**工业产线数据采集与管理系统**

**2-项目开发文档**

**1.0-2021.3.7**

分工说明

|  |  |
| --- | --- |
| 姓名 | 本文档中主要承担的工作内容 |
| 叶静波 | 后端开发、数据上云 |
| 刘子航 | 前端开发、3D建模 |

版本变更历史

|  |  |  |
| --- | --- | --- |
| 版本 | 提交日期 | 版本说明 |
| 1.0 | 2021.3.7 | 写出详细设计和具体开发过程 |

目录

[1. 数据库设计 4](#_Toc23234)

[1.1 本地MySQL 4](#_Toc9205)

[1.2 本地Redis缓存 6](#_Toc13577)

[1.3 本地InfluxDB 7](#_Toc27578)

[1.4 kafka消息队列 7](#_Toc14858)

[1.5 云服务器数据库 7](#_Toc10075)

[2. 体系结构设计 8](#_Toc2134)

[2.1 软件体系结构 8](#_Toc5516)

[2.2 各模块之间的关系 9](#_Toc7337)

[3. 接口设计 9](#_Toc32135)

[3.1 支持多协议接口设计 9](#_Toc12537)

[3.2 数据库访问接口 10](#_Toc10861)

[3.3 云服务器数据访问接口 10](#_Toc228)

[4. 详细设计 10](#_Toc31352)

[4.1 OPC UA服务器模块 10](#_Toc1667)

[4.2 集成多协议模块 13](#_Toc26876)

[4.3 数据库模块 13](#_Toc6901)

[4.4 数据上云模块 15](#_Toc29009)

[4.5 集群管理模块 15](#_Toc27)

[4.6 索引优化设计模块 16](#_Toc23058)

# 数据库设计

## 本地MySQL

数据库名称：local\_db

默认端口：127.0.0.1:3306

用户名/密码：root/123456

存放关系数据，根据id来区分该数据属于几号工位的什么操作或状态。

其中一号工位具有的状态和操作分别有：传感器状态（一号工位为1）、1/2号托盘进料/退料到位、传送带暂停、机器人关节信息、1/2号托盘累积抓取数量、1/2号托盘当前工件数量、1/2号托盘工件位姿信息、1/2号托盘图片。

二号工位具有的状态和操作分别有：当前通过的工件编号、当前工件质量情况（缺角、划痕、标签错误）、工位工件的统计（过检数、良品数量、不良品数）、良品/不良品图片。

三号工位具有的状态和操作分别有：装配动作（上盖抓取、锁轴垫片抓取、螺旋桨抓取、上板抓取、螺钉一/二抓取）、装配结果（上盖安装、锁轴垫片安装、螺旋桨安装、上板安装、螺钉一/二安装）、螺旋桨整体装配个数、装配动作/结果图片。

四号工位具有的状态和操作分别有：检测所耗时间、推力、电流等数据、检测进度。

需要构造的数据库表有：产线机器基本信息，主要是静态数据，机器型号等；图片数据表，用来存储图片数据；另外的数据库表依据工位编号建立四个数据库，分别存储每个工位的数据，数据是以时间为key存储的数据。对于具体操作和数值，采用Json字段以字典的形式存储。这样分表的形式，缓解了数据库写压力，也可以提高数据库读写性能。

* 将Django默认数据库换为Mysql，创建数据库，数据库取名为local\_db，并设置字符集为utf-8：CREATE DATABASE local\_db CHARACTER SET utf8;
* 配置settings.py文件，配置Mysql数据库引擎
* 安装mysqlclient库：pip install mysqlclient
* 执行同步操作python manage.py makemigrations, python manage.py migrate将数据更新到MySQL
* python manage.py  (查看命令列表)
* 使用数据库mysql> use local\_db;
* 查看表mysql> show tables;

settings.X\_1 = {

"id": id,

"name": "传感器状态",

"value": data["1"],

"update\_time": data["update\_time"]

}

settings.X\_C\_1 = {

"id": id,

"name": "托盘一退料到位",

"value": data["1\_1"],

"update\_time": data["update\_time"]

}

x\_7\_x = {

"name": "x",

"value": settings.X\_7["value"]["X"]

}

g1 = {

"id": "16",

"name": "机器人关节信息",

"value": robot1,

"update\_time": "0-0-0 00:00:00"

