing信息
 - 创建thing

- 删除thing
- Principal
 - 获取principal列表
 - 获取指定的principal信息
 - 创建principal
 - 重新生成密钥
 - 删除principal
- Policy
 - 获取policy列表
 - 获取指定的policy信息
 - 创建policy
 - 删除policy
- Permission
 - 获取policy下所有topic信息
 - 获取指定topic的信息
 - 在policy下设置topic
 - 更新已有的topic设置
 - 删除已有的topic
- Client
 - 获取指定MQTT客户端在线状态
 - 获取所有MQTT客户端在线状态
 - Publish Message
- 使用量
 - 获取当前账单月内使用量
 - 获取当前账单月内特定实例的使用量
 - 查询特定实例某个时间段内的使用量

6.2 介绍

6.2.1 简介

物接入是全托管的云服务，可以在智能设备与云端之间建立安全的双向连接，并通过主流的物联网协议（如MQTT）通讯，实现从设备端到云端以及从云端到设备端的安全稳定的消息传输。IoT Hub API主要提供Thing、Principal、Policy和Topic的创建、删除、查询等功能，以Restful API的形式提供。

6.2.2 调用方式

概述 物接入的设计采用了Restful风格，每个API功能（也可以称之为资源）都使用URI (Universal Resource Identifier) 来唯一确定。对资源的请求方式是通过向资源对应的URI发送标准的HTTP请求，比如GET、PUT、POST等，同时，请求需要遵守签名算法，并包含约定的请求参数。

通用约定

- 所有编码都采用UTF-8
- 日期格式采用yyyy-MM-dd方式，如2015-08-10
- 时间格式采用UTC格式：yyyy-MM-ddTHH:mm:ssZ, 如2015-08-20T01:24:32Z
- Content-type为application/json; charset=UTF-8
 - object类型的key必须使用双引号（“ ”）括起来
 - object类型的key必须使用lowerCamelCase表示

| 头域 (Header) | 是否必须 | 说明 |
|---------------|------|-------------------------------------|
| Authorization | 必须 | 包含Access Key与请求签名 |
| Host | 必须 | 包含API的域名 |
| Content-Type | 可选 | application/ json; charset=utf-8 |

公共请求头

| 头域 (Header) | 说明 |
|------------------|---|
| Content-Type | 只支持JSON格式，application/ json; charset=utf-8 |
| x-bce-request-id | 后端生成，并自动设置到响应头域中 |

公共响应头

响应状态码 返回的响应状态码遵循[RFC 2616 section 6.1.1](#)

- 1xx: Informational - Request received, continuing process.
- 2xx: Success - The action was successfully received, understood, and accepted.
- 3xx: Redirection - Further action must be taken in order to complete the request.
- 4xx: Client Error - The request contains bad syntax or cannot be fulfilled.
- 5xx: Server Error - The server failed to fulfill an apparently valid request.

通用错误返回格式 当调用接口出错时，将返回通用的错误格式。Http的返回状态码为4xx或5xx，返回的消息体将包括全局唯一的请求、错误代码以及错误信息。调用方可根据错误码以及错误信息定位问题，当无法定位到错误原因时，可以发工单联系百度技术人员，并提供requestId以便于快速地帮助您解决问题。

消息体定义

| 参数名 | 类型 | 说明 |
|-----------|--------|---------|
| requestId | String | 请求的唯一标识 |
| code | String | 错误类型代码 |
| message | String | 错误的信息说明 |

错误返回示例

```
{
  "requestId": "47e0ef1a-9bf2-11e1-9279-0100e8cf109a",
  "code": "NoSuchKey",
  "message": "The resource you requested does not exist"
}
```

| Code | Message | HTTP Status Code | 说明 |
|------------------------|--|------------------|---|
| BceValidationException | [param]: [param]=[Validation criteria] | 400 | 无效的[param]参数 |
| MoneyNotEnough | Money not enough to complete the current request | 400 | 余额不足以完成当前的请求操作 |
| SignatureDoesNotMatch | The request signature we calculated does not match the signature you provided. Check your Secret Access Key and signing method. Consult the service doc- umentation for details | 400 | Authorization 头域 中附带的签名和服 务端验证不一致 |
| InvalidAccessKeyId | The Access Key ID you provided does not exist in our records | 403 | Access Key ID不存 在 |

| Code | Message | HTTP Status Code | 说明 |
|-----------------------|---------------------------------|------------------|----------|
| ServiceInternal Error | Service internal error occurred | 500 | 内部服务发生错误 |

公共错误码

签名认证 物接入API会对每个访问的请求进行身份认证，以保障用户的安全。安全认证采用Access Key与请求签名机制。Access Key由Access Key ID和Secret Access Key组成，均为字符串，由百度云官方颁发给用户。其中Access Key ID用于标识用户身份，Access Key Secret 是用于加密签名字符串和服务器端验证签名字符串的密钥，必须严格保密。

对于每个HTTP请求，用户需要使用下文所描述的方式生成一个签名字符串，并将认证字符串放在HTTP请求的Authorization头域里。

签名字符串格式

bce-auth-v{version}/{accessKeyId}/{timestamp}/{expireTime}/{signedHeaders}/{signature}

其中：

- version是正整数，目前取值为1。
- timestamp是生成签名时的时间。时间格式符合[通用约定](#)。
- expireTime表示签名有效期限，单位为秒，从timestamp所指定的时间开始计算。
- signedHeaders是签名算法中涉及到的头域列表。头域名字之间用分号（;）分隔，如host;x-bce-date。列表按照字典序排列。当signedHeaders为空时表示取默认值。
- signature是256位签名的十六进制表示，由64个小写字母组成，生成方式由如下[签名生成算法](#)给出。

签名生成算法 有关签名生成算法的具体介绍，请参看[鉴权认证机制](#)。

6.2.3 多区域选择

“华南-广州”区域

- 区域的API地址为：iot.gz.baidubce.com
- Publish Message API地址：api.mqtt.iot.gz.baidubce.com

“华北-北京”区域

- 区域的API地址为：iot.bj.baidubce.com
- Publish Message API地址：api.mqtt.iot.bj.baidubce.com

6.3 认证

6.3.1 认证

| 相对URI | HTTP 方式 |
|--------------------------------|---------|
| /v1/auth/authenticate/password | POST |

请求参数

| 名称 | 类型 | 是否必选 | 含义 |
|----------|--------|------|---------------------|
| password | String | Y | 身份(principal)的密钥/ak |
| username | String | Y | thing的username |

返回参数

| 名称 | 类型 | 含义 |
|---------------|--------|-------------------------------|
| endpointName | String | thing 所属的 endpoint 的 Name |
| endpointUuid | String | thing 所属的 endpoint 的 Uuid |
| principalUuid | String | thing在这次认证中所用的 principal的uuid |

6.3.2 鉴权

| 相对URI | HTTP 方式 |
|--------------------|---------|
| /v1/auth/authorize | POST |

请求参数

| 名称 | 类型 | 是否必选 | 含义 |
|---------------|--------|------|----------------------|
| principalUuid | String | Y | 需要鉴权的 principal的Uuid |

| 名称 | 类型 | 是否必选 | 含义 |
|--------|--------|------|---|
| action | ENUM | Y | 动作。目前定义了5个操作(和apollo中对应):CONNECT:主体与broker建立连接CREATE:主体创建TopicSEND:主体向特定Topic发送messageRECEIVE:主体从特定Topic接收messageCONSUME: clean session=false时替代receive |
| topic | String | N | 这次操作对应的topic。SEND, RECEIVE, CONSUME操作必须要传Topic |

[返回参数](#)

无特殊返回参数。

6.4 动作

6.4.1 给一个Thing添加一个Principal

| | |
|-----------------------------------|---------|
| 相对URI | HTTP 方式 |
| /v1/action/attach-thing-principal | POST |

[请求参数](#)

| 名称 | 类型 | 是否必选 | 含义 |
|---------------|--------|------|-------------------------|
| endpointName | String | Y | thing 所属的 endpoint的Name |
| thingName | String | Y | thing的名称 |
| principalName | String | Y | principal的名称 |

[返回参数](#)

无特殊返回参数。

[请求示例](#)

```
POST /v1/action/attach-thing-principal HTTP/1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/plain; charset=UTF-8
{
    "principalName": "principal-1"
    "endpointName": "endpoint-1"
    "thingName": "thing-1"
}
```

[返回示例](#)

```
HTTP/1.1 201 Created
x-bce-request-id: fecfa1bd-aa94-42b0-99b1-b33e4b1b1c88
Content-Type: application/json; charset=UTF-8
{
    "message": "ok"
}
```

6.4.2 从一个Thing移除一个Principal

| 相对URI | HTTP 方式 |
|-----------------------------------|---------|
| /v1/action/remove-thing-principal | POST |

[请求参数](#)

| 名称 | 类型 | 是否必选 | 含义 |
|---------------|--------|------|-------------------------|
| endpointName | String | Y | thing 所属的 endpoint的Name |
| thingName | String | Y | thing的名称 |
| principalName | String | Y | principal的名称 |

[返回参数](#)

无特殊返回参数。

[请求示例](#)

```
POST /v1/action/remove-thing-principal HTTP/1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/plain; charset=UTF-8
{
    "principalName": "principal-1"
    "endpointName": "endpoint-1"
    "thingName": "thing-1"
}
```

[返回示例](#)

```
HTTP/1.1 201 Created
x-bce-request-id: 46465c45-a6da-4211-ac11-ebbb9e0538da
Content-Type: application/json; charset=UTF-8
{
    "message": "ok"
}
```

6.4.3 给一个Principal添加一个Policy

| 相对URI | HTTP 方式 |
|------------------------------------|---------|
| /v1/action/attach-principal-policy | POST |

[请求参数](#)

| 名称 | 类型 | 是否必选 | 含义 |
|---------------|--------|------|---------------------------|
| endpointName | String | Y | principal 所属的 endpoint的名称 |
| principalName | String | Y | principal的名称 |
| policyName | String | Y | policy的名称 |

[返回参数](#)

无特殊返回参数。

[请求示例](#)

```
POST /v1/action/attach-principal-policy HTTP/1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/plain; charset=UTF-8
{
    "principalName": "principal-1"
    "endpointName": "endpoint-1"
    "policyName": "policy-1"
}
```

[返回示例](#)

```
HTTP/1.1 201 Created
x-bce-request-id: 2030375e-b56e-442b-b106-84778ae46f3b
Content-Type: application/json; charset=UTF-8
{
    "message": "ok"
}
```

6.4.4 从一个Principal移除一个Policy

| 相对URI | HTTP 方式 |
|--|---------|
| /v1/action/remove-thing-principal-policy | POST |

[请求参数](#)

| 名称 | 类型 | 是否必选 | 含义 |
|---------------|--------|------|---------------------------|
| endpointName | String | Y | principal 所属的 endpoint的名称 |
| principalName | String | Y | principal的名称 |
| policyName | String | Y | thing的名称 |

[返回参数](#)

无特殊返回参数。

[请求示例](#)

```
POST /v1/action/remove-principal-policy HTTP/1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/plain; charset=UTF-8
{
    "principalName": "principal-1"
    "endpointName": "endpoint-1"
    "policyName": "policy-1"
}
```

[返回示例](#)

```
HTTP/1.1 201 Created
x-bce-request-id: b998337c-a4fb-4bc1-b557-39d2b0fd27e7
Content-Type: application/json; charset=UTF-8
{
    "message": "ok"
}
```

6.5 Endpoint

6.5.1 获取endpoint列表

| 相对URI | HTTP 方式 |
|--------------|---------|
| /v1/endpoint | GET |

[请求参数](#)

| 名称 | 类型 | 是否必选 | 默认值 | 说明 |
|----------|------------------------|------|------------|-------------------|
| order | ENUM['desc' , 'asc'] | N | desc | 排序的方式，不区分大小写 |
| orderBy | String | N | createTime | 另外一个支持排序的字段是name。 |
| pageNo | Int | N | 1 | 页码 |
| pageSize | Int | N | 50 | 每页 item 个数，最大值200 |

| 名称 | 类型 | 是否必选 | 默认值 | 说明 |
|----|--------|------|-----|------------------------|
| q | String | N | - | 模糊查询的内容。目前支持name字段模糊查询 |

[返回参数](#)

| 名称 | 类型 | 含义 |
|-------------------|--------|-----------------------|
| accountUuid | String | 创建者的Uuid |
| creadteTime | String | 创建时间 |
| endpointName | String | endpoint名称 |
| mqttHostname | String | mqtt协议的url(非加密) |
| mqttTlsHostname | String | mqtt协议的url(加密) |
| uuid | String | 系统自动生成的一个endpoint的唯一值 |
| websocketHostname | String | websocket协议的url |

[请求示例](#)

```
GET /v1/endpoint HTTP/1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/plain; charset=UTF-8
```

[返回示例](#)

```
HTTP/1.1 200 OK
x-bce-request-id: a9000b0d-c13c-4241-9ee1-0876f750716f
Content-Type: application/json; charset=UTF-8
{
    "totalCount": 2,
    "result": [
        {
            "mqttHostname": "endpoint-1.xxx.iot.baidubce.com:8061",
            "accountUuid": "myacount",
            "hostname": "endpoint-1.xxx.iot.baidubce.com:61614",
            "endpointName": "endpoint-1",
            "uuid": "a9000b0d-c13c-4241-9ee1-0876f750716f"
        }
    ]
}
```

```

    "mqttTlsHostname": "endpoint-1.xxx.iot.baidubce.com:61614",
    "createTime": "2016-08-31T03:36:24Z",
    "websocketHostname": "endpoint-1.xxx.iot.baidubce.com:8064",
    "uuid": "2013ffab-c17e-4657-839d-941bbe6c6c84"
},
{
    "mqttHostname": "endpoint-2.xxx.iot.baidubce.com:8061",
    "accountUuid": "myaccount",
    "hostname": "endpoint-2.xxx.iot.baidubce.com:61614",
    "endpointName": "endpoint-2",
    "mqttTlsHostname": "endpoint-2.xxx.iot.baidubce.com:61614",
    "createTime": "2016-08-25T08:43:42Z",
    "websocketHostname": "endpoint-2.xxx.iot.baidubce.com:8064",
    "uuid": "9e65bb32-731b-4b25-982b-13a12a0ff35e"
}
],
"order": "desc",
"orderBy": "createtime",
"pageSize": 50,
"pageNo": 1
}

```

6.5.2 获取指定的endpoint信息

| 相对URI | HTTP 方式 |
|-----------------------------|---------|
| /v1/endpoint/{endpointName} | GET |

请求参数

| 名称 | 类型 | 是否必选 | 含义 |
|--------------|--------|------|-------------|
| endpointName | String | Y | endpoint的名称 |

返回参数

| 名称 | 类型 | 含义 |
|--------------|--------|--------------------|
| accountUuid | String | 创建者的Uuid |
| createTime | String | 创建时间 |
| endpointName | String | endpoint名称 |
| hostname | String | mqtt连接时所需的host |
| mqttHostname | String | mqtt协议的url (非加密) |

| 名称 | 类型 | 含义 |
|-------------------|--------|-----------------------|
| mqttTlsHostname | String | mqtt协议的url(加密) |
| uuid | String | 系统自动生成的一个endpoint的唯一值 |
| websocketHostname | String | websocket协议的url |

请求示例

```
GET /v1/endpoint/endpoint-1 HTTP/1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/plain; charset=UTF-8
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: f494dff9-7ece-43be-a509-b8c1cf56e13f
Content-Type: application/json; charset=UTF-8
{
    "hostname": "endpoint-1.xxx.iot.baidubce.com:61614",
    "mqttHostname": "endpoint-1.xxx.iot.baidubce.com:8061",
    "createTime": "2016-08-31T03:36:24Z",
    "uuid": "2013ffab-c17e-4657-839d-941bbe6c6c84",
    "accountUuid": "mycount",
    "websocketHostname": "endpoint-1.xxx.iot.baidubce.com:8064",
    "mqttTlsHostname": "endpoint-1.xxx.iot.baidubce.com:61614",
    "endpointName": "endpoint-1"
}
```

6.5.3 创建endpoint

| 相对URI | HTTP 方式 |
|--------------|---------|
| /v1/endpoint | POST |

请求参数

| 名称 | 类型 | 是否必选 | 含义 |
|--------------|--------|------|-------------|
| endpointName | String | Y | endpoint的名称 |

返回参数

| 名称 | 类型 | 含义 |
|-------------------|--------|-----------------------|
| accountUuid | String | 创建者的Uuid |
| createTime | String | 创建时间 |
| endpointName | String | endpoint名称 |
| hostname | String | mqtt连接时所需的host |
| mqttHostname | String | mqtt协议的url (非加密) |
| mqttTlsHostname | String | mqtt协议的url (加密) |
| uuid | String | 系统自动生成的一个endpoint的唯一值 |
| websocketHostname | String | websocket协议的url |

请求示例

```
POST /v1/endpoint HTTP/1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/plain; charset=UTF-8
{
    "endpointName": "endpoint-1"
}
```

返回示例

```
HTTP/1.1 201 Created
x-bce-request-id: a80aba01-f81e-4dfc-a57b-76b05a30ee07
Content-Type: application/json; charset=UTF-8
{
    "mqttHostname": "endpoint-1.xxx.iot.baidubce.com:8061",
    "accountUuid": "myaccount",
    "hostname": "endpoint-1.xxx.iot.baidubce.com:61614",
    "endpointName": "endpoint-1",
    "mqttTlsHostname": "endpoint-1.xxx.iot.baidubce.com:61614",
    "createTime": "2016-08-31T03:36:24Z",
```

```

    "websocketHostname": "endpoint-1.xxx.iot.baidubce.com:8064",
    "uuid": "2013ffab-c17e-4657-839d-941bbe6c6c84"
}

```

6.5.4 删除endpoint

| 相对URI | HTTP 方式 |
|-----------------------------|---------|
| /v1/endpoint/{endpointName} | DELETE |

请求参数

| 名称 | 类型 | 是否必选 | 含义 |
|--------------|--------|------|-------------|
| endpointName | String | Y | endpoint的名称 |

返回参数

无特殊返回参数。

请求示例

```

DELETE /v1/endpoint/endpoint-1 HTTP/1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/plain; charset=UTF-8

```

返回示例

```

HTTP/1.1 204 No Content
x-bce-request-id: 50f11d37-54ea-44a5-9863-94441906b9bc
Content-Type: application/json; charset=UTF-8

```

6.6 Thing

6.6.1 获取thing列表

| 相对URI | HTTP 方式 |
|-----------------------------------|---------|
| /v1/endpoint/{endpointName}/thing | GET |

请求参数

| 名称 | 类型 | 是否必选 | 默认值 | 说明 |
|--------------|------------------------|------|------------|--------------------------|
| endpointName | String | Y | - | endpoint 的名称 |
| order | ENUM[‘desc’ , ‘asc’] | N | desc | 排序的方式，不区分大小写 |
| orderBy | String | N | createTime | 另外一个支持排序的字段是 name。 |
| pageNo | Int | N | 1 | 页码 |
| pageSize | Int | N | 50 | 每页 item 个数，最大值200 |
| q | String | N | - | 模糊查询的内容。目前支持 name 字段模糊查询 |

返回参数

| 名称 | 类型 | 含义 |
|--------------|--------|-------------|
| username | String | 用于mqtt认证 |
| thingName | String | thing的名称 |
| endpointName | String | 所属的Endpoint |
| createTime | String | 创建时间 |

请求示例

```
GET /v1/endpoint/endpoint-1/thing HTTP/1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/plain; charset=UTF-8
```

返回示例

```
HTTP/1.1 200 Created
x-bce-request-id: c6903ab2-bf5e-4afb-8bbf-2e537edd3c49
Content-Type: application/json; charset=UTF-8
```

```
{
  "totalCount": 1,
  "result": [
    {
      "username": "endpoint-1/thing-1",
      "thingName": "thing-1",
      "endpointName": "endpoint-1",
      "createTime": "2016-08-31T05:12:39Z"
    }
  ],
  "order": "desc",
  "orderBy": "createtime",
  "pageSize": 50,
  "pageNo": 1
}
```

6.6.2 获取指定的thing信息

| 相对URI | HTTP 方式 |
|--|---------|
| / v1/ endpoint/ {endpointName}/ thing/ {thingName} | GET |

请求参数

| 名称 | 类型 | 是否必选 | 含义 |
|--------------|--------|------|-------------|
| endpointName | String | Y | endpoint的名称 |
| thingName | String | Y | thing的名称 |

返回参数

| 名称 | 类型 | 含义 |
|--------------|--------|-------------|
| username | String | 用于mqtt认证 |
| thingName | String | thing的名称 |
| endpointName | String | 所属的Endpoint |
| createTime | String | 创建时间 |

请求示例

```
GET /v1/endpoint/endpoint-1/thing/thing-1 HTTP/1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/plain; charset=UTF-8
```

返回示例

```
HTTP/1.1 200 Created
x-bce-request-id: 360e61be-36fd-4607-b226-199fd6bc11bd
Content-Type: application/json; charset=UTF-8
{
    "username": "endpoint-1/thing-1",
    "thingName": "thing-1",
    "endpointName": "endpoint-1",
    "createTime": "2016-08-31T05:12:39Z"
}
```

6.6.3 创建thing

| 相对URI | HTTP 方式 |
|-----------------------------------|---------|
| /v1/endpoint/{endpointName}/thing | POST |

请求参数

| 名称 | 类型 | 是否必选 | 含义 |
|--------------|--------|------|-------------|
| endpointName | String | Y | endpoint的名称 |
| thingName | String | Y | thing的名称 |

返回参数

| 名称 | 类型 | 含义 |
|--------------|--------|-------------|
| username | String | 用于mqtt认证 |
| thingName | String | thing的名称 |
| endpointName | String | 所属的Endpoint |
| createTime | String | 创建时间 |

请求示例

```
POST /v1/endpoint/endpoint-1/thing HTTP/1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/plain; charset=UTF-8
{
    "thingName": "thing-1"
}
```

[返回示例](#)

```
HTTP/1.1 201 Created
x-bce-request-id: 22b69f0d-11ad-4547-a1dd-20ac37282f63
Content-Type: application/json; charset=UTF-8
{
    "username": "endpoint-1/thing-1",
    "thingName": "thing-1",
    "endpointName": "endpoint-1",
    "createTime": "2016-08-31T05:12:39Z"
}
```

6.6.4 删除thing

| 相对URI | HTTP 方式 |
|--|---------|
| / v1/ endpoint/ {endpointName}/ thing/ {thingName} | DELETE |

[请求参数](#)

| 名称 | 类型 | 是否必选 | 含义 |
|--------------|--------|------|-------------|
| endpointName | String | Y | endpoint的名称 |
| thingName | String | Y | thing的名称 |

[返回参数](#)

无特殊返回参数。

[请求示例](#)

```
DELETE /v1/endpoint/endpoint-1/thing/thing-1 HTTP/1.1
```

```
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/plain;charset=UTF-8
```

[返回示例](#)

```
HTTP/1.1 204 No Content
x-bce-request-id: 367d9dd5-4513-412d-a4f8-eacfe37dbaf3
Content-Type: application/json;charset=UTF-8
```

6.7 Principal

6.7.1 获取principal列表

| 相对URI | HTTP 方式 |
|---|---------|
| /v1/endpoint/{endpointName}/principal? thingName={thingName} | GET |

[请求参数](#)

| 名称 | 类型 | 是否必选 | 默认值 | 说明 |
|--------------|------------------------|------|------------|---|
| endpointName | String | Y | - | endpoint 的 名称 |
| thingName | String | N | - | principal 所 属的thing名称，可选参数，可查询与指定thing 绑定的principal |
| order | ENUM[‘desc’ , ‘asc’] | N | desc | 排序的方式，不区分大小写 |
| orderBy | String | N | createTime | 另外一个支持排序的字段是name。 |
| pageNo | Int | N | 1 | 页码 |
| pageSize | Int | N | 50 | 每页 item 个数，最大值200 |

| 名称 | 类型 | 是否必选 | 默认值 | 说明 |
|----|--------|------|-----|------------------------|
| q | String | N | - | 模糊查询的内容。目前支持name字段模糊查询 |

[返回参数](#)

| 名称 | 类型 | 含义 |
|---------------|--------|---------------|
| principalName | String | principal名称 |
| endpointName | String | 所属的Endpoint名称 |
| createTime | String | 创建时间 |

[请求示例](#)

```
GET /v1/endpoint/endpoint-1/principal HTTP/1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/plain; charset=UTF-8
```

[返回示例](#)

```
HTTP/1.1 200 OK
x-bce-request-id: e20e06f3-d0da-4668-ac67-ce62047e3beb
Content-Type: application/json; charset=UTF-8
{
    "totalCount": 1,
    "result": [
        {
            "principalName": "principal-1",
            "endpointName": "endpoint-1",
            "createTime": "2016-08-31T06:09:29Z"
        }
    ],
    "order": "desc",
    "orderBy": "createtime",
    "pageSize": 50,
    "pageNo": 1
}
```

6.7.2 获取指定的principal信息

| 相对URI | HTTP 方式 |
|---|---------|
| /v1/endpoint/{endpointName}/principal/{principalName} | GET |

请求参数

| 名称 | 类型 | 是否必选 | 含义 |
|---------------|--------|------|-------------|
| endpointName | String | Y | endpoint名称 |
| principalName | String | Y | principal名称 |

返回参数

| 名称 | 类型 | 含义 |
|---------------|--------|---------------|
| principalName | String | principal名称 |
| endpointName | String | 所属的Endpoint名称 |
| createTime | String | 创建时间 |

请求示例

```
GET /v1/endpoint/endpoint-1/principal/principal-1 HTTP/1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/plain; charset=UTF-8
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: e294804d-d946-4a89-a671-317d4eaeae6c
Content-Type: application/json; charset=UTF-8
{
    "principalName": "principal-1",
    "endpointName": "endpoint-1",
    "createTime": "2016-08-31T06:09:29Z"
}
```

6.7.3 创建principal

| | |
|---------------------------------------|---------|
| 相对URI | HTTP 方式 |
| /v1/endpoint/{endpointName}/principal | POST |

请求参数

| 名称 | 类型 | 是否必选 | 含义 |
|---------------|--------|------|-------------|
| endpointName | String | Y | endpoint名称 |
| principalName | String | Y | principal名称 |

返回参数

| 名称 | 类型 | 含义 |
|---------------|--------|-------------|
| principalName | String | principal名称 |
| endpointName | String | 所属的Endpoint |
| password | String | principal密钥 |
| privateKey | String | PEM格式的私钥 |
| cert | String | PEM格式的证书 |
| createTime | String | 创建时间 |

请求示例

```
POST /v1/endpoint/endpoint-1/principal HTTP/1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/plain; charset=UTF-8
{
"principalName": "principal-1"
}
```

返回示例

```
HTTP/1.1 201 Created
x-bce-request-id: 9f913092-f0ef-4018-9055-e02f2db4464f
Content-Type: application/json; charset=UTF-8
{
```

```

"principalName": "principal-1",
"endpointName": "endpoint-1",
"privateKey": "-----BEGIN RSA PRIVATE KEY----- \\\\r\\\\nMIICWwIBAAKBgQC98S6X8nCB/
3AdsK3uXpx7YfCP/.....-----END RSA PRIVATE KEY-----\\\\r\\\\n",
"password": "XM5kW9bZhRTWc/mag8TYZuVlUfdaLUju+kdwh9dfGKo=",
"createTime": "2016-08-31T06:09:28Z",
"cert": "-----BEGIN CERTIFICATE----- \\\\r\\\\nMIIC4zCCAc8CCQC3v0MerfCjdjANBgkqhkiG9w0BAQsFADBqMQs
\\\\r .....+LIJhgE/2s67BdGEGHOjWhtdi1KT1 0g1A/t\\\\r\\\\nqhn/EexhgR6CgLNmxB4+tPN2CrJtL6c6j2moVq5dtP0g6V
\\\\r\\\\n-----END CERTIFICATE-----\\\\r\\\\n"
}

```

6.7.4 重新生成密钥

| 相对URI | HTTP 方式 |
|---|---------|
| /v1/endpoint/{endpointName}/principal/{principalName} | POST |

请求参数

| 名称 | 类型 | 是否必选 | 含义 |
|---------------|--------|------|--------------------------|
| endpointName | String | Y | endpoint名称 |
| principalName | String | Y | principal名称 |
| target | String | N | 可选项: all, password, cert |

返回参数

| 名称 | 类型 | 含义 |
|---------------|--------|-------------|
| principalName | String | principal名称 |
| endpointName | String | 所属的Endpoint |
| password | String | principal密钥 |
| privateKey | String | PEM格式的私钥 |
| cert | String | PEM格式的证书 |

请求示例

```

POST /v1/endpoint/endpoint-1/principal/principal-1 HTTP/1.1
host: iot.gz.baidubce.com

```

```
authorization: {authorization}
content-type: text/plain;charset=UTF-8
```

响应示例

```
HTTP/1.1 200 Created
x-bce-request-id: 9f913092-f0ef-4018-9055-e02f2db4464f
Content-Type: application/json;charset=UTF-8
{
    "principalName": "principal-1",
    "endpointName": "endpoint-1",
    "privateKey": "-----BEGIN RSA PRIVATE KEY-----\\r\\nMIICXgIBAAKBgQCuoZEOP+3c/
ur8x0bVzVFHBApx2gCi1X+r/q/5B4EDDp33+wZe\\r\\n+tJ6MfbkgZbvNxEZ....\\n88WhhVIDdnneYTe8wIJ1PVBZG0kv6c
\\r\\n-----END RSA PRIVATE KEY-----\\r\\n",
    "password": "pN04VRhK2h0BJpp3ayyHuy+dY0t7HFA1W1rtnoRJ76g=",
    "cert": "-----BEGIN CERTIFICATE-----\\r\\nMIIC4zCCAcSCCQDHBYyIQT8cajANBhkqkiG9w0BAQsFADBqMQswCQ
\\r....\\nrSe0T4miCW7kXLNwqNW7SwB7NZ4GgG8nsK1lFSfx2hK7a41vvE18+xiszJlBHhF\\r\\
\\nAGhpPDgBUcgy77G4W3+kg0XRdVm77vI=\\r\\n-----END CERTIFICATE-----\\r\\n"
}
```

6.7.5 删除principal

| 相对URI | HTTP 方式 |
|---|---------|
| /v1/endpoint/{endpointName}/principal/{principalName} | DELETE |

请求参数

| 名称 | 类型 | 是否必选 | 含义 |
|---------------|--------|------|-------------|
| endpointName | String | Y | endpoint名称 |
| principalName | String | Y | principal名称 |

返回参数

无特殊返回参数。

请求示例

```
DELETE /v1/endpoint/endpoint-1/principal/principal-1 HTTP/1.1
host: iot.gz.baidubce.com
```

```
authorization: {authorization}
content-type: text/plain; charset=UTF-8
```

[返回示例](#)

```
HTTP/1.1 204 No Content
x-bce-request-id: 7dda6e6a-08cd-4301-90df-a5c0f7a502e8
Content-Type: application/json; charset=UTF-8
```

6.8 Policy

6.8.1 获取policy列表

| 相对URI | HTTP 方式 |
|--|---------|
| / v1/ endpoint/ {endpointName}/ policy? principalName={principalName} | GET |

