ALIBABA CLOUD

阿里云

工业互联网平台
工业应用集成

文档版本：20220606

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.什么是工业应用集成

阿里云工业互联网企业级平台（数字化工厂）通过统一系统平台、统一门户入口、统一权限管理和统一的数据模型来集成制造企业从产品研发、生产、销售、物流到售后整个价值链过程中需要的所有应用。依据数字工厂的集成指南进行改造的工业应用，就能实现整合到数字工厂中，为企业用户提供业务服务。

集成开发

阿里云工业互联网企业级平台（数字化工厂）通过统一系统平台、统一门户入口、统一权限管理和统一的数据模型来集成制造企业从产品研发、生产、销售、物流到售后整个价值链过程中需要的所有应用。依据数字工厂的集成指南进行改造的工业应用，就能实现整合到数字工厂中，为企业用户提供业务服务。

集成到数字工厂的工业应用是指使用阿里云物联网应用托管服务，能够快速部署和分发的面向制造业务场景的多租户的应用，应用分类的定义请参看应用分类参考。完成物联网应用托管的工业应用，再根据集成工作概述中定义不同应用类型免登方式与数字工厂进行集成。

在集成工作概述中定义了应用类型以及介绍了不同的应用类型的免登方式，以及数据变更通知订阅的基本数据获取方式。

注：集成中所有的接口都需要获得授权，获取授权需要在应用托管的应用配置中进行权限声明，声明了权限声明后客户购买后将授权应用使用该接口，否则调用接口将返回请求被禁止的（错误代码：403）错误。

- **交互框架集成**：交互框架集成是指工业应用需要集成到数字工厂统一的门户中，需要通过完成以下几步，来实现统一交互：
  i. 通过功能页面集成，说明应用需要集成的功能。
  ii. 账号集成，应用能获得数字工厂的账号、角色和权限相关信息。
  iii. 日志集成，应用能够使用数字工厂的日志管理功能。
  iv. 消息集成，应用能够使用数字工厂的消息功能。

  为保证工业应用的交互统一，组件层要求使用阿里云提供的标准组件库（基于可视化企业级中后台UI的解决方案 Fusion Design）和相关的视觉规范。前端工具链和数据层建议使用Umijs，如果是Node应用，建议使用Egg。
- **基础数据集成**：工业应用能直接使用数字工厂提供的主数据、工厂模型、工艺路径和库存地点，实现多应用的标准统一的基础数据来进行业务服务数据的交互：
  i. 元数据定义，可以通过接口定义本应用需要的元数据。
  ii. 主数据集成，保证应用之间的主数据统一。
  iii. 工厂模型集成，保证应用能同步数字工厂的工厂模型。
  iv. 工艺路径集成，保证应用能同步数字工厂的工艺路径。
  v. 库存地点集成，保证应用能同步数字工厂的库存地点。
- **应用服务集成**：工业应用可以集成数字工厂提供的服务，最终为数字工厂用户提供完整的业务功能，服务包括以下几类：
  i. OEE，数字工厂企业用户开通并使用数字工厂的OEE后，第三方工业应用可以通过OEE提供的服务来同步生产设备的状态变化也可以获得设备综合效率的计算结果。
  ii. 生产过程追溯，数字工厂企业用户开通并使用数字工厂的生产过程追溯后，第三方工业应用可以通过生产过程追溯应用提供的服务来同步生产过程的产质耗数据。
  iii. 获取工厂日历数据，应用能获得用户在数字工厂的工厂日历中进行排班和计划停产的数据。
  iv. 获取经营数据，在获得企业用户授权后，应用能获取用户在经营驾驶舱中指标管理定义的经营指标数据以及资质管理中的营业执照信息、税务登记信息、营业办公信息、研发能力、质量管控能力、企业

自有品牌以及其它资质能力。

- **业务数据集成**：数字工厂提供以下标准经营分析功能，工业应用通过以下工业服务接口上报业务数据，用户可以直接通过经营驾驶舱查看对应经营分析结果：
  i. 通过接口上报生产成品入库和人员出勤，得到人均产量（能）分析的结果。
  ii. 通过接口上报生产报工和过程质检数据，得到成品率分析的结果。
  iii. 通过接口上报生产报工和过程质检数据，得到质量缺陷分析的结果。
  iv. 通过接口上报设备综合效率，得到设备效率分析的结果。
  v. 通过接口上报生产成品入库和能源数据，得到能耗分析的结果。
  vi. 通过接口上报人员出勤，得到人员出勤分析的结果。

应用需要授权才能获得调用数字工厂提供的工业服务权限，权限的申请查看应用授权。

在应用集成过程中需要使用工业云端服务，需要使用物联网平台服务的SDK，SDK支持以下语言：

- Java，SDK下载地址
- C#，SDK下载地址
- Nodejs，SDK下载地址
- Python2，SDK下载地址
- Python3，SDK下载地址

调试和验证

完成以上集成工作的开发后，可以申请数字工厂账号，然后通过应用部署来调试和验证集成后的效果；

通过数字工厂的应用监控可以查看并设置应用相关的指标情况，帮助工业应用集成商进行应用运行健康状态的监控和使用情况分析。要获取实时监控的情况需要通过数字工厂开放的SDK进行相关指标的开发，指标包括三种类型：

- 应用JVM的CPU、磁盘、内存、网卡吞吐、线程池、GC情况
- 前端页面访问情况
- 免登接口心跳

应用JVM指标

需要使用数字工厂开放的SDK，Git库代码http://gitlab.alibaba-inc.com/iotx-industry/iotx-industry-alimonitor-client

第一步，在启动类中加上 @ServletComponentScan(basePackages={"com.aliyun.iotx.industry.alimonitor.jmonitor.websuppert.items"})

```
/**
 * @ServletComponentScan 自动扫描我们SpringBoot项目内的有关Servlet配置，自动装配到我们的项目中。
 */
@ServletComponentScan(basePackages={"com.aliyun.iotx.industry.alimonitor.jmonitor.websupper
t.items"})
@SpringBootApplication
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

第二步，初始化ServletContext，并传递参数servletContext.setInitParameter：

参数说明

参数名称	是否必选	介绍
Jmonitor_OMP_AppKey	是	调用工业服务接口所使用的appKey
Jmonitor_OMP_AppSecret	是	调用工业服务接口所使用的appSecret
Jmonitor_OMP_Host	是	调用工业服务接口所使用的endpoint地址，线上环境是api.link.aliyun.com
Jmonitor_Collect_Period	否	配置采集周期，单位秒，默认是60秒，配置该参数需要大于60秒
Jmonitor_Instance_Id	否	对于实例分发类型的应用，需要传instanceId用来区别应用实例

示例如下：

```
package com.aliyun.iotx;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.web.servlet.ServletContextInitializer;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.env.Environment;
/**
 * @author zhengxiang.zzx
 * @version Id:WebContextConfiguration, v0.1 2020年06月23日 6:01 下午 zhengxiang.zzx Exp $
 */
@Configuration
public class WebContextConfiguration {
    @Value("${api.gateway.appkey:}")
    protected String appKey;
    @Value("${api.gateway.appsecret:}")
    protected String appSecret;
    @Value("${api.gateway.host:}")
    private String iotGatewayHost;
    @Value("${api.gateway.stage:}")
    private String iotGatewayStage;
    public final static String JMONITOR_OMP_APPKEY = "Jmonitor_OMP_AppKey";
    public final static String JMONITOR_OMP_APPSECRET = "Jmonitor_OMP_AppSecret";
    public final static String JMONITOR_OMP_HOST = "Jmonitor_OMP_Host";
    public final static String JMONITOR_OMP_STAGE = "Jmonitor_OMP_Stage";
    public final static String JMONITOR_COLLECT_PERIOD = "Jmonitor_Collect_Period";
    public final static String JMONITOR_INSTANCE_ID = "Jmonitor_Instance_Id";
    @Autowired
    private Environment env;
    @Bean
    public ServletContextInitializer servletContextInitializer() {
        return new ServletContextInitializer() {
            @Override
            public void onStartup(ServletContext servletContext) throws ServletException {
                // 优先使用环境变量传入的参数
                String key = env.getProperty("iot.hosting.appKey", appKey);
                String secret = env.getProperty("iot.hosting.appSecret", appSecret);
                String host = env.getProperty("iot.hosting.api.domain", iotGatewayHost);
                String stage = env.getProperty("api.gateway.stage", iotGatewayStage);
                servletContext.setInitParameter(JMONITOR_OMP_APPKEY, key);
                servletContext.setInitParameter(JMONITOR_OMP_APPSECRET, secret);
                servletContext.setInitParameter(JMONITOR_OMP_HOST, host);
                servletContext.setInitParameter(JMONITOR_OMP_STAGE, stage);
                // 采集周期 60 秒
                servletContext.setInitParameter(JMONITOR_COLLECT_PERIOD, "60");
                // test
                servletContext.setInitParameter(JMONITOR_INSTANCE_ID, "Rock-PC");
            }
        };
    }
}
```

第三步，设置依赖，在module目录新建一个libs目录，把iotx-industry-alimonitor-client-1.0.0.jar放入该目录在该module的pom.xml中增加如下dependency：

```
    <!-- 在dependencies中添加dependency -->
    <dependencies>
        <dependency>
            <groupId>com.aliyun.iotx</groupId>
            <artifactId>iotx-industry-alimonitor-client</artifactId>
            <version>1.0.0</version>
            <scope>system</scope>
            <systemPath>${pom.basedir}/libs/iotx-industry-alimonitor-client-1.0.0.jar</syst
emPath>
        </dependency>
    </dependencies>
```

```
<!-- https://mvnrepository.com/artifact/com.alibaba/fastjson -->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.71</version>
</dependency>
<dependency>
    <groupId>com.aliyun.api.gateway</groupId>
    <artifactId>sdk-core-java</artifactId>
    <version>1.1.0</version>
</dependency>
<dependency>
    <groupId>com.aliyun.iotx</groupId>
    <artifactId>iotx-api-gateway-client</artifactId>
    <version>1.0.3</version>
</dependency>
```

前端页面指标

在前端代码用引入数字工厂前端监控js文件，引入后可以有两种方式上报并监控前端页面访问量：

- 针对SPA页面，支持自动监听上报数据，页面切换即时上报
- 主动上报接口，用户可以自定义上报数据

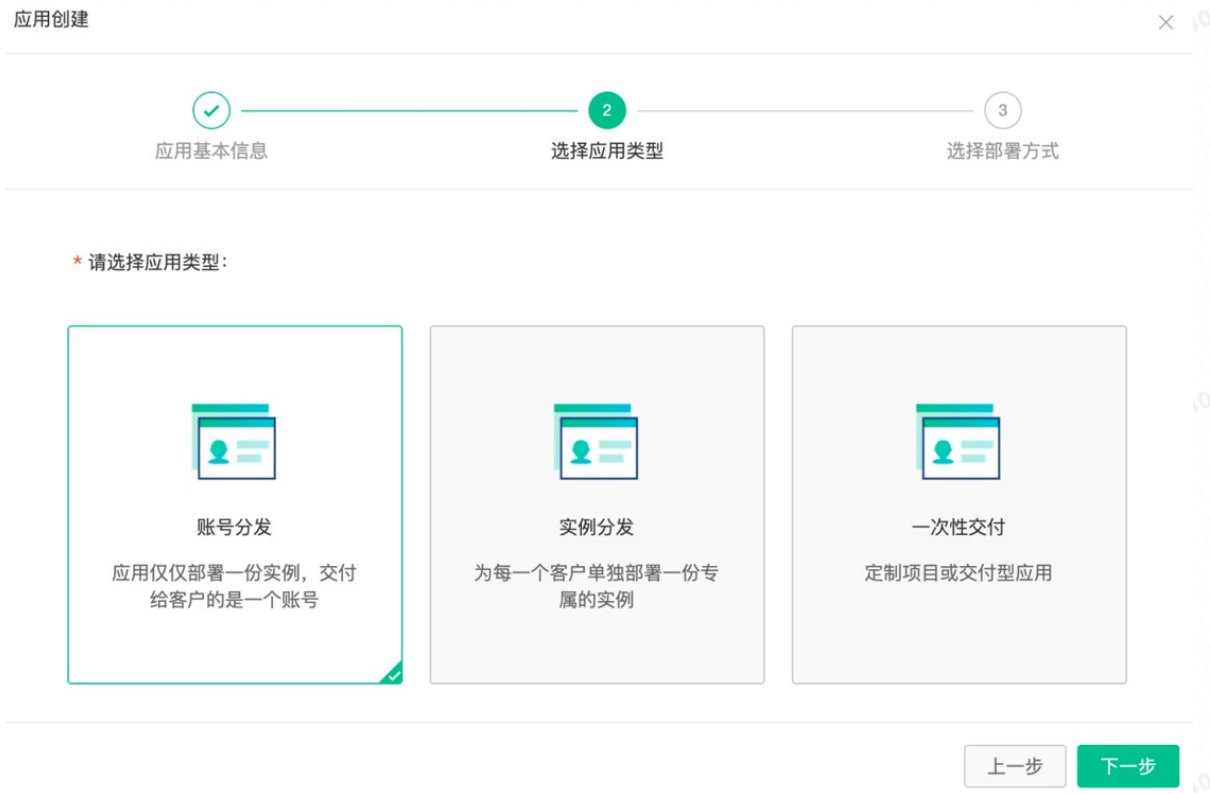
代码示例：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <script src="https://g.alicdn.com/iotx-industry-fe/iotx-fe-logger/1.0.0/index.aio.min.j
s"></script>
    <script>
        // 引入Logger
          const Logger = window['iotx-fe-logger'].default;
                // 初始化Logger
        const log = new Logger({
          autoSendPv: true, // 是否自动上报
          appType: 'THIRD_APP',
          appKey: '25219625', //三方应用appKey
        });
        // 调用主动上报接口
        log.api({
          // appconfig 定义每个页面的pageId
          pageId: '/dashboard',
        });
    </script>
</body>
</html>
```

2.集成工作概述

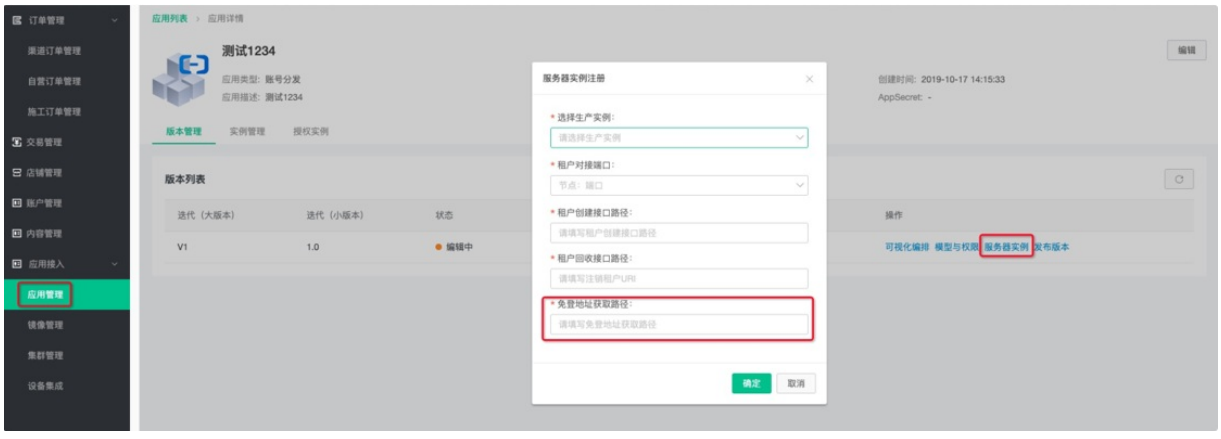
应用类型及免登方式

在物联市场的卖家后台，可以创建不同类型的应用



- 实例（分发）型：也称为单租户应用、独占式应用。ISV定义应用之后，每次交付行为都是以这份定义为基础，平台为客户分配一份独立的云资源，并部署一份应用到该云资源中，并将访问节点信息交付给客户。
- 账号（分发）型：也称为多租户应用、共享式应用，或者SaaS应用。每次交付行为，会导致平台向该SaaS应用发起一个开通新租户的请求，并将访问入口交付给客户。
- 一次性交付型：ISV定义的一份应用，本身就是一次交付。所以，不存在分发给第二个客户的行为。如果有第二个客户需要相同的能力，ISV需要再次定义一个应用。