}

x\_7\_y = {

"name": "y",

"value": settings.X\_7["value"]["Y"]

}

x\_7\_angle = {

"name": "angle",

"value": settings.X\_7["value"]["Angle"]

}

机器静态信息表

工位具体信息表

图片信息表

下图是表的信息：

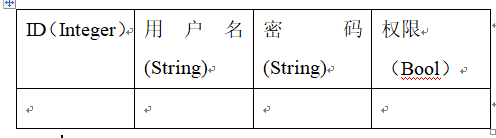


表3.1 用户信息数据库设计

## 本地Redis缓存

语音指令集储存了指令和其对应的用户语音识别关键词，主要用于语音识别模块。每条指令也有唯一的NO，图3.2表示了该数据库的设计。

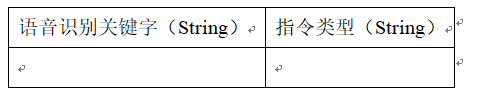


表3.2 语音指令集数据库设计

其中语音识别关键字是一个英文单词，并且在语音识别模块的关键字列表中。一个语音识别关键字只能对应一个指令类型。

## 本地InfluxDB

根据时间为key进行存储。

表3.3 互联网天气信息数据表

## 本地单机kafka消息队列

连接到服务器

send发送数据

轮询方式获取数据，持久化到local\_db数据库

## 云服务器数据库

* 云端kafka集群
* 云端influxDB集群
* 云端Redis
* 云端关系数据库Mysql

用户在使用机器人系统时，可新建任务操作，机器人可根据相应的指令执行任务；数据库系统对已创建的历史任务进行保留，方便用户查看历史。任务表有IS\_FINISHED标志位表示任务是否已经完成了，TIME字段用来表示创建的时间；主键是任务表的NO, 外键是用户表(O\_U\_NO) 和指令表(O\_I\_NO)的NO。