[请求参数](#)

| 名称 | 类型 | 是否必选 | 默认值 | 说明 |
|---------------|------------------------|------|------------|-------------------------|
| endpointName | String | Y | - | endpoint名称 |
| principalName | String | N | - | Policy 所属的 principal名称 |
| order | ENUM['desc' , 'asc'] | N | desc | 排序的方式，不区分大小写 |
| orderBy | String | N | createTime | 另外一个支持排序的字段是 name |
| pageNo | Int | N | 1 | 页码 |
| pageSize | Int | N | 50 | 每页 item 个数，最大值200 |
| q | String | N | - | 模糊查询的内容。目前支持 name字段模糊查询 |

[返回参数](#)

| 名称 | 类型 | 含义 |
|--------------|--------|-------------|
| policyName | String | policy名称 |
| endpointName | String | 所属的Endpoint |
| createTime | String | 创建时间 |

请求示例

```
GET /v1/endpoint/endpoint-1/policy HTTP/1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/plain;charset=UTF-8
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 9bcf7816-f997-4152-ad50-7c06dbb41bae
Content-Type: application/json;charset=UTF-8
{
    "totalCount": 1,
    "result": [
        {
            "endpointName": "endpoint-1",
            "policyName": "policy-1",
            "createTime": "2016-08-31T06:26:40Z"
        }
    ],
    "order": "desc",
    "orderBy": "createtime",
    "pageSize": 50,
    "pageNo": 1
}
```

6.8.2 获取指定的policy信息

| 相对URI | HTTP 方式 |
|--|---------|
| / v1/ endpoint/ {endpointName}/ policy/ {policyName} | GET |

请求参数

| 名称 | 类型 | 是否必选 | 含义 |
|--------------|--------|------|------------|
| endpointName | String | Y | endpoint名称 |
| policyName | String | Y | policy名称 |

返回参数

| 名称 | 类型 | 含义 |
|--------------|--------|---------------|
| policyName | String | policy名称 |
| endpointName | String | 所属的endpoint名称 |
| createTime | String | 创建时间 |

请求示例

```
GET /v1/endpoint/endpoint-1/policy/policy-1 HTTP/1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/plain; charset=UTF-8
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 6f0106e3-06e0-4eb3-844e-02284539d14e
Content-Type: application/json; charset=UTF-8
{
    "endpointName": "endpoint-1",
    "policyName": "policy-1",
    "createTime": "2016-08-31T06:26:40Z"
}
```

6.8.3 创建policy

| 相对URI | HTTP 方式 |
|------------------------------------|---------|
| /v1/endpoint/{endpointName}/policy | POST |

请求参数

| 名称 | 类型 | 是否必选 | 含义 |
|--------------|--------|------|------------|
| endpointName | String | Y | endpoint名称 |
| policyName | String | Y | policy名称 |

返回参数

| 名称 | 类型 | 含义 |
|--------------|--------|---------------|
| policyName | String | policy名称 |
| endpointName | String | 所属的endpoint名称 |
| createTime | String | 创建时间 |

请求示例

```
POST /v1/endpoint/endpoint-1/policy HTTP/1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/plain; charset=UTF-8
{
    "policyName": "policy-1"
}
```

响应示例

```
HTTP/1.1 201 Created
x-bce-request-id: e9f39305-d67c-45ee-9130-5e50e567fc8d
Content-Type: application/json; charset=UTF-8
{
    "endpointName": "endpoint-1",
    "policyName": "policy-1",
    "createTime": "2016-08-31T06:26:40Z"
}
```

6.8.4 删除policy

| 相对URI | HTTP 方式 |
|--|---------|
| / v1/ endpoint/ {endpointName}/ policy/ {policyName} | DELETE |

请求参数

| 名称 | 类型 | 是否必选 | 含义 |
|--------------|--------|------|------------|
| endpointName | String | Y | endpoint名称 |
| policyName | String | Y | policy名称 |

返回参数

无特殊返回参数。

请求示例

```
DELETE /v1/endpoint/endpoint-1/princy/princy-1 HTTP/1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/plain; charset=UTF-8
```

返回示例

```
HTTP/1.1 204 No Content
x-bce-request-id: e2e12a78-32cf-4da5-8fd8-87d7f76e3295
Content-Type: application/json; charset=UTF-8
```

6.9 Permission

6.9.1 获取policy下所有topic信息

| 相对URI | HTTP 方式 |
|--|---------|
| / v1/ endpoint/ {endpointName}/ permission?policyName={policyName} | GET |

请求参数

| 名称 | 类型 | 是否必选 | 默认值 | 说明 |
|--------------|--------|------|-----|------------|
| endpointName | String | Y | - | endpoint名称 |
| policyName | String | Y | - | policy名称 |

| 名称 | 类型 | 是否必选 | 默认值 | 说明 |
|----------|------------------------|------|------------|-------------------|
| order | ENUM[‘desc’ , ‘asc’] | N | desc | 排序的方式，不区分大小写 |
| orderBy | String | N | createTime | 仅支持基于createTime排序 |
| pageNo | Int | N | 1 | 页码 |
| pageSize | Int | N | 50 | 每页 item 个数，最大值200 |

[返回参数](#)

| 名称 | 类型 | 含义 |
|----------------|------------|---------------|
| operations | List[ENUM] | 允许的操作 list |
| permissionUuid | String | permission的ID |
| policyUuid | String | policy的ID |
| topic | String | 操作对应的Topic |
| createTime | String | 创建时间 |

[请求示例](#)

```
GET /v1/endpoint/endpoint-1/permission?policyName=policy-1 HTTP/1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/plain; charset=UTF-8
```

[返回示例](#)

```
HTTP/1.1 200 OK
x-bce-request-id: eaa235de-f71c-4ed4-baa0-8b380ffb90ba
Content-Type: application/json; charset=UTF-8
{
    "totalCount": 1,
    "result": [
        {
            "policyUuid": "f1615eb2-9ff7-439f-b9db-8c2c5a5476b9",
            "operations": [
                "PUBLISH",
                "SUBSCRIBE"
            ]
        }
    ]
}
```

```

        ],
        "topic": "topic1",
        "createTime": "2016-08-31T06:44:01Z",
        "permissionUuid": "ba8313a8-b2ed-4079-8160-00fc168d6d9c"
    }
],
"order": "desc",
"orderBy": "createtime",
"pageSize": 50,
"pageNo": 1
}

```

6.9.2 获取指定topic的信息

| 相对URI | HTTP 方式 |
|--|---------|
| /v1/ endpoint/ {endpointName}/ permission/{permissionUuid} | GET |

请求参数

| 名称 | 类型 | 是否必选 | 含义 |
|----------------|--------|------|---------------|
| endpointName | String | Y | endpoint名称 |
| permissionUuid | String | Y | permission的ID |

返回参数

| 名称 | 类型 | 含义 |
|----------------|------------|---------------|
| operations | List[ENUM] | 允许的操作 list |
| permissionUuid | String | permission的ID |
| policyUuid | String | policy的ID |
| topic | String | 操作对应的Topic |
| createTime | String | 创建时间 |

请求示例

```

GET /v1/endpoint/endpoint-1/permission/ba8313a8-b2ed-4079-8160-00fc168d6d9c HTTP/
1.1
host: iot.gz.baidubce.com

```

```
authorization: {authorization}
content-type: text/plain;charset=UTF-8
```

[返回示例](#)

```
HTTP/1.1 200 OK
x-bce-request-id: 139eedb4-14f9-4512-94fb-bbac2b334bf3
Content-Type: application/json;charset=UTF-8
{
    "policyUuid": "f1615eb2-9ff7-439f-b9db-8c2c5a5476b9",
    "operations": [
        "PUBLISH",
        "SUBSCRIBE"
    ],
    "topic": "topic1",
    "createTime": "2016-08-31T06:44:01Z",
    "permissionUuid": "ba8313a8-b2ed-4079-8160-00fc168d6d9c"
}
```

6.9.3 在policy下设置topic

| 相对URI | HTTP 方式 |
|--|---------|
| /v1/endpoint/{endpointName}/permission | POST |

[请求参数](#)

| 名称 | 类型 | 是否必选 | 含义 |
|--------------|------------|------|-------------|
| endpointName | String | Y | endpoint名称 |
| policyName | String | Y | 所属的Policy名称 |
| operations | List[ENUM] | Y | 允许的操作list |
| topic | String | Y | 操作对应的Topic |

[返回参数](#)

| 名称 | 类型 | 含义 |
|----------------|------------|---------------|
| operations | List[ENUM] | 允许的操作 list |
| permissionUuid | String | permission的ID |

| 名称 | 类型 | 含义 |
|------------|--------|------------|
| policyUuid | String | policy的ID |
| topic | String | 操作对应的Topic |
| createTime | String | 创建时间 |

请求示例

```
POST /v1/endpoint/endpoint-1/permission HTTP/1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/plain; charset=UTF-8
{
    "policyName": "policy-1"
    "operations": ["PUBLISH", "SUBSCRIBE"]
    "topic": "topic1"
}
```

返回示例

```
HTTP/1.1 201 Created
x-bce-request-id: 6fb3044f-b5ef-4a8f-a416-51fa0e48f510
Content-Type: application/json; charset=UTF-8
{
    "policyUuid": "f1615eb2-9ff7-439f-b9db-8c2c5a5476b9",
    "operations": [
        "PUBLISH",
        "SUBSCRIBE"
    ],
    "topic": "topic1",
    "createTime": "2016-08-31T06:44:01Z",
    "permissionUuid": "ba8313a8-b2ed-4079-8160-00fc168d6d9c"
}
```

6.9.4 更新已有的topic设置

| 相对URI | HTTP 方式 |
|---|---------|
| / v1/ endpoint/ {endpointName}/ permission/{permissionUuid} | PUT |

请求参数

| 名称 | 类型 | 是否必选 | 含义 |
|----------------|------------|------|---------------|
| endpointName | String | Y | endpoint名称 |
| permissionUuid | String | Y | permission的ID |
| operations | List[ENUM] | Y | 允许的操作list |
| topic | String | Y | 操作对应的Topic |

返回参数

| 名称 | 类型 | 含义 |
|----------------|------------|---------------|
| operations | List[ENUM] | 允许的操作 list |
| permissionUuid | String | permission的ID |
| policyUuid | String | policy的ID |
| topic | String | 操作对应的Topic |
| createTime | String | 创建时间 |

请求示例

```
PUT /v1/endpoint/endpoint-1/permission/ba8313a8-b2ed-4079-8160-00fc168d6d9c HTTP/
1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/plain; charset=UTF-8
{
    "operations": ["PUBLISH"]
    "topic": "topic2"
}
```

返回示例

```
HTTP/1.1 201 Created
x-bce-request-id: 853459a6-b933-4546-8bac-8a174516e83f
Content-Type: application/json; charset=UTF-8
{
    "policyUuid": "f1615eb2-9ff7-439f-b9db-8c2c5a5476b9",
    "operations": [
        "PUBLISH"
    ]
}
```

```
        ],
    "topic": "topic2",
    "createTime": "2016-08-31T06:44:01Z",
    "permissionUuid": "ba8313a8-b2ed-4079-8160-00fc168d6d9c"
}
```

6.9.5 删除已有的topic

| 相对URI | HTTP 方式 |
|---|---------|
| /v1/endpoint/{endpointName}/permission/{permissionUuid} | DELETE |

请求参数

| 名称 | 类型 | 是否必选 | 含义 |
|----------------|--------|------|---------------|
| endpointName | String | Y | endpoint名称 |
| permissionUuid | String | Y | permission的ID |

返回参数

无特殊返回参数。

请求示例

```
DELETE /v1/endpoint/endpoint-1/permission/ba8313a8-b2ed-4079-8160-00fc168d6d9c HTTP/
1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/plain; charset=UTF-8
```

返回示例

```
HTTP/1.1 204 No Content
x-bce-request-id: 44dcf34e-bc58-46a8-8492-cde566c69328
Content-Type: application/json; charset=UTF-8
```

6.10 Client

6.10.1 获取指定MQTT客户端在线状态

| 相对URI | HTTP 方式 |
|---|---------|
| /v2/endpoint/{endpointName}/client/{clientId}/status/online | GET |

请求参数

| 名称 | 类型 | 是否必选 | 含义 |
|--------------|--------|------|------------|
| endpointName | String | Y | endpoint名称 |
| clientId | String | Y | MQTT客户端ID |

请求示例

```
GET /v2/endpoint/endpoint-1/client/abc/status/online HTTP/1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/json;charset=UTF-8
```

返回示例

```
HTTP/1.1 200 Ok
x-bce-request-id: 367d9dd5-4513-412d-a4f8-eacfe37dbaf3
Content-Type: application/json;charset=UTF-8

true
```

6.10.2 获取所有MQTT客户端在线状态

| 相对URI | HTTP 方式 |
|---|---------|
| /v2/endpoint/{endpointName}/batch-client-query/status | POST |

请求参数

| 名称 | 类型 | 是否必选 | 含义 |
|--------------|--------|------|-------------|
| endpointName | String | Y | endpoint名称 |
| mqttID | Array | Y | MQTT客户端ID列表 |

[返回参数](#)

| 名称 | 类型 | 含义 |
|----------|---------|-----------|
| clientId | String | MQTT客户端ID |
| online | Boolean | 在线状态 |

[请求示例](#)

```
POST /v2/endpoint/endpoint-1/batch-client-query/status HTTP/1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/json;charset=UTF-8

["clientId1", "clientId2"]
```

[返回示例](#)

```
HTTP/1.1 200 Ok
x-bce-request-id: 367d9dd5-4513-412d-a4f8-eacfe37dbaf3
Content-Type: application/json;charset=UTF-8

[{"clientId": "clientId1", "online": true}, {"clientId": "clientId2", "online": false}]
```

6.10.3 Publish Message

MQTT客户端可以调用此API接口，以HTTP方式接入，实现消息发布（不支持消息订阅）。

调用此API不需要携带百度云签名认证字符串。

注意

建议使用HTTPS访问，HTTP用户名及密码泄露。

| URI | HTTP 方式 | Content Type | Return Type |
|--|---------|--------------------------|---------------------------------|
| <code>https://api.mqtt.iot.gz.baidubce.com/v1/proxy?qos=0&topic=topic1&retain=false</code> | POST | application/octet-stream | application/json; charset=UTF-8 |

请求参数

| 名称 | 是否必选 | 含义 |
|--------|------|---------------------------------------|
| qos | Y | 该消息的QoS取值，可选0或1 |
| topic | Y | topic名称，客户端将向指定的topic发布消息 |
| retain | N | retain标记，详细介绍请参看 保留消息 |

请求头参数

| 名称 | 是否必选 | 含义 |
|---------------|------|-----------------|
| auth.username | Y | 用户名，即：实例名称/设备名称 |
| auth.password | Y | 密码，即：创建身份时获得的密钥 |

发布的具体消息内容放在HTTP Content中，可以是二进制消息。消息长度不大于32K。

请求示例

```
POST /v1/proxy?qos=0&topic=topic1&retain=false HTTP/1.1
host: api.mqtt.iot.gz.baidubce.com
content-type: application/octet-stream

auth.username: test/test
auth.password: cNCcTh+xkth7Jk2EHWUo2+IhkMsDnQlmMfMi93CsifY=
```

成功：201，失败：4XX or 5XX

6.11 使用量

6.11.1 获取当前账单月内使用量

接口描述

| 相对URI | HTTP 方式 |
|-----------|---------|
| /v1/usage | GET |

输出

200: 成功

| 名称 | 类型 | 默认值 | 含义 |
|-----------------|--------|-----|-------------------|
| publishReceived | Number | | 物接入收到的PUBLISH消息 |
| publishSent | Number | | 物接入向外发送的PUBLISH消息 |

请求示例

```
GET /v1/usage HTTP/1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/json;charset=UTF-8
```

返回示例

```
HTTP/1.1 200 Ok
x-bce-request-id: 367d9dd5-4513-412d-a4f8-eacfe37dbaf3
Content-Type: application/json;charset=UTF-8
{"publishReceived":0,"publishSent":0}
```

6.11.2 获取当前账单月内特定实例的使用量

接口描述

| 相对URI | HTTP 方式 |
|----------------------------------|---------|
| /v1/endpoint/:endpointName/usage | GET |

输出

200: 成功

| 名称 | 类型 | 默认值 | 含义 |
|-----------------|--------|-----|-------------------|
| publishReceived | Number | | 物接入收到的PUBLISH消息 |
| publishSent | Number | | 物接入向外发送的PUBLISH消息 |

请求示例

```
GET /v1/endpoint/endpoint-1/usage HTTP/1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/json; charset=UTF-8
```

返回示例

```
HTTP/1.1 200 Ok
x-bce-request-id: 367d9dd5-4513-412d-a4f8-eacfe37dbaf3
Content-Type: application/json; charset=UTF-8
{"publishReceived":0,"publishSent":0}
```

6.11.3 查询特定实例某个时间段内的使用量

接口描述

| 相对URI | HTTP 方式 |
|---|---------|
| / v1/ endpoint/:endpointName/ usage-query?start=:startDate&end=:endDate | POST |

输入

| 名称 | 类型 | 默认值 | 含义 |
|-------|--------|-----|-----------|
| start | String | 无 | 开始日期(包含) |
| end | String | 无 | 结束日期(不包含) |

输出

200: 成功

| 名称 | 类型 | 默认值 | 含义 |
|-----------------|--------|-----|-------------------|
| publishReceived | Number | | 物接入收到的PUBLISH消息 |
| publishSent | Number | | 物接入向外发送的PUBLISH消息 |

请求示例

```
POST /v1/endpoint/endpoint-1/usage-query?start=2017-07-10&end=2017-0810 HTTP/
1.1
host: iot.gz.baidubce.com
authorization: {authorization}
content-type: text/json;charset=UTF-8
```

返回示例

```
HTTP/1.1 200 Ok
x-bce-request-id: 367d9dd5-4513-412d-a4f8-eacf37dbaf3
Content-Type: application/json;charset=UTF-8
{"value": [{"date": "2017-07-10", "usage": {"publishReceived": 0, "publishSent": 0}}, {"date": "2017-07-11", "usage": {"publishReceived": 0, "publishSent": 0}}, {"date": "2017-07-12", "usage": {"publishReceived": 0, "publishSent": 0}}, {"date": "2017-07-13", "usage": {"publishReceived": 0, "publishSent": 0}}, {"date": "2017-07-14", "usage": {"publishReceived": 0, "publishSent": 0}}, {"date": "2017-07-15", "usage": {"publishReceived": 0, "publishSent": 0}}, {"date": "2017-07-16", "usage": {"publishReceived": 0, "publishSent": 0}}, {"date": "2017-07-17", "usage": {"publishReceived": 0, "publishSent": 0}}, {"date": "2017-07-18", "usage": {"publishReceived": 0, "publishSent": 0}}, {"date": "2017-07-19", "usage": {"publishReceived": 0, "publishSent": 0}}, {"date": "2017-07-20", "usage": {"publishReceived": 0, "publishSent": 0}}, {"date": "2017-07-21", "usage": {"publishReceived": 0, "publishSent": 0}}, {"date": "2017-07-22", "usage": {"publishReceived": 0, "publishSent": 0}}, {"date": "2017-07-23", "usage": {"publishReceived": 0, "publishSent": 0}}, {"date": "2017-07-24", "usage": {"publishReceived": 0, "publishSent": 0}}, {"date": "2017-07-25", "usage": {"publishReceived": 0, "publishSent": 0}}, {"date": "2017-07-26", "usage": {"publishReceived": 0, "publishSent": 0}}, {"date": "2017-07-27", "usage": {"publishReceived": 0, "publishSent": 0}}, {"date": "2017-07-28", "usage": {"publishReceived": 0, "publishSent": 0}}, {"date": "2017-07-29", "usage": {"publishReceived": 0, "publishSent": 0}}, {"date": "2017-07-30", "usage": {"publishReceived": 0, "publishSent": 0}}, {"date": "2017-07-31", "usage": {"publishReceived": 0, "publishSent": 0}}, {"date": "2017-08-01", "usage": {"publishReceived": 0, "publishSent": 0}}, {"date": "2017-08-02", "usage": {"publishReceived": 0, "publishSent": 0}}, {"date": "2017-08-03", "usage": {"publishReceived": 0, "publishSent": 0}}]}
```

```
0}} , {"date": "2017-08-04" , "usage": {"publishReceived": 0 , "publishSent": 0}} , {"date": "2017-08-05" , "usage": {"publishReceived": 0 , "publishSent": 0}} , {"date": "2017-08-06" , "usage": {"publishReceived": 0 , "publishSent": 0}} , {"date": "2017-08-07" , "usage": {"publishReceived": 0 , "publishSent": 0}} , {"date": "2017-08-08" , "usage": {"publishReceived": 0 , "publishSent": 0}} , {"date": "2017-08-09" , "usage": {"publishReceived": 0 , "publishSent": 0}} ]}
```

第7章 API参考(设备型)

7.1 目录

- 介绍
 - 签名认证
 - 多区域选择
- 设备列表管理
 - 创建单个设备
 - 删除设备
 - 获取设备Profile
 - 获取设备View
 - 获取及查询影子列表
 - 获取设备接入详情
 - 更新密钥
 - 更新设备属性
 - 更新单个设备View信息
 - 更新单个设备注册表信息
 - 重置设备影子
- 设备模板管理
 - 创建模板
 - 删除模板
 - 获取模板列表
 - 获取模板
 - 更新模板
- 物管理数据写入TSDB
 - 创建规则
 - 获取规则详情
 - 修改规则
 - 删除规则
 - 禁用一条规则
 - 启用一条规则
 - 创建带TSDB格式的规则

- [获取带TSDB格式的规则详情](#)
- [修改带TSDB格式的规则](#)
- [权限管理](#)
 - [设备权限组管理](#)
- [OTA升级服务](#)
 - [上传固件包文件](#)
 - [创建固件包](#)
 - [获取固件包详情](#)
 - [获取固件包列表](#)
 - [查询设备使用固件包版本](#)
 - [创建升级任务](#)
 - [上传升级设备文件](#)
 - [查询升级任务](#)
 - [查询升级任务列表](#)
 - [查询升级任务结果](#)
 - [查询升级任务各设备升级结果](#)
- [参数定义](#)
 - [DeviceListRequest参数列表](#)
 - [DeviceListResponse参数列表](#)
 - [DeviceProfile参数列表](#)
 - [DeviceProfileResponse参数列表](#)
 - [DeviceBasicInfo参数列表](#)
 - [DeviceAttributes参数列表](#)
 - [DeviceViewResponse参数列表](#)
 - [DeviceViewAttributes参数列表](#)
 - [DeviceAccessDetailResponse参数列表](#)
 - [SchemaProperty参数列表](#)
 - [Schema参数列表](#)

7.2 介绍

7.2.1 简介

百度云IoT物管理，提供用户在云端管理设备的能力。用户能够获取并控制设备状态、进行设备的批量操作以及设备诊断。一方面，设备主动向物管理中心更新状态信息；另一方面，控制端也可以通过和设备管理中心交互反控设备的行为，比如设备状态更新、OTA远程设备升级等。因此，设备管理中心既需要负责和设备端交互，又要负责和控制端交互。设备端的交互主要基于MQTT协议，而控制端通过HTTP通信。IoT Device Management API主要包括控制端的相关功能，以Restful API的形式提供。

V3版本引入模板的概念，用户可以通过模板和视图定义比较关注的设备属性，并将相关数据点写入百度天工时序数据库。此外，在V3版本中，我们分拆并丰富了设备主题，以帮助用户更好的处理设备数据上传以及反控等信息。

7.2.2 签名认证

物管理API会对每个访问的请求进行身份认证，以保障用户的安全。安全认证采用Access Key与请求签名机制。Access Key由Access Key ID和Secret Access Key组成，均为字符串，由百度云官方颁发给用户。其中Access Key ID用于标识用户身份，Access Key Secret 是用于加密签名字串和服务器端验证签名字串的密钥，必须严格保密。

对于每个HTTP请求，用户需要使用下文所描述的方式生成一个签名字串，并将认证字符串放在HTTP请求的Authorization头域里。

签名字串格式

bce-auth-v{version}/{accessKeyId}/{timestamp}/{expireTime}/{signedHeaders}/{signature}

其中：

- version：正整数，目前取值为1。
- timestamp：生成签名时的时间。时间格式符合[通用约定](#)。
- expireTime：签名有效期限，单位为秒，从timestamp所指定的时间开始计算。
- signedHeaders：签名算法中涉及到的头域列表。头域名字之间用分号（;）分隔，如host;x-bce-date。列表按照字典序排列。当signedHeaders为空时表示取默认值。
- signature：256位签名的十六进制表示，由64个小写字母组成，生成方式由如下[签名生成算法](#)给出。

签名生成算法

有关签名生成算法的具体介绍，请参看[鉴权认证机制](#)。

7.2.3 多区域选择

“华南-广州”区域

- 区域的API地址为：iotdm.gz.baidubce.com

“华北-北京”区域

- 区域的API地址为：iotdm.bj.baidubce.com

7.3 设备列表管理

7.3.1 创建单个设备

| 方法 | API | 说明 |
|------|---------------------------|--------|
| POST | /v3/iot/management/device | 创建单个设备 |

请求参数

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|-------------|--------|------|------|
| deviceName | string | 必选 | 设备名称 |
| description | string | 必选 | 描述 |
| schemaId | string | 必选 | 物模型 |

返回参数

一个[DeviceAccessDetailResponse](#)对象

请求示例

```
POST /v3/iot/management/device HTTP/1.1
Host: iotdm.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
{
    "deviceName": "mydevice",
    "description": "device_description",
    "schemaId": "uuid"
}
```

返回示例

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054
{
    "tcpEndpoint": "tcp://test.baidu.iot.com",
    "sslEndpoint": "ssl://test.baidu.iot.com",
    "username": "endpointName/device_1",
```

```

    "key": "bWwCxGwaw3boV48NqsuG+XVaHpxfKdMPvmdJzN0bvbY="
}

```

7.3.2 删除设备

| 方法 | API | 说明 |
|-----|----------------------------------|------|
| PUT | /v3/iot/management/device?remove | 删除设备 |

请求参数

一个[DeviceListRequest](#)对象

返回参数

一个[DeviceListResponse](#)对象

请求示例

```

PUT /v3/iot/management/device?remove HTTP/1.1
Host: iotdm.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
{
  "devices": [
    "device-1",
    "device-2"
  ]
}

```

返回示例

```

HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054
{
  "devices": [
    "device-1",
    "device-2"
  ]
}

```

7.3.3 获取设备Profile

| 方法 | API | 说明 |
|-----|--|-------------|
| GET | /v3/iot/management/device/{deviceName} | 获取设备Profile |

[返回参数](#)

一个[DeviceProfileResponse](#)对象

[请求示例](#)

```
GET /v3/iot/management/device/mydevice HTTP/1.1
Host: iotdm.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
```

[返回示例](#)

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054
{
  "name": "mydevice",
  "description": "my device in Shanghai",
  "createTime": 1494904250,
  "state": "online",
  "lastActiveTime": 1494904250,
  "schemaId": "uuid",
  "schemaName": "baidu_shanghai",
  "favourite": false,
  "attributes": {
    "region": "Shanghai"
  },
  "device": {
    "reported": {
      "firewareVersion": "1.0.0",
      "light": "green"
    },
    "desired": {
      "light": "red"
    }
  }
}
```

```

    "lastUpdatedTime": {
        "reported": {
            "firewareVersion": 1494904250,
            "light": 1494904250
        },
        "desired": {
            "light": 1494904250
        }
    },
    "profileVersion": 10
}
}

```

7.3.4 获取设备View

| 方法 | API | 说明 |
|-----|--|------------------------|
| GET | /v3/iot/management/deviceView/{deviceName} | 获取设备Profile和模型合并后的View |

返回参数

一个[DeviceViewResponse](#)对象

请求示例

```

GET /v3/iot/management/deviceView/mydevice HTTP/1.1
Host: iotdm.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8

```

返回示例

```

HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054
{
    "name": "mydevice",
    "description": "my device in Shanghai",
    "createTime": 1326789000,
    "state": "online",
    "lastActiveTime": 1326789000,
}

```

```

"schemaId": "uuid",
"schemaName": "baidu_shanghai",
" favourite": false,
"profileVersion": 10,
"properties": [
{
  "attributeName": "light",
  "showName": "灯光",
  "type": "string",
  "defaultValue": "red",
  "reportedValue": "green",
  "desiredValue": "green"
  "unit": "-",
  "reportedTime": 1326789000,
  "desiredTime": 1435860000
}
]
}

```

7.3.5 获取及查询影子列表

| 方法 | API | 说明 |
|-----|--|-------------|
| GET | /v3/iot/management/device? pageNo=xx&pageSize=xx&orderBy =xx&&order=xx&name=xx&value=xx&favourite=xx | 查询设备Profile |

请求参数

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|----------|--------|------|--|
| pageNo | int | 可选 | 表示取第几页， 默认1 |
| pageSize | int | 可选 | 每页返回的数目， 默认10 |
| orderBy | String | 可选 | name, createTime, lastActiveTime, 默认name |
| order | String | 可选 | 按升序或降序返回结果, desc / asc, 默认asc |

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|-----------|--------|------|-------------------------------|
| name | String | 可选 | 查询属性名, 默认全量 |
| value | String | 可选 | 查询属性名对应的值, 默认全量 |
| favourite | String | 可选 | 收藏, true / false / all, 默认all |

返回参数

| 参数名称 | 参数类型 | 说明 |
|----------|------|-----------------------|
| devices | List | 由DeviceProfile对象组成的列表 |
| amount | int | 总数目 |
| pageNo | int | 返回页数 |
| pageSize | int | 每页返回数目 |

请求示例

```
GET /v3/iot/management/device?pageNo=1&pageSize=10&orderBy=createTime&&order=desc&name=schemaName
1.1
Host: iotdm.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
```

返回示例

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054
{
    "amount": 20,
    "pageNo": 1,
    "pageSize": 10,
    "devices": [
        {
            "name": "mydevice",
            "description": "my description",
            "state": "online",
```

```

    "createTime": 1326789000,
    "lastActiveTime": 1326789000,
    "schemaId": "uuid",
    "schemaName": "baidu_shanghai",
    "favourite": false,
    "attributes": {
        "region": "Shanghai"
    },
    "device": {
        "reported": {
            "light": "green"
        },
        "desired": {
            "light": "red"
        }
    },
    "profileVersion": 10,
    "lastUpdatedTime": {
        "reported": {
            "light": 1326789000
        },
        "desired": {
            "light": 1326789000
        }
    }
}
...
]
}

```

7.3.6 获取设备接入详情

| 方法 | API | 说明 |
|-----|--|----------|
| GET | /v3/iot/management/device/{deviceName}/access-Detail | 获取设备接入详情 |

[返回参数](#)

一个[DeviceAccessDetailResponse](#)实例

[请求示例](#)

```
GET /v3/iot/management/device/mydevice/accessDetail HTTP/1.1
Host: iotdm.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
```

[返回示例](#)

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054
{
  "tcpEndpoint": "tcp://test.baidu.iot.com",
  "sslEndpoint": "ssl://test.baidu.iot.com",
  "username": "endpointName/device_1",
  "key": "xxxxxxxxxx"
}
```