1、SSO免登



对于账号分发模式的应用，需要应用实现三个api，分别是：

租户创建接口：用户购买应用时，由托管平台调用这个接口，通知应用有新的购买行为，参数里面会有购买者的信息

租户回收接口：用户删除应用时，由托管平台调用这个接口，通知应用有用户删除应用

免登地址获取：这个接口即是对接SSO的api，用于获取免登的地址。接口参数如下：

参数	类型	必填	描述
id	String	是	该次访问唯一标示符（幂等验证ID）
tenantId	String	是	IoT平台标识一个租户的唯一ID
tenantSubUserId	String	否	IoT平台租户组织架构中的员工唯一ID，当员工账号免登时填写
appId	String	是	应用唯一ID，一个租户可以重复购买一款软件，每次购买appId都不同
userId	String	是	SaaS标识一个租户的唯一ID

tenantSubUserId对应员工的标识，appId是应用实例的标识

具体可参考SaaS应用对接。

2、OAuth免登

这种免登方式，用户通过数字工厂访问应用页面时，会在原页面路径后面加上code，类似于

https://a.b.c/page/index.html?code=XXXXXX

应用再通过open api从code中获取当前登录者的信息。具体可以参考OAuth2.0对接。

用于应用对接的开放api中，像主数据、工厂建模、工艺路径的api中，会需要appId和employeeId两个参数：

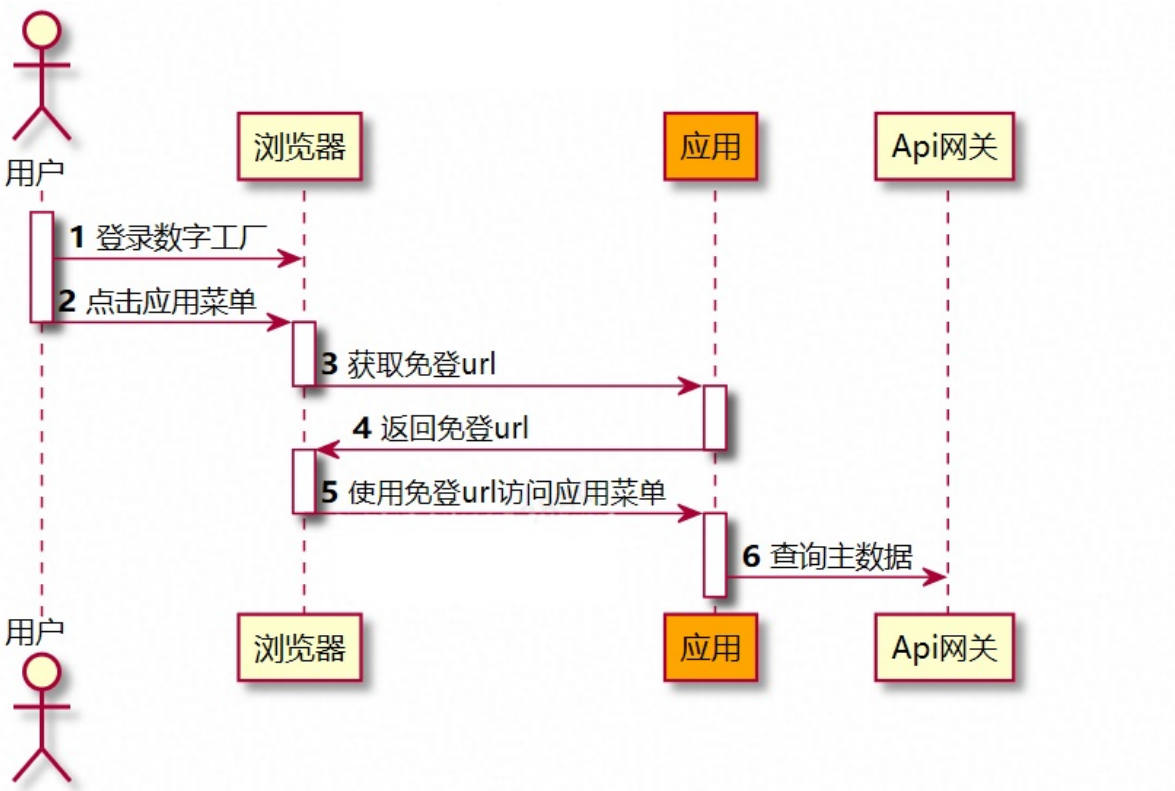
（1）参数appId来确定当前调用相关的租户。

appId是客户在市场上购买应用时生成的一个实例id，用于标识应用实例，同时也与购买者的租户信息对应。对于多租户的SaaS应用，appKey和appId对应的目标租户可能是不一样的，因此需要appId确认目标租户信息；同时为了兼容老版本，当appId没有指定时，使用appKey所属的租户信息。

（2）参数employeeId用于操作鉴权。

employeeId是阿里云的子账号id。每一种主数据在系统内部都被当做一个资源，并且给不同的子账号授予不同的操作权限，employeeId标识当前api调用者的身份。用户登录数字工厂之后，可能会通过集成进来的应用页面查询主数据，这种场景下会有employeeId的信息。如果没有登录态，例如后台的定时任务，这个参数可不传，认为是主账号操作。

这个过程大致如下



注：

① 图中第3、4步是对接SSO方式才有的步骤，会传一个tenantSubUserId给应用，这个值可以做为employeeId调用open api

② 对于oauth的方式，在第5步中，应用收到请求后，根据code换取当前登录者的信息，会拿到一个open_id，这个值可以做为employeeId调用open api

数据变更通知订阅

应用可以通过订阅消息的方式来接收数据记录变化的通知。

数字工厂采用http2的方式推送消息，使用QoS1，每条消息只能被一个客户端消费并且只能被消费一次。

举例说明如下：

① 有一个appkey注册了消息通知，多个程序使用同一个appkey进行消息接收，当有数据变化时产生一条消息，只有一个程序能收到这条消息

② 有多个不同的appkey并且都注册了消息通知，当有数据变化时会对每一个appkey产生内容同的一条消息，每个appkey都会收到单独的一条消息通知

可查看其它的服务端订阅使用限制。

1、订阅消息

通过开放api订阅消息，http2方式数据推送，注册订阅的消息类型

独享实例托管场景

```
package demo.masterdata;
import com.alibaba.cloudapi.sdk.model.ApiResponse;
import com.alibaba.fastjson.JSON;
import com.aliyun.iotx.api.client.IoTApiClientBuilderParams;
import com.aliyun.iotx.api.client.IoTApiRequest;
import com.aliyun.iotx.api.client.SyncApiClient;
import com.google.common.collect.Lists;
import com.google.common.collect.Maps;
import java.io.UnsupportedEncodingException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
/**
 * 订阅消息通知的demo，使用open api
 */
public class SubscribeDemoApplication {
    public static void main(String[] args) {
        try {
            IoTApiClientBuilderParams ioTApiClientBuilderParams = new IoTApiClientBuilderPa
rams();
            ioTApiClientBuilderParams.setAppKey("");
            ioTApiClientBuilderParams.setAppSecret("");
            SyncApiClient syncApiClient = new SyncApiClient(ioTApiClientBuilderParams);
            IoTApiRequest request = new IoTApiRequest();
            //设置api的版本
            request.setApiVer("1.0.0");
            List<Integer> msgTypes = Lists.newArrayList();
            // 主数据
            msgTypes.add(1);
            // 工厂建模
            msgTypes.add(4);
            //设置参数
            request.putParam("msgTypes", msgTypes);
            // 多租户SaaS应用的appId，新用户购买后该SaaS应用后，就要重新订阅消息
            // request.putParam("appId", "xxxxx");
            Map<String, String> properties = Maps.newHashMap();
            properties.put("http_method_type", "POST");
            request.putParam("properties", JSON.toJSONString(properties));
            // 根据需要设置http header
            Map<String, String> headers = new HashMap<>();
            //请求参数域名、path、request
            String path = "/industry/notification/http2/register";
            ApiResponse response = syncApiClient.postBody("api.link.aliyun.com", path, requ
est, true, headers);
            String responseString = new String(response.getBody(), "UTF-8");
            System.out.println("response code = " + response.getCode() + " response = " + r
esponseString);
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }
```

多租户SaaS场景

对于多租户SaaS应用，每次有用户购买时都需要用appId订阅一次消息。这个appId是调用生产租户接口时传入的

```
package demo.masterdata;
import com.alibaba.cloudapi.sdk.model.ApiResponse;
import com.alibaba.fastjson.JSON;
import com.aliyun.iotx.api.client.IoTApiClientBuilderParams;
import com.aliyun.iotx.api.client.IoTApiRequest;
import com.aliyun.iotx.api.client.SyncApiClient;
import com.google.common.collect.Lists;
import com.google.common.collect.Maps;
import java.io.UnsupportedEncodingException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
/**
 * 订阅消息通知的demo，使用open api
 */
public class SubscribeDemoApplication {
    public static void main(String[] args) {
        try {
            IoTApiClientBuilderParams ioTApiClientBuilderParams = new IoTApiClientBuilderPa
rams();
            ioTApiClientBuilderParams.setAppKey("");
            ioTApiClientBuilderParams.setAppSecret("");
            SyncApiClient syncApiClient = new SyncApiClient(ioTApiClientBuilderParams);
            IoTApiRequest request = new IoTApiRequest();
            //设置api的版本
            request.setApiVer("1.0.0");
            List<Integer> msgTypes = Lists.newArrayList();
            // 主数据
            msgTypes.add(1);
            // 工厂建模
            msgTypes.add(4);
            //设置参数
            request.putParam("msgTypes", msgTypes);
            // 多租户SaaS应用的appId，新用户购买后该SaaS应用后，就要重新订阅消息
            request.putParam("appId", "xxxxx");
            Map<String, String> properties = Maps.newHashMap();
            properties.put("http_method_type", "POST");
            request.putParam("properties", JSON.toJSONString(properties));
            // 根据需要设置http header
            Map<String, String> headers = new HashMap<>();
            //请求参数域名、path、request
            String path = "/industry/notification/http2/register";
            ApiResponse response = syncApiClient.postBody("api.link.aliyun.com", path, requ
est, true, headers);
            String responseString = new String(response.getBody(), "UTF-8");
            System.out.println("response code = " + response.getCode() + " response = " + r
esponseString);
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }
}
```

2、接收消息

pom依赖。

```
<dependency>
  <groupId>com.aliyun.api.gateway</groupId>
  <artifactId>sdk-core-java</artifactId>
  <version>1.1.0</version>
  <exclusions>
    <exclusion>
      <groupId>commons-codec</groupId>
      <artifactId>commons-codec</artifactId>
    </exclusion>
  </exclusions>
</dependency>
<dependency>
  <groupId>com.aliyun.openservices</groupId>
  <artifactId>iot-client-message</artifactId>
  <version>1.1.5</version>
</dependency>
```

java demo：

```
public static void main(String[] args) {
        String endPoint = "https://${appKey}.iot-as-http2.cn-shanghai.aliyuncs.com:443";
        // 托管应用appKey
        String appKey = "";
        // 托管应用appSecret
        String appSecret = "";
        // 连接配置
        Profile profile = Profile.getAppKeyProfile(endPoint, appKey, appSecret);
        // 构造客户端
        MessageClient client = MessageClientFactory.messageClient(profile);
        // 数据接收
        client.connect(messageToken -> {
            Message m = messageToken.getMessage();
            System.out.println("receive message from " + m);
            return MessageCallback.Action.CommitSuccess;
        });
    }
```

3.交互框架集成

3.1. 功能页面集成

第三方应用需要实现一个应用自描述的接口，它是将三方应用集成到工业互联网企业级平台（数字工厂）的桥梁。企业用户在物联网市场中订购应用以后，通过“工业互联网企业级平台（数字工厂）”或“集成工作台”的“应用集成”功能加载应用自描述接口内容，对应用进行集成。本文中的“应用描述接口”也称做“应用配置”。

应用集成效果

完成应用集成后，应用的功能菜单将会融入到“工业互联网企业级平台（数字工厂）”的左侧导航栏。例如“A生产管理应用”这个集成后如下图：

提供接口方式

- 协议

protocal	httpMethod
HTTPS	GET

- 服务的url

注意：应用描述接口的访问路径建议使用应用相对路径/appconfig，这样能够实现自动集成到数字工厂，如果采用其他的访问路径，需要应用安装和配置在数字工厂中应用集成中进行手动集成。

由域名和服务路径拼接成服务的URL，例如应用的域名为https://****.com

服务URL默认https://****.com/appconfig?appId=xxxx，appId为标识哪个租户调用该应用描述接口，可以根据不同的租户提供不同的配置。

- 返回参数

参数	类型	描述
code	Integer	调用成功返回200；若调用失败返回203

参数	类型	描述
message	String	code返回203时填写具体失败原因，code返回200填写success
data	JSON	应用描述

- 返回参数示例


```
{
  "code": 200,
  "message": "success",
  "data": {
    "appName": "A生产管理应用",
    "urlprefix": "https://abc.com",
    "appConfigPath": "/appconfig",
    "appDomain": "生产制造",
    "features": [
      {
        "name": "生产管理",
        "resType": "directory",
        "modules": [
          {
            "name": "查看生产计划",
            "resType": "page",
            "resId": "queryAllWorkorders",
            "path": "/queryAllWorkorders",
            "layoutMode": "workspace"
          },
          {
            "name": "确认生产计划",
            "resType": "page",
            "resId": "confirmWorkorder",
            "path": "/confirmWorkorder",
            "layoutMode": "workspace",
            "authorities": [
              {
                "name": "终止生产计划",
                "resType": "authority",
                "resId": "terminateOrder",
                "authorities": [
                  {
                    "name": "终止生产计划二次确认",
                    "resType": "authority",
                    "resId": "terminateOrderConfirm"
                  }
                ]
              }
            ]
          },
          {
            "name": "查看生产计划执行结果",
            "resType": "page",
            "resId": "queryWorkorderResults",
```

```
            "path": "/queryWorkorderResults",
            "layoutMode": "workspace",
            "authorities": [
              {
                "name": "查看已下发生产计划",
                "resType": "authority",
                "resId": "queryConfirmedOrder",
                "authorities": [
                  {
                    "name": "查看已终止生产计划",
                    "resType": "authority",
                    "resId": "queryTerminatedOrder"
                  }
                ]
              }
            ]
          }
        ]
      },
      {
        "name": "质量管理",
        "resType": "directory",
        "modules": [
          {
            "name": "查看质检结果",
            "resType": "page",
            "resId": "queryQCResults",
            "path": "/queryQCResults",
            "layoutMode": "workspace"
          }
        ]
      },
      {
        "name": "应用管理权限",
        "resType": "authority",
        "resId": "appadmin"
      }
    ],
    "specialpages": [
      {
        "type": "configuration",
        "path": "/config"
      },
      {
        "type": "help",
        "path": "/help"
      },
      {
        "type": "description",
        "path": "/description"
      }
    ],
    "dependencies": {
    "masterData": [
```

```
      {
        "name": "物料",
        "properties": [
          {
            "propertyCode": "code",
            "propertyDesc": "物料编码",
            "propertyType": "STRING",
            "propertyLimit": {
              "min": 1.0,
              "max": 100,
              "len": 64,
              "factoryType": "FACTORY",
              "technologyType": "TECHNOLOGY",
              "warehouseType": "WAREHOUSE",
              "enumValues": [
                {
                  "value": "0",
                  "remark": "早班"
                },
                {
                  "value": "1",
                  "remark": "中班"
                },
                {
                  "value": "2",
                  "remark": "晚班"
                }
              ],
              "booleanValues": [
                {
                  "value": "1",
                  "remark": "真"
                },
                {
                  "value": "0",
                  "remark": "假"
                }
              ]
            },
            "isUnique": 1,
            "isNull": 1,
            "defaultValue": "xxx"
          }
        ]
      }
    ]
  }
  }
}
```

描述接口格式

工业应用接入阿里云工业互联网平台，首先实现应用托管要求的接口，例如对共享式SaaS应用，需要实现免密登录（GetSSOUrl）、生产租户（CreateInstance）、注销租户（DeleteInstance）等服务，详细要求参考SaaS应用对接。

然后对于需要实现一个获取应用描述接口的服务，说明应用的基本信息。应用配置包括“基础信息”、“功能配置”、“特殊页面”、“资源依赖”这几个部分，分别说明如下：

1. **基础信息**

   描述工业应用的基础信息：

参数名称	参数说明	参数示例
appName	描述工业应用名称，不超过64个字符	生产管理
urlprefix	访问该应用的路径，不超过800字符	https://***.com
appConfigPath	访问该工业应用的描述接口的相对路径，如果是多租户SaaS将以urlprefix为路径前缀，如果是独占式应用以应用实例部署后的访问路径为前缀，不超过800字符	/appconfig
appDomain	说明应用属于哪一个域如果没有说明，默认为生产制造域，可选择：供应链管理|研发设计|生产制造|运营管理|仓储物流|运维服务	供应链管理

2. **功能配置**

   功能配置列出该应用哪些页面和权限控制点需要集成到数字工厂中，集成的功能分成三种类型：

   i. 目录（resType=directory）：显示在数字工厂该应用域的导航菜单中，对页面进行分组用，目录下级可以继续声明目录和页面类型的配置；

   ii. 页面（resType=page）：页面作为目录的下级菜单，访问路径为整合到门户以后，数字中心访问该应用页面的相对路径，页面下级可以继续声明权限类型的配置；

   iii. 权限（resType=authority）：鉴权资源列表列出该应用需要进行权限控制的功能点，鉴权资源可以是页面级也可以是按钮级甚至是数据级，页面下级可以继续声明权限类型的配置。

   目录的配置详细说明：

参数名称	参数说明	参数示例
name	目录名称	生产管理
modules	下级功能，包括目录、页面和权限	功能的JSON

   JSON表达方式为：

```
{
    "name": "生产管理",
    "resType": "directory",
    "modules": [
          {
            "name": "查看生产计划",
            "resType": "page",
            "resId": "queryAllWorkorders",
            "path": "/queryAllWorkorders",
            "layoutMode": "workspace"
          }
    ]
}
```

页面的配置详细说明：

参数名称	参数说明	参数示例
name	页面集成后显示的菜单名称和权限名称	确认生产计划
resId	resId作为一个整体名字空间，命名应该唯一，建议长度在32个字符内，最大不能超过40个字符	confirmWorkorder
path	页面访问路径，不能超过800字符，**页面整合到导航中必须实现https而不是http协议**	/confirmWorkorder
layoutMode	页面显示方式，可选择：workspace|fullscreen。workspace显示区域为工作区，左边应用导航菜单不隐藏，该模式为默认模式；fullscreen，设置该页面显示区域为全屏，左边应用导航菜单自动隐藏，该模式适合只集成单个控制台页面	workspace
authorities	属于该页面的子权限	授权的JSON

JSON表达方式为：

```
{
            "name": "确认生产计划",
            "resType": "page",
            "resId": "confirmWorkorder",
            "path": "/confirmWorkorder",
            "layoutMode": "workspace",
            "authorities": [
              {
                "name": "终止生产计划",
                "resType": "authority",
                "resId": "terminateOrder",
                "authorities": [
                  {
                    "name": "终止生产计划二次确认",
                    "resType": "authority",
                    "resId": "terminateOrderConfirm"
                  }
                ]
              }
}
```