# 系统架构设计

## 系统整体技术架构

总体设计采用调用返回体系结构，从ROS机器人分析并执行指令的角度出发，逐步细化每一个子功能。系统的软件体系结构如图4.1。

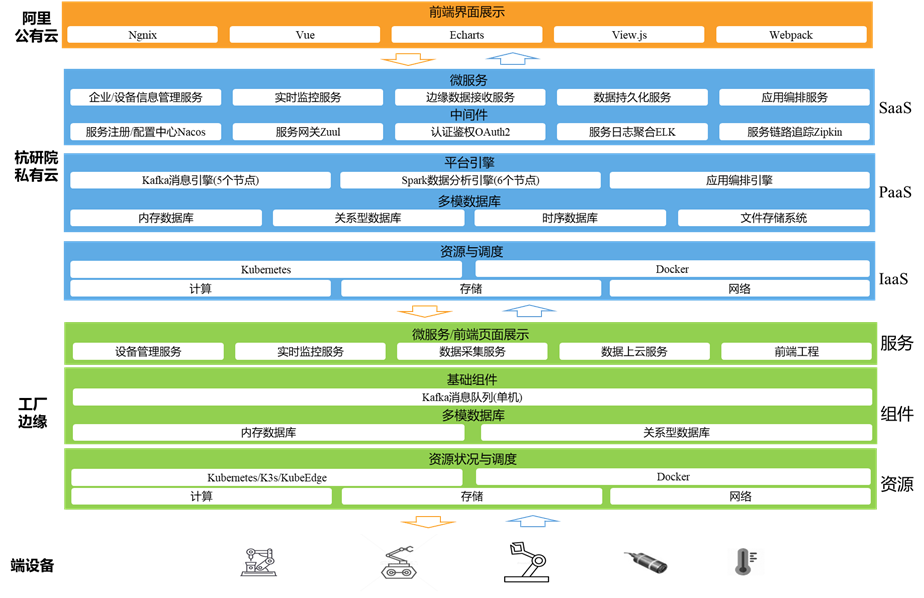
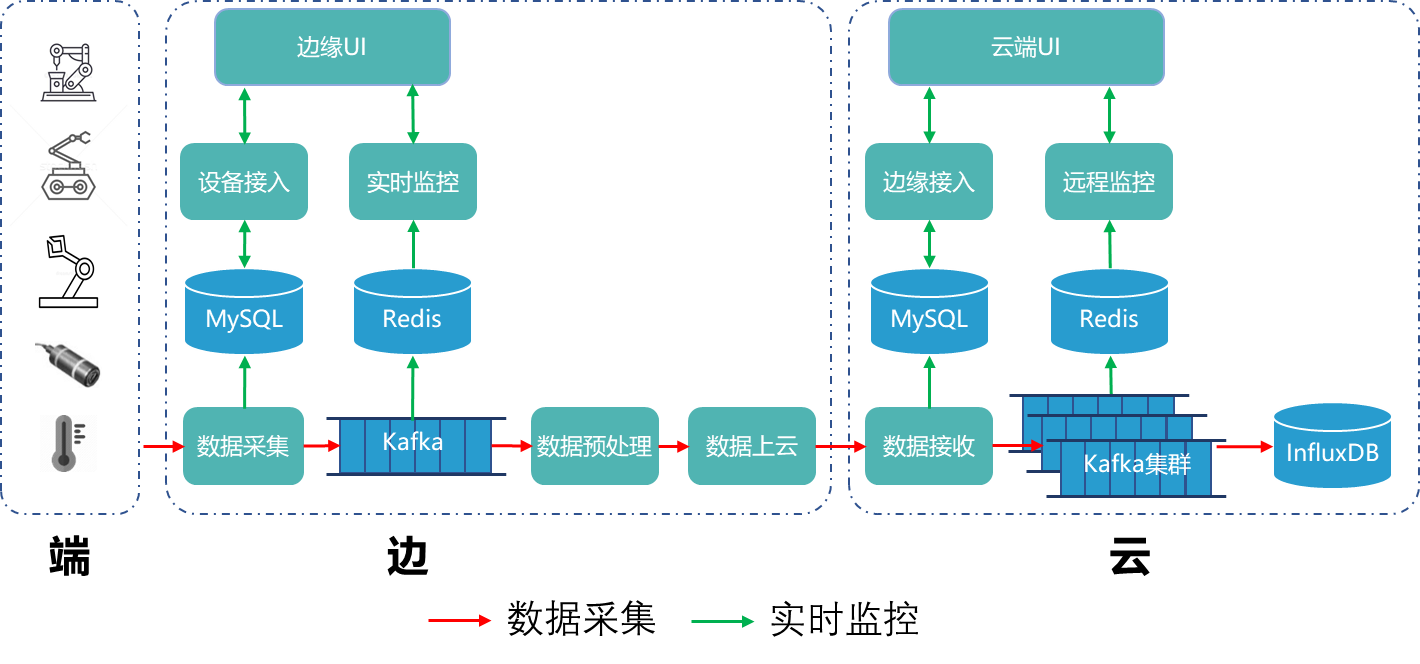


图4.1 系统技术架构

在系统的软件体系结构中，指令分析模块是最核心的模块，相当于机器人的控制中心，负责从用户处获取指令和其相关信息，然后启动指令执行模块调动机器人的响应动作结构可以用于任何应用ROS的机器人系统，拥有较高的系统可移植性。