7.3.7 更新密钥

| 方法 | API | 说明 |
|-----|--|------|
| PUT | / v3/ iot/ management/ device/ {deviceName}? updateSecretKey | 更改密钥 |

[请求参数](#)

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|------------|--------|------|------|
| deviceName | String | 必须 | 设备名称 |

[返回参数](#)

一个[DeviceAccessDetailResponse](#)实例

[请求示例](#)

```
PUT /v3/iot/management/device/deviceName?updateSecretKey HTTP/1.1
Host: iotdm.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
```

返回示例

```

HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054
{
  "tcpEndpoint": "tcp://test.baidu.iot.com",
  "sslEndpoint": "ssl://test.baidu.iot.com",
  "username": "endpointName/deviceName",
  "key": "bWwCxGwaw3boV48NqsuG+XVaHpxfKdMPvmdJzN0vbY="
}

```

7.3.8 更新设备属性

| 方法 | API | 说明 |
|-----|--|--------|
| PUT | / v3/ iot/ management/ device/ {deviceName}? updateProfile | 修改设备属性 |

请求参数

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|------------|------------------|------|------------------|
| attributes | JSONObject | 可选 | 需要更新的 attributes |
| device | DeviceAttributes | 可选 | 需要更新的 device 属性 |

返回参数

一个[DeviceProfileResponse](#)对象

请求示例

```

PUT /v3/iot/management/device/myDevice?updateProfile HTTP/1.1
Host: iotdm.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
{
  "device": {
    "desired": {
      ...
    }
  }
}

```

```
        "light": "red"
    },
    "reported": {
        "light": "green"
    }
},
"attributes": {
    "region": "Shanghai"
}
}
```

[返回示例](#)

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054
{
    "name": "mydevice",
    "description": "my device in Shanghai",
    "createTime": 1494904250,
    "state": "online",
    "lastActiveTime": 1494904250,
    "schemaId": "uuid",
    "schemaName": "baidu_shanghai",
    "favourite": false,
    "attributes": {
        "region": "Shanghai"
    },
    "device": {
        "reported": {
            "firewareVersion": "1.0.0",
            "light": "green"
        },
        "desired": {
            "light": "red"
        },
        "lastUpdatedTime": {
            "reported": {
                "firewareVersion": 1494904250,
                "light": 1494904250
            },
            "desired": {
                "light": 1494904250
            }
        }
    }
}
```

```

    },
    "profileVersion": 10
}
}

```

7.3.9 更新单个设备View信息

| 方法 | API | 说明 |
|-----|---|------------|
| PUT | /v3/iot/management/deviceView/{deviceName}?updateView | 修改设备View信息 |

请求参数

| 请求名称 | 参数类型 | 是否必须 | 说明 |
|----------------|----------|------|----------------|
| reported | JsonNode | 可选 | 需要更新的 reported |
| desired | JsonNode | 可选 | 需要更新的 desired |
| profileVersion | int | 可选 | 更新版本 |

返回参数

一个[DeviceViewResponse](#)对象

请求示例

```

PUT /v3/iot/management/deviceView/myDevice?updateView HTTP/1.1
Host: iotdm.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
{
  "desired": {
    "light": "red"
  },
  "reported": {
    "light": "green"
  }
}

```

返回示例

```

HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054
{
    "name": "myDevice",
    "description": "my device in Shanghai",
    "createTime": 1326789000,
    "state": "online",
    "lastActiveTime": 1326789000,
    "schemaId": "uuid",
    "schemaName": "baidu_shanghai",
    "favourite": false,
    "profileVersion": 1,
    "properties": [
        {
            "attributeName": "light",
            "showName": "灯光",
            "type": "String",
            "defaultValue": "red",
            "reportedValue": "green",
            "desiredValue": "red",
            "unit": "-",
            "reportedTime": 1435860000,
            "desiredTime": 1435860000
        }
    ]
}

```

7.3.10 更新单个设备注册表信息

| 方法 | API | 说明 |
|-----|---|-------------|
| PUT | /v3/iot/management/device/{deviceName}?updateRegistry | 更新单个设备注册表信息 |

请求参数

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|-------------|--------|------|-----------|
| description | String | 可选 | 需要更新的设备描述 |
| schemaId | String | 可选 | 需要更换的模板Id |

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|----------|---------|------|--------|
| avourite | boolean | 可选 | 是否需要收藏 |

[返回参数](#)

一个[DeviceProfileResponse](#)对象

[请求示例](#)

```
PUT /v3/iot/management/device/myDevice?updateRegistry HTTP/1.1
Host: iotdm.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
{
  "description": "new device description",
  "schemaId": "uuid",
  "favourite": true,
}
```

[返回示例](#)

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054
{
  "name": "mydevice",
  "description": "new device description",
  "createTime": 1494904250,
  "state": "online",
  "lastActiveTime": 1494904250,
  "schemaId": "uuid",
  "schemaName": "baidu_shanghai",
  "favourite": true,
  "attributes": {
    "region": "Shanghai"
  },
  "device": {
    "reported": {
      "firewareVersion": "1.0.0",
      "light": "green"
    },
    "desired": {

```

```

        "light": "red"
    },
    "lastUpdatedTime": {
        "reported": {
            "firewareVersion": 1494904250,
            "light": 1494904250
        },
        "desired": {
            "light": 1494904250
        }
    },
    "profileVersion": 10
}
}

```

7.3.11 重置设备影子

| 方法 | API | 说明 |
|-----|---------------------------------|------------|
| PUT | /v3/iot/management/device?reset | 重置（清空）设备影子 |

请求参数

一个[DeviceListRequest](#)对象

返回参数

一个[DeviceListResponse](#)对象

请求示例

```

PUT /v3/iot/management/device?reset HTTP/1.1
Host: iotdm.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
{
    "devices": [
        "device-1",
        "device-2"
    ]
}

```

返回示例

```

HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054
{
    "devices": [
        "device-1",
        "device-2"
    ]
}

```

7.4 设备模板管理

在新版物管理中，我们引入了模板的概念，模板定义了一类设备的schema。Schema中定义了各设备各属性的显示名称、类型、默认值等信息。可以理解为，通过模板，我们定义了设备的视图(view)。此外，我们需要支持模板的CRUD操作。IoT Device Management Schema API 主要包含管理设备模板的相关接口。

7.4.1 创建模板

| **方法** | **API** | **说明** |
|--------|---------------------------|--------|
| POST | /v3/iot/management/schema | 创建模板 |

请求参数

| **参数名称** | **参数类型** | **是否必须** | **说明** |
|-------------|----------------------|----------|--------|
| name | String | 必须 | 模板名称 |
| description | String | 可选 | 模板说明 |
| properties | List<SchemaProperty> | 必须 | 模板属性列表 |

返回参数

| **参数名称** | **参数类型** | **说明** |
|----------|----------|--------|
| schemaId | String | 模板ID |

请求示例

```

POST /v3/iot/management/schema HTTP/1.1
Host:iotdm.gz.baidubce.com
Authorization:{authorization}
Content-Type: application/json; charset=utf-8
{
  "name": "myFirstSchema",
  "description": "description",
  "properties": [
    {
      "name": "temperature",
      "type": "number",
      "displayName": "温度",
      "defaultValue": "38.2",
      "unit": "deg"
    },
    {
      "name": "pressure",
      "type": "number",
      "displayName": "压力",
      "defaultValue": "9",
      "unit": "MPa"
    }
  ]
}

```

[返回示例](#)

```

HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054
{
  "schemaid": "ab13ef935b84173a0f41f90c33daa87"
}

```

7.4.2 删除模板

| **方法** | **API** | **说明** |
|--------|--|----------------|
| DELETE | / v3/ iot/ management/ schema/{schemaId} | 根据schemaId删除模板 |

[请求示例](#)

```
DELETE /v3/iot/management/schema/mySchema HTTP/1.1
Host:iotdm.gz.baidubce.com
Authorization:{authorization}
```

[返回示例](#)

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5dddac970054
```

注：删除模板，要求没有设备引用该模板。否则，会抛出异常。

7.4.3 获取模板列表

| **方法** | **API** | **说明** |
|--------|--|------------|
| GET | GET / v3/ iot/ management/ schema? pageNo=xx&pageSize=xx&order=xx&orderBy=xx&key=xx | 根据条件查询模板列表 |

[请求参数](#)

| **参数名称** | **参数类型** | **是否必须** | **说明** |
|----------|----------|----------|---|
| pageNo | int | 可选 | 获取列表在查询结果的页码，默认为1 |
| pageSize | int | 可选 | 一页所包含的最大模板数量，默认为10 |
| order | String | 可选 | 查询结果升序或降序排列，asc desc，默认asc |
| orderBy | String | 可选 | 排序的索引列，name createTime lastUpdatedTime，默认name |
| key | String | 可选 | 查询关键字，模板名称 |

[返回参数](#)

| **参数名称** | **参数类型** | **说明** |
|------------|--------------|--------------|
| totalCount | int | 模板总数 |
| pageNo | int | 当前页码 |
| pageSize | int | 一页所包含的最大模板数量 |
| result | List<Schema> | 模板参数列表 |

[请求示例](#)

```
GET /v3/iot/management/schema?pageNo=1&pageSize=200&order=asc&orderBy=name&key=myFirstSchema HTTP/1.1
Host:iotdm.gz.baidubce.com
Authorization:{authorization}
```

[返回示例](#)

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054
{
    "totalCount": 1,
    "pageNo": 1,
    "pageSize": 200,
    "result": [
        {
            "id": "1234939554",
            "name": "myFirstSchema",
            "description": " desc",
            "createTime": 1494904250,
            "lastUpdatedTime": 1494904250,
            "properties": [
                {
                    "name": "temperature",
                    "type": "number",
                    "displayName": "温度",
                    "defaultValue": "38.2",
                    "unit": "deg"
                },
                {
                    "name": "pressure",
                    "type": "number",

```

```

        "displayName": "压力",
        "defaultValue": "9",
        "unit": "MPa"
    }
]
}
]
}

```

7.4.4 获取模板

| **方法** | **API** | **说明** |
|--------|---|--------|
| GET | /v3/iot/management/schema/{schema_id} 根据模板ID获取模板详情 | |

[返回参数](#)

一个[Schema](#)实例

[请求示例](#)

```

GET /v3/iot/management/schema/1234939554 HTTP/1.1
Host:iotdm.gz.baidubce.com
Authorization:{authorization}

```

[返回示例](#)

```

HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054
{
    "id": "1234939554",
    "name": "myFirstSchema",
    "description": " desc",
    "createTime": 1494904250,
    "lastUpdatedTime": 1494904250,
    "properties": [
        {
            "name": "temperature",
            "type": "number",

```

```

        "displayName": "温度",
        "defaultValue": "38.2",
        "unit": "deg"
    },
    {
        "name": "pressure",
        "type": "number",
        "displayName": "压力",
        "defaultValue": "9",
        "unit": "MPa"
    }
]
}

```

7.4.5 更新模板

| **方法** | **API** | **说明** |
|--------|---|--------|
| PUT | /v3/iot/management/schema/{schema_id} 根据模板ID更新设备模板 | |

请求参数

| **参数名称** | **参数类型** | **是否必须** | **说明** |
|-------------|----------------------|----------|--------|
| description | String | 可选 | 模板说明 |
| properties | List<SchemaProperty> | 可选 | 模板属性列表 |

请求示例

```

PUT /v3/iot/management/schema/1234939554 HTTP/1.1
Host:iotdm.gz.baidubce.com
Authorization:{authorization}
{
    "description": "new description",
    "properties": [
        {
            "name": "temperature",
            "type": "number",
            "displayName": "温度",
            "defaultValue": "38.3",
            "unit": "deg"
        }
    ]
}

```

```

        },
        {
            "name": "pressure",
            "type": "number",
            "displayName": "压力",
            "defaultValue": "9",
            "unit": "MPa"
        },
        {
            "name": "speed",
            "type": "number",
            "displayName": "速度",
            "defaultValue": "5",
            "unit": "mps"
        }
    ]
}

```

[返回示例](#)

HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054

7.5 物管理数据写入TSDB

7.5.1 创建规则

| 方法 | API | 说明 |
|------|-----------------------------------|-----------|
| POST | /v3/iot/rules/device/{deviceName} | 创建规则引擎的规则 |

[请求参数](#)

DeviceRuleRequest

[返回参数](#)

DeviceRuleResponse

[请求示例](#)

```
POST /v3/iot/rules/device/myDeviceName HTTP/1.1
Host: iotdm.gz.baidubce.com
Authorization:{authorization}
Content-Type: application/json; charset=utf-8

{
    "name": "device xxxx to TSDB yyyy",
    "sources": [{"description": "This is condition 1",
        "name": "name",
        "type": "string",
        "condition": "<>",
        "value": "aaa"}, {"description": "", // 可以为空
        "name": "temperature",
        "type": "number",
        "condition": ">=",
        "value": "20"}, {"description": "This is condition 3",
        "name": "speed",
        "type": "number",
        "condition": ">",
        "value": "0"}],
    "destinations": [{"value": "test.tsdb.iot.gz.baidubce.com",
        "kind": "TSDB"}]
}
```

[返回示例](#)

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054

{
    "id": "63d92c1de2bd46e0b257c6df67b4a7e9",
```

```
"deviceName": "myDeviceName",
"name": "device xxxx to TSDB yyyy",
"sources": [{
    "description": "This is condition 1"
    "name": "name",
    "type": "string",
    "displayName": "名字",
    "unit": "count",
    "defaultValue": "0",
    "condition": "<>",
    "value": "aaa",
    "lastSaveTime": 0
}, {
    "description": "",
    "name": "temperature",
    "type": "number",
    "displayName": "数量",
    "unit": "piece",
    "defaultValue": "1",
    "condition": ">=",
    "value": "20",
    "lastSaveTime": 1494904250
}, {
    "description": "This is condition 3",
    "name": "speed",
    "type": "number",
    "displayName": "速度",
    "unit": "km/s",
    "defaultValue": "60",
    "condition": ">",
    "value": "0",
    "lastSaveTime": 1494904250
}, { // 这个规则是来自Schema里的新的属性，用户在之前的规则建立的时候并没有创建对应的规则。 如果创建的规则在Schema里没有对应的属性，则那条规则不会显示
    "description": "",
    "name": "temp",
    "type": "number",
    "displayName": "温度",
    "unit": "c",
    "defaultValue": "36",
    "condition": "",
    "value": "",
    "lastSaveTime": 0
}],
"destinations": [
```

```
{
    "uuid": "6653da99bf9a4e35ba4f997e000a699f",
    "value": "test.tsdb.iot.gz.baidubce.com",
    "kind": "TSDB"
}
],
"enable": true,
"createTime": 1494904250,
"updateTime": 1494904250
}
```

7.5.2 获取规则详情

请求参数

| 方法 | API | 说明 |
|-----|-----------------------------------|-------------|
| GET | /v3/iot/rules/device/{deviceName} | 获取单个规则的规则详情 |

返回参数

DeviceRuleResponse

请求示例

```
GET /v3/iot/rules/device/myDeviceName HTTP/1.1
Host: iotdm.gz.baidubce.com
Authorization:{authorization}
Content-Type: application/json; charset=utf-8
```

返回示例

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054
```

```
{
    "id": "63d92c1de2bd46e0b257c6df67b4a7e9",
    "deviceName": "myDeviceName",
    "name": "device xxxx to TSDB yyyy",
```

```
"sources": [{

    "description": "This is condition 1",
    "name": "name",
    "type": "string",
    "displayName": "名字",
    "unit": "count",
    "defaultValue": "0",
    "condition": "<>",
    "value": "aaa",
    "lastSaveTime": 0
}, {
    "description": "This is condition 2",
    "name": "temperature",
    "type": "number",
    "displayName": "数量",
    "unit": "piece",
    "defaultValue": "1",
    "condition": ">=",
    "value": "20",
    "lastSaveTime": 1494904250
}, {
    "description": "This is condition 3",
    "name": "speed",
    "type": "number",
    "displayName": "速度",
    "unit": "km/s",
    "defaultValue": "60",
    "condition": ">",
    "value": "0",
    "lastSaveTime": 1494904250
}, {
    "description": "",
    "name": "temp",
    "type": "number",
    "displayName": "温度",
    "unit": "c",
    "defaultValue": "36",
    "condition": "",
    "value": "",
    "lastSaveTime": 0
}

], "destinations": [
```

```
{
    "uuid": "6653da99bf9a4e35ba4f997e000a699f",
    "value": "test.tsdb.iot.gz.baidubce.com",
    "kind": "TSDB"
}
],
"enable": true,
"createTime": 1494904250,
"updateTime": 1494904250
}
```

7.5.3 修改规则

| 方法 | API | 说明 |
|-----|-----------------------------------|--------|
| PUT | /v3/iot/rules/device/{deviceName} | 修改单个规则 |

请求参数

| 参数名 | 参数类型 | 是否必须 | 说明 | 示例 |
|--------------|-----------------------------|------|--|----------------|
| name | String | 可选 | 规则名称 | 过滤传感器温度过高消息的规则 |
| sources | List<DeviceRuleSource> | 可选 | 需要存储的数据的属性和条件 | |
| destinations | List<DeviceRuleDestination> | 可选 | 处理后的消息写往的目的地数组 (TSDB, KAFKA, 另一个 MQTT 主题) 目前只支持 TSDB | |

返回参数

DeviceRuleResponse

请求示例

```
PUT /v3/iot/rules/device/myDeviceName HTTP/1.1
```

```
Host: iotdm.gz.baidubce.com
Authorization:{authorization}
Content-Type: application/json; charset=utf-8

// 全修改
{
    "name": "device xxxx to TSDB yyyy",
    "sources": [
        {
            "description": "",
            "name": "name",
            "type": "string",
            "condition": "<>",
            "value": "bbbbbb"
        }, {
            "description": "This is condition 2",
            "name": "temperature",
            "type": "number",
            "condition": "<=",
            "value": "100000"
        }],
    "destinations": [
        {
            "value": "TSDB_NAME", // e.g. test.tsdb.iot.gz.baidubce.com
            "kind": "TSDB"
        }
    ]
}

// 只修改名字
{
    "name": "cccc" // 亦可为空
}
{
    "name": "cccc",
    "sources": [] // 亦可为空
}

// 只修改 sources
{
    "sources": [
        {
            "description": "",
            "name": "name",
            "type": "string",
            "condition": "<>",
            "value": "bbbbbb"
        }
    ]
}
```

```
    },
    {
        "description": "This is condition 2",
        "name": "temperature",
        "type": "number",
        "condition": "<=",
        "value": "100000"
    }]
}

// 只修改 destinations
{
    "destinations": [
        {
            "value": "TSDB_NAME", // e.g. test.tsdb.iot.gz.baidubce.com
            "kind": "TSDB"
        }
    ]
}
```

[返回示例](#)

HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054

```
{
    "id": "63d92c1de2bd46e0b257c6df67b4a7e9",
    "deviceName": "myDeviceName",
    "name": "device xxxx to TSDB yyyy",
    "sources": [
        {
            "description": "This is condition 1",
            "name": "name",
            "type": "string",
            "displayName": "名字",
            "unit": "count",
            "defaultValue": "0",
            "condition": "<>",
            "value": "aaa",
            "lastSaveTime": 0
        },
        {
            "description": "This is condition 2",
            "name": "temperature",
            "type": "number",
            "condition": "<=",
            "value": "100000"
        }
    ]
}
```

```
        "name": "temperature",
        "type": "number",
        "displayName": "数量",
        "unit": "piece",
        "defaultValue": "1",
        "condition": ">=",
        "value": "20",
        "lastSaveTime": 1494904250
    }, {
        "description": "This is condition 3",
        "name": "speed",
        "type": "number",
        "displayName": "速度",
        "unit": "km/s",
        "defaultValue": "60",
        "condition": ">",
        "value": "0",
        "lastSaveTime": 1494904250
    }, { // 这个规则是来自Schema里的新的属性，用户在之前的规则建立的时候并没有创建对应的规则。如果创建的规则在Schema里没有对应的属性，则那条规则不会显示
        "description": "",
        "name": "temp",
        "type": "number",
        "displayName": "温度",
        "unit": "c",
        "defaultValue": "36",
        "condition": "",
        "value": "",
        "lastSaveTime": 0
    }],
    "destinations": [
        {
            "uuid": "6653da99bf9a4e35ba4f997e000a699f",
            "value": "test.tsdb.iot.gz.baidubce.com",
            "kind": "TSDB"
        }
    ],
    "enable": true,
    "createTime": 1494904250,
    "updateTime": 1494904250
}
```

7.5.4 删除规则

| 方法 | API | 说明 |
|--------|-----------------------------------|------|
| DELETE | /v3/iot/rules/device/{deviceName} | 删除规则 |

请求示例

```
DELETE /v3/iot/rules/device/myDeviceName HTTP/1.1
Host: iotdm.gz.baidubce.com
Authorization: {authorization}
Content-Type: application/json; charset=utf-8
```

返回示例

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054
```

7.5.5 禁用一条规则

| 方法 | API | 说明 |
|-----|---|--------|
| PUT | /v3/iot/rules/device/{deviceName}?disable | 禁用一条规则 |

请求示例

```
PUT /v3/iot/rules/device/myDeviceName?disable HTTP/1.1
Host: iotdm.gz.baidubce.com
Authorization:{authorization}
Content-Type: application/json; charset=utf-8
```

返回示例

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054
```

7.5.6 启用一条规则

| 方法 | API | 说明 |
|-----|--|--------|
| PUT | /v3/iot/rules/device/{deviceName}?enable | 启用一条规则 |

请求示例

```
PUT /v3/iot/rules/device/myDeviceName?enable HTTP/1.1
Host: iotdm.gz.baidubce.com
Authorization:{authorization}
Content-Type: application/json; charset=utf-8
```

返回示例

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054
```

7.5.7 创建带TSDB格式的规则

| 方法 | API | 说明 |
|------|--|----------------|
| POST | /v3/iot/rules/device/{deviceName}/format | 创建带TSDB格式的转存规则 |

请求参数

[DeviceFormatRuleRequest](#)

返回参数

[DeviceFormatRuleResponse](#)

请求示例

```
POST /v3/iot/rules/device/myDeviceName/format HTTP/1.1
Host: iotdm.gz.baidubce.com
Authorization:{authorization}
Content-Type: application/json; charset=utf-8
```

```
{  
    "name": "device xxxx to TSDB yyyy",  
    "sources": [  
        {"description": "This is condition 1",  
         "name": "name",  
         "type": "string",  
         "condition": "<>",  
         "value": "aaa"},  
        {"description": "", // 可以为空  
         "name": "temperature",  
         "type": "number",  
         "condition": ">=",  
         "value": "20"},  
        {"description": "This is condition 3",  
         "name": "speed",  
         "type": "number",  
         "condition": ">",  
         "value": "0"}],  
    "destinations": [  
        {"value": "test.tsdb.iot.gz.baidubce.com",  
         "kind": "TSDB"}],  
    "format": {  
        "mode": "field",  
        "metric": "newMetricName",  
        "tags": {"tag1": "aaa"}  
    }  
}
```

[返回示例](#)

HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054

{

```
"id": "63d92c1de2bd46e0b257c6df67b4a7e9",
"deviceName": "myDeviceName",
"name": "device xxxx to TSDB yyyy",
"sources": [
    {
        "description": "This is condition 1"
        "name": "name",
        "type": "string",
        "displayName": "名字",
        "unit": "count",
        "defaultValue": "0",
        "condition": "<>",
        "value": "aaa",
        "lastSaveTime": 0
    }, {
        "description": "This is condition 2",
        "name": "temperature",
        "type": "number",
        "displayName": "数量",
        "unit": "piece",
        "defaultValue": "1",
        "condition": ">=",
        "value": "20",
        "lastSaveTime": 1494904250
    }, {
        "description": "This is condition 3",
        "name": "speed",
        "type": "number",
        "displayName": "速度",
        "unit": "km/s",
        "defaultValue": "60",
        "condition": ">",
        "value": "0",
        "lastSaveTime": 1494904250
    }, { // 这个规则是来自Schema里的新的属性，用户在之前的规则建立的时候并没有创建对应的规则。 如果创建的规则在Schema里没有对应的属性，则那条规则不会显示
        "description": "",
        "name": "temp",
        "type": "number",
        "displayName": "温度",
        "unit": "c",
        "defaultValue": "36",
        "condition": "",
        "value": "",
        "lastSaveTime": 0
    }
], {
```

```

        },
        "destinations": [
            {
                "uuid": "6653da99bf9a4e35ba4f997e000a699f",
                "value": "test.tsdb.iot.gz.baidubce.com",
                "kind": "TSDB"
            }
        ],
        "format" : {
            "mode" : "field",
            "metric" : "newMetricName",
            "tags" : {
                "tag1" : "aaa"
            }
        },
        "enable": true,
        "createTime": 1494904250,
        "updateTime": 1494904250
    }
}

```

7.5.8 获取带TSDB格式的规则详情

请求参数

| 方法 | API | 说明 |
|-----|--|-------------|
| GET | /v3/iot/rules/device/{deviceName}/format | 获取单个规则的规则详情 |

返回参数

DeviceFormatRuleResponse

请求示例

```

GET /v3/iot/rules/device/myDeviceName/format HTTP/1.1
Host: iotdm.gz.baidubce.com
Authorization:{authorization}
Content-Type: application/json; charset=utf-8

```

返回示例

HTTP/1.1 200 OK

```
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054
```

```
{
    "id": "63d92c1de2bd46e0b257c6df67b4a7e9",
    "deviceName": "myDeviceName",
    "name": "device xxxx to TSDB yyyy",
    "sources": [
        {
            "description": "This is condition 1",
            "name": "name",
            "type": "string",
            "displayName": "名字",
            "unit": "count",
            "defaultValue": "0",
            "condition": "<>",
            "value": "aaa",
            "lastSaveTime": 0
        }, {
            "description": "This is condition 2",
            "name": "temperature",
            "type": "number",
            "displayName": "数量",
            "unit": "piece",
            "defaultValue": "1",
            "condition": ">=",
            "value": "20",
            "lastSaveTime": 1494904250
        }, {
            "description": "This is condition 3",
            "name": "speed",
            "type": "number",
            "displayName": "速度",
            "unit": "km/s",
            "defaultValue": "60",
            "condition": ">",
            "value": "0",
            "lastSaveTime": 1494904250
        }, { // 这个规则是来自Schema里的新的属性，用户在之前的规则建立的时候并没有创建对应的规则。 如果创建的规则在Schema里没有对应的属性，则那条规则不会显示
            "description": "",
            "name": "temp",
            "type": "number",
            "displayName": "温度",
            "unit": "c",
            "defaultValue": "36",
        }
    ]
}
```

```

    "condition": "",
    "value": "",
    "lastSaveTime": 0

  }],
  "destinations": [
    {
      "uuid": "6653da99bf9a4e35ba4f997e000a699f",
      "value": "test.tsdb.iot.gz.baidubce.com",
      "kind": "TSDB"
    }
  ],
  "format" : {
    "mode" : "field",
    "metric" : "newMetricName",
    "tags" : {
      "tag1" : "aaa"
    }
  },
  "enable": true,
  "createTime": 1494904250,
  "updateTime": 1494904250
}

```

7.5.9 修改带TSDB格式的规则

| 方法 | API | 说明 |
|-----|--|--------|
| PUT | /v3/iot/rules/device/{deviceName}/format | 修改单个规则 |

请求参数

| 参数名 | 参数类型 | 是否必须 | 说明 | 示例 |
|---------|------------------------|------|----------------|----------------|
| name | String | 可选 | 规则名称 | 过滤传感器温度过高消息的规则 |
| sources | List<DeviceRuleSource> | 可选 | 需要存储的数据的属性和其条件 | |

| 参数名 | 参数类型 | 是否必须 | 说明 | 示例 |
|--------------|-----------------------------|------|---|--|
| destinations | List<DeviceRuleDestination> | 可选 | 处理后的消息写往的目的地数组 (TSDB, KAFKA, 另一个 MQTT 主题) 目前只支持 TSDB | |
| format | DeviceRuleFormat | 可选 | 转存TSDB数据的格式定义 | { "mode": "field", "metric": "newMetricName" } |

[返回参数](#)[DeviceFormatRuleResponse](#)[请求示例](#)

```

PUT /v3/iot/rules/device/myDeviceName/format HTTP/1.1
Host: iotdm.gz.baidubce.com
Authorization:{authorization}
Content-Type: application/json; charset=utf-8

// 全修改
{
    "name": "device xxxx to TSDB yyyy",
    "sources": [
        {
            "description": "",
            "name": "name",
            "type": "string",
            "condition": "<>",
            "value": "bbbbbb"
        },
        {
            "description": "This is condition 2",
            "name": "temperature",
            "type": "number",
            "condition": "<=",
            "value": "100000"
        }
    ],
    "destinations": [
        {
            "value": "TSDB_NAME", // e.g. test.tsdb.iot.gz.baidubce.com
            "kind": "TSDB"
        }
    ]
}

```

```
        }
    ],
    "format" : {
        "mode" : "field",
        "metric" : "newMetricName",
        "tags" : {
            "tag1" : "aaa"
        }
    }
}

// 只修改名字
{
    "name": "cccc" // 亦可为空
}
{
    "name": "cccc",
    "sources": [] // 亦可为空
}

// 只修改 sources
{
    "sources": [
        {
            "description": "",
            "name": "name",
            "type": "string",
            "condition": "<>",
            "value": "bbbbbb"
        },
        {
            "description": "This is condition 2",
            "name": "temperature",
            "type": "number",
            "condition": "<=",
            "value": "100000"
        }
    ]
}

// 只修改 destinations
{
    "destinations": [
        {
            "value": "TSDB_NAME", // e.g. test.tsdb.iot.gz.baidubce.com
            "kind": "TSDB"
        }
    ]
}
```

```
]  
}  
  
// 只修改 format  
{  
    "format": {  
        "mode": "field",  
        "metric": "newMetricName",  
        "tags": {  
            "tag1": "aaa"  
        }  
    }  
}
```

[返回示例](#)

HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5dddac970054

```
{  
    "id": "63d92c1de2bd46e0b257c6df67b4a7e9",  
    "deviceName": "myDeviceName",  
    "name": "device xxxx to TSDB yyyy",  
    "sources": [{  
        "description": "This is condition 1",  
        "name": "name",  
        "type": "string",  
        "displayName": "名字",  
        "unit": "count",  
        "defaultValue": "0",  
        "condition": "<>",  
        "value": "aaa",  
        "lastSaveTime": 0  
    }, {  
        "description": "This is condition 2",  
        "name": "temperature",  
        "type": "number",  
        "displayName": "数量",  
        "unit": "piece",  
        "defaultValue": "1",  
        "condition": ">=",  
        "value": "20",  
        "lastSaveTime": 1494904250  
    }]
```