权限的配置详细说明：

参数名称	参数说明	参数示例
name	权限的名称	终止生产计划
resId	resId作为一个整体名字空间，命名应该唯一，建议长度在32个字符内，最大不能超过40个字符	terminateOrder
authorities	属于该权限的子权限	授权的JSON

JSON表达方式为：

```
{
            {
              "name": "终止生产计划",
              "resType": "authority",
              "resId": "terminateOrder",
              "authorities": [
                {
                  "name": "终止生产计划二次确认",
                  "resType": "authority",
                  "resId": "terminateOrderConfirm"
                }
              ]
            }
}
```

3. **特殊页面**

为提高应用的易用性，应用可以通过描述接口来发布特殊功能的页面，现在描述接口支持三类特殊功能页面：应用配置页面（类型：CONFIGURATION）；应用帮助页面（类型：HELP）；应用详情介绍（类型：DESCRIPTION）

应用配置页面将出现在数字工厂中的应用配置页面：

应用帮助页面将出现在数字工厂中的帮助页面和应用详情介绍。如果没有相应的页面配置，前端将不会显示，例如下面这个配置会显示“详情”和“设置”，不会显示“帮助”。

```
"specialpages": [
        {
            "type": "configuration",
            "path": "/config"
        },
        {
            "type": "description",
            "path": "/description"
        }
    ]
```

4. **资源依赖**

dependencies部分用于描述应用对资源的依赖，其中masterData用来指定应用对元数据的依赖，如果数字工厂开通应用后中没有定义元数据则会自动创建。将不能在编辑元数据修改或者删除被依赖的元数据定义用户。

元数据定义包括了名称和属性定义，JSON结构示例如下：

```
"masterData": [{
    "name": "物料", // 主数据名称，必填
    // 元数据的属性列表
    "properties": [{
        "propertyCode": "code", // 属性标识，必填
        "propertyDesc": "物料编码",
        "propertyType": "STRING" / "INTEGER" / "DOUBLE" / "ENUM" / "FACTORY" / "TECHNOL
OGY" / "WAREHOUSE", // 属性类型，必填
        // 属性的限制描述
        "propertyLimit": {
            // 属性类型为INTEGER/DOUBLE时，这两个字段有效，值为浮点数
            "min": 1.0,
            "max": 100,
            // 属性类型为STRING时，这个限制字符串的长度
            "len": 64,
            // 属性类型为工厂模型时，这个字段表示具体的工厂节点类型
            "factoryType": "FACTORY" / "WORKSHOP" / "BELTLINE" / "MACHINING_CENTER",
            // 属性类型为工艺路径时，这个字段表示具体的工艺路径的节点类型
            "technologyType": "TECHNOLOGY" / "PROCESS" / "STEP",
            // 属性类型为库存地点时，这个字段表示具体的库存节点类型
            "warehouseType": "WAREHOUSE" / "AREA" / "LOCATION",
            // 属性类型为ENUM时，这个记录枚举的所有值
            "enumValues": [{
                "value": "0",
                "remark": "早班"
            }, {
                "value": "1",
                "remark": "中班"
            }, {
                "value": "2",
                "remark": "晚班"
            }],
            // 属性类型为BOOLEAN时，这个记录布尔值
            "booleanValues": [{
                "value": "1",
                "remark": "真"
            }, {
                "value": "0",
                "remark": "假"
            }]
        },
        "isUnique": 1 / 0, // 是否唯一键
        "isNull": 1 / 0, // 是否可空
        "defaultValue": "xxx",  // 默认值，不管属性是什么类型，这里都用字符串
    }]
}]
```

访问应用页面

集成到“工业互联网企业级平台（数字工厂）”后，可以通过左侧导航栏访问应用的页面。

1、共享式应用的页面

共享式应用按ssoURL方式实现免密登录，url分2个部分：

- 域名及登录验证信息
- 页面的url路径

例如：https://****.com/user/login?token=xxxx&oauth_callback=https://****.com/module/page

其中，

“https://****.com/user/login?token=xxxx”是通过免密登录URI返回的免密url,

“https://****.com/module/page”是appconfig配置中urlPrefix与path拼接成的资源的访问路径。

应用里面需要解析“oauth_callback”关键字进行跳转。

2、独占式应用的页面

独占式应用按Oauth2.0实现免密登录，url分为2个部分：

- 域名及登录验证信息
- 页面url路径

例如：https://****.com/module/page?code=xxxxxx

其中，

https://****.com/module/page是页面的路径；

code=xxxxxx是登录验证信息。

应用资源鉴权

应用可以自定义权限，限制不同角色的用户对应用的访问。例如应用在查看生产计划的页面中编写一段脚本，根据当前登录的账号是否有“终止生产计划”的权限，通过这个判断”终止生产计划“这个按钮是否disable。可以分3个步骤：

- 在应用描述接口中定义鉴权资源，例如把“终止生产计划”作为一个鉴权资源；
- 通过“工业互联网企业级平台（数字工厂）”或“集成工作台”的“角色管理”功能，把这个资源授权给角色；
- 在应用里面，调用资源鉴权API，查询用户的访问权限，并做访问控制。

1、平台上资源授权

完成应用集成后，应用描述接口中的“鉴权资源”将会显示在“角色管理”的“访问权限”列表中。

通过“工业互联网企业级平台（数字工厂）”或“集成工作台”的“角色管理”功能，将应用定义的资源授权给相应的角色。

2、应用内资源鉴权

资源鉴权API

name	path	version	description
authenAppResPermission	/industry/app/res/permission/authenticate	1.0.1	应用鉴权资源访问鉴权

请求参数

名称	类型	必要	描述	示例
resCode	String	是	应用鉴权资源码，来自应用配置中的resCode	"queryAllOrder"
appId	String	是	应用ID，应用唯一标识	"12312312"
employeeId	String	是	员工ID，可通过oAuth2的getEmployeeInfoByAccessToken接口获取	"cxvsdfwefs"

返回参数

名称	类型	描述
id	String	请求序号
code	Integer	调用返回码，200：成功，其他：失败，返回失败可通过message获取具体错误信息。
message	String	英文错误信息
localizedMsg	String	本地语言错误信息
data	Boolean	返回数据，只有code为200时，data才有效，true：有权限；false：无权限

正常返回示例

```
{
  "id": "70333b89-3302-4006-8559-cf6d345ae52c",
  "code": 200,
  "message": "success",
  "localizedMsg": null,
  "data": true/false
}
```

这个API相关的说明也可以参考：

为工业业务提供具体业务所使用的各类资源的定义功能，可以针对需要进行访问管控的资源进行不同级别的定义

第三方页面打开数字工厂首页tab的方法

1、CDN方式

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>三方页面通讯demo</title>
  <script src="https://iotx-components.oss-cn-beijing.aliyuncs.com/components-utils/1.0.0/message_client.js"></script>
</head>
<body>
  <button onclick="openTab()">打开链接</button>
  <script>
    const client = new IotxMessageClient.default();
    client.init();
    const openTab = () => {
      const prefix = window.location.protocol + "//" + window.location.host;
      client.sendMsg('THIRD_DOMAIN', { url: prefix + '/importorder' }, ({ allow }) => {
        if (allow) {
          console.log('成功打开tab')
        } else {
          // 可以做一些失败逻辑，如window.open
          console.log('false')
        }
      });
    }
  </script>
</body>
</html>
```

2、npm方式

- 安装sdk

```
npm install --save @iotx/components-utils
```

- 初始化sdk

```
import { MessageClient } from '@iotx/components-utils';
this.client = new MessageClient();
this.client.init();
```

- 打开url，url为已集成数字工厂应用下的url，示例如下：

```
// 打开指定页面，已指定页面的路径为/importorder为例
const prefix = window.location.protocol + "//" + window.location.host;
this.client.sendMsg('THIRD_DOMAIN', {
  url: prefix + '/importorder',
}, ({ allow }) => {
  // allow  为true时，表示访问白名单内，支持直接打开，否则不支持打开
  // 其他业务逻辑
})
```

3.2. 账号集成

数字工厂中系统管理员能通过设置中心的权限管理为企业的数字工厂管理登录账号，定义和授予用户角色权限。数字工厂账号和权限集成详细请查看SaaS应用租户对接

查询

获取全部角色列表，接口说明

名称	路径	描述
listAllRole	/industry/user/role/all/list	获取全部角色列表

java demo

```
import com.alibaba.cloudapi.sdk.model.ApiResponse;
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONArray;
import com.alibaba.fastjson.JSONObject;
import com.aliyun.iotx.api.client.IoTApiClientBuilderParams;
import com.aliyun.iotx.api.client.IoTApiRequest;
import com.aliyun.iotx.api.client.SyncApiClient;
import org.apache.commons.collections4.CollectionUtils;
import java.io.UnsupportedEncodingException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.UUID;
import java.util.stream.Collectors;
public class DemoApplication {
    private static String postBody(String path, IoTApiRequest request, Map<String, String>
headers) {
        String appKey = "";
        String appSecret = "";
        IoTApiClientBuilderParams ioTApiClientBuilderParams = new IoTApiClientBuilderParams
();
        ioTApiClientBuilderParams.setAppKey(appKey);
        ioTApiClientBuilderParams.setAppSecret(appSecret);
        SyncApiClient syncApiClient = new SyncApiClient(ioTApiClientBuilderParams);
        try {
            //设置请求参数域名、path、request ,如果使用HTTPS，设置为true
            ApiResponse response = syncApiClient.postBody("api.link.aliyun.com", path, requ
est, true, headers);
            String responseString = new String(response.getBody(), "UTF-8");
            System.out.println("response code = " + response.getCode() + " response = " + r
esponseString);
            return responseString;
        } catch (UnsupportedEncodingException uee) {
            System.out.println(uee.getMessage());
            uee.printStackTrace();
            return null;
        }
    }
    public static void main(String[] args) {
        IoTApiRequest request = new IoTApiRequest();
        //设置协议版本号
        request.setVersion("1.0");
        String uuid = UUID.randomUUID().toString();
```

```
        String uuid = UUID.randomUUID().toString();
        String id = uuid.replace("-", "");
        //设置请求ID
        request.setId(id);
        System.out.println("id = " + id);
        //设置API版本号
        request.setApiVer("1.0.1");
        Map<String, String> headers = new HashMap<>();
        String path = "/industry/user/role/all/list";
        String metaDataListString = postBody(path, request, headers);
        JSONObject jsonData = JSON.parseObject(metaDataListString);
        int code = jsonData.getInteger("code");
        if (code == 200) {
            JSONArray dtoList = jsonData.getJSONArray("data");
            dtoList.stream().forEach(dto -> System.out.print("角色:  " + dto.toJSONString())
);
        }
    }
}
```

组织树查询，接口说明

名称	路径	描述
getOrganizationTree	/industry/user/organization/tree/get	获取组织树，最深返回4级组织

分页查询账号列表，接口说明

名称	路径	描述
queryAccount	/industry/user/account/query	根据账号关键字、组织编号和角色码分页查询用户账号

获取账号信息，接口说明

名称	路径	描述
getAccount	/industry/user/account/get	根据employeeId获取员工信息，employeeId不传则获取租户主账号信息

变更订阅通知

集成工作概述中的数据变更通知订阅说明了接口的使用方式，其中serviceType参数设置为2，表示账号、组织和角色数据通知，action分别对应新增/修改/删除操作，data是一个json对象或对象数组，根据type参数表示不同类型的数据，账号、组织和角色数据格式与查询接口返回的数据对象一致。

账号数据通知格式

```
{
  "appId": "xxxxx",//对于SaaS应用，这个对应的是应用的实例id
  "serviceType": 2,//对于账号、组织和角色权限固定为2
  "type": "ACCOUNT",
  "action": "INSERT"/"MODIFY"/"DELETE",
  "data": {
    "auid": "xxxxxx",
    "phone": "xxxxxxx",
    "email": "test_1@xxxxx.com",
    "nickName": "tpc",
    "admin": false,
    "main": false,
    "roleList": [
      {
        "roleCode": "ADMINISTRATOR",
        "roleName": "系统集成商"
      }
    ],
    "orgList": [
      {
        "organizationId": "xxxxxx",
        "organizationName": "研发部"
      }
    ]
  }
}
```

组织变更通知格式

```
{
  "appId": "xxxxx",//对于SaaS应用，这个对应的是应用的实例id
  "serviceType": 2,//对于账号、组织和角色权限固定为2
  "type": "ORGANIZATION",
  "action": "INSERT"/"MODIFY"/"DELETE",
  "data": {
    "organizationId": "xxxxx",
    "organizationName": "公司",
    "order": 1,
    "parentId": null,
    "childList": [
      {
        "organizationId": "xxxxx",
        "organizationName": "工厂",
        "order": 1,
        "creator": null,
        "modifier": null,
        "leaf": false,
        "parentId": "xxxxx",
        "childList": [
          {
            "organizationId": "xxxxx",
            "organizationName": "财务部",
            "order": 1,
            "parentId": "xxxxx",
            "leaf": true,
            "childList": null
          },
          {
            "organizationId": "xxxxx",
            "organizationName": "生产部",
            "order": 2,
            "parentId": "xxxxx",
            "leaf": true,
            "childList": null
          }
        ]
      }
    ]
  }
}
```

角色变更通知格式

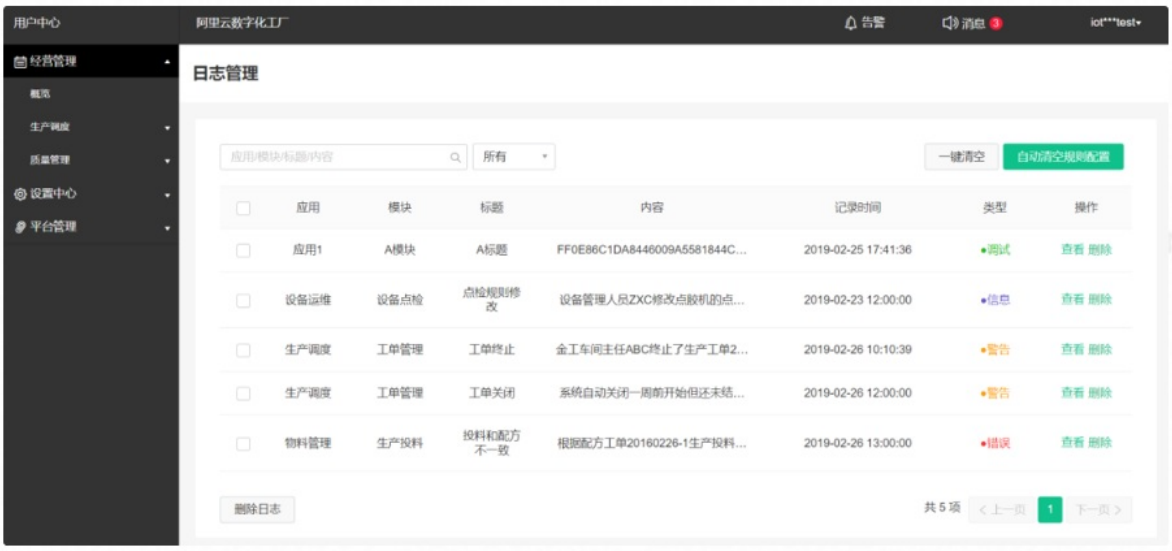
```
{
  "appId": "xxxxx",//对于SaaS应用，这个对应的是应用的实例id
  "serviceType": 2,//对于账号、组织和角色权限固定为2
  "type": "ROLE",
  "action": "INSERT"/"MODIFY"/"DELETE",
  "data": {
      "roleCode": "xxxx_FF0E86C1DA8446009A5581844C244097_xxxx",
      "roleName": "车间主管"
    }
}
```

3.3. 日志集成

为方便工厂的审计，数字化工厂提供统一的日志管理功能，日志管理能够统一管理所有的业务操作日志和应用的调试、错误日志。各个业务的日志统一通过数字工厂提供的日志创建API创建并记录到平台中，由业务用户进行日志查询、导出、归档等操作。

日志记录分成调试/信息/警告/错误四个级别：

- 调试级别的日志主要是为应用在试用阶段或者项目交付前使用，这个级别的日志保留时间将比较短；
- 信息级别的日志主要为业务操作日志，例如设备运维应用中设备管理人员创建或者修改了某类型设备的点检规则，这类信息可以作为业务追溯的数据依据；
- 警告级别的日志，主要出现在对数据进行修改和删除中，应用系统做成了提示后用户继续进行操作，例如终止生产工单的操作应用将对用户进行提示并说明操作的影响，如果用户进行操作可以进行警告级别的操作，还有一些应用动作不是由用户触发，而是根据一定规则应用触发的可以对这些动作进行警告级别的日志记录便于追溯，例如定期检查工单是否关闭，应用根据规则主动关闭一些超时工单，可以通过警告日志记录主动关闭的工单号；
- 错误级别的日志主要记录引起业务错误操作的记录，例如根据配方进行物料防误检测，发现投料和配方不一致，可以记录错误级别日志以进行质量追溯。



应用调用日志创建方法请参看接口说明

3.4. 消息集成

数字工厂提供第三方应用把提醒、故障和报警消息通知到指定员工、组织或者角色。

比如说物料管理的第三方应用判断A类原材料低于安全库存水位以后会自动创建补货单，同时需要发出报警消息通知到“原材料仓管员”这样的角色，那么首先通过应用鉴权获得角色列表，然后提供页面选择“原材料管理员”这个角色，获得角色标识以后，可以通过消息集成接口发送报警消息。

消息集成方法请参看触发用户自定义内容和级别的报警消息

Demo代码参考