## 数据采集与数据管理架构



## 集群管理架构

## 模块间的关系

用图4.2表达了在体系结构之中各模块之间的关系。

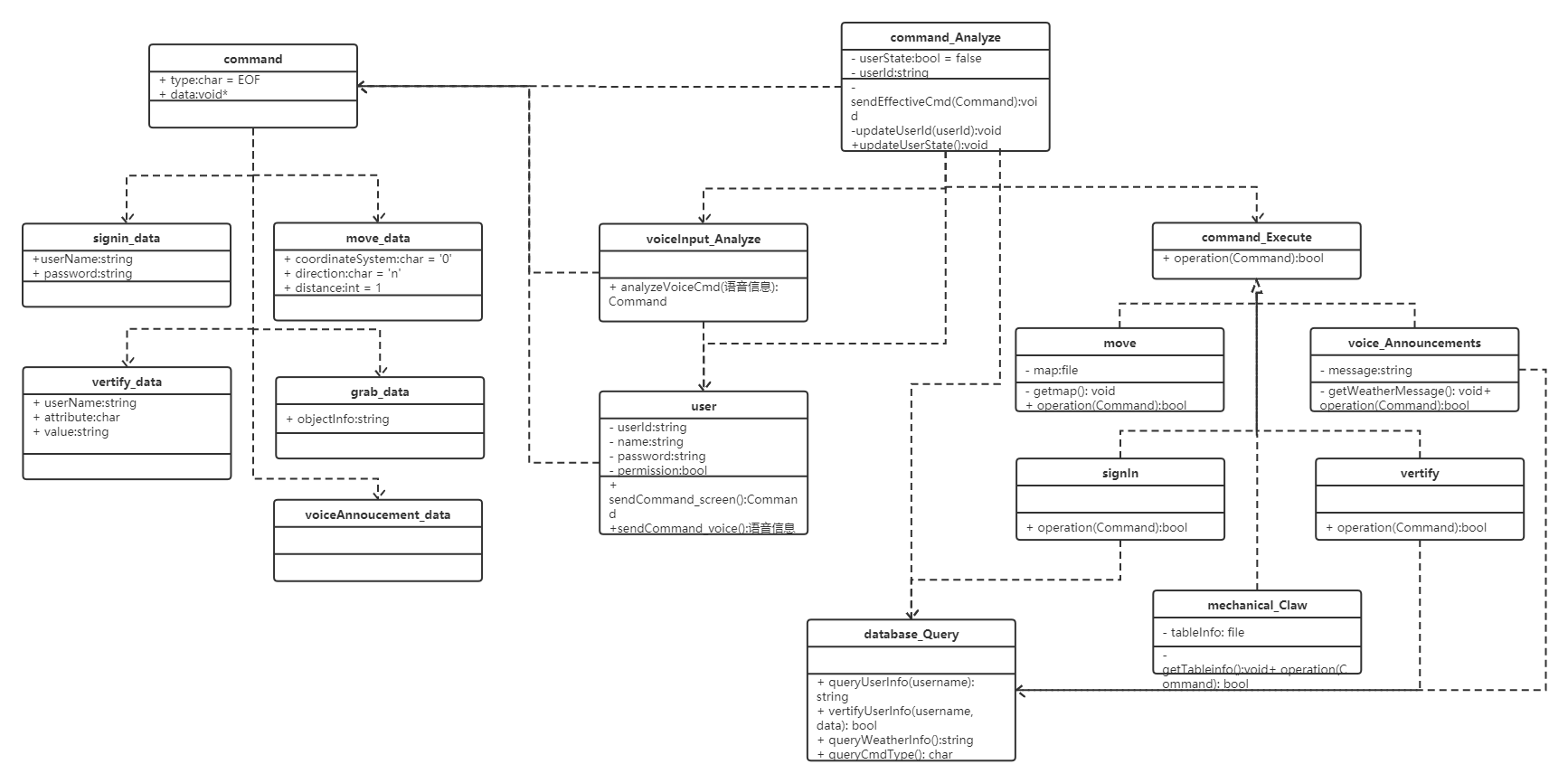


图4.2 设计单元及其之间的关系

主要描述了软件体系结构图中除了接口以外的其他模块的设计。模块之间的关系主要为依赖，通过调用对方的公用函数形成。其中command为一个指令类，描述了在机器人系统中的传递的指令。为了方便使用，其所有属性均为公有。

# 接口设计

## 支持多协议数据采集接口

数据传输由边缘端主动发送过来，客户端采用订阅方式接收数据，因此客户端不需要管边缘端怎么发送和具体使用什么协议进行传输，只关心客户端自己支持的传输协议和安全机制，因此暴露接口给边缘端，与边缘端服务器建立连接。

**1、OPC UA协议数据采集接口：**此部分是采用OPC UA协议传输数据的接口，创建OPC UA客户端，与OPC UA服务器连接，进行数据订阅。

**2、[集成多协议的数据采集接口](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F%E6%8E%A5%E5%8F%A3" \t "https://baike.baidu.com/item/%E7%94%A8%E6%88%B7%E6%8E%A5%E5%8F%A3/_blank)：**此部分在前面的基础上，加入了已实现的webSocket协议和MQTT协议的客户端，三个协议集成为一个接口，即抽象工厂模式。以便于边缘端服务器根据自己的硬件支持和需求在调用API时，通过携带相关的参数作为具体协议和安全机制的使用需求，进行数据传输。