```

    },
    {
        "description": "This is condition 3",
        "name": "speed",
        "type": "number",
        "displayName": "速度",
        "unit": "km/s",
        "defaultValue": "60",
        "condition": ">",
        "value": "0",
        "lastSaveTime": 1494904250
    }, { // 这个规则是来自Schema里的新的属性，用户在之前的规则建立的时候并没有创建对应的规则。如果创建的规则在Schema里没有对应的属性，则那条规则不会显示
        "description": "",
        "name": "temp",
        "type": "number",
        "displayName": "温度",
        "unit": "c",
        "defaultValue": "36",
        "condition": "",
        "value": "",
        "lastSaveTime": 0
    }],
    "destinations": [
        {
            "uuid": "6653da99bf9a4e35ba4f997e000a699f",
            "value": "test.tsdb.iot.gz.baidubce.com",
            "kind": "TSDB"
        }
    ],
    "format" : {
        "mode" : "field",
        "metric" : "newMetricName",
        "tags" : {
            "tag1" : "aaa"
        }
    },
    "enable": true,
    "createTime": 1494904250,
    "updateTime": 1494904250
}

```

7.5.10 参数定义

| 参数名 | 参数类型 | 说明 | 示例 |
|--------------|-----------------------------|--|---|
| name | String | 规则名称， 默认为空 | 规则名称1 |
| sources | List<DeviceRuleSource> | 必须， 需要存储的数据的属性和其条件 | |
| destinations | List<DeviceRuleDestination> | 必须， 处理后的消息写往的目的地数组 (TSDB, KAFKA, 另一个MQTT主题) 目前只支持TSDB | [{ "kind" : "TSDB" , "value" : "test.tsdb.iot.gz.baidubce.com" }] |
| format | DeviceRuleFormat | 必须， 转存TSDB数据的格式定义 | |

DeviceFormatRuleRequest

| 参数名 | 参数类型 | 说明 | 示例 |
|--------------|-----------------------------|--|---|
| name | String | 规则名称， 默认为空 | 规则名称1 |
| sources | List<DeviceRuleSource> | 必须， 需要存储的数据的属性和其条件 | |
| destinations | List<DeviceRuleDestination> | 必须， 处理后的消息写往的目的地数组 (TSDB, KAFKA, 另一个MQTT主题) 目前只支持TSDB | [{ "kind" : "TSDB" , "value" : "test.tsdb.iot.gz.baidubce.com" }] |

DeviceRuleRequest

| 参数名 | 参数类型 | 说明 | 示例 |
|-------------|--------|--------------------------------------|-------------|
| description | String | 非必须， 默认为空， 条件的描述信息， 最长 255 个字符， 默认为空 | 规则名称1 |
| name | String | 必须， 对应的模版的属性名称 | temperature |

| 参数名 | 参数类型 | 说明 | 示例 |
|-----------|--------|---|--------|
| type | String | 必须, 属性数据类型, 对应于Schema的数据类型, string, number, bool, object (与TSDB的数据类型的对应类型分别为: String, Double, Long, String) | string |
| value | String | 非必需, 默认为空, 条件对应的阈值 | 37 |
| condition | String | 非必需, 默认为空 (即表示没有约束条件), 约束条件运算符 | >= |

DeviceRuleSource 约束条件运算符包括:

运算符 | 描述

----- | -----> | 大于 >= | 大于等于

< | 小于

<= | 小于等于

= | 等于

<> | 不等于 * | 表示有数据即存储, 此属性数据都会存到TSDB里, 此时条件对应的阈值也无效

| 空, 即表示没有约束条件, 此项数据不作转发处理, 此时条件对应的阈值也无效

| 参数名 | 类型 | 说明 | 示例 |
|------|--------|--|------|
| kind | String | 必须, 目的地类型, 可能取值:MQTT, KAFKA, TSDB, BOS, 目前仅限TSDB | TSDB |

| 参数名 | 类型 | 说明 | 示例 |
|-------|--------|---|----|
| value | String | 必须, 对于 TSDB: value 是 目 的 地 TSDB 数据 库 的 访 问域 名 (也 即 end- point,e.g. test.tsdb.iot.g z.baidubce.com) 对于 MQTT: value是 目的 地 MQTT 主题 对于 KAFKA: value 是 目 的 地 KAFKA 主 题 对于 BOS: value 是 目 的 地 BOS 的 bucket, 如 bos:// mybucket 对于 MQTT_DYNAMIC: value 是 一 个 SQL SELECT 子 句, 用 于从 原始 消息 里 面选 择 字段 作 为 目的 地 主题, 如: dest.topic | |

DeviceRuleDestination

| 参数名 | 类型 | 说明 | 示例 |
|--------|--------|--|---------------|
| mode | String | 必 需, 存 储 tsdb 的 模 式, 可 选 取 值: metric, field;metric 表 示 多 metric 模 式, 每 个 属性 作 为 metric 存 入 tsdbfield 表 示 多 field 模 式, 每 个 属性 作 为 field 存 入 tsdb | metric |
| metric | String | 必 需, 自 定 义 met- ric 的 名 字, 仅 在 field 模 式 下 生 效 metric 模 式 下: metric 名字 为 对应 的 属性 名 | newMetricName |

| 参数名 | 类型 | 说明 | 示例 |
|------|-----------|---|----------------------------------|
| tags | Map<k, v> | 非必需，自定义tags，仅在field模式下生效，若不填，则默认为设备的deviceName-metric模式下：tags为desc(属性的displayName), deviceName, schemaName注：key的限制是字母、数字、下划线，1-100个字符；value的限制只有1-200个字符的限制 | { "tag1": "aaa", "tag2": "bbb" } |

DeviceRuleFormat

| 参数名 | 类型 | 说明 | 示例 |
|--------------|-----------------------------------|---|--|
| id | String | 规则对应的id | 63d92c1de2bd46e0b257c6df67b4a7e9 |
| deviceName | String | 对应的设备名称 | myDevicename |
| name | String | 规则名称 | 规则名称1 |
| sources | List<DeviceRuleSourceDetail> | 规则的具体约束条件 | |
| destinations | List<DeviceRuleDestinationDetail> | 处理后的消息写往的目的地数组(TSDB, KAFKA, 另一个MQTT主题)目前只支持TSDB | [{ "kind": "TSDB", "value": "test.tsdb.iot.gz.baidubce.com" }] |
| enable | Boolean | 条件默认开启 | true |
| createTime | Long | 创建时间 | 1494904250 |
| updateTime | Long | 更新时间 | 1494904250 |

DeviceRuleResponse

| 参数名 | 类型 | 说明 | 示例 |
|--------------|------------------|---|--|
| id | String | 规则对应的id | 63d92c1de2bd46e0b257c6df67b4a7e9 |
| deviceName | String | 对应的设备名称 | myDevicename |
| name | String | 规则名称 | 规则名称1 |
| sources | List | 规则的具体约束条件 | |
| destinations | List | 处理后的消息写往的目的地数组 (TSDB, KAFKA, 另一个MQTT主题) 目前只支持TSDB | [{"kind": "TSDB", "value": "test.tsdb.iot.gz.baidubce.com"}] |
| format | DeviceRuleFormat | 转存TSDB数据的格式定义 | {"mode": "metric"} |
| enable | Boolean | 条件默认开启 | true |
| createTime | Long | 创建时间 | 1494904250 |
| updateTime | Long | 更新时间 | 1494904250 |

DeviceFormatRuleResponse

| 参数名 | 类型 | 说明 | 示例 |
|--------------|--------|--|-------------|
| description | String | 条件的描述信息，最长255个字符 | 规则名称1 |
| name | String | 对应的模版的属性名称 | temperature |
| type | String | 属性数据类型，对应于 Schema 的数据类型，string, number, bool, object | string |
| displayName | String | 对应的Schema的属性显示名称 | 温度 |
| unit | String | 对应的Schema的数据单位 | Mpa |
| defaultValue | String | 对应的Schema的数据默认值 | 200 |

| 参数名 | 类型 | 说明 | 示例 |
|--------------|--------|----------------|------------|
| value | String | 条件对应的阀值 | 37 |
| lastSaveTime | Long | 最后一次存储时间，没有则为0 | 1494904250 |
| condition | String | 约束条件运算符 | >= |

DeviceRuleSourceDetail 约束条件运算符包括：

运算符 | 描述

----- | -----> | 大于

>= | 大于等于 < | 小于 <= | 小于等于 = | 等于 <> | 不等于

| 表示有数据即存储，此属性数据都会存到TSDB里，此时条件对应的阀值也无效

| 空，即表示没有约束条件，此项数据不作转发处理，此时条件对应的阀值也无效

| 参数名 | 类型 | 说明 | 示例 |
|------|--------|---|--|
| id | String | 规则目的地的uuid | 6653da99bf9a4e3 5ba4f997e000a 699f |
| kind | String | 目的地类型，可能取值:MQTT, KAFKA, TSDB, BOS, 目前仅限TSDB | TSDB |

| 参数名 | 类型 | 说明 | 示例 |
|-------|--------|--|----|
| value | String | 对于 TSDB: value 是目的 地 TSDB 数据库 的 访问 域名 (也即 endpoint, e.g. test.tsdb.iot.gz.baidubce.com 对于 MQTT: value 是目的地 MQTT 主题 对于 KAFKA: value 是目的地 KAFKA 主题 对于 BOS: value 是目的地 BOS 的 bucket, 如 bos:// mybucket 对于 MQTT_DYNAMIC: value 是一个 SQL SELECT 子句, 用 于从原始消息里 面选择字段作为 目的地主题, 如: dest.topic | |

DeviceRuleDestinationDetail

7.6 OTA升级服务

7.6.1 上传固件包文件

| 方法 | API | 说明 |
|------|--------------------------------------|---------|
| POST | /v3/iot/management/ota/firmware-file | 上传固件包文件 |

请求参数

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|------|--------|------|---------------------|
| name | String | 必选 | “file”，文件大小限制为10M以内 |

返回参数

一个[ResourceID](#)

[请求示例](#)

```
POST /v3/iot/management/ota/firmware-file HTTP/1.1

Host: iotdm.gz.baidubce.com

Authorization: {authorization}

Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryALn77wBBoEIJcgG

-----WebKitFormBoundaryALn77wBBoEIJcgG

Content-Disposition: form-data; name="file"; filename="xxx"

Content-Type: xxx/xxx

-----WebKitFormBoundaryALn77wBBoEIJcgG--
```

[返回示例](#)

```
HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054

{

  "id": "993ff7e9-018b-4246-a7ba-5ddqwerqrw",

}
```

7.6.2 创建固件包

| 方法 | API | 说明 |
|------|---------------------------------|-------|
| POST | /v3/iot/management/ota/firmware | 创建固件包 |

请求参数

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|-------------|--------|------|------------|
| schemaId | String | 必选 | schema id |
| description | String | 可选 | 描述 |
| version | String | 必选 | 版本号 |
| fileId | String | 必选 | 上传的固件包文件ID |

返回参数

一个[ResourceID](#)

请求示例

```
POST /v3/iot/management/ota/firmware HTTP/1.1
```

```
Host: iotdm.gz.baidubce.com
```

```
Authorization: {authorization}
```

```
Content-Type: application/json; charset=utf-8
```

```
{
```

```
    "schemaId": "agagegj-agaeg-gaeg-eg",
```

```
    "description": "device_description",
```

```
    "version": "1.0.0",
```

```
    "fileId": "993ff7e9-018b-4246-a7ba-5ddqwerqrw"
```

```
}
```

返回示例

```
HTTP/1.1 200 OK
```

```
Content-Type: application/json; charset=utf-8
```

```
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054

{
  "id": "993ff7e9-018b-4246-a7ba-5ddqwerqrw"
}
```

7.6.3 获取固件包详情

| 方法 | API | 说明 |
|-----|--|---------|
| GET | /v3/iot/management/ota/firmware/{firmwareId} | 获取固件包详情 |

请求参数

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|------------|--------|------|-------|
| firmwareId | String | 必须 | 固件包ID |

返回参数

一个[FirmwarePackageDetail](#)

请求示例

```
GET /v3/iot/management/ota/firmware/12345 HTTP/1.1
```

```
Host: iotdm.gz.baidubce.com
```

```
Authorization: {authorization}
```

```
Content-Type: application/json; charset=utf-8
```

返回示例

```
HTTP/1.1 200 OK
```

```
Content-Type: application/json; charset=utf-8
```

```
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5dddac970054

{
    "id": "12345",
    "description": "device_description",
    "version": "1.2.3",
    "fileId": "file_id",
    "fileName": "file_name",
    "downloadUrl": "https://download_url",
    "schemaName": "door",
    "schemaId": "agage-agage",
    "createdAt": "2018-05-09T03:48:59Z",
    "jobs": [
        {
            "id": "job_id",
            "name": "job_name"
        }
    ]
}
```

7.6.4 获取固件包列表

| 方法 | API | 说明 |
|-----|---|---------|
| GET | /v3/iot/manage-ment/ota/firmware? pageNo=xxx&pageSize=xxx&order=xxx&orderBy=xxx&schemaId=xxx | 查询固件包列表 |

请求参数

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|----------|--------|------|---------------------------|
| pageNo | int | 可选 | 默认是1，表示第几页 |
| pageSize | int | 可选 | 每页返回的数目， 默认10，限制为100以内 |
| order | String | 可选 | 默认是按照创建时间排序，支持“createdAt” |
| orderBy | String | 可选 | 默认是降序排列，支持“asc” “desc” |
| schemaId | String | 可选 | 按schemaId过滤 |

返回参数

| 参数名称 | 参数类型 | 说明 |
|-----------|------------------------|--------------|
| amount | int | 满足查询条件的固件包总数 |
| pageNo | int | 表示第几页 |
| pageSize | int | 每页返回的数目 |
| firmwares | List <FirmwarePackage> | 固件包列表 |

请求示例

```
GET /v3/iot/management/ota/firmware?pageNo=1&pageSize=10 HTTP/1.1
```

```
Host: iotdm.gz.baidubce.com
```

```
Authorization: {authorization}
```

```
Content-Type: application/json; charset=utf-8
```

返回示例

```
HTTP/1.1 200 OK
```

```
Content-Type: application/json; charset=utf-8  
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054  
  
{  
    "amount": 1,  
    "pageNo": 1,  
    "pageSize": 10,  
    "firmwares": [  
        {  
            "id": "12345",  
            "description": "device_description",  
            "version": "1.2.3",  
            "fileId": "file_id",  
            "fileName": "file_name",  
            "downloadUrl": "https://download_url",  
            "schemaName": "door",  
            "schemaId": "agage-agage",  
            "createdAt": "2018-05-09T03:48:59Z"  
        }  
    ]  
}
```

7.6.5 查询设备使用固件包版本

| 方法 | API | 说明 |
|------|--|------------|
| POST | /v3/iot/management/ota/device-firmware-version-query | 列举当前被设备使用的 |

请求参数

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|----------|--------|------|------|
| schemaId | String | 必选 | 模型ID |

返回参数

一个[VersionList](#)

请求示例

```
POST /v3/iot/management/ota/device-firmware-version-query HTTP/1.1
```

```
Host: iotdm.gz.baidubce.com
```

```
Authorization: {authorization}
```

```
Content-Type: application/json; charset=utf-8
```

```
{
```

```
"schemaId":"afgeagag-018b-4246-a7ba-5ddac970054"
```

```
}
```

返回示例

```
HTTP/1.1 200 OK
```

```
Content-Type: application/json; charset=utf-8
```

```
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054
```

```
{
```

```

    "versions": ["1.0.1", "1.0.2"]

}

```

7.6.6 创建升级任务

| 方法 | API | 说明 |
|------|----------------------------|--------|
| POST | /v3/iot/management/ota/job | 创建升级任务 |

请求参数

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|---------------|--------------|------|--|
| obName | String | 必选 | 升级任务名称 |
| firmwareId | String | 必选 | 固件包ID |
| description | String | 可选 | 升级任务描述 |
| devices | List<String> | 可选 | devices,versionFilter,fileId 三者选择一个，优先级依次递减 |
| versionFilter | List<String> | 可选 | 如果列表的size为0，则表明全量设备 |
| fileId | String | 可选 | 文件内容格式为每行一个设备名 |

返回参数

一个[ResourceID](#)

请求示例

```
POST /v3/iot/management/ota/job HTTP/1.1
```

```
Host: iotdm.gz.baidubce.com
```

```
Authorization: {authorization}
```

```
Content-Type: application/json; charset=utf-8
```

```
{
```

```
"jobName": "your_ota_job_name",  
"firmwareId": "your_ota_firmware_id",  
"description": "your_description",  
"devices": ["device_1", "device_2"]  
}
```

[返回示例](#)

HTTP/1.1 200 OK

```
Content-Type: application/json; charset=utf-8  
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054  
  
{  
    "id": "e3a197fe-e211-4499-8a10-8c15146550ae"  
}
```

7.6.7 上传升级设备文件

| 方法 | API | 说明 |
|------|------------------------------------|------------|
| POST | /v3/iot/management/ota/device-file | 上传需升级的设备文件 |

[请求参数](#)

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|------|--------|------|--------------------|
| name | String | 必选 | “file”，文件大小限制为1M以内 |

[返回参数](#)

一个[ResourceID](#)

[请求示例](#)

```
POST /v3/iot/management/ota/device-file HTTP/1.1

Host: iotdm.gz.baidubce.com

Authorization: {authorization}

Content-Type: multipart/form-data; boundary=----WebKitFormBoundarymgrwDgRP9sdjkN9p

-----WebKitFormBoundarymgrwDgRP9sdjkN9p

Content-Disposition: form-data; name="file"; filename="xxx.txt"

Content-Type: text/plain

-----WebKitFormBoundarymgrwDgRP9sdjkN9p--
```

[返回示例](#)

```
HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054

{

  "id": "993ff7e9-018b-4246-a7ba-5ddqwerqrw",

}
```

7.6.8 查询升级任务

| 方法 | API | 说明 |
|-----|------------------------------------|--------|
| GET | /v3/iot/management/ota/job/{jobId} | 查询升级任务 |

[请求参数](#)

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|-------|--------|------|--------|
| jobId | String | 必选 | 升级任务ID |

[返回参数](#)

一个OtaJob对象

[请求示例](#)

```
GET /v3/iot/management/ota/job/e3a197fe-e211-4499-8a10-8c15146550ae HTTP/1.1
```

```
Host: iotdm.gz.baidubce.com
```

```
Authorization: {authorization}
```

```
Content-Type: application/json; charset=utf-8
```

[返回示例](#)

```
HTTP/1.1 200 OK
```

```
Content-Type: application/json; charset=utf-8
```

```
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054
```

```
{
```

```
    "id": "e3a197fe-e211-4499-8a10-8c15146550ae",
```

```
    "jobName": "job_name",
```

```
    "description": "job_description",
```

```
    "firmwareId": "a4dfc99e-92d5-435c-a903-47fa8f042c2a",
```

```
    "createdAt": "2018-05-09T03:48:59Z"
```

```
}
```

7.6.9 查询升级任务列表

| 方法 | API | 说明 |
|-----|--|----------|
| GET | /v3/iot/management/ota/job? pageNo=xxx&pageSize=xxx&orderBy=xxx&order=xxx | 查询升级任务列表 |

请求参数

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|----------|--------|------|---------------------------|
| pageNo | int | 可选 | 默认是1，表示第几页 |
| pageSize | int | 可选 | 每页返回的数目， 默认10，限制为100以内 |
| order | String | 可选 | 默认是按照创建时间排序，支持“createdAt” |
| orderBy | String | 可选 | 默认是降序排列，支持“asc”或“desc” |

返回参数

| 参数名称 | 参数类型 | 说明 |
|----------|--------------|-------------|
| amount | int | 满足条件的升级任务总数 |
| pageNo | int | 表示第几页 |
| pageSize | int | 每页返回的数目 |
| jobs | List<Otajob> | 升级任务列表 |

请求示例

```
GET /v3/iot/management/ota/job?pageNo=1&pageSize=10 HTTP/1.1
```

```
Host: iotdm.gz.baidubce.com
```

```
Authorization: {authorization}
```

```
Content-Type: application/json; charset=utf-8
```

[返回示例](#)

```
HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054

{

    "amount": 1,

    "pageNo": 1,

    "pageSize":10,

    "jobs": [

        {

            "id": "e3a197fe-e211-4499-8a10-8c15146550ae",

            "jobName": "job_name",

            "description":"job_description",

            "firmwareId":"a4dfc99e-92d5-435c-a903-47fa8f042c2a",

            "createdAt": "2018-05-09T03:48:59Z"

        }

    ]

}
```

7.6.10 查询升级任务结果

| 方法 | API | 说明 |
|-----|---|----------|
| GET | /v3/iot/management/ota/job/{jobId}/statistics | 查询升级任务结果 |

请求参数

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|------|--------|------|--------|
| obId | String | 必选 | 升级任务ID |

返回参数

一个[JobStatistics](#)

请求示例

```
GET /v3/iot/management/ota/job/e3a197fe-e211-4499-8a10-8c15146550ae/statistics HTTP/
1.1

Host: iotdm.gz.baidubce.com

Authorization: {authorization}

Content-Type: application/json; charset=utf-8
```

返回示例

```
HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054

{

  "deviceCount": 3,

  "succeededCount": 1

}
```

7.6.11 查询升级任务各设备升级结果

| 方法 | API | 说明 |
|-----|--|-----------|
| GET | /v3/iot/management/ota/job/ {jobId}/ device-result? pageNo=xx&pageSize=xx | 查询各设备升级结果 |

请求参数

| 参数名称 | 参数类型 | 是否必须 | 说明 |
|----------|--------|------|-------------------------------|
| obId | String | 必选 | 升级任务ID |
| pageNo | int | 可选 | 默认是1，表示第几页 |
| pageSize | int | 可选 | 每页返回的数目， 默认10，限制为100 以内 |

返回参数

| 参数名称 | 参数类型 | 说明 |
|----------|---|--------------|
| amount | int | 升级任务中包含的设备总数 |
| pageNo | int | 表示第几页 |
| pageSize | int | 每页返回的数目 |
| results | List< DeviceJobResult > | 设备升级结果列表 |

请求示例

```
GET /v3/iot/management/ota/job/e3a197fe-e211-4499-8a10-8c15146550ae/device-result?  
pageNo=1&pageSize=10 HTTP/1.1
```

Host: iotdm.gz.baidubce.com

Authorization: {authorization}

Content-Type: application/json; charset=utf-8

返回示例

HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054

```
{  
    "amount": 1,  
    "pageNo": 1,  
    "pageSize": 10,  
    "results": [  
        {  
            "deviceName": "e3a197fe-e211-4499-8a10-8c15146550ae",  
            "isOnline": "job_name",  
            "startAt": "2018-05-09T03:48:59Z",  
            "finishedAt": "2018-05-09T04:48:59Z",  
            "firmwareVersionBefore": "1.0.0",  
            "jobStatus": "SUCCEEDED"  
        }  
    ]  
}
```

7.6.12 参数定义

| 参数名称 | 参数类型 | 说明 |
|---------|--------|--------|
| id | String | 固件包ID |
| version | String | 固件包版本号 |

| 参数名称 | 参数类型 | 说明 |
|-------------|--------|--|
| description | String | 固件包描述 |
| fileId | String | 固件包文件ID |
| fileName | String | 固件包文件名 |
| downloadUrl | String | 下载路径 |
| schemaName | String | 固件包对应模型名称 |
| schemaId | String | 固件包对应模型ID |
| createdAt | Date | 固 件 包 创 建 时 间，格 式 yyyy-MM- dd' T' HH:mm:ss' Z' |

FirmwarePackage

| 参数名称 | 参数类型 | 说明 |
|-------|---------------------------------------|---------------|
| 类中各字段 | FirmwarePackage | 继承其中所有字段 |
| jobs | List< SimplifiedJob > | 使用该固件包的升级任务列表 |

FirmwarePackageDetail

| 参数名称 | 参数类型 | 说明 |
|------|--------|--------|
| id | String | 升级任务ID |
| name | String | 升级任务名称 |

SimplifiedJob

| 参数名称 | 参数类型 | 说明 |
|-------------|--------|-----------|
| id | String | 任务ID |
| jobName | String | 任务名称 |
| description | String | 任务描述 |
| firmwareId | String | 任务使用固件包ID |

| 参数名称 | 参数类型 | 说明 |
|-----------|------|---|
| createdAt | Date | 任务创建时间, 格式 yyyy-MM- dd' T' HH:mm:ss' Z' |

OtaJob

| 参数名称 | 参数类型 | 说明 |
|----------------|------|-------------|
| deviceCount | int | 升级任务涉及的设备数量 |
| succeededCount | int | 成功的设备数量 |

JobStatistics

| 参数名称 | 参数类型 | 说明 |
|------------------------|---------|--|
| deviceName | String | 物影子名字 |
| isOnline | boolean | 在线状态 |
| startedAt | Date | 升级开始时间, 格式 yyyy-MM- dd' T' HH:mm:ss' Z' |
| finishedAt | Date | 升级结束时间, 格式 yyyy-MM- dd' T' HH:mm:ss' Z' |
| firmwareVersion Before | String | 升级前版本号 |
| jobStatus | String | 升级结果, 内容为 “PENDING” “RUNNING” “SUCCEEDED” “FAILED”之一 |

DeviceJobResult

| 参数名称 | 参数类型 | 说明 |
|------|--------|------|
| id | String | 资源ID |

ResourceID

| 参数名称 | 参数类型 | 说明 |
|----------|--------------|------|
| versions | List<String> | 版本列表 |

[VersionList](#)

7.7 权限管理

7.7.1 简介

在新版物管理中，我们引入了权限组的概念，权限组拥有对其中包含的所有设备的topic的访问权限。

我们定义了两种权限组的概念，普通权限组和超级权限组。其中，普通权限组即为正常的权限组，可以向权限组内增添删减设备。

而超级权限组则包含对该用户下所有设备的topic访问权限。因而超级权限组不支持组内设备的查看及更新操作。

7.7.2 设备权限组管理

| 方法 | API | 说明 |
|------|---------------------------|-------|
| POST | /v3/iot/management/domain | 创建权限组 |

创建权限组 请求参数

| 参数名 | 参数类型 | 是否必须 | 说明 |
|-------------|--------|------|-----------------------|
| name | String | 必须 | 权限组名称 |
| description | String | 必须 | 权限组的相关说明 |
| type | String | 必须 | 权限组的类型，支持ROOT, NORMAL |

返回参数

一个AccessDetailResponse实例

请求示例

```
POST /v3/iot/management/domain HTTP/1.1  
  
Host:iotdm.gz.baidubce.com  
  
Authorization:{authorization}  
  
Content-Type: application/json; charset=utf-8  
  
{  
  
"name": "myNormalDomain",  
  
"description": "description",  
  
"type": "NORMAL"  
  
}
```

返回示例

```
HTTP/1.1 200 OK  
  
Content-Type: application/json; charset=utf-8  
  
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054  
  
{  
  
"tcpEndpoint": "tcp://test.baidu.iot.com",  
  
"sslEndpoint": "ssl://test.baidu.iot.com",  
  
"wssEndpoint": "wss://test.baidu.iot.com",  
  
"username": "endpointName/mySuperDomain",  
  
"key": "bWwCxGwaw3boV48NqsuG+XVaHpxfKdMPvmdJzNObvbY="  
  
}
```

| 方法 | API | 说明 |
|--------|--|-------------------|
| DELETE | /v3/iot/management/domain/{domainName} | 根据domainName删除权限组 |

[删除权限组 请求示例](#)

```
DELETE /v3/iot/management/domain/{domainName} HTTP/1.1
```

```
Host:iotdm.gz.baidubce.com
```

```
Authorization:{authorization}
```

[返回示例](#)

```
HTTP/1.1 200 OK
```

```
Content-Type: application/json; charset=utf-8
```

```
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054
```

| 方法 | API | 说明 |
|-----|--|-------------|
| GET | GET / v3/ iot/ management/ domain? pageNo=xx&pageSize=xx&order=xx&orderBy=xx&key=xx&type=xx&deviceName=xx | 根据条件查询权限组列表 |

[获取权限组列表 请求参数](#)

| 参数名 | 参数类型 | 是否必须 | 说明 |
|----------|--------|------|--------------------------------|
| pageNo | int | 可选 | 获取列表在查询结果的页码， 默认为1 |
| pageSize | int | 可选 | 一页所包含的最大数量， 默认为10 |
| order | String | 可选 | 查询结果升序或降序排列， asc, desc， 默认desc |

| 参数名 | 参数类型 | 是否必须 | 说明 |
|------------|--------|------|---|
| orderBy | String | 可选 | 排序的索引列，name, createTime, lastUpdatedTime, 默认 createTime |
| key | String | 可选 | 查询关键字，权限组名称 |
| type | String | 可选 | 权限组类型，ROOT, NORMAL, ALL, 默认是ALL |
| deviceName | String | 可选 | 设备名称，表示查询包含该设备的权限组 |

返回参数

| 参数名 | 参数类型 | 说明 |
|----------|--------------|------------|
| amount | int | 权限组总数 |
| pageNo | int | 当前页码 |
| pageSize | int | 一页所包含的最大数量 |
| domains | List<Domain> | 权限组列表 |

请求示例

```
GET /v3/iot/management/domain?pageNo=1&pageSize=10&order=desc&orderBy=createTime&key=myDomain&type=1.1
```

Host:iotdm.gz.baidubce.com

Authorization:{authorization}

返回示例

HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054

```
{
  "amount": 1,
  "pageNo": 1,
  "pageSize": 10,
  "domains": [
    {
      "name": "myDomain",
      "description": " description",
      "type": "NORMAL",
      "createTime": 1494904250000,
      "lastUpdatedTime": 1494904250000,
      "deviceNum": 0
    }
  ]
}
```

| 方法 | API | 说明 |
|-----|--|-------------|
| GET | /v3/iot/management/domain/{domainName} | 根据名称获取权限组详情 |

获取权限组详情 返回参数

一个DomainDetail实例

请求示例

```
GET /v3/iot/management/domain/{domainName} HTTP/1.1
Host:iotdm.gz.baidubce.com
```

```
Authorization:{authorization}
```

返回示例

```
HTTP/1.1 200 OK
```

```
Content-Type: application/json; charset=utf-8
```

```
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054
```

```
{
  "name": "myDomain",
```

```

"description": "description",
"type": "NORMAL",
"createTime": 1494904250000,
"lastUpdatedTime": 1494904250000,
"deviceNum": 2,
"devices": [
    "device-1",
    "device-2"
]
}

```

| 方法 | API | 说明 |
|-----|---|----------|
| PUT | /v3/iot/management/domain/{domainName}?modify | 权限组中更改设备 |

权限组中更改设备 请求参数

| 参数名 | 参数类型 | 是否必须 | 说明 |
|----------------|--------------|------|-------------|
| addedDevices | List<String> | 可选 | 需要添加的设备名称列表 |
| removedDevices | List<String> | 可选 | 需要移除的设备名称列表 |

返回参数

| 参数名 | 参数类型 | 是否必须 | 说明 |
|----------------|--------------|------|-------------|
| addedDevices | List<String> | 可选 | 成功添加的设备名称列表 |
| removedDevices | List<String> | 可选 | 成功移除的设备名称列表 |

请求示例

```

PUT /v3/iot/management/domain/1234939554?modify HTTP/1.1
Host:iotdm.gz.baidubce.com

```

```
Authorization:{authorization}
```

```
{
```

```

"addedDevices": [
    "device-1",
    "device-2"
],
"removedDevices": [
    "device-3",
    "device-4"
]
}

```

[返回示例](#)

HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054