```
public class AlarmApplication {
    public static String postBody(String path, IoTApiRequest request, Map<String, String> h
eaders) {
        IoTApiClientBuilderParams ioTApiClientBuilderParams = new IoTApiClientBuilderParams
();
        ioTApiClientBuilderParams.setAppKey("");
        ioTApiClientBuilderParams.setAppSecret("");
        SyncApiClient syncApiClient = new SyncApiClient(ioTApiClientBuilderParams);
        try {
            //设置请求参数域名、path、request ,如果使用HTTPS，设置为true
            ApiResponse response = syncApiClient.postBody("api.link.aliyun.com", path, requ
est, true, headers);
            String responseString = new String(response.getBody(), "UTF-8");
            System.out.println("response code = " + response.getCode() + " response = " + r
esponseString);
            return responseString;
        } catch (UnsupportedEncodingException uee) {
            System.out.println(uee);
            return null;
        }
    }
    public static void main(String [] args) {
        IoTApiRequest request = new IoTApiRequest();
        request.setVersion("1.0");
        String uuid = UUID.randomUUID().toString();
        String id = uuid.replace("-", "");
        //设置请求ID
        request.setId(id);
        System.out.println("id = " + id);
        //设置API版本号
        request.setApiVer("1.0.0");
        Map<String, String> headers = new HashMap<>();
        String path = "/industry/message/alarm/triggerByContent";
        request.putParam("level", 0);
        request.putParam("title", "testTitle");
        request.putParam("content", "testContent");
        request.putParam("receiverType", "role");
        request.putParam("receiverList", Arrays.asList("role1", "role2", "role3"));
        String response = postBody(path, request, headers);
        System.out.println(response);
    }
}
```

4.基础数据集成

4.1. 元数据定义

可以通过接口定义本应用需要的元数据，在调用该接口时候需要获得用户的定义授权。

创建元数据

定义新的元数据，创建元数据的接口说明，示例代码：

```
{
  "name": "物料", // 名称，必填
  "description": "物料", // 描述
  "multiVersion": true/false, // 是否多版本，必填
  "prefixNested": true/false, // 是否通过前缀表示上下层关系，必填
  // 主数据的属性列表
  "properties": [{
    "propertyCode": "code", // 属性标识，必填
    "propertyDesc": "物料编码",
    "propertyType": "STRING"/"INTEGER"/"DOUBLE"/"ENUM"/"FACTORY"/"TECHNOLOGY"/"WAREHOUSE",
// 属性类型，必填
    // 属性的限制描述
    "propertyLimit": {
      // 属性类型为INTEGER/DOUBLE时，这两个字段有效，值为浮点数
      "min": 1.0,
      "max": 100,
      // 属性类型为STRING时，这个限制字符串的长度
      "len": 64,
      // 属性类型为FACTORY时，这个字段表示具体的工厂节点类型
      "factoryType": "FACTORY"/"WORKSHOP"/"BELTLINE"/"MACHINING_CENTER",
      // 属性类型为TECHNOLOGY时，这个字段表示具体的工艺路径的节点类型
      "technologyType": "TECHNOLOGY"/"PROCESS"/"STEP",
      // 属性类型为WAREHOUSE时，这个字段表示具体的库存节点类型
      "warehouseType": "WAREHOUSE"/"AREA"/"LOCATION",
      // 属性类型为ENUM时，这个记录枚举的所有值
      "enumValues":[{
        "value": "0",
        "remark": "早班"
      }, {
        "value": "1",
        "remark": "中班"
      }, {
        "value": "2",
        "remark": "晚班"
      }],
      // 属性类型为BOOLEAN时，这个记录布尔值
      "booleanValues": [{
        "value": "1",
        "remark": "真"
      }, {
        "value": "0",
        "remark": "假"
      }]
    },
    "isUnique": true/false, // 是否唯一键
    "isNull": true/false, // 是否可空
    "defaultValue": "xxx", // 默认值，不管属性是什么类型，这里都用字符串
  }]
}
```

创建元数据成功的话，会返回该元数据id。

修改元数据

修改/编辑元数据的接口说明：

-
- 如果id未传，用name做标识，可以修改描述、多版本；如果传入了id，则还可以修改名称
- 用propertyCode做为标识修改已有的属性信息
- 删除已有的属性
- 创建新的属性

删除元数据

元数据中如果没有主数据，可以通过API删除元数据定义。删除元数据的接口说明

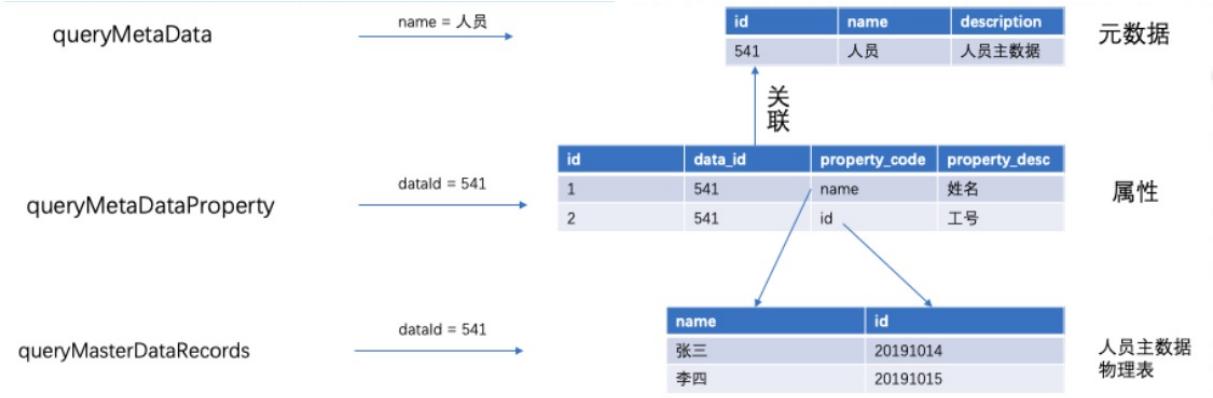
4.2. 主数据集成

企业主数据是用来描述企业核心业务实体的数据，例如供应商、员工、产品、物料、设备等；它是具有高业务价值的、可以在企业内跨越各个业务部门被重复使用的数据，并且存在于多个异构的应用中。数字工厂提供统一的工厂主数据的云数据定义和数据管理功能，第三方应用系统可以通过API查询主数据的元数据和数据实例或者采用订阅方式获得主数据变更通知。

例如设备运维的第三方应用需要用到数字工厂的设备类型、设备种类和生产设备三种设备主数据，首先通过元数据查询接口获得所有元数据信息，然后根据设备类型、设备种类和生产设备的元数据定义，去订阅这三类主数据，一旦企业的业务用户在数字工厂中对主数据进行了操作，新采购了一台设备，创建了对应的生产设备主数据，将通过接口通知设备运维的第三方应用，应用就可以对新的设备进行运维操作。主数据集成的说明请参看开放接口更新主数据记录

元数据、主数据和主数据记录

每一种主数据都对应一张物理表，对应的元数据就是这张物理表的描述信息。上面提到的供应商、员工、产品、物料、设备这些都是相应主数据（物理表）的业务描述。



系统内建主数据

系统包含了以下几组的内建主数据：

- 人员，部门
- 物料组，物料类型，物料，计量单位分组，计量单位，计量单位转换，物料包装规格
- 设备类型，设备型号，生产设备
- 供应商
- 客户分组、客户

内建主数据的属性定义如下表：

人员

属性名称	属性类型	属性标识	唯一标识	必填	显示属性	是否多值	默认值
中文名	字符串	name	否	是	是	否	
工号	字符串	id	是	是	否	否	
手机号码	字符串	phone	否	否	否	是	
入职时间	日期	hired_date	否	否	否	否	
卡号	字符串	cardno	否	否	否	否	
身份证	字符串	card_ID	否	否	否	否	
邮箱	字符串	email	否	否	否	否	
通讯地址	字符串	postadd	否	否	否	否	
性别	枚举 0-男 1-女	sex	否	否	否	否	0
邮编	字符串	postcode	否	否	否	否	
生日	日期	birthday	否	否	否	否	
职务	字符串	duty	否	否	否	否	
学历	字符串	education	否	否	否	否	
备注	字符串	remark	否	否	否	否	
编码	字符串	code	否	否	否	否	
部门	主数据（部门）	department	否	否	否	否	

部门

属性名称	属性类型	属性标识	唯一标识	必填	显示属性	是否多值	默认值
编码	字符串	code	是	是	否	否	
名称	字符串	name	否	是	是	否	
描述	字符串	description	否	否	否	否	
全称	字符串	fullname	否	否	否	否	

属性名称	属性类型	属性标识	唯一标识	必填	显示属性	是否多值	默认值
上级部门	主数据（部门）	pardepartment	否	否	否	否	

物料组

属性名称	属性类型	属性标识	唯一标识	必填	显示属性	是否多值	默认值
编码	字符串	code	是	是	否	否	
名称	字符串	name	否	是	是	否	
描述	字符串	description	否	否	否	否	
简码	字符串	brevity_code	否	否	否	否	

物料类型

属性名称	属性类型	属性标识	唯一标识	必填	显示属性	是否多值	默认值
编码	字符串	code	是	是	否	否	
名称	字符串	name	否	是	是	否	
描述	字符串	description	否	否	否	否	
简码	字符串	brevity_code	否	否	否	否	
所属物组	主数据（物料组）	material_group	否	否	否	否	未知物料组
工艺路径	工艺路径	bop	否	否	否	是	
备注	字符串	remark	否	否	否	否	
上级类型	主数据（物料类型）	pmaterial_type	否	否	否	否	

物料

属性名称	属性类型	属性标识	唯一标识	必填	显示属性	是否多值	默认值
编码	字符串	code	是	是	否	否	
名称	字符串	name	否	是	是	否	

属性名称	属性类型	属性标识	唯一标识	必填	显示属性	是否多值	默认值
描述	字符串	description	否	否	否	否	
简码	字符串	brevity_code	否	否	否	否	
所属物类型	主数据（物料类型）	material_type	否	否	否	否	未知物料类型
规格	字符串	spec	否	否	否	否	
等级	字符串	grade	否	否	否	否	
存储周期(小时)	浮点	storageperiod	否	否	否	否	
有效期(小时)	浮点	validityhours	否	否	否	否	
安全库存	浮点	safetystock	否	否	否	否	
主供应商	主数据（供应商）	supplier	否	否	否	否	
备注	字符串	remark	否	否	否	否	
计量单位	主数据（计量单位）	UOM	否	否	否	否	
工艺路径	工艺路径	bop	否	否	否	是	

计量单位分组

属性名称	属性类型	属性标识	唯一标识	必填	显示属性	是否多值	默认值
编码	字符串	code	是	是	否	否	
名称	字符串	name	否	是	是	否	
基准单位	字符串	baseunit	否	否	否	否	

计量单位

属性名称	属性类型	属性标识	唯一标识	必填	显示属性	是否多值	默认值
编码	字符串	code	是	是	否	否	
名称	字符串	name	否	是	是	否	

属性名称	属性类型	属性标识	唯一标识	必填	显示属性	是否多值	默认值
备注	字符串	remark	否	否	否	否	
分组	主数据（计量单位分组）	group	否	否	否	否	

计量单位转换

属性名称	属性类型	属性标识	唯一标识	必填	显示属性	是否多值	默认值
编号	字符串	code	是	是	否	否	
主计量单位	主数据（计量单位）	firstUOM	否	是	是	否	
数量	浮点	firstQuantity	否	是	否	否	
转换计量单位	主数据（计量单位）	secondUOM	否	是	否	否	
转换数量	浮点	secondQuantity	否	否	否	否	
物料	主数据（物料）	material	否	否	否	否	

物料包装规格

属性名称	属性类型	属性标识	唯一标识	必填	显示属性	是否多值	默认值
编码	字符串	code	是	是	否	否	
名称	字符串	name	否	是	是	否	
物料	主数据（物料）	material	否	否	否	否	
条码	字符串	brevity_code	否	否	否	否	
是否最小包装	布尔	isMPQ	否	否	否	否	1
是否默认包装	布尔	isDefault	否	否	否	否	0
高度	浮点	height	否	否	否	否	

属性名称	属性类型	属性标识	唯一标识	必填	显示属性	是否多值	默认值
长度	浮点	length	否	否	否	否	
宽度	浮点	width	否	否	否	否	
体积	浮点	volume	否	否	否	否	
毛重	浮点	grossweight	否	否	否	否	
净重	浮点	netweight	否	否	否	否	

设备类型

属性名称	属性类型	属性标识	唯一标识	必填	显示属性	是否多值	默认值
编码	字符串	code	是	是	否	否	
名称	字符串	name	否	是	是	否	
描述	字符串	description	否	否	否	否	
简码	字符串	brevity_code	否	否	否	否	

设备型号

属性名称	属性类型	属性标识	唯一标识	必填	显示属性	是否多值	默认值
编码	字符串	code	是	是	否	否	
名称	字符串	name	否	是	是	否	
所属设备类型	主数据（设备类型）	equipment_type	否	是	否	否	
描述	字符串	description	否	否	否	否	
简码	字符串	brevity_code	否	否	否	否	
额定功率	整型	powerrating	否	否	否	否	
品牌	字符串	brand	否	否	否	否	

生产设备

属性名称	属性类型	属性标识	唯一标识	必填	显示属性	是否多值	默认值
固定资产编号	字符串	asset_num	是	是	否	否	
名称	字符串	name	否	是	是	否	
描述	字符串	description	否	否	否	否	
使用状态	枚举 1-新建 2-使用中 3-停用 4-报废	usable_condition	否	是	否	否	1
所属物类型	主数据（物料类型）	equipment_model	否	是	否	否	
简码	字符串	brevity_code	否	否	否	否	
描述	字符串	description	否	否	否	否	
物联网设备	物联网设备	iot_device_id	否	否	否	否	
出厂日期	日期	out_factory_date	否	否	否	否	
入厂日期	日期	enter_factory_date	否	否	否	否	
设备管理员	主数据（人员）	owner	否	否	否	否	
设备位置	字符串	location	否	否	否	否	
设备SN码	字符串	sn_code	否	否	否	否	

供应商信息

属性名称	属性类型	属性标识	唯一标识	必填	显示属性	是否多值	默认值
编码	字符串	code	是	是	否	否	
名称	字符串	name	否	是	是	否	
地址	字符串	address	否	否	否	否	

属性名称	属性类型	属性标识	唯一标识	必填	显示属性	是否多值	默认值
邮编	字符串	postcode	否	否	否	否	1
证件号码	字符串	certificate	否	否	否	否	
联系人	字符串	contact	否	否	否	否	
联系电话	字符串	phone	否	否	否	否	
邮箱	字符串	email	否	否	否	否	
业务员	主数据（人员）	businessman	否	否	否	否	
备注	字符串	remark	否	否	否	否	
手机号码	字符串	mobile	否	否	否	否	
简称	字符串	brevity_code	否	否	否	否	

客户分组

属性名称	属性类型	属性标识	唯一标识	必填	显示属性	是否多值	默认值
编码	字符串	code	是	是	否	否	
名称	字符串	name	否	是	是	否	
上级分组	主数据（客户分组）	parentGroup	否	否	否	否	

客户

属性名称	属性类型	属性标识	唯一标识	必填	显示属性	是否多值	默认值
编码	字符串	code	是	是	否	否	
名称	字符串	name	否	是	是	否	
地址	字符串	address	否	否	否	否	
联系人	字符串	contact	否	否	否	否	
联系电话	字符串	phone	否	否	否	否	
备注	字符串	remark	否	否	否	否	
简称	字符串	brevity_code	否	否	否	否	

属性名称	属性类型	属性标识	唯一标识	必填	显示属性	是否多值	默认值
分组	主数据（客户分组）	group	否	否	否	否	

扩展说明：用户可以对内建主数据的属性进行扩展，增加新的属性，也可以增加新的主数据类型。

主数据集成概述

租户概述：租户用于确定一个企业账户。一个企业账户购买数字工厂或三方应用后，可以给企业的多个员工使用。通常，不同员工可能对应不同角色。访问时需要根据员身份进行鉴权。

对应用来说，租户与租户之间要进行区分和隔离。对于独占式应用，租户与租户间是通过不同的应用实例进行物理隔离。对于共享式应用，租户与租户之间通过逻辑隔离。

集成需要做的事，

如果应用有菜单集成到数字工厂，先确定免登方式；

确定调用的参数，以区分api调用哪个租户相关，以及确定，与某一租户的哪一个具体员工相关。

以下demo代码都是基于github的openApiSDK

https://github.com/aliyun/iotx-api-gateway-client.git

1、查询

查询接口有三个，分，别是查询元数据、查询元数据属性列表、查询主数据记录

（1）查询元数据

这个接口的作用在于：得到系统支持所有主数据的描述信息（即元数据），包括，id， name, 描述，是否支持多版本，是否内建主数据等。内建主数据不支持删除对应的主数据表。

数据id会在其它所有的接口中用到，是系统内识别主数据表的标识信息。

不指定id，可以查询系统里所有元数据的列表，也可以指定id，查询一种具体的元数据。

入参列表

参数名	描述	备注
id	元数据id	用id查询指定的元数据
name	元数据名称	用name查询，后台是模糊匹配
pageId	分面查询时的页号	
pageSize	分面查询时的页大小	
appId	应用的实例id(uuid)	
employeeId	阿里云账号的子账号id	

出参示例