## 数据库存取接口

数据库存取模块将存取操作封装成接口，便于数据库的管理，其中调用者分别为：前端request请求查询和下载、保存到本地，数据采集模块中数据持久化存储。

数据库的开发采用Django框架，model模型对应建立数据库表，view视图模块对应前端请求以及数据给前端展示，这些操作涉及到的数据库增删查改等函数都封装在该接口里。

1. **数据库表-model**

class MachineInfo(models.Model)

1. **数据存储接口**
2. **前端访问数据接口**

def get\_data\_from\_db\_send\_to\_user(request):  
 """  
 API: 给前端调用获取数据的接口  
 @param request: 前端请求  
 @return:  
 """

基于webSocket与前端建立传输通道，

## 数据上传到云服务器接口

编写数据上传到杭州研究院私有云服务器的函数，实现从本地Kafka消息队列拉取数据然后上传，保存到云端Kafka集群消息队列。

# 详细设计

## OPC UA客户端模块

该模块主要任务是接收设备边发送过来的数据，然后将数据放入本地单机kafka中，同时实现加密的安全传输。

创建OPC UA客户端，建立安全通道，创建会话，用户验证，认证证书和密钥，连接到OPC UA服务器端口，开始订阅消息。

安全策略：支持“无”、“Basic256Sha256”和“Basic128Rsa15”（OPC基金会不再将它视为安全）三种。  
 安全模式：当选择后两种安全策略时，安全模式需要选择“Sign”和“SignAndEncrypt”之一，同时需要上传数字证书和密钥证书。

相关配置：

* 用户名/密码：opcua/123456，可以对“匿名”标识符进行权限限制
* 安全策略：Basic256Sha256
* 安全模式：SignAndEncrypt
* 证书/私钥：client\_cert.pem / client\_key.pem

使用OpenSSL生成CA证书，再使用CA证书生成服务器和客户端证书、私钥，客户端检验服务器证书，判定私钥是否正确：

client.set\_security\_string("Basic256Sha256,SignAndEncrypt," + cer\_path + "," + private\_key\_path + server\_cer\_path)