```

{
    "addedDevices": [
        "device-1",
        "device-2"
    ],
    "removedDevices": [
        "device-3",
        "device-4"
    ]
}

```

| 方法 | API | 说明 |
|-----|--|--------|
| PUT | /v3/iot/management/domain/{domainName} | 更新注册信息 |

[更新权限组注册信息 请求参数](#)

| 参数名 | 参数类型 | 是否必须 | 说明 |
|-------------|--------|------|-------------|
| description | String | 必选 | 需要更新的设备描述信息 |

[请求示例](#)

```
PUT /v3/iot/management/domain/1234939554 HTTP/1.1  
Host:iotdm.gz.baidubce.com
```

```
Authorization:{authorization}
```

```
{
```

```
"description":"new description"
```

```
}
```

返回示例

```
HTTP/1.1 200 OK
```

```
Content-Type: application/json; charset=utf-8
```

```
x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054
```

| 方法 | API | 说明 |
|-----|---|------------|
| GET | / v3/ iot/ management/ domain/ {domainName}/ accessDetail | 根据名称获取连接详情 |

获取权限组接入信息 返回参数

一个AccessDetailResponse实例。

请求示例

```
GET /v3/iot/management/domain/{domainName}/accessDetail HTTP/1.1  
Host:iotdm.gz.baidubce.com
```

```
Authorization:{authorization}
```

返回示例

```
HTTP/1.1 200 OK
```

```

Content-Type: application/json; charset=utf-8

x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054

{

  "tcpEndpoint": "tcp://test.baidu.iot.com",

  "sslEndpoint": "ssl://test.baidu.iot.com",

  "wssEndpoint": "wss://test.baidu.iot.com",

  "username": "endpointName/device_1",

  "key": "xxxxxxxxxx"

}

```

| 方法 | API | 说明 |
|-----|--|------|
| PUT | /v3/iot/management/domain/{domainName}?updateSecretKey | 更改密钥 |

更新权限组密钥 返回参数

一个AccessDetailResponse实例。

请求示例

```

PUT /v3/iot/management/domain/{domainName}?updateSecretKey HTTP/1.1

Host: iotdm.gz.baidubce.com

Authorization: {authorization}

Content-Type: application/json; charset=utf-8

```

返回示例

HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054

```
{
  "tcpEndpoint": "tcp://test.baidu.iot.com",
  "sslEndpoint": "ssl://test.baidu.iot.com",
  "wssEndpoint": "wss://test.baidu.iot.com",
  "username": "endpointName/domainName",
  "key": "bWwCxGwaw3boV48NqsuG+XVaHpxfKdMPvmdJzN0vbY="

}
```

| 方法 | API | 说明 |
|-----|--|--------------------------------------|
| GET | GET / v3/ iot/ management/ domain/ {domainName}/ devices? pageNo=xx&pageSize=xx&order=xx&orderBy=xx&name=xx&value=xx&favourite=xx | 根据条件查询设备列表，返回设备名字、是否存在于权限组内、设备所在普通权限 |

获取及查询设备列表 请求参数

| 参数名 | 参数类型 | 是否必须 | 说明 |
|----------|--------|------|--|
| pageNo | int | 可选 | 获取列表在查询结果的页码， 默认为1 |
| pageSize | int | 可选 | 一页所包含的最大数量， 默认为10 |
| order | String | 可选 | 查询结果升序或降序排列， asc,desc， 默认desc |
| orderBy | String | 可选 | 排序的索引列， name, createTime, lastUpdatedTime， 默认 createTime |
| name | String | 可选 | 查询属性名 |
| value | String | 可选 | 查询属性值 |

| 参数名 | 参数类型 | 是否必须 | 说明 |
|-----------|--------|------|-----------------------------|
| favourite | String | 可选 | 收藏, true, false, all, 默认all |

[返回参数](#)

| 参数名 | 参数类型 | 说明 |
|----------|----------------------|------------|
| pageNo | int | 当前页码 |
| pageSize | int | 一页所包含的最大数量 |
| amount | int | 设备总数 |
| devices | List<DeviceInDomain> | 设备列表 |

[请求示例](#)

```
GET /v3/iot/management/domain/{domainName}/devices?pageNo=1&pageSize=10&orderBy=createTime&order=d
1.1
```

Host:iotdm.gz.baidubce.com

Authorization:{authorization}

[返回示例](#)

HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

x-bce-request-id: 993ff7e9-018b-4246-a7ba-5ddac970054

```
{
    "amount": 2,
    "pageNo": 1,
    "pageSize": 10,
    "devices": [
        {
            "deviceName": "device_1",
            "existed": true,
            "domainNum": 1
        },
    ],
}
```

```

    {
        "deviceName": "device_2",
        "existed": false,
        "domainNum": 2
    }
]
}

```

7.7.3 参数定义

[Domain参数列表](#)

| 参数名 | 参数类型 | 说明 |
|-----------------|--------|-----------------------------|
| name | String | 权限组名称 |
| description | String | 权限组说明 |
| type | String | 权限组的类型，支持ROOT, NORMAL，二者选其一 |
| createTime | long | 权限组创建时间 |
| lastUpdatedTime | long | 权限组更新时间 |
| deviceNum | int | 权限组包含设备数目 |

[DomainDetail参数列表](#)

| 参数名 | 参数类型 | 说明 |
|-----------|--------------|---------|
| Domain对应项 | Domain | 权限组基本信息 |
| devices | List<String> | 设备列表 |

[AccessDetailResponse参数列表](#)

| 参数名 | 参数类型 | 说明 |
|-------------|--------|-------------------------|
| tcpEndpoint | String | tcp协议的物接入endpoint |
| sslEndpoint | String | 支持ssl的物接入endpoint |
| wssEndpoint | String | 支持wss的物接入endpoint |
| username | String | endpointName/ thingName |
| key | String | 物接入Thing密钥 |

DeviceInDomain参数列表

| 参数名 | 参数类型 | 说明 |
|------------|---------|-----------------|
| deviceName | String | 设备名称 |
| existed | boolean | 是否存在于该权限组内 |
| domainNum | int | 设备被包含在普通权限组内的数目 |

7.8 参数定义

7.8.1 DeviceListRequest参数列表

| 参数名称 | 参数类型 | 说明 |
|---------|--------------|---------------|
| devices | List<String> | 需要进行操作的设备名称列表 |

7.8.2 DeviceListResponse参数列表

| 参数名称 | 参数类型 | 说明 |
|---------|--------------|---------------|
| devices | List<String> | 需要进行操作的设备名称列表 |

7.8.3 DeviceProfile参数列表

| 参数名称 | 参数类型 | 说明 |
|---------------------|------------------|------------------------------------|
| DeviceBasicInfo中对应项 | DeviceBasicInfo | 继承DeviceBasicInfo，描述注册基本信息以及模型使用信息 |
| attributes | JsonNode | json对象，存储设备自定义的相关属性 |
| device | DeviceAttributes | DeviceAttributes 对象，描述设备属性相关信息 |

7.8.4 DeviceProfileResponse参数列表

| 参数名称 | 参数类型 | 说明 |
|---------------------|------------------|------------------------------------|
| DeviceBasicInfo中对应项 | DeviceBasicInfo | 继承DeviceBasicInfo，描述注册基本信息以及模型使用信息 |
| attributes | JsonNode | json对象，存储设备自定义的相关属性 |
| device | DeviceAttributes | DeviceAttributes 对象，描述设备属性相关信息 |

7.8.5 DeviceBasicInfo参数列表

| 参数名称 | 参数类型 | 说明 |
|----------------|---------|---------------------|
| name | String | 设备名称 |
| description | String | 设备描述 |
| createTime | long | 设备创建时间(以毫秒为单位) |
| state | String | 设备状态 |
| lastActiveTime | long | 设备最后一次的活跃时间(以毫秒为单位) |
| schemaId | String | 设备采用模型Id |
| schemaName | String | 设备采用模型名称 |
| favourite | boolean | 是否收藏， 默认false |

7.8.6 DeviceAttributes参数列表

| 参数名称 | 参数类型 | 说明 |
|-----------------|----------|--|
| reported | JsonNode | json对象，存储设备属性实际值 |
| desired | JsonNode | json对象，对出设备属性的期望值 |
| profileVersion | Integer | 设备属性版本号 |
| lastUpdatedTime | JsonNode | 与 reported 和 desired 属性一一对应，相应的属性值为更新的时间戳，以long值表示，单位为毫秒 |

7.8.7 DeviceViewResponse参数列表

| 参数名称 | 参数类型 | 说明 |
|---------------------|----------------------|--|
| DeviceBasicInfo中对应项 | DeviceBasicInfo | 继承DeviceBasicInfo，描述注册基本信息以及模型使用信息 |
| view | DeviceViewAttributes | DeviceViewAttributes 对象，描述按照模板合并后对应的属性信息 |

7.8.8 DeviceViewAttributes参数列表

| 参数名称 | 参数类型 | 说明 |
|----------------|----------|---------------------------|
| profileVersion | Integer | 设备属性版本号 |
| properties | JsonNode | json数组，每个数组元素都是一个模板中对应的属性 |

7.8.9 DeviceAccessDetailResponse参数列表

| 参数名称 | 参数类型 | 说明 |
|-------------|--------|-------------------------|
| tcpEndpoint | String | tcp协议的物接入endpoint |
| sslEndpoint | String | 支持ssl的物接入endpoint |
| username | String | endpointName/ thingName |
| key | String | 物接入Thing密钥 |

| **参数名称** | **参数类型** | **说明** |
|--------------|----------|--------|
| name | String | 属性名称 |
| displayName | String | 属性显示名称 |
| unit | String | 数据单位 |
| defaultValue | String | 数据默认值 |

| **参数名称** | **参数类型** | **说明** |
|----------|----------|--|
| type | String | 属性数据类型, string number bool object object类型说明: object类型的数据格式与Json object相同, 其定义为: 一个无序的“‘名称/值’对”集合。一个对象以“{”(左括号)开始, “}”(右括号)结束。每个“名称”后跟一个“:”(冒号); “‘名称/值’对”之间使用“,”(逗号)分隔。格式举例如下: |

SchemaProperty参数列表 正确的格式

```
{
    "key1": "value1",
    "key2": 123
}

{
    "key1": "value1",
    "key2": [1, 3, 4],      # value 可以是Array
    "key3" : {
        "key4": "value4"    # 支持嵌套
    }
}
```

错误的格式

```
[1, 3, 5]          # 直接是一个 Array, Object 类型必须在 {} 里面
"hello world"      # 直接是一个 字符串
12.34              # 直接是一个 数字
```

| **参数名称** | **参数类型** | **说明** |
|----------|----------|--------|
| id | String | 模板ID |
| name | String | 模板名称 |

| **参数名称** | **参数类型** | **说明** |
|-----------------|----------------------|-----------------|
| description | String | 模板说明 |
| createTime | long | 模板创建时间 (以毫秒为单位) |
| lastUpdatedTime | long | 模板更新时间 (以毫秒为单位) |
| properties | List<SchemaProperty> | 模板属性列表 |

Schema参数列表

7.9 更新历史

2018-05-17

物管理数据写入TSDB增加带TSDB格式的转存规则。

2018-05-15

增加OTA升级服务API。

2017-12-01

在原有的3种数据类型string, number, bool的基础上，增加object类型。

2017-11-2

增加物管理接入规则引擎。

第8章 Java SDK文档(数据型)

8.1 目录

- 概述
- 安装SDK工具包
- 快速入门
- 创建IotHubClient
- endpoint操作
 - 创建endpoint
 - 获取endpoint列表
 - 查看指定的endpoint信息
 - 删除endpoint
- thing操作
 - 创建thing
 - 查看thing
 - 删除thing
- principal操作
 - 创建principal
 - 查看principal
 - 绑定指定的thing和principal
 - 重新获得principal的证书和密码
 - 删除principal
- policy操作
 - 创建policy
 - 查看policy
 - 绑定指定的principal和policy
 - 删除policy
- permission操作
 - 创建permission
 - 更新permission
 - 查看permission
- Certificate操作
 - 创建Cert

- 重置Cert
- 版本说明
 - v0.10.13

8.2 概述

本文档主要介绍IoT Hub Java SDK的安装和使用。在使用本文档前，您需要先了解IoT Hub的一些基本知识，并已经开通了Iot Hub服务。若您还不了解Iot Hub，可以参考[产品描述](#)和[操作指南](#)。

8.3 安装SDK工具包

运行环境

Java SDK工具包可在jdk1.6、jdk1.7、jdk1.8环境下运行。

方式一：使用Maven安装

在Maven的pom.xml文件中添加bce-java-sdk的依赖：

```
<dependency>
    <groupId>com.baidubce</groupId>
    <artifactId>bce-java-sdk</artifactId>
    <version>{version}</version>
</dependency>
```

其中，`{version}`为版本号，可以在[SDK下载页面](#)找到。

方式二：直接使用JAR包安装

1. 在[官方网站](#)下载Java SDK压缩工具包。
2. 将下载的**bce-java-sdk-version.zip**解压后，复制到工程文件夹中。
3. 在Eclipse右键“工程->Properties->Java Build Path->Add JARs”。
4. 添加SDK工具包**lib/bce-java-sdk-version.jar**和第三方依赖工具包**third-party/*.jar**。
其中，`version`为版本号。

SDK目录结构

```

com.baidubce
    └── auth                                //BCE签名相关类
    └── http                                 //BCE的Http通信相关类
    └── internal                            //SDK内部类
    └── model                               //BCE公用model类
    └── services
        └── iothub                         //IoT服务相关类
            └── model                      //IoT内部model, 如Request或
Response
    └── IoTHubClient.class                  //IoT客户端入口类
    └── util                                //BCE公用工具类
    └── BceClientConfiguration.class        //对BCE的HttpClient的配置
    └── BceClientException.class           //BCE客户端的异常类
    └── BceServiceException.class          //与BCE服务端交互后的异常类
    └── ErrorCode.class                   //BCE通用的错误码
    └── Region.class                      //BCE提供服务的区域

```

8.4 快速入门

通过Java SDK创建一个物接入应用的流程如下：

1. [创建IotHubClient](#)
2. [创建endpoint](#)
3. [创建thing](#)
4. [创建principal](#)
5. [绑定指定的thing和principal](#)
6. [创建policy](#)
7. [绑定指定的principal和policy](#)
8. [创建permission](#)

8.5 创建IotHubClient

用户可以参考如下代码新建一个IotHubClient：

```

// 初始化相关参数
String AK = "ACCESS_KEY_ID";
String SK = "SECRET_ACCESS_KEY";
String ENDPOINT = "iot.gz.baidubce.com";

```

```

//使用物接入相关参数
String TEST_ENDPOINT_NAME = "sdk_test_endpoint_01";
String TEST_THING_NAME = "sdk_test_thing_01";
String TEST_PRINCIPAL_NAME = "sdk_test_principal_01";
String TEST_POLICY_NAME = "sdk_test_policy_01";
String TEST_TOPIC = "abc";

//创建物接入client
BceClientConfiguration config = new BceClientConfiguration()
    .withCredentials(new DefaultBceCredentials(AK, SK))
    .withEndpoint(ENDPOINT);
IoTHubClient iotHubClient = new IoTHubClient(config);

```

在代码中，变量ACCESS\KEY\ID与SECRET_ACCESS_KEY是系统分配给用户的，均为字符串，用于标识用户，为访问物管理做签名认证。其中ACCESS\KEY\ID对应控制台中的“Access Key ID”，SECRET\ACCESS\KEY对应控制台的“Access Key Secret”，获取方式请参考[获取AK/SK](#)。

参数说明

BceClientConfiguration中有更多的配置项，可配置如下参数：

| 参数 | 说明 |
|--------------------------------------|--------------------------|
| connectionTimeoutInMillis | 建立连接的超时时间（单位：毫秒） |
| localAddress | 本地地址 |
| maxConnections | 允许打开的最大HTTP连接数 |
| proxyDomain | 访问NTLM验证的代理服务器的Windows域名 |
| proxyHost | 代理服务器主机地址 |
| proxyPassword | 代理服务器验证的密码 |
| proxyPort | 代理服务器端口 |
| proxyPreemptiveAuthenticationEnabled | 是否设置用户代理认证 |
| proxyUsername | 代理服务器验证的用户名 |
| proxyWorkstation | NTLM代理服务器的Windows工作站名称 |
| retryPolicy | 连接重试策略 |
| socketBufferSizeInBytes | Socket缓冲区大小 |
| socketTimeoutInMillis | 通过打开的连接传输数据的超时时间（单位：毫秒） |
| userAgent | 用户代理，指HTTP的User-Agent头 |

8.6 endpoint操作

8.6.1 创建endpoint

创建endpoint之前应先[创建IotHubClient](#)。

用户可以参考如下代码新建一个endpoint：

```
//用一个字符串创建endpoint  
QueryEndpointResponse responseQuery = iotHubClient.createEndpoint(TEST_ENDPOINT_NAME);
```

8.6.2 获取endpoint列表

用户可以参考如下代码获取endpoint列表：

```
//列出该用户下所有endpoint相关信息（ps. 所有的list操作均可加分页参数）  
ListResponse responseList = iotHubClient.listEndpoints();
```

8.6.3 查看指定的endpoint信息

用户可以参考如下代码查看endpoint：

```
//列出该用户下指定名字的endpoint相关信息  
QueryEndpointResponse responseQuery = iotHubClient.queryEndpoint(TEST_ENDPOINT_NAME);
```

8.6.4 删除endpoint

用户可以参考如下代码删除endpoint：

```
//删除endpoint  
BaseResponse responseBase = iotHubClient.deleteEndpoint(TEST_ENDPOINT_NAME);
```

8.7 thing操作

8.7.1 创建thing

创建thing之前应先[创建endpoint](#)。

请参考以下代码创建thing：

```
//在指定的endpoint下面创建thing  
QueryThingResponse responseQuery = iotHubClient.createThing(TEST_ENDPOINT_NAME, TEST_THING_NAME);
```

8.7.2 查看thing

请参考以下代码查看thing:

```
//列出指定endpoint下所有的thing  
ListResponse responseList = iotHubClient.listThings(TEST_ENDPOINT_NAME);  
  
//列出指定的某个endpoint下指定的thing的相关信息  
QueryThingResponse responseQuery = iotHubClient.queryThing(TEST_ENDPOINT_NAME, TEST_THING_NAME);
```

8.7.3 删除thing

请参考以下代码删除thing:

```
//删除指定的thing  
BaseResponse responseBase = iotHubClient.deleteThing(TEST_ENDPOINT_NAME, TEST_THING_NAME);
```

8.8 principal操作

8.8.1 创建principal

创建principal之前应先[创建endpoint](#)。

请参考以下代码创建principal:

```
//创建principal ( response里有证书、密码 )  
CreatePrincipalResponse responseCreate = iotHubClient.createPrincipal(TEST_ENDPOINT_NAME, TEST_PRIN
```

8.8.2 查看principal

请参考以下代码查看principal:

```
//列出所有的principal  
ListResponse responseList = iotHubClient.listPrincipals(TEST_ENDPOINT_NAME);
```

查看指定thing下的principal。如果thing下没有绑定principal，以下操作返回的结果为空。

```
//列出指定thing下面所有的principal  
ListResponse responseList = iotHubClient.listPrincipals(TEST_ENDPOINT_NAME, TEST_THING_NAME);
```

8.8.3 绑定指定的thing和principal

请参考以下代码绑定指定的thing和principal：

```
//绑定指定的thing和principal  
iotHubClient.attachThingToPrincipal(TEST_ENDPOINT_NAME, TEST_THING_NAME, TEST_PRINCIPAL_NAME);  
  
//解除thing和principal的绑定关系  
BaseResponse response = iotHubClient.removeThingToPrincipal(TEST_ENDPOINT_NAME, TEST_THING_NAME, TE
```

8.8.4 重新获得principal的密码

请参考以下代码重新获得密码：

```
response = iotHubClient.regeneratePassword(TEST_ENDPOINT_NAME, TEST_PRINCIPAL_NAME);
```

8.8.5 删除principal

请参考以下代码绑定删除principal：

```
//删除principal  
BaseResponse responseBase = iotHubClient.deletePrincipal(TEST_ENDPOINT_NAME, TEST_PRINCIPAL_NAME);
```

8.9 policy操作

8.9.1 创建policy

请参考以下代码创建policy：

```
//创建policy  
QueryPolicyResponse responseQuery = iotHubClient.createPolicy(TEST_ENDPOINT_NAME, TEST_POLICY_NAME);
```

8.9.2 查看policy

请参考以下代码查看policy：

```
//列出指定endpoint下所有policy  
ListResponse responseList = iotHubClient.listPolicy(TEST_ENDPOINT_NAME);  
  
//列出指定principal下的所有policy  
responseList = iotHubClient.listPolicy(TEST_ENDPOINT_NAME, TEST_PRINCIPAL_NAME);  
  
//获取指定的policy信息  
QueryPolicyResponse responseQuery = iotHubClient.queryPolicy(TEST_ENDPOINT_NAME, TEST_POLICY_NAME);
```

8.9.3 绑定指定的principal和policy

请参考以下代码绑定指定的principal和policy:

```
//绑定指定的principal和policy  
iotHubClient.attachPrincipalToPolicy(TEST_ENDPOINT_NAME, TEST_PRINCIPAL_NAME, TEST_POLICY_NAME);  
  
//解除指定的principal和policy的绑定关系  
BaseResponse response = iotHubClient.removePrincipalToPolicy(TEST_ENDPOINT_NAME, TEST_PRINCIPAL_NAME);
```

8.9.4 删除policy

请参考以下代码删除指定的policy:

```
//删除指定的policy  
BaseResponse responseBase = iotHubClient.deletePolicy(TEST_ENDPOINT_NAME, TEST_POLICY_NAME);
```

8.10 permission操作

8.10.1 创建permission

创建permission之前，必须先完成以下操作：

1. [创建endpoint](#)
2. [创建thing](#)
3. [创建principal](#)
4. [绑定指定的thing和principal](#)
5. [创建policy](#)
6. [绑定指定的principal和policy](#)

请参考以下代码创建permission:

```
//准备operation参数，可以添加"PUBLISH"或"SUBSCRIBE"，也可以都加
List<Operation> operations = new ArrayList<Operation>();
operations.add(Operation.PUBLISH);
operations.add(Operation.SUBSCRIBE);

//创建permission参数包括endpoint、policy、操作类型、topic, response里有permissionUuid
QueryPermissionResponse response = iotHubClient.createPermission(TEST_ENDPOINT_NAME,
    TEST_POLICY_NAME,
    operations,
    TEST_TOPIC);

String permissionUuid = response.getPermissionUuid();
```

8.10.2 更新permission

请参考以下代码更新permission：

```
//更新permission，不需要更新的参数填null，
QueryPermissionResponse response = iotHubClient.updatePermission(TEST_ENDPOINT_NAME, permissionUuid);
```

8.10.3 查看permission

请参考以下代码查看permission：

```
//列出指定policy下所有permission
ListPermissionResponse responseList = iotHubClient.listPermission(TEST_ENDPOINT_NAME, TEST_POLICY_NAME);

//获取指定permission信息
QueryPermissionResponse responseQuery = iotHubClient.queryPermission(TEST_ENDPOINT_NAME, permissionUuid);
```

8.11 Certificate操作

8.11.1 1. 创建Certificate

```
//创建证书认证的principal ( response里有证书、密钥。密钥丢失需要重置，请妥善保管 )
CreatePrincipalWithCertResponse response = iotHubClient.createPrincipalWithCert(TEST_ENDPOINT_NAME,
    TEST_PRINCIPAL_NAME);
```

8.11.2 2.重置Certificate

```
//重置证书，可重新获得证书、密钥 RenewCertificateResponse response = iotHubClient.renewCertificate(TEST_EN  
TEST_PRINCIPAL_NAME);
```

8.11.3 3.使用MqttSDK实现双向认证示例代码

```
package com.packt.cookbook;  
  
import java.io.ByteArrayInputStream;  
import java.io.StringReader;  
import java.security.KeyStore;  
import java.security.PrivateKey;  
import java.security.cert.Certificate;  
import java.security.cert.CertificateFactory;  
import javax.net.ssl.KeyManager;  
import javax.net.ssl.KeyManagerFactory;  
import javax.net.ssl.SSLContext;  
import javax.net.ssl.TrustManager;  
import javax.net.ssl.TrustManagerFactory;  
import org.eclipse.paho.client.mqttv3.MqttClient;  
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;  
import org.eclipse.paho.client.mqttv3.MqttMessage;  
import org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;  
import java.security.KeyFactory;  
import java.security.spec.PKCS8EncodedKeySpec;  
import sun.misc.BASE64Decoder;  
import sun.security.provider.X509Factory;  
  
public class App  
{  
  
    public static void main( String[] args )  
    {  
        String endpoint = "your-endpoint:1884";      //输入创建endpoint返回的SSL地址  
        String username = "your-endpoint/god"; //输入创建thing返回的username  
        String topic = "asdfasd"; //订阅的消息主题  
        String privateKeyStr = "-----BEGIN RSA PRIVATE KEY----.your key.....  
END RSA PRIVATE KEY----";  
        String clientCertStr = "-----BEGIN CERTIFICATE----.your certificate.....  
END CERTIFICATE----";  
        String serverCertStr = "-----BEGIN CERTIFICATE----.your certificate.....  
-----
```

```
END CERTIFICATE-----;

try {

    //generate the private key.
    PrivateKey privateKey = genPrivateKey(privateKeyStr);
    //generate the KeyManager of client, which includes client certificate and private key.
    KeyManager[] km = genKeyManager(clientCertStr, privateKey);
    //generate the trust Manager.
    TrustManager[] trustManagers = genTrustManager(serverCertStr);
    SSLContext ctx = SSLContext.getInstance("TLS");
    ctx.init(km, trustManagers, null);
    MqttConnectOptions options = new MqttConnectOptions();
    options.setCleanSession(true);
    options.setUserName(username);
    options.setSocketFactory(ctx.getSocketFactory());
    System.out.println("initial client");
    MemoryPersistence persistence = new MemoryPersistence();

    //java-client为标识设备的ID，用户可自己定义，在同一个实例下，每个实体设备需要
    有一个唯一的ID
    MqttClient client = new MqttClient(endpoint, "your_id", persistence);
    System.out.println("Connecting to broker: "+endpoint);

    client.connect(options);
    System.out.println("Connected");
    System.out.println("subscribing topic");
    client.subscribe(topic);

    MqttMessage message = new MqttMessage();
    message.setPayload("15".getBytes());
    System.out.println("publishing msg to broker");
    client.publish(topic, message);

    client.disconnect();
} catch (Exception e) {
    e.printStackTrace();
}

}

public static PrivateKey genPrivateKey(String key) throws Exception {

    String privateKeyPEM = key;
```

```
privateKeyPEM = privateKeyPEM.replace("-----BEGIN RSA PRIVATE KEY-----", "");
privateKeyPEM = privateKeyPEM.replace("-----END RSA PRIVATE KEY-----", "");
System.out.println(privateKeyPEM);
BASE64Decoder b64 = new BASE64Decoder();
byte[] decoded = b64.decodeBuffer(privateKeyPEM);
PKCS8EncodedKeySpec keySpecPKCS8 = new PKCS8EncodedKeySpec(decoded);
KeyFactory kf = KeyFactory.getInstance("RSA");
PrivateKey privKey = kf.generatePrivate(keySpecPKCS8);
return privKey;
}

public static KeyManager[] genKeyManager(String clientCertStr, PrivateKey privateKey) throws Exception {
    BASE64Decoder Base64 = new BASE64Decoder();
    byte [] decoded_cert = Base64.decodeBuffer(clientCertStr.replaceAll(X509Factory.BEGIN_CERT, ""
        .replaceAll(X509Factory.END_CERT, "")));
    Certificate clientCert = CertificateFactory.getInstance("X.509")
        .generateCertificate(new ByteArrayInputStream(decoded_cert));
    KeyStore keyStore = KeyStore.getInstance("JCEKS");
    keyStore.load(null);
    keyStore.setCertificateEntry("cert-alias", clientCert);
    keyStore.setKeyEntry("key-alias", privateKey, "changeit".toCharArray(), new Certificate[] {clientCert});
    KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
    kmf.init(keyStore, "changeit".toCharArray());
    KeyManager[] km = kmf.getKeyManagers();
    return km;
}

public static TrustManager[] genTrustManager(String serverCertStr) throws Exception {
    BASE64Decoder Base64 = new BASE64Decoder();
    TrustManagerFactory tmf = TrustManagerFactory.getInstance("X509");
    KeyStore keyStore = KeyStore.getInstance(KeyStore.getDefaultType());
    keyStore.load(null );
    byte [] decoded_server_cert = Base64.decodeBuffer(serverCertStr.
        .replaceAll(X509Factory.BEGIN_CERT, "")
        .replaceAll(X509Factory.END_CERT, ""));
    Certificate serverCert = CertificateFactory.getInstance("X.509")
        .generateCertificate(new ByteArrayInputStream(decoded_server_cert));
    keyStore.setCertificateEntry("ca", serverCert);
    tmf.init((KeyStore) null);
    TrustManager[] trustManagers = tmf.getTrustManagers();
    return trustManagers;
}
```

}

8.12 版本说明

8.12.1 v0.10.13

首次发布。

第9章 Java SDK文档(设备型)

9.1 概述

本文档主要介绍IoT Device SDK的安装和使用。在使用本文档前，您需要先了解IoT Device的一些基本知识，并已经开通了Iot Device服务。若您还不了解Iot Device，可以参考[产品描述和操作指南](#)。

9.1.1 安装SDK工具包

运行环境

Java SDK工具包可在jdk1.6、jdk1.7、jdk1.8环境下运行。

方式一：使用Maven安装

在Maven的pom.xml文件中添加bce-java-sdk的依赖：

```
<dependency>
    <groupId>com.baidubce</groupId>
    <artifactId>bce-java-sdk</artifactId>
    <version>{version}</version>
</dependency>
```

其中，`{version}`为版本号，可以在[SDK下载页面](#)找到。

方式二：直接使用JAR包安装

1. 在[官方网站](#)下载Java SDK压缩工具包。
 2. 将下载的**bce-java-sdk-version.zip**解压后，复制到工程文件夹中。
 3. 在Eclipse右键“工程 -> Properties -> Java Build Path -> Add JARs”。
 4. 添加SDK工具包**lib/bce-java-sdk-version.jar**和第三方依赖工具包**third-party/*.jar**。
- 其中，`version`为版本号。

SDK目录结构

```

com.baidubce
    └── auth                                //BCE签名相关类
    └── http                                 //BCE的Http通信相关类
    └── internal                            //SDK内部类
    └── model                               //BCE公用model类
    └── services
        └── iotdm                           //物管理服务相关类
            └── model                         //物管理内部model, 如Request

或Response
    └── IotDmV3Client.class                  //物管理客户端入口类
    └── util                                //BCE公用工具类
    └── BceClientConfiguration.class         //对BCE的HttpClient的配置
    └── BceClientException.class             //BCE客户端的异常类
    └── BceServiceException.class           //与BCE服务端交互后的异常类
    └── ErrorCode.class                     //BCE通用的错误码
    └── Region.class                        //BCE提供服务的区域

```

9.2 创建IotDmV3Client

用户可以参考如下代码创建一个IotDmV3Client:

[HTTP Client](#)