```
{
  "code": 200,
  "message": "success",
  "localizedMsg": null,
  "data": {
    "digitalFactoryMasterDataDTOList": [{
        "id": 541,
        "name": "人员",
        "description": "人员主数据",
        "multiVersion": 0,
        "builtin": 1,
        "gmtCreate": "2019-02-22T02:25:29.000Z",
        "gmtModified": "2019-02-22T02:25:29.000Z"
      }]
  }
}
```


字段名	字段描述	备注
id	元数据id	
name	元数据名称/主数据名称	
description	元数据描述	
multiVersion	是否多版本，1表示多版本	如果是多版本的主数据，唯一键和版本号合起来唯一确定一条数据记录
builtin	是否内建，1表示内建	内建的元数据不允许修改/删除，可以新增属性字段

java demo

```
import com.alibaba.cloudapi.sdk.model.ApiResponse;
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONArray;
import com.alibaba.fastjson.JSONObject;
import com.aliyun.iotx.api.client.IoTApiClientBuilderParams;
import com.aliyun.iotx.api.client.IoTApiRequest;
import com.aliyun.iotx.api.client.SyncApiClient;
import org.apache.commons.collections4.CollectionUtils;
import java.io.UnsupportedEncodingException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.UUID;
import java.util.stream.Collectors;
public class DemoApplication {
    private static String postBody(String path, IoTApiRequest request, Map<String, String>
headers) {
        String appKey = "";
        String appSecret = "";
        IoTApiClientBuilderParams ioTApiClientBuilderParams = new IoTApiClientBuilderParams
```

```
();
        ioTApiClientBuilderParams.setAppKey(appKey);
        ioTApiClientBuilderParams.setAppSecret(appSecret);
        SyncApiClient syncApiClient = new SyncApiClient(ioTApiClientBuilderParams);
        try {
            //设置请求参数域名、path、request ,如果使用HTTPS，设置为true
            ApiResponse response = syncApiClient.postBody("api.link.aliyun.com", path, requ
est, true, headers);
            String responseString = new String(response.getBody(), "UTF-8");
            System.out.println("response code = " + response.getCode() + " response = " + r
esponseString);
            return responseString;
        } catch (UnsupportedEncodingException uee) {
            System.out.println(uee.getMessage());
            uee.printStackTrace();
            return null;
        }
    }
    public static void main(String[] args) {
        IoTApiRequest request = new IoTApiRequest();
        //设置协议版本号
        request.setVersion("1.0");
        String uuid = UUID.randomUUID().toString();
        String id = uuid.replace("-", "");
        //设置请求ID
        request.setId(id);
        System.out.println("id = " + id);
        //设置API版本号
        request.setApiVer("1.0.1");
        Map<String, String> headers = new HashMap<>();
        String path = "/industry/metadata/query";
        String metaDataListString = postBody(path, request, headers);
        JSONObject jsonData = JSON.parseObject(metaDataListString);
        int code = jsonData.getInteger("code");
        if (code == 200) {
            JSONObject data = jsonData.getJSONObject("data");
            JSONArray dtoList = data.getJSONArray("digitalFactoryMasterDataDTOList");
            List<JSONObject> targetDTOList = dtoList.stream().filter(dto -> ((JSONObject)dt
o).getString("name").equals("人员")).map(dto -> (JSONObject)dto).collect(Collectors.toList()
);
            if (CollectionUtils.isNotEmpty(targetDTOList)) {
                System.out.print("人员元数据id " + targetDTOList.get(0).getLong("id"));
            }
        }
 }
}
```

（2）查询主数据属性列表

该接口的作用在于获取具体某个主数据的属性信息，包括属性类型、属性长度/大小的限制。

dataId必须传入，表示查询哪个主数据的属性信息。

id选填，查询具体的某一个属性。

以人员主数据为例来说明

入参

参数名	描述
id	属性的id
dataId	所属元数据的id
pageId	分页查询时的页号
pageSize	分页查询时的页大小
appId	应用的实例id（uuid）
employeeId	阿里云账号的子账号id

出参示例


```
{
  "code": 200,
  "data": {
    "propertyDTOList": [
      {
        "defaultValue": "",
        "builtin": 1,
        "display": false,
        "isUnique": 2,
        "propertyLimit": {
          "len": 64
        },
        "propertyCode": "name",
        "dataId": 652,
        "array": false,
        "propertyDesc": "中文名",
        "propertyType": "STRING",
        "isNull": 4,
        "propertyIndex": 3,
        "id": 1
      },
      {
        "defaultValue": "",
        "builtin": 1,
        "display": true,
        "isUnique": 1,
        "propertyLimit": {
          "len": 32
        },
        "propertyCode": "id",
        "dataId": 652,
        "array": false,
        "propertyDesc": "工号",
        "propertyType": "STRING",
        "isNull": 4,
```

```
      "propertyIndex": 4,
      "id": 2
    },
    {
      "defaultValue": "",
      "builtin": 1,
      "display": false,
      "isUnique": 2,
      "propertyLimit": {
        "len": 256
      },
      "propertyCode": "remark",
      "dataId": 652,
      "array": false,
      "propertyDesc": "备注",
      "propertyType": "STRING",
      "isNull": 3,
      "propertyIndex": 4,
      "id": 3
    },
    {
      "defaultValue": "",
      "builtin": 1,
      "display": false,
      "isUnique": 2,
      "propertyLimit": {
        "len": 32
      },
      "propertyCode": "phone",
      "dataId": 652,
      "array": false,
      "propertyDesc": "手机号码",
      "propertyType": "STRING",
      "isNull": 3,
      "propertyIndex": 5,
      "id": 4
    },
    {
      "defaultValue": "",
      "builtin": 1,
      "display": false,
      "isUnique": 2,
      "propertyLimit": {
        "len": 32
      },
      "propertyCode": "hired_date",
      "dataId": 652,
      "array": false,
      "propertyDesc": "入职时间",
      "propertyType": "DATE",
      "isNull": 3,
      "propertyIndex": 6,
      "id": 5
    }
```

```
    ]
  },
  "id": "787b121b3863478a8cde800142853dce"
}
```

字段名	字段描述	备注
id	属性id	
dataId	所属元数据id	
propertyCode	属性编码	物理表的字段名
propertyDesc	属性的描述	
propertyType	属性类型	
builtin	1表示内建属性	
defaultValue	属性的默认值	
display	true表示该属性做为显示项属性	主数据A的属性P关联了主数据B，那么展示的时候，B的显示项属性决定了A的属性P的显示内容
isUnique	1表示该属性是唯一键，2表示非唯一键	
isNull	3表示该属性可空，4表示不可空	
array	true表示该属性可以填多个值	
propertyLimit	属性的其它限制信息	字符串的长度限制、整数的范围限制、关联的主数据都在这个字段里面记录
propertyIndex	属性的下标	用于按顺序显示，小的在前面

java demo

```
import com.alibaba.cloudapi.sdk.model.ApiResponse;
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONArray;
import com.alibaba.fastjson.JSONObject;
import com.aliyun.iotx.api.client.IoTApiClientBuilderParams;
import com.aliyun.iotx.api.client.IoTApiRequest;
import com.aliyun.iotx.api.client.SyncApiClient;
import org.apache.commons.collections4.CollectionUtils;
import org.apache.commons.lang3.StringUtils;
import java.io.UnsupportedEncodingException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.UUID;
import java.util.stream.Collectors;
public class DemoApplication {
    private static String postBody(String path, IoTApiRequest request, Map<String, String>
```

```
headers) {
        String appKey = "";
        String appSecret = "";
        IoTApiClientBuilderParams ioTApiClientBuilderParams = new IoTApiClientBuilderParams
();
        ioTApiClientBuilderParams.setAppKey(appKey);
        ioTApiClientBuilderParams.setAppSecret(appSecret);
        SyncApiClient syncApiClient = new SyncApiClient(ioTApiClientBuilderParams);
        try {
            //设置请求参数域名、path、request ,如果使用HTTPS，设置为true
            ApiResponse response = syncApiClient.postBody("api.link.aliyun.com", path, requ
est, true, headers);
            String responseString = new String(response.getBody(), "UTF-8");
            System.out.println("response code = " + response.getCode() + " response = " + r
esponseString);
            return responseString;
        } catch (UnsupportedEncodingException uee) {
            System.out.println(uee.getMessage());
            uee.printStackTrace();
            return null;
        }
    }
    private static Long requestMetaDataId(String name, String appId) {
        IoTApiRequest request = new IoTApiRequest();
        //设置协议版本号
        request.setVersion("1.0");
        String uuid = UUID.randomUUID().toString();
        String id = uuid.replace("-", "");
        //设置请求ID
        request.setId(id);
        System.out.println("id = " + id);
        //设置API版本号
        request.setApiVer("1.0.1");
        request.putParam("pageId", 1);
        request.putParam("pageSize", 200);
        Map<String, String> headers = new HashMap<>();
        if (StringUtils.isNotEmpty(appId)) {
            request.putParam("appId", appId);
        }
        String path = "/industry/metadata/query";
        String metaDataListString = postBody(path, request, headers);
        JSONObject jsonData = JSON.parseObject(metaDataListString);
        int code = jsonData.getInteger("code");
        if (code == 200) {
            JSONObject data = jsonData.getJSONObject("data");
            JSONArray dtoList = data.getJSONArray("digitalFactoryMasterDataDTOList");
            List<JSONObject> targetDTOList = dtoList.stream().filter(dto -> ((JSONObject)dt
o).getString("name").equals(name)).map(dto -> (JSONObject)dto).collect(Collectors.toList())
;
            if (CollectionUtils.isNotEmpty(targetDTOList)) {
                return targetDTOList.get(0).getLong("id");
            }
        }
        return null;
```

```
    }
    private static void requestPropertyList(Long dataId, String appId) {
        IoTApiRequest request = new IoTApiRequest();
        //设置协议版本号
        request.setVersion("1.0");
        String uuid = UUID.randomUUID().toString();
        String id = uuid.replace("-", "");
        //设置请求ID
        request.setId(id);
        System.out.println("id = " + id);
        //设置API版本号
        request.setApiVer("1.0.1");
        request.putParam("pageId", 1);
        request.putParam("pageSize", 200);
        Map<String, String> headers = new HashMap<>();
        String path = "/industry/metadata/property/query";
        request.putParam("dataId", dataId);
        if (StringUtils.isNotEmpty(appId)) {
            request.putParam("appId", appId);
        }
        String propertyListString = postBody(path, request, headers);
        System.out.println(propertyListString);
    }
    public static void main(String[] args) {
        Long employeeDataId = requestMetaDataId("人员", null);
        if (employeeDataId != null) {
            requestPropertyList(employeeDataId, null);
        }
    }
}
```

（3）查询主数据具体记录

该接口用于查询某个主数据的数据记录。

dataId必填，指明查询哪一个主数据的记录。

入参

参数名称	描述	-
dataId	元数据id	指明要查哪一个物理表
dapIds	数据记录的id。后台会给每一张主数据的物理表添加几个字段，其中一个是dap_id_，是一个自增的整型字段。该字段非空说明要查指定dap_id_的数据记录	后台最终执行的sqlselect * from tablewhere dap_id_ in （dapIds）
pageId	分页查询时的页号	-
pageSize	分页查询时的页大小	-
appId	应用实例id	-

参数名称	描述	-
employeeId	阿里云账号的子账号id	-
condition	查询条件	-

举例子说明condition的用法，仍然使用人员主数据

```
// 查询dap_id_为1,2,3的记录，dap_id_ IN (1, 2, 3)
Map<String, Object> condition = new HashMap<>(3);
condition.put("col", "dap_id_");
condition.put("op", "IN");
condition.put("value", Arrays.asList(1, 2, 3));
request.putParam("condition", Collections.singletonList(condition));
// 查询名字(name)为张三的记录，name = "张三"
Map<String, Object> condition = new HashMap<>(3);
condition.put("col", "name");
condition.put("op", "EQUAL");
condition.put("value", "张三");
request.putParam("condition", Collections.singletonList(condition));
// 查询工号(id)包含2019的记录，id LIKE "%2019%"
Map<String, Object> condition = new HashMap<>(3);
condition.put("col", "id");
condition.put("op", "LIKE");
condition.put("value", "2019");
request.putParam("condition", Collections.singletonList(condition));
// 查询工号包含2019并且名字中有三字的员工，id LIKE "%2019%" AND name LIKE "%三%"
Map<String, Object> subCondition1 = new HashMap<>(3);
subCondition1.put("col", "id");
subCondition1.put("op", "LIKE");
subCondition1.put("value", "2019");
Map<String, Object> subCondition2 = new HashMap<>(3);
subCondition1.put("col", "name");
subCondition1.put("op", "LIKE");
subCondition1.put("value", "三");
Map<String, Object> condition = new HashMap<>(2);
condition.put("op", "AND");
condition.put("condition", Arrays.asList(subCondition1, subCondition2));
request.putParam("condition", Collections.singletonList(condition));
```

出参

```
{
  "code": 200,
  "data": "{\"data\":[[\"工人1\",\"01\",\"工人1\",\"1381111****\",\"2019/07/02 00:00:00\",1,
\"已发布\",0]],\1"nodes\":[\"name\",\"id\",\"remark\",\"phone\",\"hired_date\",\"dap_id_\",\
"dap_status_\",\"dap_refcnt_\"],\"page\":{\"size\":1500,\"to\":1,\"total\":1}}",
  "id": "fd3d04805b6f4dd2a8e1376eabfe5b25"
}
```

其中data部分是个json字符串，包含的字段如下

字段名称	描述
nodes	物理表的属性列表
data	物理表的数据部分，是个二维数组，外层数组元素表示数据行，里层数据元素对应属性

上面的输出对应的表数据如下

name	id	remark	phone	hired_date	dap_id_	dap_status_	dap_refcnt_
工人1	1	工人1	1381111****	"2019/07/02 00:00:00"	1	已发布	0

2、写入主数据记录

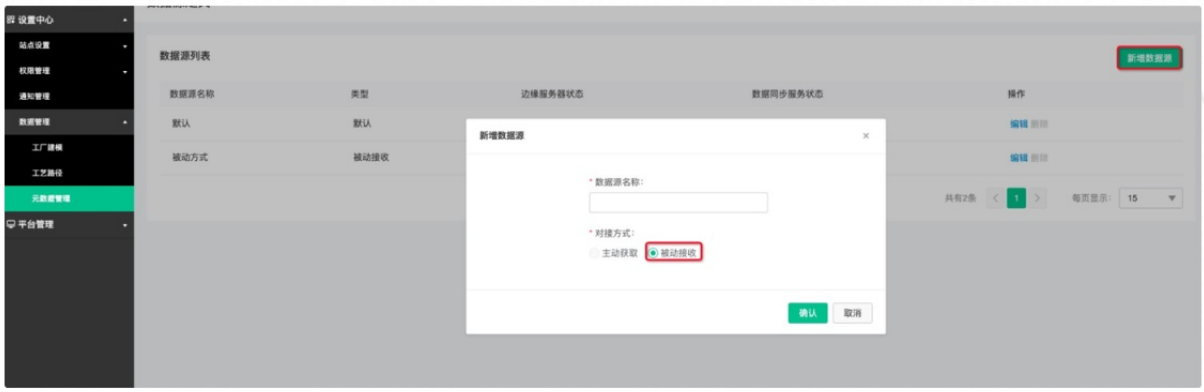
主数据记录的来源有两种：

数字工厂控制台，通过页面新增主数据

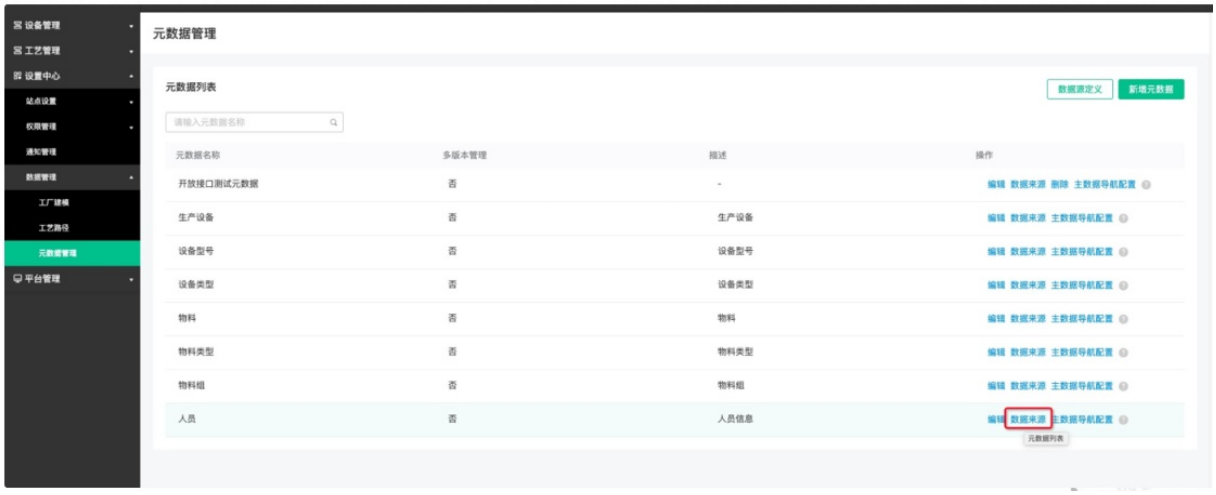
通过api写入。只能用目标租户在入驻数字工厂时生成的appkey调用api写入主数据。这个appkey可以通过数字工厂的菜单：平台管理->阿里生态查到

（1）配置主数据来源

先创建被动接收模式的数据源，被动接收模式即是允许从open api写入主数据



创建被动接收模式的数据源之后，再将一种主数据关联到这个数据源，比如说人员



配置完成之后，人员主数据就不能通过页面修改，只能通过开放api修改

（2）api说明

入参

参数名	描述	
dataId	元数据id	
action	写操作。当前支持的有：INSERT/MODIFY/ARCHIVE	INSERT操作写入的数据默认就是已发布状态
nodes	属性名	ARCHIVE操作时，该属性被忽略
data	数据部分。是个json字符串	INSERT操作，data的格式是个二维数组；MODIFY操作时，data的格式是个一维数组；ARCHIVE操作时，该属性被忽略
condition	更新操作的条件	只对MODIFY/ARCHIVE有效，与查询数据记录接口用用法一致
appId	应用的实例id	
employeeId	阿里云账号的子账号id	

java demo

INSERT