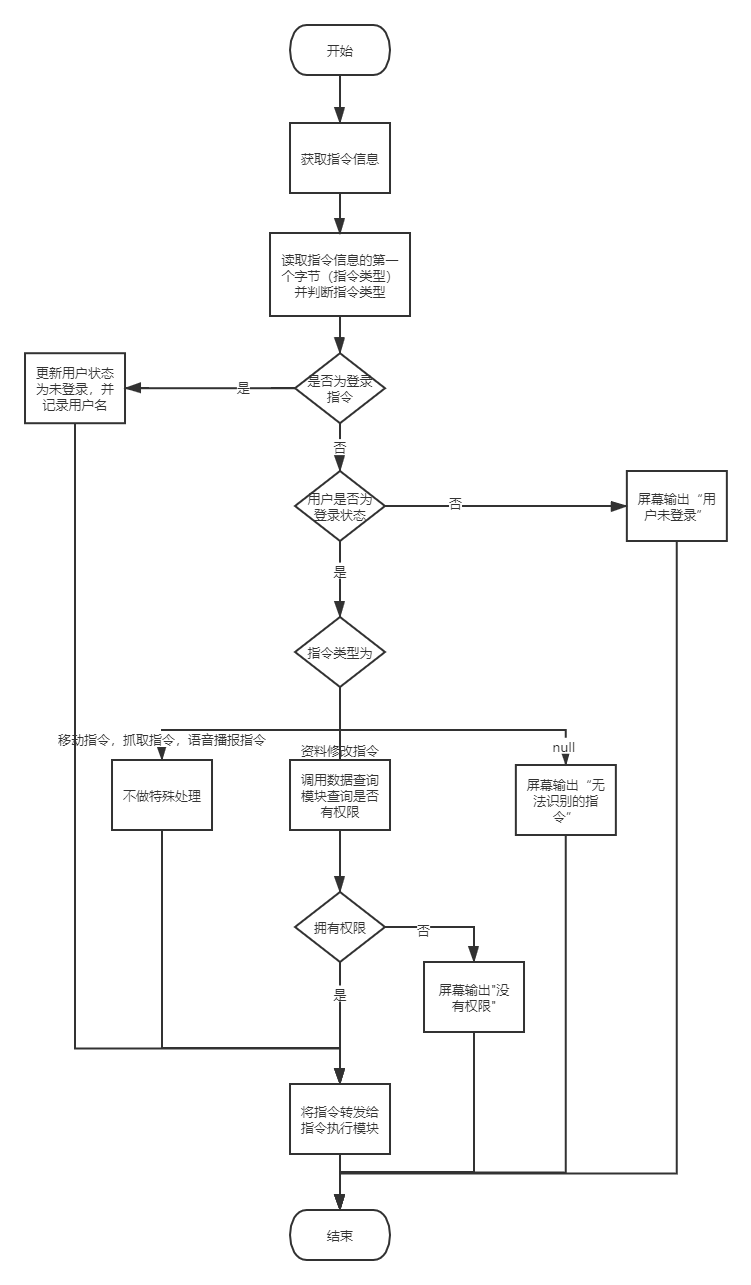


图6.1 指令分析模块流程图

## 支持多协议数据采集模块

实际上就是开启三个客户端，分别是MQTT、webSocket、OPC UA客户端，然后边缘端选择想要的协议的端口号进行连接和数据传输。

边缘端服务器主动向客户端推送数据，webSocket利用socket能够实现长连接，并且能够全双工通信，而不像HTTP协议那样需要像服务器发送请求才能获取数据；MQTT协议运行在[TCP/IP](http://itbilu.com/nodejs/core/VkcdcFq9.html" \l "OSI" \t "https://www.cnblogs.com/jiangzhaowei/p/_blank)网络协议上，使用的发布/订阅消息模式，提供有序、无损、双向连接；OPC UA协议给予TCP协议之上，增加了内部安全机制，使得数据传输安全性提高。

webSocket和MQTT中采用json格式的数据传输，OPC UA采用xml格式的数据传输

创建kafka消息队列对象，将收到的数据根据key-value解析后送入kafka队列，

数据预处理，将Json格式进行处理，Python自带函数dict与Json的转化

编写数据持久化操作函数，将消息队列里的数据存储在本地MySQL数据库中。

MySQL查询

Redis访问

## 数据库相关模块

其中models.py文件是Django框架中的模型，主要功能是创建数据库表和数据信息，首先需要让系统和连接到本地MySQL的loacl\_db，然后创建机器信息、图片信息表，以便实现数据存取的操作。

实现热点数据缓存到Redis内存数据库中，提供给前端访问数据的接口，以便前端获取数据进行实时监控。

views.py文件是Django框架中的视图，主要功能是将数据给用户，同时接收用户的请求，涉及到数据库的增删查改操作。

采用批量数据添加，对数据建立list

对于Redis缓存，建立连接池，避免重复连接，以时间为key，sort set为数据结构存储数据，对于数据设置1小时作为过期时间，过期后就失效，以免缓存容量满。

图片保存路径和数据到数据库，（考虑是否保存在本地文件夹），在settings.py文件里，以path+’/’+time+tag+’.jpg’为文件名进行存储，tag为IMG\_TYPE = {  
 1: 'L1', 2: 'L2', 3: 'action', 4: 'reg', 5: 'result'  
}

图片压缩640\*480大小

**需要调用的函数：**

## 数据传输给前端模块

这部分将所有的数据通过webSocket通信传输给前端，需要部署webSoket，

## 数据上云模块

这部分将所有的数据上传到杭州研究院云服务器，并存储到相应的云服务器数据库，以便其余工作人员拉取数据。

安装websocket-client

建立webSocket连接

## 集群管理模块

本地Kafka集群，本地Redis集群，云端Kafka也采用集群，由于云服务器数据库支持多引擎，分布式集群则需要进行统一管理，保证数据一致性，以便远程访问和监控、离线训练。

## 索引优化设计模块

针对于本地MySQL的local\_db进行索引优化，从而实现更高效的数据访问。

git版本管理

git init

git add .

git commit -m “some info”

git pull origin 远程分支名

git push origin master

启动流水线测试：

实际边缘端和后端交互的接口如下：根据http的request请求传输

192.168.13.201:8000/workshop/data/set2，本机ipv4地址需要修改为192.168.13.201

默认网关192.168.13.254

首选DNS 114.114.114.114

1. 启动所有机器
2. 调整机器连接状态，使其成功连接设备
3. 复位机器状态，点击启动按钮

4.点击1号托盘上料，开始运行