```

String ACCESS_KEY_ID = <your-access-key-id>; // 用户的Access Key ID
String SECRET_ACCESS_KEY = <your-secret-access-key>; // 用户的Secret Access Key
String ENDPOINT = "iotdm.gz.baidubce.com";

// 创建配置
BceClientConfiguration config = new BceClientConfiguration()
.withCredentials(new DefaultBceCredentials(ACCESSKEY, SECRETKEY))
.withEndpoint(ENDPOINT);

// 初始化一个IotDmV3Client
IotDmV3Client client = new IotDmV3Client(config);

```

[HTTPS Client](#)

```

String ACCESS_KEY_ID = <your-access-key-id>; // 用户的Access Key ID
String SECRET_ACCESS_KEY = <your-secret-access-key>; // 用户的Secret Access Key
String ENDPOINT = "iotdm.gz.baidubce.com";

// 创建配置

```

```
BceClientConfiguration config = new BceClientConfiguration()  
.withProtocol(Protocol.HTTPS) // 使用HTTPS协议  
.withCredentials(new DefaultBceCredentials(ACCESSKEY, SECRETKEY))  
.withEndpoint(ENDPOINT);  
  
// 初始化一个IotDmV3Client  
IotDmV3Client client = new IotDmV3Client(config);
```

在代码中，变量ACCESS_KEY_ID与SECRET_ACCESS_KEY是系统分配给用户的，均为字符串，用于标识用户，为访问物管理做签名认证。其中ACCESS_KEY_ID对应控制台中的“Access Key ID”，SECRET_ACCESS_KEY对应控制台的“Access Key Secret”，获取方式请参考[获取AK/SK](#)。

参数说明

BceClientConfiguration中有更多的配置项，可配置如下参数：

| 参数 | 说明 |
|---------------------------------------|--------------------------|
| connectionTimeo utInMillis | 建立连接的超时时间（单位：毫秒） |
| localAddress | 本地地址 |
| maxConnections | 允许打开的最大HTTP连接数 |
| protocol | 连接协议类型 |
| proxyDomain | 访问NTLM验证的代理服务器的Windows域名 |
| proxyHost | 代理服务器主机地址 |
| proxyPassword | 代理服务器验证的密码 |
| proxyPort | 代理服务器端口 |
| proxyPreemptive AuthenticationEnabled | 是否设置用户代理认证 |
| proxyUsername | 代理服务器验证的用户名 |
| proxyWorkstation | NTLM代理服务器的Windows工作站名称 |
| retryPolicy | 连接重试策略 |
| socketBufferSizeInBytes | Socket缓冲区大小 |
| socketTimeoutInMillis | 通过打开的连接传输数据的超时时间（单位：毫秒） |
| userAgent | 用户代理，指HTTP的User-Agent头 |

9.3 设备管理

9.3.1 创建设备

创建设备可以参考代码如下：

```
String deviceName = "device_name";      // 设置创建的设备名称
String schemaId = "schema_id";          // 设置绑定的模型ID
String description = "description";     // 设置对该设备的描述

CreateDeviceRequest request = new CreateDeviceRequest()
    .withDeviceName(deviceName)
    .withSchemaId(schemaId)
    .withDescription(description);

DeviceAccessDetailResponse response = client.createDevice(request);

String tcpEndpoint = response.getTcpEndpoint(); // tcp endpoint地址
String sslEndpoint = response.getSslEndpoint(); // ssl endpoint地址
String username = response.getUsername();        // 账户名，用于接入IoT hub服务
String key = response.getKey();                  // 密钥，用于接入IoT hub服务
```

9.3.2 删除设备

删除设备可以参考代码如下：

```
List<String> devices = Arrays.asList("device_name_1", "device_name_2"); // 设置
需要删除的设备列表

DeviceListRequest request = new DeviceListRequest().
    withDevices(devices);

DeviceListResponse response = client.removeDevices(request);

List<String> deletedDevices = response.getDevices(); // 删除设备列表
```

9.3.3 获取设备Profile

获取设备Profile可以参考代码如下：

```
String deviceName = "device_name";      // 设置需要查询的设备名称

DeviceProfileResponse response = client.getDeviceProfile(deviceName);

String name = response.getName();          // 设备名称
String description = response.getDescription(); // 设备描述
String state = response.getState();         // 设备当前状态
String schemaId = response.getSchemaId();   // 设备使用的模版ID
String schemaName = response.getSchemaName(); // 设备使用的模板名称
Long createTime = response.getCreateTime();   // 设备的创建时间
Long lastActiveTime = response.getLastActiveTime(); // 设备最后一次上报内容的时间
Boolean favourite = response.getFavourite();    // 设备的收藏信息
JsonNode attributes = response.getAttributes();  // 设备在服务端设置的属性
DeviceAttributes device = response.getDevice();   // 设备在设备端的属性
```

9.3.4 获取设备View

获取设备View可以参考代码如下：

```
String deviceName = "device_name";      // 设置需要查询的设备名称

DeviceViewResponse response = client.getDeviceView(deviceName);

String name = response.getName();          // 设备名称
String description = response.getDescription(); // 设备描述
String state = response.getState();         // 设备当前状态
String schemaId = response.getSchemaId();   // 设备使用的模版ID
String schemaName = response.getSchemaName(); // 设备使用的模板
名称
Long createTime = response.getCreateTime();   // 设备的创建时间
Long lastActiveTime = response.getLastActiveTime(); // 设备最后一次上
报内容的时间
Boolean favourite = response.getFavourite();    // 设备的收藏信息

int profileVersion = response.getProfileVersion(); // 设备影子的版本号
List<DeviceViewAttribute> devices = response.getProperties(); // 设备所使用模板的
属性
```

9.3.5 获取及查询影子列表

获取及查询影子列表可以参考代码如下：

```

int pageNo = 1;                                // 设置需要获取所有查询结果的第几页
int pageSize = 10;                             // 设置每页返回的最大个数
String orderBy = "name";                      // 设置排序的索引列，支持name/createTime/
lastActiveTime
String order = "asc";                         // 设置按照升序、降序排列结果，支持asc/desc
String name = "schemaName";                   // 设置需要查询的属性名
                                                // 支持设备名字查询(name)/模型名字查询(schemaName)/
服务端属性查询(attributes.***)/设备端属性查询(device.reported.***)
String value = "my_schema_name";   // 设置需要查询的属性值，对于设备端属性查询，如果相应
属性值为字符串类型，需要用""将字符串包裹
String favourite = "all";                     // 设置收藏选择，支持all/true/false

DeviceProfileListResponse response = client.getDeviceProfiles(pageNo, pageSize, orderBy, order, name);

int amount = response.getAmount();           // 满足查询条件的设备数目
int pageNo = response.getPageNo();          // 当前页页码
int pageSize = response.getPageSize();       // 返回的每页的最大个数
List<DeviceProfile> devices = response.getDevices(); // 当前页设备详情列表

```

9.3.6 获取设备接入详情

获取设备接入详情可以参考代码如下：

```

String deviceName = "device_name";      // 设置需要查询的设备名称

DeviceAccessDetailResponse response = client.getDeviceAccessDetail(deviceName);

String tcpEndpoint = response.getTcpEndpoint(); // tcp endpoint地址
String sslEndpoint = response.getSslEndpoint(); // ssl endpoint地址
String username = response.getUsername();        // 账户名，用于接入IoT hub服务
String key = response.getKey();                // 调用此方法密钥默认返回"xxxxxxxxxx",
请在创建设备时妥善保存

```

9.3.7 更新密钥

更新密钥可以参考代码如下：

```

String deviceName = "device_name";      // 设置需要更新的设备名称

DeviceAccessDetailResponse response = client.updateDeviceSecretKey(deviceName);

```

```
String tcpEndpoint = response.getTcpEndpoint(); // tcp endpoint地址
String sslEndpoint = response.getSslEndpoint(); // ssl endpoint地址
String username = response.getUsername(); // 账户名, 用于接入IoT hub服务
String key = response.getKey(); // 更新密钥后会导致原有的连接断开, 需要用新的密钥连接。
```

9.3.8 更新设备属性

更新设备属性可以参考代码如下：

```
String deviceName = "device_name"; // 设置需要更新的设备名称

// 设置需要更新的服务端属性
ObjectNode attributes = new ObjectMapper().createObjectNode();
attributes.put("regionTag", "Shanghai");

// 设置需要更新的设备影子属性期望值
ObjectNode desired = new ObjectMapper().createObjectNode();
desired.put("light", "red");

// 设置需要更新的设备端属性
DeviceAttributes device = new DeviceAttributes()
    .withDesired(desired);

UpdateDeviceProfileRequest request = new UpdateDeviceProfileRequest()
    .withAttributes(attributes)
    .withDevice(device);

DeviceProfileResponse response = client.updateDeviceProfile(deviceName, request);
```

9.3.9 更新设备View

更新设备View可以参考代码如下：

```
String deviceName = "device_name"; // 设置需要更新的设备名称

// 设置需要更新的设备端属性当前汇报值
ObjectNode reported = new ObjectMapper().createObjectNode();
reported.put("light", "red");
```

```
// 设置需要更新的设备端属性期望值
ObjectNode desired = new ObjectMapper().createObjectNode();
desired.put("light", "red");

UpdateDeviceViewRequest request = new UpdateDeviceViewRequest()
    .withReported(reported)
    .withDesired(desired);

DeviceViewResponse response = client.updateDeviceView(deviceName, request);
```

9.3.10 更新设备注册表信息

更新设备注册表信息可以参考代码如下：

```
String deviceName = "device_name";      // 设置需要更新的设备名称

String schemaId = "new_schema_id";        // 设置变更的模板ID
String description = "new_description";   // 设置变更的设备描述
boolean favourite = true;                // 设置变更的收藏选项

UpdateDeviceRegistryRequest request = new UpdateDeviceRegistryRequest()
    .withDescription(description)
    .withSchemaId(schemaId)
    .withFavourite(favourite);

DeviceProfileResponse response = client.updateDeviceRegistry(deviceName, request);
```

9.3.11 重置设备影子

重置设备影子可以参考代码如下：

```
List<String> devices = Arrays.asList("device_name_1", "device_name_2"); // 设置
需要重置的设备列表

DeviceListRequest request = new DeviceListRequest().
    withDevices(devices);

DeviceListResponse response = client.resetDevices(request);
```

9.4 模板管理

9.4.1 创建模板

创建模板可以参考代码如下：

```
String schemaName = "schema_name";      // 设置创建的模板名字
String description = "description";      // 设置创建的模板描述

String propertyName = "property_name";          // 设置
属性名称
String displayName = "display_name";           // 设置属性
显示名称
String unit = "unit";                         // 设置属性单位
String defaultValue = "default_value";         // 设置
属性默认值
SchemaProperty.PropertyType type = SchemaProperty.PropertyType.STRING; // 设置
属性类型，支持数字/布尔值/字符串
// 设置模板属性
SchemaProperty property = new SchemaProperty()
    .withName(propertyName)
    .withDisplayName(DISPLAY_NAME)
    .withUnit(UNIT)
    .withDefaultValue(defaultValue)
    .withType(type);

// 设置模板属性列表
List<SchemaProperty> properties = Arrays.asList(property);
SchemaCreateRequest request = new SchemaCreateRequest()
    .withName(schemaName)
    .withDescription(description)
    .withProperties(properties);

SchemaCreateResponse response = client.createSchema(request);

String schemaId = response.getSchemaId();      // 创建的模板ID
```

9.4.2 删除模板

删除模板可以参考代码如下：

```
String schemaId = "schema_id";      // 设置需要删除的模板ID  
  
client.deleteSchema(schemaId);
```

9.4.3 更新模板

更新模板可以参考代码如下：

```
String schemaId = "schemaId"; // 设置需要更新的模板ID  
  
String description = "new_description"      // 设置变更的模板描述  
  
String propertyName = "new_property_name"      // 设置  
变更的属性名称  
String displayName = "new_display_name"        // 设置  
变更的属性显示名称  
String unit = "new_unit";                      // 设置  
变更的属性单位  
String defaultValue = "new_default_value";      // 设置  
变更的属性默认值  
SchemaProperty.PropertyType type = SchemaProperty.PropertyType.STRING;    // 设置  
变更的属性类型，支持数字/布尔值/字符串  
// 设置变更的模板属性  
SchemaProperty property = new SchemaProperty()  
    .withName(propertyName)  
    .withDisplayName(DISPLAY_NAME)  
    .withUnit(UNIT)  
    .withDefaultValue(defaultValue)  
    .withType(type);  
  
// 设置变更的模板属性列表  
List<SchemaProperty> properties = Arrays.asList(property);  
  
SchemaUpdateRequest request = new SchemaUpdateRequest()  
    .withDescription(description)  
    .withProperties(properties);  
  
client.updateSchema(schemaId, request);
```

9.4.4 获取模板详情

获取模板详情可以参考代码如下：

```
String schemaId = "schema_id";      // 需要查询的模板ID

SchemaResponse response = client.getSchema(schemaId);

String id = response.getId();          // 模板ID
String name = response.getName();      // 模板的名称
String description = response.getDescription(); // 模板的描述
long createTime = response.getCreateTime(); // 模板的创建时间
long lastUpdatedTime = response.getLastUpdatedTime(); // 模板最后一次更新时间
List<SchemaProperty> properties = response.getProperties(); // 模板的属性列表
```

9.4.5 获取及查询模板列表

获取及查询模板列表可以参考代码如下：

```
int pageNo = 1;                      // 设置需要获取所有查询结果的第几页
int pageSize = 10;                    // 设置每页返回的最大个数
String order = "desc";                // 设置按照升序、降序排列结果，支持asc/desc
String orderBy = "createTime";        // 设置排序的索引列，支持name createTime/
lastUpdatedTime
String key = "schema_name";          // 查询关键字，支持模板名称/模板描述

SchemaListResponse response = client.getschemas(pageNo, pageSize, orderBy, order, key);

int totalCount = response.getTotalCount(); // 满足查询条件的模版数目
int pageNo = response.getPageNo();       // 当前页页码
int pageSize = response.getPageSize();    // 返回的每页的最大个数
List<Schema> result = response.getResult(); // 当前页模板详情列表
```

9.5 权限组管理

9.5.1 创建权限组

```
CreateDomainRequest request = new CreateDomainRequest()
    .withName("domainTest")           // 设置权限组名字
    .withType(Domain.DomainType.NORMAL) // 设置权限组类型
    .withDescription("domain description"); // 设置权限组描述

AccessDetailResponse response = client.createDomain(request);
```

```
String userName = response.getUsername(); // tcp协议的物接入endpoint
String tcpEndpoint = response.getTcpEndpoint(); // 支持ssl的物接入endpoint
String sslEndpoint = response.getWssEndpoint(); // 支持wss的物接入endpoint
String key = response.getKey(); // 物接入Thing密钥
```

9.5.2 删除权限组

```
client.removeDomain("domainTest");
```

9.5.3 获取权限组列表

```
int pageNo = 1;
int pageSize = 10;
String orderBy = "name";
DomainListResponse response = client.getDomains(pageNo, pageSize, orderBy, null, null, null, null);
Domain domain = response.getDomains().get(0);
String name = domain.getName(); // 权限组名称
Domain.DomainType type = domain.getType(); // 权限组类型
int deviceNum = domain.getDeviceNum(); // 权限组包含的设备数
String description = domain.getDescription(); // 权限组描述
long createTime = domain.getCreateTime(); // 权限组创建的时间
long lastUpdatedTime = domain.getLastUpdatedTime(); // 权限组更新的时间
```

9.5.4 获取权限组详情

```
DomainDetail domainDetail = client.getDomainDetail("domainTest");
String name = domainDetail.getName(); // 权限组的名称
List<String> devices = domainDetail.getDevices(); // 设备列表
Domain.DomainType type = domainDetail.getType(); // 权限组类型
int deviceNum = domainDetail.getDeviceNum(); // 设备数量
String description = domainDetail.getDescription(); // 权限组描述
long createTime = domainDetail.getCreateTime(); // 创建时间
long lastUpdatedTime = domainDetail.getLastUpdatedTime(); // 最后更新时间
```

9.5.5 权限组中更改设备

```
client.removeDomain("domainTest");
List<String> addList = new ArrayList<String>(1);
addList.add("deviceName1");
List<String> removedList = new ArrayList<String>(1);
removedList.add("deviceName2");
UpdateDomainDevicesRequest request = new UpdateDomainDevicesRequest();
request.setAddedDevices(addList);
request.setRemovedDevices(removedList);
UpdateDomainDevicesResponse response = client.modifyDomainDevices("domainTest", request);
List<String> addedDevices = response.getAddedDevices(); // 成功添加的设备名称列表
List<String> removedDevices = response.getRemovedDevices(); // 成功移除的设备名称
列表
```

9.5.6 更新权限组注册信息

```
UpdateDomainRegistryInfoRequest request = new UpdateDomainRegistryInfoRequest();
request.setDescription("new domain description");
client.modifyDomainRegistryInfo("domainTest", request); // 更新对应名称的domain的
描述信息
```

9.5.7 获取权限组接入信息

```
AccessDetailResponse accessDetailResponse = client.getDomainAccessDetail("domainTest");
accessDetailResponse.getUsername(); // endpointName/thingName
accessDetailResponse.getTcpEndpoint(); // tcp协议的物接入endpoint
accessDetailResponse.getSslEndpoint(); // ssl协议的物接入endpoint
accessDetailResponse.getWssEndpoint(); // wss的物接入endpoint
accessDetailResponse.getKey(); // 物接入Thing密钥
```

9.5.8 更新权限组密钥

```
AccessDetailResponse accessDetailResponse = client.updateDomainSecretKey("domainTest");
accessDetailResponse.getUsername(); // endpointName/thingName
accessDetailResponse.getTcpEndpoint(); // tcp协议的物接入endpoint
accessDetailResponse.getSslEndpoint(); // ssl协议的物接入endpoint
accessDetailResponse.getWssEndpoint(); // wss的物接入endpoint
accessDetailResponse.getKey(); // 物接入Thing密钥
```

9.5.9 获取及查询设备列表

```
int pageNo = 1;
int pageSize = 10;
String order = "asc";
String orderBy = "name";
DomainDeviceListResponse domainDeviceListResponse = client.getDomainDeviceList("domainTest", pageNo);
int amount = domainDeviceListResponse.getAmount(); // 设备总数
List<DeviceInDomain> devices = domainDeviceListResponse.getDevices(); // 设备列表
DeviceInDomain devicesInDomain = devices.get(0);
String deviceName = devicesInDomain.getDeviceName(); // 设备名称
boolean existed = devicesInDomain.getExisted(); // 是否存在于改权限组内
int domainNum = devicesInDomain.getDomainNum(); // 设备被包含在不同权限组内的数目
```

9.6 规则引擎

9.6.1 创建规则

```
DeviceRuleRequest request = new DeviceRuleRequest();
request.setName("ruleTest");

List<DeviceRuleDestination> destinationList = new ArrayList<DeviceRuleDestination>(1);
DeviceRuleDestination destination = new DeviceRuleDestination();
destination.setKind(DeviceRuleDestination.KindType.TSDB);
destination.setValue("test222.tsdb-106tunjjq367.tsdb.iot.gz.baidubce.com");
destinationList.add(destination);
request.setDestinations(destinationList);

List<DeviceRuleSource> sourceList = new ArrayList<DeviceRuleSource>(1);
DeviceRuleSource source = new DeviceRuleSource();
source.setCondition(">=");
source.setType(SchemaProperty.PropertyType.STRING);
source.setName("temperature");
sourceList.add(source);
request.setSources(sourceList);

DeviceRuleResponse deviceRuleResponse = client.createDeviceRule("deviceName", request);
String id = deviceRuleResponse.getId(); // 规则对应的id
String deviceName = deviceRuleResponse.getDeviceName(); // 规则对应的设备名称
String name = deviceRuleResponse.getName(); // 规则名称
```

```
List<DeviceRuleSourceDetail> sources = deviceRuleResponse.getSources(); // 规则的具体约束条件
List<DeviceRuleDestinationDetail> destinations = deviceRuleResponse.getDestinations(); // 处理后的消息写往的目的的数组
boolean enable = deviceRuleResponse.getEnable(); // 条件是否开启
long createTime = deviceRuleResponse.getCreateTime(); // 创建时间
long updateTime = deviceRuleResponse.getUpdateTime(); // 更新时间
```

9.6.2 修改规则

```
String deviceName = "deviceTest";
DeviceRuleRequest request = new DeviceRuleRequest();
request.setName("ruleName");
List<DeviceRuleSource> sourceList = new ArrayList<DeviceRuleSource>();
DeviceRuleSource source = new DeviceRuleSource();
source.setCondition(">=");
source.setType(SchemaProperty.PropertyType.STRING);
source.setName("temperature");
sourceList.add(source);
request.setSources(sourceList);
List<DeviceRuleDestination> destinationList = new ArrayList<DeviceRuleDestination>();
DeviceRuleDestination destination = new DeviceRuleDestination();
destination.setKind(DeviceRuleDestination.KindType.TSDB);
destination.setValue("test222.tsdb-106tunjjq367.tsdb.iot.gz.baidubce.com");
destinationList.add(destination);
request.setDestinations(destinationList);
DeviceRuleResponse deviceRuleResponse = client.modifyDeviceRule(deviceName, request);
String id = deviceRuleResponse.getId(); // 规则对应的id
String deviceName2 = deviceRuleResponse.getDeviceName(); // 规则对应的设备名称
String name = deviceRuleResponse.getName(); // 规则名称
List<DeviceRuleSourceDetail> sources = deviceRuleResponse.getSources(); // 规则的具体约束条件
List<DeviceRuleDestinationDetail> destinations = deviceRuleResponse.getDestinations(); // 处理后的消息写往的目的的数组
boolean enable = deviceRuleResponse.getEnable(); // 条件是否开启
long createTime = deviceRuleResponse.getCreateTime(); // 创建时间
long updateTime = deviceRuleResponse.getUpdateTime(); // 更新时间
```

9.6.3 删除规则

```
String deviceName = "deviceTest";
```

```
client.removeDeviceRule(deviceName);
```

9.6.4 禁用规则

```
String deviceName = "deviceTest";
client.disableDeviceRule(deviceName);
```

9.6.5 启动规则

```
String deviceName = "deviceTest";
client.enableDeviceRule(deviceName);
```

9.7 物管理转储TSDB

9.7.1 创建带TSDB格式的规则

```
String deviceName = "deviceTest";
DeviceFormatRuleRequest request = new DeviceFormatRuleRequest();
request.setName("TsdbRule");
List<DeviceRuleSource> sourceList = new ArrayList<DeviceRuleSource>();
DeviceRuleSource source = new DeviceRuleSource();
source.setCondition(">=");
source.setType(SchemaProperty.PropertyType.STRING);
source.setName("temperature");
sourceList.add(source);
request.setSources(sourceList);
List<DeviceRuleDestination> destinationList = new ArrayList<DeviceRuleDestination>();
DeviceRuleDestination destination = new DeviceRuleDestination();
destination.setKind(DeviceRuleDestination.KindType.TSDB);
destination.setValue("test222.tsdb-106tunjjq367.tsdb.iot.gz.baidubce.com");
destinationList.add(destination);
request.setDestinations(destinationList);
DeviceRuleFormat deviceRuleFormat = new DeviceRuleFormat();
deviceRuleFormat.setMetric("newMetricName");
deviceRuleFormat.setMode(DeviceRuleFormat.ModeType.FIELD);
request.setFormat(deviceRuleFormat);
DeviceFormatRuleResponse response = client.createTsdbFormatRule(deviceName, request);
String id = response.getId(); // 规则对应的id
```

```

String deviceName2 = response.getDeviceName(); // 对应的设备名称
String name = response.getName(); // 规则名称
List<DeviceRuleSourceDetail> sources = response.getSources(); // 规则的具体约束条件
List<DeviceRuleDestinationDetail> destinations = response.getDestinations(); // 处理后的消息写往的目的地数组
DeviceRuleFormat format = response.getFormat(); // 转存TSDB数据的格式定义
boolean enable = response.getEnable(); // 是否开启
long createTime = response.getCreateTime(); // 创建时间
long updateTime = response.getUpdateTime(); // 更新时间

```

获取带TSDB格式的规则详情

```

String deviceName = "deviceTest";
DeviceFormatRuleResponse response = client.getTsdbFormatRule(deviceName);
String id = response.getId(); // 规则对应的id
String deviceName2 = response.getDeviceName(); // 对应的设备名称
String name = response.getName(); // 规则名称
List<DeviceRuleSourceDetail> sources = response.getSources(); // 规则的具体约束条件
List<DeviceRuleDestinationDetail> destinations = response.getDestinations(); // 处理后的消息写往的目的地数组
DeviceRuleFormat format = response.getFormat(); // 转存TSDB数据的格式定义
boolean enable = response.getEnable(); // 是否开启
long createTime = response.getCreateTime(); // 创建时间
long updateTime = response.getUpdateTime(); // 更新时间

```

9.7.2 修改带TSDB格式的规则

```

String deviceName = "deviceTest";
DeviceFormatRuleRequest request = new DeviceFormatRuleRequest();
request.setName("TsdbRule");
List<DeviceRuleSource> sourceList = new ArrayList<DeviceRuleSource>();
DeviceRuleSource source = new DeviceRuleSource();
source.setCondition(">=");
source.setType(SchemaProperty.PropertyType.STRING);
source.setName("temperature");
sourceList.add(source);
List<DeviceRuleDestination> destinationList = new ArrayList<DeviceRuleDestination>();
DeviceRuleDestination destination = new DeviceRuleDestination();
destination.setKind(DeviceRuleDestination.KindType.TSDB);
destination.setValue("test222.tsdb-106tunjjq367.tsdb.iot.gz.baidubce.com");

```

```
destinationList.add(destination);
request.setSources(sourceList);
request.setDestinations(destinationList);
DeviceRuleFormat deviceRuleFormat = new DeviceRuleFormat();
deviceRuleFormat.setMetric("newMetricName");
deviceRuleFormat.setMode(DeviceRuleFormat.ModeType.FIELD);
request.setFormat(deviceRuleFormat);
client.modifyTsdbFormatRule(deviceName, request);