```
import com.alibaba.cloudapi.sdk.model.ApiResponse;
import com.alibaba.fastjson.JSON;
import com.aliyun.iotx.api.client.IoTApiClientBuilderParams;
import com.aliyun.iotx.api.client.IoTApiRequest;
import com.aliyun.iotx.api.client.SyncApiClient;
import java.io.UnsupportedEncodingException;
import java.util.*;
public class InsertTest {
    public static void main(String[] args) {
        IoTApiRequest request = new IoTApiRequest();
        request.setVersion("1.0");
        String uuid = UUID.randomUUID().toString();
        String id = uuid.replace("-", "");
        request.setId(id);
        request.setApiVer("1.0.0");
        request.putParam("dataId", 1296L);
        request.putParam("action", "INSERT");
        String path = "/industry/masterdata/record/write";
        request.putParam("nodes", Arrays.asList("id", "name"));
        request.putParam("data", JSON.toJSONString(Arrays.asList(Arrays.asList(1, "张三"), A
rrays.asList(2, "李四"))));
        IoTApiClientBuilderParams ioTApiClientBuilderParams = new IoTApiClientBuilderParams
();
        ioTApiClientBuilderParams.setAppKey("");
        ioTApiClientBuilderParams.setAppSecret("");
        SyncApiClient syncApiClient = new SyncApiClient(ioTApiClientBuilderParams);
        try {
            //设置请求参数域名、path、request ,如果使用HTTPS，设置为true
            ApiResponse response = syncApiClient.postBody("api.link.aliyun.com", path, requ
est, true, new HashMap<>());
            String responseString = new String(response.getBody(), "UTF-8");
            System.out.println("response code = " + response.getCode() + " response = " + r
esponseString);
        } catch (UnsupportedEncodingException uee) {
            System.out.println(uee.getMessage());
            uee.printStackTrace();
        }
    }
}
```

MODIFY

```
import com.alibaba.cloudapi.sdk.model.ApiResponse;
import com.alibaba.fastjson.JSON;
import com.aliyun.iotx.api.client.IoTApiClientBuilderParams;
import com.aliyun.iotx.api.client.IoTApiRequest;
import com.aliyun.iotx.api.client.SyncApiClient;
import java.io.UnsupportedEncodingException;
import java.util.*;
public class UpdateTest {
    public static void main(String[] args) {
        IoTApiRequest request = new IoTApiRequest();
        request.setVersion("1.0");
        String uuid = UUID.randomUUID().toString();
        String id = uuid.replace("-", "");
        request.setId(id);
        request.setApiVer("1.0.0");
        request.putParam("dataId", 1296L);
        request.putParam("action", "MODIFY");
        String path = "/industry/masterdata/record/write";
        request.putParam("nodes", Collections.singletonList("phone"));
        request.putParam("data", JSON.toJSONString(Collections.singletonList("1381111****")
));
        Map<String, Object> condition = new HashMap<>(3);
        condition.put("op", "EQUAL");
        condition.put("col", "name");
        condition.put("value", "张三");
        request.putParam("condition", Collections.singletonList(condition));
        IoTApiClientBuilderParams ioTApiClientBuilderParams = new IoTApiClientBuilderParams
();
        ioTApiClientBuilderParams.setAppKey("");
        ioTApiClientBuilderParams.setAppSecret("");
        SyncApiClient syncApiClient = new SyncApiClient(ioTApiClientBuilderParams);
        try {
            //设置请求参数域名、path、request ,如果使用HTTPS，设置为true
            ApiResponse response = syncApiClient.postBody("api.link.aliyun.com", path, requ
est, true, new HashMap<>());
            String responseString = new String(response.getBody(), "UTF-8");
            System.out.println("response code = " + response.getCode() + " response = " + r
esponseString);
        } catch (UnsupportedEncodingException uee) {
            System.out.println(uee.getMessage());
            uee.printStackTrace();
        }
    }
}
```

ARCHIVE

```
import com.alibaba.cloudapi.sdk.model.ApiResponse;
import com.aliyun.iotx.api.client.IoTApiClientBuilderParams;
import com.aliyun.iotx.api.client.IoTApiRequest;
import com.aliyun.iotx.api.client.SyncApiClient;
import java.io.UnsupportedEncodingException;
import java.util.*;
public class ArchiveTest {
    public static void main(String[] args) {
        IoTApiRequest request = new IoTApiRequest();
        request.setVersion("1.0");
        String uuid = UUID.randomUUID().toString();
        String id = uuid.replace("-", "");
        request.setId(id);
        request.setApiVer("1.0.0");
        request.putParam("dataId", 1296L);
        request.putParam("action", "ARCHIVE");
        String path = "/industry/masterdata/record/write";
        Map<String, Object> condition = new HashMap<>(3);
        condition.put("op", "IN");
        condition.put("col", "id");
        condition.put("value", Arrays.asList(1, 2));
        request.putParam("condition", Collections.singletonList(condition));
        IoTApiClientBuilderParams ioTApiClientBuilderParams = new IoTApiClientBuilderParams
();
        ioTApiClientBuilderParams.setAppKey("112345");
        ioTApiClientBuilderParams.setAppSecret("67890");
        SyncApiClient syncApiClient = new SyncApiClient(ioTApiClientBuilderParams);
        try {
            //设置请求参数域名、path、request ,如果使用HTTPS，设置为true
            ApiResponse response = syncApiClient.postBody("api.link.aliyun.com", path, requ
est, true, new HashMap<>());
            String responseString = new String(response.getBody(), "UTF-8");
            System.out.println("response code = " + response.getCode() + " response = " + r
esponseString);
        } catch (UnsupportedEncodingException uee) {
            System.out.println(uee.getMessage());
            uee.printStackTrace();
        }
    }
}
```

3、主数据消息通知格式

action分别对应创建/修改/删除/发布/归档的操作

nodes是实际数据表的属性，也是实际在界面上配置的属性编码。

dap_打头的属性是系统内部生成的

dap_id_是数据记录的编号

dap_status_是记录的状态，有待发布/已发布/已归档三个状态

对于多版本的主数据，还会有一个dap_row_version_的属性，记录版本号

data是一个二维数组，里面的每一个数组表示数据表中的一行数据，按顺序与nodes对应

如果某个属性是主数据类型，推送的消息中，会多一个属性，该属性的标识是原属性标识拼上“_key”；如果被关联的主数据是多版本的，还会多一个属性，该属性的标识是原属性标识拼上“_version”

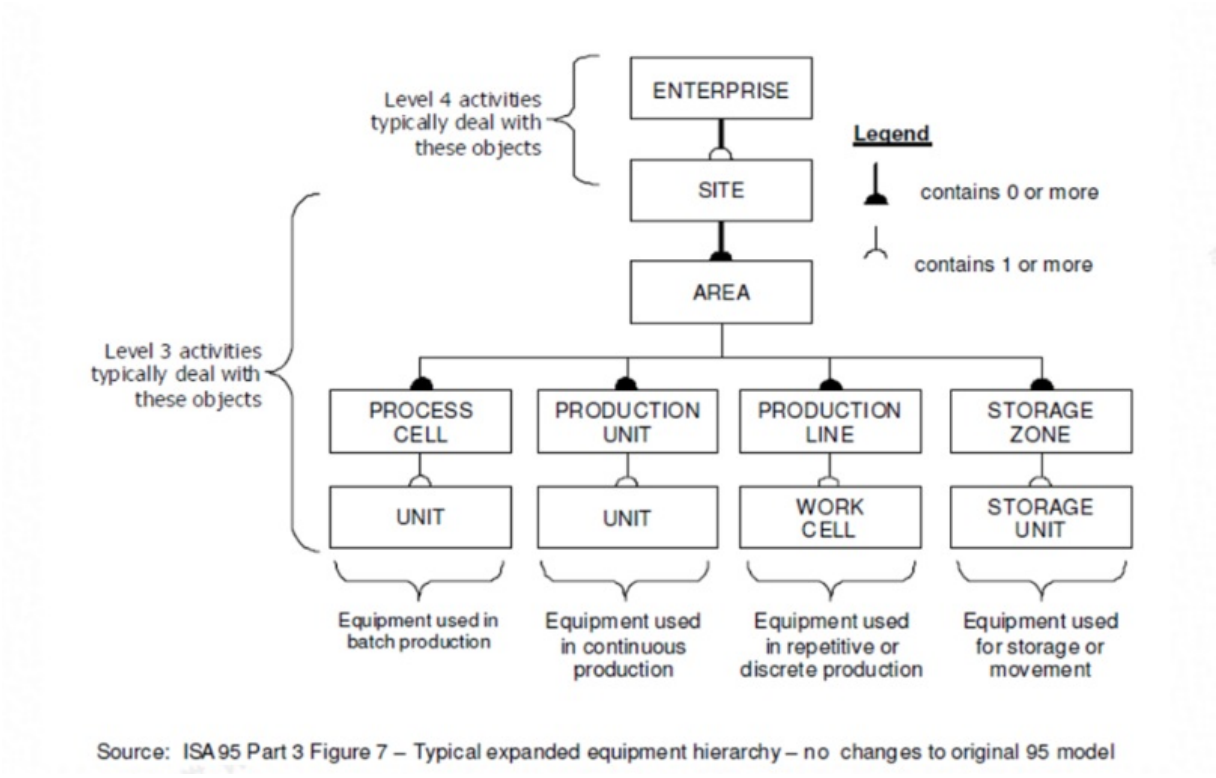
例如，生产设备有一个所属设备型号的属性（equipment_model），推送的消息中，会多一个equipment_model_key的属性，其值为该所属设备型号唯一键的值；如果设备型号是多版本的，推送的消息中会多一个

equipment_model_version的属性，其值为所属设备型号的版本号的值。

```
{
  "appId": "xxxxx", // 对于SaaS应用，这个对应的是应用的实例id
  "serviceType": 1, // 对于主数据是固定的1
  "action": "INSERT"/"MODIFY"/"DELETE"/"PUBLISH"/"ARCHIVE",
  "data": [
    [
      "jdsi",
      "sdf",
      "",
      23,
      null,
      "使用中",
      "",
      "23",
      "待发布",
      "0"
    ]
  ],
  "dataId": 660,
  "masterDataName": "生产设备",
  "nodes": [
    "name",
    "asset_num",
    "brevity_code",
    "equipment_model",
    "iot_device_id",
    "usable_condition",
    "description",
    "dap_id_",
    "dap_status_",
    "dap_refcnt_"
  ]
}
```

4.3. 工厂模型集成

企业用户在运营中心中创建工厂模型和工艺路径，工厂模型包括企业所有的工厂、车间、产线、加工中心，加工中心下再指定生产设备，工厂模型参考ISA95中的模型定义：



Source: ISA 95 Part 3 Figure 7 – Typical expanded equipment hierarchy – no changes to original 95 model

工厂模型查询

如果第三方应用业务需要用到工厂模型，可以通过API查询工厂模型或者采用订阅方式获得工厂模型变更通知。工厂模型集成的说明请参看创建工厂模型统一接口

工厂模型的集成可以分为如下几步：

创建工厂模型

在集成工作台或者数字工厂->设置中心-数据中心->工厂建模里面手动创建需要的模型，如下：

名词解释：

发布：发布的目的是确定此条数据不能修改编码，不能删除，数据发布后才能被三方应用使用，也就是只有发布的数据才能通过OPEN API读到。

归档：已发布不想使用的数据可以归档，归档后无法修改无法删除，数据归档后三方应用不能再使用。

查询工厂模型数据

参数解释：

参数名	描述	备注
id	元数据id	唯一id，主键，整个生命周期都不会变化
code	编码	租户维度唯一，发布后无法编辑，
name	名称	-
parentId	父节点id	父亲节点id，例如车间的parentId就是上一级工厂的id
parentType	父亲节点类型	factory-工厂workshop-车间beltline-产线machiningCenter-加工中心technology-工艺路径process-工序step-步骤
flag	条目的发布状态	unpublish-未发布publish-已发布archive-已归档
orderIndex	条目的排序序号	-
attrs	扩展属性	扩展属性包括：属性名称、属性标识、属性值三个主要字段
deps	关联关系	产线可以关联到工艺路径加工中心可以关联到工序，deps主要是从工艺路径和工序维度查到的关联关系
pageId	分面查询时的页号	-
pageSize	分面查询时的页大小	-
appId	应用的实例id（uuid）	-
employeeId	阿里云账号的子账号id	-

工厂模型树可以用 查询工厂模型接口

```
{
  "code": 200,
  "message": "success",
  "localizedMsg": null,
  "data": {
    "factories": [
      {
        "id": 11063,
        "name": "注塑工厂",
        "code": "code_factory",
        "workShops": [
          {
            "id": 184,
            "name": "注塑车间",
            "code": "code_workshop",
            "beltlines": [
              {
                "id": 7382,
                "name": "注塑生产线",
                "code": "code_beltline",
                "machiningCenters": [
                  {
                    "id": 136,
                    "name": "注塑中心",
                    "code": "code_center",
                    "equipments": [
                      {
                        "name": "设备001",
                        "id": 22,
                        "code": "code-01",
                      }
                    ]
                  }
                ]
              }
            ]
          }
        ]
      }
    ],
    "page": {
      "pageid": 1,
      "pageSize": 1500,
      "total": 1
    }
  },
  "success": true
}
```


工厂、车间、产线、加工中心每一级都可以按照id或者code来查询指定条目的详细信息，具体数据格式：

```
{
  "code": 200,
  "message": "success",
  "localizedMsg": null,
  "data": {
    "desc": "这是个工厂",
    "createTime": "2019-08-22 21:58:13",
    "modifyTime": "2019-10-15 19:49:58",
    "orderIndex": 11063,
    "id": 11063,
    "name": "注塑工厂",
    "code": "code_factory",
    "flag": "unpublish",
    "attrs": [
      {
        "enName": "factory_address",
        "value": "深圳南山区",
        "createTime": "2019-10-15 19:50:40",
        "modifyTime": "2019-10-15 19:50:40",
        "parentId": 11063,
        "parentType": "factory",
        "id": 39577,
        "name": "工厂位置",
        "code": null,
        "flag": null
      }
    ]
  },
  "success": true
}
```


可以获取指定加工中心下绑定的设备，具体数据格式：


```
{
  "equipments": [
    {
      "iotDeviceId": "",
      "usableCondition": "新建",
      "assetNum": "equipment2",
      "name": "生产设备2",
      "description": "equipment2",
      "id": 2,
      "equipmentModel": "2",
      "equipmentModelName": "日常模拟用设备型号",
      "brevityCode": "equipment2",
      "parentId": 136,
      "parentType": "machiningCenter"
    }
  ]
}
```


订阅工厂模型数据

除了主动查询工艺路径数据以外，工业应用也可以采用数据订阅的方式获取最新的工艺路径信息。

工厂建模变更通知数据格式：

（1）工厂建模数据格式：

```
{
  "action": "MODIFY",
  "code": "beltline_update_code",
  "id": 7203,
  "name": "beltline_update_name",
  "parentId": 151,
  "parentType": "workshop",
  "tenantId": "xxxxx",
  "type": "beltline"
}
```

参数

名称	类型	描述
action	String	消息动作：INSERT-新增DELETE-删除MODIFY-修改
id	Long	id
name	String	名称
code	String	编码
type	String	类型：factory-工厂workshop-车间beltline-产线machiningCenter-加工中心equipment-设备technology-工艺路径process-工序step-步骤

通过type来区分是什么数据发生了变化，例如step就标识步骤数据发生了变化。

（2）扩展属性变更格式：

```
{
  "action": "INSERT",
  "enName": "attribute_en_name",
  "id": 37969,
  "name": "attribute_name",
  "parentId": 151,
  "parentType": "workshop",
  "tenantId": "xxxxxxxx",
  "type": "attribute",
   "tenantId": "xxxxxxx",
  "value": "attribute_value"
}
```

参数

名称	类型	描述
action	String	消息动作：INSERT-新增DELETE-删除MODIFY-修改
id	Long	属性id
name	String	属性名称
enName	String	属性标识
value	String	属性值
parentId	Long	父节点id
parentType	String	factory-工厂workshop-车间beltline-产线machiningCenter-加工中心technology-工艺路径process-工序step-步骤
type	String	类型：attribute

（3）产线-工艺路径，加工中心-工序关联关系变更通知格式：

```
{
  "action": "INSERT",
  "id": 10120,
  "idType": "name0307",
  "depId": 10121,
  "depIdType": "",
  "depIdName": "",
  "depIdCode": "",
  "depIdDecs": "",
   "tenantId": "xxxxx",
  "type": "dependence"
}
```

参数

名称	类型	描述
action	String	消息动作： INSERT-新增 DELETE-删除 MODIFY-修改
id	Long	工艺或者工序id
idType	String	TECHNOLOGY（工艺）、 PROCESS（工序）
depId	Long	关联产线或者关联加工中心id
depIdType	String	BELTLINE（产线）、 MACHIN_CENTER（加工中心）
depIdName	String	名称
depIdCode	String	编码
depIdDecs	String	描述
type	String	类型：dependence

通过接口创建工厂模型

工业应用获得接口授权后，可以通过开放的API创建工厂模型，API说明文档参看创建工厂模型统一接口。创建工厂模型接口包括创建工厂、车间、产线、加工中心，关联设备，统一用此接口操作。输入参数的示例代码：