DeviceFormatRuleResponse response = client.getTsdbFormatRule(deviceName);
String id = response.getId(); // 规则对应的id
String deviceName2 = response.getDeviceName(); // 对应的设备名称
String name = response.getName(); // 规则名称
List<DeviceRuleSourceDetail> sources = response.getSources(); // 规则的具体约束条件
List<DeviceRuleDestinationDetail> destinations = response.getDestinations(); // 处理后的消息写往的目的地数组
DeviceRuleFormat format = response.getFormat(); // 转存TSDB数据的格式定义
boolean enable = response.getEnable(); // 是否开启
long createTime = response.getCreateTime(); // 创建时间
long updateTime = response.getUpdateTime(); // 更新时间
```

9.8 版本说明

9.8.1 v0.10.34

- 版本号: v0.10.34
- 时间: 2018-09-20
- 新增权限组管理，物管理到规则引擎，以及物管理转储TSDB的接口。

9.8.2 v0.10.21

- 版本号: v0.10.21
- 时间: 2017-08-19
- 首次发布。

第10章 IoT Edge SDK

10.1 IoT Edge SDK

为了帮助开发者能更高效地将各类设备与云端互联，并利用天工物联网平台完善的接入、存储、计算和分析能力打造物联网应用，天工物联网推出了完全开源开放的IoT Edge SDK。

IoT Edge SDK包含了物接入(IoT Hub)的C语言客户端、序列化和反序列化、设备管理、协议解析等功能组件，涵盖了实现设备上云时在断线缓存、在线检测、设备管理、数据安全传输等场景，同时更多的服务也将在未来开放。

如须下载SDK，或者了解更详细的IoT Edge SDK使用说明，请访问Github：

- <https://github.com/baidu/iot-edge-c-sdk>

同时提供了基于ST的B-L475E-IOT01物联网开发套件与百度云天工连接的例程，请访问Github：

- <https://github.com/baidu/iot-edge-sdk-samples>

天工物接入服务完全兼容标准MQTT协议。对于仅需将设备数据与云端进行双向通讯的用户，也可以使用众多的第三方MQTT SDK来进行开发。比如Paho MQTT SDK。

10.2 移植百度天工SDK到其他平台

10.2.1 介绍

本文档主要介绍如何移植C公共库到其他目前还未支持的平台。

注：当前百度云天工的IoT Hub的C语言SDK和百度云BOS的C语言SDK都是使用该C公共库，下述内容详细描述如何移植该C公共库至其他平台。

本文档不会特定介绍哪一个具体的平台移植方案，是一个通用的移植方案。

10.3 参考

规范

- tickcounter adapter specification
- agenttime adapter specification
- threadapi and sleep adapter specification
- platform adapter specification
- tlsio specification
- xio specification
- lock adapter specification

头文件列表

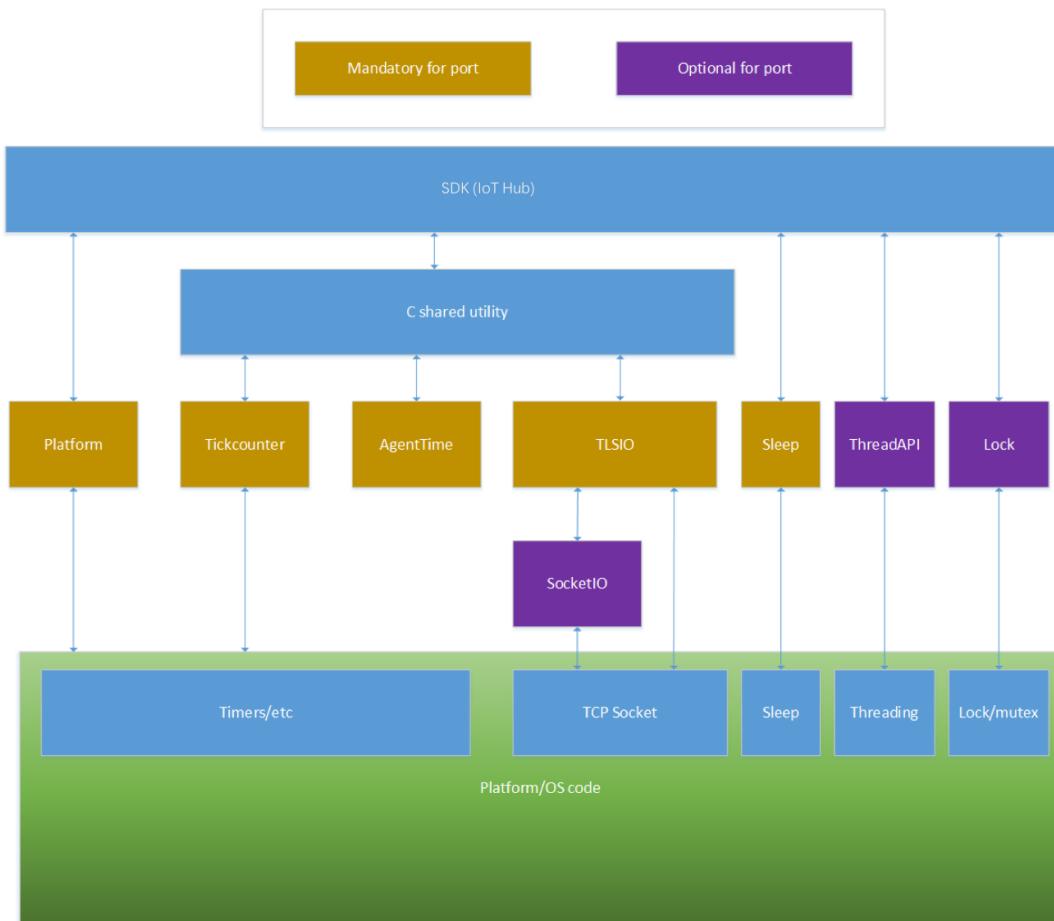
- tickcounter.h
- threadapi.h
- xio.h
- tlsio.h
- socketio.h
- lock.h
- platform.h

目录结构

- 整体概括
- tickcounter 适配器
- agenttime适配器
- sleep适配器
- platform 适配器
- tlsio adapter 适配器
- socketio 适配器概括(可选)
- tlsio适配器实现
- threadapi and lock适配器(可选)

10.3.1 概括

目前这个采用C99标准编写的C公共库可以很方便的移植到大部分平台，其中有几个组件依赖平台的特定资源来实现所需的功能。因此，这个C公共库提供PAL (platform abstraction layer) 模块来让这个库可以适配到特定平台。下面就是这个模块的整体架构：



必须适配的几个模块如下：

1. tickcounter的实现：这个接口提供一个可以获取以毫秒为单位时钟计数器。精度不一定必须是毫秒级别，但是这个接口的返回值必须是以毫秒为单位。
2. agenttime的实现：这个接口提供c运行时的函数，包括获取time, difftime等等。这个主要由于不同平台对具体实现的处理方式不一样。
3. sleep函数的实现提供一个平台无关的sleep逻辑。
4. platform接口提供执行全局初始化和反初始化的逻辑。
5. tlsio接口提供标准的基于TLS之上的通讯方式。 IoT Hub不允许不安全接入方式。

另外，有两个可选的模块，他们是threadapi和lock，这两个模块允许SDK通过特定的线程来进行数据通讯。另外一个需要适配的模块是socketio适配器，这个需要配合tlsio不同实现方式。

目前该SDK已经包含了一个标准适配实现，你可以在SDK的特定目录下找到他们。如果其中的一些适配器满足你的设备需求的话，可以直接引用这些文件到你的项目。

10.3.2 tickcounter适配器

Tickcounter的行为定义在tickcounter适配器的规范中指定。你可以通过复制一份tickcounter.c文件，并进行修改来适配你的设备，同时tickcounter规范包含了对内存大小的优化方案建议。

10.3.3 agenttime适配器

agentime适配器的逻辑在agentime适配器规范里面做了详细介绍，并提供一个与使用平台无关的time函数。

对于大多数平台/操作系统，您可以在构建中包含标准agentime.c文件。该适配器只是调用C函数time, difftime, ctime等。

如果这个文件在你的平台无法工作的，请复制该文件并进行相应的修改。

百度IoT SDK只需要get_time和get_difftime这两个函数。get_gmtime, get_mktime和get_ctime已经被弃用了，可以直接忽略不管。

10.3.4 sleep适配器

这个sleep适配器只是提供一个函数，提供一个平台无关的线程sleep函数。不同于其他适配器，这个适配器没有自己的头文件。他是将这个函数定义在threadapi.h，这个头文件还包含其他的一些可选的适配函数，这个函数的实现通常定义在threadapi.c文件。

除了ThreadAPI_Sleep是SDK必须的函数，threadapi.h里面其他的函数都是可选的。

sleep适配器的规范可以在threadapi和sleep适配器规范里面找到。

10.3.5 platform适配器

platform适配器主要执行一次性平台的初始化和反初始化，同时也提供SDK的TLSIO相关的支持。

该平台适配器规范给出了编写平台适配器的完整说明。

要开始创建您的平台适配器，请复制此platform.c文件 并进行修改以满足您的需求。

10.3.6 threadapi和lock适配器

编译SDK必须包含threadapi和lock这个两个适配器，但是功能实现是可选的。他们的规范文档详细说明了如果不需线程（threading）相关功能，每个空函数应该做什么。

这个模块允许SDK通过指定的线程和IoT Hub交互。特定的线程执行特定任务有很多额外的开销，原因在于需要为每个线程分为独立的stack，这样对于某些资源受限的设备来说，就比较困难了。

试用特定线程执行tlsio相关逻辑的好处就是当网络不可用时，可以尝试重新连接IoT Hub，同时还有其他线程可以处理其他逻辑，这样程序的响应会比较及时。

以后版本的SDK会移除潜在的blocking的行为，不过现在仍然需要指定线程来处理消息，提高系统的响应，这样就需要实现ThreadAPI和Lock适配器。

下面是threadapi和lock适配器规范：

- threadapi 和 sleep 适配器规范
- lock 适配器规范

这个规范介绍当不需要线程的时候，如何去创建null lock和threadapi的适配器。

如果需要创建自己的threadapi和lock适配器，可以拷贝windows的适配文件，通过适当修改来满足你的需求：

threadapi.c

lock.c

10.3.7 tlsio适配器介绍

tlsio适配器提供SDK，可以通过标准的安全的TLS通讯方式让device和IoT Hub进行数据交互。

Tlsio适配器通过xio接口暴露功能让SDK来调用，这个接口提供一个输入为bits，返回也为bits的接口，具体定义可以访问：https://github.com/Azure/azure-c-shared-utility/blob/master/inc/azure_c_shared_utility/xio.h

通过调用函数xio_create来创建tlsio适配器实例，在创建tlsio实例的时候，你必须配置参数const void *io_create_parameteres，这个参数是TLSIO_CONFIG的一种类型。

tlsio有两种模式：

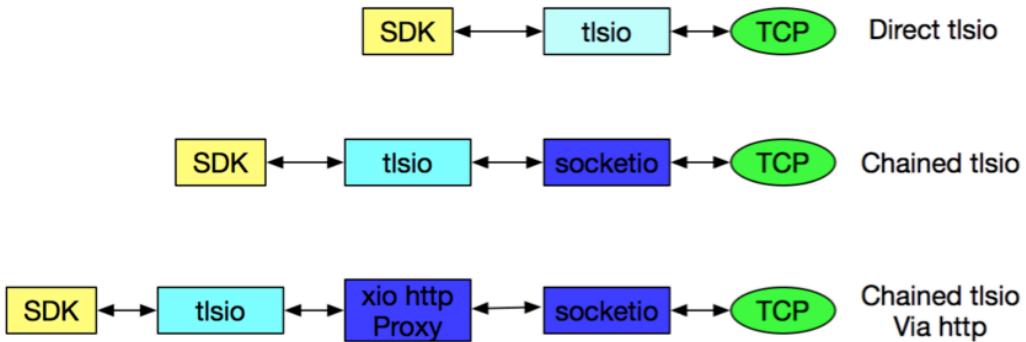
tlsio支持的模式包括两种：直接的，串联的。在直接模式，tlsio适配器直接创建自己的TCP socket，试用TCP直接和远程的服务器进行TLS通讯。在串联模式下，tlsio适配器不拥有自己的TCP socket，不直接和远程服务器通讯，但是它仍然处理TLS的所有逻辑，包括加密，解密，协商，只不过通过另外的方式和远程服务器进行通讯。

直接模式的好处是消耗资源会少很多，比较适合MCU，比如Arduino和ESP32等等。串联模式提供更大的灵活度，但是资源消耗也会多很多，所以主要主流的OS，比如windows，linux和Mac等等。

10.3.8 socketio适配简介

串联的tlsio适配器必须调用xio的适配器，这个适配器可以包含一个tcp socket。在百度的 IoT SDK里面，xio适配器是通过socketio管理tcp socket。

tls的数据流模型介绍：



10.3.9 tlsio适配器实现

开发tlsio适配器功能相对来说还是很复杂的，需要很有经验的开发人员才能完成，需要对协议有很深的理解。

tlsio适配器相关的逻辑都是在C公共库实现的，不是百度 IoT SDK的一部分。你可以参考下面的资料去了解如何开发。

有两个直接模式tlsio适配器实现

1. `tlsio_openssl_compact` for ESP32: https://github.com/Azure/azure-iot-pal-esp32/blob/master/pal/src/tlsio_openssl_compact.c

`tlsio_openssl_compact` for ESP32是一个好的例子，如果需要适配的话，可以拷贝一份，基于这个修改来满足自己的需求。

`tlsio_openssl_compact` for ESP32提供类两个文件，这两个文件是和具体平台无关的 `socket_async.c`

https://github.com/Azure/azure-c-shared-utility/blob/master/pal/socket_async.c

https://github.com/Azure/azure-c-shared-utility/blob/master/pal/dns_async.c

大部分的用户都可以直接使用这两个文件而不需要修改，对于特殊情况只需要修改 `socket_async_os.h`就可以了。

2. `tlsio_arduino` for Arduino:

https://github.com/Azure/azure-iot-pal-arduino/blob/master/pal/src/tlsio_arduino.c

10.3.10 支持设备支持仓库

推荐参考<https://github.com/Azure/azure-iot-pal-esp32>，来创建一个自己的仓库。

第11章 物接入IoT Hub服务等级协议（SLA）

11.1 IoT Hub SLA

本协议自2018年2月1日生效

服务等级协议

本服务等级协议（Service Level Agreement，简称“SLA”）规定了百度云向天工客户提供的物接入IoT Hub服务的服务可用性等级指标及赔偿方案。

服务可用性

物接入的服务可用性：不低于99.9%。

- 我们保证部署在物接入IoT Hub服务上的服务至少有99.9%的时间能够向注册设备发送消息并且从注册设备接收消息，并且物接入IoT Hub服务能够完成创建、读取、更新和删除操作。
- 当客户端连接上服务端时，服务端保证连接的成功率不低于99.9%，客户端当地网络情况若有异常则不包含在服务可用性范围内。
- 消息的发送与接收的成功率在QOS=0的情况下不做保证

服务周期：可用性按服务周期统计，一个服务周期为一个自然月，如不满一个月不计算为一个服务周期。

服务费用：客户在一个服务周期内为物接入IoT Hub服务所支付的服务费用。

有效消息：物接入服务从设备端、服务端成功接收的所有消息。不包括由于用户操作不当、产品限制、设备不在线以及网络等原因导致的消息接受不成功的情形。例如：

- 调用服务器接口返回非5xx错误的请求（5xx是系统异常），例如请求太频繁被限流，或者网路不通；
- 设备上报太频繁被限流的消息，单个设备QoS=0限制为30QPS，单个设备QoS=1限制为10QPS；
- 系统转发给设备被限流的消息，单个设备限制50QPS；
- 由于本地网络故障没有成功发送给设备的消息；
- 选用MQTT QoS=0的通信机制，但是由于设备不在线导致被丢弃的消息；

失败消息：因物接入IoT Hub服务系统故障导致的未能成功流转的有效消息。失败消息举例如下：

- 设备连接正常，但是由于系统异常导致设备上行失败的消息
- 设备发送消息成功到物接入IoT Hub服务，但是由于系统异常导致服务端不能成功收到。
- 用户调用接口发送指令给在线设备，但是由于系统异常导致不能发送给设备的消息。

不可用时间：物接入IoT Hub服务所提供的服务在连接不上后的连续的5分钟或更长时间内不可使用方计为不可用时间，不可使用的服务时间低于5分钟的，不计入不可用时间。

物接入IoT Hub服务不可用时间不包括如下原因：

- 日常系统维护时间
- 百度云预先通知客户后进行系统维护所引起的，包括割接、维修、升级和模拟故障演练；
- 任何百度云所属设备以外的网络、设备故障或配置调整引起的；
- 客户的应用程序受到黑客攻击而引起的；
- 客户维护不当或保密不当致使数据、口令、密码等丢失或泄漏所引起的；
- 客户的疏忽或由客户授权的操作所引起的；
- 客户未遵循百度云产品使用文档或使用建议引起的；
- 不可抗力引起的。

物接入IoT Hub服务可用性的计算方法如下：

- 服务可用性=((有效消息-失败消息)/有效消息)×100%
- 物接入IoT Hub服务的服务可用性不低于99.9%，如未达到上述可用性承诺，客户可以根据本协议约定获得赔偿。
- 赔偿范围不包括上述服务不可用时间：
- 月总消息量低于100万的用户不做统计；
- 对于免费版的物接入IoT Hub服务不提供SLA保障

服务赔偿条款

- 赔偿标准

物接入IoT Hub服务按一个服务周期的服务可用性，按照下表中的标准计算赔偿金额。

- 赔偿只针对使用物接入IoT Hub服务已产生费用的用户，以物接入IoT Hub服务代金券的形式赔偿，不折算现金返还；
- 赔偿总额不超过未达到服务可用性承诺当月客户就该物接入IoT Hub服务支付的该服务周期的服务费用（不含用代金券抵扣的费用）。

- 服务可用性与对应赔偿代金券金额
- 低于99.9%但等于或高于99.00%——该服务周期的服务费用的10%
- 低于99.00%但等于或高于95.00%——该服务周期的服务费用的25%
- 低于95.00% ——该服务周期的服务费用的100%
- 赔偿申请时限

客户可以在每个自然月第五（5）个工作日后对两个月内没有达到可用性承诺提出赔偿申请。赔偿申请必须限于在物接入IoT Hub服务没有达到可用性的相关月份结束后两（2）个月内提出。超出申请时限的赔偿申请视为客户放弃请求的权利。

数据可销毁性

用户主动删除的数据，该数据将无法复原。

数据私密性

用户存储在物接入服务上的数据，在未经用户合法授权的情况下，其他用户无法访问其数据。百度云的所有运维操作都会被记录并保存。

数据知情权

除应当地法律法规、或政府监管部门的监管、审计要求，用户的所有数据、应用及行为日志不会提供给第三方。用户的所有数据不会被存在境外的数据中心，也不会被用于境外业务或数据分析。目前百度云的数据中心位于北京。用户的数据存储在用户选择的区域（数据中心）。

业务可审查性

百度不会在未经合法用户授权时，公开、编辑或透露用户的个人信息及保存在百度云物接入中的内容，除非有下列情况：有关法律、法规规定或百度云物接入合法服务程序规定；在紧急情况下，为维护用户及公众的权益；为维护百度的商标权、专利权及其他任何合法权益；其他依法需要公开、编辑或透露个人信息的情况。

服务变更、终止条款

鉴于网络服务的特殊性，用户了解并同意，物接入服务有权变更、中断或终止部分或全部的物接入服务。如需变更、中断或终止服务，物接入服务会通过百度云站内信，邮件，或者短信的方式对用户进行通知。

服务故障恢复能力

百度云为付费用户的云服务提供7×24小时的运行维护，并以在线工单和电话报障等方式提供技术支持，具备完善的故障监控、自动告警、快速定位、快速恢复等一系列故障应急响应机制。

其他

百度云有权对本SLA条款作出修改。如本SLA条款有任何修改，百度云将提前30天以网站公示或发送邮件的方式通知您。如您不同意百度云对SLA所做的修改，您有权停止使用物接入服务，如您继续使用物接入服务，则视为您接受修改后的SLA。

第12章 常见问题

12.1 产品配置操作问题

12.1.1 物接入的设备型和数据型如何选择？

物接入（设备型）即原物管理，在提供原生 MQTT 支持的同时，引入基于设备的物影子、物模型等概念，提供云端的状态暂存、直接获取在线状态、数据可以直接存储TSDB、物影子数据可以直接在物可视中展示。对以设备概念构建的物联网场景中，理解将更容易，使用将更便捷。

物接入（数据型）仅提供MQTT协议层接入支持，开发者可基于MQTT数据流做物联网应用开发，数据存储TSDB需要通过规则引擎。要求开发者对 MQTT 协议本身概念较为熟悉。

12.1.2 物接入项目列表中为什么出现了不是我自己创建的项目？

客户在使用物解析、智能边缘时，也会在物接入中创建项目，物解析的项目名是parser_endpointxxxx，智能边缘的实例名是32位数字或字母。对应项目暂不支持删除。

12.1.3 物接入中是否能够批量创建设备？

可以通过调用open API来批量创建设备，详情可参考[API文档](#)。

12.1.4 物接入设备型提供了哪些 topic？

```
\$baidu/iot/shadow/{deviceName}/update  
\$baidu/iot/shadow/{deviceName}/get  
\$baidu/iot/shadow/{deviceName}/delta  
\$baidu/iot/shadow/{deviceName}/delete  
\$baidu/iot/shadow/{deviceName}/update/documents  
\$baidu/iot/shadow/{deviceName}/update/snapshot
```

用户在物接入设备型中创建设备后，系统默认提供这些topic，每个topic的具体用法请操作指南-物影子操作文档。

12.1.5 可以使用MQTT的SDK连接物影子吗？

可以，按照mqtt的连接方式，对应物接入影子提供的相应topic中发送指定格式数据就可以了。

也可以使用天工官方提供的SDK，SDK中提供了示例demo。

12.1.6 物影子如何获得设备在线状态？

如果使用MQTT开源SDK，用物影子的名称作为clientID连接，云端则可通过该连接来判断是否在线，如果clientID不是物影子的名称，则无法判断是否在线。天工官方提供的SDK，已将该功能集成好。

在设备端（此处用MQTT.fx模拟设备端）用与同物影子名称相同的Client ID连接，且用该Client ID成功向[\\$baidu/iot/shadow/myDeviceName/update](#)发布第一条消息后，控制台物影子即可转为在线状态。如物影子名称和Client ID同为_baidu_sample_pump_instance，且成功向[\\$baidu/iot/shadow/baidusample_pump_instancee/update](#)发布第一条消息。

12.1.7 物影子有什么作用？

物影子是设备在云端的状态暂存。通过reported字段和desired字段来反应设备的上报值和期望值。

12.1.8 编辑设备影子中，reported字段和desired字段代表什么意思？

reported字段代表设备向云端汇报的信息；desired字段代表控制台向云端发送的信息，云端将这些信息发送给设备端。

12.1.9 编辑设备影子中，profileVersion是什么意思？

用于版本控制，每次更新设备影子时的版本号要大于上一个版本号。此字段为可选。

12.1.10 设备上传到云端的字段会覆盖设备影子吗？

设备上传到云端的字段，在设备影子中只会更新该字段的值，设备影子中没有被更新的字段仍保留原值。

12.1.11 影子中“lastActiveTime”字段是什么意思？

lastActiveTime为最后一次活跃时间，记录设备最近一次和云端交互的时间。若一个设备从未与云端交互，则该时间默认为格林威治时间（1970年1月1日）。

12.1.12 物接入的数据能存储到哪里？

物接入的json格式数据，也可以通过规则引擎设置规则，存储到时序数据库、Kafka和转发至另一个物接入主题。

12.2 产品规格及使用限制

12.2.1 我设备数量较多，会超过上限，如何提升？

用户可以通过提交工单申请更高额度。

12.2.2 物接入上传的消息大小有限制吗？

物接入上传的单条消息大小限制是32KB，超过32KB的消息会被丢弃。如果用户上传的消息大于32KB，请在上传前将消息分成多条。

但在计费上，每512Bytes算一条。用户使用的消息条数=实际发送长度/512Bytes，计算结果向上取整。

12.2.3 每个百度云账号可以创建多少个项目？

请参考产品介绍-[系统限制](#)。

12.3 客户端及MQTT SDK相关问题

12.3.1 请问是否有基于FreeRTOS或者RTX操作系统的MQTT SDK源码？

通过paho官网查看，提供了多个版本的C/C++ client，有的是支持posix标准的（unix、linux、windows），有的是支持嵌入式系统的。可以参考这个连接：<https://www.eclipse.org/paho/clients/c/embedded/>

12.3.2 为什么publish的QOS等级为2后，则会断开服务？

目前暂不支持Qos=2的服务。

12.3.3 物接入是否支持消息缓存？

支持。当客户端连接物接入服务时，如果Clean Session位被设置为false，则该连接被认为是持久连接，其具体表现为：当该客户断开后，任何订阅的主题和QoS被设置为1或2的信息都会保存，直到该客户端再次连接上server端，物接入服务支持将该消息保留24小时。若“clean session”被设置为true，当该客户断开后，所有的订阅主题都会被移除。

本方法不推荐大量使用，如果需要保存大量数据，推荐将物接入的数据转发到Kafka。

12.3.4 与物接入服务连接成功，往一个主题发送消息，就直接断开。

此情况一般出现在对应主题无 pub 权限。项目请严格按照影子给定的主题操作；数据型项目请检查策略配置。

12.3.5 连接物接入服务时，出现连接协议错误。

大多是由于选择的端口和期望的协议不一致，正确端口号是TCP：1883， TLS：1884， Websocket:8884。

12.3.6 遗嘱消息的触发条件有哪些？

遗嘱（Will Message）消息必须被存储在服务端并且与这个网络连接关联。网络连接关闭时，服务端必须发布这个遗嘱消息，除非服务端收到Client发送的DISCONNECT报文。

遗嘱消息发布的条件，包括但不限于：

服务端检测到了一个I/O错误或者网络故障。

客户端在保持连接（Keep Alive）的时间内未能通讯。

客户端没有先发送DISCONNECT报文直接关闭了网络连接。

由于协议错误服务端关闭了网络连接。

12.3.7 MQTT客户端网络连接异常，但在keepalive时间内恢复，客户端是否需要重新建立MQTT连接？

需要，如果MQTT客户端异常断开，都需要重新建立MQTT连接并重新订阅主题。

如果MQTT客户端cleansession=false，连接异常断开后，服务器会维护session信息，重新连接后不需要再订阅主题。

12.4 API使用问题

12.4.1 使用API连接物接入时，返回connection timeout错误。

当前客户端的网络不通，需要测试客户端网络是不是好的，可以先访问一下一些公网的网站，比如www.baidu.com，如果访问成功的话，就表示网络ok，如果这个时候还是出现这个错误，估计有可能是防火墙关闭了特定端口，目前我们的MQTT提供服务的端口包括1883 (tcp)，1884 (tls)，8884 (websocket)，简单的测试办法就是telnet xxx.mqtt.iot.gz.baidubce.com port，其中xxx.mqtt.iot.gz.baidubce.com是IoT Hub返回的域名，port是使用协议对应的端口，如果是TCP，就用1883。

如果客户端网络没有问题的话，telnet也不通，这个时候就可能是电信运营商的问题，电信运营商有时候会把特定IP的端口给封闭，导致这些服务受到影响，你可能需要获取解析的IP地址，然后自己一个一个的尝试，获取解析的IP地址：nslookup xxx.mqtt.iot.gz.baidubce.com，然后使用telnet来测试：telnet xxxx_ip port。

如果每一个IP的telnet连接都是失败的话，这个时候很可能就是云端的服务出问题，你需要告知百度云的技术支持团队，可以提交工单，一般来说这种情况的可能性极低。

12.4.2 使用API连接物接入时，返回invalid ClientID

填写的clientID不合法，我们ClientID支持的长度是128，超过之后会报错，clientID的格式必须是下面这些字符的组合”0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ-“

还有一种情况，就是我们的服务有并发连接数限制，当你的并发连接数超过上限的时候（目前是每个实例最大10000个并发连接），我们就会返回这个错误给客户端。

第13章 其他参考

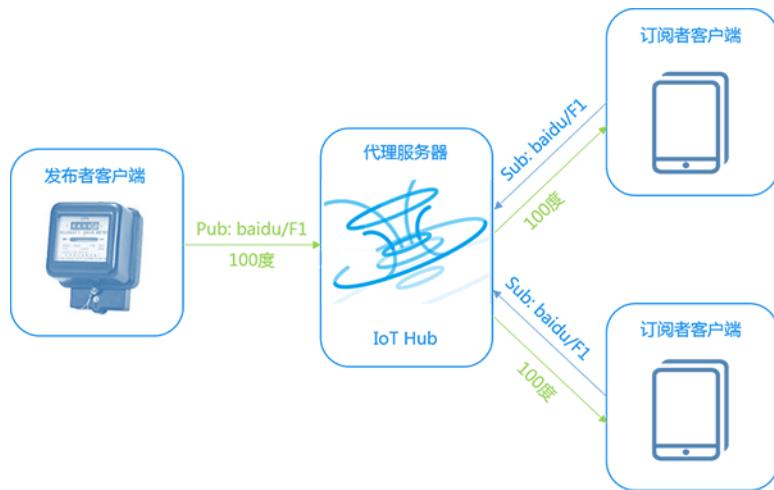
13.1 MQTT协议介绍

13.1.1 MQTT协议是什么？

MQTT (Message Queuing Telemetry Transport Protocol) 的全称是消息队列遥感传输协议的缩写，是一种基于轻量级代理的发布/订阅模式的消息传输协议，运行在TCP协议栈之上，为其提供有序、可靠、双向连接的网络连接保证。

13.1.2 MQTT协议如何工作？

MQTT采用代理的发布/订阅模式实现了发布者和订阅者的解耦(decouple)，因此，在MQTT协议中有三种角色：代理服务器、发布者客户端以及订阅者客户端，其中发布者和订阅者互不干扰，也就是说发布者和订阅者互不知道对方的存在，它们只知道代理服务器，代理服务器负责将来自发布者的消息进行存储处理并将这些消息发送到正确的订阅者中去。



代理服务器 (Server)

代理服务器可以是一个程序或者设备，作为发送消息的客户端和请求订阅的客户端之间的中介。其主要作用是接收发布者客户端发布的应用信息，然后将信息转发给符合条件的订阅者客户端。

百度云天工智能物联网平台为用户提供了代理服务器的功能。

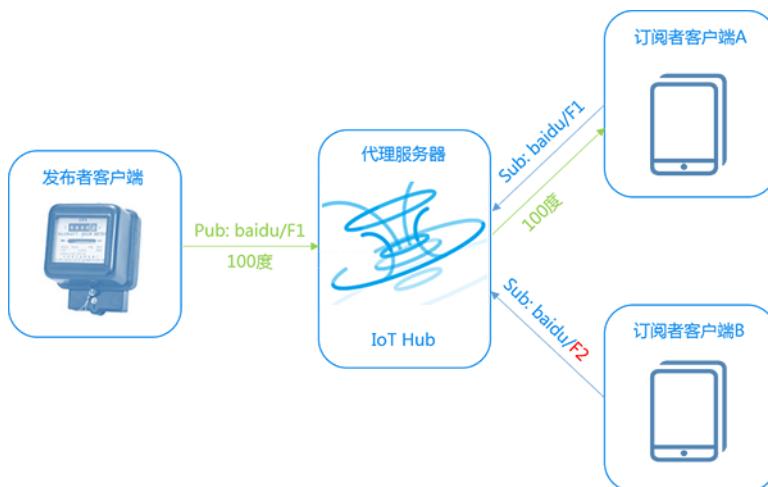
客户端 (Client)

客户端指使用MQTT协议的程序或设备。客户端包括发布者客户端和订阅者客户端，同一个客户端可以即是发布者也是订阅者。客户端可以发布消息给其它相关客户端，也可以订阅其它客户端发布的消息。

因为每个用户的设备和使用场景不同，通常用户需要自己开发客户端软件。MQTT官方提供了[Client SDK](#)，可以帮助客户快速开发MQTT客户端。

13.1.3 如何将消息正确送达？

MQTT通过“主题”实现将消息从发布者客户端送达至接收者客户端。“主题”是附加在应用消息上的一个标签，发布者客户端将“主题”和“消息”发送至代理服务器，代理服务器将该消息转发至每一个订阅了该“主题”的订阅者客户端，如下图所示：



一个主题名可以由多个主题层级组成，每一层通过“/”斜杠分隔开，如上图所示，订阅者客户端A将主题过滤器设置为“baidu/F1”；订阅者客户端B将主题过滤器设置为“baidu/F2”。发布者客户端向“baidu/F1”发布消息，因此只有订阅者客户端A可以接收到该消息。

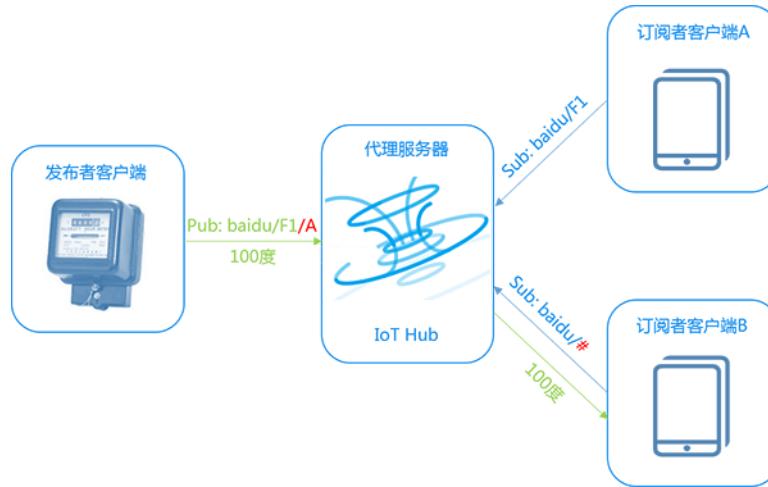
主题过滤器指客户端在订阅时包含的一个表达式，用于表示相关的一个或多个主题。

13.1.4 如何使用通配符订阅多个主题？

如果用户需要一次订阅多个具有类似结构的主题，可以在主题过滤器中包含通配符。通配符只可用在主题过滤器中，在发布应用消息时的主题名不允许包含通配符，主题通配符有两种：

- \#: 表示匹配 ≥ 0 个层次，比如a/#就匹配a/b, a/b/c（不能匹配a/，后面必须有其它主题）。单独的一个#表示匹配所有，不允许a#或a/#/c等形式。

- +：表示匹配一个层次，例如a/+匹配a/b, a/c, 不匹配a/b/c。单独的一个+是允许的，但a+为非法形式。



13.1.5 如何确保消息已被送达？

发布者客户端通过设置PUBLISH报文中的QoS标志位，对于客户端发布的消息提供三种服务质量等级，如下：

- QoS=0，协议对此等级应用信息不要求回应确认，也没有重发机制，这类信息可能会发生消息丢失或重复，取决于TCP/IP提供的尽最大努力交互的数据包服务。
- 最少一次(At least once delivery): QoS=1，确保信息到达，但消息重复可能发生，发送者如果在指定时间内没有收到PUBACK控制报文，应用信息会被重新发送。
- 仅仅一次(Exactlyonce delivery): QoS=2，最高级别的服务质量，消息丢失和重复都是不可接受的。

13.1.6 什么是临终遗嘱？

MQTT协议利用KeepAlive机制在客户端异常断开时发现问题。当客户端断开时（例如：电量耗尽、系统崩溃或者网络断开），代理服务器会采取相应措施。

客户端设置“临终遗嘱”（LWT）信息后，当代理服务器检测到客户端离线后，就会发送保存在特定主题上的LWT信息，让其它订阅该主题的客户端知道该节点已经意外离线。

13.2 MQTT客户端使用指南

13.2.1 Websockets Client

Websockets Client是百度云基于浏览器开发的MQTT客户端测试工具。用户完成物接入配置后，可以通过该工具测试连接性。

有关Websockets Client的使用方法请参看[连接测试](#)

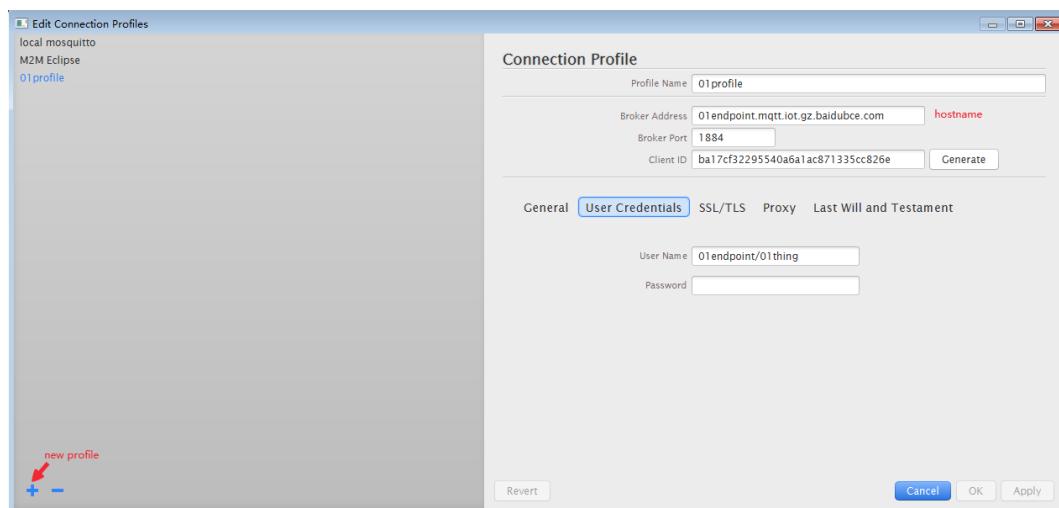
13.2.2 MQTT.fx

MQTT.fx 是目前主流的mqtt客户端，可以快速验证是否可以与IoT Hub 服务交流发布或订阅消息。设备将当前所处的状态作为MQTT主题发送给IoT Hub，每个MQTT主题topic具有不同等级的名称，如“建筑/楼层/温度。” MQTT代理服务器将接收到的主题topic发送给所有订阅的客户端。

13.2.3 连接IoT Hub服务

登录[MQTT.fx下载页面](#)，找到适合的版本下载并安装MQTT.fx客户端。

1. 打开MQTT客户端的设置页面，点击“+”按键，创建一个新的配置文件。



* 填写Connection profile相关信息：

| 参数名称 | 说明 |
|----------------|------------------------|
| profile name | 配置文件名称 |
| Broker Address | 创建endpoint后返回的hostname |

| 参数名称 | 说明 |
|-------------|---|
| Broker Port | ssl加密连接方式，端口使用1884；tcp不加密连接，端口使用1883。 |
| Client ID | 客户端ID，支持“a-z”，“0-9”，“_”，“-”字符，且不能大于128bytes，UTF8编码。在同一个实例下，每个实体设备需要有一个唯一的ID，不同实体设备使用同一个client id建立连接会导致其它连接下线 |

2. 选择User Credential，输入创建 IoT Hub 服务返回的username/password。
3. 如果您选择SSL安全认证方式连接IoT Hub 服务，需要配置SSL/TLS安全认证，勾选 [Enable SSL/TLS](#)，选择[CA signed server certificate](#)认证。
如您选择TCP连接，无需配置SSL安全认证，执行第4步骤即可。
4. 点击“Apply”按键，完成客户端配置。
5. 返回MQTT客户端界面，选择新创建的配置文件，点击“connect”按键连接服务。

13.2.4 订阅消息

注意：

IoT Hub一个主题支持的层级最多是9层（也就是最多只能出现8个斜线“/”）。

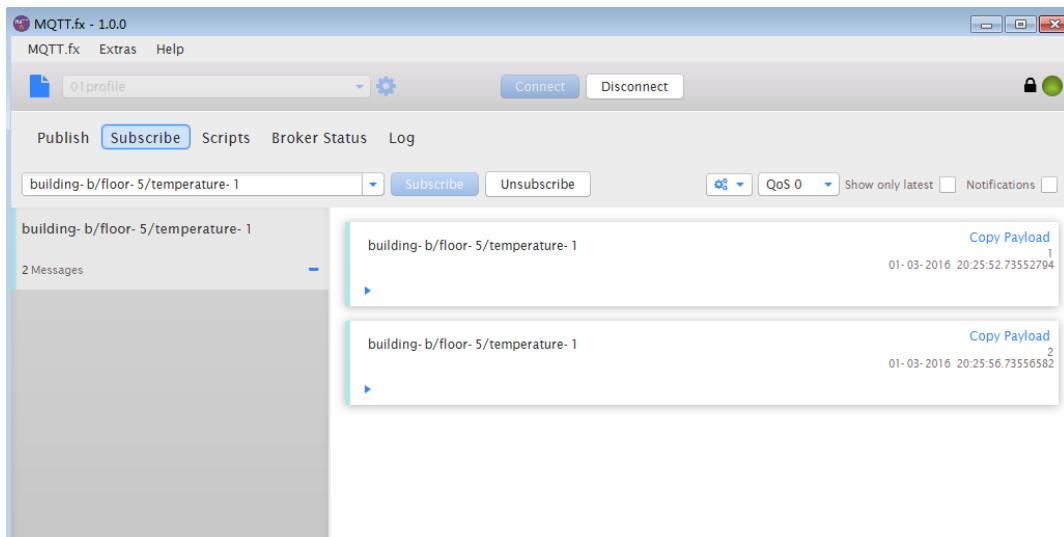
成功连接IoT Hub 服务后，即可开始订阅消息。

打开Subscribe标签，填写主题topic，例如[building-b/floor-5/temperature-1](#)，选择默认的QoS 0，点击“Subscribe”进行订阅操作。

13.2.5 发布消息

打开Publish标签，填写主题topic，例如[building-b/floor-5/temperature-1](#)，选择默认的QoS 0，点击“Publish”进行发布操作。

返回Subscribe界面，即可看到已接收的订阅消息，参见下图。



13.3 MQTT客户端代码示例

13.3.1 C代码示例

下载TLS认证文件 物接入支持SSL/TLS加密传输方式，保障用户的数据传输安全。用户在执行示例代码前，需先下载TLS认证文件，并在代码中指定认证文件的存放路径。

下载[TLS认证文件](#)，并将认证文件保存至示例代码路径下。

下载并执行示例代码 下载[MQTT-c压缩包](#)，解压MQTT-c，目录结构如下：