```
{
  "code":"xxxx",
  "name": "attribute_name",
  "parentCode": "xxx",
  "type": "workshop",
  "desc": "xxxxxxx"
}
```

通过接口更新工厂模型

工业应用获得接口授权后，可以通过开放的API更新工厂模型，API说明文档参看接口说明。更新工厂模型接口包括更新工厂、车间、产线、加工中心，关联设备的内容以及发布、归档该模型，统一用此接口操作。输入参数的示例代码：

```
{
  "code":"xxxx",
  "name": "name",
  "parentCode": "xxx",
  "type": "workshop",
  "desc": "xxxxxxx",
  "orderIndex":2,
  "flag":"publishSubNode"
}
```

或者

```
{
  "code":"xxxx",
  "name": "name",
  "parentId": "xxx",
  "type": "workshop",
  "desc": "xxxxxxx",
  "orderIndex":2,
  "flag":"publishSubNode"
}
```

通过接口删除工厂模型

工业应用获得接口授权后，可以通过开放的API删除还未发布的工厂模型，API说明文档参看接口说明。删除工厂模型接口包括删除工厂、车间、产线、加工中心，关联设备，统一用此接口操作。输入参数的示例代码：

```
{
  "code":"xxxx",
  "type": "workshop"
}
```

或者

```
{
  "id":"xxxx",
  "type": "workshop"
}
```

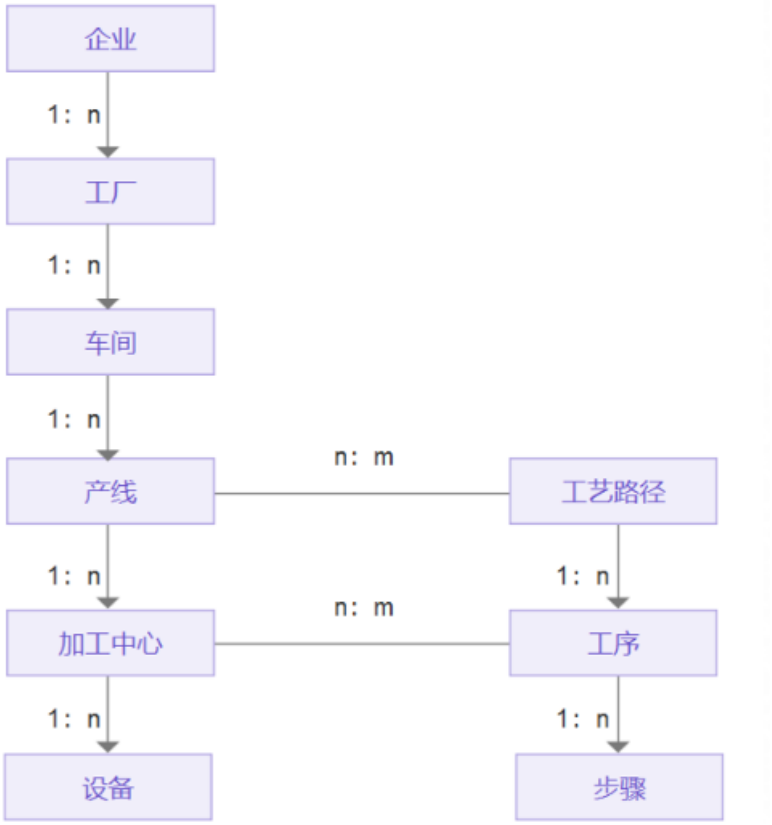
通过接口添加工厂模型属性

工业应用获得接口授权后，可以通过开放的API添加工厂模型属性，API说明文档参看文档说明。输入参数的示例代码：

```
{
  "name": "attribute_name",
  "enName":"属性标识",
  "value":"属性值",
  "parentCode": "xxx",
  "parentType": "workshop",//父节点类型，工厂:factory，车间:workshop，产线:beltline，加工中心:machiningCenter，工艺:technology，工序:process，步骤:step
  "desc": "xxxxxxx"
}
```

4.4. 工艺路径集成

工艺路径是指企业生产某一种物料类型的产品或者半成品时候的加工路线。一条工艺路径包括了多个工序，每一个工序又包括了多个工艺步骤。同一条工艺路径可以在多条产线上执行，工序可以在多个加工中心去执行。工艺步骤和工厂模型的领域模型如下：



查询工艺路径

如果第三方应用业务需要用到工艺路径，可以通过API查询工艺路径或者采用订阅方式获得工艺路径变更通知。工艺路径集成的说明请参看接口说明

工艺路径的集成可以分为如下几步：

工艺路径树创建

在集成工作台或者数字工厂->设置中心-数据中心->工艺路径里面手动创建需要的模型，如下：

注意：发布后的数据才能通过OPEN API读到。

查询工艺路径树

工艺路径树可以用 接口查询


```
{
  "code": 200,
  "message": "success",
  "localizedMsg": null,
  "data": {
    "technologies": [
      {
        "id": 12,
        "name": "注塑工艺",
        "code": "code_tech",
        "processes": [
          {
            "id": 8126,
            "name": "注塑工序一",
            "code": "code1",
            "steps": [
              {
                "id": 8842,
                "name": "注塑步骤一",
                "code": "code_step1"
              },
              {
                "id": 9360,
                "name": "注塑步骤二",
                "code": "code_step2"
              }
            ]
          }
        ]
      }
    ],
    "page": {
      "pageid": 1,
      "pageSize": 1500,
      "total": 1
    }
  },
  "success": true
}
```


工艺路径、工序、步骤每一级都可以按照id或者code来获取指定条目的详细信息，具体数据格式：

```
{
  "code": 200,
  "message": "success",
  "localizedMsg": null,
  "data": {
    "desc": "这是一个注塑工艺",
    "createTime": "2019-02-19 15:43:27",
    "modifyTime": "2019-10-15 20:12:10",
    "parentId": null,
    "parentType": null,
    "orderIndex": 12,
    "id": 12,
    "name": "注塑工艺",
    "code": "code_tech",
    "flag": "publish",
    "attrs": [
      {
        "enName": "phoneNumber",
        "value": "xxxxxx",
        "createTime": "2019-10-15 20:12:53",
        "modifyTime": "2019-10-15 20:12:53",
        "parentId": 12,
        "parentType": "technology",
        "id": 39578,
        "name": "工艺负责人电话",
        "code": null,
        "flag": null
      }
    ],
    "deps": [
      {
        "id": 7382,
        "name": "注塑生产线",
        "code": "code_beltline",
        "flag": null
      }
    ]
  },
  "success": true
}
```


订阅工艺路径数据

除了主动查询工艺路径数据以外，工业应用也可以采用http2方式数据推送，注册订阅的消息类型的方式获取最新的工艺路径信息。

工艺路径变更通知数据格式：

（1）工艺路径数据格式：

```
{
  "action": "MODIFY",
  "code": "beltline_update_code",
  "id": 7203,
  "name": "beltline_update_name",
  "parentId": 151,
  "parentType": "workshop",
  "tenantId": "xxxxx",
  "type": "beltline"
}
```

参数

名称	类型	描述
action	String	消息动作：INSERT-新增DELETE-删除MODIFY-修改
id	Long	id
name	String	名称
code	String	编码
type	String	类型：factory-工厂workshop-车间beltline-产线machiningCenter-加工中心equipment-设备technology-工艺路径process-工序step-步骤

通过type来区分是什么数据发生了变化，例如step就标识步骤数据发生了变化。

（2）扩展属性变更格式：

```
{
  "action": "INSERT",
  "enName": "attribute_en_name",
  "id": 37969,
  "name": "attribute_name",
  "parentId": 151,
  "parentType": "workshop",
  "tenantId": "xxxxxxxx",
  "type": "attribute",
   "tenantId": "xxxxxxx",
  "value": "attribute_value"
}
```

参数

名称	类型	描述
action	String	消息动作：INSERT-新增DELETE-删除MODIFY-修改

名称	类型	描述
id	Long	属性id
name	String	属性名称
enName	String	属性标识
value	String	属性值
parentId	Long	父节点id
parentType	String	factory-工厂workshop-车间 beltline-产线machiningCenter-加工中心technology-工艺路径 process-工序step-步骤
type	String	类型：attribute

（3）产线-工艺路径，加工中心-工序关联关系变更通知格式：

```
{
  "action": "INSERT",
  "id": 10120,
  "idType": "name0307",
  "depId": 10121,
  "depIdType": "",
  "depIdName": "",
  "depIdCode": "",
  "depIdDecs": "",
   "tenantId": "xxxxx",
  "type": "dependence"
}
```

参数

名称	类型	描述
action	String	消息动作： INSERT-新增 DELETE-删除 MODIFY-修改
id	Long	工艺或者工序id
idType	String	TECHNOLOGY（工艺）、 PROCESS（工序）
depId	Long	关联产线或者关联加工中心id
depIdType	String	BELTLINE（产线）、 MACHIN_CENTER（加工中心）
depIdName	String	名称
depIdCode	String	编码
depIdDecs	String	描述
type	String	类型：dependence

通过接口创建工艺路径

工业应用获得接口授权后，可以通过开放的API创建工艺路径，API说明文档参看接口说明。创建工艺路径模型接口包括创建工艺路径、工序、步骤统一用此接口操作。输入参数的示例代码：

```
{
  "code":"xxxx",
  "name": "attribute_name",
  "parentCode": "xxx",
  "type": "process",
  "desc": "xxxxxxx"
}
```

通过接口更新工艺路径

工业应用获得接口授权后，可以通过开放的API更新工艺路径，API说明文档参看接口说明。更新工艺路径模型的接口包括创建工艺路径、工序、步骤统一用此接口操作。输入参数的示例代码：

```
{
  "code":"xxxx",
  "name": "name",
  "parentCode": "xxx",//parentId、parentCode二选一
  "type": "process",
  "desc": "xxxxxxx",
  "orderIndex":2,
  "flag":"publishSubNode"
}
```

或者

```
{
  "code":"xxxx",
  "name": "name",
  "parentId": "xxx",
  "type": "process",
  "desc": "xxxxxxx",
  "orderIndex":2,
  "flag":"publishSubNode"
}
```

通过接口删除工艺路径

工业应用获得接口授权后，可以通过开放的API删除还未发布的工艺路径，API说明文档参看接口说明。删除工艺路径接口删除包括工艺路径、工序、步骤统一用此接口操作，统一用此接口操作。输入参数的示例代码：

```
{
  "code":"xxxx",
  "type": "technology"
}
```

或者

```
{
  "id":"xxxx",
  "type": "technology"
}
```

通过接口添加工艺路径属性

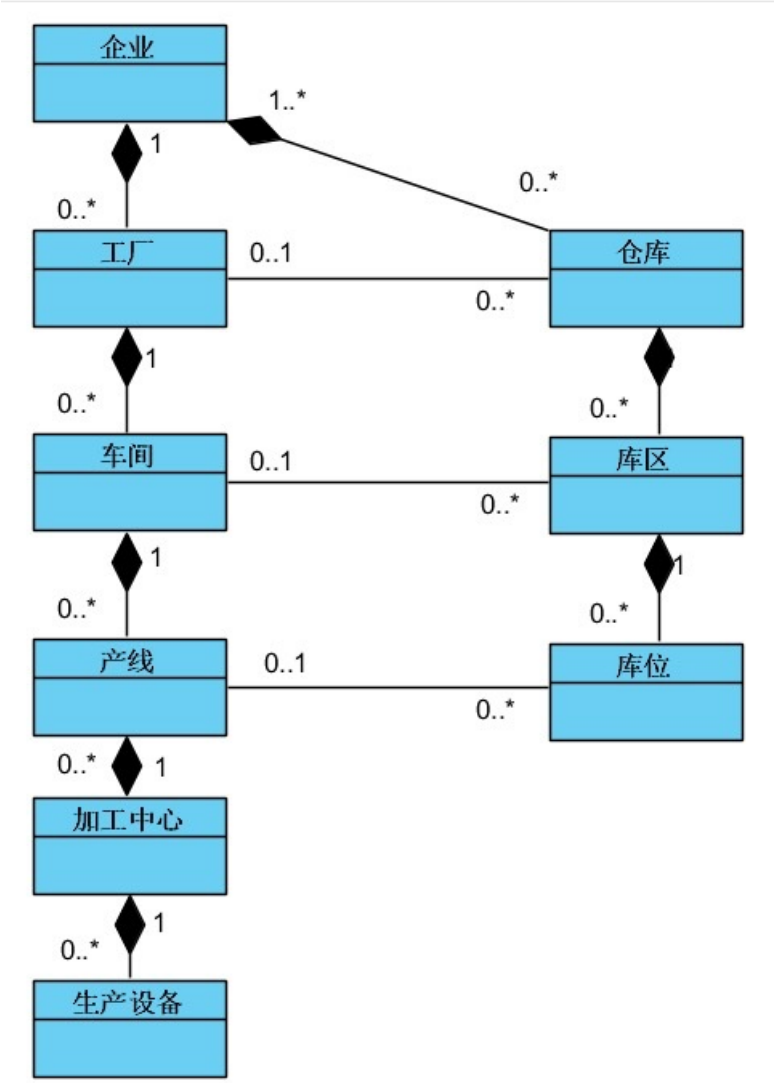
工业应用获得接口授权后，可以通过开放的API添加工厂模型属性，API说明文档参看接口说明。输入参数的示例代码：

```
{
  "name": "attribute_name",
  "enName":"属性标识",
  "value":"属性值",
  "parentCode": "xxx",
  "parentType": "technology",//父节点类型，工厂:factory，车间:workshop，产线:beltline，加工中心:machiningCenter，工艺:technology，工序:process，步骤:step
  "desc": "xxxxxxx"
}
```

4.5. 库存地点集成

库存模型是对企业仓库信息的描述，一个仓库可能包含多个库区，一个库区可能包含多个库位。

库存模型与工厂模型的关系如下图：



如果第三方应用业务需要用到库存地点，可以通过API查询库存地点或者采用订阅方式获得库存地点变更通知。

发布库存地点

提供给三方应用通过API发布库存地点，包括仓库、库区和库位。

仓库

参看接口说明

支持带属性的批量创建

```
{
  "warehouses": [
    {
      "code": "warehouse",
      "name": "仓库",
      "description": "仓库描述",
      "factoryCode": "factoryCode", // 如果仓库有关联工厂，这个字段填入工厂编码
      "attributes": [
        {
          "name": "仓库属性",
          "enName": "attribute",
          "value": "value"
        }
      ]
    }
  ]
}
```

返回id/code的映射

```
{
  "code": 200,
  "data": [
    {
      "id": 158,
      "code": "warehouse"
    }
  ]
}
```

库区

参看接口说明

支持带属性的批量创建

```
{
  "warehouseAreas": [
    {
      "code": "warehouseArea",
      "name": "库区",
      "description": "库区描述",
      "parentCode": "warehouse",
      "workshopCode": "workshopCode", // 如果库区有关联车间，这个字段填入车间编码
      "attributes": [
        {
          "name": "库区属性",
          "enName": "attribute",
          "value": "value"
        }
      ]
    }
  ]
}
```

返回id/code的映射

```
{
  "code": 200,
  "data": [
    {
      "id": 169,
      "code": "warehouseArea"
    }
  ]
}
```

库位

参看接口说明

支持带属性的批量创建

```
{
  "locations": [
    {
      "code": "location",
      "name": "库位",
      "description": "库位描述",
      "parentCode": "location",
      "beltlineCode": "beltlineCode", // 如果库位有关联产线，这个字段填入产线编码
      "attributes": [
        {
          "name": "库位属性",
          "enName": "attribute",
          "value": "value"
        }
      ]
    }
  ]
}
```

返回id/code的映射

```
{
  "code": 200,
  "data": [
    {
      "id": 160,
      "code": "location"
    }
  ]
}
```

查询库存模型数据

参看获取仓库模型的树结构

查询仓库树的数据格式如下

```
{
  "code": 200,
  "data": {
    "warehouses": [
      {
        "id": 158,
        "code": "warehouse",
        "name": "仓库",
        "warehouseAreas": [
          {
            "id": 169,
            "code": "warehouseArea",
            "name": "库区",
            "locations": [
              {
                "code": "location",
                "name": "库位",
                "id": 160
              }
            ]
          }
        ]
      }
    ]
  },
  "id": "8c9063a638044f92a0657bccaf476fea"
}
```


仓库、库区、库位每一级的模型都可以用单独的api查询，可以指定code查询指定的模型或者不指定查询模型列表。

仓库

参看查询仓库信息，可以指code查询

```
{
  "code": 200,
  "data": {
    "warehouses": [
      {
        "code": "warehouse",
        "name": "仓库",
        "attributes": [
          {
            "name": "仓库属性",
            "enName": "attribute",
            "value": "value"
          }
        ],
        "id": 158
      }
    ]
  },
  "id": "559b2848e20f49328205c9bdfd49f4c1"
}
```


库区

参看获取库区列表信息，可以指定code查询


```
{
  "code": 200,
  "data": {
    "warehouseAreas": [
      {
        "code": "warehouseArea",
        "parentCode": "warehouse",
        "name": "库区",
        "attributes": [
          {
            "enName": "attribute",
            "name": "库区属性",
            "value": "value"
          }
        ],
        "id": 169
      }
    ]
  },
  "id": "5640c14ff94346c9919d46b559992dee"
}
```


库位

参看获取库位信息，可以指定code查询

```
{
  "code": 200,
  "data": {
    "locations": [
      {
        "code": "location",
        "name": "库位",
        "attributes": [
          {
            "enName": "attribute",
            "name": "库位属性",
            "value": "value"
          }
        ],
        "id": 160
      }
    ]
  },
  "id": "fbb6f62647164488bc8f5788c5b4539c"
}
```


消息通知

消息格式

```
{
  "serviceType": 8,
  "action":"PUBLISH",
  "warehouses": [
    {
      "code": "testWarehouse2",
      "name": "测试仓库2",
      "id": 163
    }
  ],
  "warehouseAreas": [
    {
      "parentName": "测试仓库2",
      "code": "testReservoir",
      "parentCode": "testWarehouse2",
      "name": "测试库区",
      "id": 174
    }
  ],
  "locations": [
    {
      "parentName": "测试库区",
      "code": "testLocation",
      "parentCode": "testReservoir",
      "name": "测试库位",
      "id": 164
    }
  ]
}
```

5.应用服务集成

工业应用可以与数字工厂提供的官方基础服务进行集成，最终为数字工厂用户提供完整的业务功能。

5.1. 获取工厂日历数据

用户在数字工厂的工厂日历中进行排班和计划停产后，可以通过接口提供给第三方应用获取工厂日历的数据。

查询班次列表信息

通过接口可以查到用户设置的所有班次信息，分页查询工厂日历班次列表信息请参看。

返回结果示例：

```
{
    "id": "4de2c367-c1db-417c-aa15-8c585e595d92",
    "code": 200,
    "message": null,
    "localizedMsg": null,
    "data": {
        "total": 1,
        "data": [{
          "id": 1,
          "ftyModelId": 123,
          "ftyModelValue": 1,
          "ftyModelName": "测试的工厂",
          "name": "日班",
          "beginTime": "08:00:00",
          "endTime": "16:00:00",
          "beginEffectiveDate": "2019-10-01",
          "description": "日班的备注"
        }]
    }
}
```

获取指定时间点生效的班次

通过接口可以查到指定时间点，用户设置的符合条件班次信息，也可以指定查询条件为工厂、车间、产线或者加工中心。获取指定时间点生效的工厂日历班次请参看。

返回结果示例：

```
{
    "id": "4de2c367-c1db-417c-aa15-8c585e595d92",
    "code": 200,
    "message": null,
    "localizedMsg": null,
    "data": {
        "id": 1,
        "tenantId": "xxxx",
        "ftyModelId": 123,
        "ftyModelValue": 3,
        "name": "日班",
        "beginTime": "08:00:00",
        "endTime": "16:00:00",
        "beginEffectiveDate": "2019-10-01",
        "nonProdTimeMgtDTOList": [{
          "id": 1,
          "shiftMgtId": 1,
          "type": 1,
          "beginTime": "13:00:00",
          "endTime": "13:30:00"
        }]
      }
}
```


根据ID查询班次记录

查询班次ID后，可以指定具体的班次ID来查询该班次的详细时间安排，根据给定的班次管理ID获取工厂日期班次记录请参看。

返回结果示例：


```
{
    "id": "4de2c367-c1db-417c-aa15-8c585e595d92",
    "code": 200,
    "message": null,
    "localizedMsg": null,
    "data": {
        "id": 1,
        "ftyModelId": 123,
        "ftyModelValue": 3,
        "name": "日班",
        "beginTime": "08:00:00",
        "endTime": "16:00:00",
        "beginEffectiveDate": "2019-10-01",
        "nonProdTimeMgtDTOList": [{
          "id": 1,
          "shiftMgtId": 1,
          "type": 1,
          "beginTime": "13:00:00",
          "endTime": "13:30:00"
        }]
      }
}
```

查询计划停产信息

通过接口可以查到用户设置的所有计划停产信息，根据给定的ID获取工厂日历计划停产管理记录请参看。

返回结果示例：


```
{
    "id": "4de2c367-c1db-417c-aa15-8c585e595d92",
    "code": 200,
    "message": null,
    "localizedMsg": null,
    "data": {
        "total": 1,
        "data": [{
          "id": 1,
          "ftyModelId": 123,
          "ftyModelValue": 1,
          "ftyModelName": "测试的工厂",
          "name": "周末",
          "type": 2,
          "dayOfWeek": 7,
          "beginTime": "08:00:00",
          "endTime": "16:00:00",
          "description": "周日休息一天"
        }]
      }
}
```


查询指定时间点是否计划停产

通过接口可以查到指定时间点，指定工厂、车间、产线或者加工中心是否计划停产。判断指定时间点工厂日历是否安排了计划停产请参看。

代码示例：


```
{
    "id": "4de2c367-c1db-417c-aa15-8c585e595d92",
    "code": 200,
    "message": null,
    "localizedMsg": null,
    "data": true
}
```


查询指定日期的工厂日历配置

通过接口可以查到指定日期的工厂日历详细信息，也可以指定查询条件为工厂、车间、产线或者加工中心。查询指定日期有效的工厂日历配置，包括班次管理列表及计划停产管理列表请参看。

返回结果示例：

```
{
    "id": "4de2c367-c1db-417c-aa15-8c585e595d92",
    "code": 200,
    "message": null,
    "localizedMsg": null,
    "data": {
        "plannedShutdownMgtDTOList": [{
          "id": 1,
          "ftyModelId": 123,
          "ftyModelValue": 1,
          "ftyModelName": "测试的工厂",
          "name": "周末",
          "type": 2,
          "dayOfWeek": 7,
          "beginTime": "08:00:00",
          "endTime": "16:00:00",
          "description": "周日休息一天"
        }],
        "shiftMgtDTOList": [{
          "id": 1,
          "ftyModelId": 123,
          "ftyModelValue": 1,
          "ftyModelName": "测试的工厂",
          "name": "日班",
          "beginTime": "08:00:00",
          "endTime": "16:00:00",
          "beginEffectiveDate": "2019-10-01",
          "description": "日班描述"
        }]
      }
}
```


5.2. OEE

数字工厂企业用户开通并使用数字工厂的OEE后，第三方工业应用可以通过OEE提供的服务来同步生产设备的状态变化也可以获得设备综合效率的计算结果。

设置生产设备状态

生产设备的设备状态可以通过关联的物联网设备通过属性同步设备状态到数字工厂来计算设备综合效率，设置参看设置生产设备状态。

获得授权的第三方工业应用也可以通过接口设置设备状态，接口文档参看设置OEE设备在指定时间段内的状态，包括故障原因描述，开始及结束时间。

获得设备综合效率计算结果

OEE应用实时计算设备综合效率的结果，可以通过设置获得设备综合效率计算结果写入到指定的物联网设备属性中。

获得授权的第三方工业应用也可以通过接口获取设备综合效率计算结果，接口文档参看查询指定设备在“查询日期（天）”的设备OEE计算数值。查询中可以指定“查询日期”的班次名，如果不指定班次名称，就是该日的设备OEE的总和值。

5.3. 生产过程追溯

数字工厂企业用户开通并使用数字工厂的生产过程追溯后，第三方工业应用可以通过生产过程追溯应用提供的服务来同步生产过程的产质耗数据。

生产报工

工业应用可以通过生产报工接口上报和调整生产结果信息，主要信息包括在哪个工序、哪个设备上半成品或者成品的最终产量。上报数据的维度可以按照某一个批次，或者生产计划单号，甚至是成品的单体序号。

过程质检数据

质检结果包括质检记录和质检记录中的质检判定结果两个部分内容。质检记录包括质量检验的对象（生产报工中的唯一标识）、质检时间、质检人以及检验数量；质检判定结果为质检记录中的产品等级、数量以及是否合格等信息。

质检记录

通过接口可以创建质检记录，更新质检记录。

并且提供了接口给授权的工业应用查询质检记录以及根据ID查询质检记录。

质检判定结果

通过接口可以创建质检结果，更新质检结果以及删除质检结果。

并且提供了接口给授权的工业应用查询质检结果以及根据ID查询质检结果。

5.4. 获取经营数据

在获得企业用户授权后，第三应用可以通过接口获取经营驾驶舱中指标管理定义的经营指标数据和企业资质数据。

查询指标

通过接口可以查到应用可公开查询的经营指标，接口说明。

返回结果示例：

```
{
    "id": "4de2c367-c1db-417c-aa15-8c585e595d92",
    "code": 200,
    "message": null,
    "localizedMsg": null,
    "data": [
        {
            "id": 1001,
            "name": "产能利用率",
            "precision":1,
            "description":"对生产能力的利用率，衡量资源利用效率的指标，是增加投资的依据(是否需要购买新机器、产线或者招聘人员)",
            "showPercentage":true,
            "comparisonType": "YEAR_ON_YEAR",
            "periodType": "WEEK",
            "showPercentage": true
        }]
    }
}
```


查询企业资质数据

企业通过开放资质数据给第三方应用后，可以通过接口查询到企业资质管理的营业执照信息、税务登记信息、营业办公信息、研发能力、质量管控能力、企业自有品牌以及其它资质能力。接口文档请参考从工业平台中查询企业的营业执照信息、从工业平台中查询企业的税务登记信息、从工业平台中查询企业的营业办公信息、从工业平台中查询企业的研发能力、从工业平台中查询企业的质量管控能力、从工业平台中查询企业的自有品牌以及从工业平台中查询企业的其它资质能力。

5.5. 产品服务

数字工厂企业用户开通并使用数字工厂的产品管理后，第三方工业应用可以通过订单管理应用提供的服务来同步产品数据。

产品管理

应用可以通过服务接口，同步企业的产品信息，新建和更新接口使用参见创建及修改产品信息接口。

可以通过产品发布产品信息接口把产品信息发布给其他应用使用。

产品信息查询

应用获取产品信息可以通过查询产品接口，接口使用参见获取产品信息接口。

5.6. 订单服务

数字工厂企业用户开通并使用数字工厂的订单管理后，第三方工业应用可以通过订单管理应用提供的服务来同步订单数据。

订单管理

应用可以通过服务接口，同步企业的采购订单，新建和更新接口使用参见一方订单服务，采购单同步接口

订单数据获取

应用获取订单可以通过查询订单接口，接口使用参见工业订单服务查询列表接口。

除了主动查询，还可以通过集成工作概述中的数据变更通知订阅说明了接口的使用方式，其中serviceType参数设置为6，表示订单数据通知。

当订单发生变更时，消息订阅端会收到两个字段：“动作”+“订单信息”，动作参数如下：

```
新增订单：1
关闭订单：2
更新订单：3    //钉钉版更新订单该值为4
```

新增订单通知格式

```
{
  "serviceType":6,
  "orderInfo":"{\"annexes\":{},\"bizSource\":0,\"companyName\":\"dfgdsg\",\"contractType\":
1,\"dueTime\":1593964800000,\"orderId\":\"117481593591492727\",\"orderProductDTOs\":[{\"amo
unt\":\"34\",\"materialType\":\"10\",\"materialTypeName\":\"腹胀\",\"measureUnit\":\"件\",\"
price\":\"34\",\"processFlow\":\"322\",\"processFlowName\":\"服装加工\",\"productCode\":\"kk
k\",\"productId\":714,\"productModel\":\"fsda\",\"productName\":\"jjj\",\"productStatus\":1
,\"source\":\"OC\",\"version\":0,\"versionStatus\":1}],\"orderTime\":1593591493000,\"produc
eStatus\":0,\"recipientAddress\":\"fdgsgsdg\",\"recipientCity\":\"自贡\",\"recipientDistrict
\":\"荣县\",\"recipientName\":\"dfsgs\",\"recipientPhone\":\"1331231****\",\"recipientProvin
ce\":\"四川\",\"totalAmount\":\"434\",\"totalPrice\":\"34535\",\"workingType\":1}",
  "action":1
}
```

关闭订单通知格式

```
{
  "serviceType":6,
  "orderInfo":"{\"annexes\":{},\"bizSource\":0,\"companyName\":\"dfgdsg\",\"contractType\":
1,\"dueTime\":1593964800000,\"orderId\":\"117481593591492727\",\"orderProductDTOs\":[{\"amo
unt\":\"34\",\"materialType\":\"10\",\"materialTypeName\":\"腹胀\",\"measureUnit\":\"件\",\"
price\":\"34\",\"processFlow\":\"322\",\"processFlowName\":\"服装加工\",\"productCode\":\"kk
k\",\"productId\":714,\"productModel\":\"fsda\",\"productName\":\"jjj\",\"productStatus\":1
,\"source\":\"OC\",\"version\":0,\"versionStatus\":1}],\"orderTime\":1593591493000,\"produc
eStatus\":4,\"recipientAddress\":\"fdgsgsdg\",\"recipientCity\":\"自贡\",\"recipientDistrict
\":\"荣县\",\"recipientName\":\"dfsgs\",\"recipientPhone\":\"1331231****\",\"recipientProvin
ce\":\"四川\",\"totalAmount\":\"434\",\"totalPrice\":\"34535\",\"workingType\":1}",
  "action":2
}
```

6.业务数据集成

6.1. 质量缺陷分析

在数字工厂中应用开通经营驾驶舱应用后，可以在标准经营分析中查看质量分析结果。需要应用通过以下接口上报业务数据：

生产报工

通过生产报工接口上报，主要信息包括在哪个工序、哪个设备上半成品或者成品的最终产量。上报数据的维度可以按照某一个批次，或者生产计划单号，甚至是成品的单体序号。

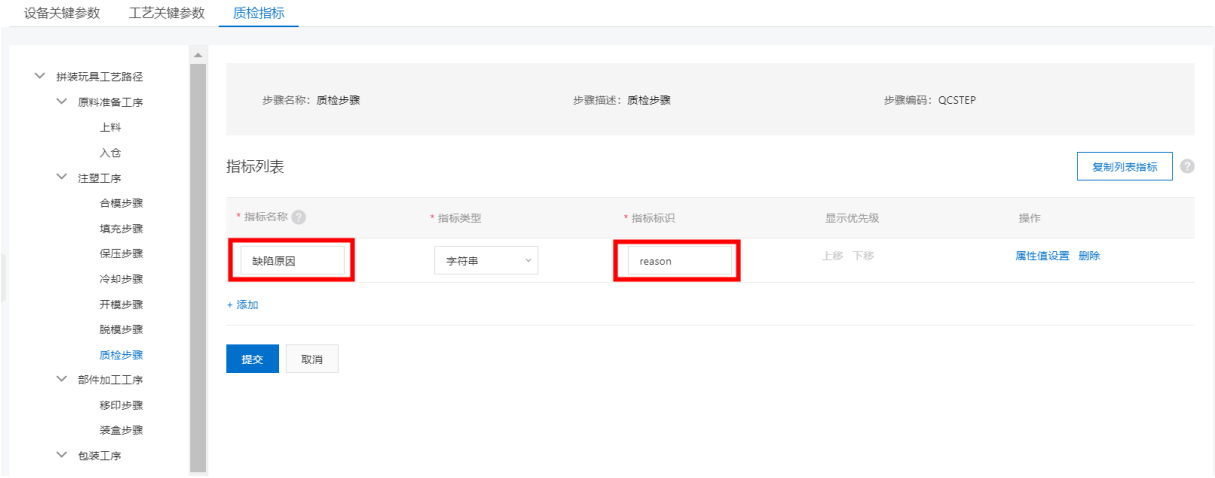
过程质检数据

质检结果包括质检记录和质检记录中的质检判定结果两个部分内容。质检记录包括质量检验的对象（生产报工中的唯一标识）、质检时间、质检人以及检验数量；质检判定结果为质检记录中的产品等级、数量以及是否合格等信息。

质检记录

通过接口可以创建质检记录，更新质检记录以及删除质检记录。

注：如果要在标准经营分析的质量缺陷分析中按照不同的缺陷原因进行分析，需要在生产过程追溯的应用配置中对需要上报质检结果的步骤增加一个质检指标，该指标的名称为**缺陷原因**，指标标识为**reason**，需要用这个质检指标来上报缺陷原因。



质检判定结果

通过接口可以创建质检结果，更新质检结果以及删除质检结果

6.2. 成品率分析

在数字工厂中应用开通经营驾驶舱应用后，可以在标准经营分析中查看成品率分析结果。需要应用通过以下接口上报业务数据：

生产报工

通过生产报工接口上报，主要信息包括在哪个工序、哪个设备上半成品或者成品的最终产量。上报数据的维度可以按照某一个批次，或者生产计划单号，甚至是成品的单体序号。

过程质检数据

质检结果包括质检记录和质检记录中的质检判定结果两个部分内容。质检记录包括质量检验的对象（生产报工中的唯一标识）、质检时间、质检人以及检验数量；质检判定结果为质检记录中的产品等级、数量以及是否合格等信息。

质检记录

通过接口可以创建质检记录，更新质检记录以及删除质检记录。

质检判定结果

通过接口可以创建质检结果，更新质检结果以及删除质检结果

6.3. 人均产量（能）分析

在数字工厂中应用开通经营驾驶舱应用后，可以在标准经营分析中查看人均产量（能）分析结果。需要应用通过以下接口上报业务数据：

生产成品入库

通过生产成品入库API, 记录入库数量和金额上报每天、每个工厂成品入库的数量以及金额（可选）。

人员出勤

通过上报人员出勤基本信息上报每天、每个工厂出勤人数、天数以及薪酬总数（可选）。

6.4. 设备效率分析

在数字工厂中应用开通经营驾驶舱应用后，可以在标准经营分析中查看设备效率结果。需要应用通过以下接口上报业务数据：

设备综合效率

通过上报设备综合效率数据上报每天、每台设备的计划开机、实际开机以及故障时长。

6.5. 能耗分析

在数字工厂中应用开通经营驾驶舱应用后，可以在标准经营分析中查看能耗分析结果。需要应用通过以下接口上报业务数据：

生产成品入库

通过生产成品入库API, 记录入库数量和金额上报每天、每个工厂成品入库的数量以及金额。

能源数据

通过能耗接口上报每天、每个工厂、每种能源介质的消耗量和耗用金额。

6.6. 人员出勤分析

在数字工厂中应用开通经营驾驶舱应用后，可以在标准经营分析中查看人员出勤分析结果。需要应用通过以下接口上报业务数据：

人员出勤

通过上报人员出勤基本信息上报每天、每个工厂出勤人数、天数以及薪酬总数（可选）。