```
MQTT-c
├── include
├── lib
└── src
    ├── ConnectorSync.c
    ├── PublisherSync.c
    └── SubscriberSync.c
├── Makefile
└── root_cert.pem
```

打开src/PublisherSync.c，配置以下参数：

| 参数名称 | 解释 |
|--------------|--|
| PRIVATE_FILE | 输入认证文件所在目录 |
| USER | 创建物接入设备后返回的用户名，参见 创建物接入设备中的步骤3 |

| 参数名称 | 解释 |
|-----------|---|
| PWD | 创建身份后返回的密钥，参见 创建物接入身份 |
| publisher | 用来标识设备的ID，用户可自己定义，在同一个实例下，每个实体设备需要有一个唯一的ID，不同实体设备使用同一个client id建立连接会导致其它连接下线。client id只支持英文大小写字母，数字0-9，中划线和下划线，不支持其它字符。 |

```
#define PRIVATE_FILE "./root_cert.pem" //认证文件
#define USER "smart-sensor/ldw"
#define PWD "O7hrHKUYJLqxwajP/5/2fqdZ8KIDZ/aR4/CWrmRt6Gg="

int main(int argc, char* argv[]) {
    // argv[1] 为host:port
    // argv[2] 为topic
    // argv[3] 为message
    MQTTClient client;

    if (argc != 4) {
        PRINT("Usage: publish [host:port] [topic] [payload]\n");
        PRINT("Cleansession 1, Qos 1\n");
        return -1;
    }
    // 检测认证文件的可用性
    if (0 == access(PRIVATE_FILE, W_OK)) {
        // 设置认证文件
        setCertification(argv[1], PRIVATE_FILE);
    } else {
        PRINT ("Certification not exist, use normal connection.");
    }

    client = createPublisher(argv[1], "publisher", USER, PWD);
    if (NULL == client) {
        return -1;
    }
    PRINT("Start to publish message[%s] by topic : %s\n", argv[3], argv[2]);
    publishMsg(&client, argv[2], argv[3]);
    PRINT("=====\\n");

    // 关闭链接
    closeConnection(&client);

    return 0;
}
```

打开src/SubscriberSync.c，配置以下参数：

| 参数名称 | 解释 |
|--------------|--|
| PRIVATE_FILE | 输入认证文件所在目录 |
| USER | 创建物接入设备后返回的用户名，参见 创建物接入设备中的步骤3 |
| PWD | 创建身份后返回的密钥，参见 创建物接入身份 |

| 参数名称 | 解释 |
|----------|---|
| clientId | 用来标识设备的ID，用户可自己定义，在同一个实例下，每个实体设备需要有一个唯一的ID，不同实体设备使用同一个client id建立连接会导致其它连接下线。client id只支持英文大小写字母，数字0-9，中划线和下划线，不支持其它字符。 |

```

#define PRIVATE_FILE "root_cert.pem" //
#define USER "smart-sensor/ldw"
#define PWD "O7hrHKUYJLqxwajP/5/2fqdZ8KIDZ/aR4/CWrmRt6Gg="

int main(int argc, char* argv[]) {
    char buffer[1024];           // 1KB space
    int messageLen = 0;
    int ret;
    char *host = argv[1];
    char *topic = argv[2];
    char *clientId = "subscriber";
    MQTTClient client;
    if (argc != 3) {
        PRINT("Usage: Subscriber [host:port] [topic]\n");
        PRINT("Clean session 1, Qos 1\n");
        return -1;
    }
    // 设置认证文件
    setCertification(host, PRIVATE_FILE);
    PRINT("Start connect and subscribe on topic : %s\n", topic);
    client = createSubscriber(host, clientId, topic, USER, PWD);
    if (NULL == client) {
        return -1;
    }
    // sub消息
    while (TRUE) {
        ret = startSubscribe(&client, topic, buffer, &messageLen);
        if (0 == ret && messageLen > 0) {
            PRINT("Topic: %s ,receive Message: %s\n", topic, buffer);
        }
    }
}

```

打开Makefile文件:

编辑{\color{emcolor}\textbf{LIB_PATH}}和{\color{emcolor}\textbf{INCLUDE_PATH}}, 路径为MQTT -c文件的当前存储位置。

LIB_PATH = /home/iot/MQTT-c/lib

INCLUDE_PATH = /home/iot/MQTT-c/include

编辑后执行{\color{emcolor}\textbf{make all}}编译文件，生成“PublisherSync”和“SubscriberSync”文件。

运行“SubscriberSync”文件，参照命令格式Subscriber [host:port] [topic]，执行订阅操作：

```
{\color{emcolor}\textbf{./SubscriberSync ssl://yourendpointname.mqtt.iot.gz.baiduce.com:1884 topic}}
```

运行“PublisherSync”文件，参照命令格式[publish \[host:port\] \[topic\] \[payload\]](#)，执行发布操作：

```
{\color{emcolor}\textbf{./PublisherSync ssl://yourendpointname.mqtt.iot.gz.baiduce.com:1884 topic publishmessage}}
```

返回message，说明发布成功。

说明：

如果提示“libpaho-mqtt3cs.so.1”无法找到，请执行[export LD_LIBRARY_PATH=\\$LD_LIBRARY_PATH:/home/iot/MQTT-c/lib](#)。

13.3.2 C#代码示例

[前提条件：](#)

- 下载并安装virtual studio 2012。
- 下载[mqtt-net压缩包](#)，解压mqtt-net，即可使用vs2012运行。如果您需要自建工程，按照以下步骤操作：

1. 打开vs 2012，新建“项目>控制台应用程序”，选择.NET Framework 4.5版本。
2. 菜单栏选择“工具>库程序包管理器>程序包管理控制台”，执行[Install-Package M2Mqtt](#)命令，自动下载并安装mqtt文件。

说明：

如果无法下载mqtt文件，请打开vs2012“工具>选项>包管理器>程序包源”，添加一个本地目录作为程序包源，下载[m2mqtt.4.3.0.nupkg](#)文件，复制到刚才添加的本地目录中。当前项目右键选择“管理NuGet程序包”，找到您添加的程序包源，安装m2mqtt程序包即可。

3. 添加表头，

```
using uPLibrary.Networking.M2Mqtt;  
  
using uPLibrary.Networking.M2Mqtt.Messages;
```

4. 参考sample code的main函数，将创建好的实例参数输入：

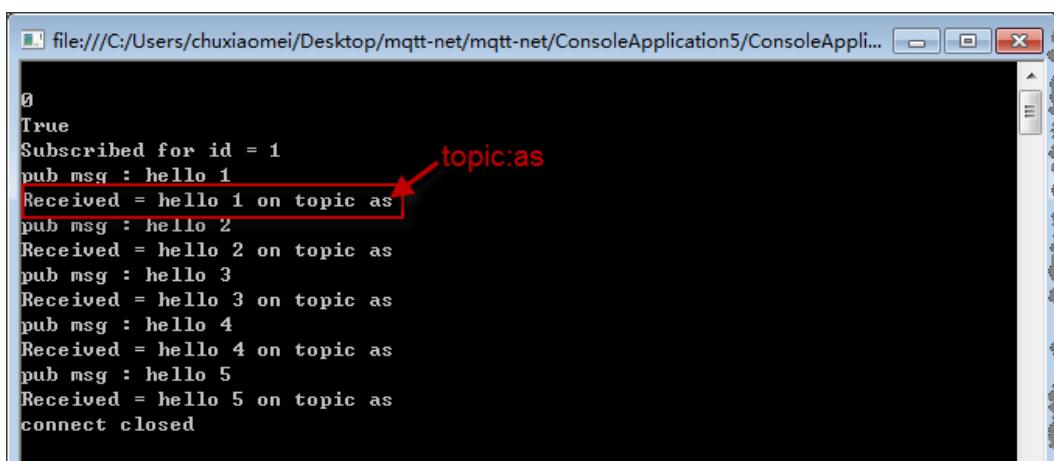
| 参数名称 | 解释 |
|----------|---|
| user | 创建物接入设备后返回的用户名，参见 创建物接入设备中的步骤3 |
| pwd | 创建身份后返回的密钥，参见 创建物接入身份 |
| endpoint | 实例地址，参见 创建物接入实例 |
| port | 实例的端口号。端口1883，不支持传输数据加密；端口1884，支持SSL/TLS加密传输。 |
| topic | 订阅的主题内容，参见 创建物接入策略 |
| clientid | 用来标识设备的ID，用户可自己定义，在同一个实例下，每个实体设备需要有一个唯一的ID，不同实体设备使用同一个client id建立连接会导致其它连接下线。client id只支持英文大小写字母，数字0-9，中划线和下划线，不支持其它字符。 |

```

string endpoint = "yourendpointname.mqtt.iot.gz.baidubce.com";
int port = 1884; // 端口默认1884
string user = "username"; // 创建thing，返回username
string pwd = "password"; // 创建principal，返回password
string clientid = Guid.NewGuid().ToString(); // 获取一个独一无二的id
string[] topic = new string[] { "yourtopic" }; // 输出订阅发布的主题
    
```

5. publish和subscribe代码参考mqtt-net代码示例。

6. 运行程序，成功订阅和发布消息，如下图所示：



13.3.3 python代码示例

下载TLS认证文件 物接入支持SSL/TLS加密传输方式，保障用户的数据传输安全。用户在执行示例代码前，需先下载TLS认证文件，并在代码中指定认证文件的存放路径。

下载[TLS认证文件](#)，并将认证文件保存至示例代码路径下。

说明：

安装Python 2.7以上版本，暂时不支持Python 3.5版本。

下载[mqtt-py压缩包](#)，解压mqtt-py，目录结构如下：

```
mqtt-py
├── sub_py2.py          //sub订阅代码示例
├── pub_py2.py          //pub发布代码示例
└── paho-mqtt-1.1.tar.gz //mqtt client端
```

以下操作步骤以Windows为例：

1. 安装Paho MQTT Python Client。打开cmd命令行，输入命令[pip install paho-mqtt](#)，自动下载并安装Python Client，如下图所示：

```
管理员: C:\windows\system32\cmd.exe
C:\Users\username>cd D:\download\Python27\Scripts
C:\Users\username>d:
D:\download\Python27\Scripts>pip2.7 install paho-mqtt
Collecting paho-mqtt
  Downloading paho-mqtt-1.1.tar.gz (41kB)
    49% :[██████████] 20kB 38kB/s eta 0:00:00
    58% :[██████████] 24kB 45kB/s eta 0:00:00
    68% :[██████████] 28kB 53kB/s eta 0:00:00
    78% :[██████████] 32kB 48kB/s eta 0:00:00
    88% :[██████████] 36kB 54kB/s eta 0:00:00
    98% :[██████████] 40kB/s eta 0:00:00
   100% :[██████████] 45kB/s eta 0:00:00
58kB/s
Installing collected packages: paho-mqtt
  Running setup.py install for paho-mqtt
Successfully installed paho-mqtt-1.1
You are using pip version 7.1.2, however version 8.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

D:\download\Python27\Scripts>python setup.py
'python' 不是内部或外部命令，也不是可运行的程序
或批处理文件。
```

用户也可以通过github下载Paho MQTT代码进行安装，具体操作如下：

```
git clone https://github.com/eclipse/paho.mqtt.python.git
cd org.eclipse.paho.mqtt.python.git
python setup.py install
```

关于Paho Python Client的详细介绍，可查看[Paho官方网站](<http://www.eclipse.org/paho/clients/python/>)。

2. 安装完成后，即可开始订阅消息。打开“sub_py2.py”文件，填写配置参数。

| 参数名称 | 解释 |
|-----------|---|
| trust | 输入认证文件所在目录 |
| user | 创建物接入设备后返回的用户名，参见 创建物接入设备中的步骤3 |
| pwd | 创建身份后返回的密钥，参见 创建物接入身份 |
| endpoint | 实例地址，参见 创建物接入实例 |
| port | 实例的端口号。端口1883，不支持传输数据加密；端口1884，支持SSL/TLS加密传输。 |
| topic | 订阅的主题内容，参见 创建物接入策略 |
| client_id | 用来标识设备的ID，用户可自己定义，在同一个实例下，每个实体设备需要有一个唯一的ID，不同实体设备使用同一个client id建立连接会导致其它连接下线。client id只支持英文大小写字母，数字0-9，中划线和下划线，不支持其它字符。 |

代码示例如下：

```
import paho.mqtt.client as mqtt

trust = "C:\\\\Users\\\\username\\\\Desktop\\\\iot\\\\mqtt-py\\\\root_cert.pem" #开启TLS时的认证文件目录
user = "01endpoint/01thing" #成功创建thing后返回的username
pwd = "07hrHKUYJLqjwajP/5/2fqdZ8KIDZ/aR4/CWrmRt6Gg=" #成功创建principal后返回的password
endpoint = "01endpoint.mqtt.iot.gz.baidubce.com" #实例(endpoint)地址
port = 1884 #endpoint端口
topic = "test iot service" #订阅的主题内容

def on_connect(client, userdata, flags, rc): #连接后返回0为成功
    print("Connected with result code "+str(rc))
    client.subscribe(topic, qos=1) #qos

def on_message(client, userdata, msg):
    print("topic:"+msg.topic+" Message:"+str(msg.payload))
```

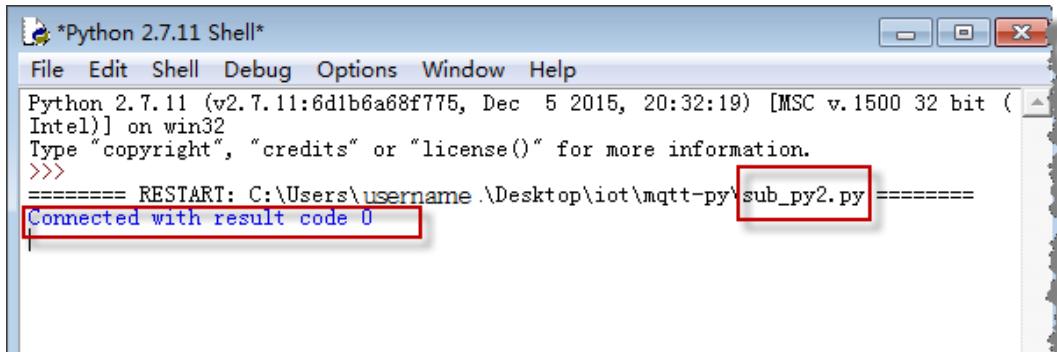
```

client = mqtt.Client(
    client_id="test_mqtt_receiver_1", #用来标识设备的ID，用户可自己定义，在同一个
实例下，每个实体设备需要有一个唯一的ID
    clean_session=True,
    userdata=None,
    protocol=mqtt.MQTTv3
)

client.tls_insecure_set(True) #检查hostname的cert认证
client.tls_set(trust) #设置认证文件
client.username_pw_set(user, pwd) #设置用户名，密码
client.on_connect = on_connect #连接后的操作
client.on_message = on_message #接受消息的操作
client.connect(endpoint, port, 60) #连接服务 keepalive=60
client.loop_forever()

```

配置相关参数后，执行上例代码，返回0说明已连接IoT Hub服务，并成功订阅主题。



3. 开始发布消息，打开“pub_py2.py”文件，填写配置参数。

| 参数名称 | 解释 |
|----------|---|
| trust | 输入认证文件所在目录 |
| user | 创建物接入设备后返回的用户名，参见 创建物接入设备 中的步骤3 |
| pwd | 创建身份后返回的密钥，参见 创建物接入身份 |
| endpoint | 实例地址，参见 创建物接入实例 |
| port | 实例的端口号。端口1883，不支持传输数据加密；端口1884，支持SSL/TLS加密传输。 |
| topic | 订阅的主题内容，参见 创建物接入策略 |

| 参数名称 | 解释 |
|-----------|---|
| client_id | 用来标识设备的ID，用户可自己定义，在同一个实例下，每个实体设备需要有一个唯一的ID，不同实体设备使用同一个client id建立连接会导致其它连接下线。client id只支持英文大小写字母，数字0-9，中划线和下划线，不支持其它字符。 |

代码示例如下：

```

import time
import paho.mqtt.client as mqtt
import datetime

def on_publish(msg, rc):    #成功发布消息的操作
    if rc == 0:
        print("publish success, msg = " + msg)

def on_connect(client, userdata, flags, rc):  #连接后的操作 0为成功
    print("Connection returned " + str(rc))

client = mqtt.Client(
    client_id="test_mqtt_sender_1", #用来标识设备的ID，用户可自己定义，在同一个实例下，每个实体设备需要有一个唯一的ID
    clean_session=True,
    userdata=None,
    protocol='MQTTv311'
)

trust = "C:\\\\Users\\\\username\\\\Desktop\\\\iot\\\\mqtt-py\\\\root_cert.pem" #开启TLS时的认证文件目录
user = "01endpoint/01thing"
pwd = "07hrHKUYJLqwxajP/5/2fqdZ8KIDZ/aR4/CWrmRt6Gg="
endpoint = "01endpoint.mqtt.iot.gz.baidubce.com"
port = 1884
topic = "test iot service"

client.tls_insecure_set(True) #检查hostname的cert认证
client.tls_set(trust) #设置认证文件
client.username_pw_set(user, pwd) #设置用户名，密码
client.connect(endpoint, port, 60) #连接服务 keepalive=60
client.on_connect = on_connect #连接后的操作
client.loop_start()
time.sleep(2)

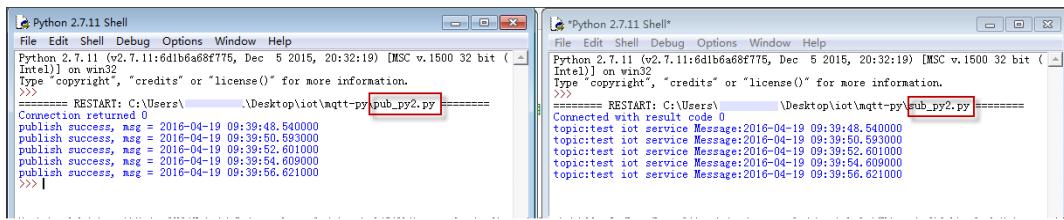
```

```

count = 0
while count < 5: #发布五条消息
    count = count + 1
    msg = str(datetime.datetime.now())
    rc, mid = client.publish(topic, payload=msg, qos=1) #qos
    on_publish(msg, rc)
    time.sleep(2)

```

执行上例代码，返回0说明已连接IoT Hub服务，pub文件成功发布主题信息，sub文件成功接收topic，如下图所示：



13.3.4 Java代码示例

Java代码示例请参看[使用MQTT Client SDK模拟实体设备](#)。

13.4 MQTT Client SDK

物接入与Paho（即MQTT Client SDK）完全兼容，如果开发者需要开发MQTT客户端，可到[Paho官方网站](#)下载SDK并获取帮助文档，详情如下：

| Client | MQTT 3.1 | MQTT 3.1.1 | LWT | SSL/TLS | Automated Re-connect | Offline Persistence | Message Broker API | WebSockets Support | MQTT Standard API | Blocking API | Non-Blocking API | High Availability |
|-----------------|----------|------------|-----|---------|----------------------|---------------------|--------------------|--------------------|-------------------|--------------|------------------|-------------------|
| Java | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Python | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ | ✗ |
| JavaScript | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ |
| GoLang | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ |
| C | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ |
| .Net (C#) | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ |
| Android Service | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ |

| Client | MQTT 3.1 | MQTT 3.1.1 | LWT | SSL/ TLS | Automated Re- connect | Offline Buffering | Message Persistence | WebSockets Support | MQTT Support | Standard API | Blocking API | Non- Blocking API | High Availability |
|-------------------------|-------------|---------------|-----|-------------|-----------------------------|----------------------|------------------------|-----------------------|-----------------|-----------------|-----------------|-------------------------|----------------------|
| Embedded C / C ++ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ | ✗ | ✗ |

13.5 相关参考

鉴权认证机制

本文档主要针对RESTful API调用者。

获取AK / SK

AK (Access Key ID) /SK (Secret Access Key)，主要用于对用户的调用行为进行鉴权和认证，相当于百度云API专用的用户名及密码。有关API认证的详细介绍，请参看[API认证机制](#)

证书管理

- 证书管理模块主要用于管理用户的SSL证书，方便用户录入、查看及应用SSL证书。
- 通过百度云[SSL证书服务](#)申请的证书将自动导入证书管理模块，无需用户手动操作。

区域选择说明

区域(Region)是指分布在全球范围内的各个物理节点，您可以根据客户群体分布的地理位置，选择在不同区域部署BCC。百度云目前提供服务区域为：华北区（北京）、华南区（广州）、华东区（苏州）。

第14章 下载专区

14.1 下载TLS认证文件

物接入支持SSL/TLS加密传输方式，保障用户的数据传输安全。用户在执行示例代码前，需先下载TLS认证文件，并在代码中指定认证文件的存放路径。下载[TLS认证文件](#)，并将认证文件保存至示例代码路径下。

14.2 下载IoT Edge SDK

为了帮助开发者能更高效地将各类设备与云端互联，并利用天工物联网平台完善的接入、存储、计算和分析能力打造物联网应用，天工物联网推出了完全开源开放的IoT Edge SDK。

IoT Edge SDK包含了物接入(IoT Hub)的C语言客户端、序列化和反序列化、设备管理、协议解析等功能组件，涵盖了实现设备上云时在断线缓存、在线检测、设备管理、数据安全传输等场景，同时更多的服务也将在未来开放。

如须下载SDK，或者了解更详细的IoT Edge SDK使用说明，请访问Github：<https://github.com/baidu/iot-edge-c-sdk>

14.3 下载示例开发版教程

ST和百度云天工基于ST新近推出的一款物联网开发套件B-L475E-IOT01A，提供了连接百度云天工的软件扩展包，内含连接到百度云天工物接入（IoT Hub），进行消息订阅和发布的应用程序。

用户基于本教程，使用ST的B-L475E-IOT01A可以快速上手搭建自己的百度云物联网应用程序。

如需下载，请访问Github：<https://github.com/baidu/iot-edge-sdk-samples>

14.4 下载MQTT相关

14.4.1 下载MQTT客户端代码示例

C代码示例：下载[MQTT-c压缩包](#)

C#代码示例：下载[mqtt-net压缩包](#)

python代码示例：下载[mqtt-py压缩包](#)

14.4.2 下载MQTT.fx客户端

源站：<http://www.jensd.de/apps/mqttx/1.3.1/>

国内：<http://mqttx.bceapp.com/>

14.4.3 下载MQTT Client SDK

物接入与Paho（即MQTT Client SDK）完全兼容，如果开发者需要开发MQTT客户端，可到[Paho官方网站](#)下载SDK并获取帮助文档，详情如下：

| Client | MQTT 3.1 | MQTT 3.1.1 | LWT | SSL/TLS | Automated Re-connect | Offline Buffering | Message Persistence | WebSockets Support | MQTT 5 Standard Support | Blocking API | Non-Blocking API | High Availability |
|-----------------|----------|------------|-----|---------|----------------------|-------------------|---------------------|--------------------|-------------------------|--------------|------------------|-------------------|
| Java | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| Python | √ | √ | √ | √ | √ | √ | × | √ | √ | √ | √ | × |
| JavaScript | √ | √ | √ | √ | √ | √ | √ | √ | × | × | √ | √ |
| GoLang | √ | √ | √ | √ | √ | √ | √ | √ | √ | × | √ | √ |
| C | √ | √ | √ | √ | √ | √ | √ | × | √ | √ | √ | √ |
| .Net (C#) | √ | √ | √ | √ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ |
| Android Service | √ | √ | √ | √ | √ | √ | √ | √ | √ | ✗ | √ | √ |
| Embedded C/C++ | ✗ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ | ✗ |

14.5 多种语言试用物接入

14.5.1 概述

说明：如果您还不了解MQTT协议，推荐您首先查看[MQTT协议介绍](#)，了解MQTT的工作原理。

本文档用于帮助用户试用物接入服务或快速完成物接入服务的部署，如果您想要了解更多功能，请参看[配置指南](#)和[快速入门](#)中多种方式试用物接入服务。

14.5.2 Python

从<http://iot-demo.cdn.bcebos.com/SampleCode/TestMQTTPython.zip>处下载工程文件，并解压至磁盘。

14.5.3 NodeJS

从<http://iot-demo.cdn.bcebos.com/SampleCode/TestMQTTNode.zip>处下载工程文件，并解压至磁盘。

14.5.4 Java

从<http://iot-demo.cdn.bcebos.com/SampleCode/TestMQTTJava.zip>处下载工程文件，并解压至磁盘。

您也可以通过导入Maven工程的方式测试消息收发，下载地址是<http://iot-demo.cdn.bcebos.com/SampleCode/TestMQTTJavaMaven.zip>

14.5.5 Arduino D1 & NodeMCU

从<http://iot-demo.cdn.bcebos.com/SampleCode/TestESP8266.zip>处下载代码，并且解压至本地磁盘。

14.5.6 Arduino Wido

从<http://iot-demo.cdn.bcebos.com/SampleCode/TestWido.zip>处下载代码，并且解压至本地磁盘。

14.5.7 Arduino Yun

从<http://iot-demo.cdn.bcebos.com/SampleCode/TestYun.zip>处下载代码，并且解压至本地磁盘。